Object Oriented Programming Project

Horse Race Simulation

ECS414U/ IOT414U

Scenario

You've been entrusted with a Java project previously handled by Sunny McFarewell, who has now departed. Your task involves continuing the development of a horse race simulation. You now have access to all of McFarewell's code, which includes two main classes: `Horse` and `Race`. You can find this source code on the same QM+ page where you accessed these instructions.

While the `Race` class is almost complete, it still contains some errors that need attention. However, the `Horse` class hasn't been developed yet. Here's the outline of the `Horse` and `Race` objects McFarewell was tasked with simulating:

`Horse` object

1. The `Horse` objects represent the horses in the race.
2. Each horse has a *name* and a *symbol* (1 character). The symbol is used for display during the race, while the name is used to announce the winner.
3. During the race, a horse may fall, resulting in its elimination from the race.
4. Horses are assigned a *confidence rating* ranging from 0 to 1. A higher confidence rating indicates faster running speed, but also increases the likelihood of falling.
5. Winning a race slightly increases a horse's confidence, while falling during a race slightly decreases it.

`Race` object

1. The `Race` object utilises `Horse` objects to simulate races, displaying them within the Command Prompt.
2. Races are configurable with varying lengths, measured in meters (or yards).
3. Each race comprises a specified number of lanes, where horses are allocated before the race begins. Some lanes may remain vacant.
4. Throughout the race, an animated display of the race progress is presented within the command-line terminal, providing a visual representation of the race in real-time. For an example, refer to simulation.mov
5. Once the race concludes, the results are promptly displayed on the terminal screen.

Part I: A Textual Racing Simulator

1. Write the `Horse` class according to the following specification:

a) The class should be named Horse

b) It should have 5 fields to store:

i. The name of the horse (e.g., "PIPPI LONGSTOCKING")

ii. A single (Unicode) character used to represent the horse (e.g., 🐴)

iii. The distance travelled by the horse as a whole number

iv. A boolean indicating whether or not the horse has fallen.

v. The confidence rating of the horse, represented as a decimal number between 0 and 1.

c) The constructor of the class should have the following signature:

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

d) The class should provide the following public methods:

- `fall()`: Sets the horse as fallen.
- `getConfidence()`: Returns the confidence rating of the horse.
- `getDistanceTravelled()`: Returns the distance traveled by the horse.
- `getName()`: Returns the name of the horse.
- `getSymbol()`: Returns the character used to represent the horse.
- `goBackToStart()`: Resets the horse to the start of the race.
- `hasFallen()`: Returns true if the horse has fallen, false otherwise.
- `moveForward()`: Increments the distance traveled by the horse by 1.
- `setConfidence(double newConfidence)`: Sets the confidence rating of the horse to the given value.
- `setSymbol(char newSymbol)`: Sets the character used to represent the horse to the given character."

e) Encapsulation should be implemented to ensure that the values of the fields are always acceptable.

(*Note: It's crucial that your Horse class precisely meets this specification to ensure compatibility with the Race class written by McFarewell*).

In your report:

- Explain the use of encapsulation to safeguard data in your class. Clearly identify which methods in the class are accessor methods (for retrieving data) and which are mutator methods (for modifying data).
- Present your testing process with screenshots and detailed explanations of the tests conducted.

2. Examine and improve the `Race` Class

Begin by thoroughly examining the Race class provided by McFarewell, which encompasses numerous features covered in the lectures.

a) The **`startRace()`** method serves as the main method. Enhance this method to display the name of the winning horse in the terminal upon race completion.

b) Conduct comprehensive testing of the Race class to identify potential areas for improvement. Document any encountered issues within the code and attempt to resolve as many of them as possible. Credit will be given for recognising problems, even if you're uncertain about the solutions. Additionally, propose alternative solutions for identified issues, regardless of whether your code implementation is successful. Below is a sample snapshot for a three-horse race[1]:

```
==================
|         🏇       | PIPPI LONGSTOCKING (Current confidence 0.6)
|               🏇 | KOKOMO (Current confidence 0.6)
|           ❌[2]   | EL JEFE (Current confidence 0.4)
==================

And the winner is KOKOMO
```

In your report:

- Provide the updated code of your class, including all the modifications you've made.
- Showcase the modification you've implemented, utilising screenshots and detailed explanations of the testing conducted.

[1] Additionally, a video version is available on the project's page.

[2] ❌ denotes a fallen horse

Part II: GUI Module Development

Now, let's take your Racing application to the next level by introducing a user-friendly graphical user interface (GUI). This GUI version will expand upon the groundwork established in Part I, bringing in fresh and new features to create an engaging and interactive experience. In this phase, while you have the freedom to be creative, it's essential to meet the following requirements for the GUI module of the project:

Interactive Track Design: Allow users to design their own racing tracks with customisable features like specifying the number of tracks, adjusting track lengths, and customising other elements to tailor the racing experience to their preferences.

Customisable Horses: Offer users the opportunity to personalise their horses with a range of customisation options, moving beyond the simple letter symbol used in Part I. From choosing breed and coat colour to selecting equipment and accessories, let them create their ultimate racing companion.

Statistics and Analytics: Provide statistics and analytics for each race, including horse performance metrics, track records, betting odds, and historical data from past simulations.

Key Features:

- Implement a statistics module that captures and analyses various aspects of each race.
- For each horse, track and display performance metrics such as average speed, finishing times, and win ratios.
- Keep track of historical data for each horse, including past race performances.
- Provide an interface for users to view and analyse horse statistics.

Virtual Betting System: Enhance the thrill of horse racing by incorporating a virtual betting system. Allow users to place bets on races using virtual currency, with odds calculated based on horse performance, track conditions, and other factors. Track users' betting history, winnings, and losses, and provide feedback on their betting strategies to help them improve their skills.

Key Features:

- Develop a robust betting odds algorithm that calculates odds based on horse performance metrics.
- Provide users with real-time access to current odds before the start of each race, allowing them to make informed betting decisions.
- Update odds dynamically as users place bets to reflect changes in betting patterns and track conditions.
- Design and implement a user-friendly interface for online betting within the horse racing game.
- The interface should allow users to select horses to bet on and enter their desired bet amount.

You will receive higher marks for the introduction of any innovative features that demonstrate creativity and add value to the application.

Part III: Git Integration

Now it's time to unlock the full potential of your project with Git integration.

- Initialise a local Git repository for your project right from the start. Name the root folder of your project `HorseRaceSimulator`.
- Organise your local repository into two main folders: `Part1` and `Part2`.
- In the `Part1` folder, include all the code related to Part I of the project (the textual version).
- Likewise, within the `Part2` folder, include the code pertinent to Part II of the project (the graphical version).
- Local repository Structure:

```
HorseRaceSimulator/
│
├── .git/
│
├── Part1/
│   └── (Code for Part I)
│
└── Part2/
    └── (Code for Part II)
```

  In this structure, the `.git` folder is hidden, representing the Git repository, while `Part1` and `Part2` contain the code for their respective parts of the project.

- Create a **private** GitHub.com repository to host your project code.
- Include a detailed README file in your repository. This document should provide clear instructions on running your code, including setup steps, dependencies, and usage guidelines.
- Use descriptive and meaningful commit messages for each change you make to the codebase. Clear commit messages make it easier to track changes and understand the purpose of each commit.
- Create a new branch named `gui-development` for implementing the GUI module from Part II. Develop the GUI module exclusively in this branch to keep it separate from the main codebase.
- Once development of the GUI module is complete, merge the `'gui-development'` branch with your `main` branch.
- During the merge process, carefully resolve any merge conflicts that arise.

Your submission

Your submission should be made through the OOP QM+ upload area designated for the Project. Ensure your code is written in a manner that is IDE-agnostic, allowing it to be compiled and executed using command-line tools without depending on specific IDE functionalities. The textual version of your Horse Race simulator should be invoked by calling the `startRace` method located in the `Part1` folder. For the graphical version, use the method named `startRaceGUI` to initiate the race.

Your submission should include the following:

- Your report as a PDF document, named `Report.pdf`, containing your responses to Part I. This report should be well-structured, organised, and clearly present your findings, analyses, and conclusions.
- The zipped working directory (`HorseRaceSimulator`) from which you initialised the local Git repository. This directory should contain your entire project, **including the `.git` folder**.

Assessment Criteria

Part I

Horse

- Horse Class Definition: This evaluates the basic definition of the Horse class to meet the specification, including adherence to standard Java style and conventions. (30%)
- Encapsulation: Assessing the implementation and understanding of encapsulation to protect the data within the Horse class. (5%)
- Testing: Demonstrating evidence of thorough testing to ensure the Horse class functions correctly, including a description of the testing process. (5%)

Race

- Class Improvements: Evaluating the enhancements made to the Race class, such as displaying the winner, and providing evidence of testing to validate these improvements. (20%)

Part II

1. **User Interface Design (5%)**:
    - Intuitive layout: Is the GUI well-organised and easy to navigate?
    - Use of appropriate controls: Are buttons, menus, and other elements used effectively to facilitate user interaction?
    - Consistency: Is the design consistent throughout the application, with a unified theme and style?
2. **Visual Appeal (5%)**:
    - Aesthetic appeal: Does the GUI have an attractive appearance, with visually pleasing elements and colour schemes?

- Attention to detail: Are visual elements such as icons, fonts, and graphics chosen thoughtfully to enhance the overall look and feel?

3. **Functionality (5%)**:
    - Interactivity: Does the GUI respond appropriately to user input, providing feedback and updates as needed?
    - Error handling: Are error messages displayed clearly and appropriately when input validation fails, or unexpected errors occur?
    - Feature completeness: Does the GUI include all necessary features and functionalities, as specified in the requirements?
4. **Innovation and Creativity (5%)**:
    - **Novelty**: Does the GUI introduce innovative features or design elements?
    - **Creativity**: Are there creative solutions or approaches implemented in the GUI design?

Part III

1. **Documentation Quality (5%):** Assess the quality and comprehensiveness of the README file included in the repository. Check if the README file provides clear instructions on running the code, including setup steps, dependencies, and usage guidelines. Evaluate the clarity of the instructions and how well they guide potential users through the process.
2. **Git Workflow and Branching (15%):** Evaluate the use of Git workflow and branching strategies. Check if used descriptive and meaningful commit messages for each change made to the codebase. Assess the branching strategy, particularly focusing on the creation of the `gui-development` branch for implementing the GUI module. Ensure that the GUI module development was carried out exclusively in this branch and merged correctly into the main branch once completed.

For each criterion your work will be evaluated as:

Judgement	***Mark***	***Description***
Not attempted	0	No attempt was made.
Fail	1 - 3	Attempt made, but significant errors or omissions indicate insufficient understanding of the course material.
Basic	4 – 5	Overall correct work, demonstrating a basic understanding of the course material. May contain minor errors.
Good	6 – 7	Work reflects a deep understanding of the course material, accompanied by independent thinking and application. Errors, if any, are minor and infrequent.
Excellent	8 – 9	Work demonstrates a thorough understanding of the course material, complemented by independent research and critical thinking. Errors are minimal or non-existent.
Exceptional	10	Work surpasses the standard expected at first-year undergraduate level, showcasing outstanding comprehension, analysis, and originality.