



GROUP ASSIGNMENT

TECHNOLOGY PARK MALAYSIA

CT038-3.5-2-ODJ

OBJECT-ORIENTED DEVELOPMENT WITH JAVA

TEAM NAMES: LAM WENG YEW (TP065720)

**PEDRO FABIAN OWONO ONDO MANGUE
(TP063251)**

HANDOUT DATE: 19 SEPTEMBER 2022

HAND IN DATE: 16 DECEMBER 2022

TABLE OF CONTENTS

No	Contents	Page No
1	Introduction	3
2	Design	4
2.1	• Use case diagram	4
2.2	• Class diagram	6
3	Implementation of OOP Concepts	7
3.1	• Admin & Customer login & registration (PEDRO FABIAN)	7
3.2	• Customer booking page & Admin pages (LAM WENG YEW)	9
4	Demonstration with screenshots	14
5	Conclusion	27
6	References	28

1. INTRODUCTION

The problem to be highlighted is the need for a car rental application system to facilitate the customer user in accessing a more advanced technology such as an efficient application in which all types of transactions can be carried out as well as obtaining cars from an application specialised in this type of management.

As time has progressed, technology and business development needs have increased, both for large companies and small industrialists. Therefore, my teammate and I have come up with a convenient and necessary solution to innovate and contribute to the development of a Java NetBeans GUI application that would be useful for the society. Our application is based on car rental whereby two types of users interact with each other. These users are the actors and the definition of our GUI program.

The first and main user is the administrator who provides the following features:

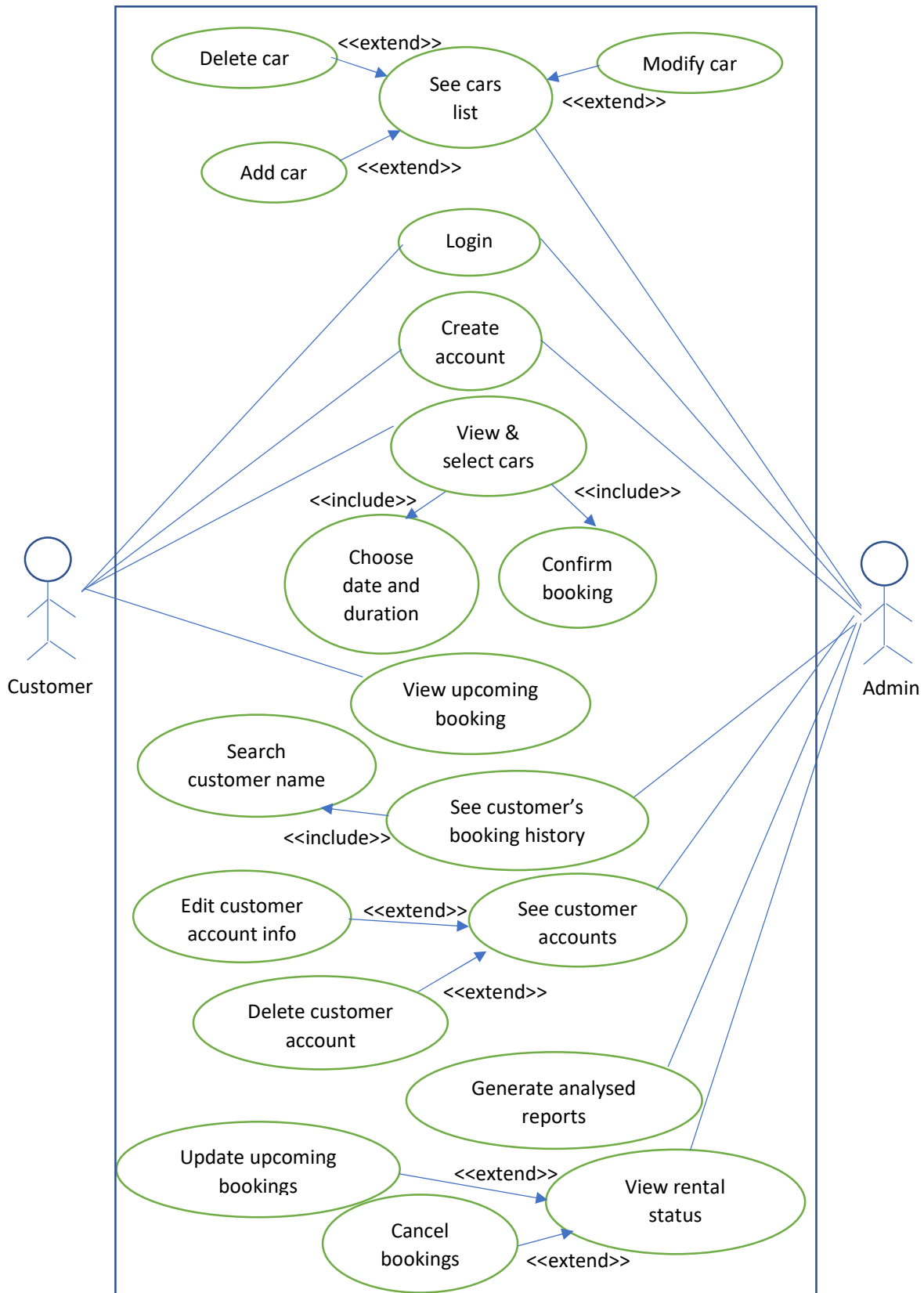
- Create new administrators.
- Regulate the car rental system through the implementation of adding new cars to the system so that the customer has access to different options or models of cars.
- Generate a report on the information of the system in terms of earnings, number of customers who have used the system, etc.
- Change specific customer information such as contact or email address.
- Verify whether the customer has returned the car after the rental as well as confirming that the customer has paid the rental in the system.
- Cancel the customer's service once the rental period has expired.
- Finally, search the customer's history to check the customer's data.

The next user and the main basis of interaction of the program is the customer with the following characteristics:

The system provides the facility for the customer user to create an account. Once the account is created, the customer will have access to the choice of the car, dates of rental, etc. On the other hand, the customer user has the option to check the cars that are already being used by other customers which he cannot access because they are being used at that moment.

2. DESIGN

USE CASE DIAGRAM



From the use case diagram above, there are 2 actors. One is customer and another one is admin. Both actors can login and create accounts. But after that, both have different use cases.

Let's start with customer. Customer can login then view, select, and book cars. In order to do that, the customer must choose date and duration as well as confirm their booking. So, both use cases must be included with the viewing and selecting cars use case. Customer can also view their booking history.

Next is admin. The admin can see the cars list, after that, the admin has 3 choices. Whether to add new car, modify car, or delete car. These 3 use cases are an extension of viewing the cars list. Because admin must view cars list first before doing any of the 3 use cases.

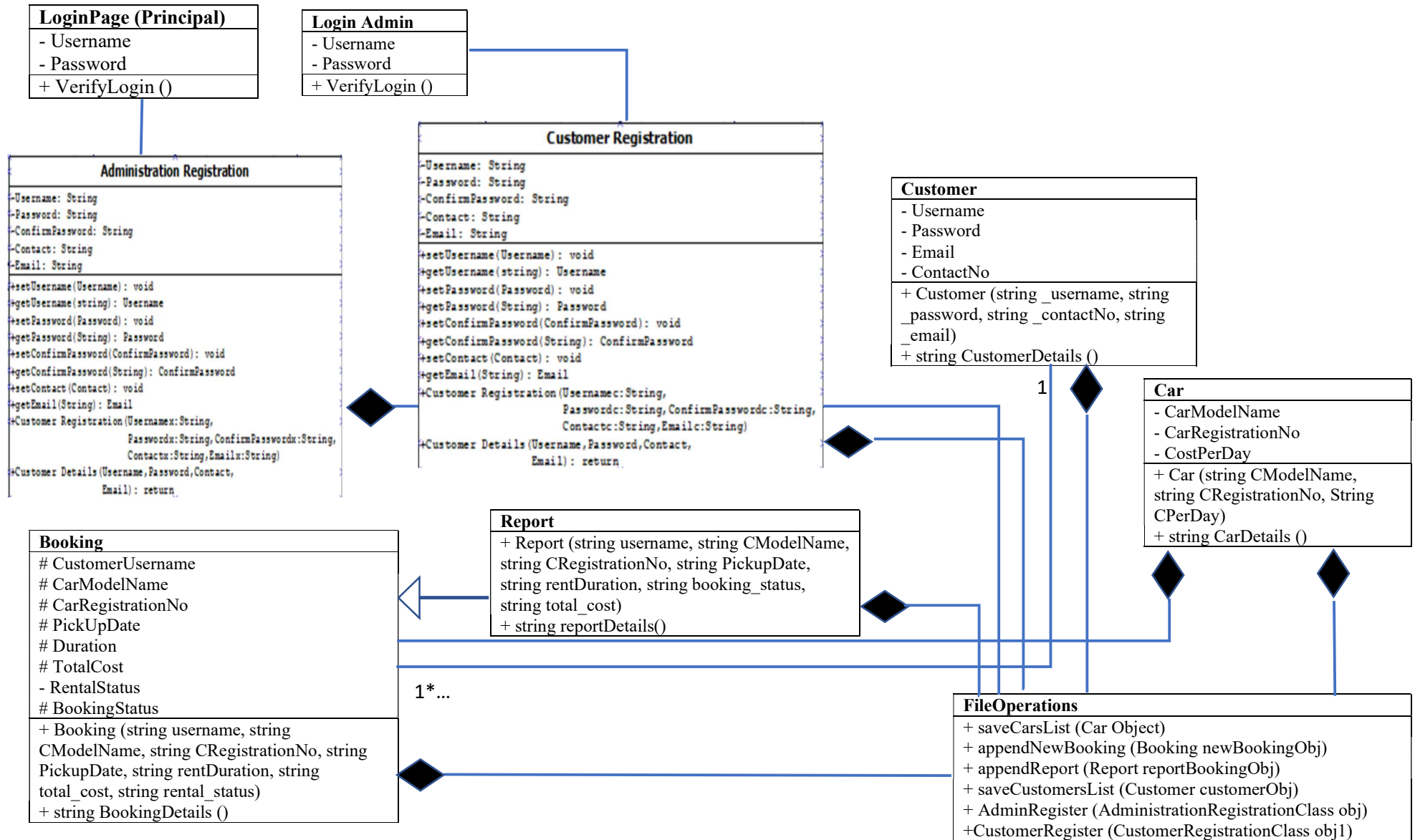
The admin can see customer's booking history. But to do that, there will be a use case where the admin must enter customer's username to find that name. So, the search customer use case is included inside seeing customer's booking history use case.

The admin can also see all customer accounts. But after that, the admin has 2 choices. Whether to modify customer account details or delete the account. These 2 use cases are an extension of seeing all customer accounts.

The admin can generate analysed report, which is a standalone use case.

The admin can view rental status of bookings. After that, the admin has 2 choices, either update the bookings or cancel the bookings. So, these 2 use cases are an extension of viewing rental status, because the admin must view the rental status first before doing the 2 use cases. The admin can also choose to do nothing after viewing the rental status.

CLASS DIAGRAM



3. IMPLEMENTATION OF OOP CONCEPTS

3.1 Admin & Customer login & registration (Pedro Fabian's part)

In the car rental programme, OOP has been implemented and the techniques are as follows:

The aggregation relationship by composition has been implemented in the program by joining a class and a JFrame as shown in the following in the next two representations(pictures).

```
class AdministrationRegistrationClass {
    private String Username;
    private String Password;
    private String ConfirmPassword;
    private String ContactNumber;
    private String Email;

    public AdministrationRegistrationClass(String Usernamex, String Passwordx, String ConfirmPasswordx, String ContactNumberx, String Emailx) {
        Username = Usernamex;
        Password = Passwordx;
        ConfirmPassword = ConfirmPasswordx;
        ContactNumber = ContactNumberx;
        Email = Emailx;
    }
}
```

In this same composition relation, it has been possible to create objects with the aim of connecting the class and the JFrame in such a way that it is possible to write in a text file when calling or invoking these created objects that come from the constructor in administration class.

```

public static void AdminRegister(AdministrationRegistrationClass obj) {

    try {
        FileWriter adminfw = new FileWriter( fileName:"adminAccount.txt", append:true);
        BufferedWriter bw = new BufferedWriter( out: adminfw);
        bw.write( str:obj.AdminDetails());
        bw.newLine();
        bw.close();
        adminfw.close();
        JOptionPane.showMessageDialog( parentComponent:null, message: "Registered successfully");

    }

}

if (txtAdminPassword.getText().equals( anObject:txtAdminConfirmPassword.getText())) {
    AdministrationRegistrationClass object = new AdministrationRegistrationClass( Username:txtAdminUsername.getText(), Password:txtAdminPassword.getText()
    FileOperations.AdminRegister( obj:object);
}

public String AdminDetails(){
    return Username + "," + Password + "," + ContactNumber + "," + Email;
}

```

The following code has to do with reading in the text file by which both the username and password are being read to access the system. First the input is taken from jframe for the username and password. The scanner scanner that is connected to the while loop is used to

```

String usernamex = txtUsername.getText();
String passwordx = txtPassword.getText();
try
if(username.equals( anObject:usernamex.trim()) && password.equals( anObject:passwordx.trim()))
{
    Scanner scan = new Scanner( source:newfile);
    scan.useDelimiter( pattern: "[, \n]");

    while(scan.hasNext())
    {
        String username =scan.next();
        String password = scan.next();
    }
}

```

check inside the text file to match the password and username that was initially taken.

The if and else is being used to confirm and validate both the password and the username so that both matches.

The method next () is for any String that is being read in the txt file.

3.2 Customer booking page & Admin pages (Weng Yew's part)

The classes that I am responsible for are:

- Car
- Booking
- Customer
- Report
- FileOperations

jFrames that I am responsible for are:

- customer_BookingPage
- admin_bookingManagementPage
- Admin_EditCustomer_AddCars
- analysedReport

I had implemented several object-oriented programming concepts in my parts.

ENCAPSULATION, AGGREGATION & COMPOSITION

```
private void ButtonSaveCarsListActionPerformed(java.awt.event.ActionEvent evt) {
    DefaultTableModel model = (DefaultTableModel) this.TableCarList.getModel();
    if (importCarsDataClicked == 0) {
        JOptionPane.showMessageDialog(parentComponent: null, message: "You haven't import data yet");
    } else {
        try {
            FileWriter fw = new FileWriter("carlist.txt");
            PrintWriter pw = new PrintWriter(out: fw, autoFlush: false);
            pw.flush(); pw.close(); fw.close();
        } catch (IOException ex) {}

        FileOperations FOP = new FileOperations();
        for (int carListRow = 0; carListRow < model.getRowCount(); carListRow++) {
            Object carModelName = TableCarList.getModel().getValueAt(rowIndex: carListRow, columnIndex: 0);
            Object carRegistrationNo = TableCarList.getModel().getValueAt(rowIndex: carListRow, columnIndex: 1);
            Object CostPerDay = TableCarList.getModel().getValueAt(rowIndex: carListRow, columnIndex: 2);
            Car carInfo = new Car(CModelName: carModelName.toString(), CRegistrationNo: carRegistrationNo.toString(), CPerDay: CostPerDay.toString());
            FOP.saveCarsList(CObject: carInfo);
        }
        JOptionPane.showMessageDialog(parentComponent: null, message: "Changes had been saved");
    }
}
```

```

public class Car {
    private String CarModelName;
    private String CarRegistrationNo;
    private String CostPerDay;

    public Car() {
    }

    public Car(String CModelName, String CRegistrationNo, String CPerDay) {
        CarModelName = CModelName;
        CarRegistrationNo = CRegistrationNo;
        CostPerDay = CPerDay;
    }

    public String CarDetails() {
        return CarModelName + ", " + CarRegistrationNo + ", " + CostPerDay + "\n";
    }
}

```

```

public class FileOperations {

    public void saveCarsList(Car CObject) {
        try {
            File file = new File( pathname: "carlist.txt");
            FileWriter fw = new FileWriter(file, append: true);
            BufferedWriter bw = new BufferedWriter( out: fw);
            bw.write( str: CObject.CarDetails());
            bw.close();
            fw.close();
        } catch (IOException ex) {}
    }
}

```

Here, I had used encapsulation, aggregation, and composition. Encapsulation is to hide file operations coding from the JFrame coding section. For the screenshots above, I used 2 classes to save data from the jTable to the text file. In the ButtonSaveCarsListActionPerformed section, it is a button to overwrite data into the “carlist.txt” file. First, it empties the “carlist.txt” file using PrintWriter. Then it creates an object to for FileOperations class, which is name FOp. Then I used For Loop to read every row of the table. For each row, I converted every cell into an object. Then, I assigned those objects to “carInfo” constructor which is from Car class. Then, I send “carInfo”, which contains all the assigned variables from a specific row of the table, over to FileOperations class, to the “saveCarsList” method. In this method, an object called “CObject” is created for Car class. It will use it to find CarDetails () method in the Car class, which will return the order and position of each attribute that must be printed into the “carlist.txt”.

JFrame class and Car class connected with each other using objects and sending data from JFrame to the constructor using objects are called Aggregation. The FileOperations class using objects to connect with the JFrame class and Car class and get data from them are called Composition. FileOperations cannot function without the JFrame and the Car class.

I used this encapsulation, aggregation and composition concept every time I want to save any jTable to text file. Here is another one:

```
private void ButtonSaveCustomersListActionPerformed(java.awt.event.ActionEvent evt) {
    DefaultTableModel model = (DefaultTableModel) this.TableCustomersList.getModel();
    if (importCustomersListClicked == 0) {
        JOptionPane.showMessageDialog(parentComponent: null, message: "You haven't import data yet");
    } else {
        try {
            FileWriter fw = new FileWriter(fileName: "customerAccount.txt");
            PrintWriter pw = new PrintWriter(out: fw, autoFlush: false);
            pw.flush(); pw.close(); fw.close();
        } catch (IOException ex) {}

        FileOperations FOp = new FileOperations();
        for (int customerListRow = 0; customerListRow < model.getRowCount(); customerListRow++) {
            Object username = TableCustomersList.getModel().getValueAt(rowIndex: customerListRow, columnIndex: 0);
            Object password = TableCustomersList.getModel().getValueAt(rowIndex: customerListRow, columnIndex: 1);
            Object contactNo = TableCustomersList.getModel().getValueAt(rowIndex: customerListRow, columnIndex: 2);
            Object email = TableCustomersList.getModel().getValueAt(rowIndex: customerListRow, columnIndex: 3);
            Customer customerInfo = new Customer(_username: username.toString(), _password: password.toString(), _contactNo: contactNo.toString(), _email: email.toString());
            FOp.saveCustomersList(customerObj: customerInfo);
        }
        JOptionPane.showMessageDialog(parentComponent: null, message: "Changes had been saved");
    }
}
```

```
public class Customer {
    private String Username;
    private String Password;
    private String Email;
    private String ContactNo;

    public Customer() {}

    public Customer(String _username, String _password, String _contactNo, String _email) {
        Username = _username;
        Password = _password;
        ContactNo = _contactNo;
        Email = _email;
    }

    public String CustomerDetails() {
        return Username + ", " + Password + ", " + ContactNo + ", " + Email + "\n";
    }
}
```

```
public void saveCustomersList(Customer customerObj) {
    try {
        File file = new File(pathname: "customerAccount.txt");
        FileWriter fw = new FileWriter(file, append: true);
        BufferedWriter bw = new BufferedWriter(out: fw);
        bw.write(str: customerObj.CustomerDetails());
        bw.close();
        fw.close();
    } catch (IOException ex) {}
}
```

For this one, I will append the “customerAccount.txt” after the admin had updated the customer accounts from the table. It empties the “customerAccount.txt” first. I created an object for FileOperations class called “FOp”. Then I used for loop to read every row of the table. I assigned every cell in that row to an object. Then I assign every object to an attribute from Customer class using a constructor in Customer class, and named it “customerInfo”. Then, using “FOp”, I send “customerInfo” to saveCustomersList () method in FileOperations class. There, it creates an object for Customer class called “customerObj”. Then uses it to call a method in Customer class named CustomerDetails(). This method will append the attributes which contains the data into the “customerAccount.txt”.

Customer class and JFrame class uses objects to connect with each other and also send data to each other using objects, particularly from JFrame to a constructor in Customer class. This is called Aggregation. FileOperations class cannot function without the JFrame class and the Customer class. So, this is Composition.

INHERITANCE & OVERRIDING (POLYMORPHISM)

```
public class Booking {
    protected String CustomerUsername;
    protected String CarModelName;
    protected String CarRegistrationNo;
    protected String PickupDate;
    protected String Duration;
    protected String TotalCost;
    private String RentalStatus;
    protected String BookingStatus;

    public Booking() {
    }

    public Booking(String username, String CModelName, String CRegistrationNo, String PickupDate, String rentDuration, String total_cost, String rental_status){
        CustomerUsername = username;
        CarModelName = CModelName;
        CarRegistrationNo = CRegistrationNo;
        PickupDate = PickupDate;
        Duration = rentDuration;
        TotalCost = total_cost;
        RentalStatus = rental_status;
    }

    public String BookingDetails(){
        return CustomerUsername + "," + CarModelName + "," + CarRegistrationNo + "," + PickupDate + "," + Duration + "," + TotalCost + "," + RentalStatus + "\n";
    }
}
```

```
public class Report extends Booking {

    public Report(String username, String CModelName, String CRegistrationNo, String PickupDate, String rentDuration, String booking_status, String total_cost){
        CustomerUsername = username;
        CarModelName = CModelName;
        CarRegistrationNo = CRegistrationNo;
        PickupDate = PickupDate;
        Duration = rentDuration;
        BookingStatus = booking_status;
        TotalCost = total_cost;
    }

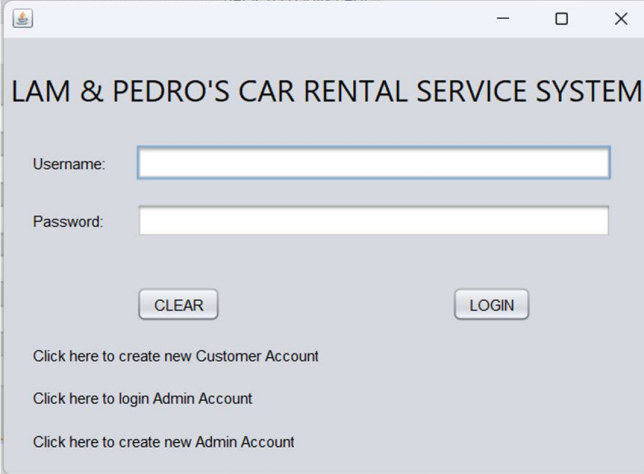
    @Override
    public String BookingDetails(){
        return CustomerUsername + "," + CarModelName + "," + CarRegistrationNo + "," + PickupDate + "," + Duration + "," + BookingStatus + "," + TotalCost + "\n";
    }
}
```

The Report class is to save data into the “report.txt”. It uses the same Encapsulation concept as the previous parts. I used Inheritance for Booking class and Report class. This is because all attributes which are needed by Report class are available inside Booking class. So, to save time and make coding easier, I applied Inheritance concept between these 2 classes. Booking class is the superclass and Report class is the subclass.

I had also used Overriding, which is a part of Polymorphism, inside the Report subclass. You can see in Booking class and Report class, there are 2 methods which are the same name, BookingDetails(). Whichever method that will be executed depends on what class the coding calls. If there is an object for Booking class called BObj, then BObj.BookingDetails() will call

the one in Booking class. If there is an object for Report class called RObj, then RObj.BookingDetails() will call the one in Report class.

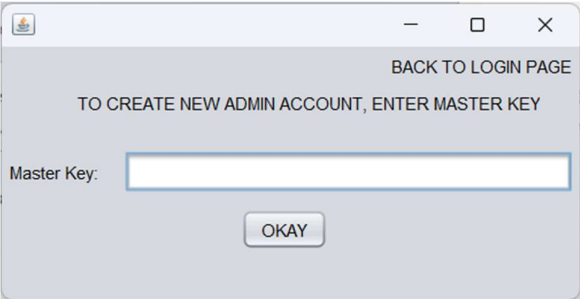
4. DEMONSTRATION WITH SCREENSHOTS



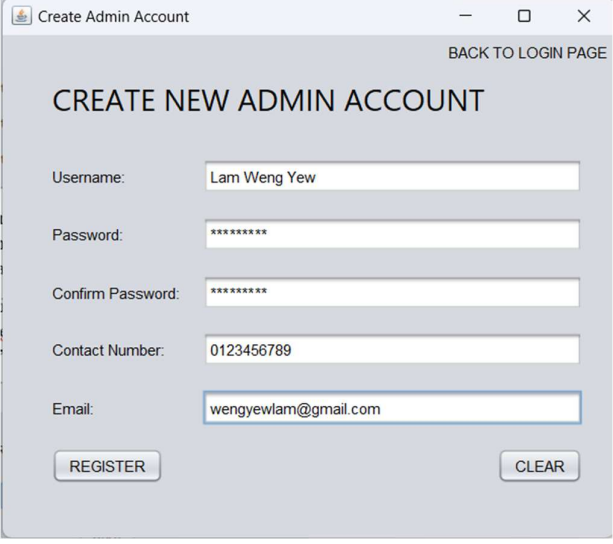
The screenshot shows a Java Swing window titled "LAM & PEDRO'S CAR RENTAL SERVICE SYSTEM". It features a login form with two text input fields labeled "Username:" and "Password:". Below these fields are two buttons: "CLEAR" and "LOGIN". At the bottom of the window, there are three links: "Click here to create new Customer Account", "Click here to login Admin Account", and "Click here to create new Admin Account".

When we run the program, the first JFrame we are going to see first is this one, the Login Page. From here, there are 4 things we can do. We can login as customer, create new customer account, login as admin, or create new admin account.

We are going to create new admin account first. Click the last label, which will bring you to this JFrame. It will ask for Master Key. Only admins know the Master Key. The Master Key is "admin".

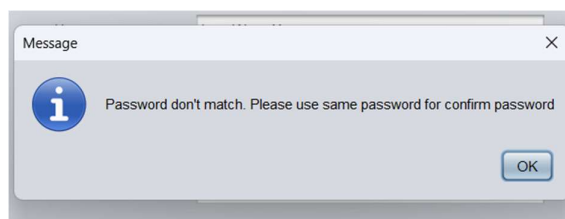


The screenshot shows a Java Swing window titled "BACK TO LOGIN PAGE". It contains the text "TO CREATE NEW ADMIN ACCOUNT, ENTER MASTER KEY". Below this text is a text input field labeled "Master Key:". At the bottom of the window is an "OKAY" button.

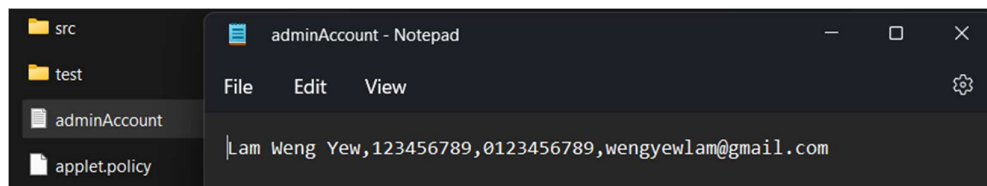


A screenshot of a Java Swing window titled "Create Admin Account". It has a standard title bar with minimize, maximize, and close buttons. In the top right corner, there is a link that says "BACK TO LOGIN PAGE". The main heading is "CREATE NEW ADMIN ACCOUNT". Below this, there are five text input fields: "Username:" with the text "Lam Weng Yew", "Password:" with masked characters "*****", "Confirm Password:" with masked characters "*****", "Contact Number:" with the text "0123456789", and "Email:" with the text "wengyewlam@gmail.com". At the bottom, there are two buttons: "REGISTER" on the left and "CLEAR" on the right.

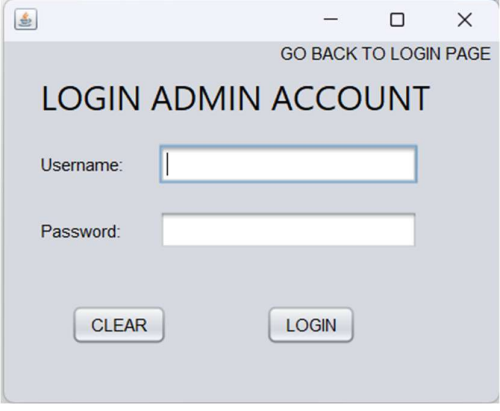
After entering Master Key, it will bring you to the JFrame where you can put in your credentials as an admin. For this demonstration, we will use Lam Weng Yew as admin. If passwords are not equal, there will be a pop-up that warns you.



If passwords match, admin account will be created and appended in the "adminAccount.txt"

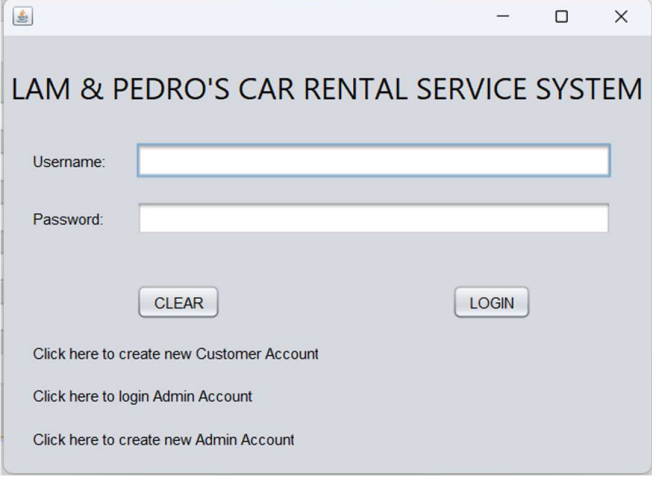


Then, it will bring us to the Login Admin JFrame. From here, Lam Weng Yew can login as admin.



A screenshot of a Java Swing window titled "LOGIN ADMIN ACCOUNT". At the top right, there is a link "GO BACK TO LOGIN PAGE". The window contains two text input fields: "Username:" and "Password:". Below these fields are two buttons: "CLEAR" and "LOGIN".

Before doing that, back to the main Login Page. We are going to create new customer account. Click the label that creates new customer account.



A screenshot of a Java Swing window titled "LAM & PEDRO'S CAR RENTAL SERVICE SYSTEM". It features "Username:" and "Password:" input fields, and "CLEAR" and "LOGIN" buttons. Below the buttons, there are three links: "Click here to create new Customer Account", "Click here to login Admin Account", and "Click here to create new Admin Account".

It will bring you to Create New Customer Account JFrame. We are going to use Pedro Fabian as the customer. After inserting all his credentials, click REGISTER. If the passwords don't match, the warning pop-up will warn you.

BACK TO LOGIN PAGE

CREATE NEW CUSTOMER ACCOUNT

Username:

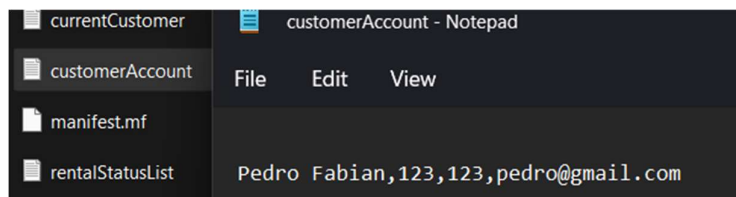
Password:

Confirm Password:

Contact Number:

Email:

The new customer account has been created and appended into “customerAccount.txt”.



It will bring us back to the main Login Page.

Now, we will login as Customer Account. Pedro Fabian, the user, will enter his credentials into the 2 text boxes. After that, he clicks Login.

LAM & PEDRO'S CAR RENTAL SERVICE SYSTEM

Username:

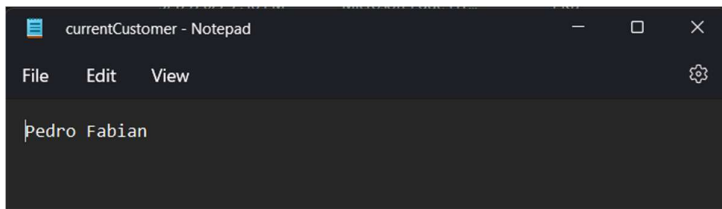
Password:

[Click here to create new Customer Account](#)

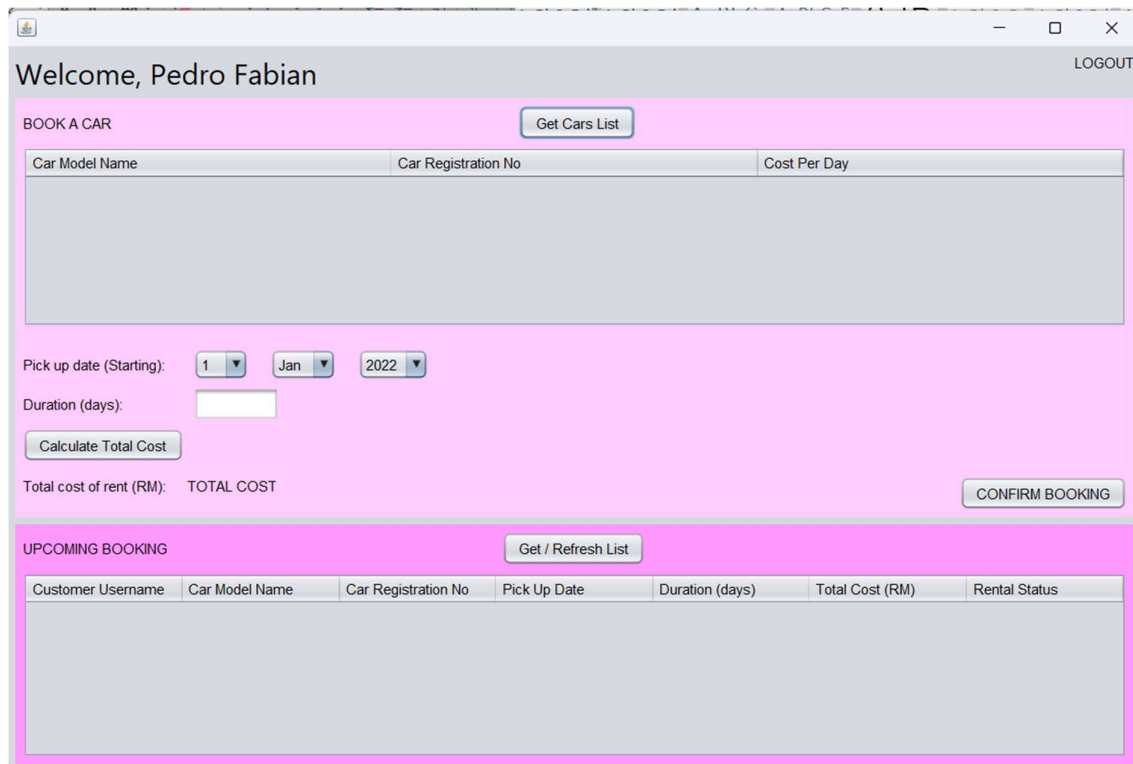
[Click here to login Admin Account](#)

[Click here to create new Admin Account](#)

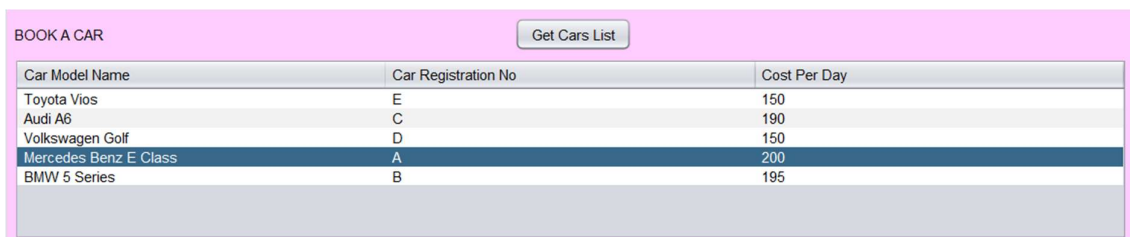
This will bring us to the Customer’s JFrame. At the same time, Pedro Fabian’s name will be saved into the “currentCustomer.txt”



Then, it will bring us to the Customer's JFrame.



“Welcome, Pedro Fabian.” The label next to “Welcome” reads the “currentCustomer.txt” for Pedro Fabian’s name and then set the label to this text. Pedro will then choose a car to book. He will click Get Cars List.



The “carlist.txt” is read and placed into this table. Pedro will click one of the cars, then choose the date, duration, then click Calculate Total Cost.

Pick up date (Starting):

Duration (days):

Total cost of rent (RM): 1000

After that, Pedro clicks CONFIRM BOOKING.

BOOK A CAR

Car Model Name	Car Registration No	Cost Per Day
Audi A6	C	190
Volkswagen Golf	D	150
BMW 5 Series	B	195
Toyota Vios	E	150

The car selected from the table is deleted and Pedro's name, the car, the car registration number, and booking details are appended into "rentalStatusList.txt"

```

rentalStatusList - Notepad
File Edit View

Pedro Fabian,Mercedes Benz E Class,A,1/Jan/2022,5,1000,Booked

```

Pedro can then click GET/REFRESH LIST to see all of his own upcoming bookings. This is done by searching his name through the "rentalStatusList.txt" and taking that line into the table.

UPCOMING BOOKING

Customer Username	Car Model Name	Car Registration No	Pick Up Date	Duration (days)	Total Cost (RM)	Rental Status
Pedro Fabian	Mercedes Benz E Cla...	A	1/Jan/2022	5	1000	Booked

Pedro can now log out as customer. Clicking the LOGOUT button on the upper right hand corner will bring us to the main login page. Now, we are going to login as admin.

GO BACK TO LOGIN PAGE

LOGIN ADMIN ACCOUNT

Username:

Password:

Just now, Lam Weng Yew had registered as admin. He is going to login now.

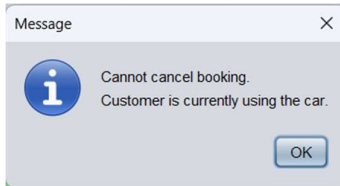
This brings us to the Admin Page 1, Booking Management. Here, the admin can click “Get Rental Status List” to see all the upcoming bookings by all customers. It is done by reading all the lines in “rentalStatusList.txt”.

Since we have only one customer booking, who is Pedro Fabian, only his booking will appear here. After clicking on the row that the admin wants to take action on, now, there are 2 things the admin can do. He can either cancel that booking, or click customer has picked up vehicle. If admin cancels the booking, that line will be deleted and some details will be appended into the “report.txt”, then in the “carlist.txt”, that car and its details will be appended in again, making it available for customers to choose.

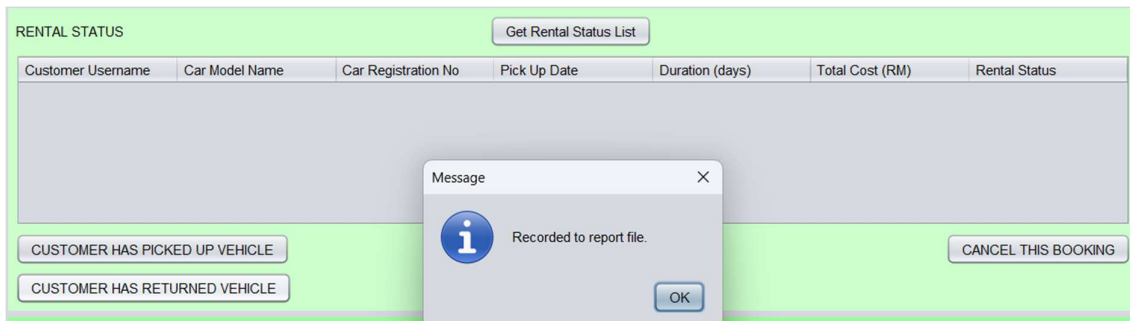
In this scenario, the admin will click CUSTOMER HAS PICKED UP VEHICLE.

Total Cost (RM)	Rental Status
1000	In Use

The Rental Status will change from “Booked” to “In Use”. Then, the admin can only click CUSTOMER HAS RETURNED VEHICLE, and not CANCEL BOOKING. A pop-up will show if the admin clicks CANCEL BOOKING.



Now, the admin will click CUSTOMER HAS RETURNED VEHICLE.



The line in Rental Status table and “rentalStatusList.txt” has been deleted. At the same time, some details of the booking will be appended into “report.txt” and that car will be added into “carlist.txt” to make it available to customers again.

Now, let’s go to Admin Page 3, Analysed Report.

GENERATE ANALYSED REPORT						
Customer Username	Car Model Name	Car Registration No	Pick Up Date	Duration (days)	Booking Status	Cost (RM)
Pedro Fabian	Mercedes Benz E Cl...	A	1/Jan/2022	5	Successful	1000

Now, below the table, the admin will click **GENERATE ANALYSIS**.

GENERATE ANALYSIS

Total number of bookings: **1**

Total revenue (RM): **1000**

This will calculate the total number of bookings, whether successful or cancelled. The total revenue is also calculated by adding up all the numbers on the last column of the table.

Back to Admin Page 1. On the bottom half, the admin can search customer's booking history by just searching their Username. When the admin enters "Pedro Fabian" and clicks SEARCH CUSTOMER USERNAME, all of his booking that has been successful or cancelled will appear. This is done by searching for Pedro Fabian name in the "report.txt" and taking those line which contains that name into the table.

CUSTOMER'S BOOKING HISTORY						
<div> <div>SEARCH CUSTOMER USERNAME</div> <div>Pedro Fabian</div> </div>						
Customer Username	Car Model Name	Car Registration No	Pick Up Date	Duration (Days)	Booking Status	Total Cost
Pedro Fabian	Mercedes Benz E Class	A	1/Jan/2022	5	Successful	1000

Now, the booking demonstration is done. We only have Admin Page 2, Cars and Customer Management to talk about.

ADMIN PAGE 2 - CARS & CUSTOMER MANAGEMENT

LOGOUT

BACK TO PAGE 1

TO PAGE 3

CARS LIST

Import Cars List

Car Model Name	Car Registration No	Cost Per Day

Car Model Name:

Car Registration Number:

Cost Per Day:

Clear

ADD NEW CAR

UPDATE / MODIFY CAR

DELETE CAR

SAVE

CUSTOMERS ACCOUNTS LIST

Import Customer Accounts

Customer Username	Password	Contact Number	Email

Contact Number:

Email:

DELETE CUSTOMER ACCOUNT

MODIFY DETAILS

SAVE

Here, customer can edit details of the cars and edit details of customer accounts. Let's start with cars. Admin will click Import Cars List. This will get every line from the "carlist.txt".

CARS LIST Import Cars List		
Car Model Name	Car Registration No	Cost Per Day
Audi A6	C	190
Volkswagen Golf	D	150
BMW 5 Series	B	195
Toyota Vios	E	150
Mercedes Benz E Class	A	200

As you can see, the car which Pedro Fabian had booked, taken and then returned, Mercedes Benz E Class with registration number of A, had appeared again in the "carlist.txt".

Now, let's add a new car.

Car Model Name:	<input type="text" value="Proton Saga"/>	<input type="button" value="ADD NEW CAR"/>
Car Registration Number:	<input type="text" value="F"/>	<input type="button" value="UPDATE / MODIFY CAR"/>
Cost Per Day:	<input type="text" value="50"/>	<input type="button" value="DELETE CAR"/>
<input type="button" value="Clear"/>		<input type="button" value="SAVE"/>

After entering the details of the new car, click ADD NEW CAR.

CARS LIST Import Cars List		
Car Model Name	Car Registration No	Cost Per Day
Audi A6	C	190
Volkswagen Golf	D	150
BMW 5 Series	B	195
Toyota Vios	E	150
Mercedes Benz E Class	A	200
Proton Saga	F	50

The new car has been added to the table.

Now the admin wants to delete a car. Let's delete Audi A6, with registration number C. Click on that line and click DELETE CAR.

CARS LIST Import Cars List		
Car Model Name	Car Registration No	Cost Per Day
Volkswagen Golf	D	150
BMW 5 Series	B	195
Toyota Vios	E	150
Mercedes Benz E Class	A	200
Proton Saga	F	50

Now, that car has been deleted.

The admin now wants to modify the details of one of the cars. Let's take Toyota Vios and change its Cost Per Day to 140. Just simply click on the row, and each of the details are set into the text boxes. The admin can change details just from there.

Car Model Name	Car Registration No	Cost Per Day
Toyota Vios	E	150
Mercedes Benz E Class	A	200
Proton Saga	F	50

Car Model Name:

Car Registration Number:

Cost Per Day:

ADD NEW CAR

UPDATE / MODIFY CAR

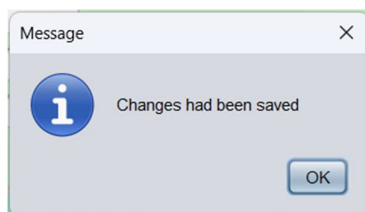
DELETE CAR

After that, admin will click UPDATE / MODIFY CAR.

CARS LIST			Import Cars List
Car Model Name	Car Registration No	Cost Per Day	
Volkswagen Golf	D	150	
BMW 5 Series	B	195	
Toyota Vios	E	140	
Mercedes Benz E Class	A	200	
Proton Saga	F	50	

Now, Toyota Vios's Cost Per Day has been changed to 140 per day.

We are not done yet. All of this must be saved manually into the "carlist.txt", by just clicking SAVE.



Now, in the "carlist.txt", you can see that the file has been overwritten with new data from the table.

```

carlist - Notepad
File Edit View

Volkswagen Golf,D,150
BMW 5 Series,B,195
Toyota Vios,E,140
Mercedes Benz E Class,A,200
Proton Saga,F,50
  
```

Next, the admin can modify the customer's account's details. When admin clicks Import Customer Accounts, it reads from "customerAccount.txt" and displays all the lines into the table.

CUSTOMERS ACCOUNTS LIST			
Import Customer Accounts			
Customer Username	Password	Contact Number	Email
Pedro Fabian	123	123	pedro@gmail.com

After clicking on the row, the admin can see modify the contact number and/or email. The admin wants to change contact number to 012345. Then, click MODIFY DETAILS.

Customer Username	Password	Contact Number	Email
Pedro Fabian	123	123	pedro@gmail.com

Contact Number:

Email:

Now the new contact number of Pedro Fabian has been updated.

CUSTOMERS ACCOUNTS LIST			
Import Customer Accounts			
Customer Username	Password	Contact Number	Email
Pedro Fabian	123	012345	pedro@gmail.com

Just like the cars list, the admin must save to "customerAccount.txt" manually by clicking SAVE.

```

customerAccount - Notepad
File Edit View

Pedro Fabian,123,012345,pedro@gmail.com
  
```

Now in "customerAccount.txt", the contact number has been updated.

The admin can now logout.

5. CONCLUSION

This last section shows that the objectives initially defined in the application have been satisfactorily achieved. This has been achieved after a detailed planning of the tasks to be carried out throughout the project based on the initial objectives, the technologies to be used and the strategies to be followed during the design phase. This has avoided organisational and time problems in project management.

After completing the project on the development of the "car rental" application, we can highlight a series of conclusions obtained during its development. - Development of application programming skills. During the development and testing of the program, we have reached a high level of knowledge in Java development. In addition, it has allowed us to understand the pillars of the behaviour of an application.

6. **REFERENCES**

- https://profile.w3schools.com/refresh-session?redirect_url=https%3A%2F%2Fwww.w3schools.com%2Fjava%2Fdefault.asp
W3Schools Java
- <https://www.geeksforgeeks.org/different-ways-reading-text-file-java/>
Different ways of reading text file Java, 20 December 2021
- <https://youtu.be/A74TOX803D0>
Java Tutorial, freeCodeCamp.org, YouTube