Visual Basic .NET

Programação Orientada a Objetos

Professor: Danilo Giacobo

Página pessoal: www.danilogiacobo.eti.br
E-mail: danilogiacobo@gmail.com

Objetivos da aula

- ✓ Classes e Objetos
- ✓ Campos, Propriedades, Métodos e Eventos
- ✓ Membros da Classe x Membros do Objeto
- ✓ Atributos e Métodos
- ✓ Abstração
- ✓ Encapsulamento
- ✓ Polimorfismo
- ✓ Herança
- ✓ *Overloading*, *Overriding* e *Shadowing*
- ✓ Construtores e Destrutores
- ✓ Um exemplo de POO na prática
- ✓ Criando Classes
- ✓ Criando Objetos
- ✓ Criando Estruturas
- ✓ Criando Elementos Estáticos
- ✓ Criando Eventos
- ✓ Criando uma biblioteca de classes
- ✓ Usando o método Finalize

Definições


As células enviam mensagens para as outras células afim de alcançar um objetivo...
As coisas importantes para o meu sistema seriam os objetos..
Como seria um sistema de software que funcionasse como um ser vivo?
Os objetos realizarão tarefas através da requisição de serviços a outros objetos...
A classe é um repositório para comportamento associado ao objeto.
Cada objeto pertencerá a uma determinada classe. Uma classe pode agrupar objetos similares.
Classes serão organizadas em hierarquias.

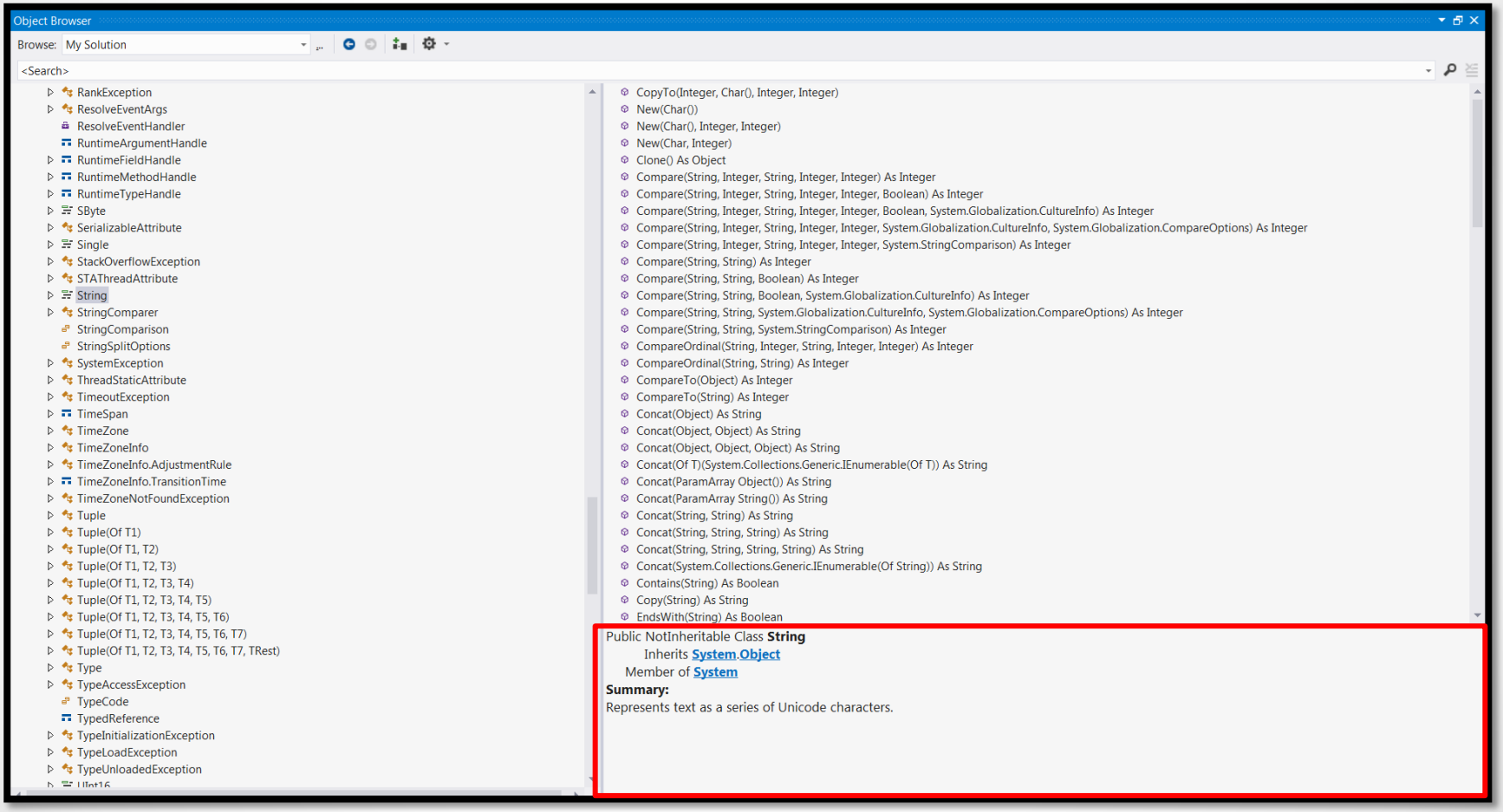
Introdução

- Tudo que você usando o **VB .NET** envolve a utilização de objetos de alguma forma - até mesmo simples variáveis (do tipo String por exemplo) são baseadas em alguma classe.
- O **VB .NET** fornece milhares de classes embutidas, prontas para serem usadas em suas aplicações.
- Sabemos que os formulários Windows são baseados na classe **System.Windows.Forms.Form**.
- A criação de elementos visuais envolve a criação de um novo objeto de uma classe, a configuração de suas propriedades e utilização de seus métodos.

Relembrando um exemplo:

```
1  Public Class Form1
2      Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
3          Dim meuTextBox As New TextBox
4          meuTextBox.Size = New Size(150, 20)
5          meuTextBox.Location = New Point(80, 20)
6          meuTextBox.Text = "Olá Mundo!"
7          Controls.Add(meuTextBox)
8      End Sub
9  End Class
```

Object Browser



Classes e Objetos

- Classes são estruturas que definem e representam um conjunto de objetos com características em comum.
- Um objeto é um exemplo desta classe, isto é, uma **instância** da mesma.
- Exemplo:

  Classe String

  Objetos da mesma: meuTexto, minhaFrase, etc.
- Para criar uma classe em VB .NET é necessário usar a declaração **Class**. Exemplo:

  Public Class *minhaClasse*

  ...

  End Class
- Para criar um objeto (instância) desta classe você deve usar a palavra **New**:

  Dim *objExemplo1* **As New** minhaClasse()
- Você pode fazer desta outra forma também:

  Dim *objExemplo2* **As** minhaClasse **= New** minhaClasse()

Campos, Propriedades, Métodos e Eventos

- São chamados de *membros* de uma **classe**.
- Dentro de uma classe, membros são declarados como:
  - **Public;**
  - **Private;**
  - **Protected;**
  - **Friend; ou**
  - **Protected Friend.**

Nome	Descrição
Public	Permite acesso público para variáveis. Sem restrições de acessibilidade.
Private	Permite acesso privado para variáveis. São apenas acessíveis dentro de sua própria classe, incluindo qualquer procedimento aninhado.
Protected	Permite acesso protegido para variáveis. São apenas acessíveis dentro de sua própria classe ou de uma classe derivada da mesma. Você só pode usar este recurso em níveis de classe (para membros). Dentro de procedimentos não é permitido.
Friend	Permite acesso amigo para variáveis. São acessíveis de dentro do programa que contém sua declaração assim como de qualquer lugar do mesmo *assembly*.
Protected Friend	Permite acesso tanto protegido quanto amigo, o que significa que elas podem ser usadas no mesmo assembly assim como em códigos de classes derivadas.

Campos, Propriedades, Métodos e Eventos

- Os **campos** de uma classe (também conhecidos como *data members*) são bem parecidos com variáveis. Por exemplo, eu posso declarar um campo chamado valor na classe que vimos anteriormente chamada *minhaClasse*.

```
Public Class minhaClasse
    Public valor As Integer
End Class
```

- Eu posso agora referenciar este campo em um objeto com a sintaxe padrão do VB .Net de acesso a objetos:

```
Dim objExemplo1 As New minhaClasse()
objExemplo1. valor  = 5
```

- Você pode fazer com que os campos de uma classe armazenem constantes:

```
Public Const campo1 As Integer = 0
```

Campos, Propriedades, Métodos e Eventos

- Usar campos que fornecem acesso direto aos dados armazenados não é um prática comum e válida em POO. Você geralmente quer verificar se o dado que está sendo armazenado em um objeto é válido e legal.
- Um modo fácil de resguardar o acesso dos dados de um objeto é usar ***propriedades***. Elas possuem o mesmo modo de escrita e leitura m as são mantidos por procedimentos **Gets** e **Sets**.

Exemplo:

```
Private strNome As String

Public Property Nome() As String
    Get
        Return strNome
    End Get
    Set(ByVal xstrNome As String)
        strNome = xstrNome
    End Set
End Property
```

Campos, Propriedades, Métodos e Eventos

- ***Métodos*** representam os procedimentos embutidos de um objeto.
- Você define os métodos de uma classe tanto por meio de procedimentos do tipo **Sub** ou **Function**.

Exemplo:

```
1  Public Class Animal
2      Public Sub Comendo()
3          MsgBox("Comendo...")
4      End Sub
5
6      Public Sub Dormindo()
7          MsgBox("Dormindo...")
8      End Sub
9  End Class
```

```
Dim objAnimal As New Animal()
objAnimal.Comendo()
objAnimal.Dormindo()
```

Campos, Propriedades, Métodos e Eventos

- ***Eventos*** permitem que objetos executem ações quando uma específica ocorrência ocorre.
- Por exemplo, quando você clicar em um botão, um evento **Click** ocorre e você pode manipular o mesmo com um tratador de eventos conforme a gente viu em vários exemplos práticos.
- Como exemplo eu vou criar um evento customizado que é acionado quando eu clico em um botão três vezes:

```
Private Sub botao_TresCliques(ByVal strMensagem As String) Handles botao.TresCliques
    TextBox1.Text = strMensagem
End Sub
```

Membros da Classe x Membros do Objeto

- Membros de um classe que são invocados a partir do nome da mesma são chamados de *shared*, *static* ou *membros da classe*.
- Membros que são aplicados a objetos de uma classe são chamados de *instâncias* ou *membros do objeto*.

Exemplo:

- Se **TextBox1** é um objeto, então a sua propriedade **Text** é uma *instância* ou *membro do objeto*, porque você usa-o com o nome do objeto:

TextBox1.Text = "Olá Mundo!"

- Em contrapartida, você pode tornar membros do tipo compartilhado ou membros da classe (que você pode usar com o nome da classe, se você usar a palavra **Shared**.

- Usando este forma você pode acessar qualquer membro da classe usando apenas o nome da mesma sem precisar instanciar um objeto.

Membros da Classe x Membros do Objeto

- O exemplo abaixo mostra uma classe com um membro (um método) *compartilhado*. Isto significa que eu posso usar o nome da classe (**Matematica**) para acessar este método (**Somar**) de forma direta e que este é ainda *compartilhado* com todas as instâncias da classe.

```
1  Public Class Matematica
2      Shared Function Somar(ByVal intX As Integer, ByVal intY As Integer) As Integer
3          Return intX + intY
4      End Function
5  End Class
```

```
1  Public Class Form1
2      Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
3          TextBox1.Text = Matematica.Somar(TextBox2.Text, TextBox3.Text)
4      End Sub
5  End Class
```

Na pasta:
ClassObjectMembers.sln

Atributos e Métodos

- **Atributos** são os dados de uma classe.
- Estão relacionados com as características que formam a estrutura da classe.
- É por meio dos atributos que os valores que personalizam uma classe são armazenados e podem ser modificados/acessados.

Exemplos:

Classe	Atributos
Pessoa	Nome, Peso, Altura
Carro	Cor, Modelo, Ano

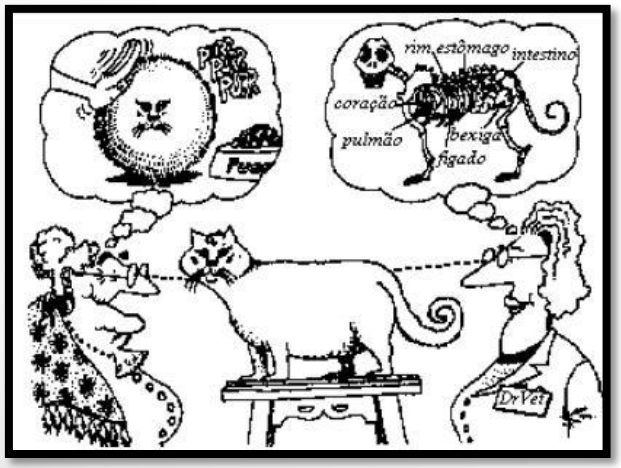
- **Métodos** são os ações /operações que a classe executa.
- Eles também são conhecidos por *procedimentos* ou *funções*.
- Eles podem receber parâmetros, retornar valores, etc.

Exemplos:

Classe	Métodos
Pessoa	Andar, Falar, Respirar
Carro	Acelerar, AbrirPorta, LigarRádio

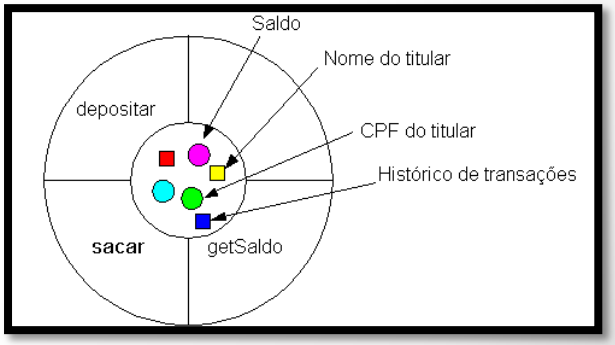
Abstração

- É a habilidade de um criar uma representação abstrata de um conceito do mundo real em código.
- Exemplo: um objeto de nome **objPessoa** é uma abstração de uma pessoa do mundo real.
- **Abstração** é a prática de nos concentrarmos somente nos detalhes mais importantes de um objeto. Possibilitando o descarte de aspectos menos importantes para a funcionalidade do mesmo.
- Em OO, se uma classe contiver apenas as informações necessárias, é muito provável que ela seja reutilizada com facilidade e uma das maiores vantagens da POO é a reutilização de código.



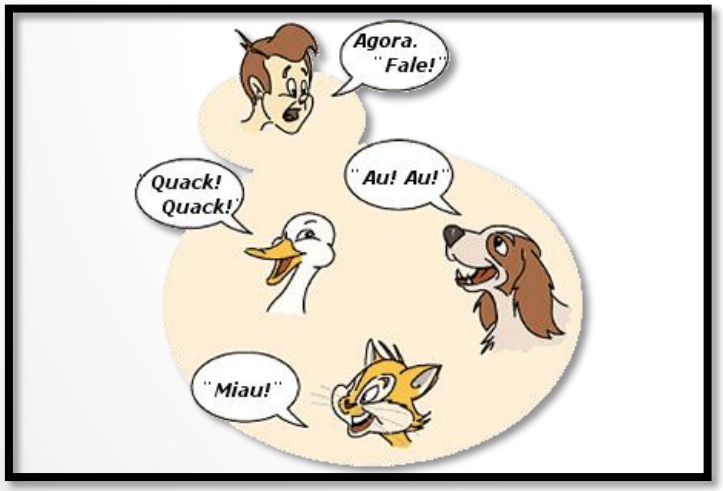
Encapsulamento

- **Encapsulamento** é o conceito de separar a implementação de um objeto se sua interface. Em outras palavras, quando você encapsula um objeto, você torna seu código e dados não acessíveis ao exterior exceto por meio de interfaces bem definidas. Este conceito é também chamado de *data hiding*.
- É a prática de esconder como um objeto executa as suas operações, quando ele for solicitado por um cliente. Isso traz muitos benefícios, pois o cliente não precisa ter a mínima ideia de como o objeto funciona internamente e, para ele, isso realmente não importa.
- **Um exemplo real:** ao pressionar o pedal acelerador de um carro (objeto), nós (clientes) não precisamos saber que ele irá aumentar a quantidade de combustível e ar enviado ao motor, que irá gerar uma maior combustão, movendo os pistões com mais força e rapidez, com isso aumentando a velocidade do carro. Para o cliente basta saber que, ao pressionar o pedal acelerador, o carro irá adquirir velocidade.



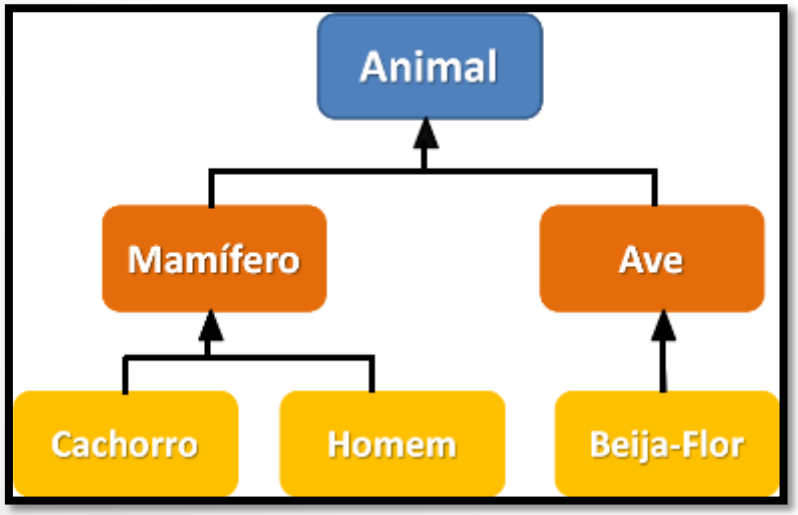
Polimorfismo

- **Polimorfismo** é o comportamento diferenciado de uma operação dependendo do objeto que a está executando.
- Criar procedimentos que possam operar em objetos de tipos diferentes.
- Por exemplo, se ambos os objetos **pessoa** e **empregado** possuem uma propriedade chamada **sobrenome** então um procedimento polimórfico pode usar a propriedade de ambos os objetos.
- O VB .NET trabalha com o conceito de polimorfismo de duas maneiras: *late binding* e *múltiplas interfaces*.



Herança

- **Herança** é um mecanismo único da OO que permite que características comuns a diversas casses sejam generalizadas em uma **Classe Base** ou **Super Classe**.
- A partir dessa **Classe Base**, outras classes podem ser derivadas.
- Cada **Classe Derivada** ou **Subclasse** apresenta as características da **Classe Base**, possibilitando não só o acréscimo de particularidades, mas também a alteração (quando possível) da estrutura que herdou.



Overloading, Overriding e Shadowing

- São conceitos realmente importantes na Orientação a Objetos em VB .NET.
- Essas técnicas permitem que você crie múltiplos membros com o mesmo nome.

❖ ***Overload*** *(Sobrecarga)*

- Membros proveem versões diferentes de uma propriedade ou método que possuem o mesmo nome, mas aceitam um número de parâmetros diferentes (ou parâmetros de tipos de dados diferente).

❖ ***Overridden*** *(Sobrescrita)*

- Propriedades e métodos são usadas para substituir um propriedade ou método herdado. Quando você sobrescreve um membro de um classe base, você substitui-o. Membros sobrescritos devem obrigatoriamente aceitar o mesmo tipo e número de argumentos do original.

❖ ***Shadowed***

- Membros são usados para criar uma versão local de um membro que tem um escopo maior. Você pode inclusive realizar o *sombreamento* de um tipo com qualquer outro tipo. Por exemplo: você pode declarar um propriedade que "sombreia" um método herdado com o mesmo nome.

Construtores e Destrutores

- Você pode criar objetos com palavra New conforme foi visto anteriormente:

  Dim *obj* **As New** Classe()
- Quando você cria um objeto você pode querer customizar o mesmo com dados. Exemplo: se eu crio um objeto da classe **Empregado**, eu posso querer armazenar o **nome**, **telefone**, **CPF** do mesmo neste objeto. Este tipo de atividade é realizada por um ***construtor***.
- Você envia dados para um construtor usando o nome da classe e colocando os parênteses quando da criação de um novo objeto. No exemplo mostrado acima não há dados sendo passados para o construtor por isso não há nada entre os "()".
- Cada classe possui um construtor padrão que não aceita argumentos.
- Se eu quiser passar algo para ser armazenado em um objeto quando este é criado eu coloco os valores dentro dos parênteses. Exemplo:

  Dim *obj* **As New** Classe(5)

Construtores e Destrutores

- Para criar um construtor é necessário adicionar um procedimento chamado **New** e listar os argumentos da mesma (se houverem).

Exemplo:

```
1  Public Class Classe
2      Private intValor As Integer
3
4      Public Sub New()
5
6      End Sub
7
8      Public Sub New(ByVal intNovoValor As Integer)
9          intValor = intNovoValor
10     End Sub
11
12     Public Function ObterDados() As Integer
13         Return intValor
14     End Function
15 End Class
```

```
1  Public Class Form1
2      Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
3          Dim obj1 As New Classe()
4          Dim obj2 As New Classe(5)
5          MsgBox(obj1.ObterDados)
6          MsgBox(obj2.ObterDados)
7      End Sub
8  End Class
```

Na pasta:
Construtor_Destrutor.sln

Construtores e Destrutores

- O ciclo de vida de um objeto termina quando o escopo do mesmo é deixado ou eles são configurados como **Nothing**.
- O VB .NET controla a liberação de recursos dos objetos da memória com os procedimentos chamados de ***destrutores***.
- O destrutor **Finalize** é usado normalmente para esta tarefa. Ele é chamado automaticamente quando um objeto é destruído (não chame este procedimento de forma direta).
- O próprio **framework do .NET** automaticamente executa o destrutor **Finalize** e destrói objetos quando o sistema determina que tais objetos não são mais necessários.
- O famoso processo que limpa da memória os recursos que não estão mais sendo usados é chamado de **Garbage Collector**.

Um exemplo de POO na prática

- Depois de tantos conceitos apresentados, vamos por em prática este conhecimento usando um programa que armazena objetos em um combo box. Cada objeto possui um nome e um índice. Quando o usuário seleciona um item, o código recupera os dados do objeto correspondente e mostra uma mensagem na tela.

Parte I - Criação da classe **clsItem**

```
1  Public Class clsItem
2      Private sngIndice As Single
3      Private strNome As String
4
5      Public Sub New(ByVal xsngIndice As Single, ByVal xstrNome As String)
6          sngIndice = xsngIndice
7          strNome = xstrNome
8      End Sub
9  End Class
```

Na pasta:
POO_NaPratica.sln

Um exemplo de POO na prática

- Dois métodos foram adicionados a esta classe:
  - **ToString:** retorna o nome do item (que foi sobrescrito da classe **Object**);
  - **ObterDados:** obtém o índice do item.

Parte 2 - Métodos da classe **clsItem**

```
1  Public Class clsItem
2      Private sngIndice As Single
3      Private strNome As String
4
5      Public Sub New(ByVal xsngIndice As Single, ByVal xstrNome As String)
6          sngIndice = xsngIndice
7          strNome = xstrNome
8      End Sub
9
10     Overrides Function ToString() As String
11         Return CStr(strNome)
12     End Function
13
14     Public Function ObterDados()
15         Return sngIndice
16     End Function
17 End Class
```

Um exemplo de POO na prática

- Para preencher o Combo Box do exemplo foi usado um simples laço de repetição que adiciona 20 itens na lista.

Parte 3 - Preenchimento dos itens do Combo Box

```
1  Public Class Form1
2      Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
3          Dim arrObjetos(20) As clsItem
4          ComboBox1.BeginUpdate()
5          Dim intIndice As Integer
6          For intIndice = 0 To 20
7              arrObjetos(intIndice) = New clsItem(CSng(intIndice), "Item " & intIndice)
8          Next
9          ComboBox1.Items.AddRange(arrObjetos)
10         ComboBox1.EndUpdate()
11     End Sub
12 End Class
```

Um exemplo de POO na prática

- Quando a pessoa muda a seleção de um item do Combo Box, os dados do mesmo são recuperados por meio da propriedade **SelectedItem**, e então o método **ObterDados** é usado para recuperar os dados do item.

Parte 4 - Selecionando um item do Combo Box

```
1  Public Class Form1
2      Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
3          Dim arrObjetos(20) As clsItem
4          ComboBox1.BeginUpdate()
5          Dim intIndice As Integer
6          For intIndice = 0 To 20
7              arrObjetos(intIndice) = New clsItem(CSng(intIndice), "Item " & intIndice)
8          Next
9          ComboBox1.Items.AddRange(arrObjetos)
10         ComboBox1.EndUpdate()
11     End Sub
12
13     Private Sub ComboBox1_SelectedIndexChanged(sender As Object, e As EventArgs) _
14         Handles ComboBox1.SelectedIndexChanged
15         MsgBox("O índice do item que você selecionou é: " & _
16                CType(ComboBox1.SelectedItem, clsItem).ObterDados())
17     End Sub
18 End Class
```

Criando Classes

- Para criar uma classe em VB .NET usamos a declaração **Class**. Sua sintaxe é:

[*<lista_de_atributos>*] [Public | Private | Protected | Friend | Protected Friend]
[Shadows] [MustInherit | NotInheritable] **Class** *nome_da_classe*
[Implements *nome_da_interface*]
[declarações]
End Class

- O **nome da classe** é obrigatório.

```
1  Public Class Filme
2      Private strTitulo As String
3
4      Public Sub New(ByVal xstrTitulo As String)
5          strTitulo = xstrTitulo
6      End Sub
7
8      Public Function ObterDados() As String
9          Return strTitulo
10     End Function
11 End Class
```

Criando Objetos

- Para criar um objeto em VB .NET usamos a declaração **Dim**. Sua sintaxe é:

[*<lista_de_atributos>*] [{ Public | Private | Protected | Friend | Protected Friend | Static }] [Shared] [Shadows] [ReadOnly] **Dim** [WithEvents] *nome_do_objeto* [(*tamanho_array*)] [As [New] *tipo_de_dados*] [= *valor_inicial*]

Exemplo:

```
1  Public Class Form1
2      Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
3          Dim objFilme1 As New Filme("O Poderoso Chefão - Parte 1")
4          Dim objFilme2 As Filme = New Filme("O Colecionador de Ossos")
5          MsgBox(objFilme1.ObterDados)
6          MsgBox(objFilme2.ObterDados)
7      End Sub
8  End Class
```

Criando Estruturas

- Estruturas permitem que você crie seu próprio tipo de dado complexo.
- Por exemplo, se você quiser quer armazenar em uma única variável o código e o nome de uma pessoa, você criar uma estrutura chamada **Pessoa** e definir ela da seguinte forma:

[*<lista_de_atributos>*] [{ Public | Private | Protected | Friend | Protected Friend }]
Structure *nome_da_estrutura*
    [*declaração de variáveis*]
    [*declaração de procedimentos*]
End Structure

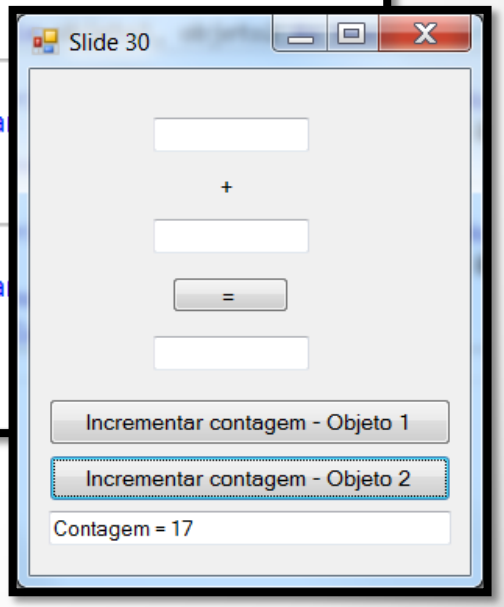
```
14      Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
15          Dim pessoa As Pessoa
16          pessoa.codigo = 1
17          pessoa.nome = "Danilo Giacobo"
18          MsgBox(pessoa.codigo & " - " & pessoa.nome)
19      End Sub
20  End Class
```

```
1  Public Structure Pessoa
2      Public nome As String
3      Public codigo As Integer
4  End Structure
```

Criando Classes com atributos Shared

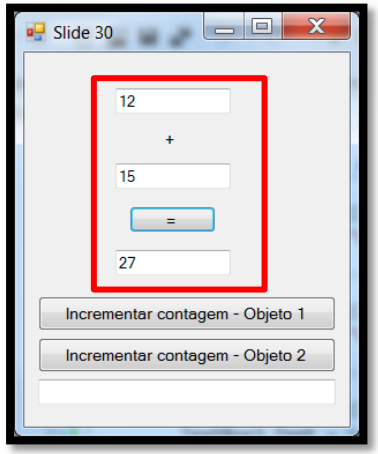
```vb
Public Class Matematica
    Public Const Pi As Double = 3.1415926535
    Shared valor As Integer = 0

    Public Function Incrementar() As Integer
        valor += 1
        Return valor
    End Function
End Class
```

```vb
Public Class Form1
    Dim objeto1, objeto2 As New Matematica

    Private Sub Button2_Click(sender As Object, e As EventArgs) Ha
        TextBox4.Text = "Contagem = " & objeto1.Incrementar
    End Sub

    Private Sub Button3_Click(sender As Object, e As EventArgs) Ha
        TextBox4.Text = "Contagem = " & objeto2.Incrementar
    End Sub
End Class
```



Criando Classes com métodos Shared

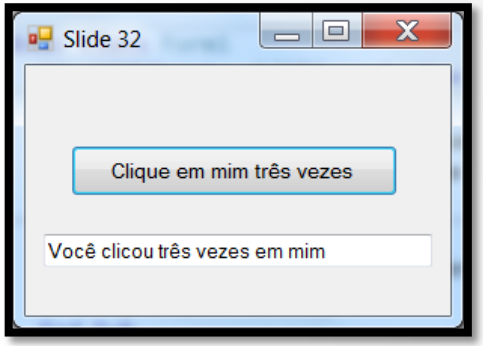
```
1  Public Class Matematica
2      Public Const Pi As Double = 3.1415926535
3      Shared valor As Integer = 0
4
5      Public Function Incrementar() As Integer
6          valor += 1
7          Return valor
8      End Function
9
10     Shared Function Somar(ByVal x As Integer, ByVal y As Integer)
11         Return x + y
12     End Function
13 End Class
```




```
1  Public Class Form1
2      Dim objeto1, objeto2 As New Matematica
3
4      Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
5          TextBox4.Text = "Contagem = " & objeto1.Incrementar
6      End Sub
7
8      Private Sub Button3_Click(sender As Object, e As EventArgs) Handles Button3.Click
9          TextBox4.Text = "Contagem = " & objeto2.Incrementar
10     End Sub
11
12     Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
13         TextBox3.Text = Matematica.Somar(TextBox1.Text, TextBox2.Text)
14     End Sub
15 End Class
```

Criando Eventos

```
1  Public Class Clique
2      Public Event TresCliques(ByVal strMensagem As String)
3
4      Public Sub Clicar()
5          Static ContagemCliques As Integer = 0
6          ContagemCliques += 1
7          If ContagemCliques >= 3 Then
8              ContagemCliques = 0
9              RaiseEvent TresCliques("Você clicou três vezes em mim")
10         End If
11     End Sub
12 End Class
```

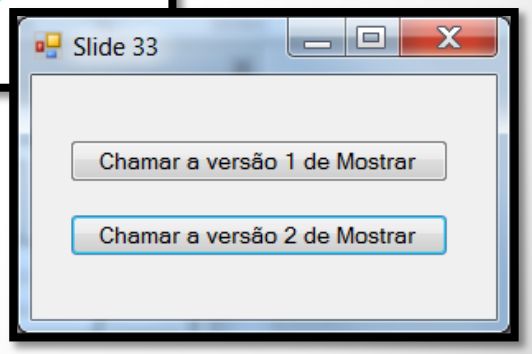



```
1  Public Class Form1
2      Dim WithEvents objClique As New Clique
3
4      Private Sub objClique_TresCliques(ByVal strMensagem As String) Handles objClique.TresCliques
5          TextBox1.Text = strMensagem
6      End Sub
7
8      Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
9          objClique.Clicar()
10     End Sub
11 End Class
```

Sobrecarga de métodos e propriedades

```
1  Public Class Notificador
2      Public Sub Mostrar(ByVal strMensagem As String)
3          MsgBox(strMensagem)
4      End Sub
5
6      Public Sub Mostrar(ByVal strMensagem As String, ByVal mbsIcone As MsgBoxStyle)
7          MsgBox(strMensagem, mbsIcone)
8      End Sub
9  End Class
```

```
1  Public Class Form1
2      Dim objNotificador As New Notificador
3
4      Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
5          objNotificador.Mostrar("Olá Mundo da OO!")
6      End Sub
7
8      Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
9          objNotificador.Mostrar("Olá Mundo da OO!", MsgBoxStyle.Exclamation)
10     End Sub
11 End Class
```



Criando uma biblioteca de classes

```
1  Public Class Class1
2      Public Sub Mostrar(ByVal strMensagem As String)
3          MsgBox(strMensagem)
4      End Sub
5  End Class
6
7  Public Class Class2
8      Public Sub Mostrar(ByVal strMensagem As String)
9          MsgBox(strMensagem, MsgBoxStyle.Exclamation)
10     End Sub
11 End Class
```

```
1  Public Class Form1
2      Dim cl1 As New Slide34.Class1
3      Dim cl2 As New Slide34.Class2
4
5      Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
6          cl1.Mostrar("Minha primeira biblioteca de classes!")
7          cl2.Mostrar("Minha primeira biblioteca de classes!")
8      End Sub
9  End Class
```

Usando o método Finalize

- Destrutores são executados quando um objeto é destruído.
- Você coloca o código de limpeza do objeto (fechar conexão, desconectar da internet, fechar arquivos, etc.) em um destrutor.
- Em VB .NET o método Finalize é usado para este propósito.
- Ele é chamado automaticamente quando o *runtime* do .NET decide que o objeto não é mais necessário.

Exemplo:

```
1  Public Class Form1
2      Dim objeto As New Class1
3  End Class
```

```
1  Public Class Class1
2      Protected Overrides Sub Finalize()
3          Beep()
4      End Sub
5  End Class
```

Referências Bibliográficas

- CAMARA, Fábio. **Orientação a Objeto com .NET**. 2ª Edição. Florianópolis: Visual Books, 2006. 114 páginas. ISBN: 85-7502-188-5.
- HOLZNER, Steven. **Visual Basic .NET: Black Book**. Arizona: Coriolis Group Books, 2002. xxxviii, 1144 p ISBN 1-57610-835-X.