Vehicle Rental and Purchasing System

Mohamed Qaail Usaim

UTS ID:14691950

**Programming Project**

***Subject Code:IPRO001***

# Table of Contents

[1. The Scenario 1](#_Toc195796013)

[2. Detailed Code Explanation 2](#_Toc195796014)

[**2.1** Planning of the code 2](#_Toc195796015)

[**2.2** Classes 3](#_Toc195796016)

[2.2.1. Vehicles Class (Parent Class) 👾 3](#_Toc195796017)

[2.2.2. Classic Car Class (Child Class 1) 👾 4](#_Toc195796018)

[2.2.3. Super Car Class (Child Class 2) 👾 5](#_Toc195796019)

[2.2.4. SUV Class (Child Class 3) 👾 6](#_Toc195796020)

[2.2.5. Electronic Vehicle Class(Child Class 4) 👾 7](#_Toc195796021)

[2.2.6. Customer Class (Other Class 1) 👾 8](#_Toc195796022)

[2.2.7. Bookings Class (Other Class 2) 👾 9](#_Toc195796023)

[**2.3** Main Class for Code Execution 12](#_Toc195796024)

[2.3.1. initalizeData() Method 👾 13](#_Toc195796025)

[2.3.2. start() Method 👾 14](#_Toc195796026)

[2.3.3. displayMenu() Method 👾 15](#_Toc195796027)

[2.3.4. login() Method 👾 16](#_Toc195796028)

[2.3.5. register() Method 👾 16](#_Toc195796029)

[2.3.6. rentVehicle() Method 👾 17](#_Toc195796030)

[2.3.7. buyVehicle() Method 👾 18](#_Toc195796031)

[2.3.8. viewallBookings() Method 👾 20](#_Toc195796032)

[2.3.9. logout() Method 👾 21](#_Toc195796033)

[2.3.10. getintegerInput() Method 👾 22](#_Toc195796034)

[3. Project Codebase 23](#_Toc195796035)

[4. Problems Encountered 36](#_Toc195796036)

[**4.1** Invalid Input 36](#_Toc195796037)

[**4.2** Nullpointer Exception Case 36](#_Toc195796038)

[5. Runtime Outputs (Terminal) 37](#_Toc195796039)

# The Scenario

The AutoNova project is a comprehensive vehicle rental and sales program which is designed to facilitate smooth transactions for both renting and buying vehicles. The system portrays a simplified version of a real-world car dealership offering services such as browsing available vehicles, renting out vehicles for certain durations and even permanently purchasing cars. In addition to this, it also securely stores various customer information through a simple login process requiring a username and password unique to each user.

The main objective of this project is to enhance the user-friendly nature, which provides customers with reliable and accurate information on a wide range of vehicles categorized accordingly. The platform allows users to rent a vehicle for a short period of time or purchase a vehicle entirely with the added benefit of warranty and insurance coverage, which usually consumes a lot of time once a vehicle has been purchased.

All vehicle, customer and transaction data are stored in a well structured and organized way, ensuring reliability and trustworthy presence when browsing. The system’s modular design supports future enhancements such as integrating payment gateways through trusted merchants like VISA and Mastercard, real-time vehicle availability based on condition of vehicle and status, dynamic pricing to support bidding and offers customers can offer, and more advanced filters like fuel type, brand, and performance specs.

This program is built around several components, including classes for vehicles, customers, bookings and transactions while utilizing efficient data structures like ArrayLists and HashMaps for data storage and data retrieval when required. This code layout also follows object-oriented principles such as making extensive use of the parent and child class relationship to reuse code which promotes modularity. The usage of predefined methods in other external classes is another factor which enhances data encapsulation and security of the components involved in the program. These methods also structure the behaviour of the program by handling key operations such as booking a vehicle for rent, calculating the rental cost, checking availability and managing purchases of vehicles. This hierarchical structure allows for easier maintenance, scalability and future enhancements.

# Detailed Code Explanation

## Planning of the code

Before diving into the explanation of the code, it’s crucial to illustrate how the program has been constructed. This can be effectively done using a flow chart to demonstrate the seamless integration of different concepts.

A diagram of a company

AI-generated content may be incorrect.

## Classes

### Vehicles Class (Parent Class) 👾

A computer screen shot of a program

AI-generated content may be incorrect.

This class will act as the parent class for all types of vehicles that come under this category because it contains most of the attributes the child classes (types of vehicles) will have. The purpose of this is to define each vehicle uniquely while being able to reuse code easily. The private attributes such as ‘id’ is important in acting as a unique identifier for the respective vehicle stored in the database of the system.

The constructor of the parent class’s main purpose is to initialize all the attributes of the class. This will allow child classes, when extended, to inherit these attributes using the super() command with ease.



The above code in the image are the respective getters and setters for the ‘Vehicles’ class. The getters in the Vehicles class are used to retrieve the values of the private attributes. Since the attributes are protected by the ‘private’ access modifier command, they can’t be accessed directly outside the class. This is where getters come into play in providing controlled access to these attributes. Meanwhile the single setter in this class is used to modify the value of an attribute. In this instance it will be the availability factor of the vehicle because it can change at any time when customers make purchases.

Finally moving to the toString()method of the class. The purpose of this is to provide a string format representation of all the information in the class in a customized format. The override command overrides the object class and displays the details of the vehicle in a readable format which can be designed by the developer.

### Classic Car Class (Child Class 1) 👾

A computer screen with text

AI-generated content may be incorrect.

This is one of the child classes defined in this program, which is inheriting the attributes of the Vehicles class. It represents all the regular cars, which include the hybrid version of them as well. The Boolean attribute ‘isHybrid’ checks whether the specified vehicle has a hybrid version or not.

The constructor initializes the attributes of the ClassicCar class, including those inherited from the Vehicles class. The super keyword has been used to call the constructor of the parent class and initialize its attributes as well. “Classic Car” is given as the type of vehicle, and 300$ is passed as the price of insurance for all vehicles under this class, making it predefined in this class. The ‘this.isHybrid’ statement assigns the value of the ‘isHybrid’ parameter to the isHybrid attribute of the ClassicCar object.

A screen shot of a computer program

AI-generated content may be incorrect.

The getter for the Boolean attribute ‘isHybrid’ is added due to the access modifier in the attribute definition being private. Its role is to simply return if the vehicle is in its hybrid version or not.

Moving on to the toString() method at the end, similar to the Vehicles class, its job is to portray a string representation of the class details. An if statement is used to check whether the car is in hybrid version or not. Utilizing the parent class through the super() keyword, I was able to reduce the lines of code and add data to the toString() method which is unique to the class specifically. This represents the data of the car which belongs to this class when displayed to the user when the program is being run.

### Super Car Class (Child Class 2) 👾

A computer screen with text on it

AI-generated content may be incorrect.

The super car class acts the same as the classic car class. The only difference this has is the special attribute in the which is the top speed of the car. Since most consumers inquire about it when purchasing such high-end vehicles. The attribute topSpeed is used to display the speed in kilometres using the private access modifier.

The constructor sees the only change in the type being Super Car and the insurance price being $1000 due to the price of parts being more expensive, but in addition to that, initialized the attribute of top speed to display it to the consumer.

The getter method for the top speed of the car is added due to there being a private access modifier, which enables us to use this method outside the class.

The toString() method utilizes the parent class contents, but in addition, the top speed factor was included, which is unique to this class due to the specific type of car the consumers are dealing with.

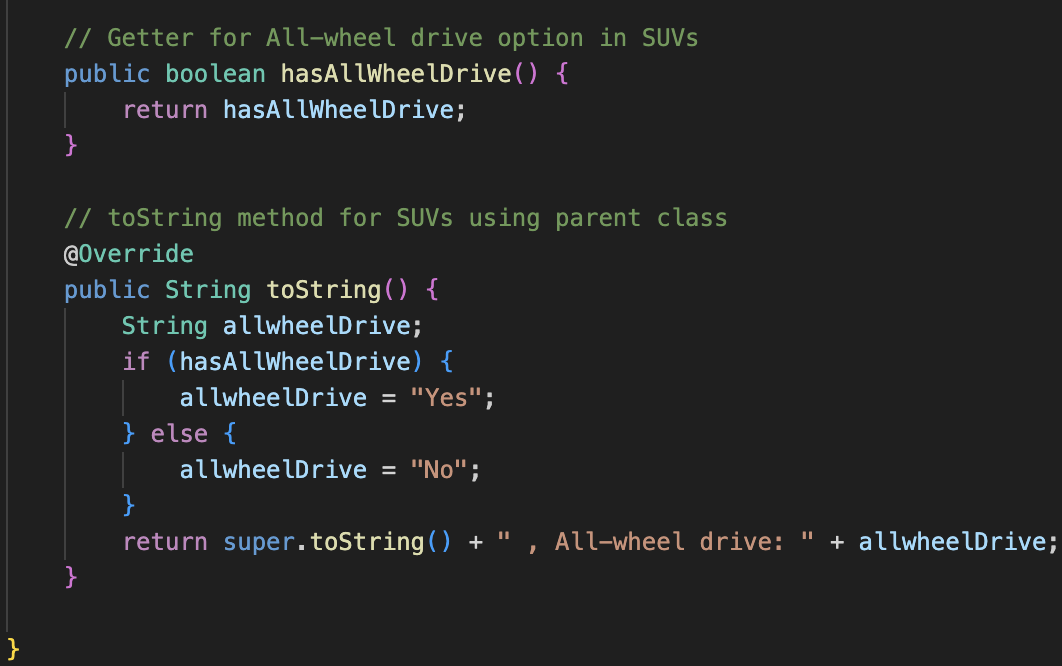
### SUV Class (Child Class 3) 👾

A screen shot of a computer

AI-generated content may be incorrect.

The SUV class is also another child class of the Vehicles class due to this also being a different type of vehicle meant for rough terrain roads. The special attribute this class has is the all-wheel drive option which does not come with all types of SUV’s. To solve this a Boolean inclusion of whether the vehicle has an all-wheel drive option or not has been included with a private access modifier infront of it.

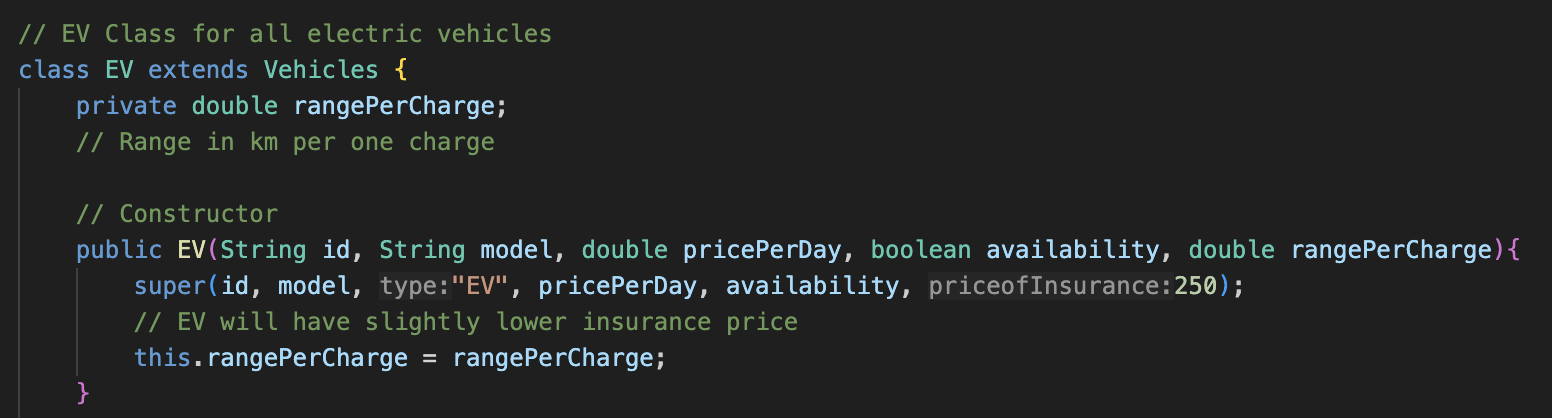
The constructor is also similar to the previous two classes which had the type, and the price of insurance already specified using the super() command to call the attributes form the parent class to initialize them.



The getter used here is due to the access modifier used for this Boolean expression so that it can be used outside the specific class.

The toString() method follows in this case to verify if the SUV has all-wheel drive or not. This is done using an ‘if’ statement which states “Yes” or “No” if the vehicle contains it or not. Then it returns the toString method utilizing the parent class contents and additionally including the option if the SUV contains all terrain option with a simple “yes” or “no”.

### Electronic Vehicle Class(Child Class 4) 👾



The EV Class is also a similar representation of the previous child classes. The unique attribute in this class is the range per charge in kilometres which is unique to an electric vehicle. For this I have added an attribute called ‘rangePerCharge’ as a double datatype to display this to the user.

The constructor in this instance is also similar to the previous classes but has that special attribute which is the range per charge of the vehicle. Initializing this in the constructor is important as users will want to know these specific types of information.

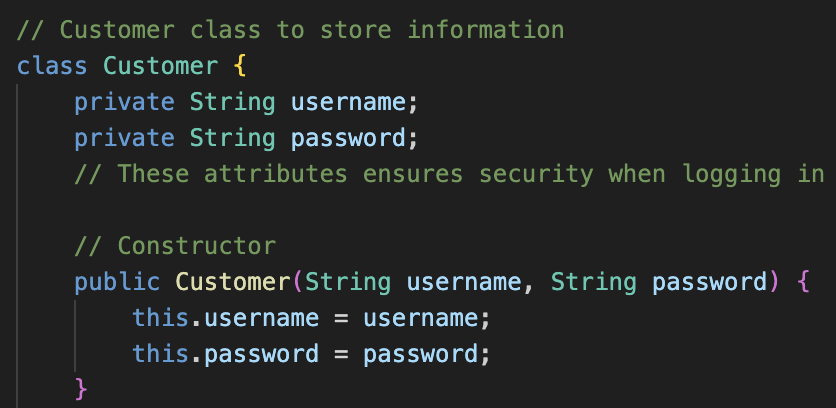
A computer screen with text

AI-generated content may be incorrect.

The getter in this code allows us to call the attribute of the range per charge of each electric vehicle when required due to it being encapsulated with a private access modifier.

The last child class’s toString() method is also pretty similar with the only change being the special attribute of data being displayed which is the range per charge in kilometers. This therefore completes the range of child classes with the assistance of the parent class and ensures reusable code and modular coding format.

### Customer Class (Other Class 1) 👾



The purpose of this class is to store the data of each customer when they enter the program and register themselves or log in. For this, I have assigned two attributes, which are username and password, to deal with this. This ensures security when logging in as the username will be unique to each consumer.

The constructor initializes these attributes simply with the username followed by the password in string format. The “this.” command is used when we want to initialize the attributes in the current class we are in.

A screen shot of a computer code

AI-generated content may be incorrect.

The getters utilized in this code are only to call on the username and password outside the class due to them being restricted with a private access modifier to ensure data encapsulation. The getters have been named with getPassword() and getUsername(), which follows Java naming conventions followed by a string datatype common to both of them. A toString() method is not required here as the need to display a string representation of this data will be of no use in the context of the code.

### Bookings Class (Other Class 2) 👾

A screen shot of a computer program

AI-generated content may be incorrect.

The purpose of this class is to store all the information of the vehicles listed. To do this, attributes like userID, vehicleModel, days, and totalPrice have been added. Since they are common data required to make a booking of a specific vehicle. An ArrayList has been added to control the list of bookings made by a single user.

A computer screen shot of text

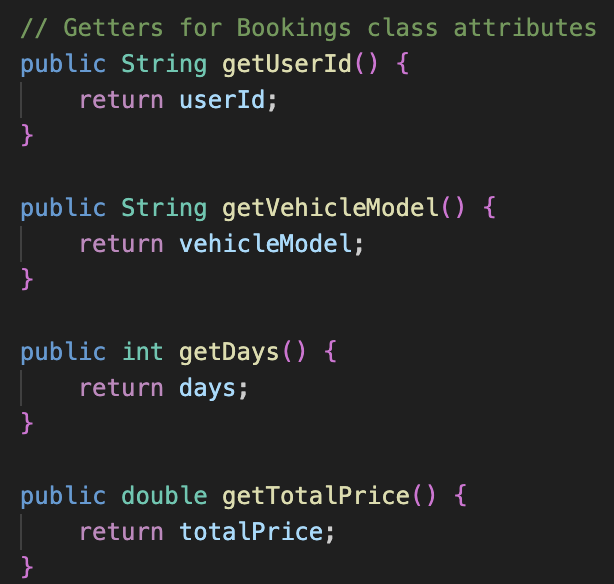
AI-generated content may be incorrect.

This class has a main constructor which controls all the bookings and initializes all the important information using the “this.” command, needing the just the data in those areas to be filled. The second constructor mentioned is to initialize the booking list. This may not exactly resemble a constructor from which most of us are familiar but it is important to initialize it separately so it can be used in our main code when running in certain extra areas when required other than including it inside our main constructor of this class. I have transformed this into an ArrayList, enabling us to dynamically add and remove elements as per our requirements.

A black background with text

AI-generated content may be incorrect.

A single method has been added that doesn’t return any value so that bookings can be added to the list each time. To do this the ‘void’ command has been used due it signalling that this method will not return any value and simply adds a single booking when called upon.



The getters in the Bookings class are simply to call on the attributes when required. This is because they have been encapsulated by the private access modifier, which is a default convention to do in Java code. getUserId(), getVehicleModel(), getDays(), and getTotalPrice() are all getters which can be used outside the class, utilizing them when required because they will return a value as assigned to their relative datatype.

A screen shot of a computer code

AI-generated content may be incorrect.

This method utilized the getters I created for the attributes made. This method main aim is to display the bookings for a single user. To do this I added an argument in the method that the userID must be a string datatype. Then a for-each loop is used to iterate through the ArrayList and then inside the for-each loop an if statement is given provided the userId of the current ‘booking’ object is the same as the userId passed to the displayBookings() method.

As a result only bookings made specifically by the user will be displayed.

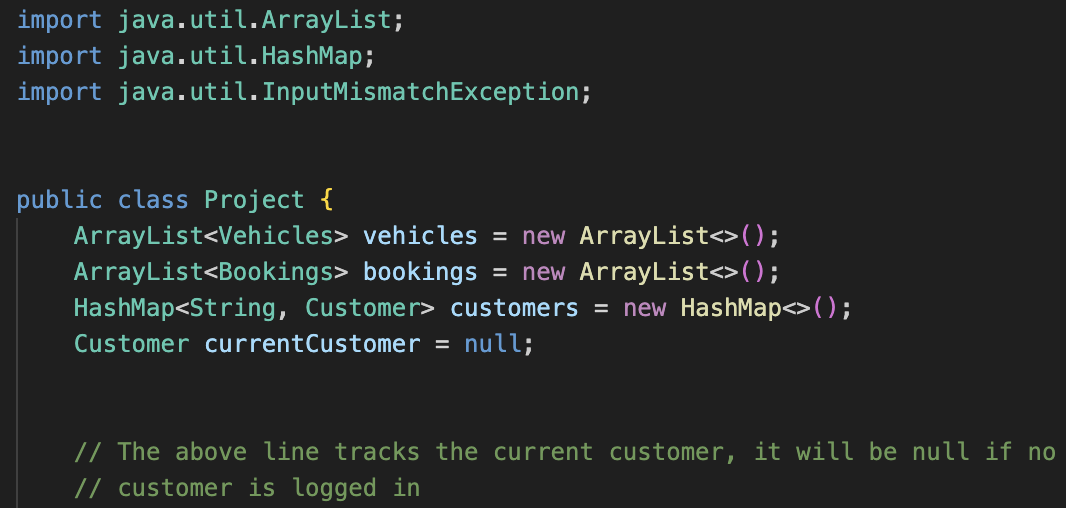
A black background with blue and white text

AI-generated content may be incorrect.

Towards the end the toString() method of this class is created. This allows the code data within the class to be displayed as a custom string representation by the user. When this class is called upon the toString() method will display all the attributes which are specified after the return keyword. This concludes the usage of classes in this program and provides clear modular code so that any developer can understand and modify such codes in the future.

***Continued on Next Page***

## Main Class for Code Execution



Now that the classes for the respective program are complete the focus can be turned on the area where the code will be executed. Firstly, the ArrayLists and HashMaps need to be imported which should be done initially before writing the code. The InputMismatchException exception case will be utilized and explained in a method where it is required.

Moving on to the ArrayList I created which is using the Vehicles class naming the variable as vehicles and an ArrayList for the Bookings class which I called the variable name ‘bookings’. Moving on to the reason I imported HashMap. This is mainly due to the fact that so that each user can securely enter their data if the respected key matches the values entered with the aid of the string datatype and the customer class which I created to handle the username and password. Note that a specific variable called ‘currentCustomer’ has been declared to be null which indicates that no one has logged in yet. This helps the program know when to show the login/register menu and when to show the main menu for logged-in users.

A screen shot of a computer code

AI-generated content may be incorrect.

This is where the program will run. To facilitate this an object from the main class called ‘project’ is created. The methods initializeData() and start() are custom methods that are defined in the below codes and will be explained in detail.

### initalizeData() Method 👾

This method is responsible for adding all the data of the vehicles requires so that consumers can see what cars are available to buy or rent. A screen shot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

The two lines using ‘customer.put()’ add two sample users to the customers HashMap. The key is the username, and the value is a new Customer object with a username and password. This allows the program to have users who can directly log in when it starts. The ‘void’ keyword has been added Infront of the method to indicate that the method does not return and value.

Next, sample vehicles are added to the ‘vehicles’ ArrayList defined earlier. To do this firstly objects are created of all the types of cars as shown in the picture of the code. Then followed by that using the “vehicles.add()” method objects are added to the ArrayList ensuring all the vehicles are added to the defined database of our system. This henceforth concludes the initializeData() method which ensures all the predefined data required in the program is stored.

### start() Method 👾

A screen shot of a computer program

AI-generated content may be incorrect.

The purpose of the start method is the main loop of the program. It will keep the program running , display the menu and handles user choices based on whether a customer is logged in or not similarly the ‘void’ keyword is used to indicate it does not return any value.

The ‘run’ variable defined in a Boolean datatype decides whether the program should keep running. Then the ‘while(run)’ loop ensures the menu will keep displaying until the user chooses to exit the program.

The displayMenu()(explained in detail below) method then prints the appropriate menu whether to login or register. Next, the getintegerInput() asks the user to select an option and reads their input.

Moving to the if statement area, if the currentCustomer variable does equal to null then within that another if statement begins. This gives us the options to login, register or exit the program which sends a goodbye message and any invalid input such shows as Invalid option.

If that condition is not satisfied the program moves to the next ‘if’ statement which is after the else clause. Inside that else clause another if statement has been placed. This is when the currentCustomer variable is not null and the user is logged in. It them prompts the user to rent a vehicle, buy a vehicle, view their bookings and purchases or logout(sets currentCustomer to null again) and as usual if any invalid input the Invalid option message displays again.

### displayMenu() Method 👾

A screen shot of a computer program

AI-generated content may be incorrect.

The displayMenu() method is responsible for showing the correct menu options to the user which depend if they are logged in or not. The ‘private’ access modifier keyword is used so that this helper method cannot be access outside the main class.

Initially a header is printed using the print line command. Then an ‘if’ statement is used to check if the user is logged in or not by checking if currentCustomer variable is equal to null. Then it prints the respective menu depending on the login status of the customer. The aim of this method is to provide a user-friendly interface when interacting with the program.

***Continued on Next Page***

### login() Method 👾

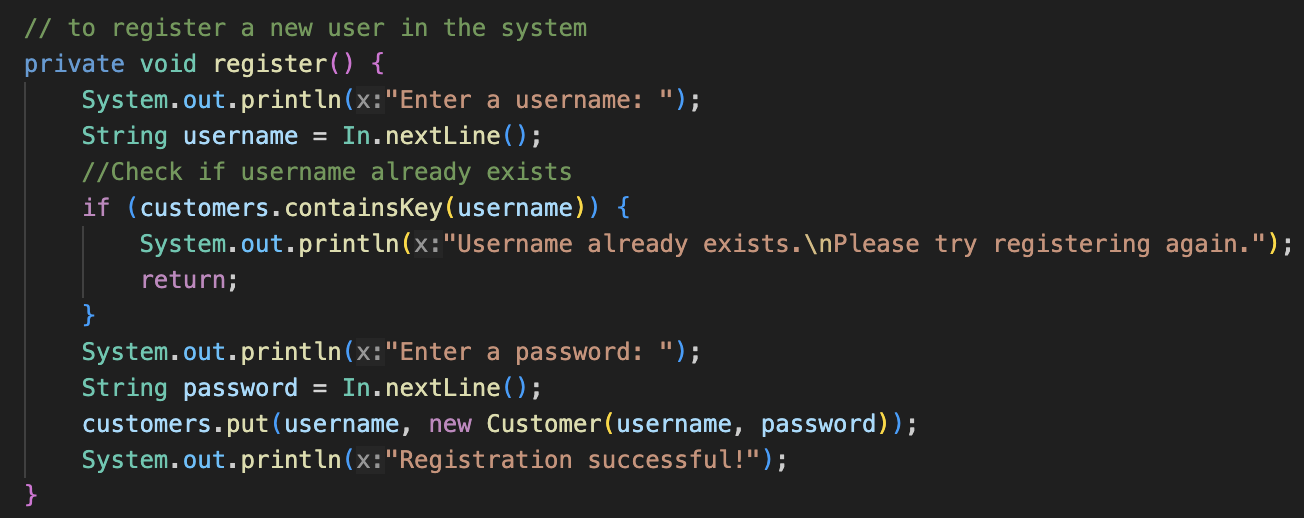
A computer screen shot of a program code

AI-generated content may be incorrect.

This method allows the customer to login to the system by checking their username and password. The method asks the user to enter their username and password, which can be entered using the In.java class provided by UTS college.

Then the program attempts to find a Customer object in the ‘customers’ HashMap created using the entered username. If a customer is found and the password matches, the login will turn out to be successful, and a login success message will be displayed. If the username does not exist or the password is incorrect, it will display “Invalid Credentials”. This is followed by error handling, which also catches cases like input mismatch, and it prints an error message.

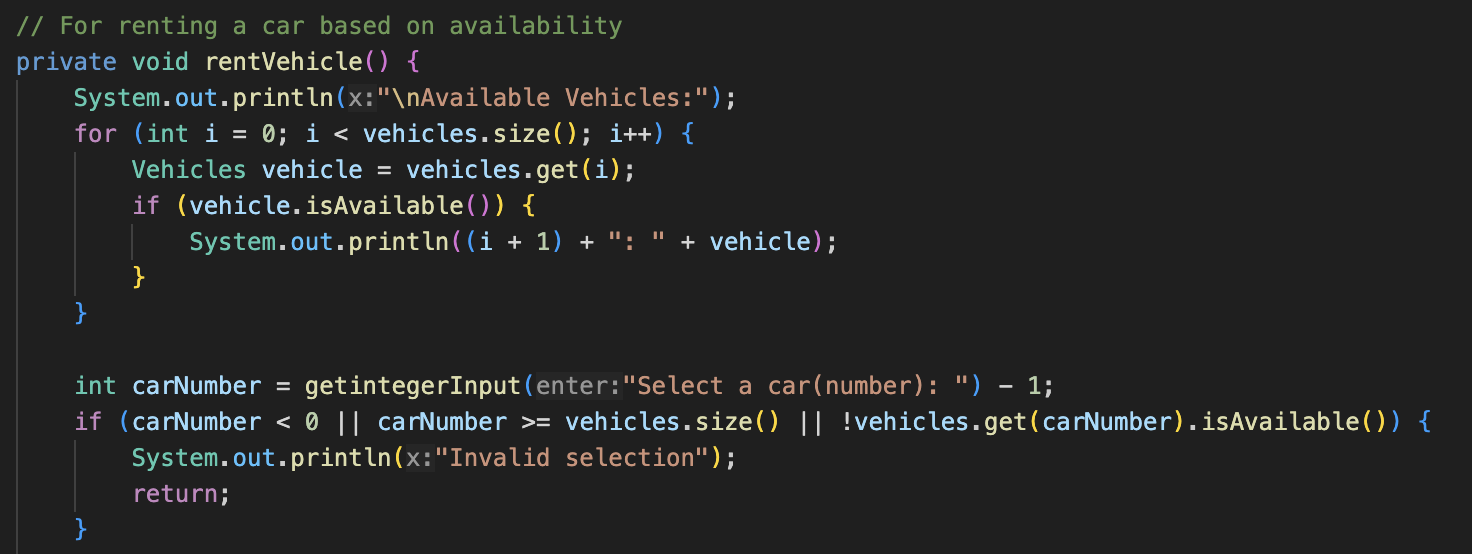
### register() Method 👾



This method allows a new user to create an account in the system. This method asks the user to enter a username, and after that, it checks if it is already in the ‘customers’ HashMap and displays the message saying that it already exists.

Then it prompts for a password only if the username is unique. After that, it proceeds to add this data by creating a new Customer object and adds the new user to the ‘customers’ HashMap using the “customers.put()” command. Then it prints a confirmation message saying that the registration was successful.

### rentVehicle() Method 👾



This method allows the user to rent an available vehicle from the list of vehicles created. A for loop is used to go through each vehicle in the ‘vehicles’ list. For each vehicle it checks if the vehicle is available through the “.isAvailable()” getter. If the vehicle is available it prints the vehicle’s number and details.

Using the getintegerInput() method the program prompts a number from the user in the ‘carNumber’ variable. Once the number is entered it subtracts 1 to convert from user-friendly numbering (starting at 1) to zero-based indexing used in java lists.

The if statement checks if the selected number is less than 0 (invalid input), if it is greater than or equal to the number of vehicles (out of range), or if the selected vehicle is not available. If any of these conditions are met, the program prints “Invalid selection”.

A screen shot of a computer code

AI-generated content may be incorrect.

Then the variable ‘selectedVehicle’ is created from the Vehicles class. This retrieves the vehicle object the user selected from the list.

The user is then asked for the rental duration period to enter the number of days they want to rent the vehicle. This allows the code to proceed to calculating the total price and print it using a formula.

Next, it inquires the user for a booking confirmation. The if statement is used to confirm the booking. If the user enters “Y” a new booking will be added to the ‘bookings’ ArrayList and sets the vehicle as unavailable which means it cannot be rented again. Then it proceeds to display a confirmation message of the booking. However, if the user enters anything else the booking will be cancelled.

### buyVehicle() Method 👾

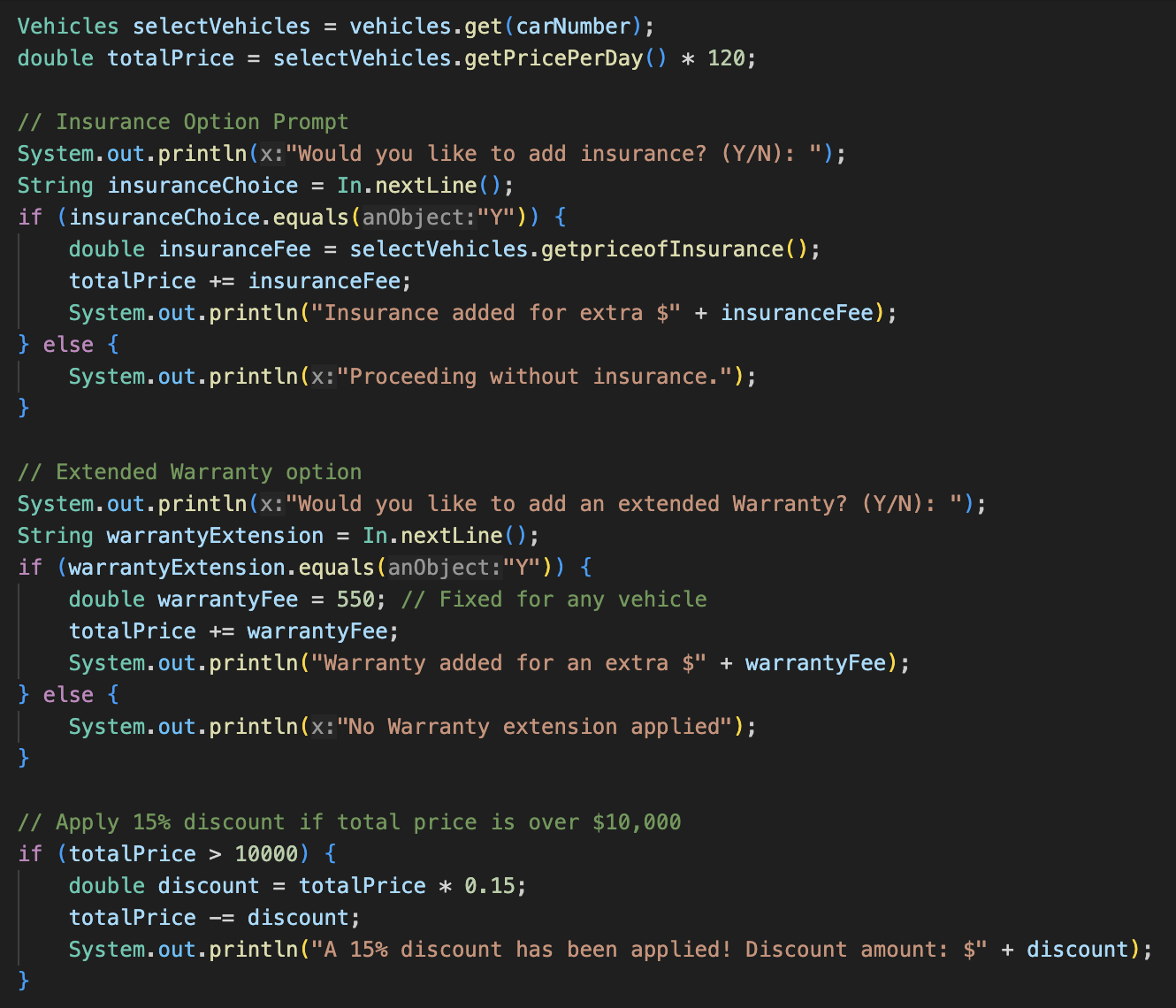
A screen shot of a computer program

AI-generated content may be incorrect.

This method allows the user to buy a vehicle from the program while also being able to obtain a discount depending on the price of the vehicle they choose to buy.

Firstly, the program displays the available vehicles for purchase. This is done by using a for loop which iterates through the size of the ArrayList created with the name ‘vehicles’. For each vehicle, if it is available, it prints the vehicle number, model, type, purchase price, and a message describing the vehicle.

Then it prompts the user below to select a vehicle based on the ‘id’ number of it in the system. Once the number is entered it uses the if statement to check through the possible errors it could have through the ‘||’(OR) operator. If there is an error, it will display “Invalid selection”.



Then the selected vehicle is retrieved using the “vehicles.get(carNumber)” function followed by the total price of the vehicle.

The user is prompted with an insurance option using the In.java class to reply to that. An ‘if’ statement is added to check if the user inputs “Y” then the insurance gets added to the total price given the respective price of the vehicle depending on the type of vehicle called. Otherwise, if the user enters anything else it proceeds without insurance.

Moving to the extended warranty option which again prompts the user for an input and follows the same convention as the insurance.

A discount is also applied through an if statement using the totalPrice variable if it is more than 10000$. Then a variable name discount is created and displays the discount value to the user.

A screen shot of a computer code

AI-generated content may be incorrect.

A preset date is defined in the date library in Java, which allows the user to check the date of purchase. The code follows up with the addition of all the extra purchases and displays it to the user, followed by a confirmation message to display the details of the vehicle purchased. The confirmed vehicle will be marked as unavailable, and a new booking will be added to the ArrayList (with days set to 0 to indicate a purchase and not a rental), followed by a confirmation message about the purchase along with the date. If the user does not confirm, a cancellation message is shown.

### viewallBookings() Method 👾

A screen shot of a computer program

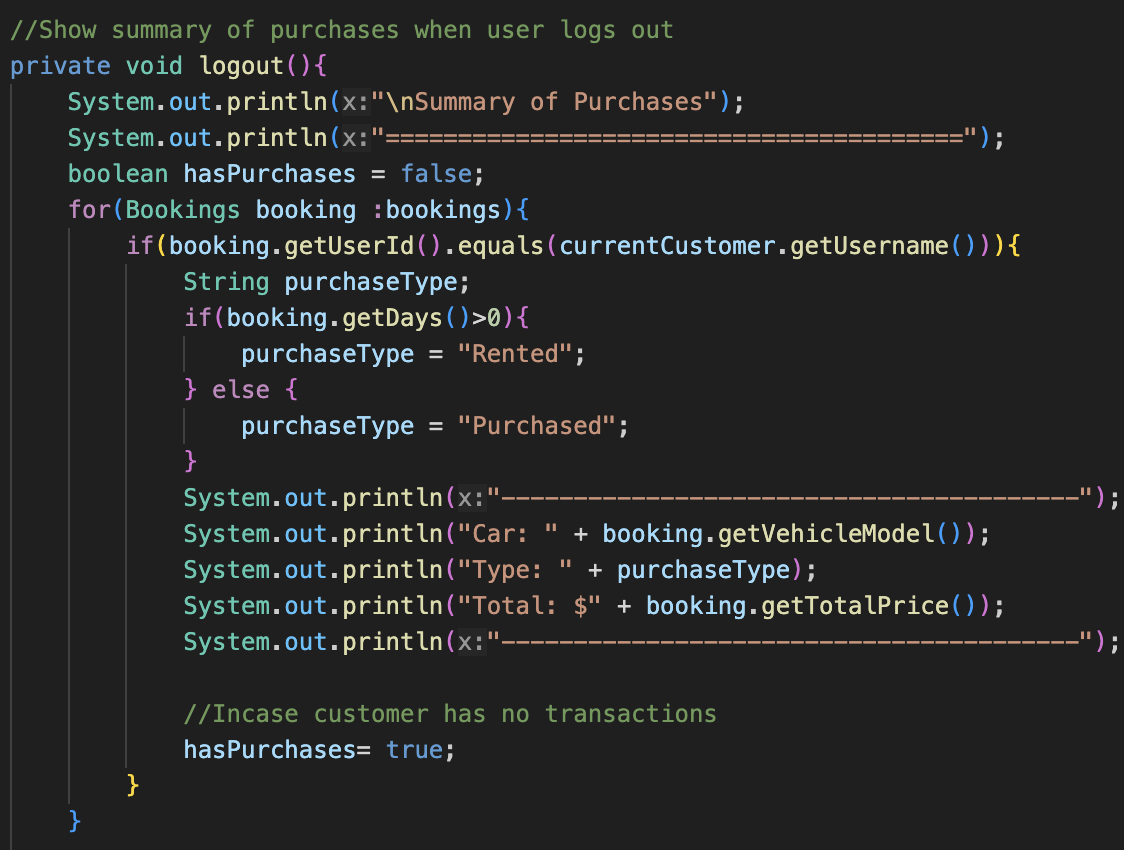
AI-generated content may be incorrect.

This method displays all the bookings and purchases made by a currently logged in user. Initially the users name is printed using the getUsername() method.

This is followed by a for-each loop that iterates through all the bookings in the ‘bookings’ list, which contains all rental and purchase records. For each booking, the program checks if the userId matches the username of the currently logged-in user. Then another attribute called ‘purchaseType’ is defined to differentiate between rented and purchased vehicles. This is done by another ‘if’ statement within the major ‘if’ statement in which it checks if the days are greater than 0 it will display “Rented” else it displays “Purchased”.

To provide a summary of the booking details a print line command shows all the details including the total price for that specific purchase.

### logout() Method 👾



The purpose of this method is to show a summary of all bookings and purchases of the logged-in user once they log out.

Firstly, a Boolean variable ‘hasPurchases’ is created and set to false. Then, a for-each loop is used to iterate through all the bookings in the ‘bookings’ ArrayList made by the current user. Then, the same method is applied as per the viewallBookings() method to check if the car is purchased or rented.

The program then displays the vehicle model, type of purchase, and the total price. The ‘hasPurchases’ variable is then set to true. This is in case the customer makes no transactions.

A screen shot of a computer code

AI-generated content may be incorrect.

Then, outside of the major ‘if’ statement, another if statement is used to check if there are no purchases by using the (NOT) ‘!’ operator and then displaying no transactions made. Then, it resets the ‘currentCustomer’ variable to null to indicate that no user is logged in.

### getintegerInput() Method 👾

A screen shot of a computer program

AI-generated content may be incorrect.

This is one of the most important methods of the program in ensuring the user does not cause the program to crash if an invalid input is entered. In this an ‘int’ datatype is used to indicate that the return value of this method will be an integer.

It starts with a while loop with the condition set to be true, which indicates to keep asking the user for an input until a valid integer is entered. The (String enter) parameter allows the user to be prompted with a specific message before this method is called.

A Java exception case study is used through the ‘try’ command, encapsulating the area in which the user may enter an invalid input through the In.java input framework. Then the catch command is also included to specify the type of error which could occur and the message to be displayed if the user does enter such an input. This is followed by the “In.nextLine()” command, which again prompts the user to try again, asking to enter a number.

This concludes the explanation of each part of the code separately, providing a clearer understanding of the concepts in Java and how to use them effectively.

***Continued on Next Page***

# Project Codebase

import java.util.ArrayList;

import java.util.HashMap;

import java.util.InputMismatchException;

public class Project {

ArrayList<Vehicles> vehicles = new ArrayList<>();

ArrayList<Bookings> bookings = new ArrayList<>();

HashMap<String, Customer> customers = new HashMap<>();

Customer currentCustomer = null;

// The above line tracks the current customer, it will be null if no

// customer is logged in

public static void main(String[] args) {

Project project = new Project();

project.initializeData();

project.start();

}

// Initializing data in the program database

private void initializeData() {

// Sample customer data for login

customers.put("admin", new Customer("admin", "admin123"));

customers.put("Qaail", new Customer("qaail", "qaail123"));

// Normal classic cars

ClassicCar car1 = new ClassicCar("1", "Toyota Corolla", 40.0, true, false);

ClassicCar car2 = new ClassicCar("2", "Honda Civic", 42.0, true, false);

ClassicCar car3 = new ClassicCar("3", "Hyundai Elantra", 39.0, true, false);

ClassicCar car4 = new ClassicCar("4", "Nissan Sentra", 38.0, true, false);

vehicles.add(car1);

vehicles.add(car2);

vehicles.add(car3);

vehicles.add(car4);

// SUVs

SUV suv1 = new SUV("5", "Toyota RAV4", 59.0, true, true);

SUV suv2 = new SUV("6", "Honda CR-V", 60.0, true, true);

SUV suv3 = new SUV("7", "Ford Explorer", 65.0, true, true);

SUV suv4 = new SUV("8", "Chevrolet Tahoe", 70.0, true, true);

vehicles.add(suv1);

vehicles.add(suv2);

vehicles.add(suv3);

vehicles.add(suv4);

// Electric Vehicles

EV ev1 = new EV("9", "Tesla Model 3", 120.0, true, 500);

EV ev2 = new EV("10", "Nissan Leaf", 55.0, true, 240);

EV ev3 = new EV("11", "BYD Atto 3", 60.0, true, 400);

EV ev4 = new EV("12", "Ford Mustang Mach-E", 130.0, true, 300);

vehicles.add(ev1);

vehicles.add(ev2);

vehicles.add(ev3);

vehicles.add(ev4);

// Super Cars

SuperCar superCar1 = new SuperCar("13", "Porsche 911 Carrera", 1000.0, true, 293);

SuperCar superCar2 = new SuperCar("14", "Lamborghini Huracán", 1500.0, true, 325);

SuperCar superCar3 = new SuperCar("15", "Ferrari 488 GTB", 1800.0, true, 330);

SuperCar superCar4 = new SuperCar("16", "McLaren 720S", 2000.0, true, 341);

vehicles.add(superCar1);

vehicles.add(superCar2);

vehicles.add(superCar3);

vehicles.add(superCar4);

}

public void start() {

boolean run = true;

while (run) {

displayMenu();

int choice = getintegerInput("\nSelect an option: ");

if (currentCustomer == null) {

if (choice == 1) {

login();

} else if (choice == 2) {

register();

} else if (choice == 3) {

run = false;

System.out.println("Thank you for using our service. Have a great day!");

} else {

System.out.println("Invalid option.");

}

} else {

if (choice == 1) {

rentVehicle();

} else if (choice == 2) {

buyVehicle();

} else if (choice == 3) {

viewallBookings();

} else if (choice == 4) {

logout();

} else {

System.out.println("Invalid option.");

}

}

}

}

// Below are specific attributes in the program which will direct user

// based on specific input provided

// All data is private so outside classes cannot access

// Menu to be displayed when program is run if null or not

private void displayMenu() {

System.out.println("\n===== AutoNova Online =====");

if (currentCustomer == null) {

System.out.println("1. Login");

System.out.println("2. Register");

System.out.println("3. Exit");

} else {

System.out.println("1. Rent a Car");

System.out.println("2. Buy a Car");

System.out.println("3. View Bookings and Purchases");

System.out.println("4. Logout");

}

}

private void login() {

try {

System.out.println("Username: ");

String username = In.nextLine();

System.out.println("Password: ");

String password = In.nextLine();

Customer customer = customers.get(username);

if (customer != null && customer.getPassword().equals(password)) {

currentCustomer = customer;

System.out.println("Login successful!");

} else {

System.out.println("Invalid Credentials");

}

} catch (InputMismatchException e) {

System.out.println("An error occurred during login. Please try again.");

}

}

// to register a new user in the system

private void register() {

System.out.println("Enter a username: ");

String username = In.nextLine();

//Check if username already exists

if (customers.containsKey(username)) {

System.out.println("Username already exists.\nPlease try registering again.");

return;

}

System.out.println("Enter a password: ");

String password = In.nextLine();

customers.put(username, new Customer(username, password));

System.out.println("Registration successful!");

}

// For renting a car based on availability

private void rentVehicle() {

System.out.println("\nAvailable Vehicles:");

for (int i = 0; i < vehicles.size(); i++) {

Vehicles vehicle = vehicles.get(i);

if (vehicle.isAvailable()) {

System.out.println((i + 1) + ": " + vehicle);

}

}

int carNumber = getintegerInput("Select a car(number): ") - 1;

if (carNumber < 0 || carNumber >= vehicles.size() || !vehicles.get(carNumber).isAvailable()) {

System.out.println("Invalid selection");

return;

}

Vehicles selectedVehicle = vehicles.get(carNumber);

int days = getintegerInput("Rental days: ");

double totalPrice = selectedVehicle.getPricePerDay() \* days;

System.out.println("Total price: $" + totalPrice);

System.out.println(("Confirm booking? (Y/N): "));

String confirmation = In.nextLine();

if (confirmation.equals("Y")) {

bookings.add(new Bookings(currentCustomer.getUsername(), selectedVehicle.getModel(), days, totalPrice));

selectedVehicle.setAvailable(false);//Removes vehicle from database

System.out.println("Booking confirmed!");

} else {

System.out.println("Booking cancelled");

}

}

// If customer wants to buy a car

private void buyVehicle() {

System.out.println("\nAvailable Vehicles for Purchase:");

System.out.println("=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=");

for (int i = 0; i < vehicles.size(); i++) {

Vehicles vehicle = vehicles.get(i);

if (vehicle.isAvailable()) {

System.out.println("---------------------------------------");

System.out.println("Vehicle Number: " + (i + 1));

System.out.println("Model: " + vehicle.getModel());

System.out.println("Type: " + vehicle.getType());

System.out.println("Price: $" + vehicle.getPricePerDay() \* 120); // To convert to actual price

if (vehicle.getType().equals("Classic Car")) {

System.out.println("This is a Classic car.");

} else if (vehicle.getType().equals("Super Car")) {

System.out.println("This is a Super car.");

} else if (vehicle.getType().equals("SUV")) {

System.out.println("This is a SUV.");

} else if (vehicle.getType().equals("EV")) {

System.out.println("This is an Electric vehicle.");

}

System.out.println("-----------------------------------");

}

}

System.out.println("=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=");

int carNumber = getintegerInput("Select a vehicle to buy(number): ") - 1; // Removes a vehicle from database

if (carNumber < 0 || carNumber >= vehicles.size() || !vehicles.get(carNumber).isAvailable()) {

System.out.println("Invalid selection.");

return;

}

Vehicles selectVehicles = vehicles.get(carNumber);

double totalPrice = selectVehicles.getPricePerDay() \* 120;

// Insurance Option Prompt

System.out.println("Would you like to add insurance? (Y/N): ");

String insuranceChoice = In.nextLine();

if (insuranceChoice.equals("Y")) {

double insuranceFee = selectVehicles.getpriceofInsurance();

totalPrice += insuranceFee;

System.out.println("Insurance added for extra $" + insuranceFee);

} else {

System.out.println("Proceeding without insurance.");

}

// Extended Warranty option

System.out.println("Would you like to add an extended Warranty? (Y/N): ");

String warrantyExtension = In.nextLine();

if (warrantyExtension.equals("Y")) {

double warrantyFee = 550; // Fixed for any vehicle

totalPrice += warrantyFee;

System.out.println("Warranty added for an extra $" + warrantyFee);

} else {

System.out.println("No Warranty extension applied");

}

// Apply 15% discount if total price is over $10,000

if (totalPrice > 10000) {

double discount = totalPrice \* 0.15;

totalPrice -= discount;

System.out.println("A 15% discount has been applied! Discount amount: $" + discount);

}

// Preset date for the purchase

String purchaseDate = "2025-04-15";

// Addition of extra purchases

System.out.println("Total price: $" + totalPrice);

System.out.println("Confirm purchase? (Y/N): ");

if (In.nextLine().equals("Y")) {

selectVehicles.setAvailable(false);

bookings.add(new Bookings(currentCustomer.getUsername(), selectVehicles.getModel(), 0, totalPrice));

System.out.println("Purchase confirmed on " + purchaseDate + "!");

} else {

System.out.println("Purchase cancelled.");

}

}

//Ability for user to check purchases and bookings

private void viewallBookings(){

//Display bookings for the logge din customer

System.out.println(("Bookings for user: ")+currentCustomer.getUsername());

for(Bookings booking:bookings){

if(booking.getUserId().equals(currentCustomer.getUsername())){

//To differentiate between purchased and rented transactions

String purchaseType;

if(booking.getDays()>0){

purchaseType = "Rented";

}else{

purchaseType = "Purchased";

}

System.out.println("Vehicle: "+booking.getVehicleModel()+" | Type: "+purchaseType+" | Total: $" + booking.getTotalPrice());

}

}

}

//Show summary of purchases when user logs out

private void logout(){

System.out.println("\nSummary of Purchases");

System.out.println("========================================");

boolean hasPurchases = false;

for(Bookings booking :bookings){

if(booking.getUserId().equals(currentCustomer.getUsername())){

String purchaseType;

if(booking.getDays()>0){

purchaseType = "Rented";

} else {

purchaseType = "Purchased";

}

System.out.println("----------------------------------------");

System.out.println("Car: " + booking.getVehicleModel());

System.out.println("Type: " + purchaseType);

System.out.println("Total: $" + booking.getTotalPrice());

System.out.println("----------------------------------------");

//Incase customer has no transactions

hasPurchases= true;

}

}

if(!hasPurchases){

System.out.println("No transactions made.");

}

System.out.println("========================================");

System.out.println("Logged out");

currentCustomer= null;

}

//Method for obtaining user input from Customer incase of wrong input

private int getintegerInput(String enter){

while(true){

System.out.println(enter);

try{

return In.nextInt();

} catch (InputMismatchException e){

System.out.println("Invalid Input,Please enter a number:");

In.nextLine();//Retry again due to invalid input

}

}

}

}

// Parent class for all types of vehicles

class Vehicles {

private String id; // Vehicle number in the database

private String model;

private String type;

private double pricePerDay;

private boolean availability;

private double priceofInsurance;

// Constructor for the parent class

public Vehicles(String id, String model, String type, double pricePerDay, boolean availability,

double priceofInsurance) {

this.id = id;

this.model = model;

this.type = type;

this.pricePerDay = pricePerDay;

this.availability = availability;

this.priceofInsurance = priceofInsurance;

}

// Getters for each attribute

public String getId() {

return id;

}

public String getModel() {

return model;

}

public String getType() {

return type;

}

public double getPricePerDay() {

return pricePerDay;

}

public boolean isAvailable() {

return availability;

}

public double getpriceofInsurance() {

return priceofInsurance;

}

// Setter for availability of that vehicle

public void setAvailable(boolean availability) {

this.availability = availability;

}

// toString method to be utilized in child classes

@Override

public String toString() {

return getModel() + " [" + getType() + "] : $" + getPricePerDay() + "/day";

}

}

// ClassicCar class (for regular vehicles)

class ClassicCar extends Vehicles {

private boolean isHybrid;

// The above checks if the car is hybrid type or not

// Constructor

public ClassicCar(String id, String model, double pricePerDay, boolean availability, boolean isHybrid) {

super(id, model, "Classic Car", pricePerDay, availability, 300);

// Initializing attributes to the parent class due to certain attributes being

// pre defined

this.isHybrid = isHybrid;

}

// Getter for whether car is hybrid or not

public boolean isHybrid() {

return isHybrid;

}

// toString method for Classic cars

@Override

public String toString() {

String hybridOption;

if (isHybrid) {

hybridOption = "Yes";

} else {

hybridOption = "No";

}

return super.toString() + ", Hybrid Option: " + hybridOption;

}

}

// SuperCar class for exclusive cars

class SuperCar extends Vehicles {

private double topSpeed;

// To indicatee top speed in km/h

// Constructor

public SuperCar(String id, String model, double pricePerDay, boolean available, double topSpeed) {

super(id, model, "Super Car", pricePerDay, available, 1000);

this.topSpeed = topSpeed;

}

// Getter for topSpeed of the Supercar

public double getTopSpeed() {

return topSpeed;

}

// toString method for Supercar class

@Override

public String toString() {

return super.toString() + ", Top Speed: " + topSpeed + " km/h";

}

}

// SUV class for all types of SUVs

class SUV extends Vehicles {

private boolean hasAllWheelDrive;

// The above ensures if the type of SUV comes with all-wheel drive option

// Constructor

public SUV(String id, String model, double pricePerDay, boolean availability, boolean hasAllWheelDrive){

super(id, model, "SUV", pricePerDay, availability, 600);

}

// Getter for All-wheel drive option in SUVs

public boolean hasAllWheelDrive() {

return hasAllWheelDrive;

}

// toString method for SUVs using parent class

@Override

public String toString() {

String allwheelDrive;

if (hasAllWheelDrive) {

allwheelDrive = "Yes";

} else {

allwheelDrive = "No";

}

return super.toString() + " , All-wheel drive: " + allwheelDrive;

}

}

// EV Class for all electric vehicles

class EV extends Vehicles {

private double rangePerCharge;

// Range in km per one charge

// Constructor

public EV(String id, String model, double pricePerDay, boolean availability, double rangePerCharge){

super(id, model, "EV", pricePerDay, availability, 250);

// EV will have slightly lower insurance price

this.rangePerCharge = rangePerCharge;

}

// Getter method for range per charge in km

public double getRangePerCharge() {

return rangePerCharge;

}

// toString method for EV class

@Override

public String toString() {

return super.toString() + ", Range per charge: " + rangePerCharge + " km";

}

}

// Customer class to store information

class Customer {

private String username;

private String password;

// These attributes ensures security when logging in

// Constructor

public Customer(String username, String password) {

this.username = username;

this.password = password;

}

// Getters for the attributes

public String getUsername() {

return username;

}

public String getPassword() {

return password;

}

// toString method not required here

}

// Bookings class to store all vehicles information

class Bookings {

private String userId;

private String vehicleModel;

private int days;

private double totalPrice;

ArrayList<Bookings> listofBookings;

// Above Arraylist will be an instance variable which will store all bookings

// Constructor

public Bookings(String userId, String vehicleModel, int days, double totalPrice) {

this.userId = userId;

this.vehicleModel = vehicleModel;

this.days = days;

this.totalPrice = totalPrice;

}

// Constructor to initialize booking list

public Bookings() {

this.listofBookings = new ArrayList<>();

}

// Method to add a single booking to the list

public void addBooking(Bookings booking) {

listofBookings.add(booking);

}

// Getters for Bookings class attributes

public String getUserId() {

return userId;

}

public String getVehicleModel() {

return vehicleModel;

}

public int getDays() {

return days;

}

public double getTotalPrice() {

return totalPrice;

}

// Method to display all bookings for a single user

public void displayBookings(String userId) {

System.out.println("Bookings for user: " + userId);

for (Bookings booking : listofBookings) {

// Use the getUserId() method to access the userId field

if (booking.getUserId().equals(userId)) {

System.out.println(booking);

}

}

}

// toString method to represent a booking

@Override

public String toString() {

return "User: " + userId + " | Vehicle: " + vehicleModel + " | Days: " + days + " | Total: $" + totalPrice;

}

}

# Problems Encountered

## Invalid Input

The problem faced in this case was the issue that the user would enter an invalid input when asked to navigate through the program and it would cause the whole program to crash. This left me wondering how senior developers resolve such issues.

To fix this issue, I recalled that we had studied Java exceptions in Week 7. With the assistance of AI and YouTube, I was able to comprehend the implementations of such cases. This enabled me to prompt the user with a message indicating that the input was invalid and prompting them to try again.

## Nullpointer Exception Case

This is one of the most common errors in Java code which require extreme supervision. I had an issue when I tried to retireve data from ArrayLists or HashMaps which did not exist.

To resolve this issue, I used YouTube and AI to learn how to add null checks of different types to prevent such cases from occurring. One of these checks is when a customer’s username doesn’t exist, and now it displays “Invalid credentials.”

***Continued on Next Page***

# Runtime Outputs (Terminal)

**Main Menu:**

A screen shot of a black screen

AI-generated content may be incorrect.

**Login Menu:**

**A screen shot of a computer

AI-generated content may be incorrect.**

**Rent a Car Option:**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**Buy a Car Option:**

**A screenshot of a black screen

AI-generated content may be incorrect.**

**View Bookings Option:**

**A black background with white text

AI-generated content may be incorrect.**

***Continued on Next Page***

**Logout Option:**

**A screen shot of a computer screen

AI-generated content may be incorrect.**

**Exit:**

**A black screen with white text

AI-generated content may be incorrect.**