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: T22

Team Leader: Ciaran Keogh

Team Members:
Matthew Hinton, Bilal Hasain, James Frith
Thomas Hulme, Abdul Lefsay

**Instruction to Students:**

The submission of the final report should contain this cover page with details of the team filled in above.

As part of the project management requirement, each team should have a weekly meeting. A weekly project diary should therefore be maintained to record the attendance of the meetings by team members, together with brief notes about the weekly project tasks allocations and how well individual team members meet the deadlines of their project tasks.

The final report should include the project diary in the appendix.

# CMP5215: Software Design UG2 Team Project

Produced By:
Ciaran Keogh- s13161658, Thomas Hulme, Matthew Hinton, Bilal Husain, James Frith, Abdul Lefsay

## Contents

I. The Initial Requirements Understanding ..... 3
System Functions ..... 3
Identifying the Essential Use Cases and Producing Expanded Use Cases ..... 7
Use Case Diagram ..... 23
Conceptual Class Diagram ..... 24
II. Functionality Analysis of System Operations ..... 28
System Sequence Diagrams ..... 28
System Operation Contracts ..... 33
III. Use Case Design ..... 45
Object Sequence Diagrams ..... 45
High cohesion: ..... 49
Design Class Diagram ..... 53
**Appendix** ..... 56
Meetings ..... 56
Group Meeting 1 – Introduction ..... 56
Group Meeting 2 – Introduction ..... 58
Group Meeting 3 – Progress update ..... 60
Group Meeting 4 – Progress update ..... 62
Group Meeting 5 – Progress update ..... 64
Group Meeting 6 – Conclusion ..... 66
Work Allocation ..... 67

# I. The Initial Requirements Understanding

The Bvis Car Hire Company have asked us to design a new, paperless car hire system. This system will record details of the company's cars and hire transactions. An analysis of the system requirements reveals that an object-oriented solution would be the most appropriate.

This solution would be appropriate because the concepts involved in a point of sale system can easily be represented as classes, as they are made up of objects which share similar structures, behavioural patterns and attributes. For example, when a car is hired, the same general process is repeated every time. This means it can be represented by a class CarHire, and every car hire will be an object of the CarHire class.

We can then use object-oriented design principles to realise use case operations through interactions among objects of the classes. This would be ideal for creating a paperless system for the company.

## System Functions

System Functions have been identified, outlining what the system needs to do in order to operate. These functions have been grouped based upon whether they will affect customers, staff members, mechanics, or involve the hire cars themselves. Each system function has been categorised as either *hidden*, *evident*, or *frill* in order prioritise them.

**Customer Based System Functions**

| Reference # | Function | Category |
|---|---|---|
| 1.1 | Register a customer into the system, creating them a customer account with the following details: Name, Telephone Number and Address. | Evident |
| 1.2 | System should be able to automatically allocate a new ID Number to a customer instance. | Hidden |
| 1.3 | Allow the customer to see their user details. | Evident |
| 1.4 | Allow the customer to see the available cars to hire. | Evident |
| 1.5 | Allow the customer to hire a particular car, using the following details: Registration Number (unique), Make, Model, Engine Capacity and Hire Class (1-6). | Evident |

| | | |
|---|---|---|
| | The dates of the beginning and end of the hire are recorded (the second is an estimate and will be updated when the vehicles is returned) as well as the number of miles at the start of the hire period. | |
| 1.6 | The customer must be able to see the daily cost of hiring a car, based on the hire class and length of hire time. | Evident |
| 1.7 | The system should be able to calculate the cost of hiring a particular class of car for a specified time period. | Hidden |
| 1.8 | A receipt should be printed with time of hire, time of return and cost of hire. | Evident |
| 1.9 | System should take note of how long it has been since the customer has registered with the system. | Hidden |
| 1.10 | The customer must be able to remove his/herself from the system. | Evident |
| 1.11 | When a car has been returned, the details of actual return date and mileage accumulated should be recorded. | Evident |
| 1.12 | The accumulated mileage should be added to the total mileage in the car's details. | Hidden |

**Staff Based System Functions**

| Reference # | Function | Category |
|---|---|---|
| 2.1 | Staff Member should be able to log into the system using their user info (Name/ID and password). | Evident |
| 2.2 | Staff Member should be able to retrieve information about a customer from the system. | Evident |
| 2.3 | Staff Member should be able to retrieve information about a certain car from the system. | Evident |
| 2.4 | Staff Member should be able to remove a customer from the system. | Evident |
| 2.5 | Staff Member should be able to remove a car from the system. | Evident |
| 2.6 | Staff Member should be able to remove a mechanic from the system. | Evident |
| 2.7 | Staff Member should be able to add a car to the system. | Evident |

**Car Based System Functions**

| Reference # | Function | Category |
|---|---|---|
| 3.1 | A record of the cars details should be stored. This should include: Registration Number (unique), Make, Model, Engine Capacity and Hire Class (1-6). | Evident |
| 3.2 | Information of the car should be accessible. This should also include Car Hire History and Car Service History. | Evident |
| 3.3 | When the car's mileage reaches 6000 miles, an automated notice should be sent to indicate that this car is in need of a "minor service" to its assigned mechanic. | Evident |
| 3.4 | When the car's mileage reaches 12000 miles, an automated notice should be sent to indicate that this car is in need of a "major service" to its assigned mechanic. | Evident |

| | | |
|---|---|---|
| 3.5 | Car should keep a record of and update amount of time the car has spent in service within the vehicle fleet. | Frill |

**Mechanic Based System Functions**

| **Reference #** | **Function** | **Category** |
|---|---|---|
| 4.1 | Mechanic should be able to add/register themselves into the system, using the following details: Name, Address, Home Telephone and Drivers License. | Evident |
| 4.2 | A Mechanic ID is automatically assigned to the Mechanic when they are registered. | Hidden |
| 4.3 | Mechanic should be able to view a list of all the cars that he is currently responsible for. | Evident |
| 4.4 | Mechanic should be able to write a service report that updates the service history of a car that he has serviced. | Evident |
| 4.5 | Mechanic should be able to remove a car from the car fleet. | Evident |

## Identifying the Essential Use Cases and Producing Expanded Use Cases

Using the system functions identified, the following essential use cases were created. The use cases lay out the process taken in order to perform certain system functions. Both High-Level and Expanded Use Cases have been created, adding depth and detail to each of them.

### Use Cases:

#### Registering a new Customer

| | |
|---|---|
| **Use Case:** | Registering a new Customer |
| **Actors:** | Customer (Initiator)<br>Staff Member |
| **Purpose:** | To **register** a new **customer** into the system. |
| **Overview:** | The **Customer** arrives at the **registration desk** and asks to register. The **Customer** supplies the required **details** to the **Staff Member**. Once supplied, the **Staff Member** *records* and *stores* the information in the system. |

**Typical Course of Events**

| **Actor Action** | **System Response** |
|---|---|
| 1. The **Customer** arrives at the **registration desk** and asks to register. | |
| 2. The **Staff Member** asks for the **Customers *name*, *telephone number, address*** and ***driving license number***. | |
| 3. The **Customer** supplies the required details to the **Staff Member**. | |
| 4. The **Staff Member** inputs the information into the system. | 5. The input information is saved by the system. |

批注 **[ZL1]:** Need to log

**Recording the Hire of a Particular Car**

| | |
|---|---|
| **Use Case:** | **Recording** the **Hire of a Car** |
| **Actors:** | Customer (Initiator)<br>Staff Member |
| **Purpose:** | To **record** the details of a car hire made by a **customer**. |
| **Overview:** | The **Registered Customer** is at the desk and wants to hire a car. The **Staff Member** records the details of the hire i.e. identification number of hired vehicle, date/time of hire, estimated return date of hire and the number of miles on the clock at the start of the hire period. The **Staff Member** issues an *invoice* of hire for the **Customer**. |

**Typical Course of Events**

| **Actor Action** | **System Response** |
|---|---|
| 1. The **Registered Customer** is at the desk and wants to hire a car. | |
| 2. The **Staff Member** asks the **Customer** for the date/time they wish to hire the car, the type of car required, and the estimated return date of the car. | |
| 3. The **Staff Member** checks the availability of the chosen car in the specified time period. | 4. Checks the database to see if chosen car and time period are available. |
| 5. **Staff Member** confirms **availability** of chosen vehicle and informs the customer of the daily hire rate. | 6. Generates **daily hire rate** for specified hire length and car type. |
| 7. **Customer** is happy with the hire and | |

confirms with **Staff Member**.

8. **Staff Member** confirms the booking and inputs required information into the system.

9. **Records** car hire in the database and generates invoice.

**Alternative Courses**

- Line 1: A **Customer** may not be in the database, indicate error. **Customer** must undertake registration.
- Line 5: **Staff Member** may inform **Customer** that chosen vehicle is not available. **Customer** can either return to *Line 2* and select another car or time period, or the transaction is cancelled.

**Recording the Return of a Particular Car**

| | |
|---|---|
| **Use Case:** | **Recording** the **Return of a car** |
| **Actors:** | Customer (Initiator)<br>Staff Member |
| **Purpose:** | To **record** the details of a car return made by a **customer** and calculate what the customer is required to **pay**. |
| **Overview:** | The **Customer** arrives at the desk and wants to **return a car**. The **Staff Member** updates the details of the specific hire with the ***actual*** *return date* and the *cars mileage* upon return. The cost of hire is calculated and the **Staff Member** informs the **Customer** of the total cost. The **Customer** makes a **payment** by **cash only** and gets a **receipt** for his purchase. |

**Typical Course of Events**

| **Actor Action** | **System Response** |
|---|---|
| 1. A **Registered Customer** arrives at the desk and wants to return a car. | |
| 2. The **Staff Member** asks for the *booking details* of the hired car. | |
| 3. The **Staff Member** checks for a record of the car hire in the **database**. | 4. Checks the database for the **record** of the car hire based upon the **user input**. |
| 5. **Staff Member** updates the hire record with the actual return date and the mileage upon the cars return. | 6. **Calculates** the cost of hire based upon the *daily hire rate* for the specific type of car and the *length of the hire*. |
| 7. **Staff Member** informs the **Customer** | |

of the **cost** of the hire.

8. **Customer** makes payment by **cash** (possibly greater than the sale total).

9. The **Staff Member** records the cash amount received.

10. Show the **balance** due back to the customer.
Generate a receipt.

11. The **Staff Member** updates and logs the hire record.

12. Specified car hire is updated, completed and stored within the database for future reference.

13. **Customer** receives **receipt** (and **owed cash**) and then leaves.

**Alternative Courses**

- Line 4: No record of car hire found, indicate error.
- Line 8: Customer does not have enough cash, cancel return of hire car.

**Recording a Service for a Car**

| | |
|---|---|
| **Use Case:** | **Recording** a Car Service |
| **Actors:** | Mechanic (Initiator) |
| **Purpose:** | To **record** the service of a car that has been highlighted as *exceeding its service interval*. |
| **Overview:** | The **Mechanic** logs into the system and is shown the list of cars assigned to them, the car(s) that require a **service** are highlighted. The **Mechanic** then performs the required service on the car. The **Mechanic** *updates* the car's service record to signify the service has been completed and logs off the system. |

**Typical Course of Events**

| **Actor Action** | **System Response** |
|---|---|
| 1. The **Mechanic** logs into the system and accesses the list of cars that are assigned to them. | 2. Compiles a list of cars that are assigned to the **ID** of the mechanic logged in. **Highlight** the cars either need a minor or major service. |
| 3. The **Mechanic** checks to see which car requires their attention first. | |
| 4. The **Mechanic** then performs the necessary service on the car. | |
| 5. Upon completion of the service, the **Mechanic** updates the cars service record in the system with the *time/date of the service*, *the type of service performed*, *car mileage at service* and *the name of the mechanic responsible*. | 6. Updates the Car Service Record to signify the changes input by the **Mechanic**. |

7. The **Mechanic** then logs off the system.

**Alternative Courses**

- Line 3: No services required on any of the other cars, mechanic can skip to Line 7 and logoff the system.
- Line 7: More cars may require a service, mechanic can return to Line 3 and select another car for service.

**Removing a Customer**

| | |
|---|---|
| **Use Case:** | **Removing** a Customer |
| **Actors:** | Staff Member (Initiator) |
| **Purpose:** | To **remove** the instance of a customer from the system. |
| **Overview:** | The **Customer** arrives at the registration desk and asks to be removed from the system. The **Staff Member** logs into the system and opens the *Customers file*. The **Staff Member** then identifies the customer(s) that they want to remove from the system and uses the system interface to perform the change. The **Staff Member** then closes the *Customers file*. |

**Typical Course of Events**

| **Actor Action** | **System Response** |
|---|---|
| 1. The **Customer** arrives at the registration desk and asks to be removed from the system. | |
| 2. The **Staff Member** logs into the system. | |
| 3. The **Staff Member** opens up the *Customers File*. | 4. Display a complete list of customers within the system. |
| 5. **Staff Member** identifies the customer(s) they want to remove from the system and *removes them through the system interface*. | 6. Removes the customer information and the instance of the customer from the system and updates the database to accommodate the change. |
| 7. The **Staff Member** closes the *Customers File*. | |

**Adding a New Car to the Fleet**

| | |
|---|---|
| **Use Case:** | **Adding** a New **Car** to the Fleet |
| **Actors:** | Staff Member (Initiator) |
| **Purpose:** | To **add** a new hire **car** into the system. |
| **Overview:** | The **Staff Member** logs into the system and opens the *Fleet Cars* file. The **Staff Member** adds the details of the new car(s) into the system using the system interface and saves the file. The **Staff Member** then closes the *Fleet Cars* file. |

**Typical Course of Events**

| **Actor Action** | **System Response** |
|---|---|
| 1. The **Staff Member** logs into the system. | |
| 2. The **Staff Member** opens up the *Fleet Cars* file. | 3. Displays a complete list of Hire Cars in the system. |
| 4. The **Staff Member** adds the details of the new car(s) into the system, i.e. *registration number*, *car make*, *car model*, *engine capacity*, *hire class* and *date of registration*. | 5. Updates the *Fleet Cars* file to save and display changes. |
| 6. **Staff Member** closes the *Fleet Cars* file. | |

**Removing a Car that is no longer part of the Fleet**

| | |
|---|---|
| **Use Case:** | **Removing** a **Car** that is no longer part of the Fleet |
| **Actors:** | Staff Member (Initiator) |
| **Purpose:** | To **remove** a car that is no longer part of the hire fleet from the system. |
| **Overview:** | The **Staff Member** logs into the system and opens the *Fleet Cars* file. The **Staff Member** then identifies the car(s) that they want to remove from the fleet and uses the system interface to perform the change. The **Staff Member** then closes the *Fleet Cars* file. |

**Typical Course of Events**

| **Actor Action** | **System Response** |
|---|---|
| 1. The **Staff Member** logs into the system. | |
| 2. The **Staff Member** opens up the *Fleet Cars* file. | 3. Displays a complete list of Hire Cars in the system. |
| 4. **Staff Member** identifies the car(s) they want to remove from the system and *removes them through the system interface*. | 5. Removes the car information and the instance of the car from the system and updates the database to accommodate the change. |
| 6. The **Staff Member** closes the *Fleet Cars* file. | |

**Adding a New Mechanic**

| | |
|---|---|
| **Use Case:** | **Adding** a New **Mechanic** |
| **Actors:** | New Mechanic (Initiator)<br>Registration Officer |
| **Purpose:** | To input the information of a newly recruited mechanic into the system. |
| **Overview:** | The **New Mechanic** arrives at the *Registration Desk* and asks to register. The **Mechanic** provides the necessary details to the **Registration Officer** who then inputs them into the system. The information is saved into the system and an ID card is printed for the **Mechanic**. |

**Typical Course of Events**

| **Actor Action** | **System Response** |
|---|---|
| 1. The **New Mechanic** arrives at the *Registration Desk* and asks to register. | |
| 2. The **Registration Officer** asks the **Mechanic** for their *name*, *date of birth*, *address*, *home telephone number* and a *valid drivers license*. | |
| 3. The **Mechanic** provides the required details to the **Registration Officer**. | |
| 4. The **Registration Officer** inputs the provided information into the system as a new mechanic entry. Along with additional info i.e. *Unique ID No.*, *Job Title* and *Workstation*. | 5. Creates a new instance of the new mechanic entry with the information provided. The information is saved into the system. |

| | |
|---|---|
| 6. With the information in the system, the **Registration Officer** prints a new ID card for the **Mechanic**. | 7. Print an ID card with the details of the new mechanic on it. |

**Alternative Courses**

- Line 2: Mechanic does not have a valid drivers license, cancel Mechanic Registration.

**Removing a Mechanic who has left the Company**

| | |
|---|---|
| **Use Case:** | **Removing** a **Mechanic** who has left the **Company**. |
| **Actors:** | Staff Member (Initiator) |
| **Purpose:** | To **remove** a mechanic who has left the company from the system. |
| **Overview:** | The **Staff Member** logs into the system and accesses the *Mechanics* file. The **Staff Member** then identifies the mechanic(s) that they want to remove from the system (via. the *Mechanic Name or ID*) and uses the system interface to perform the change. The **Staff Member** then closes the *Mechanics* file. |

**Typical Course of Events**

| **Actor Action** | **System Response** |
|---|---|
| 1.The **Staff Member** logs into the system. | |
| 2. The **Staff Member** opens up the *Mechanics* file. | 3. Displays a complete list of Mechanics in the system. |
| 4. **Staff Member** identifies the mechanic(s) they want to remove from the system and *removes them through the system interface*. | 5. Removes the mechanic information and the instance of the mechanic from the system and updates the database to accommodate the change. |
| 6. The **Staff Member** closes the *Mechanics* file. | |

**Recovering the Information of a Hire Car**

| | |
|---|---|
| **Use Case:** | **Recovering** the Information of a **Hire Car** |
| **Actors:** | Staff Member (Initiator) |
| **Purpose:** | To return the information of a specified Hire Car in the system. |
| **Overview:** | The **Staff Member** logs into the system and accesses the *Fleet Cars* file. The **Staff Member** selects a specific car by inputting it's *registration number*. The System then displays the relevant details about the car. Upon reviewing the information the **Staff Member** closes the *Fleet Cars* file. |

**Typical Course of Events**

| **Actor Action** | **System Response** |
|---|---|
| 1.The **Staff Member** logs into the system and accesses the *Fleet Cars* file. | 2. Displays a list of all hire cars in the fleet. |
| 3. The **Staff Member** selects a specific car by either looking through the file *or* entering it's *registration number*. | 4. The System checks for the input registration number and displays the information about the selected hire car i.e. *Registration Number*, *car make*, *car model*, *engine capacity*, *hire class* and *date of registration*. |
| 5. The **Staff Member** then views the information and upon completion, closes the *Fleet Cars* file. | |

**Alternative Courses**

- Line 4: Invalid registration number input, Staff Member can try again by inputting another value.

**Recovering a Car's Hire History**

| | |
|---|---|
| **Use Case:** | Recovering a Car's Hire History |
| **Actors:** | Staff Member (Initiator) |
| **Purpose:** | To return the Hire History of a specified Hire Car in the system. |
| **Overview:** | The **Staff Member** logs into the system and accesses the *Car Hire Record* file. The **Staff Member** selects a specific car by inputting it's *registration number*. The System then displays the hire history of the specified car. Upon reviewing the information the **Staff Member** closes the *Car Hire Record* file. |

**Typical Course of Events**

| **Actor Action** | **System Response** |
|---|---|
| 1.The **Staff Member** logs into the system and accesses the *Car Hire Record* file. | 2. Displays a list of all hire cars in the fleet. |
| 3. The **Staff Member** selects a specific car by either looking through the file *or* entering it's *registration number*. | 4. The System checks for the input registration number and displays the selected cars *hire status/history*. |
| 5. The **Staff Member** then views the information and upon completion, closes the *Car Hire Record* file. | |

**Alternative Courses**

- Line 4: Invalid registration number input, Staff Member can try again by inputting another value.
- Line 4: No hire history available for selected car, Staff Member can search for another car instead or close the file.

**Recovering a Car's Service History**

| | |
|---|---|
| **Use Case:** | Recovering a Car's Service History |

| | |
|---|---|
| **Actors:** | Staff Member (Initiator) |
| **Purpose:** | To return the Service History of a specified Hire Car in the system |
| **Overview:** | The **Staff Member** logs into the system and accesses the *Car Service Record* file. The **Staff Member** selects a specific car by inputting it's *registration number*. The System then displays the service history of the specified car. Upon reviewing the information the **Staff Member** closes the *Car Service Record* file. |

**Typical Course of Events**

| **Actor Action** | **System Response** |
|---|---|
| 1.The **Staff Member** logs into the system and accesses the *Car Service Record* file. | 2. Displays a list of all hire cars in the fleet. |
| 3. The **Staff Member** selects a specific car by either looking through the file *or* entering it's *registration number*. | 4. The System checks for the input registration number and displays the selected cars *service status/history*. |
| 5. The **Staff Member** then views the information and upon completion, closes the *Car Service Record* file. | |

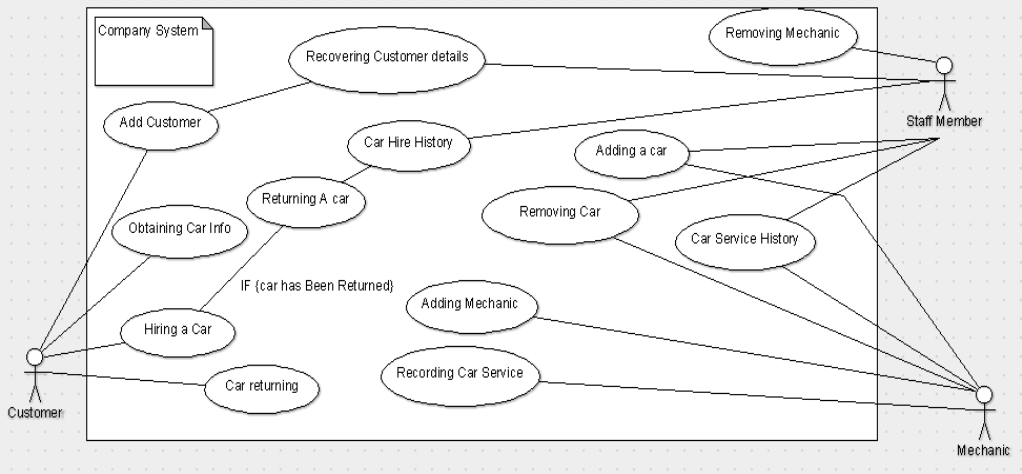
**Alternative Courses**

- Line 4: Invalid registration number input, Staff Member can try again by inputting another value.
- Line 4: No service history available for selected car, Staff Member can search for another car instead or close the file.

## Use Case Diagram

A use case diagram has been produced, illustrating the use cases that make up the company system. This shows the relationships between the actors and all the use cases, as well as the relationships between use cases.

批注 [ZL2]: Diagram does not seem to be produced by the tool

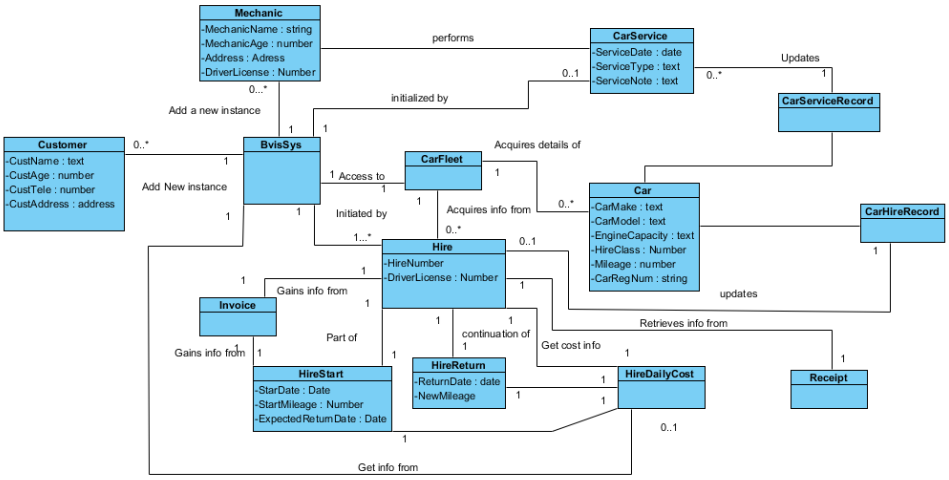



The following Use Case Diagram shows:

- The Customer actor is related to a number of use cases; such as Add Customer, Obtaining Car Info, Hiring a Car and Returning a Car.
- The Staff Member actor is also related to a number of use cases. These being Removing Mechanic, Recovering Customer details, Adding Car, Removing Car, Car Sevice History and Car Hire History.
- The Mechanic shares some of the same use cases as staff but has its own unique ones. These unique ones are Adding a Mechanic and Recording Car Service.
- This diagram will help identify the classes that will be used in the system.

## Conceptual Class Diagram

A conceptual class diagram has been produced, this is based upon all of the concepts and classes that have been identified.




### Identification of Concepts

批注 [ZL3]: There are some redundant classes in the class diagram

- BvisSys: The main electronic system that receives the main inputs from the UI interface, and manages the events between the system and the UI.
  Attribute: None.

- Customer: Concept that represents the details of the customer Actor.
  Attributes: CustName:text, CustAge:Number, CustTele: Number, CustAddress: address

- Mechanic: Concept that represents the deTails of a mechanic.
  Attributes: mechanic attributes: MechanicName, MechanicAge, address, DiversLicense.

- CarService: the concept a car service report that a mechanic makes when a car reaches a certain amount of mileage and requires a service.
  Attributes:ServiceDate, Servicetype, ServiceNote

- CarFleet:Concept of a the record and organising of the complete fleet of car currently owned by the company.
  Attributes:None.

- Car: Concept of an instance of a car being represented in the system.
  Attibutes: CarMake, CarModel, EngineCapacity, Mileage, CarRegNum,
- Hire: the concept a Hiring a car , comprised two main stages, the start hire where the customer chooses which car to hire, and the return of the car at the end of the hire.
  Attributes: HireNumber, DiverLicense

- HireStart:first stage of a hire: responsible for keeping record of the start date of the hire, gathering the details of the car that is being used for the hire, and the initial mileage.
  Attributes:StartDate, StartMileage, ExpendedReturnDate

- HireReturn: Second stage of a hire: responsible for recording the return date and update mileage.
  Attributes:ReturnDate, NewMileage.

- HireDailyCost: the cost generated by the difference in the number of days between the start hire date and the end of the hire.
  Attribute: none

- Invoice: output from the Hire start that gives the details of the hire from the start.
  Attribute: none

- Receipt: final output of the hire task, and proof of purchase.
  Attribute: none

**Glossary Terms For Conceptual Model**

| **Term** | **Category** | **Meaning/Comments** |
|---|---|---|
| BvisSys | Type(Class) | The main interface point to contact other concepts and use cases. |
| Customer | Type(Class) | Storing customer details and information. |
| Customer.CustomerName:string | Attribute | Name of the customer. |
| Customer.CustomerAge:Number | Attribute | The age of the customer. |
| Customer.CustomerTele:Number | Attribute | Telephone number for the customer. |
| Customer.Address:address | Attribute | The house number, street name and postcode of the customer. |
| Mechanic | Type(Class) | Storing the details of a new mechanic. |
| Mechanic.MechanicName:string | Attribute | The name of the mechanic. |

| | | |
|---|---|---|
| Mechanic.MechanicAge:Number | Attribute | The age of the mechanic. |
| Mechanic.Address:address | Attribute | The address of the mechanic. |
| Car | Type(Class) | Creates an instance of a new car and stores the information. |
| Car.CarRegNum:string | Attribute | Car registration number. |
| Car.CarMake:string | Attribute | Car Make, i.e. Company that makes the car. |
| Car.CarModel:string | Attribute | Model name of the car. |
| Car.EngineCapacity:string | Attribute | The engine capacity of the car. |
| Car.HireClass:number | Attribute | The hire class number of a car. |
| Car.Mileage:number | Attribute | The current mileage value of a car. |
| CarService | Type(Class) | Used to create an instance of a service report made by a Mechanic. |
| CarService.ServiceDate:date | Attribute | Storing the date of the service in question. |
| CarService.ServiceType:String | Attribute | The type of service being made (Minor/Major). |
| CarServiceRecord | Type(Class) | Stores every new instance of a car service and used to call upon a list of new instances. |
| CarHireRecord | Type(Class) | Stores every new instance of a car hire for a car. |
| Hire | Type(Class) | Creates an instance of a hire process. |
| Hire:DiverLicense.DiverLicense | Attribute | Storing the Divers License. |

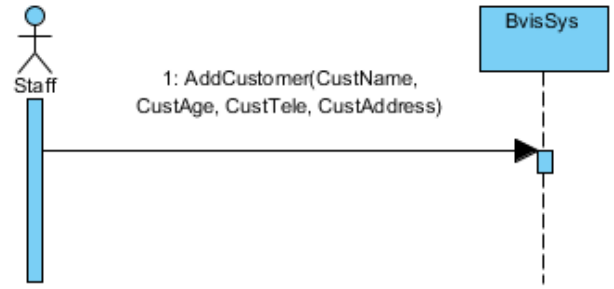
| | | |
|---|---|---|
| HireStart | Type(Class) | Creates an instance of starting process for a hire class. |
| HireStart.HireDate:Date | Attribute | Stores the start date of a Hire process. |
| HireStart.OrginalMileage:number | Attribute | Stores the original mileage record in hire process. |
| HireStart.Date.ExpReturnDate | Attribute | Stores the date of the expected return date of a hire. |
| HireReturn | Type(Class) | Contains the information of the return details of a hire process. |
| HireReturn.ReturnHireDate:Date | Attribute | The date of the car being returned. |
| HireReturn.Mileage.Number | Attribute | The new mileage of the return car in a hire process. |
| HireCost | Type(Class) | Calculates the cost of a hire by using the difference between of days between the starting hire date and the ending hire date. |
| Reciept | Type(Class) | Using the key information from the hire, this produces a printable receipt. |
| HireDailyCost | Type(Class) | Calculate the cost of a hire car based on user input, or automatically calculated based on the time difference between a hire start date and the return date. |
| Invoice | Type(Class) | Creates an invoice of the start of a hire. |

## II. Functionality Analysis of System Operations

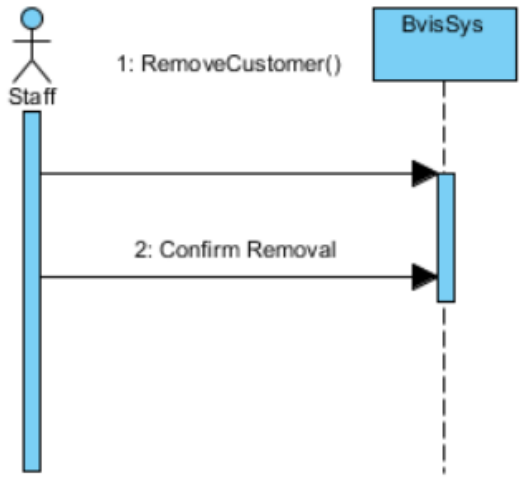
### System Sequence Diagrams

System sequence diagrams have been created in order to display actors interactions with the system. Each of the diagrams is based upon a use case identified previously.

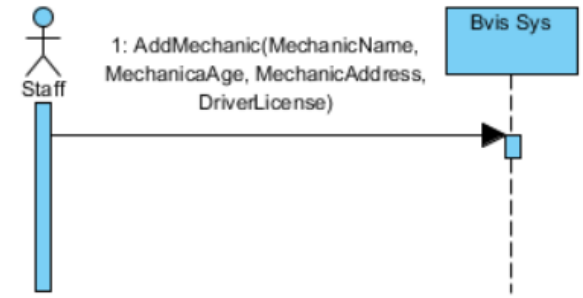
**Registering a New Customer**



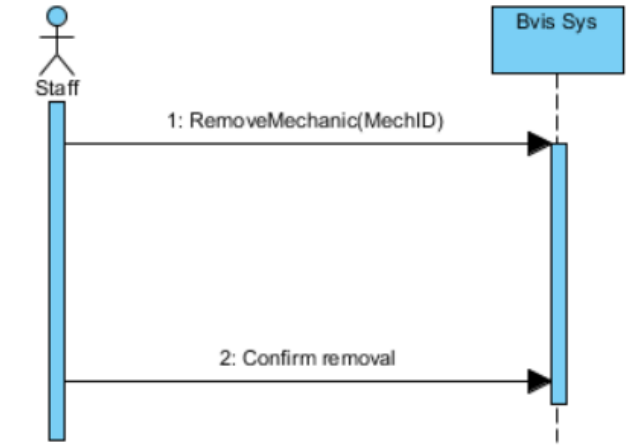

**Removing a Customer**



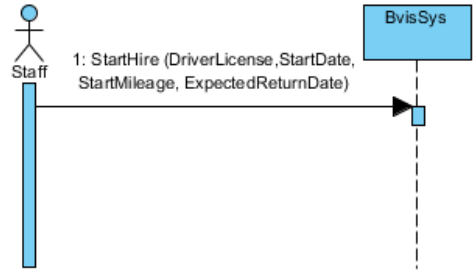
**Adding Mechanic**



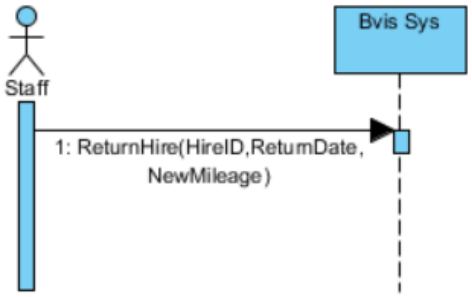

**Remove Mechanic**



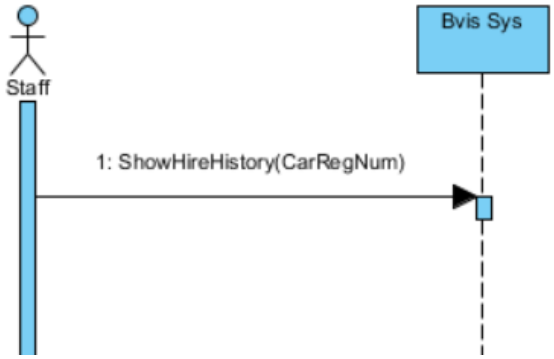

**Hiring a Car**



**Hire Car Return**

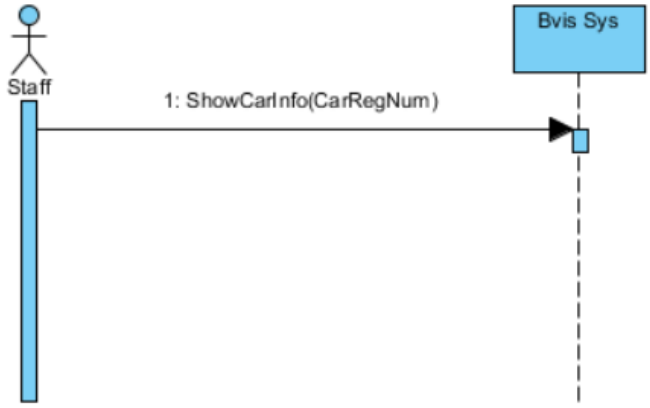



**Recovering Car's Hire History**



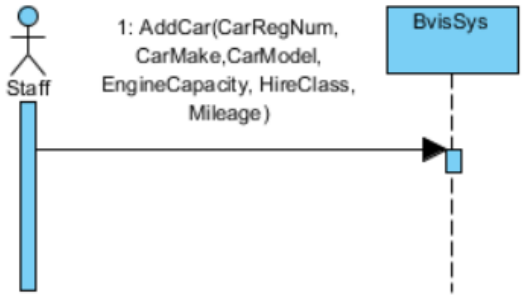

**Recovering Car's Service History**

**Recovering Car Information**

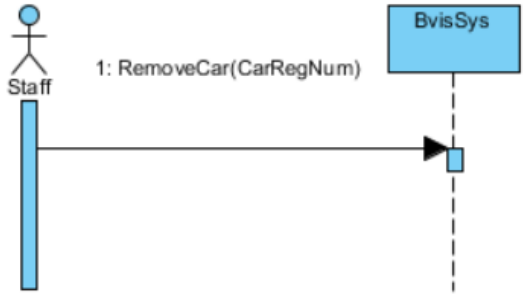



**Record Mechanical Service**


**Adding a New Car**



**Removing a Car**



## System Operation Contracts

Contracts have been produced for the most important method's that will be used in the system. Each contract defines what the method is responsible for, any exceptions noticed when performing the method, and the pre/post-conditions of the method being run.

**Contract for** *SystemInitialisation*

| | **Contract** |
|---|---|
| Name: | SystemInitialisation() |
| Responsibilities: | Initialises and boots the system |
| Type: | System |
| Cross References: | |
| Note: | System starts up |
| Exceptions: | |
| Output: | |
| Pre-conditions: | |
| Post-conditions: | • System can now perform all intended operations. |

**Contract for** *AddCustomer*

| | **Contract** |
|---|---|
| Name: | AddCustomer(CustName: text, CustAge: number, CustTele: number, CustAddress: address) |
| Responsibilities: | Enter a new Customer into the system with the required details. Ensure all details are present and correct, then register in the system. System ties a Unique ID to the new Customer. |
| Type: | System |
| Cross References: | System Functions: R1.1<br>Use Case: Registering a New Customer |
| Note: | |
| Exceptions: | If input details are missing or incorrect, indicate error. |
| Output: | A message is displayed confirming that the customer has been added to the database and the saved information is displayed on screen. |
| Pre-conditions: | • A Customer asks to register and provides their details.<br>• Staff Member inputs the details into the system i.e. name, age, address, contact number, etc. |
| Post-conditions: | • New Customer saved in the database.<br>• A message is displayed confirming customer has been added to the database. |

**批注 [ZL4]:** Not a system state

**Contract for** *RemoveCustomer*

| | **Contract** |
|---|---|
| Name: | RemoveCustomer(CustomerID: number) |
| Responsibilities: | Removes a selected customer from the system via their customer ID. |
| Type: | System |
| Cross References: | System Functions: R1.10, R2.4<br>Use Case: Removing a Customer |
| Note: | |
| Exceptions: | If chosen Customer ID does not exist on the system, indicate error. |
| Output: | A message is displayed confirming that the customer has been removed and data has been deleted from the database. |
| Pre-conditions: | ● The Customer asks to be removed from the system.<br>● The Staff Member access the customer file and request that the customer is deleted from the system. |
| Post-conditions: | ● A message is displayed confirming that the customer has been removed from the system. |

**Contract for** *AddMechanic*

| | **Contract** |
|---|---|
| Name: | AddMechanic(MechanicName: string, MechanicAge: int, MechanicAddress: string, DriverLicense: int) |
| Responsibilities: | Adds a new mechanic and stores their system. |
| Type: | System |
| Cross References: | System Functions: R4.1<br>Use Case: Addings a new mechanic |
| Note: | |
| Exceptions: | If input details are missing or incorrect, indicate error. |
| Output: | A message is displayed confirming that the mechanic has been added to the database and the saved information is displayed on the screen. |
| Pre-conditions: | ● A mechanic provides their details for registration.<br>● Details are input by a staff member. |
| Post-conditions: | ● New mechanic saved in database.<br>● A message is displayed confirming the mechanic was added. |

**Contract for** *RemoveMechanic*

| | **Contract** |
|---|---|
| Name: | RemoveMechanic(MechID: int) |
| Responsibilities: | Removes a selected mechanic from the system via their mechanic ID. |
| Type: | System |
| Cross References: | System Functions: R4.1<br>Use Case: Removing a mechanic |
| Note: | |
| Exceptions: | The ID supplied for the mechanic doesn't exist in the system. |
| Output: | A message is displayed confirming that the mechanic has been removed from the database. |
| Pre-conditions: | ● The mechanic asks to be removed from the system.<br>● Staff accesses the system and requests that the mechanic is deleted. |
| Post-conditions: | ● A message is displayed confirming that the mechanic has been removed from the system. |

**Contract for** *StartHire*

| | **Contract** |
|---|---|
| Name: | StartHire(DriverLicense: int, StartDate: date, StartMileage: int, ExpectedReturnDate: date) |
| Responsibilities: | Records the information of a car hire onto the system. |
| Type: | System |
| Cross References: | System Functions: R1.5<br>Use Case: Hiring a car |
| Note: | |
| Exceptions: | |
| Output: | |
| Pre-conditions: | ● A customer requests to hire a car.<br>● A staff member accesses the system and initiates the car hire. |
| Post-conditions: | ● The information of the car hire is saved onto the system. |

**Contract for** *ReturnHire*

| | **Contract** |
|---|---|
| Name: | ReturnHire(HireID: int, ReturnDate: date, NewMileage: int) |
| Responsibilities: | Records the return of a hire car and updates it's mileage. |
| Type: | |
| Cross References: | System Functions: R1.11<br>Use Case: Hire car return |
| Note: | |
| Exceptions: | The ID supplied for the hire ID does not exist in the system. |
| Output: | |
| Pre-conditions: | ● A car has been hired and the hire was registered on the system.<br>● The customer returns the hire.<br>● A member of staff initiates the return hire method. |
| Post-conditions: | ● The mileage of the car is updated.<br>● The car is available to be hired again. |

**Contract for** *RecordService*

| | **Contract** |
|---|---|
| Name: | RecordServive(ServiceDate: date, ServiveType: string, ServivceNote:string) |
| Responsibilities: | Recording a service performed by a mechanic on a car in the system. |
| Type: | System |
| Cross References: | System Functions: R3.5, R4.4<br>Use Case: Record Mechanical Service |
| Note: | |
| Exceptions: | |
| Output: | |
| Pre-conditions: | • A car and the mechanic must be registered in the system.<br>• A service must be completed on the car. |
| Post-conditions: | • The details of the service are recorded on the system. |

**Contract for** *AddCar*

| | **Contract** |
|---|---|
| Name: | AddCar(CarRegNum: string, CarMake: string, CarModel: string, EngineCapacity: string , HireClass: int, Mileage: int) |
| Responsibilities: | Records and saves the details of a new hire car onto the system. |
| Type: | System |
| Cross References: | System Functions: R3.1<br>Use Case: Addings a new car |
| Note: | |
| Exceptions: | |
| Output: | A confirmation message is displayed on the screen to show the car has been added to the system. |
| Pre-conditions: | • A member of staff must have the details required for the car. |
| Post-conditions: | • The car is saved on the system. |

**Contract for** *RemoveCar*

| | **Contract** |
|---|---|
| Name: | RemoveCar(CarRegNum: string) |
| Responsibilities: | Removes the information stored on a hire car from the system. |
| Type: | System |
| Cross References: | System Functions: R4.5<br>Use Case: Removing a car |
| Note: | |
| Exceptions: | Car registration number entered does not exist on the system. |
| Output: | |
| Pre-conditions: | ● The car must be registered on the system before it can be removed.<br>● A member of staff must request the car be removed. |
| Post-conditions: | ● The car is removed from the system. |

**Contract for** *ShowHireHistory*

| **Contract** | |
|---|---|
| Name: | ShowHireHistory(CarRegNum) |
| Responsibilities: | Display the Hire History of a car based on its registration number |
| Type: | System |
| Cross References: | System Functions: R3.2<br>Use Case: Recovering Car's Hire History |
| Note: | |
| Exceptions: | If selected car has no hire history, display error. |
| Output: | |
| Pre-conditions: | ● Car must have been hired previously.<br>● Registration number must be in the system. |
| Post-conditions: | ● Hire History displayed on screen. |

**Contract for** *ShowServiceHistory*

| **Contract** | |
|---|---|
| Name: | ShowServiceHistory(CarRegNum) |
| Responsibilities: | Display the Service History of a car based on its registration number. |
| Type: | System |
| Cross References: | System Functions: R3.2<br>Use Case: Recovering Car's Service History |
| Note: | |
| Exceptions: | If selected car has no service history, display error. |
| Output: | |
| Pre-conditions: | ● Car must have been hired previously.<br>● Registration number must be in the system. |
| Post-conditions: | ● Service History displayed on screen. |

**Contract for** *ShowCarInfo*

| | **Contract** |
|---|---|
| Name: | ShowCarInfo(CarRegNum) |
| Responsibilities: | Display the general information of a car based on its registration number. |
| Type: | System |
| Cross References: | System Functions: R3.1<br>Use Case: Recovering the Information of a Hire Car |
| Note: | |
| Exceptions: | If registration number is invalid, display error. |
| Output: | |
| Pre-conditions: | • Registration number must be in the system. |
| Post-conditions: | • Car information displayed on screen. |

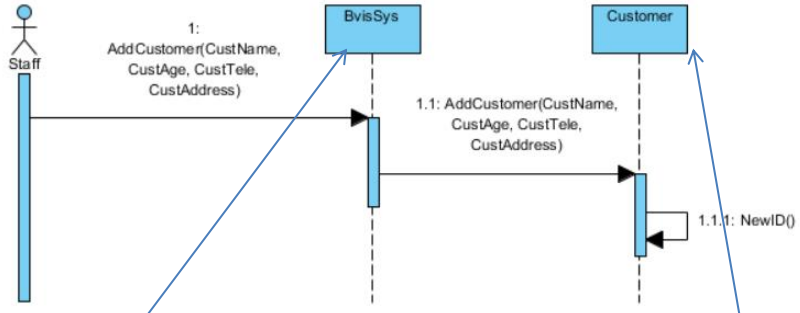
批注 [ZL5]: Misunderstanding of preconditions

# III. Use Case Design

## Object Sequence Diagrams

Object sequence diagrams have been created based upon the use cases identified. These diagrams show how objects in the conceptual class model interact with each other.

### Registering a New Customer

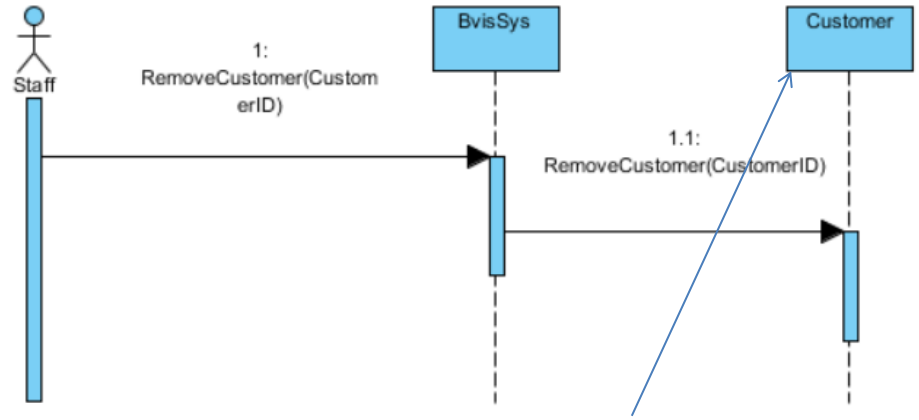



BvisSys acts as the Creator Pattern for this example as contains the initialising data that will be passed to Customer to create an instance of it.

Customer is the main expert patterns for the use case, as Customer is responsible for knowing the customer details as attributes, e.g Customer name, age, telephone number.

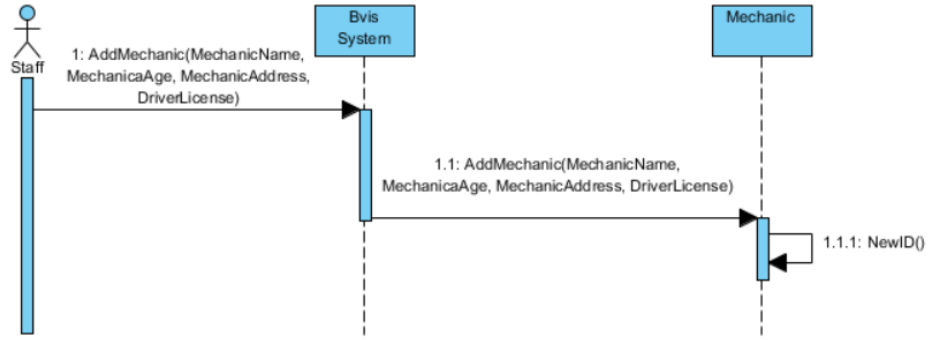
批注 [ZL6]: 1.1 should be creat()

**Removing a Customer**



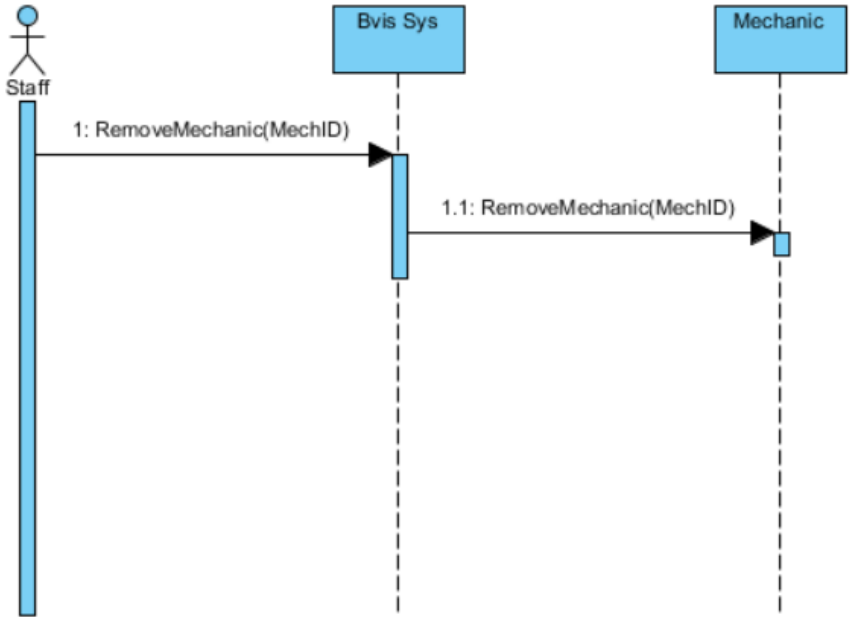

Customer is the main expert patterns for the use case, as Customer is responsible for knowing which ID related to which set of details of a Car Object.

**Adding Mechanic**




BvisSys acts as the Creator Pattern for this example as it contains the initialising data that will be passed to Mechanic to create a instance of it.

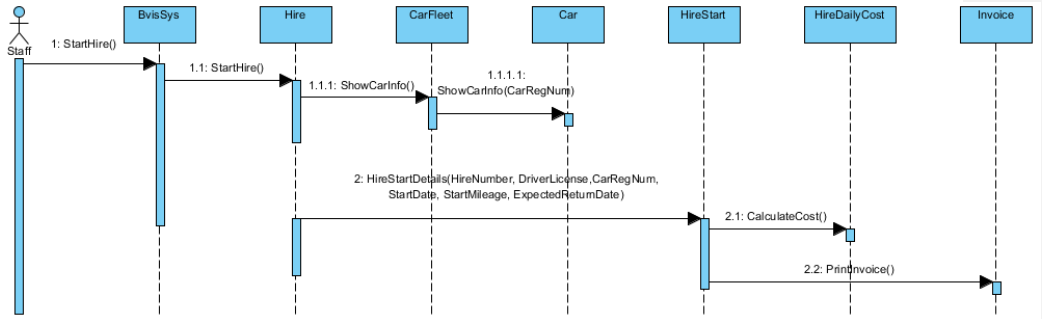
**Remove Mechanic**




BvisSys acts as the Creator Pattern for this sequence because it contains the initialising data that will be passed to create an instance of it.

Mechanic acts as an expert pattern that is responsible with knowing the instance of various mechanics, aswell as which ID to assign to which mechanic.

**Hiring a Car**




Creator Pattern:

BvisSys class is the class that has the initialising data that will be passed to the Hire Class when it creates a new instance of a hire class, thus the responsibility of the creation for a new car hire should fall to the responsibility of class System.

Expert Patterns:

The Hire class stores some of the relevant data for the hire use case, such as Hire id and drivers license, and also determines where the information about the hire, such as where to send the information about the start of the hire date and mileage. Hence the Hire Class should be responsible with call information of car info from CarFleet and also for sending information to the invoice class.

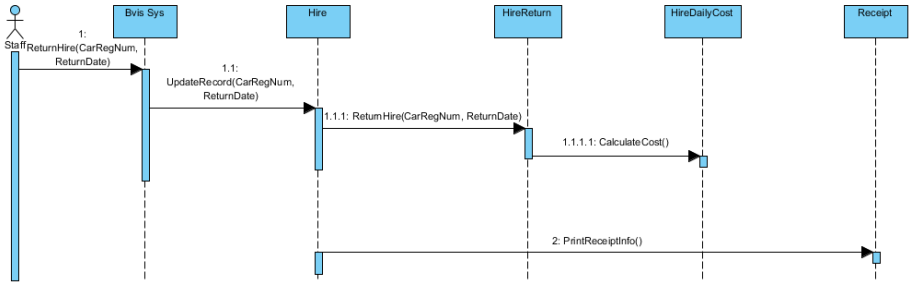
Controller Pattern:

Hire has the responsibility of controller as it is responsible for direct where the information is gathered from I.e the hire car info from the Car class, and it is also responsible for deciding where the information is meant to be assigned to, I,e the date of hire and the original mileage of the car assign with the hirestart class for example.

Cohesion:

Cohesion is reduced though the separation of a the hire task into separate classes. For example the task of keeping the information of the start date and the mileage of the start of the hire is kept within the HireStart class, and the task of printing the invoice i kept to the task of the invoice class.

**Hire Car Return**




Creator pattern:

Hire class acts as a creator pattern for the HireReturn as the class contains the data used to initialize instance of the HireReturn, and is primary one of its main associates to the rest of the system.

Expert Pattern:

HireReturn and HireStart are expert pattern type classes as they are both responsible for storing the the dates of Hire instance, as well as the mileage that the hire begins with and ends with respectively. They are responsible for deciding where the information is meant to go. HireCost is also an expert pattern as it not only is it responsible for storing the two dates, its also responsible for working out the cost of a hire based on the difference in days between the two dates times the cost of its hire class.

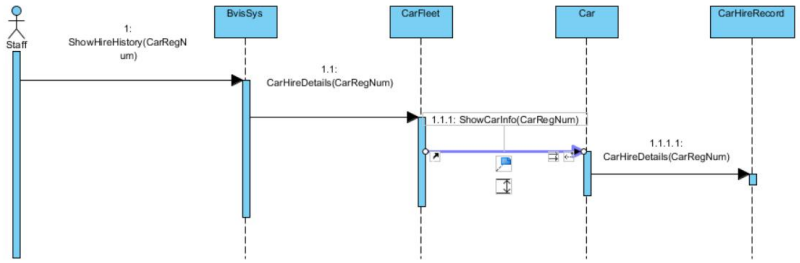
Controller Pattern:

Hire has the responsibility of controller as it is responsible for direct where the information is sent in during the hire, such as the return date to the HireReturn Class for example and the where the new mileage value should be sent.

High cohesion:

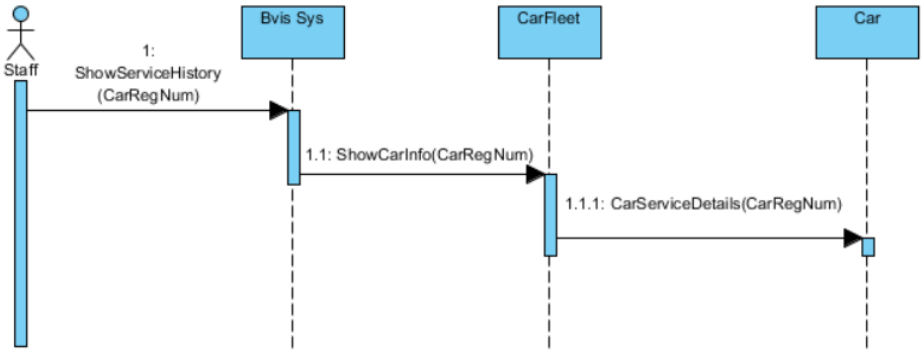
Cohesion is reduced though the separation of a the hire task into separate classes. For example the task of keeping the information of the start date and the mileage of the start of the hire is kept within the HireStart class, and the task of printing the invoice i kept to the task of the invoice class.

**Recovering Car's Hire History**



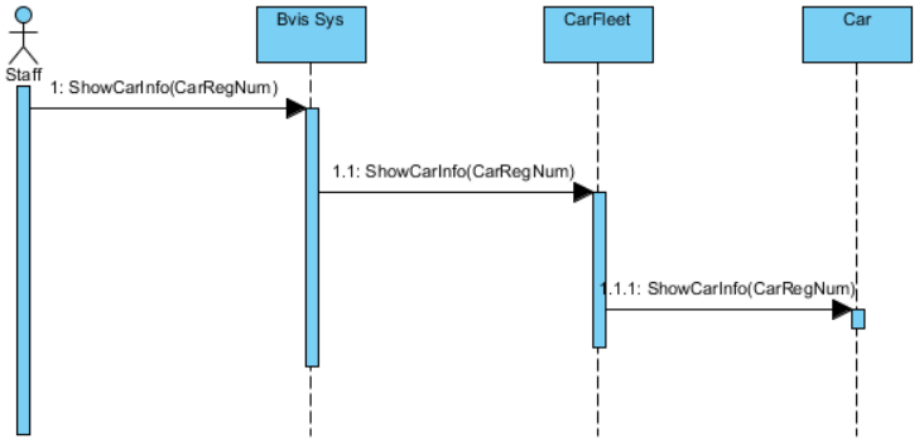

CarFleet knows where the CarHireRecords related to the Car registration number is stored as it has associations with Car, which in turn knows where to link the instances of CarHireRecord to, linking it a example to a expert pattern. Thus, CarFleet is responsible with showing the the hire history of a car.

**Recovering Car's Service History**



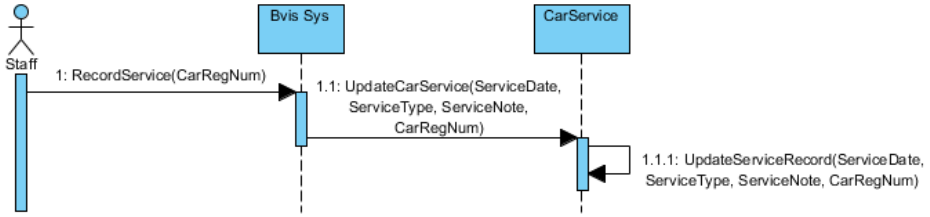

CarFleet knows where the service details related to the Car registration number is stored as it has associations with Car, which in turn knows which car registration number is link to which set of records. By this fact, CarFleet holds the responsibility of a expert pattern, thus CarFleet is responsible with showing the the hire history of a car.

**Recovering Car Information**



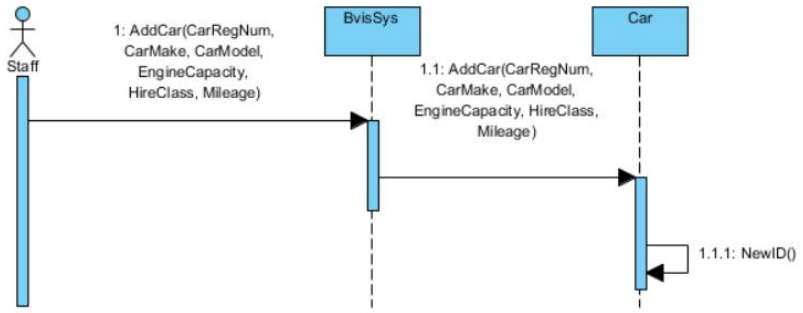

CarFleet knows where the details related Car registration number is stored as it links to the instances of cars that Car store, which intern knows where to link the instances of CarHireRecord to, linking it a example to a expert pattern. Thus, CarFleet is responsible with showing the the hire history of a car.

**Record Mechanical Service**



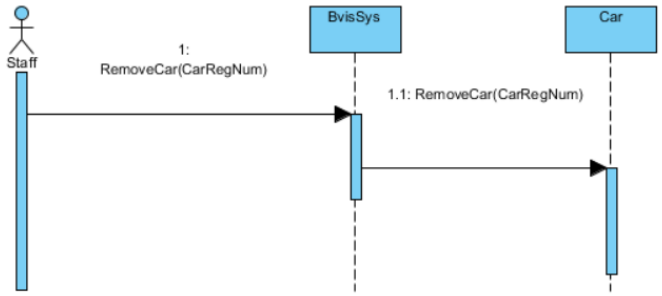

CarService stores information of the instances of a car services it receives from the BvisSys, an ability that's important for a expert pattern with the responsibility  to store service records.

**Adding a New Car**




BvisSys is the first class that contains the initializing data that is passed to class Customer to create a Customer Object. This indicates that BvisSys, by definition of the creator pattern, is responsible for creating new instances of Customer.

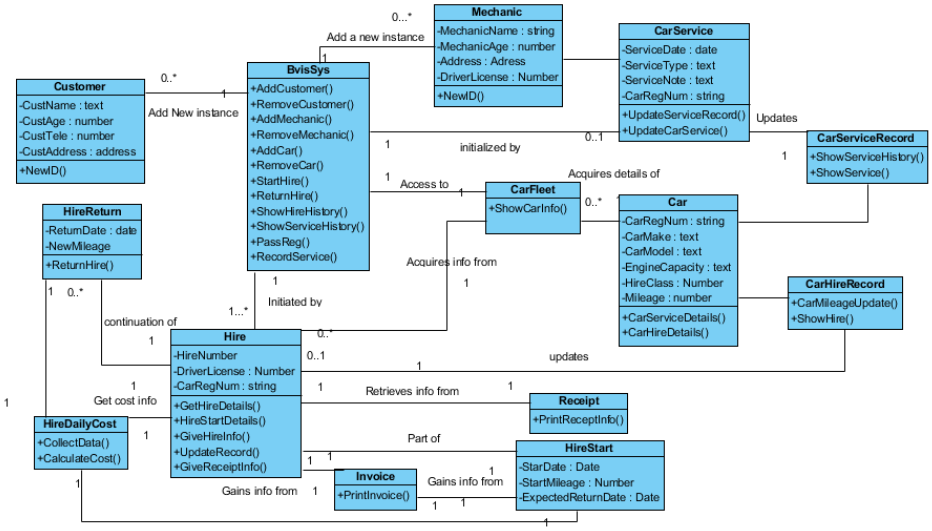
**Removing a Car**




BvisSys carries the main information that is needed to determine which instance of car to remove from the system, give that is received the registration number of car itself. This indicates that BvisSys feature of a expert pattern associated with

## Design Class Diagram

A design class diagram has been produced for the system, whilst this is similar to the conceptual class diagram, it contains the methods that are used to allow the system to operate across the different objects.




## Glossary terms

| Name | Meaning |
|---|---|
| BvisSys | The main interface point to contact other concepts and use cases. |
| NewCustomer() | Calls for a new instance of a Customer. |
| RemoveCustomer() | Removes an instance of a Customer |
| AddMechanic() | Add an instance of a Mechanic |
| RemoveMehanic() | remove an instance of a mechanic |
| AddCar() | calls for a new instance of a car |
| RemoveCar() | Removes an instance of a car |
| StartHire() | Create a new instance of a hire |

| | |
|---|---|
| ReturnHire() | Update the hire of the hire return |
| ShowHireHistory() | show the complete hire history of a car. |
| ShowServiceHistory() | show the complete service history of a specific. |
| RecordService() | Record a service report for a certain car. |
| Customer | Storing customer details and information. |
| Customer.CustomerName:string | Name of the customer. |
| Customer.CustomerAge:Number | The age of the customer. |
| Customer.CustomerTele:Number | Telephone number for the customer. |
| Customer.Address:address | The house number, street name and postcode of the customer. |
| Mechanic | Storing the details of a new mechanic. |
| Mechanic.MechanicName:string | The name of the mechanic. |
| Mechanic.MechanicAge:Number | The age of the mechanic. |
| Mechanic.Address:address | The address of the mechanic. |
| Car | Creates an instance of a new car and stores the information. |
| Car.CarRegNum:string | Car registration number. |
| Car.CarMake:string | Car Make, i.e. Company that makes the car. |
| Car.CarModel:string | Model name of the car. |
| Car.EngineCapacity:string | The engine capacity of the car. |
| Car.HireClass:number | The hire class number of a car. |
| Car.Mileage:number | The current mileage value of a car. |
| CarService | Used to create an instance of a service report made by a Mechanic. |
| CarService.ServiceDate:date | Storing the date of the service in question. |
| CarService.ServiceType:String | The type of service being made (Minor/Major). |

| | |
|---|---|
| CarServiceRecord | Stores every new instance of a car service and used to call upon a list of new instances. |
| CarHireRecord | Stores every new instance of a car hire for a car. |
| Hire | Creates an instance of a hire process. |
| Hire:DiverLicense.DiverLicense | Storing the Divers License. |
| HireStart | Creates an instance of starting process for a hire class. |
| HireStart.HireDate:Date | Stores the start date of a Hire process. |
| HireStart.OrginalMileage:number | Stores the original mileage record in hire process. |
| HireStart.Date.ExpReturnDate | Stores the date of the expected return date of a hire. |
| HireReturn | Contains the information of the return details of a hire process. |
| HireReturn.ReturnHireDate:Date | The date of the car being returned. |
| HireReturn.Mileage.Number | The new mileage of the return car in a hire process. |
| HireCost | Calculates the cost of a hire by using the difference between of days between the starting hire date and the ending hire date. |
| Reciept | Using the key information from the hire, this produces a printable receipt. |
| HireDailyCost | Calculate the cost of a hire car based on user input, or automatically calculated based on the time difference between a hire start date and the return date. |
| Invoice | Creates an invoice of the start of a hire. |

# Appendix

## Meetings

### Group Meeting 1 – Introduction

**Initial Ideas**
**Date: 10th March 2015**
**Time: 3:00PM**
**Room: MP252**

**Project Manager: Ciaran Keogh (CK)**
**Group Members: Bilal Husain (BH), Matthew Hinton (MH), James Frith (JF), Thomas Hulme (TH), Abdul Lefsay (AL)**

Minutes:

1. Apologies for absences
   None.

2. Introduce members to the projects
   The project manager and group members are introduced to one another and are conducting the following meeting to talk about the project that has been set forward.
   The project has been identified by the group as a hiring system for a car hire company in UML (Unified Modelling Language).
   It was decided between all members that group members were to be delegated tasks by the team leader (or volunteered to do tasks).
   The deadline for the project will be 21st April 2015 and was discussed during the meeting to acknowledge any future time constraints.

3. Requirements of Project
   The requirements for the project are that they should have all the functionality described in the assignment brief, this includes tasks such as registering a new customer, recording when particular cars have been hired, returned and cost of those cars based on those two factors. It should also display details of the car hired and log a completed hire and the user should be able to remove customers and cars from the database. This and many more functions of the project have been discussed during this meeting but none have been delegated as of yet. Plans to do that were discussed and were agreed to be discussed in the next meeting. CK suggested that we all research more on UML using materials such as the software design lecture slides so that we are all comfortable with the concepts by the next meeting.

4. Outline content

The content of the project will be based on UML and will implement the uses of use cases and functions. The group also discussed the assignment brief requirements and that it requires write-up of a minimum of 3000 words as well as the use of use case diagrams.

5. Purpose of the project
   The purpose of the project has been identified as a base for the design of software that any potential software developer could pick up and use to follow the design aspects for this software. This would include how the software handles, stores and transfers data as well as how the system will operate with certain actors.

6. Project layout
   It was discussed that the work would be laid out according to the assignment brief in that it would be laid out in such a way that it would have clearly defined sections each with descriptive introductions and explanations in regards to tables, UML/class/object diagrams e.t.c.
   CK had also suggested that we work on this project through the use of google docs on google drive. This way all the group members can work on one document simultaneously from their own homes and can see the changes that others are making in real-time. This was the agreed on as the method of doing/laying out the work.

7. Any other business
   None.

8. Date and time of next meeting
   17th March 2015
   3:00 PM

## Group Meeting 2 – Introduction

**Delegation of tasks**
**Date: 17th March 2015**
**Time: 3:00PM**
**Room: MP252**

**Project Manager: Ciaran Keogh (CK)**
**Group Members: Bilal Husain (BH), Matthew Hinton (MH), James Frith (JF), Thomas Hulme (TH), Abdul Lefsay (AL)**

Minutes:

1. Apologies for absences
   None.

2. Minutes from last meeting
   Minutes from last meeting were shown to the group and have been confirmed to be accurate.

3. Present updated group progress
   The tasks set from the last meeting were that the group members should research more on UML and case/object diagrams as well as use cases and then come back to the second meeting for the delegation of tasks. All group members have said to have done research and have said to have a clear understanding of the tasks needed for the project.

4. Matters arising
   After discussion on matters that could be arising it seemed that a lot of the research done by the group were mostly just from the lecture slides from the module Software Design and not much external research was done. Those that did do external research did not do much, mostly due to lack of content online or "vague" definitions of certain terms. It is therefore imperative that any group member that is not fully clear on any definition needs to make sure that they get that done or at least get help from another member that does understand more.

5. Discussing improvements to be made
   No improvements were required to be made as the write-up has not officially started. There is a room for improvement in which it was previously stated that some more external research will need to be done in order for members of the group to get a clear understanding of some of the definitions of UML such as "Patterns" and "Contracts". CK has offered to help those struggling with some terms by explaining the terms to them. This was done so that the group could quickly move onto starting the write-up.

6. Status of the Project
   The project is now at a point where all group members have an understanding of the majority (if not all) of the terms of UML as well as use cases/diagrams. During the meeting it was advised

by CK that the group should start working on tasks that will be delegated to them to which all the group members agreed on. The tasks CK had given out were that of 2 use cases per group member (adittionally TH would help improve on use cases done by other members). After this, it was suggested by CK that by the next meeting the group members doing use cases should at least have the use cases done and/or the expanded use cases also.

7. Any other business
   None.

8. Date and time of next meeting
   24th March 2015
   3:00 PM

## Group Meeting 3 – Progress update

**Date: 24th March 2015**
**Time: 3:00PM**
**Room: MP252**

**Project Manager: Ciaran Keogh (CK)**
**Group Members: Bilal Husain (BH), Matthew Hinton (MH), James Frith (JF), Thomas Hulme (TH), Abdul Lefsay (AL)**

Minutes:

1. Apologies for absences
   None.

2. Minutes from last meeting
   Minutes from last meeting were shown to the group and have been confirmed to be accurate.

3. Present updated group progress
   The tasks set from the last meeting were that the group members and the Project Manager should have at least done their use cases with some diagrams of systems operations of those use cases if applicable. CK had offered help to the group members who need it with definitions of some words or concepts of some use cases. BH, MH, JF, CK and AL have all done their use cases and put them into the coursework draft with AL also having stated the System operations diagram for his use case. TH has completed a full write-up of the introduction and has done a majority of the System Function tables as well as corrected or improved other members use cases.

4. Matters arising
   After discussion on matters that could be arising it was concluded by all group members that no matters were in need of discussion. However, CK had mentioned that there were two more use cases to be done and as such asked BH and MH to do one extra use case each in addition to their own.

5. Discussing improvements to be made
   CK had said that all the use cases were fine but requested that they were laid out as they were shown on the lecture slides. This was due to the fact that the group members just laid out the use cases as bullet points. CK also pointed out that that each use case requires and extended use case, which some members did not include. CK then suggested that those who have not done the extended use cases should do so as soon as possible and also start doing the System operations diagrams for each use case. TH was told by CK to finish off doing the rest of the functions table and start doing a conceptual class diagram based on his identifying concepts whilst CK stated he will be doing his systems operations as well as the use case diagram.

6. Status of the Project
   The project is now at a point where the write-up has started and is progressing very well. CK had then stated that we had all made good progress and that we can move onto doing our individual system operations diagrams as well as other tasks such as descriptions of system operations diagrams. The project is progressing well too and if it keeps up then as CK had stated, this could be done or close to done within another two meetings.
   CK also stated that due to next week being the Easter holidays, he had suggested that the next meeting should be the week after next. All group members agreed to this and stated they had no problem with this.

7. Any other business
   None.

8. Date and time of next meeting
   7st April 2015
   12:00 PM

## Group Meeting 4 – Progress update

**Date: 7th April 2015**
**Time: 12:00PM**
**Room: Millennium Point Common Room**
**Project Manager: Ciaran Keogh (CK)**
**Group Members: Bilal Husain (BH), Matthew Hinton (MH), James Frith (JF), Thomas Hulme (TH), Abdul Lefsay (AL)**

Minutes:

1. Apologies for absences
   None.

2. Minutes from last meeting
   Minutes from last meeting were shown to the group and have been confirmed to be accurate.

3. Present updated group progress
   The tasks set from the last meeting were that the group members and the Project Manager should have at least done their system operations diagrams and extended use cases if still required to be done. CK has stated that he has done the systems operations diagram as well as his own use case. BH, MH, JF, CK and AL have all done their use cases and extended use cases and have put them into the coursework draft. TH has further improved upon use cases as well as made suggestions for system operation diagrams as well as finishing the functions tables whilst also doing the conceptual class diagram. BH and MH have also done their additional use cases with extended use cases and only had the Sequence and Object diagrams left to do.

4. Matters arising
   After discussion on matters that could be arising it was clear to CK that we were running out of time and needed to increase our progress speed as the deadline was fast approaching. CK stated that from the last meeting to this one there has not been as much progress on the work as compared to the other previous progress meetings and that if the group want to meet the deadline then they would be required to increase the amount of work done by the next meeting. The group members all agreed that this was a matter that needed to be resolved.

5. Discussing improvements to be made
   CK had expressed that there were not many improvements to be made apart from the extended use cases which were not laid out properly, like the use cases in the previous meetings. This was due to the fact that the group members just laid out the use cases as bullet points.

6. Status of the Project

The project is now at a point where the write-up is progressing well although with the deadline fast approaching the group needs to increase the speed of their workload. However it was also pointed out that if all the contracts were to be done by the next meeting then the majority of the work should already be done leaving little left to do. CK pointed out that each use case requires a contracts as well as an extended use case (which has already been fulfilled by each group member) TH volunteered to do the use of patterns. All group members acknowledged this and have agreed on what they should do for next meeting.

7. Any other business
   None.

8. Date and time of next meeting
   17th April 2015
   12:00 PM

## Group Meeting 5 – Progress update

**Date: 17th April 2015**
**Time: 12:00PM**
**Room: Millennium Point Common Room**
**Project Manager: Ciaran Keogh (CK)**
**Group Members: Bilal Husain (BH), Matthew Hinton (MH), James Frith (JF), Thomas Hulme (TH), Abdul Lefsay (AL)**

Minutes:

1. Apologies for absences
   None.

2. Minutes from last meeting
   Minutes from last meeting were shown to the group and have been confirmed to be accurate.

3. Present updated group progress
   The tasks set from the last meeting were that the group members and the Project Manager should have at least done their contracts for the use cases and TH should have done the use of patterns. During the meeting it was confirmed that all the tasks that were set out for the members were fulfilled.

4. Matters arising
   After discussion on matters that could be arising it was clear to CK that we were running out of time only we were now on schedule to complete the work due to the group catching up with the workload. CK has stated that the group should still keep progressing at the current speed and have the project finished by the next and final meeting if they are to finish the project before the deadline without doing things at the last minute.

5. Discussing improvements to be made
   The group acknowledged there were not many improvements to be made although CK still expressed concerns about the layout of the work overall. He expresses this due to the fact that the group will be marked on layout. The group agreed and BH had stated that the reason the layout has not yet been adjusted it due to the fact that the group have just been concentrating solely on getting content into the project. CK agreed and stated that the content should be done first and that the layout will be done once the content is finished.

6. Status of the Project
   The project is now at a point where the write-up is nearing a close and the group needs to finish the content and possibly amend the layout by the next meeting in order to meet the deadline.

7. Any other business
   None.

8. Date and time of next meeting
   20th April 2015
   12:00 PM

## Group Meeting 6 – Conclusion

**Date: 20th April 2015**
**Time: 2:00PM**
**Room: MP**

**Project Manager: Ciaran Keogh (CK)**
**Group Members: Bilal Husain (BH), Matthew Hinton (MH), James Frith (JF), Thomas Hulme (TH), Abdul Lefsay (AL)**

Minutes:

1. Apologies for absences
   None.

2. Minutes from last meeting
   Minutes from last meeting were shown to the group and have been confirmed to be accurate.

3. Present updated group progress
   The tasks set from the last meeting were that the group members and the Project Manager should have finished all their tasks and should be bringing the project to a close. The layout of the project should also be done and the work should be ready for submission. The group work has all been done and the layout has been amended. CK and the group are happy with the work. The work was also presented to the group one last time and BH made the group write their names on the cover letter. Overall CK and the group were very happy with the work they did and felt that they all presented the work well.

4. Status of the Project
   The project is now finished and will be submitted. All the group members are happy with the project and are therefore happy to be submitted. CK will be submitting it tomorrow when BH has scanned the cover sheet and sends it to CK.

5. Any other business
   None.

6. Date and time of next meeting
   None.

## Work Allocation

In the following table it is shown that each use case task was assigned to each group member. When a group member is assigned a use case this means that the member has completed all the sequence and object diagrams for that use case as well as any other contracts for that use case.
Each member was then able to contribute the use case towards diagrams such as the Design Class diagram and Conceptual Class diagram.
Additionally to Note, Thomas Hulme has done the introduction for the project as well as the function tables whilst Ciaran Keogh has identified the use of patterns in Object Sequence diagrams.

| Task Name | Assigned Team Members | Completed By | Date of Completion |
|---|---|---|---|
| Registering a new customer | Abdul | Abdul | 1st April |
| Removing a customer | Abdul | Abdul | 19th April |
| Recording the Hire of a Car | Ciaran | Ciaran | 2nd April |
| Recording the return of the Car | Ciaran | Ciaran | 17th April |
| Recovering information of hire car | James | James | 19th April |
| Record a Car service | James | James | 19th April |
| Adding a new Mechanic | Matthew | Matthew | 13th April |
| Removing a Mechanic | Matthew | Matthew | 13th April |
| Adding a new Car to the fleet | Bilal | Bilal | 5th April |
| Removing a Car from fleet | Bilal | Bilal | 8th April |
| Recovering a Car's service history | Thomas | Thomas | 7th April |
| Recovering a Car's Hire history | Thomas | Thomas | 13th April |