**Horse Race Simulator Report**

This report’s purpose is to document improvements made to the **Horse** and **Race** class of the simulator. It documents identified issues alongside a thorough explanation, providing possible solutions and testing evidence. Furthermore, it also explains key programming concepts used.

**Horse Class:**

The horse class makes use of encapsulation, a fundamental concept in Object Oriented Programming. This involved bundling related data alongside methods that utilise that data into a single unit (i.e., Class). This is done by setting the access modifiers for the class fields to **private**, which prevents access to the fields outside the Class, only allowing them to be manipulated via accessor (**getter**) and mutator (**setter**) methods.

The following are the mutator methods:

* getConfidence(): Returns the confidence level of the horse
* getDistanceTravelled(): Returns the total distance travelled by the horse
* getName(): Returns the horse’s name
* getSymbol(): Returns the character used to represent the horse
* hasFallen(): Returns a Boolean value whether the horse has fallen

The following are the accessor methods:

* fall(): Sets the horse as fallen
* moveForward(): Increments the distance travelled variable by one
* setConfidence(): Sets the confidence rating of the horse to the given value
* setSymbol(): Sets the horse’s symbol to the character given

The following is a private method I have made:

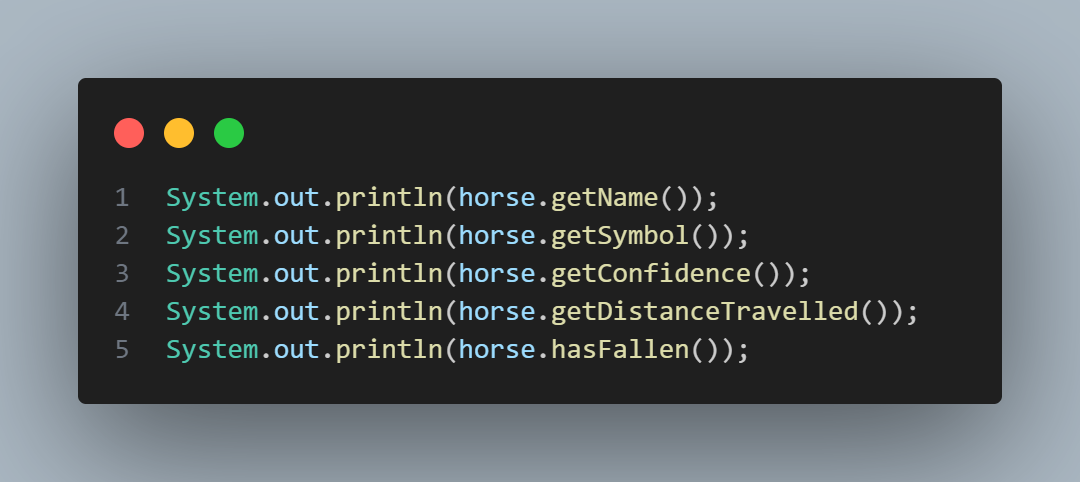
* validateConfidence():
  + This method takes one param: “double confidence”. Uses an if statement to check if the param exceeds the bounds. If the confidence is >= 1.0, it returns 0.99, else if the confidence is <= 0.0, it returns 0.01, else it returns the param as it was.
  + This method has been used within the constructor and setConfidence() to make sure the confidence value does not exceeds the bounds (0 and 1).

**Testing:**

I have provided tests for each method within the horse class to verify its correctness. Screenshots and explanations have also been provided. An external file Main.java has been used for testing. All tests will be conducted on the same instance of the Horse class:

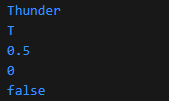
****

**Getter Method Testing:**

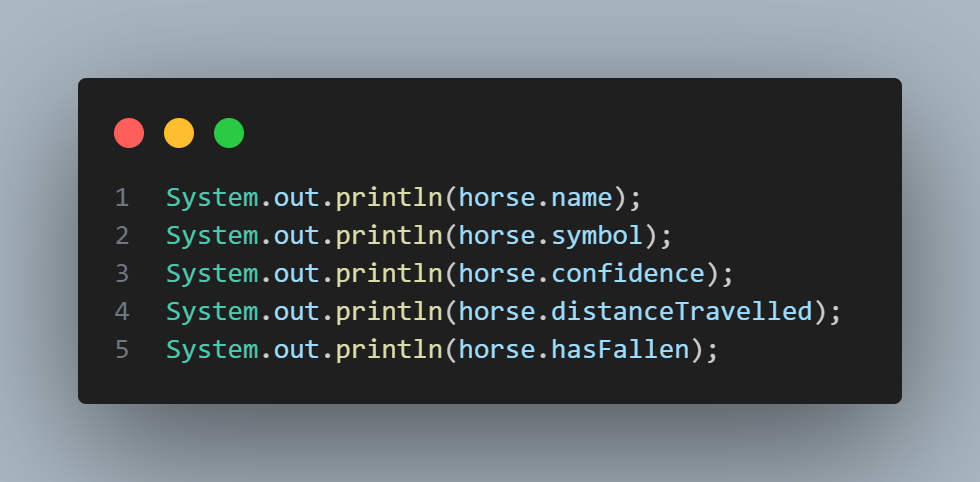


The above calls all the getter methods for each field of the Horse class.

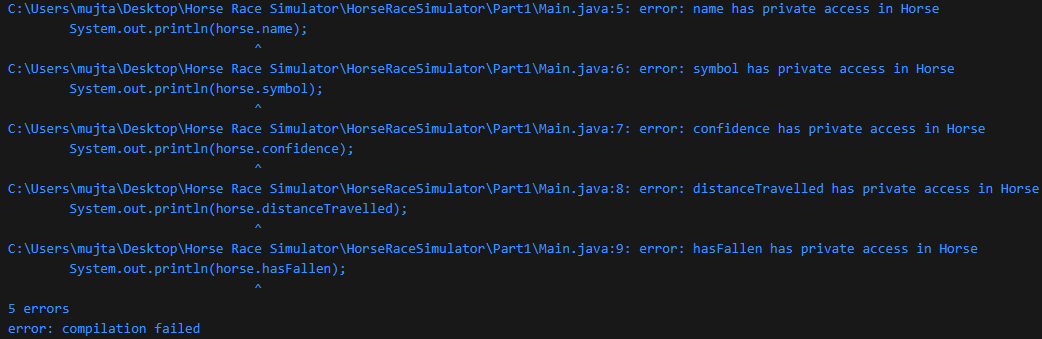
Expected output: Actual output:

* Thunder
* T
* 0.5
* 0
* False

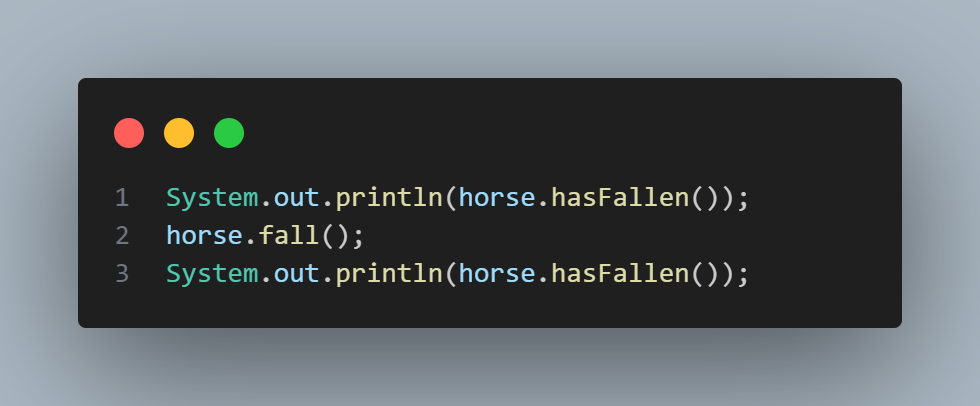
**Access Modifier Testing:**

****

The above code attempts to retrieve the Class fields using dot notation. As the access modifier has been set to **private** the code should not compile and throw an error.



**hasFallen() testing:**

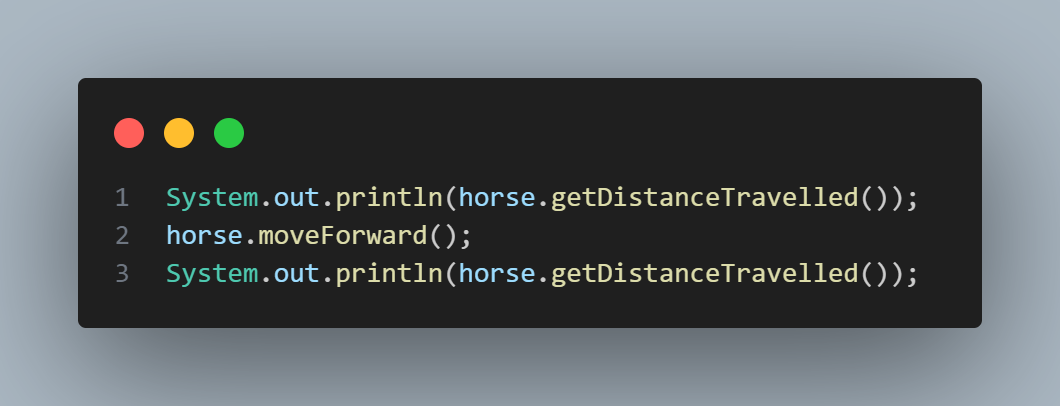


The above code first prints out the **hasFallen** field of the Horse Class. It then proceeds to call **fall()** which sets the field to true. The **hasFallen** field is then printed once more to show the change.

Expected output: Actual output:

* false
* true

**moveForward()** **testing:**

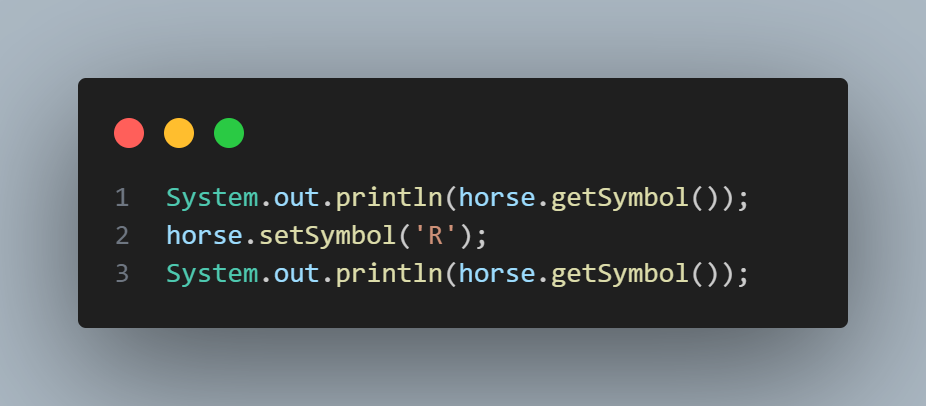


The above code prints out the **distanceTravelled** field of the Horse Class. It then proceeds to call **moveForward()** which increments the **distanceTravelled** by 1. The **distanceTravelled** field is then printed once more to show the change.

Expected output: Actual output:

* 0
* 1

**setSymbol() testing:**



The above code prints out the horse’s **symbol**. It then proceeds to call **setSymbol()** which takes a char parameter, and sets that param as the new symbol. The horse’s **symbol** is then printed once more to show the change.

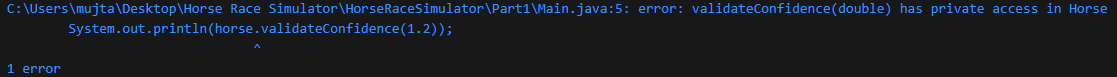
Expected output: Actual output:

* T
* R

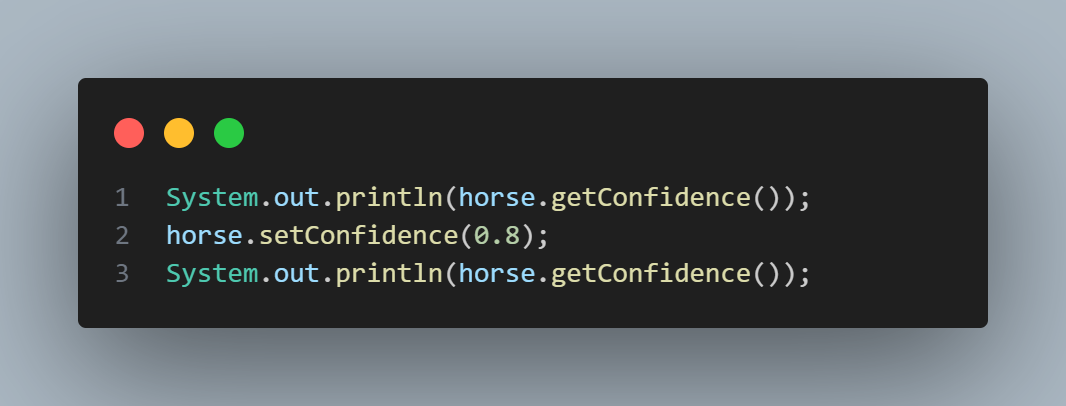
**Attempt to access validateConfidence() outside the class:**



The above code attempts to call the validateConfidece() method of the Horse Class. The access modifier has been set to **private** so the code should not compile and throw an error.



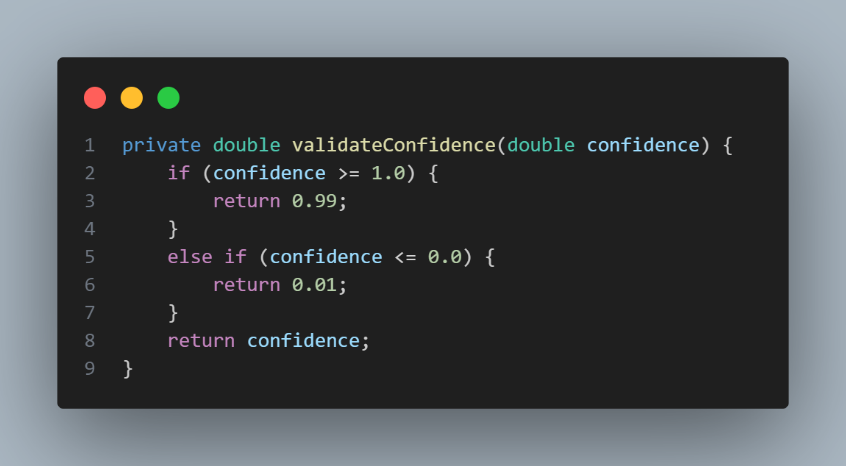
**setConfidence() testing:**



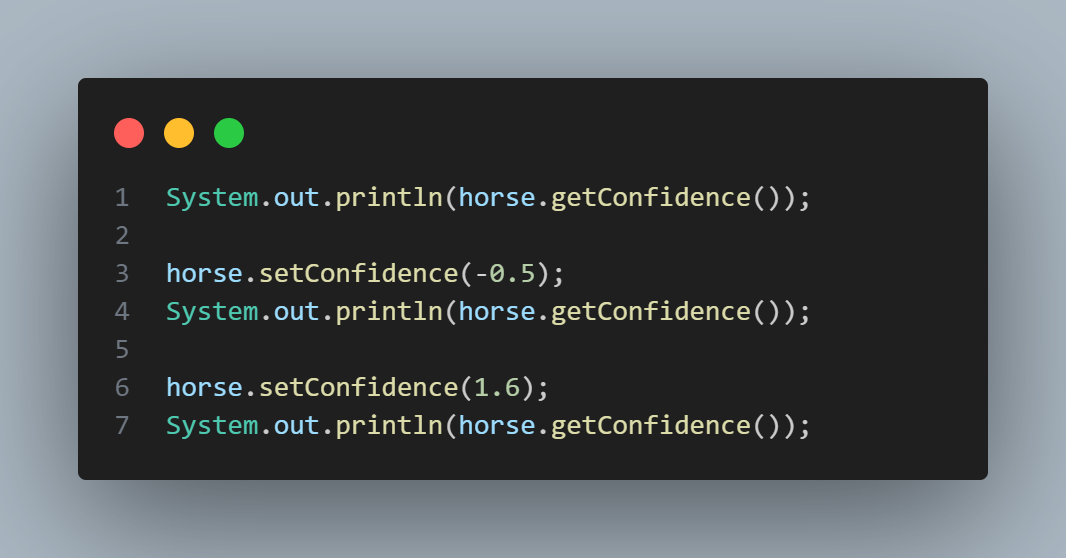
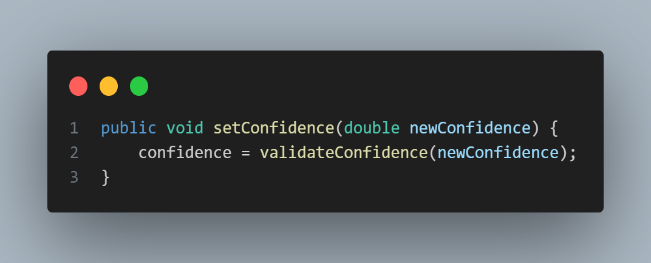
The above code prints out the horse’s **confidence**. It then proceeds to call **setConfidence()** which takes a double parameter, and sets that param as the new confidence. The horse’s **confidence** is then printed once more to show the change.

Expected output: Actual output:

* 0.5
* 0.8

**validateConfidence() testing**

The above is the **validateConfidence()** method.



The above code first prints the horse’s **confidence**. It is then set to -0.5, and the **confidence** is printed out again. The **confidence** is then set to 1.6, and is printed once more. As -0.5 is below the bounds, the confidence should be set to 0.01, and as 1.6 is greater than the bounds, the confidence should be set to 0.99.

The code to the right is how the **validateConfidence()** is called within **setConfidence().**

Expected output: Actual output:

* 0.5
* 0.01
* 0.99