PRÁCTICA 2: Arquitectura de Software

Lydia Ruiz Martínez
Asignatura: Paradigmas y Técnicas de programación
Correo: 202213363@alu.comillas.edu
Repositorio de esta práctica: LydiaRuizMartinez/POO-Vehiculos

Principios SOLID

S - Single Responsibility Principle (SRP): Las clases como `City`, `PoliceCar`, y `SpeedRadar` respetan SRP en su mayoría, aunque `PoliceCar` asume la gestión de aparatos de medición, lo que introduce múltiples responsabilidades. Para corregirlo, se propone extraer esa lógica en una interfaz separada para dispositivos de medición.

O - Open/Closed Principle (OCP): Actualmente, modificar `PoliceCar` o `City` para agregar nuevos tipos de vehículos o dispositivos rompe el OCP. Para cumplirlo, se sugiere utilizar polimorfismo o el patrón estrategia, introduciendo interfaces para los dispositivos de medición y vehículos, facilitando la extensión sin alterar las clases existentes. Sin embargo si nos aseguramos de que solo aparecen estos vehículos, no habría ningún problema.

L - Liskov Substitution Principle (LSP): El código cumple con LSP al permitir que clases derivadas como `PoliceCar` y `Taxi` sustituyan correctamente a su clase base sin alterar el comportamiento del programa. Además, se ha introducido una abstracción adicional para los vehículos con matrícula, creando la clase `RegisteredVehicle`, de la cual heredan `PoliceCar` y `Taxi`. Por otro lado, el `Scooter` deriva directamente de la clase base `Vehicle`, que no incluye matrícula, lo que asegura que cada tipo de vehículo tenga el comportamiento adecuado sin romper LSP.

I - Interface Segregation Principle (ISP): Se ha empleado la interfaz `IMessageWriter` donde todos los métodos son necesarios.

D - Dependency Inversion Principle (DIP): Actualmente, `PoliceCar` depende directamente de `SpeedRadar`, y si la estación de policía o `City` dependen de vehículos específicos como `Taxi` y `PoliceCar`, sería difícil añadir nuevos tipos de vehículos en el futuro. Sin embargo, si se sabe con certeza que en esta ciudad solo existirán estos tipos de vehículos, esta dependencia fija no representaría un problema significativo.

Alcoholímetro:

Agregar nuevos aparatos de medición como el alcoholímetro rompería el OCP si se sigue modificando `PoliceCar` directamente. La solución es crear una interfaz genérica `IMeasuringDevice`, que permita añadir dispositivos sin modificar la clase del coche de policía.

Diagrama UML (modificado para cumplir SOLID):

`PoliceCar` dependería de `IMeasuringDevice`, que define el método `Activate()`. Clases como `SpeedRadar` y `Breathalyzer` implementarán esta interfaz. Con esta arquitectura, `PoliceCar` no necesita modificarse para añadir nuevos medidores. `PoliceCar` ahora depende de la abstracción `IMeasuringDevice`, reduciendo el acoplamiento.

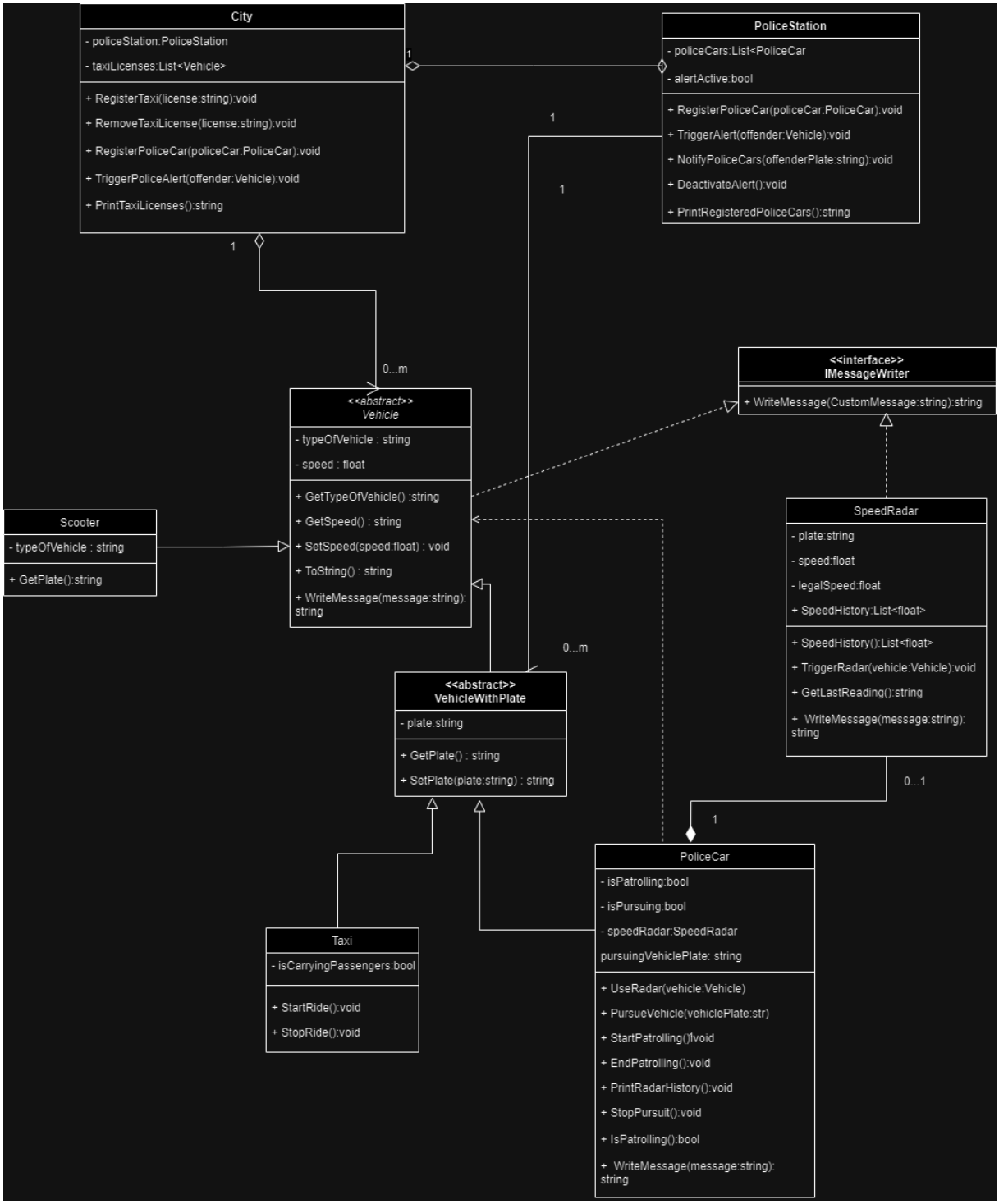

Figura 1: Diagrama UML