Horse Class Report:

Encapsulation in object oriented programming ensures that the data within the class is hidden from external classes and can only be accessed through accessor methods.In the Horse class, encapsulation is utilised by making the attributes private and providing public methods to access and modify them.

Accessor Methods:

- getConfidence(): This method retrieves the confidence level of the horse.
- getDistanceTravelled(): Retrieves the distance travelled by the horse.
- getName(): Retrieves the name of the horse.
- getSymbol(): Retrieves the symbol representing the horse.

Mutator Methods:

- fall(): Modifies the fallen field to indicate that the horse has fallen.
- goBackToStart(): Resets the distance travelled by the horse and sets the fallen state to false, effectively resetting the horse's position.
- moveForward(): Modifies the distanceTravelled field to simulate the horse's movement forward. It checks if the horse has fallen before incrementing the distance travelled.
- setConfidence(double newConfidence): Modifies the confidence level of the horse if the input is within a valid range (0 to 1).
- setSymbol(char newSymbol): Modifies the symbol representing the horse.

Testing Process:

```
Horse myHorse = new Horse( horseSymbol: 'H',  horseName: "Thunder",  horseConfidence: 0.8);
```

I started off by creating a new Horse object named myHorse, passing in values for its symbol ('H'), name ("Thunder"), and confidence level (0.8).

```
System.out.println(myHorse.getName());
System.out.println(myHorse.getSymbol());
System.out.println(myHorse.getConfidence());
System.out.println(myHorse.getDistanceTravelled());
```

These lines verify that the accessor methods (getName(), getSymbol(), getConfidence(), getDistanceTravelled()) return the expected values for the myHorse object. Over here I check that the name is "Thunder", symbol is 'H', confidence level is 0.8, and initial distance travelled is 0.

```
myHorse.moveForward();
System.out.println(myHorse.getDistanceTravelled());

myHorse.setConfidence(0.9);
System.out.println(myHorse.getConfidence());

myHorse.setSymbol('T');
System.out.println(myHorse.getSymbol());
```

moveForward(): Checks if the horse's distance travelled increases by 1.

setConfidence(): Tests if the horse's confidence level is successfully updated to 0.9.

setSymbol(): Checks if the horse's symbol is correctly updated to 'T'.

Output:

```
Horses name: Thunder
Symbol: H
Confidence Level: 0.8
Distance Travelled: 0

New distance: 1
Updated Confidence level: 0.9
New symbol: T
```

Race Class Report:

Problem 1:

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

If raceWonBy() always returns false for all horses, the loop could become infinite. This means that even if all the horses have fallen and therefore lost the race, the game would still continue regardless which is not what we want.

Solution:

```
//print the race positions
printRace();
if ( raceWonBy(lane1Horse) || raceWonBy(lane2Horse) || raceWonBy(lane3Horse) ||
        (lane1Horse.hasFallen() && lane2Horse.hasFallen() && lane3Horse.hasFallen()))
{
    finished = true;
}
```

In addition to checking if any horses have won, I have added an additional condition which also checks if all three horses have fallen. If this condition is met, the game will terminate immediately instead of continuing.

Problem 2:

When the code is executed, the confidence level of the winning horse remains the same but we want to slightly increase its confidence level once it has won the game.

Solution:

```
//Increase confidence level by 0.1, if horse wins the race.
if (raceWonBy(theHorse) && theHorse.getConfidence() <= 0.9) {
    theHorse.setConfidence(theHorse.getConfidence() + 0.1);
}
```

In order to achieve this I have added an extra if statement within the moveHorse method which checks if the confidence level is less than or equal to 0.9. This ensures the confidence level does not reach beyond 1.0.I then added a function which increases the fallen horse's current confidence level by 0.1.

Problem 3:

```
if (Math.random() < (0.1*theHorse.getConfidence()*theHorse.getConfidence()))
{
    theHorse.fall();
}
```

If a horse falls during the race, its confidence level remains the same but we want to slightly decrease the fallen horse's confidence.

Solution:

```
if (Math.random() < (0.1 * theHorse.getConfidence() * theHorse.getConfidence())) {
    if (theHorse.getConfidence() >= 0.1) {
        theHorse.setConfidence(theHorse.getConfidence() - 0.1);
    }
    theHorse.fall();
}
```

In order to achieve this I have added an extra if statement within the moveHorse method which checks if the confidence level is at least greater than or equal to 0.1. This ensures the confidence level stays within the range of 0 to 1. I then called the function setConfidence and decreased the fallen horse's current confidence level by 0.1.

Problem 4:

```
private boolean raceWonBy(Horse theHorse)
{
    if (theHorse.getDistanceTravelled() == raceLength)
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

Currently the winner of the race is determined by whether the horse has travelled the required length (racelength). This means that even if any of the horses have fallen, if it reaches the race length it still wins the game. This is not what we want because once the horse has fallen, it should lose the game.

Solution:

```
private boolean raceWonBy(Horse theHorse) {
    return !theHorse.hasFallen() && theHorse.getDistanceTravelled() == raceLength;
```

Over here I added an additional boolean condition which checks whether the horse reached the race length as well as has not fallen during the race. This ensures that the horses that win the game are not fallen horses.

Problem 5:

During the execution of the code, on each round there is no display of the horse's confidence level or the horse's name next to its symbol. This means the users are not able to see the status of the horse's confidence level at each stage of the game.

Solution:

```
//print the horse's confidence level
System.out.print(" " + theHorse.getName() +
        " (Current Confidence level " + (Math.round(theHorse.getConfidence() * 100.0) / 100.0) + ")");
```

In order to fix this I have added this line of code at the end of the printLane method that is responsible for printing information about a horse's name and its current confidence level. It concatenates the horse's name with a space and then appends the phrase " (Current Confidence level " followed by the horse's confidence level (rounded to two decimal places) expressed as a percentage, and finally adds a closing parenthesis. Overall, it creates a formatted string that displays both the horse's name and its confidence level.

Problem 6:
Currently there is no method which is responsible for determining the winner of the race based on the positions of the horses in their respective lanes. This means once the race ends, no winner is printed out at the end.

Solution:

```
private String determineWinner() {
    if (raceWonBy(lane1Horse)) {
        return lane1Horse.getName();
    } else if (raceWonBy(lane2Horse)) {
        return lane2Horse.getName();
    } else if (raceWonBy(lane3Horse)) {
        return lane3Horse.getName();
    }
    return "none of them";
}
```

To fix this, I have implemented an extra method called determineWinner. This method first checks if any of the horses have won with horses as arguments. The raceWonBy() method determines if the horse has won by checking if its distance travelled equals the race length and it hasn't fallen. If that particular horse has won, the method returns the name of the horse.

I then inserted this line of code within the startRace method in order to print out the winner at the end of the race and help the user identify which horse won the race.

```
System.out.println("And the winner is " + determineWinner());
```

```
/**
 * 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
 * @throws IllegalArgumentException if the lane number is invalid
 */
3 usages   Maimuna Nowaz *
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 {
        throw new IllegalArgumentException("Invalid lane number: " + laneNumber);
    }
}
```

To improve the addHorse method, Ive implemented error handling at the else statement instead of the original print statement. The purpose of error handling in the method for adding horses to lanes is to enhance the robustness and reliability of the code. By incorporating error handling, the method can gracefully handle invalid inputs, such as an incorrect lane number, and provide informative feedback to the user or caller of the method.

Evidence of testing:

```java
// Test case 1: Adding horses to all lanes
Race race1 = new Race( distance: 10);
Horse horse1 = new Horse( horseSymbol: 'T', horseName: "Thunder", horseConfidence: 0.8);
Horse horse2 = new Horse( horseSymbol: 'B', horseName: "Blaze", horseConfidence: 0.7);
Horse horse3 = new Horse( horseSymbol: 'S', horseName: "Swift", horseConfidence: 0.6);
race1.addHorse(horse1, laneNumber: 1);
race1.addHorse(horse2, laneNumber: 2);
race1.addHorse(horse3, laneNumber: 3);
race1.startRace();
System.out.println();

// Test case 2: Adding horse to a non-existent lane
Race race2 = new Race( distance: 10);
Horse horse4 = new Horse( horseSymbol: 'F', horseName: "Flash", horseConfidence: 0.9);
race2.addHorse(horse4, laneNumber: 1);
race2.addHorse(horse2, laneNumber: 4);
race2.startRace();
System.out.println();


// Test case 3: Starting race without adding any horses
Race race3 = new Race( distance: 10);
  race3.startRace(); // Should print an error message
```

I have included tests for different scenarios in my race.java file. My first test case works perfectly fine with all their horses assigned to each of their lanes. Also confidence level increases and decreases when expected for each of the horses.

My second test case gave me an error as expected when I tried to assign a horse to a lane number which did not exist.

For my third test case I attempted to run the race without adding any horses to the lanes. This gave me a null exception error as expected because the program detected that the lane horse object is not initialised.

Second test case:

```
Exception in thread "main" java.lang.IllegalArgumentException Create breakpoint : Invalid lane number: 4
    at Part1.Race.addHorse(Race.java:48)
    at Part1.RaceTest.main(Race.java:250)
```

<u>Output of my code after modification to race.java class:</u>

```
=============
| T         | Thunder (Current Confidence level 0.8)
| B         | Blaze (Current Confidence level 0.7)
| S         | Swift (Current Confidence level 0.6)
=============
=============
|  T        | Thunder (Current Confidence level 0.8)
| B         | Blaze (Current Confidence level 0.7)
| S         | Swift (Current Confidence level 0.6)
=============
=============
|   T       | Thunder (Current Confidence level 0.8)
|  B        | Blaze (Current Confidence level 0.7)
| S         | Swift (Current Confidence level 0.6)
=============
=============
|   ⌒       | Thunder (Current Confidence level 0.7)
|   B       | Blaze (Current Confidence level 0.7)
|  S        | Swift (Current Confidence level 0.6)
=============
=============
|   ⌒       | Thunder (Current Confidence level 0.7)
|    ⌒      | Blaze (Current Confidence level 0.6)
|  S        | Swift (Current Confidence level 0.6)
=============
```

```
|       S   | Swift (Current Confidence level 0.6)
=============
=============
|   ⌒       | Thunder (Current Confidence level 0.7)
|    ⌒      | Blaze (Current Confidence level 0.6)
|        S  | Swift (Current Confidence level 0.6)
=============
=============
|   ⌒       | Thunder (Current Confidence level 0.7)
|    ⌒      | Blaze (Current Confidence level 0.6)
|        S  | Swift (Current Confidence level 0.6)
=============
=============
|   ⌒       | Thunder (Current Confidence level 0.7)
|    ⌒      | Blaze (Current Confidence level 0.6)
|        S  | Swift (Current Confidence level 0.6)
=============
=============
|   ⌒       | Thunder (Current Confidence level 0.7)
|    ⌒      | Blaze (Current Confidence level 0.6)
|         S | Swift (Current Confidence level 0.6)
=============
=============
|   ⌒       | Thunder (Current Confidence level 0.7)
|    ⌒      | Blaze (Current Confidence level 0.6)
|          S| Swift (Current Confidence level 0.7)
=============
And the winner is Swift
```