Faculty of
Computing, Engineering and
the Built Environment

BIRMINGHAM CITY
University

Undergraduate Programme

Academic Year 2014-2015

Coursework: Team Project

Module: *CMP2515 Software Design UG2*

School: **Computing, Telecommunication and Networks**

Module Co-ordinator: **Professor Zhiming Liu**

Setup Date: **20/02/2015**

Submission Date: **21/04/2015**

Team Number: T13

Team Leader: Gurpindervir Rai

Team Members: Gurpindervir Rai, Zacharias Nur, Kasim Mehmood, Qasim Naveed Malik, Shamem Miah, Sarmad Rafiq

Table of Contents

Bvis Car Hire Company: An Introduction

Bvis Car Hire Company currently uses a paper-based system to store details of its customers, the company's fleet of cars as well as its hire transactions. This report gives a detailed overview of how a successful computer-based system can be implemented to replace the current manual system. The system which will be discussed in this report will therefore be paperless. This new system will perform the following tasks:

- Register a new customer
- Record that a particular car has been hired
- Record that a particular car has been returned
- Calculate the cost based on the daily hire rate
- Display the appropriate details, and print out a receipt
- Log a completed hire.
- Record a service for a particular car, together with the date of the service, the type of service, and the name of the mechanic responsible.
- Remove a customer.
- Add a new car to the fleet.
- Delete a car that is no longer in the hire fleet.
- Add a mechanic who has joined the company.
- Remove the details of a mechanic who has left the company.
- Determine if a particular car is due for a particular service.
- List the information (history) about all hires for a specified car.
- List the information (history) about all services that a specified car has had.

I. The initial requirements understanding

An object-oriented development approach will need to be taken to ensure that the following tasks can be implemented into the system without any major problems. Due to the complex nature of the system required, an object-oriented development approach is needed in order to look at the interaction and collaboration between multiple objects within the programme. Assessing this is key in determining how to implement the new system as going from a paper-based system to a computer-based system can create many problems. For example, during the implementation of the computer-based system several changes will need to be made to the design as generally during the development itself, developers are able to master the problem domain. Hence an object-oriented development approach would be best suited.

System Functions:

Basic Functions of Bvis Car Hire Company:

Ref #	Function	Category
1	Register a new customer	evident
2	Record that a particular car has been hired	evident
3	Record that a particular car has been returned	evident
4	Calculate the cost based on the daily hire rate	evident
5	Display the appropriate details, and print out a receipt	evident
6	Log a completed hire.	hidden
7	Record a service for a particular car, together with the date of the service, the type of service, and the name of the mechanic responsible.	evident
8	Remove a customer.	evident
9	Add a new car to the fleet.	evident
10	Delete a car that is no longer in the hire fleet.	evident
11	Add a mechanic who has joined the company.	evident
12	Remove the details of a mechanic who has left the company.	evident
13	Determine if a particular car is due for a particular service.	evident
14	List the information (history) about all hires for a specified car.	evident
15	List the information (history) about all services that a specified car has had.	evident

Payment Functions:

Ref #	Function	Category
16	Handle cash payments, capturing amount tendered and calculating balance due.	evident
17	Log the payment.	hidden

批注 [ZL1]: Generally good, but some of the functions are actually use cases, such as register a customer

Essential use cases

The essential use cases which cover and support the understanding of the required functions in the problem description are:

- Register customer ~~details~~
- Remove customer ~~details~~
- Hire car
- Cancel hire
- Add mechanic ~~details~~
- Remove mechanic
- ~~details~~

批注 [ZL2]: I would also add “Return a car” an essential use case as it handles the payments too

带格式的：正文， 无项目符号或编号

High-level use cases

Register customer details:

Use Case: Register customer details

Actor: Employee and customer

Purpose: Customer wants to register with company

Overview:

1. A customer arrives at the company.
2. The customer provides their details.
3. The company records their details in the database.
4. The customer then leaves.

Remove customer details:

Use Case: Remove customer details

Actor: Employee and customer

Purpose: Employee removes customer from company database

Overview:

1. A customer has been inactive for a number of years or a customer wishes to be removed from the company’s database.
2. The customer’s details are then removed from company’s database.

Hire Car:

Use Case: Hire Car

Actor: Employee and customer

Purpose: Customer wants to hire a car

Overview:

1. A customer arrives at the customer service desk, with their licence.
2. An employee records the customer's details, which includes name, address and car details (which is selected by the customer).
3. On completion the customer leaves with the car.

批注 [ZL3]: This is not informative enough

Cancel Hire:

Use Case: Cancel Hire

Actor: Employee and customer

Purpose: Customer wants to cancel a hire

Overview:

1. A Customer arrives at the customer service desk to cancel a hire.
2. An employee records their details, reason for cancelling and takes the cancellation fee.

Add mechanic details:

Use Case: Add mechanic details

Actor: Mechanic and the manager

Purpose: Adding a mechanic to the company's database

Overview:

1. A new mechanic arrives at the company with their details including their driving license.
2. The manager verifies the mechanic's driving license.
3. A new mechanic is hired by the manager and their details are added to the company's database.

Remove mechanic details:

Use Case: Remove mechanic details

Actor: Mechanic and the manager

Purpose: A mechanic has left the company and needs to be removed from the company's database.

Overview:

1. The mechanic may be leaving the company through their own choice or they may have been removed by the company (redundancy etc.)
2. The manager removes the mechanics details from the company's database.

批注 [ZL4]: No mention about database should be included. "Removes the mechanics" is much better than "removes the mechanics details" as "mechanics" is an object, the "details" are it attributes

Expanded version of use cases

Register customer ~~details:~~

Typical Course of Events	
Actor Action	System Response
1. This use case begins when a customer arrives at the company and wishes to hire a car.	
2. The employee asks the user to fill in their details including name, address, telephone number and driving licence number on the hire form.	
3. The employee records the customer's details after verifying the customer's license.	4. Acknowledgement that the customer's detailed have successfully been stored in the company's database.
5. The employee acknowledges that the customer's details have been stored successfully.	

批注 [ZL5]: We did not say this is a database design

Alternative courses:

- Line 3: If the customer's driving license is not valid then do not proceed with entering in their details into the company's database.

Remove customer details:

Typical Course of Events	
Actor Action	System Response
1. The customer requests the removal of their details from the company's database.	
2. The employee removes the customer details from the company's database.	3. The customer's details are then removed from company's database. 4. The action will be saved and the database will be updated.
	5. The employee will be notified that the action has been completed.
6. The employee acknowledges the notification from the system.	

带格式的：突出显示

带格式的：突出显示

Alternative Courses:

- If the customer has been inactive for a long period of time then the employee may proceed to remove their details from the company's database.

Hire car:

Typical Course of Events	
Actor Action	System Response
1. This use case begins with the customer requesting a car hire.	
2. The customer enters in their details.	
3. The employee enters the customer's details into the database.	4. The system looks up the customer in the database and finds them.
5. The customer selects a specific car.	
6. The employee searches the database to see whether the car is available.	7. The system checks the availability of the car.
8. The employee informs the customer that the car is available.	
9. The hire terms are proposed between the customer and the employee.	
10. The employee and customer accept the hire terms.	
11. The employee enters the hire terms into the system.	12. The hire terms are stored.
13. The employee asks the customer for the cash payment.	
14. The customer pays the amount.	15. The system logs the sale.
	16. The receipt is printed.
17. The customer receives the receipt.	

批注 [ZL6]: Needs to be more informative: dates, mileage…

带格式的：突出显示

Alternative Courses:

- Line 8: The car may not be available so the customer is informed to choose another car.
- Line 10: If the hire terms are not accepted then the hire cannot be processed.

Cancel Hire:

Typical Course of Events	
Actor Action	System Response
1. The customer requests to cancel a hire, giving the reason for cancelling.	
2. The customer enters in their details.	
3. The employee enters in the customer's details into the system.	4. The customer is identified on the company's database.
5. The employee requests the cancellation fee from the customer.	
6. The customer pays the amount in cash.	7. The system logs the cancelling of the hire.
	8. The receipt is printed
9. The customer receives the receipt	
10. The employee tells the system that the cancelling of the hire has been successfully completed.	11. Acknowledgement that the hire was successfully cancelled.

Alternative Courses:

- Line 4: The customer is not identified in the company's database so the customer is not the rightful owner of the car they are trying to cancel.
- Line 6: If the customer doesn't pay the cancellation fee in cash then the hire cannot be cancelled.

Add mechanic details:

Typical Course of Events	
Actor Action	System Response
1. The new mechanic arrives at the company and enters in their details including name, address and phone number.	
2. The manager asks the mechanic for their driving licence.	
3. The manager must check the DVLA (Driver and Vehicle Licensing Agency) records to ensure that the mechanic's license is valid.	
4. The manager then enters the mechanic's details into the system.	5. The mechanic's details are stored into the company's database.

Alternative Courses:

- Line 3: If the mechanic's license is not valid then the mechanic cannot be hired and consequently cannot be added to the company's database.

Remove mechanic details:

Typical Course of Events	
Actor Action	System Response
1. The mechanic is either leaving the company or the mechanic has been fired or made redundant.	
2. The manager removes the mechanic's details from the company's database.	3. The mechanic's details are successfully removed from the company's database.

Alternative Courses: None

带格式的： 突出显示

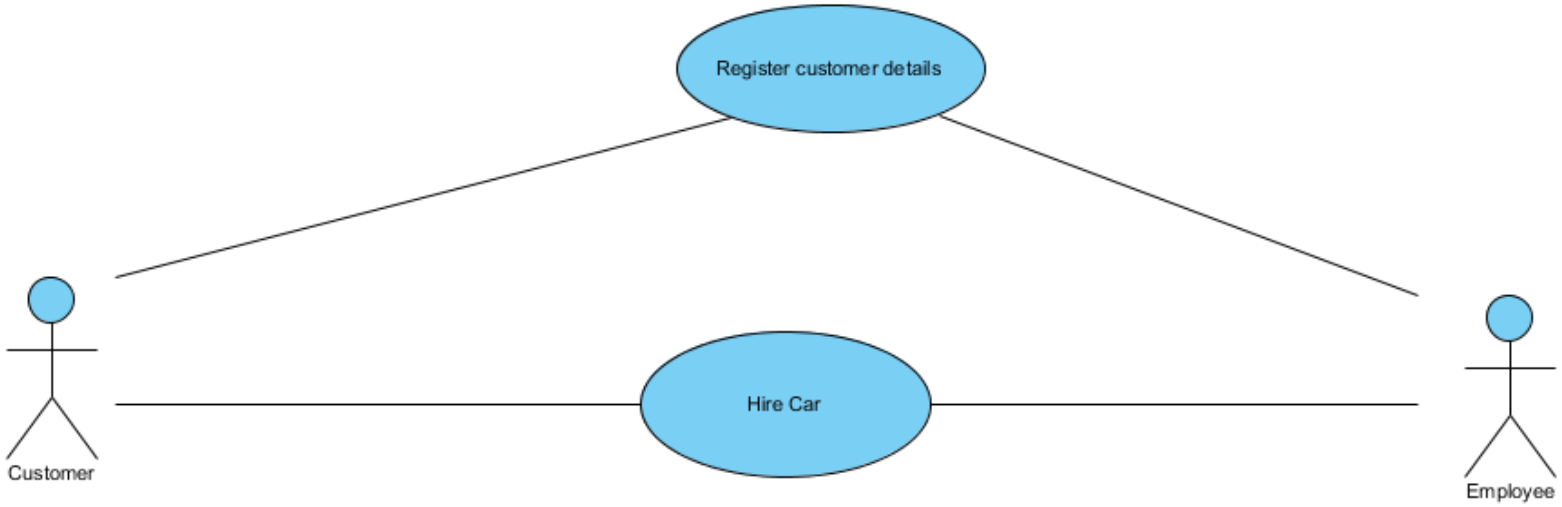
批注 [ZL7]: Use case descriptions are in general good, but should be more informative, and should especially avoid of design issues such as "database", also the understanding of object-orientation should be improved

Use Case Diagrams

批注 [ZL8]: More interesting use case diagrams would be desirable such as those with that include use cases like payment

Hiring a car:

The use case diagram for the interaction between the customer and the employee when hiring a car.

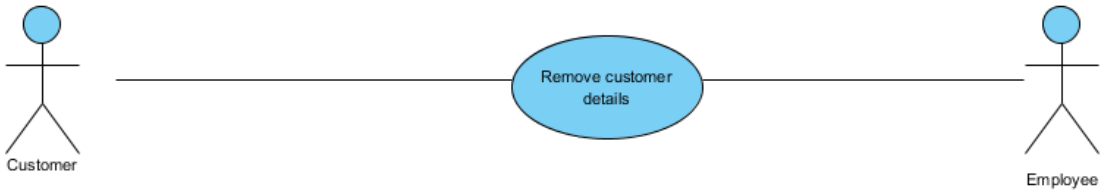


Cancelling a Hire:

The use case diagram for the interaction between the customer and the employee when cancelling a hire.

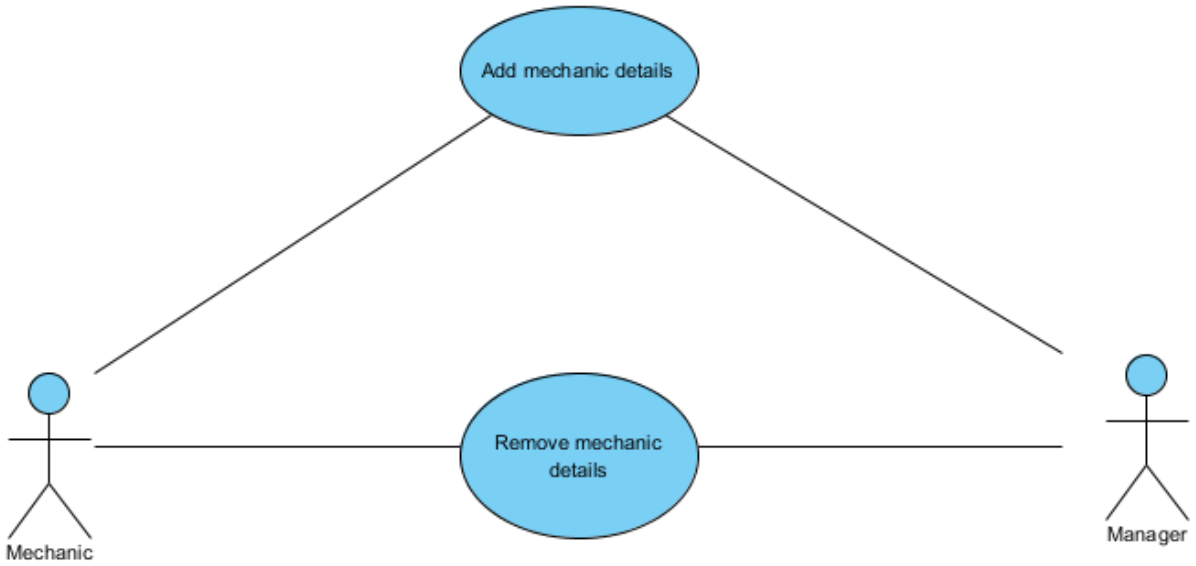
Removing Customer Details:

The use case diagram for the interaction between the customer and the employee when removing the customer's details from the company's database.



Adding/Removing Mechanic Details:

The use case diagram for the interaction between the mechanic and the manager when wishing to add a new mechanic's details or to remove a mechanic's details from the company's database.



Identifying classes (concepts), associations, and attributes in the application domain.

Classes (concepts):

- Customer
    - RegularCustomer
    - Non-RegularCustomer
- Hire System
- Hire Agreement
- Car
- Service
- Management
- Hire Class
- Mechanic

The strategy we used to identify was to find concepts from the scenario, we divided the class and objects into different categories defined by their instances, we also identified noun and noun phrases in the scenario and found that some of these were the same as the classes we identified from the list created. We ended up with the classes stated above. For the customer class we determined that this was a super-class because it consisted of two types of customer; regular and non-regular customers.

Associations:

- Customer and HireSystem
- Customer and HireAgreement
- HireSystem and HireAgreement
- HireSystem and Car
- HireAgreement and Car
- Car and HireClass
- Car and Service
- Service and Mechanic
- Mechanic and Management

When identifying the associations we took into account which of the classes will need to rely on each other for information or to perform functions/methods. With this we found that the above associations were good at representing our systems functionality and made sense in a logical approach.

Attributes:

Customer – RegularCustomer Attributes:

- Name
- Address
- PhoneNumber

Customer – Non-RegularCustomer Attributes:

- Name
- Address
- PhoneNumber
- DrivingLicence

HireSystem Attributes:

- HireDate
- ReturnDate

HireAgreement Attributes:

- CashPayment
- ActualReturnDate
- ActualMileage

Car Attributes:

- Registration
- Make
- Model
- EngineCapacity
- Class
- DateOfRegistration
- Mileage

HireClass Attributes:

- Daily
- Weekly
- Monthly

Service Attributes:

- DateOfService
- ServiceType

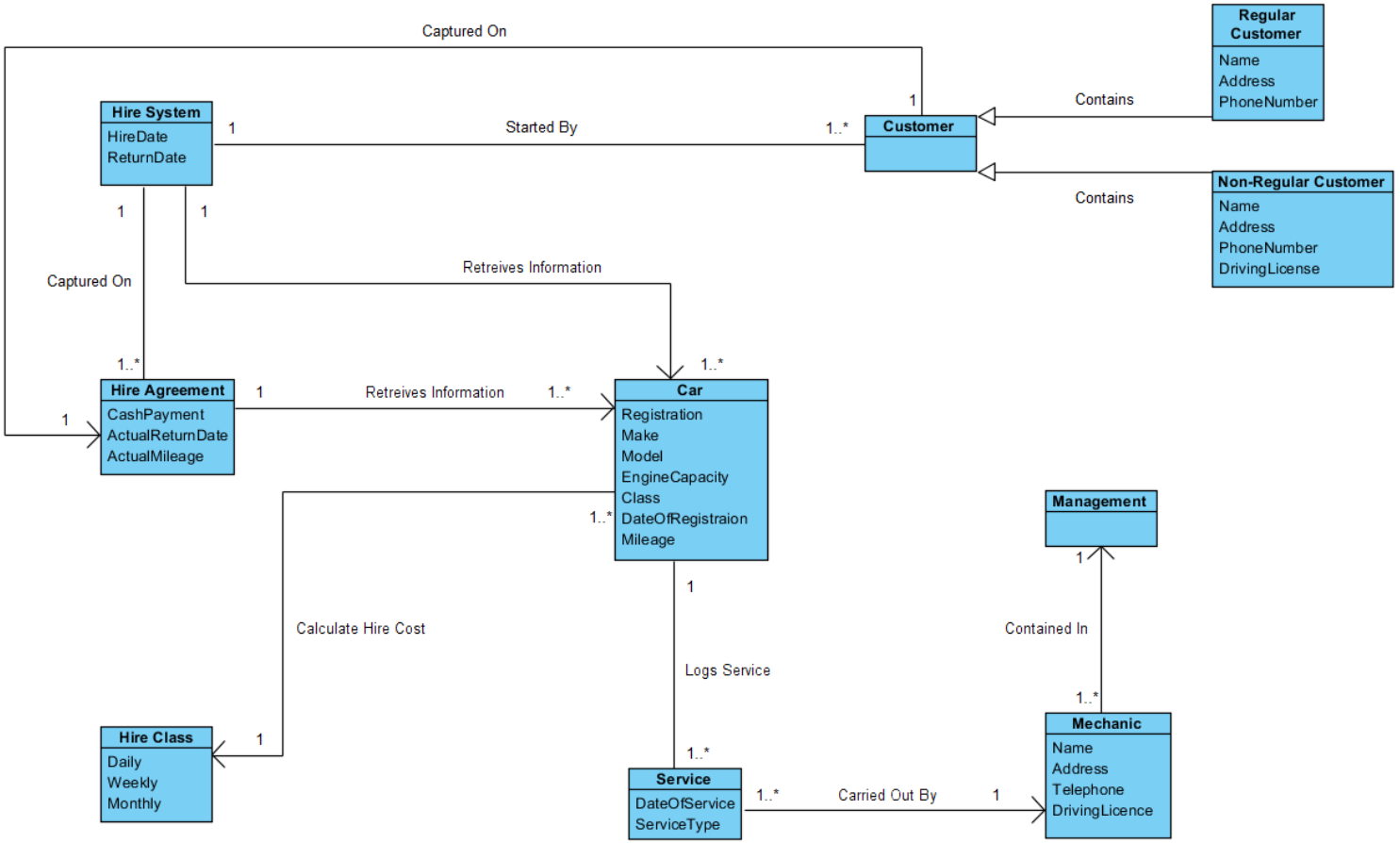
Mechanic Attributes:

- Name
- Address
- Telephone
- DrivingLicence

Management Attributes (empty):

By identifying the concepts listed within the scenario, we acknowledged certain attributes which can be referred to the concepts and it makes sense to have these implemented. The use cases gave us a clarification on the attributes as it is relatable to the actions. A majority of these listed attributes were also extracted from the scenario itself.

Conceptual Model/Conceptual Class Diagram:



批注 [ZL9]: Reasonably good, but there are some syntactical confusion, such as the direction of the associations

Also, “class” is an attribute of Car, but you also have a class Hire Class

II. Functionality Analysis of System Operations

Registering customer details system operations:

- enterDetails(Name, Phone, Address, Driving Licence Number)
- verifyDrivingLicense()
- enterCustomerDetails()

Remove customer details system operations:

- requestRemoval()
- removeCustomerDetails()

Hire car system operations:

- requestHire()
- enterDetails(Name, Phone, Address)
- enterCustomerDetails()
- selectCar(Make, Model)
- checkCarAvailability()
- hireTerms(HireDate, ReturnDate)
- agreeHireTerms()
- enterHireTerms()
- finishSale()
- cashPay(Amount):Amount
- printReceipt()

- recieveReceipt()

Cancel hire system operations:

- requestCancelHire()
- enterDetails(Name, Phone, Address)
- enterCustomerDetails()
- requestCancellationFee()
- cashPay(Amount):Amount
- printReceipt()
- recieveReceipt()
- finishCancelHire()

Add mechanic details system operations:

- enterMechanicDetails(Name, Address, Phone)
- verifyMechanicDrivingLicense()
- storeMechanicDetails()

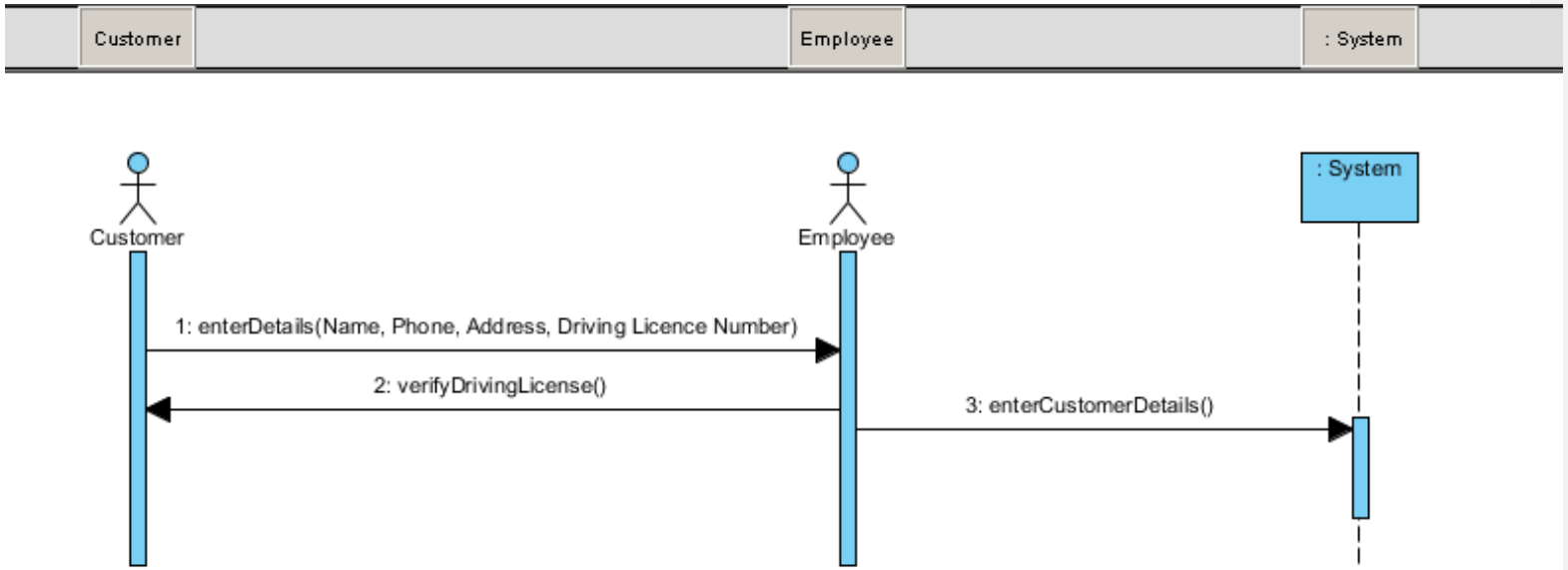
Remove mechanic details system operations:

- mechanicLeaving()
- removeMechanicDetails()

Use Case Sequence Diagrams

Register customer details:

The use case sequence diagram for registering a new customer.



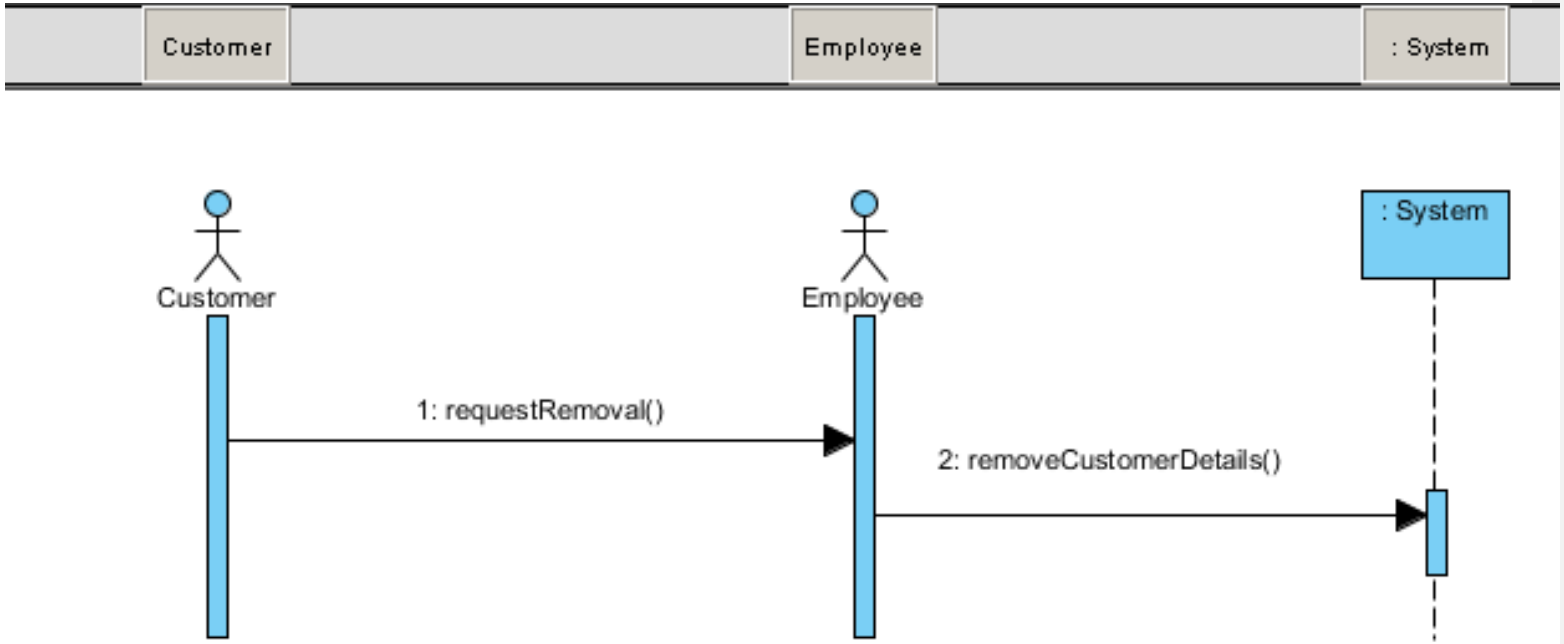
批注 [ZL10]: Actors that are not directly interact with the system should not be shown, and interactions among actors should not shown.

There are confusions, such as method 1. enterDetails() is to be executed by the system or by the employee manually? And who does verifyDrivingLicense

Apart from this confusion, though it is a serious one, the sequence diagrams are meaningful

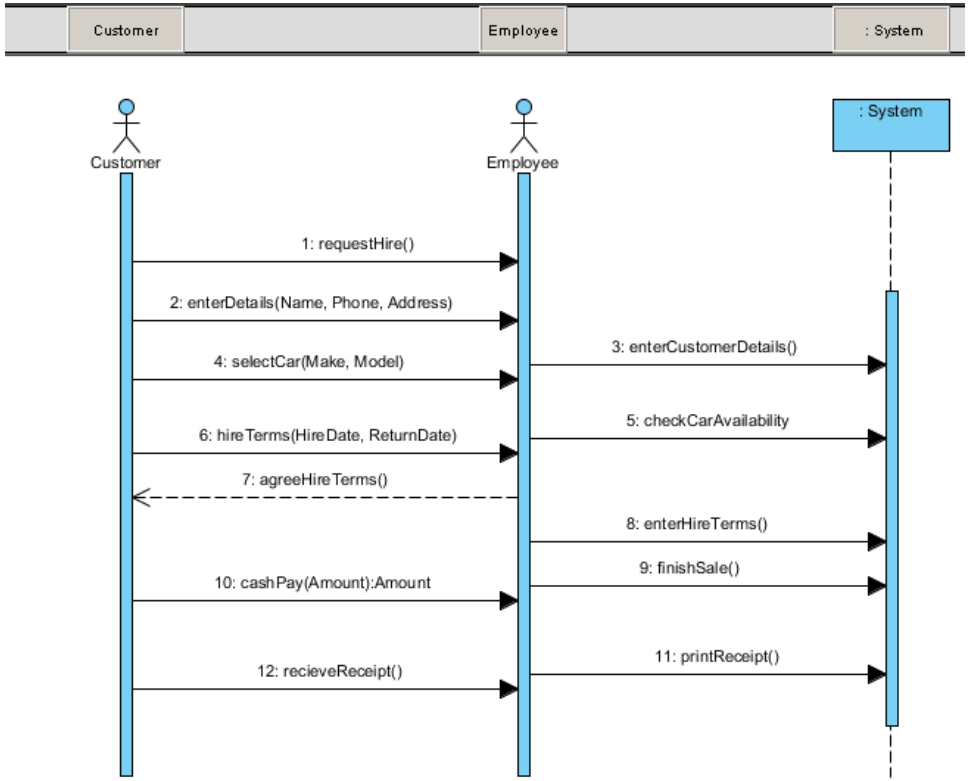
Remove customer details:

The use case sequence diagram for removing customer details.



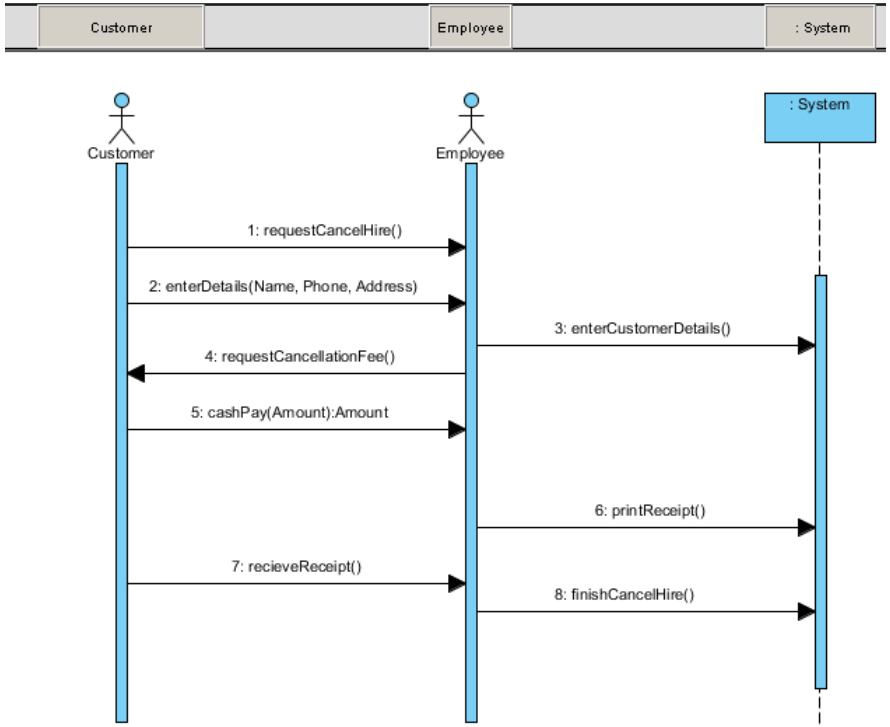
Hire Car:

The use case sequence diagram for hiring a car.



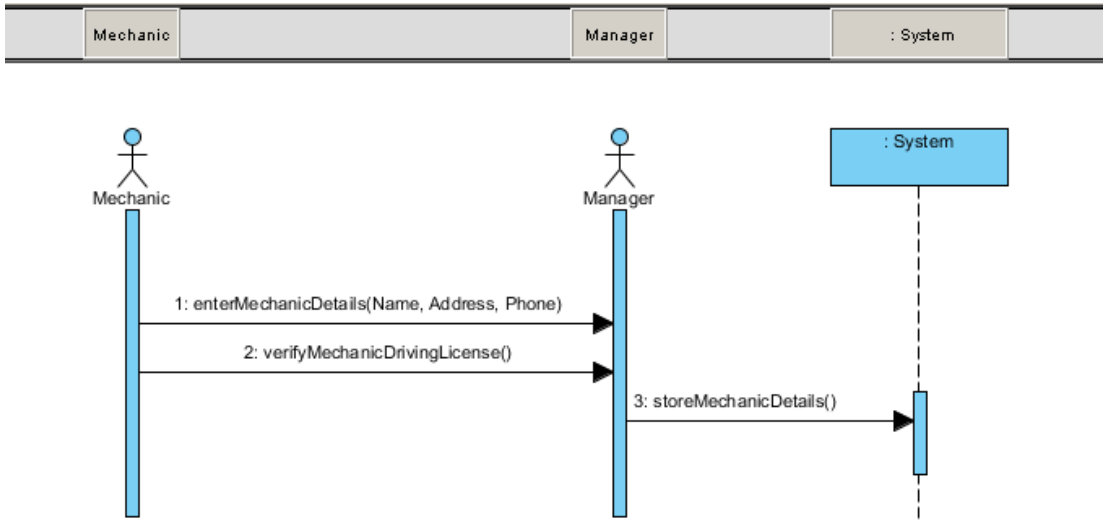
Cancel Hire:

The use case sequence diagram for cancelling a hire.



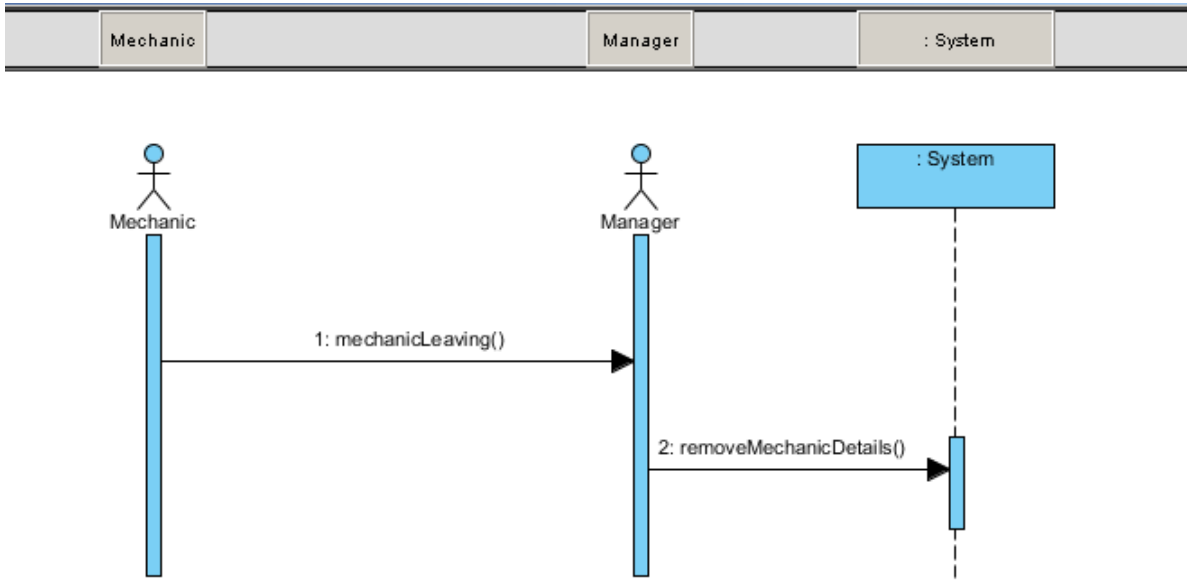
Add Mechanic Details:

The use case sequence diagram for adding a new mechanic.



Remove Mechanic Details:

The use case sequence diagram for removing the mechanic's details.



Contracts

Contract

Name: startUp()

Responsibilities: The startUp() is used to initialise the system.

Cross References: Use cases:

Note:

Exceptions:

Pre-conditions:

Post-conditions:

批注 [ZL11]: What are they?

Contract

Name: enterDetails(Name, Phone, Address, Driving Licence Number)

Responsibilities: Collects the customer's details.

Cross References: Use Cases: Register customer details

Note:

Exceptions: If phone number is entered in an invalid format indicate an error. If the name does not match the name on the license also indicate an error.

Pre-conditions: Phone number is valid and the name entered matches the license details.

Post-conditions: ???

- If the details entered are not already in the company's record then a new customer is created.

Contract

Name: enterCustomerDetails()

Responsibilities: The employee enters in the customer's details into the system.

Cross References: Use cases: Register customer details

Note: The details are only entered into the system once the customer's driving licence has been verified.

Exceptions:

Pre-conditions: The customer's driving license number is checked externally by the DVLA (Driver and Vehicle Licensing Agency) records. Once this has been verified the employee may proceed to enter in the customer's details into the system.

Post-conditions:

Contract

Name: verifyDrivingLicense()

Responsibilities: The employee must check the DVLA (Driver and Vehicle Licensing Agency) records to ensure that the customer's license is valid.

Cross References: Use cases: Register customer details

Note:

Exceptions:

Pre-conditions: The customer has entered in their details including their driving license number.

Post-conditions: The driving license is valid.

Contract

Name: requestRemoval()

Responsibilities: The customer wishes to be removed from the company's database.

Cross References: Use cases: Remove customer details

Note:

Exceptions:

Pre-conditions:

Post-conditions:

Contract

Name: removeCustomerDetails()

Responsibilities: The employee removes the customer's details from the company's database.

Cross References: Use cases: Remove customer details

Note:

Exceptions:

Pre-conditions: The customer wishes that their details be removed from the company's database or the customer has been inactive for a very long time.

Post-conditions: The customer's details have successfully been removed from the company's database.

Contract

Name: requestHire()

Responsibilities: The customer wishes to hire a car.

Cross References: Use cases: Hire car

Note:

Exceptions:

Pre-conditions: The customer wishes to hire a car.

Post-conditions:

Contract

Name: selectCar(Make, Model)
Responsibilities: The customer chooses a specific car and this data is collected by the employee.
Cross References: Use cases: Hire Car
Note: Check the system to see whether that particular car is available for hire.
Exceptions: If a car will be available very soon for hire and the customer register's an interest, then the person responsible for adding new cars to the fleet has to inform the customer about when that car will be available for hire.

Pre-conditions: That particular car is currently available and serviced ready to be hired out.
Post-conditions:

Contract

Name: hireTerms(HireDate, ReturnDate)

Responsibilities: The customer and the employee agree hire terms.

Cross References: Use cases: Hire Car

Note: If an agreement cannot be reached on hire terms or a discount can be offered to the customer, the employee must consult the manager.

Exceptions:

Pre-conditions:

Post-conditions:

Contract

Name: checkCarAvailability()

Responsibilities: The employee checks the system to see whether the car the customer has chosen is available for hire.

Cross References: Use cases: Hire Car

Note:

Exceptions:

Pre-conditions:

Post-conditions:

Contract

Name: agreeHireTerms()

Responsibilities: The employee agrees to the hire terms.

Cross References: Use cases: Hire Car

Note:

Exceptions:

Pre-conditions:

Post-conditions: The hire terms have been agreed with customer, and the customer has read the terms and conditions and signed the contract.

Contract

Name: enterHireTerms()

Responsibilities: The employee enters the hire terms into the system.

Cross References: Use cases: Hire Car

Note:

Exceptions:

Pre-conditions: The hire terms have been agreed.

Post-conditions: The hire terms have successfully been entered into the system.

Contract

Name: finishSale()

Responsibilities: The employee updates the system so that particular car is no longer available for hire to someone else. The employee also logs the sale.

Cross References: Use cases: Hire Car

Note:

Exceptions:

Pre-conditions: The hire terms have been agreed and entered into the system.

Post-conditions:

Contract

Name: cashPay(Amount):Amount

Responsibilities: The customer pays the cash amount for the hire.

Cross References: Use cases: Hire Car

Note: The customer can only make a cash payment. The cash has to be counted.

Exceptions:

Pre-conditions: The cash payment is only made once the sale has been completed.

Post-conditions:

Contract

Name: printReceipt()

Responsibilities: The employee prints the receipt off the system.

Cross References: Use cases: Hire Car

Note:

Exceptions:

Pre-conditions:

Post-conditions:

Contract

Name: recieveReceipt()

Responsibilities: The employee gives the customer the printed receipt.

Cross References: Use cases: Hire Car

Note:

Exceptions:

Pre-conditions:

Post-conditions:

Contract

Name: requestCancelHire()

Responsibilities: The customer requests to cancel a hire.

Cross References: Use cases: Cancel Hire

Note:

Exceptions:

Pre-conditions:

Post-conditions:

Contract

Name: requestCancellationFee()

Responsibilities: The employee requests the cancellation fee from the customer.

Cross References: Use cases: Cancel Hire

Note:

Exceptions:

Pre-conditions:

Post-conditions:

Contract

Name: finishCancelHire()

Responsibilities: The employee logs on the system that the hire has been cancelled and the car hired out is now available again.

Cross References: Use cases: Cancel Hire

Note:

Exceptions:

Pre-conditions:

Post-conditions:

Contract

Name: enterMechanicDetails(Name, Address , Phone)

Responsibilities: The mechanic provides their details to the manager.

Cross References: Use cases: Add mechanic details

Note:

Exceptions:

Pre-conditions:

Post-conditions:

Contract

Name: verifyMechanicDrivingLicense()

Responsibilities: The manager verifies the mechanic’s driving license.

Cross References: Use cases: Add mechanic details

Note:

Exceptions:

Pre-conditions:

Post-conditions: The mechanic's driving license is valid.

Contract

Name: storeMechanicDetails()

Responsibilities: The manager stores the mechanic's details into the system.

Cross References: Use cases: Add mechanic details

Note:

Exceptions:

Pre-conditions: The manager only stores the mechanic's details after the mechanic's driving license has been verified.

Post-conditions: The mechanic's details have been successfully stored.

Contract

Name: mechanicLeaving()

Responsibilities: The mechanic is leaving the company or the mechanic has been fired/made redundant by the manager.

Cross References: Use cases: Remove mechanic details

Note:

Exceptions:

Pre-conditions:

Post-conditions:

Contract

Name: removeMechanicDetails()

Responsibilities: The manager removes the mechanic's details from the system.

Cross References: Use cases: Remove mechanic details

Note:

Exceptions:

Pre-conditions: The mechanic is leaving the company and their details are no longer required.

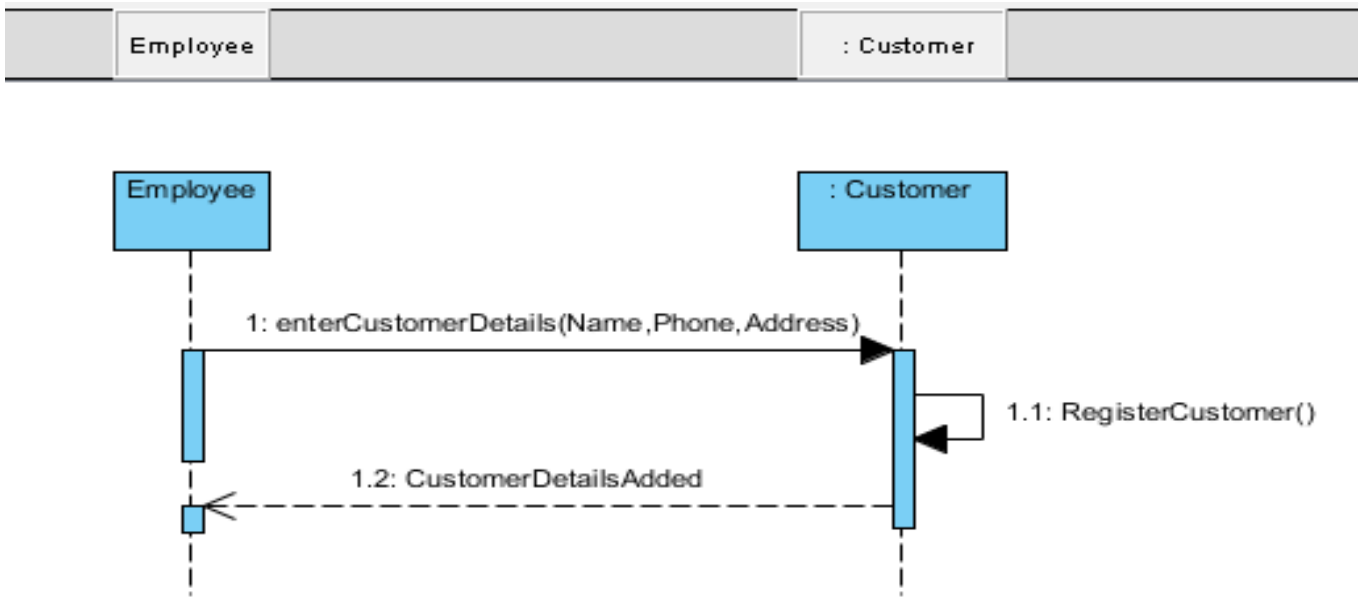
Post-conditions: The mechanic's details have been successfully removed.

批注 [ZL12]: Obviously, this team do not understand the notion of contracts

III. Use case design

Object Sequence Diagrams:

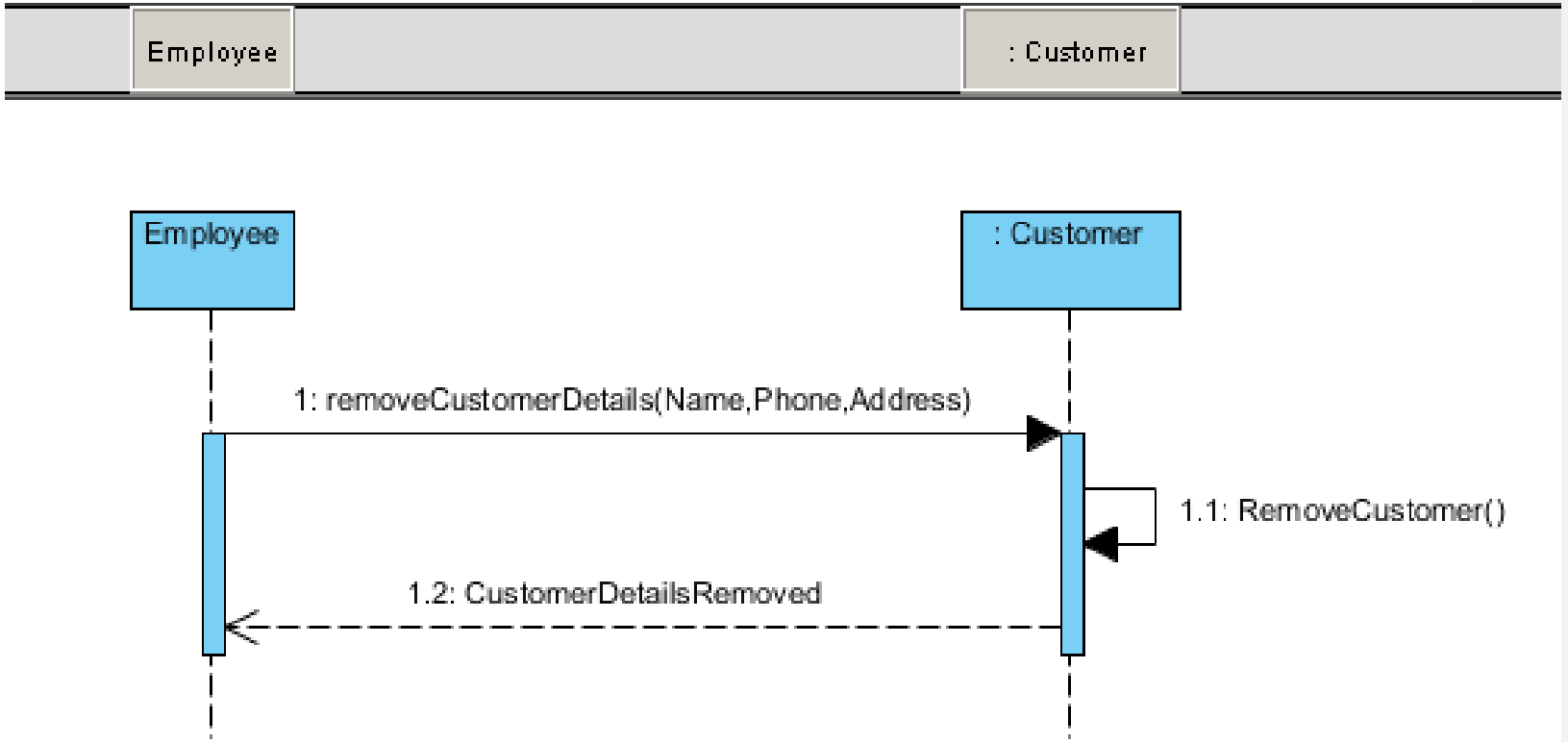
Adding a Customer:



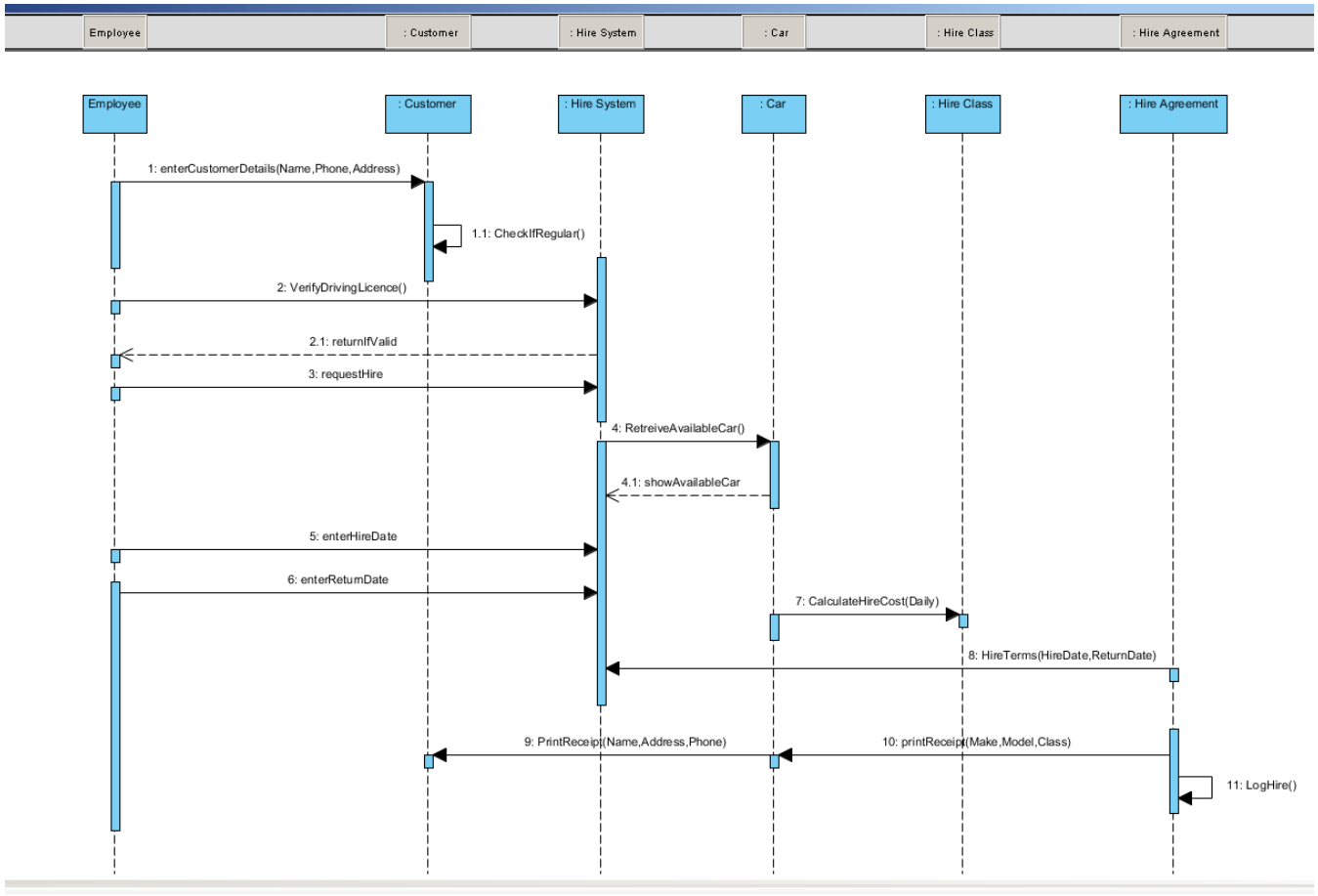
批注 [ZL13]: You called “Register customer” in the use case descriptions

批注 [ZL14]: This doe not represent a design

Removing a Customer:



Hiring a Car:



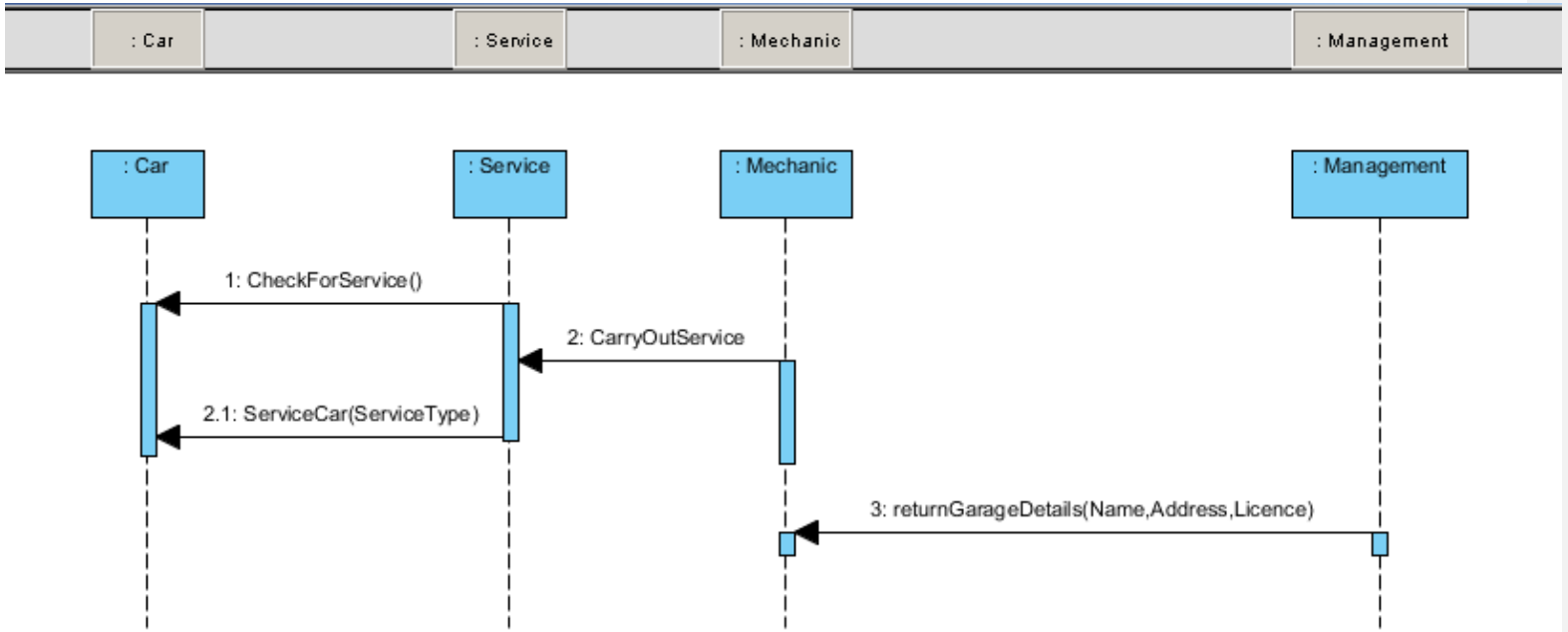
批注 [ZL15]: Why is Customer the controller of event 1?

Operation 2 is not designed

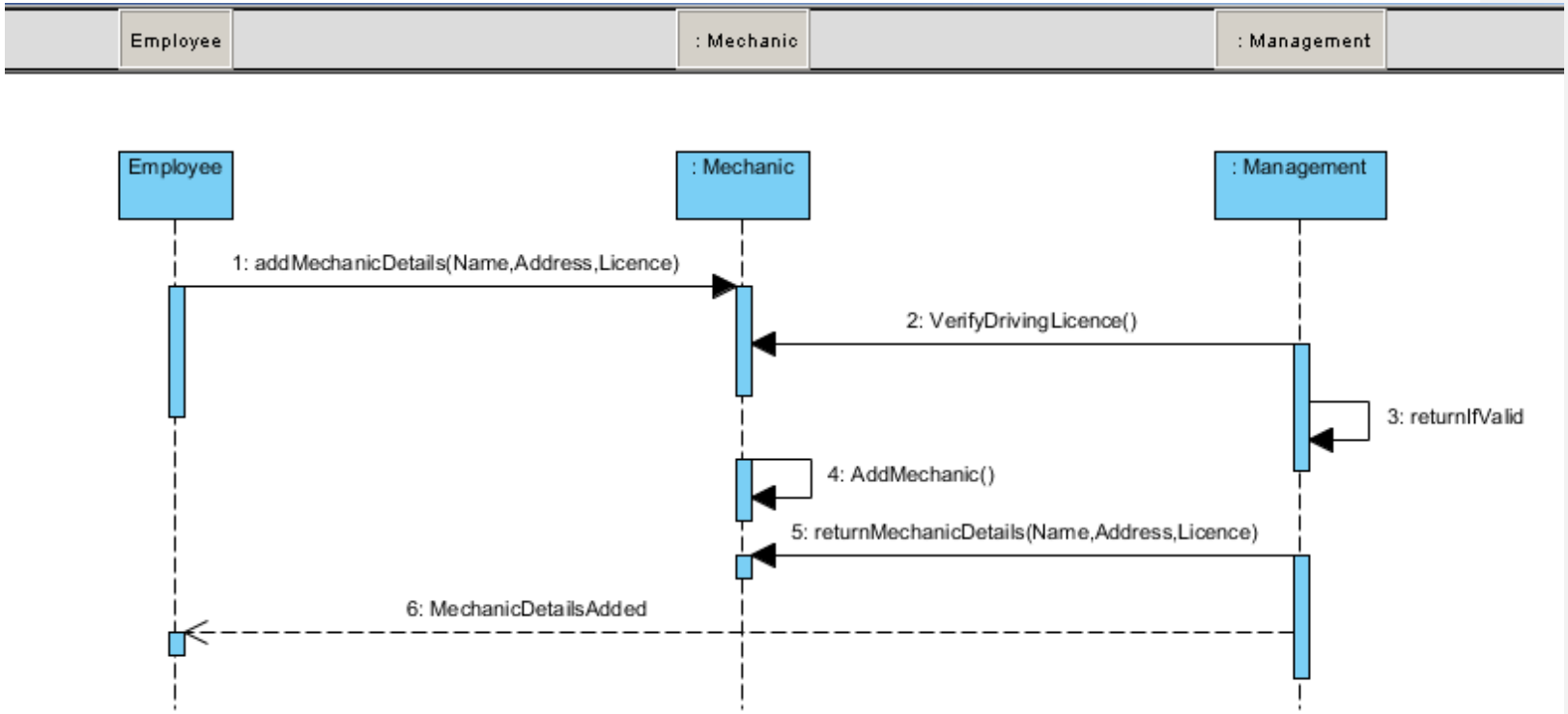
Requirehire needs to find an available car, the car class itsel cannot do this

…

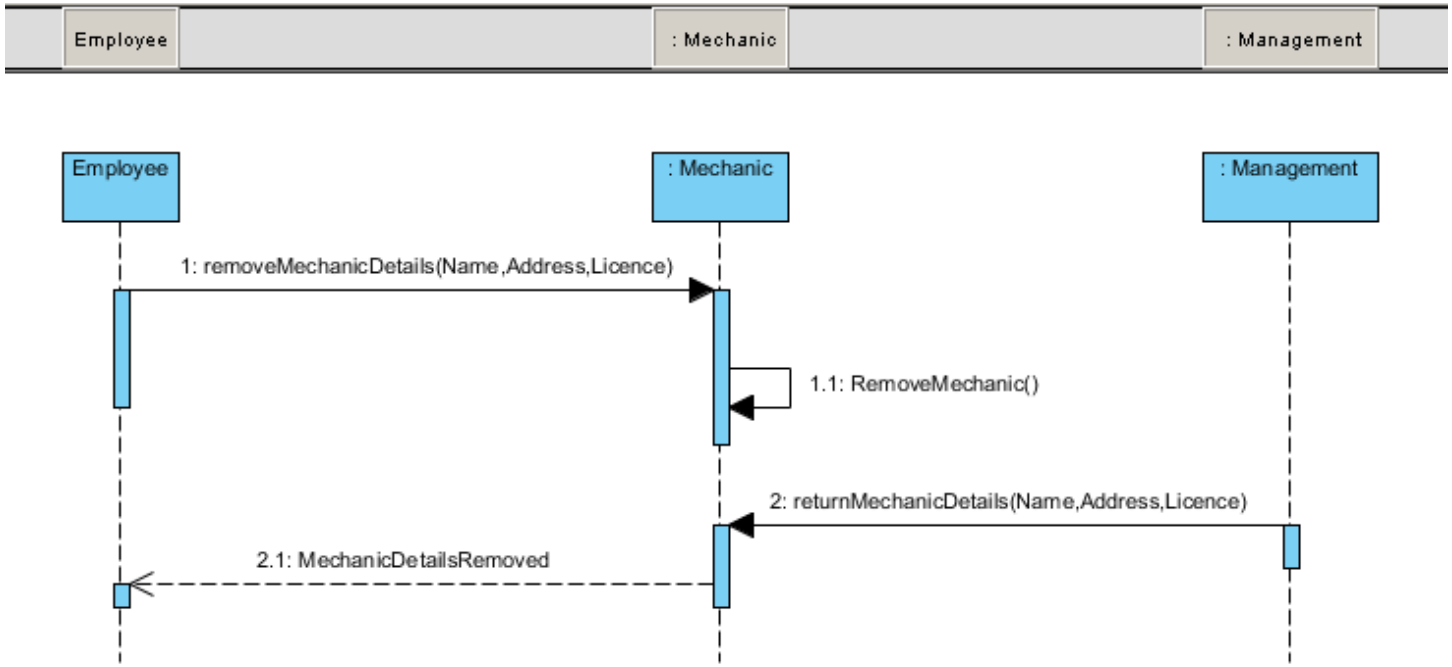
Servicing a Car:



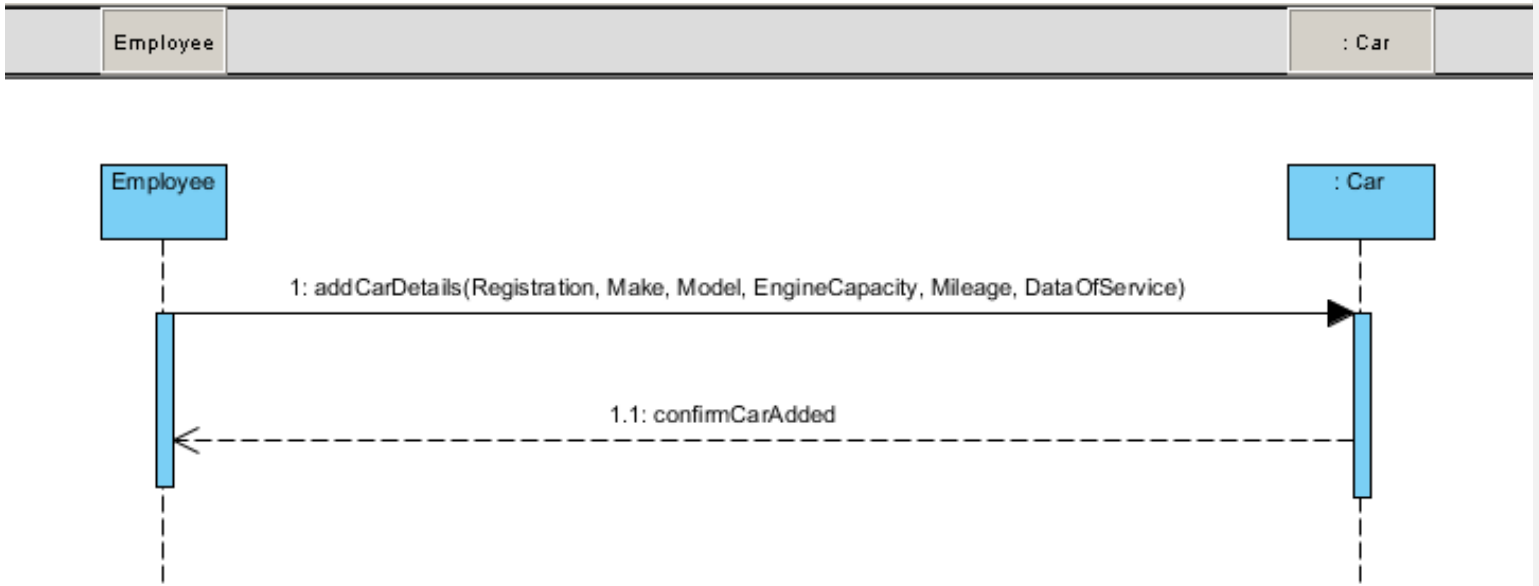
Adding a Mechanic:



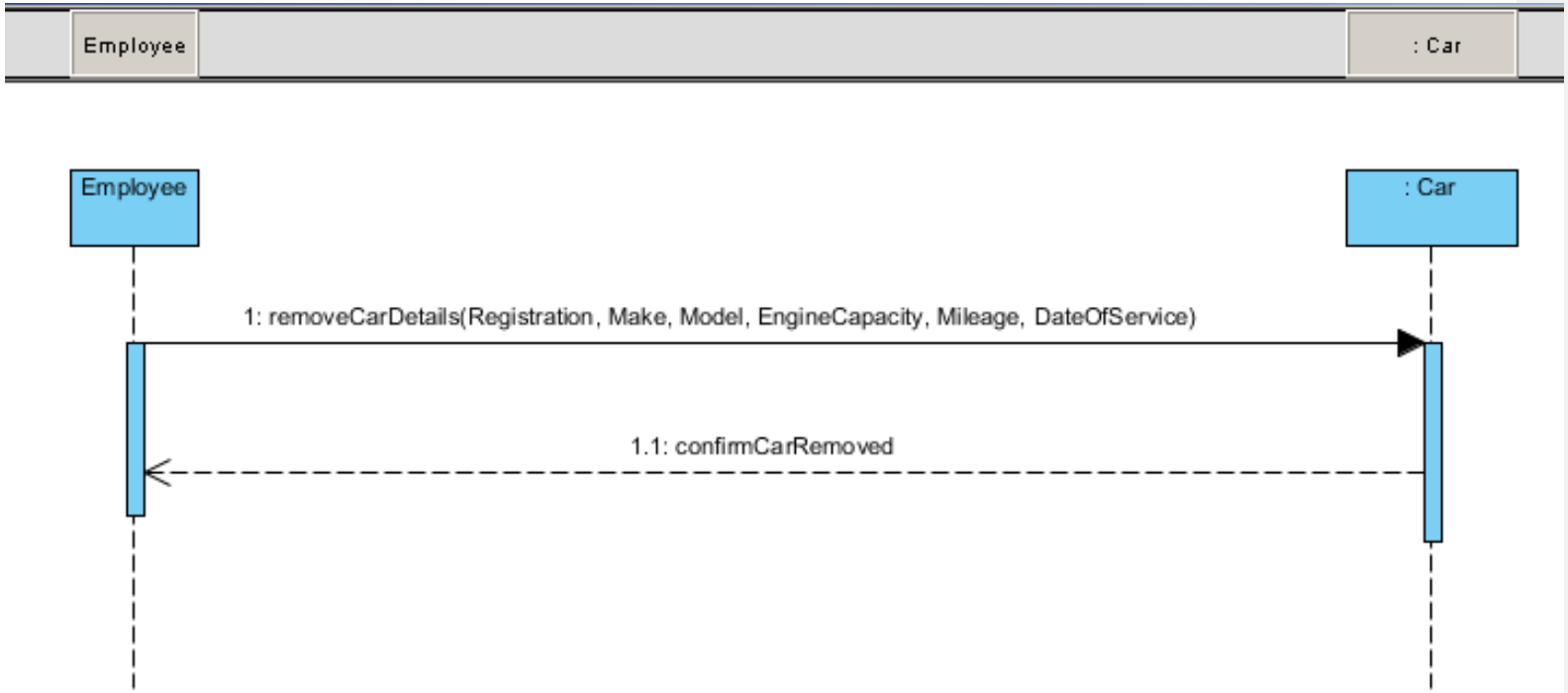
Removing a Mechanic:



Adding a Car:


Employee
: Car
Employee
: Car
1: addCarDetails(Registration, Make, Model, EngineCapacity, Mileage, DataOfService)
1.1: confirmCarAdded

Removing a Car:



带格式的：制表位： 15.56 字符，左对齐

In general, the design class diagram are poor

Patterns

Pattern Name: Expert (HireAgreement)

Solution: HireSystem.HireDate, HireSystem.ReturnDate, Customer.RegularCustomer, Customer.Non-RegularCustomer, Car.Make, Car.Model, Car.Class

Problem: The pattern will solve the problem of how to complete the sale with the creation of a receipt that will hold the hire and return dates of the car, the customer who is hiring the car, and the car information.

Pattern Name: Creator (HireSystem)

Solution: HireSystem has the initialising data (HireDate, ReturnDate) that will be passed to HireAgreement when it is created.

Problem: The pattern will initialise data that will be referred back to the HireAgreement when it is required.

Pattern Name: Low Coupling (Customer)

Solution: HireSystem, HireAgreement

Problem: The pattern is only relied on when a receipt is only needed based on the hiring of a car and to verify the licence to check authenticity.

Pattern Name: High Cohesion (HireAgreement)

Solution: HireAgreement.PrintReceipt

Problem: To keep the complexity manageable, the PrintReceipt method will only inherit attributes that are necessary such as hire and return dates of the car, the customer who is hiring the car, and the car information.

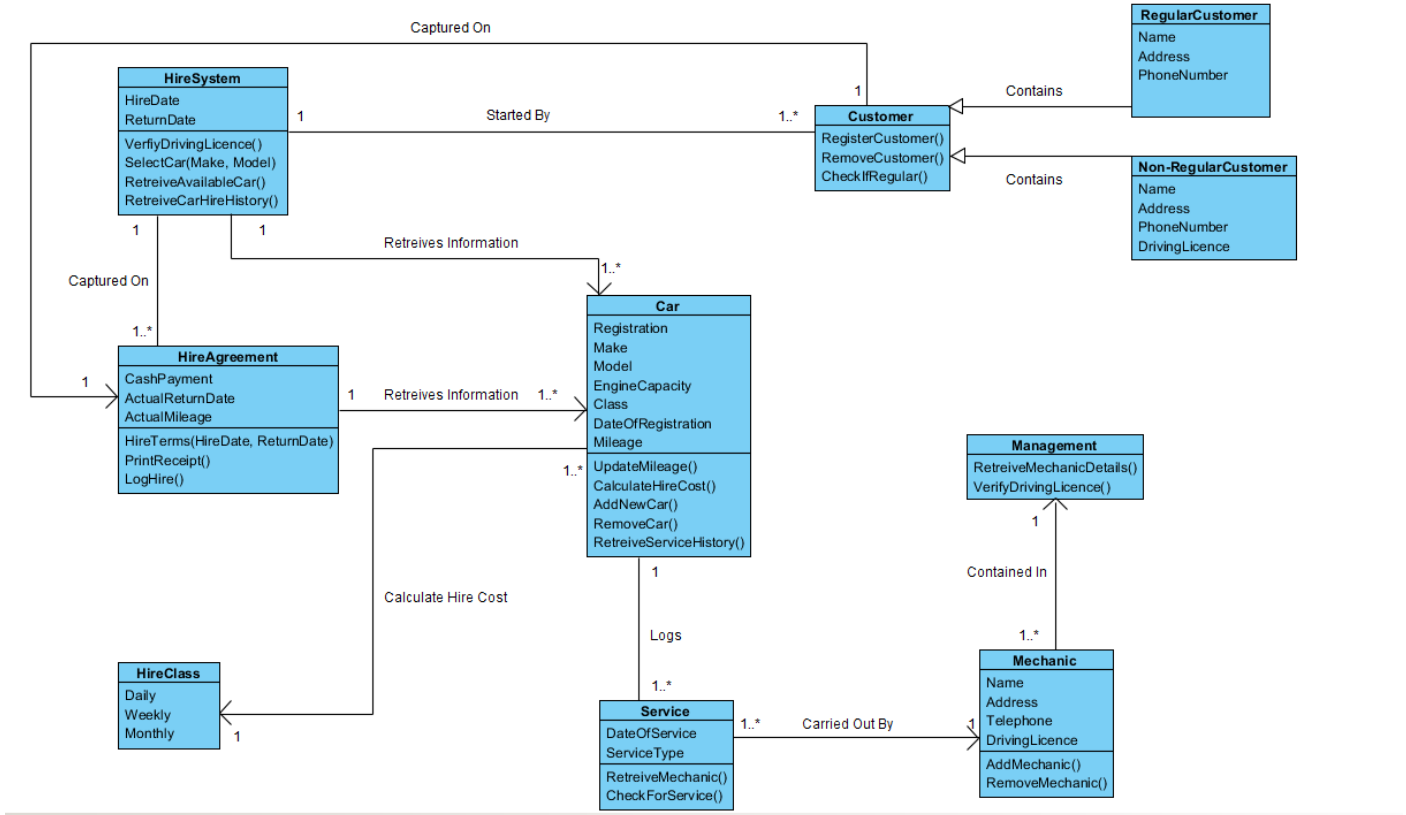
Pattern Name: Controller (Car)

Solution: Represents something in the real-world that is active that might be involved in the task, the car is the controller as it is heavily relied on by the other classes.

Problem: Employee inputs any external data

批注 [ZL16]: You are meant to use patterns to validate your object sequence diagrams

Design Class Diagram:



Consider the consistency with the conceptual class diagrams and the object sequence diagrams, this is a quite good design class diagram.

Project Management

Attendance:

		Meeting Date:	26th January	2nd February	9th February	16th February	23rd February	2nd March	9th March	16th March	23rd March	30th March	6th April	13th April	20th April
2	Name														
3	Gurpindervir Rai														
4	Zacharias Nur														
5	Kasim Mehmood														
6	Qasim Naveed Malik														
7	Shamem Miah														
8	Sarmad Rafiq														
9															
10															
11															
12		Key:		Attended											
13				Absent											
14															

The attendance by all team members was excellent. Everyone was very enthusiastic about the project.

Below is a brief discussion of what was discussed in each meeting:

- 26th January: Overview of the project- Discussion of what the project is about and how we should go about doing the project.

- 2nd February: It was agreed that everyone should read up on use cases. Everyone was assigned their individual use cases. Discussion of the theory behind the code. Detailed discussion of the problem description- key point raised: what is meant by a non-regular customer (time scale to define this).
- 9th February: It was agreed that use cases should be finished by the end of the week along with class diagrams.
- 16th February: The conceptual model was dicussed.
- 23rd February: All the use cases were completed and the class diagrams were also completed. A start on the report was made.
- 2nd March: The use cases had been updated after advice from the tutor and the use case diagrams had been completed.
- 9th March: It was agreed that everyone would have to sit down together to complete the use case sequence diagrams as everyone has their own judgement. Hence it was important for us to all get together to do them.
- 16th March: The use case sequence diagrams have been completed and we all discussed contracts which we completed together after the meeting.
- 23rd March: We assigned individual group members different object sequence diagrams.
- 30th March: The object sequence diagrams had been completed.
- 6th April: We worked together to do patterns.
- 13th April: The design class diagram was completed together as a group.
- 20th April: Finalising the report ready to submit.

Glossary:

Actor - initiates the use case.

Alternative courses of events - describes the patterns of interactions carried out under exceptional conditions.

Association - a relationship between two classes that specifies how instances of the classes can be linked together to work together.

Attribute - An attribute of a class is the abstraction of a single property of objects from the class.

Class - A class defines a set of objects which have common types of properties. It includes a description of a collection of objects, sharing structure, behavioural pattern, and attributes.

Conceptual model - A conceptual model illustrates meaningful concepts in the problem domain

Controller - responsible for handling external inputs

Creator - a class that has the responsibility to create an instance of another class

Expert - a class that has information to fulfil the responsibility

High cohesion - a class which has high functionality but low amount of work

High-level use case - A high-level use case describes a process very briefly with actors and abstract actions that the actors perform.
Instance - a single occurrence of something

Low coupling - a class which is not dependent on too many other classes

Object sequence diagram - models the interactions between objects inside the system (component). It decomposes a use case operation into interactions among objects, and defines methods of classes.
Patterns - general principles and idiomatic solutions that guide the creation of software.

Pre-conditions - conditions that the states are assumed to satisfy before the execution of the operation

Post-conditions - conditions that the states have to satisfy when the execution operation has finished.

Superclass - relationship between objects that is being inherited from

Typical course of events - describes the general pattern of interactions carried out.

Use Case - A use case identifies and describes the process that ``actors'' carry out a task in the application domain. It defines at the ``type'' level, that is it describes the pattern in which all the instances (executions) of the tasks performed by instances of the actors.

Use case sequence diagram - describes the interaction view and identifies methods required by actors.