Object-Oriented Programming – Exercises Week 9

This week, we will be programming regular Java. Our goal is to create a Rally Championship Management System. Your system will manage rally drivers, cars, and races in a championship. The system should allow registering drivers, recording race results, and calculating championship standings.

The learning goal for the exercises

1. SOLID principles
2. Static data members and methods
3. Singleton pattern

On returning the exercise to Moodle

This time, the exercises must be submitted to Moodle for peer review, not to CodeGrade for automatic evaluation.

You must submit two things:

1) Link to a public GitHub (or any other public Git) repository containing your solution code
2) Link to a video, where you explain your code and demonstrate your solution

Both of these must be submitted. No late submissions are accepted. You are not allowed to change your Git code after the submission. Having commits after the deadline leads to automatic 0 points from the exercises.

Points Received

3 points for proper implementation (minus points for missing something or unclear explanations)

2 points for evaluating two other solutions

Requirements

1. Driver Class

- Create a `Driver` class with properties for name, country, and total points
- Implement appropriate getters and setters
- Include a method to add points to a driver

2. Car Class Hierarchy (Demonstrating Single Responsibility and Open/Closed Principles)

- Create an abstract `RallyCar` class with basic properties (make, model, horsepower)
- Create two concrete subclasses: `GravelCar` and `AsphaltCar` that extend `RallyCar`
- Each subclass should implement a specific method `calculatePerformance()` that returns a performance rating based on different algorithms

3. ChampionshipManager (Demonstrating Singleton Pattern)

- Implement `ChampionshipManager` as a Singleton class
- It should maintain lists of drivers and race results
- Include static methods for:
    - Getting championship standings
    - Finding the leading driver
    - Calculating total championship points

4. RaceResult Interface (Demonstrating Interface Segregation)

- Create a `RaceResult` interface that defines methods for recording and retrieving race results
- Implement a `RallyRaceResult` class that implements this interface

5. Static Utility Class

- Create a `ChampionshipStatistics` class with static methods for:
    - Calculating average points per driver
    - Finding the most successful country
    - Counting total races held

Implementation Guidelines

1. The Singleton pattern should ensure only one instance of `ChampionshipManager` exists
2. Use static data members to track total number of drivers and races

3. Apply dependency injection where appropriate (e.g., injecting car objects into drivers)
4. Follow the Liskov Substitution Principle by ensuring subclasses can be used in place of their parent class
5. Apply the Dependency Inversion Principle by having classes depend on abstractions rather than concrete implementations

Main Class

Create a `Main` class with a main method that:

1. Creates and configures the ChampionshipManager singleton
2. Create and register drivers with appropriate cars
3. Simulate at least two races with different surfaces
4. Display championship standings, statistics, and race results
5. Demonstrate car switching between races
6. Show performance calculations for different car types

Expected Output

Your code should produce output showing:

1. Championship standings with driver names, countries, and points
2. The current championship leader
3. Championship statistics including:
    - Total drivers registered
    - Total races held
    - Average points per driver
    - Most successful country
    - Total championship points
4. Detailed race results for each competition
5. Performance ratings for different car types

Example run

The results of the code could be like this (command line results):

```
1. Sébastien Ogier (France): 40 points
2. Kalle Rovanperä (Finland): 40 points
3. Ott Tänak (Estonia): 30 points
4. Thierry Neuville (Belgium): 30 points

===== CHAMPIONSHIP LEADER =====
Sébastien Ogier with 40 points

===== CHAMPIONSHIP STATISTICS =====
Total Drivers: 4
Total Races: 2
Average Points Per Driver: 35,00
Most Successful Country: Finland
Total Championship Points: 140

===== RACE RESULTS =====
Race: Rally Finland (Jyväskylä)
  Position 1: Sébastien Ogier - 25 points
  Position 2: Ott Tänak - 18 points
  Position 3: Kalle Rovanperä - 15 points
  Position 4: Thierry Neuville - 12 points

Race: Monte Carlo Rally (Monaco)
  Position 1: Kalle Rovanperä - 25 points
  Position 2: Thierry Neuville - 18 points
  Position 3: Sébastien Ogier - 15 points
  Position 4: Ott Tänak - 12 points


===== CAR PERFORMANCE RATINGS =====
Gravel Car Performance: 423.5
Asphalt Car Performance: 472.0
```

Implementation Constraints

- Maximum of 9 Java files
- Clear documentation with JavaDoc comments & video explanation
- Proper application of SOLID principles:
    - Single Responsibility Principle
    - Open/Closed Principle
    - Liskov Substitution Principle
    - Interface Segregation Principle
    - Dependency Inversion Principle
- Effective use of static members and methods
- Correct implementation of the Singleton pattern

Evaluation Criteria

Your solution will be evaluated depending on how well you have done the following:

1. Correct implementation of all required classes and interfaces
2. Proper application of OOP concepts and SOLID principles
3. Effective use of static members/methods
4. Correct implementation of the Singleton pattern
5. Code quality, readability, and documentation

The UML Diagram for the Rally Championship Management System