

Creating the Middle Layer Part 1

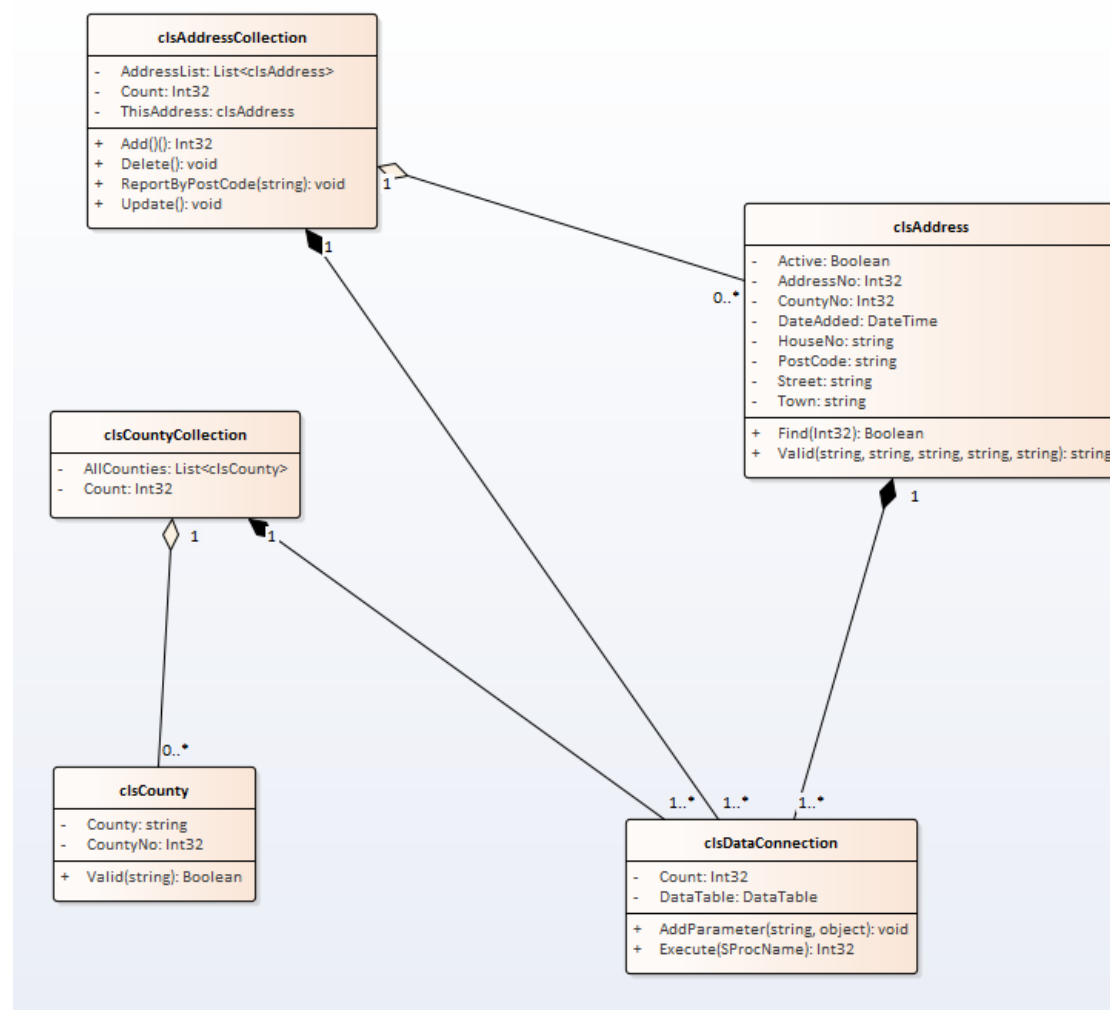
In the last two sessions, we have created the data layer and made a start on the presentation layer for the address book.

In this session, we will make a start on creating the middle layer linking the presentation layer to a middle layer class file. In the next session, we will look at linking the data layer to the middle layer. By the end of the next session you should be able to enter a value at the presentation layer and see the record with that unique identifier deleted in the data layer.

You will need to open your work so far (or download the work so far from the module web site).

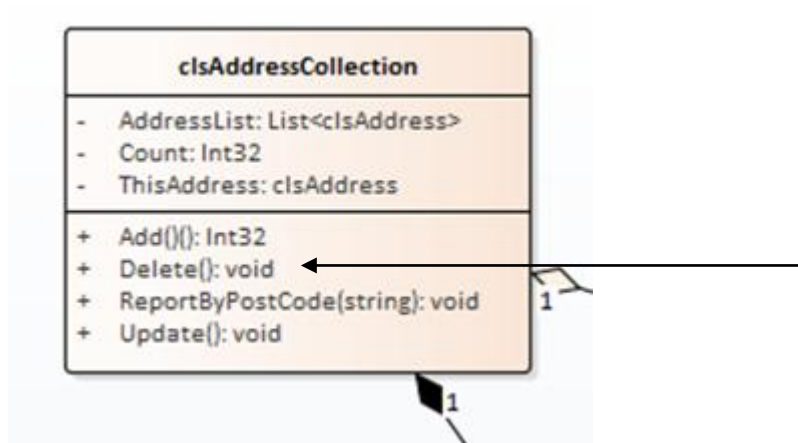
The Class Diagram

The class diagram provides us with a graphical overview of the parts of the system we will create over the year.

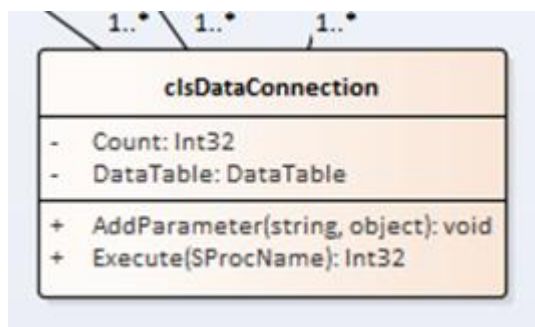


We won't be building this overnight.

We will start by building the address collection class initially allowing us to delete records.



We will also make use of a pre-prepared data connection class allowing us to link to the database.



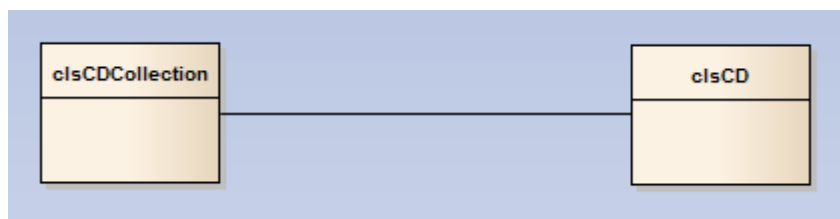
So, what is a Collection Class?

A collection class is a special kind of class that is responsible for managing lists of multiple items. You will create a lot of collection classes in object oriented programming.

The collection class handles such things adding, updating and deleting records.

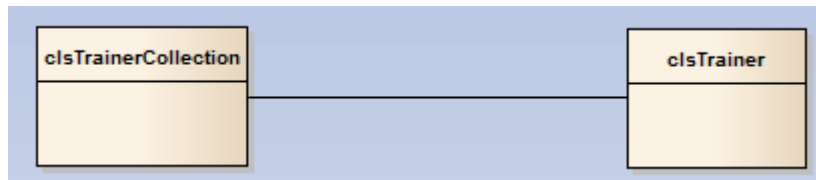
So...

If you are creating a web site dealing with CDs you would have a CD class and a CD Collection Class like so...



Or...

If your site is about trainers you would have two classes for Trainer and a Trainer Collection like so...



You will need to design your classes based on whatever topic it is you are creating for your system.

Creating the Middle Layer

Objects and Classes

There are two ways that we may create objects in a language. Some objects have been written for us as part of the programming language, other objects we need to write ourselves.

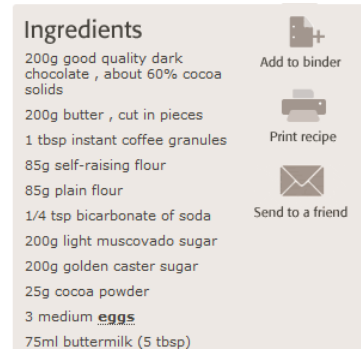
To create the middle layer for our application we are going create objects of our own and use an object written specifically for this module.

Before we may create the objects, we need to create the class definitions. (Remember a class file is just a text file containing functions. The functions create the methods and properties for the object.)

The relationship between a class and an object is rather like the relationship between a cake and a recipe.



Object



Class

We can't eat a recipe but we can eat a cake. Without the recipe, we won't know how to make a particular cake.

Creating the Class Files Based on the Class Diagram

We will not really be teaching how to draw class diagrams in this module but some discussion about them will be useful.

Much of the work that we will undertake in this module is really database programming.

At the backend of our system there is a database we want to control and we need to write classes to do this.

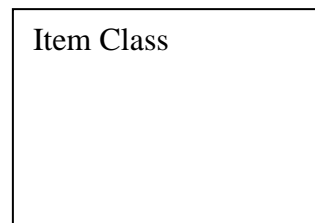
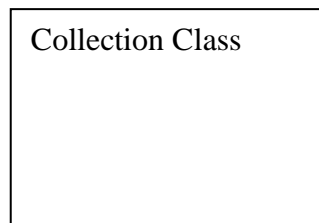
So far you will have created a table called tblAddresses...

	AddressNo	HouseNo	Street	Town	PostCode	CountyCode	DateAdded	Active
	2	1	Some Street	Leicester	LE1 1BE	35	07/09/2012	True
	3	22	The Road	Nottingham	N19 6EF	48	07/08/2012	True
	4	33	High Street	Leicester	LE1 6FG	35	07/08/2012	True
	5	22	The Road	Nottingham	N19 6EF	48	07/08/2012	True
▶	14	128	Another Stree	Leicester	LE3 1EE	35	20/05/2013	NULL
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

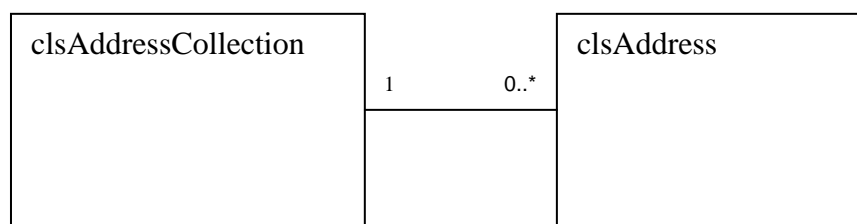
To control this table's data, we will need two class definitions.

As mentioned above we will need a collection class to control the data in the table performing such functions as add, edit, delete and list.

We will also need a class to store the data for a single record.

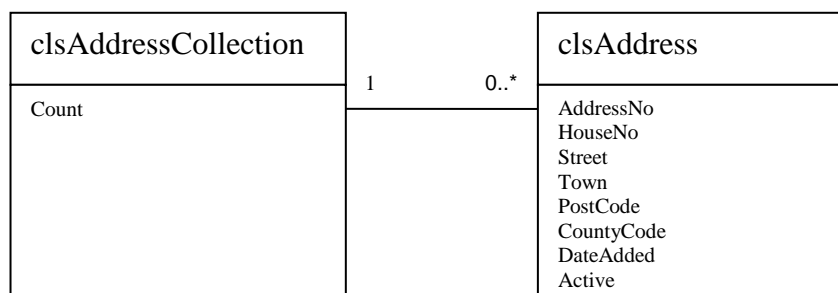


The collection class will be named `clsAddressCollection` and the item class will be named `clsAddress`.

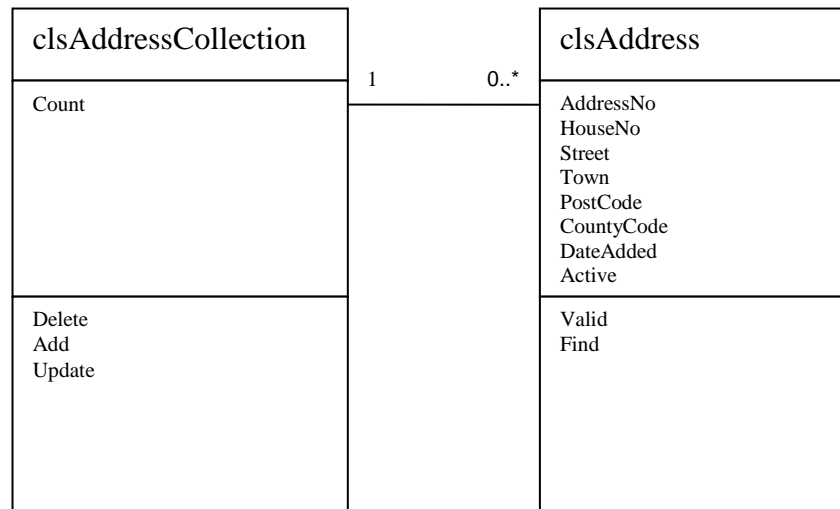


The relationship between the two classes is 1 to zero or many (0..*). (We could have an address book with no addresses in it thus the 0)

The item class `clsAddress` will have a set of properties which map to the fields in the table. The collection class will have a single property called `Count`.

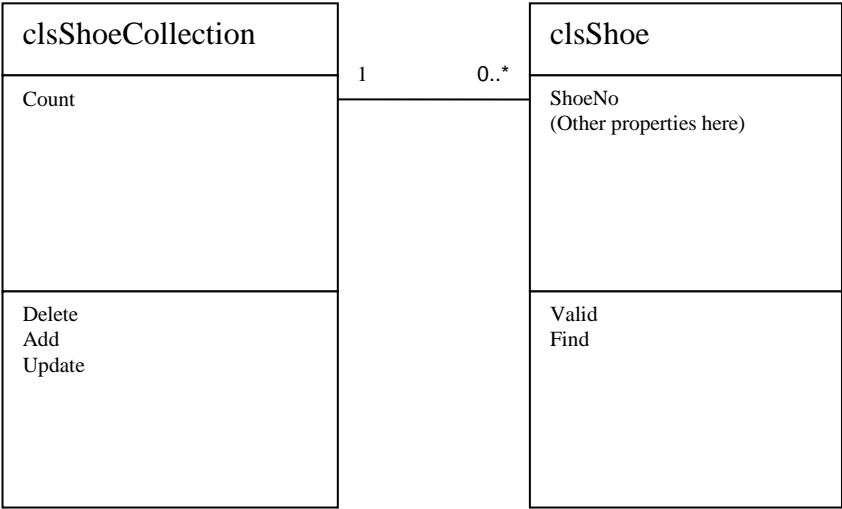


The collection class will have a set of methods allowing us to manipulate the data in the table. The item class will have two methods for find and validation.



Remember you will need to modify this design based on whatever topic you have selected.

For example, if you are creating a site about shoes the class diagram would look something like this...



Creating Your First Object – the Collection Class clsAddressCollection

A class definition is a text file containing the code that tells an object how to behave. The class contains functions that make the methods and properties work.

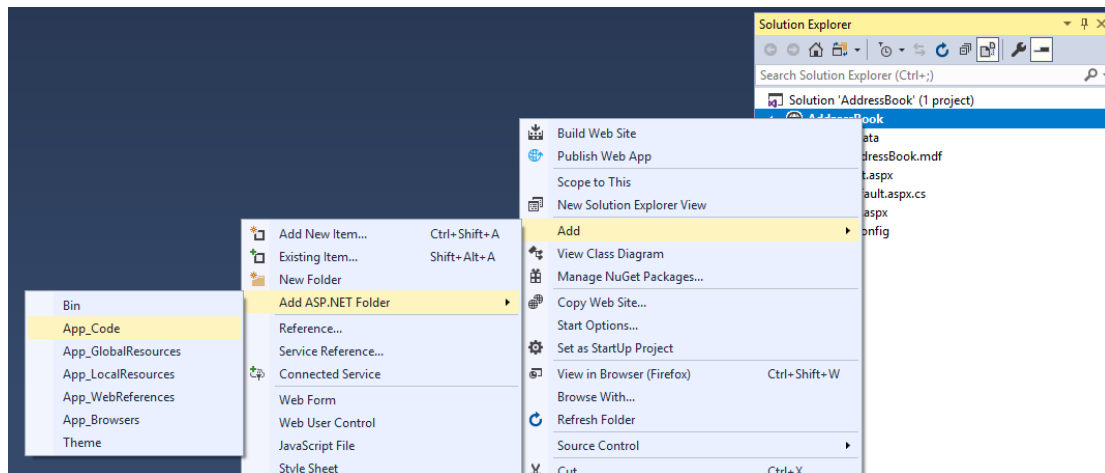
Before we create the definition, we need to think about what it is the class is going to do.

In this example, we want the object to control addresses. That being the case we will give the object the name `clsAddressCollection`. We try to base class names on real world “things” if possible as it makes it easier to understand what they do. Notice also that we add the prefix to the class name “cls” to remind us (the programmer) that we are dealing with a class. Notice also that the class file name contains the words “AddressCollection” so we have a fair idea of the kind of thing it does. (If we were writing a class file to manage kettles we would call it `clsKettleCollection`.)

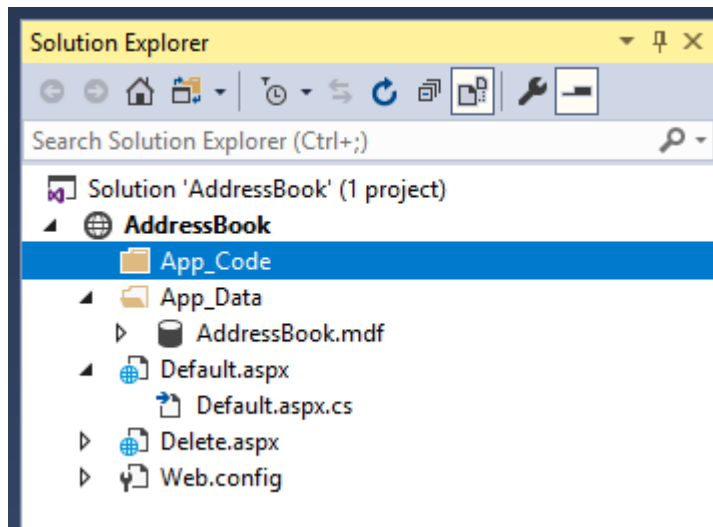
Creating the App_Code Folder

In a similar way to the `App_Data` folder it is a good idea to keep code on the server in such a way that it is secure. To do this we will create a special folder called `App_Code`.

Create the new folder like so...

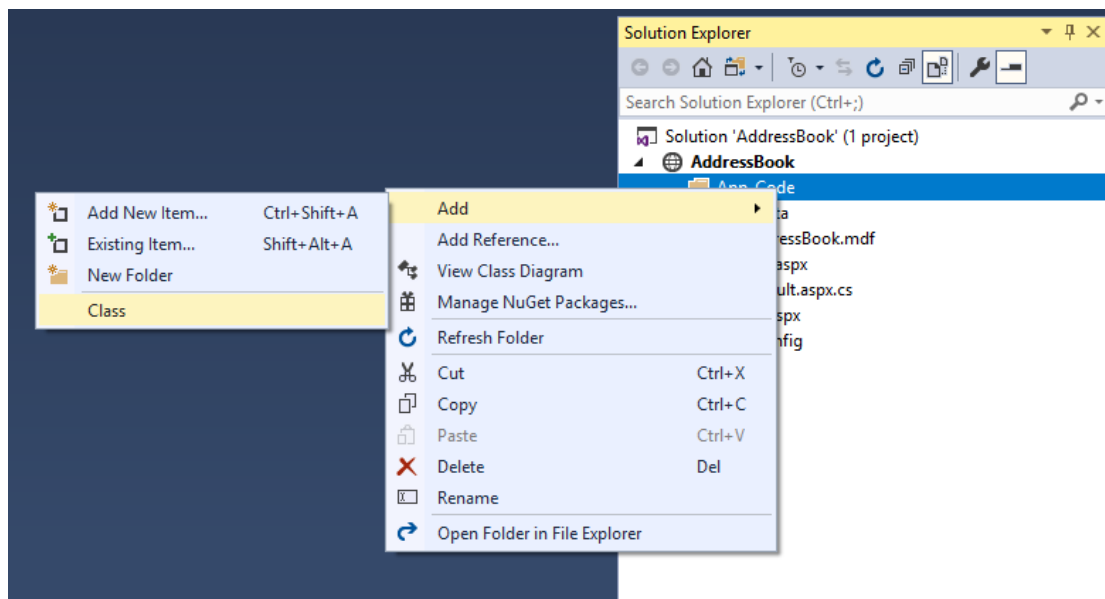


This creates the folder...

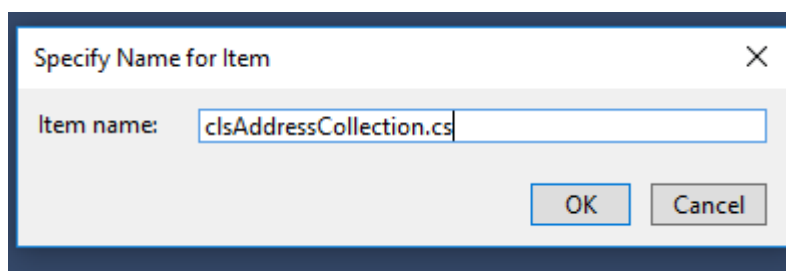


Creating the First Class

To create the new class file right click on the App_Data folder and select Add – Class.

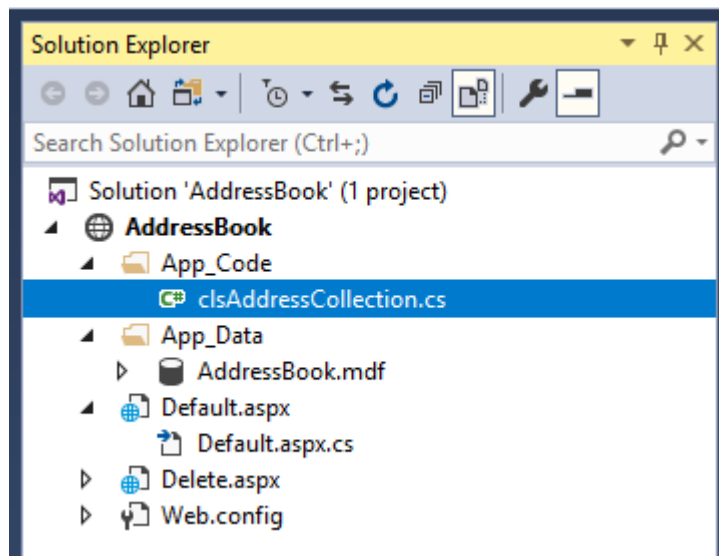


Enter the name of the class as clsAddressCollection.

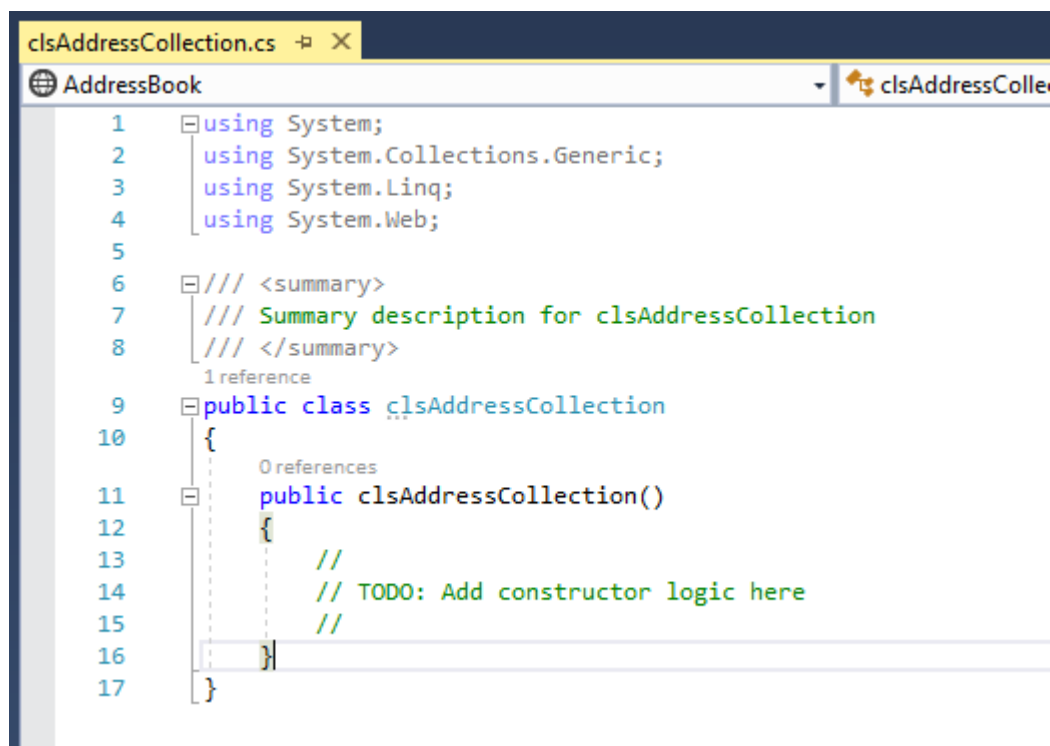


Then press OK.

You should now see the new class file in your solution explorer.

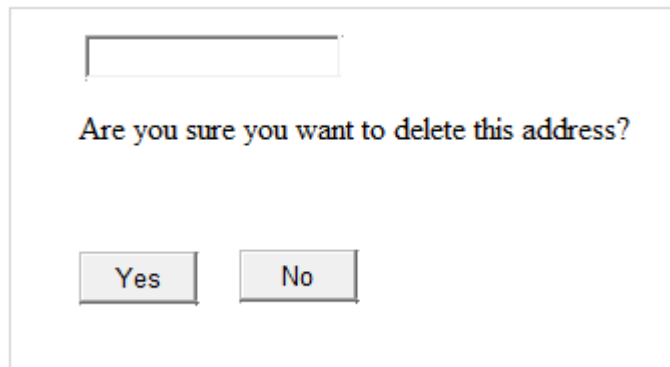


Visual Studio also creates the outline code for your class.



Even though we don't have any code written yet, close the class definition and we will now create an instance of an object based on this class.

Open the delete form you created last week.



Now double click on the Yes button to access the event handler for the button. You should have something like this from the previous week.

```
protected void btnYes_Click(object sender, EventArgs e)
{
    //this line of code re-directs back to the main page
    //Response.Redirect("Default.aspx");

    //declare a variable in ram to store the data from the interface
    Int32 AddressNo;
    //use the assignment operator to copy the data from the interface to the ram converting the data type
    AddressNo = Convert.ToInt32(txtAddressNo.Text);
    //display a message concatenating the text and the data in the variable
    lblAddressNo.Text = "You deleted address number " + AddressNo;
}
```

Delete all the code for the function.

```
protected void btnYes_Click(object sender, EventArgs e)
{
    //this line of code re-directs back to the main page
    //Response.Redirect("Default.aspx");

    //declare a variable in ram to store the data from the interface
    Int32 AddressNo;
    //use the assignment operator to copy the data from the interface to the ram converting the data type
    AddressNo = Convert.ToInt32(txtAddressNo.Text);
    //display a message concatenating the text and the data in the variable
    lblAddressNo.Text = "You deleted address number " + AddressNo;
}
```

Your event handler function should look like this.

```
protected void btnYes_Click(object sender, EventArgs e)
{
}
}
```

What next?

What we are going to do is create an instance of an object based on the class definition we have just made above. Creating this object in the presentation layer will create a link between this layer and the middle layer.

To do this, create the following line of code in the event handler.

```
protected void btnYes_Click(object sender, EventArgs e)
{
    ///this function handles the click event of the Yes button

    //create an instance of the class clsAddressBook called MyAddressBook
    clsAddressCollection MyAddressBook = new clsAddressCollection();
}
```

There is a lot going on in this single line of code. We shall explore the detail later but for now understand that this line gives us an object (a word) which we may use to send instructions to the address book.

To send instructions we need two things. We need methods and we need properties. In this example we shall create a method called “.Delete”.

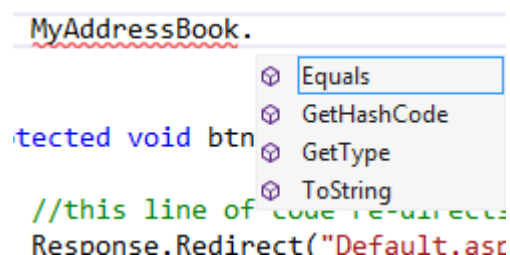
In the next line of code type the name of the object like so...

```
protected void btnYes_Click(object sender, EventArgs e)
{
    ///this function handles the click event of the Yes button

    //create an instance of the class clsAddressBook called MyAddressBook
    clsAddressCollection MyAddressBook = new clsAddressCollection();

    MyAddressBook
}
```

Now press the full stop key on the keyboard to see what methods and properties the object has...

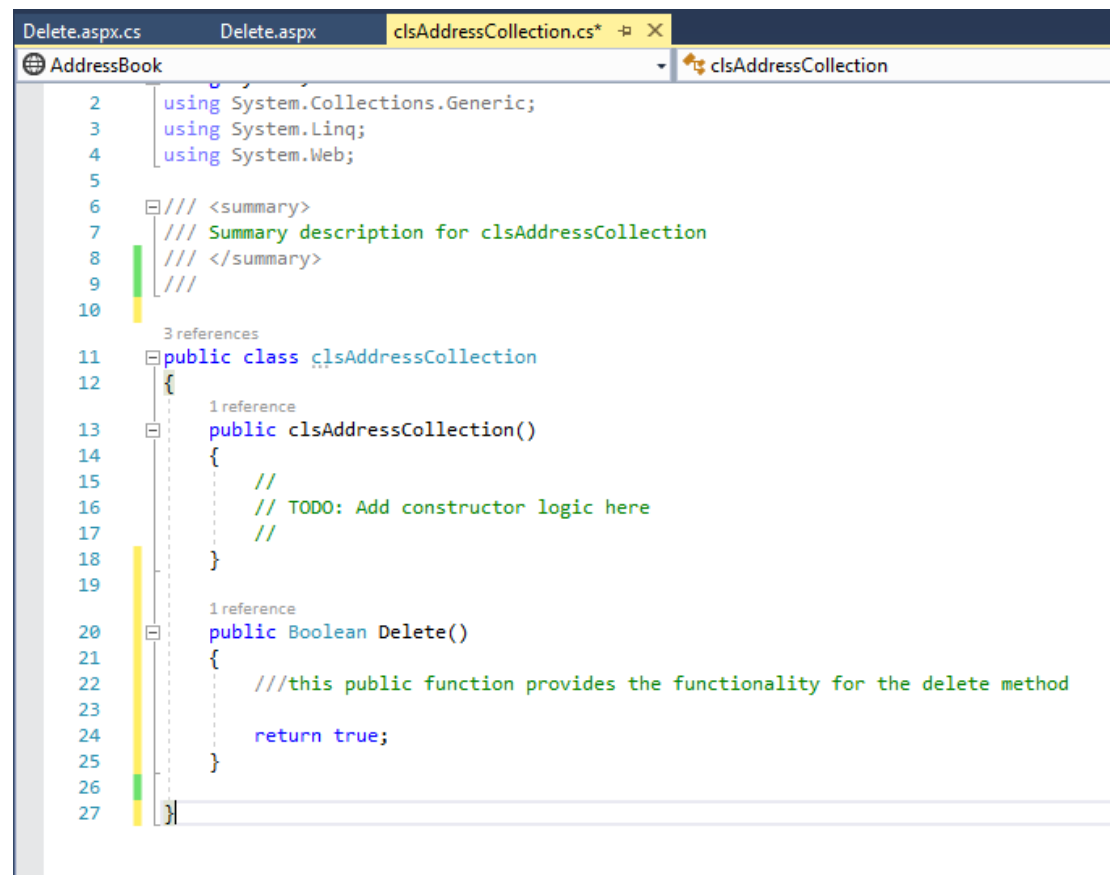


```
MyAddressBook.
protected void btn
//this line of code re-directs
Response.Redirect("Default.asp"
```

What you see are the default methods automatically created for the object. What we don't see is a Delete method.

To create the delete method we need to create a function in our class definition clsAddressCollection.

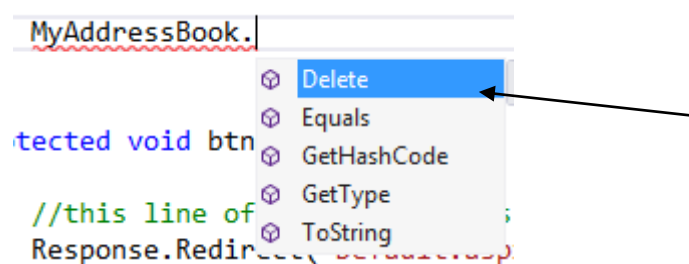
Open the class definition and modify it like so...



```
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Web;
5
6 /// <summary>
7 /// Summary description for clsAddressCollection
8 /// </summary>
9 ///
10
11 public class clsAddressCollection
12 {
13     public clsAddressCollection()
14     {
15         //
16         // TODO: Add constructor logic here
17         //
18     }
19
20     public Boolean Delete()
21     {
22         ///this public function provides the functionality for the delete method
23
24         return true;
25     }
26 }
27
```

Now go back to your presentation layer code and do the same as we did above.

Type the name of the object and enter a full stop after the object's name...



```
MyAddressBook.
protected void btn
//this line of
Response.Redirect
```

- Delete
- Equals
- GetHashCode
- GetType
- ToString

You should see the Delete method listed as one of the available methods for the object.

Public v Private

Go back to the class definition and change the key word “Public” to “Private” like so...

```
private Boolean Delete()
{
    ///this public function provides the functionality for the delete method

    //set the return value for the function
    return true;
}
```

Now try to access the object’s Delete method from the presentation layer, what happens?

You will find that the Delete method isn’t available in the presentation layer.

The key words public and private allow us to control which bits of our class definition we turn on and off for any objects based on that class.

Change the key word back to public so that our method is available as part of our object.

We shall now add a bit more code to the presentation layer.

Modify the event handler like so...

```
0 references
protected void btnYes_Click(object sender, EventArgs e)
{
    ///this function handles the click event of the Yes button

    //create an instance of the class clsAddressBook called MyAddressBook
    clsAddressCollection MyAddressBook = new clsAddressCollection();
    //declare a variable to store the address number to delete
    Int32 AddressNo;
    //copy the data from the interface to the RAM converting the data to Int32
    AddressNo = Convert.ToInt32(txtAddressNo.Text);
    //invoke the delete method of the object passing it the data entered by the user
    MyAddressBook.Delete();
}
0 references
```

What we are doing with this code is the following...

1. The user types a number into the text box
2. The user clicks the yes button
3. The event handler is triggered
4. We create an instance of our address object called MyAddressBook
5. We declare a variable called AddressNo of type Int32
6. We copy the data from the interface to the RAM using the assignment operator
7. We invoke the Delete method of our object MyAddressBook

Can you spot the first problem?

To really understand what is going on we shall place a breakpoint in the code so that we may pause execution of the program and see exactly what is going on.

Place the cursor on the first line of code and press F9.

```
protected void btnYes_Click(object sender, EventArgs e)
{
    ///this function handles the click event of the Yes button

    //create an instance of the class clsAddressBook called MyAddressBook
    clsAddressCollection MyAddressBook = new clsAddressCollection();
    //declare a variable to store the address number to delete
    Int32 AddressNo;
    //copy the data from the interface to the RAM converting the data to Int32
    AddressNo = Convert.ToInt32(txtAddressNo.Text);
    //invoke the delete method of the object passing it the data entered by the user
    MyAddressBook.Delete();
}
```

The line will be highlighted in red to indicate that there is a breakpoint.

Run the program and enter a record number to be deleted (It doesn't matter if the record exists in the database at this stage.)

Are you sure you want to delete this address?

Yes

No

Press the Yes button and the breakpoint will pause the running of the program...

```
protected void btnYes_Click(object sender, EventArgs e)
{
    ///this function handles the click event of the Yes button

    ///create an instance of the class clsAddressBook called MyAddressBook
    clsAddressCollection MyAddressBook = new clsAddressCollection();
    ///declare a variable to store the address number to delete
    Int32 AddressNo;
    ///copy the data from the interface to the RAM converting the data to Int32
    AddressNo = Convert.ToInt32(txtAddressNo.Text);
    ///invoke the delete method of the object passing in the data entered by the user
    MyAddressBook.Delete();
}
```

Press F10 once to step over this line of code to the next line...

```
protected void btnYes_Click(object sender, EventArgs e)
{
    ///this function handles the click event of the Yes button

    ///create an instance of the class clsAddressBook called MyAddressBook
    clsAddressCollection MyAddressBook = new clsAddressCollection();
    ///declare a variable to store the address number to delete
    Int32 AddressNo;
    ///copy the data from the interface to the RAM converting the data to Int32
    AddressNo = Convert.ToInt32(txtAddressNo.Text);
    ///invoke the delete method of the object passing in the data entered by the user
    MyAddressBook.Delete();
}
```

Hold the mouse over the Text property of txtAddressNo to inspect the data it contains. You should see the data value entered on the web form.

```
///copy the data from the interface to the RAM converting the data to Int32
AddressNo = Convert.ToInt32(txtAddressNo.Text);
///invoke the delete method of the object passing in the data entered by the user
```

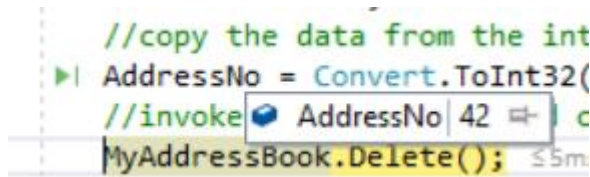
Also hold the mouse over the variable to see what data it contains...

```
///copy the data from the
> AddressNo = Convert.ToInt
///inv AddressNo 0 the
MyAddressBook.Delete();
```

It currently has a default data value of zero as we have yet to assign it with anything.

Press F10 again to trigger the assignment operation.

Now inspect the variable's data again...



The screenshot shows a debugger window with a list of variables. The variable 'AddressNo' is highlighted, showing its value as 42. The code being debugged is as follows:

```
//copy the data from the int  
AddressNo = Convert.ToInt32(  
//invoke AddressNo 42  
MyAddressBook.Delete());
```

Here we see that the assignment operator has copied the data from the Text property of the text box to the RAM at location AddressNo.

Press F10 one more time to invoke the Delete method.

Here is the question...

At what point do we copy the data from the presentation layer code to the middle layer?

And the answer is...

As the code stands we don't!

Parameters and Assignment Operators

One thing we do a lot of in programming is copying data from one place to another.

We copy data from

- Interface to the RAM
- RAM to the interface
- One layer in the architecture to another

The first tool for copying data is the assignment operator =.

The assignment operator copies data from right to left...

Destination = Source

The other tools for copying data are called parameters.

Parameters allow us to copy data from one function to another.

If we are to copy the data entered in the presentation layer we will need to create a parameter in the function in the middle layer.

Modify the function for Delete in the middle layer like so...

```

public Boolean Delete(Int32 AddressNo)
{
    ///this public function provides the functionality for the delete method

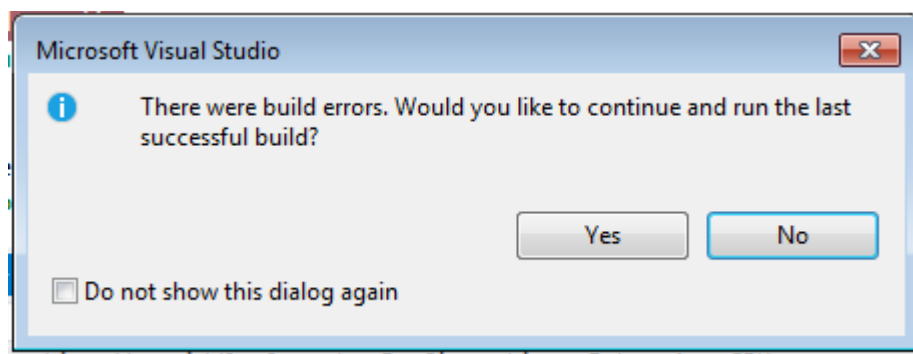
    //set the return value for the function
    return true;
}

```

What we have done here is add a parameter called AddressNo of data type Int32.

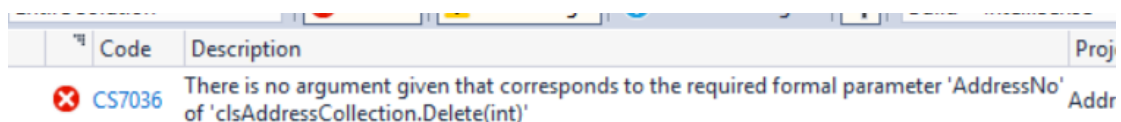
Run the program again and see what happens.

You should get an error like so...



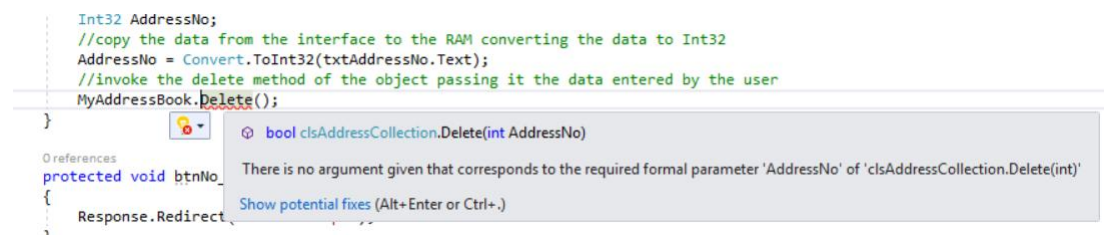
Remember, remember, remember, remember, remember, remember, that the only button you should be pressing here is the “No” button!

This will then take you to the error list...



Double click the error “No overload for method ‘Delete’ takes 0 arguments” (Which translated means that the delete method cannot have zero parameters.)

And you will see the error in the code...



(Hold the mouse over the red underlining to see the error.)

The problem is that having specified a parameter in the class definition we **MUST** include that parameter when we call the method in the presentation layer.

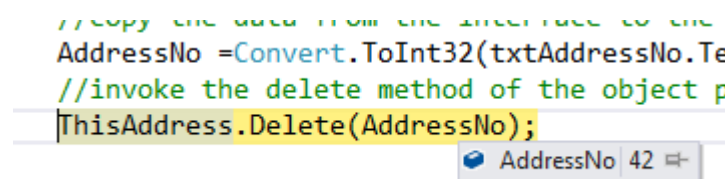
To fix the problem modify the code like so...

```
protected void btnYes_Click(object sender, EventArgs e)
{
    ///this function handles the click event of the Yes button

    //create an instance of the class clsAddressBook called MyAddressBook
    clsAddressCollection MyAddressBook = new clsAddressCollection();
    //declare a variable to store the address number to delete
    Int32 AddressNo;
    //copy the data from the interface to the RAM converting the data to Int32
    AddressNo = Convert.ToInt32(txtAddressNo.Text);
    //invoke the delete method of the object passing it the data entered by the user
    MyAddressBook.Delete(AddressNo);
}
```

Run the program again and it should work fine.

Enter a number like before and press F10 until you get to call the delete method. Inspect the value of the data to be passed in the parameter.



Don't press anything just yet as we are about to do something different!

F10 allows us to step over to the next line of code.

With the Delete method as the active line this time press F11 to step into the middle layer code...

```

public Boolean Delete(Int32 AddressNo)
{
    ///this public function provides the functionality for the delete method

    //set the return value for the function
    return true;
}

```

Inspect the data stored in the parameter AddressNo. It should contain the number entered by the user...

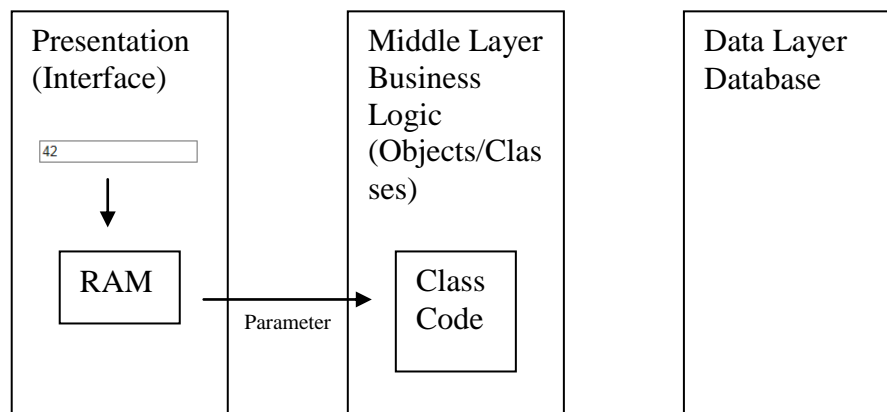
```

public Boolean Delete(Int32 AddressNo)
{
    AddressNo 42
    ///this public function provides the functionality f

```

What we have done here is the following...

1. The user entered a number into the interface
2. The data was copied from the interface to the RAM using the assignment operator
3. The data is copied from the presentation layer to the middle layer using the parameter



We have covered a lot of ground in this work.

We have...

- Seen how data may be copied from the interface to the RAM using the assignment operator
- Created a class file and made an instance of an object based on that file
- Created a public method by creating a function in the class file
- Passed data from the presentation layer to the middle layer by means of a parameter

In the next session, we shall see how to link the middle layer to the data layer completing the process of deleting a record.