Learning VB.Net

Tutorial 17 – Classes

Hello everyone… welcome to vb.net tutorials. These are going to be very basic tutorials about using the language to create simple applications, hope you enjoy it. If you have any notes about it, please send them to **notes@mka-soft.com** I will be happy to answer them. Finally if you find these tutorials are useful, it would be nice from you to send a small donation via PayPal to **donation@mka-soft.com**.

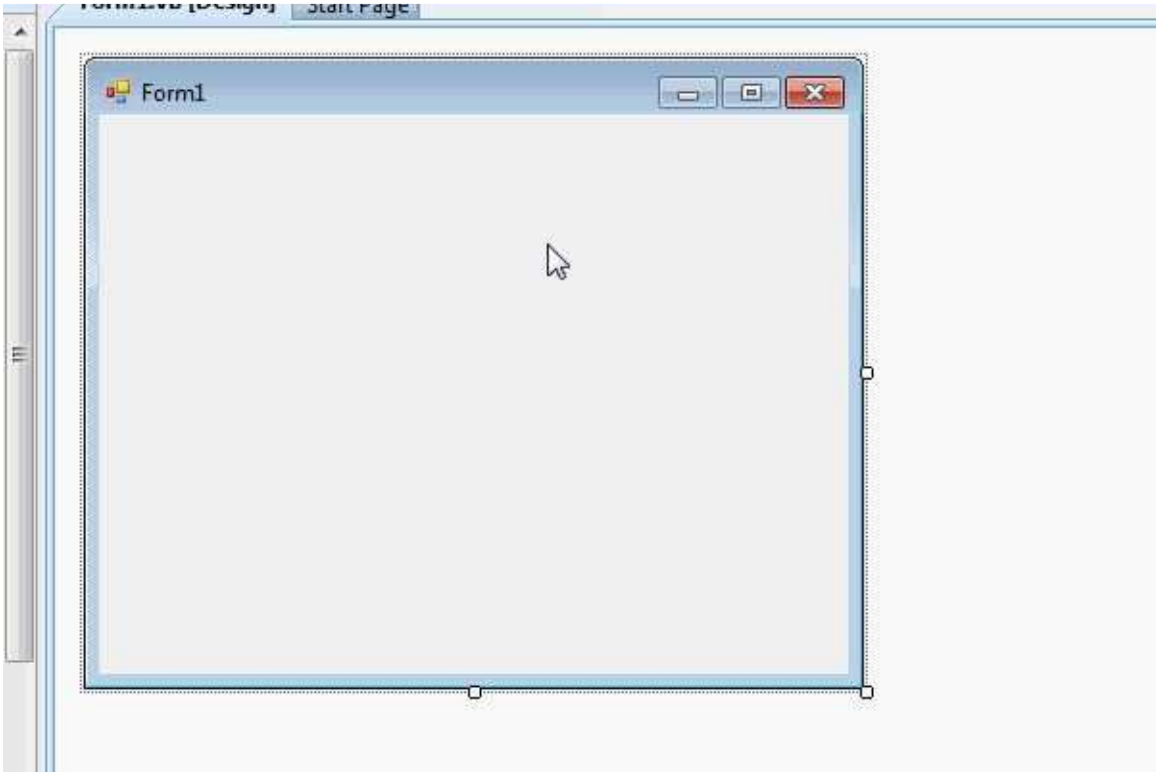
Tutorial posted on 2010-May-03.

Classes

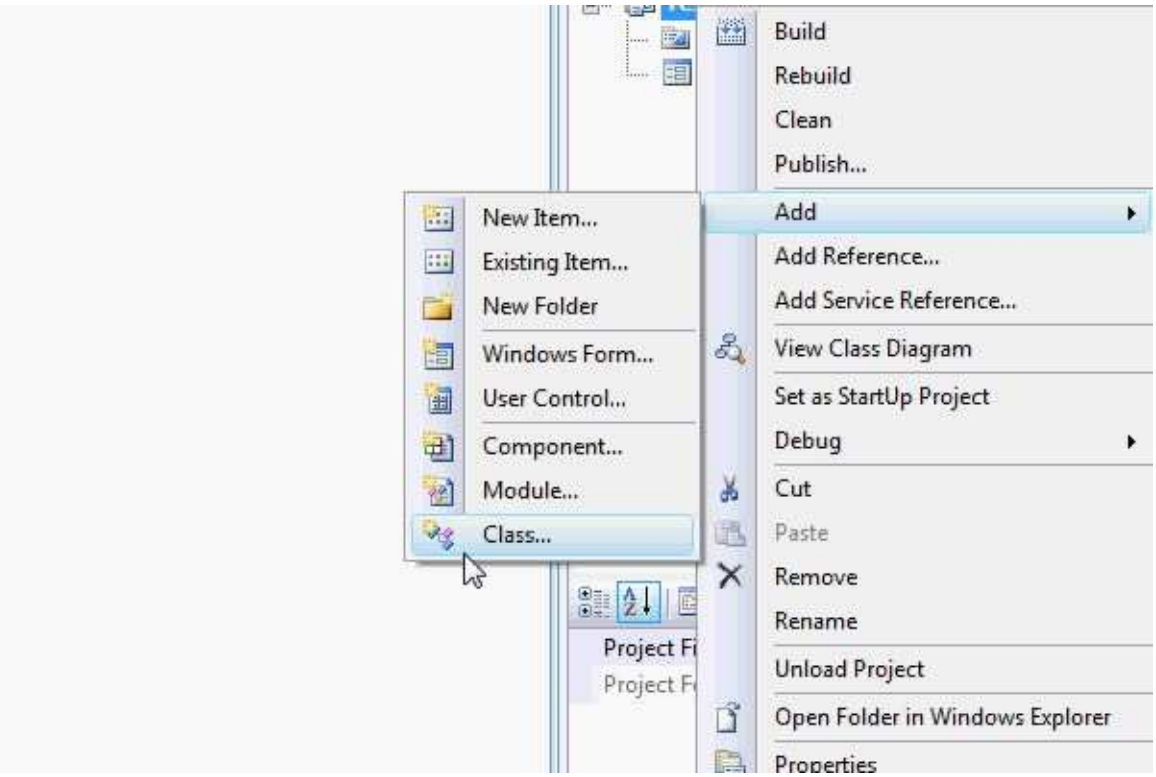
Previously we saw how to work with structures, and we examined how we can combine related information into one logical unit. Classes are very similar to structures, except that they allow you to combine the functions and subroutines that work on your information as well. It also has many other useful features that allows you to create and use frameworks to reuse the code.

To start understand classes we are going to develop a simple address book application. The application will allow you to store user information (name, address and telephone number).

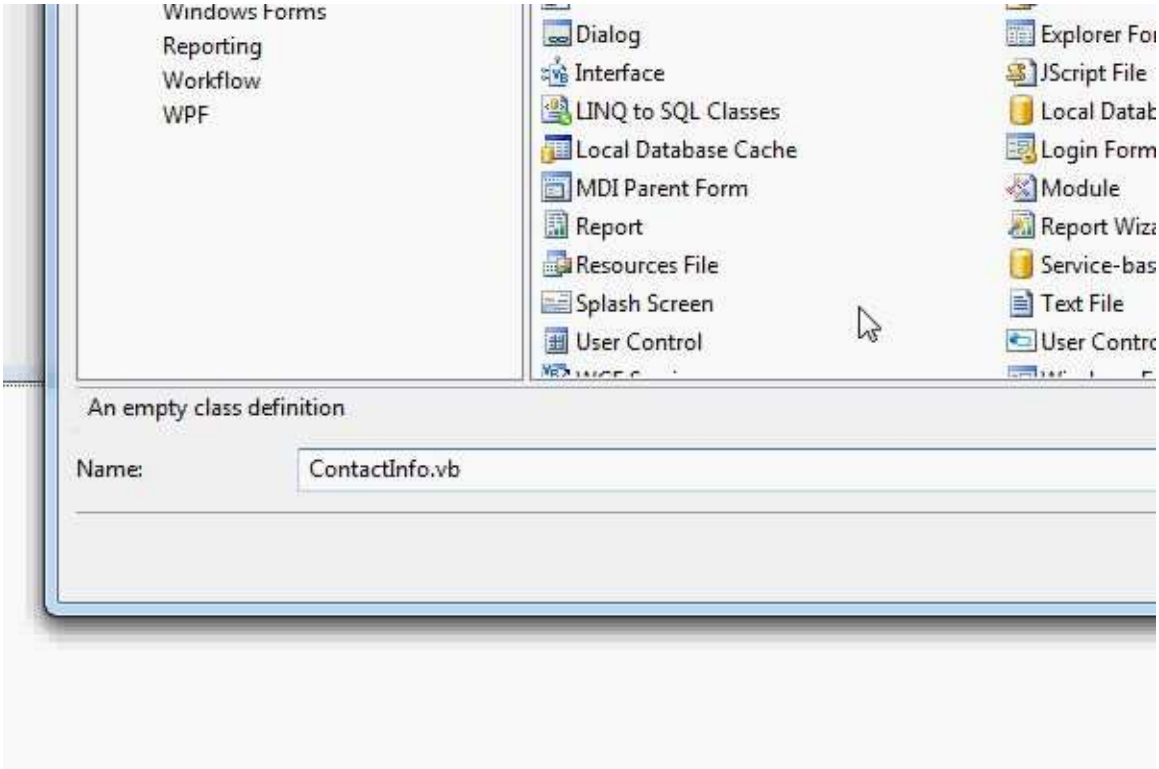
Start visual studio, and create a new project.



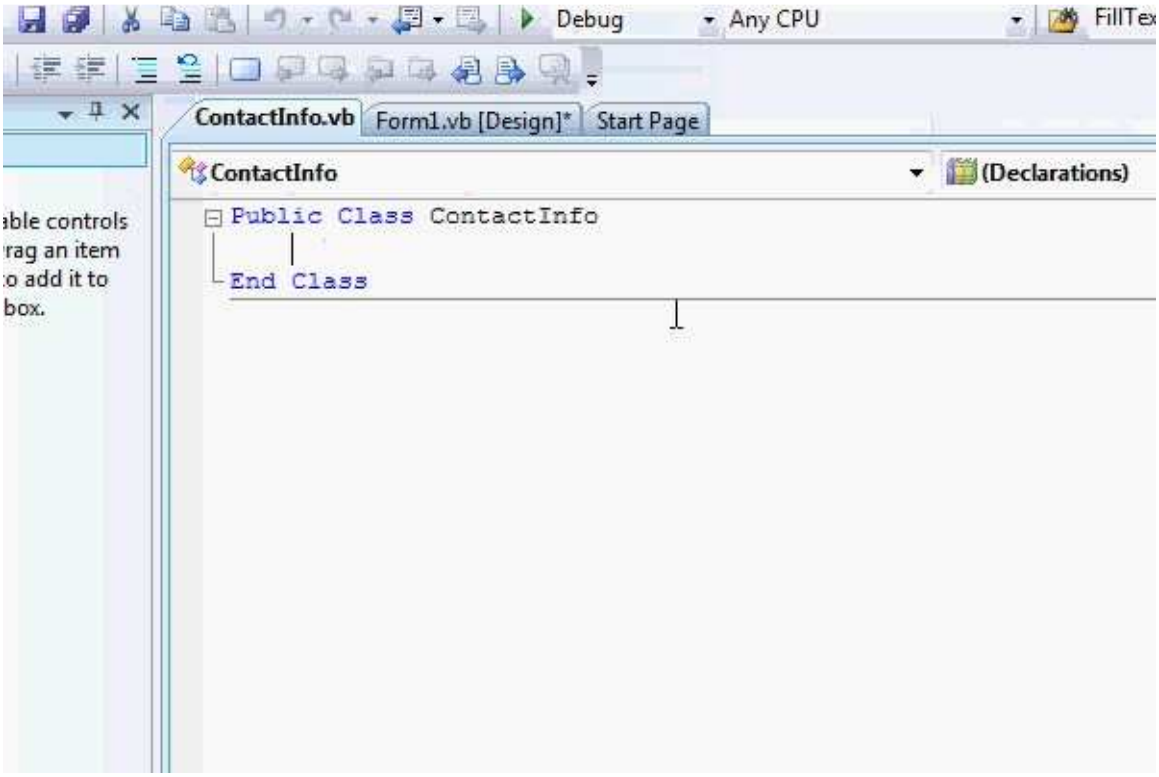
First thing we are going to do is to create a simple class that will describe the information for each contact. Usually each class is placed in a separate file. The process is similar to adding module, or adding another form to your project. Right click your project->Add->Class



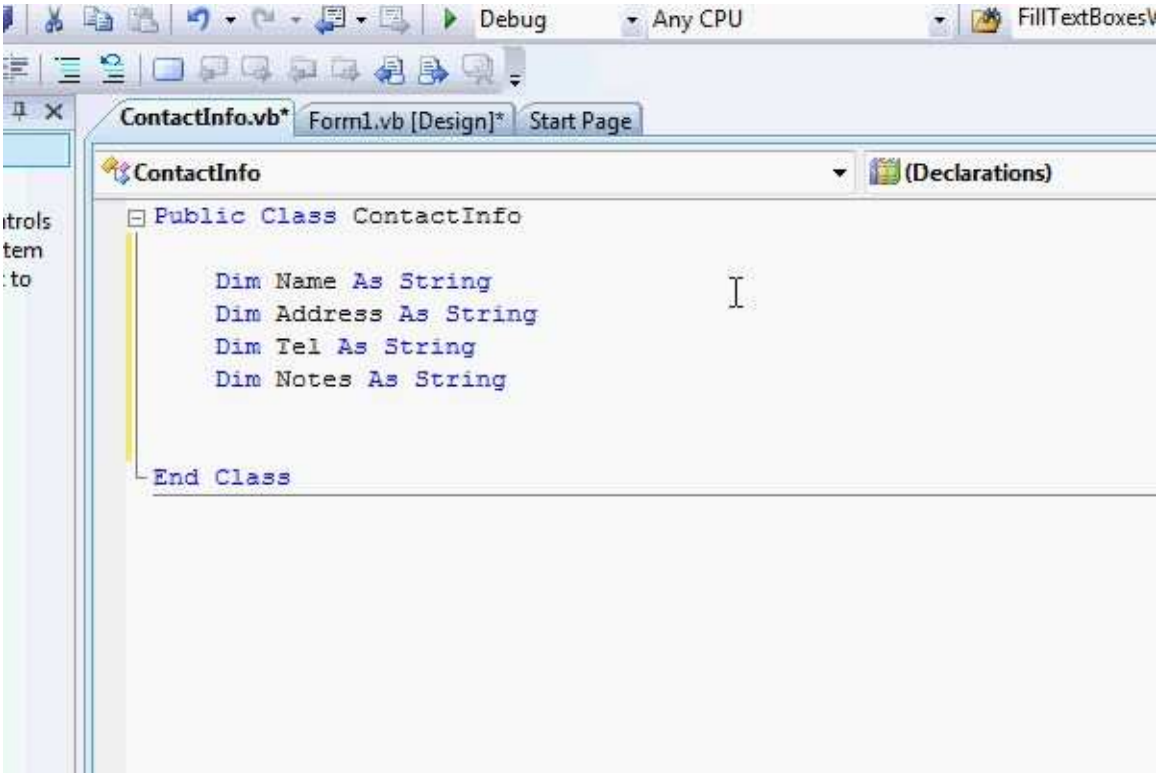
Next provide the name of the class (ContactInfo)



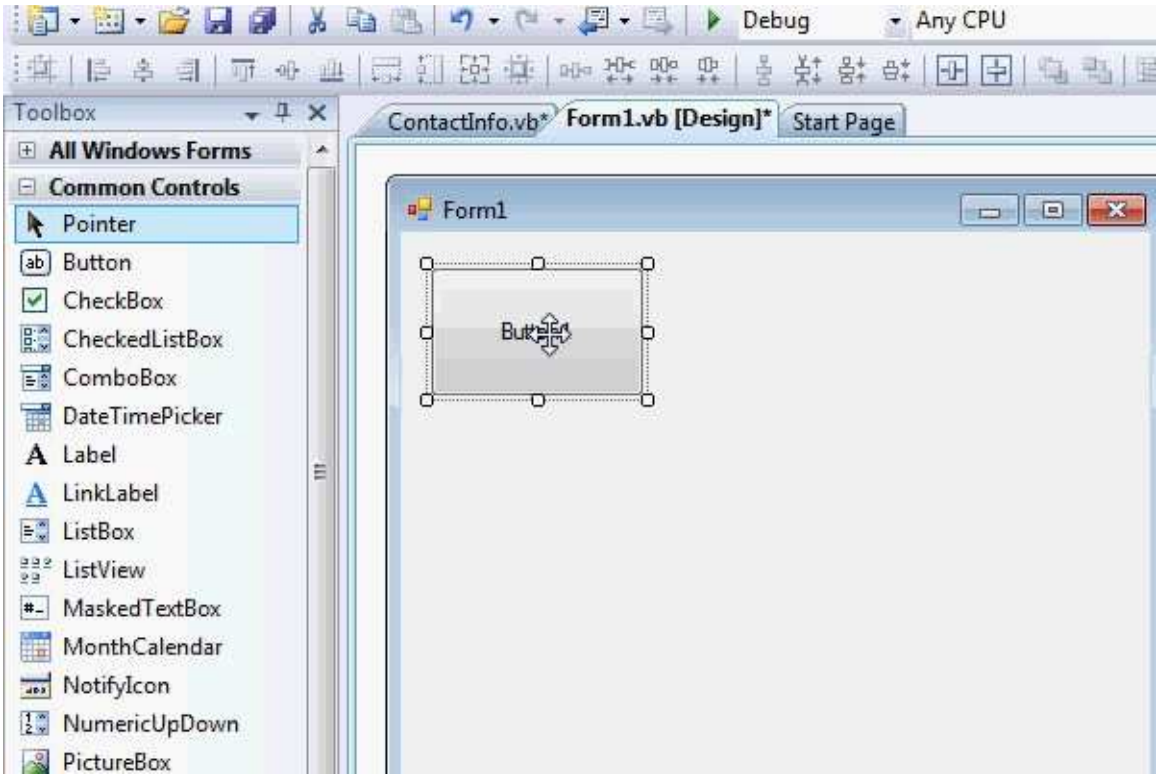
Next you will see the following:



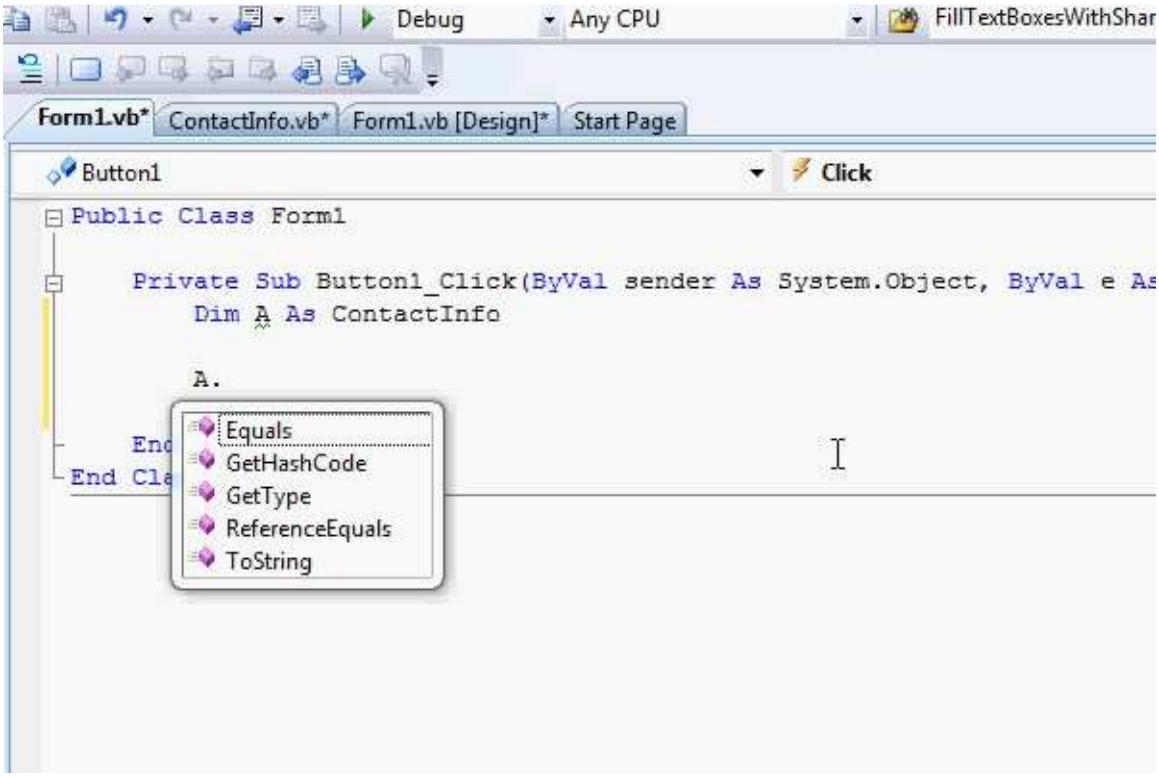
Now you should start writing the code for your class. The contact for each person should include person Name, Address and Telephone, therefore you define three variables as shown below:



To test this go to the form and place a button

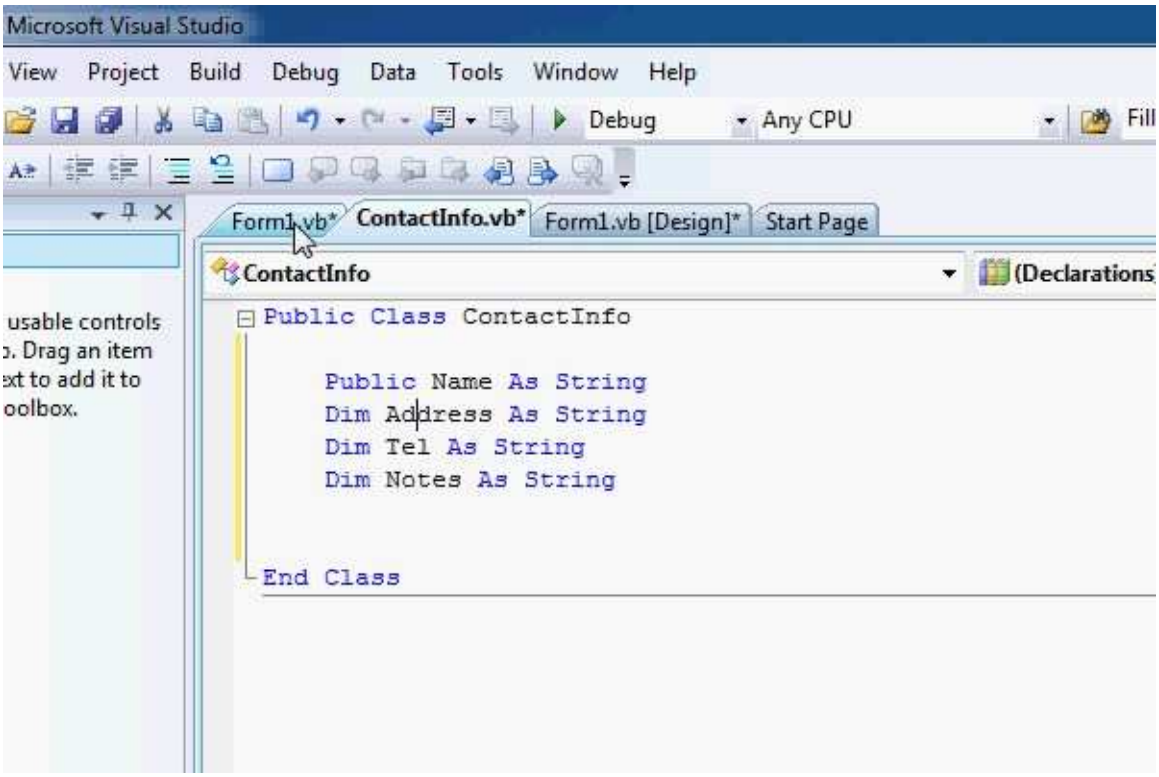


Next try to add the following to the code of the button

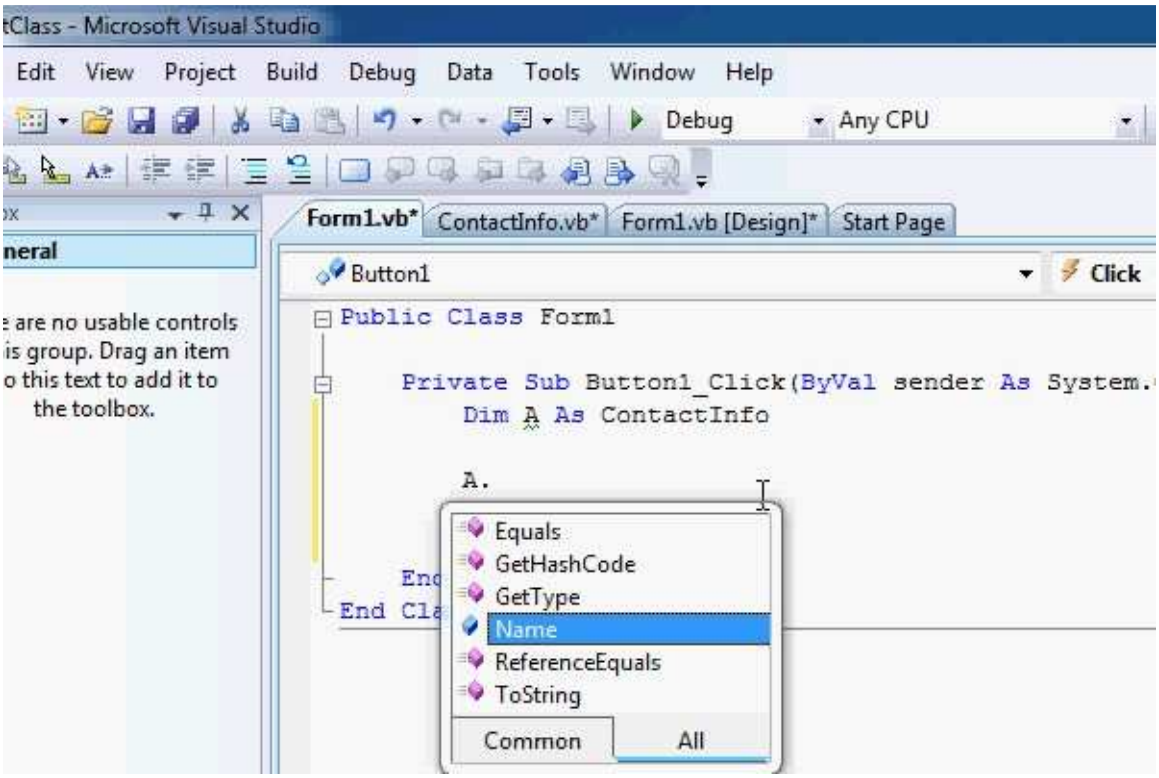


First you define a variable of type ContactInfo in a way similar to what you used to with structures, however, when you want to access the variables of the class you will find that the editor does not list them. Actually even if you write them manually you won't be able to run the program. This is because the variables within the class are protected from access outside the class code. This helps hiding complex code and the variables you don't want to be accessed by mistake.

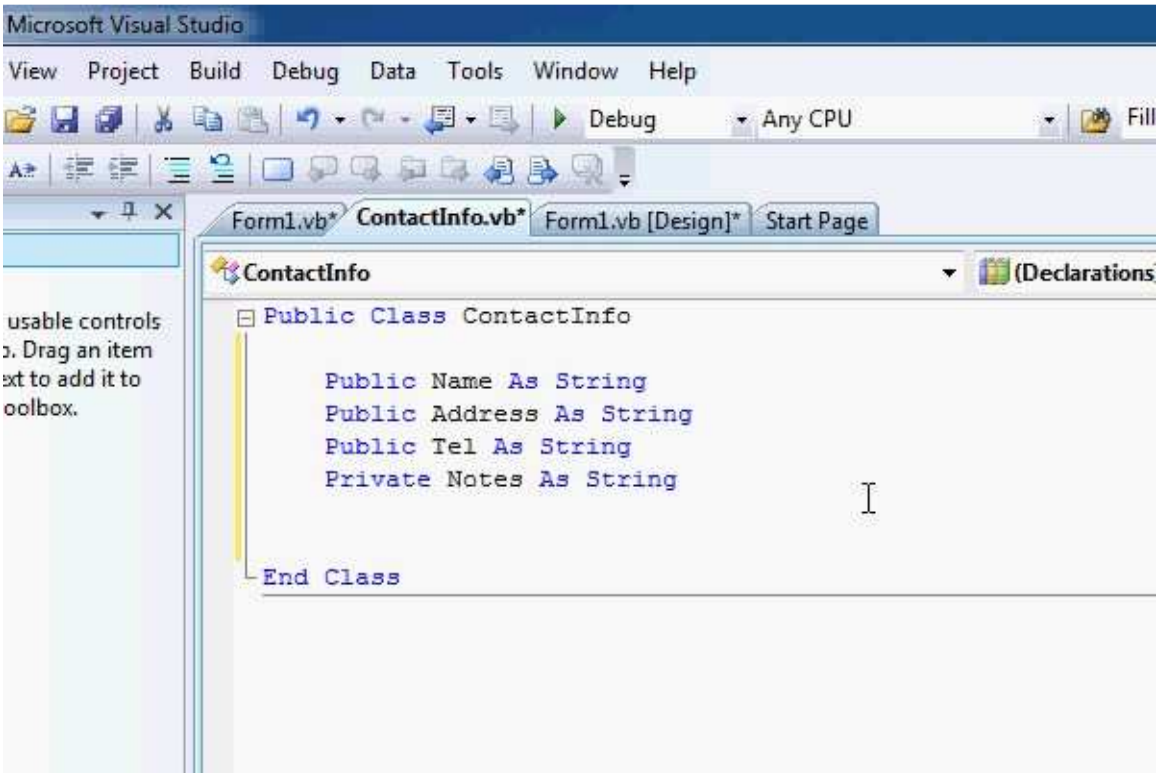
Now to make any variable accessible just change the Dim keyword in front of the variable to Public. This will grant this variable public access from any code within the project.



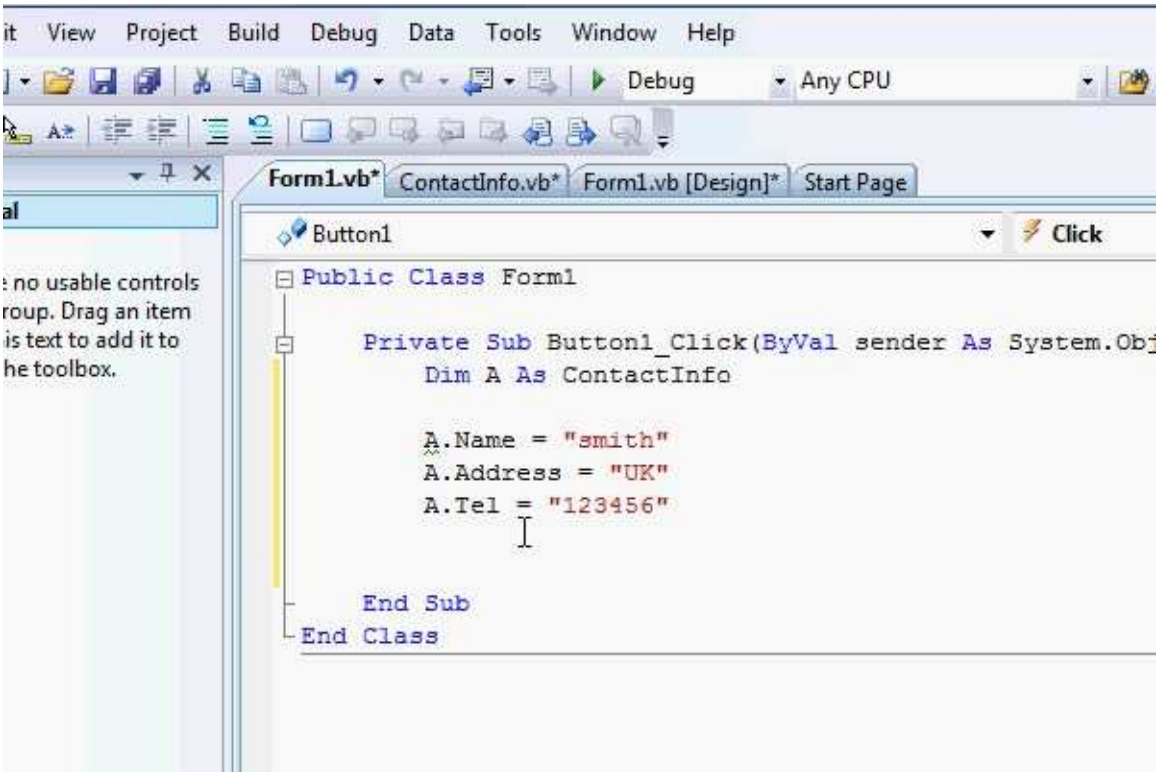
Now if you try to access the name property, you will see that the editor can detect that, and the property is listed when you press the (.) after the variable name.



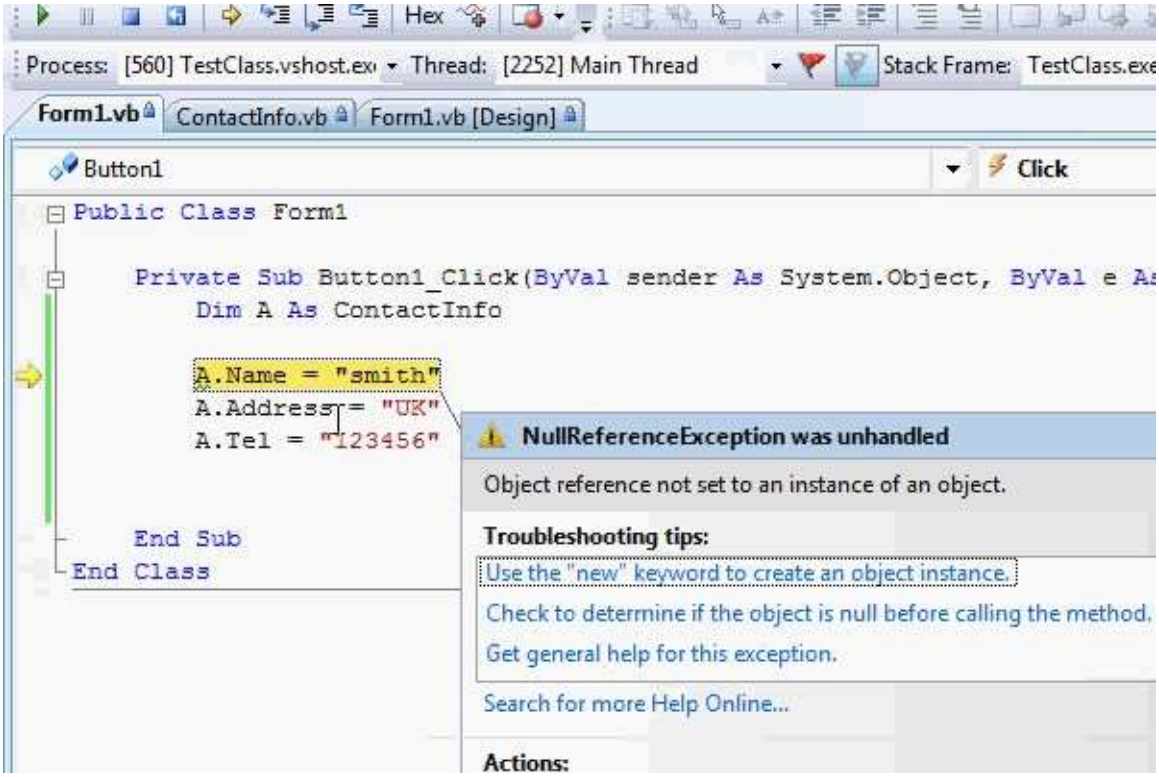
Make the Address and Tel variables within the class public similar to the way below:



Next add the following code to the event handler of the Button1



Now your code is completely correct from syntax point of view, however it will not run correctly. If you run the code and then hit the button, then you get the following error:



This brings us to the second difference of class from structure. The variable A in the example is just a pointer to where the actual data is stored in memory, and there is not memory resources allocated to store the name, address and tel values for A. This is why you are getting the error.

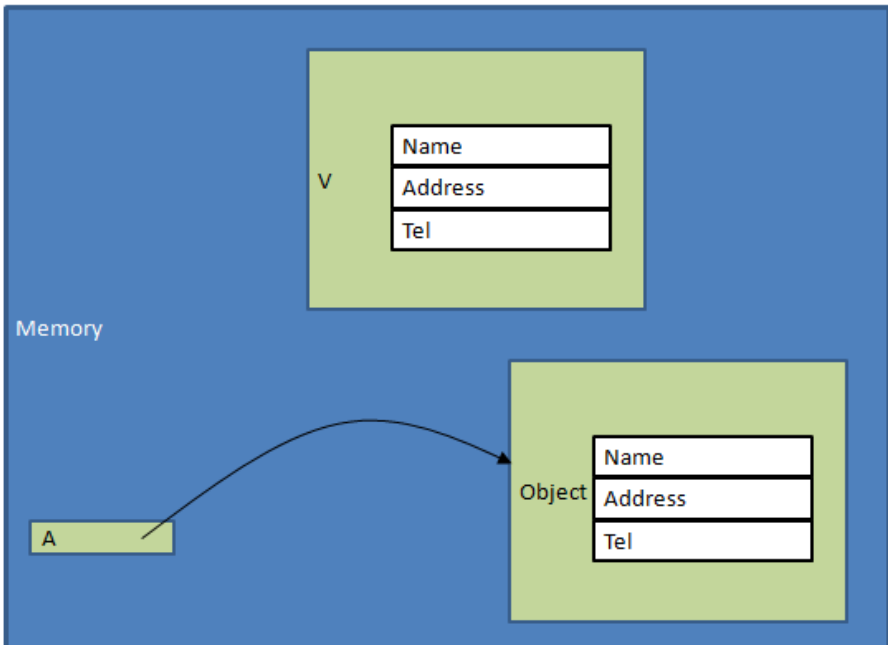
To clarify things more, Let us say we have a structure to store exactly the same information:

```
Structure ContactInfoStruct
    Dim Name As String
    Dim Address As String
    Dim Tel As String
End Structure
```

When you write

```
Dim V As ContactInfoStruct
```

Then what happens in memory is the following:



The variable V is allocated all the required memory resources. Unlike A, it only points to where the actual data are located in memory. So if there are not memory resources allocated, then A cannot point to them, and this is why you get the error. Now if you write:

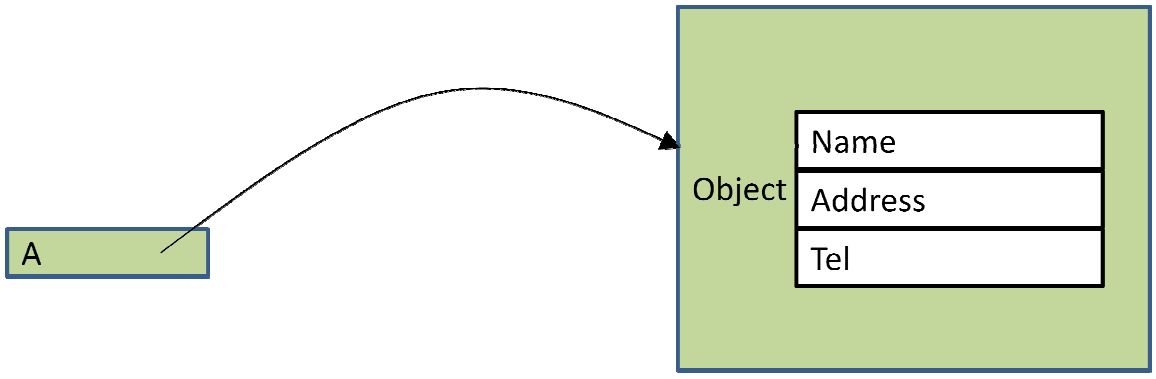
```
Dim A As ContactInfo
```

This creates a variable that points to no actual object:

But if you write:

```
Dim A As New ContactInfo
```

Then an object is created, and A points to it:



Now use the New keyword, and test the code, you will see it runs without an error.

```
Dim A As New ContactInfo
```

Another method to do it is by using two steps:

```
Dim A As ContactInfo
A = New ContactInfo
```

This will have exactly the same effect. It is up to you to select which way to use. However in some cases you need to use the second format specially if you want to create and destroy the object linked by the same variable multiple times.

Now go to the class file and write down the following:

```
    Public Sub SetContactInfo(ByVal NME As String, ByVal Addr As String, ByVal Telephone As String)
        Name = NME
        Address = Addr
        Tel = Telephone
    End Sub
```

This subroutine allows you to fill the variables in the class. It is a normal subroutine except for the `Public` keyword placed before it. This means that you can call this subroutine form any other code block. It is similar to using `Public` with variables. This is useful if you want to hide complex functions and subroutines from outside access and provide small number of function to use with your class. Now to test this subroutine, Modify the Button1 event handler to be like this:

```
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
Button1.Click
        Dim A As ContactInfo
        A = New ContactInfo
        A.SetContactInfo("Smith", "UK", "123456")
    End Sub
```

As you can see it is used the same way variables are accessed. You write the variable name (in this case A), followed by dot (.), followed by the function/subroutine (SetContactInfo). This is interpreted as call the function (SetContactInfo) and use the fields/attributes of A. If you the subroutine code:

```
Name = NME
Address = Addr
Tel = Telephone
```

This is translated to:

```
A.Name = NME
A.Address = Addr
A.Tel = Telephone
```

Now if you are using another object:

```
Dim B As ContactInfo
B = New ContactInfo
B.SetContactInfo("Michel", "US", "123456")
```

The subroutine call will be interpreted as:

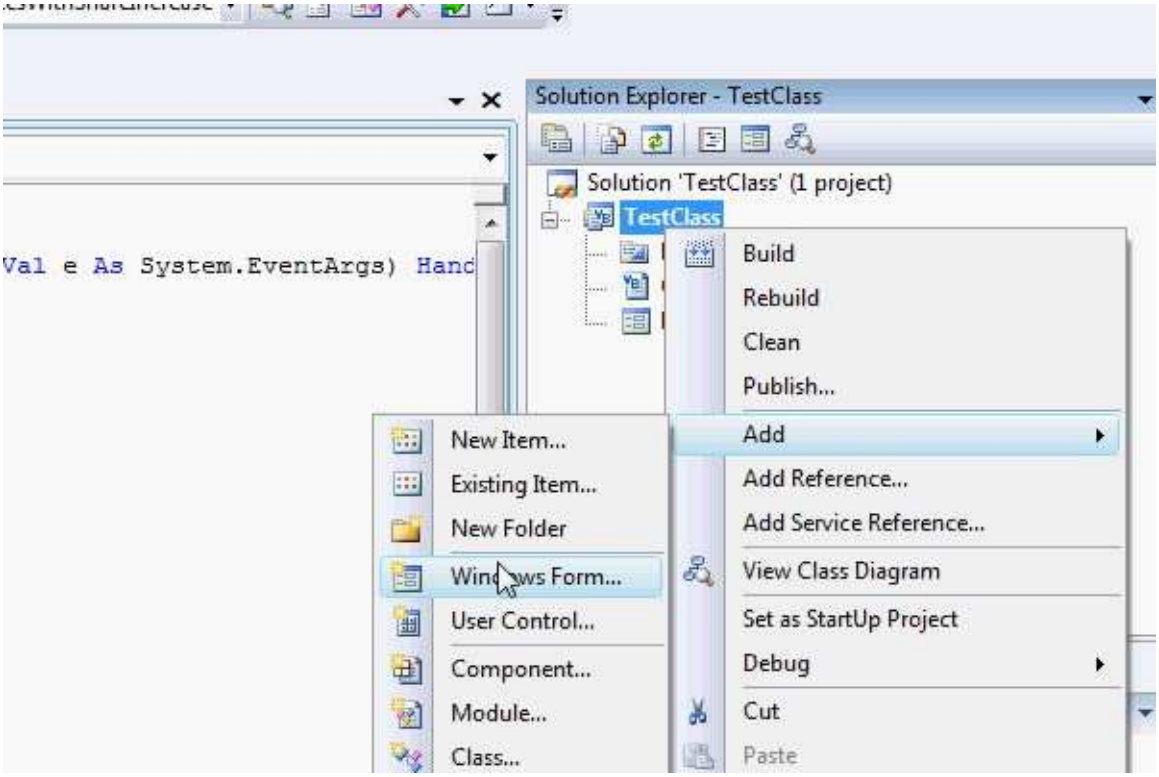
```
B.Name = NME
B.Address = Addr
B.Tel = Telephone
```

and so on.

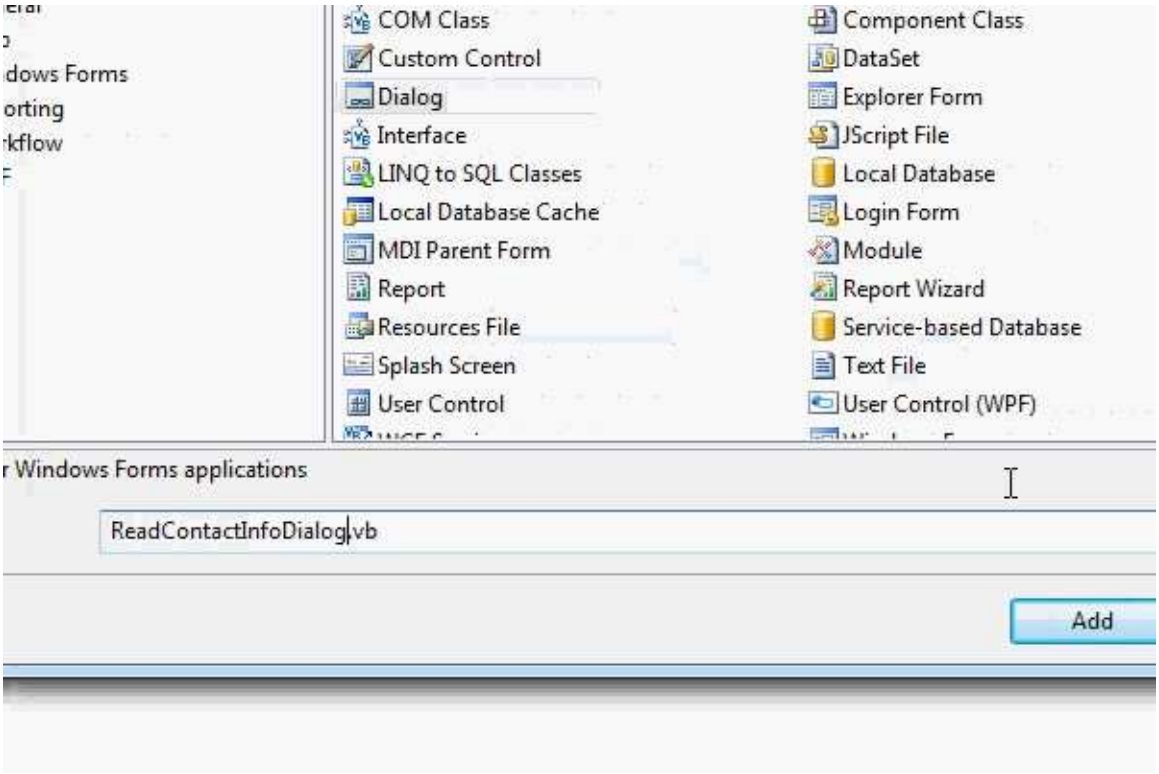
Now write down the following code in the event handler and run it:

```
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
Button1.Click
        Dim A As New ContactInfo
        Dim B As New ContactInfo

        A.SetContactInfo("Smith", "UK", "123456")
        B.SetContactInfo("Michel", "US", "456789")


        MsgBox(A.Name)
        MsgBox(A.Address)
        MsgBox(A.Tel)

        MsgBox(B.Name)
        MsgBox(B.Address)
        MsgBox(B.Tel)

    End Sub
```

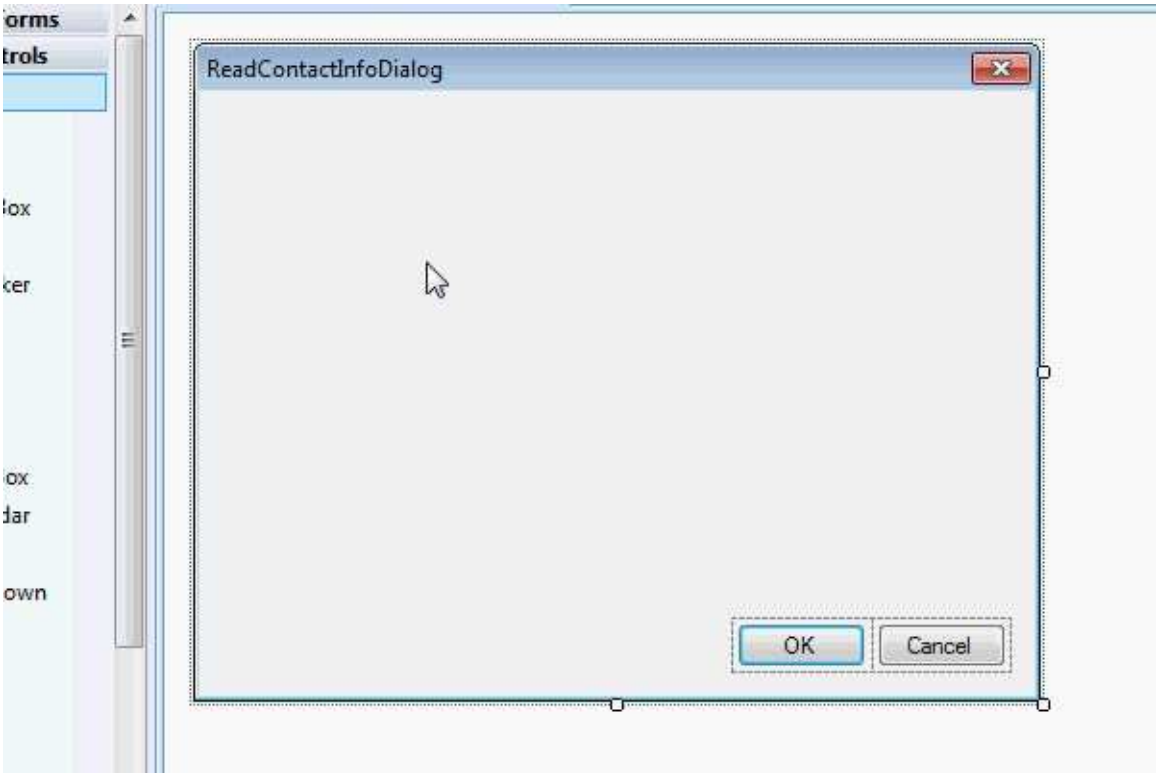
As you can see the code is easier to understand, and you don't have to fill the fields/attributes of the contact one by one. Now we will improve the way we enter the data by reading the information from a dialog. Right click your project and select Add->Windows Form



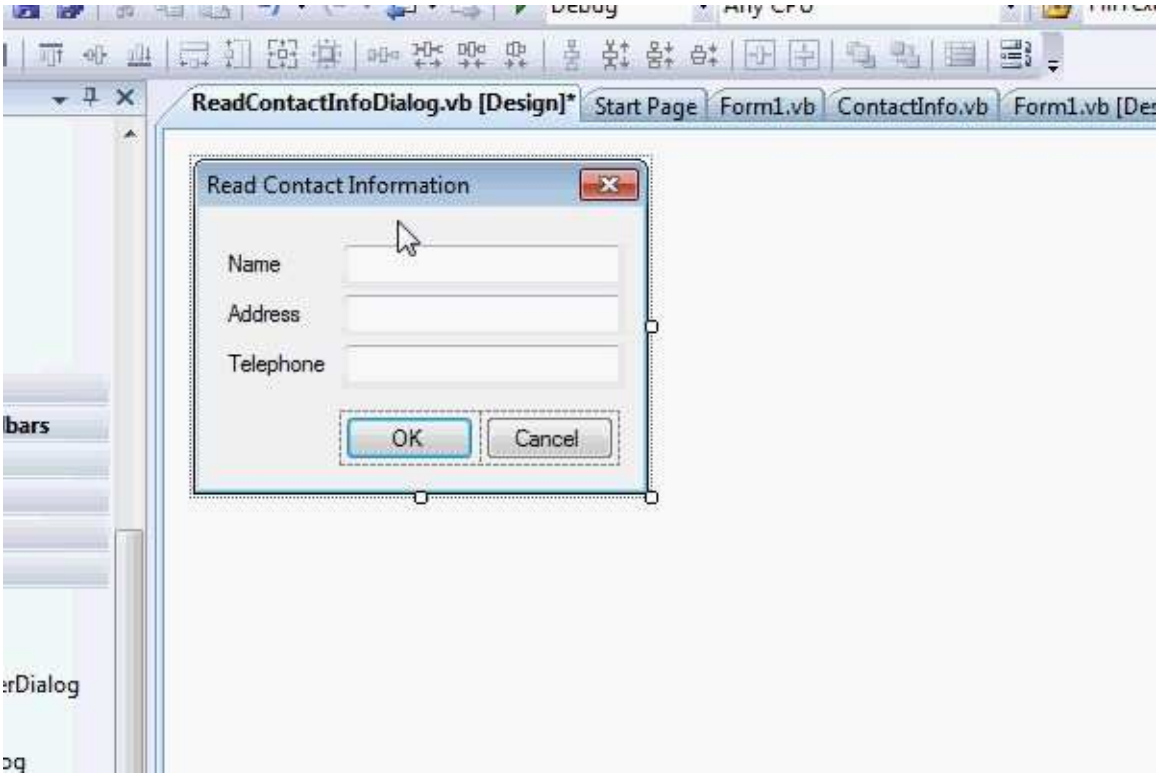
Select Dialog, and assign the name ReadContactInfoDialog:



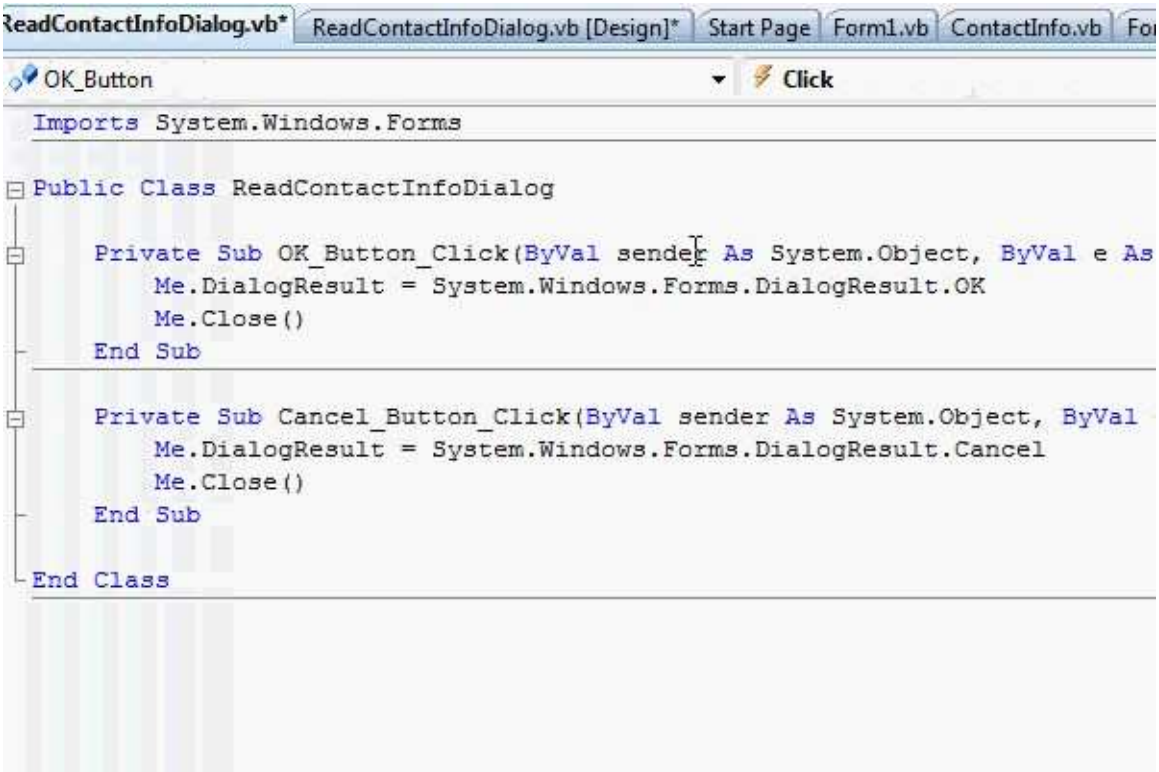
Then select Add, the dialog design appears.



Add three labels, and three text boxes, and make the dialog look like this:



If you check the code of the OK & Cancel buttons, you will find that it is already written. This code is the default behavior for a dialog, so leave it as it is.



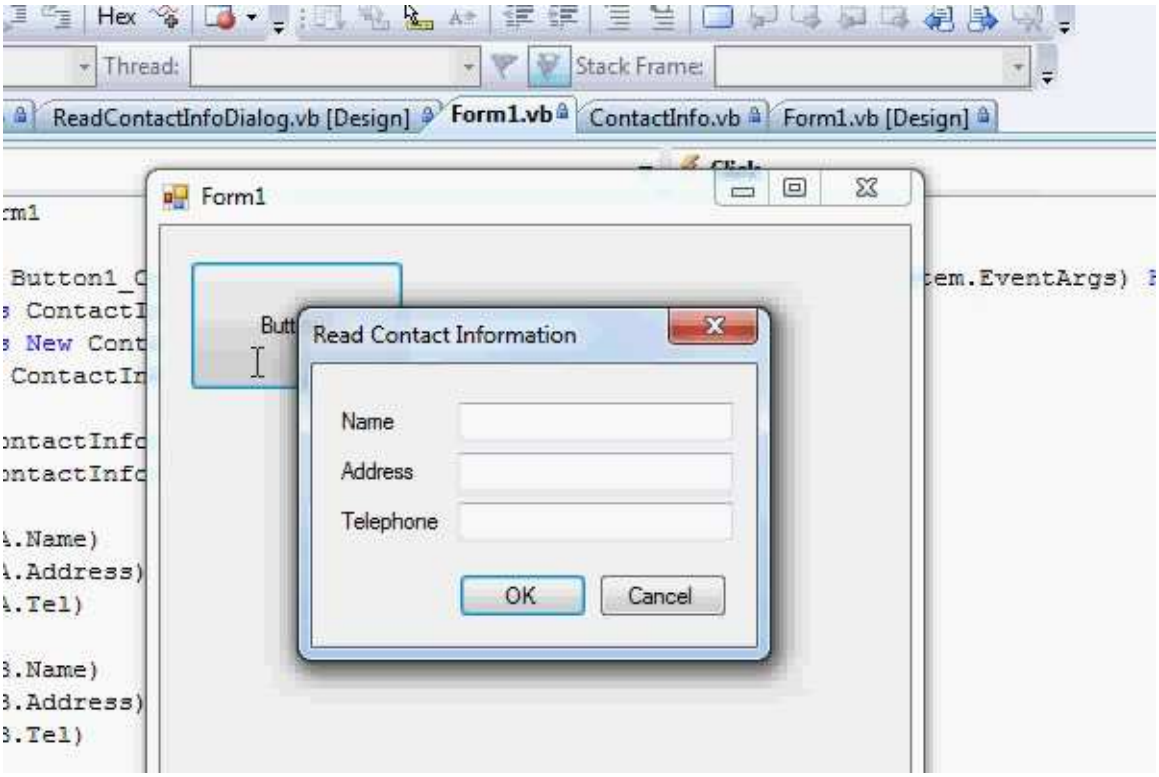
Next we will add a subroutine to read contact information. Go to the class file and write the following:

```
    Public Sub ReadContactInfo()

        ReadContactInfoDialog.TextBox1.Text = ""
        ReadContactInfoDialog.TextBox2.Text = ""
        ReadContactInfoDialog.TextBox3.Text = ""

        If ReadContactInfoDialog.ShowDialog = DialogResult.Cancel Then
            Exit Sub
        End If

        Name = ReadContactInfoDialog.TextBox1.Text
        Address = ReadContactInfoDialog.TextBox2.Text
        Tel = ReadContactInfoDialog.TextBox3.Text

    End Sub
```

The first part clears the text boxes from all previous input. The if statement part checks if the user hit the cancel button, and exits the subroutine if so. If not, the execution continues to the last part, there the content of the text boxes are copied into the variables of the class. To test it modify the code of the Button1 for the main window (Form1) to be like this:

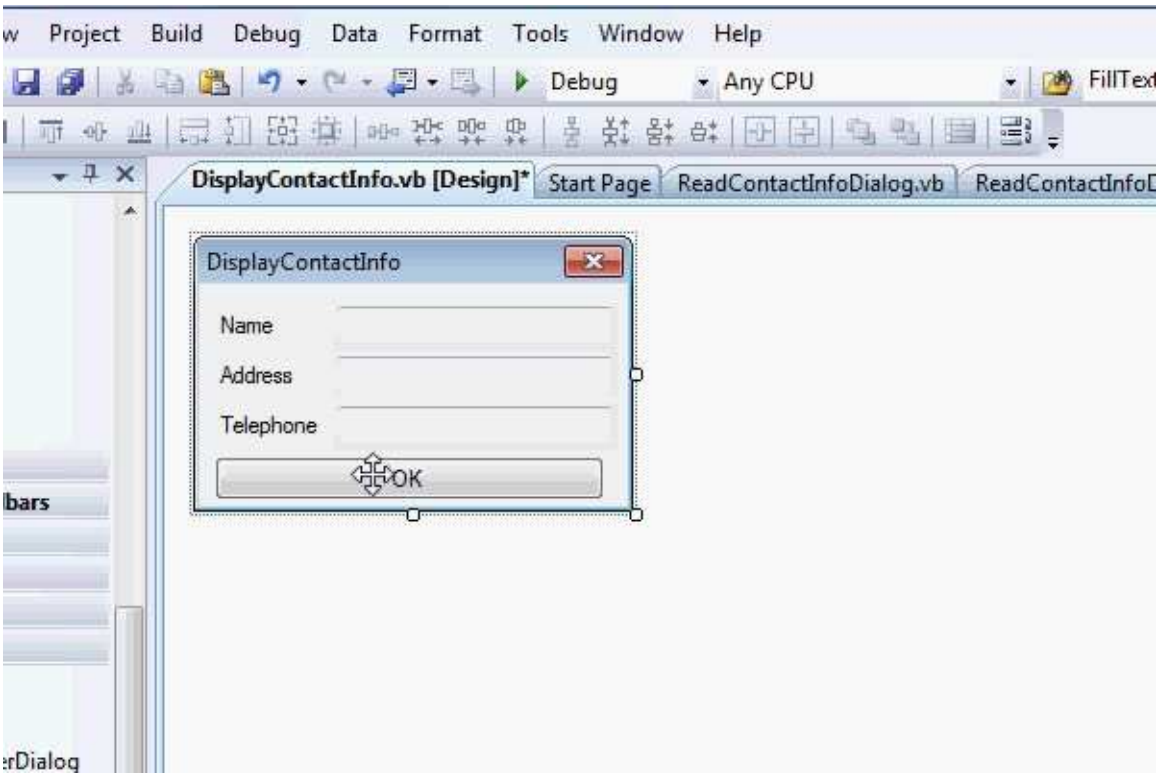
```
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
Button1.Click
        Dim A As New ContactInfo
        Dim B As New ContactInfo

        A.ReadContactInfo()
        B.ReadContactInfo()


        MsgBox(A.Name)
        MsgBox(A.Address)
        MsgBox(A.Tel)

        MsgBox(B.Name)
        MsgBox(B.Address)
        MsgBox(B.Tel)

    End Sub
```

Now run the code, and hit the button, you should see something like this:



Enter the information and hit OK. Another window appears, fill the information of the second contact and hit OK. After that you should be able to see the details of each contacts appear in separate message boxes.

Instead of using the message box to display the contact information, we will create another dialog to display such info. Just Add another dialog to the project as we did before and name it DisplayContactInfo. And make it look like the following:



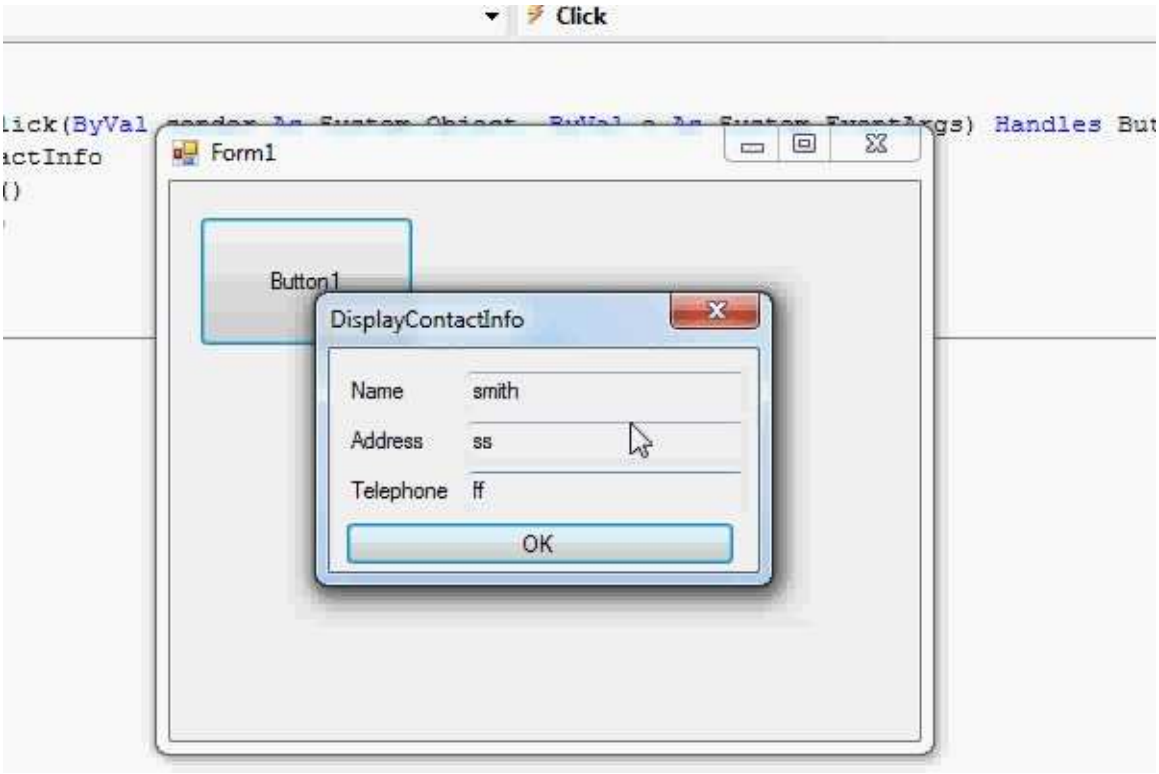
Make sure to only remove the cancel button, and keep the OK button there. Also make sure that all textboxes are read only. Go next to the class file and add the following subroutine:

```
    Public Sub DisplayContact()
        DisplayContactInfo.TextBox1.Text = Name
        DisplayContactInfo.TextBox2.Text = Address
        DisplayContactInfo.TextBox3.Text = Tel
        DisplayContactInfo.ShowDialog()
    End Sub
```

This is much smaller code since it just displays the information of the object. To test that, update the code of Button1 in Form1

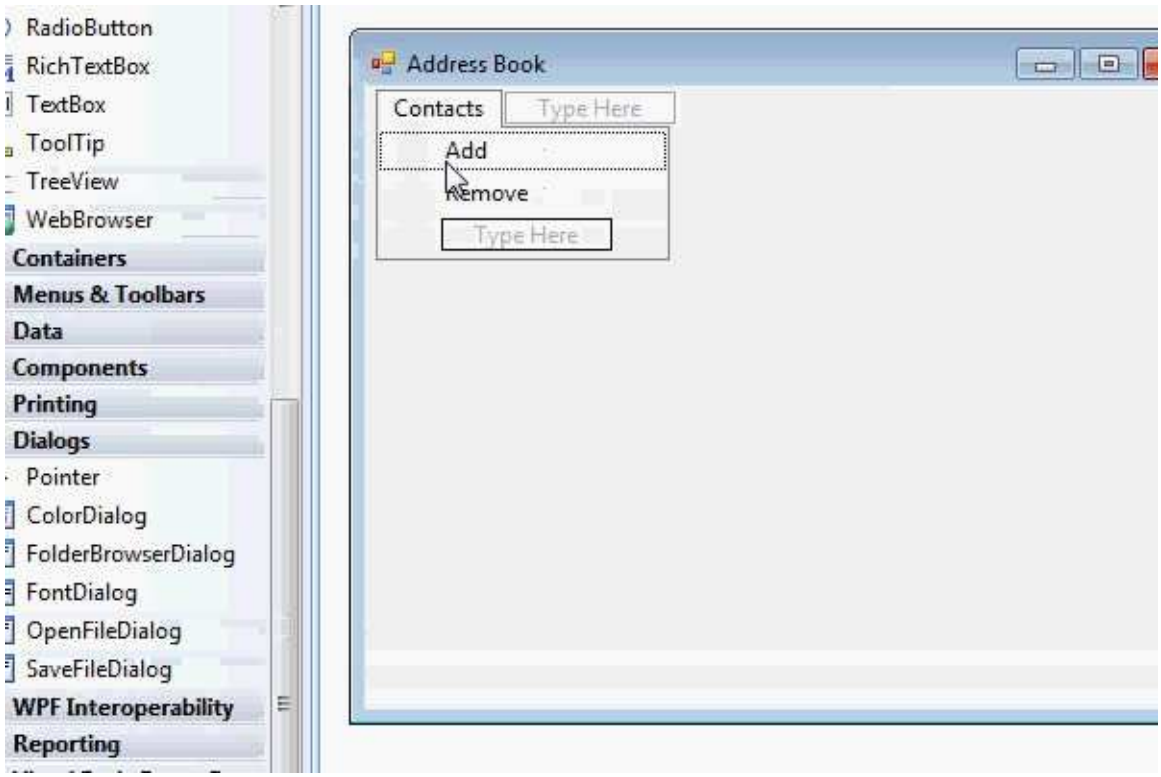
```
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
Button1.Click
        Dim A As New ContactInfo
        Dim B As New ContactInfo

        A.ReadContactInfo()
        B.ReadContactInfo()


        A.DisplayContact()
        B.DisplayContact()

    End Sub
```

Run the code and you will see that you can display the information in the form:

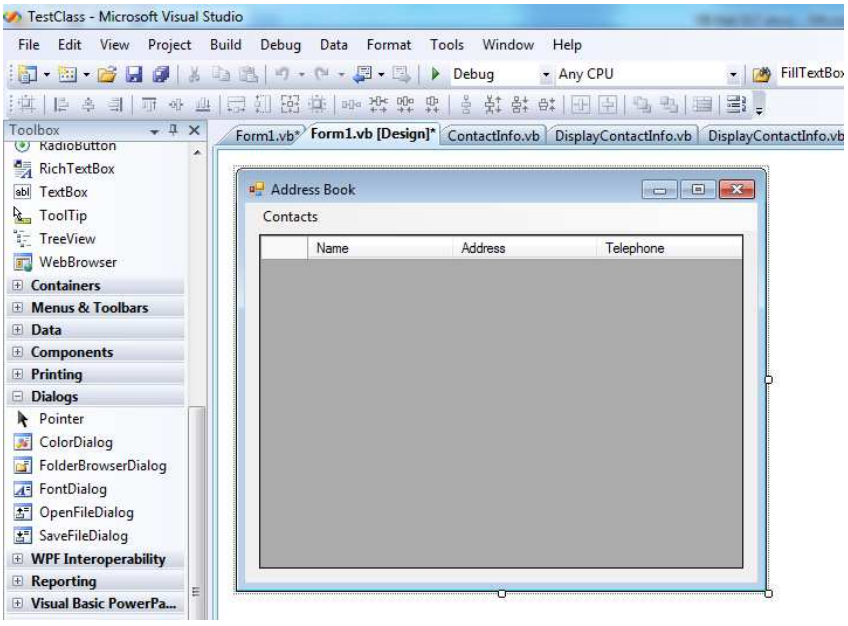


As you can see the code in Button1_Click subroutine is very straightforward and easy to understand. You don't have to worry about the internal details of the class. All you need is to break your problem/your program into a number of logical units/classes each has its own data and functions, and then you combine them together to solve the main problem. Classes makes such thing easier to do.

Now our simple class is almost ready, so we are starting to create the main user interface now. Remove the Button1 from the Form1 window and add a menu strip control. Create the menu entries shown below:



Also add a data grid view, and call it DGV, add three columns to it (one for name, one for address, and one for tel). Disable adding, editing and deletion of rows. You should have something similar to the following:



Double click the form and the editor opens, add the following code after Class Form1

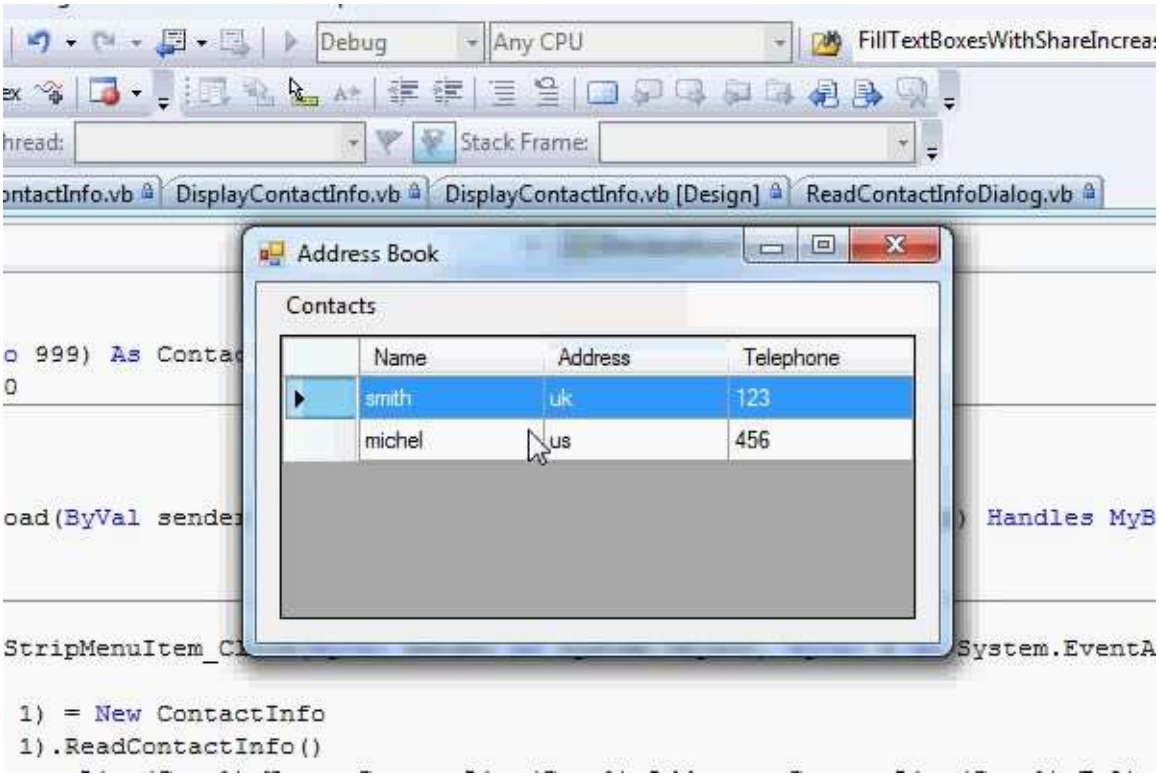
```
Dim ContactList(0 To 999) As ContactInfo
Dim C As Integer = 0
```

The ContactList is an array of type contact info. Each element of this array can point to an object of type ContactInfo, but when the array is created it is pointing to Nothing. C is used to tell how many objects are there in the array. When the program starts the number of elements is Zero.

Next add the following code to the Add menu item:

```
    Private Sub AddToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles AddToolStripMenuItem.Click
        C = C + 1
        ContactList(C - 1) = New ContactInfo
        ContactList(C - 1).ReadContactInfo()
        DGV.Rows.Add(ContactList(C - 1).Name, ContactList(C - 1).Address, ContactList(C - 1).Tel)
    End Sub
```

This subroutine will add new contact, read the information of that contact, and then update the display. Try this out and you should be getting something like this:



Now the remove code should be like this:

```
    Private Sub RemoveToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles RemoveToolStripMenuItem.Click
        If DGV.SelectedRows.Count = 0 Then
            Exit Sub
        End If

        Dim I As Integer
        Dim N As String
        N = DGV.SelectedRows(0).Cells(0).Value
        For I = 0 To C - 1
            If ContactList(I).Name = N Then
                DGV.Rows.Remove(DGV.SelectedRows(0))
                Dim J As Integer
                For J = I + 1 To 999
```

```
                ContactList(J - 1) = ContactList(J)
            Next
            C = C - 1
            Exit Sub
        End If
    Next
End Sub
```

Notice that we are removing the contact from the display and from the array itself. Try adding and removing few contacts and see how it works. So this concludes the tutorial for today. There will be more about classes in the next tutorial. However there is some important things that you must keep in mind. A variable of a class is a pointer only. A good example to understand this is if you write the following code:

```
Dim A As New ContactInfo
Dim B As ContactInfo
A.Name = "Smith"
B = A
B.Name = "John"
```

In the end of execution of such code, both A and B will have John as the name value. Any change to A or B will affect the other one because simply they both point to the same location in memory (point to the same object in memory). But if A & B are structures:

```
Dim A As ContactInfoStruct
Dim B As ContactInfoStruct
A.Name = "Smith"
B = A
B.Name = "John"
```

Then A will be independent of B and changes in A will not affect B and vice versa.

So this is all for today. If you need the source file, you can get it from the web site. If you have notes about this tutorial, email me at: **notes@mka-soft.com**.

Thanks.

mkaatr