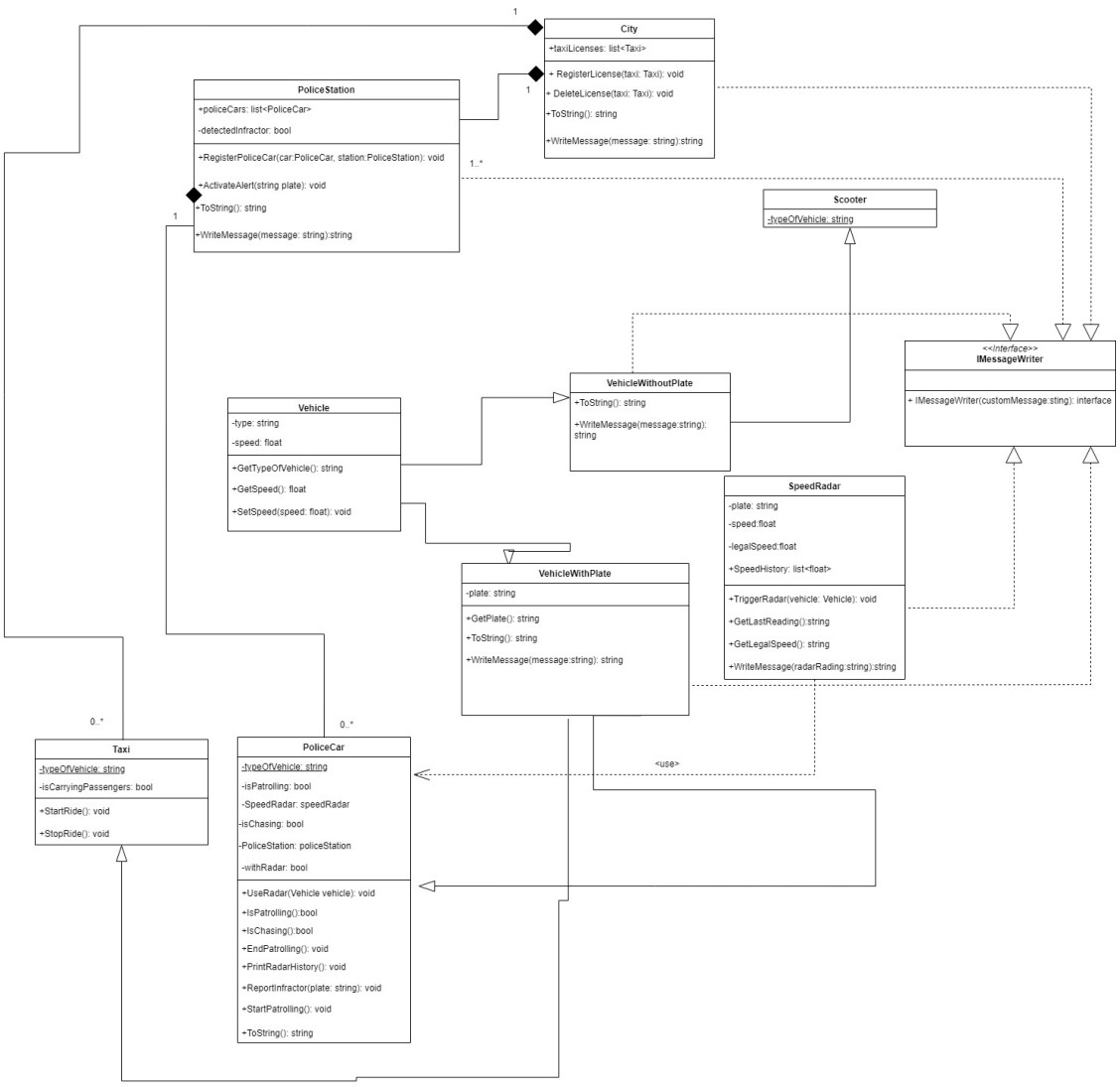
Memoria Práctica 2, Álvaro Fernández Gutiérrez: Paradigmas y técnicas de programación



Introducción

El diagrama UML representa un programa formado por distintos tipos de vehículos, una comisaría, un radar de velocidad y una ciudad (o ayuntamiento) encargada de gestionar la aprobación y retirada de las licencias de los taxis. A continuación, analizaremos si el diseño de este programa cumple los principios SOLID.

Evaluación de los principios SOLID

El principio de Responsabilidad Única (SRP) se cumple en su mayoría, ya que cada clase tiene una responsabilidad bien definida. La clase PoliceStation se encarga de registrar los coches de policía para posteriormente enviarles una alerta de persecución a un infractor, mientras que City gestiona las licencias de los taxis. Las clases de vehículos también están bien estructuradas, dividiéndose en aquellos con y sin matrícula. Aún así, hay que tener cuidado de no sobrecargar a la clase PoliceCar con demasiadas responsabilidades al combinar el patrullaje y el accionamiento del radar, aunque la creación de SpeedRadar como una clase independiente soluciona este problema.

El principio de Abierto/Cerrado (OCP) se respeta en gran medida, ya que el sistema, por ejemplo, permite agregar nuevos tipos de vehículos sin modificar el código existente. Por ejemplo, para esta práctica 2, hemos añadido la clase Scooter, y para ello hemos tenido que dividir los vehículos entre los que tienen matrícula y los que no. De esta forma, crear un nuevo vehículo en el futuro no afectaría a las clases base.

El principio de Sustitución de Liskov (LSP) también está presente. Tanto Taxi como PoliceCar pueden reemplazar a VehicleWithPlate. Además, el radar también respeta este principio, ya que puede usarse con cualquier vehículo que tenga matrícula, sin necesidad de conocer su tipo.

El principio de Segregación de Interfaces (ISP) se observa con claridad en la separación entre VehicleWithPlate y VehicleWithoutPlate, evitando que los vehículos sin matrícula deban tener atributos y métodos que no les pertenecen. También se aprecia en la interfaz IMessageWriter, que separa la funcionalidad de escritura de mensajes de la lógica principal de las clases.

Finalmente, el principio de Inversión de Dependencias (DIP) podría mejorarse. La comisaría (PoliceStation) interactúa directamente con PoliceCar, y el radar con vehículos específicos. Sería preferible desacoplar estas dependencias, lo que permitiría mayor flexibilidad en el sistema. De esta manera, por ejemplo, se podría cambiar la implementación del radar sin afectar la clase PoliceCar.

En la arquitectura de nuestro programa, la nueva funcionalidad de crear un alcoholímetro podría romper el principio de Abierto/Cerrado (OCP), que establece que las clases deben estar abiertas para la extensión, pero cerradas para la modificación. El radar está vinculado a la clase PoliceCar, lo que implica que para añadir un alcoholímetro tendríamos que modificar la clase PoliceCar y añadir condiciones para manejar ambos tipos de medidores. Esto implica que estaríamos modificando la clase cada vez que se añada un nuevo tipo de medidor, lo que va en contra del principio OCP, ya que la clase no está cerrada para modificaciones.