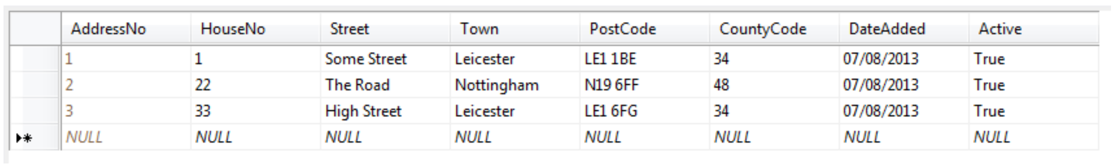
Data Tables

Now that we have a stored procedure that produces the data in the way that we want we now must think about how we are going to get that data from the data layer to the presentation layer.

When we look at a set of data such as that in a table it is quite complicated.



	AddressNo	HouseNo	Street	Town	PostCode	CountyCode	DateAdded	Active
	1	1	Some Street	Leicester	LE1 1BE	34	07/08/2013	True
	2	22	The Road	Nottingham	N19 6FF	48	07/08/2013	True
	3	33	High Street	Leicester	LE1 6FG	34	07/08/2013	True
▸*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

The best approach is to see the data as a grid which we navigate using the index and the field name.

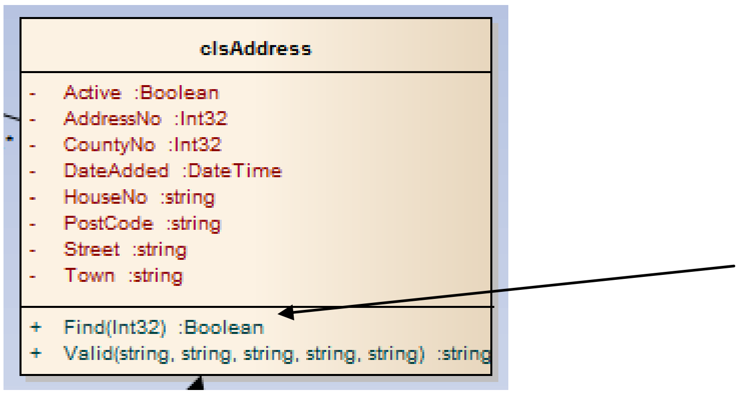
[1][“Town”] gives us Nottingham
[0][“DateAdded”] gives us 07/08/2013

One useful tool for handling this kind of data is a data table.

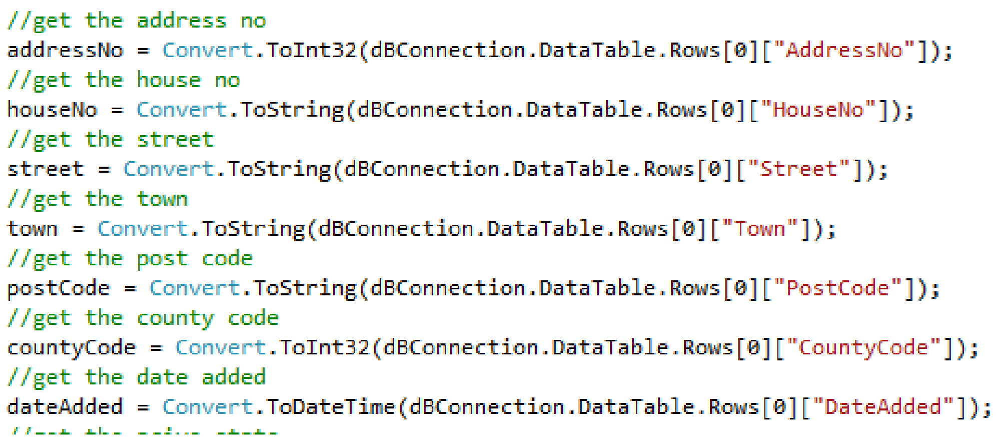
Before we do any coding, look at the following function.

```
//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
        {
            active = Convert.ToBoolean(dBConnection.DataTable.Rows[0]["Active"]);
        }
        catch
        {
            active = true;
        }
        //return success
        return true;
    }
    else
    {
        //return failure
        return false;
    }
}
```

This is a find function we will eventually create in clsAddress.



Don’t worry about the full detail of what the code is doing but notice how the data is referenced and copied from the Data Table to the variables.



```
//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"]);
```

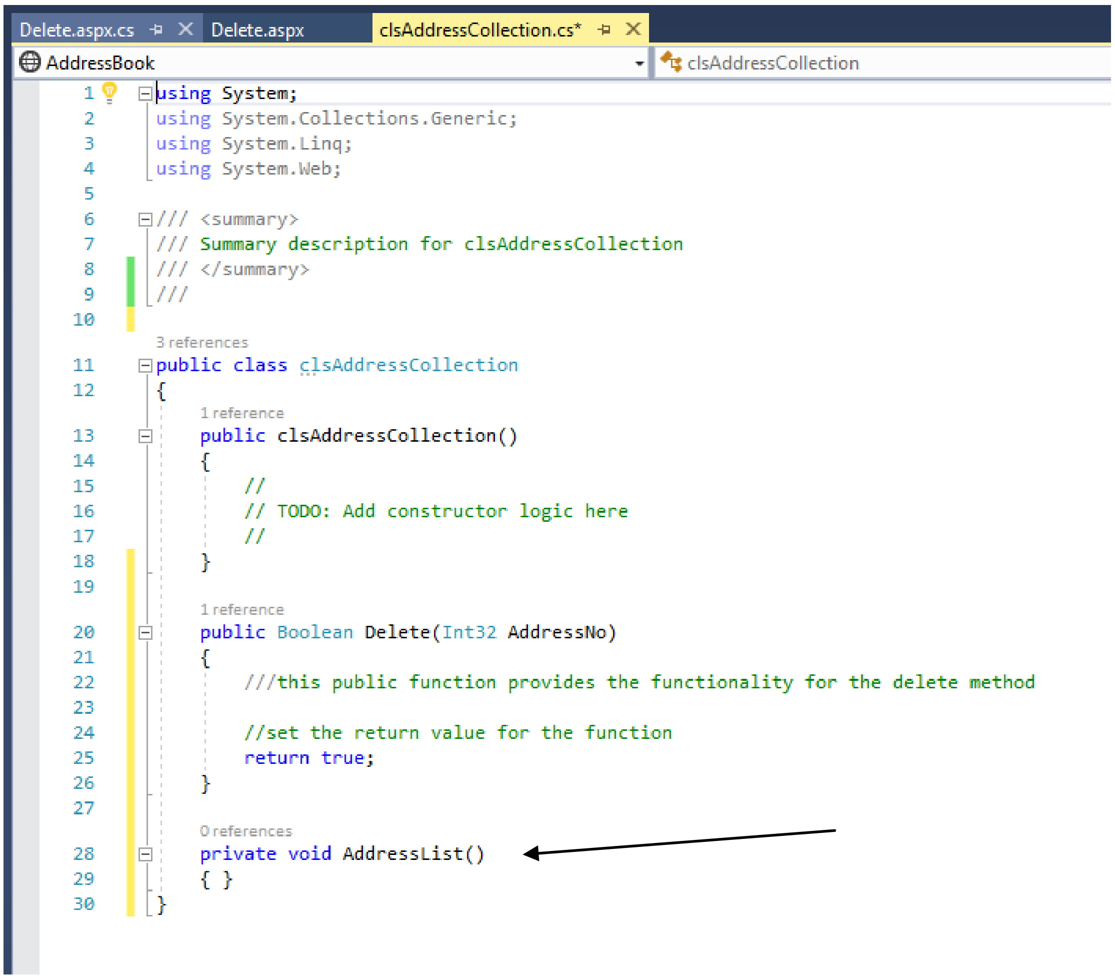
In the data table, we specify the index of the record [0] followed by the name of the field e.g. “PostCode”.

What we are going to do is create a function in the class clsAddressCollection that allows you to access the data from the data layer and then we will investigate different approaches to getting the data from the middle layer to the presentation layer.

Creating the Private AddressList Function

We are going to work on this function quite a lot over the next few weeks. The initial design of the function will not be suitable for what we want but we will extend it bit by bit.

To create the function, open the class definition for clsAddressCollection and add the following function definition...

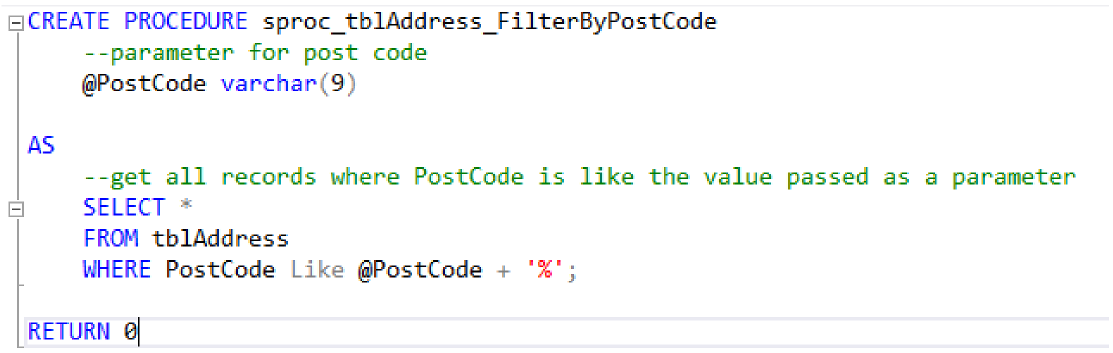


```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

/// <summary>
/// Summary description for clsAddressCollection
/// </summary>
///

3 references
public class clsAddressCollection
{
    1 reference
    public clsAddressCollection()
    {
        //
        // TODO: Add constructor logic here
        //
    }

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

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

    0 references
    private void AddressList()
    { }
}
```

We are going to start with the function set as private so that other objects can't access it. We shall later make the function public.

Note that the return data type for the function is set as "void". This means that it won't return any data when called. We shall modify the function later so that it returns some data.

We want the function to make use of the stored procedure we created last week sproc_tblAddress_FilterByPostCode



```
CREATE PROCEDURE sproc_tblAddress_FilterByPostCode
    --parameter for post code
    @PostCode varchar(9)

AS
    --get all records where PostCode is like the value passed as a parameter
    SELECT *
    FROM tblAddress
    WHERE PostCode Like @PostCode + '%';

RETURN 0
```

The function needs to...

- Open a connection to the database
- Send a parameter to the data layer specifying the post code filter
- Execute the stored procedure
- Give us access to the data table

To create a connection to the data table we need an instance of the data connection class.

```
clsDataConnection dBConnection = new clsDataConnection();
```

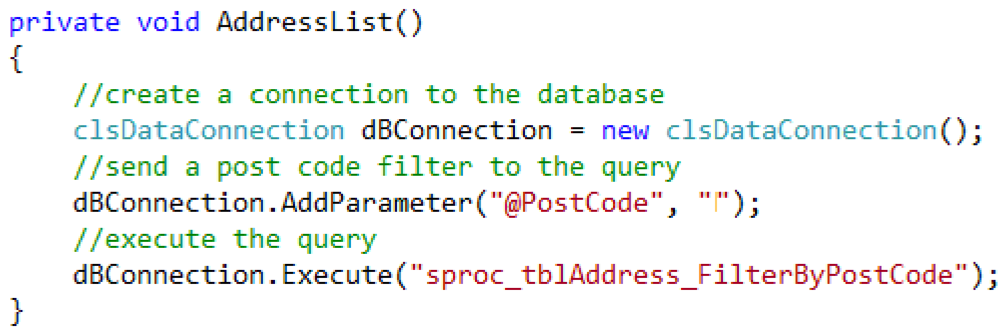
Pass the parameter data. In this case, we are hard coding a blank string "" (We will need to change this when we link to the presentation layer!)

```
dBConnection.AddParameter("@PostCode", "");
```

Execute the query.

```
dBConnection.Execute("sproc_tblAddress_FilterByPostCode");
```

The function should now look like this...

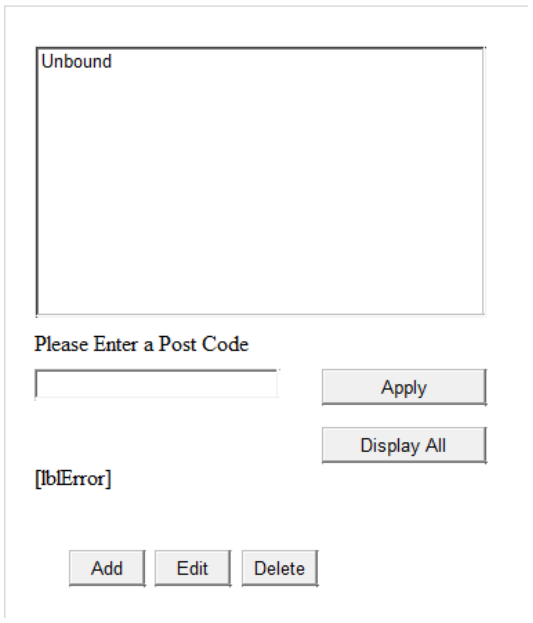


```
private void AddressList()
{
    //create a connection to the database
    clsDataConnection dBConnection = new clsDataConnection();
    //send a post code filter to the query
    dBConnection.AddParameter("@PostCode", "");
    //execute the query
    dBConnection.Execute("sproc_tblAddress_FilterByPostCode");
}
```

Testing the Function

One problem when developing code in the middle layer like this is how do we test it?

To do this open the web form Default.aspx in design view...



|Double click the Display All button to open the event handler for the button...

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

We will add some code here to allow us to test the function.

In the event handler create an instance of the class clsAddressBook.

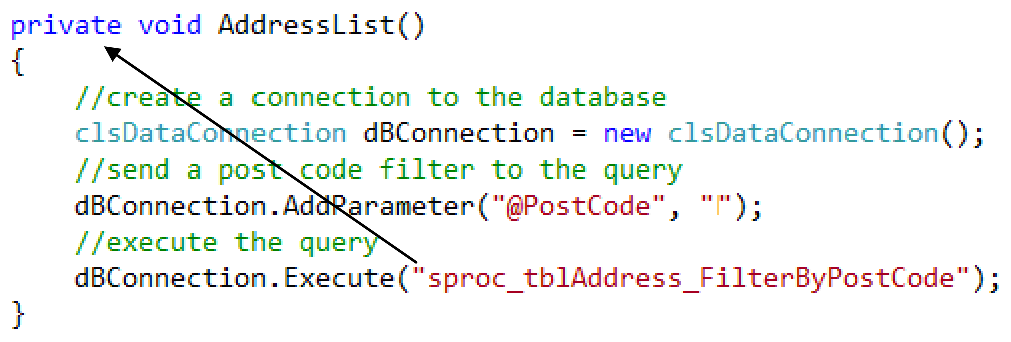
```
protected void btnDisplayAll_Click(object sender, EventArgs e)
{
    //create an instance of the address book class
    clsAddressCollection MyAddressBook = new clsAddressCollection();

}
```

The next thing we want to do is to invoke the function we have created above like so...

```
protected void btnDisplayAll_Click(object sender, EventArgs e)
{
    //create an instance of the address book class
    clsAddressCollection MyAddressBook = new clsAddressCollection();
    //access the AddressList
    MyAddressBook.AddressList();

}
```

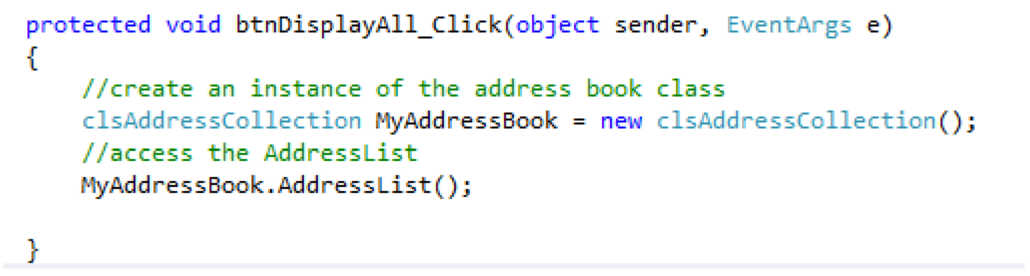
Why do we get the red underlining?

The problem is that the function in the middle layer class has been defined as private. To access the function in the presentation layer we need to make the middle layer function public.



```
private void AddressList()
{
    //create a connection to the database
    clsDataConnection dBConnection = new clsDataConnection();
    //send a post code filter to the query
    dBConnection.AddParameter("@PostCode", "");
    //execute the query
    dBConnection.Execute("sproc_tblAddress_FilterByPostCode");
}
```

Your presentation layer code should now work fine...



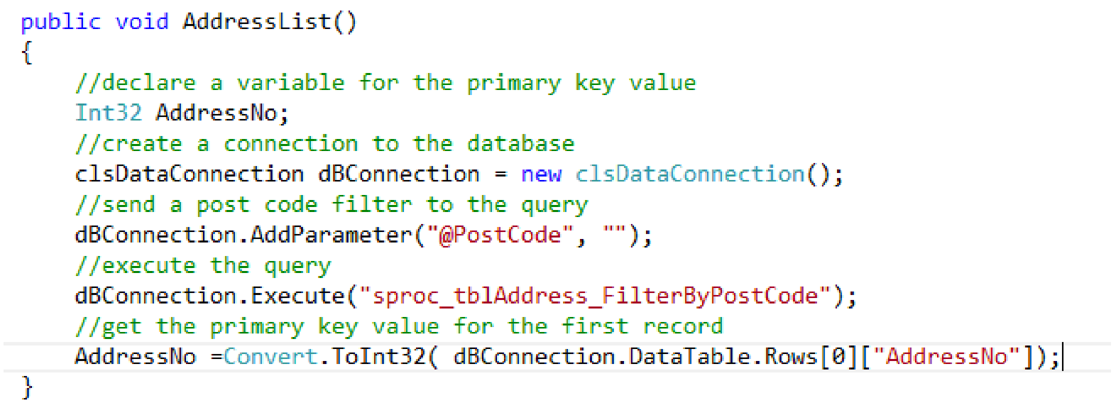
```
protected void btnDisplayAll_Click(object sender, EventArgs e)
{
    //create an instance of the address book class
    clsAddressCollection MyAddressBook = new clsAddressCollection();
    //access the AddressList
    MyAddressBook.AddressList();

}
```

The next step in testing the function is to place a breakpoint in your code and then use F10/F11 to see the code working.

It should work without any errors however the problem is that there is no data coming out of the function.

To get at the data we will need to access the data table.

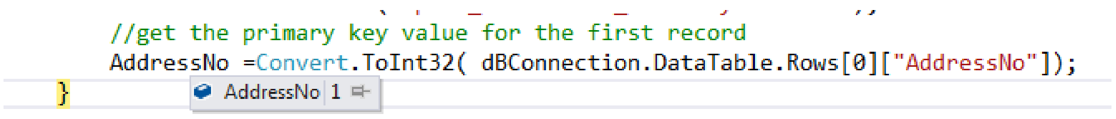
Modify the function like so...



```
public void AddressList()
{
    //declare a variable for the primary key value
    Int32 AddressNo;
    //create a connection to the database
    clsDataConnection dBConnection = new clsDataConnection();
    //send a post code filter to the query
    dBConnection.AddParameter("@PostCode", "");
    //execute the query
    dBConnection.Execute("sproc_tblAddress_FilterByPostCode");
    //get the primary key value for the first record
    AddressNo =Convert.ToInt32( dBConnection.DataTable.Rows[0]["AddressNo"]);
}
```

Using the debugger run the function to the last line of code where we assign the data from the data table to the variable AddressNo.

You should be able to see that the variable has a value from the data base...



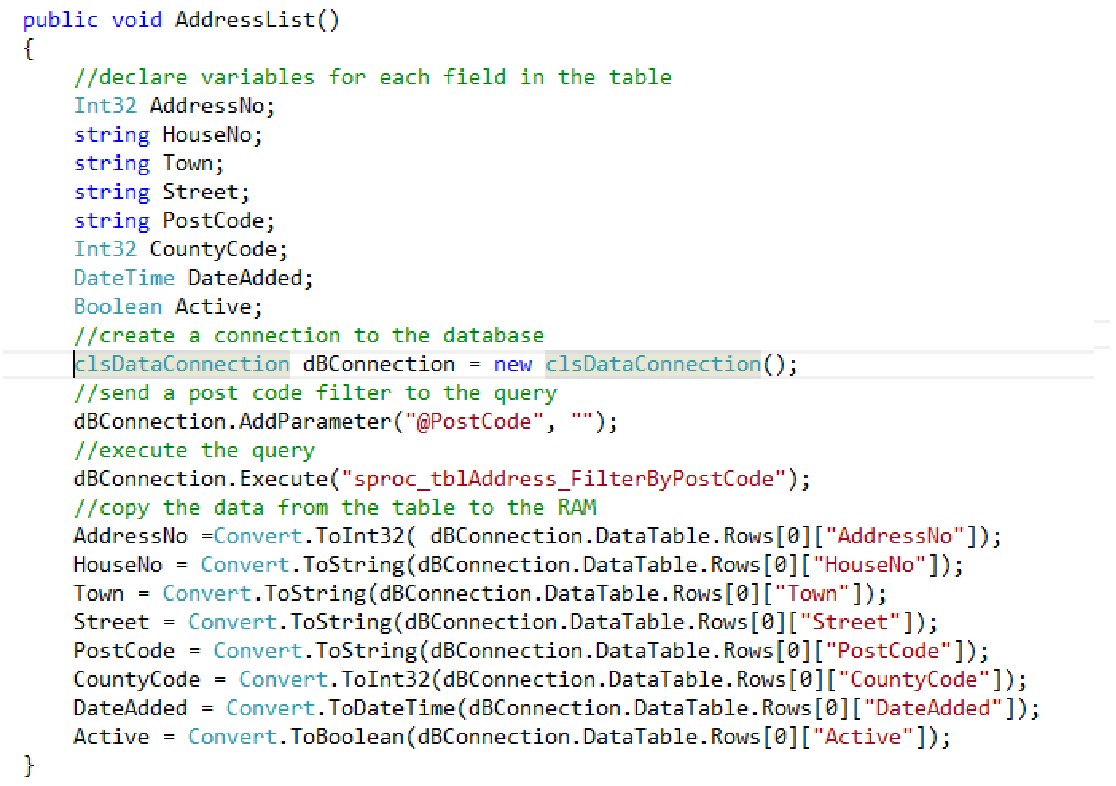
```
        //get the primary key value for the first record
        AddressNo =Convert.ToInt32( dBConnection.DataTable.Rows[0]["AddressNo"]);
    }
```

```
}
```

What we have just done is we have accessed the field “AddressNo” for the record at Index 0 (The first record) from the data table of the object dbConnection.

Modify your code so that you have variables and assignment operations copying the data from the data base to the RAM.

Your code should look something like this...



```
public void AddressList()
{
    //declare variables for each field in the table
    Int32 AddressNo;
    string HouseNo;
    string Town;
    string Street;
    string PostCode;
    Int32 CountyCode;
    DateTime DateAdded;
    Boolean Active;
    //create a connection to the database
    clsDataConnection dBConnection = new clsDataConnection();
    //send a post code filter to the query
    dBConnection.AddParameter("@PostCode", "");
    //execute the query
    dBConnection.Execute("sproc_tblAddress_FilterByPostCode");
    //copy the data from the table to the RAM
    AddressNo =Convert.ToInt32( dBConnection.DataTable.Rows[0]["AddressNo"]);
    HouseNo = Convert.ToString(dBConnection.DataTable.Rows[0]["HouseNo"]);
    Town = Convert.ToString(dBConnection.DataTable.Rows[0]["Town"]);
    Street = Convert.ToString(dBConnection.DataTable.Rows[0]["Street"]);
    PostCode = Convert.ToString(dBConnection.DataTable.Rows[0]["PostCode"]);
    CountyCode = Convert.ToInt32(dBConnection.DataTable.Rows[0]["CountyCode"]);
    DateAdded = Convert.ToDateTime(dBConnection.DataTable.Rows[0]["DateAdded"]);
    Active = Convert.ToBoolean(dBConnection.DataTable.Rows[0]["Active"]);
}
```

Run the program to see if the data is being copied to the variables correctly.

Note that there are several ways that your program may crash.

1. The filter must produce at least one record. If the filter you apply produces no records then record index zero won’t exist and the program will crash.

2. You must have some data in your table otherwise whatever filter you apply will produce zero records.

3. Beware of blank fields in your data. Blank fields create null values which can cause crashes. We shall learn how to handle them later.