

Data Structures and Object Oriented Programming
420-SF2-RE (section 00001)

DELIVERABLE 4

VEHICLE RENTAL SYSTEM

Written by: David Ayoub

Presented to Professor Yi Wang

TABLES OF CONTENTS:

- Project Description
- Program Features
- Challenges
- Learning Outcomes

PROJECT DESCRIPTION

- The vehicle rental system is a platform where customers can rent and return vehicles such as cars and motorcycles, and employees can manage those vehicles. Customers can see what's on their dashboard and their car's details and employees can see what cars are in their fleet. After requesting to rent a car, the system shows the rental cost and checks the car's availability, which is changed once the car is rented.
- The system offers multiple services based on the user type, customer or employee:
 1. Customer
 - View available vehicles: A list of all vehicles not currently rented, showing model, year, type, and rate.
 - Search for vehicles: A refined list matching search criteria (e.g., cars after 2020, motorcycles < \$30/day).
 - Rent a vehicle: Confirmation message with rental summary: vehicle info, rental days, and total cost.
 - Return a vehicle: Success message showing return confirmation.
 - View rental history: A list of previously rented vehicles with dates and costs.
 - View rental receipt: Receipt showing rental ID, vehicle, total cost, and rental period
 2. Employee
 - Add a new vehicle: Success message: "Vehicle [plate] added successfully."
 - Remove a vehicle: Confirmation message with vehicle details removed.
 - Update vehicle information: Confirmation showing updated data (e.g., new rate or availability).
 - View all vehicles: Full list of fleet with status (rented / available), model type, and rate.
 - View rented vehicles: list of vehicles currently rented, including renter's name or ID.
 - Sort vehicles: Ordered list based on selected criteria (e.g., rate, year, status).
 - Save fleet data: Message: "Fleet data saved to file successfully."
 - Load fleet data: Message: "Fleet data loaded from file successfully."

- View rental logs: A list of all past rental transactions, including customer name, vehicle, rental period, and total cost. If no logs are available, display: "No rental logs found."

- Hierarchies:
 1. User hierarchy:
 - User (abstract super class): generic system user + attributes
 - Customer (subclass): can search, rent, return, and view vehicles
 - Employee (subclass): can add, remove, update, view, and manage vehicles

 2. Vehicle hierarchy:
 - Vehicle (abstract super class): defines common vehicle attributes
 - Car (subclass): extra car attributes
 - Motorcycle (subclass): extra motorcycle attributes

- The rentable interface is needed in this project because it's a single method to calculate rental costs for the different types of vehicles and the interface promotes polymorphism. It also simplifies the process if ever another type of vehicle were to be added, ensuring the core functionality of the code.

- Runtime polymorphism is used twice in this project: In the rentable interface and in the calculateRentalCost method

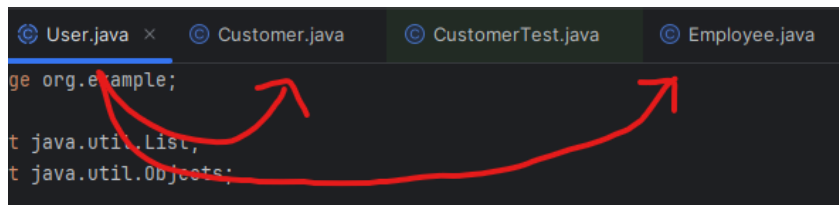
- In this project, textIO is used in the VehicleRentalSystem class to save fleet data, where it writes vehicle details to a file and load fleet data, where it reads vehicle details from a file. TextIO is also used in the RentalService class to save rental logs, writing them to a file, and loading rental logs, to read them from a file.

- Comparable will be implemented in the Vehicle class and Comparator will be used in the VehicleRentalSystem class

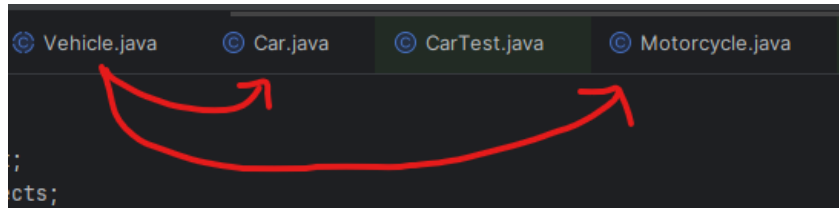
- For deliverable 2 (50%), I will implement the skeleton for User and Vehicle classes and subclasses, meaning the attributes, the setters and getters, equals method, toString and constructors as well as the comparable interface and the basic methods/actions for the users and vehicles.

PROGRAM FEATURES

The program has a variety of different features, including the many methods shown in the project description. The project meets all the requirements: it has at least two hierarchies of classes with two layers each, it has a user-defined interface, it also has a couple runtime-polymorphisms such as toString(), it contains at least one textI/O, implements comparable and has a comparator class as well as Unit testing for user defined methods.



This screenshot shows the top of an IDE with four tabs: User.java, Customer.java, CustomerTest.java, and Employee.java. The code in the active tab (User.java) shows imports for `org.example`, `java.util.List`, and `java.util.Objects`. Red arrows are drawn on the code: one from `org.example` to the `Customer.java` tab, one from `java.util.List` to the `CustomerTest.java` tab, and one from `java.util.Objects` to the `Employee.java` tab.



This screenshot shows the top of an IDE with four tabs: Vehicle.java, Car.java, CarTest.java, and Motorcycle.java. The code in the active tab (Vehicle.java) shows imports for `java.util.List` and `java.util.Objects`. Red arrows are drawn on the code: one from `java.util.List` to the `CarTest.java` tab, and one from `java.util.Objects` to the `Motorcycle.java` tab.

```
package org.example;

public interface Rentable {
    boolean rent();
    boolean returnVehicle();
}
```

```
@Override
public String toString() {
    return super.toString() + " [Trunk: " + trunkSize + " cu ft]";
}
```

```
public abstract class Vehicle implements Rentable, Comparable<Vehicle> {
```

```
@Override  
public int compareTo(Vehicle other) {  
    return Double.compare(this.rate, other.rate);  
}
```

```

// Writes vehicle information to an external file
public void saveFleetToFile(String filename) { no usages  ⚡ Damage Inc
    try (FileWriter fileWriter = new FileWriter(filename)) {
        for (Vehicle v : fleet) {
            String line = v.getClass().getSimpleName() + ", " +
                v.getPlateNumber() + ", " +
                v.getModel() + ", " +
                v.getRate() + ", " +
                v.isAvailable();
            if (v instanceof Car) {
                Car car = (Car) v;
                line += ", " + car.getTrunkSize();
            }
            else if (v instanceof Motorcycle){
                Motorcycle motorcycle = (Motorcycle) v;
                line += ", " + motorcycle.getEngineCapacity();
            }

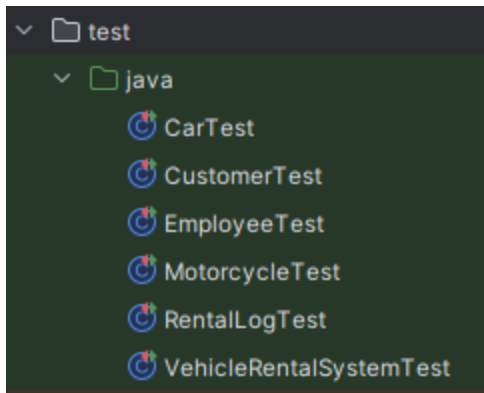
            fileWriter.write( str: line + "\n");
        }
    } catch (IOException e) {
        System.out.println("Error saving fleet: " + e.getMessage());
    }
}

```

```

// Receives vehicle information from an external file
public void loadFleetFromFile(String filename) { no usages  ⚡ Damage Inc
    try (Scanner scanner = new Scanner(new File(filename))) {
        while (scanner.hasNextLine()) {
            String[] data = scanner.nextLine().split( regex: ",");
            String type = data[0];
            String plate = data[1];
            String model = data[2];
            double rate = Double.parseDouble(data[3]);
            boolean available = Boolean.parseBoolean(data[4]);
            Vehicle v;
            if (type.equals("Car")) {
                int trunkSize = Integer.parseInt(data[5]);
                v = new Car(plate, model, rate, trunkSize: 500);
            } else {
                int engineCapacity = Integer.parseInt(data[5]);
                v = new Motorcycle(plate, model, rate, engineCapacity);
            }
            fleet.add(v);
        }
    }
}

```



```
public class Main {  Damage Inc
    public static void main(String[] args) {  Damage Inc

        // VehicleRentalSystem object
        VehicleRentalSystem system = new VehicleRentalSystem();

        // Some vehicles
        Car car = new Car(plateNumber: "A008", model: "Hyundai Elantra", rate: 70.0, trunkSize: 14);
        Motorcycle motorcycle = new Motorcycle(plateNumber: "B007", model: "Kawasaki Z650", rate: 65.0, engineCapacity: 650);

        // Add the vehicles to the fleet
        system.addVehicle(car);
        system.addVehicle(motorcycle);

        // Some customers and employees
        Customer customer = new Customer(id: "CU04", name: "Nick");
        Employee employee = new Employee(id: "EM06", name: "Matthew");

        // Display available vehicles
        System.out.println("Available vehicles for customers:");
        System.out.println(customer.viewAvailableVehicles(system.getFleet()));

        System.out.println("\nAvailable vehicles for employees:");
        System.out.println(employee.viewAvailableVehicles(system.getFleet()));

        // Rent a vehicle
        System.out.println("\nRenting a vehicle...");
        LocalDate rentDate = LocalDate.of(year: 2024, month: 5, dayOfMonth: 1);
        LocalDate returnDate = LocalDate.of(year: 2024, month: 5, dayOfMonth: 5);
        System.out.println(system.rentVehicle(customer, plate: "A008", rentDate, returnDate));

        // Return a vehicle
        system.returnVehicle(plate: "A008");
        System.out.println("\nAvailable vehicles after returning the vehicle:");
        System.out.println(customer.viewAvailableVehicles(system.getFleet()));

        // Some rental logs and durations
        RentalLog log = new RentalLog(car, customer, rentDate, returnDate);
        System.out.println("\nRental Log:");
        System.out.println(log);
        System.out.println("Rental Duration: " + log.getDuration() + " days");
```



```
"C:\Program Files\Java\jdk-20\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2024.3.3\lib\idea_rt.jar=7244:C:\Program Files
Available vehicles for customers:
Hyundai Elantra [A008] - $70.0 per day (Available) [Trunk: 14 cu ft], Kawasaki Z650 [B007] - $65.0 per day (Available) [Engine Capacity: 650cc]

Available vehicles for employees:
Hyundai Elantra [A008] - $70.0 per day (Available) [Trunk: 14 cu ft], Kawasaki Z650 [B007] - $65.0 per day (Available) [Engine Capacity: 650cc]

Renting a vehicle...
Rented: Hyundai Elantra [A008] - $70.0 per day (Rented) [Trunk: 14 cu ft]

Available vehicles after returning the vehicle:
Hyundai Elantra [A008] - $70.0 per day (Available) [Trunk: 14 cu ft], Kawasaki Z650 [B007] - $65.0 per day (Available) [Engine Capacity: 650cc]

Rental Log:
Nick (ID: CU04) rented Hyundai Elantra [A008] - $70.0 per day (Available) [Trunk: 14 cu ft] from 2024-05-01 to 2024-05-05
Rental Duration: 4 days

Process finished with exit code 0
```

CHALLENGES

I noticed uncertainty regarding the abstract class in the interface and its purpose. Another challenge I faced during the creation of this project is the unit testing because from the start I could clearly see quite a bit of it would have to be done but the problem is rather choosing which methods, then choosing which situations to test and which ones are important. Throughout the project, I also saw myself tweaking the methods and the toStrings and multiple aspects of my code I thought I was done with. Now, I understand that this is the goal of unit testing, however I did find it a bit challenging to go back and forth and solve issues that I couldn't figure out at first, without giving up.

LEARNING OUTCOMES

From this project, I gained experience on how the development of an idea actually happens in programming. While I've thought about how it would happen, I can now see how structuring your project and making it clear in your head, which was deliverable 1, is crucial to not make mistakes. I never thought about how time consuming it can be at times, especially if you become disorganized or you don't know in which direction to go next. I can also say that while we did a project in programming 1, this project requires much more understanding of how classes are linked together and how everything connects to everything so much that it's easy to get lost. I have gained a lot more respect for people working constantly on their own projects, much more complicated and elaborate. It's also helping me wait less for deadlines, because I am a big procrastinator, in the sense that it's much better to do a little bit everyday as opposed to doing deliverable 1, then waiting until the deadline day of deliverable 2 and doing everything, then waiting for deliverable 3, etc. and this also helped me not lose sight of the vision I had for the project because if I code one day and then leave the project for a week, I will likely lose my ideas.