OOP Coursework

~ Siddhant Misra, ID: 230319473

Part 1

Importance of encapsulation

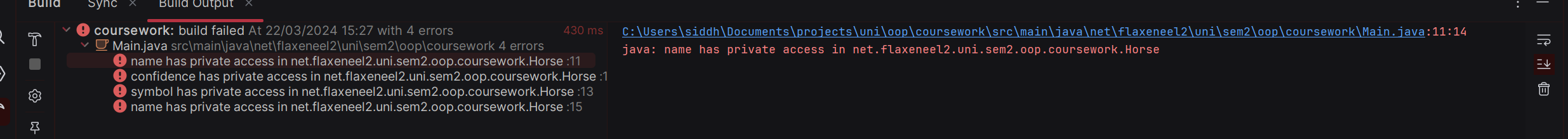
Encapsulation is used to prevent unauthorized access/modification of the properties of the Horse class. In this case it is useful as we will be forcing the rest of the code to be going through a function for reading/writing it, we can add further functionality to the getter/setter functions to log all these, this way we can know that no one cheated in the race by changing the attributes in between the races.

Testing encapsulation

Test 1: Trying to read from/write to a private field:

A computer screen shot of text

Description automatically generated



As we can see, the horse’s private properties cannot be read from/written to, since they are private.

Test 2: Using methods to read from/write to private fields:

Now, let us change the direct reads/writes to using the access/mutator (getters/setters).

A computer screen with many colorful text

Description automatically generated

Here we can see that the build completed successfully, and that there were no errors when the program was run, and it was able to edit the name without any problems.

A computer screen with white text

Description automatically generated

A black screen with red text

Description automatically generated

Improving the Race class

The race class does not show the winner when the race ends, we can make it show the winner by adding an indication of which horse won the race by adding the text for it next to the winning horse’s lane. We can use the `raceWonBy()` method that is already present in the race class:

A screen shot of a computer program

Description automatically generated

Edge cases:

1. There are times where more than one horse can win the race, for example:

A black background with white text

Description automatically generated

Here, since both b and c finished the race at the same time, it says that both won. This would make sense but we want to show the user that there was no clear win, rather that two (or more possibly) horses tied.

Changing the code for the print race, I added an array list for the horses that have won.

A screenshot of a computer

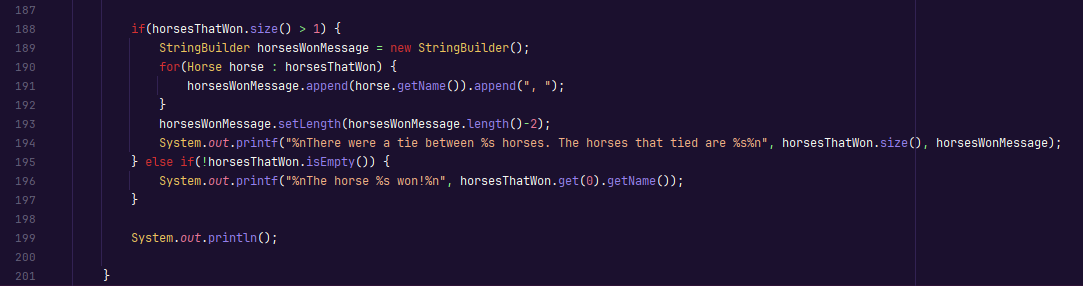
Description automatically generated

I then changed the message on the side of the lane from “Race Won” to “Finished”, as we will be listing who won or tied afterwards. I also added the horses that wont to the horsesThatWon array list

A computer screen shot of a program code

Description automatically generated

I then added a check to see if one or more horses tied, and made it display the appropriate message.



A screenshot of a computer program

Description automatically generated

A blue screen with white text

Description automatically generated

Now when 2 or more horses win, the user is shown a proper message signifying a tie.

1. There are times where a fallen horse can win. This should not happen as a horse falling means the horse is eliminated from the race. An eliminated horse cannot win. This is a simple fix.

A screen shot of a computer

Description automatically generated

We modify our raceWonBy() code to do a check to ensure the horse has not fallen.

A computer screen shot of text

Description automatically generated

With this change, a fallen horse can no longer win.

A screen shot of a computer

Description automatically generated

Here we can see even though the first horse reached the end, it was not counted as having finished the race.

1. Another issue is when all the horses fall. This would end up in an endless loop, which we can fix by doing a check to see if all the horses have fallen or not at the end of the main game loop.

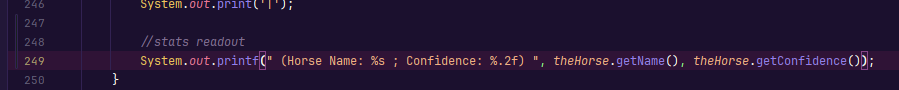
A screenshot of a computer program

Description automatically generated

A screen shot of a computer

Description automatically generated

1. The horse characters are printed in the lanes, but it can sometimes be a bit confusing to remember which horse had which symbol. To fix this problem, I added the following code to the print lane method.



It results in the races looking like this:

A computer screen shot of a code

Description automatically generated

This makes it clear which horse is which.

1. Horse confidences do not decrease when the horses fall. To fix this I made a small change to the horse class’s fall() function:

A screenshot of a computer code

Description automatically generated

This will reduce the confidence automatically when a horse falls:

A computer screen shot of a computer code

Description automatically generated

1. Horse confidence does not increase when a horse wins. This can be fixed by updating the horse confidence before printing the line. For this I can just edit the print lane function to add the update:

A screenshot of a computer program

Description automatically generated

Now the confidence readout updates at the end:

A computer screen shot of a computer code

Description automatically generated

Final code for task 1 can be found on <https://github.com/flaxeneel2/HorseRaceSimulator> (private repository) and also attached below:

Horse.java

package net**.**flaxeneel2**.**uni**.**sem2**.**oop**.**coursework**;**

**import** javax**.**swing**.\*;**

/\*\*

\* Write a description of class Horse here.

\*

\* **@author** Siddhant Misra

\* **@version** 1

\*/

public class Horse

**{**

//Fields of class Horse

private int distanceTravelled**;**

private char symbol**;**

private String name**;**

private boolean fallen**;**

private double confidence**;**

//Constructor of class Horse

/\*\*

\* Constructor for objects of class Horse

\*/

public Horse**(**char horseSymbol**,** String horseName**,** double horseConfidence**)**

**{**

**this.**setSymbol**(**horseSymbol**);**

**this.**setName**(**horseName**);**

**this.**setConfidence**(**horseConfidence**);**

**}**

//Other methods of class Horse

public void fall**()**

**{**

**this.**setFallen**(true);**

**this.**confidence**-=**0.1**;**

**}**

public int getDistanceTravelled**()** **{**

**return** distanceTravelled**;**

**}**

public void setDistanceTravelled**(**int distanceTravelled**)** **{**

**this.**distanceTravelled **=** distanceTravelled**;**

**}**

public char getSymbol**()** **{**

**return** symbol**;**

**}**

public void setSymbol**(**char symbol**)** **{**

**this.**symbol **=** symbol**;**

**}**

public String getName**()** **{**

**return** name**;**

**}**

public void setName**(**String name**)** **{**

**this.**name **=** name**;**

**}**

public boolean hasFallen**()** **{**

**return** fallen**;**

**}**

public void setFallen**(**boolean fallen**)** **{**

**this.**fallen **=** fallen**;**

**}**

public double getConfidence**()** **{**

**return** confidence**;**

**}**

public void setConfidence**(**double confidence**)** **{**

**this.**confidence **=** confidence**;**

**}**

public void moveForward**()** **{**

**this.**setDistanceTravelled**(this.**getDistanceTravelled**()+**1**);**

**}**

public void goBackToStart**()** **{**

**this.**setDistanceTravelled**(**0**);**

**}**

**}**

Race.java:

package net**.**flaxeneel2**.**uni**.**sem2**.**oop**.**coursework**;**

**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

\* **@author** Siddhant Misra

\* **@version** 1.0

\*/

public class Race

**{**

private int raceLength**;**

private Horse lane1Horse**;**

private Horse lane2Horse**;**

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...)

\*/

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

\*/

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"**);**

**}**

**}**

/\*\*

\* Start the race

\* The horse are brought to the start and

\* then repeatedly moved forward until the

\* race is finished

\*/

public void startRace**()**

**{**

//declare a local variable to tell us when the race is finished

boolean finished **=** **false;**

//reset all the lanes (all horses not fallen and back to 0).

lane1Horse**.**goBackToStart**();**

lane2Horse**.**goBackToStart**();**

lane3Horse**.**goBackToStart**();**

**while** **(!**finished**)**

**{**

//move each horse

moveHorse**(**lane1Horse**);**

moveHorse**(**lane2Horse**);**

moveHorse**(**lane3Horse**);**

//print the race positions

printRace**();**

//if any of the three horses has won the race is finished

**if** **(** raceWonBy**(**lane1Horse**)** **||** raceWonBy**(**lane2Horse**)** **||** raceWonBy**(**lane3Horse**)** **)**

**{**

finished **=** **true;**

**}**

**if(**lane1Horse**.**hasFallen**()** **&&** lane2Horse**.**hasFallen**()** **&&** lane3Horse**.**hasFallen**())** **{**

//all horses fell, game over, no one won.

System**.**out**.**println**(**"All the horses fell! There are no winners!"**);**

**break;**

**}**

//wait for 100 milliseconds

**try{**

TimeUnit**.**MILLISECONDS**.**sleep**(**100**);**

**}catch(**Exception ignored**){}**

**}**

**}**

/\*\*

\* 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

\*/

private void moveHorse**(**Horse theHorse**)**

**{**

//if the horse has fallen it cannot move,

//so only run if it has not fallen

**if** **(!**theHorse**.**hasFallen**())**

**{**

//the probability that the horse will move forward depends on the confidence;

**if** **(**Math**.**random**()** **<** theHorse**.**getConfidence**())**

**{**

theHorse**.**moveForward**();**

**}**

//the probability that the horse will fall is very small (max is 0.1)

//but will also will depends exponentially on confidence

//so if you double the confidence, the probability that it will fall is \*2

**if** **(**Math**.**random**()** **<** **(**0.1**\***theHorse**.**getConfidence**()\***theHorse**.**getConfidence**()))**

**{**

theHorse**.**fall**();**

**}**

**}**

**}**

/\*\*

\* Determines if a horse has won the race

\*

\* **@param** theHorse The horse we are testing

\* **@return** true if the horse has won, false otherwise.

\*/

private boolean raceWonBy**(**Horse theHorse**)**

**{**

**if** **(!**theHorse**.**hasFallen**()** **&&** theHorse**.**getDistanceTravelled**()** **==** raceLength**)**

**{**

**return** **true;**

**}**

**else**

**{**

**return** **false;**

**}**

**}**

/\*\*\*

\* Print the race on the terminal

\*/

private void printRace**()**

**{**

System**.**out**.**print**(**'\u000C'**);** //clear the terminal window

multiplePrint**(**'='**,**raceLength**+**3**);** //top edge of track

System**.**out**.**println**();**

ArrayList**<**Horse**>** horsesThatWon **=** **new** ArrayList**<**Horse**>();**

printLane**(**lane1Horse**);**

**if(**raceWonBy**(**lane1Horse**))** **{**

horsesThatWon**.**add**(**lane1Horse**);**

**}**

System**.**out**.**println**();**

printLane**(**lane2Horse**);**

**if(**raceWonBy**(**lane2Horse**))** **{**

horsesThatWon**.**add**(**lane2Horse**);**

**}**

System**.**out**.**println**();**

printLane**(**lane3Horse**);**

**if(**raceWonBy**(**lane3Horse**))** **{**

horsesThatWon**.**add**(**lane3Horse**);**

**}**

System**.**out**.**println**();**

multiplePrint**(**'='**,**raceLength**+**3**);** //bottom edge of track

**if(**horsesThatWon**.**size**()** **>** 1**)** **{**

StringBuilder horsesWonMessage **=** **new** StringBuilder**();**

**for(**Horse horse **:** horsesThatWon**)** **{**

horsesWonMessage**.**append**(**horse**.**getName**()).**append**(**", "**);**

**}**

horsesWonMessage**.**setLength**(**horsesWonMessage**.**length**()-**2**);**

System**.**out**.**printf**(**"%nThere were a tie between %s horses. The horses that tied are %s%n"**,** horsesThatWon**.**size**(),** horsesWonMessage**);**

**}** **else** **if(!**horsesThatWon**.**isEmpty**())** **{**

System**.**out**.**printf**(**"%nThe horse %s won!%n"**,** horsesThatWon**.**get**(**0**).**getName**());**

**}**

System**.**out**.**println**();**

**}**

/\*\*

\* print a horse's lane during the race

\* for example

\* | X |

\* to show how far the horse has run

\*/

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**();**

**if(**raceWonBy**(**theHorse**))** **{**

theHorse**.**setConfidence**(**theHorse**.**getConfidence**()+**0.1**);**

**}**

//print a | for the beginning of the lane

System**.**out**.**print**(**'|'**);**

//print the spaces before the horse

multiplePrint**(**' '**,**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**(**' '**,**spacesAfter**);**

//print the | for the end of the track

System**.**out**.**print**(**'|'**);**

//stats readout

System**.**out**.**printf**(**"\t(Horse Name: %s ; Confidence: %.2f) "**,** theHorse**.**getName**(),** theHorse**.**getConfidence**());**

**if(**raceWonBy**(**theHorse**))** **{**

System**.**out**.**print**(**"\t\t<-- Finished"**);**

**}**

**}**

/\*\*\*

\* print a character a given number of times.

\* e.g. printmany('x',5) will print: xxxxx

\*

\* **@param** aChar the character to Print

\*/

private void multiplePrint**(**char aChar**,** int times**)**

**{**

int i **=** 0**;**

**while** **(**i **<** times**)**

**{**

System**.**out**.**print**(**aChar**);**

i **=** i **+** 1**;**

**}**

**}**

**}**

package net**.**flaxeneel2**.**uni**.**sem2**.**oop**.**coursework**;**

public class Main **{**

public static void main**(**String**[]** args**)** **{**

startRace**();**

**}**

public static void testEncapsulation**()** **{**

Horse horse **=** **new** Horse**(**'a'**,** "horse1"**,** 0.5**);**

//This code will error:

// System.out.println(horse.name);

// System.out.println(horse.confidence);

// System.out.println(horse.symbol);

// horse.name = "Horse 1";

// horse.confidence = 0.4;

// horse.symbol = 'b';

//This code does not error:

horse**.**setName**(**"b"**);**

horse**.**setConfidence**(**0.4**);**

horse**.**setSymbol**(**'b'**);**

System**.**out**.**println**(**horse**.**getName**());**

**}**

/\*\*

\* Start a test race.

\*/

public static void startRace**()** **{**

Race race **=** **new** Race**(**20**);**

race**.**addHorse**(new** Horse**(**'♕'**,** "horse1"**,** 0.5**),** 1**);**

race**.**addHorse**(new** Horse**(**'b'**,** "horse2"**,** 0.9**),** 2**);**

race**.**addHorse**(new** Horse**(**'c'**,** "horse3"**,** 0.25**),** 3**);**

race**.**startRace**();**

**}**

**}**