


# Apostila de Técnicas de Programação

Professor Eduardo Rosalém Marcelino

(2010 / 2014)

[eduardormbr@gmail.com](mailto:eduardormbr@gmail.com)

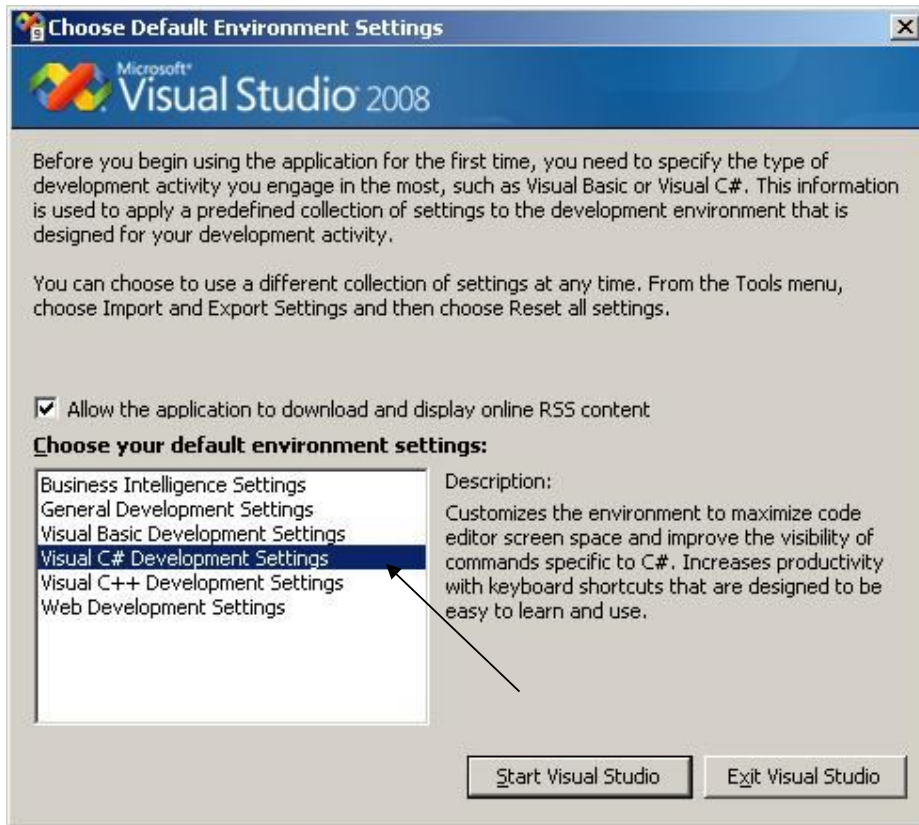
## Conteúdo

Programação Visual .....	3
Principais teclas de atalho .....	3
Criando um novo projeto Windows Forms: .....	4
Projeto Olá Mundo.....	6
Alterando propriedades dos objetos (componentes) via código .....	7
Executando métodos dos objetos.....	8
Nomeando os componentes .....	9
Componente ComboBox  .....	10
Criando um cadastro de Aluno .....	11
Classe VO (Value Object) .....	12
Classe de Conexão com o Banco de dados.....	14
Classe DAO para realização das operações de acesso ao banco de dados .....	17
Formulário de Cadastro.....	19
Melhorando o código antes de prosseguir com o cadastro .....	23
Criando um método para efetuar consulta .....	25
Métodos para navegar nos registros .....	26
Finalizando o cadastro .....	28
Listando valores de uma tabela em um ComboBox .....	32
Evitando erro de chave duplicada.....	34
Sugerindo um código de aluno automaticamente .....	35
Criando uma tela de Consulta .....	36
Recuperando o último valor inserido de um campo identity.....	40
Salvando e Recuperando Imagens no Banco de Dados .....	41
Armazenando dados temporários com DataTable e com DataGridView .....	44



## Programação Visual

Ao abrir o visual Studio, recomendo utilizar a seguinte configuração:

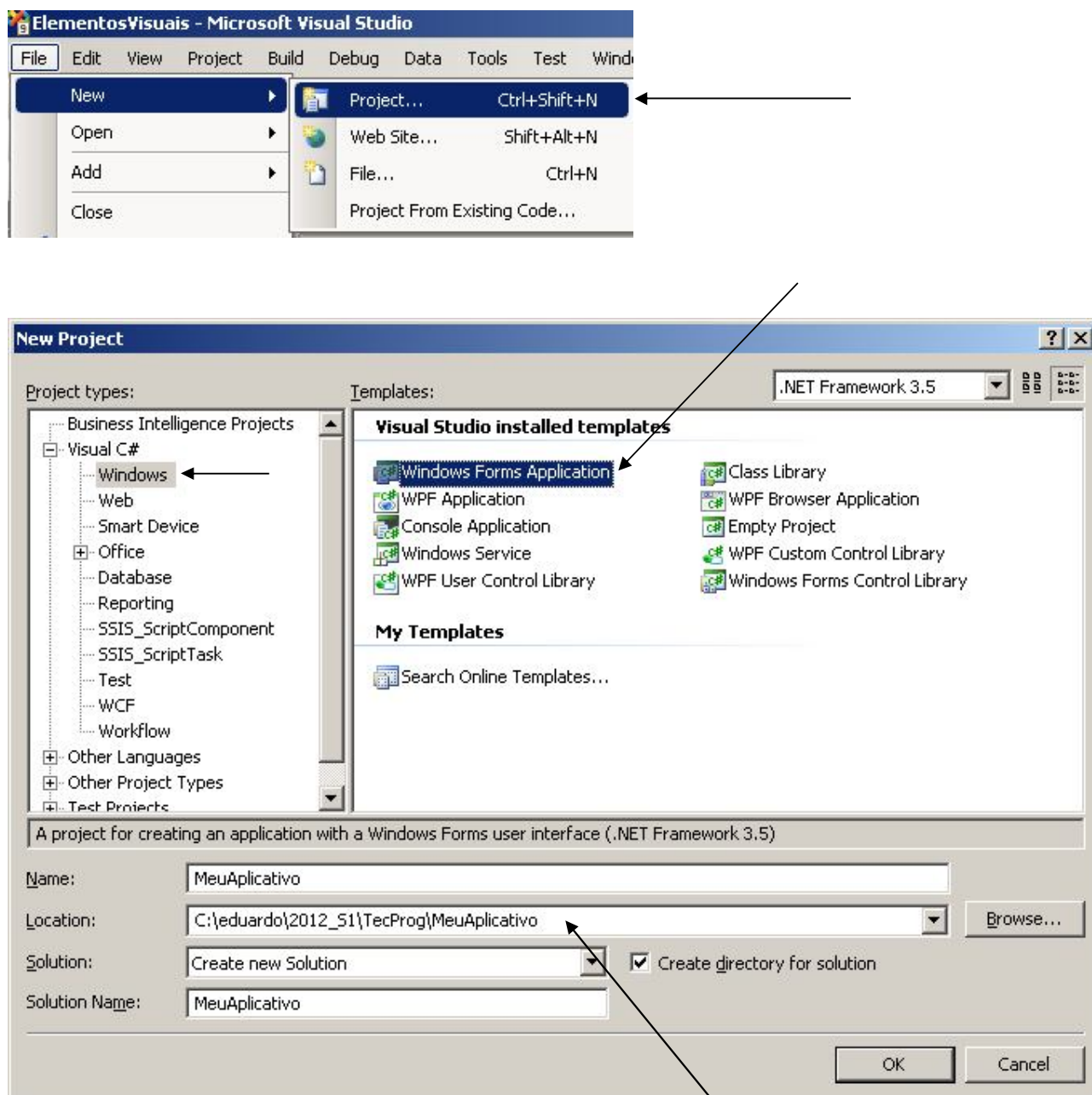


## Principais teclas de atalho

Teclas de atalho que podem ser usadas quando o Visual Studio tela estiver configurada como na imagem acima

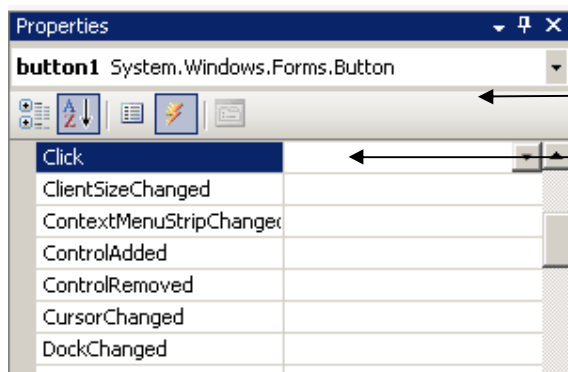
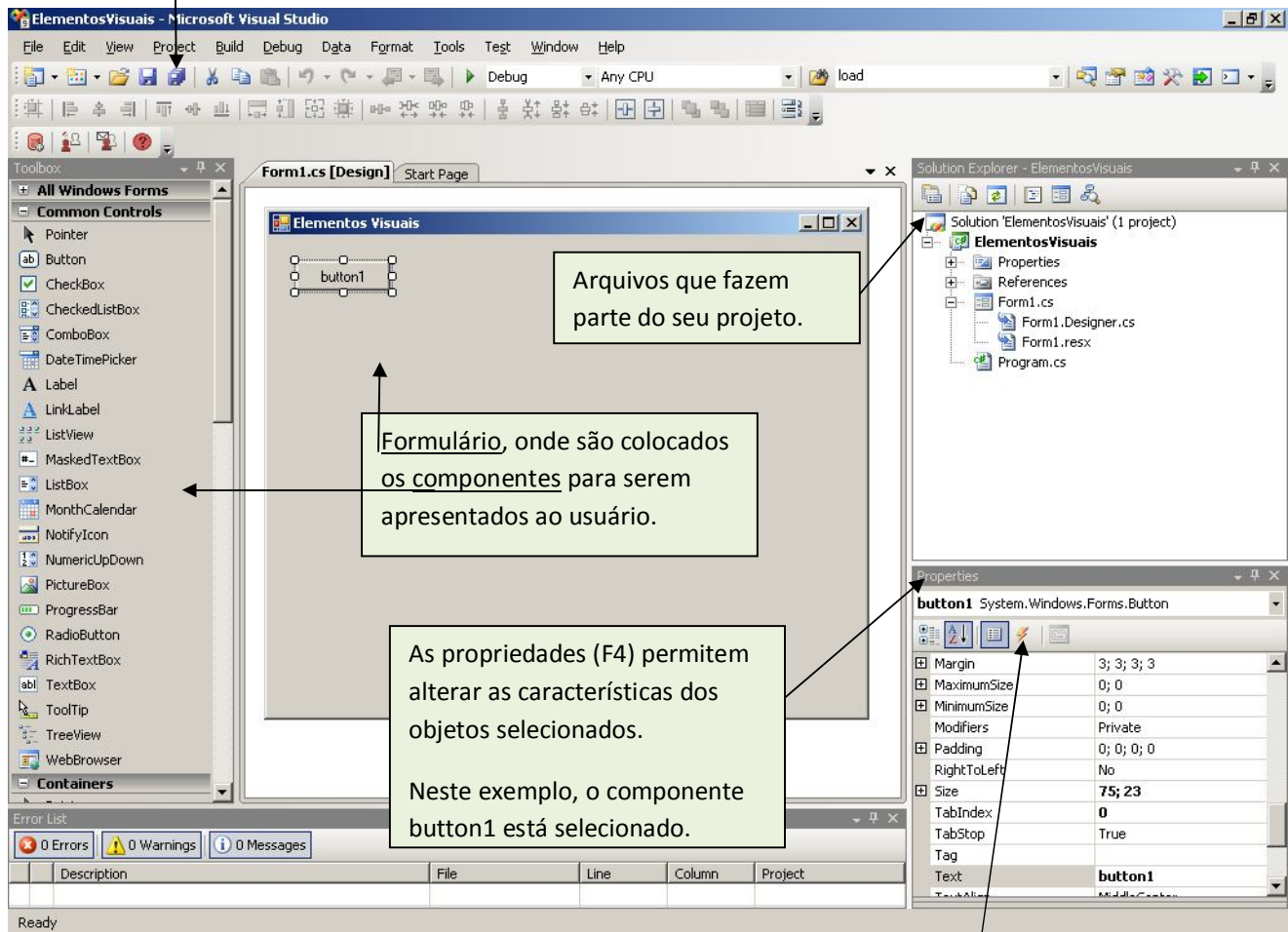
Descrição	Tecla de Atalho
Janela de propriedades do objeto	F4
Compilar	F6
Executar o programa	F5
Ir para o código fonte do formulário	F7
Retornar ao formulário	Shift + F7
Adicionar namespace	Alt + Shift + F10
Criar propriedade	CTRL + R + E
Depurar sem entrar nos métodos	F10
Depurar entrando nos métodos	F11
Adicionar Break Point	F9
Pesquisar pelo nome de um arquivo no projeto	CTRL + ,
Pesquisar no conteúdo de um arquivo	CTRL + F
Pesquisar no conteúdo de todos os arquivos do projeto	CTRL + SHIFT + F


## Criando um novo projeto Windows Forms:



Sempre que criar um novo projeto, verifique a pasta onde o mesmo será salvo. O visual Studio irá criar as pastas caso elas não existam. No exemplo acima, será criada a pasta MeuAplicativo.

Utilize sempre este botão para salvar o seu projeto. E fique atento ao local onde você salvou o projeto. Salve todos os arquivos do projeto na mesma pasta!!!!



Este botão  dá acesso aos eventos do componente selecionado. Os eventos permitem que métodos sejam executados sempre que eles ocorrem. Exemplo: O evento **click** do botão irá disparar um método sempre que o botão for pressionado. Para programar um evento, basta dar um duplo clique no espaço à direita do evento.

## Projeto Olá Mundo

Crie um novo projeto chamado OlaMundo. Adicione 1 botão.

Dê um duplo clique sobre o butto1, ou selecione o button1 e na janela de eventos selecione o evento click e dê um duplo clique sobre o local indicado na figura anterior.

Deverá aparecer uma janela com o seguinte código:

Modificador  
de acesso

Tipo de retorno  
do método

Nome do método é formado pelo nome  
do componente +nome do Evento

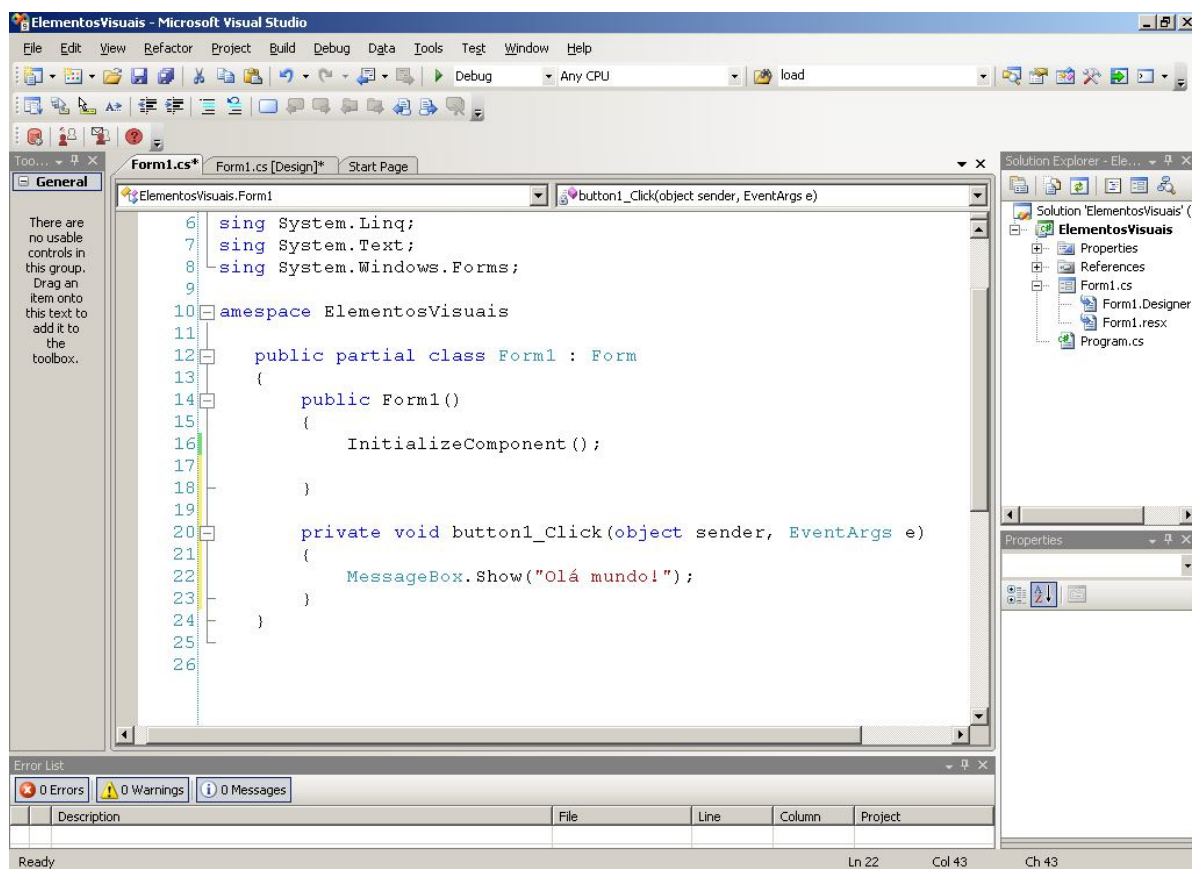
Parâmetros do método. **Sender**  
sempre será uma referência ao  
objeto que disparou o evento!

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

```

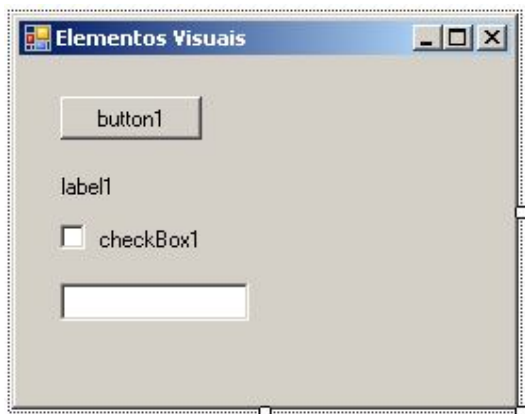
Digite o código a seguir, como na janela abaixo: Pressione F6 e teste o programa.

```
MessageBox.Show("Olá mundo!");
```



## Alterando propriedades dos objetos (componentes) via código

Crie um novo projeto e desenhe uma tela como a da imagem abaixo:



Programa no evento click do button1:

```
private void button1_Click(object sender, EventArgs e)
{
    checkBox1.Checked = true;

    label1.Text = "Eu sou o label 1";
    label1.Font = new Font("Arial", 15);
    label1.ForeColor = Color.Blue;
    label1.BackColor = Color.Yellow;

    textBox1.Text = "Pode digitar aqui";
    textBox1.ForeColor = Color.Green;
}
```

Execute o programa. Ao ser clicado o button1, o formulário ficará assim:

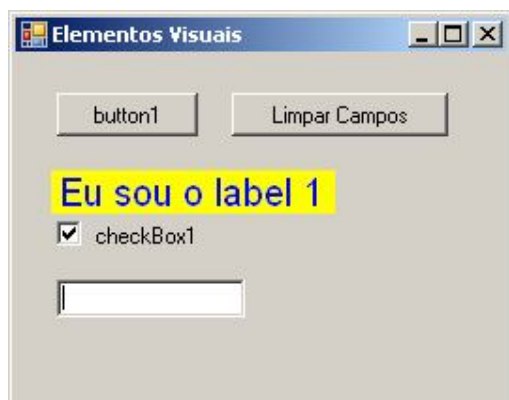


## Executando métodos dos objetos.

Adicione mais 1 botão no formulário do exemplo anterior. Altere a propriedade Text do botão para: “Limpar campos”.

No evento **click** deste novo botão, faça como abaixo:

```
private void button2_Click(object sender, EventArgs e)
{
    textBox1.Clear(); // apaga o conteúdo do objeto
    textBox1.Select(); // coloca o foco no objeto
}
```





## Nomeando os componentes

É uma boa prática que os componentes colocados no formulário que forem referenciados via código possuam um nome diferente dos nomes padrões que o Visual Studio atribui. Por exemplo, nos exemplos anteriores, os botões foram nomeados automaticamente como button1 e button2. Através da propriedade **name** é possível alterar o nome de um componente. Não pode haver 2 componentes com o mesmo nome no mesmo formulário.

Há um padrão para nomear. Por exemplo, um botão que irá efetuar um cálculo poderia ser nomeado como btnCalcular. É sempre interessante usar um prefixo antes do nome do objeto, já que isso facilita na identificação dos componentes no código.

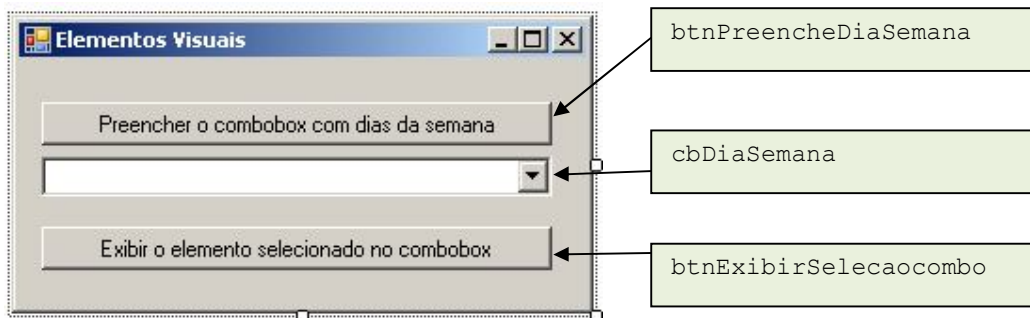
**Tabela de nomenclaturas:**

Classe	Prefixo	Exemplo
Form	frm	frmFileOpen
UserControl	usc	uscLogin
TextBox	txt	txtGetText
Label	lbl	lblTitle
ComboBox	cbo	cboCity
Image	img	imgIcon
PictureBox	pic	picHeader
Grid	grd	grdPrices
GridView	grv	grvClients
Radio Button	rbt	rbtRequerid
LinkLabel	lkl	lklSite
CheckBox	chk	chkSex
ListBox	lst	lstPrices
Button	btn	btnClientSave
TreeView	trv	trvMenu
DropDownList	Ddl	ddlVendedores
Relatórios	rpt	rptVendasMensais
Folha de estilos	css	cssSite
XML	xml	xmlArquivo
DataSet	Dts	dtsUsuarios
DataTable	dt	dtCliente

## Componente ComboBox



Desenhe uma tela como a da figura abaixo. Nomeie (propriedade name) os componentes como a seguir:



Preencher o combobox com dias da semana

```
private void btnPreencheDiaSemana_Click(object sender, EventArgs e)
{
    //apaga os dias da semana
    cbDiaSemana.Items.Clear();

    //Não permite a digitação no combo
    cbDiaSemana.DropDownStyle = ComboBoxStyle.DropDownList;

    cbDiaSemana.Items.Add("Domingo");
    cbDiaSemana.Items.Add("Segunda");
    cbDiaSemana.Items.Add("Terça");
    cbDiaSemana.Items.Add("Quarta");
    cbDiaSemana.Items.Add("Quinta");
    cbDiaSemana.Items.Add("Sexta");
    cbDiaSemana.Items.Add("Sábado");

    //Deixa a "Segunda" selecionada no combo.
    cbDiaSemana.SelectedIndex = 1;
}
```

Exibir o elemento selecionado no combobox

```
private void btnExibirSelecaocombo_Click(object sender, EventArgs e)
{
    if (cbDiaSemana.SelectedIndex == -1)
        MessageBox.Show("Nada foi selecionado!");
    else
    {
        string selecao = cbDiaSemana.Items[cbDiaSemana.SelectedIndex].ToString();
        MessageBox.Show(selecao);

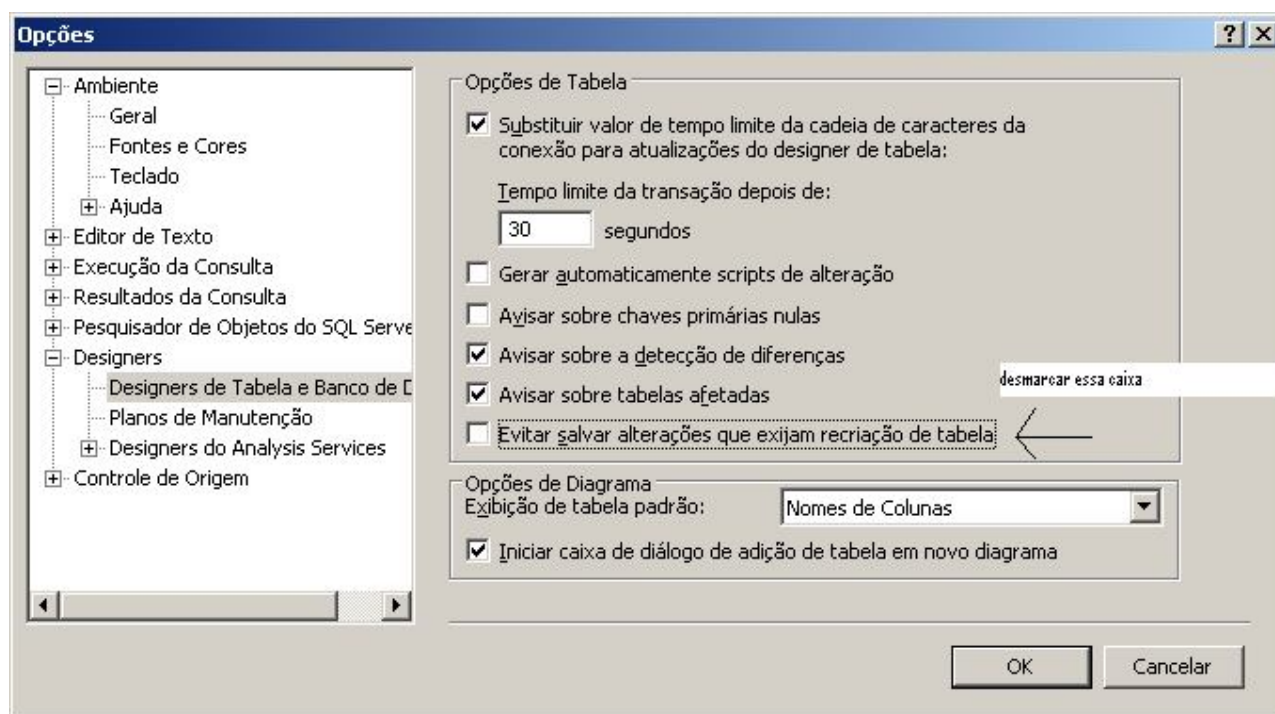
        //Se quiser pegar apenas o texto que está selecionado no
        //combo, poderia ser também assim:
        //MessageBox.Show(cbDiaSemana.Text);
    }
}
```

## Criando um cadastro de Aluno

Instrução SQL para criar a tabela aluno:

```
CREATE TABLE Alunos (  
    Id int NOT NULL PRIMARY KEY,  
    nome varchar(50) NULL,  
    mensalidade decimal (18, 2) NULL,  
    cidadeId int NULL,  
    DataNascimento datetime NULL  
)
```

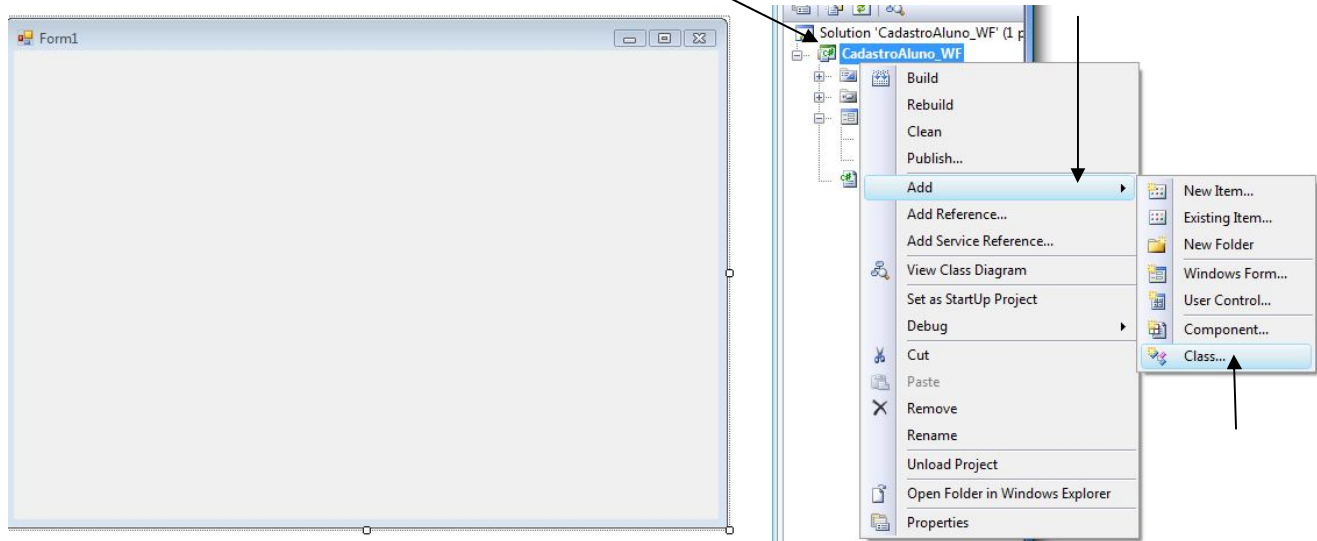
Caso tente alterar a estrutura de uma tabela no modo visual, o **SQL Server 2008** pode dar um erro dizendo que não é possível alterar a estrutura da tabela. Para permitir essa alteração, entre nas opções do SQL Server e desmarque a seguinte opção:



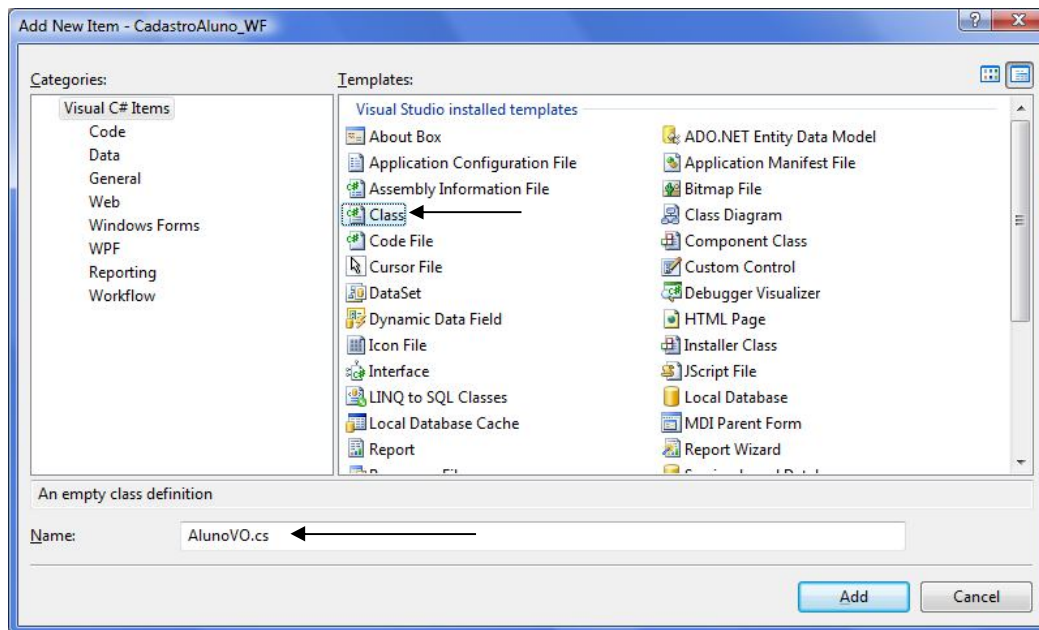
## Classe VO (Value Object)

Classe que irá conter todos os atributos da tabela Aluno. Com o objetivo de encapsular os atributos, podemos utilizar métodos de acesso públicos (Getters e Setters) para cada atributo.

Utilizando o botão direito do mouse, escolha a opção como na figura abaixo:



Nomeie a classe com “AlunoVO”



```
// Classe AlunoVO
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace CadastroAluno_WF
{
    class AlunoVO
    {
        private int id;
        private string nome;
        private double mensalidade;
        private int cidadeId;
    }
}
```

```
private DateTime dataNascimento;

// ID
public void SetId(int valor)
{
    if (valor < 0)
        throw new Exception("Id não pode ser negativo!");
    else
        id = valor;
}

public int GetId()
{
    return id;
}

//Nome
public void SetNome(string valor)
{
    if (valor.Length == 0)
        throw new Exception("Informe o nome!");
    else
        nome = valor;
}

public string GetNome()
{
    return nome;
}

//Mensalidade
public void SetMensalidade(double valor)
{
    if (valor < 0)
        throw new Exception("Mensalidade não pode ser negativa!");
    else
        mensalidade = valor;
}

public double GetMensalidade()
{
    return mensalidade;
}

//CidadeId
public void SetCidadeId(int valor)
{
    if (valor < 0)
        throw new Exception("Código da cidade não ser negativo!");
    else
        cidadeId = valor;
}

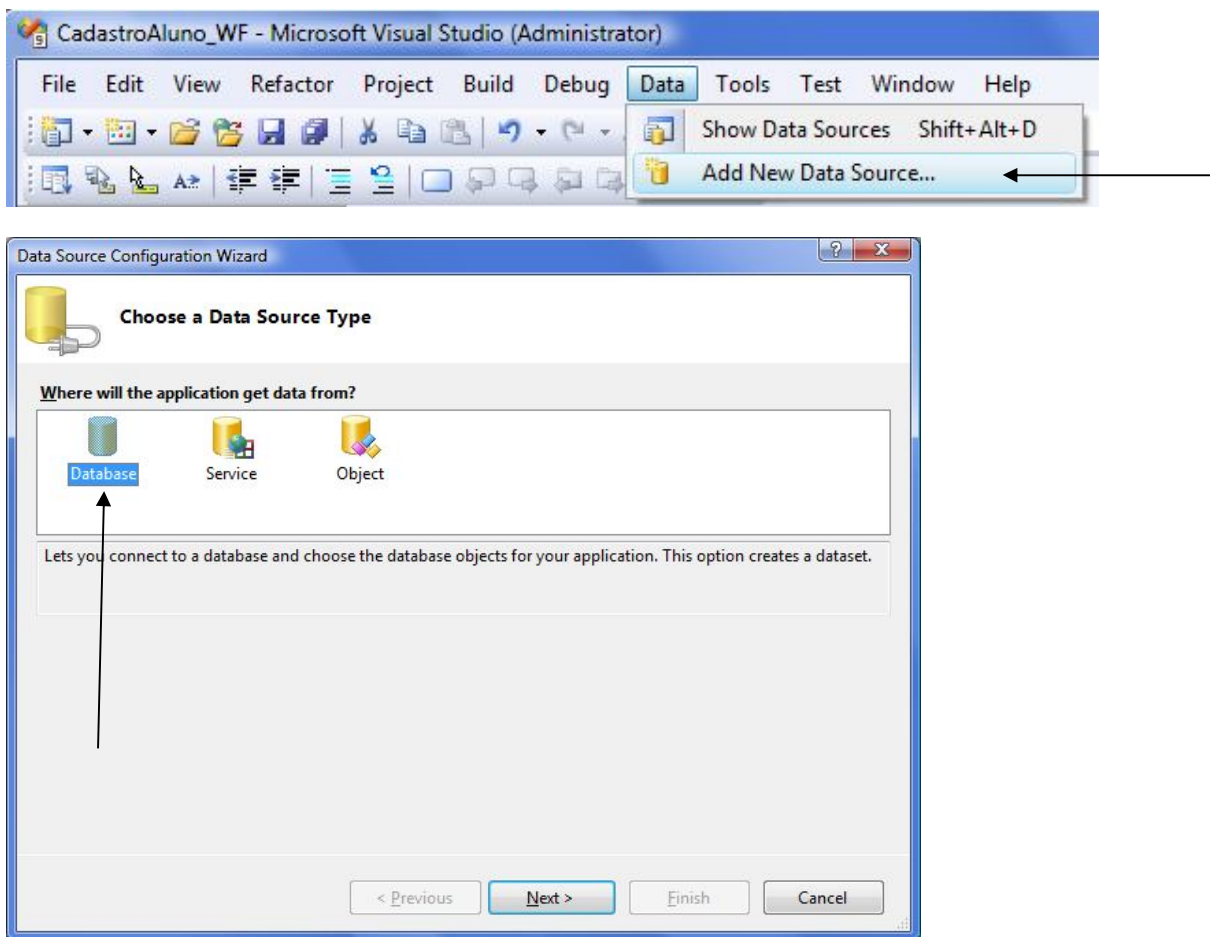
public int GetCidadeId()
{
    return cidadeId;
}

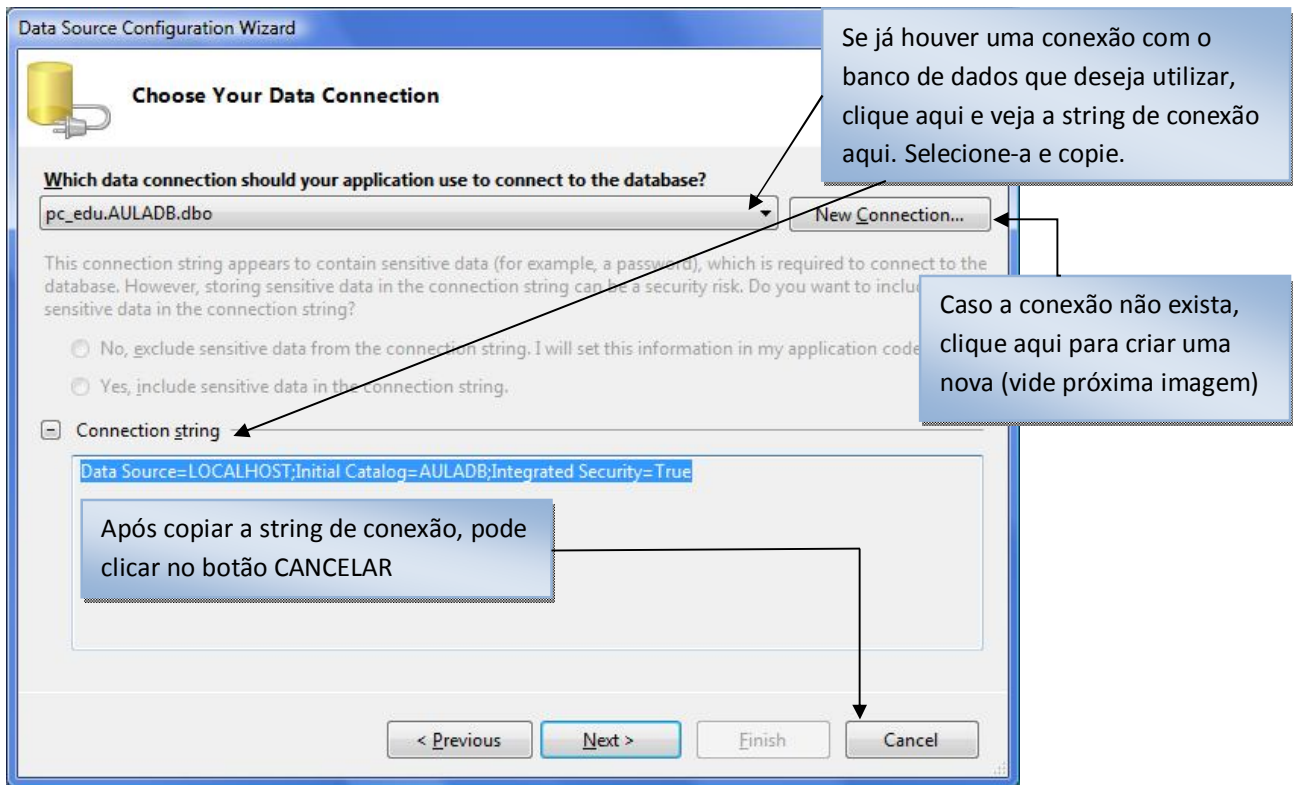
//DataNascimento
public void SetDataNascimento(DateTime valor)
{
    if (valor > DateTime.Now)
        throw new Exception("Mas a pessoa nem nasceu ainda!");
    else
        dataNascimento = valor;
}
```

```
public DateTime GetDataNascimento()  
{  
    return dataNascimento;  
}  
}
```

## Classe de Conexão com o Banco de dados

A aplicação deve ter apenas uma classe de conexão com banco de dados. Sempre que for necessário utilizar uma conexão, devemos acessar essa classe e executar o método que devolve a conexão aberta. Para se conectar ao banco de dados é necessária uma string de conexão. Para criar uma string de conexão com o seu banco de dados, faça o seguinte:





Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source: Microsoft SQL Server (SqlClient) Change...

Server name: LOCALHOST Refresh

Log on to the server

☒ Use Windows Authentication

☐ Use SQL Server Authentication

User name:

Password:

☐ Save my password

Connect to a database

☒ Select or enter a database name: AULADB

☐ Attach a database file:  Browse...

Test Connection OK Cancel Advanced...

Selecione o Banco de Dados SQL SERVER

Digite aqui o nome do seu servidor de banco de dados\nome da instância.

Se tiver instalado no seu PC, tente digitar LOCALHOST

Escolha a forma de conexão com o banco de dados

Se as informações fornecidas anteriormente estiverem corretas, os bancos de dados disponíveis irão aparecer nesta lista.

Selecione o banco de dados e clique em OK

Clique em OK e, na próxima tela, selecione e copie a string de conexão.

Você pode testar a conexão clicando aqui:

Crie uma nova classe, e nomeie-a com "ConexaoBD".

```
//Classe que manipula a conexão com o banco de dados SQL SERVER
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data.SqlClient;

namespace CadastroAluno_WF
{
    class ConexaoBD
    {
        /// <summary>
        /// Método Estático que retorna um conexao aberta com o BD
        /// </summary>
        /// <returns>Conexão aberta</returns>
        public static SqlConnection GetConexao()
        {
            string strCon = "Data Source=LOCALHOST;Initial Catalog=AULADB;Integrated Security=True";
            SqlConnection conexao = new SqlConnection(strCon);
            conexao.Open();
            return conexao;
        }
    }
}
```



## Classe DAO para realização das operações de acesso ao banco de dados

DAO (acrônimo de Data Access Object), é um padrão para persistência de dados que permite separar regras de negócio das regras de acesso a banco de dados. O padrão DAO consiste em abstrair o mecanismo de persistência utilizado na aplicação. A camada de negócios acessa os dados persistidos sem ter conhecimento se os dados estão em um banco de dados relacional ou um arquivo XML. O padrão DAO esconde os detalhes da execução da origem dos dados.

Por enquanto iremos incluir apenas o método para incluir alunos:

```
//Classe DAO para execução de instruções SQL no banco de dados
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data.SqlClient;

namespace CadastroAluno_WF
{
    class AlunoDAO
    {
        /// <summary>
        /// Método estático para inserir um aluno no BD
        /// </summary>
        /// <param name="aluno">objeto aluno com todas os atributos preenchidos</param>
        public static void Inserir(AlunoVO aluno)
        {
            SqlConnection conexao = ConexaoBD.GetConexao();
            try
            {
                //devemos substituir a ',' por '.'
                string mensalidade = aluno.GetMensalidade().ToString().Replace(',', '.');

                // set dateformat dmy; este comando serve para alterar a
                //forma como o SQL Server entende o formato de data
                string sql = String.Format("set dateformat dmy; " +
                    "insert into alunos(id, nome, mensalidade, cidadeId, dataNascimento)" +
                    "values ( {0}, '{1}', {2}, {3}, '{4}')" , aluno.GetId(),
                    aluno.GetNome(), mensalidade, aluno.GetCidadeId(),
                    aluno.GetDataNascimento());

                SqlCommand comando = new SqlCommand(sql, conexao);
                comando.ExecuteNonQuery();
            }
            finally
            {
                conexao.Close();
            }
        }

        /// <summary>
        /// Método estático para alterar um aluno no BD
        /// </summary>
        /// <param name="aluno">objeto aluno com todas os atributos preenchidos</param>
        public static void Alterar(AlunoVO aluno)
        {
            SqlConnection conexao = ConexaoBD.GetConexao();
            try
            {
                //devemos substituir a ',' por '.'
                string mensalidade = aluno.GetMensalidade().ToString().Replace(',', '.');

                // set dateformat dmy; este comando serve para alterar a
```

```

        //forma como o SQL Server entende o formato de data
        string sql = String.Format("set dateformat dmy; " +
            "update alunos set nome = '{0}', mensalidade = {1}, cidadeId = {2}, "+
            "dataNascimento = '{3}' Where id = {4}",
            aluno.GetNome(), mensalidade,
            aluno.GetCidadeId(), aluno.GetDataNascimento(),aluno.GetId() );

        SqlCommand comando = new SqlCommand(sql, conexao);
        comando.ExecuteNonQuery();
    }
    finally
    {
        conexao.Close();
    }
}

/// <summary>
/// Método para excluir um aluno
/// </summary>
/// <param name="alunoId">Id do aluno</param>
public static void Excluir(int alunoId)
{
    SqlConnection conexao = ConexaoBD.GetConexao();
    try
    {
        string sql = "delete alunos where id = " + alunoId.ToString();
        SqlCommand comando = new SqlCommand(sql, conexao);
        comando.ExecuteNonQuery();
    }
    finally
    {
        conexao.Close();
    }
}
}
}

```

## Formulário de Cadastro

Crie um formulário com a seguinte aparência:

Propriedade Mask = 99/99/9999

**Dados do aluno**

ID  Nome

Mensalidade  Cidade  Nascimento

Cidade

ComboBox Tasks

☐ Use data bound items

Unbound Mode

[Edit Items](#)

Clique no ComboBox da cidade e preencha a propriedade **itens** com algumas cidades

Para facilitar, vamos criar uma classe chamada Metodos. Nelas, iremos adicionar vários métodos estáticos para efetuar validações, enviar mensagens, etc. Como esses métodos são utilizados por vários formulários, iremos evitar redundância no código.

```
//classe Metodos
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace CadastroAluno_WF
{
    class Metodos
    {
        /// <summary>
        /// método que testa se um determinado valor string contém um inteiro válido
        /// </summary>
        /// <param name="valor">valor string a ser testado</param>
        /// <returns>true se for inteiro ou false caso contrário </returns>
        public static bool ValidaInt(string valor)
        {
            try
            {
                Convert.ToInt32(valor);
                return true;
            }
            catch
            {
                return false;
            }
        }

        /// <summary>
        /// método que testa se um determinado valor string contém um double válido
        /// </summary>
        /// <param name="valor">valor string a ser testado</param>
        /// <returns>true se for double ou false caso contrário </returns>
    }
}
```

```

public static bool ValidaDouble(string valor)
{
    try
    {
        Convert.ToDouble(valor);
        return true;
    }
    catch
    {
        return false;
    }
}

/// <summary>
/// Verifica se uma data passada via parâmetro é valida
/// </summary>
/// <param name="data">data no formato string</param>
/// <returns>True se for válida ou false caso contrário</returns>
public static bool ValidaData(string data)
{
    try
    {
        Convert.ToDateTime(data);
        return true;
    }
    catch
    {
        return false;
    }
}

public enum TipoMensagem {alerta, erro, informacao, pergunta}

/// <summary>
/// Exibe uma mensagem
/// </summary>
/// <param name="mensagem">Texto da mensagem</param>
/// <param name="tipoDaMensagem">tipo da mensagem</param>
/// <returns>Quando for mensagem de confirmação, retorna true se o
/// usuário clicar em sim e retorna False caso clique em não</returns>
public static bool Mensagem(string mensagem, TipoMensagem tipoDaMensagem)
{
    if (tipoDaMensagem == TipoMensagem.alerta)
    {
        MessageBox.Show(mensagem, "Atenção", MessageBoxButtons.OK,
            MessageBoxIcon.Exclamation);
        return true;
    }
    else if (tipoDaMensagem == TipoMensagem.erro)
    {
        MessageBox.Show(mensagem, "Erro", MessageBoxButtons.OK,
            MessageBoxIcon.Error);
        return true;
    }
    else if (tipoDaMensagem == TipoMensagem.informacao)
    {
        MessageBox.Show(mensagem, "Informação", MessageBoxButtons.OK,
            MessageBoxIcon.Information);
        return true;
    }
    else
    {
        if (MessageBox.Show(mensagem, "Atenção", MessageBoxButtons.YesNo,
            MessageBoxIcon.Question) == DialogResult.Yes)
            return true;
        else
            return false;
    }
}
}

```

No formulário, iremos criar 2 métodos auxiliares:

```

/// <summary>
/// Efetua a limpeza dos campos
/// </summary>
private void LimpaCampos()
{
    txtData.Clear();
    txtId.Clear();
    txtMensalidade.Clear();
    txtNome.Clear();
    cbCidade.SelectedIndex = -1;
}

/// <summary>
/// Efetua algumas validações básicas. Outras validações são
/// realizadas pela classe AlunoVO
/// </summary>
/// <returns>True se tudo estiver OK</returns>
private bool Validacoes()
{
    if (Metodos.ValidaInt(txtId.Text) == false)
    {
        Metodos.Mensagem("Digite apenas números no campo ID.",
            Metodos.TipoMensagem.erro);
        return false;
    }

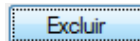
    if (Metodos.ValidaDouble(txtMensalidade.Text) == false)
    {
        Metodos.Mensagem("Digite apenas números no campo Mensalidade.",
            Metodos.TipoMensagem.erro);
        return false;
    }

    if (Metodos.ValidaData(txtData.Text) == false)
    {
        Metodos.Mensagem("Data de nascimento inválida.",
            Metodos.TipoMensagem.erro);
        return false;
    }

    return true;
}

```

Insira o seguinte código no evento click do botão

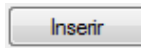


```

private void btnExcluir_Click(object sender, EventArgs e)
{
    if (Metodos.ValidaInt(txtId.Text) == false)
    {
        Metodos.Mensagem("Digite apenas números no campo ID.",
            Metodos.TipoMensagem.erro);
    }
    else
    {
        if (Metodos.Mensagem("Confirma a exclusão?",
            Metodos.TipoMensagem.pergunta) == true)
        {
            AlunoDAO.Excluir(Convert.ToInt32(txtId.Text));
            LimpaCampos();
            Metodos.Mensagem("Registro excluído!", Metodos.TipoMensagem.informacao);
        }
    }
}

```

Insira o seguinte código no evento click do botão

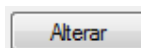


```
private void btnInserir_Click(object sender, EventArgs e)
{
    if (Validacoes() == false)
        return;

    try
    {
        AlunoVO aluno = new AlunoVO();
        aluno.SetId(Convert.ToInt32(txtId.Text));
        aluno.SetCidadeId(cbCidade.SelectedIndex);
        aluno.SetMensalidade(Convert.ToDouble(txtMensalidade.Text));
        aluno.SetNome(txtNome.Text);
        aluno.SetDataNascimento(Convert.ToDateTime(txtData.Text));

        AlunoDAO.Inserir(aluno);
        LimpaCampos();
        Metodos.Mensagem("Dados gravados!", Metodos.TipoMensagem.informacao);
    }
    catch (Exception erro)
    {
        Metodos.Mensagem(erro.Message, Metodos.TipoMensagem.erro);
    }
}
```

Insira o seguinte código no evento click do botão



```
private void btnAlterar_Click(object sender, EventArgs e)
{
    if (Validacoes() == false)
        return;

    try
    {
        AlunoVO aluno = new AlunoVO();
        aluno.SetId(Convert.ToInt32(txtId.Text));
        aluno.SetCidadeId(cbCidade.SelectedIndex);
        aluno.SetMensalidade(Convert.ToDouble(txtMensalidade.Text));
        aluno.SetNome(txtNome.Text);
        aluno.SetDataNascimento(Convert.ToDateTime(txtData.Text));

        AlunoDAO.Alterar(aluno);
        LimpaCampos();
        Metodos.Mensagem("Dados gravados!", Metodos.TipoMensagem.informacao);
    }
    catch (Exception erro)
    {
        Metodos.Mensagem(erro.Message, Metodos.TipoMensagem.erro);
    }
}
```

## Melhorando o código antes de prosseguir com o cadastro

Ante de iniciarmos a finalização do cadastro, iremos dar uma melhoria neste código, removendo a redundância através de métodos.

Observe que na classe DAO existe uma grande quantidade de código repetido. Os métodos que efetuam as instruções para incluir, alterar e excluir só possuem diferenças nas instruções SQL. Podemos criar um método para executar essas instruções, removendo grande parte da redundância. E podemos melhorar mais ainda se incluirmos este método na classe **Métodos**, pois eles serão utilizados também por outras classes DAO.

Inclua o método a seguir na classe **MÉTODOS**:

OBS: será necessário adicionar o namespace: `using System.Data.SqlClient;`

```
/// <summary>
/// Executa uma instrução SQL que não retorna nada (ex: Insert, Update, Delete)
/// </summary>
/// <param name="sql">instrução SQL</param>
public static void ExecutaSQL(string sql)
{
    SqlConnection conexao = ConexaoBD.GetConexao();
    try
    {
        SqlCommand comando = new SqlCommand(sql, conexao);
        comando.ExecuteNonQuery();
    }
    finally
    {
        conexao.Close();
    }
}
```

Agora, altere a classe `AlunoDAO` para que ela utilize este método. O código completo da classe `AlunoDAO` modificada vê a seguir:

```
//Classe DAO para execução de instruções SQL no banco de dados
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data.SqlClient;

namespace CadastroAluno_WF
{
    class AlunoDAO
    {
        /// <summary>
        /// Método estático para inserir um aluno no BD
```

```

/// </summary>
/// <param name="aluno">objeto aluno com todas os atributos preenchidos</param>
public static void Inserir(AlunoVO aluno)
{
    //devemos substituir a ',' por '.'
    string mensalidade = aluno.GetMensalidade().ToString().Replace(',', '.');

    // set dateformat dmy; este comando serve para alterar a
    //forma como o SQL Server entende o formato de data
    string sql = String.Format("set dateformat dmy; " +
        "insert into alunos(id, nome, mensalidade, cidadeId, dataNascimento)" +
        "values ( {0}, '{1}', {2}, {3}, '{4}')" , aluno.GetId(),
        aluno.GetNome(), mensalidade, aluno.GetCidadeId(),
        aluno.GetDataNascimento());
    Metodos.ExecutaSQL(sql);
}

/// <summary>
/// Método estático para alterar um aluno no BD
/// </summary>
/// <param name="aluno">objeto aluno com todas os atributos preenchidos</param>
public static void Alterar(AlunoVO aluno)
{
    //devemos substituir a ',' por '.'
    string mensalidade = aluno.GetMensalidade().ToString().Replace(',', '.');

    // set dateformat dmy; este comando serve para alterar a
    //forma como o SQL Server entende o formato de data
    string sql = String.Format("set dateformat dmy; " +
        "update alunos set nome = '{0}', mensalidade = {1}, cidadeId = {2}," +
        "dataNascimento = '{3}' Where id = {4}",
        aluno.GetNome(), mensalidade,
        aluno.GetCidadeId(), aluno.GetDataNascimento(), aluno.GetId());

    Metodos.ExecutaSQL(sql);
}

/// <summary>
/// Método para excluir um aluno
/// </summary>
/// <param name="alunoId">Id do aluno</param>
public static void Excluir(int alunoId)
{
    string sql = "delete alunos where id = " + alunoId.ToString();
    Metodos.ExecutaSQL(sql);
}
}

```



## Criando um método para efetuar consulta

Para armazenarmos o conteúdo consultado de uma tabela, iremos utilizar um objeto da classe `DataTable`. Esta classe permite criar uma tabela “virtual” (apenas em memória), onde são armazenados os registros de uma tabela consultada.

No intuito de evitar a redundância de código, iremos criar o método para executar a instrução SQL da consulta na classe **Metodos**.

Adicione o código a seguir na classe **metodos**:

Obs: será necessário adicionar o namespace `using System.Data;`

```
/// <summary>
/// Executa uma instrução Select
/// </summary>
/// <param name="sql">instrução SQL</param>
/// <returns>DataTable com os dados da instrução SQL</returns>
public static DataTable ExecutaSelect(string sql)
{
    SqlConnection conexao = ConexaoBD.GetConexao();
    try
    {
        SqlDataAdapter adapter = new SqlDataAdapter(sql, conexao);
        DataTable tabelaTemp = new DataTable();
        adapter.Fill(tabelaTemp);
        return tabelaTemp;
    }
    finally
    {
        conexao.Close();
    }
}
```

Na classe DAO, vamos criar o método consulta:

Obs: será necessário adicionar o namespace `using System.Data;`

```

/// <summary>
/// Consulta um determinado aluno no banco de dados
/// </summary>
/// <param name="alunoId">id do aluno</param>
/// <returns>Objeto AlunoVO ou NULL caso o aluno não exista</returns>
public static AlunoVO Consultar(int alunoId)
{
    string sql = "select * from alunos where id = " + alunoId.ToString();

    DataTable tabela = Metodos.ExecutaSelect(sql);

    if (tabela == null || tabela.Rows.Count == 0) // testa se o aluno existe na tabela
        return null;
    else
    {
        return MontaAlunoVO(tabela.Rows[0]);
    }
}

/// <summary>
/// Recebe uma linha de um datatable e preenche um objeto AlunoVO
/// </summary>
/// <param name="registro">registro do dataTable</param>
/// <returns>Objeto AlunoVO com os atributos preenchidos</returns>
public static AlunoVO MontaAlunoVO(DataRow registro)
{
    AlunoVO aluno = new AlunoVO();

    aluno.SetId(Convert.ToInt32(registro["id"]));
    aluno.SetNome(registro["nome"].ToString());
    aluno.SetCidadeId(Convert.ToInt32(registro["cidadeId"]));
    aluno.SetMensalidade(Convert.ToDouble(registro["mensalidade"]));
    aluno.SetDataNascimento(Convert.ToDateTime(registro["DataNascimento"]));

    return aluno;
}

```

## Métodos para navegar nos registros

Agora que temos pronto o método que consulta, vamos aproveitar para criar os métodos para efetuar a navegação nos registros da tabela.

Vamos criar os seguintes métodos:

- Primeiro = Vai para o primeiro registro da tabela
- Ultimo = Vai para o último registro da tabela
- Anterior -> Volta para o registro anterior
- Proximo – Vai para o próximo registro

```

/// <summary>
/// Vai para o primeiro registro da tabela
/// </summary>
/// <returns>objeto contendo os valores do registro consultado</returns>
public static AlunoVO Primeiro()
{
    string sql = "select top 1 * from alunos order by id";
    DataTable tabela = Metodos.ExecutaSelect(sql);
    if (tabela == null || tabela.Rows.Count == 0) // testa se o aluno existe na tabela
        return null;
    else
    {
        return MontaAlunoVO(tabela.Rows[0]);
    }
}

/// <summary>
/// Vai para o último registro da tabela
/// </summary>
/// <returns>objeto contendo os valores do registro consultado</returns>
public static AlunoVO Ultimo()
{
    string sql = "select top 1 * from alunos order by id DESC";
    DataTable tabela = Metodos.ExecutaSelect(sql);
    if (tabela == null || tabela.Rows.Count == 0) // testa se o aluno existe na tabela
        return null;
    else
    {
        return MontaAlunoVO(tabela.Rows[0]);
    }
}

/// <summary>
/// Vai pra o registro anterior ao atual
/// </summary>
/// <param name="idAtual">id do aluno atual</param>
/// <returns>objeto contendo os valores do registro consultado</returns>
public static AlunoVO Anterior(int idAtual)
{
    string sql = "select top 1 * from alunos where id < {0} order by id DESC";
    DataTable tabela = Metodos.ExecutaSelect(string.Format(sql, idAtual));
    if (tabela == null || tabela.Rows.Count == 0) // testa se o aluno existe na tabela
        return null;
    else
    {
        return MontaAlunoVO(tabela.Rows[0]);
    }
}

/// <summary>
/// Vai pra o próximo registro com base no registro atual
/// </summary>
/// <param name="idAtual">id do aluno atual</param>
/// <returns>objeto contendo os valores do registro consultado</returns>
public static AlunoVO Proximo(int idAtual)
{
    string sql = "select top 1 * from alunos where id > {0} order by id";
    DataTable tabela = Metodos.ExecutaSelect(string.Format(sql, idAtual));
    if (tabela == null || tabela.Rows.Count == 0) // testa se o aluno existe na tabela
        return null;
    else
    {
        return MontaAlunoVO(tabela.Rows[0]);
    }
}

```

Observe o código que acabamos de fazer. Quando for fazer outra classe DAO (ex: Professores.DAO), muito desse código se repetiria? O que mudaria? Daria para colocar alguma coisa na biblioteca METODOS?

## Finalizando o cadastro

O nosso objetivo agora será deixar o cadastro com a aparência da figura abaixo, onde há as opções de incluir, alterar, excluir, além dos botões de navegação. Também iremos tratar de habilitar/desabilitar controles de acordo com a opção escolhida pelo usuário.

O código completo da tela virá a seguir:

```
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 CadastroAluno_WF
{
    public partial class frCadAluno : Form
    {
        enum modoOperacao { alteracao, inclusao, consulta }

        public frCadAluno()
        {
            InitializeComponent();
        }
        /// <summary>
        /// Efetua a limpeza dos campos
        /// </summary>
        private void LimpaCampos()
        {
            txtData.Clear();
            txtId.Clear();
            txtMensalidade.Clear();
            txtNome.Clear();
            cbCidade.SelectedIndex = -1;
        }

        /// <summary>
        /// Efetua algumas validações básicas. Outras validações são
        /// realizadas pela classe AlunoVO
        /// </summary>
        /// <returns>True se tudo estiver OK</returns>
        private bool Validacoes()
        {
            if (Metodos.ValidaInt(txtId.Text) == false)
            {
                Metodos.Mensagem("Digite apenas números no campo ID.",
                                   Metodos.TipoMensagem.erro);
                return false;
            }
        }
    }
}
```

```

        if (Metodos.ValidaDouble(txtMensalidade.Text) == false)
        {
            Metodos.Mensagem("Digite apenas números no campo Mensalidade.",
                              Metodos.TipoMensagem.erro);
            return false;
        }

        if (Metodos.ValidaData(txtData.Text) == false)
        {
            Metodos.Mensagem("Data de nascimento inválida.",
                              Metodos.TipoMensagem.erro);
            return false;
        }

        return true;
    }

    /// <summary>
    /// Preenche os dados de um AlunoVO com base nos campos da tela
    /// </summary>
    /// <returns>objeto AlunoVO</returns>
    private AlunoVO PreencheDadosVO()
    {
        AlunoVO aluno = new AlunoVO();
        aluno.SetId(Convert.ToInt32(txtId.Text));
        aluno.SetCidadeId(cbCidade.SelectedIndex);
        aluno.SetMensalidade(Convert.ToDouble(txtMensalidade.Text));
        aluno.SetNome(txtNome.Text);
        aluno.SetDataNascimento(Convert.ToDateTime(txtData.Text));
        return aluno;
    }

    /// <summary>
    /// Preenche os campos da tela com base nos dados de um AlunoVO
    /// </summary>
    /// <param name="aluno">objeto AlunoVO</param>
    private void PreencheCamposTela(AlunoVO aluno)
    {
        LimpaCampos();
        if (aluno != null)
        {
            txtId.Text = aluno.GetId().ToString();
            txtNome.Text = aluno.GetNome();
            txtMensalidade.Text = aluno.GetMensalidade().ToString("0.00");
            txtData.Text = aluno.GetDataNascimento().ToString("dd/MM/yyyy");
            cbCidade.SelectedIndex = aluno.GetCidadeId();
        }
    }

    /// <summary>
    /// Altera a tela para se adequar a um determinado modo de operação
    /// que pode ser inclusão, alteração ou consulta
    /// </summary>
    /// <param name="modo">modo para o qual se deseja alterar</param>
    private void AlteraParaModo(ModoOperacao modo)
    {
        txtNome.Enabled = (modo != ModoOperacao.consulta);
        txtMensalidade.Enabled = (modo != ModoOperacao.consulta);
        txtData.Enabled = (modo != ModoOperacao.consulta);
        cbCidade.Enabled = (modo != ModoOperacao.consulta);

        btnInserir.Enabled = (modo == ModoOperacao.consulta);
        btnAlterar.Enabled = (modo == ModoOperacao.consulta);
        btnExcluir.Enabled = (modo == ModoOperacao.consulta);
        btnPrimeiro.Enabled = (modo == ModoOperacao.consulta);
        btnAnterior.Enabled = (modo == ModoOperacao.consulta);
        btnProximo.Enabled = (modo == ModoOperacao.consulta);
    }

```

```

        btnUltimo.Enabled = (modo == modoOperacao.consulta);

        btnGravar.Enabled = (modo != modoOperacao.consulta);
        btnCancelar.Enabled = (modo != modoOperacao.consulta);

        if (modo == modoOperacao.inclusao)
        {
            txtId.Enabled = true;
            LimpaCampos();
        }
        else
            txtId.Enabled = false;
    }

    private void btnInserir_Click(object sender, EventArgs e)
    {
        AlteraParaModo(modoOperacao.inclusao);
    }

    private void btnAlterar_Click(object sender, EventArgs e)
    {
        AlteraParaModo(modoOperacao.alteracao);
    }

    private void btnExcluir_Click(object sender, EventArgs e)
    {
        if (Metodos.ValidaInt(txtId.Text) == false)
        {
            Metodos.Mensagem("Digite apenas números no campo ID.",
                               Metodos.TipoMensagem.erro);
        }
        else
        {
            if (Metodos.Mensagem("Confirma a exclusão?",
                                   Metodos.TipoMensagem.pergunta) == true)
            {
                int id = Convert.ToInt32(txtId.Text);
                AlunoDAO.Excluir(id);
                Metodos.Mensagem("Registro excluído!", Metodos.TipoMensagem.informacao);

                AlunoVO aluno = AlunoDAO.Primeiro(); // após apagar, posiciona-se no
                                                       //primeiro registro

                PreencheCamposTela(aluno);
            }
        }
    }

    private void frCadAluno_Load(object sender, EventArgs e)
    {
        AlteraParaModo(modoOperacao.consulta);
        AlunoVO aluno = AlunoDAO.Primeiro();
        PreencheCamposTela(aluno);
    }

    private void btnGravar_Click(object sender, EventArgs e)
    {
        if (Validacoes() == false)
            return;

        try
        {
            if (txtId.Enabled)
                AlunoDAO.Inserir(PreencheDadosVO());
            else
                AlunoDAO.Alterar(PreencheDadosVO());

            AlteraParaModo(modoOperacao.consulta);
        }
    }

```

```

        Metodos.Mensagem("Dados gravados!", Metodos.TipoMensagem.informacao);
    }
    catch (Exception erro)
    {
        Metodos.Mensagem(erro.Message, Metodos.TipoMensagem.erro);
    }
}

private void btnPrimeiro_Click(object sender, EventArgs e)
{
    AlunoVO aluno = AlunoDAO.Primeiro();
    if (aluno != null)
        PreencheCamposTela(aluno);
}

private void btnAnterior_Click(object sender, EventArgs e)
{
    if (Metodos.ValidaInt(txtId.Text)) // tem que ver se o cadastro não está vazio...
    {
        AlunoVO aluno = AlunoDAO.Anterior(Convert.ToInt32(txtId.Text));
        if (aluno != null)
            PreencheCamposTela(aluno);
    }
}

private void btnProximo_Click(object sender, EventArgs e)
{
    if (Metodos.ValidaInt(txtId.Text)) // tem que ver se o cadastro não está vazio...
    {
        AlunoVO aluno = AlunoDAO.Proximo(Convert.ToInt32(txtId.Text));
        if (aluno != null)
            PreencheCamposTela(aluno);
    }
}

private void btnUltimo_Click(object sender, EventArgs e)
{
    AlunoVO aluno = AlunoDAO.Ultimo();
    if (aluno != null)
        PreencheCamposTela(aluno);
}

private void btnCancelar_Click(object sender, EventArgs e)
{
    if (txtId.Enabled == true)
        PreencheCamposTela(AlunoDAO.Primeiro());

    AlteraParaModo(modosOperacao.consulta);
}
}

```

## Listando valores de uma tabela em um ComboBox

Para exemplificar, vamos utilizar o cadastro criado no tópico anterior, listando as cidades no combobox. Primeiro, vamos criar uma tabela para as cidades:

```
CREATE TABLE cidades (
    id int NOT NULL primary key,
    nome varchar(30) NULL,
);
```

Insira os seguintes registros:

```
insert into cidades (id,nome) values (1, 'São Bernardo')
insert into cidades (id,nome) values (2, 'Santo André')
insert into cidades (id,nome) values (3, 'Maua')
insert into cidades (id,nome) values (4, 'Diadema')
insert into cidades (id,nome) values (5, 'Ribeirão Pires')
```

Alguns objetos, como o ComboBox, possuem a propriedade DataSource. Esta propriedade permite que seja associado uma lista ou um DataTable e, no caso do ComboBox, os valores serão listados.

A seguir, veja as modificações que foram realizadas no programa:

Criar a classe CidadeDAO:

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

namespace CadastroAluno_WF
{
    class CidadeDAO
    {
        /// <summary>
        /// Retorna um DataTable com todas as cidades por ordem alfabética
        /// </summary>
        /// <returns>DataTable com todas as cidades por ordem alfabética</returns>
        public static DataTable ListaCidades()
        {
            string sql = "select * from cidades order by id";
            DataTable tabela = Metodos.ExecutaSelect(sql);
            return tabela;
        }
    }
}
```



Alterar o código fonte do formulário de cadastros de alunos. Alterações:

```
private void frCadAluno_Load(object sender, EventArgs e)
{
    cbCidade.DataSource = CidadeDAO.ListaCidades(); // lista as cidades que estão
                                                    // cadastradas na tabela de cidades
    cbCidade.DisplayMember = "nome"; // campo que será listado no combobox (usuário pode ver)
    cbCidade.ValueMember = "id"; // campo que irá representar o valor que foi escolhido

    AlteraParaModo(modoperacao.consulta);
    AlunoVO aluno = AlunoDAO.Primeiro();
    PreencheCamposTela(aluno);
}

/// <summary>
/// Preenche os dados de um AlunoVO com base nos campos da tela
/// </summary>
/// <returns>objeto AlunoVO</returns>
private AlunoVO PreencheDadosVO()
{
    AlunoVO aluno = new AlunoVO();
    aluno.SetId(Convert.ToInt32(txtId.Text));
    aluno.SetCidadeId(Convert.ToInt32(cbCidade.SelectedValue));
    aluno.SetMensalidade(Convert.ToDouble(txtMensalidade.Text));
    aluno.SetNome(txtNome.Text);
    aluno.SetDataNascimento(Convert.ToDateTime(txtData.Text));
    return aluno;
}

/// <summary>
/// Preenche os campos da tela com base nos dados de um AlunoVO
/// </summary>
/// <param name="aluno">objeto AlunoVO</param>
private void PreencheCamposTela(AlunoVO aluno)
{
    LimpaCampos();
    if (aluno != null)
    {
        txtId.Text = aluno.GetId().ToString();
        txtNome.Text = aluno.GetNome();
        txtMensalidade.Text = aluno.GetMensalidade().ToString("0.00");
        txtData.Text = aluno.GetDataNascimento().ToString("dd/MM/yyyy");
        cbCidade.SelectedValue = aluno.GetCidadeId();
    }
}
```

## Evitando erro de chave duplicada

O ideal é que o próprio sistema indique o próximo ID a ser utilizado, porém, nem sempre isso é possível (o usuário pode querer informar ele mesmo o valor do campo ID). Para evitar erro de chave duplicada nestes casos, podemos verificar (antes de gravar) se o ID já não existe e, caso exista, avisamos o usuário para trocá-lo.

A modificação é simples: No método `Validacoes()`, inclua a linha em amarelo logo após validar o campo ID:

```
/// <summary>
/// Efetua algumas validações básicas. Outras validações são
/// realizadas pela classe AlunoVO
/// </summary>
/// <returns>True se tudo estiver OK</returns>
private bool Validacoes()
{
    if (Metodos.ValidaInt(txtId.Text) == false)
    {
        Metodos.Mensagem("Digite apenas números no campo ID.",
                          Metodos.TipoMensagem.erro);
        return false;
    }

    if (txtId.Enabled) // significa que é inclusão...
    {
        if (AlunoDAO.Consultar(Convert.ToInt32(txtId.Text)) != null)
        {
            Metodos.Mensagem("Este código já está atribuído a outro aluno!",
                              Metodos.TipoMensagem.erro);
            return false;
        }
    }
}
```

## Sugerindo um código de aluno automaticamente

Como dito no tópico anterior, o próprio sistema indicar o próximo ID de aluno disponível durante uma inclusão.

Vamos alterar o programa para que, assim que o usuário clicar no botão incluir, o sistema automaticamente preencha o campo ID com o próximo valor disponível.

Inclua o seguinte método na classe AlunoDAO:

```
/// <summary>
/// Retorna o próximo ID do aluno disponível
/// </summary>
/// <returns>Próximo ID disponível do aluno</returns>
public static int ProximoID()
{
    string sql = "select isnull(max(id) +1, 1) as 'MAIOR' from alunos";
    DataTable tabela = Metodos.ExecutaSelect(sql);
    return Convert.ToInt32(tabela.Rows[0] ["MAIOR"]);
}
```

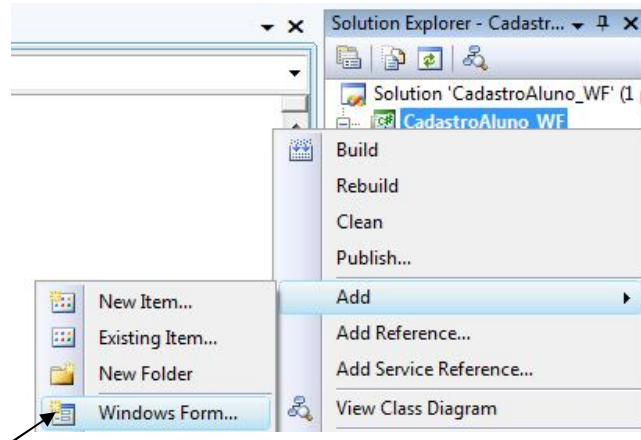
No botão de inserir, coloque as linhas abaixo em amarelo:

```
private void btnInserir_Click(object sender, EventArgs e)
{
    AlteraParaModo(modosOperacao.inclusao);
    txtId.Text = AlunoDAO.ProximoID().ToString();
    txtId.Focus();
}
```

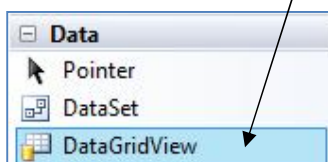
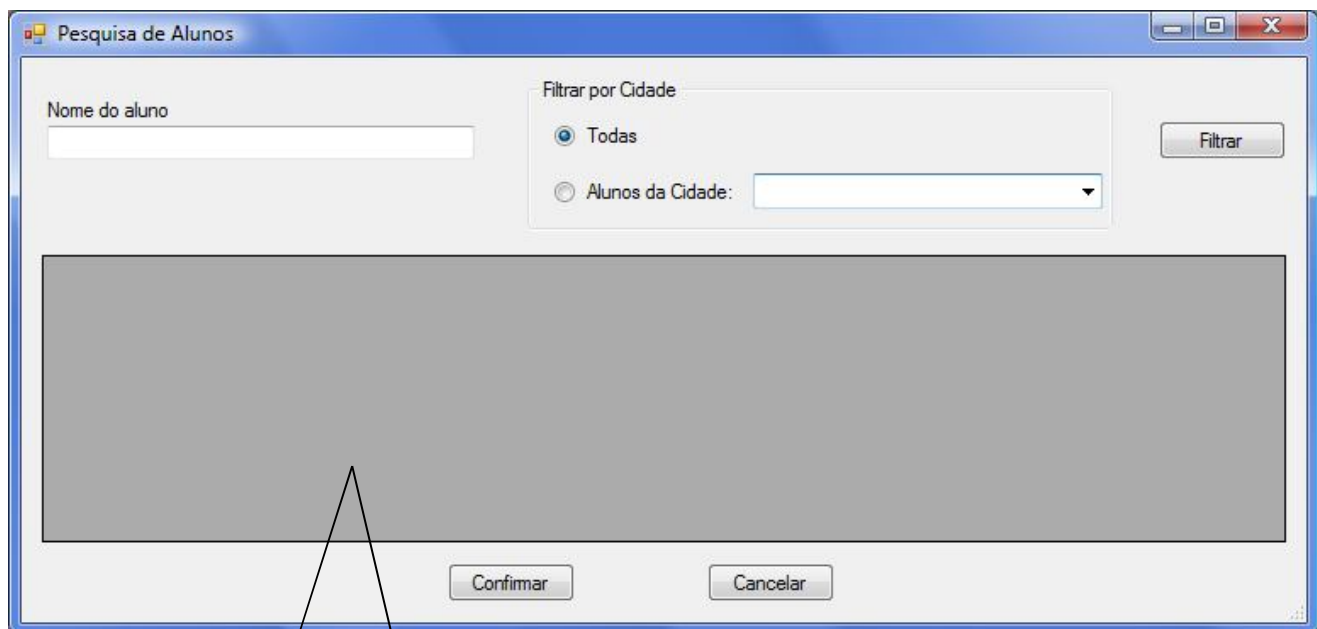
## Criando uma tela de Consulta

Vamos criar uma tela para efetuar a pesquisa de alunos. Serão permitidos 2 filtros. Pelo nome do aluno (completo ou parcial) e por cidade (todas as cidades ou uma selecionada). O usuário poderá selecionar um aluno dentre aqueles listados no Grid e este aluno será localizado e posicionado na tela de cadastro de alunos.

Adicione um novo formulário ao seu projeto: (frPesquisaAluno)



Configure o formulário para que ele fique com a seguinte aparência:



Altere as seguintes propriedades do componente DataGridView:

```
AllowUserToAddRows = False
AllowUserToDeleteRows = False
AllowUserToResizeRows = False
MultiSelect = False
ReadOnly = true
SelectionMode = FullRowSelect
```

Código fonte completo do formulário de pesquisa:

```
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 CadastroAluno_WF
{
    public partial class frPesquisaAluno : Form
    {
        public int idAlunoSelecionado = 0;

        public frPesquisaAluno()
        {
            InitializeComponent();

            cbCidades.DataSource = CidadeDAO.ListaCidades();
            cbCidades.DisplayMember = "nome";
            cbCidades.ValueMember = "id";
        }

        private void btnFiltrar_Click(object sender, EventArgs e)
        {
            int cidadeSelecionada;

            if (rbTodas.Checked)
                cidadeSelecionada = 0;
            else
                cidadeSelecionada = Convert.ToInt32(cbCidades.SelectedValue);

            DataTable tabela = AlunoDAO.ListaAlunos(txtNomeAluno.Text, cidadeSelecionada);

            dataGridView1.DataSource = tabela;
        }

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

        private void btnConfirmar_Click(object sender, EventArgs e)
        {
            if (dataGridView1.SelectedRows.Count == 0)
            {
                Metodos.Mensagem("Não há aluno selecionado!", Metodos.TipoMensagem.erro);
                return;
            }
            else
            {
                // dataGridView1.SelectedRows = coleção das linhas selecionadas no grid
                // Cells = uma linha é formada por 'células', que são as colunas da tabela.
                // Na nossa instrução SQL que efetua a consulta, a primeira coluna é o
                // id do aluno. Por isso fizemos Cells[0].Value
                idAlunoSelecionado =
                    Convert.ToInt32(dataGridView1.SelectedRows[0].Cells[0].Value);

                Close();
            }
        }
    }
}
```

Na classe `AlunoDAO`, insira o seguinte método:

```

/// <summary>
/// Retorna uma datatable com os alunos que satisfaçam o filtro de pesquisa
/// </summary>
/// <param name="nome">nome do aluno, total ou parcial</param>
/// <param name="cidadeId">codigo da cidade ou Zero para todas</param>
/// <returns>datatable com os alunos que satisfaçam o filtro de pesquisa</returns>
public static DataTable ListaAlunos(string nome, int cidadeId)
{
    string where = "";

    where = "Where alunos.nome like '%" + nome.Trim() + "%' ";

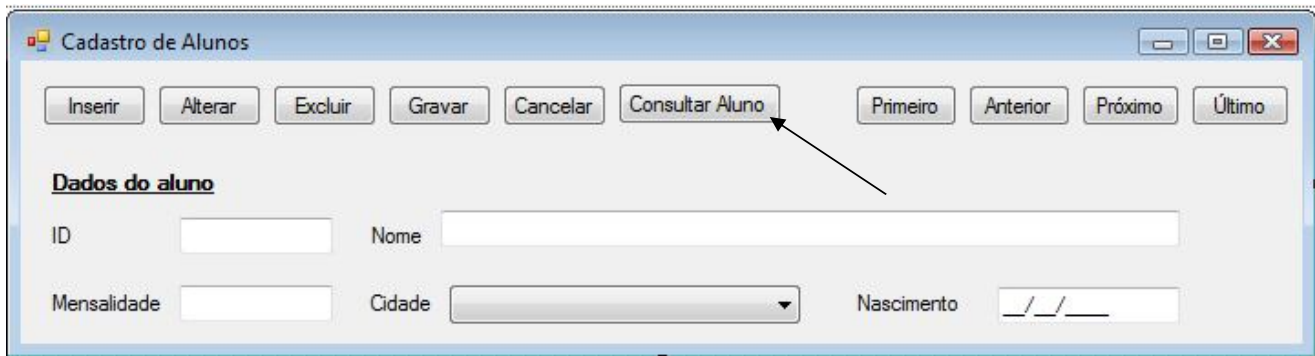
    if (cidadeId != 0)
        where = where + " and cidades.id = " + cidadeId.ToString();

    string sql =
        "select alunos.id as 'ID', alunos.nome as 'Nome do Aluno'," +
        "cidades.nome as 'Cidade' " +
        "from alunos " +
        "inner join cidades on alunos.cidadeId = cidades.Id " +
        where + " order by alunos.nome ";

    return Metodos.ExecutaSelect(sql);
}

```

Faça agora as seguintes alterações no cadastro do aluno para utilizar a tela de pesquisa:



```

/// <summary>
/// Altera a tela para se adequar a um determinado modo de operação
/// que pode ser inclusão, alteração ou consulta
/// </summary>
/// <param name="modo">modo para o qual se deseja alterar</param>
private void AlteraParaModo(modoperacao modo)
{
    bool liga = (modo != modoOperacao.consulta);

    txtNome.Enabled = liga;
    txtMensalidade.Enabled = liga;
    txtData.Enabled = liga;
    cbCidade.Enabled = liga;

    btnInserir.Enabled = (modo == modoOperacao.consulta);
    btnAlterar.Enabled = (modo == modoOperacao.consulta);
    btnExcluir.Enabled = (modo == modoOperacao.consulta);
    btnPrimeiro.Enabled = (modo == modoOperacao.consulta);
    btnAnterior.Enabled = (modo == modoOperacao.consulta);
    btnProximo.Enabled = (modo == modoOperacao.consulta);
    btnUltimo.Enabled = (modo == modoOperacao.consulta);
    btnConsultar.Enabled = (modo == modoOperacao.consulta);

    btnGravar.Enabled = (modo != modoOperacao.consulta);
    btnCancelar.Enabled = (modo != modoOperacao.consulta);
}

```

```
if (modo == modoOperacao.inclusao)
{
    txtId.Enabled = true;
    LimpaCampos();
}
else
```

```

        txtId.Enabled = false;
    }

```

```

private void btnConsultar_Click(object sender, EventArgs e)
{
    frPesquisaAluno formPesquisa = new frPesquisaAluno();
    formPesquisa.ShowDialog();

    if (formPesquisa.idAlunoSelecionado != 0)
    {
        AlunoVO aluno = AlunoDAO.Consultar(formPesquisa.idAlunoSelecionado);
        PreencheCamposTela(aluno);
    }
}

```

## Recuperando o último valor inserido de um campo identity

Por exemplo, dada a tabela abaixo, que possui um campo identity:

```

CREATE TABLE tbPessoa(
    id int NOT NULL primary key Identity (1,1),
    nome varchar(100) NULL)

```

Após inserir um ou mais registros:

```

insert into tbPessoa( nome) values ('Ana');
insert into tbPessoa( nome) values ('Maria');
insert into tbPessoa( nome) values ('Daniela');

```

Para saber qual foi o último valor identity inserido, execute uma consulta no banco de dados com a instrução abaixo:

```

select scope_identity() as 'UltimoIdInserido'

```

No C#, basta executar o comando acima como se fosse uma consulta normal, e pegar o retorno através do campo 'UltimoIdInserido'. Observe que o comando acima irá retornar apenas 1 registro no BD:

Results		Messages	
		UltimoIdInserido	
1		3	



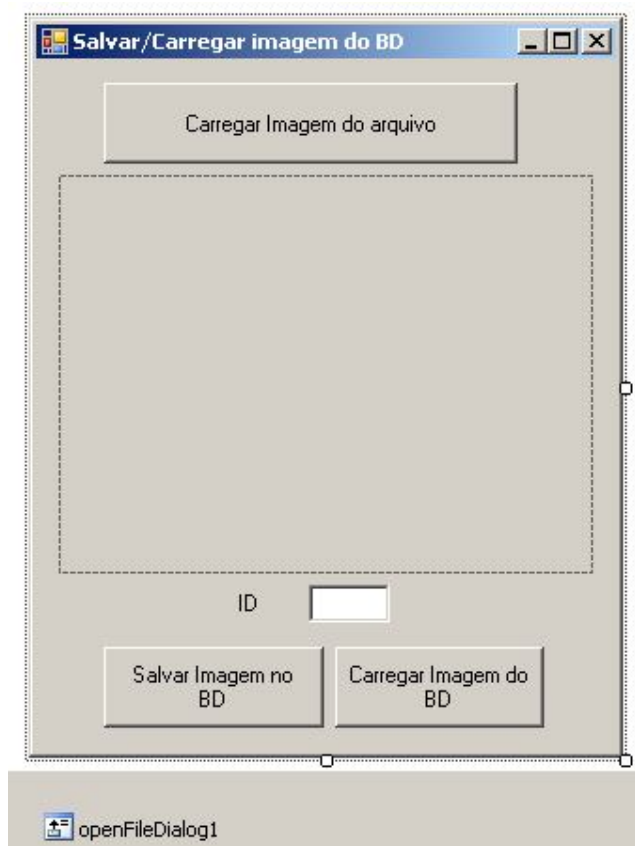
## Salvando e Recuperando Imagens no Banco de Dados

Para realizar o exemplo abaixo, crie a seguinte tabela:

```
CREATE TABLE [dbo].[TbImagens] (  
    [id] [int] NOT NULL primary key,  
    [imagem] [image] NULL)
```

Para fins didáticos, vamos fazer todo o código no formulário principal.

Crie uma tela como a abaixo:



**Botão de carregar imagem:**

```
private void btnCarregarIMG_Click(object sender, EventArgs e)  
{  
    if (openFileDialog1.ShowDialog() == DialogResult.OK)  
    {  
        pictureBox1.ImageLocation = openFileDialog1.FileName;  
    }  
}
```

Crie um método para conexão com BD:

```
private SqlConnection GetConexao()
{
    SqlConnection conexao = new SqlConnection("Data Source=localhost;Initial
Catalog=AulaDB;Integrated Security=False; user id=sa");
    conexao.Open();
    return conexao;
}
```

```
/// <summary>
/// Este método tem por objetivo testar se o valor recebido é DBNull. Se for, devolve
/// uma lista de bytes.
/// Se não for, devolve uma lista de bytes vazia.
/// </summary>
/// <param name="p_valor">Valor que se deseja testar se é DBNull.</param>
/// <returns>Retornar 0 bytes caso o valor recebido seja DBNull ou, caso não seja,
/// retornar o objeto recebido.</returns>
public static byte[] DBNullAsByteArray(object p_valor)
{
    if (p_valor == DBNull.Value)
        return new byte[0];
    else
        return (byte[])p_valor;
}
```

### Botão de salvar no BD

```
private void btnSalvar_Click(object sender, EventArgs e)
{
    if (string.IsNullOrEmpty(pictureBox1.ImageLocation))
    {
        MessageBox.Show("Selecione uma imagem.");
        return;
    }

    SqlConnection conexao = GetConexao();
    try
    {
        //criamos um vetor de bytes com os dados da imagem.
        byte[] vetor = File.ReadAllBytes(pictureBox1.ImageLocation);

        string sql = "insert into TbImagens (id,imagem) " +
            "values (@id, @imagem)"; // @imagem é um parâmetro

        int id = Convert.ToInt16(txtID.Text);

        //criamos o parâmetro no c# e informamos seu valor
        SqlParameter parametroId = new SqlParameter("@id", id);
        SqlParameter parametroImagem = new SqlParameter("@imagem", vetor);

        SqlCommand comando = new SqlCommand(sql, conexao);
        comando.Parameters.Add(parametroId); // adicionamos o parâmetro ao comando
        comando.Parameters.Add(parametroImagem); // adicionamos o parâmetro ao
            // comando

        comando.ExecuteNonQuery();
        MessageBox.Show("Gravado com sucesso!");
    }
    finally
```

```
{  
    conexao.Close();  
}  
}
```

### Botão de carregar no BD

```
private void btnCarregaImg_Click(object sender, EventArgs e)  
{  
    SqlConnection conexao = GetConexao();  
    try  
    {  
        string sql = "select * from tbImagens where id = " + txtID.Text;  
        SqlDataAdapter adapter = new SqlDataAdapter(sql, conexao);  
        DataTable tabela = new DataTable();  
        adapter.Fill(tabela);  
  
        if (tabela.Rows.Count == 0)  
        {  
            MessageBox.Show("Não há registros na tabela!");  
            return;  
        }  
  
        // uma imagem representada na linguagem é um vetor de bytes!  
        byte[] vetor = DBNull.AsByteArray(tabela.Rows[0]["imagem"]);  
        MemoryStream stmBLOBData = new MemoryStream(vetor);  
  
        pictureBox1.Image = Image.FromStream(stmBLOBData);  
    }  
    finally  
    {  
        conexao.Close();  
    }  
}
```

## Armazenando dados temporários com DataTable e com DataGridView

O objetivo deste exemplo é demonstrar o uso dos recursos do DataTable, como armazenar os dados em memória, Salvar e recuperar de arquivos XML. Como forma de apresentar os dados do DataTable será utilizado o DataGridView. Este componente também aceitará edição (para o campo quantidade).

Será demonstrado como incluir, alterar, excluir e consultar dados em um DataTable.

Todas as colunas do DataGridView foram inseridas via código.

### Código completo do programa:

```
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 MestreDetalhe
{
    public partial class Form1 : Form
    {
        DataTable TabelaTemp;
        DataTable TabelaProdutos;

        #region Métodos Auxiliares
```

```

/// <summary>
/// Método para criar e popular a tabela de produtos disponíveis para compra
/// </summary>
private void CriaTabelaProdutos()
{
    TabelaProdutos = new DataTable();

    DataColumn coluna = new DataColumn("ProdutoID", typeof(int));
    TabelaProdutos.Columns.Add(coluna);

    coluna = new DataColumn("ProdutoDescricao", typeof(string));
    TabelaProdutos.Columns.Add(coluna);

    //popula a tabela de produtos
    DataRow registro = TabelaProdutos.NewRow();
    registro["ProdutoID"] = 1;
    registro["ProdutoDescricao"] = "Borracha";
    TabelaProdutos.Rows.Add(registro);

    registro = TabelaProdutos.NewRow();
    registro["ProdutoID"] = 2;
    registro["ProdutoDescricao"] = "Lápis";
    TabelaProdutos.Rows.Add(registro);

    registro = TabelaProdutos.NewRow();
    registro["ProdutoID"] = 3;
    registro["ProdutoDescricao"] = "Caneta";
    TabelaProdutos.Rows.Add(registro);
}

/// <summary>
/// Verifica se um determinado valor string é um inteiro válido.
/// </summary>
/// <param name="valor"></param>
/// <returns></returns>
private bool ValidaInteiro(string valor)
{
    try
    {
        Convert.ToInt32(valor);
        return true;
    }
    catch
    {
        return false;
    }
}

/// <summary>
/// Pesquisa um produto na tabela de produtos
/// </summary>
/// <param name="codigoProduto"></param>
/// <returns></returns>
private DataRow PesquisaProduto(string codigoProduto)
{
    if (!ValidaInteiro(codigoProduto))
        return null;

    // o método select de um datatable retorna um vetor com todos os registros
    // do datatable que se encaixem no filtro informado.
    DataRow[] rowProduto = TabelaProdutos.Select("ProdutoId = " + txtProduto.Text);
    if (rowProduto.Length == 0)
        return null;
    else
        return rowProduto[0];
}

```

```

/// <summary>
/// Retorna o caminho onde está e/ou será salvo o arquivo de dados
/// </summary>
/// <returns></returns>
private string RetornaCaminhoXML()
{
    string arq = System.IO.Path.GetDirectoryName(Application.ExecutablePath) +
        "\\Dados.XML";

    return arq;
}

#endregion

/// <summary>
/// construtor do form
/// </summary>
public Form1()
{
    InitializeComponent();

    CriaTabelaProdutos();

    #region Criando a Tabela Temporária
    TabelaTemp = new DataTable();
    TabelaTemp.TableName = "tbDados";

    DataColumn coluna = new DataColumn("ProdutoID", typeof(int));
    TabelaTemp.Columns.Add(coluna);

    coluna = new DataColumn("ProdutoDescricao", typeof(string));
    TabelaTemp.Columns.Add(coluna);

    coluna = new DataColumn("Quantidade", typeof(int));
    TabelaTemp.Columns.Add(coluna);
    #endregion

    #region criação dinâmica das colunas do DataGridView
    DataGridViewTextBoxColumn colProdutoId = new DataGridViewTextBoxColumn();
    colProdutoId.Name = "colProdutoId";
    colProdutoId.ReadOnly = true;
    colProdutoId.Width = 100;
    colProdutoId.DataPropertyName = "ProdutoID"; // o nome do campo do datatable
    colProdutoId.HeaderText = "Produto";
    colProdutoId.DefaultCellStyle.BackColor = SystemColors.ButtonFace;

    dataGridView1.Columns.Add(colProdutoId);

    DataGridViewTextBoxColumn colProdutoDescricao = new DataGridViewTextBoxColumn();
    colProdutoDescricao.Name = "colProdutoDescricao";
    colProdutoDescricao.ReadOnly = true;
    colProdutoDescricao.Width = 300;
    colProdutoDescricao.DataPropertyName = "ProdutoDescricao"; // o nome do campo do
datatable
    colProdutoDescricao.HeaderText = "Descrição";
    colProdutoDescricao.DefaultCellStyle.BackColor = SystemColors.ButtonFace;
    dataGridView1.Columns.Add(colProdutoDescricao);

    DataGridViewTextBoxColumn colQtde = new DataGridViewTextBoxColumn();
    colQtde.Name = "colQtde";
    colQtde.ReadOnly = false; // permite escrita nesse campo
    colQtde.Width = 100;
    colQtde.DataPropertyName = "Quantidade"; // o nome do campo do datatable
    colQtde.HeaderText = "Quantidade";
    dataGridView1.Columns.Add(colQtde);

    dataGridView1.AllowUserToAddRows = false;
    dataGridView1.AllowUserToDeleteRows = false;
    dataGridView1.AllowUserToResizeRows = false;

```

```

dataGridView1.MultiSelect = false;
dataGridView1.AutoGenerateColumns = false;

dataGridView1.DataSource = TabelaTemp;

#endregion
}

private void btnGravar_Click(object sender, EventArgs e)
{
    if (!ValidaInteiro(txtQtde.Text))
    {
        MessageBox.Show("Quantidade inválida!");
        return;
    }

    DataRow rowProduto = PesquisaProduto(txtProduto.Text);
    if (rowProduto == null)
    {
        MessageBox.Show("O produto informado não existe!");
        return;
    }

    //verifica se o produto já havia sido adicionado
    //e neste caso só incrementa a quantidade.

    DataRow[] registros = TabelaTemp.Select("ProdutoID = " + txtProduto.Text);
    if (registros.Length == 1) // se já existir, só altera a quantidade
    {
        registros[0]["Quantidade"] = Convert.ToInt32(registros[0]["Quantidade"]) +
            Convert.ToInt32(txtQtde.Text);
    }
    else
    {
        // Se não encontrar, cria um novo registro.
        DataRow registro = TabelaTemp.NewRow();
        registro["ProdutoID"] = txtProduto.Text;
        registro["ProdutoDescricao"] = rowProduto["ProdutoDescricao"];
        registro["Quantidade"] = txtQtde.Text;
        TabelaTemp.Rows.Add(registro);
    }
}

/// <summary>
/// Ao sair do campo txtProduto, ele dispara este evento e preenche o nome do produto
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void txtProduto_Leave(object sender, EventArgs e)
{
    DataRow rowProduto = PesquisaProduto(txtProduto.Text);
    if (rowProduto != null)
    {
        txtDescricaoProduto.Text = rowProduto["ProdutoDescricao"].ToString();
    }
    else
    {
        txtDescricaoProduto.Clear();
    }
}

private void button2_Click(object sender, EventArgs e)
{
}

```

```

private void btnApagarTudo_Click(object sender, EventArgs e)
{
    if (MessageBox.Show("Confirma?", "Atenção", MessageBoxButtons.YesNo,
        MessageBoxIcon.Question) == DialogResult.Yes)
        TabelaTemp.Clear();
}

private void btnSalvar_Click(object sender, EventArgs e)
{
    TabelaTemp.WriteXml(RetornaCaminhoXML());
    MessageBox.Show("Dados salvos com sucesso!");
}

private void btnCarregar_Click(object sender, EventArgs e)
{
    if (!System.IO.File.Exists(RetornaCaminhoXML()))
    {
        MessageBox.Show("Arquivo de dados não existe!");
        return;
    }

    TabelaTemp.ReadXml(RetornaCaminhoXML());
    MessageBox.Show("Dados carregados com sucesso!");
}

/// <summary>
/// Este evento do grid serve para validar os dados
/// digitados diretamente no grid.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void dataGridView1_CellValidating(object sender,
DataGridViewCellValidatingEventArgs e)
{
    //verifica se é a coluna de quantidade
    if (e.ColumnIndex == dataGridView1.Columns[2].Index)
    {
        string valor = e.FormattedValue.ToString();
        if (!ValidaInteiro(valor))
        {
            MessageBox.Show("Valor inválido!");
            e.Cancel = true; //não aceita o que foi digitado.
        }
    }
}

private void btnRemoverRegistro_Click(object sender, EventArgs e)
{
    if (dataGridView1.SelectedRows.Count == 0)
    {
        MessageBox.Show("Não há linha selecionada!");
        return;
    }

    if (MessageBox.Show("Confirma?", "Atenção", MessageBoxButtons.YesNo,
        MessageBoxIcon.Question) == DialogResult.No)
        return;

    //Descobre qual a linha da tabela relacionada com o linha do grid selecionada
    DataRowView datarowview = dataGridView1.SelectedRows[0].DataBoundItem as
DataRowView;
    DataRow registro = datarowview.Row;

    //Apaga a linha da tabela temporária
    TabelaTemp.Rows.Remove(registro);
}

```



```

private void btnPreencher_Click(object sender, EventArgs e)
{
    lbProdutos.Items.Clear();

    //Varre todos os registros do datatable de produtos e preenche um listbox
    foreach (DataRow registro in TabelaProdutos.Rows)
    {
        lbProdutos.Items.Add(
            registro["ProdutoID"].ToString() + " - " +
            registro["ProdutoDescricao"].ToString());
    }
}
}

```

Exemplo do Programa em Execução:

Código do Produto: 3 Caneta Qtde: 9 Gravar

Você pode alterar a quantidade diretamente no grid

	Produto	Descrição	Quantidade
	2	Lápis	5
▶	1	Borracha	8

Salvar em XML Carregar de XML Apaga tudo! Remover Registro Selecionado

Listagem de produtos disponíveis: Preencher produtos disponíveis

1 - Borracha  
2 - Lápis  
3 - Caneta