Creating Sequence Diagrams in EA

In order to complete this work you will need partially completed design documents like so.

Detailed Description

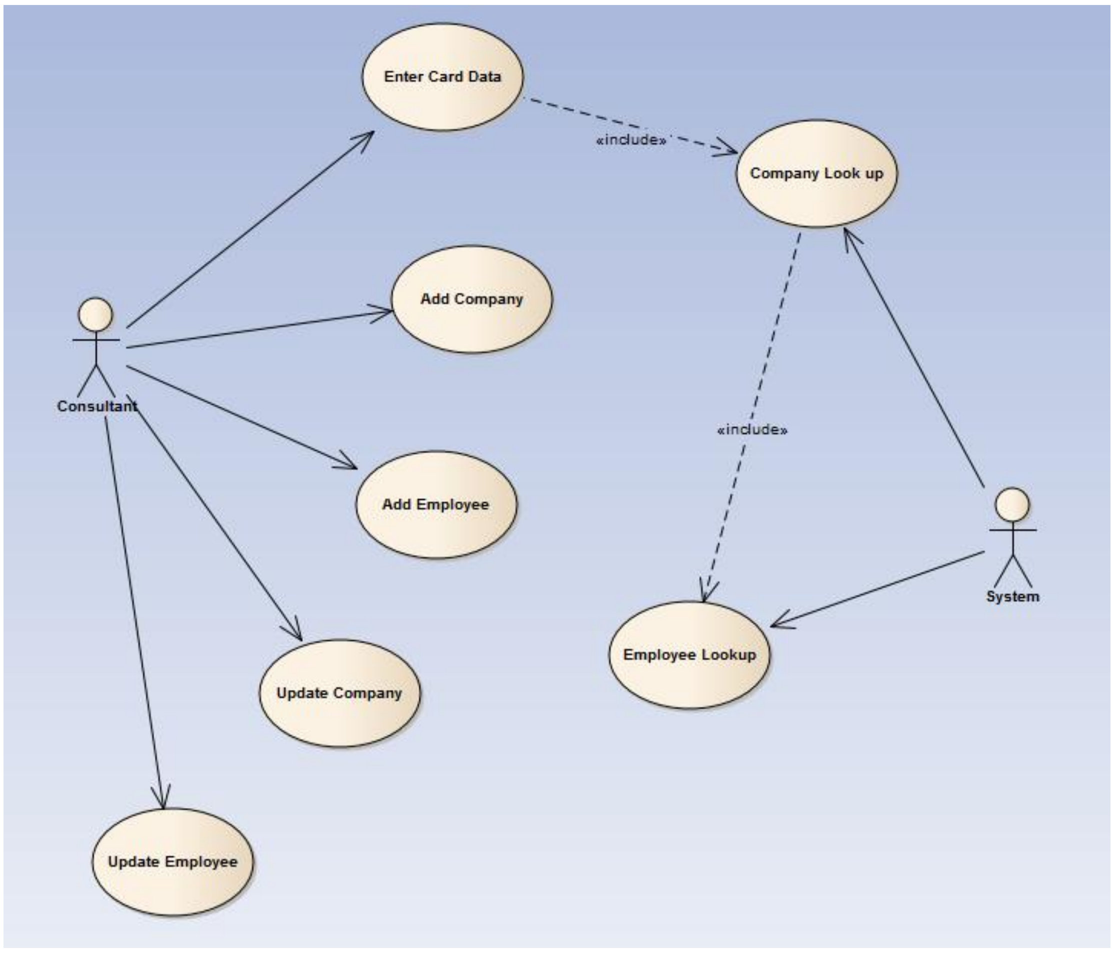
> *The consultant sits at their desk with a stack of business cards and flyers. They pick up a business card and start to input the details into system. The user is interested in recording details of the individual and associated company. The first field they enter is the name of the company. While doing this the system looks up the company name to see if it is already on the system. The next field the user enters is the name of the contact. Whilst dong this, the list of contacts for that company is displayed such that if the contact is already on the system the user may move onto another business card. At this point the user should have the opportunity to update the details on the system should they note that some aspect has change e.g. email address.*

Event Table

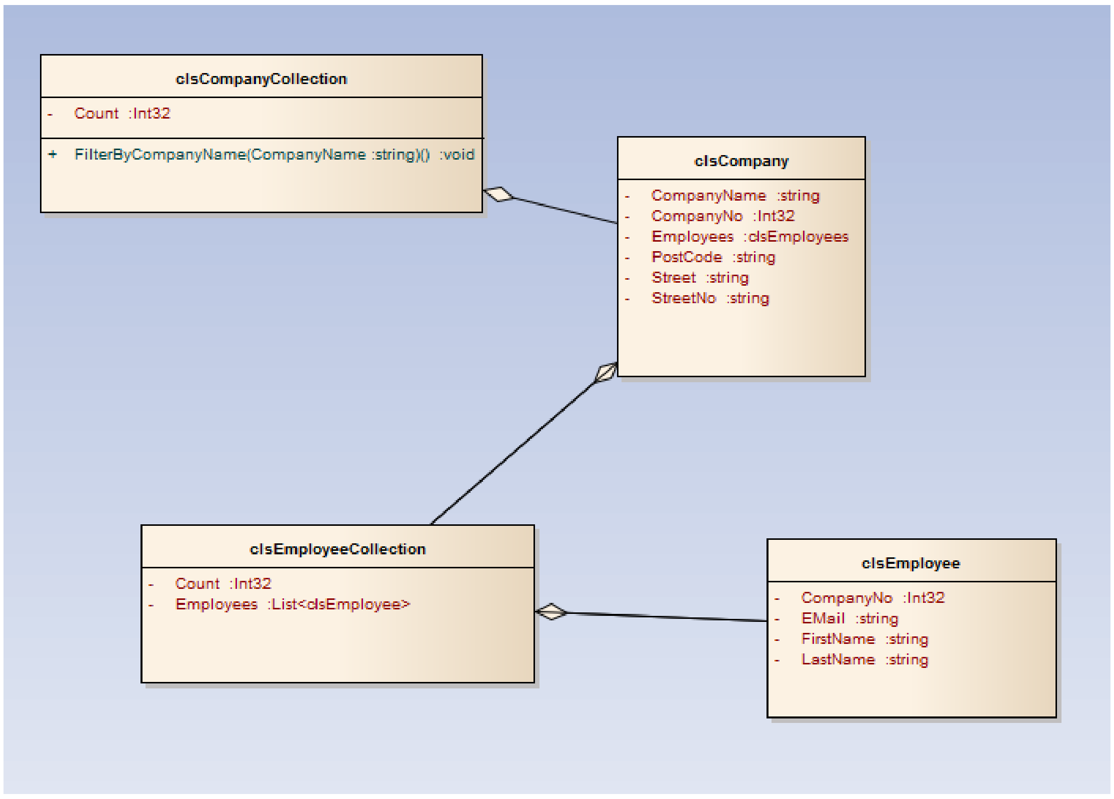
Subject	**Verb**	**Object**	**Response**
Consultant	Inputs	Company Name	System produces a list of companies with that name
System	Finds	Company	Produces a list of companies with that name
System	Lists	Employees	Produces a list of employees at each company
Consultant	Adds	Company	Company added to the system
Consultant	Adds	Employee	Employee added to the system
Consultant	Updates	Employee	Employee details updated
Consultant	Updates	Company	Company

			details updated

Use Case Diagram



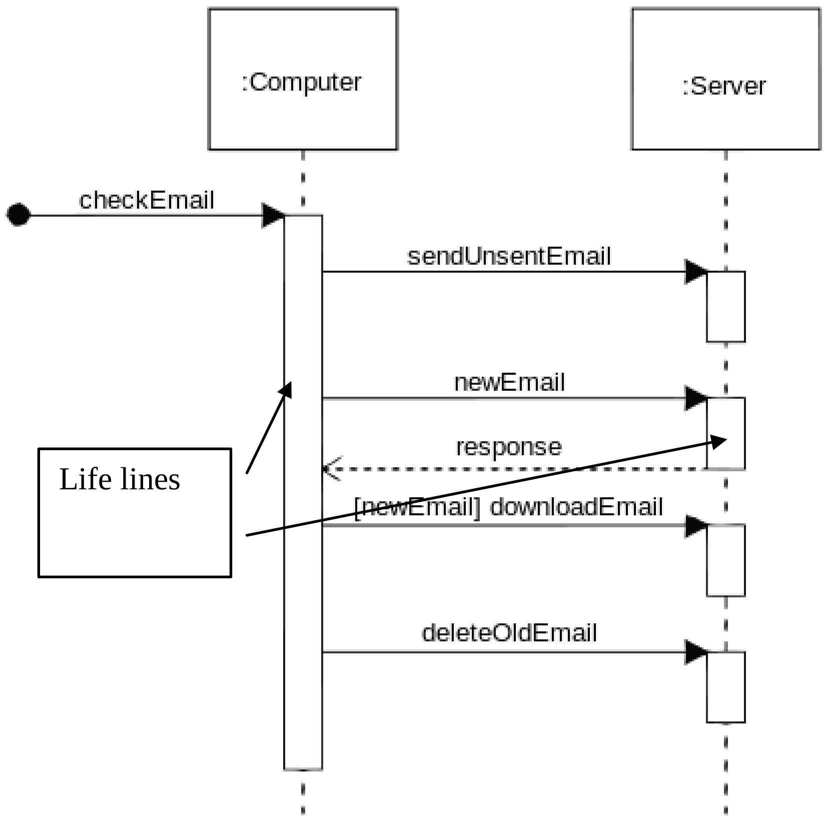
Class Diagram


clsCompanyCollection
- Count :Int32
+ FilterByCompanyName(CompanyName :string)() :void
clsCompany
- CompanyName :string
- CompanyNo :Int32
- Employees :clsEmployees
- PostCode :string
- Street :string
- StreetNo :string
clsEmployeeCollection
- Count :Int32
- Employees :List<clsEmployee>
clsEmployee
- CompanyNo :Int32
- EMail :string
- FirstName :string
- LastName :string

The Sequence Diagram

The sequence diagram allows us to think about how the outcomes of the use cases map onto the classes that we plan on using.

For example the following diagram describes the use case of checking an email. (Wikipedia)



The flow of time is indicated by the vertical life lines.

The messages between objects are indicated by the horizontal lines.

The solid arrow heads indicate a synchronous call meaning that this message must complete before the next message may be sent.

For example in this diagram sending unsent mail must complete before sending a new message.

The open arrow heads indicate asynchronous messages. In this case the message may be sent without waiting for the previous message to complete.

For example incoming emails may be received even if there is a message being sent.

The solid lines indicate outgoing messages to objects whilst a dashed line indicates a reply of some sort from the object.

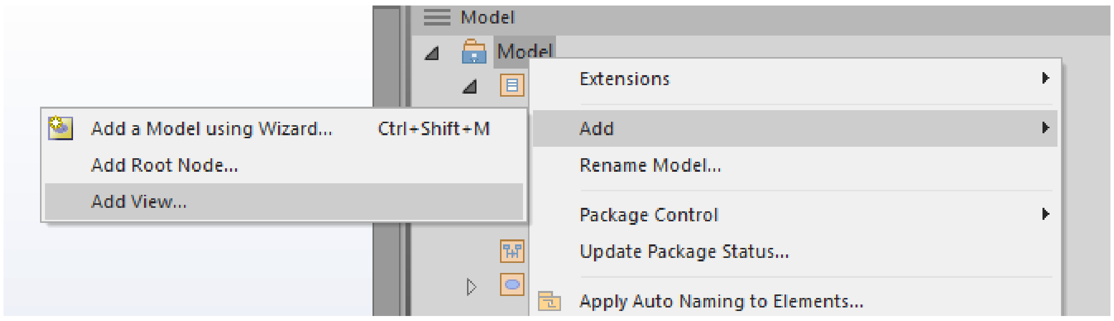
What is the Point of Sequence Diagrams?

Sequence diagrams are a very useful tool for spotting problems with your system design.

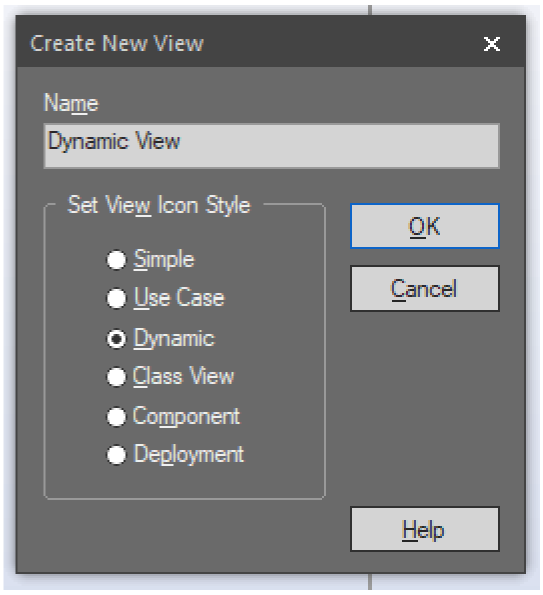
You start with two primary diagrams your class diagram and use case diagram, Using these two documents as the starting point you create a third diagram, the sequence diagram which joins the first two together. If the mapping between the three diagrams looks right then it is a useful indicator that your design is going the right way. If there are gaps in your design then the sequence diagram will help you spot them and fix them.

Creating the Dynamic Model View

If you haven't done so already create the dynamic model view in Enterprise Architect by right clicking on the model and adding a new view…



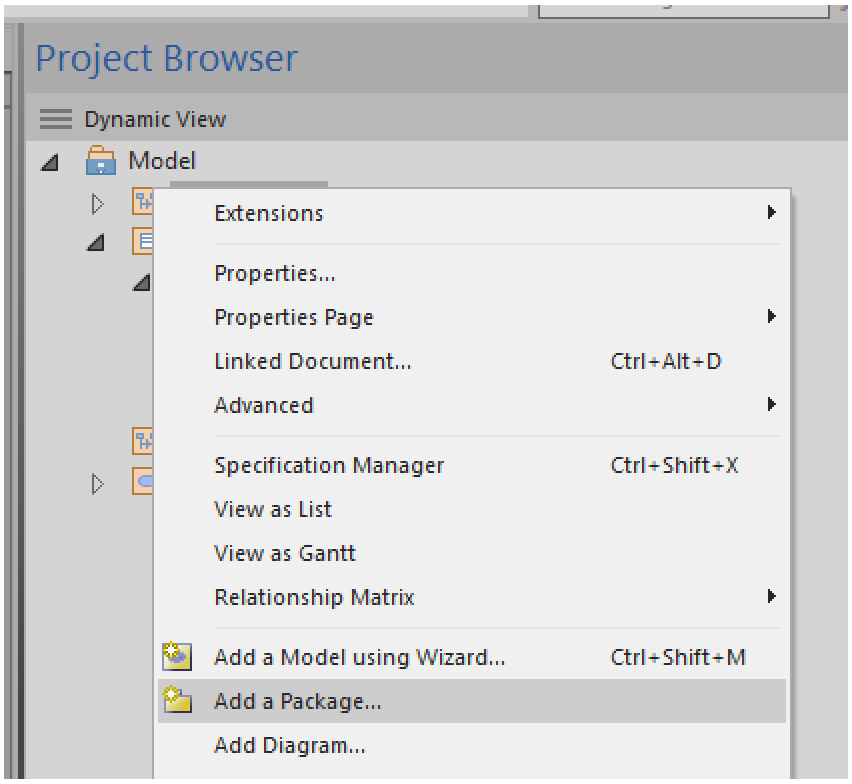
Set the view up like so…


Create New View

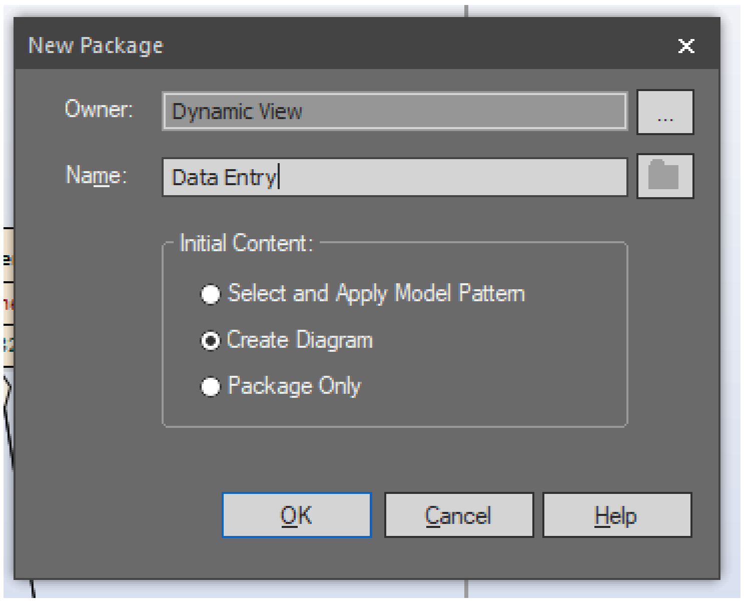
Name: Dynamic View

Set View Icon Style
- Simple
- Use Case
- Dynamic
- Class View
- Component
- Deployment

Add to the new view a package to store the diagrams in…



Set up the package like so…


New Package

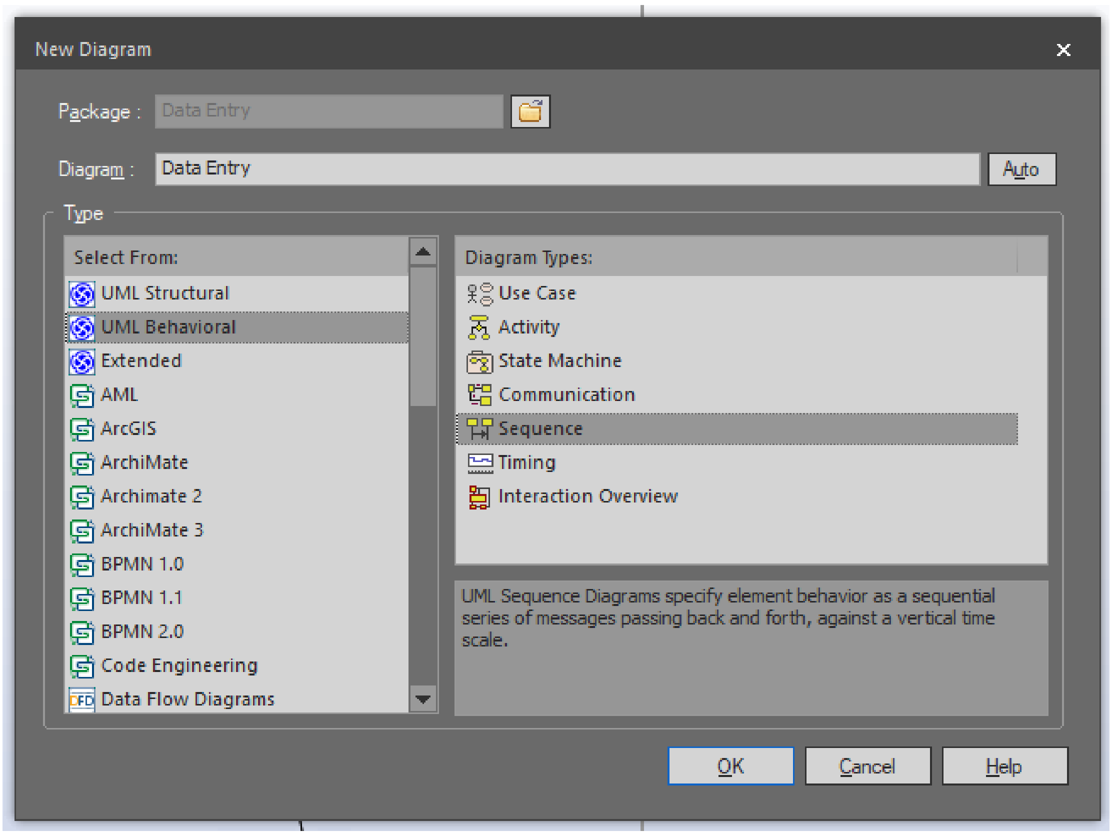
Owner: Dynamic View

Name: Data Entry

Initial Content:

- Select and Apply Model Pattern
- [✓ Create Diagram]
- Package Only

Under UML Behavioral select the sequence diagram option…



New Diagram

Package : Data Entry

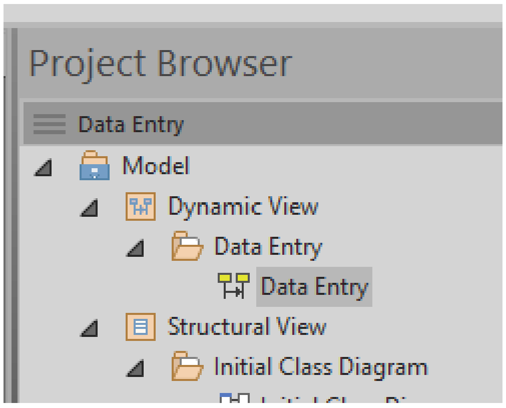
Diagram : Data Entry

Type

Select From:	Diagram Types:
UML Structural	Use Case
UML Behavioral	Activity
Extended	State Machine
AML	Communication
ArcGIS	Sequence
ArchiMate	Timing
Archimate 2	Interaction Overview
ArchiMate 3	
BPMN 1.0	
BPMN 1.1	
BPMN 2.0	
Code Engineering	
Data Flow Diagrams	

UML Sequence Diagrams specify element behavior as a sequential series of messages passing back and forth, against a vertical time scale.

This will create the sequence diagram in the project browser…



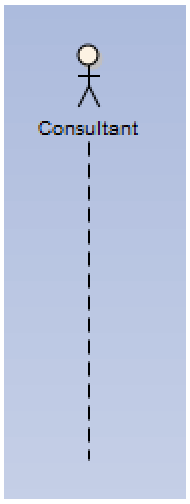
Project Browser

Data Entry

- Model
  - Dynamic View
    - Data Entry
      - Data Entry
  - Structural View
    - Initial Class Diagram

Double click the diagram to edit it.

The first thing we will add to the diagram is the actor Consultant taken from the use case diagram…

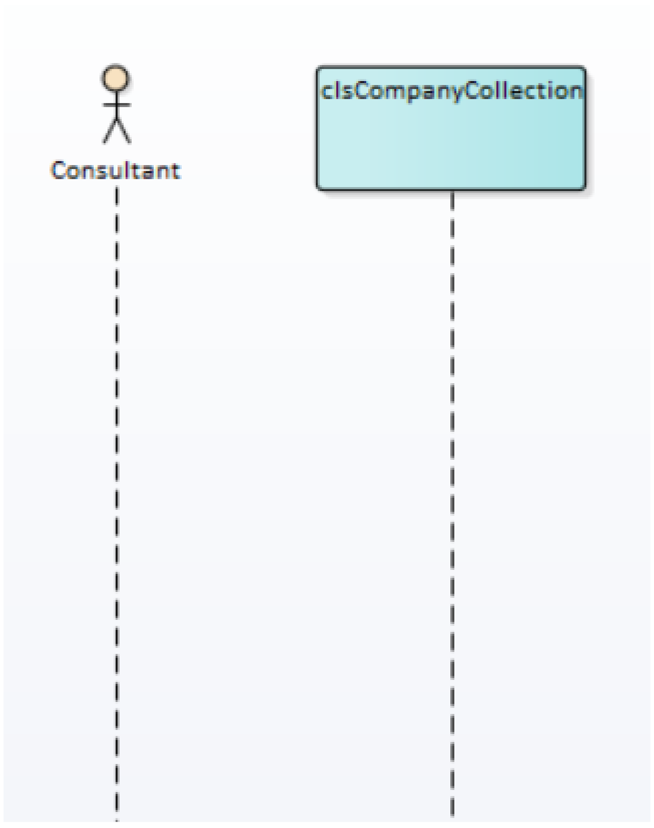


This is the actor in this case that will trigger the sequence of events.

We now need to think about what is the first action the actor will perform.

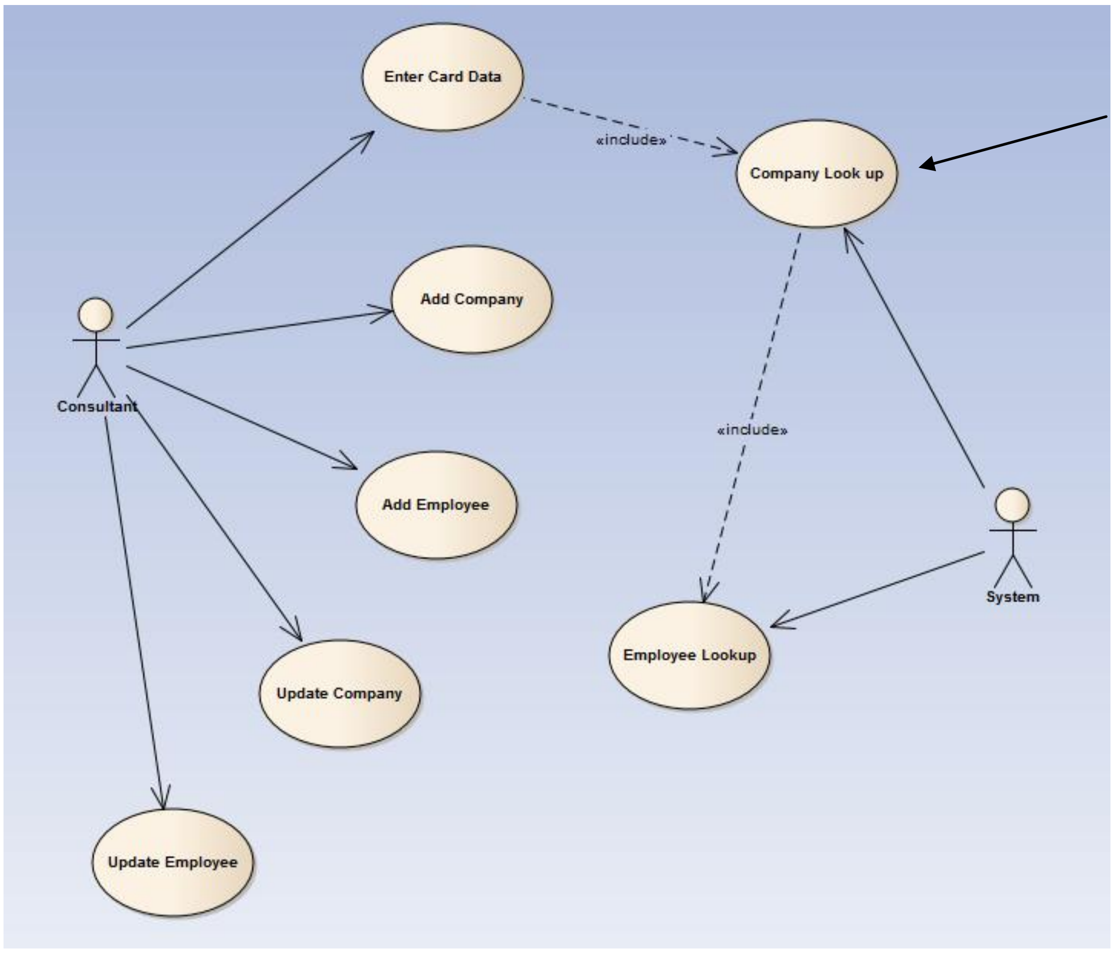
They will pick up a business card and enter the name of the company. This will cause the system to produce a filtered list of companies with their employees.

The next time line we need to add represents the class clsCompanyCollection. Do this by adding a new life line…



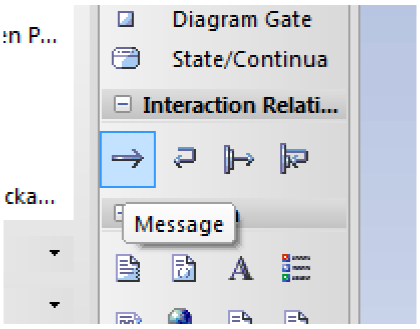
We now need to send a message to the class from the actor.

Messages may be identified by looking at the use cases on the use case diagram.

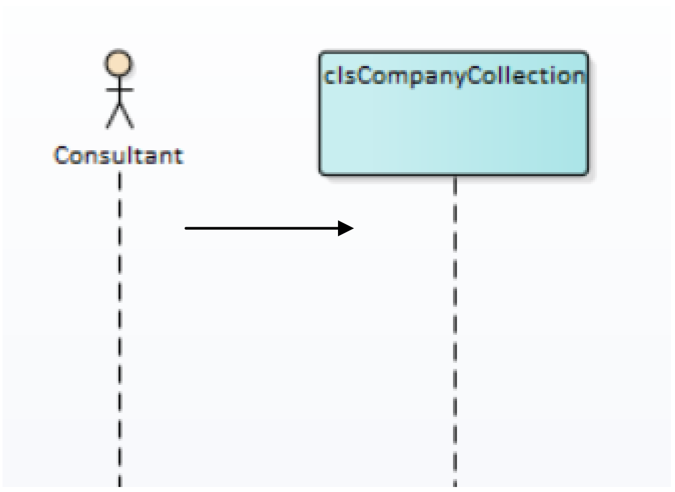


In this case “Company Look Up”

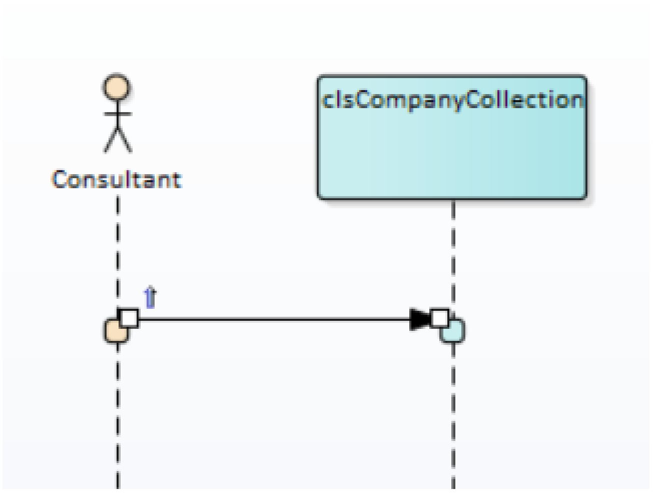
Select the message tool from the toolbox…



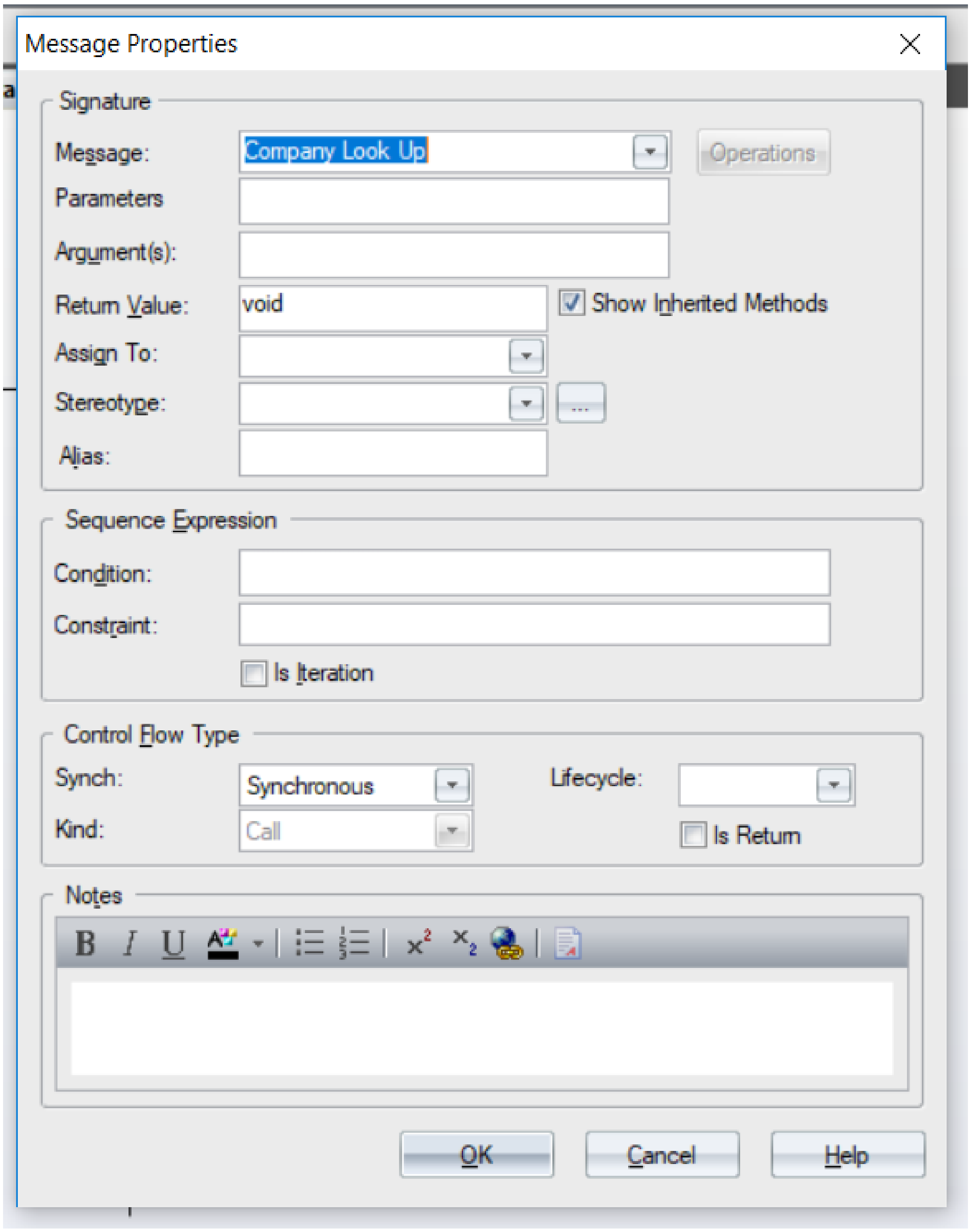
And draw between the actor and the object.



This will create the new message…



Double click the message to give it a name…



Message Properties

Signature

Message:	Company Look Up
Parameters	
Argument(s):	
Return Value:	void
Assign To:	
Stereotype:	
Alias:	

[✓] Show Inherited Methods

Sequence Expression

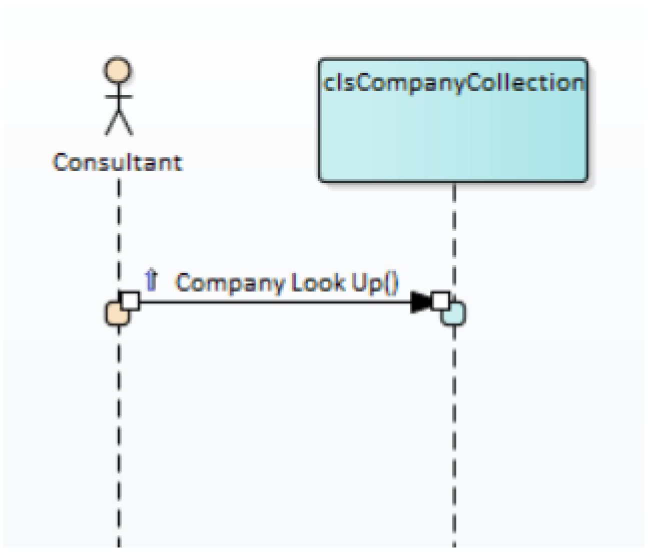
Condition:	
Constraint:	

[ ] Is Iteration

Control Flow Type

Synch:	Synchronous	Lifecycle:	
Kind:	Call		[ ] Is Return

Notes

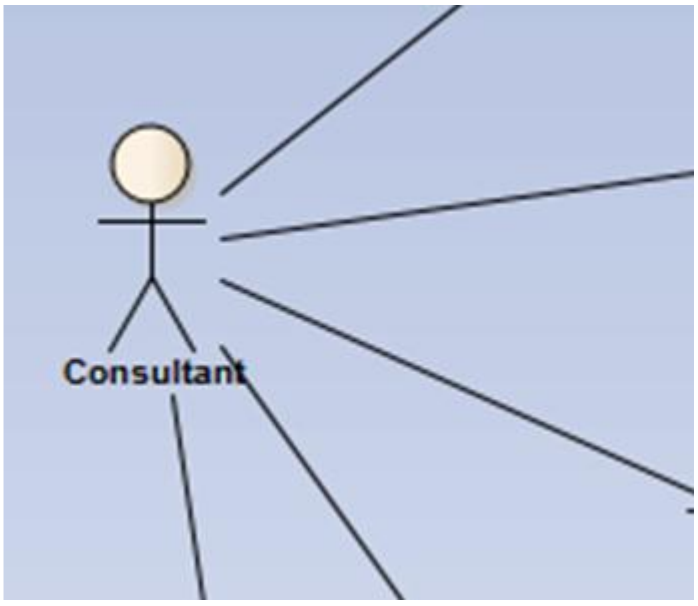


Step one complete – so what?

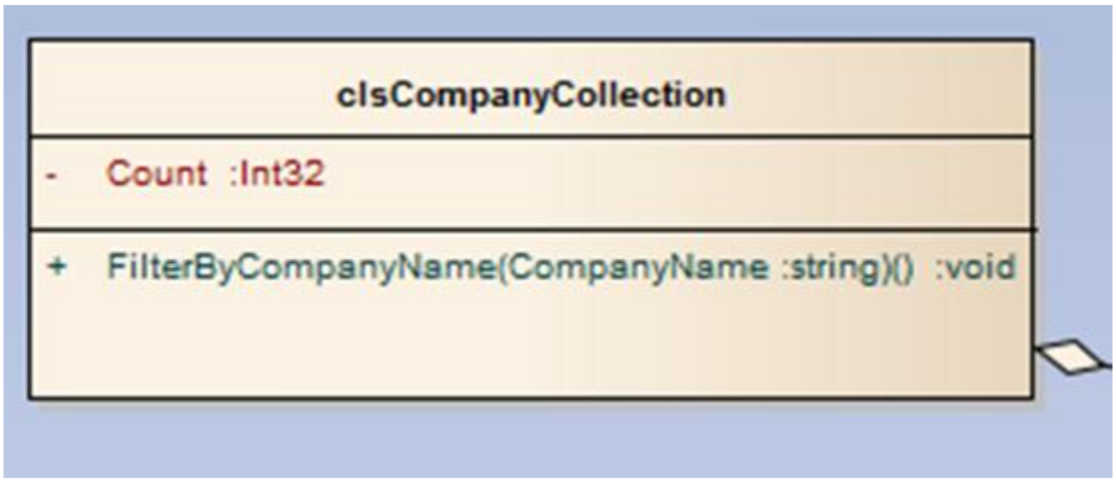
What we are doing here by linking the use case with the class diagram is creating a cross check on our assumptions.

As we create messages in the sequence diagram we need to make sure that it maps onto both the use case and the class diagram.

- Do we have an actor on the use case that matches the sequence diagram? Yes!



- Do we have a class on the class diagram? Yes!



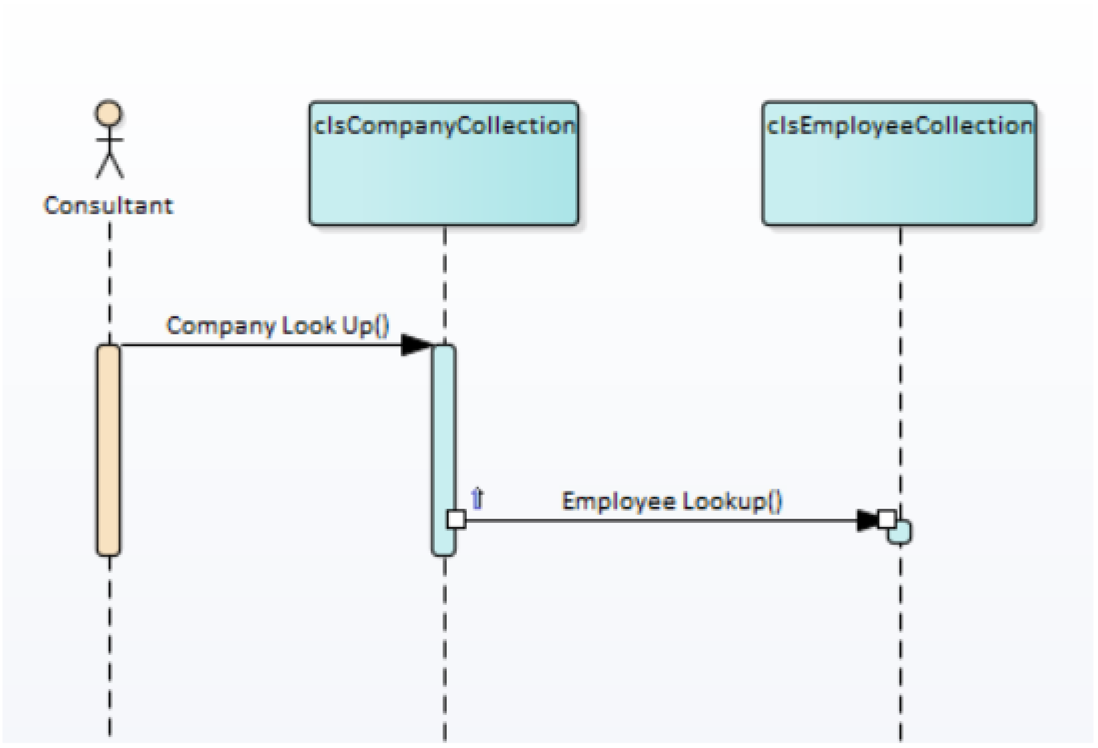
- Do we have a method in the class that is able to handle the message? Yes!

+ FilterByCompanyName(CompanyName :string)() :void

So far so good!

The next step in the use case is for the system to produce a list of employees for this company.

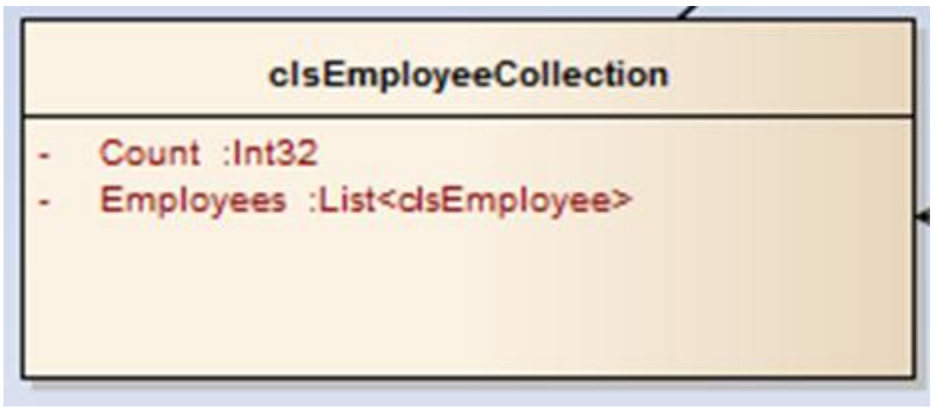
The sequence diagram might look something like this…



Let’s ask the same questions again!

- Do we have an actor on the use case that matches the sequence diagram? Yes!
- Do we have a class called clsEmployeeCollection? Yes!

- Do we have an operation in the class that is able to handle the message?

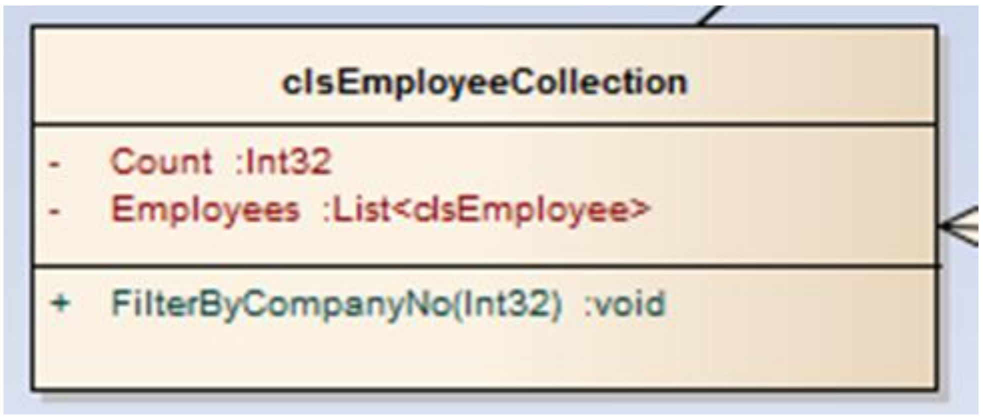


- No!

What we have done with the sequence diagram is identify a missing operation in our class clsEmployeeCollection to support the use case / message Employee Lookup.

We need some way of sending the class a company number and the class will produce a list of employees for that company.

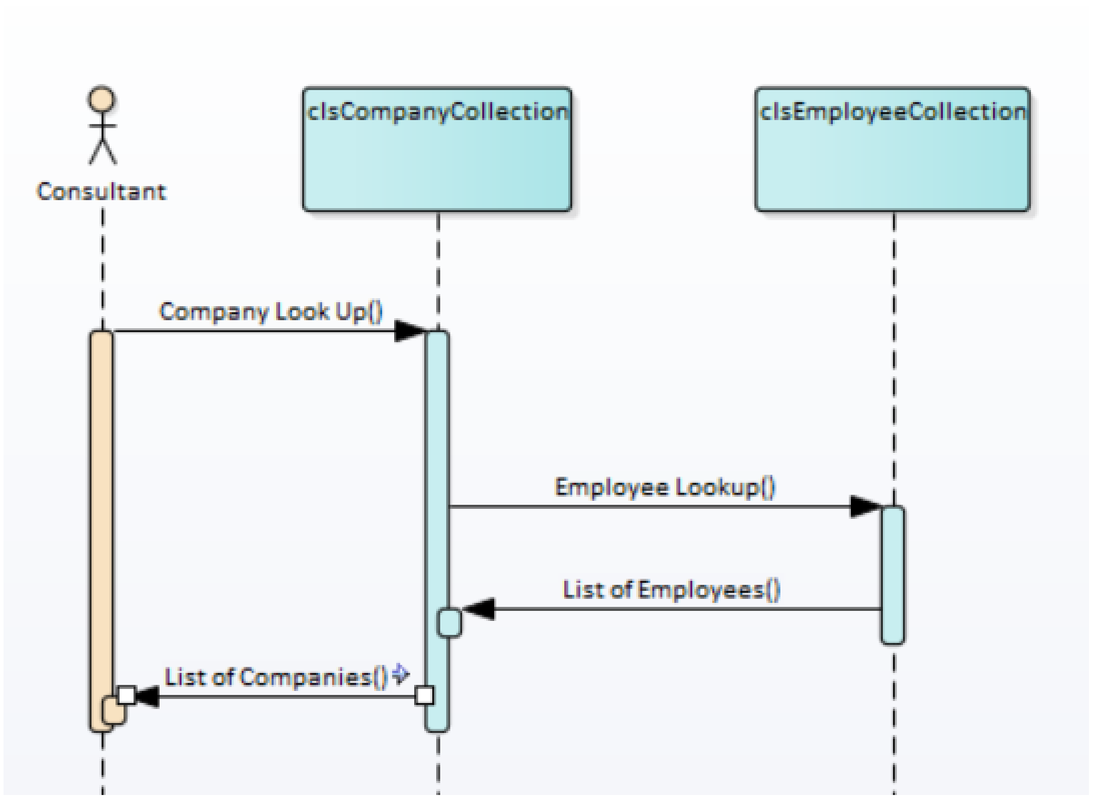
Let's modify the class like so…


clsEmployeeCollection
- Count :Int32
- Employees :List<clsEmployee>
+ FilterByCompanyNo(Int32) :void

We are now able to answer the three questions positively.

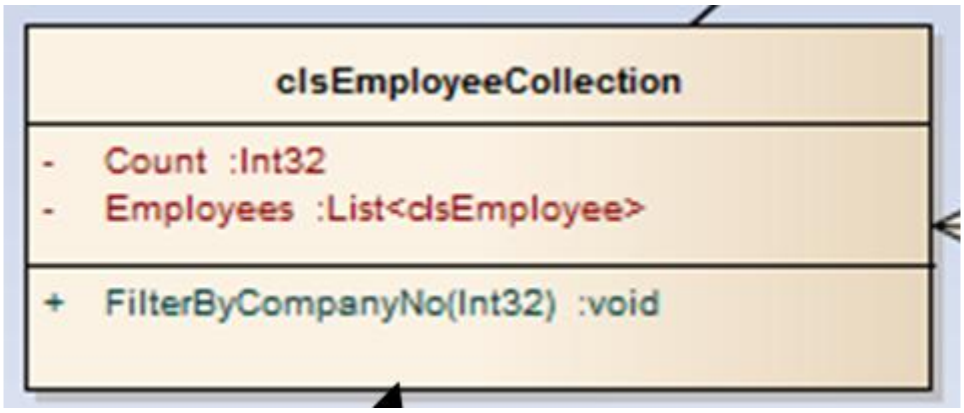
- Do we have an actor on the use case that matches the sequence diagram? Yes!
- Do we have a class on the class diagram? Yes!
- Do we have a method in the class that is able to handle the message? Yes!

Now that the sequence appears complete we need to add on the data being returned from the objects.



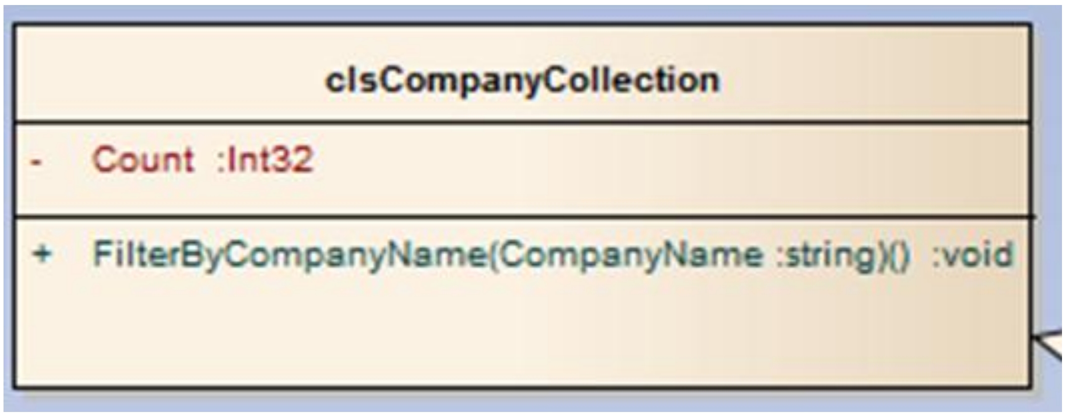
Now we have modified the sequence diagram to include data returning from the classes we need to ask the question is this functionality supported by the class diagram.

In the case of clsEmployeeCollection we have an attribute that is modified by the Employee Lookup message called Employees...


clsEmployeeCollection
- Count :Int32
- Employees :List<clsEmployee>
+ FilterByCompanyNo(Int32) :void

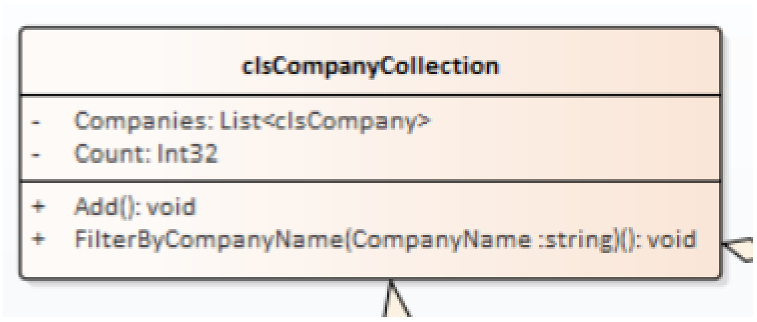
This allows us to access the list of filtered employees.

Do we have the same feature for clsCompanyCollection?



No!

So this might be added to the list of attributes like so...


clsCompanyCollection
- Companies: List<clsCompany>
- Count: Int32
+ Add(): void
+ FilterByCompanyName(CompanyName :string)(): void

We also have lots of other useful clues to check if our analysis is heading down the right track.

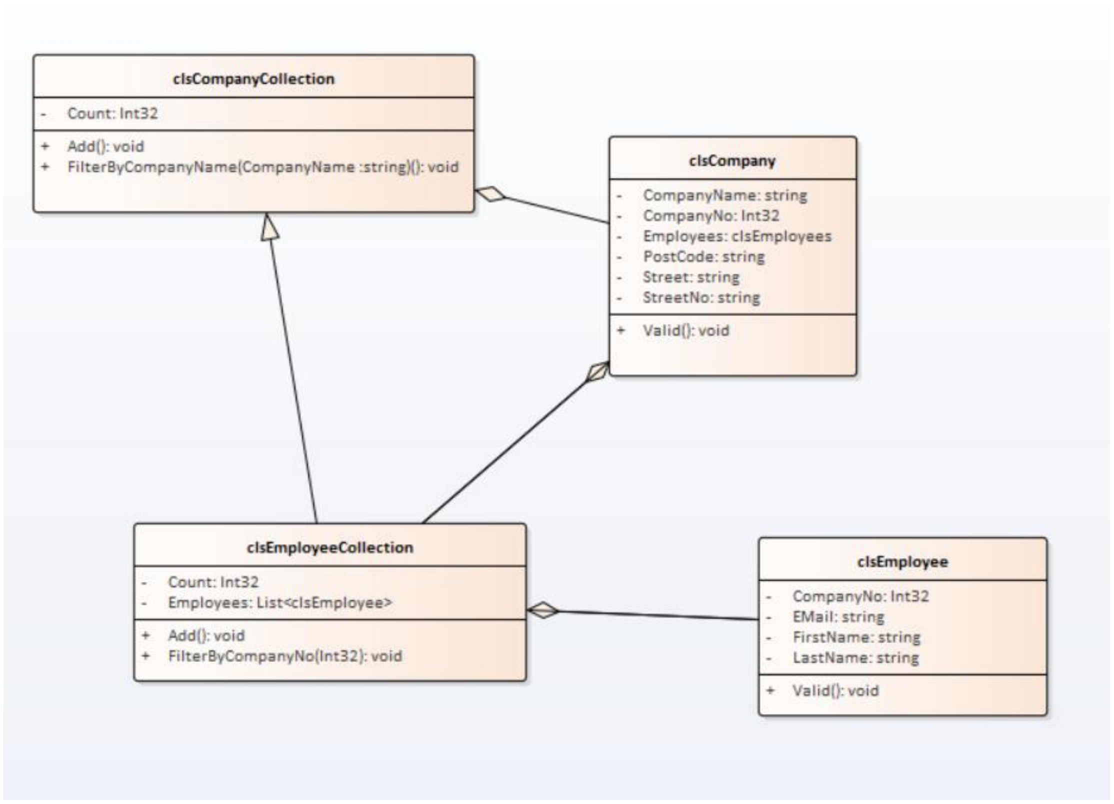
From the sequence diagram we are getting clues as to what attributes need to be present in the system. Company Name and Company No being two such examples.

The other thing to consider is objects may only communicate with each other on the sequence diagram if there is a corresponding association on the class diagram.

The objects Companies and Employees must have an association between their corresponding classes.

It seems in this case that the two classes have an association missing.

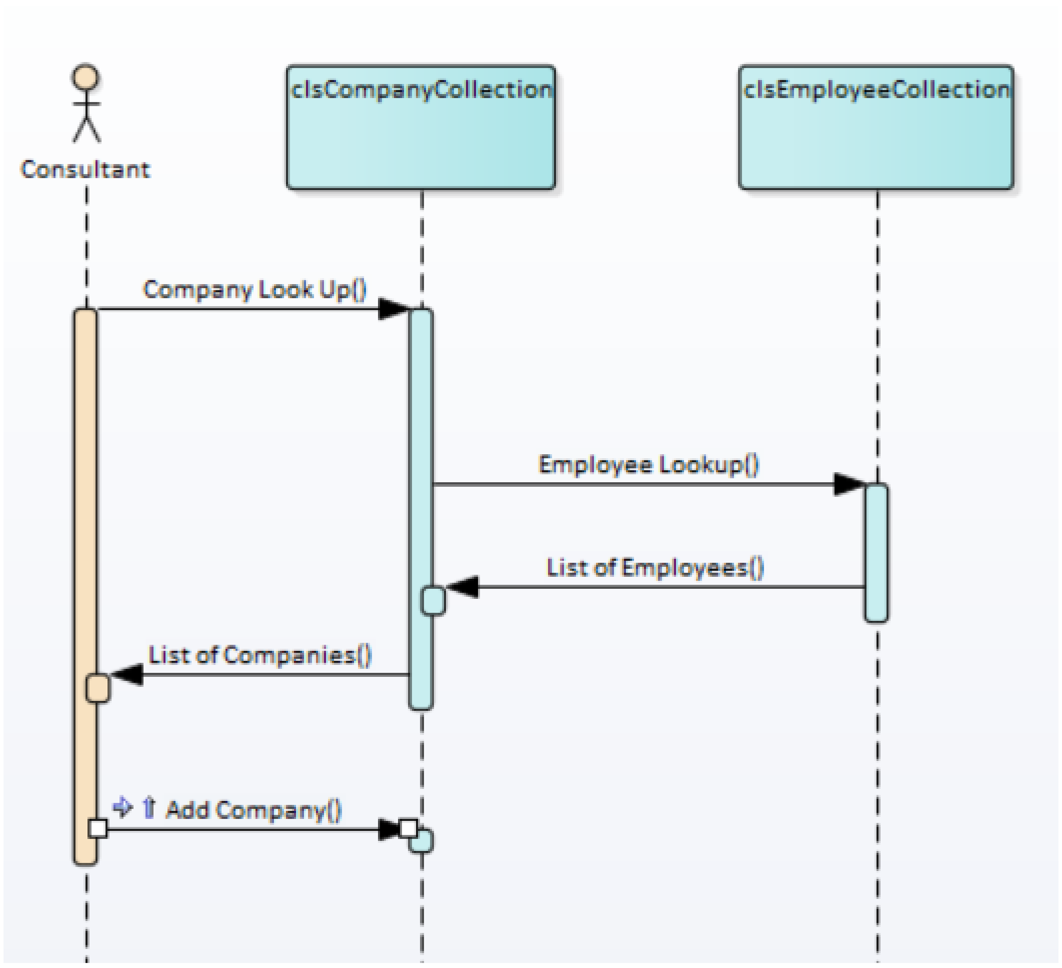
Fix it like so...



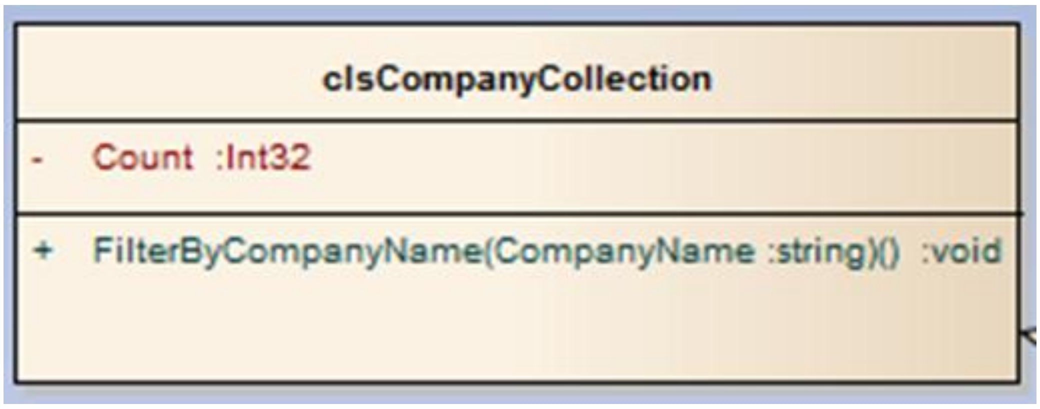
Extending the Sequence Diagram

Once we have entered the company name and established that it doesn’t exist on the system we are in a position to add a new company / employee.

Let’s add the Add Company use case / message to the diagram…

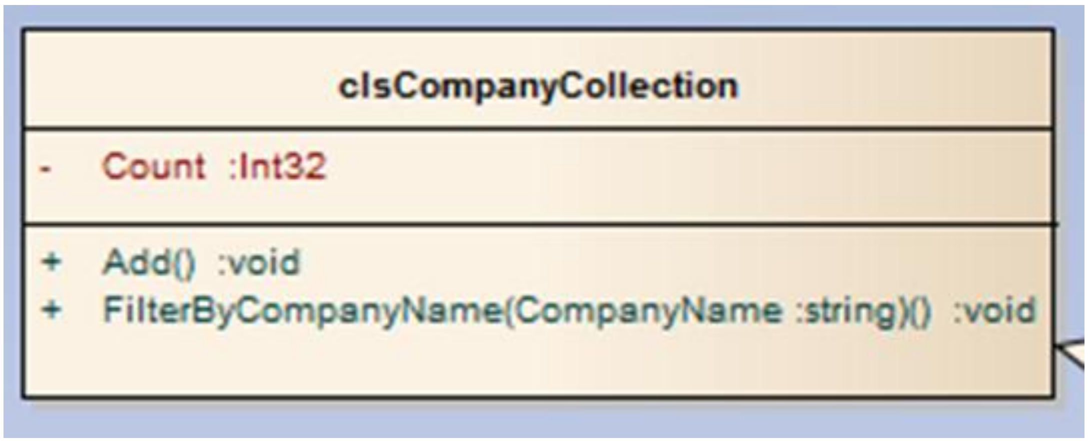


Do we have an operation in clsCompanyCollection to support this message?

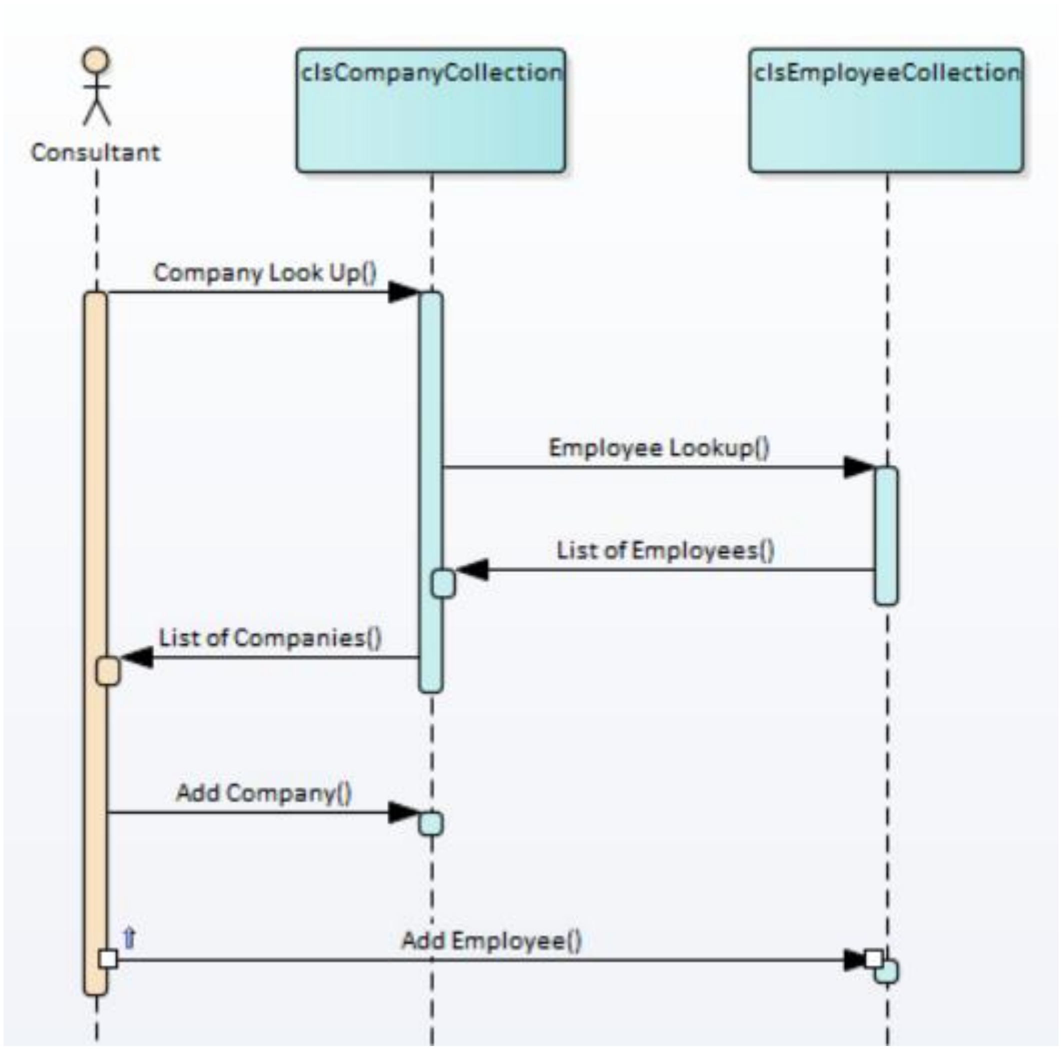


No!

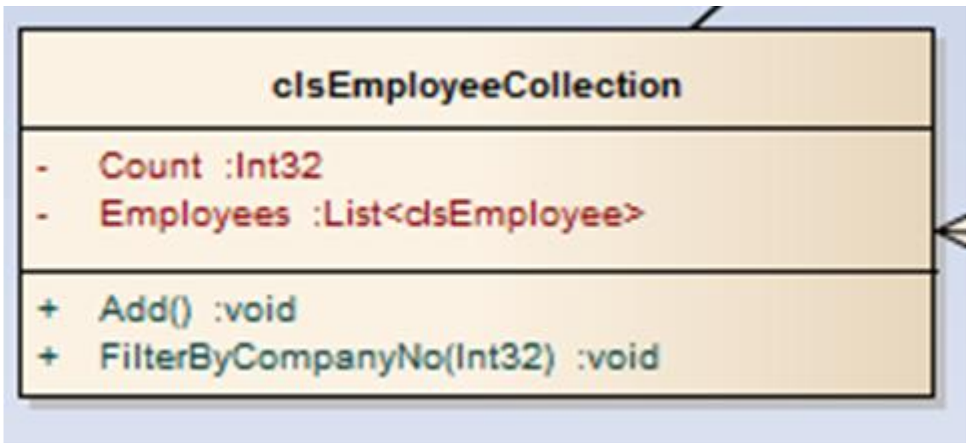
So let’s modify the class diagram accordingly…



Now we need to include the Add Employee use case / message…



Where we find the same flaw in the class diagram as before that is no Add method…


clsEmployeeCollection
- Count :Int32
- Employees :List<clsEmployee>
+ Add() :void
+ FilterByCompanyNo(Int32) :void

Note that we are adding the Add functionality to the collection classes.

We now hit another problem.

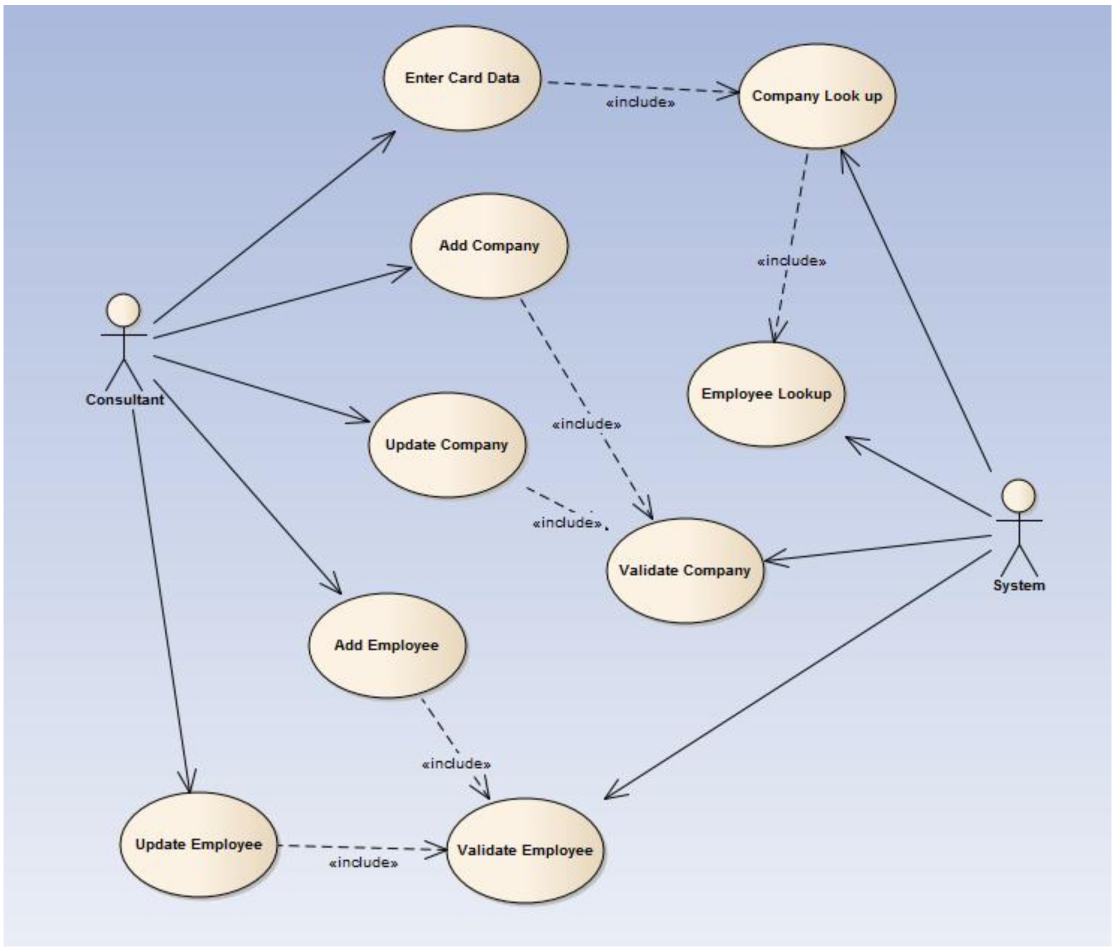
Is it a good idea to be adding data without any suitable validation?

One clue as to the benefits of a sequence diagram is its name – it helps us also to think about the sequence of messages / use cases in the system.

In creating the sequence diagram we have now spotted problems with both the use case and the class diagram.

We need to add some sort of validation to the use case diagram and we need to decide where to place the validation in the sequence of messages.

We will modify the use case as follows…



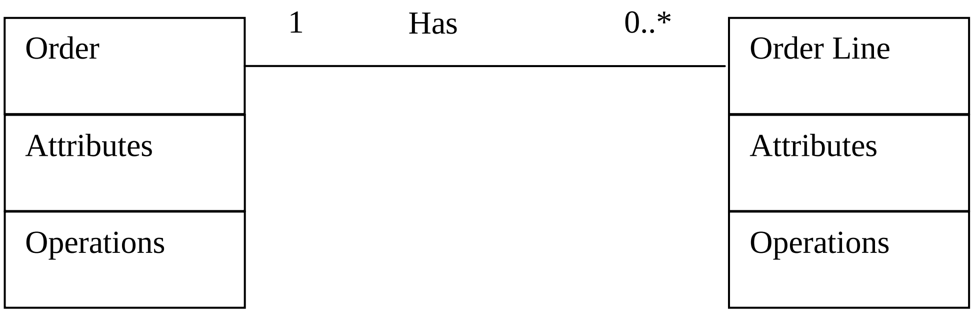
We need to decide where the validation needs to go.

Composition and Aggregation Revisited

Composition

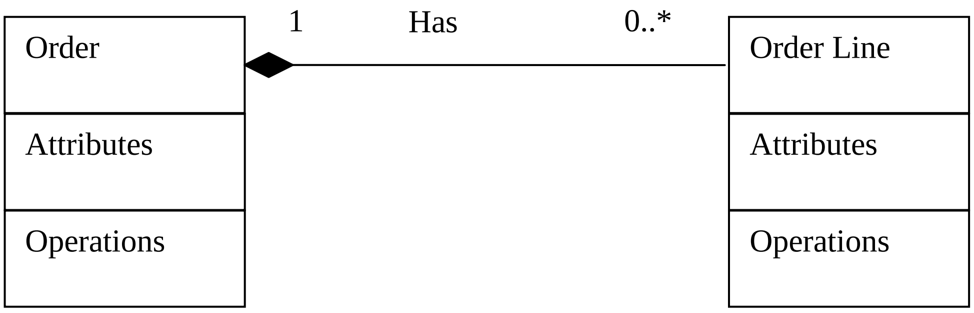
Two classes are related via composition when you cannot have an instance of one class without the parent class.

For example if we were placing an order in a system could we ever have an order line without an associated order?



The answer is "no".

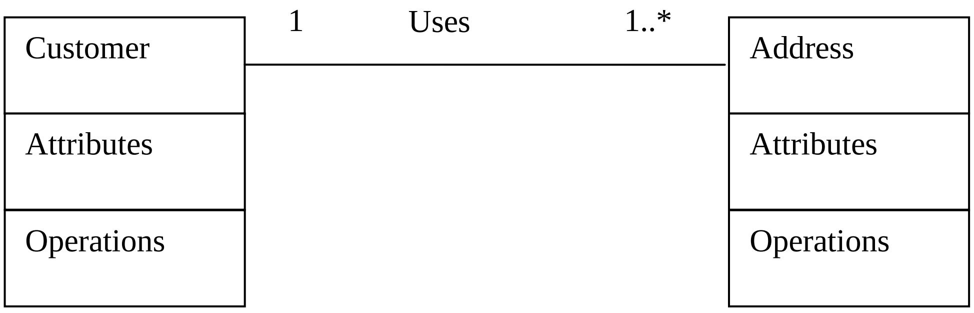
To indicate this strong relationship between the two classes we use a solid diamond like so…



Aggregation

In this case we are asking the question "can this class exist without the associated class even though they have a relationship?"

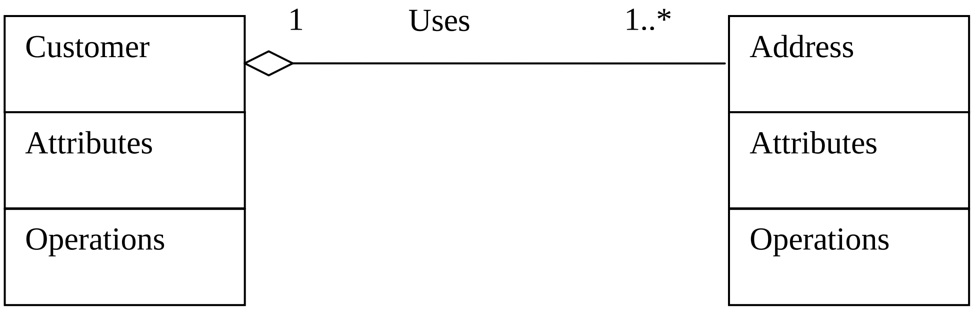
A good example of this is customer and address…



If all of the customers were to vanish from our store would addresses cease to exist?

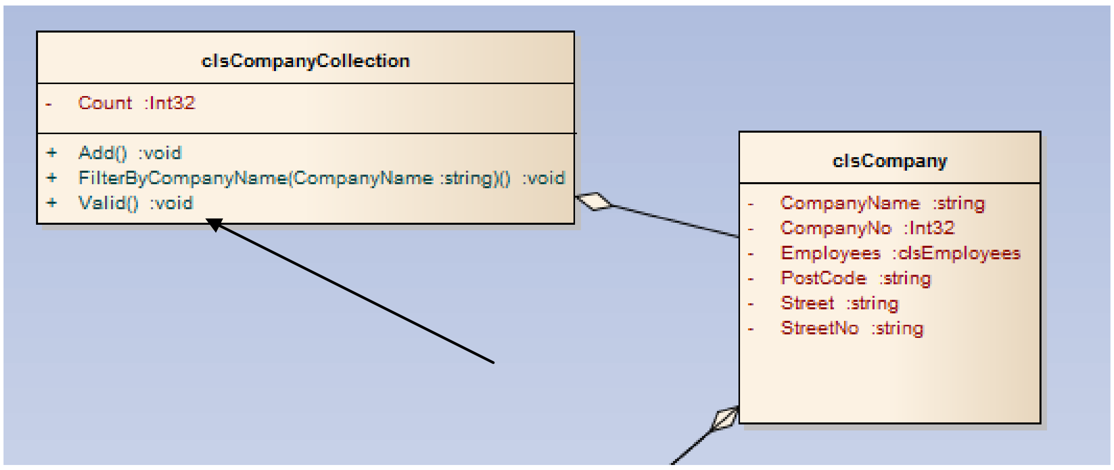
Obviously the answer is they would not!

To indicate this looser relationship a white triangle is used like so…



Composition and Aggregation give us important clues as to which class gets ownership of which attributes / operations.

We could design the classes like so…

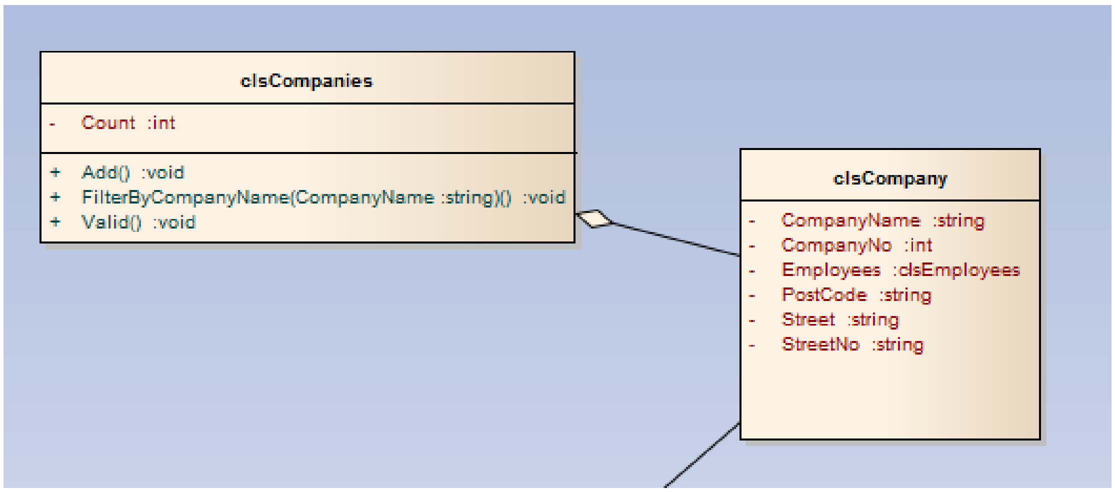


We need to ask, are we dealing with an aggregation or a composition?

Can we have an instance of a company on its own without the collection class?

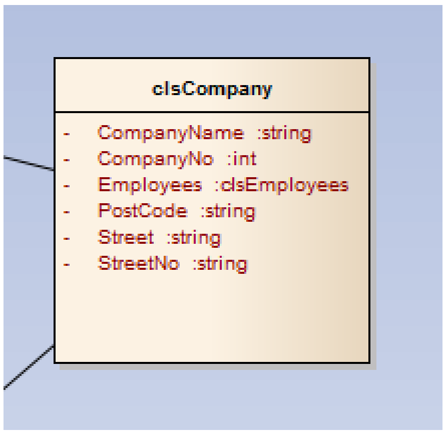
The answer is "yes" we may at times need to deal with a single company on its own.

But the diagram indicates an aggregation…



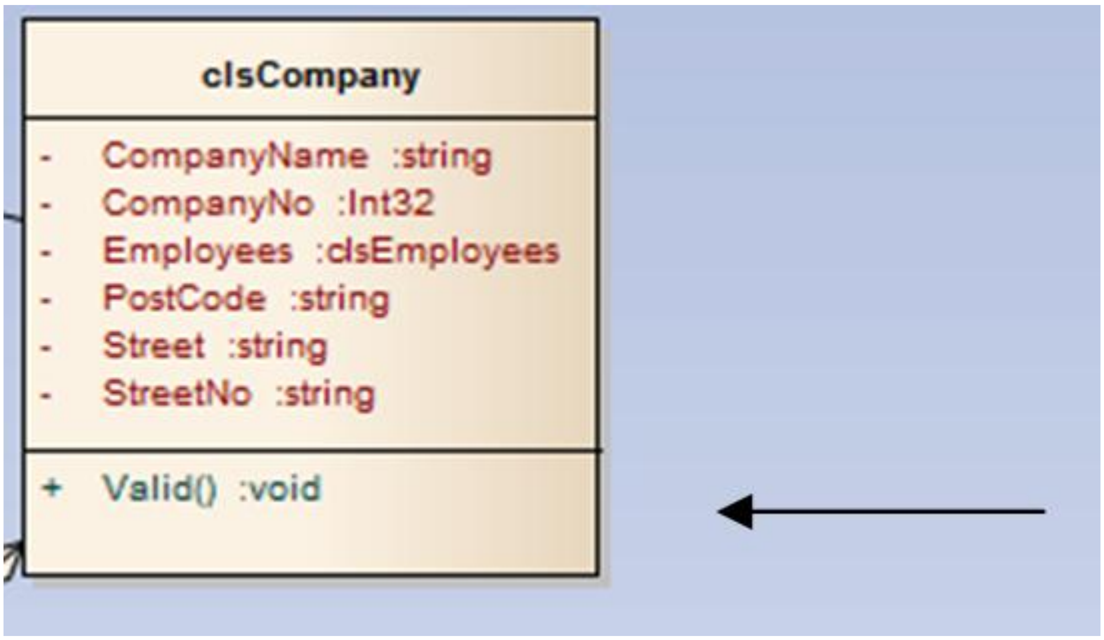
clsCompanyCollection uses clsCompany.

OK – so what happens if we want to deal with one company and we don't need an instance of clsCompanyCollection, how do we validate the company details?

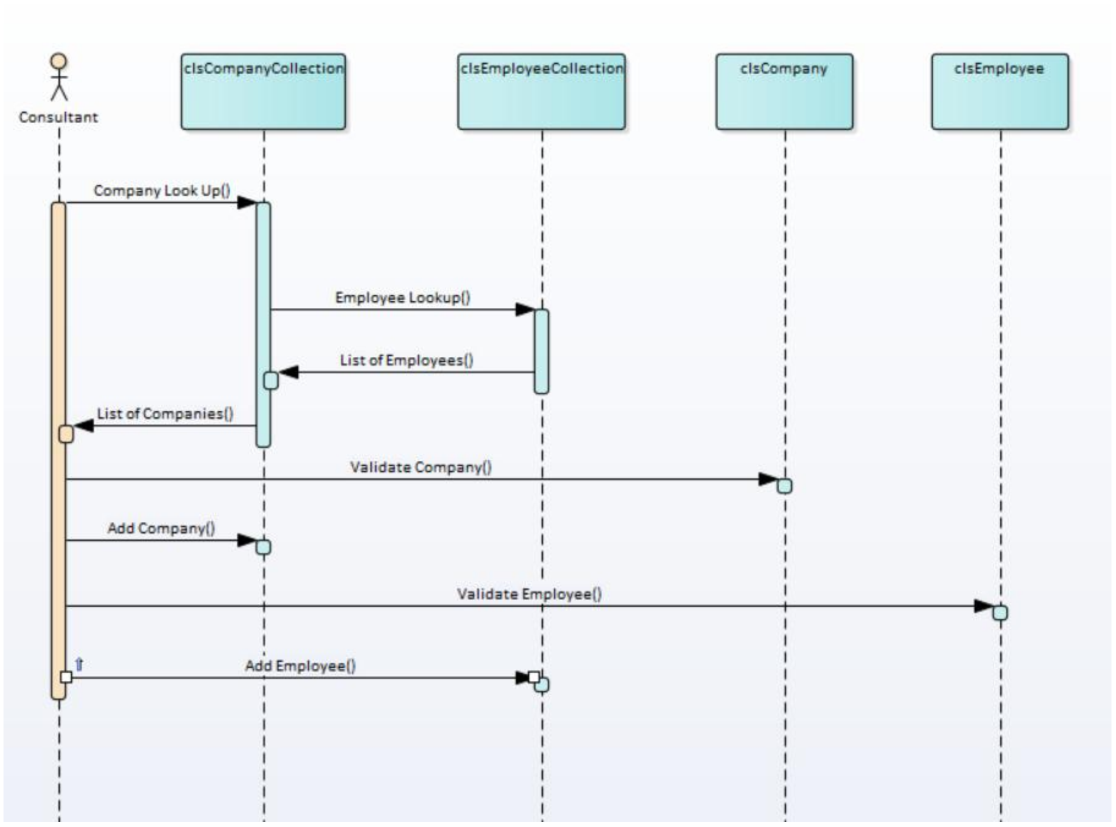


The answer is that we can't – we have placed the validation inside the wrong class!

The diagram needs to be modified like so…



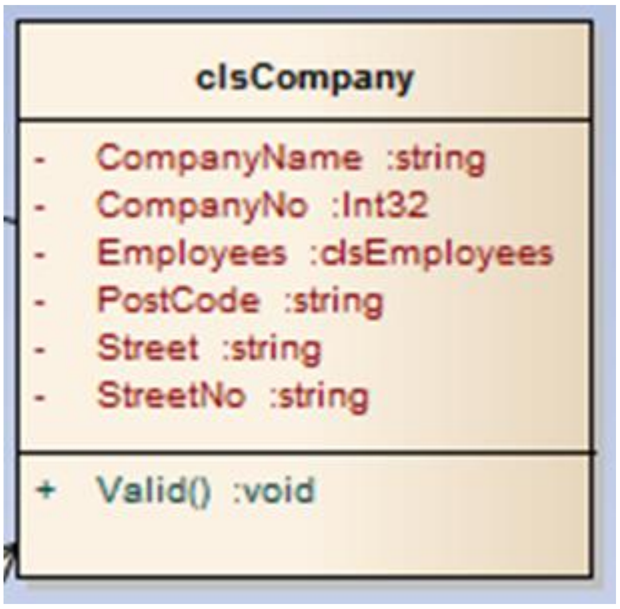
The sequence diagram starts to look something like this…



Spotting Errors in Return Values

As your sequence diagram develops it will also help you identify issues with your system when it comes to return values of operations.

Take a look at the following class...

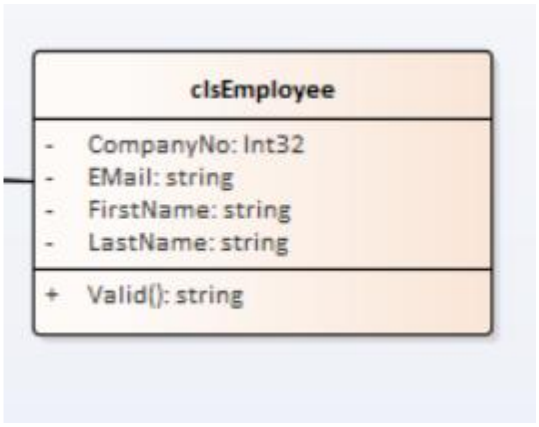


With the valid operation does it make sense that the return value if set as void?

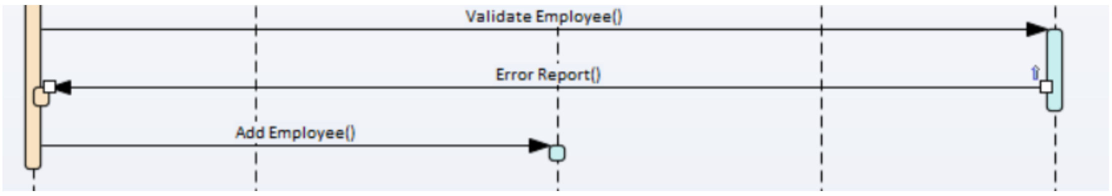
Think about it this way.

I send the operation some data to check, what data should it come back with?

Surely some sort of error message? Does this make more sense?



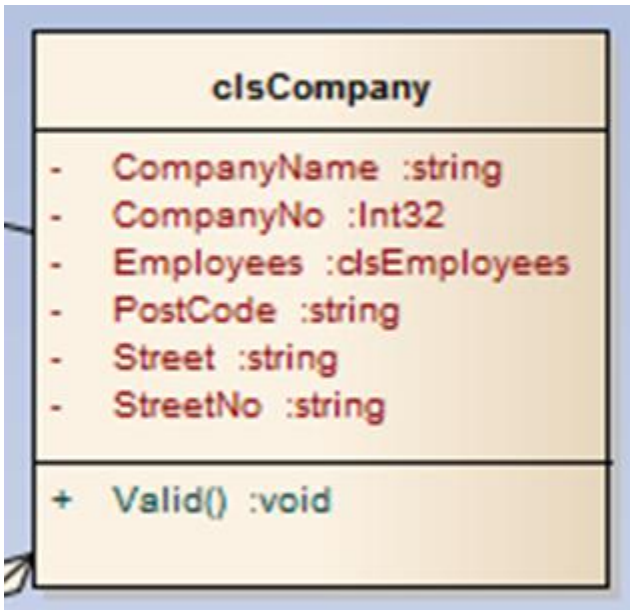
Also if the operation is returning an error message then we need to reflect this in the sequence diagram, perhaps like so...



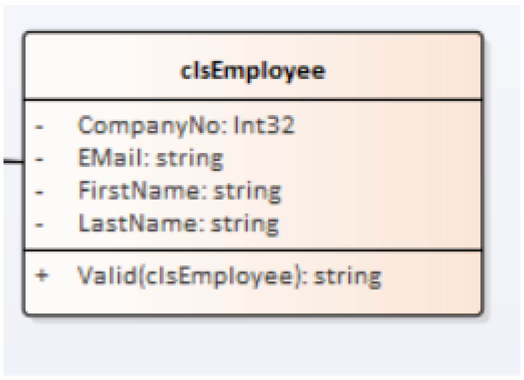
Lastly with each operation we need to think about what data needs to be sent in the message for it to do its job.

In the case of validation we need to send it some data to validate.

The operation shouldn’t then look like this...



With no parameter for Valid, it needs to accept some incoming data like so...



It is still important to appreciate that there is a lot of work still to do. This is a voyage of discovery and we are still a long way from producing a final map of the problem domain.

It is also important to appreciate the role the sequence diagram plays in helping us to cross check and refine your understanding of how the system is going to work.