

Classes Entidades

Programação OO

Mauricio Roberto
Gonzatto
mauricio.gonzatto@uno
esc.edu.br

Classes Identificadas

Customer

- Name
- Email address
- Home address
- Work address
- Validate()
- Retrieve()
- Save()

Product

- Product name
- Description
- Current price
- Validate()
- Retrieve()
- Save()

Order

- Customer
- Order date
- Shipping address
- Order items
- Validate()
- Retrieve()
- Save()

Order Item

- Product
- Quantity
- Purchase price
- Validate()
- Retrieve()
- Save()

Organizando a Estrutura da Aplicação em CAMADAS

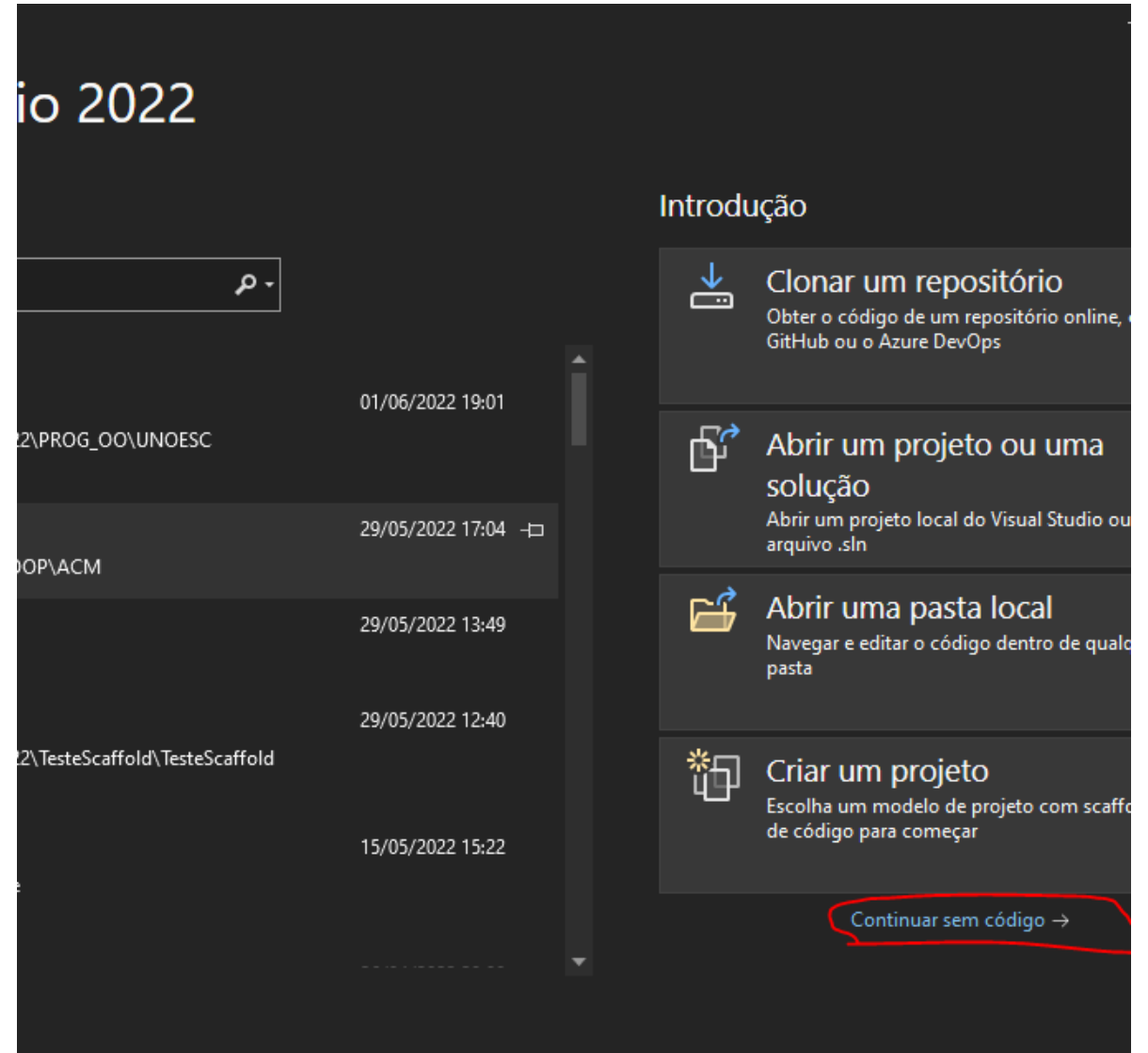
- Camada de Interface do Usuário
- Camada de Lógica de Negócios
- Camada de Acesso a Dados
- Camada de Código Comum

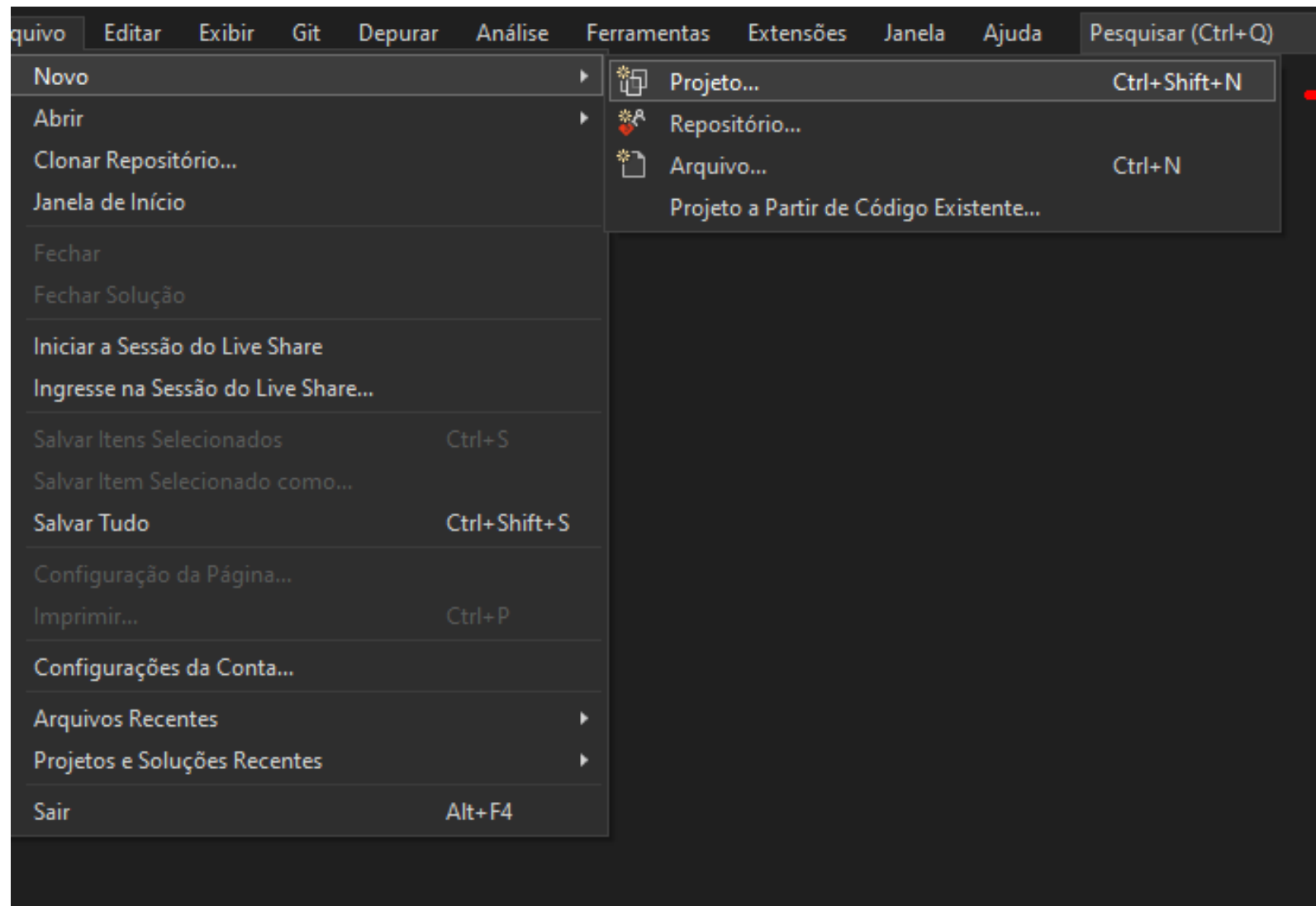


Vantagens

- Dividir a aplicação em camadas torna o trabalho mais simples e organizado.
- Separar a camada de negócios da interface de usuário, permite-nos, mais tarde, adicionar com mais tranquilidade uma interface web a nossa aplicação, por exemplo.

Organizando a Aplicação em Camadas










Novo
Projeto

Criar um novo projeto

Modelos de projeto recentes

-  Projeto de Teste de Unidade (.NET Framework) C#
-  Biblioteca de Classes C#
-  Aplicativo Web do ASP.NET Core (Model-View-Controller) C#
-  Projeto de Teste MSTest C#
-  Biblioteca de Classes (.NET Framework) C#

Pesquisar modelos (Alt+S)



Limpar

C#

Todas as plataformas

Todos os tipos de projeto

Um servidor em um aplicativo ASP.NET Core e manipula as interações com o usuário em uma conexão SignalR. Esse modelo pode ser usado para aplicativos Web com UIs (interfaces do usuário) completas e dinâmicas.

C# Linux macOS Windows Blazor Nuvem Web



API Web do ASP.NET Core

Um modelo de projeto para criar um aplicativo ASP.NET Core com um Controlador de exemplo para um serviço HTTP RESTful. Esse modelo também pode ser usado para Controladores e Exibições do ASP.NET Core MVC.

C# Linux macOS Windows Nuvem Serviço Web
WebAPI



Biblioteca de Classes

Um projeto para criar uma biblioteca de classes direcionada para o .NET ou .NET Standard

C# Android Linux macOS Windows Biblioteca



ASP.NET Core Vazio

Um modelo de projeto vazio para a criação de um aplicativo ASP.NET Core. Esse modelo não tem nenhum conteúdo.

C# Linux macOS Windows Nuvem Serviço Web

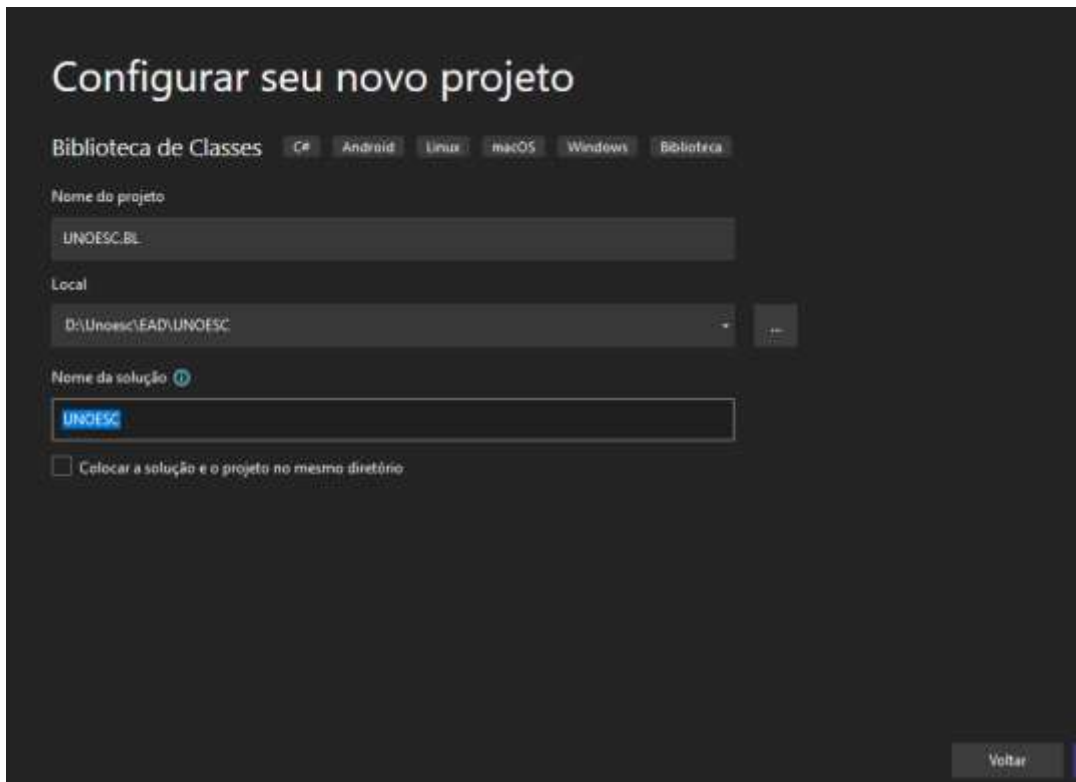


Aplicativo Web do ASP.NET Core (Model-View-Controller)

Próximo

Biblioteca de Classes

Configure o Projeto



The screenshot shows the 'Configure your new project' dialog in Visual Studio. At the top, the title is 'Configurar seu novo projeto'. Below it, there are tabs for 'Biblioteca de Classes', 'C#', 'Android', 'Linux', 'macOS', 'Windows', and 'Biblioteca'. The 'C#' tab is selected. Under 'Nome do projeto', the text 'UNOESC.BL' is entered. Under 'Local', the path 'D:\Unoesc\EAD\UNOESC' is shown. Under 'Nome da solução', the text 'UNOESC' is entered. At the bottom, there is a checkbox labeled 'Colocar a solução e o projeto no mesmo diretório' which is currently unchecked. A 'Voltar' button is visible in the bottom right corner.

- Cada camada será separada em um projeto;
- Nomeie seu projeto como UNOESC.BL;
- Escolha um diretório para armazenar o projeto;
- Defina UNOESC como nome da solução;

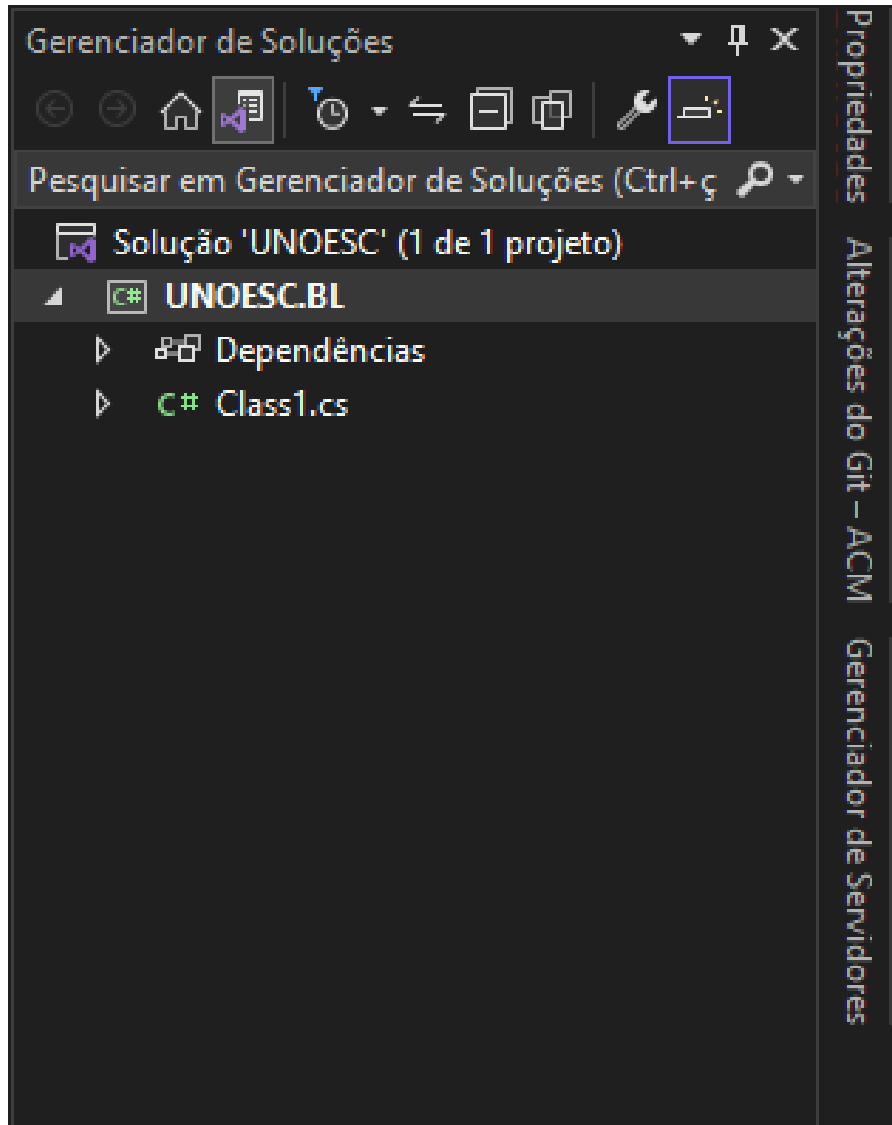
Selecione o Framework

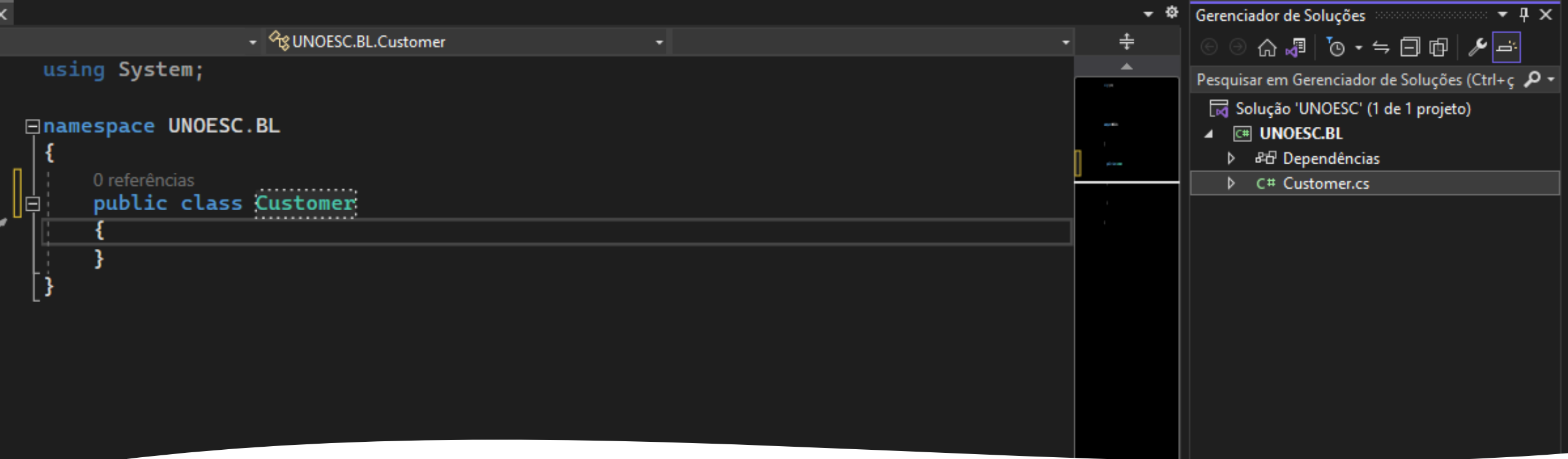


- Configure o Framework
- Caso não possua o .NET Core 3.1 instalado, acesse:
- <https://dotnet.microsoft.com/en-us/download/dotnet/3.1>

Projeto Inicial

- O Visual Studio 2022 irá criar a estrutura inicial da Solução, contendo seu projeto UNOESC.BL com uma classe Class1.cs;
- Verifique seu Gerenciador de Soluções;





A Classe Customer

- Renomeie o arquivo Class1.cs para Customer.cs através do Gerenciador de Soluções;
- O VS irá refatorar automaticamente o código, modificando a especificação da classe para o novo nome Customer;
- Confirme a alteração e salve o arquivo;

A Classe Customer

- A classe Customer deve ser definida como public, para que toda a aplicação consiga enxergá-la;
- Lembre-se que a classe deve ENCAPSULAR seus dados;
- O C# possui o modificador de acesso internal, ele especifica que um atributo da classe, por exemplo, ficará acessível apenas às classes que pertençam ao mesmo projeto.
- A especificação da classe deve ser:

A Classe Customer

```
Customer.cs x
UNOESC.BL
UNOESC.BL.Customer

0 referências
public string FullName
{
    get
    {
        string fullname = LastName;
        if (!string.IsNullOrEmpty(FirstName))
        {
            if (!string.IsNullOrEmpty(fullname))
            {
                fullname += ", ";
            }
            fullname += FirstName;
        }
        return fullname;
    }
}

0 referências
public bool Validate()
{
    var isValid = true;
    if (string.IsNullOrEmpty(LastName)) isValid = false;
    if (string.IsNullOrEmpty(EmailAddress)) isValid = false;
    return isValid;
}
```

```
namespace UNOESC.BL
{
    2 referências
    public class Customer
    {
        0 referências
        public Customer() {}
        0 referências
        public Customer(int customerId)
        {
            CustomerId = customerId;
        }

        1 referência
        public int CustomerId { get; private set; }
        1 referência
        public string EmailAddress { get; set; }
        2 referências
        public string FirstName { get; set; }
        private string _lastName;
        2 referências
        public string LastName
        {
            get { return _lastName; }
            set { _lastName = value; }
        }
    }
}
```

Criando Objetos

- Agora que definimos a primeira classe, podemos criar instâncias de objetos deste tipo:
- Customer: O Tipo da variável é a especificação do tipo do objeto
- customer: variável objeto criada na memória

```
var customer = new Customer();
```

```
Customer customer = new Customer();
```

```
var customer = new Customer();
```

Criando Objetos

- Podemos declarar uma variável utilizando a palavra reservada var:
- var define um tipo implícito de variável que é fortemente tipada de acordo com a especificação de chamada do construtor da variável;

Objetos são Tipos Referenciados

- Objetos guardam um ponteiro de memória para os dados e não os dados em si mesmos;
- Diferente dos tipos de dados por valor, que armazenam os dados diretamente;
- Observe o código, qual o valor de i1?

```
int i1;  
i1 = 42;  
  
int i2 = i1;  
i2 = 2;
```

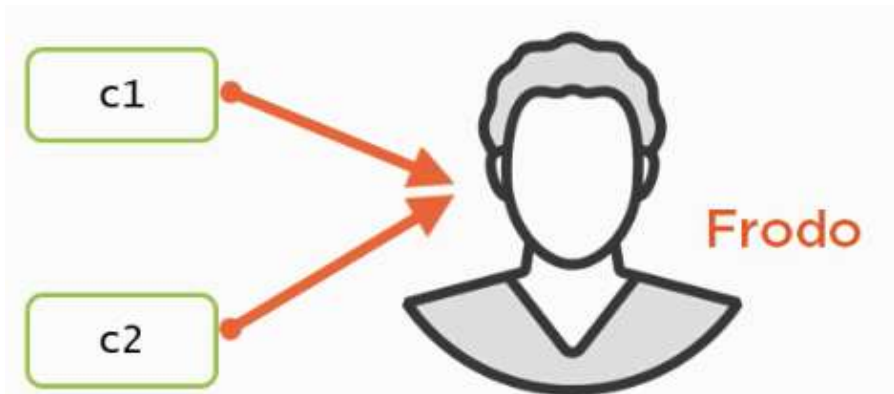
i1 42

i2 2

Objetos são Tipos Referenciados

- E agora, quando operamos com Objetos?
 - C1 atribui o nome de Bilbo;
 - C2 é criado como uma cópia de c1, um ponteiro;
 - C2 altera o nome para Frodo;
 - RESULTADO: Tanto C1 quanto C2 possuem o nome Frodo;

```
var c1 = new Customer();  
c1.FirstName = "Bilbo";  
  
var c2 = c1;  
c2.FirstName = "Frodo";
```



```
public static int InstanceCount { get; set; }
```

Modificador STATIC

- O modificador static declara um membro que pertence à classe em si mesma e não à instância do objeto.
- Útil para rastrear informações compartilhadas entre todas as instâncias de objeto daquele tipo.
- Para acessar atributos estáticos, referenciamos o tipo.propriedade:

```
Customer.InstanceCount += 1;
```

