

Carlos Olavo de Azevedo Camacho Júnior

Desenvolvimento em Camadas com C# .NET



N.Cham. 005.133 C172d 2008
Autor: Camacho Junior, Carlos Olavo de
Título: Desenvolvimento em camadas com C# .NET



171231
Ex.3 CESUMAR BC

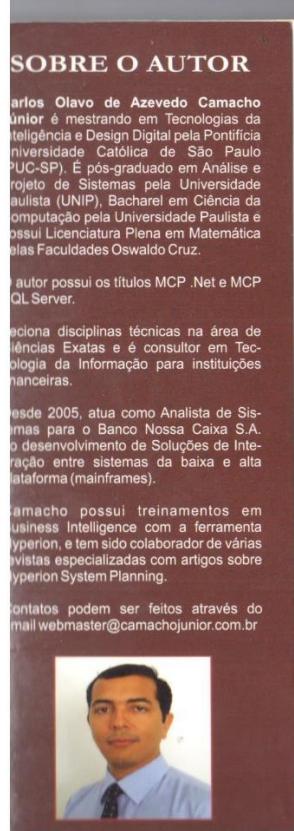
Ac. 67009

Completo

Construa Aplicações Web com Qualidade

■ Construa Aplicações Seguras com Criptografia de Dados

Visual Books



SOBRE O AUTOR

Carlos Olavo de Azevedo Camacho Junior é mestrado em Tecnologias da Inteligência e Design Digital pela Pontifícia Universidade Católica de São Paulo (PUC-SP). É pós-graduado em Análise e Projeto de Sistemas pela Universidade Paulista (UNIP), Bacharel em Ciência da Computação pela Universidade Paulista e possui Licenciatura Plena em Matemática pelas Faculdades Oswaldo Cruz.

O autor possui os títulos MCP .Net e MCP SQL Server.

Professora disciplinas técnicas na área de Ciências Exatas e é consultor em Tecnologia da Informação para instituições financeiras.

Desde 2005, atua como Analista de Sistemas para o Banco Nossa Caixa S.A. no desenvolvimento de Soluções de Integração entre sistemas da baixa e alta plataforma (mainframes).

Camacho possui treinamentos em Business Intelligence com a ferramenta Hyperion, e tem sido colaborador de várias revistas especializadas com artigos sobre Hyperion System Planning.

Contatos podem ser feitos através do e-mail webmaster@camachojunior.com.br



Av. Guedner, 1610 - Maringá - PR
Fone/Fax: (44) 3269-1541

Desenvolvimento em Camadas com C# .Net

Carlos Olavo de Azevedo Camacho Júnior

**Desenvolvimento em
Camadas com
C# .Net**

Visual Books

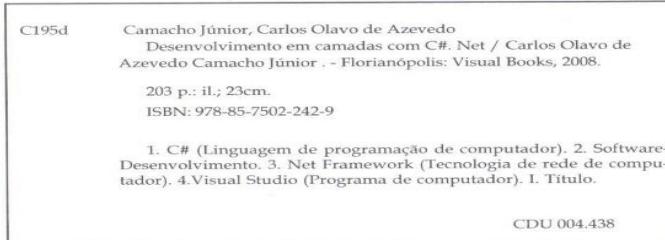
Copyright© 2008 by Carlos Olavo de Azevedo Camacho Júnior
Copyright© 2008 by Visual Books Editora

Nenhuma parte desta publicação poderá ser reproduzida sem autorização prévia e escrita da Visual Books Editora. Este livro publica nomes comerciais e marcas registradas de produtos pertencentes a diversas companhias. O editor utiliza essas marcas somente para fins editoriais e em benefício dos proprietários das marcas, sem nenhuma intenção de atingir seus direitos.

Novembro de 2008

Editora Responsável: Laura Carvalho Generini de Oliveira
Design da Capa: Júlio Cesar Winck (juliowinck@gmail.com)
Diagramação/Design: Visual Books Editora
Revisão: Visual Books Editora

Dados Internacionais de Catalogação na Publicação (CIP)
(Michele Beck Schröer - CRB 14/1059)



Direitos reservados por:
Visual Books Editora Ltda.
Rua Tenente Silveira, 209, sl 4 - Centro
Florianópolis - SC - 88.010-300
Tel: (48) 3222-1125 Fax: (48) 3324-2886

E-mail: info@visualbooks.com.br
Atendimento ao cliente: sac@visualbooks.com.br
HomePage: www.visualbooks.com.br

Dedicado à memória de Danilo Hessel Reimberg.

A Deus, pela vida.
À minha esposa Catia, pelo amor e companheirismo.
Aos meus pais Carlos e Vanda, pelo amor incondicional.
Aos meus irmãos Henrique e Alline, pelo amor, carinho, amizade e incentivo de sempre.
Aos meus sogros Benedito e Maria, por acolherem-me como um filho desde que me conheceram.
Ao amigo Nelson Paz, pelas valiosas conversas sobre tecnologia e pela revisão deste livro.
Aos amigos da PUC-SP, em especial aos professores Jorge Vieira e Sonia Allegretti, por mostrarem-me novos horizontes.
Ao amigo Milton Maruyama e toda a equipe do Banco Nossa Caixa S.A., pelo incentivo e por acreditar nas minhas idéias.
À Laura, Michele, Nilton e toda a equipe da Editora VisualBooks que muito colaboraram neste trabalho.
A todos que ajudaram direta ou indiretamente para a realização desta obra.
A você leitor, por crer que este livro pode alavancar seus conhecimentos em desenvolvimento de sistemas.

Sumário

Introdução	11
Requisitos Técnicos	13
1 Criando a Infra-estrutura de Banco de Dados	15
1.1 Criando a Tabela de Clientes	17
1.2 Executando o Script para a Criação da Tabela de Produtos	20
1.3 Criando a Tabela para Armazenar as Vendas	23
1.4 Criando Stored Procedures	25
1.5 Observações Importantes	27
2 A Camada de Acesso a Dados	31
2.1 Criando o Projeto	32
2.2 Criando o Diagrama de Classes	45
3 O Projeto DAL	49
3.1 Implementando as Classes da Camada de Acesso a Dados	51
3.1.1 Classe ClientesDAL.cs	51
3.1.2 Classe ProdutosDAL.cs	67
3.1.3 Classe VendasDAL.cs	71
4 A Camada de Regras de Negócio BLL	77
4.1 Implementando as Classes da Camada BLL	77
4.1.1 Classe ClientesBLL.cs	81
4.1.2 Classe ProdutosBLL.cs	83
4.1.3 Classe VendasBLL.cs	86
5 Formulário de Clientes	89
5.1 Criando a Interface com o Usuário	89
5.2 Adicionando Referência aos Projetos	90
5.3 Propriedades do Objeto	94
5.4 Criando Formulário de Clientes	98
6 Formulário de Produtos	115
6.1 Trabalhando com os Objetos do Formulário	116
7 Formulário de Vendas	127

8 Formulário de Produtos em Falta	133
8.1 Ocorrendo Erro	136
9 Dicas de Interface Web com Qualidade	139
9.1 Criando o Website UIWeb	139
9.2 Criando Formulário de Clientes	147
9.3 Definindo a Webpage Inicial	151
9.4 Implementando a Inclusão no Formulário de Clientes	154
10 Tópicos de Segurança da Informação	163
10.1 Dicas para se Prevenir Contra os Hackers	163
10.2 Dicas para se Prevenir Contra os Crackers	163
10.3 Implementando Segurança com Criptografia	165
10.3.1 Criando o Projeto Security	166
10.3.2 Criando o Projeto SecureApp	171
10.3.3 Criando o Projeto WindowsAppCSharp	175
10.4 Como será o Funcionamento?	180
10.5 Implementando a Segurança na nossa Aplicação	182
10.6 Adicionando a dll SecureApp.dll como Referência	191
11 Finalizando	197
11.1 Testando as Regras de Negócio na Aplicação para Windows	197
11.2 Estrutura de Arquivos	198
11.3 Vantagens do Modelo de Desenvolvimento em Camadas	199
Referências	203

Introdução

O meu objetivo ao escrever esse livro é que o leitor pratique e desenvolva uma aplicação real em .Net, adquirindo habilidades para a construção ou manutenção de aplicações mais complexas com o passar do tempo.

Esse livro foi concebido como um guia passo a passo que abrange todas as etapas da implementação relacionadas ao desenvolvimento de uma aplicação .Net em camadas.

Conforme sua evolução, você irá adquirir conhecimentos sobre:

- Comunicação entre as camadas;
- Vantagens do modelo de desenvolvimento em camadas;
- Controle de transações do banco de dados com o ADO .Net;
- Construção de uma aplicação para Windows;
- Criação de páginas Web;
- Tópicos sobre Segurança da Informação.

A metodologia utilizada é a do desenvolvimento em Três Camadas. Veremos a diferença entre:

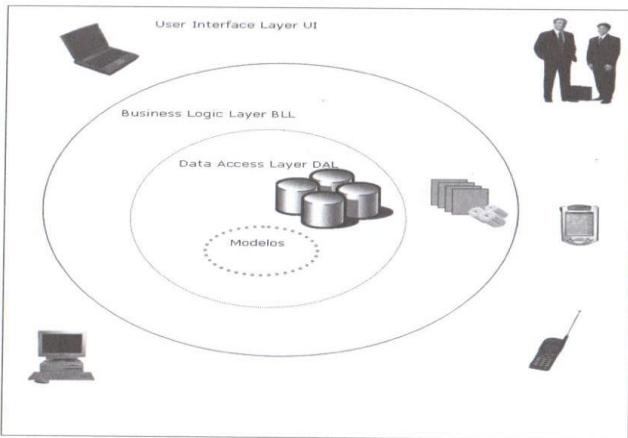
- Camada de Acesso a Dados ou Data Access Layer (DAL);
- Camada de Regras de Negócio ou Business Logic Layer (BLL);
- Camada de Interface do Usuário ou User Interface (UI).

Você aprenderá como realizar o desenvolvimento de uma aplicação em camadas usando stored procedures e também conhecerá métodos para desenvolver uma aplicação através de simples comandos SQL (Structured Query Language ou Linguagem de Consulta Estruturada).

Os comandos de SQL são usados para o gerenciamento de bancos de dados relacionais.

Na camada de interface com o usuário reutilizaremos as camadas DAL e BLL para criarmos dois projetos de interface para o usuário. O

(WinForms) e o outro versará sobre criação de uma página Web (Webpage) onde permitiremos que o usuário realize as operações de inclusão, alteração, exclusão e consulta.



Espero que com este livro, você esteja munido de conhecimentos práticos úteis para o seu dia-a-dia.

Desejo que este material agregue dinamismo e eficiência na construção de softwares para os desenvolvedores de soluções de TI.

Para os professores, espero que o guia possa ser usado como um exemplo prático da utilização de conceitos de programação orientada a objetos, integridade referencial de banco de dados e uso de transações via linguagem de programação C# .Net.

Para os alunos e entusiastas da tecnologia Microsoft .Net, desejo que este material consiga demonstrar o potencial que você tem em mãos ao fazer uso dessa tecnologia.

Requisitos Técnicos

Esse projeto foi desenvolvido utilizando os softwares:

- Microsoft Windows XP Professional;
- Microsoft SQL Server 2005 Professional;
- Microsoft Visual Studio 2005 Professional Edition.

Nota: O código-fonte em C# .Net utilizado é compatível com o Microsoft Visual Studio 2008 e os scripts de banco de dados para criação de tabelas e os stored procedures são compatíveis com o MS SQL Server 2000, 2005 e 2008.

O código-fonte em C# .Net utilizado é compatível com o Microsoft Visual Studio 2008.

Bom estudo!

1

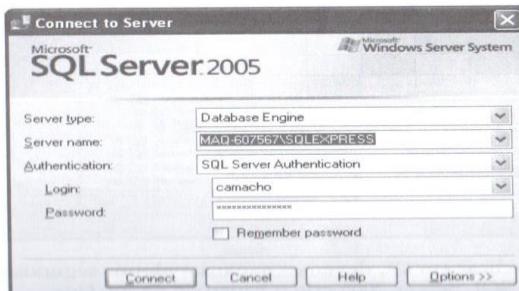
Criando a Infra-estrutura de Banco de Dados

Para o nosso projeto, vamos precisar de uma infra-estrutura simples de banco de dados contendo apenas três tabelas: clientes, produtos e vendas.

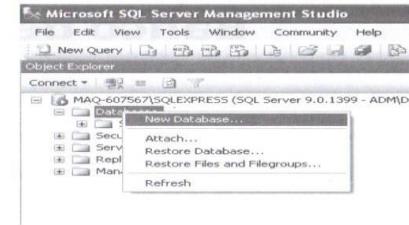
Se você estiver utilizando o MS SQL Server 2005 Professional, poderá trabalhar com o Microsoft SQL Server Management Studio. Será necessário executá-lo em sua máquina. Para executá-lo, siga as próximas instruções.

- Clique em Iniciar>Programas>Microsoft SQL Server 2005>SQL Server Management Studio;
- Selecione o servidor de banco de dados e o usuário para realizar a conexão.

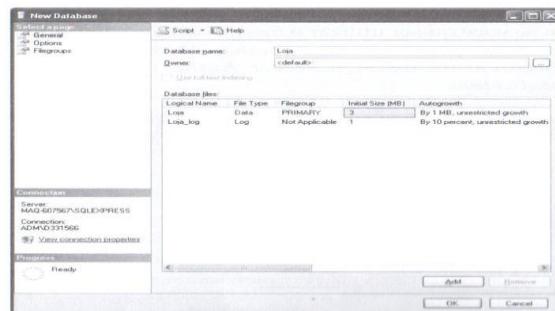
Nota: Se você quiser utilizar o mesmo usuário do Windows logado atualmente para acessar o banco de dados, altere o campo Authentication de SQL Server Authentication para Windows Authentication.



- Estabelecida a conexão, podemos criar um banco de dados para o nosso projeto. Para isso, clique com o botão direito sobre *Databases* e escolha a opção *New Database...*

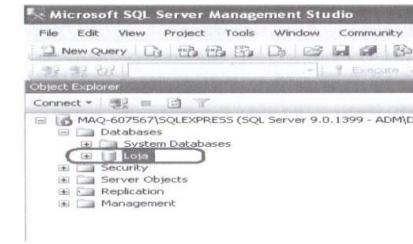


- Vamos nomear o nosso database como *Loja*;
→ Deixe a configuração padrão para o tamanho inicial dos arquivos de dados Data e Log. Clique em *Ok* para criar o database;



- Após clicar em *Ok*, devemos aguardar alguns segundos para que o MS SQL Server 2005 crie o banco de dados *Loja*;

Quando a janela New Database desaparecer, podemos verificar que o banco de dados *Loja* foi criado com sucesso conforme figura a seguir:



Agora que já temos o nosso banco de dados, vamos executar os scripts para criar as três tabelas necessárias ao nosso projeto.

1.1 Criando a Tabela de Clientes

- Clicando com o botão direito sobre o database *Loja*, escolha a opção *New Query*.

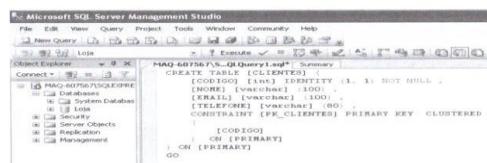


Copie o script SQL a seguir e cole na janela de Query aberta.

(Arquivo Loja_Fontes\BancodeDados\TabelaClientes.sql)

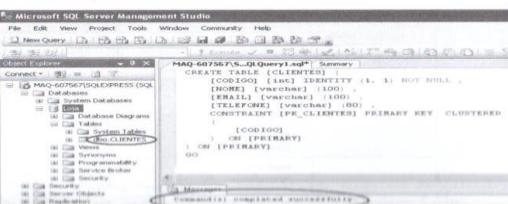
```
CREATE TABLE [CLIENTES]
(
    [CODIGO] [int] IDENTITY (1, 1) NOT NULL ,
    [NOME] [varchar] (100) ,
    [EMAIL] [varchar] (100) ,
    [TELEFONE] [varchar] (80) ,
    CONSTRAINT [PK_CLIENTES] PRIMARY KEY CLUSTERED
    (
        [CODIGO]
    ) ON [PRIMARY]
) ON [PRIMARY]
GO
```

O código deverá ficar assim:



Para executar esse script e criar a Tabela de Clientes, clique no botão Execute.

Se a mensagem *Command(s) completed successfully* for exibida, podemos expandir o database *Loja* e depois expandir *Tables* para vermos a Tabela de Clientes que acabamos de criar.



O campo *Código* da nossa Tabela de Clientes foi criado como *Identity Primary Key*. O que significa isso?

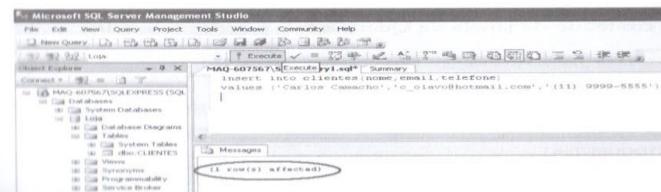
Bem, *Primary Key* (chave primária) é um campo que será único para cada registro da tabela, ou seja, só existirá um campo com o código igual a 1. O mesmo acontecerá com o código igual a 2. É um campo que não admite valores repetidos. Se tentarmos incluir um valor na chave primária já existente na tabela, o MS SQL Server vai exibir uma mensagem de erro e não permitirá essa inclusão.

A característica *Identity* desse campo determina que ele será gerenciado pelo MS SQL Server e nós não precisaremos nos preocupar em inserir um valor inexistente na tabela para evitar a duplicidade. Na prática, isso quer dizer que nós nem precisamos mencionar esse campo nas operações de inserção, pois é o gerenciador de banco de dados que se encarregará de inserir uma chave primária válida.

Vamos fazer uma inserção de registro na Tabela de Clientes mas, antes temos que apagar todos os códigos existentes na janela do Management Studio e vamos colar o comando a seguir:

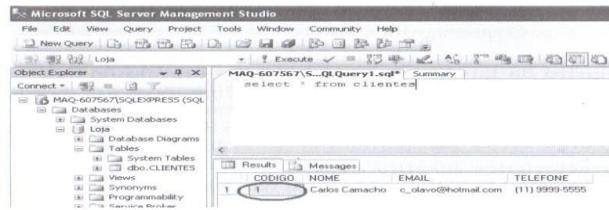
```
insert into clientes(nome,email,telefone)
values ('Carlos Camacho','email@provedor.com.br','(11) 9999-5555')
```

→ Clique em *Execute* para que o comando de inserção seja executado:



Se a mensagem *1 row(s) affected* for exibida, significa que o registro foi inserido com sucesso.

Execute o comando de seleção a seguir para visualizarmos o registro que acabamos de inserir na Tabela de Clientes:



Apesar de não mencionarmos o campo *Código* no comando de *Insert* executado anteriormente, o MS SQL Server providenciou a inserção desse valor automaticamente.

Se executarmos outro comando de *Insert*, o MS SQL Server vai gerar um código diferente para inserir no campo *Código* do novo registro.

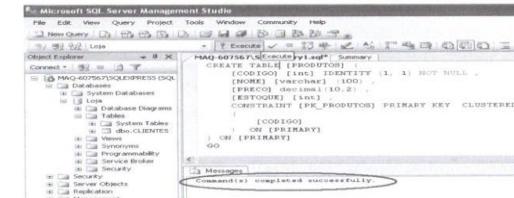
Muito bom! Acabamos de preparar a infra-estrutura do nosso projeto com a Tabela de Clientes criada corretamente.

1.2 Executando o Script para a Criação da Tabela de Produtos

O processo para a criação da Tabela de Produtos é o mesmo. Apague o conteúdo da janela de Query e cole o script a seguir:

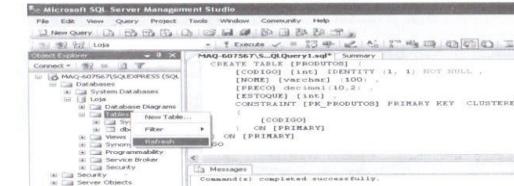
```
(Arquivo Loja_Fontes\BancodeDados\TabelaProdutos.sql)
CREATE TABLE [PRODUTOS]
(
    [CODIGO] [int] IDENTITY (1, 1) NOT NULL ,
    [NOME] [varchar] (100) ,
    [PRECO] decimal(10,2) ,
    [ESTOQUE] [int] ,
    CONSTRAINT [PK_PRODUTOS] PRIMARY KEY CLUSTERED
    (
        [CODIGO]
    ) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Com o script na janela de Query, clique em *Execute* para criarmos a

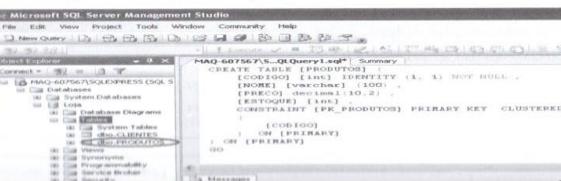


Nesse instante você deve estar pensando... depois que cliquei no *Execute*, apareceu a mensagem *Command(s) completed successfully*, mas a Tabela de Produtos não apareceu em Tables... O que aconteceu?

Seu questionamento procede! Precisamos atualizar o Management Studio para ver a tabela recém-criada. Para isso, vamos clicar com o botão direito do mouse em *Tables* e escolher a opção *Refresh*.



Veja o Management Studio após o *Refresh*:



Agora, podemos ver a Tabela de Produtos.

E, que tal inserirmos dois produtos nessa tabela?

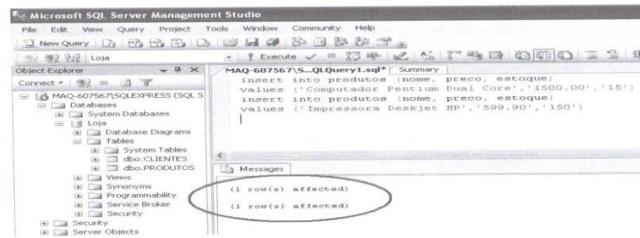
Vamos proceder da seguinte maneira:

→ Apague o código da janela de Query;

→ Copie o código a seguir na janela de Query e clique em *Execute* para a inserção dos dois produtos.

```
insert into produtos (nome, preco, estoque)
values ('Computador Pentium Dual Core','1500.00','15')
insert into produtos (nome, preco, estoque)
values ('Impressora Deskjet HP','599.90','150')
```

Veja o resultado:



Na janela de mensagens, podemos ver que os dois registros foram inseridos com sucesso.

Utilize o comando *select* para consultar os produtos cadastrados.

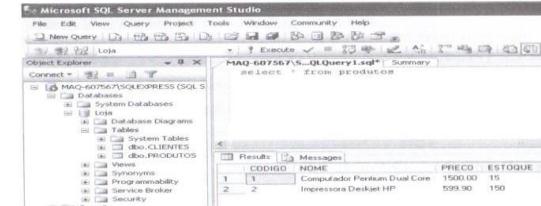
O procedimento é simples:

→ Apague o conteúdo da janela de Query;

→ Copie o comando a seguir e clique em *Execute*.

```
select * from produtos
```

Veja o resultado:



A nossa inserção de produtos funcionou! Cadastramos 15 computadores Pentium Dual Core com o valor unitário de R\$ 1.500,00 e 150 impressoras Deskjet HP com o valor unitário de R\$ 599,90.

A nossa Tabela de Produtos está pronta. Agora só falta criarmos a Tabela de Vendas!

1.3 Criando a Tabela para Armazenar as Vendas

O procedimento para criação da Tabela de Vendas é simples:

→ Apague o conteúdo da janela de Query;

→ Copie o comando a seguir e clique em *Execute*.

(Arquivo Loja_Fontes\BancodeDados\TabelaVendas.sql)

```
CREATE TABLE [VENDAS] (
    [CODIGO] [int] IDENTITY (1, 1) NOT NULL ,
    [DATA] [datetime],
    [QUANTIDADE] [int],
    [FATURADO] bit,
    [CODIGOCLIENTE] [int],
    [CODIGOPRODUTO] [int],
    CONSTRAINT [PK_VENDAS] PRIMARY KEY CLUSTERED
    (
        [CODIGO]
    ) ON [PRIMARY],
    CONSTRAINT [FK_Codigo_Cliente] FOREIGN KEY
    (
        [CODIGOCLIENTE]
    ) REFERENCES [Clientes] (
        [Codigo]
```

```

CONSTRAINT [FK_Codigo_Produto] FOREIGN KEY
(
    [CODIGOPRODUTO]
) REFERENCES [Produtos] (
    [Codigo]
)

) ON [PRIMARY]
GO

```

Veja o resultado:

```

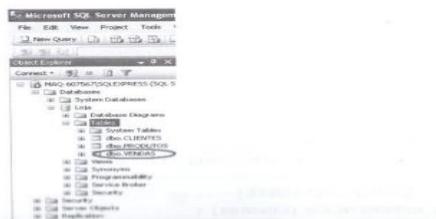
CREATE TABLE [dbo].[Venda] (
    [CODIGO] int IDENTITY (1, 1) NOT NULL,
    [DATA] datetime,
    [CUSTODIA] int,
    [FATURADO] int,
    [CODIGOCLIENTE] int,
    [CODIGOPRODUTO] int,
    CONSTRAINT [PK_VENDAS] PRIMARY KEY CLUSTERED
        ([CODIGO])
) ON [PRIMARY]
GO
CREATE TABLE [dbo].[Cliente] (
    [Codigo] int IDENTITY (1, 1) NOT NULL,
    [Nome] varchar(100),
    [Email] varchar(100),
    [Telefone] varchar(80)
) ON [PRIMARY]
GO
CREATE TABLE [dbo].[Produto] (
    [Codigo] int IDENTITY (1, 1) NOT NULL,
    [Nome] varchar(100),
    [Preco] decimal(10, 2),
    [Estoque] int,
    CONSTRAINT [PK_Produtos] PRIMARY KEY CLUSTERED
        ([Codigo])
) ON [PRIMARY]
GO
CONSTRAINT [FK_Codigo_Cliente] FOREIGN KEY
(
    [CODIGOCLIENTE]
) REFERENCES [Clientes] (
    [Codigo]
)

CONSTRAINT [FK_Codigo_Produto] FOREIGN KEY
(
    [CODIGOPRODUTO]
) REFERENCES [Produtos] (
    [Codigo]
)

) ON [PRIMARY]
GO

```

Dê um *Refresh*, clicando com o botão direito do mouse em *Tables* para que a Tabela de Vendas seja exibida.



1.4 Criando Stored Procedures

Neste tópico, iremos criar uma *stored procedure* de inclusão para a Tabela de Clientes.

- Apague o conteúdo da janela de Query;
- Copie o comando a seguir e clique em *Execute*.

(Arquivo Loja_Fontes\BancodeDados\Insere_Cliente.sql)

```

USE Loja
GO

```

```

CREATE PROCEDURE [dbo].[insere_cliente]
    @codigo int output,
    @nome varchar(100),
    @email varchar(100),
    @telefone varchar(80)
AS
    INSERT INTO CLIENTES (nome, email, telefone)
    VALUES (@nome,@email,@telefone)
    SET @codigo = (SELECT @@IDENTITY)

```

Observação: Alguns Administradores de Bancos de Dados (DBAs) costumam verificar, logo após o *USE Loja*, se a *stored procedure* que vai ser criada já existe no banco de dados. Em caso positivo, se executa o comando *DROP PROCEDURE* para excluir a antes de usar o comando *CREATE PROCEDURE*:

```

IF EXISTS (SELECT name FROM sysobjects
           WHERE name = 'insere_cliente' AND type = 'P')
    DROP PROCEDURE insere_cliente

```

Agora, vamos criar uma *stored procedure* de alteração para a Tabela de Clientes:

- Apague o conteúdo da janela de Query;
- Copie o comando a seguir e clique em *Execute*.

(Arquivo Loja_Fontes\BancodeDados\Altera_Cliente.sql)

```

USE Loja

```

```

CREATE PROCEDURE [dbo].[altera_cliente]
    @codigo int,
    @nome varchar(100),
    @email varchar(100),
    @telefone varchar(80)
AS
UPDATE Clientes SET nome = @nome, email = @email, telefone = @telefone where
    codigo = @codigo

```

A seguir, vamos criar uma stored procedure de exclusão para a Tabela de Clientes:

- Apague o conteúdo da janela de Query;
- Copie o comando a seguir e clique em Execute.

(Arquivo Loja_Fontes\BancodeDados\Exclui_Cliente.sql)

```

USE Loja
GO
CREATE PROCEDURE [dbo].[exclui_cliente]
    @codigo int
AS
DELETE FROM Clientes WHERE codigo = @codigo

```

Iremos criar também uma stored procedure de consulta para a Tabela de Clientes:

- Apague o conteúdo da janela de Query;
- Copie o comando a seguir e clique em Execute.

(Arquivo Loja_Fontes\BancodeDados\Seleciona_Cliente.sql)

```

USE Loja
GO
CREATE PROCEDURE [dbo].[seleciona_cliente]
    @filtro varchar(100) = NULL
AS
BEGIN
    IF @filtro IS NULL
        SELECT * FROM Clientes
    ELSE
        SELECT * FROM Clientes

```

```

    OR email like '%' + @filtro + '%'
    OR telefone like '%' + @filtro + '%'
END

```

1.5 Observações Importantes

A Tabela de Vendas que criamos no item 1.3, possui duas *foreign keys* (chaves estrangeiras). Essas duas chaves são os campos: *CodigoCliente* e *CodigoProduto*.

Isso significa que quando um registro for inserido nesta tabela, o campo *CodigoCliente* incluído deverá existir na Tabela de Clientes, da mesma forma como o campo *CodigoProduto* deverá existir na Tabela Produtos.

Se alguém tentar incluir valores nos campos *CodigoCliente* e *CodigoProduto* que não existam nas tabelas citadas, o Microsoft SQL Server vai informar a ocorrência de um erro e não permitirá a inclusão.

Esse mecanismo do sistema gerenciador de banco de dados existe para manter a integridade dos dados. Nomeamos esse procedimento de Integridade Referencial.

Uma regra de nosso projeto é que ao se realizar uma venda, atualizaremos o estoque do produto na Tabela de Produtos.

A fórmula matemática para que o estoque fique atualizado é a seguinte:

$$\text{Estoque} = \text{Estoque} - \text{Quantidade Vendida}$$

Essa tarefa será realizada juntamente com inclusão da venda na Tabela de Vendas.

Assim, para manter a integridade dos dados, precisaremos garantir que duas coisas aconteçam:

- A inclusão na Tabela de Vendas;
- A atualização do estoque na Tabela de Produtos.

Para isso, usaremos uma transação que nos garante a execução das duas tarefas. Se uma delas for bem sucedida e a outra falhar, a transação desfaz a primeira tarefa mantendo assim, a Integridade Referencial.

Em outras palavras, não existirá na Tabela de Vendas um registro que tenha o código do produto vendido, mas que o estoque deste produto na Tabela de Produtos esteja desatualizado.

Veremos isso com mais detalhes no capítulo sobre a Camada de Acesso a Dados (Data Access Layer), onde implementaremos as classes de acesso a dados e faremos esse controle transacional no método incluir da classe de Vendas.

Precisaremos guardar as seguintes informações sobre o nosso banco de dados pois, iremos utilizá-las futuramente em nosso projeto:

- Nome do servidor de banco de dados que estamos usando;
- Nome do banco de dados onde as três tabelas foram criadas;
- Nome do usuário e senha utilizados para acessar o banco de dados.

Essas informações são importantes para a construção da nossa string de conexão ou *connectionstring*.

A *connectionstring* será usada para a conexão com o banco de dados através da aplicação que desenvolveremos neste projeto.

Um bom site para consulta sobre *connectionstring* é: <<http://www.connectionstrings.com>>, onde existem dicas para a criação de *connectionstrings* para diferentes versões de bancos de dados.

Em nosso caso, se fecharmos a janela de Query que estávamos usando, poderemos ver o nome do servidor de banco de dados que estamos utilizando:



O usuário e senha são os usados inicialmente para a conexão no servidor de banco de dados, antes da criação do database *Loja*.

Nome do Servidor:	MAQ-607567\SQLEXPRESS
Nome do banco de dados:	Loja
Usuário:	camacho
Senha:	camacho2008a3

Dessa forma, a nossa *connectionstring* é:

```
"server=MAQ-607567\SQLEXPRESS;database=Loja;user=camacho;
pwd=camacho2008a3"
```

A sua *connectionstring* vai depender dos nomes que você irá utilizar para:

- Criar o servidor de banco de dados;
- Criar o banco de dados;
- Criar usuário e senha que terão acesso ao banco de dados.

Se estivéssemos usando o próprio usuário do Windows para acessar o banco de dados, não seria necessário informar o usuário e a senha; e a nossa *connectionstring* ficaria assim:

```
"server=MAQ-607567\SQLEXPRESS;database=Loja;Trusted_Connection=True"
```

Usaremos essas informações quando criarmos uma classe chamada Dados no Capítulo 3.

2

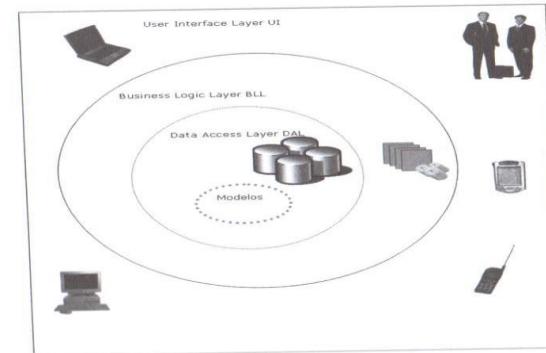
A Camada de Acesso a Dados

Essa camada é também chamada de DAL (Data Access Layer). Nessa camada, iremos implementar os métodos de inserção, atualização, exclusão e listagem referentes a todas as tabelas existentes em nosso projeto.

Essa é uma tarefa simples, pois para criar cada classe usaremos os nomes dos campos da respectiva tabela existente no banco de dados.

Você se lembra da figura que representa as três camadas?

Aqui está ela:



Sempre começamos o desenvolvimento da aplicação da camada mais interna até a mais externa. Dessa forma, iniciaremos a implementação com a Camada de Acesso a Dados (DAL).

2.1 Criando o Projeto

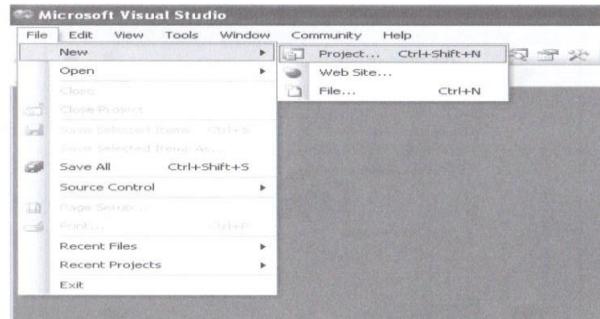
Dentro da Camada DAL, nós temos o projeto *Modelos* que iremos criar agora.

No projeto chamado *Modelos* iremos implementar as classes:

- *ClienteInformation.cs*
- *ProdutoInformation.cs*
- *VendaInformation.cs*

É necessário criar uma classe para cada tabela do nosso projeto.

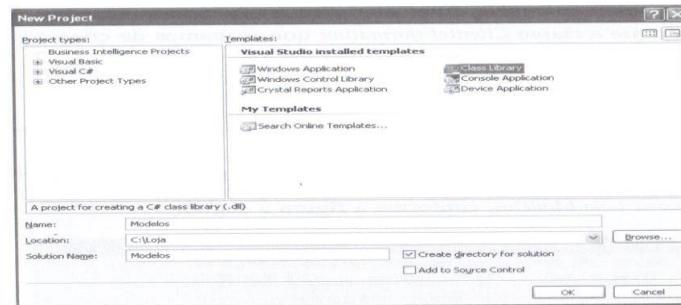
- A partir do menu *Iniciar>Programas>Microsoft Visual Studio 2005*, abra o *Microsoft Visual Studio 2005*.
- Clique no menu *File>New>Project...*



Acompanhe as seleções da janela New Project:

- No Tipo de Projeto, selecione *Visual C#*;
- No Tipo de Template, selecione *Class Library*;
- No Nome do Projeto, digite *Modelos*;
- Na Localização do Projeto, digite *C:\Loja*;

- Deixe a opção *Create directory for solution* selecionada;
- Clique em *Ok* para criar o projeto.



Abrindo o Windows Explorer, podemos ver que a pasta *Loja* foi criada no drive C, e que dentro dela foi criada a pasta do projeto *Modelos*.



Agora, vamos voltar para o Microsoft Visual Studio. Iremos trabalhar na área do Solution Explorer que fica no lado direito da tela de seu computador.

- Clique com o botão direito do mouse sobre o arquivo *Class1.cs*;
- Escolha *Rename* para renomear a classe para *ClienteInformation.cs*. Ao renomear o arquivo da classe, perceba que o nome da classe é alterado automaticamente na janela de código de *Class1* para *ClienteInformation*.

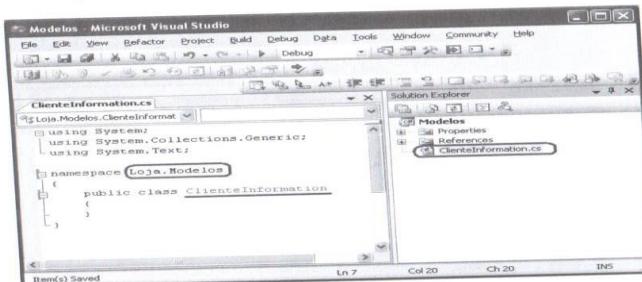
Todas as classes que criaremos no nosso projeto *Modelos* precisam ser públicas para que fiquem visíveis nos próximos projetos. Para fazer isso, sempre que criarmos uma classe no projeto *Modelos* usaremos a palavra reservada *public* antes da definição da classe.

Para que a classe *ClienteInformation* que acabamos de criar seja pública, vamos usar a palavra reservada *public*.

- Na janela de código digite a palavra *public* antes de *class ClienteInformation*.

Assim, a definição da nossa classe ficará *public class ClienteInformation* como na figura a seguir:

- Na janela de código à esquerda, no campo namespace, inclua o nome do projeto *Loja*, de forma que o namespace fique definido com *Loja.Modelos*, conforme a figura a seguir:



Agora, iremos codificar a classe *ClienteInformation*. Para isso, copie e cole o código a seguir entre as chaves da classe *ClienteInformation*:

```

private int _codigo;
public int Código
{
    get { return _codigo; }
    set { _codigo = value; }
}
  
```

A Camada de Acesso a Dados

```

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

private string _email;
public string Email
{
    get { return _email; }
    set { _email = value; }
}

private string _telefone;
public string Telefone
{
    get { return _telefone; }
    set { _telefone = value; }
}
  
```

Na implementação da classe *ClienteInformation*, podemos ver que estamos definindo campos e propriedades para cada campo da Tabela de Clientes que foi criada no banco de dados.

Agora, a codificação da classe *ClienteInformation* está completa como na listagem a seguir:

(Arquivo Loja_Fontes\Modelos\ClienteInformation.cs)

```

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

namespace Loja.Modelos
{
    public class ClienteInformation
    {
        private int _codigo;
        public int Código
        {
            get { return _codigo; }
            set { _codigo = value; }
        }

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

```

}
private string _email;
public string Email
{
    get { return _email; }
    set { _email = value; }
}

private string _telefone;
public string Telefone
{
    get { return _telefone; }
    set { _telefone = value; }
}
}

```

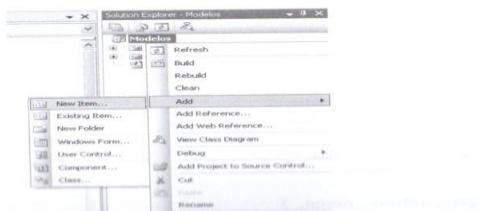
Importante: Certifique-se de que a palavra reservada `public` está antes da definição da classe:

```
public class ClienteInformation
```

Ela precisa ser formatada desta forma, porque, futuramente, podemos utilizar essa classe a partir de outros projetos. Essa observação é válida para todas as classes do projeto `Modelos`.

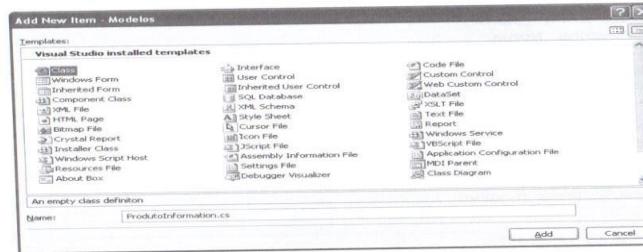
Agora, iremos implementar a classe `ProdutoInformation.cs`:

- Clique com o botão direito do mouse sobre o projeto `Modelos`;
- Escolha a opção `Add`;
- Clique em `New Item..`.

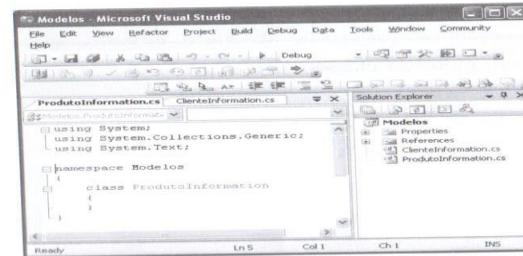


Na janela `Add New Item`:

- Escolha o template `Class` e digite `ProdutoInformation.cs` no campo nome;
- Clique em `Add` para adicionar a nova classe ao projeto `Modelos`.



Neste momento, a nova classe `ProdutoInformation.cs` está criada:



- O namespace deve ser alterado para:
`namespace Loja.Modelos`

Agora, digite o código a seguir dentro da classe `ProdutoInformation`:

```

private int _codigo;
public int Código
{
    get { return _codigo; }
    set { _codigo = value; }
}

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

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

private int _estoque;
public int Estoque
{
    get { return _estoque; }
    set { _estoque = value; }
}

```

O código completo da classe *ProdutoInformation.cs* ficará assim:

(Arquivo Loja_Fontes\Modelos\ProdutoInformation.cs)

```

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

namespace Loja.Modelos
{
    public class ProdutoInformation
    {
        private int _codigo;
        public int Código
        {
            get { return _codigo; }
            set { _codigo = value; }
        }
    }
}

```

```

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

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

private int _estoque;
public int Estoque
{
    get { return _estoque; }
    set { _estoque = value; }
}

```

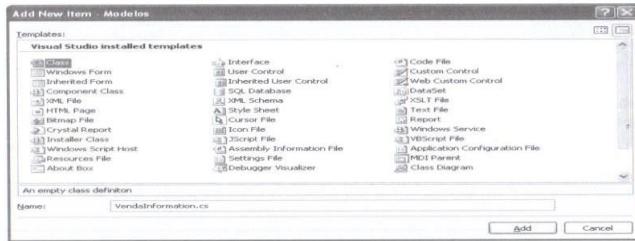
Muito bem! Já implementamos as classes do projeto *Modelo* para a Tabela de Clientes e para a Tabela de Produtos. Agora, vamos fazer o mesmo para a Tabela de Vendas:

- Clique com o botão direito do mouse sobre o projeto *Modelos*;
- Escolha a opção *Add>New Item...*

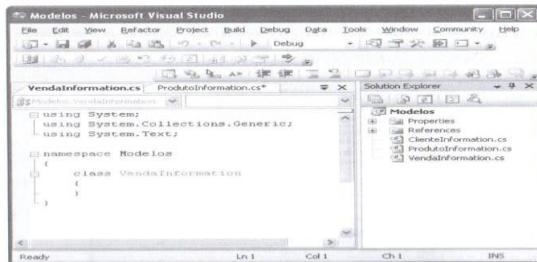


Na janela Add New Item:

- Escolha o template *Class* e digite *VendaInformation.cs* no campo nome;
- Clique em *Add* para adicionar a nova classe ao projeto *Modelos*.



Neste momento, a nova classe *VendaInformation.cs* está criada:



→ O namespace deve ser alterado para:

```
namespace Loja.Modelos
```

Acesse, digite a edição a esquerda dentro da classe *VendaInformation*:

A Camada de Acesso a Dados

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

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

private int _quantidade;
public int Quantidade
{
    get { return _quantidade; }
    set { _quantidade = value; }
}

private bool _faturado;
public bool Faturado
{
    get { return _faturado; }
    set { _faturado = value; }
}

private int _codigoCliente;
public int CodigoCliente
{
    get { return _codigoCliente; }
    set { _codigoCliente = value; }
}

private int _codigoProduto;
public int CodigoProduto
{
    get { return _codigoProduto; }
    set { _codigoProduto = value; }
}

private string _nomeCliente;
public string NomeCliente
```

```

    {
        get { return _nomeCliente; }
        set { _nomeCliente = value; }
    }
}

```

O código completo da classe *VendaInformation.cs* ficará assim:

(Arquivo Loja_Fontes\Modelos\VendaInformation.cs)

```

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

namespace Loja.Modelos
{
    public class VendaInformation
    {
        private int _codigo;
        public int Código
        {
            get { return _codigo; }
            set { _codigo = value; }
        }

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

        private int _quantidade;
        public int Quantidade
        {
            get { return _quantidade; }
            set { _quantidade = value; }
        }

        private bool _faturado;
        public bool Faturado
        {
            get { return _faturado; }
            set { _faturado = value; }
        }

        private int _codigoCliente;
    }
}

```

```

public int CódigoCliente
{
    get { return _codigoCliente; }
    set { _codigoCliente = value; }
}

private int _codigoProduto;
public int CódigoProduto
{
    get { return _codigoProduto; }
    set { _codigoProduto = value; }
}

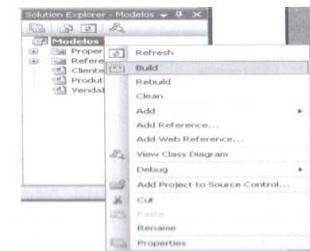
private string _nomeCliente;
public string NomeCliente
{
    get { return _nomeCliente; }
    set { _nomeCliente = value; }
}
}

```

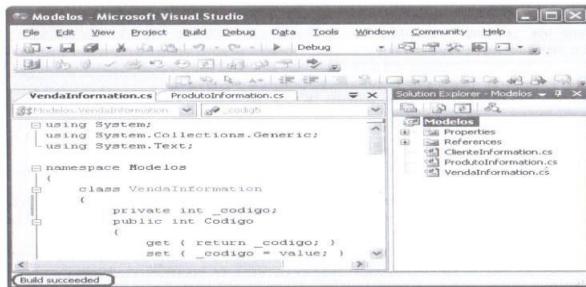
Agora que implementamos no projeto *Modelos* as classes referentes a todas as tabelas contempladas no nosso projeto, vamos compilar o projeto através da opção *Build*.

Para compilar o projeto, no Solution Explorer:

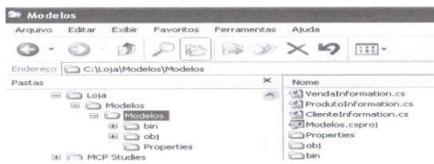
- Clique com o botão direito do mouse sobre o projeto *Modelos*;
- Escolha a opção *Build*.



Se a compilação do nosso projeto for realizada com sucesso, aparecerá a mensagem *Build succeeded* na barra de status do Microsoft Visual Studio:



Observe que na pasta do projeto *Modelos* no Windows Explorer, foi criado um arquivo para cada classe que implementamos.



Quando compilamos o projeto com a opção *Build*, o MS Visual Studio cria o arquivo *Modelos.dll*.

O arquivo *Modelos.dll* contém toda a informação que implementamos nas classes desse projeto, e é ele que será usado no próximo projeto - Camada de Acesso a Dados ou DAL (Data Access Layer) - que será implementado.

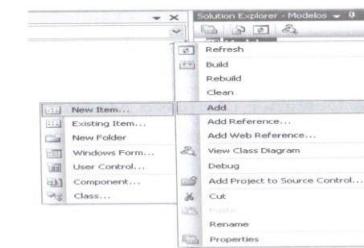


Após terminar o projeto *Modelos*, muitos programadores costumam criar um Diagrama de Classes para ter uma visão melhor da sua arte.

2.2 Criando o Diagrama de Classes

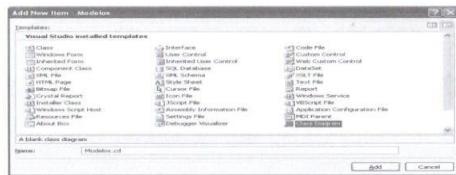
Para criar o Diagrama de Classes:

- No Visual Studio, clique com o botão direito do mouse sobre o projeto *Modelos*;
- Escolha a opção *Add>New Item...*

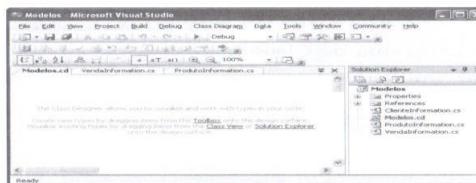


Na janela Add New Item:

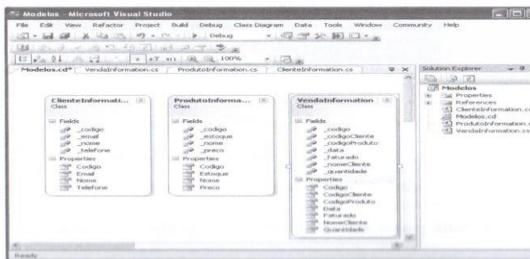
- Escolha o template *Class Diagram* e digite *Modelos.cd* no campo nome;
- Clique em *Add* para criar o Diagrama de Classes.



O ambiente chamado Class Designer é exibido na aba *Modelos.cd* como a seguir:



Para a criação do diagrama, vamos arrastar as classes *ClienteInformation*, *ProdutoInformation* e *VendaInformation* do Solution Explorer para o Class Designer (Arquivo *Modelos.cd*) como a seguir:



Com o Diagrama de Classes fica fácil a distinção entre os campos e as propriedades que implementamos no nosso projeto.

O asterisco ao lado do nome do diagrama *Modelos.cd** indica que ele ainda não foi salvo. Digite <Ctrl>+<S> para salvar o arquivo.

Se olharmos para o nosso desenho das camadas, vamos verificar que o próximo passo é a implementação das classes da Camada DAL. Faremos isso no próximo capítulo.

3

O Projeto DAL

Neste capítulo, vamos implementar as classes da Camada de Acesso a Dados ou DAL (Data Access Layer).

Criaremos as classes:

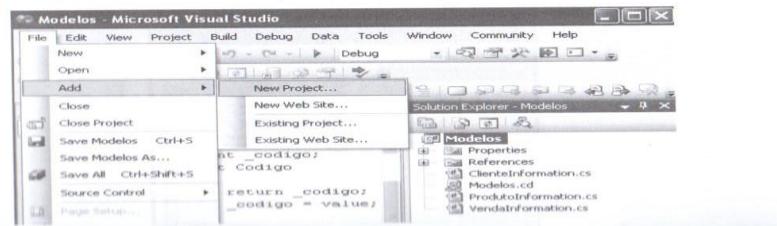
- ClientesDAL.cs
- ProdutosDAL.cs
- VendasDAL.cs
- Dados.cs

Será visto que a classe *ClientesDAL*, por exemplo, conterá os métodos de inclusão, alteração, exclusão e consulta referentes a Tabela de Clientes. O mesmo ocorrerá para as classes *ProdutosDAL* e *VendasDAL*.

Usaremos a classe *Dados* para armazenarmos a string de conexão com o banco de dados.

Vamos iniciar a nossa implementação:

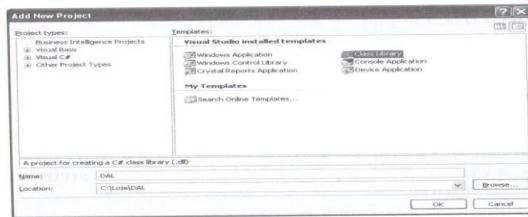
- Entre no Microsoft Visual Studio;
- Abra o projeto *Modelos* que criamos anteriormente;
- Vamos adicionar um novo projeto. No menu do Visual Studio clique em *File>Add>New Project...*



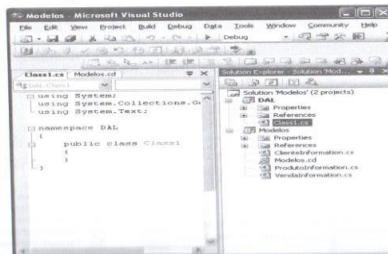
Na janela Add New Project:

- No Tipo de Projeto, selecione *Visual C#*;
- No Tipo de Template, selecione *Class Library*;
- No Nome do Projeto, digite *DAL*;
- Na Localização do Projeto, digite *C:\Loja\DAL*;
- Clique em *Ok* para adicionar o novo projeto à *Solution*.

Nota: Solution é quando temos um conjunto de projetos reunidos para atender uma necessidade.



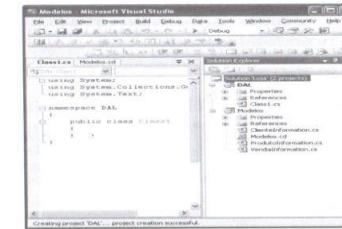
Muito bom! Agora, o nosso projeto *DAL* foi adicionado ao Solution. O Solution foi nomeado como *Modelos* porque o Microsoft Visual Studio usa o nome do primeiro projeto criado.



No Solution Explorer:

- Clique com o botão direito do mouse sobre o *Solution Modelos*;
- Escolha a opção *Rename* e renomeie o *Solution* para *Loja*.

Depois que você renomear o *Solution*, ele ficará assim:



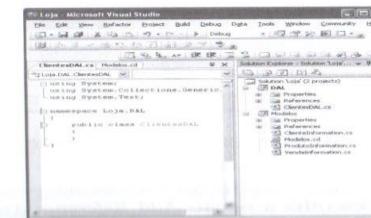
3.1 Implementando as Classes da Camada de Acesso a Dados

3.1.1 Classe ClientesDAL.cs

No projeto *DAL*:

- Renomeie a classe *Class1.cs* para *ClientesDAL.cs*;
- Altere o namespace para *Loja.DAL*.

A classe *ClientesDAL* ficará assim:



Você se lembra quando comentamos que a nossa implementação aconteceria da camada mais interna para a mais externa do desenho do nosso projeto?

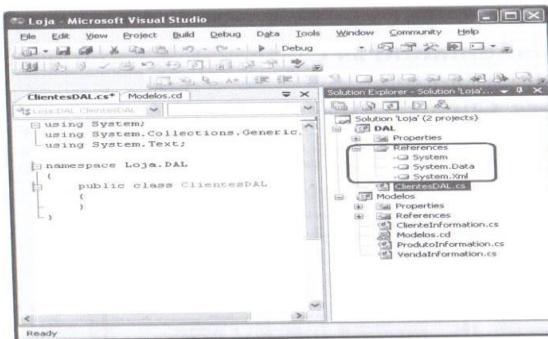
[Modelos] DAL> BLL> User Interface

Uma das vantagens do modelo de desenvolvimento em camadas é a reutilização de código. Assim, no projeto *DAL* reutilizaremos tudo o que foi desenvolvido no projeto *Modelos*.

Em termos de programação, dizemos que o projeto *DAL* faz referência ao projeto *Modelos*.

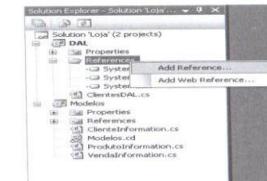
Para que o nosso projeto *DAL* consiga ver o que já construímos no projeto *Modelos*, vamos criar essa referência:

Abra a pasta *References* do projeto *DAL* e perceba que os namespaces *System*, *System.Data* e *System.Xml* já existem como referência quando criamos um novo projeto:

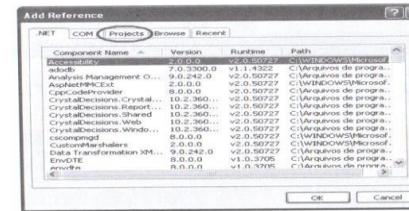


Precisamos incluir no projeto *DAL* uma referência para o *Modelos*. Faça o seguinte:

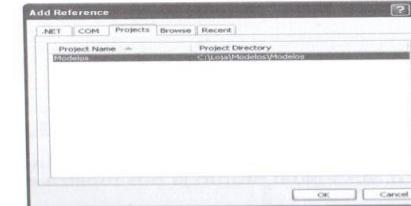
→ Clique com o botão direito do mouse na pasta *References* do projeto *DAL* e escolha a opção *Add Reference...*



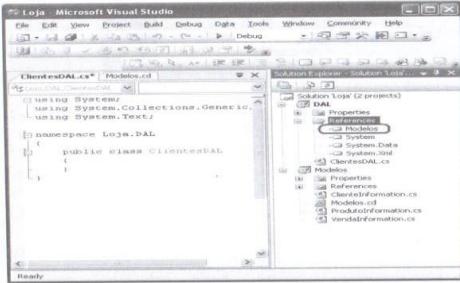
→ Na janela *Add Reference*, clique na aba *Projects*:



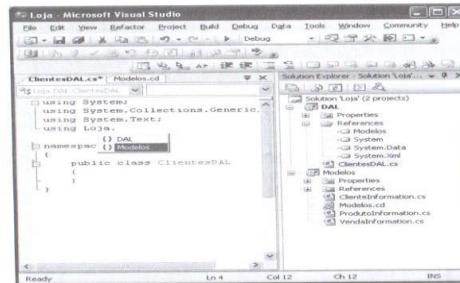
→ No projeto *Modelos*, clique sobre ele para selecioná-lo e depois em *Ok* para criar a referência.



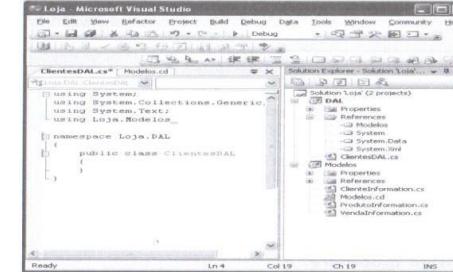
Neste momento, podemos ver a referência do projeto *Modelos* dentro do projeto *DAL*.



Para visualizar os *Modelos* dentro da nossa classe *ClientesDAL*, vamos adicionar a cláusula *using* como segue:



- Após digitarmos *using Loja.* (coloque o ponto após digitar *Loja*), verifique que o MS Visual Studio já nos mostra as referências disponíveis.
- Com as setas de direção, posicione a seleção em *Modelos* e tecle *Enter* para selecioná-lo.

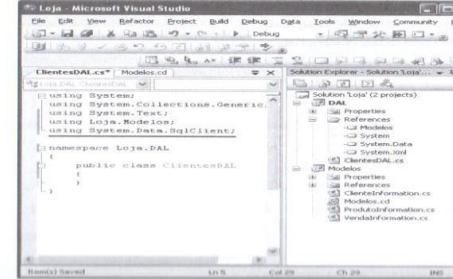


Na tela, aparecerá um tracinho vermelho após *Loja.Modelos* que indica que há algo errado. É o comando *using* que não foi fechado.

Para fechar o comando *using*:

→ Digite ponto-e-vírgula (;) após *using Loja.Modelos* para corrigir o problema.

Para que tenhamos acesso às definições do namespace do MS SQL Server, vamos inserir mais uma cláusula *using* para *System.Data.SqlClient*, como a seguir:



Para o código da classe *ClientesDAL*, digite o código a seguir entre as chaves da classe:

```
public void Incluir(ClienteInformation cliente)
{
    //conexao
    SqlConnection cn = new SqlConnection();
    try
    {
        cn.ConnectionString = Dados.StringDeConexao;
        //command
        SqlCommand cmd = new SqlCommand();
        cmd.Connection = cn;

        cmd.CommandType = CommandType.StoredProcedure;
        //nome da stored procedure
        cmd.CommandText = "insere_cliente";
        //parâmetros da stored procedure
        SqlParameter pcodigo = new SqlParameter("@codigo", SqlDbType.Int);
        pcodigo.Direction = ParameterDirection.Output;
        cmd.Parameters.Add(pcodigo);
        SqlParameter pnome = new SqlParameter("@nome", SqlDbType.NVarChar, 100);
        pnome.Value = cliente.Nome;
        cmd.Parameters.Add(pnome);
        SqlParameter pemail = new SqlParameter("@email", SqlDbType.NVarChar, 100);
        pemail.Value = cliente.Email;
        cmd.Parameters.Add(pemail);
        SqlParameter ptelefone = new SqlParameter("@telefone", SqlDbType.NVarChar, 80);
        ptelefone.Value = cliente.Telefone;
        cmd.Parameters.Add(ptelefone);

        cn.Open();
        cmd.ExecuteNonQuery();

        cliente.Codigo = (Int32)cmd.Parameters["@codigo"].Value;
    }
    catch (SqlException ex)
    {
        throw new Exception("Servidor SQL Erro:" + ex.Number);
    }
    catch (Exception ex)
    {
```

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

public void Alterar(ClienteInformation cliente)
{
    // conexao
    SqlConnection cn = new SqlConnection();
    try
    {
        cn.ConnectionString = Dados.StringDeConexao;
        //command
        SqlCommand cmd = new SqlCommand();
        cmd.Connection = cn;

        cmd.CommandType = CommandType.StoredProcedure;
        //nome da stored procedure
        cmd.CommandText = "altera_cliente";
        //parâmetros da stored procedure
        SqlParameter pcodigo = new SqlParameter("@codigo", SqlDbType.Int);
        pcodigo.Value = cliente.Codigo;
        cmd.Parameters.Add(pcodigo);
        SqlParameter pnome = new SqlParameter("@nome", SqlDbType.NVarChar, 100);
        pnome.Value = cliente.Nome;
        cmd.Parameters.Add(pnome);
        SqlParameter pemail = new SqlParameter("@email", SqlDbType.NVarChar, 100);
        pemail.Value = cliente.Email;
        cmd.Parameters.Add(pemail);
        SqlParameter ptelefone = new SqlParameter("@telefone", SqlDbType.NVarChar, 80);
        ptelefone.Value = cliente.Telefone;
        cmd.Parameters.Add(ptelefone);

        cn.Open();
        cmd.ExecuteNonQuery();

    }
    catch (SqlException ex)
    {
        throw new Exception("Servidor SQL Erro:" + ex.Number);
    }
}
```

```

        catch (Exception ex)
        {
            throw new Exception(ex.Message);
        }
        finally
        {
            cn.Close();
        }
    }

    public void Excluir(int codigo)
    {
        //conexao
        SqlConnection cn = new SqlConnection();
        try
        {
            cn.ConnectionString = Dados.StringDeConexao;
            //command
            SqlCommand cmd = new SqlCommand();
            cmd.Connection = cn;

            cmd.CommandType = CommandType.StoredProcedure;
            //nome da stored procedure
            cmd.CommandText = "exclui_cliente";

            //parâmetros da stored procedure
            SqlParameter pcodigo = new SqlParameter("@codigo", SqlDbType.Int);
            pcodigo.Value = codigo;
            cmd.Parameters.Add(pcodigo);

            cn.Open();

            int resultado = cmd.ExecuteNonQuery();
            if (resultado != 1)
            {
                throw new Exception("Não foi possível excluir o cliente " + codigo);
            }
        }
        catch (SqlException ex)
        {
            throw new Exception("Servidor SQL Erro:" + ex.Number);
        }
        catch (Exception ex)
        {
            throw new Exception(ex.Message);
        }
    }
}

```

```

        throw new Exception(ex.Message);
    }
    finally
    {
        cn.Close();
    }
}

public DataTable Listagem(string filtro)
{
    SqlConnection cn = new SqlConnection();
    SqlDataAdapter da = new SqlDataAdapter();
    DataTable dt = new DataTable();

    try
    {
        cn.ConnectionString = Dados.StringDeConexao;
        //adapter
        da.SelectCommand = new SqlCommand();
        da.SelectCommand.CommandText = "seleciona_cliente";
        da.SelectCommand.Connection = cn;
        da.SelectCommand.CommandType = CommandType.StoredProcedure;

        //parâmetros da stored procedure
        SqlParameter pfiltro;
        pfiltro = da.SelectCommand.Parameters.Add ("@filtro", SqlDbType.Text);
        pfiltro.Value = filtro;

        da.Fill (dt);
        return dt;
    }
    catch (SqlException ex)
    {
        throw new Exception("Servidor SQL Erro:" + ex.Number);
    }
    catch (Exception ex)
    {
        throw new Exception(ex.Message);
    }
    finally
    {
        cn.Close();
    }
}

```

O código completo da nossa classe *ClientesDAL.cs* ficará assim:

```
(Arquivo Loja_Fontes\DAL\ClientesDAL.cs)
using System;
using System.Collections.Generic;
using System.Text;
using Loja.Modelos;
using System.Data.SqlClient;
using System.Data;

namespace Loja.DAL
{
    public class ClientesDAL
    {
        public void Incluir(ClienteInformation cliente)
        {
            //conexao
            SqlConnection cn = new SqlConnection();
            try
            {
                cn.ConnectionString = Dados.StringDeConexao;
                //command
                SqlCommand cmd = new SqlCommand();
                cmd.Connection = cn;

                cmd.CommandType = CommandType.StoredProcedure;
                //nome da stored procedure
                cmd.CommandText = "insere_cliente";

                //parametros da stored procedure
                SqlParameter pcodigo = new SqlParameter("@codigo", SqlDbType.Int);
                pcodigo.Direction = ParameterDirection.Output;
                cmd.Parameters.Add(pcodigo);
                SqlParameter pnome = new SqlParameter("@nome",
                    SqlDbType.NVarChar, 100);
                pnome.Value = cliente.Nome;
                cmd.Parameters.Add(pnome);
                SqlParameter pemail = new SqlParameter("@email",
                    SqlDbType.NVarChar, 100);
                pemail.Value = cliente.Email;
                cmd.Parameters.Add(pemail);
                SqlParameter ptelefone = new SqlParameter("@telefone",
                    SqlDbType.NVarChar, 80);
                ptelefone.Value = cliente.Telefone;
                cmd.Parameters.Add(ptelefone);
            }
        }
    }
}
```

```

cn.Open();
cmd.ExecuteNonQuery();

cliente.Codigo = (Int32)cmd.Parameters["@codigo"].Value;

}
catch (SqlException ex)
{
    throw new Exception("Servidor SQL Erro:" + ex.Number);
}
catch (Exception ex)
{
    throw new Exception(ex.Message);
}
finally
{
    cn.Close();
}

public void Alterar(ClienteInformation cliente)
{
    // conexao
    SqlConnection cn = new SqlConnection();
    try
    {
        cn.ConnectionString = Dados.StringDeConexao;
        //command
        SqlCommand cmd = new SqlCommand();
        cmd.Connection = cn;

        cmd.CommandType = CommandType.StoredProcedure;
        //nome da stored procedure
        cmd.CommandText = "altera_cliente";

        //parametros da stored procedure
        SqlParameter pcodigo = new SqlParameter("@codigo", SqlDbType.Int);
        pcodigo.Value = cliente.Codigo;
        cmd.Parameters.Add(pcodigo);
        SqlParameter pnome = new SqlParameter("@nome",
            SqlDbType.NVarChar, 100);
        pnome.Value = cliente.Nome;
        cmd.Parameters.Add(pnome);
    }
}
```

Desenvolvimento em Camadas com C# .Net

```

        SqlParameter pemail = new SqlParameter("@email",
    SqlDbType.NVarChar, 100);
    pemail.Value = cliente.Email;
    cmd.Parameters.Add(pemail);
    SqlParameter ptelefone = new SqlParameter("@telefone", SqlDbType.
    NVarChar, 80);
    ptelefone.Value = cliente.Telefone;
    cmd.Parameters.Add(ptelefone);

    cn.Open();
    cmd.ExecuteNonQuery();

}

catch (SqlException ex)
{
    throw new Exception("Servidor SQL Erro:" + ex.Number);
}
catch (Exception ex)
{
    throw new Exception(ex.Message);
}
finally
{
    cn.Close();
}

}

public void Excluir(int codigo)
{
    //conexao
    SqlConnection cn = new SqlConnection();
    try
    {
        cn.ConnectionString = Dados.StringDeConexao;
        //command
        SqlCommand cmd = new SqlCommand();
        cmd.Connection = cn;

        cmd.CommandType = CommandType.StoredProcedure;
        //nome da stored procedure
        cmd.CommandText = "exclui_cliente";

        //parâmetros da stored procedure
        SqlParameter pcodigo = new SqlParameter("@codigo", SqlDbType.Int);

```

O Projeto DAL

```

        cmd.Parameters.Add(pcodigo);
        cn.Open();
        int resultado = cmd.ExecuteNonQuery();
        if (resultado != 1)
        {
            throw new Exception("Não foi possível excluir o cliente " + codigo);
        }
    catch (SqlException ex)
    {
        throw new Exception("Servidor SQL Erro:" + ex.Number);
    }
    catch (Exception ex)
    {
        throw new Exception(ex.Message);
    }
    finally
    {
        cn.Close();
    }
}

public DataTable Listagem(string filtro)
{
    SqlConnection cn = new SqlConnection();
    SqlDataAdapter da = new SqlDataAdapter();
    DataTable dt = new DataTable();

    try
    {
        cn.ConnectionString = Dados.StringDeConexao;
        /adapter
        da.SelectCommand = new SqlCommand();
        da.SelectCommand.CommandText = "seleciona_cliente";
        da.SelectCommand.Connection = cn;
        da.SelectCommand.CommandType = CommandType.StoredProcedure;

        //parâmetros da stored procedure
        SqlParameter pfiltro;
        pfiltro = da.SelectCommand.Parameters.Add ("@filtro", SqlDbType.Text);
        pfiltro.Value = filtro;

        da.Fill (dt);
        return dt;
    }
}

```

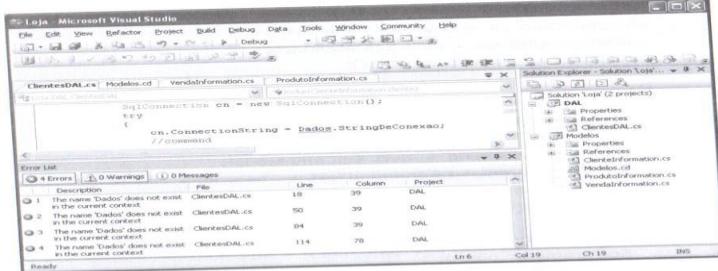
```

        catch (SqlException ex)
        {
            throw new Exception("Servidor SQL Erro:" + ex.Number);
        }
        catch (Exception ex)
        {
            throw new Exception(ex.Message);
        }
        finally
        {
            cn.Close();
        }
    }
}

```

Sempre que terminamos de codificar uma classe, é bom dar um *Build* no projeto. Esse procedimento é válido para nos certificarmos de que está tudo certo. Vamos fazer isso no projeto *DAL*:

- Clique com o botão direito do mouse sobre o projeto *DAL* e escolha *Build*:

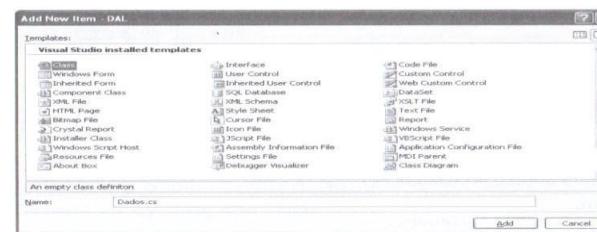


Agora, temos quatro erros para corrigir. Olhando para a descrição, podemos verificar que é o mesmo erro que ocorreu em quatro linhas diferentes da classe.

A descrição diz que o nome *Dados* não foi encontrado. A mensagem está correta, realmente ainda não codificamos a nossa classe *Dados* que armazenará a nossa string de conexão.

Tudo bem, então vamos codificá-la:

- Clique com o botão direito do mouse no projeto *DAL*;
- Selecione a opção *Add>New Item...*
- Na janela que se abrirá, vamos escolher o template *Class* e dar o nome de *Dados.cs*;
- Clique no botão *Add* para adicionar a nova classe ao projeto.



No Solution Explorer:

- Dê um duplo clique na classe *Dados.cs* para abrir o código;
- Altere o namespace para *Loja.DAL*.

Copie e cole o código entre as chaves da classe *Dados*:

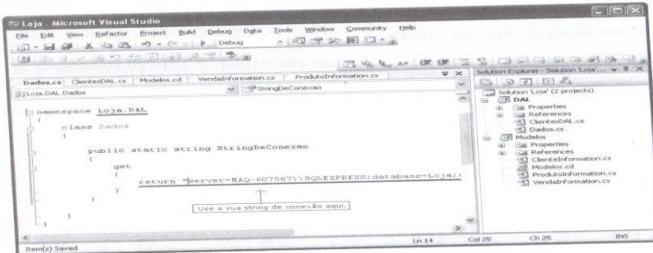
```

public static string StringDeConexao
{
    get
    {
        return "server=MAQ-607567\\SQLEXPRESS;database=Loja;user=camacho;
pwd=camacho2008a3";
    }
}

```

Desenvolvimento em Camadas com C# .Net

Importante: O conteúdo que você colocar entre as aspas após o `return` deverá ser a sua string de conexão. Os dados da sua string de conexão foram definidos no capítulo 1 que trata da criação da infra-estrutura de banco de dados.



O código completo da classe `Dados.cs` ficará assim:

(Arquivo Loja_Fontes\DAL\Dados.cs)

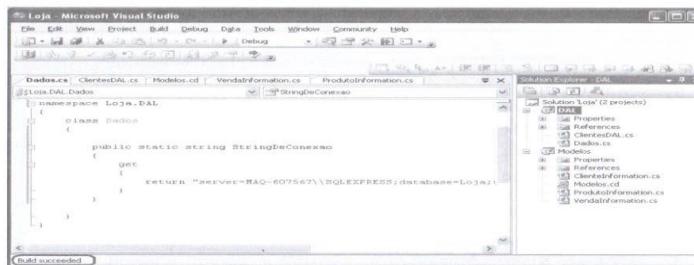
```
using System;
using System.Collections.Generic;
using System.Text;

namespace Loja.DAL
{
    class Dados
    {

        public static string StringDeConexao
        {
            get
            {
                return "Use a sua string de conexão aqui.";
            }
        }
    }
}
```

Após salvar o código, é necessário dar um *Build* no projeto *DAL*:

O Projeto DAL

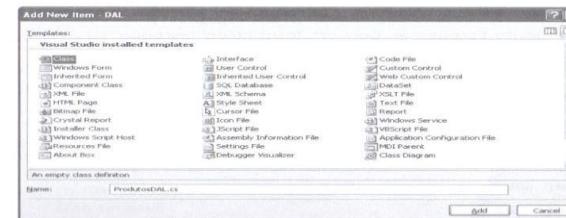


Muito bem! Podemos visualizar a frase *Build succeeded* na barra de status indicando que o nosso projeto foi compilado com sucesso.

3.1.2 Classe `ProdutosDAL.cs`

Agora, vamos implementar a classe `ProdutosDAL.cs`:

- Clique com o botão direito do mouse no projeto *DAL*;
- Selecione as opções *Add>New Item...*
- Nomeie a classe como `ProdutosDAL.cs` e clique em *Add*.



- Digite a listagem completa da classe `ProdutosDAL.cs` conforme o código a seguir:

```
(Arquivo Loja_Fontes\DAL\ProdutosDAL.cs)

using System;
using System.Collections.Generic;
using System.Text;
using System.Data;
using System.Data.SqlClient;
using System.Collections;
using Loja.Modelos;

namespace Loja.DAL
{
    public class ProdutosDAL
    {
        public ArrayList ProdutosEmFalta()
        {

            SqlConnection cn = new SqlConnection(Dados.StringDeConexao);
            SqlCommand cmd = new SqlCommand("Select * from Produtos Where
Estoque < 10", cn);

            cn.Open();

            SqlDataReader dr = cmd.ExecuteReader();
            ArrayList lista = new ArrayList();
            while (dr.Read())
            {
                ProdutoInformation produto = new ProdutoInformation();
                produto.Codigo = Convert.ToInt32(dr["codigo"]);
                produto.Nome = dr["nome"].ToString();
                produto.Estoque = Convert.ToInt32(dr["estoque"]);
                produto.Preco = Convert.ToDecimal(dr["preco"]);
                lista.Add(produto);
            }

            dr.Close();
            cn.Close();

            return lista;
        }

        public void Incluir(ProdutoInformation produto)
        {
            //conexao
        }
    }
}
```

```
SqlConnection cn = new SqlConnection();
try
{
    cn.ConnectionString = Dados.StringDeConexao;
    //command
    SqlCommand cmd = new SqlCommand();
    cmd.Connection = cn;
    cmd.CommandText = "insert into Produtos(nome,preco,estoque) values
(@nome,@preco,@estoque); select @@IDENTITY;";
    cmd.Parameters.AddWithValue("@nome", produto.Nome);
    cmd.Parameters.AddWithValue("@preco", produto.Preco);
    cmd.Parameters.AddWithValue("@estoque", produto.Estoque);

    cn.Open();
    produto.Codigo = Convert.ToInt32(cmd.ExecuteScalar());

}
catch (SqlException ex)
{
    throw new Exception("Servidor SQL Erro: " + ex.Number);
}
catch (Exception ex)
{
    throw new Exception(ex.Message);
}
finally
{
    cn.Close();
}

public void Alterar(ProdutoInformation produto)
{
    //conexao
    SqlConnection cn = new SqlConnection();
    try
    {
        cn.ConnectionString = Dados.StringDeConexao;
        //command
        SqlCommand cmd = new SqlCommand();
        cmd.Connection = cn;
        cmd.CommandType = CommandType.Text;
```

Desenvolvimento em Camadas com C# .Net

```

    cmd.CommandText = "update Produtos set nome = @nome, preco =
@preco, estoque = @estoque where codigo = @codigo;";
    cmd.Parameters.AddWithValue("@codigo", produto.Codigo);
    cmd.Parameters.AddWithValue("@nome", produto.Nome);
    cmd.Parameters.AddWithValue("@preco", produto.Preco);
    cmd.Parameters.AddWithValue("@estoque", produto.Estoque);

    cn.Open();
    cmd.ExecuteNonQuery();

}

catch (SqlException ex)
{
    throw new Exception("Servidor SQL Erro: " + ex.Number);
}
catch (Exception ex)
{
    throw new Exception(ex.Message);
}
finally
{
    cn.Close();
}

}

public void Excluir(int codigo)
{
    //conexao
    SqlConnection cn = new SqlConnection();
    try
    {
        cn.ConnectionString = Dados.StringDeConexao;
        //command
        SqlCommand cmd = new SqlCommand();
        cmd.Connection = cn;
        cmd.CommandText = "delete from Produtos where codigo = " + codigo;

        cn.Open();
        int resultado = cmd.ExecuteNonQuery();
        if (resultado != 1)
        {
            throw new Exception("Não foi possível excluir o produto " + codigo);
        }
    }
}

```

O Projeto DAL

```

    catch (SqlException ex)
    {
        throw new Exception("SQL Erro:" + ex.Number);
    }
    catch (Exception ex)
    {
        throw new Exception(ex.Message);
    }
    finally
    {
        cn.Close();
    }
}

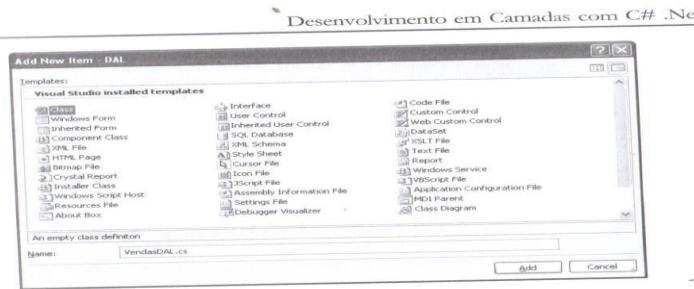
public DataTable Listagem(string filtro)
{
    DataTable tabela = new DataTable();
    string strSql;
    if (filtro == "")
    {
        strSql = "select * from produtos";
    }
    else
    {
        strSql = "select * from produtos where nome like '%" + filtro + "%'";
    }
    SqlDataAdapter da = new SqlDataAdapter(strSql, Dados.StringDeConexao);
    da.Fill(tabela);
    return tabela;
}

```

3.1.3 Classe VendasDAL.cs

Agora, vamos implementar a classe *VendasDAL.cs*:

- Clique com o botão direito do mouse no projeto *DAL*;
- Selecione as opções *Add>New Item...*
- Nomeie a classe como *VendasDAL.cs* e clique em *Add*.



→ Digite a listagem completa da classe *VendasDAL.cs* conforme o código a seguir:

```
(Arquivo Loja_Fontes\DAL\VendasDAL.cs)
using System;
using System.Collections.Generic;
using System.Text;
using System.Data;
using System.Data.SqlClient;
using Loja.Modelos;
using Loja.DAL;

namespace Loja.DAL
{
    public class VendasDAL
    {
        public DataTable ListaDeProdutos
        {
            get
            {
                SqlConnection cn = new SqlConnection();
                cn.ConnectionString = Dados.StringDeConexao;
                cn.Open();

                SqlDataAdapter da = new SqlDataAdapter("select * from produtos", cn);
                DataTable dt = new DataTable();

```

```

                cn.Close();
                return dt;
            }
        }

        //Propriedade que retorna uma Lista de Clientes
        public DataTable ListaDeClientes
        {
            get
            {
                SqlConnection cn = new SqlConnection();
                cn.ConnectionString = Dados.StringDeConexao;
                cn.Open();

                SqlDataAdapter da = new SqlDataAdapter("select * from clientes", cn);

                DataTable dt = new DataTable();
                da.Fill(dt);
                cn.Close();
                return dt;
            }
        }

        public void Incluir(VendaInformation venda)
        {
            //conexao
            SqlConnection cn = new SqlConnection();
            SqlTransaction t = null;
            try
            {
                cn.ConnectionString = Dados.StringDeConexao;
                //command
                SqlCommand cmd1 = new SqlCommand();
                cmd1.Connection = cn;
                cmd1.CommandText = @"insert into vendas
                (CodigoCliente,
                CodigoProduto,
                Data,
                Quantidade,
                Faturado)
                VALUES
                (@CodigoCliente,
                @CodigoProduto,
                @Data,
                @Quantidade,
                @Faturado)";
                cmd1.Parameters.AddWithValue("@CodigoCliente", venda.CodigoCliente);
                cmd1.Parameters.AddWithValue("@CodigoProduto", venda.CodigoProduto);
                cmd1.Parameters.AddWithValue("@Data", venda.Data);
                cmd1.Parameters.AddWithValue("@Quantidade", venda.Quantidade);
                cmd1.Parameters.AddWithValue("@Faturado", venda.Faturado);
                t = cn.BeginTransaction();
                cmd1.Transaction = t;
                cmd1.ExecuteNonQuery();
                t.Commit();
            }
            catch (Exception ex)
            {
                if (t != null)
                    t.Rollback();
                throw ex;
            }
        }
    }
}
```

```

@Quantidade,
@Faturado);select @@IDENTITY;';

SqlCommand cmd2 = new SqlCommand();
cmd2.Connection = cn;
cmd2.CommandText = @""Update Produtos
    Set Estoque = Estoque - @Quantidade
    Where Código=@CodigoProduto"";

cn.Open();

t = cn.BeginTransaction(IsolationLevel.Serializable);//default

cmd1.Transaction = t;
cmd2.Transaction = t;

cmd1.Parameters.AddWithValue("@codigocliente", venda.CódigoCliente);
cmd1.Parameters.AddWithValue("@codigoproduto", venda.CódigoProduto);
cmd1.Parameters.AddWithValue("@data", venda.Data);
cmd1.Parameters.AddWithValue("@quantidade", venda.Quantidade);
cmd1.Parameters.AddWithValue("@faturado", venda.Faturado);

cmd2.Parameters.AddWithValue("@quantidade", venda.Quantidade);
cmd2.Parameters.AddWithValue("@codigoproduto", venda.CódigoProduto);

venda.Código = Convert.ToInt32(cmd1.ExecuteScalar());
cmd2.ExecuteNonQuery();

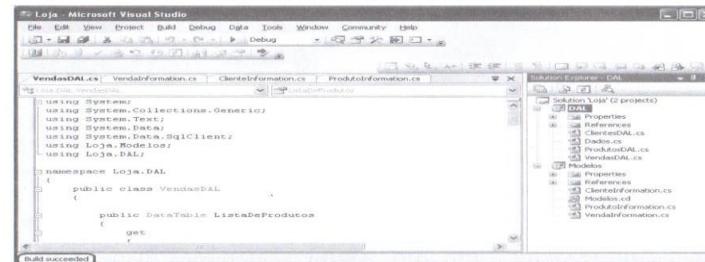
t.Commit();

}

catch (Exception ex)
{
    t.Rollback();
    throw new Exception("Erro no Servidor:" + ex.Message);
}
finally
{
    cn.Close();
}
}
}

```

→ Agora, clicando com o botão direito do mouse sobre o projeto *DAL*, dê um *Build* para compilar o projeto.



A frase *Build succeeded* indica que a compilação foi realizada com sucesso.

Parabéns!

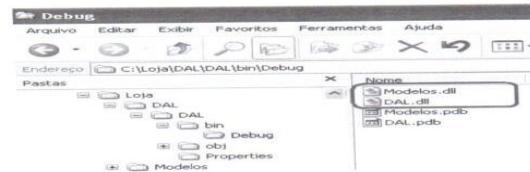
Acabamos de implementar todas as classes do nosso projeto *DAL* (Camada de Acesso a Dados).

Irei comentar brevemente alguns pontos interessantes sobre as classes que implementamos neste capítulo.

→ No método *Inserir* da classe *VendasDAL*, estamos usando uma transação. A inserção de uma venda só vai ocorrer se dois comandos forem concluídos com sucesso: a inserção na Tabela de Vendas e a atualização do estoque do produto;

→ Quando criamos a nossa classe *Dados*, usamos a palavra *static*. Com isso, nós não precisamos instanciar um objeto para ter acesso a propriedade *StringDeConexao*. Se você precisar criar uma propriedade deste tipo no VB .Net, pode usar a sintaxe: *Public Shared ReadOnly Property*;

→ Se você abrir o projeto *DAL* no Windows Explorer, poderá verificar que na pasta de dlls do projeto existem dois arquivos com a extensão *dll*. Um do próprio projeto *DAL* e outro para a referência que fizemos ao projeto *Modelos*.



Compare os dois arquivos do tipo dll que criamos com o nosso modelo em camadas a seguir:

[Modelos] DAL>BLL>User Interface

No próximo capítulo, onde implementaremos a Camada BLL (Regras de Negócio), teremos três arquivos dll na pasta do projeto *BLL* (Modelos.dll, DAL.dll e BLL.dll). Assim, poderemos distinguir claramente as nossas camadas, pois nesta solução elas estarão separadas em arquivos.

A idéia é aproveitarmos o código desenvolvido nas camadas anteriores. Mais adiante, veremos que quando for necessário fazer manutenção na camada de regras de negócios, por exemplo, precisaremos abrir e atualizar apenas o projeto *BLL*.

4

A Camada de Regras de Negócio BLL

4.1 Implementando as Classes da Camada BLL

Neste capítulo, vamos implementar as classes da Camada BLL (Business Logic Layer).

Criaremos as classes:

- ClientesBLL.cs
- ProdutosBLL.cs
- VendasBLL.cs

As regras de negócio definem como o seu negócio funciona. Essas regras podem abranger diversos assuntos como suas políticas, interesses, objetivos, compromissos éticos e sociais, obrigações contratuais, decisões estratégicas, leis e regulamentações, entre outros.

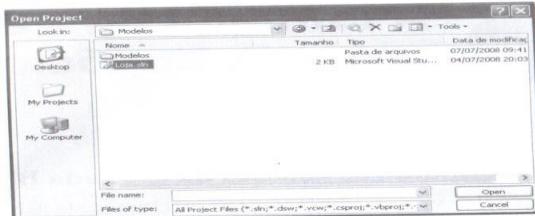
No nosso projeto *Loja*, vamos definir como regras de negócio:

- Regras para a inclusão de clientes:
 - O nome do cliente é obrigatório;
 - O e-mail do cliente será armazenado em letras minúsculas.
- Regras para a inclusão de produtos:
 - O nome do produto é obrigatório;
 - O preço não pode ser um valor negativo;
 - O estoque não pode ser um valor negativo.

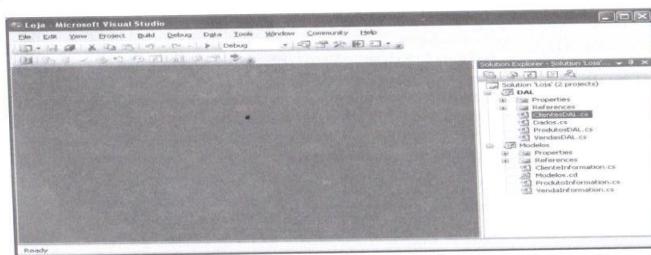
Vamos aos procedimentos :

- Abra o Microsoft Visual Studio;
- Clique em *File>Open>Project/Solution...*
- Na janela Open Project, selecione o arquivo da Solution: *C:\Loja\Modelos\Loja.sln*;
- Clique em *Open* para abrir a solução.

Desenvolvimento em Camadas com C# .Net

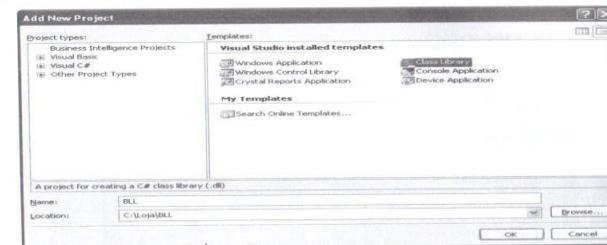


Neste ponto do projeto temos implementados os projetos *Modelos* e *DAL*:

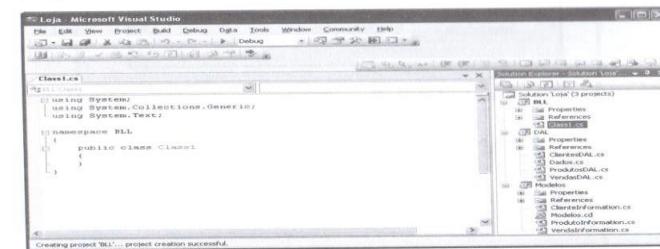


Vamos adicionar o projeto *BLL* à solução:

- Clique com o botão direito do mouse na Solution *Loja*;
 - Escolha *Add>New Project...*
- Na janela Add New Project, vamos informar os seguintes dados:
- No Tipo de Projeto, selecione *Visual C#*;
 - No Tipo de Template, selecione *Class Library*;
 - No Nome do Projeto, digite *BLL*;
 - Na Localização do Projeto, digite *C:\Loja\BLL*;

A Camada de Regras de Negócio *BLL*

O projeto *BLL* será criado:



Analisando o nosso modelo em camadas, podemos ver que a Camada *BLL* aproveita os projetos *Modelos* e *DAL*.

[*Modelos*] DAL>*BLL*>User Interface

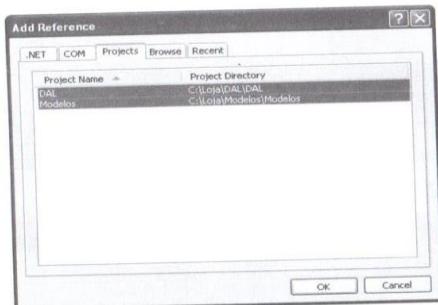
Assim, a primeira coisa que faremos no nosso projeto *BLL* será adicionar esses dois projetos como referência.

Para isso:

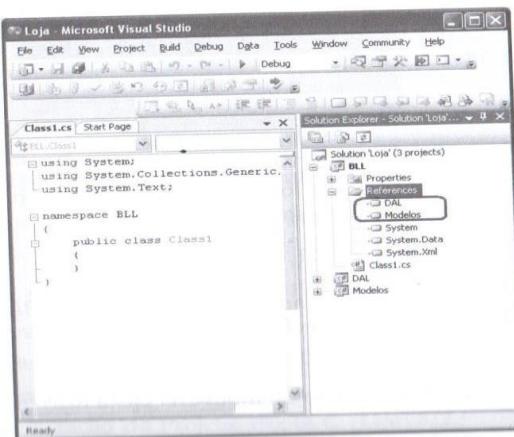
- Clique com o botão direito do mouse na pasta *References* do projeto *BLL*;
- Escolha a opção *Add Reference...*

Na janela Add Reference:

- Clique na aba *Projects*;
- Mantenha a tecla *<Ctrl>* pressionada e clique sobre os projetos *DAL* e *Modelos* para selecioná-los;
- Clique em *Ok* para adicionar as referências.



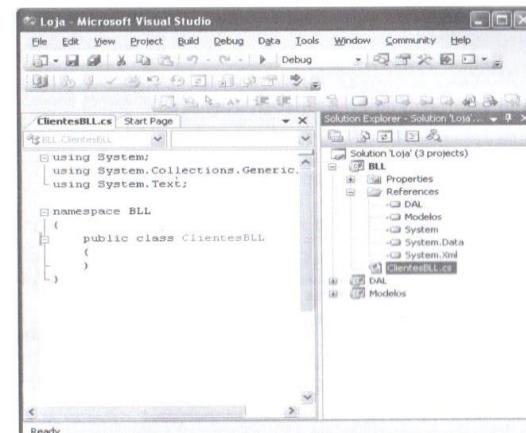
Podemos verificar as referências que acabamos de adicionar ao projeto *BLL*:



4.1.1 Classe ClientesBLL.cs

Vamos renomear a classe *Class1.cs* para *ClientesBLL.cs*:

- Clique com o botão direito do mouse sobre a classe *Class1.cs*;
- Escolha a opção *Rename* e renomeie a classe para *ClientesBLL.cs*.



Digite o seguinte código para a classe *ClientesBLL.cs*:

(Arquivo Loja_Fontes\BLL\ClientesBLL.cs)

```
using System;
using System.Data;
using Loja.Modelos;
using Loja.DAL;

namespace Loja.BLL
{
    public class ClientesBLL
    {
        public void Incluir(ClienteInformation cliente)
        {
            //O nome do cliente é obrigatório
            if (cliente.Nome.Trim().Length == 0)
            {

```

```

        throw new Exception("O nome do cliente é obrigatório");
    }

    //E-mail é sempre em letras minúsculas
    cliente.Email = cliente.Email.ToLower();

    //Se tudo está Ok, chama a rotina de inserção.
    ClientesDAL obj = new ClientesDAL();
    obj.Incluir(cliente);
}

public void Alterar(ClienteInformation cliente)
{
    //O nome do cliente é obrigatório
    if (cliente.Nome.Trim().Length == 0)
    {
        throw new Exception("O nome do cliente é obrigatório");
    }

    //E-mail é sempre em letras minúsculas
    cliente.Email = cliente.Email.ToLower();

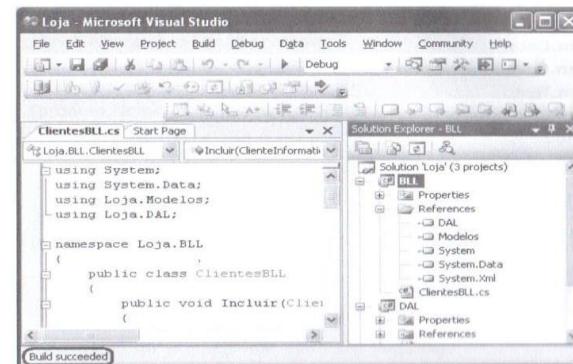
    //Se tudo está Ok, chama a rotina de alteração.
    ClientesDAL obj = new ClientesDAL();
    obj.Alterar(cliente);
}

public void Excluir(int codigo)
{
    if (codigo < 1)
    {
        throw new Exception("Selecione um cliente antes de excluí-lo.");
    }
    ClientesDAL obj = new ClientesDAL();
    obj.Excluir(codigo);
}

public DataTable Listagem(string filtro)
{
    ClientesDAL obj = new ClientesDAL();
    return obj.Listagem(filtro);
}

```

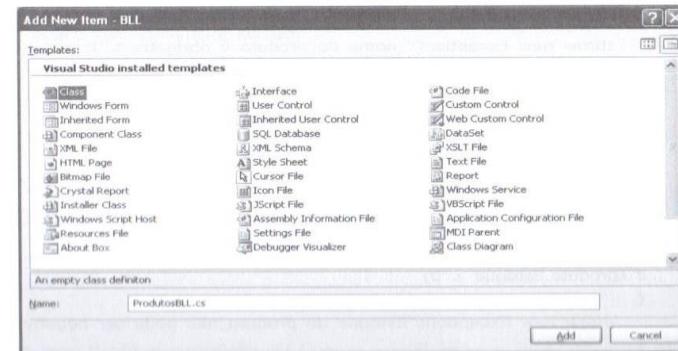
Dê um *Build* no projeto *BLL* para confirmar a inexistência de erros na codificação:



4.1.2 Classe ProdutosBLL.cs

Agora, vamos criar a classe *ProdutosBLL.cs*.

→ Adicione uma nova classe ao projeto *BLL* nomeando-a como *ProdutosBLL.cs*.



Digite o seguinte código para a classe *ProdutosBLL.cs*:

(Arquivo Loja_Fontes\BLL\ProdutosBLL.cs)

```

using System;
using System.Data;
using System.Collections.Generic;
using System.Text;
using System.Collections;
using Loja.Modelos;
using Loja.DAL;

namespace Loja.BLL
{
    public class ProdutosBLL
    {
        public ArrayList ProdutosEmFalta()
        {
            ProdutosDAL obj = new ProdutosDAL();
            return obj.ProdutosEmFalta();
        }

        public void Incluir(ProdutoInformation produto)
        {
            // Nome do produto é obrigatório
            if (produto.Nome.Trim().Length == 0)
            {
                throw new Exception("Nome do produto é obrigatório.");
            }

            // O preço do produto não pode ser negativo
            if (produto.Preco < 0)
            {
                throw new Exception("Preço do produto não pode ser negativo.");
            }

            // O estoque do produto não pode ser negativo
            if (produto.Estoque < 0)
            {
                throw new Exception("Estoque do produto não pode ser negativo.");
            }
        }
    }
}
```

```

ProdutosDAL obj = new ProdutosDAL();
obj.Incluir(produto);

}

public void Alterar(ProdutoInformation produto)
{
    // Nome do produto é obrigatório
    if (produto.Nome.Trim().Length == 0)
    {
        throw new Exception("O nome do produto é obrigatório.");
    }

    // O preço do produto não pode ser negativo
    if (produto.Preco < 0)
    {
        throw new Exception("Preço do produto não pode ser negativo.");
    }

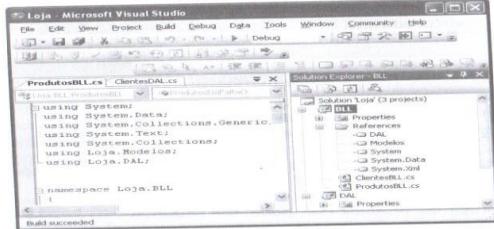
    // O estoque do produto não pode ser negativo
    if (produto.Estoque < 0)
    {
        throw new Exception("Estoque do produto não pode ser negativo.");
    }

    // Se tudo estiver ok, chama a rotina de alteração
    ProdutosDAL obj = new ProdutosDAL();
    obj.Alterar(produto);
}

public void Excluir(int codigo)
{
    ProdutosDAL obj = new ProdutosDAL();
    obj.Excluir(codigo);
}

public DataTable Listagem(string filtro)
{
    ProdutosDAL obj = new ProdutosDAL();
    return obj.Listagem(filtro);
}
}
```

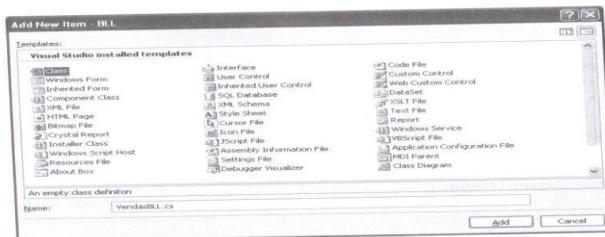
Dê um *Build* no projeto *BLL* para confirmar a inexistência de erros na codificação:



4.1.3 Classe VendasBLL.cs

Agora, vamos implementar a classe *VendasBLL.cs*:

- Adicione uma nova classe ao projeto *BLL* nomeando-a como *VendasBLL.cs*.



Digite o seguinte código para a classe *VendasBLL.cs*:

(Arquivo Loja_Fontes\BLL\VendasBLL.cs)

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data;
using Loja.DAL;
```

A Camada de Regras de Negócio *BLL*

```
namespace Loja.BLL
{
    public class VendasBLL
    {
        //Este é um campo privado para armazenar uma instância da classe DAL.
        private VendasDAL objDAL;

        //Esse é o construtor da classe VendasBLL()
        public VendasBLL()
        {
            objDAL = new VendasDAL();
        }

        public DataTable ListaDeProdutos
        {
            get
            {
                return objDAL.ListaDeProdutos;
            }
        }

        public DataTable ListaDeClientes
        {
            get
            {
                return objDAL.ListaDeClientes;
            }
        }

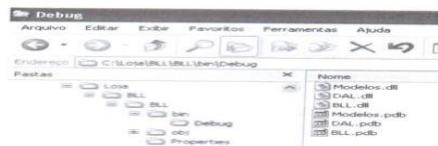
        public void Incluir(VendaInformation venda)
        {
            objDAL.Incluir(venda);
        }
    }
}
```

Dê um *Build* no projeto *BLL* para confirmar a inexistência de erros na codificação.

Parabéns! Finalizamos a implementação da Camada *BLL* – Regras de negócio.

Se você abrir o projeto *BLL* no Windows Explorer, poderá verificar que na pasta de *dlls* do projeto, existem três arquivos com a

extensão dll. Um do próprio projeto *BLL* e outros para as referências que fizemos aos projetos *Modelos* e *DAL*.



Compare isso com o nosso modelo em camadas a seguir:

[Modelos] DAL>BLL>User Interface

Veremos que a próxima camada a ser implementada é a interface com o usuário. No próximo capítulo, vamos iniciar a construção de um aplicativo para Microsoft Windows (Winforms) que vai utilizar todas as camadas já desenvolvidas.

5

Formulário de Clientes

Neste capítulo, vamos iniciar a implementação da camada de interface do usuário construindo um aplicativo para Microsoft Windows. Iremos criar o Formulário de Clientes utilizando todas as camadas já desenvolvidas nos capítulos anteriores.

5.1 Criando a Interface com o Usuário

[Modelos] DAL>BLL>User Interface

Abra o Microsoft Visual Studio:

→ Clique em *File>Open>Project/Solution...*

Na janela Open Project:

→ Selecione o arquivo da nossa Solution: *C:\Loja\Modelos\Loja.sln*;

→ Clique em *Open* para abrir a solução.

Neste momento, temos implementados os projetos: *Modelos*, *DAL* e *BLL*:

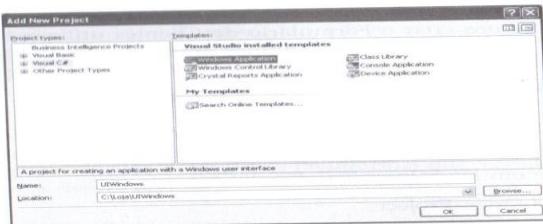


Vamos adicionar um projeto de user interface para Windows chamado *UIWindows*:

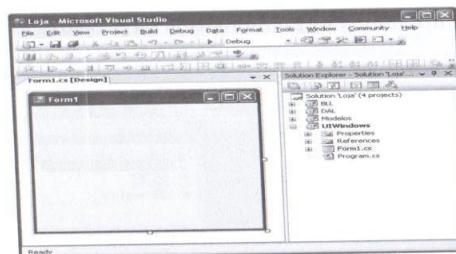
→ Clique com o botão direito do mouse na Solution *Loja*;

→ Escolha a opção: *Add>New Project...*

- Na janela Add New Project, vamos informar os seguintes dados:
- No Tipo de Projeto, selecione *Visual C#*;
 - No Tipo de Template, selecione *Windows Application*;
 - No Nome do Projeto, digite *UIWindows*;
 - Na Localização do Projeto, digite *C:\Loja\ UIWindows*;
 - Clique em *Ok* para adicionar o projeto.



O aplicativo para Windows *UIWindows* será criado:



5.2 Adicionando Referência aos Projetos

A primeira coisa a ser feita é adicionar os projetos que usaremos

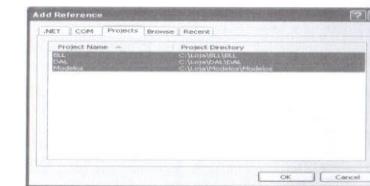
Olhando para o nosso modelo de camadas a seguir, verificamos que é preciso adicionar referências aos projetos *Modelos*, *DAL* e *BLL*.

[Modelos] DAL>BLL>User Interface

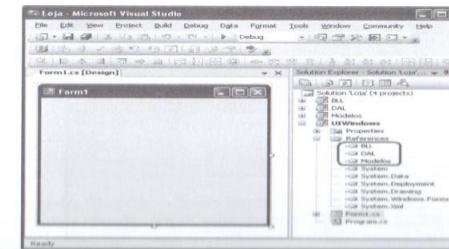
- Clique com o botão direito do mouse na pasta *References* do projeto *UIWindows*;
- Escolha a opção *Add Reference...*

Na janela Add Reference:

- Clique na aba *Projects*, mantenha a tecla *<Ctrl>* pressionada;
- Clique sobre os projetos *DAL*, *Modelos* e *BLL* para selecioná-los;
- Clique em *Ok* para adicionar as referências.



Podemos verificar as referências que acabamos de adicionar ao projeto *UIWindows*:



No nosso formulário principal, vamos criar um menu para que o usuário possa acessar os demais formulários da aplicação.

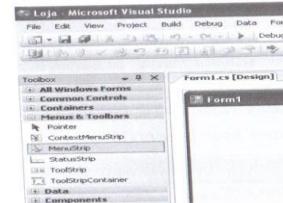
Vamos exibir a Toolbox (caixa de ferramentas) que contém o menu que precisaremos incluir.

Para exibir a Toolbox você pode:

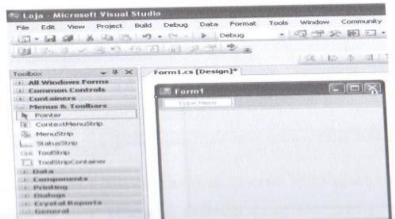
- Utilizar a combinação de teclas `<Ctrl>+<Alt>+X`;
- Utilizar o menu `View>Toolbox`.

Na Toolbox:

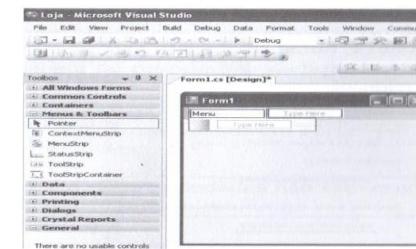
- Abra o item `Menus & Toolbars`;
- Para criar o menu, arraste o componente `MenuStrip` para o formulário `Form1`.



Ao soltar o `MenuStrip` dentro do formulário `Form1`, um menu vazio será criado, clique na área com a indicação `Type here` e digite `Menu`.

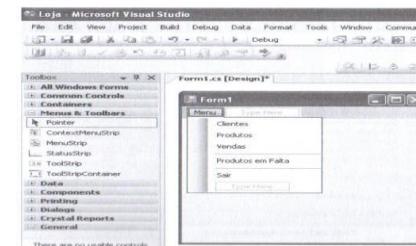


Ao clicar no campo para digitar `Menu`, percebemos que dois campos contendo `Type Here` se abrem: um a seguir e outro à direita do item `Menu`. Estes campos aparecem para o caso de querermos incluir mais itens no menu.



Insira os itens de menu `Clients`, `Products`, `Sales`, `Products in Fault` e `Exit` como na figura a seguir:

Dica: Use o hifen (-) para criar os traços de divisão que você está vendo na figura.



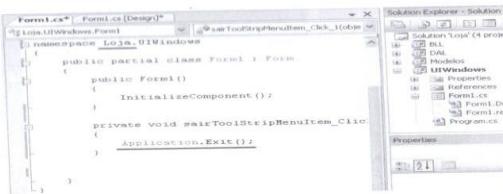
→ Dê um duplo clique com o botão esquerdo do mouse sobre o item `Sair` do menu para codificarmos o que deve acontecer quando o usuário selecionar essa opção.

Dentro do método do item de menu *Sair* que se abrirá, insira o seguinte código:

```
Application.Exit();
```

Desta forma, sempre que o usuário clicar na opção *Sair*, a aplicação será encerrada.

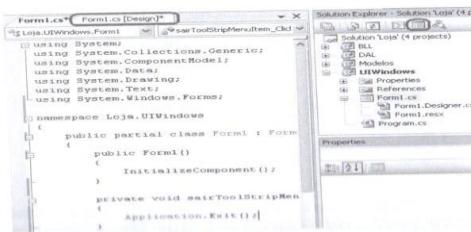
Acrescente também o nome do projeto *Loja* no namespace como a seguir:



5.3 Propriedades do Objeto

Agora, iremos para o modo de Design para visualizar novamente o formulário.

Para acessar o modo de Design, utilize a aba *[Design]* do *Form1* ou o botão *View Design* do Solution Explorer como indicado a seguir:



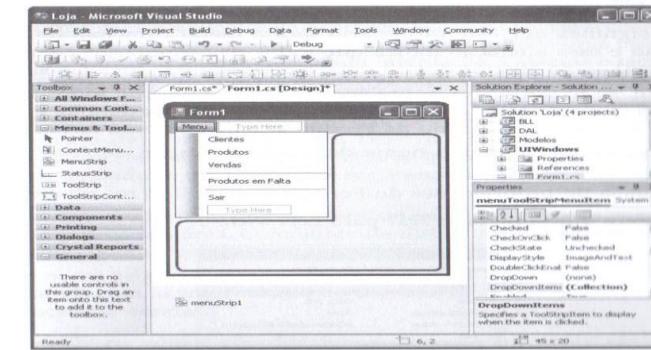
No modo de Design do formulário *Form1* percebemos que o *MenuStrip* é o objeto que está selecionado atualmente.

O MS Visual Studio sempre exibe na janela chamada *Properties*, as propriedades do último objeto que foi selecionado. Para selecionar um objeto, basta clicar sobre o mesmo.

Agora, queremos alterar uma propriedade do objeto *Form1*. Para selecioná-lo e fazer com que as propriedades do formulário *Form1* sejam exibidas na janela de propriedades, clique na barra de títulos ou em qualquer área do formulário *Form1* fora do objeto menu destacado a seguir:

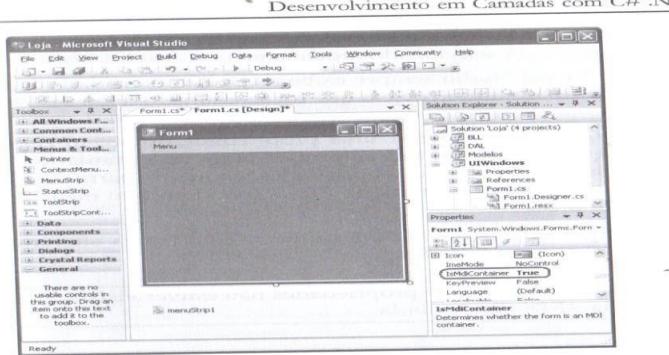
→ Clique em uma área do formulário *Form1* para alterarmos uma de suas propriedades;

Dica: Se a janela de propriedades não estiver visível, pressione a tecla <F4> para exibi-la.



Na janela de propriedades do *Form1*:

→ Altere a propriedade *IsMdiContainer* para *True*.



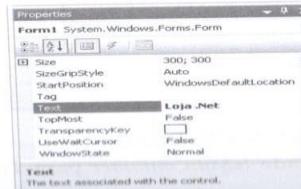
Isso significa que este formulário será o menu principal da nossa aplicação e essa alteração indica que todos os formulários chamados pelo *Form1* serão abertos dentro dele, e não em uma nova janela.

Assim, se fecharmos o formulário do *Menu Principal*, todos os formulários que estiverem abertos dentro da aplicação (exemplo: Formulário de Clientes, Formulário de Produtos) também serão fechados.

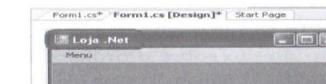
Agora, vamos alterar o nome da nossa aplicação de *Form1* para *Loja .Net*:

Na janela de propriedades do *Form1*:

→ Altere a propriedade *Text* para *Loja .Net*.



Isso altera automaticamente o nome do formulário exibido na barra de títulos.



Para testarmos tudo o que já desenvolvemos até agora, vamos criar quatro formulários:

- Clientes;
- Produtos;
- Vendas;
- Produtos em Falta.

Um dos fatores mais importantes para a popularidade de um software é uma interface atrativa e fácil de usar.

A forma como os objetos são apresentados ao usuário é chamada de design. O usuário final não sabe que nós construímos uma classe para acesso ao banco de dados utilizando orientação a objetos e técnicas de reaproveitamento de código, o que ele sabe é se o software é fácil de usar ou não.

A aparência do nosso software é implementada no que chamamos de Camada de Apresentação (ou *User Interface*). Não adianta nada construir um software com uma excelente infra-estrutura, rápido acesso a banco de dados e uma excelente segurança interna se o usuário tiver dificuldade em utilizá-lo.

Isso é tão importante que existem até cursos superiores para formar profissionais nessa área. O conjunto de disciplinas sobre esse assunto é conhecido como Design Digital.

No nosso projeto temos diferentes maneiras de permitir que o usuário interaja com as informações de clientes, informações de produtos e realização de vendas.

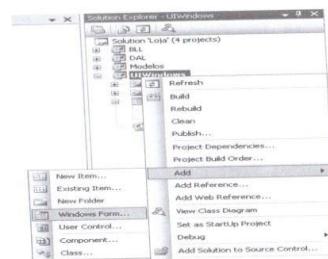
Nesse livro, vamos abordar duas formas de apresentação:

- a interação através de uma aplicação para Windows (também conhecida como Winforms);
- a interação através de páginas Web (ou Webpages).

Nessa parte, daremos início a implementação da nossa aplicação para Windows.

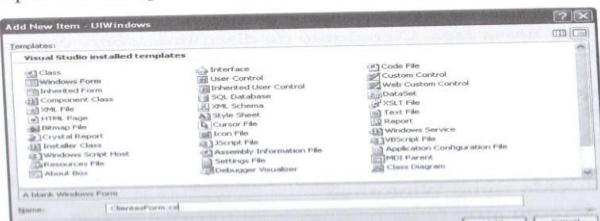
5.4 Criando Formulário de Clientes

- Clique com o botão direito sobre o projeto *UIWindows*;
- Escolha a opção *Add>Windows Form...*



Na janela Add New Item:

- Selecione o template *Windows Form*;
- Nomeie o novo formulário como *ClientsForm.cs*;
- Clique em *Add* para adicionar o formulário.



Observações: Certifique-se de que o namespace esteja como *Loja.UIWindows* tanto em *ClientsForm.cs* quanto em *ClientsForm.Designer.cs*.

Colocar *using Loja.BLL* no inicio do code behind (código-fonte) do formulário *ClientsForm.cs*.

Na janela de propriedades do formulário *ClientsForm*, defina as propriedades como a seguir:

→ Size: 448; 512

No menu Common Controls da toolbox, arraste quatro objetos do tipo *Label* para o formulário *ClientsForm*.

→ Defina as propriedades dos quatro labels como a seguir:

Name: **codigoLabel**

Location: 16;32

Text: **Código:**

Name: **nomeLabel**

Location: 16;64

Text: **Nome:**

Name: **emailLabel**

Location: 16;96

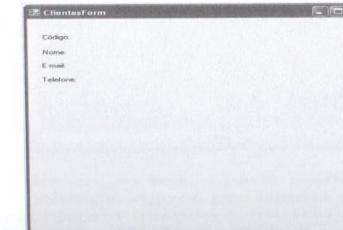
Text: **E-mail:**

Name: **telefoneLabel**

Location: 16;128

Text: **Telefone:**

Neste momento, o Formulário de Clientes estará com esta aparência:



A partir do menu Common Controls da toolbox, arraste cinco objetos do tipo *TextBox* para o formulário *CientesForm*.

→ Defina as propriedades dos cinco *TextBox* como a seguir:

Name: **codigoTextBox**
 Enable: **False**
 Location: **110;29**
 ReadOnly: **True**
 Size: **70;20**

Name: **nomeTextBox**
 Location: **110;61**
 Size: **200;20**

Name: **emailTextBox**
 Location: **110;93**
 Size: **200;20**

Name: **telefoneTextBox**
 Location: **110;125**
 Size: **200;20**

Anchor: **Bottom, Left**
 Name: **txtFiltro**
 Location: **24; 448**
 Size: **161; 20**

A partir do menu Common Controls da toolbox, arraste cinco objetos do tipo *Button* para o formulário *CientesForm*.

→ Defina as propriedades dos cinco Buttons como a seguir:

Name: **limparButton**
 Location: **110;159**
 Size: **75;23**
 Text: **Limpar**

Name: **incluirButton**
 Location: **191;159**
 Size: **75;23**
 Text: **Incluir**

Name: **alterarButton**
 Location: **272;159**
 Size: **75;23**

Name: **excluirButton**
 Location: **353;159**
 Size: **75;23**
 Text: **Excluir**

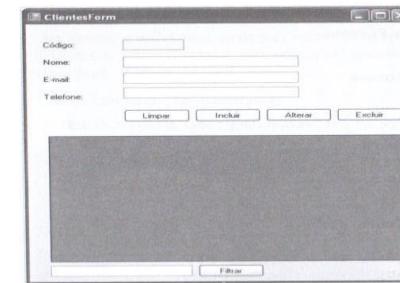
Name: **btFiltro**
 Anchor: **Bottom, Left**
 Location: **192; 447**
 Size: **75;23**
 Text: **Filtrar**

A partir do menu Data da toolbox, arraste um objeto do tipo *DataGridView* para o formulário *CientesForm*.

→ Defina as propriedades do *DataGridView* como a seguir:

Name: **clientesDataGridView**
 Anchor: **Top, Bottom, Left, Right**
 Location: **24; 208**
 Size: **408; 233**

Agora, o Formulário de Clientes está assim:



Muito bem! O design, ou seja, a parte gráfica do nosso Formulário de Clientes está finalizada. Agora, precisamos programar as ações que desejamos que aconteçam quando o usuário abrir esse formulário e interagir clicando nos botões. Vamos lá?

Para ver o código-fonte (também chamado de code behind):
 → Clique com o botão direito do mouse sobre o formulário *CientesForm* no Solution Explorer;
 → Escolha a opção *View Code*.

Após o método *CientesForm_Load()*, vamos criar o método *AtualizaGrid()*. No *AtualizaGrid*, faremos a comunicação com a Camada de Regras de Negócio (BLL) para que possamos preencher os valores dos objetos TextBox.

Dentro do método *CientesForm_Load()*, que é chamado toda vez que o Formulário de Clientes é carregado em memória, vamos fazer uma chamada para o método *AtualizaGrid()* e também posicionaremos o cursor no primeiro campo do formulário.

Digite o trecho de código para o code behind do formulário *CientesForm*:

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

namespace Loja.UIWindows
{
  public partial class ClientesForm : Form
  {
    public ClientesForm()
    {
      InitializeComponent();
    }

    public void AtualizaGrid()
    {
      // Comunicação com a Camada BLL
      ClientesBLL obj = new ClientesBLL();
      clientesDataGridView.DataSource = obj.Listagem(txtFiltro.Text);

      // Atualizando os objetos TextBox
      try
      {
        codigoTextBox.Text = clientesDataGridView[0, clientesDataGridView.CurrentRow.Index].Value.ToString();
        nomeTextBox.Text = clientesDataGridView[1, clientesDataGridView.CurrentRow.Index].Value.ToString();
        emailTextBox.Text = clientesDataGridView[2, clientesDataGridView.CurrentRow.Index].Value.ToString();
        telefoneTextBox.Text = clientesDataGridView[3, clientesDataGridView.CurrentRow.Index].Value.ToString();
      }
      catch
      {
        codigoTextBox.Text = " ";
        nomeTextBox.Text = " ";
        emailTextBox.Text = " ";
        telefoneTextBox.Text = " ";
      }
    }
  }
}
```

```
codigoTextBox.Text = clientesDataGridView[0, clientesDataGridView.CurrentRow.Index].Value.ToString();
nomeTextBox.Text = clientesDataGridView[1, clientesDataGridView.CurrentRow.Index].Value.ToString();
emailTextBox.Text = clientesDataGridView[2, clientesDataGridView.CurrentRow.Index].Value.ToString();
telefoneTextBox.Text = clientesDataGridView[3, clientesDataGridView.CurrentRow.Index].Value.ToString();

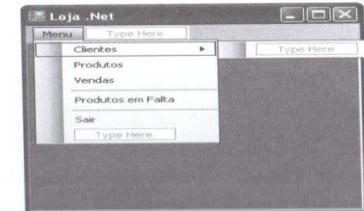
}
catch
{
  codigoTextBox.Text = " ";
  nomeTextBox.Text = " ";
  emailTextBox.Text = " ";
  telefoneTextBox.Text = " ";
}

private void ClientesForm_Load(object sender, EventArgs e)
{
  AtualizaGrid();
  nomeTextBox.Focus();
}
```

Para testarmos se essa exibição inicial dos dados dos clientes está funcionando, precisamos criar uma chamada para este formulário a partir do menu principal. Para isso:

→ Abra o *Form1* em modo de Design;

→ Clique sobre o menu para que possamos ver as opções existentes:



- Dê um duplo clique sobre a opção *Cientes* para codificarmos a chamada para o formulário correspondente;
- O MS Visual Studio abrirá o código do formulário com o cursor dentro do método que será executado quando o usuário clicar no item *Cientes* do Menu. Dentro deste método (chamado *clientesToolStripMenuItem_Click*), digite o seguinte código:

```
ClientesForm obj = new ClientesForm();
obj.MdiParent = this;
obj.Show();
```

A listagem completa do nosso code behind Form1.cs ficou assim:

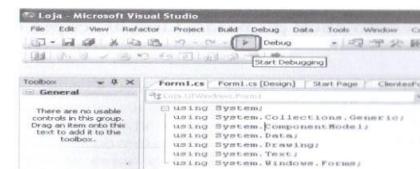
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace Loja.UIWindows
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void sairToolStripMenuItem_Click_1(object sender, EventArgs e)
        {
            Application.Exit();
        }

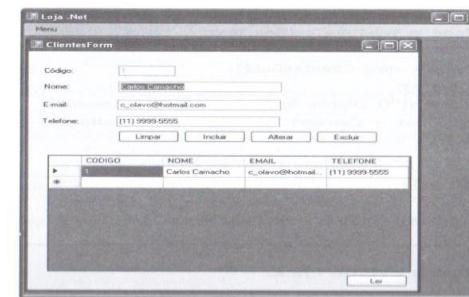
        private void clientesToolStripMenuItem_Click(object sender, EventArgs e)
        {
            ClientesForm obj = new ClientesForm();
            obj.MdiParent = this;
            obj.Show();
        }
    }
}
```

- Dê um *Start Debugging* clicando na seta verde (na tela do seu computador) da barra de botões conforme indicado a seguir. Essa opção compila o projeto e, se tudo estiver certo, já executa a nossa aplicação exibindo o menu principal.



Observações: Se a aplicação não abrir, clique com o botão direito sobre o projeto *UIWindows* e selecione a opção *Set as StartUp Project*. Com isso estamos indicando que, na nossa solução, o projeto *UIWindows* será o primeiro a ser executado.

Na aplicação *Loja .Net* em execução, clique no menu escolhendo a opção *Menu>Cientes* para abrir o formulário de clientes:



Podemos ver que a carga inicial de dados funcionou conforme o esperado.

Agora, vamos codificar as funcionalidades dos botões do Formulário de Clientes:

- Dê um duplo clique no botão *Limpar* para codificarmos o evento click;
- Dentro do evento click do botão Limpar, copie e cole o código a seguir:

```
codigoTextBox.Text = " ";
nomeTextBox.Text = " ";
emailTextBox.Text = " ";
telefoneTextBox.Text = " ";
```

- Dê um duplo clique no botão *Incluir* para codificarmos o evento click;

- Dentro do evento click do botão Incluir, copie e cole o código a seguir:

```
try
{
    ClienteInformation cliente = new ClienteInformation();
    cliente.Nome = nomeTextBox.Text;
    cliente.Email = emailTextBox.Text;
    cliente.Telefone = telefoneTextBox.Text;

    ClientesBLL obj = new ClientesBLL();
    obj.Incluir(cliente);
    MessageBox.Show("O cliente foi incluído com sucesso!");
    codigoTextBox.Text = Convert.ToString(cliente.Codigo);
    AtualizaGrid();
}
catch (Exception ex)
{
    MessageBox.Show("Erro: " + ex.Message);
}
```

Para alterarmos o evento click:

- Dê um duplo clique no botão *Alterar* para codificarmos o evento click;

- Dentro do evento click do botão Alterar, copie e cole o código a seguir:

```
if (codigoTextBox.Text.Length == 0)
{
    MessageBox.Show("Um cliente deve ser selecionado para alteração.");
}
else
try
{
    ClienteInformation cliente = new ClienteInformation();
    cliente.Codigo = int.Parse(codigoTextBox.Text);
    cliente.Nome = nomeTextBox.Text;
    cliente.Email = emailTextBox.Text;
    cliente.Telefone = telefoneTextBox.Text;

    ClientesBLL obj = new ClientesBLL();
    obj.Alterar(cliente);
    MessageBox.Show("O cliente foi alterado com sucesso!");
    AtualizaGrid();
}
catch (Exception ex)
{
    MessageBox.Show("Erro: " + ex.Message);
}
```

Para excluir:

- Dê um duplo clique no botão *Excluir* para codificarmos o evento click;
- Dentro do evento click do botão Excluir, copie e cole o código a seguir:

```
if (codigoTextBox.Text.Length == 0)
{
    MessageBox.Show("Um cliente deve ser selecionado antes da exclusão.");
}
else
try
{
    int codigo = Convert.ToInt32(codigoTextBox.Text);
    ClientesBLL obj = new ClientesBLL();
    obj.Excluir(codigo);
```

```

    MessageBox.Show("O cliente foi excluído com sucesso!");
    AtualizaGrid();
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}

```

Para filtrar:

- Dê um duplo clique no botão *Filtrar* para codificarmos o evento *click*;
- Dentro do evento click do botão *Filtrar*, copie e cole o código a seguir:

```

// Comunicação com a Camada BLL
ClientesBLL obj = new ClientesBLL();
clientesDataGridView.DataSource = obj.Listagem(txtFiltro.Text);

// Atualizando os objetos TextBox
try {
    codigoTextBox.Text = clientesDataGridView[0, clientesDataGridView.Current
Row.Index].Value.ToString();
    nomeTextBox.Text = clientesDataGridView[1, clientesDataGridView.Current
Row.Index].Value.ToString();
    emailTextBox.Text = clientesDataGridView[2, clientesDataGridView.Current
Row.Index].Value.ToString();
    telefoneTextBox.Text = clientesDataGridView[3, clientesDataGridView.Current
Row.Index].Value.ToString();
}
catch {
    codigoTextBox.Text = " ";
    nomeTextBox.Text = " ";
    emailTextBox.Text = " ";
    telefoneTextBox.Text = " ";
}

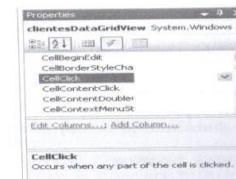
```

Para exibir a lista de eventos:

- Clique sobre o *dataGridView*;

Na janela de propriedades:

- Clique no ícone (relâmpago) para exibir a lista de eventos;
- Dê um duplo clique no evento *CellClick* para inserirmos o código para esse evento;



→ Dentro do evento *CellClick*, copie e cole o código a seguir:

```

// Atualizando os objetos TextBox
codigoTextBox.Text = clientesDataGridView[0, clientesDataGridView.CurrentRow.
Index].Value.ToString();
nomeTextBox.Text = clientesDataGridView[1, clientesDataGridView.CurrentRow.I
ndex].Value.ToString();
emailTextBox.Text = clientesDataGridView[2, clientesDataGridView.CurrentRow.
Index].Value.ToString();
telefoneTextBox.Text = clientesDataGridView[3, clientesDataGridView.Current
RowIndex].Value.ToString();

```

O code behind completo do *ClientesForm.cs* ficou assim:

```

(Arquivo Loja_Fontes\UIWindows\ClientesForm.cs)

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using Loja.BLL;
using Loja.DAL;
using Loja.Modelos;
using System.Xml;
using System.Xml.Schema;

namespace Loja.UIWindows
{
    public partial class ClientesForm : Form
    {
        public ClientesForm()
        {

```

```

        InitializeComponent();
    }

    public void AtualizaGrid()
    {
        // Comunicação com a Camada BLL
        ClientesBLL obj = new ClientesBLL();
        clientesDataGridView.DataSource = obj.Listagem(txtFiltro.Text);

        // Atualizando os objetos TextBox
        try
        {
            codigoTextBox.Text = clientesDataGridView[0, clientesDataGridView.
CurrentRow.Index].Value.ToString();
            nomeTextBox.Text = clientesDataGridView[1, clientesDataGridView.
CurrentRow.Index].Value.ToString();
            emailTextBox.Text = clientesDataGridView[2, clientesDataGridView.
CurrentRow.Index].Value.ToString();
            telefoneTextBox.Text = clientesDataGridView[3, clientesDataGridView.
CurrentRow.Index].Value.ToString();
        }
        catch
        {
            codigoTextBox.Text = " ";
            nomeTextBox.Text = " ";
            emailTextBox.Text = " ";
            telefoneTextBox.Text = " ";
        }
    }

    private void ClientesForm_Load(object sender, EventArgs e)
    {
        AtualizaGrid();
        nomeTextBox.Focus();
    }

    private void limparButton_Click(object sender, EventArgs e)
    {
        codigoTextBox.Text = " ";
        nomeTextBox.Text = " ";
        emailTextBox.Text = " ";
        telefoneTextBox.Text = " ";
    }

    private void clienteButton_Click(object sender, EventArgs e)
    {
        try
        {
            codigoTextBox.Text = clientesDataGridView[0, clientesDataGridView.
CurrentRow.Index].Value.ToString();
            nomeTextBox.Text = clientesDataGridView[1, clientesDataGridView.
CurrentRow.Index].Value.ToString();
            emailTextBox.Text = clientesDataGridView[2, clientesDataGridView.
CurrentRow.Index].Value.ToString();
            telefoneTextBox.Text = clientesDataGridView[3, clientesDataGridView.
CurrentRow.Index].Value.ToString();
        }
        catch
        {
            codigoTextBox.Text = " ";
            nomeTextBox.Text = " ";
            emailTextBox.Text = " ";
            telefoneTextBox.Text = " ";
        }
    }

    private void pesquisarButton_Click(object sender, EventArgs e)
    {
        try
        {
            AtualizaGrid();
        }
        catch
        {
            MessageBox.Show("O filtro não pode ser vazio!");
        }
    }
}

```

```

try
{
    ClienteInformation cliente = new ClienteInformation();
    cliente.Nome = nomeTextBox.Text;
    cliente.Email = emailTextBox.Text;
    cliente.Telefone = telefoneTextBox.Text;

    ClientesBLL obj = new ClientesBLL();
    obj.Incluir(cliente);
    MessageBox.Show("O cliente foi incluído com sucesso!");
    codigoTextBox.Text = Convert.ToString(cliente.Codigo);
    AtualizaGrid();

}
catch (Exception ex)
{
    MessageBox.Show("Erro: " + ex.Message);
}

private void alterarButton_Click(object sender, EventArgs e)
{
    if (codigoTextBox.Text.Length == 0)
    {
        MessageBox.Show("Um cliente deve ser selecionado para alteração.");
    }
    else
    try
    {
        ClienteInformation cliente = new ClienteInformation();
        cliente.Codigo = int.Parse(codigoTextBox.Text);
        cliente.Nome = nomeTextBox.Text;
        cliente.Email = emailTextBox.Text;
        cliente.Telefone = telefoneTextBox.Text;

        ClientesBLL obj = new ClientesBLL();
        obj.Alterar(cliente);
        MessageBox.Show("O cliente foi alterado com sucesso!");
        AtualizaGrid();

    }
    catch (Exception ex)
    {
        MessageBox.Show("Erro: " + ex.Message);
    }
}

```

```

        }

        private void excluirButton_Click(object sender, EventArgs e)
        {
            if (codigoTextBox.Text.Length == 0)
            {
                MessageBox.Show("Um cliente deve ser selecionado antes da exclusão.");
            }
            else
            {
                try
                {
                    int codigo = Convert.ToInt32(codigoTextBox.Text);
                    ClientesBLL obj = new ClientesBLL();
                    obj.Excluir(codigo);
                    MessageBox.Show("O cliente foi excluído com sucesso!");
                    AtualizaGrid();
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message);
                }
            }
        }

        private void clientesDataGridView_CellClick(object sender, DataGridViewCellEventArgs e)
        {
            // Atualizando os objetos TextBox
            codigoTextBox.Text = clientesDataGridView[0, clientesDataGridView.CurrentRow.Index].Value.ToString();
            nomeTextBox.Text = clientesDataGridView[1, clientesDataGridView.CurrentRow.Index].Value.ToString();
            emailTextBox.Text = clientesDataGridView[2, clientesDataGridView.CurrentRow.Index].Value.ToString();
            telefoneTextBox.Text = clientesDataGridView[3, clientesDataGridView.CurrentRow.Index].Value.ToString();
        }

        private void btFiltro_Click(object sender, EventArgs e)
        {
            // Comunicação com a Camada BLL
            ClientesBLL obj = new ClientesBLL();

```

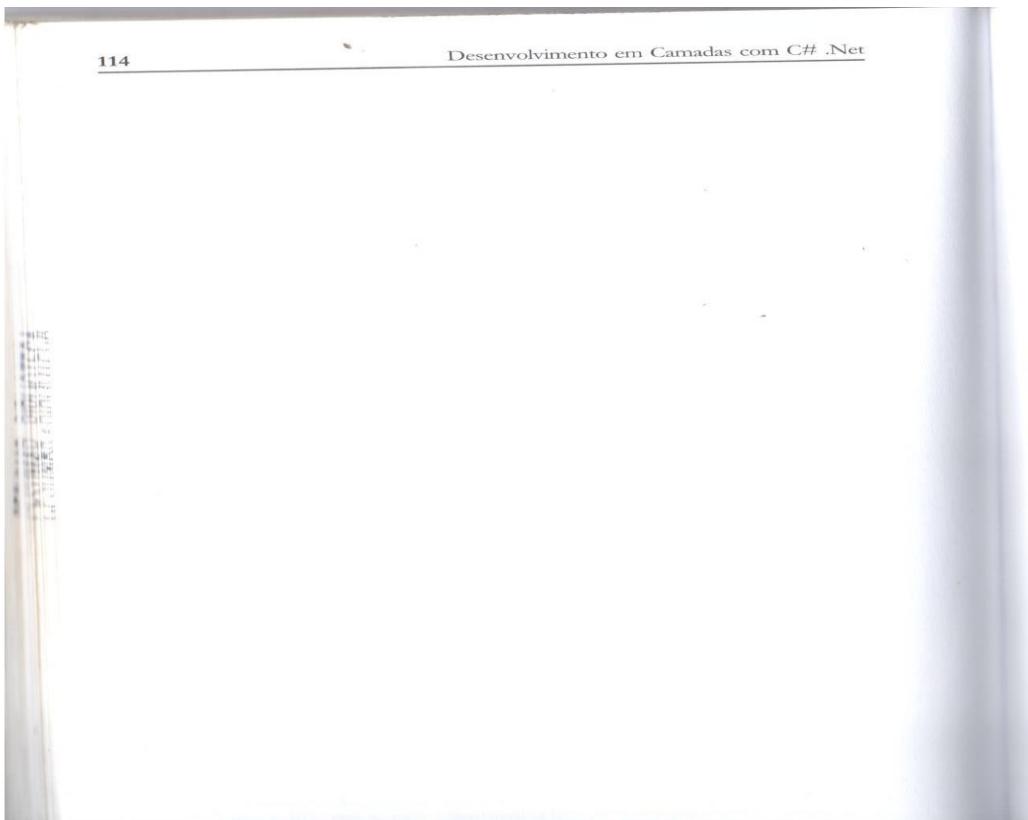
```

            // Atualizando os objetos TextBox
            try
            {
                codigoTextBox.Text = clientesDataGridView[0, clientesDataGridView.CurrentRow.Index].Value.ToString();
                nomeTextBox.Text = clientesDataGridView[1, clientesDataGridView.CurrentRow.Index].Value.ToString();
                emailTextBox.Text = clientesDataGridView[2, clientesDataGridView.CurrentRow.Index].Value.ToString();
                telefoneTextBox.Text = clientesDataGridView[3, clientesDataGridView.CurrentRow.Index].Value.ToString();
            }
            catch
            {
                codigoTextBox.Text = " ";
                nomeTextBox.Text = " ";
                emailTextBox.Text = " ";
                telefoneTextBox.Text = " ";
            }
        }
    }

```

Parabéns! Você acabou de implementar o Formulário de Clientes do projeto *Loja .Net*. Agora é só executar o projeto, acessar o Formulário de Clientes e testar todas as funcionalidades que você implementou.

No próximo capítulo, vamos implementar o Formulário de Produtos.



6

Formulário de Produto

Neste capítulo, vamos criar o Formulário de Produtos do nosso projeto.

Para tanto, siga os passos:

Abra o Microsoft Visual Studio:

→ Clique em *File >Open>Project/Solution...*

Na janela Open Project:

→ Selecione o arquivo da nossa Solution *C:\Loja\Modelos\Loja.sln*;

→ Clique em *Open* para abrir a solução.

Vamos adicionar um formulário chamado *ProdutosForm.cs*:

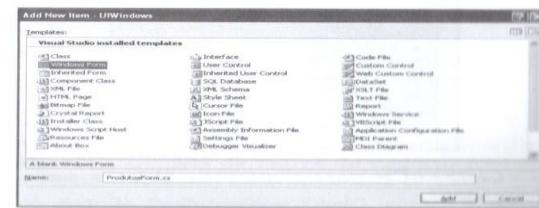
→ Clique com o botão direito no projeto *UIWindows*;

→ Escolha a opção *Add>Windows Form...*

Na janela Add New Item, vamos informar os seguintes dados:

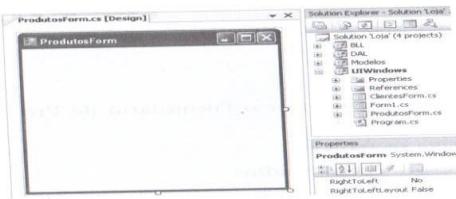
→ No Tipo de Template, selecione *Windows Form*;

→ No Nome do Item, digite *ProdutosForm.cs*



→ Clique em *Add* para adicionar o formulário.

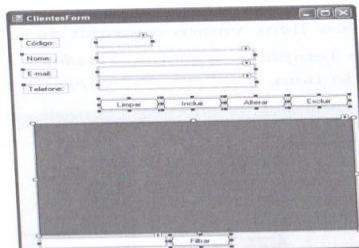
Será exibido o formulário *ProdutosForm*:
 → Altere a propriedade para *Size: 448; 512*.



6.1 Trabalhando com os Objetos do Formulário

Como as interfaces são semelhantes, vamos copiar os objetos do Formulário de Clientes para o Formulário de Produtos:

- Abra o formulário *ClientsForm* no modo de Design;
- Digite *<Ctrl>+<A>* para selecionar todos os objetos;



- Digite *<Ctrl>+<C>* para copiar os objetos para a área de transferência;
- Abra o formulário *ProdutosForm* em modo de Design;

Agora, vamos trabalhar com os objetos:

- No objeto *emaiLabel* altere os valores das seguintes propriedades:

Name: **precoLabel**

Text: **Preço:**

- No objeto *emailTextBox* altere a propriedade *Name* para *precoTextBox*.

→ No objeto *telefoneLabel* altere os valores das seguintes propriedades:

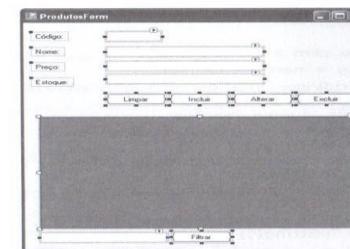
Name: **estoqueLabel**

Text: **Estoque:**

- No objeto *telefoneTextBox* altere a propriedade *Name* para *estoqueTextBox*.

- No objeto *clientesDataGridView* altere a propriedade *Name* para *produtosDataGridView*.

Neste momento, o Formulário de Produtos estará com esta aparência:



Aqui também, vamos criar o método *AtualizaGrid()* para fazer a comunicação com a camada de regras de negócios com o objetivo de preencher o objeto *DataGridView* com a lista de produtos existentes.

Dentro do método *ProdutosForm_Load()*, vamos fazer uma chamada para o método *AtualizaGrid()* e também posicionaremos o cursor no primeiro campo do formulário.

Digite o trecho de código para o code behind do formulário *ProdutosForm*:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using Loja.BLL;
using Loja.DAL;
using Loja.Modelos;

namespace Loja.UIWindows
{
    public partial class ProdutosForm : Form
    {
        public ProdutosForm()
        {
            InitializeComponent();
        }

        public void AtualizaGrid()
        {
            // Comunicação com a Camada BLL
            ProdutosBLL obj = new ProdutosBLL();
            produtosDataGridView.DataSource = obj.Listagem("");
            codigoTextBox.Text = produtosDataGridView[0, produtosDataGridView.CurrentRow.Index].Value.ToString();
            nomeTextBox.Text = produtosDataGridView[1, produtosDataGridView.CurrentRow.Index].Value.ToString();
            precoTextBox.Text = produtosDataGridView[2, produtosDataGridView.CurrentRow.Index].Value.ToString();
            estoqueTextBox.Text = produtosDataGridView[3, produtosDataGridView.CurrentRow.Index].Value.ToString();
        }

        private void ProdutosForm_Load(object sender, EventArgs e)
        {
            AtualizaGrid();
            nomeTextBox.Focus();
        }
    }
}
```

Agora, vamos codificar as funcionalidades dos botões do Formulário de Produtos:

- Dê um duplo clique no botão *Limpar* para codificarmos o evento click;
- Copie e cole o código a seguir:

```
codigoTextBox.Text = "";
nomeTextBox.Text = "";
precoTextBox.Text = "";
estoqueTextBox.Text = "";
```

- Dê um duplo clique no botão *Incluir* para codificarmos o evento click;
- Copie e cole o código a seguir:

```
try
{
    ProdutoInformation produto = new ProdutoInformation();

    produto.Nome = nomeTextBox.Text;
    produto.Preco = Convert.ToDecimal(precoTextBox.Text);
    produto.Estoque = Convert.ToInt32(estoqueTextBox.Text);

    ProdutosBLL obj = new ProdutosBLL();
    obj.Incluir(produto);
    MessageBox.Show("O produto foi incluído com sucesso!");
    codigoTextBox.Text = Convert.ToString(produto.Codigo);
}
```

- Dê um duplo clique no botão *Alterar* para codificarmos o evento click;
- Copie e cole o código a seguir:

```
if (codigoTextBox.Text == " ")
{
    MessageBox.Show("Um produto precisa ser selecionado para alteração!");
}
```

```

else
try
{
    ProdutoInformation produto = new ProdutoInformation();

    produto.Codigo = int.Parse(codigoTextBox.Text);
    produto.Nome = nomeTextBox.Text;
    produto.Preco = Convert.ToDecimal(precoTextBox.Text);
    produto.Estoque = Convert.ToInt32(estoqueTextBox.Text);

    ProdutosBLL obj = new ProdutosBLL();
    obj.Alterar(produto);
    MessageBox.Show("O produto foi atualizado com sucesso!");

}
catch (Exception ex)
{
    MessageBox.Show("Erro: " + ex.Message);
}
AtualizaGrid();

```

- Dê um duplo clique no botão *Excluir* para codificarmos o evento click;
 → Copie e cole o código a seguir:

```

if (codigoTextBox.Text.Length == 0)
{
    MessageBox.Show("Um produto deve ser selecionado antes da exclusão.");
}
else
try
{
    int codigo = Convert.ToInt32(codigoTextBox.Text);
    ProdutosBLL obj = new ProdutosBLL();
    obj.Excluir(codigo);
    MessageBox.Show("O produto foi excluído com sucesso!");
    AtualizaGrid();
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}

```

- Dê um duplo clique no botão *Filtrar* para codificarmos o evento click;
 → Copie e cole o código a seguir:

```

// Comunicação com a Camada BLL
ProdutosBLL obj = new ProdutosBLL();
produtosDataGridView.DataSource = obj.Listagem(txtFiltro.Text);

// Atualizando os objetos TextBox
try
{
    codigoTextBox.Text = produtosDataGridView[0, produtosDataGridView.CurrentRow.Index].Value.ToString();
    nomeTextBox.Text = produtosDataGridView[1, produtosDataGridView.CurrentRow.Index].Value.ToString();
    precoTextBox.Text = produtosDataGridView[2, produtosDataGridView.CurrentRow.Index].Value.ToString();
    estoqueTextBox.Text = produtosDataGridView[3, produtosDataGridView.CurrentRow.Index].Value.ToString();
}
catch {
    codigoTextBox.Text = "";
    nomeTextBox.Text = "";
    precoTextBox.Text = "";
    estoqueTextBox.Text = "";
}

```

- Clique sobre o *dataGridView*;

Na janela de propriedades:

- Clique no ícone que tem um relâmpago para exibir a lista de eventos;
 → Dê um duplo clique no evento *CellClick* para inserirmos o código para esse evento;
 → Dentro do evento *CellClick*, copie e cole o código a seguir:

```

// Atualizando os objetos TextBox
codigoTextBox.Text = produtosDataGridView[0, produtosDataGridView.CurrentRow.Index].Value.ToString();
nomeTextBox.Text = produtosDataGridView[1, produtosDataGridView.CurrentRow.Index].Value.ToString();
precoTextBox.Text = produtosDataGridView[2, produtosDataGridView.CurrentRow.Index].Value.ToString();
estoqueTextBox.Text = produtosDataGridView[3, produtosDataGridView.CurrentRow.Index].Value.ToString();

```

O code behind completo do nosso ProdutosForm.cs ficou assim:

```
(Arquivo Loja_Fontes\UIWindows\ProdutosForm.cs)
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using Loja.BLL;
using Loja.DAL;
using Loja.Modelos;

namespace Loja.UIWindows
{
    public partial class ProdutosForm : Form
    {
        public ProdutosForm()
        {
            InitializeComponent();
        }

        public void AtualizaGrid()
        {
            // Comunicação com a Camada BLL
            ProdutosBLL obj = new ProdutosBLL();
            produtosDataGridView.DataSource = obj.Listagem("");

            // Atualizando os objetos TextBox
            codigoTextBox.Text = produtosDataGridView[0, produtosDataGridView.CurrentRow.Index].Value.ToString();
            nomeTextBox.Text = produtosDataGridView[1, produtosDataGridView.CurrentRow.Index].Value.ToString();
            precoTextBox.Text = produtosDataGridView[2, produtosDataGridView.CurrentRow.Index].Value.ToString();
            estoqueTextBox.Text = produtosDataGridView[3, produtosDataGridView.CurrentRow.Index].Value.ToString();
        }

        private void ProdutosForm_Load(object sender, EventArgs e)
        {
            AtualizaGrid();
            nomeTextBox.Focus();
        }
    }
}
```

Formulário de Produtos

```
private void limparButton_Click(object sender, EventArgs e)
{
    codigoTextBox.Text = "";
    nomeTextBox.Text = "";
    precoTextBox.Text = "";
    estoqueTextBox.Text = "";
}

private void incluirButton_Click(object sender, EventArgs e)
{
    try
    {
        ProdutoInformation produto = new ProdutoInformation();

        produto.Nome = nomeTextBox.Text;
        produto.Preco = Convert.ToDecimal(precoTextBox.Text);
        produto.Estoque = Convert.ToInt32(estoqueTextBox.Text);
        ProdutosBLL obj = new ProdutosBLL();
        obj.Incluir(produto);
        MessageBox.Show("O produto foi incluído com sucesso!");
        codigoTextBox.Text = Convert.ToString(produto.Codigo);
    }
    catch (Exception ex)
    {
        MessageBox.Show("Erro: " + ex.Message);
    }
    AtualizaGrid();
}

private void alterarButton_Click(object sender, EventArgs e)
{
    if (codigoTextBox.Text == "")
    {
        MessageBox.Show("Um produto precisa ser selecionado para alteração.");
    }
    else
    {
        try
        {
            ProdutoInformation produto = new ProdutoInformation();

            produto.Codigo = int.Parse(codigoTextBox.Text);
            produto.Nome = nomeTextBox.Text;
            produto.Preco = Convert.ToDecimal(precoTextBox.Text);
        }
        catch (Exception ex)
        {
            MessageBox.Show("Erro: " + ex.Message);
        }
    }
}
```

Desenvolvimento em Camadas com C# .Net

```

        produto.Estoque = Convert.ToInt32(estoqueTextBox.Text);

        ProdutosBLL obj = new ProdutosBLL();
        obj.Alterar(produto);
        MessageBox.Show("O produto foi atualizado com sucesso!");

    }
    catch (Exception ex)
    {
        MessageBox.Show("Erro: " + ex.Message);
    }

    AtualizaGrid();
}

private void excluirButton_Click(object sender, EventArgs e)
{
    if (codigoTextBox.Text.Length == 0)
    {
        MessageBox.Show("Um produto deve ser selecionado antes da exclu-
são.");
    }
    else
    {
        try
        {
            int codigo = Convert.ToInt32(codigoTextBox.Text);
            ProdutosBLL obj = new ProdutosBLL();
            obj.Excluir(codigo);
            MessageBox.Show("O produto foi excluído com sucesso!");
            AtualizaGrid();
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }
}

private void produtosDataGridView_CellClick(object sender,
DataGridViewCellEventArgs e)
{
    // Atualizando os objetos TextBox
    codigoTextBox.Text = produtosDataGridView[0, produtosDataGridView.
CurrentRow.Index].Value.ToString();
    nomeTextBox.Text = produtosDataGridView[1, produtosDataGridView.

```

Formulário de Produtos

```

    CurrentRow.Index].Value.ToString();
    estoqueTextBox.Text = produtosDataGridView[3, produtosDataGridView.
CurrentRow.Index].Value.ToString();
}

private void btFiltro_Click(object sender, EventArgs e)
{
    // Comunicação com a Camada BLL
    ProdutosBLL obj = new ProdutosBLL();
    produtosDataGridView.DataSource = obj.Listagem(txtFiltro.Text);

    // Atualizando os objetos TextBox
    try
    {
        codigoTextBox.Text = produtosDataGridView[0, produtosDataGridView.
CurrentRow.Index].Value.ToString();
        nomeTextBox.Text = produtosDataGridView[1, produtosDataGridView.
CurrentRow.Index].Value.ToString();
        precoTextBox.Text = produtosDataGridView[2, produtosDataGridView.
CurrentRow.Index].Value.ToString();
        estoqueTextBox.Text = produtosDataGridView[3, produtosDataGridView.
View. CurrentRow.Index].Value.ToString();
    }
    catch {
        codigoTextBox.Text = "";
        nomeTextBox.Text = "";
        precoTextBox.Text = "";
        estoqueTextBox.Text = "";
    }
}

```

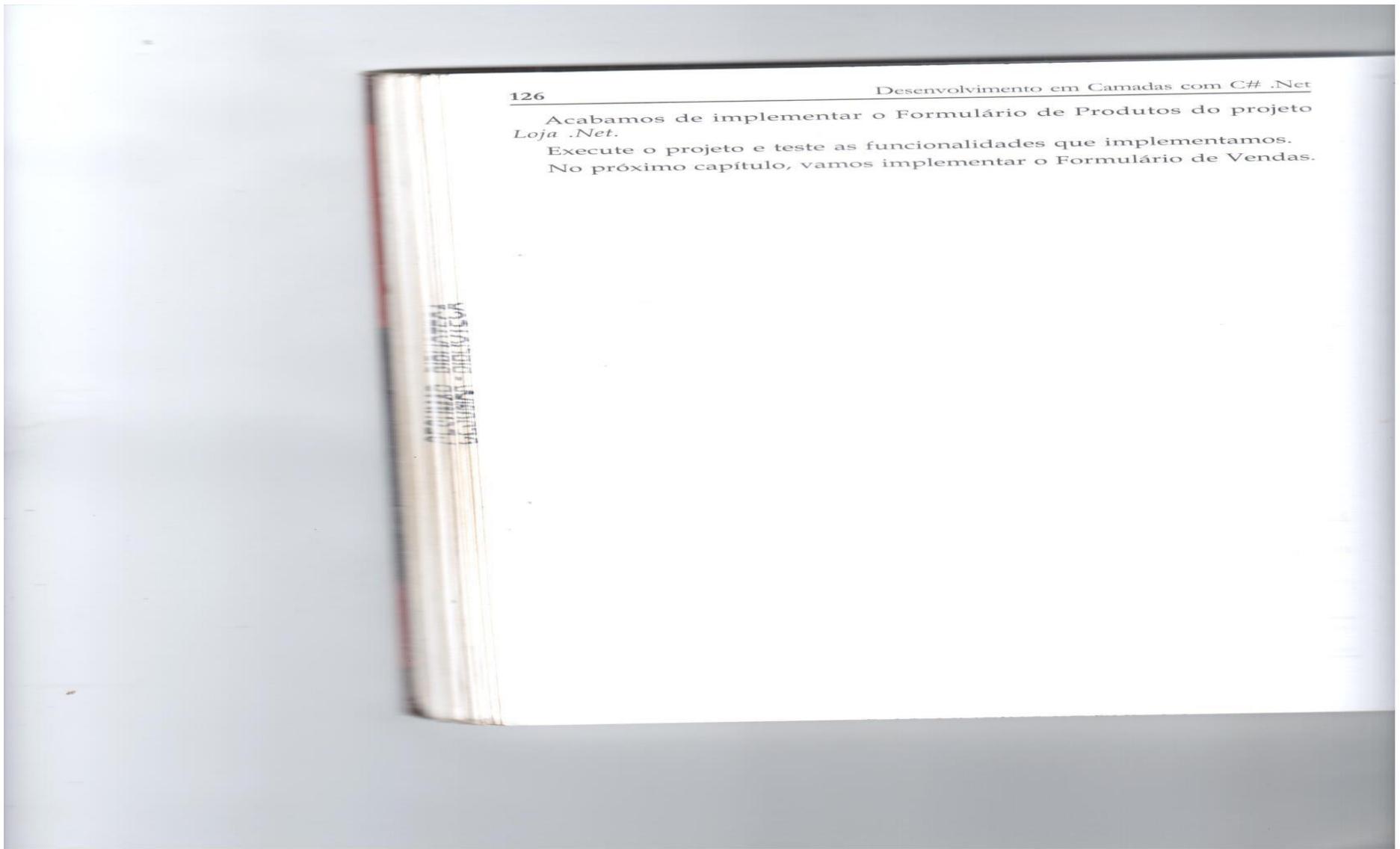
Agora só falta incluir a chamada do nosso Formulário de Produtos no menu principal do sistema:

- Abra o formulário principal *Form1* no modo Design;
- Dê um duplo clique na opção *Produtos* do menu para codificarmos a chamada para este formulário com o código a seguir:

```

ProdutosForm obj = new ProdutosForm();
obj.MdiParent = this;
obj.Show();

```



126

Desenvolvimento em Camadas com C# .Net

Acabamos de implementar o Formulário de Produtos do projeto
Loja .Net.

Execute o projeto e teste as funcionalidades que implementamos.
No próximo capítulo, vamos implementar o Formulário de Vendas.

7

Formulário de Vendas

Neste capítulo, vamos criar o Formulário de Vendas do nosso projeto:

Abra o Microsoft Visual Studio:

→ Clique em *File>Open>Project/Solution...*

Na janela Open Project:

→ Selecione o arquivo da nossa Solution *C:\Loja\Modelos\Loja.sln*;

→ Clique em *Open* para abrir a solução.

Vamos adicionar um formulário chamado *VendasForm.cs*:

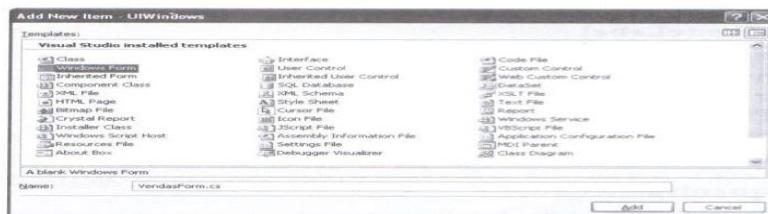
→ Clique com o botão direito no projeto *UIWindows*;

→ Escolha a opção *Add>Windows Form...*

Na janela Add New Item, vamos informar os seguintes dados:

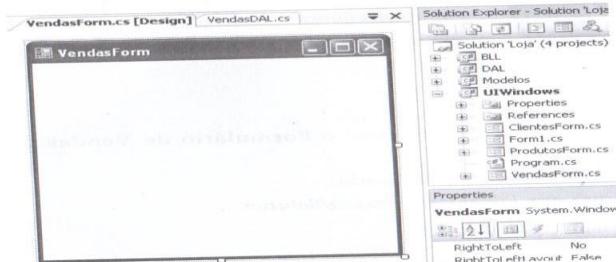
→ No Tipo de Template, selecione *Windows Form*;

→ No Nome do Item, digite *VendasForm.cs*;



→ Clique em *Add* para adicionar o formulário.

Será exibido o formulário *VendasForm*:



→ Altere a propriedade para **Size: 470; 200**.

7.1 Criando Objetos no Formulário

Vamos criar sete objetos neste formulário.

Estes objetos podem ser arrastados para o formulário a partir da opção *Common Controls* da Toolbox.

→ Arraste e solte três objetos do tipo *Label* para o nosso Formulário de Vendas e configure-os com as seguintes características:

Name: **clienteLabel**

Text: **Cliente:**

Location: **35; 37**

Name: **produtoLabel**

Text: **Produto:**

Location: **35; 64**

Name: **quantidadeLabel**

Text: **Quantidade:**

→ Arraste e solte dois objetos do tipo *ComboBox* para o nosso Formulário de Vendas e defina as seguintes propriedades:

Name: **clienteComboBox**

DisplayMember: **Nome**

DropDownStyle: **DropDownList**

DropDownWidth: **300**

Location: **106; 34**

Size: **300; 21**

ValueMember: **Codigo**

Name: **produtoComboBox**

DisplayMember: **Nome**

DropDownStyle: **DropDownList**

DropDownWidth: **300**

Location: **106; 64**

Size: **300; 21**

ValueMember: **Codigo**

→ Arraste e solte um objeto do tipo *TextBox* para o nosso Formulário de Vendas e defina as seguintes propriedades:

Name: **quantidadeTextBox**

Location: **106; 95**

Size: **66; 20**

Text: **1**

→ Arraste e solte um objeto do tipo *Button* para o nosso Formulário de Vendas e defina as seguintes propriedades:

Name: **incluirVendaButton**

Location: **106; 128**

Size: **171; 23**

Text: **Realizar a Venda**

Neste momento, nosso Formulário de Vendas estará com esta aparência:



No trecho de código a seguir, vamos incluir dados nos dois objetos *combobox* sempre que o Formulário de Vendas for carregado.

- Dê um duplo clique em uma área livre do formulário para exibir o método *VendasForm_Load()*;
- Digite o código a seguir:

```
VendasBLL obj = new VendasBLL();
clienteComboBox.DataSource = obj.ListaDeClientes;
produtoComboBox.DataSource = obj.ListaDeProdutos;
```

Observação: No início do code behind do *VendasForm.cs* devemos usar:

```
using Loja.BLL;
using Loja.DAL;
using Loja.Modelos;
```

No modo de Design do *VendasForm*:

- Dê um duplo clique no botão *Realizar a Venda*;
- Digite o código a seguir:

```
try
{
    VendaInformation venda = new VendaInformation();
    venda.Quantidade = int.Parse(quantidadeTextBox.Text);
    venda.CodigoCliente = (int)clienteComboBox.SelectedValue;
    venda.CodigoProduto = (int)produtoComboBox.SelectedValue;
    venda.Data = DateTime.Now;
    venda.Faturado = false;

    VendasBLL obj = new VendasBLL();
    obj.IncluirVenda();
}
```

```
MessageBox.Show("A venda foi realizada com sucesso!");
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
```

O code behind completo do nosso *VendasForm.cs* ficou assim:

```
(Arquivo Loja_Fontes\UIWindows\VendasForm.cs)
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using Loja.BLL;
using Loja.DAL;
using Loja.Modelos;

namespace Loja.UIWindows
{
    public partial class VendasForm : Form
    {
        public VendasForm()
        {
            InitializeComponent();
        }

        private void VendasForm_Load(object sender, EventArgs e)
        {
            VendasBLL obj = new VendasBLL();
            clienteComboBox.DataSource = obj.ListaDeClientes;
            produtoComboBox.DataSource = obj.ListaDeProdutos;
        }

        private void incluirVendaButton_Click(object sender, EventArgs e)
        {
            try
            {
                VendaInformation venda = new VendaInformation();
                venda.Quantidade = int.Parse(quantidadeTextBox.Text);
                venda.CodigoCliente = (int)clienteComboBox.SelectedValue;
                venda.CodigoProduto = (int)produtoComboBox.SelectedValue;
            }
        }
    }
}
```

```
venda.Data = DateTime.Now;
venda.Faturado = false;

VendasBLL obj = new VendasBLL();
obj.Incluir(venda);
MessageBox.Show("A venda foi realizada com sucesso!");

}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}

}
```

Agora só falta incluir a chamada do nosso Formulário de Vendas no menu principal do sistema.

- Abra o formulário principal *Form1* no modo de Design;
- Dê um duplo clique na opção *Vendas* do menu para codificarmos a chamada para este formulário com o código a seguir:

```
VendasForm obj = new VendasForm();
obj.MdiParent = this;
obj.Show();
```

Acabamos de implementar o Formulário de Vendas do nosso projeto *Loja .Net*.

Execute o projeto e teste as funcionalidades que implementamos neste artigo.

Perceba que a quantidade de produtos adquirida por um cliente na venda deverá se refletir automaticamente na quantidade de produtos em estoque. Após realizar uma venda, abra o formulário de produtos e veja que o estoque foi devidamente atualizado.

No próximo capítulo, vamos implementar o Formulário de Produtos em Falta.

8

Formulário de Produtos em Falta

Neste capítulo, vamos criar o Formulário de Produtos em Falta.

Abra o Microsoft Visual Studio:

→ Clique em *File>Open>Project/Solution...*

Na janela Open Project:

→ Selecione o arquivo da nossa Solution *C:\Loja\Modelos\Loja.sln*;

→ Clique em *Open* para abrir a solução.

Vamos adicionar um formulário chamado *ProdutosEmFaltaForm.cs*:

→ Clique com o botão direito no projeto *UIWindows*;

→ Escolha a opção *Add>Windows Form...*

Na janela Add New Item, vamos informar os seguintes dados:

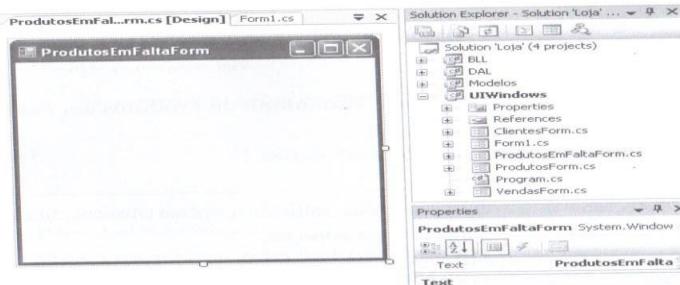
→ No Tipo de Template, selecione *Windows Form*;

→ No Nome do Projeto, digite *ProdutosEmFaltaForm*;



→ Clique em *Add* para adicionar o formulário.

Será exibido o formulário *ProdutosEmFaltaForm*:



→ Altere a propriedade para **Size: 510; 310**.

Vamos criar um *Label* e um *DataGridView*.

→ Arraste e solte um *Label* para o Formulário de Produtos em Falta e configure-o com as seguintes características:

Name: **produtosEmFaltaLabel**

Location: **9; 21**

Text: **Relatório de Produtos em Falta**

→ Arraste e solte um objeto do tipo *DataGridView* e defina as seguintes propriedades:

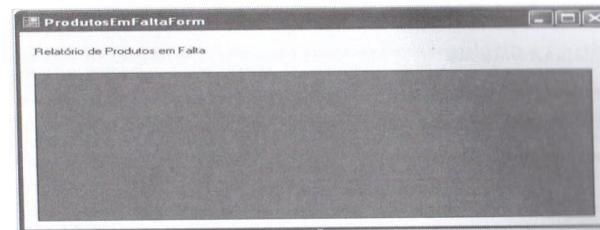
Name: **produtosEmFaltaDataGridView**

Anchor: **Top, Bottom, Left, Right**

Location: **12; 58**

Size: **474; 206**

Neste momento, nosso Formulário de Produtos em Falta estará com esta aparência:



Vamos inserir a relação de produtos em falta sempre que o formulário for carregado:

→ Dê um duplo clique em uma área livre do formulário para exibir o método *ProdutosEmFaltaForm_Load()*;

→ Copie e cole o código a seguir:

```
ProdutosBLL produto = new ProdutosBLL();
produtosEmFaltaDataGridView.DataSource = produto.ProdutosEmFalta();
```

Importante: Não se esqueça de usar no início do code behind do *ProdutosEmFaltaForm.cs*:

```
using Loja.BLL;
using Loja.DAL;
using Loja.Modelos;
```

O code behind completo do *ProdutosEmFaltaForm.cs* ficou assim:

(Arquivo *Loja.Fontes\UIWindows\ProdutosEmFaltaForm.cs*)

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

```

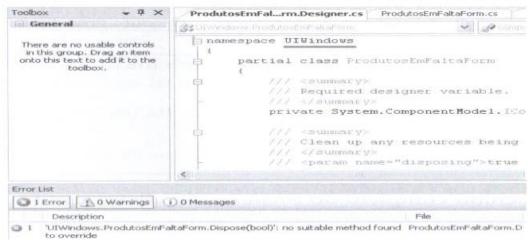
using Loja.DAL;
using Loja.Modelos;

namespace Loja.UIWindows
{
    public partial class ProdutosEmFaltaForm : Form
    {
        public ProdutosEmFaltaForm()
        {
            InitializeComponent();
        }

        private void ProdutosEmFaltaForm_Load(object sender, EventArgs e)
        {
            ProdutosBLL produto = new ProdutosBLL();
            produtosEmFaltaDataGridView.DataSource = produto.ProdutosEmFalta();
        }
    }
}

```

Compile o projeto para nos certificarmos de que tudo foi implementado corretamente.



8.1 Ocorrendo Erro

Se ocorrer o erro *no suitable method found ... to override*, como na imagem anterior:

→ Corrija o namespace do código colocando, antes do nome do projeto *UIWindows*, a palavra *Loja*.

→ Compile o projeto novamente para ter certeza de que tudo está correto.

Agora vamos fazer a chamada para este formulário a partir do menu principal do projeto *Loja.Net*:

- Abra o formulário principal *Form1* no modo de Design;
- Dê um duplo clique na opção *Produtos em Falta* do menu para codificarmos a chamada para este formulário com o código a seguir:

```

ProdutosEmFaltaForm obj = new ProdutosEmFaltaForm();
obj.MdiParent = this;
obj.Show();

```

Acabamos de implementar o Formulário de Produtos em Falta do nosso projeto *Loja .Net*.

Execute o projeto e teste as funcionalidades que implementamos neste capítulo.

Realize vendas até que um determinado produto fique com menos de dez unidades em estoque. Esse produto deverá aparecer no relatório de produtos em falta.

9

Dicas de Interface Web com Qualidade

Neste capítulo, vamos dar algumas dicas sobre a criação de uma interface Web com todas as funcionalidades necessárias.

Uma camada de interface Web pode ser implementada de diferentes maneiras. Tudo vai depender das necessidades do usuário.

Não desenvolveremos muitas Webpages (páginas Web) pois, normalmente a construção de outras páginas é bem semelhante ao que vamos aprender implementando a página para gerenciar clientes. Geralmente uma Webpage contém opções de visualização, alteração, exclusão ou inclusão de dados. Agora, vamos implementar a Webpage referente à Tabela de Clientes.

Na Webpage *clientes.aspx*, aprenderemos como consultar, incluir, excluir e alterar informações utilizando os objetos do *.Net Framework* para a Web. São essas as operações básicas que você precisa dominar para implementar novas Webpages sempre que necessário.

Nosso objetivo é transmitir técnicas de construção de Webpages como o uso de *MasterPages* e *Web Controls*. Com isso, você estará preparado para criar websites de fácil entendimento e manutenção.

Devemos nos lembrar de que o mercado de TI exige não só a velocidade no desenvolvimento de aplicações, mas também conhecimento técnico do desenvolvedor para que seja possível criar softwares profissionais de fácil manutenção.

A interface com o usuário deve ser fácil e intuitiva, procure não complicar ao implementar funcionalidades que não podem ser implementadas de forma clara e precisa.

9.1 Criando o Website UIWeb

Abra o Microsoft Visual Studio:

→ Clique em *File>Open>Project/Solution...*

Na janela Open Project:

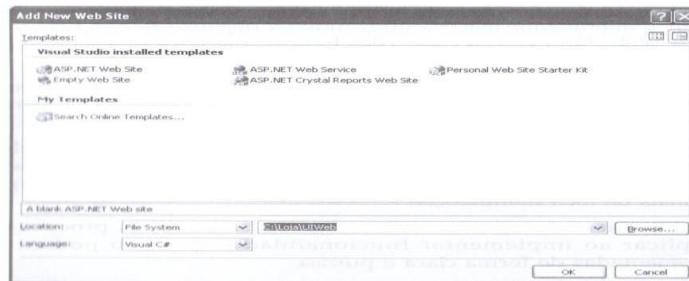
- Selecione o arquivo da nossa Solution: *C:\Loja\Modelos\Loja.sln*;
- Clique em *Open* para abrir a solução.

Vamos adicionar um projeto de interface Web:

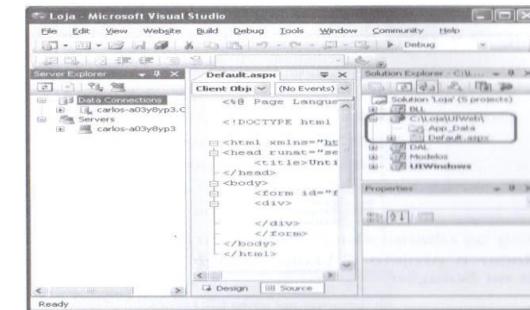
- Clique com o botão direito na Solution: *Loja*;
- Escolha *Add>New Web Site...*

Na janela Add New Web Site, vamos informar os dados listados a seguir:

- No Tipo de Template, selecione *ASP.NET Web Site*;
- Na Localização do Projeto, digite *C:\Loja\UIWeb*;
- No Tipo de Linguagem, digite *Visual C#*;
- Clique em *OK* para adicionar o website.



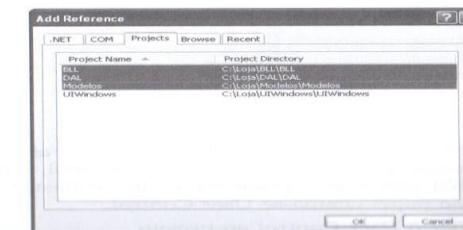
Fazendo isso, será exibido o web site no Solution Explorer como a seguir:



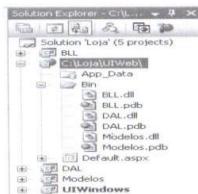
A primeira coisa a ser feita é adicionar os projetos *Modelos*, *DAL* e *BLL* como referência.

No Solution Explorer:

- Clique com o botão direito do mouse sobre o projeto e escolha *Add Reference...*
- Na aba *Projects* da janela Add Reference, mantenha a tecla *<Ctrl>* pressionada e selecione os projetos *Modelos*, *DAL* e *BLL*.
- Clique em *OK* para criar as referências.



Os projetos referenciados agora aparecem no Solution Explorer:

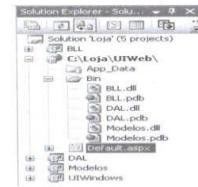


Vamos colocar o projeto C:\Loja\UIWeb como sendo o primeiro a ser executado na Solução.

Para isso:

- No Solution Explorer, clique com o botão direito do mouse sobre o projeto C:\Loja\UIWeb;
- Escolha a opção *Set as StartUp Project*.

No Solution Explorer, o nome do projeto ficará em negrito indicando que ele é o primeiro a ser executado na Solução.



Se existir na nossa aplicação Web uma área que vai se repetir em muitas páginas, como por exemplo o menu principal, é interessante usarmos um único objeto que pode ser incluído nestas páginas. O objeto que faz esse papel chama-se User Control.

Vamos adicionar o User Control ao projeto.

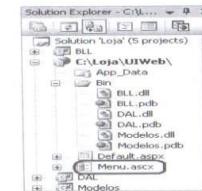
No Solution Explorer:

- Clique com o botão direito do mouse sobre o projeto UIWeb;
- Escolha a opção *Add New Item...*

Na janela Add New Item, configure:

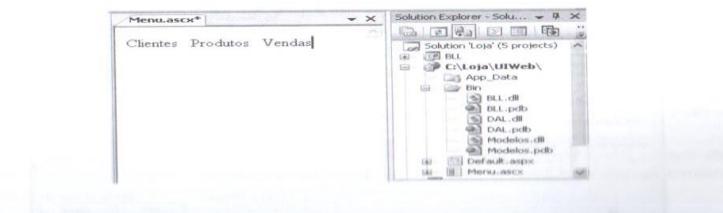
- No Tipo de Template, selecione *Web User Control*;
- No Nome do Projeto, digite *Menu.ascx*;
- Tipo de Linguagem, selecione *Visual C#*;
- Deixe selecionada a opção *Place code in separate file*;
- Clique em *Add* para adicionar o item.

A seguir, podemos ver o novo item adicionado ao projeto:



No modo de Design do User Control *Menu.ascx*:

- Digite *Clientes Produtos Vendas*, como a seguir:



→ Salve o arquivo *Menu.ascx*.

Esse arquivo será o menu principal da nossa aplicação Web e será usado no próximo item deste capítulo onde criaremos uma MasterPage.

Veremos mais adiante que ao criar um Web Form (Formulário Web), poderemos usar como modelo um item do tipo MasterPage. Usando uma MasterPage como modelo para a criação de novas páginas Web, teremos vantagens como:

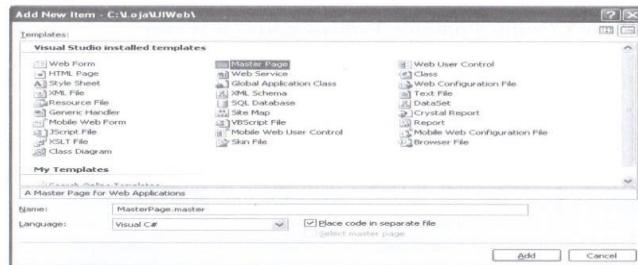
- O User Control *Menu.ascx* que é o menu principal da nossa aplicação, estará contido na MasterPage, que por sua vez, será o modelo para todas as nossas páginas Web. Desse modo, sempre que for necessária alguma alteração no menu principal, bastará alterar somente o User Control, ou seja, editar o arquivo *Menu.ascx*;
- Da mesma forma, como a MasterPage será modelo para todas as nossas páginas Web, veremos que para processar alguma alteração em todas as páginas da nossa aplicação bastará atualizar a Master Page.

Vamos adicionar ao projeto um novo item do tipo MasterPage.

No Solution Explorer:

- Clique com o botão direito do mouse sobre o projeto *UIWeb*;
- Selecione *Add New Item...*

Na janela *Add New Item*, selecione as opções como na figura a seguir:



→ No Tipo de Template, selecione *MasterPage*;

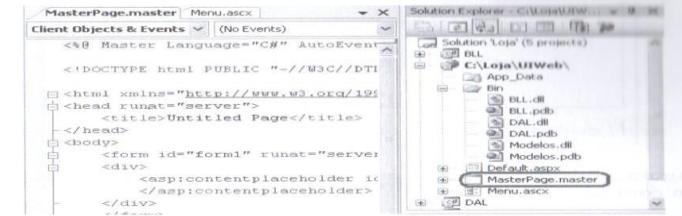
→ No Nome do Item, digite *MasterPage.master*;

→ Tipo de Linguagem, selecione *Visual C#*;

→ Deixe selecionada a opção *Place code in separate file*;

→ Clique em *Add* para adicionar o item.

Na figura a seguir, veremos que a nossa MasterPage foi criada com sucesso:



Agora, nós iremos para o modo de *Design da MasterPage* e pressaremos duas vezes a tecla *Enter* para deixar duas linhas em branco antes do ContentPlaceholder.

O ContentPlaceholder é o espaço que estará disponível nas outras páginas que terão a MasterPage como modelo. As áreas fora ContentPlaceholder, ou seja, as áreas da MasterPage, serão áreas que irão se repetir em todas as páginas do nosso website que utilizam MasterPage como modelo. Veremos isso em breve quando criarmos Formulário de Clientes.

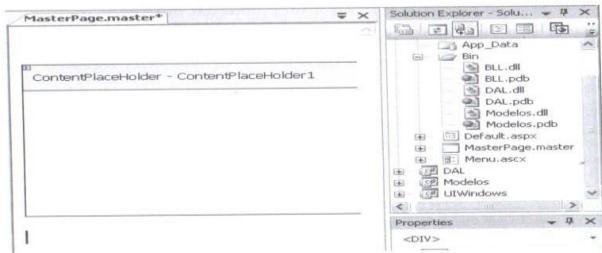
Agora, vamos definir qual será o conteúdo da nossa MasterPage.

Uma boa idéia é colocar o menu principal na MasterPage, pois das páginas que forem criadas utilizando-a já estarão com o menu principal disponível.

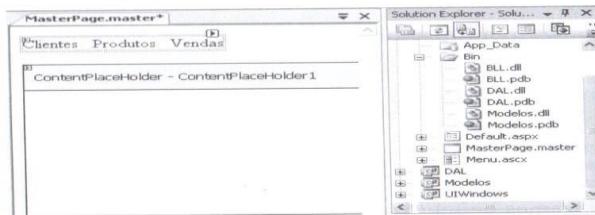
Foi por esse motivo que deixamos duas linhas em branco antes ContentPlaceholder. Este será o lugar do nosso menu principal, então vamos colocá-lo no lugar apropriado.

No Solution Explorer:

- Arraste o *Menu.ascx* e solte-o na parte superior da *MasterPage* que já está aberta em modo de Design.



Agora, podemos ver como mostrado a seguir que o menu foi adicionado com sucesso:



É interessante disponibilizar na *MasterPage* o nome do desenvolvedor ou empresa que criou a aplicação, acompanhado de e-mail ou telefone para contato.

- Digite essas informações após o *ContentPlaceHolder* como a seguir:



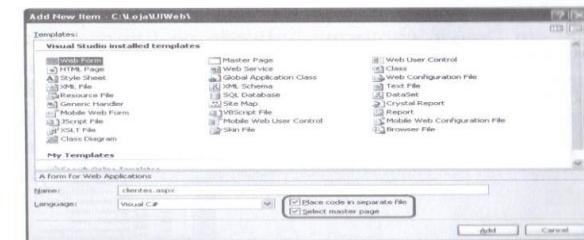
- Digite <Ctrl> + <S> para salvar a *MasterPage*.

9.2 Criando Formulário de Clientes

Vamos adicionar ao projeto um novo item do tipo *Web Form*.

No Solution Explorer:

- Clique com o botão direito do mouse no projeto *UIWeb*;
- Escolha a opção *Add New Item...*



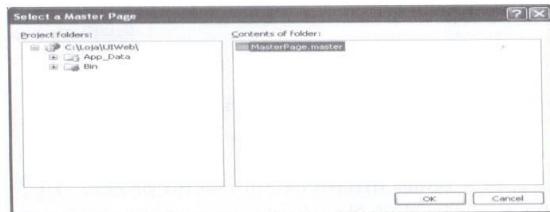
Na janela *Add New Item*, vamos definir o seguinte:

- No *Tipo de Template*, selecione *Web Form*;
- No *Nome do Item*, digite *clientes.aspx*;
- No *Tipo de Linguagem*, selecione *Visual C#*;

- Deixe selecionada a opção *Place code in separate file*;
- Deixe selecionada a opção *Select master page*;
- Clique em *Add* para prosseguir.

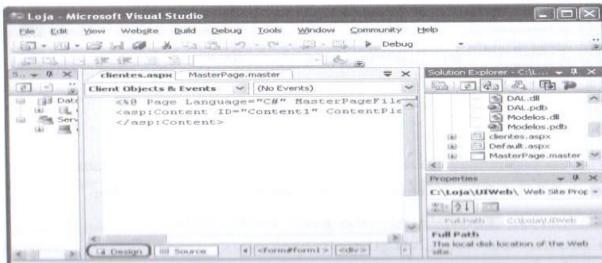
A seguir, será apresentada uma janela para a seleção da MasterPage a ser utilizada como modelo da Webpage que estamos criando.

- Selecione a *MasterPage.master*;
- Clique em *OK* para criarmos a página clientes.aspx.

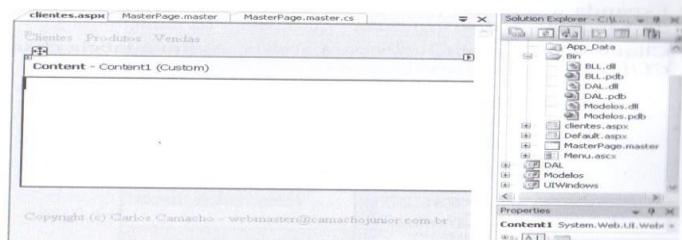


A nossa página clientes.aspx foi criada.

- Clique em *Design*, conforme indicado na figura a seguir, para exibir o modo de Design.



No modo de Design da Webpage clientes.aspx, percebemos que única área onde podemos incluir objetos e criar o que desejamos é área chamada *ContentPlaceholder*, que analisamos quando criamos MasterPage.

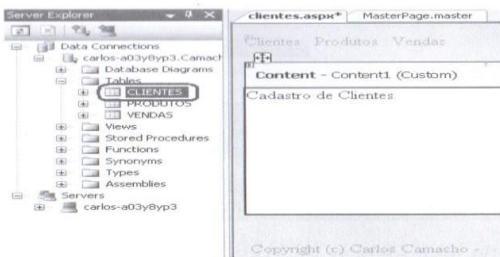


Esta página Web será o nosso Cadastro de Clientes, então vamos começar escrevendo essa informação na primeira linha da área de conteúdo.

- Clique na área *Content* da Webpage clientes.aspx;
- Escreva *Cadastro de Clientes* na primeira linha;
- Pressione *Enter* três vezes após a digitação para abrir alguma linhas para edição como a seguir:



- Digite <Ctrl> + <W>;
 - Solte a tecla <Ctrl> e <L>.
- Dessa forma, você tornará visível a janela Server Explorer.
- Expanda a conexão para exibir as pastas referentes aos objetos do banco de dados;
 - Clique sobre a pasta *Tables* para abri-la. Assim a tabela chamada *CLIENTES* ficará visível;



Agora vamos criar um objeto do tipo *gridView* no nosso Cadastro de Clientes. Com esse objeto vamos permitir que o usuário visualize, altere e exclua informações da Tabela de Clientes.

Se você nunca fez isso antes, pode estar pensando... *Nossa! Isso vai ser difícil!*

Por outro lado, se você já criou um *gridView* antes, pode estar pensando... *Ah! Isso vai ser fácil!*

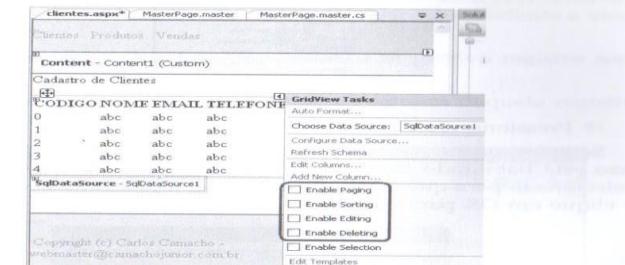
Assim, percebemos que o conceito de fácil e difícil está diretamente relacionado ao seu conhecimento.

E por que eu estou filosofando sobre tudo isso? Porque desejo que no final deste capítulo, você compreenda o que foi feito a ponto de poder repetir todo o processo com outras tabelas e em outras circunstâncias.

Mas agora chega de conversa e vamos voltar ao trabalho!

Vamos criar um *gridView* para permitir a visualização, alteração e exclusão de clientes.

- Arraste a Tabela de Clientes do Server Explorer e solte-a na Webpage *clientes.aspx*;



Um *gridView* foi criado e já podemos ver que cada campo da tabela foi colocado como uma coluna do objeto *gridView*.

As quatro primeiras opções do menu pop-up *GridView Tasks* estão destacadas na figura porque iremos selecioná-las. O simples fato de selecioná-las irá disponibilizar as funcionalidades de: paginação, ordenação, edição e exclusão ao usuário que for acessar a página.

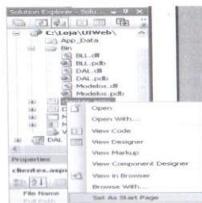
- Selecione as quatro opções destacadas na figura para habilitar essas funcionalidades.

9.3 Definindo a Webpage Inicial

Agora, vamos definir a primeira página a ser chamada pela interface Web, a Webpage *clientes.aspx*.

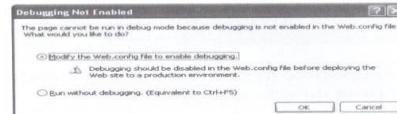
No Solution Explorer:

- Clique com o botão direito do mouse sobre a Webpage *clientes.aspx*;
- Selecione a opção *Set As Start Page* como a seguir:



→ Pressione <F5> para executar a website.

Se aparecer um alerta de que o debugging (*Debugging Not Enabled*) não está habilitado como na janela a seguir, deixe a primeira opção selecionada para que o MS Visual Studio faça as alterações necessárias e clique em *OK* para continuar.



CODIGO	NOME	EMAIL	TELEFONE
Edit Delete 1	Carlos Camacho	webmaster@camachojunior.com.br	(11) 4444-3333
Edit Delete 15	Carlos 2	webmaster@camachojunior.com.br	(11) 4444-3333
Edit Delete 16	Carlos 3	webmaster@camachojunior.com.br	(11) 4444-3333
Edit Delete 17	Carlos 4	webmaster@camachojunior.com.br	(11) 4444-3333
Edit Delete 18	Carlos 5	webmaster@camachojunior.com.br	(11) 4444-3333
Edit Delete 19	Carlos 6	webmaster@camachojunior.com.br	(11) 4444-3333
Edit Delete 20	Carlos 7	webmaster@camachojunior.com.br	(11) 4444-3333
Edit Delete 21	Carlos 8	webmaster@camachojunior.com.br	(11) 4444-3333
Edit Delete 22	Carlos 9	webmaster@camachojunior.com.br	(11) 4444-3333
Edit Delete 23	Carlos 10	webmaster@camachojunior.com.br	(11) 4444-3333

Dicas de Interface Web com Qualidade

Ok, eis a nossa Webpage em pleno funcionamento. Navegue a vontade para testar o que foi implementado.

Se você clicar no título das colunas, executará a função de ordenação das informações.

Se clicar no título de uma mesma coluna mais de uma vez, ordena as informações na ordem inversa (ascendente para descendente e vice-versa).

Clicando na opção *Delete* de uma linha, o respectivo registro se exclui da Tabela de Clientes.

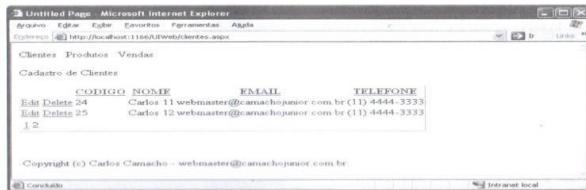
Clicando na opção *Edit* de uma linha, os campos daquele registro serão abertos para edição como a seguir:

CODIGO	NOME	EMAIL	TELEFONE
1	Carlos Camacho	webmaster@camachojunior.com.br	(11) 4444-3333
15	Registro alterado	webmaster@camachojunior.com.br	(11) 4444-3333
16	Carlos 4	webmaster@camachojunior.com.br	(11) 4444-3333
17	Carlos 5	webmaster@camachojunior.com.br	(11) 4444-3333
18	Carlos 6	webmaster@camachojunior.com.br	(11) 4444-3333
19	Carlos 7	webmaster@camachojunior.com.br	(11) 4444-3333
20	Carlos 8	webmaster@camachojunior.com.br	(11) 4444-3333
21	Carlos 9	webmaster@camachojunior.com.br	(11) 4444-3333
22	Carlos 10	webmaster@camachojunior.com.br	(11) 4444-3333
23	Carlos 10	webmaster@camachojunior.com.br	(11) 4444-3333

Na edição de um registro, você pode clicar em *Update* para salvar alterações realizadas ou em *Cancel* para descartá-las.

Os números que aparecem no rodapé do gridView correspondem ao número de páginas. O número que não está sublinhado é a página atual, e os números sublinhados correspondem as páginas para onde você pode navegar.

A seguir, um exemplo da navegação para a página 2:



Vamos encerrar a execução da aplicação e voltar ao MS Visual Studio para dar continuidade ao nosso trabalho. Para fazer isso vamos fechar a janela do browser:

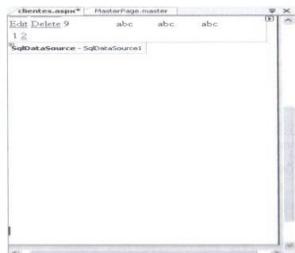
→ Feche o browser para encerrar a execução da aplicação.

Neste instante, talvez você esteja pensando... *Cadê a inclusão?*

Muito bem observado, vamos providenciar agora mesmo.

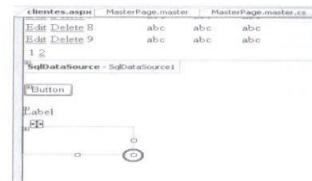
9.4 Implementando a Inclusão no Formulário de Clientes

Na página *clientes.aspx*, pressione algumas vezes a tecla *Enter* para abrir espaço na nossa área de conteúdo como a seguir:



Primeiramente, vamos incluir três objetos: um botão, um label e um panel.

→ A partir da Toolbox, arraste estes três objetos para a área de conteúdo da Webpage *clientes.aspx* como a seguir:



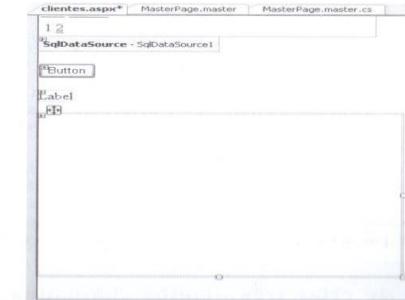
→ Clique na linha do Panel para que apareçam esses pontos. Eles nos permitem alterar o tamanho do objeto;

→ Use o ponto do canto inferior direito para aumentar o tamanho do Panel;

→ Clique sobre o ponto mantendo o botão esquerdo do mouse pressionado;

→ Arraste o ponto para aumentar o tamanho do Panel e, quando achar o tamanho adequado, solte o botão esquerdo do mouse;

→ O tamanho do Panel deverá ficar semelhante ao da figura a seguir



Agora, vamos definir as propriedades desses novos objetos:

- Clique sobre o *botão* e defina as seguintes propriedades:

ID: **btExibePanel**

Text: **Incluir**

- Clique sobre o *Label* e defina as seguintes propriedades:

ID: **lblMensagem**

Text: *(deixar em branco)*

- Clique sobre o *Panel* e defina as seguintes propriedades:

ID: **PanelClientes**

A aparência da nossa Webpage *clientes.aspx* está assim:



Para terminar o layout, ou seja, a parte gráfica da nossa Webpage, vamos incluir alguns objetos dentro do *PanelClientes*. Serão três objetos do tipo *Textbox* e dois do tipo *Button*. Deixaremos a programação para quando o layout estiver finalizado.

A possibilidade de separar o desenvolvimento do layout da programação foi um grande avanço em termos de desenvolvimento de aplicações. Assim, é possível que em um projeto exista um time de webdesigners trabalhando no desenvolvimento do layout e, ao mesmo tempo, um time de programadores trabalhando na codificação.

Por exemplo:

Nós acabamos de criar três objetos chamados *btExibePanel*, *lblMensagem* e *PanelClientes*. Agora, nós vamos criar três objetos do tipo

Textbox chamados *nomeTextBox*, *emailTextBox* e *telefoneTextBox*. Criaremos também dois objetos do tipo *Button* chamados *btCancelar* e *btIncluir*.

De posse dessas informações, nós programadores já sabemos que quando o usuário clicar no botão *btIncluir*, nossa missão é atribuir conteúdo dos objetos *textbox* na estrutura *ClienteInformation*. Lembre-se do nosso modelo em camadas?

Depois disso, instanciaremos um objeto de Regras de Negócio para validar essas informações. Se o usuário tentar incluir um cliente sem informar o seu nome, nós podemos exibir essa exceção no *label* *lblMessage*.

Bem, mas como estamos sozinhos nesse projeto e não existe nenhum time de webdesigners à disposição, vamos voltar ao trabalho!

Vamos criar o layout para permitir a inclusão de clientes:

- Clique no *PanelClientes* e digite **INCLUSÃO DE CLIENTE**;
- Pressione duas vezes a tecla *Enter* para pular duas linhas;
- Digite: *Nome*:
- Pressione mais duas vezes a tecla *Enter* ;
- Digite: *E-mail*:
- Pressione duas vezes a tecla *Enter* e digite *Telefone*:

Nossa Webpage estará assim:



- A partir da ToolBox, arraste três objetos do tipo *Textbox* e solte-os na frente dos campos que acabamos de digitar de forma que a Webpage fique assim:

The screenshot shows the 'clientes.aspx' page in design mode. It features a header with 'Inclusão de Cliente' and a footer with copyright information. Three new textboxes have been added to the form, positioned above the original ones. The original textboxes are labeled 'Nome', 'E-mail', and 'Telefone'. The new textboxes are also labeled 'Nome', 'E-mail', and 'Telefone', indicating they are copies of the originals.

- Clique sobre os objetos *TextBox* e defina a propriedade ID deles como: *nomeTextBox*, *emailTextBox* e *telefoneTextBox* nessa ordem;
→ A partir da ToolBox, arraste dois objetos do tipo *Button* para a Webpage como a seguir:

The screenshot shows the 'clientes.aspx' page in design mode. It features a header with 'Inclusão de Cliente' and a footer with copyright information. Two new buttons have been added to the form, positioned below the original ones. The original buttons are labeled 'Incluir' and 'btIMensagem'. The new buttons are also labeled 'Incluir' and 'btIMensagem', indicating they are copies of the originals.

Defina as seguintes propriedades para os botões:

ID: **btCancelar**

Text: **Cancelar**

ID: **btIncluir**

Text: **Incluir**

Com isso finalizamos a parte gráfica da nossa Webpage:

The screenshot shows the 'clientes.aspx' page in design mode. It features a header with 'Inclusão de Cliente' and a footer with copyright information. All controls from both screenshots are present: three textboxes ('Nome', 'E-mail', 'Telefone') and four buttons ('Incluir', 'btIMensagem', 'btCancelar', 'btIncluir'). The layout is identical to the second screenshot, with the new controls placed above the original ones.

Agora, vamos para a parte que eu mais gosto: a programação!

No modo de Design:

- Clique sobre o *PanelClientes*;
→ Defina a propriedade *Visible* como *False*.

Se o usuário não quiser inserir dados, não faz sentido exibirmos esses campos, não é?

Se o usuário quiser inserir um cliente, ele vai clicar no botão *btExibePanel*. E quando ele fizer isso, afim vamo exibir os campos necessários.

- Dê um duplo clique no botão *btExibePanel* e digite o seguinte código:

```
PanelClientes.Visible = true;
btExibePanel.Visible = false;
```

Ainda precisamos programar os botões Cancelar e Incluir do painel. Para isso:

- Dê um duplo clique no botão *btCancelar*;
- Digite o seguinte trecho de código:

```
//Limpando os campos do painel
nomeTextBox.Text = "";
emailTextBox.Text = "";
telefoneTextBox.Text = "";
//Esconde painel
PanelClientes.Visible = false;
//Exibe botao Incluir
btExibePanel.Visible = true;
```

→ Antes de codificarmos o botão *btIncluir*, digite essas linhas no início do code behind da nossa Webpage *clientes.aspx*:

```
using Loja.Modelos;
using Loja.DAL;
using Loja.BLL;
```

- Dê um duplo clique no botão *btIncluir* e digite o seguinte trecho de código:

```
try {
    ClienteInformation cliente = new ClienteInformation();
    cliente.Nome = nomeTextBox.Text;
    cliente.Email = emailTextBox.Text;
    cliente.Telefone = telefoneTextBox.Text;
    ClientesBLL obj = new ClientesBLL();
    obj.Incluir(cliente);
    lblMensagem.Text = "O cliente foi incluído com sucesso!";
    PanelClientes.Visible = false;
    btExibePanel.Visible = true;
    //Atualiza o gridView
    GridView1.DataBind();
}
catch (Exception ex)
{
    lblMensagem.Text = "Erro: " + ex.Message;
}
```

Bom trabalho!

Agora tecle em *<F5>* e teste todas as funcionalidades que implementamos.

Nesse capítulo aprendemos que é possível utilizar os conceitos de desenvolvimento em camadas mesmo em uma interface Web.

É claro que você vai querer melhorar tudo o que fizemos, aperfeiçoar o layout e criar novas Webpages. Por esse motivo vou deixar algumas dicas:

- Utilize sempre o MasterPage como modelos para novas páginas de um mesmo projeto. Isso vai ajudar muito nas futuras manutenções;
- Utilize o Web User Control para criar o seu menu principal. Nele você pode criar objetos do tipo *Linkbutton* para linkar opções de menu com as suas Webpages. Como o Web User Control está dentro da MasterPage, ele ficará visível para o usuário o tempo todo;
- converse calmamente com o seu usuário e faça todas as reuniões necessárias até você ter a certeza de que compreendeu as necessidades dele. Tente se colocar no lugar do usuário e tente ter a visão que ele tem;
- Não há vantagem alguma em utilizar um arsenal de conhecimento técnico se o usuário não ficar satisfeito com o produto final. O que eu mais vejo por aí é o desenvolvimento de projetos extraordinários em termos de tecnologia, mas que muitas vezes ficam inacabados ou acabam sendo pouco utilizados em relação aos propósitos para os quais foram desenvolvidos.

10

Tópicos de Segurança de Informação

Neste capítulo, vamos abordar assuntos relacionados com a segurança da informação.

10.1 Dicas para se Prevenir Contra os Hackers

Certamente você já ouviu falar neles: os hackers.

Mas o que é um hacker?

Hackers e crackers são indivíduos da sociedade em que vivem. Uma sociedade moderna onde a velocidade das inovações tecnológicas surpreende até os profissionais empreendedores. Esses indivíduos possuem conhecimentos avançados na área da Tecnologia da Informação (TI), mas há uma grande diferença entre eles.

Normalmente as pessoas relacionam um hacker a um especialista em TI que invade sistemas, rouba informações ou edita páginas web para se divertir ou protestar. Na verdade, os especialistas em TI que invadem sistemas e tem atitudes ruins são denominados crackers. crackers destroem, e quando constroem programas, esses programas são apenas para uso pessoal.

Já os hackers constroem coisas para o bem comum.

Depois dessa definição, o nosso título não se chamará *Dicas para se Prevenir Contra os Hackers*, mas sim, *Dicas para se Prevenir Contra os Crackers*.

Então vamos lá...

10.2 Dicas para se Prevenir Contra os Crackers

Neste livro, nós construímos o Sistema *Loja*. Planejamos e criamos a infra-estrutura de banco de dados, codificamos as camadas e estamos prontos para entregar o software *Loja* para o nosso cliente.

Mas, as perguntas que estão no ar são:

- Nosso software está realmente seguro?
- Ele está pronto para ser entregue ao cliente?
- Por favor, não fique bravo(a) com a resposta, mas a resposta é:
- Não, ele não está pronto.
- Mas, por que não?

A resposta está justamente na introdução deste capítulo. Nosso software não está pronto porque existem os crackers, e a existência deles exige que nós, desenvolvedores de sistemas, sejamos também conhecedores de tópicos relacionados à segurança da informação. Só assim, poderemos garantir que o sistema por nós desenvolvido não seja invadido.

Você pode estar se perguntando agora...

- Mas será que conseguiremos desenvolver uma aplicação 100% segura?

Tudo bem, vou concordar com você: mesmo com a implementação de mecanismos de segurança, nossa aplicação não estará 100% segura. Mas, certamente nós podemos dificultar bastante a ação de um cracker!

- E o que nós vamos fazer para aumentar a segurança da nossa aplicação?

A primeira coisa a fazer é um levantamento dos pontos frágeis da aplicação. Existem consultorias especializadas em fazer um estudo da empresa e sugerir melhorias em processos (computacionais ou não) para que se tenha um bom nível de segurança da informação. O resultado final desses estudos são relatórios que apontam desde pontos vulneráveis em sistemas operacionais (configurações a serem realizadas, patches que precisam ser instalados) até sugestões para alteração de regras de firewall.

- No nosso projeto, o ponto frágil que identificamos é o fato da nossa string de conexão estar em "texto aberto". O que significa isso?

Isto significa que apesar de nossa aplicação estar rodando numa máquina isolada chamada Servidor de Aplicações e que exista outra máquina dedicada ao Servidor de Banco de Dados, um cracker poderia chegar até o Servidor de Aplicações e descobrir a localização e dados de acesso ao servidor de Banco de Dados através de informações existentes na nossa camada DAL.

- Mas a nossa Camada DAL não é uma dll?

Concordo. O problema é que mesmo sem o código-fonte e somente de posse da dll, um cracker é capaz de obter a nossa string de conexão. Lembre-se que "o cara é bom", o problema é que ele não tem boas intenções...

Se ele conseguir essas informações, será possível conectar-se ao servidor de banco de dados. Uma vez feito isso, o cracker pode alterar ou excluir informações. Se isso ocorrer, com toda a certeza, a primeira coisa que o presidente da empresa que comprou o nosso software vai perguntar ao gerente de TI é:

- Quem desenvolveu esse software?

10.3 Implementando Segurança com Criptografia

Na implementação desse algoritmo de criptografia, aprenderemos uma característica muito interessante do .Net Framework: a compatibilidade entre diversas linguagens de programação.

Como o nosso algoritmo de criptografia é pequeno, vamos implementá-lo usando a linguagem VB.Net.

Veremos que no nosso projeto *UIWindows*, já criado com a linguagem C# .Net, podemos fazer referência para a nossa dll criada em VB.Net de forma transparente, ou seja, vamos instanciar objetos e chamar seus métodos normalmente.

Essa compatibilidade é possível graças ao CLR (Common Language Runtime), mas não vamos nos estender neste assunto pois, como já estamos acostumados, o nosso negócio é pôr a mão na massa.

Vejamos uma tabela com os três projetos que compõem a arquitetura da nossa solução de segurança:

Projeto Security	Projeto Secure App	Projeto WindowsAppCSharp
Escrito em VB.Net, neste projeto será criada a classe DTICrypto.	Escrito em VB.Net, neste projeto será criada a classe DTICrypto.	Será criada uma aplicação em C# .Net do tipo Winforms.
Métodos: Cifrar() Decifrar() TiraCaracteresNulos()	Métodos: Decifrar() TiraCaracteresNulos()	Essa aplicação terá como referência o projeto Security.dll
Saída: Security.dll	Saída: SecureApp.dll	Saída: WindowsAppCSharp.exe

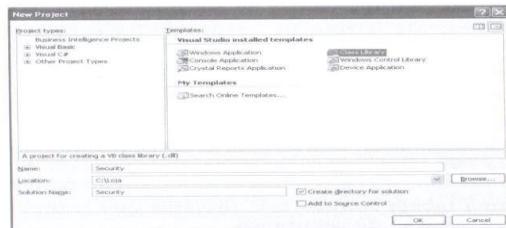
10.3.1 Criando o Projeto Security

Abra o MS Visual Studio:

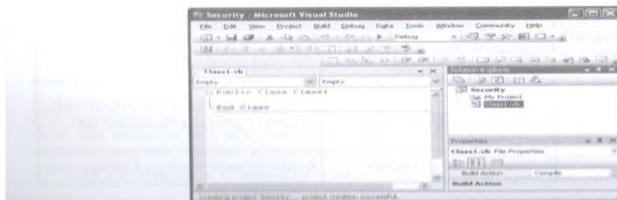
→ Clique em *File>New>Project...*

Na janela New Project, selecione:

- No Tipo de Projeto, selecione *Visual Basic*;
- No Tipo de Template, selecione *Class Library*;
- No Nome do Projeto, digite *Security*;
- Na Localização do Projeto, digite *C:\Loja*;
- No Solution Name, digite *Security*;
- Deixe selecionada a opção *Create directory for solution*;
- Clique em *Ok* para adicionar o projeto.



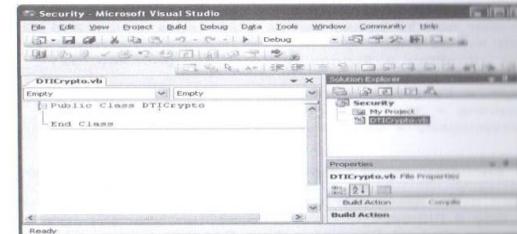
Será criado o projeto *Security*, como veremos a seguir:



O MS Visual Studio também criou automaticamente a pasta *C:\Loja\Security* para o nosso projeto.

→ Renomear o arquivo *Class1.vb* para *DTICrypto.vb*:

Nota: Com o arquivo *Class1.vb* selecionado no Solution Explorer, você pode usar <F2> ou o botão direito do mouse para renomeá-lo.



Digite o seguinte trecho de código para o arquivo *DTICrypto.vb*:

```
(Arquivo Loja_Fontes\Security\DTICrypto.vb)
Imports System.Security.Cryptography
Imports System.IO
Imports System.Text

Public Class DTICrypto

    Public Function Cifrar(ByVal vstrTextToBeEncrypted As String, ByVal
vstrEncryptionKey As String) As String

        Dim bytValue() As Byte
        Dim bytKey() As Byte
        Dim bytEncoded() As Byte
        Dim bytIV() As Byte = {122, 10, 15, 77, 131, 71, 21, 59, 255, 81, 5, 7, 14,
209, 24, 111}
        Dim intLength As Integer
        Dim intRemaining As Integer
        Dim objMemoryStream As New MemoryStream
        Dim objCryptoStream As CryptoStream
        Dim objRijndaelManaged As RijndaelManaged
```

```

' Retira caracteres nulos da palavra a ser cifrada
vstrTextToBeEncrypted = TiraCaracteresNulos(vstrTextToBeEncrypted)

' Cada valor deve existir na tabela ASCII
bytValue = Encoding.ASCII.GetBytes(vstrTextToBeEncrypted.ToCharArray)

intLength = Len(vstrEncryptionKey)

' A chave gerada será de 256 bits long (32 bytes)
' Se for maior que 32 bytes, então vamos truncar;
' Se for menor que 32 bytes, vamos alocar para atingir 256 bits.
If intLength >= 32 Then
    vstrEncryptionKey = Strings.Left(vstrEncryptionKey, 32)
Else
    intLength = Len(vstrEncryptionKey)
    intRemaining = 32 - intLength
    vstrEncryptionKey = vstrEncryptionKey & Strings.StrDup(intRemaining, "X")
End If

bytKey = Encoding.ASCII.GetBytes(vstrEncryptionKey.ToCharArray)

' Para quem quiser pesquisar sobre o algoritmo que estamos usando:
' O nome dele é Rijndael
' Foi criado por Vincent Rijment e Joan Daemen.

objRijndaelManaged = New RijndaelManaged

' Cria o valor a ser cifrado e escreve a conversão em bytes
Try

    objCryptoStream = New CryptoStream(objMemoryStream, objRijndaelManaged.CreateEncryptor(bytKey, bytIV), CryptoStreamMode.Write)
    objCryptoStream.Write(bytValue, 0, bytValue.Length)

    objCryptoStream.FlushFinalBlock()

    bytEncoded = objMemoryStream.ToArray
    objMemoryStream.Close()
    objCryptoStream.Close()
    Catch

End Try

' Retorna o valor cifrado
' (realizando conversão de byte para base64)
Return Convert.ToBase64String(bytEncoded)

```

```

Public Function Decifrar(ByVal vstrStringToBeDecrypted As String, ByVal vstrDecryptionKey As String) As String

    Dim bytDataToBeDecrypted() As Byte
    Dim bytTemp() As Byte
    Dim bytIV() As Byte = {122, 10, 15, 77, 131, 71, 21, 59, 255, 81, 5, 7, 14,
209, 24, 111}
    Dim objRijndaelManaged As New RijndaelManaged
    Dim objMemoryStream As MemoryStream
    Dim objCryptoStream As CryptoStream
    Dim bytDecryptionKey() As Byte
    Dim intLength As Integer
    Dim intRemaining As Integer
    Dim intCtr As Integer
    Dim strReturnString As String = String.Empty
    Dim achrCharacterArray() As Char
    Dim intIndex As Integer

    ' Converte de base64 cifrada para array de bytes
    bytDataToBeDecrypted = Convert.FromBase64String(vstrStringToBeDecrypted)

    ' A chave gerada será de 256 bits long (32 bytes)
    ' Se for maior que 32 bytes, então vamos truncar;
    ' Se for menor que 32 bytes, vamos alocar para atingir 256 bits.
    intLength = Len(vstrDecryptionKey)

    If intLength >= 32 Then
        vstrDecryptionKey = Strings.Left(vstrDecryptionKey, 32)
    Else
        intLength = Len(vstrDecryptionKey)
        intRemaining = 32 - intLength
        vstrDecryptionKey = vstrDecryptionKey & Strings.StrDup(intRemaining, "X")
    End If

    bytDecryptionKey = Encoding.ASCII.GetBytes(vstrDecryptionKey.ToCharArray)

    ReDim bytTemp(bytDataToBeDecrypted.Length)
    objMemoryStream = New MemoryStream(bytDataToBeDecrypted)

    ' Escreve o valor descriptografado após a conversão
    Try
        objCryptoStream = New CryptoStream(objMemoryStream, objRijndaelManaged.CreateDecryptor(bytDecryptionKey, bytIV),
CryptoStreamMode.Read)
    
```

```

    objCryptoStream.Read(bytTemp, 0, bytTemp.Length)
    objCryptoStream.FlushFinalBlock()
    objMemoryStream.Close()
    objCryptoStream.Close()

    Catch
    End Try

    ' Retorna o valor descriptografado
    Return TiraCaracteresNulos(Encoding.ASCII.GetString(bytTemp))

End Function

Private Function TiraCaracteresNulos(ByVal vstrStringWithNulls As String) As String

    Dim intPosition As Integer
    Dim strStringWithOutNulls As String

    intPosition = 1
    strStringWithOutNulls = vstrStringWithNulls

    Do While intPosition > 0
        intPosition = InStr(intPosition, vstrStringWithNulls, vbNullChar)

        If intPosition > 0 Then
            strStringWithOutNulls = Left$(strStringWithOutNulls, intPosition - 1) &
                Right$(strStringWithOutNulls, Len(strStringWithOutNulls) - intPosition)
        End If

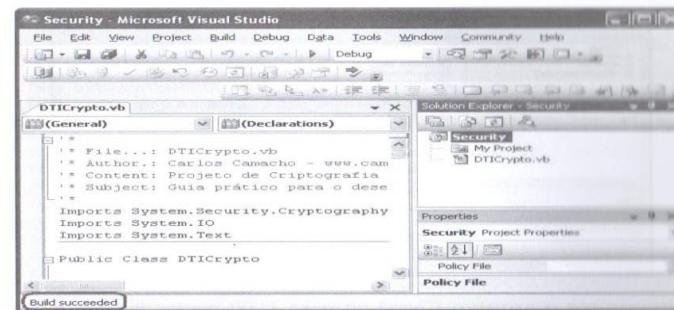
        If intPosition > strStringWithOutNulls.Length Then
            Exit Do
        End If
    Loop

    Return strStringWithOutNulls
End Function

End Class

```

→ Dê um *Build* na Solução para verificarmos se está tudo certo:



A mensagem em destaque na barra de status indica que o projeto foi compilado com sucesso. Isso também significa que o arquivo *Security.dll* foi criado em C:\Loja\Security\Security\bin\Debug.

Vamos fechar o projeto.

Para isso:

→ Clique em *File>Close Project*.

10.3.2 Criando o Projeto SecureApp

No MS Visual Studio:

→ Clique em *File>New>Project...*

Na janela New Project, selecione:

→ No Tipo de Projeto, selecione *Visual Basic*;

→ No Tipo de Template, selecione *Class Library*;

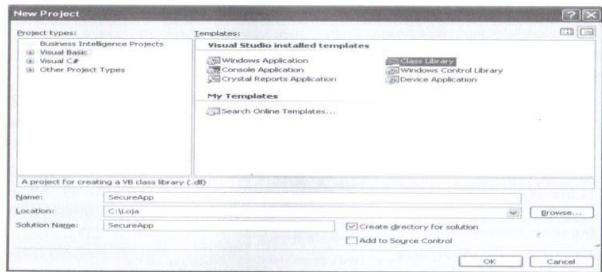
→ No Nome do Projeto, digite *SecureApp*;

→ Na Localização do Projeto, digite C:\Loja;

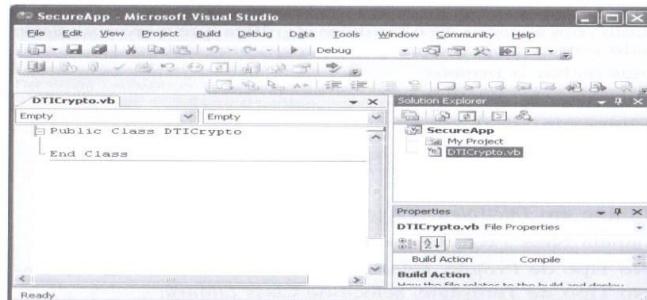
→ No Solution Name, digite *SecureApp*;

→ Deixe selecionada a opção *Create directory for solution*;

→ Clique em *Ok* para criar o projeto;



→ Renomeie o arquivo *Class1.vb* para *DTICrypto.vb*;



Digite o seguinte trecho de código para o arquivo *DTICrypto.vb*.

```
(Arquivo Loja_Fontes\SecureApp\DTICrypto.vb)
Imports System.Security.Cryptography
Imports System.IO
Imports System.Text
```

Tópicos de Segurança da Informação

Public Class DTICrypto

```
    Public Function Decifrar(ByVal vstrStringToBeDecrypted As String, ByVal vstrDecryptionKey As String) As String
        Dim bytDataToBeDecrypted() As Byte
        Dim bytTemp() As Byte
        Dim bytIV() As Byte = {122, 10, 15, 77, 131, 71, 21, 59, 255, 81, 8, 7, 209, 24, 111}
        Dim objRijndaelManaged As New RijndaelManaged
        Dim objMemoryStream As MemoryStream
        Dim objCryptoStream As CryptoStream
        Dim bytDecryptionKey() As Byte
        Dim intLength As Integer
        Dim intRemaining As Integer
        Dim intCtr As Integer
        Dim strReturnString As String = String.Empty
        Dim achrCharacterArray() As Char
        Dim intIndex As Integer

        ' Converte de base64 cifrada para array de bytes
        bytDataToBeDecrypted = Convert.FromBase64String(vstrStringToBeDecrypted)

        ' A chave gerada será de 256 bits long (32 bytes)
        ' Se for maior que 32 bytes, então vamos truncar;
        ' Se for menor que 32 bytes, vamos alocar para atingir 256 bits.
        intLength = Len(vstrDecryptionKey)

        If intLength >= 32 Then
            vstrDecryptionKey = Strings.Left(vstrDecryptionKey, 32)
        Else
            intLength = Len(vstrDecryptionKey)
            intRemaining = 32 - intLength
            vstrDecryptionKey = vstrDecryptionKey & Strings.StrDup(intRemaining, Chr(0))
        End If

        bytDecryptionKey = Encoding.ASCII.GetBytes(vstrDecryptionKey)

        ReDim bytTemp(bytDataToBeDecrypted.Length)

        objMemoryStream = New MemoryStream(bytDataToBeDecrypted)

        ' Escreve o valor descriptografado após a conversão
        Try
            objCryptoStream = New CryptoStream(objMemoryStream,
```

```

objRijndaelManaged.CreateDecryptor(bytDecryptionKey, bytIV), _
CryptoStreamMode.Read)

objCryptoStream.Read(bytTemp, 0, bytTemp.Length)

objCryptoStream.FlushFinalBlock()
objMemoryStream.Close()
objCryptoStream.Close()

Catch
End Try

' Retorna o valor descriptografado
Return TiraCaracteresNulos(Encoding.ASCII.GetString(bytTemp))

End Function

Private Function TiraCaracteresNulos(ByVal vstrStringWithNulls As String) As
String

    Dim intPosition As Integer
    Dim strStringWithOutNulls As String

    intPosition = 1
    strStringWithOutNulls = vstrStringWithNulls

    Do While intPosition > 0
        intPosition = InStr(intPosition, vstrStringWithNulls, vbNullChar)

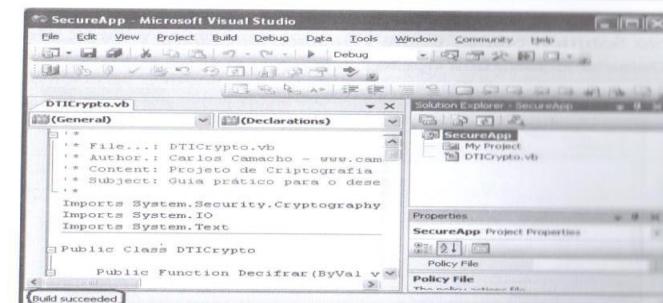
        If intPosition > 0 Then
            strStringWithOutNulls = Left$(strStringWithOutNulls, intPosition - 1) &
            Right$(strStringWithOutNulls, Len(strStringWithOutNulls) - intPosition)
        End If

        If intPosition > strStringWithOutNulls.Length Then
            Exit Do
        End If
    Loop

    Return strStringWithOutNulls
End Function
End Class

```

→ Dê um *Build* na Solução para podermos verificar se está tudo correto:



A mensagem na barra de status indica que o projeto foi compilado com sucesso. Isso também significa que o arquivo *SecureApp.dll* foi criado em *C:\Loja\SecureApp\SecureApp\bin\Debug*.

A única diferença entre os projetos *Security* e *SecureApp* é que o último não contém o método *Cifrar()*. Mais adiante, utilizaremos a *SecureApp* na camada de interface com o usuário, momento em que o método *Cifrar()* não precisa e nem deve existir.

A dll *Security* será usada no próximo projeto chamado *WindowsAppCSharp*, no momento em que criaremos a nossa string com conexão criptografada.

Vamos fechar o projeto:

→ Clique em *File>Close Project*.

10.3.3 Criando o Projeto WindowsAppCSharp

No MS Visual Studio:

→ Clique em *File>New>Project...*

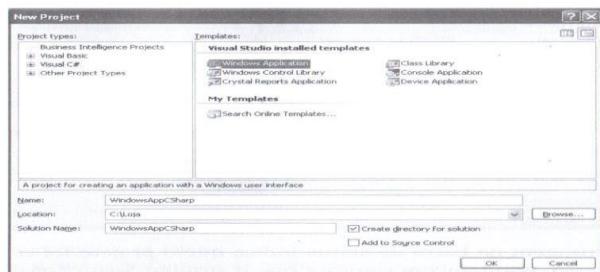
Na janela New Project, selecione:

→ No Tipo de Projeto, selecione *Visual C#*;

→ No Tipo de Template, selecione *Windows Application*;

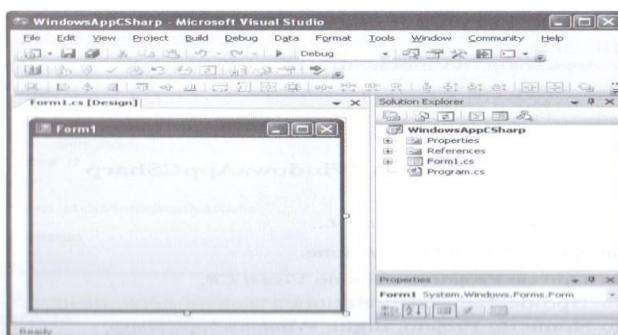
→ No Nome do Projeto, digite *WindowsAppCSharp*;

- Na Localização do Projeto, digite `C:\Loja`;
- No Solution Name, digite `WindowsAppCSharp`;
- Deixe selecionada a opção `Create directory for solution`;



→ Clique em `Ok` para criar o projeto.

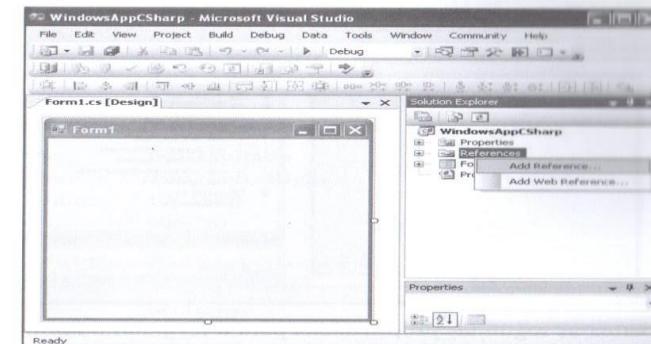
A aplicação é criada com os parâmetros que especificamos:



Tópicos de Segurança da Informação

O primeiro procedimento que faremos neste projeto será adicionar o projeto `Security` como referência. Para isso:

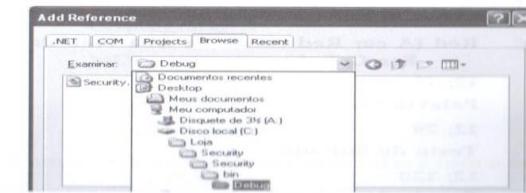
- No Solution Explorer, clique com o botão direito do mouse em `References`, e escolha a opção `Add Reference...`



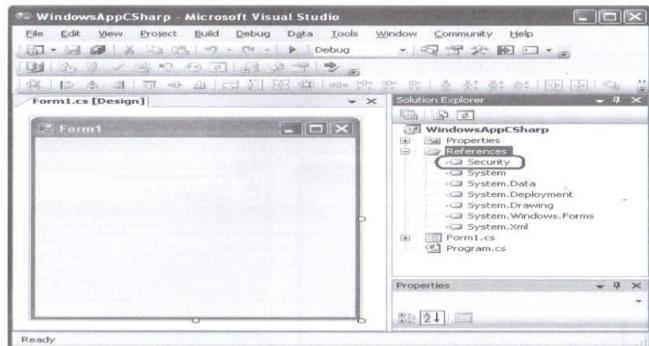
→ Na aba `Browse`, navegue até a pasta: `\Loja\Security\bin\Debug`;

→ Selecione o arquivo `Security.dll`;

→ Clique em `OK` para criar a referência.



Agora, temos a referência para o projeto *Security* criada durante o desenvolvimento de nosso exemplo:



→ Defina as seguintes propriedades para o formulário Form1:

Text: Segurança da Informação (Criptografia)
Size: 434; 252

Neste formulário criaremos três objetos do tipo *Label*, três objetos do tipo *Textbox* e três objetos do tipo *Button*.

→ Adicione três objetos do tipo *Label* ao formulário Form1 e defina as seguintes propriedades:

ForeColor: Red (A cor Red existe na Aba Web da seleção de ForeColor)
Location: 12; 13
Text: Palavra Secreta:
Location: 12; 79
Text: Texto de Entrada:
Location: 12; 120
Text: Resultado:

Tópicos de Segurança da Informação

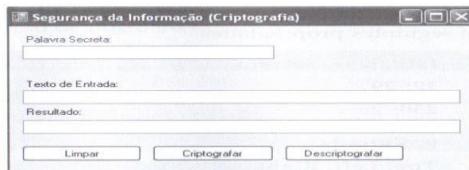
→ Adicione três objetos do tipo *Textbox* ao formulário Form1 e defina as seguintes propriedades:

Name:	txtPalavraSecreta
Location:	12; 29
Size:	230; 20
Name:	txtEntrada
Anchor:	Top, Left, Right
Location:	12; 95
Size:	400; 20
Name:	txtResultado
Anchor:	Top, Left, Right
Location:	12; 139
Size:	400; 20

→ Adicione três objetos do tipo *Button* ao formulário Form1 e defina as seguintes propriedades:

Name:	btLimpar
Location:	11; 177
Size:	110; 23
Text:	Limpar
Name:	btEncripta
Location:	131; 177
Size:	110; 23
Text:	Criptografar
Name:	btDecripta
Location:	251; 177
Size:	110; 23
Text:	Descriptografar

Neste momento, o nosso formulário estará com a seguinte aparência:



10.4 Como será o Funcionamento?

Essa aplicação será parte da sua Caixa de Ferramentas Pessoal, ou seja, será um utilitário que você vai guardar para usar sempre que necessário.

- Você vai definir uma palavra secreta que será utilizada no processo de criptografia. Esta palavra secreta, depois, será usada na sua aplicação para que seja possível descriptografar a informação em tempo de execução, obtendo assim os dados necessários;
- Após criar a sua palavra secreta, deve-se colar o texto a ser criptografado no campo *Texto de Entrada*;
- Depois, é só clicar no botão *Criptografar* para que o algoritmo de criptografia trabalhe e preencha o campo *Resultado* com a string criptografada;
- Essa string criptografada deverá substituir a string original na aplicação principal. Assim, mesmo que alguém consiga obter esse texto, dificilmente conseguirá decifrar o seu verdadeiro significado;
- Por outro lado, a aplicação principal utilizará a palavra secreta e a string criptografada, instanciará um objeto do tipo *SecureApp* e chamará o método *Decifrar()* passando essas informações como parâmetros. O resultado obtido em tempo de execução será o texto descriptografado.

Bem, já vimos como tudo isso vai funcionar, agora, vamos codificar as funcionalidades dos três botões que criamos no formulário *Form1*:

Tópicos de Segurança da Informação

Botão Limpar

- Dê um duplo clique no botão *Limpar* para codificarmos seu evento click;
- No evento click deste botão, copie e cole o seguinte trecho código:

```
txtPalavraSecreta.Text = "";
txtEntrada.Text = "";
txtResultado.Text = "";
```

Botão Criptografar

- Volte ao modo de Design e dê um duplo clique no botão *Criptografar* para codificarmos o seu evento click.

Observação: Utilize *using Security;* no início do code behind formulário.

- No evento click deste botão, copie e cole o seguinte trecho código:

```
DTICrypto myCrypto = new DTICrypto();
txtResultado.Text = myCrypto.Cifrar(txtEntrada.Text, txtPalavraSecreta.Text);
```

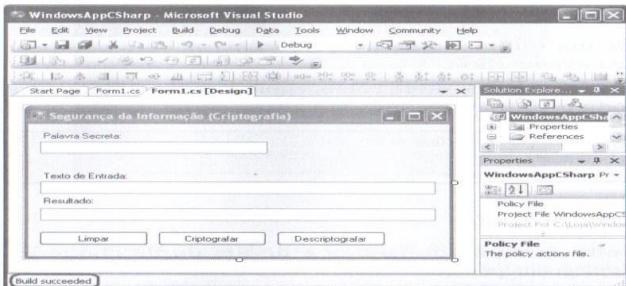
Botão Descriptografar

- Volte ao modo de Design e dê um duplo clique no botão *Descriptografar* para codificarmos o seu evento click;
- No evento click deste botão, copie e cole o seguinte trecho código:

```
DTICrypto myCrypto = new DTICrypto();
txtResultado.Text = myCrypto.Decifrar(txtEntrada.Text, txtPalavraSecreta.Text);
```

- Dê um *Build* para compilar a aplicação.

A mensagem *Build succeeded* destacada na barra de status da figura mostrada a seguir, indica que a nossa aplicação foi compilada com sucesso:



10.5 Implementando a Segurança na nossa Aplicação

Para demonstrar o funcionamento do algoritmo de criptografia que estamos estudando, vamos implementar a segurança no nosso projeto *UIWindows*.

E quais são os passos?

Para isso, vamos adicionar a dll *SecureApp.dll* como referência no projeto *UIWindows*. Assim, poderemos utilizar o método Decifrar e obter, em tempo de execução, as informações necessárias para a conexão com o banco de dados.

Vamos prosseguir com a implementação de segurança na nossa interface Windows.

No MS Visual Studio:

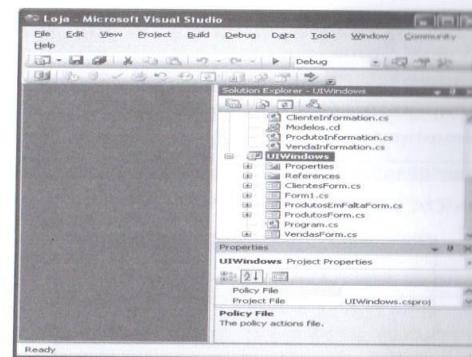
→ Abra a solução *Loja* através do arquivo Loja.sln;

No Solution Explorer:

→ Clique com o botão direito do mouse no projeto *UIWindows*;

→ Selecione a opção *Set as StartUp Project* para defini-lo como projeto inicial;

→ O nome do projeto ficará em negrito, indicando que ele é o projeto inicial da solução.

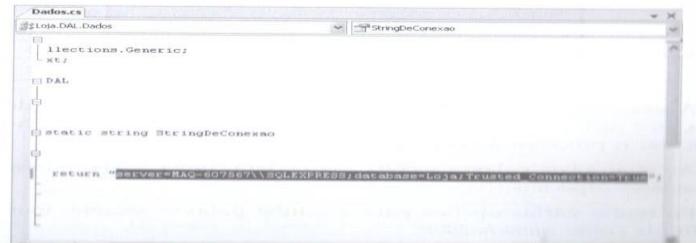


Agora, vamos copiar a string de conexão que atualmente está no texto aberto na nossa Solução.

No projeto *DAL*:

→ Abra a classe *Dados.cs*;

→ Selecione a string de conexão que está definida entre aspas no código fonte, conforme figura a seguir:



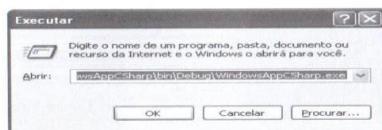
Agora é o momento de executarmos a nossa aplicação WindowsApp CSharp. Essa aplicação pode ser chamada externamente ao MS Visual Studio.

No MS-Windows:

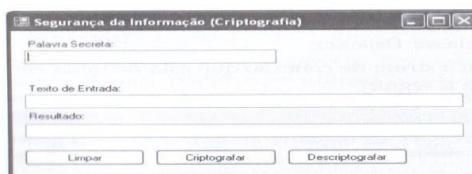
- Clique na opção *Iniciar>Executar...*
- Cole o caminho da aplicação:

`C:\Loja\WindowsAppCSharp\WindowsAppCSharp\bin\Debug\WindowsAppCSharp.exe`

→ Clique em *OK* para executá-la:



A aplicação será executada como a seguir:

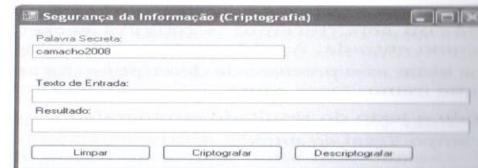


A sua string de conexão com o banco de dados é diferente da minha, por isso, você precisará criar a sua própria palavra secreta para executar o processo de criptografia.

Ao criar a sua palavra secreta, ela poderá conter letras, números e caracteres especiais.

Eu tenho várias opções para a minha palavra secreta, mas irei defini-la como *camacho2008*:

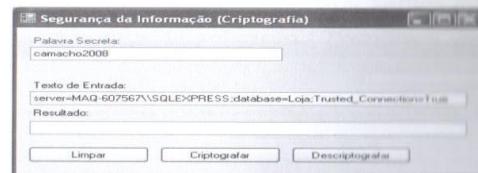
Tópicos de Segurança da Informação



O texto de entrada vai ser a string de conexão que copiamos classe *Dados.cs* do projeto *DAL*:

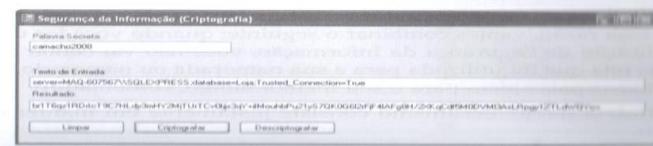
- Copie a sua string de conexão da classe *Dados.cs* do projeto *DAL*.
- Na aplicação de criptografia, cole essa string no campo *Texto Entrada*.

O modelo com os meus dados ficou assim:



→ Clique no botão *Criptografar* para ver o resultado.

Observação: É possível aumentar o tamanho da janela da aplicação de Criptografia para visualizar o resultado inteiro.

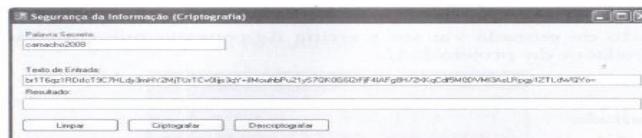


Na nossa interface UIWindows faremos o processo de descriptografia, ou seja, teremos a palavra secreta e a string criptografada como entrada. A saída será a nossa string de conexão.

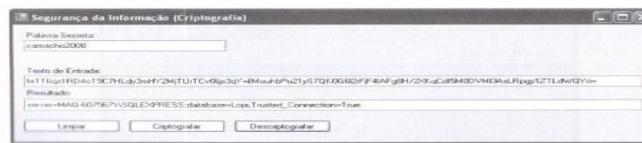
Nós podemos testar esse processo de descriptografia através da aplicação, e é isso que iremos fazer agora.

→ Recorte todo o texto do resultado criptografado;

→ Cole no campo *Texto de Entrada*:



→ Clique no botão *Descriptografar* para verificarmos o resultado.



Muito bom! Vimos que o resultado é exatamente a string de conexão original.

Perceba que a *Palavra Secreta* é a mesma usada no processo de criptografia. Se não soubermos qual palavra secreta foi utilizada no processo de criptografia, o processo de descriptografia torna-se impossível.

Por essa razão, vamos combinar o seguinte: quando você for usar a sua aplicação de Segurança da Informação, você não vai contar a palavra secreta que foi utilizada para a sua namorada ou namorado, nem para os seus pais e nem para o seu melhor amigo, combinado?!

A eficácia desse mecanismo consiste justamente em manter esse segredo.

Dicas: Sabe aqueles caracteres que não estão no teclado? Por exemplo, se você manter pressionada a tecla *<Alt>* e digitar 2, ao soltar a tecla *<Alt>* aparecerá um smiley. Esses caracteres podem ser usados para aumentar ainda mais a segurança da sua palavra secreta.

Se por acaso, a sua string de conexão também contiver o caractere barra invertida (\) para identificar o nome da instância do banco de dados, tente usar somente uma barra ao invés de duas quando estiver criptografando.

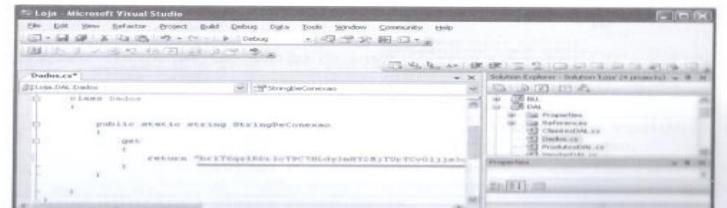
Agora que já sabemos como esse mecanismo funciona, a nossa tarefa em tornar segura a nossa interface UIWindows vai ser concluída em duas etapas:

1. Vamos substituir a string de conexão da classe dados pela string criptografada;
2. No projeto de interface UIWindows, vamos adicionar a dll *SecureApp.dll* como referência e utilizaremos o método *Decifrar()* em todos os pontos da aplicação UIWindows em que exista a conexão com o banco de dados.

Primeiro vamos executar a Etapa 1:

→ Na aplicação de Segurança da Informação, copie a string de conexão criptografada;

→ Abra a classe *Dados.cs* no projeto *DAL* e substitua a string de conexão em “texto aberto” pela string de conexão criptografada, como a seguir:



→ Salve o arquivo *Dados.cs* e dê um *Build* no projeto *DAL*.

Pronto, a Etapa 1 está concluída.

Agora, iniciaremos a Etapa 2:

→ No projeto de interface *UIWindows*, vamos adicionar a dll *SecureApp.dll* como referência e utilizaremos o método *Decifrar()* em todos os pontos da aplicação *UIWindows* em que existe a conexão com o banco de dados.

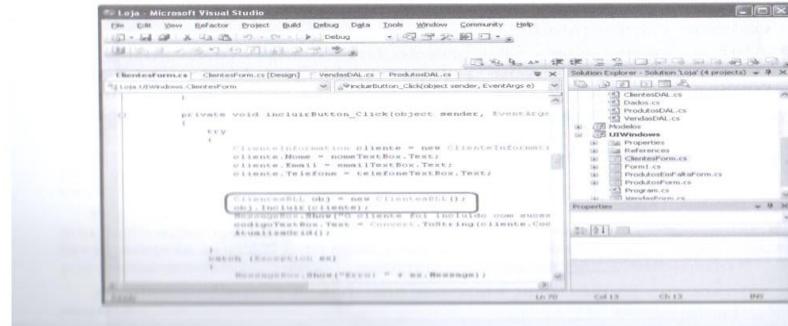
Espera um pouco...

Antes de incluirmos a dll *SecureApp.dll* como referência no projeto *UIWindows*, peço que você analise o projeto *UIWindows* e me responda:

- Em quais formulários do projeto *UIWindows* nós vamos usar o método *Decifrar()* para obter a string de conexão?

Vamos abrir qualquer formulário do projeto *UIWindows*, o *ClientesForm.cs*, por exemplo, e vamos fazer uma rápida análise.

Se clicarmos no botão *Incluir* para ver o código, perceberemos que estamos instanciando um objeto da Camada *BLL* e chamando o método *Incluir()* desta camada.



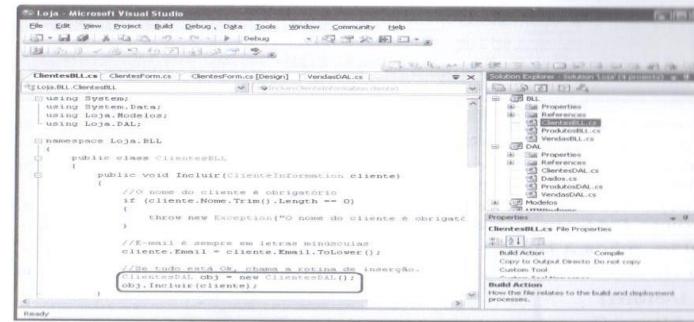
Isso significa que a string de conexão não é lida na camada de apresentação?

A sua conclusão está correta.

Na primeira linha destacada na imagem anterior, vamos clicar com o botão direito sobre *ClientesBLL* e escolher *Go to Definition* para visualizarmos a Camada *BLL*.

No Camada *BLL*, estamos instanciando um objeto da Camada *DAL* e chamando o método *Incluir()* sem mencionar a string de conexão.

Então o único lugar onde o projeto *UIWindows* obtém a string de conexão é na própria Camada *DAL*?



É isso mesmo.

Vamos clicar com o botão direito do mouse sobre *ClientesDAL* na primeira linha destacada na figura anterior e escolher a opção *Go To Definition*.

Dicas: Utilize a opção *Go To Definition* na necessidade de visualizar rapidamente o código-fonte de uma classe.

Outra opção prática é a *Find All References*, ela faz com que o Microsoft Visual Studio exiba um relatório com todos os arquivos que utilizam a classe pesquisada.

```

% Loja - Microsoft Visual Studio
File Edit View Refactor Project Build Debug Data Tools Window Community Help
ClientesDAL.cs ClientesLL.cs ClientesForm.cs ClientesForm.cs [Design] Solution Explorer Solution 'Loja' (4 projects) ...
using System;
using System.Collections.Generic;
using System.Text;
using Loja.Modelos;
using System.Data.SqlClient;
using System.Data;
namespace Loja.DAL
{
    public class ClientesDAL
    {
        public void Incluir(ClienteInformation cliente)
        {
            //conexao
            SqlConnection cn = new SqlConnection();
            try
            {
                cn.ConnectionString = Lojas.StringDeConexao();
                //comando
                SqlCommand cmd = new SqlCommand();
                cmd.Connection = cn;
                cmd.CommandType = CommandType.StoredProcedure;
                //nome da stored procedure
            }
            catch { }
        }
    }
}

```

Aí está. Como vimos, é somente na Camada DAL que fazemos a leitura da string de conexão.

Sendo assim, é nessa camada que vamos trabalhar e não no projeto de interface como tínhamos imaginado inicialmente.

A primeira coisa a ser feita é um levantamento de todos os pontos do projeto DAL em que a string de conexão é utilizada.

Então, mãos à obra:

Na classe *ClientesDAL*, a string de conexão é utilizada nos métodos:

- Incluir()
- Alterar()
- Excluir()
- Listagem()

Na classe *ProdutosDAL*, a string de conexão é utilizada em:

- ProdutosEmFalta()
- Incluir()
- Alterar()
- Excluir()
- Listagem()

Na classe *VendasDAL*, a string de conexão é utilizada em:

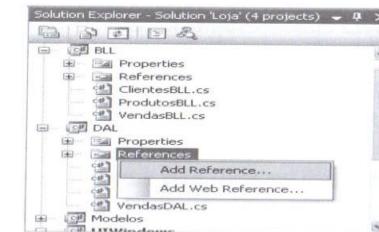
- ListaDeProdutos()
- ListaDeClientes()
- Incluir()

Bem, após esse levantamento percebemos que temos muito trabalho a fazer. Nesse caso é melhor começarmos imediatamente.

10.6 Adicionando a dll SecureApp.dll como Referência

Em References do projeto *DAL*:

- Clique com o botão direito do mouse e escolha a opção *Add Reference...*



Na janela Add Reference:

→ Digite o caminho a seguir no campo Nome do arquivo:

C:\Loja\SecureApp\SecureApp\bin\Debug\SecureApp.dll

→ Clique em OK para adicionar a referência.

Agora, já podemos ver a referência para a dll *SecureApp* criada.

Realizando o mesmo procedimento para os métodos Alterar() e Excluir():

- Localize a linha a seguir no método Listagem()

```
SqlDataAdapter da = new SqlDataAdapter(strSql, Dados.StringDeConexao);
```

- Na linha anterior a citada acima, crie uma instância do objeto DTICrypto() como a seguir:

```
DTICrypto objCrypto = new DTICrypto();
```

- Vamos alterar a linha em que estamos criando o SqlDataAdapter. Chamaremos o método Decifrar() nesse mesmo comando como segue:

```
SqlDataAdapter da = new SqlDataAdapter(strSql, objCrypto.Decifrar(Dados.StringDeConexao, "camacho2008"));
```

Lembre-se de sempre usar a sua própria palavra secreta ao invés de camacho2008.

- Dê um *Build* para compilar o Projeto *DAL* e verificar se está tudo correto;

- Dê um *Run* na aplicação e navegue no Formulário de Produtos, verificando se as funcionalidades estão sendo executadas corretamente.

Classe *VendasDAL*

Para concluir a implementação do nosso mecanismo de segurança, vamos realizar três alterações na classe *VendasDAL*.

Precisaremos alterar: ListaDeProdutos(), ListaDeClientes() e Incluir().

- Na classe *VendasDAL.cs*, localize as três vezes em que aparece a linha:

```
cn.ConnectionString = Dados.StringDeConexao;
```

- E substitua essa linha pelas linhas a seguir:

```
DTICrypto objCrypto = new DTICrypto();
```

Lembre-se de sempre usar a sua própria palavra secreta e de usar *using SecureApp;* no inicio do code behind.

- Dê um *Build* para compilar o Projeto *DAL* e verificar se está tudo correto;

- Dê um *Run* na aplicação e teste o Formulário de Vendas.

As versões dos arquivos da Camada *DAL* após a implementação da criptografia estão disponíveis em:

Loja_Fontes\DAL\ClientesDAL_Final.cs

Loja_Fontes\DAL\ProdutosDAL_Final.cs

Loja_Fontes\DAL\VendasDAL_Final.cs

Dica: Os nomes dos arquivos no projeto não devem ser alterados. Estas versões só contêm *_Final* no nome para que fiquem diferentes dos arquivos originais que foram criados inicialmente. Se você for utilizar esses códigos, abra os arquivos no notepad para copiar seu conteúdo e colar nos arquivos correspondentes dentro do MS Visual Studio.

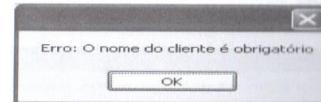
11

Finalizando

11.1 Testando as Regras de Negócio na Aplicação para Windows

Na Camada BLL, implementamos algumas regras de negócio.

Tente incluir um cliente com o campo nome em branco, e você receberá o alerta a seguir:



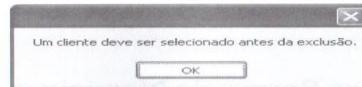
Após clicar em *OK*, a inclusão não é permitida e você estará de volta ao Formulário de Inclusão.

Outra regra implícita que colocamos, ou seja, não está visível para o usuário, é que o campo e-mail é armazenado em letras minúsculas. Experimente incluir um cliente escrevendo o e-mail em letras maiúsculas e você verá que a nossa Camada BLL se encarregará de armazenar esse campo em letras minúsculas. Se a regra aplicada na inclusão não for aplicada na alteração, a nossa regra de negócio não será garantida.

Vejamos um exemplo: se a regra do nome obrigatório existir sómente no formulário de inclusão, o usuário poderá alterar um registro através do formulário de alteração deixando o nome do cliente vazio. Assim teríamos no banco de dados um registro inconsistente ou seja, um registro com uma informação que não obedece a nossa regra de negócio.

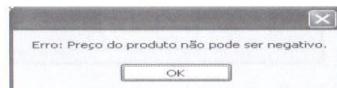
Se tentarmos alterar um cliente deixando o nome em branco, a camada de regras de negócio também entrará em ação para evitar que isso aconteça.

Se você estiver com o cursor numa linha do DataGridView que não tenha registro e clicar no botão *Excluir*, receberá o aviso:



Um alerta semelhante será exibido ao clicar no botão *Alterar*, pois, definimos como regra que um registro precisa estar selecionado para realizarmos estas operações.

Mesmo que uma informação seja fornecida, implementamos algumas especificações que devem ser respeitadas no nosso projeto. Por exemplo, se tentarmos cadastrar um valor negativo no preço de um produto, receberemos o aviso:



Dessa forma, podemos ver que a nossa camada de regras de negócio vai além da tarefa de manter a integridade dos dados, evitando o armazenamento de campos em branco. Ela também nos permite garantir que as regras especificadas para o projeto sejam respeitadas, como no exemplo da inclusão do preço do produto que não pode ser um número negativo.

11.2 Estrutura de Arquivos

Na pasta C:\Loja\UIWindows\UIWindows\bin\Debug, você encontrará os arquivos:

- Modelos.dll
- DAL.dll
- BLL.dll
- SecureApp.dll
- UIWindows.exe

Os arquivos *Modelos.dll* e *DAL.dll* referem-se a camada de banco de dados.

O arquivo *BLL.dll* refere-se a camada de regras de negócio.

O arquivo *SecureApp.dll* refere-se a camada de segurança.

E o arquivo *UIWindows.exe* é a interface com o usuário que roda no MS-Windows.

Dicas: Se você copiar somente esses cinco arquivos em qualquer pasta do microcomputador de um usuário, bastará que ele execute o aplicativo *UIWindows.exe* para iniciar o Sistema *Loja.Ne* e utilizar todas as funcionalidades.

Você pode renomear o arquivo *UIWindows.exe* para o nome que desejar.

Exemplo: MeuProjeto.exe

11.3 Vantagens do Modelo de Desenvolvimento em Camadas

Agora, imagine que você é o Gerente de Projeto do Sistema *Loja.Ne* e recebe o seguinte e-mail de um de seus clientes:

Prezado Sr.,

Como usuário do Sistema Loja .Net desenvolvido pela sua empresa, informo que precisamos que o Sr. nos envie uma atualização para atender as nossas novas necessidades.

Nossa empresa está em contínuo crescimento e em breve contrataremos uma empresa de Call Center que oferecerá promoções especiais aos nossos clientes. Dessa forma, precisamos fazer com que não seja permitido o cadastramento de clientes sem o número de telefone, pois essa informação será muito importante para o nosso sucesso.

Por favor, nos notifique assim que a nova versão do sistema estiver disponível para testes.

Atenciosamente,

Carlos Camacho

Analista de Sistemas

Algumas perguntas:

- Qual camada do Sistema *Loja.Net* você vai precisar alterar para atender a solicitação do cliente?

Isso mesmo, você vai precisar alterar somente a Camada BLL.

- Depois que você realizar as atualizações do Sistema, precisará disponibilizar os cinco arquivos que o compõem para o cliente?

Não, é só você enviar o arquivo referente a camada que foi atualizada. Neste caso, bastaria enviar o arquivo BLL.dll com instruções para que o cliente substitua o arquivo atual por este novo arquivo para realizar a atualização do sistema.

Neste livro, vimos como construir uma aplicação .Net utilizando o conceito de desenvolvimento em camadas.

Neste projeto, você teve a oportunidade de praticar todas as etapas da implementação do desenvolvimento de uma aplicação .Net, adquirindo conhecimentos sobre:

- Comunicação entre as camadas;
- Vantagens do modelo de desenvolvimento em camadas;
- Controle de transações do banco de dados com o ADO .Net;
- Construção de uma aplicação para Windows;
- Práticas comuns para a construção de uma interface Web.

Implementou as camadas:

- Camada de Acesso a Dados ou Data Access Layer (DAL);
- Camada de Regras de Negócio ou Business Logic Layer (BLL);
- Camada de Interface do Usuário ou User Interface (UI).

Você adquiriu habilidade na construção de aplicativos capazes de reutilizar a Camada de Acesso a Dados e a Camada de Regras de Negócio. Na Camada de Acesso a Dados, você aprendeu como desenvolver aplicações utilizando sempre as stored procedures (*ClientsDAL.cs*) e, também verificou que é possível o desenvolvimento de aplicações sem usar nenhuma stored procedure (*ProdutosDAL.cs*).

Se por acaso você estiver trabalhando em um projeto complexo, com uma grande infra-estrutura de banco de dados e diversos formulários,

Finalizando

Se você for o responsável pela manutenção do sistema, para a alteração ou criação de novos formulários, terá confiança no seu trabalho, pois entende como as camadas se relacionam e tem know-how no processo de criação dos objetos mais comuns de um formulário.

Tivemos a oportunidade de conhecer a importância da Segurança da Informação e vimos como um algoritmo de criptografia é implementado em uma situação real.

É importante lembrar que para ter sucesso na área de TI você deve manter-se sempre atualizado com as tecnologias mais recentes. Concordo que não é uma tarefa fácil, pois em um curto espaço de tempo surgem inovações incríveis (...). Por outro lado, investir no constante aperfeiçoamento é um mal necessário para quem almeja o sucesso.

Desejo que os conhecimentos adquiridos neste livro sejam úteis no seu dia-a-dia.

Um forte abraço,

Carlos Camacho

Referências

GUNDERLOY, Mike. *Developing and Implementing Windows-Based Applications with Microsoft Visual Basic .NET and Microsoft Visual Studio .NET*. MCAD/MCSD Training Guide Exam (70-306). Indianapolis, USA: Publishing, 2003. 1152 p.

MICROSOFT CORPORATION. *Developing Microsoft ASP .NET Web Applications Using Visual Studio .NET*. MSDN Training. Redmond, USA: Microsoft Press, 2002. 794 p.

MICROSOFT CORPORATION. *Programming with C#*. MSDN Training. Redmond, USA: Microsoft Press, 2002. 739 p.

Informação Importante:

Com o intuito de proporcionar ao leitor maior conforto e aproveitamento, os códigos utilizados neste livro foram disponibilizados pela VisualBooks Editora para download.

Visite: <www.visualbooks.com.br>.

Esta obra utiliza o conceito de desenvolvimento em camadas de forma totalmente prática.

Através da criação de uma aplicação para MS-Windows (Windows), o leitor conhecerá o desenvolvimento da Camada de Acesso a Dados ou DAL (Data Access Layer), da Camada de Regras de Negócio ou BLL (Business Logic Layer) e da Camada de Apresentação ou UI (User Interface).

O leitor aprenderá:

- O processo de criação de banco de dados por meio de tabelas e procedimentos armazenados (stored procedures);
- A criação de classes para gerenciamento de banco de dados usando stored procedures;
- Como administrar transações no banco de dados via programação;
- A implementação de classes que visam contemplar as regras de negócios;
- A criação de formulários Windows e uma página Web com opções de inserção, alteração, exclusão e visualização de dados;
- Como criar e utilizar um algoritmo de criptografia para tornar a aplicação segura;
- Como o CLR (Common Language Runtime) permite a compatibilidade entre diferentes linguagens de programação.

O livro é indicado para uso em laboratórios de informática e para profissionais, pois possui uma linguagem simples e fácil. É recomendado para estudantes e programadores que desejem iniciar ou aperfeiçoar seus conhecimentos em desenvolvimento de sistemas com a linguagem C# .Net.

Pode ser utilizado como material de apoio para cursos de graduação ou extensão.

Visual Books
www.visualbooks.com.br

