

PRINCIPIOS SOLID

- **SINGLE RESPONSIBILITY PRINCIPLE (SRP)**

Este principio se cumple en el código de la práctica, pues, por ejemplo, la clase `Vehicle` contiene las funciones generales mínimas que comparten todos los vehículos y cada clase que hereda a `Vehicle` se encarga de su funcionalidad específica, teniendo cada clase solo sus atributos necesarios. Además, las funcionalidades de estas clases están divididas en labores atómicas y funciones más generales (como `UseRadar`) que combinan estas labores atómicas para hacer procesos más complejos.

- **OPEN/CLOSE PRINCIPLE (OCP)**

Se ha cumplido mediante hacer privadas todas las variables locales de cada clase y solo dando acceso a modo de getter a las propiedades que otras clases necesiten acceder. De esta forma, mediante la herencia de clases y mediante los getters se pueden amplificar y especificar la funcionalidad de las clases, pero los parámetros privados de cada instancia no son modificables.

- **LISKOV SUBSTITUTION PRINCIPLE (LSP)**

Se cumple ya que ninguna clase derivada modifica la clase base. Todas las clases derivadas de `Vehicle` especifican la funcionalidad del vehículo, heredando las propiedades base que comparten todos los vehículos, pero las funcionalidades comunes entre todos los tipos de vehículo se comparten. Además, se ha creado la clase intermedia `PlatedVehicle` para la gestión de vehículos sin matrícula sin necesidad de comprobar cada vez que si la matrícula es o no nula.

- **INTERFACE SEGREGATION PRINCIPLE (ISP)**

Se cumple, debido a que la única interfaz existente (`IMessageWriter`) solo consta de la función `WriteMessage`, la cual se usa en su totalidad cada vez que se invoca.

- **DEPENDENCY INVERSION PRINCIPLE (DIP)**

Se cumple debido a que gracias a la implementación de la clase intermedia `PlatedVehicle` se han eliminado las dependencias entre las clases.

PREGUNTA 7

Se incumpliría el Single Responsibility Principle. Para solucionarlo, se podría hacer una función general que compruebe qué tipo de medidor tenga y llame a su función correspondiente, y así hacer una función específica para cada medidor y poder controlar aparte la implementación de cada medidor.

```

classDiagram
    class IMessageWriter {
        <<interface>>
    }
    class Vehicle {
        typeOfVehicle: string
        speed: float
        ToString(): string
        GetTypeOfVehicle(): string
        GetSpeed(): float
        SetSpeed(int): void
        WriteMessage(string): string
    }
    class Scooter {
        typeOfVehicle: string
        StartRide(): void
        StopRide(): void
    }
    class PlatedVehicle {
        plate: string
        ToString(): string
        GetPlate(): string
    }
    class Taxi {
        typeOfVehicle: string
        isCarryingPassengers: bool
        StartRide(): void
        StopRide(): void
    }
    class City {
        policeStation: PoliceStation
        taxis: List<Taxi>
        PoliceStation: policeStation
        AddPoliceCar(string, bool): PoliceCar
        AddTaxi(string): Taxi
        RemoveTaxiLicense(): void
    }
    class PoliceStation {
        policeCars: List<PoliceCar>
        infractors: List<string>
        Infractors: List<string> infractors
        AddPoliceCar(PoliceCar): void
        StartAlarm(string): void
    }
    class PoliceCar {
        typeOfVehicle: string
        isPatrolling: bool
        speedRadar: SpeedRadar?
        infractor: string?
        chasing: bool
        UseRadar(vehicle, policeStation): void
        IsPatrolling(): bool
        StartPatrolling(): void
        EndPatrolling(): void
        ChaseInfractor(): void
        PrintRadarHistory(): void
    }
    class SpeedRadar {
        plate: string
        speed: float
        legalSpeed: float
        speedHistory: List<float>
        TriggerRadar(): float
        GetLastReading(): string
        WriteMessage(string): string
    }
    Vehicle <|-- Scooter
    Vehicle <|-- PlatedVehicle
    PlatedVehicle <|-- Taxi
    Vehicle ..|> IMessageWriter
    City "1" *-- "0..n" PoliceCar
    City "1" *-- "1" SpeedRadar
    PoliceStation "1" *-- "0..n" PoliceCar
    PoliceCar "1" *-- "1" SpeedRadar
    Scooter ..|> Vehicle : Extends
    PlatedVehicle ..|> Vehicle : Extends
    Taxi ..|> PlatedVehicle : Extends
    PoliceCar ..|> City : Extends
    SpeedRadar ..|> City : Extends
    
```

The diagram illustrates the following classes and their attributes:

- Vehicle**: typeOfVehicle: string, speed: float, ToString(): string, GetTypeOfVehicle(): string, GetSpeed(): float, SetSpeed(int): void, WriteMessage(string): string.
- Scooter**: typeOfVehicle: string, StartRide(): void, StopRide(): void.
- PlatedVehicle**: plate: string, ToString(): string, GetPlate(): string.
- Taxi**: typeOfVehicle: string, isCarryingPassengers: bool, StartRide(): void, StopRide(): void.
- City**: policeStation: PoliceStation, taxis: List<Taxi>, PoliceStation: policeStation, AddPoliceCar(string, bool): PoliceCar, AddTaxi(string): Taxi, RemoveTaxiLicense(): void.
- PoliceStation**: policeCars: List<PoliceCar>, infractors: List<string>, Infractors: List<string> infractors, AddPoliceCar(PoliceCar): void, StartAlarm(string): void.
- PoliceCar**: typeOfVehicle: string, isPatrolling: bool, speedRadar: SpeedRadar?, infractor: string?, chasing: bool, UseRadar(vehicle, policeStation): void, IsPatrolling(): bool, StartPatrolling(): void, EndPatrolling(): void, ChaseInfractor(): void, PrintRadarHistory(): void.
- SpeedRadar**: plate: string, speed: float, legalSpeed: float, speedHistory: List<float>, TriggerRadar(): float, GetLastReading(): string, WriteMessage(string): string.

Relationships:

- Extends**: Scooter extends Vehicle; PlatedVehicle extends Vehicle; Taxi extends PlatedVehicle; PoliceCar extends City; SpeedRadar extends City.
- Aggregation**: City has a 1-to-0..n relationship with PoliceCar and a 1-to-1 relationship with SpeedRadar. PoliceStation has a 1-to-0..n relationship with PoliceCar. PoliceCar has a 1-to-1 relationship with SpeedRadar.
- Generalization**: Vehicle is a generalization of IMessageWriter (indicated by a dashed line with an open arrowhead).