Capítulo 7

Visual Basic.NET

- Programação
- Tipos de dados e variáveis
- Operadores
- Estrutura de decisão
- Estrutura de repetição
- Tratamento de erros e exceções
- Vetores e matrizes
- Classes
- Windows Form Application - componentes
- Eventos

O Visual Basic.NET é mais uma ferramenta que compõe o Visual Studio. Ele permite criar aplicativos Windows Cliente, Servidor, Internet, sem a necessidade de usar outra ferramenta. Ao trabalhar com a Plataforma .NET, os aplicativos são gerenciados e não mais interpretados ou nativos (VB 6.0), além de incorporar novos recursos. A Microsoft considera uma ferramenta do tipo RAD – Rapid Application Development, que possibilita o desenvolvimento rápido de aplicativos, como o próprio nome indica em inglês.

7.1. Programação

A programação em Visual Basic, ou simplesmente VB, exige os mesmos cuidados já apresentados anteriormente. É importante, porém, fazer uma recordação sucinta.

7.1.1. Console Application

Valem as mesmas aplicações em C#, nas quais teremos uma janela do tipo DOS como resposta.

7.1.2. Windows Form Application

Ao iniciarmos uma aplicação do tipo Windows Form Application, a Soluction Explorer deverá fornecer o Form1.vb (figura 266), o qual receberá a programação do nosso projeto.

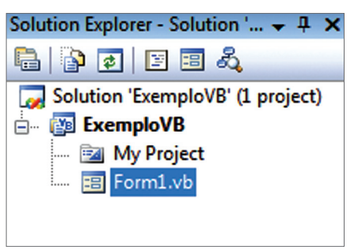


Figura 266
Solution Explorer – VB.

Na janela de desenvolvimento (figura 267), podemos verificar a presença de outro elemento já conhecido, o Form1.vb [Design], que representa o design do projeto, e a Start Page, a página inicial do Visual Studio.

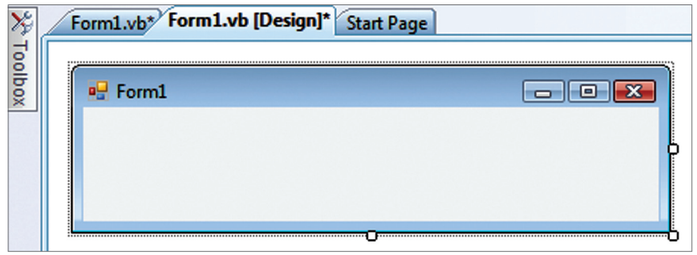


Figura 267
Abas da janela Code.

Na programação em VB (Form1.vb), envolvemos novamente os conceitos de programação orientada a objeto, usando a mesma metodologia já apresentada em C#, com suas classes, atributos, métodos e o controle de eventos, como na figura 268.

```
Public Class Form1

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal
        MsgBox("Meu primeiro Programa para Windows em VB")
    End Sub
End Class
```

Figura 268
Programação em VB.

7.2. Tipos de dados e variáveis

Os tipos de dados são tratados diretamente pelo .NET Framework. Portanto, utilizaremos, no VB, os tipos de dados semelhantes ao C#, mas com notações diferentes em alguns casos (tabela 14). Siga as mesmas recomendações especificadas anteriormente.

TIPO	IMPLEMENTAÇÃO
Byte	Inteiro de 8 bits sem sinal (0 a 255)
Sbyte	Inteiro de 8 bits com sinal (-127 a 128)
Ushort	Inteiro de 16 bits sem sinal (0 a 65 535)
Short	Inteiro de 16 bits com sinal (-32 768 a 32 767)
Uinteger	Inteiro de 32 bits sem sinal (0 a 4 294 967 295)
Integer	Inteiro de 32 bits com sinal (-2 147 483 648 a 2 147 483 647)
Ulong	Inteiro de 64 bits sem sinal (0 a 18 446 744 073 709 551 615)
Long	Inteiro de 64 bits com sinal (-9 223 372 036 854 775 808 a 9 223 372 036 854 775 807)

Tabela 14
Tipos de dados em VB.

Single	Ponto Flutuante Binário de 4 bytes
Double	Ponto flutuante binário IEEE de 8 bytes (±5.0 x10^{-324} a ±1.7 _ 10308), 15 dígitos decimais de precisão
Decimal	Ponto flutuante decimal de 128 bits. (1.0 _ 10-28 a 7.9 _ 1028), 28 dígitos decimais de precisão
Boolean	Pode ter os valores true e false. Não é compatível com inteiro
Char	Um único caractere Unicode de 16 bits. Não é compatível com inteiro
String	Até 2 bilhões de caracteres
Data	8 bytes – intervalo 01/01/100 até 31/12/9999

7.2.1. Atribuição – DIM

Para a definição das variáveis em VB, utilizaremos a instrução DIM, o nome da variável e o seu tipo. Confira no exemplo ilustrado pela figura 269.

Figura 269
Instrução DIM.

```
Dim x
Dim y As Integer
Dim nome As String
Dim endereco, cidade, estado As String
Dim md = 10
Dim salario As Double
```

7.2.2. Variáveis globais

Podemos definir, dentro das rotinas tendo visibilidade local, o código mostrado na figura 270.

Figura 270
Código para definição.

```
Private Sub Form1_Load()
    Dim nome As String
    Dim valor As Integer
    nome = ""
    valor = 0
End Sub
```
← Visibilidade dentro da rotina

Outra alternativa é fazer o mesmo dentro da classe, para que possam ser acessadas por outras rotinas da mesma classe. O código ilustrado na figura 271 implementa variáveis de visibilidade públicas dentro da aplicação.

Figura 271
Código implementando variáveis.

```
Public Class Form1
    Public frase As String
    Public calc As Double
    Private Sub Form1_Load()
        Dim nome As String
        Dim valor As Integer
        nome = ""
        valor = 0
    End Sub
End Class
```
← Visibilidade global

← Visibilidade dentro da rotina

7.3. Operadores

Os exemplos práticos para mostrar a funcionalidade dos operadores serão executados no Console Application.

7.3.1. Operadores aritméticos

ARITMÉTICOS	
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
\	Divisão de um valor por outro e retorna somente a parte inteira do resultado
Mod	Resto da Divisão
&	Combina (concatena) Strings
^	Expoente (potência)

Figura 272
Exemplo de operadores aritméticos.

```
Sub Main()
    Console.WriteLine("Verificando os Operadores")
    Dim x = 10
    Dim y = 15
    Console.WriteLine("Soma: " & (x + y))
    Console.WriteLine("Subtração: " & (x - y))
    Console.WriteLine("Multiplicação: " & (x * y))
    Console.WriteLine("Divisão: " & (y / x))
    Console.WriteLine("Parte inteira da divisão: " & (x \ 3))
    Console.WriteLine("Resto da Divisão (10/3): " & (x Mod 3))
    Console.WriteLine("Bom " & "Dia " & "aluno")
    Console.WriteLine("Quadrado de três: " & 3 ^ 2)
    Console.ReadKey()
End Sub
```

7.3.2. Operadores relacionais

RELACIONAIS	
>	Maior que
<	Menor que
>=	Maior ou igual a
<=	Menor ou igual a
=	Igual
<>	Diferente

7.3.3. Operadores aritméticos de atribuição reduzida

ARITMÉTICOS	
+=	Adição Igual
-=	Subtração Igual
*=	Multiplicação Igual
/=	Divisão Igual
&=	Concatena Igual
^=	Potência Igual

Figura 273
Exemplo de operadores aritméticos de atribuição reduzida.

```
Sub Main()
        Console.WriteLine("Operadores Reduzidos")
        Dim x As Integer = 10
        Dim y As Integer = 15
        Dim frase As String = "Bom"
        x += 2
        Console.WriteLine("Soma + igual: " & x)
        y -= 10
        Console.WriteLine("Subtração + igual: " & y)
        'x está com novo valor !!!
        x *= 2
        Console.WriteLine("Multiplicação + igual: " & x)
        x /= 2
        Console.WriteLine("Divisão + igual: " & x)
        frase &= " Dia!!!!"
        Console.WriteLine("Resto da Divisão + igual: " & frase)
        y ^= 2
        Console.WriteLine("Potência + igual: " & y)
        Console.ReadKey()
End Sub
```

7.3.4. Operadores lógicos

LÓGICOS	
And	And
Or	Or
not	Not

7.3.5. Conversões em VB.NET

O processo de conversão segue os conceitos abordados anteriormente (Java e C#). A lista ilustrada pela figura 274 sugere alguns exemplos de conversões.


```
x = CInt(idade)
y = CSng(salario)
z = CInt(Console.ReadLine())
dt = CDate("01/01/2010")
```

Figura 274
Exemplos de conversões.

7.4. Estrutura de decisão

Para realizar os desvios condicionais, utilizamos a estrutura if() ou Select Case(), nos formatos indicados a seguir.

7.4.1. Condição Verdadeiro – if

Neste exemplo, verificaremos se a variável "x" é maior do que o valor "10", sabendo-se que seu valor inicial é "5", visualizando a expressão: "A variável X é maior do que 10" (figura 275).

```
Sub Main()
        Dim x As Integer = 15
        If x >= 10 Then
            Console.WriteLine("A variável X é maior que 10")
            Console.ReadKey()
        End If
End Sub
```

Figura 275
Exemplo de verificação de variável.

7.4.2. Condição Verdadeiro ou Falso – if…else

Agora, verificaremos se a variável "x" é maior do que "10" ou não, considerando-se que seu valor inicial é "5". Será impressa uma expressão para cada alternativa (figura 276).

```
Sub Main()
        Dim x As Integer = 5
        If x >= 10 Then
            Console.WriteLine("A variável X é maior que 10")
        Else
            Console.WriteLine("A variável X é menor que 10")
        End If
        Console.ReadKey()
End Sub
```

Figura 276
Expressões diferentes para cada alternativa.

7.4.3. – Condições múltiplas – if...elseif...elseif....else

Verificaremos, agora, se a variável "x" possui o número 1, 2 ou 3, sabendo-se que o valor inicial é 03. Para outros valores, a expressão será: "A variável X é TRÊS" (figura 277).

Figura 277
Verificando condições múltiplas.

```
Sub Main()
        Dim x As Integer = 3
        If x = 1 Then
            Console.WriteLine("A variável X é UM")
        ElseIf x = 2 Then
            Console.WriteLine("A variável X é DOIS")
        ElseIf x = 3 Then
            Console.WriteLine("A variável X é TRÊS")
        Else
            Console.WriteLine("Qualquer outro valor")
        End If
        Console.ReadKey()
End Sub
```

7.4.4. – Múltiplos testes – Select Case()

Usando o mesmo exemplo anterior, a sequência de testes é realizada com a instrução Select Case(), a qual, deverá para cada teste estar implementada juntamente com a instrução break (figura 278). Assim, as outras condições não serão executadas. A instrução Case Else realiza a função da instrução else do if().

Figura 278
Múltiplos testes.

```
Sub Main()
        Dim x As Integer = 3
        Select Case x
            Case 1
                Console.WriteLine("O valor de X é UM")
            Case 2
                Console.WriteLine("O valor de X é DOIS")
            Case 3
                Console.WriteLine("O valor de X é TRÊS")
            Case Else
                Console.WriteLine("Qualquer outro valor")
        End Select
        Console.ReadKey()
End Sub
```

7.5. Estrutura de repetição

Vamos conhecer agora as estruturas de repetição utilizadas na linguagem.

7.5.1. While()

Usando a variável "cont" para controle do loop, serão visualizados os números de "0" até 10. O importante em uma instrução While() é a implementação de um contador dentro da estrutura (figura 279).

Figura 279
Instrução While().

```
Sub Main()
        Console.WriteLine("Estrutura WHILE")
        Dim cont As Integer = 0
        While (cont <= 10)
            Console.WriteLine("Numero: " & cont)
            cont = Cont + 1
        End While
        Console.ReadKey()
End Sub
```

7.5.2. Do While()...Loop

Representa a mesma estrutura da instrução anterior (figura 280).

Figura 280
Instrução Do While()...Loop.

```
Private Sub Form1_Load()
        Dim cont = 0
        Do While (cont <= 10)
            MsgBox("Numero: " & cont)
            cont = cont + 1
        Loop
End Sub
```

7.5.3. Do...Loop Until

Vale a mesma instrução anterior, mas, nesse caso, o teste é realizado no final do loop (figura 281). Esse tipo de estrutura permite que as instruções dentro do laço de repetição sejam executadas, no mínimo, uma vez.

Figura 281
Teste no fim do Loop.

```
Sub Main()
        Console.WriteLine("Estrutura WHILE")
        Dim cont As Integer = 0
        Do
            Console.WriteLine("Numero: " & cont)
            cont = cont + 1

        Loop Until cont >= 10
        Console.ReadKey()
End Sub
```

7.5.4. For

Diferentemente da instrução While(), a instrução For é capaz de definir, em uma única linha, a variável e o seu tipo, bem como estabelecer a condição para a estrutura e indicar o contador (figura 282).

7.5.5. For...Step

Contar de "0" a "10", mostrando o resultado. No entanto, dessa vez o contador será incrementado de 2 em 2 (figura 283).

Figura 282
Instrução For.

```
Sub Main()
    For x = 0 To 10
        Console.WriteLine("Número: " & x)
    Next
    Console.ReadKey()
End Sub
```

Figura 283
Instrução For...Step.

```
Sub Main()
    For x = 0 To 10 Step 2
        Console.WriteLine("Número: " & x)
    Next
    Console.ReadKey()
End Sub
```

7.6. Tratamento de erros e exceções

Nas versões anteriores do Visual Basic, o tratamento de erro era controlado pela instrução On error go to, ainda mantido por questões de compatibilidade. Porém, prefira utilizar o try-catch-finally, que possui as mesmas características estudadas anteriormente, mas com mudanças em suas linhas de programação, como podemos verificar no exemplo a seguir:

```
Try
    ' instruções que podem gerar o erro de execução
Catch
    ' o que deve ser feito se o erro ocorrer
Finally
    ' opcional, mas é executado
End Try
```

Assim, como nas outras linguagens (Java e C#), podemos capturar os valores de erros:

```
Catch erro As DivideByZeroException
```

No próximo exemplo, a estrutura Try foi organizada para verificar se existe erro no momento da conversão de dados das variáveis (figura 284).

Figura 284
Verificação de erro na conversão de dados das variáveis.

```
Try
    Dim var01, var02, resp As Double
    var01 = CDbl(TextBox1.Text)
    var02 = CDbl(TextBox2.Text)
    resp = var01 * var02
    TextBox3.Text = resp
Catch erro As DivideByZeroException
    MsgBox("Dados Incorretos")
Finally
    MsgBox("Mensagem de finalização", "Mensagem")
End Try
```

7.7. Vetores e matrizes

Vamos agora conhecer a forma de declaração, atribuição e acesso aos valores para diferentes tipos de vetores e matrizes (figura 285). Uma sugestão para consolidar o conhecimento é fazer uma pesquisa específica sobre vetores e matrizes de Lógica de Programação e Programação em Java. Há muita informação disponível, não apenas na bibliografia, como também em sites de busca.

Figura 285
Formas de declaração de vetores e matrizes.

```
' vetor de string
Dim j(2) As String
j(0) = "seg"
j(1) = "ter"
MsgBox(j(0))

' vetor de string
Dim semana() As String = {"dom", "seg", "ter", "qua", "qui", "sex"}
MsgBox(semana(0))

' vetor tipo Single
Dim y(3) As Single
y(0) = 10.5
y(1) = 7.3
y(2) = 1.9
MsgBox(y(1))

' vetor tipo Inteiro
Dim x() As Integer = {10, 5, 3}
MsgBox(x(2))

' matriz tipo double
Dim matriz(2, 2) As Double
matriz(0, 0) = 1
matriz(0, 1) = 2
matriz(1, 0) = 3
matriz(1, 1) = 4
```

```
MsgBox(matriz(1, 1))

' matriz tipo inteiro
Dim temp(,) As Integer = {{1, 4}, {2, 7}, {3, 5}}
MsgBox(matriz(1, 1))
```

7.8. Classes

Usando os mesmos conceitos de Java, o Visual Basic pode implementar classes específicas para programação, as quais seguem os mesmos princípios de formação e manipulação, incluindo os getters e setters.

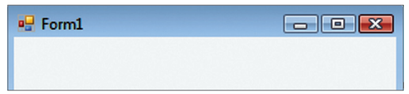
7.9. Windows Form Application – componentes

Da mesma forma que em C#, a janela ToolBox conta com vários componentes para o desenvolvimento de aplicações em VB. Para os componentes apresentados a seguir, utilize as mesmas descrições de propriedades mencionadas no capítulo 6.

7.9.1. Form

Quando a aplicação iniciar, aparecerá o Form1 (nome padrão) utilizado para o desenvolvimento dos trabalhos. Trata-se do principal repositório para os componentes, como mostra a figura 286.

Figura 286
Form.



7.9.2. Button

O Button (figura 287 a e b) é o responsável por grande parte da programação. Ao clicar nele, acessamos a janela de códigos na qual o primeiro evento, click, está previamente selecionado (figura 288).

Figura 287 a e b
Button.

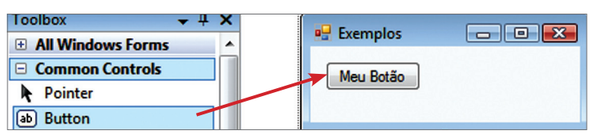


Figura 288
Evento Click.

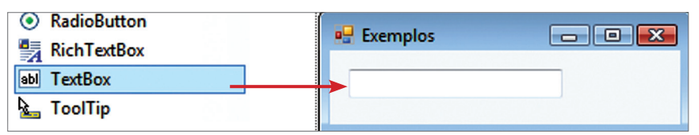
```
Private Sub btnMeuBotao_Click(ByVal sender As System.Object,


End Sub
```

7.9.3. TextBox

É o componente que recebe as informações do usuário, como ilustra a figura 289.

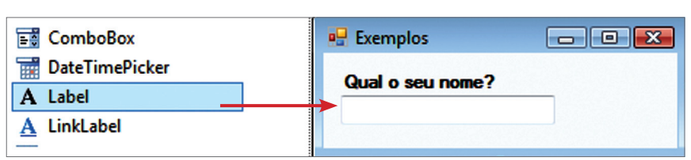
Figura 289
TextBox.



7.9.4. Label

O Label é usado para inserir rótulos nos formulários (figura 290).

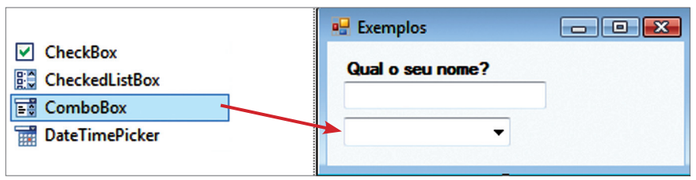
Figura 290
Label.



7.9.5. ComboBox

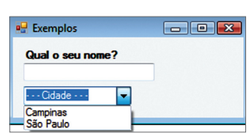
O ComboBox permite abrir uma cortina de opções ao usuário (figura 291).

Figura 291
ComboBox.



Para a inserção de itens, uma nova caixa de diálogo será aberta e os itens deverão ser colocados um abaixo do outro (one per line). Após a confirmação de tal execução, teremos o ComboBox carregado com as informações (figura 292).

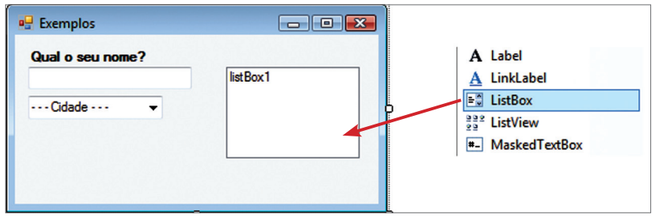
Figura 292
ComboBox carregado.



7.9.6. ListBox

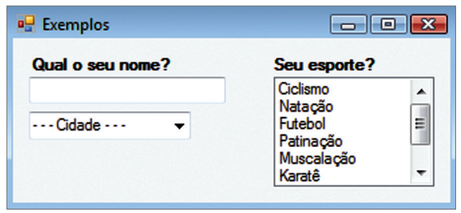
O ListBox também disponibiliza várias opções aos usuários, só que são abertas com barra de rolagem (figura 293).

Figura 293
ListBox.



Para carregar o ListBox (figura 294), use o mesmo procedimento do ComboBox.

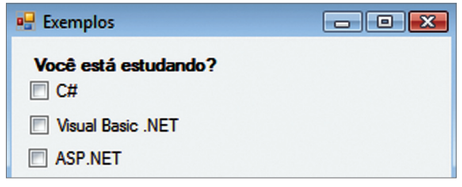
Figura 294
ListBox carregado.



7.9.7. CheckBox

Assim como no C#, vamos utilizar o CheckBox para indicar múltiplas opções ou questões de sim ou não. O exemplo da figura 295 é o mesmo utilizado em C#.

Figura 295
CheckBox.



A figura 296 mostra como fazer a verificação de seleção por meio de um botão.

O código descrito na figura 297 é referente apenas ao evento click, o qual está relacionado ao botão de verificação.

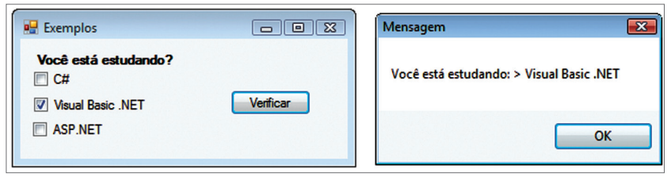


Figura 296
Verificação da caixa CheckBox.

```
Private Sub btnVerifica_Click()
        Dim frase As String
        frase = "Você está estudando:"
        If chkOpcao1.Checked = True Then
            frase = frase + " > C#"
        End If
        If chkOpcao2.Checked = True Then
            frase = frase + " > Visual Basic .NET"
        End If
        If chkOpcao3.Checked = True Then
            frase = frase + " > ASP.NET"
        End If
        MsgBox(frase, , "Mensagem")
End Sub
```

Figura 297
Código referente apenas ao evento click.

7.9.8. RadioButton

Vamos utilizar o mesmo exemplo do C# e solicitar ao usuário a escolha do estado civil (figura 298 e, com detalhes, na figura 299).

Figura 298
Verificando a opção do RadioButton.

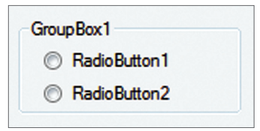
```
Dim frase As String
frase = "Seu estado Civil é:"
If rdbCasado.Checked = True Then
        frase = frase + " Casado"
End If
If rdbSolteiro.Checked = True Then
        frase = frase + " Solteiro"
End If
        MsgBox(frase, , "Mensagem")
End Sub
```

Figura 299
Detalhes da opção do RadioButton.

7.9.8.1. Agrupamento

Use um container para realizar o agrupamento dos componentes RadioButton, assim como no mesmo exemplo de C# (figura 300). Confira, em seguida, o quadro Propriedades RadioButton.

Figura 300
Agrupamento.

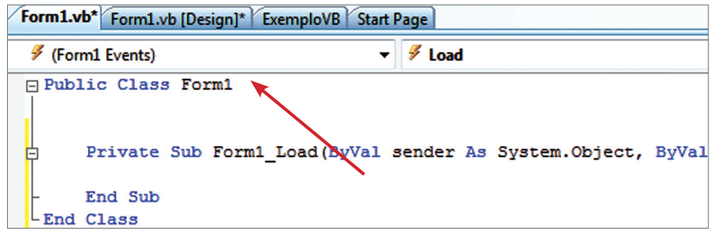


PROPRIEDADES RADIOBUTTON	
Name	Modifica o nome
Text	Título da caixa de grupo

7.10. Eventos

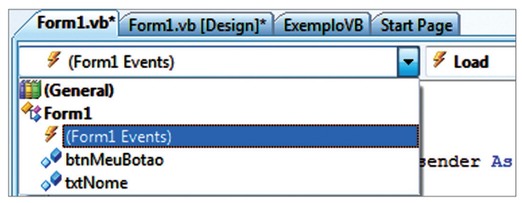
Existe uma maneira muito fácil de controlar os eventos em VB. Na parte superior da janela de Code, como mostra a figura 301, aparecem todos os componentes inseridos no formulário, incluindo o próprio formulário.

Figura 301
Componentes.



Clique para abrir o "combo" de opções (figura 302), franqueando o acesso aos componentes.

Figura 302
Componentes acionados.



Na outra extremidade da janela Code, há a lista de todos os eventos disponíveis para o componente (figura 303). Ao escolher um deles, automaticamente uma área de código será criada, para onde o cursor será deslocado.

Figura 303
Eventos.