CS608 Lecture Notes

Visual Basic.NET Programming

Object-Oriented Programming – Creating Custom Classes & Objects

(Part I)

(Lecture Notes 2A)

Prof. Abel Angel Rodriguez

Chapter 5 Introduction to Object-Oriented Programming

5.1 Components of an Object-Oriented Program

5.1.1 Understanding Classes & Objects

- ❑ Real world objects have attributes or properties that define the objects.
- ❑ Also, real world objects are based on some mold or template.
- ❑ In Object-Oriented programming, the objects are based on a **class** or template. In this section we take a look at the components that make up an Object-Oriented Program

The Class

- ❑ The mechanism VB.NET provides to implement ***Objects*** is the ***Class***.
- ❑ A ***Class*** is a template or blueprint that defines what ***Object*** of the class look like.
- ❑ A ***Class*** is a plan or template that specifies what ***Data*** , ***Methods*** & ***Events*** will reside in ***objects***
- ❑ The objects of the class contain ***data*** and the ***Methods*** (member functions & procedures) that operate on such data
- ❑ For example:
  - ▪ We can have a Class called *Automobile*, and from this class, we can define the Properties, Methods and Events of this class.
  - ▪ From this Automobile class we can create Objects of this class such as a Car Object, Truck Object, SUV Object etc.

Objects

- ❑ Think of Objects as a thing or a noun. Objects are the items that represent real-world entities, such as a person, place or thing in a program.
- ❑ In a program an ***Object*** is a software representation of a real-world entity.

Objects - vs - Class

- ▪ The concept of a Class an Object can be very confusing. A Class is NOT an Object. An Object is not a Class
- ▪ DO NOT confuse a ***Class*** with the ***Objects***, they are two different things.
- ▪ ***Objects*** are the manifestation or *instance* of a ***Class*** specification.
- ▪ A class **IS NOT** the object, but the template in which ***Objects*** will be created from!
- ▪ **Think of the *class* as the architectural floor plan of a house, and the *objects* as the houses that are built from that plan**.
- ▪ Objects behave exactly as they were specified in the Class. No more, no less

Properties or Attributes

- ❑ Properties represent the ***Data*** of the Object.
- ❑ In reality, the Property is the way the outside world access the actual data directly.
- ❑ This is confusing; in reality, the property is not the data, but a vehicle to access the data. The actual data is private and cannot be seen by the outside world, but to the outside world, the property is what they see as the data.
- ❑ For example a Car Class Object may have a ***color*** property, as well as a ***Make*** & ***Model*** property. But inside the Object, the actual data may be private variables called *carColor*, *carMake* & *carModel* etc. but to the outside world, when they want to use the data they see the property *Color*, *Make* & *Model*.

Methods

- ❑ **Methods** are actions that the Objects can take. Where Objects are the Nouns, Method are the verbs or actions of an Object.
- ❑ For example a Car Class Object can take the following actions: *Start*, *Stop*, *Speed Up*, *Slow Down*, *turn left*, *turn right* etc.
- ❑ Methods are Functions and Sub Procedures that you write to make the object do things or take some kind of action

Events

- **Events** are actions taken upon the object by an outside force (User, Program code etc).
- **Events** or actions upon the object will automatically trigger specialized Methods known as ***Event-Handlers***
- Do not confuse events with regular Methods. Events are the action taken by an outside source upon the object, while Methods are action taken by the Object itself.
- **Events** & ***methods*** may work hand in hand, but they are two different things.

  - This can be confusing. For example an Object such as a Car Object can have a method called *Stop()*. You can explicitly call the *Car.Stop()* method to so the car will stop itself.
  - Now suppose your Car Object also has a method called Drive(), and you can explicitly call the Car.Drive() method so the car will drive.
  - On the other hand, The Car Object can also have an Event programmed into it called *OnCrash* and lets supposed that we program this event to trigger or raise during the Drive() method only if there is an accident.
  - Now, in the event that the car is hit by another car or crashes during the execution of the Drive() method, the *OnCrash* event will automatically trigger and an Event-handler ***Object_OnCrash()*** will appear in the Form the object was created in.
  - When the *OnCrash()* event executes you can code in what ever you like inside this *Event-Handler* ***Object_OnCrash()***. For example you may want to put in a statement to call the *Car.Stop()* method to stop the car. Makes sense right? Here an Event occurs, triggers the Event-handler, in the Event-Handler we call a Method.
  - Confused yet? ☺

5.1.2 Creating Object-Oriented Programs

- Object-Oriented Programs (OOP) are written based on the Class Objects and not on the functionality of the program
- The three steps required to creating an Object-Oriented Programs are shown below:

I. **Create the class specification or Class Module**
   - Private Data, Properties & Methods

II. **Create Object of the Class**

III. **Use the Object of the Class**
   - Write the program to manipulate, access or modify the objects data & Call the Methods & and Trigger Events

5.2 Object-Oriented Program Implementation – A Three Part Method

5.2.1 Part I - Creating a Class

Class Components

- The components, which make up a class are: 1) ***Data***, 2) ***Property's***, 3) ***Methods*** & 4)***Events***
- You are probably confused with the word ***Property*** & ***Data***, it seems that the ***Properties*** & ***Data*** are the same thing? They are NOT, lets look at the components definition:

1) ***Data:*** The <u>variables</u> & <u>Data Structures</u> *(Arrays etc) or* other <u>Objects</u> that hold the information or data we want to manipulate and store. Usually a <u>*private*</u> variable
2) ***Property:*** Specialized <u>procedures or routines</u> whose sole purpose it to give you access to the private data inside the class, in other words it is through these Property that you can modify and extract the <u>*private*</u> data of a ***Class Module***. Through the outside world, the data is represented via the Properties.
3) ***Methods:*** The *functions/procedures* that make the object of the class do things or take an action.
4) ***Events:*** Declaration of the action that when taken upon the object will trigger an Event-Handler Procedure

- Therefore in a Class, the <u>*variables*</u> store the **Data** while the ***Methods*** & ***Property*** are both procedures whose difference is that ***Methods*** are called to make the Object do things or take an action upon the data, while ***Properties*** allow you set & get the values of the private data directly.

Accessibility of the Class Components

- Accessibility refers to what type of access does the Data, Property or Method allows to the outside world.
- VB.NET Classes offer the following access characteristics:

- **Public** – Public or can be access by anyone outside of the class or to other objects
- **Private** – Private or accessible only to members of the class. No one from the outside world can see it or access it.
- **Protected** – Allows subclasses or inherited children to have direct access but only the children or derived classes. In other words, public for children, private for everyone else.
- **Friend** – Accessible to classes in the same <u>*Assembly*</u>. An Assembly is a collection of project deployed together as an application.

General Rules for Assigning Accessibility

- In order to implement the ***Data Encapsulation*** feature of an object there are rules that must be followed when creating classes.
- The rules are as follows:

- ***Private Data Variables:*** Variables declared using the ***Private*** keyword.
- ***Public Properties:*** Property procedures are declared using the ***Public*** keyword, so the outside world has access to the data.
- ***Public Methods:*** Methods procedures are declared using the ***Public*** keyword, so the outside world can execute them
- ***Public Events:*** Events are declared using the keyword ***Public***.
- ***Protected Data Variables & Methods:*** Declare variables/methods as protected only when you want the inherited children to have direct access.

❖ Note that Property & Methods can be Private as well, but this means they cannot be accessed outside the class and can only be used internally in the class. There are circumstances where you want them to be private, more on this on later lectures.

Creating a Class Module – A Step-by-Step Approach – Part I

- In order to create ***Object*** for your programs, you need to first create the object template or ***Class*** or Template.
- A ***Class*** is a template or blueprint that define what object of the class look like
- A ***Class*** is a plan or template that specifies what ***Properties***, ***Methods*** and ***Events*** that will reside in ***objects***
- The syntax for creating a class is as follows:

```
 'Class Header
Public Class  ClassName
    Data Definitions

    Properties Definitions

    Methods

End Class
```

Example:

- **Creating a Classes:**

- Example 1 **-** Creating a Video Class:

```
Public Class Video
'Properties, Methods & Event-Procedures here

End Class
```

- Example 2 **-** Creating a class for a Form Object:

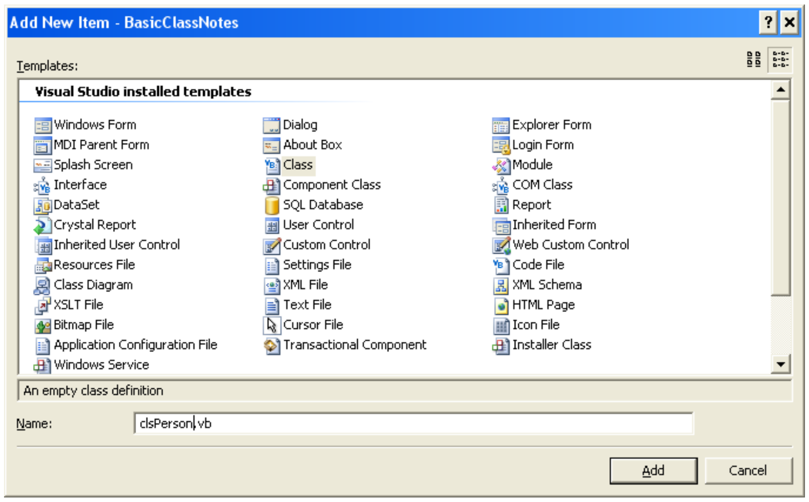
```
Public Class Form1
'Properties, Methods & Event-Procedures here

End Class
```

❖ **Note that inside the body of a Class is where you find all properties, methods & event-procedures for that object.**

- The actual steps required to create a class using the VB.NET IDE environment are as follows:

Step 1: If not yet created, create a New Project and if necessary the Forms, Controls, Program Code etc.

Step 2: In the Menu bar use ***Project | Add Class***, and select the ***Class icon***, *give the class a* ***Name*** and click ***Open***:

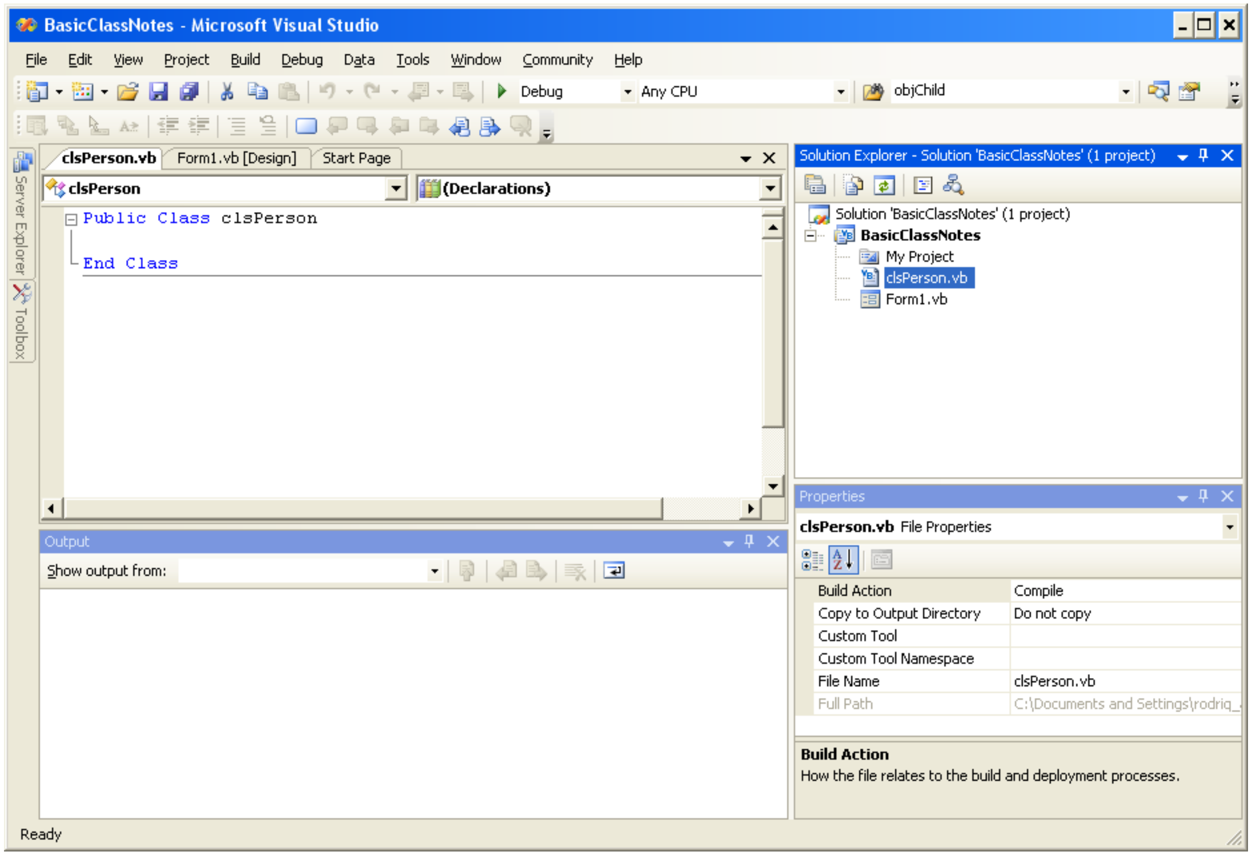


An empty class definition

Name: clsPerson.vb

Step 3: The ***Class* code** window will appear ready for you to begin to enter the program code:

- Also, a listing of the Class file will appear in the *Solution Explorer Window*
- Property Windows will display the *Class Properties*.



```
Public Class clsPerson

End Class
```

Build Action	Compile
Copy to Output Directory	Do not copy
Custom Tool	
Custom Tool Namespace	
File Name	clsPerson.vb
Full Path	C:\Documents and Settings\rodriq_

Build Action
How the file relates to the build and deployment processes.

Creating Data for the Class – Part II

- ❑ The top or declaration section of the Class Module is where the data variables are declared.
- ❑ These variables are usually declared private and therefore can only be seen by the code within the class.
- ❑ Note that this class data can be any of the following:
- ▪ Variables
- ▪ Data structures such as arrays etc..
- ▪ Other Objects such as Class Objects & Collections etc.

- ❑ Continuing our step by step approach, the syntax is as follows:

Step 4: In the Class Module Code Window enter the Private Data Variables:

- ❑ This is simply creating variables as you have done before:

Private *VariableName* As *DataType*

Example 1:

- ❑ Declaring private data members of the class:

Public Class *clsInvoice*
```
      Private m_Name As String
      Private m_Total As Decimal
      Private m_SubTotal As Decimal
      Const Private m_TAX As Decimal = 0.825
      Private m_objInvoiceItems As New clsInvoiceItems   'Class Object
```
End Class

- ❑ In Example 1, we created the following private data for the clsInvoice Class:
- ▪ Regular variable such as string, and decimals.
- ▪ An object variable of the clsInvoiceItems class. This means objects of the INVOICE CLASS ALSO HAVE A CHILD OBJECT NAME OBJINVOICEITEMS

Example 2:

- ❑ Declaring private data members of the class:

Public Class *clsCustomer*
```
      Private m_FirstName As String
      Private m_LastName As String
      Private m_CustomerID As Integer
      Private m_BirthDate As Date

      Private m_objCreditCard As clsCreditCard   'CreditCard Class POINTER
      Private m_objInvoice As New clsInvoice   'Invoice Class OBJECT
```
End Class

- ❑ In Example 2, we created private data for the ***clsCustomer*** Class as follows:
- ▪ Regular variable such as string, integer & date.
- ▪ This class also contains declarations for two CHILD OBJECTS, one the ***objCreditCard*** Object is actually a POINTER TO A ***clsCreditCard*** OBJECT not an object, yet. Eventually is expected that this pointer will point to a *clsCreditCard* object
- ▪ The second CHILD OBJECT is a full object of the *clsInvoice* class.

Naming Convention for Private Data

- There are several naming conventions used today to name the data.
- As shown in the examples above, one popular convention uses **m_** as a prefix to the private data name to indicate a module level variable (variable only seen within the Class module)
- I will use this as well in most of my examples. In addition, for private data that are objects, I will also use a combination of the **m_** and the **obj** prefix as well, to indicate the variable is an object. This is not standard, but I like to be able to identify private data that are object types right away.

Declaring Events for the Class – Part III

- The top or declaration section of the Class Module is where the declaration of the Events associated with this class is declared.
- This will be covered in future lectures.
- Continuing our step by step approach, the syntax is as follows:

Step 5: In the Class Module make the declarations for the **Events:**

- Covered in future lecture:

Creating Properties for the Class – Part IV (a)

- The ***Properties Procedures*** are the vehicles for the outside world to access the private data in the class.
- From the outside of the object, what the program sees is just the name of the property.
- The Property Procedure allows you to GET & SET the attributes or private data of the class.
- **Normally you will need a property for each private data member!**
- Continuing our step by step approach, the syntax is as follows:

Step 6: Create the Class **Property** Procedures

- Syntax for ***Public Property*** Procedures:

```
Public Property PropertyName () As DataTypeOfPrivateData
        Get
            Return PrivateData
        End Get

        Set ( ByVal VariableName As DataType)
            PrivateData = VariableName
        End Set
End Property
```

- Note that using the Visual Studio ID, once you begin and type the header of the property and hit return, the remaining syntax will appear automatically, you simple need to modify it. Let's see how this works with an example:

1. Assuming we have the following Class with a m_Name private data:

```
Public Class clsPerson
    Private m_Name As String

End Class
```

2. Now we will create a public property for this private data.
3. First type the header of the property you want to create:

```
Public Class clsPerson
    Private m_Name As String

      Public Property Name() As String

End Class
```

4. Hit the RETURN key and the following syntax appears automatically:

```
Public Class clsPerson
    Private m_Name As String


    Public Property Name() As String
        Get

        End Get
        Set(ByVal value As String)

        End Set
    End Property
End Class
```

5. Now modify the property by adding the Get code to return the private data, and the SET code to modify the private data:

```
Public Class clsPerson
    Private m_Name As String


    Public Property Name() As String
        Get
            Return m_Name
        End Get
        Set(ByVal value As String)
            m_Name = value
        End Set
    End Property

End Class
```

- Since normally you have one property for every private data, it can be cumbersome to have to create all these properties. Using this short-cut we can save some time during the creation of the properties.

Example 1: Property for regular private data member

- Declaring Property Procedures for a **Name** Property which allows access to a private string variable ***m_Name***. Note that the outside world sees only the **Name** of the property:

```
Public Class clsPerson
      Private m_Name As String

      Public Property Name () As String
            Get
                  Return m_Name
            End Get

            Set ( ByVal Value As String )
                  m_Name = Value
            End Set
      End Property
End Class
```

===
=

- Now from the outside world you can make the following statements assuming you created an object named *objEmployee* of the Class *clsPerson*:

```
'Example of Setting the object's property:
   objEmployee.Name = "Joe Smith"

'Example of Getting the object's property:
   Dim strEmployeeName As String

   strEmployeeName = objEmployee.Name

'Example of displaying the object's property using two different ways:

   MessageBox.Show ("The Employee Name is " &  strEmployeeName)

   MessageBox.Show ("The Employee Name is " &  objEmployee.Name)
```

Example 2: Properties for Private Object Data

- Declaring Property Procedures for a **CreditCard** Property which allows access to a private Object variable ***objCreditCard***. We assume that this object variable is of the class *clsCreditCard* Class:

```
Public Class clsCustomer
        Private m_FirstName As String
        Private m_LastName As String
        Private m_CustomerID As Integer
        Private m_BirthDate As Date
        Private m_objCreditCard As New clsCreditCard   'Class Object

        Public Property CreditCard () As clsCreditCard
                Get
                    Return m_objCreditCard
                End Get

                Set ( ByVal Value As clsCreditCard)
                   m_objCreditCard = Value
                End Set
        End Property
End Class
```

==
=

- Now from the outside world you can make the following statements assuming you created two objects named *objNewCreditCard* and *objCurrentCreditCard* of the Class *clsCreditCard* and an Object *objCustomer* of the *clsCustomer* Class:

```
'Example of setting an object's property that is an object as well:
   objCustomer.CreditCard = objNewCreditCard

'Example of Getting the object's property:

   objCurrentCreditCard = objCustomer.CreditCard
```

Creating Read-Only & Write-Only Properties for the Class – Part IV (b)

Read-Only Properties

- There are times when we may want a property to be Read-Only or it cannot be changed.
- In this case we simply remove the Set portion of a ***Property Procedures***.
- Continuing our step by step approach, the syntax is as follows:

Step 7: Create the Class **Property** Procedures

- Syntax for ***Public Read-Only Property*** Procedures:

```
Public Property PropertyName () As DataTypeOfPrivateData
        Get
            Return PrivateData
        End Get

End Property
```

Example 1: Read-Only Property

- Declaring Property Procedures for a **Age** Property which allows retrieval only of a private integer variable ***strAge***:

```
Public Class clsPerson
        Private m_Age As Integer

        Public Property Age () As Integer
                Get
                        Return m_Age
                End Get
        End Property
End Class
```

==

- Now from the outside world you can make the following statements assuming you created an object named *objEmployee* of the Class *clsPerson* and somewhere in the class the m_Age variable was populated with data:

```
'Example of getting the object's property:
   Dim employeeAge As Integer

   employeeAge = objEmployee.Age

   MessageBox.Show ("The Employee Name is " &  employeeAge)

   'Another alternative:
   MessageBox.Show ("The Employee Name is " &  objEmployee.Age)

'Example of setting the object's Read-Only property:
   objEmployee.Age = 22 '#ERROR..THIS WILL GENERATE COMPILER ERROR - READ-ONLY!!
```

Write-Only Properties

- There are times when we may want a property to be Write-Only or the value can be written, but not read.
- In this case we simply remove the Get portion of a ***Property Procedures***.
- Continuing our step by step approach, the syntax is as follows:

Step 8: Create the Class **Property** Procedures

- Syntax for ***Public Write-Only Property*** Procedures:

```
Public Property PropertyName () As DataTypeOfPrivateData

        Set ( ByVal Value As DataType)
            PrivateData = Value
        End Set
End Property
```

Example 1: Write-Only Property

- Declaring Property Procedures for a **PinNumber** Property which allows setting of a private integer variable ***strPinNum***:

```
Public Class clsPerson
        Private m_PinNumber As String

        Public Property PinNumber () As String

                Set ( ByVal Value As String )
                        m_PinNumber  = Value
                End Set
        End Property
End Class
```

===

- Now from the outside world you can make the following statements assuming you created an object named *objEmployee* of the Class *clsPerson*:

```
'Example of Getting the object's property:
   Dim myPin As String

   objEmployee.PinNumber = "1122r"


'Example of getting the object's Read-Only property & generating error:
   myPin = objEmployee.PinNumber '#ERROR..GENERATE COMPILER ERROR - WRITE-ONLY!!
```

Creating Class Methods – Part V

- The Class Methods make the object do or perform some action or task. Methods are usually public and as with any forms, module etc, methods are composed of the following:

  - ***Public Sub Procedures()***
  - ***Public Functions Procedures()***
  - ***Public Event-Handlers()***

- Continuing our step by step approach, the syntax is as follows:

Step 9: Create Class **Methods**

- Use the syntax shown in previous notes for ***Public Sub Procedures and Function*** declarations.

Public Sub Procedure Methods with No Parameters:

- Syntax for **Sub Procedure** with no arguments:

```
 'Syntax for Sub Procedure with no arguments:
Public Sub ProcedureName ()

        'Body Code goes here!

End Sub
```

Example 1:

- Declaring procedure to print the content of the person class:

```
Public Class clsPerson
        Private m_Name As String
        Private m_IDNumber As Integer

        Public Sub PrintPerson ()
                'code to print the person object goes here!
        End Sub

End Class
```

==
==

- Assuming you created an object named *objEmployee* of the Class *clsPerson*, you can execute its *PrintPerson()* method so the object can perform the action:

```
'Example of calling the method:
   objEmployee.PrintPerson()
```

Example 2:

- Declaring a Sub Procedure to calculate the total charge for a customer. In this example it is assumed that the variables being processed are private data variables of the class:

```
Public Class clsInvoice
      Private m_Name As String
      Private m_Total As Decimal
      Private m_SubTotal As Decimal
      Const Private m_TAX As Decimal = 0.825
      Private m_objInvoiceItems As New clsInvoiceItems   'Class Object

      Public Sub CalculateTotal ()
            m_Total = m_SubTotal + (m_SubTotal * m_TAX)
      End Sub

End Class
```

==

- Assuming you created an object named *objInvoice* of the Class *clsInvoice*, you can execute its *CalculateTotal()* method so the object can perform the action:

```
'Example of calling the method:
   objInvoice.CalculateTotal()
```

'Example of calling the method, and getting the results of the calculation assuming the clsInvoice Class has a Property named *Total* that can return the value, *we can make the following statement:*

```
   Dim decTotalCharges As Decimal

   objInvoice.CalculateTotal()

   decTotalCharges = objInvoice.Total

   MessageBox.Show ("The total charges are " &  decTotalCharges)
```

Public Sub Procedure Methods with Parameters:

- Syntax for **Sub Procedure** with **arguments**:

```
'Syntax for Sub Procedure with argument list:
Public Sub ProcedureName (ByRef|ByVal variable As Type, ByRef|ByVal variable As Type….. )

        'Body Code goes here!

End Sub
```

Example 1:

- Declaring a Sub Procedure to calculate the total charge for a customer. In this example the values for decTotal and decTax are NOT part of the Class but are passed as argument to the method. The only internal private class data is the decSubTotal variable. Note the pass-by-reference & pass-by-value:

```
Public Class clsInvoice
        Private m_Name As String
        Private m_SubTotal As Decimal
        Private m_objInvoiceItems As New clsInvoiceItems   'Class Object

        Public Sub CalculateTotal (ByRef total As Decimal, ByVal m_Tax As Decimal)

                m_Total = m_SubTotal + (m_SubTotal * m_Tax)

        End Sub

End Class
```

==

- Assuming you created an object named *objInvoice* of the Class *clsInvoice*, you can execute its *CalculateTotal()* method so the object can perform the action:

```
'Example of calling the method:
   Dim totalCharges, salesTax As Decimal

   salesTax = 0.825

   objInvoice.CalculateTotal(totalCharges, salesTax)

'The decTotalCharges argument stores the total charge due to passed-by-reference
   MessageBox.Show ("The Customer Total Charge is " &  totalCharges)
```

Example 2:

- Declaring Sub Procedure that sets or overwrites the private data m_FirstName. These types of methods are called Setter methods since they set the private data. Using setter methods are an alternative to using Property procedures. In this course we will be using Property Procedures:

```
Public Class clsCustomer
      Private m_FirstName As String
      Private m_LastName As String
      Private m_CustomerID As Integer
      Private m_BirthDate As Date
      Private m_objCreditCard As clsCreditCard  'Class Object

      Public Sub SetFirstName (ByVal fName As String)

            m_FirstName = fName

      End Sub

End Class
```

==

- Assuming you created an object named *objBankCustomer* of the Class *clsCustomer*, you can execute its *SetFirstName()* method so the object can perform the action:

```
'Example of calling the method:
   objCustomer.SetFirstName("Joe")

'Example of calling the method:
   Dim fName As String

   fName = "Joe"

   objCustomer.SetFirstName(fName)
```

Public Function Methods with No Parameters:

- Syntax for **Function** with no arguments:

```
 'Syntax for Function with no arguments:
Public Function FunctionName () As ReturnType
          'Body Code goes here!

          Return ReturnValue

End Function
```

Example 1:

- Declaring Function that gets or returns the private data strFirstName. These types of methods are called Getter methods since they get the private data. Using Getter methods are an alternative to using Property procedures. In this course we will be using Property Procedures:

```
Public Class clsCustomer
          Private m_FirstName As String
          Private m_LastName As String
          Private m_IDNumber As Integer
          Private m_BirthDate As Date
          Private m_objCreditCard As New clsCreditCard   'Class Object

          Public Function GetFirstName () As String
                    Return m_FirstName
          End Function

End Class
```

===
=

- Assuming you created an object named *objBankCustomer* of the Class *clsCustomer*, you can execute its *GetFirstName()* method so the object can perform the action and return the *strFirstName* private data:

```
'Example of calling the method:
   Dim fName As String

   fName = objCustomer.GetFirstName()

   MessageBox.Show ("The Customer First Name is " &  fName)


'Example of calling the method:
   MessageBox.Show ("The Customer First Name is " &  objCustomer.GetFirstName())
```

Example 2:

- Declaring Function to calculate the total charge for a customer and returns the total charge to the calling program. The variables being processed are private data:

```
Public Class clsInvoice
      Private m_Name As String
      Private m_Total As Decimal
      Private m_SubTotal As Decimal
      Const Private m_TAX As Decimal = 0.825

      Public Function CalculateTotal () As Decimal
            m_Total = m_SubTotal + (m_SubTotal * m_TAX)
            Return  decTotal
      End Function

End Class
```

==

- Assuming you created an object named *objInvoice* of the Class *clsInvoice*, you can execute its *CalculateTotal()* function so the object can perform the action and return a value:

```
'Example of calling the method:
   Dim totalCharges As Decimal
   totalCharges = objInvoice.CalculateTotal()

   MessageBox.Show ("The total charges are " &  totalCharges)


'Example of calling the method:

   MessageBox.Show ("The total charges are " &  objInvoice.CalculateTotal())
```

Public Function Methods with Parameters:

- Syntax for **Function** with **arguments**:

```
 'Syntax for Function with argument list:
Public Function FunctionName (ByRef|ByVal variable As Type, ByRef|ByVal variable As Type… ) As ReturnType

      'Body Code goes here!
      Return ReturnValue

End Function
```

Example 1:

- Declaring a Sub Procedure to calculate the total charge for a customer. In this example the values for ***subTotal*** and ***Tax*** are NOT part of the Class but are passed as argument to the method. The only internal private class data is the ***m_SubTotal*** variable. Note the pass-by-value the default:

```
Public Class clsInvoice
      Private m_Name As String
      Private m_Total As Decimal

      Public Function CalculateTotal (ByVal subTotal As Decimal,  ByVal Tax As Deciaml) As Decimal
            m_Total = subTotal + (subTotal * Tax)
            Return m_Total
      End Function

End Class
```

==

- Assuming you created an object named *objInvoice* of the Class *clsInvoice*, you can execute its *CalculateTotal()* function so the object can perform the action and return a value:

```
'Example of calling the Function with Arguments:
   Dim totalCharges, subCharges, salesTax As Decimal

   salesTax = 0.825
   subCharges = 99.95

   totalCharges = objInvoice.CalculateTotal(subCharges, salesTax)

   MessageBox.Show ("The total charges are " &  totalCharges)


'Example of calling the Function with Arguments:

      MessageBox.Show ("The total charges are " & _
                       objInvoice.CalculateTotal(totalCharges, salesTax))
```

Example 2:

- Declaring a Function in the employee's class which authenticates an employee username & password. Employee Object contain a Function that performs the authentication by comparing its internal username/password to values passed as arguments:

```
Public Class clsEmployee
    Private m_FirstName As String
    Private m_LastName As String
    Private m_IDNumber As Integer
    Private m_BirthDate As Date
    Private m_UserName As String
    Private m_PassWord As String


    Public Function Authenticate (ByVal User As String, ByVal Pass As String) As Boolean

        If m_UserName = User And m_PassWord = Pass Then
            Return True
        Else
            Return False
        End If

    End Function

End Class
```

===

- Assuming you created an object named *objEmployee* of the Class *clsEmployee*, you can execute its *Authenticate()* function so the object can perform the action and return the results of the authentication:

```
'Example of calling the Function with Arguments:
   Dim UserName, PassWord As String
   Dim Access As Boolean

   Dim objLoginForm As frmLogin
   objLoginForm = New frmLogin()

    'Assuming we extract values from text boxes of a login form that is displayed
   objLoginForm.ShowDialog()
   strUserName = objLoginForm.txtUserName.Txt
   strPassWord = objLoginForm.txtPassWord.Txt

   Access = objEmployee.Authenticate(strUserName, strPassWord)

   If Access Then

      MessageBox.Show ("Access Granted")
   Else
      MessageBox.Show ("Access Denied")
   End If
```

Constructor Method (Important Topic) – Part VI

- The **constructor** method is a special method that is always invoked as an Object is created.
- What this means is that every time an object is created, this method is automatically executed, thus the name **Constructor**.
- This method will contain Initialization code or code that you want executed when the object is created. For example you may want to initialize data with entry strings, 0 values, etc.
- The Constructor Method has the following characteristics:

  - It is named Public Sub ***New***()
  - Automatically executes before any other methods are invoked in the class
  - Only runs once for an object
  - Doe not return a value
  - Can contain parameters
  - You can add any code that you wish to execute upon creation of the object

- Continuing our step by step approach, the syntax is as follows:

Step 10: Creating **the** Class **Constructor:**

Constructor Method with No Parameters(DEFAULT CONSTRUCTOR):

- Syntax for **Constructor Procedure** with no arguments:

```
 'Syntax for Constructor Method with no Parameters:
Public Sub New ()

          'Initialization Code goes here!

End Sub
```

Example 1:

- Declaring a Constructor to initialize the private data members of the person class:

```
Public Class clsPerson
        Private m_Name As String
        Private m_IDNumber As Integer
        Private m_BirthDate As Date

        Public Sub New ()
                m_Name = “”
                m_IDNumber =0
                m_BirthDate = #1/1/1900#
        End Sub

End Class
```

==

- In this example we created an object named *objEmployee* of the Class *clsPerson*, we assume this object contain properties for each of the private data above. The constructor will automatically execute upon creation of the object:

```
'Assuming Object is created as follows:
Dim objEmployee As clsPerson = New clsPerson()

'Viewing the content of an object after creation using properties:
   MessageBox.Show (objEmployee.Name) 'result is a blank

   MessageBox.Show (objEmployee.IDNumber)’results 0

   MessageBox.Show (objEmployee.BirthDate)’results 1/1/1900
```

Constructor Methods with Parameters:

- A constructor method can contain a list of parameters that can be passed to the object.
- This is done by simply adding a parameter list to the header.
- This parameter list can be used for what ever functionality the programmer wants to implement, for example the values can be used to initialize the private data members:
- Normally we want one parameter for each of the private data if we would like to be able to initialize all private data members.
- But again this is optional; we can create a parameter list just for those data variables we wish to initialize.
- Syntax for **Constructor Procedure** with **arguments**:

Syntax for Constructor Method with Parameters:

```
Public Sub New (ByVal variable As Type, ByVal variable As Type….. )

        'Body Code goes here!

End Sub
```

Example 2:

- Declaring a Constructor to initialize the private data members of the person class via arguments:

```
Public Class clsPerson
        Private m_Name As String
        Private m_IDNumber As Integer
        Private m_BirthDate As Date

        'Constructor assigns parameters to each private data member
        Public Sub New (ByVal Name As String, ByVal ID As Integer, ByVal BDate As Date)
                m_Name = Name
                m_IDNumber = ID
                m_BirthDate = BDate
        End Sub

End Class
```

==

- In this example we created an object named *objEmployee* of the Class *clsPerson*, we assume this object contain properties for each of the private data above. The constructor will automatically execute upon creation of the object and parameters are passed as argument during the creation:

```
'Assuming Object is created as follows:
Dim objEmployee As clsPerson = New clsPerson("Joe Smith", 111, #12/12/1965#)

'Viewing the content of an object after creation using properties:
MessageBox.Show (objEmployee.Name) 'result is Joe Smith

MessageBox.Show (objEmployee.IDNumber)'results 111

MessageBox.Show (objEmployee.BirthDate)'results 12/12/1965
```

Do we Assign Parameters Values to Private Data or Class Properties?

- In the previous example, we assigned the parameter variables directly to the private data as follows:

```
‘Constructor assigns parameters to each private data member
Public Sub New (ByVal strNn As String, ByVal intID As Integer, ByVal dBDate As Date)
        strName = strNn
        intIDNumber = intID
        dBirthDate = dBDate
End Sub
```

- This works just fine, but there is a better alternative that gives us more flexibility. That assigning the Constructor Parameters to the Class Public Properties instead of the private data as follows:

```
‘Constructor assigns parameters to Public Properties (We assume that Name, IDNumber & Birthdate are Property Procedures
Public Sub New (ByVal strNn As String, ByVal intID As Integer, ByVal dBDate As Date)
        Name = strNn
        IDNumber = intID
        BirthDate = dBDate
End Sub
```

- Doing this gives us the flexibility to add code inside the Property Procedure that would allow us to validate or check the values before assigning it to the private data. This process is known as Validation. More on this in future lecture

Example 3:

- Declaring a Constructor to initialize the private data members of the person class via arguments:

```
Public Class clsPerson
        Private m_Name As String
        Private m_IDNumber As Integer
        Private m_BirthDate As Date

        ‘Property Declarations
        Public Property Name () As String
                Get
                        Return m_Name
                End Get

                Set ( ByVal Value As String )
                        ‘Enter code here to validate or check value before it is assigned to private data
                        ‘Validation code here
                        ‘Now assign value to private data
                        m_Name = Value
                End Set
        End Property

        ‘Other Property Declarations here for IDNumber & BirthDate….

        ‘Constructor assigns parameters to each private data member
        Public Sub New (ByVal strNn As String, ByVal intID As Integer, ByVal dBDate As Date)
                Me.Name = strNn
                Me.IDNumber = intID
                Me.BirthDate = dBDate
        End Sub
```

- Note that in example 3, in the Constructor, I use the ME keyword to identify the properties. This is optional, it would work without the use of this keyword, but it makes reading and understanding the property call easier.

The Default Constructor:

- You don't have to create a constructor for your class, but if you don't, VB.NET will automatically create on for you in the background. This Constructor is called the **Default Constructor**.
- The Default Constructor is a no-parameter constructor that is automatically created for you if you don't create your own!
- This constructor does not do anything. It's syntax is:

```
'Syntax for Sub Procedure with no arguments:
Public Sub New ()
End Sub
```

- So keep in mind that if you create a class without a constructor, VB in the background will create this Default Constructor for its use. This will be totally transparent to you.

Using a Parameterized Constructor and the Default Constructor:

- **IMPORTANT!** If you create a Parameterized Constructor Method inside your class, then when you create Objects of the class, you must send data to each Object upon creation. **You CANNOT create the object WITHOUT giving it data to match each of the parameters!**
- So what happens if you want to create an Object with data as argument and regular objects? In this case you would need to add two Constructor Methods in the class, a **Default** Constructor or a standard No-Parameter Constructor and the Parameterized Constructor.
- The syntax the same as before, just add both constructors to the class, the default, No-Parameter and the Parameterized:

```
'Syntax for Constructor Method with no Parameters:
Public Sub New ()

        'Initialization Code goes here! If NO Code is added it is then a Default Constructor

End Sub
```

```
'Syntax for Sub Procedure with argument list:
Public Sub New (ByVal variable As Type, ByVal variable As Type….. )

        'Body Code goes here!

End Sub
```

- **GOING FORWARD, ALL CLASSES IN MY HWS, PROJECTS AND EXAMS SHOULD INCLUDE BOTH THE DEFAULT AND PARAMETERIZED CONSTRUCTORS!**

Example 4:

- Declaring a Constructor to initialize the private data members of the person class via arguments:

```
Public Class clsPerson
        Private m_Name As String
        Private m_IDNumber As Integer
        Private m_BirthDate As Date

        Public Sub New ()
                m_Name = “”
                m_IDNumber = 0
                m_BirthDate = #1/1/1900#
        End Sub


        Public Sub New (ByVal strNn As String, ByVal intID As Integer, ByVal dBDate As Date)
                Me.Name = strNn
                Me.IDNumber = intID
                Me.BirthDate = dBDate
        End Sub

End Class
```

==

- In this example we created an object named *objEmployee* of the Class *clsPerson*, we assume this object contain properties for each of the private data above. The constructor will automatically execute upon creation of the object and parameters are passed as argument during the creation:

```
'Creating Objects with no Parameters and with Parameters:
Dim objEmployee1 As clsPerson = New clsPerson()
Dim objEmployee2 As clsPerson = New clsPerson(“Joe Smith”, 111, #12/12/1965#)

'Viewing the content of the No-Paremeter object after creation using properties:
   MessageBox.Show (objEmployee1.Name)  'result is a blank

   MessageBox.Show (objEmployee1.IDNumber)’results 0

   MessageBox.Show (objEmployee1.BirthDate)’results 1/1/1900



'Viewing the content of the Parameterized object after creation using properties:
MessageBox.Show (objEmployee2.Name)  'results in a Joe Smith

MessageBox.Show (objEmployee2.IDNumber)’results 111

MessageBox.Show (objEmployee2.BirthDate)’results 12/12/1965
```

Raising or Triggering the Events of the Class – Part VI

- At this point we can raise or trigger the events declared prior anywhere we want in the class:
- We will cover this in future lecture

Step 8: Raise or Trigger **Events** where desired

- Covered in future lecture.

Summary of Components for Creating a Class Module

- To summarize, the steps required to build a class are:

1) Create the ***Class***.
2) Declare the ***Private Variables***, which hold the ***Data***.
3) Declare the **Events**
4) Create a **Property** procedures for each variable declared.
5) Create the ***Methods*** or Procedures/Functions needed.
6) Create the ***Constructor Methods***: Default and Parameterized
7) ***Raise*** any required Events anywhere in the class module where desired (More on this later)

ATTENTION!

- ***DISPLAYING FORMS OR MANIPULATING FORMS OR CALLING FORM CONTROLS*** **FROM WITHIN A CLASS MODULE IS NOT GOOD OOP PRACTICE!**

- ***THIS INCLUDES MESSAGEBOXES AS WELL. WE MEAN ANY USER-INTERFACE CODE!!!!!***

- **SOME OF THE EXAMPLES IN THE NOTES AND CLASS MAY DISPLAY FORMS OR MESSAGE BOXES FROM WITHIN THE CLASS METHODS (PROCEDURE & FUNCTIONS). THIS WILL BE DONE ONLY FOR TEACHING OR TESTING PURPOSE!**

- **UNDER NO CIRCUNSTANCE SHOULD YOU DISPLAY ANY FORMS OR MESSAGEBOXES FROM WITHIN A CLASS IN EITHER YOU HOMEWORK OR PROJECTS!**

- **UNLESS OTHERWISE INSTRUCTED!**

5.2.2 Part II - Bringing a Class to Life by Creating Objects

- So far we have create the Class or the template which will server as the mold for creating objects of the class.
- Keep in mind that creating a Class Module is actually creating a template. But there is more to it than that.
- When you create a Class Module, you are actually introducing a new ***data type*** into the program, so in addition to the built-in data types such as string, integer, date etc. You will now have a new type based on the Class you create.
- In order to use a class we need to create an ***Object*** or an instance of the class.

Object Statement Declaration

- There are methods to create objects:

Method I (Simplest & Compact Method) – One statement POINTER & OBJECT Method:

- The first method is done in one step, where you declare the POINTER and create the OBJECT in one step using the keyword ***New***:

Accessibility *ObjectName* **As New** *ClassName***()**

Where Accessibility:
-*Dim*
-*Public*
-*Private*
-*Protected*
-*Friend*

Examples Using Method I:

- Assuming we have previously defined a Class named *clsCustomer*. Creating a Customer object of the class *clsCustomer* is as follows:

  Public objCustomer **As New** clsCustomer()

- Assuming we have previously defined a Class named *clsInvoice*. Creating Invoice objects is as follows:

  Dim objInvoice **As New** clsInvoice()

- Assuming we have previously defined a Class named clsEmployee and that this class contains a **Constructor Method** that initializes the class with data passed as arguments. Creating an Employee object with data is as follows:

  Private objEmployee **As New** clsEmployee("Joe", 111, #12/12/1965#, "225 Flatbush Ave")

Method II – One Statement (Two Parts) POINTER & OBJECT Creation:

- The second method is done in one statement (one line of code) but involves two steps. You declare the reference variable and assign it to the class object using the keyword ***New***:

Accessibility *ObjectName* **As** *ClassName* = **New** *ClassName***()**

Where Accessiblity:
-*Public*
-*Private*
-*Protected*
-*Friend*

Examples Using Method II:

- Assuming we have previously defined a Class named *clsCustomer*:

  Public objCustomer **As** clsCustomer **= New** clsCustomer()

- Assuming we have previously defined a Class named *clsInvoice*:

  Private objInvoicer **As** clsInvoice **= New** clsInvoice()

- Assuming we have previously defined a Class named clsEmployee and that this class contains a **Constructor Method** that initializes the class with data passed as arguments. Creating an Employee object with data is as follows:

  Dim objEmployee **As** clsEmployee **= New** clsEmployee("Joe", 111, #12/12/1965#, "225 Flatbush Ave")

Method III (Most Flexible) – Two Statement Method: POINTER created first. OBJECT created inside a method (Flexible/Preferred for Long-Lived Object):

- The third method requires a two statement process. This method is more flexible:
    1. Declare the object variable in the declaration part of a module, form etc., in the program
    2. Inside a Method (*Procedure/Function*) create the object by assigning the variable in step 1 to the class using the keyword ***New***

'Declare POINTER to Objects of the Class type:

Accessibility *ObjectName* **As** *ClassName*

'Inside a Method Procedure CREATE THE ACTUAL OBJECT as follows(NOTE THAT THIS MUST BE DONE INSIDE A METHOD:

ObjectName **= New** *ClassName***()**

Where Accessibility:
- *Dim*
- *Public*
- *Private*
- *Protected*
- *Friend*

Examples Using Method III:

- Example 1, assuming we have previously defined a Class named *clsCustomer* and this class contains a ***default Constructor*** and a ***parameterized Constructor*** that takes arguments to initialize the private data. The code to create objects of the clsCustomer Class inside the Method *Sub Main()* is as follows:

'In this example both the POINTER portion and OBJECT creation code are declared INSIDE A METHOD

```
Sub Main()

    Dim objCustomer1 As clsCustomer
    clsCustomer1  = New clsCustomer()

    Dim objCustomer2 As clsCustomer
    clsCustomer2  = New clsCustomer("Mary Jones", 444, #01/23/1972#)

End Sub
```

- Example 2, assuming we have previously defined a Class named *clsCustomer*. Now we want to create a Customer object inside a Form, we would like the object to actually be created when the Form is loaded. The code is as follows inside the Form:

In this example POINTER code is declared OUTSIDE of the method, and OBJECT creation code is done INSIDE A METHOD

```
'Declaration section of the Form
Option Explicit
Dim objCustomer As clsCustomer


'Object is actually created inside a Method or Event-handler etc.
Sub Form1_Load()
      objCustomer = New clsCustomer()
End Sub
```

- Using method III, we have the flexibility to create multiple objects using one declaration statement.
- This is done by having all the objects declaration as part of one statement, and having individual statements for each object creation.
- Keep in mind that this works as long as the objects are of the same class.
- Lets look at the following example:

Examples Creating Multiple Instances of Objects using Method III:

- Assuming we have previously defined a Class named *clsCustomer*. Then we wan to create a Customer object inside the Method Sub Main(), the code is as follows:

```
'Declaration of Customer Objects in declaration portion
Public objCustomer1, objCustomer2, objCustomer3 As clsCustomer
```

Sub Main()

```
'Creation or definition of Customer Objects inside a method
objCustomer1= New clsCustomer()
objCustomer2= New clsCustomer()
objCustomer3= New clsCustomer()
```

End Sub

5.2.3 Part III - Using the Objects

- ❑ Once the objects are created we can now use them to implement the program.

SETTING or assigning Data to the Object's Properties

- ❑ Use the ***Dot Operator*** to manipulate and assign values to the object.
- ❑ Syntax for assigning data to object via the **Property**:

Object. *Property = value*

Example:

- ❑ Assuming we have previously created a Customer Object:

```
objCustomer.FirstName = "Joe"
objCustomer.LastName = "Smith"
```

Assigning the Object with data that is also an Object (Object-to-Object Interaction)

- ❑ If the data being assigned to the Object is also an Object, then the assignment is done via the Object's ***Property*** as well.
- ❑ The syntax is the same as the regular data.
- ❑ Note that like any other variable, the Object being assigned must be of the same type as the object defined in the Property.
- ❑ Syntax:

Object. *Property = objObject*

Example:

- ❑ Assuming we have previously created the objects listed:

```
objCustomer.CreditCard = objCreditCard
objCustomer.Invoice = objInvoice
```

GETTING or accessing Data from the Object's Properties

- You retrieve values from the class Data via the Object's ***Property*** and the ***Dot Operator***
- Syntax for accessing data from the object via the **Property**:

Value = **Object.** *Property*

Example:

- Assuming we have previously created a Customer Object:

```
Dim strFName, strLName As String

strFName = objCustomer.FirstName
strLName = objCustomer.LastName
```

Accessing Object's data that is also an Object (Object-to-Object Interaction)

- If the data being retrieved from the Object is also an Object, then the assignment is done via the Object's ***Property***.
- The syntax is the same as the regular data.
- Note that in order to access an object data member, you need an object to receive it and the receiving object must be of the same type as the property.
- Syntax:

objObject = **Object.** *Property*

Example:

- Assuming we have previously created the objects listed:

```
objCreditCard = objCustomer.CreditCard
objInvoice  = objCustomer.Invoice
```

Calling the Object's Methods

- Syntax for calling an object **Methods** uses the dot operator as well:

Object. *Method()*

Examples:

Calling SUB Procedures with NO PARAMTERS

- Assuming we have previously created a Customer Object, we call a **Sub Procedure** from the Object:

```
objCustomer.PurchaseProduct()
```

- Assuming we have previously created an Invoice Object, we call a **Sub Procedure** from the Object:

```
objInvoice.Print()
```

Calling SUB Procedures WITH PARAMTERS

- Assuming we have previously created an login Form Object, we call a **Sub Procedure** which takes **arguments**:

```
objLoginForm.DisplayFormGetUserInfo(userName, passWord)
```

- Assuming we have previously created an Invoice Object, we call a **Sub Procedure** which takes **arguments**:

```
objInvoice.CalculateTotal(totalCharges, salesTax)
```

Calling FUNCTION Procedures with NO PARAMTERS

- Assuming we have previously created an Invoice Object, we call a **Function** to execute and return a value:

```
totalCharges = objInvoice.CalculateTotal()
```

Calling FUNCTION Procedures WITH PARAMTERS

- Assuming we have previously created an Invoice Object, we call a **Function** which takes **arguments** to execute and return a value:

```
access = objEmployee.Authenticate(userName, passWord)
```

Object to Object Interaction – Assigning One Object's Reference to Another Object

- ❑ You can assign one object to another Object. Keep in mind that **both objects must be of the same class**.
- ❑ But this statement can be misleading. You may think that a COPY of one Object is being made to another, this is NOT THE CASE!
- ❑ When you assign one object to another you are actually assigning REFERENCES or POINTERS!!!!!!
- ❑ The content of one object IS NOT COPIED to another but simply ONE POINTER POINTS TO THE OTHER!
- ❑ Therefore assigning the content of one object to another is simply having a pointer point to the same object that the other is pointing to.
- ❑ Here are some characteristics:
- ▪ Assigning an object to another with the = sign, means point to where the object is pointing to
- ▪ As the pointer points to this new object, the original object where the pointer was pointing to, is destroyed if there are no other references or pointers to it.
- ❑ The Syntax is the as follows:

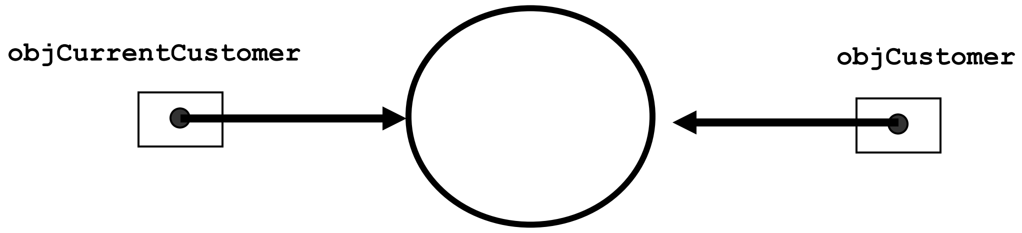
ObjObject1 = **objObject2**

Example 1:

- ❑ Assuming we have previously created the objects shown below:

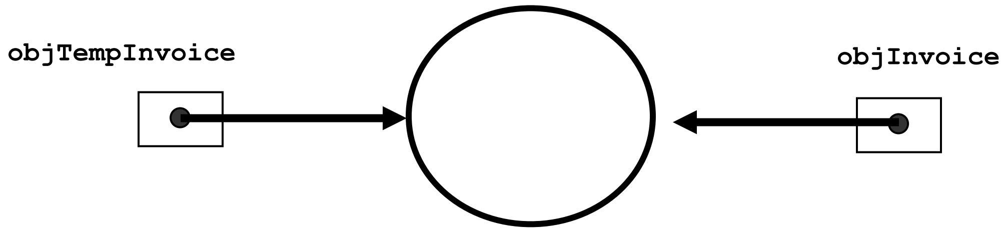
```
objCurrentCustomer = objCustomer
```

- ❑ The resultant interaction looks as follows:



- ❑ Another example:

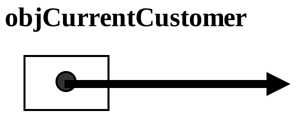
```
objTempInvoice = objInvoice
```



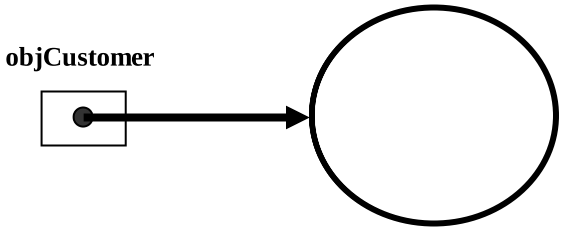
Example 2:

- Assuming we have the following declarations:

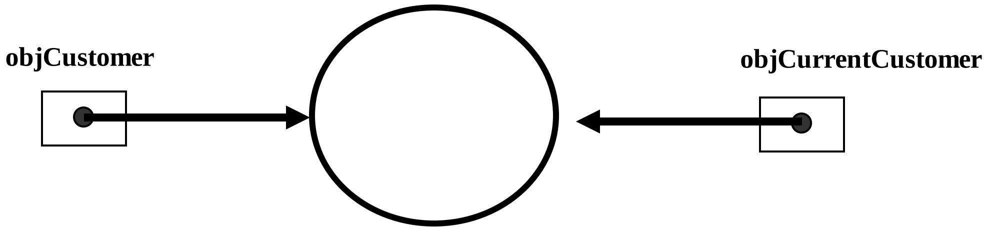
```
'Declare reference or POINTER variable,
but do not create object.
Dim objCurrentCustomer As clsCustomer
```



```
'Declare and Create an Object
Dim objCustomer As clsCustomer
objCustomer = New clsCustomer()
```



```
'Assign the references
objCurrentCustomer = objCustomer
```



5.3.2 Summary of Implementing Object-Oriented Programs

- An Object-Oriented Program(OOP) is a program that is written based on the Objects that are the protagonist of the program. The primary focus is not what the program does or functionality but of who are the members or participant of the program
- In other words, the **Class Modules are created** first, then the ***Objects of the Classes*** , finally the program is written to **use** the objects created
- In summary, the three steps required to creating an Object-Oriented Programs are shown below:

I. **Create the class specification or Class Module**
   - Private Data/Properties/Methods/Events

II. **Create Object of the Class**
   - Available Syntax:
     - One Statement Compact Method:
       - Dim TheObjectPointer As New Class
     - One Statement Two parts:
       - Dim TheObjectPointer As Class = New Class
     - Two statements:
       - 1) Declaration section or inside a method:
         - Dim TheObjectPointer As Class
       - 2) Inside a method
         - TheObjectPointer = New Class

III. **Use the Object of the Class**
   - Write the program to manipulate, access or modify the objects data & Call the Methods:
     - Set Properties
     - Get Properties
     - Call Methods
     - Trigger Events
     - Program Event-Handlers generated by Object
     - Object-to-Object interactions:
       - Object1 = Object2
       - Object1.Property = Object2
       - Object1 = Object2.Property
       - Object1.Method(Object2)
       - Object2 = Object1.Method()
       - Etc.

5.3 Additional Concepts in OOP

5.3.1 Introduction to Object-Oriented Related Statements

- Lets look at some additional object related statemet:

With Block Statement

- There are times when several code statements are being applied to the same Object.
- Visual Basics provides a statement that allows us to group of block a set of code that pertains to an Object.
- This statement is called the **With/End With** Statement. This statement works in conjunction with the Dot Operator as well.
- The syntax is as follows:

```
'With Block Statement
With ObjectName
        .Property or Method
       .Property or Method
      .etc…

End With
```

- All statements residing in the body of the With Statement before the End With relate to the Object named on the With header.

Example:

- **Using the With/End With Statement:**

  - Example 1 – *Assigning value to button*:

```
With btnExit
    .Text = "E&xit"
    .Enable = True
    .TabStop = True
    .TabIndex = 1
End With
```

  - Example 2 – *Assigning value to a Customer Object and execute methods*:

```
With Customer
    .FirstName = "Joe"
    .LastName = "Smith"
    .IDNumber = 111
    .BirthDate = 12/12/65
    .RentVideo()    'Calling Method of Object
End With
```

  - Example 3 – *Retrieving values from Customer Object*:

```
With Customer
    t = .FirstName
    Value2 = .LastName"
    .IDNumber
    .BirthDate
    .RentVideo()    'Calling Method of Object
End With
```

5.3.2 Nothing Keyword

- VB.NET offers a statement that allows you to have object pointers point to a NULL or a NO-VALUE setting
- The syntax is as follows:

Object = Nothing

- The Nothing Keyword, dissociate an OBJECT POINTER from pointing to any object.
- By default when you create a POINTER, it is pointing to Nothing.

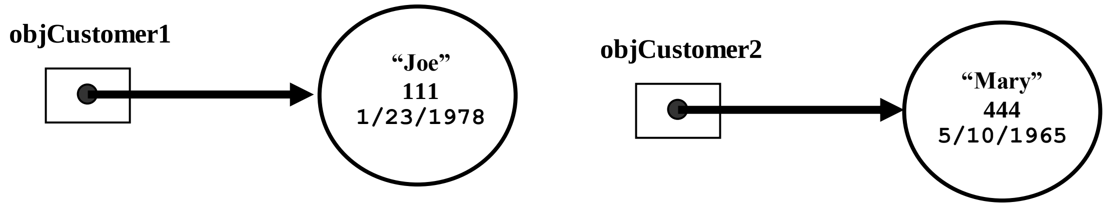
5.3.3 Termination and Cleanup (Destroying Objects)

- In VB.NET Objects are created and used. Often we use an object and no longer needed or reference to an object is removed.
- Here is a previous simple example of objects that are no longer needed or have not reference:

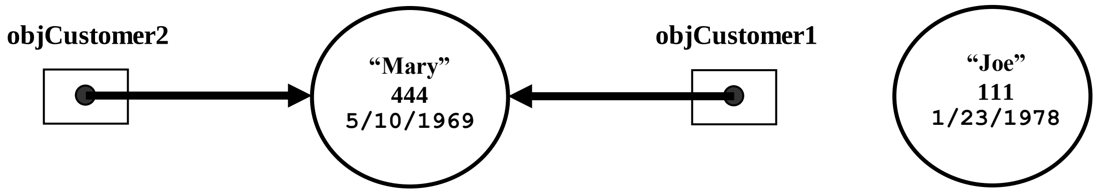
Example 1:

- Assuming we have the following declarations:

```
'Declare reference or POINTER variable,
but do not create object.
Dim objCustomer1 As New clsCustomer("Joe", 111, #1/23/1978#)
Dim objCustomer2 As New clsCustomer("Mary", 444, #05/10/1965#)
```



```
'Assign the references
objCustomer1 = objCustomer2
```



```
'Note that the Joe object no longer has reference
```

- Termination or Cleanup means destroying an object.
- Destroying an Object means that memory and resources being consumed by the Object are reclaimed and given back to the Operating System (OS) when there are ***NO References*** to the object (No pointer pointing to it).

Garbage Collection

- VB.NET provides a mechanism call garbage collection that automatically terminates and reclaims unused Objects.
- We will not go into the details in this lecture, but the Garbage Collection mechanism is automatic and when it sees an object without a reference (pointer) it will reclaim it.
- Garbage Collection runs automatically and the run time is determined by VB. We don't have control of the time when the garbage collection runs.
- However there are options available to the programmer for giving instructions to the Garbage Collector.
- At this time, I will not cover this material in my notes. Student is welcome to seek other sources of reference on this topic.

5.3.4 Testing Objects Prior to using them in a Program (Optional Step)

Testing Overview

- The main idea behind Object-Oriented Programs is to create the Objects which are the protagonist of the program first, and then you would write the program to manipulate these Objects.
- But in order to produce robust or error-free program, it's a good idea to first test the object, in a test program, prior to using it in a complete program.
- This is done by writing a small test or driver program to test the Object's basic functionality (Properties/Methods). Once the functionality of the class is tested, then we use the class in the real or intended program.

❖ Note that writing this test program is not mandatory. Sometimes programmers get lazy and choose to simply write the full program they intended to write in the first place, but think of the implication. You write a class, and then use it in a full blow program and you start having problems. You don't know if your problems are due to the objects you created or the program you are writing to use the objects. Eventually you start suspecting the Object and have to test it separately anyway. On the other hand, if you had tested your objects with a test program prior to using them, then you know that any errors you encounter in your program you would know that they are not due to the Object since the Object was tested prior to use.

Unit Testing of Objects (Driver Program)

- After the Class is created, it's a good idea to test the class by writing a small test driver program. This program will consist of the following components:

▪ Class Module – The Class Module templates you created for your program
▪ Standard Module – A Programming Code Window to enter code to test all possible functionality of Objects
▪ The program can be written using a ***Console Application*** or a ***Module-Driven Windows Application or Form-Driven***. Choice depends on the class you will test. A console application or a Module-Driven Windows application is the simplest. Since we want to just test, it makes no sense to create forms to test the class.

- The steps are as follows:

Step 1: Start a new Console Application or Module-Driven Windows Application.

Step 2: Add a Class & Class File to the project. Code in the Property Let, Property Get, Property Set, Methods and Events

```
Option Explicit

Public Class ClassName
 ' Private Data declarations.....

' Public Property  declarations...

' Public Methods declarations...

End Class
```

Step 3: In the Main Module, Create an Object

- Create an Object for testing using the methods shown previously. Create objects that use each Constructor (DEFAULT & PARAMETERIZED)

```
Dim Object1 As New Class()
Dim Object2 As New Class(value1, value2, value3 etc.,)
```

Step 4: Test and Validate the Constructors

- At this point, is a good idea to test and verify if the CONSTRUCTORS (DEFAULT & PARAMETERIZED) are correctly working.
- This is done by simply displaying the content of the properties of each object and verify if the constructor settings are valid
- This can be done in two ways:

1) GET the properties of each object and display them, if results shown match the constructor setting, test is OK, otherwise the constructors are not working and you need to debug and fix the problem. Syntax example:

```
'Testing Default Object Constructor
Console.WriteLine(Object1.FirstName)
Console.WriteLine(Object1.IDNum)
Console.WriteLine(Object1.BirthDate)


'Testing Parameterized Object Constructor
Console.WriteLine(Object2.FirstName)
Console.WriteLine(Object2.IDNum)
Console.WriteLine(Object2.BirthDate
```

2) If the object has an internal method that displays the data such as a Print() or anything else, call that method.

```
Object1.Print()
Object2.Print()
```

Step 5: In the Module Code Window, Enter Code to test every Property of the Object

- The idea now is to create an Object and test every Property of the Object.
- Is a good idea to test the SET part first, then GET, if you GET what you SET then you are OK, otherwise something is wrong and you need to test each individually.

Testing Setting each Property

- To test the Set Property is simple, just assign each property a value
- For example if the object has the following properties String property: *FirstName*, Integer property *IDNum* & Date property *BirthDate*, then simply assign a hard coded value as follows:

```
Object1.FirstName = "Joe"
Object1.IDNum = 111
Object1.BirthDate = #12/12/65#
```

- At this point, we don't know if the SET is ok, we need to view the data of the object via the properties and verify if these values are stored. One way to do it is to test the GET at this point.

Testing Get Part of a Property

- To test the Get Portion, just access the value that each property returns.
- This requires additional steps depending on the data being returned by the properties.
- There are two methods to doing this:

Method I – Create variables to store the values returned by the property Get

1) Create variables to store the values returned by every property, make sure the data types are the same as the values returned by the property.

    - For example if the object has the following properties: String property *FirstName*, Integer property *IDNum* & Date property *BirthDate*, to test simply assign the values of these properties to variables of the same type as follows:

```
Dim sVariable As String
Dim nVariable As Integer
Dim dVariable As Date

sVariable = Object1.FirstName
nVariable = Object1.IDNum
dVariable = Object1.BirthDate
```

2) Display the content of each variable using a *a Console.WriteLine* Statement in order to see if the values being returned are valid.

    - Now display the content of the variables using a Console.WriteLine:

```
Console.WriteLine( sVariable)
Console.WriteLine(nVariable)
Console.WriteLine(dVariable)
```

    - If the values you see are the values you assign with the Set Properties, then the Properties procedures are good, since you were able to display values that were assigned & retrieved, otherwise there is a problem which needs fixing.

Method II – (Simplest) Display the values directly from the Get Properties unto a Message Box

1) This method requires no variables, simply display the content of each Property directly from the Object as follows:

```
Console.WriteLine(Object1.FirstName)
Console.WriteLine(Object1.IDNum)
Console.WriteLine(Object1.BirthDate)
```

    - Again if the values displays are identical to those assigned previously, then the Object is good, otherwise fix the errors until the values display = values added.

Step 6: In the Module Code Window, Enter Code to test every Method of the Object

Testing Methods

- Note that testing the Properties is easy, all you need to do is assign & retrieve values into the Object, but executing the methods may prove a challenging depending on what action the method is to perform.
- For example, if a method is called *DisplayName* and calling it displays the FirstName data member, there is no problem testing since the result is a message box or Console.WriteLine statement displaying the name, but if the method is called SendEmail and calling it sends out an email that involves many systems and other complex programming code, then you really cannot test this Method at this point since it's dependent on many other things.

- To test methods simply call the method with the required arguments. For example:

```
objObject.Method1
objObject.Method2  arg1
objObject.Method3 arg1, arg2 ….
```

Step 6: Test any Events Triggered by the Object

- At this point you will test any ***Events*** & ***Event-Handlers*** that are triggered by the Object.
- This is done as follows:

1) Generate the event-handler outside of the object
2) Program the Event-Handler, a simple test is to display a message box from the event-handler
3) Run the program and execute the ACTION AGAINST THE OBJECT that will trigger the event
4) If the message box or code that was entered in the event-handler executes than event mechanism is working.

The Complete Test Program

- In the Standard Module screen enter the following code:

Step 1: Start a new Console Application or Module-Driven Windows Application.

Step 2: Add a Class & Class File to the project. Code in the Property Let, Property Get, Property Set, Methods and Events

Step 3-6: Add code to Create Objects, Test Property, Methods and Events

- In this example I will use a Module-Driven Windows Application. You can also use a console application simply replace the MessageBox.Show with Console.Writeline()

```
Option Explicit ON
Option Strict ON

'Step 3: Create the Object for testing:
Object1 As New Class()
Object2 As New Class(value1, value2, value3, etc.,)

'Step 4: Test the constructors:
'Method 1: Display content of objects via GETTING PROPERTIES
Console.WriteLine(Object1.Property1 & Object1.Property2 & Object1.Property3)

Console.WriteLine(Object2.Property1 & Object2.Property2 & Object2.Property3)

'Method 2: Call internal method if available to display the data
Object1.Print()
Object2.Print()

'Step 5A: Test each Property SET by assigning appropriate values:
Object1.Property1 = value
Object1.Property2 = value
Object1.Property3 = value
….
'Step 5B Option 1 - Test each Property GET
'Method 1(a) - Create variables of correct type to store the content of each Property Get:
Dim variable1 As Type
Dim variable2 As Type
Dim variable3 As Type

'Method 1(b) – Get each  using variables:
variable1 = Object1.Property1
variable2 = Object1.Property2
variable3 = Object1.Property3

'Method 1(c): Dislay the content of each variable to verify that values were stored:
MessageBox.Show (variable1 & variable2 & variable3)

'Step 5B: Option 2 – Alternate option to test each Property GET
'Method 1I - Alternate simpler option by displaying each object property directly:
MessageBox.Show (Object1.Property1 & Object1.Property2 & Object1.Property3)

'Step 6: Test each Method (If Possible) the object should perform the action dictated by the methods:
objObject.Method1
objObject.Method2 arg1
objObject.Method3 arg1, arg2 …
….
```

5.3.5 Sample Program 1 – Class Test Program

Creating and Testing a Person Class Using Console Application

Problem statement:

- ❑ Using a Console Application, create a Person Class. This class can be used to create objects with information of a Person such as name, id, and birth date, address & phone number. Also the class should contain a method that will print the object's information. Using the Console Application, write a simple driver program to test the class. This test program will be used to create objects using default & Parameterized Constructors as well as testing each Property and Methods.

Class Requirements

- ❑ The class contains the following data, properties & methods members

  Class Person Member Data:

▪ Name:	Type String
▪ IDNumber:	Type Integer.
▪ BirthDate:	Date
▪ Address:	Type String
▪ Phone:	Type String

  Class Member Properties & Methods:

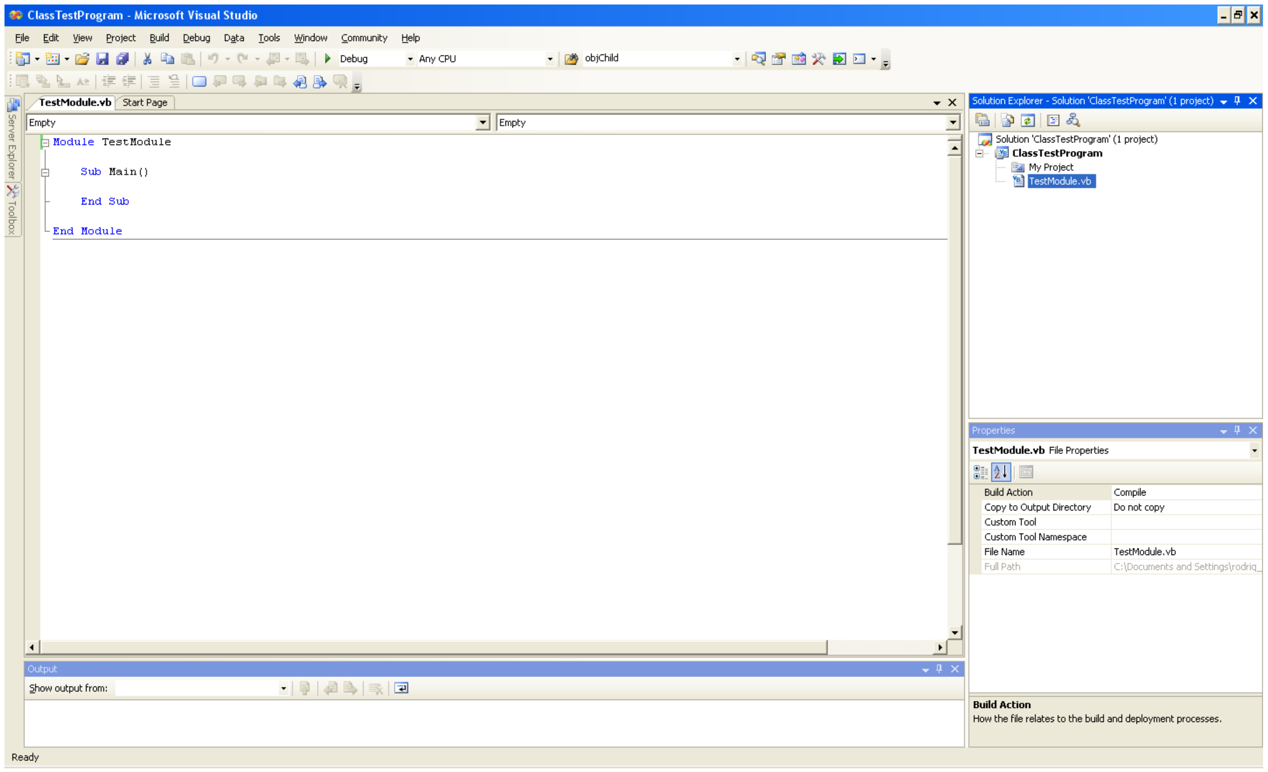
  - Let & Get Properties for each data member.
  - The Method Print(), which displays the Persons data

HOW IT'S DONE:

Part I – Create The Class:

Step 1: Start a new Console Application project and set the Module's properties as shown in table below:

Object	Property	Value
Module.vb	FileName Object Name	**modPerson.vb** **Module modPerson**


ClassTestProgram - Microsoft Visual Studio
File Edit View Project Build Debug Data Tools Window Community Help
Debug
Any CPU
objChild
TestModule.vb
Start Page
Empty
Empty
Server Explorer
Toolbox
Solution Explorer - Solution 'ClassTestProgram' (1 project)
Solution 'ClassTestProgram' (1 project)
ClassTestProgram
My Project
TestModule.vb
Properties
TestModule.vb File Properties
Output
Show output from:
Ready

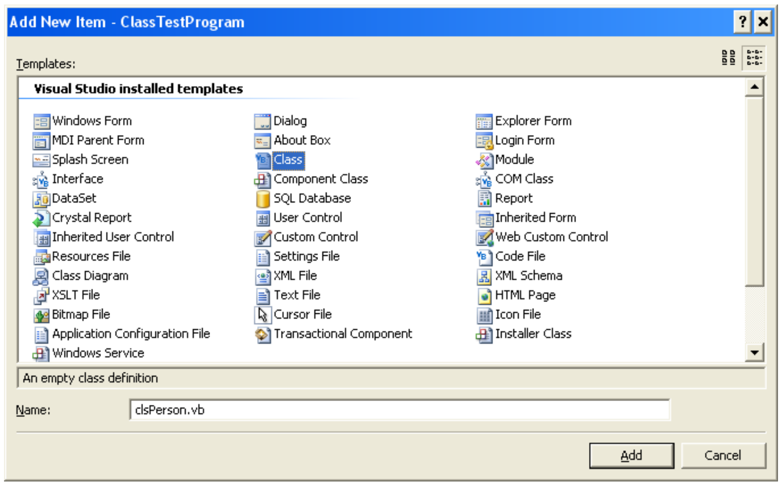
```
Module TestModule

    Sub Main()

    End Sub

End Module
```

Build Action	Compile
Copy to Output Directory	Do not copy
Custom Tool	
Custom Tool Namespace	
File Name	TestModule.vb
Full Path	C:\Documents and Settings\rodriq_

Build Action
How the file relates to the build and deployment processes.

Step 2: Add a Class to the Project and set the Class properties as shown in table below:

Object	Property	Value
Class Module 1	File Name	**clsPerson.vb**



An empty class definition

Name: clsPerson.vb

Step 3: In the Class Module code window enter the code for the private data:

```
Option Explicit On
Option Strict On


Public Class clsPerson
    '*******************************************************************
    'Class Data or Variable declarations
    Private m_Name As String
    Private m_IDNumber As Integer
    Private m_BirthDate As Date
    Private m_Address As String
    Private m_Phone As String
```

Step 4: In the Class Module code window enter the code for public Properties:

```
'**********************************************************************
'Property Procedures
Public Property Name() As String
    Get
        Return m_Name
    End Get
    Set(ByVal Value As String)
        m_Name = Value
    End Set
End Property

Public Property IDNumber() As Integer
    Get
        Return m_IDNumber
    End Get
    Set(ByVal Value As Integer)
        m_IDNumber = Value
    End Set
End Property

Public Property BirthDate() As Date
    Get
        Return m_BirthDate
    End Get
    Set(ByVal Value As Date)
        m_BirthDate = Value
    End Set
End Property

Public Property Address() As String
    Get
        Return m_Address
    End Get
    Set(ByVal Value As String)
        m_Address = Value
    End Set
End Property

Public Property Phone() As String
    Get
        Return m_Phone
    End Get
    Set(ByVal Value As String)
        m_Phone = Value
    End Set
End Property
```

Step 5: In the Class Module code window enter the code for Constructor Methods (Non-Parameter and/or Parameterized):

```
'**********************************************************************
'Class Constructor Methods

'Default Constructor
Public Sub New()
    'Note that private data members are being initialized
    m_Name = ""
    m_IDNumber = 0
    m_BirthDate = #1/1/1900#
    m_Address = ""
    m_Phone = "(000)-000-0000"
End Sub

'Parameterized Constructor
Public Sub New(ByVal Name As String, ByVal IDNum As Integer, ByVal BDate As Date, _
ByVal Address As String, ByVal Phone As String)
    'Note that as example we are NOT using the private data but
    'the Property Procedures instead when setting the data via the constructor

    Me.Name = Name
    Me.IDNumber = IDNum
    Me.BirthDate = BDate
    Me.Address = Address
    Me.Phone = Phone

End Sub
```

Step 6: In the Class Module code window enter the code for the PrintPerson Method:

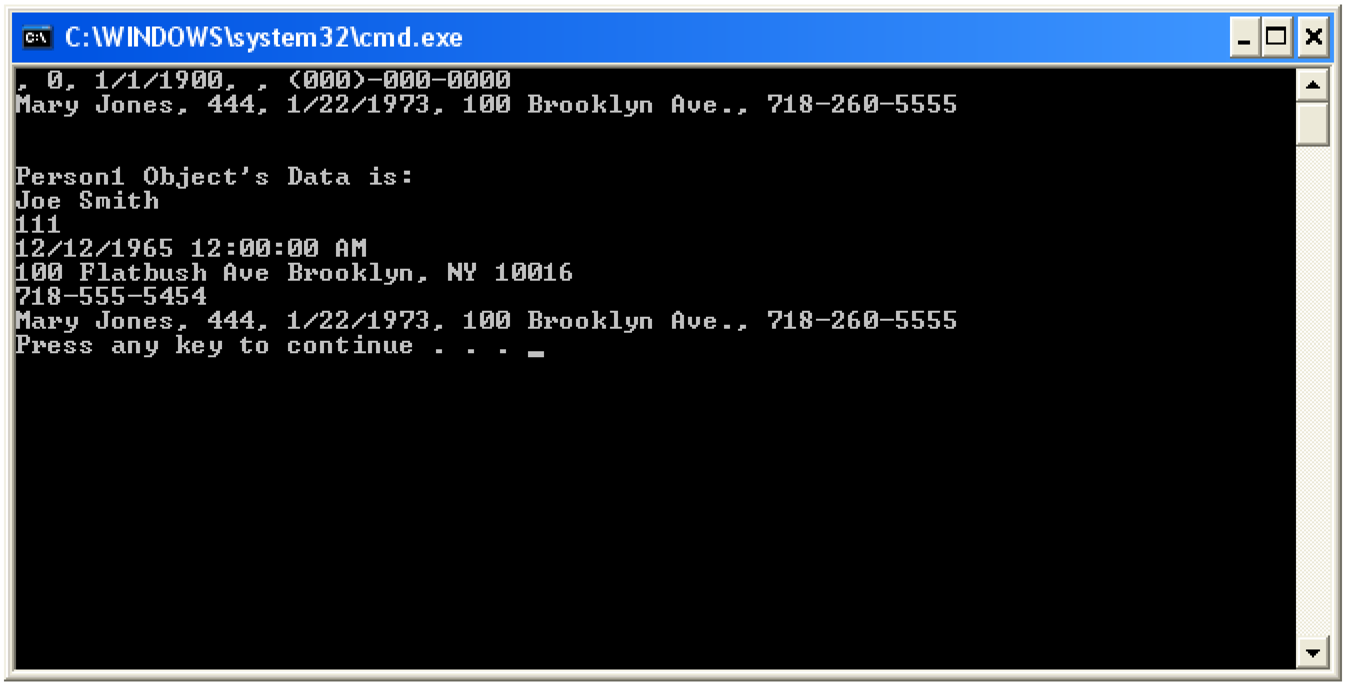
```
    '**********************************************************************
    '**********************************************************************
    'Class Methods
    '**********************************************************************

    'Author of base class allows sub classes to overide Print()
    'If they want to, it is not mandatory
    Public Overridable Sub Print()

        'Display object Content
        Console.WriteLine(m_Name & ", " & m_IDNumber & ", " & _
        m_BirthDate & ", " & m_Address & ", " & m_Phone)

    End Sub



End Class
```

Part II & III – Create The Object and Use it:

Step 7: In the Module File Sub Main() Procedure enter code to Create and Use the Object, this is know as a driver program:

```
Option Explicit On
Option Strict On

Module TestModule

    Sub Main()

        'Step 1a-Create an DEFAULT Object
        Dim objPerson1 As clsPerson
        objPerson1 = New clsPerson

        'Step 1b-Create PARAMETERIZED Object with arguments
        Dim objPerson2 As clsPerson
        objPerson2 = New clsPerson("Mary Jones", 444, #1/22/1973#, _
                                   "100 Brooklyn Ave.", "718-260-5555")


        'Step2-TEST CONSTRUCTORS by Calling an Object Print() method
        'immediately after creation of objects
        objPerson1.Print()
        objPerson2.Print()


        'Step 3-Test SET Properties
        With objPerson1
            .Name = "Joe Smith"
            .IDNumber = 111
            .BirthDate = #12/12/1965#
            .Address = "100 Flatbush Ave Brooklyn, NY 10016"
            .Phone = "718-555-5454"
        End With

        'Step 4-Test GET Properties by displaying content of objects
        Console.WriteLine()
        Console.WriteLine()
        Console.WriteLine("Person1 Object's Data is:")
        Console.WriteLine(objPerson1.Name)
        Console.WriteLine(objPerson1.IDNumber)
        Console.WriteLine(objPerson1.BirthDate)
        Console.WriteLine(objPerson1.Address)
        Console.WriteLine(objPerson1.Phone)

        'Step 5-Testing Methods. Actually we already tested the methods earlier
        objPerson2.Print()



    End Sub

End Module
```

Step 7: Compile & Run the program:



```
, 0, 1/1/1900, , (000)-000-0000
Mary Jones, 444, 1/22/1973, 100 Brooklyn Ave., 718-260-5555


Person1 Object's Data is:
Joe Smith
111
12/12/1965 12:00:00 AM
100 Flatbush Ave Brooklyn, NY 10016
718-555-5454
Mary Jones, 444, 1/22/1973, 100 Brooklyn Ave., 718-260-5555
Press any key to continue . . .
```

5.3.6 CLASSROOM LABORATORY EXERCISE – Test Program

Creating and Testing an INVOICE Class Using Module-Driven Windows Application

Problem statement:

- Using a ***Module-Driven Windows Application***, create an Invoice Class. This class has the following characteristics:
  - Invoice class represents an business transaction invoice object.
  - Contain data such as company information (Company name et.,), customer information (Name, ID etc) and transaction information (Total, tax etc.).
  - Class contain the required method that will allow invoice objects to perform transactions such as tax calculations, total charges calculations etc.

- Other program requirements:
  - Using a ***Module-Driven Windows Application***, without a FORM!, write a simple driver program to test the class. This test program will be used to create objects using default & Parameterized Constructors as well as testing each Property and Methods.
  - In your program, OPTION STRICT and OPTION EXPLICITS should be set to ON in all your code modules
  - DO NOT CREATE A FORM TO TEST YOUR CLASS! USE A STANDARD MODULE OBJECT.

I. Class Requirements

- The class contains the following data, properties & methods members
  Class Person Member Data:

Data Member	Data Member
CompanyName: Type String	TransactionDate: Type Date
CompanyPhone: Type String	SubTotal: Type Decimal
CompanyAddress: Type String	TotalTax: Type Decimal
InvoiceID: Type Integer	Total: Type Decimal
CustomerName: Type String	StateTax: Constant Decimal (8.25%)
CustomerID: Type Integer	
CustomerAddress: Type String	
CustomerPhone: Type String	

Class Member Properties & Methods:

- Create the required Properties for each data member. DECIDE IF ANY DATE MEMBER REQUIRES A PROPERTY OR NOT
- Create the following class methods:

1. **Default Constructor** – Initializes the values to default. Select appropriate defaults (Use your imagination)
2. **Parameterized constructor** – Allow the creation of objects that will initialize attributes of the class with EXTERNAL DATA, with the following EXCEPTIONS: *SubTotal*, *TotalTax*, *Total*. Do not include these values in the parameter list of the constructor!!!
3. **Sub Procedure Print()** – Displays Invoice data using a *Message Box*! Data should be displayed as a COMMA-DELIMITED STRING
4. **Function CalculateTotal()** – When called returns the total charges for this Invoice Transaction. This value is calculated as follows:

Total = SubTotal + (SubTotal * STATE_TAX)

5. **Sub Procedure CalculateSubTotal(Parmeter1, Parameter2)** – Calculates the sub total price of the Invoice based on the number of items purchased and the price of the items. The methods parameters and calculation formula are as follows:

   - **By Value Parameter 1:** NumberOfItemsPurchased
   - **By Value Parameter 2:** ItemPrice
   - **Formula:**

   SubTotal = SubTotal + (NumberOfItemsPurchased * ItemPrice)

II. Module & Program Requirements

- ❑ The module requirements are as follows:

1. Test constructors
2. Test or validate the data mechanism
3. Test all methods

III. Required Results

- ❑ Submit the following information:

a. **A printed output - Printed results of test. (**You can use MS Paint and copy/paste from the screen if necessary)

b. **A printed copy of the class code**
c. **A printed copy of the module code**
d. **All documents must have your name and ID printed with the code**
e. **Submit all documents include your name**

5.3.5 Sample Program 2 – Working With Forms & Custom Objects (VERSION 1)

Form Driven Application – Small Business Application Using Person Class

Problem statement:

- Using a Form Driven Application (Startup Object = Form) we will demonstrate various methods of working with forms and custom objects. This example creates several forms for Data Entry, and Displaying Customer Information as well as the Manager information for a small business.

- NOTE that we will implement this application ONLY USING WHAT WE HAVE LEARNED UP TO THIS POINT. Nevertheless, it will truly test our understanding of OBJECTS AND CLASSES up to this point.
- This example is for **TEACHING PURPOSE ONLY!** A true Customer Management application will use more advance .NET components which will make creating this application more practical.

Application Architecture Introduction (Separating Interface from Implementation)

- We will also begin to INTRODUCE THE class to proper application programming ARCHITECTURE and FORMATS adhering to BEST PRACTICE by making all attempts to SEPARATE INTERFACE from IMPLEMENTATION.
- What we mean is we SEPARATE USER-INTERFACE CODE with PROCESSING. This is done as follows:

  - User-interface code or the code in the FORMS will contain NO PROCESSING CODE!
  - FORMS will only contain User-Interface code or code to interact with USER ONLY!
  - FORM code includes MESSAGE BOXES, UI CONTROLS manipulation, getting data from user, displaying data to user.
  - All PROCESSING CODE will reside in the MODULE INSIDE PROCESSING METHODS!
  - PROCESSING METHODS IN MODULE will be CALLED BY THE FORMS TO DO THE WORK!
  - PROCESSING MODULE will contain LITTLE or NO FORM CODE! SUCH AS CALLS to FORMS OR FORM CONTROLS
  - FORM CODE WILL INTERACT WITH USER AND CALL PROCESSING METHODS IN MODULE TO DO THE WORK!

- WE WILL SHOW TWO VERSION OF THIS PROGRAM. THE FIRST VERSION, contains a BUG or ISSUE!
- I discuss the BUG at the end of this example. IN THE SECOND VERSION, WE WILL RESOLVE THIS BUG!

Re-using Objects

- We will also review the OOP concept of **Reusability** by reusing the Person Class from the previous Console Application Example #1. As in Example 1, this class will be used to create objects with information of a Customer such as name, id, and birth date, address & phone number. Also the class should contain a method that will print the Customer's information. The only modification we will make in the Person Class is in the *PrintPerson()* Method. From previous Console Application example, this method called the *Console.WriteLine()* method. We will change these calls to a ***MessageBox.Show*** for a Windows Application.

Form, & Module Requirements

- The main or driver program will utilize several forms. A Main Form as a startup point to invoke the Customer Management & Customer Information Forms. Each one of these Forms will perform their function and manage the objects created in the program.
- In addition the program will contain a Module where several global public Objects will reside that represent a SIMMULATED DATABASE OF CUSTOMER OBJECTS.

Additional Requirements

- This Example will demonstrate the following topics:

    - Windows Form-Driven Application Example
    - Creating Classes,
    - Reusing Classes from previous programs
    - Creating, initializing Objects
    - Using Objects as follows:
        - (Set, Get, Calling methods, triggering Form Object event-handlers, & programming Form objects event-handlers)
        - Working with Objects & Forms
        - Global Objects in a Module
        - Assigning Object Reference to one another (Object to Object Interaction)

Class Requirements

- The class contains the following data, properties & methods members

    Class Person Member Data:

Name:	Type String
IDNumber:	Type Integer.
BirthDate:	Date
Address:	Type String
Phone:	Type String

Class Member Properties & Methods:

- Set & Get Properties for each data member.
- The Method PrintPerson(), which displays the Persons data

Form Requirements

- The application will contains the following Forms:

frmMainForm:	Portal to navigate to other Forms
frmCustomerManagement:	Form contains controls to Search, Add, Edit, Delete, Print and & Print all Customers
frmManagerInformation:	Form contains controls to display the Manager's information

- Note that the Forms will create any necessary FORM-LEVEL Objects & Form Event-Handlers that respond to user interactions.

Module Requirements

- The application will contains one Module with the following requirements:

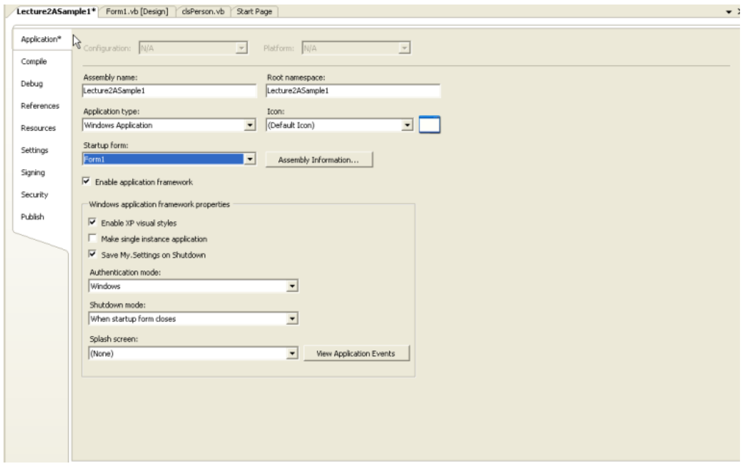
    - ***modMainModule***: Module to contain the following:
        - 5 ***Global*** Objects **POINTERS** of the Person Class. These 5 POINTERS are to be populated by the program with PERSON OBJECTS which represent the Customer DATABASE
        - One ***Global*** OBJECT of the person class. This object is created with data using the PARAMETERIZED CONSTRUCTOR and represents the company's manager.
        - The following PROCESSING METHODS:
            - ***Sub Initialize()*** – Creates 5 PERSON OBJECTS with data and assigns them to the 5 POINTERS in database.
            - ***Function Search(ID)*** – Search database (5 Object) for the object whose ID is the parameter. RETURNS a POINTER to the object found or returns a NOTHING if not found.
            - ***Function Add(OBJECT POINTER)*** – Searches database (5 Objects) for a NULL POINTER and has it POINT to the Object Pointer pass as Parameter. Returns a TRUE if empty pointer found, else FALSE
            - ***Function Remove(ID)*** – Search database (5 Object) for the object whose ID is the parameter. REMOVES by setting POINTER to NOTHING. RETURNS a TRUE if found or returns FALSE otherwise.
            - ***Function Print(ID)*** – Search database (5 Object) for the object whose ID is the parameter. Calls PRINT() method of object. RETURNS a TRUE if found or returns FALSE otherwise.
            - ***Sub PrintALL()*** – Search database and calls PRINT() method of EACH OBJECT in database.

HOW IT'S DONE:

Part I – Create The Class:

Step 1: Start a new Windows Application. By default the program will behave as a Form Driven Application:

Object	Property	Value
Project	Name Startup Object	**CustomerFormWinApp** **Form1**



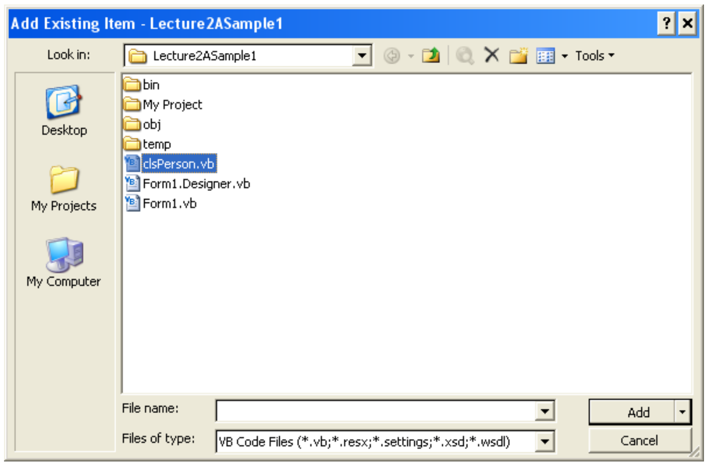
Configuration: N/A
Platform: N/A
Assembly name: Lecture2ASample1
Root namespace: Lecture2ASample1
Application type: Windows Application
Icon: (Default Icon)
Startup form: Form1
[✓] Enable application framework
Windows application framework properties
[✓] Enable XP visual styles
[ ] Make single instance application
[✓] Save My.Settings on Shutdown
Authentication mode: Windows
Shutdown mode: When startup form closes
Splash screen: (None)

Step 2: Prepare to Reuse the Person Class from Previous Console Application, by Copying the File from previous Application Folder to the Folder of this Windows Application Project

1. Using Windows Explorer, navigate to the Folder of the previous example Console Application Sample Program 1.
2. Copy/Paste the file **clsPerson.vb** to this Project folder

Step 3: Add the Class to the Project

1. In the Project Menu, select ***Add***| ***Existing Item…*** and navigate to the project folder



Look in: Lecture2ASample1
bin
My Project
obj
temp
clsPerson.vb
Form1.Designer.vb
Form1.vb
File name:
Files of type: VB Code Files (*.vb;*.resx;*.settings;*.xsd;*.wsdl)

2. Select the ***clsPerson.vb*** File and click OK
3. The class is now part of the project and ready to be Reused!

Step 4: In the Class Module code window enter the code for the private data:

```
Option Explicit On
Option Strict On

Public Class clsPerson
    '*********************************************************************
    'Class Data or Variable declarations
    Private m_Name As String
    Private m_IDNumber As Integer
    Private m_BirthDate As Date
    Private m_Address As String
    Private m_Phone As String
```

Step 5: In the Class Module code window enter the code for public Properties:

```
    'Property Procedures
    Public Property Name() As String
        Get
            Return m_Name
        End Get
        Set(ByVal Value As String)
            m_Name = Value
        End Set
    End Property

    Public Property IDNumber() As Integer
        Get
            Return m_IDNumber
        End Get
        Set(ByVal Value As Integer)
            m_IDNumber = Value
        End Set
    End Property

    Public Property BirthDate() As Date
        Get
            Return m_BirthDate
        End Get
        Set(ByVal Value As Date)
            m_BirthDate = Value
        End Set
    End Property

    Public Property Address() As String
        Get
            Return m_Address
        End Get
        Set(ByVal Value As String)
            m_Address = Value
        End Set
    End Property

    Public Property Phone() As String
        Get
            Return m_Phone
        End Get
        Set(ByVal Value As String)
            m_Phone = Value
        End Set
    End Property
```

Step 6: In the Class Module code window enter the code for Constructor Methods (Non-Parameter and/or Parameterized):

```
    '*********************************************************************
    'Class Constructor Methods

    'Default Constructor
    Public Sub New()
        'Note that private data members are being initialized
        m_Name = ""
        m_IDNumber = 0
        m_BirthDate = #1/1/1900#
        m_Address = ""
        m_Phone = "(000)-000-0000"
    End Sub

    'Parameterized Constructor
    Public Sub New(ByVal Name As String, ByVal IDNum As Integer, ByVal BDate As Date, _
    ByVal Address As String, ByVal Phone As String)
        'Note that as example we are NOT using the private data but
        'the Property Procedures instead when setting the data via the constructor

        Me.Name = Name
        Me.IDNumber = IDNum
        Me.BirthDate = BDate
        Me.Address = Address
        Me.Phone = Phone

    End Sub
```

Step 7: MODIFY the Print() method by replacing the *Console.Writeline* with *MessageBox.Show* statement:

```
    '*********************************************************************
    '*********************************************************************
    'Class Methods
    '*********************************************************************

    'Author of base class allows sub classes to overide Print()
    'If they want to, it is not mandatory
    Public Overridable Sub Print()

       'Display object Content
        MessageBox.Show(m_Name & ", " & m_IDNumber & ", " & _
        m_BirthDate & ", " & m_Address & ", " & m_Phone)

    End Sub

End Class
```

- **NOTE THAT THIS CLASS CONTAINS A USER-INTERFACE CODE VIA A MESSAGE BOX. THIS IS ONLY FOR TEACHING PURPOSE!!**
- **YOU SHOULD NOT DISPLAY ANY MESSAGE BOXEX OR USER-INTERFACE CODE FROM WITHIN A CLASS IN HWS AND PROJECTS UNLESS OTHERWISE INSTRUCTED!!**

Part II & III – Create & Use The Objects (The User Interface Code)

Standard Module:

Step 7: Add a Module to the Project and set its properties as show in table below:

Object	Property	Value
Module	Name	**MainModule**
	Text	**MainModule**

Step 8: Add Module GLOBAL declarations:

- In the module, we will declare 5 *clsPerson* OBJECT POINTERS. These POINTERS will eventually point to objects which will represent our simulated DATABASE OF CUSTOMERS!
- In addition we create one complete OBJECT using parameterized constructor with data. This object represents a business employee manager.

```
Option Explicit On
Option Strict On

Module MainModule

'******************************************************************************
' GLOBAL VARIABLES & OBJECT DECLARATIONS SECTION
'******************************************************************************

'Declare  5 POINTERS to Customer Object (REPRESENT OUR SIMMULATED DATABASE)
Public objCustomer1, objCustomer2, objCustomer3, objCustomer4, objCustomer5 As clsPerson


'Declare & Create Public Object representing THE MANAGER employee, initialized with Data
Public objManager As clsPerson = New clsPerson("Alex Rod", 777, #12/12/1971#, _
                                               "192 East 8th, Brooklyn", "718-434-6677")
```

Step 9: Add Module INITIALIZE() Method declarations:

- Mow we begin to add PROCESSING METHODS TO THE MODULE that will do the work for the FORMS.
- The first method we implement is the INITIALIZE() method. This sub procedure creates 5 OBJECTS of the PERSON CLASS and assigns them to the 5 GLOBAL POINTERS.
- At this point the simulated DATABASE OF CUSTOMERS is not POPULATED WITH OBJECTS!

```
'*****************************************************************************
' METHOD DECLARATIONS
'*****************************************************************************
''' <summary>
''' Intended to execute at the start of the program. Can be used to perform
''' any initialization. In this case, to populate EACH POINTER with OBJECT at
''' start of the program. We are simply populating the database with data.
''' </summary>
''' <remarks></remarks>
Public Sub Initialize()
    'Create objects and initialize with data via paremterized constructor
    objCustomer1 = New clsPerson("Joe", 111, #12/12/1965#, "111 Jay Street", _
                                 "718-434-5544")

    objCustomer2 = New clsPerson("Angel", 222, #1/4/1972#, "222 Flatbush Ave", _
                                 "718-234-5524")

    objCustomer3 = New clsPerson("Sam", 333, #9/21/1960#, "333 Dekalb Ave", _
                                 "718-890-3422")

    objCustomer4 = New clsPerson("Mary", 444, #7/4/1970#, "444 Jay Street", _
                                 "718-444-1122")

    objCustomer5 = New clsPerson("Nancy", 555, #12/12/1965#, "555 Flatlands Ave", _
                                 "718-434-9876")

End Sub
```

Step 10: Create SEARCH(ID) FUNCTION declarations:

- Purpose of this method is to search the database (5 Customer OBJECTS) for the object whose ID is parameter and return a POINTER to the object
- How it works:
  - Nested **ElseIf** statement is used
  - During each **ElseIf**, the OBJECT'S IDNUMBER PROPERTY is tested against the ID parameter (PROPERTY = ID?), if a match is found or TRUE, the code inside the body of the If statement is executed.
  - This code simply RETURNS the POINTER to the OBJECT. REMEMBER, THE RETURN KEYWORD ALSO EXITS THE METHOD.
  - If a match (PROPERTY = ID?) is not found or FALSE, then you simply skip the OBJECT and go to the next **ElseIf**.
  - This process repeats itself until the last ELSE is found, at this point a NOTHING is returned since we searched the database did not find the object.

```
'*******************************************************************************
''' <summary>
''' Function Searches the database for POINTER to object whose ID is a parameter
''' Interrogates object for the ID, when found Returns POINTER to OBJECT and EXITS!
''' </summary>
''' <param name="IDNum"></param>
''' <returns></returns>
''' <remarks></remarks>
Public Function Search(ByVal IDNum As Integer) As clsPerson
    'Step 1-Search for object with ID
    If objCustomer1.IDNumber = IDNum Then
        'Step 2-Return Pointer to object and EXIT
        Return objCustomer1

    ElseIf objCustomer2.IDNumber = IDNum Then
        Return objCustomer2

    ElseIf objCustomer3.IDNumber = IDNum Then
        Return objCustomer3

    ElseIf objCustomer4.IDNumber = IDNum Then
        Return objCustomer4

    ElseIf objCustomer5.IDNumber = IDNum Then
        Return objCustomer5

    Else
        'Not found
        Return Nothing

    End If
End Function
```

Step 11: Create ADD(object) FUNCTION declarations:

- Purpose of this method is to ADD a new Customer to the database (5 Customer OBJECTS). True is returned if successful, False if no EMPTY OR NULL POINTERS ARE AVAILABLE.
- How it works:
  - Nested **ElseIf** statement are used to ask EACH POINTER if is pointing to a NOTHING OR NULL.
  - During each **ElseIf**, if the POINTER IS NOTHING then parameter DATABASE POINTER is assigned to PARAMETER POINTER, thus adding the object to database.
  - A TRUE is returned and function EXITS.
  - ONLY AFTER ALL DATABASE POINTERS ARE TESTED AND NO NULLS ARE FOUND is a FALSE RETURNED.

```
    '*****************************************************************************
    ''' <summary>
    ''' Function Adds NEW objects passed as parameter to database.
    ''' Searches for the FIRST nothing or empty POINTER and adds object to that POINTER.
    ''' Returns a TRUE When OBJECT added OR FALSE when no empty POINTERS and EXITS!!
    ''' </summary>
    ''' <param name="objE"></param>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Public Function Add(ByVal objE As clsPerson) As Boolean

        'Step 1-Ask if object is NULL
        If objCustomer1 Is Nothing Then
            'Step 2-If NULL, database POINTER = Paramter POINTER
            objCustomer1 = objE
            'Step 3-Return success & EXIT
            Return True

        ElseIf objCustomer2 Is Nothing Then
            objCustomer2 = objE
            Return True

        ElseIf objCustomer3 Is Nothing Then
            objCustomer3 = objE
            Return True

        ElseIf objCustomer4 Is Nothing Then
            objCustomer4 = objE
            Return True

        ElseIf objCustomer5 Is Nothing Then
            objCustomer5 = objE
            Return True

        Else
            'No space available
            Return False

        End If
    End Function
```

Step 12: Create REMOVE(ID) FUNCTION declarations:

- This method begins to search the database (5 Customer OBJECTS) for the object whose ID is parameter. Once found it removes object from the database. Returns a TRUE if successful and FALSE if object is NOT FOUND.
- How it works:
  - Again, nested **ElseIf** statement are used
  - During each **ElseIf**, the OBJECT'S IDNUMBER PROPERTY is tested against the ID parameter (PROPERTY = ID?), if a match is found or TRUE, the code inside the body of the If statement is executed.
  - This code simply SETS THE DATABASE POINTER TO NOTHING, thus removing the object it is pointing to.
  - A TRUE is returned & function EXITS, when a match and deletion is made. Otherwise if all DATABASE POINTERS are tested and a match is NOT FOUND, then a FALSE is returned and function EXITS.

```
'*******************************************************************************
''' <summary>
''' Function Removes object from database by searching for OBJECT whose ID is
''' a parameter. Interrogates each the object for the ID.
''' When found, Removes object by setting POINTER TO NOTHING
''' Returns a TRUE When removed, FALSE when OBJECT not found and EXITS Function!
''' </summary>
''' <param name="IDNum"></param>
''' <returns></returns>
''' <remarks></remarks>
Public Function Remove(ByVal IDNum As Integer) As Boolean

    'Step 1-Search for object with ID
    If objCustomer1.IDNumber = IDNum Then
        'Step 2-When found set to nothing
        objCustomer1 = Nothing
        'Step 3-Return OK & EXIT
        Return True

    ElseIf objCustomer2.IDNumber = IDNum Then
        objCustomer2 = Nothing
        Return True

    ElseIf objCustomer3.IDNumber = IDNum Then
        objCustomer3 = Nothing
        Return True

    ElseIf objCustomer4.IDNumber = IDNum Then
        objCustomer4 = Nothing
        Return True

    ElseIf objCustomer5.IDNumber = IDNum Then
        objCustomer5 = Nothing
        Return True

    Else
        'Return FALSE & EXIT
        Return False
    End If

End Function
```

Step 13: Create PRINT(ID) FUNCTION declarations:

- Purpose of this method is to search the database (5 Customer OBJECTS) for the object whose ID is parameter. Once found it CALLS PRINT() METHOD of OBJECT. Returns a TRUE if successful and FALSE if object is NOT FOUND.
- How it works:
  - Again, nested **ElseIf** statement are used
  - During each **ElseIf**, the OBJECT'S IDNUMBER PROPERTY is tested against the ID parameter (PROPERTY = ID?), if a match is found or TRUE, the code inside the body of the If statement is executed.
  - This code simply CALLS THE PRINT() METHOD to execute, thus having the OBJECT PRINT ITSELF.
  - A TRUE is returned & function EXITS, when a match and method calling is done. Otherwise if all OBJECS are tested and NO MATCH IS FOUND, then a FALSE is returned and function EXITS.

```
'*****************************************************************************
''' <summary>
''' Function Prints object by searching for OBJECT whose ID is a parameter
''' Interrogates each object for the ID.
''' When found, CALLS the PRINT() METHOD in the object
''' Returns a TRUE When printed OR FALSE when OBJECT not found and EXITS!
''' </summary>
''' <param name="IDNum"></param>
''' <returns></returns>
''' <remarks></remarks>
Public Function Print(ByVal IDNum As Integer) As Boolean
    'Step 2-Search for object with ID
    If objCustomer1.IDNumber = IDNum Then
        'Step 3-When found set to nothing
        objCustomer1.Print()
        'Step 4-Return OK
        Return True

    ElseIf objCustomer2.IDNumber = IDNum Then
        objCustomer2.Print()
        Return True

    ElseIf objCustomer3.IDNumber = IDNum Then
        objCustomer3.Print()
        Return True

    ElseIf objCustomer4.IDNumber = IDNum Then
        objCustomer4.Print()
        Return True

    ElseIf objCustomer5.IDNumber = IDNum Then
        objCustomer5.Print()
        Return True

    Else
        'Return False Not found
        Return False
    End If

End Function
```

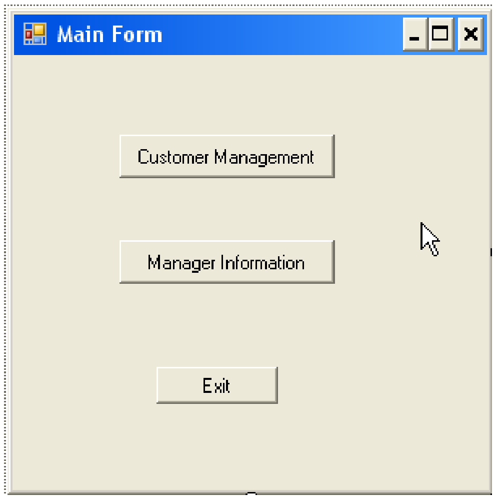
Step 14: Create PRINTALL() SUB declarations:

- Purpose of this SUB method is to simply call EACH DATABASE OBJECT'S (5 Customer OBJECTS) PRINT() method, thus printing ALL OBJECTS.
- How it works:
  - Calls each DATABASE OBJECT'S PRINT() METHOD.

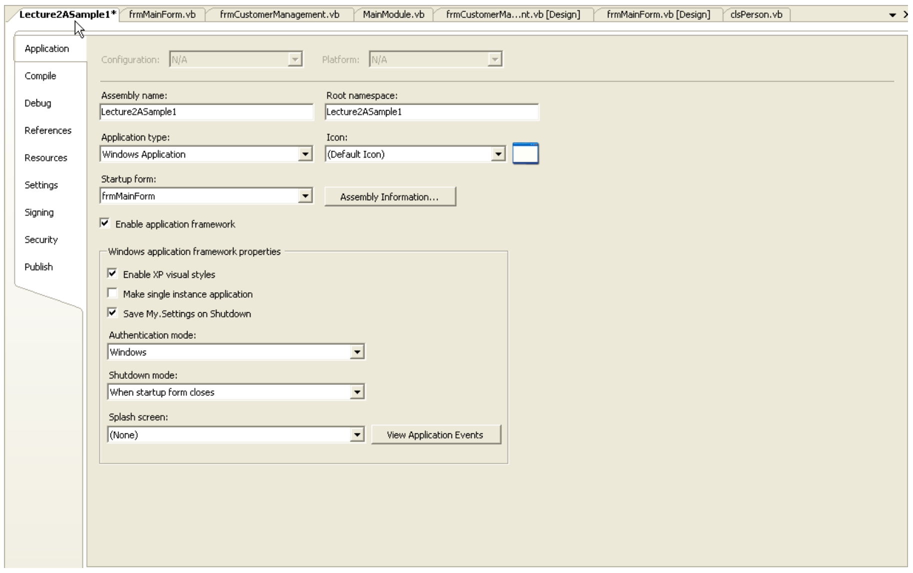
```
'*****************************************************************************
''' <summary>
''' Prints all objects in database by CALLING each object's PRINT() METHOD
''' </summary>
''' <remarks></remarks>
Public Sub PrintAll()

    objCustomer1.Print()

    objCustomer2.Print()

    objCustomer3.Print()

    objCustomer4.Print()

    objCustomer5.Print()

End Sub     End Function
```

Step 15: RENAME Form1 to frmMainForm. Set the controls as shown in figure below:

Object	Property	Value
Form1	Name Text	**frmMainForm** **Main Form**



Step 16: Since this is a Windows Form-Driven Windows Application, the Startup Object will be set to frmMainForm:



Configuration: N/A Platform: N/A

Assembly name: Lecture2ASample1

Root namespace: Lecture2ASample1

Application type: Windows Application

Icon: (Default Icon)

Startup form: frmMainForm

Assembly Information...

☑ Enable application framework

Windows application framework properties

☑ Enable XP visual styles

☐ Make single instance application

☑ Save My.Settings on Shutdown

Authentication mode: Windows

Shutdown mode: When startup form closes

Splash screen: (None)

View Application Events

Step 17: Main Form, FORM-LEVEL DECLARATIONS AND LOAD EVENT:

```
Option Explicit On
Option Strict On

Public Class frmMainForm

    '*****************************************************************************
    ' EVENT-HANDLER DECLARATIONS SECTION
    '*****************************************************************************

    '*****************************************************************************
    ''' <summary>
    ''' Calls Initialize() method in MODULE to LOAD OBJECTS to each pointer in database.
    ''' </summary>
    ''' <param name="sender"></param>
    ''' <param name="e"></param>
    ''' <remarks></remarks>
Private Sub frmMainForm_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load

        'Perform initialization by calling Module.Initialize() method
        Initialize()

End Sub
```

Step 18: Main Form CustomerManagement_Click Event:

```
    '*****************************************************************************
    ''' <summary>
    ''' Click Event creates object of Customer Management Form
    ''' Calls METHOD in Form OBJECT so object can show itself
    ''' </summary>
    ''' <param name="sender"></param>
    ''' <param name="e"></param>
    ''' <remarks></remarks>
Private Sub btnCustomerManagement_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnCustomerManagement.Click
        'Step 1-Create object of Customer Management Form
        Dim objCustomerManagementForm As New frmCustomerManagement


        'Step 2-Display Customer Management Form
        objCustomerManagementForm.ShowDialog()

End Sub
```

Step 19: Main Form MANAGER BUTTON CLICK EVENT:

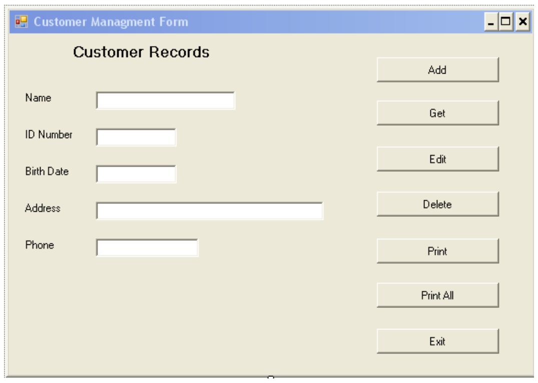
```
    '******************************************************************************
    ''' <summary>
    ''' Click Event creates object of the Manager Information Form
    ''' Calls METHOD in Form OBJECT so object can show itself
    ''' </summary>
    ''' <param name="sender"></param>
    ''' <param name="e"></param>
    ''' <remarks></remarks>
Private Sub btnManager_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnManager.Click
        'Step 1-Create object of Employee Management Form
        Dim objManagerInformationForm As New frmManagerInformation


        'Step 2-Display Employee Management Form
        objManagerInformationForm.ShowDialog()
End Sub
```

Step 20: Main Form Exit_Click Event:

```
    '******************************************************************************
    ''' <summary>
    ''' Event-handler calls Form Close() method to close the Form.
    ''' </summary>
    ''' <param name="sender"></param>
    ''' <param name="e"></param>
    ''' <remarks></remarks>
Private Sub btnExit_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnExit.Click
        'Step 1-Close yourself
        Me.Close()

End Sub


End Class
```

Step 21: Create the Customer Management Form and add the controls shown below:

Object	Property	Value
Form2	Name	**frmCustomerManagement**
	Text	**Customer Management**



Step 22 FORM-LEVEL DECLARATIONS & OBJECT POINTER:

```
Option Explicit On
Option Strict On

Public Class frmCustomerManagement
    '*******************************************************************************
    ' FORM-LEVEL VARIABLES & OBJECT DECLARATIONS SECTION
    '*******************************************************************************
    'Module-level Object POINTER Declaration
    Private objCustomer As clsPerson
```

Step 23 FORM LOAD EVENT:

```
    '*******************************************************************************
    ' EVENT-HANDLER DECLARATIONS SECTION
    '*******************************************************************************
    ''' <summary>
    ''' Form_Load event. Create object and popoulate Form controls
    ''' With object's default values
    ''' </summary>
    ''' <param name="sender"></param>
    ''' <param name="e"></param>
    ''' <remarks></remarks>
Private Sub frmCustomerManagement_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        'Create Form-Level Object
        objCustomer = New clsPerson

        'Populate Form Controls with Object's DEFAULT data
        With objCustomer
            txtName.Text = .Name
            txtIDNumber.Text = CStr(.IDNumber)
            txtBirthDate.Text = CStr(.BirthDate)
            txtAddress.Text = .Address
            txtPhone.Text = .Phone
        End With

End Sub
```

Step 24 Add code to the GET CLICK EVENT:

```
    '***************************************************************************
    ''' <summary>
    ''' Calls Search method of module to search database for object
    ''' whose ID is passed as argument. Returns a pointer to the object
    ''' found, else returns a Nothing.
    ''' </summary>
    ''' <param name="sender"></param>
    ''' <param name="e"></param>
    ''' <remarks></remarks>
Private Sub btnGet_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnGet.Click
        'Step 1-Call Overloaded Search(ID) to search for object that match ID
        'Return pointer to object found.
        objCustomer = Search(CInt(txtIDNumber.Text.Trim))

        'Step 2-If result of search is Nothing, then display customer is not found
        If objCustomer Is Nothing Then
            MessageBox.Show("Customer Not Found")

            'Step 3-Clear all controls
            txtName.Text = ""
            txtIDNumber.Text = ""
            txtBirthDate.Text = ""
            txtAddress.Text = ""
            txtPhone.Text = ""
        Else
            'Step 4-Then Data is extracted from customer object & placed on textboxes
            With objCustomer
                txtName.Text = .Name
                txtIDNumber.Text = CStr(.IDNumber)
                txtBirthDate.Text = CStr(.BirthDate)
                txtAddress.Text = .Address
                txtPhone.Text = .Phone
            End With
        End If
End Sub
```

Step 25 ADD code for the ADD CLICK EVENT:

```
    '****************************************************************************
    ''' <summary>
    ''' Calls Module Add method to Add a new object to the object database
    ''' </summary>
    ''' <param name="sender"></param>
    ''' <param name="e"></param>
    ''' <remarks></remarks>
Private Sub btnAdd_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnAdd.Click
        'Step 1-Create New Object (OBJECT MUST BE A NEW OBJECT)
        objCustomer = New clsPerson

        'Step 2-Populate Form Level Object with Data from Controls
        With objCustomer
            .Name = txtName.Text.Trim
            .IDNumber = CInt(txtIDNumber.Text.Trim)
            .BirthDate = CDate(txtBirthDate.Text.Trim)
            .Address = txtAddress.Text.Trim
            .Phone = txtPhone.Text.Trim
        End With


        'Step 3-Call Add method to add new Customer to database
        Dim result As Boolean = Add(objCustomer)

        'Step 4-Test results & prompt user
        If result Then
            MessageBox.Show("Custome Added Successfully")
        Else
            MessageBox.Show("Database Full")
        End If

End Sub
```

Step 26 Add code to the EDIT CLICK EVENT:

```
    '*****************************************************************************
    ''' <summary>
    ''' Calls Module Search method to search database for object whose ID
    ''' is passed as argument and return its pointer and sets the object's
    ''' Properties with data from Form controls.
    ''' </summary>
    ''' <param name="sender"></param>
    ''' <param name="e"></param>
    ''' <remarks></remarks>
Private Sub btnEdit_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnEdit.Click
        'Step 1-Call Overloaded GetItem(ID) to search for object that match ID
        objCustomer = Search(CInt(txtIDNumber.Text.Trim))

        'Step 2-If result of search is Nothing, then display customer is not found
        If objCustomer Is Nothing Then
            'Step 3-Inform user customer not in database
            MessageBox.Show("Customer Not Found")
        Else
       'Step 4-Populate Form Level POINTER with Data from Controls, EXCEPT THE IDNUMBER
            'What you do to the pointer, you are doing to the object in our Database.
            With objCustomer
                .Name = txtName.Text.Trim
                .BirthDate = CDate(txtBirthDate.Text.Trim)
                .Address = txtAddress.Text.Trim
                .Phone = txtPhone.Text.Trim
            End With
        End If

End Sub
```

Step 27 Add code for the DELETE CLICK EVENT:

```
    '*****************************************************************************
    ''' <summary>
    ''' Calls Module Remove method to delete the object from the database
    ''' based on ID or key
    ''' </summary>
    ''' <param name="sender"></param>
    ''' <param name="e"></param>
    ''' <remarks></remarks>
Private Sub btnDelete_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnDelete.Click

        'Step 1-Call Add method to add new Customer to database
        Dim result As Boolean = Remove(CInt(txtIDNumber.Text.Trim))

        'Step 2-Test results & promt user
        If result Then
            MessageBox.Show("Customer Deleted")
        Else
            MessageBox.Show("Customer Not Found!")
        End If

End Sub
```

Step 28 Add code for the PRINT CLICK EVENT:

```
    '*****************************************************************************
    ''' <summary>
    ''' Calls module Print method to print the object's properties
    ''' </summary>
    ''' <param name="sender"></param>
    ''' <param name="e"></param>
    ''' <remarks></remarks>
Private Sub btnPrint_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnPrint.Click

        'Step 1-Call Add method to add new Customer to database
        Dim result As Boolean = Print(CInt(txtIDNumber.Text.Trim))

        'Step 2-Test results & promt user
        If result <> True Then
            MessageBox.Show("Customer Not Found!")
        End If

End Sub
```

Step 29 Add code for the PRINT ALL CLICK EVENT:

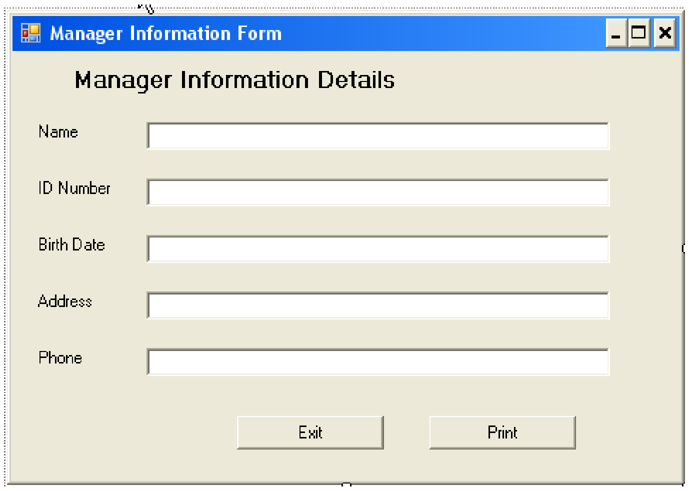
```
    '******************************************************************************
    ''' <summary>
    ''' Calls Module PrintAll method to print all the objects in database
    ''' </summary>
    ''' <param name="sender"></param>
    ''' <param name="e"></param>
    ''' <remarks></remarks>
Private Sub btnPrintAll_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnPrintAll.Click

        'Step 1-Call PrintAll method in module
        PrintAll()

End Sub
```

Step 30 Add code for the EXIT CLICK EVENT:

```
    '******************************************************************************
    ''' <summary>
    ''' Closes the Form
    ''' </summary>
    ''' <param name="sender"></param>
    ''' <param name="e"></param>
    ''' <remarks></remarks>
Private Sub btnExit_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnExit.Click

        Me.Close()

End Sub
```

Step 31 Create the MANAGER INFORMATION FORM and add the controls shown below:

Object	Property	Value
Form4	Name Text	**frmManagerInformationForm** **Manager Information Form**



Step 32 FORM-LEVEL DECLARATIONS & OBJECT POINTER:

```
Option Explicit On
Option Strict On

Public Class frmManagerInformation

    '*****************************************************************************
    ' FORM-LEVEL VARIABLES & OBJECT DECLARATIONS SECTION
    '*****************************************************************************
    'Module-level Object POINTER Declaration
    Private objEmployee As clsPerson
```

Step 33 FORM LOAD EVENT:

```
    '*****************************************************************************
    ' EVENT-HANDLER DECLARATIONS SECTION
    '*****************************************************************************

    '*****************************************************************************
    ''' <summary>
    ''' Event-handler when form is displayed. Form-Level POINTER, points to
    ''' MODULE MANAGER OBJECT POINTER. Manipulation of Form-Level pointer affect
    ''' MODULE MANAGER OBJECT.
    ''' Populates Form Textboxes with DATA from Form-Level POINTER or MANAGER OBJECT
    ''' in MODULE
    ''' </summary>
    ''' <param name="sender"></param>
    ''' <param name="e"></param>
    ''' <remarks></remarks>
Private Sub frmManagerInformation_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        'Form-Level Object POINTER points to MANAGER OBJECT
        objEmployee = objManager

        'Populate Form Controls with Object's data
        With objEmployee
            txtName.Text = .Name
            txtIDNumber.Text = CStr(.IDNumber)
            txtBirthDate.Text = CStr(.BirthDate)
            txtAddress.Text = .Address
            txtPhone.Text = .Phone
        End With
End Sub
```

Step 34 Add code to the PRINT CLICK EVENT:

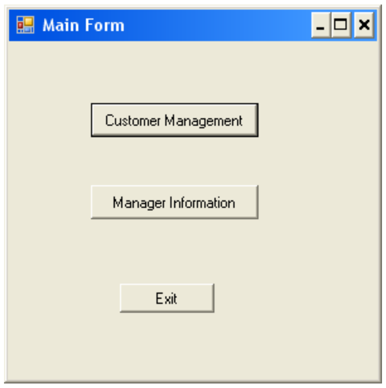
```
    '*****************************************************************************
    ''' <summary>
    ''' Event-handler call PRINT() METHOD of Form-Level object. Indirectly it is
    ''' actually calling PRINT() METHOD of MANAGER OBJECT IN MODULE
    ''' </summary>
    ''' <param name="sender"></param>
    ''' <param name="e"></param>
    ''' <remarks></remarks>
Private Sub btnPrint_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnPrint.Click
        'Tell object to print itself
        objEmployee.Print()
End Sub
```

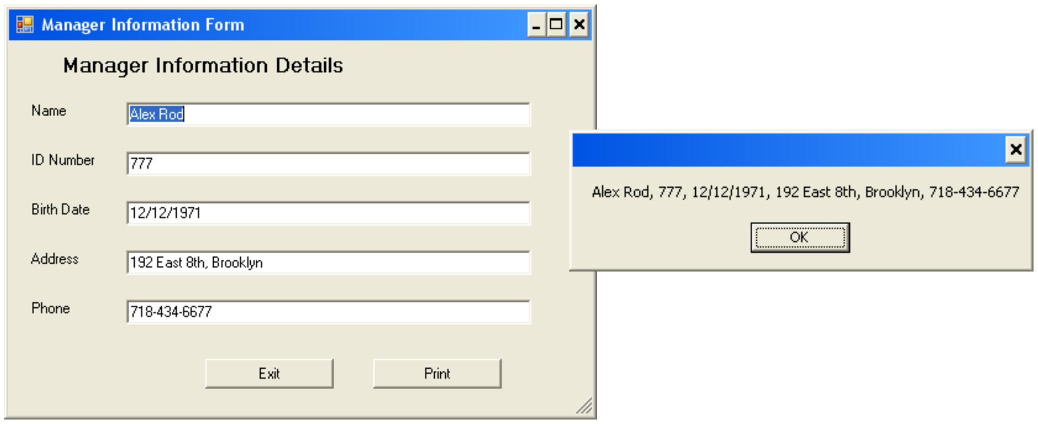
Step 35 Add code to the PRINT CLICK EVENT:

```
    '****************************************************************************
    ''' <summary>
    ''' Event-handler calls Form Close() method to close the Form.
    ''' </summary>
    ''' <param name="sender"></param>
    ''' <param name="e"></param>
    ''' <remarks></remarks>
Private Sub btnExit_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnExit.Click

        'Close yourself (Form)
        Me.Close()

End Sub

End Class
```

Step 36: BUILD & RUN APPLICATION

Test 1, 2 & 3 – Displaying Main & Manager Information Form. Also Clicking PRINT BUTTON:





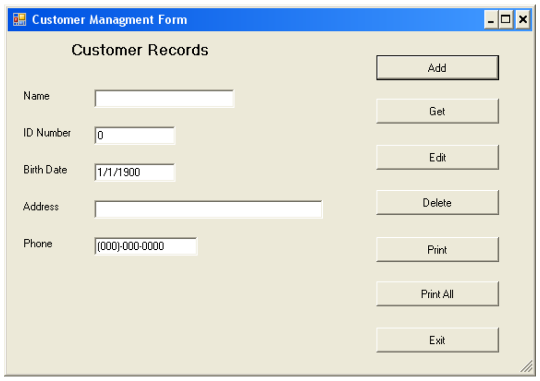
Name: Alex Rod
ID Number: 777
Birth Date: 12/12/1971
Address: 192 East 8th, Brooklyn
Phone: 718-434-6677

Alex Rod, 777, 12/12/1971, 192 East 8th, Brooklyn, 718-434-6677

RESULTS OF TESTS 1 – 3:

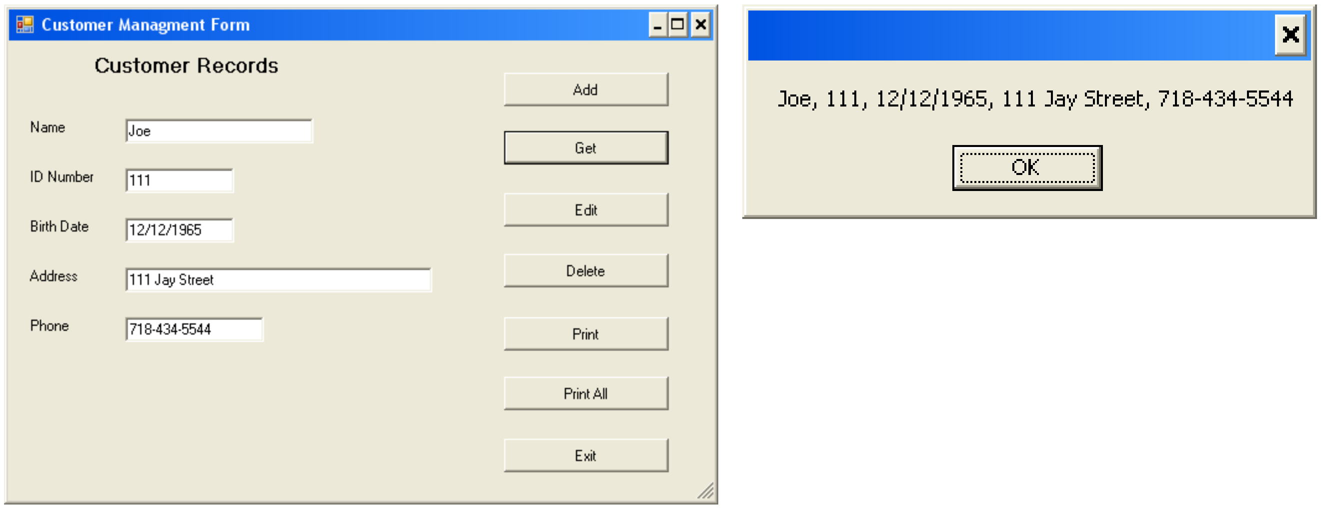
- ❖ **Main Form displayed successfully.**
- ❖ **Clicking the MANAGER INFORMATION BUTTON invoked the Manager Information Form**
- ❖ **Clicking PRINT BUTTON displayed the Manager's Information.**

Test 4 – Displaying Main & Customer Management Form:



Name:
ID Number: 0
Birth Date: 1/1/1900
Address:
Phone: (000)-000-0000

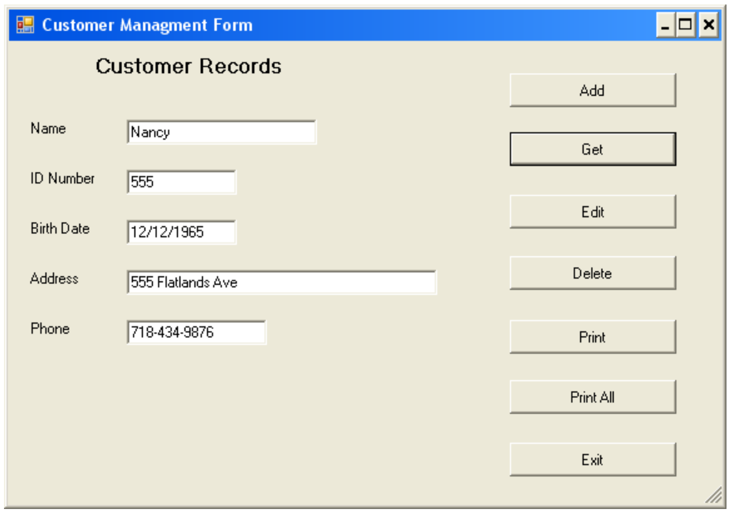
Test 5 & 6 – Enter 111 in ID text box and Click GET button to search database for Customer. Then Click PRINT button to display Customer's data:



Name: Joe
ID Number: 111
Birth Date: 12/12/1965
Address: 111 Jay Street
Phone: 718-434-5544

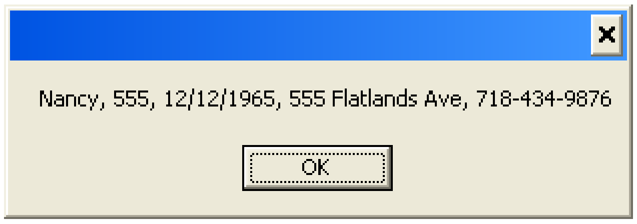
Joe, 111, 12/12/1965, 111 Jay Street, 718-434-5544

Test 7 & 8 – Enter 555 in ID text box and Click GET button to search database for Customer. Then Click PRINT button to display Customer's data:

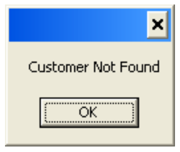

Customer Managment Form

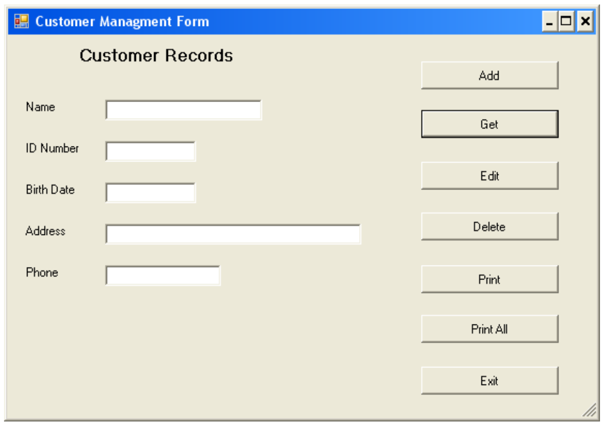
Customer Records

Name	Nancy
ID Number	555
Birth Date	12/12/1965
Address	555 Flatlands Ave
Phone	718-434-9876


Nancy, 555, 12/12/1965, 555 Flatlands Ave, 718-434-9876

Test 9 – Enter 123 in ID text box and Click GET button to search database for Customer THAT DOES NOT EXIST IN DATABASE:


Customer Not Found


Customer Managment Form

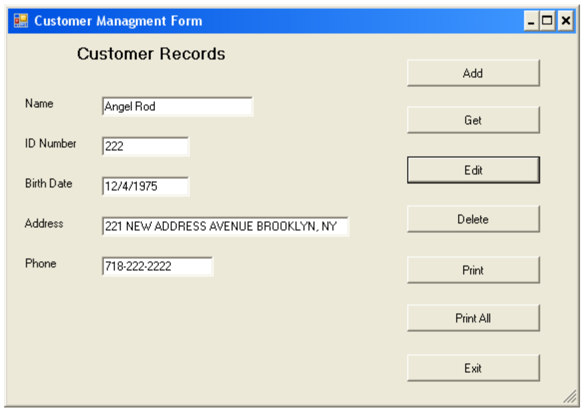
Customer Records

Name	
ID Number	
Birth Date	
Address	
Phone	

RESULTS OF TESTS 4 – 9:

- **We proved that Customer Form displayed properly.**
- **We searched for the first customer (111) and the last (555) and we successfully found the Customers**
- **We also successfully Tested PRINTING A CUSTOMER.**
- **We also tested for a Customer that DOES NOT EXIST and we got the correct response, a message box and the FORM was CLEARED!**
- **So we can now assume SEARCHING & PRINTING A CUSTOMER ARE BOTH WORKING.**

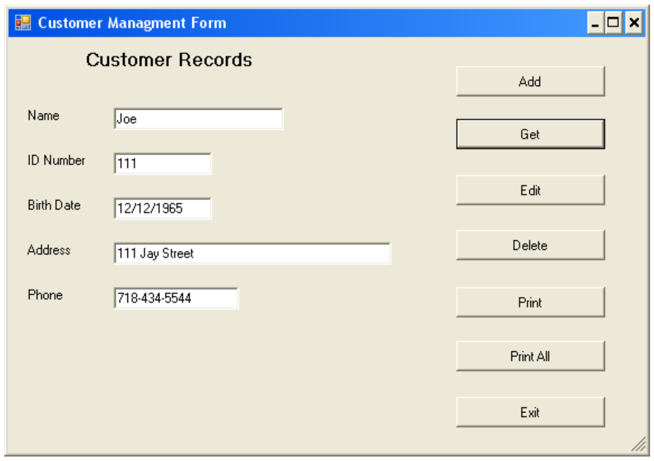
Test 10 – Enter 222 in ID text box and Click GET button to search database for Customer. MODIFY THE NAME, BIRTH DATE, ADDRESS, & PHONE fields. Then CLICK EDIT BUTTON to modify each PROPERTY:


Customer Managment Form

Customer Records

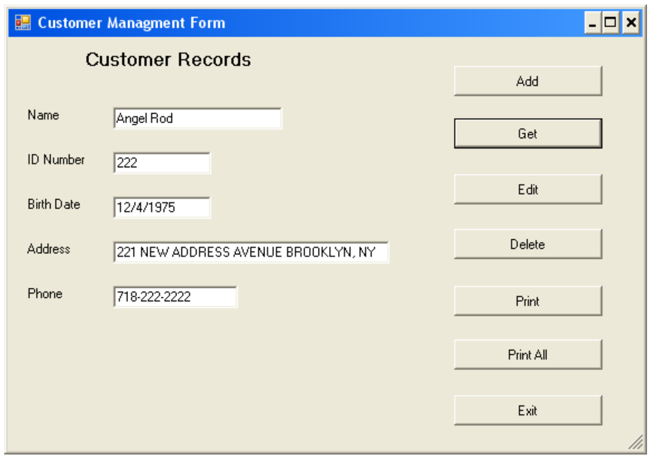
Name: Angel Rod
ID Number: 222
Birth Date: 12/4/1975
Address: 221 NEW ADDRESS AVENUE BROOKLYN, NY
Phone: 718-222-2222

Test 11 & 12 – Enter 111 in ID NUMBER text box and Click GET button to search database and display Customer 111 data. Then Enter 222 in ID NUMBER text box and Click GET button to search database for Customer 222 and VERIFY that DATA WAS MODIFIED for Customer 222:


Customer Managment Form

Customer Records

Name: Joe
ID Number: 111
Birth Date: 12/12/1965
Address: 111 Jay Street
Phone: 718-434-5544


Customer Managment Form

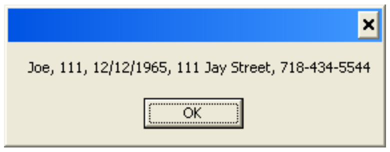
Customer Records

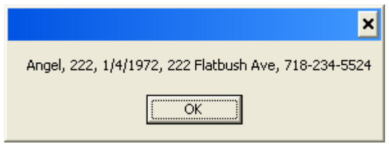
Name: Angel Rod
ID Number: 222
Birth Date: 12/4/1975
Address: 221 NEW ADDRESS AVENUE BROOKLYN, NY
Phone: 718-222-2222

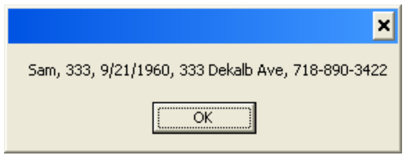
RESULTS OF TESTS 10 – 12:

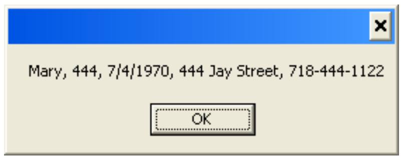
❖ **We proved that we can MOIDIFY a Customer, by changing a Customer's data, saving the record and then retrieving it again to prove that data was changed.**

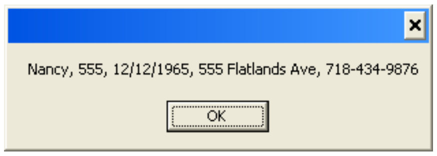
Test 13 – Click the PRINTALL BUTTON. You should have a Message Box for each OBJECT IN DATABASE:


Joe, 111, 12/12/1965, 111 Jay Street, 718-434-5544

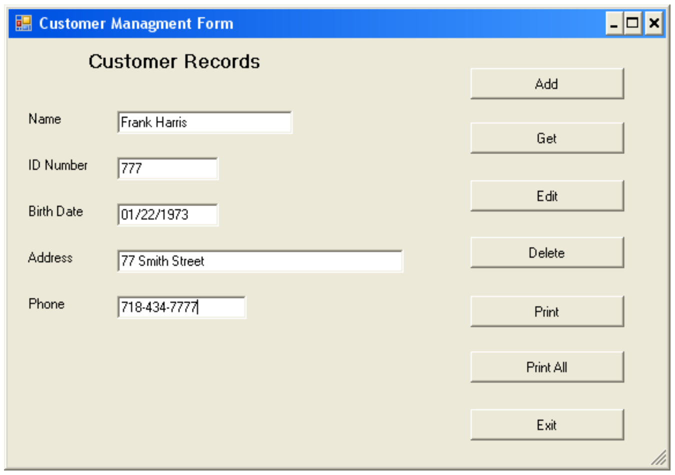

Angel, 222, 1/4/1972, 222 Flatbush Ave, 718-234-5524


Sam, 333, 9/21/1960, 333 Dekalb Ave, 718-890-3422


Mary, 444, 7/4/1970, 444 Jay Street, 718-444-1122

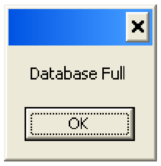

Nancy, 555, 12/12/1965, 555 Flatlands Ave, 718-434-9876

Test 14 – Enter NEW Customer data in the Form's controls to ADD a NEW CUSTOMER, as shown in screen below. Click ADD BUTTON. Note the resultant Message Box that DATABASE IS FULL:


Customer Managment Form

Customer Records

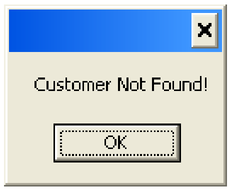
Name: Frank Harris
ID Number: 777
Birth Date: 01/22/1973
Address: 77 Smith Street
Phone: 718-434-7777


Database Full

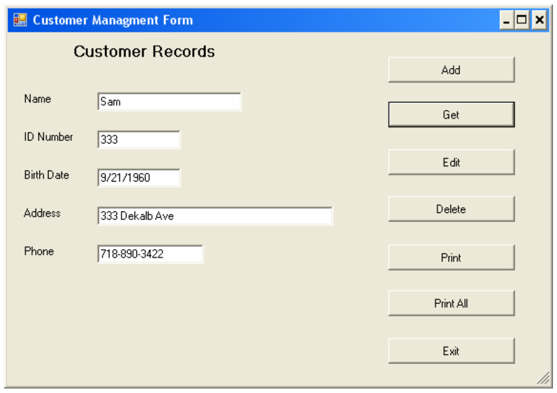
RESULTS OF TESTS 13 & 14:

- **In Test 13 we tested the PRINTALL functionality. All Customers were printed.**
- **In Test 14 we ATTEMPTED TO ADD A NEW CUSTOMER AND FAILED because DATABASE WAS FULL. This is expected since there are no EMPTY or NULL POINTERS AVAILABLE AT THIS TIME.**

Test 15– Enter 456 in ID text box and CLICK the DELETE BUTTON, to delete this record. Note the resultant Message Box that customer was NOT FOUND as expected:

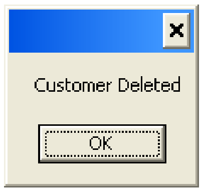

Customer Not Found!

Test 16 – Enter 333 in ID text box and Click <u>GET</u> button to search database for Customer 333. Now CLICK the DELETE BUTTON, to delete this record. Note the resultant Message Box that customer was removed:

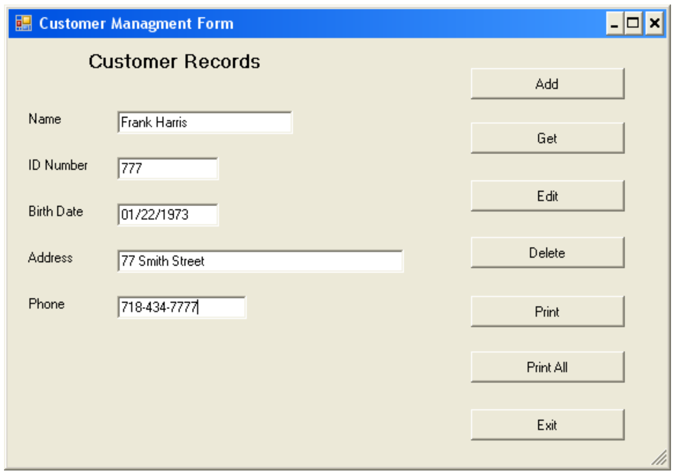

Customer Managment Form

Customer Records

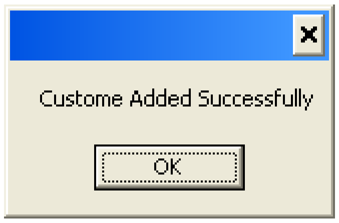
Name: Sam
ID Number: 333
Birth Date: 9/21/1960
Address: 333 Dekalb Ave
Phone: 718-890-3422


Customer Deleted

Test 17 – Again, enter NEW Customer data in the Form's controls to ADD a NEW CUSTOMER, as shown in screen below. Click ADD BUTTON. Note the resultant Message Box that Customer was added successfully:



Name: Frank Harris
ID Number: 777
Birth Date: 01/22/1973
Address: 77 Smith Street
Phone: 718-434-7777


Custome Added Successfully

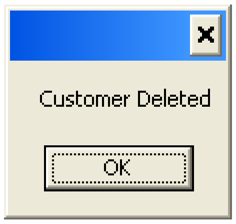
RESULTS OF TESTS 15-17:

- **In Test 15 we attempted to DELETE A CUSTOMER THAT DOES NOT EXIST IN DATABASE. The program responded successfully by displaying that CUSTOMER WAS NOT FOUND.**
- **In Test 16 we CREATED AN EMPTY SPACE IN DATABASE BY DELETING CUSTOMER 333. Now we have room in database and should be able to ADD ONE CUSTOMER TO DATABASE TO TEST ADD.**
- **In Test 17 WE WERE ABLE TO ADD A NEW CUSTOMER TO DATABASE SINCE <u>ONE</u> NULL POINTER WAS AVAILABLE.**

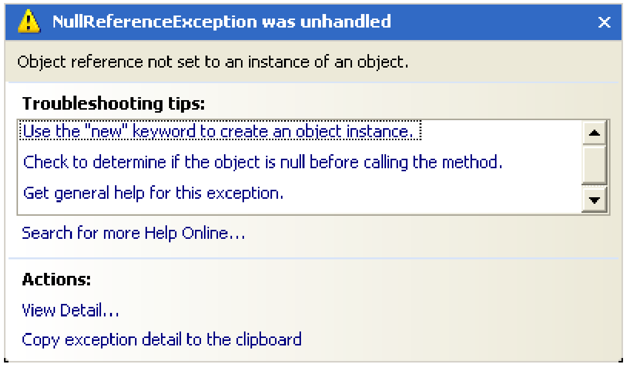
EXPOSING THE BUG IN THE PROGRAM

- We now exposed the BUG that exists in the program by deleting another Customer and searching or verifying the Customer does not exist!

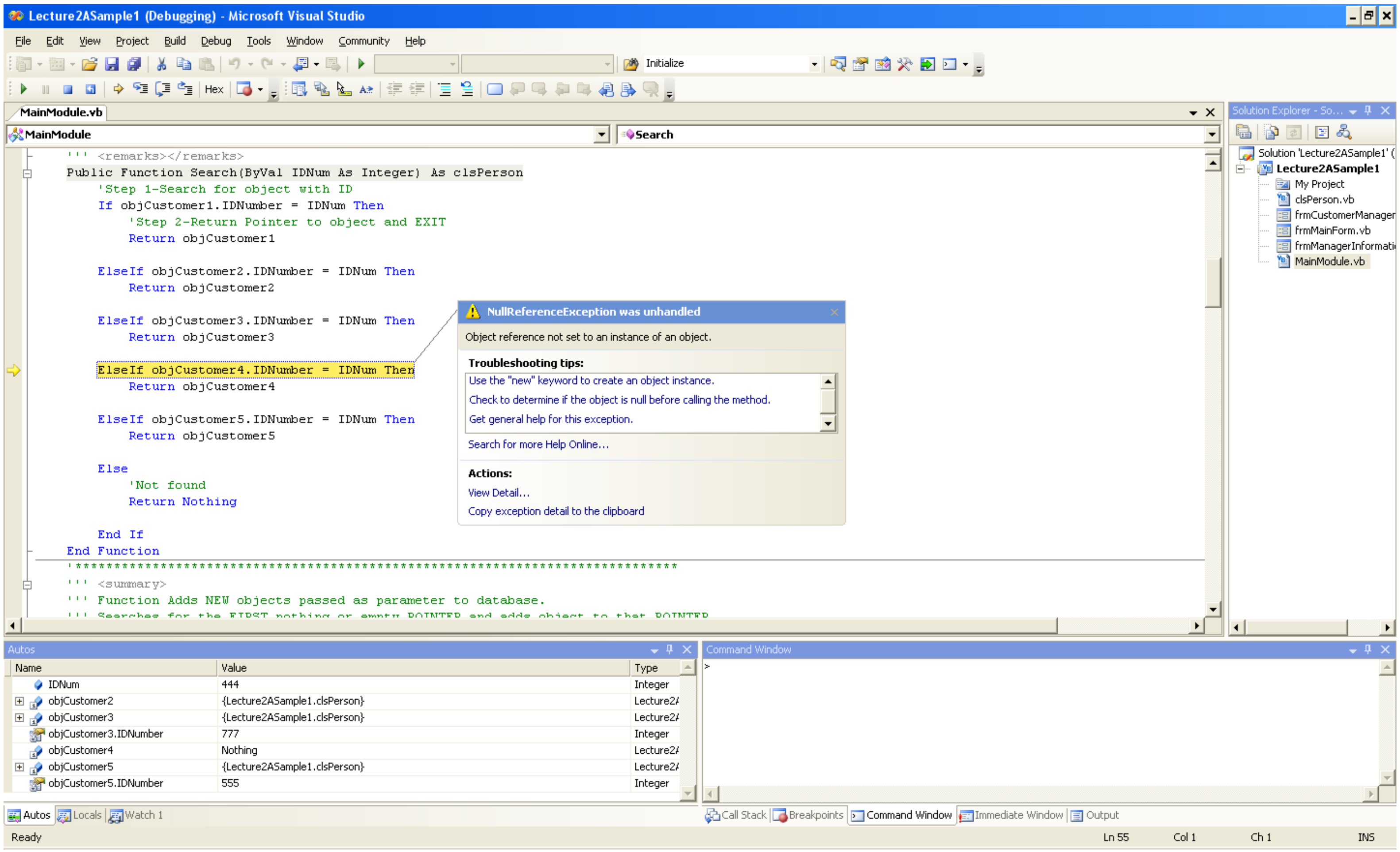
Test 18 – Enter 444 in ID text box and Click <u>GET</u> button to search database for Customer 444. Now CLICK the DELETE BUTTON, to delete this record. Note the resultant Message Box that customer was removed:


Customer Deleted

Test 19 – Enter 444 in ID text box and Click <u>GET</u> button to search database for Customer 444 to verify that it was REMOVED. THE RESULT IS THAT THE PROGRAM CRASHES AND THE FOLLOWING EXCEPTION IS RAISED:



```
NullReferenceException was unhandled

Object reference not set to an instance of an object.

Troubleshooting tips:
Use the "new" keyword to create an object instance.
Check to determine if the object is null before calling the method.
Get general help for this exception.
Search for more Help Online...

Actions:
View Detail...
Copy exception detail to the clipboard
```



Lecture2ASample1 (Debugging) - Microsoft Visual Studio
File Edit View Project Build Debug Tools Window Community Help
Initialize
Hex
MainModule.vb
MainModule
Search
Solution Explorer - So...
Solution 'Lecture2ASample1' (
Lecture2ASample1
My Project
clsPerson.vb
frmCustomerManager
frmMainForm.vb
frmManagerInformati
MainModule.vb

```
''' <remarks></remarks>
Public Function Search(ByVal IDNum As Integer) As clsPerson
    'Step 1-Search for object with ID
    If objCustomer1.IDNumber = IDNum Then
        'Step 2-Return Pointer to object and EXIT
        Return objCustomer1

    ElseIf objCustomer2.IDNumber = IDNum Then
        Return objCustomer2

    ElseIf objCustomer3.IDNumber = IDNum Then
        Return objCustomer3

    ElseIf objCustomer4.IDNumber = IDNum Then
        Return objCustomer4

    ElseIf objCustomer5.IDNumber = IDNum Then
        Return objCustomer5

    Else
        'Not found
        Return Nothing

    End If
End Function
'*****************************************************************************
''' <summary>
''' Function Adds NEW objects passed as parameter to database.
''' Searches for the FIRST nothing or empty POINTER and adds object to that POINTER
```

```
NullReferenceException was unhandled
Object reference not set to an instance of an object.
Troubleshooting tips:
Use the "new" keyword to create an object instance.
Check to determine if the object is null before calling the method.
Get general help for this exception.
Search for more Help Online...
Actions:
View Detail...
Copy exception detail to the clipboard
```

Autos

Name	Value	Type
IDNum	444	Integer
objCustomer2	{Lecture2ASample1.clsPerson}	Lecture2/
objCustomer3	{Lecture2ASample1.clsPerson}	Lecture2/
objCustomer3.IDNumber	777	Integer
objCustomer4	Nothing	Lecture2/
objCustomer5	{Lecture2ASample1.clsPerson}	Lecture2/
objCustomer5.IDNumber	555	Integer

RESULTS OF TESTS 18 &19:

- **In Test 18 WE DELETED CUSTOMER 444.**
- **In Test 19 we then attempted to SEARCH THE DATABASE for this Customer 444 to verify that it was REMOVED.**
- **At this point the program CRASHED AND WE RAISED AN EXECPTION.**
- **A matter of fact, with the EXCEPTION OF ADD operation, any operation we would have executed in the program such as GET, EDIT, REMOVE, PRINT & PRINT ALL would have CRASHED THE PROGRAM!**
- **WHY DID THE PROGRAM CRASH OR EXCEPTION RAISED? WHAT IS THE PROBLEM??????.**
- **SEE THE MESSAGE LISTED IN THE EXCEPTION FOR A CLUE!!!!!**

5.3.6 Sample Program 3 – Working With Forms & Custom Objects (VERSION 2)

Form Driven Application – Small Business Application Using Person Class

Problem statement:

- UPGRADE VERSION 1 TO TAKE INTO ACCOUNT THE BUG OR ISSUE WITH VERSION 1.
- The basic requirements are the same as the previous version 1, but we need to modify the program to take into account the issue found in version 1.

Issue with Version 1:

- In VERSION 1 we found that when we DELETED A CUSTOMER AND ATTEMPTED TO SEARCH, REMOVE, EDIT , PRINT, & PRINTALL THE PROGRAM CRASHED!!!.
- WHY?
    - ONCE WE REMOVED AN OBJECT, WE WERE LEFT WITH A POINTER POINTING TO NULL OR NOTHING!!!
    - BECAUSE OF THIS EVERY TIME WE ATTEMPTED TO QUERY OR ACCESS A PROPERTY OR CALL A METHOD IN THE POINTER THE PROGRAM CRASHED BECAUSE **WE CANNOT MANIPULATE OR ACCESS A POINTER WITH A NULL OR NOTHING!!!!!**

Solution to Version 1 BUG:

- To resolve the problem WE NEED TO TEST FOR A NOTHING OR NULL PRIOR TO ATTEMPTING TO QUERY OR MANIPULATE A PROPERTY OR METHOD IN A POINTER.
- As you can notice the MESSAGE DISPLAYED BY THE EXCEPTION stated the following:
    - Check to determine if the object is a NULL before calling the METHOD!!
- Therefore every time we attempt to WORK WITH AN OBJECT POINTER, we need to VERIFY IF OBJECT IS NOT EMPTY OR NULL!!
- You can use the following code to TEST POINTER prior to manipulation:

```
'Step 1-Ask POINTER if NOT POINTING to NOTHING OR NULL
If Not (objCustomer1 Is Nothing) Then

          'MANIPULATE OBJECT IN BODY OF IF

End If
```

- WHAT IS BEING DONE HERE IS THAT YOU ARE ONLY ACCESSING THE POINTER IF IS NOT POINTING TO NOTHING!!
- THIS IS REVERSE LOGIC, normally we would say “IF POINTER IS NOTHING DON’T ACCESS”, the REVERSE LOGIC is “IF POINTER IS NOT NOTHING, ACCESS”, note that both of these statements accomplish the same thing a POINTER IS ONLY MANIPULATED IF IS NOT POINTING TO A NOTHING OR NULL!

Application Architecture & OBJECT MODEL Requirements:

- Same as VERSION 1. Same class, same architecture.

Form Requirements

- Same as VERSION 1.

Module Requirements

- IS IN THE MODULE THAT CHANGES WILL TAKE PLACE! This is where we find the PROCESSING METHOD THAT DO THE WORK AND WHERE MANIPULATION OF THE DATABASE POINTERS TAKE PLACE.
- We need to MODIFY THE PROCESSING METHOD TO SKIP OR AVOID THE DATABASE POINTERS THAT ARE NOTHING!!!
- Because of this the IMPLEMENTATION of the PROCESSING METHODS WILL CHANGE.

HOW IT'S DONE:

Part I – Modify the Module as follows:

Standard Module:

Step 1: Add a Module to the Project and set its properties as show in table below:

Object	Property	Value
Module	Name Text	**MainModule** **MainModule**

Step 8: Add Module GLOBAL declarations (SAME AS VERSION 1):

- In the module, we will declare 5 *clsPerson* OBJECT POINTERS. These POINTERS will eventually point to objects which will represent our simulated DATABASE OF CUSTOMERS!
- In addition we create one complete OBJECT using parameterized constructor with data. This object represents a business employee manager.

```
Option Explicit On
Option Strict On

Module MainModule

'***************************************************************************
' GLOBAL VARIABLES & OBJECT DECLARATIONS SECTION
'***************************************************************************

'Declare  5 POINTERS to Customer Object (REPRESENT OUR SIMMULATED DATABASE)
Public objCustomer1, objCustomer2, objCustomer3, objCustomer4, objCustomer5 As clsPerson


'Declare & Create Public Object representing THE MANAGER employee, initialized with Data
Public objManager As clsPerson = New clsPerson("Alex Rod", 777, #12/12/1971#, _
                                               "192 East 8th, Brooklyn", "718-434-6677")
```

Step 2: Add Module INITIALIZE() Method declarations (SAME AS VERSION 1):

- Mow we begin to add PROCESSING METHODS TO THE MODULE that will do the work for the FORMS.
- The first method we implement is the INITIALIZE() method. This sub procedure creates 5 OBJECTS of the PERSON CLASS and assigns them to the 5 GLOBAL POINTERS.
- At this point the simulated DATABASE OF CUSTOMERS is not POPULATED WITH OBJECTS!

```
'*****************************************************************************
' METHOD DECLARATIONS
'*****************************************************************************
''' <summary>
''' Intended to execute at the start of the program. Can be used to perform
''' any initialization. In this case, to populate EACH POINTER with OBJECT at
''' start of the program. We are simply populating the database with data.
''' </summary>
''' <remarks></remarks>
Public Sub Initialize()
    'Create objects and initialize with data via paremterized constructor
    objCustomer1 = New clsPerson("Joe", 111, #12/12/1965#, "111 Jay Street", _
                                 "718-434-5544")

    objCustomer2 = New clsPerson("Angel", 222, #1/4/1972#, "222 Flatbush Ave", _
                                 "718-234-5524")

    objCustomer3 = New clsPerson("Sam", 333, #9/21/1960#, "333 Dekalb Ave", _
                                 "718-890-3422")

    objCustomer4 = New clsPerson("Mary", 444, #7/4/1970#, "444 Jay Street", _
                                 "718-444-1122")

    objCustomer5 = New clsPerson("Nancy", 555, #12/12/1965#, "555 Flatlands Ave", _
                                 "718-434-9876")

End Sub
```

Step 3: MODIFY SEARCH(ID) FUNCTION declarations:

- In this version, BEFORE INTERROGATING EACH POINTER we VERIFY AND SKIP the empty OR NULL POINTERS
- How it works:
  - Individual `If Not (objCustomer1 Is Nothing) Then` declarations are used to TEST EACH OBJECT PRIOR TO MANIPULATION
  - Then ANOTHER IF statement is used to TEST the OBJECT'S IDNUMBER PROPERTY against the ID NUMBER PROPERTY (PROPERTY = ID?), if a match is found THE PROPER RESPONSE IS RETURNED AND PROGRAM EXITED.
  - This process is repeated for every object in DATABASE.

```
'*******************************************************************************
''' <summary>
''' Function Searches the database for POINTER to object whose ID is a parameter
''' Skips the empty or NULL pointers before interrogating the object for the ID.
''' Returns POINTER to OBJECT and EXITS!
''' </summary>
''' <param name="IDNum"></param>
''' <returns></returns>
''' <remarks></remarks>
Public Function Search(ByVal IDNum As Integer) As clsPerson
    'Step 1-Ask POINTER1 is NOT POINTING to NOTHING OR NULL
    If Not (objCustomer1 Is Nothing) Then
        'Step 2-Search for object with ID
        If objCustomer1.IDNumber = IDNum Then
            'Step 3-If so, Return pointer to object & EXIT
            Return objCustomer1
        End If
    End If

    'Ask POINTER2 is NOT POINTING to NOTHING OR NULL
    If Not (objCustomer2 Is Nothing) Then
        If objCustomer2.IDNumber = IDNum Then
            Return objCustomer2
        End If
    End If

    'Ask POINTER3 is NOT POINTING to NOTHING OR NULL
    If Not (objCustomer3 Is Nothing) Then
        If objCustomer3.IDNumber = IDNum Then
            Return objCustomer3
        End If
    End If

    'Ask POINTER4 is NOT POINTING to NOTHING OR NULL
    If Not (objCustomer4 Is Nothing) Then
        If objCustomer4.IDNumber = IDNum Then
            Return objCustomer4
        End If
    End If

    'Ask POINTER5 is NOT POINTING to NOTHING OR NULL
    If Not (objCustomer5 Is Nothing) Then
        If objCustomer5.IDNumber = IDNum Then
            Return objCustomer5
        End If
    End If

    'Did not find object in search return a nothing
    Return Nothing

End Function
```

Step 4: Create ADD(object) FUNCTION declarations:

- THERE IS NO CHANGE REQUIRED IN THIS METHOD.
- THIS IS BECAUSE THIS METHOD IS ACTUALLY SEARHCING FOR A NULL OR NOTHING IN ORDER TO ADD A NEW OBJECT.
- How it works:
  - Nested **ElseIf** statement are used to ask EACH POINTER if is pointing to a NOTHING OR NULL.
  - During each **ElseIf**, if the POINTER IS NOTHING then parameter DATABASE POINTER is assigned to PARAMETER POINTER, thus adding the object to database.
  - A TRUE is returned and function EXITS.
  - ONLY AFTER ALL DATABASE POINTERS ARE TESTED AND NO NULLS ARE FOUND is a FALSE RETURNED.

```
'*****************************************************************************
''' <summary>
''' Function Adds NEW objects passed as parameter to database.
''' Searches for the FIRST nothing or empty POINTER and adds object to that POINTER.
''' Returns a TRUE When OBJECT added OR FALSE when no empty POINTERS and EXITS!!
''' </summary>
''' <param name="objE"></param>
''' <returns></returns>
''' <remarks></remarks>
Public Function Add(ByVal objE As clsPerson) As Boolean

    'Step 1-Ask if object is NULL
    If objCustomer1 Is Nothing Then
        'Step 2-If NULL, database POINTER = Paramter POINTER
        objCustomer1 = objE
        'Step 3-Return success & EXIT
        Return True

    ElseIf objCustomer2 Is Nothing Then
        objCustomer2 = objE
        Return True

    ElseIf objCustomer3 Is Nothing Then
        objCustomer3 = objE
        Return True

    ElseIf objCustomer4 Is Nothing Then
        objCustomer4 = objE
        Return True

    ElseIf objCustomer5 Is Nothing Then
        objCustomer5 = objE
        Return True

    Else
        'No space available
        Return False

    End If
End Function
```

Step 5: Create REMOVE(ID) FUNCTION declarations:

- In this version, BEFORE INTERROGATING EACH POINTER we VERIFY AND SKIP the empty OR NULL POINTERS
- How it works:
  - Individual `If Not (objCustomer1 Is Nothing) Then` declarations are used to TEST EACH OBJECT
  - Then ANOTHER IF statement is used to TEST the OBJECT'S IDNUMBER PROPERTY against the ID NUMBER PROPERTY (PROPERTY = ID?), if a match is found THE OBJECT IS REMOVED AND PROPER RESPONSE IS RETURNED AND PROGRAM EXITED.

```
'******************************************************************************
''' <summary>
''' Function Removes object from database by searching for OBJECT whose ID
''' is a parameter. Skips the empty pointers before interrogating the
''' object for the ID. When found, Removes object by setting POINTER TO NOTHING
''' Returns a TRUE When removed OR FALSE when OBJECT not found and EXITS!
''' </summary>
Public Function Remove(ByVal IDNum As Integer) As Boolean
    'Ask POINTER1 is NOT POINTING to NOTHING OR NULL
    If Not (objCustomer1 Is Nothing) Then
        'Step 2-Search for object with ID
        If objCustomer1.IDNumber = IDNum Then
            'Step 3-When found set to nothing
            objCustomer1 = Nothing
            'Step 4-Return OK & EXIT
            Return True
        End If
    End If

    'Ask POINTER2 is NOT POINTING to NOTHING OR NULL
    If Not (objCustomer2 Is Nothing) Then
        If objCustomer2.IDNumber = IDNum Then
            objCustomer2 = Nothing
            Return True
        End If
    End If

    'Ask POINTER3 is NOT POINTING to NOTHING OR NULL
    If Not (objCustomer3 Is Nothing) Then
        If objCustomer3.IDNumber = IDNum Then
            objCustomer3 = Nothing
            Return True
        End If
    End If

    'Ask POINTER4 is NOT POINTING to NOTHING OR NULL
    If Not (objCustomer4 Is Nothing) Then
        If objCustomer4.IDNumber = IDNum Then
            objCustomer4 = Nothing
            Return True
        End If
    End If

    'Ask POINTER5 is NOT POINTING to NOTHING OR NULL
    If Not (objCustomer5 Is Nothing) Then
        If objCustomer5.IDNumber = IDNum Then
            objCustomer5 = Nothing
            Return True
        End If
    End If

    'Did not find object in search return a nothing
    Return False

End Function
```

Step 6: Create PRINT(ID) FUNCTION declarations:

- In this version, BEFORE INTERROGATING EACH POINTER we VERIFY AND SKIP the empty OR NULL POINTERS
- How it works:
  - Individual `If Not (objCustomer1 Is Nothing) Then` declarations are used to TEST EACH OBJECT
  - Then ANOTHER IF statement is used to TEST the OBJECT'S IDNUMBER PROPERTY against the ID NUMBER PROPERTY (PROPERTY = ID?), if a match is found THE OBJECT PRINT() METHOD IS CALLED AND PROPER RESPONSE IS RETURNED AND PROGRAM EXITED.

```
'*******************************************************************************
''' <summary>
''' Function Prints object by searching for OBJECT whose ID is a parameter
''' Skips the empty pointers before interrogating the object for the ID.
''' When found, CALLS the PRINT() METHOD in the object
''' Returns a TRUE When printed OR FALSE when OBJECT not found and EXITS!
''' </summary>
Public Function Print(ByVal IDNum As Integer) As Boolean
    'Ask POINTER1 is NOT POINTING to NOTHING OR NULL
    If Not (objCustomer1 Is Nothing) Then
        'Step 1-Search for object with ID
        If objCustomer1.IDNumber = IDNum Then
            'Step 2-Call Object's Print Method
            objCustomer1.Print()
            'Step 3-Return OK & EXIT
            Return True
        End If
    End If

    'Ask POINTER2 is NOT POINTING to NOTHING OR NULL
    If Not (objCustomer2 Is Nothing) Then
        If objCustomer2.IDNumber = IDNum Then
            objCustomer2.Print()
            Return True
        End If
    End If

    'Ask POINTER2 is NOT POINTING to NOTHING OR NULL
    If Not (objCustomer3 Is Nothing) Then
        If objCustomer3.IDNumber = IDNum Then
            objCustomer3.Print()
            Return True
        End If
    End If

    'Ask POINTER2 is NOT POINTING to NOTHING OR NULL
    If Not (objCustomer4 Is Nothing) Then
        If objCustomer4.IDNumber = IDNum Then
            objCustomer4.Print()
            Return True
        End If
    End If

    'Ask POINTER2 is NOT POINTING to NOTHING OR NULL
    If Not (objCustomer5 Is Nothing) Then
        If objCustomer5.IDNumber = IDNum Then
            objCustomer5.Print()
            Return True
        End If
    End If

    'Did not find object in search return a nothing
    Return False

End Function
```

Step 7: Create PRINTALL() SUB declarations:

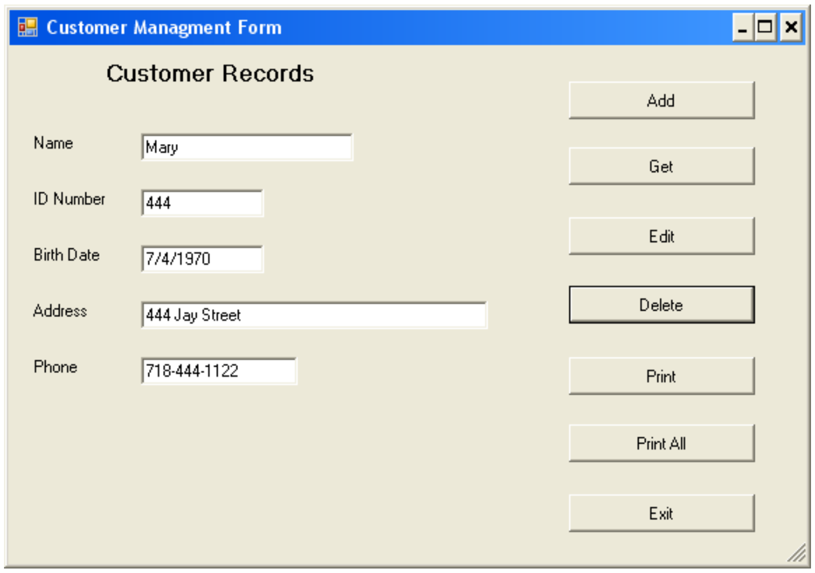
- In this version, BEFORE INTERROGATING EACH POINTER we VERIFY AND SKIP the empty OR NULL POINTERS
- How it works:
  - Individual `If Not (objCustomer1 Is Nothing) Then` declarations are used to TEST EACH OBJECT FOR NOTHING.
  - PRINT METHOD IS CALLED INSIDE THIS IF STATEMENT.

```
    '*****************************************************************************
    ''' <summary>
    ''' Sub Prints all objects in database by CALLING each object's PRINT() METHOD
    ''' Skips the empty pointers before CALLING the METHOD.
    ''' </summary>
    ''' <remarks></remarks>
    Public Sub PrintAll()

        'Ask POINTER1 is NOT POINTING to NOTHING OR NULL
        If Not (objCustomer1 Is Nothing) Then
            'Step 2-Call Object's Print Method
            objCustomer1.Print()
        End If

        'Ask POINTER2 is NOT POINTING to NOTHING OR NULL
        If Not (objCustomer2 Is Nothing) Then
            objCustomer2.Print()
        End If

        'Ask POINTER3 is NOT POINTING to NOTHING OR NULL
        If Not (objCustomer3 Is Nothing) Then
            objCustomer3.Print()
        End If

        'Ask POINTER4 is NOT POINTING to NOTHING OR NULL
        If Not (objCustomer4 Is Nothing) Then
            objCustomer4.Print()
        End If

        'Ask POINTER5 is NOT POINTING to NOTHING OR NULL
        If Not (objCustomer5 Is Nothing) Then
            objCustomer5.Print()
        End If

    End Sub

End Module
```

TESTING THE BUG ONCE AGAIN IN THE PROGRAM

- Let's test once again the BUG and see if the program crashes!

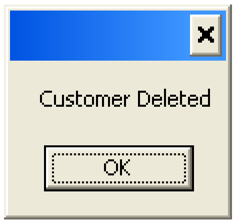
Test 1 – Enter 444 in ID text box and Click GET button to search database for Customer 444. Now CLICK the DELETE BUTTON, to delete this record. Note the resultant Message Box that customer was removed:



Customer Managment Form

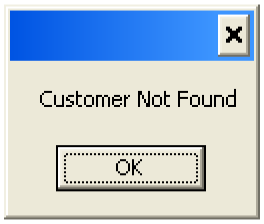
Customer Records

Name: Mary
ID Number: 444
Birth Date: 7/4/1970
Address: 444 Jay Street
Phone: 718-444-1122



Customer Deleted

Test 2 – Enter 444 in ID text box and Click GET button to search database for Customer 444 to verify that it was REMOVED. THE RESULT IS THAT THE PROGRAM INFORMS THE USER THAT THE CUSTOMER WAS REMOVED:



Customer Not Found

- **ISSUE WITH BUG WAS RESOLVED!!!**

Homework Assignment 3

- ❑ This program is an upgrade to Homework Assignment 2. Read and follow each of the following requirements. You will be graded based on all requirements being met.
- ❑ Copy HW2 to another Folder and rename it to HW3, including solution, project etc.
- ❑ Open the project and add the following Class and requirements:

Class Requirements:

I. ADD A CLASS MODULE and Create an Employee Class, this class should have the following Class requirements:

1) Private Data Members
   - UserName – ***String***
   - Password – ***String***
   - JobTitle **-** ***String***
2) Create the necessary Properties
3) Create a Default Constructor & Parameterized Constructor
4) Create the following Methods inside the class:
   - ***Function:*** **Authenticate (Arg1, Arg2)**
     - *Parameter Arguments:* This function should take two parameter arguments ByValue representing the Username & Password.
     - *Process:* Compares each of the parameters values to the private username & password variables and **returns** a **TRUE** if both of these values match, otherwise it returns a **FALSE**.

II. (**OPTIONAL BUT HIGHLY RECOMENDED**)Write a simple test driver program to test this class. In a Console Application, simply create one object and test each of the properties and methods, similar to Example in your notes.

Form & Module Requirements:

III. Modify your project to create objects of the *clsEmployee* class and use them. **Re-use the <u>Login Program in HW #2</u>** as follows:

Login Form:

1) The login form should keep all functionality from HW #2. The only exception is in the *GetUserInfoDisplayForm(u,p)*, the **password parameter should now be a *STRING***, based on the Employee Class Password being a String. Modify method accordingly.

Standard Module:

2) **DELETE** the ***UsernameDatabase*** & ***PasswordDatabase*** Arrays, which simulated our database or storage of employees.
3) In the Module create 5 individual Public EMPLOYEE Objects. These objects are now simulating the database of employees. **Do not use any list or arrays etc!** ONLY 5 individual single objects.
4) Create a <u>Sub procedure</u> named **Initialize()** that will populate the 5 public objects as follows: (Joe,111, Manager), (Angel, 222, Director), (Sam,333,Office Assistant), (Mary,444,Vice President), (Nancy,555,Secretary). This method must be called from *Main()* to populate the objects with data prior to any authentication or login method calls.
5) SUB MAIN() –THE CODE IN SUB MAIN SHOULD NOT CHANGE from HW2, THE ONLY EXCEPTION IS THAT YOU WILL NEED TO CALL Initialize() METHOD to populate the 5 OBJECTS prior to any processing in SUB MAIN().
6) In the Sub Main() procedure, keep the code from HW2 to control the program: loop, message box etc.
7) MODIFY ANY VARIABLE OR PARAMTER THAT USES PASSWORD FROM AN INTEGER TO STRING!
8) Note that most of the modifications will take place inside the Authenticate Method of the Module as shown below:

Module Authenticate Method:

9) **IMPORTANT!** Note that in HW2, you created a function named **<u>Authenticate()</u>** in the MODULE, keep this function as part of your program in the module. Do not confuse this module-level Authenticate with the internal ***Object.Authenticate()*** of the Employee Objects.
10) The main *Authenticate()* should SEARCH DATABASE and call each of the 5 object's internal *Authenticate()* to verify authenticity. The object's *Object.Authenticate()* Will RETURN a TRUE or FALSE results to the module **<u>Authenticate()</u>**, and the module **<u>Authenticate()</u>** should then return a TRUE or FALSE results to Sub Main(). Therefore, you will have two *Authenticate()* functions, one in the Module and one inside the class. DO NOT CONFUSE THEM, IT IS NOT THE SAME!
11) The Module Authenticate parameter syntax in the header, needs to be modify to reflect a password of String data type

- ❑ Due in two classes!