MINI-CURSO DE C#

(CSHARP)

Introdução a .NET

C# (CSharp) é uma linguagem de programação orientada a objetos desenvolvida pela Microsoft como parte da plataforma .Net(lê-se DOTNET). A sua sintaxe orientada a objetos foi baseada no C++ mas inclui muitas influencias de outras linguagens de programação, (Delphi e Java).

Durante o desenvolvimento da plataforma .NET, as *class libraries* foram escritas originalmente num compilador/linguagem chamada Simple Managed C (SMC). Mas, em Janeiro de 1999, o Anders Hejlsberg que fora escolhido pela Microsoft para desenvolver a linguagem, forma uma equipe de desenvolvimento e dá inicio à criação da linguagem chamada de Cool. Um pouco mais tarde, em 2000, o projeto .NET era apresentado ao publico na *Professional Developers Conference* (PDC), e a linguagem Cool fora renomeada e apresentada como C#.

Embora existam várias linguagens para a plataforma .NET (como VB.NET, C++, J#), a C# é considerada a LINGUAGEM do .NET, devendo-se isso ao seguinte:

- Foi criada praticamente do zero para funcionar na nova plataforma, sem preocupações de compatibilidade com código de legado.
- O compilador C# foi o primeiro a ser desenvolvido.
- A maior parte das classes do .NET Framework foram desenvolvidas em C#.

A criação da linguagem, embora tenha sido feita por vários programadores, é atribuída principalmente a Anders Hejlsberg, hoje um *Distinguished Engineer* na Microsoft. Anders Hejlsberg fora o arquiteto de alguns compiladores da Borland, entre suas criações mais conhecidas estão o Turbo Pascal e o Delphi.

O que podemos fazer com .NET?

Aplicações Windows, aplicações Web, Aplicações para dispositivos móveis, componentes enfim o que vier na sua mente e o melhor você pode fazer isso com uma linguagem de sua preferência, você aprende ela uma vez e pronto pode sair fazendo qualquer tipo de projeto, aliás, com .NET isso não existe mais: ”Você terá que aprender Tal linguagem”, ao invés disso existe essa pergunta: ”Em qual linguagem você trabalha?”, com certeza ela já deve ser habilitada ao .NET.

Introdução a C#

C# com certeza é uma linguagem fácil e poderosa e você poderá comprovar isso no decorrer do mini-curso, podemos dizer que é tão fácil como java e tão poderosa como o C++.
Algumas características do C#:

- Case Sensitive – Diferencia maiúsculas de minúsculas
- Trabalha em ambiente gerenciado – O programador não precisa se preocupar, por exemplo, com liberação e alocação de memória isso é feito de forma automática.
- Totalmente dentro do padrão de POO – Linguagem totalmente orientada a objetos.

Nosso primeiro programa em C#

Agora que já sabemos um pouco sobre a linguagem vamos então fazer nosso primeiro programa em C#, o famoso "Hello World".

PASSOS	DESCRIÇÃO
1	Crie um novo projeto to tipo WindowsForm
2	Localize o Componente “Button“ na ToolBox, e arraste-o para o formulário
3	Click duas vezes sobre o Button
4	No evento do Button digite: `MessageBox.Show("Bem Vindo ao Mini-Curso de C#");`
5	Tecle F5 para compilar

```
private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show("Bem Vindo ao Mini-Curso de C#");
}
```

using System;

A keyword ‘using’ importa o Namespace ‘System’ para nosso programa, ou seja, para que todas as classes e métodos do Namespace ‘System’ possam ser usados em nosso programa. Atente-se ao ponto e virgula(;) no final da instrução.

NameSpace

NameSpace no .NET tem a função de organizar as classes, para que não se misturem com outras classes de mesmo nome. São equivalentes aos pacotes do Java.

Operadores Aritméticos

Em C# temos os seguintes operadores aritméticos:

Operador	Descrição
+	(Adição)
-	(Subtração)
*	(Multiplicação)
/	(Divisão)
%	(Resto/Módulo)

Operadores de atribuição

Em C# temos os seguintes operadores de atribuição:

Operador	Descrição
=	Atribuição simples
+=	Atribuição aditiva
-=	Atribuição Subtrativa
*=	Atribuição Multiplicativa
/=	Atribuição de divisão
%=	Atribuição de módulo

Operadores relacionais

Em C# temos os seguintes operadores relacionais:

Operador	Descrição
==	Igualdade
>	Maior
<	Menor
<=	Menor igual
>=	Maior igual
!=	Diferente

Tipos de variáveis:

A seguinte tabela mostra os tipos do C# com sua referencia no Framework.
Os tipos da tabela abaixo são conhecidos como tipos internos ou Built-in.

C# Type	.NET Framework type
bool	System.Boolean
byte	System.Byte
sbyte	System.SByte

char	System.Char
decimal	System.Decimal
double	System.Double
float	System.Single
int	System.Int32
uint	System.UInt32
long	System.Int64
ulong	System.UInt64
object	System.Object
short	System.Int16
ushort	System.UInt16
string	System.String

Cada tipo no C# é um atalho para o tipo do Framework. Isso quer dizer que se declararmos a variável desta forma:

string nome;

ou dessa forma

System.String nome;

teremos o mesmo resultado. O atalho serve apenas para facilitar na hora de desenvolver a aplicação.

A seguinte tabela mostra os tipos de variáveis e os valores possíveis de se armazenar em cada uma delas.

C# Type	**Valores possíveis de se armazenar**
bool	Verdadeiro ou Falso (Valores booleandos)
byte	0 a 255 (8 bits)
sbyte	-128 a 127 (8 bits)
char	Um caractere (16 bits)
decimal	±1.0 × 10−28 a ±7.9 × 1028 (128 bits)
double	±5.0 × 10−324 a ±1.7 × 10308 (64 bits)
float	±1.5 × 10−45 a ±3.4 × 1038 (32 bits)
int	-2,147,483,648 a 2,147,483,647 (32 bits)
uint	0 a 4,294,967,295 (32 bits)
long	–9,223,372,036,854,775,808 a 9,223,372,036,854,775,807 (64 bits)
ulong	0 a 18,446,744,073,709,551,615 (64 bits)
object	Qualquer tipo.

short	-32,768 a 32,767 (16 bits)
ushort	0 a 65,535 (16 bits)
string	Seqüência de caracteres (16 bits por caractere)

Estrutura de Decisão

Basicamente temos dois tipos básicos de instruções de decisão:

1. if... else;
2. switch.. case.

if... else

A instrução 'if...else' é usada para executar blocos de código condicionalmente através de uma expressão Booleana (verdadeiro ou falso). A clausula 'else' é opcional e seu conteúdo será executado somente se o resultado da expressão booleana for false (falso). Veja o exemplo:

```
If ( a == 5 )
   MessageBox.Show("a é igual a 5");
else
    MessageBox.Show ("a é diferente de 5");
```

No código acima na instrução 'if' (Se) fazemos uma verificação muito simples, onde testamos se 'a' é igual a '5' caso seja verdadeiro (true) o código logo abaixo é executado, ou seja: 'System.Console.WriteLine('a é igual a 5');'. Caso o resultado dessa expressão lógica seja falso, ou seja, 'a' não seja igual a '5' o código do bloco 'else' (se não) será executado, nesse caso: "System.Console.WriteLine('a é diferente de 5');".

Para casos onde nosso código tem mais de uma linha (o que é mais comum) devemos usar as chave '{' e '}' para delimitarmos os blocos if e/ou else, veja:

```
if ( a == 5 && b != 2 )
{
    MessageBox.Show ("a é igual a 5");
    MessageBox.Show ("b é diferente a 5");
}
```

Neste caso temos uma verificação composta, ou seja, verificamos duas condições se 'a' é igual a '5' E(&&) se 'b' é diferente de '2' essa condição só resultará em verdadeiro caso as duas condições sejam verdadeiras pois estamos usando o operador lógico '&&'.

else if

A instrução 'else if' é usada quando temos varias condições e pra cada condição temos códigos diferentes, ou seja, fazemos coisas diferentes para cada caso veja o exemplo:

```
if  ( a== 1)
{
  // perceba que mesmo com uma linha só de código eu posso usar os delimitadores
 MessageBox.Show ("a é igual a 1");
}
else if ( b ==2 )
{
  MessageBox.Show ("b é igual a 2");
}
else if( c ==3 )
{
  MessageBox.Show ("c é igual a 3");
}
else
{
  MessageBox.Show ("Faz algo caso nenhuma condição seja atendida");
}
```

Podemos 'traduzir' o 'else if' para 'se caso'.

A instrução 'if' ainda pode ser emulada de forma mais simples através do operador '?' (interrogação) veja:

```
int a = (expressãoBooleana) ? 1 : 0;
```

Nesse caso se a expressão Booleana seja verdadeira (true) a variável 'a' receberá o valor '1' e caso seja falso (false) a variável recebe '0'. Veja um exemplo prático:

```
MessageBox.Show (a == 5 ? "Sim a é igual à 5" : "Não a é diferente de 5");
```

Se 'a' for igual a 5 o resultado será: "Sim a é igual à 5", caso 'a' não seja igual a '5' o resultado será: "Não a é diferente de 5".

switch... case

A instrução 'switch.. case' prove um caminho mais limpo para escrever múltiplas instruções 'if..else', veja o exemplo:

```
switch(a)
{
  case 1:
    MessageBox.Show ("a é igual a 1");
    break;
```

```
  case 2:
    MessageBox.Show ("a é igual a 2")
    break;
  default:
     MessageBox.Show ("a não é igual a 1 e nem igual a 2");
      break;
}
```

Colocamos a variável 'a' dentro do 'switch' e dentro de nosso código vamos verificando com o 'case' o valor de nossa variável. No nosso código temos duas possibilidades para caso seja 1 e caso seja 2 e ainda temos a opção 'default' que é conhecida como 'case else', ou seja, se nenhum 'case' for atendido ela será executada, lembrando que é opcional colocar o 'default'.

Estrutura de Repetição

As estruturas de repetição são usadas para controlar a execução de códigos repetidamente até que uma condição seja verdadeira.

Veja os tipos de estrutura de repetição que vamos aprender neste artigo:

- Laço for
- Laço while
- Laço do.. while
- Laço foreach

Laço for

Uma estrutura de repetição é também conhecida como Loop/Laço. O primeiro tipo de loop que vamos estudar é o 'for'. O loop 'for' trabalha checando uma condição para executar um bloco de código até que essa condição seja verdadeira, no caso do loop 'for' temos que em sua syntax declarar sua inicialização, sua condição e seu incremento, veja:

```
for (int i =0; i <= 10; i++)
{
        //instruções
}
```

No código acima temos a syntax de um loop 'for' onde na primeira parte declaramos uma variável do tipo inteiro (int) e a inicializamos com o valor 0 (zero), na segunda parte temos a condição nesse caso verifica se a nossa variável recém criada é menor ou igual a 10 e a terceira e ultima parte é o incremento desta variável, sendo essas três partes separadas por ';' (ponto e virgula). O funcionamento é simples todo o código dentro desse bloco do 'for' será executado dez vezes. Simples não? Mas você deve ter algumas perguntas em mente como, por exemplo: porque a variável chama 'i'? o que é incremento? Vamos as respostas!

Primeiro chama-se 'i' por um motivo simples: convenção. 'i' vem de índice e isso já se tornou meio que um padrão, mas essa variável pode ter qualquer nome, por exemplo: contador.

Segundo um incremento nada mais é do que adicionar 1 a uma variável, ou seja, se uma variável vale 0 (zero) e passa por um incremento logo essa variável vale 1 e se passa mais uma vez vale 2 e assim por diante. E o que acontece com o decremento é o inverso, ou seja, se uma variável vale 2 e passar por um decremento agora passa a valer 1 e assim por diante.

Respondidas as questões veja um exemplo pratico do loop 'for':

```
for (int i = 0; i < 20; i++)
{
   int res = i * 2;
   MessageBox.Show (res.ToString());
}
```

Veja que embora simples mostra bem o funcionamento do 'for', neste caso vai mostrar na tela o valor de 'i' multiplicado por 2 enquanto 'i' for menor que 20.

Laço while

De modo diferente do loop 'for' (embora o objetivo seja o mesmo, ou seja, repetir a execução de um código testando uma condição) o loop 'while' é mais simples de ser entendido, pois sua syntax não requer que você coloque na mesma linha variável de inicialização, condição e o seu incremento. No loop 'while' apenas colocamos a condição que queremos testar, veja como fica a syntax:

```
while (expressão booleana)
{
            //instruções
}
```

Veja como é simples o código. Expressão booleana é uma expressão que sempre retorna falso ou verdadeiro e a instruções dentro do bloco de código do loop 'while' só será executada enquanto essa expressão retornar verdadeiro. Veja um exemplo:

```
int contador = 2;

while (contador != 10)
{
      MessageBox.Show (contador.ToString());
       contador++;
}
```

Neste caso temos uma variável chamada contador e seu valor é 2, no nosso loop 'while' testamos se a variável contador é diferente de 10 caso verdadeiro mostramos

na tela o valor atual de contador e o incrementos em 1 e o loop 'while' continuará até que essa condição se torne falsa.

Cuidado: não esqueça de incrementar ou se assegurar que sua condição pode ser falsa em algum momento, pois caso contrario você entrará em um loop infinito, ou seja, a condição nunca será falsa e o loop vai continuar até travar a máquina.

Laço do.. while

Vejamos, porque é que teríamos mais um loop do tipo 'while'? Se fossemos analisar com cuidado veríamos que o loop 'while' dependendo do caso pode nunca ser executado, ou seja, se a condição do loop 'while' retorna falsa de primeira ele nunca vai ser executado. No exemplo acima se atribuíssemos o valor 10 a variável contador em sua declaração o loop 'while' nunca começaria. Com o loop 'do.. while' o código será executado ao menos uma vez porque nós fazemos a verificação da condição no final da instrução, veja:

```
do
{
//instruções
}
while (expressão booleana)
```

Podemos traduzir 'do' para 'faça', ou seja, faça as instruções enquanto (while) expressão seja verdadeira. Assim garantimos que ao menos uma vez nossas instruções serão executadas. Exemplo:

```
int contador = 10;

do
{
        MessageBox.Show (contador.ToString());
}
while(contador != 10);
```

Veja que mesmo contador sendo igual a 10 a instrução será executa ao menos uma vez porque só depois que fazemos a verificação.

Laço foreach

Nós iremos aprender sobre arrays no próximo artigo, mas para mostrar o funcionamento do loop 'foreach' tenho que usar um array, explicarei melhor no sobre arrays no próximo artigo.

O loop 'foreach' é usado para interagir (percorrer) listas. Ele opera sobre Arrays ou coleções veja sua syntax básica:

```
foreach(<tipo de dado> <nome> in <lista>)
```

```
{
        //instruções
}
```

Veja um exemplo pratico para facilitar o entendimento:

```
string[] nomes = {"Cleber", "Carol", "Denis", "Roberto"};

foreach (string pessoa in nomes)
{
    MessageBox.Show ("{0} ", pessoa);
}
```

Criamos um array de string e colocamos alguns elementos dentro e no nosso loop 'foreach' será exibida todos os elementos dentro de nosso array. Veremos mais sobre o 'foreach' no próximo artigo que trata de arrays.

O que é ADO .NET

ADO .NET é a nova tecnologia para acesso a dados da plataforma .NET estando integrada ao .NET Framework e oferecendo diversas classes que permitem realizar praticamente todas as tarefas relacionadas com o acesso e manutenção de dados.

ADO .NET oferece suporte a uma variedade de opções para desenvolvimento de soluções com acesso a dados que permitem a comunicação com qualquer fonte de dados, desde os já conhecidos gerenciadores de banco de dados relacionais (**SGBD**) como : **SQL Server, MySQL, FireBird, Oracle, Sybase, Access, XML**, arquivos textos, etc.

Os componentes considerados os pilares da ADO.NET são o **DataSet** e os **provedores .NET** que são um conjunto de componentes que incluem os objetos **:**

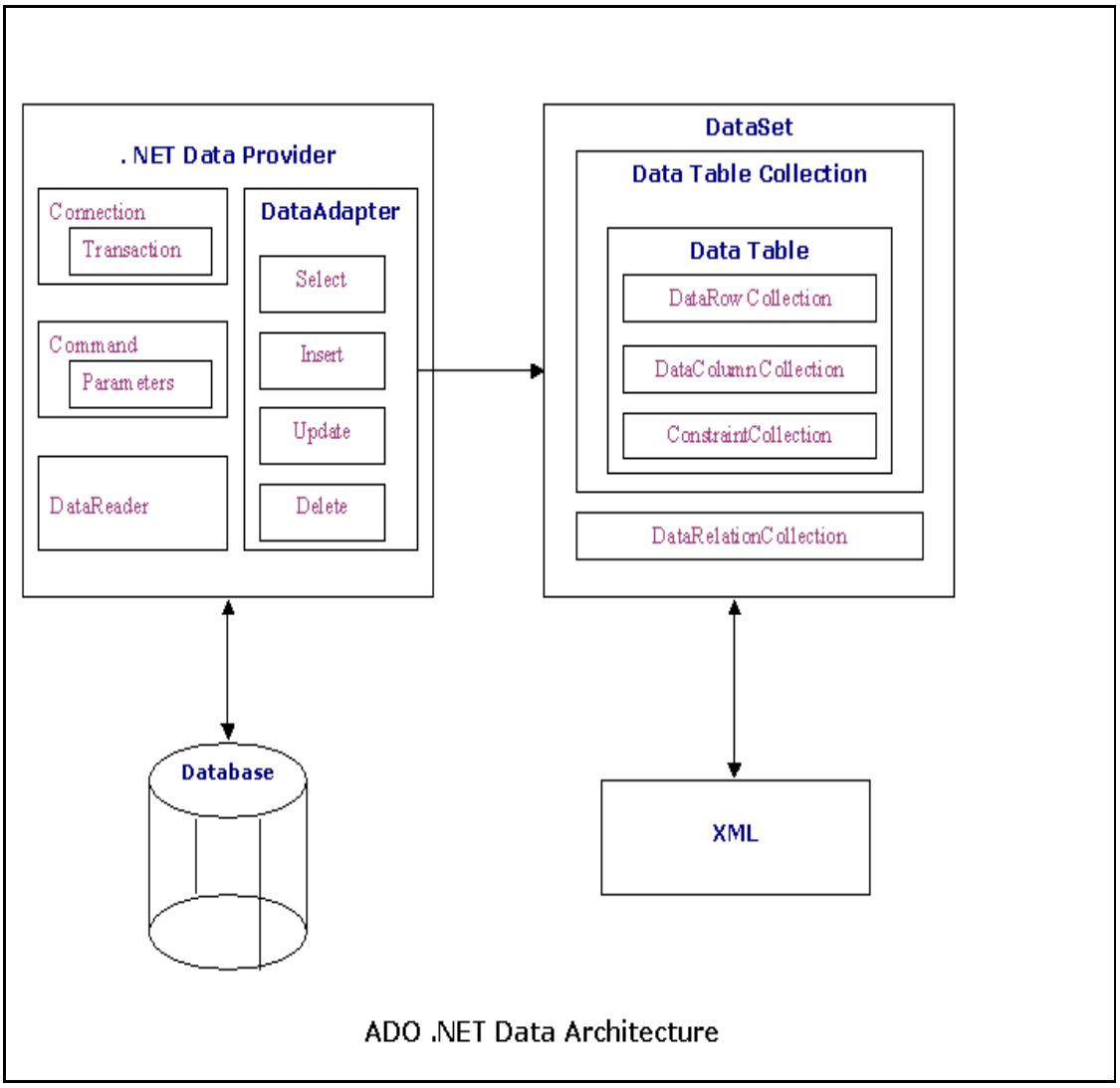
- **Connection - responsável por efetuar a conexão com o banco de dados**
- **Command - responsável por executar comandos contra o banco de dados;**
- **DataAdapter - é utilizado para preecher o objeto DataSet;**

O ADO.NET pode acessar dados de várias maneiras: **OLE DB , ORACLE, SQL, ODBC** e driver de terceiros como os do FireBirb.

Cada objeto possui uma versão para cada uma das maneiras aqui mencionadas, assim temos os objetos :

- ***OleDbConnection, OleDbCommand, OleDbDataReader, OleDataAdapter;***
- ***SqlConnection, SqlCommand, SqlDataReader, SqlDataAdapter;***

- ***OdbcConnection, OdbcCommand,etc.***

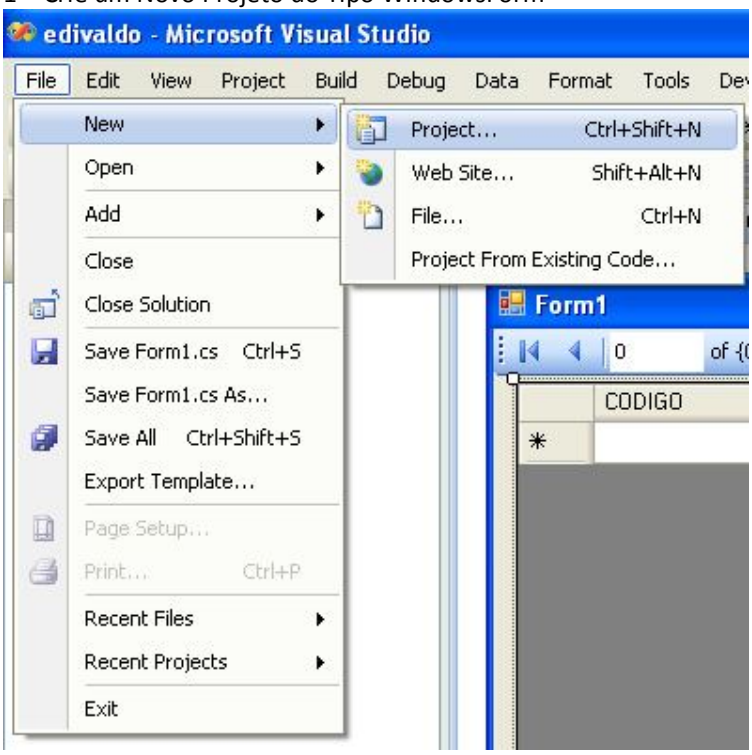


ADO .NET Data Architecture

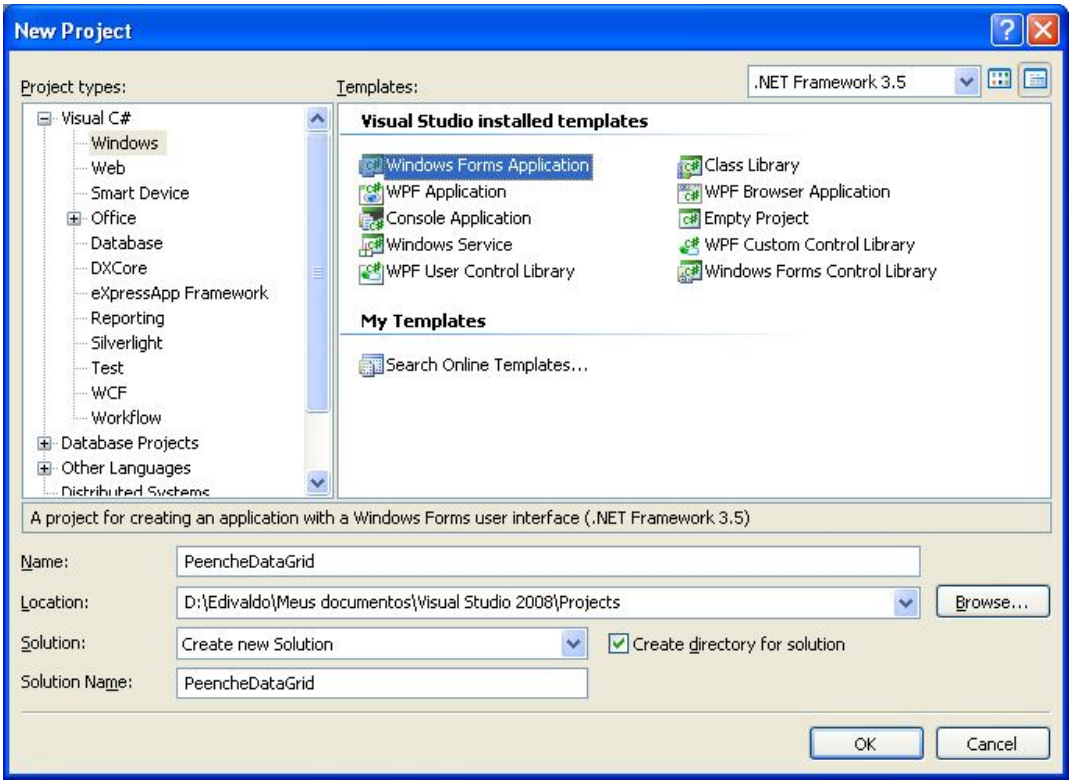
Como Preeencher uma **DataGriView utilizando DataSet**

Acompanhe o exemplo mostrado

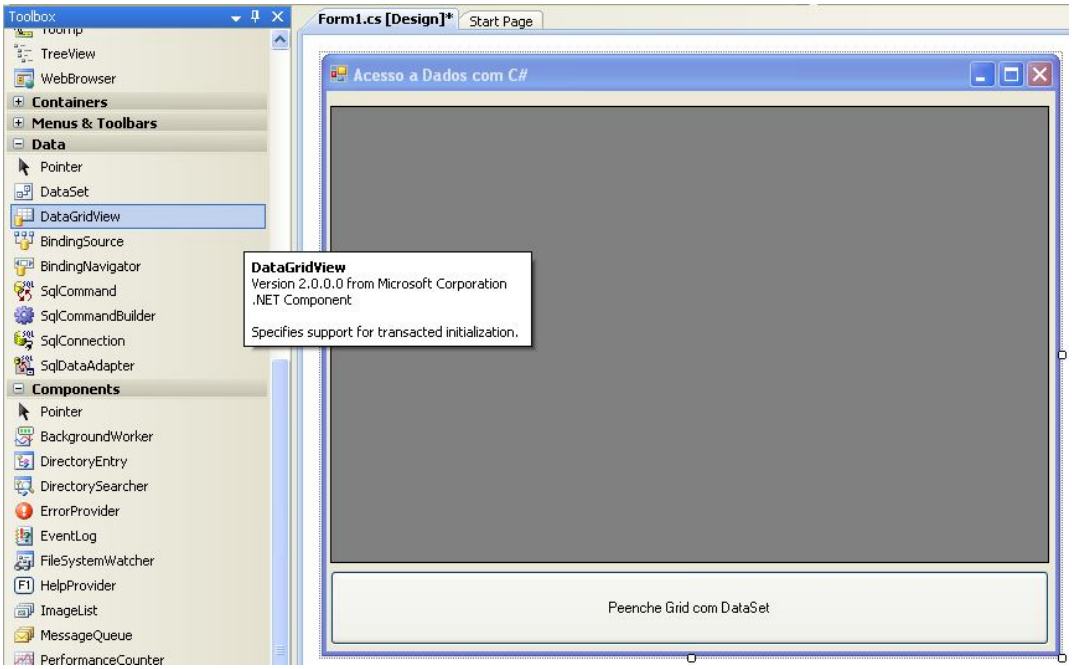
1 – Crie um Novo Projeto do Tipo WindowsForm



Na janela **Novo Projeto** selecione **C#->Aplicações Windows** e na janela **Modelos** marque **Aplicação Windows** e informe o nome da solução , no meu caso: **AcessoDadosC**;



A partir da janela **Ferramentas** abra a guia Data e selecione o controle **DataGridView** arrastando-o para o formulário e configurando sua propriedade **Dock** para **Top**. A seguir na guia **Windows Forms** arraste o controle Button para o formulário e defina sua propriedade **Dock** para **Botton**. O resultado final você vê na figura abaixo:



Inclua agora o código do botão C# associado ao evento **Click** do botão conforme abaixo:

```
void Button1Click(object sender, EventArgs e)
{
//define a string de conexao com provedor caminho e nome do banco de
dados
string strProvider = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=c:\\dados\\Cadastro.mdb";
//define a instrução SQL
string strSql = "SELECT * FROM Clientes";

//cria a conexão com o banco de dados
OleDbConnection con = new OleDbConnection(strProvider);
//cria o objeto command para executar a instruçao sql
OleDbCommand cmd = new OleDbCommand(strSql, con);

//abre a conexao
con.Open();

//define o tipo do comando
cmd.CommandType = CommandType.Text;
//cria um dataadapter
OleDbDataAdapter da = new OleDbDataAdapter(cmd);

//cria um objeto datatable
DataTable clientes = new DataTable();

//preenche o datatable via dataadapter
da.Fill(clientes);

//atribui o datatable ao datagridview para exibir o resultado
dataGridView1.DataSource = clientes;
}
```

Conhecendo o Ambiente de Desenvolvimento

Ferramentas Express

- Visual C# 2005 – Express Edition;
- Visual Web Developer 2005 – Express Edition;
- SQL Server 2005 – Express Edition;

São ferramentas de desenvolvimento voltadas à estudantes e entusiastas em geral que querem aprender/conhecer a plataforma .NET. Possui algumas limitações em se tratando de desenvolvimento corporativo mas possui recursos incríveis e produtivos para pequenas aplicações.