

Curso de C#

Aula 3

Professores

Célio Alencar de Assis

Fábio Duarte Machado



O que será abordado Hoje

- Exceções e stringBuilder
- Criando e Referenciando Bibliotecas
- Padrão de Projeto AbstractFactory, MVP e Repository

Exceções


O C# fornece um conjunto de instruções e classes para tratamento de exceções (Erros)

Uso do bloco try catch


```
protected void btnOK_Click(object sender, EventArgs e)
{
    string[] andares = new string[3];
    andares[0] = "Terreo";
    andares[1] = "Primeiro";
    andares[2] = "Segundo";

    try
    {
        int andar = int.Parse(txtAndar.Text);
        lblMsg.Text = "O andar é: " + andares[andar];
    }
    catch
    {
        lblMsg.Text = "Não existe este andar";
    }
}
```

Aqui ele tenta
executar o código



Caso ocorra uma
exceção ele executa
este:

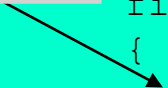


Exemplo 1

```
protected void btnOK_Click(object sender, EventArgs e)
{
    string[] andares = new string[3];
    andares[0] = "Terreo";
    andares[1] = "Primeiro";
    andares[2] = "Segundo";

    try
    {
        int andar = int.Parse(txtAndar.Text);
        lblMsg.Text = "O andar é: " + andares[andar];
    }
    catch
    {
        lblMsg.Text = "Não existe este andar";
    }
    finally
    {
        lblMsg.Text += " Obrigado, Volte sempre!!";
    }
}
```

É executado independentemente se houve a exceção ou não.



O `finally` tem como função limpar eventuais variáveis alteradas durante a exceção. Em suma, por a casa em ordem.

0 objeto exception

Na instrução `catch` é possível “pegar” um objeto derivado da classe `System.Exception` que pode trazer uma série de informações úteis para o tratamento da exceção.

Exemplo

```
class Teste {  
    static void Main(string[] args) {  
        string []andares = new string[3];  
        andares[0] = "Terreo";  
        andares[1] = "Primeiro";  
        andares[2] = "Segundo";  
  
        Console.Write("Digite o número do andar: ");  
        int andar = int.Parse(Console.ReadLine());  
  
        try {  
            Console.WriteLine(andares[andar]);  
        } catch (System.Exception e) {  
            Console.WriteLine("Erro: " + e.Message);  
            Console.WriteLine("Fonte: " + e.Source);  
        }  
        Console.WriteLine("Fim do programa.");  
        Console.ReadLine();  
    }  
}
```

O uso do Exception



1

Fazer uma pequena aplicação usando o bloco `try catch finally`

StringBuilder

É extremamente útil para a concatenação de grandes strings.

StringBuilder

```
public partial class WebStringBuilder : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        System.Text.StringBuilder mSB = new System.Text.StringBuilder();

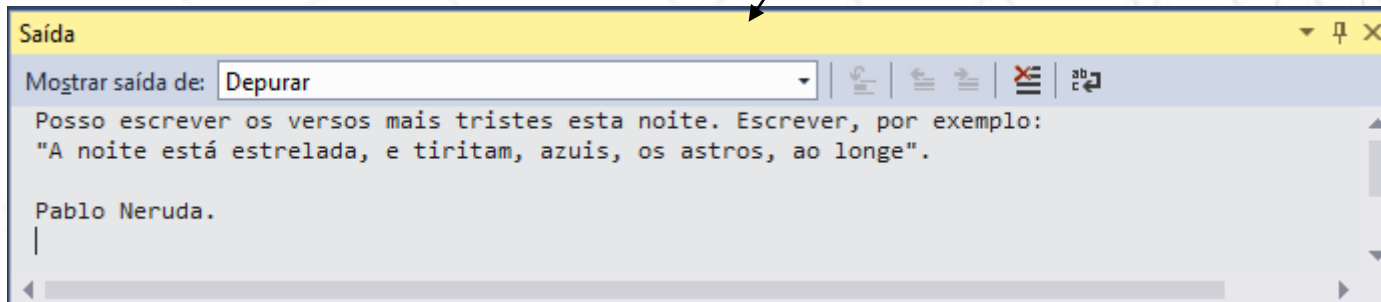
        mSB.Append("Posso escrever os versos mais tristes esta noite. ");
        mSB.Append("Escrever, por exemplo: \n\"A noite está estrelada,");
        mSB.Append(" e tiritam, azuis, os astros, ao longe\".");
        mSB.Append("\n\nPablo Neruda.");

        System.Diagnostics.Debug.WriteLine(mSB);
    }
}
```

Faz as
concatenações

Cria a instância do
StringBuilder

Imprime o resultado



1

Testar o código abaixo e observar as diferenças de tempo

```
protected void Page_Load(object sender, EventArgs e)
{
    System.Text.StringBuilder mSB = new System.Text.StringBuilder();

    DateTime dateTime = DateTime.Now;

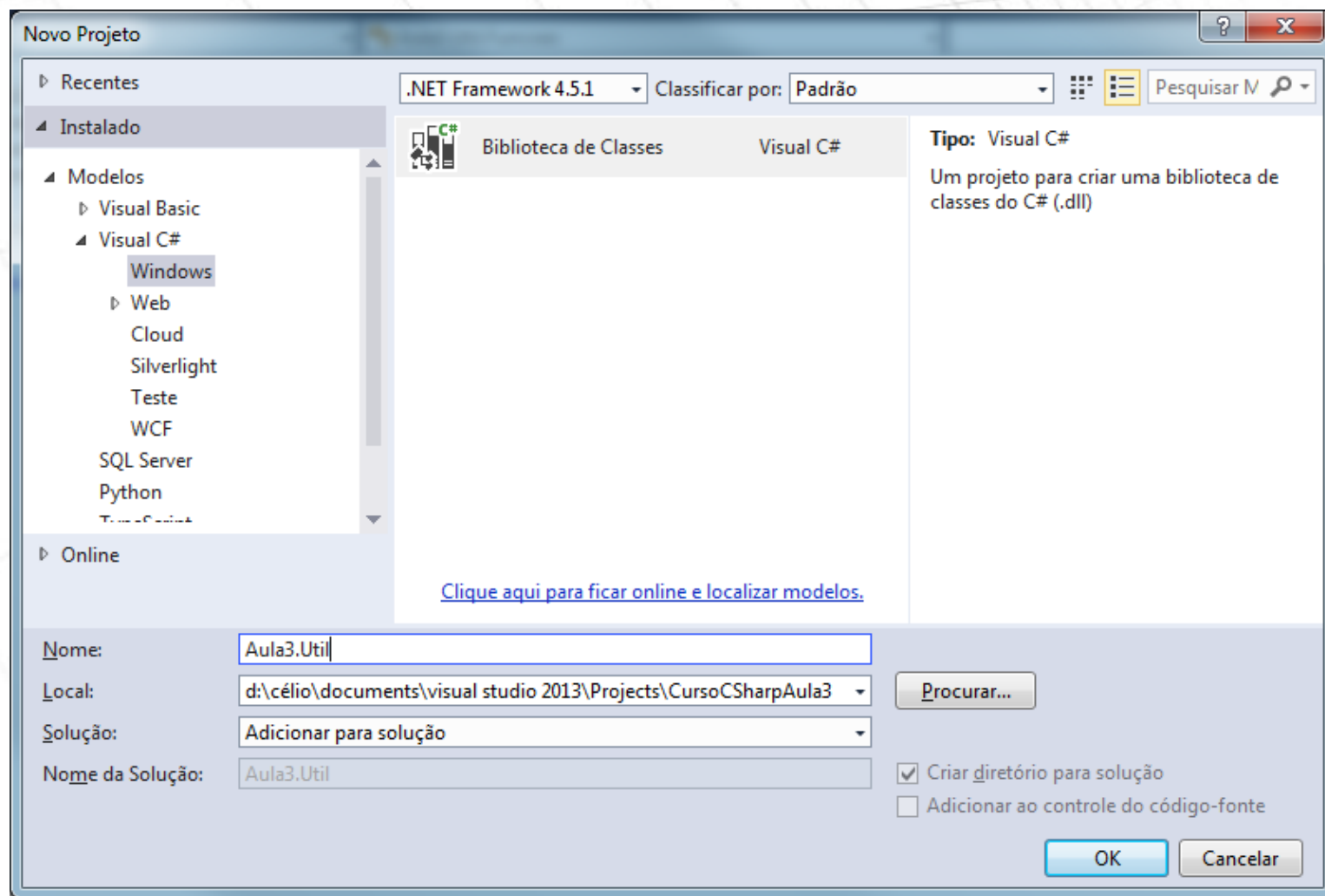
    //Concatenação com StringBuilder
    for (int i = 0; i < 20000; i++)
    {
        mSB.Append("Novo texto Novo texto Novo texto Novo texto");
    }
    System.Diagnostics.Debug.WriteLine("Concluido com stringBuilder em:\t\t" +
        DateTime.Now.Subtract(dateTime).ToString());

    dateTime = DateTime.Now;

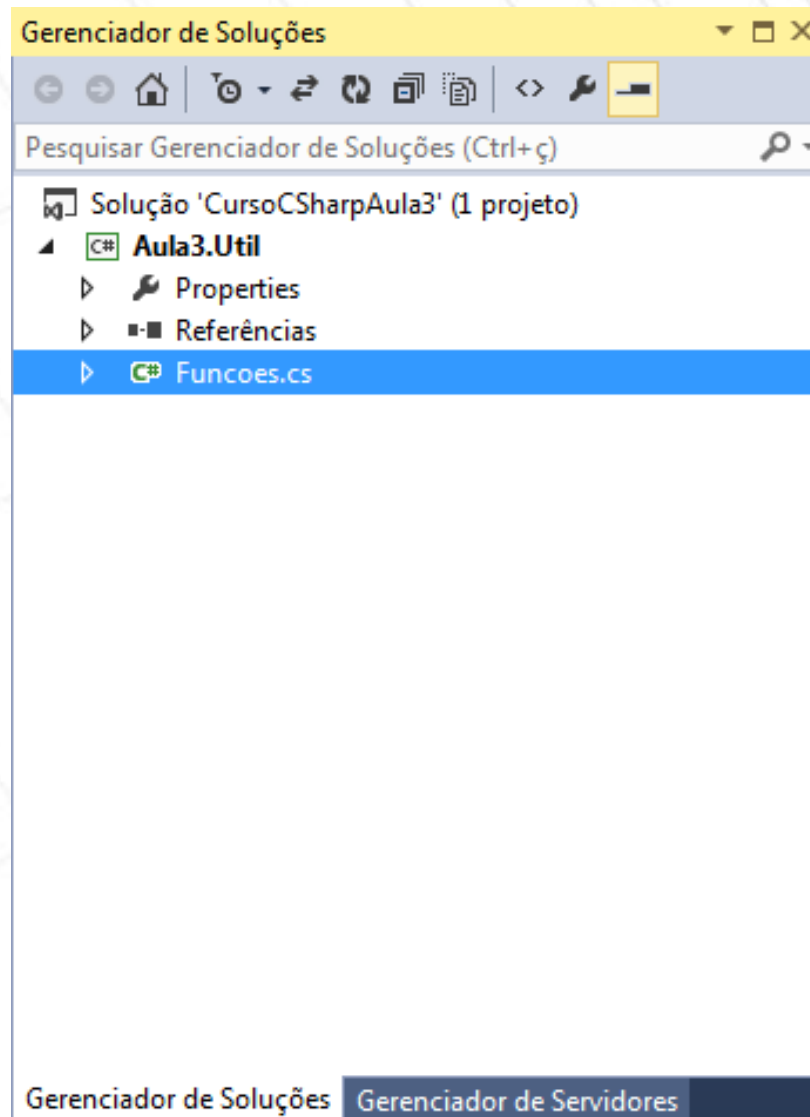
    //Concatenação de String
    string strteste = "";
    for (int i = 0; i < 20000; i++)
    {
        strteste += "Novo texto Novo texto Novo texto Novo texto";
    }
    System.Diagnostics.Debug.WriteLine("Concluido usando concatenação em :\t" +
        DateTime.Now.Subtract(dateTime).ToString());
}
```


Biblioteca de Classes, Criando e Referenciando

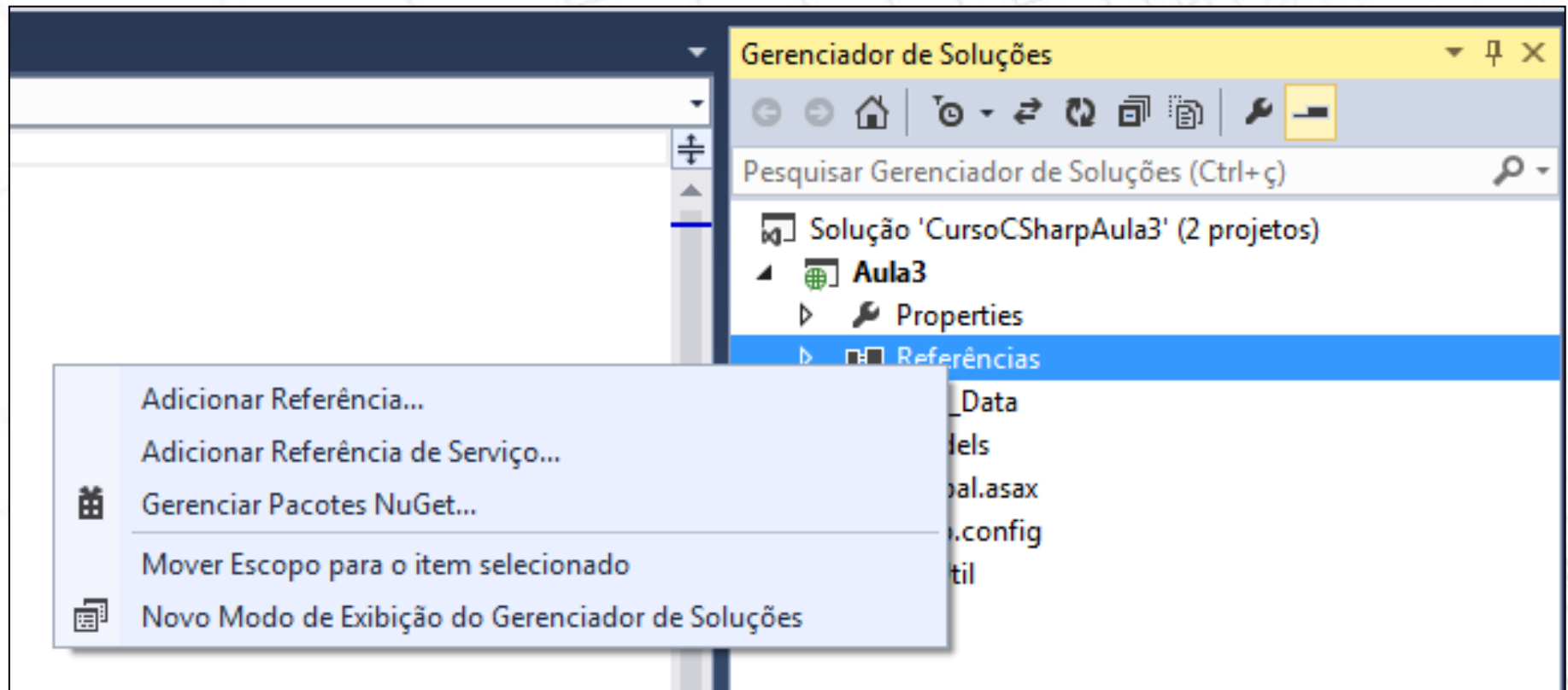
Criando uma Biblioteca de Classes



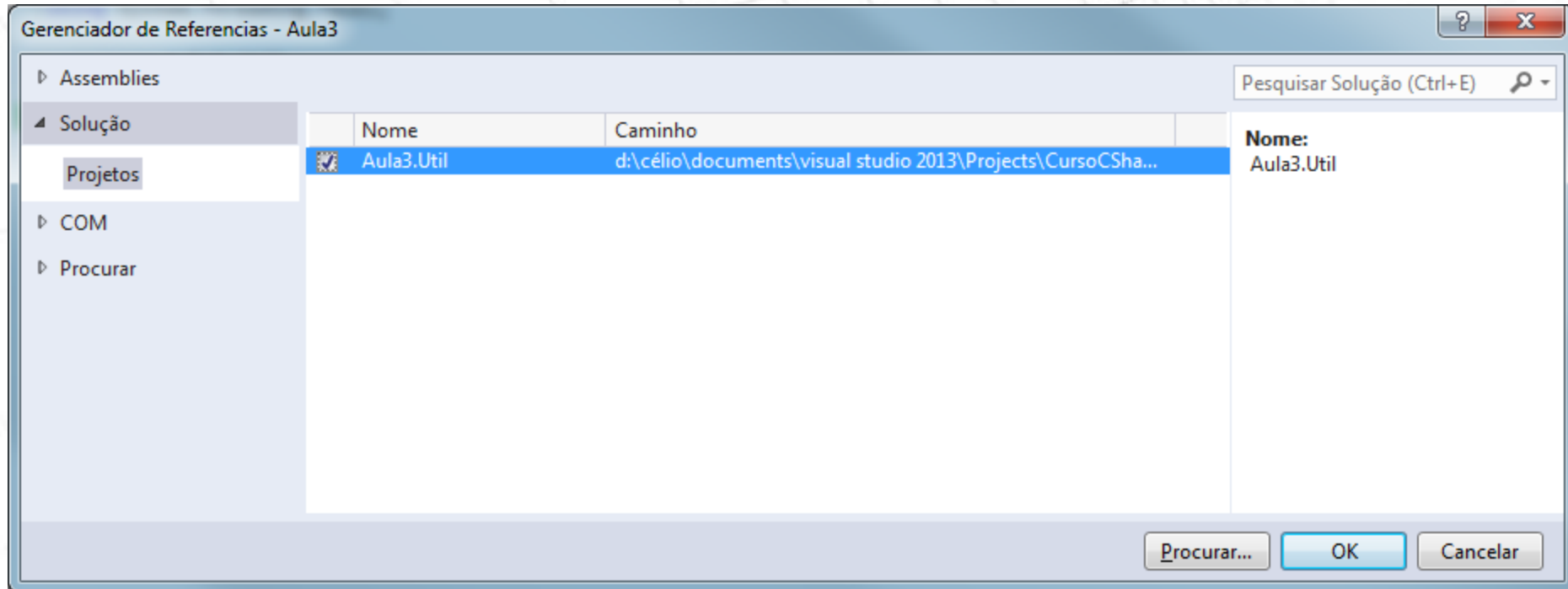
Biblioteca de Classes no Gerenciador de Soluções



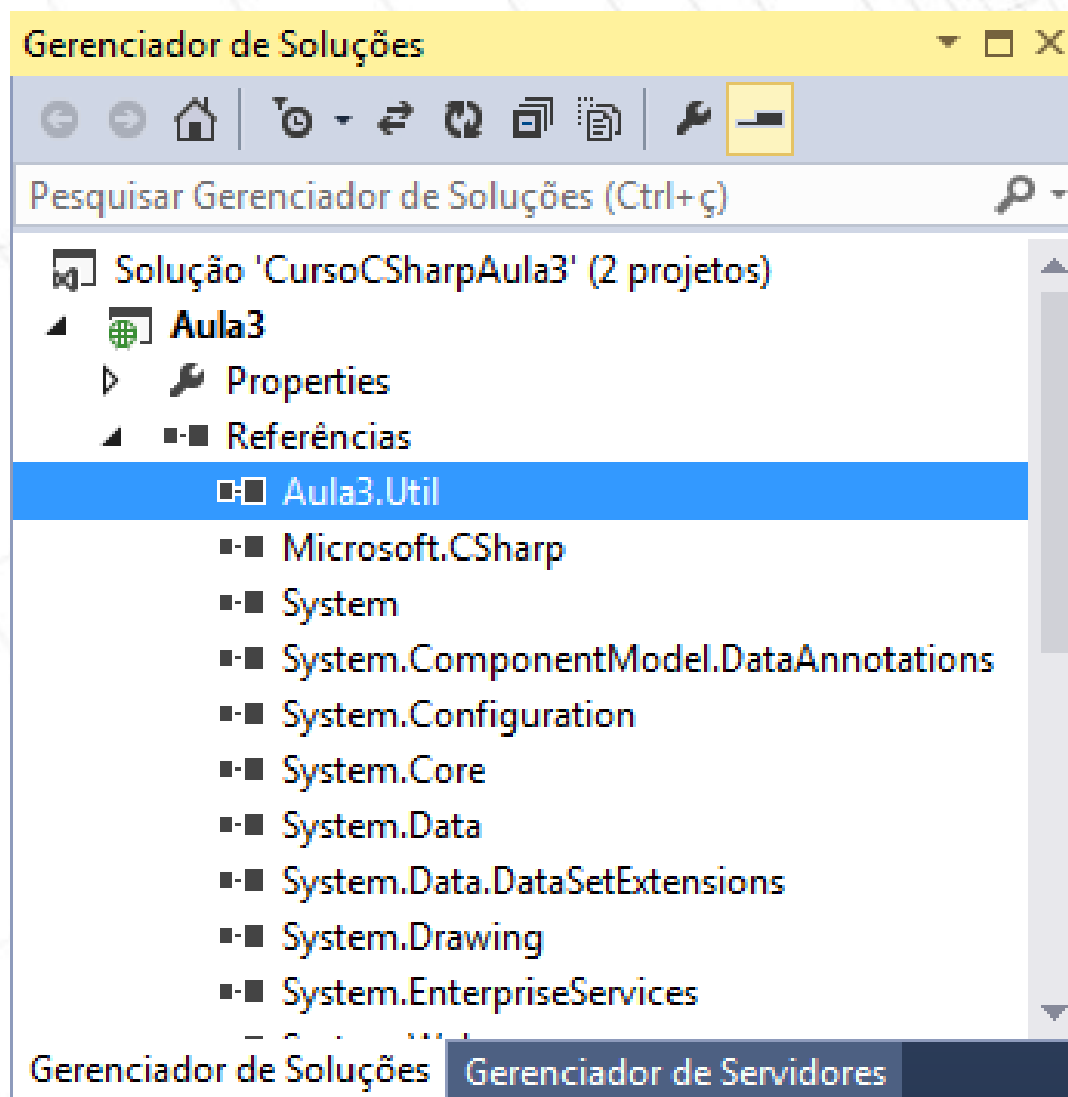
Referenciando uma Biblioteca de Classes



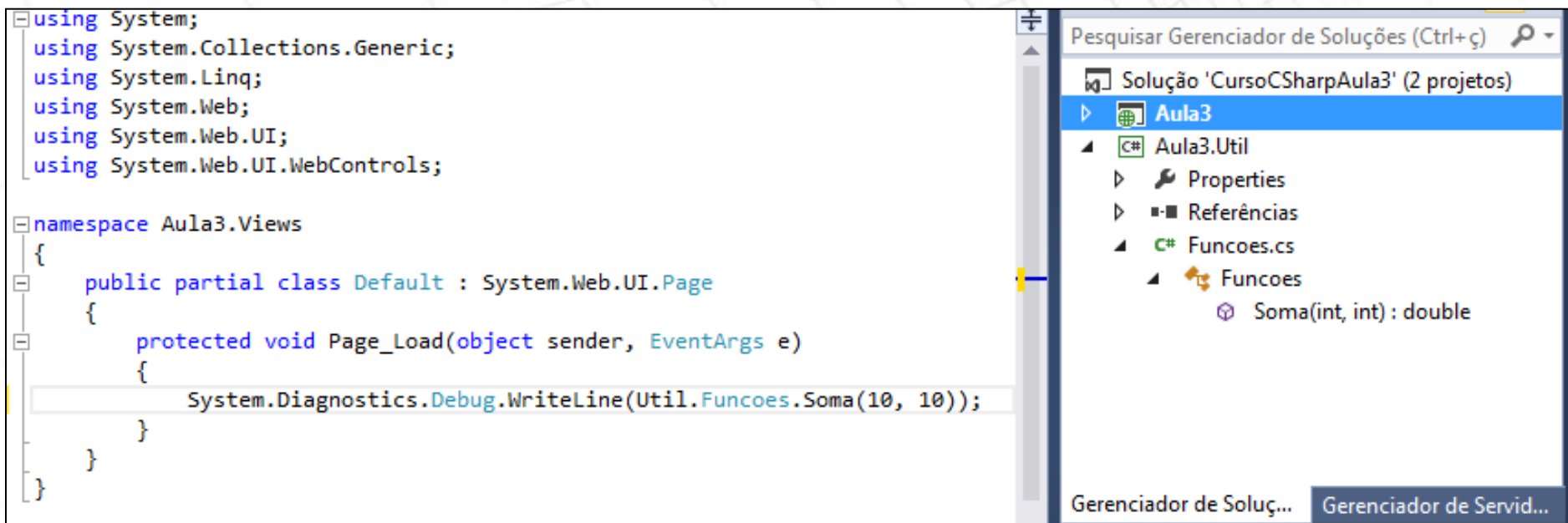
Referenciando uma Biblioteca de Classes



Biblioteca de Classes Referenciada



Usando a Biblioteca de Classes Referenciada



The image shows a Visual Studio interface with a C# code file on the left and the Solution Explorer on the right.

Code File:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace Aula3.Views
{
    public partial class Default : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            System.Diagnostics.Debug.WriteLine(Util.Funcoes.Soma(10, 10));
        }
    }
}
```

Solution Explorer:

- Solução 'CursoCSharpAula3' (2 projetos)
 - Aula3
 - Aula3.Util
 - Properties
 - Referências
 - Funcoes.cs
 - Funcoes
 - Soma(int, int) : double

At the bottom of the Solution Explorer, there are two buttons: "Gerenciador de Soluções..." and "Gerenciador de Servid...".

Abstract Factory

Intenção

- Fornecer uma interface para criar famílias de objetos relacionados ou dependentes sem especificar a sua classe concreta.

Descrição

Nesse padrão, a fábrica (conhecida como factory) recebe solicitações por objetos concretos a partir de um cliente. Este padrão pode ser utilizado quando um sistema de software precisa ser independente de como classes concretas (produtos) são criadas, compostas ou representadas. No padrão, uma fábrica fica responsável por encapsular a criação de uma família de produtos. Nesse caso, um cliente precisa conhecer somente as interfaces desses produtos, não a sua implementação, aumentando o encapsulamento e abstração.



Benefícios

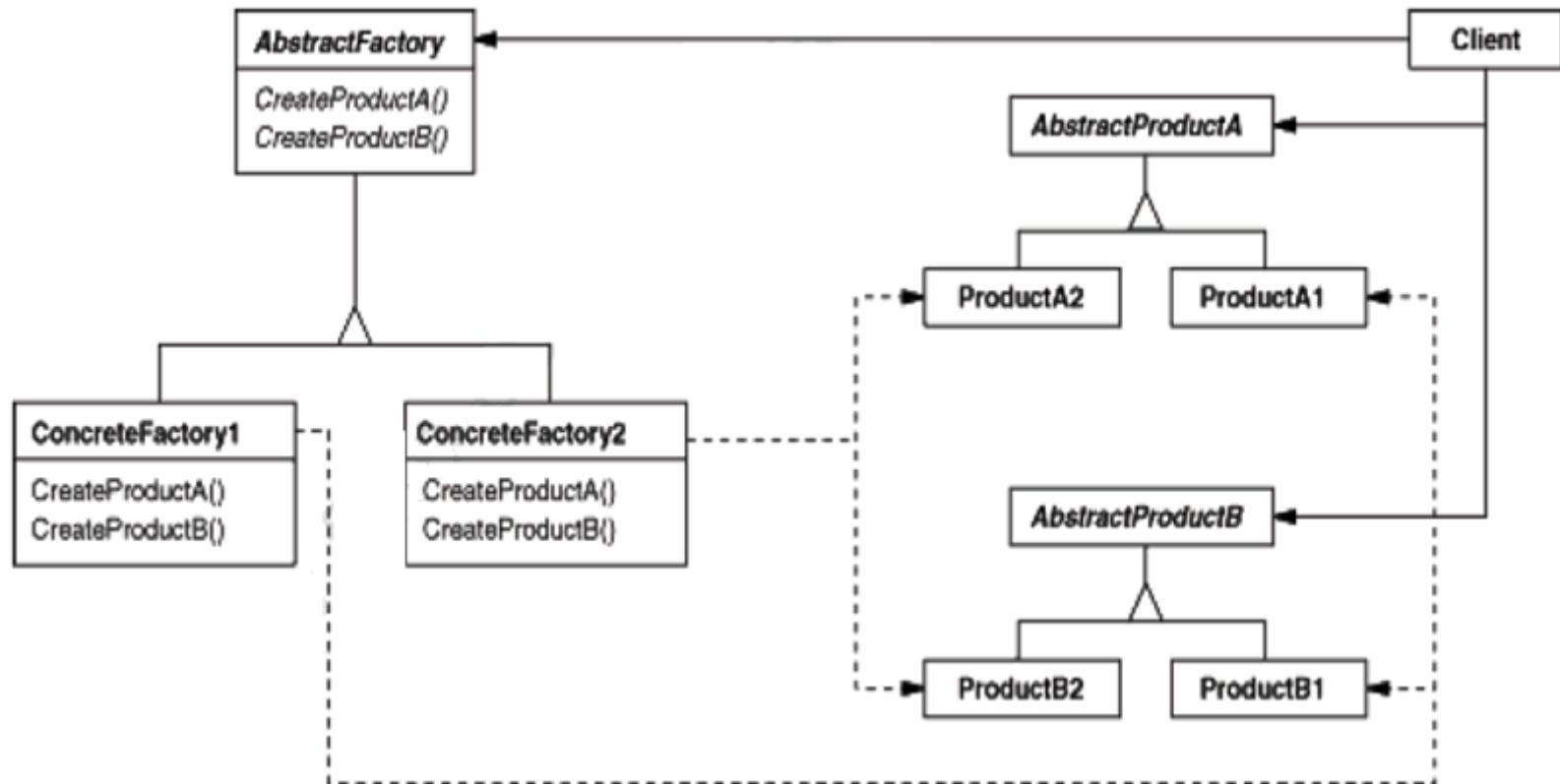
A aplicação deste padrão traz como benefício principal o isolamento de classes concretas.

Se for necessário trocar uma família inteira de produtos, esse processo se torna menos impactante nas demais partes do sistema, pois as classes-produto ficam isoladas e não expõem sua implementação a classes clientes, diminuindo o acoplamento.

Participantes

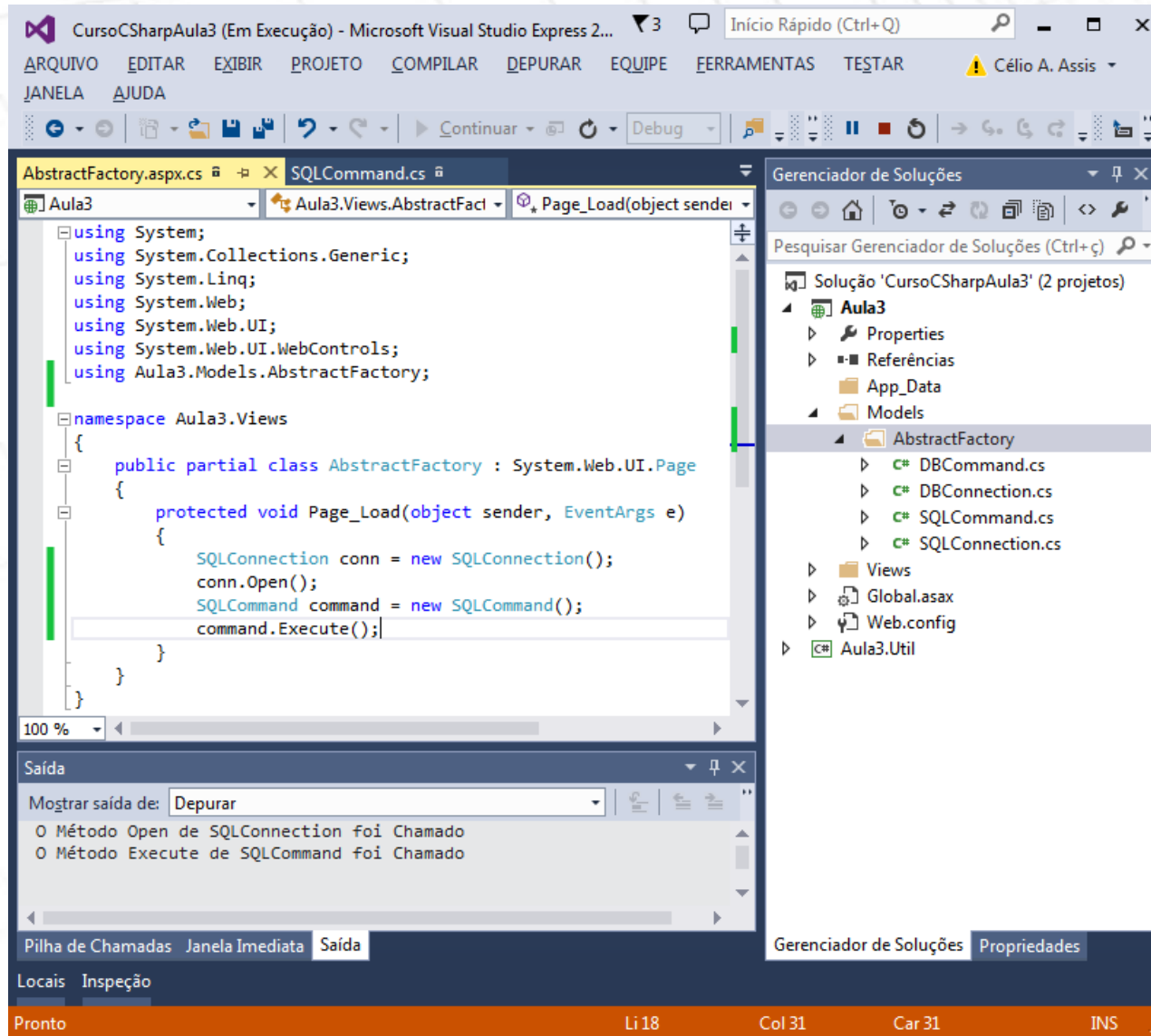
- *Abstract Factory* – declara uma interface para operações que criam objetos-produtos abstratos;
- *Concrete Factory* – implementa as operações que criam objetos-produtos concretos;
- *Abstract Product* – declara uma interface para um tipo de objeto-produto;
- *Concrete Product* – define um objeto-pruduto a ser criado pela correspondente fábrica concreta; Implementa a interface de *Abstract Product*;
- *Cliente* – usa somente as interfaces declaradas pelas classes *Abstract Factory* e *Abastract Product*;

Diagrama de Classes

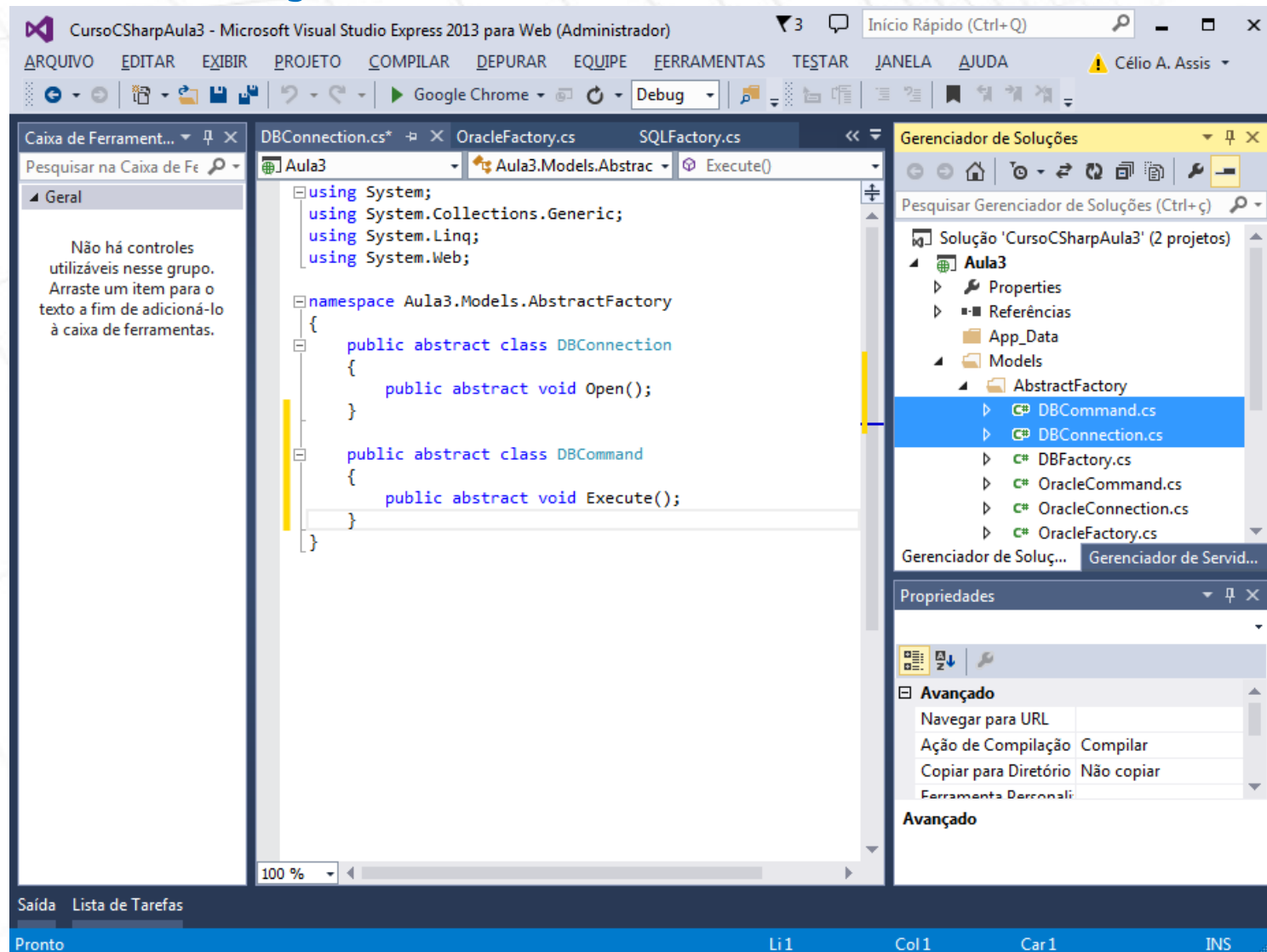


O problema a ser resolvido
pelo padrão
Abstract Factory

Aplicação Cliente



Criação da *Abstract Product*



Criação da *Concrete Product*

The screenshot displays the Microsoft Visual Studio Express 2013 for Web interface. The main editor window shows the code for `SQLConnection.cs` within the `Aula3.Models.AbstractFactory` namespace. The code defines a `SQLConnection` class that inherits from `DBConnection` and implements the `Open()` method. It also defines a `SQLCommand` class that inherits from `DBCommand` and implements the `Execute()` method. Both classes include debug output using `System.Diagnostics.Debug.WriteLine`.

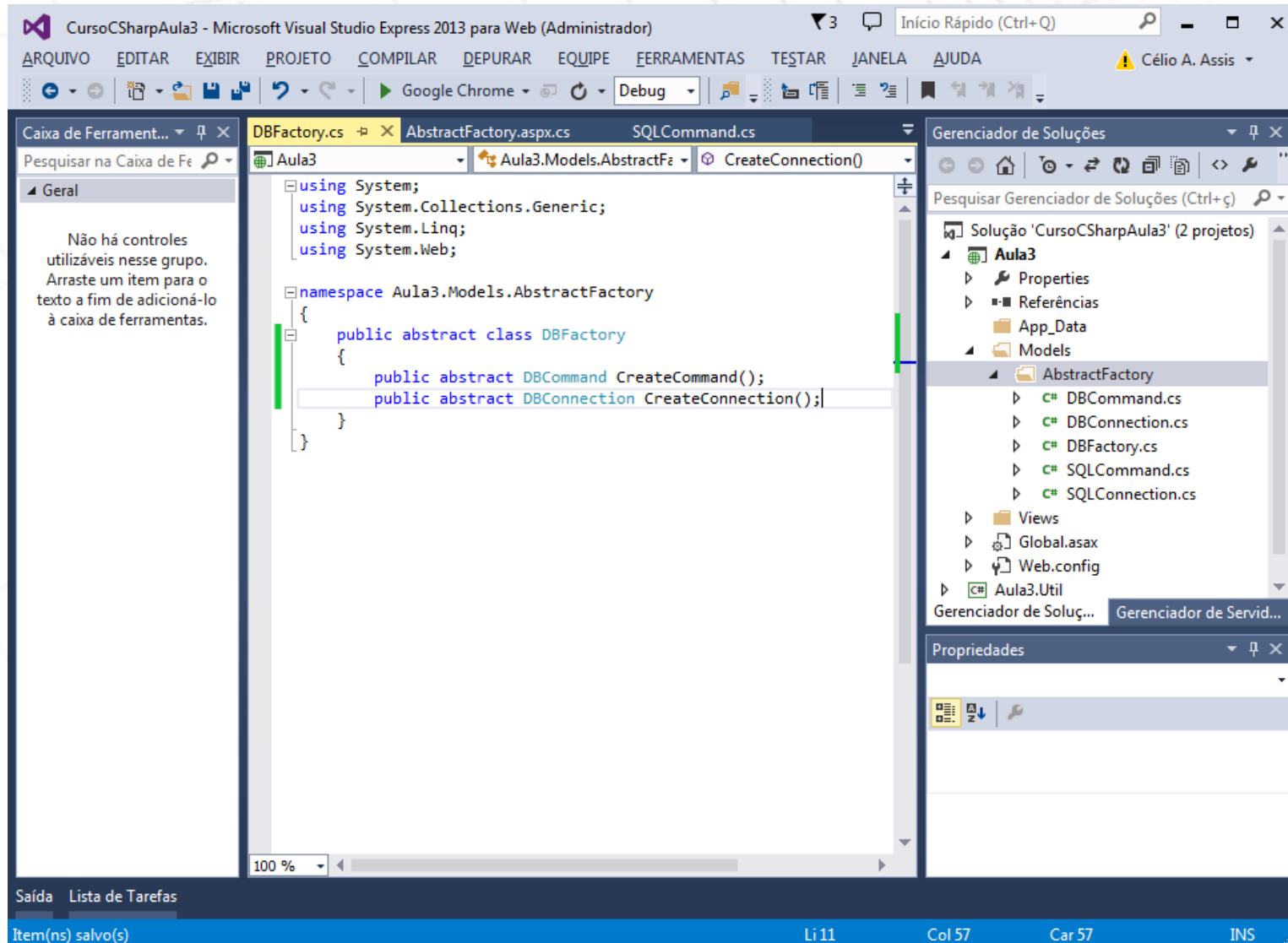
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace Aula3.Models.AbstractFactory
{
    public class SQLConnection : DBConnection
    {
        public override void Open()
        {
            System.Diagnostics.Debug.WriteLine("O Método Open de SQLConnection foi Chamado");
        }
    }

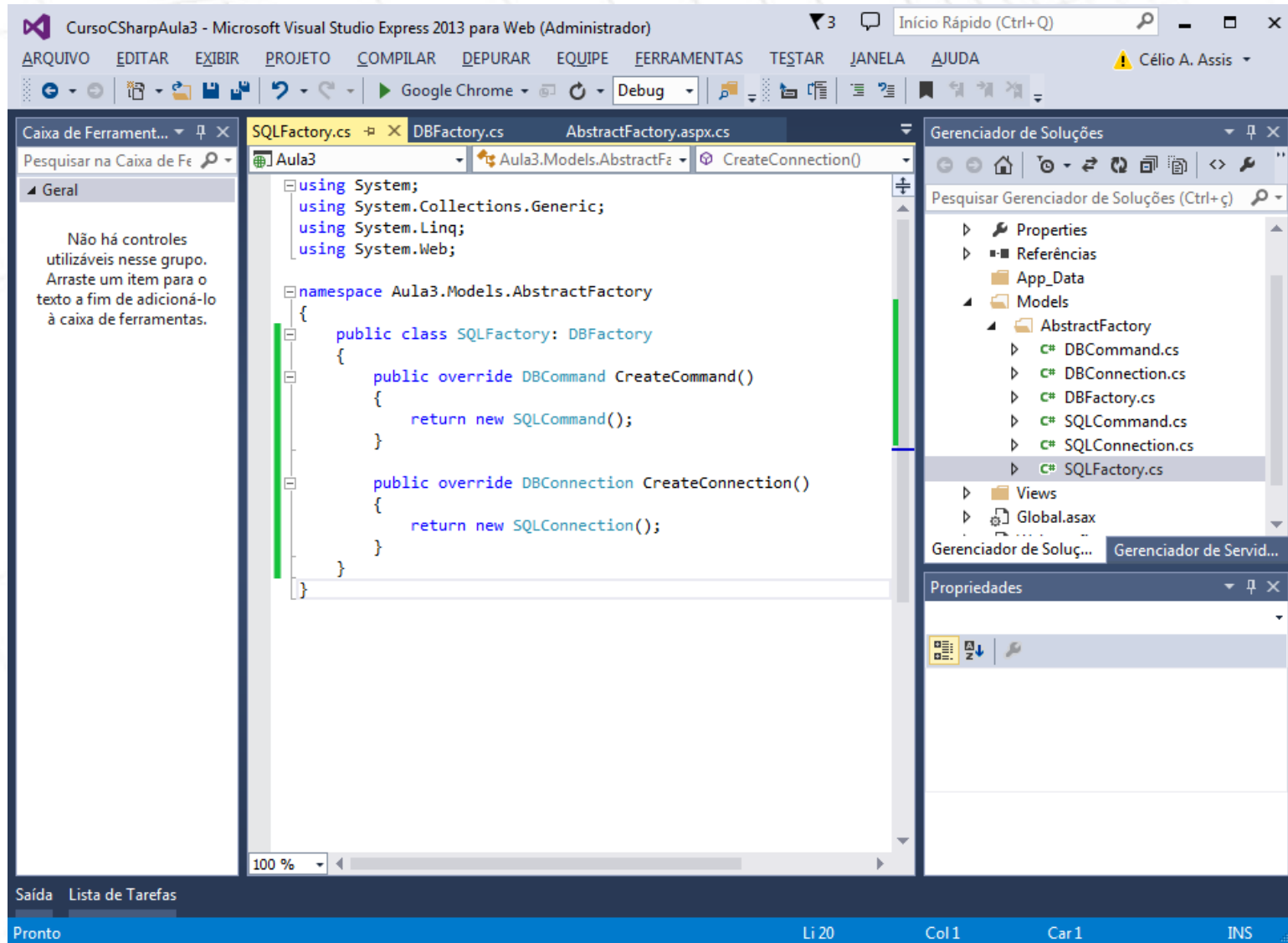
    public class SQLCommand : DBCommand
    {
        public override void Execute()
        {
            System.Diagnostics.Debug.WriteLine("O Método Execute de SQLCommand foi Chamado");
        }
    }
}
```

The Solution Explorer on the right shows the project structure, including files like `DBConnection.cs`, `DBFactory.cs`, `OracleCommand.cs`, `OracleConnection.cs`, `OracleFactory.cs`, `SQLCommand.cs`, `SQLConnection.cs`, and `SQLFactory.cs`. The Properties window at the bottom right shows the 'Avançado' (Advanced) tab with options for navigating to the URL, compiling, and copying to the directory.

Criação da *Abstract Factory*



Criação da *Concrete Factory* - SQL



Aplicação Cliente Utilizando a SQLFactory

The screenshot displays the Microsoft Visual Studio Express 2013 for Web (Administrator) interface. The main window shows the code for `AbstractFactory.aspx.cs` in the `Aula3.Views` namespace. The code implements the `Page_Load` method, which creates a `SQLFactory` instance, uses it to create a `DBConnection` and a `DBCommand`, and then executes the command.

```
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using Aula3.Models.AbstractFactory;

namespace Aula3.Views
{
    public partial class AbstractFactory : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            DBFactory factory = new SQLFactory();
            DBConnection conn = factory.CreateConnection();
            DBCommand command = factory.CreateCommand();

            conn.Open();

            command.Execute();
        }
    }
}
```

The Solution Explorer on the right shows the project structure for 'CursoCSharpAula3' (2 projetos). The 'Aula3' project contains the following files and folders:

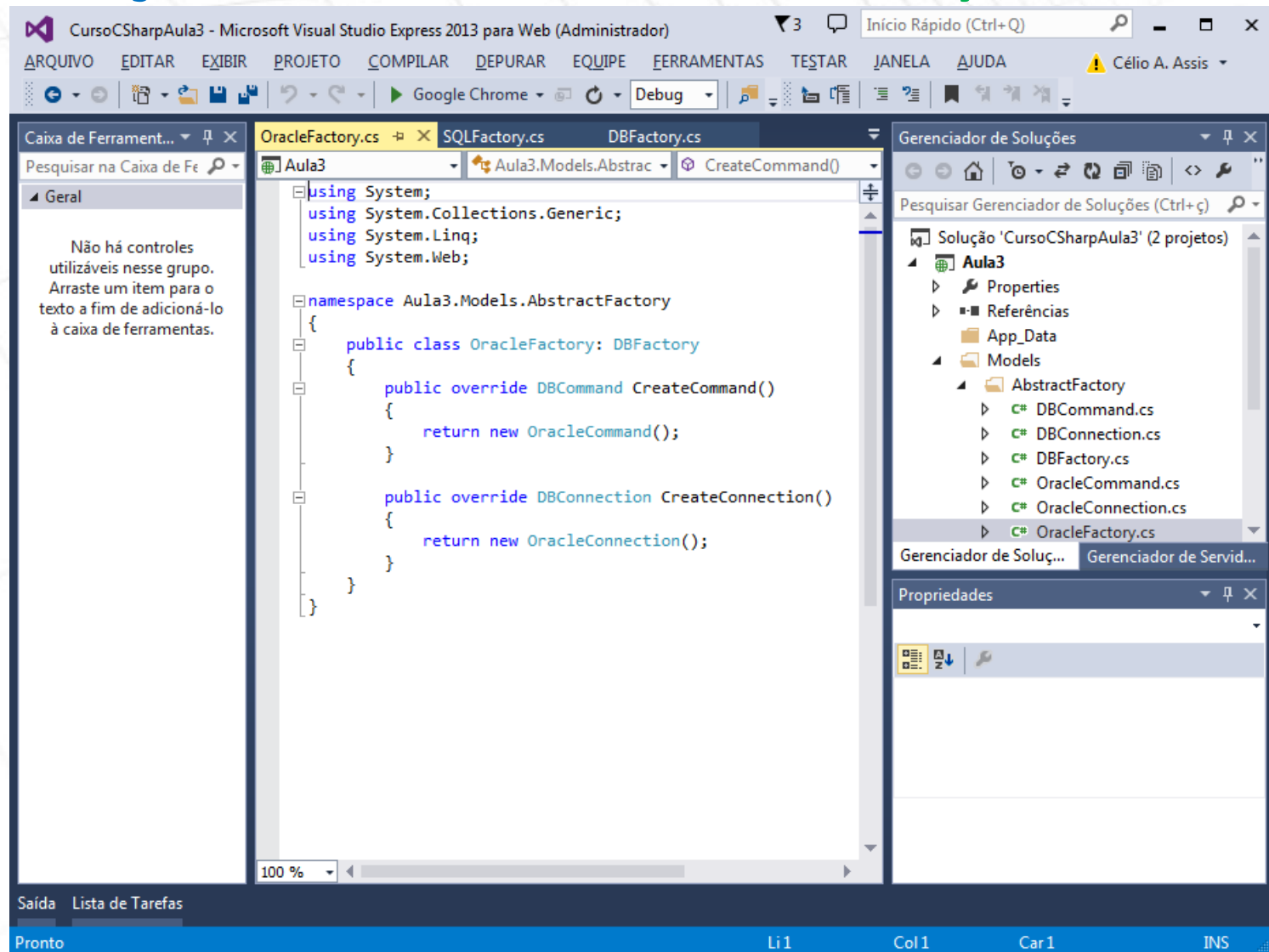
- Properties
- Referências
- App_Data
- Models
 - AbstractFactory
 - DBCommand.cs
 - DBConnection.cs
 - DBFactory.cs
 - SQLCommand.cs
 - SQLConnection.cs
 - SQLFactory.cs
- Views
- Global.asax
- Web.config
- Aula3.Util

The Output window at the bottom shows the following messages:

```
Mostrar saída de: Depurar
O Método Open de SqlConnection foi Chamado
O Método Execute de SqlCommand foi Chamado
```

The bottom status bar indicates the current position: Pronto, Li 24, Col 2, Car 2, INS.

Criação da *Concrete Factory* - Oracle



Aplicação Cliente Utilizando a OracleFactory

The screenshot displays the Microsoft Visual Studio Express 2013 IDE. The main window shows the code for `AbstractFactory.aspx.cs` in the `Aula3.Views` namespace. The code implements the `AbstractFactory` class, which inherits from `System.Web.UI.Page`. It includes a `Page_Load` method that creates an `OracleFactory` instance, uses it to create a `DBConnection` and a `DBCommand`, opens the connection, and executes the command.

```
using Aula3.Models.AbstractFactory;

namespace Aula3.Views
{
    public partial class AbstractFactory : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            DBFactory factory = new OracleFactory();
            DBConnection conn = factory.CreateConnection();
            DBCommand command = factory.CreateCommand();

            conn.Open();

            command.Execute();
        }
    }
}
```

The Solution Explorer on the right shows the project structure for 'CursoCSharpAula3' (2 projetos). The 'Aula3' project contains the following files and folders:

- Properties
- Referências
- App_Data
- Models
 - AbstractFactory
 - DBCommand.cs
 - DBConnection.cs
 - DBFactory.cs
 - OracleCommand.cs
 - OracleConnection.cs
 - OracleFactory.cs
 - SQLCommand.cs
 - SQLConnection.cs
 - SQLFactory.cs
- Views
- Global.asax
- Web.config
- Aula3.Util

The Output window at the bottom shows the debug output for the `Page_Load` method:

```
Mostrar saída de: Depurar
O Método Open de OracleConnection foi Chamado
O Método Execute de OracleCommand foi Chamado
```

The status bar at the bottom indicates the current line is 24, column 2, and the cursor is at the end of the line.

1

Utilizar os códigos apresentados nos slides para testar o padrão Abstract Factory.

MVP

O padrão MVP

- MVP – Model – View – Presenter, padrão arquitetural derivado do MVC – Model – View – Controller
- Utilizado em aplicações de User Interface, como WebForms e Windows Forms
- A camada Presenter assume o papel do “Controller” do padrão MVC
- Toda a Lógica de Apresentação é de responsabilidade da camada Presenter

Participantes

- *Model* – Objetos de negócio, definem os dados a serem exibidos na user interface (UI)
- *View* – Interface que exibe dados (Model) e roteia comandos do usuário (eventos) para a camada presenter
- *Presenter* – Meio de campo entre Model e View, obtém dados de um repositório e prepara para a View

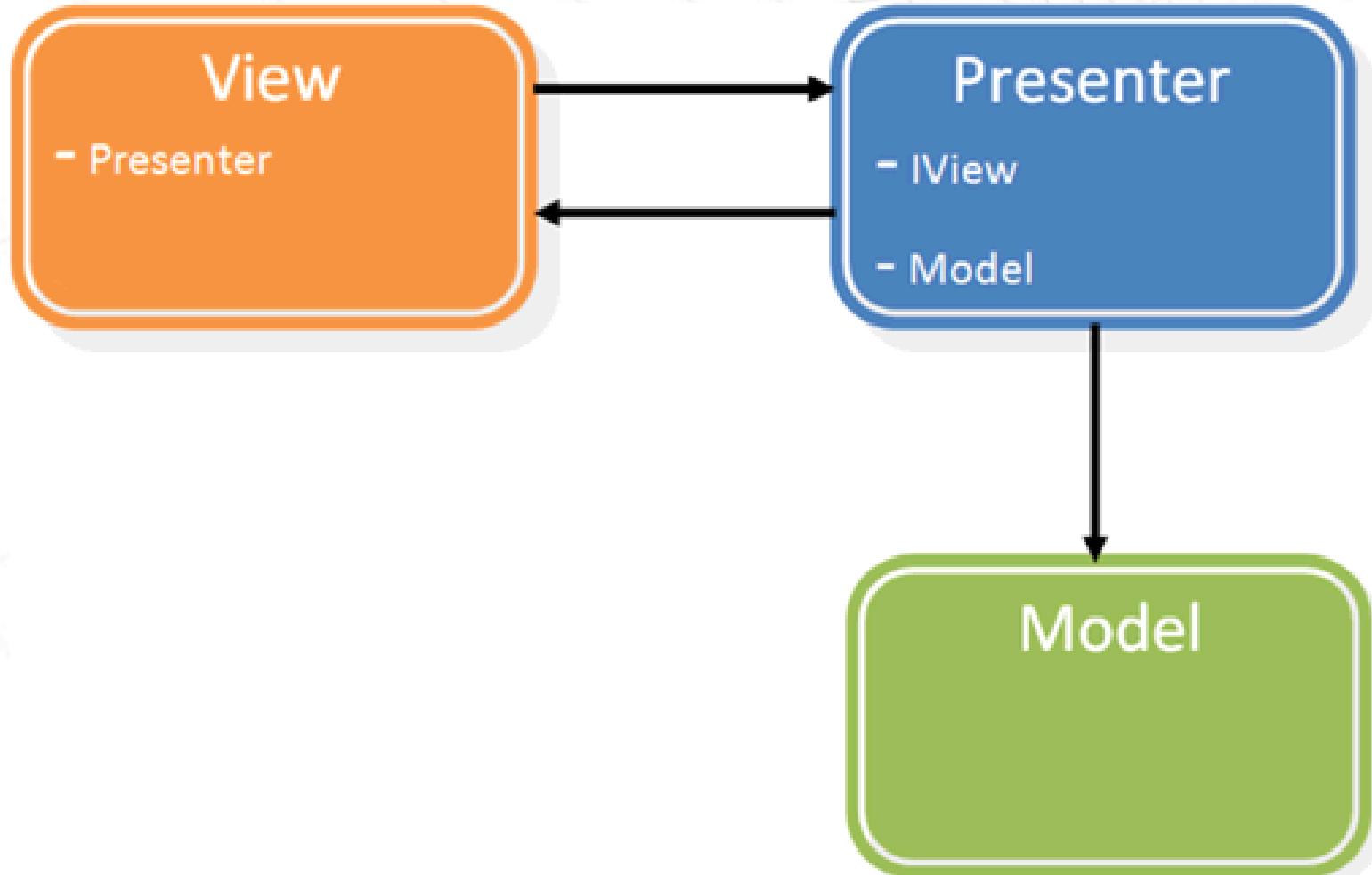
Benefícios

- Testes unitários em praticamente todas as camadas como no MVC
- Separação de responsabilidades
- Reduz código de lógica de apresentação da User Interface, por exemplo, conde-behind do formulário ASPX
- Ideal para migrar aplicações WebForms existentes para um modelo com as mesmas vantagens do MVC, sem abandonar todo o código existente.

Benefícios

- Desacoplamento das Camadas
- Reutilização das camadas e código
- Mais fácil de manter e evoluir, principalmente para sistemas de média e larga escala
- Compartilhar código entre paginas que contenham o mesmo comportamento já que o código não fica no code-behind do ASPX e sim no Presenter

O padrão MVP



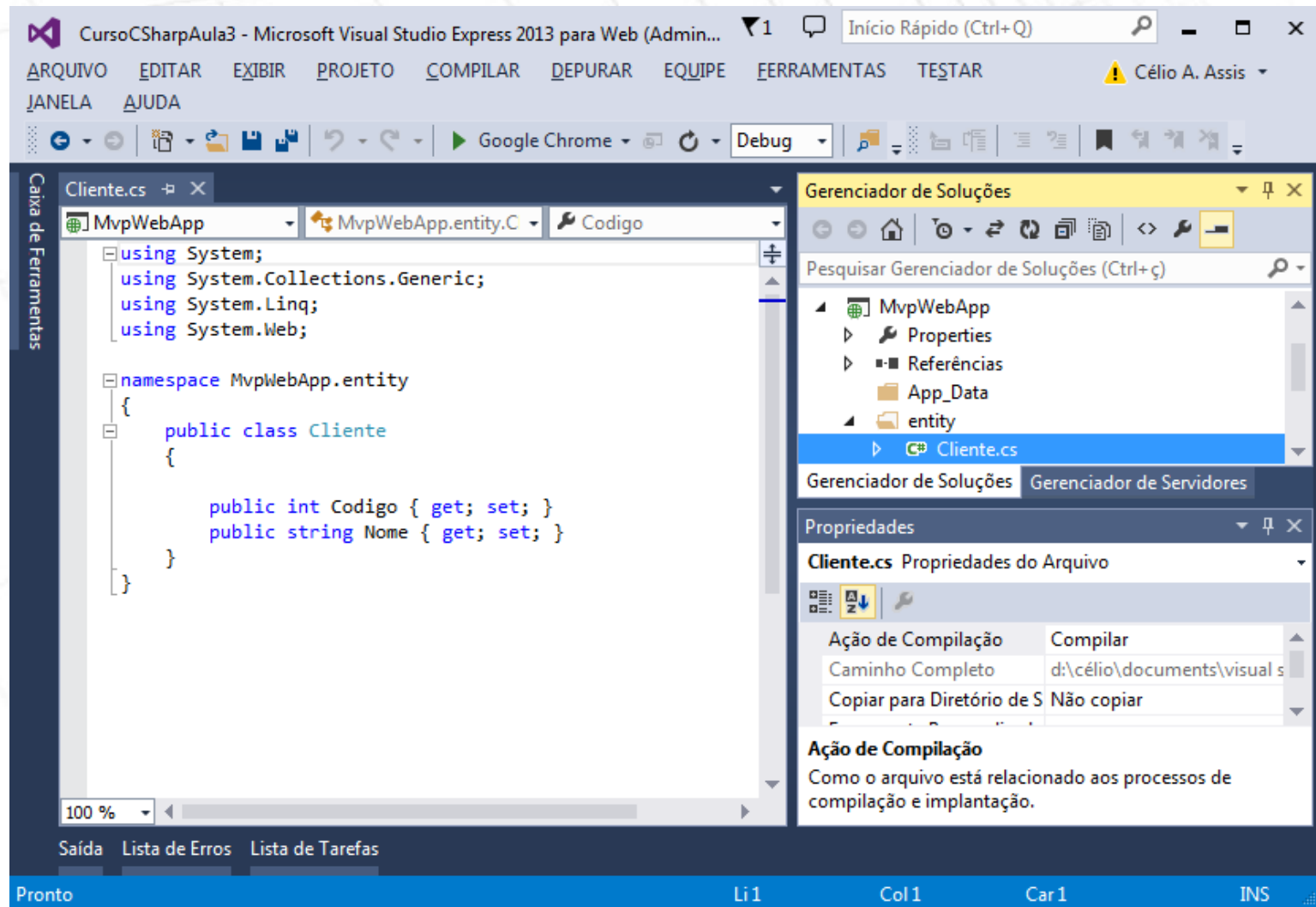
REPOSITORY

O padrão REPOSITORY

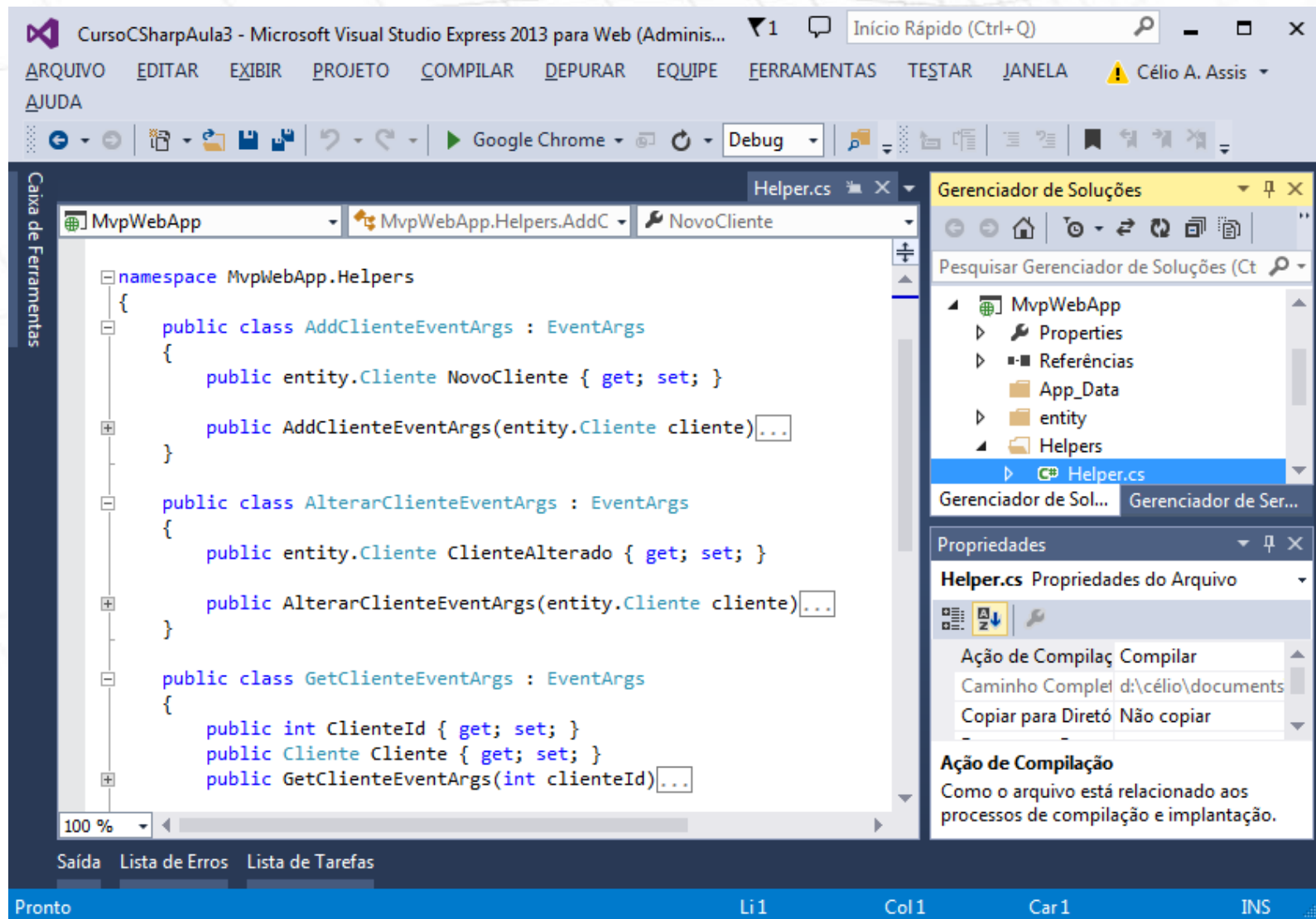
- Define e implementa uma interface para operações realizadas em um repositório (EF por exemplo)
- Cria uma abstração entre a entidade e o framework de persistência
- Permite variar “n” implementações da mesma interface do repositório, a fim de abstrair Presenter de onde os dados são persistidos realmente
- Persistence Ignorance (princípio do DDD)
- Mock Objects

Implementação básica dos padrões apresentados

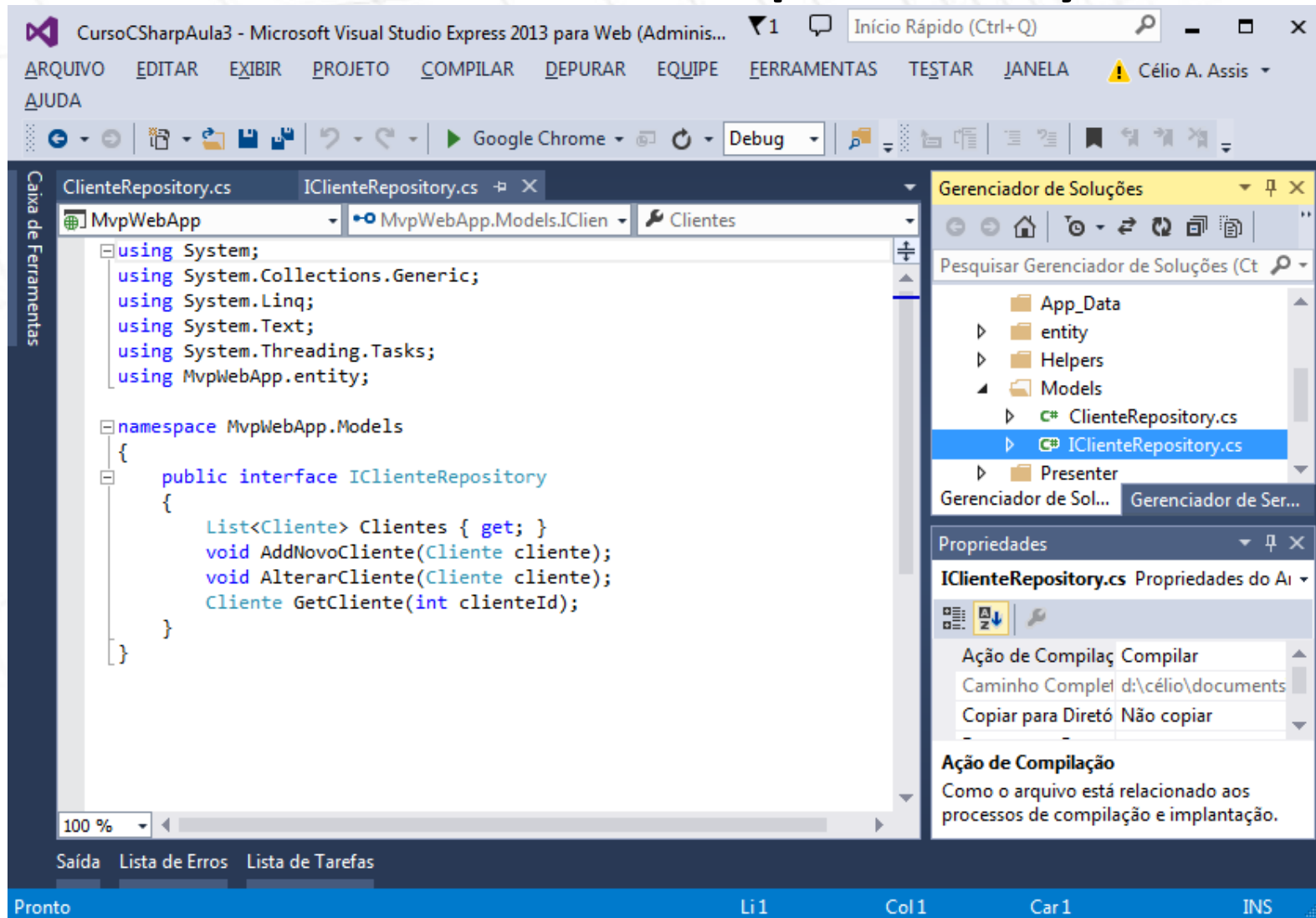
Classes POCO



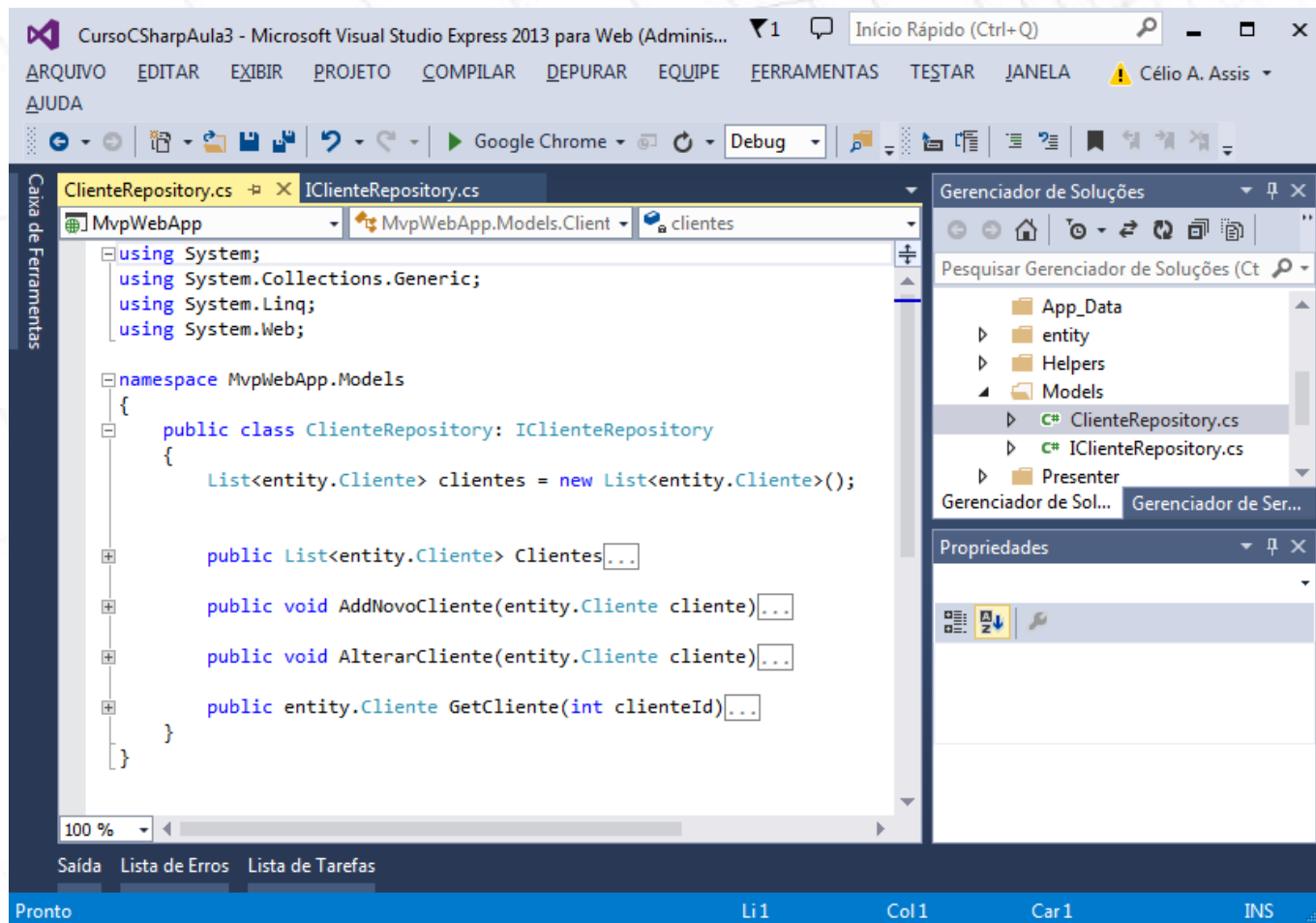
Classes de Eventos



Interface Repository



Classe Repository



Classe Presenter

The screenshot displays the Microsoft Visual Studio Express 2013 interface. The main window shows the code for `ClientePresenter.cs` within the `MvpWebApp` project. The code implements the `IClientePresenter` interface, which manages the interaction between the `IClienteView` and the `IClienteRepository`.

```
using MvpWebApp.entity;
using MvpWebApp.Views;

namespace MvpWebApp.Presenter
{
    public class ClientePresenter
    {
        IClienteView view;
        IClienteRepository repositorio;

        public ClientePresenter(IClienteView view, IClienteRepository repositorio)
        {
            this.view = view;
            this.repositorio = repositorio;
        }

        private void AttachEventsToView()
        {
            view.LoadClientes += OnLoadingClientes;
            view.AddCliente += OnAddCliente;
            view.AlterarCliente += OnAlterarCliente;
            view.GetCliente += OnGetCliente;
        }

        private void OnGetCliente(object sender, GetClienteEventArgs e)
        {
            // Implementation
        }

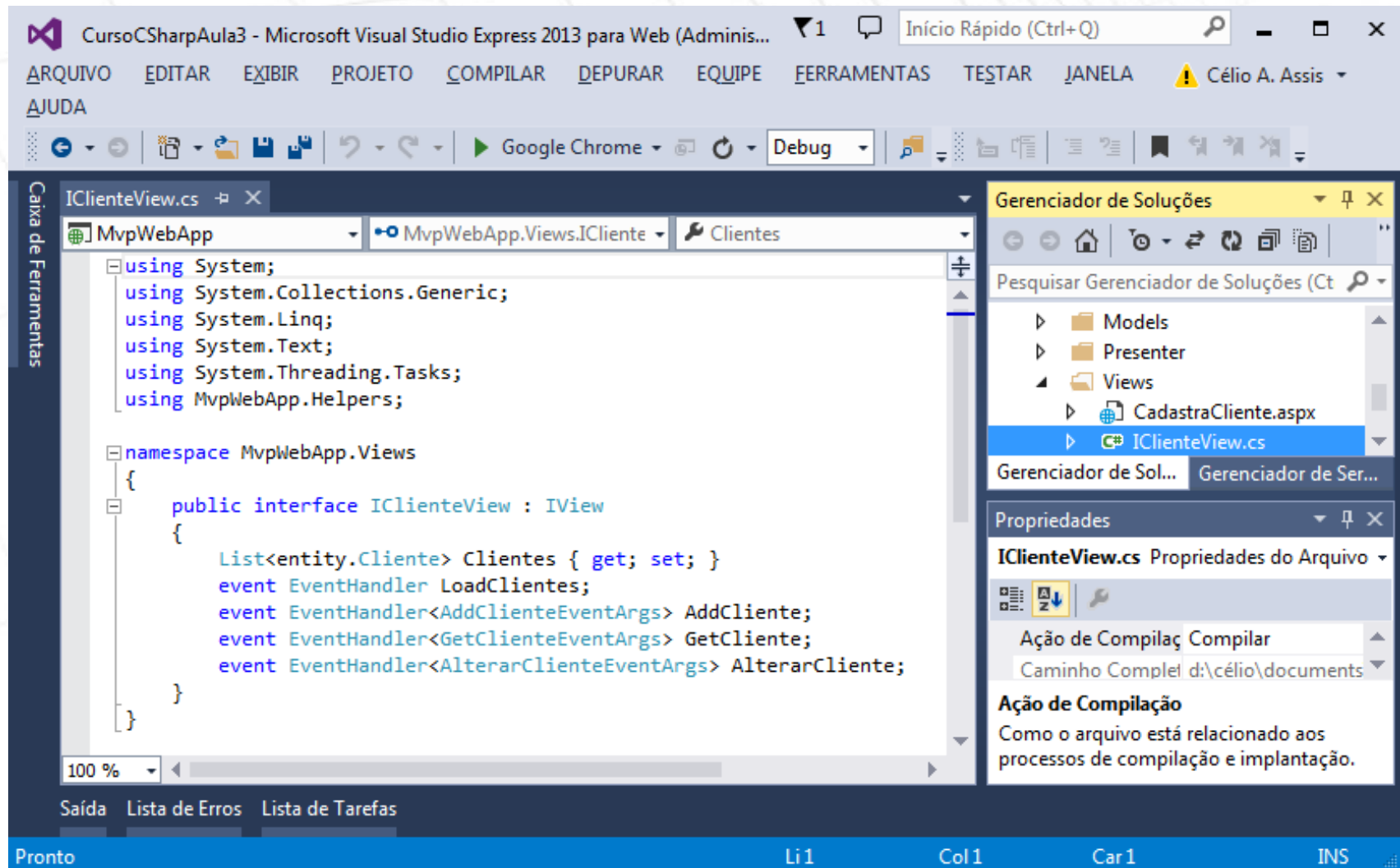
        private void OnAlterarCliente(object sender, AlterarClienteEventArgs e)
        {
            // Implementation
        }

        private void OnAddCliente(object sender, AddClienteEventArgs e)
        {
            // Implementation
        }
    }
}
```

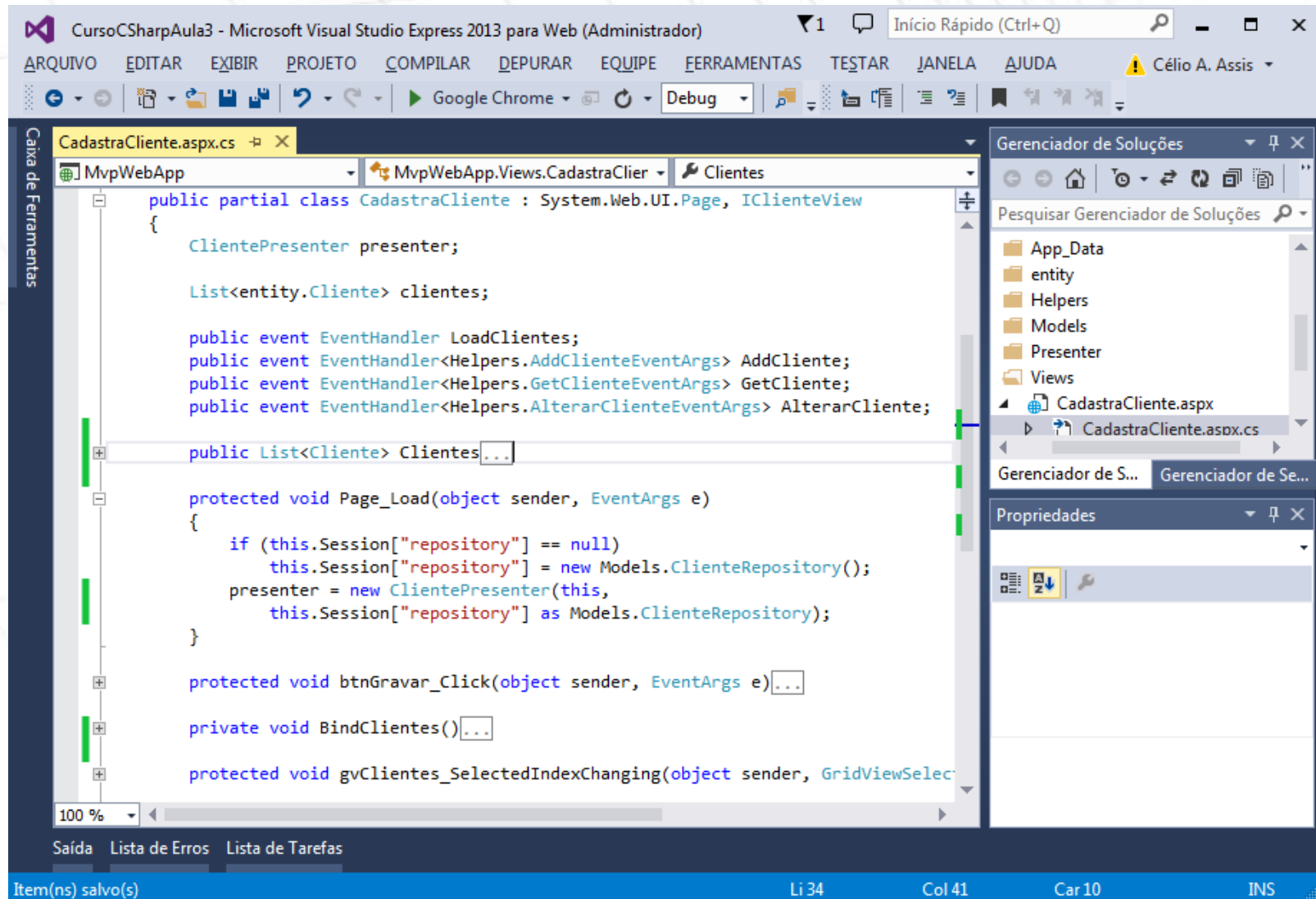
The right-hand side of the IDE features the **Gerenciador de Soluções** (Solution Explorer) and the **Propriedades** (Properties) window. The **Gerenciador de Soluções** shows the project structure, including folders like `App_Data`, `entity`, `Helpers`, `Models`, and `Presenter`. The **Propriedades** window shows the properties of the `ClientePresenter.cs` file, including the **Ação de Compilação** (Compile Action) and the **Nome da Ferramenta** (Tool Name).



Interface View



Classe View



Implementar uma aplicação de cadastro de Clientes e Fornecedores utilizando o padrão MVP e Repository.

-A Interface Repository para Cliente e Fornecedor tem que conter o seguinte contrato:

- > Método AdicionarCliente(Cliente)
- > Método ExcluirCliente(Cliente)
- > Método AlterarCliente(Cliente)
- > Método BuscarCliente(Codigo) que retorna Cliente.
- > Método ListarTodosClientes() que retorna uma Lista de Cliente

Obs.: O Nome das Interfaces devem ser as seguintes: IClienteRepository e IFornecedorRepository.

-A Interface de View deve ser a seguir:

```
List<entity.Cliente> Clientes { get; set; }  
event EventHandler ListaClientes;  
event EventHandler<AdicionarClienteEventArgs> AdicionarCliente;  
event EventHandler<ExcluirClienteEventArgs> ExcluirCliente;  
event EventHandler<BuscarClienteEventArgs> BuscarCliente;  
event EventHandler<AlterarClienteEventArgs> AlterarCliente;
```

Dados da Classe de Entidade Cliente

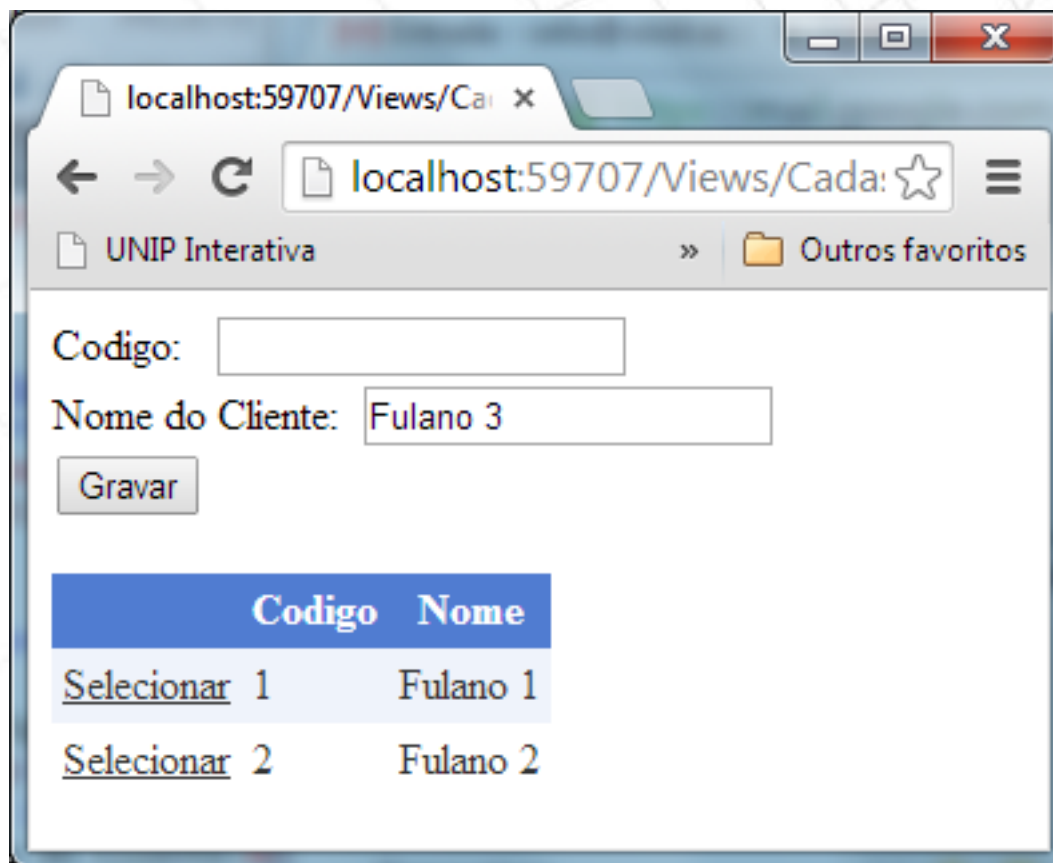
- ❖ Código
- ❖ CPF
- ❖ Nome
- ❖ SobreNome
- ❖ Endereco
- ❖ Bairro
- ❖ Cidade
- ❖ UF

Dados da Classe de Entidade Fornecedor

- ❖ Código
- ❖ CPF/CNPJ
- ❖ Nome
- ❖ TipoPessoa (Física ou Jurídica) – Utilizar Enuns
- ❖ Endereço
- ❖ Bairro
- ❖ Cidade
- ❖ UF

WebForms (View)

Os Formulários de cadastros deverão conter os campos de cadastro acima com o botão de gravar logo abaixo a lista dos itens cadastrados conforme imagem abaixo:



The screenshot shows a web browser window with the address bar displaying 'localhost:59707/Views/Cada:'. The page contains a registration form with two input fields: 'Codigo:' and 'Nome do Cliente:'. The 'Nome do Cliente' field contains the text 'Fulano 3'. Below the form is a 'Gravar' button. Underneath the button is a table with two columns: 'Codigo' and 'Nome'. The table contains two rows of data, each with a 'Selecionar' link, a numeric code, and a name.

	Codigo	Nome
Selecionar	1	Fulano 1
Selecionar	2	Fulano 2