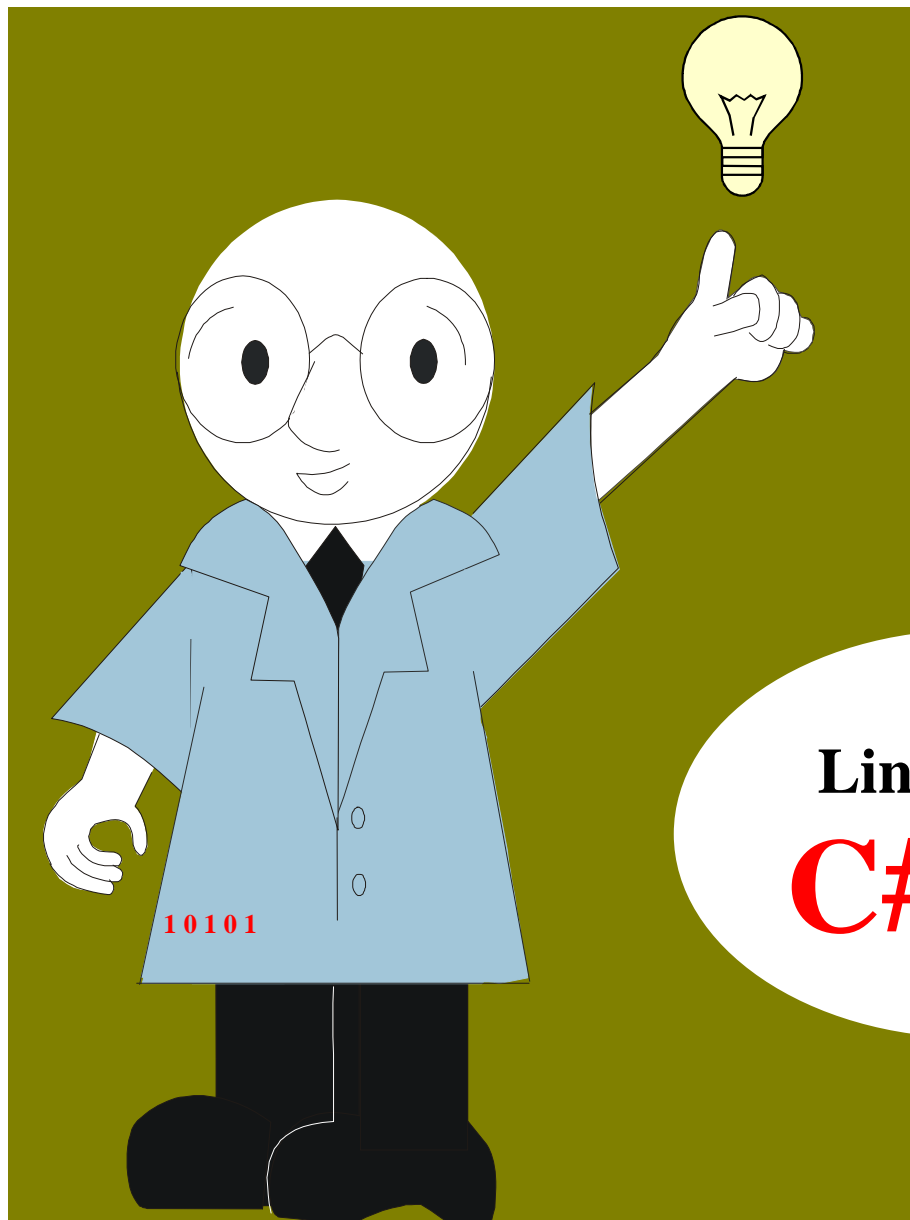


PROF^a. ANGELINA V.S. MELARÉ

ANGELINAMELARE@GMAIL.COM



Linguagem
C#.net

LINGUAGEM DE PROGRAMAÇÃO

CENTRO PAULA SOUZA

GOVERNO DO ESTADO DE
SÃO PAULO

2012



Sumário

I- Introdução a Linguagem C#	4
Glossário.....	4
Recursos do C# que facilitam o desenvolvimento	5
Diferença entre Projeto e Solução	5
Teclas de Atalho	5
II- Desenvolvimento de um Projeto.....	6
III- Ambiente de Desenvolvimento do C#.....	7
Solution Explorer	8
Caixa de Ferramentas	9
Janela de Propriedades	10
IV- Salvar toda a Solução	11
Extensões de arquivos	12
V- Exemplos	13
Exemplo de Formulário:.....	13
Exemplo de código:.....	13
VI- Namespace	14
Exemplo do Namespace	16
VII- Primeiro Projeto	17
VIII- Variáveis	23
Sintaxe:.....	24
Exemplo de declaração e atribuição	24
Alerta:.....	24
IX- Conversão de Tipos	25
Formas de conversão:	25
Exemplos de conversão:	26
X- Segundo Projeto.....	27
XI- Tratamento de String.....	30
XII- Terceiro Projeto.....	31
XIII- Quarto Projeto	36
As exceções podem ser tratadas individualmente ou de forma genérica.	37
XIV- Quinto Projeto	40
Program.cs.....	40
MenuStrip.....	41
Configuração dos formulários para a existência de uma única janela (Formulário Principal)	41
ErrorProvider-	43
XV- Estrutura foreach().....	46
Dica: Criar um método através do código existente.	46
XVI- Classes.....	48
Criação de uma classe	49

Exemplo:	50
[modificador de acesso] class <identificador>	50
Atributos de uma classe	51
Modificadores de Acesso	51
Encapsulamento.....	52
Métodos	52
Propriedades	53
Exemplo da Classe Pessoa.....	53
Construtores.....	55
Destrutor	56
Variáveis.....	56
XVII- Herança	57
XVIII- Polimorfismo	61
XIX- Classes abstratas	63
XX- Selando as classes.....	64
XXI- Compartilhar dados entre os formulários e Formulário com Validação de Usuário e Senha	65
XXII- Armazenamento de Dados	69
XXIII- Conexão com o Banco de Dados	77
XXIV- Apêndice	81
XXV- Bibliografia Utilizada.....	84

I- Introdução a Linguagem C#

A linguagem C#.Net é uma linguagem de programação orientada a objetos e eventos, com recursos de componentes gráficos e de multimídia, tratamento de exceções, strings, processamento de arquivos e banco de dados, além de implementação de aplicativos para internet.

Essa linguagem faz parte da plataforma .Net, que oferece a portabilidade dos programas, permitindo que os aplicativos residam e se comuniquem em diversas plataformas e dispositivos.

A linguagem C# foi desenvolvida para a plataforma .Net pela equipe de Anders Hejlsberg e Scott Wiltamuth da Microsoft. Para os programadores em C, C++ a migração para essa linguagem é mais fácil, com familiaridade na sintaxe e estruturas internas.

Os programas podem ser desenvolvidos na IDE do Visual Studio .Net onde pode-se criar, documentar, depurar, executar e testar os aplicativos num mesmo ambiente.

Glossário

C# - Linguagem de programação orientada a objetos. Pronuncia-se “ci-sharp” ou “cê-sharp”.

.Net – pronuncia-se “dot-net” ou “ponto net” é um novo modelo(estratégia) de desenvolvimento de software criado pela Microsoft.

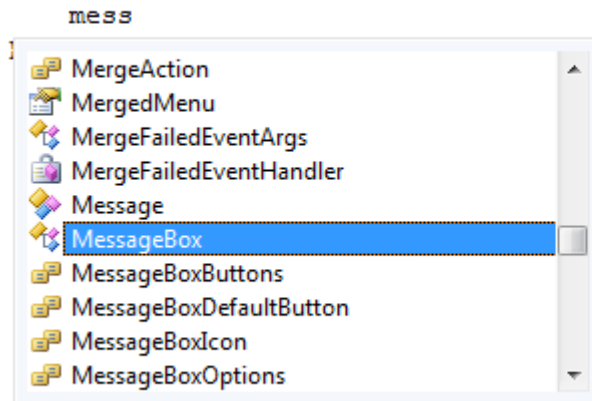
WWW – world wide web

IDE- Integrated Development Environment – Ambiente integrado de desenvolvimento.

Recursos do C# que facilitam o desenvolvimento

IntelliSense permite gerar códigos de forma automática, evitando perda de tempo e erros na digitação. Para acionar pressione CTRL + Espace.

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



Os comentários podem gerar arquivo XML de documentação da solução.

Diferença entre Projeto e Solução

O Projeto é constituído por um ou vários grupos de arquivos relacionados (códigos, formulários, imagens, classes). A Solução é constituída por um ou vários projetos, e representa o produto final a ser entregue ao cliente.

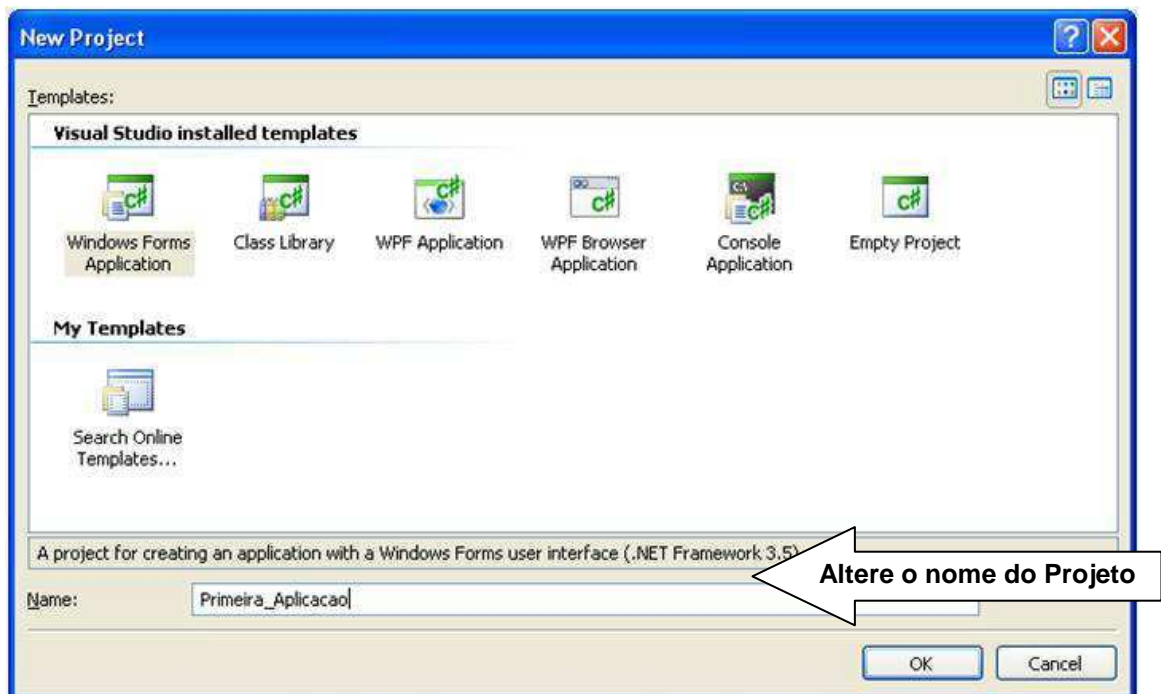
Teclas de Atalho

- F4- exibe a Janela de Propriedades
- F7- exibe o código do formulário atual
- F5- executa o projeto
- F10 e F11 – usados para debugar

II- Desenvolvimento de um Projeto

Ao iniciar o Microsoft Visual C#, irá aparecer a página inicial onde deve-se escolher no menu a opção FILE- New Project. Quando aparecer a caixa de diálogo (figura abaixo) deve-se selecionar o tipo de projeto que deseja fazer, como Windows Forms Application, Console Application, Class Library, entre outros.

O Windows Forms Application é usado para desenvolver aplicações Windows, que possuem elementos gráficos (botões, menus, caixas de texto) com os quais o usuário se interage. Essa opção é a padrão ao iniciar um novo projeto, devendo apenas mudar o nome e clicar no botão OK



O arquivo que será gerado é o Primeira_aplicacao.csproj (extensão cproj – projeto C#).

III- Ambiente de Desenvolvimento do C#

A seguir aparece a tela de desenvolvimento do projeto.

Barra de Ferramentas

Mostra a estrutura da solução, listando todos os arquivos criados e vinculados a ela.

Formulário: área principal da aplicação, que será visualizada pelo usuário e onde serão colocados os componentes gráficos (botões, caixas de textos, etc).

Guias: a janela que aparece é a Guia Ativa. Pode-se fazer intercalação com outras

Caixa de Ferramentas: possui os componentes (controles) que podem ser usados em uma aplicação.

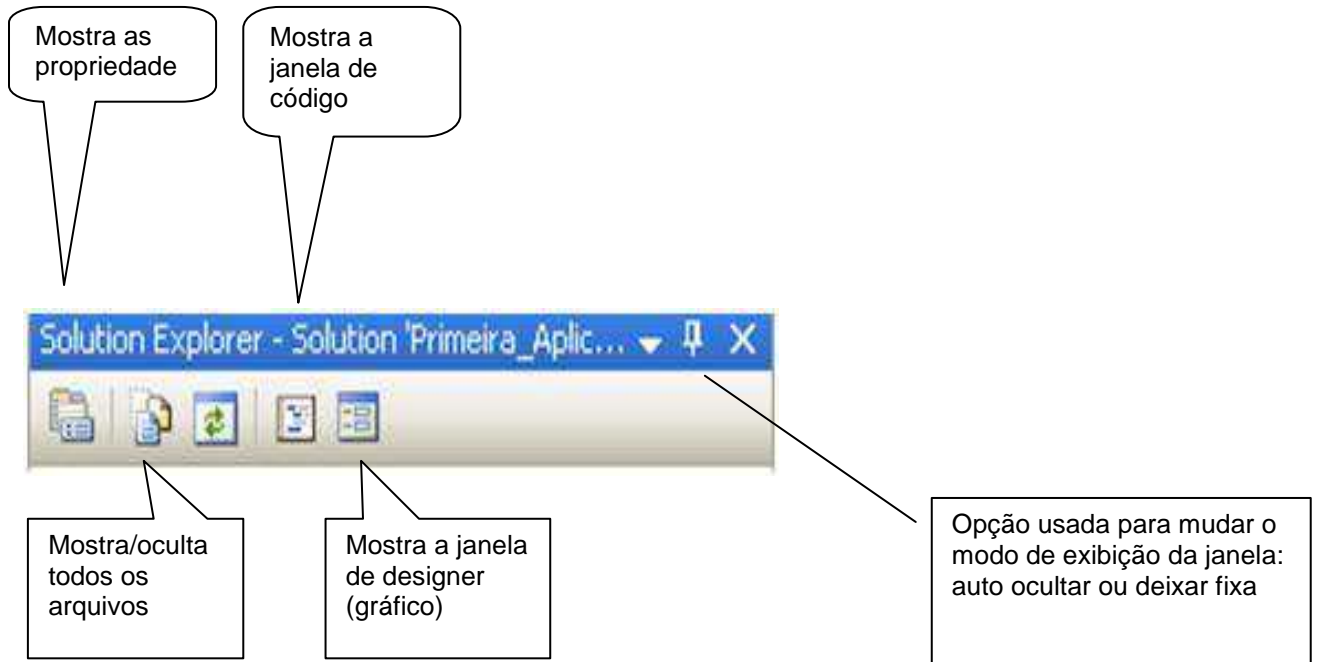
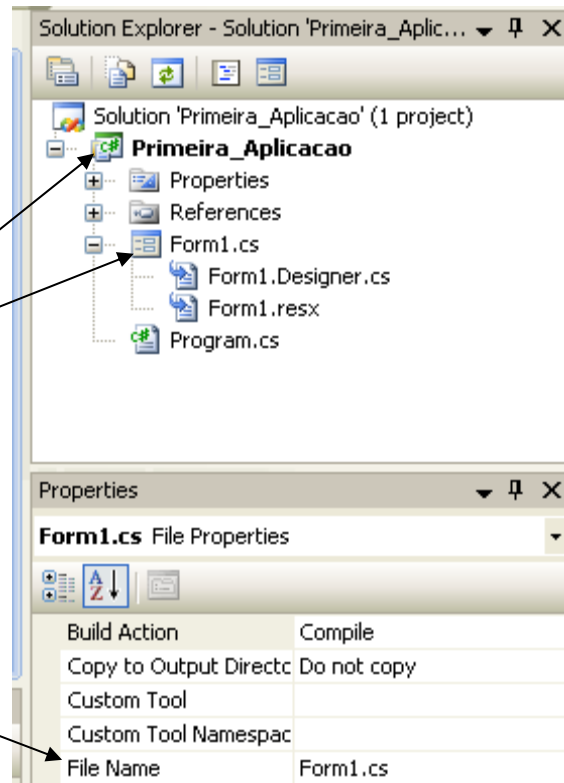
Janela de Propriedades usada para alterar as informações do controle selecionado no momento ou determinar o evento do mesmo

Solution Explorer

No Solution Explorer se gerencia todos os projetos, classes e formulários da solução (aplicação) desenvolvida.

É possível ver o nome dado para o primeiro projeto (que automaticamente ficou para a solução)

O nome inicial do formulário é Form1, que pode ser alterado diretamente clicando sobre o nome Form1.cs e alterando na Janela de Propriedades.



Dica:

Quando as janelas forem alteradas de maneira que não se consiga voltar ao normal, clique na opção Windows do menu, item Reset Windows Layout.

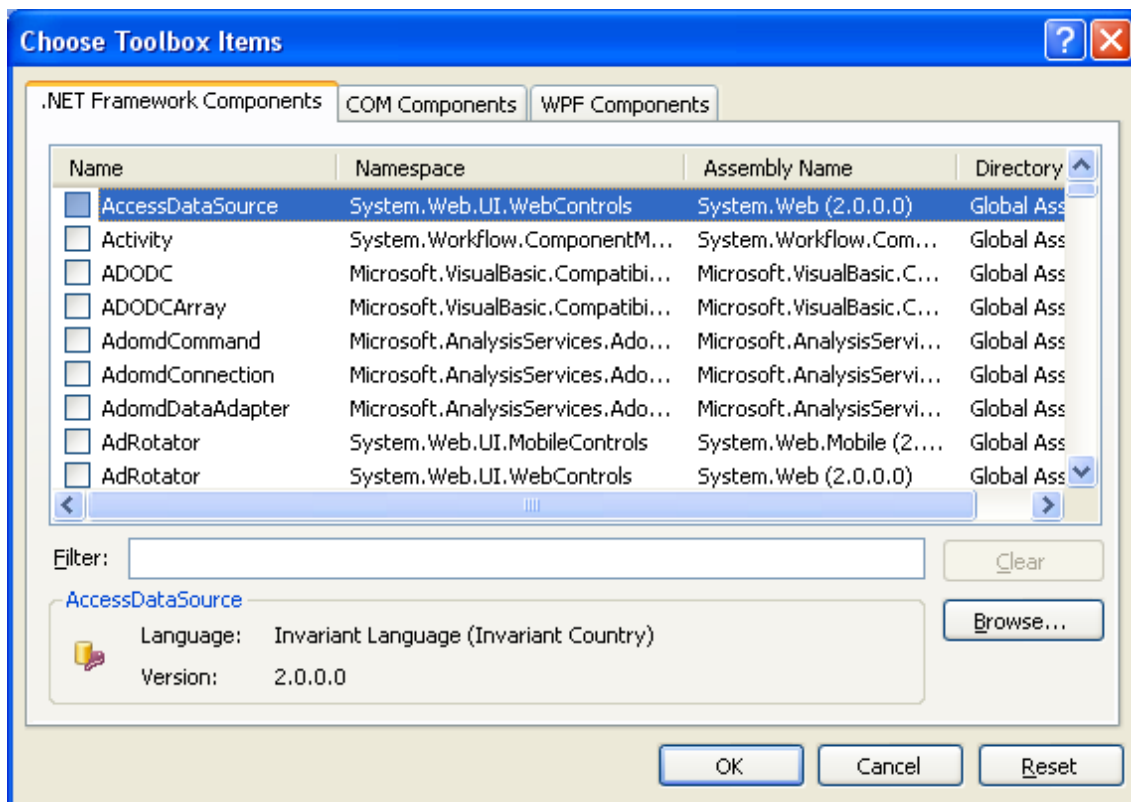
Curiosidade:

O sinal de + que oculta uma lista se chama colapsado(expandir) e o sinal de – é descolapsado.

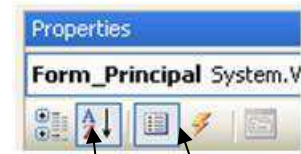
Caixa de Ferramentas

Na Caixa de Ferramentas são mostrados vários controles (componentes) que poderão ser usados. Esses controles estão relacionados em grupos.

Se for necessário pode-se incluir novos componentes, clicando com o botão da direita do mouse sobre a caixa de Ferramentas e escolhendo a opção Choose Toolbos Item. Quando for acessar essa opção pela primeira vez é normal demora um pouco para carregar todas as opções.

**Dica:**

Quando for selecionar vários componentes do mesmo tipo pressione e segure a tecla CTRL antes de selecionar o componente.




Janela de Propriedades

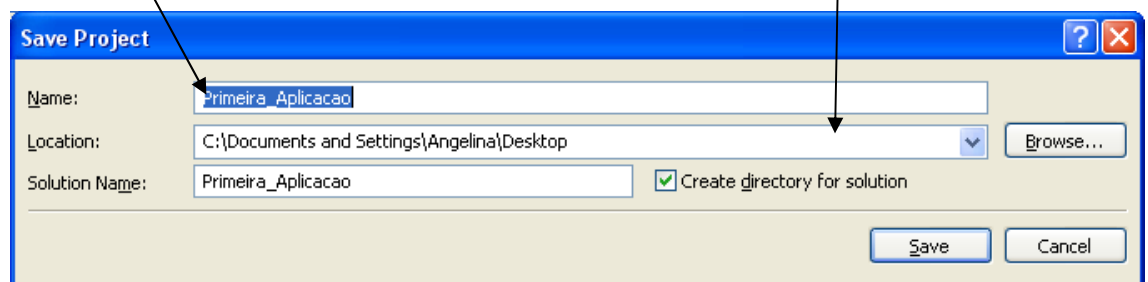
A Janela de Propriedades pode ser usada para alterar uma informação do controle selecionado (como o nome, cor) ou associar uma ação (evento) do usuário ao mesmo (como ao clicar o mouse).

Sempre deixe a exibição das propriedades em ordem alfabética para facilitar a sua procura.

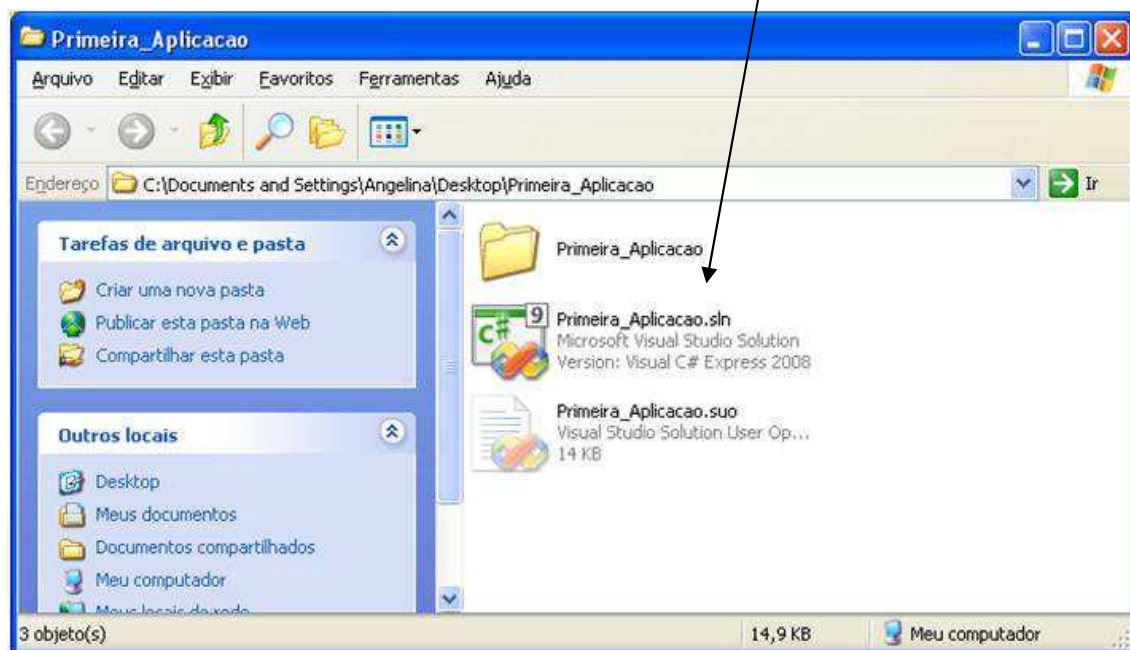
IV- Salvar toda a Solução

Para salvar corretamente o projeto e toda a aplicação desenvolvida clique na ferramenta Save All  da Barra de Ferramentas. Defina a localização e a pasta onde devem ser salvos os arquivos da solução (todos os projetos).

A opção de criar uma pasta para a solução fica ativa por padrão, ajudando a organizar os arquivos que são gerados pela solução.



Ao clicar na opção Save será gerada a pasta Primeira_Aplicação no Desktop. Ao abrir essa pasta haverá o arquivo principal, da solução, de nome Primeira_aplicação e extensão .sln. Sempre que for alterar ou visualizar a solução é esse arquivo que deve ser aberto.



Na subpasta Primeira_aplicação estão os demais arquivos, como os dos formulários criados, dos projetos, do programa principal.



Extensões de arquivos

- Sln – solução criada
- Cspj – arquivo do projeto CSharp
- Cs – arquivos do CSharp
- AssemblyInfo.cs – arquivo usado para adicionar atributos como nome do ator, data em que foi criado, entre outros.
- Exe – arquivo executável
- Pdb – armazena informações de depuração da aplicação

Program.cs – arquivo principal e inicial

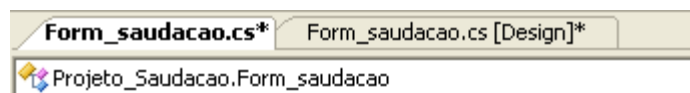
*.Designer.cs – arquivo com a definição da classe

Curiosidade

O asterisco (*) que aparece na Guia Superior significa que o arquivo atual foi modificado e não foi salvo.

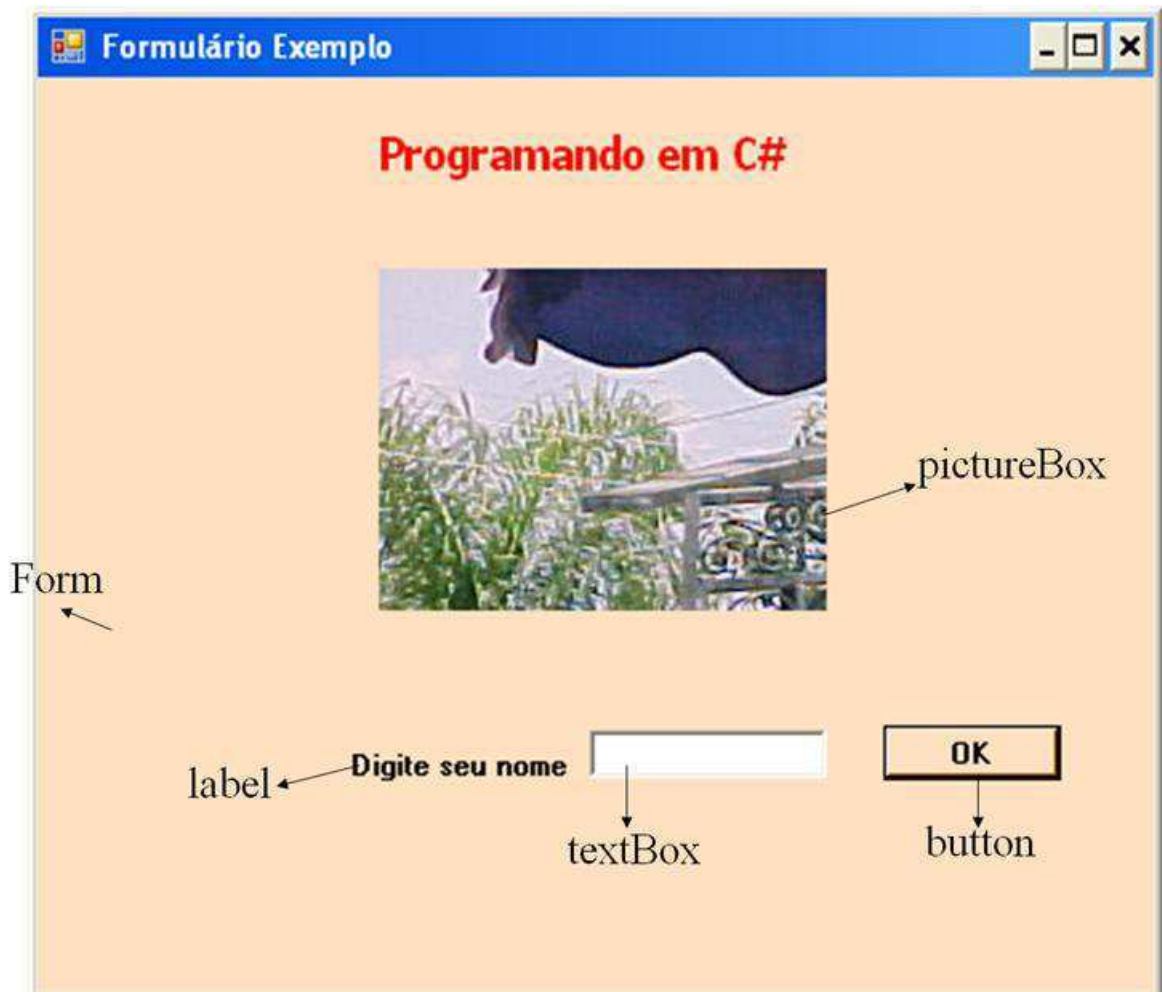
Toda vez que você executar o arquivo, pressionando o F5, os arquivos serão salvos.

Quando estiver aparecendo um cadeado significa que o programa está em execução e por isso você não pode fazer alterações (protegendo o código fonte atual).



V- Exemplos

Exemplo de Formulário:



Exemplo de código:

a) Código em uma aplicação Windows Forms

```
private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show("Bom dia! " + textBox1.Text);
}
```

b) Código em uma aplicação Console

```
static void Main()
{
    System.Console.WriteLine("Boa noite!\n");
}
```

VI- Namespace

Conceito de Namespace

No C#.Net são fornecidas várias bibliotecas de classes que agrupam seus recursos em namespace. Essas bibliotecas são conhecidas por Framework Class Library(FCL). Nesses namespace tem códigos que os programadores podem reutilizar, e que já vem na plataforma .Net (Framework .Net). O código real das classes estão localizados em um arquivo .dll.

Esses namespaces agrupam vários recursos do C# em categorias relacionadas, um exemplo é o namespace System.Windows.Forms que contém classes que ajudam o programador a definir GUIs (Interface Gráfica do Usuário- exemplo o botão de comando) para seus aplicativos.

Para fazer uso de um novo namespace é necessário incluir a linha using (diretiva).

Uma função da namespace é evitar conflito de nomes de classes, onde as classes da FCL ficam separadas das classes criadas pelo usuário, ou seja, em namespaces diferentes.

Ao iniciar um projeto do tipo Windows Forms já vêm algumas namespaces necessárias para o desenvolvimento desse tipo de projeto.

```
using System;  
//contém classes e tipos de dados fundamentais (como classe math, int)  
using System.Collections.Generic;  
using System.ComponentModel;  
using System.Data;  
//classes para acesso e manipulação de banco de dados  
using System.Drawing;  
//classes usadas para desenho e elementos gráficos  
using System.Text;  
using System.Windows.Forms;  
//classes usadas para criar interface gráfica para o usuário
```

Se for escolhida o tipo de projeto Console Application aparece namespace diferentes.

```
using System;  
using System.Collections.Generic;  
using System.Text;
```

Algumas namespaces deverão ser incluídas no projeto conforme a necessidade do mesmo. Por exemplo, se for necessário executar a Calculadora do Windows é necessário incluir a namespace System.Diagnostics, como no exemplo que se segue.

`using System.Diagnostics`

Após sua inclusão não é mais necessário referenciá-la diretamente, basta colocar o nome da classe que está nessa namespace.

```
Process.Start(@"c:\calc.exe");
```

Se isso não for feito, pode-se fazer a referência direta:

```
System.Diagnostics.Process.Start(@"C:\windows\system32\Calc.exe");
```

Dica:

O @ foi utilizado para simplificar a digitação do endereço da calculadora. Se não for colocado deve ser colocada duas barras \\ ao invés de uma \.

```
System.Diagnostics.Process.Start("C:\\windows\\system32\\Calc.exe");
```

Alguns exemplos de classes das namespace

A classe Int32 tem o método Parse, que converte uma string em um inteiro. Essa classe faz parte do namespace System

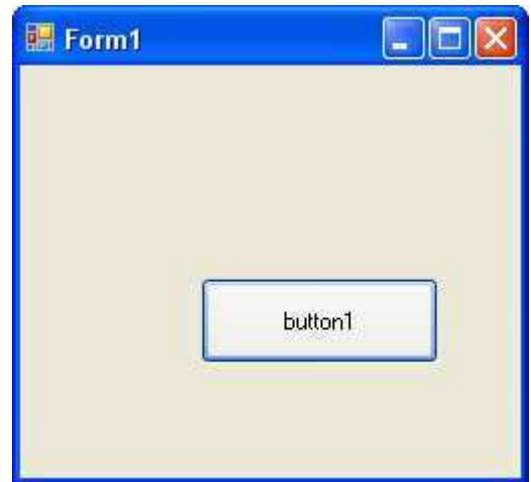
A classe Math tem o método Pow, que calcula exponenciação. Essa classe também está no System.

A classe MessageBox tem o método Show que está no System.Windows.Forms. Esse método recebe uma string como argumento e apresenta ao usuário uma mensagem através de uma caixa de diálogo.

Exemplo do Namespace

```
using System;  
using System.Collections.Generic;  
using System.ComponentModel;  
using System.Data;  
using System.Drawing;  
using System.Linq;  
using System.Text;  
using System.Windows.Forms;
```

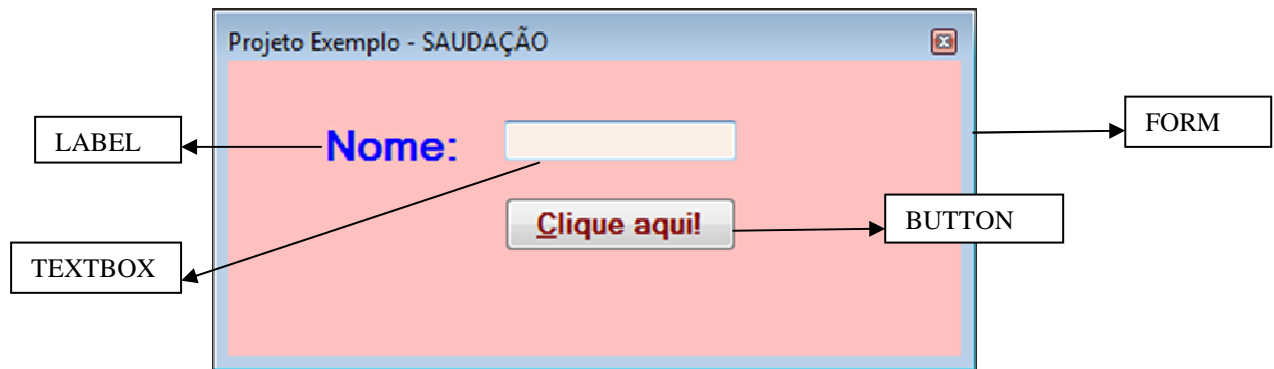
```
namespace WindowsFormsApplication1  
{  
    public partial class Form1 : Form  
    {  
        public Form1()  
        {  
            InitializeComponent();  
        }  
  
        private void button1_Click(object sender, EventArgs e)  
        {  
            MessageBox.Show("Exemplo");  
        }  
    }  
}
```



```
//using System.Windows.Forms;  
namespace WindowsFormsApplication1  
{  
    public partial class Form1 : System.Windows.Forms.Form  
    {  
        public Form1()  
        {  
            InitializeComponent();  
        }  
  
        private void button1_Click(object sender, EventArgs e)  
        {  
            System.Windows.Forms.MessageBox.Show("Exemplo");  
        }  
    }  
}
```


VII- Primeiro Projeto

Objetivo do Projeto: o usuário irá digitar o nome na caixa de texto e ao pressionar o botão “Clique aqui!” irá aparecer uma mensagem personalizada “Bom dia <nome do usuário> !”.



Propriedades

As propriedades são usadas para alterar as características (informações) do formulário, de seus componentes e controles, como por exemplo, mudar o texto, a cor, a posição, incluir uma figura.

Formulário		
O formulário é um elemento gráfico que aparece na tela.		
Ele é um contêiner para os componentes e controles		
Propriedade	Valor	Significado
FormBorderStyle	FixedToolWindow	Faz aparecer apenas a opção Fechar (X) na barra de título
Name	Form_saudacao	Altera o nome do formulário
Text	Projeto exemplo – saudação	Altera o texto que aparece na barra de título
BackColor	Red	Altera a cor do fundo do formulário
Opacity	70%	Modifica a porcentagem de transparência do formulário
StartPosition	CenterScreen	Faz com que o formulário ao ser exibido já fique no centro da tela

Controle Label		
O Rótulo, texto que aparece na tela e não pode ser alterado (editado) pelo usuário		
Propriedade	Valor	Significado
Name	textBox_nome	Altera o nome da caixa de texto
Text		Contém o conteúdo a ser digitado pelo usuário ou informado diretamente.
Font	Altera o tipo, estilo e tamanho da fonte
ForeColor	Blue	Altera a cor da fonte

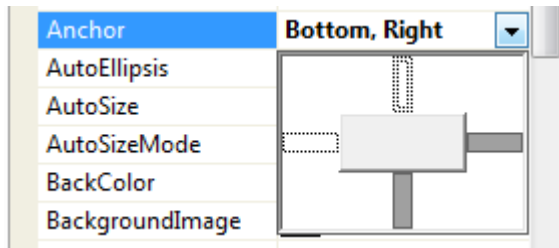
Controle TextBox		
Caixa de texto que permite o usuário entrar com dados (string)		
Propriedade	Valor	Significado
Name	label_nome	Altera o nome do label
Text	Nome:	Altera o texto que aparece na tela
Font	Altera o tipo, estilo e tamanho da fonte
ForeColor	Blue	Altera a cor da fonte
BackColor	Linen	Altera a cor do fundo

Controle Button		
Botão, é usado para chamar um evento quando clicado		
Propriedade	Valor	Significado
Name	button_exibe	Altera o nome
Text	&Clique aqui! Aparece <u>C</u> lique aqui! O operador & é usado para gerar uma tecla de atalho para o botão, ou seja, não precisa pressionar o botão com o mouse, basta pressionar o ALT+C	Altera o texto do botão
Font	Altera o tipo, estilo e tamanho da fonte
ForeColor	Maroon	Altera a cor da fonte
BackColor	Linen	Altera a cor do fundo

Dica:

1ª A propriedade Anchor determina o ajuste da posição do componente em relação ao formulário, ou seja, quando for redimensionado o formulário como será o comportamento do componente.

A opção Bottom e Righth é usada para ajustar à direita na parte de baixo.



2ª A barra de ferramenta Layout deve ser usada para agilizar a construção do formulário, onde pode-se selecionar vários componentes e alterar seus tamanhos numa única vez, modificar seus alinhamentos.



alinha os componentes selecionados à esquerda

centraliza os componentes selecionados à esquerda

alinha os componentes selecionados à direita

faz com que todos os componentes selecionados fiquem com a mesma largura

faz com que todos os componentes selecionados fiquem com a mesma altura

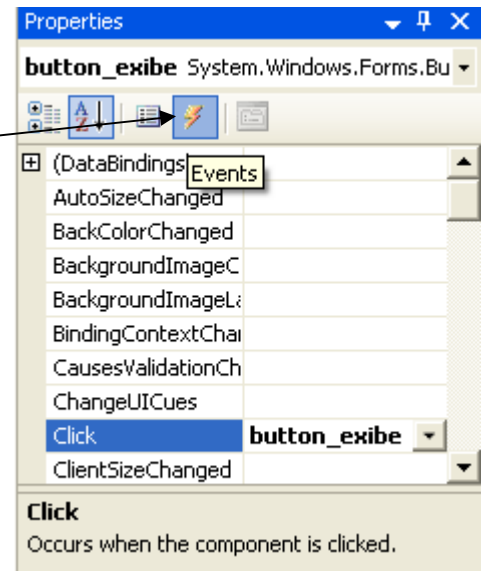
3ª Para selecionar mais de um mesmo componente, selecione o componente e segure a tecla CTRL. Vá clicando no formulário e adicionando os componentes. Após adicionar os componentes pressione a tecla ESC (desativando a seleção do componente)

Eventos

Os eventos estão associados às interações (ações) realizadas pelo usuário, como o clicar do mouse sobre um botão, mover do mouse sobre uma figura, entre outros.

Nesse primeiro projeto irá ocorrer um evento quando o usuário clicar no botão, onde irá aparecer uma mensagem.

Para definir um evento é necessário primeiro selecionar o componente/controle, depois ir janela de propriedades e clicar na opção Events.



Código que irá aparecer:

/*Namespaces usadas no desenvolvimento de uma aplicação Windows Forms */

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
```

//namespace raiz é o nome do projeto definido pelo usuário

```
namespace Projeto_Saudacao
```

```
{
```

/* a classe Form_saudacao que você desenvolve é parcial, ou seja, parte dela já foi escrita pelo C# e a outra parte você escreve*/

```
public partial class Form_saudacao : Form
{    //declaração dos métodos da classe
```

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

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

nome do botão tipo de evento
button_exibe_Click

Você deverá incluir seu código no evento selecionado (Click)

```
private void button_exibe_Click(object sender, EventArgs e)
{
```

```
/*
```

Para exibir uma mensagem usa-se a classe MESSAGEBOX e o método dela chamado SHOW

Existem 21 formas diferentes de se usar o método show (sobrecarga do método show)*/

```
MessageBox.Show (
    1 of 21 DialogResult MessageBox.Show (string text)
    text: The text to display in the message box.
)
```

Você pode digitar o MessageBox das seguintes formas:

1ª) `MessageBox.Show("Bom dia!");`

Texto que aparece no centro da Caixa de Diálogo



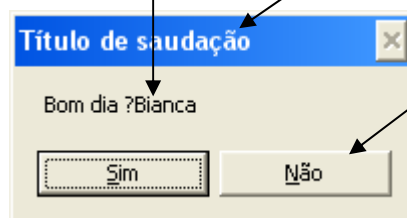
2ª) `MessageBox.Show("Bom dia! " + textBox_nome.Text);`

Faz um concatenação (une) a string "Bom dia!" com o conteúdo digitado na caixa de texto.



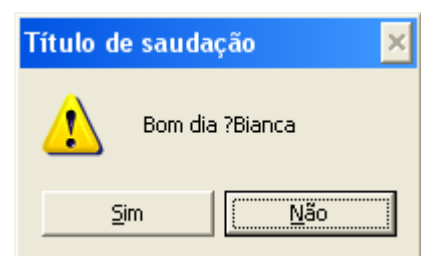
3ª) `MessageBox.Show("Bom dia? " + textBox_nome.Text, "Título de saudação", MessageBoxButtons.YesNo)`

Faz aparecer um texto central um título na caixa de diálogo, e botões de escolha.



Ficará o primeiro botão destacado (que tem o foco), se você não definir o botão padrão.

4ª)



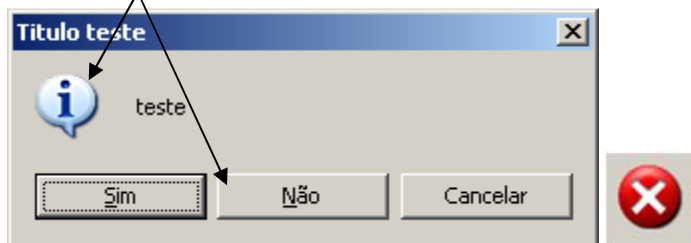
```

MessageBox.Show( "Bom dia ?"+textBox_nome.Text,
"Título de saudação",
MessageBoxButtons.YesNo,
MessageBoxIcon.Exclamation,
MessageBoxDefaultButton.Button2);

```

Tipos de botões e ícones do MessageBox

Botões	Ícones
AbortRetryIgnore OK OKCancel RetryCancel YesNo YesNoCancel	Asterisk - Mostra um balão com a letra i Error ou Stop - Mostra um balão vermelho com um X Exclamation ou Warning - Mostra um triângulo amarelo com um ponto de exclamação. Information - Mostra um círculo com a letra i None - Não é mostrado nenhum ícone Question - Mostra um balão com um ponto de interrogação



VIII- Variáveis

Em C# existem vários tipos de dados (variáveis) como: inteiros, ponto flutuante, lógico e caracter. Todas as regras quanto à forma de usar um tipo estão definidos na CTS (Common Type System), ou seja, para haver essa integração e compatibilidade entre as linguagens .Net, existem os tipos comuns compartilhados entre elas (internos).

Na plataforma .Net todas as suas linguagens de programação compartilham o mesmo sistema de CTS, ou seja, o tipo inteiro definido na linguagem C# é o mesmo tipo inteiro definido em VB.Net ou J#.Net. Abaixo segue uma tabela com os tipos de dados, sua referência na CTS, a quantidade de bytes que ocupa.

Tipo de dado	Tipo .Net	Descrição	Bytes
Boll	Boolean	V ou F	1
Byte	Byte	Sem sinal (0 até 255)	1
Sbyte	Sbyte	Com sinal (-128 até 127)	1
Short	Int16	Com sinal (-32.768 até 32.767)	2
Ushort	UInt16	Sem sinal (0 até 65.535)	2
Int	Int32	Com sinal (-2.147.483.648 até 2.147.483.647)	4
UInt	UInt32	Sem sinal (0 até 4.294.967.295)	4
Long	Int64	Com sinal (-9.223.372.036.854.775.808 até 9.223.372.036.854.775.807)	8
Ulong	UInt64	Sem sinal (0 até 18.446.744.073.709.551.615)	8
Float	Single	Ponto flutuante. (+/- $1,5 \times 10^{-45}$ até +/- $3,4 \times 10^{1038}$) 7 dígitos significativos. Exige o sufixo "f" ou "F".	4
Double	Double	Ponto flutuante de precisão dupla. (+/- $5,0 \times 10^{-324}$ até +/- $1,8 \times 10^{308}$) - 15 a 16 dígitos significativos	8
Decimal	Decimal	Precisão fixa de 28 dígitos e a posição do ponto decimal. Exige o sufixo "m" ou "M".	16
Char	Char	Um único caracter	2
String	String	Armazena cerca de 2 bilhões de caracteres	2 para cada caracter
Datetime	Datetime	Tipo data e hora (vai de 1/1/1 até 31/12/9999 e horas de 0:00 até 23:59:59)	8

Sintaxe:

Tipo da variável nome(s) da(s) variável (veis) [= valor inicial];

A inicialização da variável não é obrigatória

Exemplo de declaração e atribuição

```
byte idade;
```

```
byte idade = 33;
```

```
char sexo = "F";
```

```
string nome = "Julia";
```

```
boolean Sair = False;
```

```
Datetime Data_nasc;
```

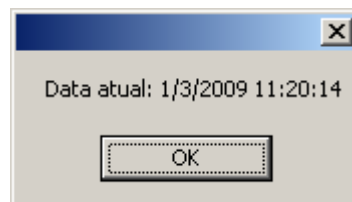
```
Datetime Data_hoje = DateTime.Now;
```

```
float nota = 7.5f;
```

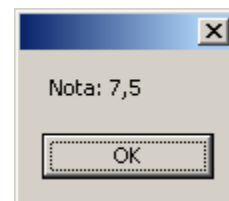
//internamente se define o separador decimal como ponto (.) e coloca o sufixo f de float.

Exemplos práticos:

```
1º) private void button1_Click(object sender, EventArgs e)
{
    DateTime data_atual = DateTime.Now;
    MessageBox.Show("Data atual: " + data_atual.ToString());
}
```



```
2º) float nota;
    nota = 7.5f;
    MessageBox.Show("Nota: " + nota.ToString());
```

**Alerta:**

Na digitação do usuário sempre é vírgula (,) o separador decimal e internamente no código para o programador é ponto(.).

IX- Conversão de Tipos

Em alguns momentos da aplicação será necessário fazer a conversão dos valores, por exemplo, você declarou a variável como tipo inteiro e depois precisa do seu valor como tipo string (para exibir na tela).

A conversão pode ser feita de diversas maneiras, tomando o cuidado de observar que alguns tipos não podem ser convertidos para qualquer outro tipo, por exemplo, não podemos converter uma string em inteiro. E também, tem que ser verificada a quantidade de bytes que o tipo ocupa, não pode ser convertido um tipo que ocupa mais bytes em um tipo que ocupa menos, ou seja, de inteiro para byte.

Veja a seguinte situação o usuário irá digitar um valor na caixa de texto (textbox). O valor que é digitado é do tipo string (mesmo que seja um número). Se for fazer um calculo com esse valor é necessário converte-lo para um tipo numérico equivalente. Abaixo segue o código de conversão e uma explicação.

//É declarada a variável x como do tipo float
float x;

//Como o usuário digitou o valor numa caixa de texto (textBox), deve-se pegar o conteúdo dessa caixa de texto e converte-lo para numérico. Não pode-se esquecer que sempre é o nome da caixa de texto (textBox1) e sua propriedade que contém o texto digitado (text)

Pode ser usada a classe Convert ou o método Parse para converter de string para float. Se for usada a classe Convert tem que saber o tipo .Net do float, no caso é Single (Ver Tabela na parte de Variáveis).

x = Convert.ToSingle(textBox1.Text) * 2.5f;

ou

x = float.Parse(textBox1.Text) * 2.5f;

//Para exibir um conteúdo numérico em uma mensagem ou mesmo num label, deve-se fazer novamente a conversão.

MessageBox.Show("Convertido: " + x.ToString("N2"));

ou

label1.Text = Convert.ToString(x);

Formas de conversão:

- Classe Convert
- Método Parse()
- Método ToString()

Formatação de string através do Método ToString()	
Formatos	Exemplos
P – Percentual	Nota.ToString("##0.00");
N – Numérico	Mostra com 3 partes inteiras e 2 decimais
F – Fixo	
C- Currency	Valor_Venda.ToString("C2");
E- Notação Científica	Mostra na moeda corrente (R\$) com duas casas decimais
	Perc_Lucro.ToString("P");
	Mostra em porcentagem

Exemplos de conversão:

1º) Conversão de uma string em tipo float

```
float peso, altura, valIMC;
peso = float.Parse(txtPeso.Text);
altura = float.Parse(txtAltura.Text);
```

2º) int n1;

```
short litros;
```

```
float valorfrete;
```

```
n1 = Int32.Parse("400"); //conversão de uma string em inteiro
```

```
litros = Convert.ToInt16(litrotextBox.Text); //conversão de uma string em short
```

```
totallabel.Text = valorfrete.ToString("C2"); //conversão de um tipo float em string
```

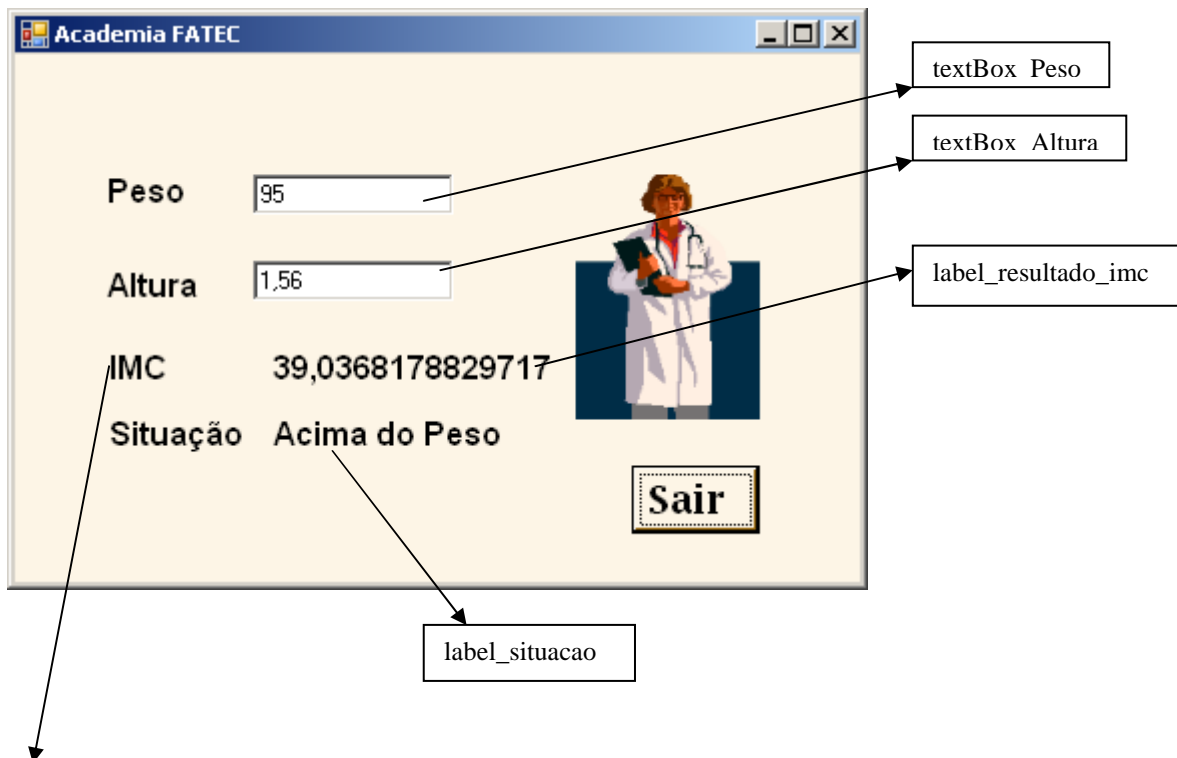
3º) int x;

```
x = Int32.Parse("100"); //conversão de uma string em inteiro
```

```
MessageBox.Show(x.ToString()); //conversão de um inteiro em string
```

X- Segundo Projeto

Objetivo do Projeto: Neste segundo projeto serão feitas duas entradas o peso e a altura. Quando o usuário clicar sobre a palavra IMC será calculado o Índice de Massa Corporal do usuário, verificada e mostrada a sua situação de peso (acima, abaixo ou com peso ideal)



Código do evento Click do Label de nome label_imc

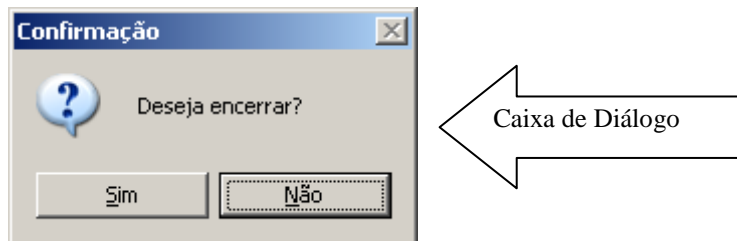
```
private void label_imc_Click(object sender, EventArgs e)
{
    double altura, peso, imc;
    altura = Convert.ToDouble(textBox_Altura.Text);
    peso = double.Parse(textBox_Peso.Text);
    // imc = peso / altura * altura; ou
    imc = peso / Math.Pow(altura, 2);
    label_resultado_imc.Text = imc.ToString();
    if (imc < 19)
        label_situacao.Text = "Abaixo do Peso";
    else if (imc < 25)
        label_situacao.Text = "Peso Ideal";
    else
        label_situacao.Text = "Acima do Peso";
}
```

Alerta:

- Foi usado o método Pow() da classe Math para calcular a altura². A Math está no namespace System.
- Foi usado o Parse e o Convert para mostrar que ambos podem ser usados para conversão de string para Double. Poderia ser usado apenas o Convert.

Código do evento Click do Button de nome button_Sair

```
private void button_Sair_Click(object sender, EventArgs e)
{
    if (MessageBox.Show("Deseja encerrar? ", "Confirmação",
        MessageBoxButtons.YesNo, MessageBoxIcon.Question,
        MessageBoxDefaultButton.Button2) == DialogResult.Yes)
    this.Close();
}
```



Quando foi usada a classe MessageBox e o método Show foram passados alguns argumentos, como:

"Deseja encerrar? " → mensagem que aparece no centro da caixa de diálogo

Confirmação" → mensagem que aparece na barra de título

MessageBoxButtons.YesNo → tipos de botões que vão aparecer

MessageBoxIcon.Question → tipo de ícone que vai aparecer

MessageBoxDefaultButton.Button2 → qual dos botões que tem inicialmente o foco (destaque)

Para fazer a verificação de qual botão o usuário pressionou da caixa de diálogo deve ser usado o DialogResult e a opção a ser testada.

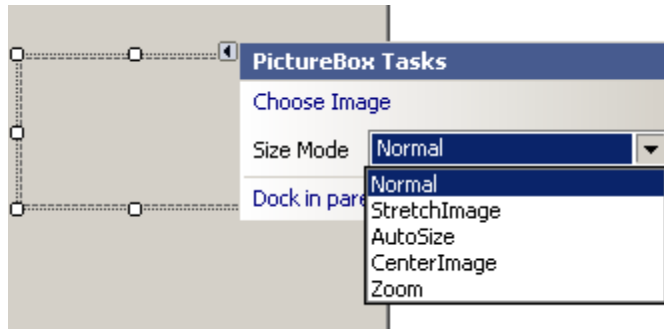
== DialogResult.Yes

Dica:

Quando for usar o controle PictureBox, não precisa ir na propriedade para escolher a figura que deve aparecer e a forma como ela vai aparecer.

Quando for inserido PictureBox e colocado o foco sobre esse controle deve ser clicado na seta superior da caixa e clicado sobre a opção Choose Image para escolher a imagem.

Na mesma seta existem as opções de visualização da imagem, através da opção Size Mode.



XI- Tratamento de String

Existem alguns métodos específicos do tipo string, como remover os espaços, transformar em maiúsculo.

Método	Funcionalidade
Trim()	Remove os espaços
Length	Retorna o tamanho da string
ToUpper()	Converte a string em maiúsculo
ToLower()	Converte a string em minúsculo
Substring()	Retorna uma parte da string a partir da posição definida

```
private void button_tamanho_Click(object sender, EventArgs e)
{
    int tamanho;
    tamanho = textBox_nome.Text.Length;
    MessageBox.Show(tamanho.ToString());
}

private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show(textBox_nome.Text.Substring(0,10));
    //estará mostrando a partir do primeiro caracter digitado até a 10 posição
    Por exemplo, se for digitado na caixa de texto:
        Programação C#
    Irá aparecer
        Programaçã
}
```

Curiosidade:

A primeira posição é considerada zero(0) – índice interno.

XII- Terceiro Projeto

Objetivo do Projeto: Neste terceiro projeto serão feitas entradas de dados para cálculo das despesas de viagem, no final serão mostrados os valores calculados. Quando não for digitada uma informação necessária, será mostrada mensagem de alerta e retornado para o campo não preenchido.

Agencia de Viagens - FATEC ITU

Reserva de Passagem

Nome:

Destino:

Data de embarque:

Meio de Transporte

☐ Avião ☒ Ônibus

Valores do Pacote

Gasto Transporte: R\$ 30,00
 Gasto com Destino: R\$ 1.500,00
 Valor Total:

Buttons: Sair, Confirmar, Limpar Dados, Calculadora, Calcular Valor da Viagem

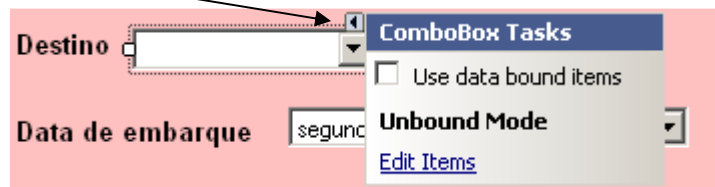
Algumas Propriedades alteradas		
Formulário	Icon	É usada para alterar o ícone que aparece na barra de título do formulário
Botões	Image	É usada para adicionar um imagem ao botão
	ImageAlign	Para definir o alinhamento da imagem em relação ao botão
	TextImageRelation	Modifica o tamanho da imagem em relação ao texto
Groupbox	Text	Meio de transporte
Groupbox	Text	Valores do Pacote
ComboBox	Itens	SP

		RJ SC
TextBox	Enabled	False

- Nesse projeto foi colocado um GrouBox usado para agrupar os dois RadioButton



- Para a inclusão dos itens do ComboBox pode se clicar sobre a seta e ir na opção Edit Items



Código do evento Click do Botão Limpar Dados

O botão limpar terá a ação de limpar o conteúdo selecionado ou digitado

```
private void button_Limpar_Click(object sender, EventArgs e)
{
    textBox_nome.Text = "";
    radioButton_aviao.Checked= false;
    radioButton_onibus.Checked = false;
}
```

Para limpar uma caixa de texto pode-se usar a propriedade text usar o método Clear().

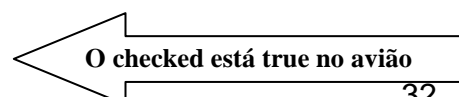
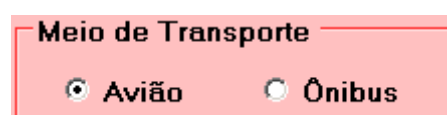
```
textBox_nome.Clear();
```

Ou

```
textBox_nome.Text = string.Empty;
```

Para tirar a opção de checado(escolhido) do radio button é usado a propriedade checked. Quando ela está false não aparece a bolinha, quando está true aparece.

```
radioButton_aviao.Checked= false;
radioButton_onibus.Checked = false;
```



Código do evento Click do Botão Calculadora

A ação desse botão será executar o processo Calculadora do Windows. Para isso é necessário saber a localização do arquivo .exe

```
private void button_Calculadora_Click(object sender, EventArgs e)
{
    Process.Start(@"C:\windows\system32\calc.exe");
}
```

Foi incluído o namespace da classe Process
`using System.Diagnostics;`

Código do evento Click do Botão Confirmar

Em todas as verificações se o valor testado for inválido deve ser exibida uma mensagem de alerta e depois desviado o foco para o campo.

```
private void button_confirmar_Click(object sender, EventArgs e)
{
    if (textBox_nome.Text.Length == 0)
    {
        MessageBox.Show("Digite o nome");
        textBox_nome.Focus();
    }

    if (comboBox_destino.Text == "")
    {
        MessageBox.Show("Destino não escolhido");
        comboBox_destino.Focus();
    }

    if (radioButton_aviao.Checked == false && radioButton_onibus.Checked == false)
    {
        MessageBox.Show("Meio de transporte não escolhido");
        radioButton_onibus.Focus();
    }
}
```

Código do evento Load do formulário

É possível definir ações que deverão ser executadas ao iniciar o formulário, através do evento Load.

No projeto atual foi redimensionado o formulário para não aparecer o GroupBox com os resultados, já que no começo não existem resultados.


```
private void Form_agencia_Load(object sender, EventArgs e)
{
    this.ClientSize = new System.Drawing.Size(533, 309);
}
```


Código do evento Click do Botão Calcular Valor da Viagem


Ao pressionar o botão calcular serão verificadas as opções selecionadas e de acordo com elas feito o calculo.


- Primeiro: é verificado qual destino o usuário escolheu através do ComboBox.
- Segundo: é verificado qual meio de transporte o usuário escolheu através dos RadioButton.
- Terceiro: é feita a soma total de gastos
- Quarto: o formulário é redimensionado para aparecer aparecer(tornar visível) o GroupBox que tem os valores calculados.
- Quinto: os valores são jogados(atribuídos) aos devidos componentes de exibição.


```
float gasto_destino, gasto_transporte, gasto_total;
```

```
switch (comboBox_destino.Text.ToUpper())
{
    case "SP": gasto_destino = 1000f;
    break;
    case "RJ": gasto_destino = 1500f; 
    break;
    case "SC": gasto_destino = 2000f;
    break;
    default: gasto_destino = 0.0f;
    break;
}
```

```
if (radioButton_aviao.Checked == true)
    gasto_transporte = 100f;
else 
    gasto_transporte = 30f;
```

```
gasto_total = gasto_transporte + gasto_destino; 
```

```
this.ClientSize = new System.Drawing.Size(533, 441); 
groupBox_Resultado.Visible = true;
```

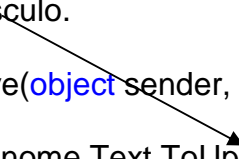
```
label_GastoDestino.Text = gasto_destino.ToString("C2");
label_GastoTransporte.Text = gasto_transporte.ToString("C2"); 
textBox_ValorTotal.Text = gasto_total.ToString("C2");
```

Código do evento Leave do TextBoxNome

O evento Leave é usado quando se deseja executar uma ação no componente após ele sair de foco.

No projeto após a pessoa digitar o nome, esse mesmo não importa como foi digitado, será transformado em maiúsculo.

```
private void textBox_nome_Leave(object sender, EventArgs e)
{
    textBox_nome.Text= textBox_nome.Text.ToUpper();
}
```



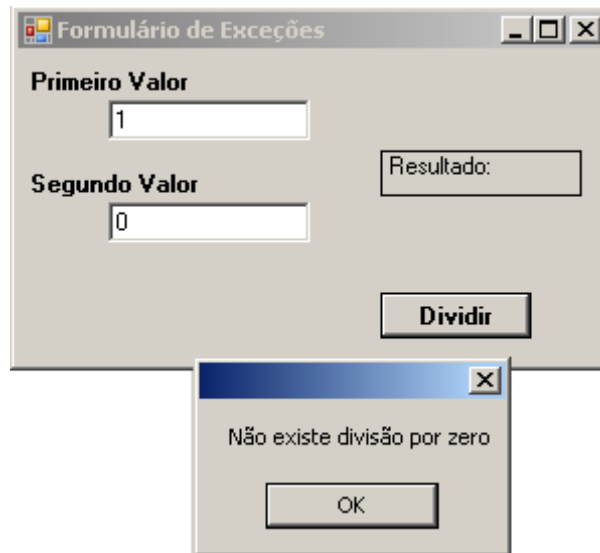
Dica: Fazer o C# trazer a estrutura pronta para uso.

1ª) Ao pressionar o botão da direita do mouse, você deve escolher a opção Insert Snippet..., depois dá um duplo clique na opção Visual C#. Após isso é só escolher a estrutura e ele a trará pronta na tela. Por exemplo, na aplicação atual você pode utilizar para trazer a do switch().

2ª) Todas as configurações iniciais do formulário e seus componentes podem ser vistas através do arquivo **Form_agencia.Designer.cs**

XIII- Quarto Projeto

Objetivo do Projeto: Nesse projeto será mostrado o tratamento de exceções. De forma que quando for digitado um valor inválido não será interrompida a execução da aplicação.



Um erro ou uma exceção pode ser: não digitação de um valor (causando um erro de conversão), digitação de um valor muito longo, erro quanto ao formato, divisão por zero, definição de um caminho (diretório) ou arquivo inválido, falha do hardware, erro do sistema operacional, erros de dispositivos.

O bloco de comandos que pode causar uma exceção deve estar dentro **do try..catch**, de forma que se possa capturar a exceção e manipulá-la. Somente o código dentro do try..catch está protegido.

Somente um catch será executado caso ocorra uma das possíveis exceções do código definido dentro do try. A estrutura do try..catch pode ser:

```
try
{
    // Código que pode estar sujeito a exceção (erro)
}
catch( TipoExcecao1 )
{
    // Tratamento para exceção tipo 1
}
catch( TipoExcecao2 )
{
    // Tratamento para exceção tipo 2
}
catch
{
    // Tratamento para qualquer tipo de exceção
}
finally
{
    // Trecho que deve sempre ser executado, havendo ou não exceção
}
```

As exceções podem ser tratadas individualmente ou de forma genérica.

Tratadas de forma genérica: Dessa forma não é tratado de forma individual o erro, a exceção.

```
private void dividirbutton_Click(object sender, EventArgs e)
{
    try
    {
        decimal valor1, valor2, resultado;
        valor1 = Convert.ToDecimal(valor1textBox.Text);
        valor2 = Convert.ToDecimal(valor2textBox.Text);
        resultado = valor1 / valor2;
        resultadolabel.Text = Convert.ToString(resultado);
    }
    catch (Exception)
    {
        MessageBox.Show("Não pode ser feita a divisão - Ocorreu uma exceção");
    }
}
```

Pode ser gerada uma instância da classe Exception

Pode ser feita a instância da classe Exception. No código abaixo o nome dado é **ex**, mas poderia ser qualquer nome. Essa classe tem a propriedade **Message** que mostra a descrição da exceção.

```
private void dividirbutton_Click(object sender, EventArgs e)
{
    try
    {
        decimal valor1, valor2, resultado;
        valor1 = Convert.ToDecimal(valor1textBox.Text);
        valor2 = Convert.ToDecimal(valor2textBox.Text);
        resultado = valor1 / valor2;
        resultadolabel.Text = Convert.ToString(resultado);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Erro Genérico");
    }
}
```



Tratadas de forma específica

Pode haver a necessidade de tratar cada exceção de forma individual. Para cada exceção tem um catch e uma classe. Todo objeto que representa uma exceção é derivada da classe Exception que está na namespace System.

```
private void dividirbutton_Click(object sender, EventArgs e)
{
    try
    {
        decimal valor1, valor2, resultado;
        valor1 = Convert.ToDecimal(valor1textBox.Text);
        valor2 = Convert.ToDecimal(valor2textBox.Text);
        resultado = valor1 / valor2;
        resultadolabel.Text = Convert.ToString(resultado);
    }

    catch (DivideByZeroException)
    {
        MessageBox.Show("Não existe divisão por zero");
    }
    //trata da exceção de divisão por zero

    catch (OverflowException)
    {
        MessageBox.Show("Erro de estouro");
    }
    //trata da digitação de um número muito grande, não permitido
    para o tipo definido ou para o cálculo a ser feito.

    catch (FormatException)
    {
        MessageBox.Show("Formato Inválido de dados");
    }
    //trata da exceção quanto a formato inválido, como exemplo
    podemos destacar: quando há a exigência de um número e é digitado um
    texto ou um espaço em branco. Principalmente para conversão de valores
}
```

Bloco finally

Tudo definido dentro do finally sempre será executado, existindo ou não a ocorrência de exceções (executando ou não o bloco catch).

```
try
{
    ...
}
catch ( )
{
    ...
}
finally
{
    MessageBox.Show("Sempre é executado dando erro ou não");
    this.close();
}
```

Exceção gerada em tempo de execução.

Uma exceção pode criar suas próprias exceções. De forma que podem ocorrer novos erros durante o tempo de execução e para isso deve ser usado o throw (que gera exceção). O throw deve estar dentro do bloco do try.

```
if (valor2 % 2 == 0)
{
    resultado = valor1 / valor2;
    resultadolabel.Text = Convert.ToString(resultado);
}
else
{
    throw new Exception("O segundo valor deve ser par");
}
```

Outro exemplo:

```
if ( CPF.Length != 11 )
{
    throw new Exception( "CPF Inválido" );
}
```

Cuidado:

A definição e tratamento das exceções deve ser feita da mais específica para a mais genérica, nessa ordem, senão ocorrerá erro de código.

Dicas:**Outras Exception:**

InvalidCastException (erro de conversão direta- cast)

IndexOutOfRangeException: índice do array fora do limite

FileNotFoundException: arquivo não encontrado

SqlException: erro na operação do Sql Server.

XIV- Quinto Projeto

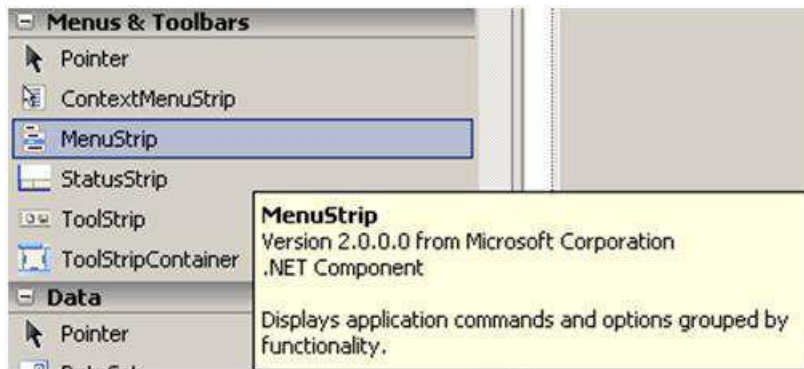
Objetivo do Projeto: Mostrar um projeto contendo um menu e configurações de alertas de erros. O menu será configurado de maneira que exista uma janela principal e que as demais sejam exibidas dentro dela.



Program.cs – É o primeiro arquivo que é executado ao iniciar a aplicação. Nele é definido qual o primeiro formulário que deverá ser exibido. Se for necessário mudar a ordem de execução, deve ser alterado o nome do formulário inicial desse arquivo.

```
namespace Projeto_menu_alerta
{
    static class Program
    {
        /// The main entry point for the application.
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new FormMenu());
        }
    }
}
```


MenuStrip - Esse controle quando arrastado no formulário ficará na parte de baixo do mesmo. Com ele é possível fazer a criação de um menu.



Para isso basta dar um único clique nele. Veja que apareceu no formulário um espaço para você digitar as opções do menu. Conforme você for digitando irá aparecer outros espaços, como ao lado, ou embaixo daquele que você está.



Para cada opção do menu você poderá criar uma tecla de atalho. Ao usar o & na frente da letra, ele irá sublinhá-la e quando o usuário pressionar a tecla ALT irá aparecer os atalhos.



Configuração dos formulários para a existência de uma única janela (Formulário Principal)

Nesse projeto existirão vários formulários (várias janelas ao mesmo tempo, mas apenas um aplicativo em execução), para isso pode-se usar o recurso de definir um MDI, onde este é o formulário principal (pai). Para isso é necessário alterar a propriedade **IsMdiContainer** do formulário menu para o valor **True**.



Quando for escolhida o item Cliente da opção Cadastro do menu irá aparecer o outro formulário dentro do formulário atual (do menu). Para isso é colocado o código a seguir:

```
private void clientesToolStripMenuItem_Click(object sender, EventArgs e )
{
    Form_Alerta obj_form_alerta;
    //foi definido o nome do objeto e de qual classe ele se
referência
    obj_form_alerta = new Form_Alerta();
    //foi criada uma nova instância da classe Form_Alerta
    obj_form_alerta.MdiParent = this;
    //é definido o pai da janela filha, ou seja, faz com que o
novo formulário (Alerta) seja exibido dentro do formulário principal
(Menu). Não esqueça que o this é a janela atual.
    obj_form_alerta.Show();
}
```

O método Show é usado para exibir (mostrar) o formulário

Poderia ser criado o objeto de outra maneira:

```
Form_Alerta obj_form_alerta = new Form_Alerta();
```

Poderia ser feita a definição do formulário pai e fechado o mesmo.

```
obj_form_alerta.MdiParent = formAtual.ActiveForm;
obj_form_alerta.Show();
this.Close();
```



ErrorProvider- Esse componente quando arrastado no formulário ficará na parte de baixo do mesmo. Com ele é possível alertar o usuário quanto a um eventual erro de digitação ou mesmo da falta de digitação. No caso abaixo foi dado o nome eP para o componente ErrorProvider.



Para que o alerta seja dado ao usuário é necessário definir as regras. Entre as regras poderia ser esquecer-se de digitar um valor, ou mesmo digitar um valor inválido.

Foram definidas as regras no evento Validating (validação) de cada caixa de texto (textBox).

Primeiro está sendo usado o ErrorProvider para mostrar o alerta quando o usuário esquecer de digitar os dados (a caixa de texto fica vazia).

```
private void textBoxnome_Validating(object sender, CancelEventArgs e)
{
    if (textBoxnome.Text == "")
    {
        eP.SetError(textBoxnome, "Você não digitou o nome");
        textBoxnome.Focus();
        return;
    }
    eP.Clear();
}
```

Veja que quando aparecer o alerta também está sendo dado o foco na caixa de texto que está vazia e finalizado o código através do return;

O return faz com que não haja a execução da sequência de códigos que estão após ele.

O alerta somente irá sumir quando o usuário digitar algum valor. Para isso é feita a execução do método Clear().

Na definição do ErrorProvider(eP) tem que ser definido na frente de qual componente deverá aparecer o aviso de erro e qual a mensagem que deve ser exibida quando o usuário passar sobre ele.

```
eP.SetError(textBoxnome, "Você não digitou o nome");
```



```
private void textBoxidade_Validating(object sender, CancelEventArgs e)
{
    if (textBoxidade.Text == string.Empty)
    {
        eP.SetError(textBoxidade, "Você não digitou a idade");
        textBoxidade.Focus();
        return;
    }
    eP.Clear();
}
```

Poderia ser colocado um outro tipo de alerta no botão Verificar. Quando o usuário digitar uma idade maior que 150. Foi feito o tratamento de exceção caso o usuário digite um texto ou vazio na idade e dessa forma não pare a execução do programa.

```
private void buttonVerificar_Click(object sender, EventArgs e)
{
    try
    {
        if (Convert.ToInt32(textBoxidade.Text) > 150)
        {
            eP.SetError(textBoxidade, "Você digitou uma idade inválida");
            textBoxidade.Focus();
            return;
        }
    }
    catch (Exception)
    {
        MessageBox.Show("Erro na digitação da idade", "Mensagem de Alerta");
    }
}
```

Outra forma de dar o alerta poderia ser através do código abaixo, onde somente ele é feito quando pressionar o botão Verificar.

```
private void buttonVerificar_Click(object sender, EventArgs e)
{
    if (textBox1.Text == "")
    {
        eP.SetError(textBoxnome, "NNNOMMMEE VAZZZIIO");
        textBox1.Focus();
    }
    if(textBox2.Text == "")
    {
        eP.SetError(textBoxidade, "Idade não informada!");
        textBox2.Focus();
        return;
    }
}
```

Opção Acessórios - Calculadora do Menu

Ao escolher essa opção será exibida a Calculadora do Windows. O namespace foi colocado na frente da classe Process.

```
private void calculadoraToolStripMenuItem_Click(object sender, EventArgs e)
{
    try
    {
        System.Diagnostics.Process.Start("calc.exe");
    }
    catch (Exception)
    {
        MessageBox.Show("Caminho inválido");
    }
}
```

XV- Estrutura foreach()

É uma estrutura usada para verificar um array (matriz) ou coleções. Não é preciso ter uma condição que define o termino da repetição e nem mesmo uma inicialização de variável.

O término da repetição ocorre quando todos os elementos do array/coleção tiverem sido varridos.

Exemplo:

Botão: Limpar

Função: deixar vazias todas as caixas de textos que estão preenchidas.

Evento: ao clicar no botão

```
foreach (Control caixatexto in this.Controls)
{
    if (caixatexto is TextBox)
    {
        caixatexto.Text = "";
    }
}
```

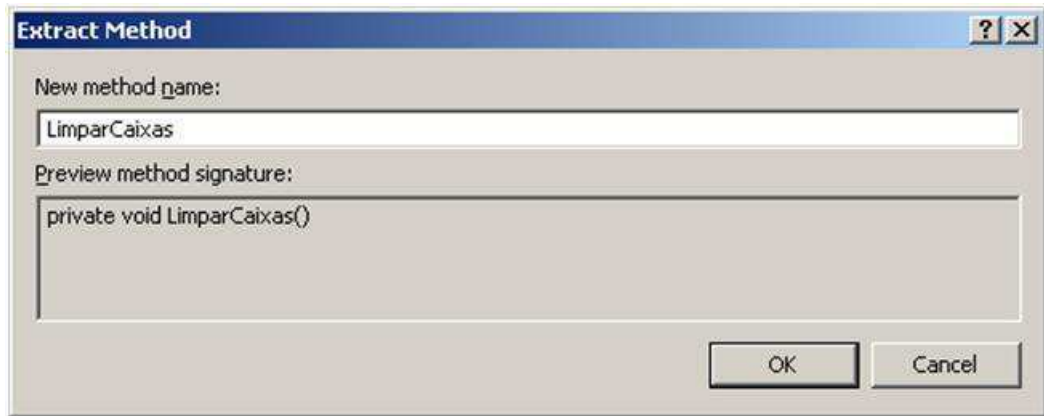
Dica: Criar um método através do código existente.

Quando existir a necessidade de executar as mesmas linhas de código diversas vezes, deve ser criado um método dessas linhas. Para isso execute os seguintes procedimentos:

1. Selecione as linhas que tem o código (por exemplo, limpar as caixas de texto)
2. Clique com o botão direito, selecione a opção Refactor, e Extract Method.



3. Depois irá aparecer uma janela solicitando um nome para o novo método (por exemplo: LimparCaixas())



4. Será criado um novo método com as linhas de código selecionadas. No local que estavam as linhas aparece a chamada do método.

//chamada do método criado

```
private void button_Limpar_Click(object sender, EventArgs e)
{
    LimparCaixas();
}
```

//Novo método criado

```
private void LimparCaixas()
{
    textBox_nome.Text = ""; //ou textBox_nome.Clear(); ou
    textBox_nome.Text = string.Empty;
    radioButton_aviao.Checked = false;
    radioButton_onibus.Checked = false;
    groupBox_Resultado.Visible = false;
    this.ClientSize = new System.Drawing.Size(533, 309);
}
```

XVI- Classes

Relembrando o que falamos em sala de aula: classes são tipos abstratos usados para representar objetos do mundo real, elas definem como um objeto deve se comportar.

Lembre da analogia que fizemos da Classe com a Receita de Bolo. A receita é a classe, é um tipo de referência. Nela temos todas as informações necessárias para fazer um bolo (modo de fazer), e quais ingredientes serão necessários. O modo de fazer seriam os métodos e os ingredientes seriam os atributos.

O fazer o bolo é chamado de instância, ou seja, é feita uma nova instância da classe bolo, ou mesmo criando um objeto.

Instâncias de classes são chamadas de objetos.

Métodos são o que o objeto faz e os campos são o que ele sabe.

No C# tudo é definido como uma classe. Para constatar vamos criar um novo projeto. Dê o nome para ele de Projeto_Classes_1 e altere o nome do Form1 para Form_Classe.

Vá no arquivo Program.cs e dê um duplo clique. Veja o código.

Dentro do namespace atual que é o nome que você deu ao projeto aparece a **classe Program**. Lembra que falamos de namespace e dá estrutura (organização) de classes que tem nele.

```
// a classe Program está dentro do namespace Projeto_Classes_1
namespace Projeto_Classes_1
{
    static class Program
    {
        [STAThread]

        //nessa classe existe o método Main() que é do tipo void, ou
        //seja, após acabar a execução desse método não haverá nenhum retorno.
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            //a definição do primeiro formulário que deverá ser exibido
            Application.Run(new Form_classe());
        }
    }
}
```

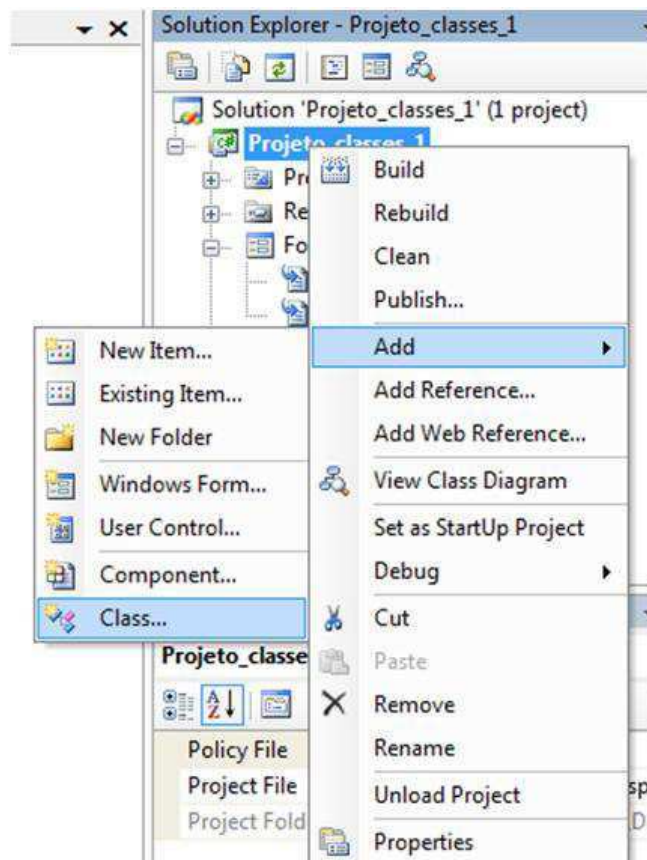

O **método Main()** é o principal da aplicação e é declarado automaticamente ao começar um novo projeto. Esse método é declarado como estático usando o modificador de acesso *static*, de forma que o método pode ser chamado sem que a classe seja instanciada.

Se você for até o Form_Classe e pressionar a tecla de atalho F7 verá o código desse formulário que também é uma classe, mesmo sendo **partial class** (a parte de herança, agregação será visto depois).

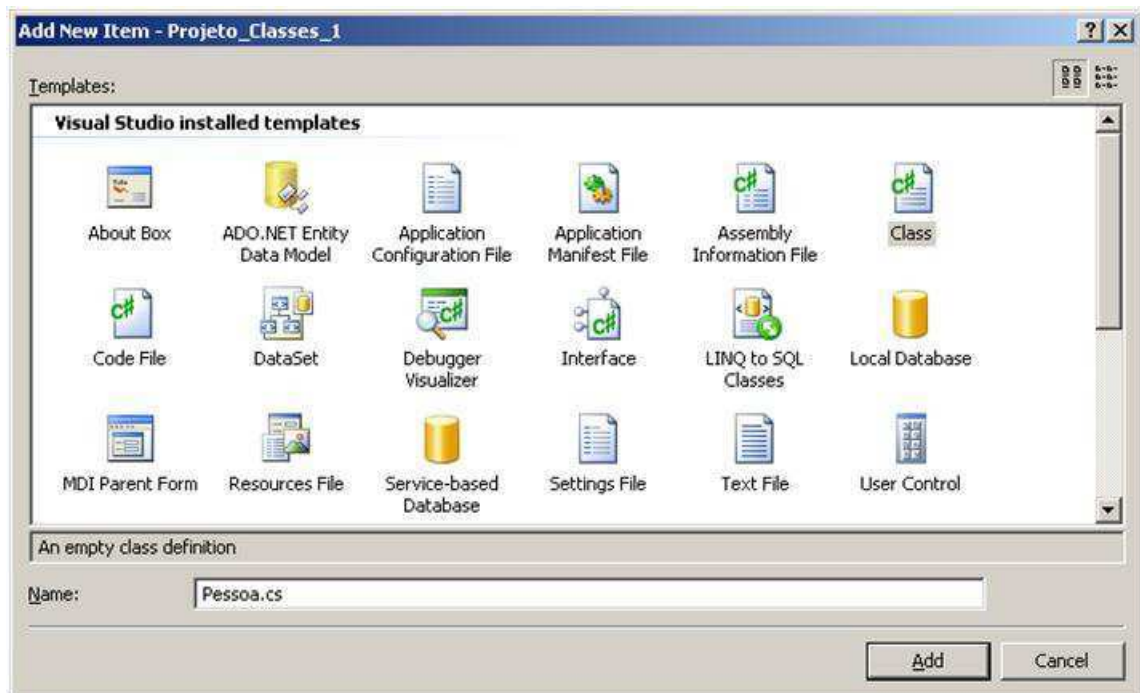
```
public partial class Form_classe : Form
{
    public Form_classe()
    {
        InitializeComponent();
    }
    //os métodos criados a partir dos eventos serão definidos dentro
    //desse escopo
}
```

Criação de uma classe

Vamos fazer a definição da classe Pessoa. Para isso vá no Solution Explorer, pressione o botão direito do mouse. Escolha a opção Add, item Class...



Define o nome da classe.



Apareceu um novo arquivo o Pessoa.cs. Vá no código e veja a estrutura da classe já montada.

```
using System;
using System.Collections.Generic;
using System.Text;

namespace Projeto_classes_1
{
    public class Pessoa
    {
    }
}
```

Quando você precisar definir a estrutura manualmente, basta colocar a palavra reservada **class** e na sequência o identificar dela (nome). Todo o conteúdo da classe deve estar delimitado pelas chaves.

Exemplo:

```
[modificador de acesso] class <identificador>
{
    // declaração dos atributos, propriedades e métodos
}
```

Você deve definir os atributos, as propriedades e métodos dentro do escopo da classe.

Para representar a classe você pode utilizar o **Diagrama de Classe**.

Classe
Atributo1
Atributo2
Método1()
Método2()
Método3()

Atributos de uma classe

Os atributos de uma classe também podem ser chamados de campos. Esses atributos definem a classe, definem suas características. A declaração pode ser feita da seguinte maneira:

[Modificador de acesso] [tipo atributo] <identificador>;

```
private string _nome;
private int _idade;
```

Quando o modificador de acesso do atributo não é especificado, o compilador assume como default o private.

Pode ser feita a declaração de vários atributos do mesmo tipo de uma vez só.

```
public int n1, n2;
```

Modificadores de Acesso

A visibilidade de uma classe e de seus membros (atributos, propriedades, métodos) também podem ser chamadas de modificadores de acesso (qualificadores ou identificadores).

Esses modificadores podem ser do tipo: **public, private, protected ou internal**.

Tipo	Função
public	O acesso pode ser feito tanto interno ou externo a classe.
private	O acesso é limitado a classe. O acesso fora dela não pode ser feito.
protected	Somente os membros da classe ou as suas classes derivadas é que tem acesso.
internal	O acesso é limitado ao assembly da classe

Esse modificadores define a forma como um membro(atributos, métodos) da classe poderá ser acessado externamente, de maneira que pode ser

determinado que alguns membros não possam ser visíveis ou acessados. Dessa maneira eles ficam escondidos (protegidos).

Pense no exemplo do carro, onde alguns elementos dele ficam acessíveis (volante, câmbio, maçaneta), enquanto que outros ficam protegidos (motor).

Outro exemplo é o componente Button, você não vê seu código, não sabe como foi implementado, mas utiliza-o.

Encapsulamento.

No conceito de encapsulamento se determina que os atributos não possam ser acessados diretamente, mas somente pela interface da classe (métodos). Para fazer isso é necessário fazer uso das operações Set e Get que representam a gravação e a leitura dos atributos através das Propriedades

O principal do encapsulamento é a preservação das informações e operações do conhecimento externo.

Métodos

Os métodos que definem as ações de um objeto, ou mesmo o seu comportamento são parecidos com as funções da linguagem C. Onde, existe um tipo de retorno, um nome (identificador do método) e uma lista de parâmetros que ele recebe.

A declaração do método pode ser feita da seguinte maneira:

```
[modificador de acesso] [tipo do método] <tipo do valor de retorno>
<identificador do método> ( [lista de parâmetros] )
{
    implementação;
}
```

Quando o método for receber valores externos terão que ser definidos parâmetros. Esses parâmetros são variáveis locais definidas para armazenar os valores enviados ao métodos.

O método pode não receber parâmetros.

Esse método pode retornar ou não valores. Quando ele não retornar seu tipo deve ser definido como void.

Propriedades

As propriedades também são responsáveis pela encapsulamento dos atributos. Quando houver necessidade de ler ou gravar dados em um atributo, primeiro é acionada a propriedade dele. Nessa propriedade existem dois tipos de ações que podem ocorrer: o **get** e o **set**.

O get pode ser definido como um método de leitura(recuperar dados) e o set como um método de modificação (armazenar dados). Nesses métodos existe a palavra reservada **value** que recebe o valor enviado à propriedade.

É possível definir uma propriedade como somente leitura ou somente gravação. Para isso é necessário implementar apenas um dos método get ou set.

Para inserir a propriedade automaticamente, digite prop, pressione a tecla TAB 2 vezes.

Exemplo:

```
public class Pessoa
{
    private int _idade;
    public int Idade
    {
        get { return _idade; }
        set { _idade = value; }
    }
}
```

```
objpessoa.Idade = Convert.ToInt32(textBoxIdade.Text);
//Nesse momento está sendo acionado o set
```

```
Idade = objpessoa.Idade;
//Aqui está sendo acionado o get
```

Exemplo da Classe Pessoa

```
namespace Projeto_classes_1
{
    public class Pessoa
    {
        //definição dos atributos
        private string _nome;
        private int _idade;

        //definição das propriedades
        public string Nome
        {
            get { return _nome; }
            set { _nome = value; }
        }

        public int Idade
        {
            get { return _idade; }
            set {
                if (value > 120 || value < 0)

```

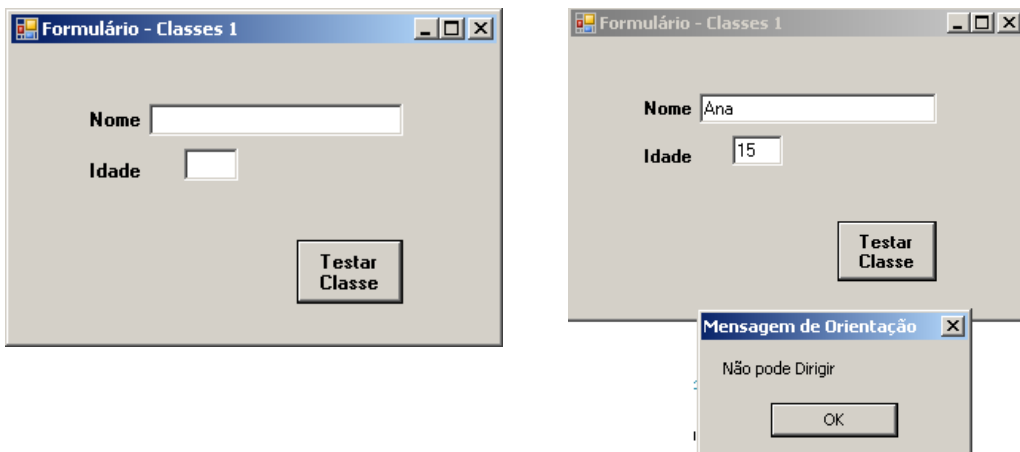
```

        _idade = 0;
    else
        _idade = value;
    }
}

//definição dos métodos
public string VerificarIdade()
{
    string msg;
    if (Idade > 18)
        msg= "Já pode Dirigir";
    else
        msg= "Não pode Dirigir";
    return msg;
}
}
}

```

No Form_Classe os dados serão da pessoa serão digitados nas caixas de texto e quando o usuário pressionar o botão serão armazenadas nas propriedades.



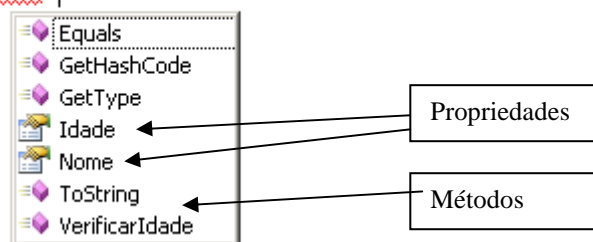
Vamos entender o que este código faz:

Primeiro foi definido o nome do objeto (objpessoa) e de qual classe ele se referência. Depois foi criada uma nova instância da classe através do seu construtor.

```

Pessoa objpessoa = new Pessoa();
objpessoa.

```



Quando for digitado o nome do objeto e pressionar o ponto(.)você verá todas as propriedades e métodos que foram definidos na classe. Mas, não se esqueça que eles só aparecerem porque foram definidos como public.

```
objpessoa.Idade = Convert.ToInt32(textBoxIdade.Text);  
//Nesse momento está sendo enviado a propriedade Idade o conteúdo  
digitado na caixa de texto idade.  
  
objpessoa.Nome = textBoxNome.Text;  
//Está sendo enviado a propriedade Nome o conteúdo digitado na caixa  
de texto nome.  
  
MessageBox.Show(objpessoa.VerificarIdade(), "Mensagem de Orientação");  
/*Primeiro será executado o método verificar. O que for retornado do  
método será exibido na caixa de diálogo criada pelo MessageBox*/
```

Construtores

Os construtores de instâncias são métodos chamados automaticamente toda vez que uma nova instância de uma classe for feita.

O construtor tem o mesmo nome da classe.

Pode haver mais de um construtor para a mesma classe, isso é chamado de sobrecarga (overload) do método.

Lembram do método Show do MessageBox, quando se digita já aparece os diferentes métodos.

O que diferencia um método do outro é o número e a combinação dos parâmetros que ele recebe. Isso é conhecido como assinatura do método.

Os construtores não possuem valor de retorno

O construtor padrão (default) não recebe parâmetros. Quando há a necessidade de iniciar o objeto com valores, é necessário criar um construtor com parâmetros, que é chamado de construtor customizado.

Como exemplo, num jogo que tem que ser definido o número de jogadores para depois ser criado.

```
Jogo objJogo = new Jogo( );  
ou  
Jogo objJogo = new Jogo(1, "Básico");
```

Cuidado: Quando se criar um novo construtor, com parâmetros, deve-se reescrever o construtor default.

Destruitor

O destrutor da classe tem quase o mesmo nome da classe, com o acréscimo do til (~). Ele é executado automaticamente quando um objeto é destruído.

```
class Jogo
{
    ~Jogo( )
    {
        Implementação;
    }
}
```

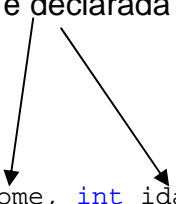
Variáveis

Todas as variáveis são declaradas dentro do escopo de uma classe e podem ser dos seguintes tipos:

- I- **Locais:** são declaradas no escopo de um método ou evento e não possuem modificadores de acesso. Ela é declarada colocando o seu tipo e seu nome (identificador).

Exemplo:

```
public void ReceberDados(string nome, int idade)
{
    _idade = idade ;
    _nome = nome;
}
```



XVII- Herança

Na programação orientada a objetos existe a característica herança, ela está relacionada com a reutilização e padronização de código. Com o processo de herança uma nova classe (classe derivada – subclasse ou classe filha) é criada a partir de uma outra classe (classe base - superclasse ou classe pai).

Essa classe derivada possui todos os atributos e métodos herdados da classe base, e seus próprios atributos e métodos. Todo o código já definido pela classe base será reutilizado pelas classes derivadas, que podem ser várias.

A classe base possui as partes (as características e operações em comum) generalizadas que podem ser definidas para qualquer classe e as classes derivadas tem as partes mais específicas (distintas).

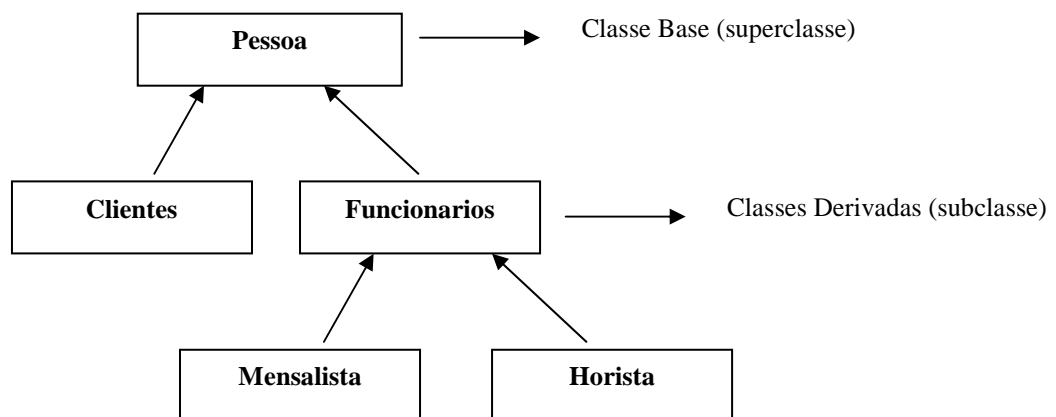
Pense nas seguintes situações:

1ª) Os empregados de uma empresa podem trabalhar em um regime horista (recebem por hora trabalhada) ou mensalista (recebem por mês).

Podemos definir a superclasse Empregado e as classes derivadas: EmpregadoHorista e EmpregadoMensalista.

2ª) Na locadora existem os clientes e os funcionários.

Na programação existirá a superclasse Pessoa e as classes derivadas: Clientes e Funcionários.



Hierarquia entre as classes

Exemplo de Projeto com Construtor e Hierarquia de Classes

Definição da Super classe – classe base

class Funcionario

```
{
    private string _nome;
    private decimal _salario;
    private char _classe;

    public string Nome
    {
        get { return _nome; }
        set { _nome = value; }
    }

    public decimal Salario
    {
        get { return _salario; }
        set { _salario = value; }
    }

    public char Classe
    {
        get { return _classe; }
        set { _classe = value; }
    }

    public Funcionario()           //Construtor Default
    {
    }

    public Funcionario(char classe) //Novo Construtor
    {
        _classe = classe;
    }
}
```

Definição da classe derivada

class Horista : Funcionario

```
{
    private int _qtdHora;
    private float _valorHora;

    public int QtdHora
    {
        get { return _qtdHora; }
        set { _qtdHora = value; }
    }
}
```

```
public float ValorHora
{
    get { return _valorHora; }
    set { _valorHora = value; }
}
```

Especificador de herança Protected

No conceito de herança deve-se tomar cuidado com o especificador de acesso usado para os membros. Qualquer membro público da classe-base é automaticamente um membro privado da classe derivada. Entretanto, nenhum membro privado da classe-base pode ser acessado pela classe derivada. Para resolver esse problema é necessário usar um especificador de acesso intermediário: o **protected**.

Os membros protected são visíveis a todos os membros de uma classe derivada, mas não podem ser acessados diretamente por um objeto declarado fora da classe. Dessa forma se for necessário usar um membro privado da classe-base deve declará-lo como protected.

Não se esqueça:

Todos os membros públicos da classe-base continua sendo públicos na classe derivada e os membros protected da classe-base serão protected da classe derivada. Os membros públicos da classe-base podem ser acessados diretamente pelo objeto da classe derivada.

Quando a declaração é private todos os membros herdados tornar-se privados, não importa qual a especificação feita (public ou protected). Esses membros são acessíveis somente na classe derivada, mas não diretamente aos objetos.

```
class Produtos
{
    private int _codigo;
    private string _nome;
    protected decimal _valor;
    protected string _categoria;

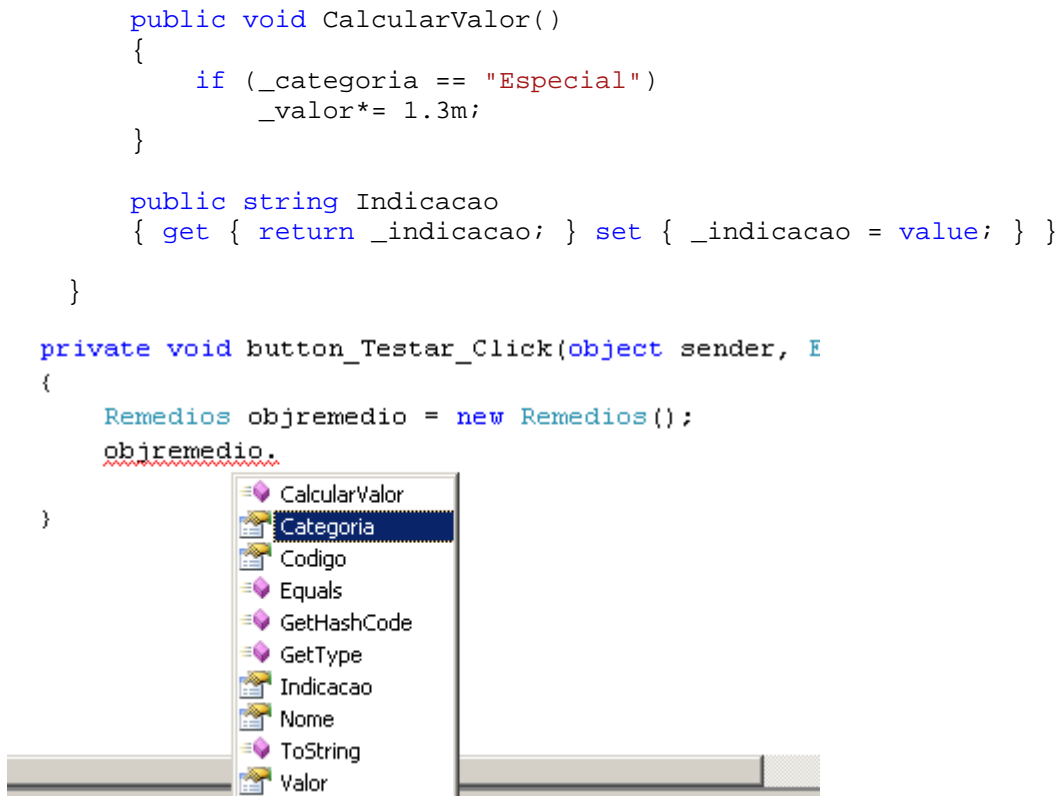
    public int Codigo
    { get { return _codigo ; } set { _codigo = value ; } }

    public string Nome
    { get { return _nome; } set { _nome= value; } }

    public decimal Valor
    { get { return _valor; } set { _valor = value; } }

    public string Categoria
    { get { return _categoria; } set { _categoria = value; } }
}
```

```
class Remedios : Produtos
{
    private string _indicacao;
```



Veja que na hora de criar o objeto aparece todos as propriedades definidas na classe base.

No final é executado o método CalcularValor da classe derivada, que usa atributos da classe base, definidos como protegidos.

```

private void button_Testar_Click(object sender, EventArgs e)
{
    Remedios objremedio = new Remedios();
    objremedio.Valor = 2;
    objremedio.Categoria = "Especial";
    objremedio.CalcularValor();
    MessageBox.Show("Valor do remédio " + objremedio.Valor.ToString());
}

```



XVIII-Polimorfismo

O polimorfismo é uma característica da POO que possibilita um método ter várias formas diferentes, ou seja, ações diferentes, com a mesma assinatura. Para que a implementação do polimorfismo seja feita é necessário definir **Métodos Virtuais**.

Cada nova classe derivada pode entender um método de forma diferente e aplicar seus próprios códigos de modo a redefinir seu comportamento.

Métodos Virtuais

Métodos virtuais são métodos definidas na classe base e que podem ser redefinidas (diferentes) em classes derivadas. O método virtual se caracteriza pela capacidade do compilador detectar mudanças de acordo com o contexto da classe e viabilizar o polimorfismo.

Para essa definição iremos usar os termos **virtual** e **override**, um completa o outro.

```
class Classe_Base
{
    //é declarado o método que pode ser sobrescrito
    public virtual string Exibedados()
    {
        string texto;
        texto = "Estou na classe base";
        return texto;
    }
}

class Classe_Derivada: Classe_Base
{
    //é sobrescrito o método definido na classe base
    public override string Exibedados()
    {
        string texto;
        texto = base.Exibedados()+" Estou na classe derivada";
        return texto;
    }
}

private void buttonPolimorfismo_Click(object sender, EventArgs e)
{
    Classe_Derivada objderivada = new Classe_Derivada();
    MessageBox.Show(objderivada.Exibedados());
}
```



Na classe base foi definido o método como virtual e na ou nas classes derivadas como override. Dentro de cada método override, é usada a palavra-chave **base**. Ela é essencial para permitir que o polimorfismo. As classes agregam mais informação.

A grande vantagem é se pode gerar novas versões dos métodos, você não precisa necessariamente conhecer o código da classe pai para fazer a nova implementação. Se quiser, pode até mesmo descartar o que foi escrito anteriormente, basta não incluir a chamada através da palavra-chave base.

XIX- Classes abstratas

Com as classes abstratas não é possível realizar um instância. São classes feitas para servir como modelos para suas classes derivadas. Ela não traz uma implementação concreta. Ela serve para dar forma a um projeto de classe.

Criar uma classe abstrata significa definir premissas, onde nada é codificado concretamente na primeira geração da classe.

```
abstract class EntradaDeDados
{
    public string _salario;
    public abstract string Conteudo( );
    public abstract string Exibir( );
    public abstract void Reajustar ( );
}
```

Os métodos abstract somente existe na classe abstract.

As classes filhas (através de override) é que definirão a funcionalidade de cada método ou propriedade. Se você tentar instanciar uma classe abstrata, obterá um erro do compilador:

Na classe filha, você teria as implementações:

```
class EntradaDeOcorrencias : EntradaDeDados
{
    public override string Conteudo( )
    {
        // Implementação
        return "";
    }

    public override string Exibir( )
    {
        // Implementação
        return "";
    }

    public override void Reajustar()
    {
        _salario+=500;
    }
}
```

Um método abstrato é implicitamente virtual.

Todo método declarado como abstract deve ser necessariamente sobreposto.

XX- Selando as classes

A linguagem C# permite criar classes seladas. Por definição, classes seladas não permitem nenhum tipo de herança ou derivação. Uma classe selada deve ser usada diretamente no programa, é o oposto de classe abstrata.

```
sealed class FimDePapo
{
    public string UltimaPalavra;
}

class AindaQueroMaisPapo : FimDePapo
{
    // Erro de compilação:
    // cannot inherit from sealed class 'FimDePapo'
}
```


XXI- Compartilhar dados entre os formulários e Formulário com Validação de Usuário e Senha

Como no C# não existe variáveis globais, usado na linguagem C, que pode ser usada e alterada por qualquer função, é feito uso da declaração **estática (static)** de uma classe ou de seus atributos. Como exemplo, podemos ter no formulário principal a entrada de um usuário e sua senha e depois precisar fazer uso desses dados nos outros formulários (validação, privilégios, segurança).

Quando o membro da classe for definido como estático, ele só poderá ser chamado usando o nome da classe, ou seja, não se cria uma instância (objeto) para usá-lo.

Então temos os membros estáticos e instâncias. Os membros de instância precisam antes de ser usados terem instanciado a sua classe. Por default os membros são instâncias, quando for necessário ter um estático é necessário usar a definição (modificador) static.

Somente existe uma cópia na memória de um atributo estático, por isso quando é feito seu uso a alteração de seu valor é direta. No projeto usado como exemplo será explicado seu uso.

1º Passo:

Você deve criar 3 formulários: um para o menu, um para a abertura (usuário e senha) e outro para entrada de dados.

2º Passo:

Inicialmente não faça nada no formulário Menu. Ele está sendo feito para fazer a chamada do formulário abertura. Através deste formulário você mandará exibir o formulário de Abertura, onde se faz a validação de usuário. Para isso deve ser colocada a instrução na hora em que ele está sendo carregando, no evento Load.

```
private void FormMenu_Load(object sender, EventArgs e)
{
    Form_abertura objform_abertura = new Form_abertura();
    objform_abertura.ShowDialog();
}
```

3º Passo:

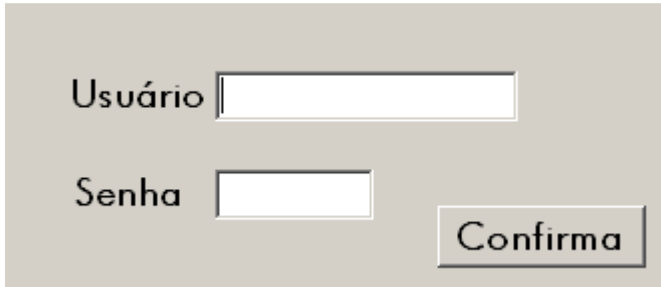
Nesse formulário sera definido dois atributos estáticos, um para usuário e outro para senha.

```
public partial class Form_abertura : Form
{
    public static string senha;
```

```
public static string usuario;

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

Depois disso no formulário de Abertura faça a validação através do botão Valida.



```
private void buttonValida_Click(object sender, EventArgs e)
{
    if (textBoxSenha.Text == "123")
    {
        FormMenu objform_menu = new FormMenu();
        usuario = textBoxUsuario.Text;
        this.Close();
    }
    else
        MessageBox.Show("Senha Inválida", "Cuidado:");
}
```

4º Passo:

No formulário Menu, altere a propriedade **IsMdiContainer** para **True**. Coloque o menuStrip e adicione a opção Cadastrar Dados



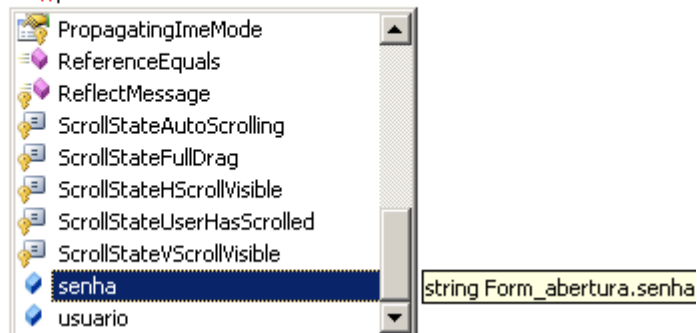
```
private void
cadastroToolStripMenuItem_Click(object sender, EventArgs e)
{
    FormEntrada objform_entrada = new FormEntrada();
    objform_entrada.MdiParent = this;
    objform_entrada.Show();
}
```

}

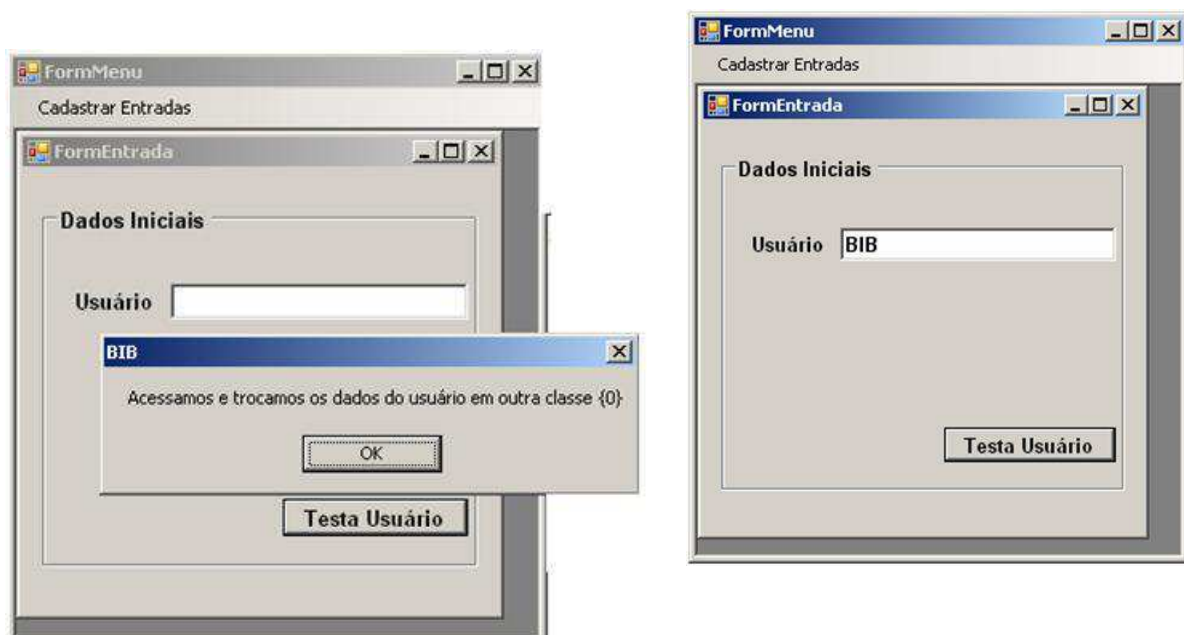
4º Passo:

Como será necessário usar o conteúdo digitado no formulário de Abertura na caixa de texto do usuário, podemos utilizar o atributo definido como estático que está na classe formulário de abertura. Veja que somente podemos fazer o acesso a ele através da sua classe. Quando digitarmos o nome da classe ele já aparece sem precisamos fazer uma nova instância dessa classe formulário.

```
private void buttonTestar_Click(object sender, EventArgs e)
{
    if (Form_abertura.
```



```
private void buttonTestar_Click(object sender, EventArgs e)
{
    if (Form_abertura.usuario == "ADM")
    {
        Form_abertura.usuario = "BIB";
        MessageBox.Show("Acessamos e trocamos os dados do
usuário em outra classe {0}", Form_abertura.usuario);
        textBoxUsuario.Text = Form_abertura.usuario;
    }
}
```



Dicas:**1ª) Para substituir a tecla TAB pelo ENTER**

Primeiro modifique a propriedade KeyPreview para True.
No evento KeyDown

```
private void Form_abertura_KeyDown(object sender, KeyEventArgs e)
{
    if (e.KeyCode == Keys.Return)
    {
        SelectNextControl(this.ActiveControl, true, false, false, true);
    }
}
```

2ª) Fazer uso do botão Sair e da opção Sair da Barra de Título

No botão Sair deve ser colocada a instrução:

```
private void buttonSair_Click(object sender, EventArgs e)
{
    this.Close();
}
```

Não pode ser esquecido de colocar no evento Closing do formulário a definição da caixa de diálogo.

```
private void Form_abertura_FormClosing(object sender,
FormClosingEventArgs e)
{
    if (MessageBox.Show("Deseja Sair", this.Text,
        MessageBoxButtons.YesNo, MessageBoxIcon.Question,
        MessageBoxDefaultButton.Button2) == DialogResult.No)
    {
        e.Cancel = true;
    }
}
```

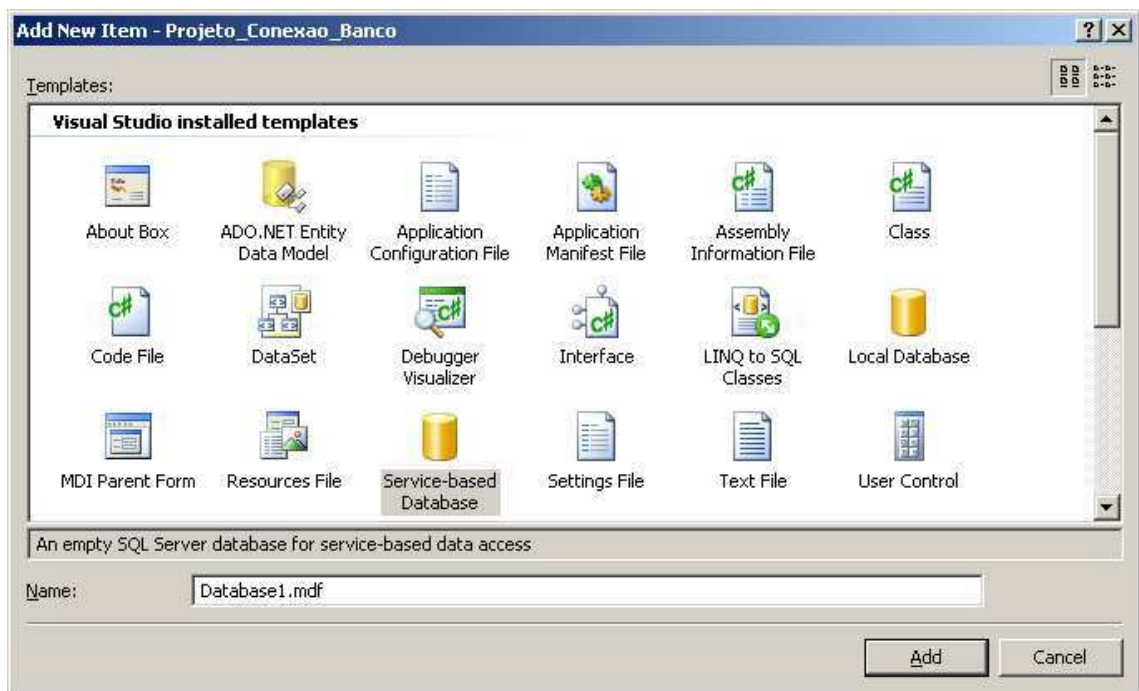
XXII- Armazenamento de Dados

Primeira Forma

Para criar uma base de dados (banco de dados) para armazenar os dados cadastrados ou a serem usados pelos formulários é necessário criar essa base de dados e sua conexão com o projeto. O Gerenciador de Banco de Dados padrão .Net é o Microsoft SQL Server , versão SQL Express. O projeto que vou criar como exemplo para vocês é em Express, mas você poderia criar bancos de dados em Oracle, Access, entre outros.

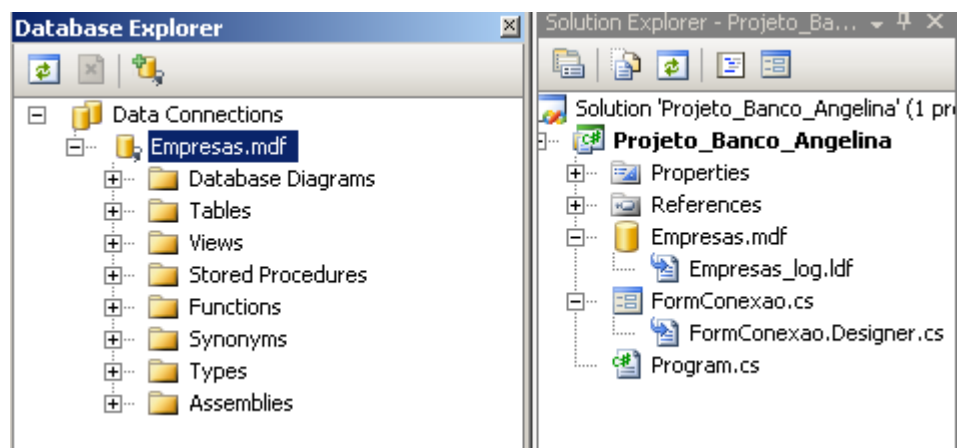
Abaixo serão colocados os passos necessários para isso:

Adicione um novo Item ao seu projeto e escolha a opção Service-Based Database (versão 2008 ou SQL Database versão 2005). Altere o nome para Empresas.mdf e pressione o botão Add.



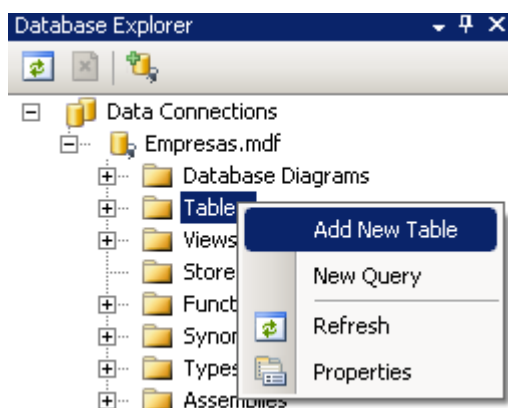
Quando aparecer uma janela Data Source Configuration Wizard, pressione o botão Cancel.

Ao olhar no Solution Explorer estará aparecendo o arquivo Empresas.mdf que você criou, ao dar um duplo clique irá aparecer no lado da esquerda o explorador (Database Explorer) do banco de dados. Dependendo da versão é Server Explorer (2005).

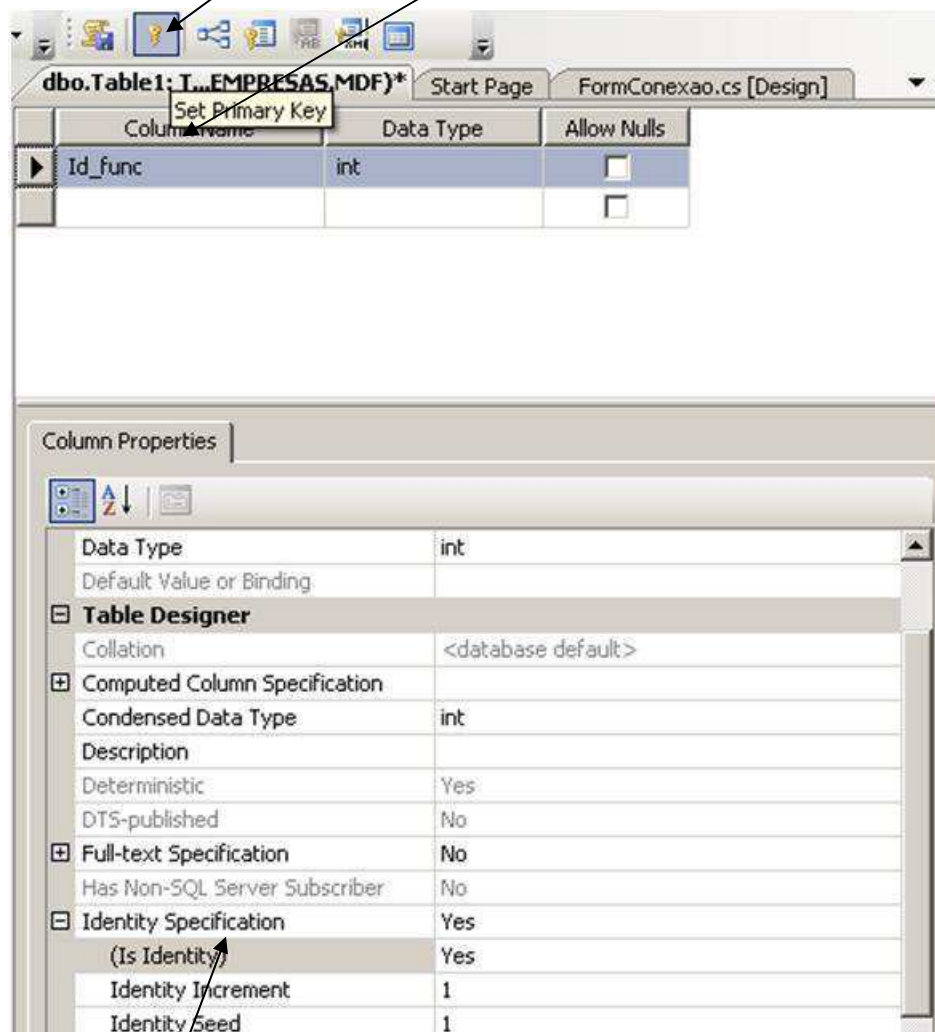


Através desse gerenciador de banco de dados é possível armazenar de forma organizada e inter relacionada os dados, além de fazer consultas, exclusões e alterações de forma rápida. Essa organização que ele faz é através de tabelas, iguais ao Access, que nós já vimos. Essas tabelas tem os registros (linhas) e campos (colunas).

Vamos começar criando a tabela de funcionário. Pressione o botão direito em cima da opção Tables e escolha a opção Add New Table.



Para inserir os campos dos funcionários, clique na coluna e informe o nome (identificação) do campo, depois o tipo de dado que ele irá conter, a última coluna é para definir se aceita ou não nulos (allow nulls). Se o campo tiver que ser definido como campo principal (chave primária), você deve ativar a ferramenta com o desenho de chave.



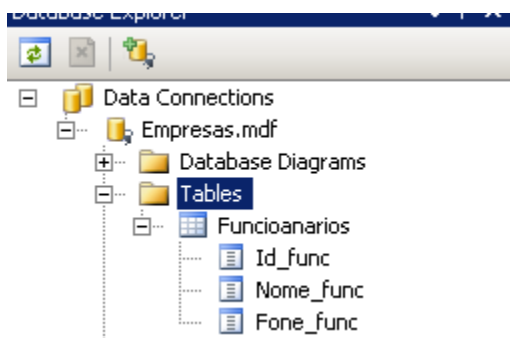
Abaixo existem as propriedades desses campos, quando o campo for autonumeração (deve ser atualizado automaticamente) deve ser ativa a opção (Is Identity) como Yes.

Devem ser criados os campos:

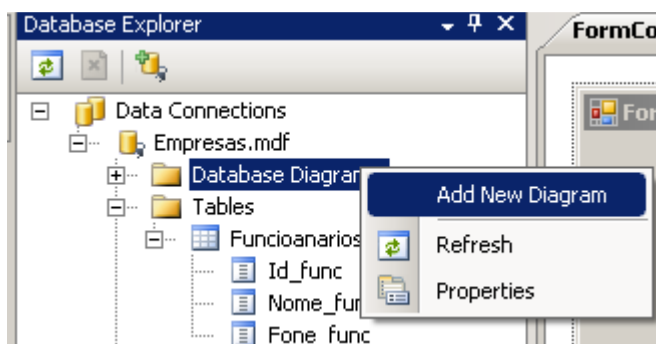
Nome do campo (Column Name)	Data Type	Allow Nulls
Id_func	int	<input type="checkbox"/>
Nome_func	nvarchar(30)	<input type="checkbox"/>
Fone_func	nvarchar(20)	<input type="checkbox"/>

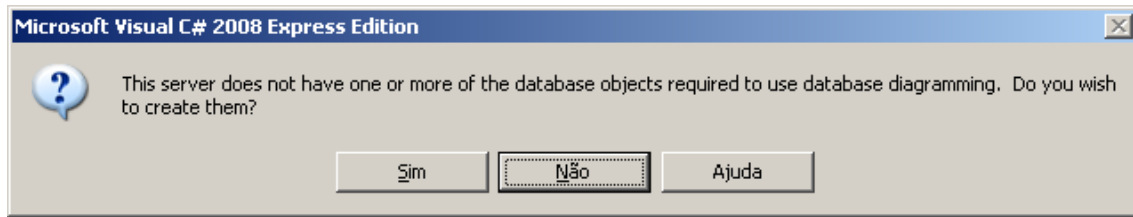
Após adicionar todos os campos, pressione a ferramenta Salve All do projeto. Aparecerá uma tela para salvar a tabela, coloque o nome Funcionario.

Dê um duplo clique na pasta Tables e veja a tabela criada com suas respectivas colunas.



Após criarmos as tabelas devemos fazer o Diagrama de Banco de Dados. Para isso clique com o botão direito sobre a opção Database Diagrams.

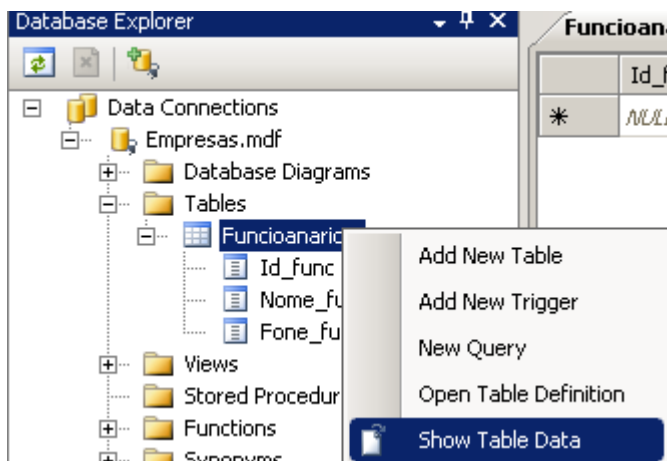




Como ele podemos ter comandos SQL gerados automaticamente, além de fazermos os relacionamentos das tabelas. Quando aparecer a mensagem acima pressione SIM, de forma que ele vai criar procedimentos que permitam a interação do seu banco com seus códigos. Selecione a tabela, clique no botão ADD e depois no CLOSE.

Salve (Salve All) o projeto e dê um nome para seu Diagrama, como Diagrama_Empresa. A partir de agora serão gerados códigos automáticos a partir desse Diagrama.

Se você quiser adicionar dados diretamente, você pode, basta clicar na opção Show Table Data (Mostrar dados da tabela).

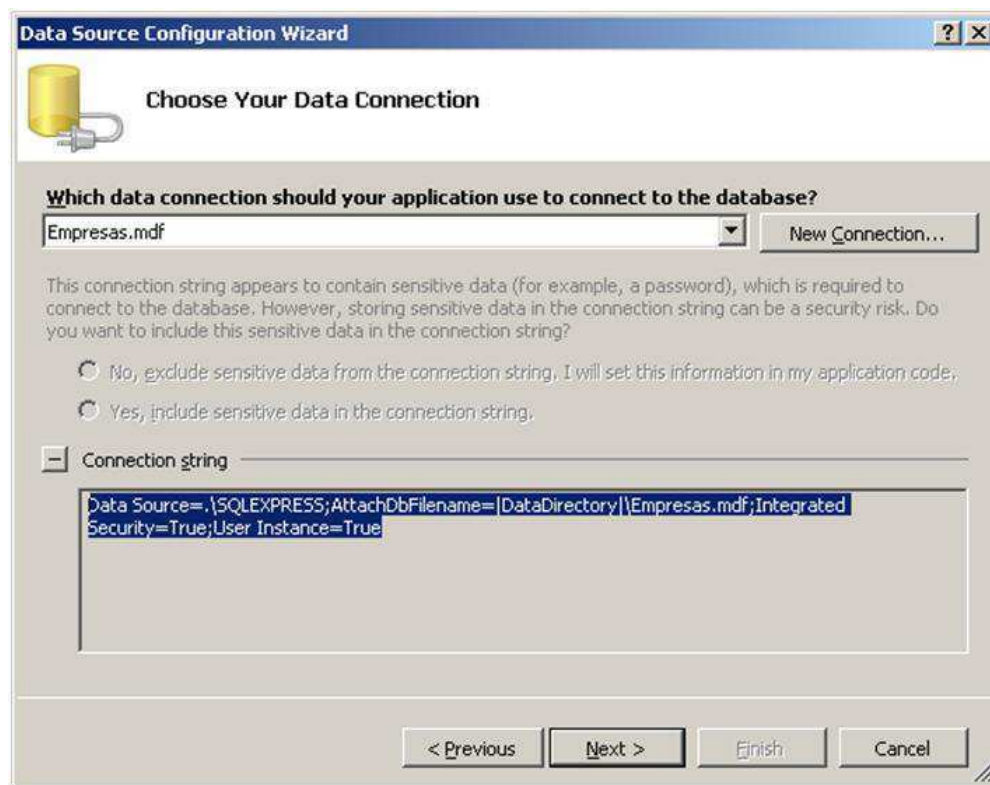


Basta digitar os dados dos funcionários, não se esqueça que não pode digitar o ID (que é automaticamente inserido). Tudo o que você digitar estará sendo salvo no banco criado.

Depois vá no menu e escolha a opção Add New Data Source (Adicionar Nova Origem de Dados).

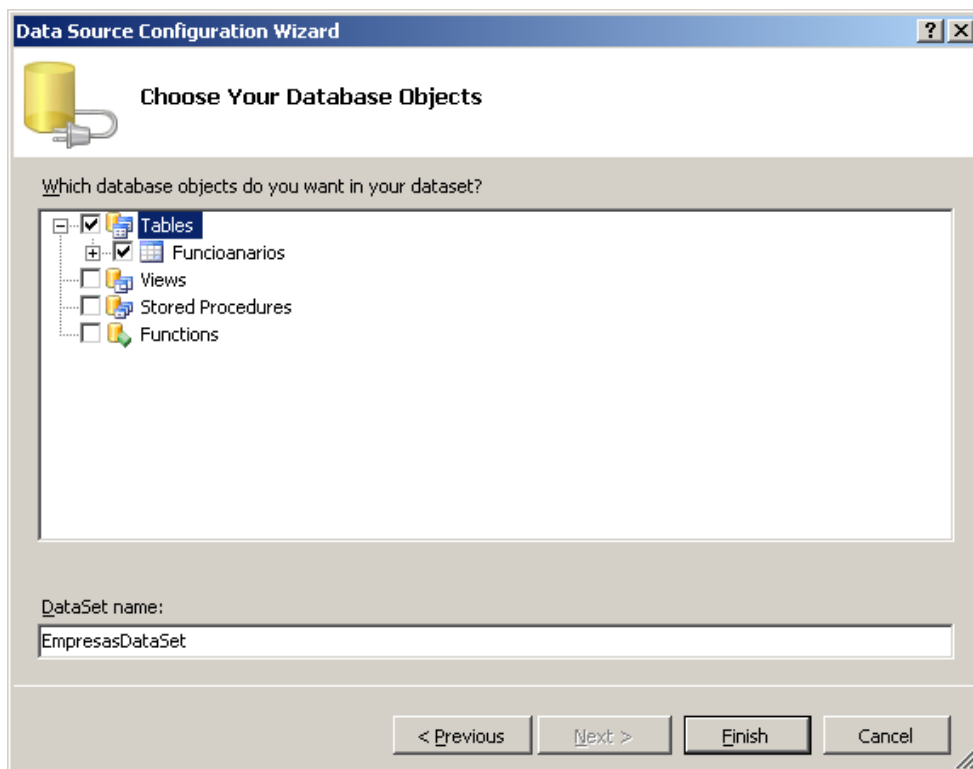


Como você vai se conectar ao Banco de Dados criado escolha a opção Database e depois selecione o Empresas.MDF que você criou anteriormente.



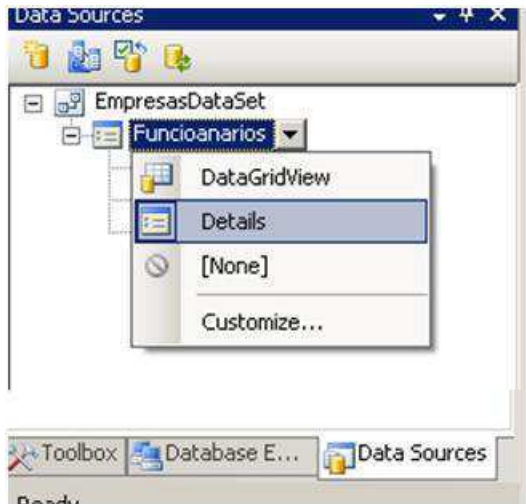
Veja a string de conexão que ele já criou para você automaticamente.

Pressione o Next e na próxima tela ative a opção Tables para selecionar as tabelas criadas.



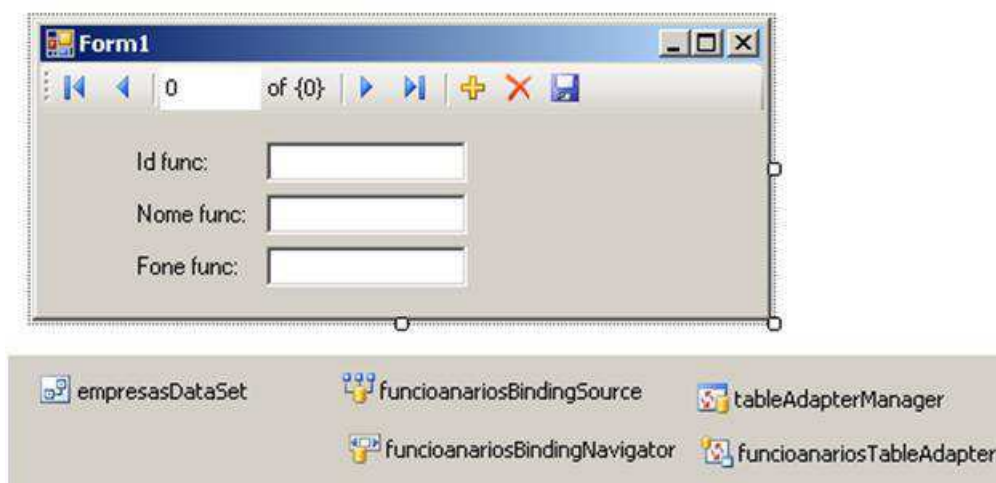
Pressione o Finish e pronto.

Clique no Menu na opção DATA e escolha Show Data Source, essa aba irá aparecer do lado da esquerda.



Pressione a seta para escolher a forma de inserir a tabela na tela. A primeira faz aparecer numa tabela (datagrid) e o segundo em caixas de textos (details).

Depois clique no nome da tabela e arraste-a para o formulário.



Pronto!!! Você já pode cadastrar, consultar, excluir, alterar os dados dos seus funcionários.

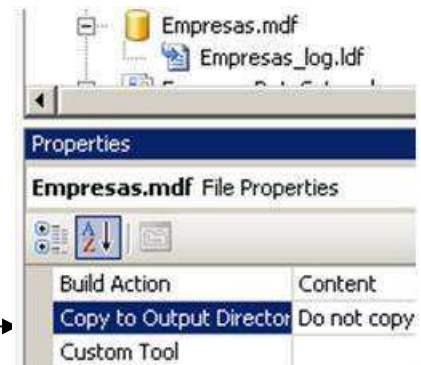
Veja os componentes que ele automaticamente inseriu para você.

- empresasDataSet
- funcionariosBindingSource: conecta com a tabela funcionário
- funcionariosBindingNavegador: faz aparecer a barra de navegação (botão para inserir, salvar, excluir) ligada a tabela funcionário (pelo funcionariosBindingSource)

- `tableAdapterManager`: permite que seus controles interajam com os comandos SQL que foram gerados automaticamente.

Cuidado:

Existe duas cópias do banco que você criou `Empresas.mdf`, uma na pasta do projeto e outro na pasta `Debug`. Qual é o original? É o da pasta projeto. O outro é inserido sempre quando compilamos (F5). Para não haver conflito e perda de dados, altere a propriedade do Banco de Dados. Dessa forma o compilador não mais criará uma cópia no `Debug` ao compilar.



Depois altere o arquivo **app.config**, que tem as configurações da sua aplicação. Nesse arquivo tem o caminho que ele procura em tempo de execução pelo banco de dados. O caminho deve estar correto, não importa em que máquina vá ser executada, na sua, na faculdade ou na máquina do professor (minha em casa). Quando estiver numa rede deve ser informado o ip do servidor.

```
connectionString="Data
Source=.\SQLEXPRESS;AttachDbFilename=|DataDirectory|\Empresas.mdf;Integrated Security=True;User Instance=True"
```

Altere esse caminho, pode ser na raiz da máquina:

```
connectionString="Data
Source=.\SQLEXPRESS;AttachDbFilename=|DataDirectory|c:\Apostila\Projeto_Banco_Angelina\Projeto_Banco_Angelina\Empresas.mdf;Integrated Security=True;User Instance=True"
```

XXIII-Conexão com o Banco de Dados

Diretivas

Ao usar um banco do SQL Server é necessário incluir a diretiva `SQLCliente`, que tem as classes para se trabalhar com esse tipo de banco.

using System.Data.SqlClient; //tem as classes para trabalhar com o SQLServer

Referências globais

A variável usada como referência a string de conexão e a referência da conexão e comando podem ser definidas dentro da classe do formulário como um atributo. Dessa maneira poderá ser usada em todos os eventos.

```
string conexao;  
SqlConnection con;  
SqlCommand com;
```

String de Conexão (Connection String)

Pode ser criada nas propriedades do projeto. Indo no projeto, na opção Properties, na Settings. Também pode ser retirada nas propriedades do Banco de Dados.

```
conexao = "Data Source=.\SQLEXPRESS;  
AttachDbFilename=C:\Caminho\ \Clientes.mdf;Integrated Security=True;User  
Instance=True";
```

```
conexao = @"Data Source=.\SQLEXPRESS;  
AttachDbFilename=C:\Caminho\ \Clientes.mdf;Integrated Security=True;User  
Instance=True";
```

```
conexao = Projeto.Properties.Settings.Default["projstring"].ToString();
```

Inserção de dados

Não se esqueça que primeiro tem que haver a conexão para depois fazer a inserção dos dados, como no código explicado a seguir:

- 1º Atribui a variável de conexão
- 2º Cria uma instância de conexão – usando a variável de conexão. Se quiser pode enviar a string de conexão diretamente pelo construtor.
- 3º Crie uma variável com a instrução SQL de inserção de dados
- 4º Inicia a conexão com o banco
- 5º Cria uma instância de comando SQL, passando para o construtor a instrução SQL (inserir no caso)
- 6º Executa a instrução SQL na conexão atual
- 7º Fecha a conexão no término

```
SqlConnection objconexao = new SqlConnection(conexao);
```

//faz a criação da instancia de conexão, passando para seu construtor a string de conexão atribuída na variável conexão.

```
string nome;
```

```
nome = textBoxNome.Text;
```

```
//declara e atribui ou primeiro declara e depois atribui o valor da variável
```

```
int idade = int.Parse(textBoxIdade.Text);  
    //cria variáveis com as devidas conversões de dados – string para inteiro
```

```
string inserir;  
inserir = "INSERT INTO cliente(nome, idade) VALUES (' " +nome+" ', '+idade+"  
');" ;
```

Outra forma de fazer a atribuição dos valores digitados – usando parâmetros

```
string inserir = "INSERT INTO cliente(nome, idade) VALUES (@nome,@idade)";  
objcom.Parameters.Add("@nome", SqlDbType.VarChar).Value =  
textBoxNome.Text;  
ou  
objcom.Parameters.AddWithValue("@idade", textBoxIdade.Text);
```

```
SqlCommand objcom = new SqlCommand(inserir, objconexao);  
    Cria-se uma instancia do tipo SqlCommand, passando para ela a instrução SQL  
e em qual conexão será executada.
```

Também poderia ser:

```
SqlCommand objcom = new SqlCommand();  
objcom.CommandText = consulta;  
objcom.Connection = con;
```

```
objconexao.Open();  
    //abre-se (inicia) a conexão com o Banco de Dados
```

```
objcomando.ExecuteNonQuery();  
    //executa a instrução SQL usando o método ExecuteNonQuery()  
    //esse método executa uma ação no banco de dados - inserção, alteração ou  
exclusão
```

Consulta/Pesquisa de um ou mais registros

Pode ser feita a consulta/pesquisa de registros específicos em uma tabela. Para isso o procedimento é muito parecido com o de inclusão, com algumas mudanças:

```
string consulta = "select count(*) from cliente where nome = '"+  
textBoxNome.Text +"' " ;
```

```
SqlCommand objcon = new SqlCommand(consulta, con);
```

```
if ((int) objcon.ExecuteScalar() > 0)  
    MessageBox.Show("Nome cadastrado");  
else  
    MessageBox.Show("Nome não cadastrado");
```

O método ExecuteScalar retorna os registros de acordo com a instrução SQL

Exibição de todos os registros retornados

```
string consulta = "select * from cliente ";
```

```
SqlDataReader objler; //retorna as linhas da tabela como somente leitura
objler= objcon.ExecuteReader();
```

```
//ou
```

```
SqlDataReader objler = objcon.ExecuteReader();
```

```
while (objler.Read() == true)
```

```
{
    if (objler.Read() == true)//o método Read retorna True se houver linhas
    {
        textBox1.Text = ler["nome"].ToString();
        textBox2.Text = ler["idade"].ToString();
        registro = string.Concat(textBox1.Text, " ", textBox2.Text);
        listBox1.Items.Add(registro);
        //ou o índice textBox1.Text = ler[0].ToString();
    }
    MessageBox.Show("Cadastro "+i.ToString());
    i++;
}
}
```

OU

```
DataSet objds = new DataSet();
SqlDataAdapter objdados = new SqlDataAdapter(consulta, con);
objdados.Fill(objds);
dataGridView1.DataSource = objds.Tables[0];
```

Exclusão de Dados

```
DELETE FROM Clientes where nome = " + vNome + ""
```

Alteração dos Dados

```
UPDATE Clientes SET Nome =" + vNome+ "", "endereço=" + vEndereço + "" WHERE
nome=" + vNome + "";
```

Linguagem SQL

É através da instrução SQL (Linguagem Estruturada de Consulta – **Structured Query Language**) é possível fazer acesso as tabelas. Essa linguagem possui uma sintaxe especial e tem instruções relacionadas a banco de dados (armazenamento, exclusão, alteração, consulta).

ADO.NET

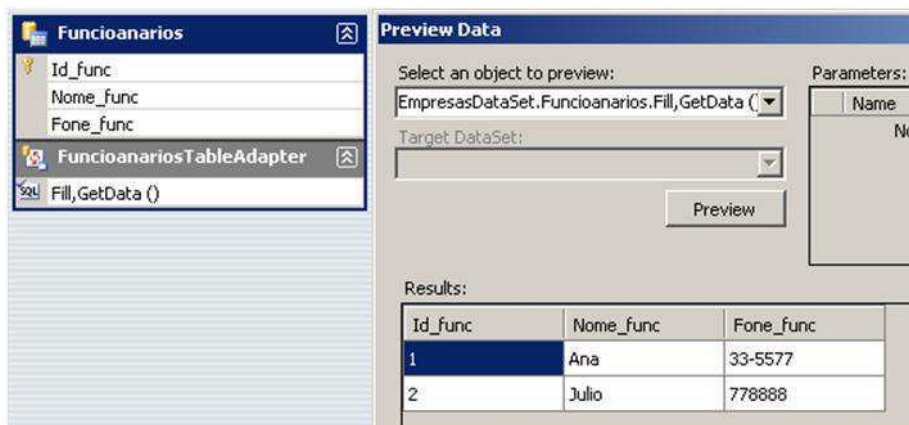
O ADO.net (que vem no framework) tem todas as classes necessárias para trabalhar com banco de dados, desde seu acesso até a sua manipulação. Elas estão no namespace System.Data. Com o ADO você não precisa estar conectado com o banco para trabalhar.

Dependendo do banco que você deseja acessar precisará incluir outras namespaces, como:

- System.Data.OleDb para Access ou SQL Server versão anterior a 7.0
- System.Data.SqlClient para SQL Server versão a partir da 7.0
- System.Data.OracleClient para Oracle.

DataSet

Quando se clica no EmpresasDataSet.xsd você visualiza a uma representação do seu banco de dados com as tabelas que foram criadas. Nele existe os métodos que manipulam os dados. Para você testá-lo basta dar um clique com o botão direito no método Fill, GetData() e escolher o Preview Data, depois o botão Preview.



XXIV- Apêndice

a) .Net

- Possibilita que os aplicativos baseados na Web possam ser distribuídos para vários dispositivos (telefones celulares, PDAs, desktops).
- Oferece um novo modelo de desenvolvimento de software que permite que aplicativos feitos em linguagens diferentes se comuniquem. Dessa forma facilitando para a equipe de desenvolvimento (integração da solução)
- Reutilização de software (uso da biblioteca de classes)
- Os programas escritos em diferentes linguagens (que são compilados para a Linguagem Intermediária) podem ser unificados.
- O software (escrito uma única vez) pode ser executado em multi-plataformas, baste ter a plataforma .net

b) Orientação a eventos

- Uma linguagem orientada a eventos é uma linguagem capaz de responder a determinados “acontecimentos” dentro de um determinado ambiente. Como por exemplo: o clique do mouse, uma tecla pressionada
- Programar com orientação a objetos combinada com orientação a eventos significa criar métodos que serão acionados quando determinadas situações ocorrerem.
- Geralmente, esses métodos têm nomes como Obj_Click (Obj_Clicar), Obj_KeyPress (Obj_PressionarTecla), assim facilitando sua identificação (Lima, 2002).

c) Tipos de Janelas e aplicações

- MDI (Multiple Document Interface): Suporta vários documentos abertos simultaneamente, como o Word por exemplo.
- SDI (Single Document Interface): Permite a abertura de apenas um documento por vez, como Paint, calculadora.
- Janelas modais. São janelas do tipo informativas (diálogos).

d) Programação Orientada a Objetos

Classes:

- As classes são criadas para representar (categorizar) um conjunto de entidades (objetos) que possuem as mesmas características e comportamento (ou mesmo similares) dentro de um contexto.
- Qualquer substantivo pode ser representado por um objeto. Os objetos têm propriedades (atributos: cor, tamanho) e executam ações (comportamento: mover, dormir, desenhar).
- Ou, as classes representam grupos de objetos relacionados.
- Com as linguagens orientadas a objetos os programadores podem programar de modo a refletir o modo em que eles percebem o mundo.
 - Atributo: são as características (dados) de uma classe.
 - Métodos: é uma ação executada por um objeto (comportamento)

Objeto (instância de uma classe):

- Esse objeto pode ser concreto ou abstrato, ou seja, que exista fisicamente (carro) ou apenas conceitualmente (reserva de viagem).
- Quando definimos um objeto estamos fazendo a instância de uma classe. É nesse momento que é usado o espaço de memória.

Exemplos

- Exemplos de atributos:
 - aluno: nome, telefone, data de nascimento, curso, ...
 - professor: nome, telefone, data de nascimento, graduação,
 - funcionário: nome, cargo, salário, data de nascimento, data de admissão, ...
 - carro: ano, marca, modelo, cor,...
 - retângulo: largura, altura, cor da linha, cor de preenchimento,...
- Exemplos de operações:
 - pessoa física: cadastrar dados, consultar dados, ...
 - aluno: lançar notas, calcular desconto, lançar faltas, alterar dados, ...
 - professor: alterar dados, alterar plano de carreira, ...
 - funcionário: reajustar salário, obter salário líquido, obter desconto refeição,
 - carro: obter preço de venda, consultar tabela, efetuar a venda ...
 - retângulo: desenhar, mudar cor da linha, mover, redimensionar,...
- A receita de bolo pode ser um exemplo de classe

- Cada bolo feito com aquela receita é um objeto, ou seja, uma instância da classe bolo.
- Cada bolo feito através daquela receita é diferente, pode ser de sabor diferente,

Biblioteca de Classes da Plataforma .Net

- (FCL – Framework Class Library)
 - Usar componentes da biblioteca reduz o tempo de implementação de uma aplicação

XXV- Bibliografia Utilizada

- **C# como Programar – H. M. Deitel; P.J. S.P: Pearson Makron Books, 2003**
- **C# Desenvolvendo uma Aplicação Comercial - Série Visual Studio 2005 Express Editions. Alexandre Tarifa, Facunte, Marcus Garcia: Brasport.**
- **C# e .Net para desenvolvedores. Edwin Lima, Eugênio Reis. Campus, 2002**
- **Desenvolvendo Aplicações com UML 2.0- do conceitual à implementação. Ana Cristina Melo, Brasport, 2004.**
- **Use a Cabeça! C#- Andrew Stellman & Jennifer Greene, Alta Books.**