

APOSTILA DE VISUAL STUDIO 2013

C#

V 1.0

**Hamilton Martins Viana
Paulo de Tarso P. Rezende
Cecilia Keiko Adati Tomomitsu**

Índice

Conteúdo

AULA 1	5
Visual Studio .Net	5
C#	5
O site ”TIOBE Programming Community Index”	5
O ambiente C#.....	6
Nomes de variáveis	12
Nomes de componentes (Controles).....	12
Palavras reservadas em C#	13
Tipos de variáveis.....	13
AULA 2	15
Projeto Calculadora	15
Tratador de evento “Click”	17
Tratador de evento “KeyPress”	18
Instrução de Seleção (IF).....	19
Uso do bloco “try / catch.....	20
Métodos	20
Exercício.....	21
Instrução de Seleção Switch.....	22
Propriedade KeyPreview	23
AULA 3	25
Assinatura do método	25
Métodos sem Retorno.....	25
Sobrecarga de método	25
Sobrecarga de operador	25
Exercício: Projeto Calculo de Área	26
Criação de uma classe	28
Componente tabControl	29
Declaração de Jump.....	31
Componente ListBox.....	31
Comandos de Iteração – Loops	32
AULA 4	34
Depurando uma aplicação	34
Pontos de Interrupção.....	34
Incluir Pontos de Interrupção	34

APOSTILA DE VISUAL C#

<i>Usando o recurso de depuração</i>	35
Excluir um ponto de interrupção.....	40
Acesso a Métodos de Outras Classes	42
Alinhamento de Componentes	43
Exercício: Programa Folha de Pagamento	44
AULA 5	47
Inicialização de Componentes - InitializeComponent().....	47
Tratador de evento “TextChanged”	49
Exercício1: Projeto Senha	50
Exercício2: Programa Editor	55
AULA 6	57
Exercício – Projeto Foguete	57
Comando de Iteração (For).....	58
Componente Timer.....	60
AULA 7	65
Criação e Utilização de Objetos	65
Objeto	65
Classes	65
Criação de uma Classe	65
Declaração de variáveis de uma classe.....	66
Declaração de Métodos de uma Classe	68
Utilização de Objetos	68
Exercício - “Uso de Objetos”	70
AULA 8	74
Herança.....	74
Exercício: Projeto Herança.....	75
AULA 9	80
Acesso a Banco de Dados utilizando assistente	80
Acesso a banco de dados por meio de componente e código:	93
AULA 10	101
Elaborando consultas ao banco de dados	101
Componente ErrorProvider.	103
Componentes DataGridView	106
Consultas – Uso do Query.....	107
Alteração de uma query.....	110
AULA 11	113
Exercício: projeto SCA	113
Bibliografia	120

APOSTILA DE VISUAL C#

AULA 1

Visual Studio .Net

É um ambiente de desenvolvimento integrado (IDE). Possui um conjunto de ferramentas de desenvolvimento para construção de aplicações Web ASP.NET, serviços Web XML, aplicações desktop e aplicativos móveis.

C#

- A linguagem **C#** faz parte do Visual Studio .Net.
- Tem todo o poder da linguagem C e possui um ambiente de desenvolvimento gráfico bastante eficiente.
- Há uma versão gratuita que pode ser baixada pela Internet (C# Express).
- Tem-se notado uma utilização crescente dessa linguagem por desenvolvedores, tornando-se um importante conhecimento que os profissionais de informática devem possuir.

O site "TIOBE Programming Community Index"

O site <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html> divulga mensalmente uma cotação da utilização das linguagens de programação, onde se pode notar a evolução das várias linguagens. Veja abaixo a avaliação da linguagem C# em julho de 2015:

TIOBE Index for July 2015

July Headline: Revival of C++

The TIOBE Programming Community index is an indicator of the popularity of programming languages. The index is updated once a month. The ratings are based on the number of skilled engineers world-wide, courses and third party vendors. Popular search engines such as Google, Bing, Yahoo!, Wikipedia, Amazon, YouTube and Baidu are used to calculate the ratings. It is important to note that the TIOBE index is not about the *best* programming language or the language in which *most lines of code* have been written.

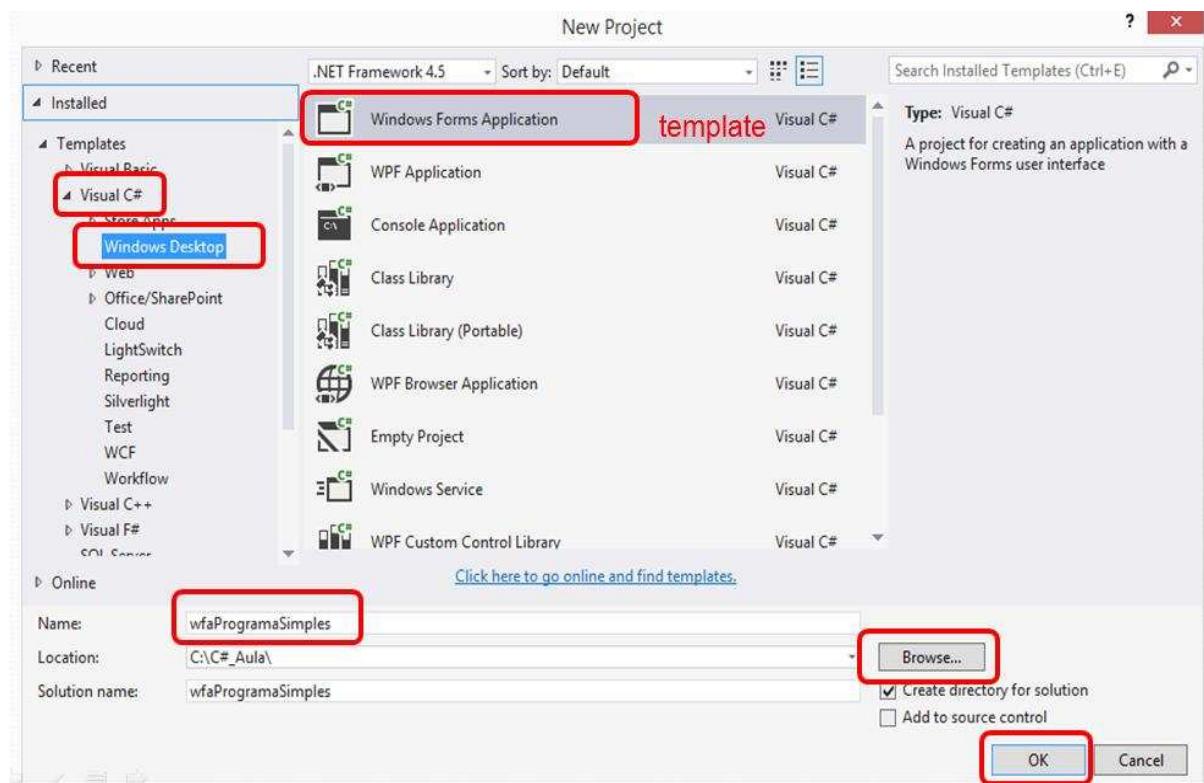
The index can be used to check whether your programming skills are still up to date or to make a strategic decision about what programming language should be adopted when starting to build a new software system. The definition of the TIOBE index can be found [here](#).

Jul 2015	Jul 2014	Change	Programming Language	Ratings	Change
1	2	▲	Java	17.728%	+2.04%
2	1	▼	C	16.147%	-1.00%
3	4	▲	C++	8.641%	+3.12%
4	6	▲	C#	5.652%	+1.60%
5	8	▲	Python	4.257%	+1.60%
6	3	▼	Objective-C	3.344%	-6.95%

O ambiente C#

Para iniciar nossos estudos de C#, vamos fazer um “Programa Simples”.

- Para isso, execute o programa Microsoft Visual Studio 2013 e,
- Inicie um novo projeto: **Arquivo / Novo / Projeto (File/New/Project)**.
- Na janela parecida com a figura abaixo, selecione “**Aplicativo de Windows Forms**”,
- Mude o nome (Nome) do projeto para “**WFAProgramaSimples**”,
- Em "Localização" digite o caminho onde quer gravar o projeto e dê um click no botão OK.

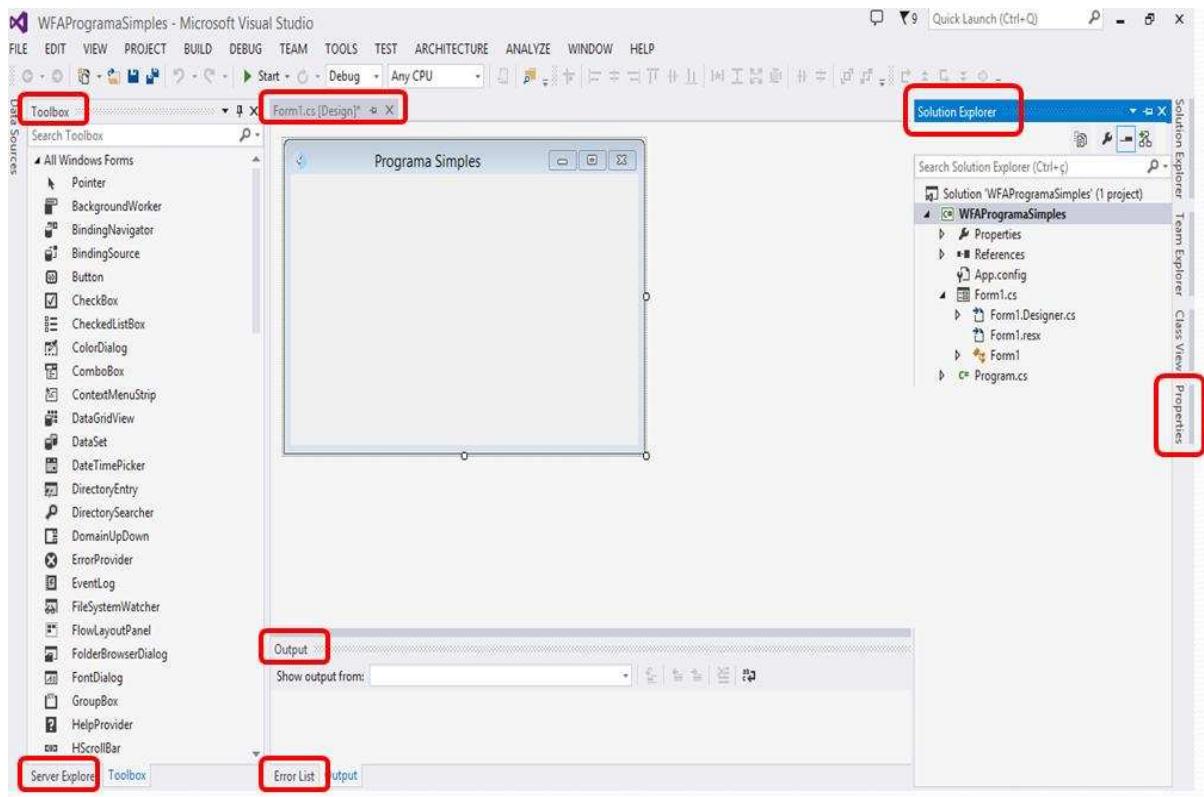


Obs:

1 - Dircione a gravação para seu pendrive e salve o projeto regularmente (Arquivo / Salvar Tudo).

2 - Nomes de arquivos não podem conter caracteres especiais, tais como: # ; : ? % etc. e também não podem ser palavras reservadas, tais como class, prn, etc.

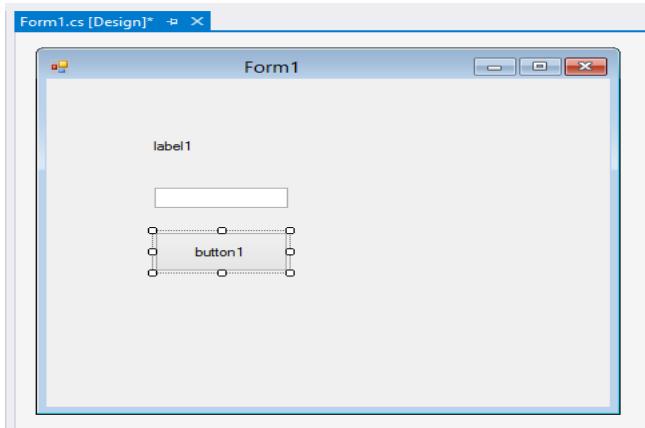
É aberto o ambiente de desenvolvimento de aplicações para Windows como na figura abaixo:



Configure o ambiente do C# igual ao da figura acima, click com o botão direito na barra de título de cada janela e selecione "flutuar". Dessa maneira, as janelas ficarão flutuantes e você as coloca onde quiser.

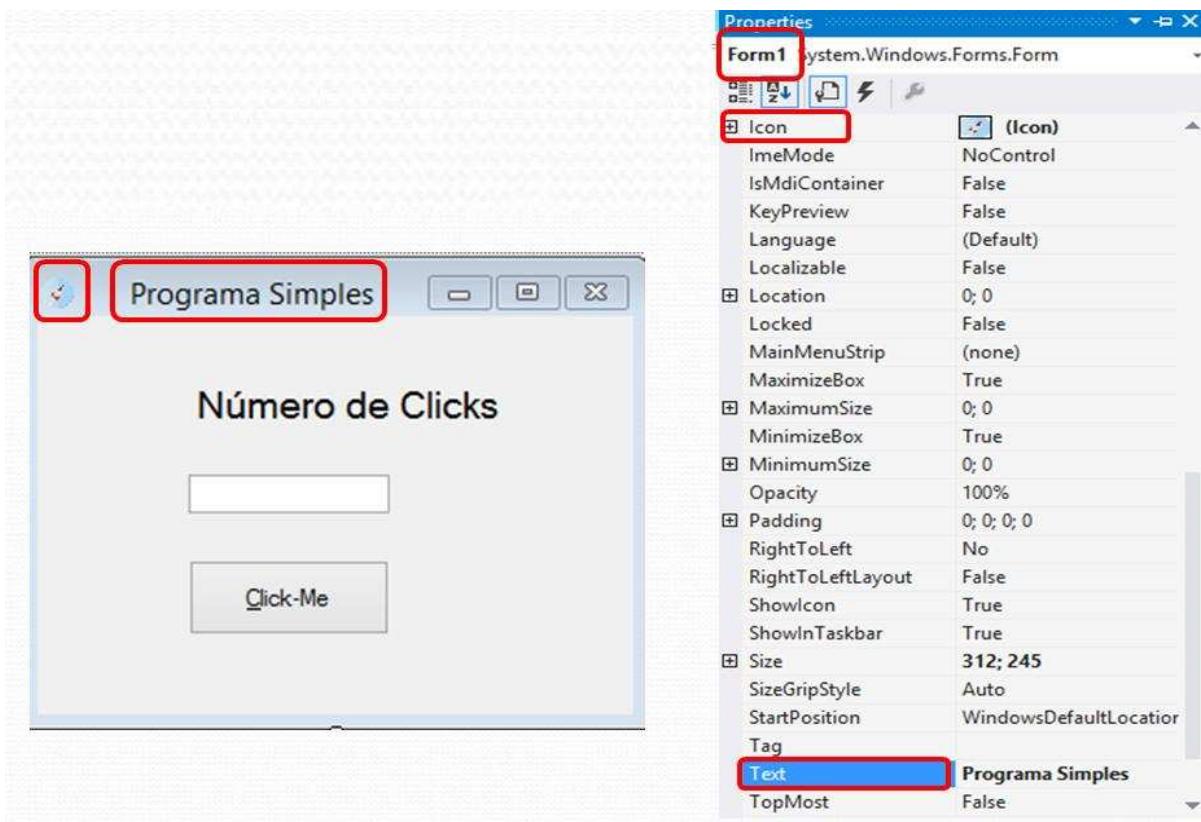
- Na janela “**Caixa de Ferramentas**” estão os ícones dos componentes que você poderá utilizar para elaborar a interface do programa, tais como caixas de texto, botões, objetos de acesso a banco de dados, etc.
- O “**Form1**” é o formulário no qual você irá colocar os componentes descritos acima.
- A janela “**Gerenciador de Soluções**” contém todos os itens pertencentes ao projeto.
- A janela “**Propriedades**” lista as propriedades do componente que está selecionado.

Vamos incluir no formulário alguns componentes de nossa **Caixa de Ferramentas**: um label (**Label**), uma caixa de texto (**TextBox**) e um botão de comando (**Button**), de acordo com a figura a seguir:



À medida que se inserem componentes no formulário, devemos ir configurando suas propriedades de acordo com as funcionalidades que deles desejamos.

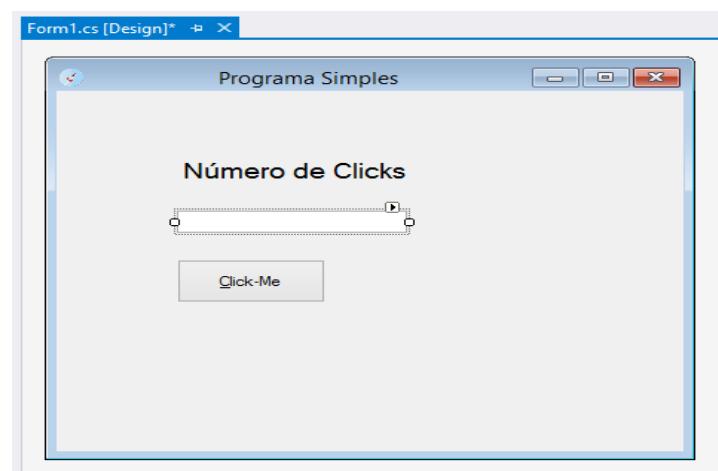
Por exemplo, para deixar a barra de título do formulário com “Programa Simples” selecionar o formulário e digitar esse texto na propriedade “Text”. Agora inclua um ícone no formulário – propriedade Icon.



Agora configure as propriedades dos outros componentes:

- Mude as propriedades do botão: Text para “&Click-me” e Name para “btnClickMe”.
- Mude a propriedade **Text** do label para “Número de Clicks”.
- Altere a propriedade “**AutoSize**” do label para “False”, e na propriedade “**Font**”, altere o “**Size**” para 14.
- Na propriedade “ **TextAlign**” do label, coloque “MiddleCenter”.
- Mude a propriedade **Name** da caixa de textos para “txtContaClicks”;

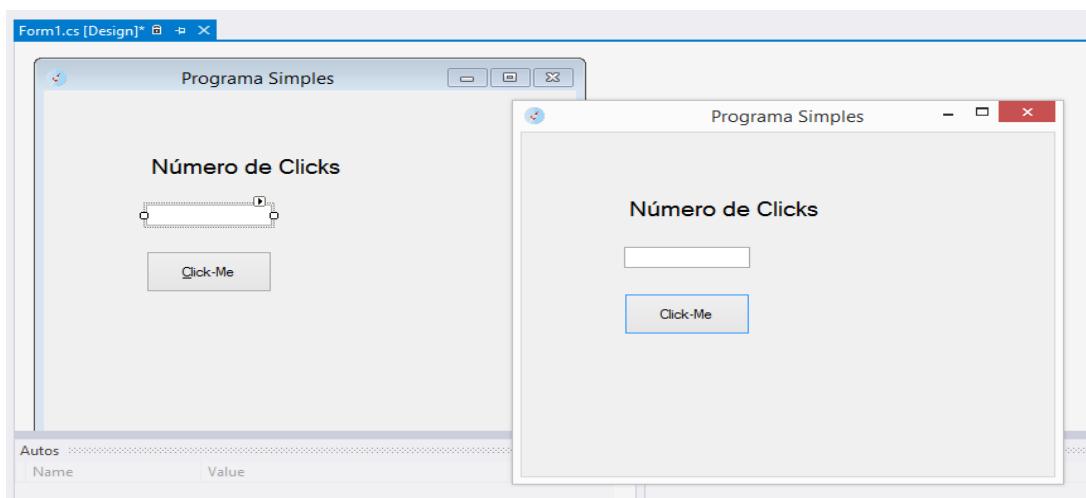
Seu formulário deve estar mais ou menos assim:



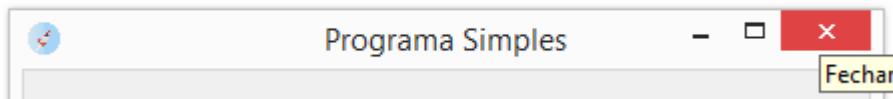
Execute o programa para ver se tudo funciona. Para executá-lo, dê um click no ícone “Iniciar Depuração”, ou pressione a tecla **F5**.



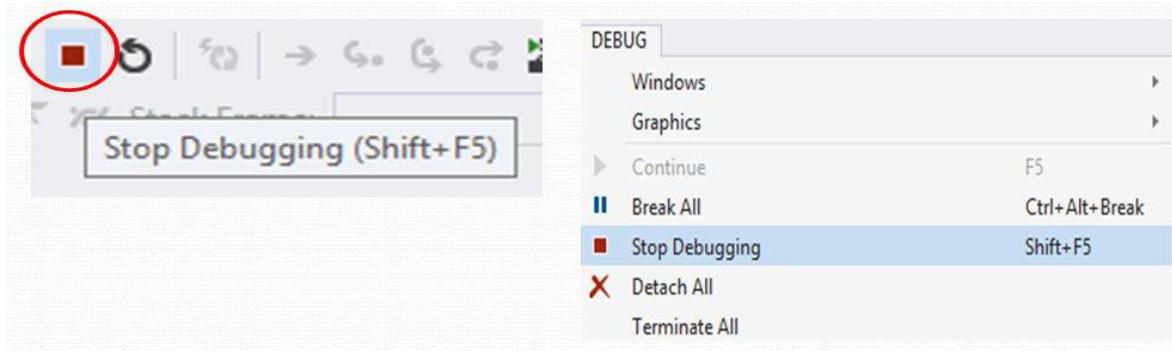
Abaixo vemos o ambiente de desenvolvimento do C# e a janela de nosso programa sendo executado. Veja se tudo funciona na interface. Note que os componentes ainda não têm programação nenhuma. Por esse motivo, nada acontece quando se pressiona o botão "Click-me".



Para parar a execução, dê um click no ícone “Fechar” do formulário de sua aplicação.



Ou click nos itens de menu Depurar / Parar Depuração.



Continuando, vamos inserir um pouco de programação. Dê um click duplo no botão “Click-me” para abrir a janela de código e no tratador do evento click desse botão, digite o código a seguir.

Obs.:

Chamamos de “Tratador de Evento” o procedimento que é executado quando ocorre um evento específico no componente. Por exemplo, o procedimento de nome “btnClickMe_Click” da figura abaixo é executado, quando ocorre o click do mouse no botão de nome “btnClickMe”.

```
namespace WFAProgramaSimples
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void btnClickMe_Click(object sender, EventArgs e)
        {
            int contador = 0;
            contador = contador + 1;
            txtContaClicks.Text = contador.ToString();
        }
    }
}
```

Execute o programa p/ ver seu funcionamento – deve ficar como abaixo:



Mas por que será que esse programa não funciona? A cada click do mouse, o contador deveria ser acrescido de uma unidade.

Parece que a cada click do mouse no botão, a variável contador é criada, inicializada com zero e incrementada. Esta é convertida para string e atribuída à propriedade Text de txtContaClicks. Quando o procedimento btnClickMe_Click encerra sua execução, a variável contador é destruída.

Conserte o programa!

Uma sugestão é declarar a variável contador antes do procedimento. Dessa maneira, essa variável seria declarada globalmente e seria reconhecida por qualquer procedimento que a chamassee.

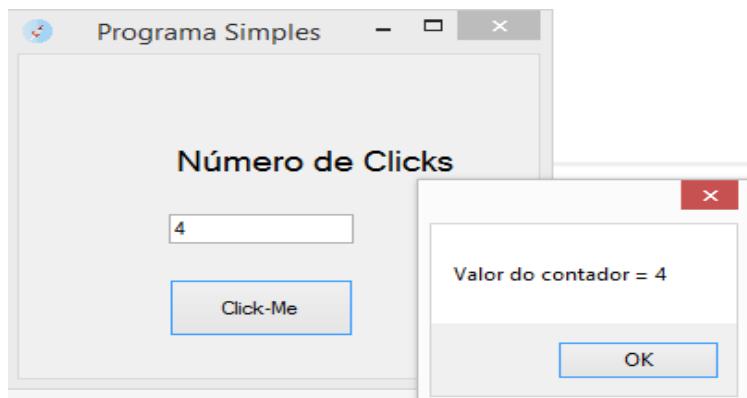
Exercício: Verifique se o que foi afirmado acima é verdadeiro...

Inclua mais uma funcionalidade no programa de modo que ao ocorrer um click no formulário, deve ser exibida uma caixa de mensagem informando o conteúdo da variável contador, que foi declarada globalmente.

A instrução que faz isso é mais ou menos assim:

```
private void Form1_Click(object sender, EventArgs e)
{
    string mensagem = "Valor do contador = " + contador;
    MessageBox.Show(mensagem);
}
```

E nossa interface ficará mais ou menos como a seguir:



Obs.:

Ao compilar o programa, o compilador do C# gera o módulo executável na pasta “Bin”. Para verificar, salve tudo (Arquivo / Salvar Tudo), feche o C# e abra a pasta Bin / Debug. Localize o programa executável e dê-lhe um click duplo. Verifique que o programa é executado sem a necessidade do editor do C#.

Nomes de variáveis

A atribuição de nomes a variáveis deve obedecer a critérios que tem por objetivo facilitar o entendimento do programa, facilitar futura manutenção e adequação ao C#. Esses critérios são:

- O nome começa com letra minúscula
- Não pode conter caracteres especiais;
- Deve ter relação com o significado da variável;
- Se o nome da variável é formado por mais de uma palavra, as demais palavras começam com letra maiúscula – exemplo “impostoDeRenda”. Esse tipo de notação é chamado “Notação Húngara”.

Nomes de componentes (Controles)

Os critérios abaixo listados têm por objetivo identificar o tipo de componente, facilitar o entendimento do programa, sua futura manutenção e adequação ao C#. Do mesmo modo como os nomes das variáveis:

- Não pode conter caracteres especiais;
- Deve ter relação com o objetivo do componente;
- Se o nome do componente é formado por mais de uma palavra, as demais palavras começam com letra maiúscula – exemplos abaixo.
- Deve iniciar por letra minúscula, sendo que as três primeiras letras identificam o tipo de componente, como por exemplo:

Componente	Identificação	Exemplo
Botão de comando	btn	btnClickMe
Label	lbl	lblMensagem
Caixa de Texto	txt	txtContaClicks
Botão de Opção	opt	optSimOuNao

Observe que o nome do componente (name) é tão importante, que ele aparece no topo da lista, na janela de propriedades. É através do nome do componente que acessamos o componente, bem como os tratadores de eventos do componente.

Portanto, um dos primeiros passos ao criarmos a interface do aplicativo, é configurarmos as propriedades dos componentes, **principalmente a propriedade name**.

OBS.:

C# é “case sensitive”, ou seja, diferencia letras maiúsculas de minúsculas. Por esse motivo, as referências às propriedades dos componentes devem respeitar as letras maiúsculas e minúsculas de sua definição. Por exemplo, uma referência ao texto de uma caixa de texto, deve ser **txtMensagem.Text="Olá";** e não **txtMensagem.text="Olá!"**; Neste caso, ocorrerá erro e a mensagem sobre o erro não é muito clara!

Palavras reservadas em C#

C# tem palavras reservadas. Portanto, não devemos utilizar como nomes de variáveis ou componentes, os nomes abaixo:

abstract, as, base, bool, break, byte, case, catch, char, checked, class, const, continue, decimal, default, delegate, do, double, else, enum, event, explicit, extern, false, finally, fixed, float, for, foreach, goto, if, implicit, in, int, interface, internal, is, lock, long, namespace, new, null, object, operator, out, override, params, private, protected, public, readonly, ref, return, byte, sealed, short, sizeof, stackalloc, static, string, struct, switch, this, throw, true, try, typeof, uint, ulong, unchecked, unsafe, ushort, using, virtual, volatile, void, while

Tipos de variáveis

Em nosso “ProgramaSimples” digitamos a instrução: **int contador=0;** Essa instrução declara a variável de nome “contador”, informa ao compilador que ela conterá valores inteiros e a inicializa com o valor zero.

C# fornece os tipos de dados a seguir, chamados de “tipos de dados primitivos”:

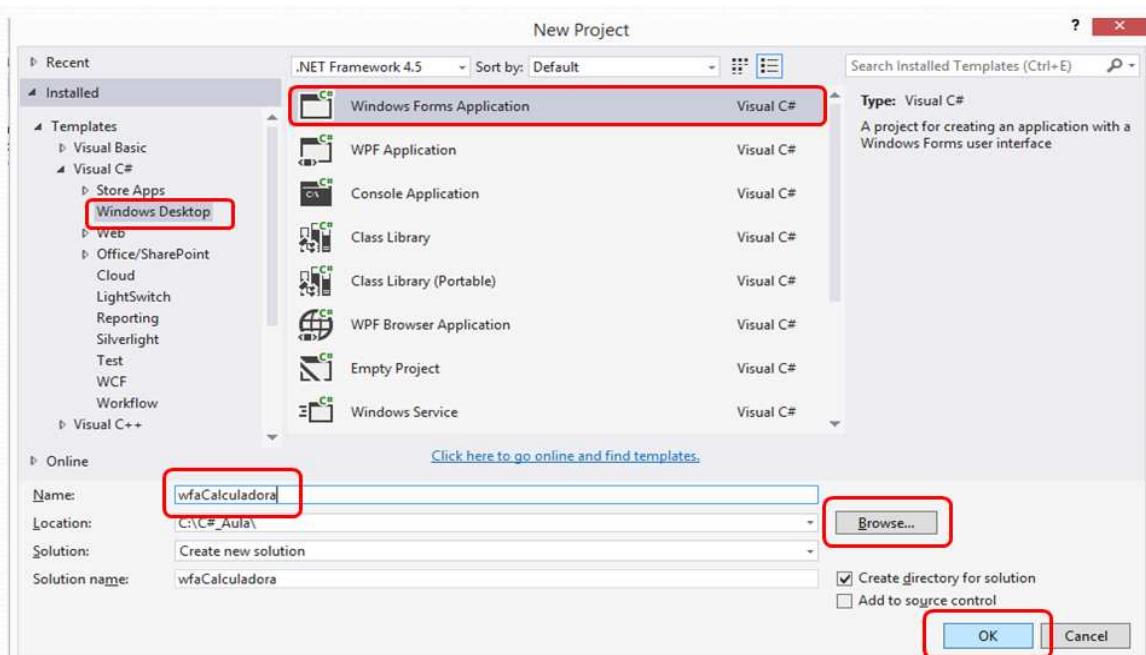
TIPO	CLASSE .NET	DESCRÍÇÃO	TAMANHO EM BITS	CAPACIDADE
int	Int32	Signed integer	32	-2,147,483,648 to 2,147,483,647
long	Int64	Signed integer	64	-922337203685477508 to 922337203685477507
float	Single	Single-precision floating point type	32	-3.402823e38 to 3.402823e38
double	Double	Double-precision floating point type	64	-1.79769313486232e308 to 1.79769313486232e308
char	Char	A single Unicode character	16	Unicode symbols used in text
bool	Boolean	Logical Boolean type	8	True or false
string	String	A sequence of characters		
decimal	Decimal	Precise fractional or integral type that can represent decimal numbers with 29 significant digits	128	$\pm 1.0 \times 10^{-28}$ to $\pm 7.9 \times 10^{28}$

AULA 2

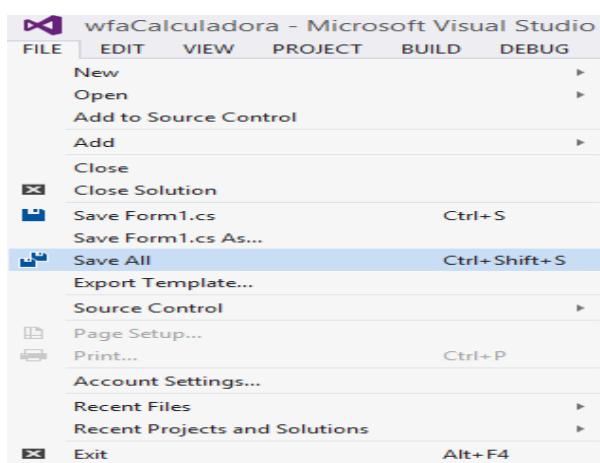
Projeto Calculadora

Vamos iniciar um novo projeto.

- Para isso, dê um click nos itens de menu “Arquivo / Novo / Projeto”.
- Selecione o template “Aplicativo de Windows Forms”, mude o “Nome” do projeto para wfaCalculadora.
- No item "Localização" direcione a gravação para o dispositivo e pasta onde você pretende gravar seu projeto. Serão gravadas várias pastas e arquivos. Se a caixa "Criar diretório para solução" estiver selecionada, a nova pasta será criada.
- Dê um click no botão "Ok"



- Abra o item “Arquivo” do menu e dê um click no item “Salvar Tudo”.



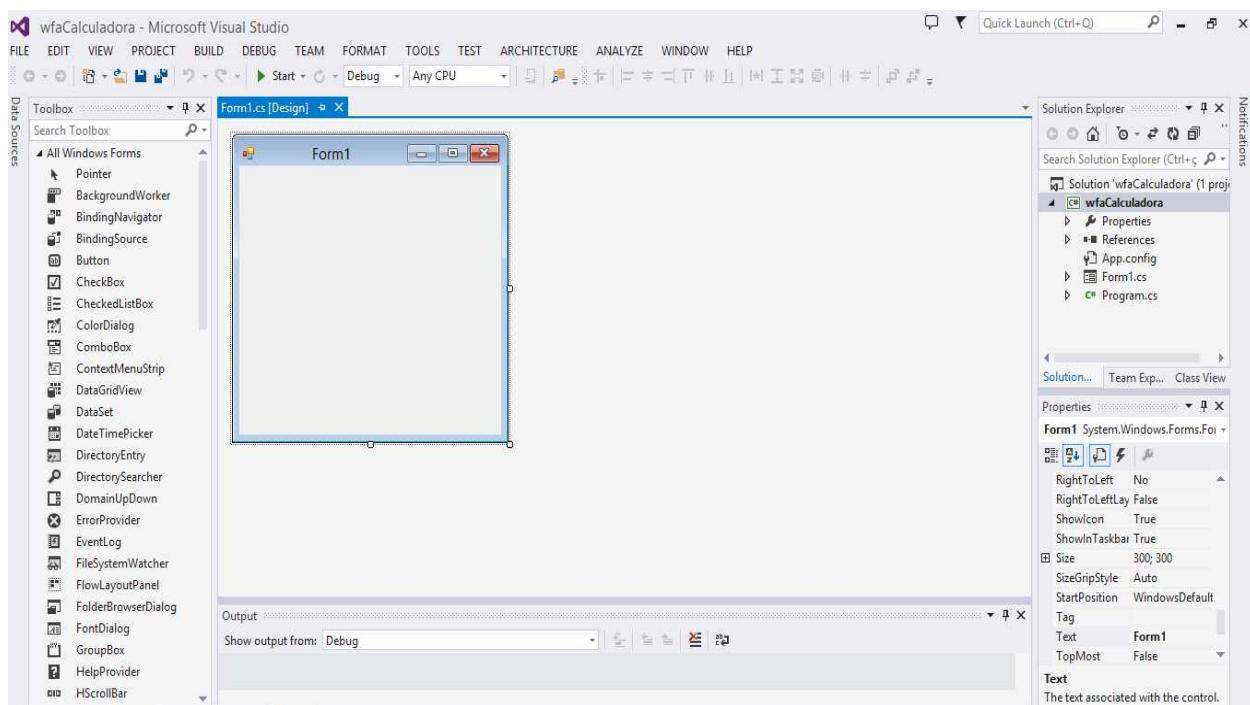
- Pronto! Seu projeto, com todos os arquivos associados, foi salvo. Agora feche o C#.

Abra o Windows Explorer e veja as pastas e os arquivos que o sistema gravou. A primeira pasta é da Solução, ou seja wfaCalculadora. Dentro dessa pasta está o arquivo com o nome do projeto (wfaCalculadora.sln), que será usado quando você for abrir o projeto.

Junto com esse arquivo está a pasta com o mesmo nome, contendo várias pastas e arquivos que serão explorados posteriormente.

Dê um click duplo no arquivo de projeto wfaCalculadora.sln (solução) para continuarmos o desenvolvimento.

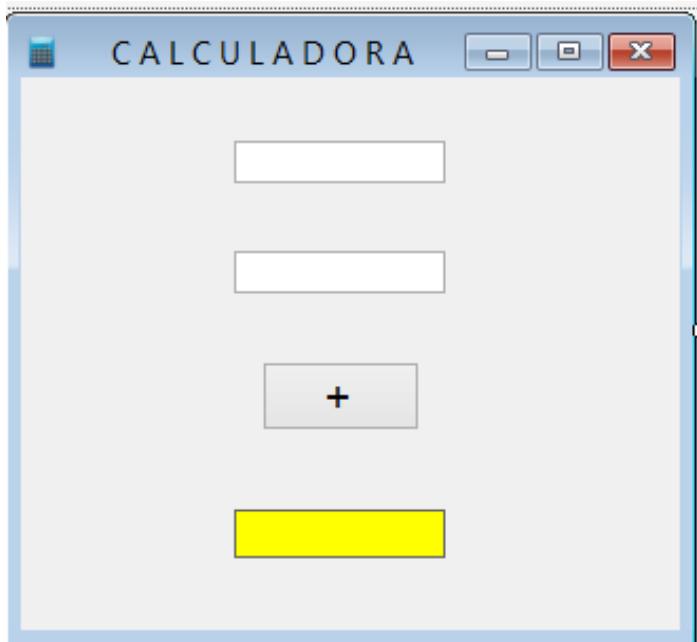
Movimento as janelas do ambiente de desenvolvimento do C# da mesma maneira que você fez em nosso "Programa Simples" da aula anterior. O ambiente deverá ficar mais ou menos como a figura seguinte.



Configure o formulário segundo as instruções a seguir para desenvolvemos uma calculadora simples.

- coloque no formulário duas caixas de texto, de nomes txtN1 e txtN2;
- coloque um botão de nome btnSoma, com Text="+" e Font=16;
- coloque um label de nome lblResult com Text nulo, AutoSize=false, BorderStyle=FixedSingle e TextAligned=MiddleCenter e Backcolor = yellow;
- mude o nome do formulário para frmCalculadora e o Text para "C A L C U L A D O R A"
- inclua um ícone no formulário.

O formulário ficará como a figura a seguir:



Tratador de evento “Click”

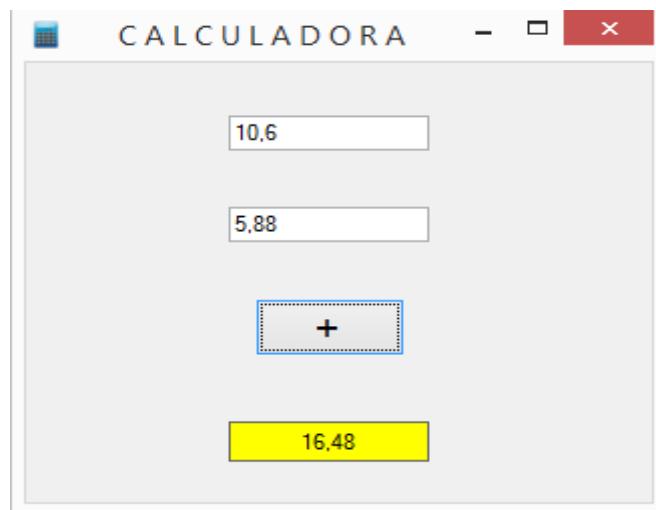
Quando o programa estiver funcionando, se digitarmos valores nas caixas de texto e pressionarmos o botão btnSoma, a soma será realizada. Para programar o recebimento do click no botão, dê um click duplo no botão para abrir a janela de código do C# e digite o seguinte código:

```
namespace wfaCalculadora
{
    public partial class frmcalculadora : Form
    {
        public frmcalculadora()
        {
            InitializeComponent();
        }

        private void btnSoma_Click(object sender, EventArgs e)
        {
            float valor1, valor2, resultado; //declaração de variáveis locais
            valor1 = float.Parse(txtN1.Text);
            valor2 = float.Parse(txtN2.Text);
            resultado = valor1 + valor2;
            lblResult.Text = resultado.ToString();
        }
    }
}
```

Salve o programa, execute-o e teste o programa p/ verificar se está Ok.

Sua interface deverá estar mais ou menos como a seguir:

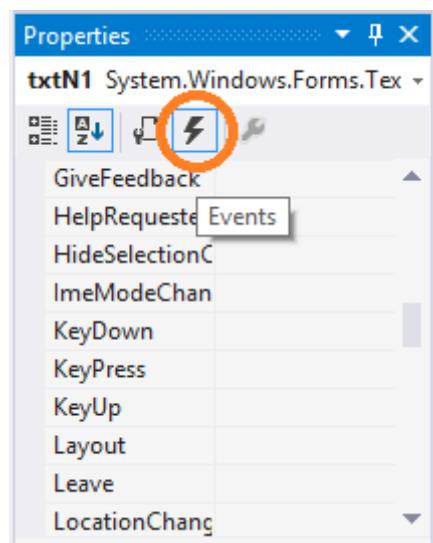


Note que o programa faz soma para número em ponto flutuante, pois nossas variáveis e métodos utilizados são do tipo float. Porém se digitarmos caracteres inválidos, no cálculo da soma ocorrerá erro de execução do programa e este abortará.

Uma das maneiras de evitarmos a digitação de caracteres inválidos é bloquearmos sua digitação, permitindo apenas a digitação de números e da vírgula decimal.

Tratador de evento “KeyPress”

Para isso, podemos usar o método tratador do evento KeyPress da caixa de texto txtN1. Para acioná-lo, selecione a caixa de texto txtN1 e dê um click no ícone de tratadores de eventos da janela “Properties” (aquele ícone do raio) e dê um duplo clique em Keypress.



A codificação do procedimento que trata o evento keypress de txtN1, ficará como abaixo:

```
private void txtN1_KeyPress(object sender, KeyPressEventArgs e)
{
    MessageBox.Show("Você digitou : " + e.KeyChar);
    if ((e.KeyChar < '0' || e.KeyChar > '9') && e.KeyChar != ',')
        {MessageBox.Show("tecla inválida");
        e.KeyChar = (Char)0; // transforma a tecla digitada no carácter nulo
    }
}
```

Instrução de Seleção (IF)

Em C#, a instrução de desvio condicional "if" tem a seguinte forma:

```
if ( expressão lógica)
    Instrução1;
else
    Instrução2;
```

O complemento else é opcional.

Se houver mais de uma instrução na parte do if ou do else, deve-se utilizar chaves. Exemplo:

```
if (a == b)
{
    X = 1;
    Y = 2;
};
```

Isso parece ter resolvido boa parte dos problemas. Porém, há várias outras maneiras de se alimentar a calculadora com valores inválidos. Por exemplo, tente fazer o cálculo sem digitar nenhum valor. Ou então, copie um valor qualquer do Word e cole na calculadora. Ao executar, o cálculo, o programa aborta.

Como resolver isso? Uma maneira é utilizar o bloco try / catch. Veja o exemplo abaixo.

Uso do bloco “try / catch”

```

private void btnSoma_Click(object sender, EventArgs e)
{
    float valorN1, valorN2, resultado; //declaração de variáveis locais

    valorN1 = 0;
    valorN2 = 0;

    try
    {
        valorN1 = float.Parse(txtN1.Text);
        valorN2 = float.Parse(txtN2.Text);
        resultado = valorN1 + valorN2;
        lblResult.Text = resultado.ToString();
    }
    catch (Exception caught)
    {
        lblResult.Text = "";
        lblResult.Text = caught.Message;
    }
}

```

Obs:

Dentro de um bloco try / catch as instruções vão sendo executadas normalmente. Caso ocorra alguma exceção, como por exemplo, erro de atribuição, divisão por zero, erro de formatação, etc., o processamento é desviado para as instruções dentro do “catch”.

Veremos a forma do bloco try / catch com mais detalhes, posteriormente.

Métodos

É o nome dado a funções e procedimentos, quando se utiliza a filosofia de orientação a objetos (OO). Um método em C# tem a seguinte forma geral:

```

TipoRetorno nomeDoMetodo (lista de argumentos)
{
    Instrucao1 ;
    Instrucao2 ;
    ...
    Return ;
}

```

Exemplo:

```
void soma()
{
    float valorN1, valorN2, resultado; //declaração de variáveis locais

    valorN1 = 0;
    valorN2 = 0;

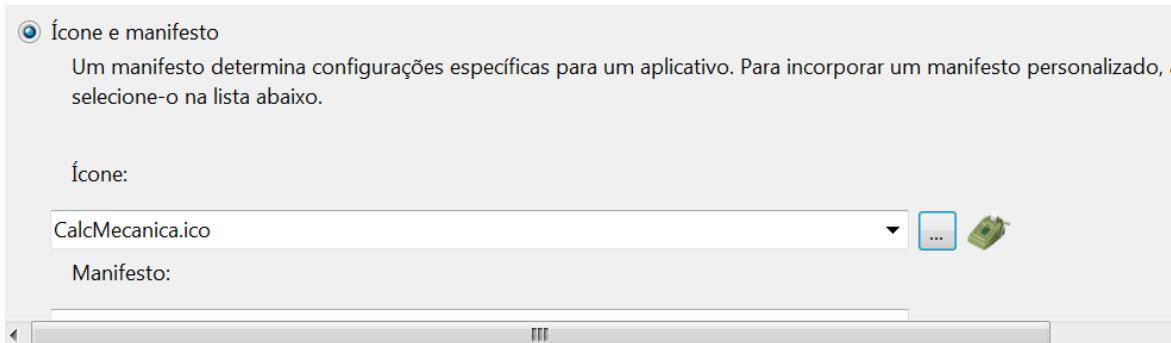
    try
    {
        valorN1 = float.Parse(txtN1.Text);
        valorN2 = float.Parse(txtN2.Text);
        resultado = valorN1 + valorN2;
        lblResult.Text = resultado.ToString();
    }
    catch (Exception caught)
    {
        lblResult.Text = "";
        lblResult.Text = caught.Message;
    }
    return;
}
```

Note que no exemplo acima, para criar o método “soma()” nós copiamos o código contido no tratador de evento click do botão btnSoma e incluímos nele a ultima instrução – ou seja “return”. Agora, no tratador de evento click do botão btnSoma, basta chamarmos o método “soma()”, para realizar a soma, ou seja, ficará assim:

```
private void btnSoma_Click(object sender, EventArgs e)
{
    soma();
}
```

Exercício

1. Incluir o botão “Limpa”, que ao ser pressionado, limpa os conteúdos da calculadora e põe o foco em txtN1. Para mudar o foco de um controle, use o método Focus. Exemplo txtN1.Focus().
2. Se o foco estiver em txtN1 e for pressionada a tecla Enter o foco vai para txtN2. Obs. o valor ASC da tecla Enter, é 13.
3. Aceitar como tecla válida, a tecla BackSpace – seu valor ASC é 8.
4. Incluir um ícone no projeto, para que **o projeto tenha um ícone diferente daquele atribuído ao formulário.** P/ isso, abrir o item de menu Projeto / Propriedades de nomeProjeto, dê um click no botão “...”, igual ao da figura a seguir: Selecione o ícone desejado e compile novamente o programa. Verifique que o módulo executável foi gerado na pasta “Bin”, com o ícone escolhido.



5. Modificar a calculadora para fazer as 4 operações. Sugestão utilize o layout usado na apostila.

Obs. Para todos os botões ficarem do mesmo tamanho, selecione os 4 botões e mude a propriedade Size por exemplo: 110;30. Pode-se também usar o item “Formatar” do menu da interface do C#. Lá existem várias opções de alinhamento.

Crie os métodos “subtração”, “multiplicação” e “divisão” copie o código do método soma() e mude o sinal de operação.

6. Ao ser pressionada uma das teclas: + - * /, a operação correspondente deverá ser realizada. Obs. A partir de agora, estas teclas não serão mais inválidas. Ao ser pressionada uma delas, chamar o método correspondente. Utilizar a estrutura switch.

Instrução de Seleção Switch

Instrução de controle que lida com várias seleções e enumerações, passando o controle para uma das instrução **case** dentro da sua estrutura.

```
private void btnSwitch_Click(object sender, EventArgs e)
{
    int wnumero = 1;

    switch (wnumero)
    {
        case 1:
            MessageBox.Show("Case 1");
            break;
        case 2:
            MessageBox.Show("Case 2");
            break;
        default:
            MessageBox.Show("Default case");
            break;
    }
}
```

transfere o fluxo
de execução
para fora do
corpo do **case**

Se não houver
nenhum rótulo
default, o fluxo é
transferido para fora
do **switch**.

7. Alterar o programa para que não haja a possibilidade de divisão por zero, pois essa divisão é impossível. Nesse caso, o programa deve:

- exibir uma caixa de mensagem (`MessageBox.Show`), avisando o usuário sobre o erro;
 - posicionar o cursor no local do denominador **e deixar os números selecionados** para que o valor seja corrigido. Para isso, utilizar o método `SelectAll` - ex: `txtN2.SelectAll()`;
- 8) Aceitar uma única vírgula decimal. Sugestão: utilizar o método `IndexOf()`. Esse método retorna a posição de um caractere no string, iniciando na posição zero. Se o caractere não for encontrado, retorna -1.

Exemplo

```
private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
{
    int i;
    i = textBox1.Text.IndexOf(',');
    if (i >= 0)
        MessageBox.Show("vírgula encontrada na posição " + i.ToString());
    else
        MessageBox.Show("vírgula não encontrada no string");
}
```

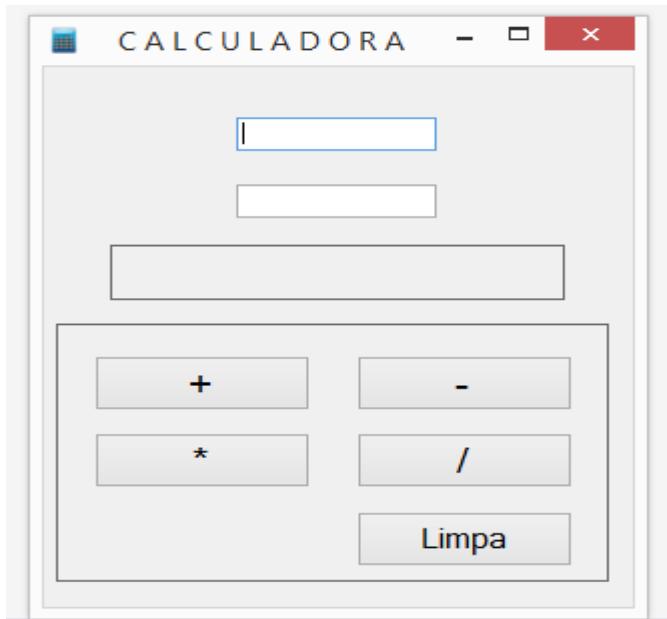
- 9) Se o foco estiver em outro componente que não as caixas de texto quando se pressiona uma das teclas + - * /, note que o programa não realiza a operação. Para resolver esse problema, utilize a propriedade “KeyPreview” do formulário.

Propriedade KeyPreview

Quando essa propriedade é definida como `true`, o formulário receberá qualquer `KeyPress`, `KeyDown`, e eventos de `KeyUp`. Depois que os manipuladores de eventos do formulário terminar o processamento do pressionamento da tecla, o pressionamento de tecla é atribuído ao controle com foco. Por exemplo, se a propriedade de `KeyPreview` é definida como `true` e o controle selecionado é `TextBox`, depois que o pressionamento de tecla ser tratado pelos manipuladores de eventos do formulário o controle `TextBox` receberá a tecla que foi pressionada.

- 10) Se o valor do resultado for negativo, este deve ser na cor vermelha. Se positivo, na cor preta. Sugestão: utilize o tratador de eventos “`TextChanged`”.
 Cuidado! Quando você pressionar o botão “`btnLimpa`”, o valor nulo é atribuído a `lblResult.text`. Aí também ocorre o “`TextChanged`” e nesse caso pode ocorrer um erro de execução se você estiver usando a instrução `if (float.Parse(lblResult.Text)<0)`, pois o método `float.Parse` não converte nulo em zero. Então, antes de verificar se o texto convertido para número é negativo, verifique se o texto é nulo.
- 11) A caixa de texto `txtN2` deve ter as mesmas funcionalidades de `txtN1`, com a diferença de que se o foco estiver em `txtN2` e for pressionada a tecla Enter, a calculadora deve fazer a soma dos valores.

Layout sugerido para a calculadora com as 4 operações:



OBS.

Podemos executar diretamente do código de nosso programa, o método tratador de evento associado a um componente do programa. Por exemplo se quisermos executar o código do tratador do evento Click do botão btnSoma, basta executar o código: **btnSoma.PerformClick();** que o mesmo será executado.

AULA 3

MÉTODO: Vimos que "método" é o nome que se dá a funções associadas a objetos, quando se usa a filosofia de orientação a objetos.

Assinatura do método

- O nome do método e a lista de argumentos formam a assinatura do método, algo que o identifica de maneira única.
- Toda vez que um método for invocado (chamado), a assinatura deve ser obedecida, uma maneira que torna possível identificar um método em relação aos demais.
- Se a assinatura for diferente, ou seja, se qualquer um dos componentes não coincidir com a declaração, ocorrerá erro.

Métodos sem Retorno

- Não retomam valores e são semelhantes às **procedures** encontradas na maioria das linguagens de programação. Os métodos que não retornam valores devem ser definidos como **void**.

Sobrecarga de método

Sobrecarga de método é uma técnica que permite a uma classe possuir mais de um método com um único nome, porém com comportamentos e assinaturas diferentes.

A linguagem C# permite que vários métodos sejam definidos como mesmo nome, desde que eles tenham **assinaturas diferentes**.

Essas diferenças podem ser com relação aos argumentos:

- Com base na quantidade de argumentos,
- Nos tipos dos argumentos ou
- Na ordem dos argumentos recebidos,

Quando um método sobrecarregado é chamado, o compilador avalia e seleciona o método mais adequado à situação, examinando a assinatura correspondente.

Sobrecarga de operador

Sobrecarga de operador acontece quando um mesmo operador executa mais de uma função.

Por exemplo, o operador soma (+) pode realizar a operação de soma aritmética (quando seus operandos forem numéricos) ou pode realizar a concatenação (quando seus operandos forem não numéricos).

Exemplo:Fazer um programa com um botão e um label, com o código abaixo:

```
private void button1_Click(object sender, EventArgs e)
{
    label1.Text = "Área de um quadrado: " + area(3).ToString() + "\n" +
        "Área de um retângulo: " + area(3, 2).ToString() + "\n" +
        "Volume de um paralelepípedo:" + area(3,2,5).ToString();
}

int area (int x)
{
    return (x * x);
}

int area(int x, int y)
{
    return (x * y);
}

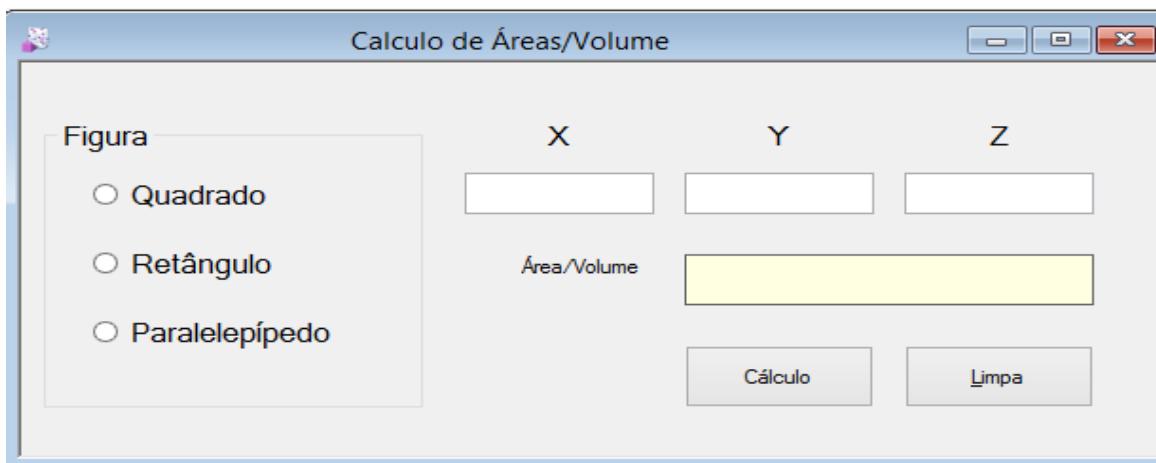
int area(int x, int y, int z)
{
    return (x * y * z);
}
```

Execute o programa, verifique o resultado e analise o código acima.

Exercício: Projeto Calculo de Área

Inicie um novo projeto de nome “wfaCalculaArea” e a partir do conhecimento anterior, fazer um programa com formulário similar ao abaixo que use o método “area()”, com sobrecarga.

O container "Figura" é um GroupBox e os botões de opção são componentes RadioButton.



Componentes utilizados: **GroupBox**, **RadioButton** (rdbQuadrado, etc.), **TextBox** (txtX, etc.), **Button** (btnCalculo), **Label** (lblArea, etc.).

- 1) Ao iniciar a execução da aplicação, o botão de opção “rdbQuadrado” deve estar selecionado (propriedade Checked=true)
- 2) Execute o programa para verificar a ordem de tabulação (propriedade TabIndex), que deve ser: txtX, txtY, txtZ, btnCalculo, optQuadrado, etc. Obs.: Se a propriedade “TabStop” de um componente for configurada “False”, esse componente não será acessado através da tecla Tab.
- 3) O método “area()” deve receber argumento do tipo float e retornar valor float.
- 4) Ao ser pressionado o botão “btnCalculo”, verificar se os valores das caixas de texto necessárias para o cálculo estão com valores.

Obs. Lembre-se que a instrução “return;” encerra a execução do procedimento e a função “Trim”, elimina brancos de uma string. Veja o exemplo abaixo:

```
private void btnCalcular_Click(object sender, EventArgs e)
{
    float x;
    if (rdbQuadrado.Checked)
    {
        if (txtX.Text.Trim() != "")
        {
            x = float.Parse(txtX.Text);
            lblArea.Text = area(x).ToString();
        }
        else
        {
            MessageBox.Show("Preenchimento Obrigatório X ");
            txtX.Focus(); // põe o foco em txtX
            return;
        }
    }
}
```

- 5) As caixas de texto devem aceitar apenas números e uma única vírgula decimal, da mesma maneira que fizemos com o exercício da calculadora, porém neste exercício utilizaremos métodos em uma classe de nome "testes", que iremos criar.

Criação de uma classe

Abaixo da classe Form, criar a **Classe "testes"**, onde deverão estar os métodos **consistNum** e **soUmaVirgula**.

O método consistNum abaixo, recebe como argumento o caractere digitado e só aceita números, vírgula e tecla backspace. Qualquer outro caractere resultará no caractere nulo.

Considerando-se que um campo numérico em ponto flutuante deve conter apenas uma vírgula decimal, o método deverá ser chamado quando o usuário digitar o caractere vírgula. Esse método recebe o conteúdo da caixa de texto onde se está digitando a vírgula e a aceita ou não.

O código do método consistNum e soUmaVirgula, dentro da classe "testes", ficará assim:

```
public class testes
{
    public static char consistNum(char c) // só aceita números, vírgula e backspace
    {
        if (((c < '0') || (c > '9')) && c != ',' && c != (char)8)
        {
            MessageBox.Show("Tecla inválida...");
            c = (char)0;
        }
        return (c);
    }

    public static char soUmaVirgula(string texto)
    {
        int i;
        i = texto.IndexOf(',');
        if (i >= 0)
        {
            MessageBox.Show(" Vírgula já existente...");
            return ((char)0);
        }
        else
            return (',');
    }
}
```

Depois que criarmos a classe **testes** contendo os métodos **soUmaVirgula** e **consistNum**, o código no tratador do evento KeyPress na caixa de texto **txtX**, deverá ficar mais ou menos como o exemplo abaixo:

```

private void txtX_KeyPress(object sender, KeyPressEventArgs e)
{
    if (e.KeyChar == ',')
        e.KeyChar = testes.sóUmaVirgula(txtX.Text);

    e.KeyChar = testes.consistNum(e.KeyChar);
}

```

- 6) Acrescentar um botão que ao ser clicado, limpa as caixas de texto e o label colocando o foco em txtX. A opção quadrado deve ser selecionada.
- 7) Ao selecionar as figuras, inibir as caixas de texto que não estão sendo usadas. Por exemplo se for selecionado a figura quadrado as caixas de texto Y e Z deverão ser inibidas. Você pode programar no evento CheckedChanged de cada figura.
- 8) Elaborar o método consistNumReal(char c, string texto), que faz a consistência de números reais digitados nas caixas de texto.

Sugestão: Juntar a programação dos exemplos acima em uma lógica única e construir o método "consistNumReal".

A chamada poderá ser como abaixo:

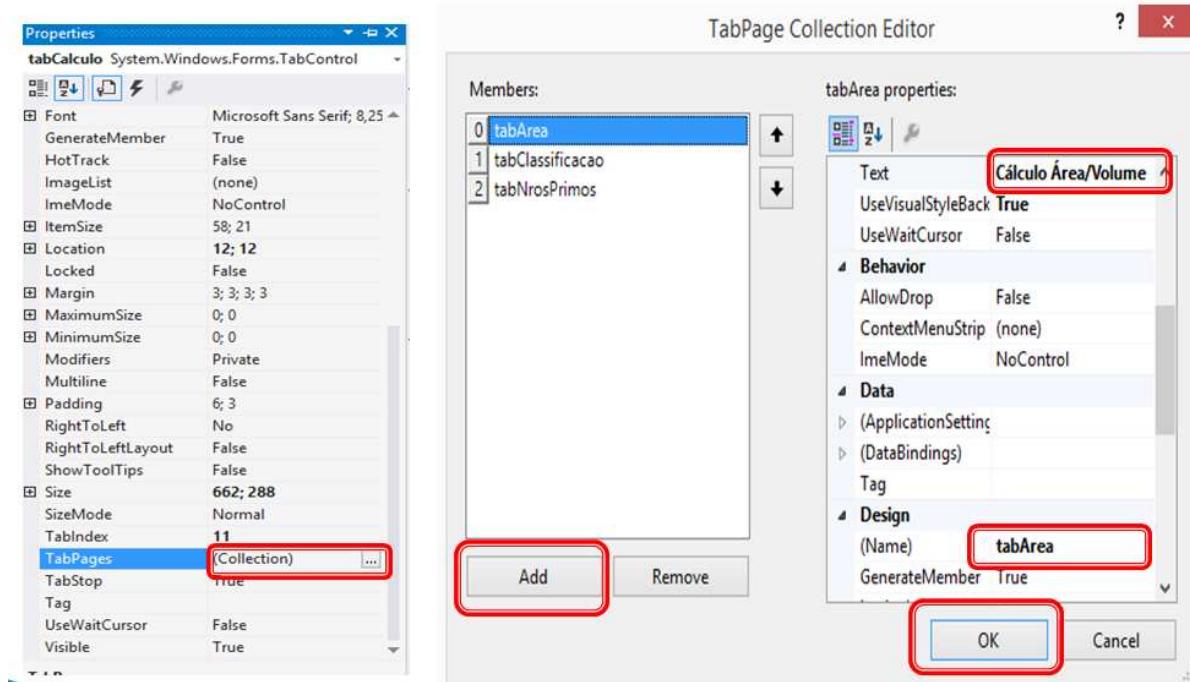
```

private void txtX_KeyPress(object sender, KeyPressEventArgs e)
{
    e.KeyChar = testes.consistNumReal(e.KeyChar, txtX.Text);
}

```

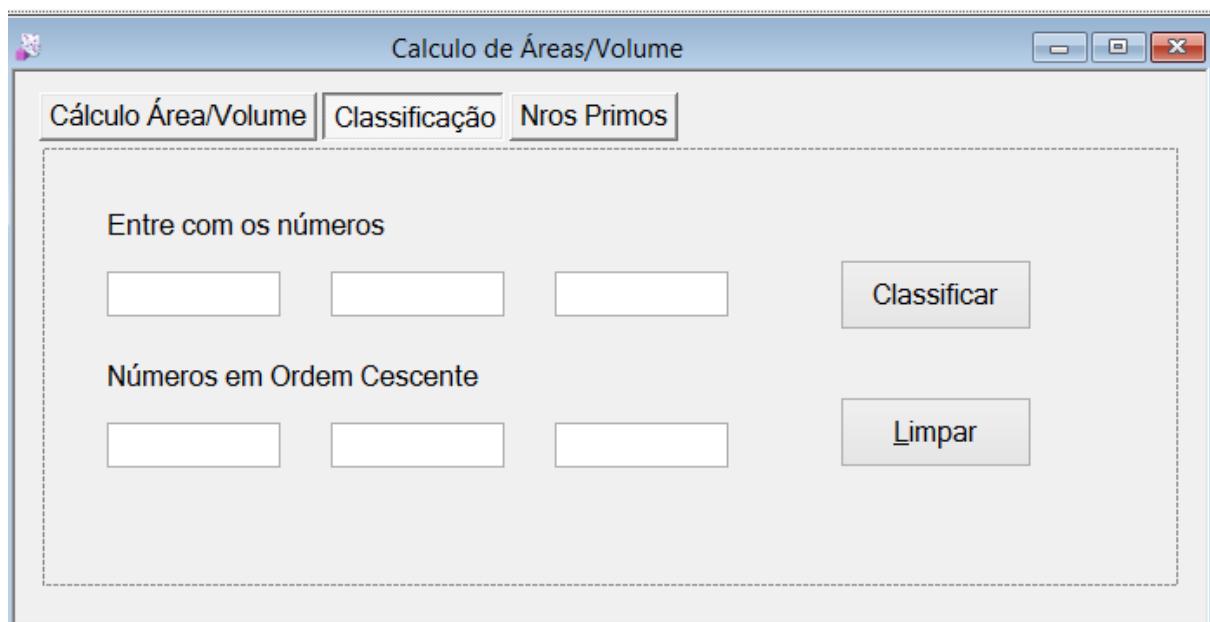
Componente tabControl

- 9) Inclua o componente TabControl e altere as seguintes propriedades
 - Name: tabCalculo
 - Appearance: Buttons
 - Tabpages: clique em “Collection” e vai aparecer a tela de formatação das abas. Altere o texto da primeira aba para “Cálculo da Área” e o name para tabArea, o texto da segunda aba para “Classificação” e o name para tabClassificacao e o texto da terceira aba para “Números Primos” e name tabNrosPrimos.



10) Arraste os componentes que você já tinha inserido (Cálculo da Área/Volume) na primeira aba e veja se a aplicação está funcionando.

11) Insira os componentes da segunda aba como na figura abaixo e faça a seguinte implementação:



- Considerando que são informados A, B e C (variáveis numéricas), pede-se listá-los em ordem crescente com o uso de 2 (dois) ou menos comandos IF (comparação).
- Validar os campos

- Não permitir que as caixa de texto dos números classificados sejam alteradas.
- Após clicar no botão limpar, o foco deve ir para a primeira caixa de texto

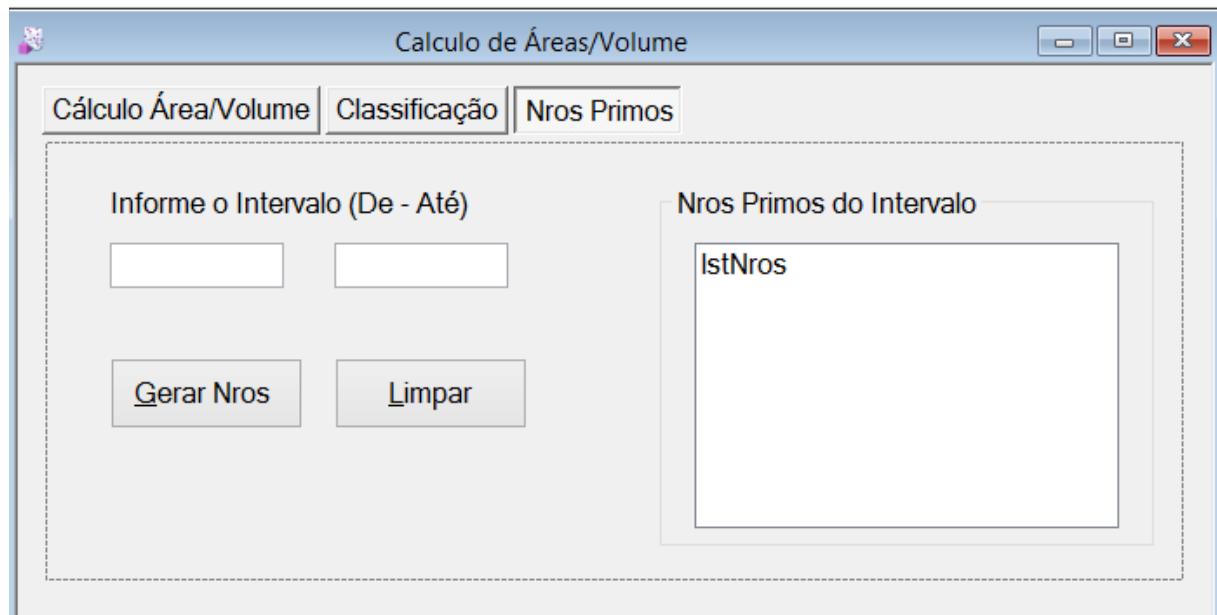
Declaração de Jump

Goto

Permite pular diretamente para outra linha específica do programa, indicada por um label.

```
Goto Teste;
MessageBox.Show("Não deve ser executado");
Teste:
MessageBox.Show("Vamos continuar desse ponto");
```

12) Inclua os componentes na terceira aba conforme figura abaixo.



Componente ListBox

O componente ListBox permite mostrar uma lista de itens.

Para adicionar itens no listbox

```
lstNros.Items.Add("Item " + x.ToString());
```

Para limpar a lista use o seguinte:

```
lstNros.Items.Clear();
```

13) Implemente o seguinte:

- Dado um intervalo de números, liste os números primos.
- **Números primos** são os números naturais maior que zero, que têm **apenas dois divisores diferentes**: o 1 e ele mesmo.

Exemplos

- a. **2** tem apenas os divisores **1** e **2**, portanto **2** é um número primo
- b. **17** tem apenas os divisores **1** e **17**, portanto **17** é um número primo.
- c. **10** tem os divisores **1, 2, 5** e **10**, portanto **10 não** é um número primo.
- d. **3** tem apenas os divisores **1** e **3**, portanto é um número primo.

Observações:

- ⇒ **1 não é um número primo**, porque ele tem apenas um divisor que é ele mesmo.
- ⇒ **2** é o único número primo que é par.

- Validar os campos
- Após clicar no botão Limpar o foco deverá ir para a primeira caixa de texto e a lista deverá ser limpa.

Comandos de Iteração – Loops

Permite executar uma instrução ou bloco de instruções repetidamente até que uma condição seja encontrada.

For

```
for (int i = 1; i <= 5; i++)
{
    // bloco de código
}
```

```
MessageBox.Show(i.ToString());
}
```

While

```
int n = 1;  
while (n < 6)  
{  
    Console.WriteLine("Valor atual de n é {0}", n);  
    n++;  
}
```

14) Verifique se todas as tabulações estão corretas.

Testar o programa e apresentar ao professor.

AULA 4

Depurando uma aplicação

O Debug permite executar os programas em um ambiente de testes, onde podemos executá-los passo-a-passo, examinar valores de variáveis e expressões e alterar variáveis. Isso é importante para o desenvolvimento dos softwares, pois permite verificar o que acontece com a aplicação durante a execução e com isso eliminar os possíveis erros dos programas.

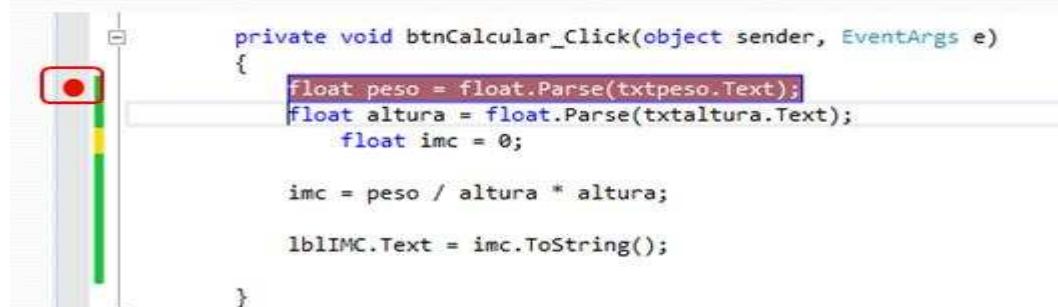
Pontos de Interrupção

Um ponto de interrupção é um sinal que diz ao depurador para suspender a execução do programa num certo ponto.

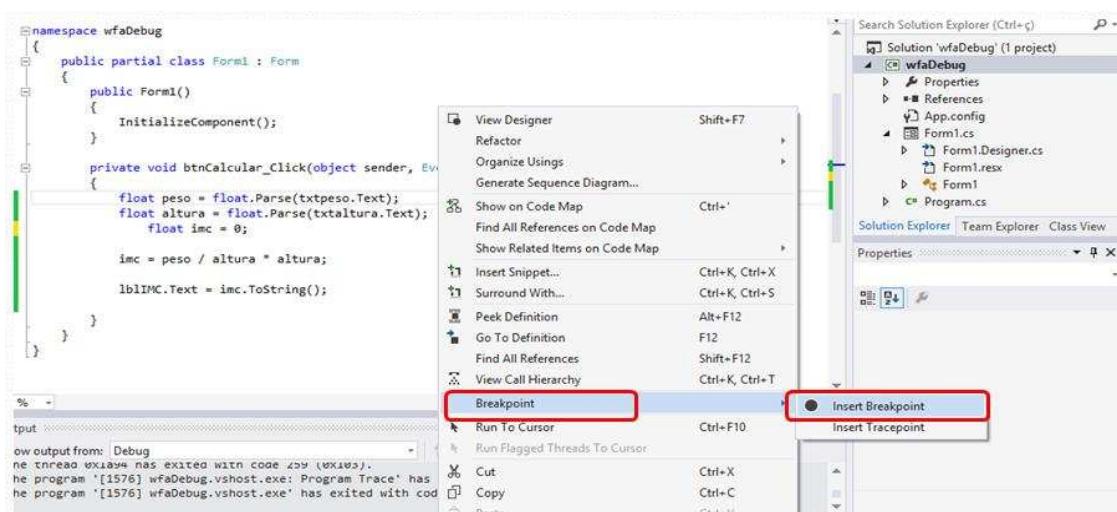
Para executar uma aplicação passo a passo, devemos marcar as linhas do nosso programa onde queremos examinar. Podemos marcar os pontos de interrupção da seguinte forma:

Incluir Pontos de Interrupção

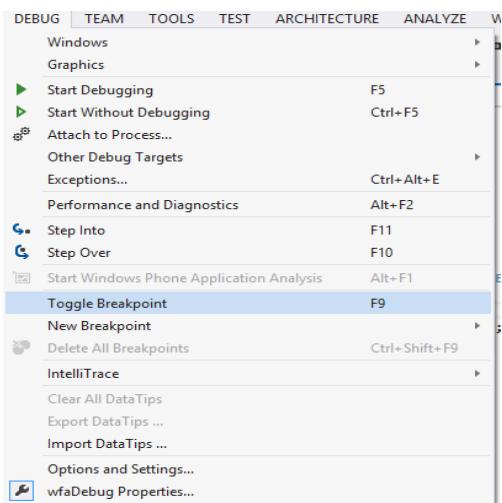
1. Na janela source, click na linha do código onde você quer marcar o ponto de interrupção.



2. No menu do atalho (botão direito do mouse), click no item **Ponto de Interrupção**, e então click em **Inserir Ponto de Interrupção**.



3. No menu Depurar, click Novo Ponto de Interrupção ou (F9).



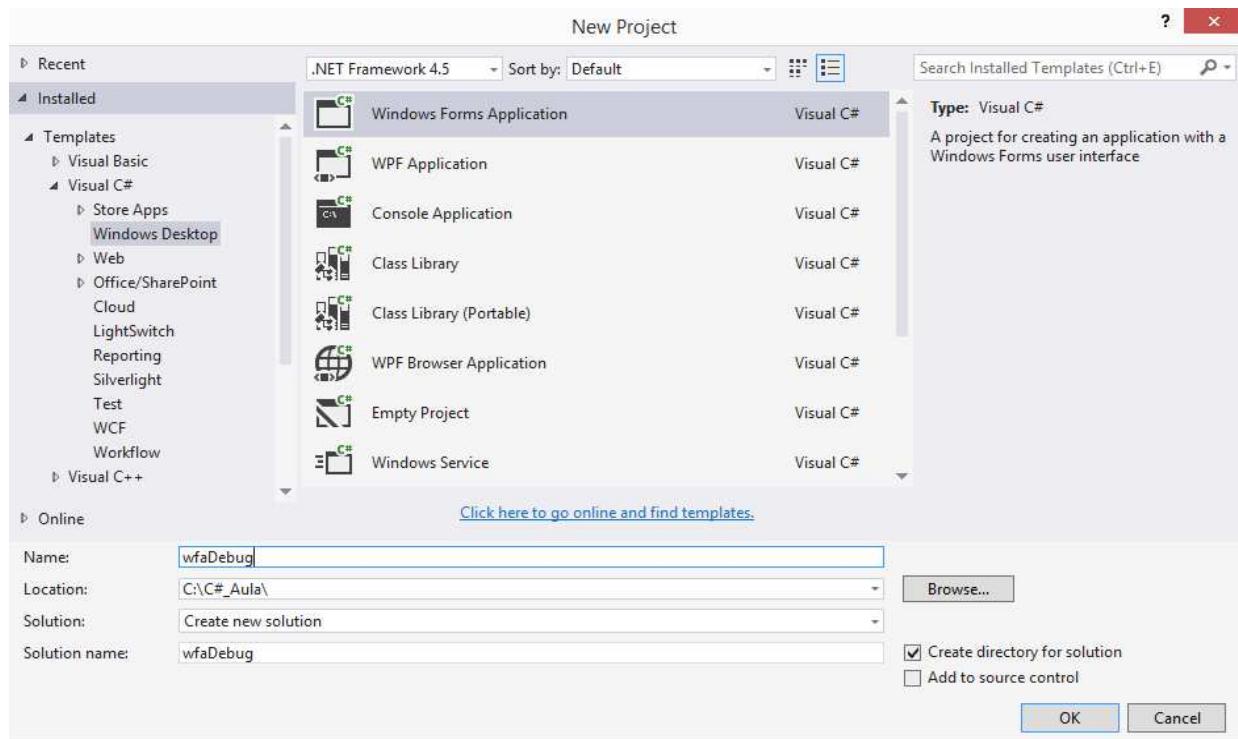
Ao iniciarmos aplicação a execução irá parar quando chegar no local do ponto de interrupção. Uma vez parado, temos várias opções para examinar a aplicação, como por exemplo:

- F10 - Executar sem entrar nas funções
- F11 – Executar entrando nas funções
- F5 - Continuar executando até o próximo ponto de interrupção

Usando o recurso de depuração

Para iniciar o trabalho

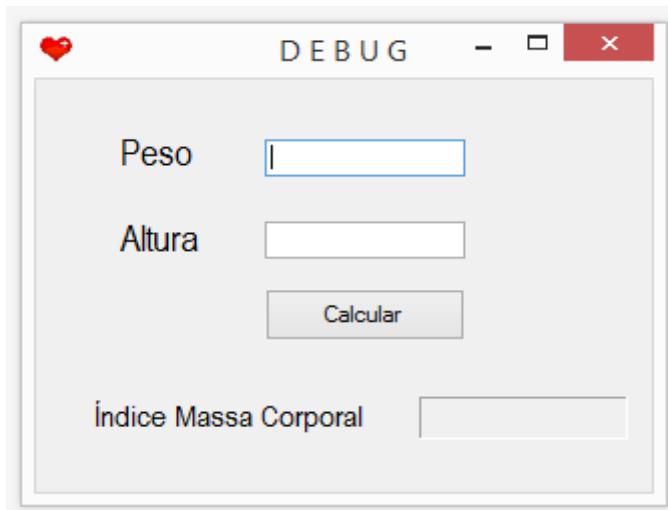
1. Crie um novo projeto chamado wfaDebug



2. Adicione ao formulário:

- 2 controles TextBox (Name: txtPeso e txtAltura),
- 1 Button (Name: btnCalcular) e
- 1 label (Name: lblIMC; BorderStyle = Fixed3D; AutoSize = false).

3. Posicione e configure os controles da forma mostrada na figura abaixo.



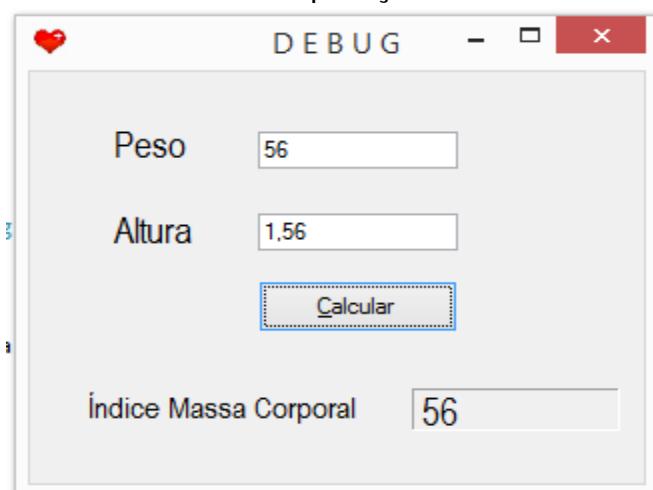
4. Adicione o código abaixo no manipulador de evento Click do controle btnCalcular. Este possui um erro de lógica. Para "descobri-lo", usaremos os recursos de debug.

```
private void btnCalcular_Click(object sender, EventArgs e)
{
    float peso = float.Parse(txtPeso.Text);
    float altura = float.Parse(txtAltura.Text);
    float imc = 0;

    imc = peso / altura * altura;

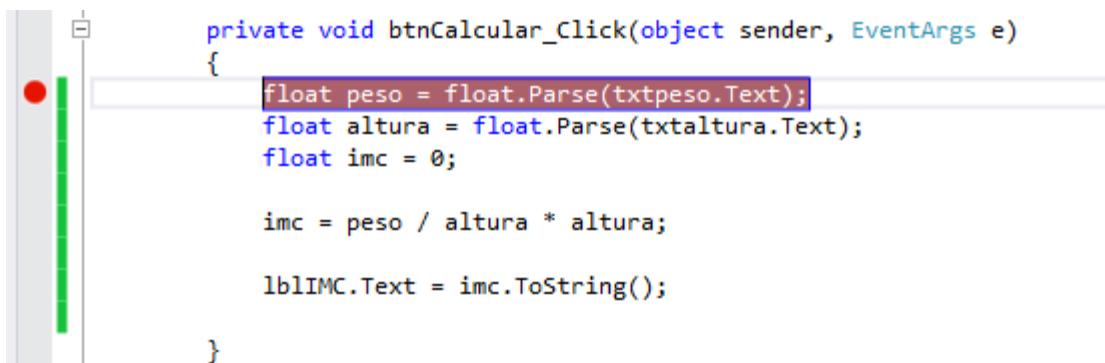
    lblIMC.Text = imc.ToString();
}
```

5. Execute e teste a aplicação. Entre com os seguintes valores e veja o resultado.



O IMC correto seria 23,01. O que está errado? Vamos analisar o que aconteceu.

6. Encerre a aplicação e vamos colocar um ponto de interrupção na primeira linha do método movendo o cursor para lá e pressionando F9. Note que a linha fica destacada em vermelho. Podemos visualizar todos os ponto de interrupção clicando "Ctrl+Alt+B" para exibir uma janela específica:



```
private void btnCalcular_Click(object sender, EventArgs e)
{
    float peso = float.Parse(txtpeso.Text);
    float altura = float.Parse(txtaltura.Text);
    float imc = 0;

    imc = peso / altura * altura;

    lblIMC.Text = imc.ToString();

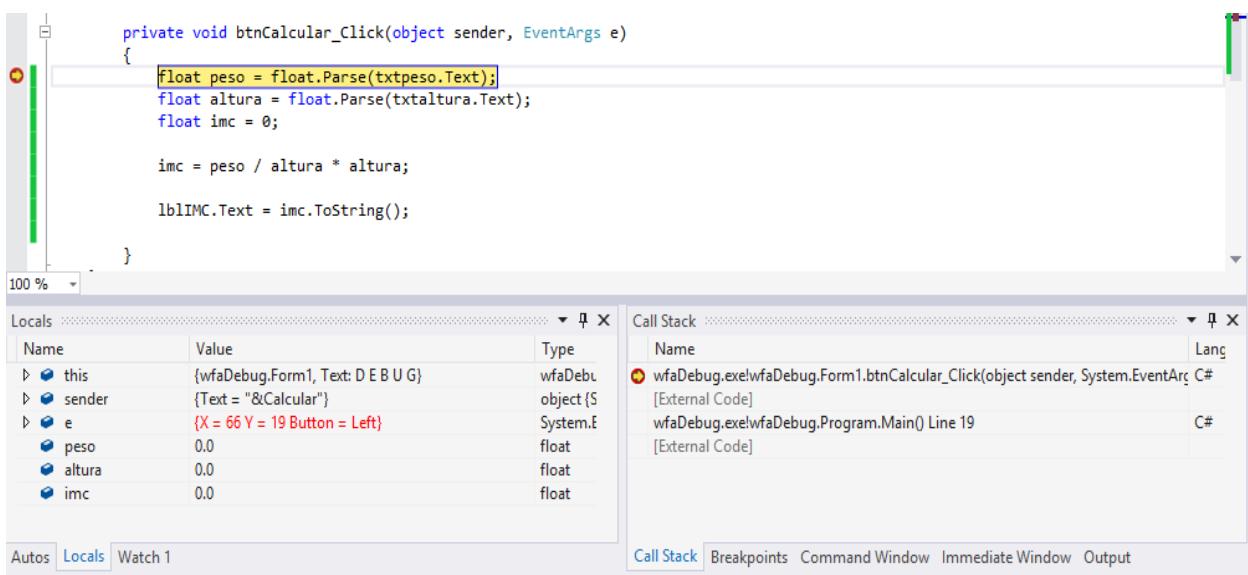
}
```

7. Clique F5 para rodar o programa em modo de debug.

8. Entre números no navegador, por exemplo 56 e 1,56 e clique no botão "Calcula".

Observe que o ponto de interrupção fica destacado com a cor amarela.

As janelas **Locals** e **Call Stack** são apresentadas. Na Janela Locals é possível visualizar todos os objetos (variáveis, controles, etc.) da aplicação e seus valores.



Também é possível "estacionar" o cursor sobre algum elemento no código por alguns instantes e ver seu conteúdo.

```

private void btnCalcular_Click(object sender, EventArgs e)
{
    float peso = float.Parse(txtpeso.Text);
    float altura = float.Parse(txtaltura.Text);
    float imc = 0;

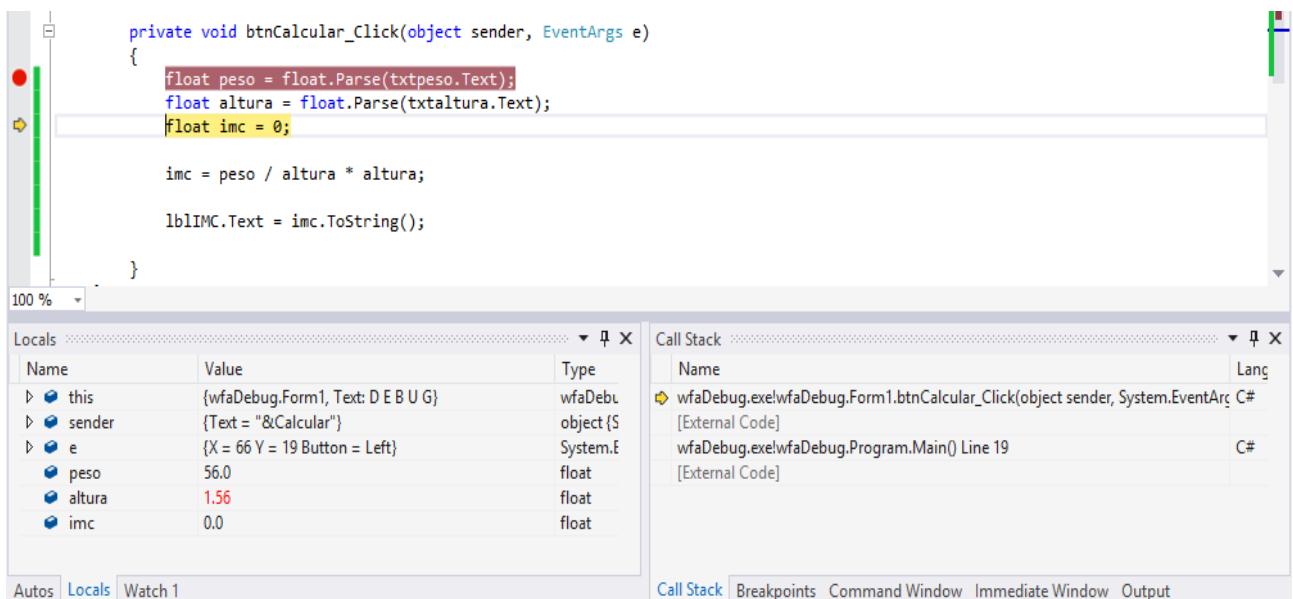
    imc = peso / altura * altura;

    lblIMC.Text = imc.ToString();

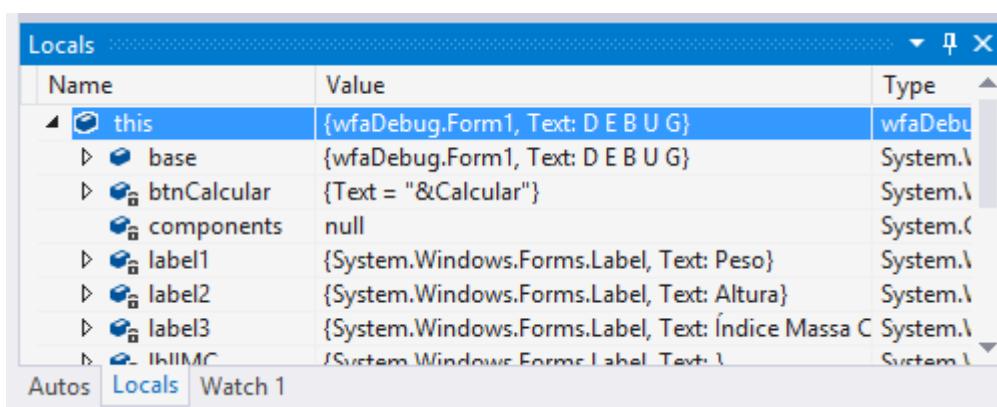
}

```

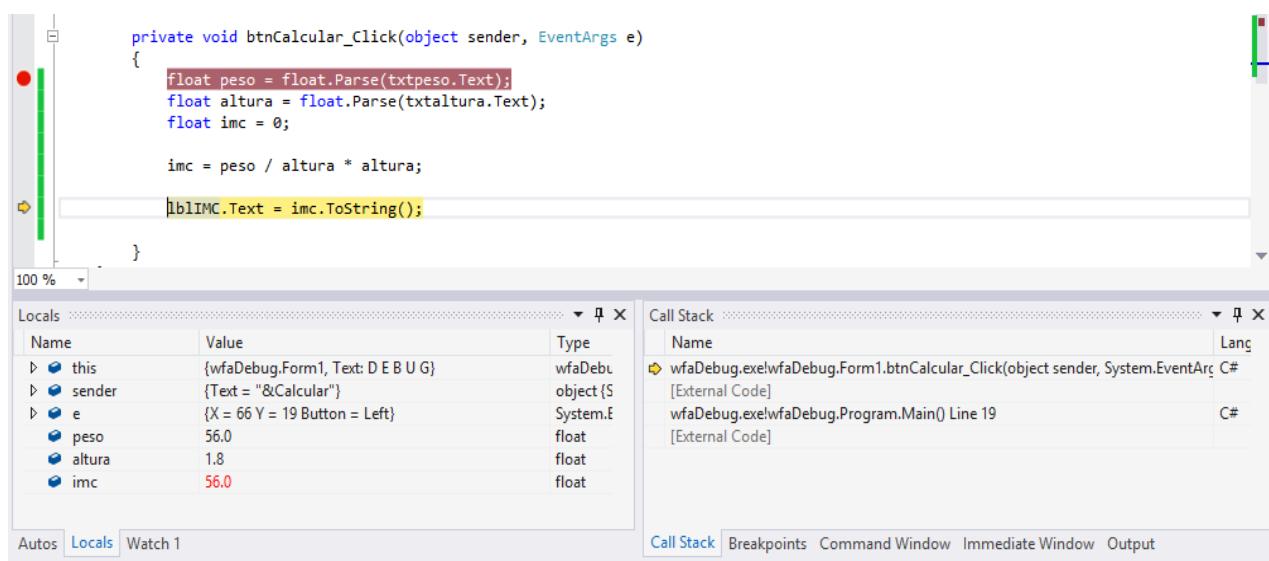
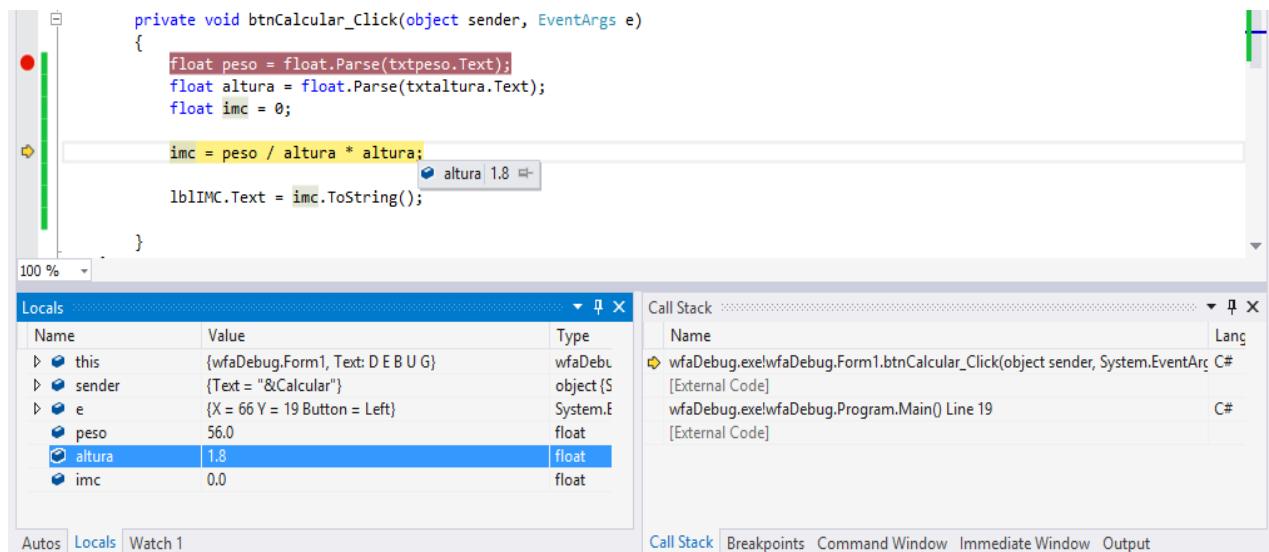
9. Pressione F11 para dar continuidade na execução do código, a faixa amarela se moverá linha a linha, pressione por duas vezes e verifique o valor das variáveis peso e altura na janela Locals. A figura abaixo ilustra esse procedimento.



A janela Locals contém as variáveis locais, argumentos das funções e campos. Note que os valores podem ser expandidos para revelar maiores detalhes, como mostra a imagem abaixo.



Podemos alterar um valor, basta clicar sobre ele e digitar um valor novo, como ilustra a figura abaixo. É com esse valor que a execução da aplicação continuará.



A janela **Inspecção** contém somente os valores usados na função que se deseja ver, basta digitar o nome do elemento. A janela **Inspecção** permite colocar qualquer expressão para ser visualizada. Note que você pode colocar praticamente qualquer expressão, não apenas simples variáveis, como ilustra as figuras abaixo.



Poderíamos continuar o debug pressionando F11 para executar o próximo passo, seguindo o fluxo do programa ou F5 para executar a próxima função, caso uma seja chamada.

Uma vez que analisamos o comportamento do programa e o conteúdo das variáveis, temos que alterar o nosso código para acertar o cálculo do IMC.

```
private void btnCalcular_Click(object sender, EventArgs e)
{
    float peso = float.Parse(txtpeso.Text);
    float altura = float.Parse(txtaltura.Text);
    float imc = 0;

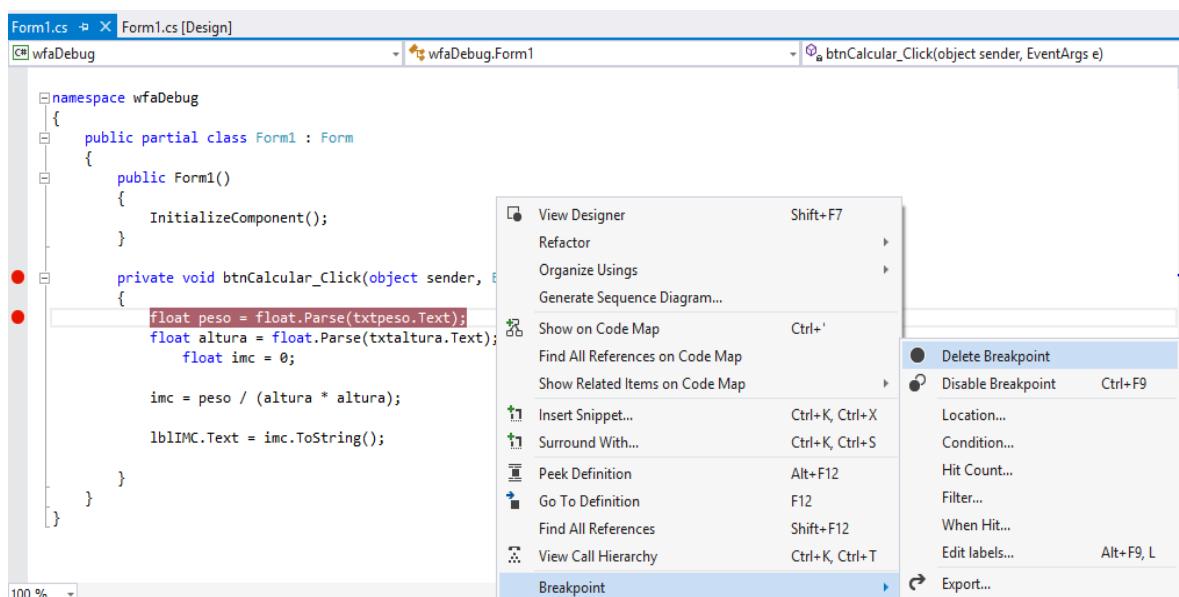
    imc = peso / (altura * altura);

    lblIMC.Text = imc.ToString();

}
```

Excluir um ponto de interrupção

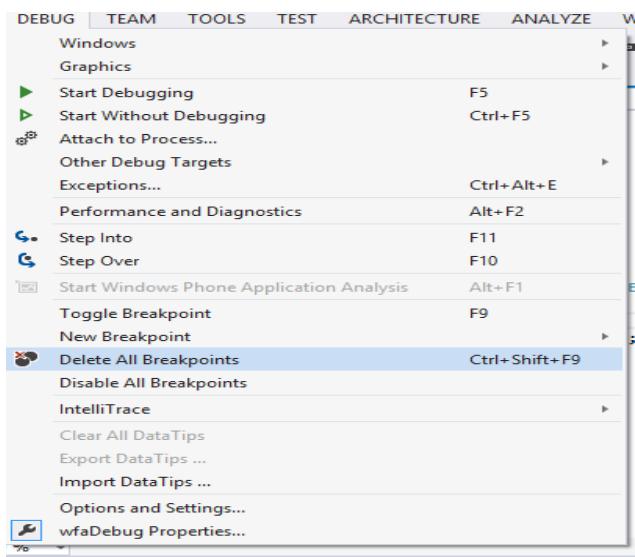
1. Na janela source clique com mouse da direita e selecione **ponto de interrupção**, e **Apagar Ponto de Interrupção**.



2. Excluir todos os Pontos de Interrupção

Podemos excluir todos os pontos de interrupção da aplicação pressionando Ctrl+Shift+F9 ou usando a opção de menu: Depurar>Apagar Todos os Pontos de Interrupção.

APOSTILA DE VISUAL C#



Acesso a Métodos de Outras Classes

O acesso a métodos externos, pertencentes a outra classe é mais comum do que a utilização de métodos da mesma classe.

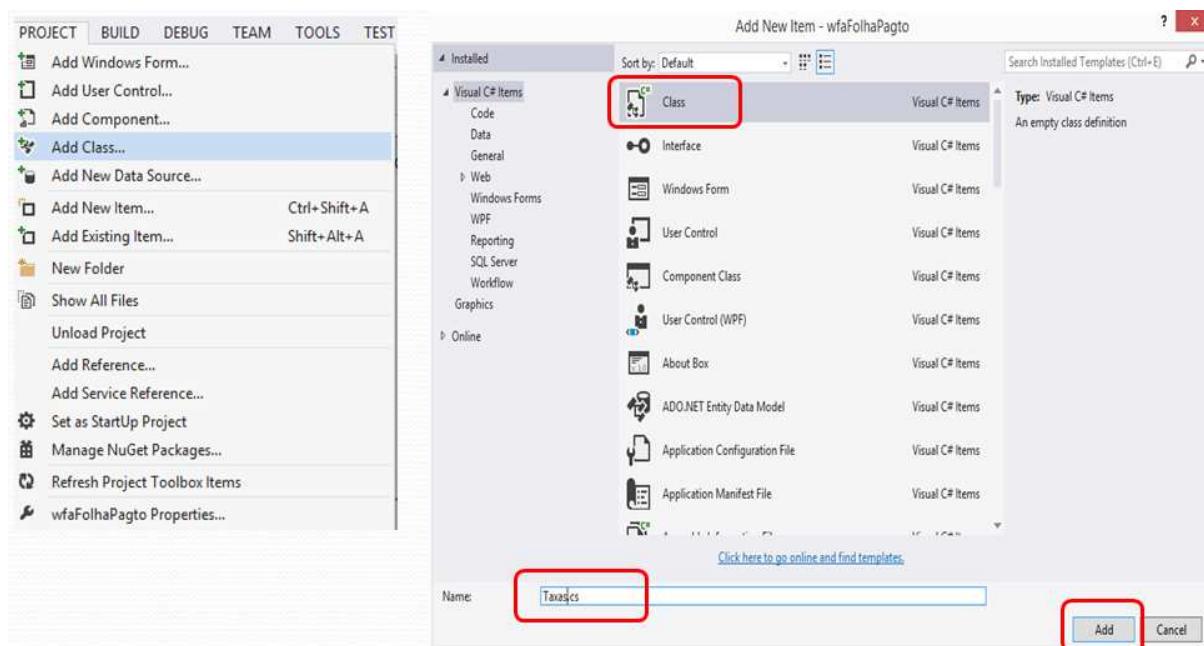
Qualquer método do tipo **static** criado em uma classe pode ser utilizado em outra classe pelo formato:

nome-da-classe.nome-do-método

Com isso, um método necessita ser criado apenas uma vez em uma única classe.

Para que a invocação de um método de outra classe seja possível, é necessário que a classe utilizada esteja no **mesmo diretório** da classe que contém o método.

Para criar uma nova classe, para somente armazenar métodos, utilize os itens de menu **Projeto**, opção **Adicionar classe...** mude o "Nome" para "Taxes". **Veja as figuras abaixo:**



Obs:

Uma classe e seus métodos podem ser reaproveitados em outros projetos.
Ficará mais fácil se você declará-los com os atributos public e static:
public - deixa a classe visível para chamadas externas;
static - deixa a classe sempre disponível na memória.

Exemplo:

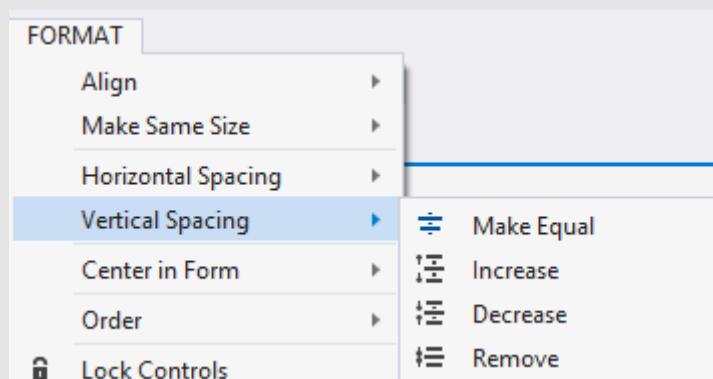
```
public static class Taxas
{
    public static double CalculaINSS(double salBruto)
    {
        if (salBruto <= 965.67) return (salBruto * 0.08);
        else if (salBruto <= 1609.45) return (salBruto * 0.09);
        else if (salBruto <= 3218.90) return (salBruto * 0.11);
        else return (354.08);
    }
}
```

Obs:

Classes são gravadas em arquivos específicos, com extensão ".cs" (salve seu projeto e verifique) e podem ser incluídos posteriormente em outros projetos.

*Alinhamento de Componentes***Obs:**

Para alinhar componentes, a interface do C# dispõe do item de menu “Formatar”, que permite vários tipos de alinhamento. Selecione simultaneamente os componentes a alinhar e escolha o alinhamento desejado.



Ou escolha a formatação da barra de menu



Exercício: Programa Folha de Pagamento

Crie um novo projeto chamado wfaFolhaPagto que atenda às seguintes especificações:

- 1) Ler o nome de um trabalhador, a quantidade de horas trabalhadas, o valor que recebe por hora e a quantidade de dependentes que possui, conforme modelo abaixo:

- 2) Ao ocorrer o click no botão “**Calcular**” o programa deverá calcular e exibir os valores de:

- a. Salário bruto: horas trabalhadas * valor da hora;
- b. INSS (de acordo com a tabela 1): salário bruto * alíquota;
- c. Imposto de renda (de acordo com a tabela 2): ((salário bruto – INSS – desconto por dependente) * alíquota) – dedução;
- d. Salário líquido: salário bruto – INSS – imposto de renda.
- e. Faça a formatação dos valores de saída utilizando máscara como do exemplo:

```
lblSB.Text = sb.ToString("###,###,##0.00");
```

- 3) Com base nas tabelas a seguir: Tabela-1 e Tabela-2, criar a classe “taxas”, contendo os métodos calculaINSS() e calculaIR() para os cálculos correspondentes de INSS e Imposto de Renda.

Tabelas para desenvolvimento dos métodos para cálculo de INSS e Imposto de Renda:

Tabela-1

**TABELA DO INSS
VIGENTE A PARTIR DE 01.02.2009**

SALÁRIO-DE-CONTRIBUIÇÃO (R\$)	A L Í Q U O T A S %
Até 965,67	8%
de 965,68 até 1.609,45	9,%
de 1.609,46 até 3.218,90	11%
acima de 3.218,91	R\$354,08 (valor fixo)

Tabela-2

**TABELA DO IRPF - IRRF
VIGENTE DE 01.01.2009 A 31.12.2009**

Base: [MP 451](#) - DOU de 16.12.2008

Base de Cálculo em R\$ Obs. Valores deduzidos de INSS e dependentes	Alíquota %	Parcela a Deduzir do Imposto em R\$
Até 1.434,59	-	-
De 1.434,60 até 2.150,00	7,5	107,59
De 2.150,01 até 2.866,70	15	268,84
De 2.866,71 até 3.582,00	22,5	483,84
Acima de 3.582,00	27,5	662,94

Dedução por dependente: R\$ 144,20 (cento e quarenta e quatro reais e vinte centavos).

- 4) Incluir o botão “Limpa”, que ao ser pressionado limpa os conteúdos dos campos e põe o foco em txtNome.
- 5) Se o foco estiver em txtNome e for pressionada a tecla Enter o foco deve ir para txtHT e assim sucessivamente.
- 6) Aceitar como tecla válida, a tecla BackSpace.
- 7) O programa não deve aceitar a entrada de quantidade de dependentes negativo ou com vírgula. Nesse caso, o programa deve:
 - exibir uma caixa de mensagem, avisando o usuário sobre o erro;
 - posicionar o cursor no local da quantidade de dependentes **e deixar os números selecionados** para que o valor seja corrigido.

- 8) Aceitar uma única vírgula decimal nos campos valor hora e horas trabalhadas.
Esses campos só podem aceitar números e não podem estar nulos.
- 9) Se o foco estiver em txtDep e for pressionada a tecla Enter, o programa deverá realizar os cálculos.
- 10) A caixa de texto de nome do funcionário deve aceitar apenas letras incluindo as acentuadas, espaço, backspace, Enter e tecla Tab. Não pode estar nulo ou com apenas espaços.
- 11) Incluir o botão sair, que ao ser pressionado solicita a confirmação de saída.

Obs: Para facilitar os próximos projetos, crie uma classe de validação com os métodos de validar entrada numérica, só uma vírgula, etc...

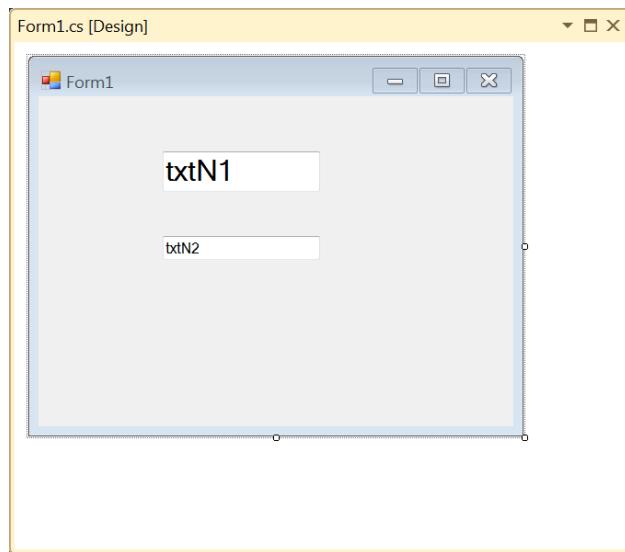
Testar o programa e apresentar ao professor.

AULA 5

Inicialização de Componentes - InitializeComponent()

Vamos iniciar um novo projeto que tem por objetivo verificar uma característica importante de uma linguagem baseada na resposta a eventos, como é o caso da C#.

Inicie um novo projeto com duas caixas de texto (textBox) de nome txtN1 e txtN2, de acordo com o layout abaixo:

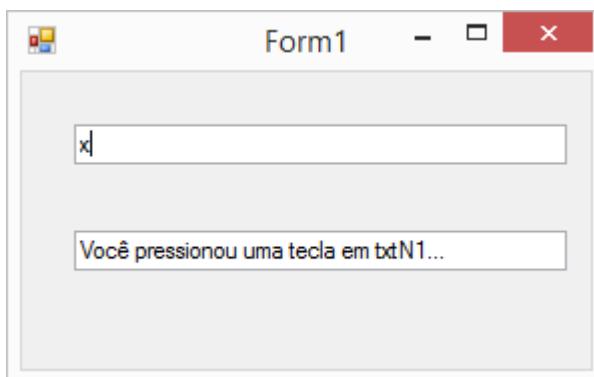


Sem nenhuma programação, por enquanto, execute o programa e digite algo na caixa de texto txtN1, verificando que os caracteres digitados ecoam normalmente na caixa de texto.

Vamos fazer a seguinte programação: Ao ser pressionada qualquer tecla em txtN1, deverá ser exibida em txtN2, a frase "Você pressionou uma tecla em txtN1...". Portanto, devemos programar o tratador do evento txtN1_KeyPress, da maneira indicada na apostila nas aulas anteriores. O código ficará assim:

```
namespace wfatest
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void txtN1_KeyPress(object sender, KeyPressEventArgs e)
        {
            txtN2.Text = "Você pressionou uma tecla em txtN1...";
        }
    }
}
```



Teste e verifique se funciona!

Se quisermos fazer o mesmo procedimento em txtN2 ao invés de em txtN1, parece que bastaria mudar manualmente as instruções, de:

```
private void txtN1_KeyPress(object sender, KeyPressEventArgs e)
{
    txtN2.Text = "Você pressionou uma tecla em txtN1...";
}
```

Para:

```
private void txtN2_KeyPress(object sender, KeyPressEventArgs e)
{
    txtN1.Text = "Você pressionou uma tecla em txtN2...";
}
```

Tente fazer isso e verifique que ocorre um erro! O erro ocorre porque C# registrou e espera o tratador do evento KeyPress em txtN1 e não em txtN2. Veja a figura abaixo:

```
private void InitializeComponent()
{
    this.txtN1 = new System.Windows.Forms.TextBox();
    this.txtN2 = new System.Windows.Forms.TextBox();
    this.SuspendLayout();
    //
    // txtN1
    //
    this.txtN1.Location = new System.Drawing.Point(26, 26);
    this.txtN1.Name = "txtN1";
    this.txtN1.Size = new System.Drawing.Size(246, 20);
    this.txtN1.TabIndex = 0;
    this.txtN1.KeyPress += new System.Windows.Forms.KeyPressEventHandler(this.txtN1_KeyPress);
    //
    // txtN2
    //
    this.txtN2.Location = new System.Drawing.Point(26, 79);
    this.txtN2.Name = "txtN2";
    this.txtN2.Size = new System.Drawing.Size(246, 20);
    this.txtN2.TabIndex = 1;
}
```

Faça a linha de registro de txtN1.KeyPress virar um comentário.
Veja a figura abaixo:

```
/*
this.txtN1.Location = new System.Drawing.Point(26, 26);
this.txtN1.Name = "txtN1";
this.txtN1.Size = new System.Drawing.Size(246, 20);
this.txtN1.TabIndex = 0;
//this.txtN1.KeyPress += new System.Windows.Forms.KeyPressEventHandler(this.txtN1_KeyPress);
*/
```

Execute o programa e note que ele executa normalmente mas não funciona, tanto quando se digita algo em txtN1 ou em txtN2. Por que isso ocorre?

Restaure as condições iniciais corrigindo o programa e teste-o.

Obs:

Se você copiar e colar um tratador de evento e mudar o seu nome, este não será executado, pois não terá sido gerada a linha de código de registro em “InitializeComponent”.

A inclusão dos tratadores de evento no código devem ser feitas da maneira indicada na apostila, pois o editor do C# precisa gerar uma linha de registro para cada tratador de evento incluído no código.

Tratador de evento “TextChanged”

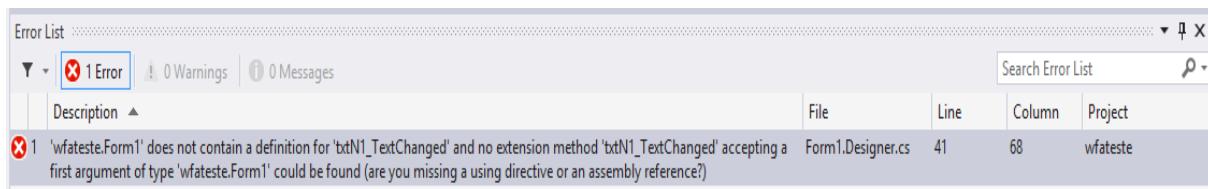
Vamos agora testar outra particularidade de linguagens orientadas a eventos. Vamos programar o tratador de eventos TextChanged da caixa de texto txtN1, incluindo nela a instrução misteriosa txtN1.Text=txtN1.Text + "X". Ou seja, quando houver alguma alteração no texto de txtN1, ao conteúdo anterior de seu texto será acrescida a letra "X". Providencie a programação, execute o programa e verifique o que acontece. Explique o que houve.

A programação deverá ficar como abaixo:

```
private void txtN1_TextChanged(object sender, EventArgs e)
{
    txtN1.Text = txtN1.Text + "X";
}
```

Verificamos que a programação acima provoca um erro (looping infinito) e portanto não a queremos mais. Apague o procedimento e execute o programa.

Xiiii... Parece que ocorreu agora um erro de sintaxe. Como você o corrige?



Quando incluímos no código o tratador de evento KeyPress para a caixa de texto txtN1, o editor incluiu a linha de registro no grupo InitializeComponent. Note que isso ocorreu também para o tratador de evento TextChanged, que também estava sendo usado.

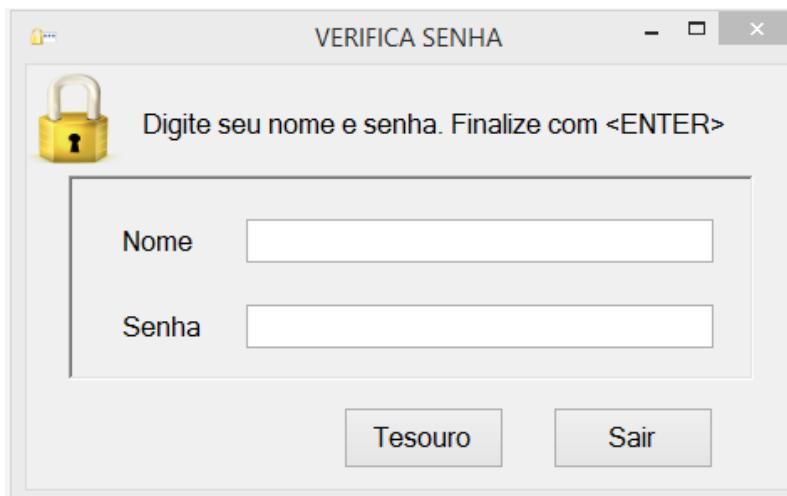
```
// txtN1
//
this.txtN1.Location = new System.Drawing.Point(26, 26);
this.txtN1.Name = "txtN1";
this.txtN1.Size = new System.Drawing.Size(246, 20);
this.txtN1.TabIndex = 0;
this.txtN1.TextChanged += new System.EventHandler(this.txtN1_TextChanged);
this.txtN1.KeyPress += new System.Windows.Forms.KeyPressEventHandler(this.txtN1_KeyPress);
```

Pelo motivo de existir essa linha de registro para cada tratador de evento no programa, se você apagar um tratador de evento que você não precisa mais, deverá apagar também a linha de registro no grupo InitializeComponent, caso contrário ocorrerá erro na compilação do programa. Ver figura abaixo.

```
// txtN1
//
this.txtN1.Location = new System.Drawing.Point(26, 26);
this.txtN1.Name = "txtN1";
this.txtN1.Size = new System.Drawing.Size(246, 20);
this.txtN1.TabIndex = 0;
//this.txtN1.TextChanged += new System.EventHandler(this.txtN1_TextChanged);
this.txtN1.KeyPress += new System.Windows.Forms.KeyPressEventHandler(this.txtN1_KeyPress);
```

Exercício 1: Projeto Senha

Abrir um novo projeto de nome “wfaSenha” e fazer um programa com interface parecida com a figura abaixo que solicita um nome e a senha ao usuário. Se a senha for correta, o programa exibe a mensagem: “Bem Vindo Senhor **nome**!”. Caso contrário, o programa exibe: “Caia Fora!...”.



O projeto deve obedecer às especificações a seguir:

- 1) Os caracteres digitados na senha aparecem como “*”.
 - 2) A sequência de caracteres do nome e da senha são finalizados pela tecla “Enter”.
 - 3) Não pode haver espaços em branco antes e nem depois do nome do usuário – usar o método “trim”.
 - 4) A senha correta é “1234”.
 - 5) A caixa de texto txtSenha deve aceitar a tecla backspace. Seu valor ASC é 8.
 - 6) Todos os caracteres digitados no nome do usuário devem ficar em maiúsculo – Para a programação, podemos levar em consideração que o valor ASCII da letra “A”=65, da letra “a”=97, que existem 26 letras no alfabeto e que a distância entre “a” e “A” é de 32 caracteres.
- Vide o trecho de programa abaixo e a partir dele, crie o método toMaiusc() que ao receber um caractere minúsculo, o converte para maiúsculo.

```
private void txtNome_KeyPress(object sender, KeyPressEventArgs e)
{
    char c;
    int i;
    i = (int)e.KeyChar;
    if (i >= 97 && i <= 122)
    {
        c = e.KeyChar;
        c = (char)((int)c - 32);
        e.KeyChar = c;
    }
    if (e.KeyChar == (Char)13)
    {
        txtNome.Text = txtNome.Text.Trim();
        if (txtNome.Text.Trim().Length == 0)
        {
            MessageBox.Show("Preenchimento Obrigatório");
            txtNome.Focus();
            return;
        }
        txtSenha.Focus();
    }
}
```

A programação para verificação da senha pode ser algo como o seguinte:

```
private void txtSenha_KeyPress(object sender, KeyPressEventArgs e)
{
    if (e.KeyChar != (Char)13)
    {
        senha = senha + e.KeyChar;
        e.KeyChar = '*';
    }
    else
    {
        if (senha == "1234")
        {
            MessageBox.Show("Bem vindo Sr. " + txtNome.Text);
        }
        else
        {
            MessageBox.Show("Caia fora!...");
        }
    }
}
```

- 7) O usuário pode fazer até três tentativas p/ acertar a senha. A cada tentativa, o programa informa que a senha informada foi incorreta e quantas tentativas faltam. Se o usuário acertar a senha ou se ocorrerem mais de 3 tentativas, o programa deve encerrar a execução – use o método close().
- 8) Inclua o botão “btnSair” com text & Sair que ao receber o click do mouse solicita confirmação. Para isso, use o tratador de evento FormClosing.

Verifique o trecho de código abaixo:

```
private void btnSair_Click(object sender, EventArgs e)
{
    Close();
}

private void frmLogin_FormClosing(object sender, FormClosingEventArgs e)
{
    if (MessageBox.Show("Deseja mesmo sair do sistema?", "P e r g u n t a",
        MessageBoxButtons.YesNo, MessageBoxIcon.Question) == DialogResult.No)
    {
        e.Cancel = true;
    }
}
```

- 9) Inclua o botão “btnTesouro” com text “Acesso ao Tesouro”. Esse botão está inicialmente invisível e só fica visível quando o usuário acerta a senha. Ao

receber o Click, o botão exibe o formulário “frmTesouro”. Após isso, o botão volta a ficar invisível. Veja o código abaixo:

```
private void btnTesouro_Click(object sender, EventArgs e)
{
    Form tstTesouro = new frmTesouro();
    tstTesouro.Show();
    btnTesouro.Visible = false;
}
```

Obs.

Se quiser exibir o formulário na forma Modal, use tstTesouro.ShowDialog().

10) Inclua o formulário frmTesouro : Project / Add Windows Form / Windows Form / Ok

11) Inclua na propriedade BackGroundImage de frmTesouro, a figura do tesouro de acordo com layout da apostila e configure a propriedade BackGroundImageLayout para Stretch.

12) Se o usuário errar a senha 3 vezes, o programa deve fechar sem solicitar confirmação. Neste caso, você pode programar o tratador de evento Form.Closing para exibir a mensagem de confirmação somente se o número de tentativas não foi excedido.

13) Quando o usuário acertar a senha, as caixas de texto txtNome e txtSenha deverão ficar desabilitadas, pois estas já terão cumprido suas funções.

Layout final do projeto “Programa Senha”:



Obs.

Um programa em C ou C# ou seus derivados, sempre inicia o processamento por um módulo de nome "main". É esse módulo que define o primeiro formulário que será instanciado quando começar o processamento. O nome desse primeiro formulário está especificado em "Application.Run(new nomefrm());", que se pode visualizar quando açãonamos o item "Program.cs" da janela "Gerenciador de Soluções".

The screenshot shows the Visual Studio interface. On the left is the Solution Explorer pane, which lists a single project named 'wfaSenha' containing files like App.config, frmLogin.cs, frmLogin.resx, frmTesouro.cs, and Program.cs. The 'Properties' tab is selected. On the right is the main code editor window displaying the 'Program.cs' file. The code defines a static class 'Program' with a Main() method that runs a Windows application. The code is as follows:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace wfaSenha
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new frmLogin());
        }
    }
}

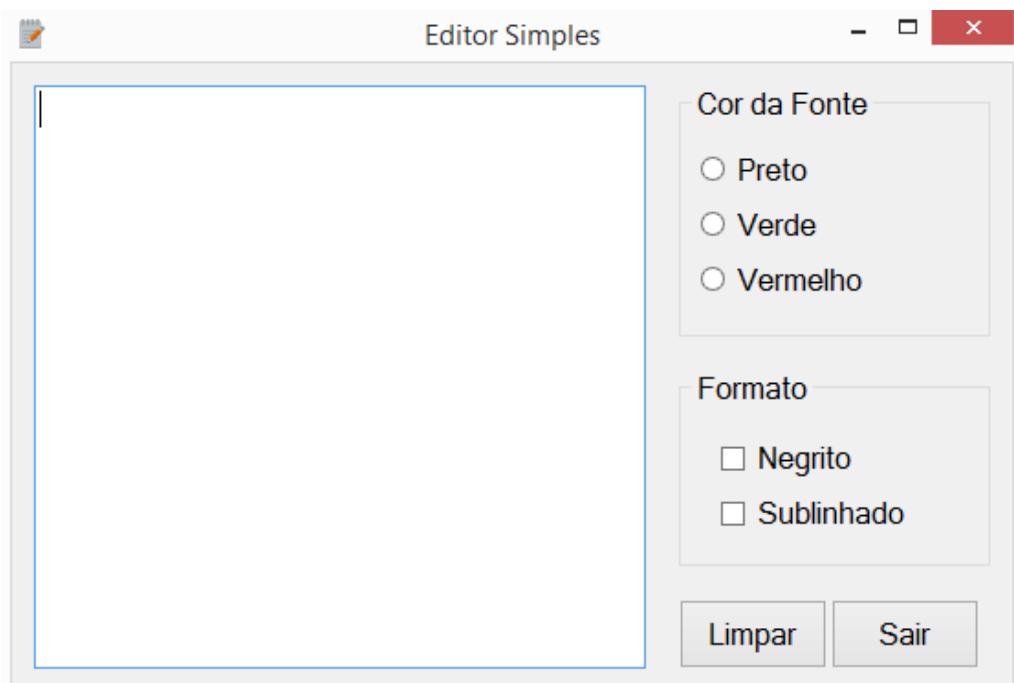
```

Testar o programa e apresentar ao professor.

Exercício2: Programa Editor

Tem por objetivo a utilização de botões de opção (RadioButton) e caixa de verificação (CheckBox).

- 1) Inicie um projeto chamado wfaEditor e elabore a interface do programa de acordo com o layout abaixo:



Obs.:

Para adequar a caixa de texto txtTexto igual à da figura, fazer a propriedade MultiLine=True.

- 2) No evento checkedChanged das radiobuttons mude a cor de acordo com a opção selecionada.

Exemplo:

```
private void rdbPreto_CheckedChanged(object sender, EventArgs e)
{
    txtTexto.ForeColor = Color.Black;
}
```

- 3) No tratador de evento checkedChange da opção Negrito altere o fonte da caixa de texto.

Exemplo:

```
private void chkNegrito_CheckedChanged(object sender, EventArgs e)
{
    if (chkNegrito.Checked && !chkSublinhado.Checked)
        txtTexto.Font = new Font(txtTexto.Font, FontStyle.Bold);

    if (chkNegrito.Checked && chkSublinhado.Checked)
        txtTexto.Font = new Font(txtTexto.Font, FontStyle.Bold | FontStyle.Underline);

    if (!chkNegrito.Checked && !chkSublinhado.Checked)
        txtTexto.Font = new Font(txtTexto.Font, FontStyle.Regular);

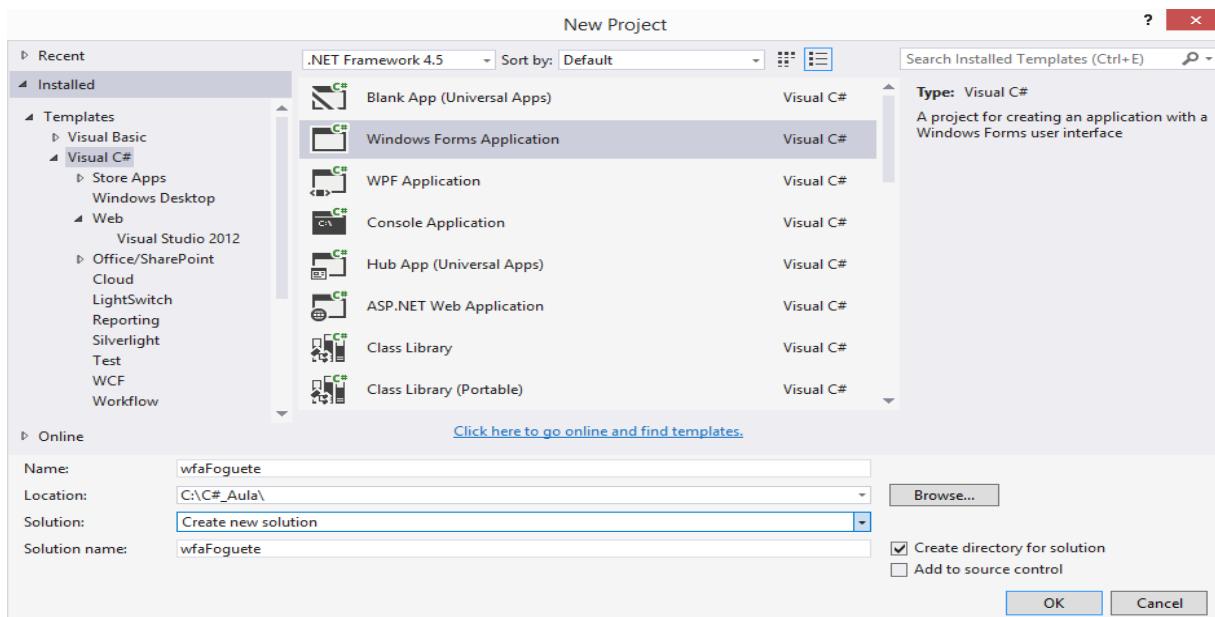
    if (!chkNegrito.Checked && chkSublinhado.Checked)
        txtTexto.Font = new Font(txtTexto.Font, FontStyle.Underline);
}
```

- 4) Implementar a programação para o tratador de evento CheckedChanged de chkSublinhado.
- 5) Incluir um botão Sair que ao ser clicado finaliza o programa.
- 6) Incluir um botão Limpar que ao ser clicado limpa a caixa de texto, marca a opção de cor preta e limpa as seleções de fonte.

AULA 6

Exercício – Projeto Foguete

- Iniciar um novo projeto de nome wfaFoguete.



- Alterar as seguintes propriedades do formulário:
 - Na propriedade BackColor do formulário, digite os valores **0;180;255** ou **selecione a cor da lista de opções**. Esses são valores RGB que darão uma tonalidade próxima da BlueSky.
 - Inserir um ícone no formulário
- Para conter a figura do foguete inclua um PictureBox de nome picFoguete, cuja propriedade Image é a figura foguete.JPG que está em nossa pasta de trabalho. Configure a propriedadeSizeMode como StretchImage. Essa propriedade faz com que a figura acompanhe o tamanho do container PictureBox.
- Incluir 2 botões de comando de nome btnDispara e btnPara.

A interface ficará parecida com a figura abaixo



Comando de Iteração (For)

Incluir o código abaixo nos botões correspondentes.

```
private void btnDispara_Click(object sender, EventArgs e)
{
    int i;
    this.Cursor = Cursors.WaitCursor;
    for (i = 1; i < 30000; i++)
    {
        picFoguete.Top = picFoguete.Top - 1;
        picFoguete.Left = picFoguete.Left + 1;
    }
    this.Cursor = Cursors.Default;
}

private void btnPara_Click(object sender, EventArgs e)
{
    Close();
}
```

Analise o código e execute o programa. Note o seguinte:

- a) Antes de executar a malha de 30.000 vezes, mudamos a forma do cursor para indicar que um processo consumidor de tempo está em andamento. Quando terminar de executar a malha, a forma do cursor volta ao normal.
- b) A cada execução da malha, a propriedade Top do foguete, que é a distância do objeto até o topo do formulário, é diminuída de 1 pixel. E também a propriedade left, que a distância do objeto até a borda esquerda do formulário, é aumentada de 1 pixel.
- c) Execute o programa e tente pará-lo, dando um click no botão btnPara. Porque nada acontece?

Explique por que o foguete não parou imediatamente ou por que nada mais funciona por certo intervalo de tempo.

O que acontece é que quando você dá um click no botão btnDispara, o programa começa a executar uma malha de 30000 iterações e nada mais funcionará enquanto a malha que está sendo processada não terminar sua execução e “soltar” o processador.

Por esse motivo, deve-se pensar muito bem antes de usar malhas em programas com orientação a eventos. Muito provavelmente, o usuário do programa vai pensar que o computador travou ou então quebrou!. Talvez pense até que seu programa não é muito bom (que absurdo!...).

Uma solução não muito recomendada para esse problema é o uso do método “DoEvents()”. **Veja o exemplo abaixo:**

```
private void btnDispara_Click(object sender, EventArgs e)
{
    int i;
    this.Cursor = Cursors.WaitCursor;
    for (i = 1; i < 30000; i++)
    {
        Application.DoEvents(); //***** uso de DoEvents();
        picFoguete.Top = picFoguete.Top - 1;
        picFoguete.Left = picFoguete.Left + 1;
    }
    this.Cursor = Cursors.Default;
}
```

Utilizando o método DoEvents(), ao ocorrer um primeiro click do mouse na interface, esta passará a responder aos próximos clicks. Teste-a.

Componente Timer

Uma solução melhor para esse tipo de problema (neste caso particular), é o uso de um temporizador, que fará o trabalho sem “prender” o processador. O temporizador é um componente que processa o código contido no tratador de evento **“nomeTimer_Tick” a cada intervalo.**

- 5) Inclua no formulário o controle Timer e configure sua propriedade Interval = 1 milisegundo. Deixe sua propriedade Enabled=False, que deverá ficar True quando ocorrer o click no botão btnDispara. Quando quisermos que o foguete pare, daremos um click no botão btnPara, que desabilitará o temporizador.

Implemente o código abaixo e teste-o.

```
private void btnDispara_Click(object sender, EventArgs e)
{
    timer1.Enabled = true;
}

private void btnPara_Click(object sender, EventArgs e)
{
    timer1.Enabled = false;
}

private void timer1_Tick(object sender, EventArgs e)
{
    picFoguete.Top = picFoguete.Top - 1;
    picFoguete.Left = picFoguete.Left + 1;
}
```

- 6) Vamos incrementar um pouco mais nosso programa do foguete, para que o foguete ao subir, choque-se com um avião que vem voando em sua direção. O procedimento pode ser algo assim:

- O foguete vai subindo e diminuindo de tamanho;
- O avião vem se aproximando e aumentando de tamanho;

Teste o código abaixo:

```
private void timer1_Tick(object sender, EventArgs e)
{
    picFoguete.Top = picFoguete.Top - 1;
    picFoguete.Left = picFoguete.Left + 1;
    picFoguete.Height = picFoguete.Height - 1;
    picFoguete.Width = picFoguete.Width - 1;

    picAviao.Top = picAviao.Top + 1;
    picAviao.Left = picAviao.Left + 1;
    picAviao.Height = picAviao.Height + 1;
    picAviao.Width = picAviao.Width + 1;

}
```

7) Ao testarmos o programa, temos a impressão que avião aumenta de tamanho muito rápido. E parece que o tamanho do foguete diminui muito rápido. Vamos melhorar um pouco isso, maximizando a tela e diminuindo a velocidade de variação no tamanho dos componentes do avião e do foguete. Além disso vamos posicionar o foguete na posição inferior da tela.

Para maximizar a tela usaremos o código abaixo, quando o formulário for carregado na memória. Neste caso usamos o tratador do evento “frmFoguete_Load”. Esse tratador de evento é utilizado quando se inicializam variáveis, quando se faz a configuração inicial de componentes, etc. A instrução que maximiza tela é a seguinte:

```
private void frmFoguete_Load(object sender, EventArgs e)
{
    this.WindowState = FormWindowState.Maximized;
}
```

Para colocar o componente do foguete na posição inferior da tela, usaremos:

```
private void frmFoguete_Load(object sender, EventArgs e)
{
    this.WindowState = FormWindowState.Maximized;
    picFoguete.Top = this.ClientSize.Height - picFoguete.Height;
}
```

Para tornar mais lenta a velocidade de variação no tamanho dos componentes, nós utilizaremos o operador módulo “%”, executando as instruções que provocam a alteração de tamanho só quando a propriedade “Top” for múltipla de 7. Também vamos incrementar a velocidade horizontal do avião e do foguete em 2 pixels a cada ciclo do temporizador.

Acompanhe e implemente o código a seguir:

```

private void timer1_Tick(object sender, EventArgs e)
{
    picFoguete.Top = picFoguete.Top - 1;
    picFoguete.Left = picFoguete.Left + 2;

    if (picFoguete.Top % 7 == 0)
    {
        picFoguete.Height = picFoguete.Height - 1;
        picFoguete.Width = picFoguete.Width - 1;
        picAviao.Height = picAviao.Height + 1;
        picAviao.Width = picAviao.Width + 1;
    }

    picAviao.Top = picAviao.Top + 1;
    picAviao.Left = picAviao.Left + 2;
}

```

- 8) E agora que maximizamos o formulário, vamos fazer o choque ocorrer no meio da tela. Quando houver certa distância mínima entre os dois, eles param de se movimentar, desaparecem, e surge a imagem de uma explosão.

Inclua na interface outra PictureBox de nome picExplosao com a imagem explosao.JPG, que está em nossa pasta de trabalho. Configurar a propriedade Visible=false e a propriedadeSizeMode = StretchImage.

Mude o intervalo do temporizador para 50 ms.

Faça os ajustes necessários e teste o programa.

A interface e o código devem ficar mais ou menos assim:



```

private void timer1_Tick(object sender, EventArgs e)
{
    if (choque == true)
        picExplosao.Visible = !picExplosao.Visible; // troca o valor boolean
    else
    {
        picFoguete.Top = picFoguete.Top - 1;
        picFoguete.Left = picFoguete.Left + 2;
        if (picFoguete.Top % 7 == 0)
        {
            picFoguete.Height = picFoguete.Height - 1;
            picFoguete.Width = picFoguete.Width - 1;
            picAviao.Height = picAviao.Height + 1;
            picAviao.Width = picAviao.Width + 1;
        }

        picAviao.Top = picAviao.Top + 1;
        picAviao.Left = picAviao.Left + 2;

        if ((Math.Abs(picFoguete.Top - picAviao.Top) < 60) && (Math.Abs(picFoguete.Left - picAviao.Left) < 60))
        {
            choque = true; //No próximo ciclo do Timer, a picExplosao ficará visível.
            picFoguete.Visible = false; //No próximo ciclo, foguete e aviao invisíveis.
            picAviao.Visible = false;
        }
    }
}
}

```

- 9) Assim que ocorrer o choque, a figura da explosão vai diminuindo até desaparecer e um sobrevivente salta de pára-quedas e vai descendo até desaparecer.
- 10) Para grande azar do sobrevivente o choque ocorre sobre o mar e ele não sabe nadar. Ao chegar à água, surge “glub glub glub...”.

Obs.: Para alternar a visibilidade das pictures "glub... glub..." (ficar piscante), podemos incluir em local adequado do procedimento o seguinte código:

```

picGlub1.Visible = !picGlub1.Visible; //inverte a prop. visible de pictGlub1
picGlub2.Visible = !picGlub1.Visible; //visibilidade de pictGlub2 é oposta de pictGlub1.

```

- 11) Incluir o botão “Acelera...”, que a cada click que recebe, acelera a velocidade horizontal do avião em 1 pixel a cada ciclo do temporizador.
- 12) Os botões devem ficar alinhados na parte inferior esquerda.
- 13) Incluir o botão “Fim”, que ao receber o click solicita confirmação do encerramento.
- 14) Colocar um ícone no projeto.

OBS.

Se você quiser fazer um componente ficar atrás ou na frente de outro, como por exemplo o Sol atrás do avião, de um click com o botão direito do mouse e escolha a opção adequada.

Testar o programa e apresentar ao professor.

AULA 7

Criação e Utilização de Objetos

Objeto

- Dentro da terminologia das linguagens de programação, um **objeto** passa a existir a partir de um "**molde**". Este "molde", definido como **classe do objeto**, define os limites, seus atributos e suas funções. Podem ser criados vários **objetos ou instâncias de uma classe**.
- Apesar de existirem diferentes tipos de objetos, eles compartilham duas características principais: todos possuem um **estado** (conjunto de propriedades do objeto) e um **comportamento** (as ações possíveis sobre o objeto).
- Para armazenar o **estado**, um objeto utiliza-se de uma ou diversas **variáveis**.
- O **comportamento** do objeto é definido pelo conjunto de **métodos** que ele possui.

Classes

- Os **objetos** são criados a partir das **classes**, sendo **instâncias** em memória que mantêm seus valores de maneira individual.
- A criação de uma **classe** deve anteceder o uso de um objeto.
- Uma **classe é um molde**, um modelo, um protótipo a partir do qual os objetos podem ser criados.

Criação de uma Classe

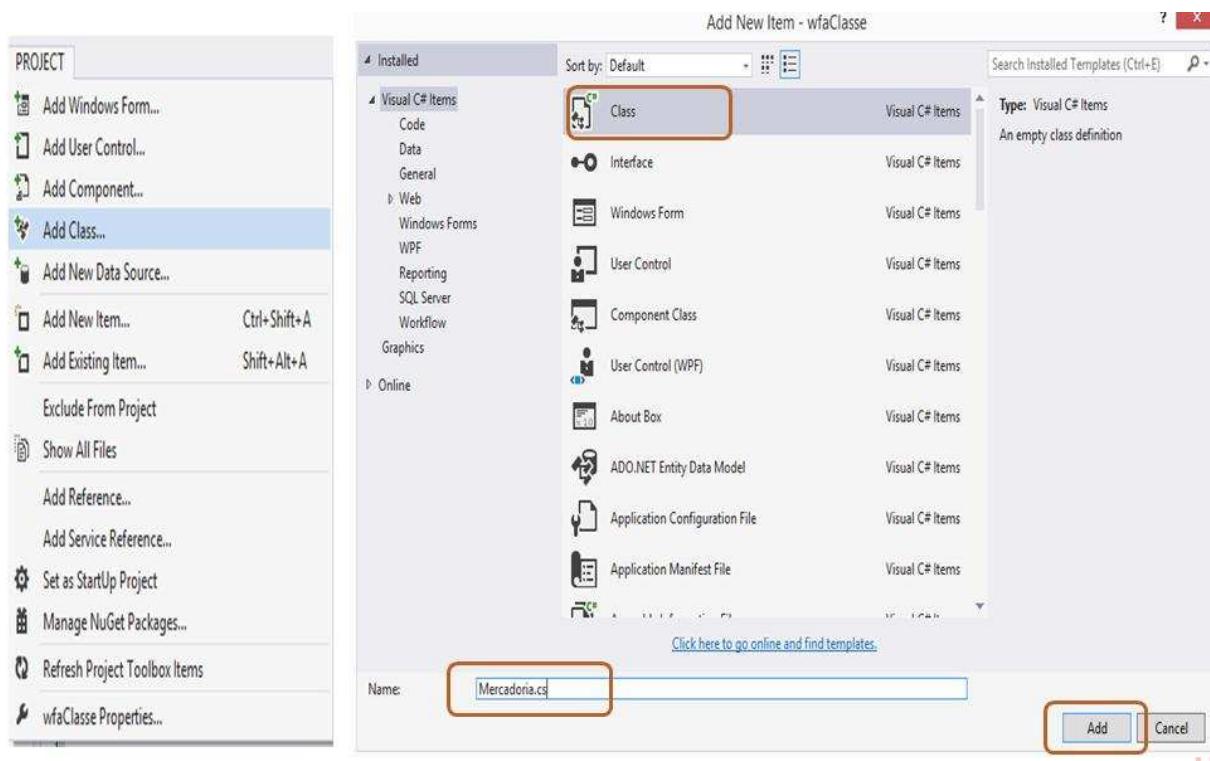
- Uma classe é composta basicamente por declaração de variáveis e implementação de métodos.
- Vamos criar diversos exemplos acrescentando, pouco a pouco, os diversos conceitos.

Adicionando uma Classe

Os exemplos serão desenvolvidos numa classe que manipula informações sobre um produto qualquer.

- 1) Iniciar um novo projeto Windows Forms de nome wfaClasse.

- 2) Adicionar uma Classe ao projeto. No item Projeto do menu selecione Adicionar classe... e na janela Adicionar selecione o template Classe. De um nome para a classe e click no Adicionar.



Note que o editor do C# incluiu uma guia para a classe.

```

Mercadoria.cs*  Form1.cs      Form1.cs [Design]
wfaMercadoria

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace wfaMercadoria
{
    public class Mercadoria
    {
    }
}

```

The screenshot shows the 'Mercadoria.cs' code editor. The code defines a class named 'Mercadoria' within a namespace 'wfaMercadoria'. The code includes standard using statements for System, System.Collections.Generic, System.Linq, System.Text, and System.Threading.Tasks.

Declaração de variáveis de uma classe

Modifique a classe para ficar igual ao código abaixo:

```

Mercadoria.cs*  X  Form1.cs      Form1.cs [Design]
C# wfaMercadoria                                     wfaMercadoria.Mercadoria

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace wfaMercadoria
{
    public class Mercadoria
    {
        public string nome;
        public double preco;
    }
}

```

Descrição do código:

```

1. public class Mercadoria
2. {
3.     public string nome;
4.     public double preco;
5. }

```

- Linha1: indica que a classe **Mercadoria** é pública (public), isto é, pode ser utilizada por outras classes, independentemente de pertencerem ao mesmo pacote.
- Linhas 3 e 4: declaram as variáveis **nome** e **preco** que são conhecidas como **variáveis de instância**, pois serão instanciadas pelos objetos. Todos os objetos criados a partir dessa classe contêm essas variáveis. Elas podem ser acessadas por meio de um objeto usando a seguinte sintaxe:

nome-do-objeto.nome-da-variável

- Supondo um objeto chamado **produto**, o acesso às variáveis seria **produto.nome** e **produto.preco**.
- Essa classe não é executável. Essa classe será usada para criar objetos que serão usados em outras classes.

Declaração de Métodos de uma Classe

- Da mesma forma que as variáveis de instância, ao criar um objeto, ele recebe os métodos presentes na classe.
- A partir da definição de uma classe podem ser criados inúmeros objetos que passam a conter o estado e o comportamento declarados na classe.
- Esta é a base de funcionamento de toda a linguagem orientada a objetos.
- O exemplo abaixo apresenta a classe **Mercadoria** dotada do método **atualizaPreco** cuja função é atualizar o valor do preço de um produto de acordo com o valor de porcentagem passado como argumento.

```
class Mercadoria
{
    public string nome;
    public double preco;

    public void atualizaPreco(double percent)
    {
        preco = preco + (preco * percent / 100);
    }
}
```

- Para utilizar o método de um objeto, a sintaxe é a mesma usada para as variáveis:

nome-do-objeto.nome-do-metodo()

Obs:

- O que diferencia o uso do método é a presença do abre e fecha parênteses no final de seu nome.
 - Quando os parênteses estão vazios, quer dizer que o método não recebe nenhum argumento.

Utilização de Objetos

Para utilizar um objeto, existem três partes a serem consideradas:

1. A declaração do objeto: segue o mesmo padrão de declarações para tipos primitivos, isto é, **nome-do-tipo nome-da-variável**, como por exemplo a declaração: string nome. Neste caso estamos dizendo que a variável nome é do tipo string.

Para declarar objeto, é usada a seguinte sintaxe:

nome-da-classe nome-do-objeto

No caso, como desejamos gerar um objeto a partir da classe **Mercadoria**, criada anteriormente, a sintaxe será:

Mercadoria produto1

Mercadoria se refere ao nome da classe a partir da qual o objeto **produto1** será criado. Vale a pena ressaltar que a declaração não cria o objeto em si, trata-se apenas de uma declaração dizendo que **produto1** é um objeto do tipo **Mercadoria**.

2. A instanciação do objeto: corresponde à criação do objeto pela alocação de memória para armazenar informações sobre ele, semelhante ao que ocorre na declaração de uma variável qualquer, isto é, são reservados endereços de memória para armazenar os dados correspondentes.

Para realizar a instanciação de um objeto qualquer, é usado o operador **new**.

Seguindo o exemplo, a sintaxe será:

nome-do-objeto = new ("inicialização-do-objeto")

3. A inicialização do objeto: corresponde ao processo de definir valores iniciais às variáveis do objeto. Conforme apresentado no item anterior, a inicialização é precedida pelo operador **new**.

Para inicializar um objeto, é usado o **método construtor**, estudado mais à frente.

Por enquanto o método construtor será usado em sua forma default (como mesmo nome da classe sem nenhum parâmetro associado) e as variáveis terão seus valores iniciais conforme definidos na classe.

Quando as variáveis não possuem um valor inicial definido na classe, é considerado "0" (zero) para variáveis numéricas, "false" para variáveis boolean e "null" para Strings e outros objetos.

Os procedimentos de declaração, instanciação e inicialização de objetos podem ser realizados em uma ou duas linhas de código. Segue a sintaxe das duas formas possíveis:

Em uma única linha:

nome-da-classe nome-do-objeto = new nome-do-construtor();

Em duas linhas:

```
nome-da-classe nome-do-objeto;
nome-do-objeto = new nome-do-construtor();
```

Considerando a classe Produto1, a sintaxe será:

Mercadoria produto1 = new Mercadoria();

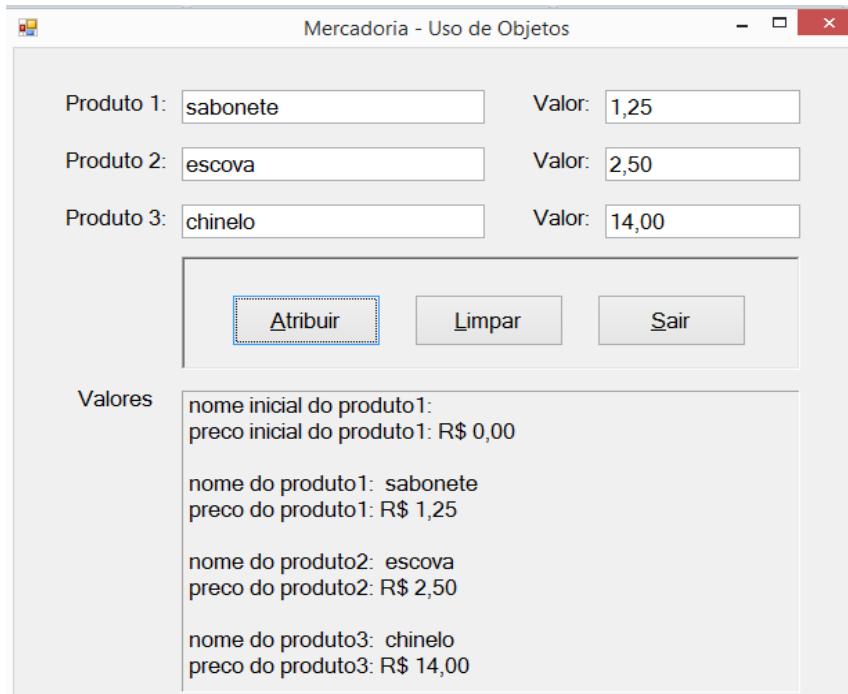
ou

Mercadoria produto1;
produto1 = new Mercadoria();

Exercício - “Uso de Objetos”

Fazer o programa a seguir visando a utilização de objetos da classe **Mercadoria**. Para que este programa possa ser executado com sucesso, as classes devem estar armazenadas na mesma pasta.

- 1) Iniciar um novo projeto chamado wfaMercadoria. A interface deve ser parecida com a figura abaixo:



- 2) Incluir uma classe chamada Mercadoria, definindo 2 variáveis e 1 método.

```

class Mercadoria
{
    public string nome;
    public double preco;

    public void atualizaPreco(double percent)
    {
        preco = preco + (preco * percent / 100);
    }
}

```

- 3) No botão Atribuir instanciar os objetos produto1, produto2 e produto3.
- Mostrar os valores do objeto produto1 antes de qualquer atribuição.
 - Atribuir os valores nome e valor aos respectivos objetos.
 - Mostrar os valores atribuídos para o objeto no label lblSaida.

```

private void btnAtribuir_Click(object sender, EventArgs e)
{
    Mercadoria produto1 = new Mercadoria();

    /* nome e preço ainda não inicializados */
    lblSaida.Text = ("nome inicial do produto1: " + produto1.nome + "\n" +
                    "preço inicial do produto1: " +
                    produto1.preco.ToString("R$ #,##0.00") + "\n\n");

    produto1.nome = txtProd1.Text; // ***** agora inicializou...
    produto1.preco = double.Parse(txtVal1.Text);
    lblSaida.Text += ("nome do produto1: " + produto1.nome + "\n" +
                      "preço do produto1: " +
                      produto1.preco.ToString("R$ #,##0.00") + "\n\n");
}

```

Faça o mesmo para o objeto produto2 e produto3

Obs.:

Podemos instanciar os objetos produto1, produto2 e produto3, dentro do botão Atribuir. Nesse caso, os objetos só tem existência enquanto estiver sendo processado o procedimento acima.

Se quisermos utilizá-los em outros locais, devemos instanciá-los globalmente.

```

public frmMercadoria()
{
    InitializeComponent();
}

Mercadoria produto1 = new Mercadoria();
Mercadoria produto2 = new Mercadoria();
Mercadoria produto3 = new Mercadoria();

```

- 4) Incluir o botão btnReajuste que ao ser pressionado executa o código abaixo. Note que ao digitar o ponto (.) após o objeto produto1, que é uma instância *agora global*, da classe Mercadoria, o ambiente c# exibe o método atualizaPreco, pois este está disponível naquela classe para uso, caso você deseje.

```

private void btnReajuste_Click(object sender, EventArgs e)
{
    lblSaida.Text += ("nome do produto1: " + produto1.nome + "\n" +
                      "preço inicial do produto1: " +
                      produto1.preco.ToString("#,##0.00") + "\n");

    produto1.atualizaPreco(20); //reajuste de 20 sobre o preço.

    lblSaida.Text += ("novo preço do produto1: " +
                      produto1.preco.ToString("#,##0.00") + "\n\n");
}

```

- 5) Incluir a caixa de texto txtReajuste, onde será digitado o valor do reajuste do preço dos produtos.
- 6) Modificar a programação do botão btnReajuste que ao ser pressionado fará o reajuste do valor dos três produtos, com o valor do reajuste de txtReajuste, usando o método atualizaPreco. Os valores reajustados deverão ser mostrados nas caixas de texto do formulário. Somente serão mostrados no label lblSaida se for pressionado o botão Atribuir.
- 7) Validar todos campos.
- Nomes dos produtos não podem estar nulos
 - Valores só podem ser preenchidos com números e uma vírgula.
 - Reajuste só pode ser preenchido com números e uma vírgula.
- 8) Verificar a tabulação dos campos. Ao iniciar o projeto o cursor deve estar posicionado no campo nome do produto1. Depois o cursor deve ir para o campo valor1 e assim por diante.
- 9) Limpar o label de saída antes de cada atribuição.

Exemplo da interface do programa, com valores após reajuste de 10%

Mercadoria - Uso de Objetos

Produto 1:	<input type="text" value="sabonete"/>	Valor:	<input type="text" value="110,00"/>
Produto 2:	<input type="text" value="escova"/>	Valor:	<input type="text" value="220,00"/>
Produto 3:	<input type="text" value="chinelo"/>	Valor:	<input type="text" value="330,00"/>
Valor do Reajuste em %	<input type="text" value="10"/>	<input type="button" value="Reajuste"/>	
<input type="button" value="Atribuir"/> <input type="button" value="Limpar"/> <input type="button" value="Sair"/>			

Valores

nome inicial do produto1: sabonete preco inicial do produto1: R\$ 121,00
nome do produto1: sabonete preco do produto1: R\$ 100,00
nome do produto2: escova preco do produto2: R\$ 200,00
nome do produto3: chinelo preco do produto3: R\$ 300,00

Testar o programa e apresentar ao professor.

AULA 8

Herança

A herança ocorre quando uma classe (conhecida como classe derivada) herda as características (variáveis e métodos) definidas em uma outra classe, especificada como sua classe base (ancestral ou superclasse).

Ao herdar as características de outra, uma classe pode implementar partes específicas não contempladas na classe original (superclasse), tornando-se especializada em algum processo.

A técnica da herança é muito utilizada, pois possibilita o compartilhamento ou reaproveitamento de recursos definidos anteriormente em outra classe.

A sintaxe para indicação da herança é a seguinte:

```
class ClasseDerivada : ClasseBase
{
    ...
}
```

Exemplo:

```
class Pessoa //classe Base
{
    private string nome;

    public void setNome(string nome)
    {
        this.nome = nome;
    }

    public string getNome()
    {
        return (nome);
    }
}
```

A classe **Pessoa** possui:

- Uma variável de instância **nome** (encapsulada, pois o qualificador de acesso **private** somente permite acesso à mesma por métodos da própria classe);
- Dois métodos, **setNome** e **getNome**, para armazenamento e obtenção de valor da variável **nome**.

Vamos criar, a seguir, a classe **PessoaJuridica**, que herdará as características da classe **Pessoa** e ainda possui sua variável encapsulada **cnpj**, além dos métodos **setCnpj** e **getCnpj**.

```
class PessoaJuridica : Pessoa
{
    private string cnpj;

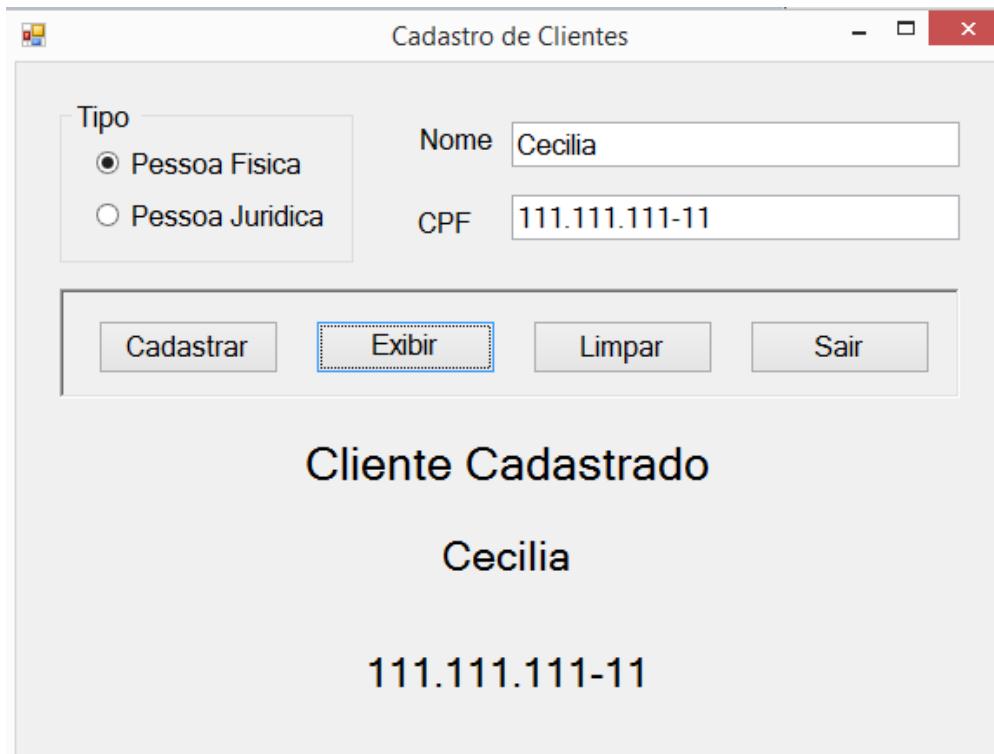
    public void setCnpj(string cnpj)
    {
        this.cnpj = cnpj;
    }

    public string getCnpj()
    {
        return (cnpj);
    }
}
```

Após definirmos as classes **base** e **derivada**, vamos criar o objeto **empresa**, instanciando a classe **PessoaJuridica**. O objeto **empresa** possuirá todas as características da classe **PessoaJuridica**, além das recebidas da classe **Pessoa**, por herança.

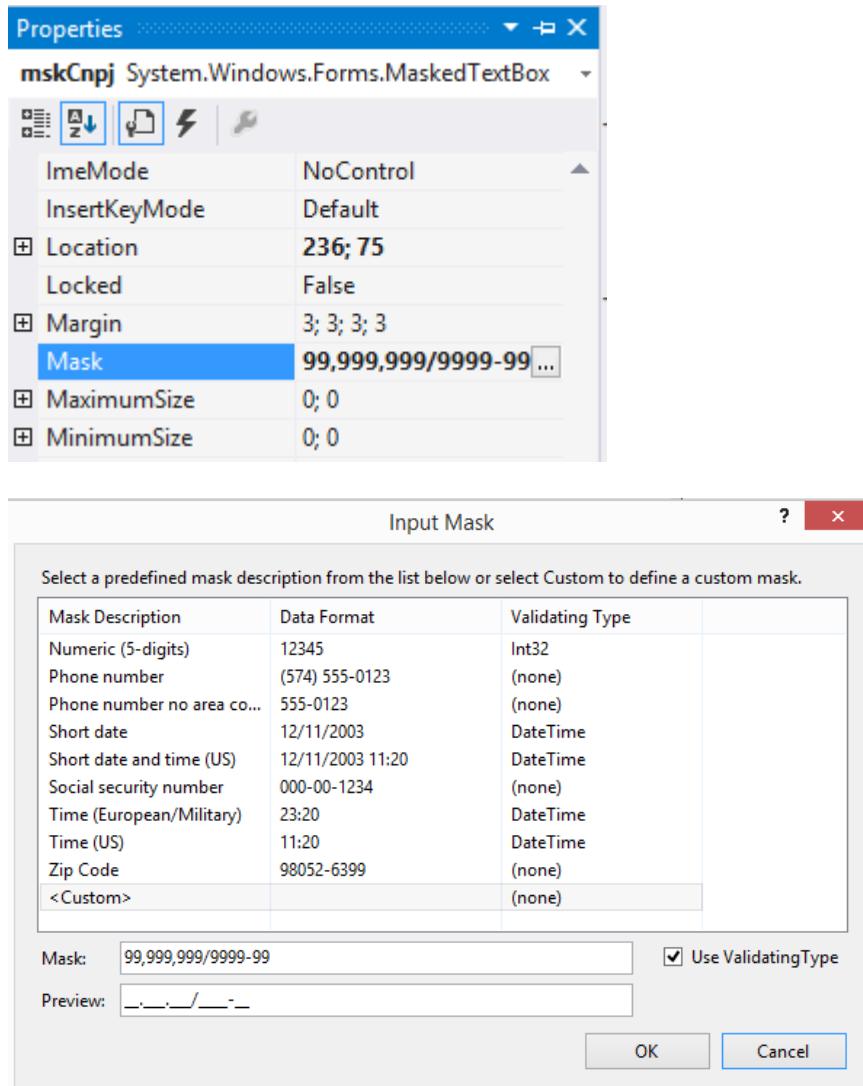
Exercício: Projeto Herança

- 1) Iniciar um projeto chamado wfaHeranca, com a interface parecida com a figura abaixo:



Obs: Vamos usar o componente maskedTextBox para entrada do CPF e CNPJ.

Na propriedade Mask configure a formatação dos campos.



- 2) Incluir a classe base chamada Pessoa com uma variável de instância **nome** e dois métodos, **setNome** e **getNome** para armazenamento e obtenção de valor da variável **nome**.
- 3) Incluir a classe derivada chamada PessoaJuridica (classe base Pessoa) com uma variável de instância **cnpj** e dois métodos, **setCnpj** e **getCnpj** para armazenamento e obtenção de valor da variável **cnpj**.
- 4) Incluir a classe derivada chamada PessoaFisica (classe base Pessoa) com uma variável de instância **cpf** e dois métodos, **setCpf** e **getCpf** para armazenamento e obtenção de valor da variável **cpf**.

- 5) Quando a opção pessoa física for selecionada o label deverá aparecer como CPF e o componente mskcpf visível para edição. O mesmo vale para a opção pessoa jurídica.
- 6) O campo nome deve estar preenchido.
- 7) O botão Cadastrar deve atribuir os valores digitados para o objeto instanciado de acordo com a opção do tipo de pessoa escolhida.

Por exemplo, no caso de pessoa física vamos atribuir da seguinte forma:

```
clientepf.setNome(txtNome.Text);
clientepf.setCpf(mskCpf.Text);
```

Não se esqueça de instanciar os objetos clientepf e clientepj.

- 8) Ao clicar no botão Exibir deve-se pegar o valor armazenado na classe e mostrar nos labels.

```
lblNome.Text = clientepf.getNome();
lblNro.Text = clientepf.getCpf();
```

- 9) Ao clicar no botão Limpar deve-se limpar todos os controles do formulário, selecionar a opção Pessoa Física, desabilitar o botão Exibir e colocar o foco no nome.
- 10) Incluir na classe PessoaFisica um método para validar o CPF e na classe PessoaJuridica um método para validar o CNPJ. Agora antes de atribuir, devemos verificar se o CPF ou CNPJ estão corretos. Se o CPF ou CNPJ estiver errado deve-se enviar uma mensagem avisando que o numero digitado está errado.
- 11) Regra de validação do CPF

Considere o seguinte número de CPF
A B C . D E F . G H I - J K

Para validar o primeiro digito

Multiplicar os números como se segue, somando seus resultados

$$\begin{aligned}SA &= A * 10 \\SB &= B * 9 \\SC &= C * 8 \\SD &= D * 7 \\SE &= E * 6\end{aligned}$$

$$\begin{aligned}SF &= F * 5 \\SG &= G * 4 \\SH &= H * 3 \\SI &= I * 2\end{aligned}$$

$$Soma = SA + SB + SC + SD + SE + SF + SG + SH + SI$$

Dividir o resultado (Soma) por 11
Se o resto da divisão = 1 ou resto = 0
O digito será 0
Senão
O dígito será 11 – resto

Verificar se o digito bate com o primeiro digito informado (J)

Para validar o segundo digito

$$\begin{aligned}SA &= A * 11 \\SB &= B * 10 \\SC &= C * 9 \\SD &= D * 8 \\SE &= E * 7 \\SF &= F * 6 \\SG &= G * 5 \\SH &= H * 4 \\SI &= I * 3 \\SJ &= J * 2\end{aligned}$$

$$Soma = SA + SB + SC + SD + SE + SF + SG + SH + SI + SJ$$

Dividir o resultado (Soma) por 11
Se o resto da divisão = 1 ou resto = 0
O digito será 0
Senão
O dígito será 11 – resto

Verificar se o digito bate com o segundo digito informado (K)

12) Regra para validação do CNPJ

Considere o seguinte numero de CNPJ

4 4 . 1 4 5 . 8 5 4 / 0 0 1 1 – 1 2
A B C D E F G H I J K L M N

Para validar o primeiro digito

Multiplicar os numeros como se segue, somando seus resultados

$$SA = A * 5$$

SB = B * 4
SC = C * 3
SD = D * 2
SE = E * 9
SF = F * 8
SG = G * 7
SH = H * 6
SI = I * 5
SJ = J * 4
SK = K * 3
SL = L * 2

Soma = SA + SB + SC + SD + SE + SF + SG + SH + SI + SJ + SK + SL

Dividir o resultado (Soma) por 11
Se o resto da divisão = 1 ou resto = 0
 O dígito será 0
Senão
 O dígito será 11 – resto

Verificar se o dígito bate com o primeiro dígito informado (M)

Para validar o segundo dígito

SA = A * 6
SB = B * 5
SC = C * 4
SD = D * 3
SE = E * 2
SF = F * 9
SG = G * 8
SH = H * 7
SI = I * 6
SJ = J * 5
SK = K * 4
SL = L * 3
SM = M * 2

Soma = SA + SB + SC + SD + SE + SF + SG + SH + SI + SJ + SK + SL + SM

Dividir o resultado (Soma) por 11
Se o resto da divisão = 1 ou resto = 0
 O dígito será 0
Senão
 O dígito será 11 – resto

Verificar se o dígito bate com o segundo dígito informado (N)

Testar o programa e apresentar ao professor.

AULA 9

Acesso a Banco de Dados utilizando assistente.

- 1) Primeiro vamos criar o banco de dados de nome alunos com as seguintes tabelas e definições:

- a) Tabela cursos

Nome do campo	Tipo de dados	Descrição
cur_sgl	Texto	Sigla do curso - texto 10
cur_nom	Texto	Nome do curso - texto 70
cur_dep_sgl	Texto	Sigla do depto - texto 10

- b) Tabela departamentos

Nome do campo	Tipo de dados	Descrição
dep_sgl	Texto	Sigla do depto - texto (10)
dep_nom	Texto	Nome do depto - texto (70)

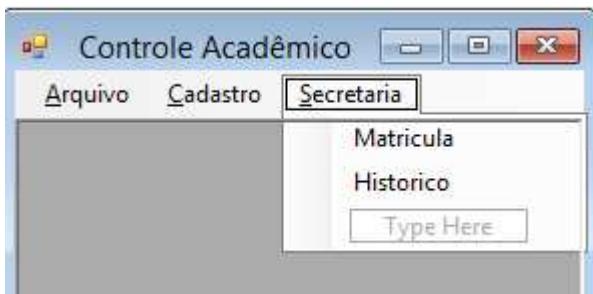
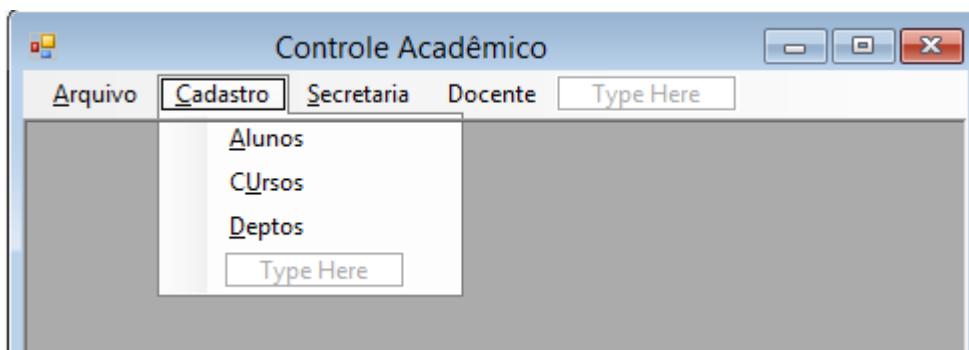
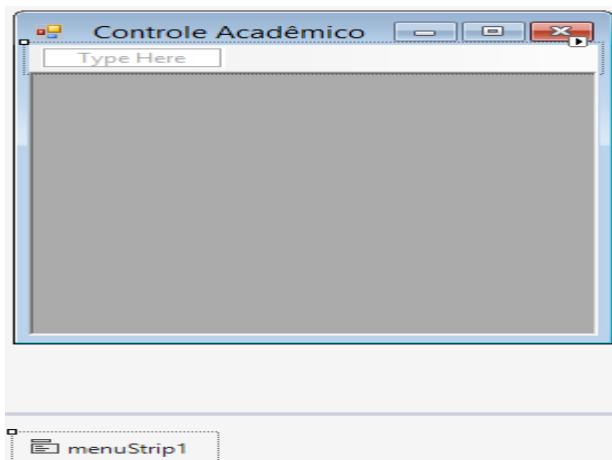
- c) Tabela Alunos

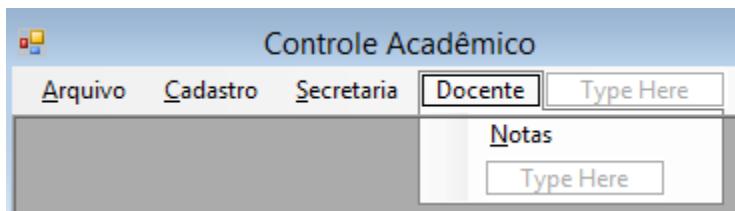
Nome do campo	Tipo de dados	Descrição
alu_mat	Texto	Matricula do Aluno - texto (10)
alu_nom	Texto	Nome do Aluno - texto(70)
alu_Ing	Texto	ano/semestre de ingresso - texto(6) AAAA/S
alu_dta_nas	Data/Hora	Data de nascimento - data abreviada
alu_sexo	Texto	Sexo do Aluno - texto(1)
alu_end_cep	Texto	Endereço - cep - texto (10)
alu_end_log	Texto	Endereço - Logradouro - texto (10)
alu_end	Texto	Endereço - nome - texto (70)
alu_end_nro	Número	Endereço numero - numero (inteiro)
alu_end_cpl	Texto	Endereço - Complemento - texto(70)
alu_end_bai	Texto	Endereço - Bairro - texto (70)
alu_end_mun	Texto	Endereço - município - texto (70)
alu_end_uef	Texto	Endereço - estado - texto (2)
alu_cur_sgl	Texto	Sigla do curso - texto (10)

- 2) Popule as tabelas com alguns registros.
- 3) Iniciar um novo projeto chamado wfaBancoDados
- 4) Renomear o Form1 para frmPrincipal configurando com as seguintes propriedades:
 - backgroundImage = imagem da Fatec
 - backgroundImageLayout = stretch
 - text = Controle Academico

- `windowState = maximized`
- `IsMdiContainer = true`

5) Inserir o componente menuStrip e configurar o menu como as figuras abaixo.





O formulário ficará parecido com a figura abaixo:



- 6) Incluir um novo formulário chamado frmAlunos com a interface parecida com a figura abaixo:

Matrícula	<input type="text"/>	Nome	<input type="text"/>
Data Nascimento	<input type="text"/>	Ingresso	<input type="text"/>
Sexo	<input type="radio"/> Feminino <input type="radio"/> Masculino		
	Curso <input type="text"/>		
Endereço			
CEP	<input type="text"/>	Logradouro	<input type="text"/>
Endereço	<input type="text"/>		Nro <input type="text"/>
Complemento	<input type="text"/>		
Bairro	<input type="text"/>		
Município	<input type="text"/>		Estado <input type="button" value="SP"/> <input type="button" value="AC"/>

- 7) Carregar no comboBox as siglas dos estados brasileiros. Para isso utilize a propriedade “Items” do comboBox.

Acre - AC
Alagoas - AL
Amapá - AP
Amazonas - AM
Bahia - BA
Ceará - CE
Distrito Federal - DF
Espírito Santo - ES
Goiás - GO
Maranhão - MA
Mato Grosso - MT
Mato Grosso do Sul - MS
Minas Gerais - MG
Pará - PA
Paraíba - PB
Paraná - PR
Pernambuco - PE
Piauí - PI
Rio de Janeiro - RJ
Rio Grande do Norte - RN
Rio Grande do Sul - RS
Rondônia - RO
Roraima - RR
Santa Catarina - SC
São Paulo - SP
Sergipe - SE
Tocantins – TO

- 8) No frmPrincipal ao clicar no menu Alunos chamar o frmAlunos que acabamos de criar. Para exibir o segundo formulário use o código abaixo.

```
private void alunosToolStripMenuItem_Click(object sender, EventArgs e)
{
    Form objalunos = new frmAlunos();
    objalunos.Show();
}
```

- 9) No frmPrincipal ao clicar no menu Arquivo/Sair fechar a aplicação.

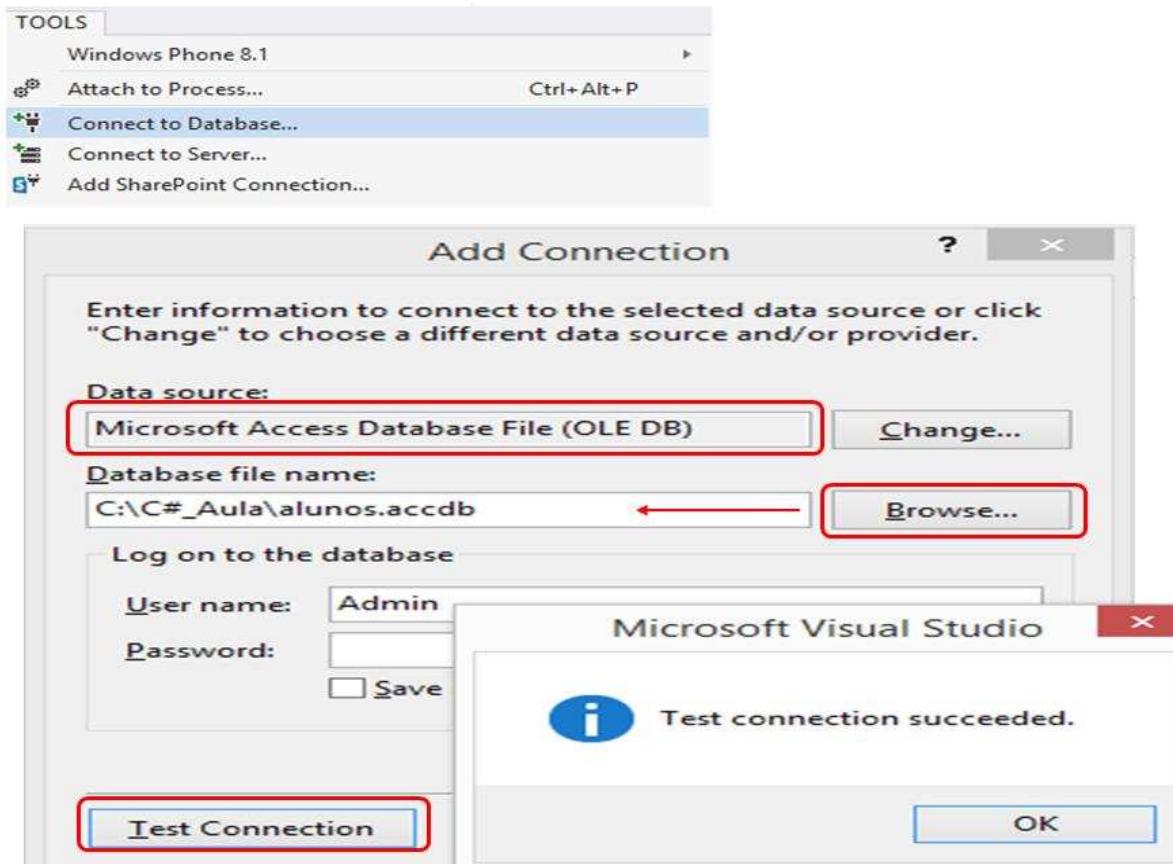
10) Salve o projeto e Teste a aplicação.

Agora vamos acessar o banco de dados do Access “Alunos” que acabamos de criar.

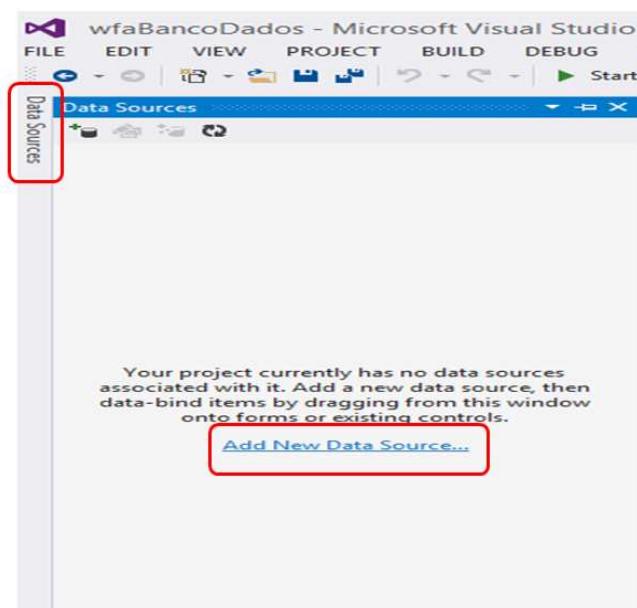
- 11) Abra o formulário frmAlunos e dê um click no item de menu “Ferramentas / Conectar ao Banco de Dados”.

Na janela Adicionar Conexão selecione Arquivo de Banco de Dados do Microsoft Access (OLE DB).

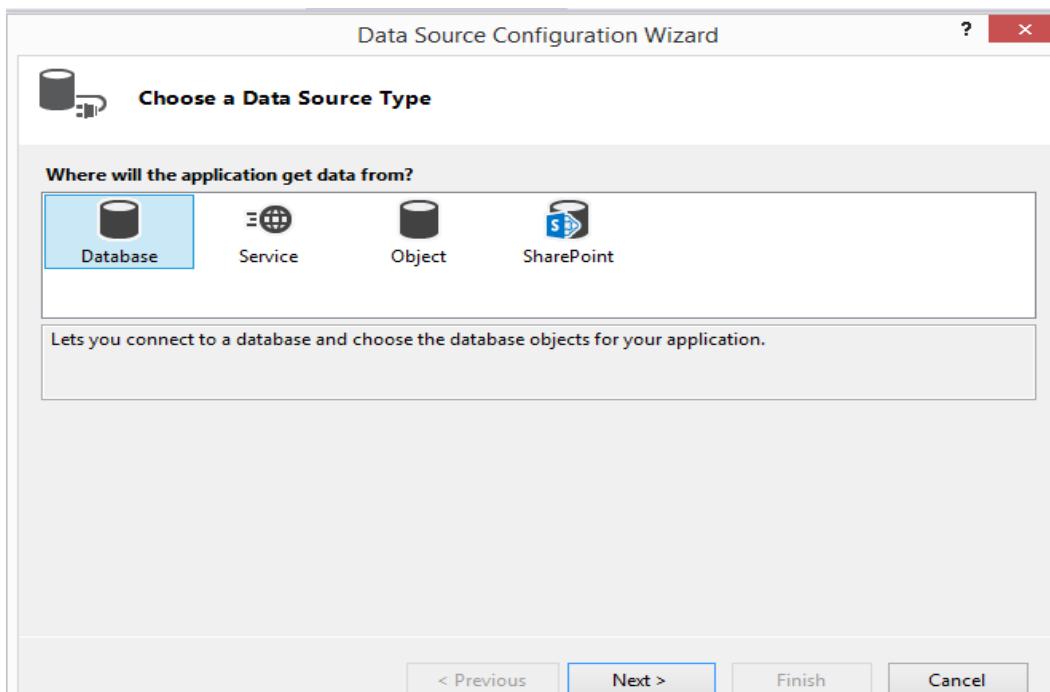
Localize o banco de dados do Access “Alunos” através do botão Procurar... e teste a conexão.



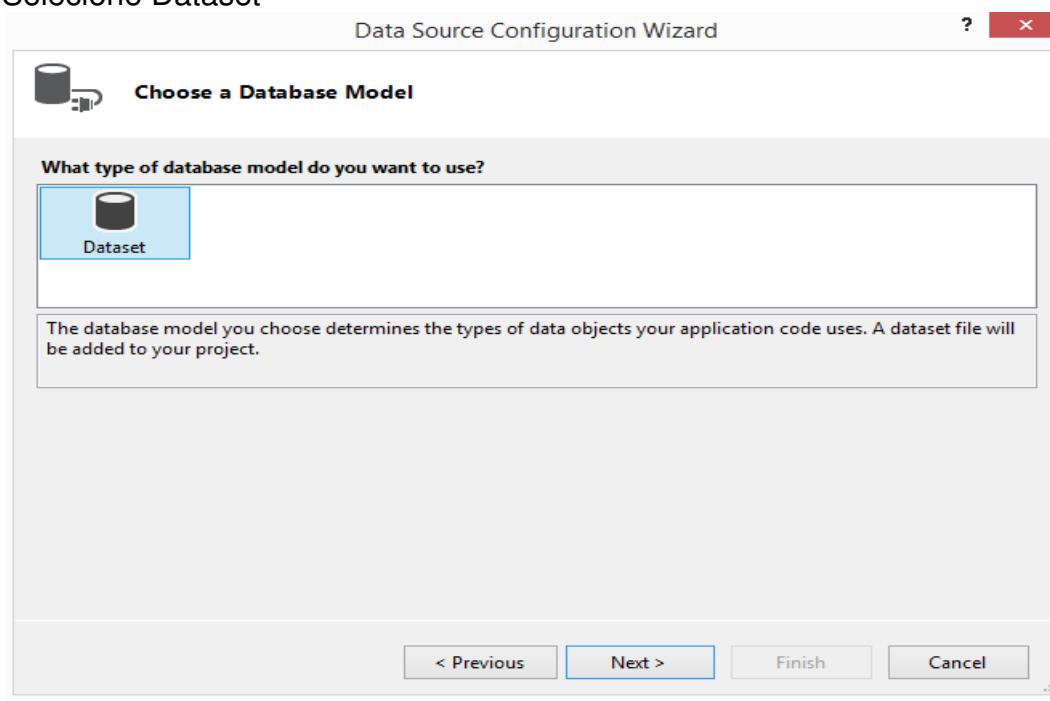
A seguir click na barra Data Source que se encontra no lado esquerdo selecione Adicionar Nova Fonte de Dados.



Selecione Banco de Dados e siga os passos do assistente.

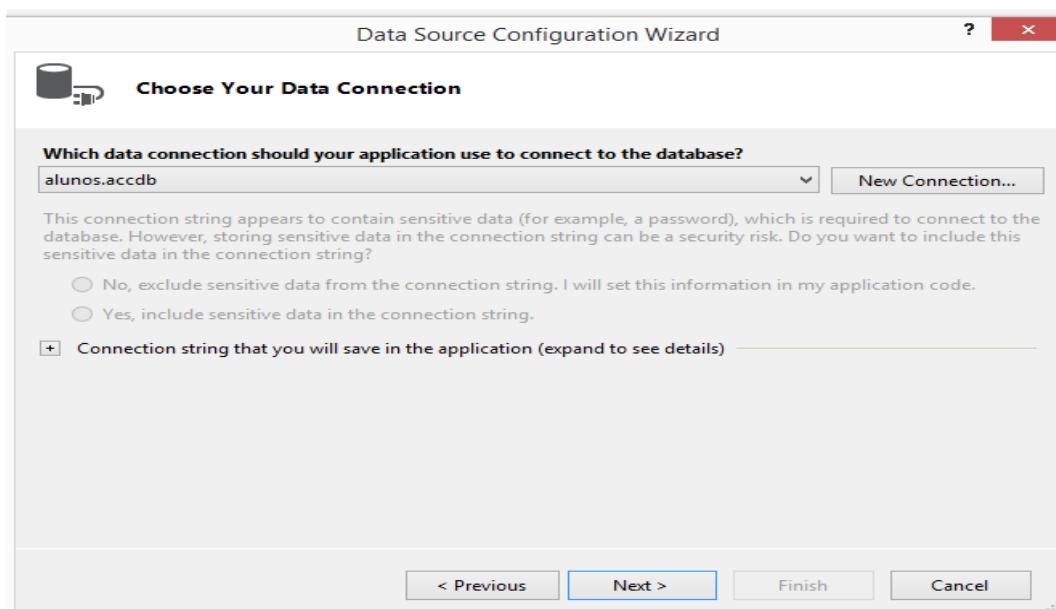


Selecione Dataset

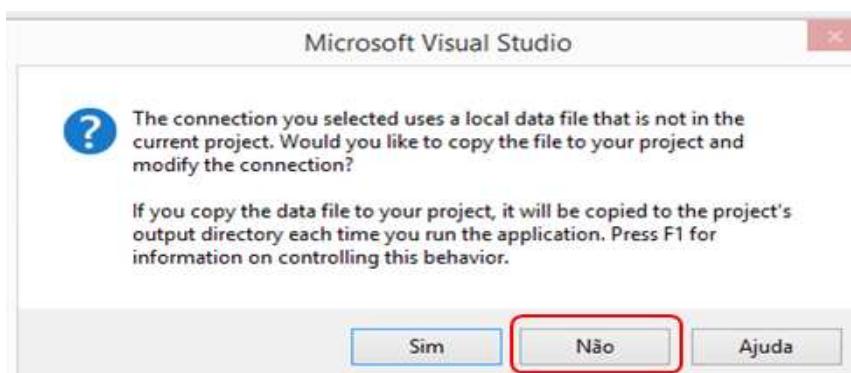


APOSTILA DE VISUAL C#

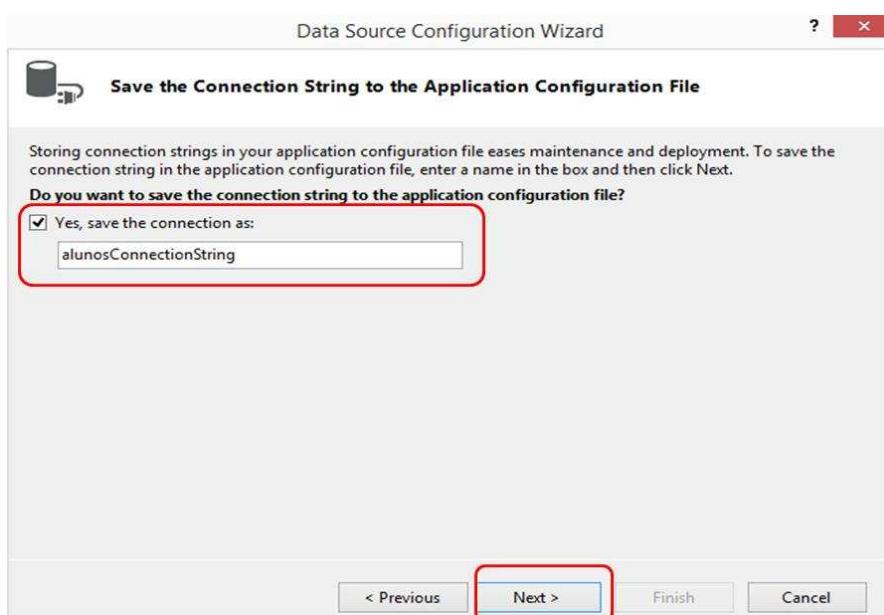
Selecione a conexão que criamos ou crie uma nova conexão.



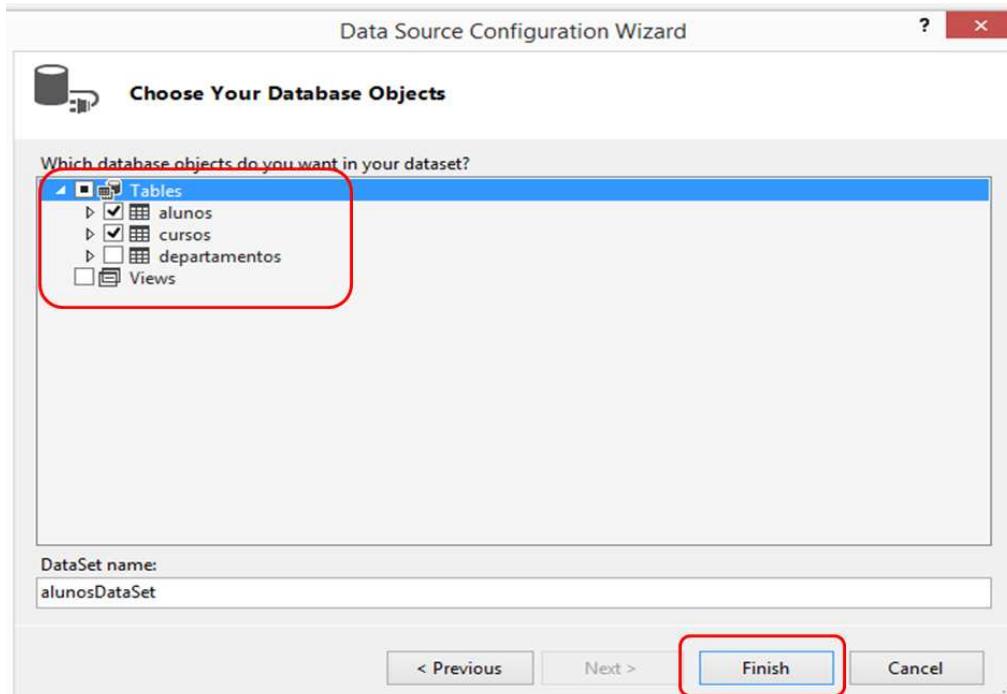
Quando aparecer a mensagem abaixo, selecione Não.



Salve a conexão no arquivo de configuração.



Selecione a(s) tabela(s) do banco de dados que vamos utilizar no programa e click no botão Concluir.



Na Fonte de Dados, verifique as tabelas do nosso Dataset.

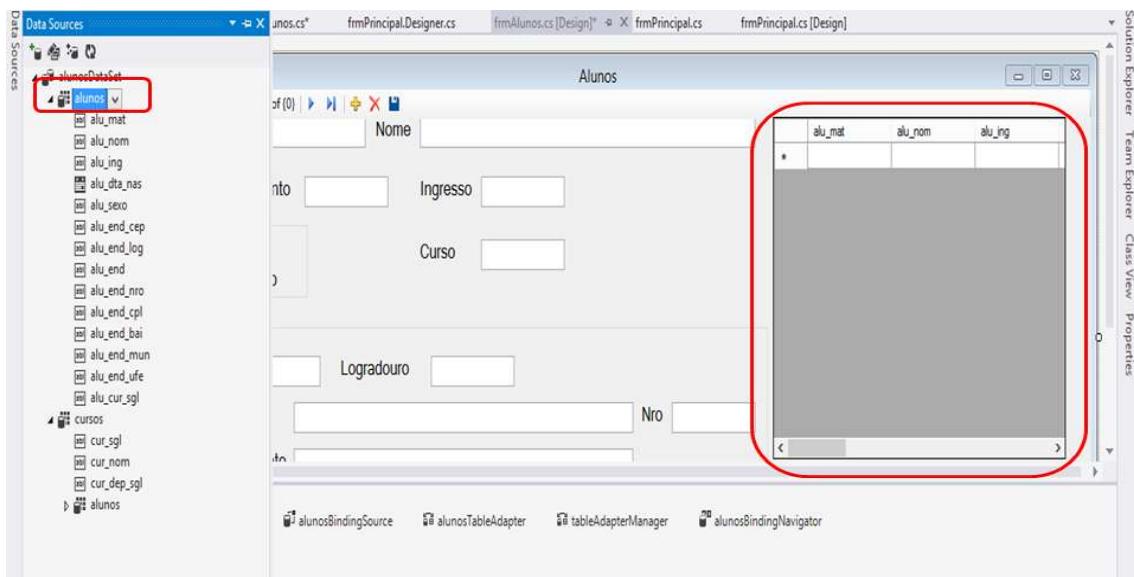


Note que o C# incluiu alguns itens na janela Gerenciador de Soluções, como por exemplo o módulo app.config, onde está descrito o string de conexão (ConnectionString) para acesso ao banco de dados:

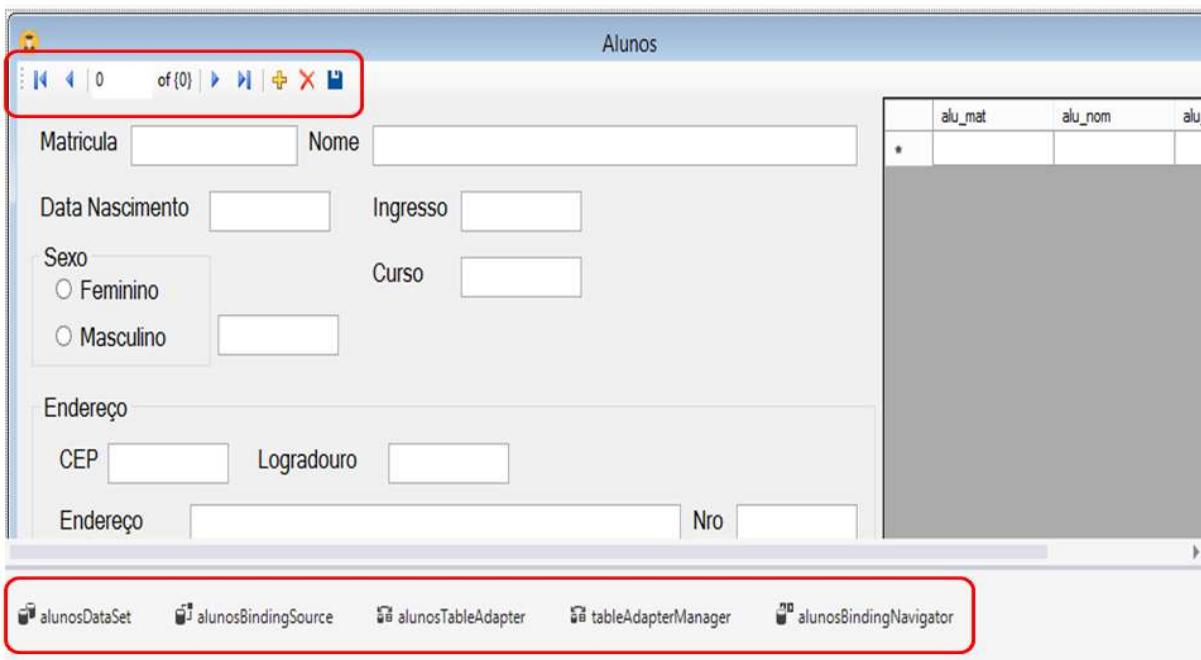


Abra a janela Data Sources e arraste a tabela Alunos com o mouse e solte-a no formulário.

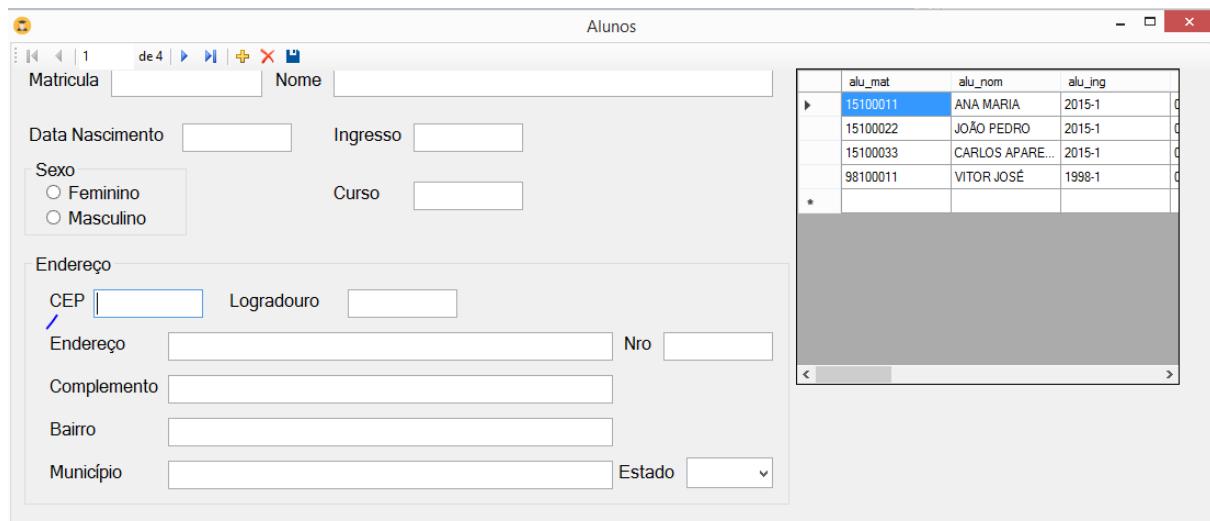
Você também pode arrastar um campo da tabela e soltá-lo no formulário.



Note que o c# incluiu vários componentes de acesso e manipulação de banco de dados em seu projeto.



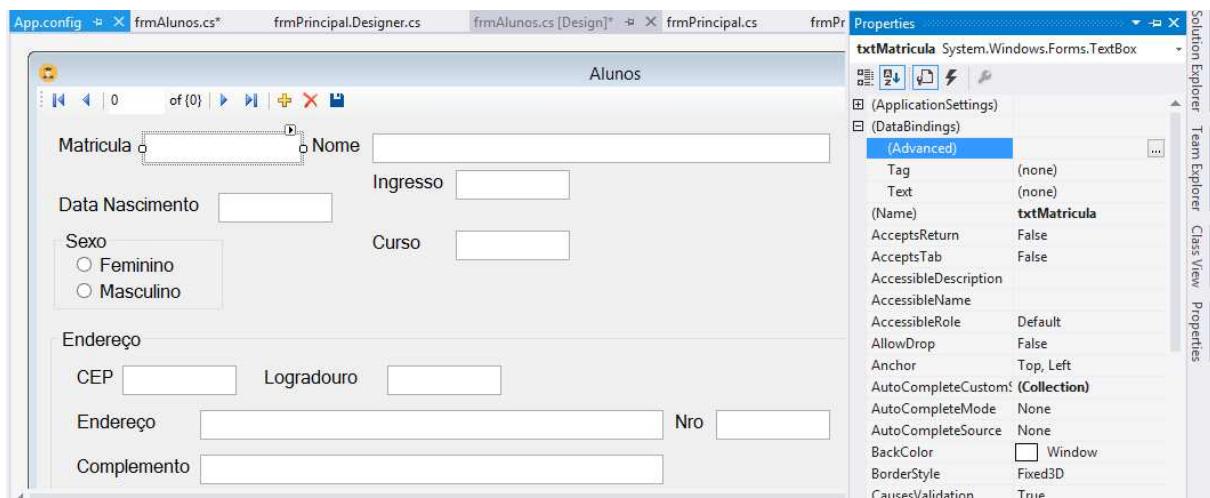
Execute o programa. Este deve estar mais ou menos como a janela abaixo:



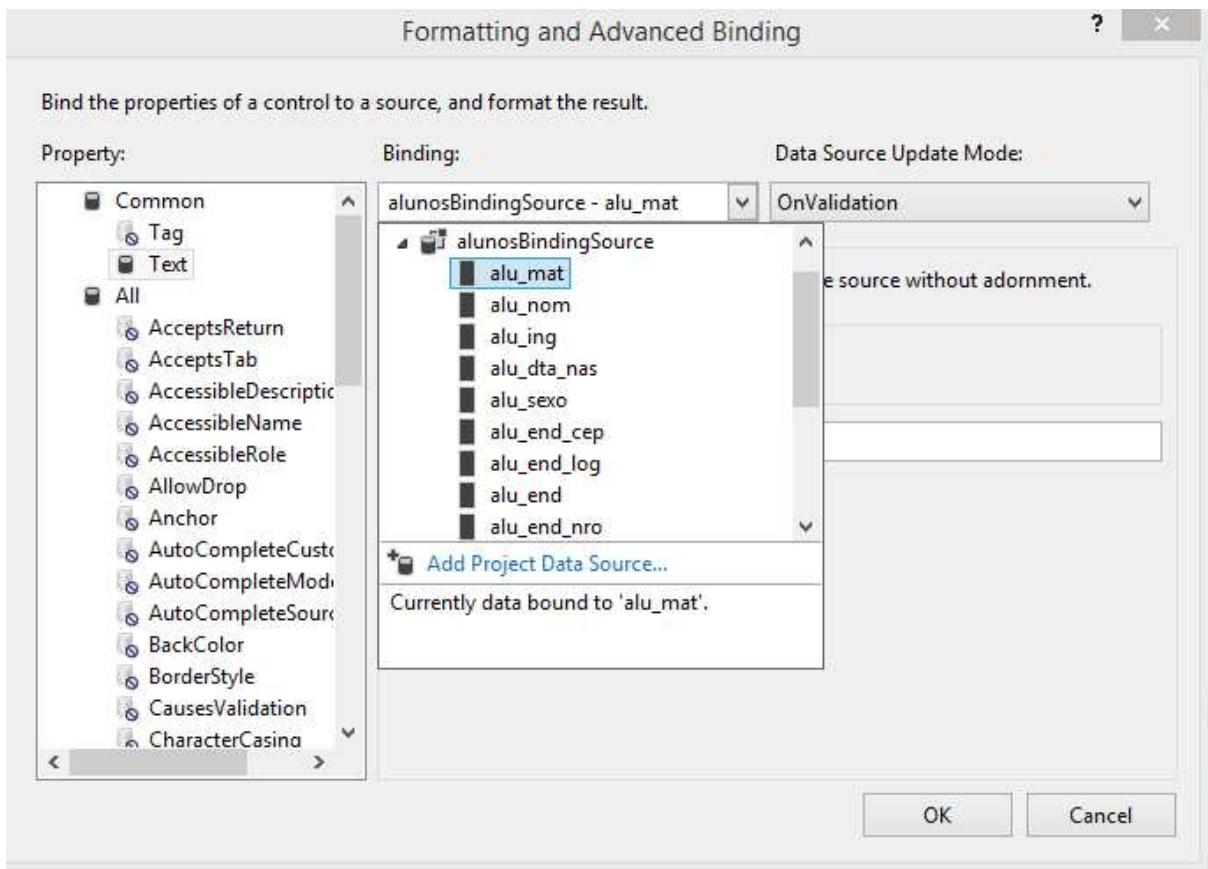
Insira alguns registros através do DataGridView do formulário.

Vamos agora fazer a ligação do banco de dados às caixas de texto do Formulário. Para isso precisamos selecionar a caixa de texto na qual queremos acessar o banco de dados (Por exemplo, a caixa de texto txtMatricula) e configurar algumas propriedades: (DataBindings) / (Avançado) – Veja as figuras a seguir:

APOSTILA DE VISUAL C#



Dê um click nos (...) e configure a janela Formatação e Ligação Avançada, escolhendo o campo que lhe interessa exibir na caixa de texto.



Execute o programa e verifique que o campo Matrícula no banco de dados, é exibido na respectiva caixa de texto do formulário.

APOSTILA DE VISUAL C#

alu_mat	alu_nom	alu_ing
15100011	ANA MARIA	2015-1
15100022	JOÃO PEDRO	2015-1
15100033	CARLOS APARE...	2015-1
98100011	VITOR JOSÉ	1998-1
*		

Faça o mesmo para as demais caixas de texto do formulário.

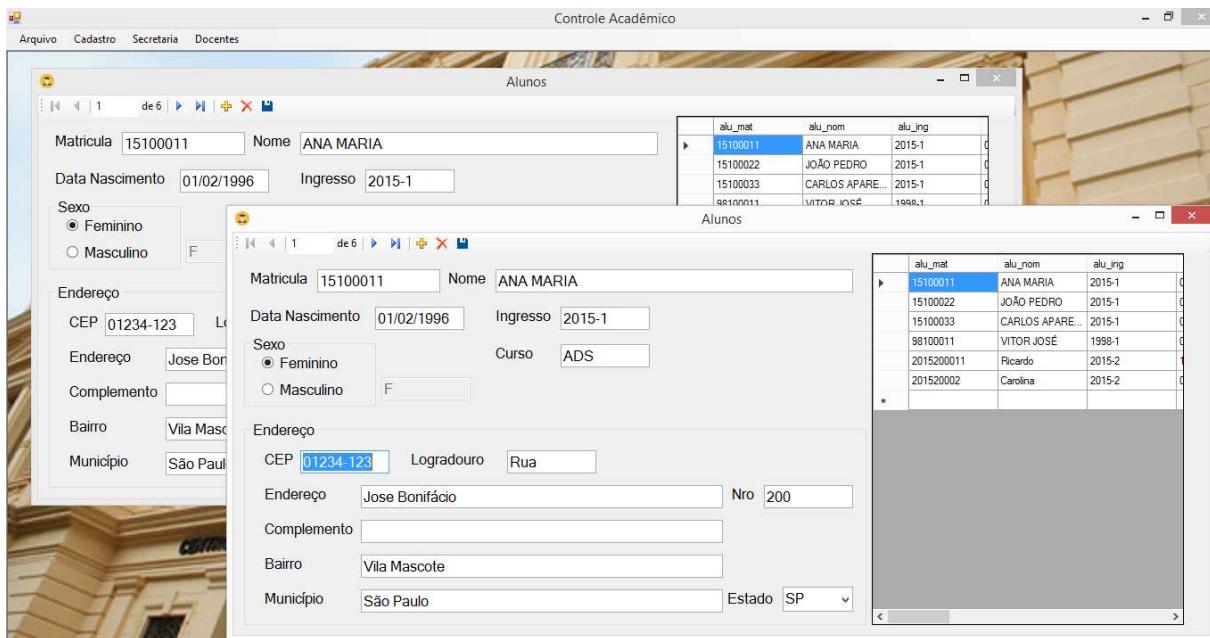
Inclua a caixa de texto “Sexo” no formulário para exibir esse campo do banco de dados. Obs. Essa caixa de texto deverá estar desabilitada (propriedade enabled=false). Ao ocorrer o click do mouse em um dos botões de opção a caixa de texto exibirá o valor do sexo escolhido.

12) Salve seu projeto e teste o programa.

13) Note que processamento do sistema inicia-se pelo formulário “frmPrincipal” e ao ocorrer o click do mouse nos itens de menu Cadastro / Aluno, é instanciado o formulário frmAlunos.

alu_mat	alu_nom	alu_ing
15100011	ANA MARIA	2015-1
15100022	JOÃO PEDRO	2015-1
15100033	CARLOS APARE...	2015-1
98100011	VITOR JOSÉ	1998-1
2015200011	Ricardo	2015-2
201520002	Carolina	2015-2
*		

Clique novamente no item de menu Alunos. O que acontece? Minimize o formulário e perceba que ele criou 2 instâncias do formulário Alunos. Isso não deveria acontecer...



Podemos fazer com que a aplicação abra apenas uma única vez o formulário. Experimente usar o método `ShowDialog`. Veja a diferença.

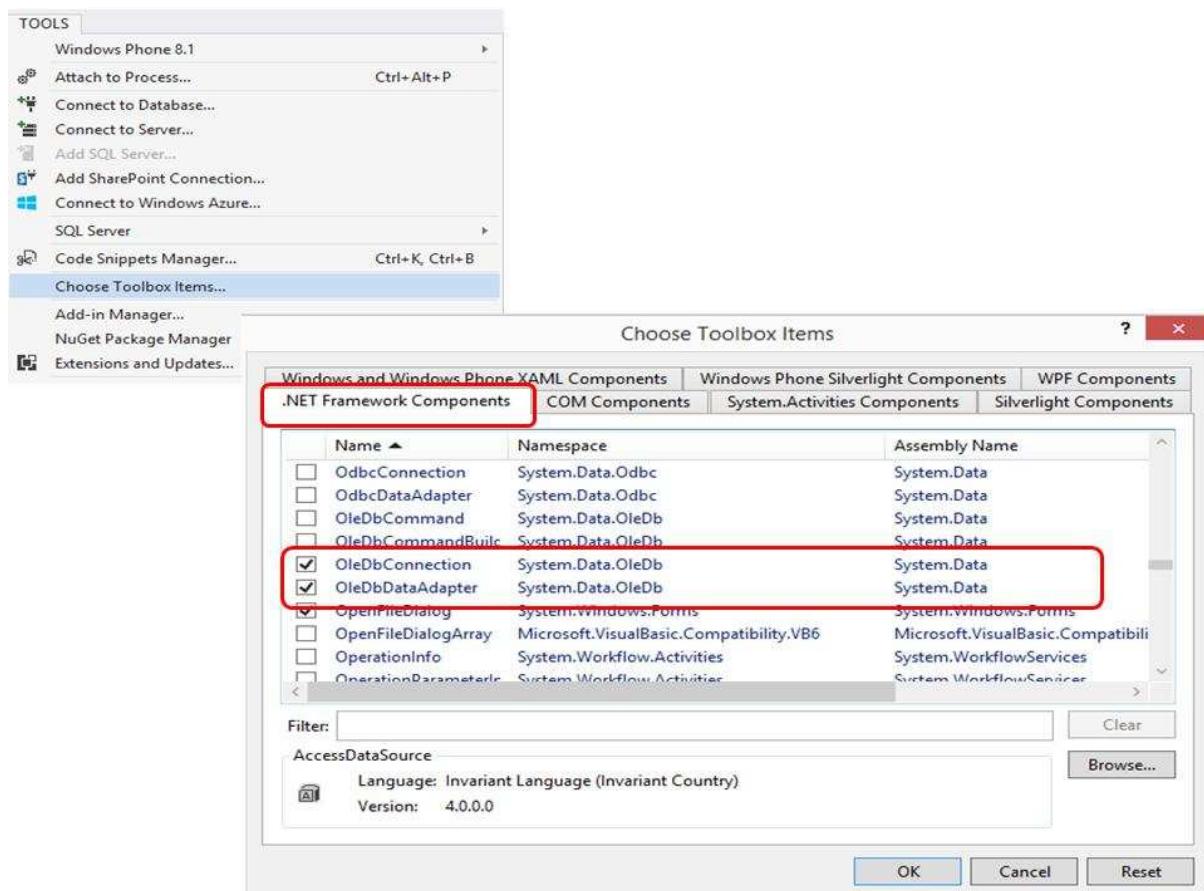
```
private void alunosToolStripMenuItem_Click(object sender, EventArgs e)
{
    Form objalunos = new frmAlunos();
    objalunos.ShowDialog();
}
```

O método `Window.ShowDialog` mostra a janela e desabilita todas as outras janelas no aplicativo. Devolve o controle somente quando for fechada.

Acesso a banco de dados por meio de componente e código:

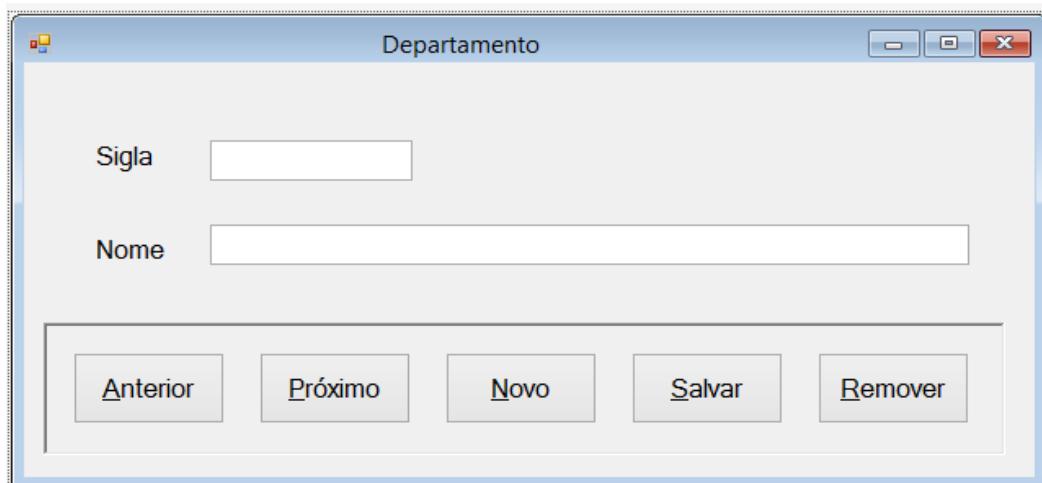
Podemos acessar um banco de dados utilizando os componentes OleDbConnection e OleDbDataAdapter. Para isso verifique se os componentes OleDbConnection e OleDbDataAdapter estão presentes na Caixa de Ferramentas, no conjunto de componentes "Dados" ou no grupo Geral, ao final da lista. Se não estiverem, inclua-os através dos itens de menu “Ferramentas / Escolher Itens da Caixa de Ferramentas...”.

Procure na lista de componentes e selecione os componentes OleDbConnection e OleDbDataAdapter.



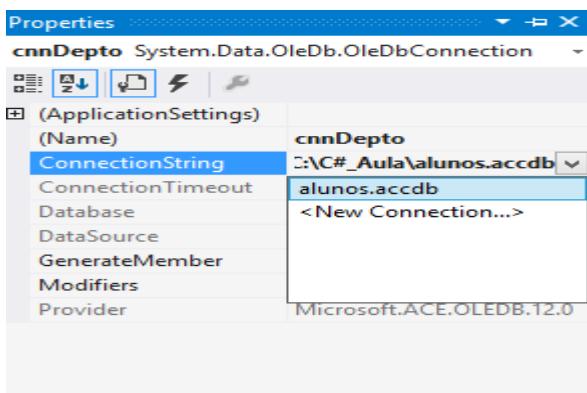
Uma vez que os componentes estão na caixa de ferramentas, vamos testar seu uso, seguindo os passos abaixo:

- 1) Incluir um novo formulário chamado frmDept e deixe a interface parecida com a figura abaixo:

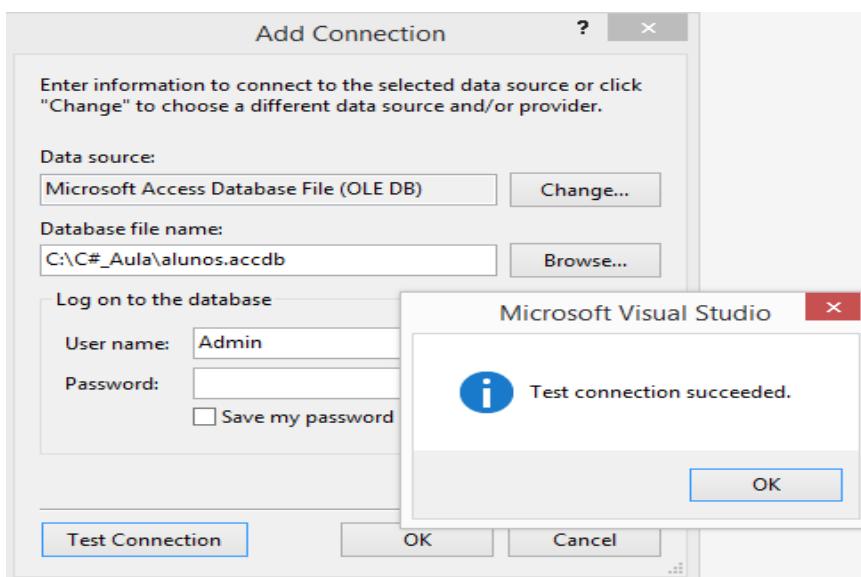


Após isso, dê um duplo click no componente OleDbConnection para incluí-lo no formulário. Através da janela de propriedades, mude seu nome para cnnDepto.

Em sua propriedade ConnectionString, escolha <Nova Conexão>.

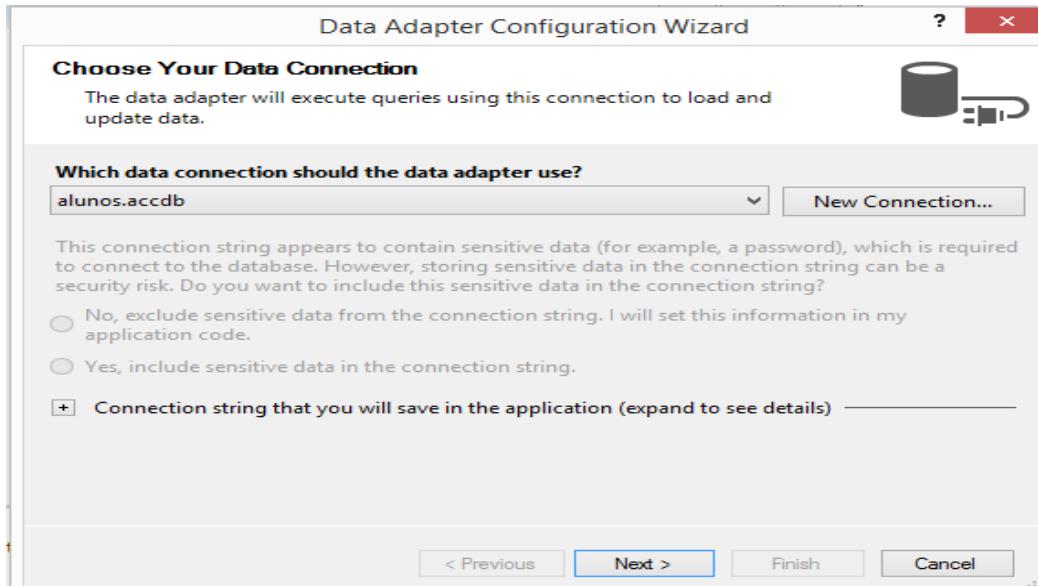


Na janela que abrir, selecione o “Data source” correspondente a arquivo de Access e em “Database file name” indique o caminho e o arquivo “Alunos”, teste a conexão e dê um click no botão Ok.

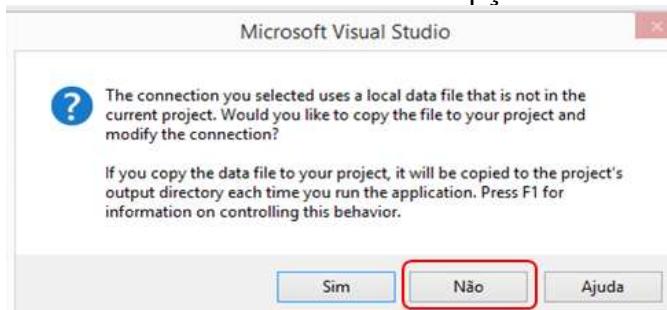


APOSTILA DE VISUAL C#

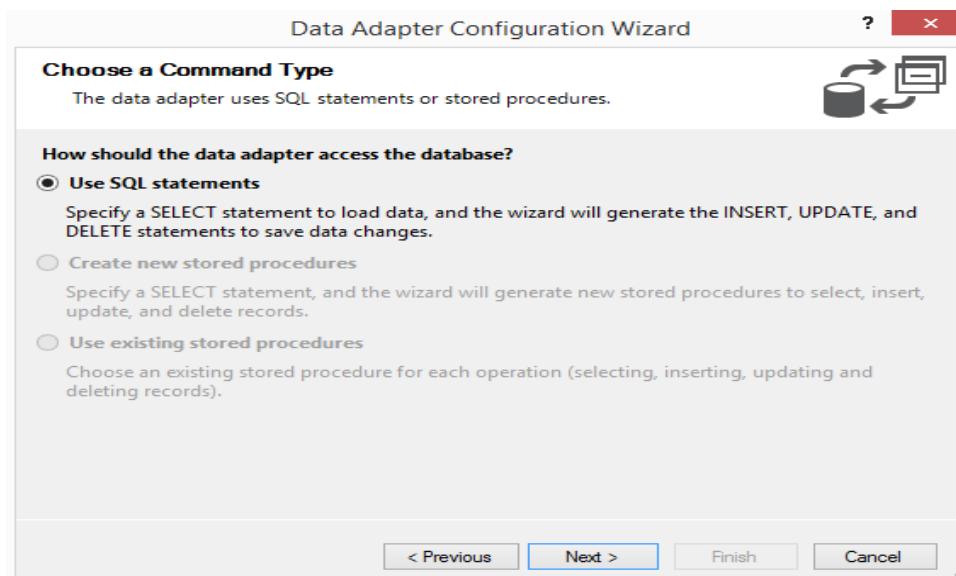
Em seguida inclua no formulário o componente OleDbDataAdapter. Deverá surgir a janela do assistente para guiá-lo na configuração. Na primeira janela, selecione Nova Conexão... ou selecione a conexão já criada. No caso de nova conexão siga os passos que fizemos anteriormente. Selecione <Próximo>



Como da outra vez seleciona a opção Não.

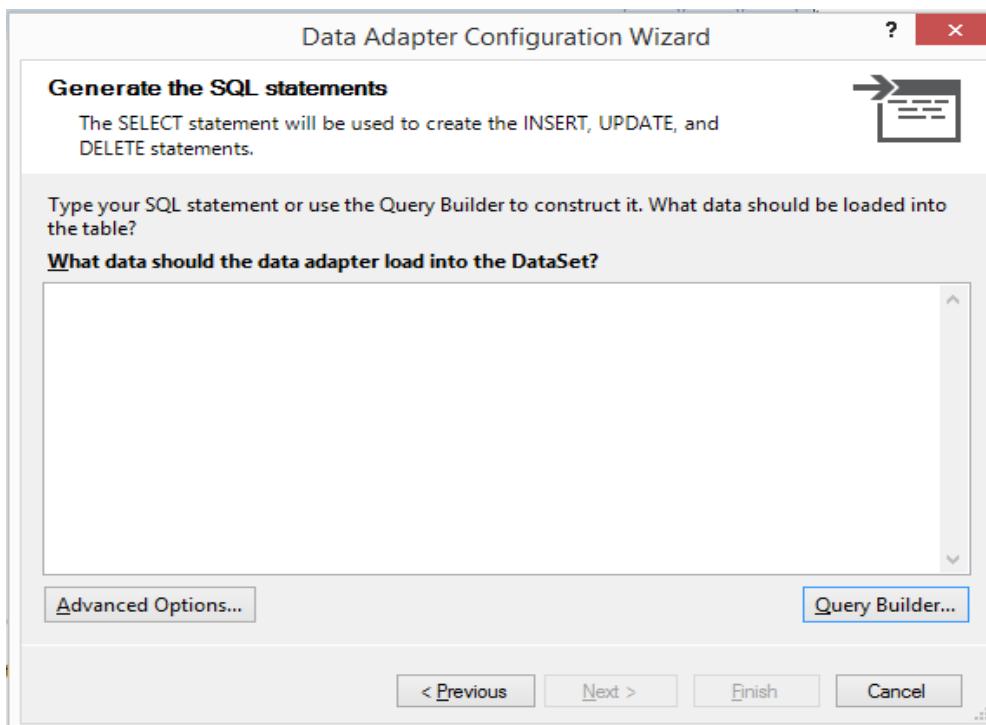


Clique em <Próximo>

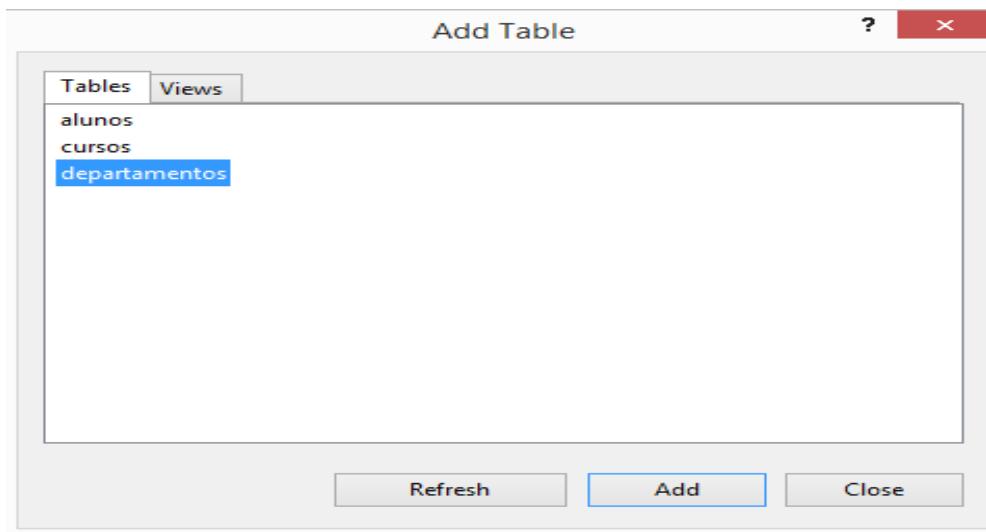


APOSTILA DE VISUAL C#

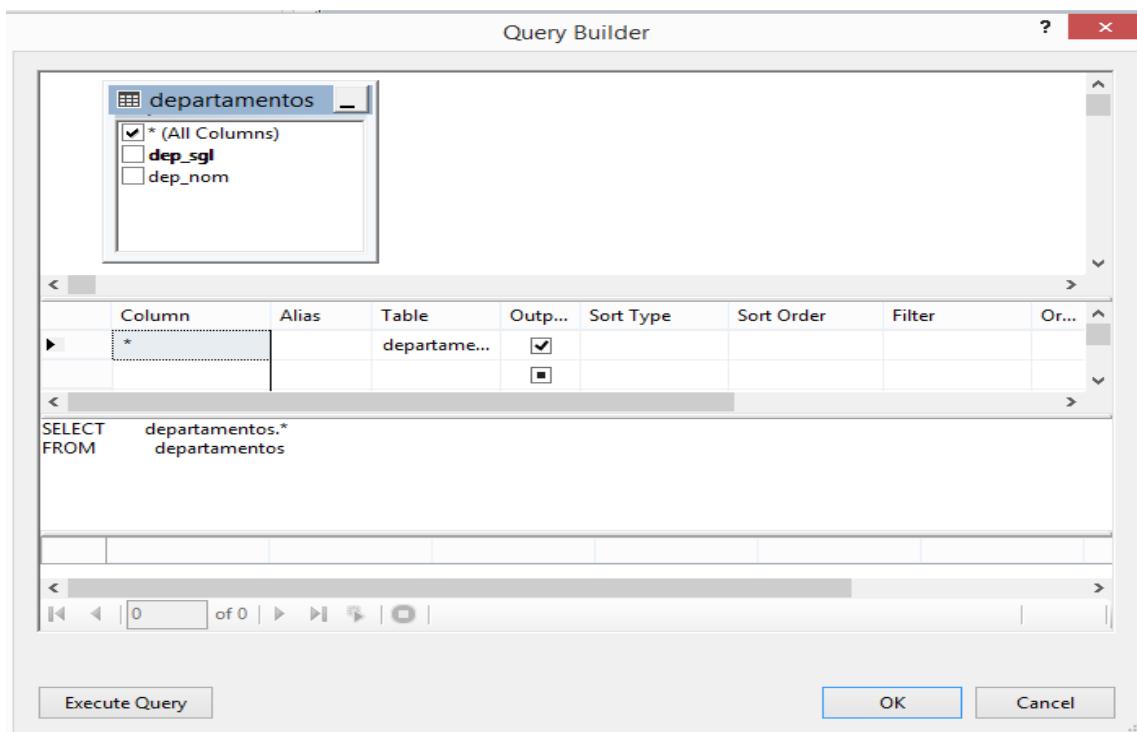
Na tela de configuração do Data Adapter selecione <Construtor de Consultas>.



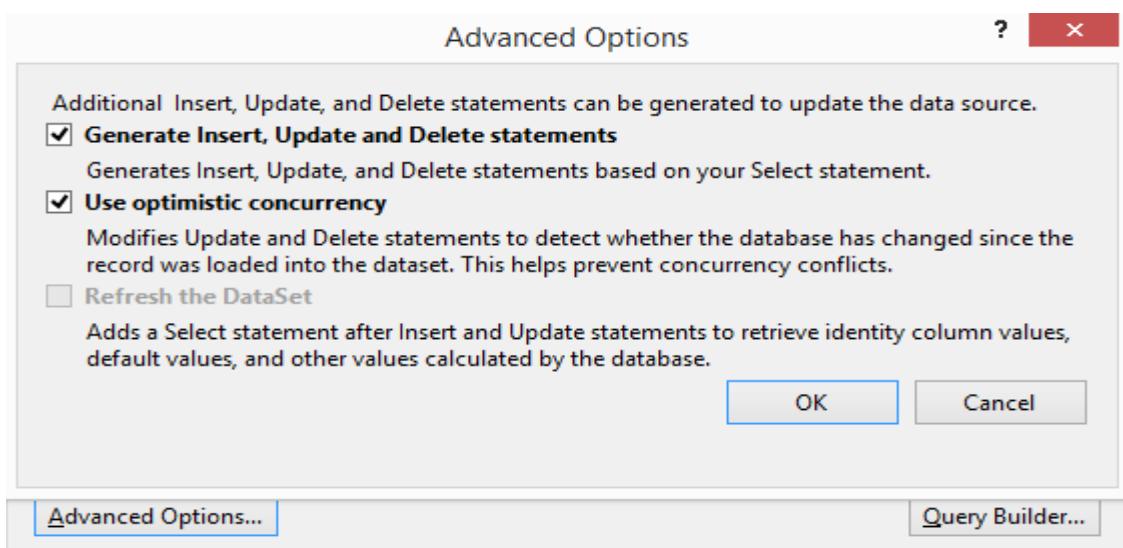
Adicione as tabelas que deseja utilizar.



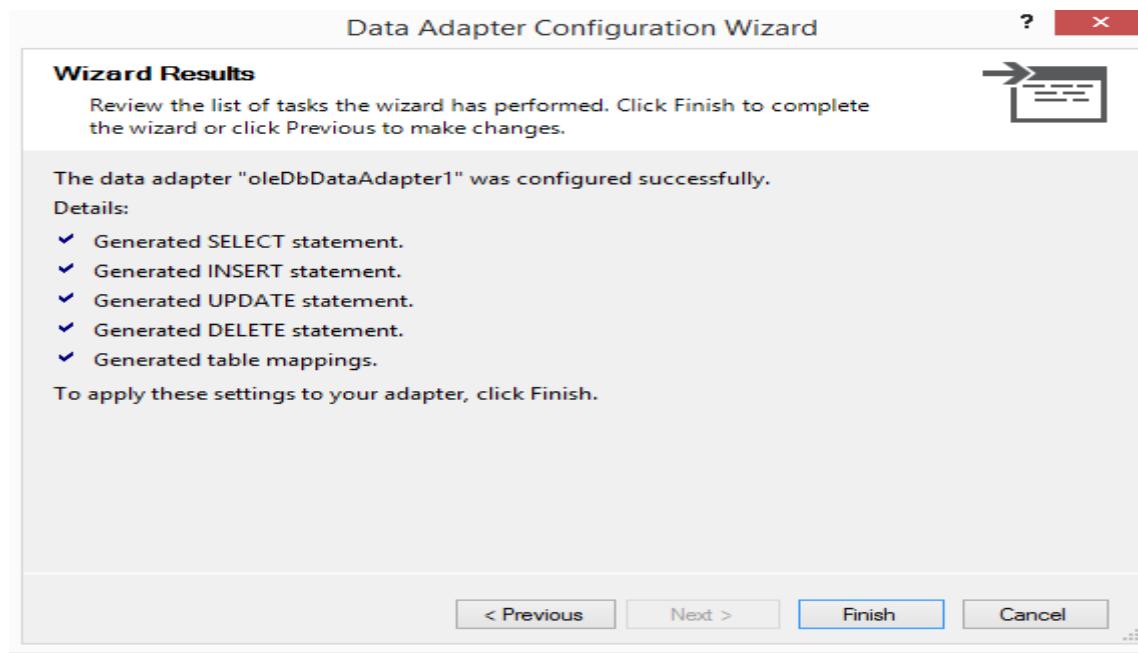
Selecione os campos da tabela que vai utilizar no seu programa e de OK.



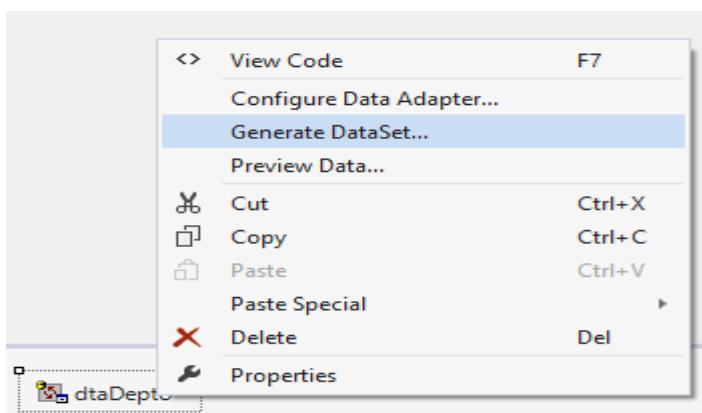
Na tela de configuração do Data Adapter selecione <Opções Avançadas> e verifique se as opções seguintes estão marcadas e clique OK.



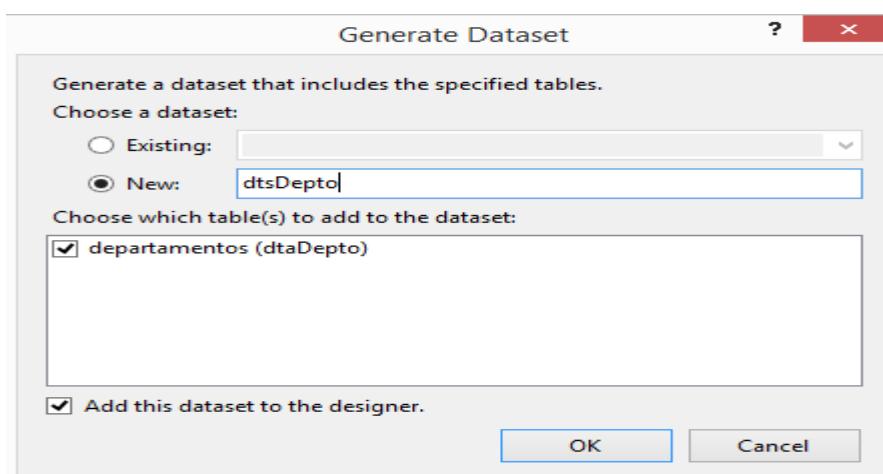
Na tela de configuração do Data Adapter selecione <Proximo> e será mostrado os comandos gerados para o adapter.



Clique em <Terminar> e vamos agora configurar algumas propriedades do componente OleDbDataAdapter. Mude seu nome para dtaDept. Dê um click no componente dtaDept com o botão direito do mouse e selecione Gerar Conjunto de Dados...



Crie um novo Dataset de nome dtsDept – veja a janela a seguir.



Note que o assistente incluiu o componente dtsDept01. Mude seu nome para dtsDept0.

Finalmente temos todos os componentes configurados. Vamos agora partir para a programação:

Dê um duplo click no formulário para abrir o tratador do evento frmBD_Load. E Digite o código a seguir:

```
public partial class frmDept0 : Form
{
    public frmDept0()
    {
        InitializeComponent();
    }

    private BindingManagerBase bmbDept0;

    private void frmDept0_Load(object sender, EventArgs e)
    {
        this.bmbDept0 = this.BindingContext[dtsDept0, "departamentos"];
        this.txtSgl.DataBindings.Add("Text", this.dtsDept0, "departamentos.dep_sgl");
        this.txtNome.DataBindings.Add("Text", this.dtsDept0, "departamentos.dep_nom");
        this.dtaDept0.Fill(dtsDept0, "departamentos");
    }
}
```

Esclarecimentos sobre o código acima:

É criado o objeto bmbDept0, do tipo BindingManagerBase. Quando o formulário for carregado, a fonte de dados de dtsDept0 é associada ao objeto bmbDept0 através da propriedade BindingContext do formulário.

Em seguida, é atribuído à propriedade DataBindings da caixa de texto txtSgl o campo dep_sgl e para caixa de texto txtNome o campo “dep_nom” do banco de dados, através do método Add.

Na sequencia, são carregados os dados em dtsCadastro, através do método Fill

Execute o programa e verifique que o conteúdo do campo “dep_sgl” e “dep_nom” do banco de dados aparece na caixa de texto do formulário.

Podemos melhorar um pouco nosso programa visando tratar erros que porventura ocorram, utilizando um bloco try...catch – acompanhe o exemplo:

```
private void frmDept0_Load(object sender, EventArgs e)
{
    try
    {
        this.bmbDept0 = this.BindingContext[dtsDept0, "departamentos"];
        this.txtSgl.DataBindings.Add("Text", this.dtsDept0, "departamentos.dep_sgl");
        this.txtNome.DataBindings.Add("Text", this.dtsDept0, "departamentos.dep_nom");
        this.dtaDept0.Fill(dtsDept0, "departamentos");
    }
    catch (Exception errAplicação)
    {
        MessageBox.Show(this, errAplicação.Message, "Erro...");
    }
    finally
    {
        if (cnnDept0.State == ConnectionState.Open) this.cnnDept0.Close();
    }
}
```

- 2) Altere a propriedade name dos botões para btnProximo, btnAnterior, btnNovo, btnExclui e btnSalvar.

Vamos agora à programação dos botões Próximo, Anterior, Incluir, Novo, Excluir e Salvar.

Acompanhe a programação:

```

private void btnProximo_Click(object sender, EventArgs e)
{
    this.bmbDeptos.Position++;
}

private void btnAnterior_Click(object sender, EventArgs e)
{
    this.bmbDeptos.Position--;
}

private void btnNovo_Click(object sender, EventArgs e)
{
    this.bmbDeptos.AddNew();
}

private void btnSalvar_Click(object sender, EventArgs e)
{
    this.bmbDeptos.EndCurrentEdit();
    this.dtaDeptos.Update(this.dtsDeptos);
    this.dtsDeptos.AcceptChanges();
}

private void btnExclui_Click(object sender, EventArgs e)
{
    this.bmbDeptos.RemoveAt(bmbDeptos.Position);
}

```

- 3) Salve seu projeto.

Obs:

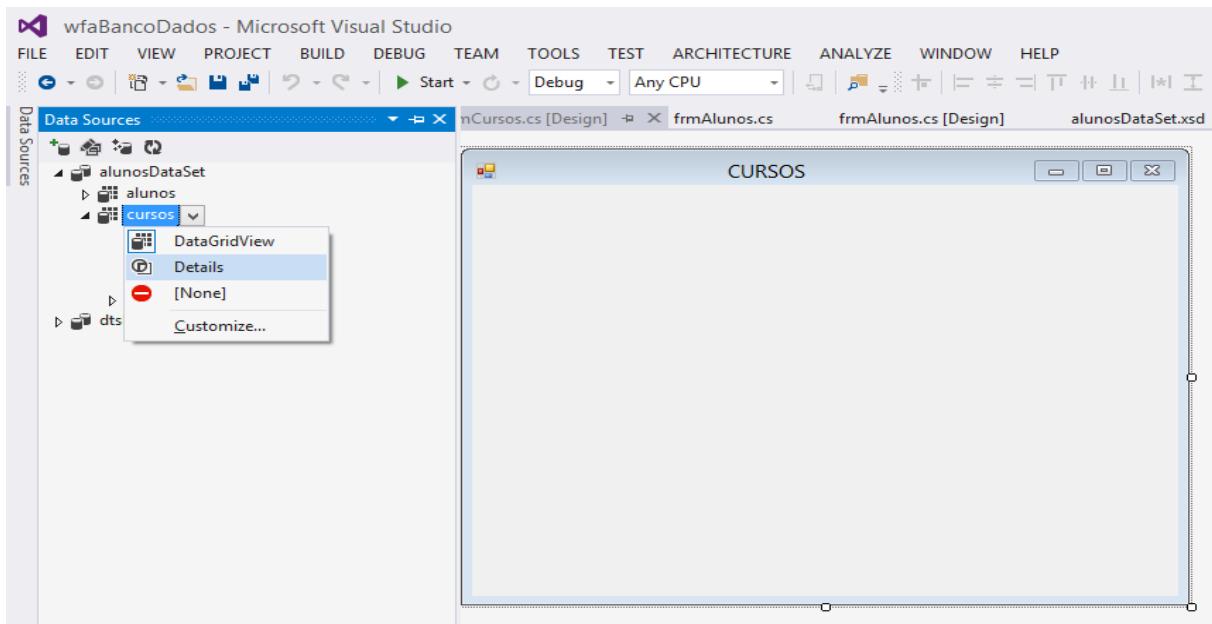
Resumidamente, vimos que o acesso a banco de dados se dá através das etapas:

- 1-Conexão do formulário ao banco de dados (Inclusão dos componentes de acesso do BD ao projeto);
- 2-Teste da conexão dos componentes ao banco de dados;
- 3-Adição da fonte de dados ao projeto;
- 4-Criação do string de conexão;
- 5-Vinculação dos campos do BD às caixas de texto do formulário.

AULA 10

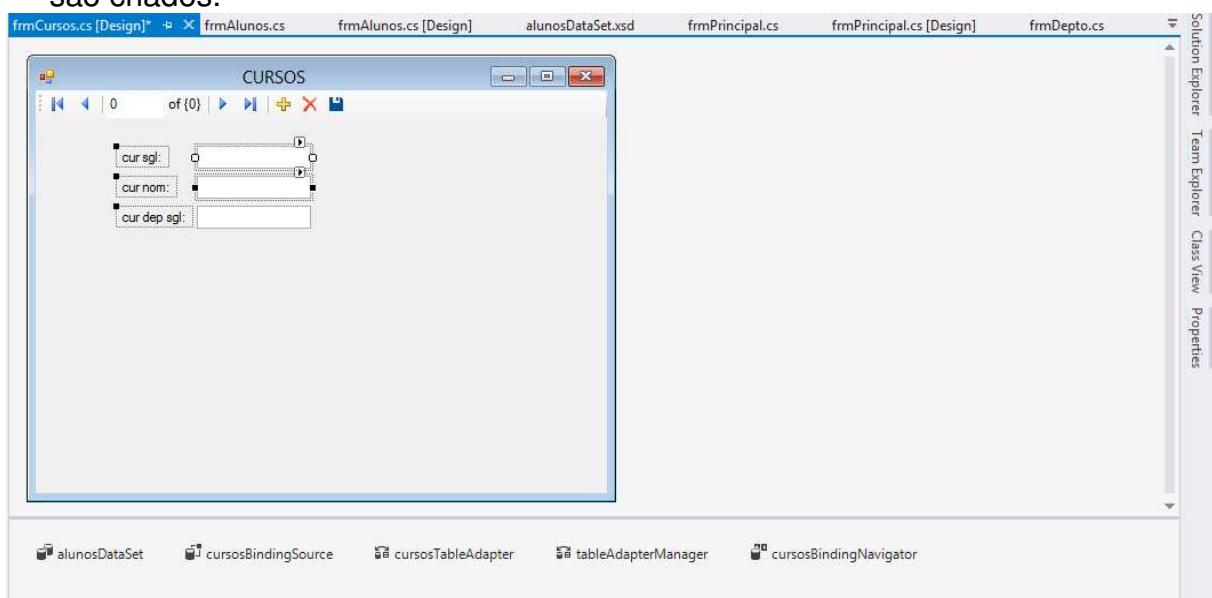
Elaborando consultas ao banco de dados

- 1) Abra o projeto da aula passada e adicione um novo formulário chamado frmCursos.
- 2) Na janela Fontes de Dados, selecione o datatable cursos e escolha o modo Detalhes.

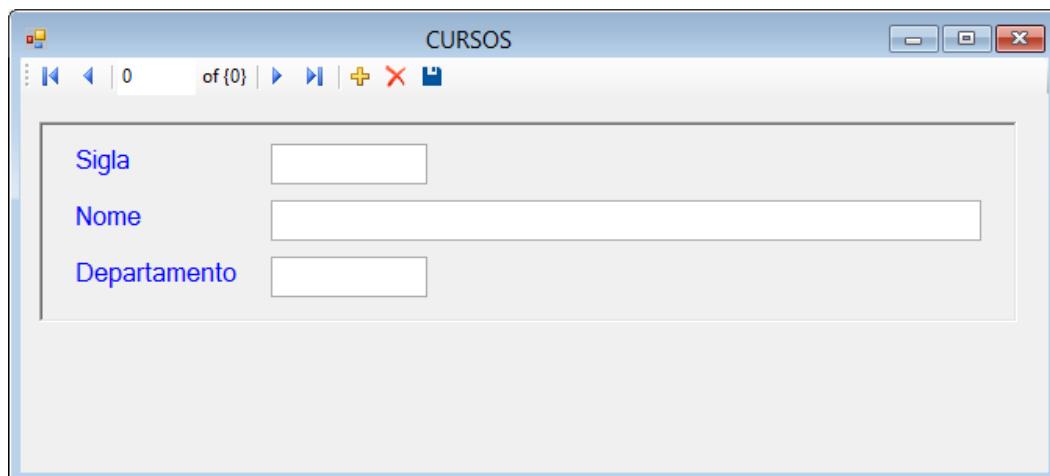


Arraste o datatable Cursos para o formulário.

O formulário cursos ficará como a figura abaixo. Observe os componentes que são criados.

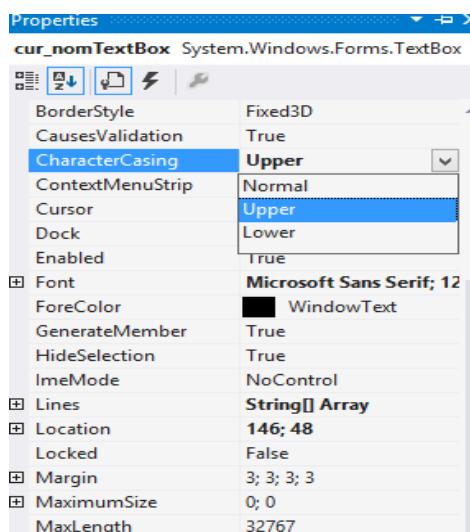


- 3) Altere a interface conforme layout abaixo.
- Insira um painel (Panel) e altere a propriedade borderstyle para Fixed3D.
 - Altere a cor dos labels para azul.



- Salve o projeto.
- Faça a chamada do curso através do item Cadastro, SubItem Cursos nos itens de menu do formulário frmPrincipal.
- Para a tabela de curso vamos definir as seguintes validações:
 - Todos os dados referentes ao curso devem ser gravados em letra maiúscula.
 - Não permitir que o usuário ultrapasse o tamanho definido nas tabelas.
 - Não permitir a entrada de dados em branco.

Para resolver o item a podemos alterar a propriedade das caixas de texto CharacterCasing para Upper ou então executar a rotina feita nas aulas anteriores.



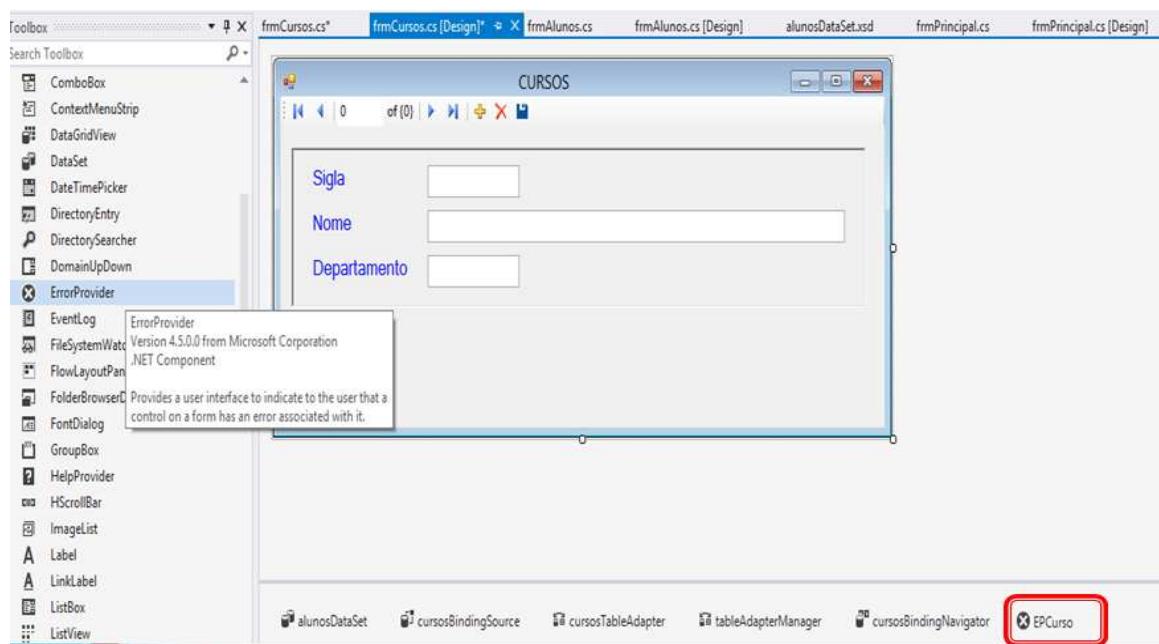
Como você pode resolver o item b? Implemente a sua solução. Talvez uma solução rápida seja utilizar a propriedade MaxLength das caixas de texto.

Componente ErrorProvider.

Para não permitir a entrada de dados em branco podemos usar o componente ErrorProvider, que mostra que um determinado controle do formulário tem um erro.

Note que esse componente não aparece no formulário em tempo de execução e por isso, fica ancorado fora do formulário, assim como os componentes de acesso a dados. Confira na figura abaixo:

- 7) Insira o componente ErrorProvider e altere seu nome para EPCurso.



O método **SetError** do componente ErrorProvider lhe atribui um erro. Para validar ou não uma entrada, utilizamos o tratador de evento **Validating** da caixa de texto. Veja o exemplo abaixo.

A instrução que vem na cláusula "else", limpa o erro, passando-lhe uma string vazia como argumento.

Exemplos de utilização do ErrorProvider para as caixas de texto cur_codTextBox e cur_nomTextBox, usando o tratador do evento Validating:

```
private void cur_sglTextBox_Validating(object sender, CancelEventArgs e)
{
    if (cur_sglTextBox.Text.Trim().Length == 0)
    {
        EPCurso.SetError(cur_sglTextBox, "Entre com a sigla do curso");

    }
    else
        EPCurso.SetError(cur_sglTextBox, "");
}
```

```

private void cur_nomTextBox_Validating(object sender, CancelEventArgs e)
{
    if (cur_nomTextBox.Text.Trim().Length == 0)
    {
        EPCurso.SetError(cur_nomTextBox, "Entre com o nome do curso");

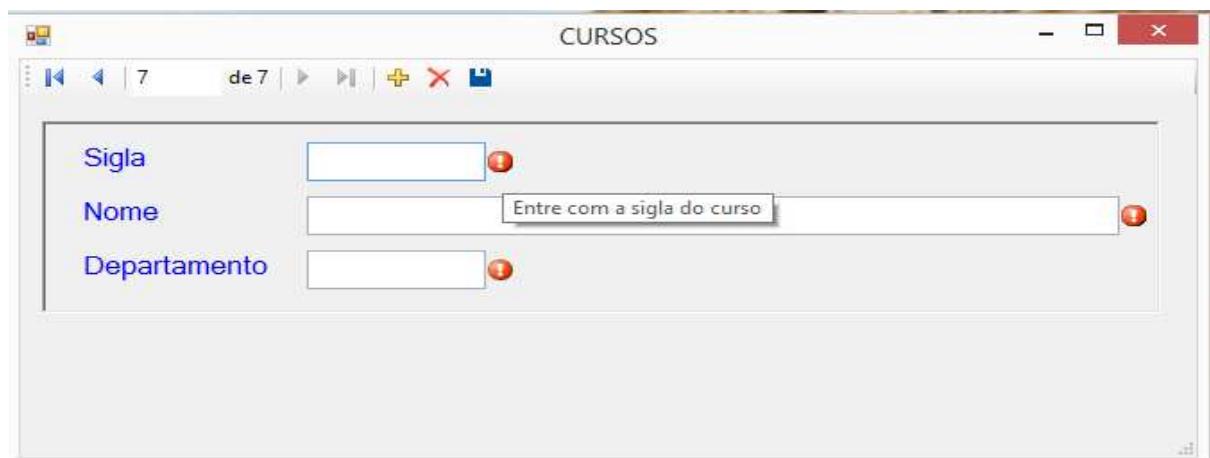
    }
    else
        EPCurso.SetError(cur_nomTextBox, "");
}

```

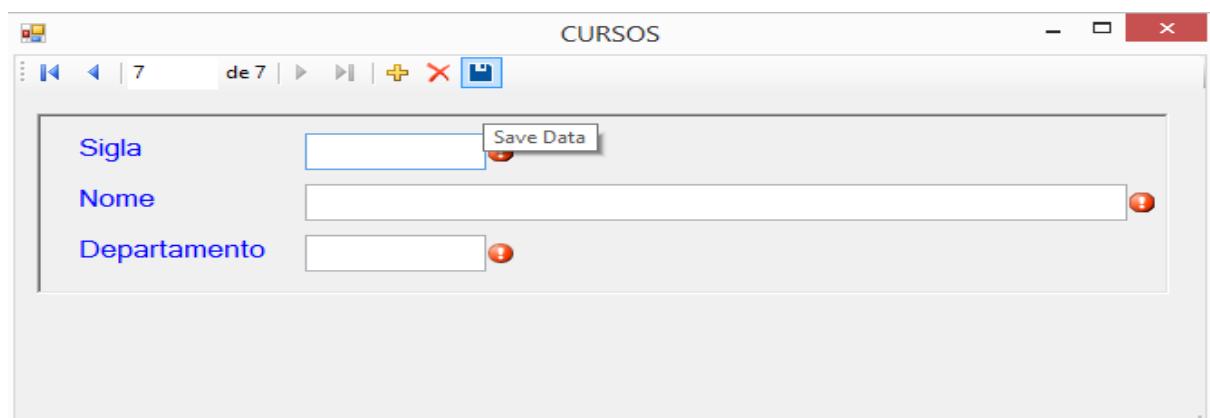
Com esse procedimento, se o usuário não informar o código do aluno, ao mudar o foco para outro controle, é acionado o ErrorProvider mostrando o seu ícone e exibindo a mensagem de erro.

Faça o mesmo para a sigla do departamento e execute o programa para verificar seu funcionamento.

Ao tentarmos inserir um registro em branco, ao mudar o foco das caixas de texto, vamos ter a seguinte interface:



Verifique que se ocorrer o click do mouse no ícone Salvar Dados ocorre erro de execução.



Para resolver isso, podemos criar uma função que faça a validação dos campos código, nome e departamento como no exemplo abaixo:

```
private bool valida()
{
    bool bStatus = false; // Se retornar true, há campos em branco

    if (cur_sglTextBox.Text == "")
    {
        EPCurso.SetError(cur_sglTextBox, "Entre com a sigla do curso");
        bStatus = true;
    }
    else
        EPCurso.SetError(cur_sglTextBox, "");

    if (cur_nomTextBox.Text == "")
    {
        EPCurso.SetError(cur_nomTextBox, "Entre com o nome do curso");
        bStatus = true;
    }
    else
        EPCurso.SetError(cur_nomTextBox, "");

    if (cur_dep_sglTextBox.Text.Trim().Length == 0)
    {
        EPCurso.SetError(cur_dep_sglTextBox, "Entre com a sigla do depto");
    }
    else
        EPCurso.SetError(cur_dep_sglTextBox, "");

    return bStatus;
}
```

Ao ser chamada essa função, é verificado se um dos campos está em branco. Caso isso aconteça, há o retorno da variável booleana bStatus, com valor True.

Podemos fazer a chamada dessa função no botão "Salvar Dados"  , da seguinte maneira:

```
private void cursosBindingNavigatorSaveItem_Click(object sender, EventArgs e)
{
    if (valida())
    {
        MessageBox.Show("Campos em branco");
        return;
    }
    cur_sglTextBox.Focus();

    this.Validate();
    this.cursosBindingSource.EndEdit();
    this.tableAdapterManager.UpdateAll(this.alunosDataSet);

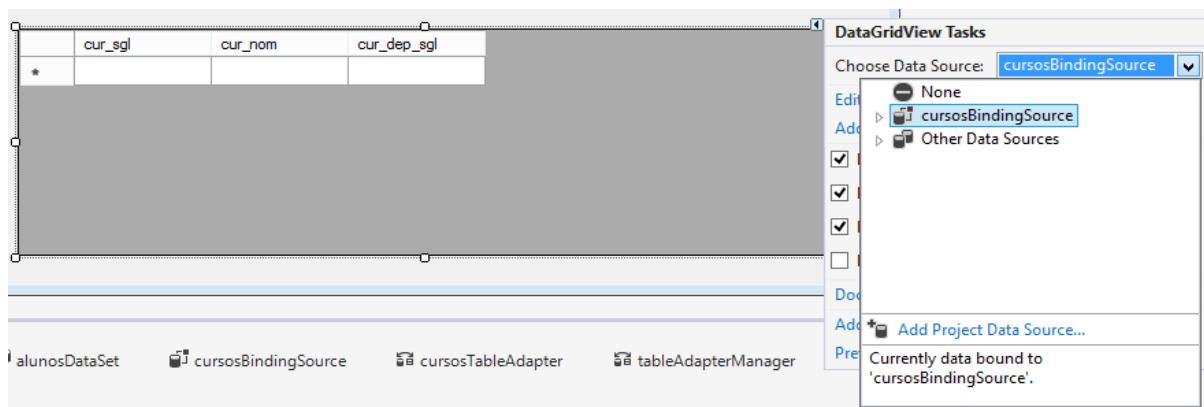
}
```

Componentes DataGridView

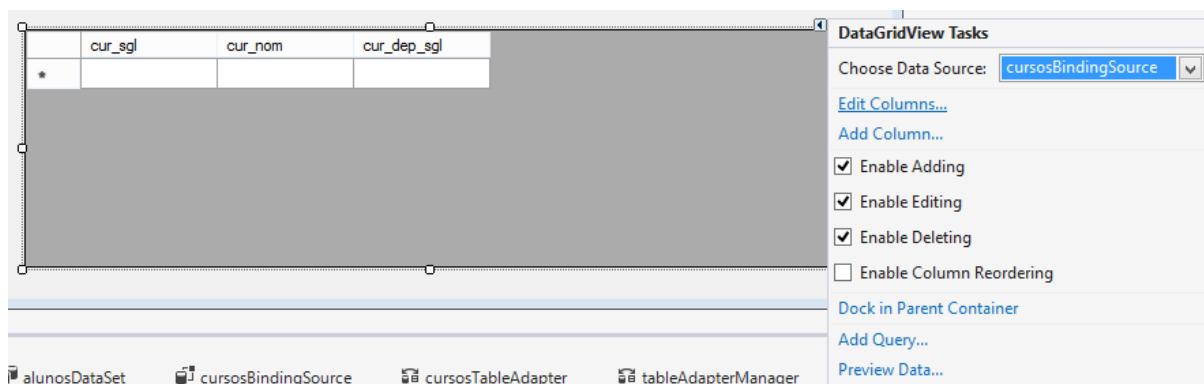
8) Primeiro vamos configurar o componente DataGridView.

Aumente um pouco o tamanho do painel de forma a suportar um controle DataGridView.

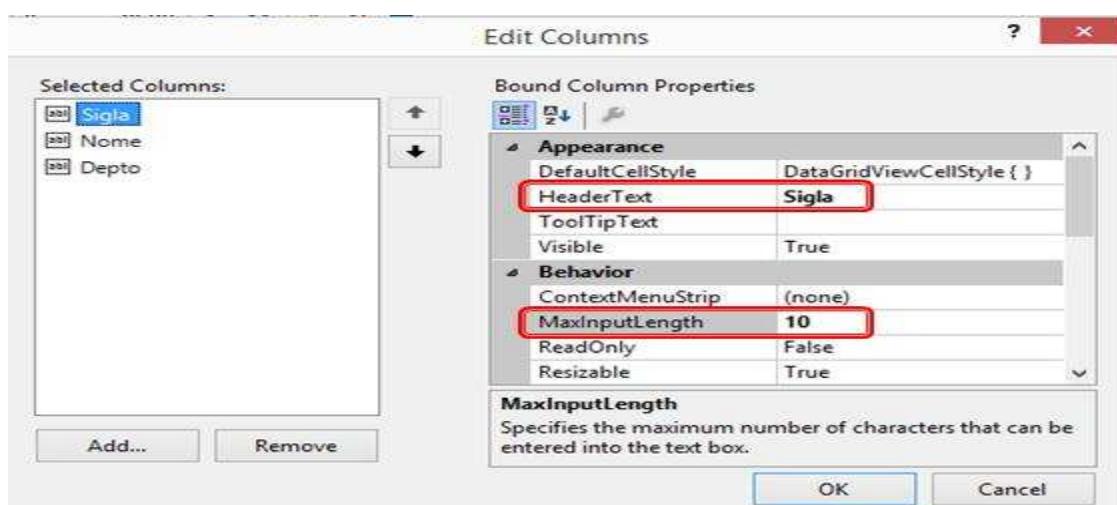
Dentro do painel insira o controle DataGridView. Configure conforme figura a seguir:



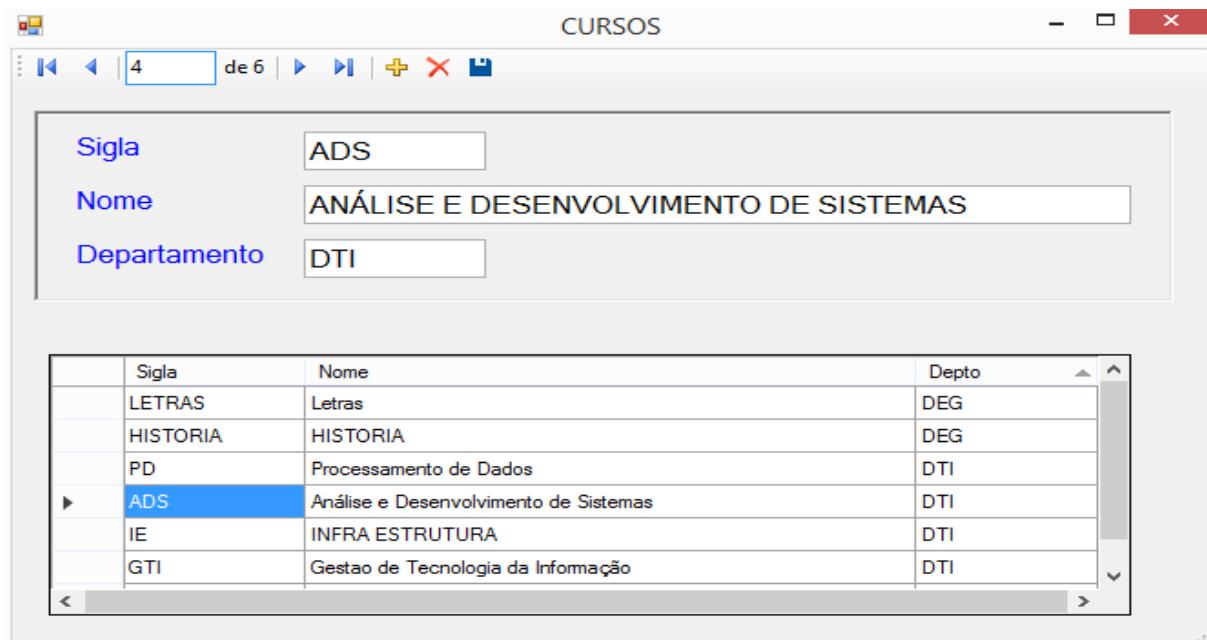
Selecione Editar Colunas



Edita as colunas alterando o cabeçalho do grid e tamanho máximo de caracteres permitido para cada campo.



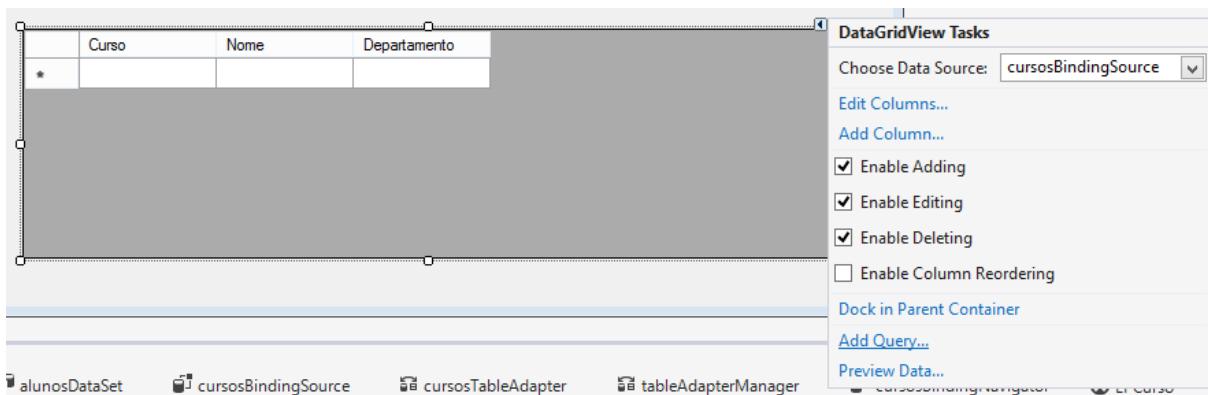
Execute para verificar o funcionamento do programa até agora - veja a figura a seguir:



- 9) Agora vamos adicionar uma consulta ao DataGridView. Clique em Add Query e monte a sua consulta conforme mostrado abaixo.

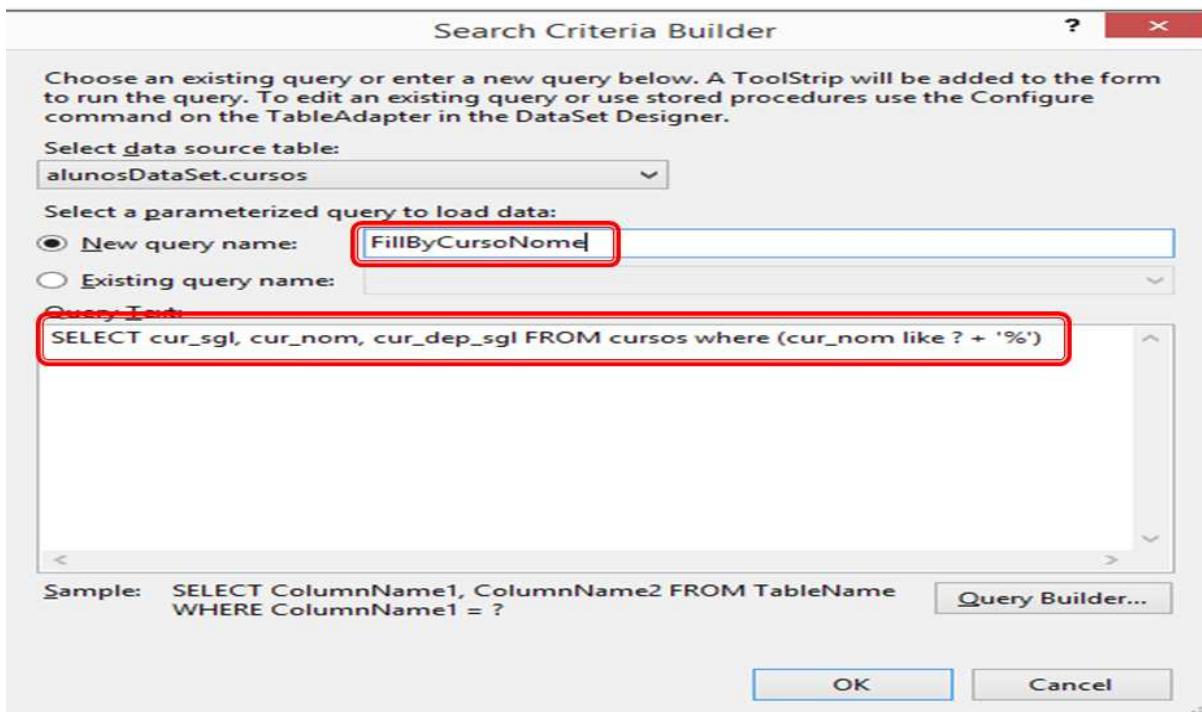
Consultas – Uso do Query:

Nas tarefas do DataGridView click em Adicionar Consultas.

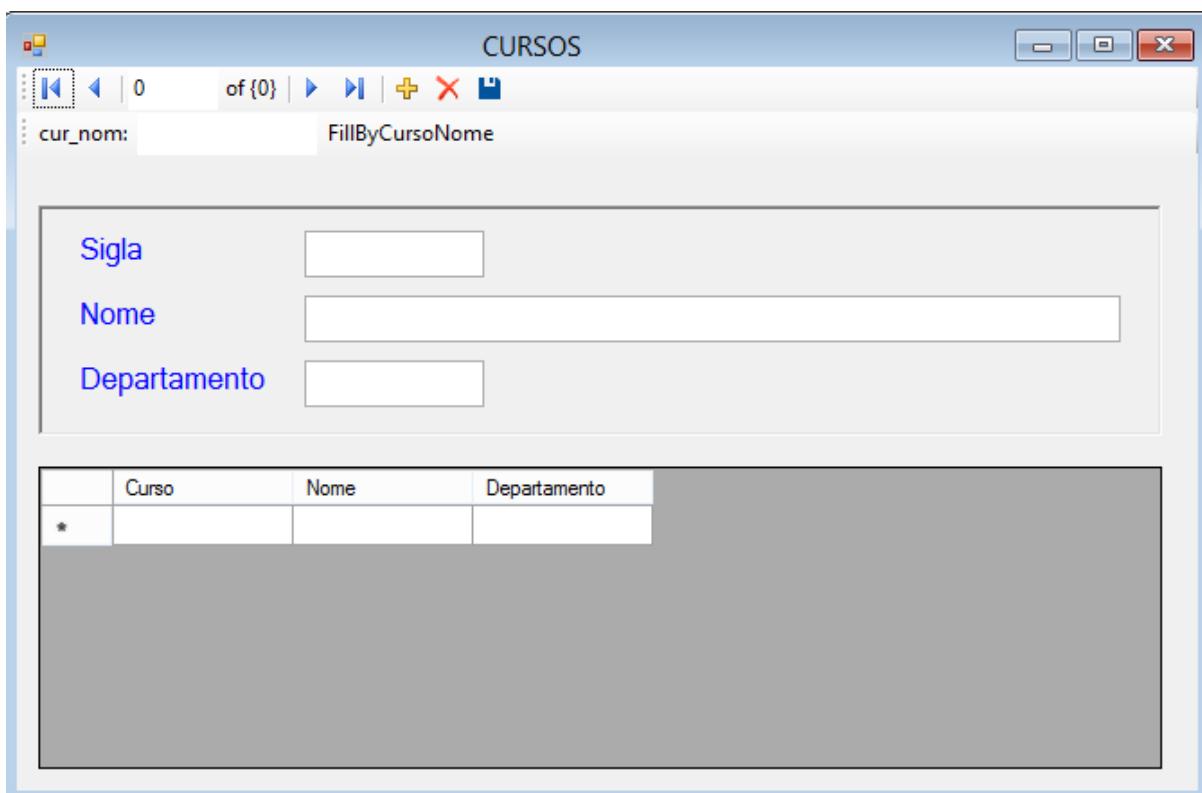


É exibida a janela a seguir, para configurar a consulta. Altere para ficar como abaixo:
Altere o item New query name para FillByCursoNome, altere o Query Text para:

```
SELECT cur_sgl, cur_nom, cur_dep_sgl FROM cursos where (cur_nom like ? + '%')
```

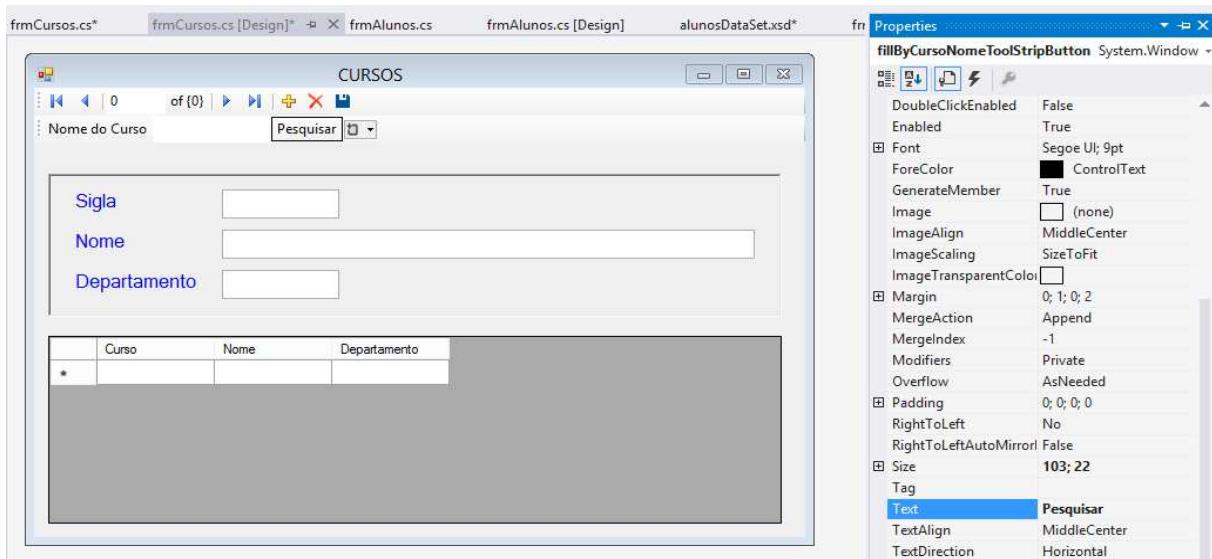
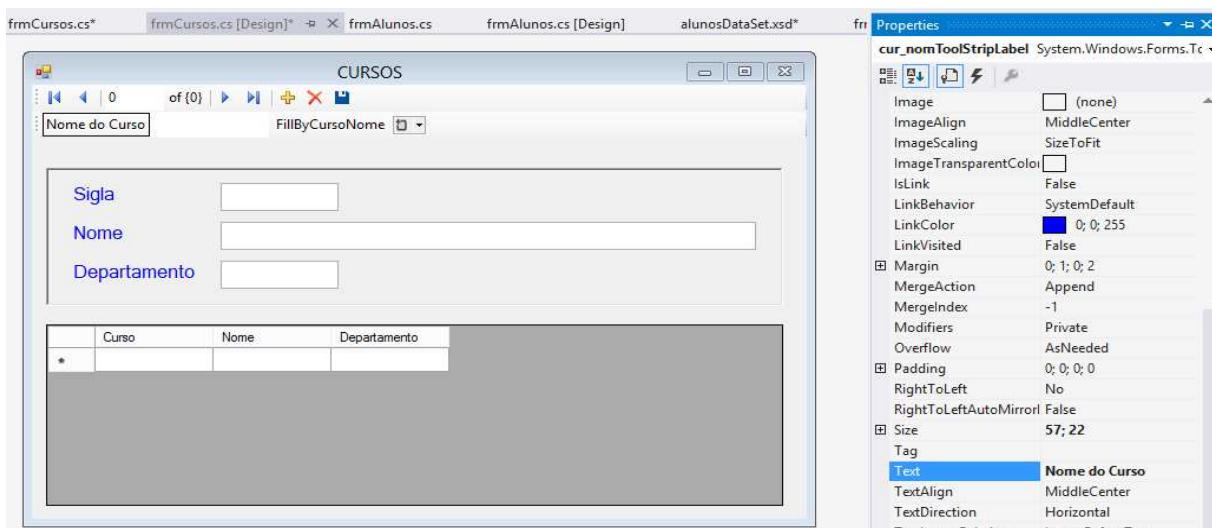


Verifique se está de acordo com a figura acima e pressione OK p/ montar a consulta.

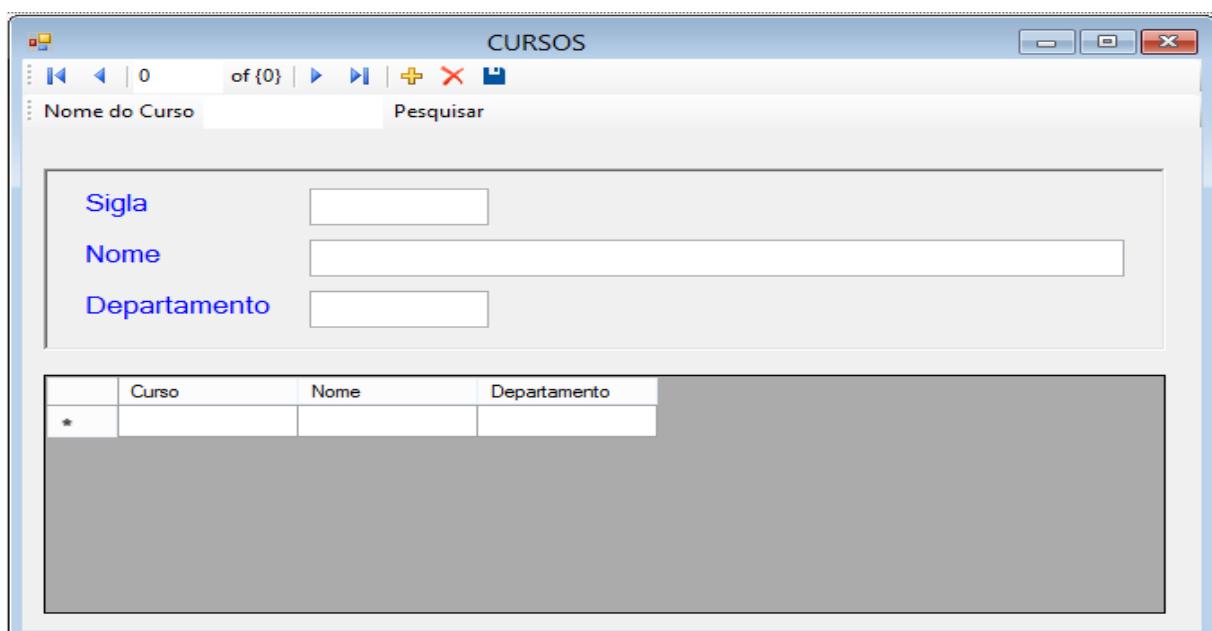


Altere o Text dos labels Cur_Nom e FillByCursoNome para nomes mais sugestivos, como por exemplo Nome do Curso e Pesquisar.

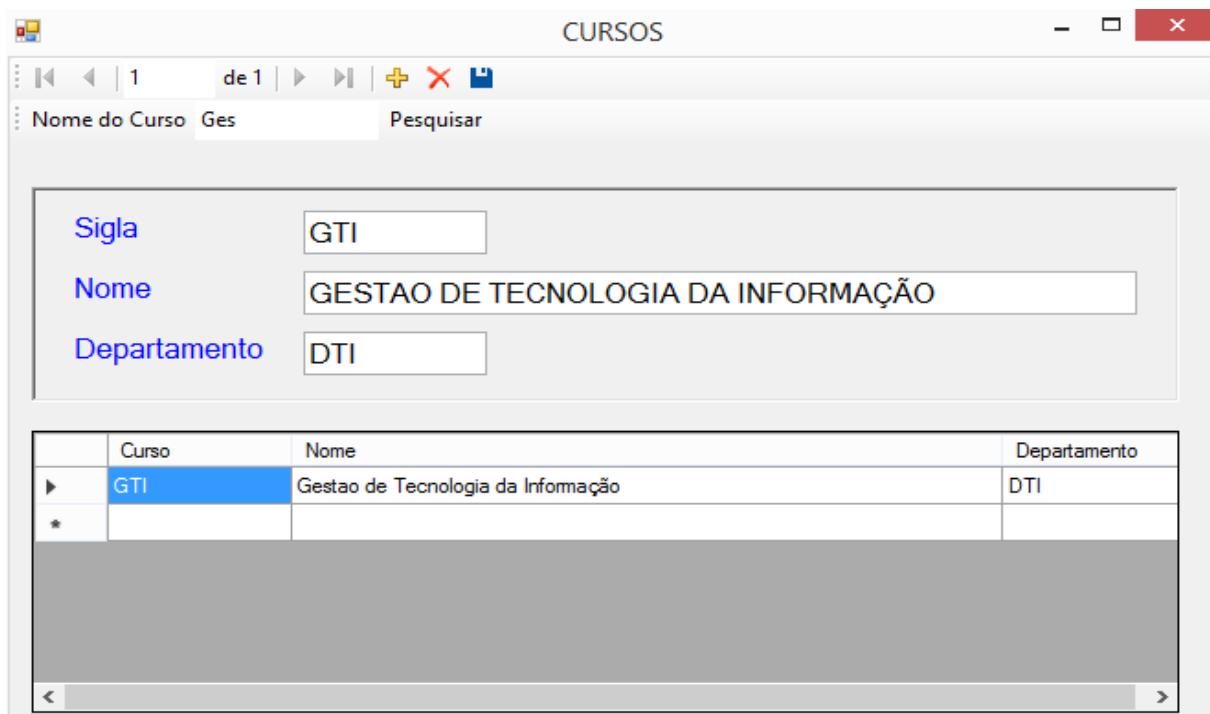
APOSTILA DE VISUAL C#



Execute e teste o programa. Deverá ficar mais ou menos como a figura a seguir:



Note que você pode digitar no campo Nome do Curso, apenas um trecho inicial do nome a pesquisar, que a pesquisa encontrará o nome desejado.

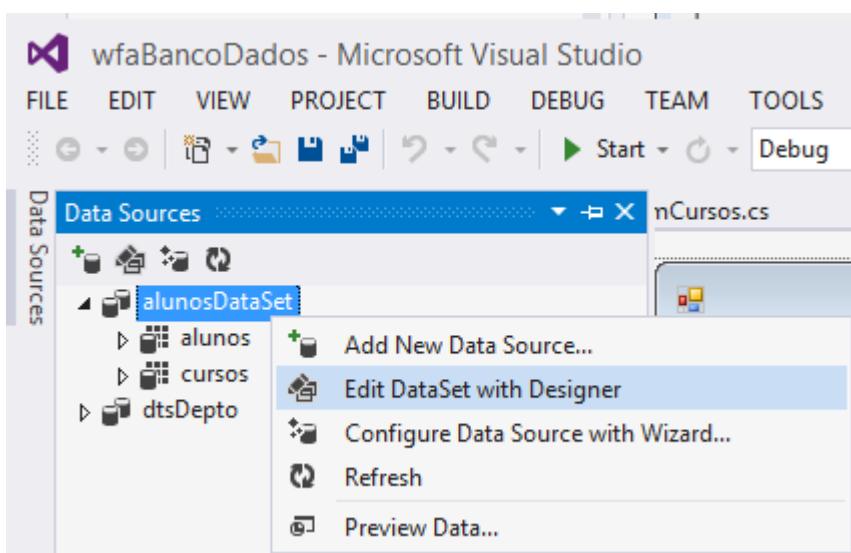


10) Execute e teste a aplicação.

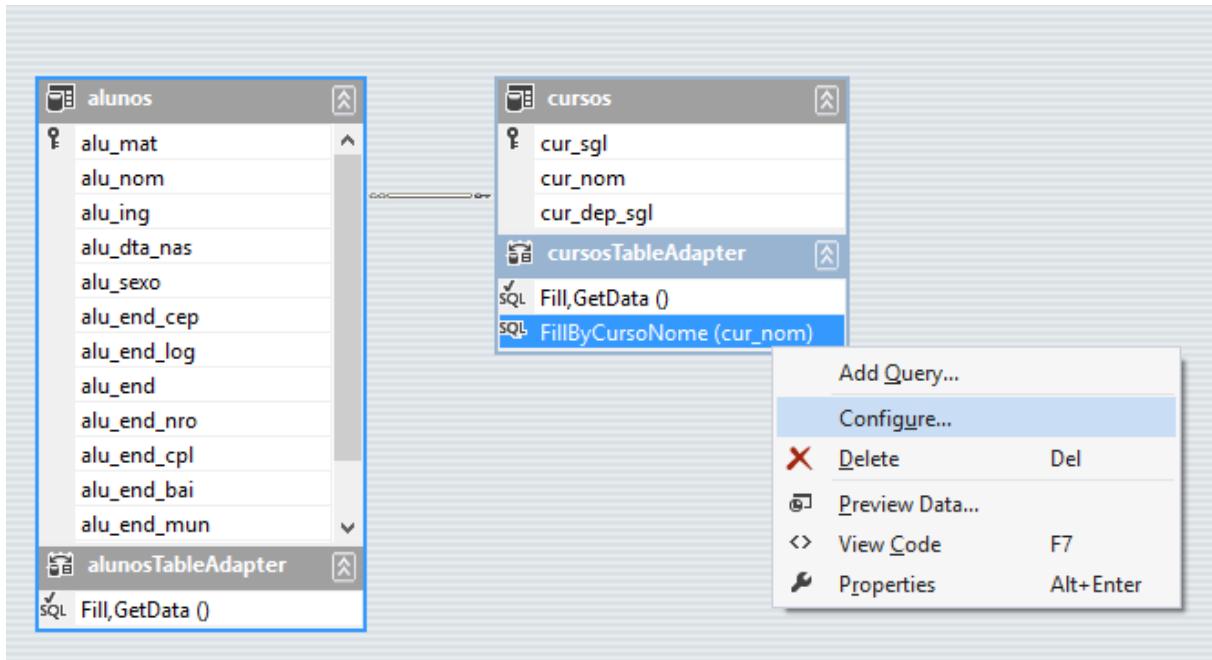
Alteração de uma query

Caso você queira modificar uma query existente, proceder da seguinte maneira:

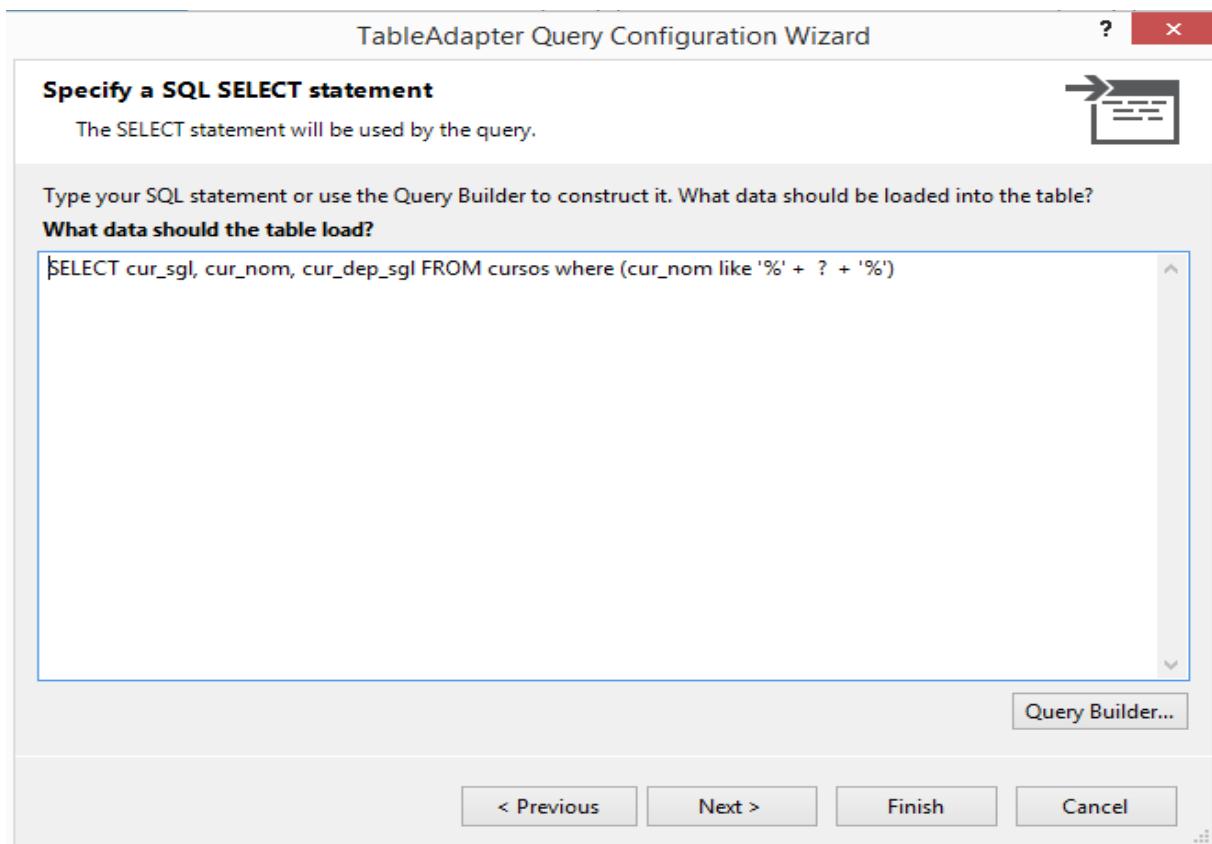
1 – Click em Data Sources e click com o botão direito do mouse no Dataset a ser alterado, no nosso caso alunosDataset. Click em Editar Conjunto de Dados com o Designer - veja a figura:



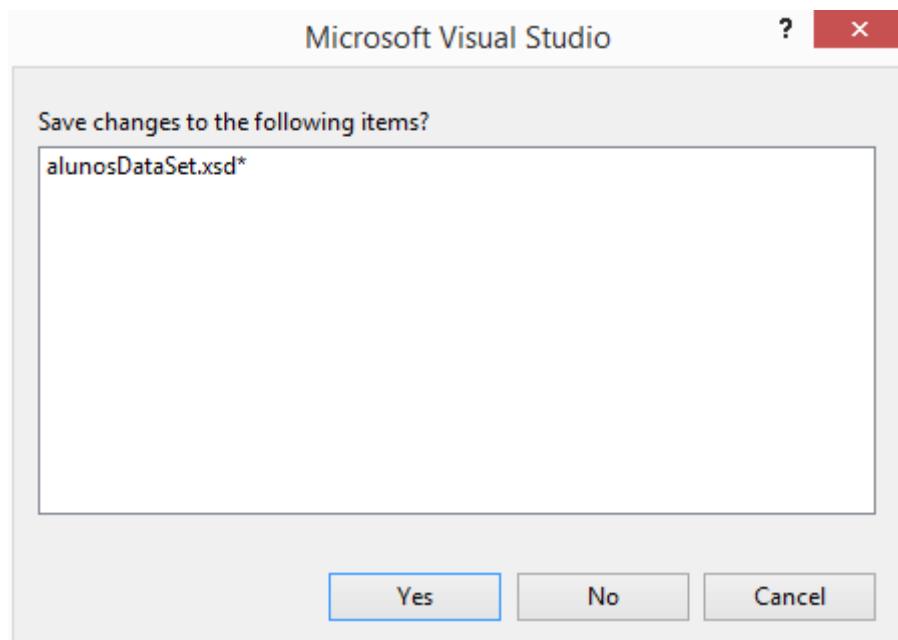
2 - Na janela que abrir, selecione a query que lhe interessa modificar, dê um click com o botão direito e em seguida click em configure - veja a figura abaixo:



3 - Faça as modificações que lhe interessarem e click em Concluir. Veja a figura a seguir:



Ao fechar a janela `scaDataSet.xsd*`, o editor pergunta se você quer salvar as modificações. Responda Sim.



AULA 11*Exercício: projeto SCA*

Crie o banco de dados chamado sca, com as seguintes especificações:

Departamentos

dep_sgl (PK)	Texto (10)	Sigla do departamento
dep_nom	Texto (70)	Nome do departamento

Cursos

cur_sgl (PK)	Texto (10)	Sigla do curso
cur_nom	Texto (70)	Nome do curso
cur_dep_sgl (FK)	Texto (10)	Sigla do departamento

Disciplinas

dis_cod (PK)	Inteiro	Codigo da disciplina
dis_sgl	Texto (10)	Sigla da disciplina
dis_nom	Texto(70)	Nome da disciplina
dis_crg_hor	Inteiro	Carga horária
dis_cur_sgl (FK)	Texto (10)	Sigla do curso

Alunos

alu_mat (PK)	Texto (10)	Matricula do aluno
alu_nom	Texto(70)	Nome do aluno
Alu_ing	Texto (6)	Ano/Semestre de Ingresso AAAA/S
alu_dta_nas	Date abreviada	Data de nascimento
alu_CPF	Texto(15)	CPF do aluno
alu_Sexo	Texto(1)	Sexo F (feminino) M (masculino)
Alu_email	Texto (70)	Email do aluno
alu_end_cep	Texto(10)	CEP do aluno (99999-999)
alu_end_log	Texto(10)	Logradouro
alu_end	Texto(70)	Endereço
alu_end_nro	Inteiro	Nro
alu_end_cpl	Texto(20)	Complemento do endereço
alu_end_bai	Texto(70)	Bairro
alu_end_cid	Texto(70)	Cidade
alu_end_ufe (FK)	Texto(2)	Sigla do estado
alu_cur_sgl (FK)	Texto(10)	Sigla do curso

Matriculas

mat_ano (PK)	Inteiro	Ano da matricula
mat_sem (PK)	Inteiro	Semestre da matricula
mat_alu_cod (PK)	Texto (10)	Matricula do aluno
mat_dis_cod (PK)	Inteiro	Codigoda disciplina

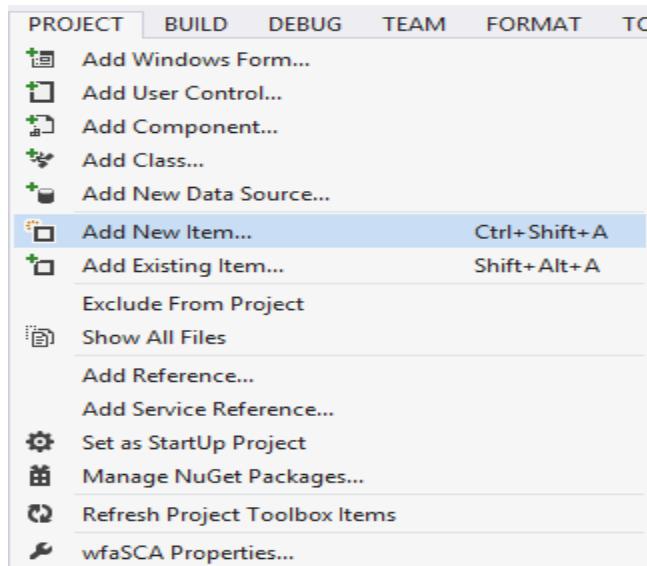
mat_avl_001	Double	Nota da P1
mat_avl_002	Double	Nota da P2
mat_trb_001	Double	Média dos Trabalhos
mat_med	Double	Média Final
mat_con	Texto (1)	Conceito Final
mat_fal	Inteiro	Total de Faltas

estados (UFE)

ufe_sig (*)	Texto (2)	Sigla do estado
ufe_nom	Texto (5)	Nome do estado

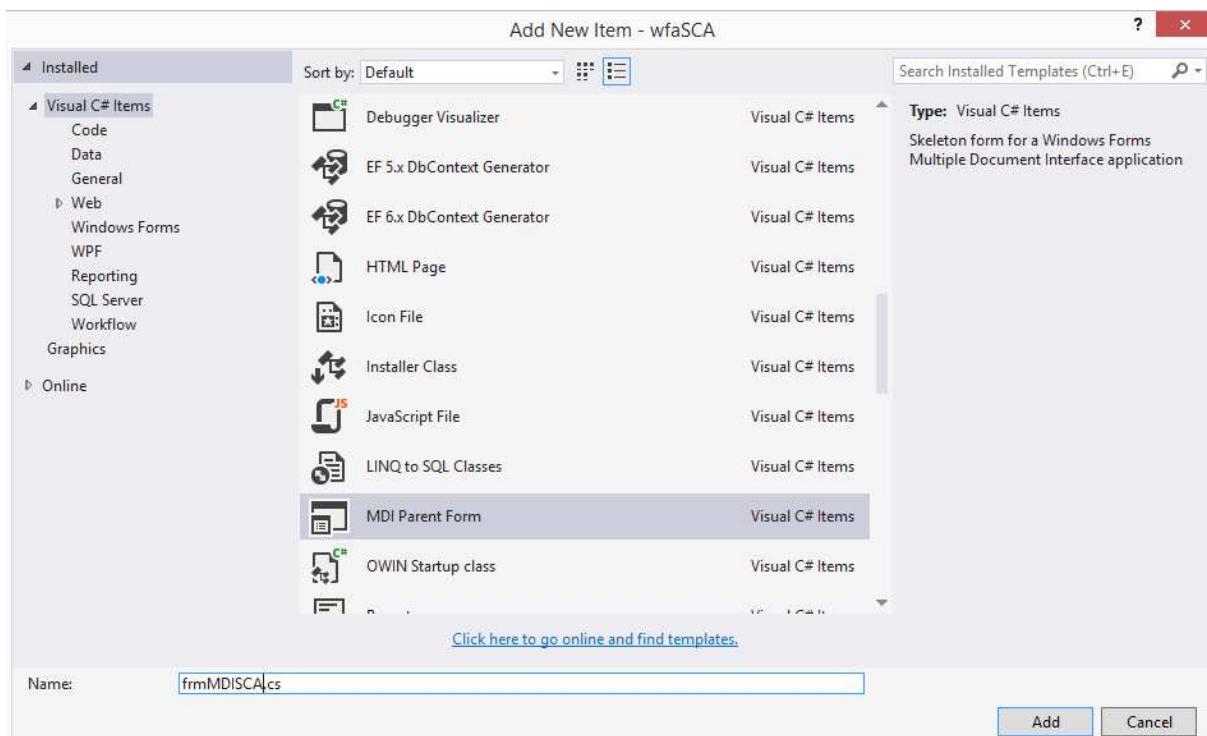
1. Iniciar um novo projeto chamado wfaSCA (sistema de controle acadêmico).
2. Incluir um novo ítem ao projeto com o nome de *frmMDISCA* e utilize o template Formulário Pai MDI.

Acompanhe as figuras abaixo:

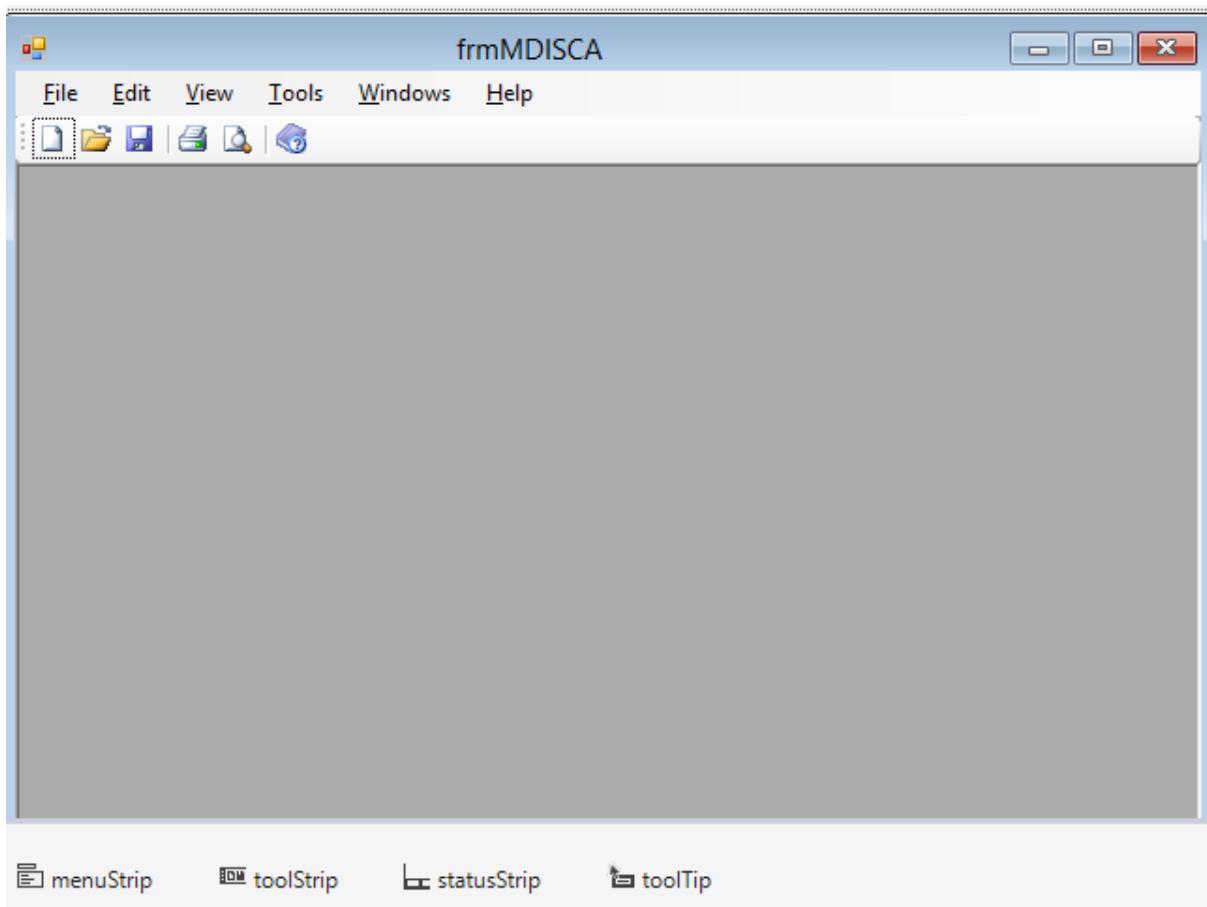


Mude o “Nome” do nome item para *frmMDISCA* e click em adiconar.

APOSTILA DE VISUAL C#

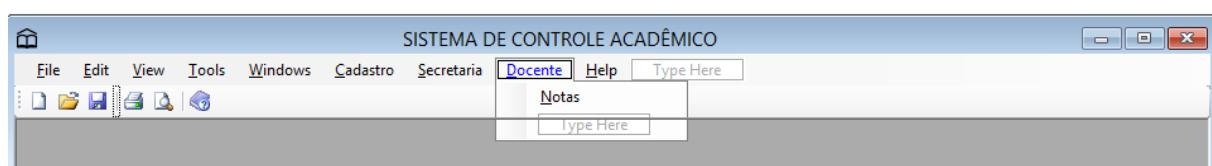
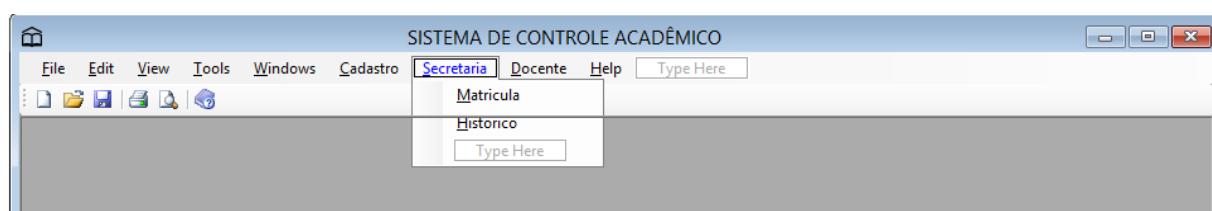
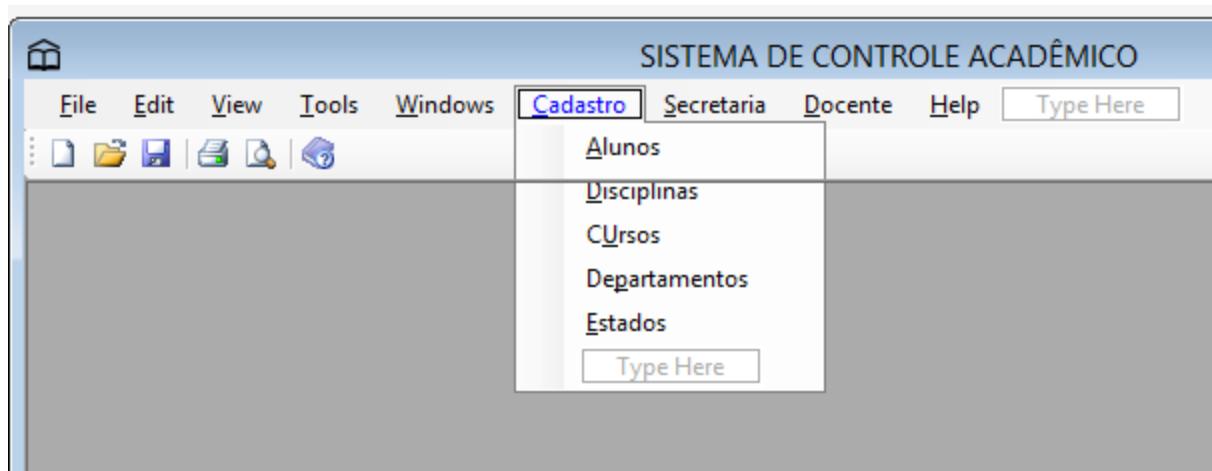


Observe os controles que são inseridos no formulário.

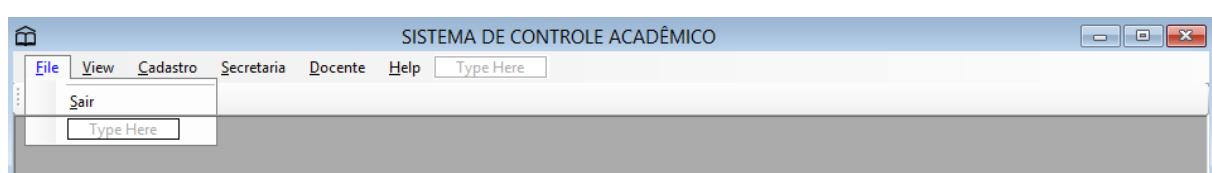


Altere a propriedade do text do formulário para Sistema de Controle Acadêmico, adicione um ícone no formulário e altere a propriedade do formulário WindowState para Maximized. Na propriedade BackgroundImage, atribuir a figura CEETEPS.jpg e na propriedade BackgroundImageLayout selecionar Stretch.

Inclua o menu Cadastro, Secretaria e Docente com os itens de acordo com as figuras abaixo.



Exclua os menus Edit, Tools e Windows. No menu File deixe apenas o item Sair.



3. Antes de fazer o primeiro teste vamos alterar o formulário principal a ser chamado. Para isso selecione o Program.cs do Gerenciador de Soluções, e altere o primeiro formulário a ser chamado para frmMDISCA.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace wfaSCA
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}

```

4. Elaborar os cadastros do sistema de controle acadêmico levando em consideração as seguintes restrições:

4.1. Cadastro de Estados

- A sigla do estado não pode estar em branco/nulo. Deve ser único. A sigla do estado deve estar em maiúscula.
- Nome do estado é obrigatório.
- Caso o estado esteja relacionado em alguma tabela não pode ser excluído e o sistema deve enviar uma mensagem ao usuário.
- Deve-se permitir a consulta pelo nome do estado.

4.2. Cadastro de Departamentos

- A sigla do departamento não pode estar em branco/nulo. Deve ser único. A sigla do departamento deve estar em maiúscula.
- Nome do departamento é obrigatório.
- Caso o departamento esteja relacionado em alguma tabela não pode ser excluído e o sistema deve enviar uma mensagem ao usuário.

- d) Deve-se permitir a consulta pelo nome do departamento.

4.3. Cadastro de Cursos

- a) O código do curso não pode estar em branco/nulo. Deve ser único. O código do curso deve estar em maiúscula.
- b) Nome do curso é obrigatório.
- c) Caso o curso esteja relacionado em alguma tabela não pode ser excluído e o sistema deve enviar uma mensagem ao usuário.
- d) Deve-se permitir a consulta pelo nome do curso.

4.4. Cadastro de Disciplinas

- a) O código da disciplina não pode estar em branco/nulo. Deve ser único. A sigla da disciplina deve estar em maiúscula.
- b) Nome da disciplina é obrigatório.
- c) A carga horária é obrigatória e deve aceitar apenas números inteiros.
- d) O curso deve estar cadastrado previamente.
- e) Caso a disciplina esteja relacionada em alguma tabela não pode ser excluída e o sistema deve enviar uma mensagem ao usuário.
- f) Deve-se permitir a consulta pelo código e nome da disciplina.
- g) Deve-se permitir a consulta pela sigla do curso.

4.5. Cadastro de Alunos

- a) A matrícula não pode estar em branco/nulo e deve permitir a entrada de apenas número
- b) O CPF deve ser válido conforme regra de validação entregue e deve ser único
- c) Nome do aluno é obrigatório
- d) O curso deve estar cadastrado previamente
- e) O estado deve estar cadastrado previamente.
- f) Caso o aluno esteja relacionado em alguma tabela não pode ser excluído e o sistema deve enviar uma mensagem ao usuário.
- g) Deve-se permitir a consulta pela matrícula e pelo nome do aluno.

4.6. Matrícula

- a) Listar os alunos pelo curso e permitir a matrícula dos alunos nas disciplinas que ele pode cursar. As disciplinas que ele já se matriculou no ano/semestre não deve aparecer na relação de disciplinas para matrícula.
- b) Caso haja lançamento de notas do aluno nas disciplinas, esta matrícula não pode ser excluída.
- c) Permitir somente a matrícula de alunos previamente cadastrados no curso.

4.7.Histórico

- a) Permitir a consulta das notas dos alunos pelo curso
- b) Permitir a consulta das notas dos alunos pelo nome do aluno
- c) Permitir a consulta das notas dos alunos pela matrícula do aluno
- d) Permitir a consulta das notas pela disciplina

4.8.Conceitos

- a) O lançamento das notas deve ser feito por disciplina do ano/semestre
- b) A media final deve ser calculada utilizando o seguinte critério: $(P1*2) + (P2*3) + T*5)/10$
- c) O conceito deve ser atribuído utilizando a tabela abaixo:

Média Final	Conceito
< 6,0	C
$\geq 6,0$ e $< 7,5$	B
$\geq 7,5$ e $< 9,0$	A
$\geq 9,0$	E
Faltas ≥ 20	F

5. Faça os testes e apresente ao professor.

Obs:

O tempo de duração previsto é de três semanas (12 aulas) e vale de 0 a 30 pontos.

Deve ser feita a apresentação do sistema pelo aluno na data marcada pelo professor, quando será entregue um CD ou DVD com todos os arquivos necessários ao trabalho.

Bibliografia

SHARP, John. Microsoft Visual C# 2008 Passo a Passo, São Paulo, Bookman, 2008.

STELLMAN, Andrew & GREENE, Jeniffer. Use a Cabeça C#. Editora Alta Books Ltda. 2008.