CS608 Lecture Notes

Visual Basic.NET Programming

Object-Oriented Programming

Inheritance (Part II)

(Part II of II)

(Lecture Notes 3B)

Prof. Abel Angel Rodriguez

Chapter 8 Inheritance

8.2 Inheritance Concepts (Cont)

8.2.4 Inheritance Features (Cont)

- Below is a listing of the inheritance topics already covered and those that we will go over in this document.
- Topics already covered:
    - Overloading Methods & Properties
    - Overriding Methods & Properties
- Remaining topics:
    - MyBase Keyword
    - Level of Inheritance
    - Constructors
    - Protected Scope
    - Abstract Base Class

8.2.5 MyBase Keyword

Introduction

- In the previous section we learned *Method Overrinding*, which allows us to completely replace a property or method of the Base class
- Method Overriding gives us the ability to completely replace the implementation of a base class method or property with a NEW or overridden method in the SubClass with the Same Name and signature.
- Method Overriding was implemented using the keywords ***Overridable*** and ***Overrides***
- We implemented Example 4 & 5 to show how Overriding works and we added Error Trapping code to handle the Exception generated by the Overridden *BirthDate* Property.

Introduction to MyBase Keyword

- The Keyword **MyBase** explicitly exposes the *Base Class* methods to the *Derived Classes*.
- Don't get confused, a derived Class automatically inherits the Public Base class feature but we could if we want use the keyword **MyBase** as well.
- For example:
  - In our previous examples we created a Base Class called *clsPerson* which contained the properties *Name*, *BirthDate* and *IDNumber* and a method named Print()
  - We derived from *clsPerson* a derived class named *clsEmployee* which inherited *Name*, *BirthDate* and *IDNumber* and added *HireDate* & *Salary* and a method named *PrintEmployee()* which called the Base class *Print()* as follows:

```
Public Sub PrintEmployee()
    'Call Base Class Method
    Print()

    'Display Derived Class Data
    MessageBox.Show("Printing Employee Data " _
    & mdHireDate & ", " & mdbSalary)

End Sub
```

  - Point here is that we automatically inherit *Print()* and can simply call it.
  - Nevertheless, if we wanted, we could have also used the Keyword **MyBase** to explicitly reference the Method *Print()* as follows:

```
Public Sub PrintEmployee()
    'Call Base Class Method Using MyBase Keyword
    MyBase.Print()

    'Display Derived Class Data
    MessageBox.Show("Printing Employee Data " _
    & mdHireDate & ", " & mdbSalary)

End Sub
```

  - OK we are really not gaining anything here, but just simply showing that using the Keyword **MyBase** we can explicitly reference Base Class Properties & Methods to achieve the same thing.

Application of the MyBase Keyword

- Now let's see were this keyword is important.
- We implemented Example 4 & 5 to show how Overriding works and we added Error Trapping code to handle the Exception generated by the Overridden *BirthDate* Property.
- There was one issue with Example 4 & 5 and Overriding the *BirthDate* Property.
- If you recall, we were FORCED to create a new *Private* Variable ***mdBirthDate*** in the Derived Class *clsEmployee* in order to store the New *BirthDate* Data
- We were forced to do this because the Base Class ***m_dBirthDate*** is private an inaccessible. More important we cannot call the Base Class Public *BirthDate* Property to access it's ***m_dBirthDate*** from within our NEW version of *BirthDate* Property since the compiler will get confused with which *Birthdate* Property you are referring to, the NEW **Overridden** *BirthDate* Property in the Derived Class or the inherited *BirthDate* Property in the Base Class. The compiler cannot tell, therefore you will have a run time error.
- Nevertheless we added this new BirthDate variable in the derived class and that was that.
- Well this is OK, but we were forced to create a new variable thus add more memory. Would it have been nice just to be able to call the Base Class BirthDate Property directly without confusing the compiler with the Overriden Birthdate Property of the subclass?
- This is where the Keyword **MyBase** comes into play.
- Instead of creating this additional ***mdBirthDate*** in the derived class *clsEmployee* to store the Birth Date data, we can simply explicitly call the Base Class BirthDate Property using the Keyword **MyBase** as follows:

```
MyBase.BirthDate
```

- Here the compiler WON'T get confused since we are explicitly telling it that the *BirthDate* Property we are referring to is the one in the Base Class and NOT the new Overridden one in the Derived Class.
- Now we can implement the Overridden *BirthDate* Property in the *clsEmployee* Class without the need to create the private variable ***mdBirthDate***.

Implementing MyBase Keyword

- To use the MyBase feature simply use when you desire to call the Base Class Methods & Properties directly.
- Remember that you automatic inherit the Public Methods & Properties, so the MyBase keyword is usually NOT necessary, but there will be times when you may wish to call Base Class Methods & Properties but the Overriden ones have the same name and the compiler will yield errors, in these situation use the keyword MyBase to explicitly tell the compiler that is the base class method version you want executed.

Example 6 – Example 5 using **MyBase** Keyword in Overridden Property & Methods

- Let's redo Example 5, this time removing the ***mdBirthDate*** private data from the derived class and using the ***MyBase*** Keyword.
- In this example we will prove the following:
  - As in Example 4 & 5, Overriding Properties & Methods, and handling exceptions thrown by program code.
  - **MyBase** Keyword can be used to explicitly call base class methods & properties.

Creating the Base Class

- Same as before, use the keyword ***Overridable*** to allow the *Birthdate* & *Print()* method to be overridden:

Example 6 (Base-Class):

- Declaring the base class:

```
Option Explicit On
Public Class clsPerson
    '*************************************************************************
    'Class Data or Variable declarations
    Private m_strName As String
    Private m_intIDNumber As Integer
    Private m_dBirthDate As Date

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

    Public Property IDNumber() As Integer
        Get
            Return m_intIDNumber
        End Get
        Set(ByVal Value As Integer)
            m_intIDNumber = Value
        End Set
    End Property
    'We allow Property to be overridden
    Public Overridable Property BirthDate() As Date
        Get
            Return m_dBirthDate
        End Get
        Set(ByVal Value As Date)
            m_dBirthDate = Value
        End Set
    End Property


    '*************************************************************************
    'Regular Class Methods

    'We allow Method to be overridden
    Public Overridable Sub Print()
        MessageBox.Show("Printing BASE CLASS Person Data " _
        & m_strName & ", " & m_intIDNumber & ", " & _
        m_dBirthDate)
    End Sub

End Cl
```

clsPerson	
strName:	String
intIDNumber:	Integer
dBirthDate:	Date
Name():	String
IDNumber():	Integer
BirthDate():	Date
Print()	

Creating Derived Class & Overriding the BirthDate Property

- We create the clsEmployees class and as usual we use the ***Inherit*** keyword in a class declaration to inherit from the *clsPerson* Class.
- We create a New *BirthDate* Property inside the *clsEmployee* Class and we use the keyword ***Overrides*** in the declaration of the property to always use this *BirthDate* Property instead of the Base *BirthDate* version.
- This new implementation of *BirthDate*, implements the policy that every employee must be at least 16 years old. If an employee is under 16, we need to throw an execption.
- This time we are NOT creating a private variable to store the Birth Date data since we are using the **MyBase** Keyword to explicitly call the Base Class *BirthDate* Property to give us access to the Base Class Private ***m_dBirthDate*** data.
- Lets look at the derived class *clsEmployee*:

Example 6 (SubClass):

- Declaring the SubClas:

```
Public Class clsEmployee
    Inherits clsPerson
    '*********************************************************************
    'Class Data or Variable declarations
    Private mdHireDate As String
    Private mdbSalary As Double

    '*********************************************************************
    'Property Procedures
    Public Property HireDate() As String
        Get
            Return mdHireDate
        End Get
        Set(ByVal Value As String)
            mdHireDate = Value
        End Set
    End Property

    Public Property Salary() As Integer
        Get
            Return mdbSalary
        End Get
        Set(ByVal Value As Integer)
            mdbSalary = Value
        End Set
    End Property

    'We Override the Birthdate Property
    Public Overrides Property BirthDate() As Date
        Get
            'Use Base Class Property
            Return MyBase.BirthDate
        End Get
        Set(ByVal Value As Date)
            'Test to verify that Employee meets age requirement
            If DateDiff(DateInterval.Year, Value, Now()) >= 16 Then

                'Use Base Class Property
                MyBase.BirthDate = Value
            Else
            Throw New System.Exception("Under Age Employee, an Employee must be 16 Years old")
            End If

        End Set
    End Property
```

clsEmployee	
dHireDate: dbSalary:	Date Double
HireDate(): Salary(): Name(String):	Date Double String
Print(X) PrintEmployee()	

Overriding the Print() Method

- Now we override the *Print()* Method using the keyword ***Overrides*** as we did in Examples 4 & 5.
- In this case, implemented the overridden *Print()* method differently. Here I take advantage that the Base Class already has a Print() method, so why not utilize it.
- Therefore I use the keyword ***MyBase*** to explicitly call the Base Class *Print()*, then I add any new features I want an so on.
- In the PrintEmployee() method we also make a call to a Print() method, but this time the compiler will automatically use the one from this class or the overridden one, so here we DON'T need to worry about the compiler getting confused.
- Lets continue our implementation of the class *clsEmployee*:

Example 4 (SubClass-(Cont)):

- Declaring the SubClass Methods:

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

   'NEW Overridden Method
    Public Overrides Sub Print()
       'Using MyBase to directly call the Base Class Print() Method
        MyBase.Print()

        'Adding NEW features inside this NEW overridden method
        MessageBox.Show("Implementing ADDITIONAL NEW IMPROVED Features for Birthdate"
                        & BirthDate)
    End Sub

    Public Sub PrintEmployee()
        'Call Overriden Print() Method to display Base Class Data
        Print()

        'Display Derived Class Data
        MessageBox.Show("Printing Employee Data " _
        & mdHireDate & ", " & mdbSalary)

    End Sub
End Class
```

Main Program

- Ok the Main program is still the same, we will continue to trap errors using the ***Try-Catch-Finally*** statement to satisfy the under 16 years old trap.
- For easy of explanation, I will NOT use a birth date that will force the error in this example.
- ***Main()*** test program:

Example 2 (Main Program):

- Driver Program for testing inheritance:

```
Option Explicit On
Module modMainModule
    'Declare & Create Public Person & Employee Objects
    Public objEmployee1 As clsEmployee = New clsEmployee()
    Public objEmployee2 As clsEmployee = New clsEmployee()
    Public objPerson As clsPerson = New clsPerson()

    Public Sub Main()
        'Begin Error Trapping section
        Try
            'Populating Person Object with Data
            With objPerson
                .Name = "Frank Lee"
                .IDNumber = 123
                .BirthDate = #4/23/1968#

            End With

            'Call Person Print Method to Execute Base Class Print()
            objPerson.Print()


            'Populating Employee Object with Data
            '(Note that BirthDate Property used is actually the overridden Version)
            With objEmployee1
                .Name = "Joe Smith"
                .IDNumber = 111
                .BirthDate = #1/2/1965#
                .HireDate = #5/23/2004#
                .Salary = 50000
            End With

            'Call Employee Print Method which Executes embedded Overridden Print()
            objEmployee1.PrintEmployee()


            'Populating Employee Object with Data
            '(Note that BirthDate Property used is actually the overridden Version)
            '(Also note that BirthDate = Date < 16, thus Error will be raised)
            With objEmployee2
                .Name = "Mary Johnson"
                .IDNumber = 444
                .BirthDate = #4/12/1970#
                .HireDate = #5/23/2004#
                .Salary = 30000
            End With

            'Call Employee Print Method which Executes embedded Overridden Print()
            objEmployee2.PrintEmployee()

            'End Error Trapping section & Begin Error Handling Section
        Catch objException As Exception
            MessageBox.Show(objException.Message)
        End Try

    End Sub
```

Explanation & Results of Main Program:

- When we execute the program, the following occurs:

1. **We create one Person Object and two Employee Objects:**

```
'Declare & Create Public Person & Employee Objects
Public objEmployee1 As clsEmployee = New clsEmployee()
Public objEmployee2 As clsEmployee = New clsEmployee()
Public objPerson As clsPerson = New clsPerson()
```

2. **We populate the Base Class Object data and call it's Print() Method to print Base Class data:**

```
'Populating Person Object with Data
With objPerson
    .Name = "Frank Lee"
    .IDNumber = 123
    .BirthDate = #4/23/1968#
End With

'Call Person Print Method to Execute Base Class Print()
objPerson.Print()
```

Results and Explanation:

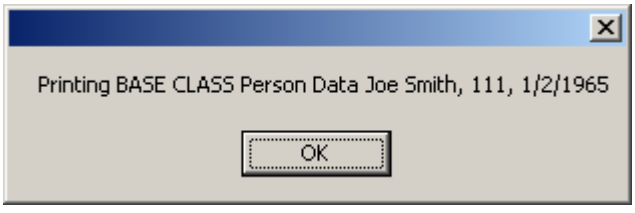
- Note that the Print() method for the Base is still operational as the resultant message box indicates, but only for Base Class Objects:

3. **We populate the first Employee Object using the Inherited properties from the Base Class, the Overridden Birthdate Property of the derived class and the remaining properties added by the Employee Class. In addition and we call it's PrintEmployee() Method to print the Overridden Base Class Print() method & Derived Class data:**

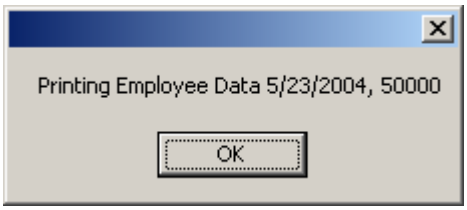
```
'Populating Employee Object with Data
'(Note that BirthDate Property used is actually the overridden Version)
With objEmployee1
    .Name = "Joe Smith"
    .IDNumber = 111
    .BirthDate = #1/2/1965#
    .HireDate = #5/23/2004#
    .Salary = 50000
End With

'Call Employee Print Method which Executes embedded Overridden Print()
objEmployee1.PrintEmployee()
```

Results and Explanation:

- The BirthDate Property used here is the Overridden Property not the one from the Base as we have shown in previous examples.
- The *PrintEmployee()* method first calls the Overridden Print() which calls the Base Class Print() method using the MyBase keyword as shown below



- The *PrintEmployees()* method then displays the derived class data:



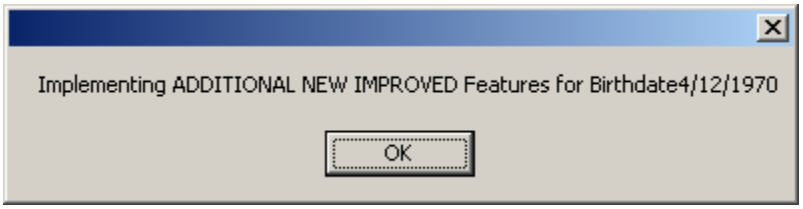
4. **We now populate the Second Object using two of the Inherited Properties from the Base Class, the Overridden BirthDate properties of the Employee Class and the other added Employee Class properties (Salary & HiredDate). In addition we call it's PrintEmployee() Method to print Overridden Base Class Print() method and Derived Class data:**

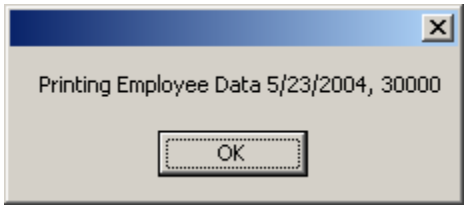
```
'Populating Employee Object with Data
'(Note that BirthDate Property used is actually the overridden Version)
'(Also note that BirthDate = Date < 16, thus Error will be raised)
With objEmployee2
    .Name = "Mary Johnson"
    .IDNumber = 444
    .BirthDate = #4/12/1970#
    .HireDate = #5/23/2004#
    .Salary = 30000
End With

'Call Employee Print Method which Executes embedded Overridden Print()
objEmployee2.PrintEmployee()
```

Results and Explanation:

- In this object we populated the Base Class *Name* and *IDNumber*. For the derived class we populate the Overridden *BirthDate* Property, *HireDate* & *Salary*.
- The NEW *BirthDate* Property has code that will test to make sure that the employee is over 16 years of age. Here the value chosen for the *BirthDate* Property will NOT trigger the exception; we tested that feature in Example 5.

- Again here the *PrintEmployee()* method first calls the Overridden *Print()* which calls the Base Class *Print()* method using the ***MyBase*** keyword followed by displaying the derived class data:





8.2.6 Shadows Keyword

Introduction

- In the previous section we learned *Method Overrinding*, which allows us to completely replace a property or method of the Base class
- With Method Overriding we were able to completely replace the implementation of a method or property in the Base Class NEW or overridden method in the **SubClass** with the Same Name and signature.
- To implement Method Overriding the Base Class must have the keywords ***Overridable*** and in the Sub Class version the key word ***Overrides***
- Permission to override the Base Class method is given by the Base Class designer via the keyword ***Overridable*** otherwise you cannot override the method.
- VB.NET provides another way of overriding a Base Class Method or Property, without the *Base Class* Method having the keyword ***Overridable***. This feature is called *Shadowing*, using the keyword ***Shadows***
- *Shadowing* means you don't need permission from the Base Class to override.
- This feature gives the Sub Class developer the freedom to change any method and alter the behavior of the Sub Class; therefore it no longer behaves like the Base Class.
- This is a radical deviation of the principles of inheritance and should be used with caution. Use Shadowing only when necessary.

Using the Shadows Keyword

- To implement shadow, simply create the new method or property in the Sub Class with the same name as the Base Class using the keyword ***Shadows***.

Example 7 – Shadows Keyword

- In this example we will prove the following:
  - **Shadows** Keyword can be used to replace the implementation of a property or method in the Base class with a new one in the Sub Class, without the consent of the Base Class.
  - To implement with leave the Base Class as is from the previous examples. On the other hand, we Shadow or replace the implementation of the Base Class Phone Property in the Employee Sub Class.
  - To prove that this works, we replace the implementation of the Employee Overriden Print method from the previous example by displaying the Base Class Public Properties. This will show that the public Phone property displayed is not the one from the Base Class but the new one that was Shadowed.

Creating the Base Class

- Same as before:

Example 7 (Base-Class):

- Declaring the base class:

```
Option Explicit On
Public Class clsPerson
    '*************************************************************************
    'Class Data or Variable declarations
    Private m_strName As String
    Private m_intIDNumber As Integer
    Private m_dBirthDate As Date
    Private m_strAddress As String
    Private m_strPhone As String
    Private m_intTotalItemsPurchased As Integer
    '*************************************************
    'Property Procedures
    Public Property Name() As String
        Get
            Return m_strName
        End Get
        Set(ByVal Value As String)
            m_strName = Value
        End Set
    End Property

    Public Property IDNumber() As Integer
        Get
            Return m_intIDNumber
        End Get
        Set(ByVal Value As Integer)
            m_intIDNumber = Value
        End Set
    End Property
    'We allow Property to be overridden
    Public Overridable Property BirthDate() As Date
        Get
            Return m_dBirthDate
        End Get
        Set(ByVal Value As Date)
            m_dBirthDate = Value
        End Set
    End Property

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

clsPerson	
strName:	String
intIDNumber:	Integer
dBirthDate:	Date
Name():	String
IDNumber():	Integer
BirthDate():	Date
Address():	String
Phone():	String
TotalItemsPurchase():	String
Print()	

Creating the Base Class

- Same as before:

Example 7 (Base-Class Cont):

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

    Public Property TotalItemsPurchased() As Integer
        Get
            Return m_intTotalItemsPurchased
        End Get
        Set(ByVal Value As Integer)
            m_intTotalItemsPurchased = Value
        End Set
    End Property


    '*********************************************************************
    'Regular Class Methods

    'We allow Method to be overridden
    Public Overridable Sub Print()
        MessageBox.Show("Printing BASE CLASS Person Data " _
        & m_strName & ", " & m_intIDNumber & ", " & _
        m_dBirthDate)
    End Sub

End Class
```

Creating Derived Class & Shadowing the Phone Property

- We create the clsEmployees class and as usual we use the ***Inherit*** keyword in a class declaration to inherit from the *clsPerson* Class.
- We create a New *Phone* Property inside the *clsEmployee* Class and we use the keyword ***Shadows*** in the declaration of the property to always use this *Phone* Property instead of the Base *Phone* version.
- This new implementation of *Phone*, implements simply appends the text "(Cell)" to the Get portion of the property. This really has no meaning and is done simply for teaching purpose to differentiate it from the Base Class Phone..
- We use the keyword **MyBase** to explicitly call the Base Class *BirthDate* Property to give us access to the Base Class Private ***m_dBirthDate*** data.
- Lets look at the derived class *clsEmployee*:

Example 7 (SubClass):

- Declaring the SubClas:

```
Public Class clsEmployee
    Inherits clsPerson
    '*********************************************************************
    'Class Data or Variable declarations
    Private mdHireDate As String
    Private mdbSalary As Double

    '*********************************************************************
    'Property Procedures
    Public Property HireDate() As String
        Get
            Return mdHireDate
        End Get
        Set(ByVal Value As String)
            mdHireDate = Value
        End Set
    End Property

    Public Property Salary() As Integer
        Get
            Return mdbSalary
        End Get
        Set(ByVal Value As Integer)
            mdbSalary = Value
        End Set
    End Property

    'Shadowing the Phone Property.  This new implementation
    'will override the Base Class.
    'To distinguish from the Base Class Phone
    'We will append the word (Cell)
    Public Shadows Property Phone() As String
        Get
            Return MyBase.Phone & "(Cell)"
        End Get
        Set(ByVal Value As String)
            MyBase.Phone = Value
        End Set
    End Property
```

clsEmployee	
dHireDate: dbSalary:	Date Double
HireDate(): Salary(): Phone():	Date Double String
Print(X) PrintEmployee()	

New Implementation of the Overridden Print() Method

- The *Print()* Method is overridden using the conventional keyword ***Overridable*** & ***Overrides*** combination.
- But the focus here is not the override, but a different implementation of *Print()* which displays the Properties of the classes.
- This is done to prove which Phone property is actually executing. By calling the Phone Property, the program needs to decide which *Phone* to print, the Base Class or the Sub Class? But since we are using Shadows, the one printed is the one in the Sub Class

Example 7 (SubClass-(Cont)):

- Declaring the SubClass Methods:

```
    'We Override the Birthdate Property
    Public Overrides Property BirthDate() As Date
        Get
            'Use Base Class Property
            Return MyBase.BirthDate
        End Get
        Set(ByVal Value As Date)
            'Test to verify that Employee meets age requirement
            If DateDiff(DateInterval.Year, Value, Now()) >= 16 Then

                'Use Base Class Property
                MyBase.BirthDate = Value
            Else
  Throw New System.Exception("Under Age Employee, an Employee must be 16 Years old")
            End If

        End Set
    End Property

    '*********************************************************************
    'Regular Class Methods

    'Different Implementation of the Overriden Print Method.
    'Attempting to Display the Base Class Properties.  All can be called
    'But the Phone.  Phone property displayed is not the Base but the
    'Shadowed version.  Nevertheless, the same applies to the Birthdate
    'Property which is overriden, but using the conventional overridable
    'keyword
    Public Overrides Sub Print()

            MessageBox.Show("Printing Employee Data " _
            & Name & ", " & IDNumber & ", " & _
            BirthDate & ", " & Phone)
    End Sub


    Public Sub PrintEmployee()
        'Call Overriden Print() Method to display Base Class Data
        Print()

        'Display Derived Class Data
        MessageBox.Show("Printing Employee Data " _
        & mdHireDate & ", " & mdbSalary)

    End Sub
```

Main Program

- Ok the Main program is still the same, we will continue to trap errors using the ***Try-Catch-Finally*** statement to satisfy the under 16 years old trap.
- But we will show that is the new implementation of Phone that is being executed and displayed since we will see the word (Cell) appended to the phone number when print is called since we shadowed the method in the Sub Class.
- ***Main()*** test program:

Example 2 (Main Program):

- Driver Program for testing inheritance:

```
Option Explicit On
Module modMainModule
    'Declare & Create Public Person & Employee Objects
    Public objEmployee1 As clsEmployee = New clsEmployee()
    Public objEmployee2 As clsEmployee = New clsEmployee()
    Public objPerson As clsPerson = New clsPerson()

    Public Sub Main()
        'Begin Error Trapping section
        Try
            'Populating Person Object with Data
            With objPerson
                .Name = "Frank Lee"
                .IDNumber = 123
                .BirthDate = #4/23/1968#
                .Phone = "718 260 1212"

            End With

            'Call Person Print Method to Execute Base Class Print()
            'Displaying the Base Class Phone as expected
            objPerson.Print()

            'Populating Employee Object with Data
            '(Note that Phone property is the one that was shadowed)
            With objEmployee1
                .Name = "Joe Smith"
                .IDNumber = 111
                .BirthDate = #1/2/1965#
                .HireDate = #5/23/2004#
                .Phone = "718 223 5454"
                .Salary = 50000
            End With

            'Call Employee Print Method which Executes embedded Overridden Print()
            'We will see the Shadowed Phone displayed with the (Cell) string
            'Appended, proving that the Sub Class method is executing.
            objEmployee1.PrintEmployee()

            'Populating Employee Object with Data
            With objEmployee2
                .Name = "Mary Johnson"
                .IDNumber = 444
                .BirthDate = #4/12/1990#
                .HireDate = #5/23/2004#
                .Phone = "718 555 2121"
                .Salary = 30000
            End With

            'Call Employee Print Method which Executes embedded Overridden Print()
            'The Shadowed Phone is displayed with the (Cell) string here as well.
            'Note that Because of the Birthdate rule this method may not execute.
            objEmployee2.PrintEmployee()

            'End Error Trapping section & Begin Error Handling Section
        Catch objException As Exception
            MessageBox.Show(objException.Message)
```

Explanation & Results of Main Program:

- When we execute the program, the following occurs:

1. **We create one Person Object and two Employee Objects:**

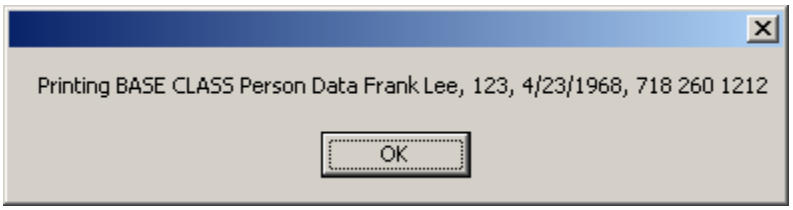
```
'Declare & Create Public Person & Employee Objects
Public objEmployee1 As clsEmployee = New clsEmployee()
Public objEmployee2 As clsEmployee = New clsEmployee()
Public objPerson As clsPerson = New clsPerson()
```

2. **We populate the Base Class Object data and call it's Print() Method to print Base Class data:**

```
'Populating Person Object with Data
    With objPerson
        .Name = "Frank Lee"
        .IDNumber = 123
        .BirthDate = #4/23/1968#
        .Phone = "718 260 1212"
    End With

'Call Person Print Method to Execute Base Class Print()
objPerson.Print()
```

Results and Explanation:

- Note that the Print() method prints the Base class properties including the Base Class Phone:

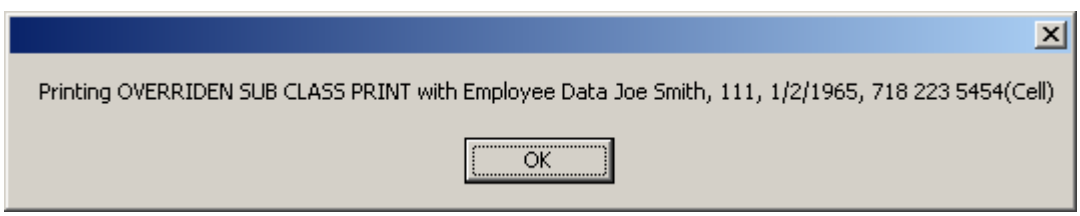


3. **We populate the first Employee Object using the Inherited properties from the Base Class, the Overridden Birthdate Property of the derived class and the remaining properties added by the Employee Class. In addition and we call it's PrintEmployee() Method to print the Overridden Base Class Print() method & Derived Class data:**

```
'Populating Employee Object with Data.  The phone property is set
    With objEmployee1
        .Name = "Joe Smith"
        .IDNumber = 111
        .BirthDate = #1/2/1965#
        .HireDate = #5/23/2004#
        .Phone = "718 223 5454"
        .Salary = 50000
    End With
    'Call Employee Print Method which Executes embedded Overridden Print()
    'The (Cell) string is appended to the phone, proving that the Shadowed
    'Phone property of the Sub Class is executed
    objEmployee1.PrintEmployee()
```

Results and Explanation:

- The Shadowed Phone property is displayed proving the Shadows process works.



- The remaining print code is executed:

4. **We now populate the Second Object with phone information as well as :**

```
'Populating Employee Object with Data
'(Note that BirthDate Property used is actually the overridden Version)
'(Also note that BirthDate = Date < 16, thus Error will be raised)
    With objEmployee2
        .Name = "Mary Johnson"
        .IDNumber = 444
        .BirthDate = #4/12/1990#
        .HireDate = #5/23/2004#
        .Phone = "718 555 2121"
        .Salary = 30000
    End With
'Call Employee Print Method which Executes The Shadowed Phone property
objEmployee2.PrintEmployee()
```

Results and Explanation:

- The value assigned to the *BirthDate* Property is chosen to raises an exception because of the under 16 years of age. Therefore, according to the code in Main() the exception is raised and the program termintes.

8.2.7 Constructors in Inheritance

Introduction

- So far we have with the features of inheritance we have covered, we can pretty much create applications that will utilize the benefits of inheritance. Nevertheless, we have one MAJOR problems, how do we intitialize the Base Class Data when we create a Derived Class Object?
- Here we need to review Contructors and see how they play a role in inheritance.
- As you recall, the **constructor** method is a special method that automatically 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.
- The Constructor Method has the following characteristics:

    - It is named Public Sub ***New***()
    - Automatically executes before any other methods are invoked in the class
    - We can overload the constructor method as we wish
    - Default Constructor is created by default but we can explicitly create it with our own initialization coed = New()
    - Parameterized Constructor take arguments and assign the private data with the parameters passed = New(ByVal par1 As Type, ByVal par2 As Type…..)

Simple Constructor

- Simple Constructors are those that we have been using so far, that is using default and Parameterized constructors inside the class.
- We have seen examples of these in HW & Exams.

Review - Implementing Regular or Simple Constructors

- The following example is a brief review of using simple Constructors.
- We will create the Base Class of previous examples but this time we will add a default & parameterized Constructor.
- We then create two object of the class, one which calls the default constructor and the other which takes arguments and invokes the parameterized constructor.

Example 8a – Simple Constructor Methods

Creating the Base Class

- Re-using the clsPerson class from the previous example:

Example 8a (Base-Class):

- Declaring the base class:

```
Option Explicit On
Public Class clsPerson
    '*********************************************************************
    'Class Data or Variable declarations
    Private m_strName As String
    Private m_intIDNumber As Integer
    Private m_dBirthDate As Date

    '*********************************************************************
    'Property Procedures
    Public Property Name() As String
        Get
            Return m_strName
        End Get
        Set(ByVal strTheName As String)
            m_strName = strTheName
        End Set
    End Property

    Public Property IDNumber() As Integer
        Get
            Return m_intIDNumber
        End Get
        Set(ByVal intTheID As Integer)
            m_intIDNumber = intTheID
        End Set
    End Property

    Public Property BirthDate() As Date
        Get
            Return m_dBirthDate
        End Get
        Set(ByVal dTheBDate As Date)
            m_dBirthDate = dTheBDate
        End Set
    End Property
```

clsPerson	
strName: intIDNumber: dBirthDate:	String Integer Date
Name(): IDNumber(): BirthDate():	String Integer Date
New() New(String, Integer , Date) Print()	

Example 8a (Base-Class Cont):

- Declaring the remaining base members:

```
    '*********************************************************************
    'Class Constructor Methods
    Public Sub New()
        Name = ""
        IDNumber = 0
        BirthDate = #1/1/1900#

       'Demostrate that constructor is actually executing
        MessageBox.Show("Base Class Default Constructor executed....")
    End Sub


   Public Sub New(ByVal strN As String, ByVal intIDNum As Integer, _
    ByVal bBDate As Date)
        Name = strN
        IDNumber = intIDNum
        BirthDate = bBDate

       'Demostrate that constructor is actually executing
        MessageBox.Show("Base Class Parametize Constructor executed....")
   End Sub


    '*********************************************************************
    'Regular Class Methods
    Public Sub Print()
        MessageBox.Show("Printing Person Data "
        & m_strName & ", " & m_intIDNumber & ", " & _
        m_dBirthDate)

    End Sub

End Class
```

Using Simple Constructor Methods

- Now let's look at the driver program.
- In this example we create two objects of the ***clsPerson*** class, one using the default constructor and the other the parameterized constructor.
- We then demonstrate that that objects are initialized to the values assigned in the *default* constructor. On the other hand the object created with arguments is initialized via the *parameterized* constructor.
- We the call the *print* method of each object to demonstrate that the constructor method did it's job
- ***Main()*** test program:

Example 8a (Main Program):

- Driver Program for testing inheritance:

```
Module modMainModule

    'Declare & Create Public Person Objects
    'Create Person objects that invokes default & Parametized Constructors
    Public objPerson1 As clsPerson = New clsPerson()
    Public objPerson2 As clsPerson = New clsPerson("Joe Smith", 111, #1/2/1965#)


    Public Sub Main()
        'DEMONSTRATING SIMPLE CONSTRUCTORS IN REGULAR CLASS
        'Call Person Object to display data initialized by default constructor
        objPerson1.Print()

        'Call Person Object to display data initialized by Paremetized constructor
        objPerson2.Print()


    End Sub


End Module
```

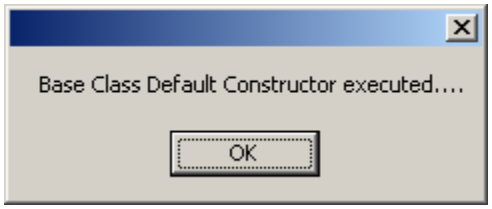
Explanation of Test program:

- When we execute the program, the following occurs:

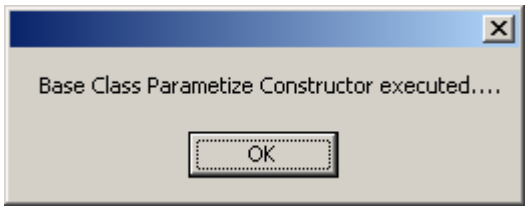
1. **We create two Person Objects, one using the defult constructor and the other the parameterized:**

```
Public objPerson1 As clsPerson = New clsPerson()
Public objPerson2 As clsPerson = New clsPerson("Joe Smith", 111, #1/2/1965#)
```

   - When we create each object, their constructor is executed, for example for the first object, the default constructor is executed. Since we placed a message box as a test to show us that the constructor executed, the message box will display:



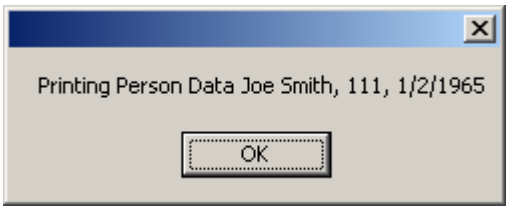
   - For the second object the parameterized constructor is automatically executed:



2. **We Person Class Print() Method to print each object's data to verify initialization values:**

   - Calling the Print() method of the first object will show that the private data was initialized to the default values:

   - Calling the Print() method of the second object will show that the private data was initialized to the parameterized values:



Summary of Results:

- Nothing unusual happened here, a class constructors were invoked when the object is created.
- So, constructors are operating as normal.

Constructor and Inheritance

- Now let's see what happens to constructors when we have a Base Class and Derived Classes.

Example 8b – Constructor Methods in Base and Derived Classes

Creating the Base Class

- Re-using the clsPerson class from the previous example:

Example 8b (Base-Class):

- Declaring the base class:

```
Option Explicit On
Public Class clsPerson
    '*********************************************************************
    'Class Data or Variable declarations
    Private m_strName As String
    Private m_intIDNumber As Integer
    Private m_dBirthDate As Date

    '*********************************************************************
    'Property Procedures
    Public Property Name() As String
        Get
            Return m_strName
        End Get
        Set(ByVal strTheName As String)
            m_strName = strTheName
        End Set
    End Property

    Public Property IDNumber() As Integer
        Get
            Return m_intIDNumber
        End Get
        Set(ByVal intTheID As Integer)
            m_intIDNumber = intTheID
        End Set
    End Property

    Public Property BirthDate() As Date
        Get
            Return m_dBirthDate
        End Get
        Set(ByVal dTheBDate As Date)
            m_dBirthDate = dTheBDate
        End Set
    End Property
```

clsPerson	
strName: intIDNumber: dBirthDate:	String Integer Date
Name(): IDNumber(): BirthDate():	String Integer Date
New() New(String, Integer , Date) Print()	

Example 8b (Base-Class):

- Declaring the remaining base members:

```
    '*********************************************************************
    'Class Constructor Methods
    Public Sub New()
        Name = ""
        IDNumber = 0
        BirthDate = #1/1/1900#

       'Demostrate that constructor is actually executing
        MessageBox.Show("Base Class Default Constructor executed....")
    End Sub


   Public Sub New(ByVal strN As String, ByVal intIDNum As Integer, _
    ByVal bBDate As Date)
        Name = strN
        IDNumber = intIDNum
        BirthDate = bBDate

       'Demostrate that constructor is actually executing
        MessageBox.Show("Base Class Parametize Constructor executed....")
   End Sub


    '*********************************************************************
    'Regular Class Methods
    Public Sub Print()
        MessageBox.Show("Printing Person Data "
        & m_strName & ", " & m_intIDNumber & ", " & _
        m_dBirthDate)

    End Sub

End Class
```

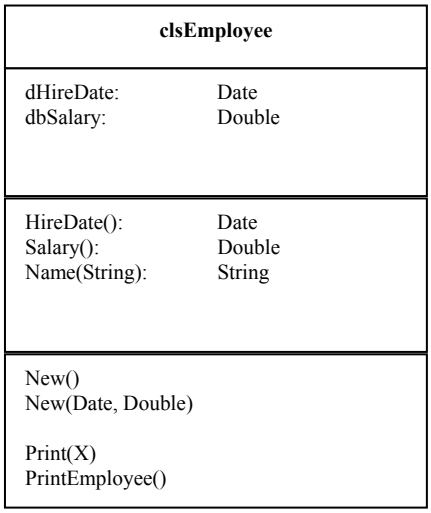
Derived or Sub Class

- The derived class has it' own constructors are well.
- We will use straight forward or simple constructor to demonstrate issues with the constructor implementation.
- Lets look at the derived class *clsEmployee*:

Example 8b (SubClass):

- Declaring the SubClas:

```
Option Explicit On
Public Class clsEmployee
    Inherits clsPerson
    '*********************************************************************
    'Class Data or Variable declarations
    Private mdHireDate As Date
    Private mdbSalary As Double

    '*********************************************************************
    'Property Procedures
    Public Property HireDate() As Date
        Get
            Return mdHireDate
        End Get
        Set(ByVal Value As Date)
            mdHireDate = Value
        End Set
    End Property

    Public Property Salary() As Double
        Get
            Return mdbSalary
        End Get
        Set(ByVal Value As Double)
            mdbSalary = Value
        End Set
    End Property
```



Example 8b (SubClass-(Cont)):

- Declaring the SubClass Methods:

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

   Public Sub New()
        HireDate = #1/1/1900#
        Salary = 0.0

       'Demostrate that constructor is actually executing
        MessageBox.Show("Sub Class Default Constructor executed....")
   End Sub


    Public Sub New(ByVal dHDate As String, ByVal dbSal As Double)
        HireDate = dHDate
        Salary = dbSal

       'Demostrate that constructor is actually executing
        MessageBox.Show("Sub Class Parametize Constructor executed....")
    End Sub


'*********************************************************************
    'Regular Class Methods
    Public Sub PrintEmployee()
        'Call Inherited Print Method to display Base Class values
        Print()

        'Now display Derived Class values
        MessageBox.Show("Printing Employee Data " _
        & mdHireDate & ", " & mdbSalary)

    End Sub

End Class
```

Using Constructor in Inheritance (Main)

- Now let's look at the driver program.
- In this example we create two objects of the ***clsEmployee*** class, one using the default constructor and the other the parameterized constructor.
- We then demonstrate that in the *clsEmployee* class objects, the default constructors of the derived class *clsEmployee* are invoked, but since *clsEmployee* is a child of *clsPerson*, the default constructor to person is also automatically invoked.
- On the other hand the object created with arguments initialize the parameterized constructor of the *clsEmployee* class, it DOES NOT in turn automatically invoke the parameterized constructor of the *clsPerson* Class, but instead automatically calls the default constructor of the Base class only!

    - These two situations are important. Base class **default** constructors were automatically called by the Sub Class default and parameterized constructors. But the Base class parameterized constructor was NOT called automatically!!!
    - This is a problem!!!

- We the call the print method of each object to demonstrate that the constructor method did its job.
- ***Main()*** test program:

Example 8b (Main Program):

- Driver Program for testing inheritance:

```
Module modMainModule

    'Create Employee objects that invokes default & Parametized Constructors
    Public objEmployee1 As clsEmployee = New clsEmployee()
    Public objEmployee2 As clsEmployee = New clsEmployee(#3/9/2004#, 30000)

    Public Sub Main()

        'DEMONSTRATING CONSTRUCTOR OPERATION IN SUB CLASSES
        'Call Employee Object to display data initialized by default constructor
        objEmployee1.PrintEmployee()

        'Call Employee Object to display data initialized by Paremetized constructor
        objEmployee2.PrintEmployee()

    End Sub
    End Sub


End Module
```

Explanation of Test program:

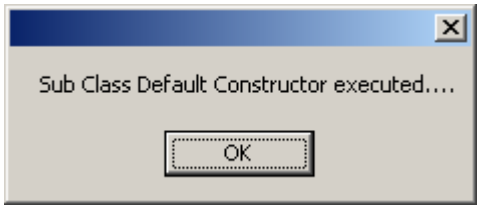
- When we execute the program, the following occurs:

1. **We create two Employee Objects, one using the defult constructor and the other the parameterized:**

```
'Create Employee objects that invokes default & Parametized Constructors
Public objEmployee1 As clsEmployee = New clsEmployee()
Public objEmployee2 As clsEmployee = New clsEmployee(#3/9/2004#, 30000)
```

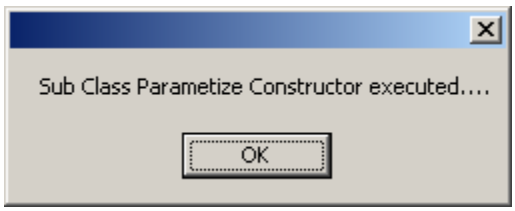
- When we create the first object, there are no arguments so the default constructor is executed. But since the *clsEmployee* class is derived from *clsPerson*, the *clsPerson* default constructor is invoked automatically by the *clsEmployee* class default constructor. Since we placed a message box as a test to show us that the Base Class default constructor is executed, the message box will display:

- Then of course the *clsEmployee* class default constructor continues to execute its code as shown by the message box:



- When we create the second object, the parameterized constructor of the *clsEmployee* Class is executed. Since the *clsEmployee* class is derived from *clsPerson*, you would expect that the *clsPerson* parameterized constructor will be invoked automatically by the *clsEmployee* class parameterized constructor, but it DOES NOT! Instead the DEFAULT constructor of the Base Class is invoked again! Note the messaged box showing that the Base Class default constructor is executed, the message box will display:

- Then of course the *clsEmployee* class parameterized constructor continues to execute its code as shown by the message box:



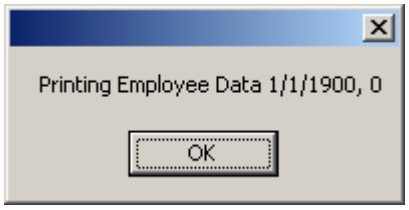
- NOTE here how the Base Class default constructor was automatically executed by the derived class clsEmployee default and parameterized constructor.
- The Base Class parameterized constructor was never called automatically!!!

2. **We call the Employee Class Print() Method to print each object's data to verify initialization values:**

```
'Call Employee Object to display data initialized by default constructor
objEmployee1.PrintEmployee()

'Call Employee Object to display data initialized by Paremetized constructor
objEmployee2.PrintEmployee()
```

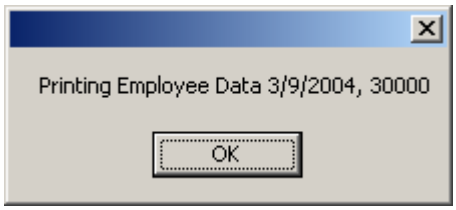
- Calling the `objEmployee1.PrintEmployee()` method of the first object will show that the Base class Print() is executed as expected and initialized to the default values:

- Now the `objEmployee1.PrintEmployee()` method continues to execute it's code and prints the employee information as expected:



- Calling the `objEmployee2.PrintEmployee()` method of the second object will show that the Base Class Print() is executed:

- Calling the `objEmployee2.PrintEmployee()` method of the second object will show the Derived Class data printed:



Summary of Results:

- Some interesting things happened in this example:

1) Creating default objects of the derived class the *default* Constructor automatically called the Base Class *default* constructor to unitize the Base Class data and then continue it's own execution to initialize it's default data.
2) On the other hand, the parameterized constructor also called the Base Class *default* constructor INSTEAD of calling the Parameterized one as we would expect. This means that the Base Class Parameterized constructor is never called automatically.
3) To summarize, for the parameterized Constructors, VB.NET cannot automatically make the call to the Parameterized constructor of the Base Class on our behalf. This means that the Base Class data cannot be initialized via the Parameterized constructor. THIS IS A BIG PROBLEM!!

Additional Discussion of Constructors Example 7b.

- To understand what is going on here, let's take a look at this issue in more detail.
- I am now going to re-do the default & Parameterized constructor of the derived class clsEmployee and explicitly show what VB.NET is automatically doing for us by writing the code ourselves.
- What VB.NET is doing for us is that it inserts the **MyBase** keyword in the background as follows:

Example 8b (SubClass-(Cont)):

- Declaring the SubClass Methods:

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

   Public Sub New()

        MyBase.New()

        HireDate = #1/1/1900#
        Salary = 0.0

       'Demostrate that constructor is actually executing
        MessageBox.Show("Sub Class Default Constructor executed....")
   End Sub


    Public Sub New(ByVal dHDate As String, ByVal dbSal As Double)

        MyBase.New()

        HireDate = dHDate
        Salary = dbSal

       'Demostrate that constructor is actually executing
        MessageBox.Show("Sub Class Parametize Constructor executed....")
    End Sub


'*********************************************************************
    'Regular Class Methods
    Public Sub PrintEmployee()
        'Call Inherited Print Method to display Base Class values
        Print()

        'Now display Derived Class values
        MessageBox.Show("Printing Employee Data " _
        & mdHireDate & ", " & mdbSalary)

    End Sub
```

- The final point here is whether we explicitly call the Base class default constructor by using **MyBase.New()** keyword, or let VB.NET automatically do it for us, we are still NOT able to execute the parameterize constructor:

```
        MyBase.New()
```

Solution to the Problem with Parameterized Constructor.

- The solution to the problem is to call the explicitly call the Base Class Parameterized constructor from the Sub Class Parameterized constructor.
- Lets look at the example again.

Example 8c – Handling Parameterized Constructor Methods in Base and Derived Classes

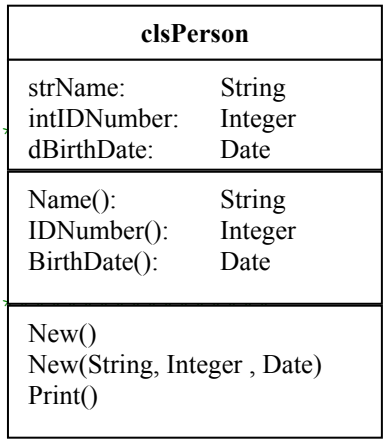
Creating the Base Class

- Re-using the clsPerson class from the previous example:

Example 8c (Base-Class):

- Declaring the base class:

```
Option Explicit On
Public Class clsPerson
    '*********************************************************
    'Class Data or Variable declarations
    Private m_strName As String
    Private m_intIDNumber As Integer
    Private m_dBirthDate As Date

    '*********************************************************
    'Property Procedures

      'SAME AS BEFORE.....
```



Example 8c (Base-Class):

- Declaring the remaining base members:

```
    '*********************************************************************
    'Class Constructor Methods
    Public Sub New()
        Name = ""
        IDNumber = 0
        BirthDate = #1/1/1900#

       'Demostrate that constructor is actually executing
        MessageBox.Show("Base Class Default Constructor executed....")
    End Sub


   Public Sub New(ByVal strN As String, ByVal intIDNum As Integer, _
    ByVal bBDate As Date)
        Name = strN
        IDNumber = intIDNum
        BirthDate = bBDate

       'Demostrate that constructor is actually executing
        MessageBox.Show("Base Class Parametize Constructor executed....")
   End Sub

    '*********************************************************************
    'Regular Class Methods
    Public Sub Print()
        MessageBox.Show("Printing Person Data " _
        & m_strName & ", " & m_intIDNumber & ", " & _
        m_dBirthDate)

    End Sub

End Class
```

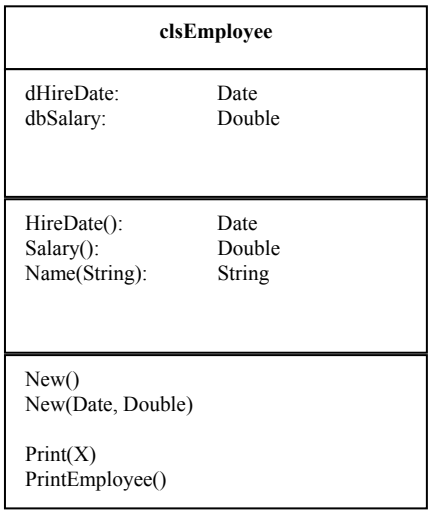
Derived or Sub Class Handling of Parameterized Constructor

- Now we see how the derived class must accommodate for the values to initiate the Base Class Parameterized constructor.
- Lets look at the derived class *clsEmployee*:

Example 8c (SubClass):

- Declaring the SubClas:

```
Option Explicit On
Public Class clsEmployee
    Inherits clsPerson
    '*********************************************************************
    'Class Data or Variable declarations
    Private mdHireDate As Date
    Private mdbSalary As Double

    '*********************************************************************
    'Property Procedures
    Public Property HireDate() As Date
        Get
            Return mdHireDate
        End Get
        Set(ByVal Value As Date)
            mdHireDate = Value
        End Set
    End Property

    Public Property Salary() As Double
        Get
            Return mdbSalary
        End Get
        Set(ByVal Value As Double)
            mdbSalary = Value
        End Set
    End Property
```



Example 8c (SubClass-(Cont)):

- Declaring the SubClass Methods:

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

   Public Sub New()

        MyBase.New()

        HireDate = #1/1/1900#
        Salary = 0.0

       'Demostrate that constructor is actually executing
        MessageBox.Show("Sub Class Default Constructor executed....")
   End Sub


    Public Sub New(ByVal strN As String, ByVal intIDNum As Integer, _
    ByVal bBDate As Date, ByVal dHDate As String, ByVal dbSal As Double)

        MyBase.New(strN, intIDNum, bBDate)

        HireDate = dHDate
        Salary = dbSal

        MessageBox.Show("Sub Class Parametize Constructor executed....")
    End Sub

'*********************************************************************
    'Regular Class Methods
    Public Sub PrintEmployee()
        'Call Inherited Print Method to display Base Class values
        Print()

        'Now display Derived Class values
        MessageBox.Show("Printing Employee Data " _
        & mdHireDate & ", " & mdbSalary)

    End Sub
```

Explanation:

- Note that the Parameterized constructor must contain in the heading the parameters to initialize the Base Class constructor as well as it's own data.

```
    Public Sub New(ByVal strN As String, ByVal intIDNum As Integer, _
    ByVal bBDate As Date, ByVal dHDate As String, ByVal dbSal As Double)
```

- In addition, we explicitly must explicitly call the Base Class Parameterized constructor with the arguments being passed to the Sub Class Parameterized constructor.

```
        MyBase.New(strN, intIDNum, bBDate)
```

Using Constructor in Inheritance (Main)

- Now let's look at the driver program.
- Note that now the second object has to include values for the Base Class Parameterized constructor as well.
- ***Main()*** test program:

Example 8c (Main Program):

- Driver Program for testing inheritance:

```
Module modMainModule

    'Create Employee objects that invokes default & Parametized Constructors
    Public objEmployee1 As clsEmployee = New clsEmployee()
    Public objEmployee2 As clsEmployee = New clsEmployee("Joe Smith", 111, #1/12/1965#, _
    #3/9/2004#, 30000)

Public Sub Main()

        'DEMONSTRATING CONSTRUCTOR OPERATION IN SUB CLASSES
        'Call Employee Object to display data initialized by default constructor
        objEmployee1.PrintEmployee()

        'Call Employee Object to display data initialized by Paremetized constructor
        objEmployee2.PrintEmployee()

    End Sub
    End Sub


End Module
```

Explanation of Test program:

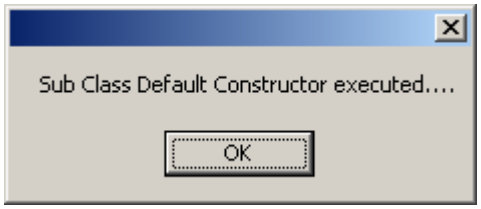
- When we execute the program, the following occurs:

1. **We create two Employee objects Objects, one using the defult constructor and the other the parameterized constructor. But this time we initialize the Parameterized Object with data for the Base Class:**

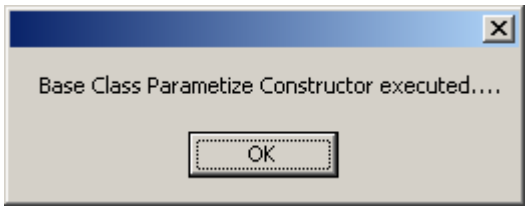
```
'Create Employee objects that invokes default & Parametized Constructors
Public objEmployee1 As clsEmployee = New clsEmployee()
Public objEmployee2 As clsEmployee = New clsEmployee("Joe Smith", 111, #1/12/1965#, _
#3/9/2004#, 30000)
```

   - When we create the first object, there are no arguments so the default constructor is executed and the *clsPerson* default constructor is invoked from the *clsEmployee* class default constructor. The message box will display:

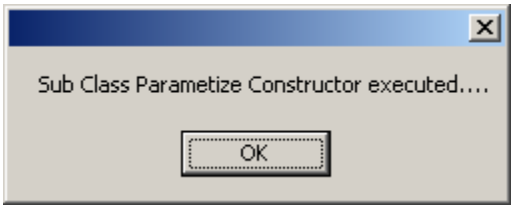
   - Then of course the *clsEmployee* class default constructor continues to execute its code as shown by the message box:



- When we create the second object, the parameterized constructor of the *clsEmployee* Class is executed. Since explicitly call the *clsPerson*, parameterized constructor in the Base Class, the message box will display:



- Then of course the *clsEmployee* class parameterized constructor continues to execute its code as shown by the message box:

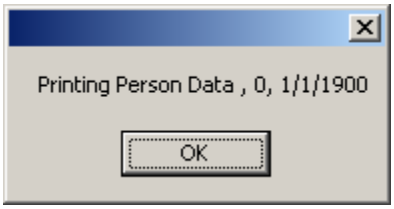


❖ NOTE here how the Base Class Parameterized constructor was executed by the derived class clsEmployee parameterized constructor as it should be.

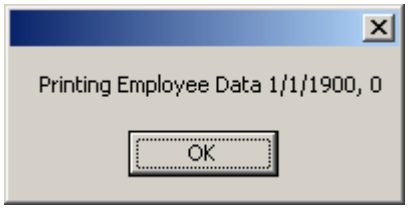
2. We call the Employee Class Print() Method to print each object's data to verify initialization values:

```
'Call Employee Object to display data initialized by default constructor
objEmployee1.PrintEmployee()

'Call Employee Object to display data initialized by Paremetized constructor
objEmployee2.PrintEmployee()
```

- Calling the `objEmployee1.PrintEmployee()` method of the first object will show that the Base class Print() is executed as expected and initialized to the default values:

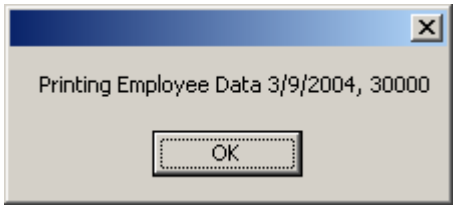


- Now the `objEmployee1.PrintEmployee()` method continues to execute it's code and prints the employee information as expected:



- Calling the `objEmployee2.PrintEmployee()` method of the second object will show that the Base Class Print() is executed. Note how this time the values stored in the base class are displayed, showing that the parameterized constructor did its job:

- Calling the `objEmployee2.PrintEmployee()` method of the second object will show the Derived Class data printed:



Summary of Results:

- By passing the Base class parameters and explicitly calling the Base Class Parameterized constructor as follows, we were able to initialize both the Base and Derived Class appropriately:

```
    Public Sub New(ByVal strN As String, ByVal intIDNum As Integer, _
    ByVal bBDate As Date, ByVal dHDate As String, ByVal dbSal As Double)

        MyBase.New(strN, intIDNum, bBDate)

        HireDate = dHDate
        Salary = dbSal

        MessageBox.Show("Sub Class Parametize Constructor executed....")
    End Sub
```

8.2.8 The Protected Scope

Introduction

- We saw how Sub or Derived Class automatically **inherit** all the Public **Methods** and **Properties** of the Base Class.
- This is also true for **Friend Methods** and **Properties** which are seen to everyone in the Project.
- But if you noticed, Private Methods, Data and Properties are NOT inherited or seen by the Sub Classes.
- Private data is only accessible to members of the class NOT it's children or anyone else.
- That is great that Sub Classes can automatically inherit the Public **Methods** and **Properties** of the Base Class, but what are we gaining, besides encapsulation and convenience, everyone else can also see or get the data?
- There are times when we would like the Sub Classes to have direct access to certain data and properties of the Base Class, but not allow anyone else. That is private for others, but Public for the Sub Classes.
- That is where the **Protected** keyword comes into play.
- The table below is a summary of the basic access specification for classes in general:

ACCESS SPECIFIER	ACCESSIBLE FROM ITSELF	ACCESSIBLE FROM DERIVED CLASS	ACCESSIBLE FROM OBJECTS OUTSIDE CLASS
Public	Yes	Yes	Yes
Protected	Yes	Yes	No
Private	Yes	No	No

- The Protected scope can be applied to Data variables, Sub , Functions and Properties.

Protected Variables

- We can use Protected when declaring variables that we want to make accessible to the Sub Classes, but private to everyone else.
- There are times when this is useful, but this is NOT recommended. Exposing variables to subclasses is typically not ideal.
- It is best to expose Properties using the Protected instead of the variables, this way we can enforce business rules on the Properties at the Base Class Level instead taking the chance that the author of the Sub Class will do it for you.
- In the next section we show example of the recommended way of using protected, that is in the Properties and methods of the Base Class only, NOT the data variables.

Example 9 – Protected Properties in Base Class

Creating the Base Class

- We now create the base class. We will Create a Protected SocialSecurityNumber Property that sets and gets the IDNumber variable.
- This Protected Property will be available to the Sub Classes only. No one else can call this property:

Example 9 (Base-Class):

- Declaring the base class:

```
Option Explicit On
Public Class clsPerson
    '*****************************************************
    'Class Data or Variable declarations
    Private m_strName As String
    Private m_intIDNumber As Integer
    Private m_dBirthDate As Date

    '*****************************************************
    'Property Procedures
    Public Property Name() As String
        Get
            Return m_strName
        End Get
        Set(ByVal strTheName As String)
            m_strName = strTheName
        End Set
    End Property

    Protected Property SocialSecurityNumber() As Integer
        Get
            Return m_intIDNumber
        End Get
        Set(ByVal intSSNum As Integer)
            m_intIDNumber = intSSNum
        End Set
    End Property

    Public Property BirthDate() As Date
        Get
            Return m_dBirthDate
        End Get
        Set(ByVal dTheBDate As Date)
            m_dBirthDate = dTheBDate
        End Set
    End Property
```

clsPerson	
strName:	String
intIDNumber:	Integer
dBirthDate:	Date
Name():	String
IDNumber():	Integer
BirthDate():	Date
New()	
New(String, Integer , Date)	
Print()	

Example 9 (Base-Class):

- Declaring the remaining base members:

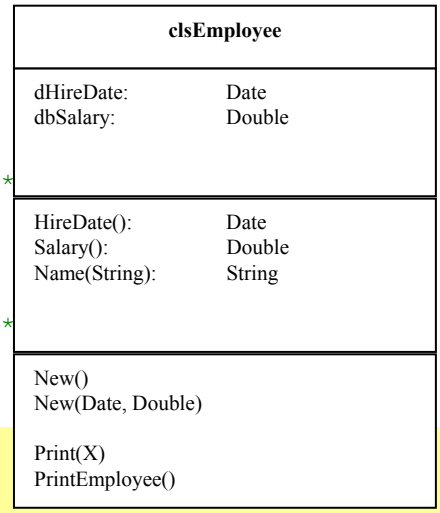
```
    '*********************************************************************
    'Class Constructor Methods
 Public Sub New()
        'Note that private data members are being initialized
        Name = ""
        SocialSecurityNumber = 0
        BirthDate = #1/1/1900#
 End Sub


 Public Sub New(ByVal strN As String, ByVal intIDNum As Integer, ByVal bBDate As Date)
        Name = strN
        SocialSecurityNumber = intIDNum
        BirthDate = bBDate
 End Sub


    '*********************************************************************
    'Regular Class Methods
 Public Sub Print()
        MessageBox.Show("Printing Person Data " _
        & m_strName & ", " & m_intIDNumber & ", " & _
        m_dBirthDate)

 End Sub

End Class
```

Derived or Sub Class

- The derived class has it' own constructors are well.
- We will use straight forward or simple constructor to demonstrate issues with the constructor implementation.
- Lets look at the derived class *clsEmployee*:

Example 9 (SubClass):

- Declaring the SubClas:




```
Option Explicit On
Public Class clsEmployee
    Inherits clsPerson
    '*************************************************
    'Class Data or Variable declarations
    Private mdHireDate As Date
    Private mdbSalary As Double

    '*************************************************
    'Property Procedures

    'Calling Protected Property from Base Class
    Public Property IDNumber() As Integer
        Get
            Return SocialSecurityNumber
        End Get
        Set(ByVal intTheID As Integer)
            SocialSecurityNumber = intTheID
        End Set
    End Property


    Public Property HireDate() As Date
        Get
            Return mdHireDate
        End Get
        Set(ByVal Value As Date)
            mdHireDate = Value
        End Set
    End Property

    Public Property Salary() As Double
        Get
            Return mdbSalary
        End Get
        Set(ByVal Value As Double)
            mdbSalary = Value
        End Set
    End Property
```

Example 9 (SubClass-(Cont)):

- Declaring the SubClass Methods:

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

   Public Sub New()

        MyBase.New()
        HireDate = #1/1/1900#
        Salary = 0.0
   End Sub


    Public Sub New(ByVal strN As String, ByVal intIDNum As Integer, _
    ByVal bBDate As Date, ByVal dHDate As String, ByVal dbSal As Double)

        MyBase.New(strN, intIDNum, bBDate)
        HireDate = dHDate
        Salary = dbSal
    End Sub

'*********************************************************************
    'Regular Class Methods
    Public Sub PrintEmployee()
        'Call Inherited Print Method to display Base Class values
        Print()

        'Now display Derived Class values
        MessageBox.Show("Printing Employee Data " _
        & mdHireDate & ", " & mdbSalary)

    End Sub

End Class
```

Calling Protected Base Class Member from Sub Class Public Property (Main)

- Now let's look at the driver program.
- In this example we create two objects of the ***clsEmployee*** class and one object of the Base Class **clsPerson**.
- The object of the clsPerson class will be used to demonstrate that we cannot call the Protected member since it is Private to everyone else and only available to the Sub Classes.
- ***Main()*** test program:

Example 9 (Main Program):

- Driver Program for testing inheritance:

```
Module modMainModule
  'Create Employee objects that invokes default & Parametized Constructors
  Public objPerson As clsPerson = New clsPerson()
  Public objEmployee1 As clsEmployee = New clsEmployee()
  Public objEmployee2 As clsEmployee = New clsEmployee("Joe Smith", 111, #1/12/1965#, _
    #3/9/2004#, 30000)

    Public Sub Main()

     'YOU CANNOT CALL THE FOLLOWING PROPERTY SINCE IT IS PRIVATE!!!
     'objPerson.SocialSecurityNumber = 1123507865

     'FOR EMPLOYEE OBJECTS ONLY THE SSNUMBER IS AVAILABLE THROUGH THE PROPERTY IDNUMBER
        With objEmployee1
            .Name = "Angel Rodriguez"
            .BirthDate = #5/12/1972#
            .IDNumber = 1123507865
            .HireDate = #7/8/2004#
            .Salary = 75000
        End With

        'Call Employee Object to display data of Employee1
        objEmployee1.PrintEmployee()

        'Call Employee Object to display data initialized by Paremetized constructor
        objEmployee2.PrintEmployee()

    End Sub

End Module
```

Explanation of Test program:

- When we execute the program, the following occurs:

1. **We create three Objects as follows:**

```
Public objPerson As clsPerson = New clsPerson()
Public objEmployee1 As clsEmployee = New clsEmployee()
Public objEmployee2 As clsEmployee = New clsEmployee("Joe Smith", 111, #1/12/1965#, _
  #3/9/2004#, 30000)
```

- We show that if we attempt to use the Protected member of the Person Class we will get a compiler error: :

```
'YOU CANNOT CALL THE FOLLOWING PROPERTY SINCE IT IS PRIVATE!!!
'objPerson.SocialSecurityNumber = 1123507865
```

- We then show that we populate the objEmployee1 members via the properties including the IDNumber which got it's data from the protected Base Class SocialSecurityNumber Property:

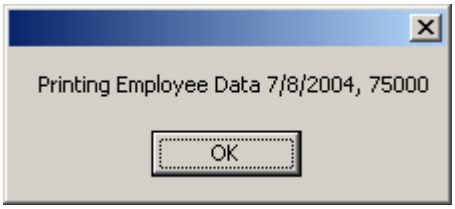
```
With objEmployee1
    .Name = "Angel Rodriguez"
    .BirthDate = #5/12/1972#
    .IDNumber = 1123507865
    .HireDate = #7/8/2004#
    .Salary = 75000
End With
```

2. **We call the Employee Class Print() Method to print each object's data to verify initialization values:**

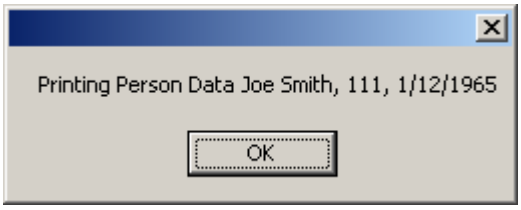
```
'Call Employee Object to display data of Employee1
objEmployee1.PrintEmployee()

'Call Employee Object to display data initialized by Parameterized constructor
objEmployee2.PrintEmployee()
```

- Calling the `objEmployee1.PrintEmployee()` method of the first object will show that the Base class Print() is executed populated values are shown:

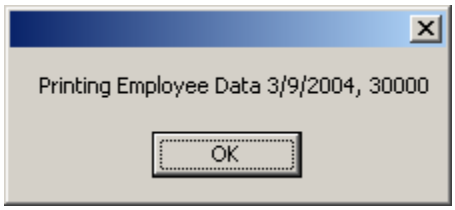
- Now the `objEmployee1.PrintEmployee()` method continues to execute it's code and prints the employee information as expected:



- Calling the `objEmployee2.PrintEmployee()` method of the second object will show that the Base Class Print() is executed:



- Calling the `objEmployee2.PrintEmployee()` method of the second object will show the Derived Class data printed:



Summary of Results:

- In this example we proved the following:

1) Using Protected scope for Property of the Base Class
2) Protected members can only be seen by the Sub Classes. They are private for everyone else.

8.2.9 MustInherit & MustOverride Keywords

MustInherit Keyword

- From what we have learned of Inheritance, we can create Base Classes and derived Sub Classes.
- In addition we can create Objects of the Sub or Derived Classes as well as the Base Class.
- But, there are circumstances when we may want to create a class such that it can only be used as a Base Class ONLY!
- This means that we CANNOT CREATE OBJECTS from this class. It MUST be used as a Base Class ONLY!
- To implement this we need declare the Base Class using the Keyword **MustInherit**.
- Once Base Class is declared with keyword MustInherit, we can NEVER CREATE OBJECTS of the Base Class.
- This is so strict that you will not be able to see the Base Class in the list of classes when making declarations of object.
- The syntax for using this keyword is:

```
'Class Header
Public MustInherit Class BaseClassName

    Data Definitions

    Properties Definitions

    Methods

End Class
```

Example:

- **Creating a MustInherit Base Class:**

- Creating Base Class *Products* using MustInherit keyword:

```
Public MustInherit Class Products
'Properties,
'Methods
' Event-Procedures
End Class
```

- Creating an Sub Class *VideoTape* from Base class *Product*:

```
Public Class VideoTape
        Inherits Product
        'Properties,
        'Methods
        ' Event-Procedures

End Class
```

- Declaring Object of Sub Class *VideoTape*:

```
Dim objVideosForSale As New VideoTape

'The following statement will be illegal!!!
Dim objTemProduct As New Products   '## Illegal ##
```

Example 10 – MustInherit Base Class

Creating the Base Class

- We now create the base class.
- We will use the keyword **MustInherit**. This will not allow the creation of objects of this Base Class:

Example 10 (Base-Class):

- Declaring the base class:

```
Option Explicit On
'Declare Class for MustInherit
Public MustInherit Class clsPerson
    '*********************************************************************
    'Class Data or Variable declarations
    Private m_strName As String
    Private m_intIDNumber As Integer
    Private m_dBirthDate As Date

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

    Public Property IDNumber() As Integer
        Get
            Return m_intIDNumber
        End Get
        Set(ByVal Value As Integer)
            m_intIDNumber = Value
        End Set
    End Property
    'We allow Property to be Overridden
    Public Overridable Property BirthDate() As Date
        Get
            Return m_dBirthDate
        End Get
        Set(ByVal Value As Date)
            m_dBirthDate = Value
        End Set
    End Property
```

clsPerson	
strName:	String
intIDNumber:	Integer
dBirthDate:	Date
Name():	String
IDNumber():	Integer
BirthDate():	Date
New()	
New(String, Integer , Date)	
Print()	

Example 10 (Base-Class):

- Declaring the remaining base members:

```
    '*********************************************************************
    '*********************************************************************
    'Class Constructor Methods
    Public Sub New()
        'Note that private data members are being initialized
        m_strName = ""
        m_intIDNumber = 0
        m_dBirthDate = #1/1/1900#
    End Sub


Public Sub New(ByVal strN As String, ByVal intIDNum As Integer, ByVal bBDate As Date)
  'Note that we are NOT using the private data but the Property Procedures instead
        Name = strN
        IDNumber = intIDNum
        BirthDate = bBDate
End Sub

'*********************************************************************
'Regular Class Methods
'We allow Method to be Overridden
Public Overridable Sub Print()
        MessageBox.Show("Printing BASE CLASS Person Data " _
        & m_strName & ", " & m_intIDNumber & ", " & _
        m_dBirthDate)
End Sub
```

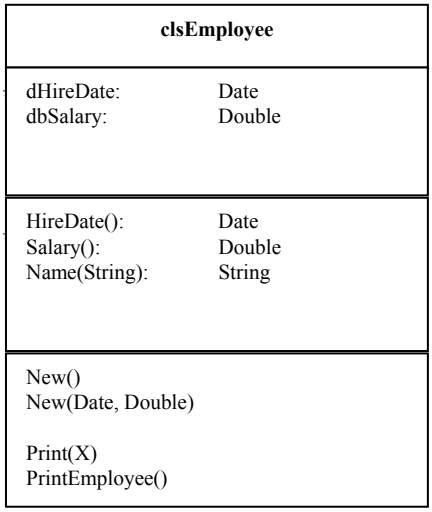
Derived or Sub Class

- Lets look at the derived class *clsEmployee*:

Example 10 (SubClass):

- Declaring the SubClass:

```
Option Explicit On
Public Class clsEmployee
    Inherits clsPerson
    '*************************************************
    'Class Data or Variable declarations
    Private mdHireDate As Date
    Private mdbSalary As Double

    '*************************************************
    'Property Procedures
    Public Property HireDate() As Date
        Get
            Return mdHireDate
        End Get
        Set(ByVal Value As Date)
            mdHireDate = Value
        End Set
    End Property

    Public Property Salary() As Double
        Get
            Return mdbSalary
        End Get
        Set(ByVal Value As Double)
            mdbSalary = Value
        End Set
    End Property
```



Example 10 (SubClass-(Cont)):

- Declaring the SubClass Methods:

```
'*********************************************************************
'Default Constructor Using MyBase to invoke Base Class Constructor

Public Sub New()
        MyBase.New()

        HireDate = #1/1/1900#
        Salary = 0.0
End Sub

'Parameterized Constructor using MyBase to invoke Base Class Parameterized Constructor
Public Sub New(ByVal strN As String, ByVal intIDNum As Integer, ByVal bBDate As Date, _
               ByVal dHDate As String, ByVal dbSal As Double)

        MyBase.New(strN, intIDNum, bBDate)
        HireDate = dHDate
        Salary = dbSal
End Sub
    '*********************************************************************
    'Regular Class Methods
    Public Sub PrintEmployee()
        'Call Inherited Print Method to display Base Class values
        Print()

        'Now display Derived Class values
        MessageBox.Show("Printing Employee Data " _
        & mdHireDate & ", " & mdbSalary)

    End Sub
    '*********************************************************************
End Class
```

Creating Sub Class Objects ONLY!(Main)

- Now let's look at the driver program.
- Since the Base Class was created using the keyword MustInherit, we can only create objects of the Sub Class **clsEmployee**.
- ***Main()*** test program:

Example 10 (Main Program):

- Driver Program for testing inheritance:

```
Module modMainModule


    'You can only Create Employee object
    Public objEmployee As clsEmployee = New clsEmployee("Joe Smith", 111, #1/12/1965#, _
                                                        #3/9/2004#, 30000)

    'CANNOT DECLARE OBJECT OF CLSPERSON! VB.NET & COMPILER WILL NOT LET YOU!!
    'Public objPerson As New clsPerson()

    Public Sub Main()

        'Call Employee Object to display data
        objEmployee.PrintEmployee()

    End Sub


End Module
```

Explanation of Test program:

- When we execute the program, the following occurs:

1. **We create on Objects of the clsEmployee Class as follows:**

```
'You can only Create Employee object
Public objEmployee As clsEmployee = New clsEmployee("Joe Smith", 111, #1/12/1965#, _
                                                    #3/9/2004#, 30000)
```

   - We show that if we attempt to create an Object of the Base Class clsPerson we will get a compiler error: :

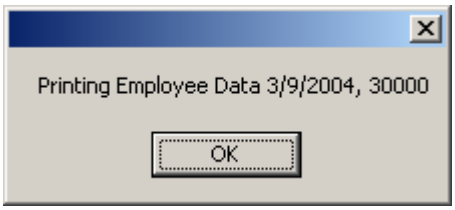
```
'CANNOT DECLARE OBJECT OF CLSPERSON! VB.NET & COMPILER WILL NOT LET YOU!!
'Public objPerson As New clsPerson()
```

2. **We call the Employee Class Print() Method to print each object's data to verify initialization values:**

```
'Call Employee Object to display data
objEmployee.PrintEmployee()
```

   - Calling the `objEmployee.PrintEmployee()` method of the first object will show that the Base class Print() is executed populated values are shown:

   - Now the `objEmployee.PrintEmployee()` method continues to execute it's code and prints the employee information as expected:



MustOverride Keyword (Abstract Method or Pure Virtual Function)

- The **MustOverride** Keyword works in conjunction with the **MustInherit** keyword.
- This keyword gives us the ability to create Methods (Sub, Function or Property) that MUST be overridden in the derived class.
- This means that the implementation of this class MUST be done in the Sub Class.
- Method using the keyword **MustOverride**, DO NOT contains any sort of implementation; there is no body or the keyword End Sub or End Function or End Property. **This type of method is also known as *Abstract Method* or *Pure Virtual Function*.**
- The idea is that the Base Class contains a DECLARATION of the method ONLY! Implementation MUST be done inside the Sub Class.
- NOTE THAT YOU MUST IMPLEMENT OR CREATE THE overridden METHOD IN THE SUB CLASS, YOU CANNOT CREATE THE SUB CLASS WITHOUT THE IMPLEMENTED VIRTUAL OR ABSTRACT METHOD, OTHEWISE A COMPILER ERROR WILL OCCUR WHEN CREATING OBJECTS OF THE SUB CLASS.
- Rules:
  - Base Class: Declaration only of Abstract or Virtual function using keyword **MustOverride**.
  - Sub Class: You must implement or create the method using the keyword: **Overrides**

Example 10B – MustOverride Keyword

Creating the Base Class

- We now create the base class.
- Again we use the keyword **MustInherit**:

Example 9B (Base-Class):

- Declaring the base class:

```
Option Explicit On
'Declare Class for MustInherit
Public MustInherit Class clsPerson
    '*********************************************************************
    'Class Data or Variable declarations
    Private m_strName As String
    Private m_intIDNumber As Integer
    Private m_dBirthDate As Date

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

    Public Property IDNumber() As Integer
        Get
            Return m_intIDNumber
        End Get
        Set(ByVal Value As Integer)
            m_intIDNumber = Value
        End Set
    End Property
    'We allow Property to be Overridden
    Public Overridable Property BirthDate() As Date
        Get
            Return m_dBirthDate
        End Get
        Set(ByVal Value As Date)
            m_dBirthDate = Value
        End Set
    End Property
```

clsPerson	
strName:	String
intIDNumber:	Integer
dBirthDate:	Date
Name():	String
IDNumber():	Integer
BirthDate():	Date
New()	
New(String, Integer , Date)	
Print()	

Example 10B (Base-Class):

- Declaring the remaining base members:

```
    '*********************************************************************
    '*********************************************************************
    'Class Constructor Methods
    Public Sub New()
        'Note that private data members are being initialized
        m_strName = ""
        m_intIDNumber = 0
        m_dBirthDate = #1/1/1900#
    End Sub


Public Sub New(ByVal strN As String, ByVal intIDNum As Integer, ByVal bBDate As Date)
  'Note that we are NOT using the private data but the Property Procedures instead
        Name = strN
        IDNumber = intIDNum
        BirthDate = bBDate
End Sub

'*********************************************************************
'Regular Class Methods
'We allow Method to be Overridden
Public Overridable Sub Print()
        MessageBox.Show("Printing BASE CLASS Person Data " _
        & m_strName & ", " & m_intIDNumber & ", " & _
        m_dBirthDate)
End Sub

    'Declaration of MustOverride Method (Note that there is no End Sub)
    'This method is also Known as Abstract Method or Virtual Function
    Public MustOverride Sub Shop(ByVal intItems As Integer)
```

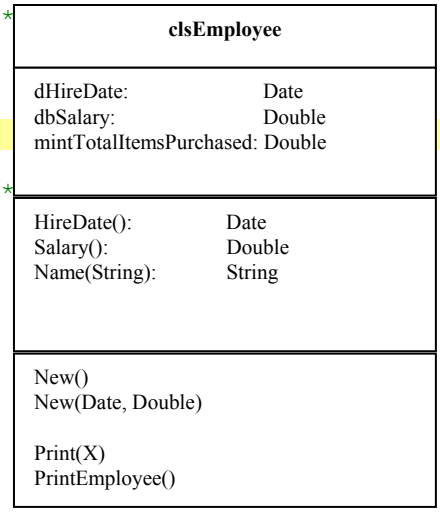
Derived or Sub Class

- In this example we will add a data member to store the total items purchased by employee object.
- We will also add the corresponding Property TotalItemsPurchased
- In addition, we will implement the *Pure Virtual Function* or *Abstract Method* declared in the Base Class Shop()
- Lets look at the derived class *clsEmployee*:

Example 10B (SubClass):

- Declaring the SubClass:

```
Option Explicit On
Public Class clsEmployee
    Inherits clsPerson
    '*************************************************************************
    '*************************************************
    'Class Data or Variable declarations
    Private mdHireDate As Date
    Private mdbSalary As Double
    Private mintTotalItemsPurchased As Integer

    '*************************************************
    'Property Procedures
    Public Property HireDate() As Date
        Get
            Return mdHireDate
        End Get
        Set(ByVal Value As Date)
            mdHireDate = Value
        End Set
    End Property

    Public Property Salary() As Double
        Get
            Return mdbSalary
        End Get
        Set(ByVal Value As Double)
            mdbSalary = Value
        End Set
    End Property

    Public Property TotalItemsPurchased() As Integer
        Get
            Return mintTotalItemsPurchased
        End Get
        Set(ByVal Value As Integer)
            mintTotalItemsPurchased = Value
        End Set
    End Property
```



Example 10B (SubClass-(Cont)):

- Declaring the SubClass Methods:

```
'*********************************************************************
    Public Sub New()
        MyBase.New()

        HireDate = #1/1/1900#
        Salary = 0.0
        TotalItemsPurchased = 0
    End Sub

Public Sub New(ByVal strN As String, ByVal intIDNum As Integer, ByVal bBDate As Date, _
               ByVal dHDate As String, ByVal dbSal As Double)

        MyBase.New(strN, intIDNum, bBDate)
        HireDate = dHDate
        Salary = dbSal
    End Sub
'*********************************************************************
    'Regular Class Methods
    Public Sub PrintEmployee()
        'Call Inherited Print Method to display Base Class values
        Print()

        'Now display Derived Class values
        MessageBox.Show("Printing Employee Data " _
        & mdHireDate & ", " & mdbSalary & ", " & mintTotalItemsPurchased)

    End Sub
    '*********************************************************************
'Implementation of Pure Virtual Function
'Shop() must use keyword Overrides since it's declared MustOverride in Base Class
    Public Overrides Sub Shop(ByVal intItems As Integer)

        mintTotalItemsPurchased = mintTotalItemsPurchased + intItems
    End Sub

End Class
```

Creating Sub Class Objects ONLY!(Main)

- Now let's look at the driver program.
- Since the Base Class was created using the keyword MustInherit, we can only create objects of the Sub Class **clsEmployee**.
- We also show the use of the Implemented Virtual Method **Shop().**
- ***Main()*** test program:

Example 10B(Main Program):

- Driver Program for testing inheritance:

```
Module modMainModule

 'Create Object of Sub Class Employee
 Public objEmployee As clsEmployee = New clsEmployee("Joe Smith", 111, #1/12/1965#, _
                                                     #3/9/2004#, 30000)

    'CANNOT DECLARE OBJECT OF CLSPERSON! VB.NET & COMPILER WILL NOT LET YOU!!
    'Public objPerson As New clsPerson()

    Public Sub Main()

        'Call Employee Object PrintEmployee to display data
        objEmployee.PrintEmployee()

        'Call to Employee Object Shop() method to purchase 10 items
        objEmployee.Shop(10)

        'Call Employee Object PrintEmployee again to display data
        'The data displayed will show that the purchase Item value is equal to 1o items.
        objEmployee.PrintEmployee()

    End Sub
End Module
```

Explanation of Test program:

- When we execute the program, the following occurs:

1. **We create on Objects of the clsEmployee Class as follows:**

```
'You can only Create Employee object
Public objEmployee As clsEmployee = New clsEmployee("Joe Smith", 111, #1/12/1965#, _
                                                    #3/9/2004#, 30000)
```

   - We show that if we attempt to create an Object of the Base Class clsPerson we will get a compiler error: :

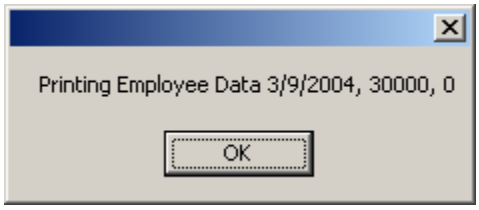
```
'CANNOT DECLARE OBJECT OF CLSPERSON! VB.NET & COMPILER WILL NOT LET YOU!!
'Public objPerson As New clsPerson()
```

2. **We call the Employee Class Print() Method to print each object's data to verify initialization values:**

```
'Call Employee Object to display data
objEmployee.PrintEmployee()
```

   - Calling the `objEmployee.PrintEmployee()` method of the first object will show that the Base class Print() is executed populated values are shown:

   - Now the `objEmployee.PrintEmployee()` method continues to execute it's code and prints the employee information as expected. **Note the additional TotalPurchasedItems value = 0**:



3. **We call the Implementation of Employee Class Shop() Method purchase 10 items:**

```
'Call to Employee Object Shop() method to purchase 10 items
objEmployee2.Shop(10)
```

4. **We call the Employee Class Print() Method again to print each object's data to verify that 10 items were purchased:**

```
'Call Employee Object to display data
objEmployee.PrintEmployee()
```

- Calling the `objEmployee.PrintEmployee()` method of the first object will show that the Base class Print() is executed populated values are shown:

- Now the `objEmployee1.PrintEmployee()` method continues to execute it's code and prints the employee information as expected**. Note the additional TotalPurchasedItems value = 10**: