

Apostila de C# & Asp.Net

.Net Framework

Tópicos:

- Microsoft .NET
- CLR – Common Language Runtime
- Metadata
- Assemblies
- Linguagens habilitadas ao .NET
- Common Type System
- Web Services
- ADO.NET

Microsoft .NET

Microsoft .NET (comumente conhecido por .NET Framework em inglês dotNet) é uma iniciativa da empresa Microsoft, que visa uma plataforma única para desenvolvimento e execução de sistemas e aplicações. Todo e qualquer código gerado para **.NET**, pode ser executado em qualquer dispositivo que possua um framework de tal plataforma.

Com idéia semelhante à plataforma Java, o programador deixa de escrever código para um sistema ou dispositivo específico, e passa a escrever para a plataforma **.NET**.

A plataforma **.NET** é executada sobre uma Common Language Runtime - CLR (Ambiente de Execução Independente de Linguagem) interagindo com um Conjunto de Bibliotecas Unificadas (framework). Esta CLR é capaz de executar, atualmente, mais de 20 diferentes linguagens de programação, interagindo entre si como se fossem uma única linguagem. Estas são:

- | | | |
|--------------------|--------------------------|----------------|
| • APL | • Java | • Perl |
| • Boo | • JScript | • PowerBuilder |
| • Clarion | • J# | • PowerShell |
| • COBOL | • Lua | • Python |
| • Component Pascal | • Mercury | • RPG |
| • C# | • Mondrian | • Ruby |
| • C++ | • Oberon | • Scheme |
| • Delphi | • Object Pascal / Delphi | • SmallTalk |
| • Eiffel | Language | • Standard ML |
| • Forth | • Oz | • Visual Basic |
| • Fortran | • Pascal | |
| • Haskell | | |

Esta plataforma permite a execução, construção e desenvolvimento de Web Services (Aplicações Web) de forma integrada e unificada.

A plataforma **.NET** baseia-se em um dos princípios utilizados na tecnologia Java ((Just In Time Compiler - JIT), os programas desenvolvidos para ela são duplo-compilados (compilados duas vezes), uma na distribuição (gerando um código que é conhecido como "bytecodes") e outra na execução.

Um programa é escrito em qualquer das mais de vinte linguagens de programação disponíveis para a plataforma, o código fonte gerado pelo programador é então compilado pela linguagem escolhida gerando um código intermediário em uma linguagem chamada MSIL (Microsoft Intermediate Language).

Este novo código fonte gera um arquivo na linguagem de baixo nível Assembly, de acordo com o tipo de projeto:

- EXE - Arquivos Executáveis, Programas
- DLL - Biblioteca de Classes
- ASPX - Página Web
- ASMX - Web Service

No momento da execução do programa ele é novamente compilado, desta vez pelo compilador JIT, de acordo com a utilização do programa, por exemplo: Temos um Web Site desenvolvido em ASP.NET, ao entrar pela primeira vez em uma página o JIT irá compila-la, nas outras vezes que algum outro usuário acessar esta página, ele usará esta compilação.

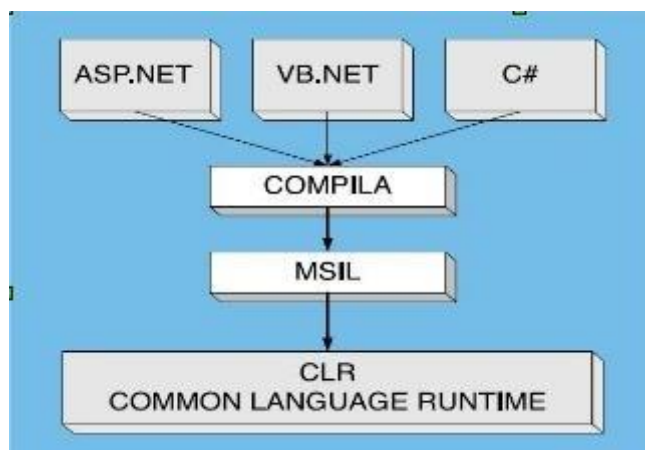
Também é possível, através de ferramentas específicas, "pré-compilar" o código para que não se tenha o custo da compilação JIT durante a execução.

O fato desta arquitetura utilizar a MSIL gera uma possibilidade pouco desejada entre os criadores de software que é a de fazer a "engenharia reversa", ou seja, a partir de um código compilado, recuperar o código original. Isto não é uma ideia agradável para as empresas que sobrevivem da venda de softwares produzidos nesta plataforma. Por causa disso, existem ferramentas que "ofuscam" o código MSIL, trocando nomes de variáveis, métodos, interfaces e etc para dificultar o trabalho de quem tentar uma engenharia reversa no mesmo.

CLR(Common Language Runtime)

O CLR é um ambiente de execução, e poderíamos até dizer que é o "Coração do .NET", o qual dá suporte a todas as linguagens de programação habilitadas para o .NET.

O Runtime (ambiente de execução) é o ambiente que dá suporte à execução das aplicações .NET. Quando um programa .NET é executado, todo o controle do mesmo é feito através do CLR. Para aplicações anteriores, desenvolvidas utilizando COM/COM+, o programador era responsável por inserir no código do programa uma série de funções necessárias ao correto funcionamento do mesmo, como por exemplo o Gerenciamento de memória, criação e destruição de objetos. A codificação destas funções não era uma tarefa fácil, o que exigia muito tempo do programador, além de conhecimentos avançados. Com aplicações .NET, todas estas funções são executadas pelo CLR, ou seja, o programador não precisa preocupar-se com as mesmas. Desta forma somente precisamos nos preocupar com a funcionalidade do nosso programa, o que poupa tempo e agiliza o processo de desenvolvimento. As aplicações criadas em uma das linguagens habilitadas para o .NET (como VB.NET, C# ou ASP.NET), ao serem compiladas, geram um código intermediário conhecido como MSIL – Microsoft Intermediate Language, o qual é abreviado simplesmente como IL – Intermediate Language. Este código é que é executado pelo CLR. Vamos analisar o diagrama apresentado na abaixo:



CTS(Common Type System)

O Framework .NET disponibiliza, na forma de objetos, um conjunto de tipos de dados comuns, os quais podem ser utilizados por todas as linguagens habilitadas ao .NET. Isso significa que uma variável do tipo Int terá a mesma estrutura e ocupará o mesmo número de bytes, quer seja no C#, no VB.NET, no C++ ou em qualquer outra linguagem habilitada ao .NET. Este conjunto de tipos comuns, que pode ser utilizado por qualquer linguagem, é chamado de Common Type System, que a partir de agora abreviaremos por CTS.

Dentre outras coisas, é o CTS que facilita a integração entre os programas e serviços criados, utilizando-se de diferentes linguagens do .NET. No modelo antigo, uma das dificuldades de fazer com que um Componente COM+ criado com o Visual C++ pudesse ser utilizado por um programa escrito em Visual Basic é que as linguagens possuíam um diferente conjunto de tipos básicos. Para que os componentes, escritos em diferentes linguagens, pudessem se comunicar, o programador tinha que mapear os tipos de uma linguagem, para os tipos correspondentes em outra linguagem, fazendo as conversões necessárias.

Vejam o quanto este procedimento é trabalhoso. Com o CTS do .NET simplesmente esta preocupação não existe, uma vez que todas as linguagens têm acesso a um conjunto de tipos comum a todas elas. Conforme descrito na documentação do Framework .NET, são as seguintes as principais funções do CTS: Fornece uma estrutura que possibilita a integração entre diferentes linguagens habilitadas ao .NET, com uma execução mais rápida, uma vez que a sobrecarga para a conversão entre os diferentes tipos de diferentes linguagens deixa de existir.

- Fornece uma estrutura de tipos com base em um modelo orientado a objetos, o que facilita a criação de novas linguagens habilitadas ao .NET, favorecendo a utilização de boas práticas de programação, como por exemplo a herança. Uma vez que os tipos são objetos, podemos criar tipos derivados dos objetos básicos, os quais herdam todas as características dos objetos básicos.
- O CTS define algumas regras que toda linguagem deve seguir, para ser habilitada ao .NET. Por seguirem um conjunto de regras comum, a interação entre programas escritos em diferentes linguagens fica bem mais fácil.

CLS (Common Language Specification)

O CLS, ou Especificação Comum da Linguagem, é um subconjunto do CTS, e define um conjunto de regras que qualquer linguagem que implemente a .NET deve seguir a fim de que o código gerado resultante da compilação de qualquer peça de software escrita na referida linguagem seja perfeitamente entendido pelo runtime .NET. Seguir essas regras é um imperativo porque, caso contrário, um dos grandes ganhos do .NET, que é a independência da linguagem de programação e a sua interoperabilidade, fica comprometido. A grosso modo, dizer que uma linguagem é compatível com o CLS significa dizer que mesmo quando esta é sintaticamente diferente de qualquer outra que implemente .NET, semanticamente ela é igual, porque na hora da compilação será gerado um código intermediário (e não código assembly dependente da arquitetura do processador) equivalente para duas peças de código iguais, porém escritas em linguagens diferentes. É importante entender esse conceito para não pensar que o código desenvolvido em C# não pode interagir com código desenvolvido em VB ou outras linguagens, porque mesmo estas sendo diferentes, todas são compatíveis com o CLS.

MSIL (Microsoft Intermediate Language)

A MSIL – ou simplesmente IL – é a linguagem intermediária para qual é interpretado qualquer programa .NET, independente da linguagem em que este for escrito. Essa tradução é feita para código intermediário (como em JAVA com os byte codes) sintaticamente expresso na IL. Por sua vez, qualquer linguagem .NET compatível, na hora da compilação, gerará código IL e não código assembly específico da arquitetura do processador onde a compilação do programa é efetuada, conforme aconteceria em C++ ou Delphi, por exemplo. E por que isso? Isso acontece para garantir duas coisas: a independência da linguagem e a independência da plataforma (arquitetura do processador). Arquitetura .NET A MSIL é a linguagem intermediária para qual é interpretado qualquer programa .NET na hora da compilação, independente da linguagem em que este for escrito.

Metadata

Ao registrarmos um componente COM/COM+, uma série de informações sobre o mesmo são gravadas na Registry do sistema. Estas informações são utilizadas pelos programas que precisam acessar o componente. Se alguma destas informações estiver errada ou tiver sido alterada, o componente não poderá ser acessado e os programas que utilizam o componente deixarão de funcionar corretamente.

No Framework .NET, para utilizarmos os componentes .NET (.NET Componentes), não é necessário que os mesmos sejam registrados. O próprio componente .NET possui todas as informações necessárias ao seu funcionamento, bem como as informações necessárias para que outros aplicativos possam utilizá-los. Estas informações, que fazem parte do próprio componente .NET ficam gravadas no arquivo que compõe o componente, na forma de Metadata. Uma tradução “popularmente conhecida” para Metadata seria: dados sobre dados. No nosso caso, Metadata seriam as informações que o componente .NET possui a respeito de si mesmo, informações estas que podem ser utilizadas por outros componentes e serviços, para acessar o componente em questão.

Além de fazer com que não seja necessário o registro do componente, as informações de Metadata facilitam a interoperabilidade entre diferentes componentes, mesmo entre componentes escritos em diferentes linguagens. Estas informações são geradas, automaticamente, no momento da compilação do componente e são gravadas no arquivo .DLL ou .EXE do componente. São muitas as informações que podem ser inseridas no componente, na forma de Metadata, tais como:

- Nome e versão do componente.
- Uma chave pública para verificação da origem e da autenticidade do componente.
- Informações sobre todas as classes ou componentes, dos quais o componente depende para funcionar.
- Tipos disponibilizados (exportados) pelo componente.
- Permissões de segurança para o acesso ao componente, seus métodos e propriedades.
- Classes básicas e interfaces do Framework .NET, utilizadas pelo componente.
- Atributos personalizados, implementados no componente.
- Membros do componente (métodos, campos, propriedades, eventos, etc).

Quando o componente é acessado, o CLR carrega os metadados do componente na memória e faz referência a estes metadados, para obter informações sobre as classes, membros, herança e dependências do componente. São diversos os benefícios do uso de metadados, dentre os quais podemos destacar os seguintes:

- Módulos de código auto descritivos: O próprio componente contém toda a informação necessária para interagir com outros componentes. Desta forma o componente pode ser implementado como um único arquivo, o qual contém a sua definição (na forma de metadados) e a sua implementação, o código do componente.
- Nunca é demais repetir: A utilização de metadados facilita, enormemente, a interoperabilidade entre componentes criados usando diferentes linguagens.

Assemblies

Uma aplicação .NET é constituída de um conjunto de “blocos” chamados Assembly. Através dos Assemblies é que podemos controlar a distribuição de uma aplicação, fazer o controle de versões, além de definir as configurações de segurança. Um assembly é uma coleção de tipos e recursos que foram construídos para trabalharem juntos, formando,

com isso, uma unidade com funcionalidade e escopos bem definidos. Um assembly fornece ao CLR importantes informações sobre a implementação de tipos da aplicação. Para o CLR, um tipo somente existe no contexto de um assembly. De uma maneira mais simples, poderíamos dizer que um assembly é o mecanismo utilizado pelo .NET, para “empacotar” todos os elementos e informações necessárias ao funcionamento de uma aplicação ou componente.

Vamos simplificar mais ainda: o assembly é uma maneira de juntar e organizar os diversos elementos que formam uma aplicação ou componente.

Os assemblies foram criados para simplificar a distribuição de aplicações e resolver o problema de “versões”, existentes em aplicações baseadas em componentes.

Com este termo estávamos nos referindo ao problema de um programa, ao ser instalado, substituir uma DLL por uma versão mais nova ou mais antiga, fazendo com que programas que dependiam da versão anterior da DLL deixassem de funcionar. Através do uso de assemblies e dos metadados contidos em cada componente, é possível que diferentes versões, do mesmo componente, estejam disponíveis, ao mesmo tempo, em um computador. Desta forma, cada programa utiliza a versão do componente para o qual o programa foi criado. Ao instalarmos uma nova versão do componente, o qual vem embutido em um assembly, as versões anteriores serão mantidas, se as mesmas estiverem sendo utilizados por outros programas. Isso faz com que o inferno das DLLs (DLL Hell) seja coisa do passado.

Para resolver o problema de versões e evitar o inferno das DLLs, o CLR utiliza assemblies da seguinte maneira:

- Permite que o desenvolvedor defina regras sobre o uso de diferentes versões entre diferentes componentes .NET.
- Fornece a infraestrutura necessária para que as regras de versão definidas pelo desenvolvedor sejam respeitadas.
- Fornece a infraestrutura necessária, para que diferentes versões de um mesmo componente de software possam rodar, simultaneamente. Esta execução simultânea é conhecida como “syde-by-syde execution.”

Um assembly é composto de dois elementos básicos:

- Manifesto.
- Um conjunto de módulos.

GC (Garbage Collector) – Coletor de Lixo / Gerenciamento da Memória

O gerenciamento da memória é efetuado pelo runtime, permitindo que o desenvolvedor se concentre na resolução do seu problema específico. O que diz respeito ao sistema operacional, como o gerenciamento da memória, é feito pelo runtime.

Como isso é efetuado? À medida que uma área de memória é necessária para alocar um objeto, o GC ou coletor de lixo (Garbage Collector) realizará essa tarefa, assim como a liberação de espaços de memória que não estiverem mais em uso. Para os que não trabalham com linguagens de programação como C ou C++, que permitem o acesso direto à memória heap via ponteiros, essa é uma das maiores dores de cabeça que os programadores sofrem, ora por fazer referência a espaços de memória que não foram alocados, ora porque estes espaços já foram liberados anteriormente; é exatamente esse tipo de erro que o coletor de lixo nos ajuda a evitar. O gerenciamento da memória, quando efetuado diretamente pelo programador, torna os programas mais eficientes em termos de desempenho, mas ao mesmo tempo o penaliza, obrigando-o a alocar e desalocar memória quando assim é requerido. A .NET permite que o programador faça esse gerenciamento também, o que é chamado de "unsafe code" (código não seguro); entretanto, por default, o GC é o encarregado dessa tarefa, e o contrário não é recomendado.

A linguagem C#

A estrutura básica de uma aplicação C#

O pequeno trecho de código a seguir implementa o clássico programa “Olá mundo”:

```
using System;
class HelloWorld
{
    static void Main( )
    {
        // escrevendo no console
        Console.WriteLine("Hello World !!!");
        Console.ReadLine( );
    }
}
```

O Cabeçalho do programa

A primeira linha do nosso programa, que escreve no console “Olá mundo em C#”, contém a informação do namespace System, que contém as classes primitivas necessárias para ter acesso ao console do ambiente .NET. Para incluir um namespace em C#, utilizamos a cláusula using seguida do nome do namespace.

A declaração de uma classe

O C# requer que toda a lógica do programa esteja contida em classes. Após a declaração da classe usando a palavra reservada class, temos o seu respectivo identificador. Para quem não está familiarizado com o conceito de classe, apenas adiantamos que uma classe é um tipo abstrato de dados que no paradigma de programação orientada a objetos é usado para representar objetos do mundo real. No exemplo acima, temos uma classe que contém apenas o método Main() e não recebe nenhum parâmetro.

O Método Main()

Todo programa C# deve ter uma classe que defina o método Main(), que deve ser declarado como estático usando o modificador static, que diz ao runtime que o método pode ser chamado sem que a classe seja instanciada. É através desse modificador que o runtime sabe qual será o ponto de entrada do programa no ambiente Win32, para poder passar o controle ao runtime .NET. O “M” maiúsculo do método Main é obrigatório, e seu valor de retorno void significa que o método não retorna nenhum valor quando é chamado.

Algumas variantes do método Main()

```
// Main recebe parâmetros na linha de comando via o array
// args
static void Main(string[ ] args)
{
    // corpo do método
}
// Main tem como valor de retorno um tipo int
static int Main( )
{
    // corpo do método
}
```

A forma do método Main() a ser usada vai depender dos seguintes fatores:

O programa vai receber parâmetros na linha de comando? Então esses parâmetros serão armazenados no array args.

Quando o programa é finalizado, é necessário retornar algum valor ao sistema? Então o valor de retorno será do tipo int.

Um programa escrito em C# pode ter mais de uma classe que implementa o método `Main()`. Nesse caso, deverá ser especificado em tempo de compilação em qual classe se encontra o método `Main()`, que deverá ser chamado pelo runtime quando a aplicação for executada. Exemplo:

```
using System;
class class1
{
    static void Main( )
    {
        Console.WriteLine("Método Main( ) da classe 1");
    }
}

class class2
{
    static void Main( )
    {
        Console.WriteLine("Método Main( ) da classe 2");
    }
}
```

Alguns últimos detalhes adicionais

Blocos de código são agrupados entre chaves `{ }`.

Cada linha de código é separada por ponto-e-vírgula.

Os comentários de linha simples começam com duas barras`//`.

Comentários em bloco são feitos usando os terminadores `/*` (de início) e `*/` (de fim).

```
/* Este é um comentário de bloco
Segue o mesmo estilo de C/C++
*/
```

O C# é sensível ao contexto, portanto `int` e `INT` são duas coisas diferentes. `int` é uma palavra reservada que é um alias do tipo `System.Int32`. `INT` poderia ser um identificador, entretanto não é recomendado usar como identificadores de variáveis o nome de um tipo ou palavra reservada como no exemplo citado.

Sempre declare uma classe onde todos os aspectos inerentes à inicialização da aplicação serão implementados, e obviamente, que conterá o método `Main()` também. No decorrer deste livro seguiremos fielmente essa regra nos nossos exemplos.

Palavras reservadas em C#

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

Formatando a saída padrão

A formatação da saída padrão é feita usando os chamados “caracteres de escape” (veja a tabela abaixo).

Vejamos um exemplo:

```
// \t = TAB
```

```
// \n = quebra de linha e retorno de carro (CR LF)
```

```
Console.WriteLine( "var1: {0} \t var2: {1}\t var3: {2}\n", var1, var2, var3);
```

Caractere de Escape Significado

\n Insere uma nova linha

\t TAB

\a Dispara o som de um alarme sonoro simples

\b Apaga o caractere anterior da string que está sendo escrita no

console(backspace)

\r Insere um retorno de carro

\0 Caractere NULL (nulo)

Recebendo parâmetros na linha de comando

Para receber parâmetros na linha de comando, ou seja, na chamada de um programa quando digitamos o nome do executável no prompt da linha de comando do DOS (como “ScanDisk /All /AutoFix”, por exemplo), o método Main() precisa ser declarado da seguinte forma:

```
// não retorna nenhum valor ao sistema
```

```
static void Main(string[ ] args)
```

ou,

```
// retorna um valor do tipo int ao sistema
```

```
static int Main(string[ ] args)
```

O parâmetro args é um array de strings que recebe os parâmetros passados quando a aplicação é chamada na linha de comando. A seguir mostramos uma das formas de varrer os parâmetros recebidos:

```
foreach (string cmd in args)
{
    int i = 0;
    Console.WriteLine("Par {0}: {1}", i, cmd);
    i++;
}
```

Para saber o número de argumentos que foram passados, usamos o método Length() do array args da seguinte forma:

```
numArgs = args.Length( );
```

Quando na linha de comando são recebidos parâmetros numéricos, estes devem ser convertidos de string para o tipo numérico respectivo usando a classe Convert. Exemplo:

```
Convert.ToInt32(varString)
```

Variáveis

São utilizadas para armazenar dados na memória do computador. Em C#, todas as variáveis são declaradas dentro do escopo de uma classe e podem ser dos seguintes tipos:

Locais: são declaradas no escopo de um método, indexador ou evento e não possuem modificadores de acesso. A sua declaração se limita ao tipo seguido do identificador da variável.

Atributos de uma classe ou campos da classe: a variável é declarada como membro de uma classe. A declaração deve ser efetuada como se segue:

[Modificador de acesso] [tipo atributo] <tipo da variável> <identificador>

Exemplo:

```
public class App
{
    public int varInt;
    static void Main( )
    {
        int varLocal;
    }
}
```

O Sistema de Tipos em C#

Em C#, todo tipo é derivado da classe System.Object, que constitui o núcleo do sistema de tipos de C#. Entretanto, os projetistas da linguagem, e não apenas da linguagem, mas de .NET como um todo, sabem perfeitamente das implicações de ter um sistema de tipos onde tudo é um objeto: queda de desempenho. Para resolver este problema, eles organizaram o sistema de tipos de duas formas:

Tipos Valor: variáveis deste tipo são alocadas na pilha e têm como classe base System.ValueType, que por sua vez deriva de System.Object.

Tipos Referência: variáveis deste tipo são alocadas na memória heap e têm a classe System.Object como classe base.

Boxing e UnBoxing

A razão de se criar uma origem comum de tipos é para facilitar a interação entre tipos valor e referência.

O processo de conversão explícita de um tipo valor para um tipo referência é conhecido em C# como Boxing (encaixotar).

O processo contrário a Boxing é conhecido como Unboxing. Nesse caso, o compilador verifica se o tipo valor a receber o conteúdo do tipo referência é equivalente a este último.

No processo de Boxing, o que de fato está ocorrendo é que um novo objeto está sendo alocado na memória heap e o conteúdo da variável de tipo valor é copiado para a área de memória referenciada por esse objeto.

Exemplo:

```
int intNumero = 10;
// Faz o boxing para o tipo referencia.
Object objNumero = intNumero;
// Faz o unboxing para o tipo valor
int intValor = (int)objNumero;
```

Quando essa operação é efetuada entre tipos que não são equivalentes, uma exceção é gerada pelo runtime.

Tipos Valor

Tipos valor não podem ser usados como classes base para criar novos tipos por que estes são implementados usando classes chamadas "seladas", a partir das quais não é possível implementar o mecanismo de herança.

Antes de serem usados, os tipos valor devem ser inicializados, caso contrário o compilador acusará um erro.

Os tipos valor são subdivididos em duas categorias:

1. Estruturas
2. Enumerados

Estruturas

Estruturas são usadas para implementar tipos simples chamados de primitivos em outras linguagens de programação,

são criadas na pilha e ainda oferecem muito do potencial de uma classe a um custo menor. Os seguintes tipos são implementados usando estruturas:

Tipos primitivos

Numéricos: inteiros, ponto flutuante e decimal

Booleanos: verdadeiro e falso

Tipos definidos pelo usuário: estruturas propriamente ditas que permitem que o usuário crie seus próprios tipos.

Enumerados

São usados para implementar listas de valores constantes, os quais podem ser de qualquer tipo inteiro (long, int etc.); porém não podem ser do tipo char. Cada constante tem um valor inteiro associado, o qual pode ser sobrescrito quando assim definido na lista enumerada. Os valores inteiros associados a cada constante da lista enumerada começam a partir de zero.

Tipos Referência

Os seguintes tipos referência fazem parte do namespace System e derivam diretamente do System.Object:

```
class
object
string
delegate
interface
```

Strings

O tipo string representa uma sequência de caracteres Unicode. string é um alias para System.String no .NET Framework.

```
string a = "hello";  
string b = "h";
```

O operador + concatena strings:

```
string a = "Bom " + "dia";
```

O [] operador acessa caracteres individuais de uma string:

```
char x = "teste"[2]; // x = 's';
```

O arroba (@) evita que sequência de escape sejam processadas:

```
@ "c:\Docs\Source\a.txt" // o mesmo que "c:\\Docs\\Source\\a.txt"
```

Operadores

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)

Operadores lógicos

Em C# temos os seguintes operadores lógicos:

Operador Descrição

&&	(E)
	(OU)

Instruções de Controle

Uma instrução de controle é usada para controlar o fluxo de execução do programa baseando-se em uma condição verdadeira ou falsa.

Instrução if

A declaração if seleciona uma declaração para a execução com base no valor de uma expressão Booleana. No exemplo a seguir um flag flagCheck Booleano é definido como verdadeiro e, em seguida, verificado no caso declaração.

```
bool flagCheck = true;
if (flagCheck == true)
{
    Console.WriteLine("O flag é verdadeiro.");
}
else
{
    Console.WriteLine("O flag é falso.");
}
```

Exercício:

```
using System;
class TesteIfElse
{
    static void Main()
    {
        Console.WriteLine("Digite um dos seguintes números: 1, 2, 3, 4");
        int a = Int32.Parse(Console.ReadLine());
        string mensagem = "Variável a igual: ";
        if (a==1)
        {
            Console.WriteLine(mensagem + a);
        }
        else if (a == 2)
        {
            Console.WriteLine(mensagem + a);
        }
        else if (a == 3)
        {
            Console.WriteLine(mensagem + a);
        }
        else
        {
            Console.WriteLine(mensagem + a);
        }
        Console.Read();
    }
}
```

Instrução switch

O switch é uma declaração de controle que trata múltiplas seleções e enumerações passando por um controle para um dos casos dentro do seu corpo como o seguinte exemplo:

```
int caseSwitch = 1;
switch (caseSwitch)
{
    case 1:
        Console.WriteLine("Case 1");
        break;
    case 2:
        Console.WriteLine("Case 2");
        break;
    default:
        Console.WriteLine("Default case");
        break;
}
```

Exercício

```
using System;
class Testeswitch
{
    static void Main()
    {
        Console.WriteLine("Digite um dos seguintes números: 1, 2, 3, 4, 5");
        int a = Int32.Parse(Console.ReadLine());
        string mensagem = "Variável a igual: ";
        switch (a)
        {
            case 1:
                Console.WriteLine(mensagem + "1");
                break;
            case 2:
                Console.WriteLine(mensagem + "2");
                break;
            case 3:
                goto case 1;
            case 4:
            case 5:
                Console.WriteLine(mensagem + a);
                break;
            default:
                Console.WriteLine(mensagem + a);
                break;
        }
        Console.Read();
    }
}
```


Instruções de Iteração

As instruções de iteração permitem repetir um determinado trecho do algoritmo de acordo com uma condição (iteração condicional) ou de acordo com um número de vezes fixo (iteração definida).

Instrução for

O loop for executa uma instrução ou um bloco de instruções repetidamente até que uma expressão especificada é avaliada como false. O loop for é útil para iterar em matrizes e para processamento seqüencial.

Exemplo:

```
class ForLoopTest
{
    static void Main()
    {
        for (int i = 1; i <= 5; i++)
        {
            Console.WriteLine(i);
        }
    }
}
/*
Output:
1
2
...
5
*/
```

Exercício

```
using System;
class foraninhado
{
    static void Main()
    {
        for (int i = 1; i <= 10; i++)
        {
            for (int j = 1; j <= 10; j++)
            {
                Console.WriteLine(j.ToString());
            }
            Console.WriteLine(i.ToString());
        }
        Console.Read();
    }
}
```

Instrução while

A instrução while executa uma instrução ou um Bloquear de instruções até que uma expressão especificada é avaliada como false.

```
class WhileTest
{
    static void Main()
    {
        int n = 1;
        while (n < 5)
        {
            Console.WriteLine("Current value of n is {0}", n);
            n++;
        }
    }
}
/*
Output:
Current value of n is 1
Current value of n is 2
Current value of n is 3
Current value of n is 4
*/
```

Instrução do .. while

O laço do..while (faça..enquanto) é usado quando queremos repetir uma instrução ou bloco de instruções ENQUANTO uma condição for satisfatória. A única diferença entre este laço e o laço while, é que, no laço do..while a condição de parada é testada após a iteração, o que garante que o laço do..while será executado no mínimo uma vez. No laço while a condição é testada antes da iteração.

```
public class TestDoWhile
{
    public static void Main ()
    {
        int x = 0;
        do
        {
            Console.WriteLine(x);
            x++;
        } while (x < 5);
    }
}
/* Output:
0
1
2
3
4
*/
```

Instrução for each

Esta instrução fornece uma maneira simples e limpa para iterar Através os elementos de uma matriz. Através exemplo, o código a seguir cria uma matriz chamada numbers e itera-lo com a instrução foreach:

```
// cs_foreach.cs
class ForEachTest
{
    static void Main(string[] args)
    {
        int[] fibarray = new int[] { 0, 1, 2, 3, 5, 8, 13 };
        foreach (int i in fibarray)
        {
            System.Console.WriteLine(i);
        }
    }
}

0 1 2 3 5 8 13
```

break

O comando break é usado para a saída de laços (while, for , switch, etc..) .

Exemplo:

```
using System;
class BreakTest
{
    public static void Main()
    {
        for (int i = 1; i <= 100; i++)
        {
            if (i == 5)
                break;
            Console.WriteLine(i);
        }
    }
}
```

Saída

```
1
2
3
4
```

Continue

O comando continue também é usado em laços(while, for, etc.) quando em execução o comando continue irá mover a execução para o próxima iteração no laço sem executar as linhas de código depois de continue.

Exemplo:

```
using System;
class ContinueTest
{
    public static void Main()
    {
        for (int i = 1; i <= 10; i++)
        {
            if (i < 9)
                continue;
            Console.WriteLine(i);
        }
    }
}
```

Saída

```
9
10
```

Programação Orientada a Objetos

Uma vez que o Framework .NET é fortemente baseado em conceitos de orientação a objetos, também a linguagem C# que utilizaremos nesta apostila para a criação de páginas ASP.NET é baseada nos conceitos de orientação a objetos; é bastante oportuno que revisemos os seguintes conceitos:

- O que é um objeto?
- O que são Classes?
- Métodos e propriedades
- Herança
- Polimorfismo e Funções Virtuais
- Encapsulamento e Visibilidade

Todo o Framework .NET foi criado com base nos conceitos de Orientação a Objetos. A linguagem C# é totalmente orientada a objetos, onde tudo são classes. Claro que alguns "puristas" poderiam dizer que o C# não é completamente orientado a objetos porque não suporta herança múltipla. Na verdade veremos que o C# foi projetado para conciliar os benefícios da orientação a objetos e ao mesmo tempo ser simples e prático. Para isso foram eliminadas algumas características da orientação a objetos, características estas que mais causavam problemas do que propriamente forneciam soluções.

Objeto

Um objeto é uma entidade que contém, além dos dados, todas as funções (métodos) que atuam sobre estes dados. Ou seja, um objeto é composto dos dados que descrevem o objeto (propriedades) e das operações que podem ser realizadas sobre estes dados (métodos). Esta abordagem já é um pouco mais próxima da abordagem entendida pelos usuários finais do

sistema. Vamos imaginar um sistema para uma Escola de Nível Médio. Os usuários falam em termos de alunos, séries, etc. Poderíamos ter um objeto chamado Alunos. Este objeto poderia conter diversas propriedades, tais como:

Matrícula.
Nome.
Endereço.
Data de Entrada.
Nome do Pai.
Nome da Mãe.
Número da Identidade.
Número do CPF.

Com isso podemos observar que as propriedades descrevem as características de um determinado objeto. O conjunto de valores contidos nas propriedades de um determinado objeto define o seu estado atual.

Além das propriedades o objeto pode conter métodos. Os métodos descrevem ações que podem ser realizadas pelo objeto ou no objeto. Por exemplo, o nosso objeto Alunos poderia ter um método chamado Recebimento, outro chamado transferência, mais um chamado Promoção e assim por diante. Um método, na prática, é uma função ou procedimento que realiza uma série de ações. Os métodos de um objeto podem receber parâmetros e ter o seu

comportamento alterado, dependendo do valor dos parâmetros. Por exemplo, o método Promoção de um objeto Alunos pode receber, como parâmetros, a Matrícula do Aluno, a data da promoção e o código do novo curso em que aluno estará cursando.

Dentro do método Promoção pode ser chamado um método AtualizaPagamento, o qual atualiza o valor das mensalidades dos alunos, de acordo com a nova série que o mesmo irá cursar.

Classes

Se pesquisarmos a bibliografia sobre orientação a objetos encontraremos um mundo de definições para classes. Vamos inicialmente apresentar algumas das definições formais encontradas na bibliografia. Depois vamos a uma, digamos, explicação mais fácil:

- 1) Classes constituem modelos que são utilizados para a criação de objetos. Nas classes são descritas a estrutura de dados (através das propriedades) e o comportamento (através de seus métodos) de um ou mais objetos similares que possuem seus dados estruturados da mesma forma e são manipulados pelos mesmos métodos.
- 2) Um objeto é uma instância de uma classe que é criada em tempo de execução. Classes são puramente uma descrição estática de um conjunto de possíveis objetos.

Na prática, o que significa, por exemplo, termos uma classe chamada Alunos? Esta classe serve como modelo para a criação de objetos do tipo Aluno. Na classe Alunos estão as definições das propriedades e dos métodos para um objeto Aluno. Ou seja, sempre que criarmos um objeto do tipo Aluno, o mesmo será criado com todas as propriedades e métodos da classe Alunos.

"Classe é uma forma para fazer objetos". (Pensamento de um Programador → Google.)

Grande parte da funcionalidade do Framework .NET é fornecida por um grande número de classes, as quais fazem parte de ".NET Framework Class Library", já descrita anteriormente. O Framework .NET agrupa as classes de acordo com suas funcionalidades. Um agrupamento de classes criadas para um determinado fim é também conhecido como um namespace (espaço de nomes). Por exemplo, temos o namespace System.Data. Dentro deste namespace existem várias classes que fornecem os métodos necessários para a conexão e manipulação de fontes variadas de dados.

Existem classes para a conexão com o SQL Server 2000, outras para a conexão com fontes ODBC e assim por diante.

A biblioteca de classes do Framework .NET é organizada de uma forma hierárquica, onde as classes de níveis inferiores herdam todas as características da classe mãe. Falaremos mais sobre herança no próximo item.

Exemplo:

Declarando classes

```
class ClasseTeste
{
    // Métodos, propriedades, campos e eventos
}
```

Para criar uma instância:

```
Tipo Referência = new Tipo();
```

No nosso exemplo:

```
ClasseTeste teste = new ClasseTeste();
```

Podemos criar uma referência sem criar um objeto:

```
Tipo Referência;
ClasseTeste teste;
```

Modificadores Para os Membros de uma Classe

Conforme descrito anteriormente existem modificadores que definem a visibilidade dos membros de uma classe. Por exemplo, o modificador `public` torna o membro acessível de fora da classe; já o modificador `private` torna o membro disponível somente dentro da própria classe. A seguir uma descrição dos principais modificadores.

- **public:** Torna o membro acessível de fora da definição da classe.
- **protected:** O membro não pode ser acessado fora da classe, porém o membro está disponível para outras classes derivadas da classe base.
- **private:** O membro não pode ser acessado fora da classe, nem mesmo por outras classes derivadas da classe base.
- **internal:** O membro somente é visível na unidade de código onde o mesmo está definido. É um meio-termo entre `public` e `protected`, uma vez que o membro pode ser acessado por todas as classes definidas na mesma unidade (público para as classes da mesma unidade), porém não pode ser acessado por classes definidas em outras unidades (`protected` para unidades definidas em outras unidades).

Estruturas (Structs)

Um tipo `struct` é um tipo de valor normalmente usado para encapsular pequenos grupos de variáveis relacionadas, como as coordenadas de um retângulo ou as características de um item em um inventário. O exemplo a seguir mostra uma declaração de `struct` simples:

```
public struct Livro
{
    public decimal preco;
    public string titulo;
    public string autor;
}
```

As estruturas também podem conter construtores, constantes, campos, métodos, propriedades, indexadores, operadores, eventos e aninhados tipos, embora se vários membros forem necessários, você deve considerar tornar o tipo de uma classe em vez disso.

As estruturas podem implementar uma interface, mas elas não herdam de outra estrutura. Por esse motivo, `struct` membros não podem ser declarados como protegido.

Criando Uma Classe em C# no Visual Studio

Visando exemplificar a criação de Classes na linguagem C#, esteja com o ambiente do VisualStudio em execução, procedendo da seguinte forma:

Passo 1

Acessando o Menu File | New | Project, selecione em Templates **Class Library**. Desta maneira estamos iniciando um projeto para acomodar nosso exemplo de uso de Classes, na forma de um Assembly, ou seja um aplicativo .Net (byte Code).

Preencha na caixa de dialogo representada pela **Figura 01** os campos **Name**, **Location** e **Solution Name** conforme indicado. O campo **Name** representa o nome da NameSpace que acomodara as Classes do projeto de exemplo, **Location** refere-se ao diretório onde os fontes do projeto serão guardados e **Solution Name** o nome do projeto em si.

Por fim clique no botão **Ok**.

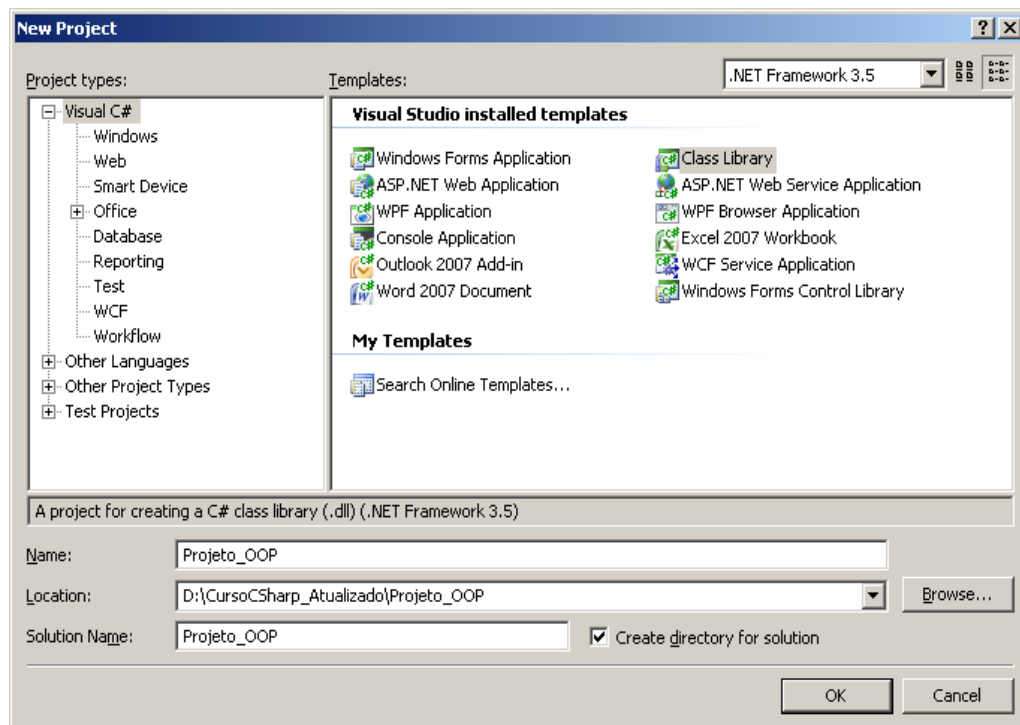


Figura 01

Passo 2

Como resultado do passo anterior, teremos na janela Solution Explorer representada pela **Figura 02** exibindo as Classes do nosso projeto, até o momento somente a Classe inicial **Calss1**, representada pelo arquivo **Class1.cs**.

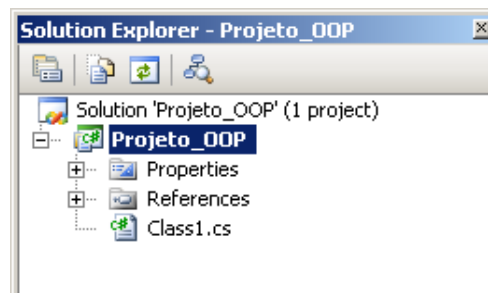


Figura 02

Outra observação, se refere ao nome da Classe, aqui Class1, que conforme exibido na **Listagem 01**, promoveremos imediatamente uma alteração para Produto. Proceda como exibido nas **Figura 03** e **Figura 04**, pois desta forma, não só o nome da Classe será alterada, como o nome do arquivo que a representa na Solution Explore também.

Listagem 01

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

```
namespace Projeto_OOP  
{  
    public class Class1  
    {  
    }  
}
```

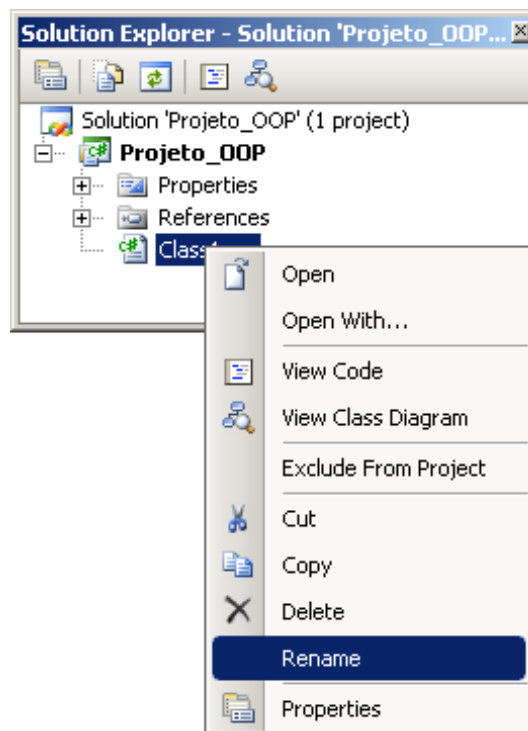


Figura 03

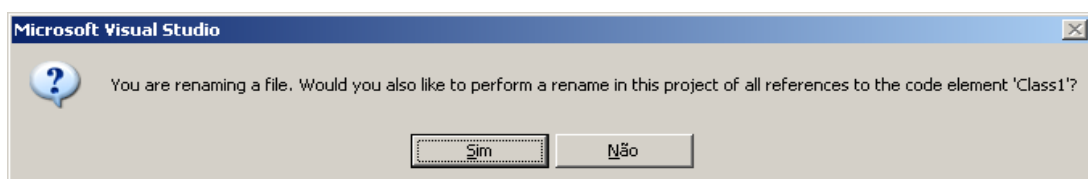


Figura 04

O resultado final pode ser observado na **Listagem 02** com o nome da Classe já alterado para **Produto**.

Listagem 02

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

namespace Projeto_OOP
{
    public class Produto
    {
    }
}
```

Aqui vamos nos deter somente no desenvolvimento da Classe e seus aspectos, deixando as explicações mais aprofundadas sobre o conceito de Namespace para o tópico na sequência que trata do assunto. Mas para não ficarmos no vazio, consideremos Namespace como um agrupamento (lógico) de Classes. Para este exemplo, uma referência para nossa Classe Produto, seria algo como: Projeto_OOP.Produto.

Com referência as quatro declarações **using** no início da **Listagem 02**, são referências as Namespaces do Framework .Net, que o ambiente do VisualStudio acrescentou julgando úteis no desenvolvimento do código que se seguirá. Essas e outras Namespace do Framework serão necessárias para que possamos fazer uso pleno da linguagem C#.

Campos (Fields)

Um campo é uma variável de qualquer tipo que é declarado diretamente em uma Classe ou Struct. Campos são membros da Classe compondo um modelo.

[modificador] tipo nome;

Exemplo:

```
public class Calendario
{
    public string dia = "Domingo";
    //...
}
```

Propriedades (Properties)

Em C#, uma propriedade é um membro nomeado de uma classe, struct ou interface oferecendo uma maneira sistemática de acessar campos particulares através do que é chamado o get e set, ou seja, métodos acessores.

Exemplo:

```
class Pessoa
{
    private string nome;
    public string Nome
    {
        get { return nome; }
        set { nome = value; }
    }

    private int idade;
    public int Idade
    {
        get { return idade; }
        set { idade = value; }
    }
}
```

Constantes (Constants)

A palavra-chave const é usada para modificar uma declaração de um campo ou variável local. Ela especifica que o valor do campo ou a variável local é constante, o que significa que ele não pode ser modificado.

Exemplo:

```
public class TesteConstante
{
    static void Main()
    {
        const int c = 1984;
        Console.WriteLine("Minha constante = {0}", c);
    }
}
```

// Saída: Minha constante = 1984

Acrescentando Membros (Fields e Properties) a Classe de Exemplo

Visando exemplificar a criação de Classes na linguagem C#, esteja com o ambiente do VisualStudio em execução, procedendo da seguinte forma:

Passo 1

Sobre a janela Solution Explorer do projeto Projeto_OOP, clique no nó Produto.cs com o botão direito do mouse, selecionando View Class Diagram conforme Figura 01. Este procedimento cria um módulo diagrama permitindo não só visualizar a Classe Produto em forma de componente, mas acrescentar Membros sem que seja necessário implementa-los manualmente no código fonte.

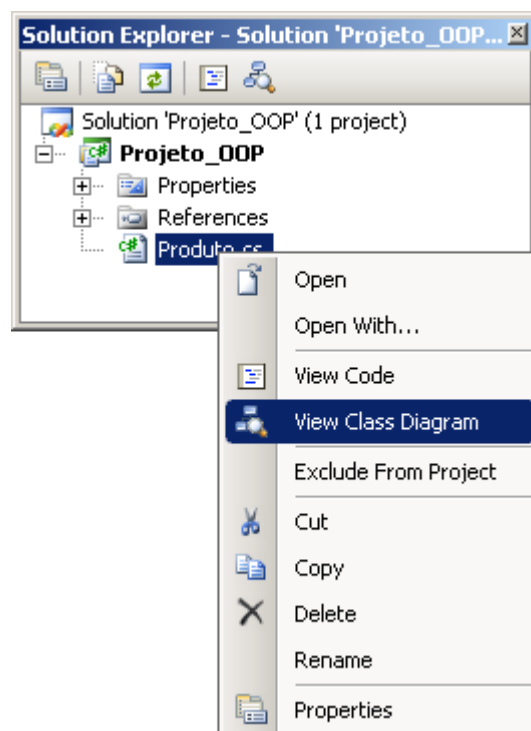


Figura 01

O resultado é a representação gráfica da Classe Produto conforme imagem da Figura 02.



Figura 02

Passo 2

Vamos agora definir os campos privados e seus métodos set get, públicos, possibilitando leitura e atribuição de valores. Se baseie na Figura 03 para acrescentar os campos privados na Classe Produto, clicando com o botão direito do mouse sobre a representação da Classe Produto no Diagrama.

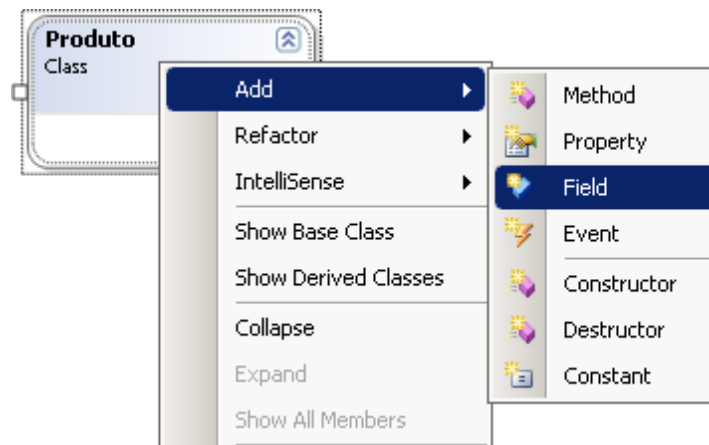


Figura 03

Em seguida proceda como observado na Figura 04, definindo na janela Class Details o tipo long (inteiro longo) para o campo _id.

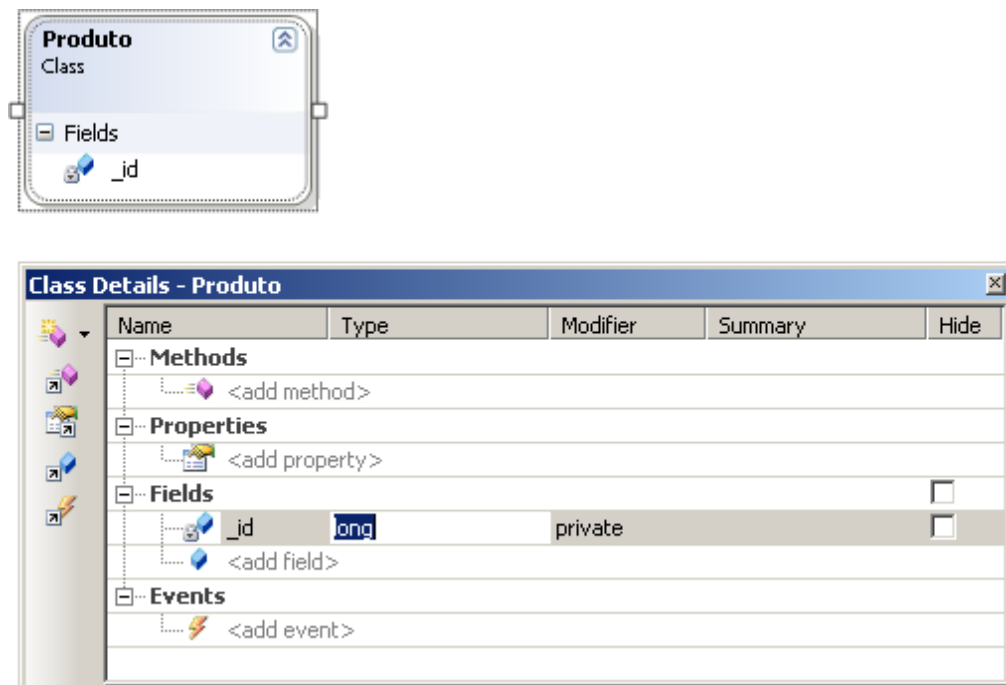


Figura 04

Encerre esta etapa repetindo os passos anteriores para criar os campos `_descricao` de tipo `string` e `_preco` de tipo `decimal`. A representação gráfica da Classe `Produto` deverá estar semelhante a Figura 05.

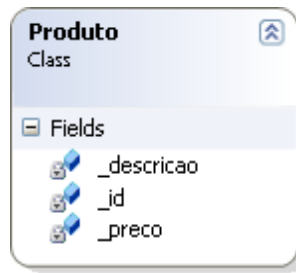


Figura 05

Perceba como ficou o código fonte após adicionarmos campos privados em nossa Classe `Produto`, representado a seguir pela Listagem 01.

Listagem 01

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

namespace Projeto_OOP
{
    public class Produto
    {
        private long _id;
        private string _descricao;
        private decimal _preco;
    }
}
```

Passo 3

Objetivando criar os métodos acessores aos campos privados, orientando-se pela Figura 06, selecione individualmente cada membro Field no Diagrama da Classe Produto. Para tanto clique com o botão direito do mouse sobre o campo `_descricao`, escolhendo no menu de contexto Refactor | Encapsulate Field..., confirmando as etapas na sequência da Figura 06, 07 e 08.

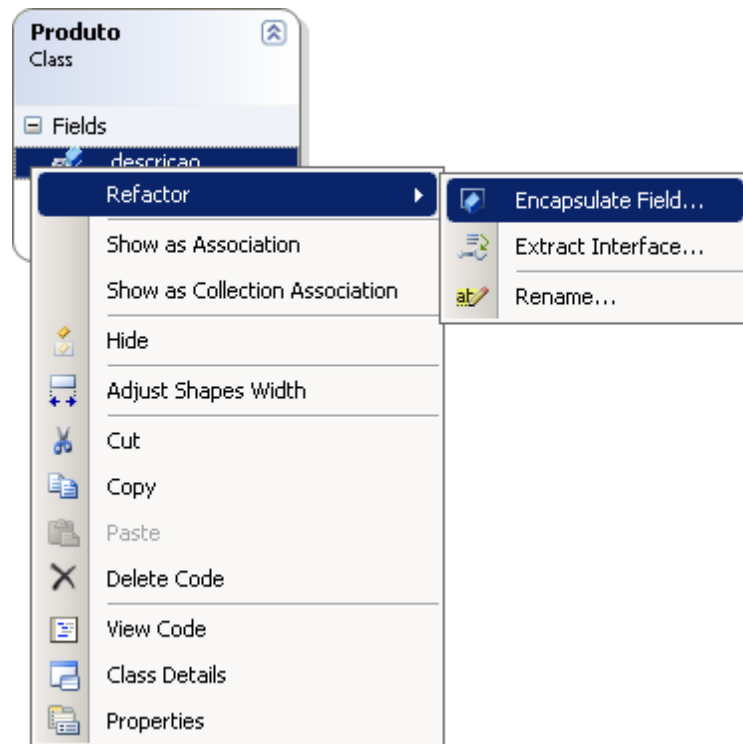


Figura 06

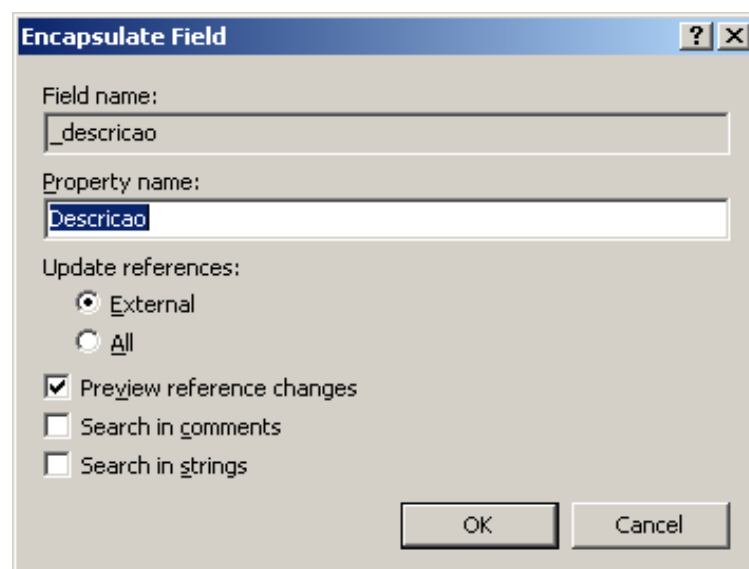


Figura 07

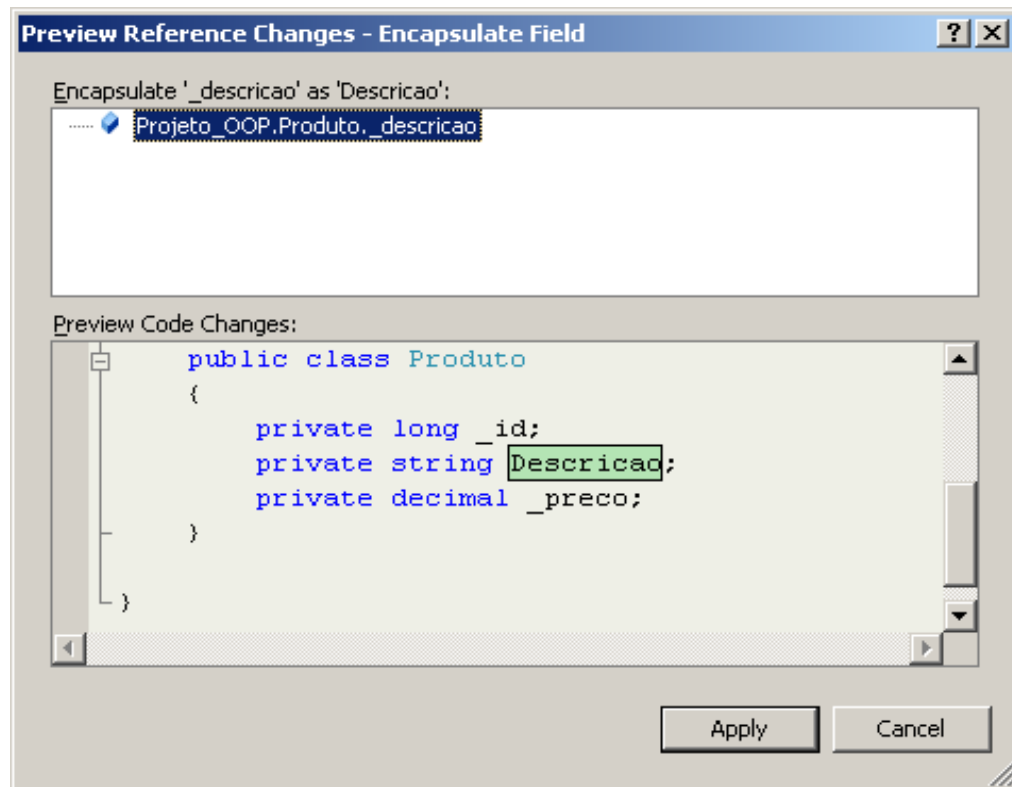


Figura 08

Complete o processo repetindo a etapa para os campos `_id` e `_preco`. Por consequência, teremos a representação gráfica da Classe conforme a imagem da Figura 09 e o código fonte implementando os métodos set e get conforme demonstrado na Listagem 02.

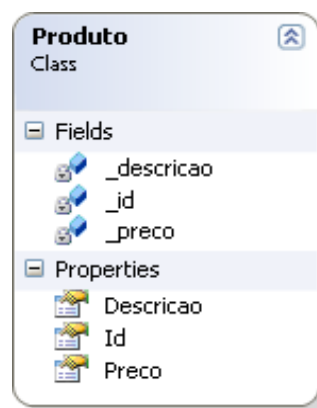


Figura 09

Listagem 02

```
public class Produto
```

```
{  
    private long _id;  
  
    public long Id  
    {  
        get { return _id; }  
        set { _id = value; }  
    }  
  
    private string _descricao;  
  
    public string Descricao  
    {  
        get { return _descricao; }  
        set { _descricao = value; }  
    }  
  
    private decimal _preco;  
  
    public decimal Preco  
    {  
        get { return _preco; }  
        set { _preco = value; }  
    }  
}
```

Construtores (Constructs) e Destrutores (Destructors)

Ao criarmos uma classe podemos definir um método que será executado toda vez que uma instância da classe for criada. Este método especial é chamado de Construtor (Constructor) da classe. Colocamos neste método todo o código que deve ser executado na criação da classe. Por exemplo, se estamos criando uma classe que possui métodos para fazer pesquisas em um banco de dados do SQL Server 2000. No método Construtor podemos incluir o código que estabelece a conexão com o banco de dados. Para que a classe seja de uso genérico, podemos definir parâmetros a serem passados para o método Construtor, de tal forma que a conexão seja estabelecida de acordo com o valor dos parâmetros passados para o método construtor. Se não definirmos um método construtor explicitamente, o Framework .NET define e exige um método construtor sem parâmetros.

Exemplo:

```
public class Taxi
{
    public bool isInitialized;
    public Taxi()
    {
        isInitialized = true;
    }
}

class TestTaxi
{
    static void Main()
    {
        Taxi t = new Taxi();
        Console.WriteLine(t.isInitialized);
    }
}
```

Exercício

```
using System;

public class Livro
{
    public string titulo;
    public string autor;

    public Livro()
    {
        titulo = "ASP.NET com C#";
        autor = "Alfredo Lotar";
    }

    public Livro(string tit,string aut)
    {
        titulo = tit;
        autor = aut;
    }
}
```

```
class TesteLivro
{
    static void Main()
    {
        Livro lv = new Livro();
        Console.WriteLine(lv.titulo);
        Console.WriteLine(lv.autor);
        Console.Read();
    }
}
```

Para chamar um destrutor, o C# usa um **~ (tio)** antes do nome do construtor (sem parâmetros). Ao atribuir **null** a um objeto, o garbage collector não é executado no mesmo momento, o que significa que o objeto continua na memória e o método destrutor não é chamado na mesma hora. Não é possível determinar quando o garbage collector é chamado, portanto é sempre aconselhável evitar o uso de destrutores.

Exemplo:

```
class Car
{
    ~Car() // destructor
    {
        // cleanup statements...
    }
}
```

Acrescentando Métodos Construtores na Classe de Exemplo

Para tornarmos nossa Classe “instanciável”, ou seja ser executável em um aplicativo cliente (inclusive por outras Classes dentro ou fora da NameSpace), devemos adicionar os métodos abaixo em nossa Classe de exemplo.

Particularmente este método construtor não implementa nenhuma funcionalidade, e naturalmente não exige nenhum parâmetro.

```
public Produto()  
{  
}
```

Namespaces

A palavra-chave namespace é usada para declarar um escopo. Este escopo de namespace permite organizar o código (Classes) e oferece uma maneira para criar tipos globalmente exclusivos.

A palavra-chave using pode ser usada para que o nome completo não seja necessário.

Exemplo:

```
using System;

namespace NamespaceSamples
{
    class ClassePrimeira
    {
        public void MetodoTeste()
        {
            Console.WriteLine(" MetodoTeste inside NamespaceSamples");
        }
    }
}
```

Partial

É possível dividir a definição de uma Classe ou uma Struct, interface ou método em dois ou mais arquivos fonte. Cada arquivo contém uma seção para definição de tipo ou método, e todas as partes são combinadas quando a aplicação é compilada.

Exemplo:

```
public partial class Produto
{
    private long _id
    private string _descricao
    private decimal _preco
    //Métodos get e set omitidos
}

public partial class Produto
{
    public void IncluirProduto()
    {
        //código omitido
    }

    public DataTable ObterProduto(long id)
    {
        //código omitido
    }
}
```

Alterando a Classe de Exemplo para Suportar Partial Class

Basicamente devemos adicionar um outro arquivo (Class) ao projeto, nomeando por exemplo como ProdutoBL.cs (caracterizando a função de acomodar as Regras de Negócio – Métodos).

Passo 1

Buscando adicionar a nova Classe, clique com o botão direito do mouse sobre o nó que representa o projeto, em seguida, no menu de contexto selecione Add |New Item. Na sequência, já com a caixa de dialogo Add New Item exibida (Figura 01), selecione o Template Class nomeando-o como ProdutoBL.cs.

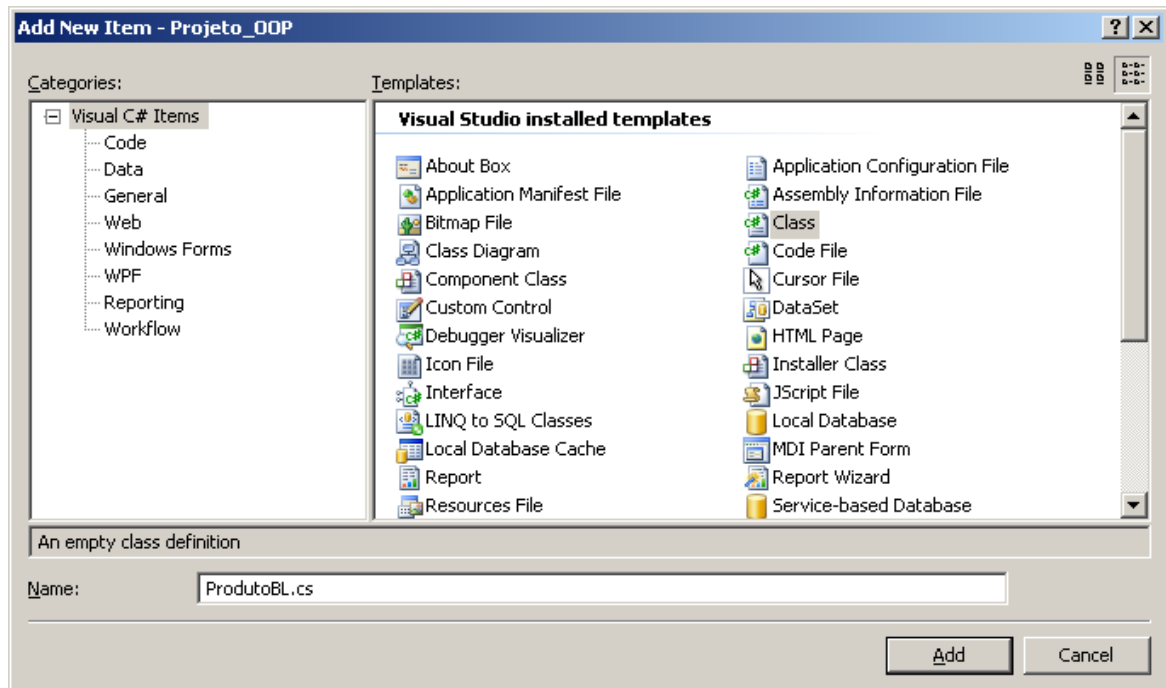


Figura 01

Passo 2

Repare o resultado do código gerado para a nova Classe, observando a Listagem 01.

Listagem 01

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

namespace Projeto_OOP
{
    class ProdutoBL
    {
    }
}
```

Passo 3

Altere a assinatura da classe para que suporte Partial Class, conforme trecho em negrito na Listagem 02.

Finalizando, promova também alteração na Classe Produto onde temos o modelo de dados (Membros), para que possa fazer parte de uma única Classe Produto, ainda que assinada como Partial. A Listagem 03 exibe o código em negrito que deve ser alterado na Classe.

Listagem 02

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

```
namespace Projeto_OOP  
{  
    public partial class Produto  
    {  
    }  
}
```

Listagem 03

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

```
namespace Projeto_OOP  
{  
    public partial class Produto  
    {  
        private long _id;  
  
        public long Id  
        {  
            get { return _id; }  
            set { _id = value; }  
        }  
  
        private string _descricao;  
  
        public string Descricao  
        {  
            get { return _descricao; }  
            set { _descricao = value; }  
        }  
  
        private decimal _preco;  
  
        public decimal Preco  
        {  
            get { return _preco; }  
            set { _preco = value; }  
        }  
  
        public Produto()  
        {  
        }  
    }  
}
```

Métodos (Methods)

Um método contém código que atua sobre os dados do objeto. Os métodos, falando em termos de Orientação a Objetos, descrevem as ações que podem ser realizadas pela classe.

Na prática, a funcionalidade de uma classe é implementada através dos seus métodos.

Exemplo: [modificador] tipo_de_retorno nome (tipo parâmetro)

```
public class Operacoes
{
    public int Soma(int n1, int n2)
    { return n1 + n2; }
}
```

Modificadores de Acesso de Métodos

Tipos e membros de tipo têm um nível de acessibilidade, que controla se que possam ser usados no outro código no seu Assembly ou outros assemblies. Você especificar a acessibilidade de um tipo ou membro quando você declará-lo usando um desses modificadores de acesso:

Modificador	Descrição
public	O tipo ou membro pode ser acessado por qualquer outro código no mesmo assembly ou outro conjunto que faz referência a ele.
private	O tipo ou membro só pode ser acessado pelo código na mesma classe ou struct.
protected	O tipo ou membro só pode ser acessado pelo código no mesmo classe ou struct, ou em uma classe derivada.
internal	O tipo ou membro pode ser acessado por qualquer código no mesmo assembly, mas não no outro conjunto.
protected internal	O tipo ou membro pode ser acessado por qualquer código no mesmo assembly, ou por qualquer classe derivada em outro assembly.

Exemplo:

```
// public class:
public class Tricycle
{
    // protected method:
    protected void Pedal() { }

    // private field:
    private int wheels = 3;

    // protected internal property:
    protected internal int Wheels
    {
        get { return wheels; }
    }
}
```


Implementando Métodos de Negócio na Classe Produto

Implementaremos métodos de negócio na Classe Produto, lançando mão da parte da Classe representada pelo arquivo ProdutoBL.cs, fazendo claramente uma divisão das funções de ação da Classe, com relação ao modelo e os métodos construtores.

Passo 1

Criar um método para funcionalidade InserirProduto, sem parâmetros e que retorne uma mensagem de sucesso ou não. Confira a Listagem 01.

Listagem 01

```
public string InserirProduto()
{
    //Código omitido aqui
    return "Produto inserido com sucesso.";
}
```

Passo 2

Criar um método para exclusão de Produto, que implemente um parâmetro do tipo long. Confira na Listagem 03 o código.

Listagem 02

```
public void ExcluirProduto(long id)
{
    //Código omitido aqui
}
```

Passo 3

Criar um método que retorne um conjunto de registros (DataTable), implementando um parâmetro para pesquisa pelas iniciais da descrição de Produto.

Ressaltamos aqui, a necessidade de adicionarmos uma referência para a NameSpace System.Data para podermos utilizar o tipo DataTable. Para tanto, adicione então using System.Data; na área superior do código fonte.

Listagem 03

```
public DataTable ListaProduto(string descricao)
{
    DataTable DT = new DataTable();
    //Código omitido aqui
    return DT;
}
```

Encapsulamento

Encapsulamento vem de encapsular, que em programação orientada a objetos significa separar o programa em partes, o mais isoladas possível. A ideia é tornar o software mais flexível, fácil de modificar e de criar novas implementações. Uma grande vantagem do encapsulamento é que toda parte encapsulada pode ser modificada sem que os usuários da classe em questão sejam afetados. O encapsulamento protege o acesso direto (referência) aos atributos de uma instância fora da classe onde estes foram declarados. Esta proteção consiste em se usar modificadores de acesso mais restritivos sobre os atributos definidos na classe. Depois devem ser criados métodos para manipular de forma indireta os atributos da classe.

Herança

“É o mecanismo que permite definir uma nova classe a partir de uma classe já existente. A classe que está sendo criada é dita subclasse ou classe filha da classe já existente. Em contrapartida a classe já existente é chamada de superclasse da classe que está sendo criada. A subclasse herda a estrutura de dados e os métodos da superclasse, podendo adicionar variáveis na estrutura de dados herdada, bem como adicionar novos métodos e reescrever métodos herdados. Uma classe pode possuir uma única superclasse – herança simples, ou pode conter mais do que uma superclasse – herança múltipla. A herança múltipla tem sido alvo de muitas discussões e controvérsias. A única linguagem, do Framework .NET, que implementa diretamente a herança múltipla é o C++. A grande discussão em torno da herança múltipla tem a ver com a relação custo x benefício, uma vez que a mesma é de difícil implementação e concepção, embora os benefícios nem sempre sejam os esperados.

Herança é um conceito fundamental para a orientação a objetos. Através do mecanismo de herança podemos criar uma classe baseada em outra já existente. A nova classe que está sendo criada “herda” todas as propriedades e métodos da classe base, também chamada de classe mãe ou superclasse conforme descrito anteriormente. A herança evita que as propriedades e métodos da classe mãe tenham que ser redefinidos na classe filho, embora a classe filho ou subclasse possa redefinir os métodos da classe mãe, através de um mecanismo conhecido como Overwrite.

Para ilustrar o mecanismo de herança vamos a um exemplo prático. Vamos imaginar que você esteja projetando um programa – baseado na orientação a objetos, evidentemente, para uma Mercadoria. Uma das prováveis classes seria a classe Clientes. Nesta classe poderíamos definir as características e métodos básicos para um Cliente básico.

Para a classe Clientes poderíamos definir as seguintes propriedades:

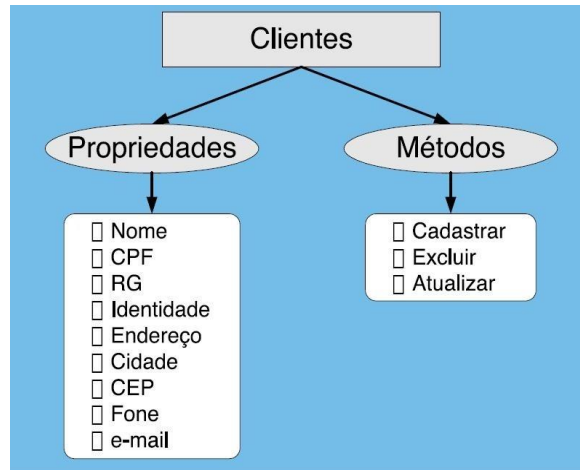
Nome
CPF
RG
Identidade
Endereço
Cidade
CEP
Fone
E-Mail
Crédito

Para a classe Clientes poderíamos definir os seguintes métodos:

- Cadastrar
- Excluir
- Atualizar

Observe que a nossa classe Clientes possui apenas as propriedades e métodos comuns a qualquer Cliente, Este é um dos princípios da utilização de classes: nas classes de primeiro nível definimos apenas as propriedades e métodos comuns, os quais deverão ser utilizados pelas classes dos demais níveis.

Na Figura abaixo temos uma visão geral da classe Clientes.



Exemplo de Herança

```
using System;
public class Conta
{
    public virtual double Calculo()
    {
        return (6*15);
    }
}

public class Conta1 : Conta
{
    public override double Calculo()
    {
        return (6*5);
    }
}
```

Modificadores de classe

Os seguintes modificadores podem ser usados em classes, no C#:

Modificador	Descrição
new	é usado em classes aninhadas. Se a classe tiver o mesmo nome da classe externa, devemos usar esse moderador para indicar a sobreposição.
sealed	a classe não pode ter classes derivadas.
abstract	a classe não pode ser instanciada.

Modificadores que devem ser para ocultar ou redefinir métodos da classe-base.

Modificador	Descrição
abstract	Indica que o método não possui implementação.
override	Redefine um método definido como virtual.
virtual	Indica que o método pode ser redefinido na classe derivada.
new	Indica que o método deixa de ser virtual em uma classe derivada.

Virtual

A palavra-chave virtual é usada para modificar um método, propriedade, indexador ou declaração de evento e permitir que ser substituído em uma classe derivada. Por exemplo, esse método pode ser substituído por qualquer classe que herda a ele:

```
public virtual double Area()  
{ return x * y; }
```

Base

A palavra-chave base é usada para acessar membros da classe base da dentro uma classe derivada:

- Chame um método na classe base que foi substituído por outro método.
- Especifica o construtor de classe base deve ser chamado durante a criação de instâncias da classe derivada.

Um acesso de classe base é permitido apenas em um construtor, um método de instância ou um acessador de propriedade de instância.

Exemplo:

```
public class Person  
{  
    protected string ssn = "444-55-6666";  
    protected string name = "John L. Malgraine";  
    public virtual void GetInfo()  
    {  
        Console.WriteLine("Name: {0}", name);  
        Console.WriteLine("SSN: {0}", ssn);  
    }  
}
```

```
class Employee : Person
{
    public string id = "ABC567EFG";
    public override void GetInfo()
    {
        // Calling the base class GetInfo method:
        base.GetInfo();
        Console.WriteLine("Employee ID: {0}", id);
    }
}

class TestClass
{
    static void Main()
    {
        Employee E = new Employee();
        E.GetInfo();
    }
}

/*
Saída
Name: John L. Malgraine
SSN: 444-55-6666
Employee ID: ABC567EFG
*/
```

Override

O modificador de override é necessário para estender ou modificar a implementação abstrata ou virtual de um método herdado, propriedade, indexador ou evento.

Exemplo:

```
abstract class ShapesClass
{
    abstract public int Area();
}

class Square : ShapesClass
{
    int side = 0;

    public Square(int n)
    {
        side = n;
    }

    // Area method is required to avoid
    // a compile-time error.
    public override int Area()
    {
        return side * side;
    }
}
```

```
static void Main()
{
    Square sq = new Square(12);
    Console.WriteLine("Area of the square = {0}", sq.Area());
}

interface I
{
    void M();
}
abstract class C : I
{
    public abstract void M();
}

}
// Saída: Area of the square = 144
```

Um método override fornece uma implementação nova de um membro que é herdada de uma classe base. O método que é substituído por uma declaração de override é conhecido como o método base substituído. O método de base substituído deve ter a mesma assinatura que o método override.

Você não pode substituir um método ou virtual não-estático. O método substituído base deve ser virtual, abstract ou override.

Uma declaração de override não é possível alterar a acessibilidade do método virtual. Tanto o método override e o método virtual devem ter o mesmo acessar modificador nível.

Você não pode usar a new, static, virtual ou abstract modificadores para modificar um método override.

Uma declaração de propriedade substituição deve especificar exatamente o mesmo modificador de Acessar, tipo e nome como a propriedade herdada, e a propriedade substituída deve ser virtual, abstract ou override.

Interfaces

Uma interface é um tipo de referência que é um pouco semelhante a uma classe base abstrata que consiste apenas membros abstratos. Quando uma classe deriva de uma interface, ele deve fornecer uma implementação para todos membros da interface. Uma classe pode implementar múltiplas interfaces mesmo que ele possa herdar da apenas uma única direta classe base.

Interfaces são usadas para definir recursos específicos para classes que não têm necessariamente uma relação "é um". Por exemplo, a interface de `IEquatable<T>` pode ser implementada por qualquer classe ou struct que deve permitir a determinar se dois objetos do tipo são equivalentes (no entanto, o tipo define equivalência) de código do cliente. `IEquatable<Of <T>>>` não implica o mesmo tipo de "é um" relação que existe entre uma classe base e uma classe derivada (por exemplo, um Cachorro é uma Animal).

Exemplo:

```
interface IMyInterface
{
    void MethodToImplement();
}
```

Classe implementando interface:

```
class InterfaceImplementer : IMyInterface
{
    static void Main()
    {
        InterfaceImplementer iImp = new InterfaceImplementer();
        iImp.MethodToImplement();
    }

    public void MethodToImplement()
    {
        Console.WriteLine("MethodToImplement() called.");
    }
}
```

Erros e Manipulação de Exceção

Quando algo der errado enquanto um programa C# está Executando, uma exceção ocorre. Exceções interromper o fluxo atual do programa e se nada for feito, o programa simplesmente pára Executando. Exceções podem ser causadas por um bug em que o programa — por exemplo, se um número é dividido por zero — ou pode ser um resultado de alguns entrada inesperado, tais como um usuário selecionar um arquivo que não existe. Como um programador, você precisará permitir que o programa lidar com esses problemas sem falhando até parar.

C# fornece várias palavras-chave — try, catche finally — que permite que programas detectar exceções, lidar com eles e continuar a Executando. Eles são uma ferramenta muito útil para tornar seus aplicativos mais confiáveis.

Try e Catch

O try e catch as palavras-chave são usadas juntos. Coloque try o Bloquear de código que você está preocupado pode gerar uma exceção e catch para conter o código que será executado se a exceção ocorrer. Neste exemplo, um cálculo cria uma divisão por zero exceção, que, em seguida, é detectada. Sem o try e catch blocos, esse programa falhará.

Exemplo:

```
class ProgramTryCatch
{
    static void Main()
    {
        int x=0, y=0;

        try
        {
            x = 10 / y;
        }

        catch (System.DivideByZeroException)
        {
            System.Console.WriteLine("There was an attempt to divide by zero.");
        }
    }
}
```

Finally

Código contido em um Bloquear de finally sempre é executado, se ocorrer uma exceção ou não. Use o Bloquear de finally para fazer se recursos são retornados: Por exemplo, para certificar-se que um arquivo esteja fechado.

Exemplo:

```
try
{
    // Code to try here.
}
catch (SomeSpecificException ex)
{
    // Code to handle exception here.
}
finally
{
    // Code to execute after try (and possibly catch) here.
}
```


Windows Forms

Que o Framework .Net foi todo projetado tendo em vista a internet nos já sabemos , porém nem todas as aplicações serão desenvolvidas para a web. A utilização de Windows Forms (Win Form) é o mecanismo que nos permite criar as tradicionais aplicações para Windows.

Win Forms é o Novo Mecanismo para construção de Aplicativos Windows, baseados no Framework .Net.

Um Windows Form é bastante semelhante ao conceito de formulário utilizado pelas versões atuais do VB e Delphi. O Formulário é o Elemento Básico, sobre o qual adicionaremos controles e códigos para determinados eventos associados com o Formulário e seus Controles. O Windows Form é tudo isso, porém com a diferença que pode usar todos os mecanismos do .Net Framework, dentre os principais mecanismos disponíveis, destaca-se o mecanismo de herança, o qual é chamado, para o caso do Win Forms, de Herança Visual.

O Win Forms, como tudo no Framework .Net é um Objeto, o qual é obtido a partir de uma classe básica. Todos os formulários no Framework .Net são baseados uma das seguintes classes:

- System.Windows.Forms.
- São baseados em um formulário padrão criado pelo usuário, através de mecanismos de herança.

Em Resumo, o Windows Form é o Elemento básico de Interação com o Usuário; em outras palavras, o Win Forms é o Elemento visual das aplicações, elemento este com o qual o usuário irá trabalhar.

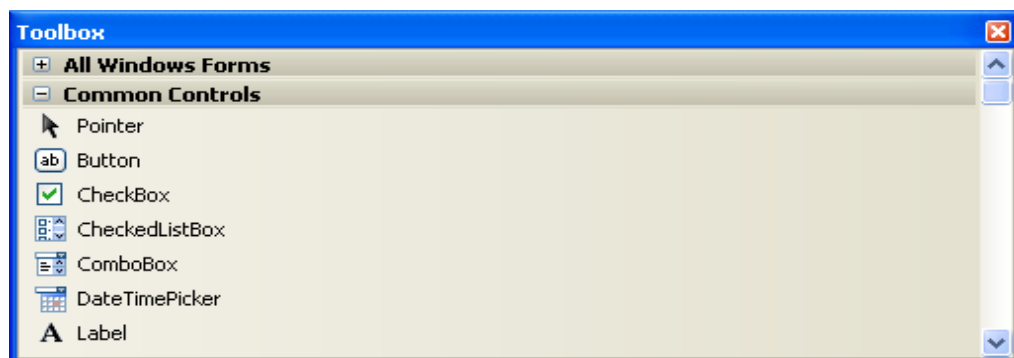
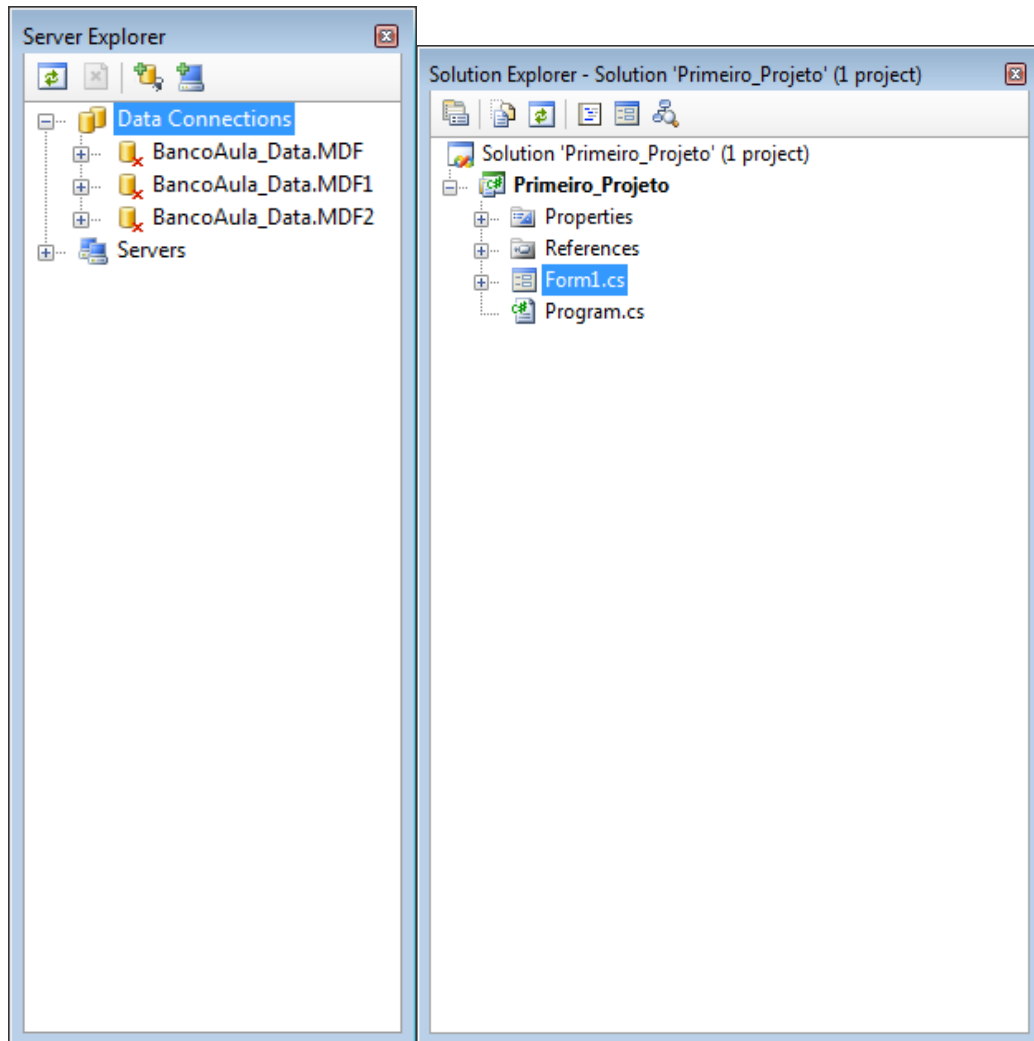
Tendo em vista o nosso breve aprendizado sobre o Framework, Orientação a Objetos e algumas funcionalidades do Visual Studio 2005 como IDE de Desenvolvimento, vamos praticar.

Conhecendo o Visual Studio 2008

Server Explorer → Aba responsável por toda a parte de conexão com o Banco.

Solution Explorer → Arvore do Projeto, local onde se vê todos os Forms, Classes, Refêrencias e Propriedades do Projeto.

ToolBox → Caixa de Componentes(Objetos Visíveis e não visíveis) (Win Form, Web Form).



Primeiro Projeto em C#

Parte 1

Proposta: Criar um projeto Windows Application contendo um modulo de calculadora e um modulo de cadastro de produtos. Criaremos um formulário principal em nossa aplicação com o controle Menu para acessar os seus módulos. Neste formulário teremos o primeiro contato com a linguagem C# e a IDE do Visual Studio. Aprenderemos também sobre variáveis e tipos existentes, além de iniciar a codificação na linguagem. Para isto, vamos iniciar o Visual Studio e seguir os passos descritos abaixo:

Passo 1: Acessando a opção de menu File | New | Project, obteremos acesso à caixa de diálogo representada na figura 01. Selecione Windows Application para iniciarmos um projeto baseado em WinForm. Preencha o campo nome com AppAula e aponte o projeto para sua pasta.

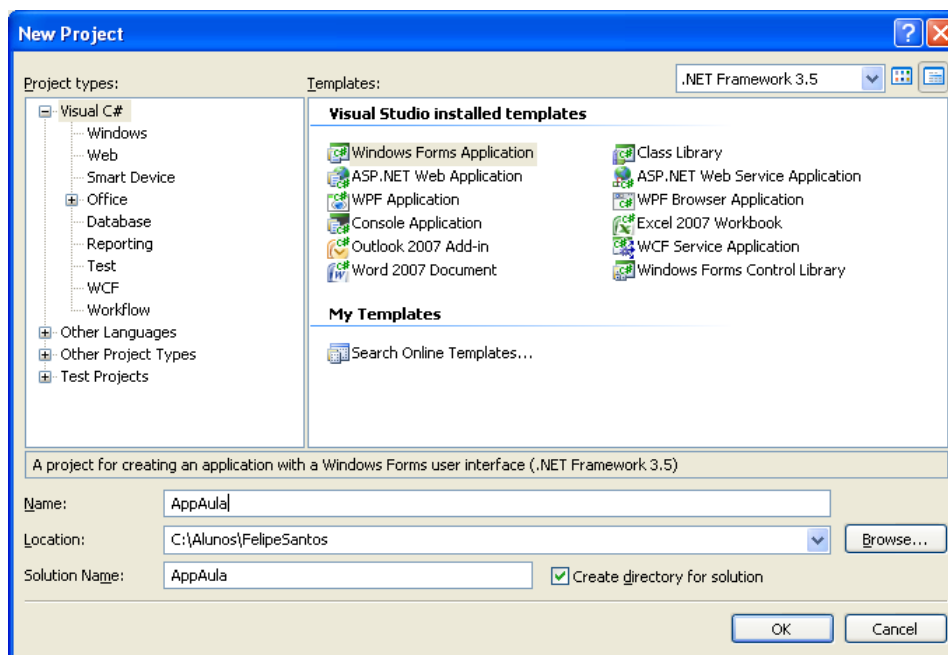


Figura 01 – Janela de diálogo para criação de um novo projeto

Como resultado desta etapa, o Visual Studio é apresentado já com um novo projeto e um formulário disponível. Este formulário será o formulário principal da nossa aplicação, então, usando o Solution Explorer, renomeie o formulário de Form1.cs para FormPrincipal.cs.

O Visual Studio irá perguntar se você deseja renomear todas as referências de Form1. Clique em 'Sim' para continuar.

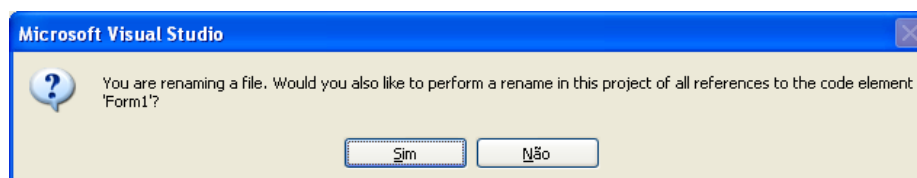


Figura 02 – Confirmação para alterar nome do formulário

Passo 2: Acrescente ao formulário FormPrincipal o controle MenuStrip localizado na Toolbox dentro da seção Menus & Toolbars. O menu irá automaticamente se posicionar no topo do formulário como padrão Windows. Onde aparece Type Here você pode escrever os itens do menu conforme a lista abaixo. O resultado será apresentado na figura 03.

Cadastro

- Clientes
- Produtos

Utilitários

- Calculadora

Sair

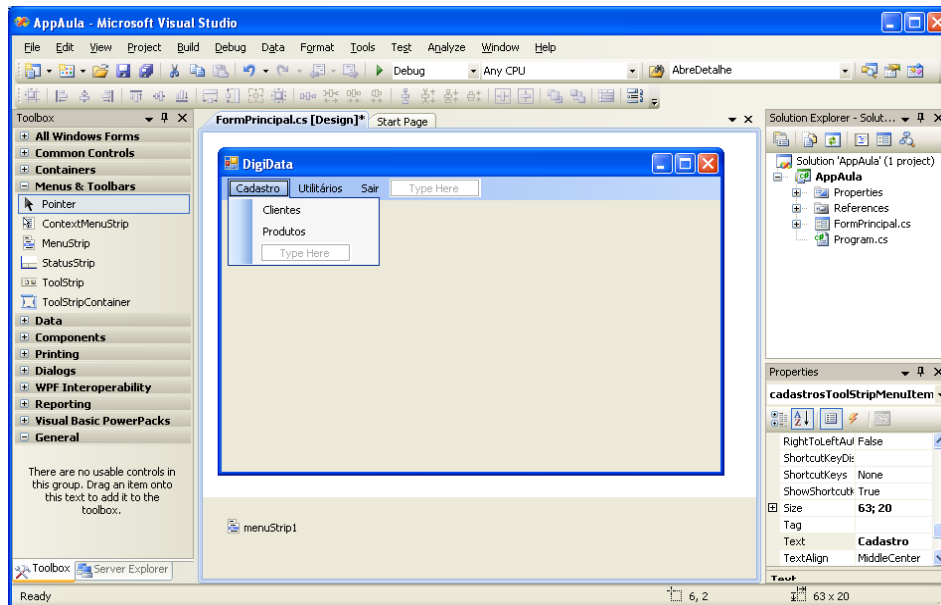


Figura 03 – Formulário principal com controle MenuStrip

Passo 3: Adicione ao formulário um controle StatusStrip, também localizado na seção Menus & Toolbars da Toolbox. Adicione dois StatusLabel ao StatusStrip e mude suas propriedades Name para statusData para o primeiro e statusHora para o segundo.

Passo 4: Adicione um controle Timer localizado na seção Componentes da Toolbox e mude suas propriedades conforme a tabela abaixo:

Propriedade	Valor
Name	tmRelogio
Interval	1000
Enabled	True

Passo 5: Dando um duplo clique no controle Timer, vc irá para o código no evento Tick do mesmo. Agora vamos iniciar a codificação conforme descrito abaixo:

```
using ...
namespace AppAula
{
    public partial class FormPrincipal : Form
    {
        public FormPrincipal()
        {
            InitializeComponent();

            private void tmRelogio_Tick(object sender, EventArgs e)
            {
                statusData.Text = DateTime.Now.ToShortDateString();
                statusHora.Text = DateTime.Now.ToLongTimeString();
            }
        }
    }
}
```

Podemos agora executar a aplicação para vermos o resultado.

Note o delay que ocorre para que o tmRelogio inicie. Para evitar este bug, podemos chamar o método tmRelogio_Tick após a inicialização do formulário. Então, dentro da declaração do formulário, após InitializeComponent();, faça a chamada do método.

```
public FormPrincipal()
{
    InitializeComponent();
    tmRelogio_Tick(null, null);
}
```

Parte 2 – Calculadora

Agora adicionaremos um formulário na aplicação para aprenderemos a manipular classes, métodos, atributos e propriedades.

Passo 1: Clique com o botão direito do mouse no nome da aplicação dentro do Solution Explorer e escolha a opção Add | New Item... e na janela que se abre, selecione Windows Form e preencha seu nome de FormCalculadora conforme a figura 04.

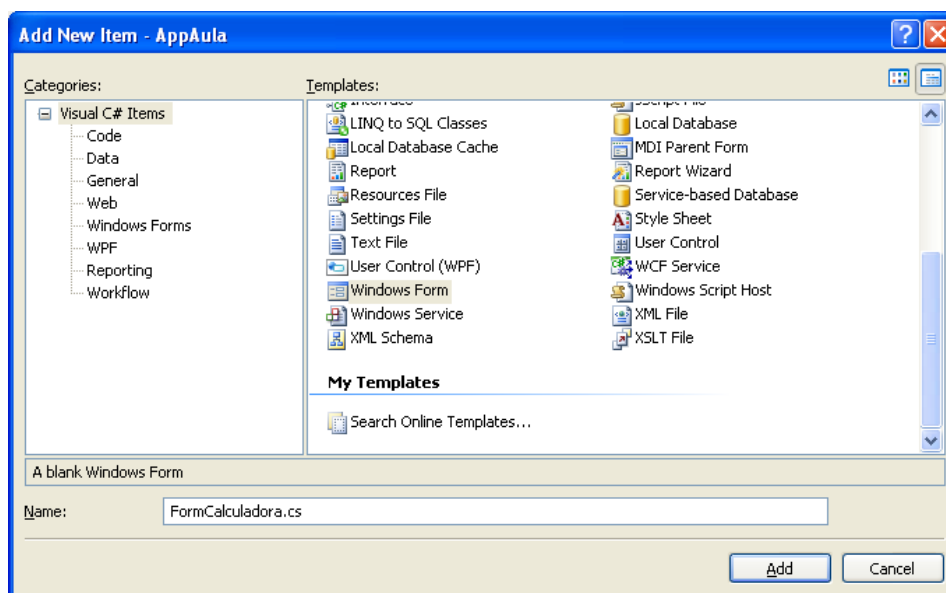


Figura 04 – Adicionando uma nova classe ao projeto

Passo 2: Monte a tela da calculadora com os controles abaixo alterando suas propriedades conforme descrito. Teremos um formulário semelhante ao da figura 05.

Controle	Propriedade	Valor
Label	Text	Valor 1:
TextBox	Name	textValor1
Label	Text	Valor 2:
TextBox	Name	textValor2
Label	Text	Resultado:
	Name	labelResultado
Label	Text	Operação
ComboBox	Name	CmbOperacao
Button	Name	BtCalcular
	Text	Calcular

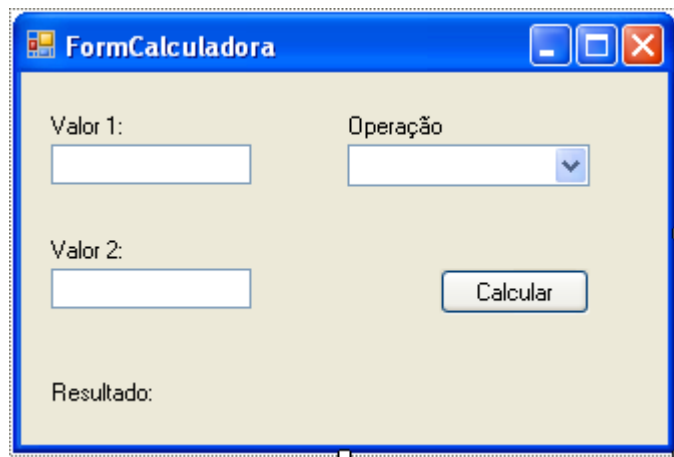


Figura 05 – Formulário Calculadora

Passo 3: Vamos adicionar uma classe ao projeto que será responsável por ter as propriedades e métodos da calculadora. Clique com o botão direito do mouse sobre o nome do projeto no Solutio Explorer e escolha Add | New Item.... Selecione Class na caixa de diálogo e dê o nome de Calculadora.cs e clique em Add. Feito isso, a classe é exibida em nossa janela para iniciarmos a codificação das suas propriedades, atributos e métodos conforme o código abaixo:

```
class Calculadora
{
    private Double _valor1;

    public Double Valor1
    {
        get { return _valor1; }
        set { _valor1 = value; }
    }

    private Double _valor2;

    public Double Valor2
    {
        get { return _valor2; }
        set { _valor2 = value; }
    }

    public enum Operacao { Soma, Subtracao, Multiplicacao, Divisao }

    public Calculadora()
    {
    }

    public Calculadora(Double V1, Double V2)
    {
        this._valor1 = V1;
        this._valor2 = V2;
    }

    public Double Somar()
    {
        return this._valor1 + this._valor2;
    }

    public Double Subtrair()
    {
        return this._valor1 - this._valor2;
    }
}
```

```

    public Double Multiplicar()
    {
        return this._valor1 * this._valor2;
    }

    public Double Dividir()
    {
        return this._valor1 / this._valor2;
    }

    public Double Calcular(Operacao oper)
    {
        switch (oper)
        {
            case Operacao.Soma:
                return Somar();
            case Operacao.Subtracao:
                return Subtrair();
            case Operacao.Multiplicacao:
                return Multiplicar();
            case Operacao.Divisao:
                return Dividir();
            default:
                return 0;
        }
    }
}

```

Passo 4: Com a classe pronta, vamos voltar ao formulário FormCalculadora pra implementarmos o código abaixo nos eventos que seguem:

```

public partial class FormCalculadora : Form
{
    public FormCalculadora()
    {
        InitializeComponent();
        cmbOperacao.DataSource = Enum.GetValues(typeof(Calculadora.Operacao));
    }

    private void btnCalcular_Click(object sender, EventArgs e)
    {
        Double V1 = Convert.ToDouble(textValor1.Text);
        Double V2 = Double.Parse(textValor2.Text);

        Calculadora calc = new Calculadora(V1, V2);

        Double resultado = calc.Calcular((Calculadora.Operacao)cmbOperacao.SelectedItem);

        labelResultado.Text = "Resultado: " + resultado.ToString("N2");
    }
}

```

Passo 5: Resta agora fazer a chamada do formulário FormCalculadora no item do menu Calculadora no formulário FormPrincipal. Para isto, dê um duplo clique no item do menu e proceda com o código abaixo e execute a aplicação para ver o resultado.

```

private void calculadoraToolStripMenuItem_Click(object sender, EventArgs e)
{
    FormCalculadora frmCalc = new FormCalculadora();
    FrmCalc.ShowDialog(this);
}

```

Segundo Projeto em C# Assembly(DLL) Contendo DataSet Tipado

Parte I

Proposta

Criar uma camada de acesso a dados, que forneça uma interface com métodos para manutenção completa de uma tabela. Para tanto criaremos um objeto DataSet para uma tabela, com dois métodos de pesquisa nesta além das funcionalidades básicas (Insert, Update e Delete). Também iremos implementar uma classe que forneça acesso aos métodos desta classe DataSet, ocultando detalhes relativo a classe que se relaciona com o BD.

Entre outras questões relevantes, considerando o paradigma do modelo de Programação Orientada a Objetos (OOP), a classe última citada, além de ocultar detalhes da implementação de acesso e manipulação de dados, permite uma mudança de MiddleWare (Providers) de acesso a dados.

Como última ressalva, citamos o fato de inicialmente estarmos utilizando estas duas classes (DataSet e Acesso) em um projeto Winform, permitindo sua reutilização em projetos outros como: Aplicativos Web e Webservices. Esta camada será desenvolvida em formato de dll.

A Estrutura do Projeto

Na prática serão duas classes. Uma será o DataSet e a outra uma classe de acesso aos métodos do DataSet. Depois faremos outro aplicativo baseado em Winform, fazendo acesso as funcionalidades do primeiro aplicativo. Na sequencia relacionamos a lista de módulos de cada aplicativo:

Projeto Produto Projeto WinForm

- | | |
|---------------|------------------|
| -ClassProduto | -FormPrincipal |
| -DSProduto | -FormConsProduto |
| | -FormCadProduto |

Passo 01: Acessando a opção de menu File|New Project, obteremos acesso a caixa de dialogo representada pela figura 01. Selecione Class Library para iniciarmos um projeto. Preencha a propriedade Name como Produto. Finalizando clique no botão OK.

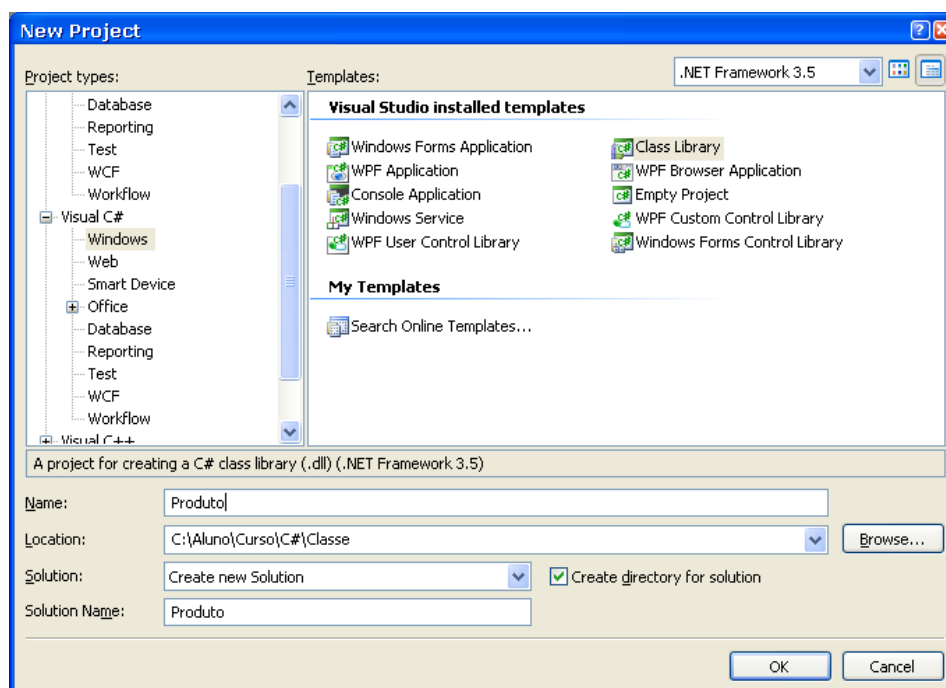


Figura 01

Em seguida nomeie a Class1.cs como ClassProduto.cs, salvando em seguida com este nome.

Para o nome do projeto, confirme os dados conforme

Passo 02: Vamos agora implementar o esqueleto da classe ClassProduto, definindo os Campos, Propriedades e Métodos para em seguida realmente implementarmos as funcionalidades. Veja a figura 01 que expõem toda a classe e também os comentários sobre as instruções ali contidas.

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

namespace Produto
{
    public class ClassProduto
    {
        private int _Id;
        private string _Descricao;
        private decimal _Preco;
        private int _Original_Id;
        private string _Original_Descricao;
        private decimal _Original_Preco;

        public int Original_Id
        {
            get { return _Original_Id; }
            set { _Original_Id = value; }
        }

        public decimal Original_Preco
        {
            get { return _Original_Preco; }
            set { _Original_Preco = value; }
        }

        public string Original_Descricao
        {
            get { return _Original_Descricao; }
            set { _Original_Descricao = value; }
        }
    }
}
```

```
public decimal Preco
{
    get { return _Preco; }
    set { _Preco = value; }
}
```

```
public string Descricao
{
    get { return _Descricao; }
    set { _Descricao = value; }
}
```

```
public int Id
{
    get { return _Id; }
    set { _Id = value; }
}
```

```
/// Altera um produto existente
public void Alterar()
{
}
```

```
/// Cria um novo produto
public void Incluir()
{
}
```

```
/// Exclui um produto
public void Excluir()
{
}
```

```
public DataTable ObterProduto(int Id_Produto)
{
}
```

```
/// Retorna Vários produtos, com base nas iniciais do campo descrição
public DataTable ObterProduto(string Descricao)
{
}
```

```
}
}
```

Com os Fields (campos) da nossa Classe Criados precisamos encapsular os campos criando assim as Properties, como segue na figura abaixo:

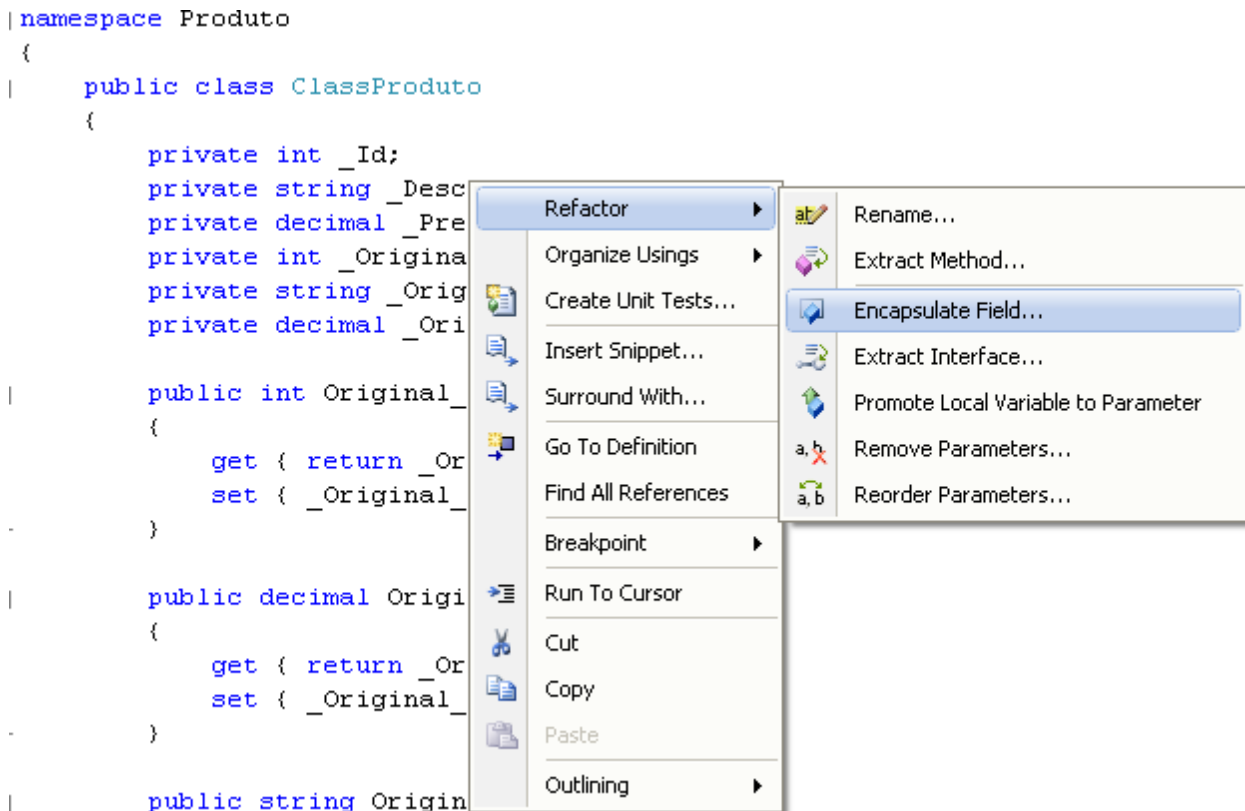


Figura 02

Passo 03: Um recurso bastante interessante do Visual Studio 2008, estendendo-se naturalmente ao C#.Net, é a possibilidade de exibirmos uma Classe em forma de diagrama (figura 02). De forma inversa, podemos inclusive criar elementos, Fields(campos), Properties(propriedades) e Methods(métodos) diretamente no diagrama.

Como exemplo dessas possibilidades, confira as figuras a seguir, observando como é fácil e produtivo tal recurso.

Com objetivo de experimentar tal recurso, clique sobre o diagrama da classe com o botão direito do mouse, fazendo opção Add. Com este menu de contexto ativo, escolha Field.

O resultado é a declaração automática no corpo da classe no local próprio, bastando alternar para a página de código da classe, observando a declaração abaixo.

```
private double _Teste;
```

Procedendo da mesma maneira para criar uma propriedade, faça a opção Method, observando em seguida a declaração de tal propriedade e seus métodos Get e Set.

Para criar um método, basta também proceder como nas duas instruções anteriores e observar o resultado.

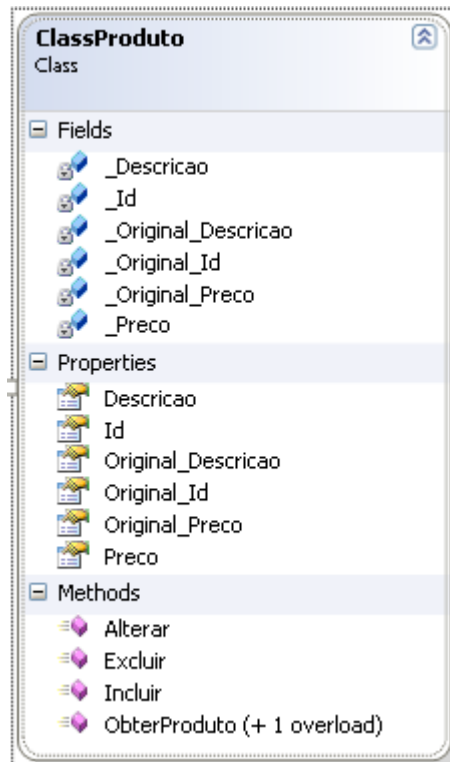


Figura 03

Passo 04: Uma vez definido toda a estrutura base da classe, necessitamos agora criar um DataSet Tipado que realmente implemente o acesso aos dados, bem como, possibilite por intermédio da classe ClassProduto a inclusão, exclusão e atualização de registros no banco de dados.

Um passo importante se faz necessário antes de criarmos tal DataSet. Refirimo-nos a própria conexão com o banco de dados.

Acesse o menu View, habilitando a exibição do utilitário Server Explorer. Este recurso permite não só criarmos várias conexões para Providers diferentes, como permite criarmos objetos de banco de dados entre outros. Observe a figura 04, que expõe este recurso.

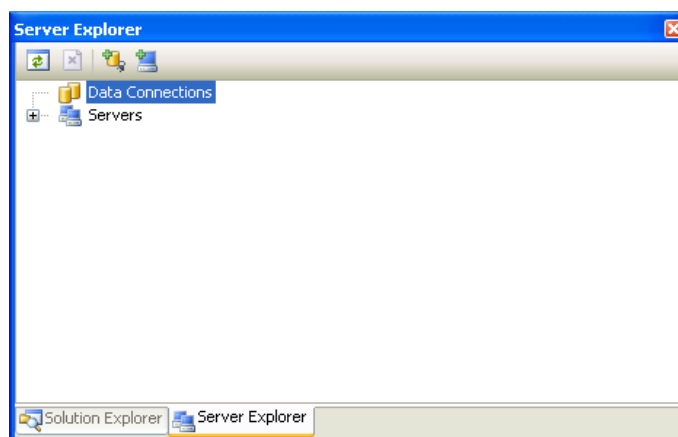


Figura 04

Para criar uma nova conexão, clique com o botão direito do mouse sobre Data Connection na janela do Server Explorer e será exibido a caixa de diálogo Add Connection (figura 05).

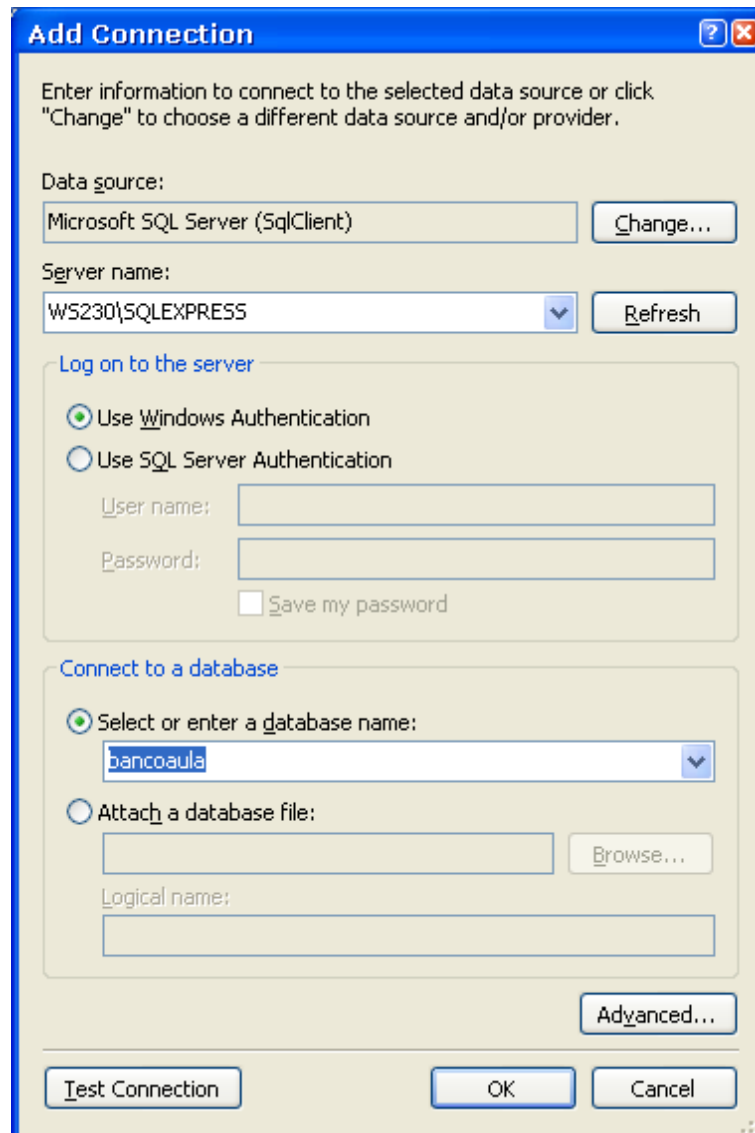


Figura 05

Figura 04 - Clicando no botão Change, nesta caixa de dialogo, será exibido a caixa de dialogo Change Data Source, onde podemos trocar o estabelecer o Provider de acesso a dados específico par o banco de dados desejado. Neste caso optamos pelo Provider do banco de dados MS-SQL Server, mas em outras etapas faremos uso de outros Providers para comprovar a possibilidade de troca de provedor de dados, sem grande impacto sobre a aplicação num todo. Confira na imagem da figura 06.

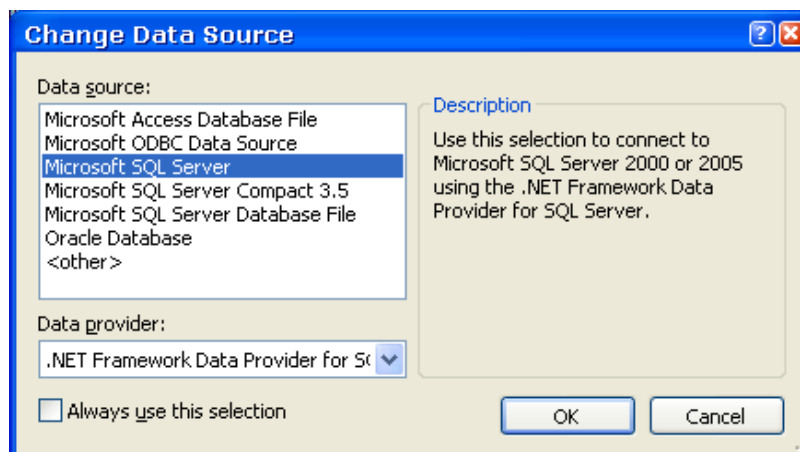


Figura 06

Uma vez criado o nó que representa a conexão, podemos partir para a próxima etapa.

Passo 05: Iniciando o processo, clique no menu Data, escolhendo a opção New Data Source. Observe a figura 7, onde escolhemos a origem dos dados, neste caso Database

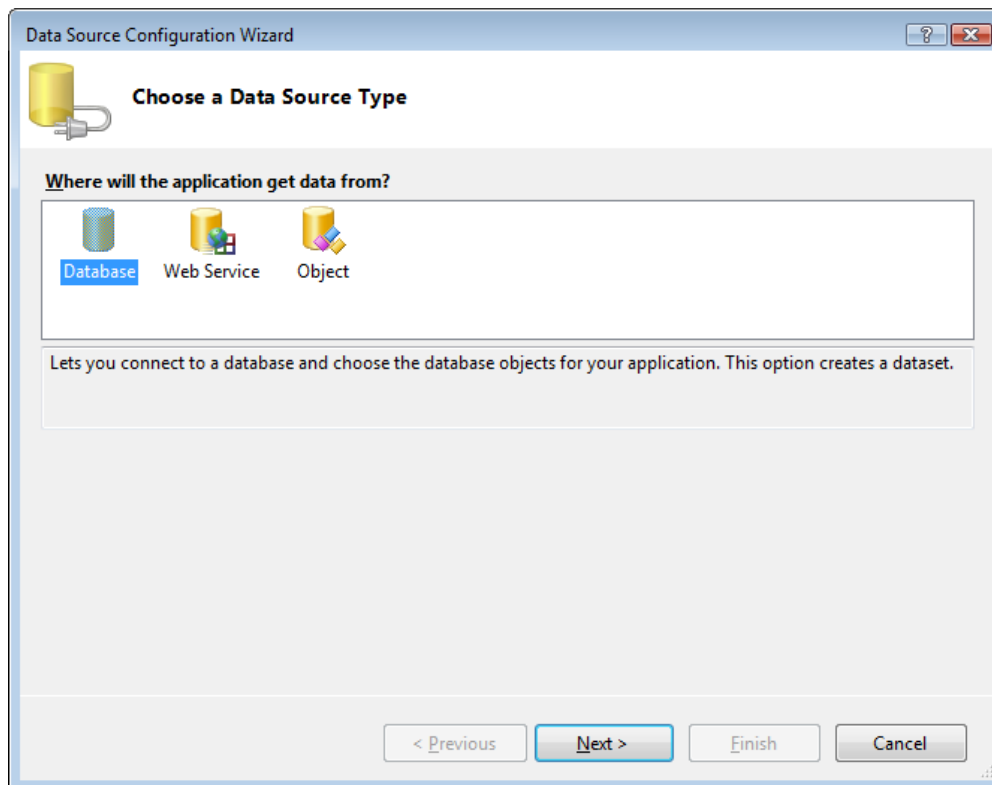


Figura 7

Em seguida, conforme demonstra a figura 8, iremos selecionar a conexão recém-criada.

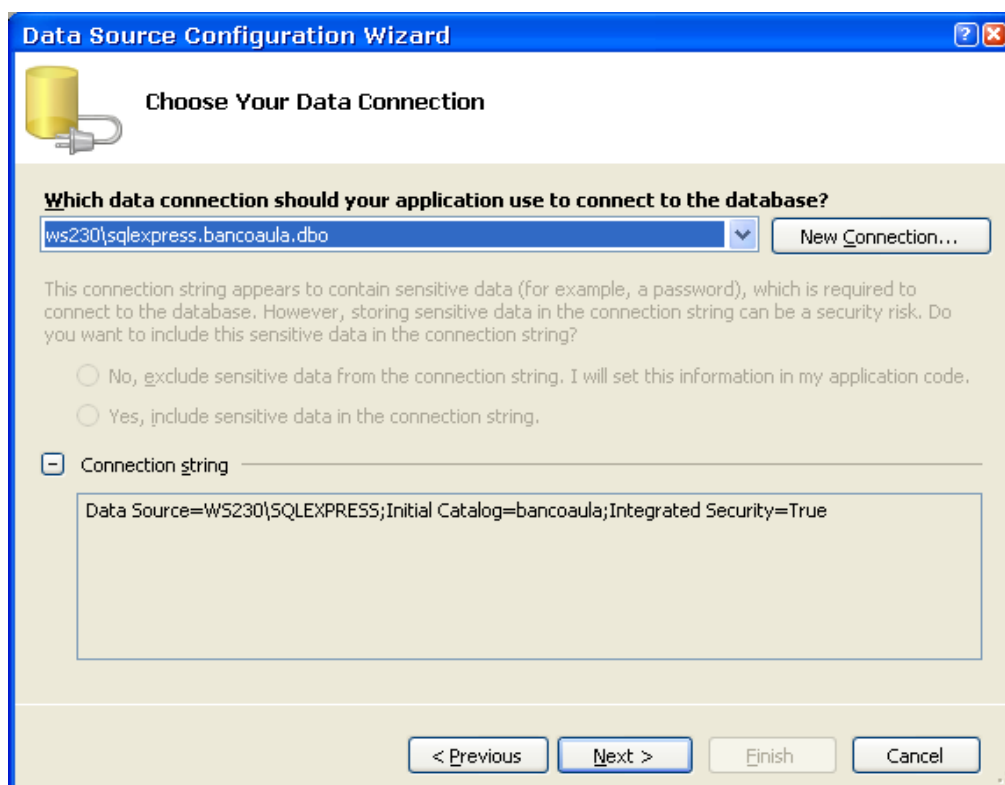


Figura 8

Clicando em Next, na caixa de dialogo Data Source Configuration Wizard (figura 8) basta escolher na estrutura de árvore a tabela Produtos e dando um nome para o DataSet. Em seguida clique em Finish e teremos um objeto visual colocado no módulo DSProduto representando um DataTable, objeto esse que onde implementaremos os métodos necessários a nossa classe. Observe o resultado exposto na figura 9.

Figura 9

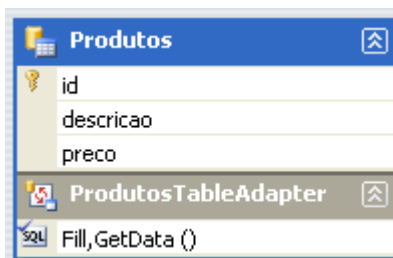


Figura 10

Observe a figura 9 onde um DataTable denominado Produto, implementa um TableAdapter nomeado como ProdutoTableAdapter. Nele encontramos criado automaticamente um método nomeado como GetData com a seguinte instrução SQL: **SELECT id, descricao, preco FROM Produtos**

Esta instrução deverá ser alterada para que possamos obter somente um registro, baseado na chave-primária da tabela. Lembre-se que nosso método ObterProduto, sobrecarregado, em uma de suas implementações, requer o identificador de um produto para retornar um objeto DataTable com tal registro. Assim o código correto é:

SELECT id, descricao, preco FROM Produtos WHERE (id = @id)

Repare a inclusão do parâmetro @id, que será substituído com o valor passado por parâmetro no método.

Existe duas maneiras de se alterar a instrução SQL deste método. A primeira, representada pela figura 11, é acessar a caixa de propriedade deste método, clicando sobre o mesmo com o botão direito do mouse e manipulando diretamente na propriedade CommandText.

Aproveitando o ensejo, altere também o nome do método para ObterProdutoID em GetMethodName, evidenciando sua habilidade para retornar um registro baseado na pesquisa sobre o campo Id.

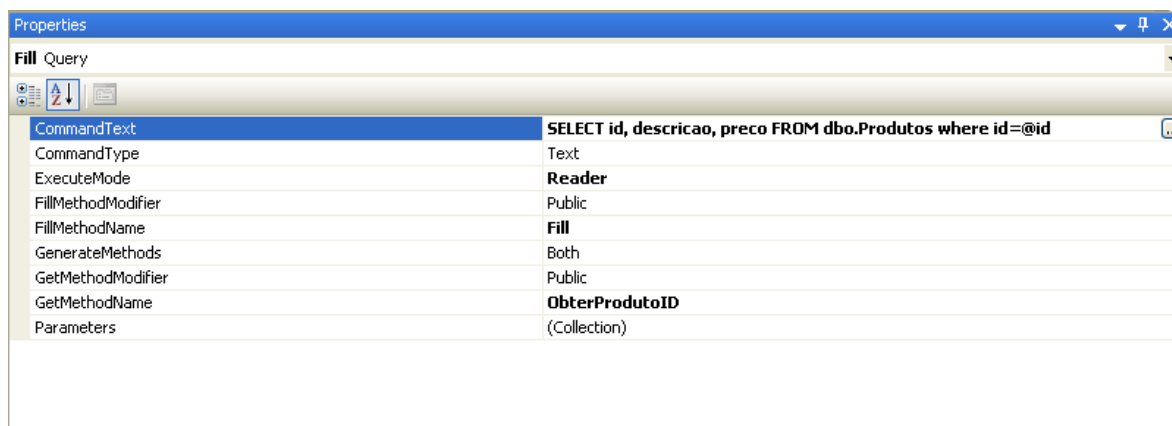


Figura 11

A outra maneira sugerida, conforme podemos observar na figura 12, clicando nas reticências da propriedade CommandText, ter acesso ao utilitário Query Builder, onde temos recursos avançados para montagem de consultas

parametrizadas. Faça da maneira que melhor lhe convir.

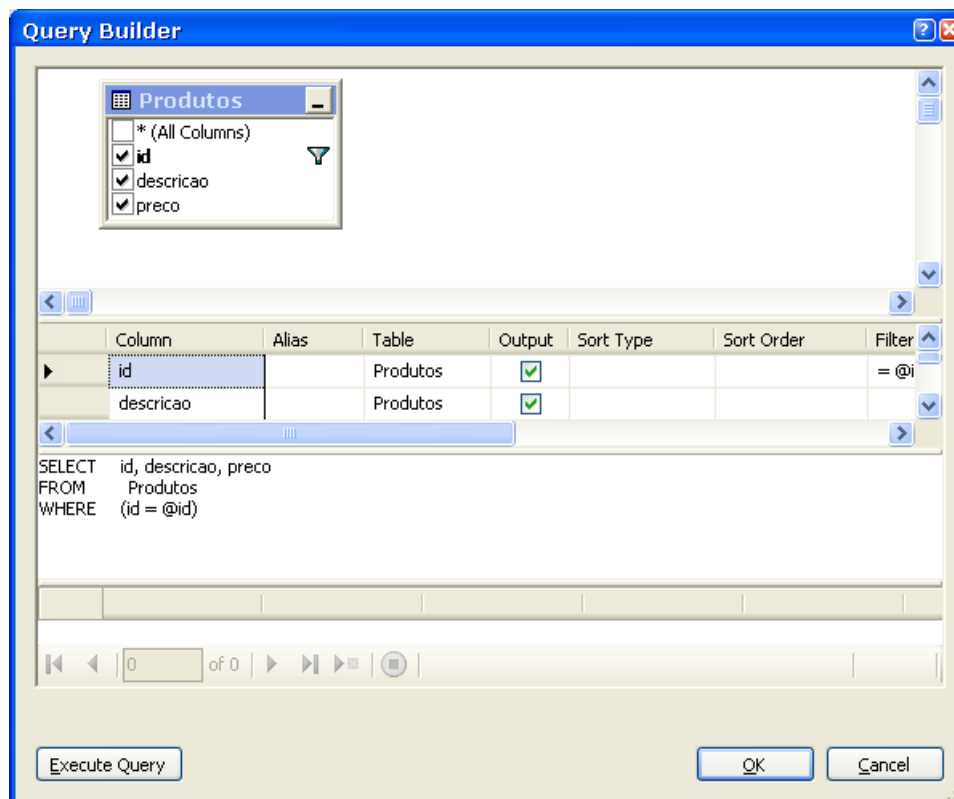


Figura 12

Passo 07: Devemos agora adicionar mais um método de pesquisa ao DataSet, denominado ObterProdutodescricao, permitindo a pesquisa na tabela Produtos pelas iniciais do Produto (campo descricao). Assim, clique sobre ProdutosTableAdapter com o botão direito do mouse, escolhendo Add Query no menu de contexto.

A caixa de dialogo (figura 13) TableAdapter Query Configuration Wizard é apresnetada, para a qual faremos a escolha da primeira opção (Use SQL Statements), clicando em seguida no botão Next.

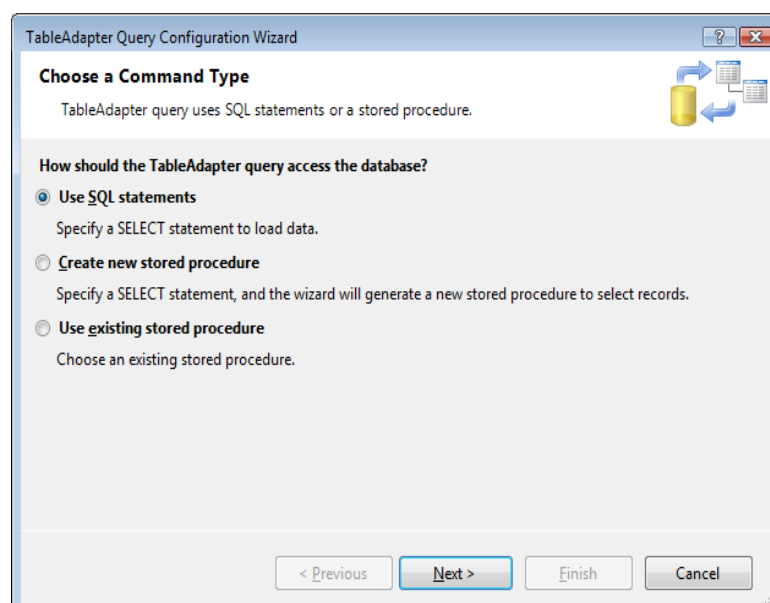


Figura 13

Caracterizando a Query a ser criada, no nosso caso uma instrução Select, faça a opção SELECT which returns rows indicando que desejamos criar uma instrução que retorne registros. Clique em Next para prosseguir.

Veja a figura 14 para essas orientações.

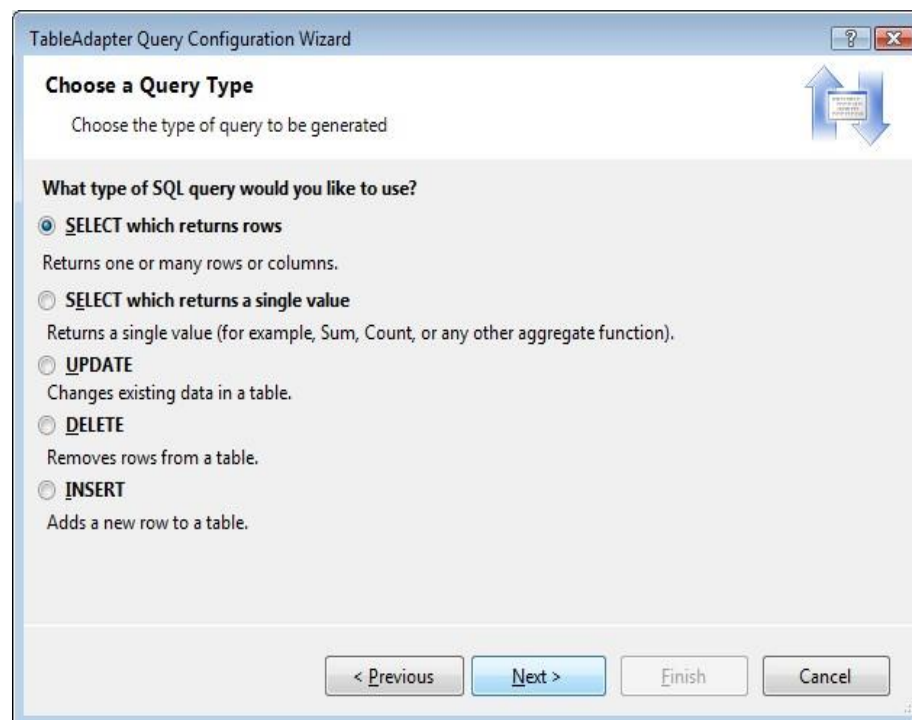


Figura 14

Nesta etapa basta digitar a instrução Select desejável, ou ainda, utilizar o Query Builder para ajuda neste sentido clicando em Next para proseguir. A figura 15 reflete esta etapa.

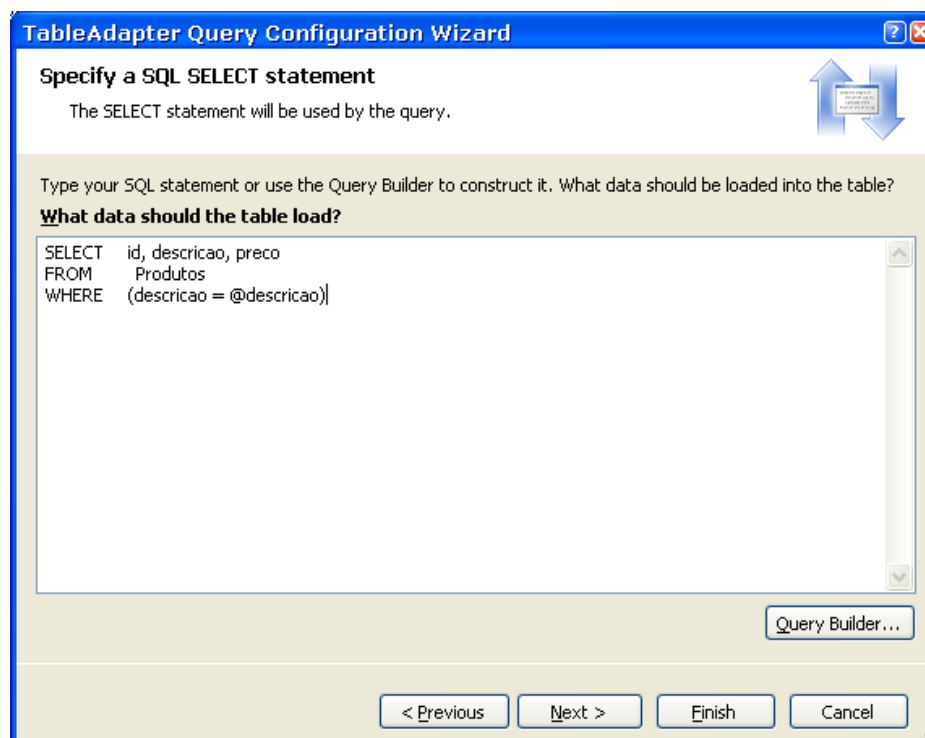


Figura 15

Imediatamente antes de finalizar, devemos adaptar o nome do método para nossa realidade. Repare na figura 16 os nomes alterados para FillByDescricao e ObterProdutoDescricao respectivamente.

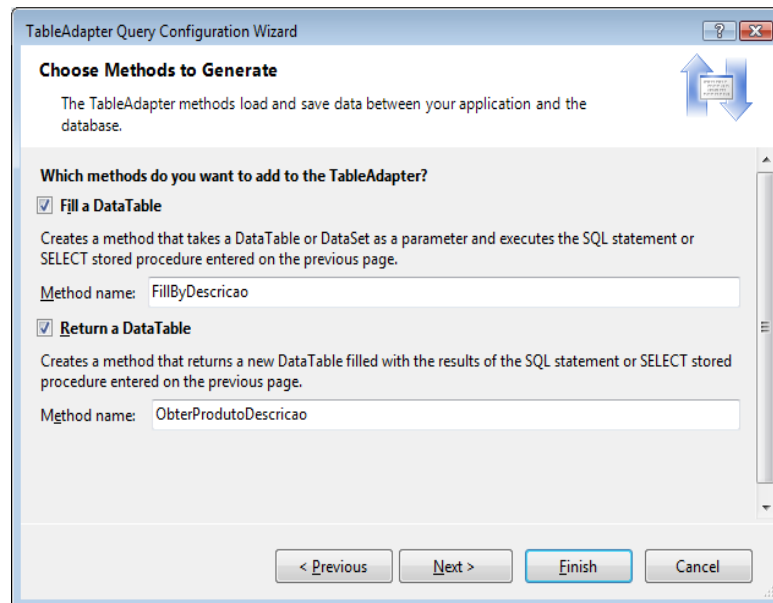


Figura 16

Clicando em Finish obtemos o resultado final conforme figuras 17 e 18. Pronto o novo método do DataSet está concluído.

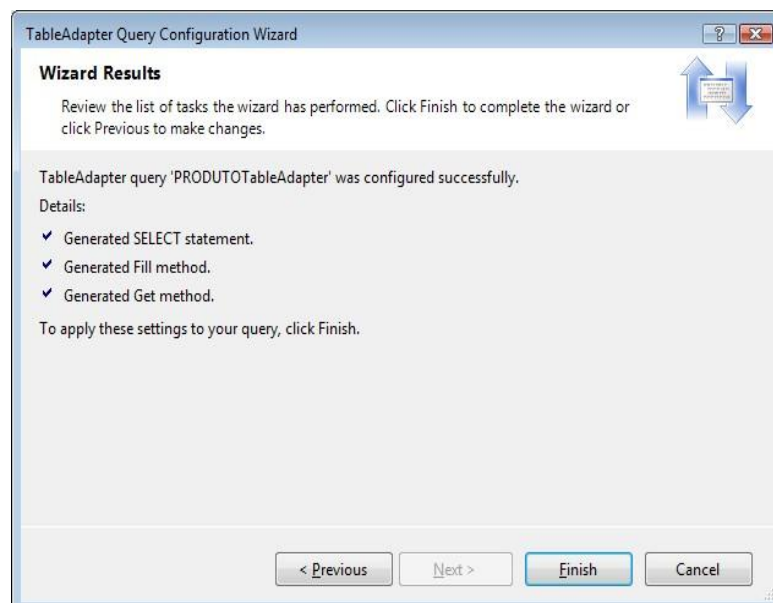


Figura 17

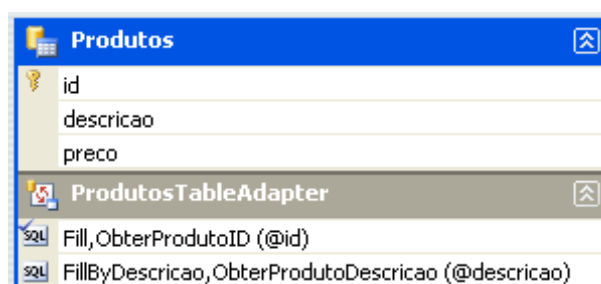


Figura 18

É importantíssimo ressaltar, que ainda que não tenhamos feito nenhum esforço, os métodos necessários para inclusão, alteração e exclusão de registros na tabela Produto, foram criados, podendo ser acessado por uma instância do objeto ProdutoTableAdapter, presente na classe DSProduto. Tais métodos são denominados como: Insert, Update e Delete, podendo ser acessados conforme exemplo abaixo.

```
DSProdutoTableAdapters.PRODUTOTableAdapter Produtos =  
new DSProdutoTableAdapters.PRODUTOTableAdapter();
```

```
Produtos.Update(_Id, _Descricao, _Preco, _Original_Id, _Original_Descricao, _Original_Precos);
```

Na verdade, podemos e devemos interferir nas instruções criadas para as propriedades InsertCommand, DeleteCommand e UpdateCommand, ajustando as instruções conforme objetivo. Para acessar tais propriedades basta clicar com o botão direito do mouse sobre ProdutoTableAdapter, escolhendo Properties no menu de contexto. Observe tal caixa de dialogo conforme figura 19.

Para a instrução Insert, nada além de quatro parâmetros, um para cada campo da tabela produto. Tais parâmetros sempre devem ter o nome iniciado com "@", sendo opcional mas muito importante, que tenham o nome da coluna para qual serão atribuídos. Ver código abaixo.

```
INSERT INTO Produtos(id, descricao, preco) VALUES (@id,@descricao,@preco)
```

Com relação a instrução Delete, mais simples e objetiva, basta um único parâmetro para identificar o registro pela chave-primária da tabela. No caso o campo Id. Confira o código a seguir.

```
DELETE FROM Produtos WHERE id = @Original_id
```

A exceção a simplicidade não se aplica a instrução Update. Não basta definirmos os parâmetros na quantidade de atribuições, sendo necessário parâmetros extras para implementação da cláusula Where. Este fato deve-se a necessidade considerarmos a concorrência de dados. Quando simultaneamente dois ou mais processos obterem o mesmo registro para alteração, devemos prever que no momento da atualização de um desses processos, um entre os três já teria finalizado com sucesso, obrigando ser sinalizado quando da tentativa de update por dos dois processos restantes. Isto não é uma regra, mas se não assim procedermos, teríamos uma situação que um registro obtido, com campos tendo valores originais no momento da efetiva alteração já teriam sido mudados. Caso isso não seja uma restrição, basta manter como parâmetro, a chave-primária, neste caso, o campo Id.

```
UPDATE Produtos SET descricao = @descricao, preco = @preco
```

```
WHERE
```

```
(id = @Original_id) AND
```

```
(descricao = @Original_descricao) AND
```

```
(preco = @Original_preco)
```

Variação do comando Delete levando-se em conta as mesmas condições do Update, ou seja, concorrência de dados.

```
DELETE FROM Produtos
```

```
WHERE (id = @Original_id) AND
```

```
(descricao = @Original_descricao) AND
```

```
(preco = @Original_preco)
```

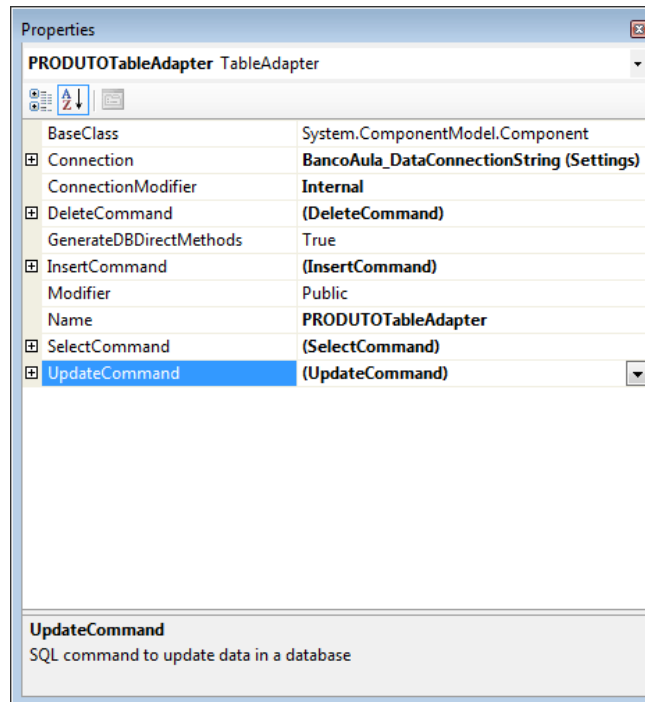


Figura 19

Passo 08: Já que toda implementação do DataSet DSProduto está finalizada, com métodos necessários a atender nossa classe ClassProduto, vamos agora fazer uso dos métodos deste DataSet em nossa classe, visando assim dar funcionalidades aos métodos da classe classProduto para que aplicativos possam fazer uso. Observe o código em questão na listagem 02.

```

/// Altera um produto existente
/// </summary>
public void Alterar()
{
    DSProdutoTableAdapters.ProdutosTableAdapter
    Produtos = new DSProdutoTableAdapters.ProdutosTableAdapter();
    Produtos.Update(_Descricao, _Preco, _Original_Id, _Original_Descricao, _Original_Preco);
}

/// Cria um novo produto
public void Incluir()
{
    DSProdutoTableAdapters.ProdutosTableAdapter
    Produtos = new DSProdutoTableAdapters.ProdutosTableAdapter();
    Produtos.Insert(_Id, _Descricao, _Preco);
}

```

```
/// Exclui um produto
public void Excluir()
{
    DSProdutoTableAdapters.ProdutosTableAdapter
    Produtos = new DSProdutoTableAdapters.ProdutosTableAdapter();
    Produtos.Delete(_Id, _Descricao, _Preco);
}

public DataTable ObterProduto(int id_produto)
{
    DSProdutoTableAdapters.ProdutosTableAdapter
    Produtos = new DSProdutoTableAdapters.ProdutosTableAdapter();
    return Produtos.ObterProdutoID(id_produto);
}

/// Retorna Vários produtos, com base na descrição recebida
public DataTable ObterProduto(string Descricao)
{
    DSProdutoTableAdapters.ProdutosTableAdapter
    Produtos = new DSProdutoTableAdapters.ProdutosTableAdapter();
    return Produtos.ObterProdutoDescricao(Descricao);
}
```

Listagem 02

Acima vemos um nítido modelo de Polimorfismo, temos um método denominado ObterProduto com dois parâmetros. Sendo assim terminamos os conceitos básicos de OO.

Parte II

O Projeto Cliente

Desenvolveremos nesta etapa um aplicativo baseado Windows Forms, que deverá gerar as interfaces necessárias para consumir os métodos da nossa DLL anteriormente desenvolvida.

Já que investimos na separação da lógica de negócio e acesso a dados, livrando nosso aplicativo de camada de apresentação de ter que lidar com código que não diga respeito a funcionalidade e aparência, vamos então investir nosso tempo em um aplicativo que preza por utilizar as melhores práticas e padrões do desenvolvimento.

Neste artigo complementar, faremos uso de um método exposto pela DLL, implementando uma consulta parametrizada permitindo a interação do usuário que poderá fazer submissões consecutivas por intermédio de um formulário.

Consulta de Produtos.

Construa uma Nova Aplicação no Menu File → New Project.

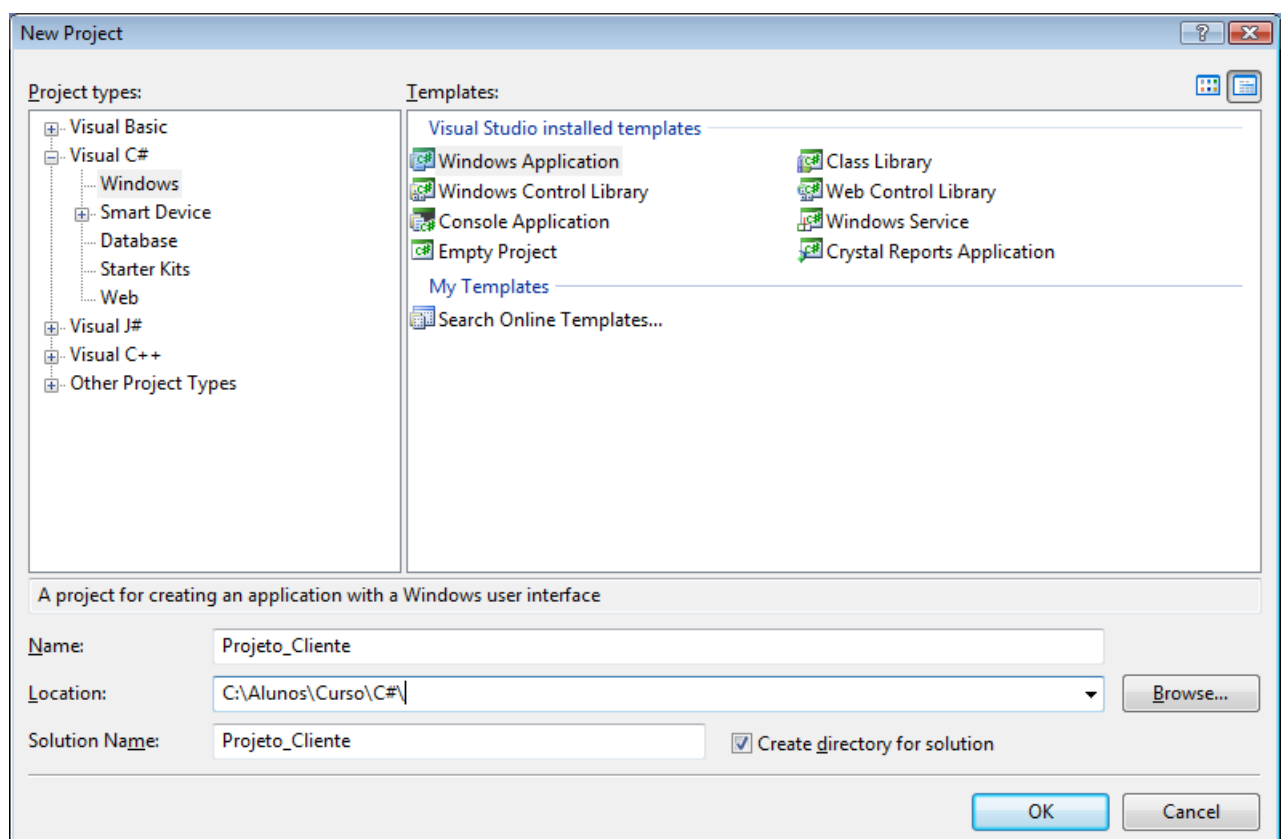


Figura 01

O Nome do Form criado automaticamente deverá ser FormPrincipal.cs. Adicione ao FormPrincipal os seguintes componentes.

Controle	Origem (Toolbox)	Propriedade	Valor
MenuStrip	Menus e ToolBars		

Devendo Criar os seguintes itens de menus → Cadastros → Produtos → Sair. Devendo ficar assim:

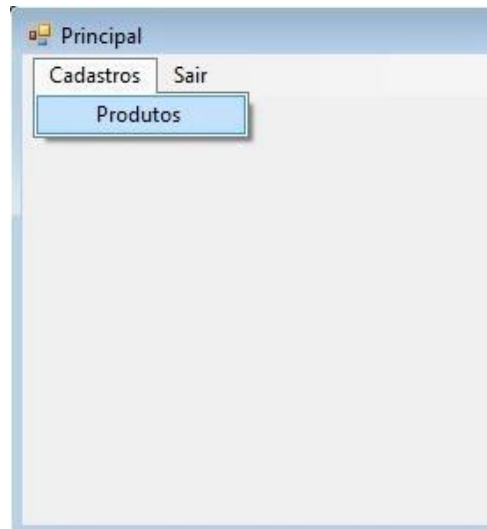


Figura 02

Com o Botão direito do Mouse sobre o Solution Explorer, selecione a opção ADD → Windows Forms.

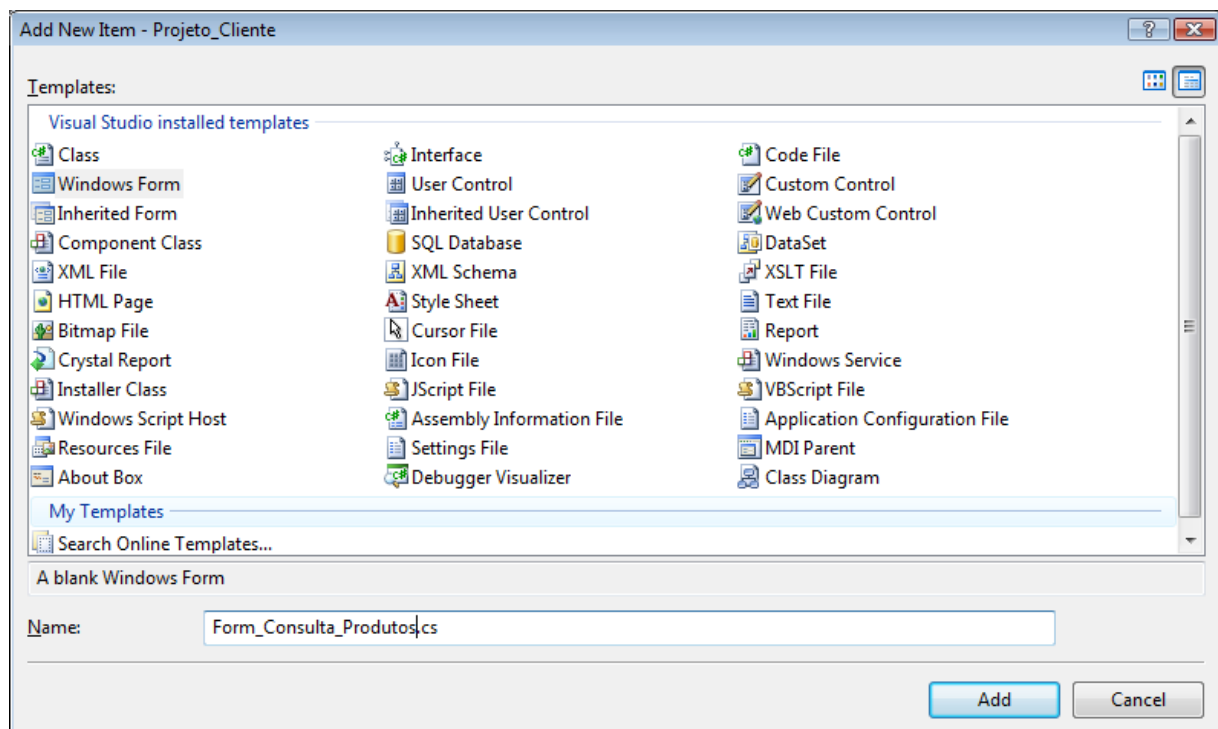


Figura 03

Salvando-o como Form_Consulta_Produto, e colocando os Seguinte Componentes sobre o mesmo.

Controle	Origem (Toolbox)	Propriedade	Valor
MenuStrip	Menus e ToolBars		
Button	Commom Controls	Name	ButtonPesquisar
		Text	Pesquisar
Button	Commom Controls	Name	ButtonIncluir
		Text	Incluir
Button	Commom Controls	Name	ButtonAlterar
		Text	Alterar
Button	Commom Controls	Name	ButtonExcluir
		Text	Excluir
Button	Commom Controls	Name	Button_Sair
		Text	Sair
Label	Commom Controls	Text	Nome Parcial do Produto
TextBox	Commom Controls	Name	TextBox_Descricao
DataGridView	Data	Name	GridConsulta
GroupBox	Containers	Dock	Top
		Name	GroupBox_1
		Text	
GroupBox	Containers	Dock	Fill
		Name	GroupBox_2
		Text	

O Form deverá ficar da seguinte forma.

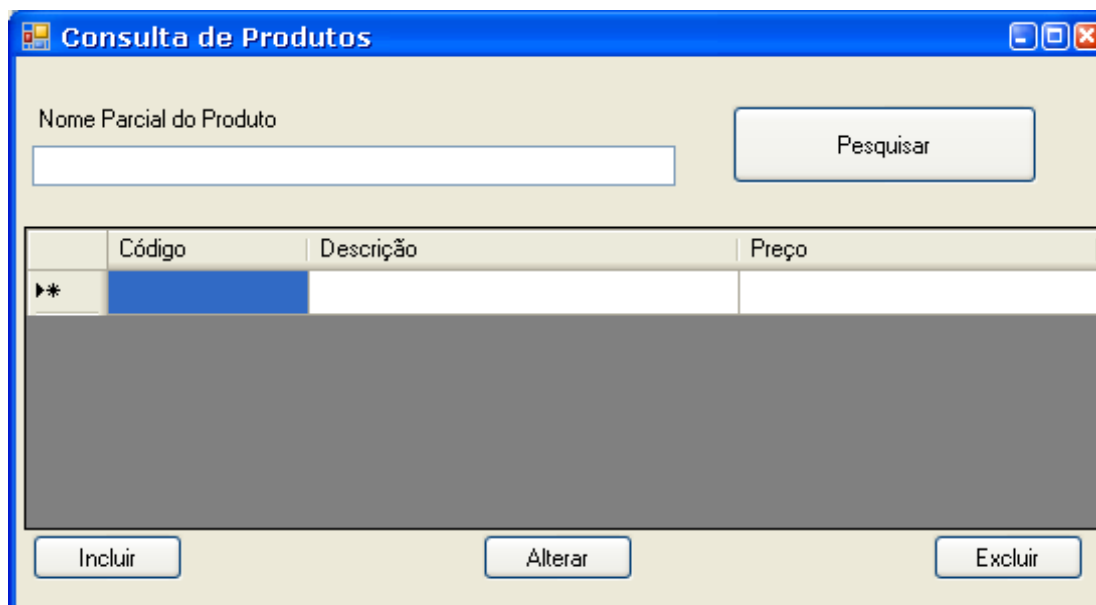


Figura 04

Configurando o Objeto Grid View.

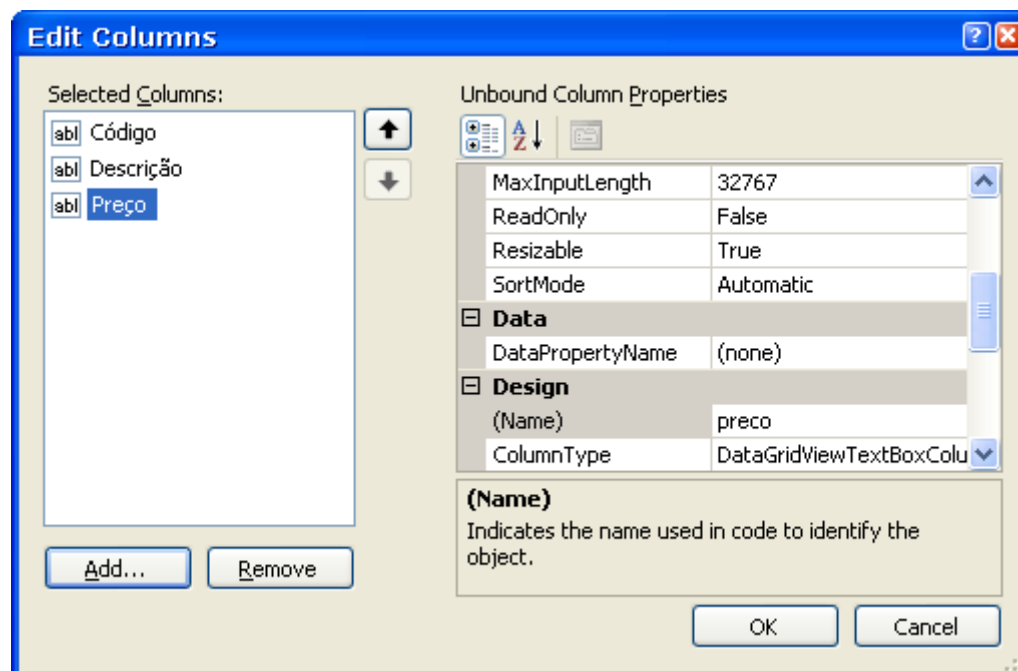


Figura 05

Passo 3: Devemos linkar a aplicação com a Dll, para tanto, devemos clicar com o botão direito sobre a estrutura de árvore apresentada na Solution Explorer e clicar em "Add Reference", na tela que abrirá (figura 06), clique em Browse e indique o caminho da dll.

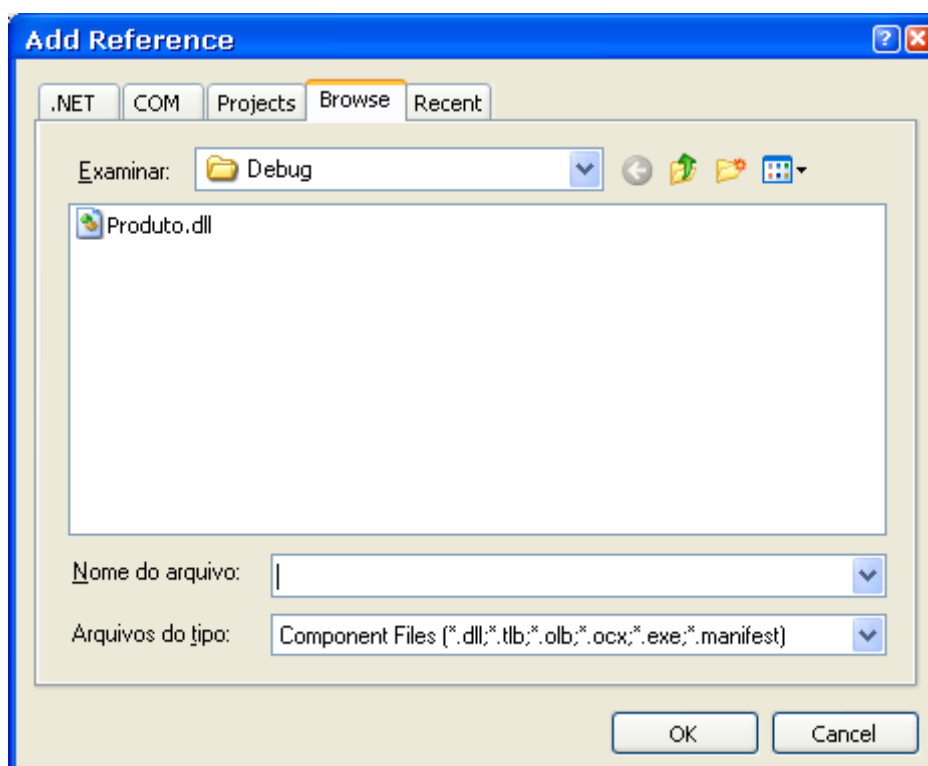


Figura 06

Nossa DLL está na Pasta Bin → Debug → Produto.dll

Passo 4: Agora que já temos o desing pronto e as aplicações ligadas, partiremos para a Implementação do formulário de consulta de produtos, segue abaixo o código e comentários. Ativando o Evento Click do Botão Pesquisar(Dar dois Cliques no Componente) e adicionar as seguintes referências. Consulta de Produtos

```
private void ButtonPesquisar_Click(object sender, EventArgs e)
{
    Produto.ClassProduto Class = new Produto.ClassProduto();
    GridConsulta.DataSource = Class.ObterProduto(TextBox_Descricao.Text);
}
```



	Código	Descrição	Preço
▶	6	Computador Pent...	2000,00
*			

Figura 07

Ficando assim nossa Consulta.Sendo necessário ao Evento dos Botões Incluir e Alterar, vamos criar o Formulário de Cadastro de Produtos. Da Mesma forma em que Implementamos os Outros Forms, iremos criar esse. O Nosso Formulario deverá ficar assim:




Figura 08

Agora sim poderemos prosseguir com os Códigos do Form de Consulta.

```
private void ButtonIncluir_Click(object sender, EventArgs e)
{
    FormCadastro_Produtos frmCadProduto = new FormCadastro_Produtos();
    frmCadProduto.set_Opcao("I");
    frmCadProduto.ShowDialog(this);
}

private void ButtonAlterar_Click(object sender, EventArgs e)
{
    DataGridViewRow linha = new DataGridViewRow();
    linha = GridConsulta.CurrentRow;
    FormCadastro_Produtos frmCadProduto = new FormCadastro_Produtos();
    frmCadProduto.set_Opcao("A");
    frmCadProduto.Preencher(linha.Cells["Id"].Value.ToString(),
    linha.Cells["Descricao"].Value.ToString(),linha.Cells["Preco"].Value.ToString());
    frmCadProduto.ShowDialog(this);
}

private void ButtonExcluir_Click(object sender, EventArgs e)
{
    /*O Código Referente à Exclusão de Produtos, fica para exercicio, podemos nos
    basear nos eventos acima para atingir nosso objetivo. */
}
```

Na sequência exercitaremos o Cadastro de Produtos.

Cadastro de Produtos

```
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 Projeto_Cliente
{
    public partial class FormCadastro_Produtos : Form
    {
        private string stOpcao;
        private int _Old_id;
        private string _Old_Descricao;
        private decimal _Old_Preco;
        // C#, utilizando conceitos de OO, ou seja, atributos sempre privados
        public FormCadastro_Produtos()
        {
            InitializeComponent();
        }

        public void set_Opcao(string prstOpcao)
        {
            this.stOpcao = prstOpcao;
            //this seta o próprio objetos
        }

        public void Limpar()
        {
            //Instancia um Objeto Controle que é da Classe FormCadProduto
            //foreach roda itens dentro de arrays
            Control controles = this;
            foreach (Control ctr in controles.Controls)
            {
                if (ctr.GetType().ToString().Equals("System.Windows.Forms.TextBox"))
                {
                    ((TextBox)ctr).Text = "";
                }
            }
        }
    }
}
```

```
if (MessageBox.Show("Deseja sair da tela ?", "Interação", MessageBoxButtons.YesNo,
    MessageBoxIcon.Question, MessageBoxDefaultButton.Button1) == DialogResult.Yes)
{
    this.Close();
}
}
```

// Irá rodar em todos os controles do form, caso o controle seja do tipo TextBox, irá atribuir "", ou seja, limpará o controle

```
public void Preencher(string Id, string descricao, string preco)
{
    txtCodigo.Text = Id;
    txtDescricao.Text = descricao;
    txtPreco.Text = preco;
    _Old_id = int.Parse(Id);
    _Old_Descricao = descricao;
    _Old_Precio = decimal.Parse(preco);
}
```

```
private void btCancelar_Click(object sender, EventArgs e)
{
    Limpar();
}
```

```
private void btConfirmar_Click(object sender, EventArgs e)
{
    // Instancio a classe de cadastro, observe que não uso o DataSet, mas a outra classe ClasseCadastro
    Produto.ClassProduto classeProduto = new Produto.ClassProduto();
    // Atribuo valores para as propriedades da classe,
    // e estas por sua vez, atribuirão estes valores para os atributos.
    classeProduto.Original_Descricao = _Old_Descricao;
    classeProduto.Original_Precio = _Old_Precio;
    classeProduto.Id = int.Parse(txtCodigo.Text);
    classeProduto.Original_Id = _Old_id;
    classeProduto.Descricao = txtDescricao.Text;
    classeProduto.Precio = decimal.Parse(txtPreco.Text);

    if (this.stOpcao.Equals("I"))
    {
        classeProduto.Incluir();
        this.Limpar();
    }else if (this.stOpcao.Equals("A"))
    {

```

```
        classeProduto.Alterar();  
        this.Limpar();  
    }else{  
        MessageBox.Show("Implementar a Exclusão");  
    }  
}  
}  
}
```

Pronto, agora podemos começar os testes e implementar o que achar necessário para o “Bom Funcionamento desta tela”.

Camada Model e Business - Adotando Linguagem SQL

Esta camada implementa as classes de negócio do projeto, incluindo o modelo que representa a classe. Será responsabilidade desta camada conectar a base de dados, fornecer um modelo para classe bem como implementar os métodos de negócio, principalmente o CRUD.

A história básica do proposto é: Prover métodos básicos de CRUD para tabela de Produto bem como a tabela Movimento de forma relacionada. Alguns métodos de consulta tanto para Produto como Movimento serão igualmente implementados.

Toda implementação fará uso direto da linguagem SQL para implementar os métodos de negócio, o que caracteriza um mecanismo clássico, não fazendo uso de Frameworks como n-Hibernate ou Linq.

Iniciando o Projeto

1- Com o ambiente do VisualStudio em execução, selecione Menu File → New → Project. Conforme demonstrado na figura 01, defina o nome do projeto (NameSpace) como NameSpaceSQL.

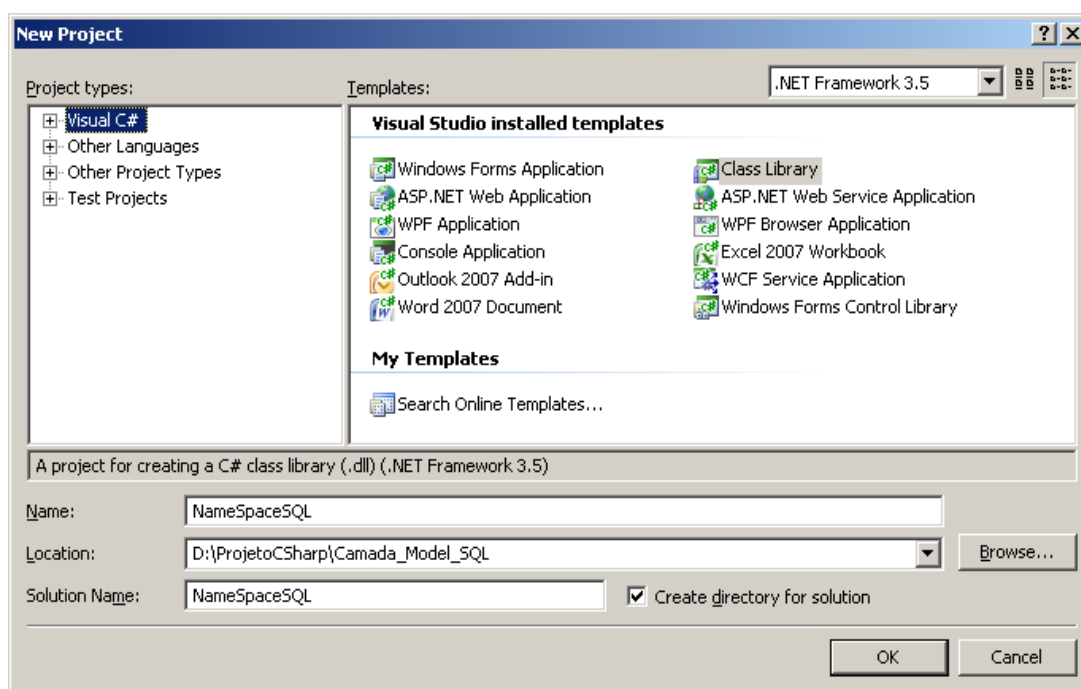


Figura 01

2- Inicialmente a janela Solution Explorer deverá ficar como exibido na figura 0, após renomear a Classe denominada Class1 para Produto.

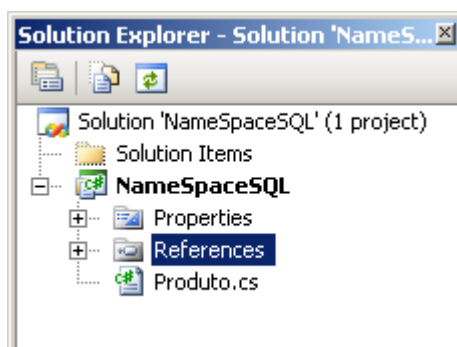


Figura 02

3- Adicione mais duas Classes, a saber, Classe Movimento e ListaMovimento. Isto é possível clicando com o botão direito do mouse sob o nó que representa a aplicação (NameSpaceSQL), escolhendo na opção Add → New Item, conforme figura 03, o Template Class.

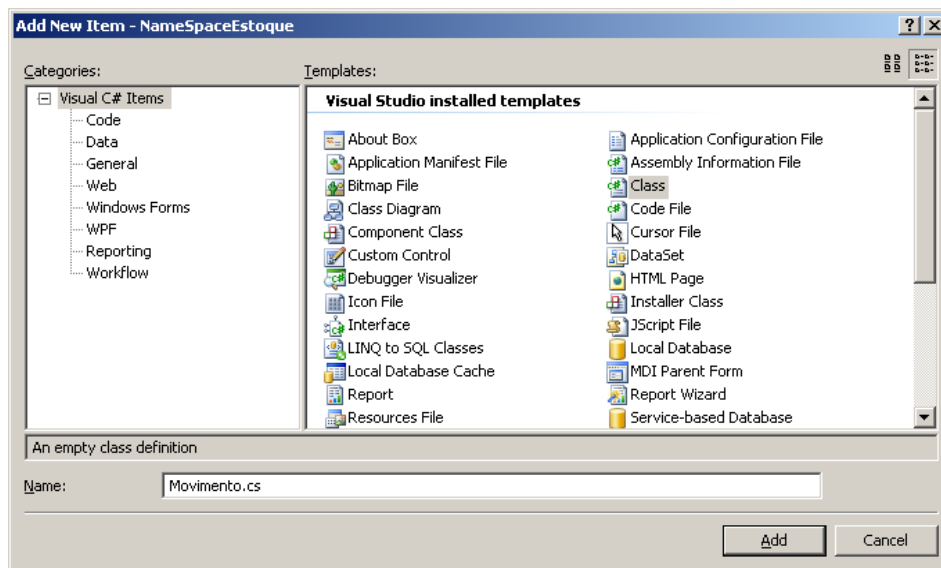


Figura 03

4- Agora providenciaremos a conexão com a base de dados, para tanto, clique com o botão direito do mouse sobre o nó Properties e faça a opção Open. Em seguida, baseando-se na figura 04, clique em Settings para adicionarmos uma propriedade que representará a conexão com o Banco de Dados.

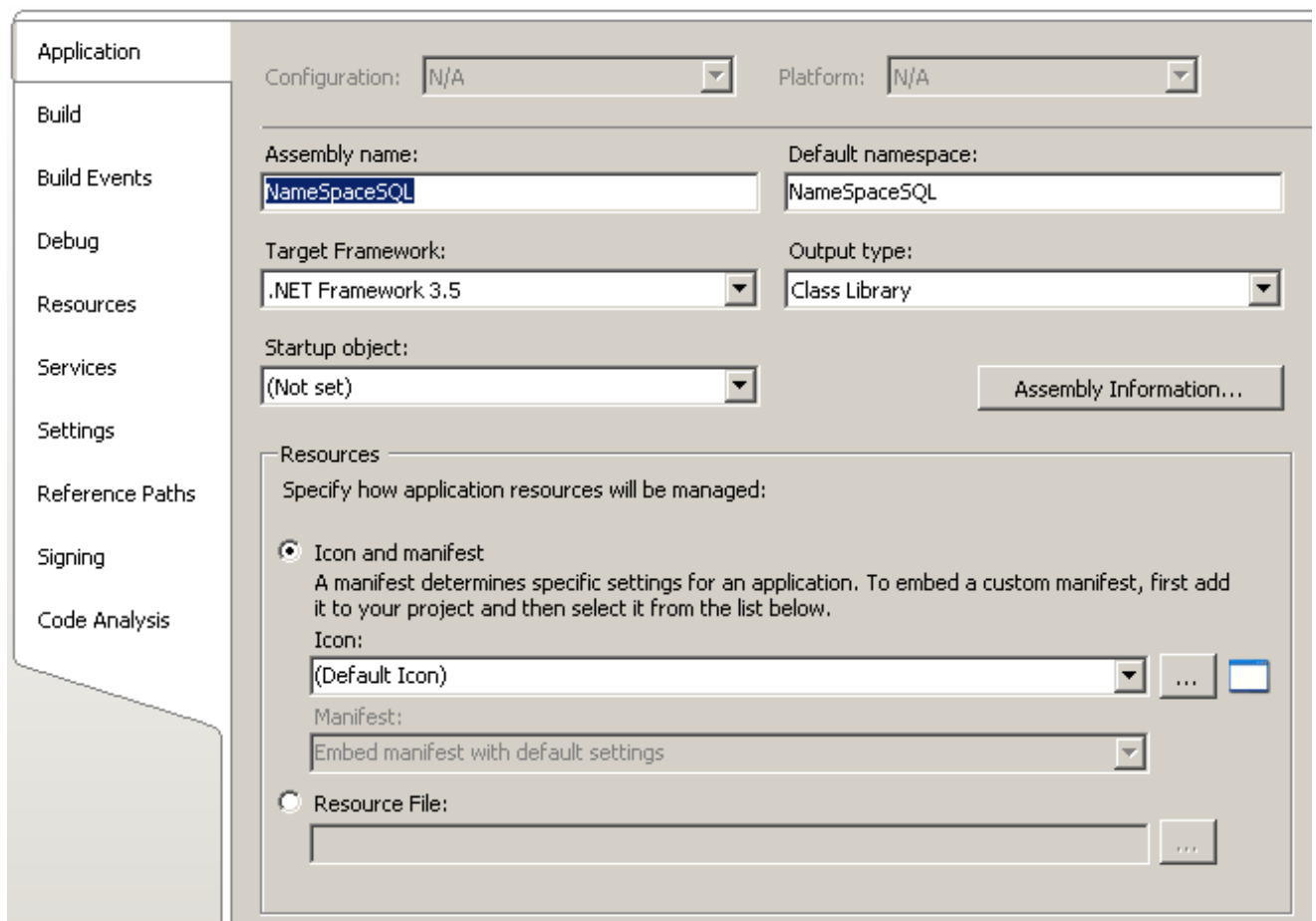


Figura 04

5- Preencha as propriedades Name, Type e Scope conforme exibido na figura 06. Para a propriedade Value, que define a string de conexão, clique nas reticências na própria propriedade Value, tomando por base a figura 05 que conecta uma base SQL Server.

Figura 05

	Name	Type	Scope	Value
▶	DBAula	(Connection string)	Application	Data Source=WS222\SQLEXPRESS;Initial Catalog=BancoAula;Persist Security Info=True;User ID=sa;Password=123
*				

Figura 06

6- Observe o código fonte da classe Properties, principalmente o trecho que define a propriedade DBAula que implementa a string de conexão com o Banco de Dados. Sendo necessário poderíamos alterar parâmetros desta conexão.

```
namespace NameSpaceSQL.Properties {
    [global::System.Runtime.CompilerServices.CompilerGeneratedAttribute()]
    internal sealed partial class Settings : global::System.Configuration.ApplicationSettingsBase {
        //... código omitido aqui
        [global::System.Configuration.DefaultSettingValueAttribute(
            "Data Source=WS222\SQLEXPRESS;Initial Catalog=BancoAula;Persist Security Info=True" +
            ";User ID=sa;Password=123")]
        public string DBAula {
            get {
                return ((string)(this["DBAula"]));
            }
        }
    }
}
```

Comentando Passo a Passo o Código Fonte da Classe Produto Partial (Model)

1- Referências (using) as classes necessárias ao desenvolvimento do código da Classe Produto.

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

2- Definição da Namespace que acomodará as classes Produto (partial class).

```
namespace NameSpaceSQL  
{
```

3- Definição da Classe Produto.

```
public partial class Produto  
{
```

Nota: Repare que na definição da classe foi utilizado da declaração "partial", indicando que o código do Modelo de Produto, aqui implementados, fazem parte de um todo. Conforme será comentado posteriormente, a Classe Produto também implementa o código relativo as Regras de Negócio e o CRUD.

4- Definição dos campos privados do Modelo Produto.

```
private long _id;  
private string _descricao;  
private decimal _preco;  
private List<Movimento> _movimento;  
private long _oldid;  
private string _olddescricao;  
private decimal _oldpreco;
```

5- Método Construtor da Classe Produto, que cria uma List da classe Movimento.

```
public Produto()  
{  
    this.Movimento = new List<Movimento>();  
}
```

Nota: Este método é necessário em função do projeto em questão, contemplar um relacionamento entre as Classes Produto e Movimento. Na Prática manipulamos uma coleção de objetos Movimento para cada representação de objeto Produto. Isso ocorrerá quando da inclusão de um Produto, poderemos também incluir um ou mais objetos Movimento. Os comentários relativo a Inclusão de Produto, implementado na classe Produto Partial que expressa a Lógica de Negócio da classe darão luz a este tópico.

6- Os métodos descritos abaixo definem Get(s) e Set(s) para os membros privados da classe, comentados no tópico 4.

Nota: Para criar esses métodos, não é necessário digitá-los no corpo da classe, bastando apenas invocar os recursos representados pela figuras abaixo, da seguinte forma: Clique com o botão direito do mouse, sobre o nome do membro privado ID, faça a opção Refactor → Encapsulate Field... e a caixa de dialogo representada pela figura 07 será exibida. Pronto, basta clicar no botão OK e os métodos Set e Get para este membro privado será criado. Repita os passos para os campos privados restantes.

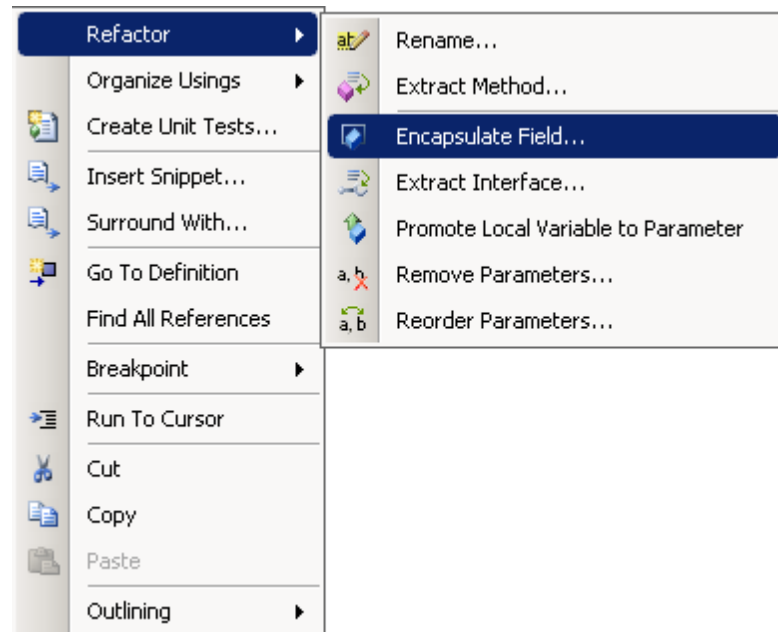


Figura 06

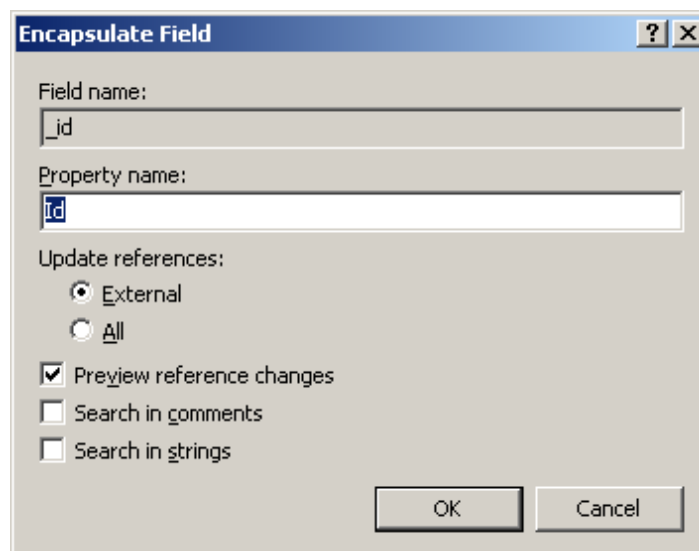


Figura 07

7- Métodos Set e Get para o campo privado ID, criado a partir do tópico 7.

```
public long Id
{
    get { return _id; }
    set { _id = value; }
}
```

8- Restante dos métodos Set e Get para os campos privados restantes da Classe Produto (Modelo).

```
public string Descricao
{
    get { return _descricao; }
    set { _descricao = value; }
}

public decimal Preco
{
    get { return _preco; }
    set { _preco = value; }
}

public long OldId
{
    get { return _oldid; }
    set { _oldid = value; }
}

public string OldDescricao
{
    get { return _olddescricao; }
    set { _olddescricao = value; }
}

public decimal OldPreco
{
    get { return _oldpreco; }
    set { _oldpreco = value; }
}

public List<Movimento> Movimento
{
    get { return _movimento; }
    set { _movimento = value; }
}
```

9- Encerra o bloco que define a Classe Produto (partial).

```
}
```

10- Encerra o bloco que define a Classe Produto (partial)

```
}
```

Comentando Passo a Passo o Código Fonte da Classe Produto Partial (Regras de Negócio)

1- Referências (using) as classes necessárias ao desenvolvimento do código da Classe Produto.

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

2- Definição da Namespace que acomodará as classes Produto (partial class).

```
namespace NameSpaceSQL {
```

3- Definição da Classe Produto.

```
public partial class Produto
{
```

Nota: Repare que na definição da classe foi utilizado da declaração "partial", indicando que o código (regras de negócio) aqui implementados, fazem parte de um todo. Conforme apresentado anteriormente, a Classe Produto também implementa o código que define o Modelo Produto.

3.1- Atribuição da String de conexão a variável strconn do tipo String.

```
String strconn = NameSpaceSQL.Properties.Settings.Default.DBAula;
```

3.2- Criando instância da Classe de Tratamento de Erro.

```
ClasseErro ClasseErro = new ClasseErro();
```

Nota: A conexão com o banco de dados na Classe, e foi abordada nos tópicos relativo as figuras 04 e 05.

4- Comentário detalhado do Método Recursivo RetornaChave.

4.1-

Define o método como uma função que retorna um Inteiro Longo e parâmetro de entrada String. O objetivo do método é obter junto a tabela GerarChave (ver dados e estrutura listados abaixo), um valor que represente o último valor mais um, para o registro cujo o campo NomeCampoChave seja igual ao parâmetro passado para o método.

	ID	NOMECAMPOCHAVE	ULTIMONUMERO
▶	1	ID_CLIENTE	100
	2	ID_PRODUTO	120
	3	ID_MOVIMENTO	71
	4	ID_PEDIDO	100
	5	ID_ITEM	1001
*	NULL	NULL	NULL

```
public long RetornaChave(string NomeCampo)
{
```

4.2-

Respectivamente cria um objeto denominado conn (implementa conexão com base de dados), que recebe como parâmetro a variável strconn, definida em no passo 4. Na sequência é executado pelo método Open.

```
SqlConnection conn = new SqlConnection( strconn);
conn.Open();
```

4.3-

Define a string que contem o código SQL para obtenção do valor do campo UltimoNumero.

```
String sqlUltimoNumero = " Select UltimoNumero From GerarChave " +
    " Where NomeCampoChave = " + "" + NomeCampo + "";
```

4.4-

Define e cria um objeto do tipo SqlCommand tendo como parâmetro a instrução SQL sqlUltimoNumero e o objeto de conexão conn implementado e executado no passo 4.2.

```
SqlCommand cmdGerarChave = new SqlCommand(sqlUltimoNumero, conn);
```

4.5-

Cria variável Numérica Inteira que recebe o valor retornado do método ExecuteScalar() do objeto cmdGerarChave. Duas observações importantes são: O Próprio método ExecuteScalar(), utilizado quando espera-se um único valor de retorno, a outra observação é o uso da Classe Int64 colocada entre parenteses para converter o valor retornado pla instrução SQL.

```
Int64 ValorCampoOld = (Int64)cmdGerarChave.ExecuteScalar();
```

4.6-

Instrução SQL responsável pela atualização do campo UltimoNumero acrescido de um. Repare que por questões de trtar a concorrência de dados, a cláusula Where da instrução compara não só o valor para o campo NomeCampoChave, como o valor do campo UltimoNumero, obtido a partir da variável ValorCampoOld, que contem o valor deste campo lido anteriormente na tabela. Isso garante que somente será executado o comando Update, caso entre a leitura (Select) e o momento do Update, o registro não tenha sido modificado.

```
String sqlAtualizaNumero =
    " Update GerarChave Set UltimoNumero = " +
    " UltimoNumero + 1 " +
    " Where NomeCampoChave = " + "" + NomeCampo + "" +
    " AND UltimoNumero = " + Convert.ToString(ValorCampoOld);
```

4.7-

Define e cria um objeto do tipo SqlCommand tendo como parâmetro a instrução SQL sqlAtualizaNumero e o objeto de conexão conn implementado e executado no passo 4.2.

```
SqlCommand cmdAtualizaNumero = new SqlCommand(sqlAtualizaNumero, conn);
```

4.8-

Executa o método `ExecuteNonQuery()` para o objeto `cmdAtualizaNumero` em uma condição IF que verifica se o retorno numérico é zero. Caso seja, é em função do fato de que a instrução `Update` do comando `cmdAtualizaNumero` não afetou o registro definido em sua cláusula `Where`. O motivo para tanto foi descrito no tópico 4.7.

Sendo verdadeiro a condição IF, o método é executado recursivamente até retornar um valor (sucesso na execução do comando `Update`).

```
if (cmdAtualizaNumero.ExecuteNonQuery() == 0)
{
    RetornaChave(NomeCampo);
}
```

4.9-

Na sequência, é definido um objeto do tipo `Long` (Numérico Inteiro Longo), é atribuído ao mesmo o valor da variável `ValorCampoOld` mais um, e por fim, o método retorna tal valor.

```
long ValorNovo;
ValorNovo = (ValorCampoOld + 1);
return ValorNovo;
```

4.10-

Encerra o bloco relativo ao método `RetornaChave`.

```
}
```

5- Código relativo ao Método de Negócio `InserirProdutoMovimento` (CRUD)

5.1-

Define o método sem parâmetros com retorno do tipo `String`, Cria e executa o método `Open` para um objeto tipo `SqlConnection`. Na sequência este objeto será responsável por controlar a transação da regra de negócio que o método implementa.

Vale ressaltar aqui, que se faz necessário um controle explícito de transação com a base de dados, já que o método tem como objetivo, incluir um produto na tabela `Produto`, e um movimento na tabela `Movimento`. Logo, devemos garantir a condição atômica para a transação, onde ambos os registros são criados ou nenhum registro é criado.

```
public string InserirProdutoMovimento()
{
    SqlConnection conn = new SqlConnection(strconn);
    conn.Open();
```

5.2-

Define um objeto do tipo `SqlTransaction`, sendo atribuído ao mesmo, o início de transação para a conexão representada pelo objeto `conn`.

```
SqlTransaction transacao = conn.BeginTransaction();
```

5.3-

Inicia um bloco `try / catch / finally`. Seu objetivo, é a partir de cláusula `try`, monitorar a execução das instruções do método, observando alguma exceção e desviando para área de tratamento de exceções denominada `catch`. Ocorrendo ou não exceção, a cláusula (todo o bloco) `finally` é executada.

```
try
{
```

5.8-

Define uma string que recebe como atribuição a instrução Insert para Produto. Repare nos parâmetros posicionados como valores a serem atribuídos aos campos da tabela Produto.

```
String sqlProduto = " Insert Into Produto Values(@Id, @Descricao, @Preco) ";
```

5.5-

Define o objeto do tipo SqlCommand para inserção de Produto. Repare que o método recebe como parâmetro sqlProduto contendo a instrução Insert, conn que representa o objeto de conexão, e por fim, transacao, que representa o objeto de transação em curso.

```
SqlCommand cmdProduto = new SqlCommand(sqlProduto, conn, transacao);
```

5.6-

Atribuição dos membros públicos da Classe Produto, a saber, Id, Descricao e Preco para os parâmetros @Id, @Descricao e @Preco pertencentes a instrução Insert contida no objeto cmdProduto.

```
cmdProduto.Parameters.AddWithValue("@Id", this._id);  
cmdProduto.Parameters.AddWithValue("@Descricao", this._descricao);  
cmdProduto.Parameters.AddWithValue("@Preco", this._preco);
```

5.7-

Executa o método ExecuteNonQuery() para o objeto cmdProduto, instruindo o banco de dados na inclusão de um Produto. Este método é útil para execução de instruções Insert, Update e Delete.

```
cmdProduto.ExecuteNonQuery();
```

5.8-

Cria um laço do tipo foreach, que navega pelos registros contidos no membro this.Movimento (objetos Movimento). Para cada objeto Movimento é montado uma instrução Insert tendo como valores os campos do objeto Movimento da lista.

Observe que o primeiro parâmetro de foreach é um tipo Movimento e o segundo, após in, representa a coleção Movimento definida no modelo Produto.

```
foreach (Movimento Movimento in this.Movimento)  
{  
    String sqlMovimento = " Insert Into Movimento Values(@Id, @ProdutoId, " +  
        " @Tipo_MovimentoId, @Data, @Quantidade, @Valor) ";  
    SqlCommand cmdMovimento = new SqlCommand(sqlMovimento, conn, transacao);  
    cmdMovimento.Parameters.AddWithValue("@Id", Movimento.Id);  
    cmdMovimento.Parameters.AddWithValue("@ProdutoId", Movimento.ProdutoId);  
    cmdMovimento.Parameters.AddWithValue("@Tipo_MovimentoId", Movimento.Tipo_movimentoid);  
    cmdMovimento.Parameters.AddWithValue("@Data", Movimento.Data);  
    cmdMovimento.Parameters.AddWithValue("@Quantidade", Movimento.Quantidade);  
    cmdMovimento.Parameters.AddWithValue("@Valor", Movimento.Valor);  
    cmdMovimento.ExecuteNonQuery();  
}
```

5.9-

Executa o método Commit() para o objeto transacao, instruindo o banco de dados concluir a inclusão do Produto e dos objetos Movimento em definitivo, encerrando a transação.

```
transacao.Commit();
```


5.10-

Atribui uma string vazia para retorno do método. Em seguida encerra o bloco try, desviando de catch.

```
    return "";  
}
```

5.11-

Caso ocorra erro entre o início do bloco try e catch, as instruções deste último bloco são executadas, instruindo o banco de dados a encerrar cancelando a transação (transacao.Rollback();) e retornando pelo método, a mensagem de erro retornada pelo método TrataErro da Classe ClasseErro tendo como parâmetro a propriedade Message da instância de Exception representada pelo objeto Erro.

```
    catch (Exception Erro)  
    {  
        transacao.Rollback();  
        conn.Close();  
        return ClasseErro.TrataErro(Erro.Message);  
    }
```

5.12-

Sempre executado, o bloco finally implementa o fechamento da conexão representada pelo objeto conn. Na prática, como observado no assunto Garbage Collector, sabemos que na verdade quem controla o destruir de um objeto é este mecanismo, ficando a cargo do método simplesmente indicar que o recurso está liberado para este. Isso também vale para os métodos Open e Close das conexões de banco de dados, que são gerenciadas pelo pooled de conexões da middleware ADO.NET.

```
    finally  
    {  
        conn.Close();  
    }  
}
```

5.13-

Os métodos AlterarProduto e ExcluirProduto não serão comentados pois sua lógica já foi analisada e comentada nos métodos anteriores.

```
public string AlterarProduto()  
{  
    SqlConnection conn = new SqlConnection(strconn);  
    conn.Open();  
    SqlTransaction transacao = conn.BeginTransaction();  
    try  
    {  
        String sqlProduto = " Update Produto Set " +  
                             " Descricao = @Descricao, " +  
                             " Preco = @Preco Where " +  
                             " Id = @Old_ID AND " +  
                             " Descricao = @Old_descricao AND " +  
                             " Preco = @Old_Precos ";  
        SqlCommand cmdProduto = new SqlCommand(sqlProduto, conn, transacao);  
        cmdProduto.Parameters.AddWithValue("@Descricao", this._descricao);  
        cmdProduto.Parameters.AddWithValue("@Preco", this._preco);  
        cmdProduto.Parameters.AddWithValue("@Old_ID", this._oldid);  
        cmdProduto.Parameters.AddWithValue("@Old_descricao", this._olddescricao);  
        cmdProduto.Parameters.AddWithValue("@Old_Precos", this._oldpreco);  
    }  
}
```

```
        if cmdProduto.ExecuteNonQuery() == 0
        {
            transacao.Rollback();
            conn.Close();
            return "Produto não Alterado. Ou o registro foi alterado ou excluído desde a obtenção.";
        }
        transacao.Commit();
        return "";
    }

    catch(Exception Erro){
        transacao.Rollback();
        conn.Close();
        return ClasseErro.TrataErro(Erro.Message);
    }
    finally {
        conn.Close();
    }
}

public string ExcluirProduto()
{
    SqlConnection conn = new SqlConnection(strconn);
    conn.Open();
    SqlTransaction transacao = conn.BeginTransaction();
    try{
        String sqlProduto =
            " Delete Produto Where Id = @ID AND Descricao = @Descricao AND Preco = @Preco ";
        SqlCommand cmdProduto = new SqlCommand(sqlProduto, conn, transacao);
        cmdProduto.Parameters.AddWithValue("@Old_ID", this._oldId);
        cmdProduto.Parameters.AddWithValue("@Old_descricao", this._olddescricao);
        cmdProduto.Parameters.AddWithValue("@Old_Preco", this._oldpreco);
        if cmdProduto.ExecuteNonQuery() == 0
        {
            transacao.Rollback();
            conn.Close();
            return "Produto não Excluído. Ou o registro foi alterado ou excluído desde a obtenção.";
        }
        transacao.Commit();
        return "";
    }
    catch(Exception Erro){
        transacao.Rollback();
        conn.Close();
        return ClasseErro.TrataErro(Erro.Message);
    }
    finally{
        conn.Close();
    }
}
```

5.14-

Método responsável por retornar um registro da tabela Produto, baseado no valor de sua chave-primária. O método retorna um tipo DataTable, ideal para acomodar registros de uma tabela. Para implementar este mecanismo é necessário criar além de um objeto SqlCommand, um objeto SqlDataAdapter e por fim um objeto DataSet. Utilizamos o método Fill deste último para preencher o objeto Table com os valores retornados pelo command cmdproduto.

```
public DataTable RetornaProduto(long Id)
{
    SqlConnection conn = new SqlConnection(strconn);
    conn.Open();
    String sqlProduto = " Select * From Produto Where Produto.Id = @Id ";
    SqlCommand cmdProduto = new SqlCommand(sqlProduto, conn);
    cmdProduto.Parameters.AddWithValue("@Id", Id);
    SqlDataAdapter da = new SqlDataAdapter();
    da.SelectCommand = cmdProduto;
    DataSet ds = new DataSet();
    da.Fill(ds);
    return ds.Tables[0];
}
```

5.15-

Os métodos RetornaProduto sobrecarregados e o método RetornaMovimento, não serão comentados já que são semelhantes ao método anterior, devidamente comentado e explicado.

```
public DataTable RetornaProduto(string Descricao)
{
    SqlConnection conn = new SqlConnection(strconn);
    conn.Open();
    String sqlMovimento = " Select * From Produto Where Produto.Descricao Like @Descricao ";
    SqlCommand cmdMovimento = new SqlCommand(sqlMovimento, conn);
    cmdMovimento.Parameters.AddWithValue("@Descricao", Descricao + "%");
    SqlDataAdapter da = new SqlDataAdapter();
    da.SelectCommand = cmdMovimento;
    DataSet ds = new DataSet();
    da.Fill(ds);
    return ds.Tables[0];
}
```

```
public DataTable RetornaProduto(Decimal Preco)
{
    SqlConnection conn = new SqlConnection(strconn);
    conn.Open();
    String sqlProduto = " Select * From Produto Where Produto.Preco >= @Preco";
    SqlCommand cmdProduto = new SqlCommand(sqlProduto, conn);
    cmdProduto.Parameters.AddWithValue("@Preco", Preco);
    SqlDataAdapter da = new SqlDataAdapter();
    da.SelectCommand = cmdProduto;
    DataSet ds = new DataSet();
    da.Fill(ds);
    return ds.Tables[0];
}
```

```
public DataTable RetornaMovimento(String DecriProduto, DateTime DataIni, DateTime DataFin)
{
    SqlConnection conn = new SqlConnection(strconn);
    conn.Open();
    String sqlMovimento = " Select * From vw_ListaMovimento Where " + " Descricao_Produto Like " +
        " @Produto AND Data Between @DataIni AND @DataFin";
    SqlCommand cmdMovimento = new SqlCommand(sqlMovimento, conn);
    cmdMovimento.Parameters.AddWithValue("@Produto", DecriProduto + "%");
    cmdMovimento.Parameters.AddWithValue("@DataIni", String.Format("{0:MM/dd/yyyy}", DataIni));
    cmdMovimento.Parameters.AddWithValue("@DataFin", String.Format("{0:MM/dd/yyyy}", DataFin));
    SqlDataAdapter da = new SqlDataAdapter();
    da.SelectCommand = cmdMovimento;
    DataSet ds = new DataSet();
    da.Fill(ds);
    return ds.Tables[0];
}
```

5.16-

Encerra o bloco da Casse.

```
}
```

5.17-

Encerra o bloco da NameSpace.

```
}
```

Código Fonte da Classe Movimento (Model)

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

namespace NameSpaceSQL
{
    public class Movimento
    {
        private long _id;
        private long _produtoId;
        private long _tipo_movimentoid;
        private DateTime _data;
        private Decimal _quantidade;
        private Decimal _valor;

        public Movimento() { }

        public long Id
        {
            get { return _id; }
            set { _id = value; }
        }

        public long ProdutoId
        {
            get { return _produtoId; }
            set { _produtoId = value; }
        }

        public long Tipo_movimentoid
        {
            get { return _tipo_movimentoid; }
            set { _tipo_movimentoid = value; }
        }

        public DateTime Data
        {
            get { return _data; }
            set { _data = value; }
        }

        public Decimal Quantidade
        {
            get { return _quantidade; }
            set { _quantidade = value; }
        }

        public Decimal Valor
        {
            get { return _valor; }
            set { _valor = value; }
        }
    }
}
```

Código Fonte da Classe ListaMovimento (Model)

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

namespace NameSpaceSQL
{
    public class ListaMovimento
    {

        public ListaMovimento()
        {

        }

        public ListaMovimento(
            long Id, long ProdutoId,
            long Tipo_movimentoid,
            DateTime Data,
            Decimal Quantidade,
            Decimal Valor,
            string Descricao_Movimento,
            string Descricao_Produto)
        {

        }

        private long _id;
        private long _produtoId;
        private long _tipo_movimentoid;
        private Decimal _quantidade;
        private DateTime _data;
        private Decimal _valor;
        private string _Descricao_Movimento;
        private string _Descricao_Produto;

        public long Id
        {
            get { return _id; }
            set { _id = value; }
        }

        public long ProdutoId
        {
            get { return _produtoId; }
            set { _produtoId = value; }
        }

        public long Tipo_movimentoid
        {
            get { return _tipo_movimentoid; }
            set { _tipo_movimentoid = value; }
        }
    }
}
```

```
public DateTime Data
{
    get { return _data; }
    set { _data = value; }
}
```

```
public Decimal Quantidade
{
    get { return _quantidade; }
    set { _quantidade = value; }
}
```

```
public Decimal Valor
{
    get { return _valor; }
    set { _valor = value; }
}
```

```
public string Descricao_Movimento
{
    get { return _Descricao_Movimento; }
    set { _Descricao_Movimento = value; }
}
```

```
public string Descricao_Produto
{
    get { return _Descricao_Produto; }
    set { _Descricao_Produto = value; }
}
}
```

```
}
```

Código Fonte da Classe ClasseErro

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

namespace NameSpaceSQL
{
    class ClasseErro
    {
        public string TrataErro(String MsgErro)
        {
            if (MsgErro.IndexOf("Violation") != -1)
            {
                return "Erro por violação de chave única.";
            }
            else if (MsgErro.IndexOf("Cannot insert duplicate key row in object") != -1)
            {
                return "Erro por violação de chave primária.";
            }
            else if (MsgErro.IndexOf("conflicted with the REFERENCE constraint") != -1)
            {
                return "Existe registro tem relacionamentos, logo não pode ser excluído.";
            }
            else
            {
                return "Erro Ocorrido: " + MsgErro + " >>> Consulte o suporte <<<.";
            }
        }
    }
}
```


Camada View e Controller

Teremos aqui um projeto Web baseado em Asp.Net que representará a camada View, nela, teremos uma classe que servirá como camada Controller, objetivando a ligação entre as páginas Asp.Net (Camada View) e os métodos de negócio na camada Model.

A história básica do proposto é: Implementar interfaces para cadastramento de produtos (Classe Produto), apresentar uma consulta para pesquisa de produtos, e por fim uma listagem contendo movimentos.

Criando a Camada View

1- Com o ambiente do VisualStudio em execução, no menu File → Web Site, faça a opção ASP.Net Web Site conforme figura 01.

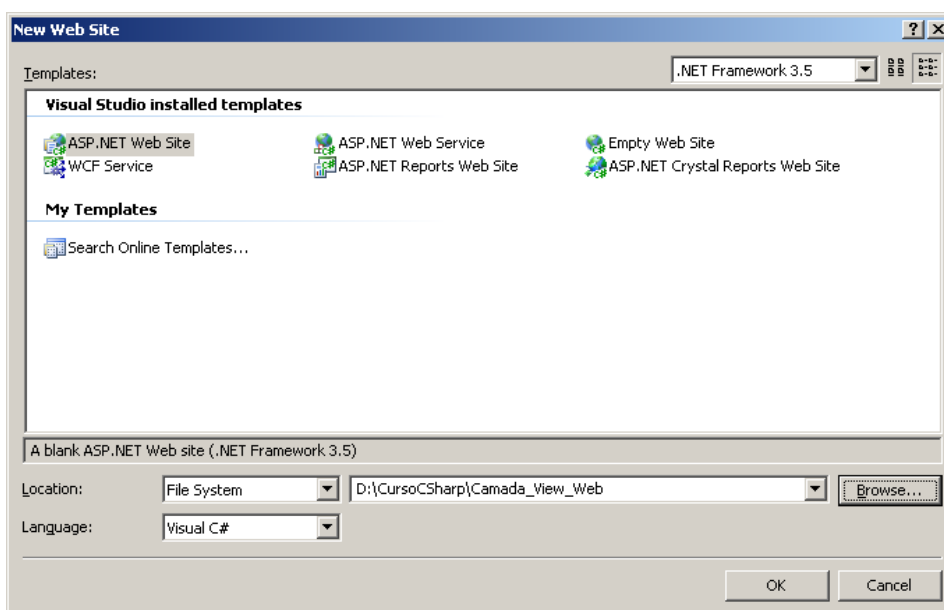


Figura 01

2- Inicialmente a janela Solution Explorer deverá ficar como o exibido na figura 02.

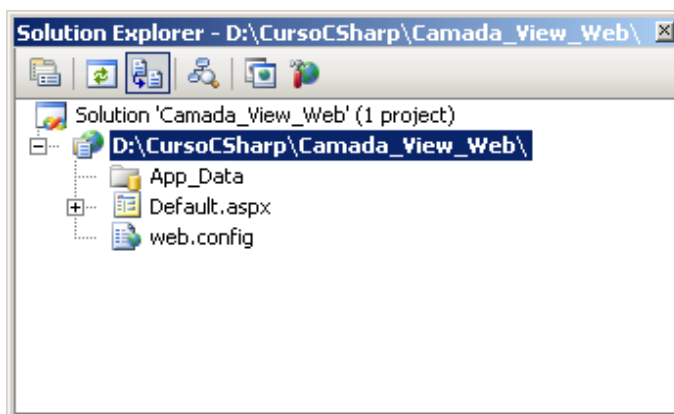


Figura 02

3- Dê um duplo clique sobre Default.aspx na árvore da Solution Explorer visualizando a página no modo Design. A área de aderência agora pode receber controles para compor a página. Clique na aba que define o modo Source observando o código resultante.

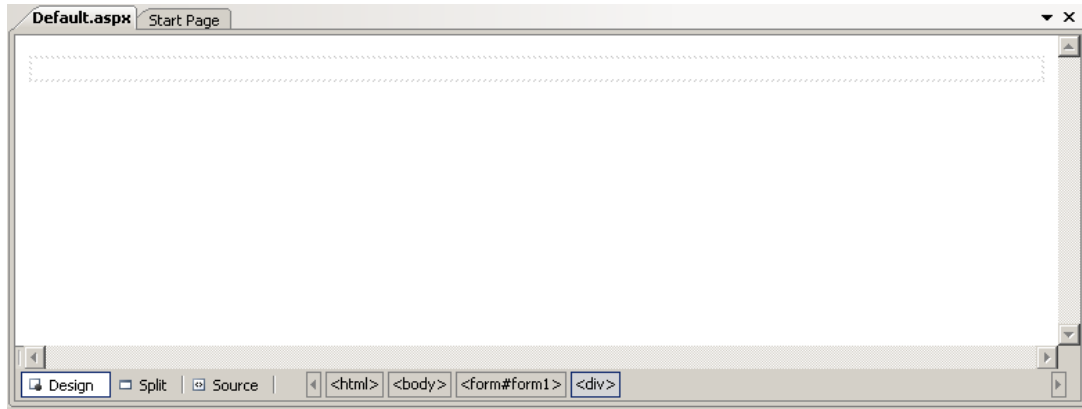


Figura 03

4- Visando ter um modelo que sirva de base para todas as páginas de nosso projeto, vamos adicionar agora uma Master Page. Clique sob o nó raiz do projeto escolhendo Add → New Item escolhido o Template Master Page. A figura 04 nos orienta neste sentido. Observe ainda o código fonte (aba Source) resultante.

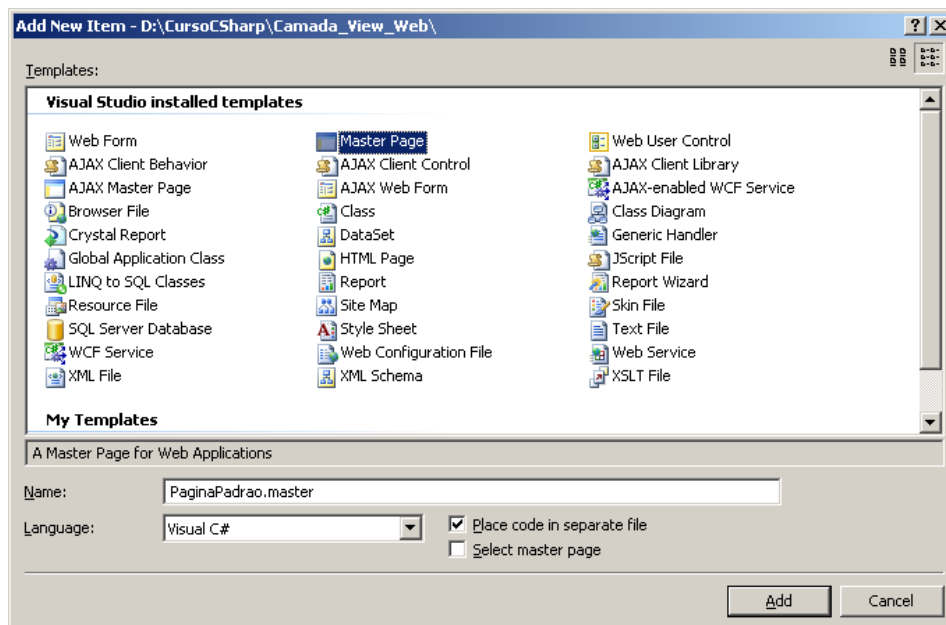


Figura 04

```
<%@ Master Language="C#" AutoEventWireup="true" CodeFile="PaginaPadrao.master.cs"
    Inherits="PaginaPadrao" %>
<!DOCTYPE html PUBLIC
    "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head runat="server">
    <title>Untitled Page</title>
    <asp:ContentPlaceHolder id="head" runat="server"> </asp:ContentPlaceHolder>
  </head>
  <body>
    <form id="form1" runat="server">
      <div>
        <asp:ContentPlaceHolder id="ContentPlaceHolder1" runat="server">
          </asp:ContentPlaceHolder>
        </div>
      </form>
    </body>
  </html>
```

5- Adicione os diretórios Css, Img e Js contendo respectivamente o arquivo de folha de estilo, as imagens a serem aplicadas ao projeto, e por fim, o arquivo contendo as instruções JavaScript. Conforme demonstrado na figura 05 clique em Refresh Folder para que a janela Solution Explore possa refletir a nova estrutura de diretórios.

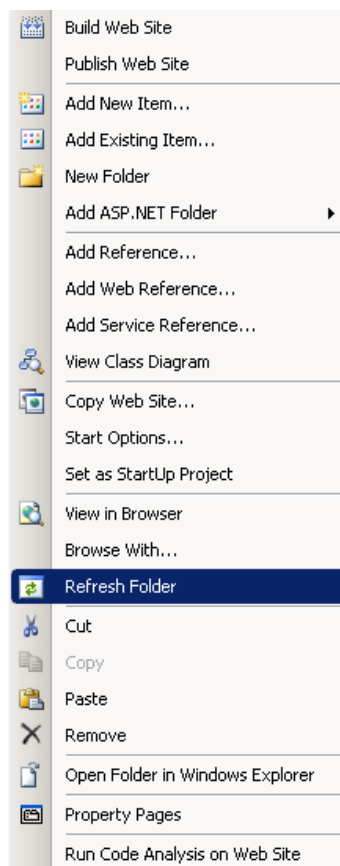


Figura 05

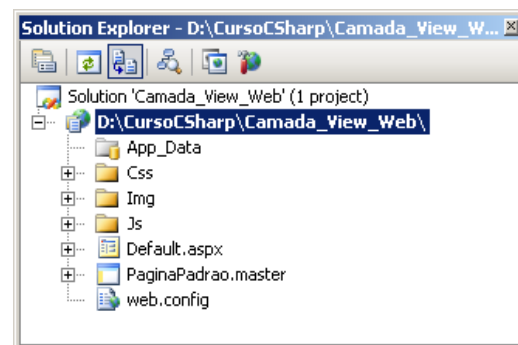


Figura 06

6- Aplique sob a página Master Page um controle Image oriundo da janela Tool Box, configurando a propriedade ImageURL conforme figuras 07 e 08.

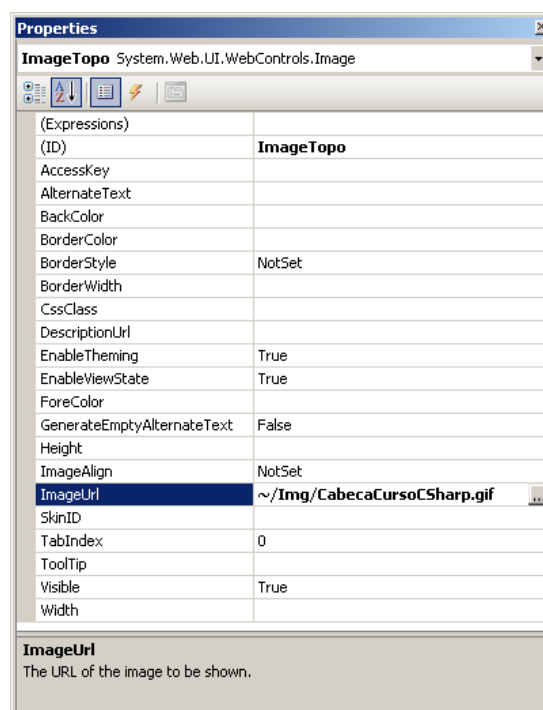


Figura 07

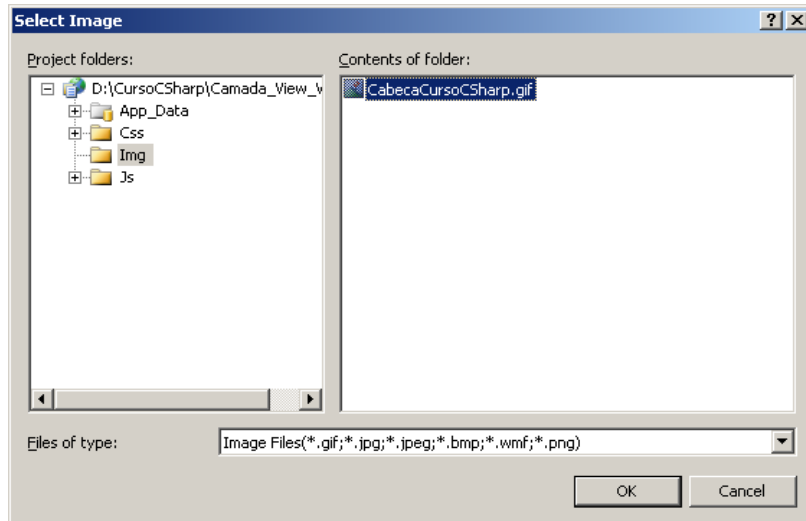


Figura 08

7- Observe, clicando na aba source da página, o código fonte resultante. Especial atenção a tag Image e suas propriedades descrita no código fonte da página. Atente também para o resultado visual da página Master Page.

```
<%@ Master Language="C#" AutoEventWireup="true" CodeFile="PaginaPadrao.master.cs"
    Inherits="PaginaPadrao" %>
```

//... Código omitido aqui

```
<asp:ContentPlaceholder
    id="head"
    runat="server">
</asp:ContentPlaceholder>
</head>
<body>
    <form id="form1" runat="server">
        <div align="center">
            <div style="position: relative; width: 800px; height:100px">

                <asp:Image
                    ID="ImageTopo" runat="server"
                    Height="80px" Width="800px"
                    ImageUrl="~/Img/CabecaCursoCSharp.gif"
                    style="z-index: 1;
                    left: 0px; top: 0px;
                    position: relative;
                    border: 1px solid #808080;"

                />

            <asp:ContentPlaceholder id="ContentPlaceholder1" runat="server">
            </asp:ContentPlaceholder>
            </div>
        </div>
    </form>
</body>
</html>
```

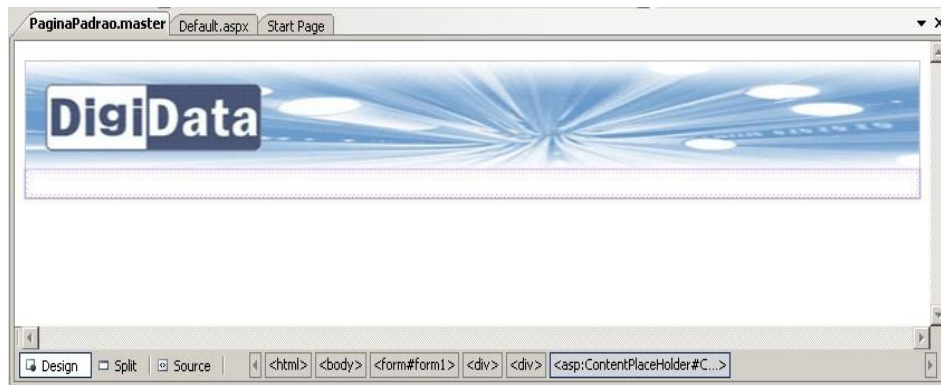


Figura 09

8- Adicione uma nova página ao projeto definindo seu nome como Index.aspx. Para tanto, clicando no botão direito do mouse sob o nó raiz do projeto escolhendo Add New Item. Repare neste caso, o checkbox Select master page assinalado, gerando a caixa de dialogo representada pela figura 11. Nesta, selecione uma Master Page para servir de Template para a página Index.

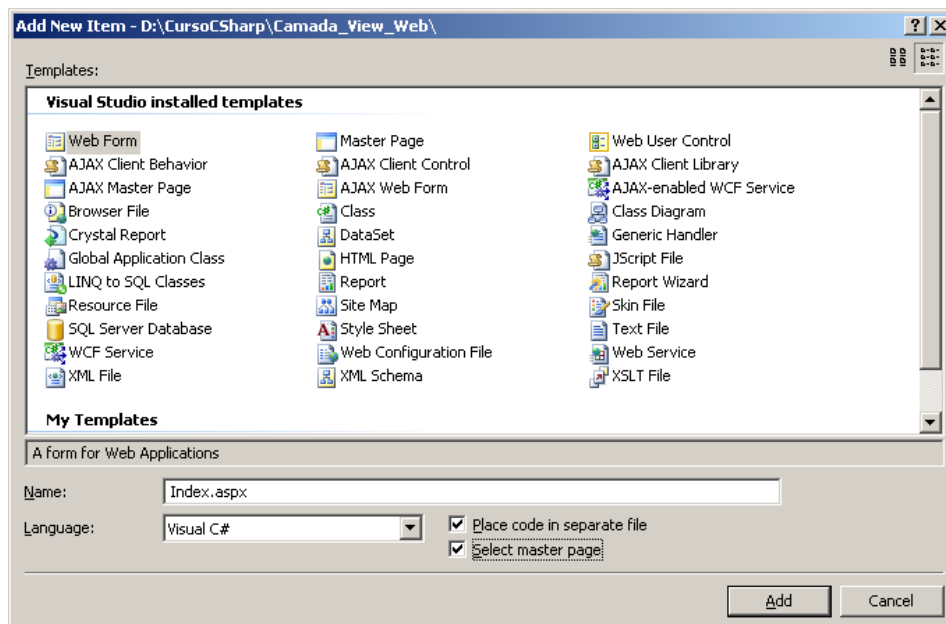


Figura 10

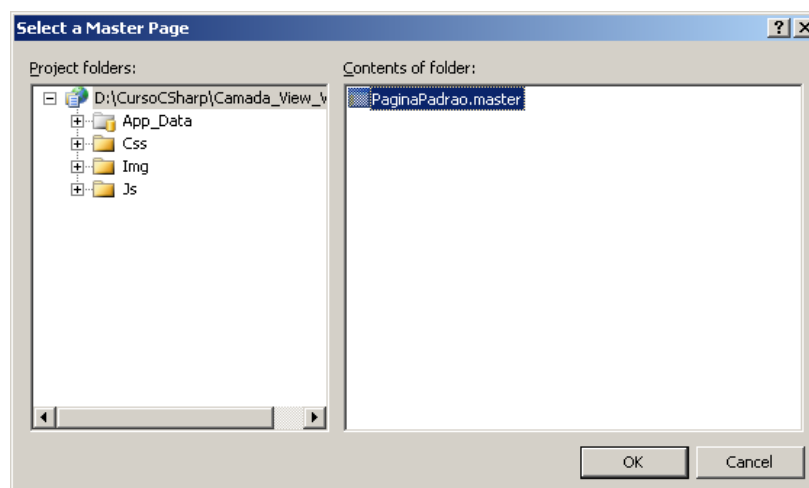


Figura 11

9- Adicione um controle Menu oriundo da caixa de dialogo ToolBox → Navigation. Utilize o Menu Tasks localizado na parte superior esquerda na forma de uma seta, selecionando a opção Edit Menu Items..., representada pela figura 12. Adicione menus e itens de menu conforme ilustrado na sequência representada pelas figuras 12, 13, 14, 15, 16 e 17.

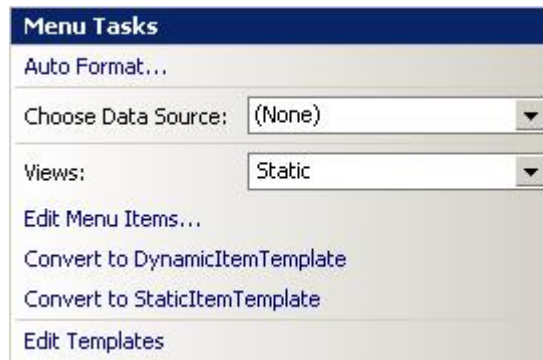


Figura 12

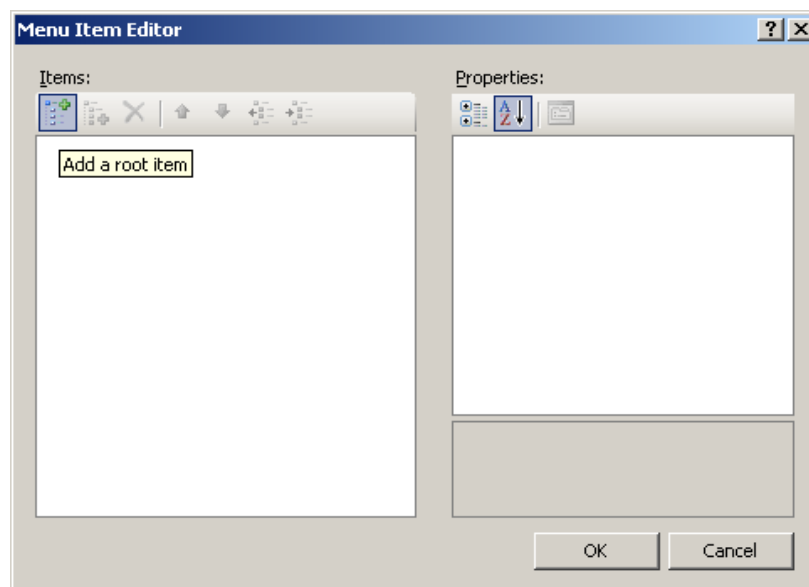


Figura 13

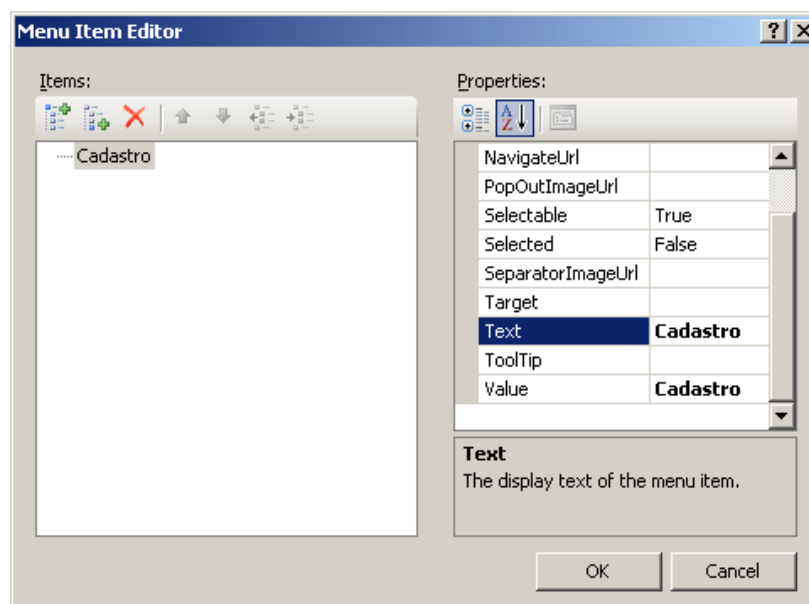


Figura 14

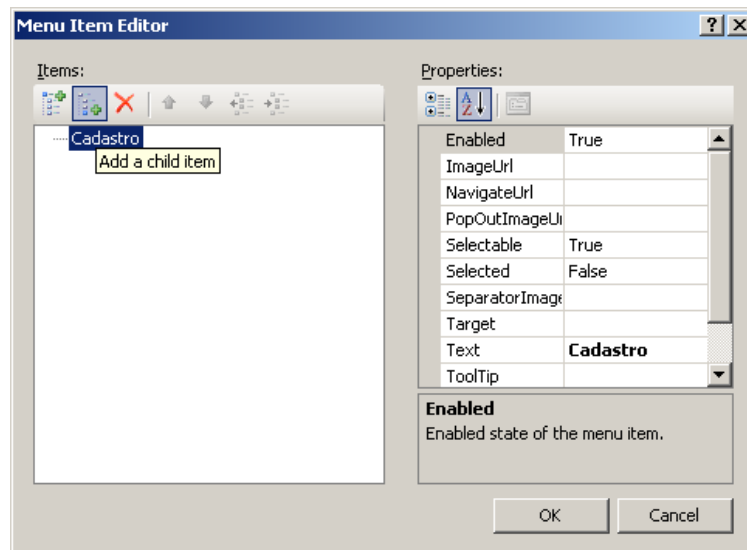


Figura 15

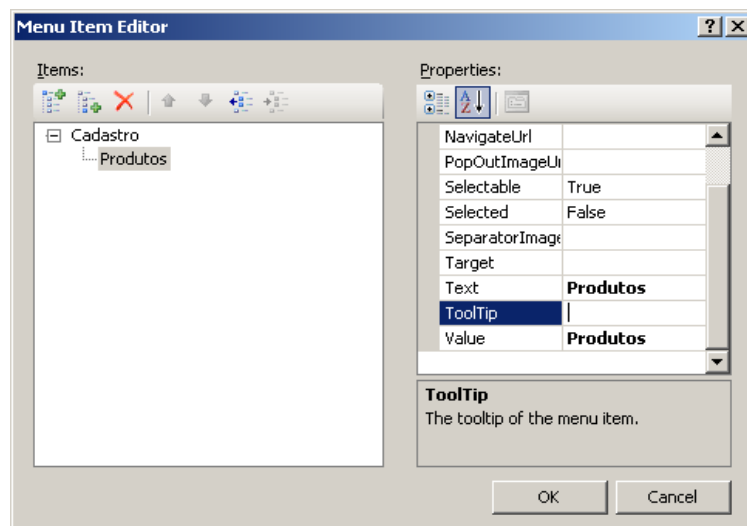


Figura 16

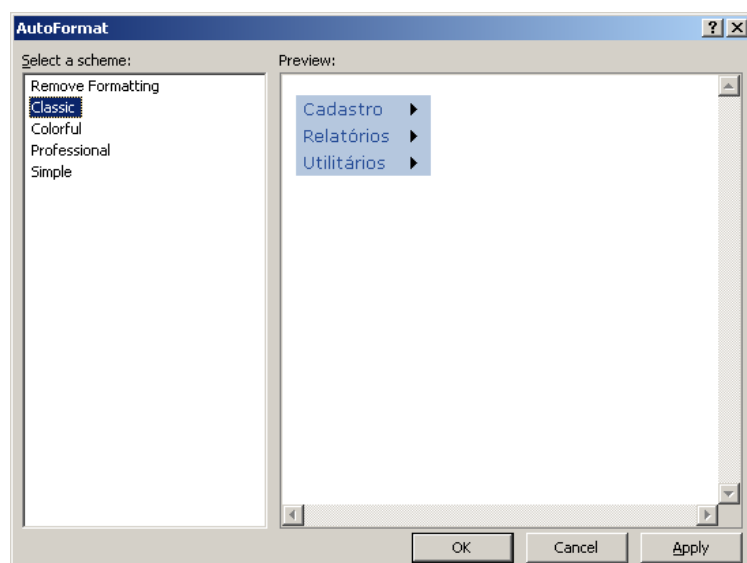


Figura 17

10- Agora acrescente a tag Div antes da tag Menu, conforme listagem parcial do código fonte da página Index provendo a configuração que colocará o objeto Menu centralizado na página.

```
<div align="center"
  style="border: 1px solid #808080;
  width: 800px;
  height:500px">

<asp:Menu ID="MenuSistema" runat="server" BackColor="#B5C7DE"
  DynamicHorizontalOffset="2" Font-Names="Verdana" Font-Size="0.8em"
  ForeColor="#284E98" Orientation="Horizontal" StaticSubMenuIndent="10px"
  style="margin-left: 0px" Width="800px">
  <StaticSelectedStyle BackColor="#507CD1" />
  <StaticMenuItemStyle HorizontalPadding="5px" VerticalPadding="2px" />
  <DynamicHoverStyle BackColor="#284E98" ForeColor="White" />
  <DynamicMenuStyle BackColor="#B5C7DE" />
  <DynamicSelectedStyle BackColor="#507CD1" />
  <DynamicMenuItemStyle HorizontalPadding="5px" VerticalPadding="2px" />
  <StaticHoverStyle BackColor="#284E98" ForeColor="White" />
  <Items>
    <asp:MenuItem Text="Cadastro" Value="Cadastro">
      <asp:MenuItem Text="Produtos" Value="Produtos"></asp:MenuItem>
      <asp:MenuItem Text="Pedidos" Value="Pedidos"></asp:MenuItem>
      <asp:MenuItem Text="Clientes" Value="Clientes"></asp:MenuItem>
    </asp:MenuItem>
    <asp:MenuItem Text="Relatórios" Value="Relatórios">
      <asp:MenuItem Text="Lista de Produtos" Value="Lista de Produtos">
        </asp:MenuItem>
      <asp:MenuItem Text="Movimentos" Value="Movimentos">
        </asp:MenuItem>
    </asp:MenuItem>
    <asp:MenuItem Text="Utilitários" Value="Utilitários">
      <asp:MenuItem Text="Envio de Email" Value="Envio de Email">
        </asp:MenuItem>
    </asp:MenuItem>
  </Items>
</asp:Menu>
```

11- Execute o projeto neste momento, acionando o menu Debug → Start Debugging, observando seu funcionamento conforme imagem da figura 18.



Figura 18

12- Conforme já orientado em passos anteriores, adicione mais uma página aspx, tendo como Template a página Master Page denominada PaginaPadrao. Se orientando pelo design representado pela figura 19 e pela listagem abaixo aplique e configure propriedades para os controles definidos na listagem do código fonte da página CadastroProduto.aspx.

Figura 19

```
<%@ Page Language="C#" MasterPageFile="~/PaginaPadrao.master" AutoEventWireup="true"
CodeFile="CadastroProduto.aspx.cs" Inherits="CadastroProduto" Title="Untitled Page" %>
<asp:Content ID="Content1" ContentPlaceHolderID="head" Runat="Server"> </asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" Runat="Server">
```

```
<div align="center" style="border: 1px solid #808080; width: 800px; height:470px">
  <asp:Label ID="LabelTituloPagina" runat="server" Text="CADASTRO DE PRODUTOS"
    style="z-index: 1; left: 0px; top: 80px; position: absolute; width: 800px; text-align: center"
  </asp:Label>
  <asp:Label ID="LabelDescricao" runat="server" Text="Descrição do Produto"
    style="z-index: 1; left: 50px; top: 180px; position: absolute; width: 200px; text-align: left"
  </asp:Label>
  <asp:TextBox ID="TextBoxDescricao" runat="server"
    style="z-index: 1; left: 50px; top: 200px; position: absolute; width: 350px;"
  </asp:TextBox>
  <asp:Label ID="LabelPreco" runat="server" Text="Preço do Produto"
    style="z-index: 1; left: 50px; top: 260px; position: absolute; width: 200px; text-align: left"
  </asp:Label>
  <asp:TextBox ID="TextBoxPreco" runat="server"
    style="z-index: 1; left: 50px; top: 280px; position: absolute; width: 150px;"
  </asp:TextBox>
  <asp:Button ID="ButtonCancelar" runat="server"
    style="z-index: 1; left: 50px; top: 370px; position: absolute; width: 100; height: 25px"
    Text="Cancelar" />
  <asp:Button ID="ButtonConfirmar" runat="server"
    style="z-index: 1; left: 150px; top: 370px; position: absolute; width: 100; height: 25px"
    Text="Confirmar" />
  <asp:Label ID="LabelErro" runat="server"
    style="z-index: 1; left: 0px; top: 530px; position: absolute; width: 800px; text-align: center" >
  </asp:Label>
</div>

</asp:Content>
```

Nota: Vale ressaltar a tag Div, novamente aqui sendo configurada para centralizar os controles nela acomodada. Note também a propriedade position: absolute definida como propriedade de style para todos os componentes internos a Tag Div. Isso proporciona que todos os controles obedecerão suas posições definidas pelas propriedades top e left, mas com relação a margem esquerda da Tag Div e não do browser que hora irá exibi-los.

13- Crie agora respectivamente as páginas ConsultaProduto.aspx e ConsultaMovimento.aspx. Tomo por base as imagens das figuras 20 e 21. Os detalhes sobre a configuração dos controles destas páginas podem ser observados nas listagens de código fonte que seguem as imagens. As notas referentes a página aspx anterior valem para essas.

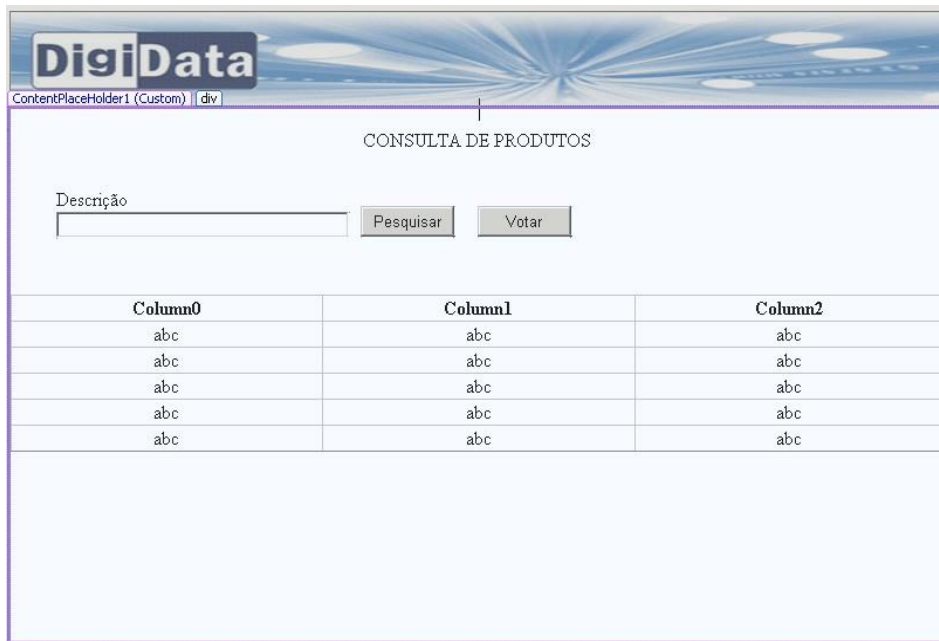


Figura 20

```
<div align="center" style="border: 1px solid #808080; width: 800px; height:450px">
  <asp:Label ID="LabelTituloPagina" runat="server" Text="CONSULTA DE MOVIMENTOS"
    style="z-index: 1; left: 0px; top: 80px; position: absolute;
    width: 800px; text-align: center" >
  </asp:Label>

  <asp:Label ID="LabelDescricaoProduto" runat="server" Text="Descrição"
    style="z-index: 1; left: 20px; top: 150px; position: absolute; width: 70px;" >
  </asp:Label>

  <asp:Label ID="LabelPeriodo" runat="server" Text="Período"
    style="z-index: 1; left: 300px; top: 150px; position: absolute" >
  </asp:Label>

  <asp:Label ID="LabelEntreData" runat="server" Text="À"
    style="z-index: 1; left: 410px; top: 170px; position: absolute; width: 11px;" >
  </asp:Label>

  <asp:TextBox ID="TextBoxDescricaoProduto" runat="server"
    style="z-index: 1; left: 20px; top: 170px; position: absolute; width: 250px;" >
  </asp:TextBox>

  <asp:TextBox ID="TextBoxDataIni" runat="server"
    style="z-index: 1; left: 300px; top: 170px; position: absolute; width: 100px;" >
  </asp:TextBox>
```

```
<asp:TextBox ID="TextBoxDataFin" runat="server"
    style="z-index: 1; left: 430px; top: 170px; position: absolute; width: 100px;">
</asp:TextBox>
```

```
<asp:Button ID="ButtonPesquisar" runat="server"
    style="z-index: 1; left: 560px; top: 165px; position: absolute; width: 75px; height: 25px"
    Text="Pesquisar" CssClass="Botao" />
```

```
<asp:Button ID="ButtonVoltar" runat="server" style="z-index: 1; left: 640px; top: 165px;
position: absolute; width: 75px; height: 25px" Text="Votar" />
```

```
<asp:Button ID="ButtonImprimir" runat="server"
    style="z-index: 1; left: 720px; top: 165px; position: absolute; width: 75px; height: 25px"
    Text="Imprimir" />
```

```
<asp:GridView ID="GridViewListaMovimento" runat="server" style="z-index: 1; left: 0px;
    top: 220px; position: absolute; height: 90px; width: 800px" >
```

```
<Columns>
```

```
<asp:BoundField DataField="Id" HeaderText="Id" SortExpression="Id" />
```

```
<asp:BoundField DataField="ProdutoId" HeaderText="ProdutoId"
    SortExpression="ProdutoId" />
```

```
<asp:BoundField DataField="Tipo_movimentoid" HeaderText="Tipo_movimentoid"
    SortExpression="Tipo_movimentoid" />
```

```
<asp:BoundField DataField="Data" HeaderText="Data" SortExpression="Data" />
```

```
<asp:BoundField DataField="Quantidade" HeaderText="Quantidade"
    SortExpression="Quantidade" />
```

```
<asp:BoundField DataField="Valor" HeaderText="Valor" SortExpression="Valor" />
```

```
<asp:BoundField DataField="Descricao_Movimento"
    HeaderText="Descricao_Movimento" SortExpression="Descricao_Movimento" />
```

```
<asp:BoundField DataField="Descricao_Produto" HeaderText="Descricao_Produto"
    SortExpression="Descricao_Produto" />
```

```
</Columns>
```

```
</asp:GridView>
```

```
</div>
```

```
</asp:Content>
```

The screenshot shows a web application interface with a header 'DigiData' and a title 'CONSULTA DE MOVIMENTOS'. Below the title, there is a search form with three input fields: 'Descrição', 'Período', and 'À', followed by 'Pesquisar' and 'Votar' buttons. Below the form is a table with three columns: 'Column0', 'Column1', and 'Column2'. The table contains five rows of data, each with the value 'abc' in all three columns.

Column0	Column1	Column2
abc	abc	abc
abc	abc	abc
abc	abc	abc
abc	abc	abc
abc	abc	abc

Figura 21

```
<div align="center" style="border: 1px solid #808080; width: 800px; height:450px">
```

```
<asp:Label ID="LabelTituloPagina" runat="server" Text="CONSULTA DE MOVIMENTOS"
    style="z-index: 1; left: 0px; top: 100px; position: absolute;
    width: 800px; text-align: center">
</asp:Label>
<asp:Label ID="LabelDescricaoProduto" runat="server" Text="Descrição"
    style="z-index: 1; left: 40px; top: 150px; position: absolute">
</asp:Label>
<asp:Label ID="LabelPeriodo" runat="server" Text="Período"
    style="z-index: 1; left: 320px; top: 150px; position: absolute">
</asp:Label>
<asp:Label ID="LabelEntreData" runat="server" Text="À"
    style="z-index: 1; left: 430px; top: 170px; position: absolute">
</asp:Label>
<asp:TextBox ID="TextBoxDescricaoProduto" runat="server"
    style="z-index: 1; left: 40px; top: 170px; position: absolute; width: 250px;">
</asp:TextBox>
<asp:TextBox ID="TextBoxDataIni" runat="server"
    style="z-index: 1; left: 320px; top: 170px; position: absolute; width: 100px;">
</asp:TextBox>
<asp:TextBox ID="TextBoxDataFin" runat="server"
    style="z-index: 1; left: 450px; top: 170px; position: absolute; width: 100px;">
</asp:TextBox>
<asp:Button ID="ButtonPesquisar" runat="server"
    style="z-index: 1; left: 580px; top: 165px; position: absolute;
    width: 80px;" Text="Pesquisar" />
<asp:Button ID="ButtonVoltar" runat="server"
    style="z-index: 1; left: 680px; top: 165px; position: absolute;
    width: 80px;" Text="Votar" />
<asp:GridView ID="GridView1" runat="server"
    style="z-index: 1; left: 1px; top: 240px; position: absolute;
    height: 90px; width: 798px">
</asp:GridView>
```

```
</div>
```

14- Neste ponto iremos retornar a página Index, selecionando o Menu Task do controle Menu selecionando a opção Edit Menu Items..., conforme imagem da figura 22.

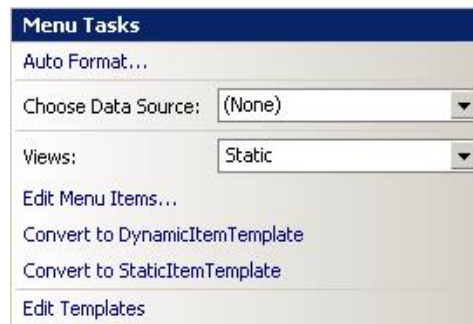


Figura 22

15- Conforme observado na figura 23, selecione o item de menu Produtos definindo sua propriedade NavigateUrl para chamar a página ConsultaProduto.aspx.

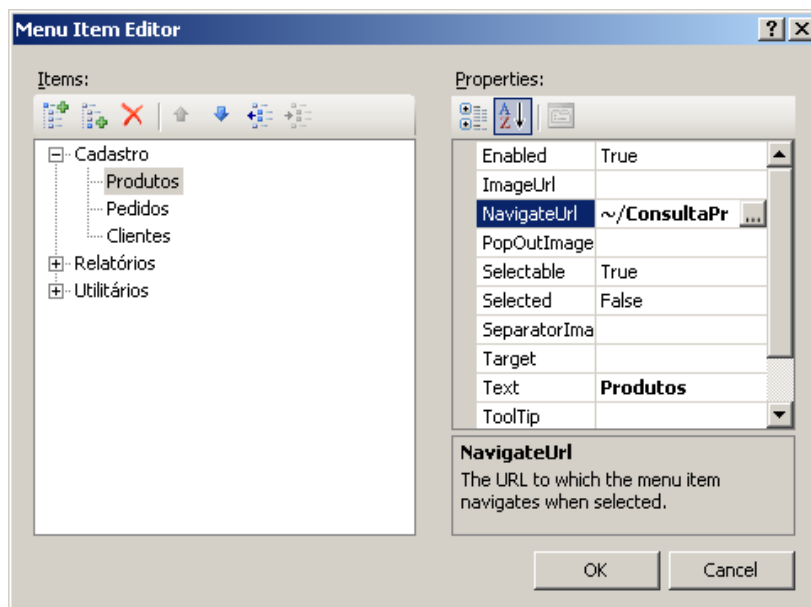


Figura 23

16- Objetivando definir a primeira página a ser exibida quando da execução do projeto, com o botão direito do mouse sobre o diretório raiz do projeto, selecione no menu de contexto a opção Start Options..., e em seguida proceda conforme observado nas figuras 24, 25 e 26.

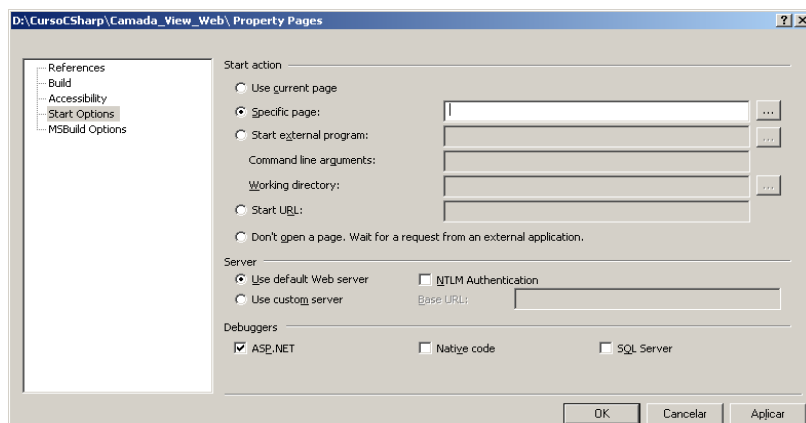


Figura 24

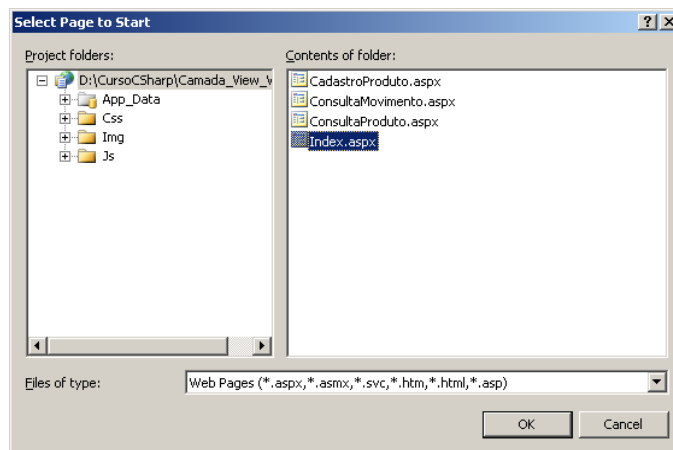


Figura 25

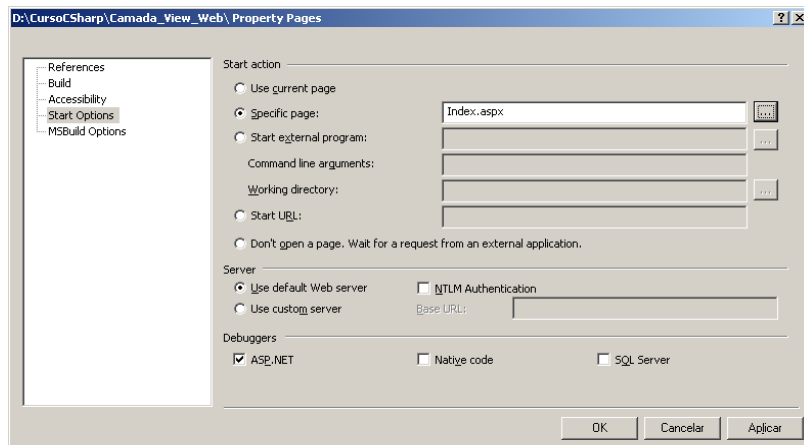


Figura 26

17- Teste a execução da página `ConsultaProduto.aspx`, a partir da execução do menu da página `Index`. Veja a figura 27 que ilustra este tópico.

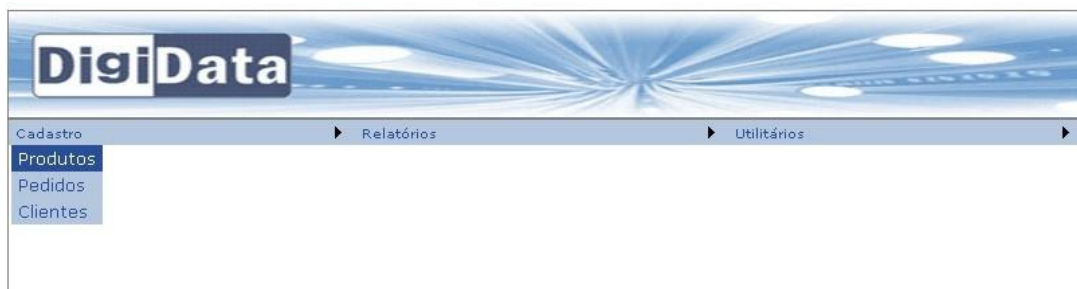


Figura 27

18- O fraguimento de código abaixo é o conteúdo do arquivo de Folha de Estilo denominado `Estilo.css` que servirá para configurar os aspectos visuais dos controles de nossa página. Numa etapa a seguir aplicaremos esses estilos aos controles do projeto.

```
.Botao {  
    font-family: verdana, helvetica;  
    font-size: 11px;  
    font-weight: bold;  
    color: #004080;  
    background-color: #DAD7CF;  
}
```

```
.CaixaTexto {
    font-family: verdana, helvetica;
    font-size: 11px;
    font-weight: normal;
    color: #000000;
    border: 1px solid #000000;
}

.LabelTexto {
    font-family: verdana, helvetica;
    font-size: 11px;
    font-weight: bold;
    color: #004080;
}

.LabelTitulo {
    font-family: verdana, helvetica;
    font-size: 15px;
    font-weight: bold;
    color: #004080;
    background-color: #DAD7CF;
}

.ComboBox {
    font-family: verdana, helvetica;
    font-size: 11px;
    color: #000000;
    background-color: #ffffff;
    border: 1px solid #000000;
}

.TextoCelulaGrid {
    font-family: verdana, helvetica;
    font-size: 11px;
    color: #000000;
}

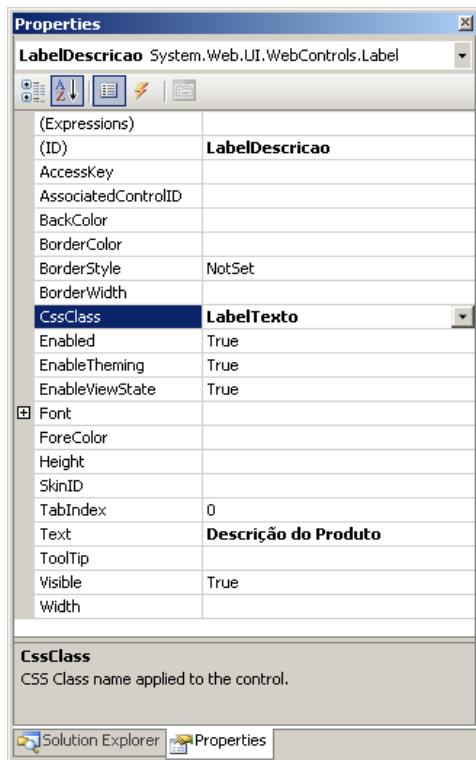
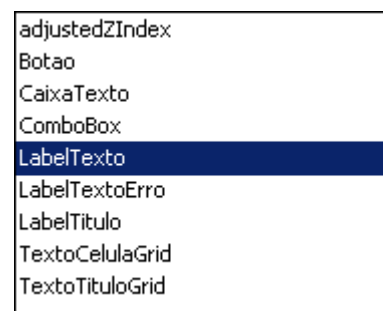
.TextoTituloGrid {
    font-family: verdana, helvetica;
    font-size: 11px;
    font-weight: bold;
    color: #ffffff;
    background-color: #004080;
}
```

19- Acrescente a Tag Link ao código fonte da Master Page, para que os controles das páginas derivadas possam ser configurados com as Classes da folha de estilo. Esta tag deve ser aplicada dentro da tag head.

```
<link href="\css\estilo.css" type="text/css" rel="stylesheet" />
```

20- Aplique estilo aos controles Label, TextBox e Button da página CadastroProduto, em modo Design, procedendo da seguinte forma:

20.1 – Selecione o label com a propriedade text definida como Descrição do Produto, e em sua propriedade (precione F4 caso a caixa de dialogo Properties não esteja sendo exibida) CssClass escolhendo a Classe LabelText conforme ilustrado nas figuras 28 e 29..

**Figura 28****Figura 29**

20.2 – Aplique o mesmo para os controles **TextBox** e **Button**, escolhendo respectivamente as Classes **CaixaTexto** e **Botao**. Por fim veja o resultado refletido na página **CadastroProduto.aspx** conforme figura 30.

**Figura 30**

20.3 – Proceda assim para todos os formulário do projeto, e aproveite para alterar algumas Classes da Folha de Estilo para ver o efeito.

21- Como ajuste final, visando corrigir incompatibilidade em algumas funcionalidades em alguns browsers, aplique os trechos de código abaixo conforme orientação.

21.1 – A tag abaixo deve ser aplicada dentro da Tag Head da Master Page.

```
<style type="text/css">
    .adjustedZIndex{z-index: 1; }
</style>
```

21.2 – Deve ser aplicado como propriedade de Style do objeto Menu na página Master Page.

```
DynamicMenuStyle BackColor="#B5C7DE" CssClass="adjustedZIndex"/>
```

21.3 – Aplicado ao evento Page_PreInit da Master Page.

```
protected void Page_PreInit(object sender, EventArgs e)
{
    if (Page.Request.ServerVariables["http_user_agent"].ToLower().Contains("safari"))
    {
        Page.ClientTarget = "uplevel";
    }
}
```

22- Adicione o fraguimento de código descrito abaixo dentro da Tag Head da Master Page, para podermos utilizar funções de JavaScript em nossas páginas. Veja no **Anexo 1** desta apostila uma abordagem mais profunda sobre JavaScript e sua aplicação em páginas aspx.

```
<script
    language="javascript"
    type="text/javascript"
    src="JS/FuncoesJavaScript.js">
</script>
```

22- Aplicando funções JavaScript a controles do projeto

22.1- Primeiro iremos criar um arquivo contendo funções JavaScript (ver listagem abaixo), na sequência referenciá-lo em nossas páginas ou na pagina Mestre do projeto. Desta forma, poderemos fazer uso das referidas funções em eventos JavaScript dos controles Asp.Net contidos em nossas páginas. Crie um arquivo com formato texto padrão, nomeando-o como FuncoesJavaScript.JS, salvando-o no diretório JS do projeto web. O código do conteúdo do arquivo em questão é exibido na listagem 01.

```
function FormataData(input)
{
    if (input.value.length==10)
    {
        event.keyCode=0;
    }

    var tecla = event.keyCode;
    if (tecla > 47 && tecla < 58){

        input.value=input.value;
    }
    else
    {
        event.keyCode=0;
    }
}
```

```
        if ((input.value.length==2)|| (input.value.length==5))
        {
            input.value=input.value + "/" ;
        }
    }
```

```
function SomenteNumeros(input)
{
    var tecla = event.keyCode;
    if (tecla > 47 && tecla < 58){

        input.value=input.value;
    }
    else
    {
        event.keyCode=0;
    }
}
```

```
function Formatar(src, mask)
{
    var i = src.value.length;
    var saida = mask.substring(0,1);
    var texto = mask.substring(i)
    if (texto.substring(0,1) != saida)
    {
        src.value += texto.substring(0,1);
    }
}
```

```
function FormatarCnpj(src, mask)
{
    if (src.value.length==18)
    {
        event.keyCode=0;
    }
    var tecla = event.keyCode;
    if (tecla > 47 && tecla < 58){
        src.value=src.value;
    }
    else
    {
        event.keyCode=0;
    }
    var i = src.value.length;
    var saida = mask.substring(0,1);
    var texto = mask.substring(i)
    if (texto.substring(0,1) != saida)
    {
        src.value += texto.substring(0,1);
    }
}
```

22.2- Outra maneira de injetar referência a métodos de JavaScript a controles de Asp.Net, é utilizando a propriedade `Attributes` de um controle, usando o método `Add` do mesmo para acrescentar chamada a uma função JavaScript. No exemplo abaixo, será acrescentado ao controle no lado Client-Side o uso da função `FormataData` quando ocorrer o evento `onkeypress` de java excript para os controles `TextBoxDataIni` e `TextBoxDataFin`. O evento da página ideal para codificar o proposto no parágrafo anterior é `Page_Load`, pois assim, ao carregar da página o fraguimento de código é renderizado no lado Client-Side (Browser).

```
protected void Page_Load(object sender, EventArgs e)
```

```
{
    TextBoxDataIni.Attributes.Add("onkeypress", "return FormataData(this);");
    TextBoxDataFin.Attributes.Add("onkeypress", "return FormataData(this);");
}
```

22.3- No formulário (ou página se preferir) `Index.aspx`, acione o Menu Task do controle Menu conforme Figura 31 clicando em `Edit Menu Items`. Emseguida selecione o item `Movimentos` atribuindo a propriedade `NavigateUrl` o formulário `ConsultaMovimento.aspx`. As Figuras 32 e 33 ilustram estes passos.

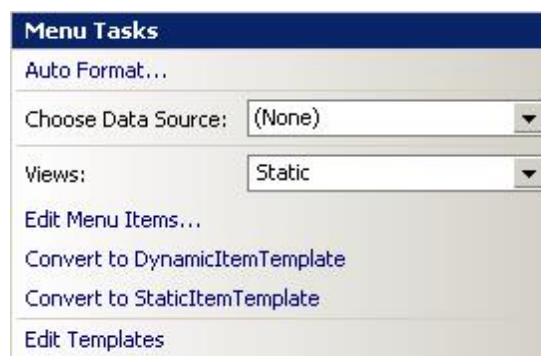


Figura 31

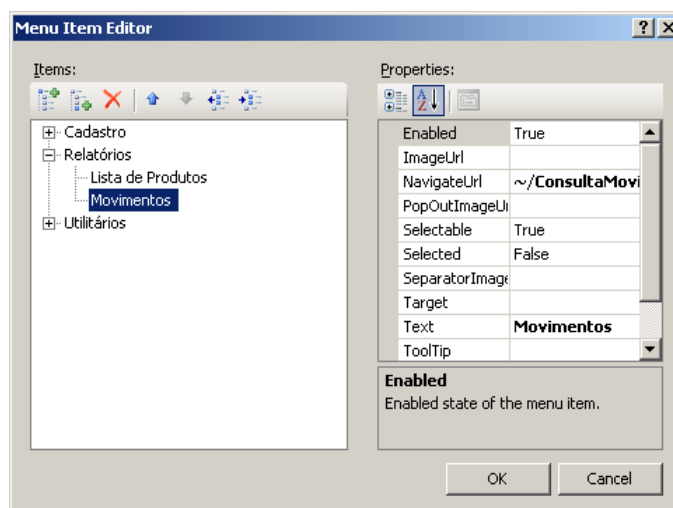


Figura 32

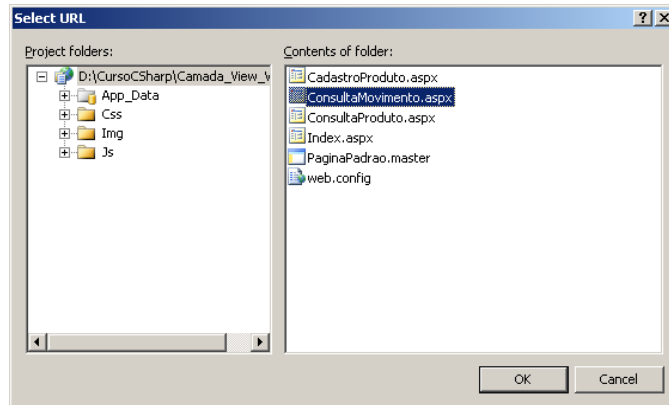


Figura 33

22.4- Execute o site, testando a função JavaScript conforme ilustrado na Figura 34 representando o formulário ConsultaMovimento.Aspx em execução.



Figura 34

22.5- Por fim, estude também o conteúdo do **Anexo 2** desta apostila, que trata de uma outra forma de formatar controles de Aspx com a adoção de **Expressões Regulares**.

Criando a Camada Controller

Comentando os passos:

1- Precisamos neste momento adicionar uma classe ao projeto Web, para que implemente o código necessário ao acesso aos métodos da camada de negócio (Model). Clique com o botão direito do mouse sobre o nó que representa o projeto em Solution Explorer, selecione Add New Item..., tendo como resposta o exposto na figura 01. Selecione o Template Class, defina o nome como ProdutoController e clique no botão Add para finalizar

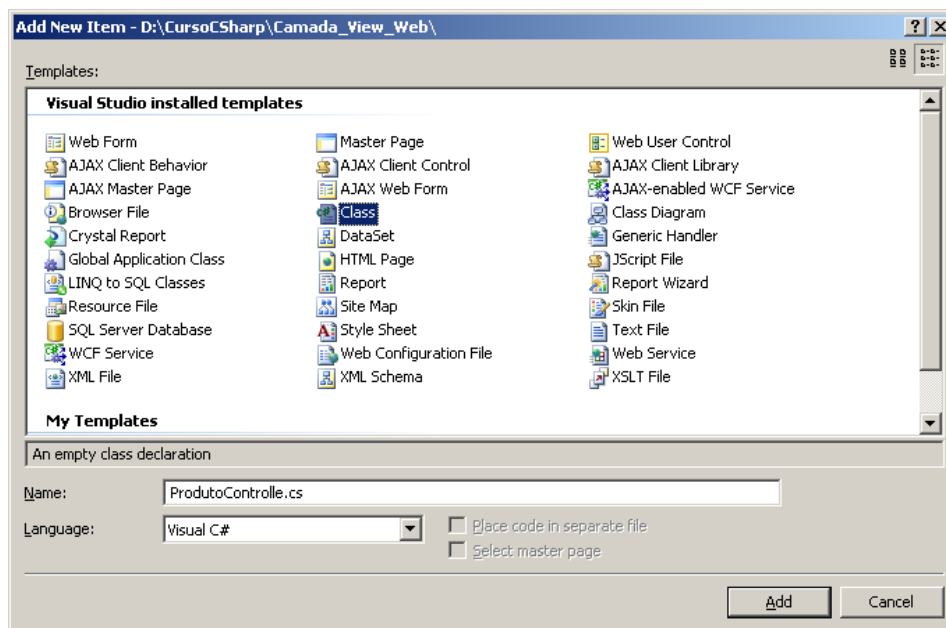


Figura 01

2- Na sequência responda Sim para a caixa de diálogo representada pela figura 02, autorizando acomodar a classe recém criada no diretório App_code do projeto Web.

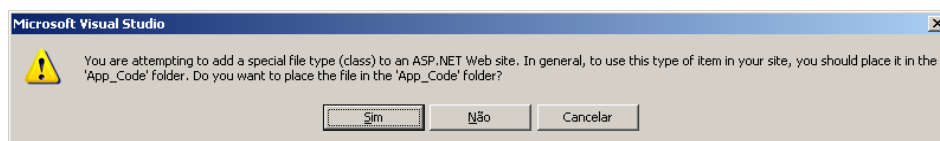


Figura 02

3- Por fim, repare a figura 03 exibindo a classe no diretório App_Code.

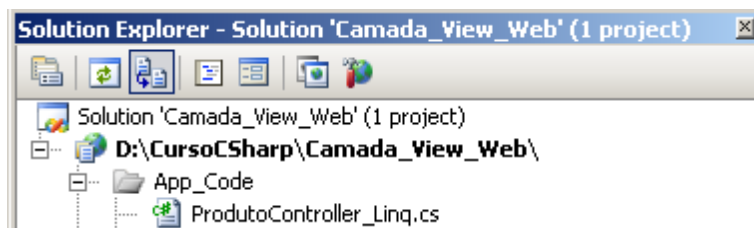


Figura 03

4- Adicionemos agora referência para para os assemblies que representam a Camada de Negócio (Model), objetivando assim, permitir a Classe ProdutoController ter acesso aos métodos de negócio.

4.1- Clicando com o botão direito do mouse sob o nó do projeto (conforme figura 04), escolha Add Reference no menu de contexto..

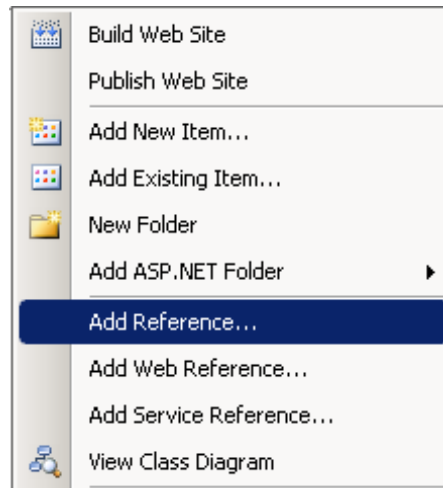


Figura 04

4.2- Após aguardar uns instantes, a imagem da figura 05 será exibida, permitindo que pela caixa de diálogo Add Reference possamos escolher a biblioteca (aplicativo) NameSpaceSQL.dll, completando assim o processo de referência a camada de negócio.

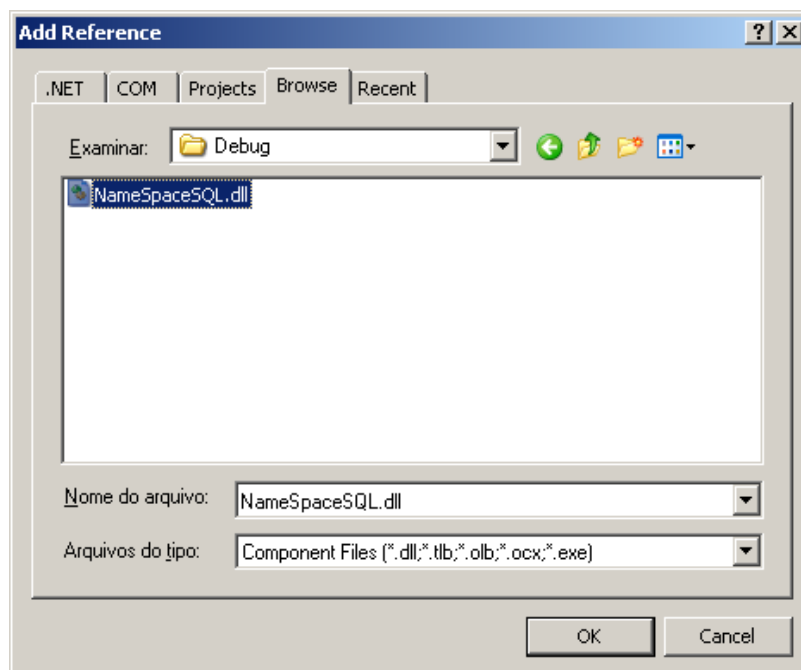


Figura 05

5- Declarações (referências as NameSpaces necessárias ao código fonte do projeto.)

```
using System;  
using System.Data;  
using System.Configuration;  
using System.Linq;  
using System.Web;  
using System.Web.Security;  
using System.Web.UI;  
using System.Web.UI.HtmlControls;
```

```
using System.Web.UI.WebControls;  
using System.Web.UI.WebControls.WebParts;  
using System.Xml.Linq;  
using System.Data;  
using System.Collections.Generic;  
using System.Collections.ObjectModel;
```

5- Definição do nome da classe, do escopo (Pública ou Privada) da classe e do bloco inicial.

```
public class ProdutoController_SQL  
{
```

6- Instâncias para as classes Produto, Movimento e ListaMovimento contidas na NameSpaceSQL previamente referenciada.

```
NameSpaceSQL.Produto Produto = new NameSpaceSQL.Produto();  
NameSpaceSQL.Movimento Movimento = new NameSpaceSQL.Movimento();  
NameSpaceSQL.ListaMovimento ListaMovimento = new NameSpaceSQL.ListaMovimento();
```

7- Definição dos campos públicos que armazenarão os valores relativo aos campos do modelo Produto. Serão úteis para armazenar os valores referentes a um Produto quando obtido, e nas operações de Update, a série Old dos mesmos, poderão reter os valores obtidos, distinguindo-os dos valores alterados pelo processo.

```
public long Id;  
public string Descricao;  
public decimal Preco;  
public long OldId;  
public string OldDescricao;  
public decimal OldPreco;
```

8- Método que retorna um DataTable (collection) contendo um registro da tabela Produto baseado na chave primária passada como parâmetro. Também preenche os campos públicos que representam o registro produto.

```
public DataTable GetProduto(long Id)  
{  
    DataTable dt = new DataTable();  
    dt = Produto.RetornaProduto(Id);  
    this.Id = (long)dt.Rows[0].ItemArray[0];  
    this.Descricao = (string)dt.Rows[0].ItemArray[1];  
    this.Preco = (decimal)dt.Rows[0].ItemArray[2];  
    return dt;  
}
```

9- Método que recebe como parâmetro um tipo DataTable (em geral um registro de Produto), armazenando os valores em variáveis que representarão no processo de Update os valores antigos, aquele úteis para cláusula Where da instrução Update subjacente.

```
public void SetOldProduto(DataTable OldDT)  
{  
    this.OldId = (long)OldDT.Rows[0].ItemArray[0];  
    this.OldDescricao = (string)OldDT.Rows[0].ItemArray[1];  
    this.OldPreco = (decimal)OldDT.Rows[0].ItemArray[2];  
}
```

10- Método que encapsula o método RetornaProduto da Classe Produto, sobrecarregado pelo parâmetro string, que retorna um DataTable contendo uma lista de registros da tabela Produto

```
public DataTable ObterProduto(string Descricao)
{
    return Produto.RetornaProduto(Descricao);
}
```

11- O mesmo que o item anterior, mas para o método RetornaMovimento.

```
public DataTable ObterListaMovimento(string Descricao, DateTime DataIni, DateTime Datafin)
{
    return Produto.RetornaMovimento(Descricao, DataIni, Datafin);
}
```

12- Método que encapsula o método de inserção de Produto e Movimento, implementado na Classe Produto, retornando string.

12.1 – Define: O Nome do método, seus parâmetros de entrada e o bloco inicial do método.

```
public string InserirProdutoMovimento(string Descricao, decimal Preco)
{
```

12.2 – Preenche o campo Id da instância de Produto com o valor retornado pelo método gerador de chave única da Classe Produto.

```
    Produto.Id = Produto.RetornaChave("ID_PRODUTO");
```

12.3 – Preenche os campos Descricao e Preco, da instância de Produto com os valores obtidos pelos parâmetros do método InserirProdutoMovimento.

```
    Produto.Descricao = Descricao;
    Produto.Preco = Preco;
```

12.4 – Idêntico ao item 12.3, mas para o campo Id da instância de Movimento.

```
    Movimento.Id = Produto.RetornaChave("ID_MOVIMENTO");
```

12.5 – Preenche o restante dos campos da instância de Movimento. São valores arbitrados para conclusão do exemplo.

```
    Movimento.ProdutoId = Produto.Id;
    Movimento.Tipo_movimentoid = 1;
    Movimento.Quantidade = 0;
    Movimento.Valor = 0;
    Movimento.Data = DateTime.Now.Date;
    Produto.Movimento.Add(Movimento);
```

13.6 – Finalizado a lógica do método, invoca a execução do método de negócio InserirProdutoMovimento, que retorna uma string contendo a mensagem de erro caso ocorram. Por fim encerra o bloco de código do método.

```
    return Produto.InserirProdutoMovimento();
}
```


15- Método que encapsula a chamada ao método `AlterarProduto` da Classe `Produto`. Repare os parâmetros `OldId`, `OldDescricao` e `OldPreco`, que servem para serem referenciados na cláusula `Where` da instrução `Update`. Assim, temos garantia de execução da instrução `Update` na condição de Concorrência Otimista. Este método também retorna uma string com a mensagem de erro, retornado pelo método subjacente caso exista.

```
public string AlterarProduto( string Descricao,
                             decimal Preco,
                             long OldId,
                             string OldDescricao,
                             decimal OldPreco
                             )
{
    Produto.Descricao = Descricao;
    Produto.Preco = Preco;
    Produto.OldId = OldId;
    Produto.OldDescricao = OldDescricao;
    Produto.OldPreco = OldPreco;
    return Produto.AlterarProduto();
}
```

16- O mesmo que o item anterior sendo que para a exclusão de `Produto`.

```
public string ExcluirProduto( long OldId,
                             string OldDescricao,
                             decimal OldPreco
                             )
{
    bool Sucesso = false;
    Produto.OldId = OldId;
    Produto.OldDescricao = OldDescricao;
    Produto.OldPreco = OldPreco;
    return Produto.ExcluirProduto();
}
```

17- Encerra o bloco de código da Classe `ProdutoController`.

```
}
```

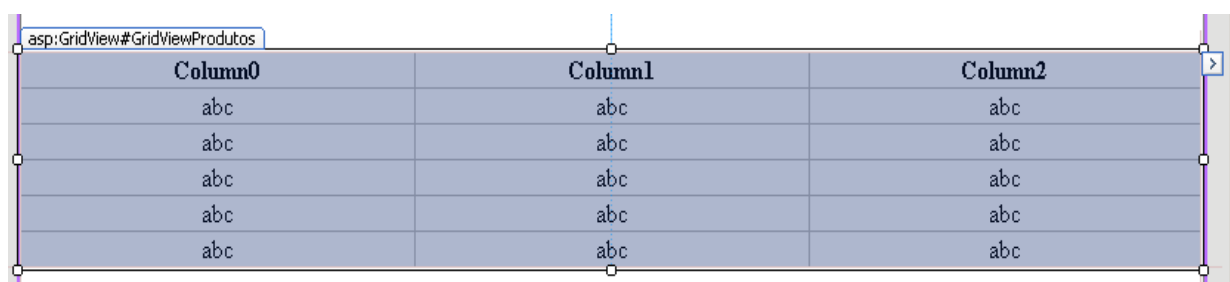
Vinculando a Camada View com a Camada Controller

Configurando o Controle GridView da Página ConsultaProduto

Responsável por exibir o resultado de consultas, neste caso, a consulta de Produtos retornada pelo método ObterProduto, que retorna para a propriedade DataSource do GridView um conjunto de dados na forma de um DataTable, veremos na sequência os passos para ainda em tempo de projeto configurar as colunas que serão exibidas.

1- Selecione o GridView (figura 01) clicando na seta (Menu Task) localizada na canto superior esquerdo do controle. Selecione no combo da opção Choose Data Source (figura 02), New Data Source para vincularmos o GridView a um método que possa retornar a estrutura de uma consulta.

Um Datasource é um objeto que, uma vez vinculado a um método, pode ser usado para acesso a estrutura de retorno do método, como em tempo de execução acessar o próprio método.



Column0	Column1	Column2
abc	abc	abc
abc	abc	abc
abc	abc	abc
abc	abc	abc
abc	abc	abc

Figura 01

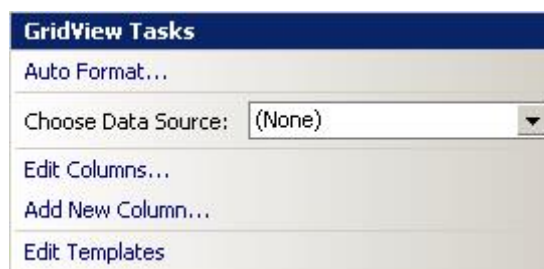


Figura 02

2- Cumprido o passo anterior, a caixa de dialogo Data Source Configuration Wizard, representada pela imagem da figura 03 será exibida, para a qual faremos opção para o objeto Object.

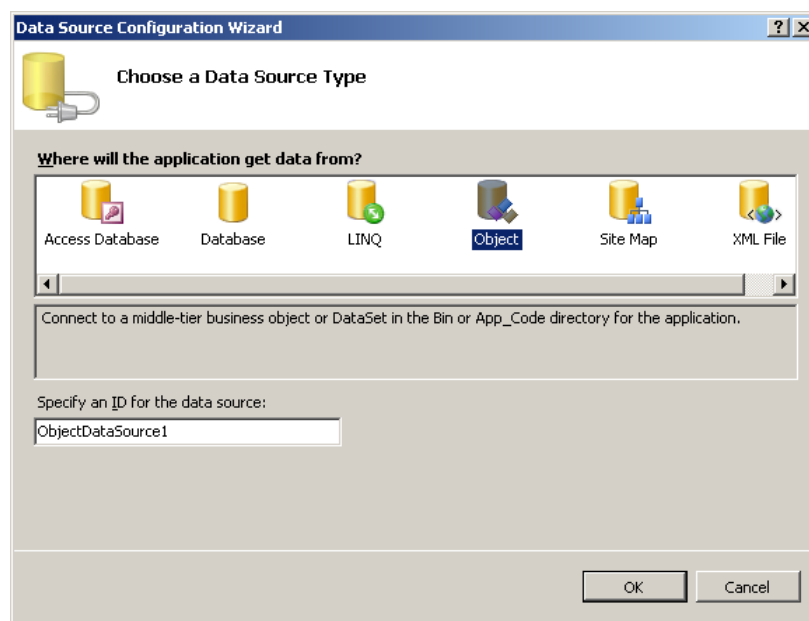


Figura 03

3- Clicando em OK na opção anterior somos conduzidos para caixa de dialogo Configure Data Source – ObjectDataSource1. Na opção Choose your business object, selecione a classe ProdutoController_SQL clicando no botão Next para prosseguir na configuração do objeto.

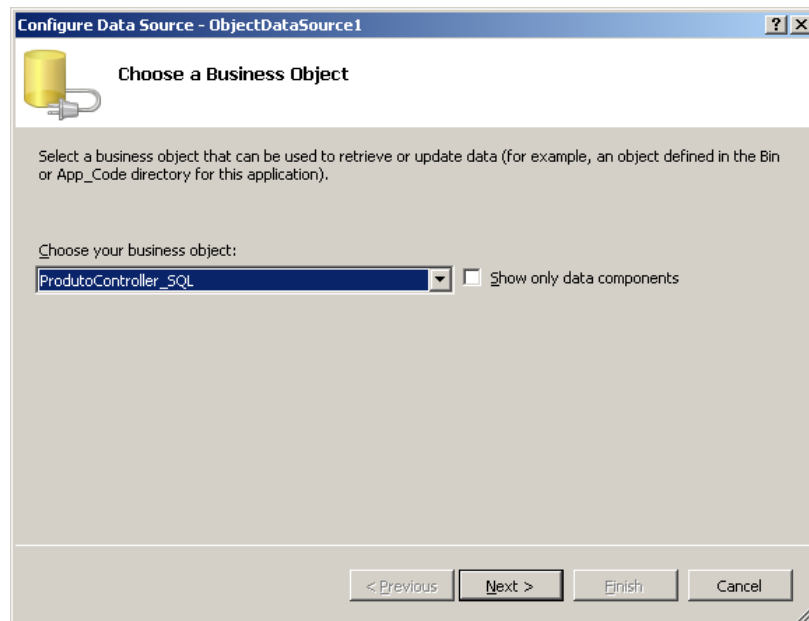


Figura 04

4- Nesta etapa, na aba SELECT, escolha no combo Choose a method o método ModeloProduto conforme ilustrado pela figura 05. Clique no botão Next, prosseguindo na configuração.

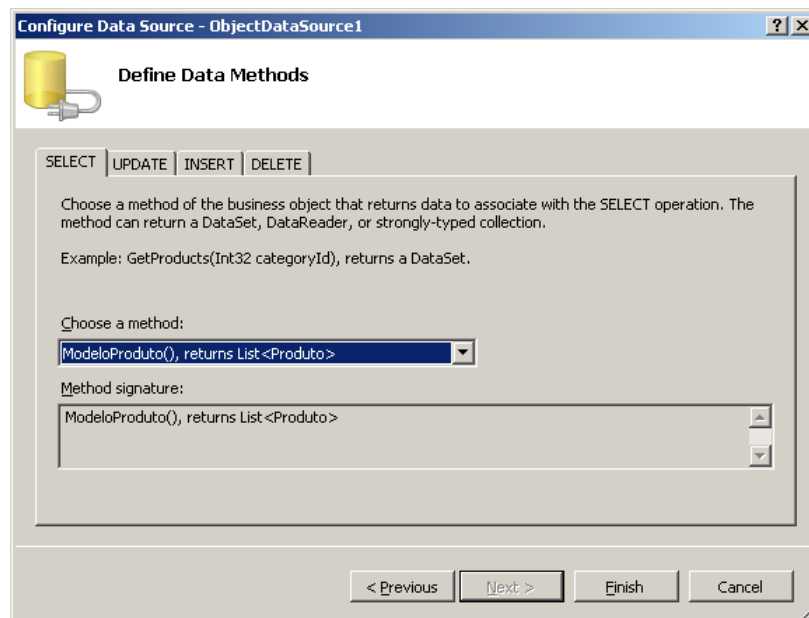


Figura 05

5- Passado a passo anterior, será exibido a imagem da figura 06 janela GridView Tasks (Menu Task do controle) tendo agora na propriedade Choose Data Source uma referência a ObjectDataSource1, que é o DataSource que definirá, em tempo de projeto, as colunas do GridView. Confira a Figura 07 para visualizar o resultado no GridView.

Repare que a estrutura de colunas do GridView (figura 07) passou refletir naturalmente a estrutura de campos da Classe Produto. Devemos agora ajustar para nossa realidade, excluindo colunas desnecessárias.

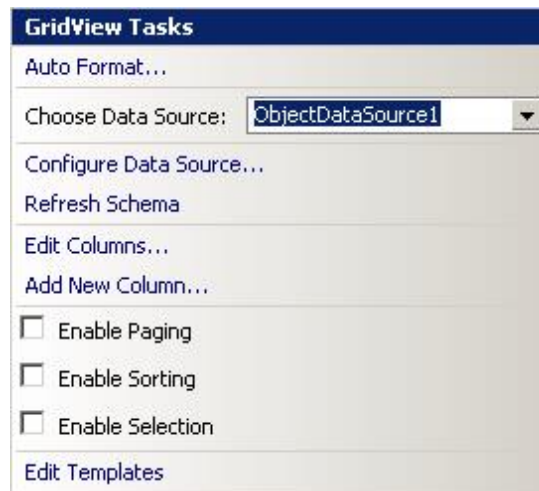


Figura 06

Id	Descricao	Preco	OldId	OldDescricao	OldPreco
0	abc	0	0	abc	0
1	abc	0,1	1	abc	0,1
2	abc	0,2	2	abc	0,2
3	abc	0,3	3	abc	0,3
4	abc	0,4	4	abc	0,4

Figura 07

6- Se orientando pela figura 06, clique em Edit Columns... para poder manipular e configurar as colunas do GridView. Conforme exibido na figura 08, selecione os campos OldId, OldDescricao e OldPreco, individualmente, clicando no botão de exclusão. Estes campos estão localizados na lista Selected Fields.

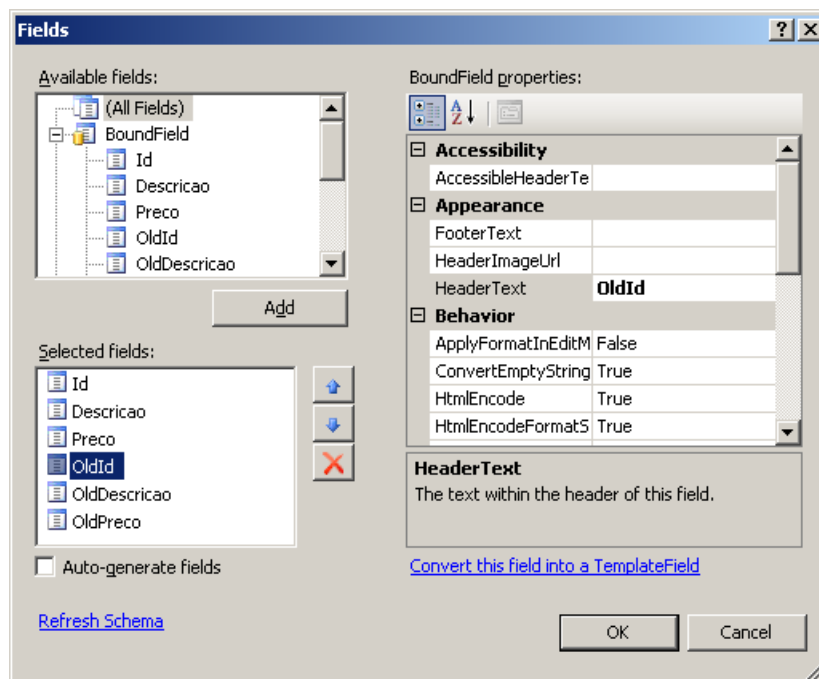


Figura 08

7- Ainda na caixa de dialogo Fields, selecione em Selectd Fields a coluna Id manipulando suas propriedades conforme sequência.

7.1 – Propriedade HeaderText igual a Id. Esta propriedade define o título da coluna no grid.

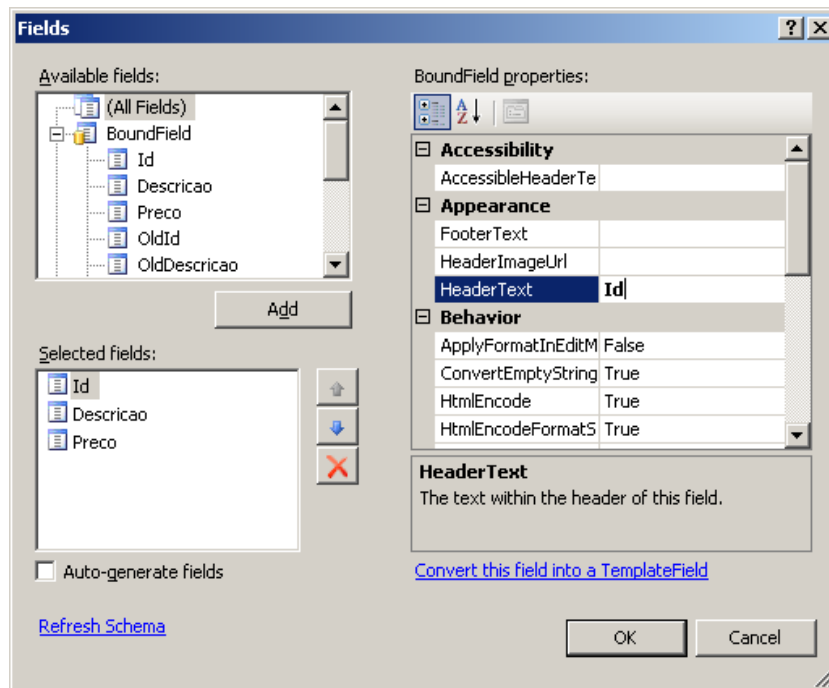


Figura 09

7.2 – Propriedade DataField igual a Id. O valor atribuido a esta propriedade deve ser igual ao nome de um campo retornado pelo DataTable (collection) para que ocorra a sincronia, logo, representar uma coluna da coleção resultante do método subjacente.

Repare a figura 11, exibida quando houve a seleção, que representa a estrutura resultante do método retornado pelo ObjectDataSource ao qual o GridView está selecionado. Na prática não necessitamos de um ObjectDataSource vinculado a um método em tempo de projeto. Bastaria irmos adicionando colunas ao GirdView, e em seguida, configurar a propriedade DataField com nomes de colunas que antecipadamente sabemos que retornará do método que será matribuido a propriedade DataSource do GridView.

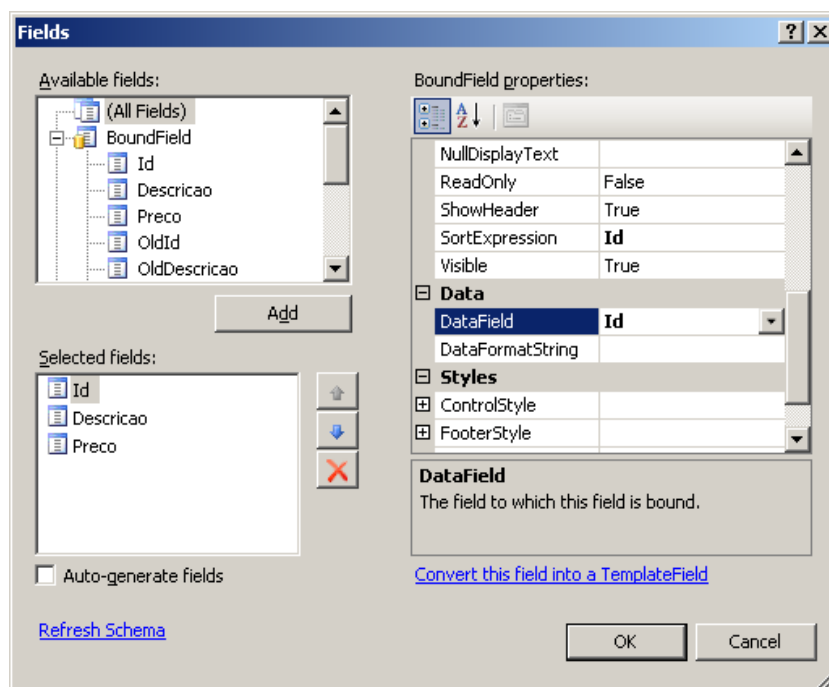


Figura 10

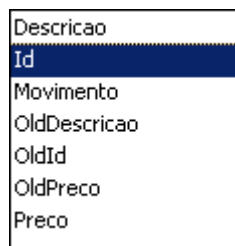


Figura 11

7.3 – Por fim, poderíamos manipular as propriedades HeaderStyle e ItemStyle, definindo a aparência final dos títulos das colunas bem como das colunas, ou melhor, das células das colunas. Não faremos isso, optaremos aqui por utilizar a propriedade CssClass tanto de HeaderStyle quanto ItemStyle

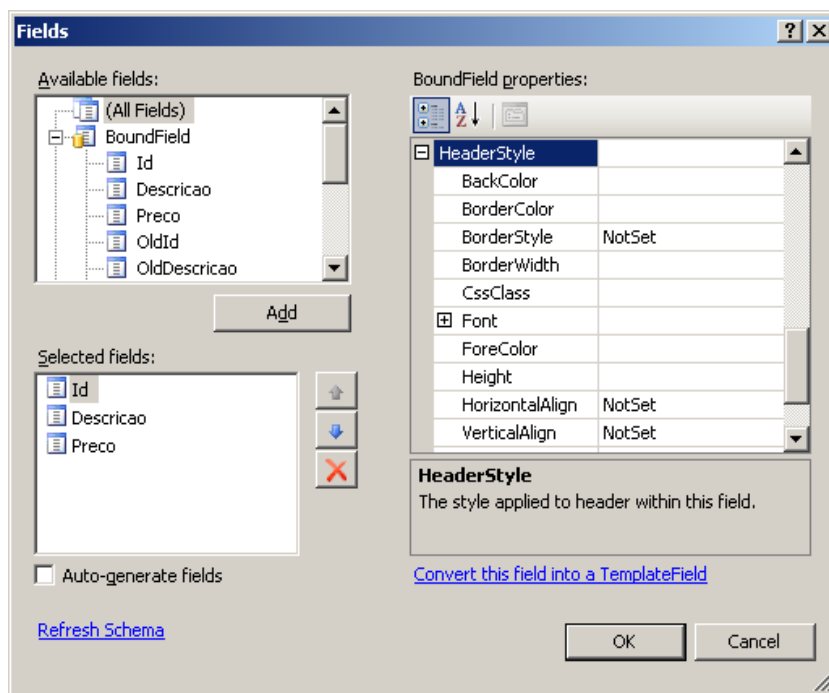


Figura 12

7.4 – Conforme sugerido no item 7.3, aplique a propriedade CssClass de HeaderStyle de todas as colunas, o nome da Classe TextoTituloGrid que consta em nosso arquivo de Folha de Estilo (Estilo.Css). Isto funcionará devido ao fato de nossa Página Padrão, que é a base da página atual, já referenciar o arquivo de Folha de Estili citado. Proceda o mesmo para a propriedade ItemStyle das colunas, atribuindo a propriedade CssClass, a Classe TextoCelulaGrid da mesma Folha de Estilo.

A imagem da figura 13 reflete o efeito das classes de Folha de Estilo aplicado ao GridView.

Id	Descrição	Preço
0	abc	0
1	abc	0,1
2	abc	0,2
3	abc	0,3
4	abc	0,4

Figura 13

7.5 – Como ajustes final, iremos adicionar uma coluna especial, visando criar um Link para que o usuário possa selecionar uma linha no GridView, e por consequência obtermos associado a um evento do mesmo GridView o valor da chave do registro selecionado. Neste caso o valor da célula da priemira coluna para a linha selecionada. Se oriente então pela figura 14, vendo o resultado na imagem da figura 15.

7.5.1 – Conforme figura 14, obtida a partir do clique na opção Edit Columns... do menu GridView Tasks, na lista Avaleables fields, selecione Select no nó CommandField clicando no botão Add.

Em seguida configure a propriedade HeadText da coluna recém criada para vazio, a propriedade ButtonType para Link e a propriedade SelectText como Selecionar. Manipule também a propriedade ItemStyle → Width com o valor de 20% (determina a largura da coluna em termos percentuais com relação a largura do GridView).

Se desejar, aplique classes da Folha de Estilo a gosto. Talvez queira experimentar criar uma classe de Folha de Estilo específica para este tipo de controle (coluna).

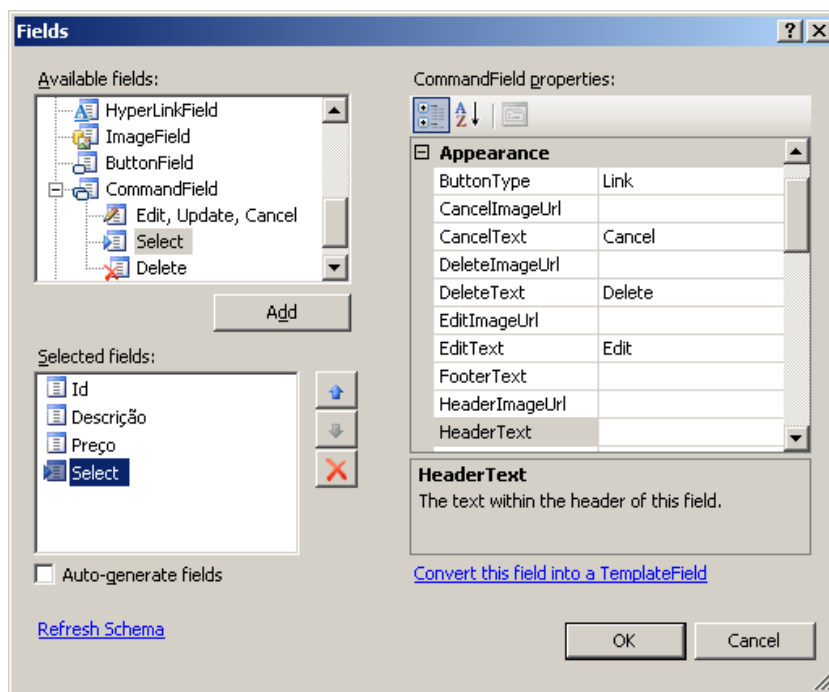


Figura 14

Id	Descrição	Preço	
0	abc	0	Selecionar
1	abc	0,1	Selecionar
2	abc	0,2	Selecionar
3	abc	0,3	Selecionar
4	abc	0,4	Selecionar

Figura 15

7.5.2 – Finalizando, na propriedade DataSourceID, remova a referência ao ObjectDataSource1, que a partir deste momento não mais tem importância, já que como informado, só tem utilidade para configurarmos o GridView em tempo de projeto. Responda "Não" para caixa de diálogo da figura 16 para não perder a configuração de colunas até aqui alcançada.

Remova também a Tag ObjectDataSource no código fonte da página (listado a partir da imagem da figura 16). Para tanto, saia do modo Design e clique na aba Source da página ConsultaProduto

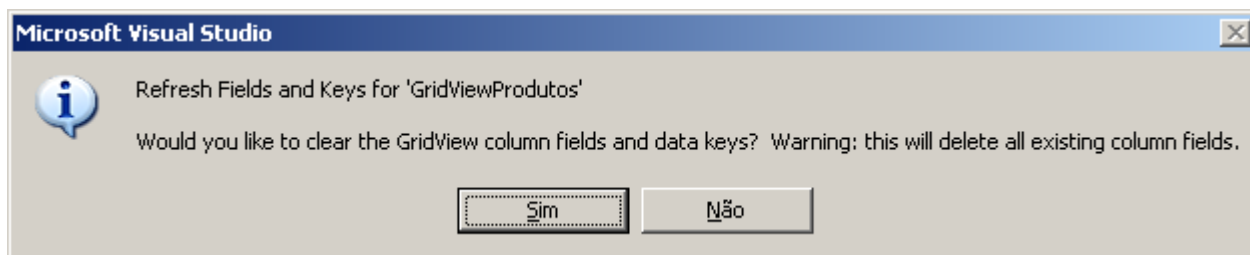


Figura 16

```
<asp:ObjectDataSource ID="ObjectDataSource1" runat="server"
    SelectMethod="ModeloProduto" TypeName="ProdutoController_SQL">
</asp:ObjectDataSource>
```

Comentando o código da Página ConsultaProduto

1- Cria uma instância da Classe ProdutoController (Prod) ao ser criado uma instância da Classe ConsultaProduto. A partir de Prod temos acesso aos métodos da Camada Controller.

```
public partial class ConsultaProduto : System.Web.UI.Page
{
    ProdutoController_SQL Prod = new ProdutoController_SQL();
}
```

2- Procedimento para o evento Click do objeto ButtonPesquisar. Primeiro é preenchido a propriedade DataSource do GridView com dados retornados do método ObterProduto da classe ProdutoController. Repare o valor da propriedade Text do controle TextBoxDescricaoProduto passado com o parâmetro. Por fim, é aplicado o método DataBind (funciona como um Refresh) para o controle GridView, para que os dados retornados pelo método ObterProduto possam ser exibidos.

```
protected void ButtonPesquisar_Click(object sender, EventArgs e)
{
    GridViewProdutos.DataSource = Prod.ObterProduto(TextBoxDescricaoProduto.Text);
    GridViewProdutos.DataBind();
}
```

3- O código acionado para o evento Click do objeto ButtonIncluir implementa:

```
protected void ButtonIncluir_Click(object sender, EventArgs e)
{
}
```

3.1 - Atribui para variável PaginaRetorno, do tipo Session, o nome da página aspx ConsultaProduto. Esta variável pode ser acessada por outra página dentro da seção, possibilitando a lógica desta página utilizá-la como parâmetro para um possível método de retorno (navegabilidade) de página.

```
Session["PaginaRetorno"] = "ConsultaProduto.aspx";
```

3.2 – Define o valor da variável Operação, do tipo Session. Esta variável servirá de Flag para o processo de manutenção de Produto, na página aspx CadastroProduto.

```
Session["Operacao"] = "Incluir";
```

3.3 – Usando o objeto Response, redireciona (método Redirect) o fluxo para a página aspx CadastroProduto.

```
Response.Redirect("CadastroProduto.aspx");
```

3.4 – Encerra o bloco de código do procedimento ButtonIncluir_Click.

```
}
```


4- Código para o evento Click do objeto ButtonAlterar. Identico ao item 3.

```
protected void ButtonAlterar_Click(object sender, EventArgs e)
{
    Session["PaginaRetorno"] = "ConsultaProduto.aspx";
    Session["Operacao"] = "Alterar";
    Response.Redirect("CadastroProduto.aspx");
}
```

5- Código para o evento Click do objeto ButtonExcluir. Identico ao item 3.

```
protected void ButtonExcluir_Click(object sender, EventArgs e)
{
    Session["PaginaRetorno"] = "ConsultaProduto.aspx";
    Session["Operacao"] = "Excluir";
    Response.Redirect("CadastroProduto.aspx");
}
```

6- Código para o evento Click do objeto ButtonVoltar. Usa o método Redirect de Response tendo como parâmetro a variável de seção PaginaRetorno. Repare o método ToString que converte o valor da variável d tipo Session para string, já que o método Redirect requer uma string.

```
protected void ButtonVoltar_Click(object sender, EventArgs e)
{
    Response.Redirect(Session["PaginaRetorno"].ToString());
}
```

7- O evento SelectedIndexChanged do objeto GridView é ideal para obter valores da linha selecionadas (com ação do click do mouse). Neste caso atribuímos a variável de seção PKProduto, o valor da propriedade Text da primeira coluna (index zero). O valor da variável PKProduto será utilizado na lógica da página Aspx CadastroProduto.

```
protected void GridViewProdutos_SelectedIndexChanged(object sender, GridViewSelectEventArgs e)
{
    Session["PKProduto"] = GridViewProdutos.Rows[e.NewSelectedIndex].Cells[0].Text;
}
```

8- Encerra o bloco de código da Classe ConsultaProduto.

```
}
```

```
if (Session["Operacao"].Equals("Incluir"))  
{  
    MsgErro = Produto.InserirProdutoMovimento(TextBoxDescricao.Text,  
                                                decimal.Parse(TextBoxPreco.Text));  
}
```

3.3- Verifica em else if (primeira condição else da estrutura If) se Operacao é igual a "Alterar". Sendo, passa para o método SetOld (seu parâmetro tipo DataTable) da instância de ProdutoController o conteúdo da variável de escopo Session, convertendo-o para um tipo DataTable. O método SetOldProduto armazena nos campos públicos da Classe ProdutoController relativo aos campos que serão utilizados na chamada ao método AlterarProduto, que encapsulando o mesmo método na Classe Produto, utilizará para os parâmetros da cláusula Where do comando Update. Em seguida chama o Método AlterarProduto passando os devidos valores para seus parâmetros. Aqui, novamente, é atribuído para variável MsgErro, a mensagem de erro resultante da execução do método caso exista.

```
else if (Session["Operacao"].Equals("Alterar"))
{
    Produto.SetOldProduto((DataTable)Session["OldProduto"]);
    MsgErro = Produto.AlterarProduto(TextBoxDescricao.Text,
                                     decimal.Parse(TextBoxPreco.Text),
                                     Produto.OldId,
                                     Produto.OldDescricao,
                                     Produto.OldPreco);
}
```

3.4- O mesmo que o item 3.3, mas para operação de exclusão.

```
else if (Session["Operacao"].Equals("Excluir"))
{
    Produto.SetOldProduto((DataTable)Session["OldProduto"]);
    MsgErro = Produto.ExcluirProduto(Produto.OldId,
                                     Produto.OldDescricao,
                                     Produto.OldPreco);
}
```

3.5- Atribui a propriedade Text do controle LabelErro, o valor da variável MsgErro. Este controle exibe por intermédio de sua propriedade Text, na parte inferior da página a mensagem de erro resultante da execução dos métodos de negócio.

```
LabelErro.Text = MsgErro;
MsgErro = "";
```

4- Utilizando como parâmetro o valor da variável de seção PaginaRetorno, invoca o método Redirect do objeto Response para orientar o fluxo para o formulário ConsultaProduto, neste caso.

```
protected void ButtonCancelar_Click(object sender, EventArgs e)
{
    Response.Redirect(Session["PaginaRetorno"].ToString());
}
```

5- Encerra o bloco de código da Classe CadastroProduto.

```
}
```

Testando a Aplicação

1- Execute a aplicação, e pelo menu da página de Index, selecione o item Produto fazendo logo em seguida uma busca de produto iniciado pela letra "C". Confira esses passos observando as figuras 01 e 02.

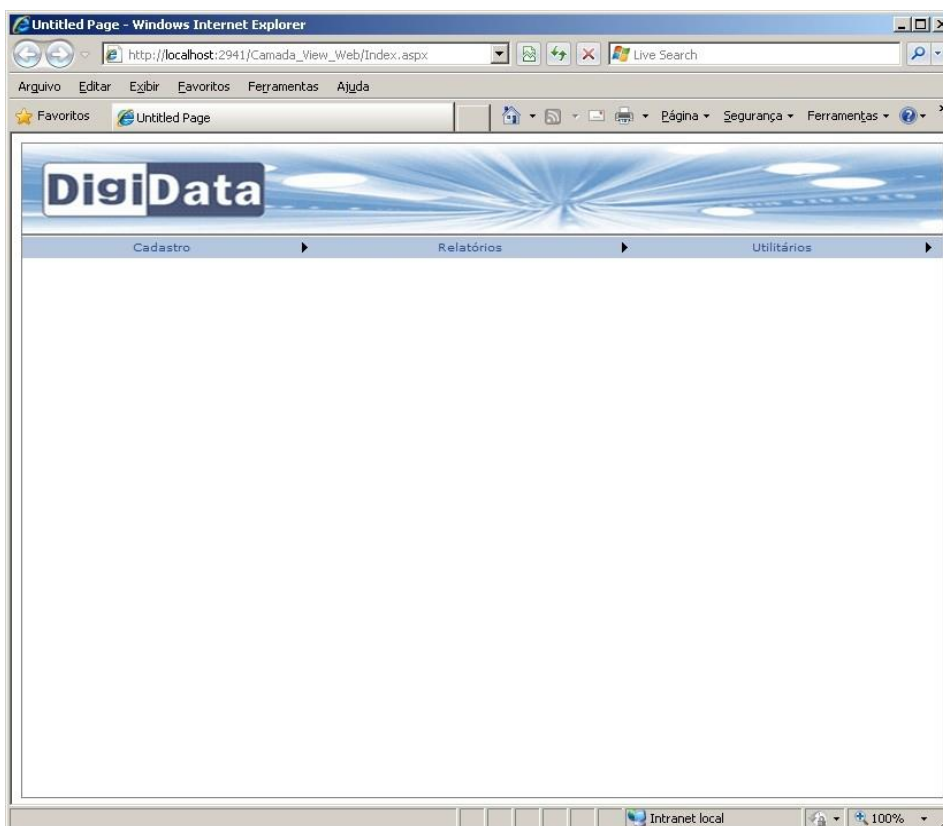


Figura 01

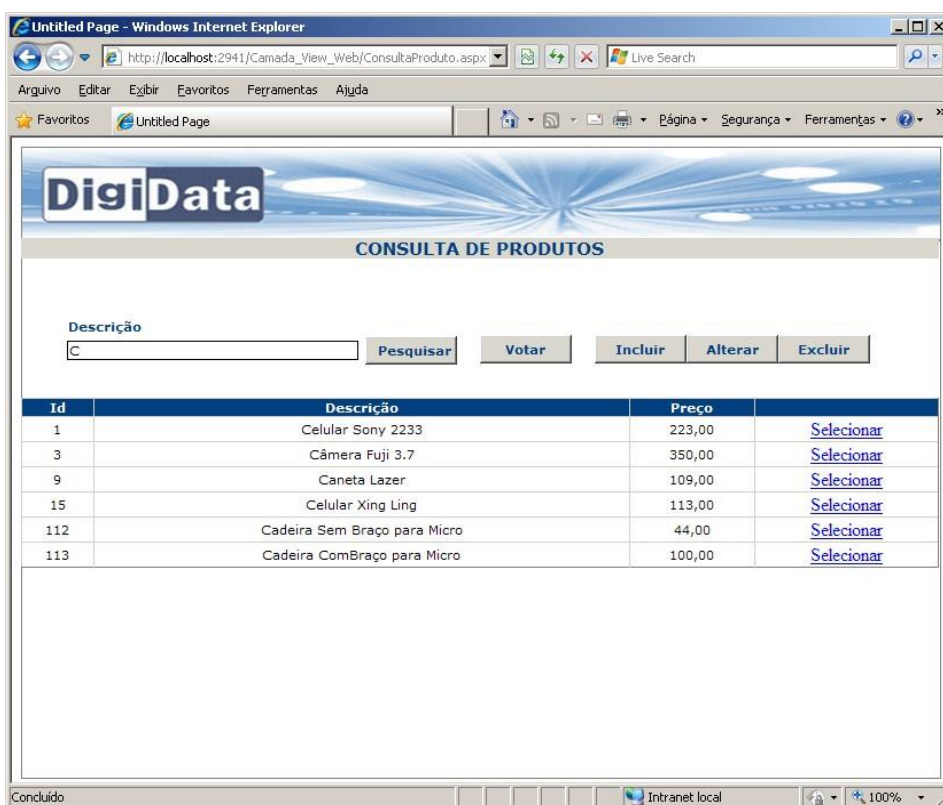


Figura 02

2- Clique sobre uma linha do GridView, por exemplo, na linha do produto Celular Xing Ling, e em seguida no botão Alterar. Como consequência será exibido a página representada pela figura 03. Promova alguma alteração, confirme e volte a página de consulta.

The screenshot shows a web browser window titled 'Untitled Page - Windows Internet Explorer'. The address bar displays 'http://localhost:2941/Camada_View_Web/CadastroProduto.aspx'. The page features a header with the 'DigiData' logo and the title 'CADASTRO DE PRODUTOS'. The main content area contains two text input fields: 'Descrição do Produto' with the value 'Celular Xing Ling - New' and 'Preço do Produto' with the value '113,00'. Below these fields are two buttons: 'Cancelar' and 'Confirmar'. The status bar at the bottom indicates 'Concluído' and 'Intranet local'.

Figura 03

Projeto WebServices em C#

Proposta: Desenvolver um Webservice para controlar as vendas em nosso site.

Web service é uma solução utilizada na integração de sistemas e na comunicação entre aplicações diferentes. Com esta tecnologia é possível que novas aplicações possam interagir com aquelas que já existem e que sistemas desenvolvidos em plataformas diferentes sejam compatíveis. Os Web services são componentes que permitem às aplicações enviar e receber dados em formato XML. Cada aplicação pode ter a sua própria "linguagem", que é traduzida para uma linguagem universal, o formato XML.

Para as empresas, os Web services podem trazer agilidade para os processos e eficiência na comunicação entre cadeias de produção ou de logística. Toda e qualquer comunicação entre sistemas passa a ser dinâmica e principalmente segura, pois não há intervenção humana.

Essencialmente, o Web Service faz com que os recursos da aplicação do software estejam disponíveis sobre a rede de uma forma normalizada. Outras tecnologias fazem a mesma coisa, como por exemplo, os browsers da Internet acedem às páginas Web disponíveis usando por norma as tecnologias da Internet, HTTP e HTML. No entanto, estas tecnologias não são bem sucedidas na comunicação e integração de aplicações. Existe uma grande motivação sobre a tecnologia Web Service pois possibilita que diferentes aplicações comuniquem entre si e utilizem recursos diferentes. Utilizando a tecnologia Web Service, uma aplicação pode invocar outra para efetuar tarefas simples ou complexas mesmo que as duas aplicações estejam em diferentes sistemas e escritas em linguagens diferentes. Por outras palavras, os Web Services fazem com que os seus recursos estejam disponíveis para que qualquer aplicação cliente possa operar e extrair os recursos fornecidos pelo Web Service. Os Web Services são identificados por um URI (Unique Resource Identifier), descritos e definidos usando XML (Extensible Markup Language). Um dos motivos que tornam os Web Services atrativos é o fato deste modelo ser baseado em tecnologias standards, em particular XML e HTTP (Hypertext Transfer Protocol). Os Web Services são utilizados para disponibilizar serviços interativos na Web, podendo ser acessados por outras aplicações usando, por exemplo, o protocolo SOAP (Simple Object Access Protocol).

O objetivo dos Web Services como é a comunicação aplicação para aplicação através da Internet. Esta comunicação é realizada com intuito de facilitar EAI (Enterprise Application Integration) que significa a integração das aplicações de uma empresa, ou seja, interoperabilidade entre a informação que circula numa organização nas diferentes aplicações como, por exemplo, o comércio eletrônico com os seus clientes e seus fornecedores. Esta interação constitui o sistema de informação de uma empresa. E para além da interoperabilidade entre as aplicações, a EAI permite definir um workflow entre as aplicações e pode constituir uma alternativa aos ERP (Enterprise Resource Planning). Com um workflow é possível otimizar e controlar processos e tarefas de uma determinada organização.

Padrão

O W3C, OASIS são as instituições responsáveis pela padronização dos Web Services. Empresas como IBM e Microsoft, duas das maiores do setor de tecnologia, apóiam o desenvolvimento deste padrão.

Segundo o W3C (World Wide Web Consortium) um Web Service define-se como: um sistema de software projetado para suportar a interoperabilidade entre máquinas sobre rede. Tem uma relação descritiva num formato machine-processable, especificamente WSDL (Webservice Description Language).

Outros sistemas interagem com o Web Service usando as mensagens SOAP, tipicamente sobre HTTP com XML na junção com outros standards da Web.

Tecnologias

As bases para a construção de um Web service são os padrões XML e SOAP. O transporte dos dados é realizado normalmente via protocolo HTTP ou HTTPS para conexões seguras (o padrão não determina o protocolo de transporte). Os dados são transferidos no formato XML, encapsulados pelo protocolo SOAP.

Segurança

Muitas empresas temiam, no passado, prover funcionalidades na Internet devido ao medo de expor seus dados. Mas com advento dos Web Services elas podem publicar serviços de forma simples e que são totalmente isolados da base de dados.

A segurança dos Web Services é um dos pontos fracos desta tecnologia. O problema não é a falta de mecanismos de segurança mas sim a falta de consenso em qual deve ser o mecanismo a ser adoptado pela tecnologia Web Service. As questões mais relevantes na segurança são as seguintes: - Autenticidade (ter a certeza que uma transacção do Web Service ocorreu entre o servidor e seu cliente); - Privacidade (todas as mensagens trocadas entre o servidor e o cliente não são interceptadas por uma pessoa não autorizada); - Integridade (as mensagens enviadas tanto pelo servidor ao cliente, como o contrário, devem permanecer inalteradas). Em seguida descreve-se os principais mecanismos de segurança.

SSL

O SSL (Secure Socket Layer) [Netscape 1996] quando aplicado a pequenos dispositivos oferece autenticação, integridade de dados e privacidade de serviços. Atualmente, a solução para enviar informação confidencial para Web Services é utilizar um mecanismo de segurança SSL sobre HTTP também conhecido como HTTPS (Hypertext Transfer Protocol Secure). Este mecanismo protege informações confidenciais e é fácil de ser configurado. Tem como desvantagem ser mais lento do que as transações HTTP não encriptadas pelo que não é adequado para taxas de transferências de dados elevadas.

XML SIGNATURE

A XML Signature [IETF e W3C 2000] é uma iniciativa conjunta da IETF (Internet Engineering Task Force) e do W3C para especificar uma sintaxe XML e regras de processamento para criação e representação digital de assinaturas. As vantagens na utilização da XML Signature, ao contrário de outras normas de assinaturas digitais, estão baseadas na independência da linguagem de programação, fácil interpretação humana e independência do fabricante. Esta tecnologia também permite assinar digitalmente subconjuntos de um documento XML.

XML ENCRYPTION

A XML Encryption [IETF e W3C 2002] especifica um processo para encriptação de dados e sua representação em formato XML. Os dados podem ser dados arbitrários (incluindo um documento XML), elementos XML ou conteúdos de elementos XML. Um documento XML que utiliza a XML Encryption pode ser visto por qualquer utilizador, mas apenas o proprietário da chave de decodificação conseguirá compreender o conteúdo codificado.

WS-SECURITY

O WS-Security (Web Services Security) é uma iniciativa conjunta de empresas como Microsoft, IBM e Verisign destinada ao uso da XML-Signature e da XML-Encryption para fornecer segurança às mensagens SOAP. O WS-Security é um esforço destinado a fazer com que os Web Services trabalhem melhor em um ambiente global. O WS-Security também inclui alguns importantes componentes como encaminhamento, confiança e tratamento de transações.

SAML

O SAML (Security Assertion Markup Language) [OASIS 2001] é uma norma emergente para a troca de informação sobre autenticação e autorização. O SAML soluciona um importante problema para as aplicações da próxima geração, que é a possibilidade de utilizadores transportarem seus direitos entre diferentes Web Services. Isto é importante para aplicações que tencionam integrar um número de Web Services para formar uma aplicação unificada.

Limitações associados aos Web Services

Apesar da sua grande popularidade e relativa simplicidade, o SOAP tem várias limitações, que por sua vez os Web Services também sofrem essas limitações pois utilizam o SOAP. As limitações são descritas em seguida:

- Segurança e privacidade – nenhuma das versões do SOAP define qualquer tipo de segurança. Isto é devido ao SOAP utilizar HTTP, mas para implementar mecanismos de segurança ao nível da rede pode utilizar o protocolo SSL no HTTP (também conhecido como HTTPS) para garantir a confidencialidade, a integridade e a autenticação do cliente, do servidor e da comunicação cifrada. Como não existe um suporte para segurança, que inclui a privacidade, nas normas que compõem os Web Services, tem levado cada projeto a procurar diferentes soluções para resolver o problema da segurança o que se torna incompatível com a promessa de implementar uma normalização a nível global.
- Mensagens e encaminhamento – para suportar as funcionalidades das mensagens assíncronas tradicionais
- Qualidade de serviço e fiabilidade – para garantir tempos de resposta e detectar exceções
- Processamento transaccional – para suportar comunicação transaccional, para associar essa comunicação transaccional com as transações locais e para participar em transações distribuídas
- Gestão – para controlar o estado e comportamento dos Web Services
- Desempenho – para otimizar a execução dos Web Services que tem implicações ao nível do desenho das aplicações, chamadas remotas, características da rede e armazenamento/processamento dos documentos
- Interoperabilidade – suportar a interoperação sem problemas é o grande objetivo dos Web Services e do SOAP, ou seja, fornecerem uma plataforma de integração entre aplicações e diferentes linguagens e implementados em qualquer sistema operativo. Assim esta tecnologia seria uma tecnologia normalizada mas, no entanto, existem rivalidades entre fornecedores. Por exemplo, o AXIS (implementação SOAP do projeto Apache) não é compatível com .Net da Microsoft, por isso gera interfaces em dois formatos, uma para consumo próprio e outra diferente para o .Net ser compatível com o Axis.

Para resolver este problema existe várias abordagens para resolver este problema como o SOAPBuilders Interoperability Lab que fornece uma plataforma para testar a interoperabilidade dos produtos.

Integração de sistemas

Muitos consideram que os Web services corrigem um grande problema da informática: a falta de integração de sistemas.

Os Web services permitem que a integração de sistemas seja realizada de maneira compreensível, reutilizável e padronizada.

É uma tentativa de organizar um cenário cercado por uma grande variedade de diferentes aplicativos, fornecedores e plataformas.

O futuro dos Web Services

Acredita-se que no futuro as empresas irão listar seus Web services em diretórios públicos (UDDI), de onde poderão ser vendidos como serviços para outras empresas, instituições ou usuários comuns...

Tecnologias Utilizadas

Para a representação e estruturação dos dados nas mensagens recebidas/enviadas é utilizado o XML (eXtensible Markup Language). As chamadas às operações, incluindo os parâmetros de entrada/saída, são codificadas no protocolo SOAP (Simple Object Access Protocol, baseado em XML). Os serviços (operações, mensagens, parâmetros, etc.) são descritos usando a linguagem WSDL (Web Services Description Language). O processo de publicação/pesquisa/descoberta de Web Services utiliza o protocolo UDDI (Universal Description, Discovery and Integration).

XML

Extensible Markup Language (XML) é a base em que os Web Services são construídos. O XML fornece a descrição, o armazenamento, o formato da transmissão para trocar os dados através dos Web Services e também para criar tecnologias Web Services para a troca dos dados.

A sintaxe de XML usada nas tecnologias dos Web Services especifica como os dados são representados genericamente, define como e com que qualidades de serviço os dados são transmitidos, pormenoriza como os serviços são publicados e descobertos. Os Web Services descodificam as várias partes de XML para interagir com as várias aplicações.

SOAP

O SOAP (Simple Object Access Protocol) baseia-se numa invocação remota de um método e para tal necessita de especificar o endereço do componente, o nome do método e os argumentos para esse método. Estes dados são formatados em XML com determinadas regras e enviados normalmente por HTTP para esse componente. Não define ou impõe qualquer semântica, quer seja o modelo de programação, quer seja a semântica específica da implementação. Este aspecto é extremamente importante, pois permite que quer o serviço, quer o cliente que invoca o serviço sejam aplicações desenvolvidas sobre diferentes linguagens de programação. Por esta razão, o SOAP tornou-se uma norma aceite para se utilizar com Web Services, uma tecnologia construída com base em XML e HTTP. Desta forma, pretende-se garantir a interoperabilidade e intercomunicação entre diferentes sistemas, através da utilização da linguagem XML e do mecanismo de transporte HTTP ou outro como, por exemplo, SMTP. O SOAP permite que os documentos XML de envio e de recepção sobre a Web suportem um protocolo comum de transferência de dados para uma comunicação de rede eficaz, ou seja, o SOAP providencia o transporte de dados para os Web Services. Em relação a Web, o SOAP é um protocolo de RPC que funciona sobre HTTP (ou SMTP, ou outro) de forma a ultrapassar as restrições de segurança/firewalls normalmente impostas aos sistemas clássicos de RPC (RMI, DCOM, CORBA/IIOP) suportando mensagens XML. Em vez de usar HTTP para pedir uma página HTML para ser visualizada num browser, o SOAP envia uma mensagem de XML através do pedido HTTP e recebe uma resposta, se existir, através da resposta do HTTP. Para assegurar corretamente a transmissão da mensagem de XML, o servidor de HTTP, tais como Apache ou IIS (Microsoft Internet Information Server), recebe mensagens SOAP e deve validar e compreender o formato do documento XML definido na especificação SOAP v1.1.

WSDL

É a sigla de Web Services Description Language, padrão baseado em XML para descrever o serviço como no COM, onde ele traz os métodos do Web Service. Funciona como uma espécie de "TypeLibrary" do Web Service, além de ser usado para a validação das chamadas dos métodos.

O WSDL (Web Services Description Language) é uma especificação desenvolvida pelo W3C que permite descrever os Web Services segundo um formato XML. O WSDL é extensível para permitir a descrição dos serviços e suas mensagens, independentemente dos formatos de mensagem e dos protocolos de rede que sejam usados. No entanto, é comum usar-se o MIME (Multipurpose Internet Mail Extensions) e o HTTP/SOAP. O WSDL descreve os serviços disponibilizados à rede através de uma semântica XML, este providencia a documentação necessária para se chamar um sistema distribuído e o procedimento necessário para que esta comunicação se estabeleça. Enquanto que o SOAP especifica a comunicação entre um cliente e um servidor, o WSDL descreve os serviços oferecidos.

UDDI

Protocolo desenvolvido para a organização e registro de Web Services.

O UDDI (Universal Description Discovery and Integration) é uma iniciativa em desenvolvimento no âmbito do consórcio industrial UDDI promovido originalmente pela IBM, Microsoft e Arriba, com objetivo de acelerar a interoperabilidade e utilização dos Web Services, pela proposta de um serviço de registo de nomes de organizações e de descrição do serviço.

Um registro UDDI contém três tipos de informação: • informações gerais de cada organização, tais como o nome, morada, telefone e contatos; • informações de organizações e serviços por categorias de negócios; • informações técnicas sobre os serviços providenciados pelas organizações.

O UDDI providencia três funções principais, conhecidas como publicação, descoberta e ligação:

- 1) publicação: permite que uma organização divulgue o(s) seu(s) serviço(s);
- 2) descoberta: permite que o cliente do serviço, procure e encontre um determinado serviço;
- 3) ligação (bind): permite que o cliente do serviço, possa estabelecer a ligação e interagir com o serviço.

WS-I

É o consórcio que garante a integração entre os Web Services para garantir sempre que os Web Services possam "conversar entre-si".

Iniciativas em curso

O sucesso que os Web Services possam vir a apresentar passa necessariamente pela vontade da indústria, pela partilha e abertura dos processos de normalização e das próprias especificações daí resultantes. Parte significativa desse processo tem sido desenvolvida no âmbito do W3C. No entanto, dever-se-á também referir outros esforços e consórcios que têm vindo a ser desenvolvidos, designadamente o UDDI, o ebXML, ou o XML/EDI. Por exemplo, o ebXML é um esforço patrocinado pela UN/CEFACT e pela OASIS, cujo objetivo é a produção de um conjunto de especificações para permitir colaborações de negócio eletrónico. O standard ebXML pode ser visto como uma extensão às funcionalidades de descrição, publicação e descoberta de serviços (definidas no âmbito do UDDI), ao tratar os seguintes aspectos: como especificar os processos de negócio; como identificar os Web Services participantes e respectivas colaborações; ou, que padrões de negociação existem na colaboração entre os participantes. Estes aspectos, são tratados nomeadamente nas seguintes especificações:

- 1) esquemas para especificação de processos de negócio, BPSS (business process specification schema);
- 2) acordos de protocolos de colaboração, CPA (collaboration protocol agreement);
- 3) ou perfis de protocolos de colaboração, CPP (collaboration protocol profile).

Contribuição das empresas

As principais empresas, para além de promoverem e participarem ativamente nos vários consórcios de normalização, têm vindo a incorporar nas suas próprias infra-estruturas de desenvolvimento e suporte de aplicações implementações das normas ligadas aos Web Services. Entre outras, merece referência a plataforma da Microsoft, ".Net", da Sun, "Java ONE (Open Net Environment)", da Hewlett-Packard, "e-speak" e da IBM, "IBM Web Services".

Novos Modelos de Negócio

Só o futuro dirá quem tem razão: se os cépticos ou conservadores, se os que arriscam e concretizam a sua visão. Com o conceito dos Web Services talvez o mais importante nem seja a tecnologia em si, mas toda uma discussão à volta dos fatores económico-políticos que este paradigma poderá suscitar, bem como os modelos de negócio que poderão emergir. Parece natural a emersão de novos portais, não para as pessoas consultarem e usarem, mas para as aplicações, i.e., para os serviços se registarem/publicarem de modo a tornarem-se conhecidos, descobertos e usados. Esses portais de serviços (tecnicamente consiste em serviços de registos UDDI e/ou ebXML) poderão ser definidos a nível global, regional, para domínios de negócio horizontais ou verticais.

Novos Requisitos Tecnológicos

No entanto e naturalmente, novos problemas e requisitos tecnológicos são colocados com o conceito dos Web Services. Desde logo, ao nível da modelação destes serviços e dos processos de negócio em que aqueles participam. Aspectos como a composição de serviços, coordenação de fluxos de trabalho, identificação e privacidade, segurança, negociação, contratos e pagamentos, tratamento de exceções, categorização e taxonomias de serviços, etc., deverão ser adequadamente investigados e tratados de forma que este paradigma possa vir a apresentar um largo consenso e sucesso.

Vantagens e Desvantagens

Os Web Services são modelos que surgiram para o desenvolvimento de aplicações típicas de negócio eletrônico, envolvendo e suportando o estabelecimento da colaboração e negociação de forma aberta, distribuída e dinâmica entre distintos parceiros. Os Web Services podem no futuro representar um sucesso significativo por causa de existir um esforço significativo, por parte da maioria dos parceiros industriais, na normalização das tecnologias envolvidas. As tecnologias subjacentes aos Web Services (tais como HTTP, SOAP, WSDL, UDDI, XML) são abertas, amplamente divulgadas e consensuais. Por outro lado, existe potencial para haver uma real independência das linguagens de programação (Java, C++, VB, Delphi, C#), das arquiteturas de computadores e sistemas operativos, o que permite uma evolução mais suave e econômica para este modelo computacional.

No entanto, existe críticas que demonstram medos ou falsas expectativas que os investimentos em Web Services podem suscitar. Uma dessas críticas diz respeito ao fato do SOAP é menos eficiente do que os sistemas de RPC existentes. Por exemplo, as mensagens (com os respectivos envelopes e descrição de tipos) trocadas entre as partes são descritas em formato de texto/XML enquanto que nos sistemas clássicos de RPC são trocadas em formato binário. No entanto, esta desvantagem é compensada significativamente pela facilidade de interoperação entre os serviços, sem os problemas conhecidos de segurança/firewalls, e pela facilidade de se esconder os detalhes proprietários das infra-estruturas de suporte.

Criando WebService

O objetivo deste exercício, é exemplificar a criação de um WebService para servir de Interface Remota (pelo protocolo http) das classes de negócio do exercício NameSpaceSQL, acessadas pela camada Vieww e Controller.

Com essa abordagem, poderemos facilmente estender as funcionalidades da camada de negócio (Model) a aplicativos, sejam baseado em Asp.Net ou WinForm, para acesso remoto. Para tanto, necessitamos apenas construir o WebService com métodos que exponham as funcionalidades da camada Model permitindo acesso via http (Web Reference no VisualStdio).

Com essa atitude, podemos classificar nosso projeto (Model, WebService, Controller e View) não somente adequada ao paradigma MVC, mas também agora, como um aplicativo baseado em Objetos Distribuídos.

Passo 01

Conforme exibido na figura 01, com o seu ambiente VisualStudio em execução, crie um projeto ASP.NET Web Service, definindo o local conforme definido. Repare que optamos pela linguagem Visual C#.

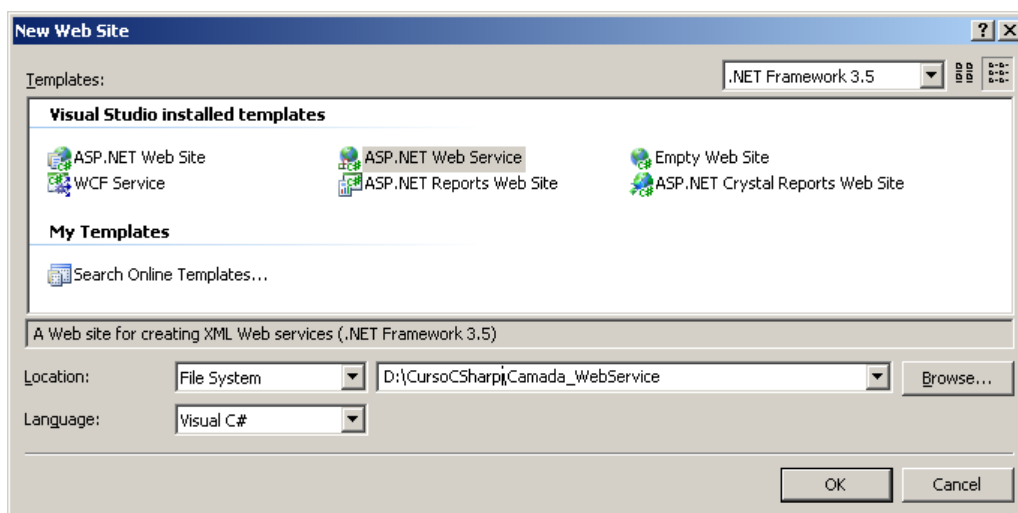


Figura 01

Passo 02

Observe o código da Listagem 01, gerado automaticamente, representando a classe nomeada Service. Repare ainda o método público do tipo string, denominado HelloWorld, criado automaticamente para projeto.

Listagem 01

```
using System;
using System.Linq;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Xml.Linq;

[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
// To allow this Web Service to be called from script, using ASP.NET AJAX, uncomment the following line.
// [System.Web.Script.Services.ScriptService]
public class Service : System.Web.Services.WebService
{
    public Service () {
        //Uncomment the following line if using designed components
        //InitializeComponent();
    }
}
```

```
[WebMethod]
public string HelloWorld() {
    return "Hello World";
}

}
```

Nota: Vale observar o método HelloWorld, que é precedido de uma anotação "[WebMethod]" garantido ao mesmo exposição para os aplicativos consumidores do WebService. Outros métodos podem existir, mas não necessariamente sendo expostos aos aplicativos consumidores, bastando apenas não serem precedido da referida anotação. Naturalmente métodos não anotados com [WebMethod], só podem ser acessados por outros métodos da classe, anotados ou não.

Passo 03

Execute o projeto e tenha uma surpresa, pois sem nenhum esforço temos uma aplicação de teste expondo um link para o método HelloWorld. Confira na Figura 03.

Este procedimento, executar o WebService e termos uma resposta na Browse é uma facilidade que temos no ambiente do VisualStudio, mas que é limitado para testes preliminares e principalmente para debug.

Nota: Conforme demonstrado na Figura 02, é exibido a caixa de dialogo Debugging Not Enabled solicitando permissão ou não para configuração do arquivo Web.Config, que se positivo, habilitará o recurso de "debug" pra nosso WebService.

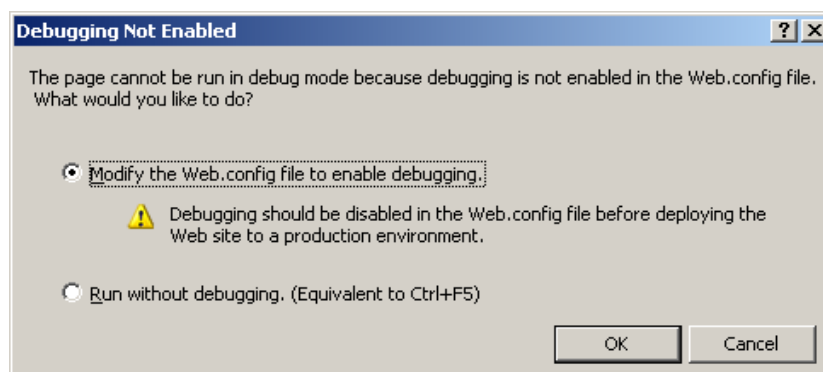


Figura 02

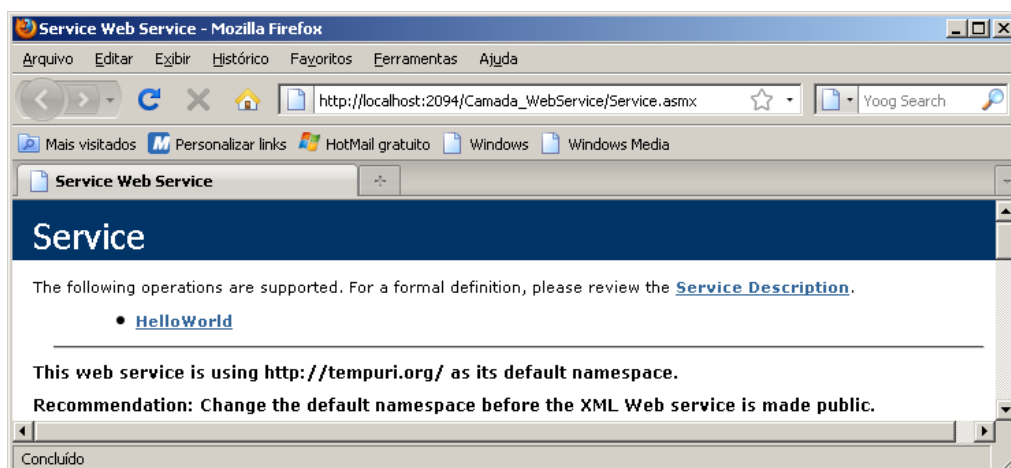


Figura 03

Dando continuidade ao Passo 03, se orientando pelas Figuras 04 e 05, finalize o teste conferindo o resultado.

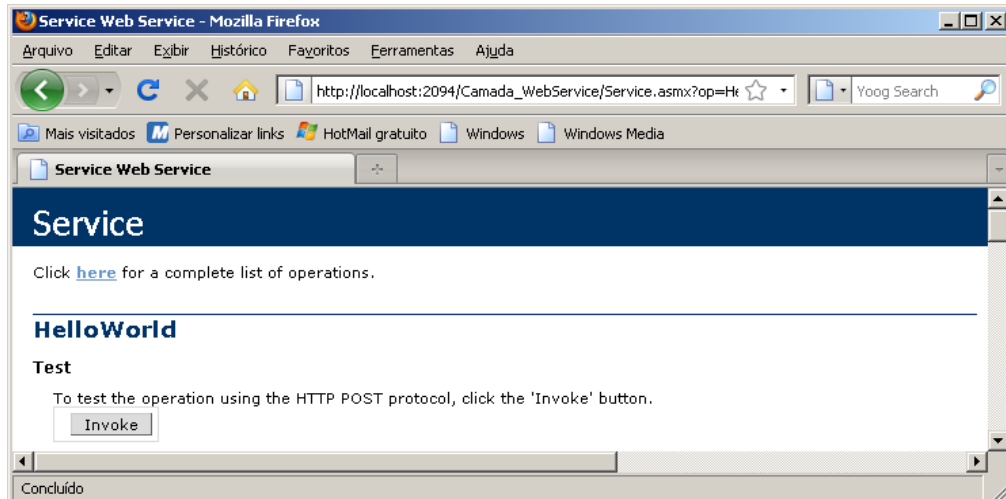


Figura 04

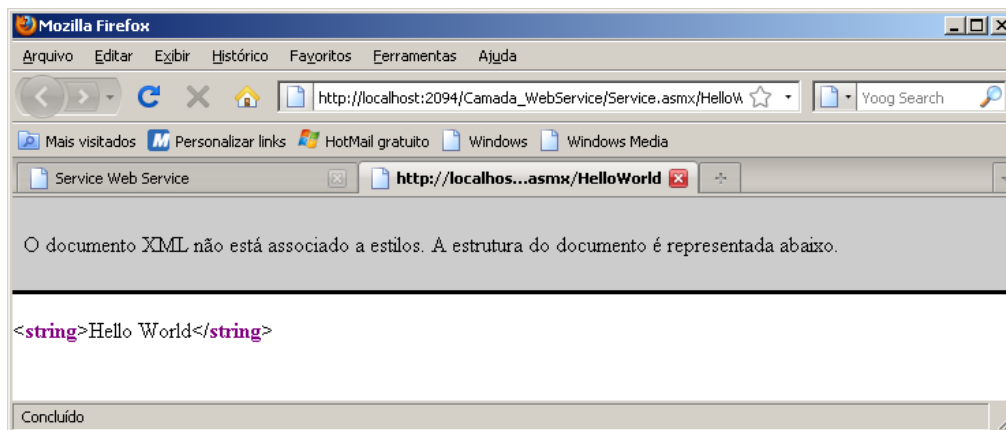


Figura 05

Passo 04

Visando alinhar os nomes das classes de nosso WebService com os objetos de negócio da camada Modelo, renomeie tanto o nome da classe quanto o método construtor para **WSProduto** refletindo o definido nas Listagem 02. A Figura 06 exibe a caixa de dialogo Solution Explorer do projeto após as alterações promovidas.

Listagem 02

```
public class WSProduto : System.Web.Services.WebService
{
    public WSProduto() {
    }
    ... Código omitido aqui.
```

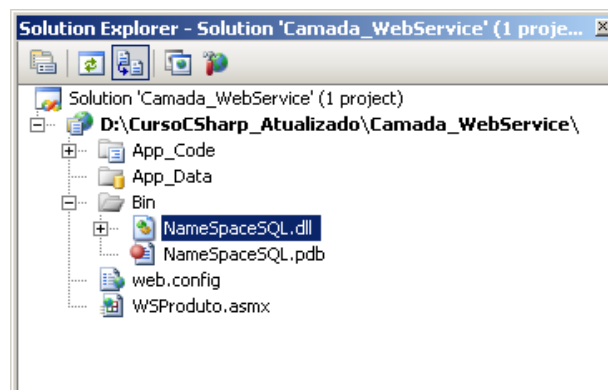


Figura 06

Passo 05

Acrescente o método `RetornaProduto`, preocupando-se em anotá-lo com `[WebMethod]` para que tenha visibilidade para os clientes consumidores do `WebService`. A Listagem 03 contém todo o código fonte da classe `WSProduto`, incluindo é claro as modificações e acréscimos relativos Passos 04 e 05, em negrito para facilitar.

Listagem 03

```
1. using System;
2. using System.Linq;
3. using System.Web;
4. using System.Web.Services;
5. using System.Web.Services.Protocols;
6. using System.Xml.Linq;
7. using System.Data;
8.
9. [WebService(Namespace = "http://tempuri.org/")]
10. [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
11.
12. public class WSProduto : System.Web.Services.WebService
13. {
14.     public WSProduto()
15.     {
16.     }
17.
18.     [WebMethod]
19.     public string HelloWorld() {
20.         return "Hello World";
21.     }
22.
23.     [WebMethod]
24.     public DataTable RetornaProduto(String Descricao)
25.     {
26.         NamespaceSQL.Produto Produto = NamespaceSQL.Produto();
27.         return Produto.RetornaProduto(Descricao);
28.     }
29.
30. }
```

Comentando a Listagem 03

Linha	Comentário
14	Método construtor da classe.
23	Anotação que torna o método acessível aos clientes (consumidores do <code>WebService</code>).
24	Define um método público que implementa um parâmetro do tipo <code>String</code> , retornando um tipo <code>DataTable</code> (<code>Collection</code>).
26	Cria uma instância do objeto (classe) <code>Produto</code> da <code>NamespaceSQL</code> anteriormente referenciada.
27	Executa o método da Classe <code>produto</code> , retornando uma lista (<code>DataTable</code>) de produtos.

Passo 06

Finalizando a implementação, altere o código do arquivo WSProduto.asmx, definindo o novo nome da Classe. Originalmente o nome era Service, aqui estamos renomeando para WSProduto conforme linhas de código abaixo.

Versão Original

```
<%@ WebService Language="C#" CodeBehind="~/App_Code/Service.cs" Class="Service" %>
```

Versão Modificada

```
<%@ WebService Language="C#" CodeBehind="~/App_Code/Service.cs" Class="WSProduto" %>
```

Passo 07

Fazendo valer o benefício fornecido pela ferramenta, em implementar um aplicativo de teste de consumo de métodos de nosso WebService, utilize a opção de menu Debug | Start Debugging para executar o teste.

Baseado na imagem da Figuras 07, clique no link RetornaProduto, e em seguida, agora se orientando pela imagem da Figura 08, preencha o campo "Descricao" com a letra "C" clicando no botão "Invoke" para que o método seja executado no WebService. A Listagem 04 exibe em formato XML o pacote de dados retornado pelo método, exibindo os registros da tabela Produto que tem descrição iniciada com a letra "C" passado como parâmetro.

Nota: Apesar da afirmação anterior, na prática, nosso WebService não acessa a tabela Produto no banco de dados. Na verdade ela submete uma chamado ao método relativo na Camada de Negócio, e esse sim interge com a base de dados.

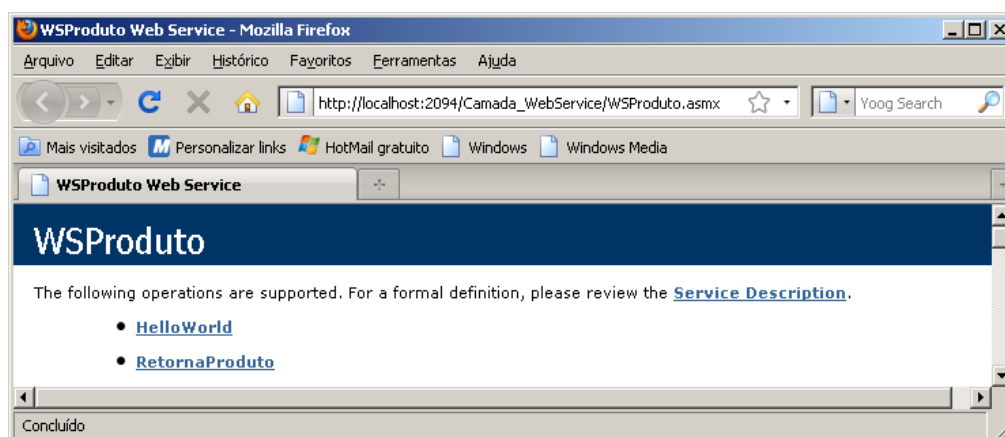


Figura 07

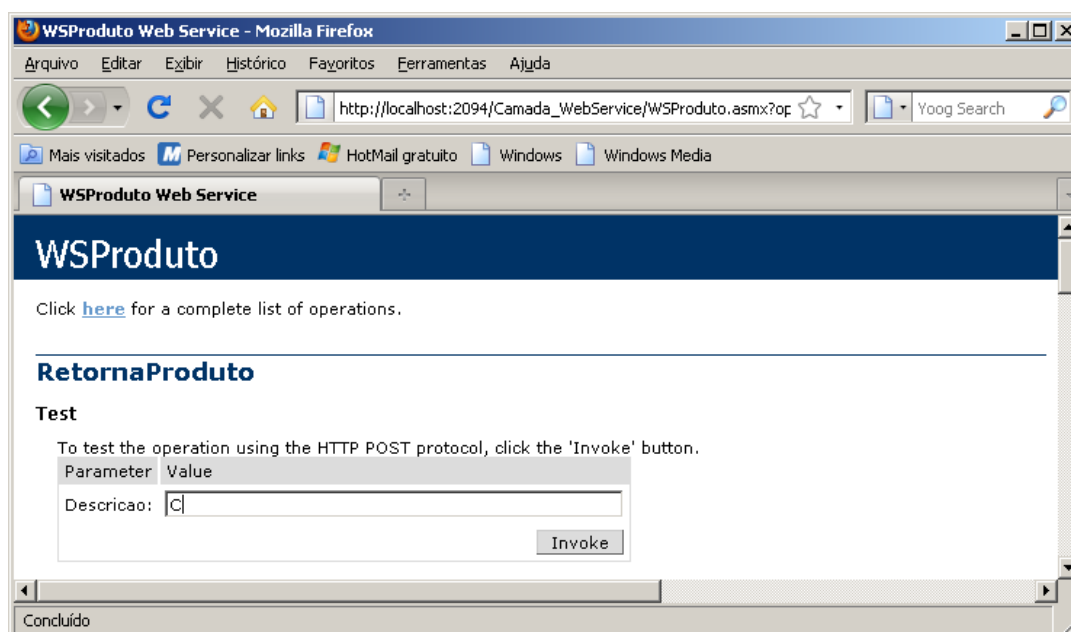


Figura 08

Listagem 04

<DataTable>

```
<xs:schema id="NewDataSet">
  <xs:element name="NewDataSet" msdata:IsDataSet="true"
    msdata:MainDataTable="Table" msdata:UseCurrentLocale="true">
    <xs:complexType>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="Table">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Id" type="xs:long" minOccurs="0"/>
              <xs:element name="Descricao" type="xs:string" minOccurs="0"/>
              <xs:element name="Preco" type="xs:decimal" minOccurs="0"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:choice>
    </xs:complexType>
  </xs:element>
</xs:schema>
<diffgr:diffgram>
  <NewDataSet>
    <Table diffgr:id="Table1" msdata:rowOrder="0">
      <Id>1</Id>
      <Descricao>Celular Sony 2233</Descricao>
      <Preco>223.00</Preco>
    </Table>
    <Table diffgr:id="Table2" msdata:rowOrder="1">
      <Id>3</Id>
      <Descricao>Câmera Fuji 3.7</Descricao>
      <Preco>350.00</Preco>
    </Table>
    <Table diffgr:id="Table3" msdata:rowOrder="2">
      <Id>9</Id>
      <Descricao>Caneta Lazer</Descricao>
      <Preco>109.00</Preco>
    </Table>
    <Table diffgr:id="Table4" msdata:rowOrder="3">
      <Id>15</Id>
      <Descricao>Celular Xing Ling</Descricao>
      <Preco>113.00</Preco>
    </Table>
    <Table diffgr:id="Table5" msdata:rowOrder="4">
      <Id>112</Id>
      <Descricao>Cadeira Sem Braço para Micro</Descricao>
      <Preco>44.00</Preco>
    </Table>
    <Table diffgr:id="Table6" msdata:rowOrder="5">
      <Id>113</Id>
      <Descricao>Cadeira ComBraço para Micro</Descricao>
      <Preco>100.00</Preco>
    </Table>
  </NewDataSet>
</diffgr:diffgram>
</DataTable>
```


Configurando o IIS (Internet Information Services)

Para que possamos fazer uso do WebService WSProduto, referenciando-o em nosso projeto Asp.Net (Camada View) e consumir seus métodos siga os passos a seguir.

Vale ressaltar, que apesar de podermos testar nosso WebService diretamente no ambiente do VisualStudio, é necessário disponibilizarmos o serviço em um Servidor Web, em nosso caso é claro o IIS da Microsoft.

Passo 01

Execute o utilitário IIS, tendo como referência a Figura 01, clique com o botão direito do mouse sobre o nó "Site da Web Padrão", escolhendo a opção "Novo | Diretório Virtual...". Com isso iniciamos o passo-a-passo para criação de um diretório virtual que servirá de referência de acesso ao nosso WebService por intermédio de uma URL. Para tanto, os clientes consumidores do WebService implementarão uma URL por intermédio do protocolo HTTP conforme exemplo a seguir.

Exemplo de URL com Protocolo HTTP

http:\\MeuServidor\\NomeDiretorioVirtual\\WSProduto.asmx

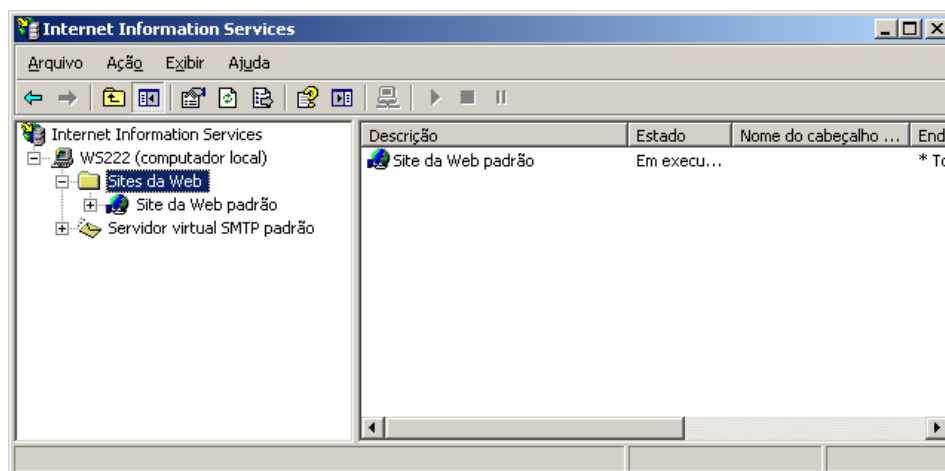


Figura 01

Ainda neste passo, clique no botão "Avançar" na caixa de dialogo apresentada em função da ação anterior para prosseguir na criação do Diretório Virtual para acesso ao projeto WebService. A Figura 02 resume isso.

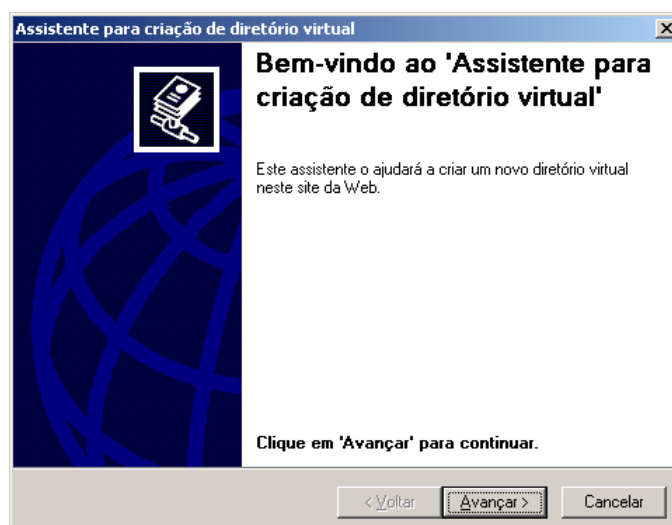


Figura 02

Passo 02

Dando continuidade, defina o nome "WSProduto" na propriedade "Alias" da caixa de dialogo representada pela Figura 03. Esse naturalmente será o nome do Diretório Virtual, o melhor resumindo, o nome de referência na URL para acesso ao projeto WebService.

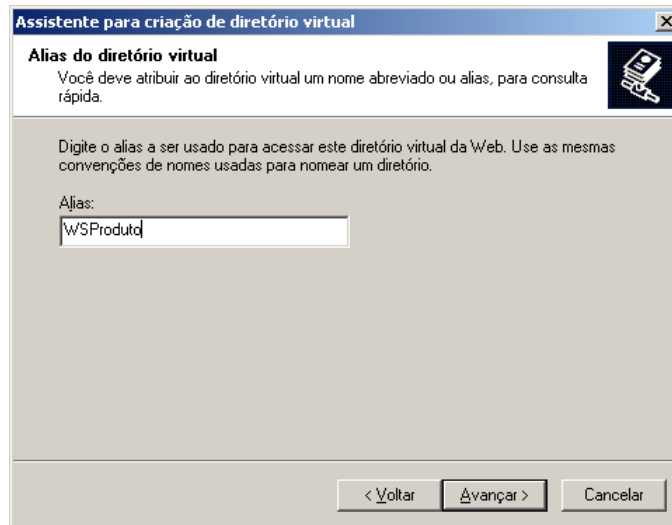


Figura 03

Passo 03

Conforme demonstrado na imagem da Figura 04 aponte para o diretório que contem o projeto WebService, indicando ao IIS onde encontrar os serviços solicitados pela URL que invocará classes de WSProduto. Clique no botão "Avançar" para prosseguir no processo.

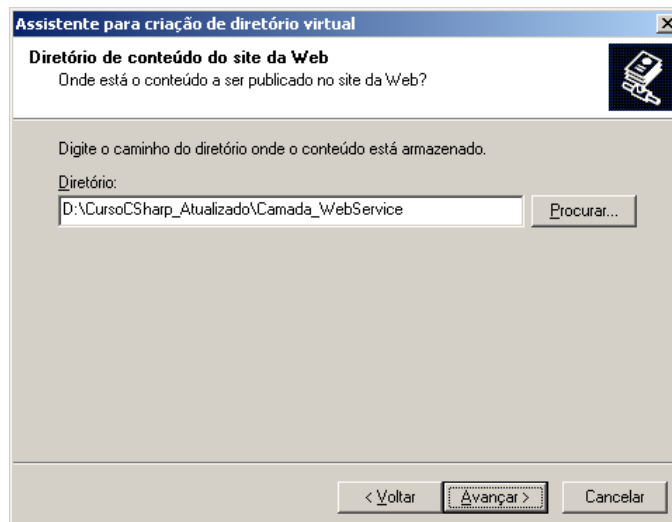


Figura 04

Passo 04

A Caixa de dialogo exibida após o passo anterior, Figura 05, permite configurar características que irão autorizar ou não (dar permissão) ao ISS de execução aos scripts Asp.Net. Proceda como exibida na imagem da Figura 05, clicando no botão "Avançar".

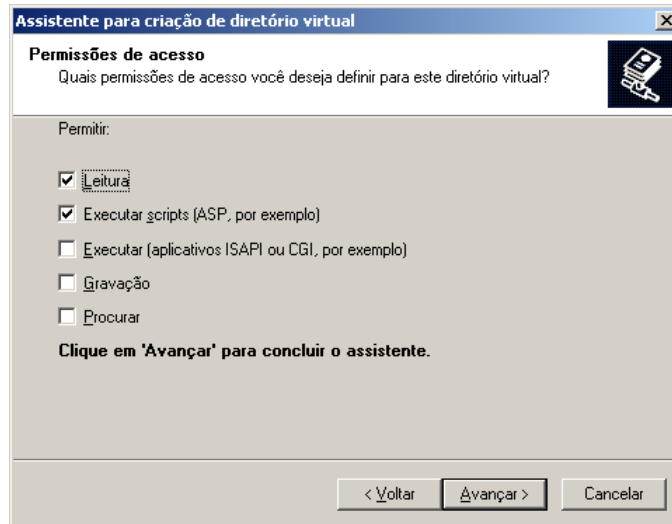


Figura 05

Passo 05

Basta clicar no botão "Concluir" da Figura 06, assim, encerrando os passos necessários para divulgarmos nosso Webservice. A Figura 07 demonstra nosso Diretório Virtual (Alias de acesso ao Webservice) devidamente configurado.

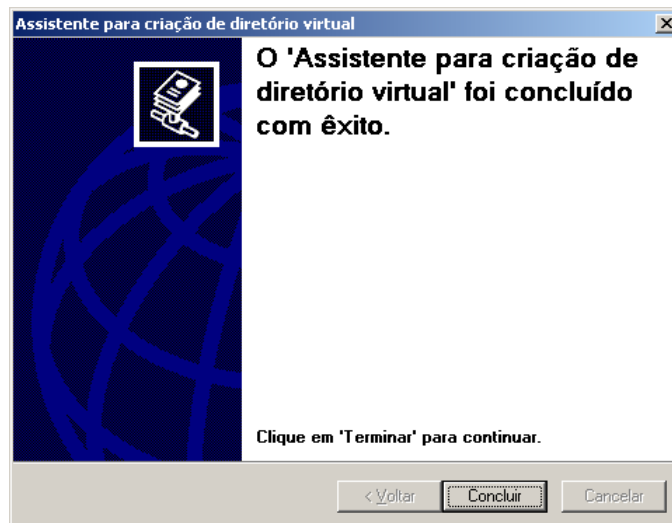


Figura 06

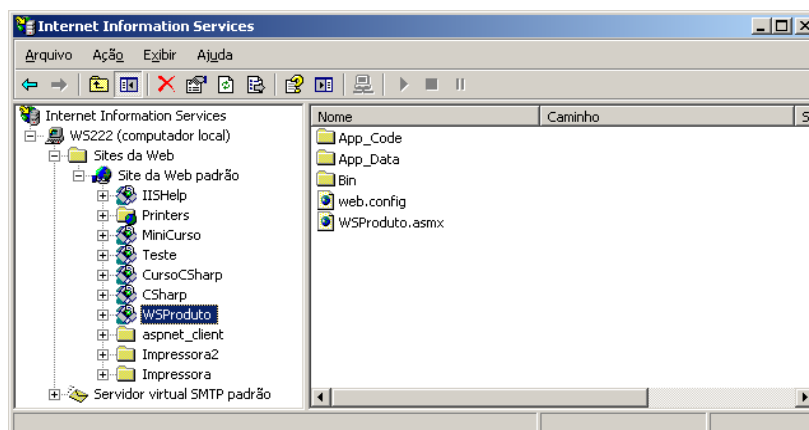


Figura 07

Consumindo o WebService na Aplicação Web

Passo 01

Basicamente fazemos uma referência ao WebService alvo, clicando com o botão direito do mouse no nó do projeto Web e selecionando a opção do menu de contexto "Add Web Reference...". Você pode acionar esta opção orientando-se pela imagem da Figura 01.

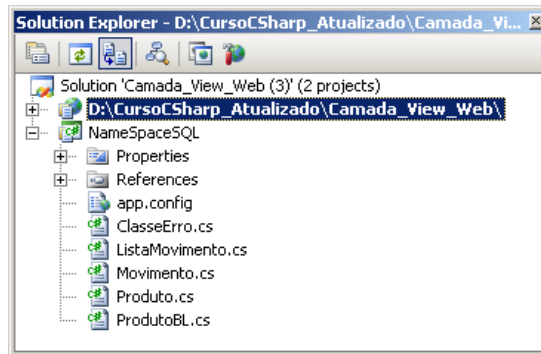


Figura 01

Passo 02

Orientando-se pela Figura 02, complete a ação anterior preenchendo a URL de acesso ao WebService. Naturalmente devemos montar a referida URL conforme esquema a seguir:

- HTTP://
- Número IP do Servidor ou Nome do Servidor/
- Nome do Alias criado no IIS/
- Nome do arquivo ASMX que representa o WebService

Em nosso exemplo ficaria então assim: **http://WS222/WSProduto/WSProduto.aspx**

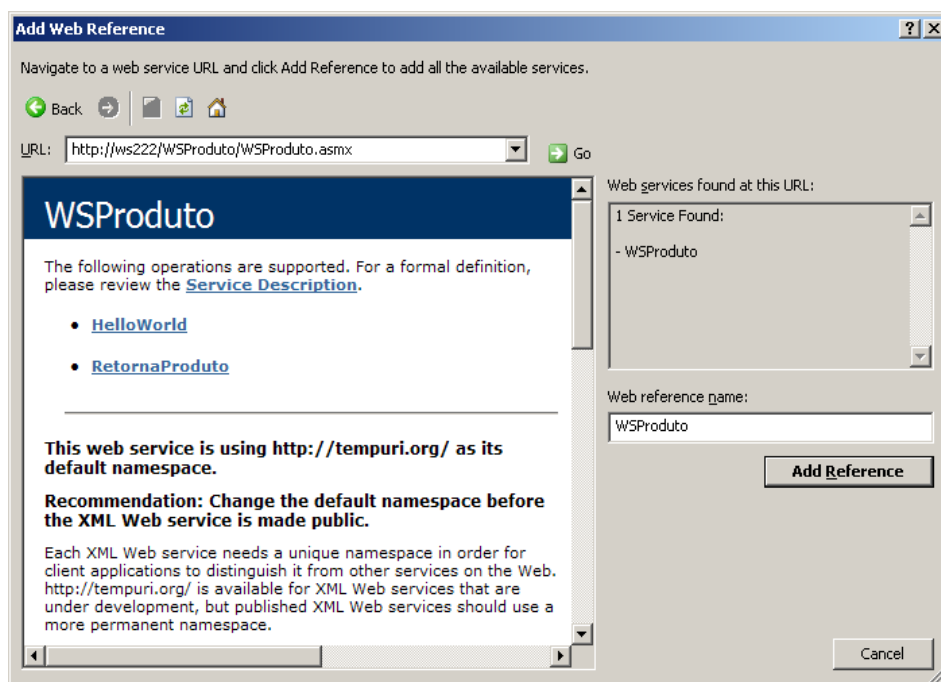


Figura 02

Defina um nome para o Proxi no campo "Web reference name" clicando em seguida no botão "Add Reference". Assim procedendo, conforme observado na Solution Explore do projeto (Figura 03) Obtivemos uma classe (Proxi) mapeando os métodos e o endereço dos mesmos para acesso ao WebService.

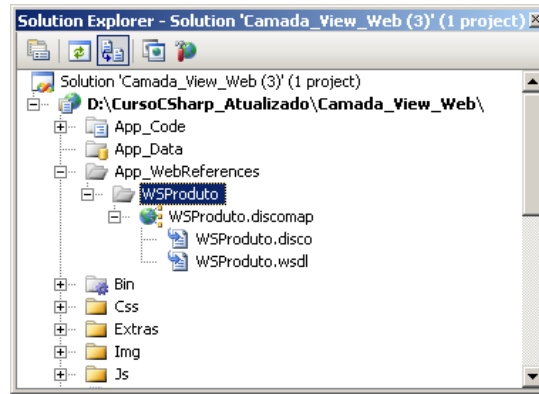


Figura 03

Passo 03

Finalizando esta etapa, observe a o código da Listagem 01 onde, em negrito, procedemos as alterações necessárias para utilização do método RetornaProduto do WebService. Basicamente criamos um objeto denominado WSPproduto que herda da classe (Proxi) WSPproduto, e em seguida fizemos uso do método referido, atribuindo o tipo de retorno (DataTable) a propriedade DataSource do GridView.

```
public partial class ConsultaProduto : System.Web.UI.Page
{
    //ProdutoController_SQL Prod = new ProdutoController_SQL();
    WSPproduto.WSPproduto WSPproduto = new WSPproduto.WSPproduto();

    protected void ButtonPesquisar_Click(object sender, EventArgs e)
    {
        //GridViewProdutos.DataSource = Prod.ObterProduto(TextBoxDescricaoProduto.Text);
        GridViewProdutos.DataSource =
        WSPproduto.RetornaProduto(TextBoxDescricaoProduto.Text);
        GridViewProdutos.DataBind();
    }
}
```

. . . Código restante da classe omitido aqui.

Gerando Relatórios com CrystalReport

O gerador de relatórios CrystalReport que utilizaremos aqui, é parte integrante do VisualStudio. Na prática, o produto, é distribuído (comercializado) em separado pela proprietária do produto, e nada impede que utilizamos o mesmo para desenvolver nossos modelos de relatórios.

Uma prática comum no desenvolvimento de relatórios com CrystalReport, é injetar instrução SQL para obter os dados junto a uma fonte de dados, e a partir desta instrução, em tempo de desenvolvimento, utilizar o IDE do produto para modelar os dados do relatório. Isto não faremos, pois desde o início de nosso projeto temos tido a devida atenção para não fugir ao padrão MVC, logo, neste caso, em se tratando o relatório de um componente da camada View, não poderíamos cometer o engando de tornarmos o mesmo dependente e acoplado a fonte de dados (Banco de Dados).

O que faremos aqui então? Vamos desenvolver o Lay-Out de nossos relatórios a partir dos atributos de nossa classes, e em seguida, utilizarmos os mesmos na camada View, atribuindo aos mesmos o resultado retornado pelos métodos de negócio que retornam coleções de dados por intermédio de um DataTable. Igualmente aos controles de dadosde coleção como GridView, ListBox entre outros, os relatórios gerados a partir do CrystalReport implementam a propriedade DataSource para qual atribuiremos os tais DataTable retornados pelos métodos de obtenção de registros.

Passo 01

Baseado no exposto na Figura 01, adicione um diretório na estrutura de diretórios do nosso projeto Camada_Model_SQL nomeando-o como RPT. Este diretório acomodará nossos modelos de relatórios gerados pelo CrystalReport. O resultado ficará semelhante ao representado na Figura 02.

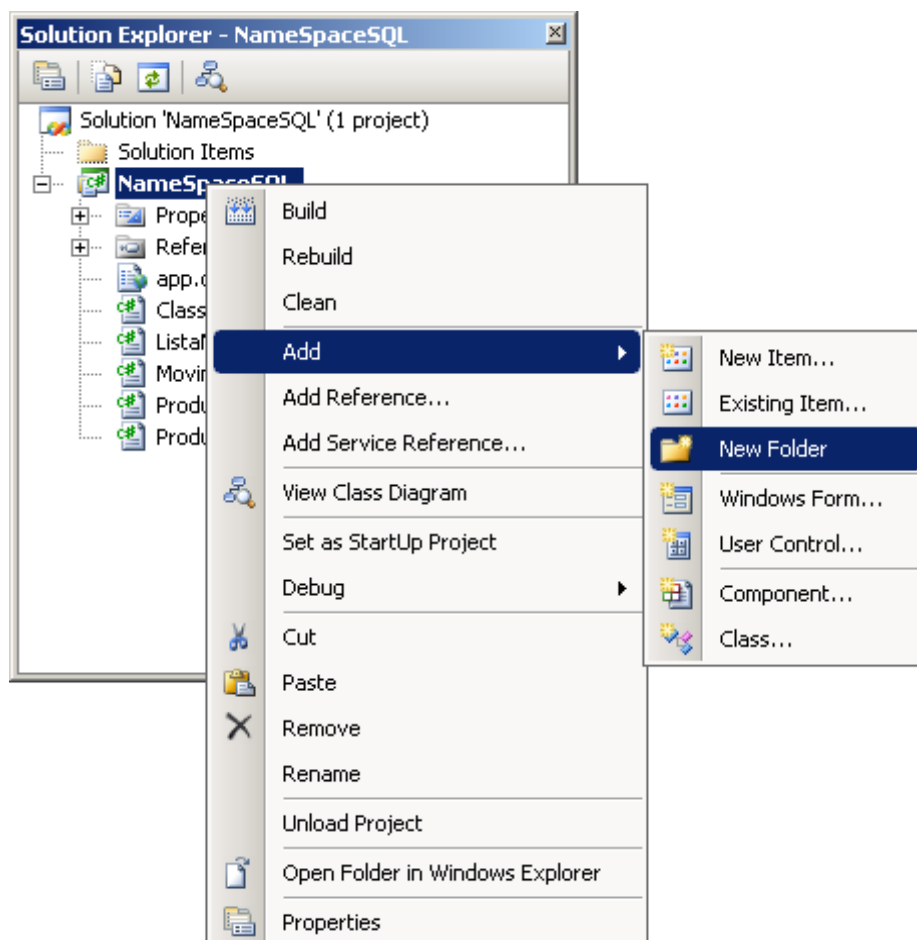


Figura 01

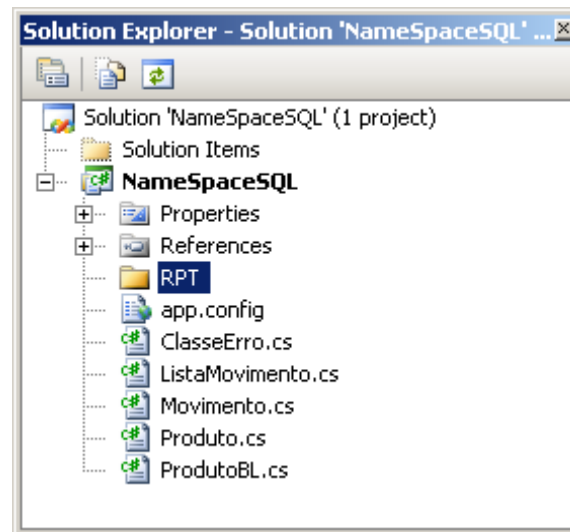


Figura 02

Passo 02

Orientando-se pela Figura 03, 04 e 05, adicione um novo item ao projeto, a saber, um relatório CrystalReport. Repare que optamos nesta fase por um relatório em branco, abrindo mão do ajudador do ambiente CrystalsalReport.

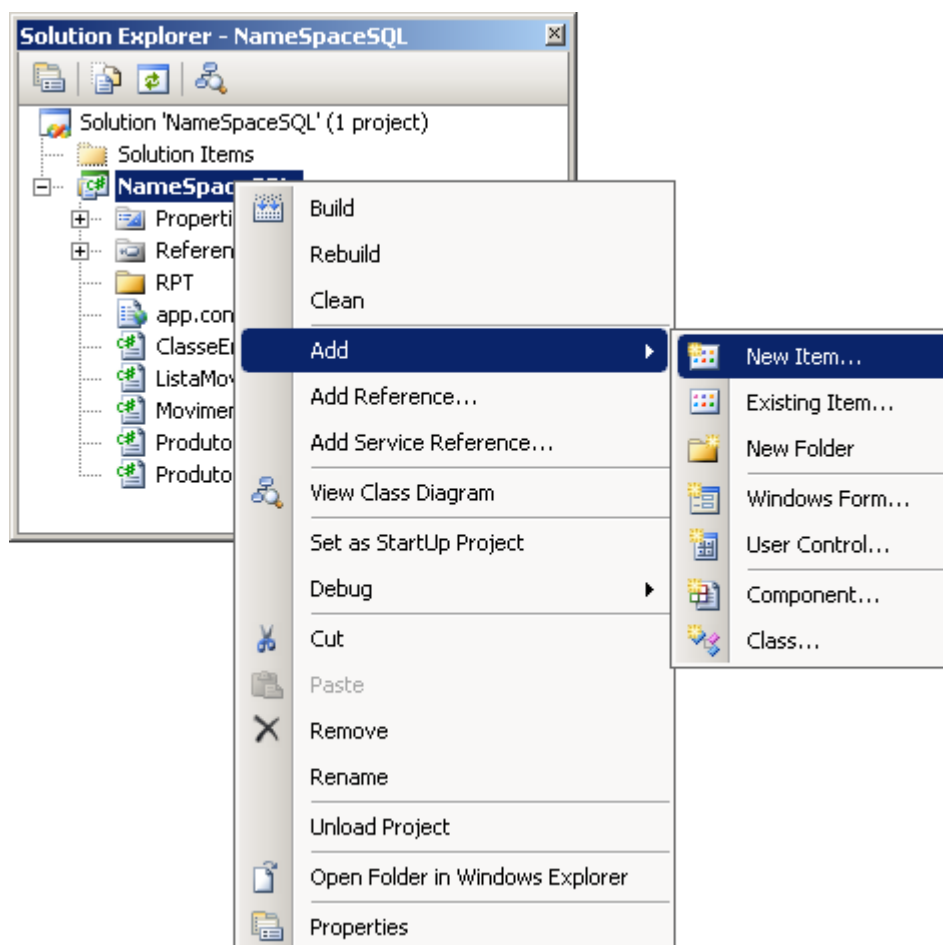


Figura 03

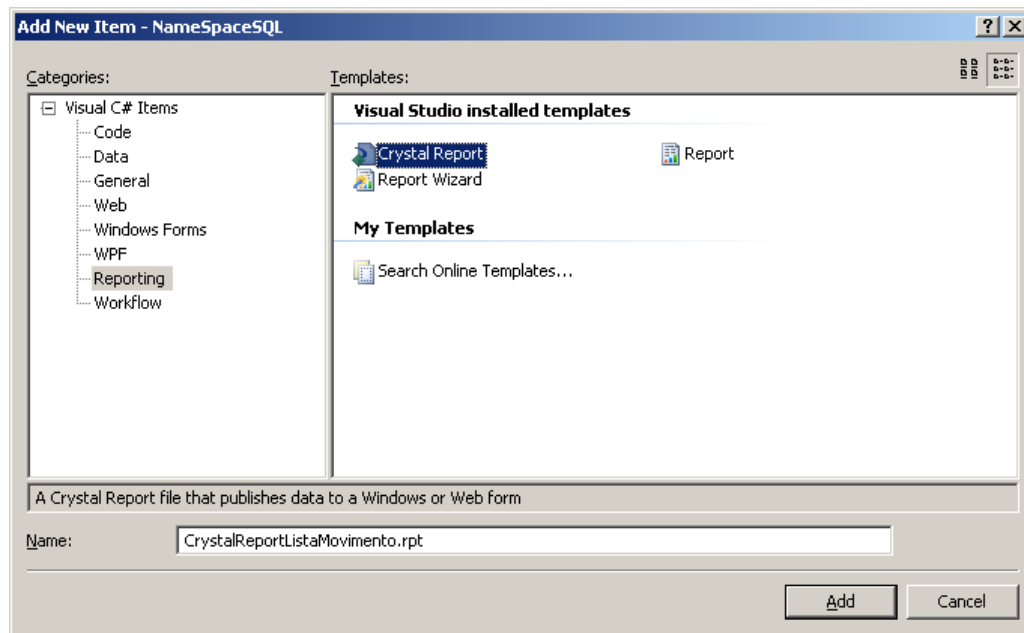


Figura 04

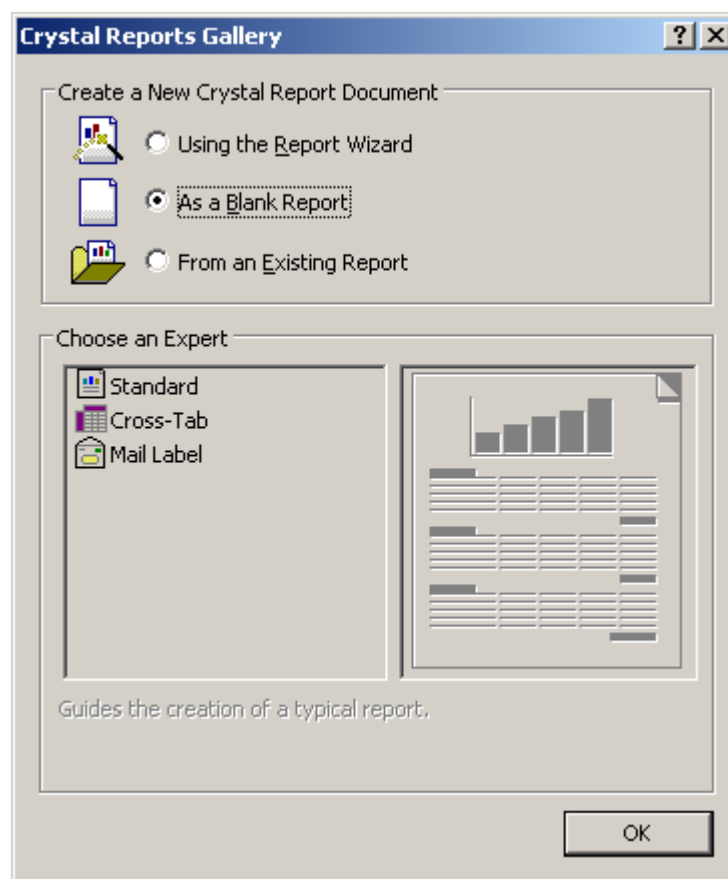


Figura 05

Passo 03

As Figuras 06 e 07, respectivamente exibem a área de Lay-Out do relatório e a paleta de componentes (Tollbox) onde podemos buscar controles visuais para texto e forma a ser aplicado a área de design. A área de Lay-Out como podemos observar, é dividida em Sessions que representam as bandas de um relatório.

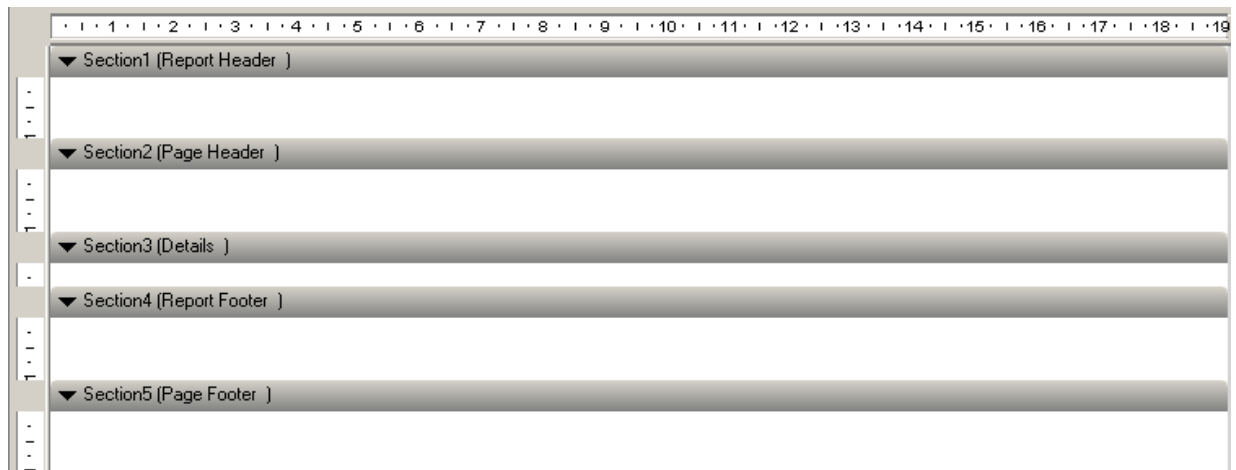


Figura 06

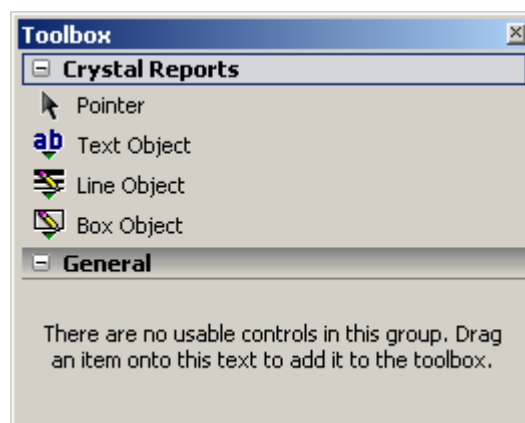


Figura 07

Passo 04

Iniciando o desenho de nosso relatório, no tocante a modelagem de dados a ser exibido, acesse a caixa de dialogo Field Explorer, precisamente no nó Database Fields, clique com o botão direito do mouse sobre este nó selecionando Database Expert entre as opções.

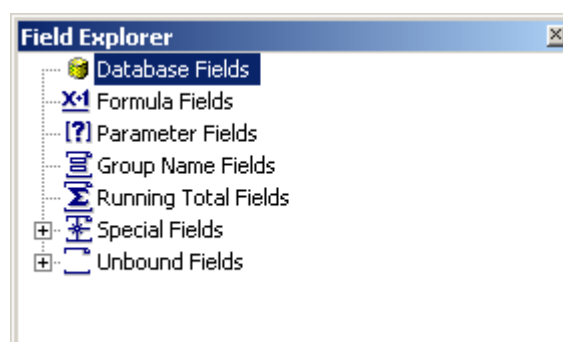


Figura 08

Passo 05

Na caixa de dialogo representada pela Figura 09, acesse o nó Net Objects dentro da estrutura de nós de Project Data. Assim, teremos acesso a todas as classes (modelos) do projeto, sendo que para nosso exemplo optaremos por ListaMovimento da NamespaceSQL. Por fim clique na seta (>) para selecionar o modelo que vai fornecer os campos do nosso relatório. Clique no botão OK para finalizar.

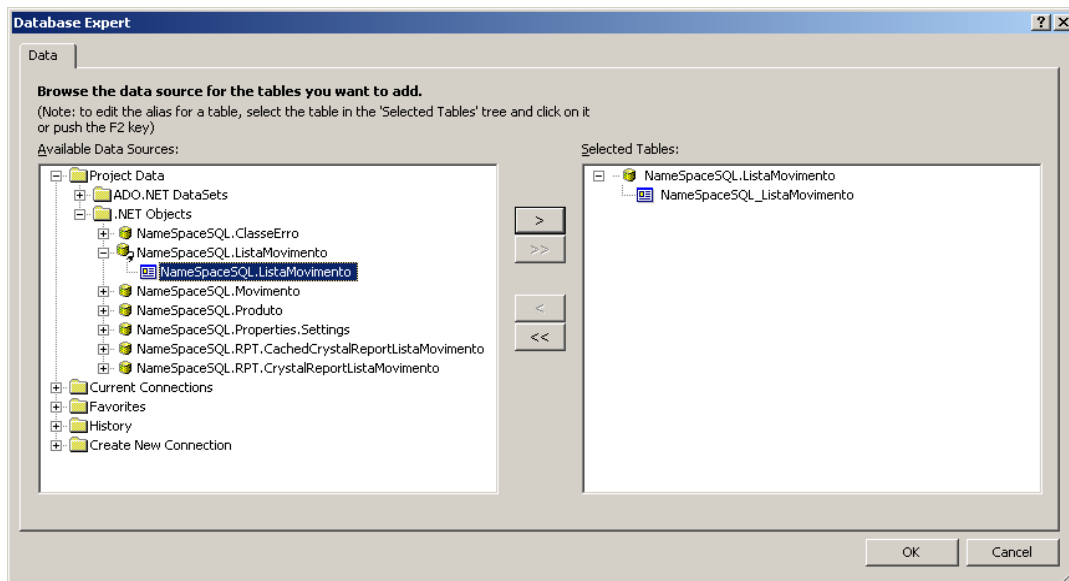


Figura 09

Passo 06

Agora em Field Explorer no nó Database Fields (Figura 10) temos acesso aos campos da classe selecionada, bastando arrastar, por exemplo os campos Data, Descricao_Movimento, Descricao_Produto e Quantidade para a banda Detail (Session3 Detail) para definir quais campos desejamos exibir no relatório.

Observe que quando arrastamos um campo para Session3 Detail, na banda superior, para cada campo adicionado, e colocado também um componente que serve de título para a coluna. As figuras 11 e 12 exibem o retratado.

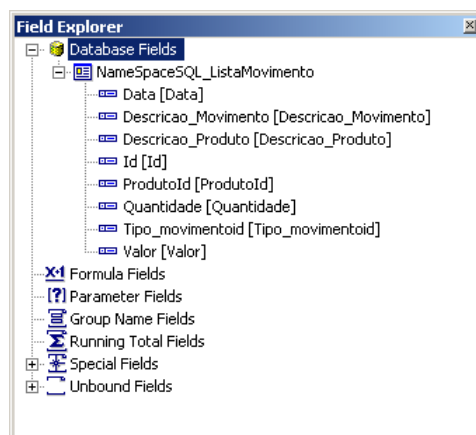


Figura 10

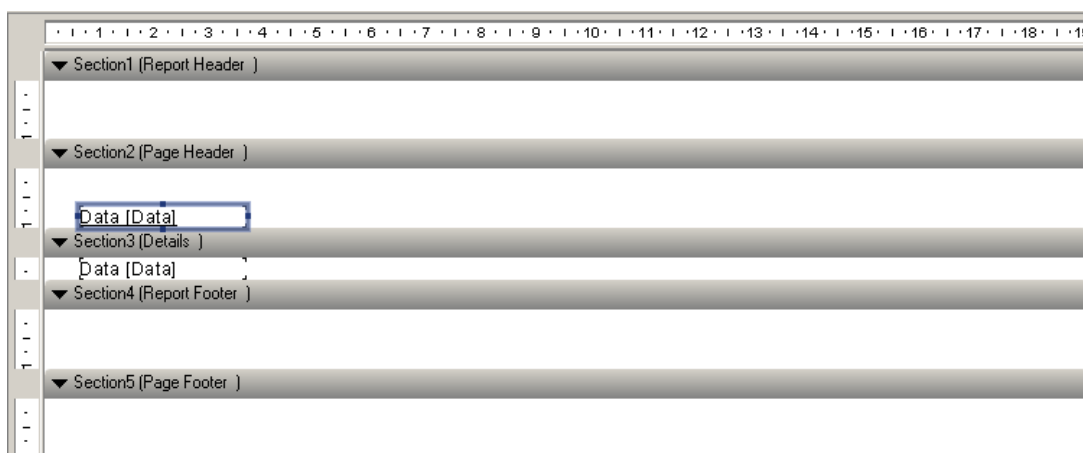


Figura 11

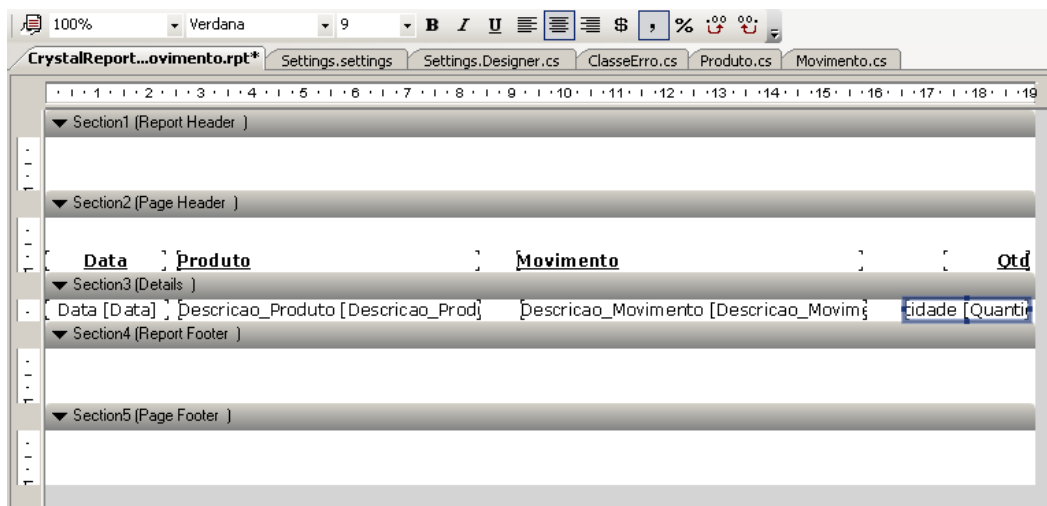


Figura 12

Passo 07

Vamos agora aplicaremos um controle Text Object oriundo da Tool Box (Figura 14), servindo o mesmo como título do relatório. Naturalmente aplicaremos tal controle na banda de cabeçalho (Session2 Page Header). Dê um duplo clique no controle para editá-lo, digitando o texto conforme imagem da Figura 15.

Ainda nesta etapa, aplique na banda cabeçalho um controle Print Date e um Print Time, disponíveis no nó Special Fields em Field Explore (Figura 14). Repare que nesta coleção temos várias variáveis úteis para o relatório.

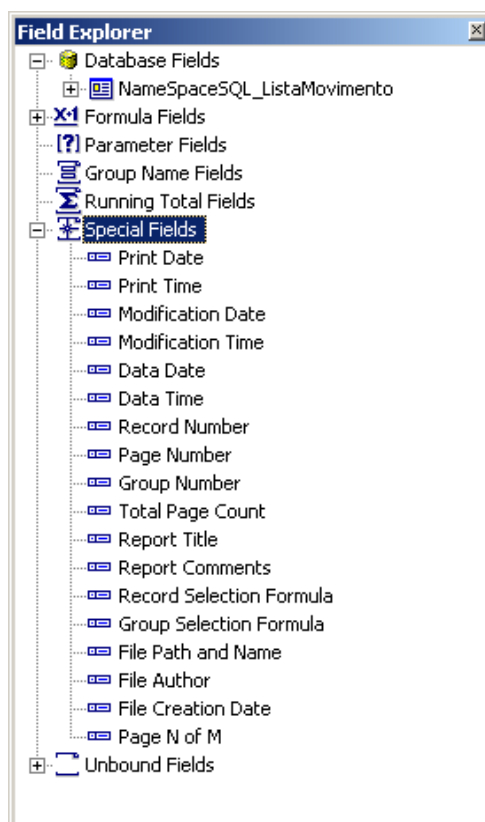


Figura 13

Para finalizar o design, aplique conforme ilustrado na Figura 15, duas linhas horizontais para visualmente separar os títulos das colunas dos dados da mesma. Aplique também o controle Page Number a banda rodapé, obtendo-o da coleção Special Fields em Field Explore (Figura 13).

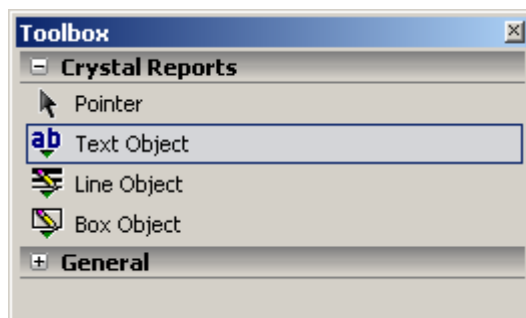


Figura 14

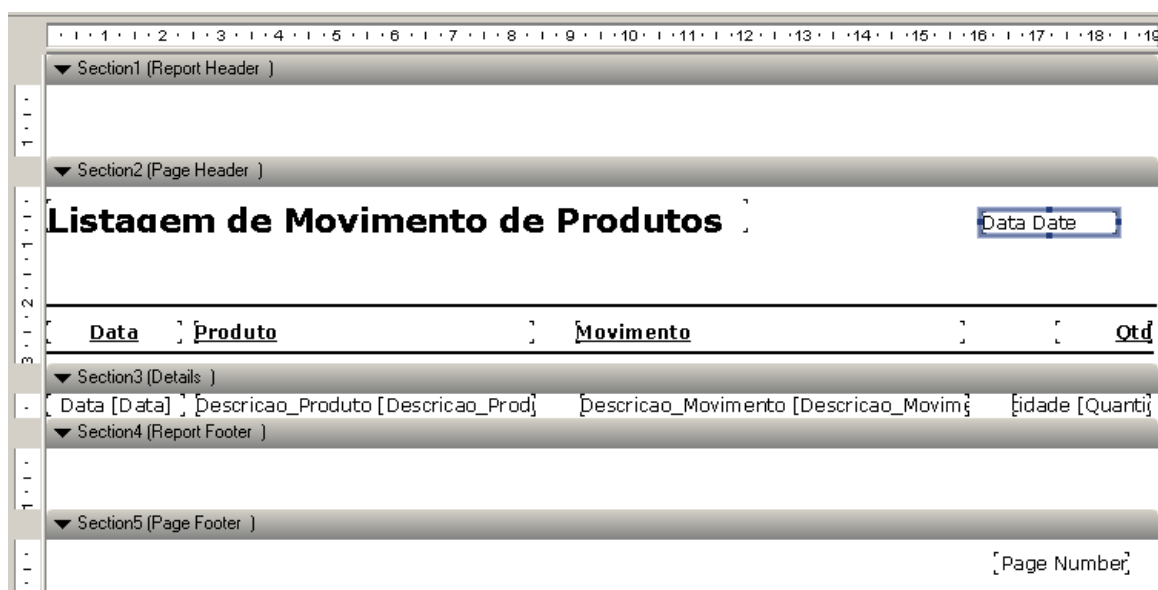


Figura 15

Passo 08

Clicando com o botão direito do mouse sobre a área de design, selecione no menu de contexto (Figura 16) a opção Preview Report para antes mesmo de atribuírmos dados ao relatório termos uma visão realista do design final. O resultado pode ser observado na Figura 17.

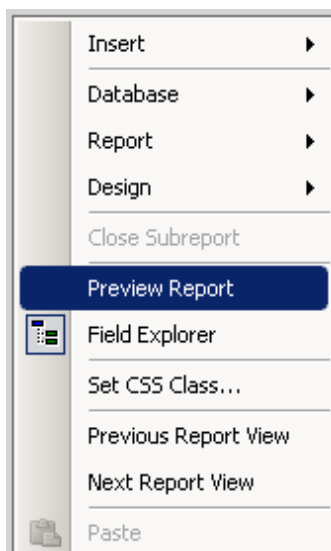


Figura 16

Listagem de Movimento de Produtos

07/08/2010

Data	Produto	Movimento	Otd
05/12/2010	Saturday	Aqua	217,95
01/20/2010	Friday	Teal	323,30
01/29/2010	Friday	Blue	143,38
02/02/2010	Tuesday	Aqua	7,28
05/03/2010	Sunday	Teal	15,57
12/31/2009	Tuesday	Teal	302,26
08/07/2010	Thursday	Fuchsia	154,84
08/01/2010	Friday	Aqua	211,37
01/19/2010	Thursday	White	79,95
04/13/2010	Monday	White	313,91

Figura 17

Passo 09

Agora daremos uma guinada extrema, saindo do projeto que estamos, e abrindo o projeto Web. Mas antes, temos que copiar o diretório RPT do projeto atual todos os arquivos do CrystalReport e incluímos no projeto Web, pois assim, no contexto do projeto Web (Camada View), poderemos utilizá-los. Proceda como sugerido na Figura 18 para visualizarmos o diretório RPT recém copiado para o projeto Web.

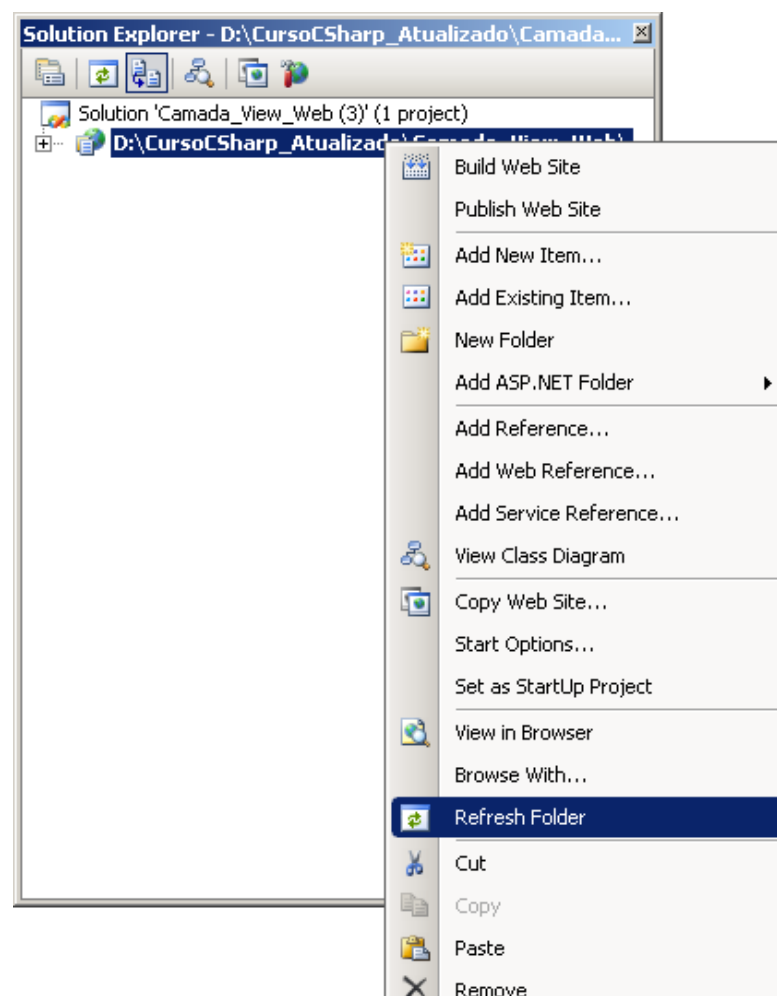


Figura 18

Passo 10

Adicione uma nova página Aspx ao projeto, nomeando-a como ExibeRelatorio.Aspx. Imediatamente após salvar este arquivo, aplique a ele um componente CrystalReportSource e um componente CrystalReportViewer. O primeiro controle, naturalmente, servirá para atribuímos um coleção de dados retornada por exemplo por um método de negócio que retorne um tipo DataTable.

Selecione o controle CrystalReportViewer, e no menu Task (canto superior direito do componente) e em Choose Report Source vincule-o ao controle CrystalReportSource, definindo o mesmo como fonte de dados do controle CrystalReportViewer.

Não é difícil concluir que o controle CrystalReportViewer servirá para prover o design final (quando em execução) dos relatórios no browser, acrescentando funcionalidades como: salvar em vários formatos, imprimir e zoom.

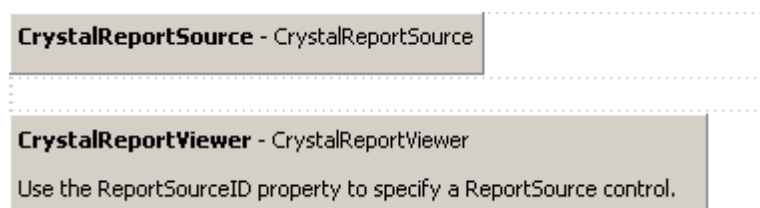


Figura 19

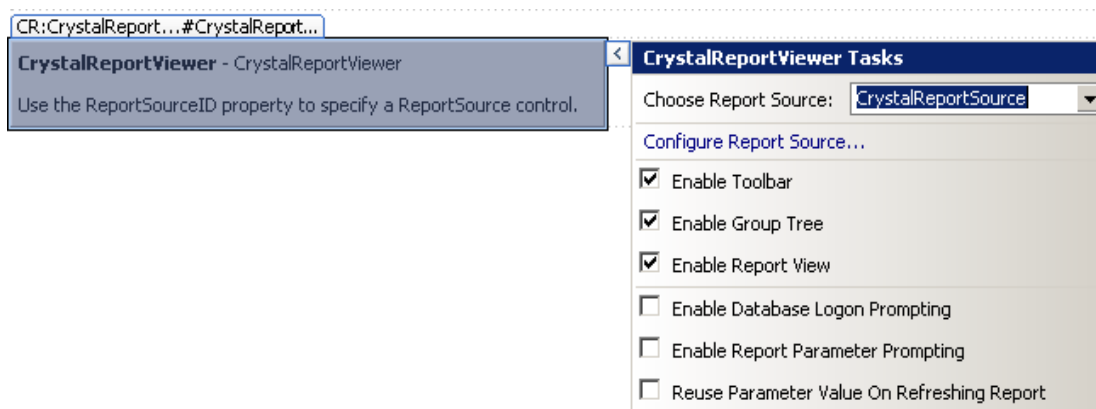


Figura 20

Passo 11

Acrescente, conforme Listagem 01, o procedimento para o evento Page_Load da página ExibeRelatorio.Aspx.

Listagem 01

```
1. protected void Page_Load(object sender, EventArgs e)
2. {
3.     CrystalReportSource.Report.FileName = (String) Session["NomeRelatorio"];
4.     CrystalReportSource.ReportDocument.SetDataSource((DataTable)Session["DataTableRelatorio"]);
5.     CrystalReportSource.DataBind();
6. }
```

Comentando o código fonte da Listagem 01

Linha	Descrição
3	Define o nome do relatório oriundo previamente atribuído a variável de sessão (Session).
4	Atribui ao método a coleção de dados (DataTable) previamente atribuído a variável de sessão.
5	Invoca o método DataBind do controle CrystalReportSource para renderizar o conteúdo do relatório.

Passo 12

Finalizando a codificação, observando a Listagem 02, aplique o código constante ao evento Click do botão Imprimir da Page ConsultaMovimento.Aspx. Caso não exista este botão em sua página, acrescente.

Listagem 02

```
1. protected void ButtonImprimir_Click(object sender, EventArgs e)
2. {
3.     Session["NomeRelatorio"] = "CrystalReportListaMovimento.rpt";
4.
5.     Session["DataTableRelatorio"] = Prod.ObterListaMovimento(TextBoxDescricaoProduto.Text,
6.                                                             DateTime.Parse(TextBoxDataIni.Text),
7.                                                             DateTime.Parse(TextBoxDataFin.Text)
8.                                                             );
9.
10.    Response.Redirect("ExibirRelatorio.aspx");
11. }
```

Comentando o código fonte da Listagem 02

Linha	Descrição
3	Atribui a variável de seção (Session) o nome do relatório que desejamos exibir..
5	Atribui a variável de seção (Session) o DataTable (coleção de registros) retornado pelo método.
10	Redireciona para a página ExibirRelatório.Aspx, naturalmente desejando ver os dados no browser.

Passo 13

Execute o projeto e verifique o resultado.

Anexo 01 - JavaScript Aplicado a Asp.Net

O ASP.NET foi uma grande revolução no desenvolvimento web, trouxe uma grande produtividade para todo o processo de desenvolvimento.

Mas o ASP.NET não mudou a arquitetura de desenvolvimento web. Assim como o ASP 3, o ASP.NET é uma tecnologia para desenvolvimento no lado do servidor, realizando acesso a dados e gerando o conteúdo a ser enviado para o client.

Mas em um sistema será necessária também a interatividade com o usuário, programando o lado client. Tarefas como exibir uma mensagem popup ou definir o foco de uma caixa apenas podem ser feitas com programação no lado client, o que continua sendo feito com javascript.

A programação javascript não mudou em nada. O que mudou foi a forma como o javascript é gerado em conjunto com a tecnologia do servidor. Torna-se necessário entender adequadamente a relação de execução entre o que está no servidor e o que está no lado client para ser possível gerar o javascript corretamente. Então vamos analisar em detalhes situações em que o javascript seja necessário e como ele pode ser gerado.

Inserindo eventos em webControls

O javascript permite que façamos a programação de eventos no lado client. Eventos como click, onmouseover, onmouseout, e outros, podem ser programados em diversos objetos da interface web, tal como botões, imagens, links e outros objetos.

Os eventos no javascript são inseridos na forma de atributos das tags. Através da propriedade

Atributes

podemos ter controle sobre os atributos de tags que serão renderizados no lado client, assim podemos adicionar eventos client aos objetos.

Por exemplo, se desejarmos adicionar um evento onclick em um botão podemos utilizar o seguinte código:

```
button1.attributes.add("onclick","alert('Esse botao foi clicado')")
```

Neste exemplo ao clicarmos no botão veremos uma mensagem popup antes que o postBack típico do botão ocorra, para finalmente rodar o evento click deste botão no servidor.

Cancelamento de um postBack

Conforme característica do próprio javascript, podemos utilizar o evento onclick para cancelar a ocorrência de um postBack, bastando retornar do evento um valor boolean.

Assim podemos, ao invés de fazer uso da função alert, utilizar a função confirm do javascript para gerar uma mensagem de confirmação em botões, por exemplo. Veja:

```
Button1.attributes.add("onclick","return confirm('Tem certeza ?')")
```

Gerando eventos no HTML

Para alguns eventos, podemos inseri-los diretamente na tag ao invés de fazer a inserção através do Attributes.

Mas apenas para alguns eventos. A regra é a seguinte : Se existe um evento de servidor com o mesmo nome então você não pode inserir o evento direto no HTML, pois será interpretado errado, será confundido com o evento do servidor. Por exemplo, o onclick do botão, que no servidor tem o mesmo nome.

Mas se não existe nenhum evento de servidor com o mesmo nome, fique a vontade, insira o evento na tag normalmente, o webcontrol, ao não reconhecer o atributo, renderizará o atributo no client exatamente como estiver e desta forma irá gerar o evento no javascript.

Um exemplo disso são os eventos `onmouseover` e `onmouseout` no objeto `Image`, veja como fica, neste exemplo preenchendo a linha de status do browser :

```
<asp:Image id="Image1" runat="server"
    ImageUrl="SNAG-0002.jpg" onmouseover="javascript:window.status='testando 1,2,3'"
    onmouseout="javascript:window.status=''">
</asp:Image>
```

Utilizando funções JavaScript

Seria por demais desagradável ter que inserir rotinas mais longas em eventos javascript utilizando apenas a propriedade `attributes`.

As regras e lógicas de programação javascript, porém, continuam valendo. Assim sendo, ao invés de inserir todo o código através do `attributes` podemos apenas inserir uma chamada de função através do `attributes`.

Assim sendo podemos criar funções diretamente no arquivo `.ASPX` e através do `Attributes` inserir apenas as chamadas dessas funções.

```
<script language=javascript>
    function Confirmar(){
        return(window.confirm("Tem certeza ?"))}
</script>
```

E no CodeBehind:

```
Button1.attributes.add("onclick","return Confirmar()")
```

Gerando javascript para o client

Existem alguns casos em que o código javascript deve ser executado imediatamente no load da página e não em um evento de um objeto específico.

Os dois casos mais comuns são a exibição de uma mensagem popup no client e a definição do foco para um determinado objeto.

Existem 2 métodos na classe `page` para a geração de código JavaScript, são os seguintes:

RegisterClientScriptBlock:

Gera o script imediatamente após a tag `form`

RegisterStartupScript:

Gera o script imediatamente antes do fechamento da tag `form` (`</form>`)

Essa sutil diferença tem impacto na geração do código client : com o `RegisterClientScriptBlock` garantimos que o script seja executado tão logo possível, mas deveremos ter cuidado para que o script não faça referência a objetos que ainda nem tiveram suas tags recebidas pelo browser, do contrário o script irá gerar erro.

Já com o `RegisterStartupScript` o script será executado apenas depois de todos os objetos terem chegado ao client, justamente devido a posição que o script é inserido na página.

Ambos os métodos recebem 2 parâmetros, o script a ser inserido dentro da página e um parâmetro chamado "key". O parâmetro `key` tem por objetivo impedir que um script seja duplicado dentro da página, o que é

especialmente importante durante o processo de criação de custom web controls.

Neste parâmetro, key, criamos uma espécie de nome para o script. Assim sendo se por algum motivo a instrução de geração do script for executada mais de uma vez, o ASP.NET se encarrega de ignorar as execuções duplicadas. São casos incomuns quando o script é gerado diretamente na página, mas um recurso fundamental para a criação de custom web controls.

Exibindo uma mensagem de conclusão:

```
Me.RegisterClientScriptBlock("x", "<script>alert('Operação concluída !')</script>")
```

Definindo o foco para uma caixa:

```
Me.RegisterStartupScript("z", "<script>document.all.TextBox1.focus()</script>")
```

Cuidados com os nomes dos objetos

No exemplo acima utilizamos o nome do objeto - TextBox1 - para poder definir o foco para o objeto. Mas nem sempre o nome que o objeto possui no servidor será o mesmo nome que o objeto possui no client, os nomes poderão ser diferentes especialmente quando o objeto estiver dentro de algum container, tal como uma DataGrid, um Repeater ou outro objeto do gênero.

Para resolver este problema os objetos possuem uma propriedade chamada clientID. Esta propriedade guarda o nome que o objeto terá quando estiver no client. Assim sendo, mudando o código acima, para termos maior segurança com relação ao nome do objeto, o código fica da seguinte forma :

```
Me.RegisterStartupScript("z", "<script>document.all." & TextBox1.ClientID & ".focus()</script>")
```

Também é possível ter expressões de binding dentro de blocos de javascript, quer seja para a tradução do nome de um objeto ou para algum outro objetivo. Veja :

```
<script language="javascript">
    function DefinirFoco(){
        document.all.<%# Textbox1.clientid %>.focus()}
</script>
```

Pode-se então utilizar a instrução Page.DataBind para provocar o processamento da expressão e desta forma gerar o id do objeto na posição correta.

Manipulando janelas com JavaScript

Manipular janelas popup é outro recurso para o qual precisamos utilizar amplamente o javascript.

Em um sistema é frequente a necessidade de criarmos janelas popup modais para perguntarmos informações adicionais ao usuário. Mais uma vez precisamos de javascript para isso.

É importante destacar que não podemos utilizar response.redirect ou server.transfer e indicar em qual janela ou frame a página será aberta. Os dois primeiros são instruções do servidor e portanto não tem como controlar a atividade no client.

Abrindo um popup

Essa primeira parte é simples. Podemos utilizar o window.open do javascript normalmente. Podemos inseri-lo diretamente no ASPX ou utilizar uma das instruções Register para fazer a geração do script com o window.open

Utilizando uma janela modal

Neste caso a tarefa complica um pouco mais, pois em geral quando utilizamos uma janela modal desejamos uma resposta, um resultado. Assim sendo precisamos realizar uma comunicação entre as janelas utilizando javascript

Vamos supor um WebForm1.aspx chamando em uma janela modal um Webform2.aspx.

```
<script language="javascript">
    function abrirjanela()
    {
        var ret;
        ret=window.showModalDialog("webform2.aspx","", "");
        document.all.TextBox1.value=ret;
    }
</script>
```

Neste exemplo temos uma função abrirjanela que deverá estar no webForm1.aspx. Esta função faz a abertura de uma janela modal, obtém um valor de resposta e atribui esse valor de resposta na TextBox1.

Esta função pode ser disparada de diversas formas diferentes : utilizando um hyperlink, o click de uma imagem, um botão html, enfim, objetos client.

Porém só é justificável usar um objeto de servidor para fazer esse disparo se for necessário realizar algum processamento adicional antes da janela popup. Nesse caso podemos fazer o disparo da função no click de um botão no servidor (não "fazer o disparo", mas sim gerar o javascript para isso).

```
Me.RegisterStartupScript("x", "<script>abrirjanela()</script>")
```

No WebForm2 precisaremos definir o valor de retorno através da propriedade returnValue do objeto window e fechar o form.

```
<script language="javascript">
    function terminou()
    {
        window.returnValue=document.all.TextBox1.value;
        window.close();
    }
</script>
```

Mais uma vez temos a mesma questão : Só é justificável usar objetos de servidor se houver algum processamento adicional a ser realizado antes do fechamento. A forma de fazer o disparo, neste caso, fica idêntica a anterior :

```
Me.RegisterStartupScript("x", "<script>terminou()</script>")
```

Detalhes adicionais : Funções javascript podem ser inseridas em arquivos .JS e estes vinculados a página através da própria tag <script>.

O trabalho entre frames também precisa ser feito em javascript. Deve-se ter cuidado com o autoPostBack, que insere código javascript em alguns eventos do objeto. Se ao mesmo tempo você tentar inserir seu próprio código javascript para o mesmo evento, gerará erro.

O evento onSubmit do form é um evento especial, pois todos os webControls podem manipulá-lo. Por isso você não deve tentar lidar diretamente com esse evento. Para isso existe um método chamado RegisterOnSubmitStatement que irá controlar o uso do evento OnSubmit por todos os webControls.

Anexo 2 - Expressões Regulares

O que é Expressão Regular?

Em ciência da computação, uma expressão regular (ou o estrangeirismo *regex*, abreviação do inglês *regular expression*) provê uma forma concisa e flexível de identificar cadeias de caracteres de interesse, como caracteres particulares, palavras ou padrões de caracteres. Expressões regulares são escritas numa linguagem formal que pode ser interpretada por um processador de expressão regular, um programa que ou serve um gerador de analisador sintático ou examina o texto e identifica partes que casam com a especificação dada.

O termo deriva do trabalho do matemático norte-americano Stephen Cole Kleene, que desenvolveu as expressões regulares como uma notação ao que ele chamava de álgebra de conjuntos regulares. Seu trabalho serviu de base para os primeiros algoritmos computacionais de busca, e depois para algumas das mais antigas ferramentas de tratamento de texto da plataforma Unix.

O uso atual de expressões regulares inclui procura e substituição de texto em editores de texto e linguagens de programação, validação de formatos de texto (validação de protocolos ou formatos digitais), realce de sintaxe e filtragem de informação.

Expressões Regulares em JavaScript

Expressão Regular (ou Regular Expression, que é o nome original) é uma forma de efetuar a validação sobre um texto através de um determinado padrão, eliminando um longo trecho de código para fazer esta função.

Uma Regular Expression é composta de uma seqüência de símbolos e caracteres especiais que é interpretada como uma regra, que indicará se um texto posterior segue todas as condições nela impostas.

Mas e como isso funciona? Vamos a um exemplo prático: validação de e-mail.

```
function validaEmail(mail){
var er = RegExp(/^([A-Za-z0-9_\-\.]+@[A-Za-z0-9_\-\.]{2,}\.[A-Za-z0-9]{2,}(\.[A-Za-z0-9])?)/);
if(mail == ""){
    window.alert("Informe seu e-mail!");
}else if(er.test(mail) == false){
    window.alert("E-mail inválido!");
}
}
```

Listagem 01 – Função contendo um exemplo de Expressão Regular (linha 2)

Vamos entender o código. Na segunda linha temos a declaração da Regular Expression escrita em uma das duas formas possíveis. Pode-se utilizar o método construtor `RegExp`, assim como no exemplo, ou atribuir diretamente a expressão (quadro 02).

- `var er = /^[A-Za-z0-9_\-\.]+@[A-Za-z0-9_\-\.]{2,}\.[A-Za-z0-9]{2,}(\.[A-Za-z0-9])?/;`

Listagem 02 – Expressão regular sem `RegExp`

Mas qual é a diferença? Da primeira forma você pode-se atribuir a expressão regular dinamicamente, passando uma string por parâmetro, por exemplo. Na segunda forma, o conteúdo da expressão é estático, sendo definido no código somente.

E o que significa aquela sopa de letrinhas? Vamos por partes:

Primeira parte: `^[A-Za-z0-9_\-\.]+`: os caracteres devem estar entre A e Z (apenas maiúsculo), entre a e z

Segunda parte: `@`: após a primeira seqüência de caracteres deve encontrar uma arroba (`@`).

Terceira parte: `[A-Za-z0-9_\-\.]{2,}`: deve encontrar duas ou mais seqüências de caracteres que satisfaçam a condição de ser alfanumérico, underline, traço ou ponto.

Quarta parte: `\.[A-Za-z0-9]{2,}`: deve encontrar duas ou mais seqüências de caracteres alfanuméricos antecidos por ponto.

Quinta parte: `(\.[A-Za-z0-9])?`: pode encontrar ou não uma seqüência contendo os caracteres que estão entre os parênteses.

Por fim, a função `test` irá validar a expressão, retornando um booleano que irá indicar se a string passada por parâmetro está validada dentro da expressão.

Para entender melhor, a Tabela 01 traz os sinalizadores utilizados nas expressões regulares e seus respectivos significados.

Tabela 01: Sinalizadores de expressões regulares

Sinalizador	Descrição	Exemplo
Sinalizadores		
i	Não faz distinção entre letras maiúsculas e minúsculas	/Java/i encontrará tanto "java" quanto "Java" e "JAVA"
g	Pesquisa global de todas as ocorrências de um padrão	/ca/g encontrará as duas ocorrências de "ca" na frase "É cada macaco no seu galho"
gi	Pesquisa global, sem distinção entre letras maiúsculas e minúsculas	/ca/g encontrará as duas ocorrências de "ca" na frase "Cada macaco no seu galho"
Caracteres literais		
\n	Encontra um caractere de quebra de linha.	
\r	Encontra um caractere de retorno de carro.	
\t	Encontra um caractere de tabulação horizontal.	
\v	Encontra um caractere de tabulação vertical.	
\f	Encontra um indicador de quebra de página.	
\xxx	Encontra o caractere ASCII expresso pelo valor octal xxx.	"\50" encontra o caractere de abertura de parenteses "("
\xdd	Encontra o caractere ASCII expresso pelo valor hexadecimal dd.	"\x28" encontra o caractere de abertura de parenteses "("
\uxxxx	Encontra o caractere ASCII expresso pelo valor UNICODE xxxx.	"\u00A3" encontra "£"
Posições dos itens de pesquisa		
^	Encontra somente no início da string	/^The/ encontra "The" in "The night" mas não encontra em "In The Night"
\$	Encontra somente no final da string.	/and\$/ encontra "and" em "Land" mas não encontra em "landing"
\b	Encontra nos extremos de uma palavra (os caracteres de teste devem existir no início ou fim de uma palavra pertencente à string).	/ly\b/ encontra "ly" em "This is really cool."
\B	Encontra no meio de uma palavra da string, ou seja, o inverso de \b.	/\Bor/ encontra "or" em "normal" mas não encontra em "origami."
Classes de caracteres		

[xyz]	Encontra qualquer caractere pertencente ao grupo de caracteres dentro dos colchetes. Você pode usar o hífen para definir uma faixa. Por exemplo: /[a-z]/ encontra qualquer letra do alfabeto, /[0-9]/ encontra qualquer dígito de 0 a 9.	/[AN]BC/ encontra "ABC" e "NBC" mas não encontra "BBC" uma vez que o "B" fora dos colchetes não faz parte do conjunto.
[^xyz]	Encontra qualquer caractere que não esteja no conjunto entre colchetes. O símbolo ^ nos colchetes indica negação. Obs: Não confunda o símbolo ^ (negação) dentro dos colchetes com o símbolo ^ que indica que o padrão deve ser encontrado no início da string. A negação deve ser usada somente dentro de colchetes.	/[^AN]BC/ encontra "BBC" mas não encontra "ABC" ou "NBC".
.	(ponto). Encontra qualquer caractere exceto quebras de linha ou outro caractere UNICODE que significa término de linha.	/b.t/ encontra "bat", "bit", "bet" e assim por diante.
\w	Encontra qualquer caractere alfanumérico incluindo o sublinhado. Equivale a [a-zA-Z0-9_].	/\w/ encontra "200" em "200%"
\W	Encontra qualquer caractere que não esteja no conjunto [a-zA-Z0-9_]. Equivale a [^a-zA-Z0-9_].	/\W/ encontra "%" em "200%"
\d	Encontra qualquer dígito de 0 a 9. Equivale a [0-9].	
\D	Encontra qualquer caractere que não seja um dígito. Equivale a [^0-9].	/\D/ encontra "No" em "No 342222"
\s	Encontra um caractere que provoca um espaço. Equivale a [\t\r\n\v\f].	
\S	Encontra qualquer caractere que não provoque um espaço. Equivale a [^\t\r\n\v\f].	
Padrões de repetições		
{x}	Encontra exatamente x ocorrências de um padrão.	/\d{5}/ encontra 5 dígitos.
{x,}	Encontra x ou mais ocorrências de um padrão.	/\s{2,}/ encontra no mínimo 2 caracteres de espaço.
{x,y}	Encontra de x até y ocorrências de um padrão.	/\d{2,4}/ encontra no mínimo 2 mas não mais que 4 dígitos.
?	Encontra zero ou uma ocorrência. Equivale a {0,1}.	/a?s?b/ encontra "ab" ou "a b".
*	Encontra zero ou mais ocorrências. Equivale a {0,}.	/we*/ encontra "w" em "why" e "wee" em "between", mas não encontra em "bad".
+	Encontra uma ou mais ocorrências. Equivale a {1,}.	/fe+d/ encontra tanto "fed" quanto "feed"
Agrupamento de padrões		
()	Agrupa caracteres para criar uma cláusula. Pode ser aninhado.	/(abc)+(def)/ encontra uma ou mais ocorrências de "abc" seguida(s) por uma ocorrência de "def".
	Oferece alternativa para o padrão. É similar a uma instrução "OU".	/(ab) (cd) (ef)/ encontra "ab" ou "cd" ou "ef".

Existem alguns métodos para efetuar a validação e outras operações tendo como base a expressão regular construída, os quais estão descritos na Tabela 02.

Tabela 02: Métodos de validação em expressões regulares

Método	Descrição
exec	Executa pesquisas em uma string e retorna um array das informações obtidas.
test	Indica se a string satisfaz os parâmetros da expressão regular.
match	Executa uma pesquisa em uma string e retorna um array das informações obtidas ou nulo se nada for encontrado.
search	Faz uma pesquisa na string e retorna o índice do resultado, ou -1 se não for encontrado.
replace	Executa uma pesquisa em uma string e substitui o resultado encontrado por uma substring.
split	Quebra uma string em um array de substrings.
lastIndex	Armazena a posição da última pesquisa bem sucedida realizada na string. Se nada for encontrado, a propriedade lastIndex será colocada como -1.
\$n	n representa o número de 1 a 9 Armazena os nove mais recentes pesquisa dentro de parênteses encontradas. Por exemplo, se o padrão usado por uma regular expression para a última pesquisa foi / (Hello)(\s+)(world)/ e a string pesquisada foi "Hello world" o conteúdo de RegExp.\$2 poderia ser qualquer caractere entre "Hello" e "world".
source	Armazena a cópia do padrão da regular expression.

Sabendo utilizar os caracteres especiais nas expressões regulares já é 50%, os outros 50% ficam por conta da lógica para montar a expressão.

Para fixar bem o assunto vamos a mais um exemplo: validação de datas. Você já deve ter visto pela Internet alguns scripts "medonhos", com algumas dezenas ou centenas de linhas de código para validar datas, aqueles que você olha e nem sabe direito como funciona. Vamos simplificar um pouco isso, seguindo o código do quadro 03.

```
function validaData(data){
    var er = RegExp("(0[1-9]|[012][0-9]|3[01])/(0[1-9]|1[012])/[12][0-9]{3}");
    if(er.test(data)){
        var barras = data.split("/");
        var dia = barras[0];
        var mes = barras[1];
        var ano = barras[2];
        var d = new Date(ano, mes-1, dia);
        if(dia != d.getDate()){
            return false;
        } else if(mes != (d.getMonth() + 1)){
            return false;
        } else if(ano != d.getFullYear()){
            return false;
        } else{
            return true;
        }
    } else{
        return false;
    }
}
```

Listagem 03 – Validação de datas com Expressão Regular.

Na segunda linha temos a declaração da nossa expressão regular, vamos entender como ela funciona.

Primeira parte: `(0[1-9]|[012][0-9]|3[01])`: valida os dias indicando que expressão irá considerar como válido tudo o que se iniciar com 0 e vier seguido de um dígito entre 1 e 9, ou (pipe `|`) é o operador lógico or) iniciar com 2 ou 3 e vier seguido de um dígito entre 0 e 9, ou então iniciar com 3 e vier seguido por 0 ou 1. Isto irá validar os dias entre 1 e 31.

Segunda parte: `(0[1-9]|1[012])`: agora a validação dos meses, que deve ser considerado os que iniciarem com zero, seguidos por um dígito entre 1 e 9, ou iniciar com 1, seguido de 0, 1 ou 2.

Terceira parte: `[12][0-9]{3}`: aqui irá validar o ano, que deverá iniciar com 1 ou 2 e após isso ter uma sequência de 3 dígitos entre 0 e 9, desta forma, valida os anos entre 1000 e 2999.

As barras indicam o formato da data, que deve estar como `dd/MM/aaaa`. Mas não se pode validar tudo apenas com expressão regular, pois temos os anos bissextos e os meses com 30 ou 31 dias. Por isso temos a segunda parte, que irá criar uma variável do tipo `data` passando os parâmetros de dia, mês e ano e depois verificar se cada parte da data criada confere com a data informada.

A questão de informar `mes-1` e depois comparar o mês com `d.getMonth() + 1` se dá pelo motivo do JavaScript tratar o mês iniciando de 0, e não por 1, ou seja, o JavaScript conta os meses de 0 a 11.

Enfim, Expressões Regulares se resume basicamente a isto, não é muito complicado, basta dar uma olhada mais atentamente para entender.