Visual Basic.NET Introduction

ROAD MAP

- Introduction
- Building a class
- A “Real” Class
- Inheritance
- Polymorphism
- Object Constructors and Destructors

INTRODUCTION

- Classes are at the very heart of Visual Studio
- Classes are used routinely in team development
- The major driving force behind object-oriented programming
  - *code reuse*

Class

- a program that doesn’t run on its own
- Must be used by another application
- similar to Windows controls
  - only they don’t have a visible interface

Building the Minimal Class

- May reside in the same file as a Form
- Customary to implement custom classes in a separate module
  - a Class module
- a class doesn't run on its own
  - can't test it without a form

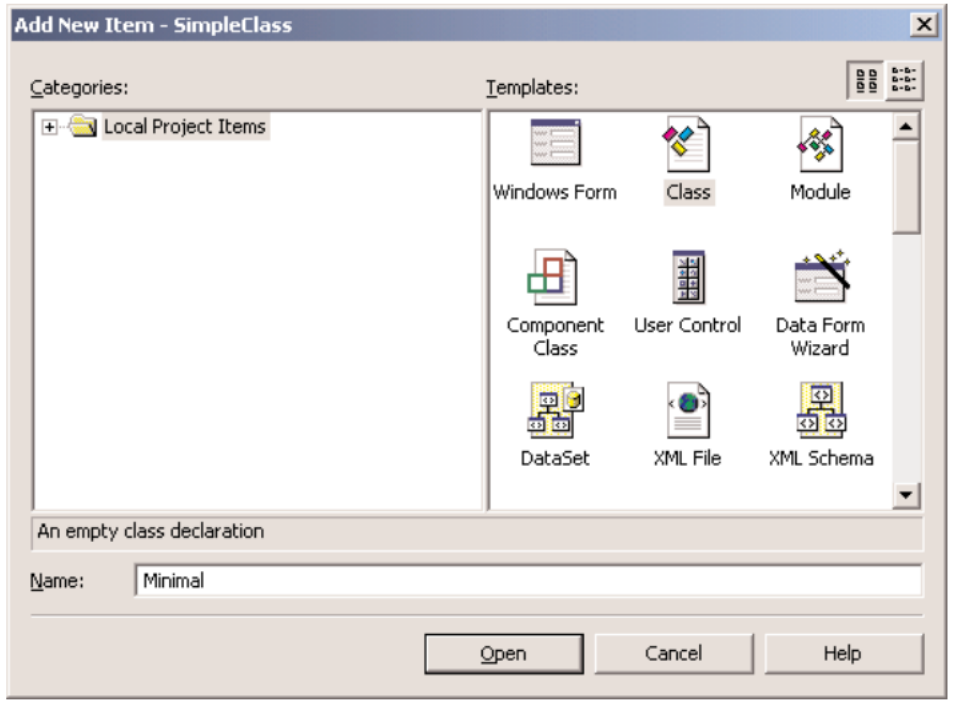
- Create a Windows application
- Add the class to it
- Test it by adding the appropriate code to the form

Building a Minimal Class

- Start a new class - -simple class

FIGURE 8.1

Adding a Class item to a project



Adding Code to the Minimal Class

- *property1* (a String)
- *property2* (a Double)

```
Public property1 As String, property2 As Double
```

- ReverseString method
  - reverses the order of the characters in *property1*
  - Return New String
- NegateNumber method
  - returns the negative of *property2*
  - Don't accept arguments

Adding Code to the Minimal Class

LISTING 8.1: ADDING A FEW MEMBERS TO THE MINIMAL CLASS

```
Public Class Minimal
   Public property1 As String, property2 As Double
   Public Function ReverseString() As String
      Return (StrReverse(property1))
   End Function
   Public Function NegateNumber() As Double
      Return (-property2)
   End Function
End Class
```

Adding Code to the Minimal Class

- Test

Listing 8.2: Testing the Minimal Class

```
Dim obj As New Minimal()
obj.property1 = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
obj.property2 = 999999
Console.WriteLine(obj.ReverseString)
Console.WriteLine(obj.NegateNumber)
```

Property Procedures

- accept any value
  - as long as the type is correct and the value of the numeric property is within the acceptable range
- Generic properties were meaningful entities
  - Using Property Procedures

LISTING 8.3: IMPLEMENTING PROPERTIES WITH PROPERTY PROCEDURES

```
Private tAge As Integer
Property Age() As Integer
   Get
      Age = tAge
   End Get
   Set (ByVal Value As Integer)
      If Value < 0 Or Value >= 125 Then
         MsgBox("Age must be positive and less than 125")
      Else
         tAge = Value
      End If
   End Set
End Property
```

Raising Exceptions

- To receive an exception and handle it from within your code

LISTING 8.4: THROWING AN EXCEPTION FROM WITHIN A PROPERTY PROCEDURE

```
Private tAge As Integer
Property Age() As Integer
   Get
      Age = tAge
   End Get
   Set (ByVal Value As Integer)
      If Value < 0 Or Value >= 125 Then
         Dim AgeException As New ArgumentException()
```

```
         Throw AgeException
      Else
         tAge = Value
      End If
   End Set
End Property
```

Catching Exception

LISTING 8.5: CATCHING THE AGE PROPERTY'S EXCEPTION

```
Dim obj As New Minimal()
Dim userAge as Integer
UserAge = InputBox("Please enter your age")
Try
   obj.Age = userAge
Catch exc as ArgumentException
   MsgBox("Can't accept your value, " & userAge.ToString & VbCrLf & _
          "Will continue with default value of 30")
   obj.Age = 30
End Try
```

Customizing Default Members

- Default members when building a class
  - ToString method
- Provide your custom implementation for these members
  - Example:

```
Public Overrides Function ToString() As String
   Return "The infamous Minimal class"
End Function
```

Customizing Default Members

- Implementing of a custom equal method

LISTING 8.10: A CUSTOM EQUALS METHOD

```
Public Overloads Function Equals(ByVal obj As Object) As Boolean
   Dim O As Minimal = CType(obj, Minimal)
   If O.BDate = tBDate Then
      Equals = True
   Else
      Equals = False
   End If
End Function
```

A “Real” Class

- A more practical class
- ExtractPathName and ExtractFileName methods
  - extract the file and path name from a full filename
- Num2String Method
  - converts a numeric value (an amount) to the equivalent string

Parsing a Filename String

- Implemeting ExtractFileName and ExtractPathName Methods

LISTING 8.23: THE EXTRACTFILENAME AND EXTRACTPATHNAME METHODS

```
Public Function ExtractFileName(ByVal PathFileName As String) As String
   Dim delimiterPosition As Integer
   delimiterPosition = PathFileName.LastIndexOf("\")
   If delimiterPosition > 0 Then
      Return PathFileName.Substring(delimiterPosition + 1)
   Else
      Return PathFileName
   End If
End Function
Public Function ExtractPathName(ByVal PathFileName As String) As String
   Dim delimiterPosition As Integer
   delimiterPosition = PathFileName.LastIndexOf("\")
   If delimiterPosition > 0 Then
      Return PathFileName.Substring(0, delimiterPosition)
   Else
      Return ""
   End If
End Function
```

Reusing the StringTools Class

- Create the class's executable file
- Exclude the test form from the project
- Right-click the name of the test form
- Select Exclude From Project
- Now, only contains the StringTools class

Inheritance

- The ability to create a new class based on an existing one
- Parent class, or base class – the existing class
- Subclass, or, derived class -- the new class (that inherit the base class)
- Overriding
  - The replacement of existing members with other ones

Inherit Existing Class

- Call the new class *myArrayList*

```
Class myArrayList
Inherits ArrayList

End Class
```

- If you don't add a single line of code, the functionality remains same

Inheriting Existing Class

- Add a EliminateDuplicates() subroutine to arraylist

LISTING 8.33: THE ELIMINATEDUPLICATES METHOD FOR THE ARRAYLIST CLASS

```
Sub EliminateDuplicates()
   Dim i As Integer = 0
   Dim delEntries As ArrayList
   While i <= MyBase.Count - 2
      Dim j As Integer = i + 1

    While j <= MyBase.count - 1
       If MyBase.Item(i).ToString = MyBase.item(j).ToString Then
          MyBase.RemoveAt(j)
       End If
       j = j + 1
    End While
    i = i + 1
 End While
End Sub
```

Polymorphism

- The ability of a base type to adjust itself to accommodate many different derived types
- Eg. English word : “run”

- Shape class (parent)
  - Triangular, Circular…..
- Area Method is applied in these classes

```
Dim shape1 As New Square, area As Double
area = shape1.Area
```

- If *shape2* represents a circle

```
Dim shape2 As New Circle, area As Double
area = shape2.Area
```

Polymorphism

- Shape class

LISTING 8.37: THE SHAPE CLASS

```
Class Shape
   Overridable Function Area() As Double
   End Function
   Overridable Function Perimeter() As Double
   End Function
End Class
```

Polymorphism

- Square class

```
Public Class Square
   Inherits Shape
   Private sSide As Double
   Public Property Side() As Double
      Get
         Side = sSide
      End Get
      Set
         sSide = Value
      End Set
   End Property
   Public Overrides Function Area() As Double
      Area = sSide * sSide
   End Function
   Public Overrides Function Perimeter() As Double
      Return (4 * sSide)
   End Function
End Class
```

Polymorphism

- Triangular Class

```
Public Class Triangle
    Inherits Shape
    Private side1, side2, side3 As Double
    Property SideA() As Double
        Get
            SideA = side1
        End Get
        Set
            side1 = Value
        End Set
    End Property
    Property SideB() As Double
        Get
            SideB = side2
        End Get
        Set
            side2 = Value
        End Set
```

Polymorphism

- Triangular Class

```
        End Property
        Public Property SideC() As Double
            Get
                SideC = side3
            End Get
            Set
                side3 = Value
            End Set
        End Property
        Public Overrides Function Area() As Double
            Dim perim As Double
            perim = Perimeter()
            Return (Math.Sqrt(perim * (perim - side1) * (perim - side2) * _
                              (perim - side3)))
        End Function
        Public Overrides Function Perimeter() As Double
            Return (side1 + side2 + side3)
        End Function
    End Class
```

Polymorphism

- Circle Class

```
Public Class Circle
   Inherits Shape
   Private cRadius As Double
   Public Property Radius() As Double
      Get
         Radius = cRadius
      End Get
      Set
         cRadius = Value
      End Set
   End Property
   Public Overrides Function Area() As Double
      Return (Math.Pi * cRadius ^ 2)
   End Function
   Public Overrides Function Perimeter() As Double
      Return (2 * Math.Pi * cRadius)
   End Function
End Class
```

Object Constructors and Destructors

- Recall:
  - objects are created and then disposed of when no longer needed
- To construct an object
  - Declare it and then set it to a new instance of the class it represents
- Example: to construct a triangle

```
Dim shape1 As Triangle = New Triangle()
Dim shape1 As New Triangle()
```

- Specify the properties of an object in the same line that creates the object
  - This is called *constructor*

```
Dim rect1 As Rectangle = New Rectangle(10, 10, 50, 90)
```

Object Constructor and Destructor

- Constructor
  - initialize the object by setting some or all of its properties
- *parameterized* constructor
  - allow you to pass arguments to an object as you declare it
- Constructors are implemented with the New subroutine
  - called every time a new instance of the class is initialized

Object Constructor and Destructor

- Initialize Triangle Class

```
Sub New(ByVal sideA As Double, ByVal sideB As Double, ByVal sideC As Double)
   MyBase.New()
   side1 = sideA
   side2 = sideB
   side3 = sideC
End Sub
```

Object Constructor and Destructor

- parameterized constructor of the Circle class:

```
Sub New(ByVal radius As Double)
    MyBase.New()
    cRadius = radius
End Sub
```

- When we enter "Dim shape1 As New Triangle("

FIGURE 8.11

The members of the various Shape constructors displayed by IntelliSense

```
dim shape1 as New Triangle (
                    New (sideA As Double, sideB As Double, sideC As Double)

dim shape1 as New Square (
                    New (Side As Double)

dim shape1 as New Circle (
                    New (radius As Double)
```

Exercise

- (1) If Full filename is“c:\Documents\Recipes\Chinese\Won Ton.txt”, what will be returned by ExtractFileName method and ExtractPathName method?

- (2) What will be returned by Num2String Method?

$12,544

Answer

- (1) “Won Ton.txt”

  “c:\Documents\Recipes\Chinese\”

- (2) “Twelve Thousand, Five Hundred And Forty Four.”