CS708 Lecture Notes

Visual Basic.NET Programming

Object-Oriented Programming

Inheritance Review

(Part I of I)

(Lecture Notes 1A)

Prof. Abel Angel Rodriguez

Chapter 1 Inheritance Review

1.1 Introduction to Inheritance

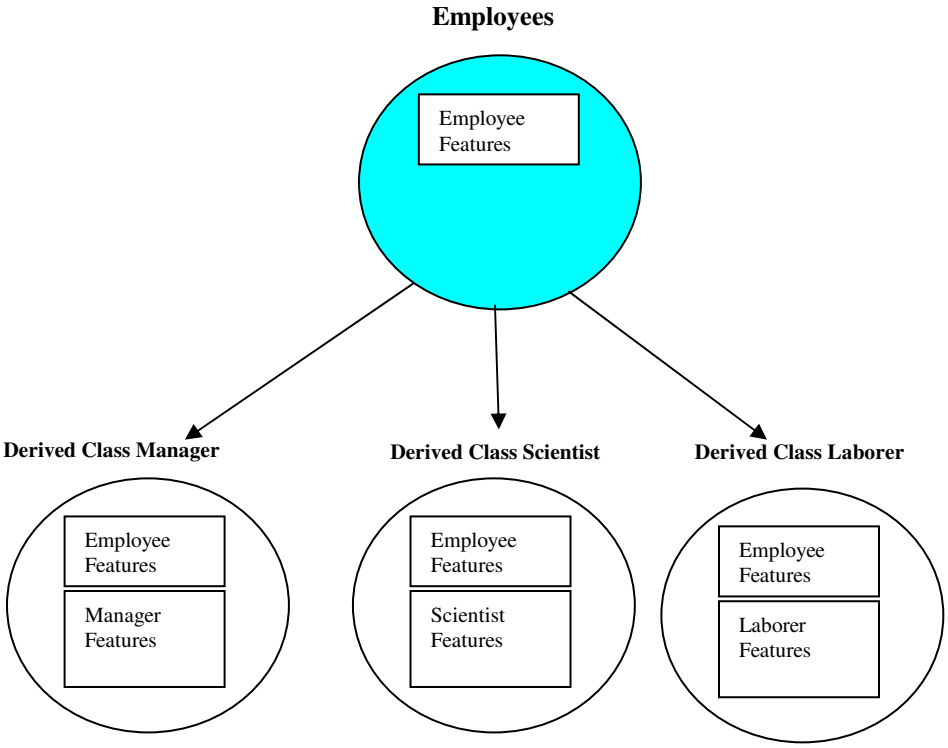
1.1.1 Introduction to Inheritance

Reusability

- Reusability is the concept of re-using objects that we create in other programs.
- This concept has revolutionized the field of programming. Applications which took longer to developed are now being created at a much faster rate since objects from other applications are being reused, thus saving time on programming and testing.
- The Objects re-used have already been tested in previous programs so they are guaranteed to work safely thus yielding a robust program.
- This concept of *reusability* spawned a new software industry where companies were established whose sole business is to create ready tested Objects to sell to other software development houses.
- The main Object-Oriented Programming concept provided to implement reusability is ***Inheritance***.

Inheritance

- Inheritance is probably the most powerful feature of Object-oriented programming.
- Inheritance is the process of creating new class, called ***Sub Class***, (*Derived Class*) from an existing parent class. The parent class is called a ***base class***.
- The derived class inherits all the capabilities of the ***base class*** but can add features of its own. Note that the base class is unchanged by this process.
- Any class you created can be a base class and any derive class can become a base class to its derived children classes.
- Inheritance is a big payoff since it permits code *reusability*. Once the base class is written and debugged, it needs not to be touched again, but can be adapted to work in different situations. Reusing existing code saves time and money and increases program reliability.
- For example supposed we create an Employee Class, which contain standard employee features such as name, id, address, benefits etc. We can then derive classes for each of the different category of employees in the company, such as managers, scientist, laborers etc.
- The UML illustration below demonstrates this concept:



1.1.2 Implementing Basic Inheritance

Creating the Base Class

- Any class we create can be a base class.
- Note that I will use as a convention of using the prefix **m_** for all private variables of the base class to differentiate them from the variables of the derived class. I will use the prefix **m** for all private variables of the derived class.

Creating the Derived Class

- The Syntax to creating the **derived** or **SubClass** to inherit from a Base class is as follows:

```
'Class Header
Public Class  SubClassName

Inherits BaseClassName

    Data Definitions

    Properties Definitions

    Methods

End Class
```

Example:

- **Creating a Classes:**

- Example a) - Creating a Derived Class Video from a Base Class Product:

```
Public Class Video
        Inherits Products

'Properties,
'Methods
' Event-Procedures


End Class
```

- Example b) - Creating an Employees class from a Person Class:

```
Public Class Employee
        Inherits Person
'Properties,
'Methods
' Event-Procedures


End Class
```

- Lets look at the following **clsPerson** class example (Note the UML diagram):

Example 1 (Base Class):

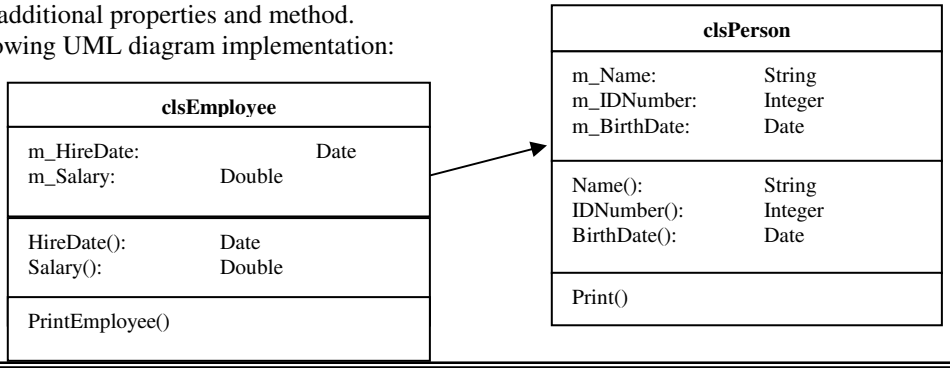
- Declaring the base class:

```
Option Explicit On
Public Class clsPerson
    '*********************************************************************
    'Class Data or Variable declarations
    Private m_Name As String
    Private m_IDNumber As Integer
    Private m_BirthDate As Date

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

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

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

    '*********************************************************************
    'Regular Class Methods
    Public Sub Print()
        MessageBox.Show("Printing Person Data " _
        & m_Name & ", " & m_IDNumber & ", " & _
        m_BirthDate)

    End Sub

End Class
```

clsPerson	
m_Name:	String
m_IDNumber:	Integer
m_BirthDate:	Date
Name():	String
IDNumber():	Integer
BirthDate():	Date
Print()	

Creating the Subclass (Derived Class)

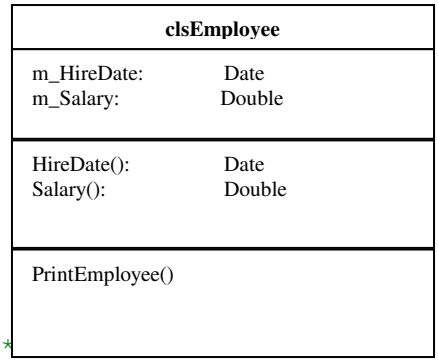
- Using the ***Inherit*** keyword in a class declaration, we can derive other classes from the clsPerson class.
- For example supposed we wished to create an Employee class ***clsEmployee*** as a subclass to ***clsPerson***, which inherits the feature from ***clsPerson***, but adds additional properties and method.
- Suppose we want the following UML diagram implementation:



Example 1 (SubClass):

- Declaring the SubClass:

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

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

    Public Property Salary() As Integer
        Get
            Return m_Salary
        End Get
        Set(ByVal Value As Integer)
            m_Salary = Value
        End Set
    End Property
    '*********************************************
    '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 " _
        & m_HireDate & ", " & m_Salary)

    End Sub
End Class
```



Using the Base Class & SubClass

- Now that our subclass is derived from the base class, we can use the properties of the subclass.
- Due to inheritance, objects of the subclass will inherit the functionality of the base class
- For example, the subclass ***clsEmployee*** does not implement the properties *Name*, *BirthDate* and *IDNumber*, but objects of this class will show that *Name*, *BirthDate* and *IDNumber* are property members but they are really not, they are implemented by ***clsPerson*** the base class.
- Note that the private variables m_int*Name*, m_*BirthDate* and m_*IDNumber* will not be accessible by the child class, since they are private. The child or subclass only has access to public members and inherits them directly
- Let's look at a main test program. We will create an object of the base class as well as the subclass in order to demonstrate inheritance.
- ***Main()*** test program:

Example 1 (Main Program):

- Driver Program for testing inheritance:

```
Module modMainModule

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


    Public Sub Main()

        'Testing & Populating Person Object with Data
        With objPerson
            .Name = "Joe Smith"
            .IDNumber = 111
            .BirthDate = #1/2/1965#
        End With

        'Call Person Object Only Method
        objPerson.Print()



        'Populating Employee Object with Data. Note Base Class Member Access
        With objEmployee
            .Name = "Mary Johnson"
            .IDNumber = 111
            .BirthDate = #4/12/1970#
            .HireDate = #3/9/2004#
            .Salary = 30000
        End With

        'Call Employee Object Method
        objEmployee.PrintEmployee()

    End Sub


End Module
```

Summary:

- We clearly showed that we can inherit all the features of the Base Class and add features of our own in the subclasses.
- We took advantage of the interface and behavior (Methods) of the Person class and extended it via an Employee class to represent an employee.
- By using an existing Person class we saved development time when creating an Employee class. Another example of ***reusability***!

1.1.3 Available Access to Base Class Members from SubClasses

Access Public & Private Members of the Base Class

- The rule data encapsulation of Object-Oriented-Programming always hold

Private data is <u>private</u> and only members of the class have access to it!

- Therefore derived classes DO NOT have access to their parent's ***Private*** data only to the ***<u>Public</u>*** Interface (*Properties & Methods*)

The "Protected" Access Keyword

- In inheritance there is another level of security in the Base Class offered for *SubClasses*. This level is known as Protected Data, using the keyword "**<u>Protected</u>**".
- The Protected keyword means that derived classes are the only ones that can access *protected* members of the base class
- To any other class a variable declared with the keyword Protected is Private. The rule is:

No other classes other than a derived class have access to a Protected Member!

Summary

- 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

1.2 Inheritance Concepts

1.2.1 Inheritance Features

- In this section we will cover some of the features available via inheritance.
- Inheritance is a powerful tool of VB.NET and contains much functionality. I will only cover the following:

  - MyBase Keyword
  - Overloading Methods & Properties
  - Overriding Methods & Properties
  - Shadowing
  - Level of Inheritance
  - Constructors
  - Protected Scope
  - Abstract Base Class

1.2.2 MyBase Keyword

Introduction

- The Keyword **MyBase** explicitly or directly exposes the *Base Class* methods to the *Derived Classes*.
- Don't get confused, a derived Class automatically inherits and can see the Public Base class members, but if we can use the keyword **MyBase** as well to refer to the base class member.
- For example:
  - In our previous examples we derived from *clsPerson* a 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 in this example 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.

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 directly.
- There will be situation where the compiler will yield errors, because of name conflicts between the Base class and the Sub Class, in these situations use the keyword MyBase to explicitly tell the compiler that is the base class method version you want to execute.

- This will be clear in topics such as Constructors in inheritance and Method Overriding in future lectures.

1.2.3 Method Overloading in Inheritance

Method Overloading

- In normal circumstances, ***Method Overloading*** gives us the ability to implement methods with the same name, but Signature or parameter list is different. As long as the numbers of arguments are different, we can create methods having the same name.
- In inheritance, ***Method Overloading*** is used to extend or provide the Derived or ***Sub Classes*** with a new version of a property or method. Both the Base Class member and Sub Class member have the same name, but the number or type of parameters is different.
- Note that the original Base class method is still available, but in the child class we extended it by adding another method or property that performs some other implementation or upgrade of the base class version. This is the beauty of inheritance, not only can we inherit, but we can extend the features currently available by the Base Class.
- Let's look at another version of the previous example where we will overload the *Print()* method of the Base Class by adding a ***Print(int X)*** method in the derived class that will Print the Base Class data X times. We will also overload the ***Name Property*** to add a comment to the Name string.

Example 2 – Overloading Methods

Creating the Base Class

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

Example 2 (Base-Class):

- Declaring the base class:

```
Public Class clsPerson
    '*********************************************************************
    'Class Data or Variable declarations
    Private m_Name As String
    Private m_Number As Integer
    Private m_BirthDate As Date
    '*********************************************************************
    'Property Procedures
    Public Property Name() As String
        Get
            Return m_Name
        End Get
        Set(ByVal Value As String)
            m_Name = Value
        End Set
    End Property
    Public Property IDNumber() As Integer
        Get
            Return m_IDNumber
        End Get
        Set(ByVal Value As Integer)
            m_IDNumber = Value
        End Set
    End Property
    Public Property BirthDate() As Date
        Get
            Return m_BirthDate
        End Get
        Set(ByVal Value As Date)
            m_BirthDate = Value
        End Set
    End Property
    '*********************************************************************
    'Regular Class Methods
    Public Sub Print()
        MessageBox.Show("Printing Person Data " _
        & m_Name & ", " & m_IDNumber & ", " & _
        m_BirthDate)
    End Sub
End Class
```

clsPerson	
m_Name:	String
m_IDNumber:	Integer
m_BirthDate:	Date
Name():	String
IDNumber():	Integer
BirthDate():	Date
Print()	

Overloading the Print Method using the OverLoads Keyword

- We create the ***clsEmployees*** class and as usual we use the ***Inherit*** keyword in a class declaration to inherit from the ***clsPerson*** Class.
- In order to implement method overloading we need to use the keyword Overload in the declaration of the method or property.
- Using the keyword **Overload**, we add another Name Property which takes as argument a string representing a comment that will be added to the Name string.
- Using the keyword Overload, we overload the Base Class *Print()* method by adding another Method named *Print(X)* which takes one argument.
- Lets look at the derived class *clsEmployee*:

Example 2 (SubClass):

- Declaring the SubClas:

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

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

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

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

```
'Overloading the Base Class Name Property
    Public Overloads Property Name(ByVal knownAlias As String) As String
        Get
            Return MyBase.Name
        End Get
        Set(ByVal Value As String)
            'Add the Comment to the end of the name
            MyBase.Name = Value & "  (" & knownAlias & ")"

        End Set
    End Property
```

Example 2 (SubClass-(Cont)):

- Declaring the SubClass Methods:

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

    'Overloaded Base Class Method
    Public Overloads Sub Print(ByVal intNumberOfPrints As Integer)
        Dim i As Integer

        For i = 1 To intNumberOfPrints
            MessageBox.Show("Multiple Print Jobs for: " _
            & Name & ", " & IDNumber & ", " & _
            BirthDate)
        Next

    End Sub

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

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

    End Sub

End Class
```

Using the SubClass and Calling the Overloaded Property & Method

- In this example we create two objects of the ***clsEmployee*** class. We will no longer need to create objects of the Base Class, unless necessary, since the derived class objects contain everything from the base and more.
- We assign values to the first Employee Object using the standard Properties inherited by the Base Class: ***Name***, ***BirthDate*** and ***IDNumber***, those provided by the derived class: ***HireDate*** & ***Salary***.
- We call the first Employee Object **PrintEmployee** Method to print both the Base Class data and Derived Class data.
- In the second Employee Object, we assign values to only two of the properties inherited by the Base Class: ***BirthDate*** and ***IDNumber***, we have the option of using the inherited property *Name*, or the overloaded properties provided by the derived class: Overloaded Property ***Name(X)***, and the regular ***HireDate*** & ***Salary***
- In the second Employee object we call the **PrintEmployee()** method to print both Base & Derived Class data and in addition we call the overloaded method Print(X) to print only the Base Class data X times.
- ***Main()*** test program:

Example 3 (Main Program):

- Driver Program for testing inheritance:

```
Module modMainModule

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

    Public Sub Main()

        'Populating Person Object with Data
        With objEmployee1
            .Name = "Joe Smith"
            .IDNumber = 111
            .BirthDate = #1/2/1965#
            .HireDate = #5/23/2004#
            .Salary = 50000
        End With

        'Call Employee Object Method
        objEmployee1.PrintEmployee()

        'Populating Employee2 Object with Data
        With objEmployee2
            'Assign Overloaded Property
            .Name = "Mary Johnson"  'Regular Base Class name property
            .Name("Chicky") = "Mary Johnson" 'version appends the Alias "Chicky"
            .IDNumber = 444
            .BirthDate = #4/12/1970#
            .HireDate = #3/9/2004#
            .Salary = 30000
        End With
        'Call Employee Class PrintEmployee method
        objEmployee2.PrintEmployee()

        'Call Overloaded PrintPerson method
        objEmployee2.Print(3)

    End Sub

End Module
```

Summary:

- We clearly showed that we can not only inherit all the features of the Base Class and add features of our own in the subclasses but also extend the Base Class features by **Overloading** them and extending them to perform more functionalities.

1.2.4 Method Overriding

Introduction

- In the previous section we learned *Method Overloading*. Overloading allowed us to extend the functionality of a Base class Method or Property by adding a new version in the Derived Class with the same name, but as long as the parameter list is different.
- The key point to *Overloading* is that we kept the original functionality of the base and just added a new or additional functionality in the child or *SubClass*.
- Now let's supposed we want NOT just extend an implementation of the base class, but change or completely replace a functionality of a method or property.
- This is where ***Method Overriding*** comes in to play.
- 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.
- The key point here is that we are replacing! The new method has the same signature (Name, # of parameters, return type etc).

Implementing Method Overriding

- To implement Method Overriding we need to use two keyword: ***Overridable*** & ***Overrides***
- To implement we first need to realize that we just can't simply override a Base Class. The base class needs to give us permission to do so, in other words the Method or Property in the Base Class must grant this feature. This is where the keyword ***Overridable*** is used.

Overridable keyword

- The ***Overridable*** must be stated in the Base Class on every Method or Property in which the Base Class allows the Derived Classes to override.
- The idea here is that the Base Class is in control of which Methods and Property a Derived class can override.

Overrides keyword

- Once a Property or Method has the *Overridable* keyword, the derived class can override the Method/Property using the keyword ***Overrides***. This keyword tells the SubClass that this Method/Property is to override the one in the Parent or Base Class.
- The overridden method in the Base class will not execute at all via the Sub Class. Only the new version will execute.
- Now don't get confuse by this statement. Note that we are saying that the overridden method in the derive class will run and not the one in the base class. But this is only when we are trying to call the method from and object of the child or derived class that the new one executes. You can still run the original but only if you create an object of the Base Class as expected.

Example 4 – Overriding Property & Methods

- Lets look at another version of the previous example where this time we will override the ***BirthDate*** Property and the *Print()* method of the Base Class by replacing it with a NEW version of ***BirthDate*** and *Print()* method in the derived class.

Creating the Base Class

- Using the keyword ***Overridable*** the BASE Class designer allows the *Birthdate* & *Print()* method to be overridden:

Example 4 (Base-Class):

- Declaring the base class:

```
Option Explicit On
Public Class clsPerson
    '*******************************************************************
    'Class Data or Variable declarations
    Private m_Name As String
    Private m_IDNumber As Integer
    Private m_BirthDate As Date

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

    Public Property IDNumber() As Integer
        Get
            Return m_IDNumber
        End Get
        Set(ByVal Value As Integer)
            m_IDNumber = Value
        End Set
    End Property
    'We allow Property to be overridden
    Public Overridable Property BirthDate() As Date
        Get
            Return m_BirthDate
        End Get
        Set(ByVal Value As Date)
            m_BirthDate = 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_Name & ", " & m_IDNumber & ", " & _
        m_BirthDate)
    End Sub

End Class
```

clsPerson	
m_Name:	String
m_IDNumber:	Integer
m_BirthDate:	Date
Name():	String
IDNumber():	Integer
BirthDate():	Date
Print()	

Overriding the BirthDate Property

- 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 a new policy within the company that every employee must be at least 16 years old and we ***Throw*** an Exception. This will help us review Throwing Exceptions.
- The new Birthdate uses ***MyBase*** Keyword to explicitly direct the compiler to the Base Class *Birthdate* property so we can use that mechanism to store the date.
- Let's look at the derived class *clsEmployee*:

Example 4 (SubClass):

- Declaring the SubClas:

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

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

    Public Property Salary() As Integer
        Get
            Return m_Salary
        End Get
        Set(ByVal Value As Integer)
            m_Salary = 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:	Date
dbSalary:	Double
HireDate():	Date
Salary():	Double
BirthDate :	Date
Print()	
PrintEmployee()	

Overriding the Print() Method

- Now we override the *Print()* Method using the keyword ***Overrides*** .
- Here I take advantage that the Base Class already has a Print() method, so why not utilize it.
- Therefore we use the keyword ***MyBase*** to explicitly call the Base Class *Print()*, then we add any new features we 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 Birth date"
                         & 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 " _
        & m_HireDate & ", " & m_Salary)

    End Sub
End Class
```

Validating our theory by Calling the Overridden Property & Method

- Now let's look at the driver program.
- In this example we create three objects, one of the Base Class ***clsPerson*** and two of the ***clsEmployee*** class.
- The will use the Base Class Object simply to prove that the Print() Method of this object is still valid for Person Objects, but NOT for the Derived Classes. We will do this by assigning values to this object and calling the Print() method.
- In the first Employee object we will assign values using the standard Properties inherited by the Base Class: *Name* and *IDNumber*, (Note that *BirthDate* is overridden and no longer inherited) those provided by the derived class: *BirthDate (Overridden)*, *HireDate* & *Salary*.
- We call the first Employee Object *PrintEmployee()* Method to print both the Base Class data and Derived Class data.
- In the second Employee Object perform the same operations.
- ***Main()*** test program:

Example 4 (Main Program):

- Driver Program for testing inheritance:

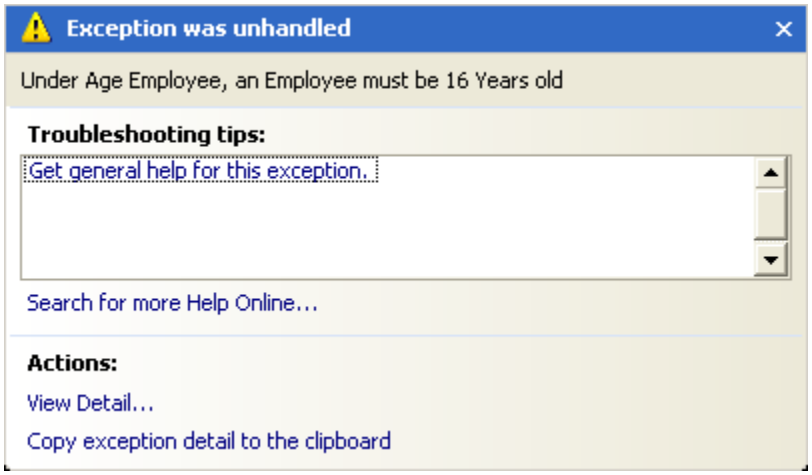
```
Option Explicit On
Module modMainModule
    'Declare & Create Public Person & Employee Objects
    Public objEmployee1 As clsEmployee = New clsEmployee()
    Public objEmployee2 As clsEmployee = New clsEmployee()
    Public objPerson As clsPerson = New clsPerson()

    Public Sub Main()

        '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 = #4/12/1993#  'This date will raise and exception!
            .HireDate = #5/23/2004#
            .Salary = 30000
        End With

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

    End Sub

End Module
```

Explanation & Results of Main Program:

- When we execute the program, the following occurs:

1. **We populate the Second Object with values and set the Overridden BirthDate properties of the Employee:**

```
'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 = #4/12/1993#  'This date will raise and exception!
    .HireDate = #5/23/2004#
    .Salary = 30000
End With
```



Results and Explanation:

- In this object we populate the populate from the Base Class the *Name* and *IDNumber*. For the derived class we populate the Overridden *BirthDate* Property, *HireDate* & *Salary*.
- Remember that the NEW *BirthDate* Property has code that will test to make sure that the employee is over 16 years of age. Yet the value chosen for the *BirthDate* Property is a year which will indicates that the employee is under 16, therefore an Exception is thrown by our code.
- Since our code contain no Error Handling Code (Try-Catch-Finally Statement) the PROGRAM WILL STOP RIGHT HERE AND stop execution
- THE PROGRAM CANNOT CONTINUE AT THIS POINT. THE FOLLOWING CODE IS NEVER EXECUTED:

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

Example 5 – Example 4 with Error Handling (Overriding Property & Methods Cont)

- In our previous Example 3 we clearly showed how Method Overriding works. But our example raised and Exception.
- Now we add error handling code using the ***Try-Catch-Finally*** Statement in order to prevent the program from stopping.

Creating the Base Class & Derived Class

- Same as Example 3

Main Program with Error Handling Code

- Ok the Main program is still the same, but this time we will add a ***Try-Catch-Finally*** statement to trap and handle the error.

Example 5 (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.  BirthDate 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. BirthDate is < 16
            With objEmployee2
                .Name = "Mary Johnson"
                .IDNumber = 444
                .BirthDate = #4/12/1993#
                .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
End Module
```

Explanation & Results of Main Program:

- When we execute the program, the following occurs:

1. **Now when we populate the Second Object with values and set the Overridden BirthDate properties of the Employee as follows:**

```
'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 = #4/12/1993#  'This date will raise and exception!
    .HireDate = #5/23/2004#
    .Salary = 30000
End With
```

2. **We get the following results:**

Results and Explanation:

- In this case when the NEW *BirthDate* Property traps and under age employee, since our code contain Exception Error Handling Code (Try-Catch-Finally Statement) the PROGRAM WILL STOP NOT STOP THE EXECUTION RIGHT HERE , BUT INSTEAD JUMP TO THE CATCH STATEMENT TO HANDLE THE EXCEPTION.
- A MESSAGE BOX IS DISPLAYED TO GRACEFULLY PROMPT THE USER OF THE ERROR.
- NOTE THAT THE FOLLOWING CODE SECTION IS NEVER EXECUTED BECAUSE IT IS SKIPPED BY THE ERROR HANDLING MECHANISM:

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

1.2.5 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 6 – 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.

Creating the Base Class

- Same as before:

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

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

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

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

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 6 (Base-Class Cont):

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

    Public Property TotalItemsPurchased() As Integer
        Get
            Return m_TotalItemsPurchased
        End Get
        Set(ByVal Value As Integer)
            m_TotalItemsPurchased = 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_Name & ", " & m_IDNumber & ", " & _
        m_BirthDate & ", " & m_Phone)
    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 6 (SubClass):

- Declaring the SubClas:

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

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

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

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

```
    '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
```

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 6 (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 " _
            & MyBase.Name & ", " & MyBase.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 " _
        & m_HireDate & ", " & m_Salary)

    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.
- 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.

Example 6 (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 Displaying the Base Class Phone as expected
            objPerson.Print()

            'Populating Employee Object (The Phone property 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 with Shadowed Phone with the (Cell) string
            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)
        End Try
    End Sub
End Module
```

Explanation & Results of Main Program:

- When we execute the program, the following occurs:

1. **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.

1.2.7 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 initialize the Base Class Data when we create a Derived Class Object?
- Here we need to review Constructors 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…..)

Constructor and Inheritance

- Constructors play an important role in inheritance. It is the job of the derived or Sub-Class constructor to call and populate the Base Class Constructor.

Example 7 – Constructor Methods in Base and Derived Classes

Creating the Base Class

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

Example 7 (Base-Class):

- Declaring the base class:

```
Option Explicit On
Public Class clsPerson
    '*********************************************************************
    'Class Data or Variable declarations
    Private m_Name As String
    Private m_IDNumber As Integer
    Private m_BirthDate As Date

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

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

    Public Property BirthDate() As Date
        Get
            Return m_BirthDate
        End Get
        Set(ByVal dTheBDate As Date)
            m_BirthDate = 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 7 (Base-Class):

- Declaring the remaining base members:

```
    '*********************************************************************
    'Class Constructor Methods
    Public Sub New()
        'Note that private data members are being initialized
        m_Name = ""
        m_IDNumber = 0
        m_BirthDate = #1/1/1900#

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


    Public Sub New(ByVal N As String, ByVal IDNum As Integer, _
    ByVal BDate As Date)
   'Note that we are NOT using the private data but the Property Procedures
        Me.Name = N
        Me.IDNumber = IDNum
        Me.BirthDate = BDate

        '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_IDNumber & ", " & _
        m_dBirthDate)

    End Sub

End Class
```

Results and Explanation:

DEFAULT CONSTRUCTOR NEW():

- The DEFAULT CONSTRUCTOR New() initializes itself with default data.
- Note that the default constructor sets the private data directly and NOT a property. I am doing this for performance and simply to show that we can directly set the private data since we know the value we are setting is GOOD DATA.

PAREMETERIZED CONSTRUCTOR NEW(X,Y,Z . . .):

- The PARAMETERIZED CONSTRUCTOR New(x,y,z..) initializes itself with data passed as parameters.
- IMPORTANT! Note that the parameterized constructor sets the PUBLIC PROPERTIES, instead of private data. It is important that you understand this. We use the property so that the data coming from the outside world as parameters can be VALIDATED IN THE PROPERTY. If we were to directly set to the private data, we could set our class with BAD DATA. THE PROPERTIES CAN CONTAIN VALIDATION CODE TO VALIDATE THE DATA BEFORE ASSIGNING TO THE PRIVATE VARIABLES.

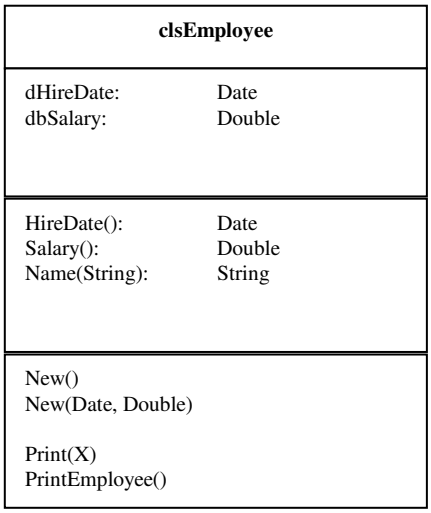
Derived or Sub Class Constructors

- The derived class has it' own constructors are well. But the derived class must provide the values to initiate the Base Class Parameterized constructor.
- Lets look at the derived class *clsEmployee*:

Example 7 (SubClass):

- Declaring the SubClas:

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

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

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



Example 7 (SubClass-(Cont)):

- Declaring the SubClass Methods:

```
    '*********************************************************************
    'Constructor Class Methods
   Public Sub New()

        MyBase.New()

        m_HireDate = #1/1/1900#
        m_Salary = 0.0

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

    Public Sub New(ByVal N As String, ByVal IDNum As Integer, _
    ByVal BDate As Date, ByVal HDate As String, ByVal Sal As Double)

        MyBase.New(N, IDNum, BDate)

        Me.HireDate = HDate
        Me.Salary = Sal
       MessageBox.Show("Sub Class Parameterize Constructor executed....")
    End Sub

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

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

    End Sub
End Class
```

Explanation:

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

```
    Public Sub New(ByVal N As String, ByVal IDNum As Integer, _
    ByVal BDate As Date, ByVal HDate As String, ByVal Sal As Double)
```

  - IMPORTANT! Note that the parameterized constructor sets the PUBLIC PROPERTIES, instead of private data providing a mechanism for the properties to implement VALIDATION CODE and validate the data from the outside world.

- 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(N, IDNum, BDate)
```

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:

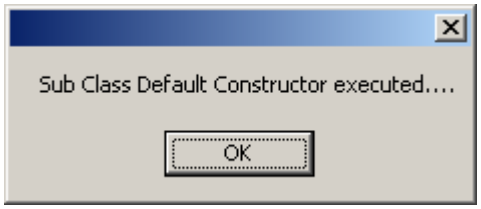
- When we execute the program, the following occurs:

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:**

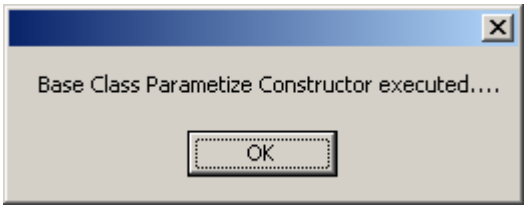
```
'Create Employee objects that invokes default & Parametized Constructors
Public objEmployee1 As clsEmployee = New clsEmployee()
Public objEmployee2 As clsEmployee = New clsEmployee("Joe Smith", 111, #1/12/1965#, _
#3/9/2004#, 30000)
```

   a. When we create the first object ***objEmployee1***, there are no arguments so the <u>default</u> constructors execute. The *clsEmployee* class <u>default</u> constructor will call the *clsPerson* <u>default</u> constructor. The message box will display:

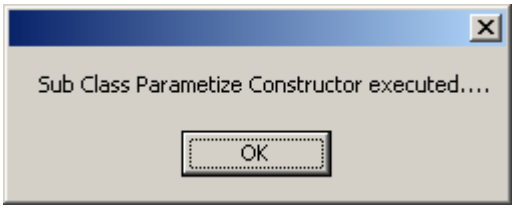
   b. Then of course the *clsEmployee* class <u>default</u> constructor continues to execute its code as shown by the message box:



c. 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:



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

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 N As String, ByVal IDNum As Integer, _
ByVal BDate As Date, ByVal HDate As String, ByVal Sal As Double)

    MyBase.New(N, IDNum, BDate)

    Me.HireDate = HDate
    Me.Salary = Sal

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

1.2.8 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 8 – 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 8 (Base-Class):

- Declaring the base class:

```
Option Explicit On
Public Class clsPerson
    '*****************************************************************
    'Class Data or Variable declarations
    Private m_Name As String
    Private m_IDNumber As Integer
    Private m_BirthDate As Date

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

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

    Public Property BirthDate() As Date
        Get
            Return m_BirthDate
        End Get
        Set(ByVal dTheBDate As Date)
            m_BirthDate = 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 8 (Base-Class):

- Declaring the remaining base members:

```
    '*******************************************************************
    'Class Constructor Methods
    Public Sub New()
        'Note that private data members are being initialized
        m_Name = ""
        m_IDNumber = 0
        m_BirthDate = #1/1/1900#

    End Sub

Public Sub New(ByVal N As String, ByVal IDNum As Integer, ByVal BDate As Date)
    'Note that we are NOT using the private data but the Property Procedures instead
        Me.Name = N
        Me.SocialSecurityNumber = IDNum
        Me.BirthDate = BDate
    End Sub

    '*******************************************************************
    'Regular Class Methods
    Public Sub Print()
        MessageBox.Show("Printing Person Data " _
        & m_Name & ", " & m_IDNumber & ", " & _
        m_BirthDate)

    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 8 (SubClass):

- Declaring the SubClas:

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

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

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

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

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

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

Example 8 (SubClass-(Cont)):

- Declaring the SubClass Methods:

```
    '*********************************************************************
    Public Sub New()
        MyBase.New()
        m_HireDate = #1/1/1900#
        m_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)
        Me.HireDate = dHDate
        Me.Salary = dbSal

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

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

    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 8 (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 PROTECTED!!!
     '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
```

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.

1.2.9 MustInherit & MustOverride Keywords (Important Topics for CS708)

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 9 – 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 9 (Base-Class):

- Declaring the base class:

```
Option Explicit On
'Declare Class for MustInherit
Public MustInherit Class clsPerson
    '*******************************************************************
    'Class Data or Variable declarations
    Private m_Name As String
    Private m_IDNumber As Integer
    Private m_BirthDate As Date

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

    Public Property IDNumber() As Integer
        Get
            Return m_IDNumber
        End Get
        Set(ByVal Value As Integer)
            m_IDNumber = Value
        End Set
    End Property
    'We allow Property to be Overridden
    Public Overridable Property BirthDate() As Date
        Get
            Return m_BirthDate
        End Get
        Set(ByVal Value As Date)
            m_BirthDate = 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 9 (Base-Class):

- Declaring the remaining base members:

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


    Public Sub New(ByVal N As String, ByVal IDNum As Integer, ByVal BDate As Date)
    'Note that we are NOT using the private data but the Property Procedures instead
        Me.Name = N
        Me.IDNumber = IDNum
        Me.BirthDate = BDate
    End Sub

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


End Class
```

Derived or Sub Class

- Lets look at the derived class *clsEmployee*:

Example 9 (SubClass):

- Declaring the SubClass:

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

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

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

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

Example 9 (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

    Public Sub New(ByVal N As String, ByVal IDNum As Integer, ByVal BDate As Date, _
    ByVal HDate As String, ByVal Sal As Double)

        MyBase.New(N, IDNum, BDate)

        Me.HireDate = HDate
        Me.Salary = Sal
    End Sub
    '*********************************************************************
    'Regular Class Methods
    Public Sub PrintEmployee()
        'Call Inherited Print Method to display Base Class values
        MyBase.Print()

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

    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
    '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
```

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, NOT THE BASE 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:
  a. Base Class: Declaration only of Abstract or Virtual function using keyword **MustOverride**.
  b. Sub Class: You must implement or create the method using the keyword: **Overrides**

Example 10 – MustOverride Keyword

Creating the Base Class

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

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_Name As String
    Private m_IDNumber As Integer
    Private m_BirthDate As Date

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

    Public Property IDNumber() As Integer
        Get
            Return m_IDNumber
        End Get
        Set(ByVal Value As Integer)
            m_IDNumber = Value
        End Set
    End Property
    'We allow Property to be Overridden
    Public Overridable Property BirthDate() As Date
        Get
            Return m_BirthDate
        End Get
        Set(ByVal Value As Date)
            m_BirthDate = 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_Name = ""
    m_IDNumber = 0
    m_BirthDate = #1/1/1900#
End Sub


Public Sub New(ByVal N As String, ByVal IDNum As Integer, ByVal BDate As Date)
 'Note that we are NOT using the private data but the Property Procedures instead
    Me.Name = N
    Me.IDNumber = IDNum
    Me.BirthDate = BDate
End Sub

'*********************************************************************
'Regular Class Methods
'We allow Method to be Overridden
Public Overridable Sub Print()
    MessageBox.Show("Printing BASE CLASS Person Data " _
    & m_Name & ", " & m_IDNumber & ", " & _
    m_BirthDate)
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 purchasedItems 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 10 (SubClass):

- Declaring the SubClass:

```
Option Explicit On
Public Class clsEmployee
    Inherits clsPerson
    '*********************************************************************
    '*********************************************************************
    '*********************************************************************
    'Class Data or Variable declarations
    Private m_HireDate As Date
    Private m_Salary As Double
    Private m_TotalItemsPurchased As Integer

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

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

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

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

Example 10 (SubClass-(Cont)):

- Declaring the SubClass Methods:

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

        Me.HireDate = #1/1/1900#
        Me.Salary = 0.0
        m_TotalItemsPurchased = 0

    End Sub

    Public Sub New(ByVal N As String, ByVal IDNum As Integer, ByVal BDate As Date, _
    ByVal HDate As String, ByVal Sal As Double)

        MyBase.New(N, IDNum, BDate)

        Me.HireDate = HDate
        Me.Salary = Sal
    End Sub

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

        'Now display Derived Class values
        MessageBox.Show("Printing Employee Data " _
        & m_HireDate & ", " & m_Salary & ", " & m_TotalItemsPurchased)

    End Sub


   '*********************************************************************
    'Shop() Method must be implemented, even if we leave the body empty
    'In this case we implement and add code to the body of the method.
    'Note that the keyword Overrides must be used since it's declared
    'MustOverride in Base Class
    Public Overrides Sub Shop(ByVal purchasedItems As Integer)

        m_TotalItemsPurchased = m_TotalItemsPurchased + purchasedItems
    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
```

1.2.10 Sample Program #1 – Employee Management & Authentication

- In this example, we demonstrates some of the Inheritance features shown in this lecture notes. In the example we implement an Employee Management System. The employees are represented by Employee Objects of the ***clsEmployee*** class which is a Sub Class of the Base Class ***clsPerson***. An **Array** is used to manage the employee objects and review some array concepts as well, such as adding, removing, modifying, searching, skipping nothings (empty cells) etc. An **Employees Management Form** is used as the User-Interface to allow users to retrieve, add, edit, remove, print & print all employees. In addition, we will implement an authentication feature using a **login form** to allow employee objects to logon to the system in order to get access to the data. Proper Exception handling is applied and the user prompted accordingly.
- This example program has the following features:

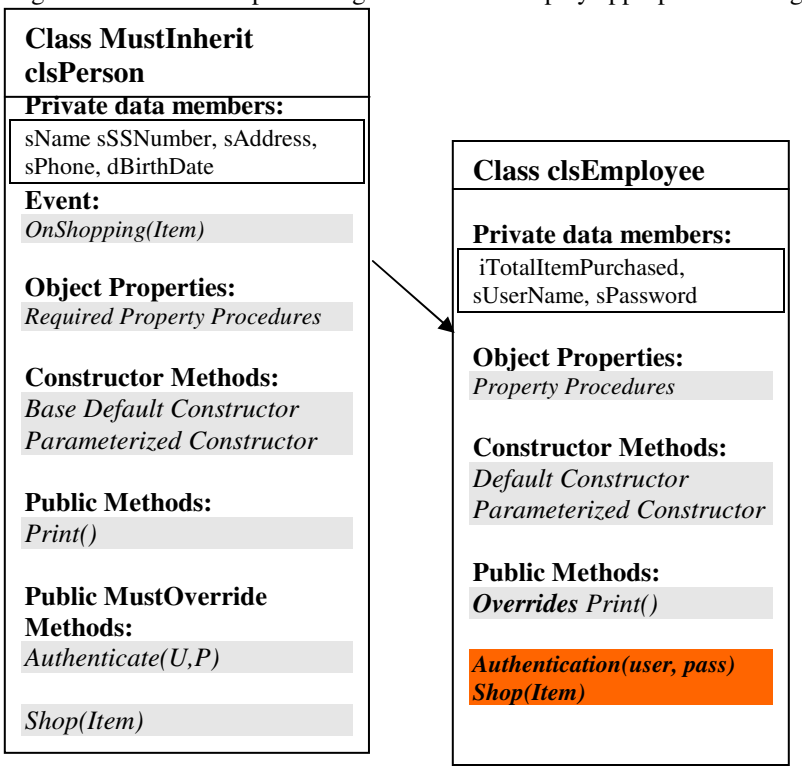
  - Inheritance feature such as: *Basic Inheritance*, *Constructors* in Inheritance, use of *MyBase*, *Overriding* methods, *MustInherit*, & *MustOverride*
  - Employee Management using Array to manage the objects
  - Authentication feature via login form etc.
  - Add exception handling to trap errors.
  - Set **Option Strict ON**, and make sure all data types are properly handled.

- We will continue to keep our application architecture in mind and perform all user interactions in the Form. That is all messages displayed to the user is from the Forms.

Example 1 – Array, Inheritance & Employee Management & Authentication with Exception Handling

Problem statement:

- Create an Employee Management application with authentication.
- Create Employee Management & Login Form to handle interaction with user.
- Create *MustInherit* Base class ***clsPerson*** and derive a sub class ***clsEmployee***. Follow the object model below
- In the Person Class create all data and properties, the function named Authenticate(U,P) which authenticates the object by comparing the username & password passed as arguments and returns a Boolean value indicating if the object is the employee being authenticated.
- Module contains logic in *Sub Main* to perform authentication via a method in the module Public Method *Authenticate(U,P)* which does the work of searching database for user
- In the module, implement the processing methods to ***Add()***, ***Edit()*** ***Search()***, ***Remove()***, ***Print()*** & ***PrintAll()***.
- Add code in Employee Management Form to call processing methods and display appropriate messages to user



HOW IT'S DONE:

Part I – Create The Application:

Step 1: Start a new Windows Application project:

Step 2: Add a Forms to the project for Employee management and Login Form. Add controls as required:

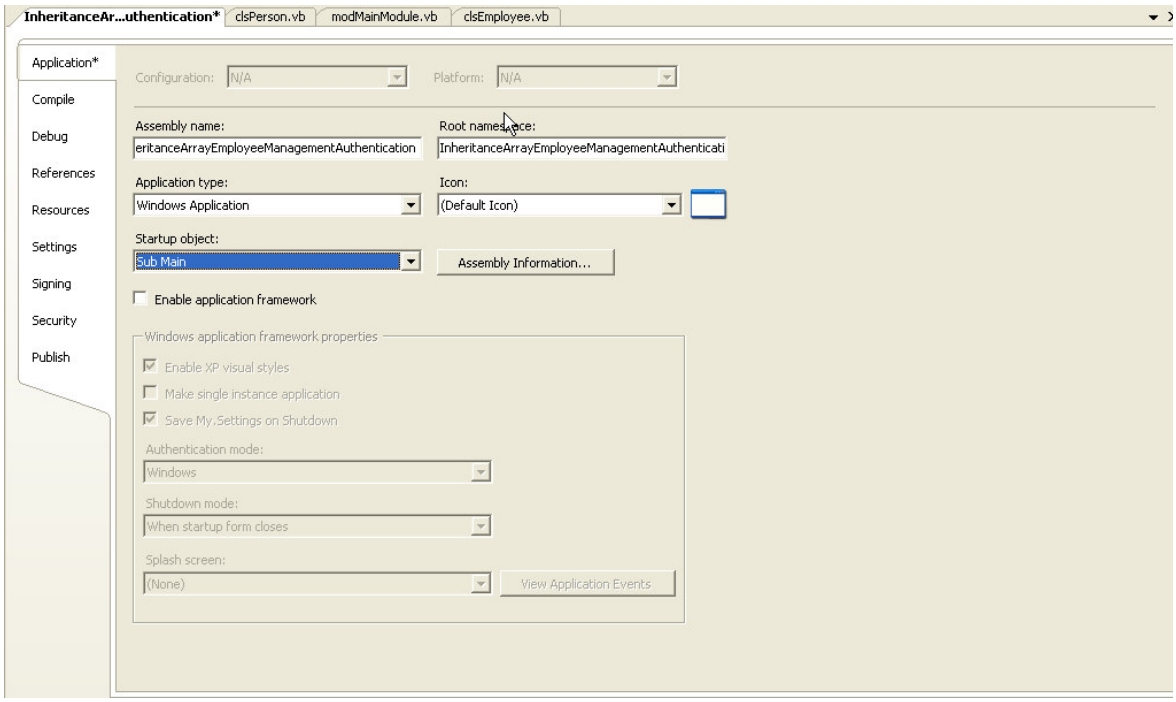
Object	Property	Value
Project	Name	**frmEmployeesForm** **frmLogin**

Step 3: Add a Standard Module set its properties as previous example:

Object	Property	Value
Project	Name	**modMainModule**

Step 4: Set the Project's properties to behave as a Module-Driven Windows Application:

Object	Property	Value
Project	Startup Object	**Sub Main()**



Business Object Layer – Class Objects

Step 5: Create to Reuse the Person Class from Previous Examples, by Copying the File from previous Application Folder to the Folder of this Windows Application Project. IF YOU LIKE YOU CAN SKIP THESE TWO STEPS AND CREATE THE CLASS FROM SCRATCH.

1. Using Windows Explorer, navigate to the Employees Application folder of the previous example.
2. Copy/Paste the file clsPerson.vb, to this Project folder

Step 6: Add the Class to the Project

1. In the Project Menu, select Add Existing Item… and navigate to the project folder
2. Select the ***clsPerson.vb*** File and click OK
3. The class is now part of the project and ready to be reused!

Step 7: Modify the Class as necessary to follow Object Model of this Example

```
Option Explicit On
Option Strict On

'Impoted Libraries
Imports System.IO  'For file access code

Public MustInherit Class clsPerson
    '*******************************************************************
    'Class Data or Variable declarations
    Private m_Name As String
    Private m_SSNumber As String
    Private m_BirthDate As Date
    Private m_strAddress As String
    Private m_Phone As String
```

Step 8: Property Procedures:

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

Public Property SocialSecurity() As String
    Get
        Return m_SSNumber
    End Get
    Set(ByVal Value As String)
        m_SSNumber = Value
    End Set
End Property

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

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

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

Step 9: Modify the Constructors Methods Accordingly:

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

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

'Parameterized Constructor
Public Sub New(ByVal N As String, ByVal SSNum As String, ByVal BDate As Date, _
ByVal Adr As String, ByVal Ph As String)
    'Note that as example we are NOT using the private data but
    'the Property Procedures instead when setting the data via the constructor

    Name = N
    SocialSecurity = SSNum
    BirthDate = BDate
    Address = Adr
    Phone = Ph

End Sub
```

Step 10: Modify PrintPerson() Method to Save to an EmployeePrinter File:

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

'Author of base class allows sub classes to overide Print()
'If they want to, it is not mandatory
Public Overridable Sub Print()
    'Create StreamWriter Object for append to file listed
    Dim objPrinter As New StreamWriter("PersonPrinter.txt", True)

    'Call StreamWriter Object WriteLine method to write the string to file
    objPrinter.WriteLine(m_Name & ", " & m_SSNumber & ", " & _
    m_BirthDate & ", " & m_Address & ", " & m_Phone)

    'Close StreamWriter Object
    objPrinter.Close()
End Sub
```

Step 11: MustOverride Methods: Authenticate(u,p) & Shop():

```
    '*********************************************************************
    'Declaration of MustOverride Methods (Note that there is no End Sub)
    'These methods are also Known as Abstract Methods or Virtual Functions
    'The author of base class is forcing the sub classes to implement
    'these two methods, if they don't they can never compile the sub classes.

    'Must overide Shop()
    Public MustOverride Sub Shop(ByVal itemsPurchased As Integer)

    'Must overide Authenticate()
 Public MustOverride Function Authenticate(ByVal uName As String, ByVal pWord As String)
As Boolean

End Class
```

Step 12: Create a new clsEmployee Class and Inherit from clsPerson. Add Private Data and OnShopping Event

```
Option Explicit On
Option Strict On

'Impoted Libraries
Imports System.IO  'For file access code

Public Class clsEmployee
    Inherits clsPerson
    '*********************************************************************
    'Class Data or Variable declarations
    Private m_TotalItemsPurchased As Integer
    Private m_UserName As String
    Private m_PassWord As String

    'Event Declarations
    Public Event OnShopping(ByVal totalItemsPurchased As Integer)
```

Step 13: Property Procedures:

```
    '*********************************************************************
    'Property Procedures
    Public Property TotalItemsPurchased() As Integer
        Get
            Return m_TotalItemsPurchased
        End Get
        Set(ByVal Value As Integer)
            m_TotalItemsPurchased = Value
        End Set
    End Property

    'Username Property
    Public Property Username() As String
        Get
            Return m_UserName
        End Get
        Set(ByVal Value As String)
            m_UserName = Value
        End Set
    End Property
    'Password Property
    Public Property Password() As String
        Get
            Return m_PassWord
        End Get
        Set(ByVal Value As String)
            m_PassWord = Value
        End Set
    End Property
```

Step 14: Create Constructors Methods, handle Base Class Constructors via MyBase keyword:

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

'Default Constructor
Public Sub New()
    'Call Base Class default constructor
    MyBase.New()

    m_UserName = ""
    m_PassWord = ""
    m_TotalItemsPurchased = 0

End Sub

'Parameterized Constructor
Public Sub New(ByVal N As String, ByVal SSNum As String, ByVal BDate As Date, _
ByVal Addr As String, ByVal Ph As String, ByVal uName As String, _
ByVal pWord As String)

    'Call Base Class parameterized constructor
    MyBase.New(N, SSNum, BDate, Addr, Ph)

    'Note that parameters are set to Property Procedures
    Me.Username = uName
    Me.Password = pWord

    'Note,not part of parameters. No need,
    'new employees don't shop as they are created
    Me.TotalItemsPurchased = 0

End Sub
```

Step 15: Modify PrintPerson() Method to Save to an EmployeePrinter File:

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

'Author of sub class has decided to overide Print()
'This is not mandatory, but author wants to implement a new Print()
Public Overrides Sub Print()
    'Call Inherited PrintPerson Method to display Base Class values
    MyBase.Print()

    'Create StreamWriter Object for append to file listed
    Dim objPrinter As New StreamWriter("EmployeePrinter.txt", True)

    'Call StreamWriter Object WriteLine method to write the string to file
    objPrinter.WriteLine(m_TotalItemsPurchased & ", " & m_UserName & ", " & _
    m_PassWord)

    'Close StreamWriter Object
    objPrinter.Close()

End Sub
```

Step 16: Add the Authenticate() Method:

```
    '*********************************************************************
    'Author of sub class MUST implement this method, it is MANDATORY! since
    'it was forced by base class when declared MustOverride in base class.
    'Point is, If you want to inherit from the base than you must implement
    'this method, even if you leave it blank in the body

    'Authenticate, is a function that accepts two arguments(user & pass),
    'compares these values to it's internal user & pass and returns true
    'if match else false.
Public Overrides Function Authenticate(ByVal uName As String, ByVal pWord As String) As
Boolean

        If m_UserName = uName And m_PassWord = pWord Then
            Return True
        Else
            Return False
        End If

End Function
```

Step 17: Shop() Method. Note that it is not being used in this example:

```
    '*********************************************************************
    'Author of sub class MUST implement this method, it is MANDATORY! since
    'it was forced by base class when declared MustOverride in base class.
    'Point is, If you want to inherit from the base than you must implement
    'this method, even if you leave it blank in the body

    'Shop Method adds items passed as argument to total and raises the OnShopping Event
    Public Overrides Sub Shop(ByVal totalItemsPurchased As Integer)
        m_TotalItemsPurchased = m_TotalItemsPurchased + totalItemsPurchased

        'Raise or trigger event & send information with the event
        RaiseEvent OnShopping(m_TotalItemsPurchased)

    End Sub

End Class
```

Presentation Layer (UI) – Module & Forms

Part II – Module

Overview

- We will add Exception Handling using Try/Catch Blocks to trap for general errors generated.

Step 1: In Module Add the Following Code:

- Code any Global & Private Variable declarations and Sub Main()
  1. **Option Strict ON**.
  2. Import the *System.Collections* Library to support the File I/O features
  3. Use a Array to store employees
  4. Declare Global Employee Form Object & Login Form object
  5. Add methods to support the processing required by the forms; Add, Edit, Search, Remove, Print, Print All objects, & Authenticate method to search and authenticate an employee.

```
Option Explicit On
Option Strict On


Module modMainModule

    'Declare Constant SIZE for use by arrays
    Private Const SIZE As Integer = 10

    'Declare Public Array to store Employee Objects
    'Represents the database of Employees
    Public arrEmployeeList(SIZE) As clsEmployee

    'Form objects Declarations
    Public objLoginForm As frmLogin = New frmLogin
    Dim objEmployeeForm As frmEmployeesForm = New frmEmployeesForm
```

Step 2: Sub Main:

```
    '*****************************************************************************
'Name:          Main Method                                                      *
'Purpose:       Execution starup point.                                          *
'Algorithm:     Step 0-Perfom initialization                                     *
'               Step 1-Displays login form and gets username & password          *
'               Step 2-Begin loop, end loop when user & pass = -1                *
'               Step 3-Call Module Authenticate Function to search and authenticate *
'               Step 4-Step 4-Based on results of authenticate either display form *
'               Step 5-Display Login Form & extract Values from Form             *
'*****************************************************************************
Public Sub Main()
        Dim userName, passWord As String
        Dim isAuthenticated As Boolean

        'Step 0-Perfom initialization (populate array with objects)
        InitializeList()

        'Step 1-Display Login Form & extract user/pass values
        'Note: This block of code only runs once, at the begining
        'Display Login Form
        objLoginForm.ShowDialog()
        'After Login form hides, extract Data from login Form
        userName = objLoginForm.txtUsername.Text
        passWord = objLoginForm.txtPassword.Text

        'Step 2-Loop if user/pass are not -1
        Do While (userName <> "-1" And passWord <> "-1")

  'Step 3-Call Module Authenticate Function to search database and authenticate the user
            isAuthenticated = Authenticate(userName, passWord)

            'Step 4-Based on results of authenticare either display form
            'or prompt & reject user
            If isAuthenticated Then
                objEmployeeForm.ShowDialog()
            Else
                MessageBox.Show("Access Denied")
            End If

            'Step 5-Display Login Form & extract Values from Form
            'Note: This block of code runs as many times as the loop
            objLoginForm.ShowDialog()
            userName = objLoginForm.txtUsername.Text
            passWord = objLoginForm.txtPassword.Text
        Loop

    End Sub
```

Step 3: IntializeList Method – Create Objects, Initialized and add to list:

```
'*****************************************************************************************
'Name:          InitializeList() Method                                                 *
'Purpose:       Populates Collection object with an object                              *
'Algorithm:     Step 1-Creates temp objects populated with data                         *
'               Step 2-Add objects to array                                             *
'*****************************************************************************************
Public Sub InitializeList()

'Declare Object Pointers
Dim objE1 As clsEmployee
Dim objE2 As clsEmployee
Dim objE3 As clsEmployee
Dim objE4 As clsEmployee
Dim objE5 As clsEmployee


'Create and initialize Objects with data via Constructors
objE1 = New clsEmployee("Joe", "111", #12/12/1965#, "111 Jay Street", "718-434-5544",
"joe", "111")

objE2 = New clsEmployee("Angel", "222", #1/4/1972#, "222 Flatbush Ave", "718-234-5524",
"angel", "222")

objE3 = New clsEmployee("Sam", "333", #9/21/1960#, "333 Dekalb Ave", "718-890-3422",
"sam", "333")

objE4 = New clsEmployee("Mary", "444", #7/4/1970#, "444 Jay Street", "718-444-1122",
"mary", "444")

objE5 = New clsEmployee("Nancy", "555", #12/12/1965#, "555 Flatlands Ave", "718-434-9876",
"nancy", "555")


        'Add objects to Array
        arrEmployeeList(0) = objE1
        arrEmployeeList(1) = objE2
        arrEmployeeList(2) = objE3
        arrEmployeeList(3) = objE4
        arrEmployeeList(4) = objE5


End Sub
```

Step 4: Add Module Level Authenticate() Method:

```
    '*****************************************************************************
    'Name:          Function Authenticate() Method                              *
    'Purpose:       Search the Array & authenticate by interrogating every object *
    '               Trap any general exception errors                           *
    '               Throw an ArgumentExecption. This Throw must be trapped in Form *
    '*****************************************************************************
Public Function Authenticate(ByVal strUser As String, ByVal strPass As String) As Boolean
        'Step 0-Loop index variable
        Dim i As Integer
        'Step 1-Begins Exeception handling.
        Try
            'Step 2-Use For loop to iterate through array
            For i = 0 To SIZE
                If Not arrEmployeeList(i) Is Nothing Then
                    'Step 3-Call each object's authenticate method
                    If arrEmployeeList(i).Authenticate(strUser, strPass) Then
                        'Step 4-Object found, return and exit function
                        Return True
                    End If
                End If

            Next
            'Step 5-End of Search, Object not found, return False & Exit
            Return False

            'Step 6-Traps for General exceptions.
        Catch objE As Exception
          'Step 7-Throw an Execption to calling programs. Throw Must be trapped in Form
            Throw New System.Exception(objE.Message)
        End Try
    End Function
```

Step 5: Implement the Search method to manage the retrieval of objects from the Array:

```
'*****************************************************************************
'Name:          Function - Search(Key)Method                                 *
'Purpose:       Retrieves object pointer from array.                         *
'               Trap any general exception errors                            *
'               Throw an ArgumentExecption. This Throw must be trapped in Form   *
'*****************************************************************************
Public Function Search(ByVal strSSNum As String) As clsPerson
    'Step 0-Loop index variable
    Dim i As Integer
    'Step 1-Begins Exeception handling.
    Try
        'Step 2-Search array
        For i = 0 To SIZE
            'Step 3-Skip empty cells
            If Not arrEmployeeList(i) Is Nothing Then
                'Step 4-Ask object who it is
                If arrEmployeeList(i).SocialSecurity = strSSNum Then
                    'Step 5-Found object so return pointer
                    'to object inside Array at index i. Function exits as well
                    Return arrEmployeeList(i)
                End If
            End If
        Next i
        'Step 6-Return an empty object since searched entire array and not found
        Return Nothing

        'Step 7-Traps for General exceptions.
    Catch objE As Exception
    'Step 8-Throw an Execption to calling programs. Throw must be trapped in Form
        Throw New System.Exception(objE.Message)
    End Try
End Function
```

Step 6: Implement the Add method to manage the addition of objects into the list:

```
'*****************************************************************************
'Name:          Add(value1, value2..)Function Method                          *
'Purpose:       Adds new object to the array.                                 *
'               Trap any general exception errors                             *
'               Throw an ArgumentExecption. This Throw must be trapped in Form *
'*****************************************************************************
Public Function Add(ByVal strName As String, ByVal strSSNum As String, _
ByVal dBDate As Date, ByVal strAddress As String, ByVal strPhone As String, _
ByVal strUser As String, ByVal strPass As String) As Boolean
    'Step 0-Loop index variable
    Dim i As Integer
    'Step 1-Creates Temp Object
    Dim objTempEmployee As New clsEmployee

    'Step 2-Begins Exeception handling.
    Try

        'Step 3-Populates object it with data passed as argument
        With objTempEmployee
            .Name = strName
            .SocialSecurity = strSSNum
            .BirthDate = dBDate
            .Address = strAddress
            .Phone = strPhone
            .Username = strUser
            .Password = strPass
        End With

        'Step 4-Search array
        For i = 0 To SIZE
       'Step 5-Find first avalilable empty cells asking if is a nothing or empty
            If arrEmployeeList(i) Is Nothing Then
                'Step 6-Found empty cell, assign temp object to empty cell
                'simple pointer assingment
                arrEmployeeList(i) = objTempEmployee
                'Step 7-Retrun true & exit function
                Return True
            End If
        Next i

        'Step 8-Delete Temp Object
        objTempEmployee = Nothing

        'Step 9-Return false since searched entire array and found NO empty cells
        Return False

        'Step 10-Traps for General exceptions
    Catch objE As Exception
        'Step 11-Throw General Exception. This Throw must be trapped in Form
        Throw New System.Exception(objE.Message)
    End Try
End Function
```

Step 7: Implement the EditItem method to manage the process of modifying objects in the list:

```
'*****************************************************************************
    'Name:          EditItem(value1, value2..) Function Method                 *
    'Purpose:       Sets or overwrites object located at specified location in array *
    '               Trap any general exception errors                          *
    '               Throw an ArgumentExecption. This Throw must be trapped in Form   *
    '*****************************************************************************
    Public Function Edit(ByVal strName As String, ByVal strSSNum As String, _
    ByVal dBDate As Date, ByVal strAddress As String, ByVal strPhone As String, _
    ByVal strUser As String, ByVal strPass As String) As Boolean
        'Step 0-Loop index variable
        Dim i As Integer

        'Step 1-Begins Exeception handling.
        Try

            'Step 2-Search array
            For i = 0 To SIZE
                'Step 3-Skip empty cells
                If Not arrEmployeeList(i) Is Nothing Then
                    'Step 4-Ask object who it is
                    If arrEmployeeList(i).SocialSecurity = strSSNum Then
                        'Step 5-Found object so modify it by setting properties
                        'Note DO NOT modify SSNumber property, this is the key
                        arrEmployeeList(i).Name = strName
                        arrEmployeeList(i).BirthDate = dBDate
                        arrEmployeeList(i).Address = strAddress
                        arrEmployeeList(i).Phone = strPhone
                        arrEmployeeList(i).Username = strUser
                        arrEmployeeList(i).Password = strPass

                        'Step 6-Retrun true & exit function
                        Return True
                    End If
                End If
            Next i

            'Step 7-Return false since searched entire array and did not find object
            Return False

            'Step 8-Traps for General exceptions
        Catch objE As Exception

            'Step 9-Throws an Exeption.  This execption must be trapped in the Form
            Throw New System.Exception(objE.Message)
        End Try

    End Function
```

Step 8: Implement the Remove method to manage the removal of objects from the list:

```
'*****************************************************************************
'Name:          Remove(Key) Function Method                                  *
'Purpose:       Remove object from collection based on key.                  *
'               Trap any general exception errors                            *
'               Throw an ArgumentExecption. This Throw must be trapped in Form  *
'*****************************************************************************
Public Function Remove(ByVal strSSNum As String) As Boolean
    'Step 0-Loop index variable
    Dim i As Integer

    'Step 1-Begins Exeception handling.
    Try

        'Step 2-Search array
        For i = 0 To SIZE
            'Step 3-Skip empty cells
            If Not arrEmployeeList(i) Is Nothing Then
                'Step 4-Ask object who it is
                If arrEmployeeList(i).SocialSecurity = strSSNum Then
                    'Step 5-Found object so delete it by setting pointer to nothing
                    arrEmployeeList(i) = Nothing
                    'Step 6-Retrun true & exit function
                    Return True
                End If
            End If
        Next i

        'Step 7-Return false since searched entire array and did not find object
        Return False

        'Step 8-Traps for General exceptions
    Catch objE As Exception
        'Step 9-Throws an Exception.  This Throw must be trapped in Form
        Throw New System.Exception(objE.Message)
    End Try
End Function
```

Step 9: Implement the Print method to manage the process of printing an objects to File:

```
'*****************************************************************************
'Name:          Print(Key) Function Method                                   *
'Purpose:       Prints object from array to Printer File.                    *
'               Trap any general exception errors                            *
'               Throw an ArgumentExecption. This Throw must be trapped in Form  *
'*****************************************************************************
Public Function Print(ByVal strSSNum As String) As Boolean
    'Step 0-Loop index variable
    Dim i As Integer
    'Step 1-Begins Exeception handling.
    Try

        'Step 2-Search array
        For i = 0 To SIZE
            'Step 3-Skip empty cells
            If Not arrEmployeeList(i) Is Nothing Then
                'Step 4-Ask object who it is
                If arrEmployeeList(i).SocialSecurity = strSSNum Then
                    'Step 5-Found object calls Print() to print the object to file
                    arrEmployeeList(i).Print()

                    'Step 6-Retrun true & exit function
                    Return True
                End If
            End If
        Next i

        'Step 7-Return false since searched entire array and did not find object
        Return False

        'Step 8-Traps for General exceptions
    Catch objE As Exception
        'Step 9-Throws an Exception.  This Throw must be trapped in Form
        Throw New System.Exception(objE.Message)
    End Try
End Function
```

Step 10: Implement the PrintAll method to print all the Employees in the list to File:

```
'*****************************************************************************
    'Name:          PrintAll()Sub  Method                                     *
    'Purpose:       Prints ALL objects from array to Printer File.            *
    '               Trap any general exception errors                         *
    '               Throw an ArgumentExecption. This Throw must be trapped in Form   *
    '*****************************************************************************
    Public Sub PrintAll()
        'Step 0-Loop index variable
        Dim i As Integer

        'Step 1-Begins Exeception handling.
        Try

            'Step 2-Search array
            For i = 0 To SIZE
                'Step 3-Skip empty cells
                If Not arrEmployeeList(i) Is Nothing Then
                    'Step 4-calls Print() Method to print the object to file
                    arrEmployeeList(i).Print()
                End If
            Next i

            'Step 5-Traps for General exceptions
        Catch objE As Exception
            'Step 6-Throws an Exception.  This Throw must be trapped in Form
            Throw New System.Exception(objE.Message)
        End Try

    End Sub


End Module
```

Brief Discussion of Module Code

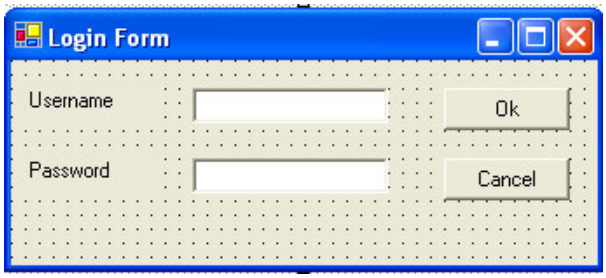
- Processing Methods were used to manage the Array, we added Try/Catch blocks to trap the errors generated.
- We kept all PROCESSING code in the Module.
- No user interface code in Module, only in Sub Main

Part III – User Interface Form

Overview

- We will add Try/Catch Block to the Form in order to trap the errors generated by the Method that manage the Collection in the Module.

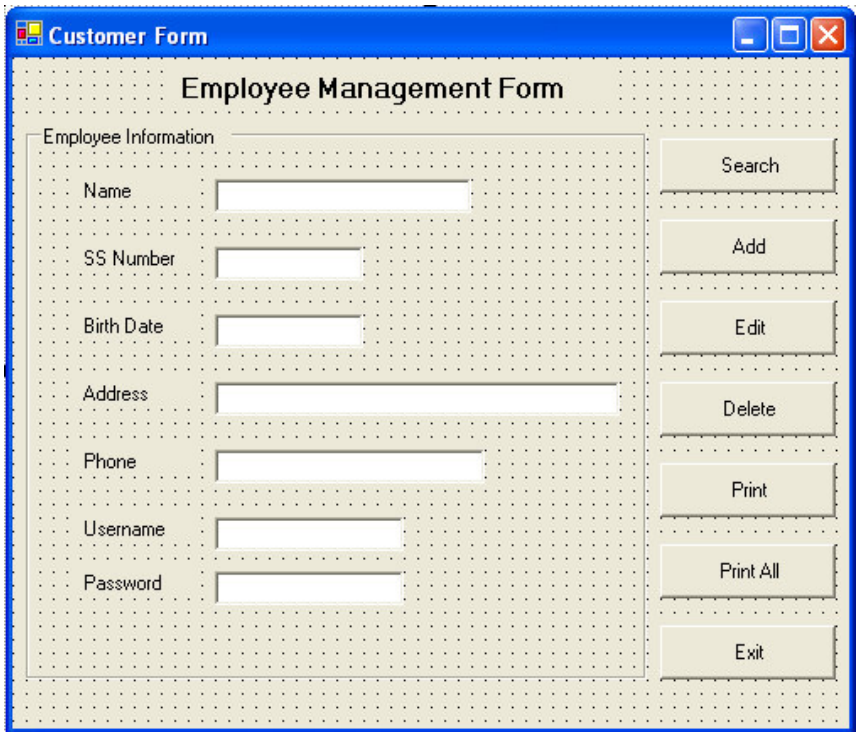
Step 1: Add the required controls to the Login Form:



Step 2: Add the required Code to the Login Form:

```
Private Sub btnOK_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnOK.Click

        'Hides
        Me.Hide()

End Sub


Private Sub btnCancel_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnCancel.Click


     'Clear the text boxes
        txtUsername.Text = ""
        txtPassword.Text = ""

End Sub
```

Step 3: Add the required controls to the Login Form:



Step 4 In the Form frmEmployeesForm Add code for a Module Level Object, Load(), Close() & Exit() handlers:

```
    '*****************************************************************************
    'Name:          Event-Handler Form_Load                                     *
    'Purpose:       Automatically executes when Form is displayed               *
    'Algorithm:     Creates object, disable previous purchase text box          *
    '*****************************************************************************
Private Sub frmEmployeeForm_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

End Sub


    '*****************************************************************************
    'Name:          Event-Handler for Form_Close()                              *
    'Purpose:       Automatically executes when Form is closed and destroys object  *
    'Algorithm:     Destroys object                                             *
    '*****************************************************************************
Private Sub frmEmployeeForm_Closed(ByVal sender As Object, ByVal e As System.EventArgs)
Handles MyBase.Closed

End Sub

    '*****************************************************************************
    'Name:          Event-Handler for for OK button                             *
    'Purpose:       Closes the Form                                             *
    'Algorithm:     Calls this Form's Close() method to close the Form.         *
    '*****************************************************************************
Private Sub btnExit_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnExit.Click

        Me.Close()
End Sub
```

Step 5: Code Search Event-Handler:

```
        '*****************************************************************************
    'Name:          Event-Handler for btnSearch button                                *
    'Purpose:       To retrieve an object from the array base on ID or Key            *
    'Algorithm:     Calls Search() method to get the object.                          *
    '               Traps for General exceptions and displays appropriate messages    *
    '*****************************************************************************
Private Sub btnSearch_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnSearch.Click
        'Step 0-Declare Temp Object
        Dim objEmployee As clsEmployee

        'Step 1-Begins Exeception handling.
        Try

            'Step 2-Calls Search() method with Key passed as argument from texbox
            'Method returns pointer which is assigned to temp Employee Object pointer
            objEmployee = Search(txtSSNumber.Text)

            'Step 3-Verify if Employee not found
            If objEmployee Is Nothing Then
                MessageBox.Show("Employee Record Not Found")

                'Step 4-Clear all textbox controls
                txtName.Text = ""
                txtSSNumber.Text = ""
                txtBirthDate.Text = ""
                txtAddress.Text = ""
                txtPhone.Text = ""
                txtUser.Text = ""
                txtPass.Text = ""

            Else
                'Step 5-Data extracted from Employee object & displayed on Form
                With objEmployee
                    txtName.Text = .Name
                    txtSSNumber.Text = .SocialSecurity
                    txtBirthDate.Text = .BirthDate
                    txtAddress.Text = .Address
                    txtPhone.Text = .Phone
                    txtUser.Text = .Username
                    txtPass.Text = .Password
                End With
            End If

            'Step 6-Delete Temp Object
            objEmployee = Nothing

            'Step 7-Traps for General exceptions and displays appropriate message
        Catch objE As Exception
            MessageBox.Show("Search Error: " & objE.Message)
        End Try

End Sub
```

Step 6: Enter Code for the Add_Click Event-handler:

```
    '*****************************************************************************
    'Name:          Event-Handler for btnAdd button                              *
    'Purpose:       To add new object to the array.                              *
    'Algorithm:     Calls Add() method of module, textboxes data passed as argument  *
    '               Traps for General exceptions and displays appropriate messages   *
    '*****************************************************************************
Private Sub btnAdd_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnAdd.Click

        'Step 0-Declare variable to store result of function call
        Dim bfound As Boolean

        Try

            'Step 1-Calls Add() method of module, textboxes data is passed as argument
bfound = Add(txtName.Text, txtSSNumber.Text, txtBirthDate.Text,   txtAddress.Text, _
            txtPhone.Text, txtUser.Text, txtPass.Text)

            'Step 2-test results and prompt user appropriately
            If bfound Then
                MessageBox.Show("New Employee Record Added to Database")
            Else
                MessageBox.Show("Database FULL!")
            End If

            ' Step 3-Traps for General exceptions and displays appropriate messages
        Catch objE As Exception
            MessageBox.Show("Add Error: " & objE.Message)
        End Try
End Sub
```

Step 7: Enter code for Edit Event:

```
'*****************************************************************************
    'Name:          Event-Handler for btnEdit button                            *
    'Purpose:       Initiate the Edit process to modify an object in the array  *
    'Algorithm:     Call Module Edit() method, pass Textboxes data as argument  *
    '               Traps for General exceptions and displays appropriate messages *
    '*****************************************************************************
Private Sub btnEdit_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnEdit.Click
        'Step 0-Declare variable to store result of function call
        Dim bfound As Boolean

        Try

            'Step 1-Call Module Edit() method, pass Textbox data as argument
            bfound = Edit(txtName.Text, txtSSNumber.Text, txtBirthDate.Text, _
            txtAddress.Text, txtPhone.Text, txtUser.Text, txtPass.Text)

            'Step 2-test results and prompt user appropriately
            If bfound Then
                MessageBox.Show("Employee record Modified")
            Else
                MessageBox.Show("Employee record Not found in database")
            End If

            'Step 3-Traps for General Error and displays appropriate messages
        Catch objE As Exception
            MessageBox.Show("Edit Error: " & objE.Message)
        End Try

End Sub
```

Step 8: Enter Code for the Delete_Click Event:

```
        '*****************************************************************************
    'Name:          Event-Handler for btnDelete button                              *
    'Purpose:       Delete an object from the array base on ID or Key               *
    'Algorithm:     Calls Remove() method of module. Key passed as argument.        *
    '               Traps for General exceptions and displays appropriate messages  *
    '*****************************************************************************
Private Sub btnDelete_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnDelete.Click
        'Step 0-Declare variable to store result of function call
        Dim bfound As Boolean

        Try

            'Step 1-Calls Remove() method of module. ID/Key from texbox passed as argument
            bfound = Remove(txtSSNumber.Text)

            'Step 2-test results and prompt user appropriately
            If bfound Then
                MessageBox.Show("Employee Record Deleted")
            Else
                MessageBox.Show("Employee record Not found in database")
            End If


            'Step 3-Traps for General exceptions and displays appropriate messages
        Catch objE As Exception
            MessageBox.Show("Delete Error: " & objE.Message)
        End Try
End Sub
```

Step 9: Add Code for Print Event:

```
'*****************************************************************************
    'Name:          Event-Handler for btnPrint button                          *
    'Purpose:       Prints object to file                                      *
    'Algorithm:     Call Module PrintEmployee() method to print to file.       *
    '               Traps for General exceptions and displays appropriate messages  *
    '******************************************************************************
Private Sub btnPrint_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnPrint.Click
        'Step 0-Declare variable to store result of function call
        Dim bfound As Boolean

        Try

            'Step 1-Calls Print() method of module. Key from texbox passed as argument
            bfound = Print(txtSSNumber.Text)

            'Step 2-test results and prompt user appropriately
            If bfound Then
                MessageBox.Show("Employee Record Printed")
            Else
                MessageBox.Show("Employee record Not found in database")
            End If

            'Step 3-Traps for General exceptions and displays appropriate messages
        Catch objE As Exception
            MessageBox.Show("Print Error: " & objE.Message)
        End Try
End Sub
```

Step 10: Add Code for PrintAll Event:

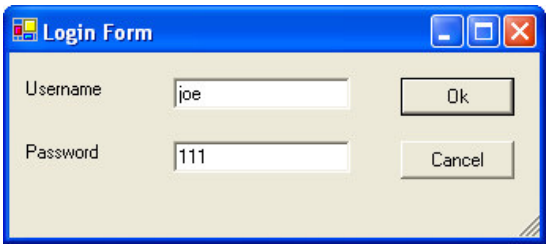
```
'*****************************************************************************
    'Name:          Event-Handler for btnPrintAll button                       *
    'Purpose:       Prints all Objects in the list                             *
    'Algorithm:     Calls PrintAllEmployees() method of module to perform the work.  *
    '               Traps for General exceptions and displays appropriate messages  *
    '******************************************************************************
Private Sub btnPrintAll_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnPrintAll.Click

        Try

            'Step 1-Calls PrintAllEmployees() method of module.
            PrintAll()

            'Step 2-Traps for General exceptions and displays appropriate message
        Catch objE As Exception
            MessageBox.Show("Print All Error: " & objE.Message)
        End Try
End Sub
```

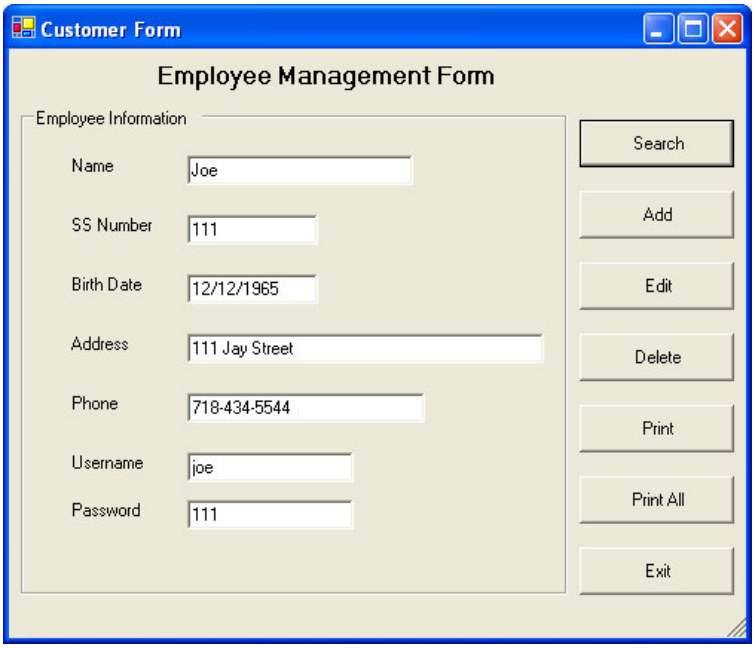
Part IV – Output & Summary

Summary

- Run the program and you can then perform the necessary operations on the list.

Form Output:





File Output: