OO- Engenharia Eletrônica

Orientação a Objetos
-
Programação em C++

Slides 17: *Programação Visual ou Orientada a Objetos Gráficos (Formulários, Botões, Caixas de Texto etc) – Exemplificado em* ***Microsoft Visual C++ Express Edition****.*

Prof. Jean Marcelo SIMÃO
Aluno: Vagner Vengue

Programação Visual.

• Estudar os códigos de exemplo.

– Entender que objetos gráficos são relacionados com código ‘especial’ em C++.

– Entender que este código C++ ‘especial’ serve para tratar os eventos sobre objetos gráficos.

– Entender que, em um projeto correto, o código essencial do sistema é o mais independente possível do código relacionado a objetos gráficos.

• No ProjetoOOJanelaExemplo é tratada a classe Universidade e o seu Relacionamento para com a classe Departamento. De forma análoga, tratar as demais classes existentes nas versões precedentes do ‘sistema de universidade’ (Disciplina, Alunos etc).

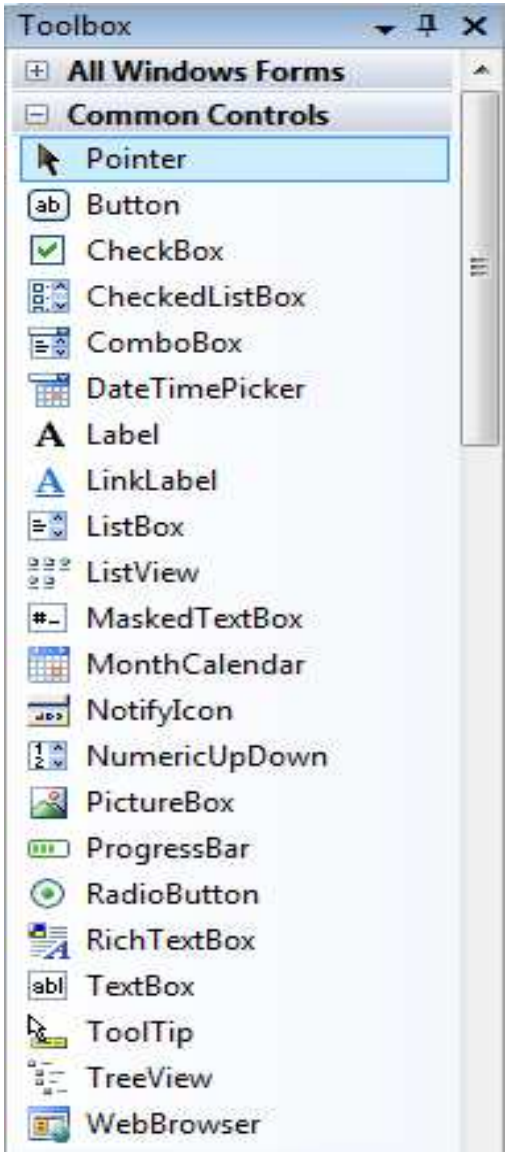
Microsoft Visual C++ .net

- Ambiente *Rapid Application Development* (*RAD*).

- Ambiente visual: *Button*, *TextBox*, *CheckBox*, *ComboCox*, *Label* etc...

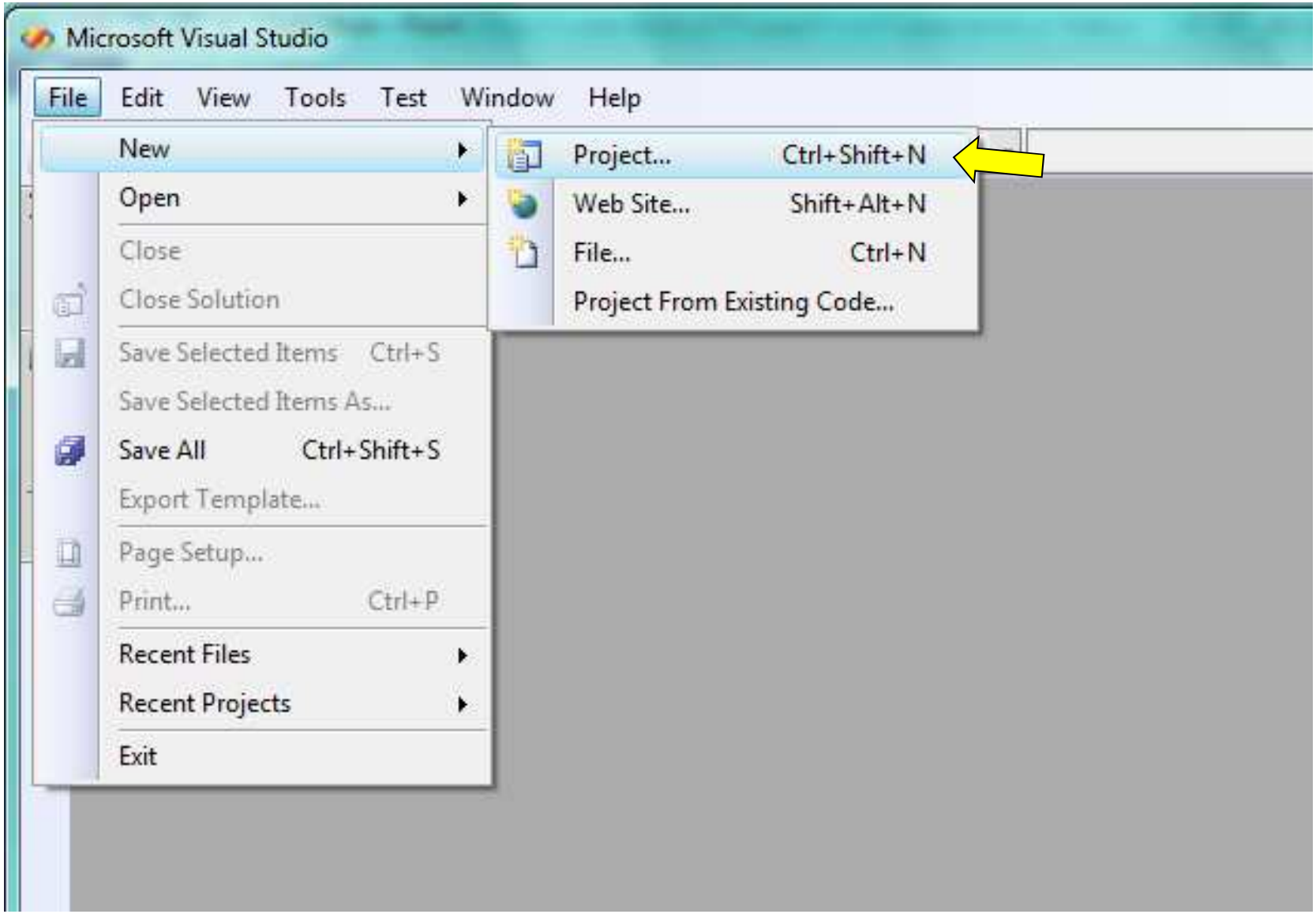
- Pode ser usado código gerenciado (Microsoft .NET Framework):
  - Tecnologia Baseada em Máquina Virtual.
  - *CLR* (*Common Language Runtime*), torna C++ uma linguagem semi-compilada.
  - Conjunto rico de bibliotecas.

-Pode ser usado código não gerenciado:
  - Usufrui-se do Visual C++ como gerenciador de projetos e compilador, porém, perde-se todo os recursos do .NET (inclusive gráficos).

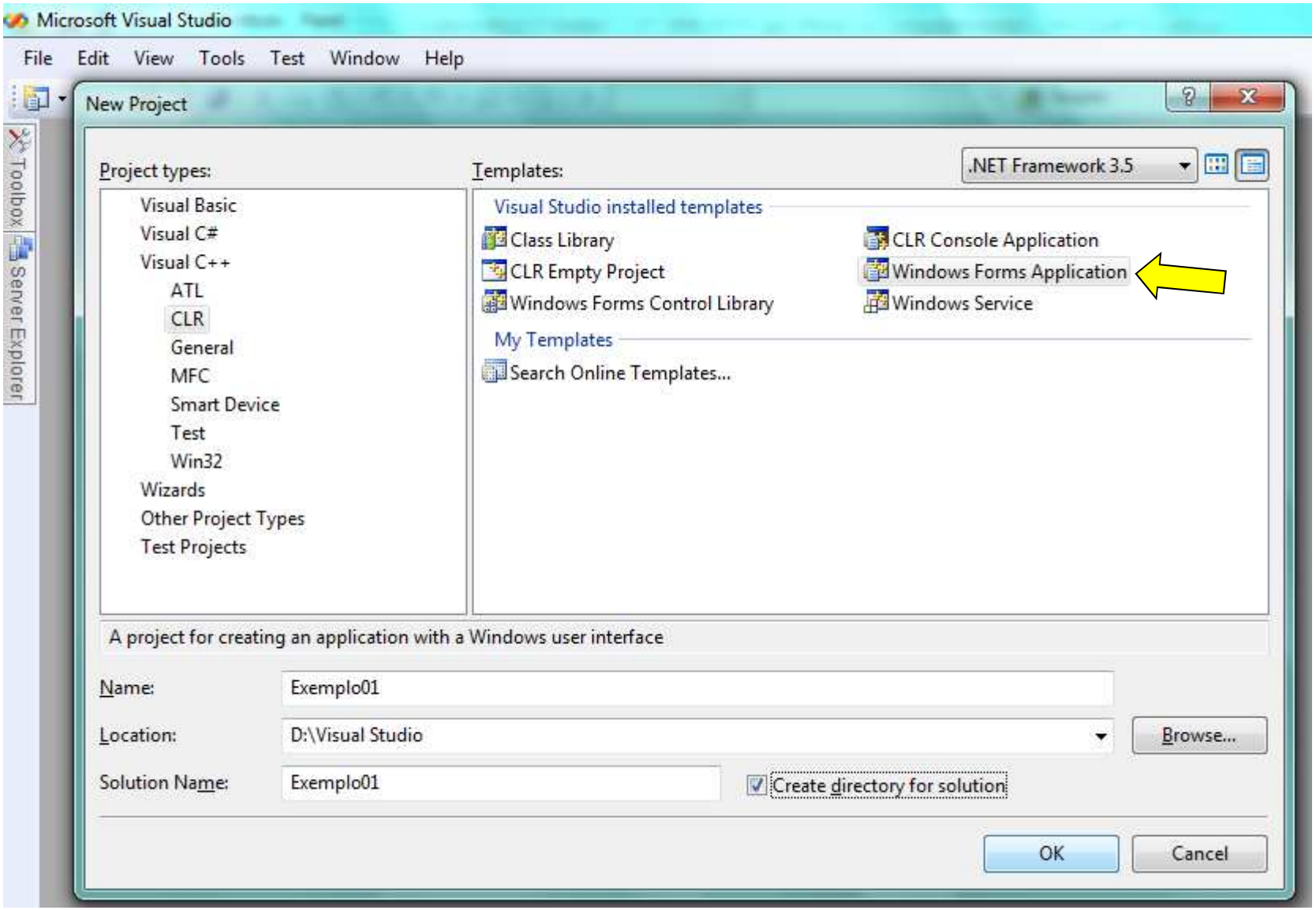


Exemplo de Introdução

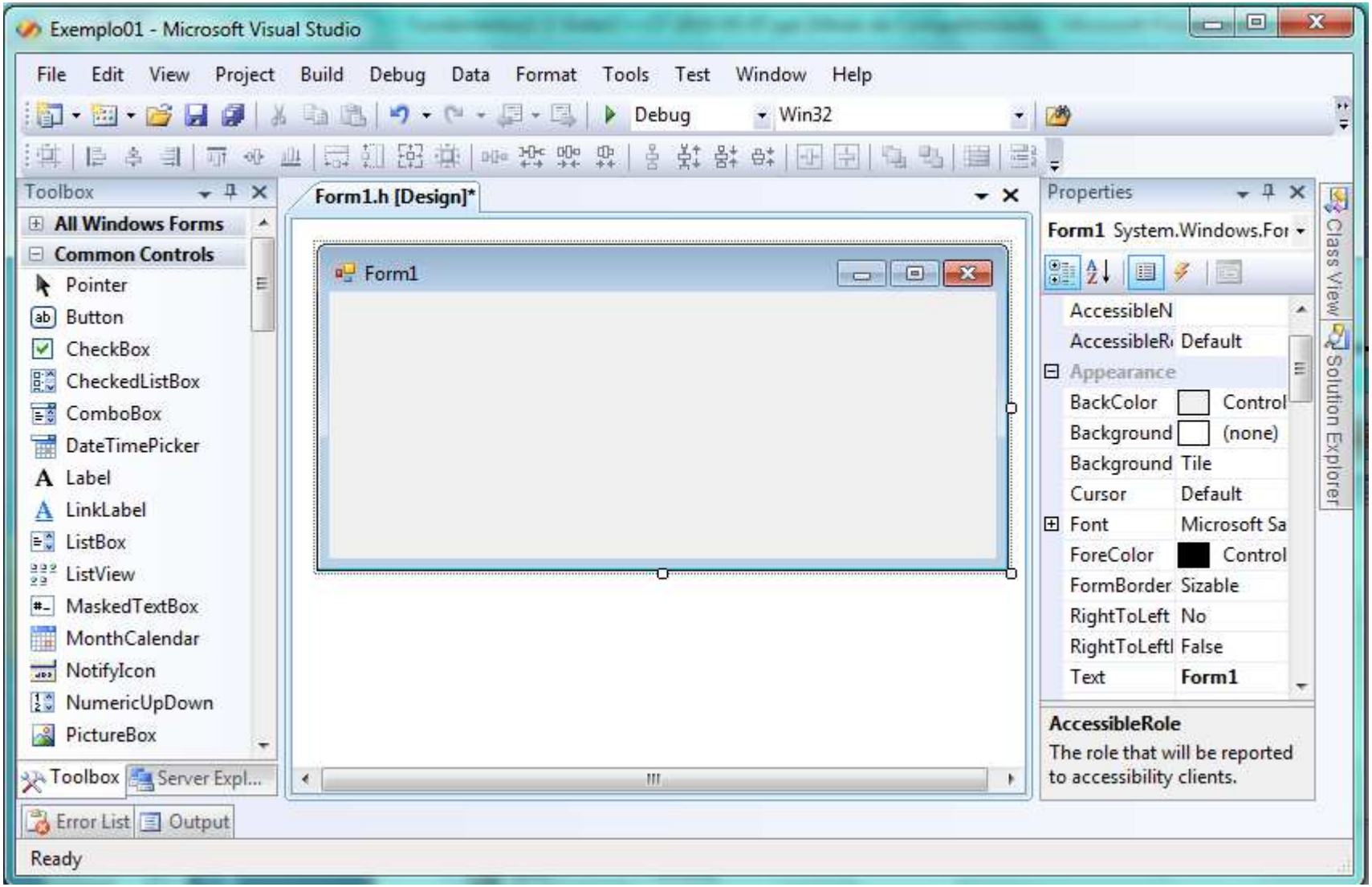
Microsoft Visual C++ .net



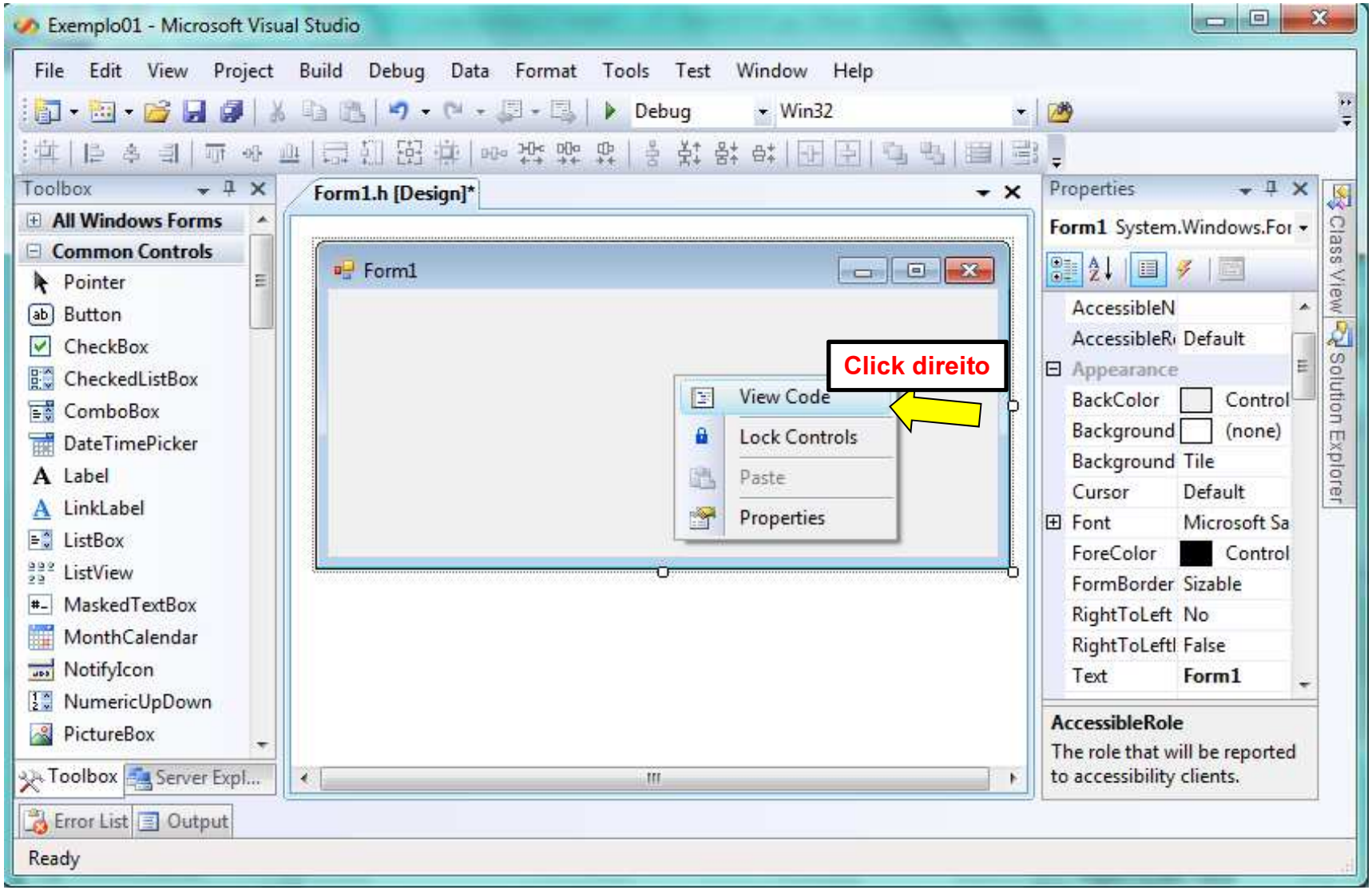
Microsoft Visual C++ .net



Microsoft Visual C++ .net



Microsoft Visual C++ .net



Microsoft Visual C++ .net

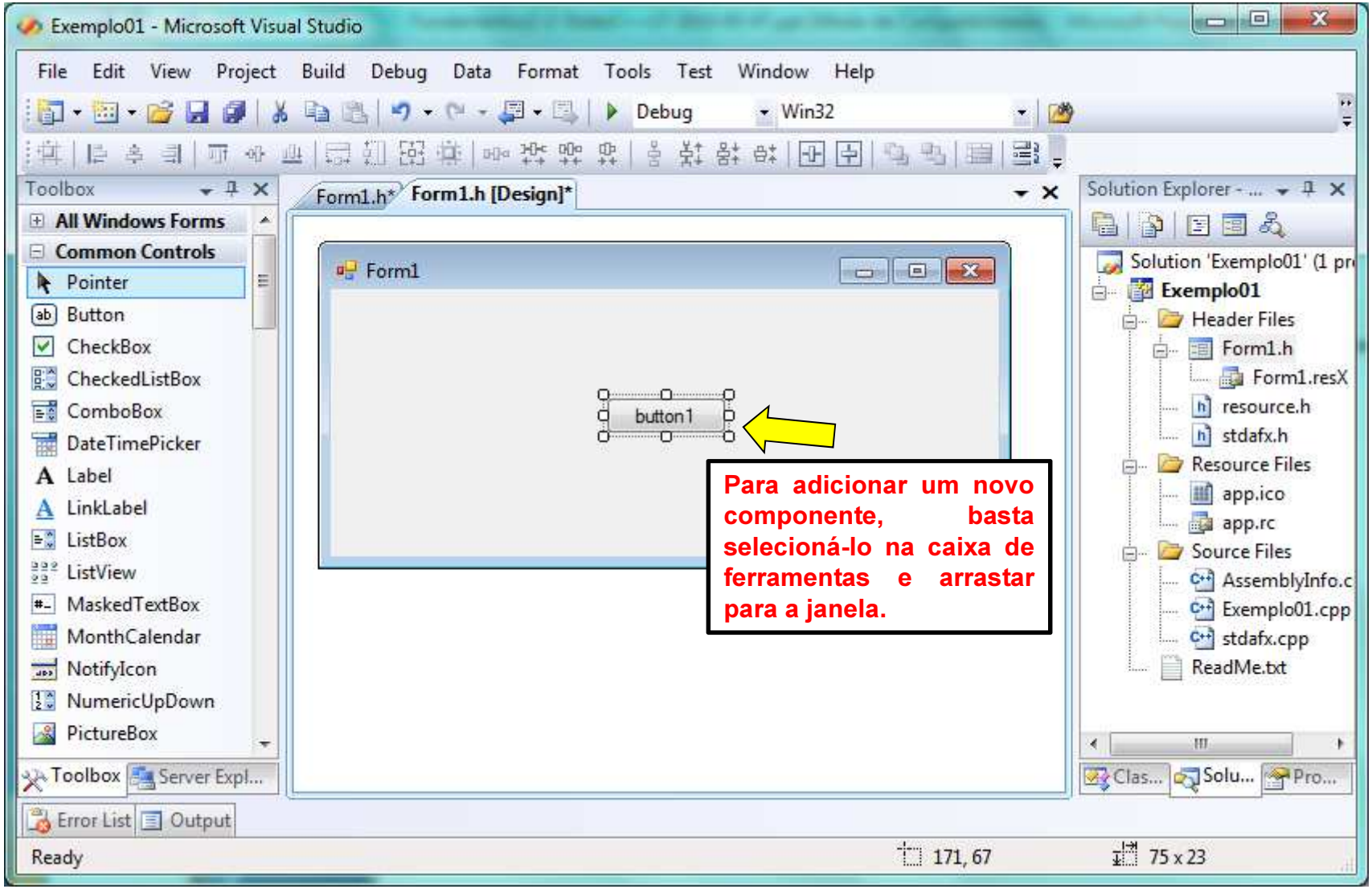
```
#pragma once
namespace Exemplo01 {
	using namespace System;
	using namespace System::ComponentModel;
	using namespace System::Collections;
	using namespace System::Windows::Forms;
	using namespace System::Data;
	using namespace System::Drawing;
	/// <summary>
	/// Summary for Form1
	/// </summary>
public ref class Form1 : public
			System::Windows::Forms::Form
	{
	public:
		Form1(void)
		{
			InitializeComponent();
	// TODO: Add the constructor code
	// here

		}
	protected:
	/// <summary>
	/// Clean up any resources being used.
	/// </summary>
		~Form1()
		{
			if (components)
			{
				delete components;
			}
		}
```

Código gerado automaticamente pelo Visual Studio

```
	private:
		/// <summary>
		/// Required designer variable.
		/// </summary>
		System::ComponentModel::Container ^components;
#pragma region Windows Form Designer generated code
		/// <summary>
		/// Required method for Designer support - do not modify
		/// the contents of this method with the code editor.
		/// </summary>
		 void InitializeComponent(void)
		 {
			this->SuspendLayout();
			//
			// Form1
			//
			this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
			this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
			this->ClientSize = System::Drawing::Size(488, 194);
			this->Name = L"Form1";
			this->Text = L"Form1";
			this->ResumeLayout(false);
		}
#pragma endregion
	};
}
```

Microsoft Visual C++ .net



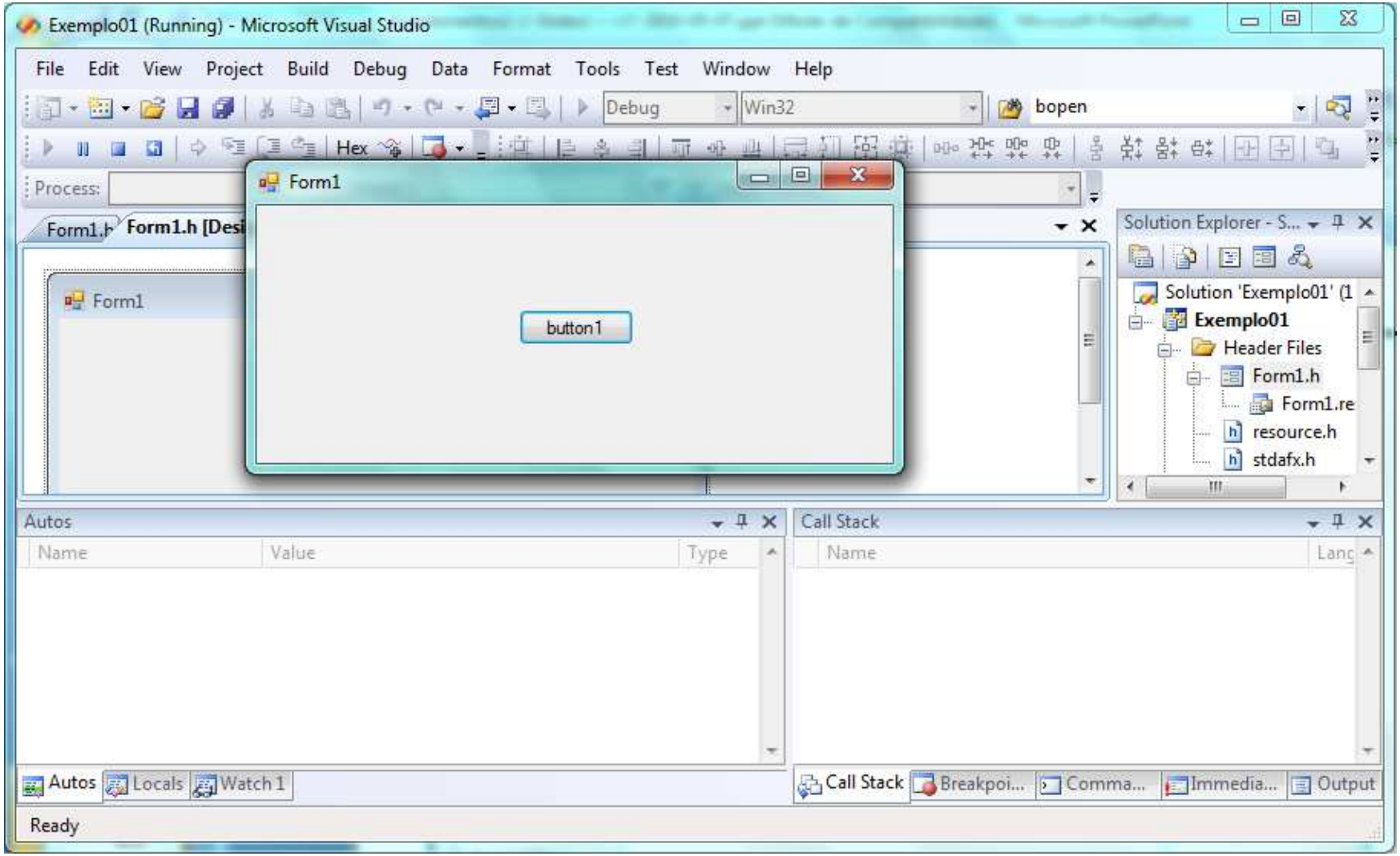
Microsoft Visual C++ .net

```
            //
            // button1
            //
            this->button1->Location = System::Drawing::Point(184, 127);
            this->button1->Name = L"button1";
            this->button1->Size = System::Drawing::Size(75, 23);
            this->button1->TabIndex = 0;
            this->button1->Text = L"button1";
            this->button1->UseVisualStyleBackColor = true;
            this->button1->Click += gcnew System::EventHandler(this, &Form1::button1_Click);
            //
            // Form1
            //
            this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
            this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
            this->ClientSize = System::Drawing::Size(488, 194);
            this->Controls->Add(this->button1);
            this->Name = L"Form1";
            this->Text = L"Form1";
            this->ResumeLayout(false);
        }
#pragma endregion
        private: System::Void button1_Click(System::Object^  sender, System::EventArgs^  e) {
            }
        };
}
```

Código adicionado automaticamente para o design do botão.

Código adicionado automaticamente para o evento do botão.

Microsoft Visual C++ .net

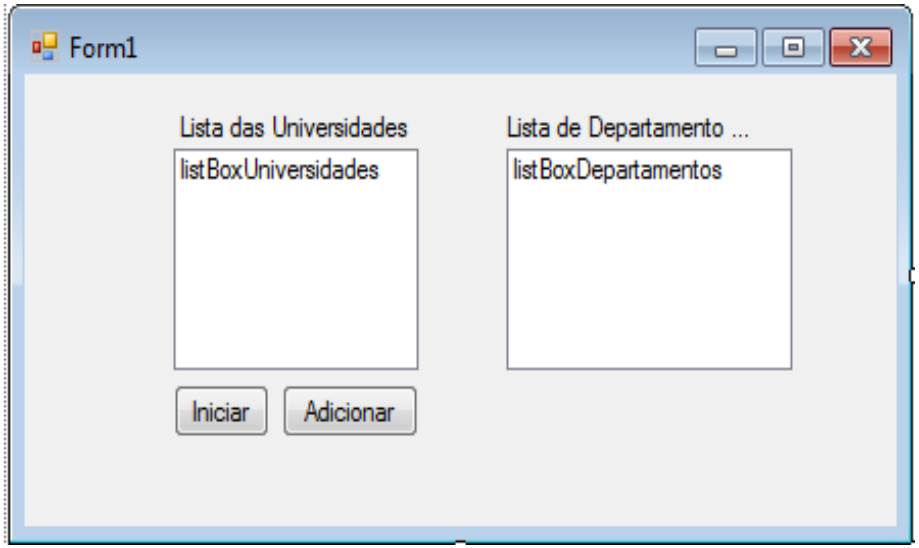


Para compilar e executar, pressione F5.

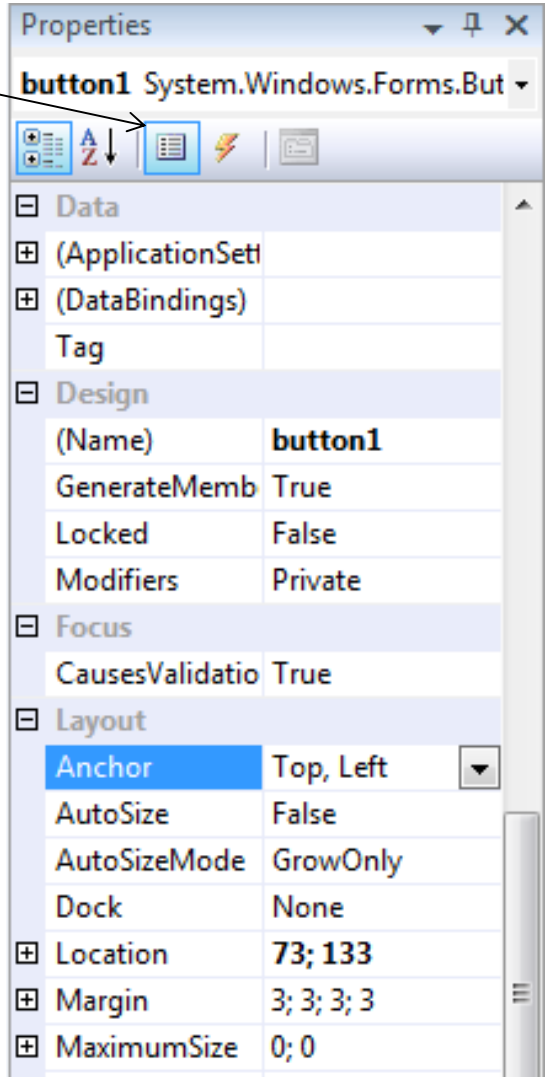
Microsoft Visual C++ .net

ProjetoOOJanelaExemplo

Propriedades



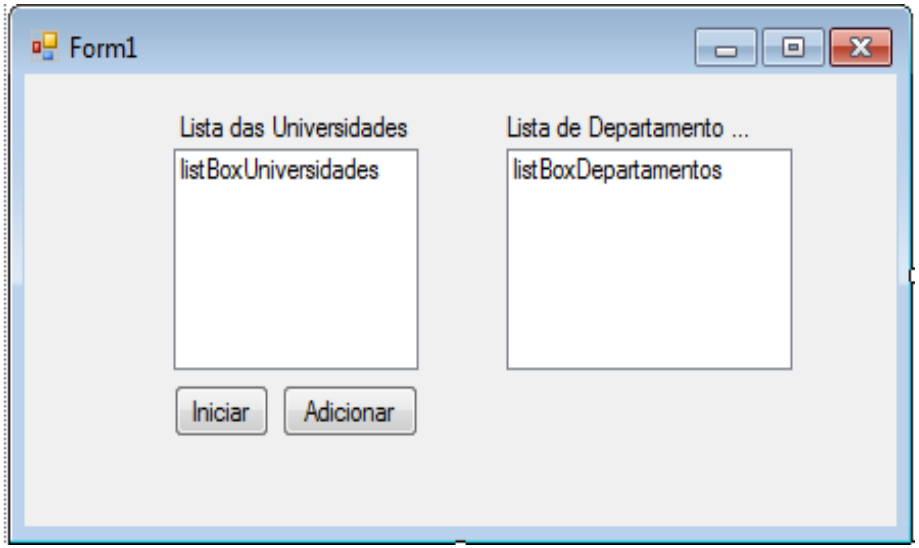
Propriedades

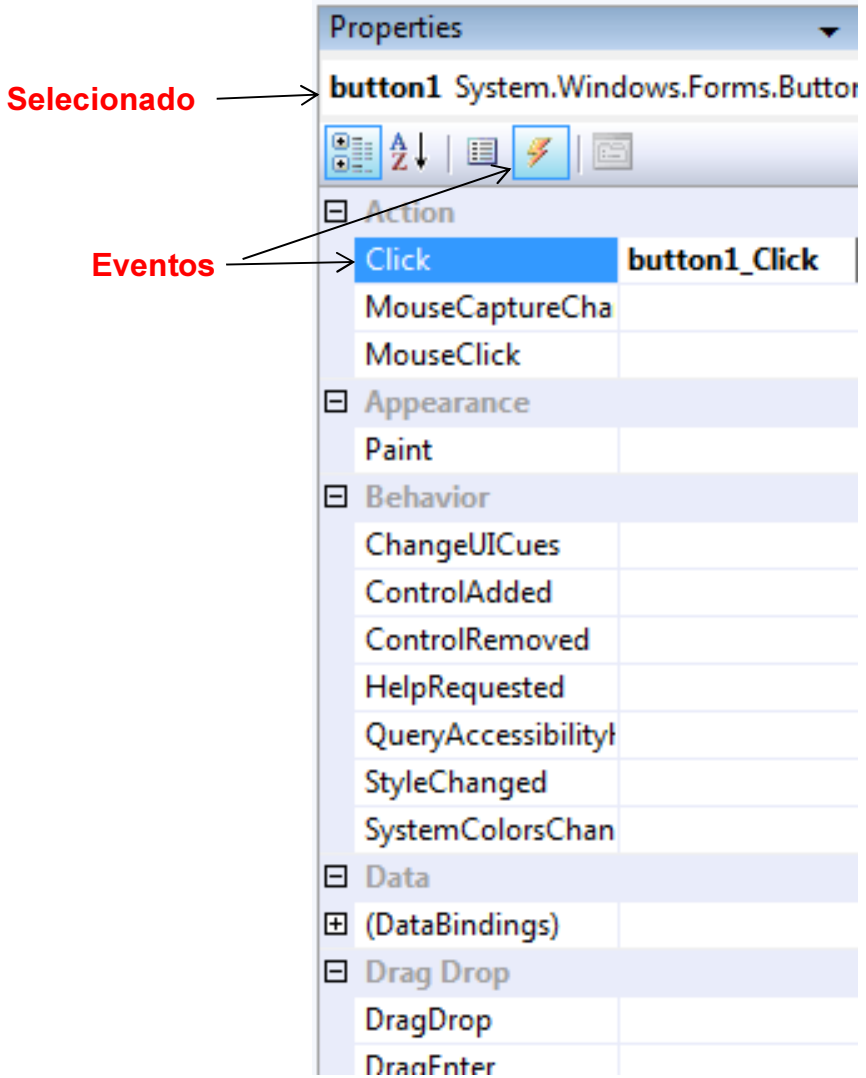


- Através da caixa de propriedades, pode-se definir as características de cada componente, tais como nome, cor, posição na tela, tamanho, imagem de fundo etc...

- É uma boa prática de programação sempre definir características básicas, como o nome, o título (para telas) ou texto (para botões).

Eventos

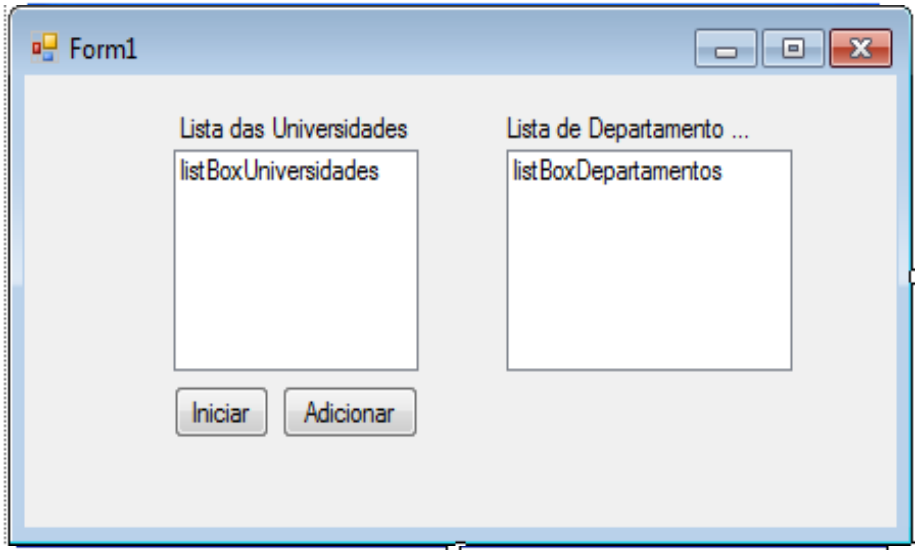




- É uma ação externa ao programa, por exemplo: um click de mouse, uma tecla pressionada, um movimento da tela, uma tela abrindo ou fechando.

- Com um duplo click sobre o componente é selecionado o evento padrão, onde pode-se inserir o código a ser executado quando o evento ocorrer.

Eventos



- Quando o evento é selecionado, é adicionado automaticamente à classe o método que trata o evento.

- Pode-se usar tanto as classes do .NET, quanto as classes padrões da API Win32.

- O objeto 'e' que é passado como parâmetro traz informações sobre o evento ocorrido e pode ser utilizado no código. Um exemplo é a posição (x, y) do mouse quando ele sobrepõe um componente.

Click de mouse do button1

```
private: System::Void button1_Click(System::Object^  sender, System::EventArgs^  e)
        {
                listBoxUniversidades->Items->Clear();
                objeto_principal->IteradorLUniversidades = objeto_principal->LUniversidades.begin();

                while (objeto_principal->IteradorLUniversidades!= objeto_principal->LUniversidades.end())
                {
                        string aux;
                        aux = (*(objeto_principal->IteradorLUniversidades))->getNome();

                        String^ NomeUniv = gcnew System::String(aux.data());
                        listBoxUniversidades->Items->Add(NomeUniv);
                        objeto_principal->IteradorLUniversidades++;
                }
        }
```

Agregação de Janelas (objetos)

```
public:
        Principal* objeto_principal;        // soh aceita ponteiros
        AdicionaUniversidade^ addUniv;

private: System::Void button2_Click(System::Object^  sender, System::EventArgs^  e)
        {
            if (!addUniv)
            {
                addUniv = gcnew AdicionaUniversidade();
                addUniv->setPrincipal(objeto_principal);

            }

            //addUniv->MdiParent = this;
            addUniv->limpaTextBox1();
            addUniv->Show();
            addUniv->BringToFront();
        }
```

Declaração de um ponteiro para a janela a ser chamada.

Verifica se a janela ainda não foi criada ou esta fechada e instancia o objeto (aloca memória).

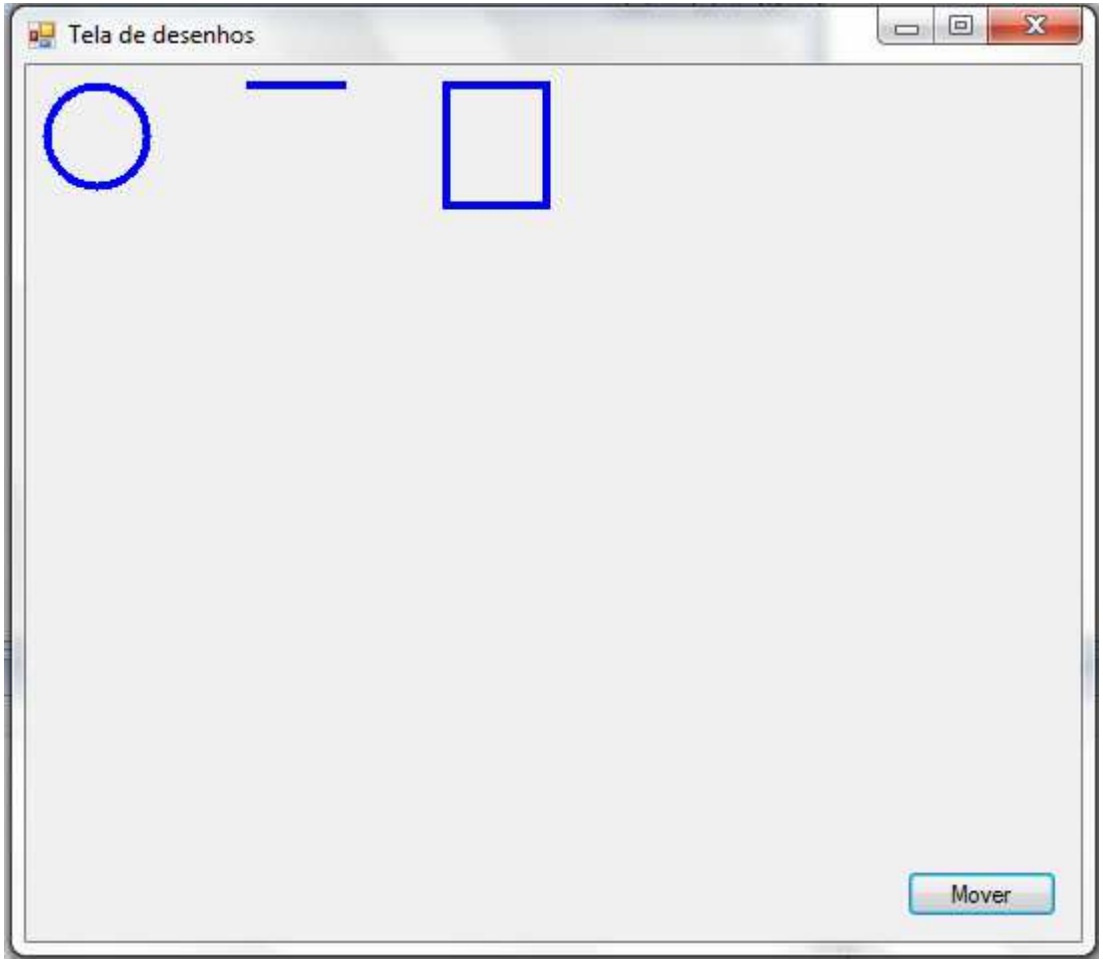
Mostra a janela.

Importante:

- Sempre que apagar um método de evento no código, lembrar de apagar o seu nome na caixa de propriedades.

Exemplo com figuras geométricas

Exemplo com desenhos



- Pode-se utilizar as classes do .NET para desenhar, pintar ou até mesmo carregar figuras na tela.

- Neste exemplo é mostrado uma forma simples de como redefinir o método padrão de pintura da janela.

Exemplo com desenhos

```
System::Void frm_Desenhos_Desenha(System::Object^ sender,
                                  System::Windows::Forms::PaintEventArgs^ e)
    {
        Graphics^ g = e->Graphics;
        Pen^ pen = gcnew Pen(Color::Blue, 4); // cria uma caneta;
        int x, y;

        x = 10;
        y = 10 + valor_aumentar;
        g->DrawEllipse(pen, x, y, 50, 50);      // desenha uma elipse;

        x += 100;
        //y += valor_aumentar;
        g->DrawLine(pen, x, y, x+50, y);        // desenha uma linha;
        x += 100;
        //y += valor_aumentar;
        g->DrawRectangle(pen, x, y, 50, 60);  // desenha um retângulo;
        delete pen;
    }
private: System::Void btn_Mover_Click(System::Object^  sender, System::EventArgs^  e) {
        // incrementa um valor para a posição y dos desenhos;
        valor_aumentar += 10;
        // repinta a tela;
        this->Refresh();
    }
```

- Este método utiliza o argumento de evento: e.

Exemplo com desenhos

```
//
// frm_Desenhos_Desenha
//
this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
this->ClientSize = System::Drawing::Size(527, 437);
this->Controls->Add(this->btn_Mover);
this->Name = L"frm_Desenhos_Desenha";
this->Text = L"Tela de desenhos";
this->ResumeLayout(false);

// acrescenta a função criada às funções padrões;
this->Paint += gcnew System::Windows::Forms::PaintEventHandler(this,
        &frm_Desenhos::frm_Desenhos_Desenha);
```

- Para controlar o modo de pintura da janela, precisa-se modificar um evento do programa, ou melhor, adicionar funcionalidades ao método que já existe para desenhar a janela.

- Conforme mostra o código acima, para o exemplo foi incrementado o método de evento Paint, usando-se um manipulador de eventos (forma padrão do .NET) para indicar ao evento existente qual a função a ser adicionada e o seu emissor (representada pelo ponteiro this).

- Para os métodos de evento gerados em modo design, esse código é gerado automaticamente, mas nada impede que se manipule eventos manualmente, como foi feito nesse exemplo.