ERROR 1: if all horses fell then the race would go on forever

**Encapsulation Explanation**

Encapsulation combines attributes and methods into a class such that in order to modify or access data in an encapsulated class you would need to interface with the appropriate methods. This ensures that data isn’t mishandled by external code thus making programs using encapsulation less vulnerable to bugs.

In the class of the Horse class, there are 5 getter methods and 5 setter methods. The getter methods are: getConfidence, getDistanceTravelled, getSymbol, getName and hasFallen. The setter methods are: fall, moveForward, setConfidence, setSymbol and goBackToStart. The getter methods prevent unwanted changes while maintaining accessibility by acting as an interface to the data which means that a programmer does not have to deal with the actual data but a copy returned by a getter function. The setter methods prevent unwanted changes by only allowing certain changes to data since you can only change the class’s attributes using a setter method. For example, the setter method goBackToStart is one of the two ways to change the value of the horseDistanceTravelled attribute but goBackToStart method only sets the horseDistanceTravelled attribute to zero, this behaviour protects the attribute from unwanted changes because the only reason that the horseDistanceTravelled should be changed is if the race is restarting (Set to zero all the horse start at the same point) or when the horse needs to be moved forward which is set by moveForward method.

**Issues and areas of improvement Identified:**

1. **Indefinite looping of race**: If all the horse fell then the race loop will go on forever since the only case that the while loop will end is if a horse won but if a horse fell it can no longer move forward therefore can’t win.
2. **The race could start with the three lanes with null values**: Since the horses are set to null initially and there was no way to add horses without externally adding them the race could start with the three lanes set to null thus causing a null pointer error.
3. **Only a single round**: There was only a single round in the game which makes the fact that the confidence of a horse lowering after a fall irrelevant.
4. **Customisation**: being able to customise the horses will make the game more entertaining.
5. **Final winner**: Since there are multiple rounds a final winner should be found by totalling the wins of each horse.

**Updated code:**

import java.util.concurrent.TimeUnit;  
import java.lang.Math;  
import java.util.\*;  
  
*/\*\*  
 \* A three-horse race, each horse running in its own lane  
 \* for a given distance  
 \*   
 \* @author Taima Holden  
 \* @version 28/04/2025  
 \*/*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  
 \* ERROR: this was not used when assigning  
 \* @param distance the length of the racetrack (in metres/yards...)  
 \*/* public Race(int distance)  
 {  
 // initialise instance variables  
 this.raceLength = distance;  
 this.lane1Horse = null;  
 this.lane2Horse = null;  
 this.lane3Horse = null;  
 }  
   
 */\*\*  
 \* ERROR:this was not use so: lane1Horse = theHorse instead of this.....  
 \* 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()  
 {  
 addHorses();  
 for (int i = 0; i < 6; i++){  
 //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 all of the horse have fallen then there is a draw  
 else if (lane1Horse.hasFallen() && lane2Horse.hasFallen() && lane3Horse.hasFallen())  
 {  
 System.out.println("Draw. All horses have fallen!");  
 finished = true;  
 }  
  
  
 //wait for 100 milliseconds  
 try{  
 TimeUnit.MILLISECONDS.sleep(100);  
 }catch(Exception e){}  
 }  
 }  
  
 //printWinnerMessage(WinnerHorse);  
 }  
  
 */\*\*  
 \* Tell the user to name and give a symbol to the three horses in the race.  
 \*/* private void addHorses()  
 {  
 for (int i = 1; i <= 3; i++)  
 {  
 String HorseName = input("Enter horse name:");  
 String HorseSym = input("Enter horse symbol:");  
 addHorse(new Horse(HorseSym.charAt(0), HorseName, Math.random()), i);  
 }  
 }  
  
 */\*\*  
 \* Outputs a message then takes user input and returns it.  
 \*/* private String input(String mes)  
 {  
 System.out.println(mes);  
 Scanner Keyboard = new Scanner(System.in);  
 return Keyboard.nextLine();  
 }  
   
 */\*\*  
 \* 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.setConfidence(theHorse.getConfidence() - 0.1);  
 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.getDistanceTravelled() == raceLength)  
 {  
 printWinnerMessage(theHorse);  
 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();  
   
 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('\u274C');  
 }  
 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('|');  
  
 //prints the horse's name and confidence rating  
 printNameConfidence(theHorse);  
 }  
  
 */\*\*\*  
 \* prints the name and confidence rating of a horse  
 \* e.g PIPPI LONGSTOCKINGS (Current confidence 0.6)  
 \*/* private void printNameConfidence(Horse TheHorse)  
 {  
 String HorseName = TheHorse.getName();  
 double Confidence = TheHorse.getConfidence();  
  
 System.out.print(" " + HorseName + " (Current confidence " + Confidence + ')');  
 }  
  
 */\*\*\*  
 \* 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;  
 }  
 }  
  
 */\*\*\*  
 \* prints out a given string.  
 \*  
 \* @param Message The message we want to print  
 \*/* private void println(String Message)  
 {  
 System.out.println(Message);  
 }  
  
 */\*\*\*  
 \* prints out the winner message when a horse wins.  
 \*  
 \* @param WinnerHorse The horse that won that we want to print out its name in the winner message.  
 \*/* private void printWinnerMessage(Horse WinnerHorse)  
 {  
 System.out.println(WinnerHorse.getName() + " has won!");  
 }  
}

**List of changes:**

1. A message stating which horse won each race is printed now.
2. Users can now customise the name and symbol of three horses.
3. The horses name and confidence level is now displayed on the right of the race track during a race.
4. There are 5 races in the game now.
5. If all horses fall the race ends preventing an endless loop.