List Based Objects

What we have covered so far

We have seen how we may use objects to control computer technology.

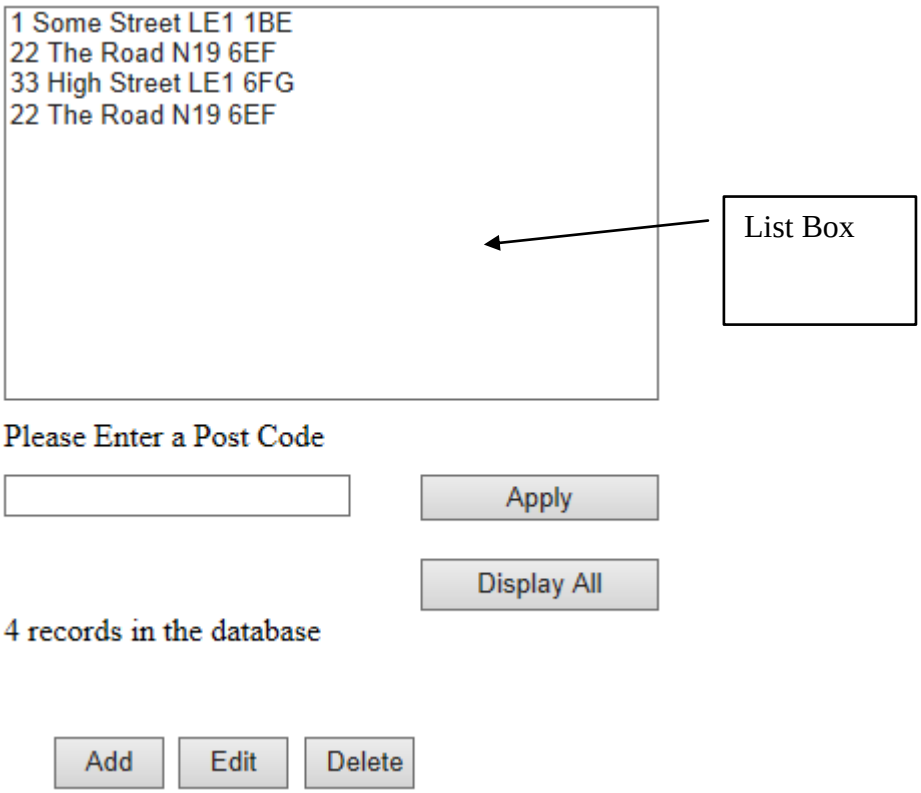
We have looked at the following:

- How a system may be split into different layers allowing us greater maintainability and reuse
- How data may be stored in a table organised into rows and columns
- How a stored procedure may be used to delete data and how a parameter may be included in the query to identify the record we want to modify (in this case delete)
- How the functionality may be placed inside a class file and by means of a public function we have created a method
- We have used the DataConnection class to trigger a query and pass data to the query via the AddParameter method
- We have used variables to store data in the RAM
- We have used control objects in the presentation layer to allow for input and output of data

Overview of the next stage of Development

We shall start by taking a look at the next stages of development and then we shall introduce various objects that will be essential to making the program work.

When the program runs, we see the following screen...

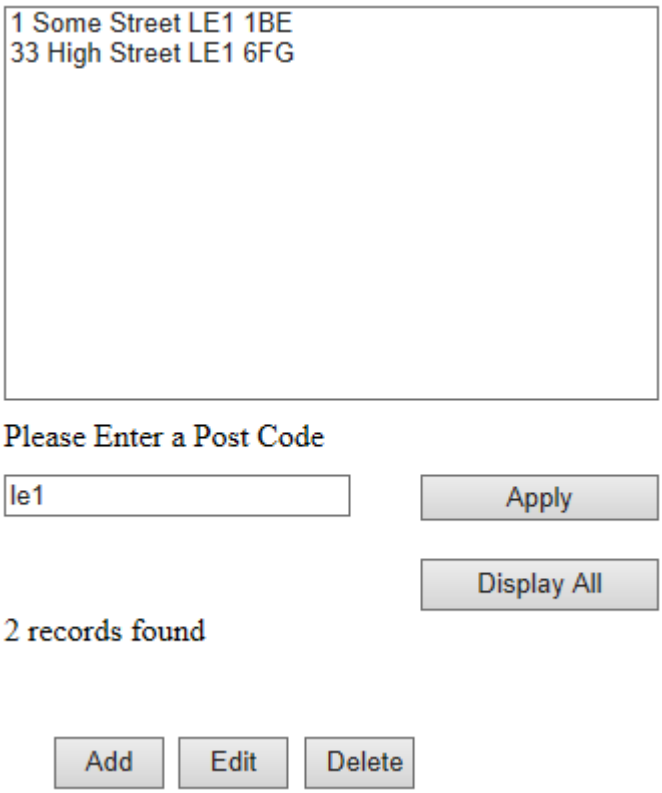


The large control in the middle of the screen is called a list box and is used to display lists of data.

Notice that the number of records is also displayed in the label control.

The other feature is the filter.

If the user types in LE1 and presses Apply Filter the following data is displayed...



This provides the user with a facility for searching through the records in the table.

Array Lists, Data Tables

In this lecture, we are going to look at two objects that allow us to control lists of data.

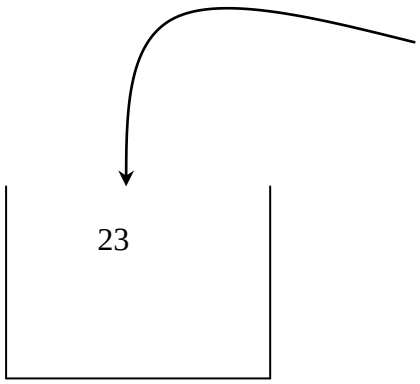
The first object we shall look at is called an array list. We will see how items may be added, counted and changed using this object.

Once we have looked at how an array list functions we will look at a related object called a data table. The data table has similar features to an array list however many of the features we see in the array list are automated so we don't need to worry about them.

Storing data using Variables

In our programs so far, we have used variables to store our data in RAM.

Variables are fine if we want to store a single item of data.

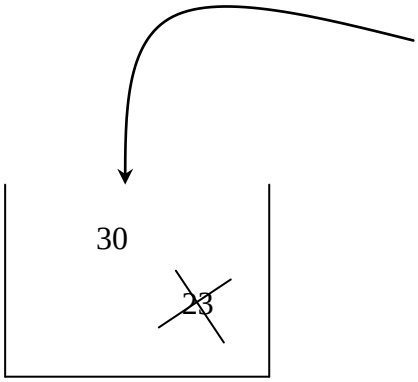


Int32 Age;
(Allocates a section of RAM)

Age = 23;

(Places the value 23 into the memory location called age)

The problem with variables though is that they may only store a single item of data at a time. What happens if we want to store another age in the same box?



Age = 30;

(Copies the value 30 into the variable age)

The original value of 23 is overwritten by the new value of 30.

The problem with much real-life data is that it tends to come not as single items of data, it tends to come in lists.

For example, if we wanted to store the names of people attending a meeting how would we do it?

Fred
Wilma
Barney
Betty

To store data that is a list or a table we would use an array list.

The following line of code declares an array list to store meeting attendees…

```
List<string> Attendees = new List<string>();
```

The keyword List is used to specify a list. We also place in triangular braces the class or data type to be used in the list.

To add our four attendees to the meeting we could use the following lines of code…

```
Attendees.Add("Fred");
Attendees.Add("Wilma");
Attendees.Add("Barney");
Attendees.Add("Betty");
```

Counting the Number of Items

Once we have data in the list it would be helpful to have a mechanism for counting the number of items.

To find out the number of items in an array list we access the Count property of the object...

```
List<string> Attendees = new List<string>();
Attendees.Add("Fred");
Attendees.Add("Wilma");
Attendees.Add("Barney");
Attendees.Add("Betty");
Int32 ItemCount;
ItemCount = Attendees.Count;
```

In this case ItemCount would have the value of 4.

1. Fred
2. Wilma
3. Barney
4. Betty

Index Numbers

A very important concept in programming is the idea of an index.

An index is used to reference a specific item in the array list.

The system is like how we number our homes...

If we want flat 54 then it is clear above which bell we need to ring.

Zero Bound Lists

The problem comes however in that most indexing systems for computers don't start at one, they start at zero!

If we look again at the following data and add the indexes we see the following...

Value	Index
Fred	0
Wilma	1
Barney	2
Betty	3

Even though Fred is item number 1 on the list the index is in fact zero!

This can be a bit of a pain when we start to manipulate the data but we won't worry about that now just remember that the index starts at zero for now!

Removing an Item

To remove an item from an array list we need to know that index of the item that we want to remove.

If this is our data

Value	Index
Fred	0
Wilma	1
Barney	2
Betty	3

To remove a specific item, we need to use the RemoveAt method of the Array List specifying the index of the item to be removed, so…

```
Attendees.RemoveAt(2);
```

This would remove the third item in the list.

Changing an Entry in the List

What do you think the following code would do?

```
List<string> Attendees = new List<string>();
Attendees.Add("Fred");
Attendees.Add("Wilma");
Attendees.Add("Barney");
Attendees.Add("Betty");
Attendees[2] = "Bamm Bamm";
```

The following line of code would overwrite the value “Barney” (entry number 2) with “Bamm Bamm”.

```
Attendees[2] = "Bamm Bamm";
```

Data Tables

Data Tables are related to Array Lists in that they allow us to manipulate data in a list. Data Tables however are specifically designed to allow us to manipulate data that has been drawn from a database.

Imagine that we have the following data in the table called tblAddress...

	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
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

In the data layer, we create a query called sproc_tblAddress_SelectAll which contains the following SQL...

SELECT *
FROM tblAddress

This query will select all records in the table tblAddress.

Imagine next that we create some code that will allow us to execute the query using the Data Connection class...

```
//initialise the DBConnection
clsDataConnection dBConnection = new clsDataConnection();
//execute the stored procedure to delete the address
dBConnection.Execute("sproc_tblAddress_SelectAll");
```

So how do we access the data?

The data from the table is now stored within the object called dbConnection. We may get to it via the DataTable.

If this is our data...

	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
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

What do you suppose the following lines of code will give us?

```
MyAddresses.DataTable.Rows[1]["HouseNo"]

MyAddresses.DataTable.Rows[3]["Street"]

MyAddresses.DataTable.Rows[2]["AddressNo"]
```

The data returned is...

```
HouseNumber = 22

Street = “The Road”

AddressNo = 4
```

This provides us with an easy mechanism for accessing data at any specific row or column in the query results.

Converting Data Tables to Array Lists

One thing that you need to keep in mind with how data is handled in a system is that different tools are needed for different layers of the architecture.

When tabular data is handled in the data layer it is typically stored in a table. Stored procedures process this data however the output from a query is normally indistinguishable from a table.

	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
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

When data is transferred from the data layer to the middle layer it often comes in the form of a data table.

We access the data in the data table using indexes and field names like so...

```
//get the house no from the query results
NewAddress.HouseNo = Convert.ToString(dBConnection.DataTable.Rows[Index]["HouseNo"]);
//get the street from the query results
NewAddress.Street = Convert.ToString(dBConnection.DataTable.Rows[Index]["Street"]);
//get the post code from the query results
NewAddress.PostCode = Convert.ToString(dBConnection.DataTable.Rows[Index]["PostCode"]);
//get the address no from the query results
NewAddress.AddressNo = Convert.ToInt32(dBConnection.DataTable.Rows[Index]["AddressNo"]);
```

When we transfer the data from the middle layer to the presentation layer we typically convert the data table into an array list.

Why?

Encapsulation and Data Hiding

One of the reasons for splitting our systems into three layers is that we do not want the presentation layer to have any knowledge of the structure of the database.

The idea is that if we rename one of the fields in the database the interface won't notice.

Think of it this way. If we have a field in the database called PostCode and this field is used on twenty different pages in our site what happens if we rename PostCode to PCode? If the pages "know" the name of the field we need to modify all twenty pages. If only the middle layer knows the field name and it is used in the presentation layer via an alias we only need to make one change in the middle layer.

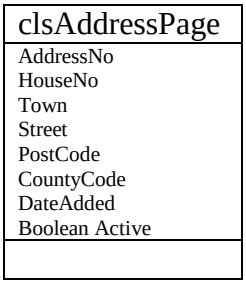
This process of hiding the true nature of the data and functionality is called encapsulation.

To create a collection class, we are going to have to have a class acting as parent and a class acting as child.

In the address book example, the parent is the address book and the child is the address page.

The first thing we need to do is create a class allowing us to model the address page.

The address page class (clsAddressPage) has the following properties...



Spend a bit of time examining the following code and think about what is going on.

```
///this function exposes the DataTable via the public collection AllAddresses
public List<clsAddressPage> AllAddresses
{
    get
    {
        List<clsAddressPage> allAddresses = new List<clsAddressPage>();
        Int32 Index=0;
        while (Index < dBConnection.Count)
        {
            clsAddressPage NewAddress = new clsAddressPage();
            //get the house no from the query results
            NewAddress.HouseNo = Convert.ToString(dBConnection.DataTable.Rows[Index]["HouseNo"]);
            //get the street from the query results
            NewAddress.Street = Convert.ToString(dBConnection.DataTable.Rows[Index]["Street"]);
            //get the post code from the query results
            NewAddress.PostCode = Convert.ToString(dBConnection.DataTable.Rows[Index]["PostCode"]);
            //get the address no from the query results
            NewAddress.AddressNo = Convert.ToInt32(dBConnection.DataTable.Rows[Index]["AddressNo"]);
            //increment the index
            Index++;
            //add the address to the list
            allAddresses.Add(NewAddress);
        }
        //return the list of addresses
        return allAddresses;
    }
}
```

We shall come back to this code again but note the following.

The line of code...

```
List<clsAddressPage> allAddresses = new List<clsAddressPage>();
```

Creates a new list based on the class clsAddressPage – a list of pages

The line of code...

```
clsAddressPage NewAddress = new clsAddressPage();
```

Creates a new single address page.

The following lines of code...

```
//get the house no from the query results
NewAddress.HouseNo = Convert.ToString(dBConnection.DataTable.Rows[Index]["HouseNo"]);
//get the street from the query results
NewAddress.Street = Convert.ToString(dBConnection.DataTable.Rows[Index]["Street"]);
//get the post code from the query results
NewAddress.PostCode = Convert.ToString(dBConnection.DataTable.Rows[Index]["PostCode"]);
//get the address no from the query results
NewAddress.AddressNo = Convert.ToInt32(dBConnection.DataTable.Rows[Index]["AddressNo"]);
```

Copies the data for a single record from the data table to the single page.

This line of code...

```
allAddresses.Add(NewAddress);
```

Adds the single page to the list allAddresses

We shall see this procedure followed in a few examples over the coming weeks.

- Create a list of items
- Create a single item
- Copy the data for one record to the single item
- Add the single item to the list
- Repeat until all records are processed