28/03/24

Horse Race Simulation

Yaseen Mehraf Alam, SID: 230069383

Part I

1)

Encapsulation is a fundamental pillar of Object-Oriented Programming (OOP) that specifies how related data are grouped together into one unit (a class) and methods that use the data for certain actions, which can be separated into *accessor* and *mutator* methods. Where *accessors* fetch data and return it, whilst *mutators* change the value of existing variables. This ensures that data cannot be changed in unexpected ways and have only certain methods to access or mutate data with, which increases the security of the data stored.

Within the class Horse, the methods in each category:

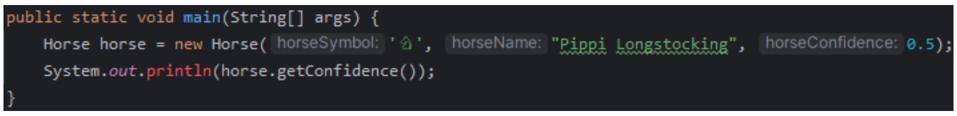
Accessors	Mutators
getConfidence()	Horse()
getDistanceTravelled()	fall()
getName()	goBackToStart()
getSymbol()	moveForward()
hasFallen()	setConfidence()
	setSymbol()

Testing:

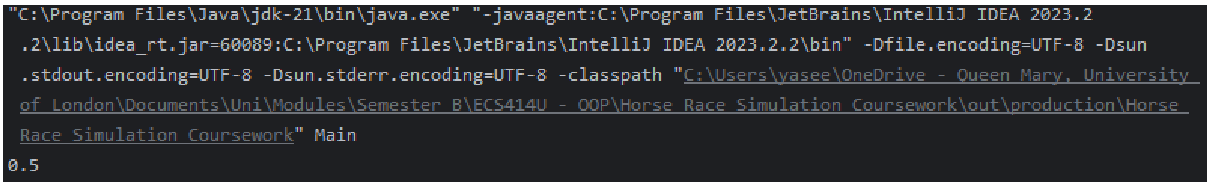
Accessors:

getConfidence():

This method fetches the value of horseConfidence after it has been stored and returns it. In this test we can see the value from the method is the same as the value set before using Horse().



```
public static void main(String[] args) {
    Horse horse = new Horse( horseSymbol: '♘',  horseName: "Pippi Longstocking",  horseConfidence: 0.5);
    System.out.println(horse.getConfidence());
}
```

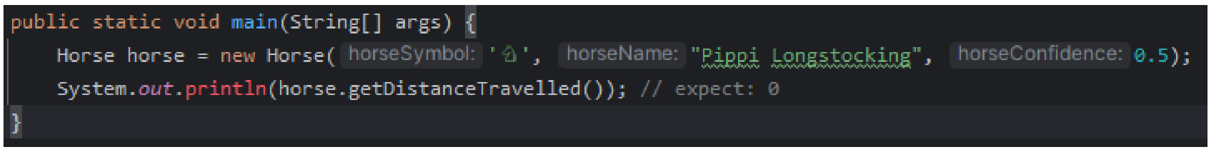


```
"C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2
 .2\lib\idea_rt.jar=60089:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2.2\bin" -Dfile.encoding=UTF-8 -Dsun
 .stdout.encoding=UTF-8 -Dsun.stderr.encoding=UTF-8 -classpath "C:\Users\yasee\OneDrive - Queen Mary, University
 of London\Documents\Uni\Modules\Semester B\ECS414U - OOP\Horse Race Simulation Coursework\out\production\Horse
 Race Simulation Coursework" Main
0.5
```

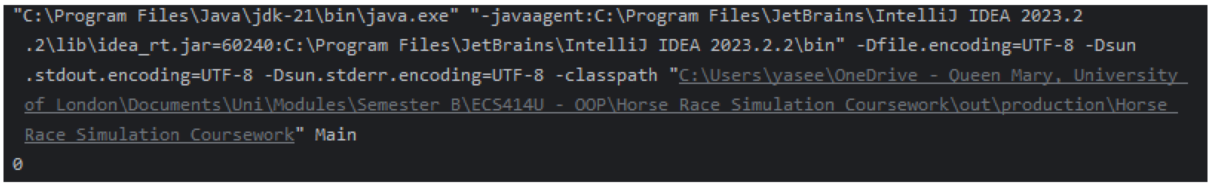
getDistanceTravelled():

This method fetches the value of distanceTravelled after it has been stored and returns it. In the 1st test we see the default value of distanceTravelled being returned as not methods of moveForward() had been called. In the 2nd test we run the method moveForward() five times and the expected value of 5 for distanceTravelled is returned when getDistanceTravelled is called.

1.

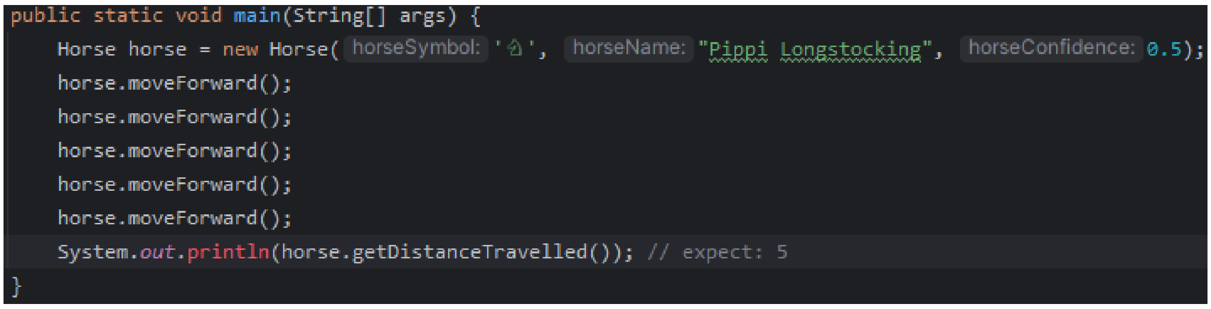


```
public static void main(String[] args) {
    Horse horse = new Horse( horseSymbol: '♘',  horseName: "Pippi Longstocking",  horseConfidence: 0.5);
    System.out.println(horse.getDistanceTravelled()); // expect: 0
}
```

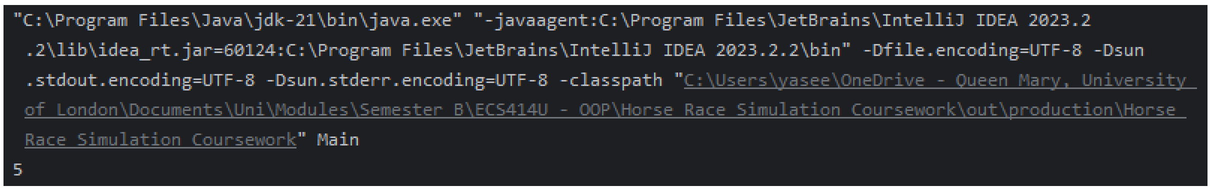


```
"C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2
 .2\lib\idea_rt.jar=60240:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2.2\bin" -Dfile.encoding=UTF-8 -Dsun
 .stdout.encoding=UTF-8 -Dsun.stderr.encoding=UTF-8 -classpath "C:\Users\yasee\OneDrive - Queen Mary, University
 of London\Documents\Uni\Modules\Semester B\ECS414U - OOP\Horse Race Simulation Coursework\out\production\Horse
 Race Simulation Coursework" Main
0
```

2.



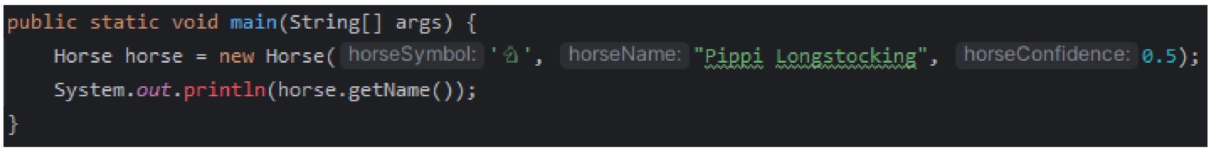
```
public static void main(String[] args) {
    Horse horse = new Horse( horseSymbol: '♘',  horseName: "Pippi Longstocking",  horseConfidence: 0.5);
    horse.moveForward();
    horse.moveForward();
    horse.moveForward();
    horse.moveForward();
    horse.moveForward();
    System.out.println(horse.getDistanceTravelled()); // expect: 5
}
```



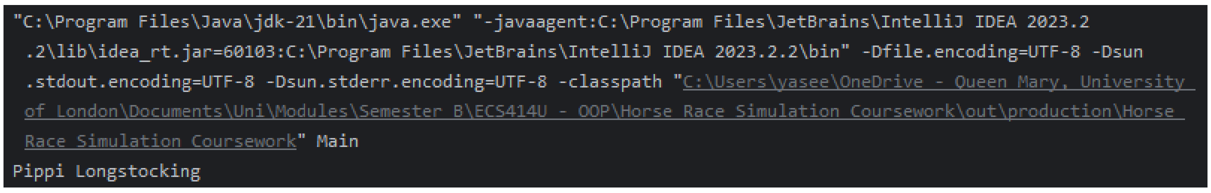
```
"C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2
 .2\lib\idea_rt.jar=60124:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2.2\bin" -Dfile.encoding=UTF-8 -Dsun
 .stdout.encoding=UTF-8 -Dsun.stderr.encoding=UTF-8 -classpath "C:\Users\yasee\OneDrive - Queen Mary, University
 of London\Documents\Uni\Modules\Semester B\ECS414U - OOP\Horse Race Simulation Coursework\out\production\Horse
 Race Simulation Coursework" Main
5
```

getName():

This method fetches the value of horseName after it has been stored and returns it. In this test we can see the value from the method is the same as the value set before using Horse().



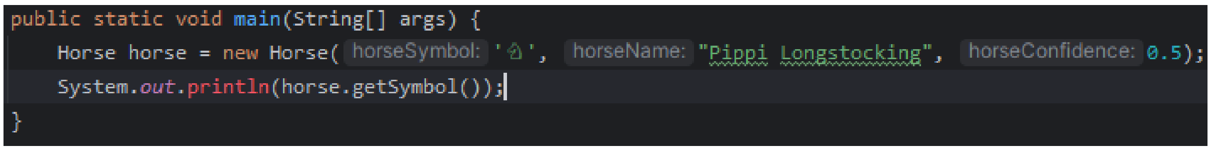
```
public static void main(String[] args) {
    Horse horse = new Horse( horseSymbol: '♘',  horseName: "Pippi Longstocking",  horseConfidence: 0.5);
    System.out.println(horse.getName());
}
```



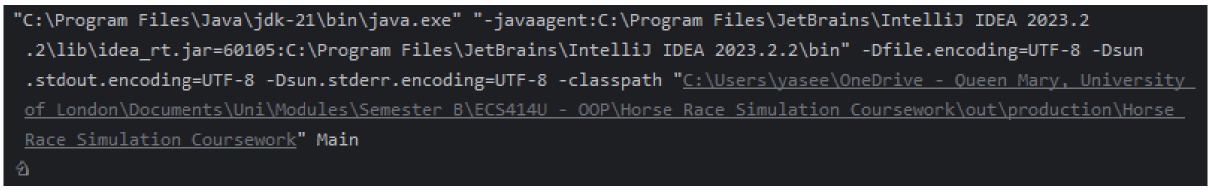
```
"C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2
 .2\lib\idea_rt.jar=60103:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2.2\bin" -Dfile.encoding=UTF-8 -Dsun
 .stdout.encoding=UTF-8 -Dsun.stderr.encoding=UTF-8 -classpath "C:\Users\yasee\OneDrive - Queen Mary, University
 of London\Documents\Uni\Modules\Semester B\ECS414U - OOP\Horse Race Simulation Coursework\out\production\Horse
 Race Simulation Coursework" Main
Pippi Longstocking
```

getSymbol():

This method fetches the value of horseSymbol after it has been stored and returns it. In this test we can see the value from the method is the same as the value set before using Horse().



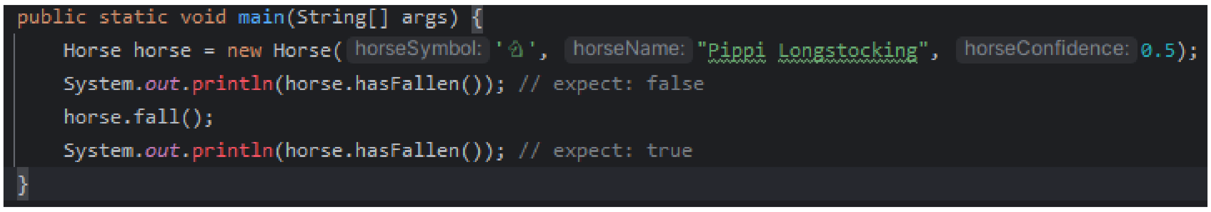
```
public static void main(String[] args) {
    Horse horse = new Horse( horseSymbol: '♘',  horseName: "Pippi Longstocking",  horseConfidence: 0.5);
    System.out.println(horse.getSymbol());
}
```



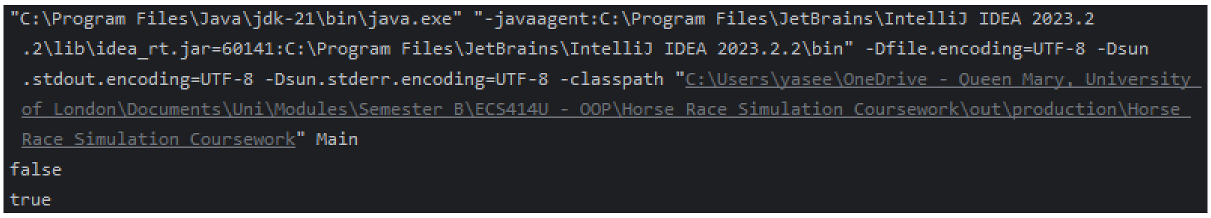
```
"C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2
 .2\lib\idea_rt.jar=60105:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2.2\bin" -Dfile.encoding=UTF-8 -Dsun
 .stdout.encoding=UTF-8 -Dsun.stderr.encoding=UTF-8 -classpath "C:\Users\yasee\OneDrive - Queen Mary, University
 of London\Documents\Uni\Modules\Semester B\ECS414U - OOP\Horse Race Simulation Coursework\out\production\Horse
 Race Simulation Coursework" Main
 ♘
```

hasFallen():

This method fetches the value of horseFallen after it has been stored and returns it. In this test the default value can be seen to be *false* after using this method. Then we can also see that after using the method fall(), the fetched value is *true* which matches the expected value.



```
public static void main(String[] args) {
    Horse horse = new Horse( horseSymbol: '♘',  horseName: "Pippi Longstocking",  horseConfidence: 0.5);
    System.out.println(horse.hasFallen()); // expect: false
    horse.fall();
    System.out.println(horse.hasFallen()); // expect: true
}
```



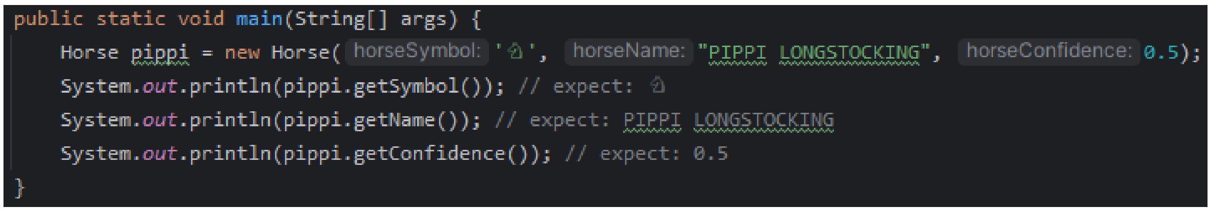
```
"C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2
 .2\lib\idea_rt.jar=60141:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2.2\bin" -Dfile.encoding=UTF-8 -Dsun
 .stdout.encoding=UTF-8 -Dsun.stderr.encoding=UTF-8 -classpath "C:\Users\yasee\OneDrive - Queen Mary, University
 of London\Documents\Uni\Modules\Semester B\ECS414U - OOP\Horse Race Simulation Coursework\out\production\Horse
 Race Simulation Coursework" Main
false
true
```

Mutators:

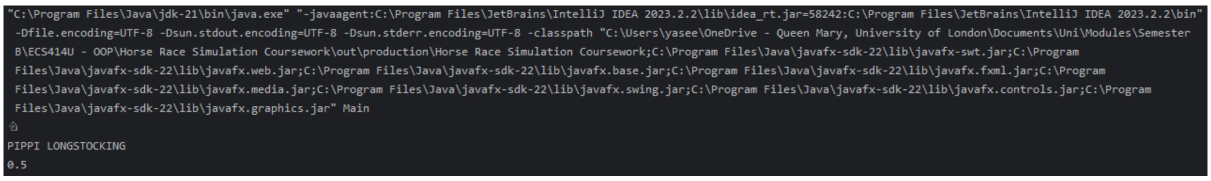
Horse():

This method acts as a constructor for the Horse class, and sets the values of horseSymbol, horseName and horseConfidence, which can be seen to match the values when used the methods getSymbol(), getName() and getConfidence(). I also included edge cases for when the value of horseConfidence is set to a value less than 0 and greater than 1, to be defaulted to a value of 0.5.

1.

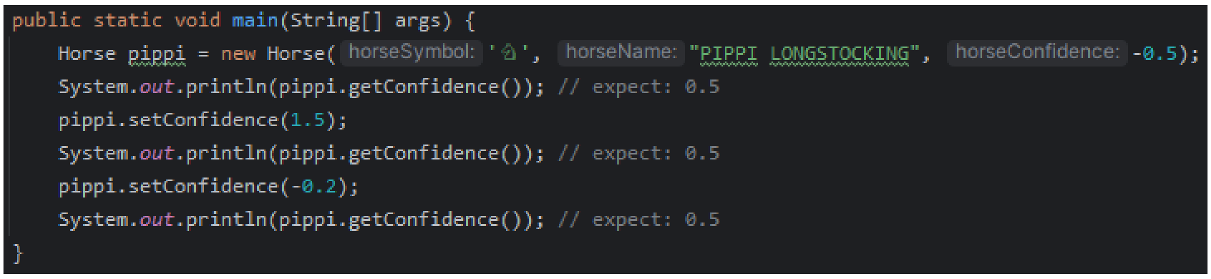


```
public static void main(String[] args) {
    Horse pippi = new Horse( horseSymbol: '♘',  horseName: "PIPPI LONGSTOCKING",  horseConfidence: 0.5);
    System.out.println(pippi.getSymbol()); // expect: ♘
    System.out.println(pippi.getName()); // expect: PIPPI LONGSTOCKING
    System.out.println(pippi.getConfidence()); // expect: 0.5
}
```

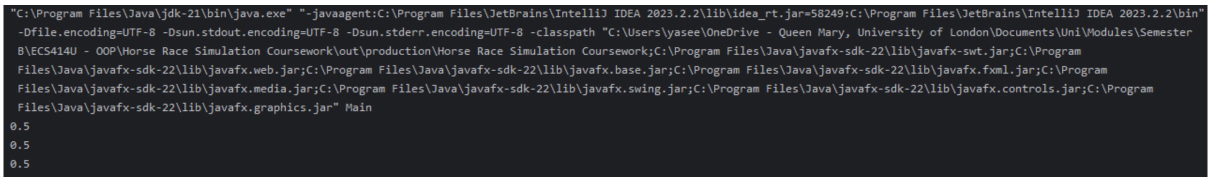


```
"C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2.2\lib\idea_rt.jar=58242:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2.2\bin"
 -Dfile.encoding=UTF-8 -Dsun.stdout.encoding=UTF-8 -Dsun.stderr.encoding=UTF-8 -classpath "C:\Users\yasee\OneDrive - Queen Mary, University of London\Documents\Uni\Modules\Semester
 B\ECS414U - OOP\Horse Race Simulation Coursework\out\production\Horse Race Simulation Coursework;C:\Program Files\Java\javafx-sdk-22\lib\javafx-swt.jar;C:\Program
 Files\Java\javafx-sdk-22\lib\javafx.web.jar;C:\Program Files\Java\javafx-sdk-22\lib\javafx.base.jar;C:\Program Files\Java\javafx-sdk-22\lib\javafx.fxml.jar;C:\Program
 Files\Java\javafx-sdk-22\lib\javafx.media.jar;C:\Program Files\Java\javafx-sdk-22\lib\javafx.swing.jar;C:\Program Files\Java\javafx-sdk-22\lib\javafx.controls.jar;C:\Program
 Files\Java\javafx-sdk-22\lib\javafx.graphics.jar" Main
♘
PIPPI LONGSTOCKING
0.5
```

2.



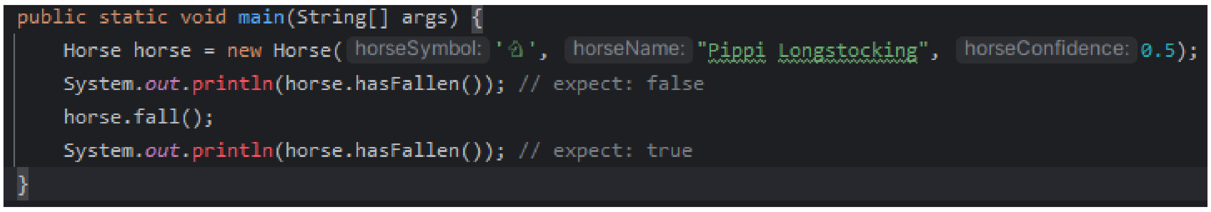
```
public static void main(String[] args) {
    Horse pippi = new Horse( horseSymbol: '♘', horseName: "PIPPI LONGSTOCKING", horseConfidence: -0.5);
    System.out.println(pippi.getConfidence()); // expect: 0.5
    pippi.setConfidence(1.5);
    System.out.println(pippi.getConfidence()); // expect: 0.5
    pippi.setConfidence(-0.2);
    System.out.println(pippi.getConfidence()); // expect: 0.5
}
```



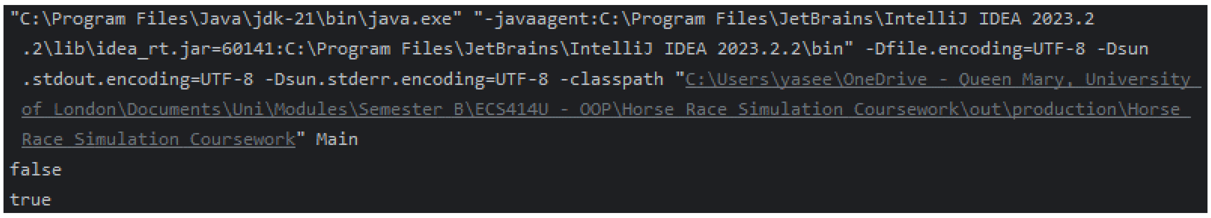
```
"C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2.2\lib\idea_rt.jar=58249:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2.2\bin"
 -Dfile.encoding=UTF-8 -Dsun.stdout.encoding=UTF-8 -Dsun.stderr.encoding=UTF-8 -classpath "C:\Users\yasee\OneDrive - Queen Mary, University of London\Documents\Uni\Modules\Semester
 B\ECS414U - OOP\Horse Race Simulation Coursework\out\production\Horse Race Simulation Coursework;C:\Program Files\Java\javafx-sdk-22\lib\javafx-swt.jar;C:\Program
 Files\Java\javafx-sdk-22\lib\javafx.web.jar;C:\Program Files\Java\javafx-sdk-22\lib\javafx.base.jar;C:\Program Files\Java\javafx-sdk-22\lib\javafx.fxml.jar;C:\Program
 Files\Java\javafx-sdk-22\lib\javafx.media.jar;C:\Program Files\Java\javafx-sdk-22\lib\javafx.swing.jar;C:\Program Files\Java\javafx-sdk-22\lib\javafx.controls.jar;C:\Program
 Files\Java\javafx-sdk-22\lib\javafx.graphics.jar" Main
0.5
0.5
0.5
```

fall():

This method sets the value of horseFallen to true whenever the method is called. In this test we can see the default value of false is printed, and when fall() has been used the value assigned is true, these results match the expected values.



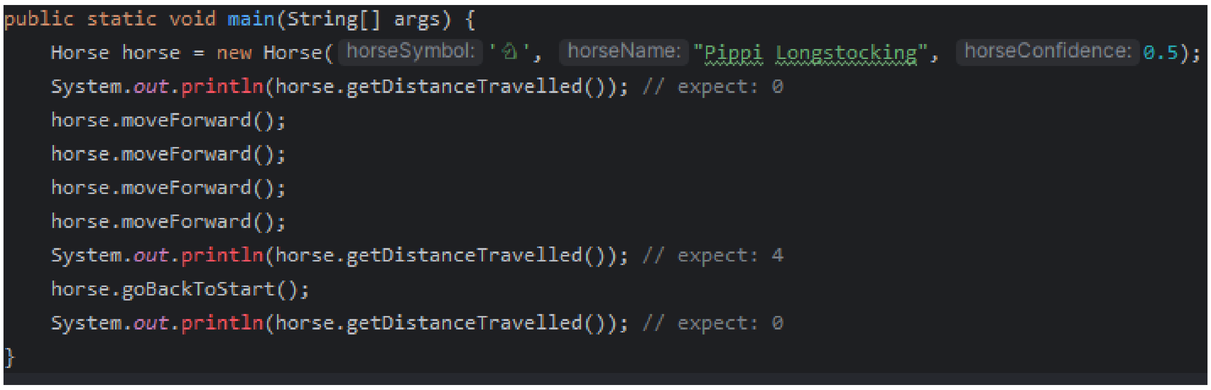
```
public static void main(String[] args) {
    Horse horse = new Horse( horseSymbol: '♘', horseName: "Pippi Longstocking", horseConfidence: 0.5);
    System.out.println(horse.hasFallen()); // expect: false
    horse.fall();
    System.out.println(horse.hasFallen()); // expect: true
}
```



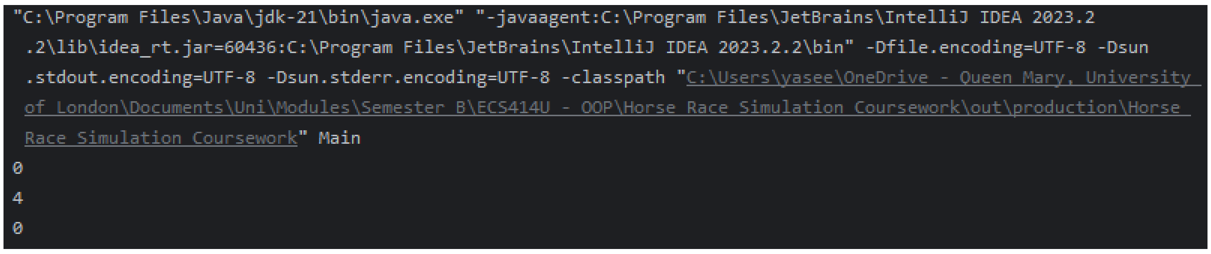
```
"C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2
 .2\lib\idea_rt.jar=60141:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2.2\bin" -Dfile.encoding=UTF-8 -Dsun
 .stdout.encoding=UTF-8 -Dsun.stderr.encoding=UTF-8 -classpath "C:\Users\yasee\OneDrive - Queen Mary, University
 of London\Documents\Uni\Modules\Semester B\ECS414U - OOP\Horse Race Simulation Coursework\out\production\Horse
 Race Simulation Coursework" Main
false
true
```

goBackToStart():

In this method the value of distanceTravelled is set to 0 whenever the method is called. In this test we can see that the value initially is 0, as the default value of distanceTravelled and then when the method moveForward() has been called exactly 4 times, the value of distanceTravelled becomes 4, and when goBackToStart() is called the value of distanceTravelled is set to 0 again. The expected values match the results.



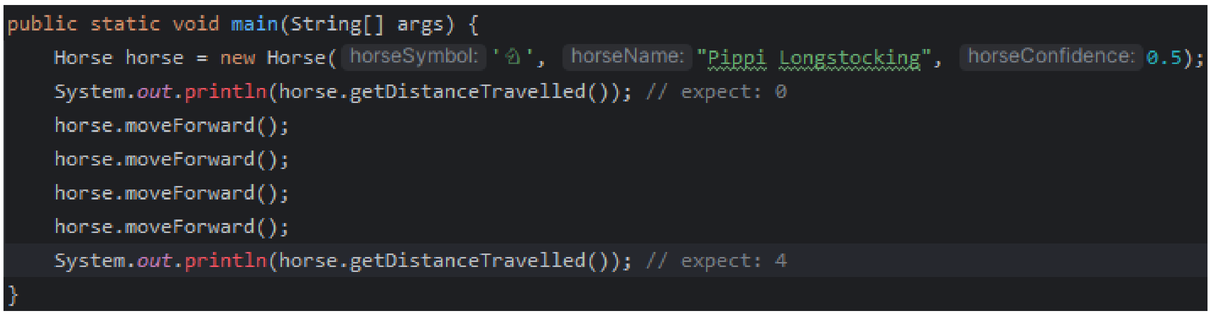
```
public static void main(String[] args) {
    Horse horse = new Horse( horseSymbol: '♘', horseName: "Pippi Longstocking", horseConfidence: 0.5);
    System.out.println(horse.getDistanceTravelled()); // expect: 0
    horse.moveForward();
    horse.moveForward();
    horse.moveForward();
    horse.moveForward();
    System.out.println(horse.getDistanceTravelled()); // expect: 4
    horse.goBackToStart();
    System.out.println(horse.getDistanceTravelled()); // expect: 0
}
```



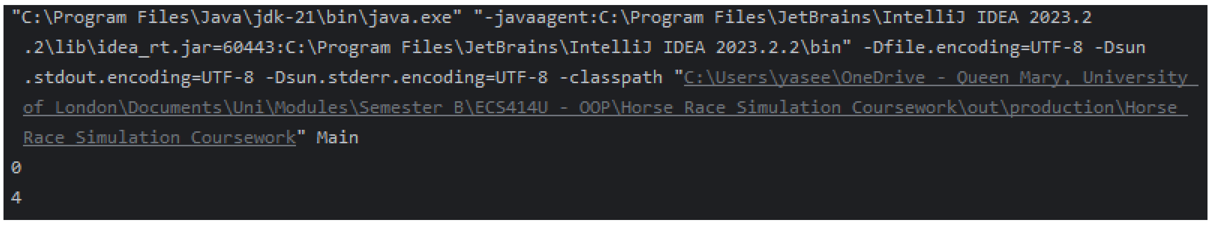
```
"C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2
 .2\lib\idea_rt.jar=60436:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2.2\bin" -Dfile.encoding=UTF-8 -Dsun
 .stdout.encoding=UTF-8 -Dsun.stderr.encoding=UTF-8 -classpath "C:\Users\yasee\OneDrive - Queen Mary, University
 of London\Documents\Uni\Modules\Semester B\ECS414U - OOP\Horse Race Simulation Coursework\out\production\Horse
 Race Simulation Coursework" Main
0
4
0
```

moveForward():

This method is used to increment the value of distanceTravelled by 1 everytime the method is called, and we can see the effect of it after four calls of the method in this test. The initial value of 0, can be seen to be then set to a value of 4 at the end of the test. This test's results match the expected values.



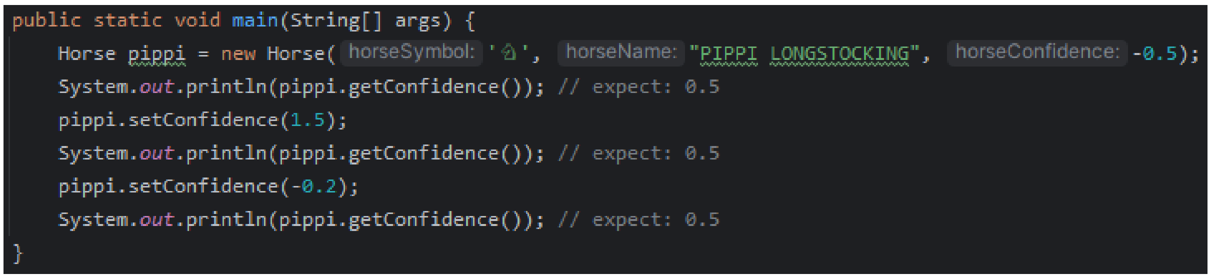
```
public static void main(String[] args) {
    Horse horse = new Horse( horseSymbol: '♘', horseName: "Pippi Longstocking", horseConfidence: 0.5);
    System.out.println(horse.getDistanceTravelled()); // expect: 0
    horse.moveForward();
    horse.moveForward();
    horse.moveForward();
    horse.moveForward();
    System.out.println(horse.getDistanceTravelled()); // expect: 4
}
```



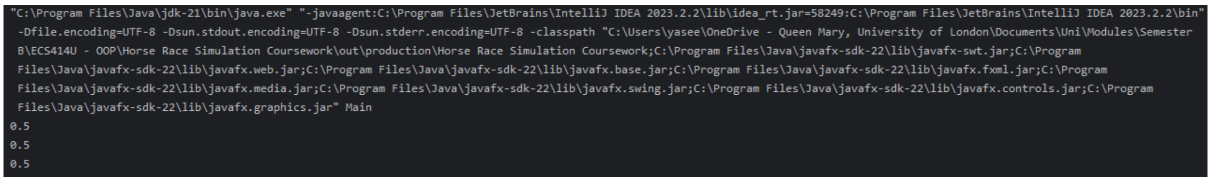
```
"C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2
 .2\lib\idea_rt.jar=60443:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2.2\bin" -Dfile.encoding=UTF-8 -Dsun
 .stdout.encoding=UTF-8 -Dsun.stderr.encoding=UTF-8 -classpath "C:\Users\yasee\OneDrive - Queen Mary, University
 of London\Documents\Uni\Modules\Semester B\ECS414U - OOP\Horse Race Simulation Coursework\out\production\Horse
 Race Simulation Coursework" Main
0
4
```

setConfidence():

This method sets the value of horseConfidence to a new value whenever the method is called with the value newConfidence from its parameter. I also included edge cases for when the value of horseConfidence is set to a value less than 0 and greater than 1, to be defaulted to a value of 0.5. In this test several calls of the setConfidence() method is used to see The results match the expected values.



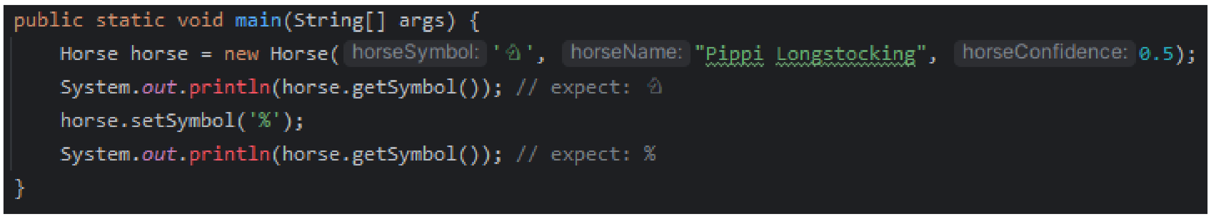
```
public static void main(String[] args) {
    Horse pippi = new Horse( horseSymbol: '♘', horseName: "PIPPI LONGSTOCKING", horseConfidence: -0.5);
    System.out.println(pippi.getConfidence()); // expect: 0.5
    pippi.setConfidence(1.5);
    System.out.println(pippi.getConfidence()); // expect: 0.5
    pippi.setConfidence(-0.2);
    System.out.println(pippi.getConfidence()); // expect: 0.5
}
```



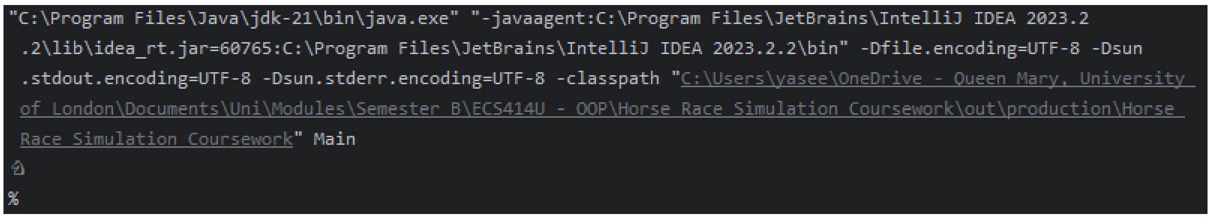
```
"C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2.2\lib\idea_rt.jar=58249:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2.2\bin"
 -Dfile.encoding=UTF-8 -Dsun.stdout.encoding=UTF-8 -Dsun.stderr.encoding=UTF-8 -classpath "C:\Users\yasee\OneDrive - Queen Mary, University of London\Documents\Uni\Modules\Semester
 B\ECS414U - OOP\Horse Race Simulation Coursework\out\production\Horse Race Simulation Coursework;C:\Program Files\Java\javafx-sdk-22\lib\javafx-swt.jar;C:\Program
 Files\Java\javafx-sdk-22\lib\javafx.web.jar;C:\Program Files\Java\javafx-sdk-22\lib\javafx.base.jar;C:\Program Files\Java\javafx-sdk-22\lib\javafx.fxml.jar;C:\Program
 Files\Java\javafx-sdk-22\lib\javafx.media.jar;C:\Program Files\Java\javafx-sdk-22\lib\javafx.swing.jar;C:\Program Files\Java\javafx-sdk-22\lib\javafx.controls.jar;C:\Program
 Files\Java\javafx-sdk-22\lib\javafx.graphics.jar" Main
0.5
0.5
0.5
```

setSymbol():

In this method the value of horseSymbol is assigned to a new value using the value of newSymbol form the parameter. In this test the initial value of horseSymbol is set by the constructor Horse() to ♘ and then using setSymbol() the value of horseSymbol is set to %. The results match the expected values.

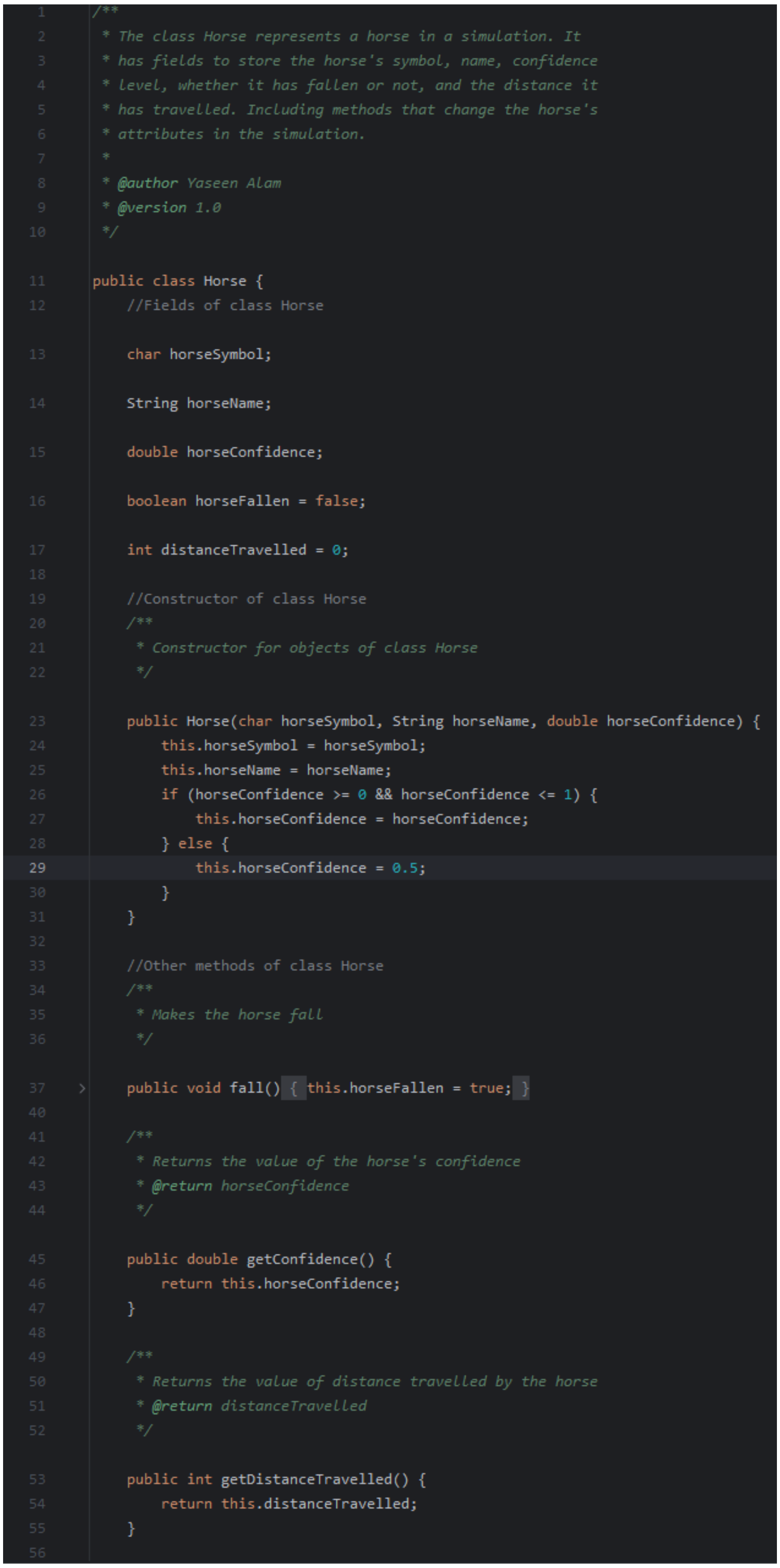


```
public static void main(String[] args) {
    Horse horse = new Horse( horseSymbol: '♘', horseName: "Pippi Longstocking", horseConfidence: 0.5);
    System.out.println(horse.getSymbol()); // expect: ♘
    horse.setSymbol('%');
    System.out.println(horse.getSymbol()); // expect: %
}
```

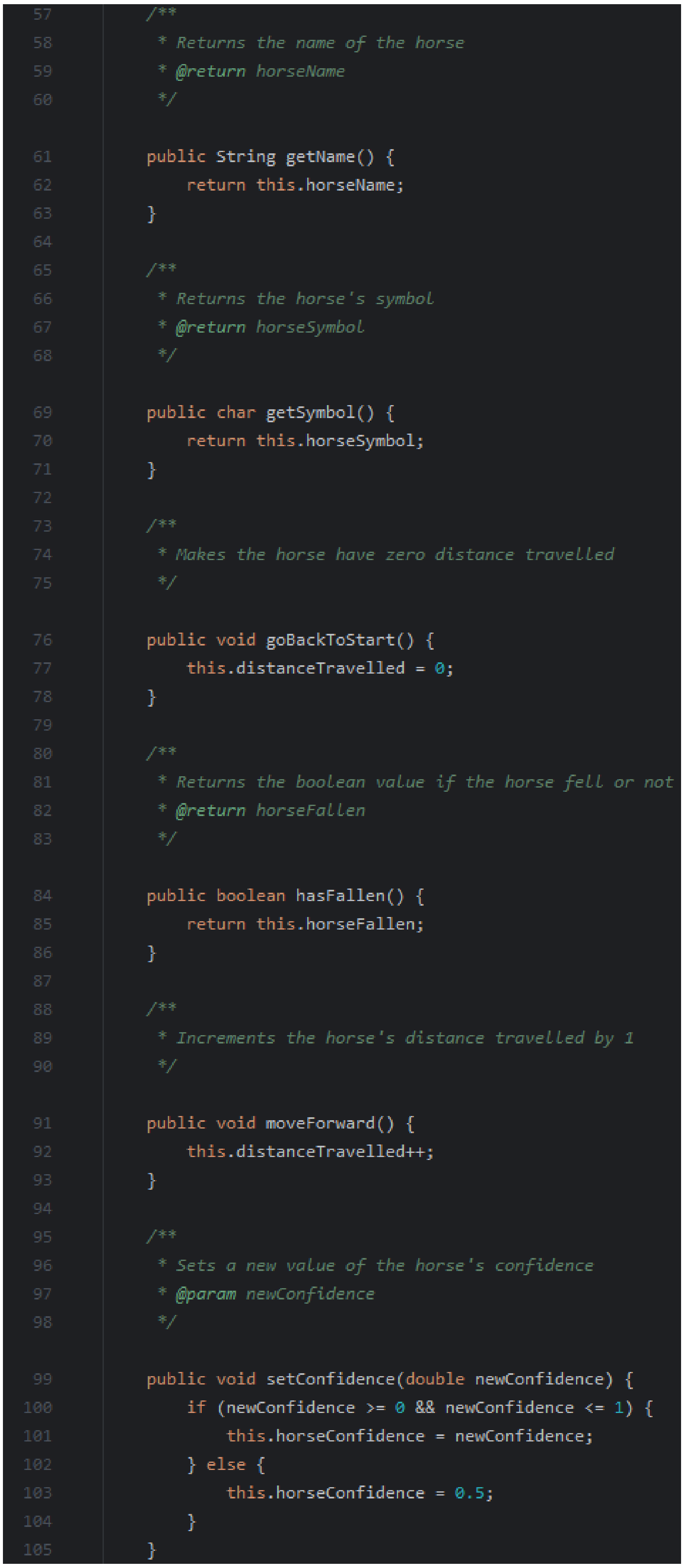


```
"C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2
 .2\lib\idea_rt.jar=60765:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2.2\bin" -Dfile.encoding=UTF-8 -Dsun
 .stdout.encoding=UTF-8 -Dsun.stderr.encoding=UTF-8 -classpath "C:\Users\yasee\OneDrive - Queen Mary, University
 of London\Documents\Uni\Modules\Semester B\ECS414U - OOP\Horse Race Simulation Coursework\out\production\Horse
 Race Simulation Coursework" Main
♘
%
```

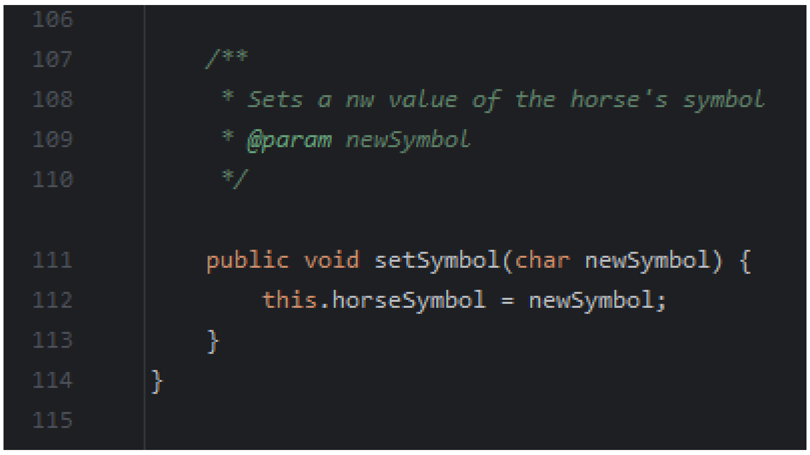
Horse.java code:



```java
/**
 * The class Horse represents a horse in a simulation. It
 * has fields to store the horse's symbol, name, confidence
 * level, whether it has fallen or not, and the distance it
 * has travelled. Including methods that change the horse's
 * attributes in the simulation.
 *
 * @author Yaseen Alam
 * @version 1.0
 */

public class Horse {
    //Fields of class Horse

    char horseSymbol;

    String horseName;

    double horseConfidence;

    boolean horseFallen = false;

    int distanceTravelled = 0;

    //Constructor of class Horse
    /**
     * Constructor for objects of class Horse
     */

    public Horse(char horseSymbol, String horseName, double horseConfidence) {
        this.horseSymbol = horseSymbol;
        this.horseName = horseName;
        if (horseConfidence >= 0 && horseConfidence <= 1) {
            this.horseConfidence = horseConfidence;
        } else {
            this.horseConfidence = 0.5;
        }
    }

    //Other methods of class Horse
    /**
     * Makes the horse fall
     */

    public void fall() { this.horseFallen = true; }

    /**
     * Returns the value of the horse's confidence
     * @return horseConfidence
     */

    public double getConfidence() {
        return this.horseConfidence;
    }

    /**
     * Returns the value of distance travelled by the horse
     * @return distanceTravelled
     */

    public int getDistanceTravelled() {
        return this.distanceTravelled;
    }

```



```java
    /**
     * Returns the name of the horse
     * @return horseName
     */

    public String getName() {
        return this.horseName;
    }

    /**
     * Returns the horse's symbol
     * @return horseSymbol
     */

    public char getSymbol() {
        return this.horseSymbol;
    }

    /**
     * Makes the horse have zero distance travelled
     */

    public void goBackToStart() {
        this.distanceTravelled = 0;
    }

    /**
     * Returns the boolean value if the horse fell or not
     * @return horseFallen
     */

    public boolean hasFallen() {
        return this.horseFallen;
    }

    /**
     * Increments the horse's distance travelled by 1
     */

    public void moveForward() {
        this.distanceTravelled++;
    }

    /**
     * Sets a new value of the horse's confidence
     * @param newConfidence
     */

    public void setConfidence(double newConfidence) {
        if (newConfidence >= 0 && newConfidence <= 1) {
            this.horseConfidence = newConfidence;
        } else {
            this.horseConfidence = 0.5;
        }
    }
```



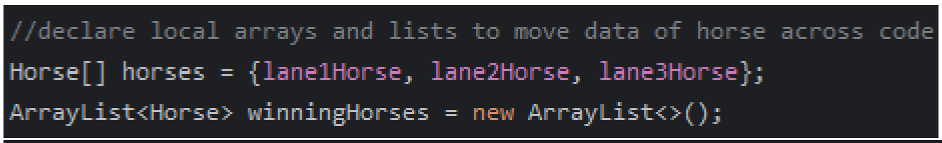
```
    /**
     * Sets a nw value of the horse's symbol
     * @param newSymbol
     */

    public void setSymbol(char newSymbol) {
        this.horseSymbol = newSymbol;
    }
}
```

2)

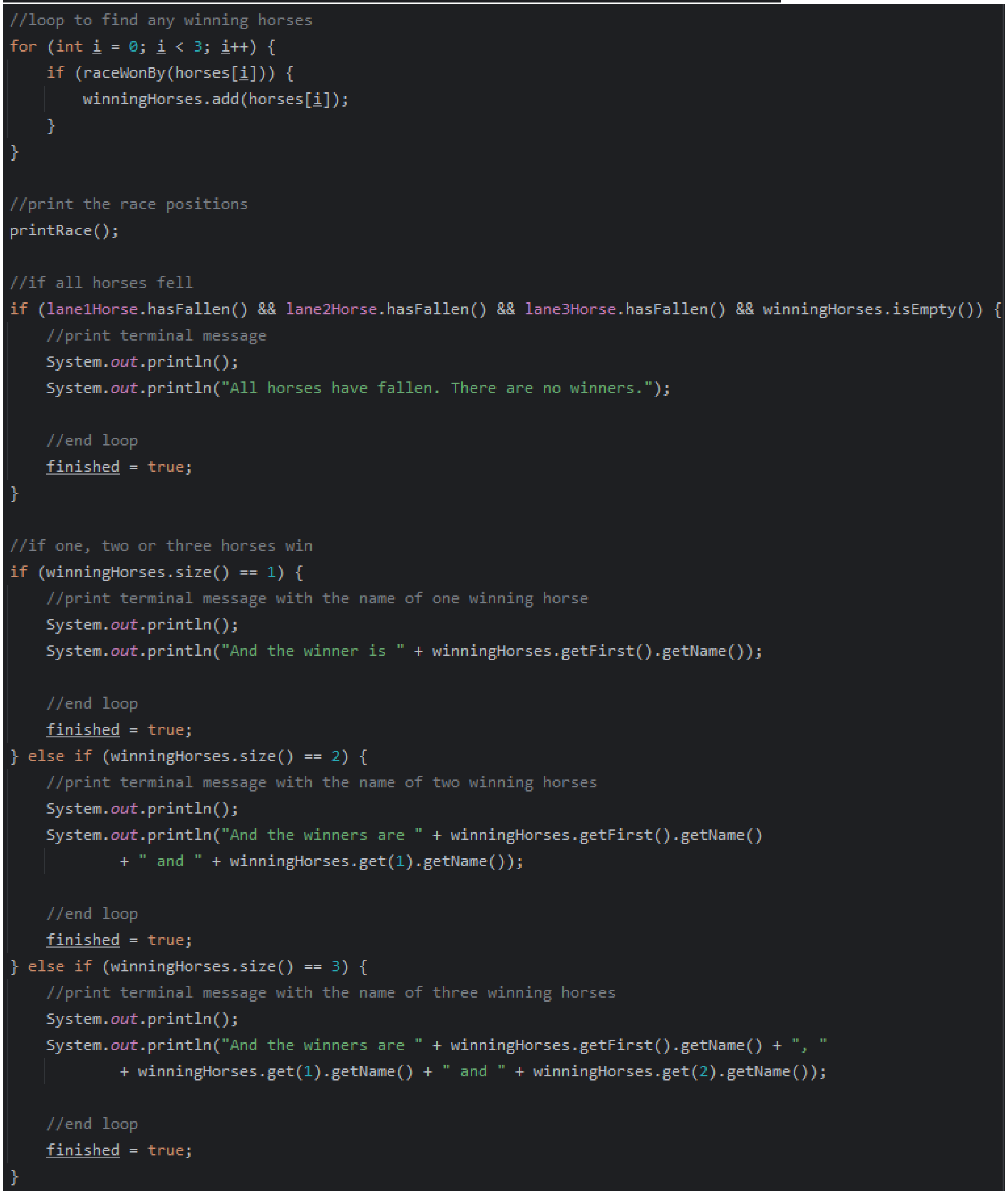
Adding terminal message

Once the race is finished, there is a message is displayed when a horse has won the match, when one or more horses tied and when no horse wins and all horses fall.

Main Code:

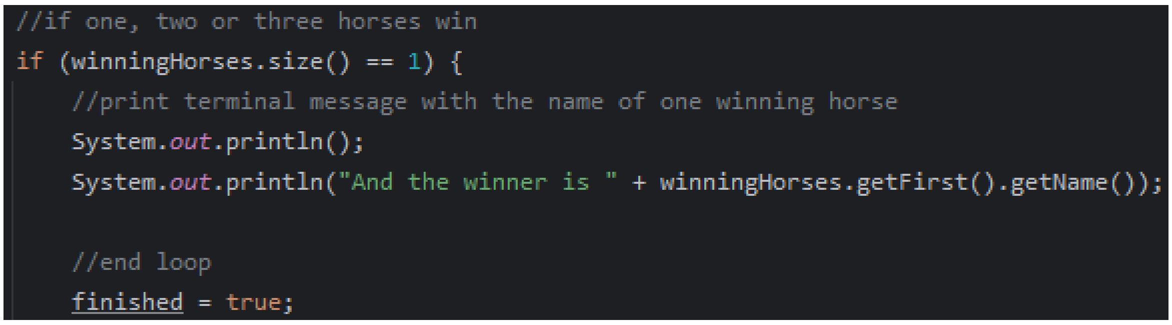


```
//declare local arrays and lists to move data of horse across code
Horse[] horses = {lane1Horse, lane2Horse, lane3Horse};
ArrayList<Horse> winningHorses = new ArrayList<>();
```

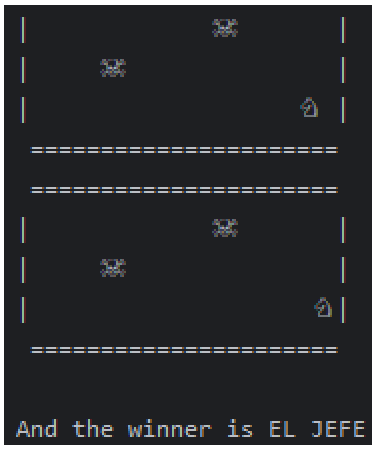


```
//loop to find any winning horses
for (int i = 0; i < 3; i++) {
    if (raceWonBy(horses[i])) {
        winningHorses.add(horses[i]);
    }
}

//print the race positions
printRace();

//if all horses fell
if (lane1Horse.hasFallen() && lane2Horse.hasFallen() && lane3Horse.hasFallen() && winningHorses.isEmpty()) {
    //print terminal message
    System.out.println();
    System.out.println("All horses have fallen. There are no winners.");

    //end loop
    finished = true;
}

//if one, two or three horses win
if (winningHorses.size() == 1) {
    //print terminal message with the name of one winning horse
    System.out.println();
    System.out.println("And the winner is " + winningHorses.getFirst().getName());

    //end loop
    finished = true;
} else if (winningHorses.size() == 2) {
    //print terminal message with the name of two winning horses
    System.out.println();
    System.out.println("And the winners are " + winningHorses.getFirst().getName()
            + " and " + winningHorses.get(1).getName());

    //end loop
    finished = true;
} else if (winningHorses.size() == 3) {
    //print terminal message with the name of three winning horses
    System.out.println();
    System.out.println("And the winners are " + winningHorses.getFirst().getName() + ", "
            + winningHorses.get(1).getName() + " and " + winningHorses.get(2).getName());

    //end loop
    finished = true;
}
```

1. When one horse has won a match:

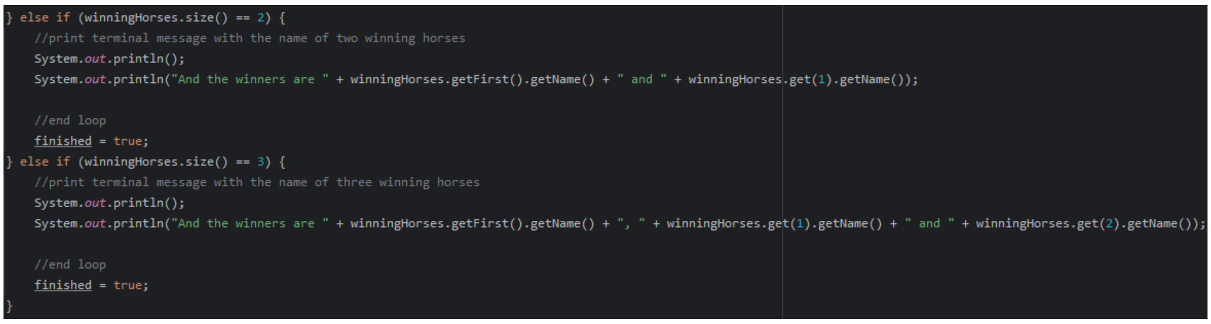


```java
//if one, two or three horses win
if (winningHorses.size() == 1) {
    //print terminal message with the name of one winning horse
    System.out.println();
    System.out.println("And the winner is " + winningHorses.getFirst().getName());

    //end loop
    finished = true;
```



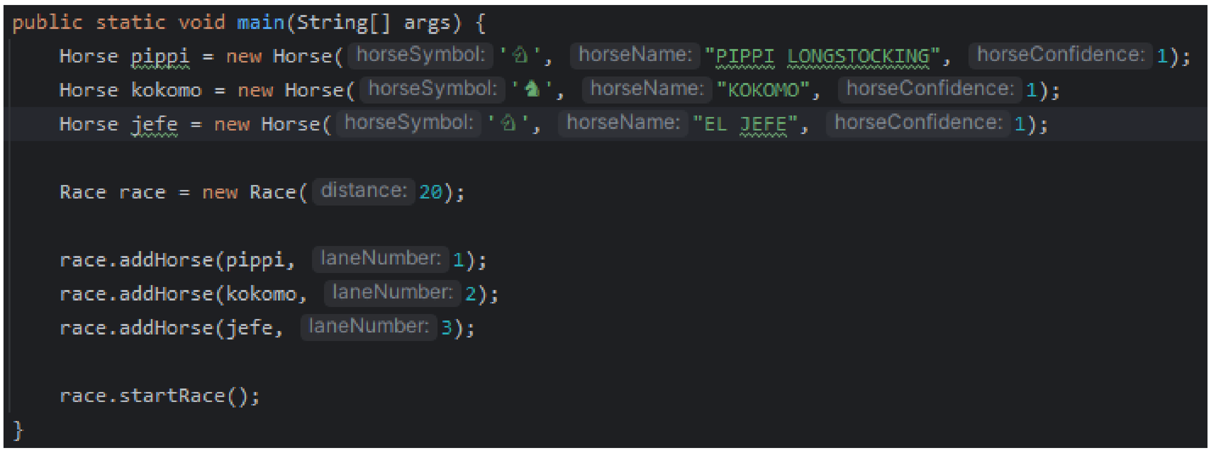
```
|            ☠         |
|     ☠                |
|                   ♘  |
 ======================
 ======================
|            ☠         |
|     ☠                |
|                    ♘ |
 ======================

And the winner is EL JEFE
```

2. When one or more horses tie:

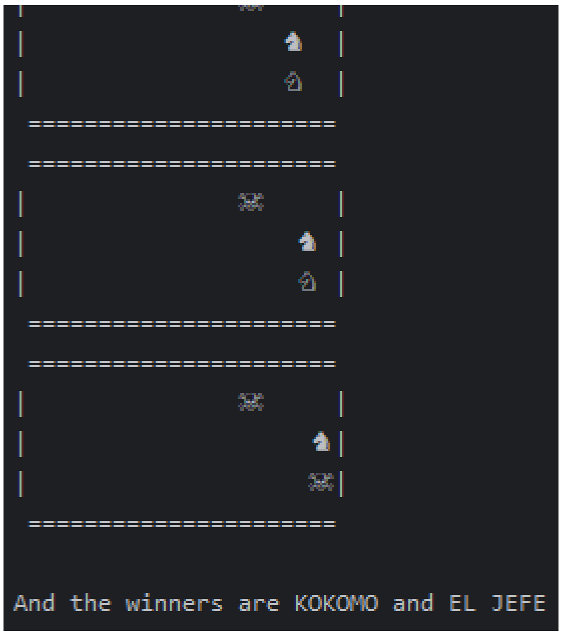


```java
} else if (winningHorses.size() == 2) {
    //print terminal message with the name of two winning horses
    System.out.println();
    System.out.println("And the winners are " + winningHorses.getFirst().getName() + " and " + winningHorses.get(1).getName());

    //end loop
    finished = true;
} else if (winningHorses.size() == 3) {
    //print terminal message with the name of three winning horses
    System.out.println();
    System.out.println("And the winners are " + winningHorses.getFirst().getName() + ", " + winningHorses.get(1).getName() + " and " + winningHorses.get(2).getName());

    //end loop
    finished = true;
}
```

Test:

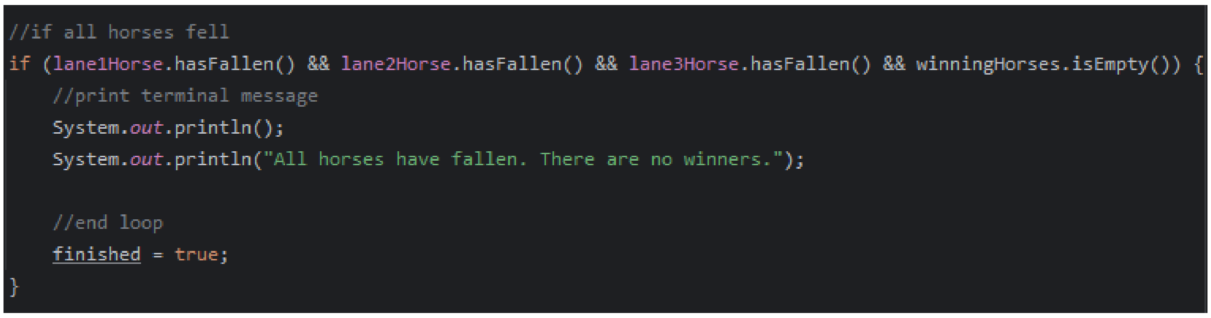


```java
public static void main(String[] args) {
    Horse pippi = new Horse( horseSymbol: '♘', horseName: "PIPPI LONGSTOCKING", horseConfidence: 1);
    Horse kokomo = new Horse( horseSymbol: '♞', horseName: "KOKOMO", horseConfidence: 1);
    Horse jefe = new Horse( horseSymbol: '♘', horseName: "EL JEFE", horseConfidence: 1);

    Race race = new Race( distance: 20);

    race.addHorse(pippi, laneNumber: 1);
    race.addHorse(kokomo, laneNumber: 2);
    race.addHorse(jefe, laneNumber: 3);

    race.startRace();
}
```

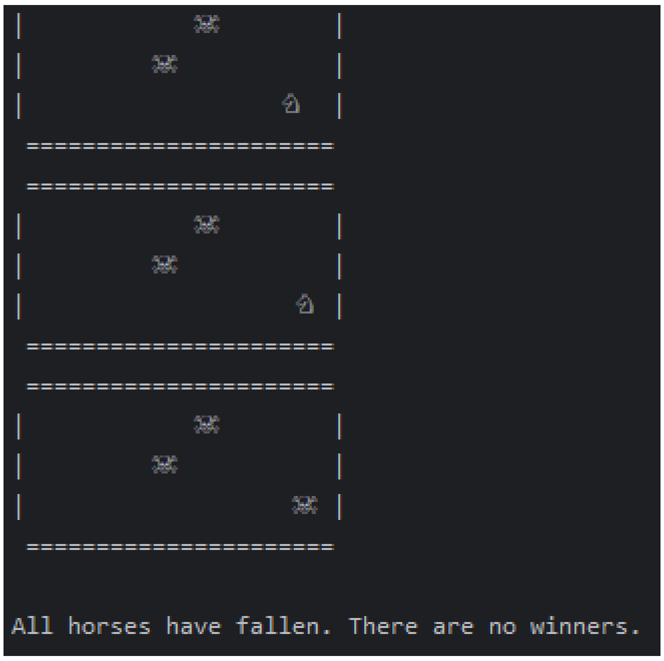


```
|               ☠       |
|                  ♞   |
|                  ♘   |
 ======================
 ======================
|               ☠       |
|                   ♞  |
|                   ♘  |
 ======================
 ======================
|               ☠       |
|                    ♞|
|                    ☠|
 ======================

And the winners are KOKOMO and EL JEFE
```

3. When all horses fall:

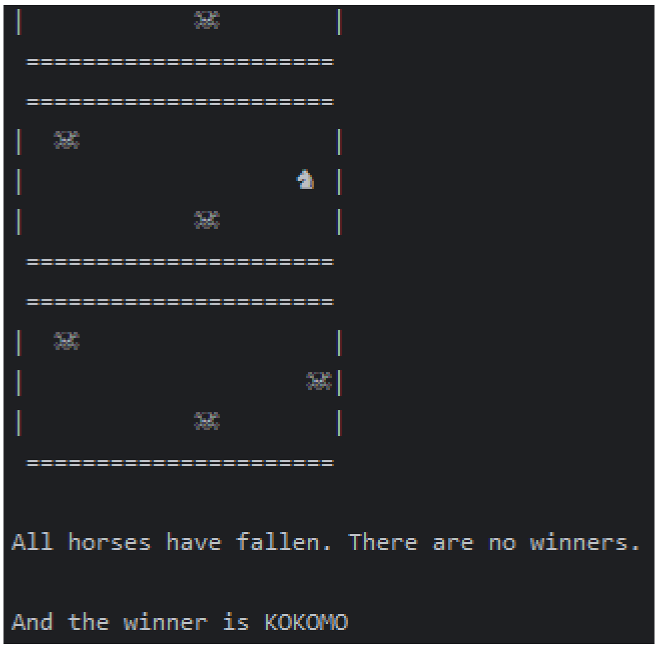


```java
//if all horses fell
if (lane1Horse.hasFallen() && lane2Horse.hasFallen() && lane3Horse.hasFallen() && winningHorses.isEmpty()) {
    //print terminal message
    System.out.println();
    System.out.println("All horses have fallen. There are no winners.");

    //end loop
    finished = true;
}
```



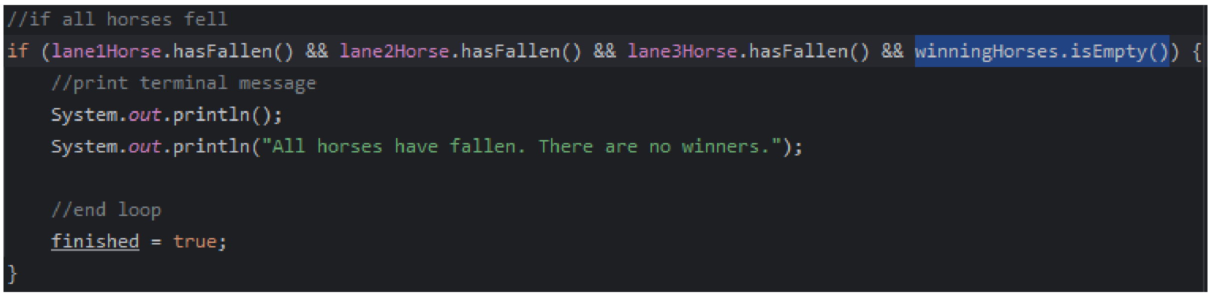
```
|             ☠         |
|          ☠            |
|                  ♘   |
 ======================
 ======================
|             ☠         |
|          ☠            |
|                   ♘  |
 ======================
 ======================
|             ☠         |
|          ☠            |
|                   ☠  |
 ======================

All horses have fallen. There are no winners.
```

Problem 1: All horses fell but the horse had also won the race.



```
|            ☠          |
 =====================
 =====================
|   ☠                  |
|                    ♞ |
|            ☠          |
 =====================
 =====================
|   ☠                  |
|                     ☠|
|            ☠          |
 =====================

All horses have fallen. There are no winners.

And the winner is KOKOMO
```

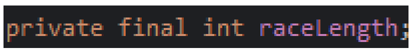
Problem 1 Solution: Added an extra condition in the conditional statement (winningHorses.isEmpty()) before deeming all horses have fallen and none of them won.



```java
//if all horses fell
if (lane1Horse.hasFallen() && lane2Horse.hasFallen() && lane3Horse.hasFallen() && winningHorses.isEmpty()) {
    //print terminal message
    System.out.println();
    System.out.println("All horses have fallen. There are no winners.");

    //end loop
    finished = true;
}
```

Making raceLength a final variable

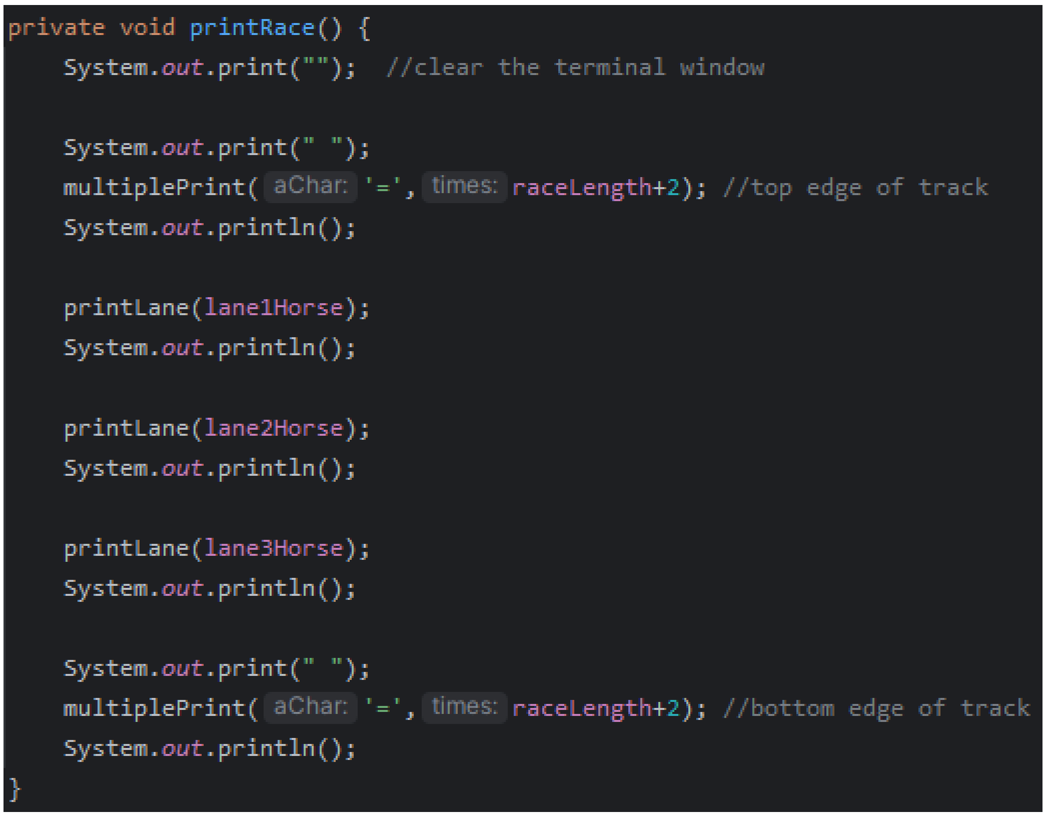
raceLength should be a final variable as it will not be changed later for a specific instance of Race.



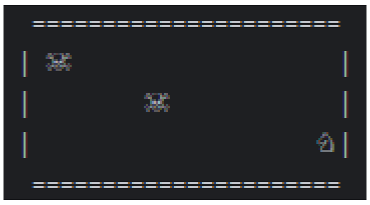
```java
private final int raceLength;
```

Fixing aesthetics of race lanes in terminal

The '=' did not line up with the center of the edges, made using '|' on each side of race track. So I fixed it using " " and one less of a '='.



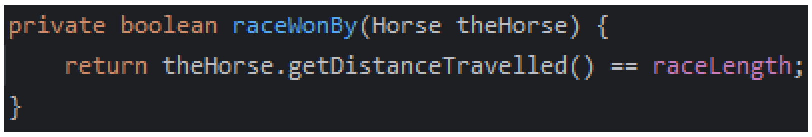
```java
private void printRace() {
    System.out.print("");  //clear the terminal window

    System.out.print(" ");
    multiplePrint( aChar: '=', times: raceLength+2); //top edge of track
    System.out.println();

    printLane(lane1Horse);
    System.out.println();

    printLane(lane2Horse);
    System.out.println();

    printLane(lane3Horse);
    System.out.println();

    System.out.print(" ");
    multiplePrint( aChar: '=', times: raceLength+2); //bottom edge of track
    System.out.println();
}
```



```
 ======================
| ☠                   |
|        ☠            |
|                    ♘|
 ======================
```

Simplifying raceWonBy() method

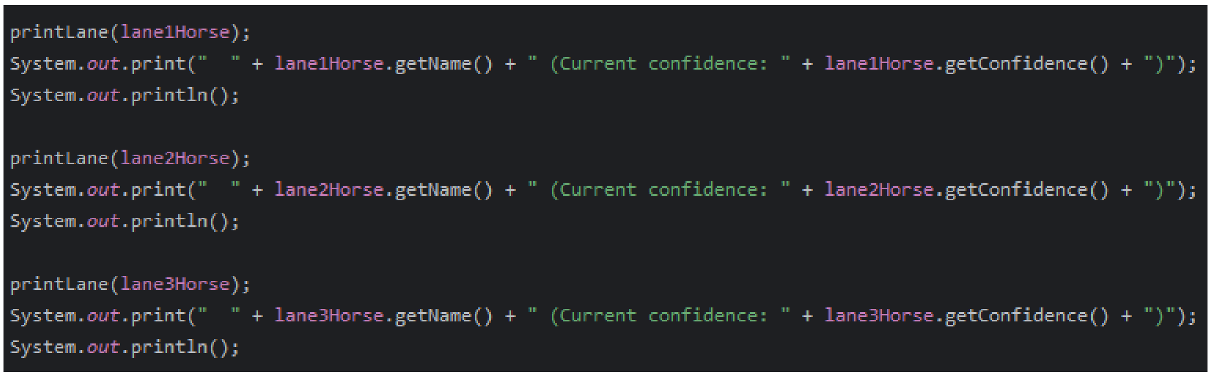
The conditional statements can be removed to simplify the method into one line by directly returning the Boolean result of the comparison expression.



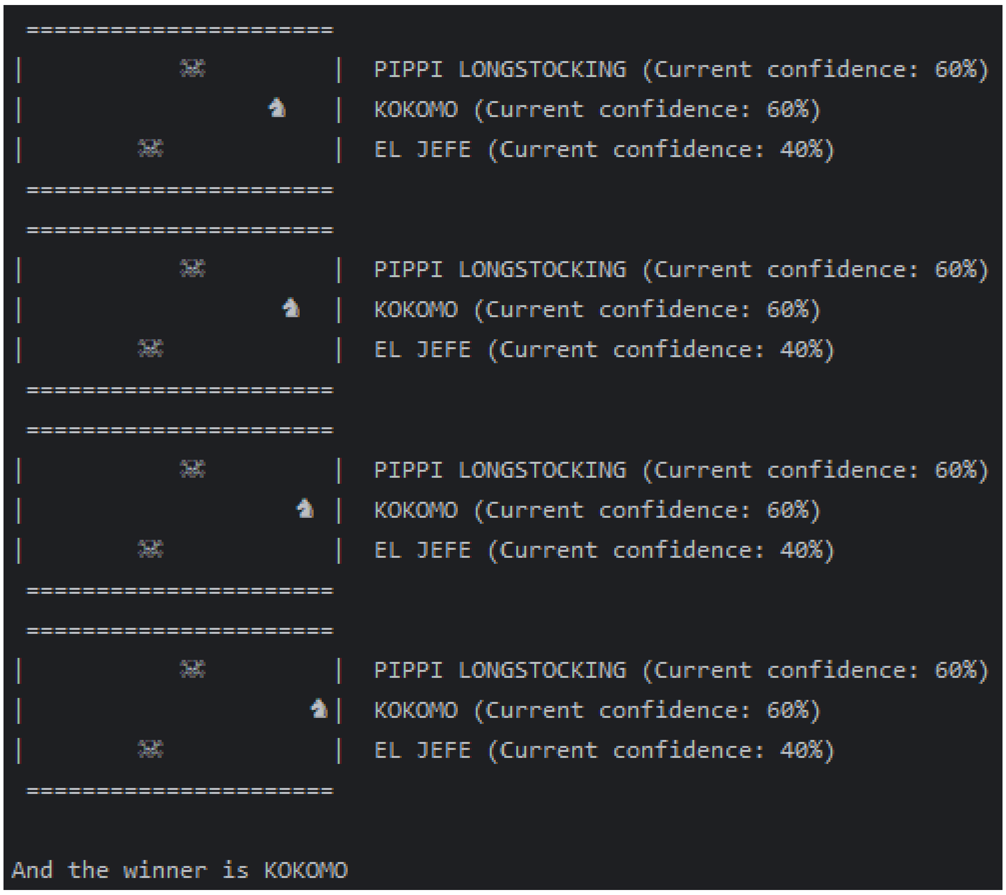
```java
private boolean raceWonBy(Horse theHorse) {
    return theHorse.getDistanceTravelled() == raceLength;
}
```

Adding horse names and confidence values next to track

This is to identify the names of the horses and confidence levels, in percentage, corresponding to the lane the horse is in. This is for easability in reading the details of the race, making it more immersive. Implemented in printRace().



```
printLane(lane1Horse);
System.out.print("  " + lane1Horse.getName() + " (Current confidence: " + lane1Horse.getConfidence() + ")");
System.out.println();

printLane(lane2Horse);
System.out.print("  " + lane2Horse.getName() + " (Current confidence: " + lane2Horse.getConfidence() + ")");
System.out.println();

printLane(lane3Horse);
System.out.print("  " + lane3Horse.getName() + " (Current confidence: " + lane3Horse.getConfidence() + ")");
System.out.println();
```



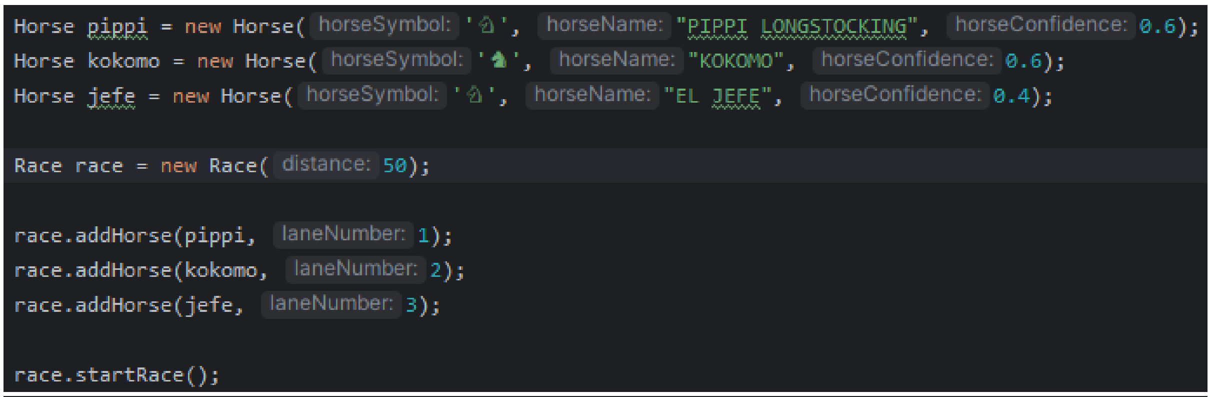
```
 ======================
|           ☠          |  PIPPI LONGSTOCKING (Current confidence: 60%)
|                 ♞    |  KOKOMO (Current confidence: 60%)
|        ☠             |  EL JEFE (Current confidence: 40%)
 ======================
 ======================
|           ☠          |  PIPPI LONGSTOCKING (Current confidence: 60%)
|                  ♞   |  KOKOMO (Current confidence: 60%)
|        ☠             |  EL JEFE (Current confidence: 40%)
 ======================
 ======================
|           ☠          |  PIPPI LONGSTOCKING (Current confidence: 60%)
|                   ♞  |  KOKOMO (Current confidence: 60%)
|        ☠             |  EL JEFE (Current confidence: 40%)
 ======================
 ======================
|           ☠          |  PIPPI LONGSTOCKING (Current confidence: 60%)
|                     ♞|  KOKOMO (Current confidence: 60%)
|        ☠             |  EL JEFE (Current confidence: 40%)
 ======================

And the winner is KOKOMO
```

Updating moveHorse() to be dependant on raceLength

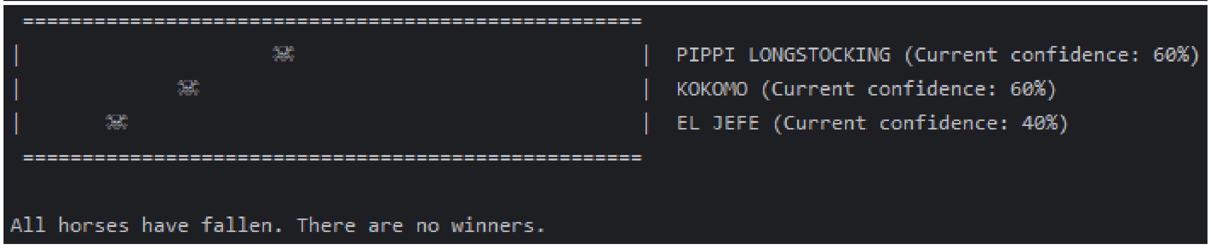
This issue rose when using a raceLength of 50 instead of 20, which were used for every test above until now (which none are affected by, but was an oversight), which caused many tests for the horses to not even reach anywhere close to the finish line.

Add to this as well, McFarewell's method of determining probability a horse falls is opposite to the way the intention of the horse's confidence is used for. For example, Pippi has a confidence of 0.4, and Kokomo has a confidence of 0.6. Pippi's upper bound for the random number to land on would be 0.016 (0.1x0.4x0.4) and Kokomo's would be 0.036 (0.1x0.6x0.6), meaning Kokomo would have a higher chance of falling which doesn't make sense if the idea of the confidence is sort of "the higher the better." Therefore this is also a concern.

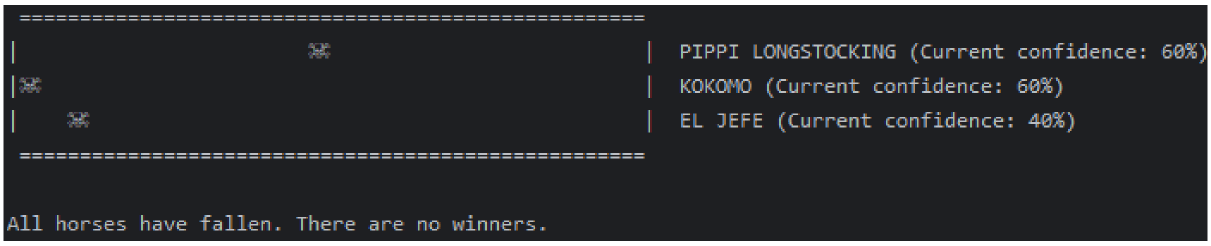
Initial Tests:



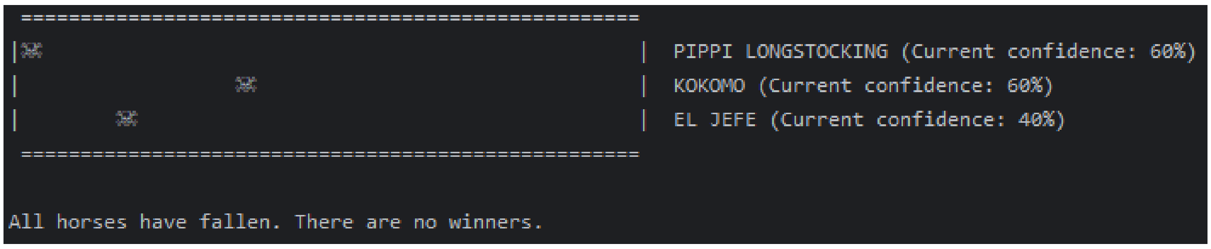
```
Horse pippi = new Horse( horseSymbol: '♘', horseName: "PIPPI LONGSTOCKING", horseConfidence: 0.6);
Horse kokomo = new Horse( horseSymbol: '♞', horseName: "KOKOMO", horseConfidence: 0.6);
Horse jefe = new Horse( horseSymbol: '♘', horseName: "EL JEFE", horseConfidence: 0.4);

Race race = new Race( distance: 50);

race.addHorse(pippi, laneNumber: 1);
race.addHorse(kokomo, laneNumber: 2);
race.addHorse(jefe, laneNumber: 3);

race.startRace();
```



```
 ====================================================
|                      ☠                             |  PIPPI LONGSTOCKING (Current confidence: 60%)
|              ☠                                     |  KOKOMO (Current confidence: 60%)
|        ☠                                           |  EL JEFE (Current confidence: 40%)
 ====================================================


All horses have fallen. There are no winners.
```



```
 ====================================================
|                         ☠                          |  PIPPI LONGSTOCKING (Current confidence: 60%)
|☠                                                   |  KOKOMO (Current confidence: 60%)
|    ☠                                               |  EL JEFE (Current confidence: 40%)
 ====================================================


All horses have fallen. There are no winners.
```



```
 ====================================================
|☠                                                   |  PIPPI LONGSTOCKING (Current confidence: 60%)
|                   ☠                                |  KOKOMO (Current confidence: 60%)
|         ☠                                          |  EL JEFE (Current confidence: 40%)
 ====================================================


All horses have fallen. There are no winners.
```

Theory:

Updating the method using a mathematical theory in this case could be unnecessary for a model as the one developed. So, I had to reinvent a newer and better method of determining wins and losses that complied with realistic measures and any track in question.

Through research, in a realistic horse race setting there was an 8% chance the horse may fall, noting this statistic is dependent on jumps racing but nevertheless it can act as a realistic goal.

I then developed this method:

Let: horseConfidence = the horse's confidence between [0,1]

raceLength = the length of the track in metres/yards

o = 1 − horseConfidence

z = 1/raceLength

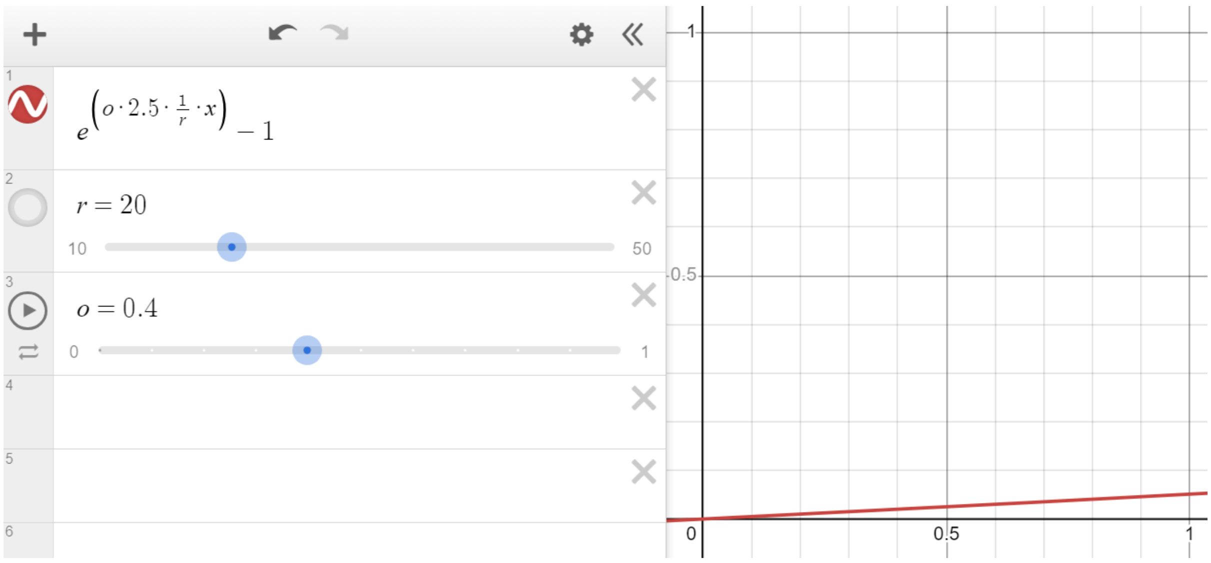
Equation:

$$y = e\hat{}(o \times z \times 2.5x) - 1$$

Examples

where: $z = 1/r$

Track length = 20:

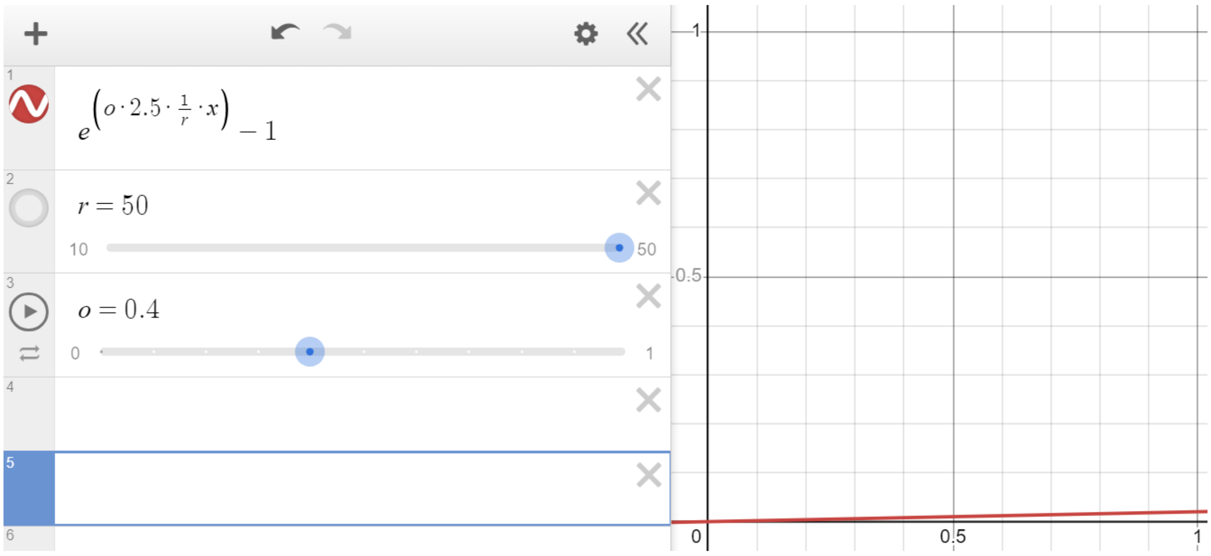


$$e^{\left(o\cdot 2.5\cdot\frac{1}{r}\cdot x\right)}-1$$

$r = 20$

$o = 0.4$

Track length = 50:



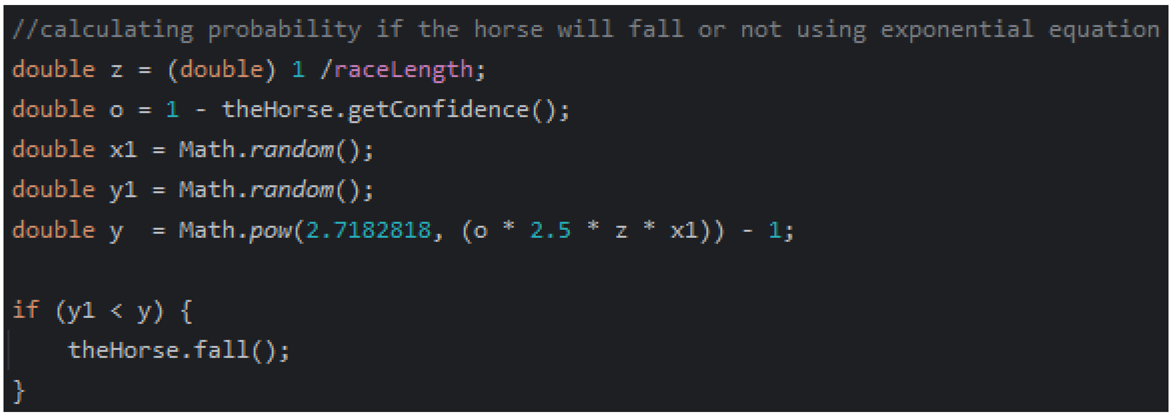
$$e^{\left(o \cdot 2.5 \cdot \frac{1}{r} \cdot x\right)} - 1$$

$r = 50$

$o = 0.4$

How is Randomness included?
Well, the program generates two different random numbers between [0,1), that act as a coordinate, let's say (x_1,y_1). x_1 is used to generate a target value of y that is then compared against y_1, and if it is less than y then the horse will fall.

Resulting code:



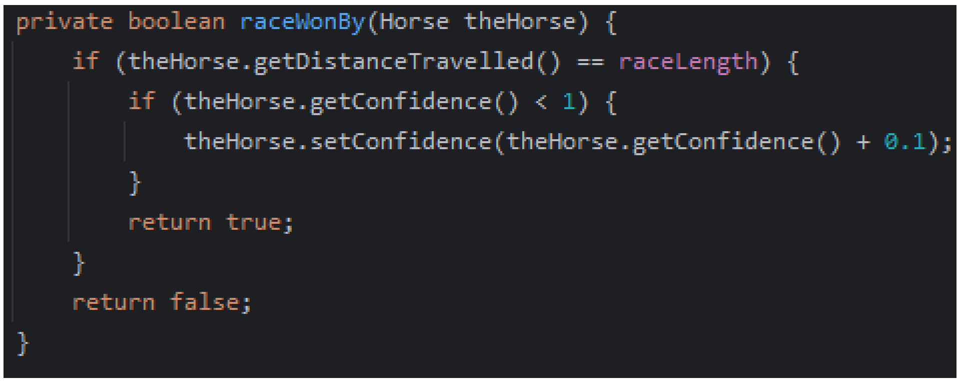
```
//calculating probability if the horse will fall or not using exponential equation
double z = (double) 1 /raceLength;
double o = 1 - theHorse.getConfidence();
double x1 = Math.random();
double y1 = Math.random();
double y  = Math.pow(2.7182818, (o * 2.5 * z * x1)) - 1;

if (y1 < y) {
    theHorse.fall();
}
```

Changing confidence values of horses depending on matches

Changing the confidence of a specific horse to decrease by 0.1 whenever it falls and increases by 0.1 every time it wins a race.

Increasing confidence level, in raceWonBy():

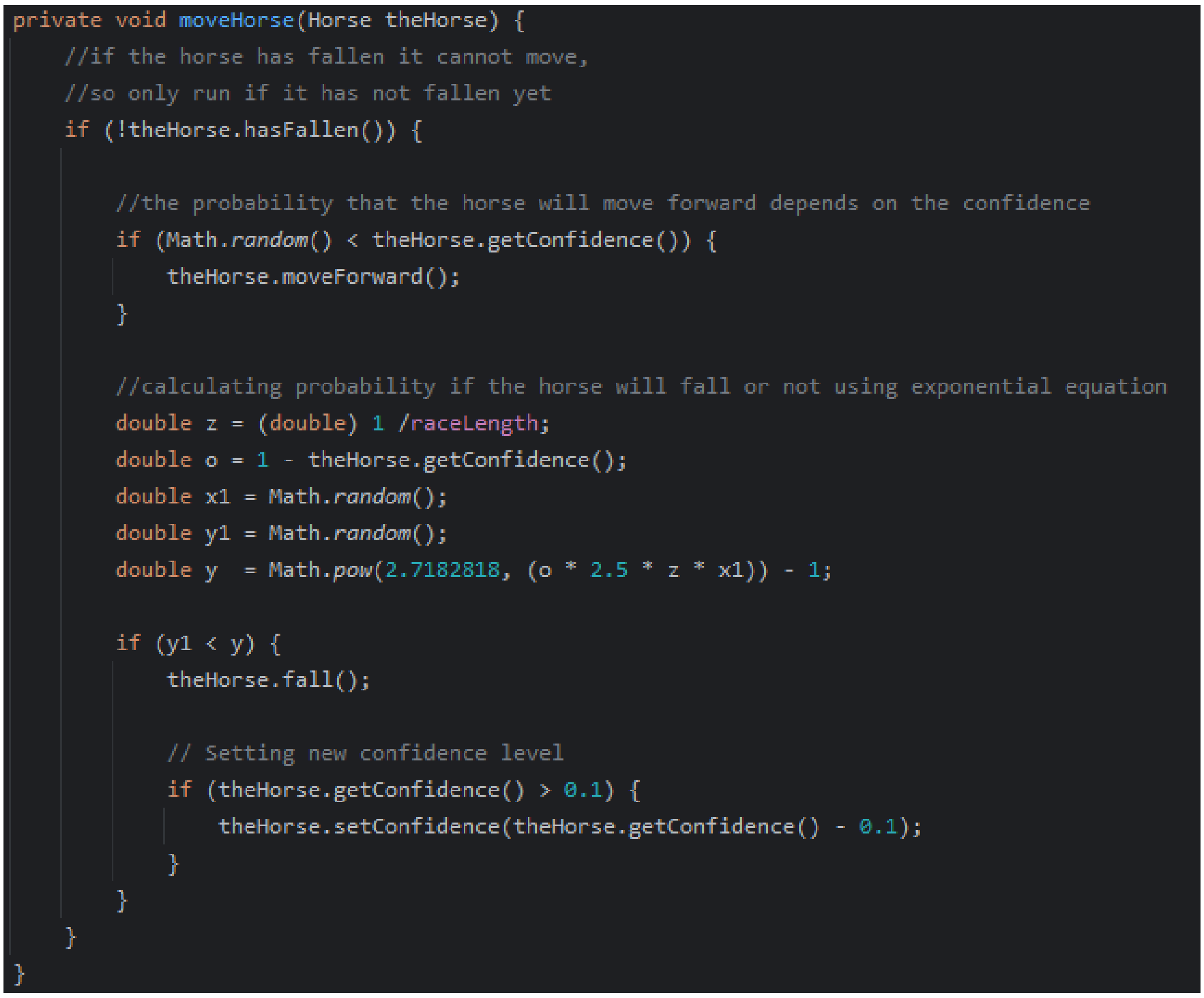
If the confidence level is less than 1, then increase by 0.1.



```java
private boolean raceWonBy(Horse theHorse) {
    if (theHorse.getDistanceTravelled() == raceLength) {
        if (theHorse.getConfidence() < 1) {
            theHorse.setConfidence(theHorse.getConfidence() + 0.1);
        }
        return true;
    }
    return false;
}
```

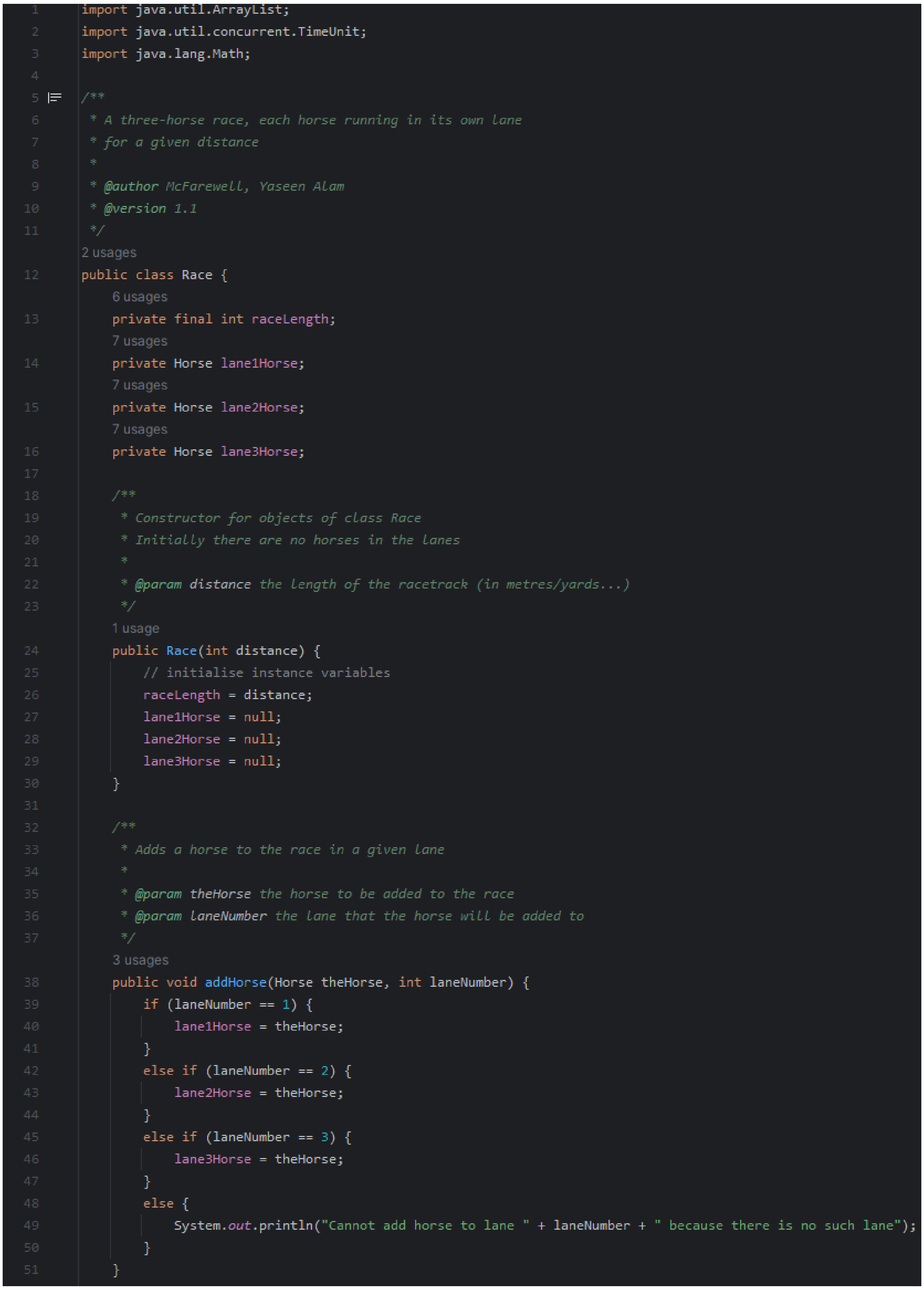
Decreasing confidence level, in moveHorse():

If the confidence level is greater than 0.1 then decrease it by 0.1.

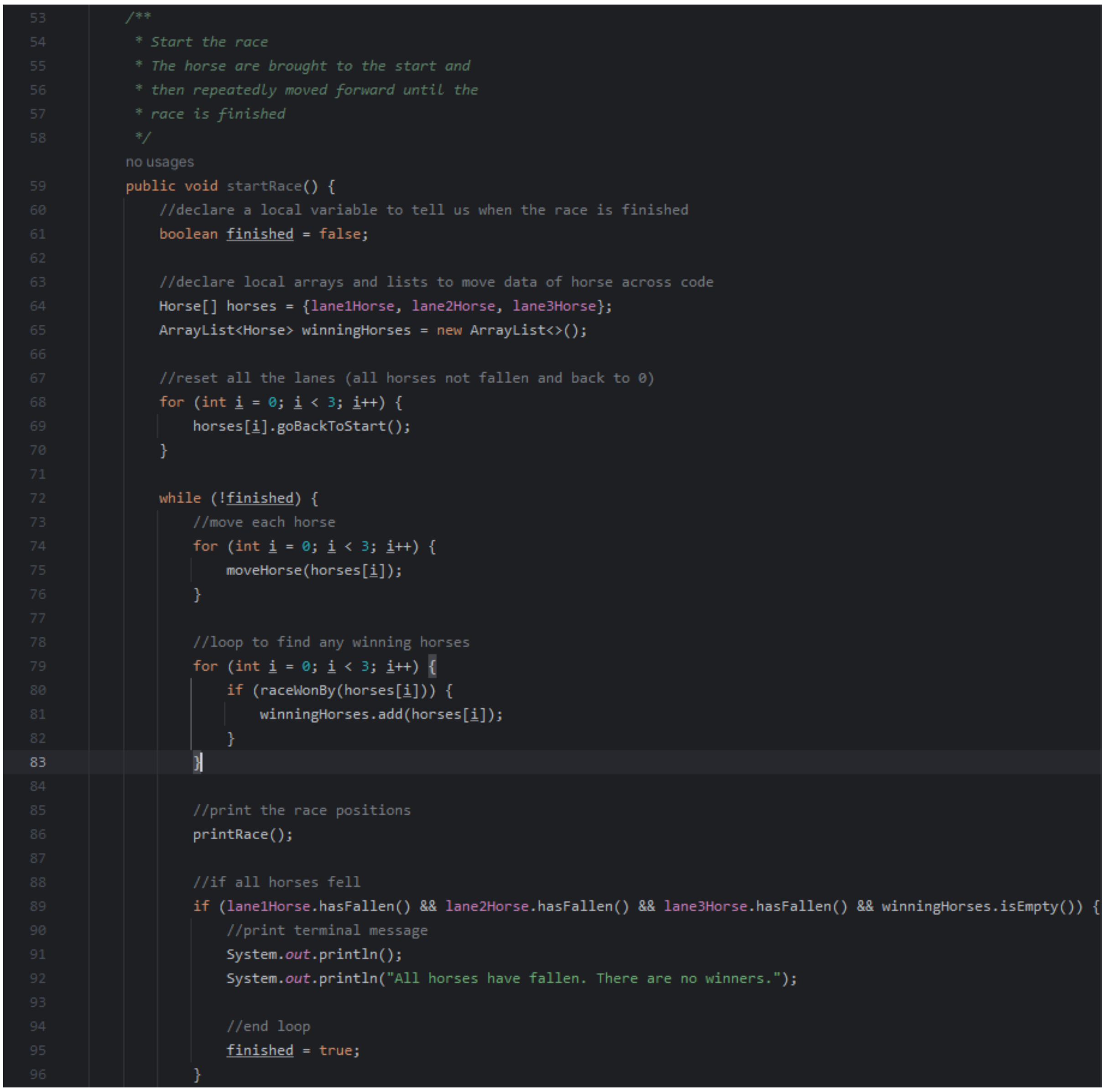


```java
private void moveHorse(Horse theHorse) {
    //if the horse has fallen it cannot move,
    //so only run if it has not fallen yet
    if (!theHorse.hasFallen()) {

        //the probability that the horse will move forward depends on the confidence
        if (Math.random() < theHorse.getConfidence()) {
            theHorse.moveForward();
        }

        //calculating probability if the horse will fall or not using exponential equation
        double z = (double) 1 /raceLength;
        double o = 1 - theHorse.getConfidence();
        double x1 = Math.random();
        double y1 = Math.random();
        double y  = Math.pow(2.7182818, (o * 2.5 * z * x1)) - 1;

        if (y1 < y) {
            theHorse.fall();

            // Setting new confidence level
            if (theHorse.getConfidence() > 0.1) {
                theHorse.setConfidence(theHorse.getConfidence() - 0.1);
            }
        }
    }
}
```

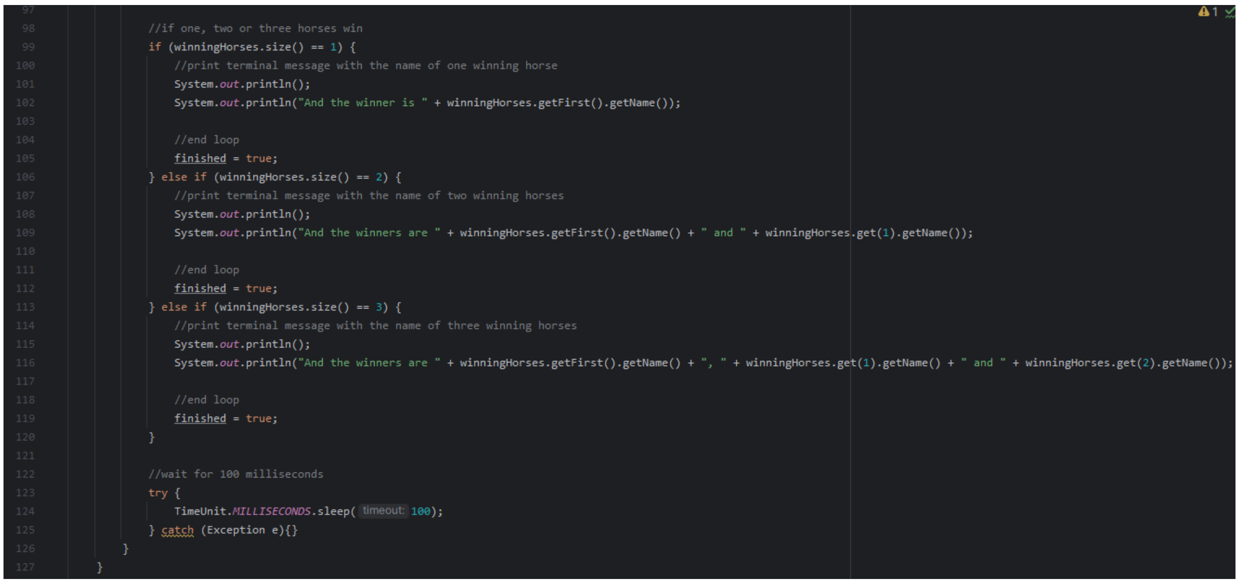
Race.java Code



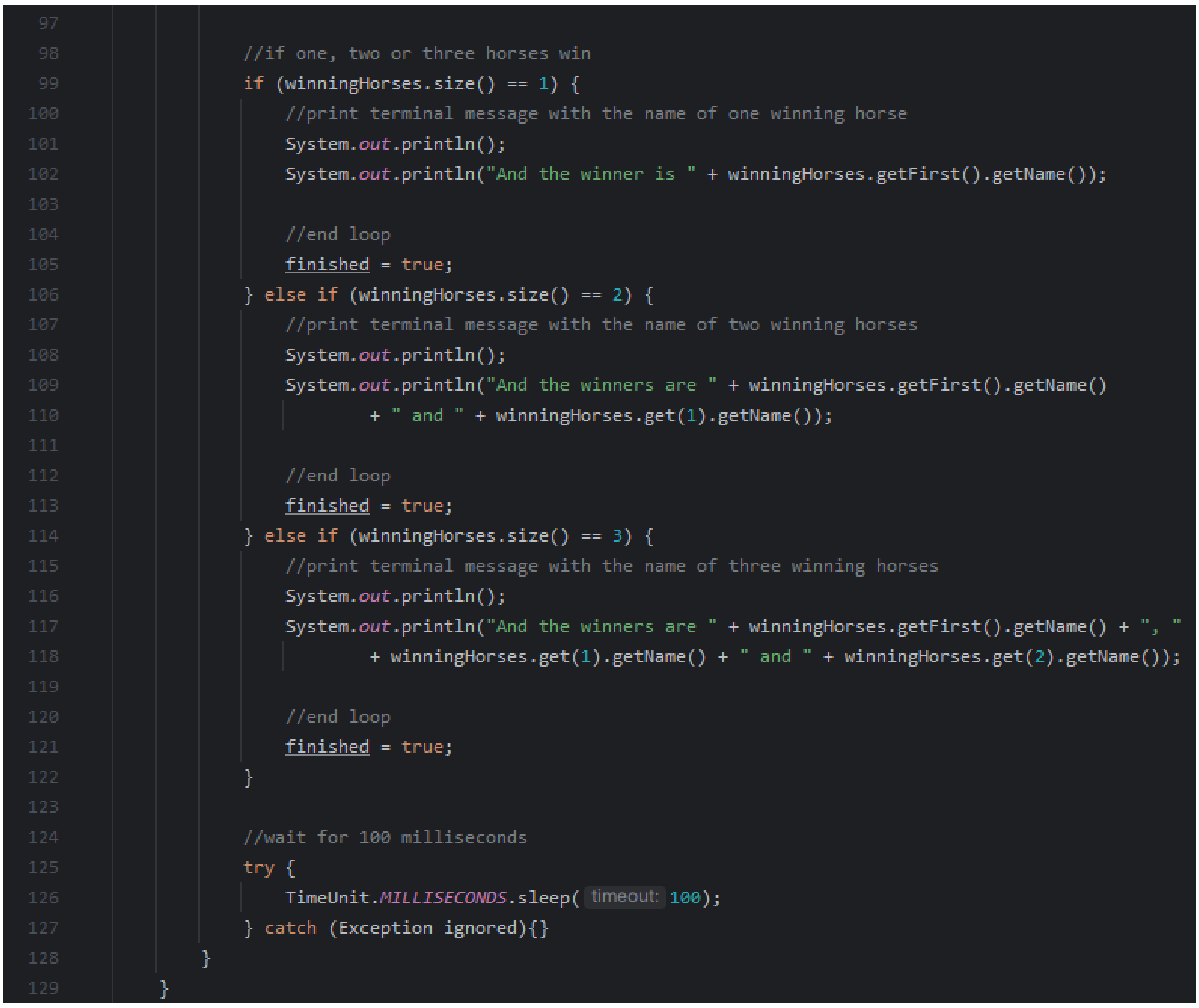
```java
import java.util.ArrayList;
import java.util.concurrent.TimeUnit;
import java.lang.Math;

/**
 * A three-horse race, each horse running in its own lane
 * for a given distance
 *
 * @author McFarewell, Yaseen Alam
 * @version 1.1
 */
2 usages
public class Race {
    6 usages
    private final int raceLength;
    7 usages
    private Horse lane1Horse;
    7 usages
    private Horse lane2Horse;
    7 usages
    private Horse lane3Horse;

    /**
     * Constructor for objects of class Race
     * Initially there are no horses in the lanes
     *
     * @param distance the length of the racetrack (in metres/yards...)
     */
    1 usage
    public Race(int distance) {
        // initialise instance variables
        raceLength = distance;
        lane1Horse = null;
        lane2Horse = null;
        lane3Horse = null;
    }

    /**
     * Adds a horse to the race in a given lane
     *
     * @param theHorse the horse to be added to the race
     * @param laneNumber the lane that the horse will be added to
     */
    3 usages
    public void addHorse(Horse theHorse, int laneNumber) {
        if (laneNumber == 1) {
            lane1Horse = theHorse;
        }
        else if (laneNumber == 2) {
            lane2Horse = theHorse;
        }
        else if (laneNumber == 3) {
            lane3Horse = theHorse;
        }
        else {
            System.out.println("Cannot add horse to lane " + laneNumber + " because there is no such lane");
        }
    }
```



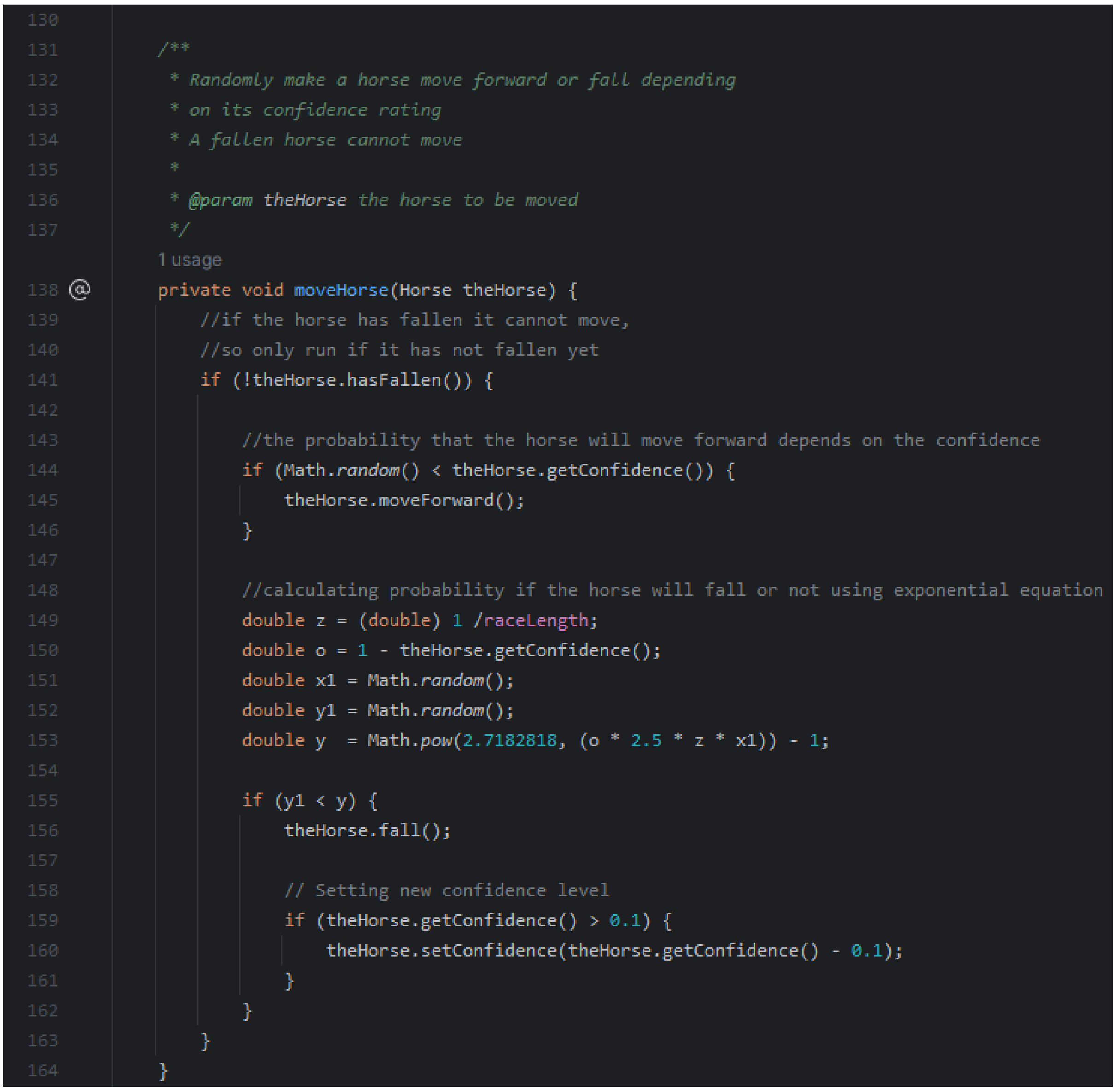
```java
    /**
     * Start the race
     * The horse are brought to the start and
     * then repeatedly moved forward until the
     * race is finished
     */
    no usages
    public void startRace() {
        //declare a local variable to tell us when the race is finished
        boolean finished = false;

        //declare local arrays and lists to move data of horse across code
        Horse[] horses = {lane1Horse, lane2Horse, lane3Horse};
        ArrayList<Horse> winningHorses = new ArrayList<>();

        //reset all the lanes (all horses not fallen and back to 0)
        for (int i = 0; i < 3; i++) {
            horses[i].goBackToStart();
        }

        while (!finished) {
            //move each horse
            for (int i = 0; i < 3; i++) {
                moveHorse(horses[i]);
            }

            //loop to find any winning horses
            for (int i = 0; i < 3; i++) {
                if (raceWonBy(horses[i])) {
                    winningHorses.add(horses[i]);
                }
            }

            //print the race positions
            printRace();

            //if all horses fell
            if (lane1Horse.hasFallen() && lane2Horse.hasFallen() && lane3Horse.hasFallen() && winningHorses.isEmpty()) {
                //print terminal message
                System.out.println();
                System.out.println("All horses have fallen. There are no winners.");

                //end loop
                finished = true;
            }
```



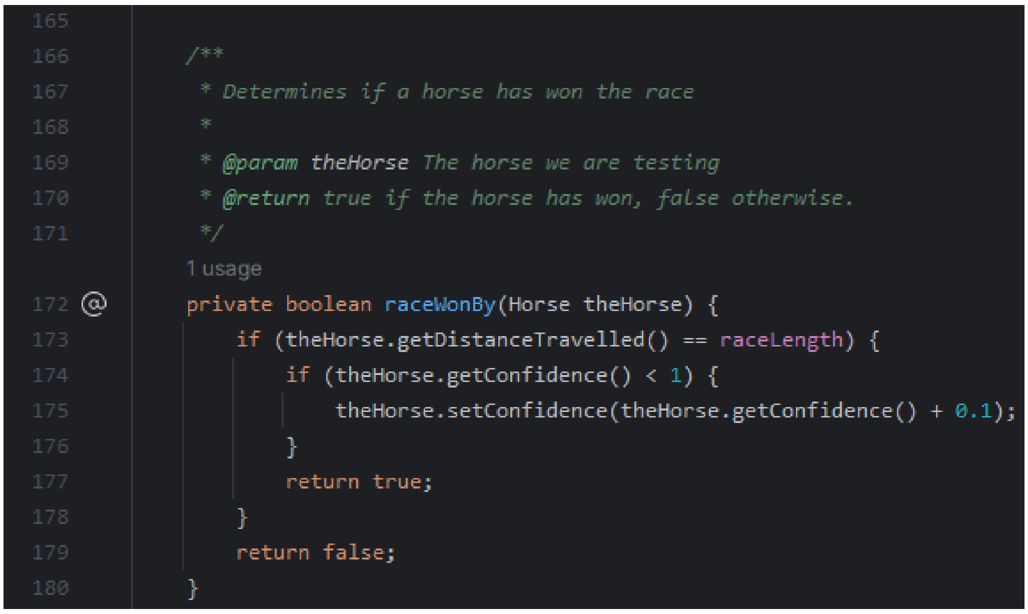
```java
            //if one, two or three horses win
            if (winningHorses.size() == 1) {
                //print terminal message with the name of one winning horse
                System.out.println();
                System.out.println("And the winner is " + winningHorses.getFirst().getName());

                //end loop
                finished = true;
            } else if (winningHorses.size() == 2) {
                //print terminal message with the name of two winning horses
                System.out.println();
                System.out.println("And the winners are " + winningHorses.getFirst().getName() + " and " + winningHorses.get(1).getName());

                //end loop
                finished = true;
            } else if (winningHorses.size() == 3) {
                //print terminal message with the name of three winning horses
                System.out.println();
                System.out.println("And the winners are " + winningHorses.getFirst().getName() + ", " + winningHorses.get(1).getName() + " and " + winningHorses.get(2).getName());

                //end loop
                finished = true;
            }

            //wait for 100 milliseconds
            try {
                TimeUnit.MILLISECONDS.sleep( timeout: 100);
            } catch (Exception e){}
        }
    }
```



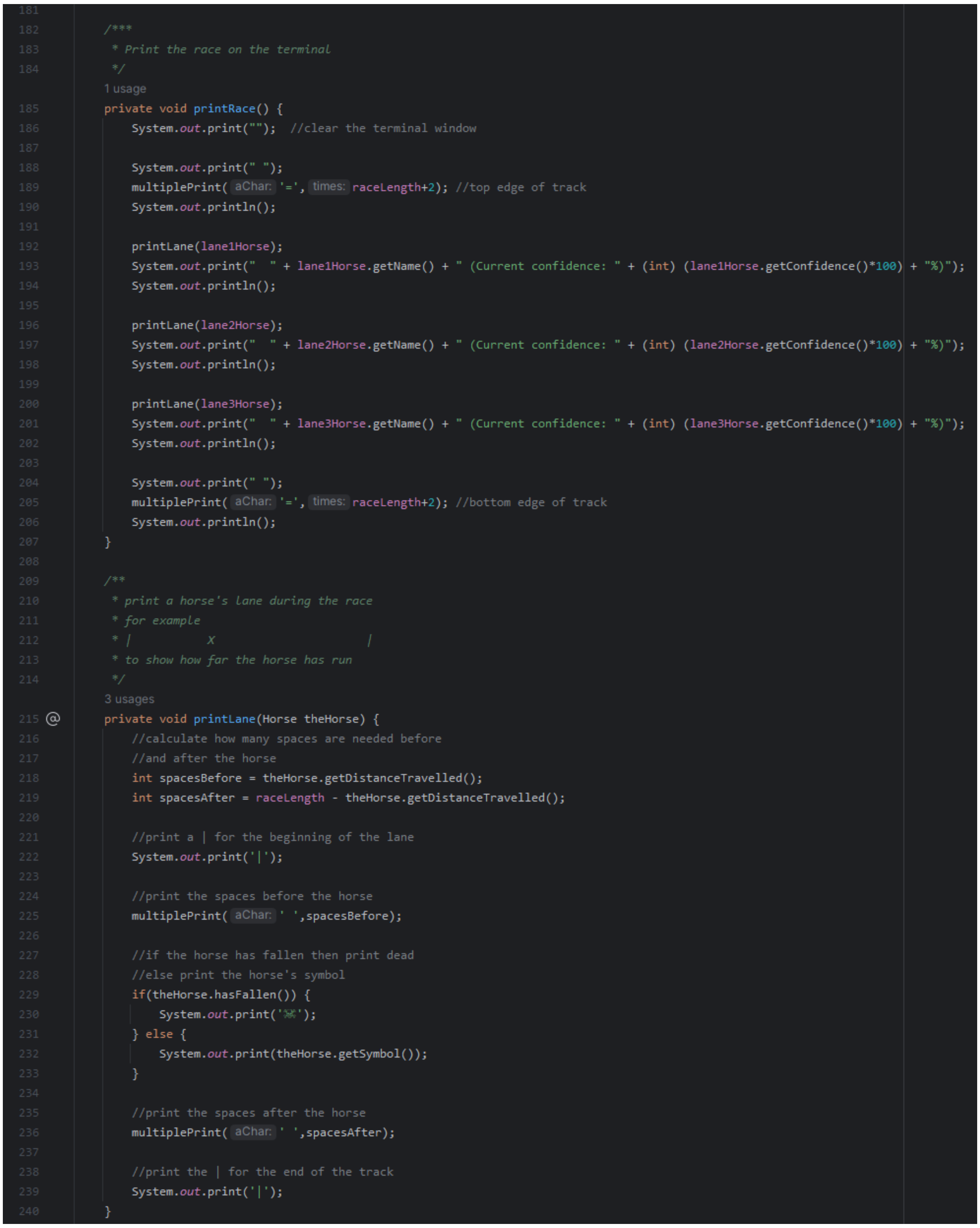
```java

            //if one, two or three horses win
            if (winningHorses.size() == 1) {
                //print terminal message with the name of one winning horse
                System.out.println();
                System.out.println("And the winner is " + winningHorses.getFirst().getName());

                //end loop
                finished = true;
            } else if (winningHorses.size() == 2) {
                //print terminal message with the name of two winning horses
                System.out.println();
                System.out.println("And the winners are " + winningHorses.getFirst().getName()
                        + " and " + winningHorses.get(1).getName());

                //end loop
                finished = true;
            } else if (winningHorses.size() == 3) {
                //print terminal message with the name of three winning horses
                System.out.println();
                System.out.println("And the winners are " + winningHorses.getFirst().getName() + ", "
                        + winningHorses.get(1).getName() + " and " + winningHorses.get(2).getName());

                //end loop
                finished = true;
            }

            //wait for 100 milliseconds
            try {
                TimeUnit.MILLISECONDS.sleep( timeout: 100);
            } catch (Exception ignored){}
        }
    }
```



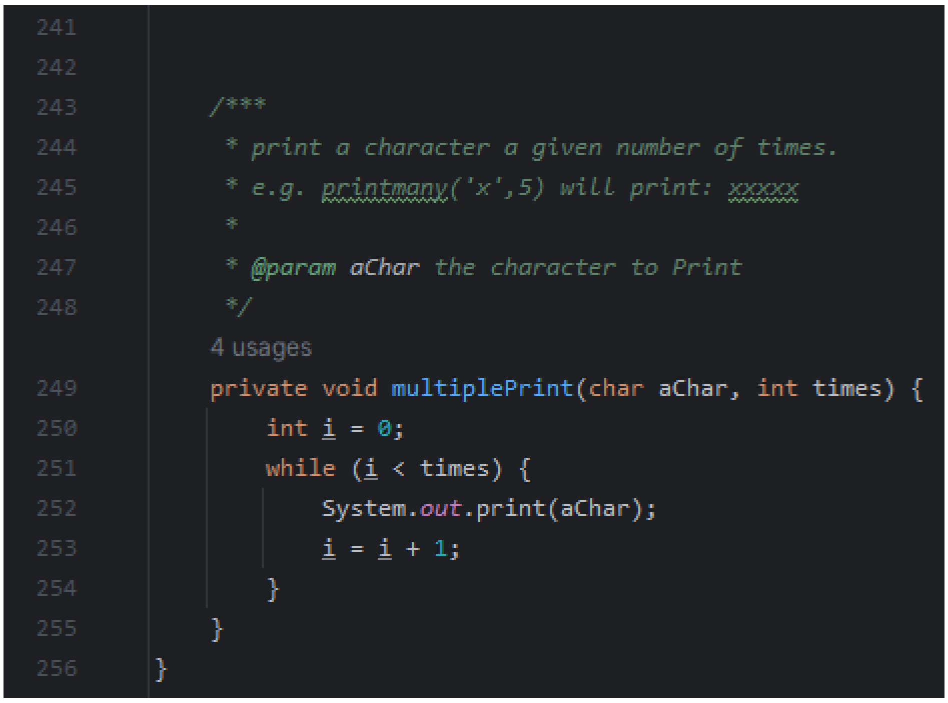
```java

    /**
     * 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
     */
    1 usage
    private void moveHorse(Horse theHorse) {
        //if the horse has fallen it cannot move,
        //so only run if it has not fallen yet
        if (!theHorse.hasFallen()) {

            //the probability that the horse will move forward depends on the confidence
            if (Math.random() < theHorse.getConfidence()) {
                theHorse.moveForward();
            }

            //calculating probability if the horse will fall or not using exponential equation
            double z = (double) 1 /raceLength;
            double o = 1 - theHorse.getConfidence();
            double x1 = Math.random();
            double y1 = Math.random();
            double y  = Math.pow(2.7182818, (o * 2.5 * z * x1)) - 1;

            if (y1 < y) {
                theHorse.fall();

                // Setting new confidence level
                if (theHorse.getConfidence() > 0.1) {
                    theHorse.setConfidence(theHorse.getConfidence() - 0.1);
                }
            }
        }
    }
```



```java

    /**
     * Determines if a horse has won the race
     *
     * @param theHorse The horse we are testing
     * @return true if the horse has won, false otherwise.
     */
    1 usage
    private boolean raceWonBy(Horse theHorse) {
        if (theHorse.getDistanceTravelled() == raceLength) {
            if (theHorse.getConfidence() < 1) {
                theHorse.setConfidence(theHorse.getConfidence() + 0.1);
            }
            return true;
        }
        return false;
    }
```



```java

    /***
     * Print the race on the terminal
     */
    1 usage
    private void printRace() {
        System.out.print("");  //clear the terminal window

        System.out.print(" ");
        multiplePrint( aChar: '=', times: raceLength+2); //top edge of track
        System.out.println();

        printLane(lane1Horse);
        System.out.print("  " + lane1Horse.getName() + " (Current confidence: " + (int) (lane1Horse.getConfidence()*100) + "%)");
        System.out.println();

        printLane(lane2Horse);
        System.out.print("  " + lane2Horse.getName() + " (Current confidence: " + (int) (lane2Horse.getConfidence()*100) + "%)");
        System.out.println();

        printLane(lane3Horse);
        System.out.print("  " + lane3Horse.getName() + " (Current confidence: " + (int) (lane3Horse.getConfidence()*100) + "%)");
        System.out.println();

        System.out.print(" ");
        multiplePrint( aChar: '=', times: raceLength+2); //bottom edge of track
        System.out.println();
    }

    /**
     * print a horse's lane during the race
     * for example
     * |           X                      |
     * to show how far the horse has run
     */
    3 usages
    private void printLane(Horse theHorse) {
        //calculate how many spaces are needed before
        //and after the horse
        int spacesBefore = theHorse.getDistanceTravelled();
        int spacesAfter = raceLength - theHorse.getDistanceTravelled();

        //print a | for the beginning of the lane
        System.out.print('|');

        //print the spaces before the horse
        multiplePrint( aChar: ' ',spacesBefore);

        //if the horse has fallen then print dead
        //else print the horse's symbol
        if(theHorse.hasFallen()) {
            System.out.print('☠');
        } else {
            System.out.print(theHorse.getSymbol());
        }

        //print the spaces after the horse
        multiplePrint( aChar: ' ',spacesAfter);

        //print the | for the end of the track
        System.out.print('|');
    }
```



```java


    /***
     * print a character a given number of times.
     * e.g. printmany('x',5) will print: xxxxx
     *
     * @param aChar the character to Print
     */
    4 usages
    private void multiplePrint(char aChar, int times) {
        int i = 0;
        while (i < times) {
            System.out.print(aChar);
            i = i + 1;
        }
    }
}
```