Muhammad Shaheer | FA24-BSE-104(B)

Object Oriented Programming

Sir Ameer Gillani

COMSATS UNIVERSITY ISLAMABAD, SAHIWAL

CLO-02: Apply OOP concepts, interfaces, and exception handling to real-world problems.

23 October 2025

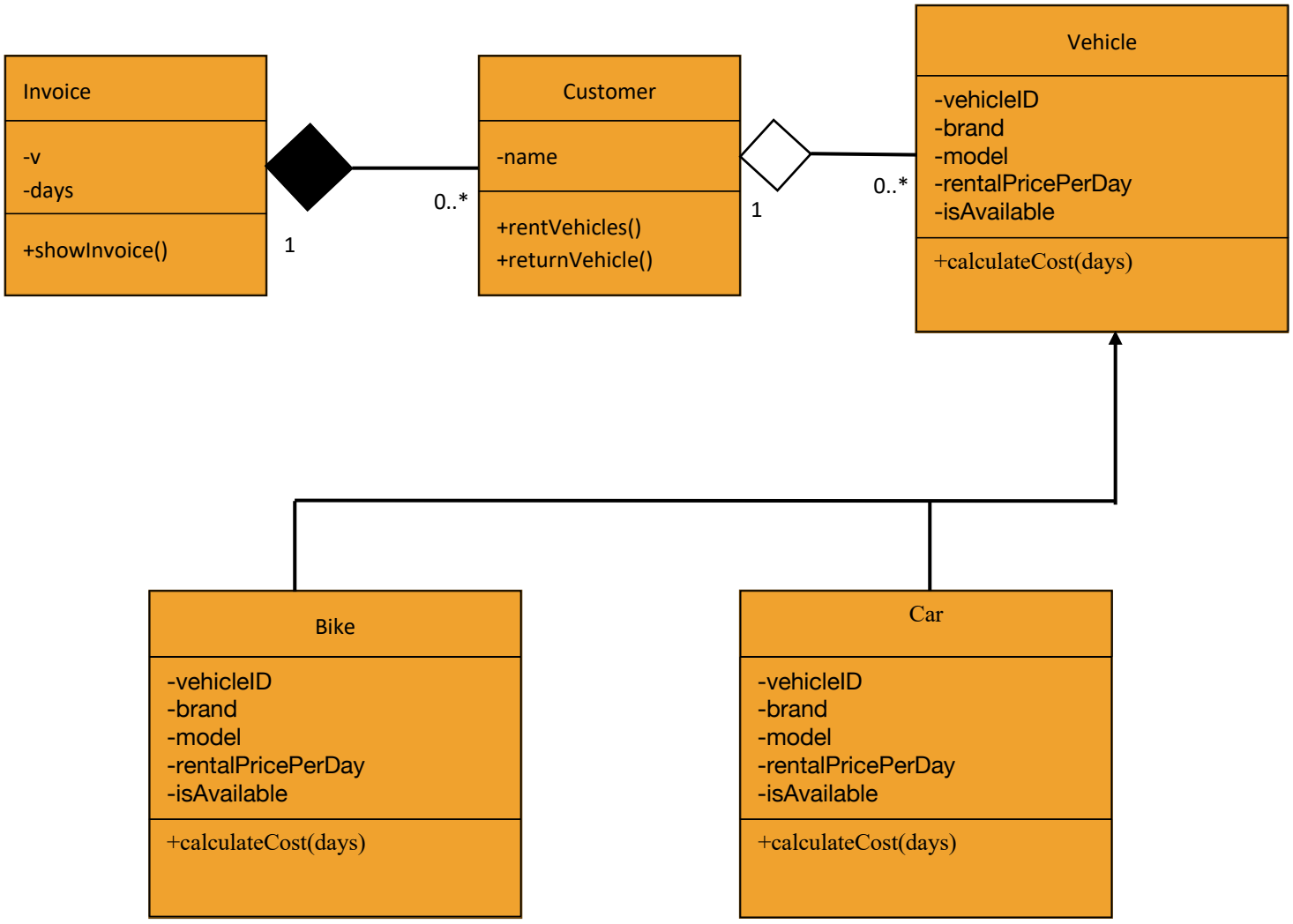
ASSIGNMENT#02

SCENARIO 1: VEHICLE REGISTRATION SYSTEM

1. IDENTIFY AND LIST THE **CLASSES** NEEDED FOR THE BELOW MENTIONED SCENARIOS.

- **Vehicle class**
- **Car (Subclass of Vehicle)**
- **Bike (Subclass of Vehicle)**
- **Customer class**
- **Invoice class**

2. IMPLEMENT A **CLASS DIAGRAM** REPRESENTING EACH SYSTEM.



3. EXPLAIN THE RELATIONSHIPS BETWEEN THE CLASSES (E.G., COMPOSITION, AGGREGATION, OR INHERITANCE).

Inheritance: Car inherit from Vehicle (is a relation).

Inheritance: Bike inherit from Vehicle (is a relation).

Composition: Invoice has composition with both Customer and Vehicle (weak has a relation).

Aggregation: Vehicle and Customer show aggregation (weak has a relation).

4. SUGGEST HOW **ENCAPSULATION** CAN BE APPLIED TO ENSURE DATA SECURITY.

All attributes of the classes such as **Vehicle**, **Car**, **Bike**, **Customer**, and **Invoice** are declared as **private** to protect the internal data from direct access or modification.

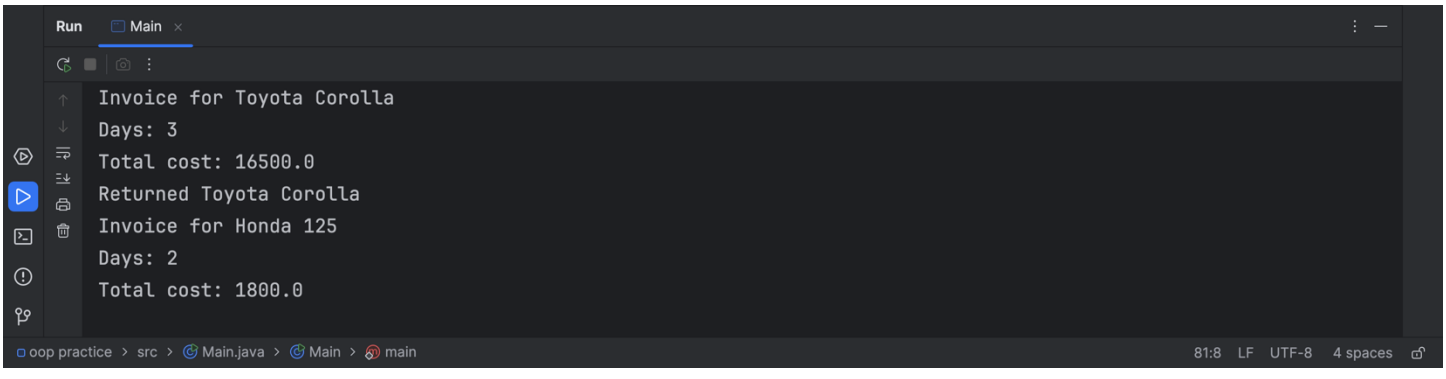
Public **getter and setter methods** are provided to access and modify these private attributes in a controlled way. This allows interaction with class data while keeping the implementation details hidden, ensuring proper encapsulation and secure data handling.

5. WRITE **CODES** IN JAVA TO FOR EACH SCENARIO TO DEMONSTRATE THE FUNCTIONALITY WITH APPROPRIATE CONSTRUCTORS, GETTERS/SETTERS, AND METHOD OVERRIDING CONCEPTS.

```
class Vehicle {
    String vehicleID;
    String brand;
    String model;
    double rentalPricePerDay;
    boolean isAvailable = true;

    Vehicle(String id, String b, String m, double price) {
        vehicleID = id;
        brand = b;
        model = m;
        rentalPricePerDay = price;
    }

    double calculateCost(int days) {
```

```
        return rentalPricePerDay * days;
    }
}

class Car extends Vehicle {
    Car(String id, String b, String m, double price) {
        super(id, b, m, price);
    }

    double calculateCost(int days) {
        return super.calculateCost(days) * 1.1;
    }
}

class Bike extends Vehicle {
    Bike(String id, String b, String m, double price) {
        super(id, b, m, price);
    }

    double calculateCost(int days) {
        return super.calculateCost(days) * 0.9;
    }
}

class Customer {
    String name;

    Customer(String n) {
        name = n;
    }

    void rentVehicle(Vehicle v, int days) {
        if (v.isAvailable) {
            v.isAvailable = false;
            Invoice invoice = new Invoice(v, days);
            invoice.showInvoice();
        } else {
            System.out.println("Not available");
        }
    }

    void returnVehicle(Vehicle v) {
        v.isAvailable = true;
```

```
        System.out.println("Returned " + v.brand + " " +
v.model);
    }
}

class Invoice {
    Vehicle v;
    int days;

    Invoice(Vehicle v, int d) {
        this.v = v;
        days = d;
    }

    void showInvoice() {
        double total = v.calculateCost(days);
        System.out.println("Invoice for " + v.brand + " " +
v.model);
        System.out.println("Days: " + days);
        System.out.println("Total cost: " + total);
    }
}

public class Main {
    public static void main(String[] args) {
        Car c1 = new Car("C1", "Toyota", "Corolla", 5000);
        Bike b1 = new Bike("B1", "Honda", "125", 1000);
        Customer cust = new Customer("Shaheer");

        cust.rentVehicle(c1, 3);
        cust.returnVehicle(c1);
        cust.rentVehicle(b1, 2);
    }
}
```

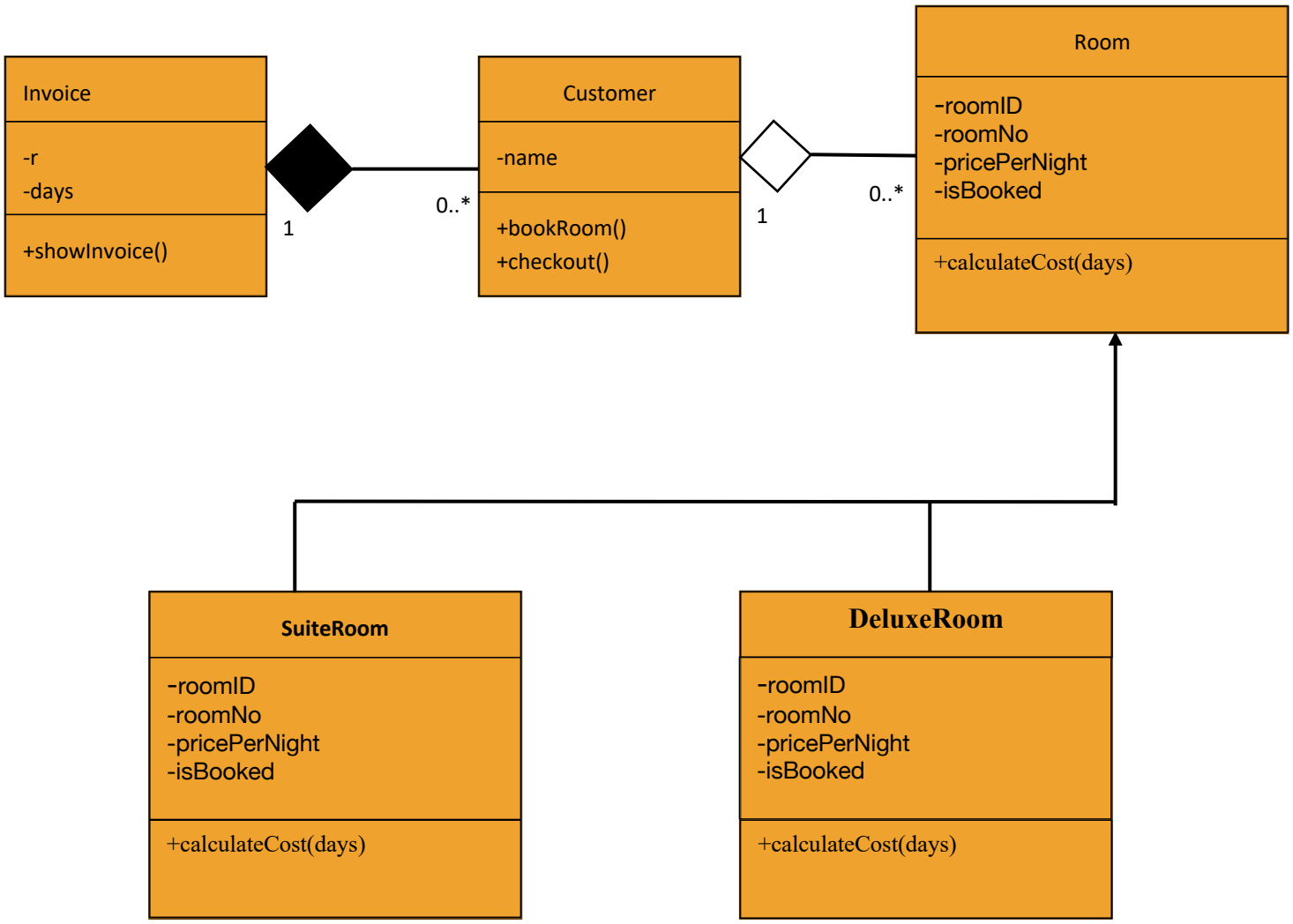


SCENARIO 2: HOTEL MANAGEMENT SYSTEM

1. IDENTIFY AND LIST THE **CLASSES** NEEDED FOR THE BELOW MENTIONED SCENARIOS.

- **Room**
- **DeluxeRoom (Subclass)**
- **SuiteRoom (Subclass)**
- **Guest**
- **Booking**

2. IMPLEMENT A **CLASS DIAGRAM** REPRESENTING EACH SYSTEM.



3. EXPLAIN THE RELATIONSHIPS BETWEEN THE CLASSES (E.G., COMPOSITION, AGGREGATION, OR INHERITANCE).

Inheritance: DeluxeRoom inherits from Room (is-a relation).

Inheritance: SuiteRoom inherits from Room (is-a relation).

Composition: Booking has composition with both Guest and Room (strong has a relation).

Aggregation: Room has Aggregation with Guest (weak has a relation).

4. SUGGEST HOW **ENCAPSULATION** CAN BE APPLIED TO ENSURE DATA SECURITY.

All attributes of the classes such as Room, DeluxeRoom, SuiteRoom, Guest, and Booking are declared as **private** to protect internal data from unauthorized access or modification.

Public getter and setter methods are used to access and modify these private attributes in a **controlled manner**, ensuring that any interaction with the data follows predefined validation rules.

This maintains **data security and proper encapsulation**, preventing accidental or malicious changes to sensitive class data.

5. WRITE **CODES** IN JAVA TO FOR EACH SCENARIO TO DEMONSTRATE THE FUNCTIONALITY WITH APPROPRIATE CONSTRUCTORS, GETTERS/SETTERS, AND METHOD OVERRIDING CONCEPTS.

```
class Room {
    String roomID;
    int roomNo;
    double pricePerNight;
    boolean isBooked = false;

    Room(String id, int no, double price) {
        roomID = id;
        roomNo = no;
        pricePerNight = price;
    }
```

```
    double calculateCost(int days) {
        return pricePerNight * days;
    }
}

class DeluxeRoom extends Room {
    DeluxeRoom(String id, int no, double price) {
        super(id, no, price);
    }

    double calculateCost(int days) {
        return super.calculateCost(days) * 1.2;
    }
}

class SuiteRoom extends Room {
    SuiteRoom(String id, int no, double price) {
        super(id, no, price);
    }

    double calculateCost(int days) {
        return super.calculateCost(days) * 1.5;
    }
}

class Guest {
    String name;

    Guest(String n) {
        name = n;
    }

    void bookRoom(Room r, int days) {
        if (!r.isBooked) {
            r.isBooked = true;
            Invoice i = new Invoice(r, days);
            i.showInvoice();
        } else {
            System.out.println("Room already booked");
        }
    }

    void checkout(Room r) {
```

```
        r.isBooked = false;
        System.out.println("Checked out from room " +
r.roomNo);
    }
}

class Invoice {
    Room r;
    int days;

    Invoice(Room r, int d) {
        this.r = r;
        days = d;
    }

    void showInvoice() {
        double total = r.calculateCost(days);
        System.out.println("Invoice for Room " + r.roomNo);
        System.out.println("Days: " + days);
        System.out.println("Total cost: " + total);
    }
}

public class Main {
    public static void main(String[] args) {
        DeluxeRoom d1 = new DeluxeRoom("D1", 101, 5000);
        SuiteRoom s1 = new SuiteRoom("S1", 202, 8000);
        Guest g1 = new Guest("Shaheer");

        g1.bookRoom(d1, 3);
        g1.checkout(d1);
        g1.bookRoom(s1, 2);
    }
}
```