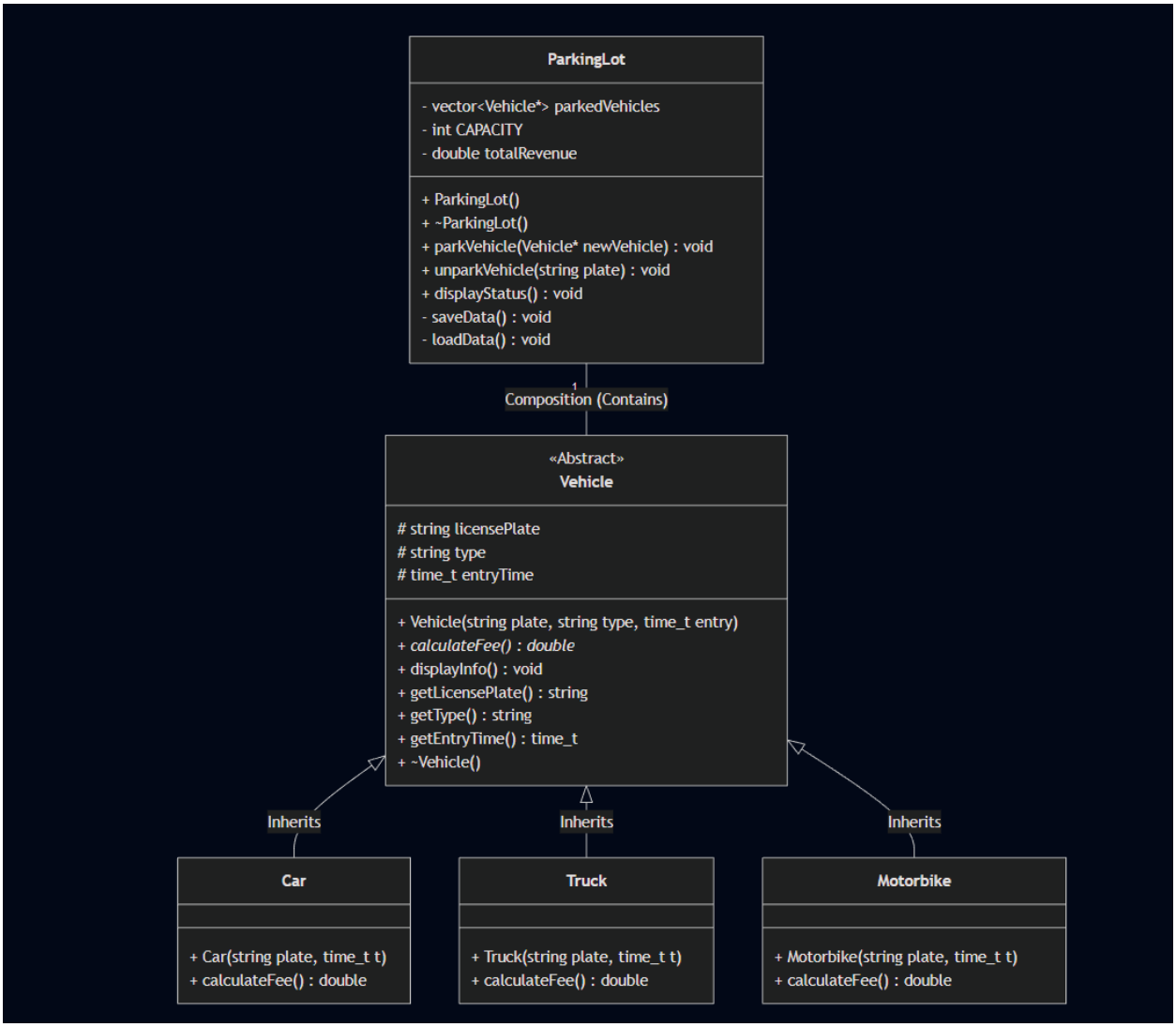
Course: Object-Oriented Programming (CSE)

Project: Parking Lot Simulation System

Student Name: Ali Bal **Date:** December 20, 2025

PART 1: UML Class Diagram



PART 2: Project Report (Functional Report)

1. Project Objective

The primary objective of this project is to design and implement a console-based **Parking Lot Simulation System** using C++. The project aims to demonstrate the practical application of core Object-Oriented Programming (OOP) principles, such as abstraction, encapsulation, inheritance, and polymorphism. The system manages vehicle entries and exits, calculates dynamic parking fees based on vehicle type, and ensures data persistence through file I/O operations .

2. System Architecture & Class Design

The system is built upon a modular architecture consisting of a base class, derived classes, and a manager class.

- **`Vehicle` (Abstract Base Class):** This class serves as a blueprint for all vehicle types. It holds common attributes like `licensePlate` and `entryTime`. The `calculateFee()` method is defined as a **pure virtual function**, making `Vehicle` an abstract class and enforcing derived classes to implement their specific fee logic.
- **`Car`, `Truck`, `Motorbike` (Derived Classes):** These classes inherit from `Vehicle`. They override the `calculateFee()` method to apply specific pricing rules (e.g., Trucks have a higher hourly rate than Motorbikes).
- **`ParkingLot` (Manager Class):** This class acts as the central controller. It uses the **Composition** principle, managing a collection of `Vehicle` objects. It handles the logic for parking (adding vehicles), unparking (calculating fees and removing vehicles), and displaying the current status.

3. Key OOP Concepts Implemented

The project successfully implements the following OOP pillars as required:

1. **Polymorphism:** The system uses a `std::vector<Vehicle*>` to store pointers to different vehicle objects. This allows the program to treat `Car`, `Truck`, and `Motorbike` objects uniformly while executing their specific `calculateFee()` and `displayInfo()` methods at runtime.

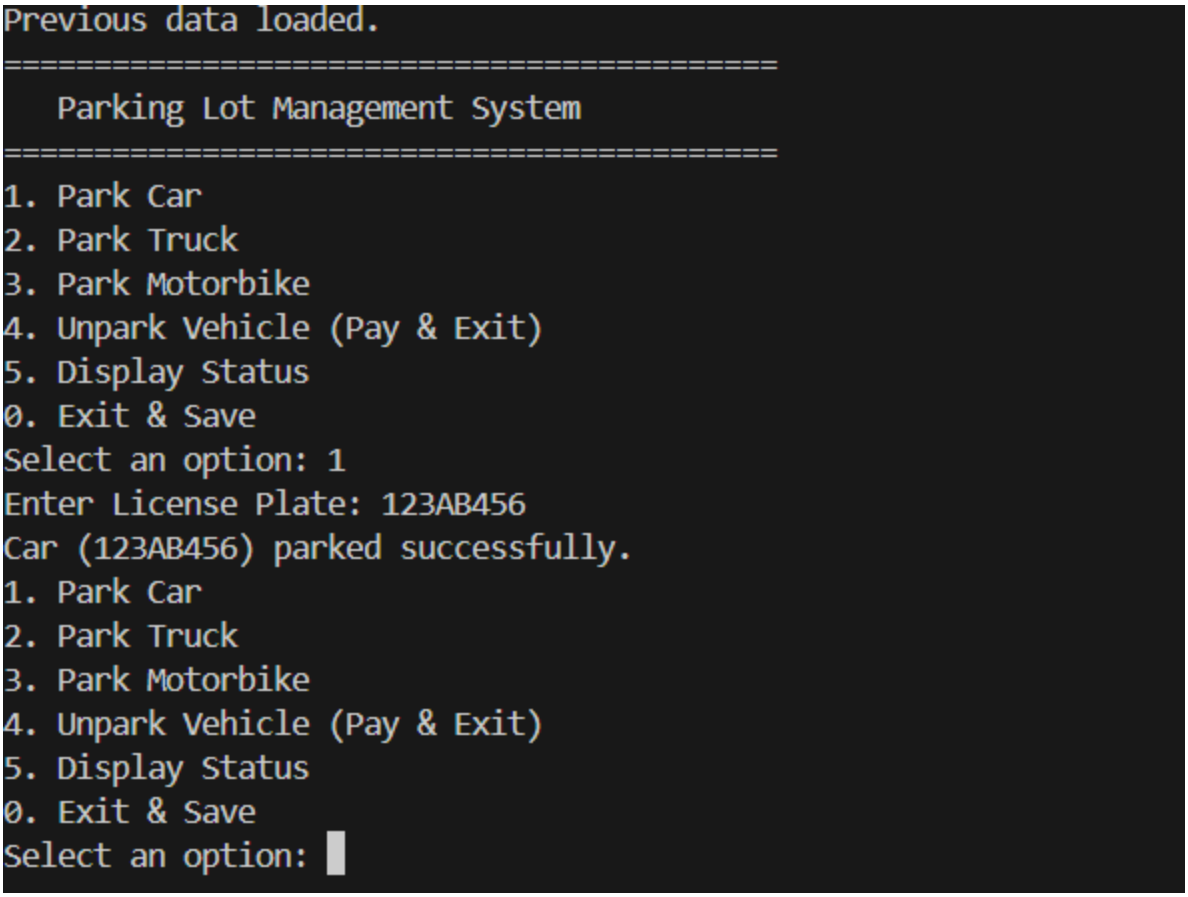
2. **Inheritance:** The `Car`, `Truck`, and `Motorbike` classes inherit attributes and methods from the `Vehicle` class, promoting code reusability and a hierarchical structure.
3. **Encapsulation:** Member variables (e.g., `totalRevenue`, `parkedVehicles`) are declared as `private` or `protected`. Access to these variables is controlled through public methods (Getters) and the constructor, ensuring data integrity.
4. **Abstraction:** The complexity of fee calculation is hidden behind the `calculateFee()` interface. The main program does not need to know *how* the fee is calculated, only that it *can* be calculated.

4. Advanced Features & Implementation Details

- **Dynamic Memory Management (STL Vector):** Instead of fixed-size arrays, the `std::vector` container is used to manage the list of parked vehicles dynamically. This allows the parking lot to handle varying numbers of vehicles efficiently.
- **Data Persistence (File I/O):** To ensure data is not lost when the program terminates, the system implements `saveData()` and `loadData()` methods.
    - **On Exit:** The state of all parked vehicles (Type, License Plate, Entry Time) is saved to `parking_data.txt`.
    - **On Startup:** The system reads the file and reconstructs the objects, restoring the parking lot to its previous state.

5. Usage & Screenshots

Scenario 1: Vehicle Entry The user selects a vehicle type and enters the license plate. The system verifies capacity and logs the entry time.



Scenario 2: Parking Status Report The system displays all currently parked vehicles with their details formatted in a readable table.

```
=== PARKING LOT STATUS (3/7) ===
Total Revenue: $0
-----------------------------------------------------
Car             123AB456        Entry: Sun Dec 28 01:37:36 2025
Truck           321MP56         Entry: Sun Dec 28 01:39:02 2025
Motorbike       547PY78         Entry: Sun Dec 28 01:39:20 2025
-----------------------------------------------------

1. Park Car
2. Park Truck
3. Park Motorbike
4. Unpark Vehicle (Pay & Exit)
5. Display Status
0. Exit & Save
Select an option:
```

Scenario 3: Vehicle Exit & Fee Calculation The user enters a license plate to unpark. The system calculates the duration and the total fee based on the vehicle type.

```
Select an option: 4
Enter License Plate to Unpark: 123AB456

--------------------------------
[EXIT] 123AB456 is leaving.
Vehicle Type: Car
Total Fee: $20
--------------------------------
```

6. Conclusion

This project successfully simulates a functional Parking Lot Management System. By leveraging C++ features like Virtual Functions, Pointers, and File Streams, the system achieves a robust and extensible design. The architecture allows for easy future expansions, such as adding new vehicle types or implementing different pricing strategies, without modifying the core logic.