ISTANBUL MEDIPOL UNIVERSITY

OBJECT ORIENTED PROGRAMMING COURSE PROJECT REPORT

PROJECT NAME:

Parking Lot Management System

STUDENT : SEMANUR ÇAM

ID: 63210108

DEPARTMENT: IE

LECTURER: DR. AHMET KAPLAN

Project Overview

Project Information:

The Parking Management System is a Java-based application that is designed to make parking simple. It helps users to park vehicles, print tickets, calculate fees and keep track of which spots are open and which are occupied.

It aims to improve the control of parking lots by making it automated and more efficient, and easier for people to operate.

The Goals and Objectives:

- Making it simpler to use an easier-to-understand method in the parking lot.
- Reducing the manual mistakes by automatically automating the ticket issuing process and calculate charges.
- View the real-time availability of parking spaces to get the most benefit from the parking lot.

Potential Users or Audience:

The owners and managers of parking garages, drivers who often use parking lots, and companies that manages large parking lots (such as shopping malls and airports) are potential users or audiences.

Problem Description:

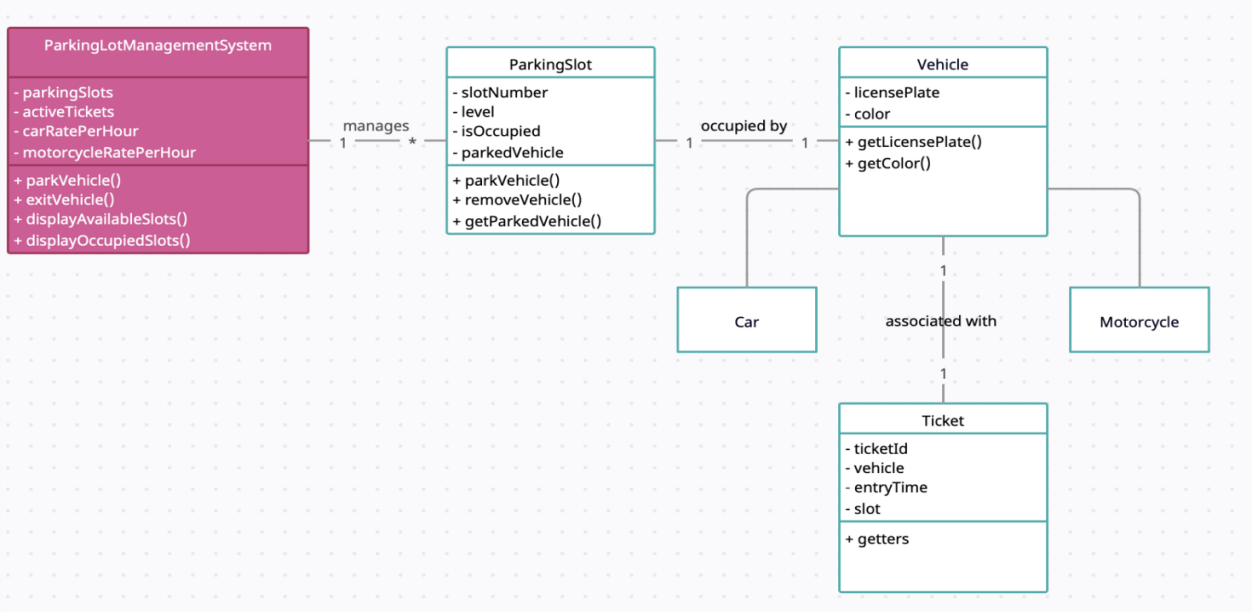
Manually management of parking spaces often results in inefficiency, consumer dissatisfaction resulting from inaccuracies and incorrectly filed registrations, erroneous fee assessment or lack of space.

Why It Is Important?

An automatic Car Parking Management System improves customer experience and reduces operational costs by supporting and delivering operational efficiency.

A reliable, precise and user-friendly platform for the management of parking facilities.

System Architecture:



Inheritance:

- Car and Motorcycle inherit from the abstract class Vehicle.

Associations:

- ParkingLotManagementSystem is associating with more than one ParkingSlot object (1 to Many).

- ParkingSlot is associates with one Vehicle (1 to 1).

- Ticket is associated with one ParkingSlot and one Vehicle (1 to 1).

Composition:

- ParkingLotManagementSystem has a composition relationship with ParkingSlot.

(Slots do not exist without the system).

Key Features :

Feature 1: Vehicle Parking and Ticket Management

This system helps to make parking and ticketing more efficient. Each car has a parking space and a UUID system creates a unique ticket ID to enable tracking. The system checks the availability of parking places before allowing car parking. After parking, the system stores the entry time, vehicle information and slot number in an active ticket database. This mechanism ensures smooth and precise parking operations in large multi-level complexes.

Ticket operations are automated, thus human error is reduced.

With this functionality, users can sell specific tickets for slots and track slot occupancy in real time. This can help regulate busy parking lots where cars and motorcycles are constantly entering and exiting. The system speeds up ticket purchase at the exit, speeding up departure.

Feature 2: Parking Fee Calculator

A very detailed parking fee calculation algorithm can be included. The fee is based on the type of vehicle (car or motorcycle) and the duration of parking. Cars are 3 dollars per hour and motorcycles are 1.50 dollars. The system calculates the parking time in minutes using arrival and departure timestamps and rounds up to the closest hour for the invoicing. It prevents over- and under-charging, ensuring fair and accurate fee charging. This feature is beneficial to parking facility owners and operators who want to maximize revenues. Auto-billing minimizes the administrative overheads and conflicts by eliminating tedious calculations. The fee based on vehicle type allows for transparency and helps parking lot management to keep track of finances.

Feature 3: Slot Availability Screen

The system shows current and occupied parking spaces. In real time, the users can slot all parking spaces by rank and by number. The license number, color and ticket IDs of parked vehicles are shown next to occupied spaces. In garages where it can be difficult to see empty spaces in a multi-story parking garage, this information gives consumers a place where they can quickly find a space.

This feature improves customer satisfaction by displaying real-time availability, making it easier to find open spaces and improving the user experience. It is additionally providing managers with a full view of slot availability, improving parking lot management. This functionality increases efficiency and customer satisfaction in high-traffic parking lots.

Feature 4: Multilevel Parking Support

Multi-tier multi-layered parking lot is implemented in the system. Each parking space has a level, so the users can choose their own level. This design accommodates several cars while keeping the parking space orderly. The slot managing system ensures that vehicles park in the spaces below, reduces parking lot clutter.

This multi-layered assistance is very important for the urban areas with limited space and vertical parking facility construction. The technology optimizes vehicle distribution by efficiently using all floors of the parking facility. It makes navigating simple by allowing users to see the free spaces on each floor and make a choice.

Feature 5: Vehicle Type Separation

The system distinguishes between automobiles and motorcycles. It is important because in parking facilities there may be zones or areas and different charges for different categories of vehicles. Since motorcycles are smaller, motorcycle charging rates are less than car charging rates. In this way, the parking system is able to accommodates a variety of users while maintaining equity and order. To maximize space utilization, this system allocates slots according to vehicle type. It is especially useful for parking facilities with limited space or different customers. The function helps to arrange the parking lot so that vehicles are parked according to their needs.

OOP Concepts Implemented

1. Classes and Objects

Core Classes in the Project:

The parking lot Management System is based on the core classes ParkingLot, ParkingSlot, Vehicle, Ticket and FeeCalculator.

- ParkingLot: It manages the overall parking structure, which includes the levels and slots.

- ParkingSlot: Represents any individual parking slots with characteristics such as slot ID, level and usability.

- Vehicle: Refers to a car with attributes such as type, license plate and color.

- Ticket: Handles ticket associated data such as time of entry, time of exit and associated vehicle.

- FeeCalculator: It is responsible for charging parking fees by vehicle type and duration.

The objects are instances of these classes to manage real-world entities, such as vehicles parked in a parking lot or tickets that are produced for users.

2. Encapsulation

The encapsulation is applied by using special characteristics and methodologies to guarantee data protection and proper access control. For example:

- In the ParkingSlot class, the availability attribute is private (__availability).

The public getter and setter methods (is_available() and set_availability(status)) can control this access to this attribute.

- In the Ticket class, the input and output time-stamps are private, and methods such as get_entry_time() and get_exit_time() have controlled access.

It is important to note that any sensitive data such as encapsulation, slot availability or ticket details cannot be directly accessible or manipulated without appropriate verification.

3. Inheritance

The project uses inheritance to create a hierarchy of related classes.

Example: The classes Car and Motorcycle both inherit from the base class Vehicle.

- The Vehicle class contains common attributes such as license plate, color and type.

- Car and Motorcycle extend the Vehicle class by adding specific methods or attributes, such as parking space size requirements.

This allows code to be reusable and reduces code redundancy, as common properties and methods are implemented in the base class.

4. Polymorphism

Polymorphism is represented by methods overriding and overloading:

- Method Override:

In the FeeCalculator class, the calculate_fee() method is overridden for different tool types. For example: Automobiles are charged $3 per hour. Motorcycles are charged at $1.5 per hour.

- Method Overloading (if the programming language supported):

In the ParkingSlot class, the assign_vehicle() method can be overloaded to accept different parameters such as vehicle type or vehicle object. Polymorphism improves project scalability by giving flexibility in terms of how methods are called and executed based on the object type.

5. Abstraction

Abstraction is applied by using either abstract types of classes or interfaces to define the blueprint of a specific functionality.

Example:

- The FeeCalculator class can be instantiated as an abstraction base class with the abstract method calculate_fee().

- The implementation of concrete classes for CarFeeCalculator and MotorcycleFeeCalculator supports im calculate_fee()method with a specific logic. By using the abstraction, the system permits the realization of base methods in derived classes, while keeping unnecessary details hidden from external users

Evaluation of OOP Concepts

1. Classes & Objects

The class structure is well-defined, with each class representing a real-world entity

or functionality. For instance, the Vehicle class models vehicle attributes, while

the Ticket class handles ticketing operations. These objects interact seamlessly

to represent the parking lot ecosystem.

2. Encapsulation

Getters and setters are used extensively to control access to private attributes. For

example:

• ParkingSlot uses private attributes

like __slot_id and __availability, ensuring data security.

• Controlled access prevents unauthorized modification of sensitive attributes.

3. Inheritance

The class hierarchy includes base and derived classes like Vehicle, Car,

and Motorcycle. This structure eliminates redundancy and promotes code reuse.

It also allows future extension, such as adding new vehicle types.

4. Polymorphism

Overriding the calculate_fee() method in FeeCalculator demonstrates

polymorphism effectively. Different implementations for cars and motorcycles

showcase how the system adapts behavior dynamically based on the object type.

5. Abstraction

Abstract classes like FeeCalculator define essential methods, ensuring a

consistent interface across different implementations. This provides a clear

structure for developers while hiding internal details from the end-user.

Overall, the implementation of OOP concepts ensures modularity, scalability, and

maintainability in the Parking Lot Management System.

Challenges Faced During the Project Implementation

Challenges During Development

System Design:

Designing a modular system with appropriate class relationships was initially challenging.

For example, establishing clear relationships between ParkingLot, ParkingSlot, and Vehicle required multiple iterations while avoiding circular dependencies. UML diagram.

Fee Calculation Logic:

It was complex to apply a flexible fee calculation mechanism that could adapt to different pricing models for various types of vehicles. Management of edge cases, such as partial hours or vehicles exceeding their time, required extra logic.

Concurrency Management:

Handling simultaneous parking lot assignments in a multi-threaded environment posed challenges. To ensure that two cars could not occupy the same slot, it was necessary to implement thread-safe mechanisms.

Overcoming Challenges

For System Design:

The use of iterative protocol prototyping has helped refine the UML diagrams. Peer feed-back and consulting online resources on Object-oriented design patterns has provided clarity on how to efficiently create class relationships.

For the Fee Calculation Logic:

A design strategy model for wage calculation has been applied. This allowed the dynamic allocation of wage calculation strategies (e.g. per hour, flat rate) to support different types of tools. Unit testing helped to validate edge cases.

For Concurrency Management:

A deadlock locking mechanism for slot allocations was introduced and only a single thread can modify the state of a slot at any time.

Code Generator Tools Used

The Eclipse IDE offers intelligent code suggestions, fast error management and detection, and built-in refactoring tools, clean and bug-free code.

PlantUML is especially useful for real-time updates to UML diagrams,

Ensure that any design changes are consistently reflected in the documentation.

Codding Datas:

```
=== Parking Lot Management System ===
1. Display Available Slots
2. Display Occupied Slots
3. Park Vehicle
4. Remove Vehicle
5. Exit
Choose an option: 1
Available Parking Slots:
Level: 1, Slot Number: 1
Level: 1, Slot Number: 2
Level: 1, Slot Number: 3
Level: 1, Slot Number: 4
Level: 1, Slot Number: 5
Level: 2, Slot Number: 6
Level: 2, Slot Number: 7
Level: 2, Slot Number: 8
Level: 2, Slot Number: 9
Level: 2, Slot Number: 10
Level: 3, Slot Number: 11
Level: 3, Slot Number: 12
Level: 3, Slot Number: 13
Level: 3, Slot Number: 14
Level: 3, Slot Number: 15

=== Parking Lot Management System ===
1. Display Available Slots
2. Display Occupied Slots
3. Park Vehicle
4. Remove Vehicle
5. Exit
Choose an option: 3
Enter Vehicle Type (1 for Car, 2 for Motorcycle): 1
Enter License Plate: 56ght789
Enter Color: blue
Enter Slot Number to Park: 7
Vehicle parked at Level 2, Slot 7. Ticket ID: 2e695ab7-ba48-406b-8cf5-1d59b4d49f6c

=== Parking Lot Management System ===
1. Display Available Slots
2. Display Occupied Slots
3. Park Vehicle
4. Remove Vehicle
5. Exit
Choose an option: 2
Occupied Parking Slots:
Level: 2, Slot Number: 7, Vehicle: 56ght789 (blue)

=== Parking Lot Management System ===
1. Display Available Slots
2. Display Occupied Slots
3. Park Vehicle
4. Remove Vehicle
5. Exit
Choose an option: 4
Enter Ticket ID to Remove Vehicle: 2e695ab7-ba48-406b-8cf5-1d59b4d49f6c
Vehicle with Ticket ID 2e695ab7-ba48-406b-8cf5-1d59b4d49f6c exited. Parking Fee: $3.0
```