C++ ASSIGNMENT REPORT

The following is a report about the Rental Vehicle System code I did in C++.

PROGRAM STRUCTURE

In my code, the main program is structured around three main classes: ‘RentalCompany’, ‘Customer’ and ‘Vehicle’. The ‘Vehicle’ class is a base class and is generic (allowing for polymorphism). I then made specific sub-classes like car, van and minibus derived from the Vehicle class (their main difference is the late fee rate assigned to each of them - £10, £20 and £30 respectively). The RentalCompany aggregates the Vehicle and the Customer classes. I made many. The program appears as a menu showing different activities that can be done on the program like customer and vehicle addition and removal, as well as renting/returning a vehicle, displaying all available customers or available vehicles (and being able to search for specific vehicles), and reserving a vehicle if it is already rented (and alerting the customer when that vehicle is returned).

I chose to use a map to allow for faster searching. I used a vector container to store vehicles and customers because it can be sized dynamically and so is not memory-wasting, and because I could use methods like my ‘findcarbyId’ and findcustomerbyId’ to iterate through the vectors to find anything with less complexity O(log n). I also stored the customers in a map so that they can be easily looked up or called upon by their customer ID.

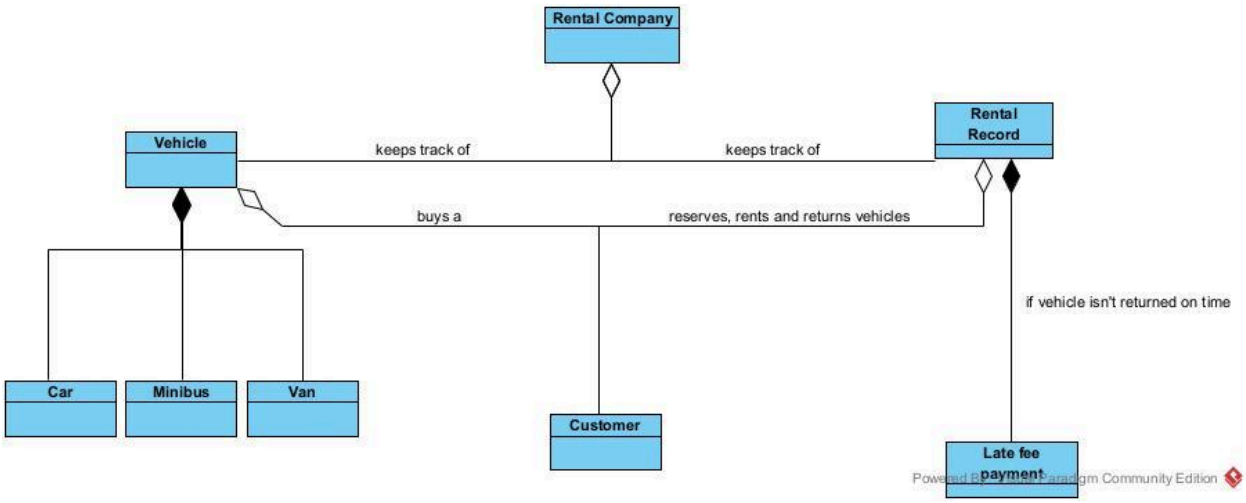
FUNCTIONAL HIGHLIGHTS

1. I implemented a void function ‘setDueDate’ which checks the time returned(as an input) against the due date to return(also as an input). I then implemented a ‘calculateLateFee’ function which identifies the number of days late and multiplies it by the standard fees for the type of vehicle it is.I made sure it didn’t give a fee if it was returned before or on the due date.

2. For error handling, I implemented try-catch blocks. I also put many inputs in loops (while, if, for) and made sure they were validated properly. For example, I put the ID input in a loop to make sure the ID is between 2 and 4 characters, that its first character is a capital letter ‘V’ and that the remaining characters are numbers. I also made sure for the passenger and storage capacity that any negative input or 0 will be flagged. I made sure duplicate vehiclesIds were avoided unless they want to replace the existing vehicle. I also implemented my code so that that once a vehicle has been rented it won’t be able to be rented again. All these help to reduce runtime errors and user frustration. I incorporated the #include <regex > library to validate input length and characters. In my implementation of the returnVehicle function, I used a regular expression pattern idPattern to validate the format of both the customer ID and vehicleID before proceeding with the return process. I also used it for validating only three numerical values in my customer ID.

3. I used many void functions to manage the reservation status of vehicles, toggling between reserved and available (and not available) as needed. I also created a queue system which stores the reservation list and alerts the top customer when the vehicle is returned ( prompting them immediately to rent the vehicle). I also incorporated some input validations in case of potential mistakes or wanting to update their details, like re-prompting the customer to confirm wishing to update their existing details, or asking if the customer if they want to rent or reject the car they reserved when it is now free.

4. Ultimately the choice of a menu-interface was to make it more user friendly, with many instructions and clear representations of what they re doing/to do.




INSTRUCTIONS ON HOW TO USE THE CODE:

1. It is very simple and is a menu based interface. Select the number of what you want to do and other instructions (if needed ) will appear. Remember that certain inputs are very specific (Like you should start your vehicleID with a capital letter ‘V’).