


XUNIT

Adicionando XUnit ao projeto:

Basicamente existem duas formas de adicionarmos os testes a um projeto, podemos adicionar a dependência do XUnit ao projeto e criar uma estrutura de pastas específicas, ou podemos adicionar um projeto de testes na solução e adicionarmos os projetos que serão testados como dependências nesse projeto de teste:

```
[using System.Threading.Tasks;

namespace ClassLibrary2
{
    2 referências
    public class Calculadora
    {
        1 referência | 1/1 passando
        public double Soma(double a, double b)
        {
            return a + b;
        }
    }
}
```

 Projeto de Teste do xUnit

Um projeto que contém testes xUnit.net que podem ser executados no .NET em Windows, Linux e MacOS.

C#

Linux

macOS

Windows

Teste

Pesquisar em Gerenciador de Soluções (Ctrl+Q)

Solução 'Solution2' (2 de 2 projetos)

ClassLibrary2

Properties

Referências

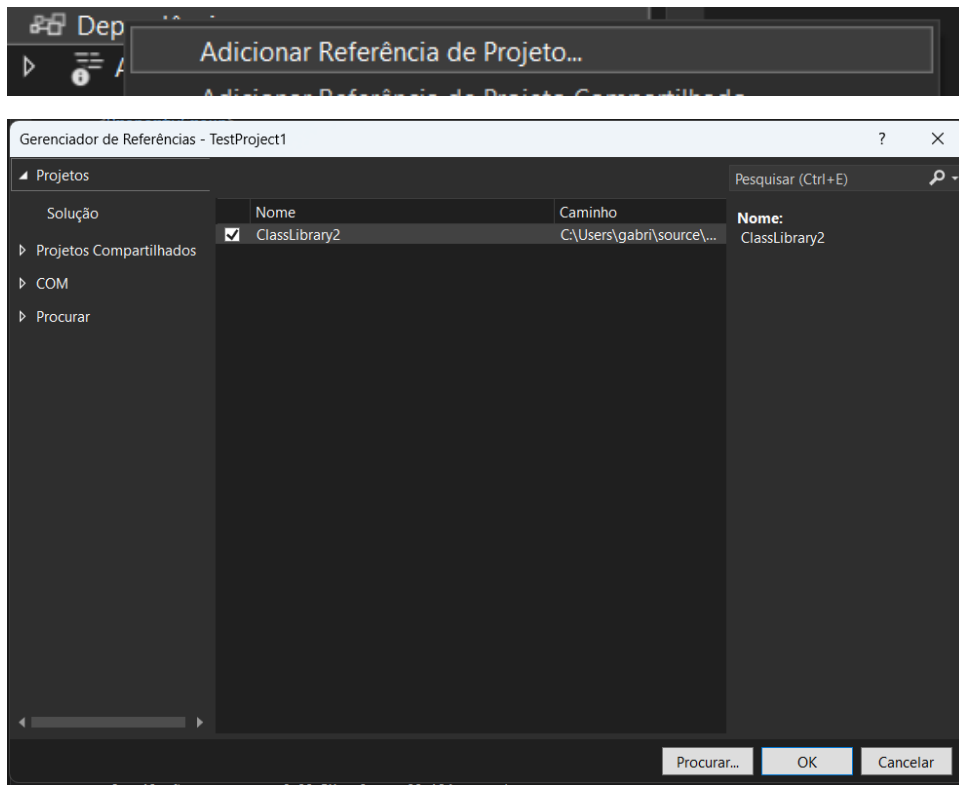
Calculadora.cs

TestProject1

Dependências

CalculadoraTeste.cs

Usings.cs



Após isso podemos criar as nossas classes de teste:

```
{
    0 referências
    public class CalculadoraTeste
    {
        [Fact]
        0 referências
        public void Soma()
        {
            Calculadora calculadora = new Calculadora();
            Assert.Equal(20, calculadora.Soma(10, 10));
        }
    }
}
```

Entendendo fatos e teorias:

Fatos são testes que são sempre verdadeiros. Eles testam condições invariantes.

As **teorias** são testes que só são verdadeiros para um determinado conjunto de dados.

Em suma se usarmos apenas um teste que use Fatos, mas teste diferentes valores se um desses valores não corresponder com a análise todo o teste terá um erro, porém se usarmos uma teoria nesse método com valores diferentes cada valor será tratado como um teste separado:

```
[Theory]
[InlineData(3)]
[InlineData(5)]
[InlineData(6)]
public void MyFirstTheory(int value)
{
    Assert.True(IsOdd(value));
}

bool IsOdd(int value)
{
    return value % 2 == 1;
}
```

Paralelismo de testes:

Apartir da versão 2.0 do xUnit foi introduzido o conceito de coleções de testes, uma coleção tem os seus testes executados sem paralelismo e os testes que não estão na mesma coleção são executados em paralelo.

Por padrão todas as classes têm a sua própria coleção e todos os métodos dentro dela são executados um após o outro, porém se esses métodos estivessem em classes separadas eles seriam executados simultaneamente.

Podemos compartilhar coleções entre as classes através da anotação “[Collection]”:

```
[Collection("Our Test Collection #1")]
public class TestClass1
{
    [Fact]
    public void Test1()
    {
        Thread.Sleep(3000);
    }
}

[Collection("Our Test Collection #1")]
public class TestClass2
{
    [Fact]
    public void Test2()
    {
        Thread.Sleep(5000);
    }
}
```

Contexto compartilhado entre classes:

Contexto compartilhado na mesma classe:

Para isso usamos o construtor como um método “before” e a interface “dispose” para eliminar da memória:

```
public class StackTests : IDisposable
{
    Stack<int> stack;

    public StackTests()
    {
        stack = new Stack<int>();
    }

    public void Dispose()
    {
        stack.Dispose();
    }

    [Fact]
    public void WithNoItems_CountShouldReturnZero()
    {
        var count = stack.Count;

        Assert.Equal(0, count);
    }

    [Fact]
    public void AfterPushingItem_CountShouldReturnOne()
    {
        stack.Push(42);

        var count = stack.Count;

        Assert.Equal(1, count);
    }
}
```

Contexto compartilhado entre outras classes:

Basicamente para termos isso temos que criar uma `IClassFigure` que basicamente é uma figura de uma classe que será passada entre as demais classes:

```
public class DatabaseFixture : IDisposable
{
    public DatabaseFixture()
    {
        Db = new SqlConnection("MyConnectionString");

        // ... initialize data in the test database ...
    }

    public void Dispose()
    {
        // ... clean up test data from the database ...
    }

    public SqlConnection Db { get; private set; }
}

public class MyDatabaseTests : IClassFixture<DatabaseFixture>
{
    DatabaseFixture fixture;

    public MyDatabaseTests(DatabaseFixture fixture)
    {
        this.fixture = fixture;
    }

    // ... write tests, using fixture.Db to get access to the SQL Server ...
}
```