Getting Started with Enterprise Architect

Creating Class Diagrams

Having seen how we create our use case diagrams we will have a go at creating a class diagram.

What is a class diagram?

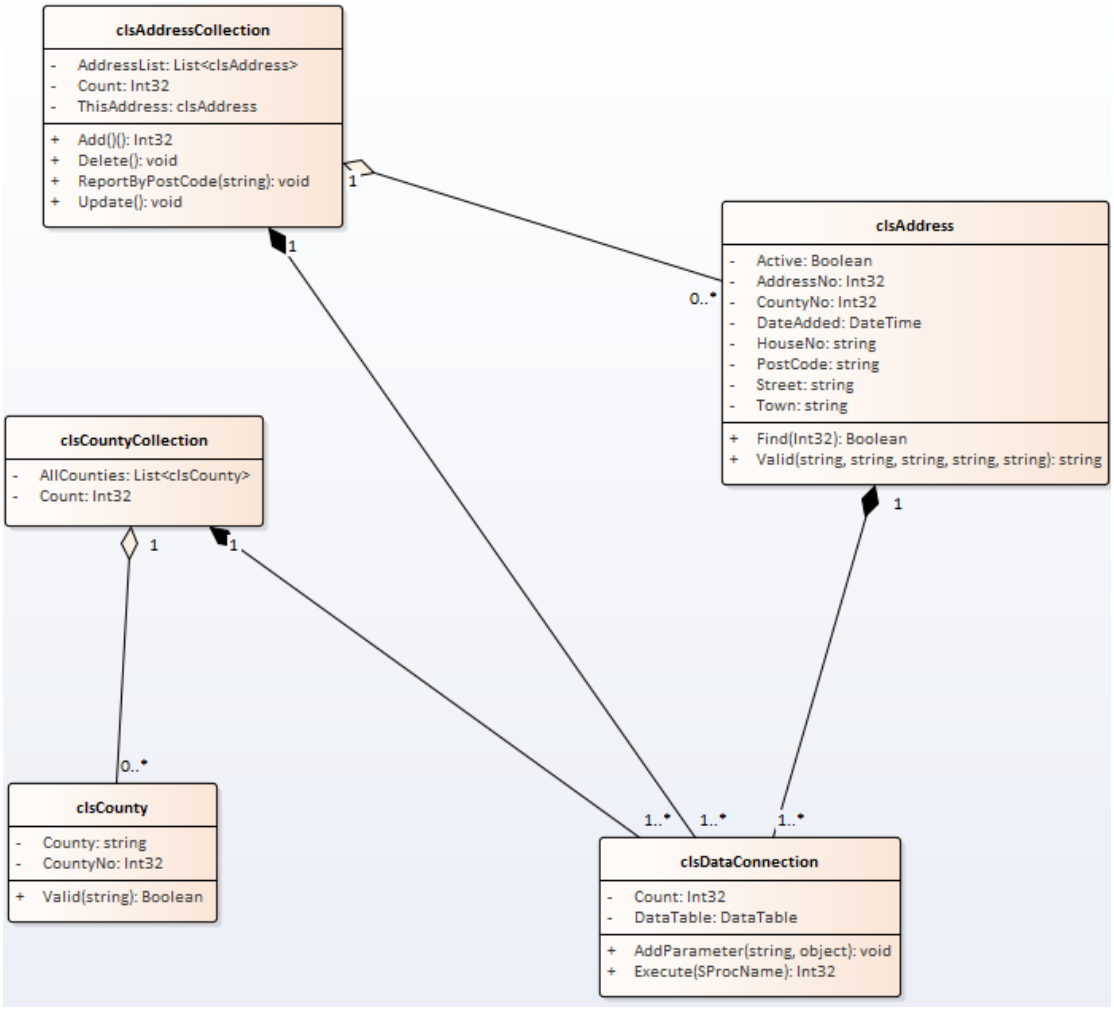
In creating our system we will at some point need to create some code.

This code will inevitably be organised into classes that will in turn be stored in a class library.

Before we get to the stage of thinking about the code we need to spend some time thinking about the problem at hand.

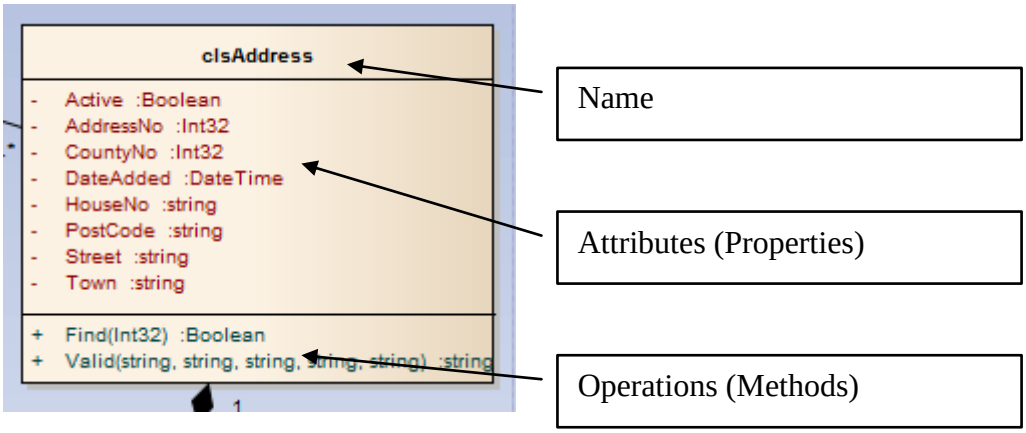
Class diagrams allow us to step back from the problem and think about what the classes are going to do.

For example here is a class diagram for the address book…

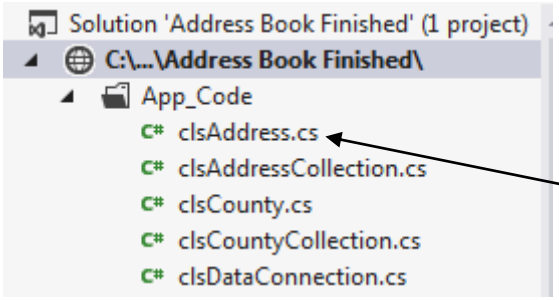


Each class is defined as a rectangle linked by different types of associations.

Within the rectangles we specify the class name along with attributes (properties) and operations (methods).



The class diagram is ultimately implemented as code in our system…



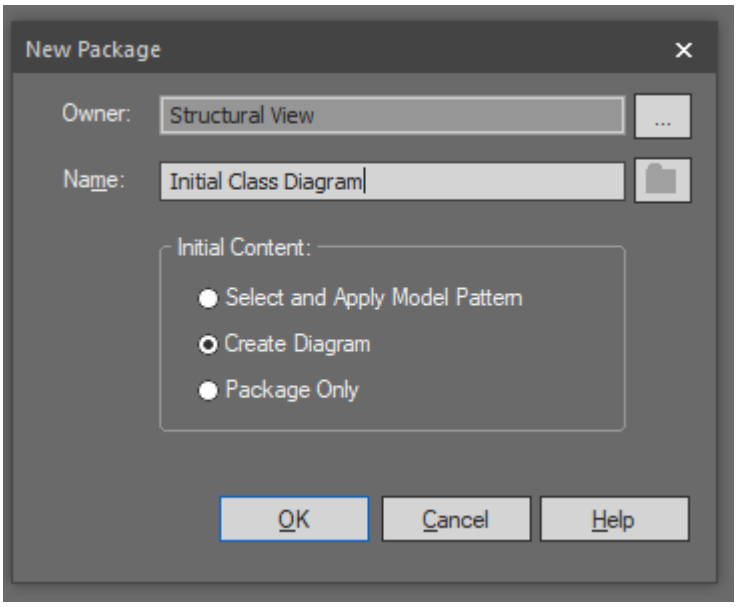
Containing the code that implements the attributes and operations as methods and properties…

```
    //AddressNo public property
    public Int32 AddressNo...

    //HouseNo public property
    public string HouseNo...

    //Town public property
    public string Town...

    //Street public property
    public string Street...

    //PostCode public property
    public string PostCode...

    //CountyCode public property
    public Int32 CountyCode...

    //DateAdded public property
    public DateTime DateAdded...

    //Active public property
    public Boolean Active...

    //function for the public validation method
    public string Valid(string houseNo,
                        string street,
                        string town,
                        string postCode,
                        string dateAdded)
    ///this function accepts 5 parameters for validation
    ///the function returns a string containing any error message
    ///if no errors found then a blank string is returned...

    //function for the find public method
    public Boolean Find(Int32 AddressNo)...
```

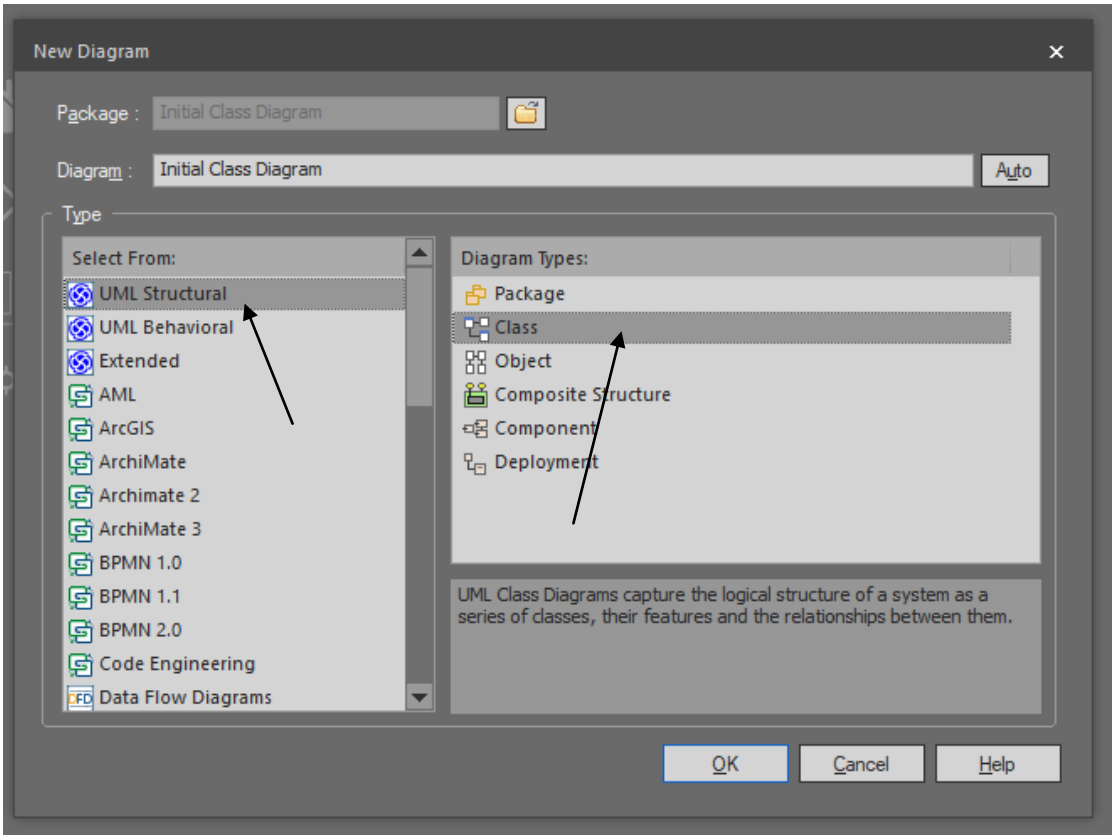
Creating the Diagram

As with the use case diagram we will create a package to store the initial diagram. We are likely to have many such diagrams in the finished system.

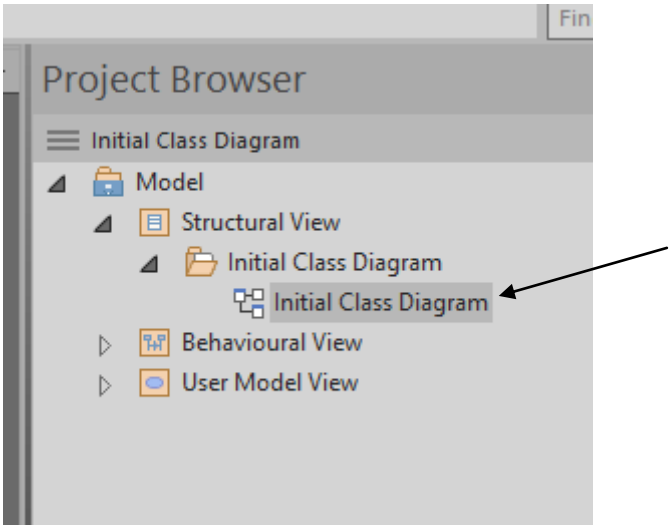
Right click on the structural view and add a package called initial class diagram.



When asked for which diagram you need to create select the following...



Press OK and you should see the diagram created for you...



Double click the diagram to edit it.

In order to create the initial class diagram we need to revisit the specification to identify candidate classes. I say candidate as we may not be right in our first go and like all documents there will be revisions made to them as the picture of how the system works comes together.

The first thing we will look at is the detailed specification.

> *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 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.*

One way to approach this is to sit down as a team and identify the nouns in the specification. The nouns give us an indication of the things (objects) that need to be handled by the system.

So as a first stab...

> *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 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.

Which gives us the following initial list...

consultant
desk
business card
flyer
system
company
company name
client name
contact
user

We now need to filter the list to decide if these really are classes. This takes some thought and skill however we face similar problems for when we were thinking about use cases.

The Desk?
Is the desk really part of our computer system? Probably not!

What about System?
We don't include the system on the class diagram as the diagram is the system. (Unlike when the system is an actor in a use case!)

Have we identified the same class by different names?
Consultant and User - are they the same? Probably!

Do hedge your bets a little bit though you may find later that you need to change your mind.

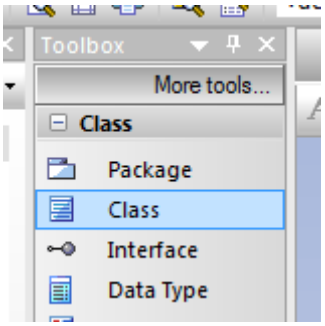
What about business card and flyer? This may turn out to be quite subtle so at this point I am treating them as separate items.

What about company and company name? This illustrates an instance where one is a class the other is an attribute of that class. We will probably have a class called Company and it will have an attribute Company Name. This is probably also true for contact and client name. The contact will be a class the client name will be an attribute of the contact.

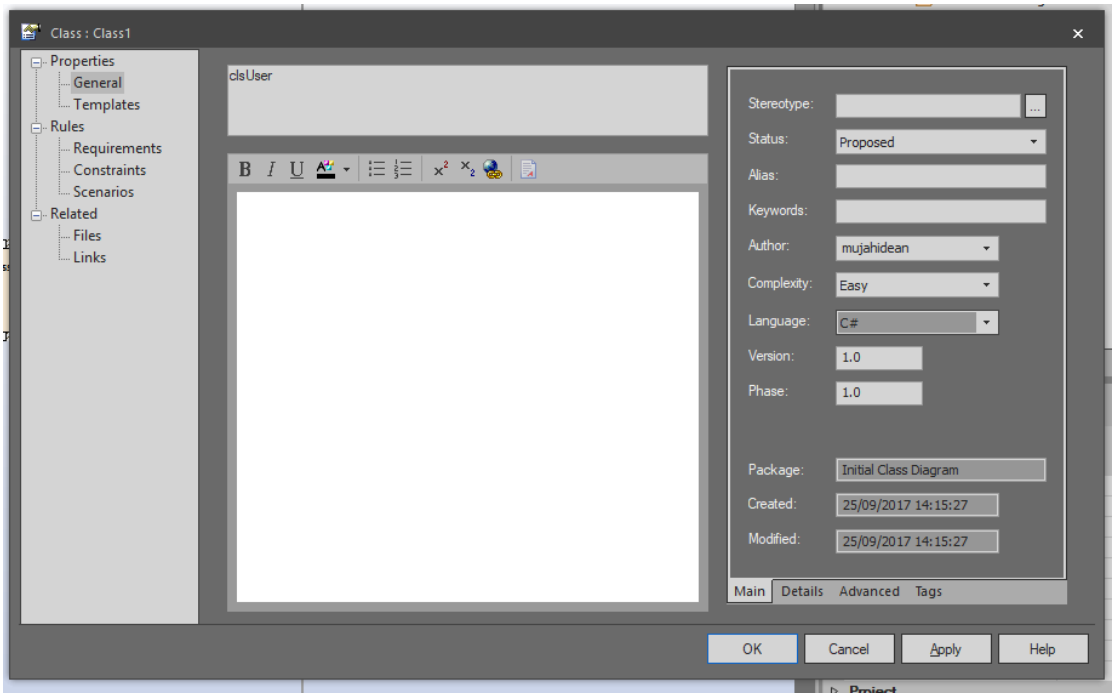
So let's filter the list like so and see how we get on...

user
business card
flyer
company
client

To create a class on the diagram select class in the toolbox and draw it onto the diagram...

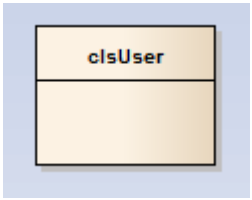


Give the class the name clsUser...



Change the language for the class from Java to C# so that it supports the data types for the language we are using.

You should then see the class on the diagram...



Creating Relationships

The first sort of relationship we are interested in at this stage is the association relationship.

An association describes how and if two classes are going to interact with each other in some way.

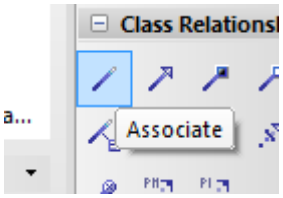
The first step in identifying associations is to take another look at the event table.

Subject	Verb	Object	Response
Consultant	Inputs	Card	Data accepted by the system
System	Checks	Card	Identifying duplicates
Consultant	Updates	Card	New data input

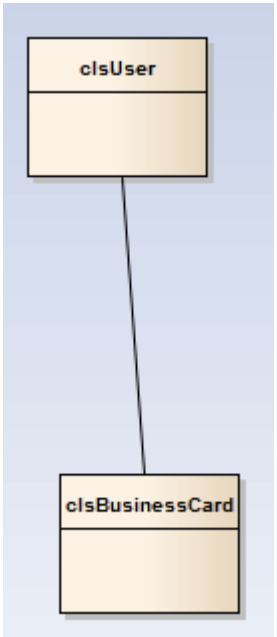
From the event table we can see a clear link between the user (Consultant) and the business card.

To create the association in EA we again draw the link between the two classes.

From the toolbox select the associate tool…



And then draw the association between the two classes.

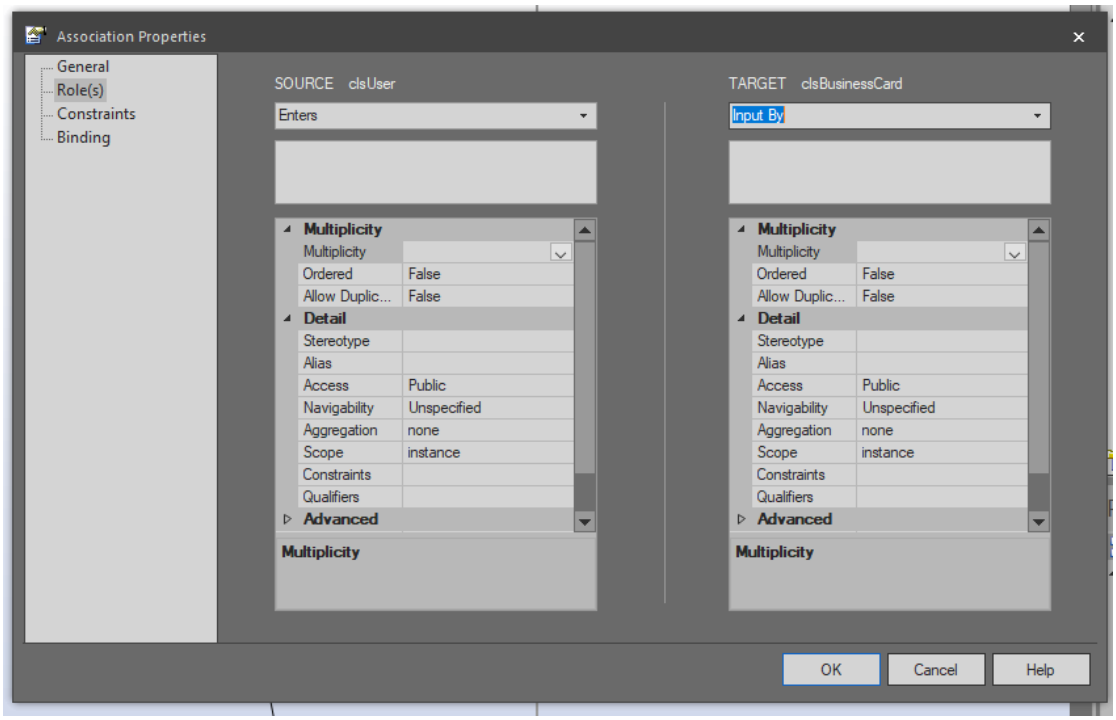


Specifying the Role

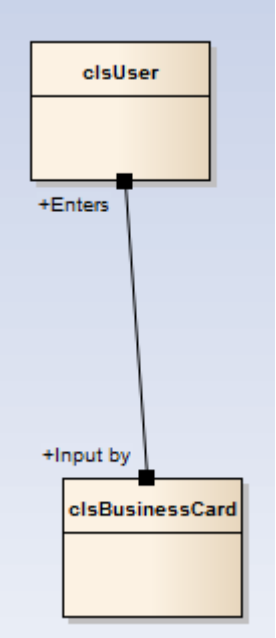
The next step is to specify the role between the classes. This describes what is going on.

Double click on the association to access the properties for the association.

For clsUser the role is “Enters” and for clsBusinessCard the role is “Input By”…



Set the Target role as “Input by” such that the diagram looks like this…



Multiplicity

The multiplicity describes how many of a certain class are associated to another.

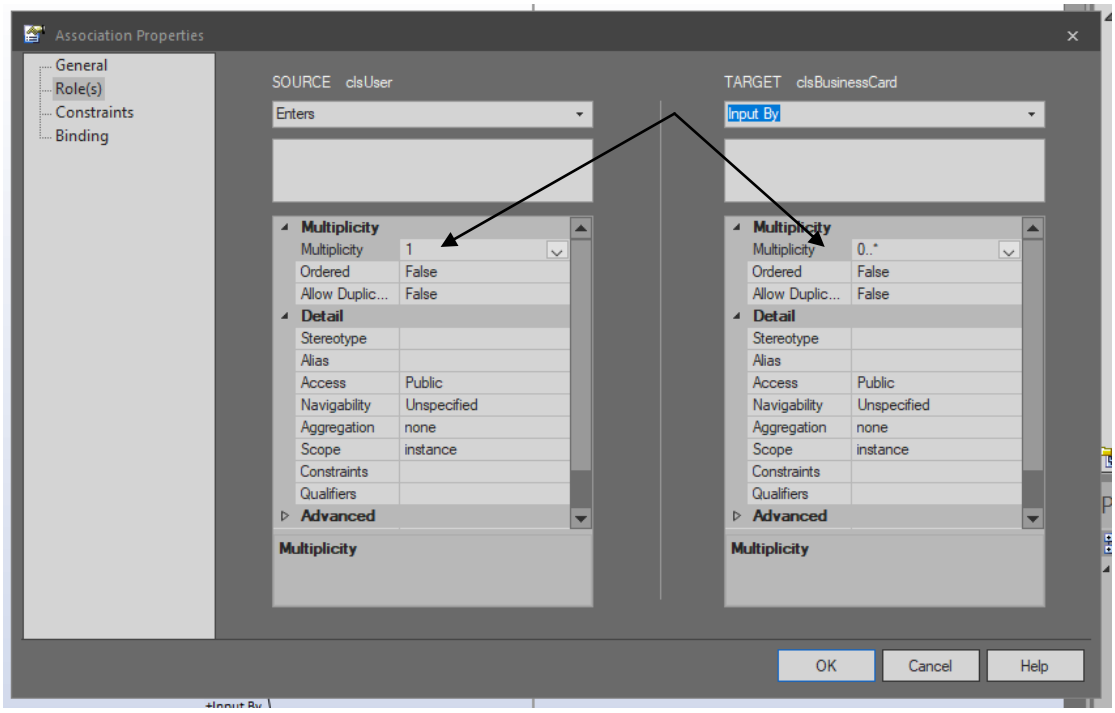
This is where it pays to apply a bit of thought.

How many cards will a user enter?
How many users is a card input by?

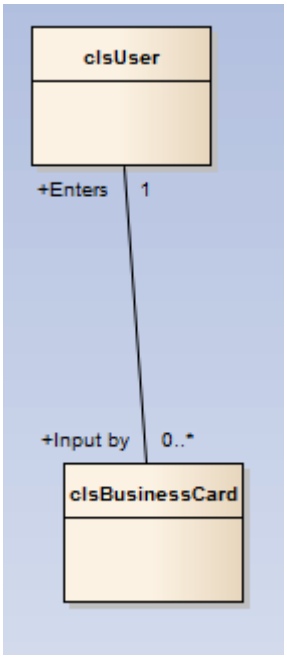
(There may be room for debate depending on how you understand the business rules!)

In this example I shall assume that a card may only be input by a single user and a user enters 1, zero or many cards.

To set this up in EA double click the association to set the multiplicity…



Do the same for each end such that it looks like this…



The symbols indicate…

1 only one
1..* one or many
0..* zero or many

Aggregation and Composition

Aggregation and composition are useful perspectives on the association that help us to think more clearly about the design. Aggregation and composition will ultimately influence how we write our code.

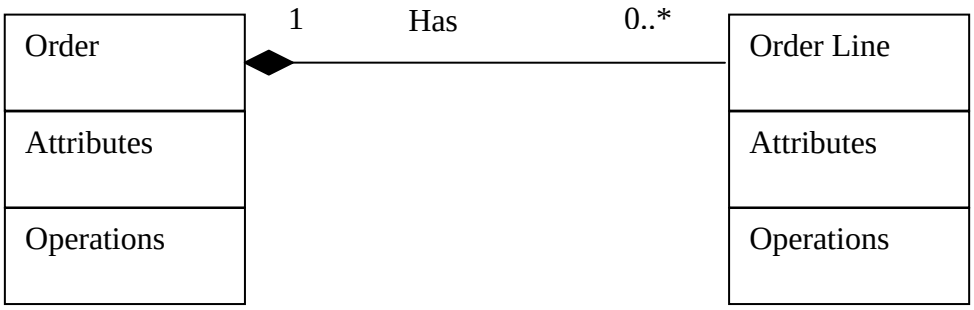
Composition

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…

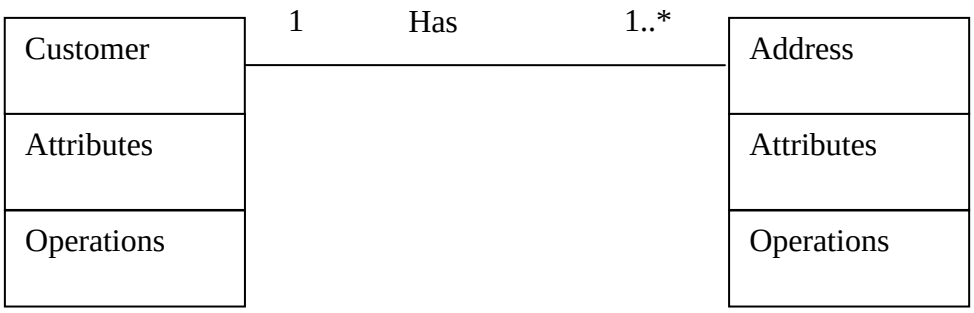


Aggregation

The other association type between classes is an aggregation.

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

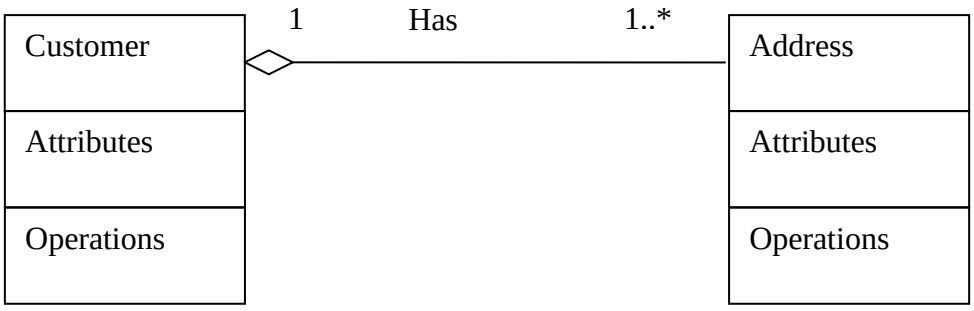
A good example of this is customer and address…



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…



So in the case of the user and card classes we need to ask the question.

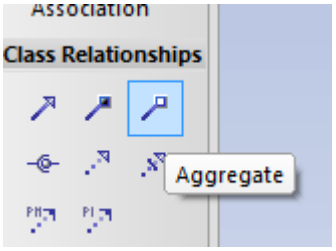
Can one exist without the other?

Would we ever want to use code for handling users independently of business cards?

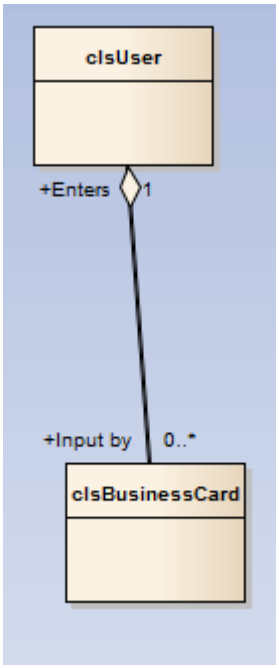
Would we ever want to manipulate business cards without worrying about who entered them?

The answer to both of these questions is probably “yes”.

In which case the association is an aggregation.



Use the aggregation tool to aggregate the two classes such that it looks like this…



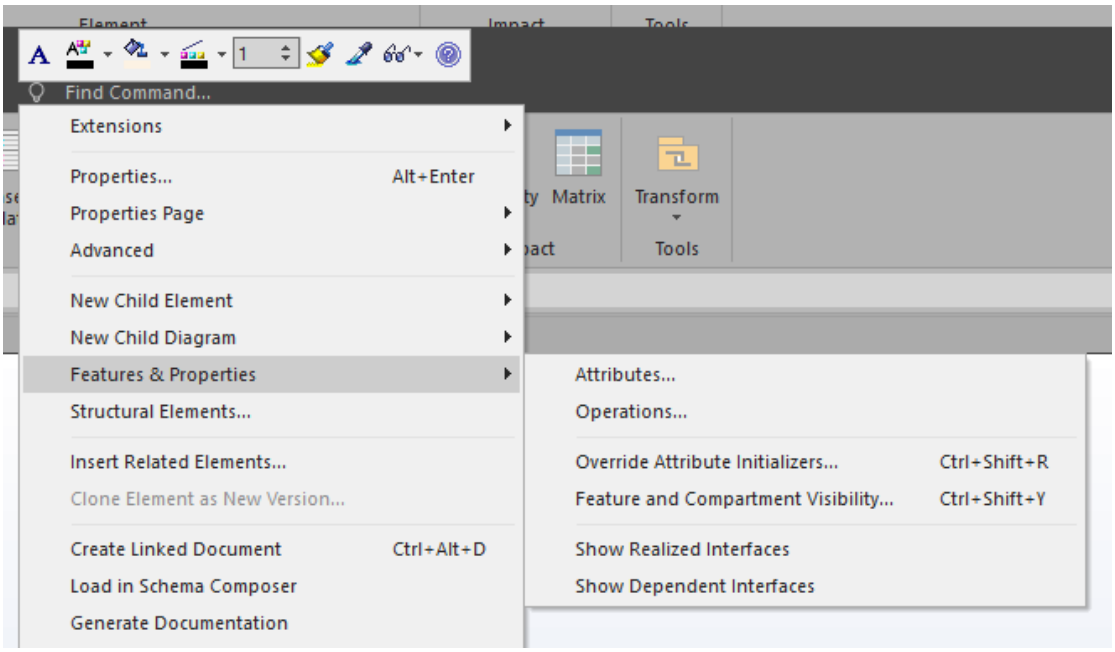
Creating Attributes

Having created the candidate classes we may now add some attributes.

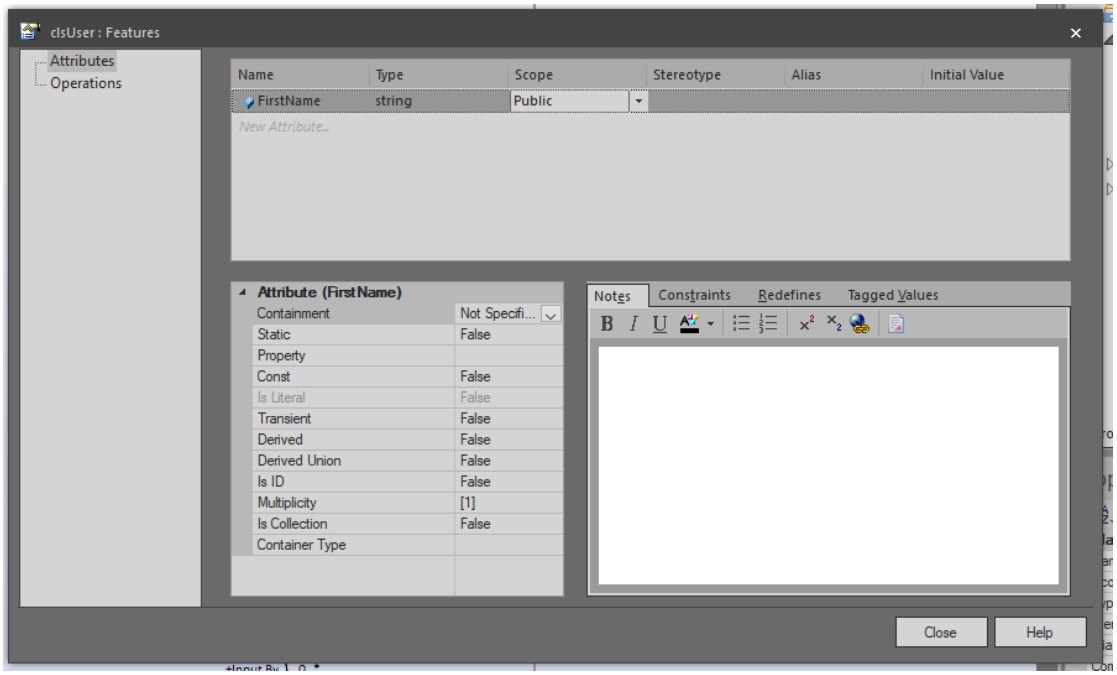
For this example we shall look at the User class.

Two obvious attribute for this class are first name and last name.

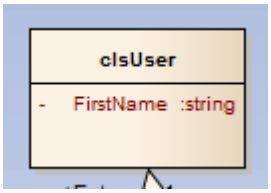
To add attributes, right click on the class and select features and properties, then attributes…



Set up the attribute FirstName like so…



Press Close and you will see the attribute in the class…



Do the same for the LastName attribute.

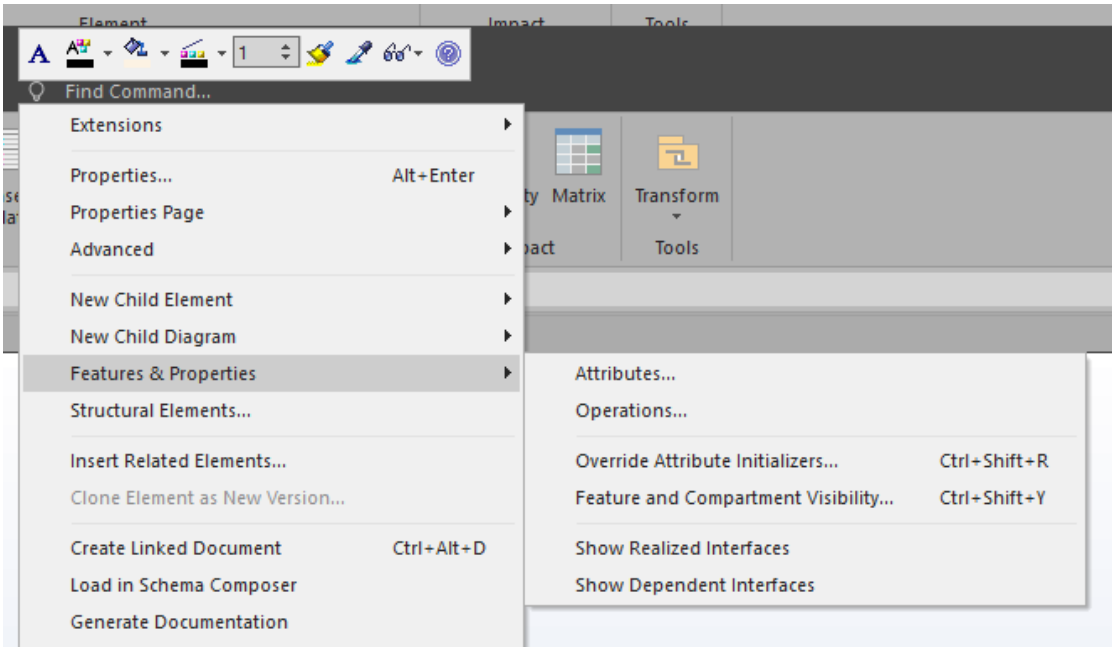
Creating Operations

Operations will ultimately become our methods when we write the code.

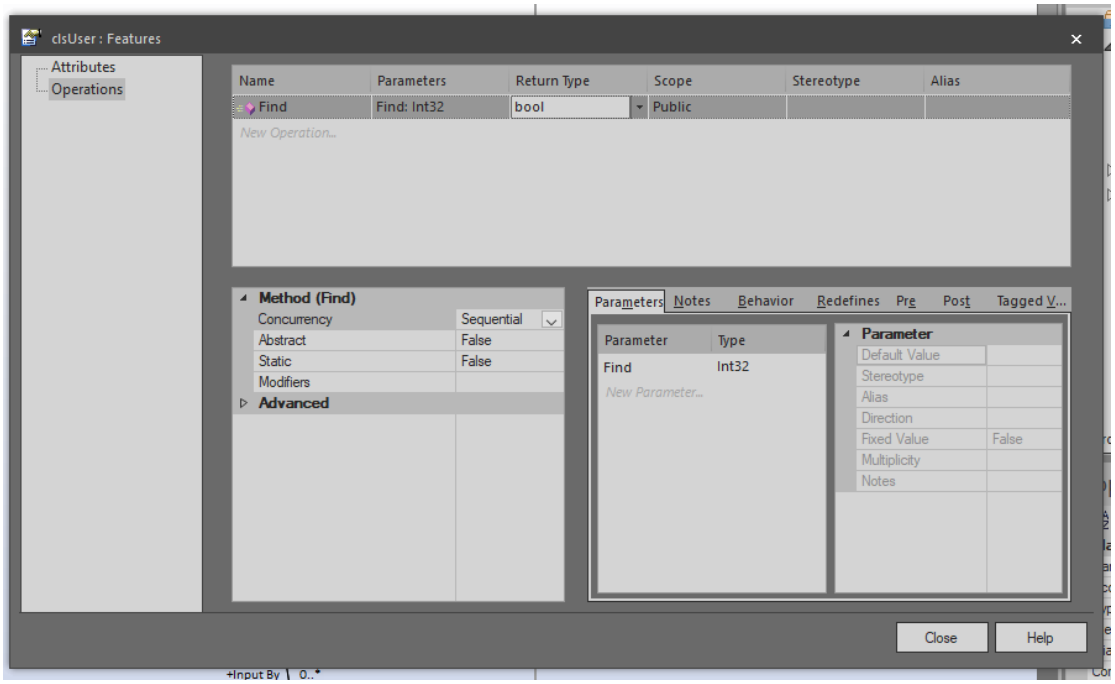
We need to think about a candidate operation for this class. How about Find?

Find(Int32 UserNo) Boolean

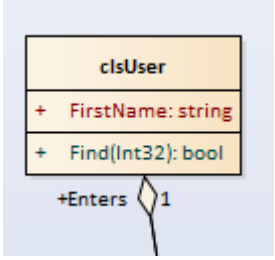
As with attributes we right click the class to add the operation…



Set up the operation like so…



The class should now look like this…



Re-Factor

What does that mean?

It means that at every stage of the project we need to stop and think about two things.

1. Is what we have done actually correct?
2. Is there a better way of doing the same thing?

The issue here is we need to stop and think about what we have done so far.

Ask the question “what are we actually recording in our system?”
Are we actually recording business cards or are we recording details of potential clients?

In this example we have recorded the data against the business card.

However we have another class called clsClient that is currently not doing anything!

Remember from week one. We are obtaining data about our clients from multiple sources not just business cards!

Exercise

As a team develop the class diagram working on the other classes. Make sure you have plenty of discussion on your interpretation of the business rules.