# Task: Design a Parking Lot Management System

## Objective

Design and model a Parking Lot Management System using the concepts from The Object-Oriented Thought Process. Focus on demonstrating your understanding of encapsulation, inheritance, polymorphism, and abstraction.

## Your Assignment Details

### Part 1: Class Design

Design the following classes and their relationships:  
1. ParkingLot  
 - Represents the entire parking lot. Manages parking spaces and handles vehicle entry and exit.  
2. Vehicle  
 - Base class with attributes like license number and size (e.g., compact, regular, large).  
 - Derived classes: Motorcycle, Car, and Truck.  
3. ParkingSpace  
 - Represents a single parking space. Should specify the type of vehicle it can accommodate.  
4. Ticket  
 - Represents a parking ticket issued to a vehicle upon entry. Should track time of entry and calculate fees.

### Part 2: Diagrams

1. Class Diagram  
 - Create a UML diagram showing the relationships between the classes above. Use appropriate relationships (e.g., inheritance, composition).  
2. Sequence Diagram  
 - Create a sequence diagram illustrating the workflow for:  
 - A vehicle entering the parking lot.  
 - A vehicle exiting and the payment process.

### Part 3: Code Skeleton

Write a basic implementation of your design in a programming language of your choice. Focus on creating the structure of the classes and methods (without full implementations).  
  
Example methods to include:  
 - ParkingLot.ParkVehicle(vehicle)  
 - ParkingLot.RemoveVehicle(ticket)  
 - ParkingSpace.IsAvailable()  
 - Ticket.CalculateFee()

### Part 4: Explanation Document

Write a short explanation (1-2 pages) answering the following:  
1. How does your design use encapsulation to protect data?  
2. Where did you use inheritance to avoid code duplication?  
3. How did you apply polymorphism to make your code flexible?  
4. How did you simplify the complexity using abstraction?

## Evaluation Criteria

1. Class Design: Clarity of design and alignment with object-oriented principles.  
2. UML Diagrams: Accuracy and completeness of diagrams.  
3. Code Structure: Proper use of OOP concepts in the code skeleton.  
4. Explanation: Clear reasoning and justification for design decisions.

You will work individually on this task and submit all deliverables by [insert deadline here]. Let me know if you have any questions!