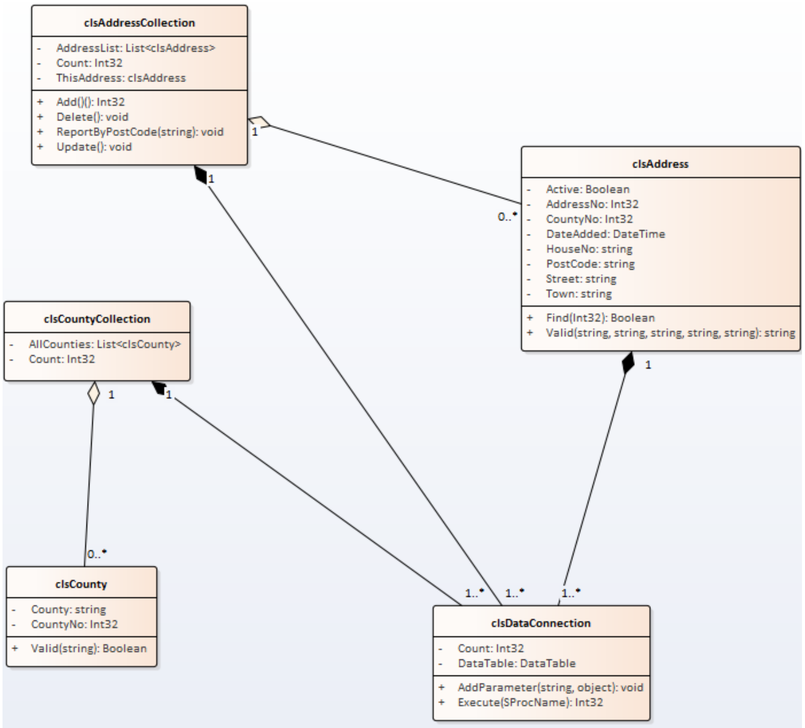
Discovering Classes

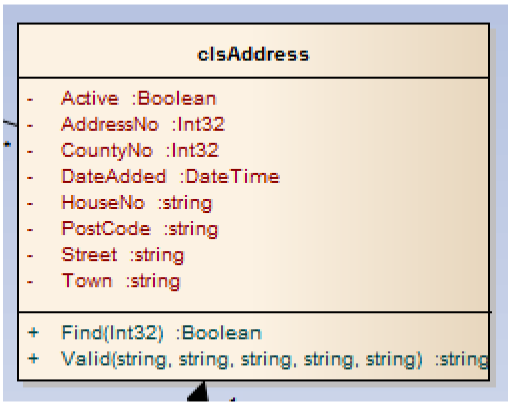
The class diagram allows us to plan out the code that will drive our system without having to worry about the actual code.

Below is the class diagram for the address book application...



When the system is completed each of the rectangles will be implemented as a class in the middle layer of the system.

For example clsAddress on the diagram looks like this...



clsAddress

- Active :Boolean
- AddressNo :Int32
- CountyNo :Int32
- DateAdded :DateTime
- HouseNo :string
- PostCode :string
- Street :string
- Town :string

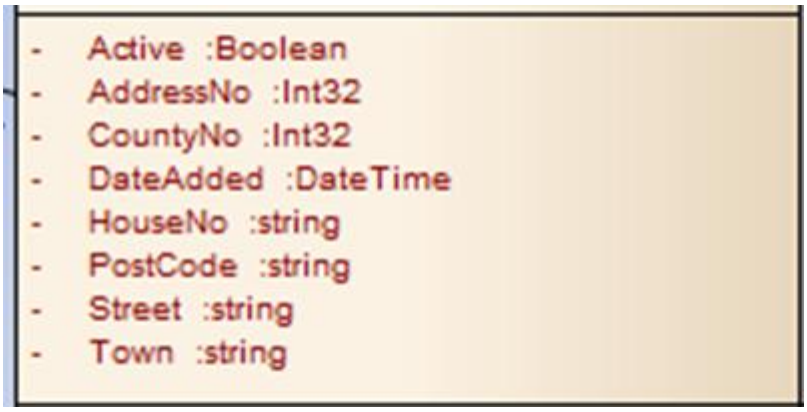
+ Find(Int32) :Boolean
+ Valid(string, string, string, string, string) :string

Notice the three sections to the class.

At the top we have the name of the class...

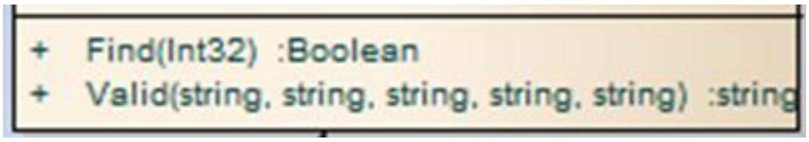
clsAddress

The next section describes the attributes of the class...



- Active :Boolean
- AddressNo :Int32
- CountyNo :Int32
- DateAdded :DateTime
- HouseNo :string
- PostCode :string
- Street :string
- Town :string

The last section describes the operations of the class...



```
+   Find(Int32)  :Boolean
+   Valid(string, string, string, string, string)  :string
```

Just to clear up a possible point of confusion...

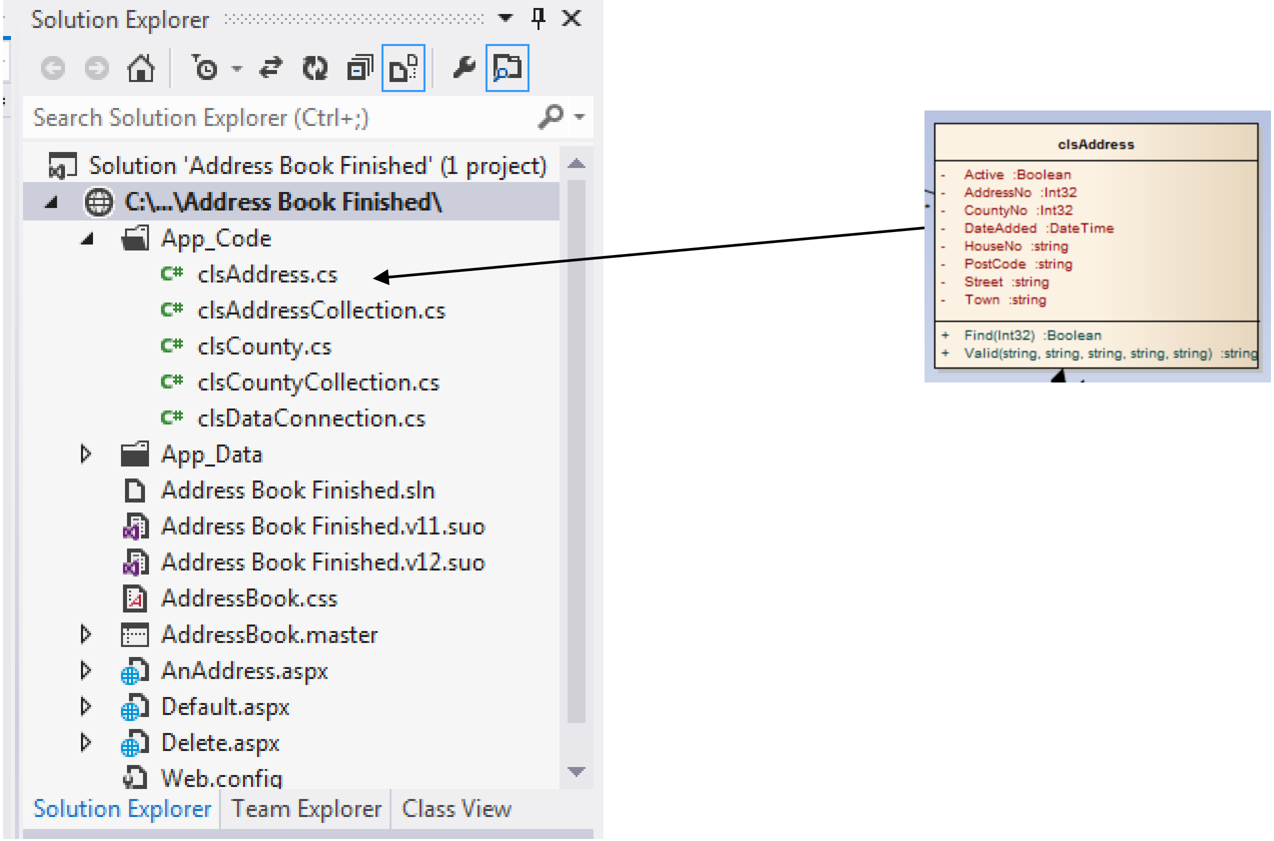
Systems analysts tend to talk in terms of attributes and operations when creating class diagrams.

Programmers tend to talk in terms of properties and methods.

In reality they amount to the same thing.

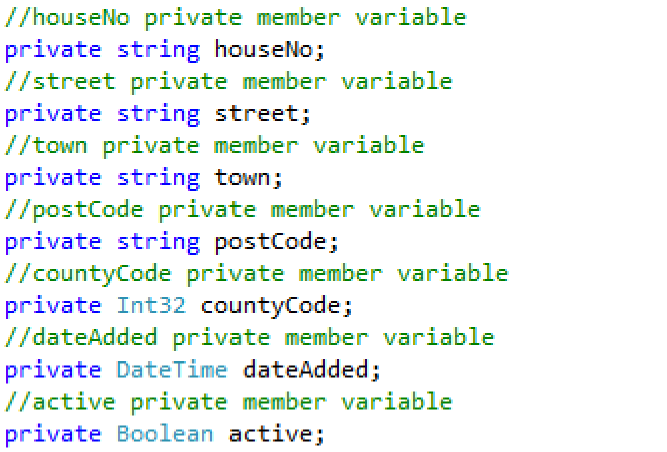
Translated into Code

In Visual Studio this class on the diagram will translate into a class in our middle layer...

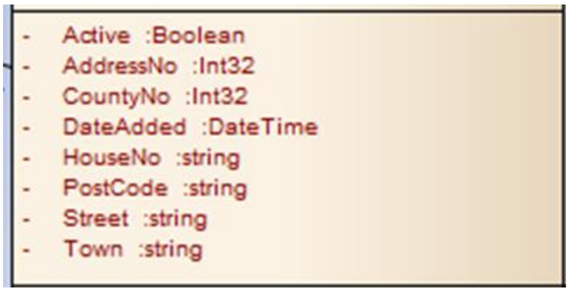


```
clsAddress
-   Active  :Boolean
-   AddressNo  :Int32
-   CountyNo  :Int32
-   DateAdded  :DateTime
-   HouseNo  :string
-   PostCode  :string
-   Street  :string
-   Town  :string

+   Find(Int32)  :Boolean
+   Valid(string, string, string, string, string)  :string
```

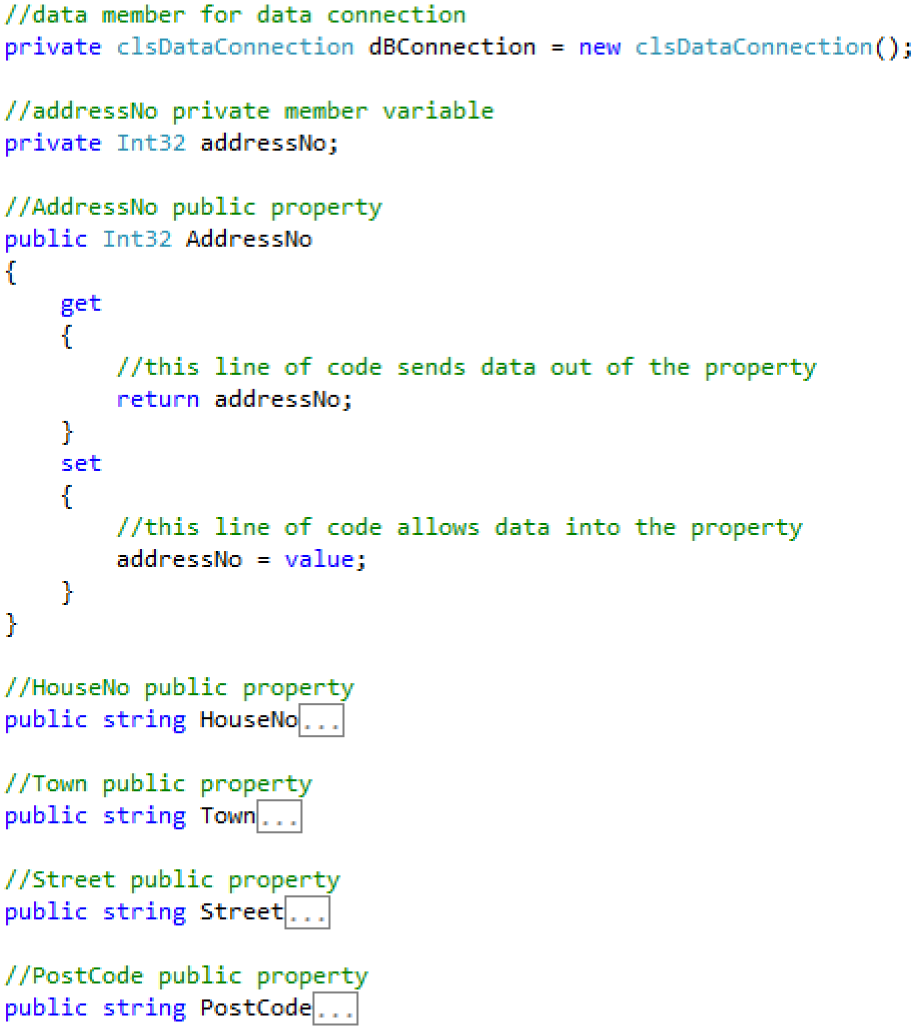
With code for each attribute implemented as properties...



```
//houseNo private member variable
private string houseNo;
//street private member variable
private string street;
//town private member variable
private string town;
//postCode private member variable
private string postCode;
//countyCode private member variable
private Int32 countyCode;
//dateAdded private member variable
private DateTime dateAdded;
//active private member variable
private Boolean active;
```

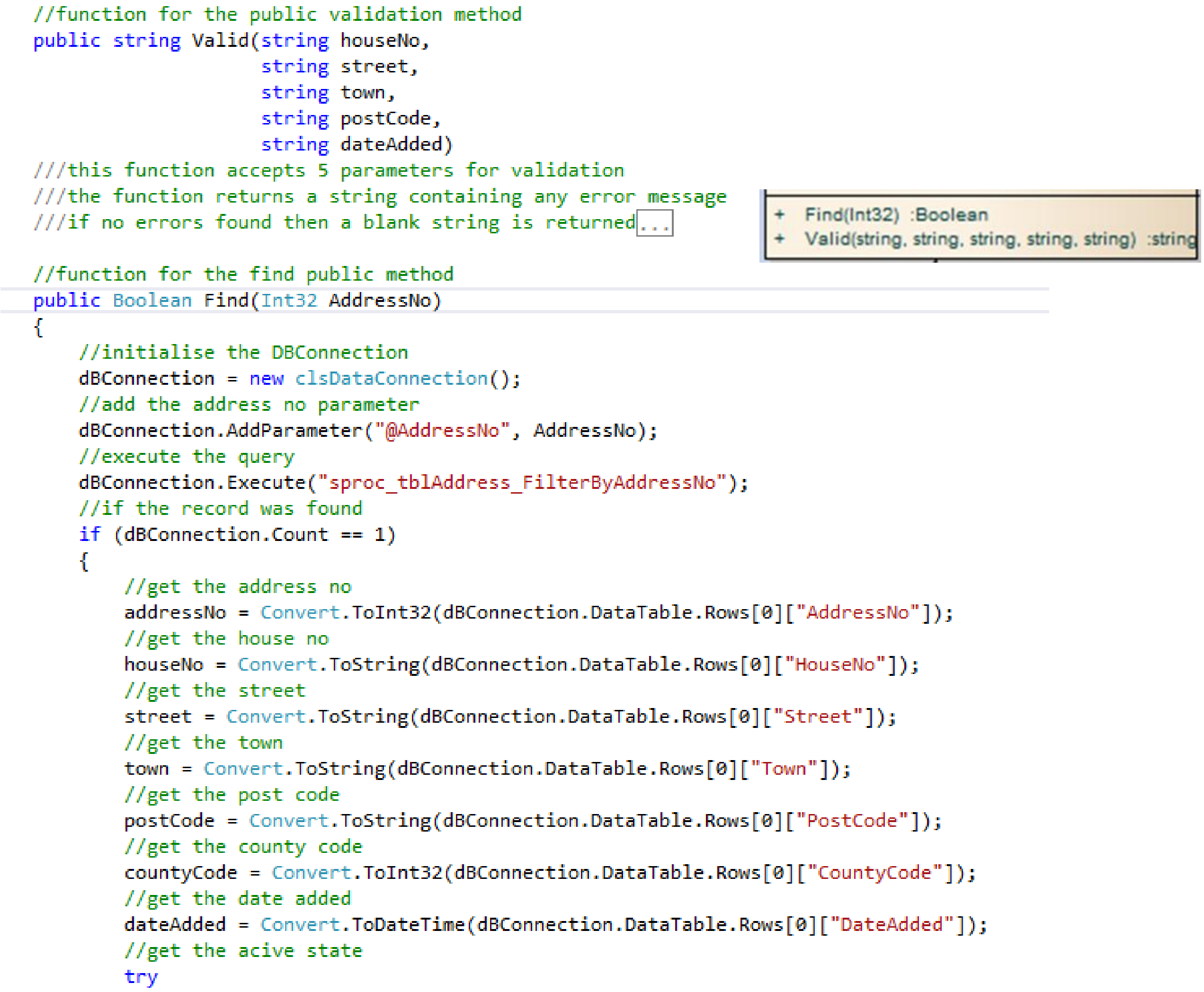


```
-   Active  :Boolean
-   AddressNo  :Int32
-   CountyNo  :Int32
-   DateAdded  :DateTime
-   HouseNo  :string
-   PostCode  :string
-   Street  :string
-   Town  :string
```



```
//data member for data connection
private clsDataConnection dBConnection = new clsDataConnection();

//addressNo private member variable
private Int32 addressNo;

//AddressNo public property
public Int32 AddressNo
{
    get
    {
        //this line of code sends data out of the property
        return addressNo;
    }
    set
    {
        //this line of code allows data into the property
        addressNo = value;
    }
}

//HouseNo public property
public string HouseNo...

//Town public property
public string Town...

//Street public property
public string Street...

//PostCode public property
public string PostCode...
```

And the operations implemented as methods...



```
//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)
{
    //initialise the DBConnection
    dBConnection = new clsDataConnection();
    //add the address no parameter
    dBConnection.AddParameter("@AddressNo", AddressNo);
    //execute the query
    dBConnection.Execute("sproc_tblAddress_FilterByAddressNo");
    //if the record was found
    if (dBConnection.Count == 1)
    {
        //get the address no
        addressNo = Convert.ToInt32(dBConnection.DataTable.Rows[0]["AddressNo"]);
        //get the house no
        houseNo = Convert.ToString(dBConnection.DataTable.Rows[0]["HouseNo"]);
        //get the street
        street = Convert.ToString(dBConnection.DataTable.Rows[0]["Street"]);
        //get the town
        town = Convert.ToString(dBConnection.DataTable.Rows[0]["Town"]);
        //get the post code
        postCode = Convert.ToString(dBConnection.DataTable.Rows[0]["PostCode"]);
        //get the county code
        countyCode = Convert.ToInt32(dBConnection.DataTable.Rows[0]["CountyCode"]);
        //get the date added
        dateAdded = Convert.ToDateTime(dBConnection.DataTable.Rows[0]["DateAdded"]);
        //get the acive state
        try
```

```
+  Find(Int32)  :Boolean
+  Valid(string, string, string, string, string)  :string
```

The class diagram allows us to plan out the code concentrating on what the code should do without worrying about how it will do it.

Initial class diagram

To make a start on the class diagram the event table provides us with a useful starting point.

Revisiting the event table for the address book we need to focus on the Object column as this will give us some clues as to what objects exist within the system.

Subject	**Verb**	**Object**	**Response**
User	Views	Address List	Addresses are listed by the system
User	Filters	Address List	Address list is filtered based on pattern
User	Adds	Address	Address is added to the system
User	Updates	Address	Address is updated on the system
User	Deletes	Address	Address is deleted from the system
System	Validates	Address	Address data is accepted or error is displayed

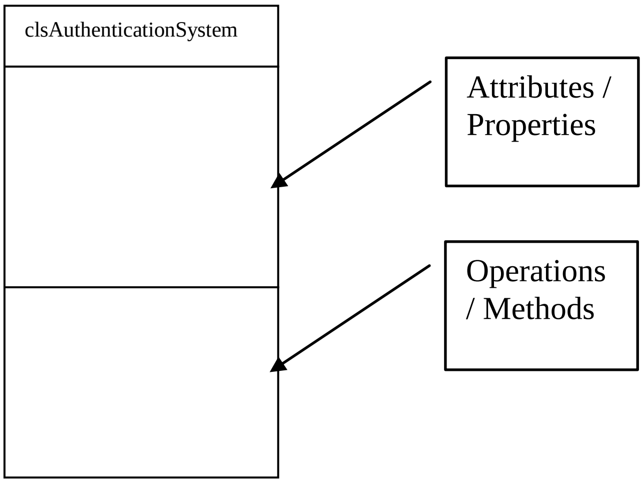
It is also possible to identify classes by highlighting all of the nouns in the system description…

At the end of each year students have purchased a number of books that are surplus to requirements by the end of the year / course. In order to recycle the books and possibly make a little money a system is required allowing students to sell their books on to students following them on lower years. The application will be web based allowing students to access sales and account details at home. Students will sign up to the site providing contact details. Once authenticated on the system a student will be able to add books that they no longer require. Initially books will be viewable only to the student themselves. To sell on a book the student must take the book to the book shop on the ground floor of Gateway house. The book will be handed to the system administrator (Brenda) who will place the book into stock, provide a receipt for the student and then flag the book on the system as "in stock". Now that the book is in stock it is visible to other students on the system who may reserve the book for collection. Once reserved the purchasing student must visit the book shop and pay for the book. They will be issued with the book along with a receipt or they have the option of cancelling the transaction if the book was not what they expected (wrong edition of poor condition). As soon as the transaction is completed the seller of the book is notified. They need to visit the bookshop to claim the money paid by the buyer.

Like many of the documents at this stage we make a rough and ready stab at candidates. We may refine the list later on.

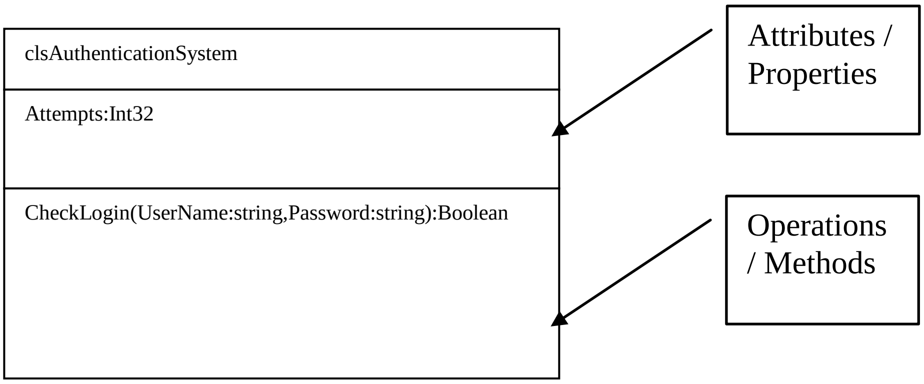
Authentication Example

One part of the Project Bank we are going to have to consider is the process of authentication. We shall have a go at thinking about this and put together some candidate classes.



What properties and methods might such a class have?

How about…



Is it correct?

Who knows at this stage!

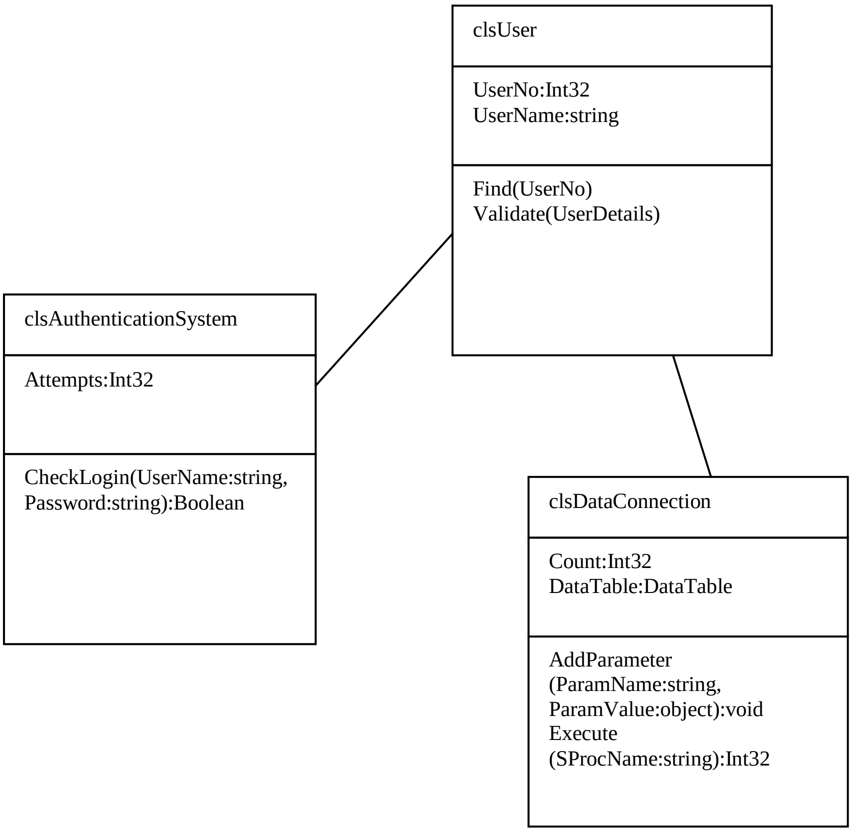
That isn’t the point.

Being right first time is not the object of the exercise. The object of the exercise is to make us think about how we might model the system.

After a bit more thought we might decide that there is going to be some sort of data store required for the password username data.

Let’s keep it simple and assume that it is going to be a user table in a database.

This could mean that we extend the design just a bit more to include some other classes.



Is it right?

Who knows at this stage – let’s just go for it!

How do we find 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.

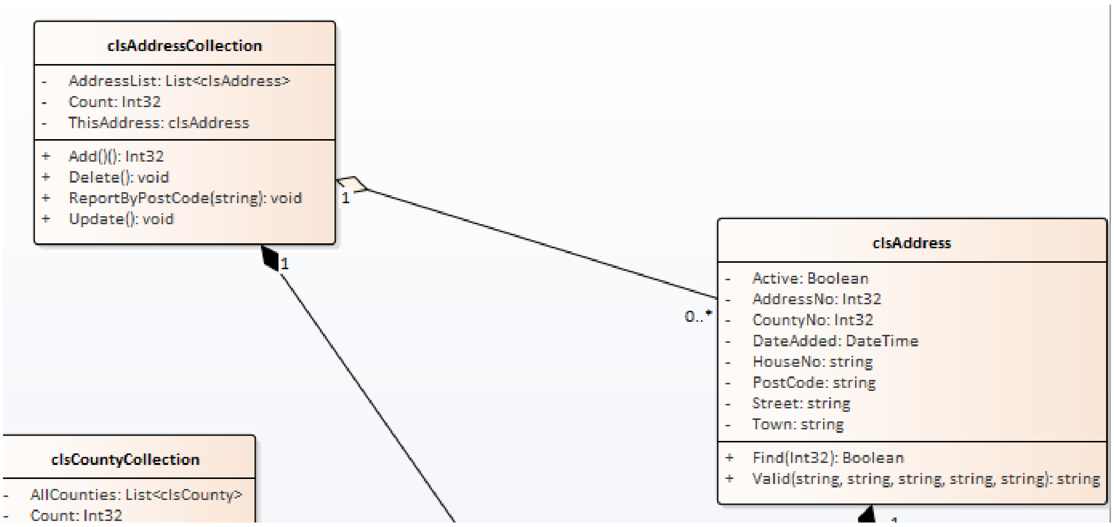
Multiplicity

Having had a first go at drawing up the classes we need to start thinking about how they are related to each other.

The first factor to think about is that of multiplicity.

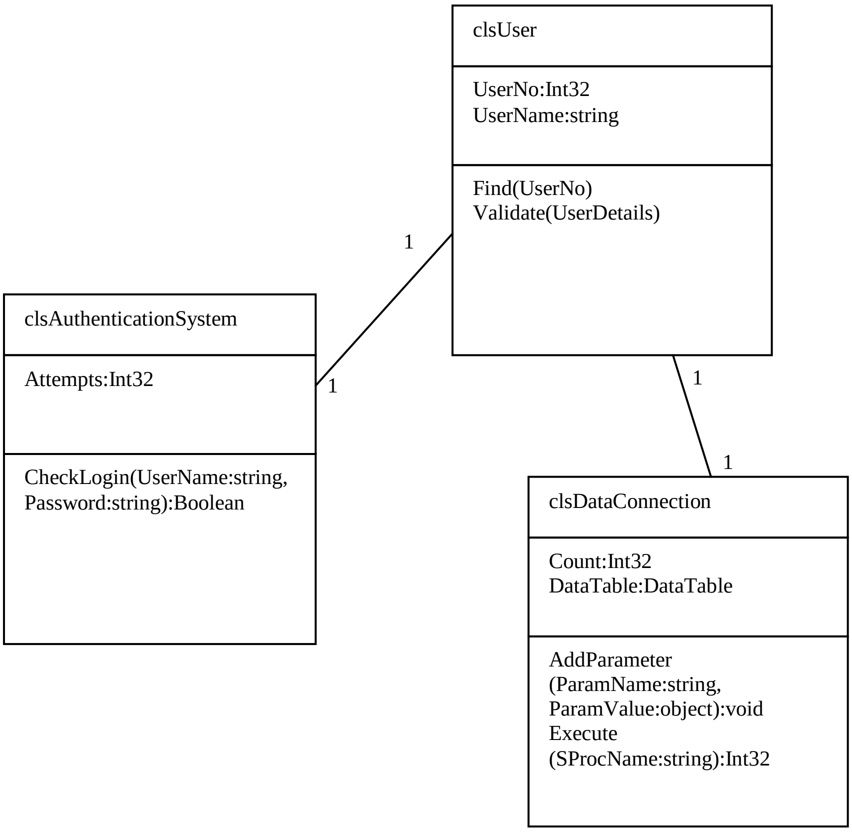
Multiplicity tells us how many of one class may be used by another class.

In the address book example



Tells us that an address collection has zero or many addresses:1 --- 0..*

In the authentication example we might go for something like this…

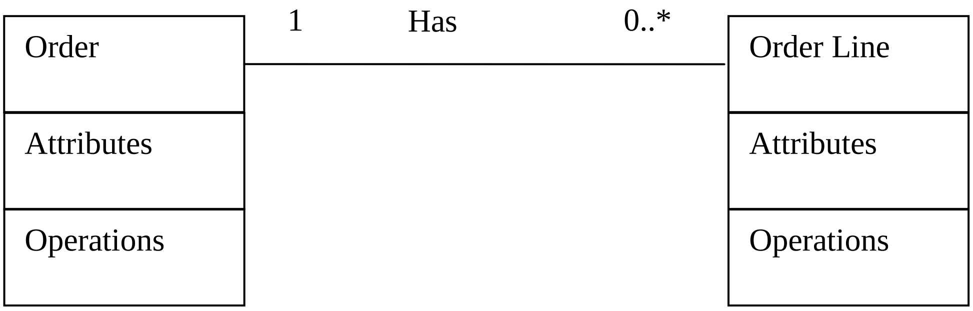


If we have identified the multiplicity of the relationship we next need to think about the type of relationship.

Composition

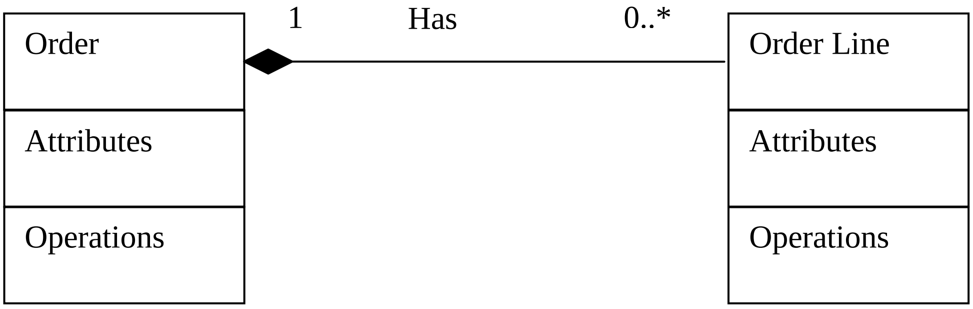
Two classes are related via composition when you cannot have an instance of one class without the parent class.

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…

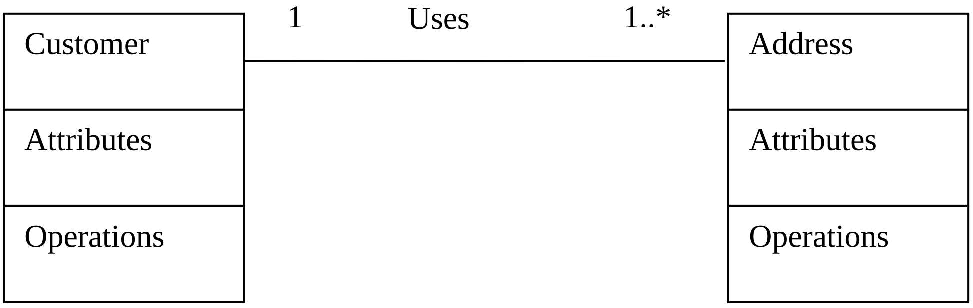


Aggregation

Another relationship 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?”

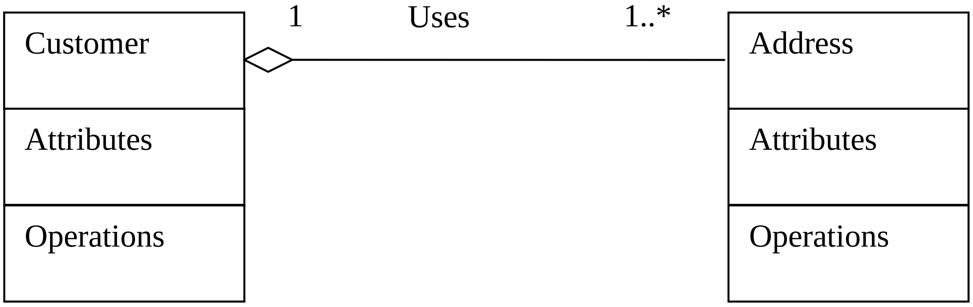
A good example of this is customer and address…



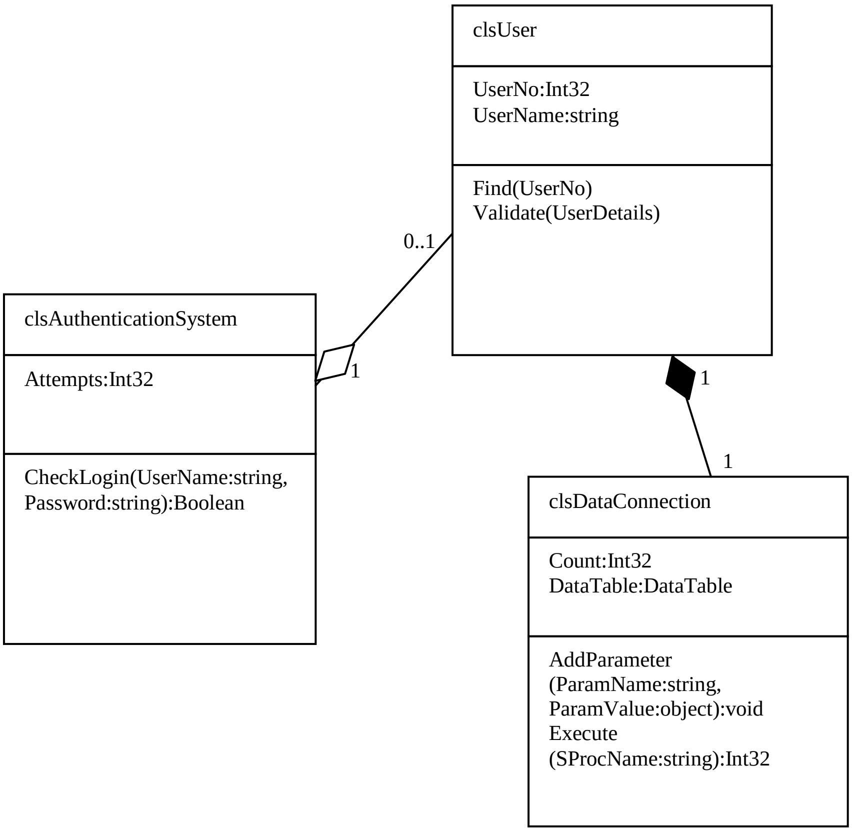
If all of the customers were to vanish from our store would addresses cease to exist?

Obviously the answer is they would not!

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



If we identify the composite and aggregated relationships in the diagram we get something like this…

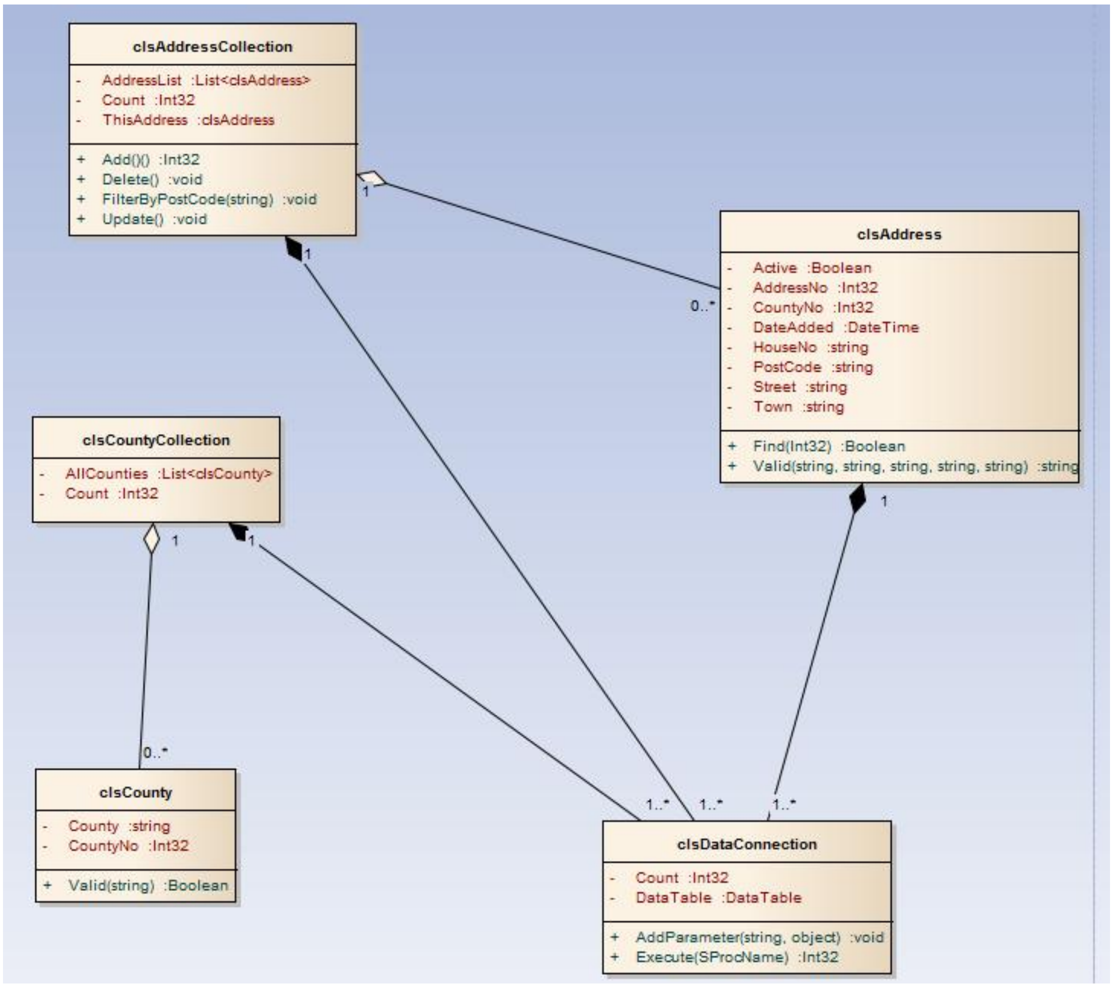


Collection Classes

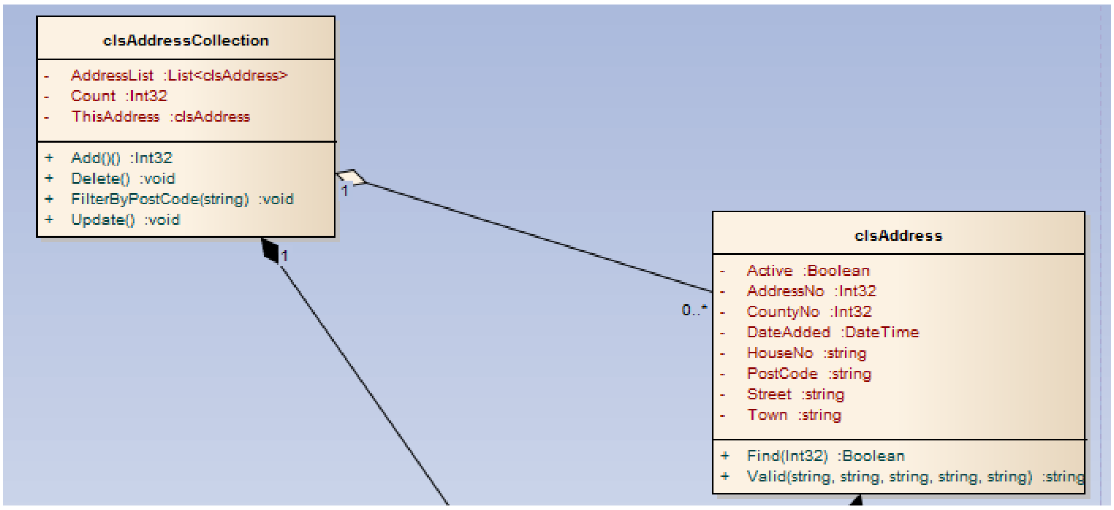
As we draw up our class diagrams collection classes may be assumed to be required at some point.

A collection class is a class that specialises in the management of a number of sub class items.

In this diagram...



clsAddressCollection is the collection class for clsAddress



The collection class typically offers a set of methods for add, edit delete etc.

There is no point re-inventing the wheel!