Horse Race Simulation (Documentation)

Use of Encapsulation

Considering that the start to the project encourages an object-oriented approach, utilising encapsulation was a natural first step in its continuing it. By making private the fields in the Horse class, I have ensured that none of these fields may be accessed or changed by anything outside of that class. Of course, I have created methods within the class to change them when necessary but, in doing so, I have given full control over how these fields are changed to these methods. As I have designed them to catch invalid values being set to the fields, no other parts of the program may change any fields directly, only by calling my methods. This also prevents users from directly changing the values in any way that I have not allowed.

As a result, the data associated with the Horse class is safeguarded in such a way that all fields can only contain accepted values.

Horse Accessor Methods (retrieve data)

getConfidence()

getDistanceTravelled()

getName()

getSymbol()

hasFallen()

Horse Mutator Methods (modify data)

fall()

goBackToStart()

moveForward()

setConfidence()

setSymbol()

Horse Testing

Current Horse Code

```
/**
 * A class to represent Horse objects. These are to be used
 * with the Race class to represent the horses within the
 * race.
 *
 * @author Shaan Shah
 * @version 1.3
 */
public class Horse
{
    //Fields of class Horse
    private final String horseName;
    private char horseSymbol;
    private int horseDistance;
    private Boolean horseFallen;
    private double horseConfidence;



    //Constructor of class Horse
    /**
     * Constructor for objects of class Horse
     */
    public Horse(char horseSymbol, String horseName, double
horseConfidence)
    {
```

```
        this.setConfidence(horseConfidence);
        this.setSymbol(horseSymbol);
        this.horseName = horseName;

        this.horseFallen = false;
        this.horseDistance = 0;
    }

    /**
     * Print all details of the horse
     */
    public void printDetails() {
        System.out.println("Horse details\n");
        System.out.println("Name: " + this.getName());
        System.out.println("Symbol: " + this.getSymbol());
        System.out.println("Fallen: " + this.hasFallen());
        System.out.println("Confidence: " + this.getConfidence());
        System.out.println("Distance: " + this.getDistanceTravelled());
        System.out.println("\n");
    }

    /**
     * Perform various tests
     */
    public static void tests() {
        Horse horse = new Horse('🐴', "horse", 0.5);  // Create new horse
        System.out.println("Created new horse with symbol: 🐴, name: horse
and confidence: 0.5\n");

        horse.printDetails();

        // Change horse details
        horse.setConfidence(0.7);
        horse.setSymbol('🐴');
        horse.moveForward();
        System.out.println("Horse confidence changed to 0.7, symbol changed
to 🐴 and moved forward by 1\n");

        horse.printDetails();

        // Fall
        horse.fall();
        System.out.println("Horse has fallen\n");

        horse.printDetails();

        // Go back to start
        horse.goBackToStart();
        System.out.println("Horse has moved back to the start\n");

        horse.printDetails();

        // Attempt to enter invalid data
        horse.setConfidence(-1);
        System.out.println("Attempted to set confidence to -1\n");

        horse.printDetails();

        horse.setConfidence(1);
        System.out.println("Attempted to set confidence to 1\n");
```

```
        horse.printDetails();

        horse.setConfidence(0);
        System.out.println("Attempted to set confidence to 0\n");

        horse.printDetails();

        horse.setConfidence(2);
        System.out.println("Attempted to set confidence to 2\n");

        horse.printDetails();
    }

    /**
     * Main method (only used for testing and may be removed)
     */
    public static void main(String[] args) {
        tests();
    }

    //Other methods of class Horse
    public void fall()
    {
        horseFallen = true;
        this.setConfidence(this.getConfidence() - 0.1);
    }

    public double getConfidence()
    {
        return horseConfidence;
    }

    public int getDistanceTravelled()
    {
        return horseDistance;
    }

    public String getName()
    {
        return horseName;
    }

    public char getSymbol()
    {
        return horseSymbol;
    }

    public void goBackToStart()
    {
        horseDistance = 0;
    }

    public boolean hasFallen()
    {
        return horseFallen;
    }

    public void moveForward()
    {
        horseDistance += 1;
```

```
    }

    public void setConfidence(double newConfidence)
    {
        if (newConfidence <= 0) {
            this.horseConfidence = 0.1;  // Sets a lower limit of 0.1
confidence
            return;
        }
        if (newConfidence >= 1) {
            this.horseConfidence = 0.9;  // Sets an upper limit of 0.9
confidence
            return;
        }
        horseConfidence = newConfidence;
    }

    public void setSymbol(char newSymbol)
    {
        horseSymbol = newSymbol;
    }

}
```

Test Explanation

I have created a main method that will call the 'tests()' method, which uses every other method within the Horse class to ensure they are all working as intended.

To start, the method creates a new object of class Horse with certain values. It then calls a method on it that I added called printDetails, which outputs all details of the horse in a clear manner using the accessor methods. This allows us to see if the values chosen were applied to the object correctly while also testing the accessor methods.

In the next section of tests, we call several of the other methods within the class to change the values we started with, after which printDetails is called again to ensure this worked correctly.

Two mutator methods, goBackToStart and fall, have not been called yet. Since they can change the details of the horse, they would have interfered with the testing of the previous methods, so they have been given their own sections. Calling fall should set horseFallen to true and lower horseConfidence, and goBackToStart should set horseDistance to 0 and set horseFallen back to false. We call fall and print the details, then call goBackToStart and print the details.

The following tests are made to check for inputting invalid data to the setConfidence method, as this method has some extra validation within it. The other fields can hold any value so long as it pertains to a certain data type, but confidence must specifically be a decimal number between 0 and 1. Therefore, it has an if statement checking if the value is out of range, and setting confidence to an appropriate replacement if it is. More specifically, it sets confidence to either 0.1 (the limit specified by McFarewell), if the value is less than or equal to 0, or to 0.9, if the value is greater than or equal to 1. The invalid values tested were -1, 0, 1 and 2.

Test Results

```
Created new horse with symbol: 🏇, name: horse and confidence: 0.5

Horse details

Name: horse
Symbol: 🏇
Fallen: false
Confidence: 0.5
Distance: 0


Horse confidence changed to 0.7, symbol changed to 🐎 and moved forward by 1

Horse details

Name: horse
Symbol: 🐎
Fallen: false
Confidence: 0.7
Distance: 1


Horse has fallen

Horse details

Name: horse
Symbol: 🐎
Fallen: true
Confidence: 0.6
Distance: 1
```

```
Horse has moved back to the start

Horse details

Name: horse
Symbol: 🐴
Fallen: true
Confidence: 0.6
Distance: 0


Attempted to set confidence to -1

Horse details

Name: horse
Symbol: 🐴
Fallen: true
Confidence: 0.1
Distance: 0


Attempted to set confidence to 1

Horse details

Name: horse
Symbol: 🐴
Fallen: true
Confidence: 0.9
Distance: 0
```

```
Attempted to set confidence to 0

Horse details

Name: horse
Symbol: 🐴
Fallen: true
Confidence: 0.1
Distance: 0


Attempted to set confidence to 2

Horse details

Name: horse
Symbol: 🐴
Fallen: true
Confidence: 0.9
Distance: 0
```

All tests have been passed based on the explanations, code comments and print statements.

Examining Race

At a first glance, I noticed a few main changes I wanted to make: an empty catch statement when using TimeUnit sleep in startRace and the lack of a winner message (as outlined in the instructions).

The winner message could be implemented using a series of if statements existing outside of the while loop, so that it is only sent after the race is finished. These if statements would check the conditions of who won by calling raceWonBy on each horse in the race and seeing if it is true. The only exception would be the third horse, since their message would be placed in the else statement that runs if neither of the other horses won. The resulting code would look like this:

```
if (raceWonBy(lane1Horse)) {
    System.out.println("And the winner is " + lane1Horse.getName());
}
else if (raceWonBy(lane2Horse)) {
    System.out.println("And the winner is " + lane2Horse.getName());
}
else {
    System.out.println("And the winner is " + lane3Horse.getName());
}
```

This would be a viable alternative solution, but I noticed that there was much repetition of the same code. Therefore, I created the following function:

```
private void announceWinner(Horse horse) {
    System.out.println("And the winner is " + horse.getName());
}
```

And replaced the above if else blocks with this:

```
if (raceWonBy(lane1Horse)) {
    announceWinner(lane1Horse);
}
else if (raceWonBy(lane2Horse)) {
    announceWinner(lane2Horse);
}
else {
    announceWinner(lane3Horse);
}
```

This approach achieves the same goal, but uses a new method instead of repeating code.

In the try-catch section, I changed the exception to an InterruptedException (more specific than the general Exception) and handled it using 'Thread.currentThread().interrupt()' to set the thread's interrupt flag. An alternative solution would have been to use e.printStackTrace(), but I decided not to go with this approach due to it being a less specific way of handling an exception than the one I implemented.

On top of this, I decided to make some extra changes that were not entirely necessary, but still somewhat helpful. For example, the raceWonBy method was implemented in such a way that it returned true when an if statement's condition is met and false otherwise. To simplify this, I

altered it to simply return the Boolean value of the expression within the condition (‘theHorse.getDistanceTravelled() == raceLength’).

I also replaced the Unicode escape code ‘\u2322’ to the character ‘⌒’ directly for clarity’s sake, within the section of the printLane function that prints this character when a horse is fallen.

Finally, I noticed that the comment for the multiplePrint function used a printmany call in its example instead of multiplePrint, so I corrected this.

Race Testing

Current Race Code

```
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
 * @version 1.2
 */
public class Race
{
    private final 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;
    }

    /**
     * Perform various tests
     */
```

```
    public static void tests() {
        Race race = new Race(5);  // Create new race

        // Create new horses
        Horse horse1 = new Horse('♘', "Epona", 0.5);
        Horse horse2 = new Horse('♞', "Torrent", 0.5);
        Horse horse3 = new Horse('♚', "Ludwig", 0.9);

        // Add the horses to the race
        race.addHorse(horse1, 1);
        race.addHorse(horse2, 2);
        race.addHorse(horse3, 3);

        race.startRace();
    }

    /**
     * Main method to run tests (may be removed later)
     */
    public static void main(String[] args) {
        tests();
    }

    /**
     * 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;
            }

            //wait for 100 milliseconds
            try{
                TimeUnit.MILLISECONDS.sleep(100);
            }catch(InterruptedException e){
                Thread.currentThread().interrupt();
            }
        }

        if (raceWonBy(lane1Horse)) {
            announceWinner(lane1Horse);
        }
        else if (raceWonBy(lane2Horse)) {
            announceWinner(lane2Horse);
        }
        else {
            announceWinner(lane3Horse);
        }
    }

    private void announceWinner(Horse horse) {
        System.out.println("And the winner is " + horse.getName());
    }

    /**
     * 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)
    {
        return theHorse.getDistanceTravelled() == raceLength;
    }

    /***
     * 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();

        printLane(lane1Horse);
        System.out.println();

        printLane(lane2Horse);
        System.out.println();

        printLane(lane3Horse);
        System.out.println();

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

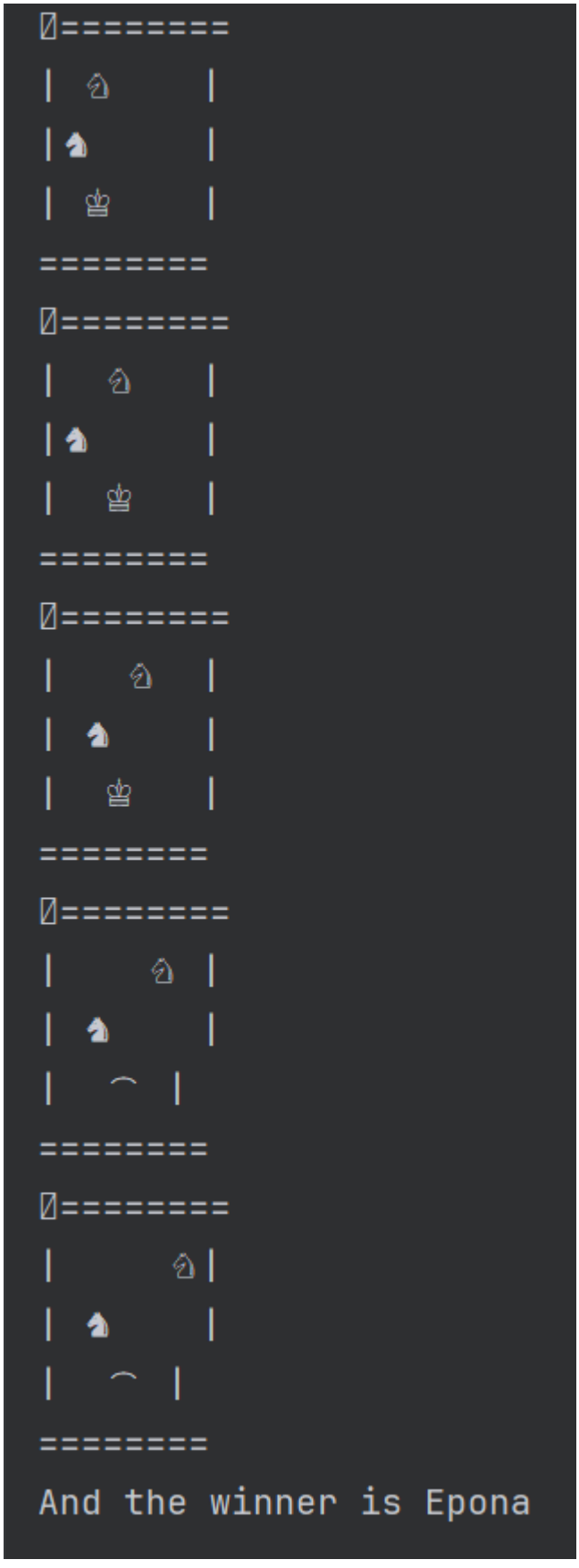
```
        //calculate how many spaces are needed before
        //and after the horse
        int spacesBefore = theHorse.getDistanceTravelled();
        int spacesAfter = raceLength - theHorse.getDistanceTravelled();

        //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('|');
    }


    /***
     * print a character a given number of times.
     * e.g. multiplePrint('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;
        }
    }
}
```

Test Explanation

I conducted this test in a similar way to with Horse. I created a main method that calls a method named tests. This method creates a new race of distance 5 and 3 new horses with distinct names and symbols that are then added to this race. It then calls startRace on this race.

In terms of using this test to debug the code, it was a much less complicated procedure than with Horse, hence why the tests method is so much simpler. This is because it all mostly happened through observation as I ran the code multiple times.

Test Results

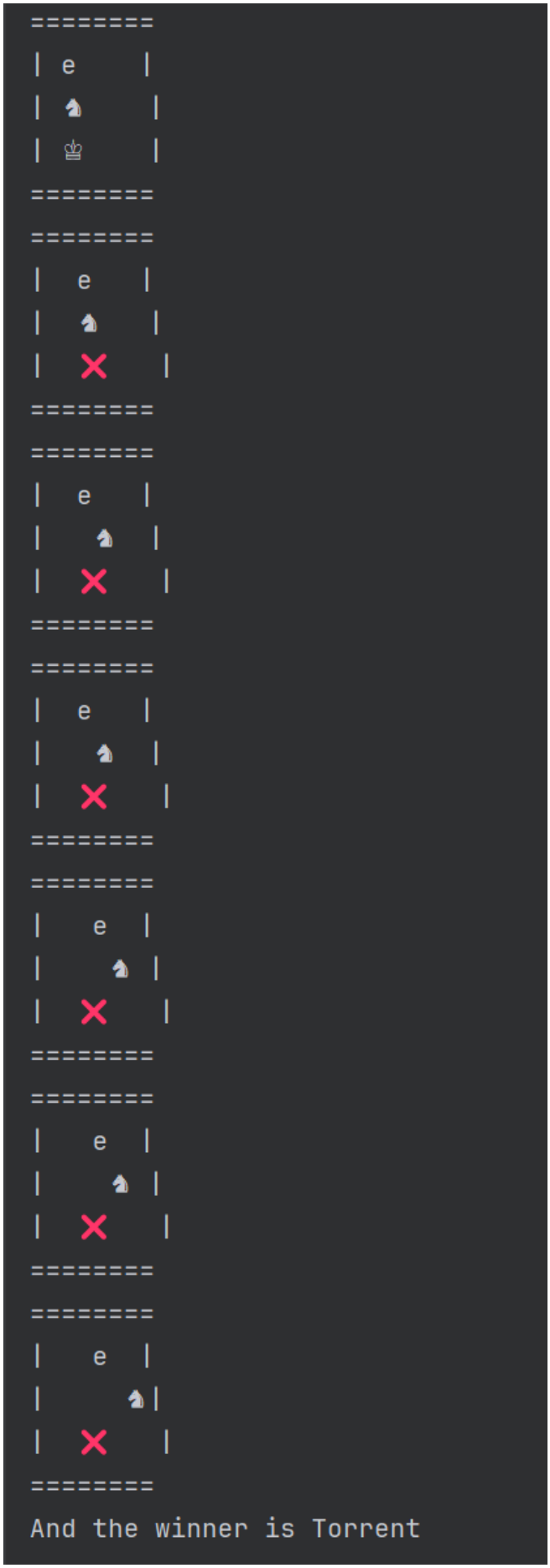

And the winner is Epona

Debugging

Immediately, it can be observed that the Unicode escape character to clear the terminal is not working. However, this is likely due to me using an IDE to run the code rather than a real terminal. Fortunately, this does aid in the testing of the other aspects of Race by allowing us to view each stage of the race separately, so we will not further examine that specific element just yet.

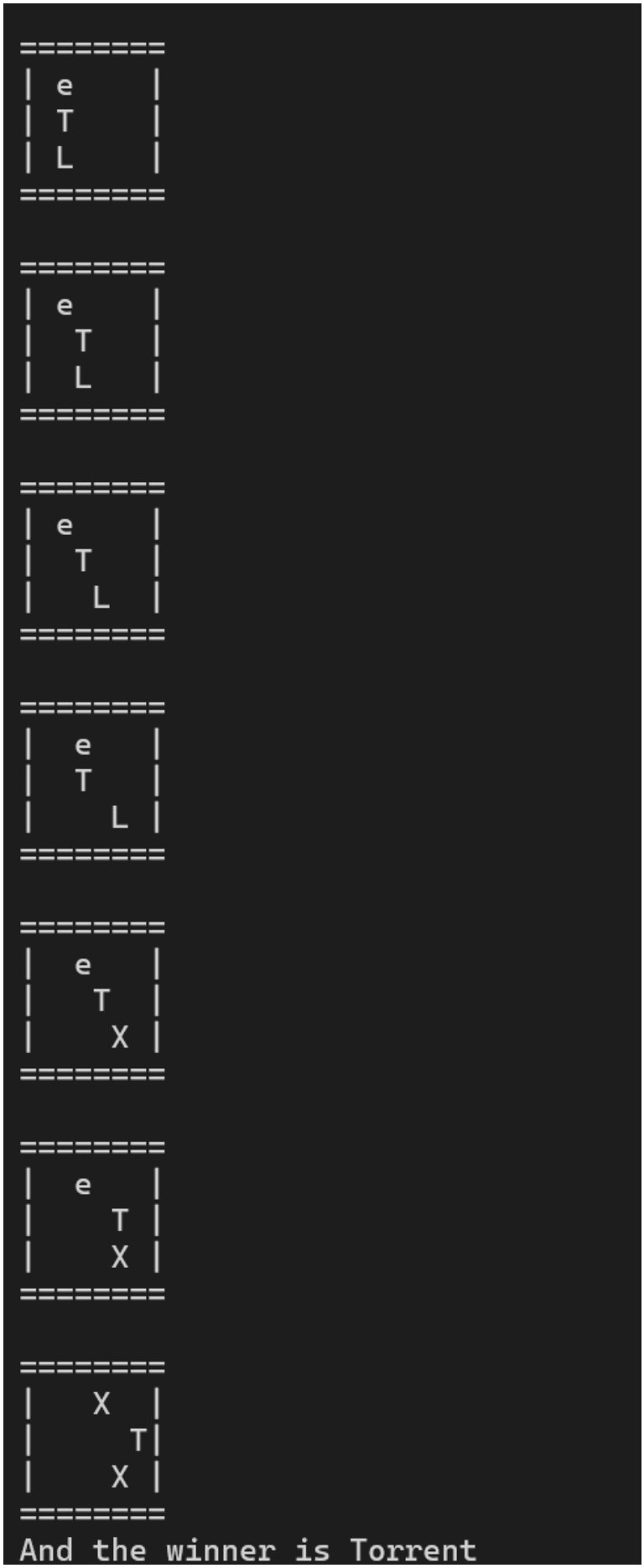
Another noticeable issue is the layout of the tracks: there is an invalid character at the top and the end of the track appears to change position once the horse's symbol changes to the fallen icon.

The unaccepted character turned out to be the Unicode escape character to clear the terminal, so this was removed. To further understand why the end of the track was not appearing properly, I changed one of the horse symbols to something else as well as change the dead symbol to a red cross (which better matched the example image anyway). When I called the main method, I saw the following:


========
| e |
| ♞ |
| ♔ |
========
========
| e |
| ♞ |
| ✕ |
========
========
| e |
| ♞ |
| ✕ |
========
========
| e |
| ♞ |
| ✕ |
========
========
| e |
| ♞ |
| ✕ |
========
========
| e |
| ♞ |
| ✕ |
========
========
| e |
| ♞|
| ✕ |
========
And the winner is Torrent

This all but confirms that the end of the track is out of place simply due to different characters taking up different amounts of space. In printLane, it seems to be assumed that each character would be the same length; the amount of spaces before and after the symbol is calculated based on horseDistance and the raceLength, neither of which change depending on the length of the symbol.

It is understandable that it would be nigh impossible to go through every single Unicode character and adjust the spaces between the symbols accordingly. Therefore, I found that the best solution would be to use a monospace font, to give each character an equal length. However, this is not an issue when running the program in the real terminal, rather than the IDE's terminal. Therefore, some testing shall be conducted in the real terminal:

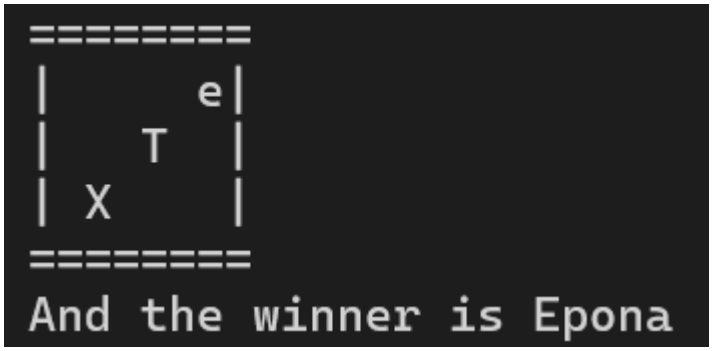

========
| e |
| T |
| L |
========
========
| e |
| T |
| L |
========
========
| e |
| T |
| L |
========
========
| e |
| T |
| L |
========
========
| e |
| T |
| X |
========
========
| e |
| T |
| X |
========
========
| X |
| T|
| X |
========
And the winner is Torrent

I have changed the symbols to be upper case and lower case, with there being no issue with spacing. However, even when adding back the code to clear the terminal, we can still see each stage of the race. To remedy this, I tried a different method using the following code:

```
private void clearScreen() {
    System.out.print("\033[H\033[2J");
    System.out.flush();
}
```

This creates a new method that uses an ANSI escape code to clear the screen instead. To use it, I simply call this method instead of using 'System.out.print('\u000C')'.

Testing again, the following is displayed in the terminal at the end of the race:

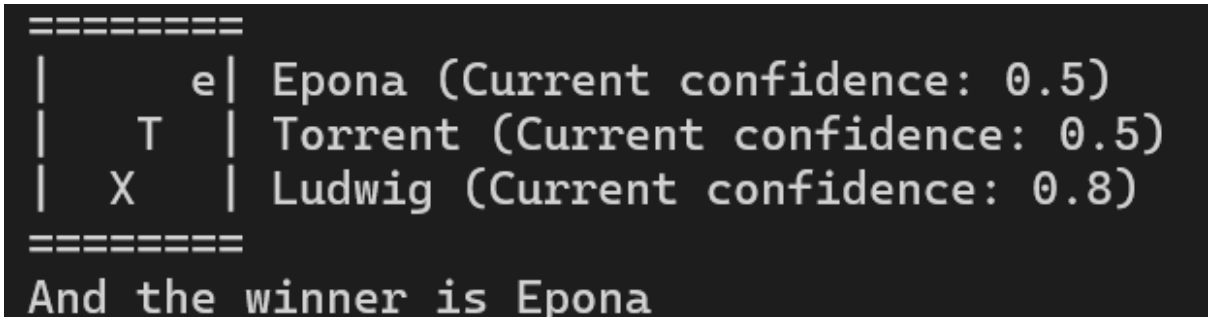


The terminal is now cleared.

This leaves one more noticeable issue: the lack of information on the horses. The specification shows each track having the name of its horse alongside their confidence rating. This feature can be easily added by appending this code to the end of the printLane method:

```
//print the horse's details
System.out.print(" " + theHorse.getName() + " (Current confidence: " +
theHorse.getConfidence() + ")");
```

Doing this will print the horse's details right after the '|' character at the end of each lane is printed. Testing the code again to verify this gives the following:



Note that Ludwig has confidence 0.8 instead of 0.9 because he had fallen during the race. With that, all the functionality outlined for Race is now in place.

Conclusion

My implementation of the Horse class alongside the changes to the Race class has resulted in a program that simulates horse races according to the specifications and does so in a way that closely resembles the examples.

I now consider the textual version of the Horse Race Simulator to be complete.