Illia Dutyi - 240871428

Part I — Horse Race Simulator Report

In my implementation of the Horse class, the horse's internal state was safeguarded as well as controlled via encapsulation principles that I applied.

Each and every one of the fields of the Horse class has been declared as being private.

• name: (String) is the name of each horse.

The horse is represented via a visual symbol (char). That particular symbol is indeed the very symbol.

• distanceTravelled: that racehorse moves a certain distance (int).

A boolean flag shows whether the horse has fallen. It is called fallen.

• confidence: A double from 0 to 1 representing how confident the horse is.

The fields cannot be accessed from outside the class or directly modified by making them private. Instead, I provide getter (accessor) and setter (mutator) methods in order to control the way that these fields are accessed and changed.

Access into the horse's private attributes, allowing for no direct modification, is granted by way of an accessor (getter).

Setter (Mutator) | The internal attributes belonging to the horse are modified safely, with validation where it is appropriate and necessary.

Notably, the setConfidence(double newConfidence) method ensures that the horse's confidence stays within bounds. The permissible bounds are usually from 0 up to 1. The method amends it automatically to the closest boundary that is valid if one provides an invalid value.

Specific methods encapsulate the horse's movement (moveForward()), fall (fall()), and reset (goBackToStart()), which prevents external classes modifying critical internal states directly and accidentally.

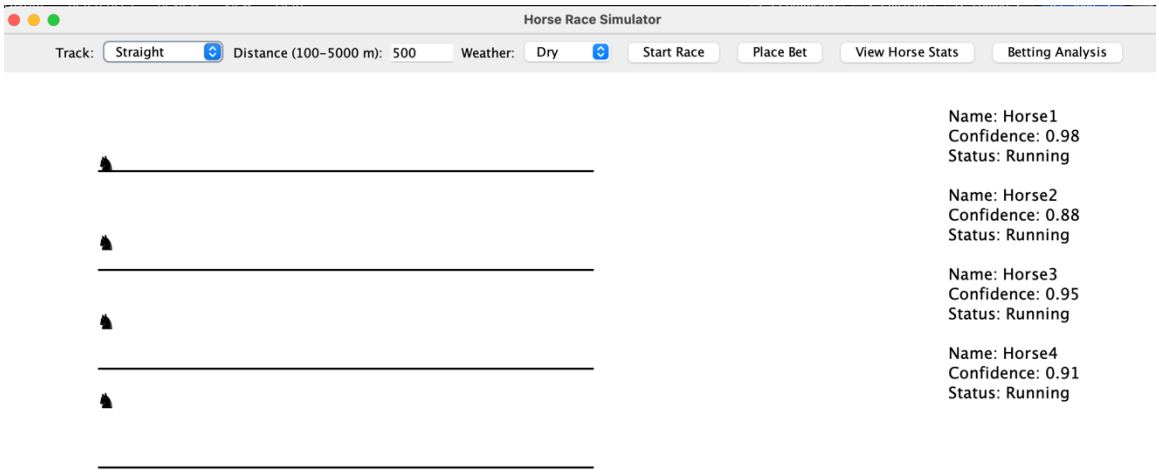
To conclude, encapsulation in my Horse class makes certain data is kept protected, logical rules are enforced within, and data's external misuse or corruption is prevented.

I did create as well as execute the RaceTest.java program so as to verify the Horse class's correctness, and also how it integrates with the Race class.

A few of the key tests that I performed were the following.

Test 1: (moveForward()) Horse's Movement Capability

The horses' symbol which moved from the left to the right indicated that their movement across the race track was then observed.

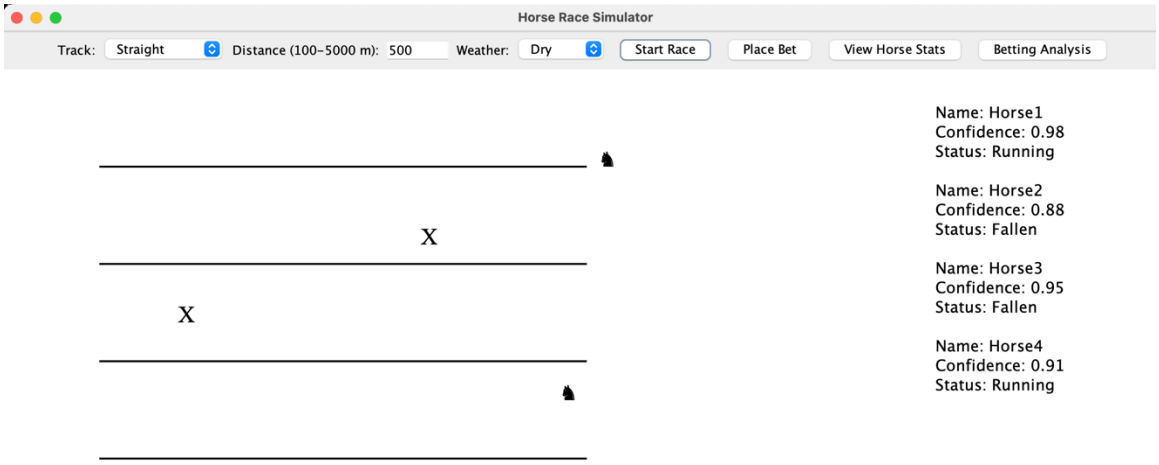


This confirmed for certain that the moveForward() method correctly and surely increases by one the distanceTravelled when the horse moves at all.

Test 2: The Falling (fall()) Horse is falling.

A certain 'X' was displayed by a number of horses, in place of the usual symbol of theirs during the race, indicating that they had indeed fallen.

This verification showed just how the fall() method does update the fallen flag to be true, and just how the visual output correctly switches the horse's symbol to be 'X' when fallen.

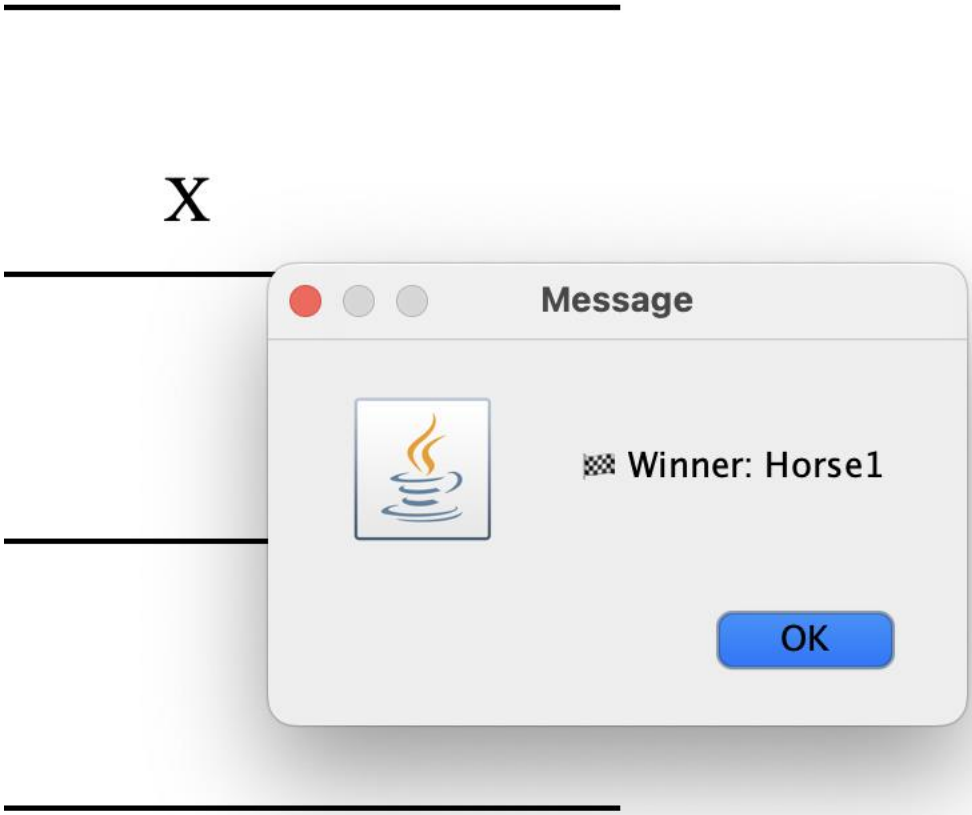


Test 3 examines Confidence Bounds by means of setConfidence().

The confidence was clamped either to 0 or to 1, as I observed after I manually tested setting the horse's confidence to invalid values, such as -1 and 2.

The setConfidence() method does indeed ensure that confidence remains within the required kind of range. This provides protection for the horse's attributes as a result of that.

Test 4 is with regard to race completion. The winner announcement will follow on.



Description: When that race had concluded, they then displayed the winner's name in a correct way based on the horse which reached the finish line in the first instance.

This confirmed the raceWonBy() logic and the printWinners() method worked correctly.

The tests do show that both the Horse and Race classes work in a correct manner, according to each of the Part I specifications.

Data integrity has indeed been successfully protected by means of encapsulation, and all of the methods function as they are intended to.

Race Class Report

Issue 1: No winning horse announcement immediately after the race

Problem: In the current startRace() method, the winners are only announced after asking if the user wants to play again.
Impact: The user has to wait for input before seeing who won, which is confusing.
Solution: Announce the winner immediately after the race finishes, before asking the player to play again.

Issue 2: System.out.print('\u000C') in printRace()

Problem: The code tries to clear the console by printing Unicode character \u000C. However, this does not work on most modern IDEs or command lines.
Impact: Makes the output messy or confusing — previous race frames clutter the display.
Solution: You can either remove this line and just print the new race state normally, or print multiple blank lines manually to simulate a clear screen effect.

Issue 3: Poor Confidence Formatting

Problem: When displaying horse confidence, you show it as a long decimal (e.g., 0.7343423).
Impact: It looks messy and unprofessional in the race output.
Solution: Format confidence to two decimal places using String.format("%.2f", value).