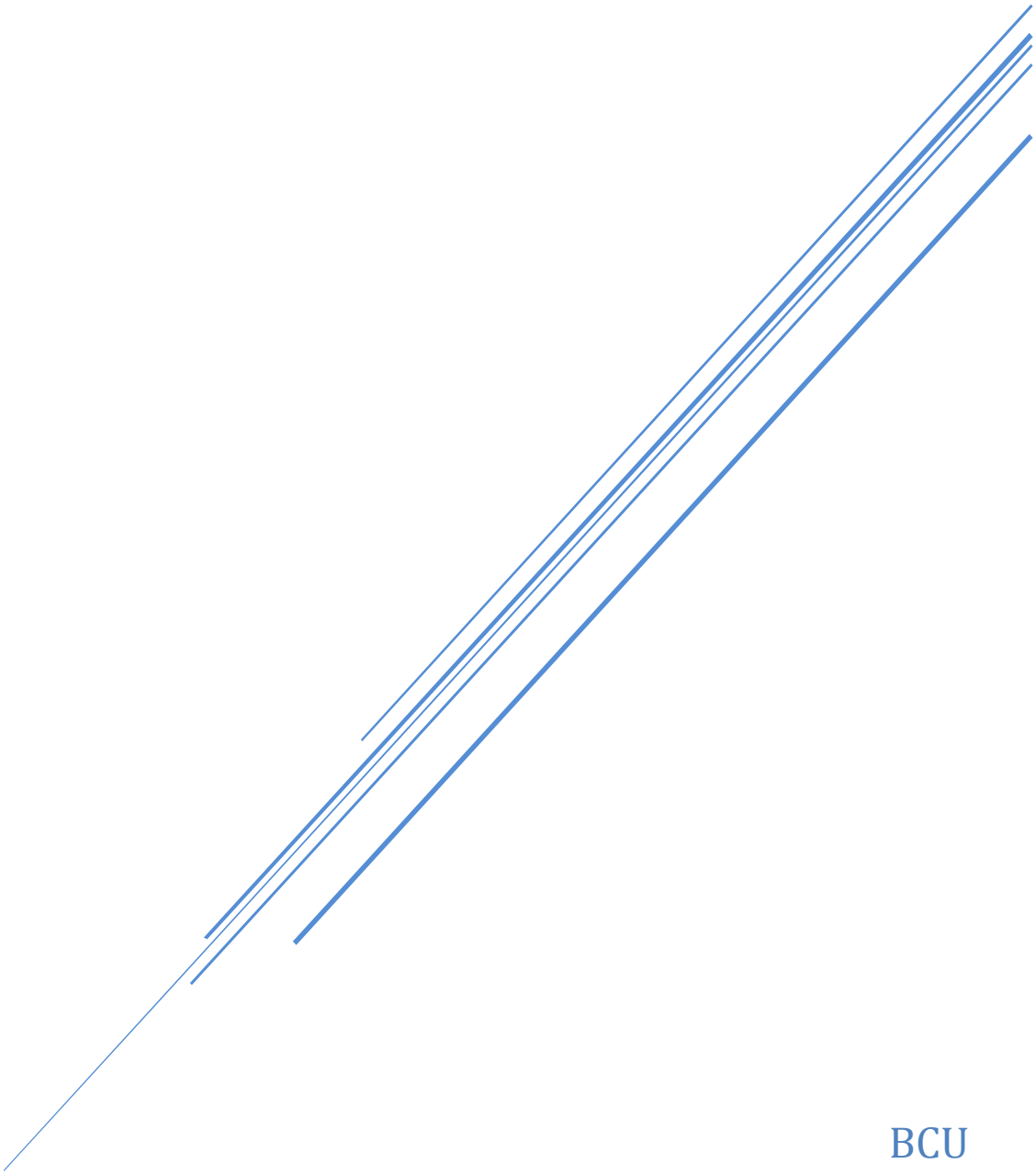
TEAM PROJECT

Software Design



BCU
Rowan Bloomfield, Michael O'Keefe, Samuel Jervis, Ashley Hussey, Joshua Allen

Team Names /ID's

Rowan Bloomfield – S12777225

Mike O'Keefe – S12783789

Samuel Jervis – S11712018

Josh Allen – S13173637

Ashley Hussey – S12768787

Part 1

Initial discussion

Bvis car hire is a car hiring service as the name suggests which uses a paper based system to do everything such as: log customer details, car details and the overall services. Using a paper-based system in such a complicated business structure can bring many problems, which is why a new system should be put in place.

Current system

For a car to be hired by a customer using the current system a customer must register by providing this information: name, telephone number and address, these details are then filed into the paper system which means they are registered and free to rent one of the available vehicles.

The next set of details in the system are that of the vehicles, each car will have these details within the system: registration number which are all unique, make of car, model of car, engine capacity, hire class and the date of registration / date of each service, the mileage at each service together with the name of the mechanic who serviced the vehicle. Again all this data is kept within the paper based system, each vehicle that the company has needs to have all this data included before and after each hire, especially the service and mileage information.

Certain information for the cars need to be updated after every rental such as the mileage, this is because after every 6,000 miles each car needs to have a minor service and after 12,000 miles the cars need to have a major service. The company services its entire car as it has its own garage and mechanics, which do the jobs services as soon as possible after any rental.

When new customers hire a car all there details are logged onto the hire form for the car, the details on this form needed from the customer are: name, address, telephone number and driving license number. To complete the rest of the hire form further details are needed which are: the ID of the hired vehicle, the start/end dates of the hire (end date been a estimate and not final value) and finally the mileage on the vehicle at the start of the hire (needed for when the vehicle is returned to see if a service is needed). Once the hired vehicle has been used and returned the final details are added to this form, which the actual, return date and the end are of hire mileage.

Finally once a vehicle has been returned the daily hire amount is used to calculate a price for the customer, this must be paid cash only and once paid a receipt will be provided, once all this has been completed the customer details must be logged. Certain records are also kept within the company, first been a record of the hire class of vehicles (daily, weekly, monthly), secondly

is rates of hire which varies per vehicle, finally the garage records for each mechanic (each mechanic must have a driving license)

System problems

The main and obvious problem with the current system is that it is paper based; this draws a number of concerns and problems. This car hiring service collects a lot of data which makes using a paper based system very dangerous. Customer's details, car details and mechanic details are some of the types of data that are kept within the system some of this are very sensitive data.

One problem with a paper based system is that it makes it hard to back log all the data, it may be easy to fill out forms but once those forms are filed they will be very hard to retrieve as time goes on where as a computer based system wouldn't have this problem. Another problem could be the safety of the data a computer can be very easily protected were as files can not only be stolen but lost within the company (epically if they are not backed up)

Also once cars are returned / need servicing all the data for the car will need to be calculated by a human then rerecorded which takes a lot of time which could be spent doing other jobs. These are a few of the problems, which arise when using a paper-based system for such a complicated business structure.

The way to make complex structures a lot simpler is to add simple systems or systems that make the structure a lot simpler, this is done by identifying key areas and building the system around these areas eradicating any problems stemming from them areas.

New system

The new system should be object orientated as object orientated programs revolve around using many objects which data which then combine together to create a bigger program or system, this would work in this instance as this system will have big classes such as: car, mechanic and customer.

Each class can be an object within the system but each class will break down into many different sub classes which is where the complexity of the object orientation comes in, below are examples of how the classes will work in the new system which will no longer be paper based.

The new system will work the same but there will be a transition from paper based to a computer based object orientated system, which will work better for the reasons previously stated. Once this new system is implemented many things should improve such as: faster and more effective car hires, more consistency with data and data handling, better customer service and easier back logging.

Records include (Classes and attributes):

- Car - Registration number, make, model, engine capacity, hire class (1-6), registration date, mileage of service, mechanic, hire history, service history
- Customer - Name telephone number, address
- Mechanic – Name, address, telephone number, license (required)

Symbol, intention, extension

Example:

- Symbol – Car
- Intention – Car to be hired by customers
- Extension – Ford Focus. LS45 GHT rented by John Jackson with customer id 1111

Example 2:

- Symbol – Customer
- Intention – Customer wants to hire vehicle
- Extension – The customer John Jackson with customer id 1111 would like to hire the Ford Focus. LS45 GHT

Objects defining features.

Car

- Identity = Registration number
- State = make, model, engine capacity, hire class (1-6) , registration date , mileage of service , mechanic , hire history , service history
- Persistence = registration date, hire history, service history
- Behavior = Interacts with customer class plus mechanic class

Part 2

System functions

Here are the system functions which will be used within the new system, system functions are provided for every section of the car hire and also the system overall.

Customer sign up function

REF NO	Function	Category
1.1	Customer gives details	Evident
1.2	Checks if customer is already registered	Hidden
1.3	New customer created if needed, ID created	Evident
1.4	Customer details stored	Hidden

Car rental

REF NO	Function	Category
2.1	Car requested	Evident
2.2	Car details logged	Evident
2.3	Car given	Evident

Payment

REF NO	Function	Category
3.1	Car returned	Evident
3.2	Mileage logged	Hidden
3.3	Price calculated through hire class etc.	Hidden
3.4	Payment taken	Evident
3.5	Receipt printed and given	Evident

Service

REF NO	Function	Category
4.1	Car mileage checked	Evident
4.2	Given to mechanic if needed	Evident
4.3	Minor or major service	Evident
4.4	Ready for next hire	Evident

System function

Attribute	Constraints
Operating system	Windows 7 or 8
Payments	Cash only
Customer / car / mechanic details	Stored within database

System attributes

Rental system

- Order – request from the company to hire one of the cars
- Item – the cars available for hire
- Invoice – a request for payment sent from the seller to the customer
- Account – payment

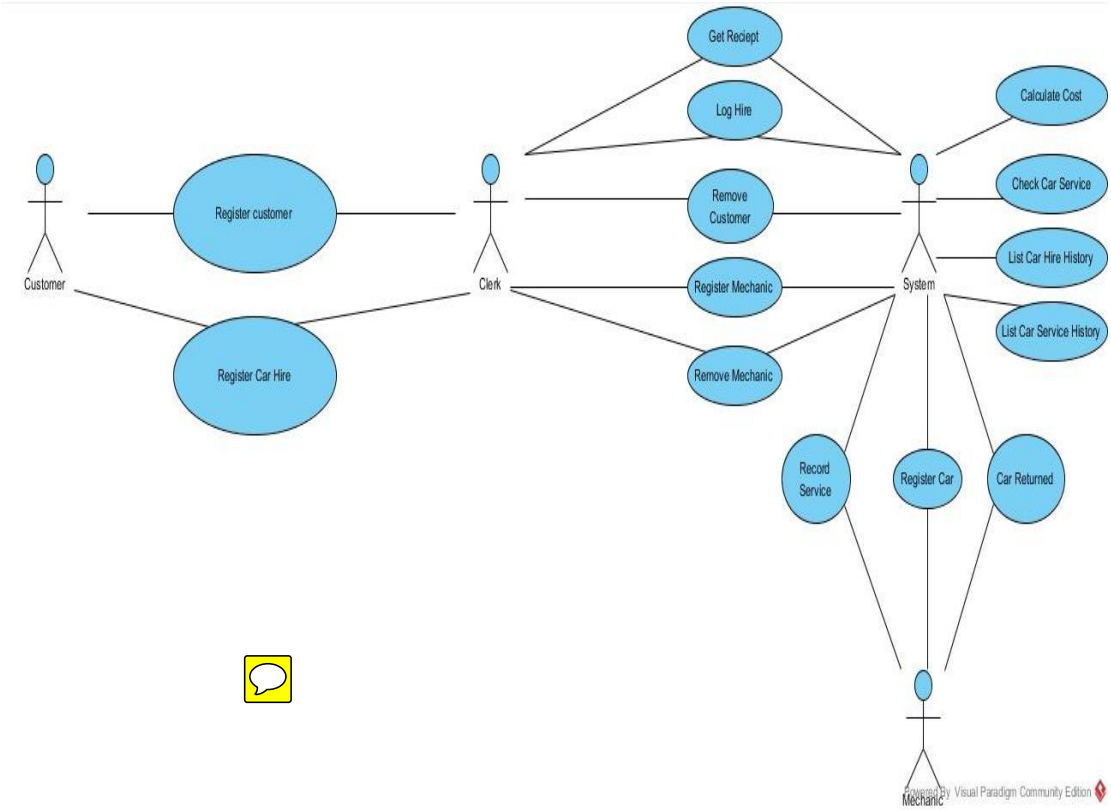
Part 3, 4 and 5

A use case diagram is a graphic depiction of the interactions among the elements of a system, in other words it's almost like a flow chart of processes carried out by actors within a system.
Use case diagrams are used for software design to plan and show what the main aim of the software is, what processes will be carried out for the software to function. Use case diagrams usually have 3 main functions which include Actors, Use cases and the Relationships.

Actors are usually individuals involved in use case diagrams to show the people involved in the process specified by their roles. For example a customer would be classed as an actor talking to a shop assistant who would be classed as another actor.
Actors communicate with the system by sending messages to and receiving messages from the system. Actors can be human users or a digital device/computing system.

The use cases are the tasks/roles that are carried out by the actors which are involved in the process within the system. For example a customer would ask the sales assistant for a specific item or assistance and that would be a use case as it is a process being carried out by the actors.

The Relationships are what links the use cases together with the actors. Relationships are the task that are carried out in the process by the actors which can be linked to other use cases and actors within the use case diagram.



The picture above is our group's Use Case Diagram used for our team project. We were tasked with turning a paper based car rental service into a more modern digital/electronic based service now.

As you can see in our use case diagram we have 4 actors, 3 being human actors and one being a computer system. It covers the initial interaction from the customer to the sales clerk handing over details about themselves to create an account to the mechanic telling the system a car has been serviced and ready to be used. Everything links back the System Actor which has use cases direct from the Mechanic and the Clerk but takes the data from the Customer which then passed onto the Clerk then inputted into the system.

The main use cases in this diagram are Register Customer, Register Car Hire, Record Service and Payment.

Register Customer is important because the business obviously needs customers as well as having a record of this person who is renting a car. Registering a customer to the database makes things run smoother and easier for the company because if they ever needed to get in touch with one of their customers regarding a payment issue or checking if they are satisfied with the car they can easily be found on the system database.

Register Car Hire is also important because the clerk needs to know what cars are available for new customers or existing when they come to rent a car. Making a record of every car that is brought into the company is essential.

Record Service is a main use case because the cars need to be checked over before they are allowed to be rented as it could be dangerous to the customer and costly to the business. One of the actors in this process is the Mechanic which deals with everything related to checking the cars thoroughly then inputting the data in the system if the car is fine or if car has been serviced/repaired.

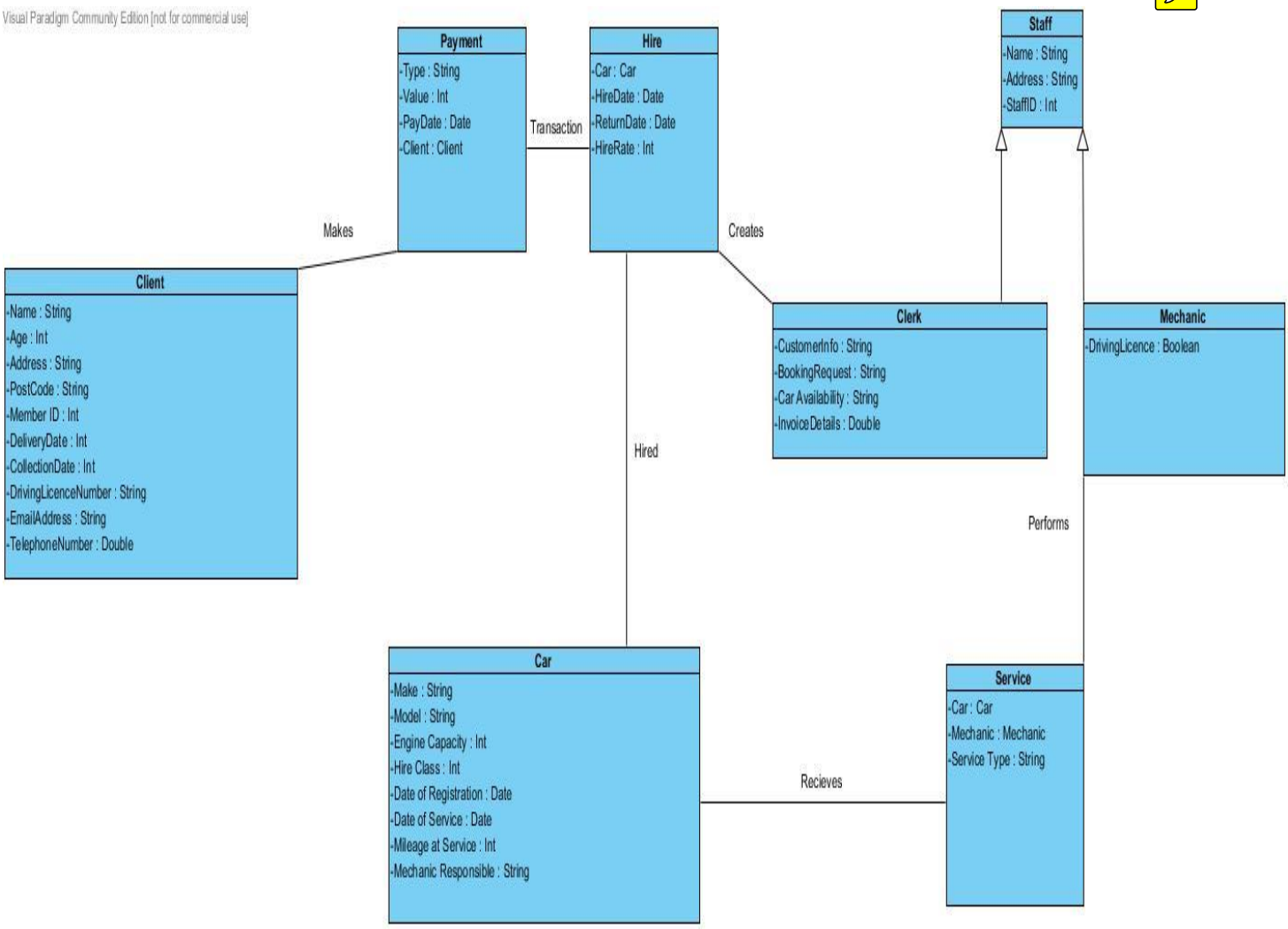
Payment is the last main use case in our diagram. The reason Payment is a main use case is payment is needed for cars to be rented.

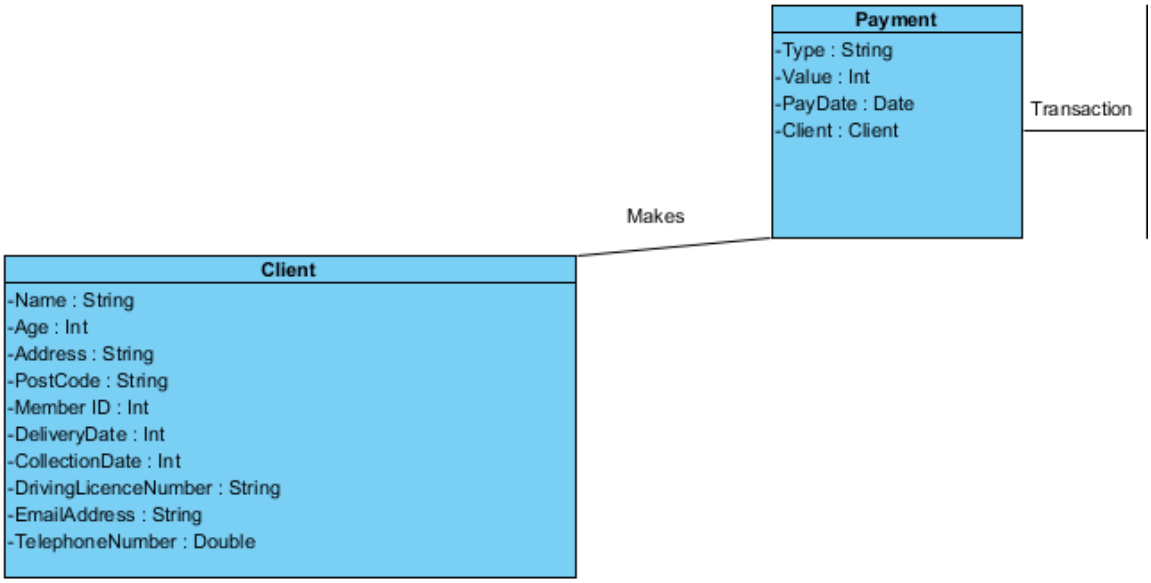
All of the Use Cases are clear and concise nothing to difficult to understand, they are straight to the point no underlying messages. A drawback is maybe they are too simple where they could be condensed down to have few less use cases.

Part 6

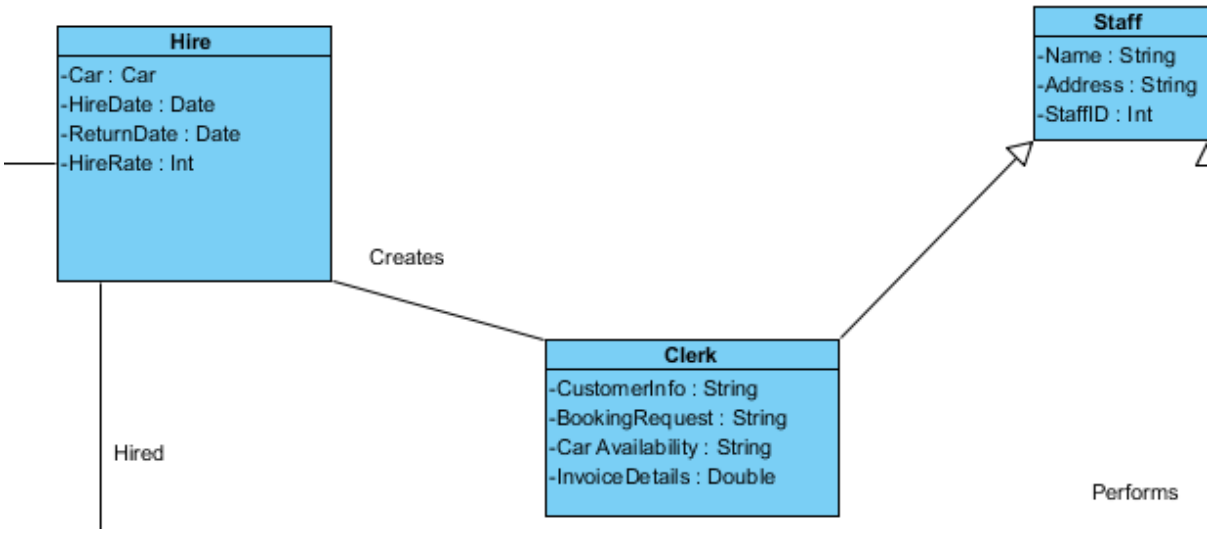
Class diagrams

A Class Diagram is a static view of a proposed software system. They are used to analyse and design an application and show the responsibilities and behaviours of a software system. The purpose of a Class diagram is to specify how the structure of the system will look. Class diagrams show how the software system will function and how it will be written before any code has been implemented. Class diagrams are very useful when designing large systems; they will also show the developer what classes will need to be created to meet the user requirements of the desired system. The class diagram will also show what attributes and methods will be needed to create software systems. The class diagram is the main building block of object oriented programming. A class diagram is a box that is split into 2-3 sections. The top section of the box contains the name of the class, which is typed in Bold and the first letter of each word is a capital letter e.g. **ClientDetails**. The second section of the class contains the attributes needed to design the software. The attributes are left-aligned and letters are typed in lowercase. The third section of a class diagram will show what methods will be used while developing the system.





In the first section of the above class diagram, the initial process of the car hire system is shown. The new Client will have to supply details to the company and make a payment. As you can see, the association between the two classes is the client making the payment. Once the transaction has been made the car will be hired to the customer. In the client class, a number of attributes are being used with the data type string; these are Name, Address, Postcode, Driving Licence Number and E-mail address. The reason for using this as a data type is the data will consist of letters and numbers. The attributes Age, Member ID, Delivery Date, and Collection Date are using the data type integer as only numbers are required.



The above diagram demonstrates the next process of the proposed system for the car hire company. The clerk will deal with all booking requests from clients and will authorise the hire of the car. In the clerk class the following attributes and (data types) have been chosen as they are suitable for the new computer based system.

- CustomerInfo (String)
- BookingRequest (String)
- Car Availaibility (String)
- Invoice Details (Double

The Hire class contains the following attributes: Car, HireDate, ReturnDate and HireRate.
A Staff Class has also been created which will hold the details of the clerk and mechanic. The following attributes and (data types) have been used for the Staff class.

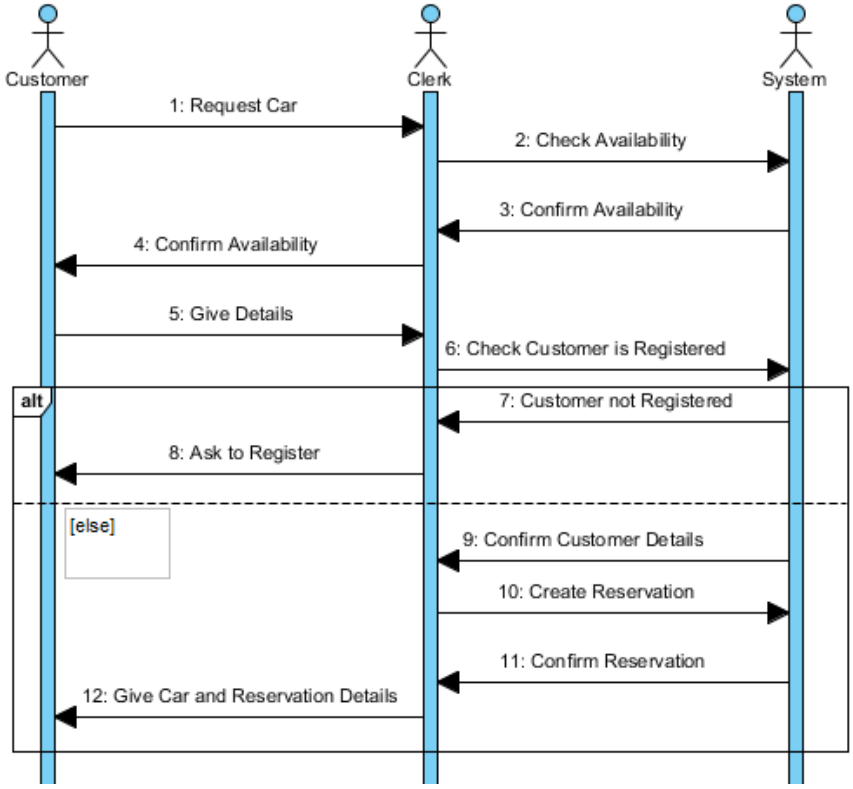
- Name (String)
- Address (String)
- StaffID (Int)

The hire class will also contain the details of each car as the hire rate will be dependent on what type of car is hired by the client. When the car is returned by the client, the actual mileage of the car is recorded and compared to the estimate and is there is a difference this will be updated and the client will need to make a cash payment.

Part 7

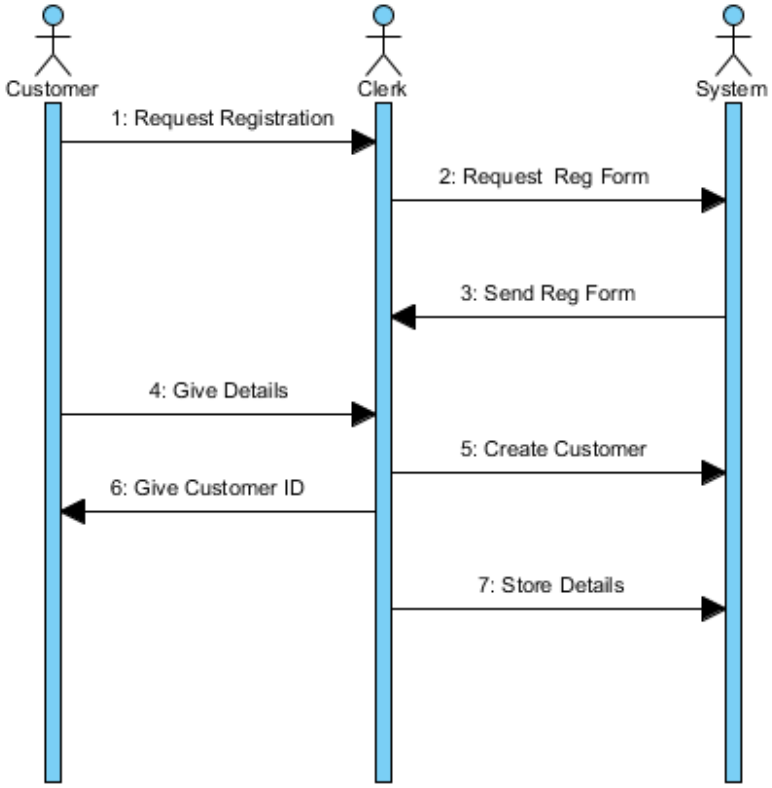
System Sequence Diagrams

The following diagrams are System sequence diagrams of the essential use cases that we identified earlier on in the project. These diagrams help to illustrate the events that the actors generate, the orders that they occur in and any inter-system events for an individual use case. All of these diagrams have been created using Visual Paradigm.
Register Car Hire:



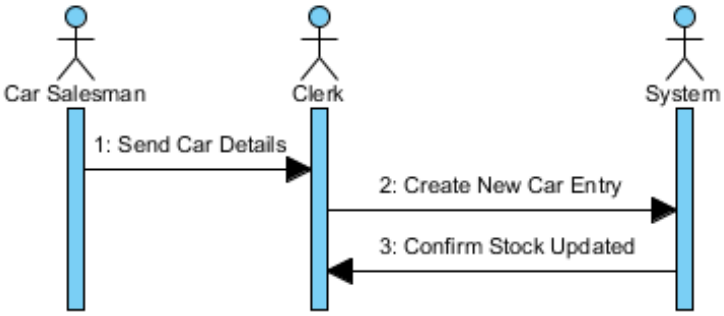
This sequence diagram was created for the Use Case: Register Car Hire. This was agreed to be one of the essential use cases of the design as the whole business is focused around this interaction. This diagram contains 3 actors, Customer, Clerk and System. The actor 'Clerk' interacts with both 'Customer' and 'System' however 'Customer' and 'System' never interact. This is an intentional design choice as 'Clerk' is the only actor in this scenario who will have access to the system. The sequence is initiated by 'Customer' requesting a car from 'Clerk' who then has to access the 'System' to check the availability of said car. This is necessary as otherwise the 'Clerk' may start to create a booking then find out part way through the required car is unavailable which would be a waste of both time and money. If the car is available this information is shown to the 'Clerk' who then relays it to the 'Customer', after which the customer is asked to give his details. At this point the diagram splits into two paths, if the 'Customer' is not registered then they will be asked to register which would start the 'Customer Registration' use case. If the 'Customer' is already registered then their details are confirmed and a reservation is created. The confirmation details are sent from the 'System' to the 'Clerk' then relayed to the 'Customer' which ends the sequence.

Register Customer:



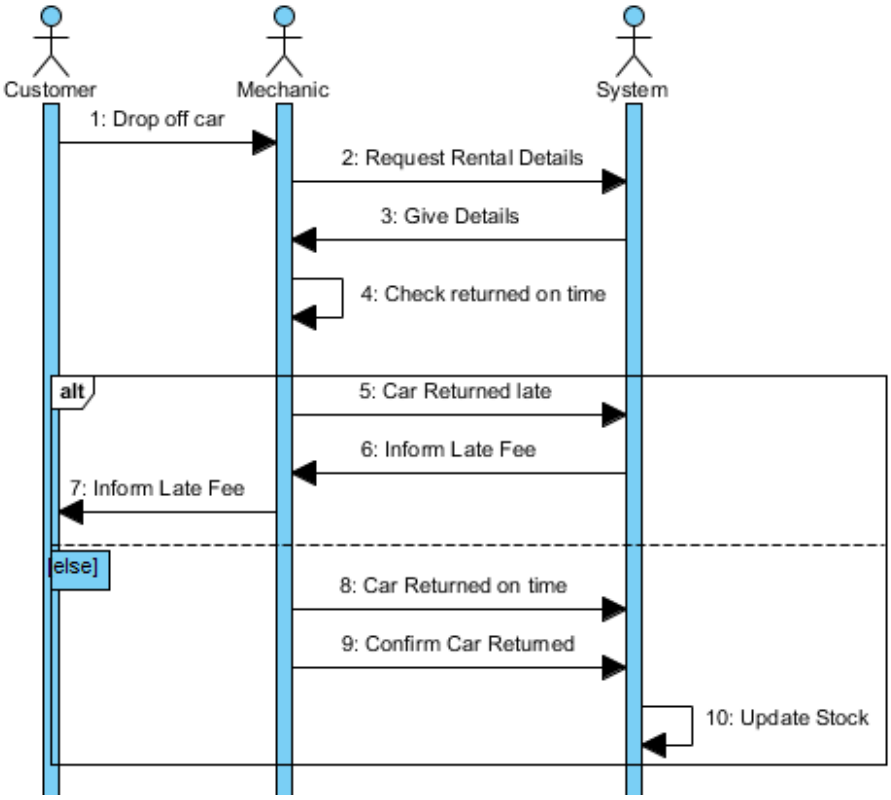
This sequence diagram was created for the Use Case: Register Customer. This was also a potential off shoot of the previous diagram if the customer was not registered when they wished to rent a car. As such it uses the same actors from the 'Register Car Hire' diagram and they each have the same level of interaction with each other. This was considered an essential use case as without being able to register a customer a hire cannot be completed as the company will have no details of the customer stored. This sequence is initiated by the actor 'Customer' requesting a registration whether this is by choice or necessity for completing a hire. The 'Clerk' then requests a registration form from the system which is then displayed for them an improvement over the existing system where they would be required to find a paper form. The 'Customer' then gives them their details which the 'Clerk' will enter into the form to create an account. When an account is created a Customer ID will be generated in order to keep each account unique, this ID is then given to the 'Customer'. Finally the account details are then stored within the system, again an improvement over the existing system as all records are stored digitally for increased security and easy access.

New Car Registration:



This sequence diagram was created for the Use Case: New Car Registration. This use case was considered essential as without this operation no new car stock could be recorded within the system meaning they could not update their stock. This diagram does not use the actor ‘Customer’ but instead uses ‘Car Salesman’ who is a unique actor for this use case. The sequence is initiated when the ‘Car Salesman’ sends over the details of the new car to the ‘Clerk’, this is done separately to the actual delivery of the new car ahead of time in order to be as efficient as possible. The ‘Clerk’ will then use the ‘System’ to either create a new car entry or update the stock total of an existing car. The ‘System’ will then confirm the stock has been updated to the ‘Clerk’.

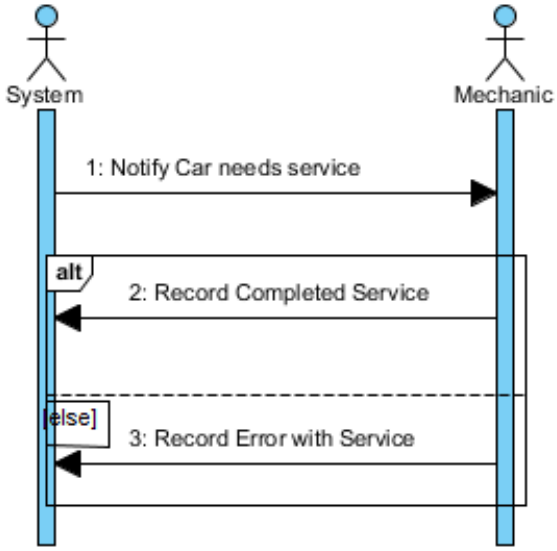
Car Return:



This sequence diagram was created for the Use Case: Car Return. This use case was considered essential as it is a key part of the hire process and

without it there would be no way of recording which cars had been returned and if they were on time or not. The actors used are ‘Customer’, ‘Mechanic’ and ‘System’. The sequence is initiated by the customer dropping off the car for the mechanic who then requests the rental details from the system in order to compare with the customers, this adds an additional level of security to the process. The ‘Mechanic’ then checks whether the car has been returned on time as outlined in the hire agreement and the diagram splits into two paths. If the car has been returned late the ‘Mechanic’ notifies the ‘System’ of this, which in turn informs him of the required late fee. The ‘Mechanic’ will then inform the customer of this and the car will be returned. If the hire care has been returned on time the ‘Mechanic’ confirms the return in the ‘System’ and the stock is updated.

Car Service:



This sequence diagram was created for the Use Case: Car Service. This use case was considered essential as cars are serviced every 6000 and 12000 miles and servicing is an integral part of the company’s day to day operations. Despite it being an essential use case it is a fairly simple diagram involving only the ‘System’ and ‘Mechanic’ actors. The sequence is initiated when a car has tracked 6000 or 12000 miles and the ‘System’ alerts the ‘Mechanic’ that a service is required. Then there are two paths, either the service is completed and the ‘Mechanic’ logs this with the ‘System’ or there is an error and the service cannot be completed in which case the ‘Mechanic’ still logs this with the system.

Part 8

System contracts

System contracts are essentially a list and specifications of what a certain function should do. In our examples we have register customer which is basically what happens when you register a new customer and there are some different attributes such as the name of the contract this can include parameters, the responsibilities of said contract, this just what the contract is responsible for, then we have cross references this is just if it has any reference with something else in the design such as use cases. Exceptions are what should produce an error message. The next two are important and they are the pre-conditions and the post-conditions, the pre-conditions are what we should have prior the post-conditions such as if you need certain details or to check if something exists. Post-conditions are what the part of the contract is supposed to achieve.

Register Customer:
Name: CreateCust
Responsibilities: Enter a customer's details and add to system. Generate customer ID.
Cross references: Use Cases: Register Customer
Exceptions: If any field is invalid generate an error
Pre-conditions: Customer Details needed, Registration Form Created
Post-conditions: Customer registered
Name: StoreCust
Responsibilities: Store the newly created customer in the system
Cross References: Use Cases: Register Customer
Exceptions: If no new customer created indicate an error
Pre-Conditions: Customer account must be created
Post-Conditions: Customer.Stored set to true

This contract registers a customer and has 2 parts to it the first part creates the customer which takes the details and adds it to the system then generating a customer id, if any field is invalid it will generate an error message, if all is successful this part will register a customer.

The second part stores the customer into the system, it needs to check that the customer account exists and if it does it sets a variable to true.

Car hire:
Name: CreateRes
Responsibilities: Enter Customer details to create car reservation
Cross References: Use Case: Register Car Hire
Exceptions: If customer is not registered indicate an error
Pre-Conditions: Customer.Stored = True, Car.CarAvailable = True
Post-Conditions: Reservation Created

In this contract it controls the car hire for when someone wants to hire a car, it needs the customers details but if the customer is not registered it will have an error, it also needs to check if the car is available and if the customer is stored into the system if this is correct then the reservation gets created.

Registration of new car:
Name: CreateCar
Responsibilities: Enter Car details to create new car object.
Cross References: Use Case: New Car Registration
Exceptions: If no new car details indicate an error
Pre-Conditions: none
Post Conditions: New Car object created

In this contract it registers a new car to the system which takes the car details to create a new car object, if there isn't a new car ready to be added it will create an error, when completed it will create a new car object.

Hire Car Return:
Name: ReturnCar
Responsibilities: Confirm return of hire car
Cross References: Use Case: Car Return
Exceptions: If return date exceeds Hire.Returndate then indicate error
Pre-Conditions: Car.CarAvailable = False, Mechanic.CarDetails = True
Post Conditions: Car.CarAvailable = True, Car.Mileage = Mileage

This contract controls the returning of cars and confirms if It has been returned this will need to check if the car is available and if the car details are correct, it will produce an error if the date exceeds the specified return date, if no error the car gets made available and it logs the mileage.

Car Service:
Name: ServiceCar(String ServiceType)
Responsibilities: Confirm that car has been serviced
Cross References: Use Case: Service Car
Exceptions: If system has not notified mechanic, no service needed
Pre-Conditions: Mechanic.ServiceType = (ServiceType), Car.Mileage = +6000 || + 12000 since last service
Post Conditions: Car.Dateofservice = Date, Car.MechanicResponsible = Staff.Name

In this contract it controls if the car has been serviced and takes 1 parameter which is the service type, there will be an error if the system has not notified a mechanic because there will be no service needed, the car will be checked every 6000 or 12000 miles since last service in the end it will log the date of the service and the mechanic responsible for the service.

Part 9

Object sequence diagrams

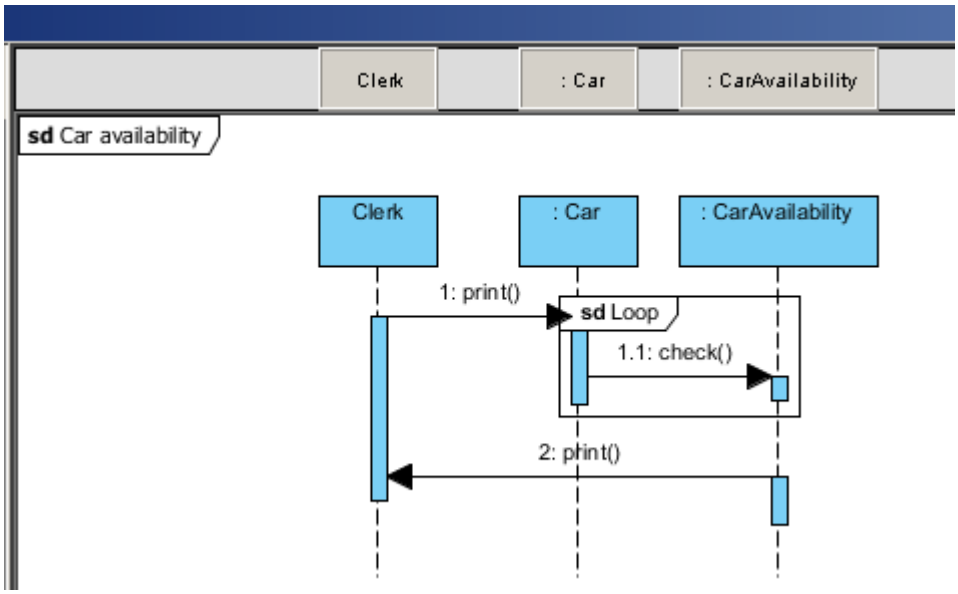
An object sequence diagram is an altered version of sequence diagram, the sequence diagrams show interactions between objects in time sequence as we have shown earlier in the report . An object sequence diagram takes a particular section of the system and shows the interactions within that.

Within this car hire system there many functions that are processed for example: car availability, create new car and car return. An object sequence diagram can be created for each of these, the diagram will show how each process is taken out within the system and the order in which it is done in. The object sequence diagram will also show which object is used within that part of the system and its involvement.

Below are the created system sequence diagrams from/for the new system for the car hire business, a diagram has been created for each essential part of the system showing the interactions between the objects within the system for that specific job.

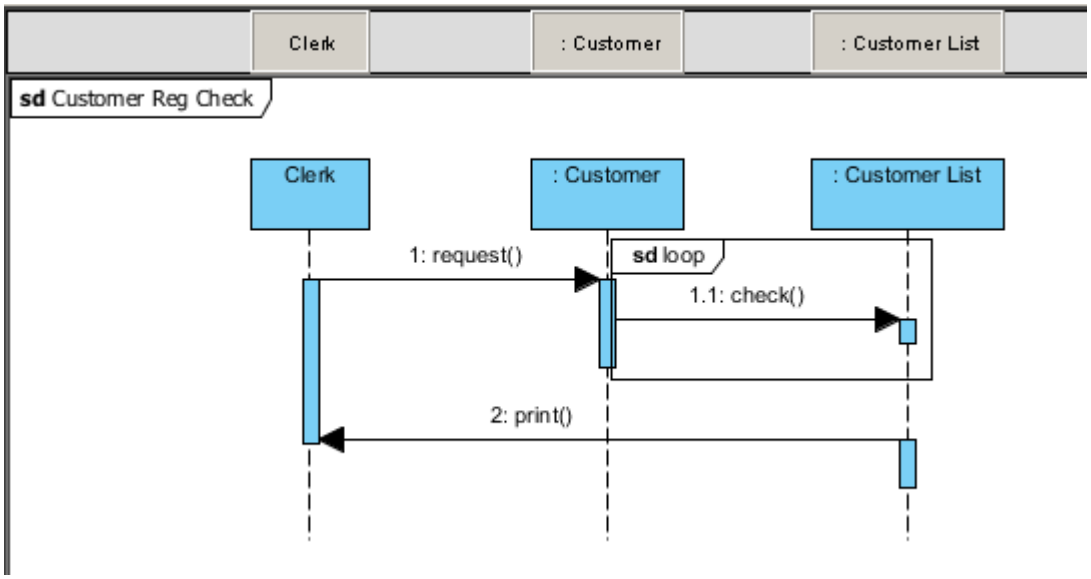
Car availability

This object sequence diagram shows how the car availability is checked within the system, as shown in the diagram the objects involved are the: clerk, car and car availability. The clerk uses the car availability information which is within the car details to check the car availability.



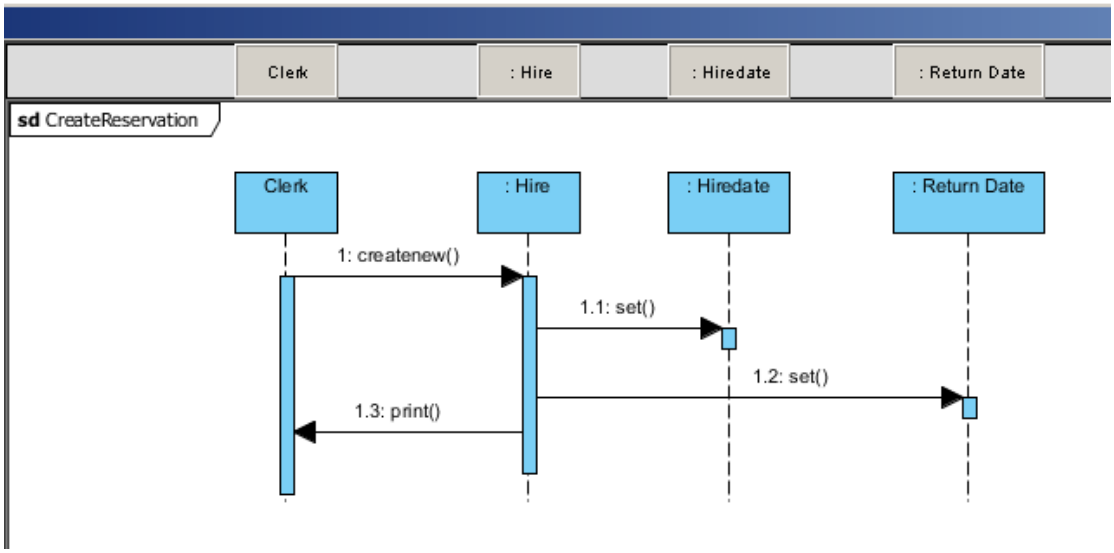
Customer reg check

This object sequence diagram shows how the customer's registration is checked, this sequence diagram uses the objects: clerk, customer and customer list. The clerk uses a request form which checks the customer and customer list within the system, the answer/reply is then printed back to the clerk as shown in the diagram below.



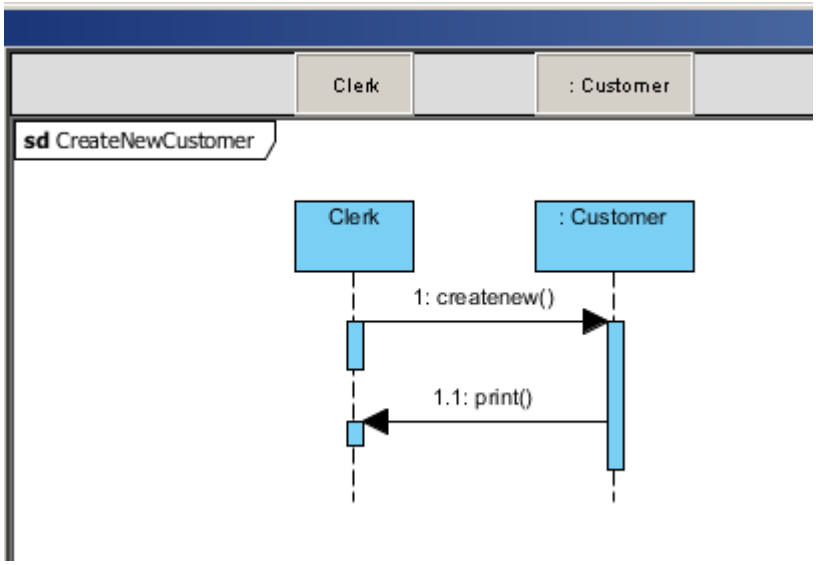
Create reservation

This object sequence diagram shows how a reservation is created within the system. This diagram uses quite a lot of objects, it uses the objects: clerk, hire, hire date and return date. This diagram has quite an easy route as it doesn't need loops the whole diagram is pretty straight forward , the clerk sets the : hire , hire date and return date then created the reservation , the system then prints the reservation back to the clerk .



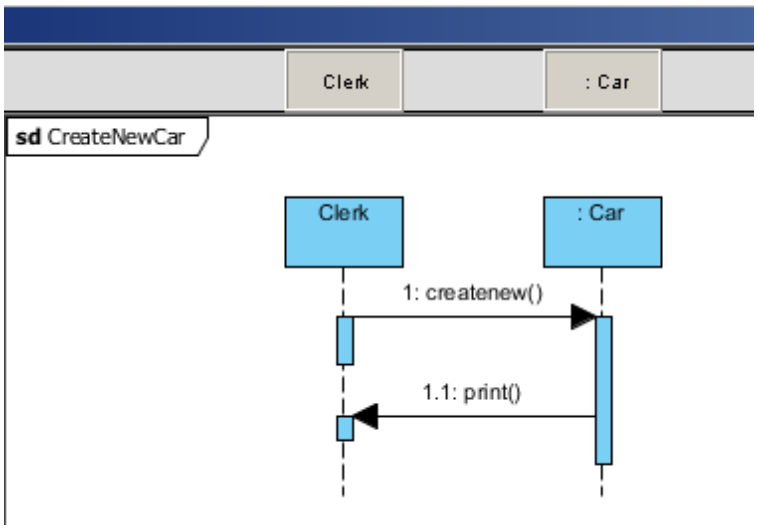
Create new customer

This object sequence diagram shows how to create a new customer, this diagram is quite simple the objects used are the clerk and the customer. The clerk requests the information from the customer which is given back to the clerk who then logs it to create a new customer within the system.



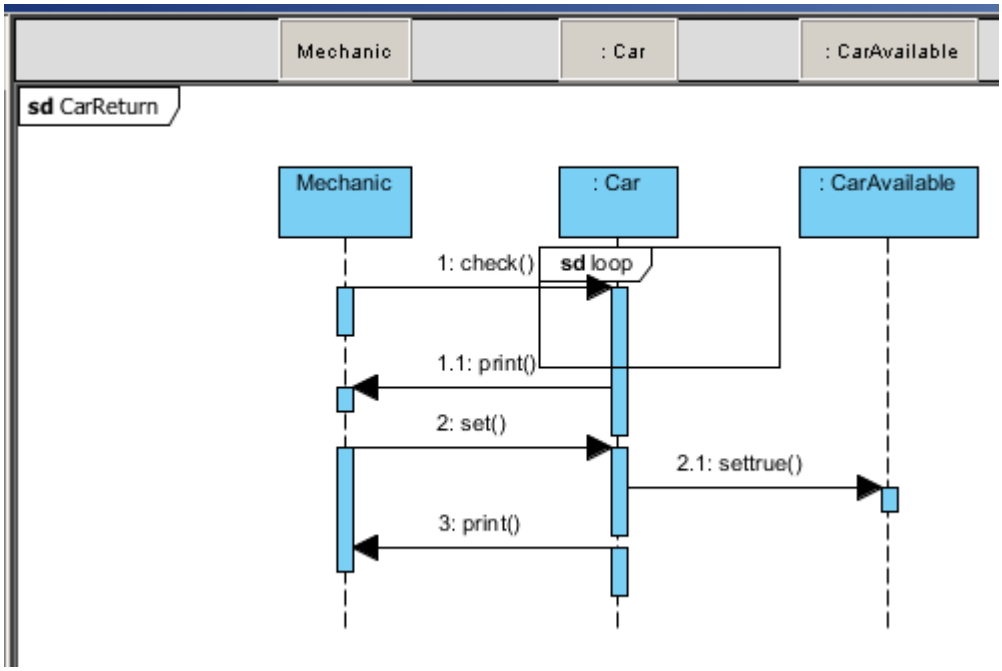
Create new car

This object sequence diagram shows how a new car is created, this is very similar to the create new customer diagram except the object used are obviously clerk and car. The clerk collects and adds all the details for the new car into the system which is then saved to create a new car within the system.



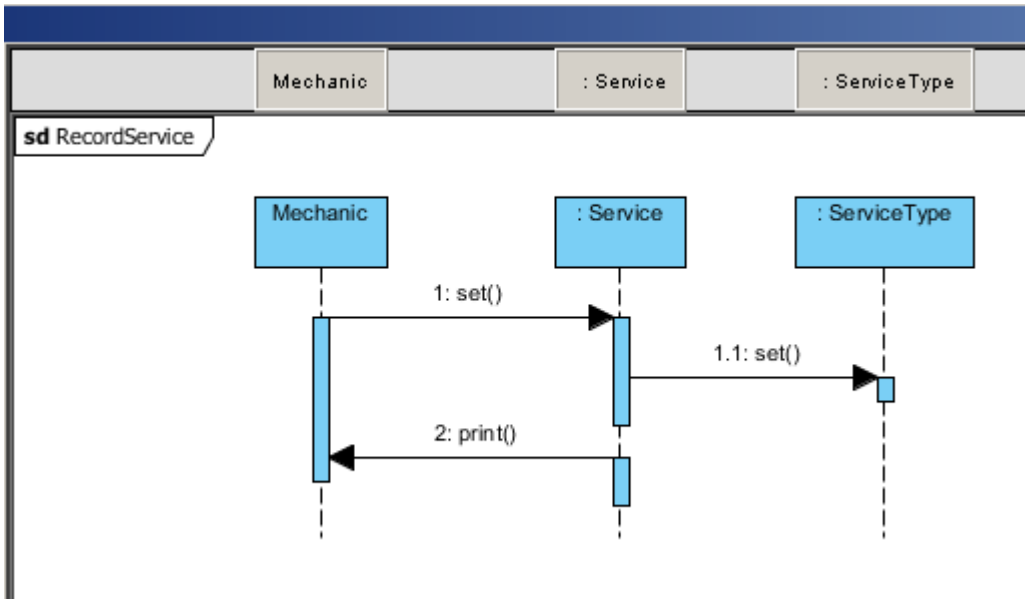
Car return

The next object sequence diagram is for the car return section of the system. The object involved within this diagram are: mechanic, car and car available. The mechanic checks the details of the car within the system, the car availability is within the car details this is then sent back to the mechanic. Not only can the mechanic check the available cars but the mechanic can set the availability of the car by setting the car details within the system which changed the car available option within the car object.



Record service

The final object sequence diagram is for the record service, this diagram uses the: mechanic, service and service type objects within the system. As shown in the diagram the mechanic sets the service and the service type (minor or major) which have been done on the car this is then confirmed to the mechanic once done via print.



Part 10

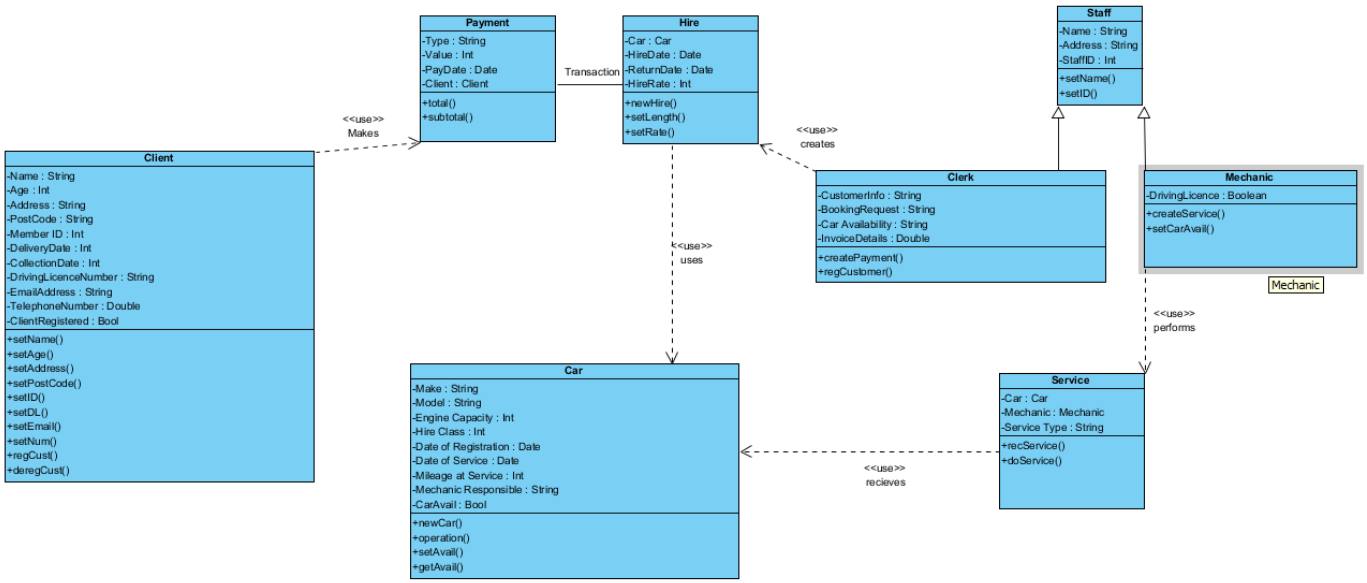
Patterns

Throughout the design several patterns have been identified and used in the creation of the diagrams. Patterns are the general principles used to guide the creation of software and each of them describe a problem to be solved and a solution to the problem.
The first type of pattern used was the 'Expert' pattern. This pattern is assigning a responsibility to a class that has the information necessary in order to fulfil the responsibility. This is evident in our work in several classes in our class diagram. For example in order to complete a hire the 'Hire' class gets information from the 'Car' class attributes such as availability and hire class. Without these a hire could not be completed as the required details would not be available.
Another pattern used was the 'Creator' pattern. This pattern is when one object is responsible for the creation of another object. Again like the 'Expert' pattern this is used frequently within our designs. For example the 'Client' class creates the class 'Payment'. Without the 'Client' class 'Payment' would not exist and there would be no way to complete the hire transaction. In a similar way to this the 'Hire' class would not be created without the 'Clerk' class.
The 'Low Coupling' pattern has also been used. In this pattern the object is to assign responsibility but with the lowest possible coupling. In our designs this describes the interaction between the 'Client' and 'Payment' classes. 'Client' is only responsible for the creation of the 'Payment' class which reduces the number of dependencies on 'Client. In contrast to this the 'High Cohesion' pattern is assigning responsibility whilst cohesion remains high. This is shown in our diagrams with the 'Hire' class which is responsible for the use of the 'Car' class and the 'Payment' class after being created by the 'Clerk' class.
Finally the 'Controller' pattern has also been used. An example of its use is the 'Car' class. This is because when a new car is registered it has to accept input from external actors. It is also representative of the business as a whole because without the 'Car' class the business itself could not exist.

Part 11

Design class diagram

We previously had a conceptual class diagram which was explained previously. This is the design class diagram it is similar to conceptual class diagram but has methods and operations.



Each of the classes has methods and the client class has these methods

Method: setName()
Purpose: Sets the name of the client.

Method: setAge()
Purpose: Sets the age of the client.

Method: setAddress()
Purpose: Sets the address of the client.

Method: setPostcode()

Purpose: Sets the postcode of the client.

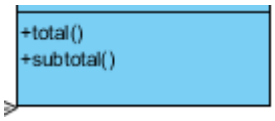
Method: setID()
Purpose: Sets the ID of the client.

Method: setEmail()
Purpose: Sets the E-Mail of the client.

Method: setNum()
Purpose: Sets the number of the client.

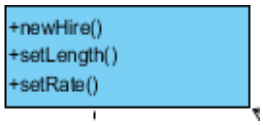
Method: regCustomer()
Purpose: Registers the customer into the system.

Method: deregCustomer()
Purpose: deregisters the customer from the system

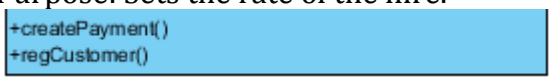


Method: total()
Purpose: Calculates the total of the payment.

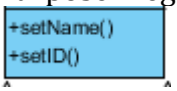
Method: subtotal()
Purpose: Calculates the subtotal of the payment.



Method: newHire()
Purpose: Creates a new hire object.
Method setLength()
Purpose: Sets the length of the hire duration.
Method setRate()
Purpose: Sets the rate of the hire.



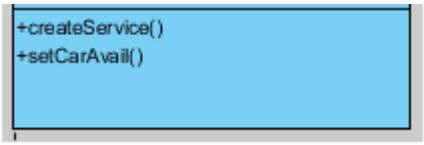
Method: createPayment()
Purpose: Creates a payment object.
Method: regCustomer()
Purpose: Registers the customer to the system.



Method: setName()
Purpose: Sets the name of the staff.

Method: setID()

Purpose: Sets the ID of the staff.



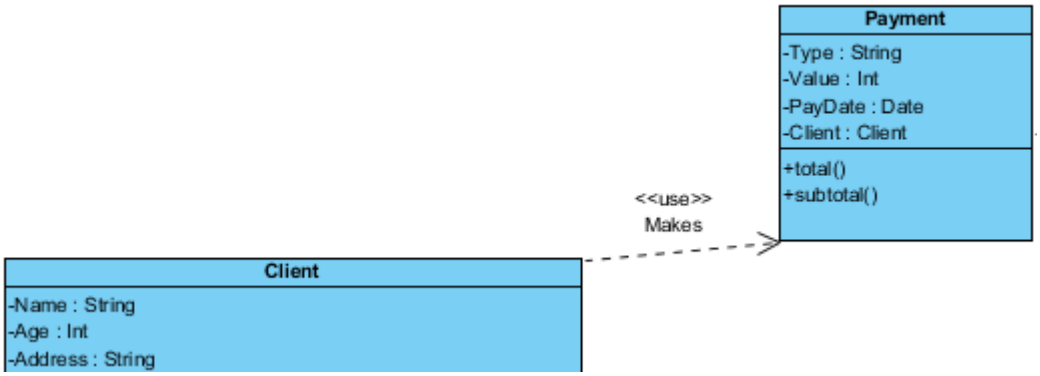
Method: createService()
Purpose: Creates the service check.

Method setCarAvail()
Purpose: Sets the CarAvailable variable to true in the Car class.

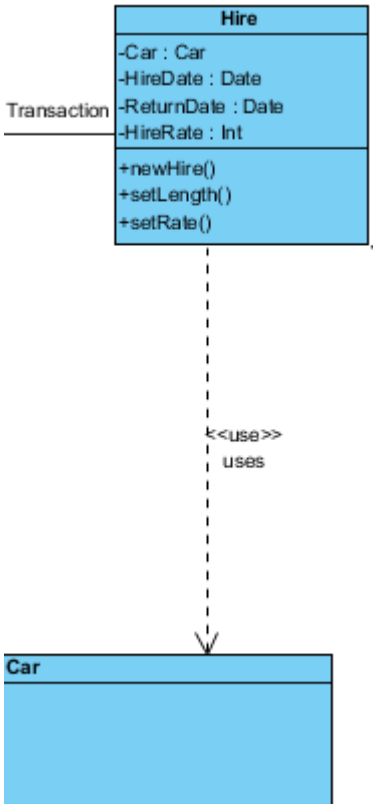
Method: recService()
Purpose: Records the service and what happened.

Method: doService()
Purpose: Does the service to the car.

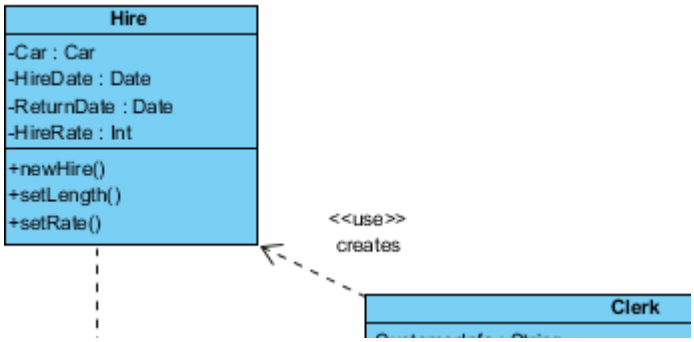
Method: newCar()
Purpose: Registers new car into the system.
Method: setAvail()
Purpose: Sets the car to available.
Method getAvail()
Purpose: Gets if the car is available or not.



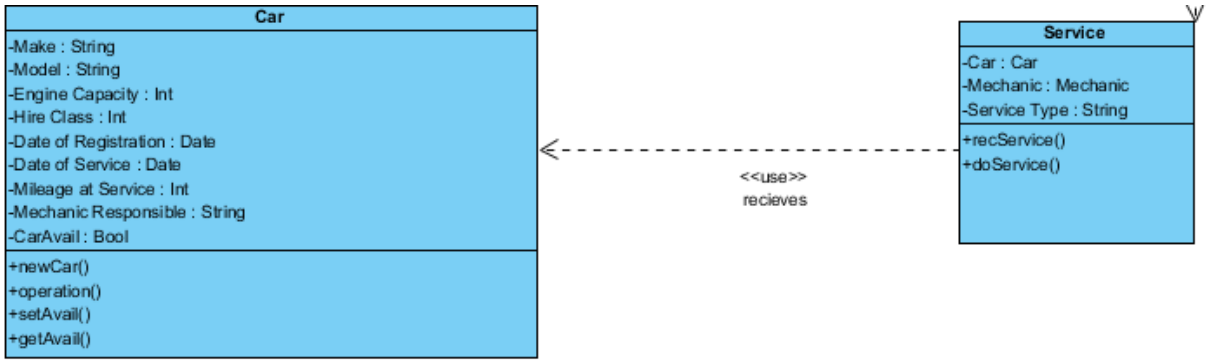
The client makes the payment which is shown on the design class diagram.



Hire class uses the car class to determine what car is going to be used.



The clerk uses the hire class to allow the customer to hire a car.



The service class uses the car class to determine what car and type the service will be.

Conclusion

We have successfully used extensive software design tools to completely change the car hire system. The system had many problems as a paper based system which have addressed using various different techniques, this has led to the new to the new computer based system which we have shown all the diagrams and write up for in the above report. We have worked as a five man team to create this new system combining individual work and combined meetings to complete the work load.

Team diary

Meetings

Week	What was done	Attendance
1	Overviewed the coursework requirements then discussed each section	Rowan , Mike , Samuel ,Josh ,Ashley
2	Created a rough template for parts 1 and 2 in a note format on word	Rowan , Mike , Samuel ,Josh ,Ashley
3	Started on the use case diagrams for the new system	Rowan , Mike , Samuel ,Josh ,Ashley
4	Started on the class diagrams for the new system	Rowan , Mike , Samuel ,Josh ,Ashley
5	Started on the system sequence diagrams	Rowan , Mike , Samuel ,Josh ,Ashley
6	Created the contracts based on the new system	Rowan , Mike , Samuel ,Josh ,Ashley
7	Split up the write up into sections for each group member	Rowan , Mike , Samuel ,Josh ,Ashley
8	Put all the documentation and the diagrams together plus finalized the coursework	Rowan , Mike , Samuel ,Josh ,Ashley

Individual input

Name	Individual input
Rowan Bloomfield	Attended all group meetings plus done the write up for parts 1 and 2 , finalized the initial brief and the system functions also put all the documentation together for the final upload
Mike O'Keefe	Attended all meetings plus done the final write up for part 7, also finalized the system sequence diagrams. Created most of the diagrams throughout the coursework using UML
Samuel Jervis	Attended all meetings plus done the final write up for parts: 3, 4 and 5. Finalized the use case diagrams for the final documentations
Josh Allen	Attended all meetings plus done the final write up for parts 8 and 11. Created all the contracts based on the use cases previously created also finalized the design class diagrams for part 11.
Ashley Hussey	Attended all meetings plus done the final write up for 6. Finalized all the class diagrams for the final documentation.