VEHICLE RENTAL SYSTEM

1. Problem Statement

- **Objective**: To create a system for managing vehicle rentals, allowing customers to book vehicles and admins to manage vehicles and bookings.
- **Goals**:
    - Allow customers to book vehicles for specified hours.
    - Enable admins to manage vehicle inventory (add/remove vehicles, view bookings).

2. Requirement Analysis

Functional Requirements

1. **Vehicle Management**:
    - Admins can add/remove vehicles.
    - Admins can view all vehicles and their availability.
2. **Booking Management**:
    - Customers can book vehicles for specified hours.
    - Only available vehicles can be booked.
    - System displays vehicle details, availability, and price per hour.
3. **Availability Management**:
    - Admins can mark vehicles as available after a booking is completed or canceled.
    - Vehicles are unavailable during booking periods.

Non-Functional Requirements

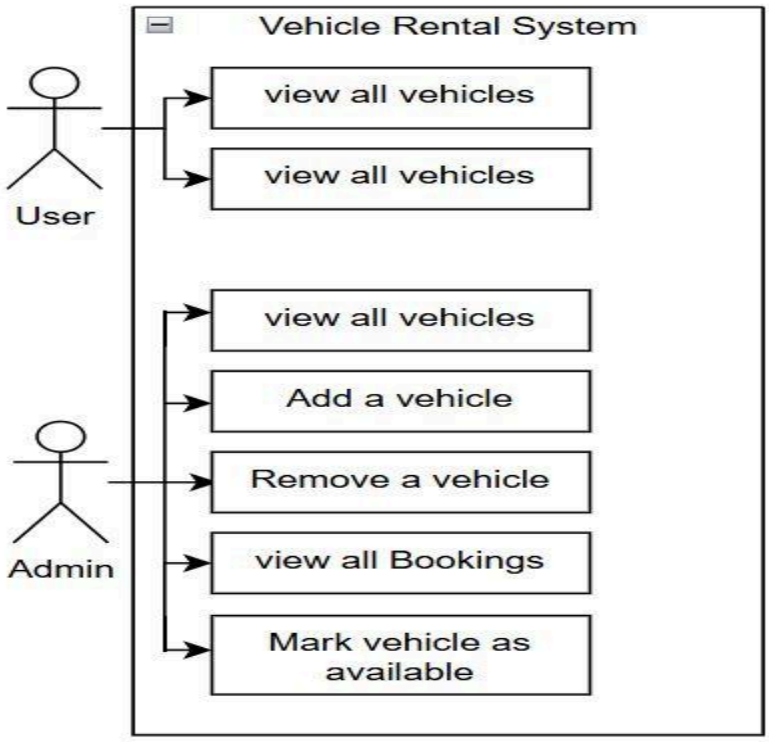
- **Performance**: Handle up to 500 vehicles and 1000 bookings efficiently.
- **Scalability**: System should scale for more users and vehicles.

3. Use Case Modeling

Use Case Diagrams

- **Actors**:
    - **Customer**: Views vehicles, books a vehicle, manages bookings.

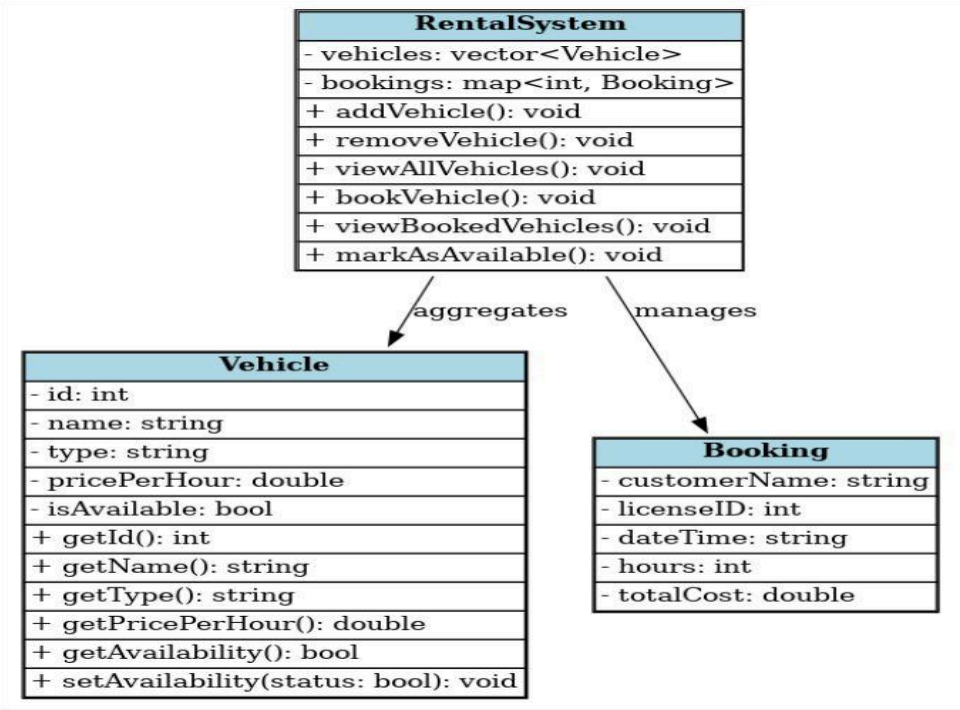
    - **Administrator**: Manages vehicles, adds/removes vehicles, views all bookings.
- **Use Cases**:
    - **Customer**:
        - View all vehicles.
        - Book a vehicle.
    - **Administrator**:
        - Add a vehicle.
        - Remove a vehicle.
        - View all vehicles.
        - View all bookings.
        - Mark vehicle as available.



4. Class Diagram

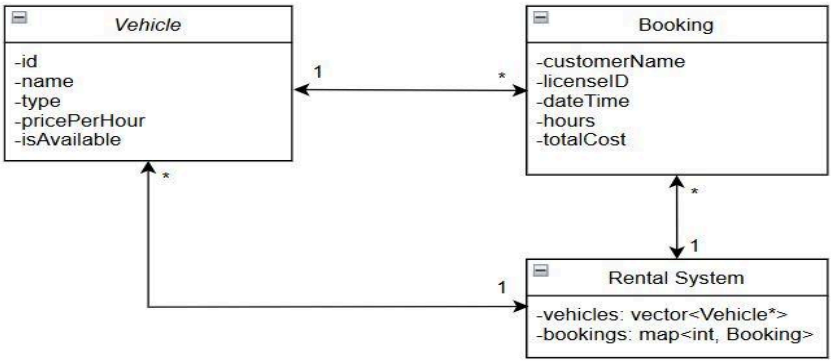
- **Classes**:
    - **Vehicle**:
        - Attributes: `id`, `name`, `type`, `pricePerHour`, `isAvailable`.
        - Methods: `getId()`, `getName()`, `setAvailability()`.

- **Booking**:
  - Attributes: `customerName`, `licenseID`, `dateTime`, `hours`, `totalCost`.
  - Methods: `calculateTotalCost()`.
- **RentalSystem**:
  - Manages `vehicles` and `bookings`.
  - Methods: `addVehicle()`, `removeVehicle()`, `bookVehicle()`, `viewBookedVehicles()`.



5. Domain Model:

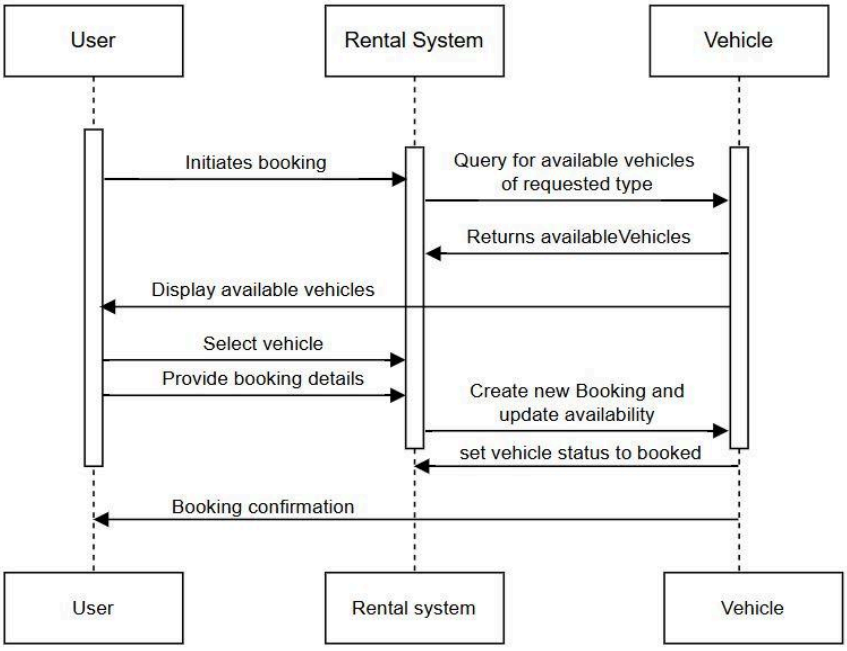
- The `RentalSystem` manages multiple `Vehicle` objects.
- A `Vehicle` can have one active `Booking` at a time but may have multiple historical bookings.
- The `RentalSystem` keeps track of multiple `Booking` instances.



6. Sequence Diagram

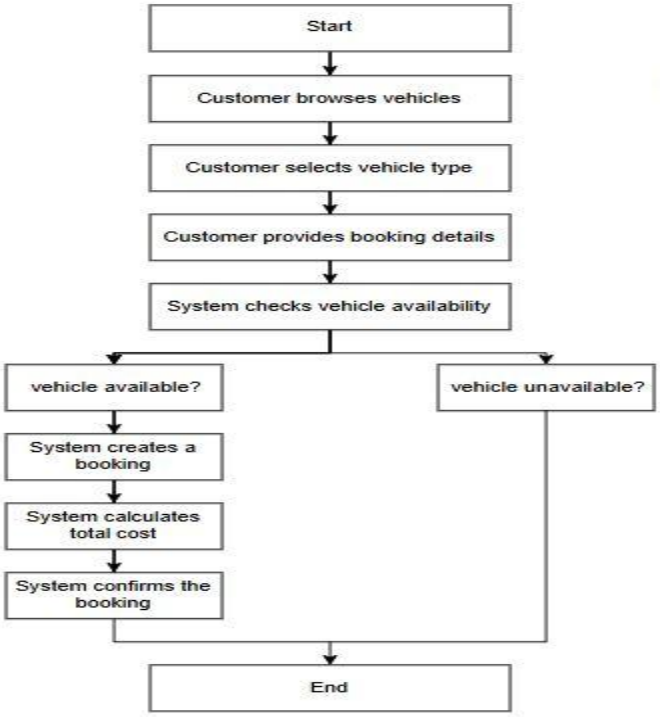
Booking a Vehicle:

1. The **Customer** selects a vehicle from the system.
2. The **RentalSystem** checks the vehicle's availability.
3. The **Customer** provides booking details and confirms the reservation.
4. The **RentalSystem** creates a new **Booking** object.
5. The system calculates the total cost and finalizes the booking.



7. Activity Diagram

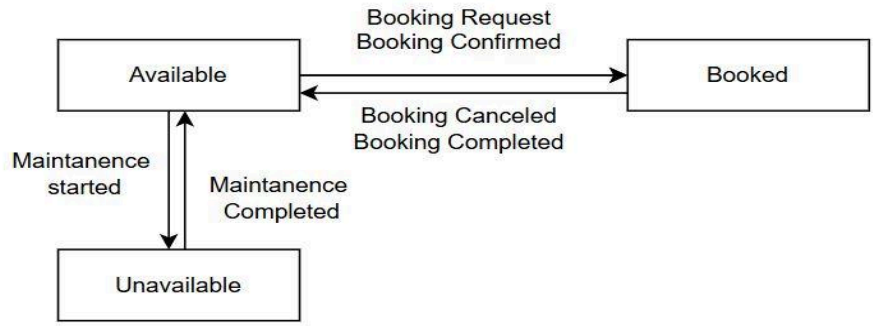
Customer Booking Flow



8. State Diagram

Vehicle States:

- **Available**: Vehicle is available for booking.
- **Booked**: Vehicle is booked for a time frame.
- **Unavailable**: Vehicle is not available



9. Code Implementation

- **Classes**: `Vehicle`, `Booking`, `RentalSystem`.
- **OOP Principles**:
  - **Encapsulation**: Attributes are hidden; accessed via getters/setters.
  - **Inheritance**: `Vehicle` class can be extended for specific types (e.g., `Two-Wheeler`, `Four-Wheeler`).
  - **Polymorphism**: Common method (`bookVehicle()`) for different vehicle types.
  - **Abstraction**: Simplified user interaction without exposing implementation details.
- **Code**:

```
#include <iostream>

#include <vector>

#include <string>

#include <iomanip>

#include <algorithm>

#include <map>


// Vehicle Class

class Vehicle {

protected:
```

```
    int id;
    std::string name;
    std::string type;
    double pricePerHour;
    bool isAvailable;

public:
    Vehicle(int id, const std::string& name, const std::string& type, double pricePerHour)
        : id(id), name(name), type(type), pricePerHour(pricePerHour), isAvailable(true) {}

    int getId() const { return id; }
    std::string getName() const { return name; }
    std::string getType() const { return type; }
    double getPricePerHour() const { return pricePerHour; }
    bool getAvailability() const { return isAvailable; }

    void setAvailability(bool status) { isAvailable = status; }
};

// Booking Class
struct Booking {
    std::string customerName;
    int licenseID;
    std::string dateTime;
```

```
    int hours;
    double totalCost;
};

// RentalSystem Class
class RentalSystem {
private:
    std::vector<Vehicle*> vehicles;
    std::map<int, Booking> bookings; // Maps vehicle ID to Booking details

public:
    ~RentalSystem() {
        for (Vehicle* vehicle : vehicles) {
            delete vehicle;
        }
    }

    void addVehicle() {
        int id;
        std::string name, type;
        double pricePerHour;

        std::cout << "Enter Vehicle ID: ";
        std::cin >> id;
```

```
    std::cin.ignore(); // Ignore leftover newline
    std::cout << "Enter Vehicle Name: ";
    std::getline(std::cin, name);
    std::cout << "Enter Vehicle Type (Two-Wheeler/Four-Wheeler): ";
    std::getline(std::cin, type);
    std::cout << "Enter Price per Hour: ";
    std::cin >> pricePerHour;

    vehicles.push_back(new Vehicle(id, name, type, pricePerHour));
    std::cout << "Vehicle added successfully!" << std::endl;
}

void removeVehicle() {
    int id;
    std::cout << "Enter Vehicle ID to remove: ";
    std::cin >> id;

    auto it = std::remove_if(vehicles.begin(), vehicles.end(),
        [id](Vehicle* vehicle) { return vehicle->getId() == id; });

    if (it != vehicles.end()) {
        delete *it;
        vehicles.erase(it, vehicles.end());
        std::cout << "Vehicle removed successfully!" << std::endl;
```

```
    } else {
        std::cout << "Vehicle ID not found!" << std::endl;
    }
}

void viewAllVehicles() {
    if (vehicles.empty()) {
        std::cout << "No vehicles available." << std::endl;
        return;
    }

    std::cout << std::left << std::setw(10) << "ID"
              << std::setw(20) << "Name"
              << std::setw(15) << "Type"
              << std::setw(15) << "Price/Hour"
              << std::setw(15) << "Availability" << std::endl;

    for (Vehicle* vehicle : vehicles) {
        std::cout << std::left << std::setw(10) << vehicle->getId()
                  << std::setw(20) << vehicle->getName()
                  << std::setw(15) << vehicle->getType()
                  << std::setw(15) << vehicle->getPricePerHour()
                  << std::setw(15) << (vehicle->getAvailability() ? "Available" : "Booked")
                  << std::endl;
```

```
    }
}

void bookVehicle() {
    std::string type;
    std::cout << "Enter Vehicle Type (Two-Wheeler/Four-Wheeler): ";
    std::cin.ignore();
    std::getline(std::cin, type);

    std::vector<Vehicle*> availableVehicles;
    for (Vehicle* vehicle : vehicles) {
        if (vehicle->getType() == type && vehicle->getAvailability()) {
            availableVehicles.push_back(vehicle);
        }
    }

    if (availableVehicles.empty()) {
        std::cout << "No vehicles available of type: " << type << std::endl;
        return;
    }

    std::cout << "Available Vehicles:" << std::endl;
    for (size_t i = 0; i < availableVehicles.size(); ++i) {
        std::cout << i + 1 << ". " << availableVehicles[i]->getName()
```

```
                << " (ID: " << availableVehicles[i]->getId() << ")"
                << " - $" << availableVehicles[i]->getPricePerHour() << "/hour" <<
std::endl;
        }

        int choice;
        std::cout << "Enter the number of the vehicle to book: ";
        std::cin >> choice;

        if (choice > 0 && choice <= static_cast<int>(availableVehicles.size())) {
            Vehicle* vehicle = availableVehicles[choice - 1];
            vehicle->setAvailability(false);

            Booking booking;
            std::cout << "Enter your name: ";
            std::cin.ignore();
            std::getline(std::cin, booking.customerName);
            std::cout << "Enter your Driving License ID: ";
            std::cin >> booking.licenseID;
            std::cin.ignore();
            std::cout << "Enter booking date and time (e.g., 2024-12-31 14:00): ";
            std::getline(std::cin, booking.dateTime);
            std::cout << "Enter number of hours: ";
            std::cin >> booking.hours;
```

```
            booking.totalCost = booking.hours * vehicle->getPricePerHour();
            bookings[vehicle->getId()] = booking;

            std::cout << "Booking successful! Total Cost: $" << booking.totalCost <<
std::endl;
        } else {
            std::cout << "Invalid choice." << std::endl;
        }
    }

    void viewBookedVehicles() {
        if (bookings.empty()) {
            std::cout << "No vehicles are currently booked." << std::endl;
            return;
        }

        std::cout << std::left << std::setw(10) << "ID"
                  << std::setw(20) << "Customer"
                  << std::setw(15) << "License ID"
                  << std::setw(25) << "Booking Time"
                  << std::setw(10) << "Hours"
                  << std::setw(10) << "Total Cost" << std::endl;

        for (const auto& [vehicleID, booking] : bookings) {
            std::cout << std::left << std::setw(10) << vehicleID
```

```
                << std::setw(20) << booking.customerName
                << std::setw(15) << booking.licenseID
                << std::setw(25) << booking.dateTime
                << std::setw(10) << booking.hours
                << std::setw(10) << booking.totalCost
                << std::endl;
    }
}

void markAsAvailable() {
    int id;
    std::cout << "Enter Vehicle ID to mark as available: ";
    std::cin >> id;

    auto it = bookings.find(id);
    if (it != bookings.end()) {
        bookings.erase(it);
        for (Vehicle* vehicle : vehicles) {
            if (vehicle->getId() == id) {
                vehicle->setAvailability(true);
                std::cout << "Vehicle marked as available." << std::endl;
                return;
            }
        }
```

```
        } else {
            std::cout << "No booking found for this vehicle ID." << std::endl;
        }
    }
};

void adminMenu(RentalSystem& system) {
    int choice;
    do {
        std::cout << "\n--- Admin Menu ---" << std::endl;
        std::cout << "1. Add Vehicle\n2. Remove Vehicle\n3. View All Vehicles\n4. View Booked Vehicles\n5. Mark Vehicle as Available\n6. Back" << std::endl;
        std::cout << "Enter your choice: ";
        std::cin >> choice;

        switch (choice) {
            case 1: system.addVehicle(); break;
            case 2: system.removeVehicle(); break;
            case 3: system.viewAllVehicles(); break;
            case 4: system.viewBookedVehicles(); break;
            case 5: system.markAsAvailable(); break;
            case 6: break;
            default: std::cout << "Invalid choice. Try again." << std::endl;
        }
    } while (choice != 6);
```

```
}

void userMenu(RentalSystem& system) {
    int choice;
    do {
        std::cout << "\n--- User Menu ---" << std::endl;
        std::cout << "1. View All Vehicles\n2. Book Vehicle\n3. Back" << std::endl;
        std::cout << "Enter your choice: ";
        std::cin >> choice;

        switch (choice) {
            case 1: system.viewAllVehicles(); break;
            case 2: system.bookVehicle(); break;
            case 3: break;
            default: std::cout << "Invalid choice. Try again." << std::endl;
        }
    } while (choice != 3);
}

int main() {
    RentalSystem system;

    int role;
    do {
```

```
        std::cout << "\n--- Rental System ---" << std::endl;
        std::cout << "1. Admin\n2. Customer\n3. Exit\nEnter your role: ";
        std::cin >> role;

        if (role == 1) {
            std::string password;
            std::cout << "Enter admin password: ";
            std::cin >> password;

            if (password == "ad@rent") {
                adminMenu(system);
            } else {
                std::cout << "Invalid password." << std::endl;
            }
        } else if (role == 2) {
            userMenu(system);
        } else if (role != 3) {
            std::cout << "Invalid role. Try again." << std::endl;
        }
    } while (role != 3);

    return 0;
}
```

10. Future Enhancements

- Develop a web or mobile application for users to browse, book, and pay online.
- Implement a system to track vehicle maintenance schedules.
- Integrate GPS tracking for rented vehicles to ensure security.
- Introduce demand-based pricing algorithms and offer discounts for long-term rentals or frequent customers.

11.Conclusion

The Vehicle Rental System is a structured, object-oriented solution for managing vehicle rentals by administrators and customers. Using OOAD principles, it ensures modularity, scalability, and easy maintenance. It offers features like vehicle availability tracking, booking management, and user interaction, with potential for future enhancements to meet modern rental system demands.