Tic Tac Toe

Contents

[1. Overview 2](#_Toc188272458)

[2. Requirement Gathering 2](#_Toc188272459)

[Requirement from Visualization: 2](#_Toc188272460)

[Requirements from User Journey 3](#_Toc188272461)

[Clarify Requirements 3](#_Toc188272462)

[3. Use Case & Class Diagram 5](#_Toc188272463)

[Use Case Diagram 5](#_Toc188272464)

[Class Diagram 5](#_Toc188272465)

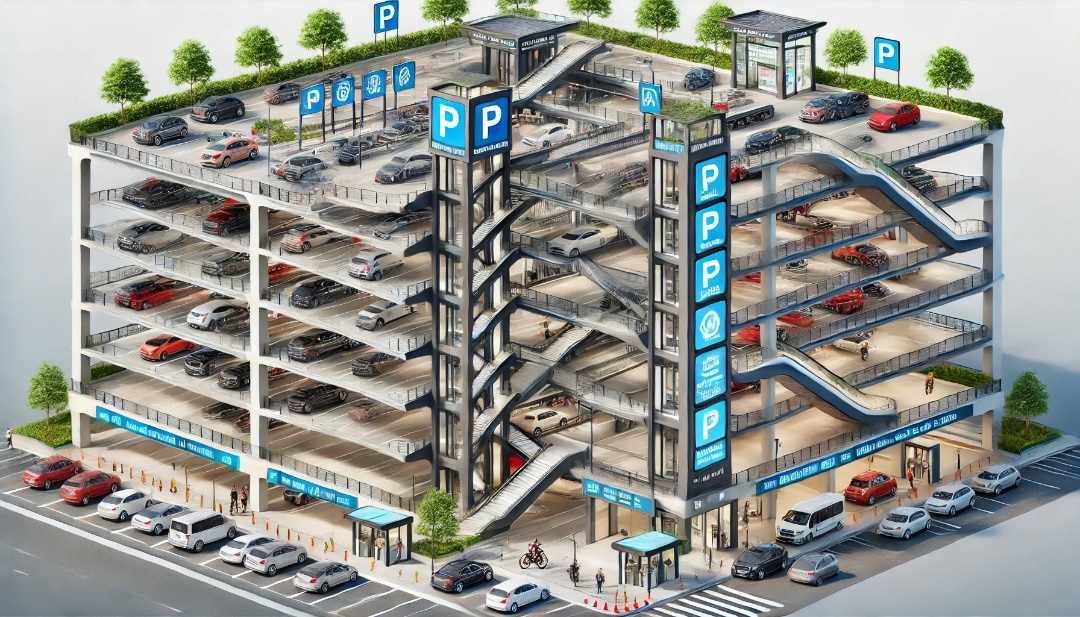
[Key Classes and Attributes 6](#_Toc188272466)

[Design Patterns Used 10](#_Toc188272467)

[Additional Notes 10](#_Toc188272468)

# Overview

* There are 2 different paths to get the overview of the system.
  + We know the system.
  + We don't know the system.
* Assume, we don't know the system.
* If we don't know the system, as questions. What kind of parking lot you are looking for?
  + In response to this question, we get...
    - Design a parking management system for public parking lots.
      * For example, Airport, Hospitals, Malls.
    - We need to design Multi-Level Car Parking.



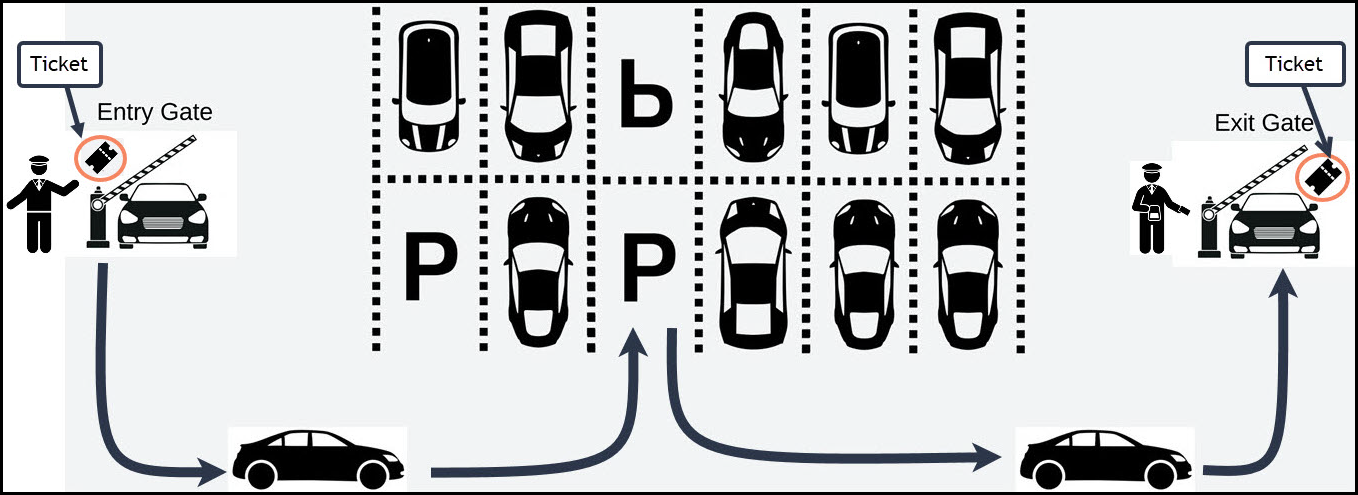
# Requirement Gathering

* Once we get the overview, let's talk about the requirements.
* If we remember, from the design point of view/class diagram point of view, we have to come from outside of the system to inside...

## Requirement from Visualization:

* Parking lot will have multiple floors.
* Multiple entry points.
* Multiple exit points.
* Multiple types of vehicles. (2-wheeler, 3-wheeler, cars, buses)
* Different types of parking spots for different types of vehicles.

## Requirements from User Journey

* At the entry gate, a ticket will be generated.
* Where the vehicle gets parked?
  + User will decide where to park the vehicle.
  + Operator will assign a parking spot at the entry gate.
* Spot assignment will happen at the entry gate.
* Store the operator details for entry gate and exit gate.
* Payment will happen at exit gate.
* User can make the payment via cash/UPI. For all online transactions, we will use Razor pay.
* Ticket should contain the vehicle information, entry time, entry gate details.

Note: Electric Vehicle parking spots are also supported.

## Clarify Requirements

* Parking Spot assignment can be done in multiple ways.
  + Near to lift lobby.
  + Near to exit gate.
  + Particular floor.
  + EV Parking Spot.
* We can use Strategy design pattern to assigning parking spot.
* Fee Calculation can be done in multiple ways.
  + Charges per hour

|  |  |  |
| --- | --- | --- |
| Hours | | Charge/Hour |
| Base Price | 0 to 4 | 50 |
|  | 5 to 10 | 1.2 X Base Price |
| 11 to 15 | 1.5 X Base Price |

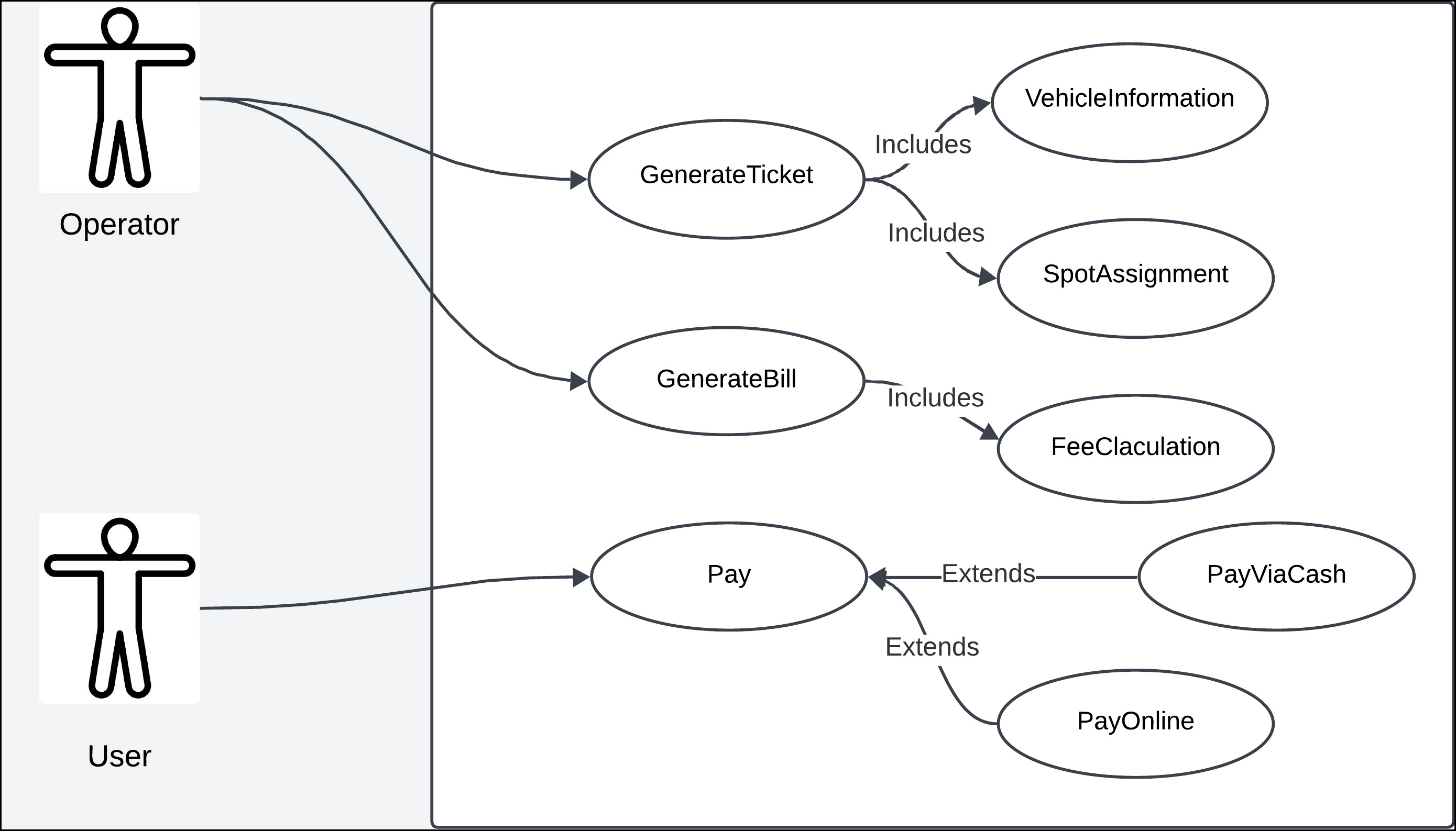
* + - Example: For 12 hours of parking…

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Hours | | Charge/Hour | For 12 Hours | |
| Base Price | 0 to 4 | 50 | 50 X 5 | 250 |
|  | 5 to 10 | 1.2 X Base Price | 1.2 X 5 X 5 | 300 |
|  | 11 to 15 | 1.5 X Base Price | 1.5 X 5 X 2 | 150 |
| Total | | | | 700 |

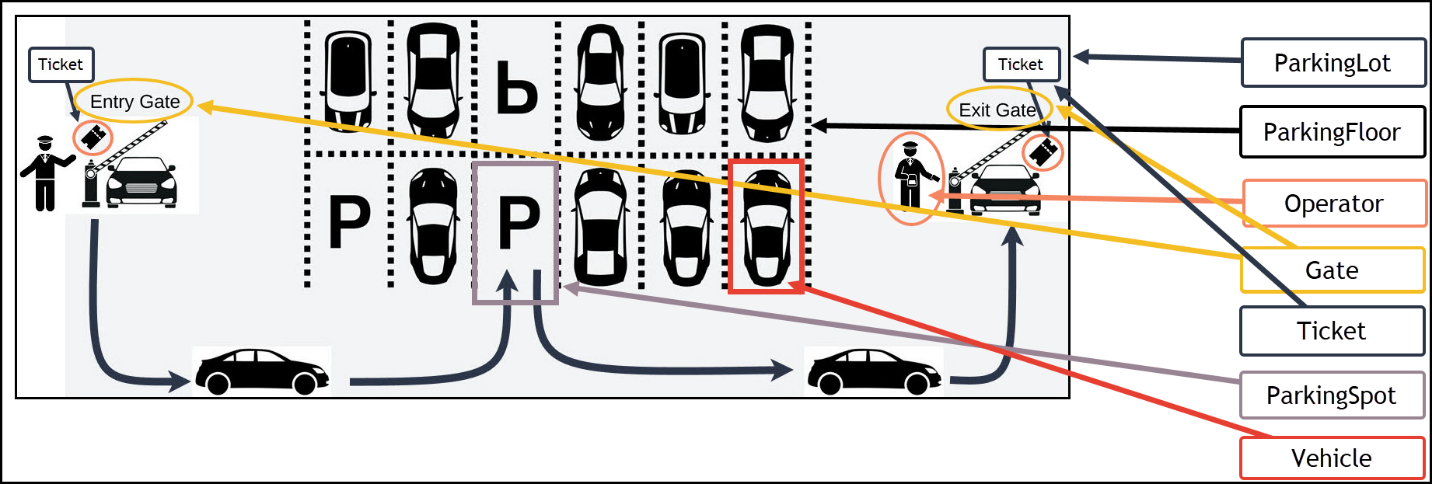
* + Flat charge/day
* We can use Strategy design pattern to calculate fee.
* Partial payment feature.
  + User can pay the fee from different modes of payment.
  + Example: If fee is 500Rs, 200 from wallet + 300 from UPI.
* For EV, if the vehicle is charged, then total fee = parking fee + electricity consumed.

# Use Case & Class Diagram

## Use Case Diagram



## Class Diagram

* To come up with class diagram,
  + Visualize the system.
  + Identify the nouns in the requirement.
  + Visualize the system
* Let’s visualize the system and list all the nouns first.
* Let’s come from Outside while identifying the nouns…
  + ParkingLot class
  + ParkingFloor class
  + ParkingSpot class
    - ParkingSpot is internal to a floor. So, ParkingFloor has list of ParkingSpot.
    - ParkingFloor is internal to parking lot. So, ParkingLot has list of ParkingFloor.
  + GateOperator class
  + Gate class
    - ParkingLot should have all the gates which belong to that parking lot. Hence, we need list of Gate.
    - ParkingLot might also store all gate operator details. Hence, we need list of GateOperator.
  + Ticket class
  + Vehicle class

## Key Classes and Attributes

### 1. **Parking Lot**

* Represents the overall system.
* Attributes:
  + ID
  + Address
  + Total Parking Spots
  + Available Parking Spots
* Functionality:
  + Manages entry and exit gates.
  + Tracks vehicle entries and exits.
  + Assigns parking spots.

### 2. **Gate**

* Represents an entry or exit point.
* Attributes:
  + Gate ID
  + GateType (Entry/Exit)
  + Operator (the person managing the gate)
* Entry Gate:
  + Assigns parking spots.
* Exit Gate:
  + Handles payments and exits.

enum class GateType

{

ENTRY\_GATE,

EXIT\_GATE,

};

### 3. **Operator**

* Manages operations at a gate.
* Attributes:
  + Operator ID
  + Name
  + Contact Information
* Functionality:
  + Issues tickets at entry.
  + Processes bills and payments at exit.

### 4. **Vehicle**

* Represents a vehicle entering the parking lot.
* Attributes:
  + Vehicle Number
  + VehicleType (e.g., Car, Bike, Truck)
  + Owner Information

enum class VehicleType

{

TWO\_WHEELER,

SEDAN,

SUV,

TRUCK,

ELECTRIC\_VEHICLE,

};

### 5. **Parking Spot**

* Represents an individual parking spot.
* Attributes:
  + Spot ID
  + Spot Type (e.g., Compact, Large, Handicapped, Electric)
  + ParkingSpotStatus
  + ChargingPointAvailability (for electric vehicles)
* Functionality:
* Assigned at the entry gate.

enum class ParkingSpotState {

FREE, // Spot is available.

OCCUPIED, // Spot is currently in use.

RESERVED, // Spot is reserved for a specific user/vehicle.

MAINTENANCE, // Spot is temporarily unavailable due to maintenance.

BLOCKED // Spot is blocked for external reasons.

};

### 6. **Ticket**

* Issued at the entry gate.
* Attributes:
  + Ticket ID
  + Vehicle Information
  + Parking Spot
  + Entry Time
  + Gate Information (Entry Gate)
* Functionality:
  + Tracks the vehicle's entry information.
  + Serves as the basis for bill generation.

### 7. **Bill**

* Generated at the exit gate.
* Attributes:
  + Bill ID
  + Ticket Information
  + Exit Time
  + Exit Gate
  + Amount
  + BillStatus (Paid, Unpaid, Partially Paid)
  + List of Payments
* Functionality:
  + Calculates the total amount based on parking duration.
  + Tracks payment details.

enum class BillStatus

{

PAID,

UNPAID,

REFUND,

PROCESSING,

};

### 8. **Payment**

* Represents individual payments.
* Attributes:
  + Payment ID
  + PaymentMode (e.g., Cash, Card, Net Banking)
  + Amount
  + PaymentStatus (Successful, Unsuccessful, Refunded)
  + Payment Time
* Functionality:
  + Supports partial payments.
  + Handles retries for failed payments.

enum class PaymentMode

{

CASH,

CREDIT\_CARD,

UPI,

NET\_BANKING,

};

enum class PaymentStatus

{

SUCCESSFUL,

FAILED,

INPROGRESS,

PARTIAL,

};

## Design Patterns Used

### 1. **Strategy Pattern**

* **Use Cases**:
  + Parking Spot Assignment: Different strategies to assign spots (e.g., closest available, random).
  + Fee Calculation: Different algorithms to calculate fees (e.g., flat rate, hourly rate).
  + Payment Processing: Handles multiple payment methods.

### 2. **Adapter Pattern**

* **Use Case**:
  + Payment Modes: Adapts various external payment APIs (e.g., for credit card, Amazon Pay).

### 3. **Prototype Pattern**

* **Use Case**:
  + Parking Spot: To create parking spots with similar attributes but differing details.

### 4. **Builder Pattern** (If Required)

* **Use Case**:
  + Could be used for constructing complex objects like tickets or bills.

## Additional Notes

* **Database Storage**:
  + Parking Tickets and Bills should be stored in a database for historical tracking and auditing.
* **Bill and Payment Logic**:
  + A bill is marked as successful only if all partial payments are completed successfully.
  + Failed or partially successful payments trigger retries.
* **Dynamic Enhancements**:
  + The design may evolve during implementation based on new requirements or edge cases.
* **Implementation Notes**:
  + Exit time and fees are calculated during billing, not stored in the ticket.
  + Payment and billing classes are separate to handle complex scenarios like refunds or partial payments.