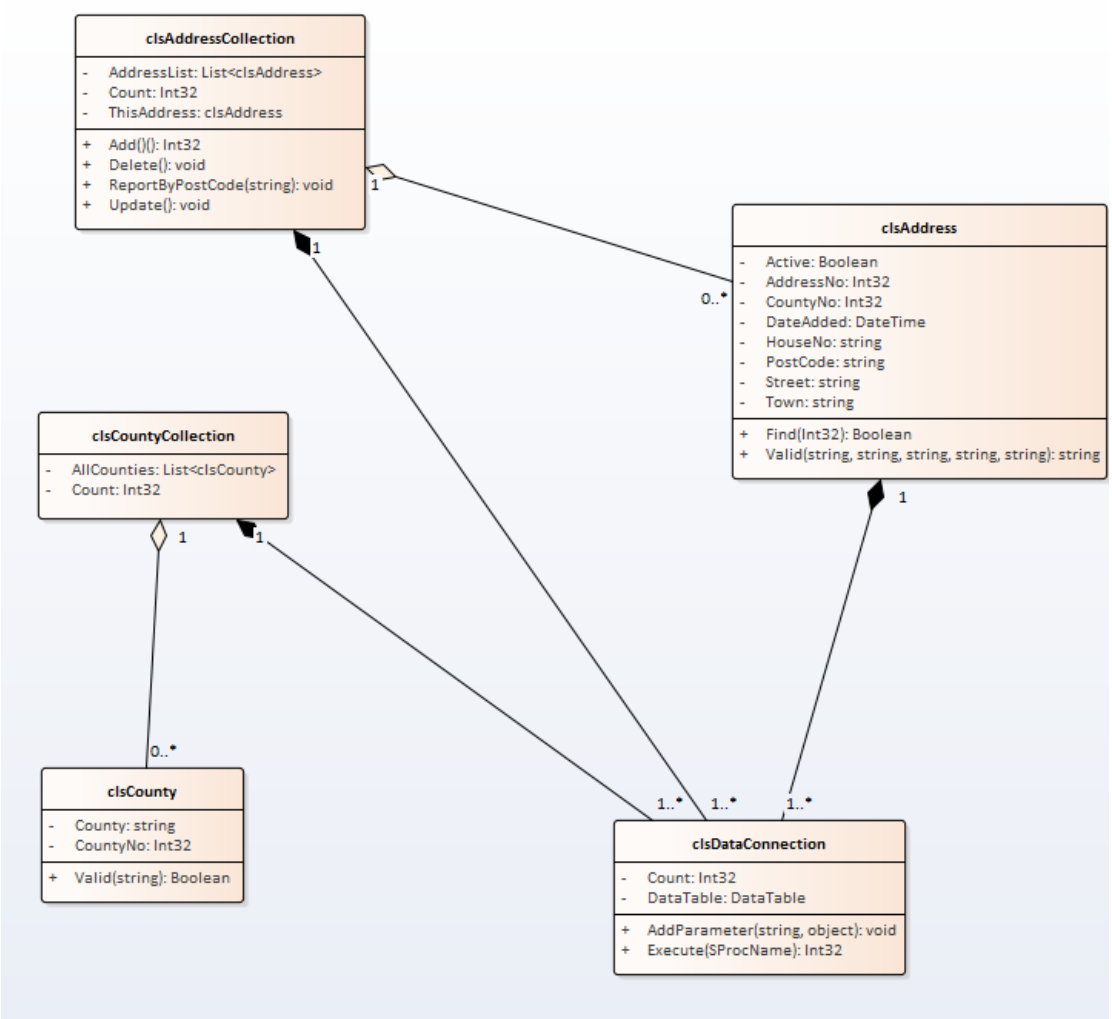
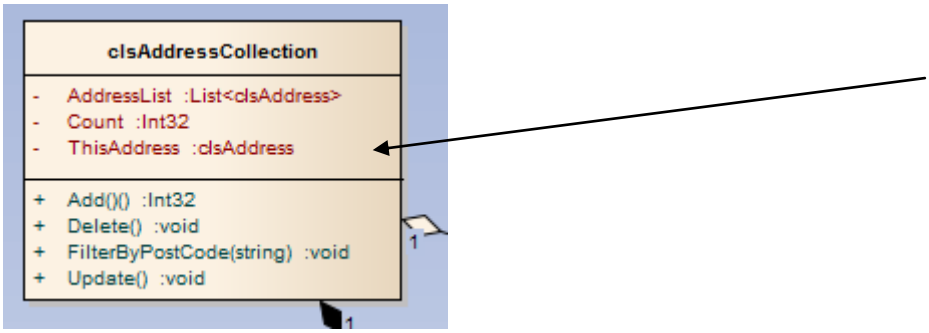
Finishing the Class Diagram – ThisAddress Property

We have come a long way in completing the system.

If we review the class diagram we started with in week two we can see that much of the work is done.



There is one last bit of work to complete in clsAddressCollection, that is the ThisAddress property.



The idea of this property is that it contains the result of any find operations and may be used to delete and update records.

It means that we may ditch the parameters we currently have for the methods Add and Update – these are not present on the class diagram.

The first step is to create the new public property in clsAddressCollection like so…

```
//public ThisAddress property
public clsAddress ThisAddress
{
    get
    {
        return mThisAddress;
    }
    set
    {
        mThisAddress = value;
    }
}
```

Notice that the object mThisAddress is underlined in red since we have not declared it yet.

To fix this declare the object at the top of the class giving it class level scope…

```
public class clsAddressCollection
{

    clsDataConnection dBConnection = new clsDataConnection();//create a connection to the database
    //member var for current address
    clsAddress mThisAddress = new clsAddress();

    public clsAddressCollection()
```

Modifying the Code for Add

The first bit of code we shall modify is the code for Add.

```
public Int32 Add(clsAddress NewAddress)
    ///this function will add a new address to the database
    ///it accepts a single parameter an object of type clsAddressPage
    ///once the record is added the function returns the primary key value of the new record
    ///
    //INSERT INTO tblAddress
    //    ( HouseNo, Street, Town, PostCode, CountyCode, DateAdded, Active )
    //SELECT
    //    @HouseNo, @Street, @Town, @PostCode, @CountyCode, @DateAdded, @Active;
{
    //connect to the database
    clsDataConnection NewDBAddress = new clsDataConnection();
    //add the parameters
    NewDBAddress.AddParameter("@HouseNo", NewAddress.HouseNo);
    NewDBAddress.AddParameter("@Street", NewAddress.Street);
    NewDBAddress.AddParameter("@Town", NewAddress.Town);
    NewDBAddress.AddParameter("@PostCode", NewAddress.PostCode);
    NewDBAddress.AddParameter("@CountyCode", NewAddress.CountyCode);
    NewDBAddress.AddParameter("@DateAdded", NewAddress.DateAdded);
    NewDBAddress.AddParameter("@Active", NewAddress.Active);
    //execute the stored procedure returning the primary key value of the new record
    return NewDBAddress.Execute("sproc_tblAddress_Insert");
}
```

Firstly we shall remove the parameter from the function definition like so…

Notice this generates red underlining…

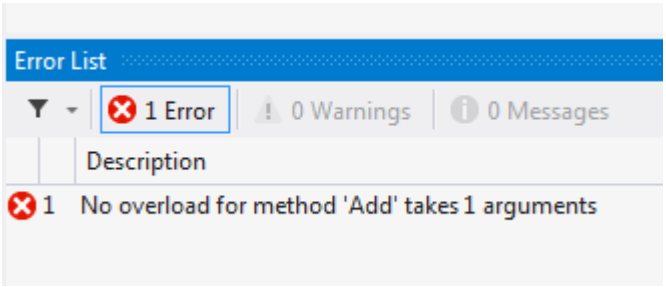
```
public Int32 Add()
    ///this function will add a new address to the database
    ///it accepts a single parameter an object of type clsAddressPage
    ///once the record is added the function returns the primary key value of the new record
    ///
    //INSERT INTO tblAddress
    //    ( HouseNo, Street, Town, PostCode, CountyCode, DateAdded, Active )
    //SELECT
    //    @HouseNo, @Street, @Town, @PostCode, @CountyCode, @DateAdded, @Active;
{
    //connect to the database
    clsDataConnection NewDBAddress = new clsDataConnection();
    //add the parameters
    NewDBAddress.AddParameter("@HouseNo", NewAddress.HouseNo);
    NewDBAddress.AddParameter("@Street", NewAddress.Street);
    NewDBAddress.AddParameter("@Town", NewAddress.Town);
    NewDBAddress.AddParameter("@PostCode", NewAddress.PostCode);
    NewDBAddress.AddParameter("@CountyCode", NewAddress.CountyCode);
    NewDBAddress.AddParameter("@DateAdded", NewAddress.DateAdded);
    NewDBAddress.AddParameter("@Active", NewAddress.Active);
    //execute the stored procedure returning the primary key value of the new record
    return NewDBAddress.Execute("sproc_tblAddress_Insert");
}
```

Rather than making use of the now non existent object NewAddress we shall instead use mThisAddress…

```
public Int32 Add()
    ///this function will add a new address to the database
    ///it accepts a single parameter an object of type clsAddressPage
    ///once the record is added the function returns the primary key value of the new record
    ///
    //INSERT INTO tblAddress
    //    ( HouseNo, Street, Town, PostCode, CountyCode, DateAdded, Active )
    //SELECT
    //    @HouseNo, @Street, @Town, @PostCode, @CountyCode, @DateAdded, @Active;
{
    //connect to the database
    clsDataConnection NewDBAddress = new clsDataConnection();
    //add the parameters
    NewDBAddress.AddParameter("@HouseNo", mThisAddress.HouseNo);
    NewDBAddress.AddParameter("@Street", mThisAddress.Street);
    NewDBAddress.AddParameter("@Town", mThisAddress.Town);
    NewDBAddress.AddParameter("@PostCode", mThisAddress.PostCode);
    NewDBAddress.AddParameter("@CountyCode", mThisAddress.CountyCode);
    NewDBAddress.AddParameter("@DateAdded", mThisAddress.DateAdded);
    NewDBAddress.AddParameter("@Active", mThisAddress.Active);
    //execute the stored procedure returning the primary key value of the new record
    return NewDBAddress.Execute("sproc_tblAddress_Insert");
}
```

Press F5 to run your program to see what happens.

You should see an error…



What this means is that having removed the parameter in the function definition we may no longer use the parameter when we call the function.

Double click the error to see where the offending line of code is in the presentation layer…

```
//if there is no error message
if (ErrorMessage == "")
{
    //create a new instance of the address book class
    clsAddressCollection AddressBook = new clsAddressCollection();
    //do something with the data - insert or update
    //
    //if the Address Number is -1
    if (AddressNo == -1)
    {
        //copy the data from the interface to the object
        ThisAddress.HouseNo = txtHouseNo.Text;
        ThisAddress.Street = txtStreet.Text;
        ThisAddress.Town = txtTown.Text;
        ThisAddress.PostCode = txtPostCode.Text;
        ThisAddress.CountyCode =Convert.ToInt32( ddlCounty.SelectedValue);
        ThisAddress.DateAdded = Convert.ToDateTime(txtDateAdded.Text);
        //add the new record
        AddressBook.Add(ThisAddress);
    }
    else
    {
```

To make this work we will need to remove the parameter on the Add function.

```
            //copy the data from the interface to the object
            ThisAddress.HouseNo = txtHouseNo.Text;
            ThisAddress.Street = txtStreet.Text;
            ThisAddress.Town = txtTown.Text;
            ThisAddress.PostCode = txtPostCode.Text;
            ThisAddress.CountyCode =Convert.ToInt32( ddlCounty.SelectedValue);
            ThisAddress.DateAdded = Convert.ToDateTime(txtDateAdded.Text);
            //add the new record
            AddressBook.Add();
        }
        else
        {
```

Then the question is how do we send the data to the middle layer if we are no longer using the parameter?

The answer is that we make use of the public ThisAddress property like so…

```
    //if the Address Number is -1
    if (AddressNo == -1)
    {
        //copy the data from the interface to the object
        AddressBook.ThisAddress.HouseNo = txtHouseNo.Text;
        AddressBook.ThisAddress.Street = txtStreet.Text;
        AddressBook.ThisAddress.Town = txtTown.Text;
        AddressBook.ThisAddress.PostCode = txtPostCode.Text;
        AddressBook.ThisAddress.CountyCode = Convert.ToInt32(ddlCounty.SelectedValue);
        AddressBook.ThisAddress.DateAdded = Convert.ToDateTime(txtDateAdded.Text);
        //add the new record
        AddressBook.Add();
    }
```

Modifying the Code for Update

We may do something similar for the Update method.

As before remove the parameter in the middle layer for the Update function and also use the private thisAddress data member…

This…

```
public void Update(clsAddress ExistingAddress)
{
    ///this function will update an existing address in the database
    ///it accepts a single parameter an object of type clsAddressPage
    ///the AddressNo property must have a valid value for this to work
    //SET   HouseNo = @HouseNo,
    //        Street = @Street,
    //        Town = @Town,
    //        PostCode = @PostCode,
    //        CountyCode = @CountyCode,
    //        DateAdded = @DateAdded,
    //        Active = @Active
    //--where the AddressNo matches the value of @AddressNo passed as the parameter
    //WHERE AddressNo=@AddressNo;    {
    //connect to the database
    clsDataConnection ExistingDBAddress = new clsDataConnection();
    //add the parameters
    ExistingDBAddress.AddParameter("@AddressNo", ExistingAddress.AddressNo);
    ExistingDBAddress.AddParameter("@HouseNo", ExistingAddress.HouseNo);
    ExistingDBAddress.AddParameter("@Street", ExistingAddress.Street);
    ExistingDBAddress.AddParameter("@Town", ExistingAddress.Town);
    ExistingDBAddress.AddParameter("@PostCode", ExistingAddress.PostCode);
    ExistingDBAddress.AddParameter("@CountyCode", ExistingAddress.CountyCode);
    ExistingDBAddress.AddParameter("@DateAdded", ExistingAddress.DateAdded);
    ExistingDBAddress.AddParameter("@Active", ExistingAddress.Active);
    //execute the query
    ExistingDBAddress.Execute("sproc_tblAddress_Update");
}
```

Becomes this…

```
public void Update()
{
    ///this function will update an existing address in the database
    ///it accepts a single parameter an object of type clsAddressPage
    ///the AddressNo property must have a valid value for this to work
    //SET   HouseNo = @HouseNo,
    //        Street = @Street,
    //        Town = @Town,
    //        PostCode = @PostCode,
    //        CountyCode = @CountyCode,
    //        DateAdded = @DateAdded,
    //        Active = @Active
    //--where the AddressNo matches the value of @AddressNo passed as the parameter
    //WHERE AddressNo=@AddressNo;    {
    //connect to the database
    clsDataConnection ExistingDBAddress = new clsDataConnection();
    //add the parameters
    ExistingDBAddress.AddParameter("@AddressNo", mThisAddress.AddressNo);
    ExistingDBAddress.AddParameter("@HouseNo", mThisAddress.HouseNo);
    ExistingDBAddress.AddParameter("@Street", mThisAddress.Street);
    ExistingDBAddress.AddParameter("@Town", mThisAddress.Town);
    ExistingDBAddress.AddParameter("@PostCode", mThisAddress.PostCode);
    ExistingDBAddress.AddParameter("@CountyCode", mThisAddress.CountyCode);
    ExistingDBAddress.AddParameter("@DateAdded", mThisAddress.DateAdded);
    ExistingDBAddress.AddParameter("@Active", mThisAddress.Active);
    //execute the query
    ExistingDBAddress.Execute("sproc_tblAddress_Update");
}
```

As before when we press F5 we will get an error which needs fixing in the presentation layer…

```
    }
    else
    {
        //this is an existing record
        //copy the data from the interface to the object
        ThisAddress.AddressNo = Convert.ToInt32(AddressNo);
        ThisAddress.HouseNo = txtHouseNo.Text;
        ThisAddress.Street = txtStreet.Text;
        ThisAddress.Town = txtTown.Text;
        ThisAddress.PostCode = txtPostCode.Text;
        ThisAddress.CountyCode = Convert.ToInt32(ddlCounty.SelectedValue);
        ThisAddress.DateAdded = Convert.ToDateTime(txtDateAdded.Text);
        //update the existing record
        AddressBook.Update(ThisAddress);
    }
    //redirect back to the main page
```

Remove the parameter and modify the code like so…

```
    }
    else
    {
        //this is an existing record
        //copy the data from the interface to the object
        AddressBook.ThisAddress.AddressNo = Convert.ToInt32(AddressNo);
        AddressBook.ThisAddress.HouseNo = txtHouseNo.Text;
        AddressBook.ThisAddress.Street = txtStreet.Text;
        AddressBook.ThisAddress.Town = txtTown.Text;
        AddressBook.ThisAddress.PostCode = txtPostCode.Text;
        AddressBook.ThisAddress.CountyCode = Convert.ToInt32(ddlCounty.SelectedValue);
        AddressBook.ThisAddress.DateAdded = Convert.ToDateTime(txtDateAdded.Text);
        //update the existing record
        AddressBook.Update();
    }
    //redirect back to the main page
    Response.Redirect("Default.aspx");
```

Modifying the Code for Delete

The last method to modify is the delete method.

In the middle layer modify the code like so…

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

    try //try to delete the record
    {
        //create an instance of the data connection class called MyDatabase
        clsDataConnection MyDatabase = new clsDataConnection();
        //add the AddressNo parameter passed to this function to the list of parameters to use in the database
        MyDatabase.AddParameter("@AddressNo", mThisAddress.AddressNo);
        //execute the stored procedure in the database
        MyDatabase.Execute("sproc_tblAddress_Delete");
        //set the return value for the function
        return true;
    }
    catch //do this only if the code above failed for some reason
    {
        //report back that there was an error
        return false;
    }
}
```

Again this will create an error when you press F5.

Fix the error with the following code…

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

    //create an instance of the class clsAddressCollection called ThisAddress
    clsAddressCollection MyAddressBook = new clsAddressCollection();
    //declare a boolean variable for Found
    Boolean Found;
    //try and find the record to delete
    Found = MyAddressBook.ThisAddress.Find(AddressNo);
    //if the record is found
    if (Found)
    {
        //invoke the delete method of the object
        MyAddressBook.Delete();
    }
    Response.Redirect("Default.aspx");
}
```

This may seem like a trivial change but it makes our code tidier in two ways.

Firstly it means that Add, Update and Delete all have a common signature and work in similar ways.

Secondly it means that we have the facility to validate if a record exists or not by making use of the clsAddress Find method.