Capítulo III – Aplicações Windows Forms

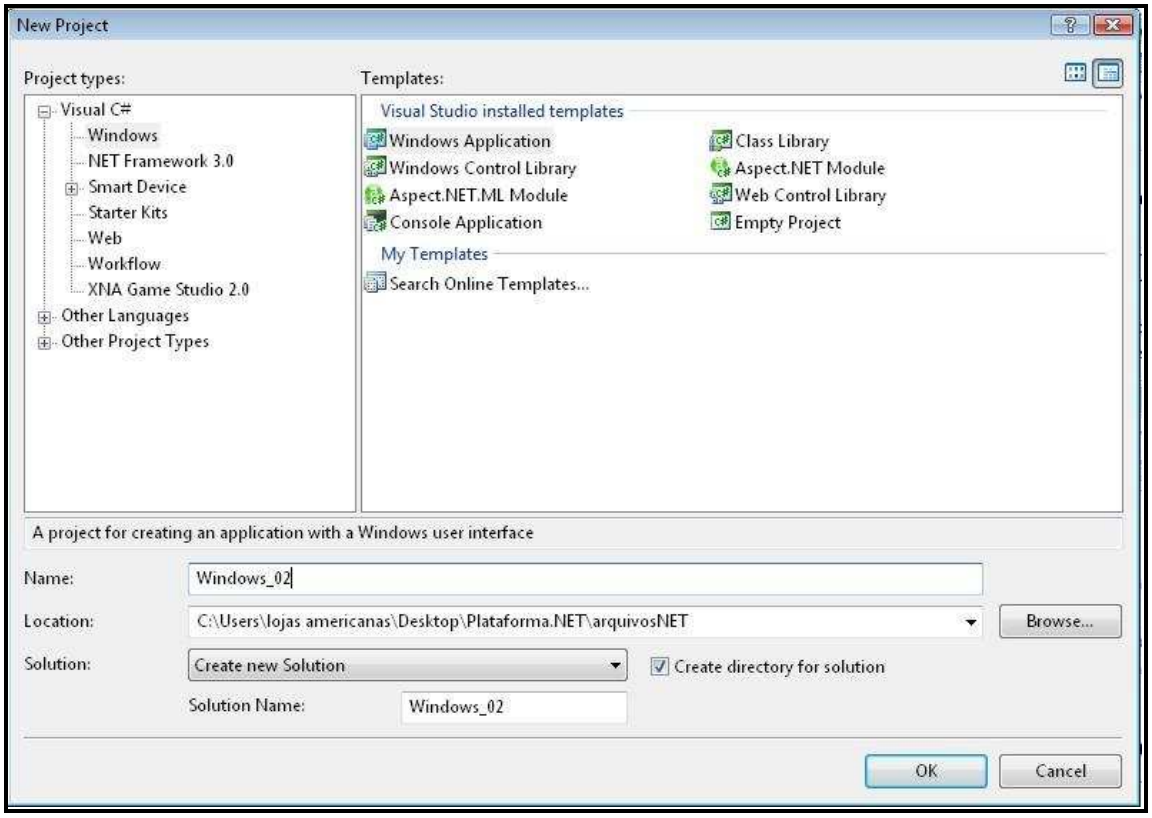
Introdução

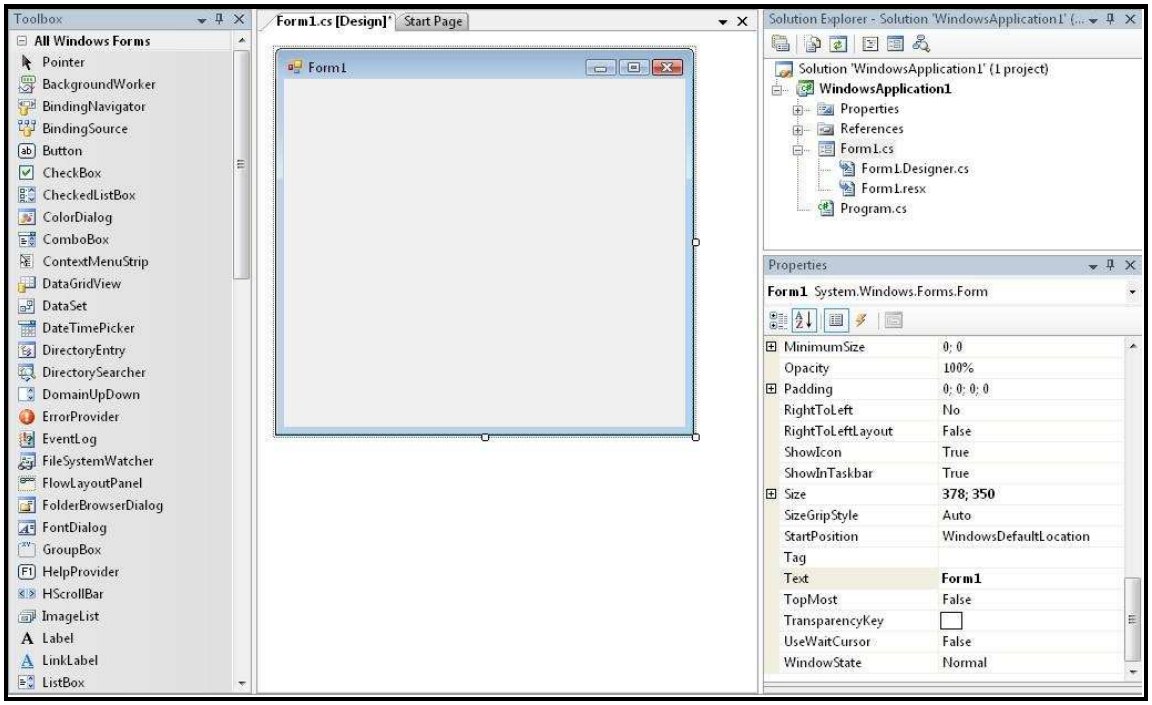
Aplicativos **Windows Forms** são os aplicativos com janelas muito predominantes na plataforma Windows e que tiveram as suas origens nas pesquisas sobre a Interface Gráfica com o Usuário (GUI), dando origem ao Mac nos anos 70.

As mesmas idéias apresentadas acima sobre a **Herança** e **Reutilização de código** são predominantes neste tipo de aplicação. Vamos desenvolver uma aplicação Windows Forms que sonda os processos ativos do sistema, lista threads de um dado processo e permite matar um processo. Não faremos tratamento de exceções na primeira abordagem deste programa, e após a conclusão da primeira parte do mesmo, procederemos a uma fase de testes, forçando entradas espúrias no programa para abordarmos o tratamento de exceções que deverá deixar o programa mais robusto.

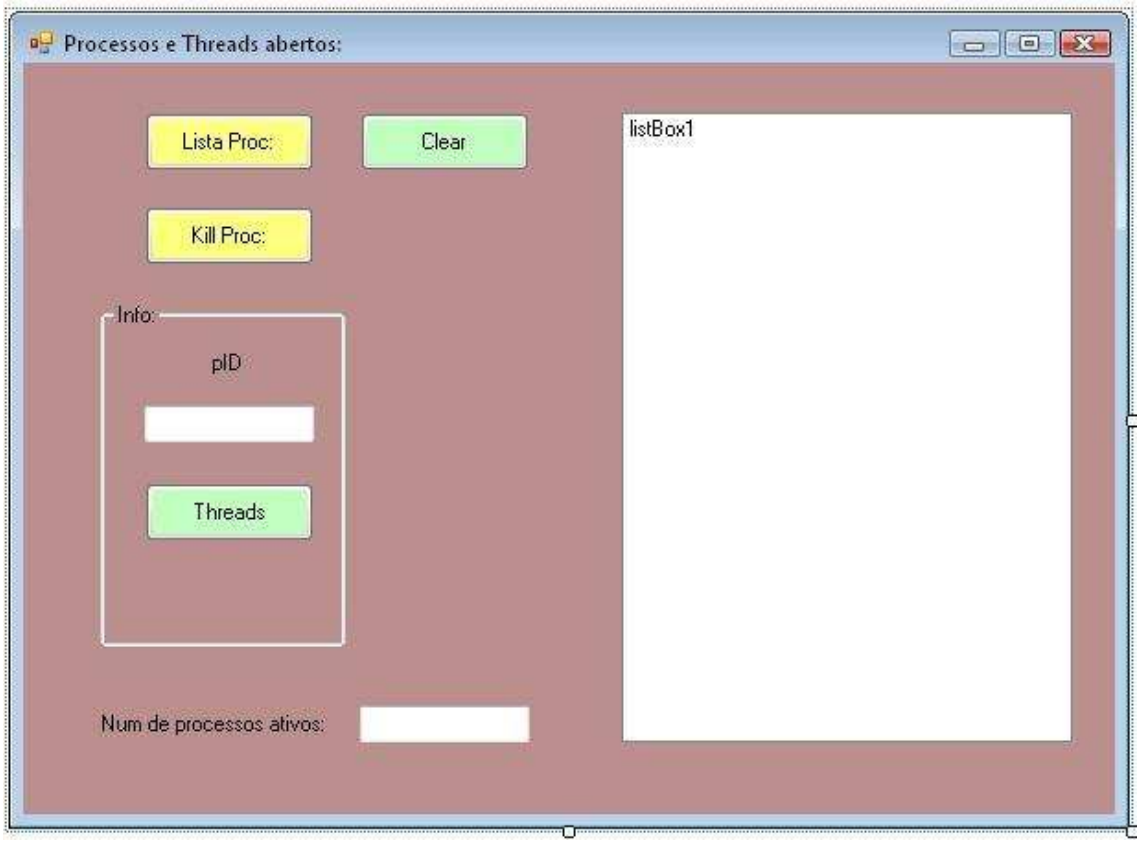
1) Sondando processos e threads do Sistema Operacional

Inicie uma nova aplicação Windows, agora do tipo **Windows Forms**. Denomine a solução Windows_02:

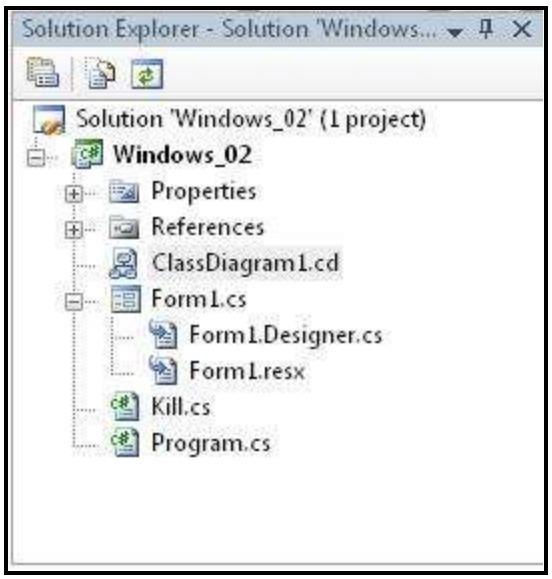




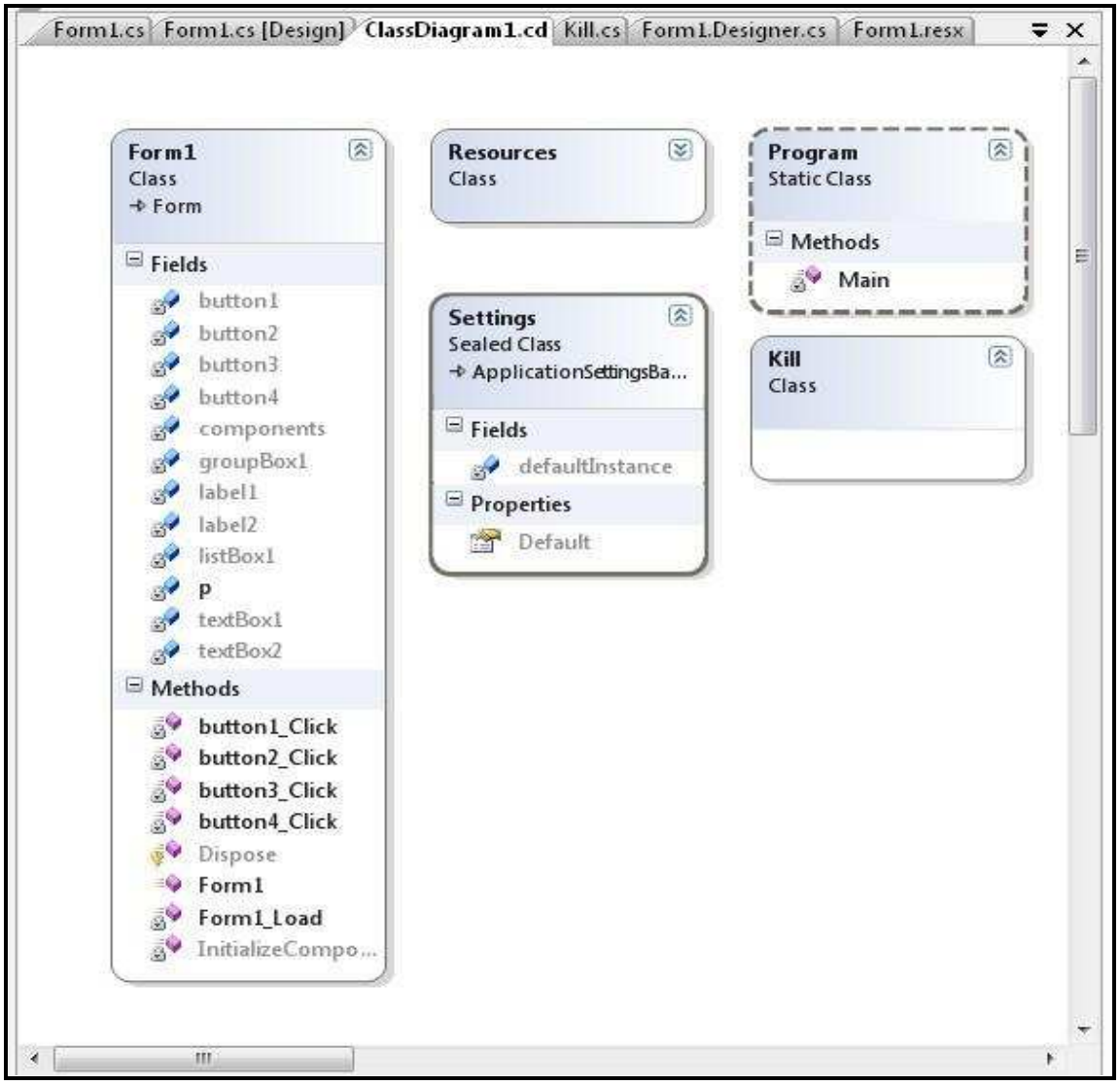
A configuração ideal deverá ser acima, com a toolbox à esquerda, o **solution explorer** à esquerda e o painel de propriedades abaixo. O formulário está no centro. Sobre o formulário que se abre, arraste quatro **botões de controle**, duas **caixas de texto** e uma **listBox.** Em seguida coloque um **groupBox** e um **label**, configurando-os muito semelhante à figura abaixo, o formulário acaba se tornando (detalhes serão vistos em sala de aula):



O **solution explorer** desta solução pode ser apresentado para verificação:



Criamos um **Diagrama de Classes** (que entre outras coisas serve para documentação da solução):



Em seguida, uma vez configuradas as propriedades dos objetos do formulário, podemos dar duplo clique em cada um dos botões de comando do formulário e escrever os códigos abaixo. Para o botão **Lista Proc:** o código é o seguinte (adicione a cláusula **using System.Diagnostics;** no corpo de diretivas da classe deste formulário, que engloba tais botões):

```
private void button1_Click(object sender, EventArgs e)
{
    Process[] runningProcs = Process.GetProcesses(".");

    StringBuilder builder = new StringBuilder();

  foreach (Process p in runningProcs)
  {
    string info = string.Format("->PID: {0}\tNome:
{1}", p.Id, p.ProcessName + "\n");
```

```
    listBox1.Items.Add(info);

    }
   textBox1.Text = builder.Append("\n" +
runningProcs.Length.ToString()).ToString();

}
```

Após a inclusão da clásula **using System.Diagnostics;** podemos declarar a expressão:

```
Process[] runningProcs = Process.GetProcesses(".");
```

Esta expressão declara um objeto runningProcs, o qual resgata todos os processos ativos do sistema operacional. O colchete indica que este objeto é na verdade uma coleção de itens enumeráveis.
Em seguida, as linhas:

```
  foreach (Process p in runningProcs)
  {
    string info = string.Format("->PID: {0}\tNome:
{1}", p.Id, p.ProcessName + "\n");

    listBox1.Items.Add(info);

  }
```

Realizam as seguintes tarefas: após declarada a coleção **runningProcs**, o laço foreach (para cada) pega cada processo p pertencente à coleção **runningProcs** e adiciona à caixa de listagem **listBox1**. O método para a adição é a combinação de:

.Items.Add(info); após a listBox1.

O campo info é do tipo string e foi formatado para receber as variáveis p.Id e p.ProcessName. Estas variáveis são a combinação do objeto processo p e lhe é aplicado o método de recuperação do **Id** e de **ProcessName.**

As demais expressões neste código são auto-evidentes.

O botão **Kill Proc:** apresenta o seguinte código:

```
private void button2_Click(object sender, EventArgs e)
{
     Int32 pID = Convert.ToInt32(textBox2.Text);
     p = Process.GetProcessById(pID);
     p.Kill();

}
```

Na primeira linha, nós recuperamos o pID (número de identificação do processo), a partir da caixa de texto textBox2. Acionamos a sua propriedade Text, para recuperarmos o conteúdo texto da mesma e em seguida convertê-mo-la para inteiro. Quando a variável pID recupera esta informação, ela passa tal parâmetro por valor, para o campo p, que é um processo (dentre os ativos). O método para isto é GetProcessById (obtenha processo via Id).

Pronto, o processo está identificado! A linha abaixo:

p.Kill( ); mata o processo identificado, usando o métod Kill( );

Usem sempre o Intellisense para obterem acesso aos métodos e propriedades.

Em seguida, o botão **Threads**, dentro da **groupBox**, apresenta o seguinte código:

```
private void button3_Click(object sender, EventArgs e)
{

        listBox1.Items.Clear();

        Int32 pID = Convert.ToInt32(textBox2.Text);

        p = Process.GetProcessById(pID);

        ProcessThreadCollection m_Thread = p.Threads;

       foreach(ProcessThread pT in m_Thread)
       {
          string info = string.Format("Thread
ID:{0},Prioridade: {1}",pT.Id,pT.PriorityLevel);
          listBox1.Items.Add(info);

       }
label1.Text = "Número de threads: ";
textBox1.Text = "Número de threads: " +
m_Thread.Count;

 }
```

A primeira linha: listBox1.Items.Clear( ); limpa a caixa de listagem de quaisquer itens anteriores. As duas próximas linhas obtém o pID do processo e um processo em si, a partir do pID. Em seguida, declaramos uma coleção, que é a coleção de **threads** ativos que pertencem a um dado processo:

```
ProcessThreadCollection m_Thread = p.Threads;
```

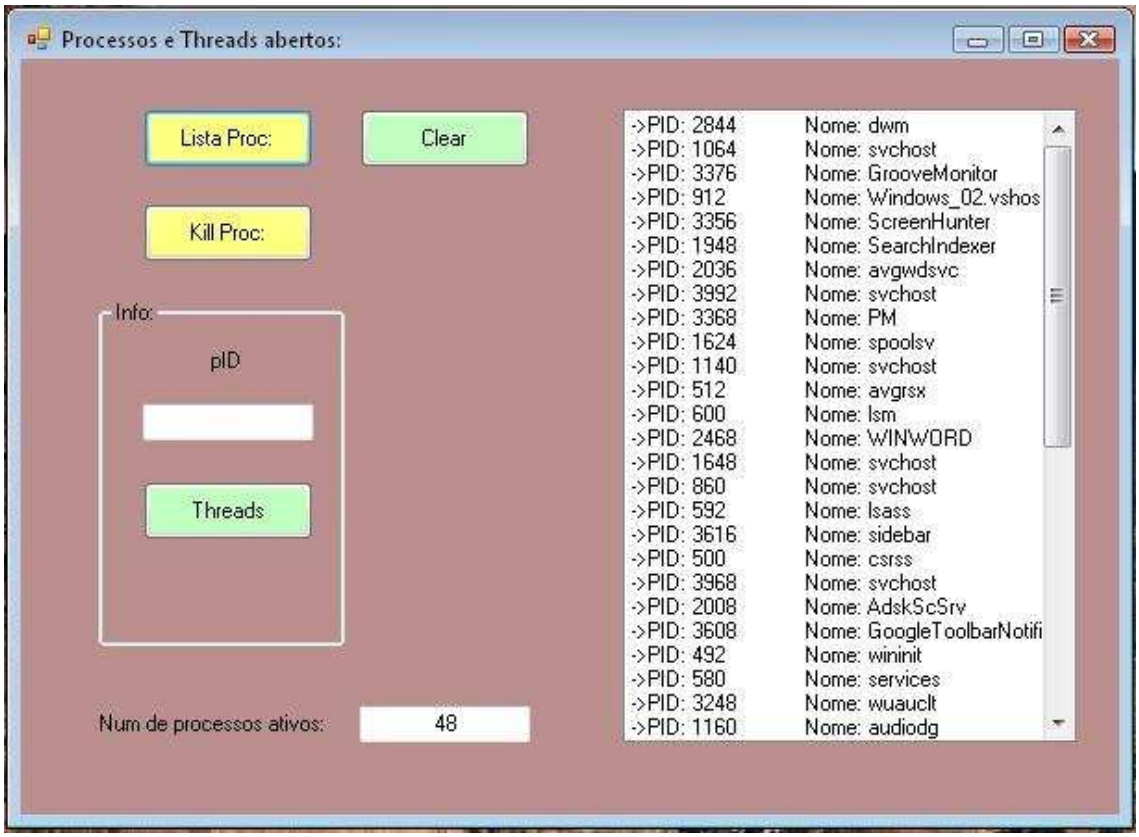
m_Thread é a coleção de threads do processo p. Em seguida, o laço foreach( ) abaixo carrega a lista de threads ativos do processo p, o objeto thread ativo é declarado através de ProcessThread pT in m_Thread. As variáveis pT.Id e pT.PriorityLevel resgatam a Id e o nível de prioridade do thread pT pertencente à coleção m_Thread:

```
foreach(ProcessThread pT in m_Thread)
{
      string info = string.Format("Thread
ID:{0},Prioridade: {1}",pT.Id,pT.PriorityLevel);
      listBox1.Items.Add(info);

}
```

A linha abaixo carrega o número de threads deste processo na caixa de texto correspondente:

```
textBox1.Text = "Número de threads: " +
m_Thread.Count;
```

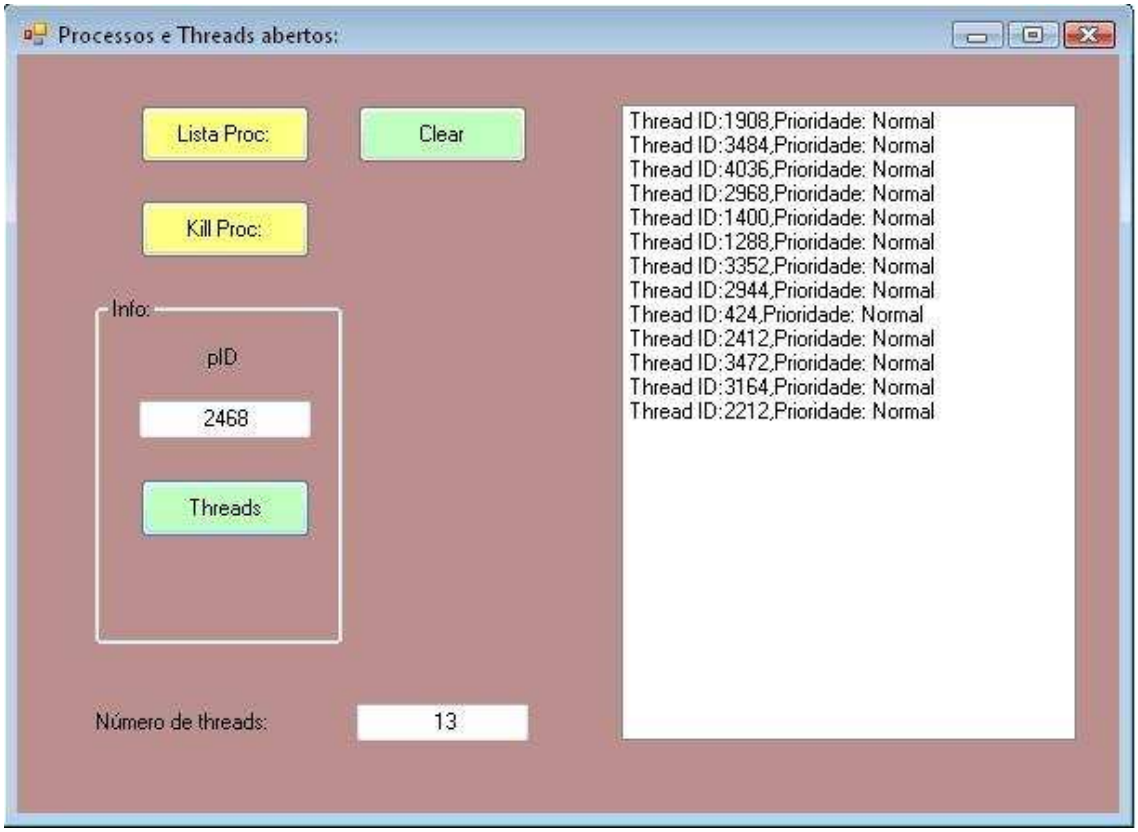
Não mudamos os nomes dos **botões de comando** acima, mas devemos em tempo de projeto dar nomes sugestivos aos botões de comando. Agora, vamos executar este programa, e capturar tela:



Clicamos no botão **Lista Proc:** e na **Caixa de Listagem** aparecem os processos ativos e o número de processos na caixa de texto. Podemos copiar e colar um número de processo e inserir na caixa pID do groupBox Info. Lembrem-se das aulas de sistemas operacionais e não matem processos de alta prioridade, para certificar-se da sua funcionalidade abra o PowerPoint e mate o processo que lhe é associado.

Vamos escrever o pID do Word para descobrirmos quantos threads estão associados a este programa:

Quando digitamos o pID do Word, na caixa de texto, e clicamos no botão **Threads**, aparece a lista de **Threads** do Word:



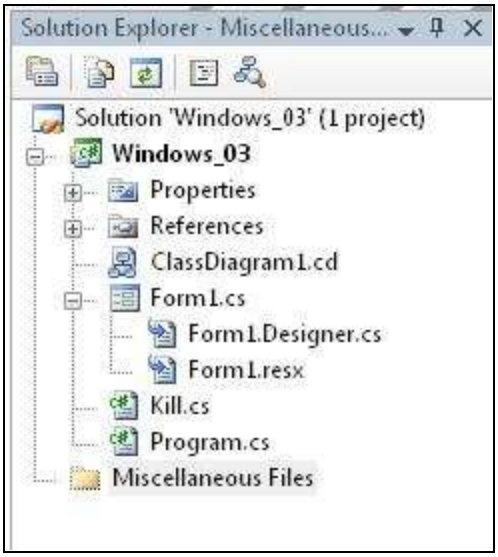
Se clicarmos em **Clear** a caixa de listagem é limpa e o label muda o seu texto.

Ainda não fizemos o tratamento de exceções deste programa, pois note que se escrevermos uma palavra ou errarmos a digitação na caixa de texto do groupBox Info, o programa deve travar. Desta forma, devemos proceder a uma análise mais fina do código e inserirmos tratamentos de erro (exceções), no programa como um todo.

Agora, devemos levar em conta que a aplicação atual é uma versão do programa. Podemos evoluir a partir desta versão indo em direção de uma nova e mais atualizada versão, contendo os tratamentos de erro, guardando a primeira versão como backup.

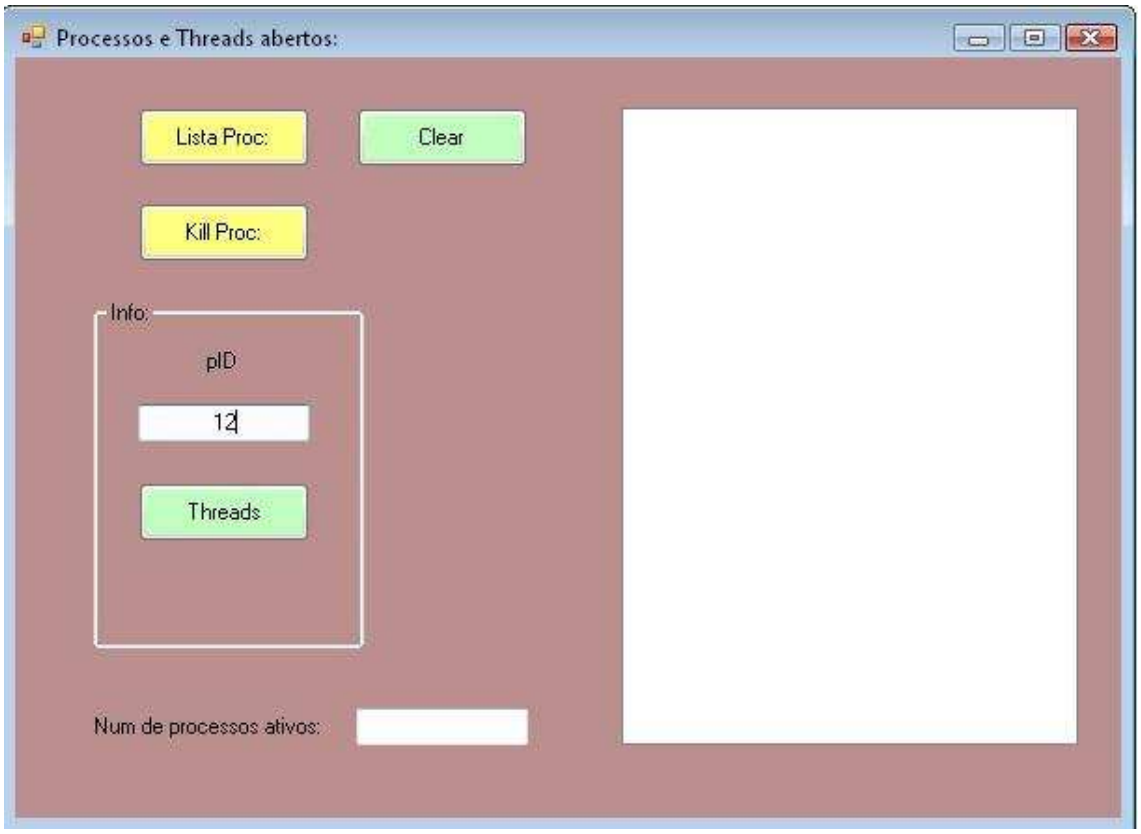
Para isso, copie a pasta que contém todos os arquivos do projeto antigo, renomeie a nova pasta. Quando você abrir a solução, desta pasta, ela deverá ter o mesmo nome da solução anterior. Você pode clicar com o botão direito do mouse e Renomear a solução e o nome do projeto.

Dê uma olhada no solution explorer da solução mais atualizada:



Esta solução mais atualizada está contida numa outra pasta, pois estamos trabalhando numa versão diferente, onde procederemos a testes forçando tratamento de exceções.

Rode o aplicativo, e desta vez, insira um número de pID na caixa de texto sem ter passado pela fase de listar os processos ativos, teremos a situação abaixo:



Se você clicar em Threads, agora, o programa deve travar, pois a aplicação não encontra qualquer processo com aquele número. O depurador volta à seção de código onde ocorreu a exceção:

```
private void button3_Click(object sender, EventArgs e)
{
    //(4)

    listBox1.Items.Clear();

    Int32 pID = Convert.ToInt32(textBox2.Text);

    p = Process.GetProcessById(pID);

    ProcessThreadCollection m_Thread = p.Threads;
```

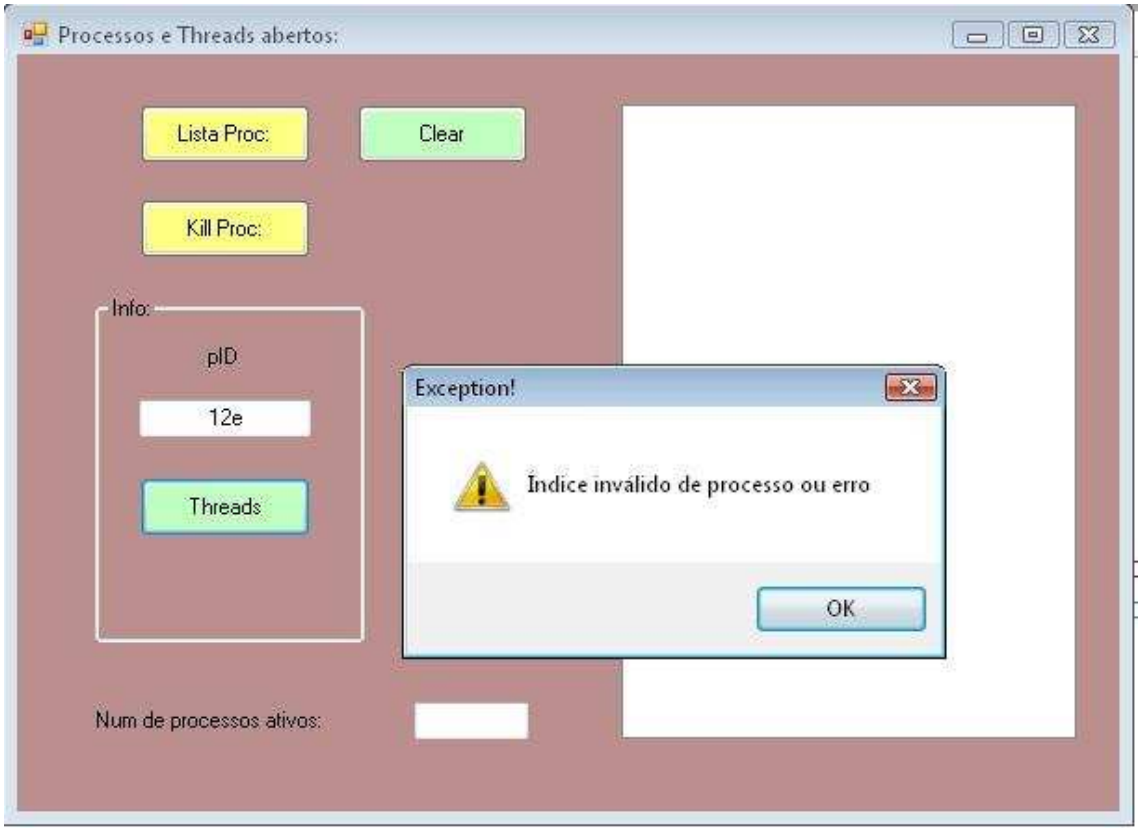
Falta tratarmos a exceção que foi disparada pela falta da ocorrencia do processo com aquele número. No código acima, vamos tratar a exceção com a diretiva try catch.

Vamos encapsular uma seção do código em region, e tratar o erro com try catch:

```
try
{
    Int32 pID = Convert.ToInt32(textBox2.Text);


    p = Process.GetProcessById(pID);

    ProcessThreadCollection m_Thread = p.Threads;

 foreach (ProcessThread pT in m_Thread)
 {
         string info = string.Format("Thread
ID:{0},Prioridade: {1}", pT.Id, pT.PriorityLevel);
         listBox1.Items.Add(info);


         label1.Text = "Número de threads: ";
         textBox1.Text = " " + m_Thread.Count;


 }


catch
{



}
```

Podemos experimentar a entrada de pIDs não pertencentes à coleção de processos ativos ou a entrada de caracteres não válidos. Este tratamento pode capturar e evitar o travamento, embora não seja capaz de indicar onde pode estar o erro. Você pode tentar tratar melhor, inserindo alguma comunicação e ação com o usuário, então o catch vazio acima pode ser substituído por:

```
catch
{

MessageBox.Show("Índice inválido de processo ou erro",
"Exception!", MessageBoxButtons.OK,
MessageBoxIcon.Exclamation);

  textBox2.Clear();
}
```



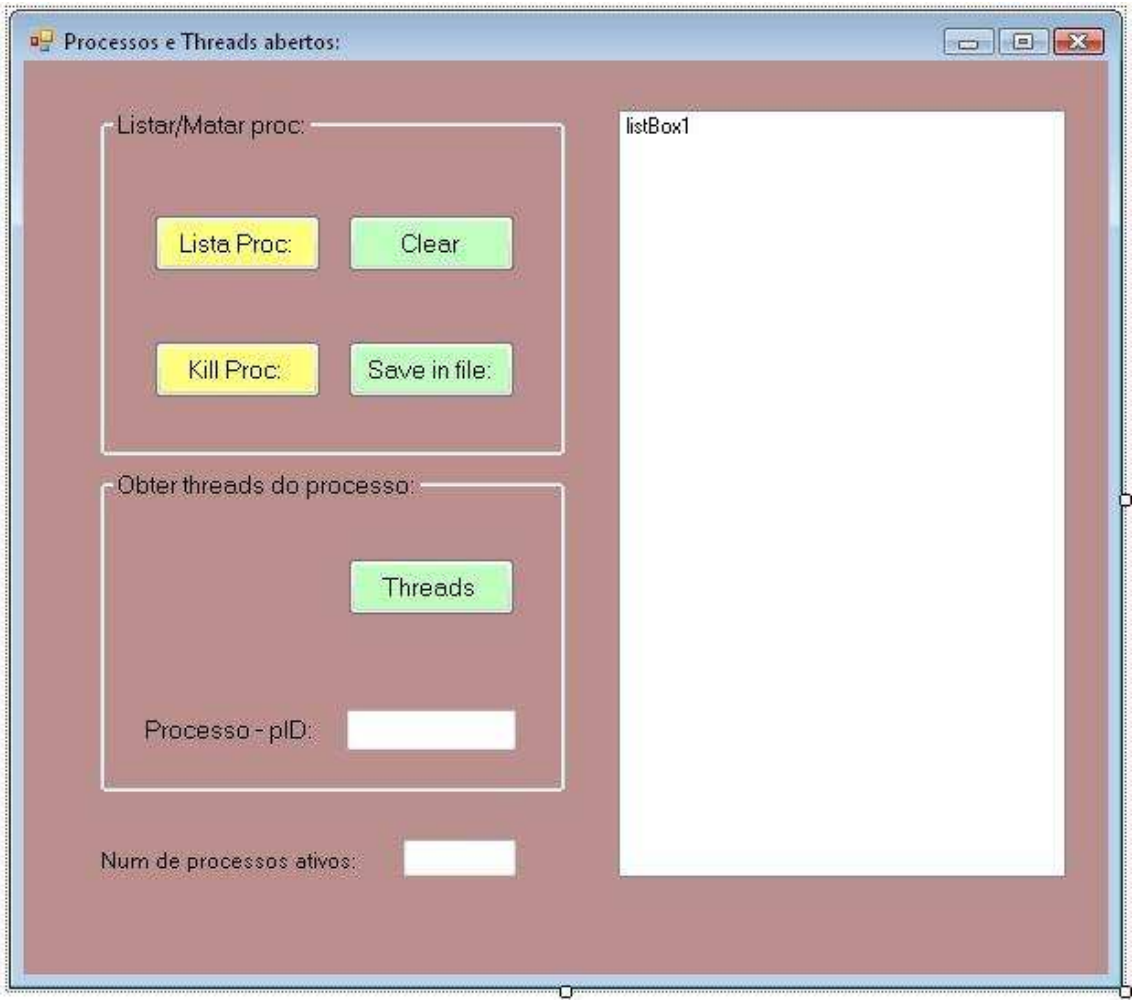
A figura acima mostra como a exceção foi tratada: o usuário digitou uma entrada inválida na caixa de texto, um tratamento de exceção foi disparado, comunicando ao usuário uma mensagem. Quando o usuário tecla OK com o mouse, a caixa de mensagem se fecha e a caixa de texto é limpa.

Deixo como exercício ao discente aplicar tratamento de exceção aos demais botões desta aplicação.

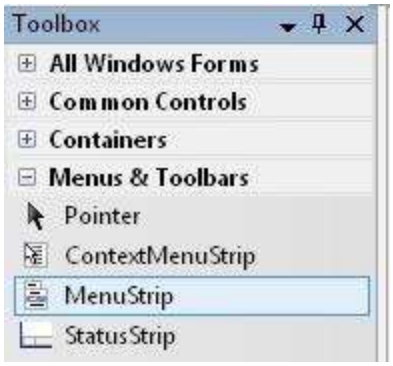
Vou supor que o discente tratou a exceção do botão **Kill Proc**: e vamos prosseguir com o projeto e lidar com o salvamento dos dados em arquivo. Vamos usar este mesmo projeto.

Para lidarmos com o fluxo de arquivos, devemos inserir a cláusula **using System.IO;** isto é, Input/Output.

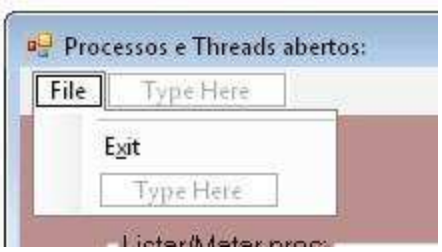
Antes disso, vamos mudar um pouco a configuração visual do nosso formulário, ajuste o tamanho do formulário, coloque outra groupBox e obtenha o seguinte projeto visual:



Antes de prosseguirmos com outras funções para os botões, Salvar em arquivo, etc, vamos inserir um item de menu que permite o encerramento desta aplicação. Na barra de ferramentas do Visual Studio, escolha MenuStrip. Dê duplo clique.



No canto superior esquerdo aparece uma faixa com os itens de menu para serem preenchidos, escreva File no primeiro item, clique no item logo abaixo, insira um separador e no próximo item escreva E&xit. A sua configuração final deve ser como a que está abaixo:



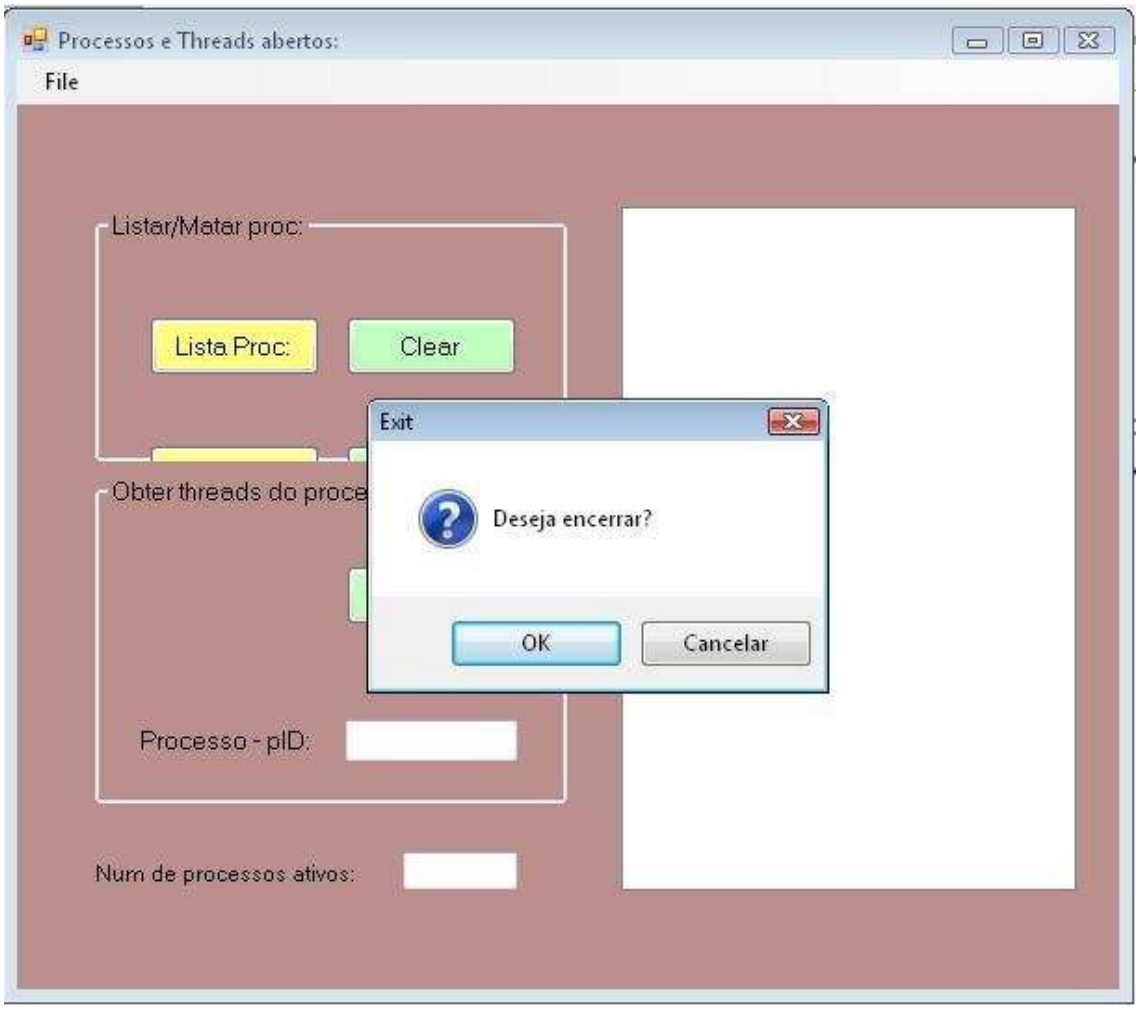
Agora selecione Exit acima e dê duplo clique no mesmo. Deve aparecer uma área de código para preenchermos:

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

  //código a ser escrito aqui.

}
```

Vamos escrever código que comanda a finalização da aplicação mediante confirmação do usuário. Para isto o código acima completo deve ser:

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

    DialogResult m_saida = new DialogResult();

    m_saida = MessageBox.Show("Deseja encerrar?",
"Exit", MessageBoxButtons.OKCancel,
MessageBoxIcon.Question);

    if (m_saida == DialogResult.OK)
    {
        Application.Exit();
    }

}
```

Declaramos uma variável tipo Caixa de Diálogo (DialogResult) cuja função é receber a entrada do usuário. Esta entrada deve ser a escolha entre as opções Ok ou Cancel, de uma caixa de diálogo. Se o usuário clicar em Ok, a aplicação será encerrada.

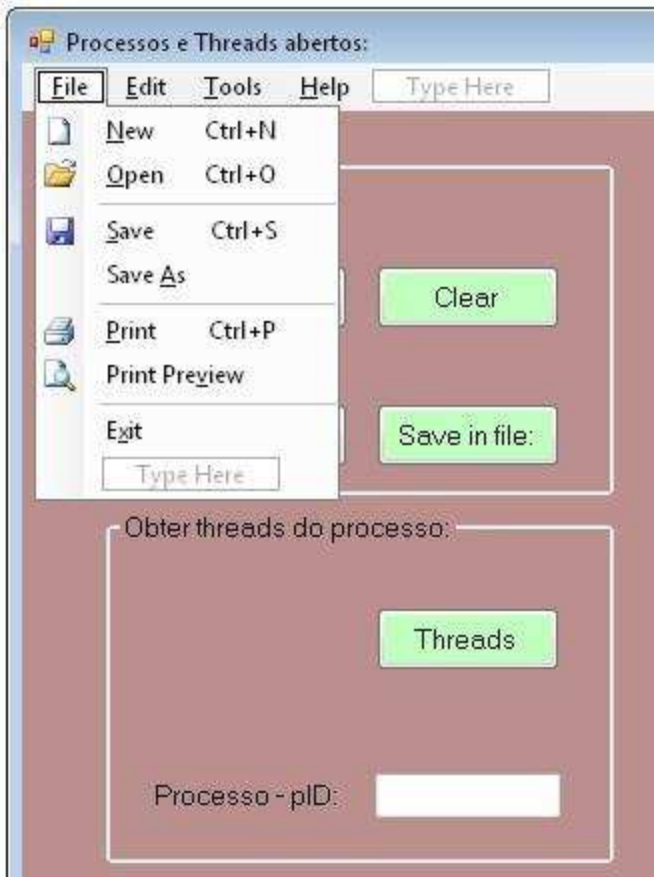


Caixa de diálogo mostrando as opções OK, Cancelar. A aplicação é encerrada Application.Exit( ), quando o usuário clicar em no botão OK.

Na plataforma .NET, toda a comunicação com caixas de diálogo exigem a declaração de uma variável tipo **DialogResult.**

Agora, vamos inserir uma lista de itens de menu do tipo padrão.

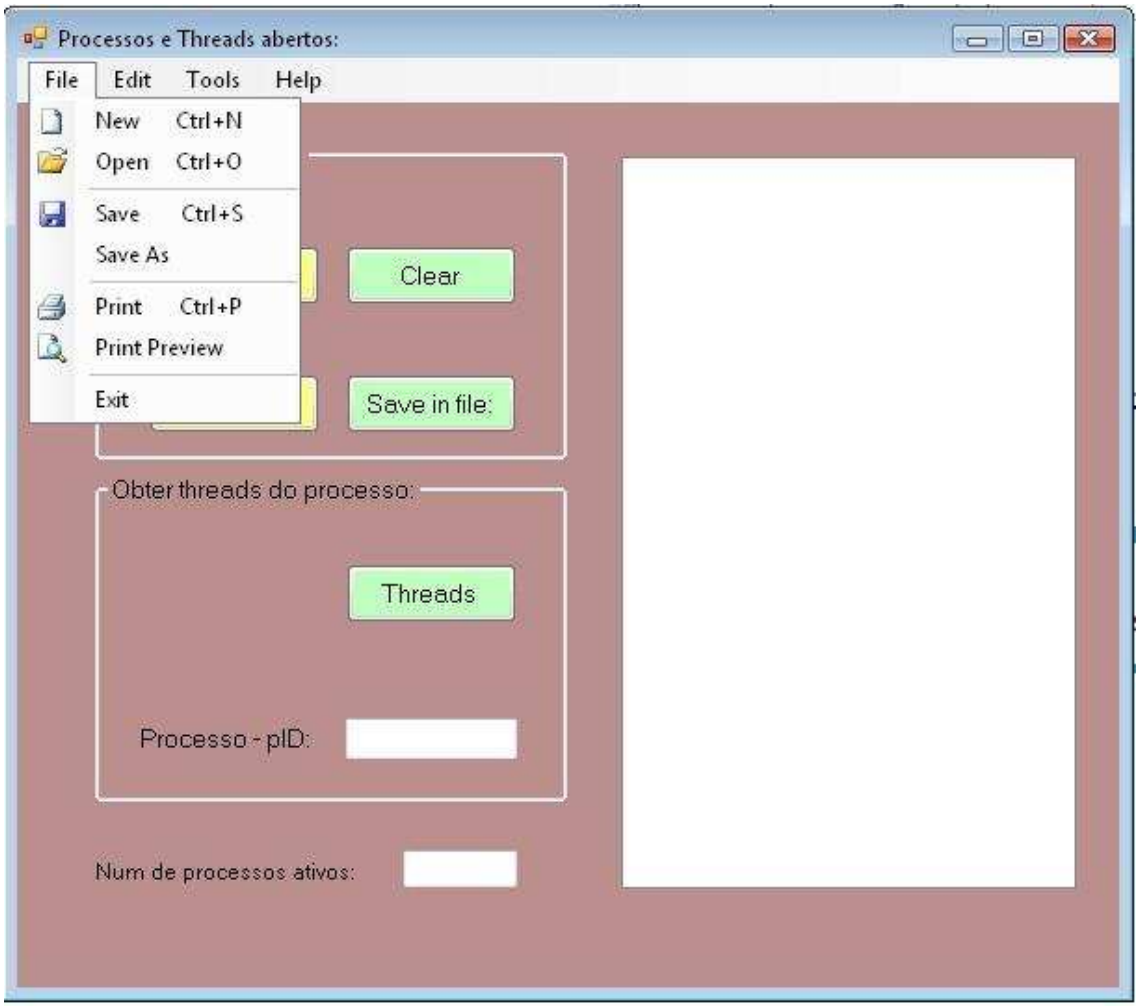
A vantagem de se usar uma lista de itens de menu padrão, é como o nome indica, usar uma lista que é padronizada, de acordo com o padrão Windows. Se clicarmos no item File por exemplo, veremos:



Dê duplo clique no item Exit nesta lista e escreva o mesmo código acima:

```
private void exitToolStripMenuItem1_Click(object
sender, EventArgs e)
{
        DialogResult m_saida = new DialogResult();

        m_saida = MessageBox.Show("Deseja encerrar?",
"Exit", MessageBoxButtons.OKCancel,
MessageBoxIcon.Question);

        if (m_saida == DialogResult.OK)
        {
           Application.Exit();
        }
 }
```

A execução do seu aplicativo é agora mais personalizada:



Clicando em Exit, a tela de encerramento da caixa de diálogo, aparece em meio ao programa:

Trabalhando com arquivos

Adicione um botão, com propriedade texto Save in file, e cor verde. Este botão vai salvar o conteúdo da caixa de listagem num arquivo tipo texto em C:\.

Trabalhando com arquivos

Vamos aproveitar o projeto acima e trabalhar o salvamento de informações em arquivos. O conceito de **stream** de dados é um dos mais importantes para sistemas de informação e TI. Isto é devido ao fato de que dados e informações devem fluir (stream) através do sistema/ou redes, e os sistemas de bancos de dados (SGBDs) são generalizações destes conceitos. Técnicas de programação com arquivos são muitas vezes ignoradas em muitos cursos, e a formação de um profissional na área de TI sem dominar estas técnicas torna-se duvidosa.

O **System.IO** namespace na plataforma.NET controla tudo o que você precisa para trabalhar com arquivos e diretórios em seu computador, tão bem quanto os streams.

A classe FileSystemWatcher

Esta é uma classe muito versátil, usada para sondar mudanças no sistema de arquivos. Você pode obter as informações sobre quando um arquivo é deletado, renomeado ou mudado. Outro grande uso é para **ambientes de rede**.

2) Um aplicativo WEB Browser

Estabeleça um novo tipo de aplicação **Windows** e denomine **WEB_Browser.**

Em **Location** clique em browse e escolha o diretório para a solução. Em solution name (nome da solução) marque a caixa **Create directory for solution**. Isto permitirá que você crie novos diretórios de solução, para cada nova solução, na mesma pasta em **Location** que você escolheu no passo anterior.

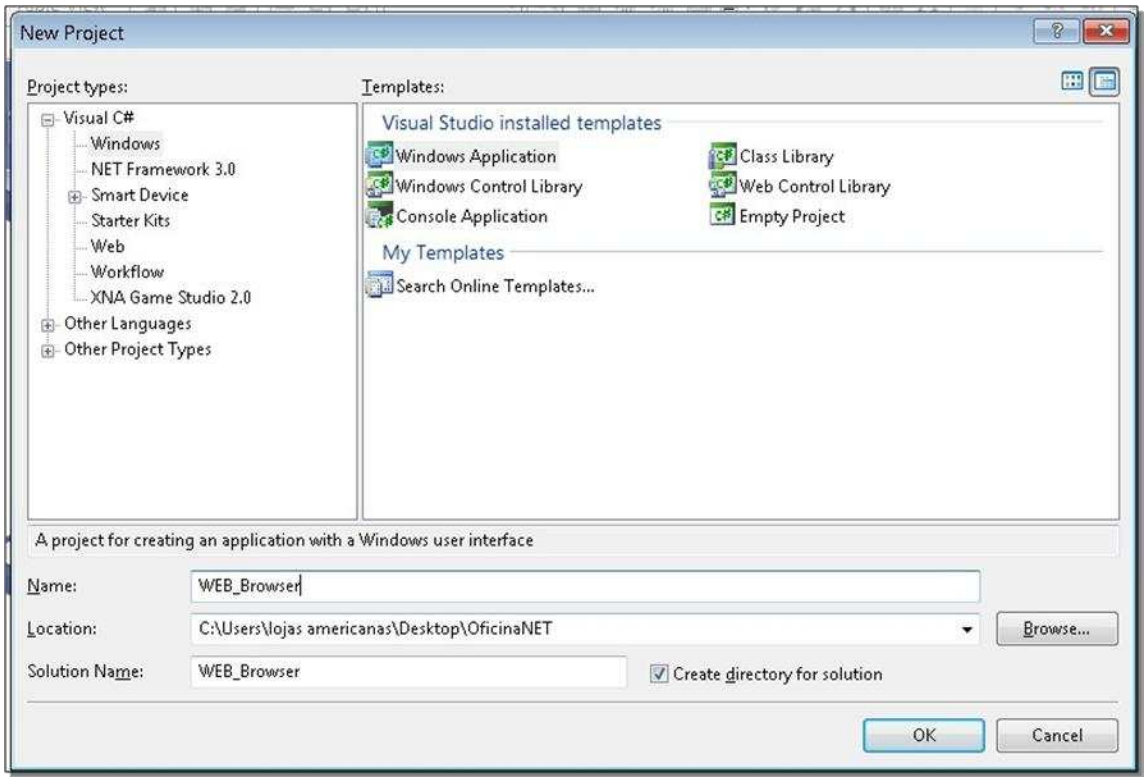


figura: criando uma aplicação windows.

A configuração do seu formulário principal deverá ter as seguintes propriedades: Name = WebBrowser, Net e escapi (UNIFIGSystems) como propriedade Text deste formulário. Se a caixa de ferramentas (Toolbox) não estiver visível, clique em View -> Toolbox e você terá à sua esquerda, ao lado do formulário, a caixa de ferramentas com os botões principais para você colocar no seu aplicativo. Procure o botão **WebBrowser**, deve se situar entre os últimos botões e arraste (drag and drop) ao formulário principal:

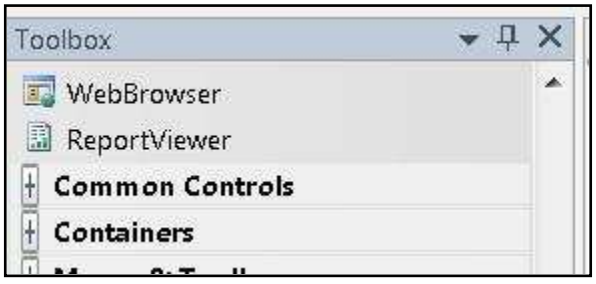


figura : controle WebBrowser

Ao ser arrastado para o formulário principal, obteremos:

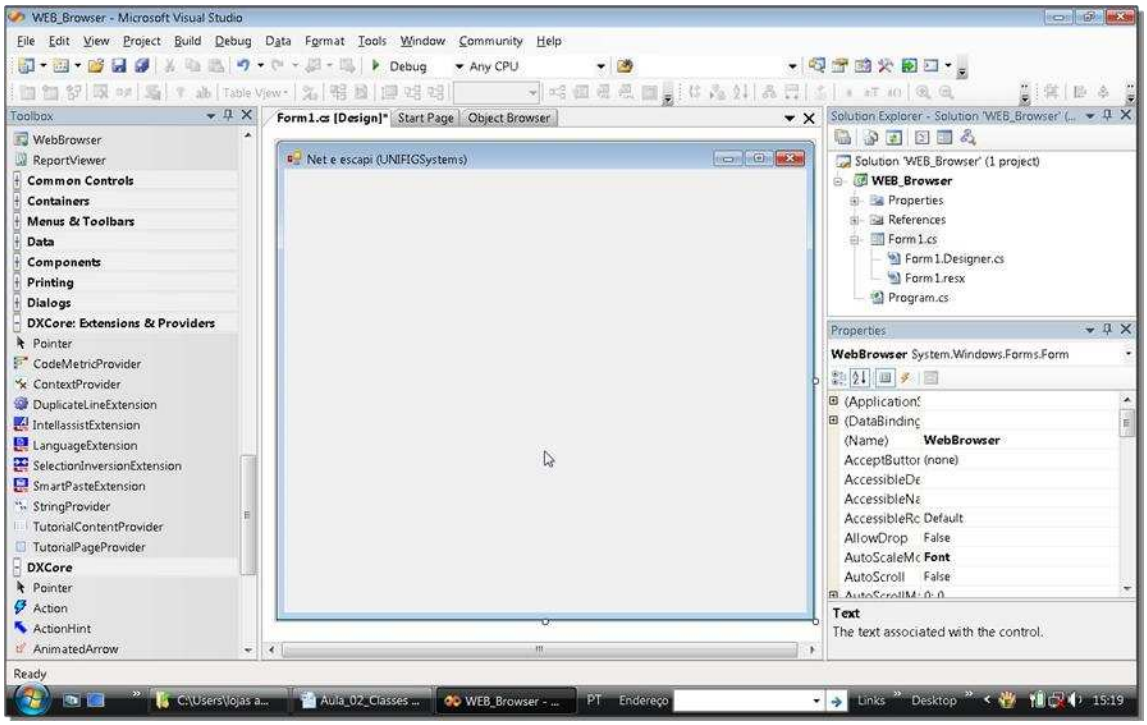


figura : arrastando o controle WebBrowser para o formulário recém criado.

Após você selecionar e arrastar o controle **WebBrowser** em direção ao formulário, este deverá ter o seguinte aspecto:

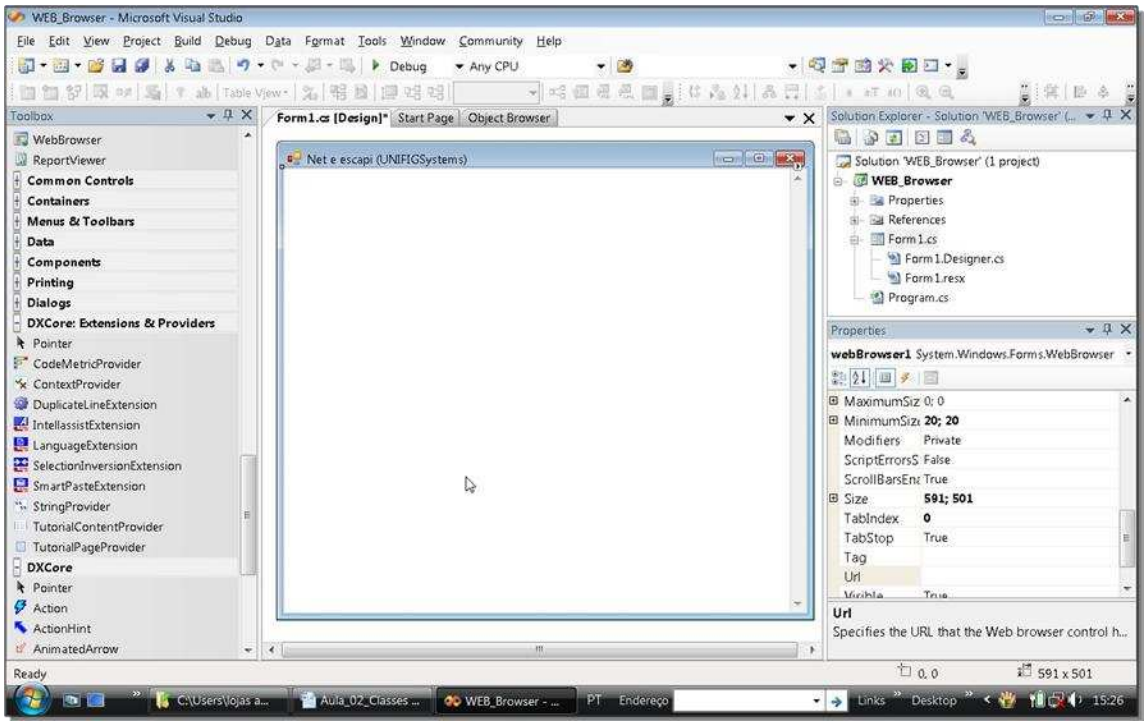


figura : o controle WebBrowser.

Aparece um menu suspenso onde se lê: **undock in parent container**. Clique neste item de menu. Realizando esta operação, você pode deixar espaço para mais dois controles em seu formulário: uma caixa de texto e um botão de comando. Veja como fica o aspecto de seu formulário:

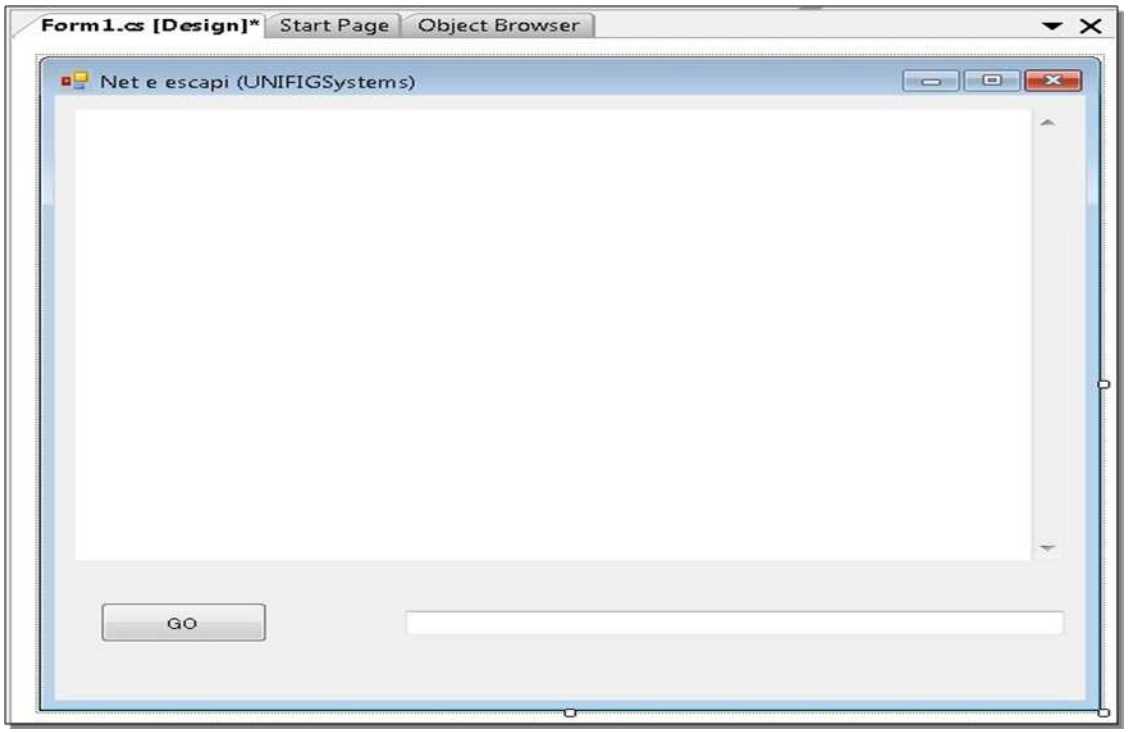


figura : inserindo botão e caixa de texto.

Lembre-se que o seu objeto tipo **webbrowser** leva o seguinte nome: webbrowser1. Agora dê duplo clique no botão de comando acima, que tem propriedade texto: GO.

Deve-se abrir um formulário para edição de código com o aspecto:

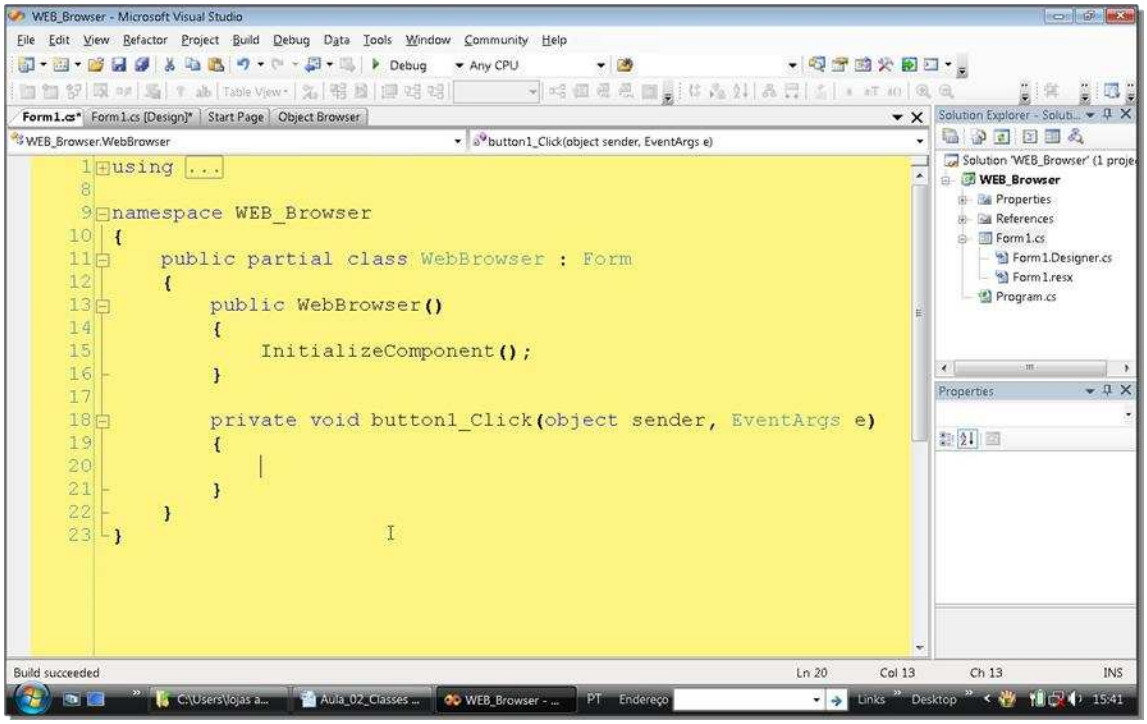


figura : O evento Click do objeto button1 da classe Button.

Quando você deu duplo clique no objeto visual botão de comando, com a propriedade texto GO, a IDE abre um formulário já pré-escrito onde se encontra algum código C# que define algumas propriedades importantes. Antes de escrevermos algum código C# para obtermos a funcionalidade da aplicação, vamos entender quem são estes códigos pré-definidos. O namespace WEB_Browser define o nome do projeto, que você configurou numa das primeiras etapas. Note que a solução é um conjunto de projetos, você possui aqui uma solução de mesmo nome que o projeto, observe a janela Project Explorer no canto superior à direita.

O arquivo de nome Form1.cs está marcado em cinza, pois é ele que está selecionado para edição, na porção central da tela. Note as abas, que proporcionam a seleção dos diversos arquivos da solução. Quando você deu duplo clique no botão de comando, a IDE criou para você um espaço de edição onde você pode escrever código C#.

Antes disso, vamos explicar os outros elementos fundamentais presentes no arquivo Form1.cs (a extensão .cs refere-se à linguagem C#).

Note a expressão: public partial class **WebBrowser: Form** na linha 11.

A expressão **public partial class WebBrowser: Form** indica várias coisas:

1) A classe principal do projeto tem nome **WebBrowser.** Ela contém uma classe herdada, ou seja, o conceito de herança aparece neste exemplo, denominada Form.

O operador : que separa o nome da classe pai e a classe herdada indica a relação de herança. Neste caso é herança simples.

O segmento de código:

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

denomina-se construtor da classe (no caso a classe WebBrowser). Por boas práticas de programação, a IDE já constrói para nós este segmento, com um construtor que leva o mesmo nome da classe-pai. O método **InitializeComponent**( ) chama todos os recursos para a construção do aplicativo Windows deste tipo.

Em seguida, como resultado da ação duplo clique sobre o botão button1 (este tipo de nome não é padrão, deve-se por boa prática escrever algo do tipo btnGO, o prefixo btn indica que o objeto é tipo botão de comando, mas não mudamos tudo aqui por questão de escopo) abre-se o seguinte bloco de código:

```
private void button1_Click(object sender, EventArgs e)
{
   //código a ser acrescentado
}
```

private significa que o método executado pelo botão button1 é privado, isto é, as suas ações estão disponíveis apenas para o formulário Form1. Toda e qualquer variável local declarada dentro deste escopo será considerada variável ou campo privado devido ao modificador de escopo private.

Nenhum outro formulário novo, criado neste projeto terá acesso às ações (ou será influenciado) definidas pelo método privado deste botão. Para que as suas ações possam influenciar outras classes e objetos em outros formulários é só trocarmos private por public manualmente. Podemos fazê-lo neste exemplo sem qualquer problema. Como nossa aplicação contém um formulário e uma classe-pai, não precisamos nos preocupar em modificar o tipo de acesso aos dados para público.

Por default, todo botão de comando criado por drag and drop nos formulários de aplicação Windows terão modificador de acesso **private**, isto é, são locais em termos de acesso. void significa que a ação do botão não retorna qualquer valor para uso pela aplicação principal. Em seguida aparece **button1_Click(...).**

Por **button1** quer dizer o nome do botão (este tipo de nome não é padrão como explicamos acima). _Click quer dizer que selecionamos o tipo de evento **Click** o qual é acessado via clique simples do usuário da aplicação.

Esta ação só se dá em **runtime** (tempo de execução do aplicativo).

A lista de parâmetros identifica os itens: ...(object sender, EventArgs e).

Por **object sender** entendemos um objeto que dispara o evento chamado, e e em EventArgs apresenta lista de parâmetros adicionais, passando parâmetros de volta ao código quando há mais opções disponíveis (itens de lista, por exemplo).

Agora, estamos em condições de escrever algum código para o evento deste botão, escreva o seguinte: **webBrowser.Navigate(textBox1.Text);**

O código deste botão deverá ter o seguinte aspecto:

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

    private void button1_Click(object sender, EventArgs e)
    {
        //código a ser acrescentado.
        webBrowser1.Navigate(textBox1.Text);
    }
}
```

figura : código de button1_Click(...)

Quando você termina a digitação de **webBrowser1**, aparece o Intellisense com uma lista de opções disponíveis. Navegue até a opção Navigate e clique **ENTER** e em seguida abra parênteses. Dentro dos parênteses você escreve textBox1 (o nome padrão deveria ser algo tipo txt...) com a terminação de propriedade Text.

O método Navigate( ) vai executar a operação principal de todo browser para a Internet (o **Internet Explorer** não é exceção): Ele captura o argumento escrito (URL) na caixa de texto que está ao lado do botão, e o aplicativo procura a página WEB correspondente.

Pronto, com poucas linhas, a plataforma .NET nos disponibiliza um browser de nossa autoria! Essencialmente, podemos colocar outros botões neste aplicativo para salvarmos as páginas visitadas, etc e tal, com poucas linhas a mais de código. Não é do nosso escopo ou interesse aqui terminarmos um InternetExplorer completo, isto levaria em

conta dezenas de horas de escrita de código e muita codificação, além do que já temos à nossa disposição o IExplorer e outros aplicativos.

Agora, se é claro que você está com problemas com seu IExplorer (coisa não muito incomum) e não consegue re-instalá-lo e precisa de visitar uma página WEB naquele fim de noite (contando que seu modem ou conexão wireless estão OK), você pode escrever este simples aplicativo, depurá-lo e executá-lo e não ficará sem navegar na rede pelo menos enquanto não consegue depurar o Internet Explorer.

E tudo isto numa aplicação que tomará de você 10 minutos de desenvolvimento, no máximo! Vamos executar este aplicativo:

figura : Digitando a URL xxx.lanl.gov chegamos à página acima.

Diagrama de Classes

Para finalizarmos a nossa discussão sobre o simples aplicativo que acima foi construído, vamos mostrar uma ferramenta muito interessante para o desenvolvedor, disponibilizada apenas nas versões Standard do Visual Studio 2005 a 2008 (as versões Express não

possuem esta ferramenta): o **Diagrama de Classes.** Após você ter executado a aplicação acima, volte ao Visual Studio 2005 (ou 2008) e abra o **Project Explorer:**

Dê clique simples com o botão direito do mouse sobre **WEB_Browser.** Aparece um menu suspenso, vá até a guia View Class Diagram.

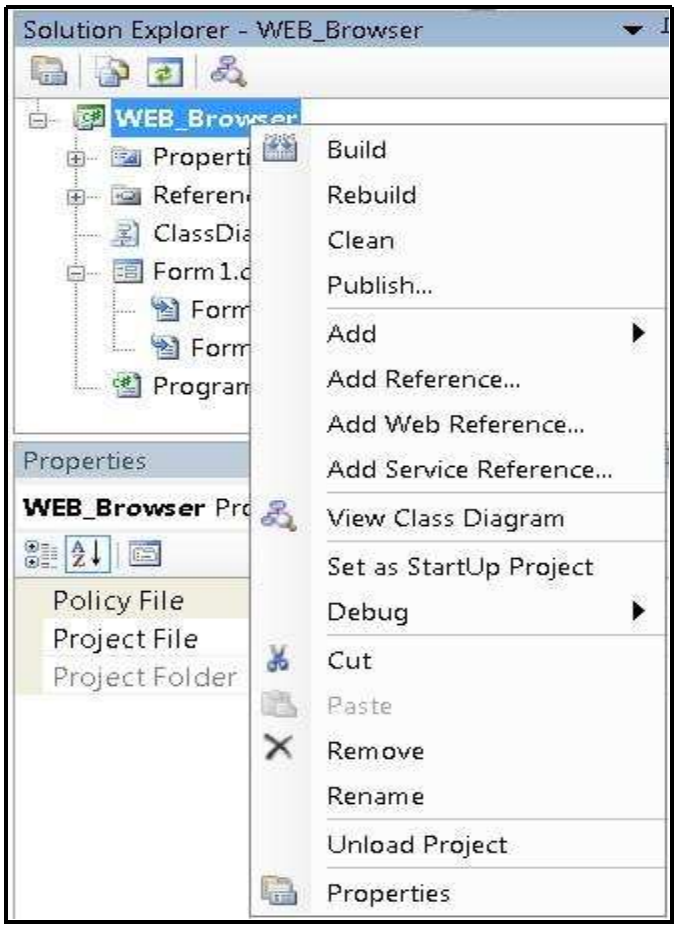


figura : criando o diagrama de Classes.

Ao clicar em **View -> Class Diagram,** a IDE constrói um arquivo com formato .cd (class diagram, evidentemente):

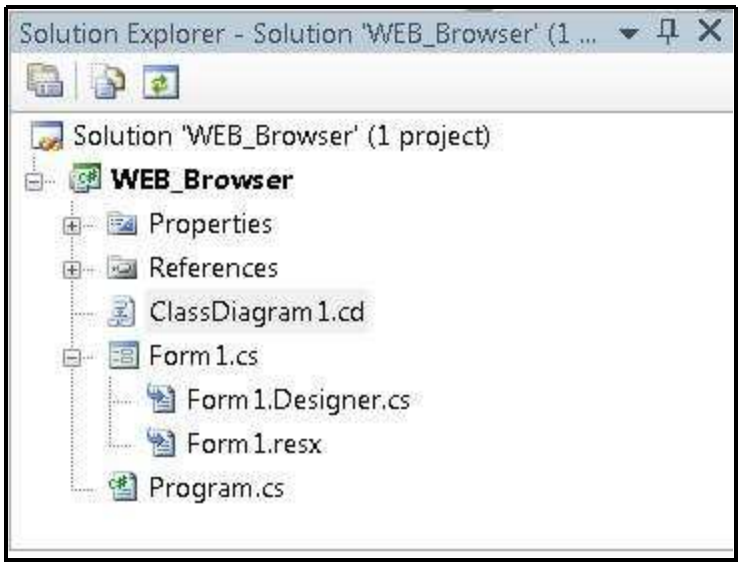


figura : arquivo ClassDiagram.cd criado.

O **Diagrama de Classes** permite que você tenha uma visão completa das classes do seu aplicativo e mostra a relação de herança entre classes através de setas. Além disso, na janela **Class Details,** você pode incluir em tempo de depuração algum novo campo ou item, basta selecionar alguma estrutura e escolher o nome do novo campo.

Ao clicar em **Build**, a IDE automaticamente acrescenta o novo item à classe em consideração. Com o **Class Diagram** você obtém as informações essenciais do projeto como um todo.

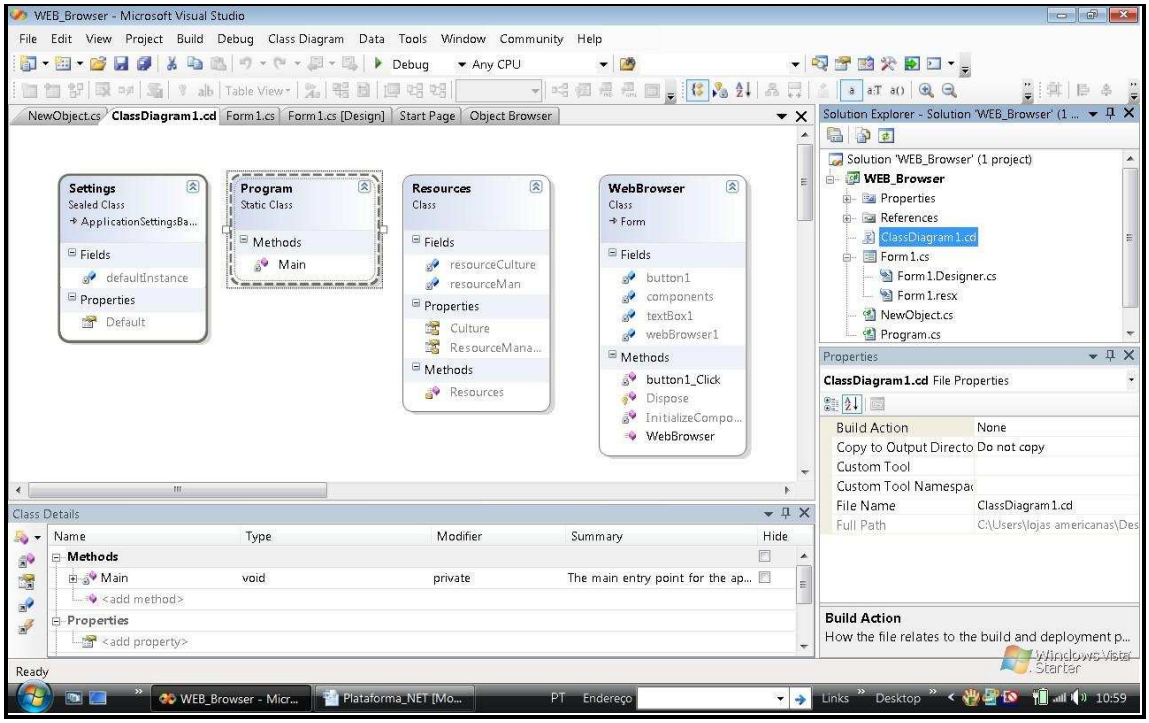


figura : as classes do programa WebBrowser.

Veja um zoom de **ClassDiagram.cd:**

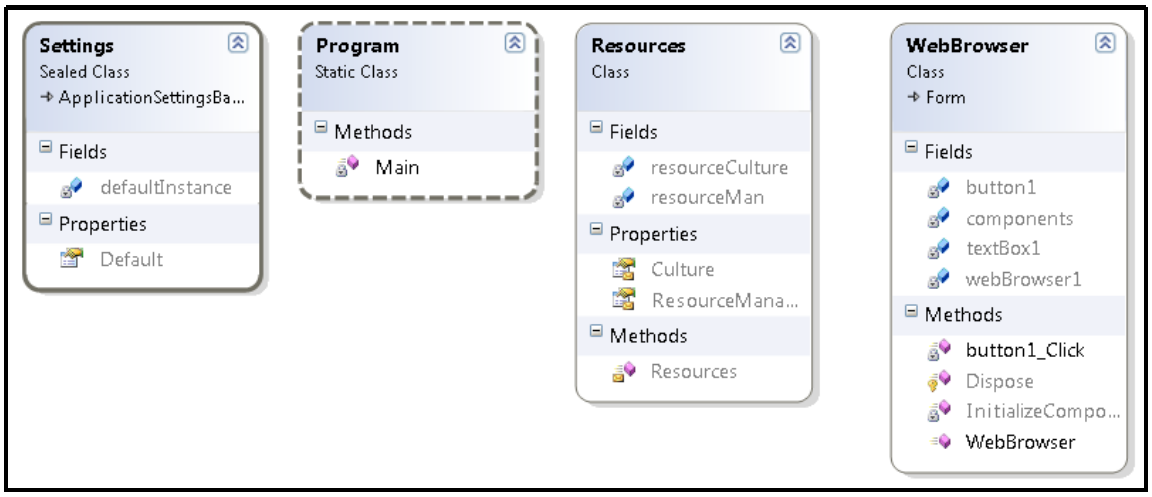


figura : zoom do diagrama de classes.

Há quatro membros dos diagramas a serem detalhados aqui:

1) **Settings** estabelece algumas configurações que a IDE faz compartilhar entre o aplicativo e o sistema.

2) **Program** é a classe estática que possui o método principal, Main, o qual chama a instância de todos os demais objetos e recursos. O seu código já foi gerado automaticamente pela IDE, mas você pode realizar acréscimos manuais se desejar.

O seu código é:

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;

namespace WEB_Browser
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new WebBrowser());
        }
    }
```

}

Se o seu programa deve apresentar uma tela de splash (em tempo de execução) antes do aparecimento de WebBrowser, você pode acrescentar a linha de código: Application.Run(new _nomeObjeto), onde _nomeObjeto é o nome de um novo formulário do seu projeto. Esta linha pode ser acrescentada acima da linha Application.Run(new WebBrowser) se você deseja que o novo formulário apareça antes ou embaixo desta linha se você deseja que ele apareça depois do objeto WebBrowser. As outras linhas acima capacitam que o Windows use os recursos gráficos para apresentar os formulários (janelas) do seu programa.

3) **Resources** é uma tabela dos diagramas que está relacionada aos recursos que o aplicativo consome do Windows para o funcionemento do aplicativo.

4) **WebBrowser** é o último diagrama e mais interessante para nós. Veja que ele possui quatro campos (fields): button1, components, webBrowser1e textBox1. Estes são os objetos de aparência gráfica que você dispõe em tempo de design. Components está relacionado à janela de suporte onde você arrastou um botão (da classe Button), uma caixa de texto (da classe TextBox) e o objeto webBrowser1 que carrega a página da internet visitada.

Os ícones que representam os métodos deste diagrama estão inicialmente desenhados com um pequeno cadeado à sua esquerda. Este cadeado indica que o método é private. Você pode mudar o tipo de acesso agora nos diagramas ao invés de fazê-lo no editor de código!

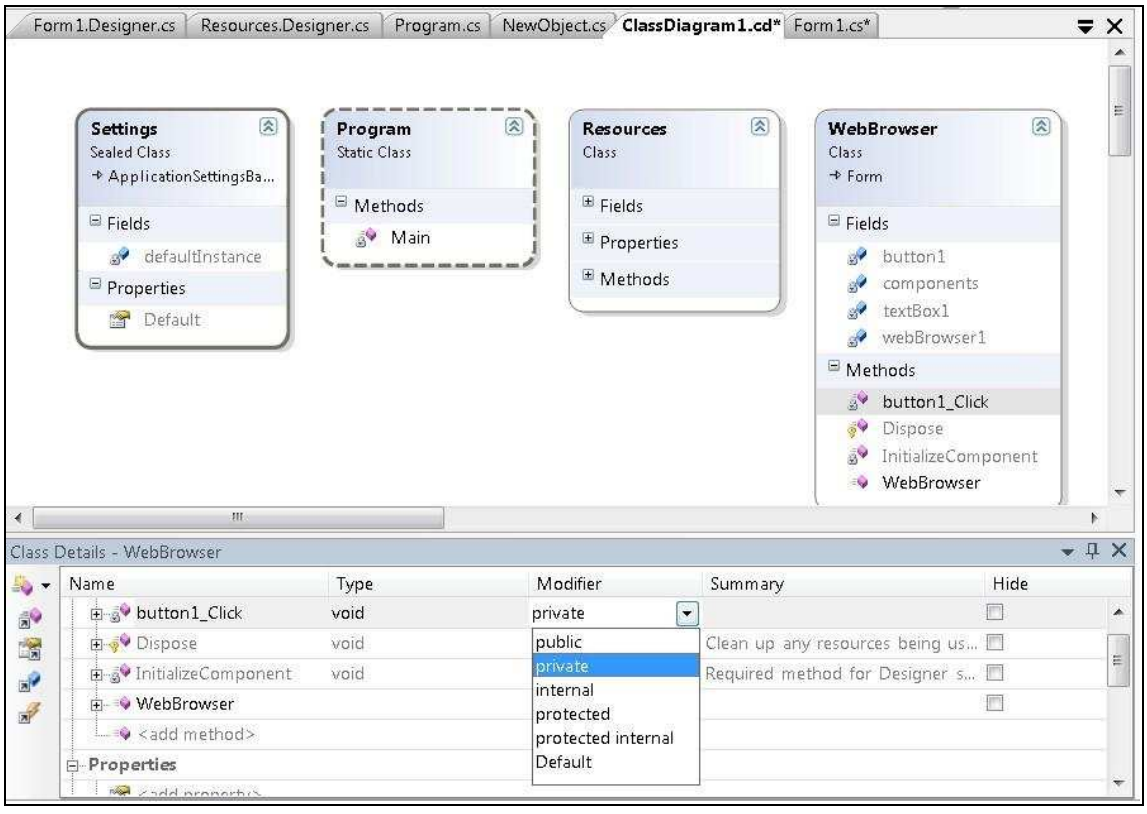


figura : A janela Class Details.

Na aba deslizante de **button1_Click** aparece um leque de opções à nossa disposição: public, private, etc. Se você clicar em public o acesso é modificado e o cadeado desaparece, veja na figura abaixo ao executarmos a ação:

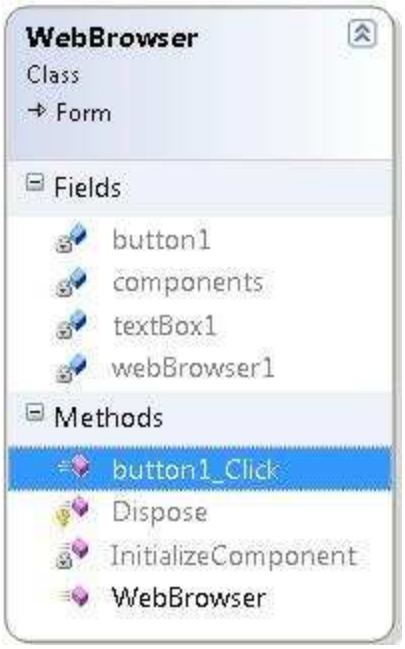


figura 34: método button1_Click( )

Para terminarmos esta discussão de diagrama de classes, vou mostrar um exemplo em que os diagramas de classe mostram a herança (não vou apresentar o código-fonte deste exemplo para não fugirmos do escopo de nossa apresentação, cujo objetivo é apresentar um panorama das ferramentas e métodos do Visual Studio como principal IDE para o desenvolvimento de aplicativos .NET):

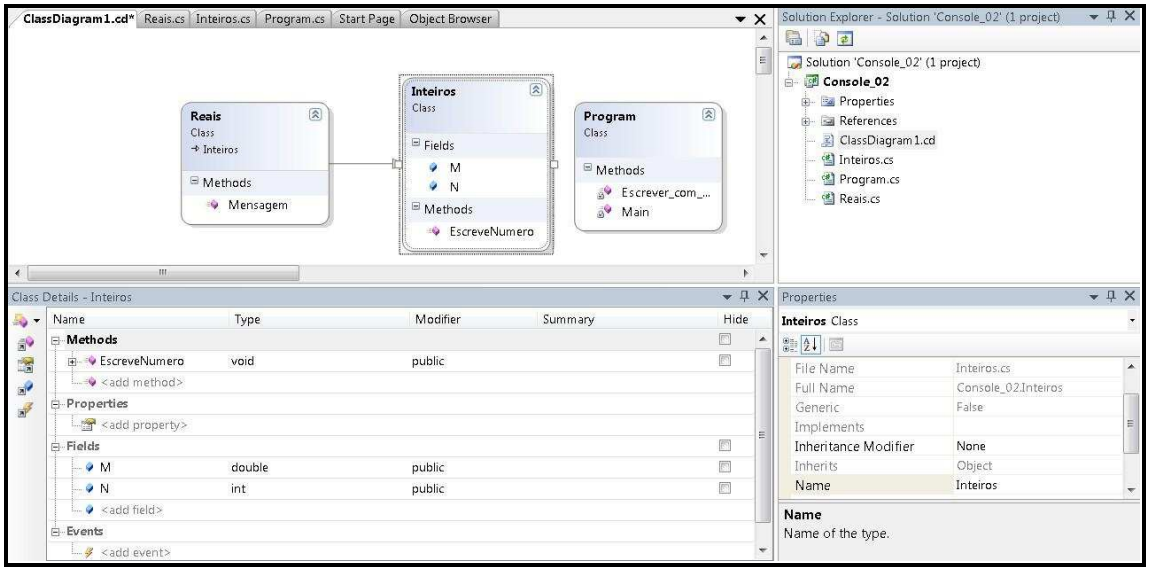


figura : diagrama mostrando a herança entre as classes.

A classe **Reais.cs** herda propriedades e métodos da classe Inteiros.cs, isto é, se você declarar por referência um objeto desta classe, em Program.cs, você pode aplicar os métodos da classe Inteiros.cs sobre os objetos da classe Reais.cs. Note que o diagrama de Classes monta uma seta indo da subclasse Reais.cs para a classebase Inteiros.cs (depois que construímos as classes e operarmos **Build**).

Para finalizar, em **Class Details** – Inteiros, note que esta janela mostra a lista de campos da classe Inteiros: M e N, logo abaixo há uma guia adicional denotada <add field>. Esta guia permite que você adicione mais um campo à esta classe.

E na guia **Modifier** você modifica o tipo de acesso do campo recém acrescentado, e tudo isto sem que você precise ir ao editor de código da classe Inteiros.cs!

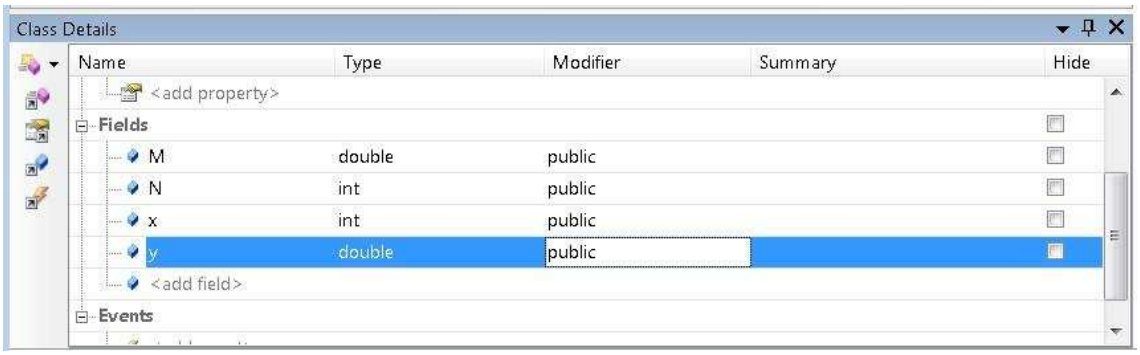


figura : adição manual de campos fora da edição de código.

Após acrescentarmos dois novos campos utilizando a janela **Class Details**, clicamos em **Build** e pronto, quando abrimos o editor de código da classe Inteiros encontramos a declaração das novas variáveis x e y (que são tecnicamente classificadas com campos, isto é, fields). Manualmente modificamos o seu acesso para **public** como pode ser visto na figura acima.

Agora, vamos criar **Herança** usando a **Toolbox** do **Diagrama de Classes.**

Considere novamente o projeto Carros e Utilitários. Abra o **Class Diagram** daquele projeto. Considere a que classe **Utilitários** ainda não herda de nenhuma outra classe. Clique em View a abra a guia Toolbox: a seguinte área deve aparecer do lado direito do diagrama:

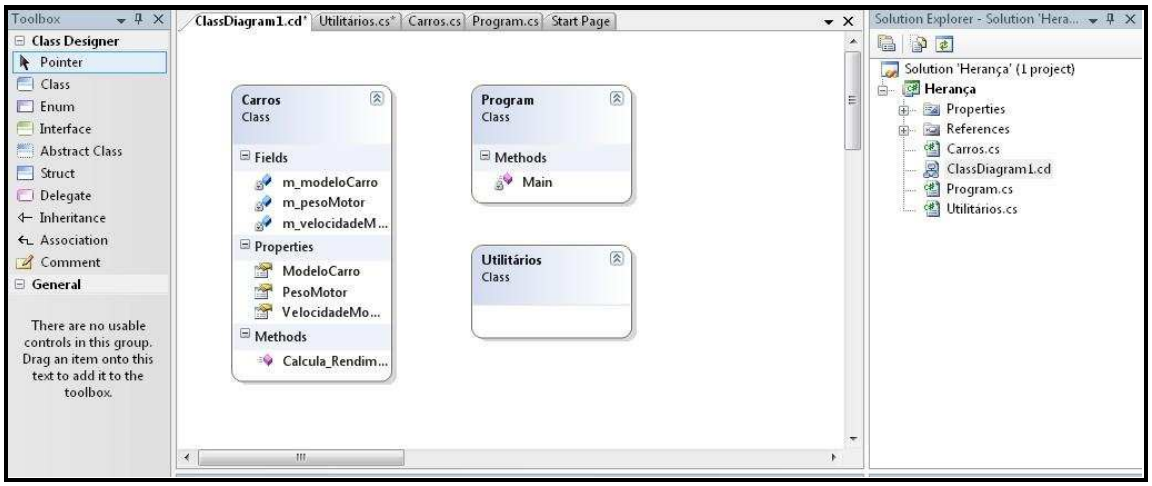
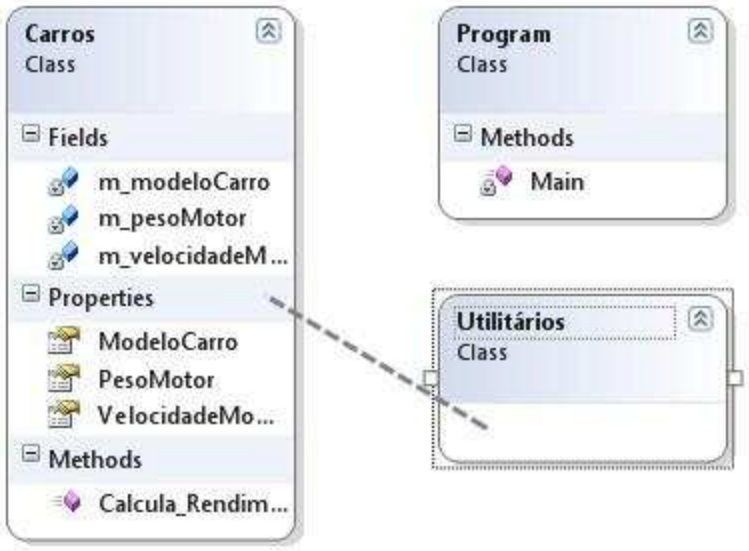


figura : a toolbox (caixa de ferramentas) do diagrama de classes

Agora, da toolbox selecione a seta Inheritance, e arraste a seta para cima da caixa Utilitários.



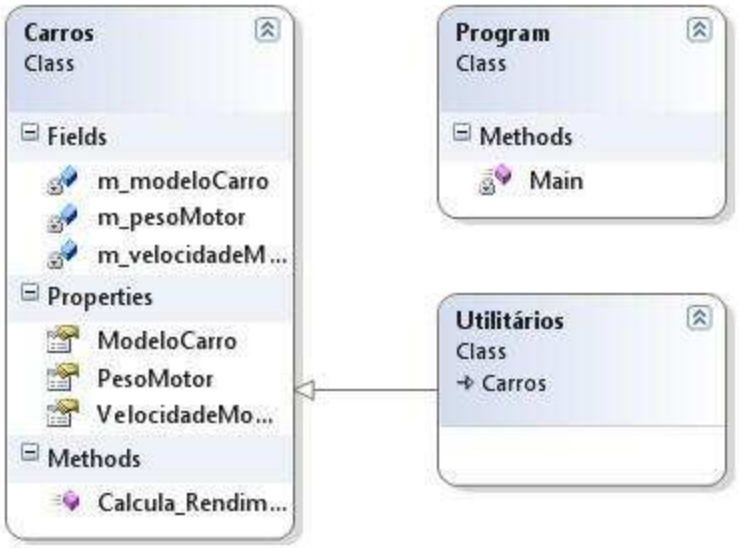


figura: implementando visualmente a herança.

Note que uma seta aparece ligando as duas classes: Carros.cs e Utilitários.cs, esta será a classe derivada e a primeira a classe base. Utilitários herda campos, propriedades e métodos da classe Carros.cs. Agora, clique no formulário Utilitários.cs para abrirmos o seu código:

```
namespace Herança
{
    class Utilitários : Carros
    {


    }
}
```

Note que a expressão : Carros aparece automático no código! O diagrama de classes se ocupa da implementação em tempo de projeto para os nossos programas.

Agora, vamos mostrar como implementar métodos e campos visualmente. Considere a janela **Class Details,** no Class Diagram:

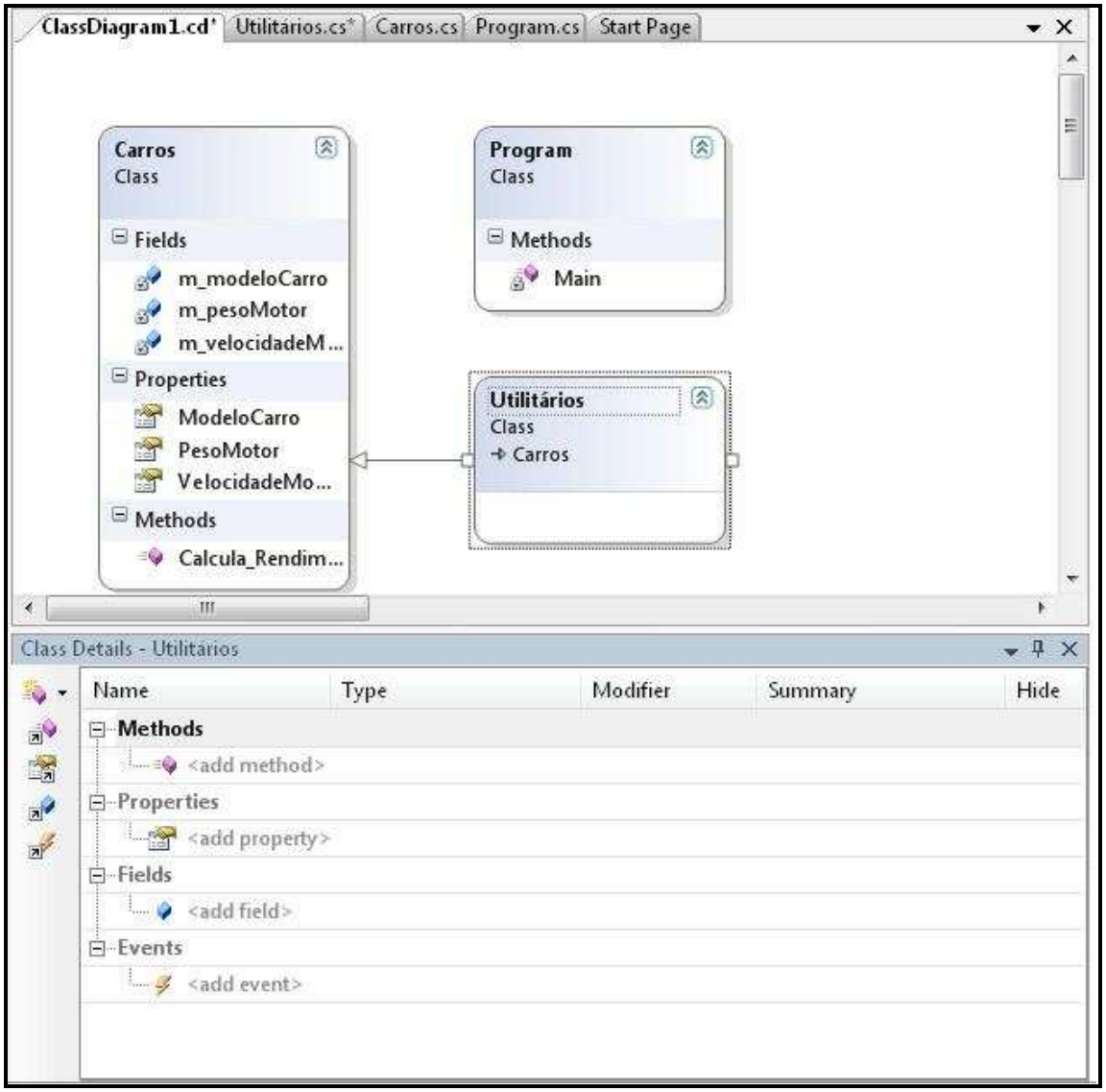


figura: visualizando a janela Class Details.

Selecione o diagrama Utilitários.cs, na janela Class Details aparecem os métodos, propriedades, campos e eventos desta classe. Como vimos, não há nada escrito no código da classe, portanto, add method, add property, add field e add event estão vazios, evidentemente.

Selecione **add method** e escreva um nome de método: Calcular, configure os outros campos da figura de acordo com a figura abaixo:

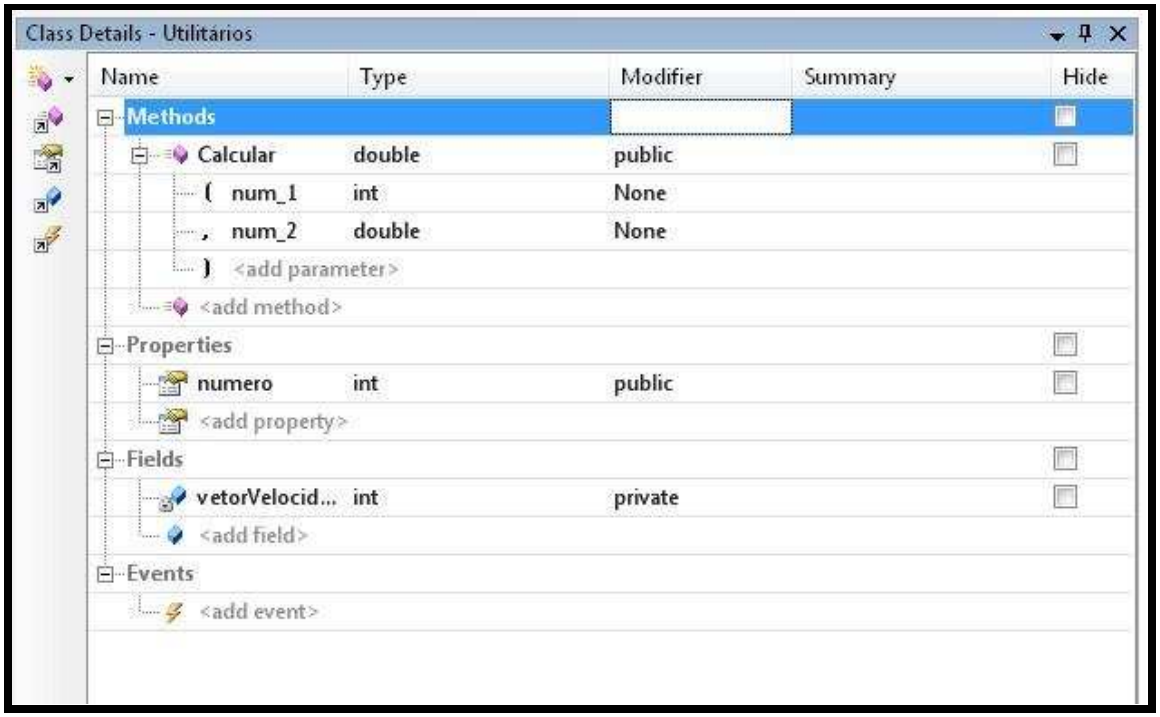


figura: configurando visualmente métodos, parâmetros e demais propriedades.

Agora, clique no diagrama da classe **Utilitários** e veja o código que o **Class Diagram** gerou para você:

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

namespace Herança
{
    class Utilitários : Carros
    {
        private int vetorVelocidade;

        public int numero
        {
          get
          {
           throw new System.NotImplementedException();
          }
          set
          {

          }
        }
```

```
    public double Calcular(int num_1, double num_2)
    {
        throw new System.NotImplementedException();
    }

  }
}
```

Veja que todo o código acima foi gerado visualmente no Diagrama de Classes da sua aplicação. Você também pode exportar a figura do **Diagrama de Classes** com o objetivo de documentação.

Finalmente, para terminarmos a nossa jornada C#, vamos criar uma conexão com o **Banco de Dados Northwind**, disponível gratuitamente na MSDN como parte do SQL Server.

3) Aplicativos Windows para Bancos de Dados

Introdução

Sistemas **Gerenciadores de Bancos de Dados** ou **SGBDs**, são aplicações que lidam com armazenamento local ou remoto de dados, permitindo a sua modelagem de dados, persistência e segurança, entre outras funções de segurança e manutenção.

O SQL Server 2005 ou superior, é um SGBD proprietário, da Microsoft, trazendo inúmeras vantagens sobre outros sistemas do mercado. No site do MSDN pode ser baixado o Banco de Dados Northwind, que é gratuito e pode ser usado para fins acadêmicos.

As aplicações Windows Forms podem se comunicar com bancos de dados, sejam do SQL Server, Acess do pacote Office, etc. Tais aplicações permitem alterar campos e registros e reatualizar a fonte de dados local ou remota.

Além disso, o incrivelmente versátil controle DataGridView contém mais de uma centena de propriedades e métodos, podendo ser usado para realizar cálculos com os dados de um banco de dados ou funcionar como uma espécie de planilha local para ser preenchida manualmente. Nesta pequena apostila, vamos estudar apenas como realizar a conexão com o Banco de Dados NorthWind.

Crie um novo projeto tipo **Windows** e denomine **Banco_Dados_Northwind**.

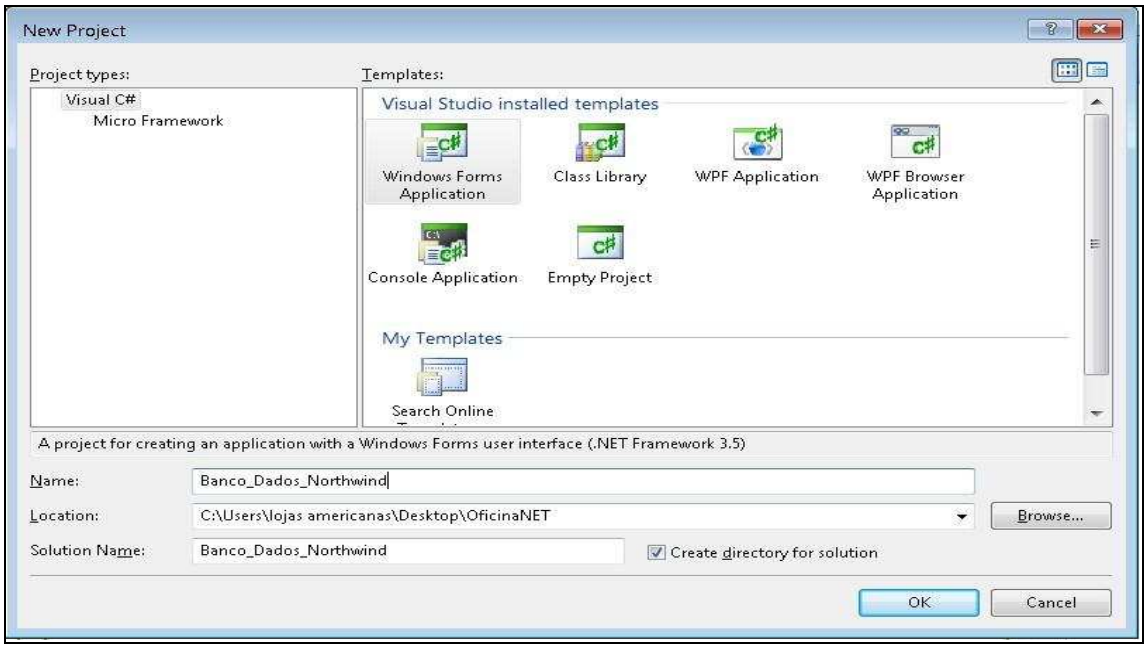


figura 37: criando um aplicativo de banco de dados.

O nosso formulário deve ser:

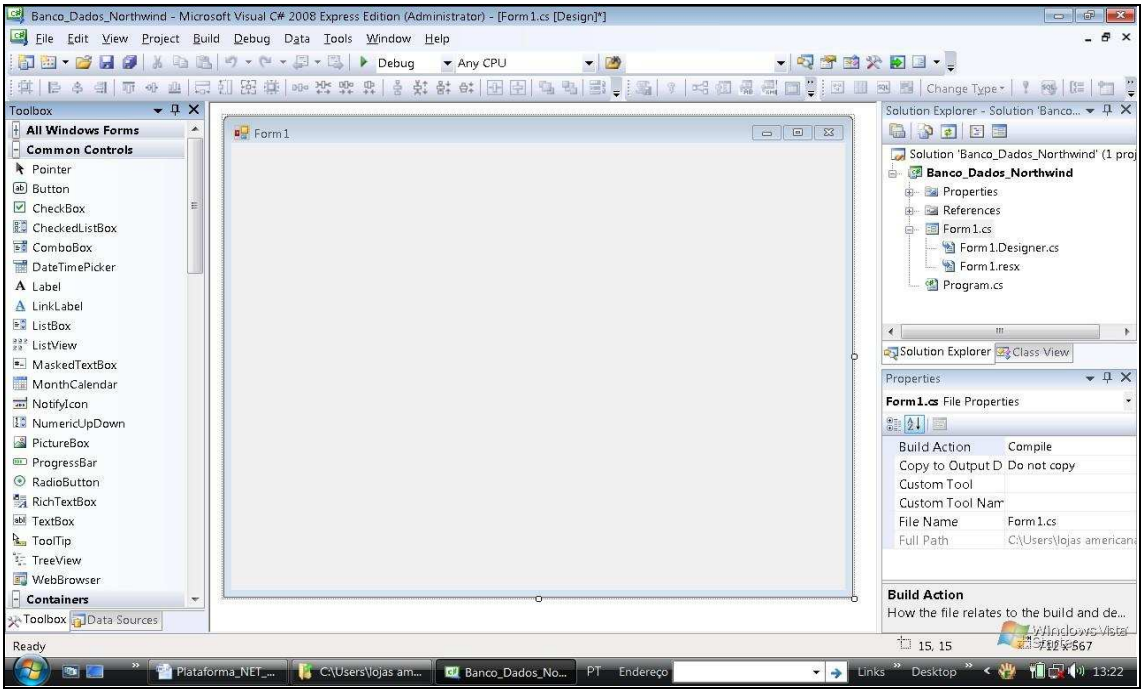


figura : o aplicativo visual Windows Banco_Dados_Northwind

Este **Banco de Dados** pode ser adquirido na Internet pelo MSDN, é um banco de dados gratuito, uma empresa fictícia, e este banco de dados pode ser usado e alterado para fins acadêmicos. Não vamos criar um aplicativo para banco de dados completo (DBA) pois isto exigiria um curso completo, com desenvolvimento a camadas, criptografia e segurança, vamos no entanto montar um pequeno aplicativo que acessa a conexão de dados deste banco e ver como é fácil inciar um aplicativo comercial com o Visual Studio! Selecione o item de menu **Data** e clique **Add New Data Source**:

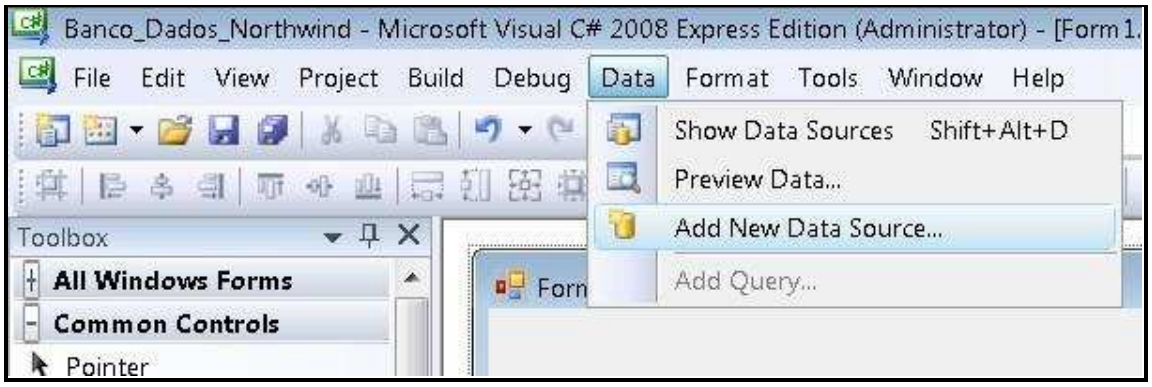
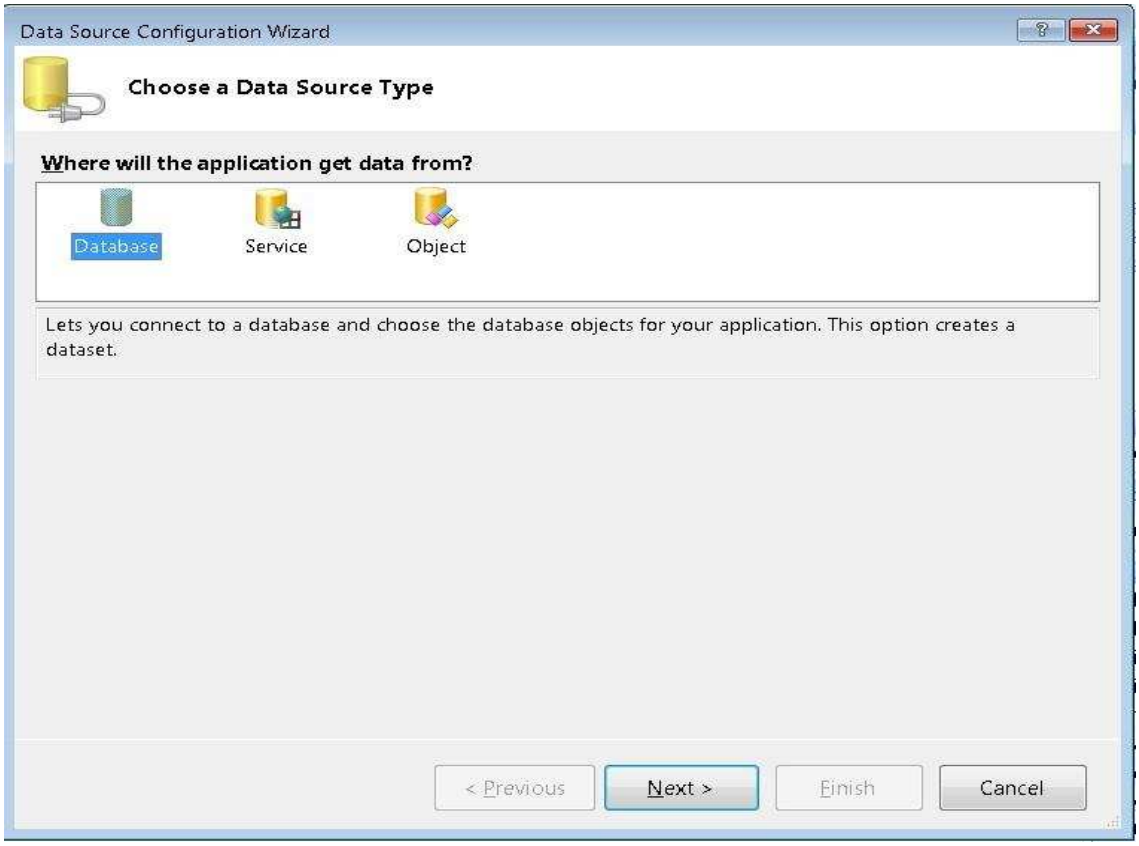


figura : adicionando uma nova fonte de dados.



figuras -: criando uma fonte para o banco de dados.

Clique Next >

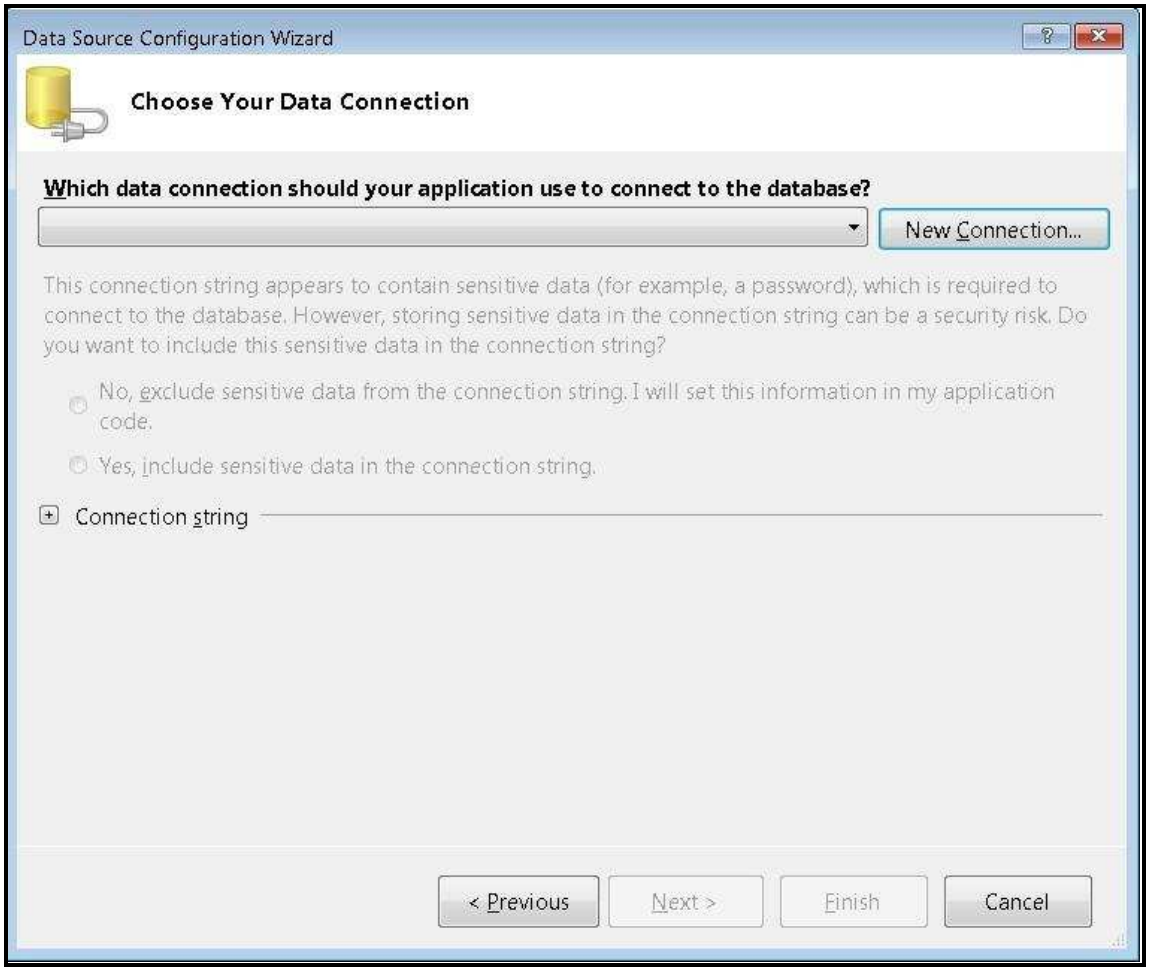


figura : escolha da conexão.

Clique **New Connection** e abre-se a seguinte caixa de diálogo:

figura : procurando a conexão de dados.

Clique **Browse** ... para buscarmos o banco de dados para a nossa aplicação.

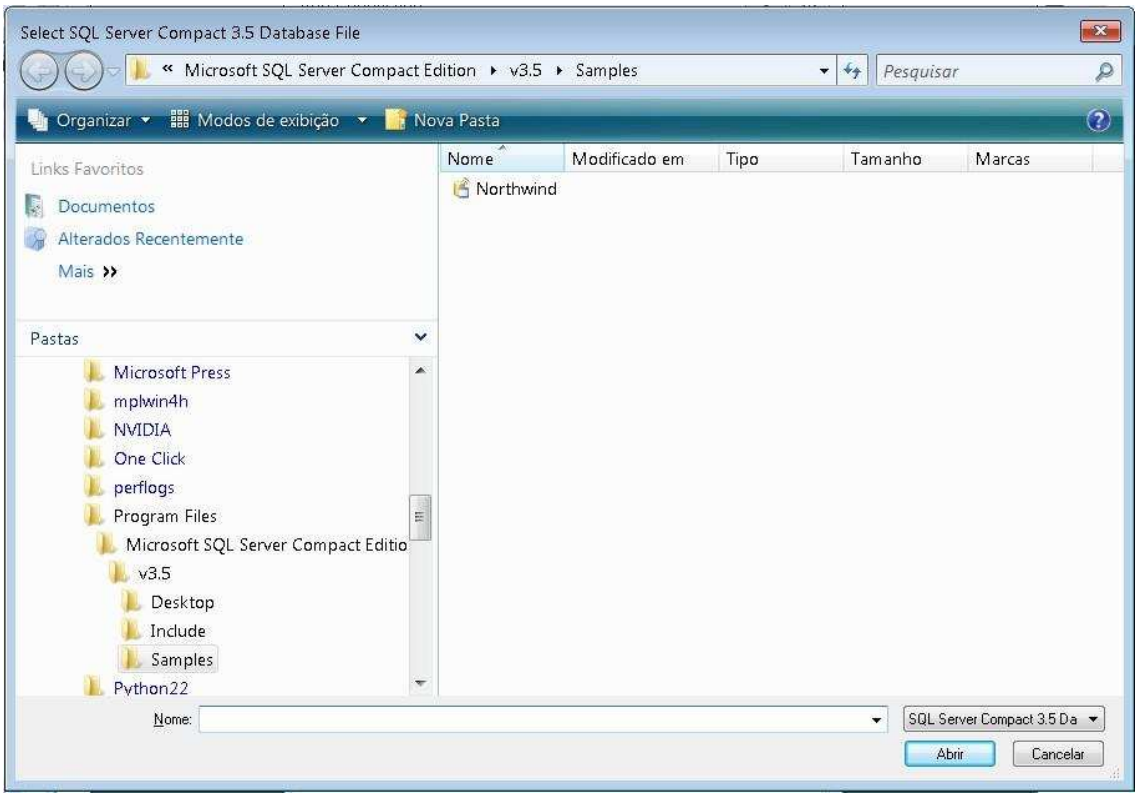


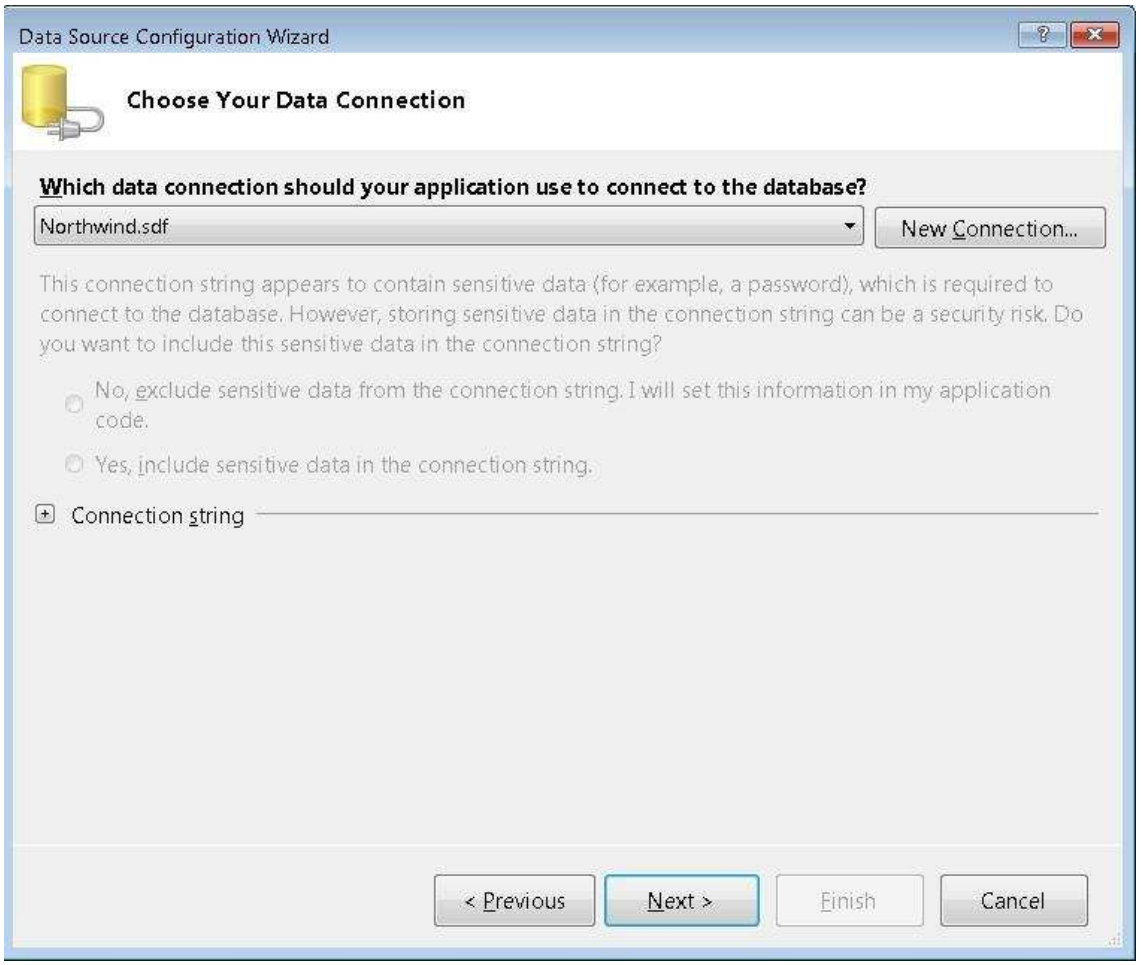
figura : encontrando o **Northwind**, banco de dados

A IDE já encontra o Banco de Dados **Northwind** na pasta apropriada de Microsoft SQL Server Compact Edition em samples: Clique em **Northwind** e em abrir, o que gera:

figura 44: criando a conexão.

Dentro da figura 42 acima! Clique -> **Test Connection** para testarmos se há conexão do aplicativo com o **Banco de Dados:**

A conexão aparece agora:



figuras 45 e 46: terminando a configuração de string de conexão.

Quando você clica em **Next** a IDE pergunta se você deseja copiar o arquivo em seu projeto e modificar a conexão.

Clique em **Sim:**

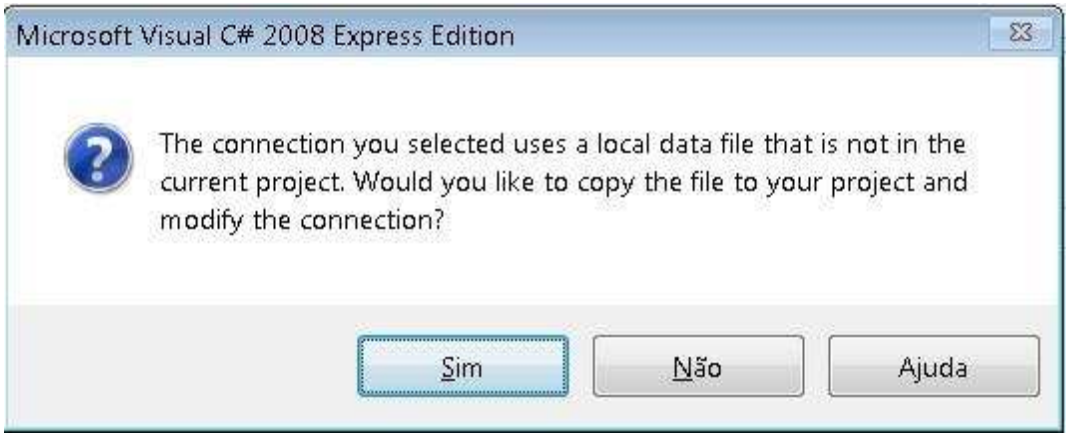


figura 47: confirmando a transferência de arquivos para o aplicativo.

A caixa de diálogo mostra a conexão criada, clique em **Next >:**

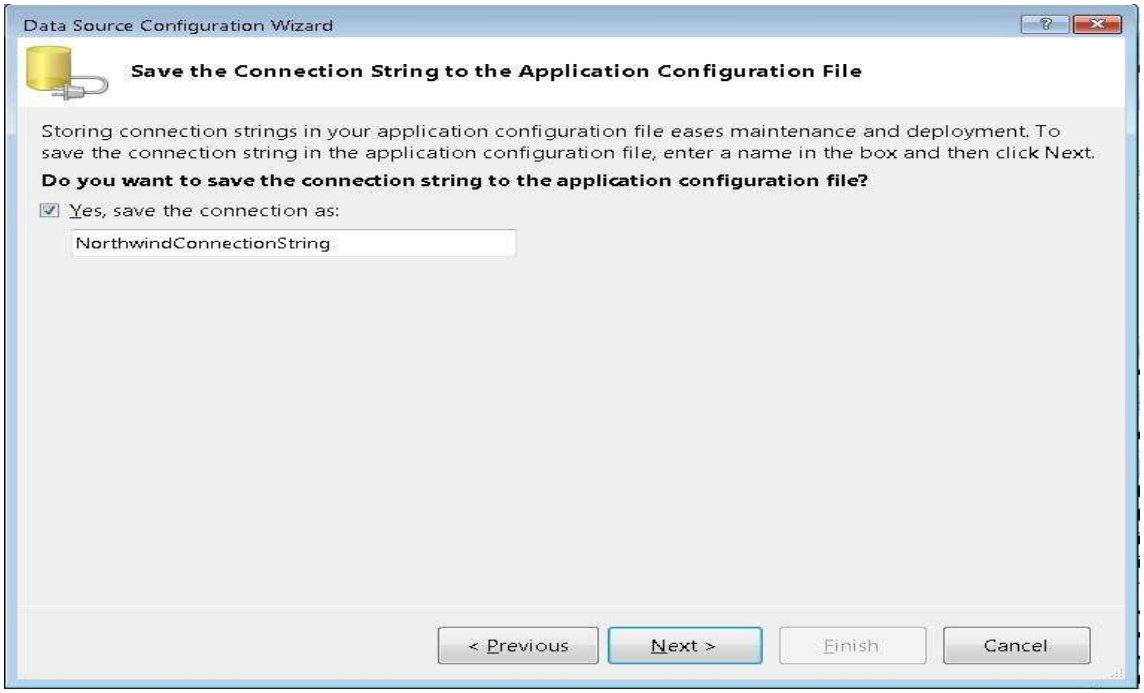


figura 48: salvando a string de conexão.

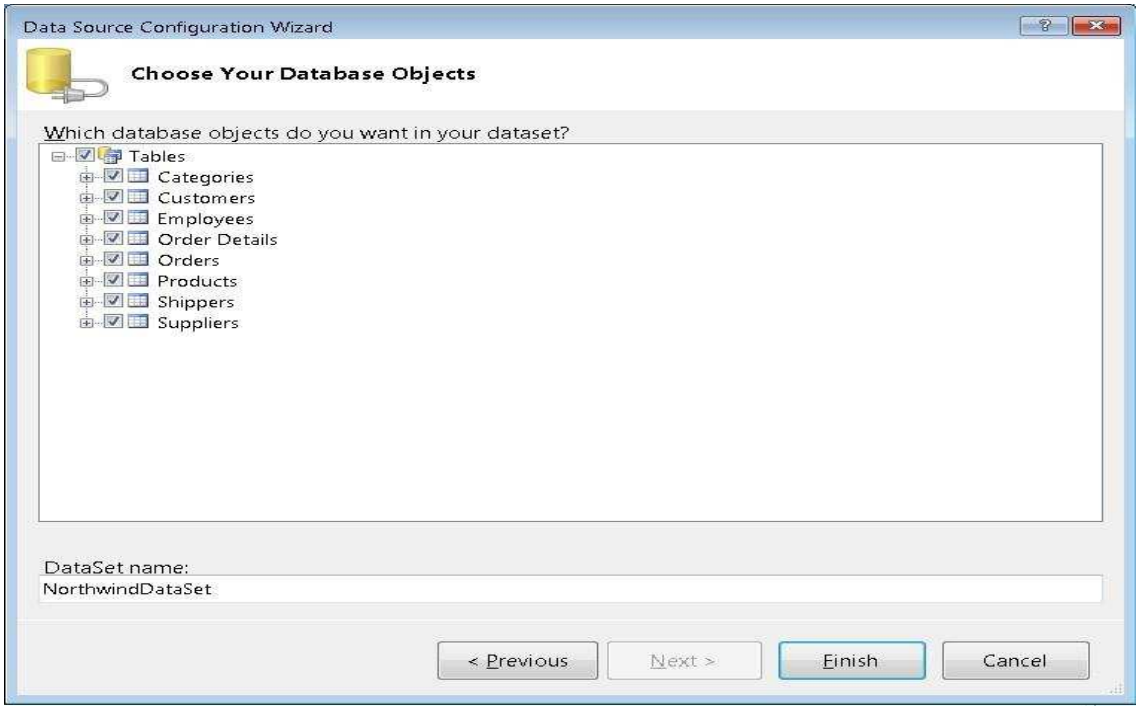


figura 49: selecionando as tabelas do Banco de Dados.

E, clique Finish: O **solution explorer** agora mostra os arquivos relacionados ao banco de dados Northwind, que foram acrescentados à aplicação:

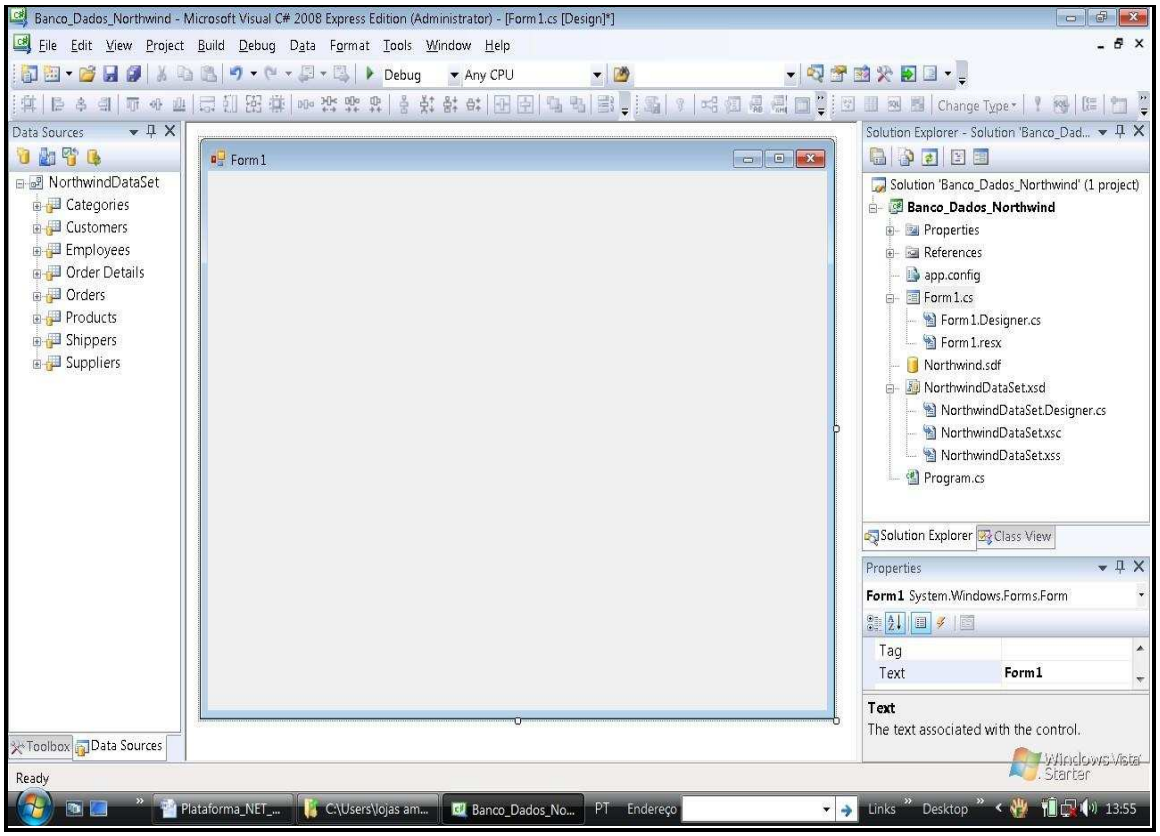


figura 50: a solução com os arquivos do BD prontos p/ serem usados no aplicativo.

No lado esquerdo, basta selecionarmos com o mouse as tabelas que desejamos e acrescentarmos ao formulário. Temos à nossa disposição dois modelos de apresentação: tipo **DataGridView** (objeto da classe **DataGridView**) e tipo **Details**:

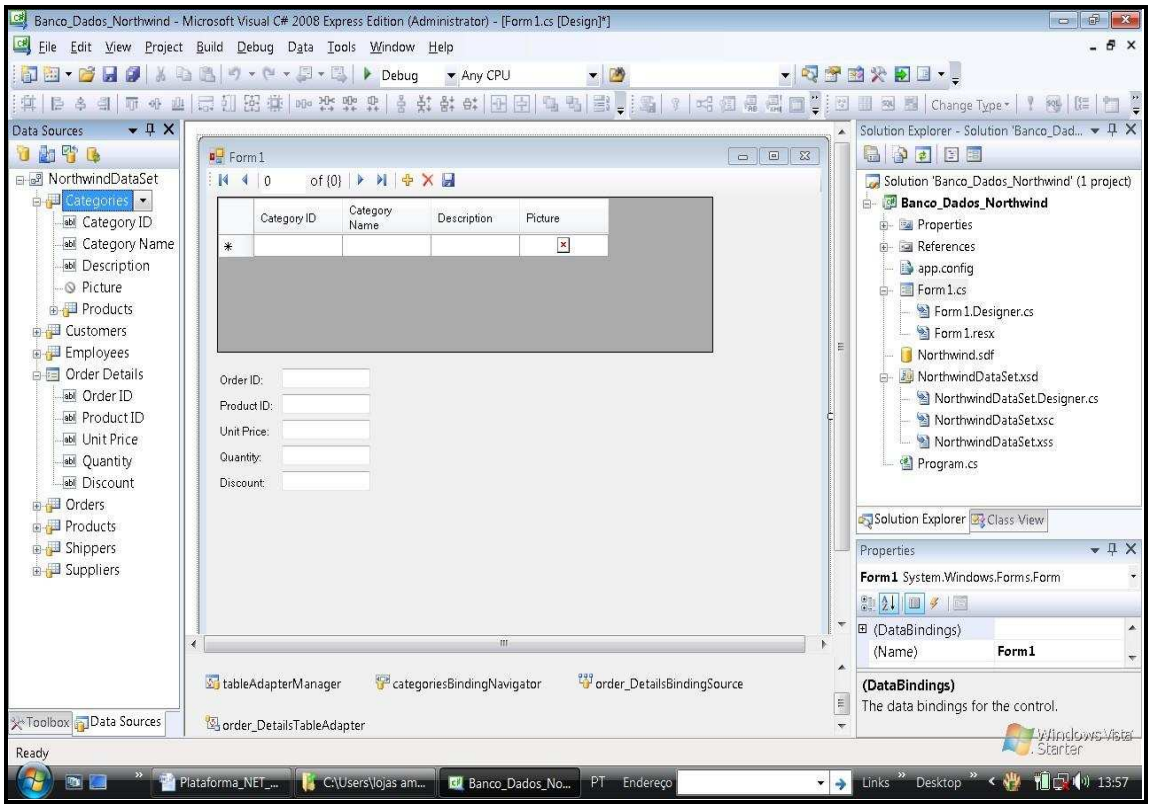


figura 51: escolhemos duas tabelas: Categories e Order Details.

Clique em **Build** para compilar o seu projeto e execute-o. Pronto, aqui está a nossa aplicação construída, ela acessa diretamente o conteúdo das tabelas do banco de dados Northwind à nossa disposição.

Aplicativo em execução:

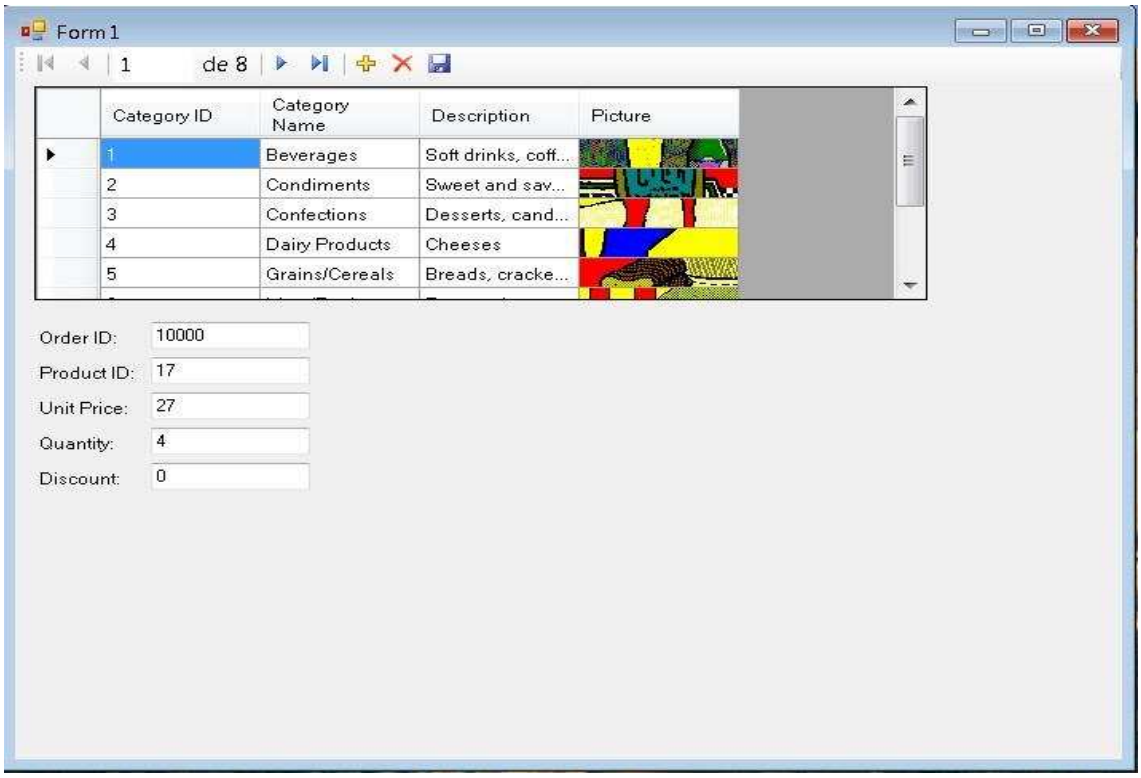


figura 52: aplicativo em execução!

O aplicativo permite a inclusão e deleção de registros, a primeira tabela está na forma de um **DataGrid**, como se fosse uma planilha Excel e a segunda tabela, abaixo, aparece como uma seleção de registros daquela tabela.

A conclusão de um aplicativo completo de banco de dados envolve muita coisa que não tratatermos aqui para encurtar a nossa discussão, há muitos bons textos apresentando os detalhes de como construírmos esses aplicativos. Se você montou um banco de dados Acess 2007 ou anterior, é muito fácil fazer a conexão de dados, bastando seguir os mesmos passos que fizemos no início da seção. Agora alguns comentários: o objeto **DataGridView** sofreu muitas modificações de dois anos para cá, modificações efetuadas pela Microsoft. Este objeto contém mais de 200 propriedades diferentes e capacidades muito extensas para se conectar com a C# e ADO.NET.

Não é surpresa que algum guru da Microsoft venha a escrever um livro apenas descrevendo todas as capacidades deste controle. Você pode declarar uma variável da classe **DataGridView** que se conecta ao objeto acrescentado ao formulário e usar as propriedades de rotulação e posicionamento de linhas e colunas e facilmente você escreve um aplicativo que realiza cálculos em determinadas linhas e colunas.

Note que este objeto, **DataGridView** está à nossa disposição na **ToolBox**, mesmo que você crie um aplicativo que não acesse diretamente banco de dados algum, pois você pode desejar criar um pequeno aplicativo que aceite o preenchimento manual de dados

numéricos em linhas e colunas, e realizar cálulos que você codificar, como se fosse uma mini-planilha!

Conclusões

O mundo da computação sofreu uma grande ruptura conceitual quando do surgimento do paradigma da orientação a objetos, mas nos anos 70 ainda não existiam grandes plataformas e IDEs que tornassem a programação tão largamente difundida. Isto só veio a ocorrer com o surgimento dos produtos da Borland, Microsoft e outras poucas grandes companhias. No entanto, ainda antes de 2002 (à exceção da plataforma Java que surgiu alguns anos antes) os desenvolvedores amadores e profissionais ainda estavam às voltas com os grandes problemas presentes nas tecnologias pré-.NET.

O inferno das DLLs era um destes grandes problemas. Com o surgimento da plataforma .NET surgiram os aplicativos seguros e escaláveis: não comprometem seriamente a integridade dos Sos e possuem independência de plataforma. Além disso, as gerências de memória que os aplicativos escritos em C e C++ exigiam não estão mais presentes na maioria dos aplicativos que são escritos em C#.

É verdade que um desenvolvedor pode se beneficiar da construção de variáveis **ponteiros** em aplicativos C# para gerenciar, digamos manualmente, a memória em tempo de execução, mas estes aplicativos são mais particulares. Neste caso, dizemos que o aplicativo é **unsafe** (código inseguro) pois é não gerenciado. Lembre-se de que o Garbage Collector realiza a gerência automática de recursos para você, mas você pode acessar manualmente as áreas de memória usando os ponteiros.

Você pode escrever um aplicativo com uma seção de código unsafe e o restante pode ser aplicativo seguro e as variáveis na seção segura podem ainda se comunicar com a seção **unsafe** que contém ponteiros.

As novas tecnologias elaboradas pela Microsoft são um convite aos desenvolvedores para migrarem para a .NET seja com fins comerciais ou mesmo entusiásticos.

As nossas discussões ao longo deste texto estão a anos luz atrás dos verdadeiros processos de criação de software profissional; aqui apenas apresentamos alguns vislumbres das características gerais da plataforma. Há muita coisa que deve ser levada em conta: a padronização da construção (design) de seus aplicativos, centenas de horas de testes que a sua equipe deve realizar para testar as aplicações contra uma centena de bugs possíveis, controle de versões, documentação técnica e de usuário (arquivos de help) e muita coisa a mais. Tudo isto será tratado pelo curso ao longo dos semestres.

Novas Tecnologias: Os aplicativos tipo WPF (Windows Presentation Foudation) escritos em XMl, mas que fornecem suporte a C# e o Expression Blend são únicos em seu ramo, e mesmo a plataforma Java da Sun está muito atrás destas tecnologias.

Até mesmo engine (motor) para game development está disponível no **Visual Studio**: a plataforma **XNA** é um grande pluguin que se agrega ao **Visual Studio** e é gratuito, permite que você desenvolva jogos para PC e para o Xbox 360 e é extretmamente robusta, usando os recursos gráficos e de som dos PCs de modo maximizado. É uma grande arena para os hobbistas e mesmo profissionais da área.

O mundo competitivo do mercado de aplicativos mostra que os usuários são muito mais aderentes aos aplicativos de elaborada interface gráfica (são mais intuitivos) e neste sentido, a plataforma WPF e o Expression Blend trazem ferramentas sem igual para o mercado. Com certeza teremos a oportunidade de descrevermos estas novas plataformas de desenvolvimento, e as novas tendências para o futuro, numa nova oficina.

Obrigado pela atenção, seu professor: Paulo Sérgio Custódio, UNIFIG.

Bibliografia recomendada: Os seguintes livros são leitura obrigatória para o aspirante a desenvolvedor **profissional** ou **amador sério**, e a Santa Inquisição está à procura dos programadores que ainda não leram estes livros! Se apresse!

1) Descreve os fundamentos da construção de algoritmos, complexidade de algoritmos e assuntos afins

PROJETO DE ALGORITMOS
FUNDAMENTOS, ANALISE E EXEMPLOS DA INTERNET
Autor: TAMASSIA, ROBERTO
Autor: GOODRICH, MICHAEL T.
Editora: BOOKMAN COMPANHIA ED

2) Guia da Microsoft Press sobre as práticas de programação

CODE COMPLETE
GUIA PRATICO PARA A CONSTRUÇAO DE SOFTWARE
Autor: MCCONNELL, STEVE
Editora: BOOKMAN COMPANHIA ED

3) Excelente livro de Programação em C# e em português

MICROSOFT VISUAL C# 2008 PASSO A PASSO

Autor: SHARP, JOHN
Editora: BOOKMAN COMPANHIA ED
Assunto: INFORMATICA-PROGRAMAÇAO

4) Livros da série Use a Cabeça, exemplos (leitura muito agradável e ótimo conteúdo)

USE A CABEÇA ANALISE & PROJETO ORIENTADO AO OBJETO

Autor: WEST, DAVID
Autor: MCLAUGHLIN, BRETT
Autor: POLLICE, GARY

Editora: ALTA BOOKS