High-Quality Code Exam – Vehicle Park System

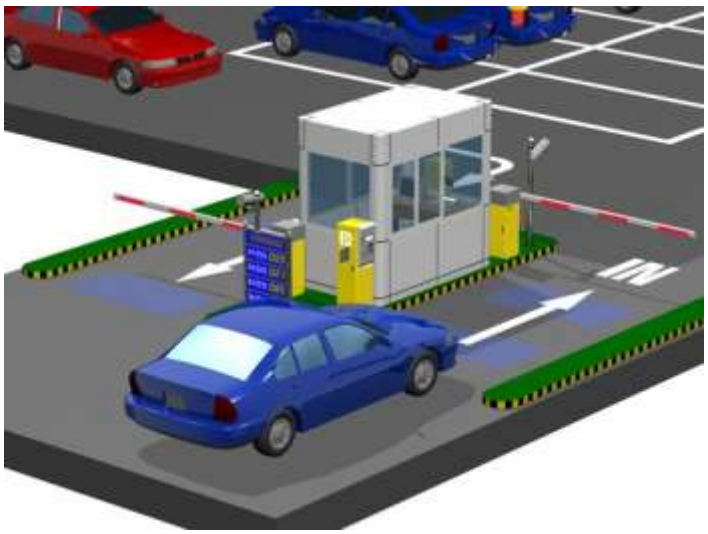
You have been assigned to work on an international project with a Brazilian company. The task is to implement a new vehicle parking system in C#. Your partner, **Himineu Casamenticio das Dores Conjugais** has done most of the work, but he doesn't have much experience in programming and was in a great hurry. That's why he wrote some really awful code. Now since you're the newest employee in your software company and you want to show how good your code-writing skills are, you have taken the task to refactor Himineu's code.

Your task is to **refactor the code**, using all best practices in **object-oriented design** and **object-oriented programming**, **SOLID** principles, and **design patterns**. You have to **improve the code quality** so it is easy to read and maintain. You also have to **fix any bugs** your Brazilian friend might have left, and **improve the general performance** (execution speed) of the code. Since the Brazilian company didn't have time to **write** any **unit tests**, they also left all of this to you.

You are given the original code and the design document, specifying the task at hand. The Brazilians also provided you with two cases to check the application. These documents are provided below.

Overview

A **vehicle park** consists of several **parking sectors**. Each parking sector has some **parking places**. The vehicle park has an **automated system**. It tracks the vehicles inside the park space, and provides an automated interface for **issuing parking tickets** and **charging**.



The vehicle park has two barriers. When a vehicle passes through the **entrance barrier**, a worker fills in its **type** (the park accepts **cars**, **motorbikes**, and **trucks**) and gives it **a sector** and **place** to park. A camera reads the vehicle's **license plate**, then registers the current time. The driver then tells the worker **how long they are planning to stay in the park**. With all of this information ready, the barrier opens and the vehicle is free to go to its assigned place.

When the vehicle is ready to exit the park and passes through the **exit barrier**, a similar procedure follows. Another camera reads the car's license number again, and registers the current time. Then, the automated system generates a parking ticket for the driver to pay.

A ticket price is calculated as follows: Each type of vehicle has a fixed hourly rate. The number of hours that the vehicle has spent in the park, multiplied by the hourly rate, gives the final price. However, there is one more detail: if

the car spends more time than planned initially, the driver has to pay an overtime rate (which is again fixed for each type of vehicle).

The rates for all types of vehicles have been given in the following table:

	Regular rate	**Overtime Rate**
Car	$2.00/h	$3.50/h
Motorbike	$1.35/h	$3.00/h
Truck	$4.75/h	$6.20/h

Here are some ticket price examples:

1. If a car driver declares they will stay in the park for four hours, but leaves after three hours, the rate is calculated as follows:
   Car price ($2.00/h) *** number of hours** (4) **+ overtime rate** ($3.50/h) *** number of overtime hours** (0) = $8.00
2. If a bike driver declares they will stay in the park for two hours, but leaves after four hours, the rate is calculated as follows:
   Bike price ($1.35/h) *** number of hours** (2) **+ overtime rate** ($3.00/h) *** number of overtime hours** (2) = $8.70

Note that if the driver reserved a parking place for four hours and stayed three hours only, they will be charged for four hours of use. That is, **if the driver stays too much, they are charged for overtime, but if they stay too little, they are not charged less**.

After the system issues the ticket, the driver pays for it, and the automated system prints how much change should be given back to the driver (for example, if the car driver from example 1 gave $10.00, the change would be $4.00).

The vehicle park system also provides tools for monitoring its current status. At any time, a worker can execute one of the three queries:

1. Check the **park status**. The system generates a report which says how many full spaces there are in each park sector. For example, a park has two sectors with four spaces each. There are two vehicles in the first sector, and one in the second sector. The generated report would be:
   ```
   Sector 1: 2 / 4 (50% full)
   Sector 2: 1 / 4 (25% full)
   ```
2. Find a vehicle **by its license plate number**. The system prints **some information** about the vehicle (type, license plate number, and owner) **and provides its location** within the parking lot. A sample report for a vehicle with license plate CA1001HH, parked in the second space of the first parking sector is given below:
   ```
   Car [CA1001HH], owned by Jay Margareta
   Parked at (1,2)
   ```
3. Find all currently parked vehicles **by their owner**. For each vehicle, the system prints **some information** about the vehicle (type, license plate number, and owner) **and provides its location** within the parking lot. A sample report for owner Jay Margareta who has two vehicles is given below:
   ```
   Car [CA1111HH], owned by Jay Margareta
   Parked at (1,2)
   Truck [CA5899AH], owned by Jay Margareta
   Parked at (3,2)
   ```

System design and functions

The main part of the system is **the engine**. It provides the interface (connection) between the parking lot employees and the automated system. **The engine ignores all whitespace around commands and all empty lines.** The engine catches any errors that might occur in the code and writes their messages back to the user.

The engine passes commands to a **command executor**. It executes all given commands. A command consists of **name** and **parameters**. The parameters are given as a JSON string.

A sample command is shown below:

```
CommandName {"key1": "value1", "key2": "value2", ...}
```

In case of a **correct command**, the execution engine returns a string. It contains either a success message (if everything went as expected), or an error message (if there was any problem executing the command). In case of **incorrect command**, the engine throws an **InvalidOperationException** with the message "Invalid command".

The execution engine delegates all commands to **an object which contains all information about the vehicle park** (which is an implementation of the **IVehiclePark** interface).

The vehicle park object contains a **layout**. The layout specifies how many sectors and how many parking places per sectors there are in the parking lot. It also contains a **database** which helps maintain all data operations in the park.

All the supported vehicles are **cars**, **motorbikes**, and **trucks**. Each vehicle has **license plate number** (to understand how to check if a license plate number is valid, look near the end of this document), **owner name**, and **reserved hours** (the time the driver has decided to stay in the park and pay for). Each type of vehicle has a **regular rate** and **overtime rate**. To see the values for the rates, look at the table above.

The commands supported by the automated vehicle park system are:

- **`SetupPark {"sectors": number, "placesPerSector": number}`**
  Sets up the park with the specified number of sectors and number of places per sector. This should always be the first command.
  In case of success, the system prints **"Vehicle park created"**.
  If the number of sectors is not positive, the system prints **"The number of sectors must be positive"**.
  If the number of places per sector is not positive, the system prints **"The number of places per sector must be positive".**
  If another command is executed before **SetupPark**, the system prints **"The vehicle park has not been set up"**.
- **`Park {"type": string, "time": datetime_string, "sector": number, "place": number, "licensePlate": string, "owner": string, "hours": number}`**
  "type" is always one of **"car"**, **"motorbike"**, or **"truck"**. There is no need to check it explicitly. **"licensePlate"** must be a valid license plate number.
  In case of success, the system prints **"<type> parked successfully at place (<sector>,<place>)"** where **<type>** is one of **"Car"**, **"Motorbike"**, or **"Truck"**.
  If the sector does not exist (for example, searching for the fifth sector of a parking lot which only has two sectors), the system prints **"There is no sector <sector> in the park"**.
  If the place does not exist (for example, searching for the tenth place of a parking lot which only has two places per sector), the system prints **"There is no place <place> in sector <sector>"**
  If there is already a parked vehicle in the place, the system prints **"The place (<sector>,<place>) is**

occupied".
If there is already a vehicle with the provided license plate in the park, the system prints **"There is already a vehicle with license plate <license_plate> in the park"**. The license plate number is unique and this means something tries to enter the park twice without exiting first.

- **`Exit {"licensePlate": string, "time": datetime_string, "paid": number}`**
  The system calculates the number of hours the vehicle has stayed in the park. If needed, it rounds them up to the nearest hour (for example, 3:40 hours is rounded to 4:00 hours, and 3:10 is rounded to 3:00 hours). Then it calculates the price for the ticket. After paying the amount, the system prints the ticket.
  In case of success, the system prints a ticket in the following format:

```
**********
<type> [<license_plate>], owned by <owner>
at place (<sector>,<place>)
Rate: <rate>
Overtime rate: <overtime_rate>
---
Total: <total_amount>
Paid: <paid>
Change: <change>
**********
```

  Stars mark places where the printer must cut the ticket tape. **<type>** is one of **"Car"**, **"Motorbike"**, or **"Truck"**. **<rate>** and **<overtime_rate>** are calculated by the formula given in the Overview section. **<total_amount>** is the sum of the normal and overtime rate. **<change>** is the difference between the paid and total owed amount.
  If there is no vehicle with the given license plate, the system prints **"There is no vehicle with license plate <license_plate> in the park"**.
  Exit time will always be greater than or equal to entrance time. The amount paid will always be greater than or equal to the total owed amount. There is no need to check these things explicitly.
- **`Status {}`**
  Prints the current status of the parking lot. An example report is given below. The parking lot used has two sectors, and four spaces in each. There are two vehicles parked in the first sector, and one vehicle parked in the second sector.

```
Sector 1: 2 / 4 (50% full)
Sector 2: 1 / 4 (25% full)
```

- **`FindVehicle {"licensePlate": string}`**
  Tries to find a vehicle with the specified license plate number in the parking lot.
  In case of success, the system prints information about the vehicle in the following format:

```
<type> [<license_plate>], owned by <owner>
Parked at (<sector>,<place>)
```

  <type> is one of **"Car"**, **"Motorbike"**, or **"Truck"**.
  If there is no vehicle with the given license plate, the system prints **"There is no vehicle with license plate <license_plate> in the park"**.
- **`VehiclesByOwner {"owner": string}`**
  Lists all vehicles by the specified owner in the parking lot, ordered by arrival time (in ascending order) first, and by license plate number (in ascending order) next.
  In case of success, the system prints information about all found vehicles in the following format:

```
<type> [<license_plate>], owned by <owner>
```

`Parked at (<sector>,<place>)`
<type> is one of **"Car"**, **"Motorbike"**, or **"Truck"**.
If there are no vehicles by the specified owner, the system prints **"No vehicles by <owner>"**.

Model the system and all entities (vehicles, vehicle park, tickets, owners, etc.) using the best established practices in object-oriented design and object-oriented programming.

The input should be read from the console. It may contain up to 50 000 commands, so the park system must work as efficiently as possible. The output is written to the console. The input and output formats have been specified in the command descriptions.

Not all license plates are valid (the camera may miss something). A valid license plate consists of one or two letters (specifying the town), followed by four digits (the number), and two more letters (specifying the license plate series). Examples of valid license plates are "CA2564HH", "H2299AH", and "A2442KK". Examples of invalid license plates are "AAA22A", "CA0052245A", and "C22A".

All dates are in ISO-8601 format (for example, 2015-05-04T10:40:00.0000000 means 4th May 2015, 10:40 am). All decimal separators are points (`.`). All prices should be rounded to two places after the decimal separator, using US dollars for the units (for example, **`$5.00`**, or **`$12.`** You are given the original source code from Nashmat designed to solve the above problem. Your task is to refactor it to improve its quality, fix any bugs, write unit tests, write some documentation and fix the performance bottlenecks.

Sample Input 1

```
SetupPark {"sectors": 3, "placesPerSector": 5}

Status {}

Park {"type": "car", "time": "2015-05-04T10:30:00.0000000", "sector": 1, "place": 5,
"licensePlate": "CA1001HH", "owner": "Jay Margareta", "hours": 1}
Park {"type": "motorbike", "time": "2015-05-04T10:40:00.0000000", "sector": 2, "place": 3,
"licensePlate": "CA5555AH", "owner": "Guy Sheard", "hours": 2}
Park {"type": "truck", "time": "2015-05-04T10:45:00.0000000", "sector": 1, "place": 1,
"licensePlate": "C5842CH", "owner": "Jessie Raul", "hours": 1}

Status {}

FindVehicle {"licensePlate": "CA5555AH"}
FindVehicle {"licensePlate": "CA1001HH"}
FindVehicle {"licensePlate": "C5842CH"}

Park {"type": "car", "time": "2015-05-04T11:30:00.0000000", "sector": 1, "place": 2,
"licensePlate": "CA1111HH", "owner": "Jay Margareta", "hours": 1}
Park {"type": "truck", "time": "2015-06-04T10:30:00.0000000", "sector": 3, "place": 2,
"licensePlate": "CA5899AH", "owner": "Jay Margareta", "hours": 4}

VehiclesByOwner {"owner": "Jay Margareta"}

Exit {"licensePlate": "CA5555AH", "time": "2015-05-04T11:40:00.0000000", "paid": 100.00}
Exit {"licensePlate": "CA1001HH", "time": "2015-05-04T13:30:00.0000000", "paid": 40.00}
Exit {"licensePlate": "C5842CH", "time": "2015-05-04T11:40:00.0000000", "paid": 10.00}

VehiclesByOwner {"owner": "Jay Margareta"}

Status {}
```

Sample Output 1

```
Vehicle park created
Sector 1: 0 / 5 (0% full)
Sector 2: 0 / 5 (0% full)
Sector 3: 0 / 5 (0% full)
Car parked successfully at place (1,5)
Motorbike parked successfully at place (2,3)
Truck parked successfully at place (1,1)
Sector 1: 2 / 5 (40% full)
Sector 2: 1 / 5 (20% full)
Sector 3: 0 / 5 (0% full)
Motorbike [CA5555AH], owned by Guy Sheard
Parked at (2,3)
Car [CA1001HH], owned by Jay Margareta
Parked at (1,5)
Truck [C5842CH], owned by Jessie Raul
Parked at (1,1)
Car parked successfully at place (1,2)
Truck parked successfully at place (3,2)
Car [CA1001HH], owned by Jay Margareta
Parked at (1,5)
Car [CA1111HH], owned by Jay Margareta
Parked at (1,2)
Truck [CA5899AH], owned by Jay Margareta
Parked at (3,2)
********************
Motorbike [CA5555AH], owned by Guy Sheard
at place (2,3)
Rate: $2.70
Overtime rate: $0.00
--------------------
Total: $2.70
Paid: $100.00
Change: $97.30
********************
********************
Car [CA1001HH], owned by Jay Margareta
at place (1,5)
Rate: $2.00
Overtime rate: $7.00
--------------------
Total: $9.00
Paid: $40.00
Change: $31.00
********************
********************
Truck [C5842CH], owned by Jessie Raul
at place (1,1)
Rate: $4.75
Overtime rate: $0.00
--------------------
Total: $4.75
Paid: $10.00
Change: $5.25
********************
Car [CA1111HH], owned by Jay Margareta
Parked at (1,2)
Truck [CA5899AH], owned by Jay Margareta
Parked at (3,2)
Sector 1: 1 / 5 (20% full)
```

```
Sector 2: 0 / 5 (0% full)
Sector 3: 1 / 5 (20% full)
```

Sample Input 2

```
Status {}

SetupPark {"sectors": 3, "placesPerSector": 5}

Park {"type": "car", "time": "2015-05-04T10:30:00.0000000", "sector": 1, "place": 5,
"licensePlate": "CA1001HH", "owner": "Jay Margareta", "hours": 1}
Park {"type": "motorbike", "time": "2015-05-04T10:40:00.0000000", "sector": 1, "place":
5, "licensePlate": "CA5555AH", "owner": "Guy Sheard", "hours": 2}
Park {"type": "car", "time": "2015-06-04T10:30:00.0000000", "sector": 2, "place": 5,
"licensePlate": "CA1001HH", "owner": "Geena Dolly", "hours": 1}
Park {"type": "truck", "time": "2015-04-04T10:30:00.0000000", "sector": 2, "place": 1,
"licensePlate": "CA1001HH", "owner": "Michael Stephenson", "hours": 1}

Status {}

Exit {"licensePlate": "AA2233HH", "time": "2015-05-04T14:30:00.0000000", "paid": 40.00}
Exit {"licensePlate": "CA1001HH", "time": "2015-05-04T13:30:00.0000000", "paid": 40.00}
Exit {"licensePlate": "CA1001HH", "time": "2015-05-04T13:30:00.0000000", "paid": 50.00}

Status {}

VehiclesByOwner {"owner": "John Smith"}

VehiclesByOwner {"owner": "Jay Margareta"}
```

Sample Output 2

```
The vehicle park has not been set up
Vehicle park created
Car parked successfully at place (1,5)
The place (1,5) is occupied
There is already a vehicle with license plate CA1001HH in the park
There is already a vehicle with license plate CA1001HH in the park
Sector 1: 1 / 5 (20% full)
Sector 2: 0 / 5 (0% full)
Sector 3: 0 / 5 (0% full)
There is no vehicle with license plate AA2233HH in the park
********************
Car [CA1001HH], owned by Jay Margareta
at place (1,5)
Rate: $2.00
Overtime rate: $7.00
--------------------
Total: $9.00
Paid: $40.00
Change: $31.00
********************
There is no vehicle with license plate CA1001HH in the park
Sector 1: 0 / 5 (0% full)
Sector 2: 0 / 5 (0% full)
Sector 3: 0 / 5 (0% full)
No vehicles by John Smith
No vehicles by Jay Margareta
```

Problem 1. Code Refactoring

Refactor the source code to improve its quality following the best practices introduced in the course "High-Quality Code". You may refactor anything except the **IVehiclePark** interface, as long as it improves the code quality. You may create as many classes, interfaces, enumerations, structures, etc. as you wish.

34 score

Problem 2. StyleCop

Make StyleCop run without any errors on your code (ignore all documentation-related errors).

3 score

Problem 3. Bug Fixing

Debug the code and fix any bugs you find.

6 score

Problem 4. Code Documentation

Document the `IVehiclePark` interface declaration and all methods in it using C# XML documentation. Any other documentation is **not** required. Each documentation gives 0.75 score.

6 score

Problem 5. Unit Testing

Design and implement **unit tests for the following methods of the `IVehiclePark` interface:**

- One of **InsertCar(), InsertMotorbike() or InsertTruck()**, by your choice
- **ExitVehicle()**
- **GetStatus()**
- **FindVehiclesByOwner()**

Any other code is not required to be tested. The **code coverage** should be **at least 90% for the specified methods** (you do not need to cover the class that parses the input commands and prints the output). Be sure to test **all major execution scenarios** + all interesting **border cases** and **special cases**. Use Visual Studio Team Test (VSTT) and VS code coverage.

25 score

Problem 6. Performance Bottlenecks

Find any **performance bottlenecks** and briefly describe them with the following **comment in the code**:

```
// PERFORMANCE: <your description of why you think this is a performance bottleneck>
```

Fix the problems if possible (and leave the bottlenecks descriptions in addition to the fixes).

6 score

Problem 7. Dependency Injection

Try to decouple the implementation of the input and output commands from the console using dependency injection. Use the provided **`IUserInterface`** interface.

4 score

Problem 8. Correct Results in the Judge System

You are given an automated judge system to submit your solution. If your code is correct (all bugs are fixed) and runs fast enough (the performance bottlenecks are fixed), your solution will pass all the tests. The last 2 tests measure performance. The others measure correctness.

16 score

Bonus

Problem 9. Mocking

Test the **`FindVehicle`** and **`VehicleByOwner`** commands by introducing a **mock** of the data layer (the repository that stores data about vehicles). You'll need to use **Dependency Injection** within the class implementing **`IVehiclePark`**.

10 score