

Finalmente uma linguagem orientada a objetos

Não sei se você sabe , mas o Visual Basic é acusado com frequência de ser uma linguagem livremente estruturada ; dependendo do programador o código pode degenerar para um estilo de programação direcionado a eventos que espalha fragmentos de código para todos os lados causando muitos problemas.

Pela sua facilidade de aprendizado e utilização o VB atraiu um grande número de pessoas que nunca pensariam em criar programas para a plataforma Windows usando linguagens como C ou C++ ; se por um lado isto popularizou a linguagem , por outro , fez com que uma grande parte do código produzido fosse de baixa qualidade e isto atraiu uma má fama á linguagem.

Nestes últimos tempos a programação orientada a objetos(*OOP - Object Oriented Programming Language*) tem chamado muito a atenção e ganhado muito espaço entre os desenvolvedores. A utilização das técnicas de orientação a objeto no desenvolvimento de aplicativos facilita a depuração, o aperfeiçoamento , a escalabilidade e a reutilização do código.

Novamente o VB se viu envolvido em uma celeuma: Ser ou não ser uma linguagem orientada a objetos ? ([leia o artigo : Criando Classes no VB](#))

Rigorosamente falando , até a versão 6.0 , o VB não podia ser considerada uma linguagem que reunisse as características típicas de uma verdadeira linguagem orientada a objetos , como Java por exemplo. O VB 6.0 não dava suporte a herança (embora suportasse a implementação de interfaces). Com a chegada do VB.NET , você pode dizer de boca cheia : [O Visual Basic é uma linguagem orientada a Objetos](#). Isto trouxe novas perspectivas e também novas frustrações , a principal delas é a mudança de mentalidade ao se programar em VB.NET com a orientação a objetos.

Geralmente a grande maioria que usa a linguagem Visual Basic conhece a programação tradicional estruturada ; neste modelo o problema geralmente é dividido em duas partes : os dados e os processos. Você tem um problema e imediatamente pensa em criar funções que processem determinada tarefa para resolver o problema ou uma parte dele. Ao final , após muito trabalho, você acaba com uma coleção de funções que juntas compõem a solução do seu problema. A utilização das técnicas das orientação traz um resultado muito melhor como veremos.

OOP - O princípio

O precursor dos objetos foram as estruturas ([Tipos Definidos pelo Usuário - UDT no VB 6.0](#)) , que são uma forma de agrupar dados reunidos. Vamos a um exemplo prático: suponha que você queira reunir informações sobre uma pessoa (nome, telefone) , a primeira coisa que lhe vem a mente talvez seja criar variáveis para representar os dados que você quer reunir. Isto leva a um problema : as variáveis soltas no código além de torná-lo mais complicado (imagine se você tiver 100 dados) e trabalhoso (imagine fazer a mesma coisa para 100 pessoas diferentes) , fazem com que a probabilidade a erros aumente com a complexidade do seu problema. As estruturas tentaram resolver este problema. Vejamos um exemplo de código usando estruturas para o caso :

VB 6.0 - UDT ([User Defined Types](#))

Para definir um UDT usamos a seguinte sintaxe:

```
Type NomeTipo  
elementos as TipoDados
```

```
****  
End
```

- **Type** determina o início do bloco.
- **NomeTipo** o nome que você dá ao seu tipo
- **Elementos** e cada membro do tipo
- **TipoDados** é o tipo de dados definido

Vamos criar uma UDT chamada `Clientes`. Inicie o VB e crie um novo projeto adicionando a ele um módulo-padrão ([Type somente pode ser usado em módulos](#)) e digite:

```
Public Type Clientes  
Nome as string * 40  
telefone as string  
End Type
```

Uma vez declarado o `Tipo clientes` basta informar o nome da variável de tipo clientes e a seguir definir os elementos que o compõem. Supondo um formulário (form1) com duas `TextBox - text1, text2`, vejamos como atribuir/exibir os valores do tipo clientes.

```
'General Declarations
```

```
Dim gclientes As clientes
```

```
'Código do evento Click do formulário
```

```
(Atribui valores a TextBox ao clicar)
```

```
Private Sub Form_Click()
```

```
Text1 = "Jose Carlos Macoratti"
```

```
Text2 = "0XX-15-321-4565"
```

```
End Sub
```

```
'Código do evento Load do formulário
```

```
(Atribui valores aos elementos do Tipo)
```

```
Private Sub Form_Load()
```

```
gclientes.Nome = "Jose Carlos Macoratti"
```

```
gclientes.telefone = "0XX-15-321-4565"
```

```
End Sub
```

Ao rodar o projeto o evento `Form Load` carrega os elementos do Tipo e quando você clica no formulário os valores são exibidos nas caixas de texto. Embora as UDT sejam eficientes em muitos aspectos devemos lembrar que as variáveis que são instâncias de uma UDT (**gclientes é uma instância da UDT clientes**) estão sujeitas às mesmas regras de visibilidade e escopo das demais variáveis.

Vamos a seguir mostrar como o mesmo tratamento é feito no VB.NET

No VB.NET temos :

```
Public Structure Clientes
    Dim Nome As String
    Dim Telefone As String
End Structure
```

O código declara uma estrutura do tipo **Public** , por isso , pode ser colocado em qualquer lugar do arquivo , exceto dentro de funções ou rotinas. Se a estrutura for **Private** podemos colocar o código dentro de uma **classe ou módulo**. Feito isto é só usar a estrutura e criar um objeto do tipo **Clientes** em qualquer lugar do nosso código e atribuir as informações :

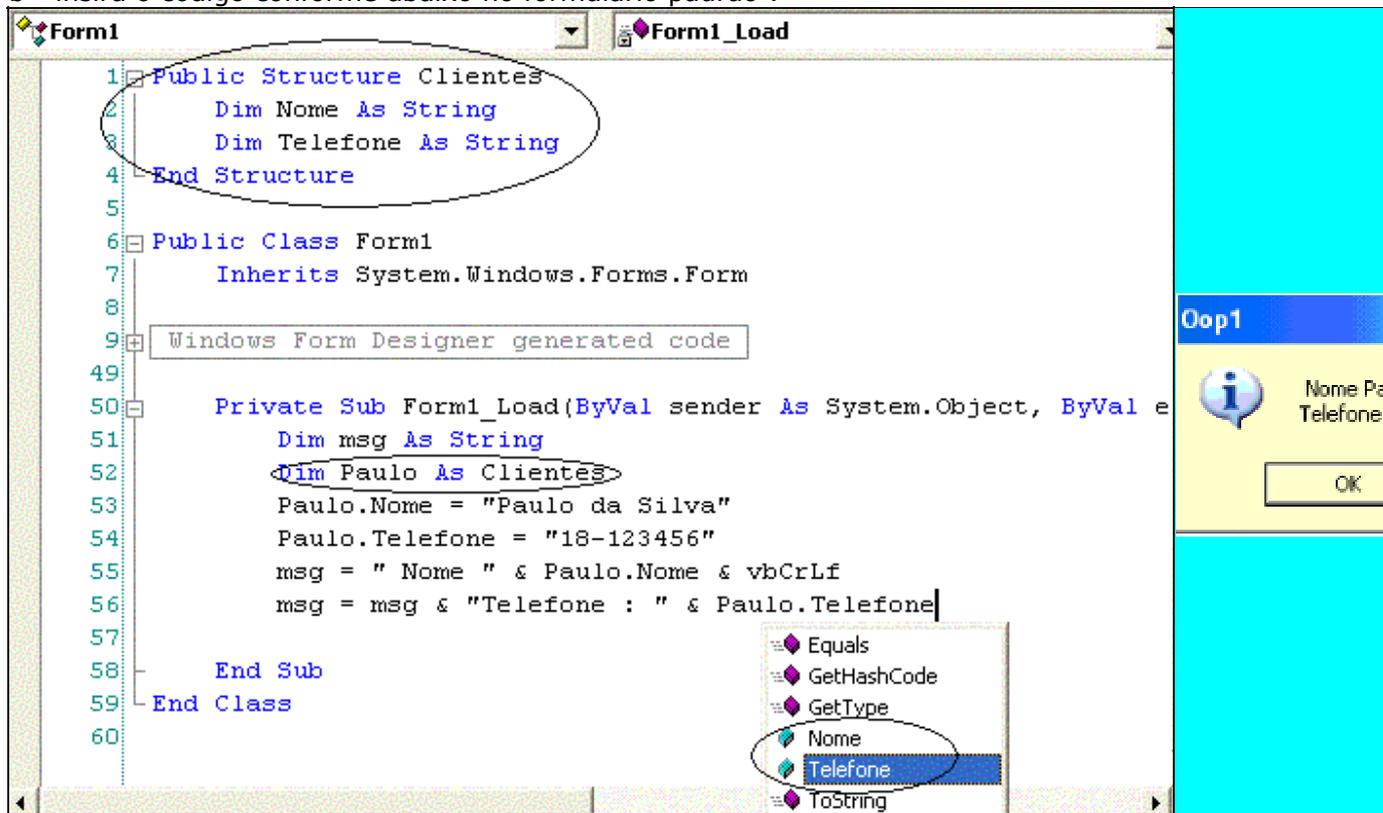
```
Dim Paulo As Clientes
Paulo.Nome = "Paulo da Silva"
Paulo.Telefone = "18-123456"
```

Para testar o exemplo faça o seguinte :

a - Inicie um novo projeto no Visual Studio.NET com as seguintes características (*sinta-se a vontade para alterar a seu gosto.*)

1. Project Types : **Visual Basic Projects**
2. Templates : **Windows Application**
3. Name : **Oop1**
4. Location : **c:\vbnet\oop**

b- insira o código conforme abaixo no formulário padrão :



```
1 Public Structure Clientes
2     Dim Nome As String
3     Dim Telefone As String
4 End Structure
5
6 Public Class Form1
7     Inherits System.Windows.Forms.Form
8
9     Windows Form Designer generated code
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50 Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
51     Dim msg As String
52     Dim Paulo As Clientes
53     Paulo.Nome = "Paulo da Silva"
54     Paulo.Telefone = "18-123456"
55     msg = " Nome " & Paulo.Nome & vbCrLf
56     msg = msg & "Telefone : " & Paulo.Telefone
57
58 End Sub
59 End Class
60
```

The screenshot also shows a message box titled "Oop1" with the following text:

```
Nome Paulo da Silva
Telefone 18-123456
```

An "OK" button is visible at the bottom of the message box.

Trabalhando com Classes no VB.NET

Vamos agora começar a trabalhar com classes no VB.NET ; eu perdi um tempo falando em estruturas pois existe muita semelhança entre estruturas e classes como você vai ver a seguir . As classes vão além pois é como se fossem estruturas com inteligência própria ; numa classe temos uma estrutura que pode incluir dados e código (métodos e funções). Já a partir da versão 4.0 o VB permitia a criação de classes com o VB.NET houve uma série de melhorias .Abaixo uma lista dos recursos para a OOP que foram aperfeiçoados no VB.NET :

1. No VB.NET você pode agrupar as suas classes de qualquer maneira desde que as coloque dentro de declarações. Uma classe começa com a palavra **Class** e termina com **End Class**. (*Você pode usar `Public Class` / `Private Class` , etc...*)
2. O VB.NET reconhece automaticamente as operações de objeto , com isto a declaração **Set** não é mais suportada.
3. No VB.NET para verificar se dois objetos são iguais usamos a palavra-chave **Is** : **If Obj_1 Is Obj_2 then** . (O sinal de igual não é suportado)
4. O VB.NET suporta a criação de **construtores** que iniciam variáveis e informações em uma classe.
5. No VB.NET você pode usar a palavra-chave **New** sem os efeitos colaterais que o VB 6 apresentava. Ao usar a sintaxe :

Dim variavel As New ClassNome o VB.NET cria e copia imediatamente um objeto.

Agora vamos criar a classe clientes no VB.NET com três campos de dados(*codigo, nome e telefone*) e dois métodos(*Exibir()* e *Definir()*). A seguir vamos criar uma instância da classe cliente que usará os métodos definidos para a classe:

- 1- O código para a **Classe Clientes** e seus dois métodos :

```

1 Class Clientes
2
3     'declaração dos membros de dados
4     Dim codigo As Integer
5     Dim nome As String
6     Dim telefone As String
7
8     'construtor da classe : define os dados de cada cliente
9     Sub Clientes(ByVal cod As Integer, ByVal nom As String, ByVal fone As String)
10        Me.codigo = cod
11        Me.nome = nom
12        Me.telefone = fone
13    End Sub
14
15    'exibe as propriedades para cada cliente
16    Sub exibir()
17        Dim msg As String
18        msg = " Cliente " & vbCrLf
19        msg = msg & "Código   " & CStr(Me.codigo) & vbCrLf
20        msg = msg & "Nome      " & Me.nome & vbCrLf
21        msg = msg & "Telefone " & Me.telefone
22        MsgBox(msg, MsgBoxStyle.Information, "Clientes")
23    End Sub
24 End Class

```

2- O código do formulário padrão - form1.vb :

```

27 Public Class Form1
28     Inherits System.Windows.Forms.Form
29
30     Windows Form Designer generated code
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51 Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
52     Dim cliente As New Clientes()
53     'define um cliente e exibe suas propriedades
54     cliente.Clientes(123, "Paula Lima", "19-9870-4512")
55     cliente.exibir()
56
57     'define outro cliente e exibe suas propriedades
58     cliente.Clientes(456, "Helena Paula Souza", "61-447-4512")
59     cliente.exibir()
60
61 End Sub
62 End Class

```

Executando o projeto iremos ter a exibição das propriedades para os dois objetos [Cliente](#) definidos :



Muito bem , a classe esta criada , já definimos seus métodos e conseguimos acessar suas propriedades . *Vamos tentar alterar alguns dados declarados na classe ?*

Vamos acrescentar três linhas de código de forma a tentar atribuir valores para as propriedades : [codigo](#) , [nome](#) e [telefone](#).

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
Dim cliente As New Clientes()
'define um cliente e exibe suas propriedades
cliente.Clientes(123, "Paula Lima", "19-9870-4512")
cliente.exibir()
'define outro cliente e exibe suas propriedades
cliente.Clientes(456, "Helena Paula Souza", "61-447-4512")
cliente.exibir()

-----
'codigo para atribuir valores diretamente as propriedades
-----
cliente.codigo = 100
cliente.nome = "Testolino"
cliente.telefone = "14-258-9874"
End Sub
```

Ao tentar executar iremos obter uma mensagem de erro , pois não podemos ter acesso direto aos dados declarados na classe , ou seja , as variáveis membros : [codigo](#) , [nome](#) e [telefone](#) . O motivo é que as variáveis declaradas na classe somente são visíveis dentro da classe , somente os métodos da classe conseguem enxergar as variáveis , dizemos que elas estão [encapsuladas](#) .

O [encapsulamento](#) ou [abstração](#) é um dos requisitos que uma linguagem orientada a objetos deve possuir ; com o [encapsulamento](#) os dados estão protegidos de alterações acidentais ou mal intencionadas por parte do usuário e isto é muito importante.

O usuário somente vai conseguir acessar os dados , quer incluindo ou alterando , através do [método Clientes](#) que definimos na classe. Você pode estar pensando : "*Qual a diferença ??*" , - A diferença meu amigo é que você pode colocar dentro da classe um código para validar ou não um valor atribuído a uma propriedade e com isto o usuário somente vai conseguir atribuir os dados que a classe deixar. Abaixo um exemplo de como poderíamos não permitir que o código atribuído fosse menor que 100 :

```

Sub Clientes(ByVal cod As Integer, ByVal nom As String, ByVal fone As String)
If cod > 100 Then
    Me.codigo = cod
    Me.nome = nom
    Me.telefone = fone
Else
    MsgBox("Código inválido !", MsgBoxStyle.Critical)
End If
End Sub

```

Percebeu ??? Não é a melhor forma de fazer isto , como você verá abaixo , mas já dá para perceber que podemos controlar o acesso aos dados da classe.

Outra coisa , se você precisar obter os valores dos dados da classe vai ter que criar outro método para fazer isto. Vamos mostrar como podemos alterar ou obter os dados a partir de uma instância da classe , fazemos isto incluindo propriedades a nossa classe. As propriedades são atributos que guardamos na classe com rotinas próprias para atribuir ou devolver valores . Vejamos como fica a nossa classe clientes reescrita usando propriedades :

Class Clientes

'declaração dos membros de dados

```

Dim icodigo As Integer
Dim snome As String
Dim stelefone As String

```

```

Property codigo() As Integer

```

```

    Get

```

```

        codigo = icodigo

```

```

    End Get

```

```

    Set(ByVal Value As Integer)

```

```

        If Value < 100 Then

```

```

            MsgBox("Código inválido !", MsgBoxStyle.Critical)

```

```

        Else

```

```

            icodigo = Value

```

```

        End If

```

```

    End Set

```

```

End Property

```

```

Property nome() As String

```

```

    Get

```

```

        nome = snome

```

```

    End Get

```

```

    Set(ByVal Value As String)

```

```

        snome = Value

```

```

    End Set

```

```

End Property

```

```

Property telefone() As String

```

```

    Get

```

```

        telefone = stelefone
    End Get
    Set(ByVal Value As String)
        stelefone = Value
    End Set
End Property

' exibe as propriedades para cada cliente
Sub exibir()
    Dim msg As String
    msg = "Cliente " & vbCrLf
    msg = msg & "Código " & CStr(Me.codigo) & vbCrLf
    msg = msg & "Nome " & Me.nome & vbCrLf
    msg = msg & "Telefone " & Me.telefone
    MsgBox(msg, MsgBoxStyle.Information, "Clientes")
End Sub

End Class

```

Usamos a declaração *Property Get/End Get e Property Set/End Set End Property* para definir propriedades , vejamos os detalhes :

Property - declara o nome de um propriedade usada para armazenar ou devolver o valor da propriedade

Get - Inicia uma propriedade usada para retornar o valor da propriedade . **End Get** - Termina a Propriedade Get.

Set - Inicia uma propriedade usada para definir/atribuir um valor para a propriedade . O novo valor da propriedade é passado para a mesma através do parâmetro Value. **End Set** - termina a propriedade Set.

End Property - termina a definição de propriedade.

Agora podemos atribuir valores para os dados da classe instanciando o objeto da classe. Veja o código como ficou :

```

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

Dim cliente As New Clientes()

'define um cliente e exibe suas propriedades
cliente.codigo = 110
cliente.nome = "Joao da Silva"
cliente.telefone = "014-258-7898"
cliente.exibir()

'define outro cliente e exibe suas propriedades
cliente.codigo = 489
cliente.nome = "Maria Sebastiana da Silva"

```

```
cliente.telefone = "011-457-1111"  
cliente.exibir()
```

```
End Sub
```

Então para retornar o valor de uma propriedade podemos usar a instância da classe e exibir os dados via propriedades definidas. Assim :

1 - Quando fazemos : `TextBox1.text = cliente.nome` o código da declaração `Get` é executada e nome é retornado:

```
Property nome() As String
```

```
    Get
```

```
        nome = snome
```

```
    End Get
```

```
    Set(ByVal Value As String)
```

```
        snome = Value
```

```
    End Set
```

```
End Property
```

2- Quando fazemos : `cliente.nome = "Testolino "` o código da declaração `Set` é executado e um valor é atribuído ao dado.

```
Property nome() As String
```

```
    Get
```

```
        nome = snome
```

```
    End Get
```

```
    Set(ByVal Value As String)
```

```
        snome = Value
```

```
    End Set
```

```
End Property
```