Encapsulation is the process of restricting/ protecting attributes, to reinforce their scope. This is mainly done via scope keywords ‘public’ (accessed from inside and outside class), ‘protected’ (accessed only from the class, and sub-classes), ‘private’(accessed only from the class). The compiler ensures these scopes directly, preventing compilation if the scope is not respected.

Accessor and mutator methods allow for retrieving and modifying the data from outside the class, ensuring there is no accidental change to the data. Mutators can also be used to validate the new value of an attribute e.g. in my code the setConfidence() method will only change the attribute if the new value is between 0.0 and 1.0(inclusive).

For Horse:

* Accessor methods: getConfidence(); getDistanceTravelled(); getName(); getSymbol()
* Mutator methods: setConfidence(); setSymbol()

A screenshot of a computer

AI-generated content may be incorrect.

A screen shot of a computer code

AI-generated content may be incorrect.

(trying to access a private attribute directly leading to error)

A screen shot of a computer program

AI-generated content may be incorrect.

(using an accessor to get the value of a private attribute)

A screen shot of a computer program

AI-generated content may be incorrect.

(Using a mutator to change the value of an attribute of an object)

A computer screen shot of text

AI-generated content may be incorrect.

(Passing an invalid input to a mutator, validation of the mutator prevents incorrect change)

A screen shot of a computer program

AI-generated content may be incorrect.

(Testing with boundary confidence value 0.0- denied)

A screen shot of a computer program

AI-generated content may be incorrect.

(Passing a valid value to a mutator)

A screen shot of a computer program

AI-generated content may be incorrect.

(Testing if the value of fallen attribute changes correctly after falling via fall(), and resetting after a fall via goBackToStart())

A screen shot of a computer program

AI-generated content may be incorrect.

(Testing horse movement, via moveForward() and movement reset via goBackToStart() )

Issues:

* Race moves too fast, playing out and ending faster than the video.
  + Increase the timeout parameter in TimeUnit.MILLISECONDS.sleep().

A screen shot of a computer

AI-generated content may be incorrect.

* + Compiler now waits longer between frames/states of the race providing a smoother progression, closer to the video.
* Current code does not display the horse’s name and current confidence, adjacent to their lane.
  + Added line of code at the end of the printLane() method to print these attributes of the passed in horse object when the lane is done printingA screen shot of a computer program

    AI-generated content may be incorrect.

A blue screen with white text

AI-generated content may be incorrect.

* Code does not account for the event of a tie between horses(Not in the video or the instructions but should be pointed out)
  + There is also nothing in Phase 2 that requires us to implement it hence this is potential area of improvement identified by me

A screenshot of a computer program

AI-generated content may be incorrect.

* + There can be 3 outcomes: tie, one winner, no winner yet. This means ideally, we would look at displaying the winner(s) only after checking each horse for winning. The winners would need to somehow be recorded to display them later(in the event of a tie). Due to the ability to add more horses which will be added in Phase 2, I chose to use an ArrayList that can be dynamically extended. This records any winning horse and passes it to printWinners()
  + printWinners() could be inside startRace() but I believe separating it makes the code more legible. If there are no winners it simply moves on, but if there are 1 or more winners it only prints them appropriately.
* Invalid races – the current code has no way of dealing with races that go on forever. Specifically, if all horses have fallen
  + Horse has fallen
  + Horse has had their confidence reduced to 0.0

A blue screen with white text

AI-generated content may be incorrect.

A computer screen shot of text

AI-generated content may be incorrect.

(also modified the code to use a loop to call moveHorse() on all horses

* Calls the method that inspects all horses, checking if all of them are unable to continue at which point, since it will come up false in the while statement, it output the suspending race message and ends the code.
* Its position in the while loop should ensure it is called both before the race begins, and before the next race state is printed.
* Terminal would not clear correctly, instead outputting states of the race consecutively. (this is more of a personal issue likely due to how the windows terminal works) A blue screen with white text

  AI-generated content may be incorrect.
  + Ended up altering the string in the line meant to clear the terminal as someone with the same problem received feedback in the forum. This is likely due to terminal commands in Windows simply being different.
  + <https://qmplus.qmul.ac.uk/mod/forum/discuss.php?d=584924>
* Ability to run races subsequently. Makes testing easier and will be useful for Part 2 where the horses’ confidence will need to be changed.

A computer screen shot of text

AI-generated content may be incorrect.

A computer screen shot of text

AI-generated content may be incorrect.

* Added the generic method allowing the user to input a string, which would ask the user if they wanted to start another race. If the user inputs “y” (yes), another race will be started.
* The looping calling each horse’s goBackToStart() (resetting them), has been moved to a separate method, this is so it can be called before the while loop, and when the user inputs “y”
* Added an if statement checking after every new race state, whether there have been any winners
  + Instead of running 2 checks (1 for tie, 1 for solo winner) every iteration it only runs one, until the final iteration where it will be 3, however it would be more efficient this way
  + E.g. 10 operations would have 9+3=12 checks, instead of 10\*2=20.
* The if statement also works as a sort of “has the race ended?” check, checking if there are winners, or horses can no longer move on.
  + The call to canRaceContinue() has also been moved here as logically it used to be checked before the race starts, and before every new state. However, due to the reset there is no need to check before the first iteration, with its new position here also working with the ability to start a new race, asking the user in both scenarios.