Learning VB.Net

Tutorial 19 – Classes and Inheritance

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-17.

Classes and Inheritance

In the previous two tutorials, the definition of classes, methods, and their initialization is discussed. This tutorial is about how to perform inheritance. The same example used in the last tutorial is being used here as well.

In many cases you want to take an existing class and extend its functionality. In our previous example the class **ContactList** has **ContatArr()** which is an array used to store the contacts, and the counter **C** which is used to tell how many elements we are using in the array. It also has methods to add a contact, remove a contact, and display the contacts in a **DataGridView**. What we want here is to create a new class that has the same methods and properties as **ContactList**, and also has a **Sort** method which allows you to sort the contacts by name.

To do so simply add another class to your project, and call it **`ContactsWithSort`**. The first line of code in the class should be:

```
Inherits ContactList
```

The keyword **Inherits** here tells the complier that the class has behave in the same way as **ContactList**. In other words it is like copying the code of ContactList and pasting it to the new class (for now you can think about it like this, it makes things easier).

Now let us try to add the new method to the class…

```
Public Sub Sort()           ' this class has another method called sort.
    Dim I As Integer
    Dim F As Boolean
    Dim Contact As ContactInfo

    Do
        F = False

        For I = 0 To C - 2
            If ContactArr(I).Name > ContactArr(I + 1).Name Then
                F = True
                Contact = ContactArr(I)
                ContactArr(I) = ContactArr(I + 1)
                ContactArr(I + 1) = Contact
            End If
        Next

    Loop While F

End Sub
```

Now if you try to run the program (even though you did not use the Sort method or the new class itself) you will get an error. The error is for using **C** and **ContactArr**. The error says that these variables are private. This brings up the issue of the accessibility of variables.

When you define a variable in a class you can set its accessibility level to the following:

Public : this means that the variable can be access inside or outside the class.

Private: this means that the variable can be accessed only inside the original class it is created in.

Protected: this means that the variable can be accessed only in the class and all inherited classes.

So let us check this using the following example:

```
Public Class test
    Dim A As Integer
    Private B As Integer
    Public C As Integer
    Protected D As Integer

    Public Sub SetA(ByVal I As Integer)
        A = I
    End Sub

    Public Sub SetB(ByVal I As Integer)
        B = I
    End Sub

    Public Sub SetC(ByVal I As Integer)
        C = I
    End Sub

    Public Sub SetD(ByVal I As Integer)
        D = I
    End Sub

End Class
```

In the example, **A** is treated as private. So if you add this method to the class:

```
Public Sub SetA(ByVal I As Integer)
    A = I
End Sub
```

It works perfectly fine. However, if you add the following code into a form or module:

```
Dim Q As New test
Q.A = 10
```

This would trigger an error because **A** should only be accessed from within the class. Now let us check **B** which is private. If this is a method in the class, then it works.

```
Public Sub SetB(ByVal I As Integer)
    B = I
End Sub
```

But if you add the following code into any place other than the class test, you get an error.

```
Dim Q As New test
Q.B = 10
```

So it works exactly like private. Next let us try to work with **C** which is Public.

```
Public Sub SetC(ByVal I As Integer)
    C = I
End Sub
```

This obviously works fine since it is in the same class (test). If you write the following code in any other place other than the class test, then it works perfectly fine.

```
Dim Q As New test
Q.C = 10
```

This works because the variable C here is public which means it can be accessed from any other place. Now let us check the last one D which is protected. The method within the class again has no problem

```
Public Sub SetD(ByVal I As Integer)
    D = I
End Sub
```

If you want to access the variable D from outside the class it is treated like private, but it has some special treatment, which we will see later.

```
Dim Q As New test
Q.D = 10
```

So this triggers an error. Now let us go back to our example and see why we can't access the variable **C** and **ContactArr**. We used (Dim) for these two which means they are treated like private. As we have seen before that private variables in a class can not be accessed from outside the class itself. So we want to make them accessible. Making these variables public means that they will be accessed from any part of the project, which is not a good idea. If you change these variables' visibility to protected, then the classes inherited from them will be able to access these. An access from any other location is denied. To test this try to create a class test2 inherited from test.

```
Public Class test2
    Inherits test

    Public Sub SetAll()
        A = 10     ' error
        B = 20     ' error
        C = 30     ' correct
        D = 40     ' correct
    End Sub
End Class
```

Here A is not accessible in this class simply because it is private in the original class. B is the same so it causes the same problem. C is public in class test, so it is accessible here and everywhere else. D is protected so it is accessible in test2. The table below summarizes how these work:

Accessibility	Base Class	Inherited Class	Outside the Class
Dim	Accessible	Not Accessible	Not Accessible
Private	Accessible	Not Accessible	Not Accessible
Public	Accessible	Accessible	Accessible
Protected	Accessible	Accessible	Not Accessible

So going back to our example, set each of **C** and **ContactArr** in **ContactList** class to **protected**. You will see the code now is correct.

Next, modify the object in the form to use the new class:

```
    Dim OBJ As ContactsWithSort
```

And modify the code of initialization of OBJ in the load event of the form:

```
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
MyBase.Load
        OBJ = New ContactsWithSort()
        OBJ.FillDGV(DGV)
    End Sub
```

Finally, add a menu item to sort the contacts, and write the following in the event handler:

```
        OBJ.sort()
        OBJ.FillDGV(DGV)
```

Run the code check it out. Below is the full code of the **ContactList** class:

```
Public Class ContactList

    Protected ContactArr() As ContactInfo      ' the array of object, all elements points to nothing
    Protected C As Integer                     ' the number of objects in the array

    Public Sub AddNewContact()
        C = C + 1                              ' the number of objects increases by one
        ContactArr(C - 1) = New ContactInfo    ' create the object
        ContactArr(C - 1).ReadContactInfo()    ' read the information
    End Sub

    Public Sub RemoveContact(ByVal Name As String)
        ' search for the contact
        For I = 0 To C - 1
            If ContactArr(I).Name = Name Then


                ' next remove the contact from the array by shifting the other objects
                Dim J As Integer
                For J = I + 1 To 999
                    ContactArr(J - 1) = ContactArr(J)
                Next

                ' the number of elements reduces by one
                C = C - 1

                ' exit the block
                Exit Sub
            End If
        Next

    End Sub


    Public Sub FillDGV(ByVal DGV As DataGridView)
```

```
        ' clear the data grid view
        DGV.Rows.Clear()

        Dim I As Integer

        ' loop over all the contacts
        For I = 0 To C - 1
            ' add contact information
            DGV.Rows.Add(ContactArr(I).Name, ContactArr(I).Address, ContactArr(I).Tel)
        Next

    End Sub

    Public Sub New()
        ' first constructor, set the number of elements to zero, and set array size to 1000
        C = 0
        ReDim ContactArr(0 To 999)
    End Sub

    Public Sub New(ByVal NoOfReads As Integer)
        ' second constructor, set number of elements to zero, and set array size to 1000
        C = 0
        ReDim ContactArr(0 To 999)

        ' add the contacts
        Dim I As Integer
        For I = 0 To NoOfReads - 1
            Me.AddNewContact()
        Next
    End Sub

    Protected Overrides Sub Finalize()
        ' this is how to terminate a class
        Dim I As Integer
        For I = 0 To C - 1
            ContactArr(I) = Nothing
        Next


        MyBase.Finalize()
    End Sub
End Class
```

Next is the code for the **ContactsWithSort** class

```
Public Class ContactsWithSort

    Inherits ContactList
         ' this tells the compiler that this class has the same behaviour of ContactList

    Public Sub Sort()           ' this class has another method called sort.
        Dim I As Integer
        Dim F As Boolean
        Dim Contact As ContactInfo

        Do
            F = False

            For I = 0 To C - 2
                If ContactArr(I).Name > ContactArr(I + 1).Name Then
                    F = True
                    Contact = ContactArr(I)
                    ContactArr(I) = ContactArr(I + 1)
                    ContactArr(I + 1) = Contact
                End If
            Next

        Loop While F

    End Sub

End Class
```

And finally the code of the form:

```
Public Class Form1

    Dim OBJ As ContactsWithSort


    Private Sub AddToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles AddToolStripMenuItem.Click
        OBJ.AddNewContact()
        OBJ.FillDGV(DGV)
    End Sub

    Private Sub RemoveToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles RemoveToolStripMenuItem.Click
        ' check if no rows are selected, if so no need to execute further code, just exit the
subroutine
        If DGV.SelectedRows.Count = 0 Then
            Exit Sub
        End If

        Dim N As String

        ' get the selected name, it is the first column (cell zero)
        N = DGV.SelectedRows(0).Cells(0).Value

        OBJ.RemoveContact(N)
        OBJ.FillDGV(DGV)
    End Sub

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
MyBase.Load
        OBJ = New ContactsWithSort()
        OBJ.FillDGV(DGV)
    End Sub


    Private Sub SortToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles SortToolStripMenuItem.Click
        OBJ.sort()
        OBJ.FillDGV(DGV)
    End Sub
End Class
```

The rest of the files don't need modification, they are the same. So as you can see inheritance allows us to extend the functionality of an existing class, and add some features to them. 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