

Teste simples/unitário

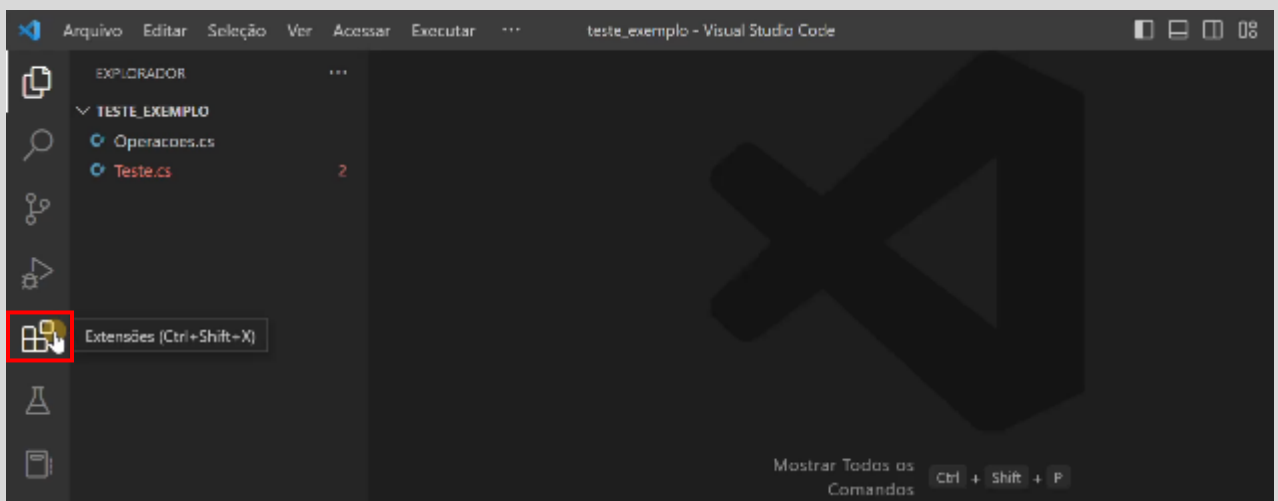
Introdução

Neste conteúdo, você aprenderá a configurar o VS Code para projetos de teste simples ou unitários em C#, usando o xUnit e o MSTeste.

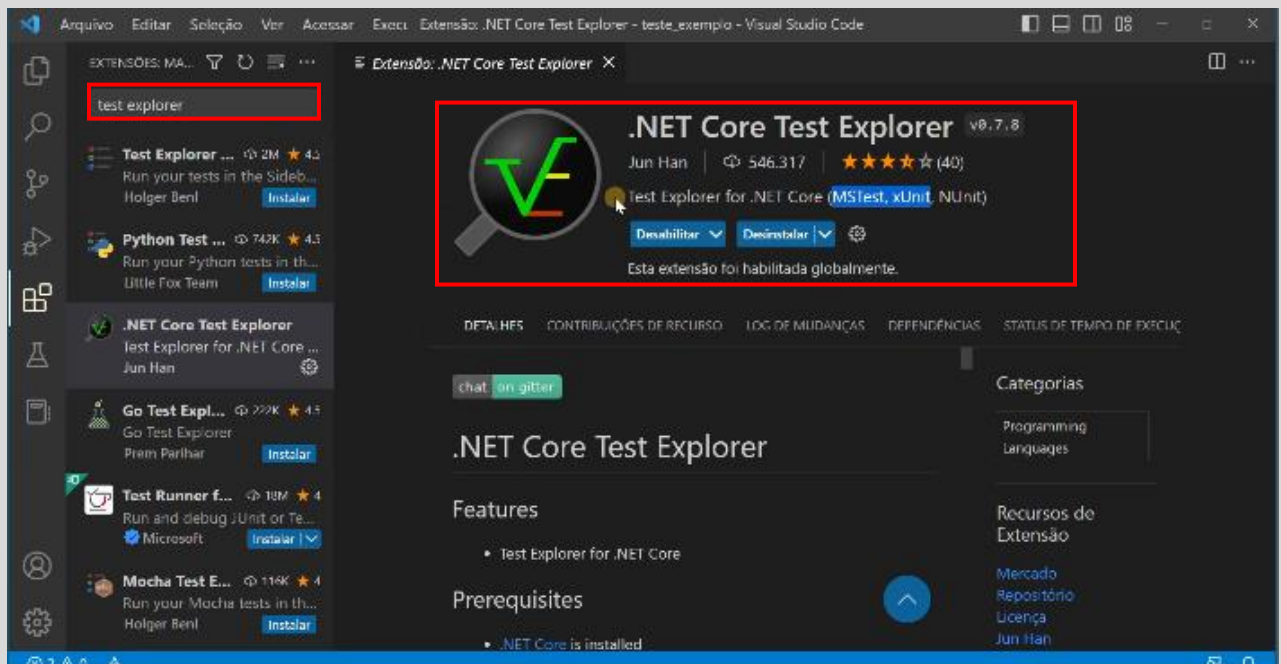
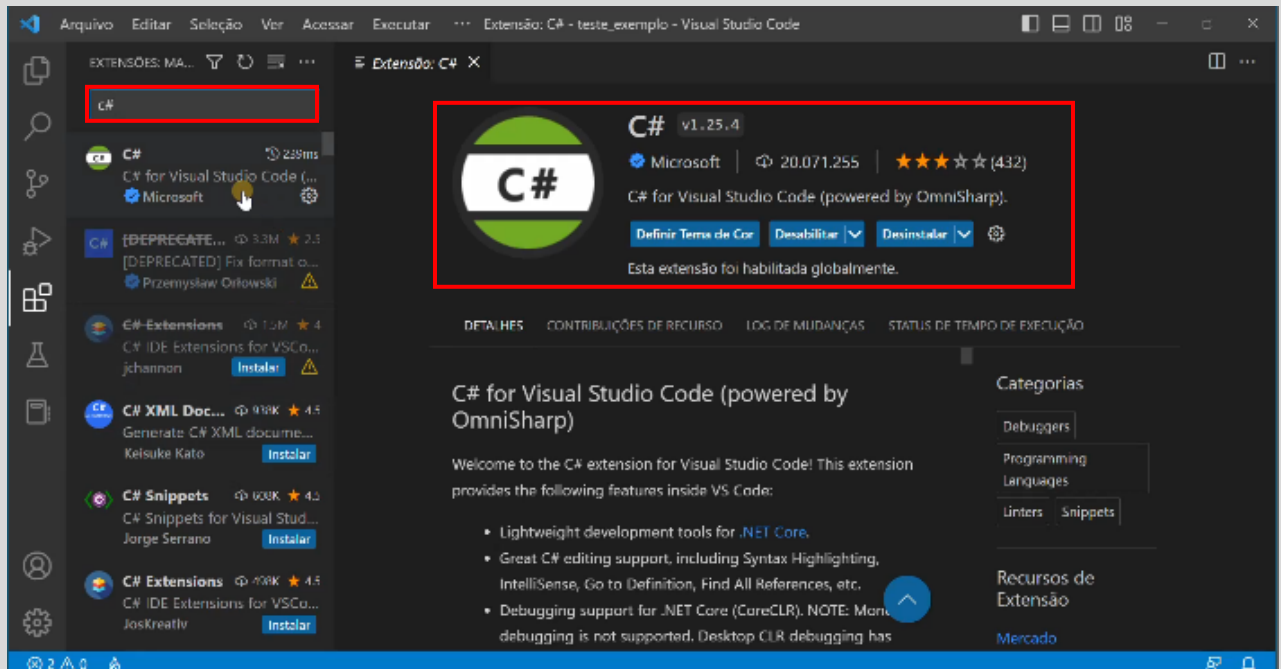
Para isso, você precisa ter instalado em sua máquina o VS Code com as extensões da linguagem C# e de teste.

Extensões

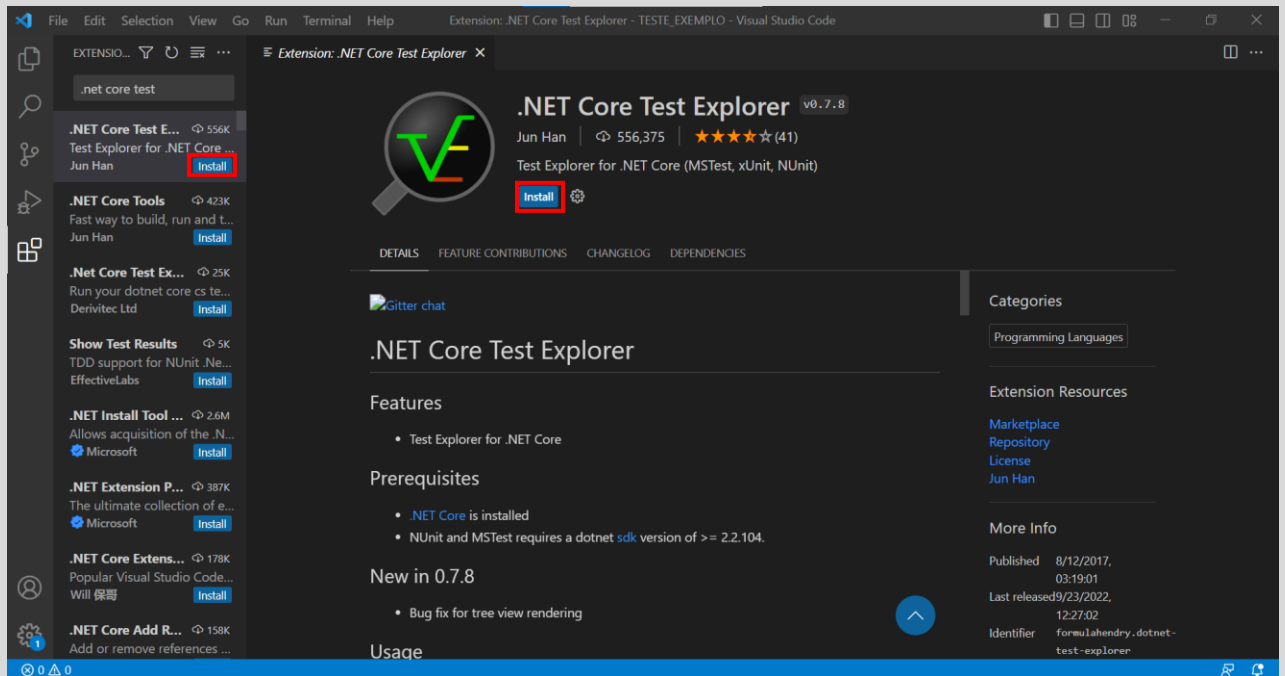
1. Para instalar as extensões, clique em “Extensões” (quinto ícone de cima para baixo no menu lateral esquerdo).



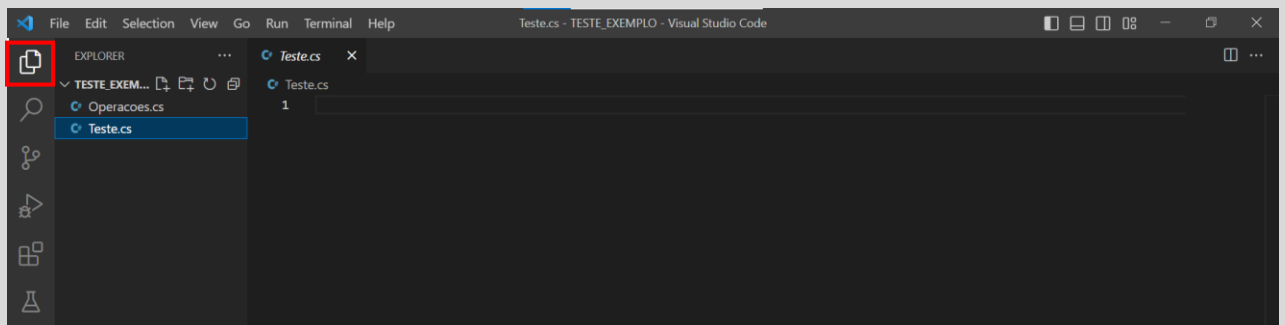
2. Na busca, digite parte do nome das extensões desejadas e aperte enter. As duas extensões necessárias são **C# for Visual Studio Code** e **.NET Core Test Explorer** e serão usadas tanto no xUnit quanto no MSTest.



3. Clique em “Instalar” para instalar a extensão necessária.

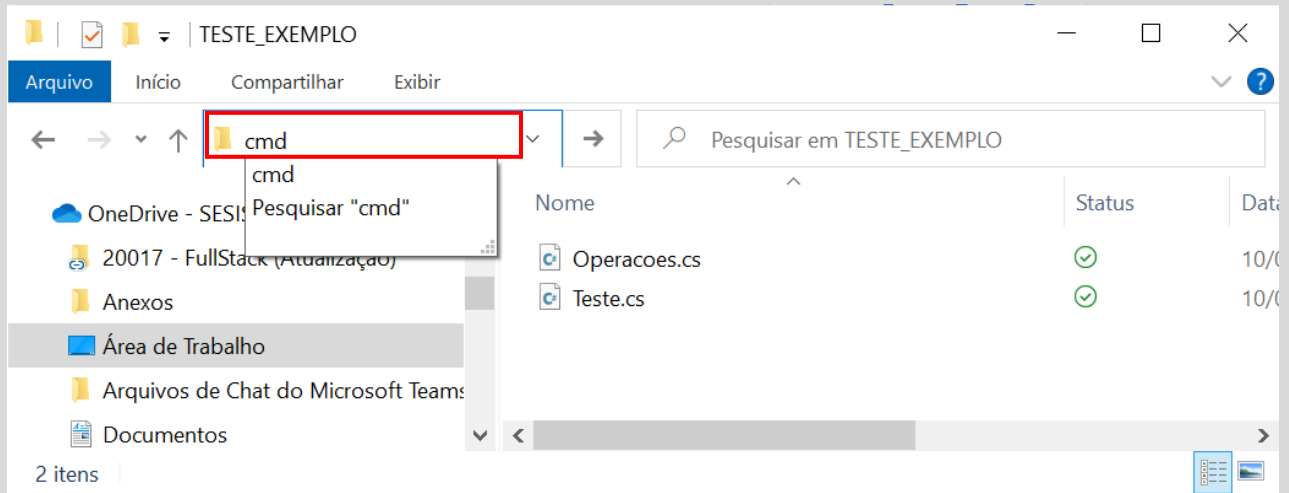


4. Clique em “Explorer” (primeiro ícone de cima para baixo no menu lateral esquerdo) para retornar à pasta do exemplo.

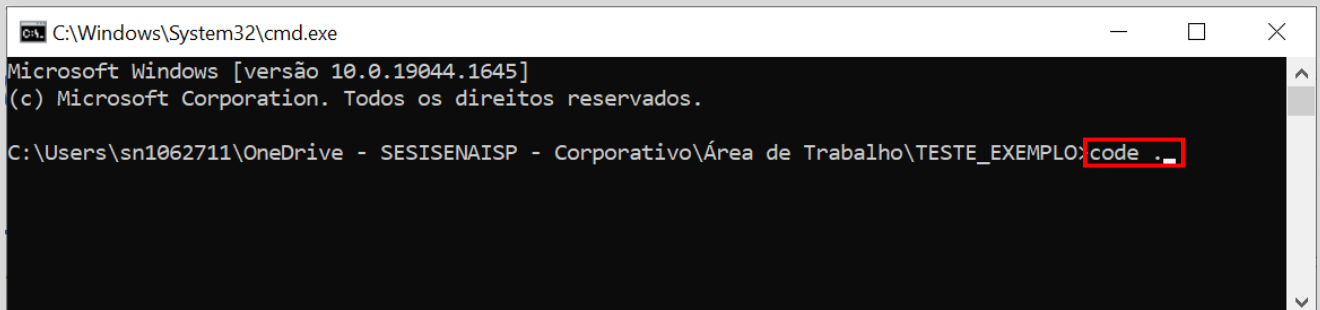


Preparação

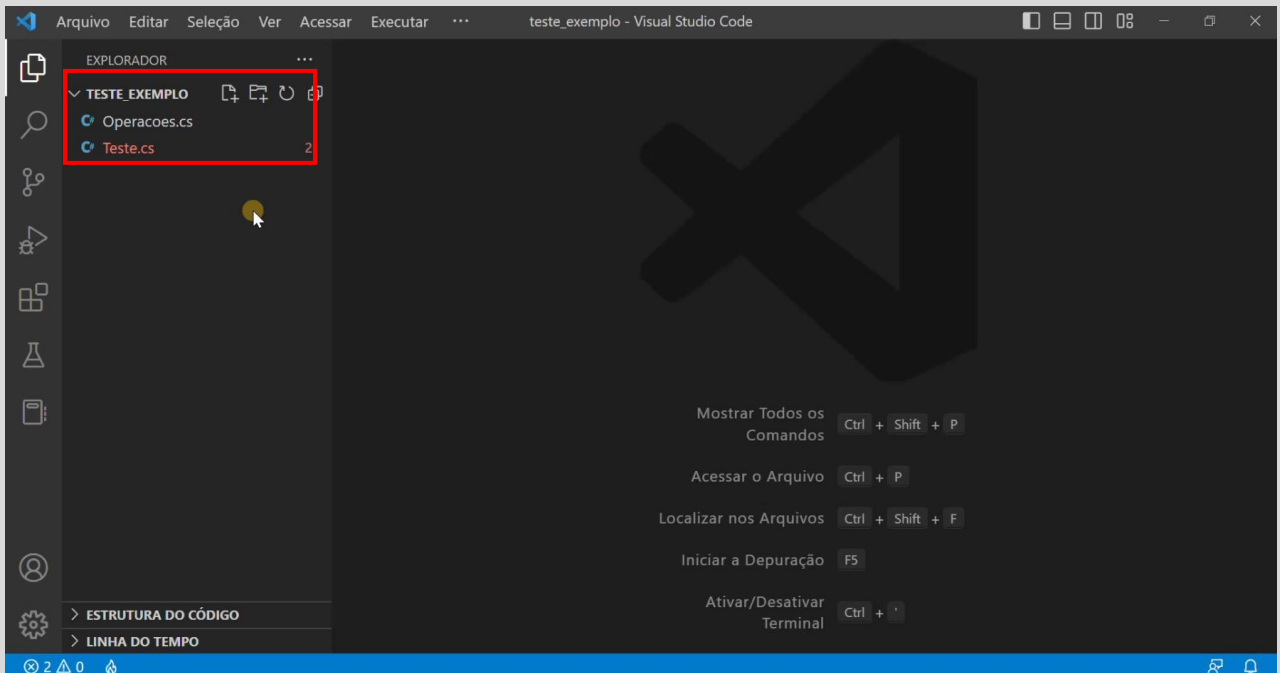
1. Com a pasta do exemplo aberta, digite **cmd** na barra de endereços e aperte **enter**.



2. No terminal, digite **code .** e aperte **enter** para abrir o VS Code.



3. Ao abrir, o VS Code mostrará a estrutura de pastas mapeada.

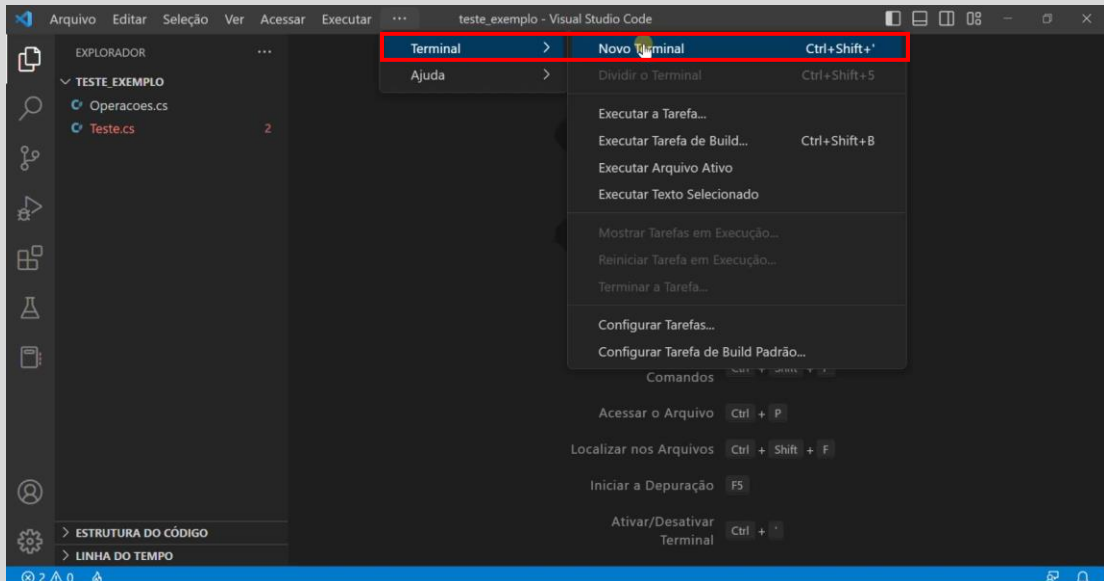


A classe **Operacoes.cs**, fornecida como arquivo de apoio, tem apenas dois métodos: um de soma e outro de multiplicação. O método **Somar** precisa de dois parâmetros do tipo **double** para serem somados: **primeiroNumero** e **segundoNumero**. O método **Multiplicar** também precisa de dois parâmetros do tipo **double** para serem multiplicados, as mesmas variáveis do método **Somar**.

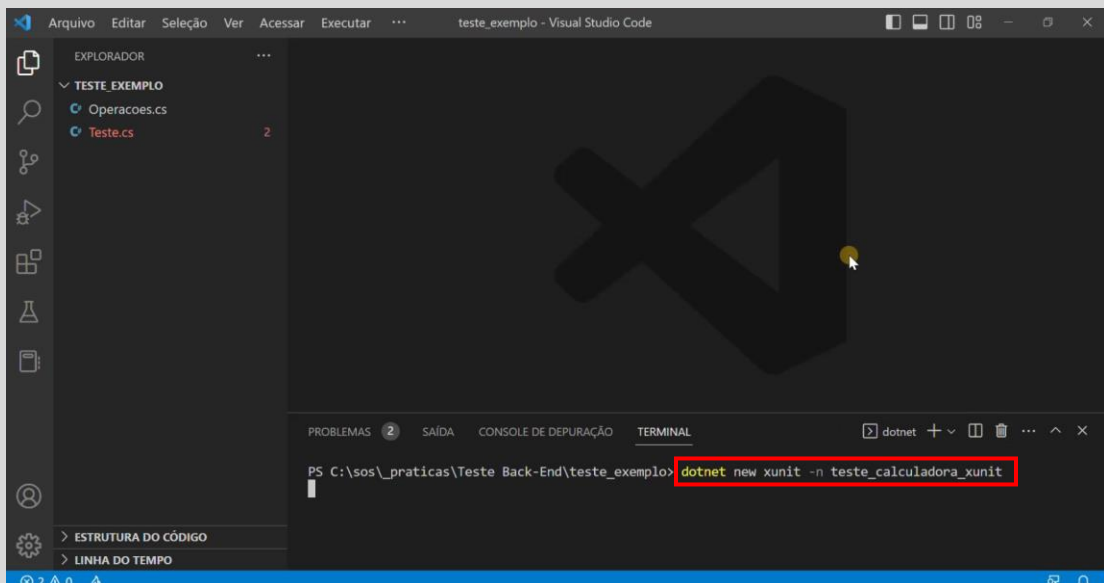
A classe **teste.cs** é a que usaremos para testar a classe **Operacoes.cs**.

Teste com xUnit

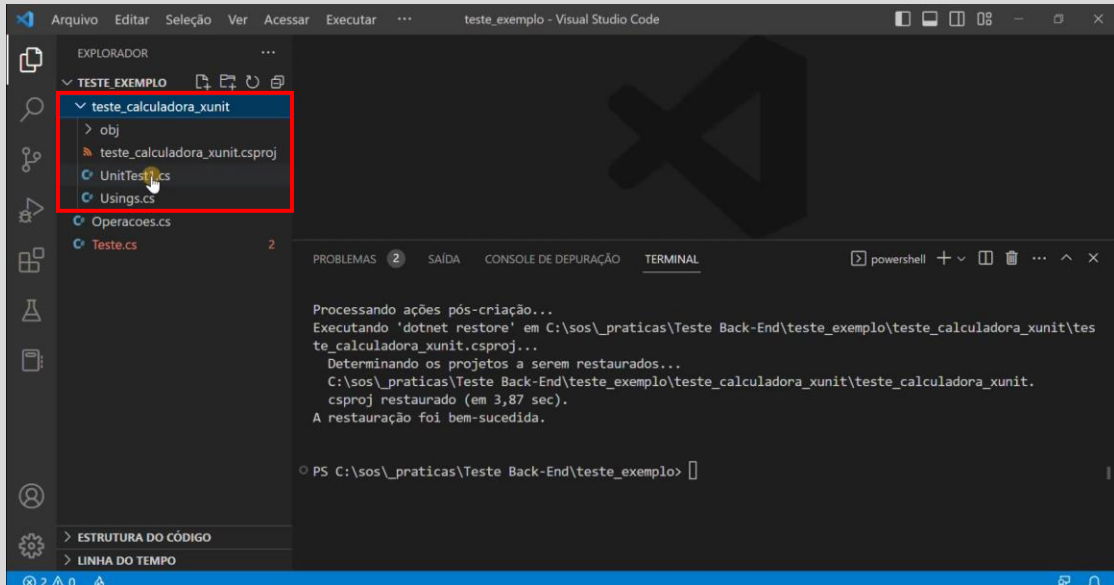
1. No menu superior do VS Code, selecione “**Terminal**” e, depois, em “**Novo Terminal**”.



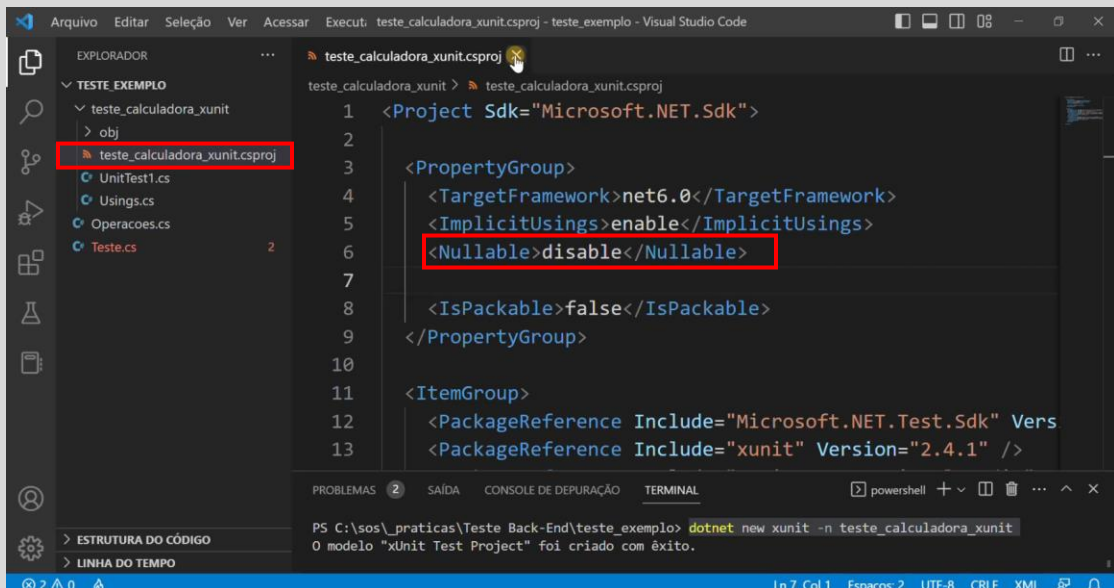
2. No terminal, digite **dotnet new xunit -n teste_calculadora_xunit** e clique em **enter** para criar uma aplicação de teste.



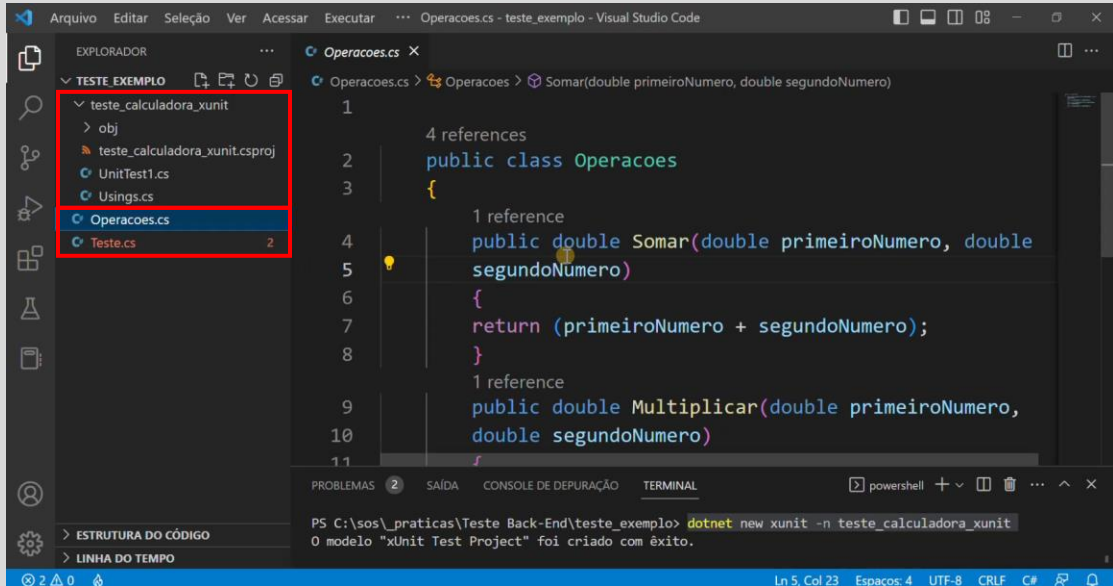
3. Note que, após a execução bem-sucedida, o explorador de arquivos na lateral esquerda mostra a pasta **teste_calculadora_xunit** criada com a estrutura principal pronta e as dependências necessárias.



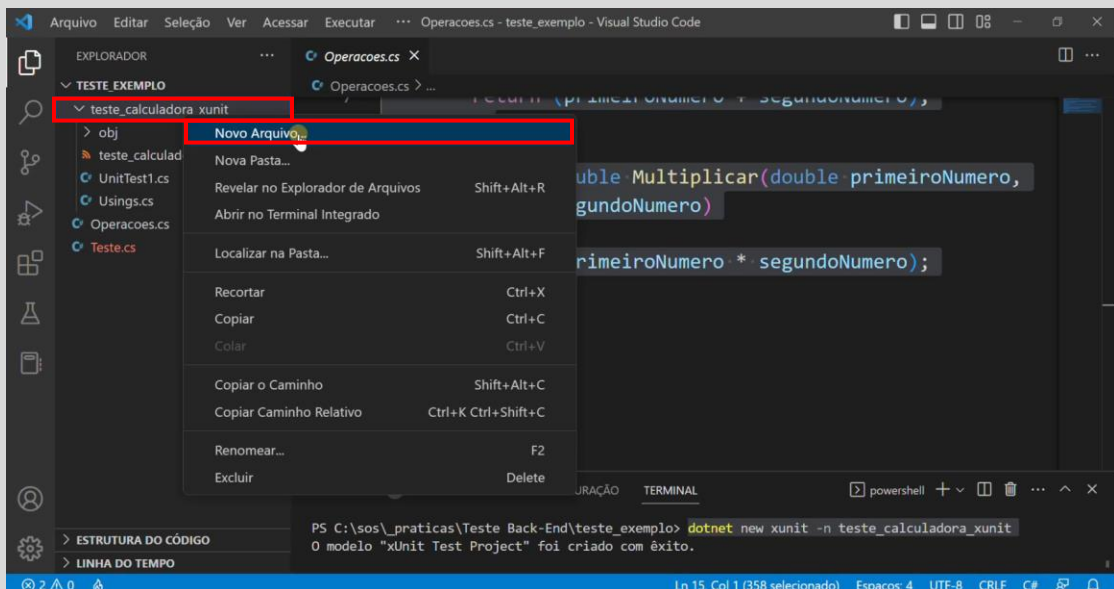
4. No arquivo teste_calculadora_xunit.csproj, altere a linha **<Nullable>enable</Nullable>** para **<Nullable>disable</Nullable>**. Assim, todas as variáveis nulas não serão destacadas.



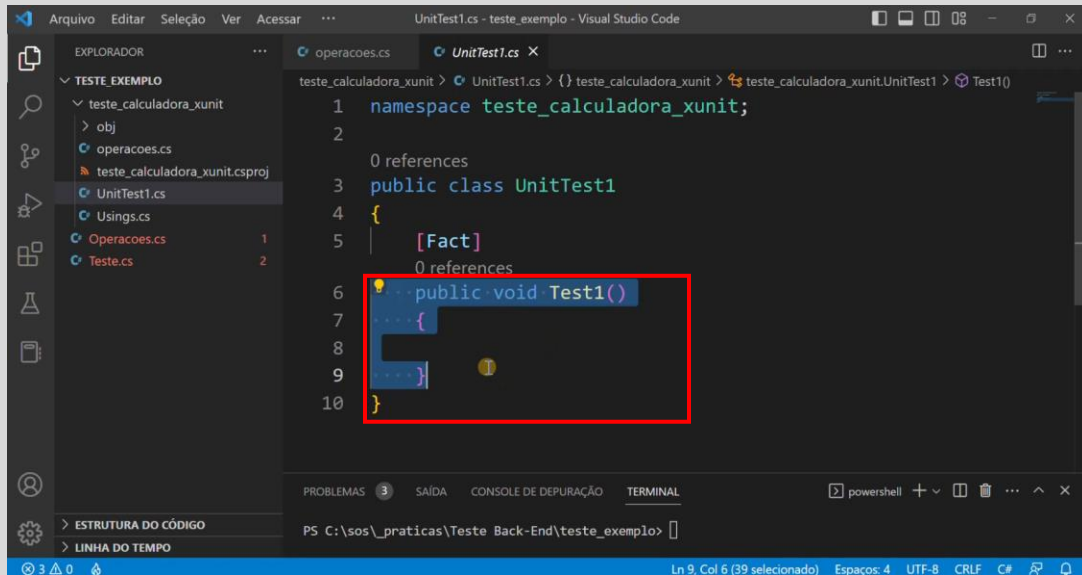
5. A próxima etapa é acrescentar a classe a ser testada à estrutura recém-criada. Note que a classe **Operacoes.cs** está fora da pasta **teste_calculadora_xunit**.



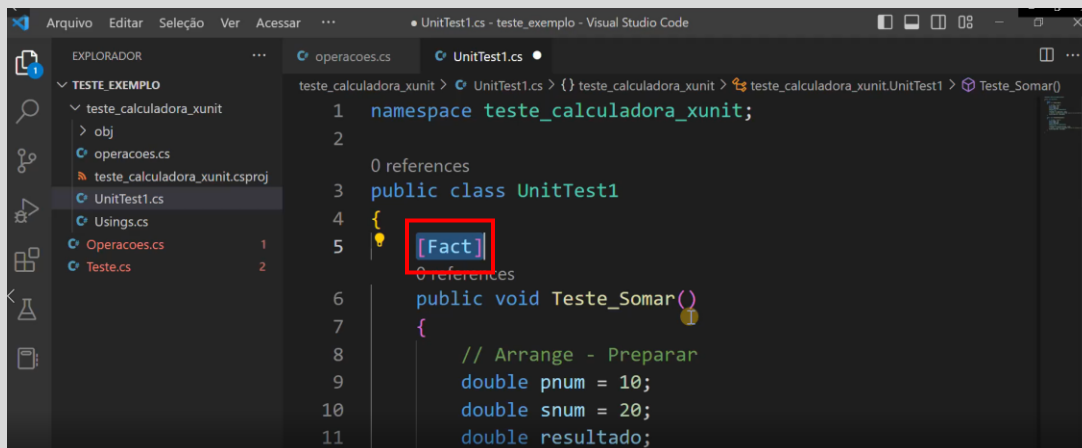
6. Clique com o botão direito na pasta **teste_calculadora_xunit** e selecione **Novo Arquivo**. Nomeie-o como **operações.cs**.



9. Selecione e copie todo o código de **Teste.cs** (arquivo fora da pasta) e cole-o em **UnitTest1.cs** (classe dentro da pasta teste_calculadora_xunit), substituindo o método **public void Test1()**, em destaque na imagem a seguir.



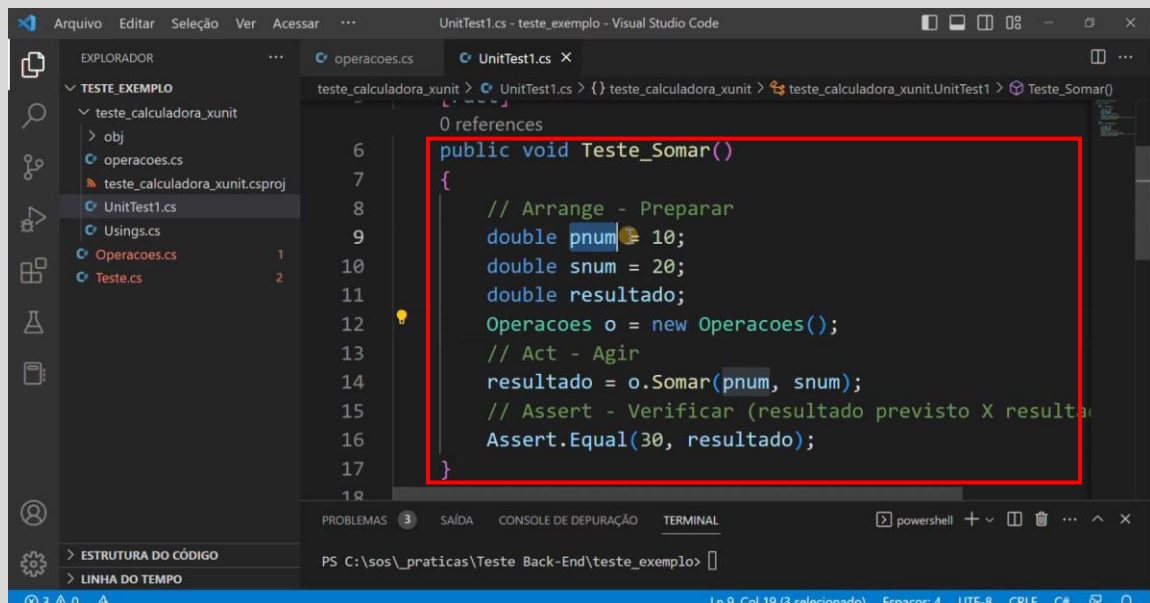
10. Selecione e copie o marcador **[Fact]**, na linha anterior ao método **Teste_Somar()**. Cole-o na linha anterior ao método **Teste_Multiplicar()**. Cada teste deve ter seu marcador **[Fact]**. Salve as alterações.



11. Em **UnitTest1.cs**, o método **Teste_Somar()** pode ser subdividido em três partes:

- **Arrange (preparar)**: na qual são fornecidos os dados de entrada, a variável resultado e o construtor de **Operacoes()**.
- **Act (agir)**: na qual se determina que o valor de resultado é a soma dos dados de entrada.
- **Assert (verificar)**: na qual se testa se o valor calculado é igual ao previsto.

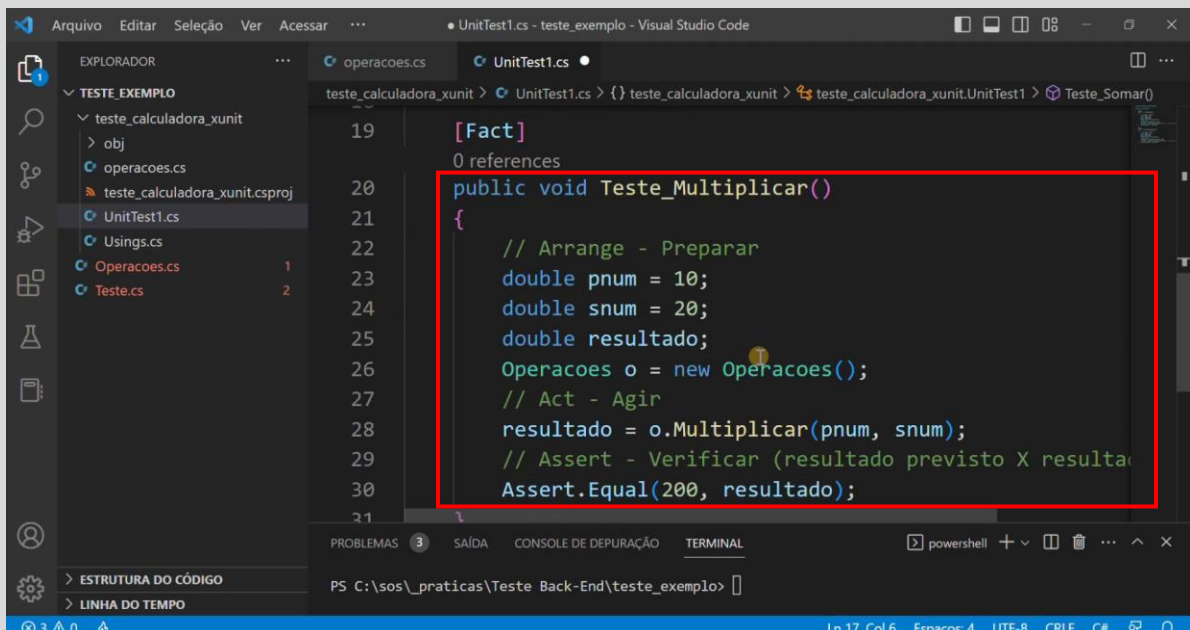
```
public void Teste_Somar()
{
    // Arrange - Preparar
    double pnum = 10;
    double snum = 20;
    double resultado;
    Operacoes o = new Operacoes();
    // Act - Agir
    resultado = o.Somar(pnum, snum);
    // Assert - Verificar (resultado previsto X resultado)
    Assert.Equal(30, resultado);
}
```



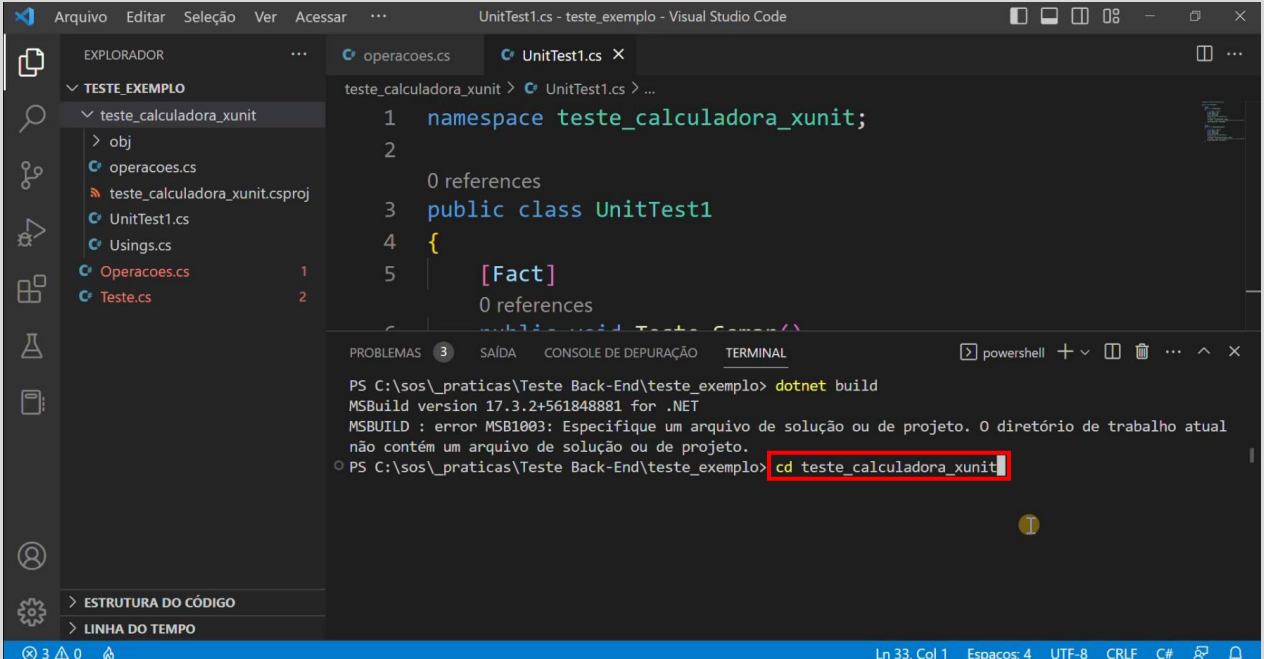
12. Ainda em **UnitTest1.cs**, logo após o método **Teste_Somar()**, há o método **Teste_Multiplicar()**, que também pode ser subdividido em três partes:

- **Arrange (preparar)**: na qual são fornecidos os dados de entrada, a variável resultado e o construtor de **Operacoes()**.
- **Act (agir)**: na qual se determina que o valor de resultado é a multiplicação dos dados de entrada.
- **Assert (verificar)**: na qual se testa se o valor calculado é igual ao previsto.

```
public void Teste_Multiplicar()
{
    // Arrange - Preparar
    double pnum = 10;
    double snum = 20;
    double resultado;
    Operacoes o = new Operacoes();
    // Act - Agir
    resultado = o.Multiplicar(pnum, snum);
    // Assert - Verificar (resultado previsto X resultado)
    Assert.Equal(200, resultado);
}
```



13. Agora com os módulos completos, é preciso compilar o projeto. Antes disso, devemos entrar na pasta correta. Para isso, digite no terminal **cd teste_calculadora_xunit** e aperte **enter**.



The screenshot shows the Visual Studio Code interface. On the left, the Explorer pane shows a project named 'TESTE_EXEMPLO' with a subfolder 'teste_calculadora_xunit'. Inside this folder are files: 'obj', 'operacoes.cs', 'teste_calculadora_xunit.csproj', 'UnitTest1.cs', 'Usings.cs', 'Operacoes.cs', and 'Teste.cs'. The main editor shows the content of 'UnitTest1.cs', which contains the following code:

```
1 namespace teste_calculadora_xunit;
2
3 0 references
4 public class UnitTest1
5 {
6     [Fact]
7     0 references
8     public void Teste_Soma()
```

At the bottom, the Terminal pane shows the command prompt output:

```
PS C:\sos\praticas\Teste Back-End\teste_exemplo> dotnet build
MSBuild version 17.3.2+561848881 for .NET
MSBUILD : error MSB1003: Especifique um arquivo de solução ou de projeto. O diretório de trabalho atual
não contém um arquivo de solução ou de projeto.
PS C:\sos\praticas\Teste Back-End\teste_exemplo> cd teste_calculadora_xunit
```

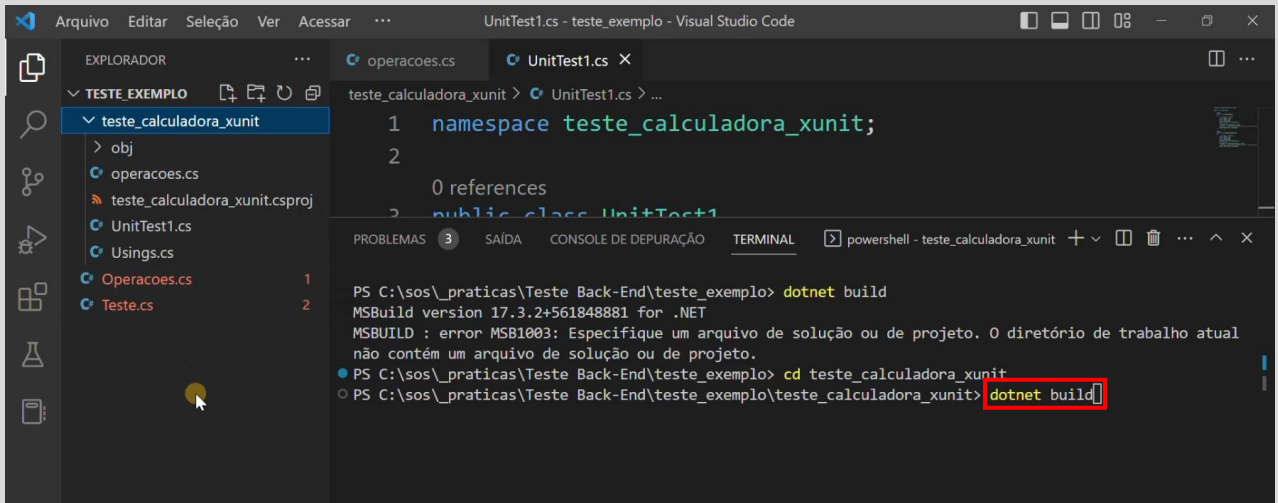
The command 'cd teste_calculadora_xunit' is highlighted with a red box.

Importante!

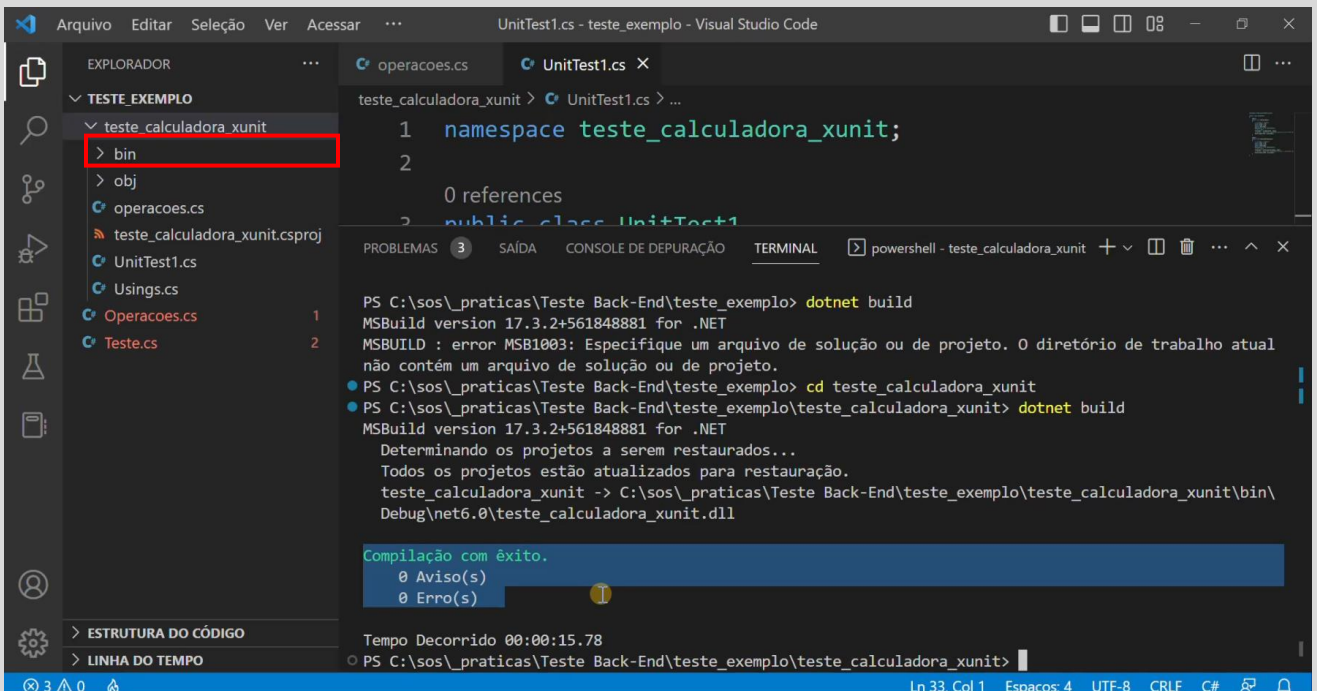
O nome da pasta deve ser o mesmo do namespace. Além disso, você deve estar dentro da pasta do projeto antes de seguir para o próximo passo.



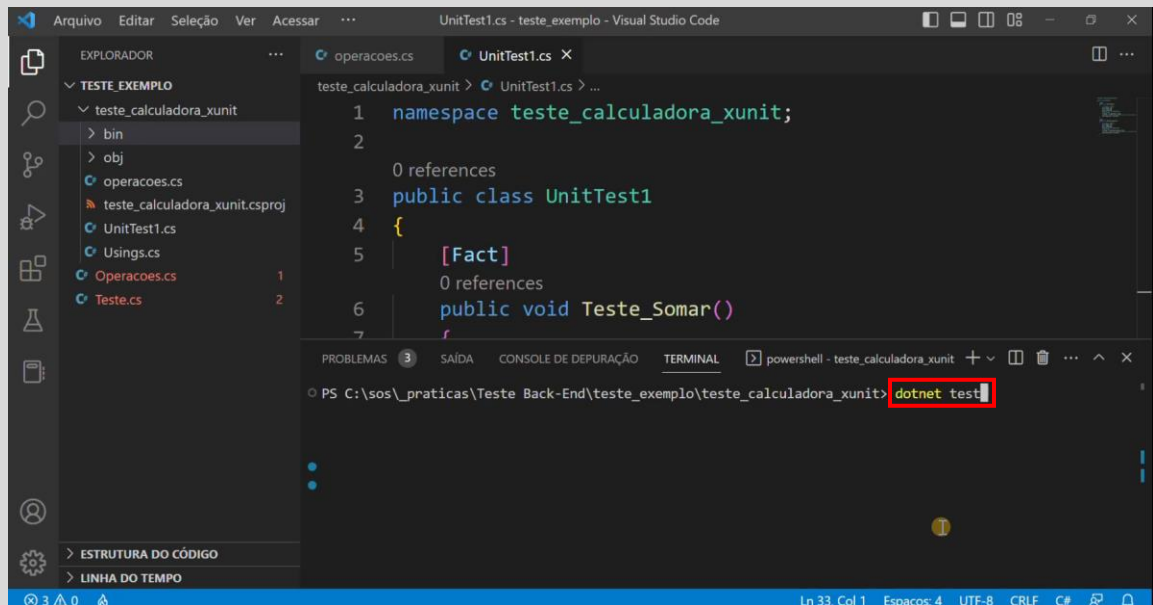
14. Agora, dentro da pasta correta, digite no terminal **dotnet build** e aperte **enter**.



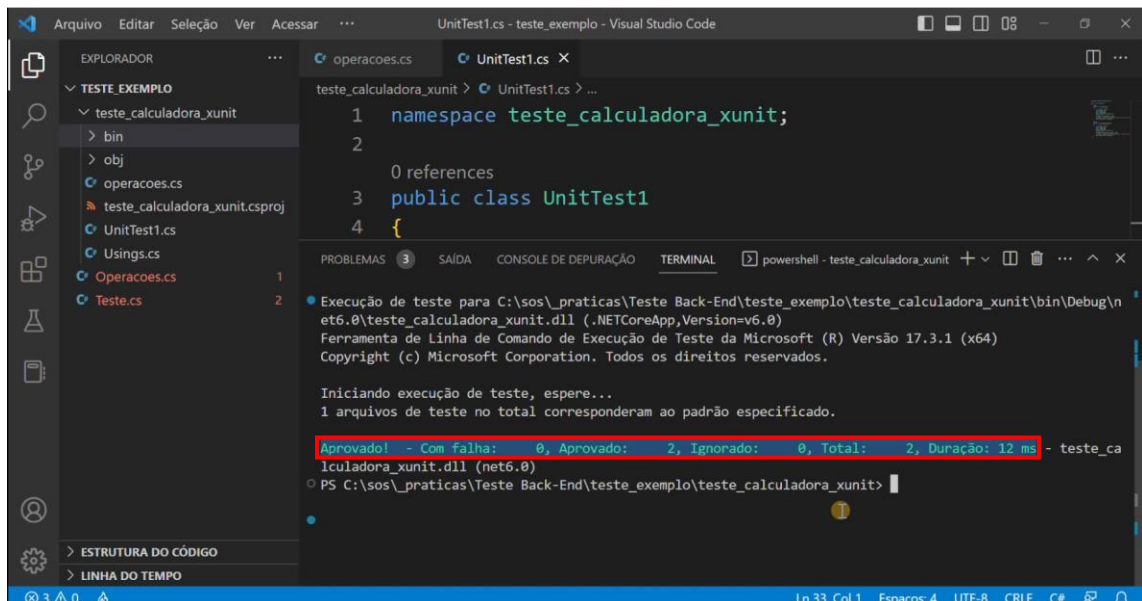
15. No explorador de arquivos, na lateral esquerda, aparecer3 a pasta **bin** ap3s a compila3o bem-sucedida.



16. Para executar o teste propriamente dito, digite no terminal **dotnet test** e aperte **enter**.



17. Confira o resultado no terminal: 0 falhas, 2 aprovados, 0 ignorados (portanto, total 2) e duração do teste (12 ms).



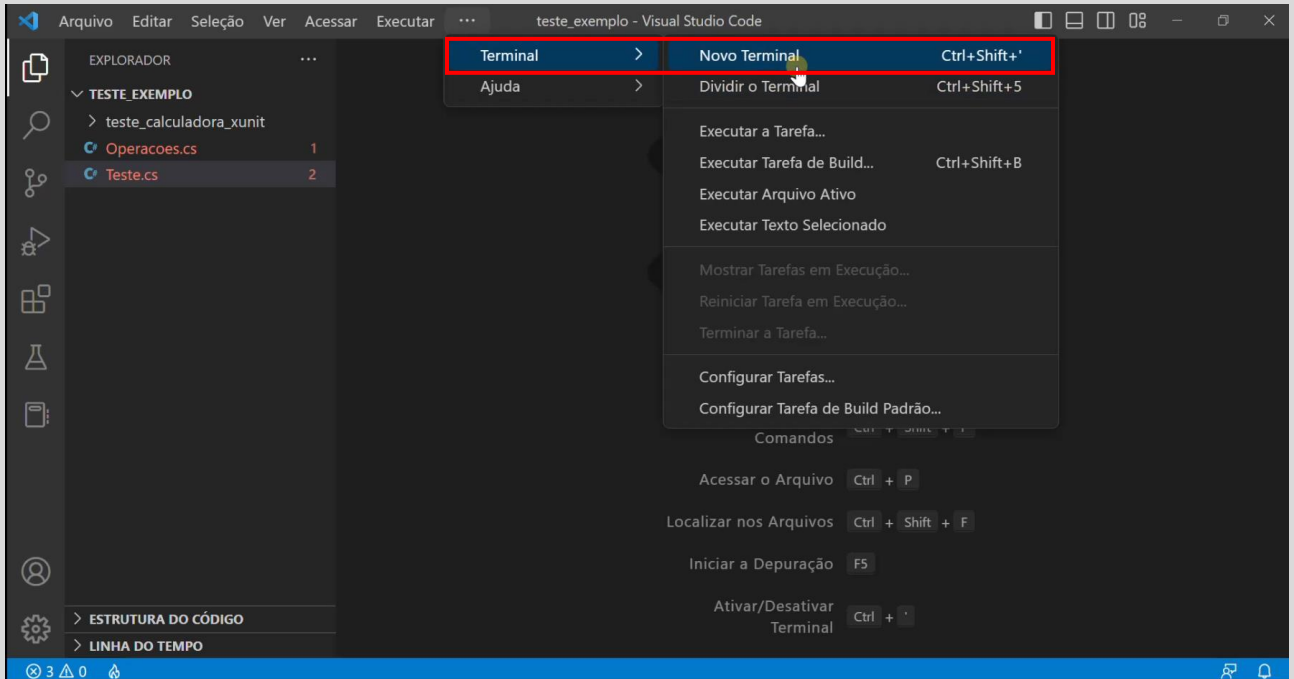
Dica!

Altere os valores de entrada, do resultado previsto ou (no operador aritmético) dos métodos testados. Salve as alterações e faça o teste de novo, mas agora simulando um erro, e analise os resultados.

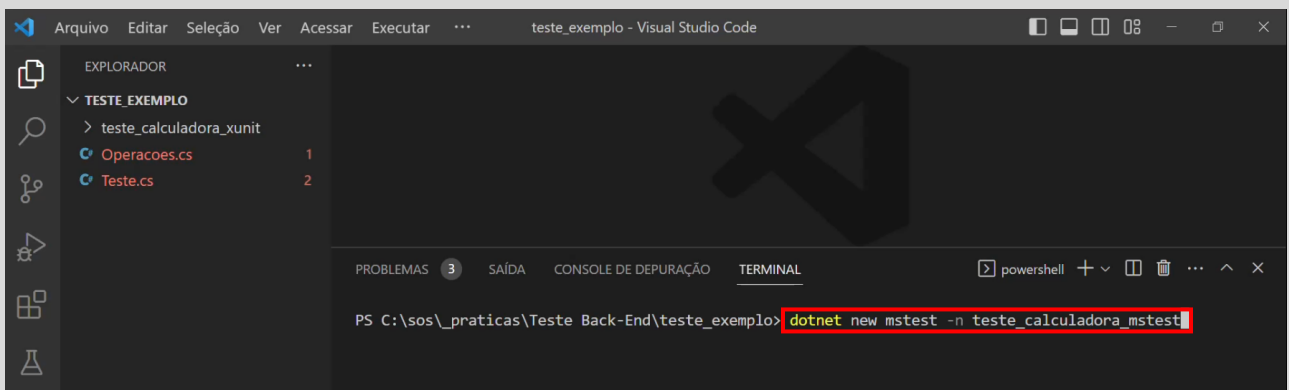


Teste com MSTest

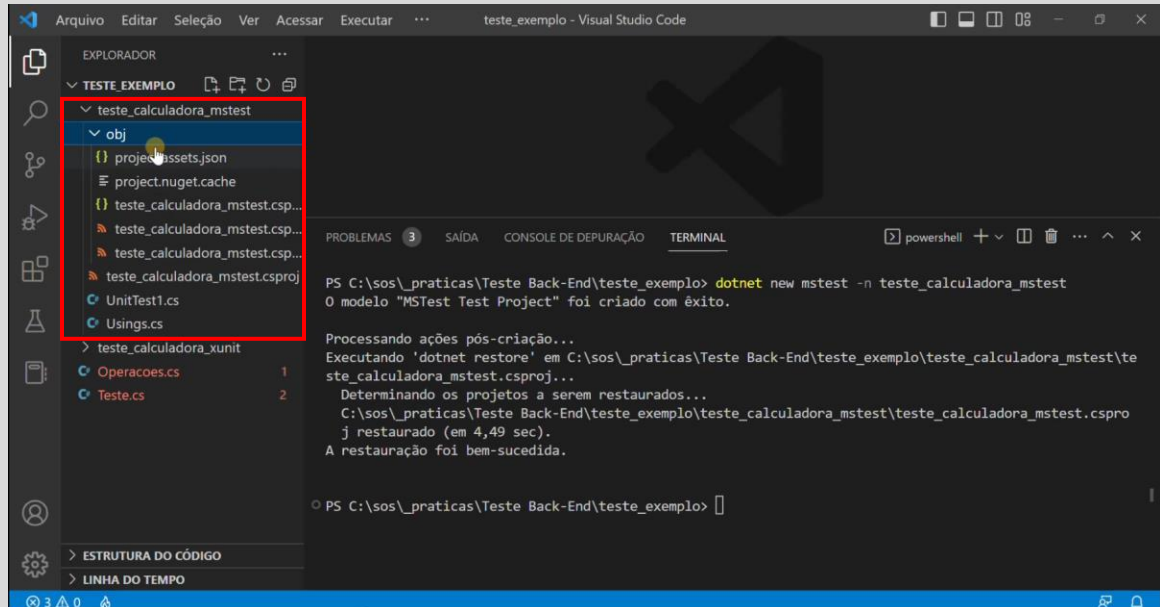
1. No menu superior do VS Code, clique em “**Terminal**” e, depois, em “**Novo Terminal**”.



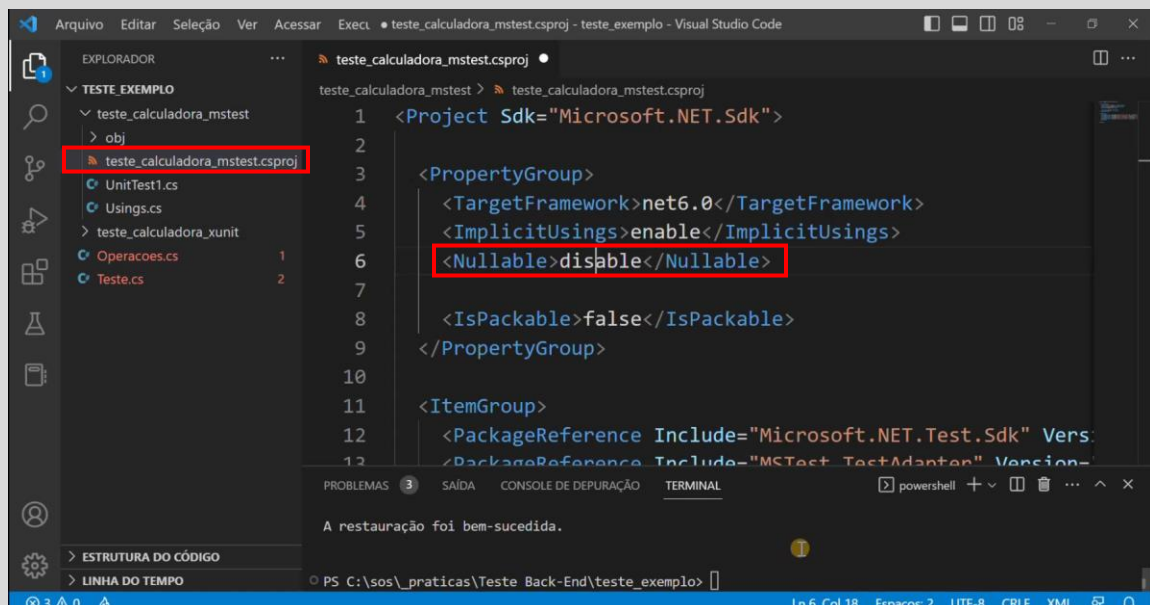
2. No terminal, digite **dotnet new mstest -n teste_calculadora_mstest** e aperte **enter** para criar uma aplicação de teste. Certifique-se de estar na pasta correta. No exemplo, a pasta deve ser TESTE-EXEMPLO.



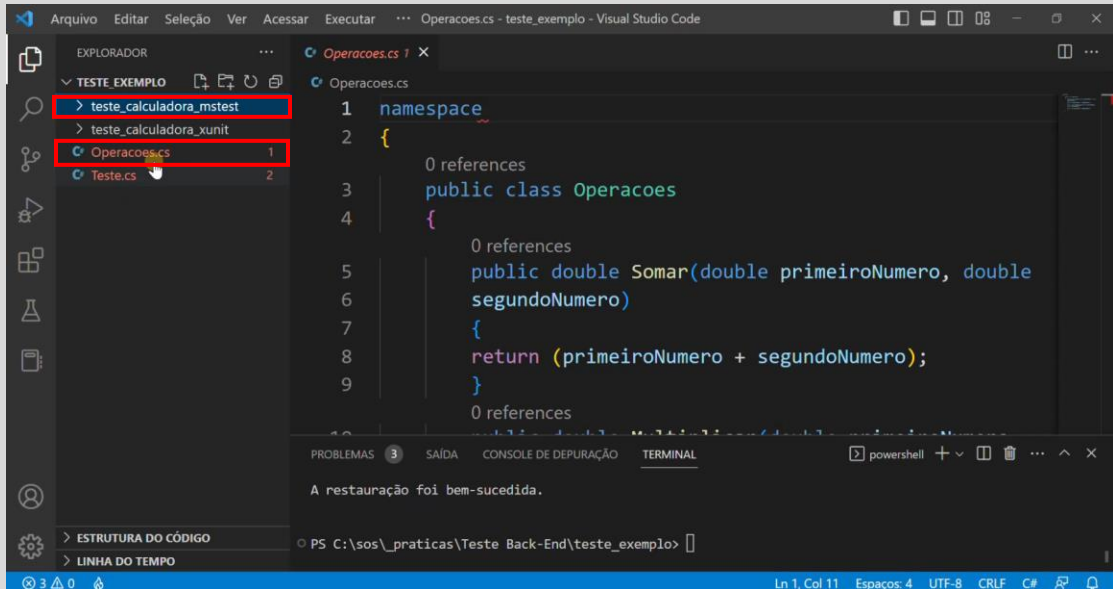
3. Após a execução, o explorador de arquivos localizado no canto esquerdo mostrará a pasta **teste_calculadora_mstest**, criada com a estrutura principal pronta e as dependências necessárias.



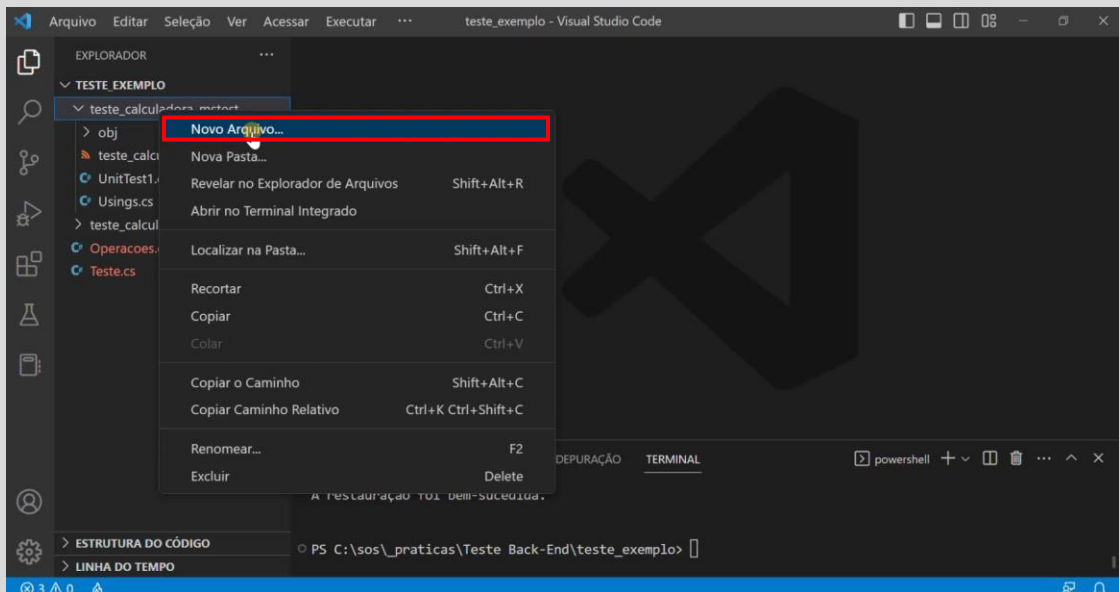
4. No arquivo **teste_calculadora_mstest.csproj**, altere a linha **<Nullable>enable</Nullable>** para **<Nullable>disable</Nullable>**. Assim, todas as variáveis nulas não serão destacadas.



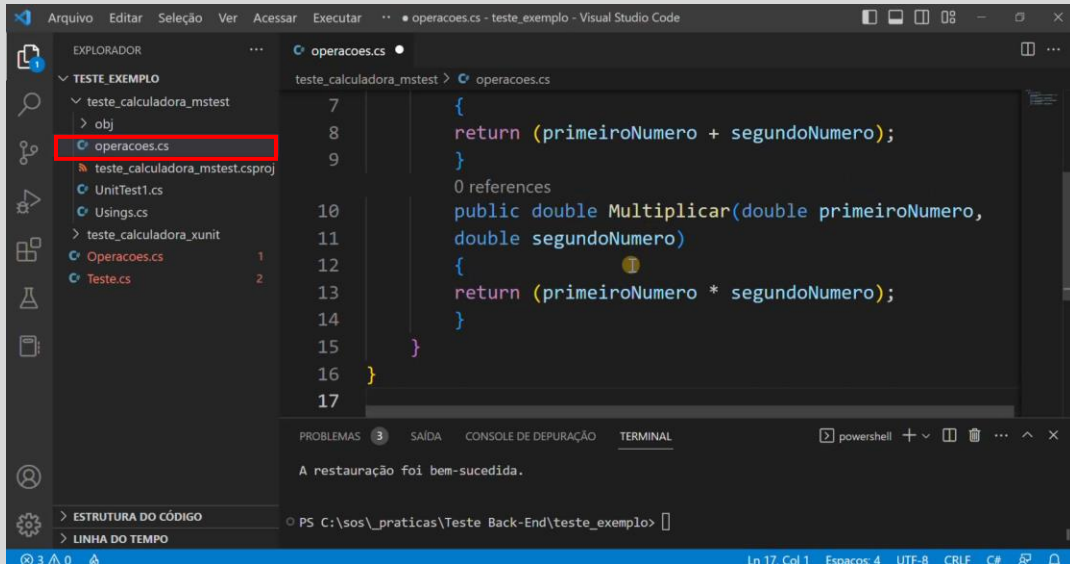
5. A próxima etapa é acrescentar a classe a ser testada à estrutura recém-criada. Note que a classe **Operacoes.cs** está fora da pasta **teste_calculadora_mstest**.



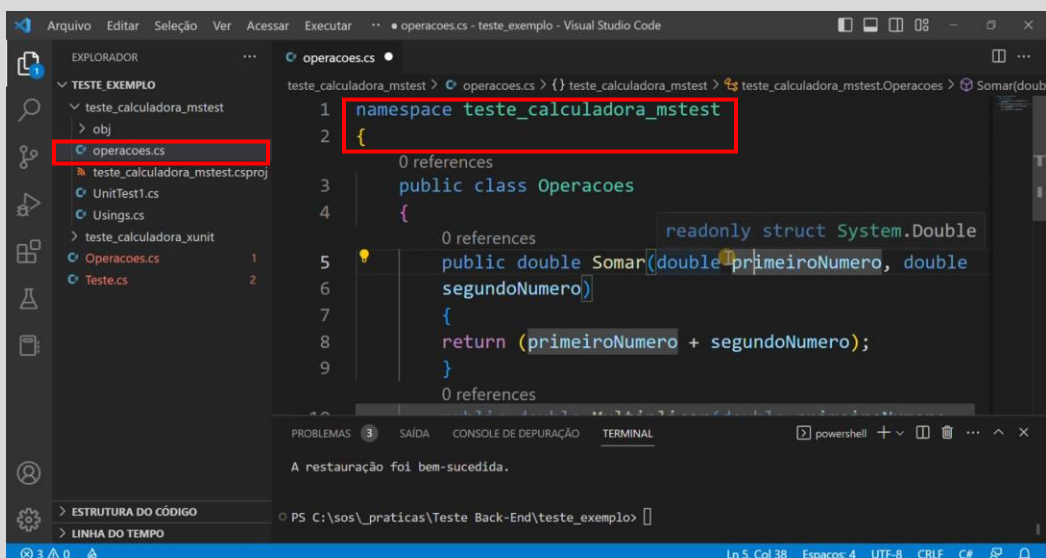
6. Clique com o botão direito na pasta **teste_calculadora_mstest** e clique em “**Novo Arquivo**”. Nomeie-o como **operações.cs**.



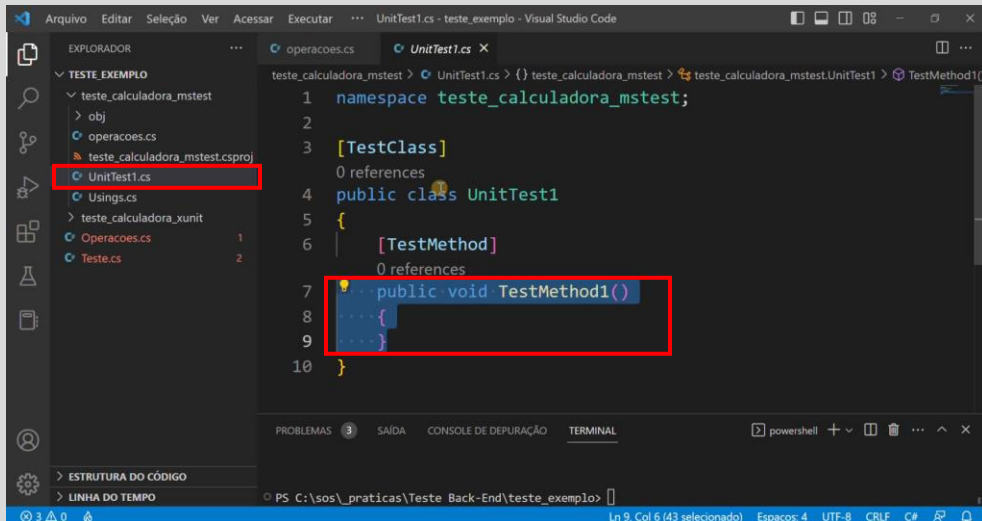
7. Selecione e copie todo o código de **operacoes.cs** (arquivo fora da pasta) e cole em **operacoes.cs** (classe dentro da pasta teste_calculadora_mstest). Salve as alterações.



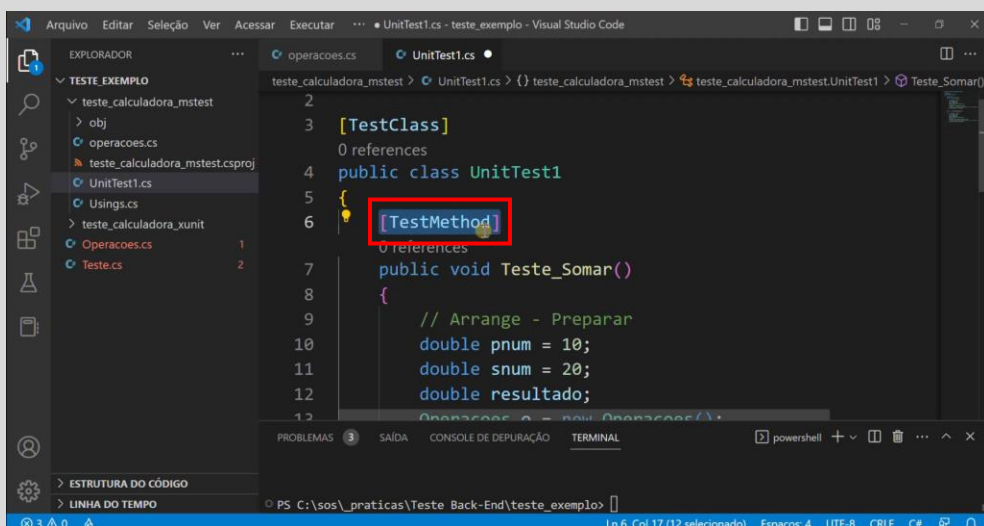
8. Em **operacoes.cs**, na primeira linha, acrescente **namespace teste_calculadora_mstest**. Lembre-se de **abrir chaves** depois do namespace e de **fechá-las** no final, fazendo com que o namespace envolva todo o código. Salve as alterações.



9. Selecione e copie todo o código de **Teste.cs** (arquivo fora da pasta) e cole-o em **UnitTest1.cs** (classe dentro da pasta **teste_calculadora_mstest**), substituindo o método **public void Test1**. Salve as alterações.



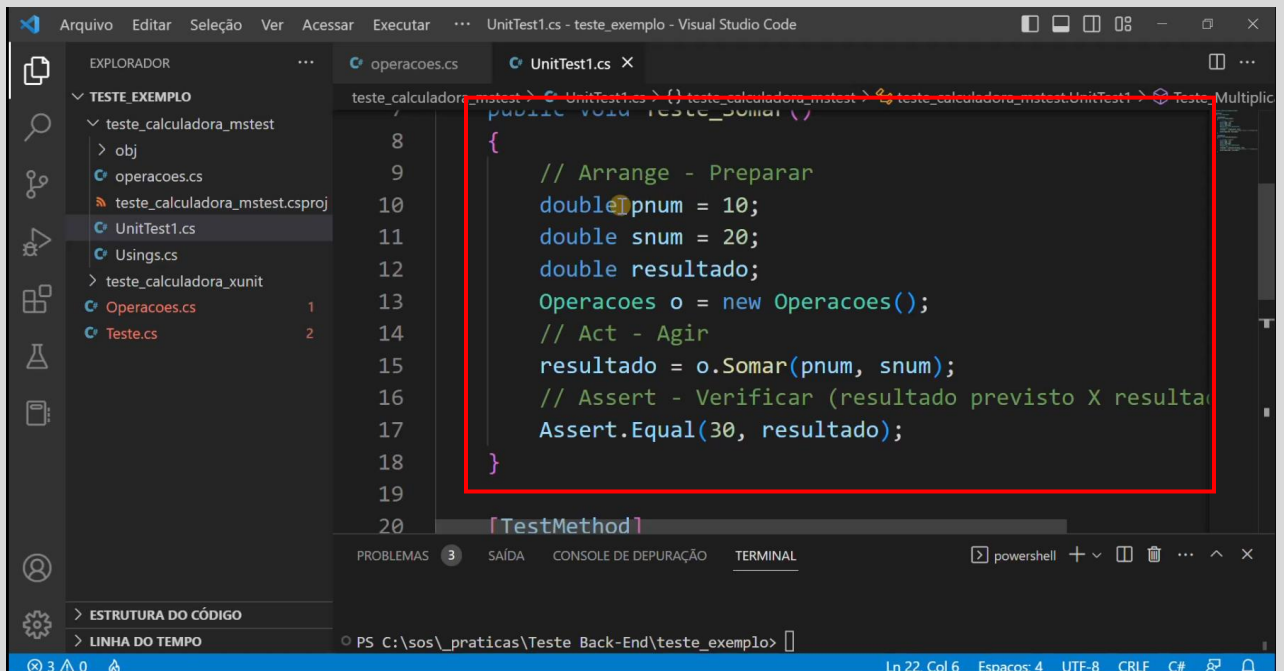
10. Selecione e copie o marcador **[TestMethod]**, localizado na linha anterior ao método **Teste_Somar()**. Cole-o na linha anterior ao método **Teste_Multiplicar()**. Cada teste deve ter o próprio marcador **[TestMethod]**. Salve as alterações.



11. Em **UnitTest1.cs**, o método **Teste_Somar()** pode ser subdividido em três partes:

- **Arrange (preparar)**: na qual são fornecidos os dados de entrada, a variável resultado e o construtor de **Operacoes()**.
- **Act (agir)**: na qual se determina que o valor de resultado é a soma dos dados de entrada.
- **Assert (verificar)**: na qual se testa se o valor calculado é igual ao previsto.

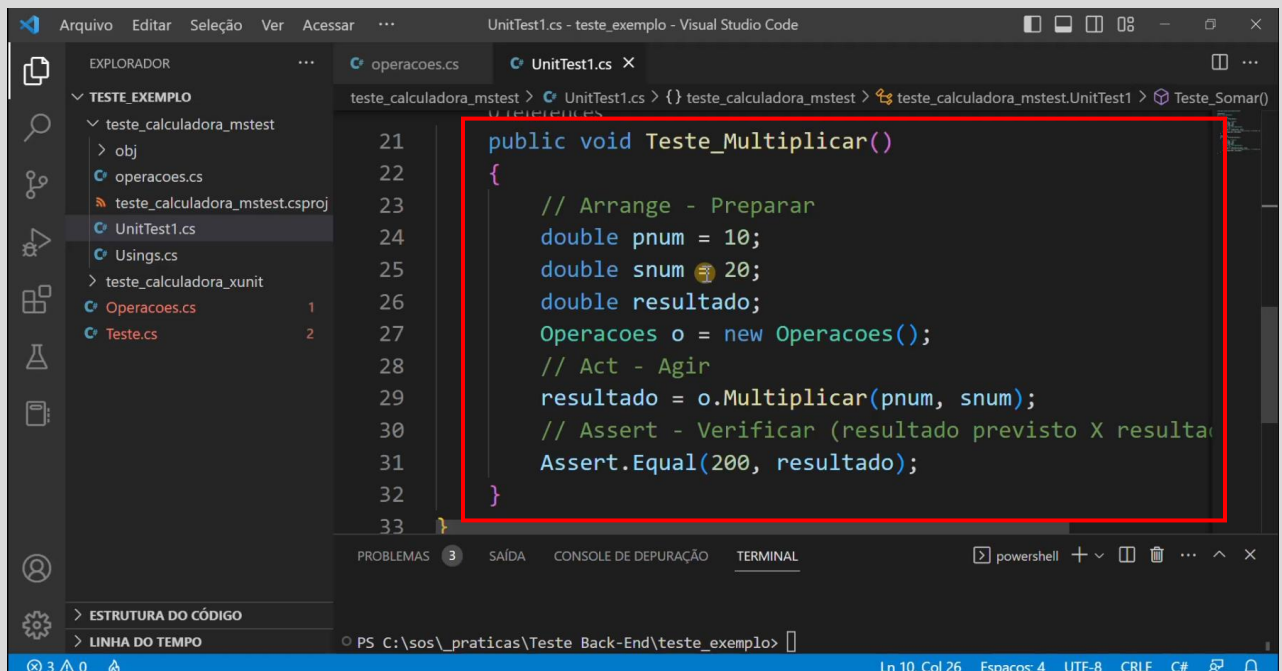
```
public void Teste_Somar()
{
    // Arrange - Preparar
    double pnum = 10;
    double snum = 20;
    double resultado;
    Operacoes o = new Operacoes();
    // Act - Agir
    resultado = o.Somar(pnum, snum);
    // Assert - Verificar (resultado previsto X resultado)
    Assert.Equal(30, resultado);
}
```



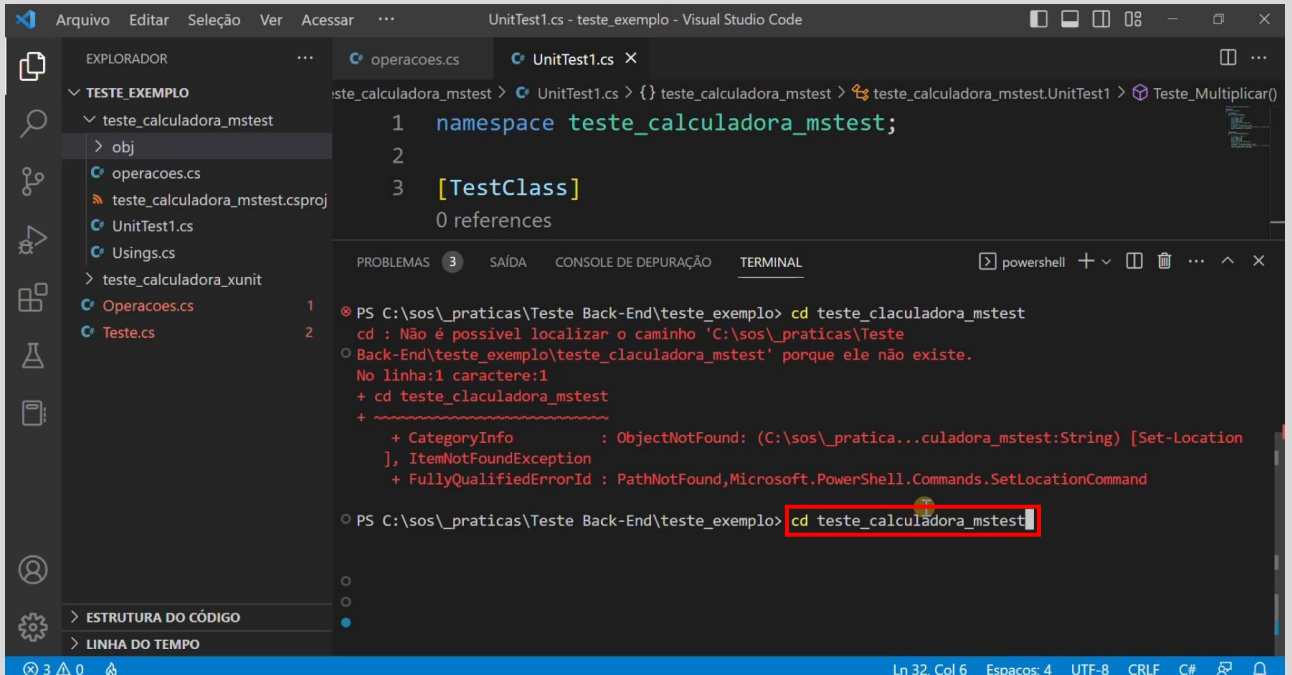
12. Ainda em **UnitTest1.cs**, logo após o método **Teste_Somar()**, há o método **Teste_Multiplicar()**, que também pode ser subdividido em três partes:

- **Arrange (preparar)**: na qual são fornecidos os dados de entrada, a variável resultado e o construtor de **Operacoes()**.
- **Act (agir)**: na qual determinamos que o valor de resultado é a multiplicação dos dados de entrada
- **Assert (verificar)**: na qual se testa se o valor calculado é igual ao previsto.

```
public void Teste_Multiplicar()
{
    // Arrange - Preparar
    double pnum = 10;
    double snum = 20;
    double resultado;
    Operacoes o = new Operacoes();
    // Act - Agir
    resultado = o.Multiplicar(pnum, snum);
    // Assert - Verificar (resultado previsto X resultado)
    Assert.Equal(200, resultado);
}
```



13. Agora com os módulos completos, é preciso compilar o projeto. Antes disso, porém, é necessário entrar na pasta correta. Para isso, digite no terminal **cd teste_calculadora_mstest** e aperte **enter**.



The screenshot shows the Visual Studio Code interface. The Explorer pane on the left shows the project structure with the folder `teste_calculadora_mstest` selected. The main editor shows the `UnitTest1.cs` file with the following code:

```
1 namespace teste_calculadora_mstest;
2
3 [TestClass]
4
5 0 references
```

The TERMINAL pane at the bottom shows the following commands and output:

```
PS C:\sos\_praticas\Teste Back-End\teste_exemplo> cd teste_calculadora_mstest
cd : Não é possível localizar o caminho 'C:\sos\_praticas\Teste Back-End\teste_exemplo\teste_calculadora_mstest' porque ele não existe.
No linha:1 caractere:1
+ cd teste_calculadora_mstest
+ ~~~~~
+ CategoryInfo          : ObjectNotFound: (C:\sos\_pratica...culadora_mstest:String) [Set-Location], ItemNotFoundException
+ FullyQualifiedErrorId : PathNotFound,Microsoft.PowerShell.Commands.SetLocationCommand

PS C:\sos\_praticas\Teste Back-End\teste_exemplo> cd teste_calculadora_mstest
```

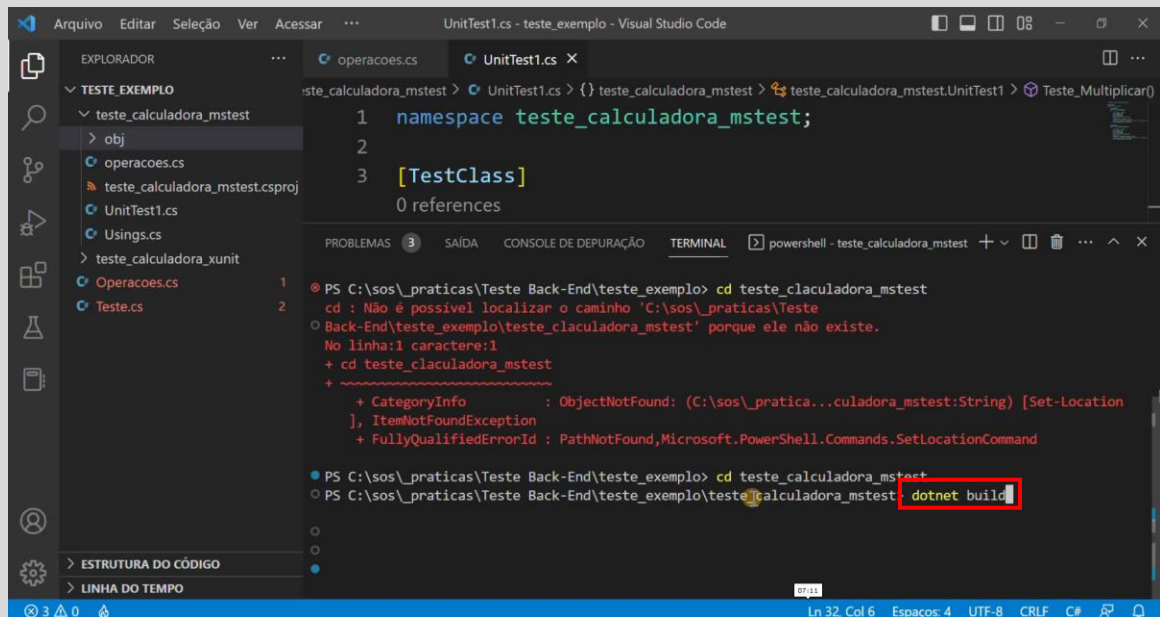
The command `cd teste_calculadora_mstest` is highlighted with a red box in the terminal.

Importante!

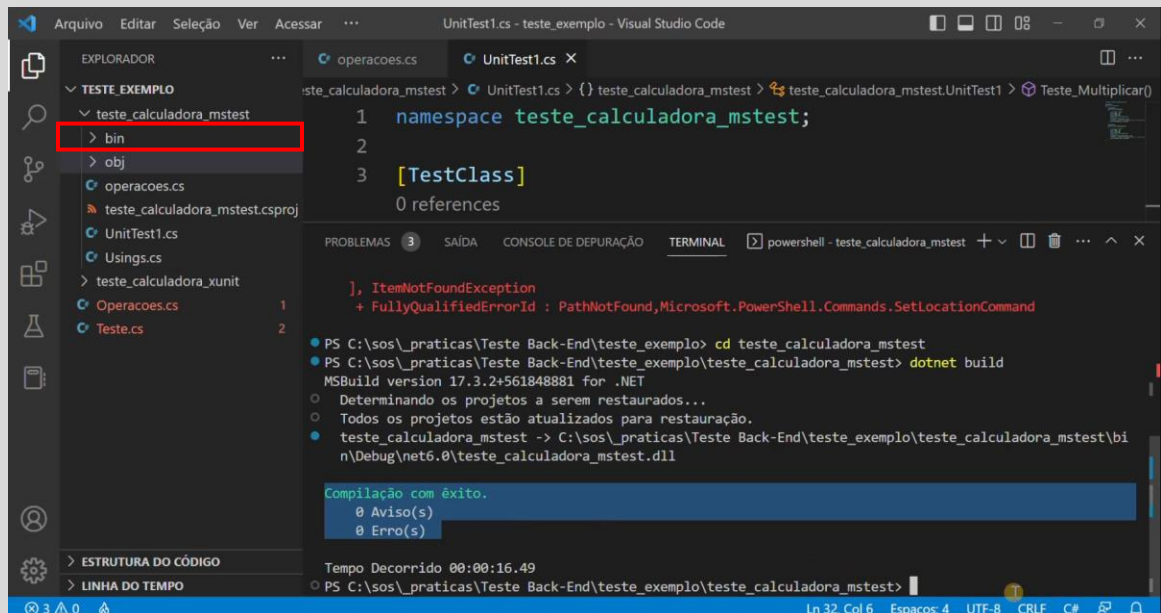
O nome da pasta deve ser o mesmo do namespace. É preciso que você esteja dentro da pasta do projeto antes de seguir para o próximo passo.



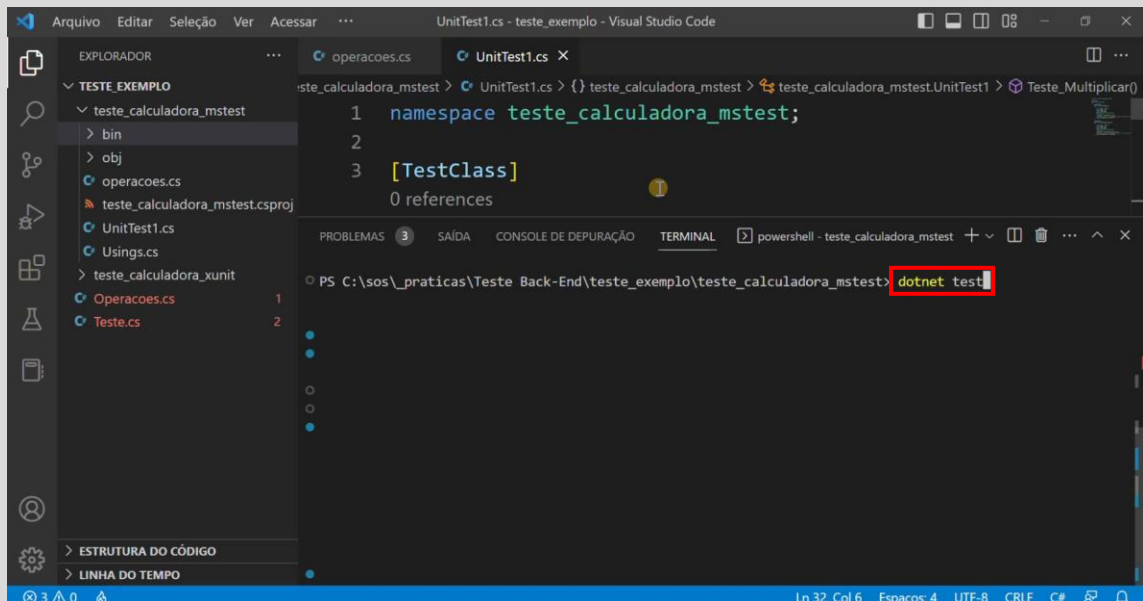
14. Agora, dentro da pasta correta, digite no terminal **dotnet build** e aperte **enter**.



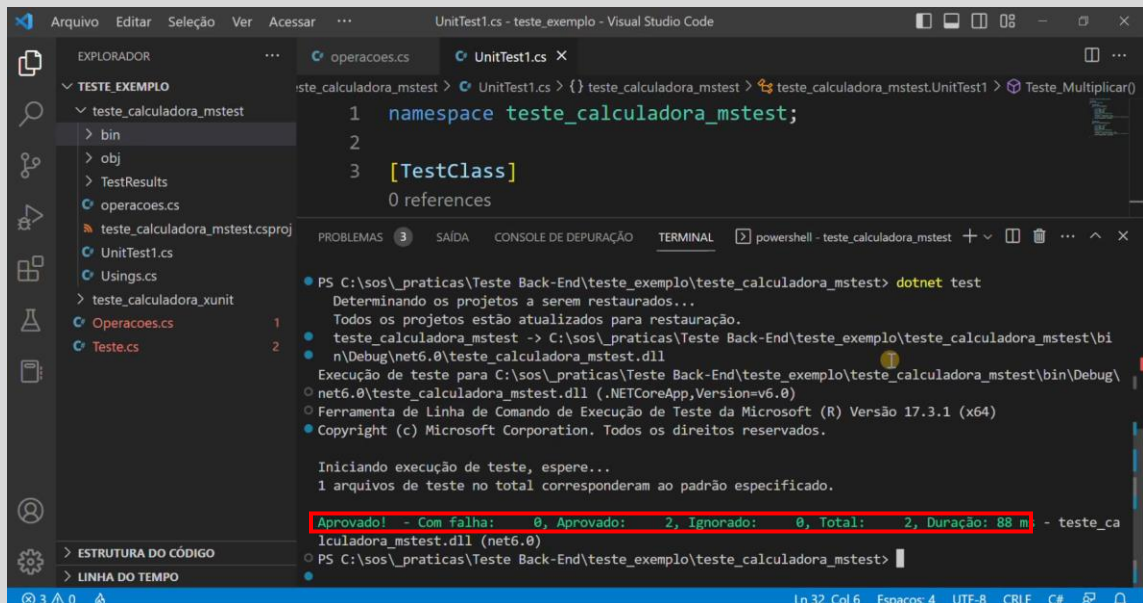
15. No explorador de arquivos, na lateral esquerda, aparecerá a pasta **bin** após a compilação bem-sucedida.



16. Para executar o teste propriamente dito, digite no terminal **dotnet test** e aperte **enter**.



17. Confira o resultado no terminal: 0 falhas, 2 aprovados, 0 ignorados (portanto, total 2) e duração do teste (12 ms).



Dica!

Altere os valores de entrada, do resultado previsto ou (no operador aritmético) dos métodos testados. Salve as alterações e faça o teste de novo, mas agora simulando um erro, e analise os resultados.

