UTS College

Semester 2, 2024

IPRO002 Project A Group Assignment

: OOP design for the car rental business

Student ID	Student Name	% contribution
14556068	**Ina Song**	65 %
14579130	**Jongmin Kim**	35 %

Declaration

I confirm that the material contained in this assignment is my own work and has not been submitted in any other subject or course. I am aware of plagiarism and its penalties, and have acknowledged all material and sources used in the preparation of this assignment.

Date: 31 / JUL / 2024

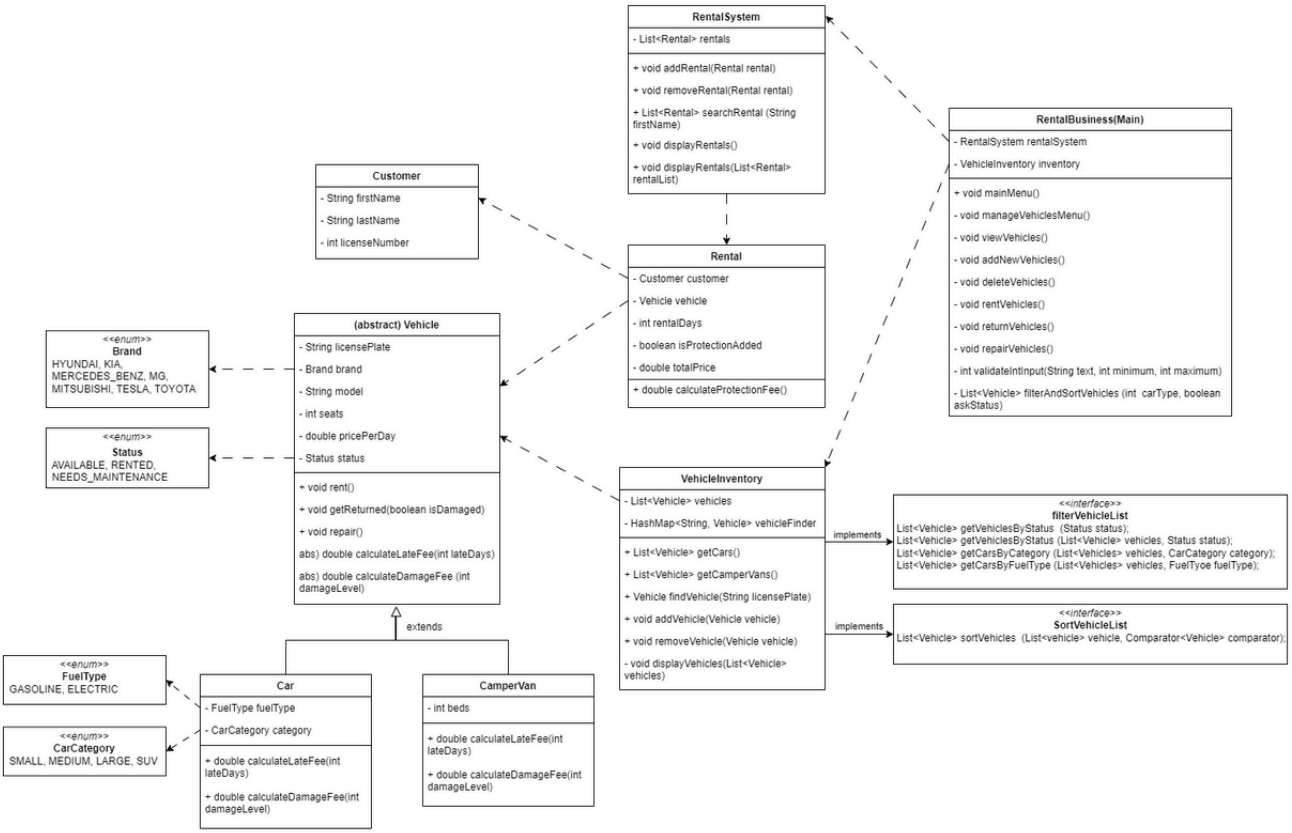
Signiture: Ina Song

Jongmin Kim

UTS College

Table of contents

Scenario



This program implements a sophisticated system for managing a vehicle rental business. It is specifically designed for use by employees of a rental car company, facilitating four operational use cases such as **managing vehicles**(view / add / delete), **renting vehicles**, **returning vehicles**, and **repairing vehicles**.

At the base, the abstract class **Vehicle** branches into two subclasses, **Car** and **CamperVan**, each representing individual vehicles in the real world. These vehicles are managed within the **VehicleInventory** class, which holds a list of these vehicles and represents the fleet of a rental car company. The VehicleInventory class is key to adding or removing vehicles from the fleet. It also offers functionalities to filter and sort this list based on various vehicle attributes, playing a critical role in efficient vehicle management.

The **Rental** class denotes a single rental transaction, capturing information about the customer and the rented vehicle, along with additional rental details. Collectively, these rental transactions are managed by the **RentalSystem** class, which efficiently organizes and maintains the rental records, making it easy to effectively handle rental and return operations.

The core classes, VehicleInventory and RentalSystem, are encapsulated within the main driver class, **RentalBusiness**. This class interacts with users through an menu interface, allowing seamless navigation across various menus to utilize the functionalities provided by core classes. This design not only simplifies the management tasks but also enriches the user experience by offering a structured and clear workflow for managing a rental business.

Explanation

1) Inheritance

In the vehicle rental management system, the abstract class Vehicle provides a foundational framework for Car and CamperVan class without being instantiated. It centralizes common properties like license plate, model, and brand, ensuring that all vehicles share a consistent set of characteristics to enhance the manageability of the vehicle database. Moreover, subclasses introduce specific attributes and behaviors, extending the Vehicle class to inherit its properties while adding unique features relevant to each vehicle type.

The Vehicle class includes two abstract methods, calculateLateFee(int lateDays) and calculateDamageFee(int damageLevel), essential for implementing varying rates of additional charges appropriate to each vehicle type upon their return. While more details on polymorphism will be discussed later, this design also allows the efficient management of both types of vehicles in the inventory.

2) Interfaces

There are two crucial interfaces in the program: **FilterVehicleList** and **SortVehicleList**. The VehicleInventory class is specifically chosen to implement the FilterVehicleList and SortVehicleList interfaces due to its central role in managing the fleet of vehicles. By implementing these interfaces, the class significantly benefits from streamlined handling of vehicle data without additional intermediary structures or classes.

The implementation of FilterVehicleList interface allows the system to filter the inventory's vehicle list based on specific needs for each functionality and user preferences, using criteria such as vehicle status, category, and fuel type. This filtering process is critical, as it ensures that users are shown only vehicles that align with their specific requirements for every use, facilitating a more focused and efficient selection process.

The implementation of SortVehicleList interface enables the system to sort the filtered list according to criteria defined by the users, such as price, model, or the number of seats. This sorting is achieved by utilizing comparators, which enables this functionality by providing a single, flexible method for sorting that can adapt to any specified attribute. This feature enhances the system's usability by efficiently organizing the vehicle options, making it easier for users to navigate their choices and select the best vehicle for their needs.

The integration of these interfaces improves system performance and usability by enabling immediate application of filters and sorting criteria, facilitating quick and accurate vehicle management.

```
interface FilterVehicleList{
    List<Vehicle> getVehiclesByStatus(Status status);
    List<Vehicle> getVehiclesByStatus(List<Vehicle> vehicles, Status status);
    List<Vehicle> getCarsByCategory(List<Vehicle> vehicles, CarCategory category);
    List<Vehicle> getCarsByFuelType(List<Vehicle> vehicles, FuelType fuelType);
}
interface SortVehicleList{
    List<Vehicle> sortVehicles(List<Vehicle> vehicles, Comparator<Vehicle> comparator);
}
```

3) Overloaded methods

Method overloading within the VehicleInventory and RentalSystem classes serves specific functions tailored to make the system more adaptable and efficient.

For VehicleInventory class, method overloading is strategically utilized to facilitate multiple layers of filtering based on different criteria. One method allows filtering directly from the inventory's entire collection of vehicles, while the overloaded method applies a filter to an already filtered list of vehicles taken as a parameter. This design addresses the need for dynamic data handling, where users can progressively apply multiple filters to the same set of vehicle data.

Similarly, the RentalSystem class incorporates overloading to enhance the display functionalities for rental records. One method displays all rentals from the system's comprehensive list, while the other overloaded method is designed to accept a subset of rentals filtered by user search, and display this specific list.

```
public List<Vehicle> getVehiclesByStatus(Status status) {
    List<Vehicle> updatedList = new ArrayList<>();
    for (Vehicle v : this.vehicles) {
        if (v.getStatus() == status) {
            updatedList.add(v);
        }
    }
    return updatedList;
}
public List<Vehicle> getVehiclesByStatus(List<Vehicle> vehicles, Status status) {
    List<Vehicle> updatedList = new ArrayList<>();
    for (Vehicle v : vehicles) {
        if (v.getStatus() == status) {
            updatedList.add(v);
        }
    }
    return updatedList;
}
```

```
void displayRentals(){
    System.out.println(x:"\n▼ ▼ ▼ ▼ ▼ ▼ Rental List ▼ ▼ ▼ ▼ ▼ ▼");
    for (int i = 0; i < this.rentals.size(); i++){
        System.out.println("[" + (i + 1) + "]\n" + rentals.get(i) + "\n");
    }
}

void displayRentals(List<Rental> rentalList){
    System.out.println(x:"\n▼ ▼ ▼ ▼ ▼ ▼ Rental List ▼ ▼ ▼ ▼ ▼ ▼");
    for (int i = 0; i < rentalList.size(); i++){
        System.out.println("[" + (i + 1) + "]\n" + rentalList.get(i));
    }
}
```

4) Overriding methods

Method overriding is employed in the Car and CamperVan subclasses to specifically implement the abstract methods 'calculateLateFee' and 'calculateDamageFee', which are declared in the Vehicle superclass. This overriding is essential to customize the calculation of fees based on the unique characteristics and associated costs of different vehicle types. Camper vans, which generally incur higher repair costs than standard cars due to their complex build, necessitate a distinct approach to fee calculation. Accordingly, the overridden methods in the CamperVan class apply higher penalty rates for damages compared to those in the Car class. This ensures that the fees reflect the financial impact of damages and late returns for the specific vehicle type.

5) Enum classes

Enum classes such as Status, Brand, CarCategory, and FuelType are utilized specifically to standardize and categorize attributes related to the vehicles in the fleet. These enums serve the purpose of enforcing consistency and reliability in how vehicle attributes are managed and utilized throughout the system. By employing enums, the system ensures that all vehicle attributes are consistently defined and managed, avoiding the operational errors associated with arbitrary string usage, such as typographical errors or inconsistent formatting. This uniformity is essential for maintaining precise control over vehicle classifications, greatly aiding in the accurate filtering and retrieval of vehicle data.

```
enum Status {
    AVAILABLE, RENTED, NEEDS_MAINTENANCE
}
enum Brand {
    HYUNDAI, KIA, MERCEDES_BENZ, MG, MITSUBISHI, TESLA, TOYOTA
}
enum CarCategory {
    SMALL, MEDIUM, LARGE, SUV
}
enum FuelType{
    ELECTRIC, GASOLINE
}
```

6) Polymorphism

The VehicleInventory class utilizes polymorphism by managing Car and CamperVan objects within a single list of type Vehicle. This approach allows for comprehensive inventory management by handling all vehicle types through a unified list representing entire inventory. Additionally, the 'instanceof' keyword is used in some methods to distinguish between Car and CamperVan objects in the list, enabling the system to manage and process only the vehicle types needed by the user. This design facilitates flexibility in processing and retrieving vehicles based on their type while also maintaining an overarching management system. It reduces the need for separate methods for each vehicle type that can lead to code redundancy, enhancing both efficiency and maintainability.

```
List<Vehicle> getCars() {
    List<Vehicle> carList = new ArrayList<>();
    for (Vehicle v : this.vehicles){
        if (v instanceof Car){
            carList.add(v);
        }
    }
    return carList;
}
```

7) Boilerplates Reduction

The rental business system offers users numerous choices, requiring integer inputs at 37 different points. Instead of implementing a loop for each input instance, the program significantly reduces repetitive structures by using a single validateIntInput method to manage all integer inputs. This method interacts with the user by displaying a prompt message and ensuring that the input is a valid integer within a specified range. If the input is invalid, it repeatedly displays an error message and re-prompts the user until a valid input is provided. The validateIntInput method is extensively utilized throughout the RentalBusiness class, effectively streamlining the code and improving maintainability by consolidating the input validation logic into one reusable method.

```java
private int validateIntInput(String text, int minimum, int maximum){
    int input;
    while(true){
        try{
            System.out.print(text);
            input = In.nextInt();
            if (input >= minimum && input <= maximum){
                break;
            }
            System.out.println("\n[ Invalid input: please enter a number from " + minimum + " to " + maximum +". ]");

        } catch (InputMismatchException e){
            System.out.println("\n[ Invalid input: please enter an integer value (" + minimum + "-" + maximum +"). ]");
            In.nextLine();
        }
    }
    return input;
}
```

```
==================================
      VEHICLE RENTAL SYSTEM
==================================
       1. Manage vehicles
       2. Rent vehicles
       3. Return vehicles
       4. Repair vehicles
       5. Exit system
==================================
 ▷Select a menu (1-5): s

[ Invalid input: please enter an integer value (1-5). ]
 ▷Select a menu (1-5): 6

[ Invalid input: please enter a number from 1 to 5. ]
 ▷Select a menu (1-5): 1
```

```
==================================
      VEHICLE RENTAL SYSTEM
==================================
       1. Manage vehicles
       2. Rent vehicles
       3. Return vehicles
       4. Repair vehicles
       5. Exit system
==================================
 ▷Select a menu (1-5): 2

■ STEP1: CUSTOMER INFORMATION
 ▷Enter the customer's first name: Ina
 ▷Enter the customer's last name: Song
 ▷Is the customer over 21 and holding full license? (1- yes, 2- no): 1
 ▷Enter the customer's license number(8 digit): 1234567

[ Invalid input: please enter a number from 10000000 to 99999999. ]
 ▷Enter the customer's license number(8 digit): 12345678
```

In addition, use cases such as viewing, renting, and deleting vehicles in the RentalBusiness class follow a similar pattern of filtering and sorting vehicles based on user choices. To streamline this process, the filterAndSortVehicles method was created. This method takes user inputs to filter and sort the vehicle list according to specified criteria, then returns the filtered and sorted list. By consolidating these repetitive tasks into a single method, the program reduces code redundancy and enhances maintainability, ensuring that the filtering and sorting logic is applied consistently across different use cases.

8) Vehicle Lookup with HashMap

In the VehicleInventory class, a HashMap is utilized to map license plates to their corresponding Vehicle objects. This design allows for efficient retrieval and verification of vehicles without the need to iterate through the entire list. The findVehicle method leverages this HashMap to quickly retrieve a specific vehicle by its license plate number or to check its existence in the inventory. This approach significantly enhances performance and reduces the complexity of operations that require vehicle lookups.

```
vehicleFinder = new HashMap<>();
for(Vehicle v : getAllVehicles()){
    this.vehicleFinder.put(v.LICENSE_PLATE, v);
}
```

```
Vehicle findVehicle(String licensePlate){
    Vehicle v = vehicleFinder.get(licensePlate);
    return v;
}
```

```
System.out.print(s:" ▷ Enter the license plate (3 Capitalised Alphabet + 3 Numbers): ");
licensePlate = In.nextLine();
if (inventory.getVehicleFinder().containsKey(licensePlate)){
    System.out.println("[ The car '" + licensePlate + "' already exists in the system. ]");
    return;
}
```

```
  ==========================
        Delete Vehicles
  ==========================
     1. Enter license plate
     2. Select from the list
  ==========================
   ▷Select the option (1-2): 1
   ▷Enter the license plate (3 Capitalised Alphabet + 3 Numbers): YUP649

  Selected Vehicle:
  ..........................................
    TOYOTA / Rav-4 (YUP649)
     - GASOLINE
     - SUV
     - 7 seats
     - $107.8 /day
  ..........................................
   ▷Do you want to delete this vehicle? (1- delete, 2- cancel): 1
   [ TOYOTA Rav-4 is deleted from the system. ]
```

```
============================
        Add vehicles
============================
     1. Add a car
     2. Add a camper van
============================
 ▷Select a menu (1-2): 1
 ▷Enter the license plate (3 Capitalised Alphabet + 3 Numbers): FJC719
[ The car 'FJC719' already exists in the system. ]
```

Program Execution

1) Add/delete vehicles

```
============================
        Add vehicles
============================
     1. Add a car
     2. Add a camper van
============================
 ▷Select a menu (1-2): 2
 ▷Enter the license plate (3 Capitalised Alphabet + 3 Numbers): HUS293

-----------------------------------
   1. Hyundai         5. Mitsubishi
   2. KIA             6. Tesla
   3. Mercedes Benz   7. Toyota
   4. MG
-----------------------------------
 ▷Select the brand (1-7): 7
 ▷Enter the model name: SpeedyCamper
 ▷Enter the number of seats: 3
 ▷Enter the price per day ($): 115.5
 ▷Enter the number of beds: 2

[ The vehicle is successfully added to the system. ]
▶TOYOTA / SpeedyCamper [HUS293]
   - 3 seats
   - 2 beds
   - $ 115.5 /day

 ▷Do you want to add more cars? (1- yes, 2- no): 2
```

```
============================
      Manage Vehicles
============================
    1. View vehicles
    2. Add new vehicles
    3. Delete vehicles
    4. Back to main menu
============================
 ▷Select a menu (1-4): 3

============================
      Delete Vehicles
============================
   1. Enter license plate
   2. Select from the list
============================
 ▷Select the option (1-2): 1
 ▷Enter the license plate (3 Capitalised Alphabet + 3 Numbers): HUS293

Selected Vehicle:
-----------------------------------------
  TOYOTA / SpeedyCamper [HUS293]
   - 3 seats
   - 2 beds
   - $ 115.5 /day
-----------------------------------------
 ▷Do you want to delete this vehicle? (1- delete, 2- cancel): 1
 [ TOYOTA SpeedyCamper is deleted from the system. ]
```

2) Rent vehicles

```
■ STEP1: CUSTOMER INFORMATION
 ▷Enter the customer's first name: Ina
 ▷Enter the customer's last name: Song
 ▷Is the customer over 21 and holding full license? (1- yes, 2- no): 1
 ▷Enter the customer's license number(8 digit): 14556068

■ STEP2: CAR SELECTION
============================
    1. Rent a car
    2. Rent a camper van
============================
 ▷Select a menu (1-2): 1

----------------------------
         Fuel Type
----------------------------
       1. All
       2. Gasoline
       3. Electric
----------------------------
 ▷Select a fuel type (1-3): 3

----------------------------
         Category
----------------------------
         1. All
         2. Small
         3. Medium
         4. Large
         5. SUVs
----------------------------
 ▷Select a category (1-5): 3
```

```
----------------------------
        How to Sort
----------------------------
    1. Sort by brand/model
    2. Sort by price
    3. Sort by seats
----------------------------
 ▷Select an option (1-3): 2

▼ ▼ ▼ ▼ ▼ ▼ Vehicle List ▼ ▼ ▼ ▼ ▼ ▼
1. MG / MG-4 EV (LRT815)
   - ELECTRIC
   - MEDIUM
   - 5 seats
   - $70.9 /day
2. HYUNDAI / Ioniq 5 (QFJ658)
   - ELECTRIC
   - MEDIUM
   - 5 seats
   - $90.2 /day
3. TESLA / Model-3 (XNC473)
   - ELECTRIC
   - MEDIUM
   - 5 seats
   - $109.3 /day

▷Select the vehicle (1-3 / press 0 to cancel): 2

Selected Vehicle:
```

```
Selected Vehicle:
------------------------------------------
  HYUNDAI / Ioniq 5 (QFJ658)
    - ELECTRIC
    - MEDIUM
    - 5 seats
    - $90.2 /day
------------------------------------------
 ▷Do you want to proceed with this vehicle? (1- yes, 2- no): 1

■ STEP3: RENTAL INFORMTAION
 ▷Enter the number of rental days: 6
----------------------------
      PROTECTION OPTION
 - $15 per day
 - Cover up to $10,000 fee
   for any damage or theft
----------------------------
 ▷Apply protection option? (1- yes, 2- no): 1
```

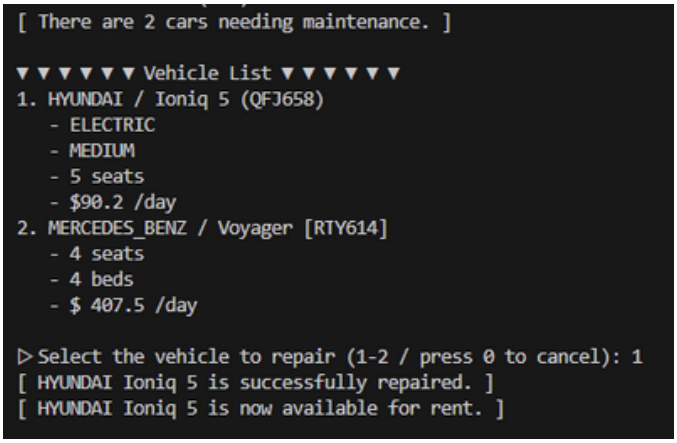
```
■ STEP4: CONFIRMATION
------------------------------------------
 [Customer] Ina Song
   - license number: 14556068
 [Car] HYUNDAI / Ioniq 5 (QFJ658)
   - ELECTRIC
   - MEDIUM
   - 5 seats
   - $90.2 /day
 [Rental Days] 6 days
 [Protection Fee] $ 90.0
 ▶Total Price: $631.2
------------------------------------------
 ▷Do you confirm this rental (1- confirm, 2- cancel): 1
[ The rental is successfully confirmed. ]
```

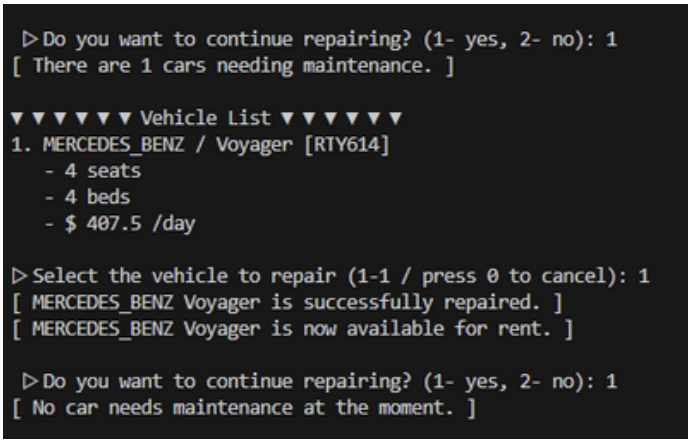
3) Return vehicles

```
■ STEP1: RENTAL SELECTION
==========================
    1. Search rental
    2. View all rentals
==========================
 ▷Select a menu (1-2): 1
 ▷Enter the customer's first name: Ina

[ 2 results found with this first name. ]

▼▼▼▼▼▼ Rental List ▼▼▼▼▼▼
[1]
------------------------------------------
 [Customer] Ina Song
   - license number: 14556068
 [Car] HYUNDAI / Ioniq 5 (QFJ658)
   - ELECTRIC
   - MEDIUM
   - 5 seats
   - $90.2 /day
 [Rental Days] 6 days
 [Protection Fee] $ 90.0
 ▶Total Price: $631.2
------------------------------------------
[2]
------------------------------------------
 [Customer] Ina Kim
   - license number: 25789874
 [Camper Van] TOYOTA / HiAce HI5 [QPR123]
   - 4 seats
   - 2 beds
   - $ 120.0 /day
 [Rental Days] 10 days
 [Protection Fee] $ 0.0
 ▶Total Price: $1200.0
------------------------------------------

 ▷Select the rental (1-2 / press 0 to cancel): 1
```

```
Selected Rental:
------------------------------------------
 [Customer] Ina Song
   - license number: 14556068
 [Car] HYUNDAI / Ioniq 5 (QFJ658)
   - ELECTRIC
   - MEDIUM
   - 5 seats
   - $90.2 /day
 [Rental Days] 6 days
 [Protection Fee] $ 90.0
 ▶Total Price: $631.2
------------------------------------------
 ▷Do you want to return this rental? (1- yes, 2- cancel): 1

■ STEP2: FEE CALCULATION
 ▷Is the vehicle being returned late? (1- yes/ 2- no): 1
 ▷How many days is the vehicle overdue? 2
 ▷Is there any damage to the vehicle? (1- yes/ 2- no): 1

-----------------------------------
  1. Minor damage
     - small scratches or dents
  2. Moderate damage
     - larger dents, broken mirrors
  3. Severe damage
     - major bodywork
-----------------------------------
 ▷Select the damage level (1-3): 3
[ The damage fee is $496.1, but it is covered by the protection plan. ]

Additional Fee:
------------------------------------------
  - Late Fee: $ 216.48
  - Damage Fee: $ 496.1
                (covered by protection)
  ▶Total Fee: $ 216.48
------------------------------------------
 ▷Do you confirm this return (1- confirm, 2- cancel): 1
[ The vehicle is successfully returned. ]
```

4) Repair Vehicles





List of Contribution

Ina Song:

- Designed class structure and made class diagram
- Wrote RentalBusiness(driver) class, designing menu interface
- Wrote VehicleInventory, Rental, RentalSystem classes
- Wrote filterVehicleList, sortVehicleList interface
- Wrote Scenario in the report
- Wrote Explanations in the report

Jongmin Kim:

- Wrote enum Status, enum CarCategory, enum Brand classes
- Wrote Vehicle, Car, CamperVan, Customer classes
- Did research about how real-world rental business work
- Created all vehicle objects for initial inventory of the system
- Tested all functionalities thoroughly and pointed out some possible errors.