Aula 33

Prof: Henrique Augusto Maltauro

Codificar Back-end de Aplicações Web

NUnit

- IResolveConstraint (Has)

A classe Has fornece diversos métodos estáticos e atributos para definir restrições baseadas em listas e objetos, entre os quais os principais são:

- Some
- None
- Exactly
- Property
- Count
- Lenght
- No

NUnit

- IResolveConstraint (Has Some)

O atributo Some vai retornar uma restrição para validar se a lista recebida possui pelo menos um elemento que esteja de acordo com uma restrição que virá em seguida.

NUnit

- IResolveConstraint (Has Some)

```
[TestFixture]
public class ClasseDeTeste
{
    [Test]
    public void MetodoDeTeste()
    {
        List<int> numeros = new List<int>()
        { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

        Assert.That(numeros, Has.Some.EqualTo(4));
    }
}
```

NUnit

- IResolveConstraint (Has None)

O atributo None vai retornar uma restrição para validar se a lista recebida não possui nenhum elemento que esteja de acordo com uma restrição que virá em seguida.

NUnit

- IResolveConstraint (Has None)

```
[TestFixture]
public class ClasseDeTeste
{
    [Test]
    public void MetodoDeTeste()
    {
        List<int> numeros = new List<int>()
        { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

        Assert.That(numeros, Has.None.EqualTo(20));
    }
}
```

NUnit

- IResolveConstraint (Has Exactly)

O método Exactly vai receber um inteiro como parâmetro e retornar uma restrição para validar se a lista recebida possui a quantidade específica de elementos que foi informada no parâmetro.

NUnit

- IResolveConstraint (Has Exactly)

```
[TestFixture]
public class ClasseDeTeste
{
    [Test]
    public void MetodoDeTeste()
    {
        List<int> numeros = new List<int>()
        { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

        Assert.That(numeros, Has.Exactly(10));
    }
}
```

NUnit

- IResolveConstraint (Has Property)

O método Property vai receber uma string como parâmetro e retornar uma restrição para validar se o objeto recebido possui a propriedade que foi informada no parâmetro.

NUnit

- IResolveConstraint (Has Property)

```
[TestFixture]
public class ClasseDeTeste
{
    [Test]
    public void MetodoDeTeste()
    {
        Pessoa pessoa = new Pessoa();
        Assert.That(numeros, Has.Property("Nome"));
    }
}
```

NUnit

- IResolveConstraint (Has Count)

O atributo Count vai retornar uma restrição com o número de elementos de uma lista, para validar se a lista está de acordo com uma restrição que virá em seguida.

NUnit

- IResolveConstraint (Has Count)

```
[TestFixture]
public class ClasseDeTeste
{
    [Test]
    public void MetodoDeTeste()
    {
        List<int> numeros = new List<int>()
        { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

        Assert.That(numeros, Has.Count.GreaterThan(1));
    }
}
```

NUnit

- IResolveConstraint (Has Length)

O atributo Length vai retornar uma restrição com o número de caracteres de uma string, para validar se a string está de acordo com uma restrição que virá em seguida.

NUnit

- IResolveConstraint (Has Length)

```
[TestFixture]
public class ClasseDeTeste
{
    [Test]
    public void MetodoDeTeste()
    {
        string texto = "MeuTexto";
        Assert.That(texto, Has.Length.EqualTo(8));
    }
}
```

NUnit

- IResolveConstraint (Has No)

O método No vai retornar a negação de uma restrição que virá em seguida.

NUnit

- IResolveConstraint (Has No)

```
[TestFixture]
public class ClasseDeTeste
{
    [Test]
    public void MetodoDeTeste()
    {
        List<int> numeros = new List<int>()
        { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

        Assert.That(numeros, Has.No.Exactly(89));
    }
}
```

NUnit

- IResolveConstraint (Does)

A classe Does fornece diversos métodos estáticos e atributos para definir restrições baseadas em listas e strings, entre os quais os principais são:

- Contain
- StartWith
- EndWith
- Not

NUnit

- IResolveConstraint (Does Contain)

O método Contain vai receber um objeto como parâmetro e retornar uma restrição para validar se a lista recebida possui o elemento que foi informado no parâmetro.

NUnit

- IResolveConstraint (Does Contain)

```
[TestFixture]
public class ClasseDeTeste
{
    [Test]
    public void MetodoDeTeste()
    {
        List<int> numeros = new List<int>()
        { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

        Assert.That(numeros, Does.Contain(8));
    }
}
```

NUnit

- IResolveConstraint (Does StartWith)

O método StartWith vai receber uma string como parâmetro e retornar uma restrição para validar se a string recebida começa com a string informada no parâmetro.

NUnit

- IResolveConstraint (Does StartWith)

```
[TestFixture]
public class ClasseDeTeste
{
    [Test]
    public void MetodoDeTeste()
    {
        string texto = "MeuTexto";
        Assert.That(texto, Does.StartWith("Me"));
    }
}
```

NUnit

- IResolveConstraint (Does EndWith)

O método EndWith vai receber uma string como parâmetro e retornar uma restrição para validar se a string recebida termina com a string informada no parâmetro.

NUnit

- IResolveConstraint (Does EndWith)

```
[TestFixture]
public class ClasseDeTeste
{
    [Test]
    public void MetodoDeTeste()
    {
        string texto = "MeuTexto";
        Assert.That(texto, Does.EndWith("xto"));
    }
}
```

NUnit

- IResolveConstraint (Does Not)

O atributo Not vai retornar a negação de uma restrição que virá em seguida.

NUnit

- IResolveConstraint (Does Not)

```
[TestFixture]
public class ClasseDeTeste
{
    [Test]
    public void MetodoDeTeste()
    {
        string texto = "MeuTexto";
        Assert.That(texto, Does.Not.EndWith("sdafs"));
    }
}
```

NUnit

- IResolveConstraint (And)

O atributo And permite concatenar uma restrição a outra, fazendo com que ambas necessitam da validação de sucesso para passar.

NUnit

- IResolveConstraint (And)

```
[TestFixture]
public class ClasseDeTeste
{
    [Test]
    public void MetodoDeTeste()
    {
        string texto = "MeuTexto";
        Assert.That(texto, Has.Length.EqualTo(4)
                                .And.EndWith("xto"));
    }
}
```

NUnit

- IResolveConstraint (Throws)

A classe Throws fornece diversos métodos estáticos para definir restrições baseadas em exceções, entre os quais os principais são:

- Exception
- ArgumentException
- ArgumentNullException
- TypeOf
- Nothing

NUnit

- IResolveConstraint (Throws Exception)

O método Exception vai retornar uma restrição para validar se o processo executado estoura uma exceção do tipo Exception.

NUnit

- IResolveConstraint (Throws Exception)

```
[TestFixture]
public class ClasseDeTeste
{
    [Test]
    public void MetodoDeTeste()
    {
        Assert.That(() => MetodoDeExcecao(), Throws.Exception);
    }

    public void MetodoDeExcecao()
    {
        throw new Exception();
    }
}
```

NUnit

- IResolveConstraint (Throws ArgumentException)

O método ArgumentException vai retornar uma restrição para validar se o processo executado estoura uma exceção do tipo ArgumentException.

NUnit

- IResolveConstraint (Throws ArgumentException)

```
[TestFixture]
public class ClasseDeTeste
{
    [Test]
    public void MetodoDeTeste()
    {
        Assert.That(() => MetodoDeExcecao(), Throws.ArgumentException);
    }

    public void MetodoDeExcecao()
    {
        throw new ArgumentException();
    }
}
```

NUnit

- IResolveConstraint (Throws ArgumentNullException)

O método ArgumentNullException vai retornar uma restrição para validar se o processo executado estoura uma exceção do tipo ArgumentNullException.

NUnit

- IResolveConstraint (Throws ArgumentNullException)

```
[TestFixture]
public class ClasseDeTeste
{
    [Test]
    public void MetodoDeTeste()
    {
        Assert.That(() => MetodoDeExcecao(), Throws.ArgumentNullException);
    }

    public void MetodoDeExcecao()
    {
        throw new ArgumentNullException();
    }
}
```

NUnit

- IResolveConstraint (Throws TypeOf)

O método TypeOf vai receber um parâmetro de tipo e retornar uma restrição para validar se o processo executado estoura uma exceção do mesmo tipo que o tipo definido no parâmetro.

NUnit

- IResolveConstraint (Throws TypeOf)

```
[TestFixture]
public class ClasseDeTeste
{
    [Test]
    public void MetodoDeTeste()
    {
        Assert.That(() => MetodoDeExcecao(), Throws.TypeOf<DivideByZeroException>());
    }

    public void MetodoDeExcecao()
    {
        throw new DivideByZeroException();
    }
}
```

NUnit

- IResolveConstraint (Throws Nothing)

O método Nothing vai retornar uma restrição para validar se o processo executado não estoura nenhuma exceção.

NUnit

- IResolveConstraint (Throws Nothing)

```
[TestFixture]
public class ClasseDeTeste
{
    [Test]
    public void MetodoDeTeste()
    {
        Assert.That(() => MetodoDeExcecao(), Throws.Nothing);
    }

    public void MetodoDeExcecao()
    { }
}
```