Practica 2: Arquitectura software

Enlace github: https://github.com/JorgeVanco/practica2_paradigmas_programacion.git

Principios SOLID en el código:

- SRP:
  En un principio el método StartPersecution de la clase PoliceCar también alertaba a la comisaría si no había sido alertado por ella el propio coche. Sin embargo, esto no cumple Single Responsibility Principle porque el método hace más cosas de las que dice su nombre.

```
2 referencias
public void StartPersecution(string plate, bool wasAlerted = false) {
    if (!isInPersecution || persecutionPlate != plate) {
        isInPersecution = true;
        persecutionPlate = plate;
        if (!wasAlerted) {
            Console.WriteLine(WriteMessage($"Sending alert for vehicle with plate: {persecutionPlate}"));
            policeStation.Alert(plate);
        }
        else {
            Console.WriteLine(WriteMessage($"Was alerted"));
        }
    }
}
```

  Esto has sido modificado a dos métodos, uno que es AlertPoliceStation, que manda la alerta a la estación de policías y StartPersecution, que simplemente hace que el coche entre en la persecución. Este último método es llamado por la comisaría.

```
1 referencia
public void AlertPoliceStation(string plate) {
    Console.WriteLine(WriteMessage($"Sending alert for vehicle with plate: {persecutionPlate}"));
    policeStation.Alert(plate);
}

1 referencia
public void StartPersecution(string plate) {
    isInPersecution = true;
    persecutionPlate = plate;
    Console.WriteLine(WriteMessage($"Started persecution on vehicle with plate {plate}"));
}
```

- OCP:
  Para cumplir este principio se ha tenido que tener especial cuidado en el método TriggerRadar de la clase SpeedRadar, pues con el nuevo Scooter, se tiene que tener en cuenta el caso en el que es un vehículo sin motor.

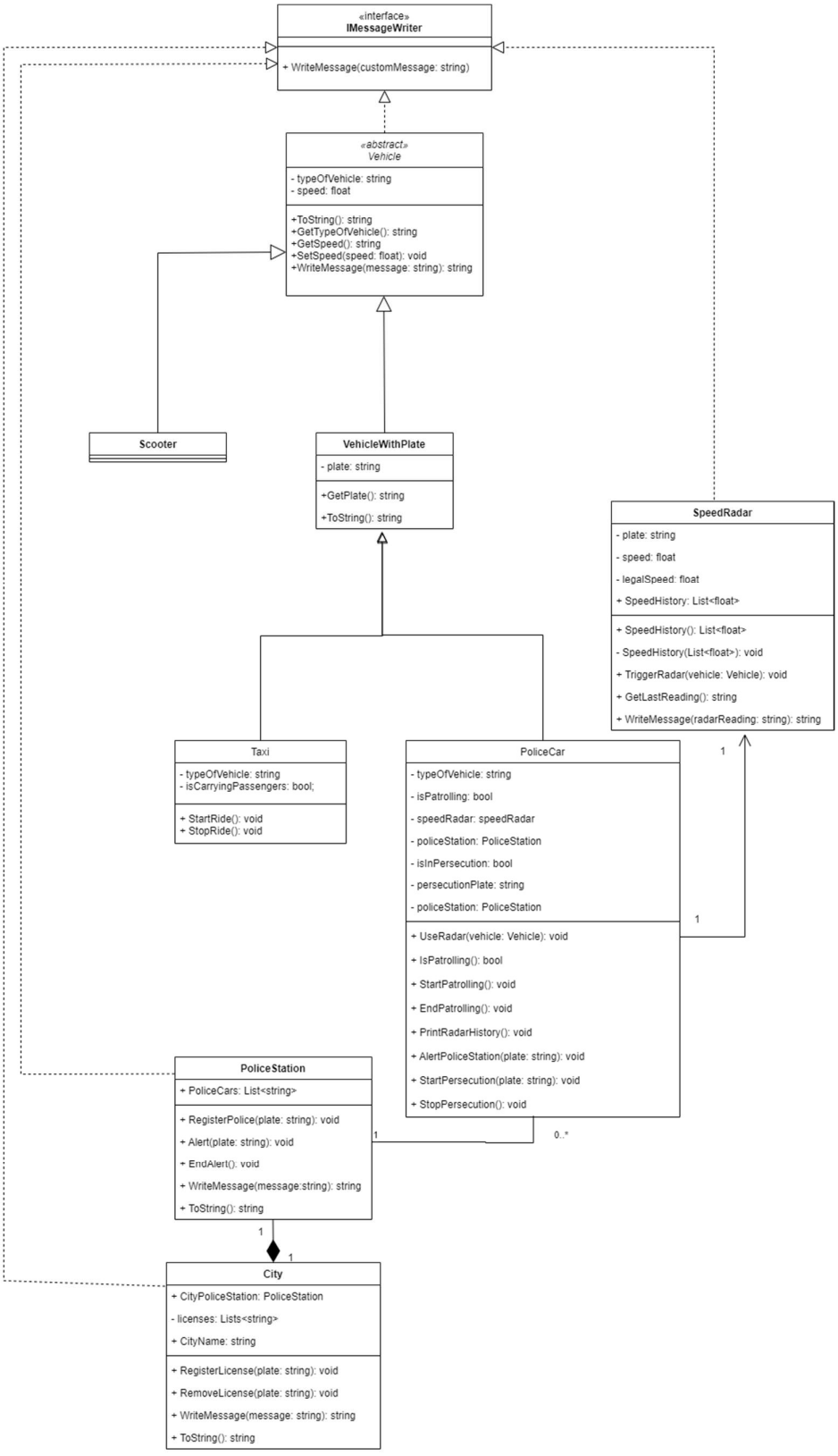
```
1 referencia
public void TriggerRadar(Vehicle vehicle)
{
    if (vehicle is VehicleWithPlate vehicleWithPlate) {
        plate = vehicleWithPlate.GetPlate();
    }else {
        plate = "";
    }
    speed = vehicle.GetSpeed();
    SpeedHistory.Add(speed);
}
```

- LSP:
  Se ha creado una clase VehicleWithPlate que hereda de Vehicle para diferenciar entre los vehículos con y sin matrícula cumpliendo el Liskov Substitution Principle.

- ISP:
  Este se cumple porque solo hay una interfaz con un método, todas las clases que la implementan necesitan el método.

- DIP:
  No hay ninguna clase que dependa de los requisitos de otra para su funcionalidad.

UML:


«interface»
IMessageWriter
+ WriteMessage(customMessage: string)
«abstract»
Vehicle
- typeOfVehicle: string
- speed: float
+ToString(): string
+GetTypeOfVehicle(): string
+GetSpeed(): string
+SetSpeed(speed: float): void
+WriteMessage(message: string): string
Scooter
VehicleWithPlate
- plate: string
+GetPlate(): string
+ToString(): string
SpeedRadar
- plate: string
- speed: float
- legalSpeed: float
+ SpeedHistory: List<float>
+ SpeedHistory(): List<float>
- SpeedHistory(List<float>): void
+ TriggerRadar(vehicle: Vehicle): void
+ GetLastReading(): string
+ WriteMessage(radarReading: string): string
Taxi
- typeOfVehicle: string
- isCarryingPassengers: bool;
+ StartRide(): void
+ StopRide(): void
PoliceCar
- typeOfVehicle: string
- isPatrolling: bool
- speedRadar: speedRadar
- policeStation: PoliceStation
- isInPersecution: bool
- persecutionPlate: string
- policeStation: PoliceStation
+ UseRadar(vehicle: Vehicle): void
+ IsPatrolling(): bool
+ StartPatrolling(): void
+ EndPatrolling(): void
+ PrintRadarHistory(): void
+ AlertPoliceStation(plate: string): void
+ StartPersecution(plate: string): void
+ StopPersecution(): void
PoliceStation
+ PoliceCars: List<string>
+ RegisterPolice(plate: string): void
+ Alert(plate: string): void
+ EndAlert(): void
+ WriteMessage(message:string): string
+ ToString(): string
City
+ CityPoliceStation: PoliceStation
- licenses: Lists<string>
+ CityName: string
+ RegisterLicense(plate: string): void
+ RemoveLicense(plate: string): void
+ WriteMessage(message: string): string
+ ToString(): string
1
1
1
1
1
0..*
1
1