Estácio

Desenvolvimento de Software

Prof. Henrique Mota

mota.henrique@gmail.com

http://www.henriquemota.com.br

Programando “Forms”

Windows Forms

Estácio

A plataforma .NET fornece dois *toolkits* para criação de GUI:

Windows Forms

Namespace `System.Windows.Forms`

Aplicações *Desktop* tradicionais

Interfaces de apresentação sofisticadas em aplicações distribuídas (*fat clients*)

Web *Forms*

Namespaces `System.Web.UI` e `System.Web.UI.WebControls`

Utilizado no desenvolvimento de ASP.NET

Interfaces independentes do *User Agent* (*browser*) cliente, tendo por base os standards da indústria (HTML, HTTP, XML, etc.).

Uma aplicação Windows Forms

Estácio

Aplicativos de GUI são baseadas na noção de formulários e controles ...

- Um formulário representa uma janela
- Um formulário contém 0 ou mais controles
- Um controle interage com o usuário

Namespace `System.Windows.Forms`

Estácio

Windows Form class	*Description*
`Application`	*Representa as entranhas de um aplicativo Windows Forms. Usa os métodos de aplicação, que são capazes de processar as mensagens do Windows, iniciar e encerrar um aplicativo Windows Forms, e assim por diante.*
`ButtonBase, Button, CheckBox, ComboBox, DataGrid, GroupBox, ListBox, LinkLabel, PictureBox`	*Representam tipos que correspondem a vários widgets GUI.*
`Form`	*Representa uma janela principal (ou caixa de diálogo) de um aplicativo Windows Forms.*
`ColorDialog, FileDialog, FontDialog, PrintPreviewDialog`	*Define uma série de diálogos., Você é livre para construir diálogos personalizados.*
`Menu, MainMenu, MenuItem, ContextMenu`	*Usados para construir sistemas de menu e menu sensível ao contexto (pop-up).*
`Clipboard, Help, Timer, Screen, ToolTip, Cursors`	*Representam vários tipos de serviços públicos para facilitar a interaçao com GUIs.*
`StatusBar, Splitter, ToolBar, ScrollBar`	*Representam vários tipos para enfeitar um formulário.*

Classe Application

Estácio

- Classe utilitária para controle dos diversos comportamentos de uma aplicação *Windows Form*.
- Define um conjunto de eventos que representam acontecimentos da aplicação (p.ex. shutdown e estado *idle*)
- Conjunto de propriedades (na maioria *read-only*) que representam características da aplicação (algumas destas são formas mais simples de acesso à *metadata*)

Métodos de `Application`

Method	***Description***
`AddMessageFilter()` `RemoveMessageFilter()`	*These methods allow your application to intercept messages for any necessary preprocessing. When you add a message filter, you must specify a class that implements the IMessageFilter interface (as you will do shortly).*
`DoEvents()`	*Provides the ability for an application to process messages currently in the message queue, during a lengthy operation (such as a looping construct).* *Think of DoEvents() as a quick and dirty way to simulate multithreaded behaviors.*
`Exit()`	*Terminates the application.*
`ExitThread()`	*Exits the message loop on the current thread and closes all windows owned by the current thread.*
`OLERequired()`	*Initializes the OLE libraries. Consider this the .NET equivalent of manually calling OleInitialize().*
`Run()`	*Begins running a standard application message loop on the current thread.*

Propriedades de `Application`

Estácio

Property	***Description***
`CommonAppDataRegistry`	*Retrieves the registry key for the application data that is shared among all users.*
`CompanyName`	*Retrieves the company name associated with the current application.*
`CurrentCulture`	*Gets or sets the locale information for the current thread.*
`CurrentInputLanguage`	*Gets or sets the current input language for the current thread.*
`ProductName`	*Retrieves the product name associated with this application.*
`ProductVersion`	*Retrieves the product version associated with this application.*
`StartupPath`	*Retrieves the path for the executable file that started the application.*

Eventos de `Application`

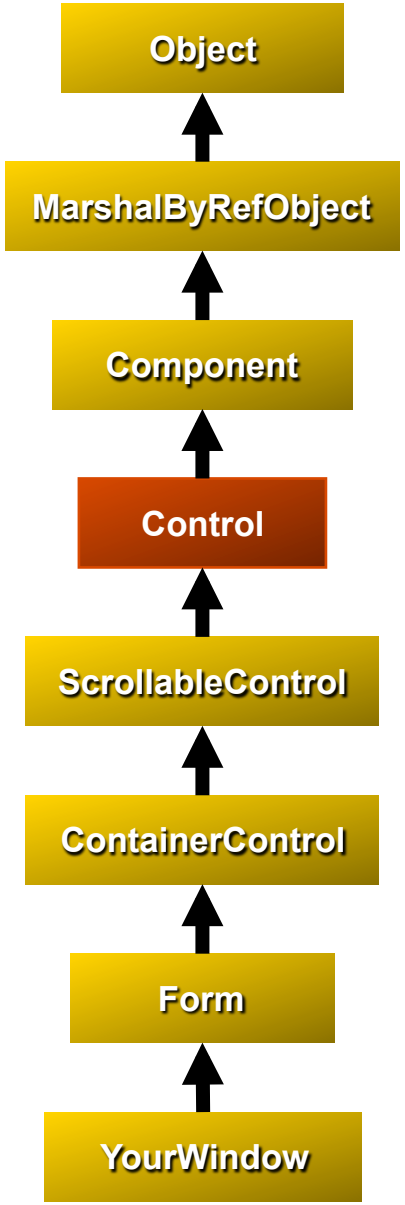
Event	*Description*
`ApplicationExit`	*Occurs when the application is just about to shut down.*
`Idle`	*Occurs when the application message loop has finished processing and is about to enter an idle state (meaning there are no messages to process at the current time).*
`ThreadExit`	*Occurs when a thread in the application is about to terminate. If the main thread for the application is about to shut down, this event will be raised before the ApplicationExit event.*

Hierarquia - (Control)

Define o comportamento de um componente gráfico (GUI *widget*)

Possui membros para:

- Definir dimensão e posição
- Obter HWND (*handle*) subjacente
- Capturar eventos de mouse e teclado
- Configurar Cores de *foreground* e background
- Imagens de background
- Características da Fonte
- *Drag-and-drop*
- *Context Menus*
- Comportamento de *docking* e *anchoring* (*Layout*)
- Desenhar (pintar) a área cliente (GDI+)



Propriedades de `Control`

Control Property	***Description***
`Top, Left, Bottom, Right, Bounds, ClientRectangle, Height, Width`	*Each oh these properties specifies various attributes about the current dimensions of the Control-derived object.* *Bounds returns a Rectangle that specifies the size of the control. ClientRectangle returns a rectangle that corresponds to the size of the client area of the control.*
`Created, Disposed, Enabled, Focused, Visible`	*These properties each return a Boolean that specifies the state of the current control.*
`Handle`	*Returns a numerical value (integer) which represents the HWND of this control.*
`ModifierKeys`	*This static property checks the current state of the modifier keys (shift, control, and alt) and returns the state in Keys type.*
`MouseButtons`	*This static property checks the current state of the mouse buttons (left, right, and middle mouse buttons) and returns this state in a MouseButtons type.*
`Parent`	*Returns a Control object that represents the parent of the current control.*
`TabIndex, TabStop`	*These properties are used to configure the tab order of the Control.*
`Text`	*The current text associated with this Control.*

Propriedades de `Control`

Control Property	*Description*
`AllowDrop`	*If AllowDrop is set to true then this control allows drag-and-drop operations and events to be used.*
`Anchor`	*The anchor property determines which edges of the control are anchored to the container´s edges.*
`BackColor, BackGroundImage, Font, ForeColor, Cursor`	*These properties configure how the client area should be displayed.*
`ContextMenu`	*Specifies which context menu (e.g., pop-up menu) will be shown when the user right clicks the control.*
`Dock`	*The dock property controls to which edge of the container this control is docked to. For example, when docked to the top of the container, the control is displayed flush at the top of the container, extending the length of the container.*
`Region`	*This property configures a Region object that specifies the outline/silhouette/ boundary of the control.*
`RegionToLeft`	*This is used for international applications where the language is written from right to left.*

Métodos de `Control`

Control Method	Description
`GetStyle()`, `SetStyle()`	*These method are used to manipulate the style flags of the current Control using the ControlStyles enumeration.*
`Hide()`, `Show()`,	*These methods indirectly set the state of the Visible property.*
`Invalidate()`	*Forces the Control to redraw itself by forcing a paint message into the message queue.* *This method is overloaded to allow you to specify a specific Rectangle to refresh, rather than the entire client area.*
`OnXXXX()`	*The Control class define numerous methods that can be overridden by a subclass to respond to various events (e.g, OnMouseMove(), OnKeyDown(), OnResize(), etc.). As you will see later in this chapter, when you do wish to intersect a GUI-based event, you have two approaches. One approach is to simply override one of the existing event handlers. Another is to add a custom event handler to a given delegate.*
`Refresh()`	*Force the control to invalidate and immediately repaint itself and any children.*
`SetBounds()`, `SetLocation()`, `SetClientArea()`	*Each of these methods is used to establish the dimensions of the Control derived object.*

Métodos de `Control`

Control Method	***Description***
`DoDragDrop()`, `OnDragDrop()`, `OnDragEnter()`, `OnDragLeave()`, `OnDragOver()`	*These methods are used to monitor drag-and-drop operations for a given Control descendent.*
`ResetFont()`, `ResetCursor()`, `ResetForeColor()`, `ResetBackColor()`	*These method reset various UI attributes of a child control to the corresponding value of the parent..*
`OnPaint()`	*Inheriting classes should override this method to handle the Paint event.*

Eventos de `Control`

Estácio

A classe *Control* também define um conjunto de eventos que podem ser logicamente agrupados em quatro categorias:

1. eventos do mouse
2. eventos do teclado.
3. *Drag-and-drop*
4. Operações de *painting*

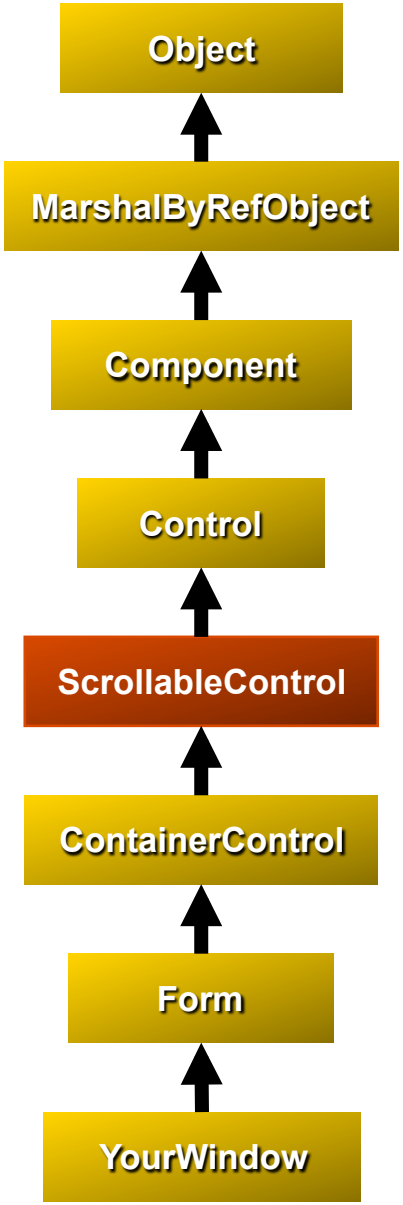
Control Event	***Description***
`Click, DoubleClick, MouseEnter, MouseLeave, MouseDown, MouseUp, MouseMove,MouseWhell`	*The Control class defines numerous events triggered in response to mouse input.*
`KeyPress, KeyUp, KeyDown,`	*The Control class also defines numerous events triggered in response to keyboard input.*
`DragEnter, DragLeave, DragDrop, DragOver`	*These events are sent in response to drag-and-drop operations.*
`Paint`	*This event is sent whenever the Control has become "dirty" and needs to be repainted.*

Hierarquia – ScrollableControl

Definição de membros para suporte ao *scroll* vertical e horizontal

```
// This could be set in the class constructor or InitializeComponent().
// Note that you need to reference the System.Drawing namespace to
// gain access to the Size type.
   this.AutoScroll = true;
   this.AutoScrollMinSize = new System.Drawing.Size(300, 300);
```

Properties	Description
AutoScroll	*Gets or sets a value indicating whether the container will allow the user to scroll to any controls placed outside of its visible boundaries.*
AutoScrollMinSize	*Gets or sets the minimum size of the auto-scroll.*
VScroll	*Gets or sets a value indicating whether the vertical scroll bar is visible.*
HScroll	*Gets or sets a value indicating whether the horizontal scroll bar is visible.*

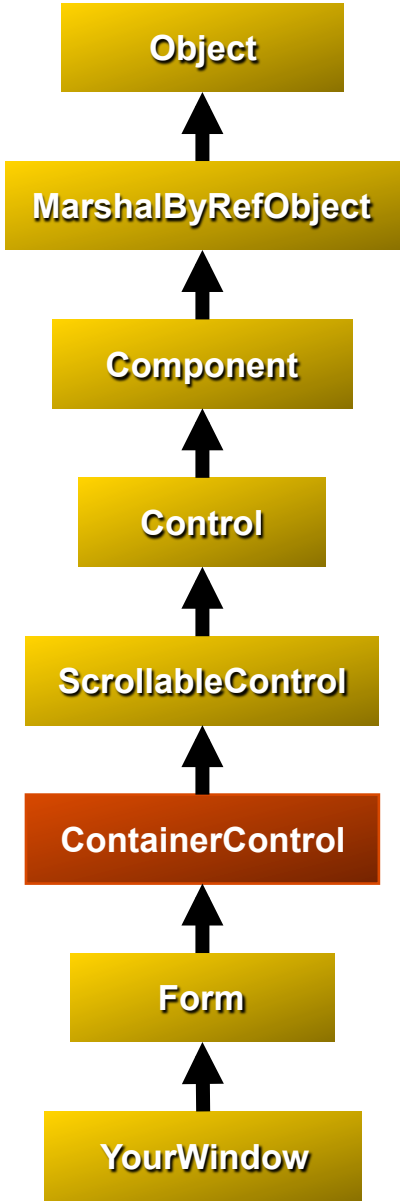


Hierarquia – `ContainerControl`

Controle que pode servir de *container* de outros controles (filhos) e fornece mecanismos para gerir o foco dos controlos filhos

Container Control Members	***Description***
`ActiveControl` `ParentControl`	*These properties allow you to obtain and set the active control, as well as retrieve a reference to the Form that is hosting the item.*
`ProcessTabKey()`	*This method allows you to programmatically activate the Tab key to set the focus to the next available control.*

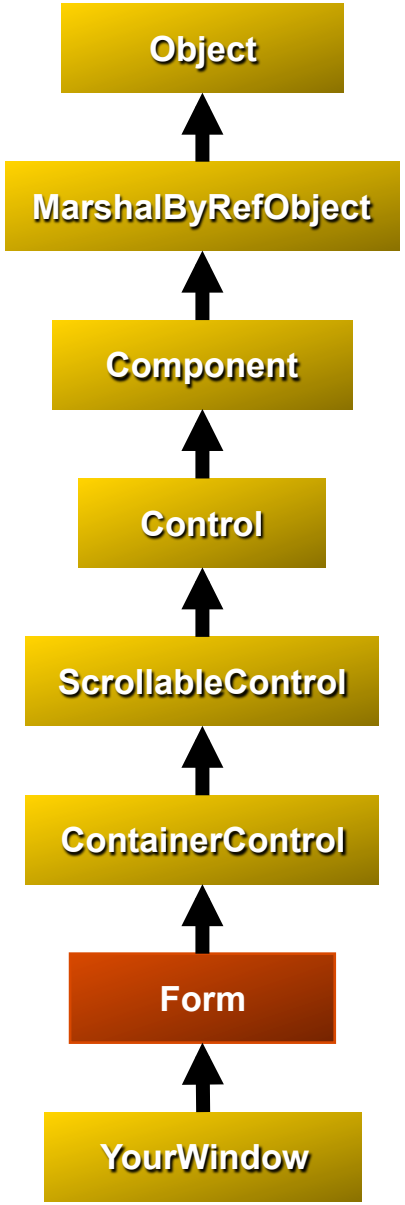
Note que `ContainerControl` já herda as propriedades `TabStop` e `TabIndex` de `Control`



Hierarquia – `Form`

Classe base de `Forms` específicos.

Além do grande conjunto de membros herdados das classes *Control, ScrollableControl* e *ContainerControl*, a classe *Form* não acrescenta grande funcionalidade:



Propriedades do formulário

Propriedades do formulário normalmente controlam o aspecto visual:

- `AutoScroll`
- `BackgroundImage`
- `ControlBox`
- `FormBorderStyle` (sizable?)
- `Icon`
- `Location`
- `Size`
- `StartPosition`
- `Text` (i.e. window's caption)
- `WindowState` (minimized, maximized, normal)

```
Form1  form;
form = new Form1();
form.WindowState = FormWindowState.Maximized;
form.Show();
```

Propriedades de `Form`

Form Properties	*Description*
`AcceptButton`	*Gets or sets the button on the form that is clicked when the user presses the ENTER key.*
`ActiveMDIChil, IsMDIChild IsMDIContainer`	*Each of these properties is used within the context of an MDI application.*
`AutoScale`	*Gets or sets a value indicating whether the form will adjust its size to fit the height of the font used on the form and scale its controls.*
`BorderStyle`	*Gets or sets the border style of the forma. Used in conjunction with the FormBorderStyle enumeration.*
`CancelButton`	*Gets or sets the button control that is used to be clicked when the user presses the ESC key.*
`ControlBox`	*Gets or sets a value indicating whether the form has a control box.*
`Menu, MergedMenu`	*Gets or sets the (merged) menu for the Form.*
`MaximizeBox, MinimizeBox`	*Used to determine if this form enables the maximize and minimize boxes.*
`ShowInTaskBar`	*Should this form be seen on the Windows taskbar*
`StartPosition`	*Gets or sets the starting position of the form at run time, as specified by the FormStartPosition enumeration.*
`WindowState`	*Configures how the Form is to be displayed on startup. Used in conjunction with the FormWindowState enumeration.*

Métodos de formulário

Ações que você pode realizar em um formulário:

- `Activate`: give this form the focus
- `Close`: close & release associated resources
- `Hide`: hide, but retain resources to show form later
- `Refresh`: redraw
- `Show`: make form visible on the screen, & activate
- `ShowDialog`: show modally

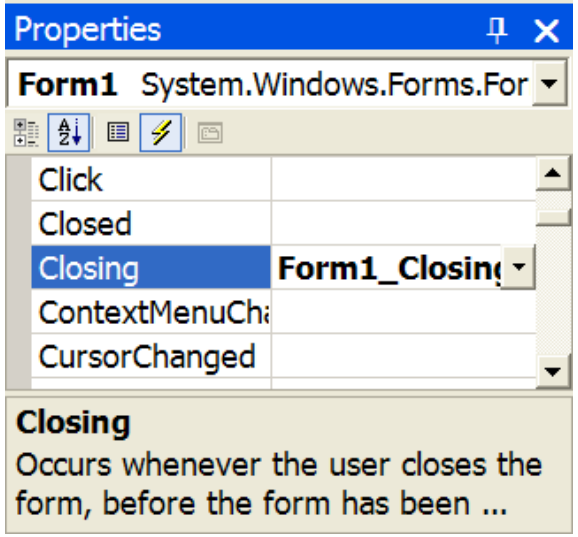
```
form.Hide();
  .
  .
  .
form.Show();
```

Eventos de formulário

você pode encontrar:

abrir a janela de propriedades,

clicar duas vezes no nome do evento



- `Load`: occurs just before form is shown for first time
- `Closing`: occurs as form is being closed (ability to cancel)
- `Closed`: occurs as form is definitely being closed
- `Resize`: occurs after user resizes form
- `Click`: occurs when user clicks on form's background
- `KeyPress`: occurs when form has focus & user presses key

Métodos e eventos de `Form`

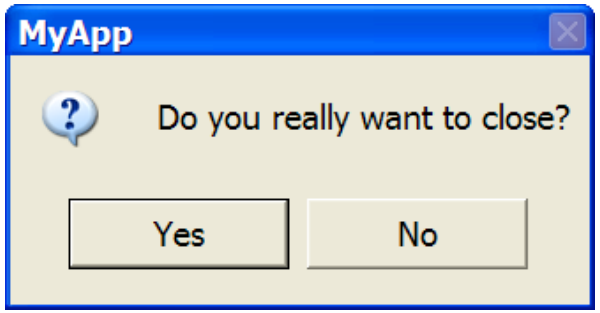
Estácio

Form Methods	*Description*
`Activate()`	*Activates the form and gives it focus.*
`Close()`	*Closes the form.*
`CenterToScreen()`	*Places the Form cantered on the screen*
`LayoutMDI()`	*Arranges the multiple document interface (MDI) child forms within the MDI parent form.*
`ShowDialog()`	*Shows the form as a modal dialog box.*

Form Events	*Description*
`Activate`	*Occurs when the form is activated in code or by the user.*
`Closed, Closing`	*Occurs when the form is about to close or has closed.*
`MDIChildActive`	*Occurs when a multiple document interface (MDI) child form is activated or closed within an MDI application.*

Exemplo

Perguntar ao usuário antes de fechar o form




```
private void Form1_Closing(object sender,
                           System.ComponentModel.CancelEventArgs e)
{
   DialogResult  r;

   r = MessageBox.Show("Do you really want to close?",
                       "MyApp",
                       MessageBoxButtons.YesNo,
                       MessageBoxIcon.Question,
                       MessageBoxDefaultButton.Button1);
   if (r == DialogResult.No)
     e.Cancel = true;
}
```

Convenções de nome

Estácio

Defina o nome do controle via propriedade "*Name*"

Um esquema comum de nomenclatura é baseado em prefixos:

- `cmdOK` refere-se ao controle "*button*"
- `lstNames` refere-se ao controle "*listbox*"
- `txtFirstName` refere-se ao controle "*textbox*"

File Dialog

Estácio

A caixa de diálogo de arquivo permite que você navegue através de diretórios e carregar ou salvar arquivos

Esta é uma classe abstrata e herdada por:

```
OpenFileDialog
SaveFileDialog
```

File Dialog

- `InitialDirectory` – string representando o diretório inicial
- `Filter` – string identificando o tipo de arquivo que será exibido

Um conjunto de pares de nome para exibição e padrão separados por barras verticais

```
Windows Bitmap|*.bmp|JPEG|*.jpg|GIF|*.gif
```

File Dialog

- `FileName` – o nome do arquivo selecionado
- `ShowDialog` – um método para mostrar a caixa de diálogo e um bloco de comandos que será executado até Cancelar ou OK ser clicado

```
if (openDialog.ShowDialog() == DialogResult.OK) {
    Image img = Image.FromFile(openDialog.FileName);
    pictureBox1.Image = img;
}
```

Image Class

Estácio

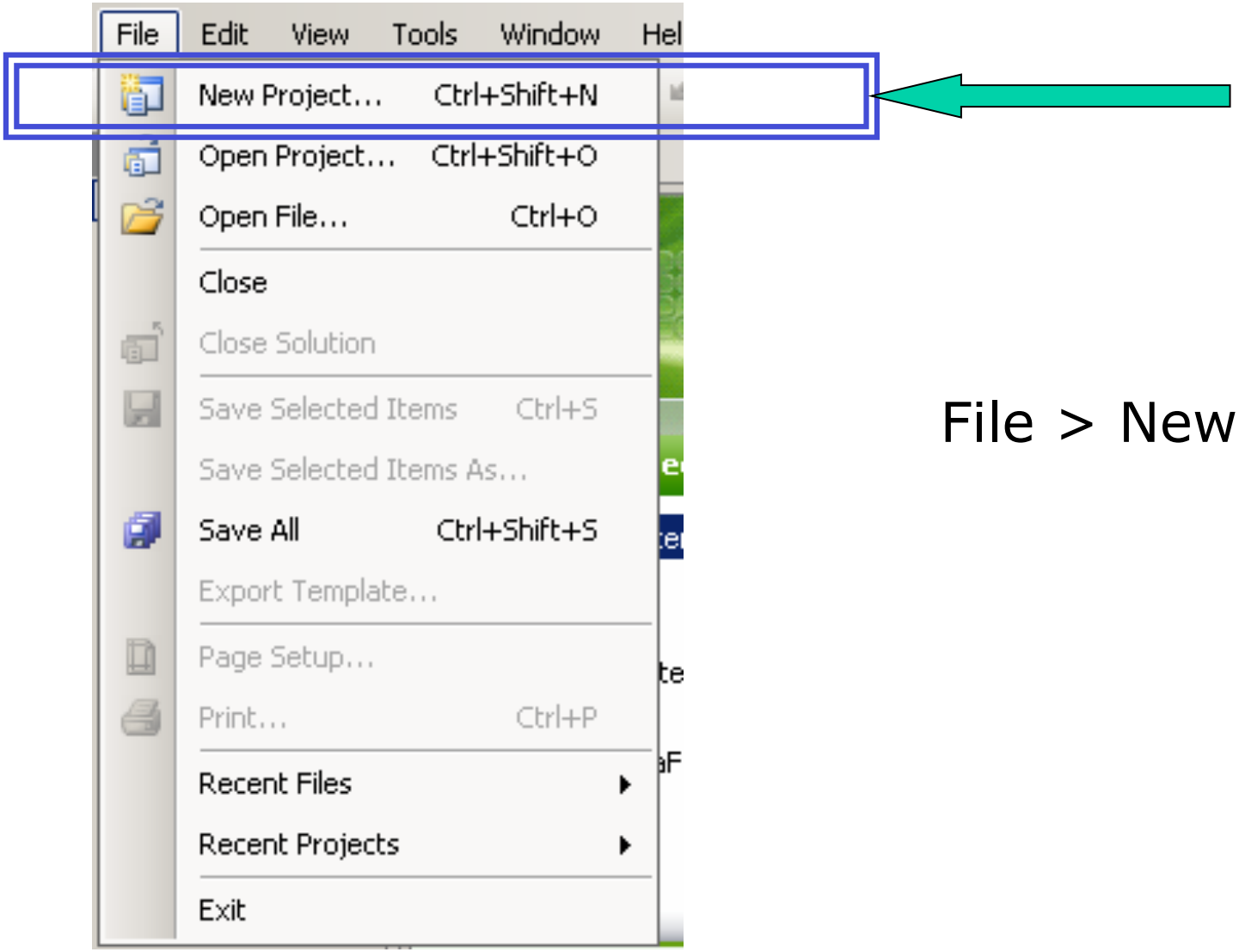
Uma classe abstrata que pode armazenar uma imagem

Várias classes são usadas para tipos de imagem, como BMP, GIF, JPG ou

- `FromFile(string fname)` – carrega um tipo de imagem suportada
- `FromStream(stream)` – carrega uma imagem de um fluxo (stream)
- `Height` – altura da imagem
- `Width` – largura da imagem

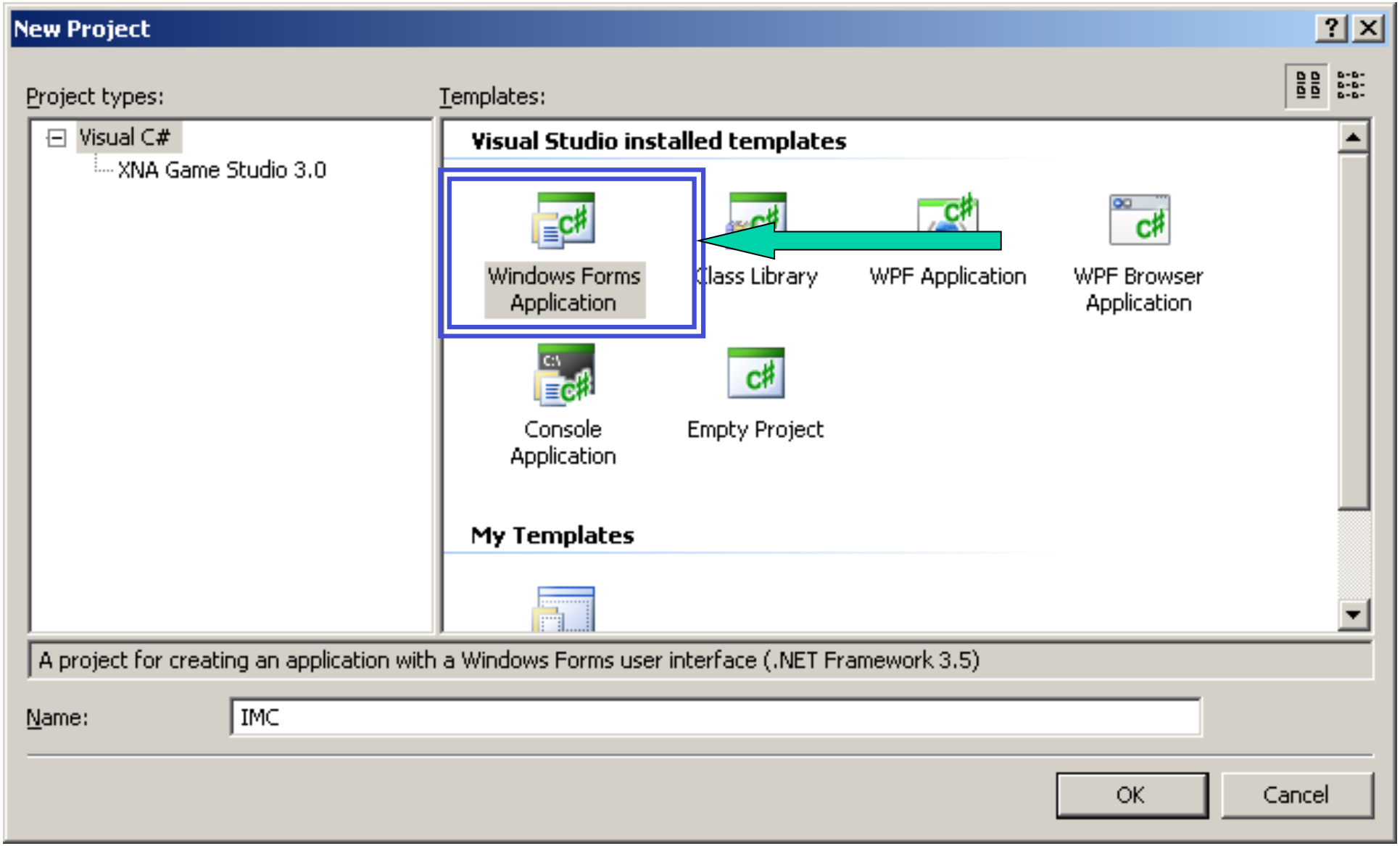
Primeiro projeto em Windows Forms
Conhecendo o ambiente

Criar um Projeto

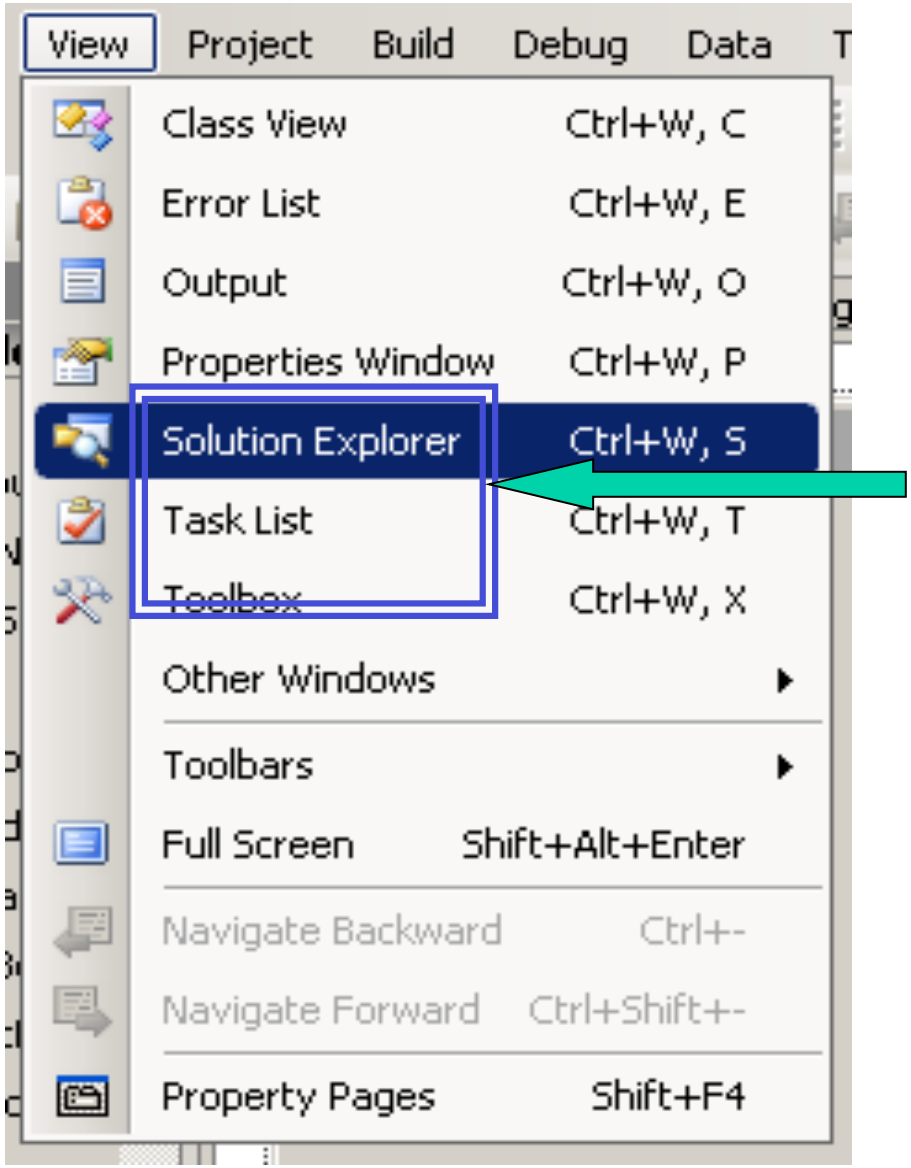


File > New Project

Escolher o tipo de projeto

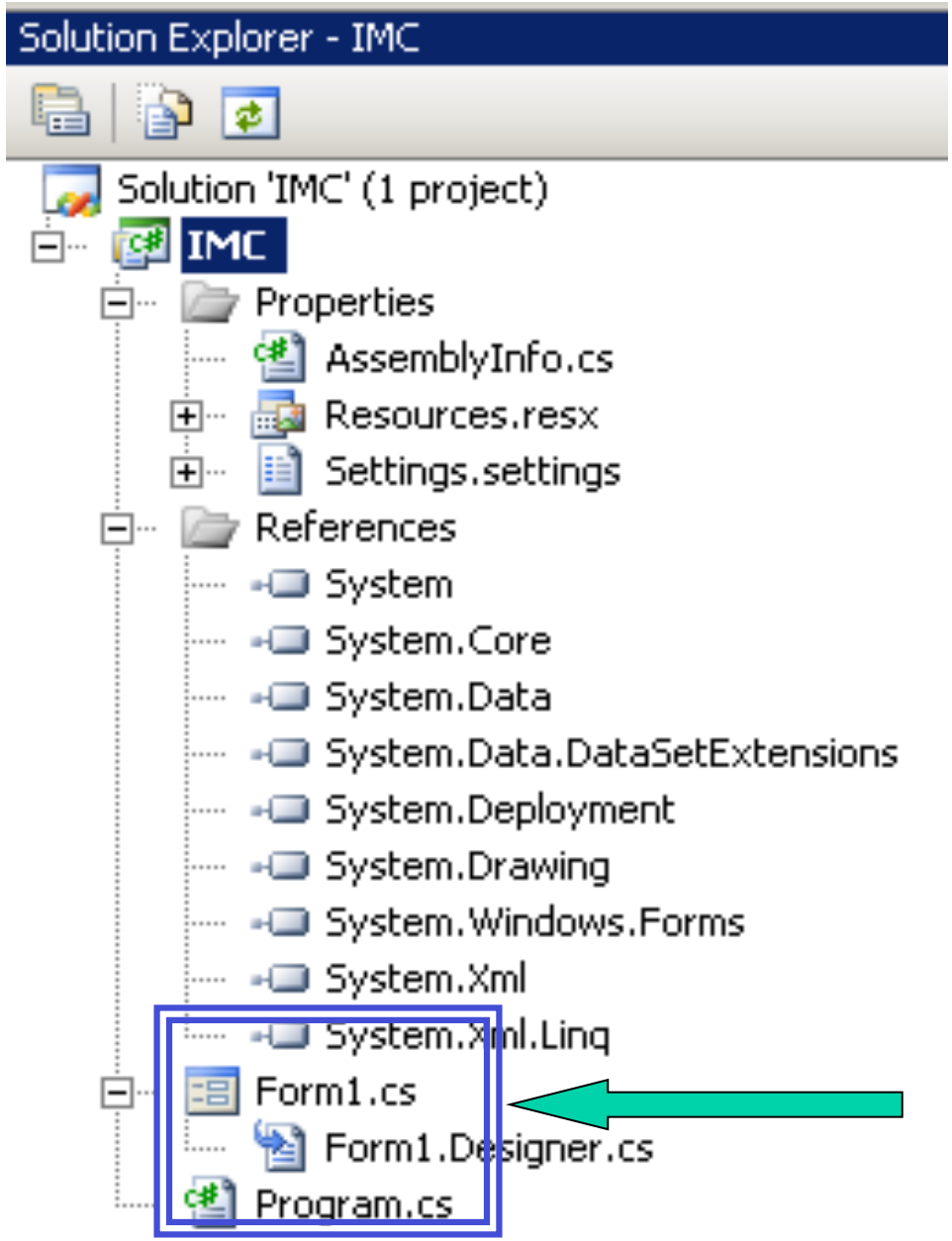


Visualizar o esqueleto da solução



Solução = Conjunto de Projetos

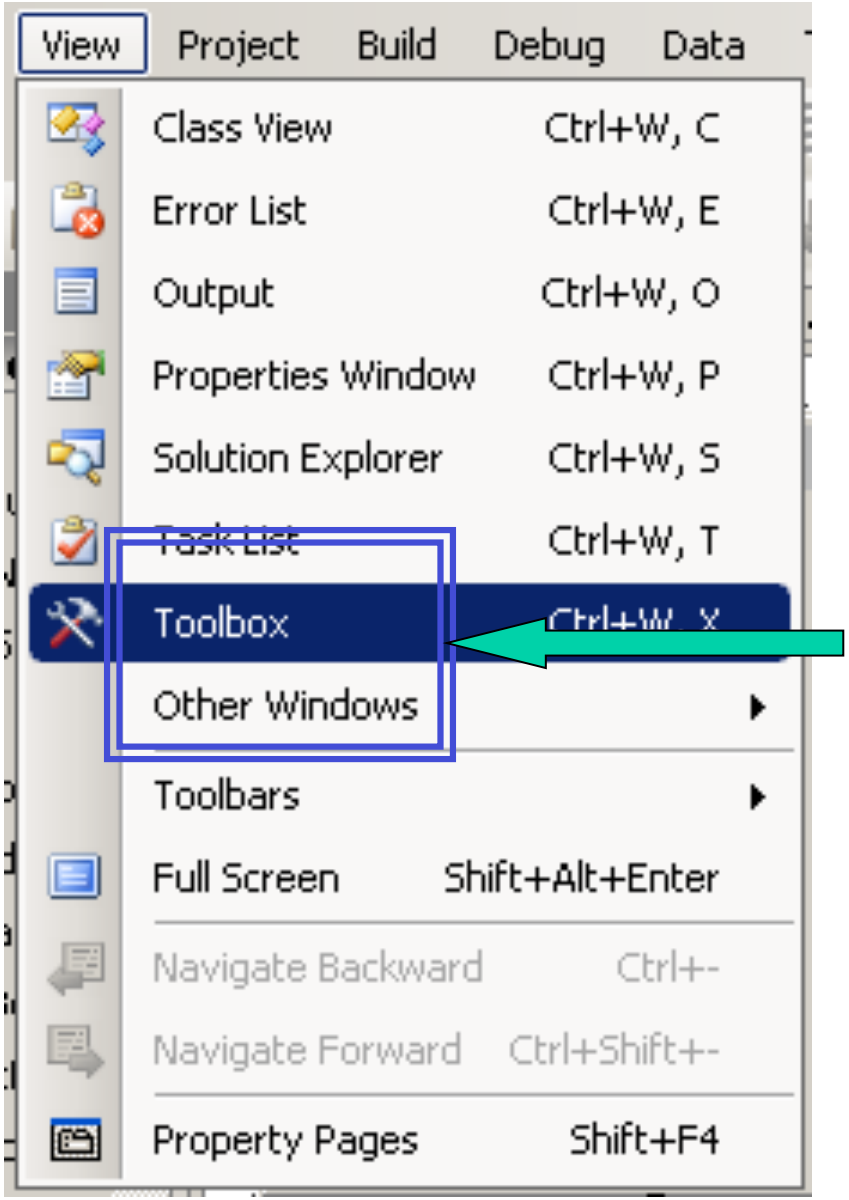
Esqueleto de solução



Formulário:

- ✓ Form1.cs: **comportamento**;
- ✓ Form1.Designer.cs: **interface**.

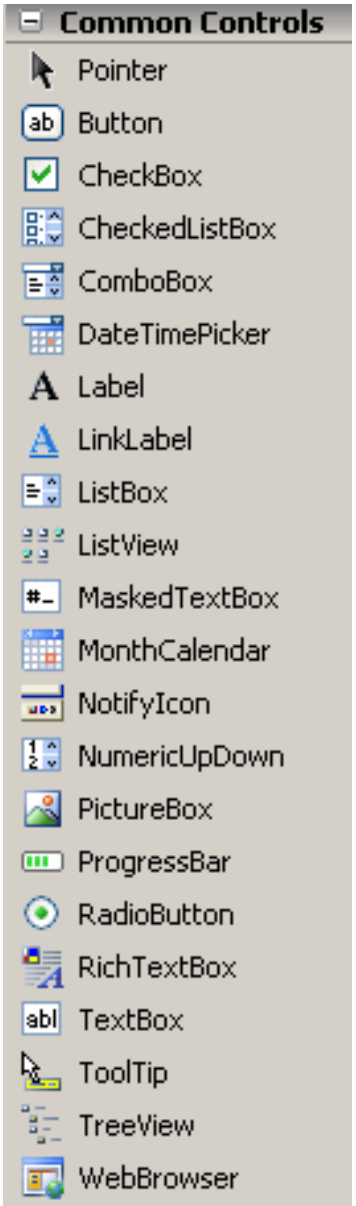
Exibir a barra de controles



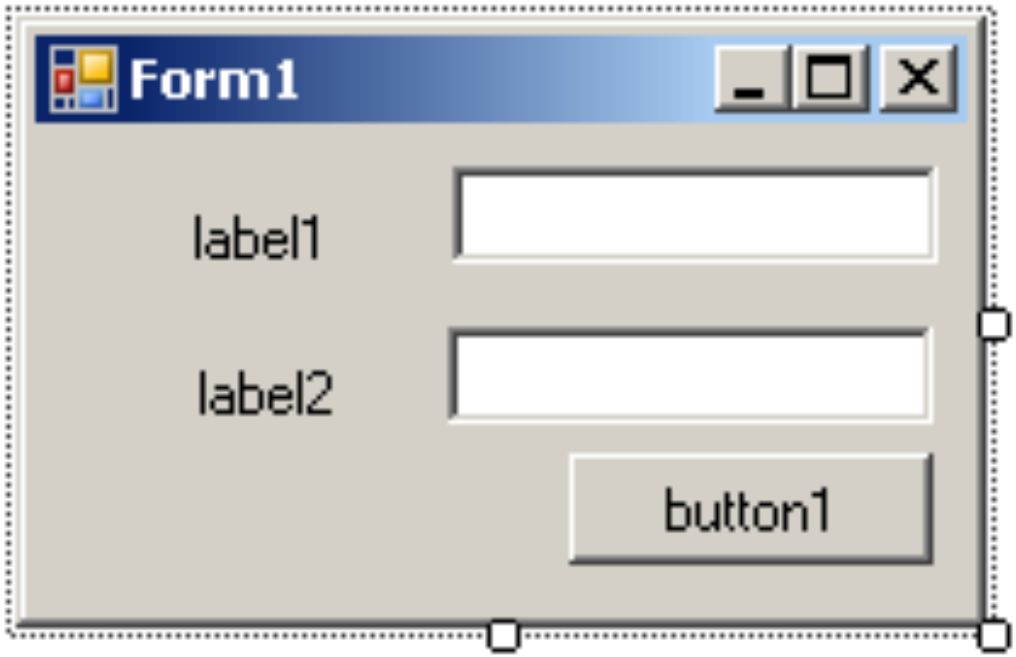
Exibir a barra de ferramentas em:

View > Toolbox

Criar a interface



- ✓ Arrastar no formulário:
  - **2 Label;**
  - **1 Button;**
  - **2 TextBox**.

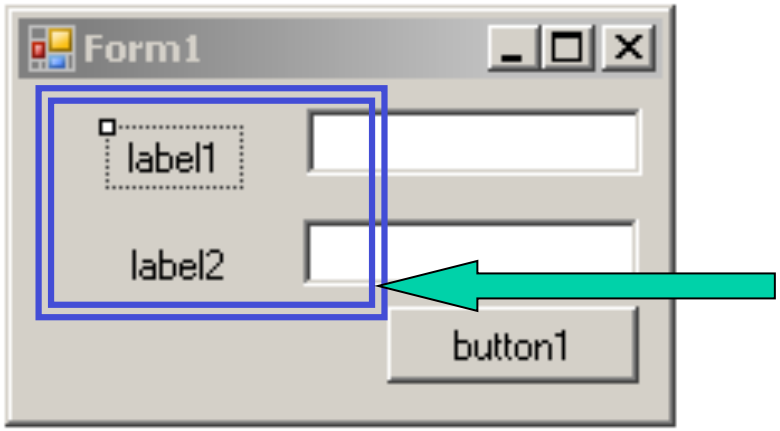


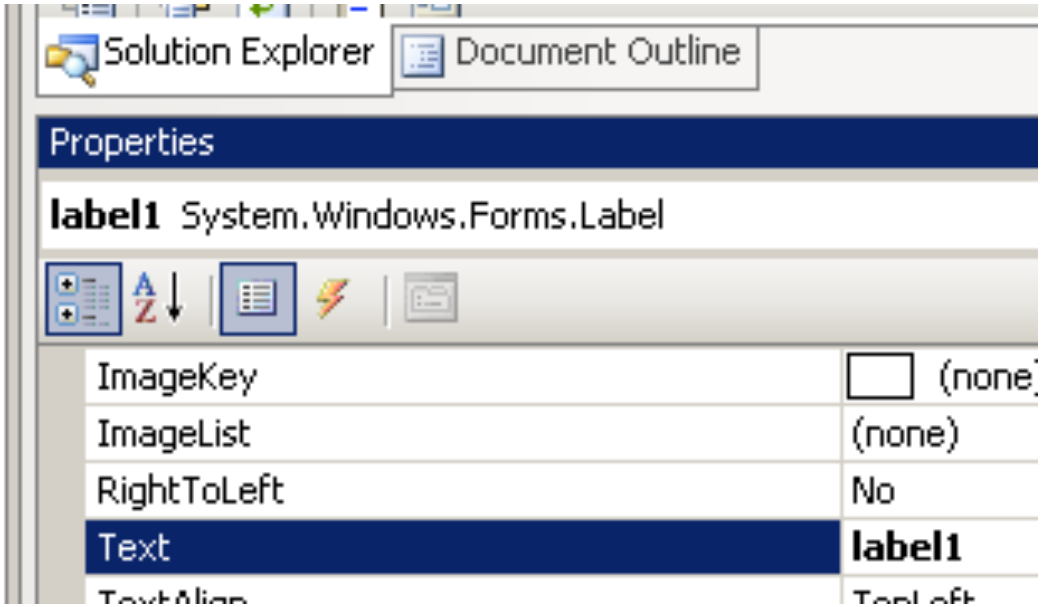
Alterar a interface

Alterar o valor de cada um dos rótulos;

Selecione o rótulo e pressionar F4 para exibir a **janela de propriedades**;

Altere a **propriedade Text**.





Alterar a interface

Estácio

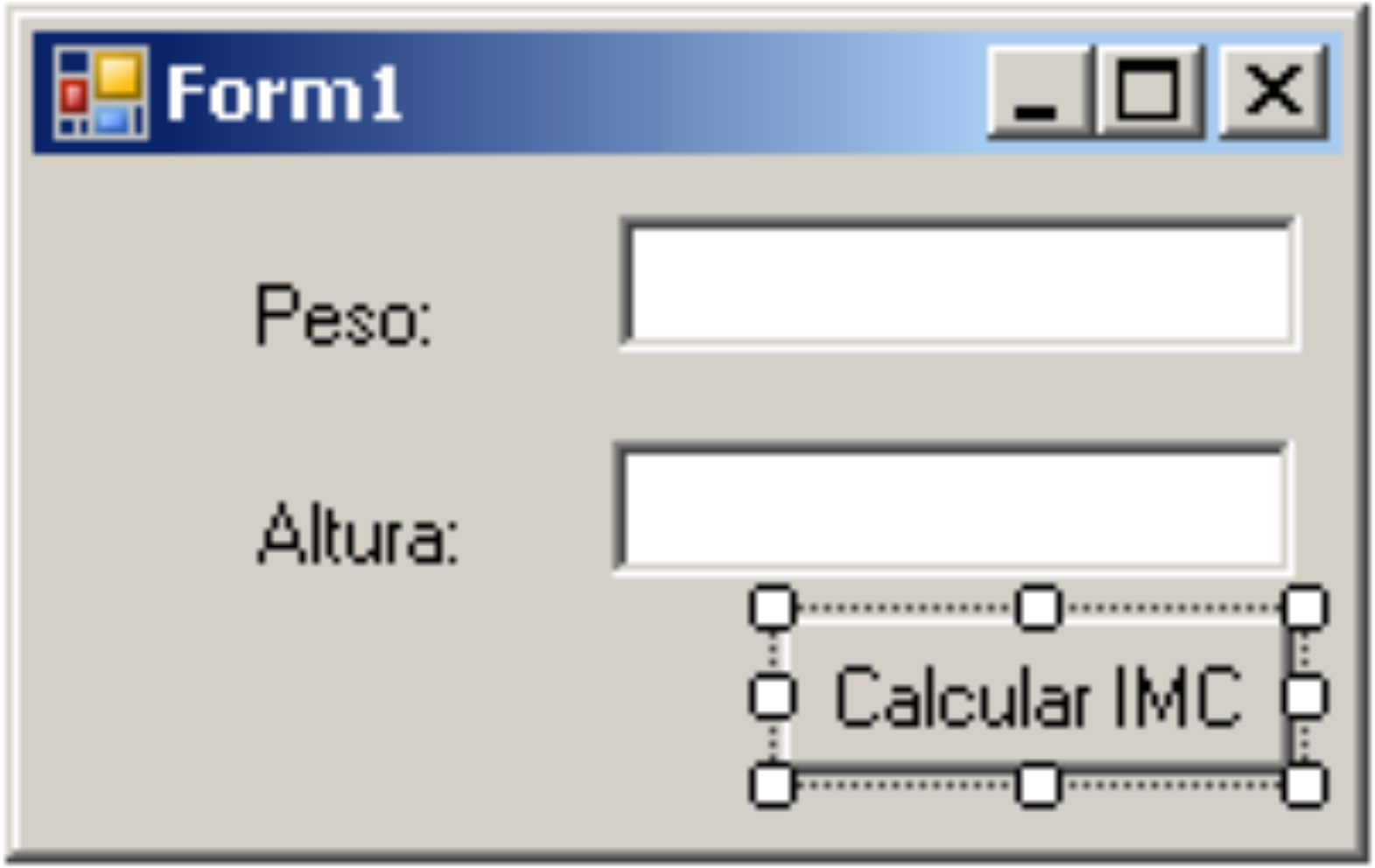
Propriedade Text: do Botão para "Calcular IMC";

Propriedade Text: de um Label para peso;

Propriedade Text: de outro para altura;

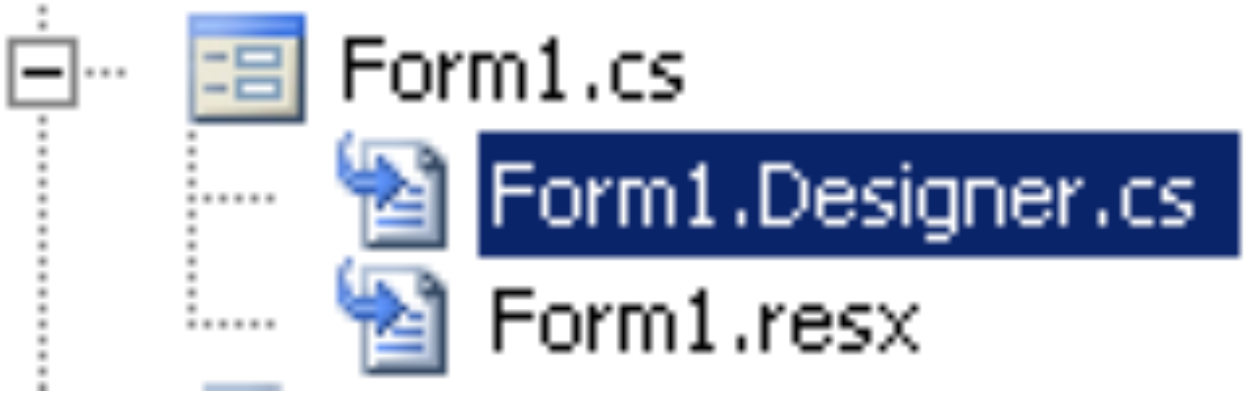
Propriedade (Name): de uma caixa de texto para textBoxPeso;

Propriedade (Name): de outra para textBoxAltura.


Form1
Peso:
Altura:
Calcular IMC

Onde está o código para criação desta interface?

Ver arquivo Form1.Designer.cs



Onde está o código para criação desta interface?

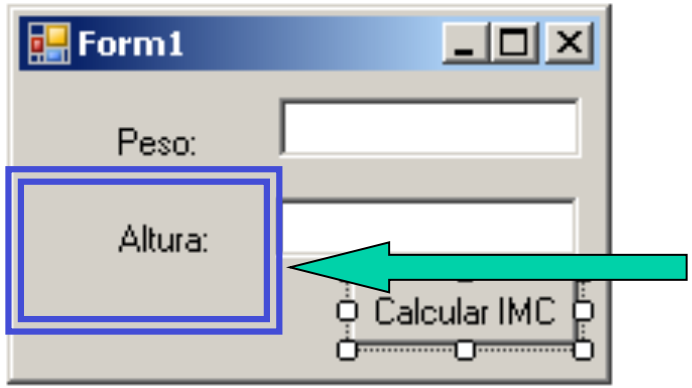
```
Windows Form Designer generated code

private System.Windows.Forms.Button button1;
private System.Windows.Forms.Label label1;
private System.Windows.Forms.Label label2;
private System.Windows.Forms.TextBox textBoxPeso;
private System.Windows.Forms.TextBox textBoxAltura;
```

Cada instância corresponde a um elemento na interface.

Onde está o código para criação desta interface?

```
// label2
//
this.label2.AutoSize = true;
this.label2.Location = new System.Drawing.Point(30, 48);
this.label2.Name = "label2";
this.label2.Size = new System.Drawing.Size(37, 13);
this.label2.TabIndex = 2;
this.label2.Text = "Altura:";
```



O código de preenchimento de cada valor de propriedade é gerado automaticamente.

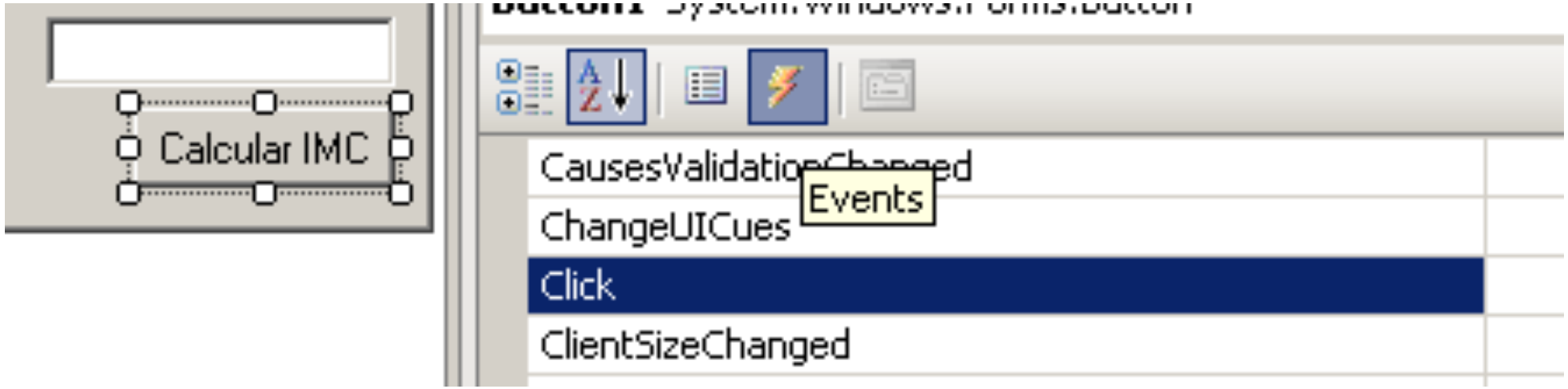
Associar o evento de clique ao botão

Selecionar o botão;

Pressionar F4 (**propriedades**);

Selecionar o “raio” (**lista de eventos**);

Clique 2 x na palavra Click.



Preencher o evento de clique de botão

Foi gerado o **esqueleto do código** que será chamado quando o botão for clicado.

```
public Form1()
{
    InitializeComponent();
}

private void button1_Click(object sender, EventArgs e)
{
    |
}
```

Preencher o evento de clique do botão

```
private void button1_Click(object sender, EventArgs e)
{
    double peso = Convert.ToDouble(textBoxPeso.Text);
    double altura = Convert.ToDouble(textBoxAltura.Text);
    double imc = peso / (altura * altura);
    MessageBox.Show("O seu imc é: " + Math.Round(imc,2) );
}
```

Form1

Peso: 90

Altura: 1,84

Calcular IMC

Ajuda

```
n.Collections.Gen
n.ComponentModel;
n.Data;
n.Drawing;
n.
n.
using System.

namespace IMC
{
```

O seu imc é: 26,58

OK