VB.NET - Orientação a objetos : conceitos básicos em 10 lições.

Lição 1 - Como criar classe e métodos . Instanciando um objeto e acessando o método da classe.

- Para criar uma classe basta usar a palavra reservada **Class <nome da classe>**. A classe termina com a designação **End Class**.
- Na classe você pode criar métodos ou funções usando as palavras reservadas - **Public, Private ,Shared,Friend,Function, Sub <nome do método ou função>**
- A palavra chave **New** gera a instância de um objeto.
- Lembre-se : Um objeto é sempre uma instância de uma classe. *(Não existe objeto sem uma classe)*
- Quando você cria uma variável pública em uma classe esta criando uma **propriedade**.
- Para ter maior controle sobre propriedades você deve criar propriedades por meio de uma rotina. Para fazer isto determine o nome e o tipo de dado da propriedade. Ex: Public Property Nome() As String. Quando você teclar ENTER ao terminar esta linha o VB.NET irá montar a estrutura da propriedade para você conforme abaixo:

```
Public Property Nome()
As String
Get
End Get
Set(ByVal Value As
String)
End Set
End Property
```

```
Namespace Exemplo1

  'criei uma classe - ClasseSimples - e um método public -
MetodoSimples
  Class ClasseSimples

    Public Sub MetodoSimples()
      System.Console.Write("Classe Simples")
    End Sub

  End Class

  'criei outra classe - Prog1 - para em Main acessar o método da classe
anterior via instância do objeto obj.
  Class Prog1

    Public Shared Sub Main()
```

```
    'criei uma instancia do objeto e acessei o método da classe
    Dim obj As New ClasseSimples()
    obj.MetodoSimples()
  End Sub

 End Class

End Namespace
```

lição 2 - escopo de variáveis - Verificando o escopo e visibilidade de variáveis.

- Membros **private** em uma classe são visíveis apenas no escopo da classe. *(variáveis locais)*
- Membros declarados como **public** são visíveis em todo o aplicativo. (*são variáveis globais*)
- Um membro **Protected** só pode ser acessado por classes derivadas mas não pela instância da classe.
- **Shared** indica que um determinado membro da classe é compartilhado e pode ser acessado por todas as instâncias da classe. (*também conhecidos como membros estáticos*)
- **Friend** - Torna um membro da classe visível por todo o projeto , mas não a um controlador de uma instância do objeto.

Dica - *procure criar variáveis o mais próximo possível de onde você vai usá-la.(classe , procedimento , laço, etc.)*

```
Namespace Exemplo2
  Class ClasseSimples2
   'definição do tipo e escopo das variáveis  ( ou propriedades )
    Private MemberVal1 As Long
    Protected MemberVal2 As Long
    Public MemberVal3 As Long
 End Class

 Class Prog2
    Public Shared Sub Main()
       Dim obj As New ClasseSimples2()
       'obj.MemberVal1 = 1  'errado porque a variável é um
membro private
       'obj.MemberVal2 = 2  'errado porque a variável é um
membro protected
      obj.MemberVal3 = 3  'correto porque a variável é um membro
public
   End Sub
 End Class

End Namespace
```

lição 3 - inicialização de membros de um objeto via construtor da classe.

- No VB.NET o construtor padrão recebe o nome de **Sub New()/ End Sub.**
- Um construtor é um procedimento que controla a inicialização de um objeto.
- Ao instanciar um objeto de uma classe o construtor da classe é executado automaticamente.
(*Você geralmente usa o construtor para iniciar membros de um objeto..)*

```
Namespace Exemplo3

  Class ClasseSimples3
    Public Var As Long                'definição do tipo da variável
    Public Sub New()
      Var = 10                        'valor de inicialização no construtor
da classe - valor fixo
    End Sub
  End Class

  Class Prog4
   Public Shared Sub Main()
     Dim obj As New ClasseSimples3()    'instanciando a classe
     System.Console.Write(obj.Var)
   End Sub
 End Class

End Namespace
```

lição 4 - Construtores com parâmetros - ao instanciar o objeto o construtor da classe é iniciado com parâmetro passado.

Você pode criar construtores com parâmetros , e , indo mais além , você pode criar vários construtores , cada um recebendo um número de parâmetros diferentes.

```
Namespace Exemplo4

  Class ClasseSimples4
    Public Var As Long
    Public Sub New(ByVal ValorInicial As Long)
      Var = ValorInicial                    'valor de inicialização no
construtor recebido como parâmetro
    End Sub
 End Class

 Class Prog5
   Public Shared Sub Main()
     Dim obj As New ClasseSimples4(1000)     'criando a instância da
classe com valor de inicialização igual a 1000
     System.Console.Write(obj.Var)
   End Sub
 End Class

End Namespace
```

lição 5 - Criando métodos que retornam valores. (Funções)

- Você pode criar um método na classe que realiza um cálculo ou operação e retorna um valor.
- Você instancia o objeto e passa os parâmetros para o método para obter o resultado

```
Namespace Exemplo5
   Class ClasseSimples5
      Public Shared Function Multiplica(ByVal Num1 As Long, ByVal Num2 As
Long) As Long
         Return Num1 * Num2
      End Function
   End Class
   Class Prog6
      Public Shared Sub Main()
         System.Console.Write(ClasseSimples5.Multiplica(10, 10))      'instancia o
objeto e chama o método Multiplica()
      End Sub
   End Class
End Namespace
```

lição 6 - Herança no VB .NET.

- A palavra chave **Inherits** permite criar a herança no VB.NET.
- Uma classe filha que herda de uma classe Pai tem acesso a todos os métodos e propriedades dessa Classe.
- Se C herda de B que herda de A , C tem acesso a todos os métodos e propriedades de B e A.
- Todas as classes declaradas podem ser herdadas. Para impedir que uma classe seja hereditária (ou herdada) use em sua declaração o modificador **NotInheritable.**

```
Namespace Exemplo6
   Class A   'classe Pai ou classe base.
      Public Sub A()
         System.Console.Write("A")
      End Sub
   End Class
   Class B         ' primeira classe filha ou subclasse da classe Pai
      Inherits A     ' esta linha indica que esta classe herda da classe A
      Public Sub B()
         System.Console.Write("B")
      End Sub
   End Class
   Class C          'segunda classe filha
      Inherits B      ' esta linha indica que esta classe herda da classe B
      Public Sub C()
         System.Console.Write("C")
      End Sub
   End Class
   Class Prog7
```

```
    Public Shared Sub Main()
        Dim objC As New C()      'instancia um objeto da classe C (sempre use
parenteses)
        objC.A()                 'como a classe C herda de B e B herda de A , então o
objeto objC tem acesso ao método da classe A
        objC.B()                 'aqui o método B() é herdado diretamente da classe  B
        objC.C()
    End Sub
  End Class
End Namespace
```

lição 7 - Herança no VB.NET.

- O modificador **MustInherit** faz com que uma classe seja destinada apenas a servir como base para criação de outras classes. Elas não podem ser instanciadas. (*Conhecida como classe **abstrata***)
- A única forma de usar uma classe com o modificador **MustInherit** é criar classes filhas que herdem suas propriedades e métodos.

```
Namespace Exemplo7
    MustInherit Class A       'Classe  Pai
      Public Sub A()
        System.Console.Write("A")
      End Sub
    End Class
    Class B         ' primeira classe filha
      Inherits A    ' esta linha indica que esta classe herda da classe A
      Public Sub B()
        System.Console.Write("B")
      End Sub
    End Class
    Class Prog8
        Public Shared Sub Main()
            'Dim objA as New A()   'Errado - pois você não pode criar uma
instância de uma classe  mustinherit class
          Dim objB As New B()     'Cria uma instância da classe B (sempre use
parênteses com a palavra-chave New)
          objB.A()
          objB.B()
        End Sub
    End Class
End Namespace
```

lição 8 - Herança no VB.NET.

- Como as classes que você cria podem servir como base para criação de outras classes , você deve sempre pensar no nível de acesso , ou na visibilidade que os membros da sua classe terá.
- Quando uma nova classe herda os métodos e propriedades de outra classe , pode ser necessário sobrepor alguns de seus membros. Você pode querer implementar um método

herdado de forma diferente.
- Para que seja possível sobrepor um método ou propriedade de uma classe ele precisa ser declarado com o modificador **Overridable.**
- Você pode desejar que certos métodos sejam **sobrepostos** e outros não ; podem ainda obrigar que certos métodos sejam sempre sobrepostos. Os modificadores usados neste caso são :

- **Overridable** : Permite que uma propriedade ou método seja sobreposto em uma classe herdada da classe atual. (*Métodos públicos não podem ser sobrepostos, ou seja , são NotOverridable.)*
- **Overrides** - Permite a sobreposição de um método ou propriedade definida na classe Pai.
- **NotOverridable** - Não permite que uma propriedade ou classe seja sobreposta em uma classe herdada da classe atual.

Nota: Para acessar membros da classe Pai ou classe Base usamos a palavra-chave **MyBase** . Com você pode referenciar a classe base imediata , seus membros públicos(Public) herdados mas não pode acessar métodos privados(Private).

```
Namespace Exemplo8
    Class A ' classe filha
      Public Overridable Sub A()
        System.Console.Write("A.A")
      End Sub
    End Class
   Class B         ' classe filha
      Inherits A     ' esta linha indica que esta classe herda da classe A
      Public Overrides Sub A()          'sobrepõe o método A() da classe
Pai.
         System.Console.Write("B.A")
      End Sub
  End Class
  Class Prog9
      Public Shared Sub Main()
          Dim objB As New B()
          objB.A()                    ' A saida será :  "B.A"  e NÃO  "A.A"
pois o método A() foi sobreposto
     End Sub
  End Class
End Namespace
```

lição 9 - Eventos

- Eventos são notificações que sua classe envia e recebe para o código que a criou.
- Os eventos são criados com a declaração **Events** seguido do nome do evento e dos argumentos que ele vai receber
- Eventos devem ser sempre declarados como públicos
- Eventos não podem retornar um valor
- Quando definimos um evento definido em uma classe , todos os objetos da classe podem causar o evento.
- Para disparar o evento usamos a declaração - **RaiseEvent**

- Você deve associar o evento a uma rotina que gerencia os eventos usando a declaração - **WithEvents ou AddHandler.**

```
Namespace Exemplo9
   Class A
     Public Event AoIniciar()      ' declara o evento como public
     Public Sub New()
        RaiseEvent AoIniciar()    ' executa o evento no construtor da classe A
     End Sub
     Public Overridable Sub A()
        System.Console.Write("A.A")
     End Sub
  End Class
  Class Prog9
    Private WithEvents objA As New A()       ' A palavra-chave 'WithEvents'  declara que a
variável objA do tipo A() terá a capacidade
                                          ' de receber e manipular eventos
    Public Shared Sub Main()
      Dim objB As New A()
      objB.A()                             ' a saida será  "objA - evento disparado"   e
NÃO   "A.A"
    End Sub
    'A palavra-chave "Handles"  informa que esta rotina é o gerenciador de eventos para o
evento "objA.AoIniciar - evento disparado"
    Private Sub objA_AoIniciar() Handles objA.AoIniciar
        System.Console.WriteLine("objA.AoIniciar - evento disparado")
    End Sub
 End Class
End Namespace
```

lição 10 - Interfaces

- Uma **Interface** é um tipo especial de classe que não contém código , mas é usada como uma alternativa para descrever a aparência de um objeto.
- O objetivo da interface é permitir separar uma definição de classe de sua implementação.
- Uma interface define propriedades , métodos e eventos relacionados , mas não contém nenhum código , apenas a estrutura dos membros necessários. *(Neste aspecto uma interface é semelhante a uma classe abstrata)*
- Uma definição de interface começa com a palavra -chave **Interface**
- Para utilizar uma interface uma classe deve especificar que implementa a interface usando a palavra-chave **Implements** , e , deve definir cada método especificado na interface.
- Quando você cria e implementa uma interface você não a altera. Se precisar criar novos métodos definidos na interface terá que criar outra interface.

```
Module Exemplo10
   'A interface não tem código apenas o protótipo para outra classes
   Public Interface Forma ' declara o evento como public
        Sub Area(ByVal x As Integer, ByVal y As Integer)
        Sub Volume(ByVal x As Integer, ByVal y As Integer, ByVal z As Integer)
  End Interface
  ' Classe que implementa a interface deverá definir a interface Forma
  Public Class Prog10
        Implements Forma
        Public Sub Area(ByVal x As Integer, ByVal y As Integer) Implements
Forma.Area
            Dim area As Integer
            area = x * y
        End Sub
        Public Sub Volume(ByVal x As Integer, ByVal y As Integer, ByVal z As
Integer) Implements Forma.Volume
            Dim volume As Integer
            volume = x * y * z
       End Sub
  End Class
End Module
```

Nestas 10 lições procurei abordar os conceitos básicos relacionados a orientação a objetos no VB.NET. É claro que não esgotei o assunto , isto é apenas a ponta do Iceberg. Espero que a forma objetiva e resumida da apresentação ajude aos que estão começando a compreender e assimilar os conceitos.

Bom estudo !

José Carlos Macoratti - www.macoratti.net