Modelarea Sistemelor Software

Student: Ciprian - Mihai Ceaușescu
Grupa: 506
Specializarea: Inginerie Software

Cuprins

- Problemă
- Diagrama de clase
- Diagrama de secvență
- Design Patterns
- Frameworks
- Demo
- Concluzii - Why UML?

Problemă

PROJECT #1

Proiectați o parcare pe următoarele ipoteze:

1. Parcarea are mai multe niveluri.
2. În parcare puteți parca motociclete, autoturisme și autobuze.
3. Parcarea are locuri destinate celor 3 categorii de vehicule.
4. O motocicleta poate parca în oricare dintre cele 3 locuri.
5. O mașină poate parca fie într-un loc de mașină, fie într-un singur loc de autobuz.
6. Un autobuz poate parca în cinci locuri destinate autobuzelor care sunt consecutive și pe același rând.

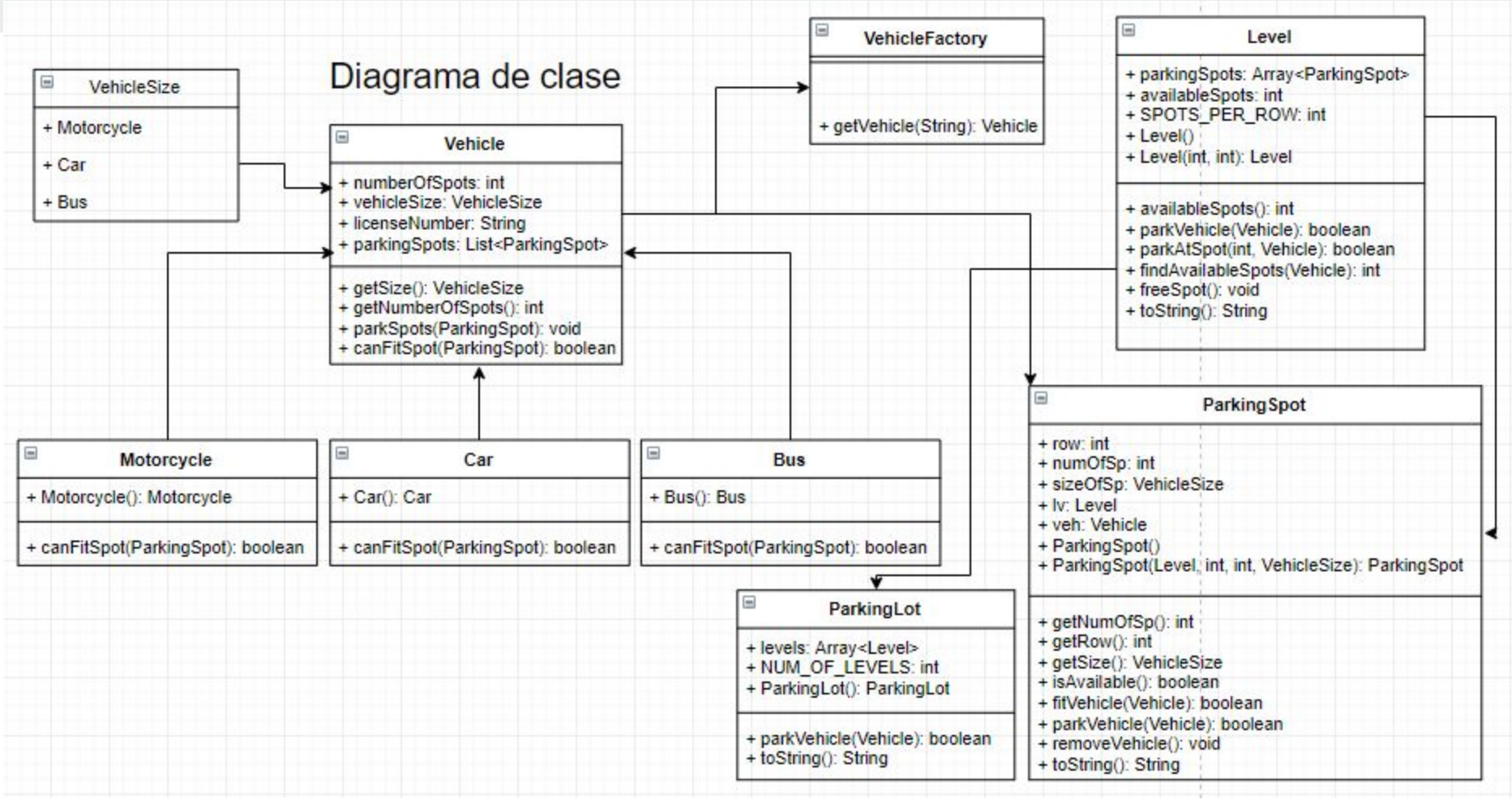
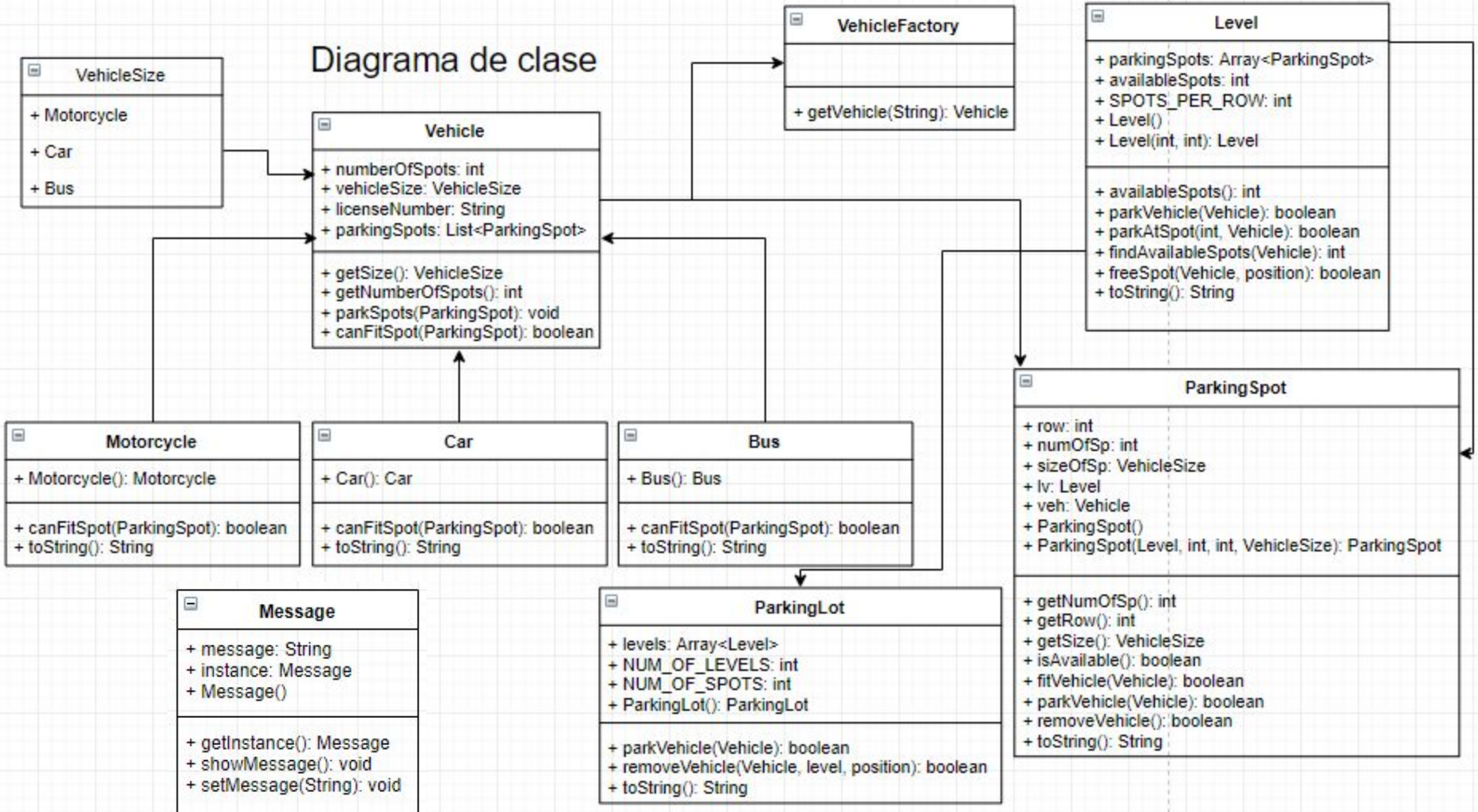


Diagrama de clase
VehicleSize
+ Motorcycle
+ Car
+ Bus
Vehicle
+ numberOfSpots: int
+ vehicleSize: VehicleSize
+ licenseNumber: String
+ parkingSpots: List<ParkingSpot>
+ getSize(): VehicleSize
+ getNumberOfSpots(): int
+ parkSpots(ParkingSpot): void
+ canFitSpot(ParkingSpot): boolean
VehicleFactory
+ getVehicle(String): Vehicle
Level
+ parkingSpots: Array<ParkingSpot>
+ availableSpots: int
+ SPOTS_PER_ROW: int
+ Level()
+ Level(int, int): Level
+ availableSpots(): int
+ parkVehicle(Vehicle): boolean
+ parkAtSpot(int, Vehicle): boolean
+ findAvailableSpots(Vehicle): int
+ freeSpot(): void
+ toString(): String
Motorcycle
+ Motorcycle(): Motorcycle
+ canFitSpot(ParkingSpot): boolean
Car
+ Car(): Car
+ canFitSpot(ParkingSpot): boolean
Bus
+ Bus(): Bus
+ canFitSpot(ParkingSpot): boolean
ParkingSpot
+ row: int
+ numOfSp: int
+ sizeOfSp: VehicleSize
+ lv: Level
+ veh: Vehicle
+ ParkingSpot()
+ ParkingSpot(Level, int, int, VehicleSize): ParkingSpot
+ getNumOfSp(): int
+ getRow(): int
+ getSize(): VehicleSize
+ isAvailable(): boolean
+ fitVehicle(Vehicle): boolean
+ parkVehicle(Vehicle): boolean
+ removeVehicle(): void
+ toString(): String
ParkingLot
+ levels: Array<Level>
+ NUM_OF_LEVELS: int
+ ParkingLot(): ParkingLot
+ parkVehicle(Vehicle): boolean
+ toString(): String

Diagrama de clase



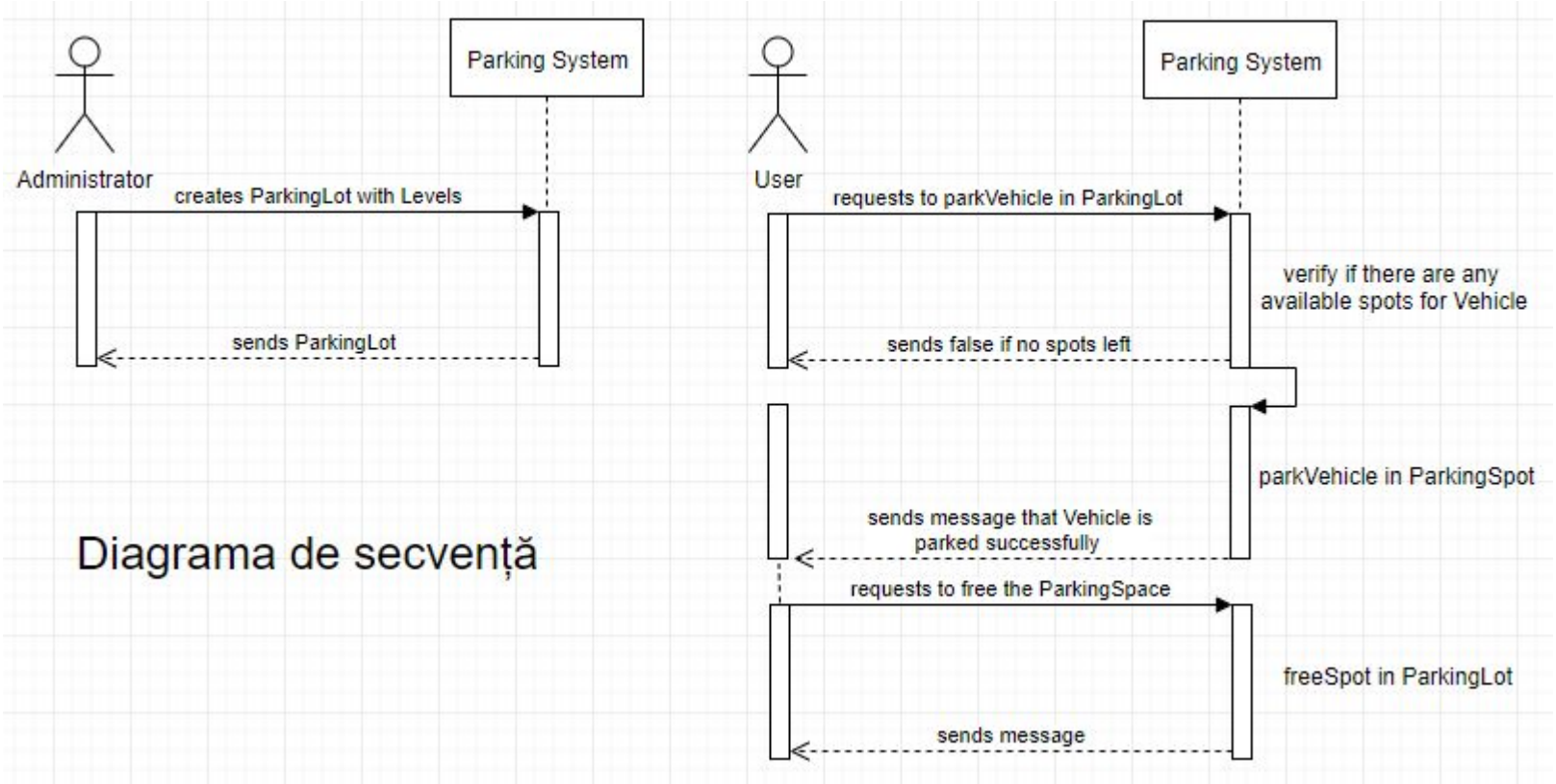
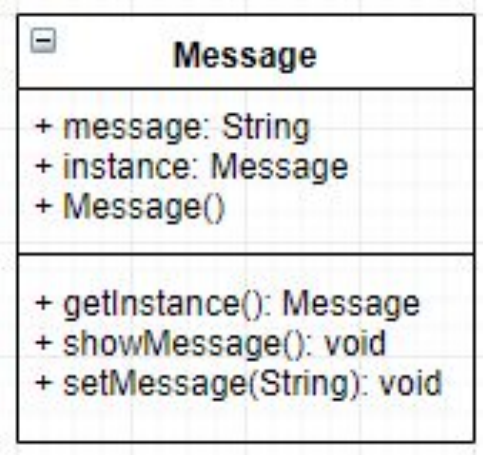


Diagrama de secvență

Design Patterns

Singleton Design Pattern - este un model de design software care restricționează instanțiarea unei clase la un singur obiect.

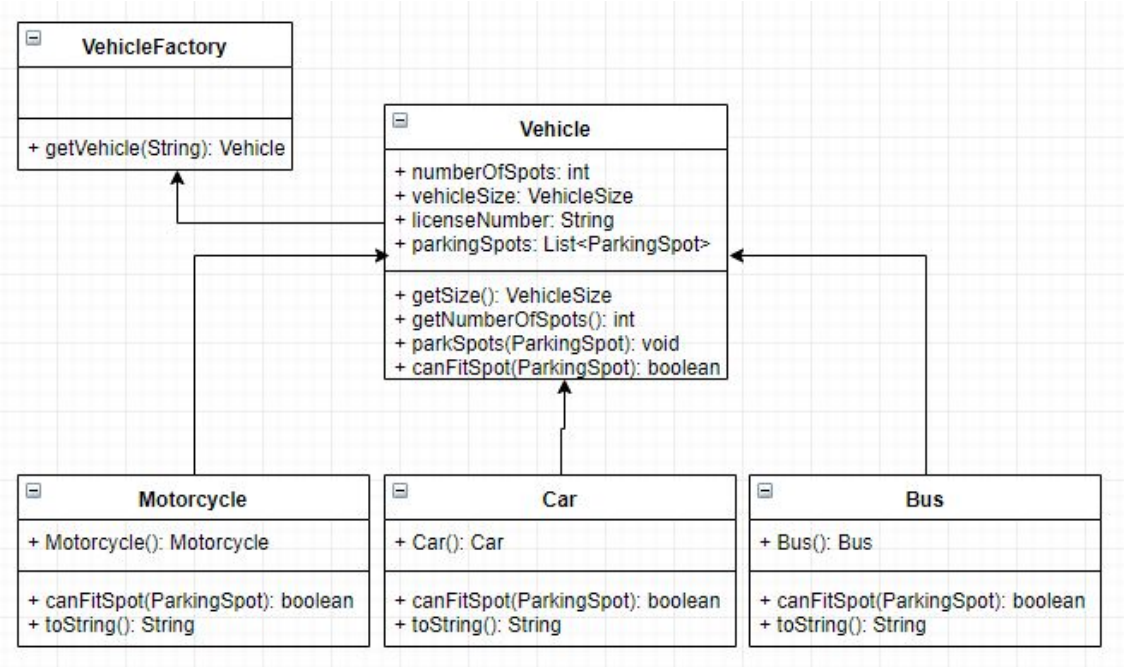
Am utilizat acest Design Pattern pentru afișarea către utilizator a informațiilor referitoare la acțiunile pe care acesta le poate face.



Design Patterns

Factory Design Pattern - este unul dintre cele mai utilizate modele de design, oferind o modalitate ușoară de a crea un obiect.

Instanțierea acestor obiecte, se realizează fără a expune logica utilizatorului, folosind o interfață comună.



Frameworks

Lombok - este un framework pentru limbajul Java care generează automat metode importante pentru diverse clase create, și anume:

- Getters
- Setters
- Constructors with no arguments
- Constructors with all arguments
- Builder design pattern

Frameworks

Student.java file:

```
import lombok.*;

@Setter
class Student {
    private String firstname;
    private String lastname;
    private String id;
    private int age;
    private String pin;
}
```

Frameworks

Student.class file:

```
class Student {
    private String firstname;
    private String lastname;
    private String id;
    private int age;
    private String pin;

    Student() {
    }

    public void setFirstname(String firstname) { this.firstname = firstname; }

    public void setLastname(String lastname) { this.lastname = lastname; }

    public void setId(String id) { this.id = id; }

    public void setAge(int age) { this.age = age; }

    public void setPin(String pin) { this.pin = pin; }
}
```

Demo

Proiectul realizat este încărcat pe GitHub:
https://github.com/cciprianmihai/-Modeling-of-Software-Systems.git

Concluzii - Why UML?

- Standard - un design foarte cunoscut.
- Flexibil - elemente de modelare pentru domeniul vizat.
- Modele portabile - XMI format - suportat de diverse tool-uri.
- Diferite tipuri de diagrame - se pot folosi doar o parte din ele.
- Arhitectura proiectului este importantă.

Concluzii - Why UML?

- Diagrama de clase a fost:
  - ❖ Utilă în momentul în care am realizat diagrama de secvență.
  - ❖ Utilă în structurarea proiectului.
  - ❖ Modificată în momentul în care structura proiectului s-a schimbat.
- Diagrama de secvență a fost:
  - ❖ Utilă pentru a urmări acțiunile pe care le poate face un utilizator în aplicație.
- Design pattern-urile utilizate:
  - ❖ Singleton - util pentru a avea o singură instanță a clasei care afișează mesajele în aplicație.
  - ❖ Factory - util pentru a crea instanțe ale claselor existente ușor.

Vă mulțumesc!