Visual Basic .NET

Image Lists, Tree e List Views, Toolbars, Status e Progress Bars e Tab Controls

Professor: Danilo Giacobo

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

Objetivos da aula

- ✓ Trabalhar com o controle **Image List**
- ✓ Trabalhar com o controle **Tree View**
- ✓ Trabalhar com o controle **List View**
- ✓ Trabalhar com o controle **Toolbar**
- ✓ Trabalhar com o controle **Status Bar**
- ✓ Trabalhar com o controle **Progress Bar**
- ✓ Trabalhar com o controle **Tab Control**

Image Lists

- Componente que serve como um "repositório de imagens" para outros controles que possuem uma propriedade associada (**ImageList** e **ImageIndex**).
- São mais usados em controles do tipo **Tree View** e **List View**.
- A principal propriedade deste componente é a **Images** que contém as imagens a serem usadas pelo mesmo.
- A propriedade **ColorDepth** determina o número de cores que serão renderizadas.
- Todas as imagens serão mostradas com o mesmo tamanho definido na propriedade **ImageSize** (o padrão é 16x16, isto é, o tamanho de um ícone).

Usando a classe ImageList

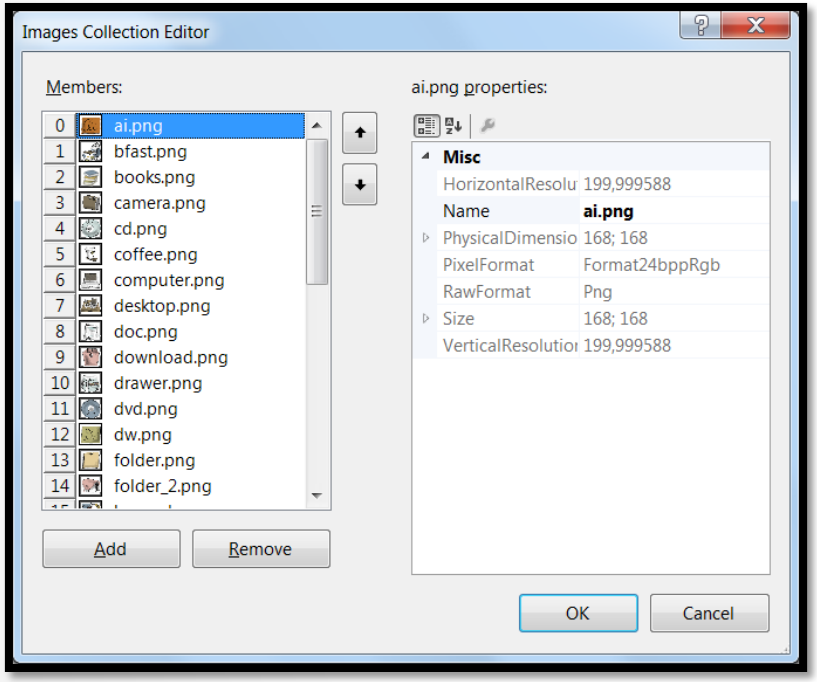
- Este componente quando adicionado ao projeto aparece na bandeja do VS .NET. Ele não aparece em tempo de execução.
- O método mais usado desta classe se chama **Draw**.
- As propriedades dignas de nota desta classe são:
  - Name
  - ColorDepth
  - Images
  - ImageSize
  - TransparentColor

- A hierarquia de classes desta classe é:

System
  Windows
    Forms
      ImageList

Criando uma Image List

- Você pode adicionar imagens ao um componente Image List clicando na propriedade **Images**. Ela abrirá uma janela onde você pode selecionar as imagens que farão parte da coleção a ser usada nos controles associados.



- Para alterar o tamanho das imagens use a propriedade **ImageSize**.
- Você pode também adicionar imagens em tempo de execução.

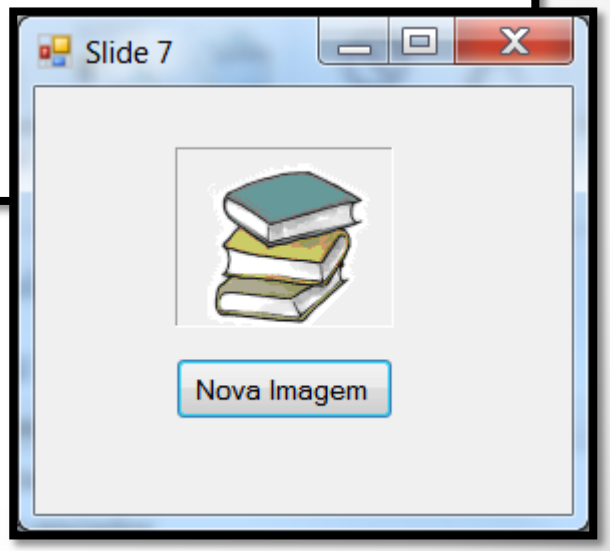
Usando uma Image List

- Este componente é designado para trabalhar com controles que suportam duas propriedades: **ImageList** e **ImageIndex**.
- A lista de controles inclui:
  - List Views
  - Tree Views
  - Toolbars
  - Checkboxes
  - Buttons
  - Radio Buttons
  - Labels
- Você associa uma lista de imagens a um controle usando a propriedade **ImageList** e configura a imagem a ser exibida usando a propriedade **ImageIndex** (começando em 0).

Usando uma Image List - Exemplo

- O exemplo abaixo usa um controle do tipo **Label** para exibir as imagens de um componente **ImageList**.
- Quando a pessoa clica no botão uma nova imagem é mostrada.
- Quando a última imagem é exibida o processo é reiniciado.

```
1  Public Class Form1
2      Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
3          If Label1.ImageIndex < ImageList1.Images.Count - 1 Then
4              Label1.ImageIndex += 1
5          Else
6              Label1.ImageIndex = 0
7          End If
8      End Sub
9  End Class
```



Usando uma Image List

- Você pode também trabalhar com controles que possuem apenas a propriedade **Image** ou **BackgroundImage** .
- O controle **Picture Box**, por exemplo, possui apenas a propriedade **Image**.
- O exemplo abaixo mostra como trabalhar com um **Image List** e um **Picture Box**.

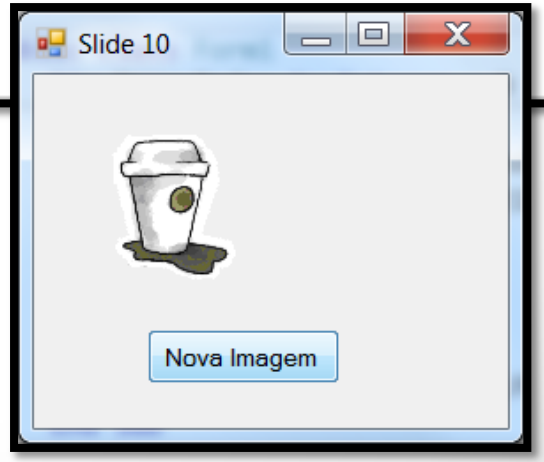
```
1  Public Class Form1
2      Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
3          Static ImageIndex As Integer = 0
4          If ImageIndex < ImageList1.Images.Count - 1 Then
5              ImageIndex += 1
6          Else
7              ImageIndex = 0
8          End If
9          PictureBox1.Image = ImageList1.Images(ImageIndex)
10     End Sub
11
12     Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
13         PictureBox1.Image = ImageList1.Images(0)
14     End Sub
15 End Class
```

Usando uma Image List com outros controles

- Você pode usar o método **Draw** desenhar uma imagem em um controle que você nem pensaria em mostrar uma imagem.
- O método Draw possui 4 parâmetros: um objeto do tipo **Graphics**, as coordenadas X e Y e o índice da imagem.
- Neste caso o evento **Paint** deve ser usado para realizar a tarefa.
- No próximo slide é apresentado um exemplo completo e o respectivo programa em execução.

Usando uma Image List com outros controles

```
1  Public Class Form1
2      Dim ImageIndex As Integer = 0
3
4      Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
5          If ImageIndex < ImageList1.Images.Count - 1 Then
6              ImageIndex += 1
7          Else
8              ImageIndex = 0
9          End If
10         ImageList1.Draw(Graphics.FromHwnd(Panel1.Handle), 0, 0, ImageIndex)
11     End Sub
12
13     Private Sub Panel1_Paint(sender As Object, e As PaintEventArgs) Handles Panel1.Paint
14         ImageList1.Draw(e.Graphics, 0, 0, ImageIndex)
15     End Sub
16 End Class
```

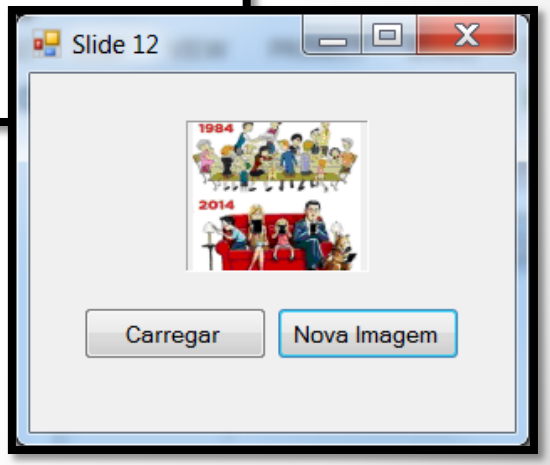


Adicionando Imagens a um Image List via código

- Se as imagens a serem adicionadas a um **Image List** não forem do mesmo tamanho elas serão redimensionadas.
- Você pode usar um componente do tipo **Open Dialog** para deixar a pessoa escolher quais imagens ela quer adicionar na lista.
- O método **Add** é usado para adicionar uma imagem ao **Image List**.
- No exemplo do próximo slide é mostrado um programa que deixa a pessoa selecionar uma ou várias imagens para colocar na lista.

Adicionando Imagens a um Image List via código

```
1  Public Class Form1
2      Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
3          If Label1.ImageIndex < ImageList1.Images.Count - 1 Then
4              Label1.ImageIndex += 1
5          Else
6              Label1.ImageIndex = 0
7          End If
8      End Sub
9
10     Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
11         If OpenFileDialog1.ShowDialog() = Windows.Forms.DialogResult.OK Then
12             If Not (OpenFileDialog1.FileNames Is Nothing) Then
13                 Dim intIndice As Integer
14                 For intIndice = 0 To OpenFileDialog1.FileNames.Length - 1
15                     ImageList1.Images.Add(Image.FromFile(OpenFileDialog1.FileNames(intIndice)))
16                 Next intIndice
17             Else
18                 ImageList1.Images.Add(Image.FromFile(OpenFileDialog1.FileName))
19             End If
20         End If
21     End Sub
22 End Class
```



Tree Views

- Controle usado para mostrar uma hierarquia de **nós**.
- Cada nó pode ou não possui **nós filhos**.
- Na plataforma Windows um exemplo de aplicação usando este controle é o próprio **Windows Explorer**.
- Você pode usar um componente **ImageList** para mostrar um ícone em cada nó da árvore.
- Você pode usar **checkboxes** próximos aos nós se a propriedade **CheckBoxes** estiver configurada para **True**.
- As principais propriedades são **Nodes** e **SelectedNode**.
- Os nós são suportados pela classe **TreeNode**.
- Para mudar o texto de um nó use a propriedade **Text**.
- Para navegar por um TreeView você pode usar as seguintes propriedades:
  - FirstNode
  - LastNode
  - NextNode
  - PrevNode
  - SelectedNode

Usando as classes TreeView e TreeNode

- A hierarquia de classes do controle **TreeView** é:

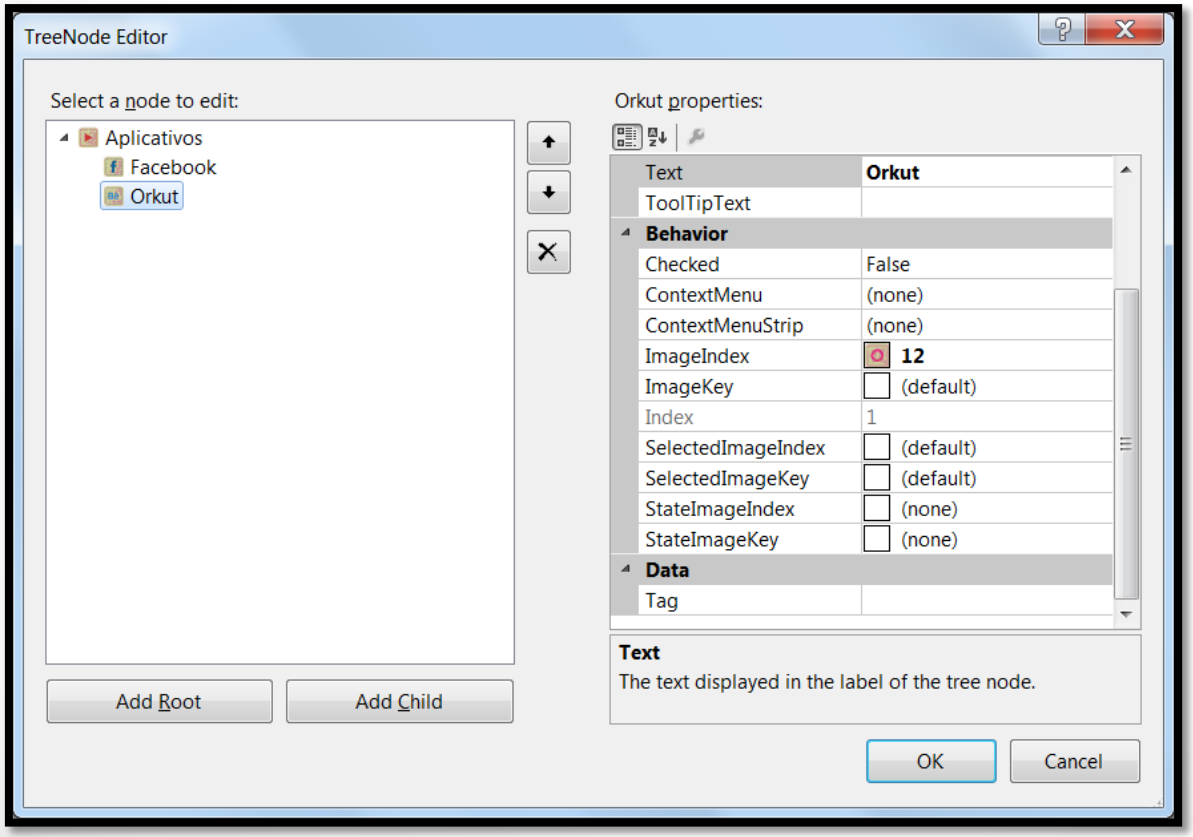
System
    Windows
        Forms
            TreeView

- A hierarquia de classes do controle **TreeNode** é:

System
    Windows
        Forms
            TreeNode

Criando Tree Views

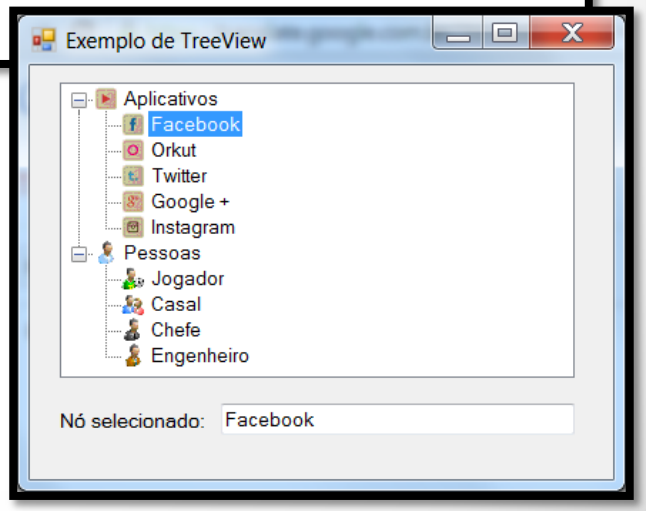
- Para criar um Tree View basta arrastar um controle deste tipo para um Windows form.
- Clique na propriedade **Nodes** para editar os itens da árvore.



Tree View - Eventos

- Controle que possui um grande número de eventos.
- O evento padrão é chamado de **AfterSelect** que ocorre quando um nó foi selecionado.
- Você pode determinar qual nó foi selecionado usando o objeto passado pela ocorrência do evento.
- O exemplo abaixo exibe o nó selecionado pela pessoa em um **Text Box**.

```
1  Public Class Form1
2      Private Sub TreeView1_AfterSelect(sender As Object, e As TreeViewEventArgs) Handles TreeView1.AfterSelect
3          TextBox1.Text = e.Node.Text
4      End Sub
5  End Class
```



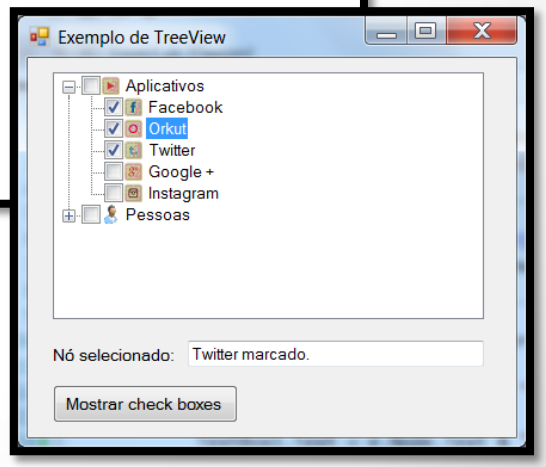
Tree View - Adicionando Checkboxes

- O controle **Tree View** suporta **checkboxes**.
- Para configurar a exibição do mesmo em cada nó da árvore é necessário configurar a propriedade **CheckBoxes** para **True**.
- O evento disparado quando um nó é marcado é o **AfterCheck**.

Exemplo:

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    TreeView1.CheckBoxes = Not TreeView1.CheckBoxes
End Sub

Private Sub TreeView1_AfterCheck(sender As Object, e As TreeViewEventArgs) Handles TreeView1.AfterCheck
    If e.Node.Checked Then
        TextBox1.Text = e.Node.Text & " marcado."
    Else
        TextBox1.Text = e.Node.Text & " desmarcado."
    End If
End Sub
```



Tree View - Criando em código

Atenção!

Este exemplo é longo e deve ser visto no próprio Visual Studio .NET.

List Views

- Controle usado para mostrar uma lista de itens.
- O Windows Explorer é uma mistura de **Tree View** e **List View**.
- O List View possui 5 modos de visualização (propriedade **View**):
  - LargeIcon
  - Details
  - SmallIcon
  - List
  - Tile
- A propriedade central deste controle é a **Items** que contém todos os itens exibidos pelo mesmo.
- Você pode usar a propriedade **SelectedItems** para obter os valores selecionados.
- Se a propriedade **MultiSelect** estiver como **True** a pessoa pode selecionar vários itens da lista.
- Para mostrar um check box ao lado do item basta alterar a propriedade **CheckBoxes** para **True**.
- O principal **evento** desta classe se chama **SelectedIndexChanged**.

Usando a classe List View

- A hierarquia de classes do controle **List View** é a seguinte:

System
    Windows
        Forms
            ListView

- As **propriedades** mais usadas desta classe são:
    - FullRowSelect
    - GridLines
    - Items
    - MultiSelect
    - SelectedIndices
    - SelectedItems
    - View

Usando a classe List View

- Os **métodos** mais utilizados desta classe são:
  - ArrangeIcons
  - BeginUpdate
  - Clear
  - EndUpdate
  - EnsureVisible
  - GetItemAt
- Os **eventos** mais importantes desta classe são:
  - AfterLabelEdit
  - BeforeLabelEdit
  - ColumnClick
  - ItemActivate
  - ItemCheck
  - SelectedIndexChanged

Usando a classe List View Item

- Os itens contidos em uma **List View** são objetos da clssse **ListViewItem**.
- A hierarquia de classes é a seguinte:

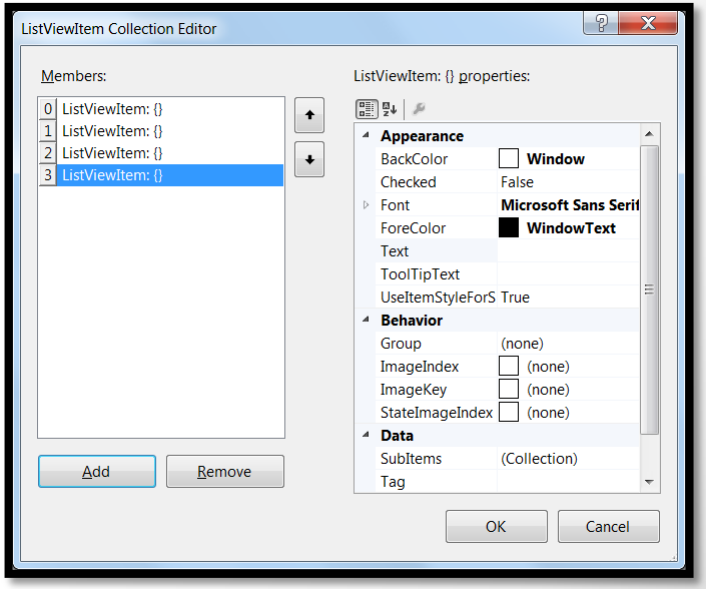
System
    Windows
        Forms
            ListViewItem

- As **propriedades** mais usadas desta classe são:
    - Checked
    - Index
    - Selected
    - SubItems
    - Text
- Os **métodos** mais usadas desta classe são:
    - BeginEdit
    - EndEdit

Criando List Views

- Para adicionar uma List View em um Windows form é necessário apenas arrastar um controle deste tipo para a área de design.
- Para adicionar itens simplesmente abra a propriedade **Items**.
- Você pode adicionar, remover e configurar cada item da lista a partir das propriedades disponíveis.
- A propriedade **View** controla a forma de apresentação dos itens. Ela possui os seguintes valores:

  - LargeIcon
  - Details
  - SmallIcon
  - List
  - Tile



List View - Criando em código

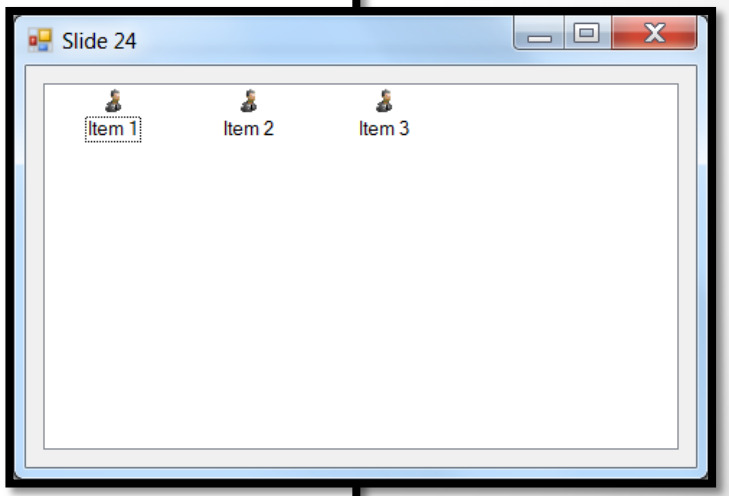
```vb
Public Class Form1
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        'Colunas
        ListView1.Columns.Add("Campo 1", CInt(ListView1.Width / 4), HorizontalAlignment.Center)
        ListView1.Columns.Add("Campo 2", CInt(ListView1.Width / 4), HorizontalAlignment.Center)
        ListView1.Columns.Add("Campo 3", CInt(ListView1.Width / 4), HorizontalAlignment.Center)
        ListView1.Columns.Add("Campo 4", CInt(ListView1.Width / 4), HorizontalAlignment.Center)

        'Linha 1
        Dim ListItem1 As ListViewItem
        ListItem1 = ListView1.Items.Add("Item 1", 1)
        ListView1.Items(0).SubItems.Add("Campo 2")
        ListView1.Items(0).SubItems.Add("Campo 3")
        ListView1.Items(0).SubItems.Add("Campo 4")

        'Linha 2
        Dim ListItem2 As ListViewItem
        ListItem2 = ListView1.Items.Add("Item 2", 1)
        ListView1.Items(1).SubItems.Add("Campo 2")
        ListView1.Items(1).SubItems.Add("Campo 3")
        ListView1.Items(1).SubItems.Add("Campo 4")

        'Linha 3
        Dim ListItem3 As ListViewItem
        ListItem3 = ListView1.Items.Add("Item 3", 1)
        ListView1.Items(2).SubItems.Add("Campo 2")
        ListView1.Items(2).SubItems.Add("Campo 3")
        ListView1.Items(2).SubItems.Add("Campo 4")

        ListView1.SmallImageList = ImageList1
        ListView1.LargeImageList = ImageList2
    End Sub
End Class
```

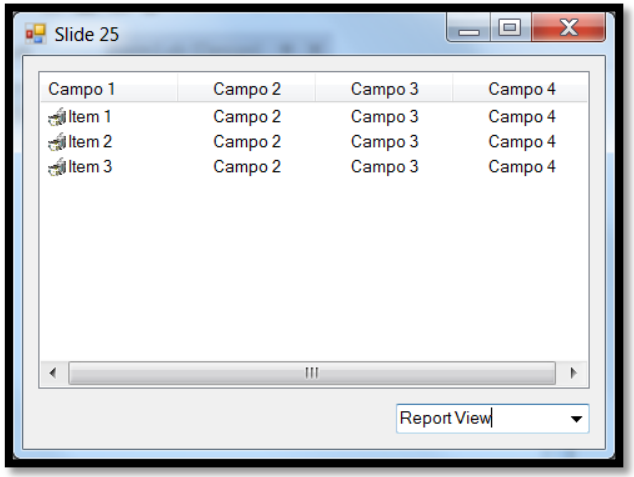


Selecionando as visões de um List View

- Há 5 tipos de visualizações em um **List View** configurados a partir da propriedade **View**.
- Usando o exemplo anterior um controle combo box é adicionado ao form para que a pessoa possa escolher o modo de visualização que desejar.

Exemplo:

```
'Itens do ComboBox
With ComboBox1
     .Items.Add("Large Icon View")
     .Items.Add("Report View")
     .Items.Add("Small Icon View")
     .Items.Add("List View")
     .Items.Add("Tile View")
End With
```



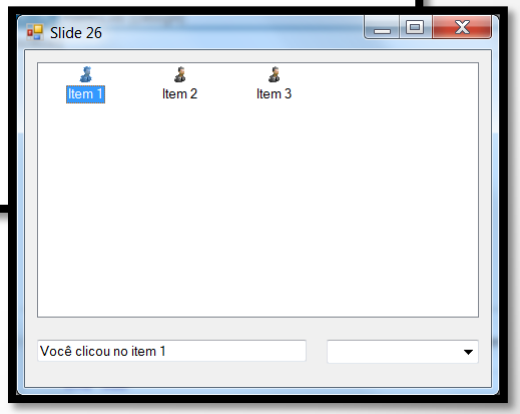

```
Private Sub ComboBox1_SelectedIndexChanged(sender As Object, e As EventArgs) Handles ComboBox1.SelectedIndexChanged
    ListView1.View = ComboBox1.SelectedIndex
End Sub
```

Obtendo a seleção de uma List View

- Para determinar qual item foi selecionado pela pessoa em um List View você pode usar o evento **SelectedIndexChanged** e verificar a propriedade **SelectedIndices** para determinar quais itens estão atualmente selecionados.
- A propriedade **SelectedIndices** contém os índices dos itens selecionados se a propriedade **MultiSelect** for igual a **True**.
- Você pode usar também a propriedade **SelectedItems** para obter uma coleção dos itens selecionados.

Exemplo de aplicação:

```
47      Private Sub ListView1_SelectedIndexChanged(sender As Object, e As EventArgs) _
48          Handles ListView1.SelectedIndexChanged
49          If ListView1.SelectedIndices.Count() > 0 Then
50              TextBox1.Text = "Você clicou no item " & _
51                  ListView1.SelectedIndices(0) + 1
52          End If
53      End Sub
```

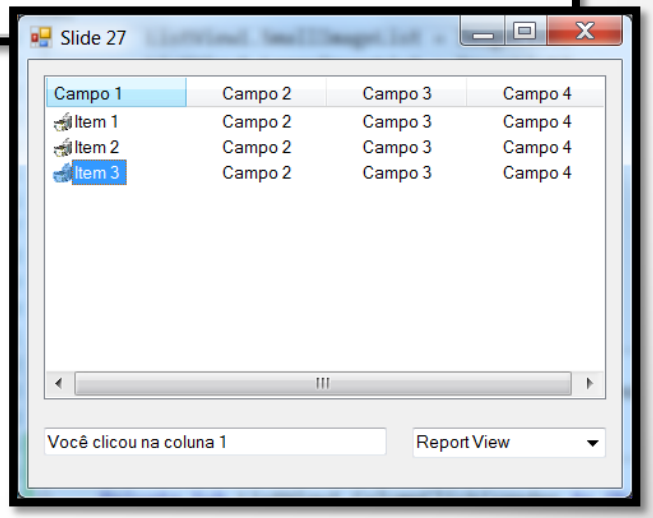


List Views - Eventos da Coluna

- Na visão **Report View**, o **List View** pode apresentar **colunas**. Você pode capturar o evento do clique em uma coluna usando o evento **ColumnClick**.
- Este evento pode ser útil se você quiser ordenar os itens de uma coluna, por exemplo.

Exemplo:

```
47      Private Sub ListView1_ColumnClick(sender As Object, e As ColumnClickEventArgs) _
48          Handles ListView1.ColumnClick
49          TextBox1.Text = "Você clicou na coluna " & (e.Column + 1)
50      End Sub
```



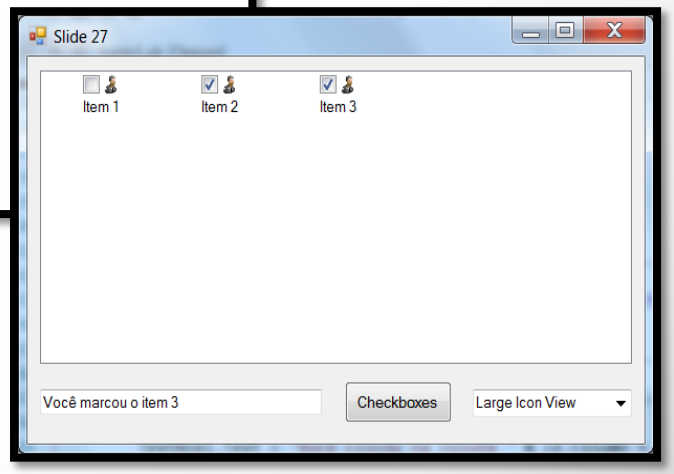
Usando Checkboxes em List Views

- **List Views** podem apresentar **checkboxes**.
- Tudo que você precisa fazer é alterar a propriedade **CheckBoxes** para **True**.
- Para manipular eventos é necessário usar o evento **ItemCheck**.

Exemplo:

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    ListView1.CheckBoxes = True
End Sub

Private Sub ListView1_ItemCheck(sender As Object, e As ItemCheckEventArgs) _
    Handles ListView1.ItemCheck
    If e.NewValue = CheckState.Checked Then
        TextBox1.Text = "Você marcou o item " & (e.Index() + 1)
    Else
        TextBox1.Text = "Você desmarcou o item " & (e.Index() + 1)
    End If
End Sub
```



Toolbars

- São barras com vários tipos de botões que ficam embaixo da barra de menu.
- Tipicamente ela é usada para apresentar os itens do menu mais acessados de uma aplicação.
- Elas podem ser colocadas em qualquer parte da janela usando a propriedade **Dock**.
- Elas também podem apresentar dicas quando a pessoa passa o mouse em cima de um botão. A propriedade **ShowToolTips** deve estar configurada como **True**).
- Para mudar a aparência de uma tool bar use a propriedade **Appearance** (Flat ou Normal).
- A propriedade **TextAlign** controla o alinhamento do texto em relação a imagem do botão.
- Em modo de design você pode usar a propriedade **Buttons** para adicionar os itens na sua barra de ferramentas.
- O evento mais importante deste componente se chama **ButtonClick**.
- Você consegue descobrir qual botão da barra foi clicado usando o objeto da classe **ToolBarButtonClickEventArgs**.

Usando a classe ToolBar

- A **hierarquia de classes** do controle **ToolBar** é a seguinte:

System
    Windows
        Forms
            ToolBar

- As **propriedades** dignas de nota desta classe são:
    - Appearance
    - AutoSize
    - Buttons
    - ImageList
    - ShowToolTips
    - TextAlign

- Os **eventos** mais usados são: **ButtonClick** e **ButtonDropDown**.

Usando a classe ToolBarButton

- Os botões em uma Tool Bar na realidade são objetos da classe **ToolBarButton**.
- A **hierarquia de classes** do controle **ToolBarButton** é a seguinte:

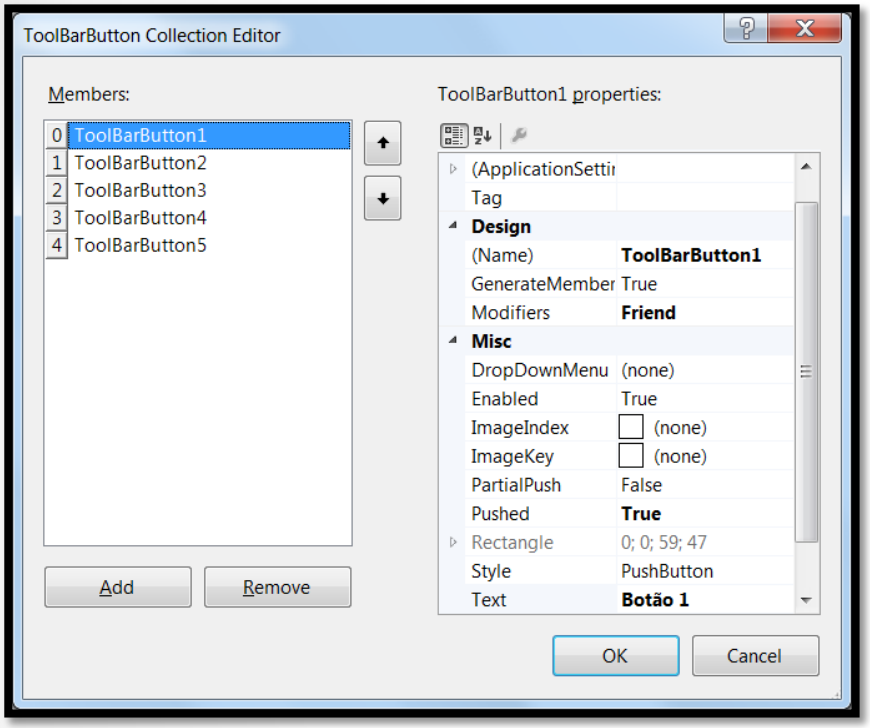
System
    Windows
        Forms
            ToolBarButton

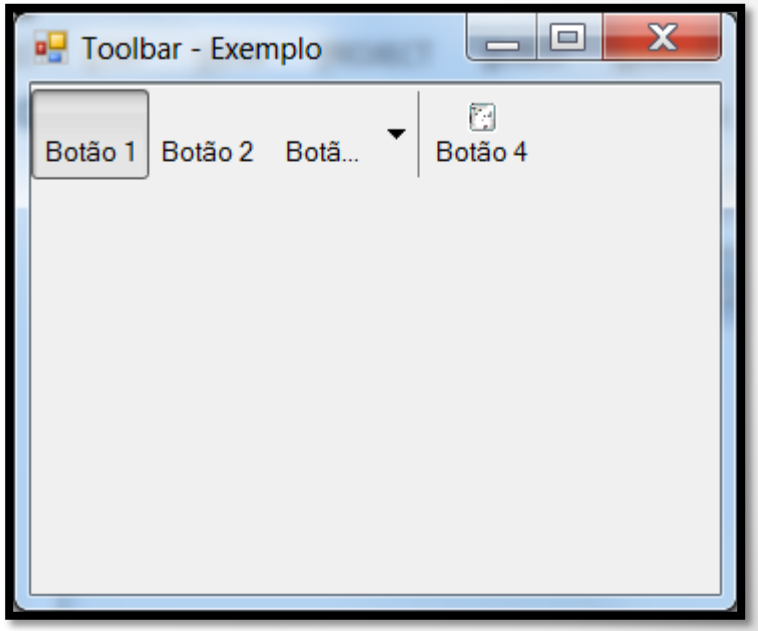
- As **propriedades** dignas de nota desta classe são:
    - DropDownMenu
    - Enabled
    - ImageIndex
    - Style
    - Text
    - ToolTipText
    - Visible

Criando Toolbars

- Depois que você adicionar uma Tool Bar em um Windows form você pode fixar ela em qualquer borda do mesmo usando a propriedade **Dock**.
- Para realmente adicionar botões a uma barra de ferramentas em tempo de design é necessário clicar na propriedade **Buttons**. Um editor será aberto para você poder adicionar, remover e configurar os itens.
- Para mudar o texto do botão use a propriedade **Text**.
- A imagem do botão pode ser configurando usando um componente **ImageList** e usando a propriedade **ImageIndex**.
- A propriedade **Style** contém as seguintes configurações:
  - PushButton
  - ToggleButton
  - Separator
  - DropDownButton

Criando Toolbars





ToolBars - Eventos

- Quando o botão de uma Toolbar é clicado, o evento **ButtonClick** ocorre.
- Você pode descobrir qual dos botões foi clicado usando a propriedade **Button** do objeto **ToolBarButtonClickEventArgs**.

Exemplo:

```
1  Public Class Form1
2      Private Sub ToolBar1_ButtonClick(sender As Object, e As ToolBarButtonClickEventArgs) _
3          Handles ToolBar1.ButtonClick
4          MsgBox("Você clicou no " & e.Button.Text)
5
6          If e.Button Is ToolBar1.Buttons(1) Then
7              MsgBox("Você clicou no segundo botão!")
8          End If
9      End Sub
10 End Class
```

ToolBars - Criando Drop-down Buttons

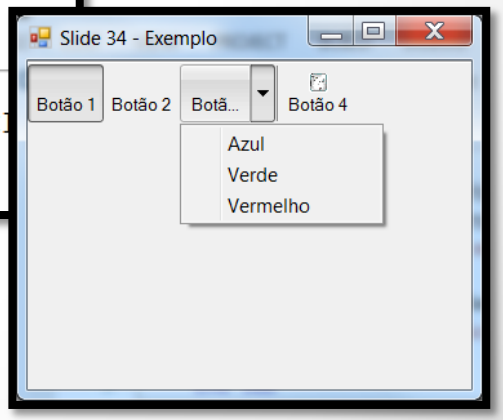
- Um botão do tipo drop-down mostra uma lista de opções para a pessoa quando ela clica no mesmo.
- Você pode adicionar um menu do tipo **ContextMenu** ao projeto e associar o mesmo a este botão na barra de ferramentas usando a propriedade **DropDownMenu**.

Exemplo:

```
Private Sub MenuItem1_Click(sender As Object, e As EventArgs) Handles MenuItem1.Click
    MsgBox("Você clicou na opção Azul.")
End Sub

Private Sub MenuItem2_Click(sender As Object, e As EventArgs) Handles MenuItem2.Click
    MsgBox("Você clicou na opção Verde.")
End Sub

Private Sub MenuItem3_Click(sender As Object, e As EventArgs) Handles MenuItem3.Cl
    MsgBox("Você clicou na opção Vermelho.")
End Sub
```



ToolBars - Conectando itens ao Menu

- Tipicamente botões da barra de ferramentas correspondem a itens de menu comumente usados. Para associar uma toolbar a um item de menu basta usar método **PerformClick**.

Exemplo:

```
If e.Button Is ToolBar1.Buttons(4) Then
    MenuItem1.PerformClick()
End If
```

```
Private Sub MenuItem1_Click(sender As Object, e As EventArgs) Handles MenuItem1.Click
    MsgBox("Você clicou na opção Azul.")
End Sub
```

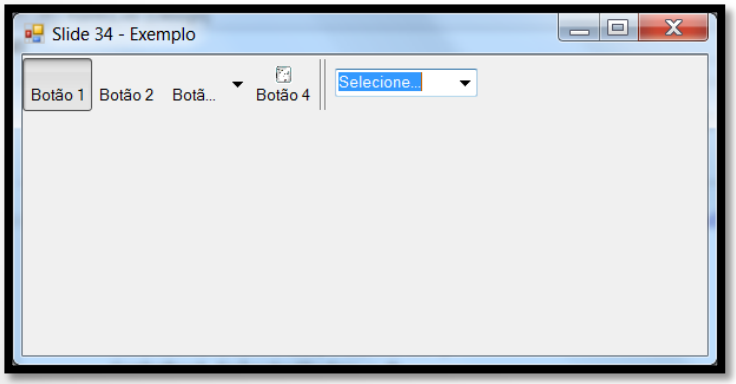
Dica: Para mostrar imagens em um botão de uma toolbar é necessário adicionar um componente do tipo **Image List** e configurar a propriedade **ImageIndex** com a imagem desejada.

ToolBars - Adicionando um Combo Box

- Você consegue adicionar outros componentes em uma Toolbar desde que tenha espaço na mesma para eles.
- O exemplo abaixo mostra um controle do tipo **ComboBox** inserido dentro da barra de ferramentas manipulando um evento.

```
Private Sub ComboBox1_SelectedIndexChanged(sender As Object, e As EventArgs) Handles ComboBox1.SelectedIndexChanged
    Dim selectedIndex As Integer
    selectedIndex = ComboBox1.SelectedIndex
    Dim selectedItem As Object
    selectedItem = ComboBox1.SelectedItem

    MsgBox("Texto do item selecionado: " & selectedItem.ToString & _
           " índice selecionado: " & selectedIndex.ToString)
End Sub
```

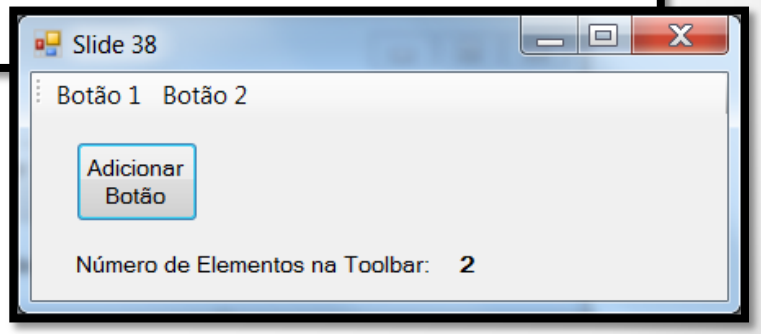


ToolBars - Adicionando botões via código

- Você pode adicionar botões em uma toolbar em tempo de execução. Para isso veja o exemplo abaixo no qual a partir do click de um botão um novo botão é inserido na barra de ferramentas da aplicação.

Exemplo:

```
1  Public Class Form1
2      Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
3          Static intBotao As Integer = 0
4          Dim ToolStripButton As New ToolStripButton("Botão " & intBotao + 1)
5          ToolStrip1.Items.Add(ToolStripButton)
6          intBotao = intBotao + 1
7          Label2.Text = intBotao.ToString
8      End Sub
9  End Class
```



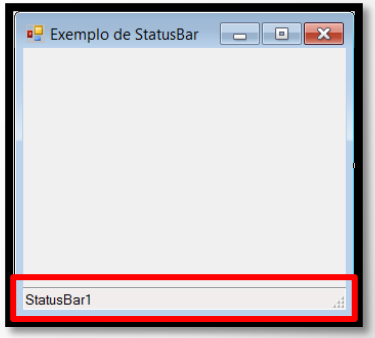
Status Bar

- Embora uma barra de status apareça logo em baixo de um menu, elas usualmente aparecem no rodapé da janela para informar o usuário sobre algum dado adicional (por exemplo: se o programa está conectado com a internet, a data e a hora, usuário conectado, etc.)
- Este controle pode apresentar painéis. Se ela não possui algum esta é conhecida como "simples" e exibe uma mensagem qualquer.
- A propriedade **ShowPanel** controla a exibição ou não dos painéis e por padrão ela é **False**.
- A propriedade **Text** configura a mensagem a ser exibida.
- Cada painel de uma barra de status pertence à classe **StatusBarPanel**.
- Acessando a coleção **Panels** você consegue verificar todos os painéis da mesma.
- O método **Add** da coleção **Panels** adiciona um novo painel à barra de status.
- Um objeto da classe **StatusBarPanel** contém os seus próprios métodos e propriedades.
- O evento principal desta classe se chama **PanelClick**.

Usando a classe StatusBar

- Uma Status Bar geralmente é usada para mostrar mensagens de status. Elas usualmente ficam no rodapé do form.
- As propriedades dignas de nota desta classe são;
  - Dock
  - Font
  - Panels
  - ShowPanels
  - Text
- O evento padrão desta classe é o **PanelClick**.
- A hierarquia de classes do controle **StatusBar** é a seguinte:

System
  Windows
    Forms
      StatusBar



Usando a classe StatusBarPanel

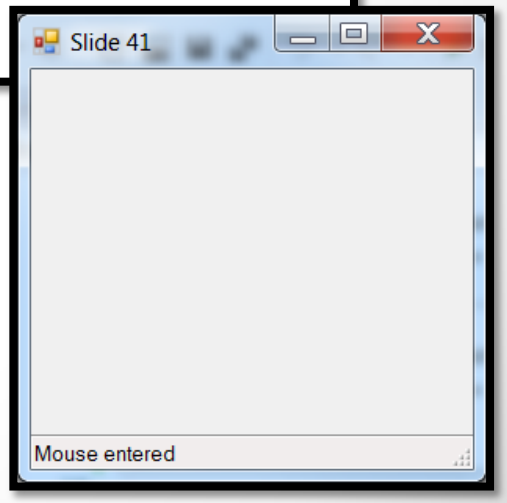
- Esta classe representa cada painel adicionado a uma barra de status.
- As propriedades dignas de nota desta classe são;
  - Alignment
  - BorderStyle
  - Icon
  - MinWidth
  - Parent
  - Style
  - Text
  - ToolTipText
  - Width
- A hierarquia de classes do controle **StatusBarPanel** é a seguinte:

System
    Windows
        Forms
            StatusBarPanel

Criando uma simples Status Bar

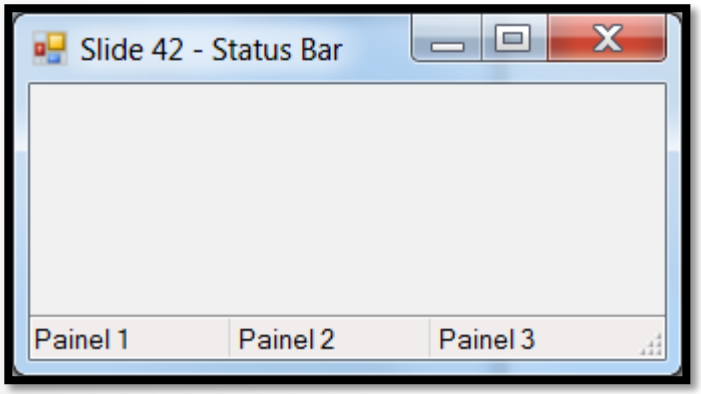
- Por padrão, quando você adiciona barra de status em um form, ela mostra apenas um simples texto. Usando a propriedade Text você consegue mudar o texto que está sendo exibido na mesma.
- No exemplo abaixo eu estou mostrando alguns eventos do mouse ocorrendo.

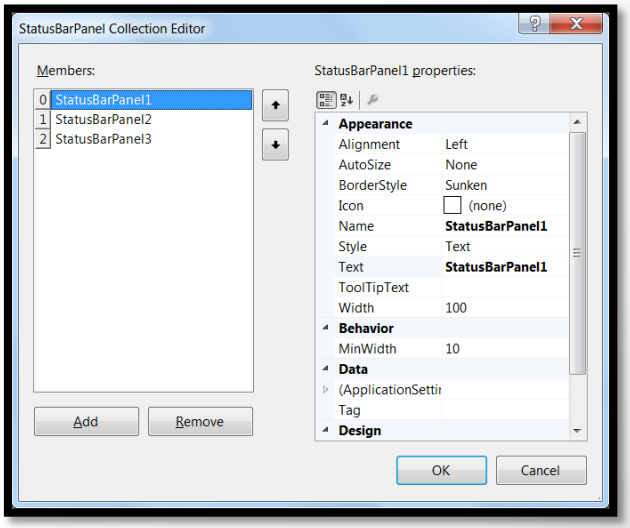
```
1  Public Class Form1
2      Private Sub Form1_MouseEnter(sender As Object, e As EventArgs) Handles Me.MouseEnter
3          StatusBar1.Text = "Mouse entered"
4      End Sub
5
6      Private Sub Form1_MouseLeave(sender As Object, e As EventArgs) Handles Me.MouseLeave
7          StatusBar1.Text = "Mouse left"
8      End Sub
9  End Class
```



Adicionando Painéis a uma Status Bar

- Você pode adicionar painéis a uma barra de status em modo de design usando a propriedade **Panels**. Ele abrirá um editor.
- Você pode usar o editor para adicionar novos painéis, configurar as propriedades destes e removê-los quando quiser.
- Para adicionar painéis em tempo de execução use os métodos **StatusBar.Panels.Add** e **StatusBar.Panels.AddRange**.
- Para remover painéis em tempo de execução use os métodos **StatusBar.Panels.Remove** e **StatusBar.Panels.RemoveAt**.



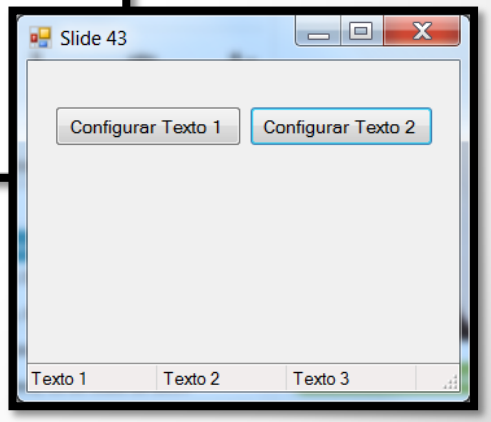


Colocando texto em um Status Bar Panel

- Para mostrar um texto em um painel de uma barra de status, simplesmente use a propriedade **Text**.
- Você pode usar a coleção **Panels** ou objeto **StatusBarPanel** diretamente.

Exemplo:

```
1  Public Class Form1
2      Private Sub Button3_Click(sender As Object, e As EventArgs) Handles Button3.Click
3          StatusBarPanel1.Text = "Text One"
4          StatusBarPanel2.Text = "Text Two"
5          StatusBarPanel3.Text = "Text Three"
6      End Sub
7
8      Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
9          StatusBar1.Panels(0).Text = "Texto 1"
10         StatusBar1.Panels(1).Text = "Texto 2"
11         StatusBar1.Panels(2).Text = "Texto 3"
12     End Sub
13 End Class
```

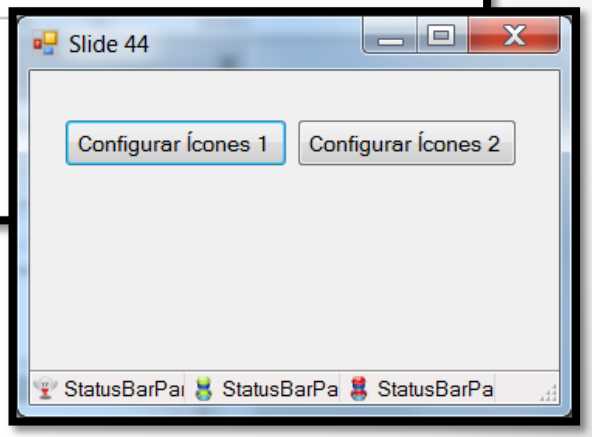


Adicionando ícones a uma Status Bar Panel

- Para adicionar um ícone em um painel (em modo de design) basta acessar a propriedade **Icon** e associar um arquivo com extensão **.ico** ao mesmo.
- Você pode usar a coleção **Panels** ou objeto **StatusBarPanel** diretamente para adicionar um ícone.

Exemplo:

```
1  Public Class Form1
2      Public Const CAMINHO = "C:\Users\Danilo Giacobo\Documents\Visual Studio 2012\Projects\Imagens\Iconshock-tiny-icons-lumina\ico"
3
4      Private Sub Button3_Click(sender As Object, e As EventArgs) Handles Button3.Click
5          StatusBarPanel1.Icon = New Icon(CAMINHO & "\boo.ico")
6          StatusBarPanel2.Icon = New Icon(CAMINHO & "\luigui.ico")
7          StatusBarPanel3.Icon = New Icon(CAMINHO & "\mario.ico")
8      End Sub
9
10     Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
11         StatusBar1.Panels(0).Icon = New Icon(CAMINHO & "\start.ico")
12         StatusBar1.Panels(1).Icon = New Icon(CAMINHO & "\toad.ico")
13         StatusBar1.Panels(2).Icon = Nothing
14     End Sub
15 End Class
```



Status Bar Panel - Eventos

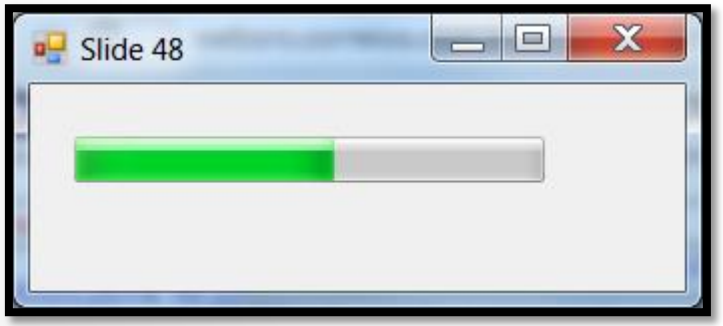
- Você usar o evento **PanelClick** para manipular eventos ocorrido por meio de cliques do mouse em um painel de uma status bar.
- O objeto passado na variável **e** contém qual o painel que foi clicado.

Exemplo:

```
1  Public Class Form1
2      Private Sub StatusBar1_PanelClick(sender As Object, e As StatusBarPanelClickEventArgs) _
3          Handles StatusBar1.PanelClick
4          If e.StatusBarPanel Is StatusBar1.Panels(0) Then
5              MsgBox("Você clicou no painel da esquerda!")
6          End If
7          If e.StatusBarPanel Is StatusBar1.Panels(1) Then
8              MsgBox("Você clicou no painel do centro!")
9          End If
10         If e.StatusBarPanel Is StatusBar1.Panels(2) Then
11             MsgBox("Você clicou no painel da direita!")
12         End If
13     End Sub
14 End Class
```

Progress Bar

- Simples controles que mostram o progresso de alguma operação por meio de retângulos em uma barra horizontal.
- Este controle é útil para que a pessoa saiba se o processamento de algo está no fim ou não. As vezes este valor é estimado apenas.
- As propriedades principais deste controle são: **Value**, **Minimum** e **Maximum**.
- Exemplo: se o valor mínimo é 10 e o máximo é 100 e o valor atual da barra é 60, então 6 retângulos irão aparecer.



Usando o controle ProgressBar

- Uma **Progress Bar** é um controle que apresenta vários retângulos dentro de um barra horizontal que permite ao usuário assistir o progresso de algumas operações.
- As propriedades dignas de nota desta classe são;
  - Font
  - ForeColor
  - Maximum
  - Minimum
  - Step
  - Value
- Os métodos dignos de nota desta classe são **Increment** e **PerformStep**.
- A hierarquia de classes do controle **ProgressBar** é a seguinte:

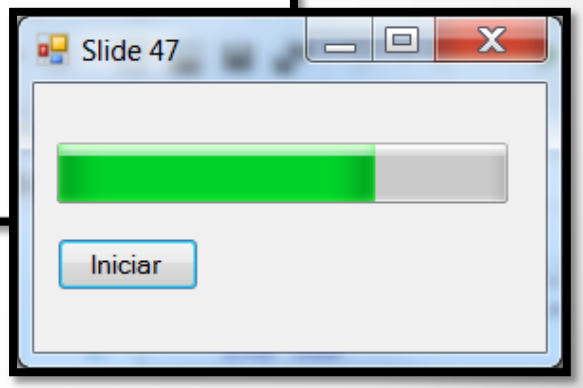
System
    Windows
        Forms
            ProgressBar

Criando uma Progress Bar

- As propriedades primárias de um controle Progress Bar, que semelhantes as de uma scroll bar, são **Minimum**, **Maximum** e **Value**.
- No exemplo abaixo foi usado um componente **Timer** para incrementar o valor da barra de progresso a cada ocorrência do evento **Tick**. Quando é atingido o valor máximo da barra o timer é parado.

Exemplo:

```
1  Public Class Form1
2      Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
3          Timer1.Enabled = True
4      End Sub
5
6      Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles Timer1.Tick
7          ProgressBar1.Value += 1
8          If ProgressBar1.Value = ProgressBar1.Maximum Then
9              Timer1.Enabled = False
10         End If
11     End Sub
12 End Class
```



Tab Control

- Controle que ajuda a aumentar o espaço de um form organizando os controles e funcionalidades em abas.
- Cada aba contém um espaço para que você possa adicionar novos componentes. Muitas aplicações Windows usam este tipo de controle. Você divide a sua tela em várias partes lógicas e com isso a mesma não fica "poluída" e para o usuário torna a interação mais simples e intuitiva.
- A propriedade central deste controle se chama **TabPages** que contém as abas individuais. Cada aba é um objeto da classe **TabPage**.
- Quando uma aba é clicada ela gera o evento **Click**.
- Você pode usar o método **Add** da coleção **TabPages** para adicionar novas abas e também o método **Remove** para excluir as mesmas.

Usando a classe TabControl

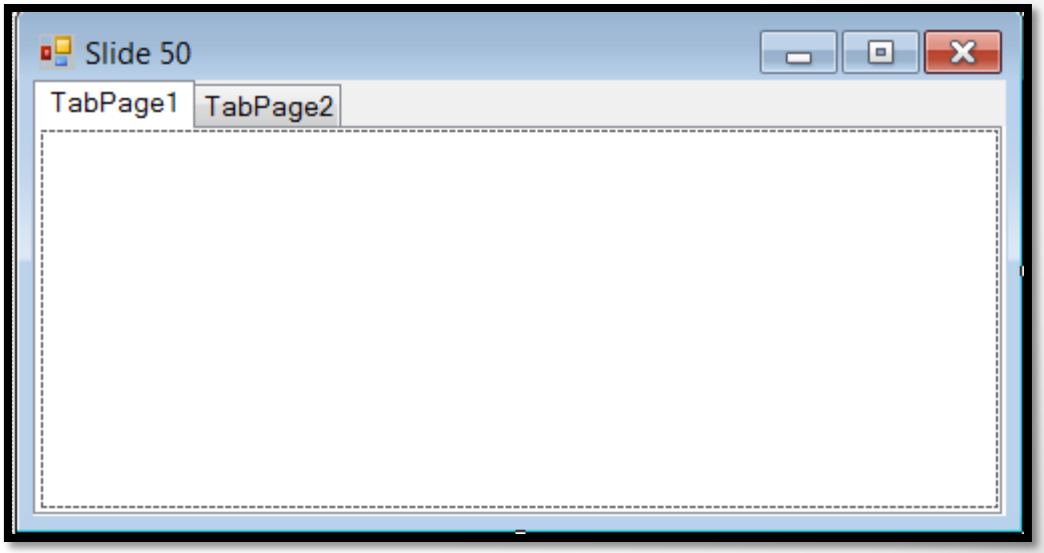
- Um **TabControl** permite você dividir sua área de visualização em várias abas que podem conter outros controles.
- As propriedades dignas de nota desta classe são;
  - RowCount
  - SelectedIndex
  - SelectedTab
  - TabCount
  - TabPages
- O evento digno de nota desta classe se chama **SelectedIndexChanged**.
- A hierarquia de classes do controle **TabControl** é a seguinte:

System
    Windows
        Forms
            TabControl

Usando a classe TabPage

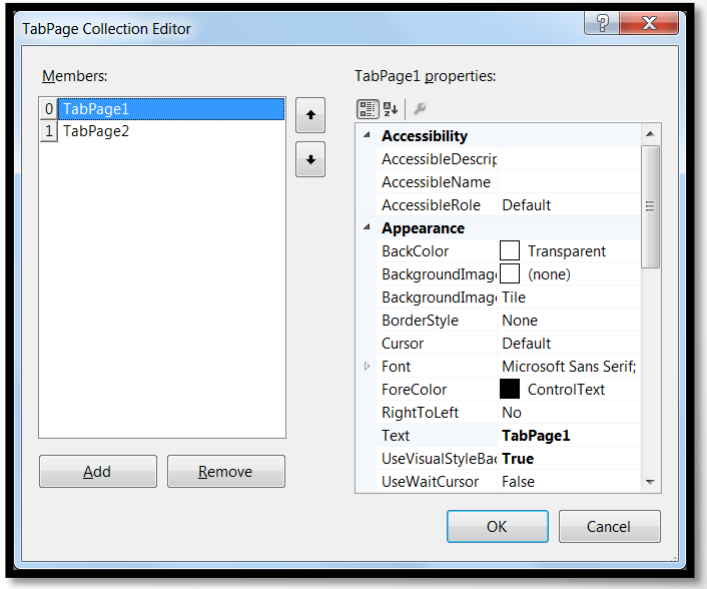
- Uma **TabPage** representa uma aba em um **TabControl**.
- As propriedades dignas de nota desta classe são;
  - ImageIndex
  - Text
  - ToolTipText
- A hierarquia de classes do controle **TabPage** é a seguinte:

System
  Windows
    Forms
      TabPage



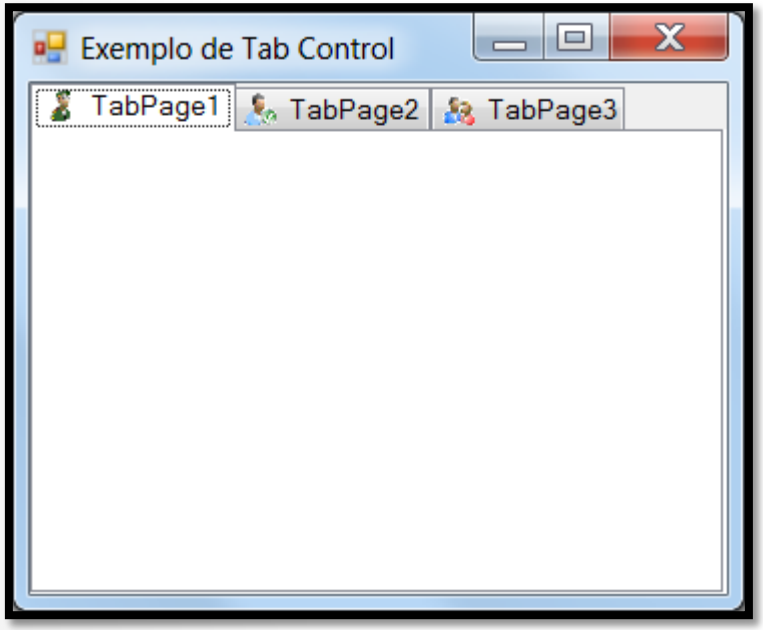
Criando uma Tab Control

- Depois que você adicionar um controle do tipo Tab Control ao projeto, você pode, em modo de design, adicionar abas à mesma usando a propriedade **TabPages**.
- No editor de abas do controle você pode adicionar, configurar e remover as mesmas.
- Configurando a propriedade **HotTrack** para **True**, o texto da aba muda quando o mouse é movido sobre a mesma.



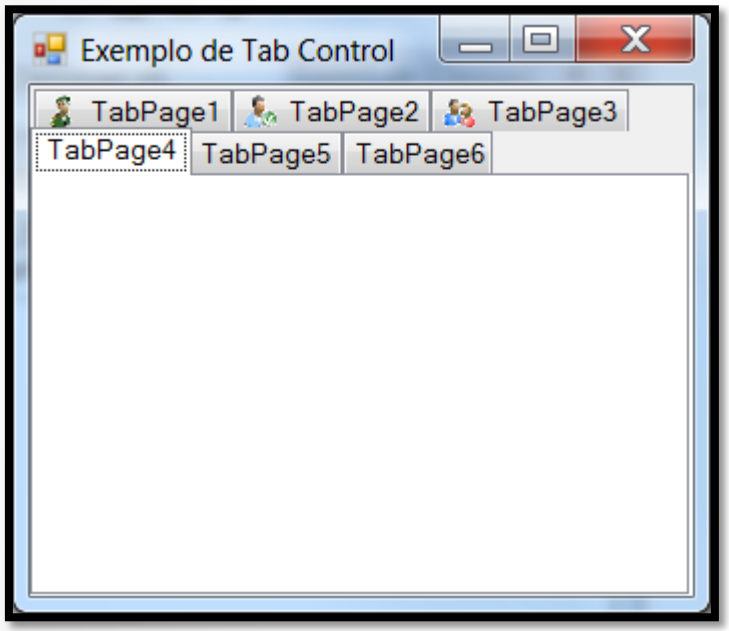
Tab Control - Outras configurações

- Para mostrar uma imagem em uma aba é necessário antes adicionar um componente **ImageList** ao projeto. Depois basta configurar a propriedade **ImageIndex** do objeto **TabPage**.



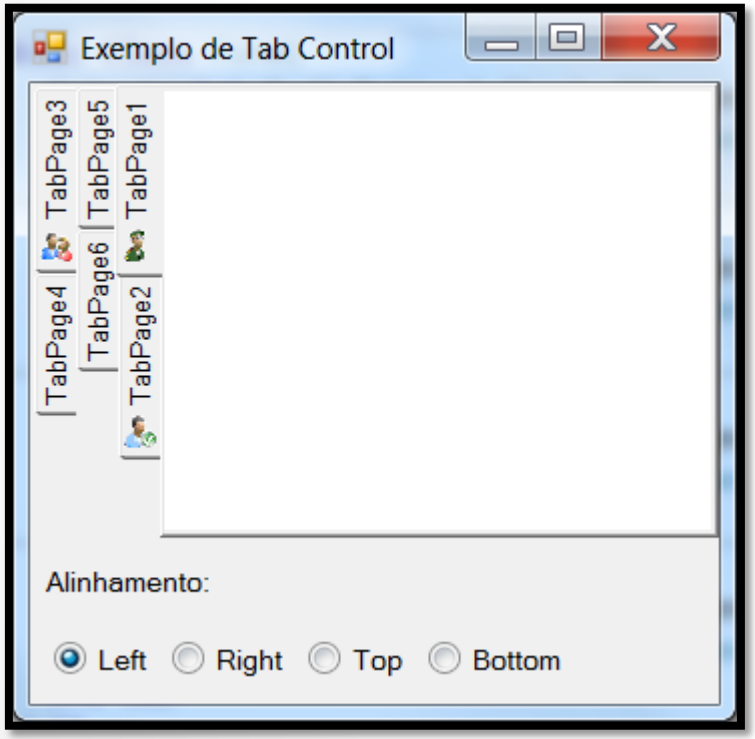
Tab Control - Outras configurações

- Para mostrar várias linhas de abas em um Tab Control, configure a propriedade **Multiline** para **True**.



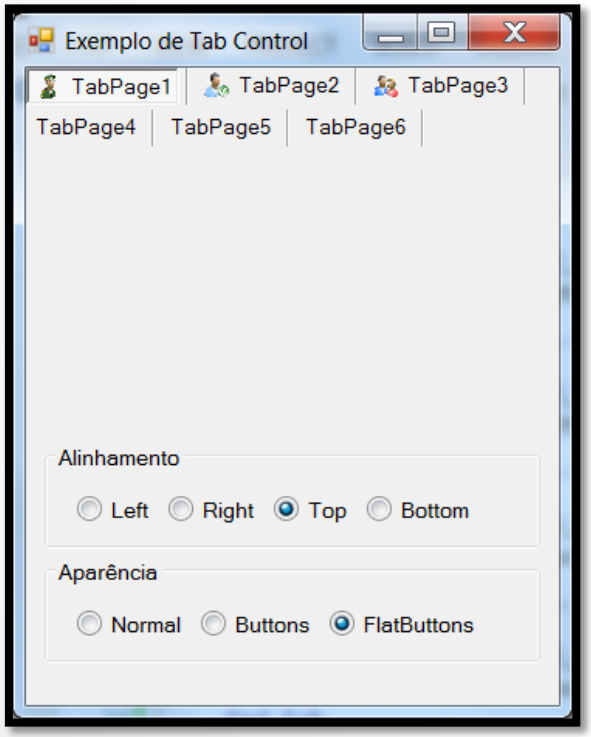
Tab Control - Outras configurações

- Você alterar a propriedade Alignment para especificar o alinhamento deste controle para **Left**, **Right**, **Top** ou **Bottom**.



Tab Control - Outras configurações

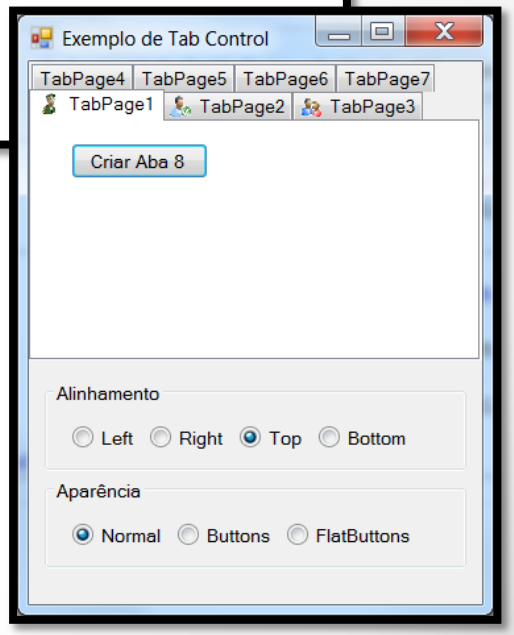
- Você pode fazer as abas de um Tab Control ficarem parecidas com botões. Apenas ajuste a propriedade **Appearance** do controle para **Buttons** ou **FlatButtons**.



Adicionando abas em tempo de execução

- Você pode usar o método **Add** da coleção **TabPages** para adicionar uma aba em um Tab Control em modo de execução.
- No exemplo abaixo quando a pessoa clica no botão **Criar Aba** uma nova aba é inserida no controle.

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    Dim tabpage As New TabPage
    tabpage.Text = "TabPage" & TabControl1.TabCount + 1
    TabControl1.TabPages.Add(tabpage)
    Button1.Text = "Criar Aba " & TabControl1.TabCount + 1
End Sub
```

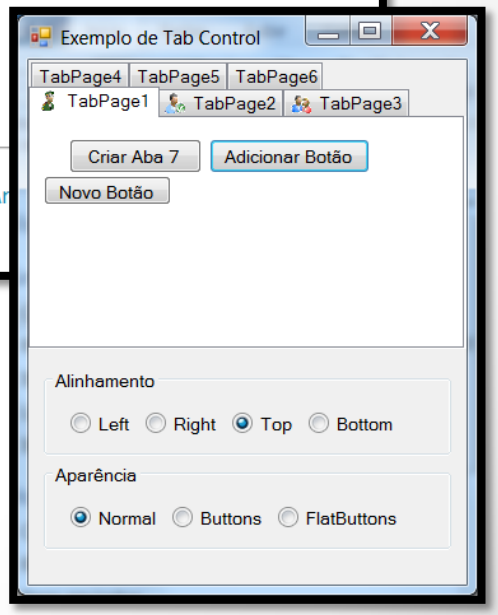


Tab Page - Adicionando controles em tempo de execução

- Para adicionar um controle em templo de execução em uma Tab Page é necessário usar o método **Add** da coleção **Controls** da mesma.

Exemplo:

```
41      Dim WithEvents MeuBotao As New Button()
42
43      Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
44          AddHandler MeuBotao.Click, AddressOf MeuBotao_Click
45          MeuBotao.Size = New Size(100, 23)
46          MeuBotao.Location = New Point(8, 45)
47          MeuBotao.Text = "Novo Botão"
48          TabControl1.TabPages(0).Controls.Add(MeuBotao)
49      End Sub
50
51      Private Sub MeuBotao_Click(ByVal sender As System.Object, ByVal e As System.EventAr
52          MsgBox("Você clicou no botão!")
53      End Sub
```



Referências Bibliográficas

- HOLZNER, Steven. **Visual basic.NET: black book**. Arizona: Coriolis Group Books, 2002. xxxviii, 1144 p ISBN 1-57610-835-X.