Object Oriented Programming Project - Horse Race Simulation

Part 1:

1. Write the Horse class:

In writing up the horse class, encapsulation is very important for safeguarding the data tied to the horse object. It prevents the user from directly interacting and changing the data through means other than the accessor and mutator methods. Directly interacting with the data can be an issue where some field variables are supposed to have specific requirements. For example, in the case of the horse class, the horse's distance is only intended to be increased by 1 at a time during a race or set to 0 if the race is restarted.

The methods provided as accessor methods are:

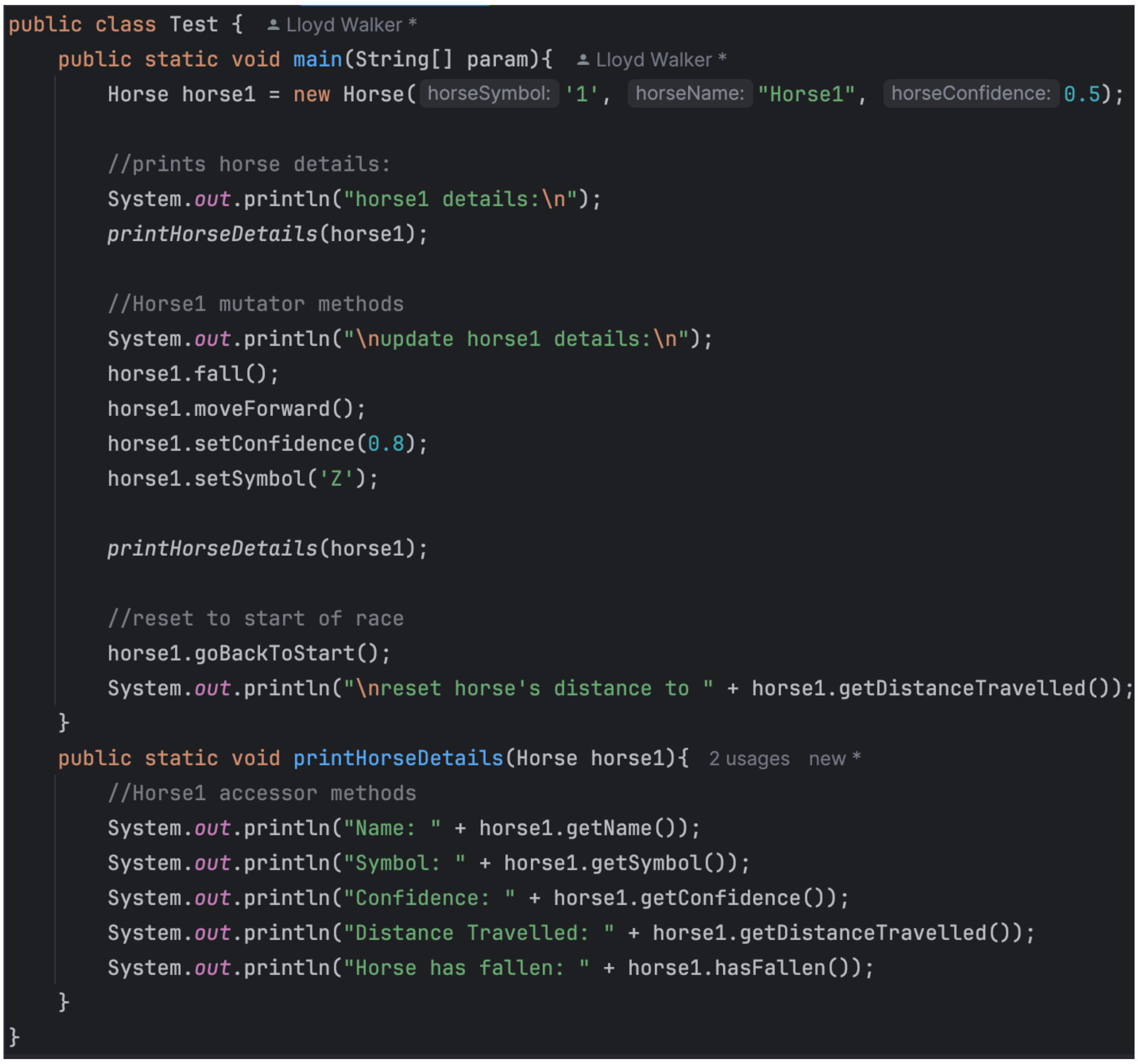
- getConfidence()
- getDistanceTravelled()
- getName()
- getSymbol()
- hasFallen()

The methods provided as mutator methods are:

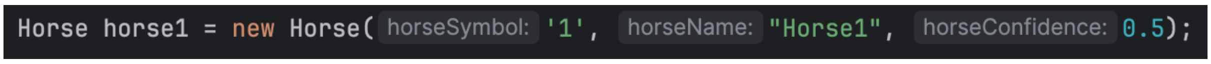
- fall()
- goBackToStart()
- moveForward()
- setConfidence(double newConfidence)
- setSymbol(char newSymbol)

Testing Process:

The following code is the test written for the Horse class:



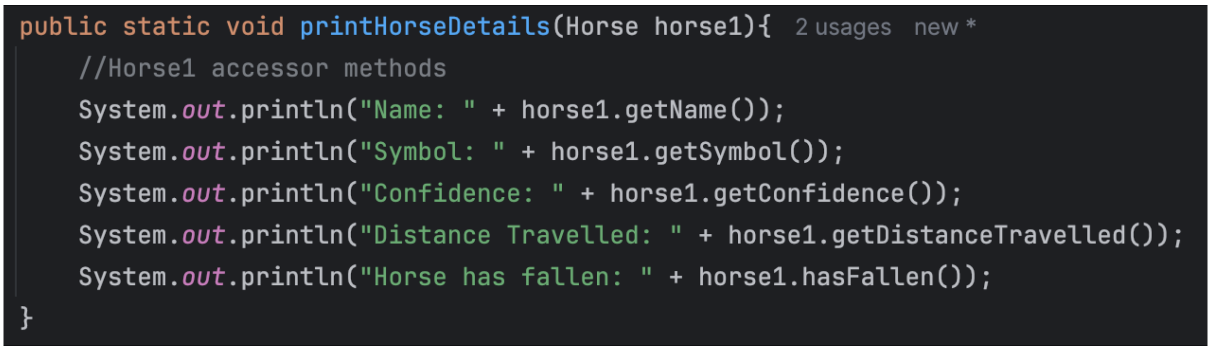
```
public class Test {  Lloyd Walker *
    public static void main(String[] param){  Lloyd Walker *
        Horse horse1 = new Horse( horseSymbol: '1', horseName: "Horse1", horseConfidence: 0.5);

        //prints horse details:
        System.out.println("horse1 details:\n");
        printHorseDetails(horse1);

        //Horse1 mutator methods
        System.out.println("\nupdate horse1 details:\n");
        horse1.fall();
        horse1.moveForward();
        horse1.setConfidence(0.8);
        horse1.setSymbol('Z');

        printHorseDetails(horse1);

        //reset to start of race
        horse1.goBackToStart();
        System.out.println("\nreset horse's distance to " + horse1.getDistanceTravelled());
    }
    public static void printHorseDetails(Horse horse1){  2 usages  new *
        //Horse1 accessor methods
        System.out.println("Name: " + horse1.getName());
        System.out.println("Symbol: " + horse1.getSymbol());
        System.out.println("Confidence: " + horse1.getConfidence());
        System.out.println("Distance Travelled: " + horse1.getDistanceTravelled());
        System.out.println("Horse has fallen: " + horse1.hasFallen());
    }
}
```



```
Horse horse1 = new Horse( horseSymbol: '1', horseName: "Horse1", horseConfidence: 0.5);
```

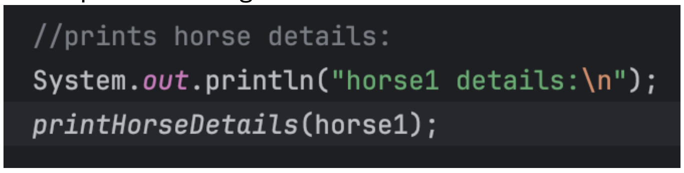
For this test I will be using the horse object ‘horse1’. This was created using the constructor seen above so the horse was also created with a symbol of ‘1’, a name of “Horse1”, and a confidence of 0.5. Note that a horse cannot be created without the above parameters as this is the only constructor available.

The first test written was for the accessor methods:



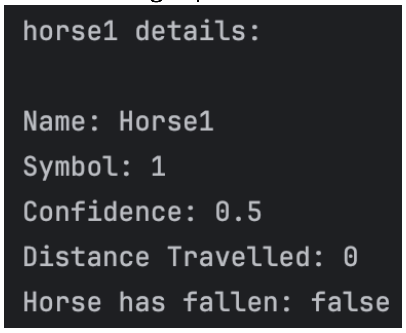
```
public static void printHorseDetails(Horse horse1){  2 usages   new *
    //Horse1 accessor methods
    System.out.println("Name: " + horse1.getName());
    System.out.println("Symbol: " + horse1.getSymbol());
    System.out.println("Confidence: " + horse1.getConfidence());
    System.out.println("Distance Travelled: " + horse1.getDistanceTravelled());
    System.out.println("Horse has fallen: " + horse1.hasFallen());
}
```

When passed through the main method as:



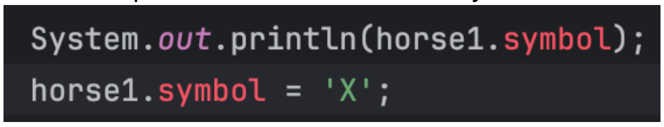
```
//prints horse details:
System.out.println("horse1 details:\n");
printHorseDetails(horse1);
```

The following is printed:



```
horse1 details:


Name: Horse1
Symbol: 1
Confidence: 0.5
Distance Travelled: 0
Horse has fallen: false
```

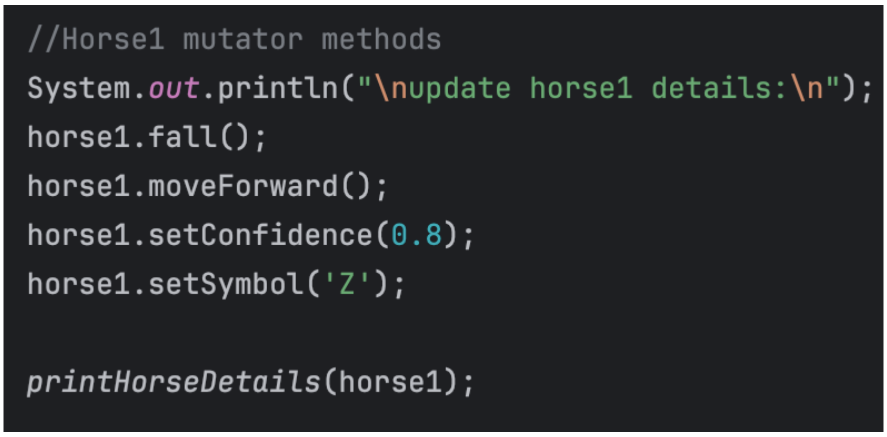
If an attempt is made to interact directly with the field variables like:



```
System.out.println(horse1.symbol);
horse1.symbol = 'X';
```

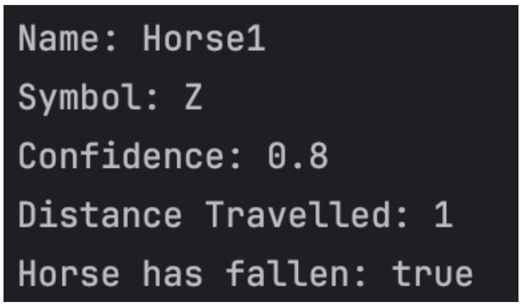
An error is thrown as the field variables are private, this ensures only the accessor/mutator methods can directly interact with the field variables.

Next the data is updated with some of the mutator methods and printed using the printHorseDetails method:



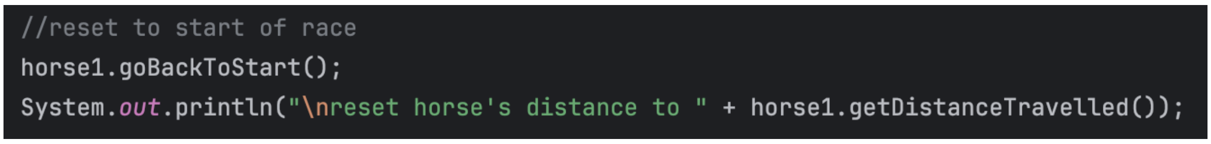
```
//Horse1 mutator methods
System.out.println("\nupdate horse1 details:\n");
horse1.fall();
horse1.moveForward();
horse1.setConfidence(0.8);
horse1.setSymbol('Z');

printHorseDetails(horse1);
```

These changes can be seen here when the details are printed:

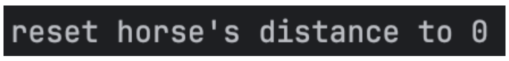


```
Name: Horse1
Symbol: Z
Confidence: 0.8
Distance Travelled: 1
Horse has fallen: true
```

Finally, the last mutator method is used and the result is printed:



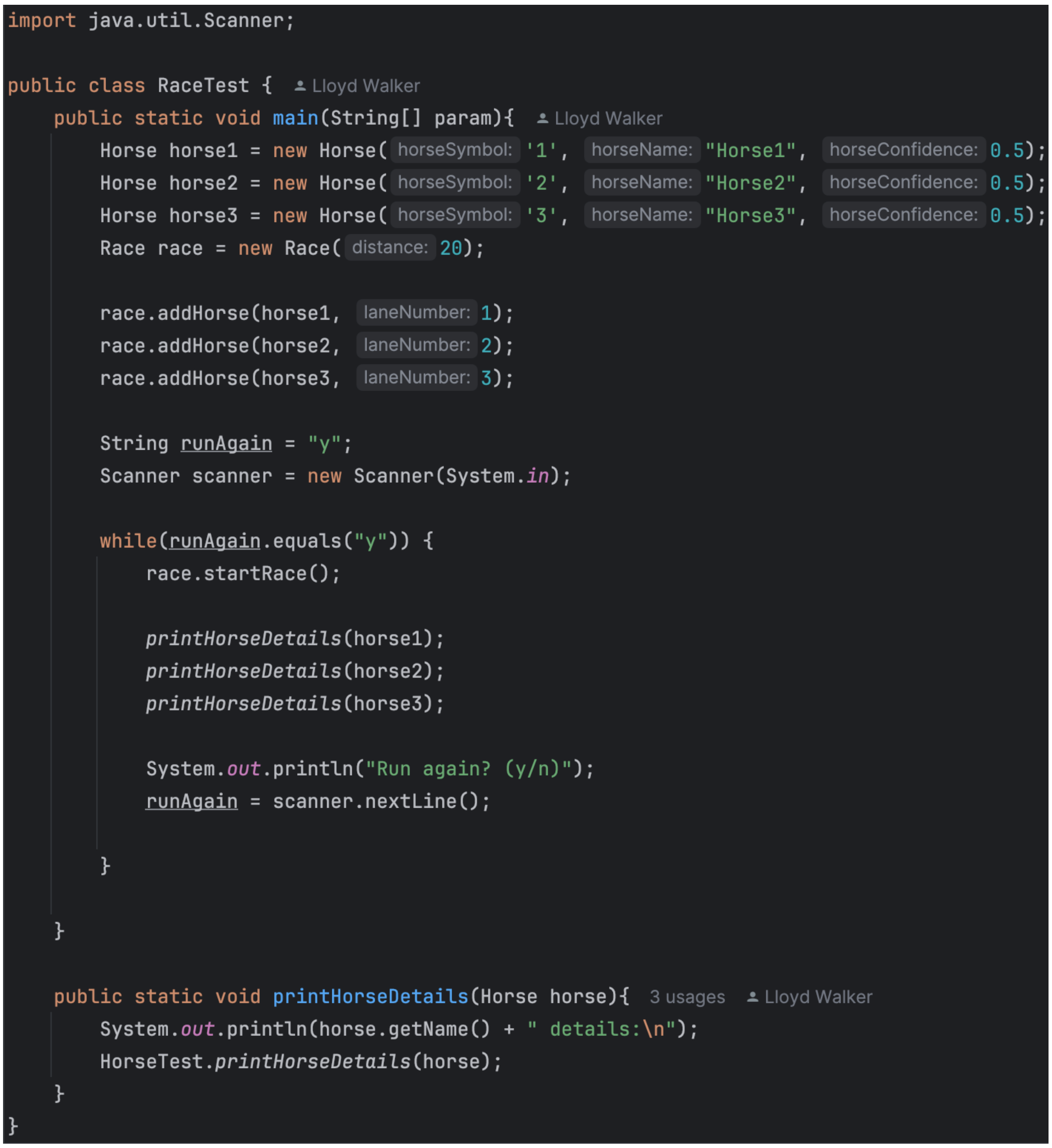
```
//reset to start of race
horse1.goBackToStart();
System.out.println("\nreset horse's distance to " + horse1.getDistanceTravelled());
```



```
reset horse's distance to 0
```

2. Examine and improve the Race Class:

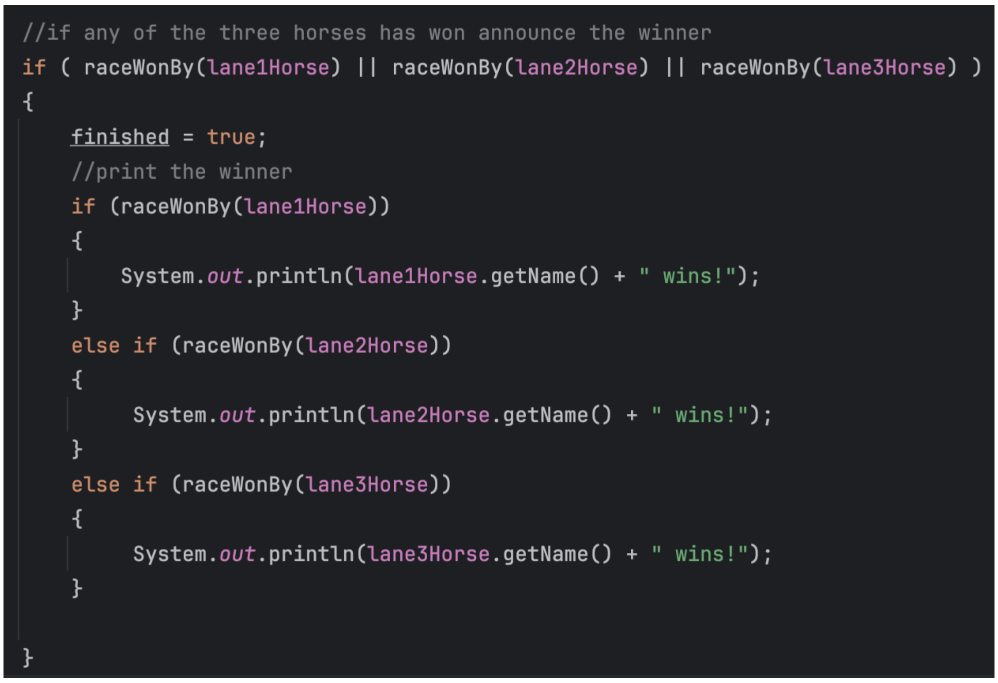
For testing the race class I implemented a new class RaceTest. The main method in this class worked by initiating 3 horses as well as new race. The 3 horses were then added to the race object using addHorse(). Parameters for a while loop are also set up so the race can be run multiple times with the same horses. In the loop startRace() is first called to start the race in the terminal. After the race concludes the details of all three horses are printed via the method printHorseDetails(). This method calls upon a predefined method ‘printHorseDetails() of the HorseTest class. Finally the user is asked if they want to run the race again. This method can be run to test whenever something was changed in the Race class:



```java
import java.util.Scanner;

public class RaceTest {  Lloyd Walker
    public static void main(String[] param){  Lloyd Walker
        Horse horse1 = new Horse( horseSymbol: '1',  horseName: "Horse1",  horseConfidence: 0.5);
        Horse horse2 = new Horse( horseSymbol: '2',  horseName: "Horse2",  horseConfidence: 0.5);
        Horse horse3 = new Horse( horseSymbol: '3',  horseName: "Horse3",  horseConfidence: 0.5);
        Race race = new Race( distance: 20);

        race.addHorse(horse1,  laneNumber: 1);
        race.addHorse(horse2,  laneNumber: 2);
        race.addHorse(horse3,  laneNumber: 3);

        String runAgain = "y";
        Scanner scanner = new Scanner(System.in);

        while(runAgain.equals("y")) {
            race.startRace();

            printHorseDetails(horse1);
            printHorseDetails(horse2);
            printHorseDetails(horse3);

            System.out.println("Run again? (y/n)");
            runAgain = scanner.nextLine();

        }

    }

    public static void printHorseDetails(Horse horse){  3 usages   Lloyd Walker
        System.out.println(horse.getName() + " details:\n");
        HorseTest.printHorseDetails(horse);
    }
}
```

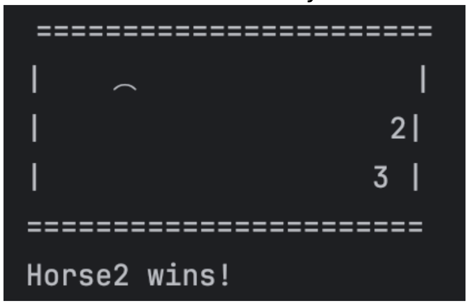
The following changes will be numbered to make it clear when a new change is being addressed.

1) The first change made to the Race class, as tasked by the project guide, was to enhance the startRace() method by displaying the winner of the race, if applicable. This was done by adding the following:



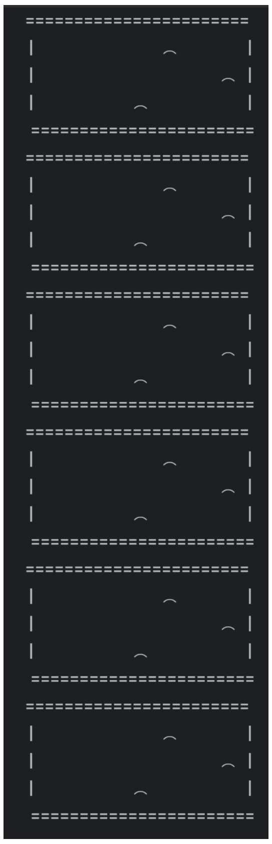
```java
//if any of the three horses has won announce the winner
if ( raceWonBy(lane1Horse) || raceWonBy(lane2Horse) || raceWonBy(lane3Horse) )
{
    finished = true;
    //print the winner
    if (raceWonBy(lane1Horse))
    {
        System.out.println(lane1Horse.getName() + " wins!");
    }
    else if (raceWonBy(lane2Horse))
    {
         System.out.println(lane2Horse.getName() + " wins!");
    }
    else if (raceWonBy(lane3Horse))
    {
         System.out.println(lane3Horse.getName() + " wins!");
    }

}
```

This way at the end of the race, if a horse wins a line is printed declaring them the winner. When the test method above is ran you can see the horse announced in the terminal:



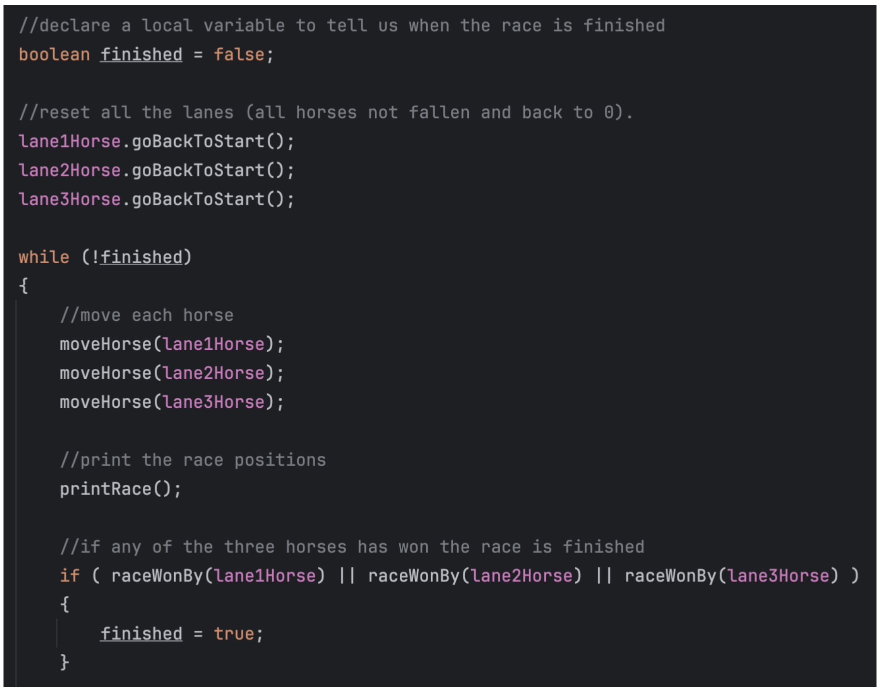
```
 ======================
|    ⌢                |
|                   2|
|                  3 |
======================
Horse2 wins!
```

2) Upon examination of the Race class the first standout issue I see is that if all the horses fall during the race the program will continue to loop infinitely. This is as the only check to terminate the while loop is if a horse has finished the race, however if all horses fall then there is no way for a horse to finish and the loop will keep looping, printing the following over and over to the terminal:



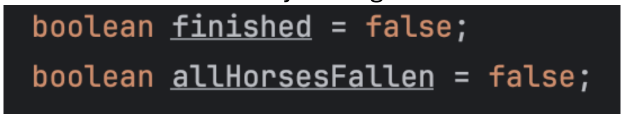
```
======================
|             ⌢      |
|                  ⌢ |
|          ⌢         |
 ======================
======================
|             ⌢      |
|                  ⌢ |
|          ⌢         |
 ======================
======================
|             ⌢      |
|                  ⌢ |
|          ⌢         |
 ======================
======================
|             ⌢      |
|                  ⌢ |
|          ⌢         |
 ======================
======================
|             ⌢      |
|                  ⌢ |
|          ⌢         |
 ======================
======================
|             ⌢      |
|                  ⌢ |
|          ⌢         |
 ======================
```

Here the problem in the code can be seen:



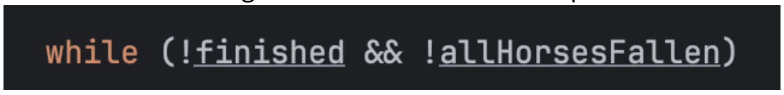
```java
//declare a local variable to tell us when the race is finished
boolean finished = false;

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

while (!finished)
{
    //move each horse
    moveHorse(lane1Horse);
    moveHorse(lane2Horse);
    moveHorse(lane3Horse);

    //print the race positions
    printRace();

    //if any of the three horses has won the race is finished
    if ( raceWonBy(lane1Horse) || raceWonBy(lane2Horse) || raceWonBy(lane3Horse) )
    {
        finished = true;
    }
```

My solution for this was to add another test to the while loop to also check if all the horses have fallen. This was done by adding an additional Boolean:



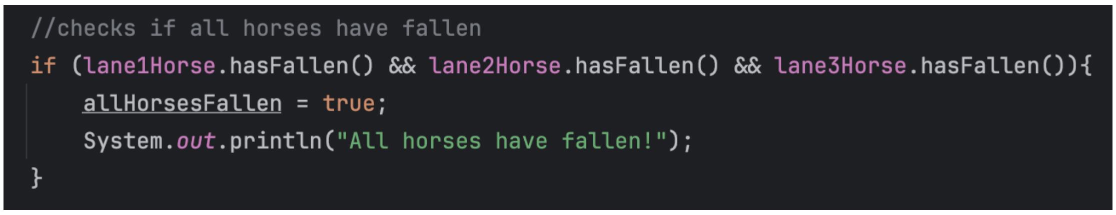
```
boolean finished = false;
boolean allHorsesFallen = false;
```

And then referencing both tests at the while loop initiation:



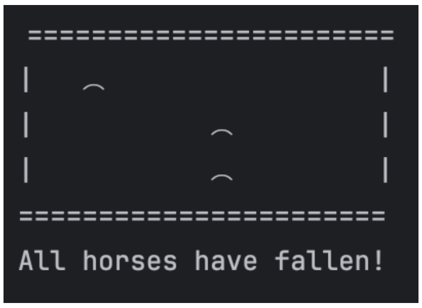
```
while (!finished && !allHorsesFallen)
```

This way if either test fails the loop will terminate. I also added an additional test within the while loop so in the case of all horses failing a message is printed:



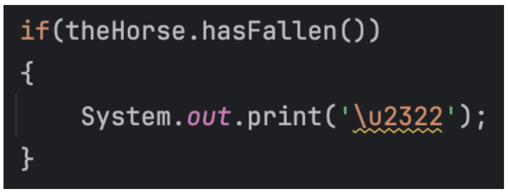
```
//checks if all horses have fallen
if (lane1Horse.hasFallen() && lane2Horse.hasFallen() && lane3Horse.hasFallen()){
    allHorsesFallen = true;
    System.out.println("All horses have fallen!");
}
```

This can be seen printed to the terminal:



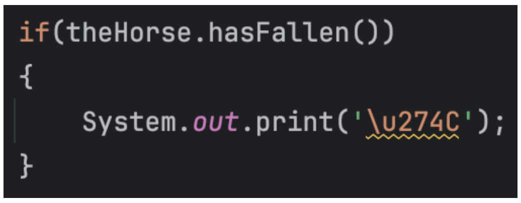
```
=======================
|  ⌢               |
|          ⌢       |
|          ⌢       |
=======================
All horses have fallen!
```

3) A minor change I also decided to implement was changing the fallen symbol assigned within the printLane() method from a ⌢ to ❌. The reason for this was to make it clearer when a horse had fallen, this was done by changing:



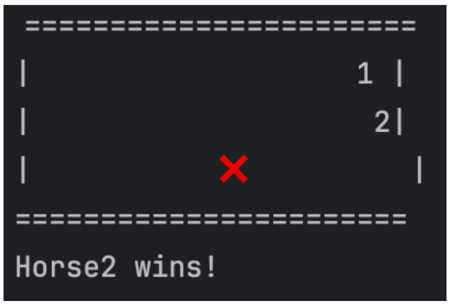
```
if(theHorse.hasFallen())
{
    System.out.print('\u2322');
}
```

To:



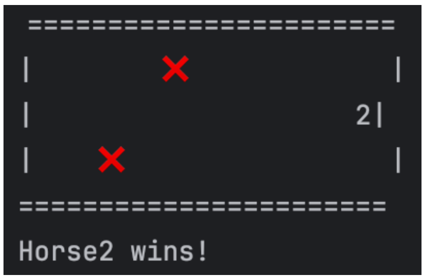
```
if(theHorse.hasFallen())
{
    System.out.print('\u274C');
}
```

So now in the terminal the following is instead printed:



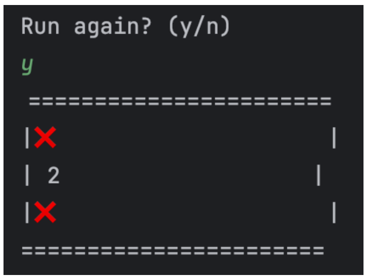
```
=======================
|                  1 |
|                   2|
|           ❌          |
=======================
Horse2 wins!
```

4) Another issue is if multiple races are run with the same horse and the horse has already fallen this will not be reset at the start of the next race and so the horse will stay fallen. This can be seen where one race finishes like:



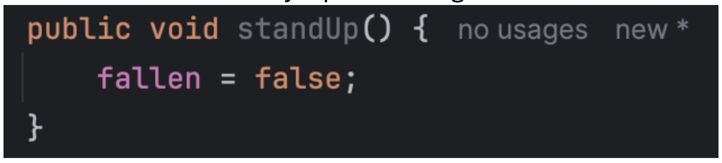
```
=======================
|       ❌             |
|                  2|
|   ❌                 |
=======================
Horse2 wins!
```

The next will start like:



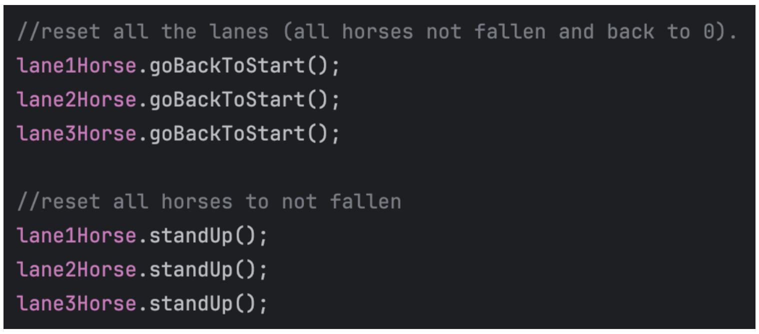
```
Run again? (y/n)
y
=======================
|❌                    |
| 2                  |
|❌                    |
=======================
```

To amend this, I first had to add a method to reverse the fallen status of the horse to the Horse class as there was not already a pre-existing method. The method I added was:



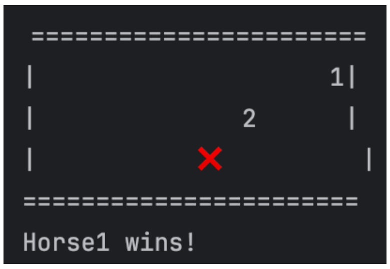
```java
public void standUp() {  no usages   new *
    fallen = false;
}
```

This was then added to the startRace() method:



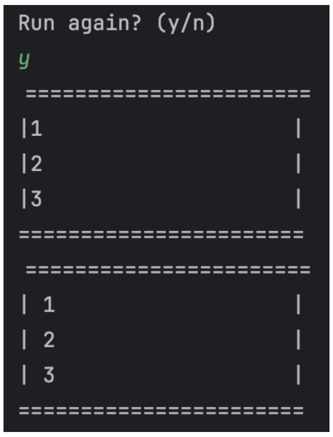
```java
//reset all the lanes (all horses not fallen and back to 0).
lane1Horse.goBackToStart();
lane2Horse.goBackToStart();
lane3Horse.goBackToStart();

//reset all horses to not fallen
lane1Horse.standUp();
lane2Horse.standUp();
lane3Horse.standUp();
```

This way if some horses in the race fall and another race is run after all horses will be reset to fallen = false. An example of this change working can be seen where if a race ends like:



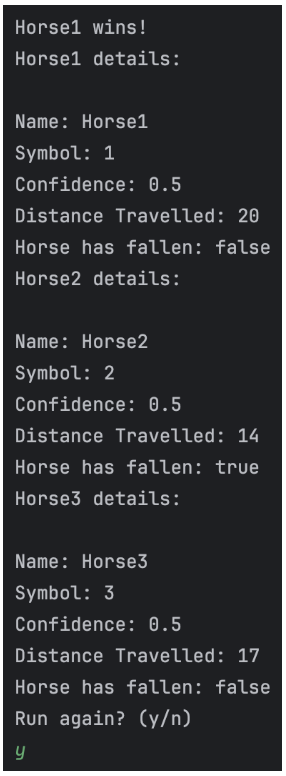
```
 =======================
|                    1|
|              2      |
|           ❌         |
=======================
Horse1 wins!
```

The following race will look like:



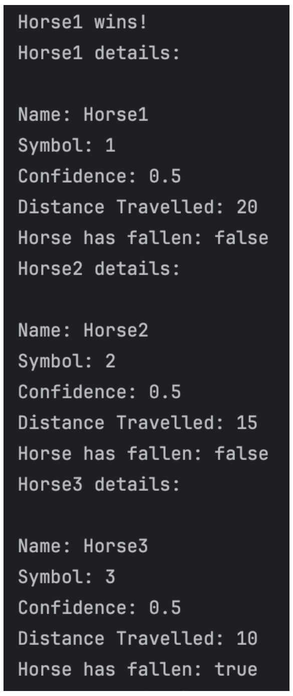
```
Run again? (y/n)
y
 ======================
|1                    |
|2                    |
|3                    |
======================
 ======================
| 1                   |
| 2                   |
| 3                   |
======================
```

5) Another issue found is according to the project guide “Winning a race slightly increases a horse's confidence, while falling during a race slightly decreases it.” However, there is no implementation of this in the Race class. For example, if 2 races are run sequentially the first results printed are:



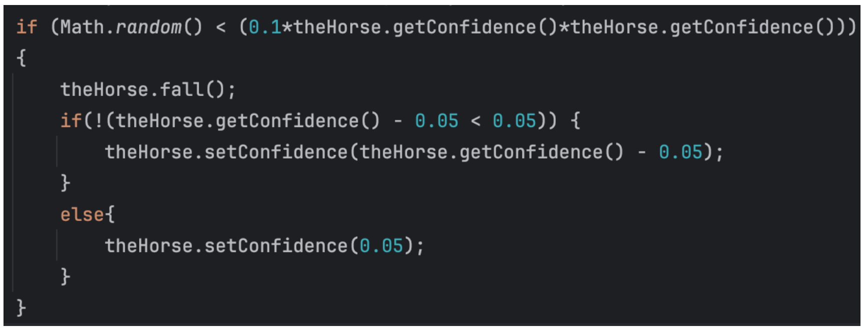
```
Horse1 wins!
Horse1 details:

Name: Horse1
Symbol: 1
Confidence: 0.5
Distance Travelled: 20
Horse has fallen: false
Horse2 details:

Name: Horse2
Symbol: 2
Confidence: 0.5
Distance Travelled: 14
Horse has fallen: true
Horse3 details:

Name: Horse3
Symbol: 3
Confidence: 0.5
Distance Travelled: 17
Horse has fallen: false
Run again? (y/n)
y
```

In the second race the results are:



```
Horse1 wins!
Horse1 details:

Name: Horse1
Symbol: 1
Confidence: 0.5
Distance Travelled: 20
Horse has fallen: false
Horse2 details:

Name: Horse2
Symbol: 2
Confidence: 0.5
Distance Travelled: 15
Horse has fallen: false
Horse3 details:

Name: Horse3
Symbol: 3
Confidence: 0.5
Distance Travelled: 10
Horse has fallen: true
```

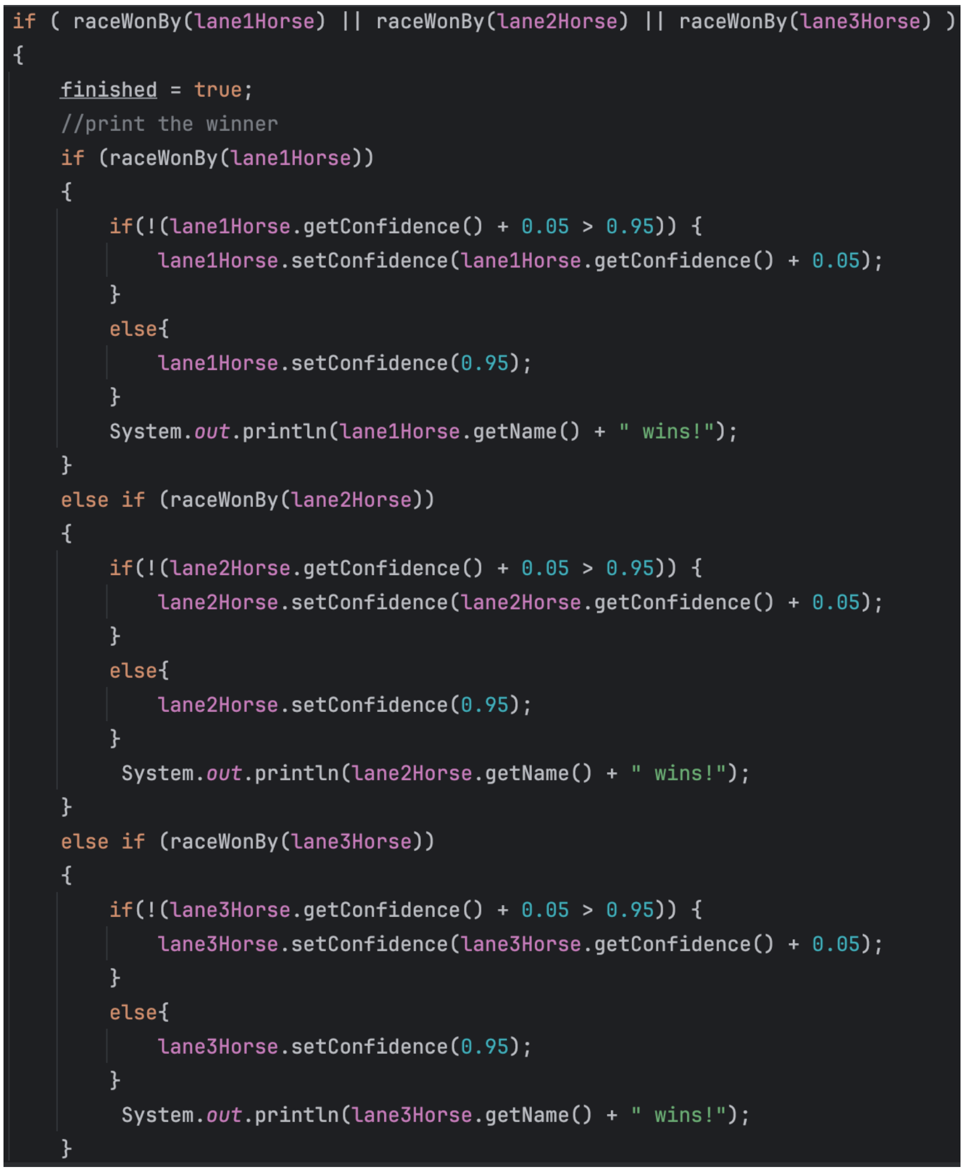
Despite horse 2 falling in the first race and horse 3 falling in the second their confidence remained at 0.5, the same applying for horse 1 who won both races without their confidence being affected. To amend this, I first made it so if a horse has fallen 0.05 will be taken away from their confidence.



```java
if (Math.random() < (0.1*theHorse.getConfidence()*theHorse.getConfidence()))
{
    theHorse.fall();
    if(!(theHorse.getConfidence() - 0.05 < 0.05)) {
        theHorse.setConfidence(theHorse.getConfidence() - 0.05);
    }
    else{
        theHorse.setConfidence(0.05);
    }
}
```

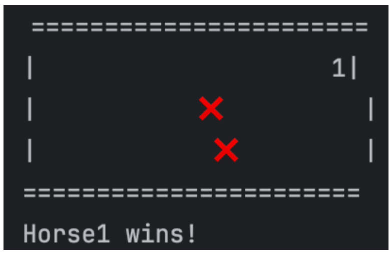
This implementation will also check that the calculation will not result in a number outside of the 0-1 confidence range.

For the case of a horse winning the following was implemented:



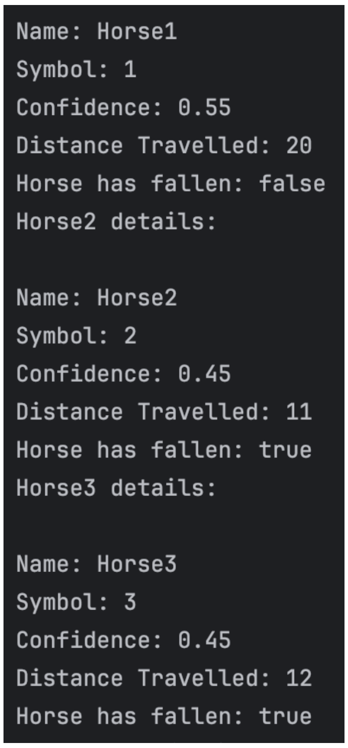
```java
if ( raceWonBy(lane1Horse) || raceWonBy(lane2Horse) || raceWonBy(lane3Horse) )
{
    finished = true;
    //print the winner
    if (raceWonBy(lane1Horse))
    {
        if(!(lane1Horse.getConfidence() + 0.05 > 0.95)) {
            lane1Horse.setConfidence(lane1Horse.getConfidence() + 0.05);
        }
        else{
            lane1Horse.setConfidence(0.95);
        }
        System.out.println(lane1Horse.getName() + " wins!");
    }
    else if (raceWonBy(lane2Horse))
    {
        if(!(lane2Horse.getConfidence() + 0.05 > 0.95)) {
            lane2Horse.setConfidence(lane2Horse.getConfidence() + 0.05);
        }
        else{
            lane2Horse.setConfidence(0.95);
        }
         System.out.println(lane2Horse.getName() + " wins!");
    }
    else if (raceWonBy(lane3Horse))
    {
        if(!(lane3Horse.getConfidence() + 0.05 > 0.95)) {
            lane3Horse.setConfidence(lane3Horse.getConfidence() + 0.05);
        }
        else{
            lane3Horse.setConfidence(0.95);
        }
         System.out.println(lane3Horse.getName() + " wins!");
    }
```

Now if a race is run here’s how the results are affected:



```
=======================
|                    1|
|           ✕          |
|            ✕         |
=======================
Horse1 wins!
```

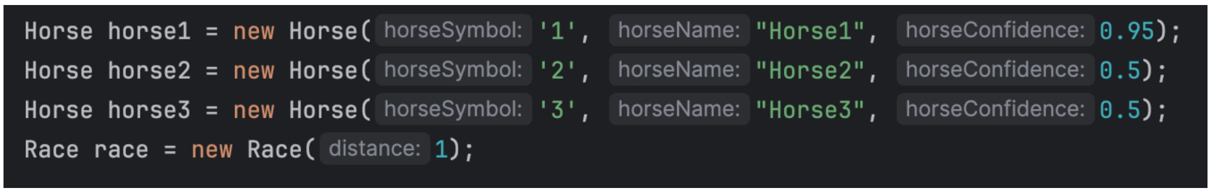
In the case of horse 1 winning the race and horse 2 and 3 both falling here is how the stats are affected:



```
Name: Horse1
Symbol: 1
Confidence: 0.55
Distance Travelled: 20
Horse has fallen: false
Horse2 details:

Name: Horse2
Symbol: 2
Confidence: 0.45
Distance Travelled: 11
Horse has fallen: true
Horse3 details:

Name: Horse3
Symbol: 3
Confidence: 0.45
Distance Travelled: 12
Horse has fallen: true
```

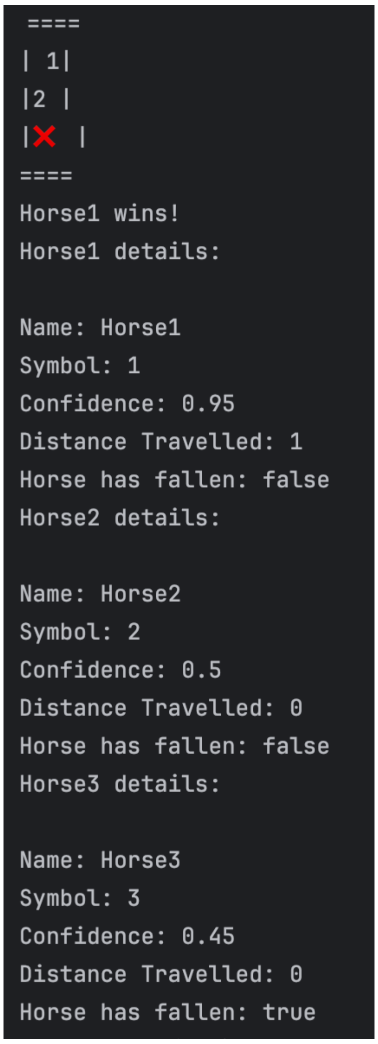
Note that this was the first instance of a race, with all horses starting with a default 0.5 confidence rating.

To test that the maximum confidence was working I assigned horse1 a confidence of 0.95 and limited the race to a distance of 1. This can be seen being implemented in the Race test here:



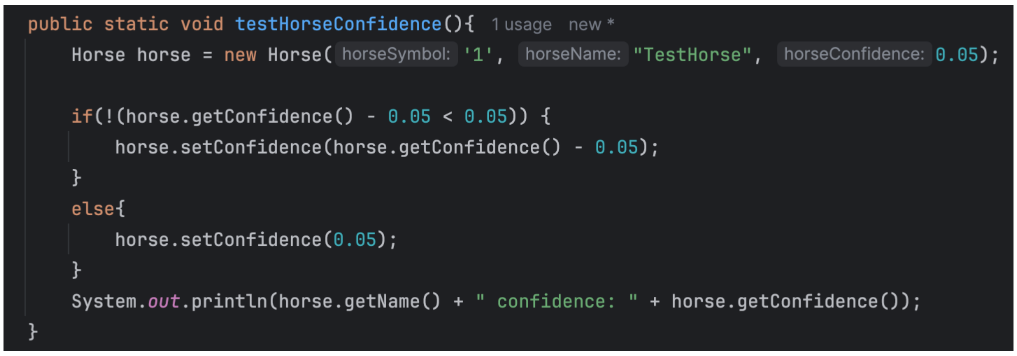
```
Horse horse1 = new Horse( horseSymbol: '1', horseName: "Horse1", horseConfidence: 0.95);
Horse horse2 = new Horse( horseSymbol: '2', horseName: "Horse2", horseConfidence: 0.5);
Horse horse3 = new Horse( horseSymbol: '3', horseName: "Horse3", horseConfidence: 0.5);
Race race = new Race( distance: 1);
```

When the test is run and the winner is announced as Horse1 the following is printed:



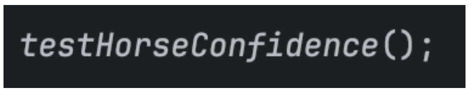
```
 ====
| 1|
|2 |
|❌  |
====
Horse1 wins!
Horse1 details:

Name: Horse1
Symbol: 1
Confidence: 0.95
Distance Travelled: 1
Horse has fallen: false
Horse2 details:

Name: Horse2
Symbol: 2
Confidence: 0.5
Distance Travelled: 0
Horse has fallen: false
Horse3 details:

Name: Horse3
Symbol: 3
Confidence: 0.45
Distance Travelled: 0
Horse has fallen: true
```

As indicated above, despite winning Horse1s confidence remained at 0.95. In the case of a horse's confidence not falling below 0.05 in the case of it falling the a new test method had to be created as the chances of a Horse with such low confidence falling naturally during a race is very low. The following method was created which uses the same calculation applied to a horse's confidence if it falls:



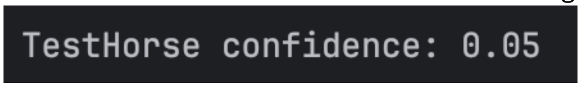
```
public static void testHorseConfidence(){  1 usage   new *
    Horse horse = new Horse( horseSymbol: '1',  horseName: "TestHorse",  horseConfidence: 0.05);

    if(!(horse.getConfidence() - 0.05 < 0.05)) {
        horse.setConfidence(horse.getConfidence() - 0.05);
    }
    else{
        horse.setConfidence(0.05);
    }
    System.out.println(horse.getName() + " confidence: " + horse.getConfidence());
}
```

This was then applied to the bottom of the main method with the line:



```
testHorseConfidence();
```

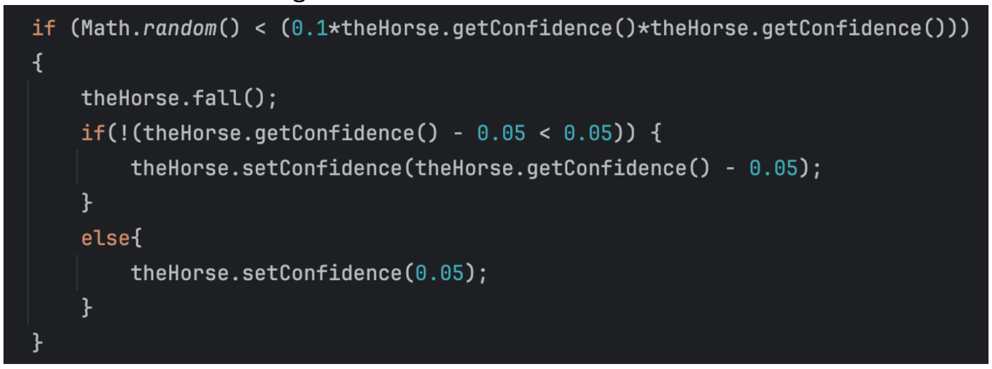
Now when the main method is run the following is printed at the bottom:



```
TestHorse confidence: 0.05
```

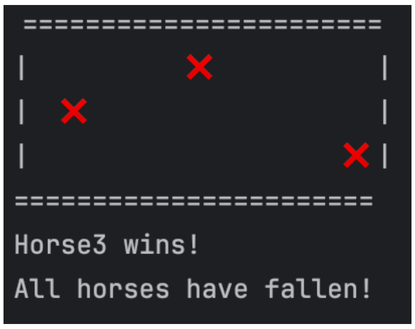
As expected, the horse's confidence did not drop below 0.05.

6) Another issue is that it is possible for a horse to win and fall at the same time. This is as in moveHorse() it will run the possibility for a horse to move and the horse to fall sequentially without checking if the horse has finished:



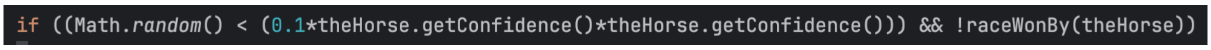
```java
if (Math.random() < (0.1*theHorse.getConfidence()*theHorse.getConfidence()))
{
    theHorse.fall();
    if(!(theHorse.getConfidence() - 0.05 < 0.05)) {
        theHorse.setConfidence(theHorse.getConfidence() - 0.05);
    }
    else{
        theHorse.setConfidence(0.05);
    }
}
```

This results in the following to print in a race:



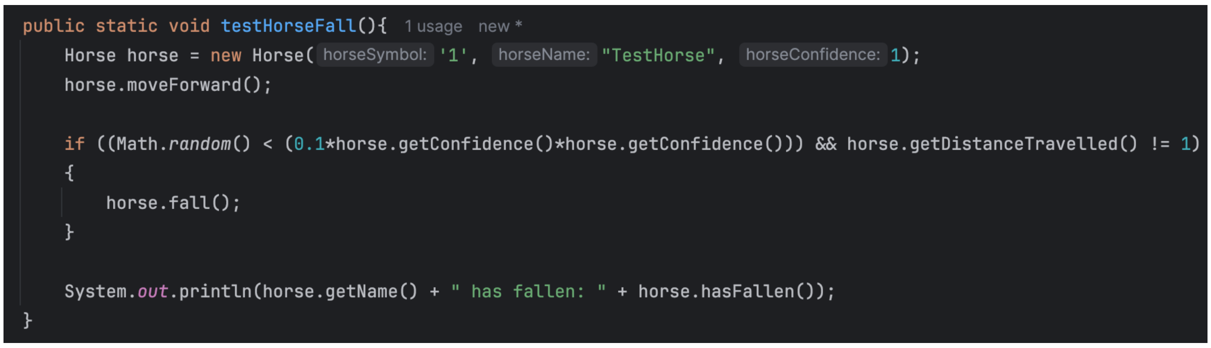
```
=======================
|         ❌         |
|  ❌                |
|                  ❌|
=======================
Horse3 wins!
All horses have fallen!
```

To amend this, I added an additional check for the test of the horse falling:



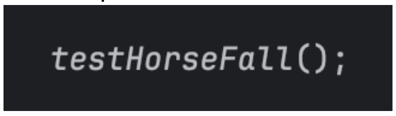
```java
if ((Math.random() < (0.1*theHorse.getConfidence()*theHorse.getConfidence())) && !raceWonBy(theHorse))
```

This way in the case of a horse finishing the race it cannot still lose. To validate this, I created the test:



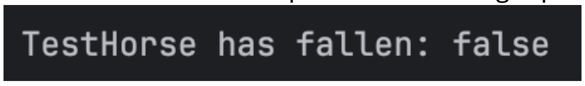
```java
public static void testHorseFall(){  1 usage  new *
    Horse horse = new Horse( horseSymbol: '1',  horseName: "TestHorse",  horseConfidence: 1);
    horse.moveForward();

    if ((Math.random() < (0.1*horse.getConfidence()*horse.getConfidence())) && horse.getDistanceTravelled() != 1)
    {
        horse.fall();
    }

    System.out.println(horse.getName() + " has fallen: " + horse.hasFallen());
}
```

Which I pushed the main method with:



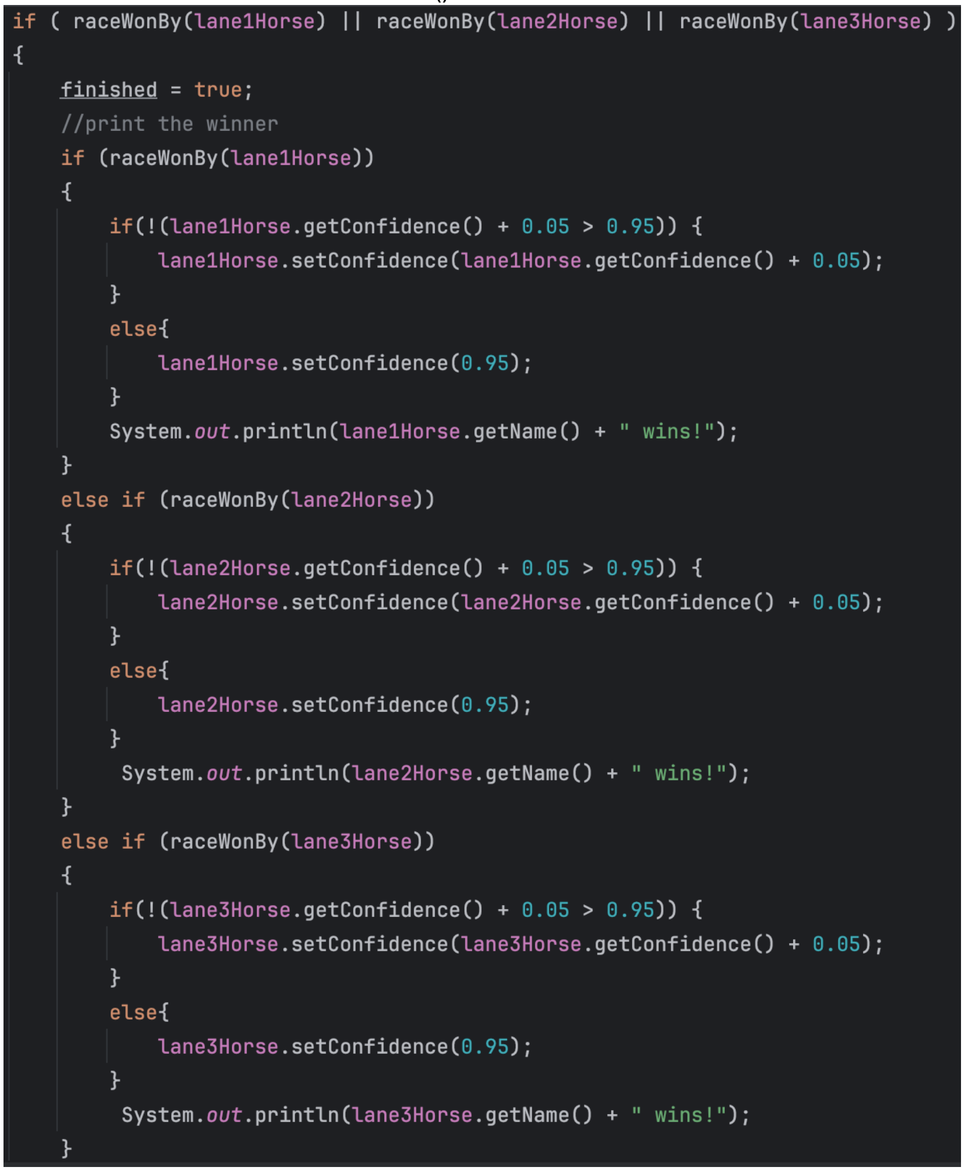
```java
testHorseFall();
```

The way the test method works is it creates a test horse which has a confidence of 1 which with the horse fall calculation, means it would be guaranteed to fall. The horse is then moved forward by 1 which we are saying is the distance of the race. The method then runs the calculation found in moveHorse() in the Race class. As the horse is at the finish line already the expected outcome is fallen: false. When compiled the following is printed as expected:



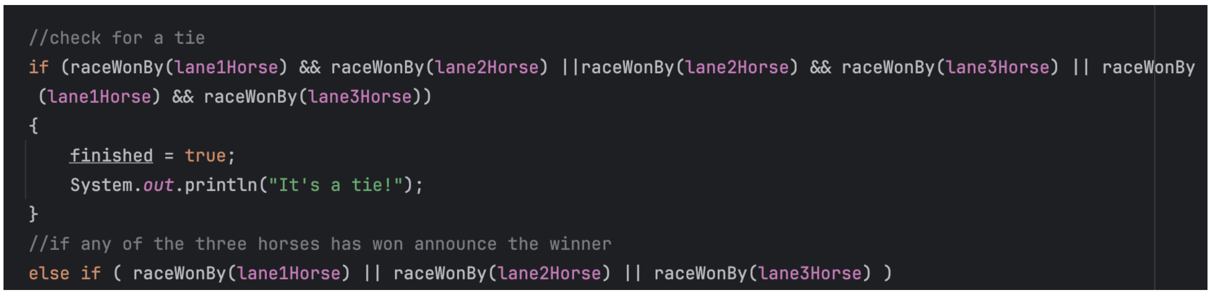
```
TestHorse has fallen: false
```

7) The final amendment I decided to make was for the instance of multiple horses finishing a race at the same time. At the moment if multiple horses finish at the same time, the horse in the front most lane is declared the winner. In the case of a tie as there is no winner there also shouldn't be any increase of confidence. This is a result of the structure of the if-else statements in the startRace() method seen here:



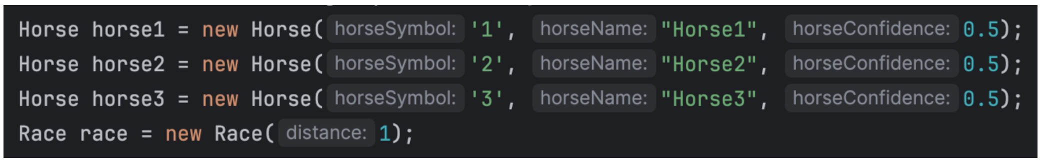
```java
if ( raceWonBy(lane1Horse) || raceWonBy(lane2Horse) || raceWonBy(lane3Horse) )
{
    finished = true;
    //print the winner
    if (raceWonBy(lane1Horse))
    {
        if(!(lane1Horse.getConfidence() + 0.05 > 0.95)) {
            lane1Horse.setConfidence(lane1Horse.getConfidence() + 0.05);
        }
        else{
            lane1Horse.setConfidence(0.95);
        }
        System.out.println(lane1Horse.getName() + " wins!");
    }
    else if (raceWonBy(lane2Horse))
    {
        if(!(lane2Horse.getConfidence() + 0.05 > 0.95)) {
            lane2Horse.setConfidence(lane2Horse.getConfidence() + 0.05);
        }
        else{
            lane2Horse.setConfidence(0.95);
        }
         System.out.println(lane2Horse.getName() + " wins!");
    }
    else if (raceWonBy(lane3Horse))
    {
        if(!(lane3Horse.getConfidence() + 0.05 > 0.95)) {
            lane3Horse.setConfidence(lane3Horse.getConfidence() + 0.05);
        }
        else{
            lane3Horse.setConfidence(0.95);
        }
         System.out.println(lane3Horse.getName() + " wins!");
    }
```

To amend this, I decided to implement a third end condition to go along with "x-horse wins!" and "All horses have fallen!". The third option being "It's a tie", showcasing the event of a draw. To add this, I added a check above the above test and made the above test into an else statement, following the tie condition. This was implemented here:



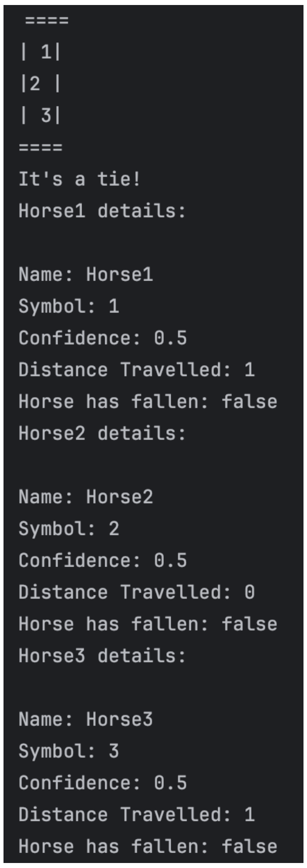
```
//check for a tie
if (raceWonBy(lane1Horse) && raceWonBy(lane2Horse) ||raceWonBy(lane2Horse) && raceWonBy(lane3Horse) || raceWonBy
 (lane1Horse) && raceWonBy(lane3Horse))
{
    finished = true;
    System.out.println("It's a tie!");
}
//if any of the three horses has won announce the winner
else if ( raceWonBy(lane1Horse) || raceWonBy(lane2Horse) || raceWonBy(lane3Horse) )
```

This way it checks between all 3 horses if a tie has occurred before it checks for a single winner. It also changes finished to true to ensure the while loop isn't repeated. To test for this, in my test method I applied to following conditions to a race:



```
Horse horse1 = new Horse( horseSymbol: '1',  horseName: "Horse1",  horseConfidence: 0.5);
Horse horse2 = new Horse( horseSymbol: '2',  horseName: "Horse2",  horseConfidence: 0.5);
Horse horse3 = new Horse( horseSymbol: '3',  horseName: "Horse3",  horseConfidence: 0.5);
Race race = new Race( distance: 1);
```

Then when the test is run and in the case of a race that a tie does occur the following is printed:



```
 ====
| 1|
|2 |
| 3|
====
It's a tie!
Horse1 details:

Name: Horse1
Symbol: 1
Confidence: 0.5
Distance Travelled: 1
Horse has fallen: false
Horse2 details:

Name: Horse2
Symbol: 2
Confidence: 0.5
Distance Travelled: 0
Horse has fallen: false
Horse3 details:

Name: Horse3
Symbol: 3
Confidence: 0.5
Distance Travelled: 1
Horse has fallen: false
```

In this example both horse 1 and 3 finished the race at the same time, resulting in a tie. As this was a tie no confidence is awarded for reaching the finish line. This is seen where Horse1 and Horse3's confidence remained at 0.5.