DCIT 201 ASSIGNMENT FEEDBACK

```
// Vehicle.java (Abstract Class)
public abstract class Vehicle {
    private String vehicleId;
    private String model;
    private double baseRentalRate;
    private boolean isAvailable;

    public Vehicle(String vehicleId, String model, double baseRentalRate) {
        if (vehicleId == null || vehicleId.isEmpty()) throw new IllegalArgumentException("Vehicle ID cannot be null or empty");
        if (model == null || model.isEmpty()) throw new IllegalArgumentException("Model cannot be null or empty");
        if (baseRentalRate <= 0) throw new IllegalArgumentException("Base rental rate must be greater than 0");

        this.vehicleId = vehicleId;
        this.model = model;
        this.baseRentalRate = baseRentalRate;
        this.isAvailable = true;
    }

    public String getVehicleId() {
        return vehicleId;
    }

    public String getModel() {
        return model;
    }

    public void setModel(String model) {
        if (model == null || model.isEmpty()) throw new IllegalArgumentException("Model cannot be null or empty");
        this.model = model;
    }

    public double getBaseRentalRate() {
        return baseRentalRate;
    }
```

```
    public void setBaseRentalRate(double baseRentalRate) {
        if (baseRentalRate <= 0) throw new IllegalArgumentException("Base rental rate must be
greater than 0");
        this.baseRentalRate = baseRentalRate;
    }

    public boolean isAvailable() {
        return isAvailable;
    }

    public void setAvailable(boolean available) {
        isAvailable = available;
    }

    public abstract double calculateRentalCost(int days);

    public abstract boolean isAvailableForRental();
}

// Car.java (Concrete Class)
public class Car extends Vehicle implements Rentable {
    private boolean hasGps;

    public Car(String vehicleId, String model, double baseRentalRate, boolean hasGps) {
        super(vehicleId, model, baseRentalRate);
        this.hasGps = hasGps;
    }

    @Override
    public double calculateRentalCost(int days) {
        double cost = getBaseRentalRate() * days;
        if (hasGps) {
            cost += 10 * days;
        }
        return cost;
    }

    @Override
    public boolean isAvailableForRental() {
        return isAvailable();
    }
```

```
    @Override
    public void rent(Customer customer, int days) {
        if (!isAvailableForRental()) throw new IllegalStateException("Car is not available");
        customer.addRental(new RentalTransaction(customer, this, days));
        setAvailable(false);
    }

    @Override
    public void returnVehicle() {
        setAvailable(true);
    }
}

// Motorcycle.java (Concrete Class)
public class Motorcycle extends Vehicle implements Rentable {
    public Motorcycle(String vehicleId, String model, double baseRentalRate) {
        super(vehicleId, model, baseRentalRate);
    }

    @Override
    public double calculateRentalCost(int days) {
        return getBaseRentalRate() * days * 0.8;
    }

    @Override
    public boolean isAvailableForRental() {
        return isAvailable();
    }

    @Override
    public void rent(Customer customer, int days) {
        if (!isAvailableForRental()) throw new IllegalStateException("Motorcycle is not available");
        customer.addRental(new RentalTransaction(customer, this, days));
        setAvailable(false);
    }

    @Override
    public void returnVehicle() {
        setAvailable(true);
    }
```

```
}

// Truck.java (Concrete Class)
public class Truck extends Vehicle implements Rentable {
    private double cargoCapacity;

    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 + cargoCapacity * 5;
    }

    @Override
    public boolean isAvailableForRental() {
        return isAvailable();
    }

    @Override
    public void rent(Customer customer, int days) {
        if (!isAvailableForRental()) throw new IllegalStateException("Truck is not available");
        customer.addRental(new RentalTransaction(customer, this, days));
        setAvailable(false);
    }

    @Override
    public void returnVehicle() {
        setAvailable(true);
    }
}

// Rentable.java (Interface)
public interface Rentable {
    void rent(Customer customer, int days);
    void returnVehicle();
}

// Customer.java (Class)
```

```
import java.util.ArrayList;
import java.util.List;

public class Customer {
    private String name;
    private List<RentalTransaction> rentalHistory;

    public Customer(String name) {
        if (name == null || name.isEmpty()) throw new IllegalArgumentException("Name cannot be
null or empty");
        this.name = name;
        this.rentalHistory = new ArrayList<>();
    }

    public void addRental(RentalTransaction rentalTransaction) {
        rentalHistory.add(rentalTransaction);
    }

    public List<RentalTransaction> getRentalHistory() {
        return rentalHistory;
    }

    public String getName() {
        return name;
    }
}

// RentalTransaction.java (Class)
public class RentalTransaction {
    private final Customer customer;
    private final Vehicle vehicle;
    private final int days;

    public RentalTransaction(Customer customer, Vehicle vehicle, int days) {
        this.customer = customer;
        this.vehicle = vehicle;
        this.days = days;
    }

    public double getTotalCost() {
        return vehicle.calculateRentalCost(days);
```

```
    }
}

// RentalAgency.java (Class)
import java.util.ArrayList;
import java.util.List;

public class RentalAgency {
    private List<Vehicle> vehicleFleet;

    public RentalAgency() {
        vehicleFleet = new ArrayList<>();
    }

    public void addVehicle(Vehicle vehicle) {
        vehicleFleet.add(vehicle);
    }

    public Vehicle findAvailableVehicle(String model) {
        for (Vehicle vehicle : vehicleFleet) {
            if (vehicle.getModel().equals(model) && vehicle.isAvailableForRental()) {
                return vehicle;
            }
        }
        return null;
    }

    public List<Vehicle> getVehicleFleet() {
        return vehicleFleet;
    }
}

// Main.java (Main Class)
public class Main {
    public static void main(String[] args) {
        RentalAgency rentalAgency = new RentalAgency();

        Car car = new Car("CAR123", "Toyota", 50, true);
        Truck truck = new Truck("TRUCK123", "Ford", 100, 500);
        Motorcycle motorcycle = new Motorcycle("MOTO123", "Harley", 30);
```

```
        rentalAgency.addVehicle(car);
        rentalAgency.addVehicle(truck);
        rentalAgency.addVehicle(motorcycle);

        Customer customer = new Customer("John Doe");

        car.rent(customer, 3);
        System.out.println("Total rental cost for car: $" + car.calculateRentalCost(3));

        truck.rent(customer, 5);
        System.out.println("Total rental cost for truck: $" + truck.calculateRentalCost(5));

        motorcycle.rent(customer, 2);
        System.out.println("Total rental cost for motorcycle: $" +
motorcycle.calculateRentalCost(2));

        car.returnVehicle();
        truck.returnVehicle();
        motorcycle.returnVehicle();
    }
}
```