DAYSTAR UNIVERSITY

SCHOOL OF SCIENCE, ENGINEERING AND HEALTH

PROJECT TITLE:CAR RENTAL SYSTEM

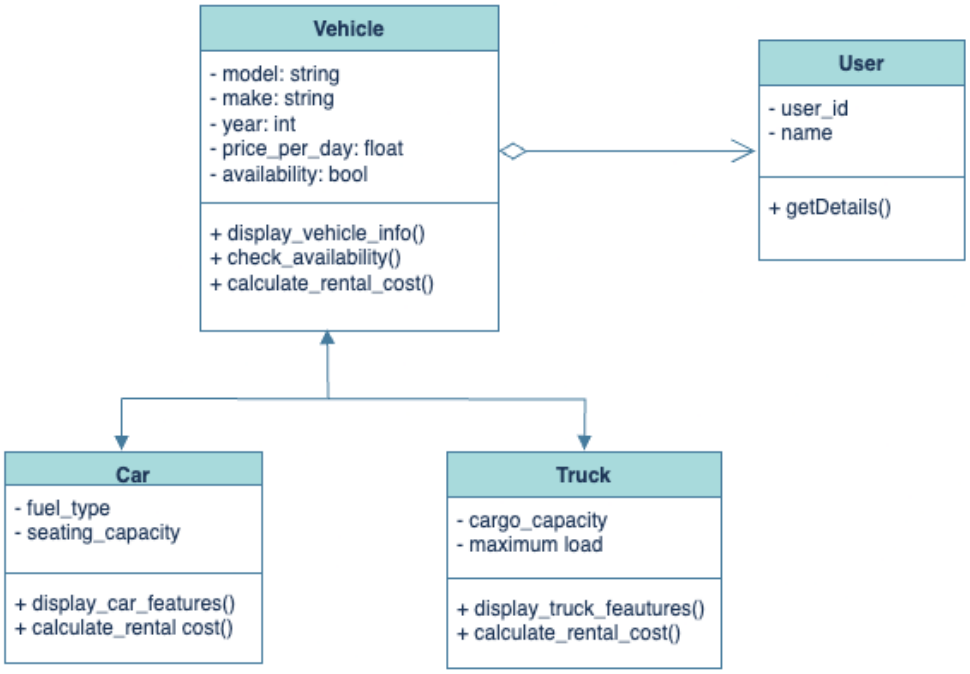
LAVENDER ILA

CAR RENTAL SYSTEM

The **Car Rental System** is designed to **allow users to rent vehicles**, including cars and trucks. The system provides functionality to display vehicle details, check availability, and calculate rental costs.

Major functionalities include:

a. **User Management**: Users can view their details and interact with the rental system.
b. **Vehicle Management**: Display information about the vehicles, check their availability, and calculate rental costs based on the price per day.
c. **Car Features**: Display specific details for cars, such as fuel type and seating capacity.
d. **Truck Features**: Display specific details for trucks, such as cargo capacity and maximum load.


Vehicle
- model: string
- make: string
- year: int
- price_per_day: float
- availability: bool
+ display_vehicle_info()
+ check_availability()
+ calculate_rental_cost()
User
- user_id
- name
+ getDetails()
Car
- fuel_type
- seating_capacity
+ display_car_features()
+ calculate_rental cost()
Truck
- cargo_capacity
- maximum load
+ display_truck_feautures()
+ calculate_rental_cost()

Base Class: Vehicle

- Attributes:
    - Model
    - make,
    - year,
    - price_per_day
    - availability.
- Methods:
    - getDetails(). for input.
    - displayDetails(). to display general details.
    - isAvailable(). to check availability.
    - calculateRentalCost(). to compute cost for a specified number of rental days.

Derived Class: Car

- Public derivation.
- Attributes:
    - fuel_type
    - seating_capacity.
- Methods:
    - getCarDetails(). for input.
    - displayCarDetails(). for detailed display.

Derived Class: Truck

- Private derivation.
- Additional attributes:
    - cargo_capacity. The cargo volume the truck can hold
    - maximum_load. The maximum weight the truck can carry.
- Methods:
    - getTruckDetails(). for input.
    - displayTruckDetails(). for detailed display.

CODE

```
#include <iostream>
#include <iomanip> // Input and output formatting
using namespace std;

// Base Class: Vehicle
class Vehicle {
protected:
    string model;
    string make;
    int year;
    double price_per_day;
    bool availability;

public:
    // Default Constructor
    Vehicle()
        : model("Unknown"), make("Unknown"), year(0), price_per_day(0.0), availability(false) {}

    // Parameterized Constructor
    Vehicle(string mod, string mk, int yr, double price, bool avail)
        : model(mod), make(mk), year(yr), price_per_day(price), availability(avail) {}

    void getDetails();        // Input vehicle details
    void displayDetails();    // Display vehicle details
    double calculateRentalCost(int days); // Calculate rental cost
};

// Derived Class: Car (Public Derivation)
class Car : public Vehicle {
    string fuel_type;
    int seating_capacity;

public:
    // Default Constructor
    Car() : Vehicle(), fuel_type("Unknown"), seating_capacity(0) {}

    // Parameterized Constructor
    Car(string mod, string mk, int yr, double price, bool avail, string fuel, int seating)
        : Vehicle(mod, mk, yr, price, avail), fuel_type(fuel), seating_capacity(seating) {}

    void getCarDetails();
    void displayCarDetails();
};
```

```
// Derived Class: Truck (Public Derivation)
class Truck : public Vehicle {
    double cargo_capacity; // tons
    int maximum_load;      // kg

public:
    // Default Constructor
    Truck() : Vehicle(), cargo_capacity(0.0), maximum_load(0) {}

    // Parameterized Constructor
    Truck(string mod, string mk, int yr, double price, bool avail, double cargo, int load)
        : Vehicle(mod, mk, yr, price, avail), cargo_capacity(cargo), maximum_load(load) {}

    void getTruckDetails();
    void displayTruckDetails();
};

// Implementation of Vehicle class methods
void Vehicle::getDetails() {
    cout << "\nEnter vehicle model: ";
    cin >> model;
    cout << "Enter vehicle make: ";
    cin >> make;
    cout << "Enter manufacturing year: ";
    cin >> year;
    cout << "Enter price per day: $";
    cin >> price_per_day;
    cout << "Is the vehicle available (1 for Yes, 0 for No)? ";
    cin >> availability;
}

void Vehicle::displayDetails() {
    cout << "Vehicle Model: " << setw(10) << model << endl;
    cout << "Vehicle Make: " << setw(10) << make << endl;
    cout << "Year: " << setw(10) << year << endl;
    cout << "Price per Day: $" << setw(10) << price_per_day << endl;
    cout << "Availability: " << setw(10) << (availability ? "Yes" : "No") << endl;
}

double Vehicle::calculateRentalCost(int days) {
    return price_per_day * days;
}

// Implementation of Car class methods
void Car::getCarDetails() {
    getDetails();
    cout << "Enter fuel type (Petrol, Diesel, Electric): ";
```

```
    cin >> fuel_type;
    cout << "Enter seating capacity: ";
    cin >> seating_capacity;
}

void Car::displayCarDetails() {
    displayDetails();
    cout << "Fuel Type: " << setw(10) << fuel_type << endl;
    cout << "Seating Capacity: " << setw(10) << seating_capacity << endl;
}

// Implementation of Truck class methods
void Truck::getTruckDetails() {
    getDetails();
    cout << "Enter cargo capacity (in tons): ";
    cin >> cargo_capacity;
    cout << "Enter maximum load (in kg): ";
    cin >> maximum_load;
}

void Truck::displayTruckDetails() {
    displayDetails();
    cout << "Cargo Capacity: " << setw(10) << cargo_capacity << " tons" << endl;
    cout << "Maximum Load: " << setw(10) << maximum_load << " kg" << endl;
}

// Main Function
int main() {
    while (true) {
        int choice;
        cout << "\nChoose vehicle type to add:\n";
        cout << "1. Car\n";
        cout << "2. Truck\n";
        cout << "3. Exit\n";
        cout << "Enter choice: ";
        cin >> choice;

        if (choice == 1) {
            Car car;
            car.getCarDetails();
            cout << "\nCar Details Entered:\n";
            car.displayCarDetails();

            int rental_days;
            cout << "\nEnter number of days to rent the car: ";
            cin >> rental_days;
            cout << "Rental cost for the car: $" << car.calculateRentalCost(rental_days) << endl;
```

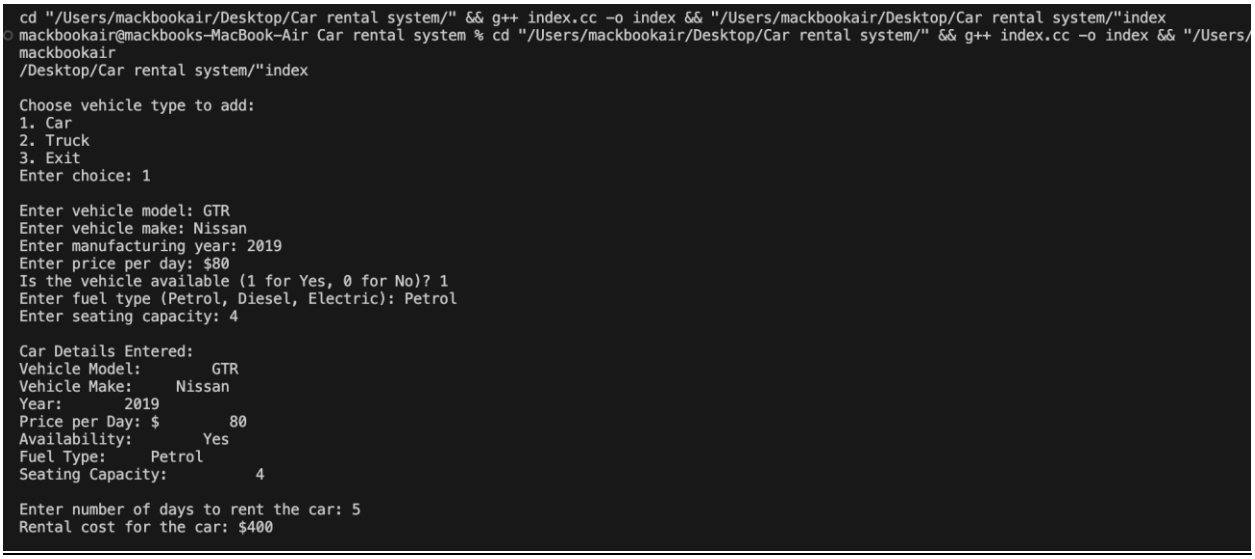
```
        } else if (choice == 2) {
            Truck truck;
            truck.getTruckDetails();
            cout << "\nTruck Details Entered:\n";
            truck.displayTruckDetails();

            int rental_days;
            cout << "\nEnter number of days to rent the truck: ";
            cin >> rental_days;
            cout << "Rental cost for the truck: $" << truck.calculateRentalCost(rental_days) << endl;

        } else if (choice == 3) {
            cout << "Exiting program. Goodbye!" << endl;
            break;
        } else {
            cout << "Invalid choice. Please try again." << endl;
        }
    }

    return 0;
}
```

OUTPUT



Object

Object is an instance of a class.
In this system, from **vehicles, the cars and trucks are represented as objects.** Each object encapsulates data such as **model, make, year, price_per_day and methods** such as **displayDetails, calculateRentalCost**

Class

A class is a blueprint for creating objects.
The system defines classes such as Vehicle, Car, and Truck. The **Vehicle class serves as the base class**, while **Car and Truck are derived classes**.

Inheritance

Inheritance allows a class to derive properties and behaviors from another class.
In this system, the **Car class publicly inherits from the Vehicle class**, while the **Truck class uses private inheritance**. This allows for code reuse and the addition of specialized features in the derived classes.

Data Hiding

Data hiding restricts access to certain details of an object.
In this system, attributes like **cargo_capacity and maximum_load in the Truck class are private**.
They are **accessed through methods**, ensuring controlled interaction with the data.

Encapsulation

Encapsulation binds data and methods together and restricts direct access to some of the object's components.
In this system, **attributes such as model and make are accessed** and **modified through methods like getDetails and displayDetails**.

Abstraction

Abstraction involves hiding implementation details and showing only essential features.

In the system, **methods like calculateRentalCost** abstract away the calculations, exposing only the functionality required by the user.

Polymorphism

Polymorphism allows methods to be used in different ways.

In this system, **the displayDetails method** is overridden in the **derived classes (Car and Truck)** to include specific details relevant to the respective vehicle type.