**Core Requirements Breakdown:**

1. **Abstraction**: The Vehicle class is abstract and defines the necessary methods that all vehicle types must implement.
2. **Inheritance**: We create specific vehicle types (Car, Motorcycle, and Truck) that inherit from the Vehicle class and implement their own behavior for rental calculations and availability checks.
3. **Encapsulation**: All vehicle details are encapsulated as private fields, and we provide public getters and setters with validation.
4. **Polymorphism**: We implement the Rentable interface and use method overriding in vehicle classes to handle rental behavior.

// Abstraction - Vehicle Class (Abstract)

abstract class Vehicle {

private String vehicleId;

private String model;

private double baseRentalRate;

private boolean isAvailable;

// Constructor with validation

public Vehicle(String vehicleId, String model, double baseRentalRate) {

if (vehicleId == null || model == null || baseRentalRate <= 0) {

throw new IllegalArgumentException("Invalid vehicle details");

}

this.vehicleId = vehicleId;

this.model = model;

this.baseRentalRate = baseRentalRate;

this.isAvailable = true; // Default availability

}

// Getters and Setters with validation

public String getVehicleId() {

return vehicleId;

}

public void setVehicleId(String vehicleId) {

this.vehicleId = vehicleId;

}

public String getModel() {

return model;

}

public void setModel(String model) {

this.model = model;

}

public double getBaseRentalRate() {

return baseRentalRate;

}

public void setBaseRentalRate(double baseRentalRate) {

if (baseRentalRate <= 0) {

throw new IllegalArgumentException("Base rental rate must be positive");

}

this.baseRentalRate = baseRentalRate;

}

public boolean isAvailable() {

return isAvailable;

}

public void setAvailable(boolean available) {

isAvailable = available;

}

// Abstract methods

public abstract double calculateRentalCost(int days);

public abstract boolean isAvailableForRental();

}

// Rentable Interface

interface Rentable {

void rent(Customer customer, int days);

void returnVehicle();

}

// Concrete Vehicle Classes - Inheritance and Polymorphism

class Car extends Vehicle implements Rentable {

private String carType; // e.g., Sedan, SUV

public Car(String vehicleId, String model, double baseRentalRate, String carType) {

super(vehicleId, model, baseRentalRate);

this.carType = carType;

}

@Override

public double calculateRentalCost(int days) {

return getBaseRentalRate() \* days \* 1.1; // 10% surcharge for car rentals

}

@Override

public boolean isAvailableForRental() {

return isAvailable();

}

@Override

public void rent(Customer customer, int days) {

if (!isAvailableForRental()) {

System.out.println("Car is not available for rent.");

} else {

System.out.println("Car rented to " + customer.getName() + " for " + days + " days.");

setAvailable(false);

}

}

@Override

public void returnVehicle() {

System.out.println("Car returned.");

setAvailable(true);

}

@Override

public String toString() {

return "Car[ID=" + getVehicleId() + ", Model=" + getModel() + ", Type=" + carType + "]";

}

}

class Motorcycle extends Vehicle implements Rentable {

private boolean hasSidecar;

public Motorcycle(String vehicleId, String model, double baseRentalRate, boolean hasSidecar) {

super(vehicleId, model, baseRentalRate);

this.hasSidecar = hasSidecar;

}

@Override

public double calculateRentalCost(int days) {

return getBaseRentalRate() \* days \* (hasSidecar ? 1.2 : 1.0); // Extra for sidecar

}

@Override

public boolean isAvailableForRental() {

return isAvailable();

}

@Override

public void rent(Customer customer, int days) {

if (!isAvailableForRental()) {

System.out.println("Motorcycle is not available for rent.");

} else {

System.out.println("Motorcycle rented to " + customer.getName() + " for " + days + " days.");

setAvailable(false);

}

}

@Override

public void returnVehicle() {

System.out.println("Motorcycle returned.");

setAvailable(true);

}

@Override

public String toString() {

return "Motorcycle[ID=" + getVehicleId() + ", Model=" + getModel() + ", Has Sidecar=" + hasSidecar + "]";

}

}

class Truck extends Vehicle implements Rentable {

private double cargoCapacity; // in tons

public Truck(String vehicleId, String model, double baseRentalRate, double cargoCapacity) {

super(vehicleId, model, baseRentalRate);

this.cargoCapacity = cargoCapacity;

}

@Override

public double calculateRentalCost(int days) {

return getBaseRentalRate() \* days \* 1.5; // Extra for large vehicle

}

@Override

public boolean isAvailableForRental() {

return isAvailable();

}

@Override

public void rent(Customer customer, int days) {

if (!isAvailableForRental()) {

System.out.println("Truck is not available for rent.");

} else {

System.out.println("Truck rented to " + customer.getName() + " for " + days + " days.");

setAvailable(false);

}

}

@Override

public void returnVehicle() {

System.out.println("Truck returned.");

setAvailable(true);

}

@Override

public String toString() {

return "Truck[ID=" + getVehicleId() + ", Model=" + getModel() + ", Cargo Capacity=" + cargoCapacity + " tons]";

}

}

// Supporting Classes - Encapsulation

class Customer {

private String name;

private int customerId;

private boolean isLoyaltyMember;

public Customer(String name, int customerId) {

this.name = name;

this.customerId = customerId;

this.isLoyaltyMember = false; // Default non-loyalty member

}

public String getName() {

return name;

}

public int getCustomerId() {

return customerId;

}

public boolean isLoyaltyMember() {

return isLoyaltyMember;

}

public void setLoyaltyMember(boolean loyaltyMember) {

isLoyaltyMember = loyaltyMember;

}

}

class RentalAgency {

private Vehicle[] vehicles;

private int rentalCount;

public RentalAgency(Vehicle[] vehicles) {

this.vehicles = vehicles;

this.rentalCount = 0;

}

public void rentVehicle(int vehicleIndex, Customer customer, int days) {

if (vehicleIndex >= 0 && vehicleIndex < vehicles.length) {

vehicles[vehicleIndex].rent(customer, days);

} else {

System.out.println("Invalid vehicle index.");

}

}

public void returnVehicle(int vehicleIndex) {

if (vehicleIndex >= 0 && vehicleIndex < vehicles.length) {

vehicles[vehicleIndex].returnVehicle();

} else {

System.out.println("Invalid vehicle index.");

}

}

}

// Main class to demonstrate functionality

public class VehicleRentalSystem {

public static void main(String[] args) {

// Create sample vehicles

Vehicle car1 = new Car("C001", "Toyota Camry", 50, "Sedan");

Vehicle bike1 = new Motorcycle("M001", "Harley Davidson", 30, true);

Vehicle truck1 = new Truck("T001", "Volvo", 100, 5.0);

// Create a Rental Agency

RentalAgency agency = new RentalAgency(new Vehicle[]{car1, bike1, truck1});

// Create customer

Customer customer1 = new Customer("John Doe", 1);

// Rent vehicles

agency.rentVehicle(0, customer1, 5); // Renting car for 5 days

agency.rentVehicle(1, customer1, 3); // Renting bike for 3 days

// Return vehicles

agency.returnVehicle(0);

agency.returnVehicle(1);

}

}