FREE eBook

LEARNING

nunit

Free unaffiliated eBook created from
Stack Overflow contributors.

#nunit

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: nunit

It is an unofficial and free nunit ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official nunit.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with nunit

Remarks

This section provides an overview of what nunit is, and why a developer might want to use it.

It should also mention any large subjects within nunit, and link out to the related topics. Since the Documentation for nunit is new, you may need to create initial versions of those related topics.

Versions

Version	Release Date
2.2	2004-08-08
2.2.1	2004-10-26
2.2.2	2004-12-07
2.2.3	2005-02-14
2.2.4	2005-12-14
2.2.5	2005-12-22
2.2.6	2006-01-21
2.2.7	2006-02-18
2.2.8	2006-04-21
2.2.9	2006-11-26
2.2.10	2007-03-15
2.4 RC1	2007-02-25
2.4 (Final Release)	2007-03-16
2.4.1	2007-05-03
2.4.2	2007-08-02
2.4.4	2007-10-20
2.4.5	2007-11-23
2.4.6	2007-12-31

Version	Release Date
2.4.7	2008-03-30
2.4.8	2008-07-21
2.5	2009-05-02
2.5.1	2009-07-08
2.5.2	2009-08-10
2.5.3	2009-12-11
2.5.4	2010-04-08
2.5.5	2010-04-22
2.5.6	2010-07-24
2.5.7	2010-08-01
2.5.8	2010-10-14
2.5.9	2010-12-14
2.5.10	2011-04-02
2.6	2012-02-20
2.6.1	2012-08-04
2.6.2	2012-10-22
2.6.3	2013-10-10
2.6.4	2014-12-16
3.0 (Alpha 1)	2014-09-22
3.0 (Beta 1)	2015-03-25
3.0 RC1	2015-11-01
3.0.0 Final Release	2015-11-15
3.0.1	2015-12-01
3.2	2016-03-05
3.2.1	2016-04-19

Version	Release Date
3.4	2016-06-25

Examples

Installation Using NuGet

```
Install-Package NUnit
```

This package includes all assemblies needed to create unit tests.

Tests can be executed using one of the following methods:

- Visual Studio Unit Test Window
- Console runner
- Third party runner that supports NUnit 3

Visual Studio Unit Test Window

To execute tests using the Visual Studio Unit Test Window, install the NUnit 3 Test Adapter.
https://visualstudiogallery.msdn.microsoft.com/0da0f6bd-9bb6-4ae3-87a8-537788622f2d

Console Runner

Install the NUnit Console Runner via NuGet

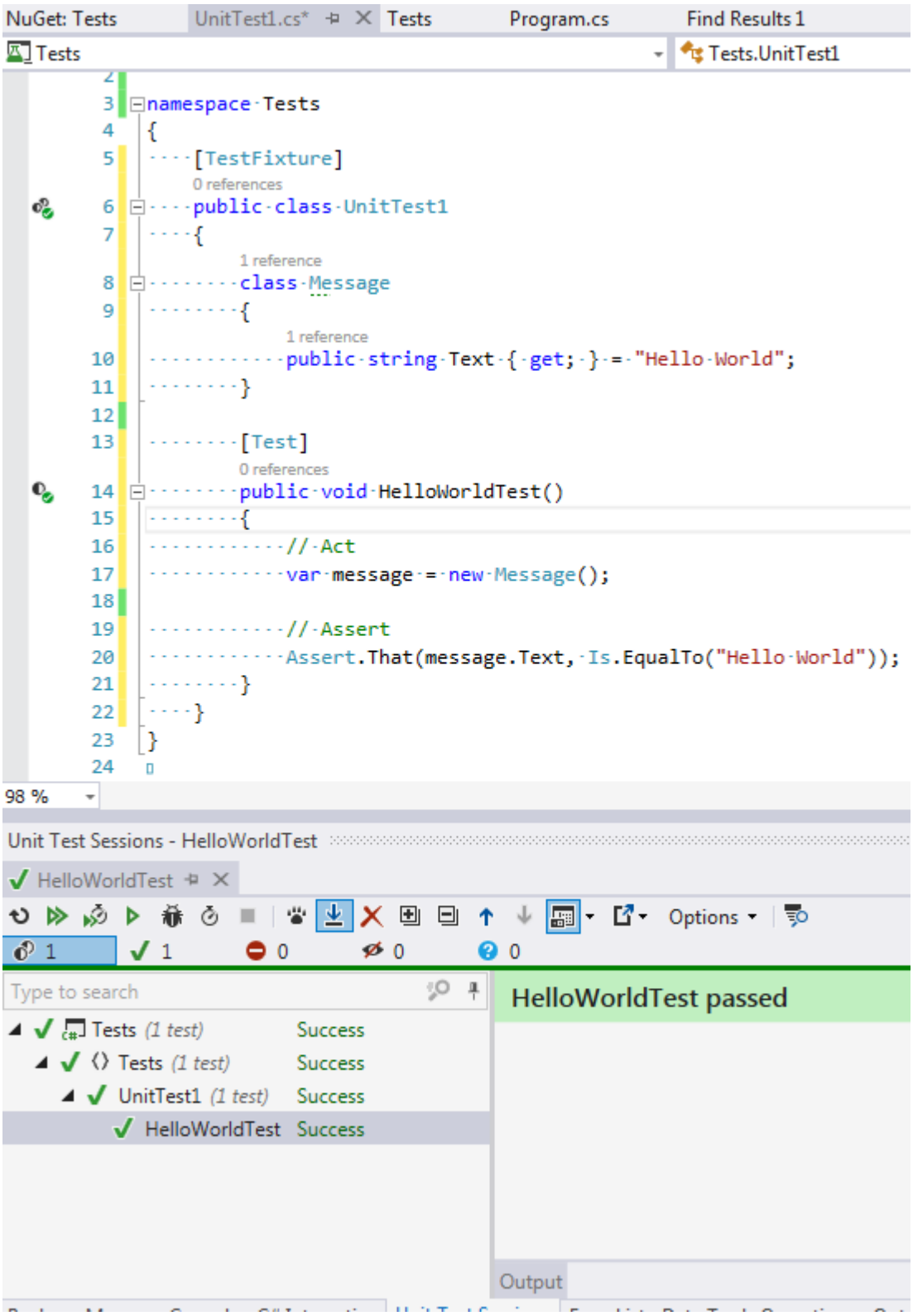
```
Install-Package NUnit.Console
```

The executable nunit3-console.exe is located in packages\NUnit.3.X.X\tools

Hello World

```
[TestFixture]
public class UnitTest1
{
    class Message
    {
        public string Text { get; } = "Hello World";
    }

    [Test]
    public void HelloWorldTest()
    {
        // Act
        var message = new Message();
```

```
        // Assert
        Assert.That(message.Text, Is.EqualTo("Hello World"));
    }
}
```



Why you can't use Assert.Equals

Ever wondered why you cannot use Assert.Equals() for both Nunit and MSTest. If you have not then maybe as a start you need to be aware that you cannot use this method. Instead you would

use Assert.AreEqual() to compare two objects for equality.

The reason here is very simple. Like any class the Assert class is inheriting from System.Object that has a public virtual Equals method meant to check if a given object is equal to the current object. Therefor calling that equals method would be a mistake as in a unit test you would instead to compare two objects that have nothing to do with the Assert class. As a result Nunit and MSTest both chose to provide a method Assert.AreEqual for that purpose.

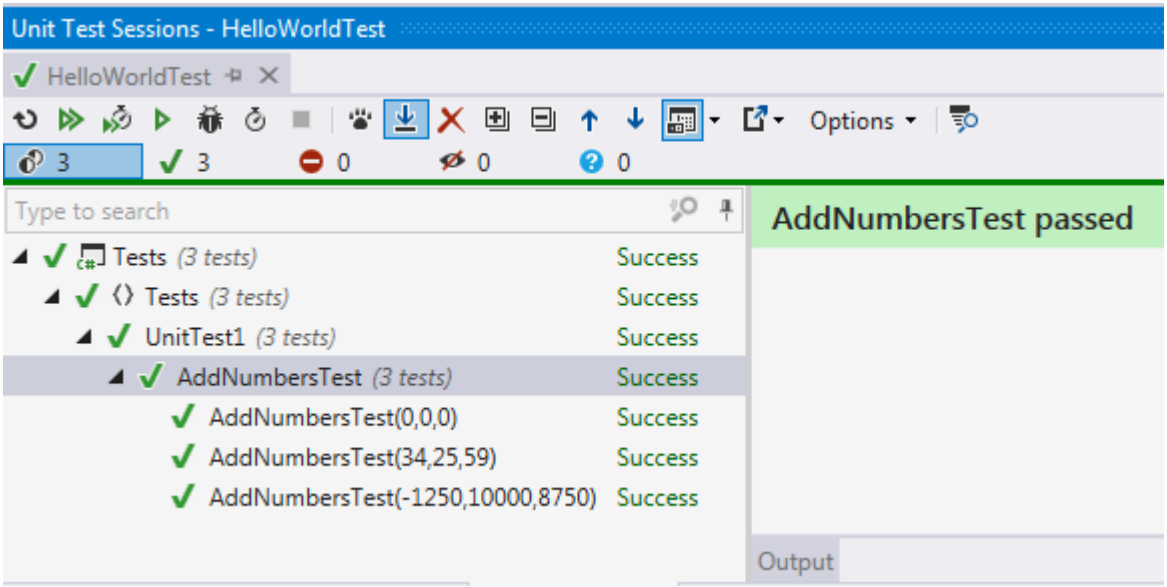
Furthermore to ensure that you do not use the Equals method by mistake they have decided to throw Exceptions to warn you if you do use this by mistake.

Nunit Implementation:

```
        [EditorBrowsable(EditorBrowsableState.Never)]
    public static new bool Equals(object a, object b)
    {
        // TODO: This should probably be InvalidOperationException
        throw new AssertionException("Assert.Equals should not be used for Assertions");
    }
```

TestCaseAttribute

```
[TestCase(0, 0, 0)]
[TestCase(34, 25, 59)]
[TestCase(-1250, 10000, 8750)]
public void AddNumbersTest(int a, int b, int expected)
{
    // Act
    int result = a + b;

    // Assert
    Assert.That(result, Is.EqualTo(expected));
}
```



Read Getting started with nunit online: https://riptutorial.com/nunit/topic/1738/getting-started-with-nunit

Chapter 2: Attributes

Remarks

Version 1 of NUnit used the classic approach to identifying tests based on inheritance and naming conventions. From version 2.0 on, NUnit has used custom attributes for this purpose.

Because NUnit test fixtures do not inherit from a framework class, the developer is free to use inheritance in other ways. And because there is no arbitrary convention for naming tests, the choice of names can be entirely oriented toward communicating the purpose of the test.

All NUnit attributes are contained in the NUnit.Framework namespace. Each source file that contains tests must include a using statement for that namespace and the project must reference the framework assembly, nunit.framework.dll.

Beginning with NUnit 2.4.6, NUnit's attributes are no longer sealed and any attributes that derive from them will be recognized by NUnit.

Examples

TestCaseAttributeExample

```
[TestCase(0, 0, 0)]
[TestCase(34, 25, 59)]
[TestCase(-1250, 10000, 8750)]
public void AddNumbersTest(int a, int b, int expected)
{
    // Act
    int result = a + b;

    // Assert
    Assert.That(result, Is.EqualTo(expected));
}
```

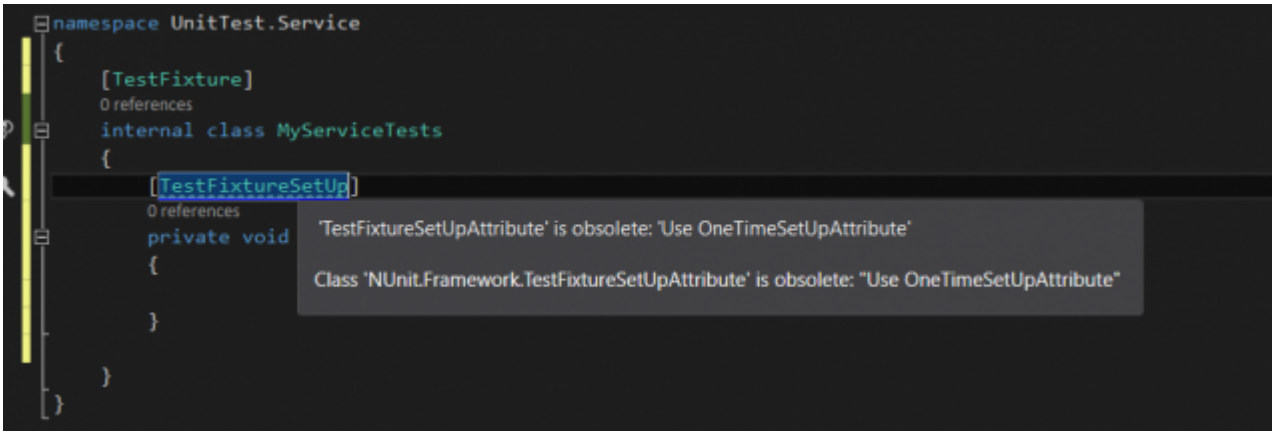
TestFixture

```
[TestFixture]
public class Tests {

    [Test]
    public void Test1() {
        Assert.That(true, Is.EqualTo(true));
    }

}
```

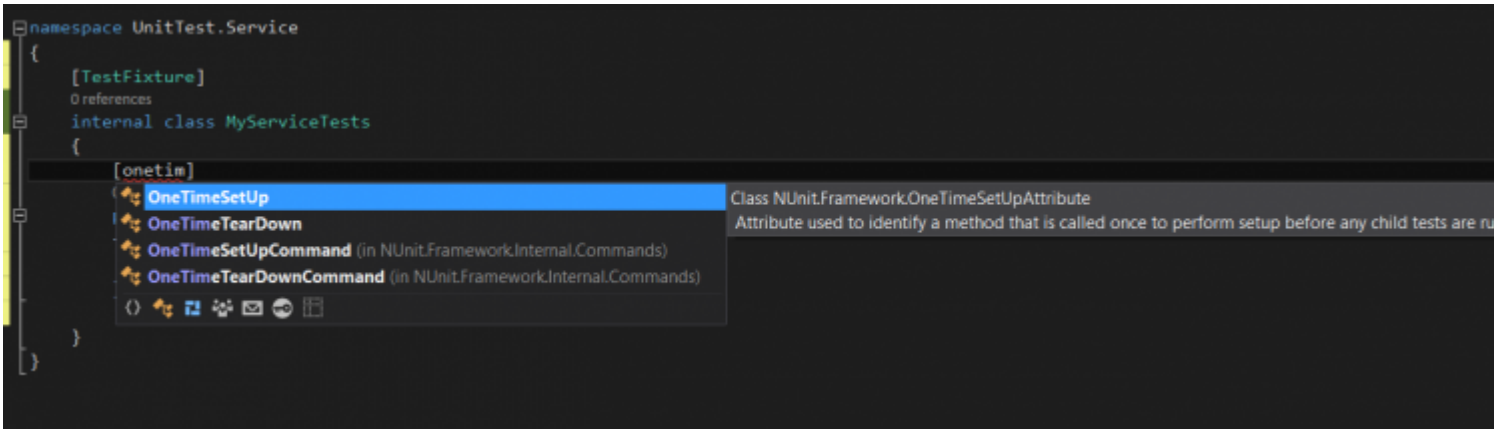
A test fixture marks a class as containing tests.

TestFixtureSetUp

This attribute used t identify a method that is called once to perform setup before any child tests are run. For the new versions we are using *OneTimeSetUp* as the TestFixtureSetUp is obsolete.



OneTimeSetUp




```
[OneTimeSetUp]
public void SetUp()
 {

 }
```

TearDown

This attribute is used to identify a method that is called immediately after each tests, it will be called even if there is any error, this is the place we can dispose our objects.

```
 [TearDown]
 public void CleanAfterEveryTest()
 {

 }
```

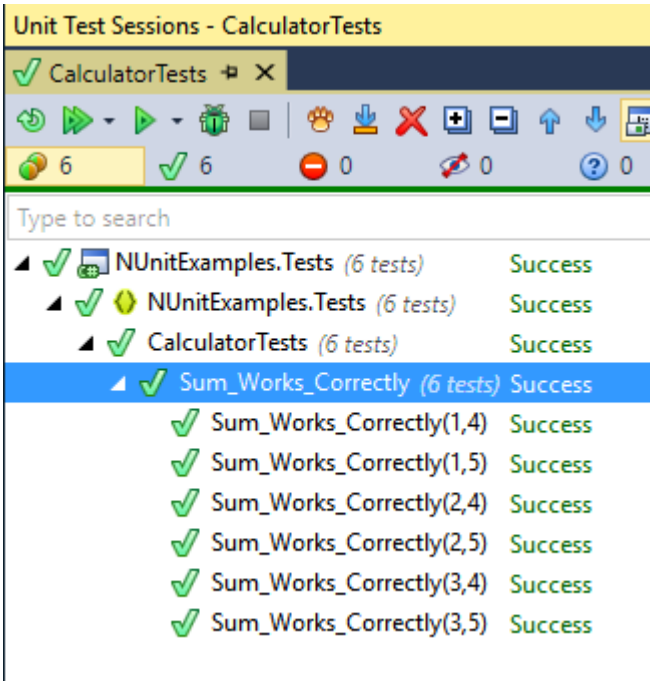
ValuesAttribute

The **ValuesAttribute** is used to specify *a set of values* for an individual parameter of a test

method with parameters.

```
[Test]
public void Sum_Works_Correctly(
    [Values(1, 2, 3)] int x,
    [Values(4, 5)] int y)
{
    // Arrange
    var calculator = new Calculator();

    // Act
    int result = calculator.Sum(x, y);

    // Assert
    Assert.That(result, Is.EqualTo(x + y));
}
```

Here we can see which test cases are run against these values:



Read Attributes online: https://riptutorial.com/nunit/topic/6512/attributes

Chapter 3: Fluent Assertions

Remarks

NUnit's `Assert.That()` form supports the use of constraints as its second parameter. All constraints provided out of the box by NUnit are available through the static classes `Is`, `Has` and `Does`. Constraints can be combined into fluent expressions using the built in methods `And`, `Or` and `With`. Expressions can be conveniently expanded up using the many methods in `ConstraintExpression`, such as `AtMost` and `Contains`.

Examples

Basic fluent assertion

```
Assert.That(actual, Is.EqualTo(expected));
```

Advanced Constraint Usage

Large fluent assertions do become harder to read, but when combined with classes that have good implementations of `ToString()`, they can generate very useful error messages.

```
[Test]
public void AdvancedContraintsGiveUsefulErrorMessages() {
    Assert.That(actualCollection, Has
        .Count.EqualTo(4)
        .And.Exactly(1).Property("Age").GreaterThan(60)
        .And.Some.Property("Address").Null
        .And.No.Property("Age").LessThanOrEqualTo(17));
}
```

On failure, this assertion generates messages like this:

```
Expected: property Count equal to 4 and exactly one item property Age greater
than 60 and some item property Address null and not property Age less than or
equal to 17
But was:  < <Steve Taylor (23) lives in Newcastle
>, <Michelle Yung (65) lives in San Francisco
>, <Ranjit Saraman (49) lives in Milano
>, <LaChelle Oppenheimer (16) lives in
> >
```

Collections

```
var a = new List<int> { 1, 2 };
var b = new List<int> { 2, 1 };

Assert.That (a, Is.EqualTo(b)); // fails
Assert.That (a, Is.EquivalentTo(b)); // succeeds
```

Read Fluent Assertions online: https://riptutorial.com/nunit/topic/2788/fluent-assertions

Chapter 4: Test execution and lifecycle

Examples

Executing tests in a given order

Normally your tests should be created in such a way that execution order is no concern. However there is always going to be an edge case were you need to break that rule.

The one scenario I came across was with R.NET whereby in a given process you can only initialize one R Engine and once disposed you cannot reinitialize. One of my test happened to deal with disposing the engine and if this test were to run before any other test(s) they would fail.

You will find below a code snippet of how I managed to get this to run in order using Nunit.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Reflection;
using NUnit.Framework;
using RSamples;

public class OrderedTestAttribute : Attribute
{
    public int Order { get; set; }

    public OrderedTestAttribute(int order)
    {
        this.Order = order;
    }
}

public class TestStructure
{
    public Action Test;
}

public class SampleTests
{
    [TearDown]
    public void CleanUpAfterTest()
    {
        REngineExecutionContext.ClearLog();
    }

    [OrderedTest(0)]
    public void Test1(){}

    [OrderedTest(1)]
    public void Test2(){}

    [OrderedTest(2)]
    public void Test3(){}

    [TestCaseSource(sourceName: "TestSource")]
```

```
    public void MainTest(TestStructure test)
    {
        test.Test();
    }

    public static IEnumerable<TestCaseData> TestSource
    {
        get
        {
            var assembly = Assembly.GetExecutingAssembly();
            Dictionary<int, List<MethodInfo>> methods = assembly
                .GetTypes()
                .SelectMany(x => x.GetMethods())
                .Where(y => y.GetCustomAttributes().OfType<OrderedTestAttribute>().Any())
                .GroupBy(z => z.GetCustomAttribute<OrderedTestAttribute>().Order)
                .ToDictionary(gdc => gdc.Key, gdc => gdc.ToList());

            foreach (var order in methods.Keys.OrderBy(x => x))
            {
                foreach (var methodInfo in methods[order])
                {
                    MethodInfo info = methodInfo;
                    yield return new TestCaseData(
                        new TestStructure
                        {
                            Test = () =>
                            {
                                object classInstance =
Activator.CreateInstance(info.DeclaringType, null);
                                info.Invoke(classInstance, null);
                            }
                        }).SetName(methodInfo.Name);
                }
            }
        }
    }
}
```

Read Test execution and lifecycle online: https://riptutorial.com/nunit/topic/2789/test-execution-and-lifecycle

Chapter 5: Write a custom constraint for the constraint model

Examples

Match an integer approximatively

Suppose we want to write a constraint which matches a number, but approximatively. Say, you are supposed to have `95` people in a survey, but `93` or `96` will do as well. We can write a custom constraint of the form:

```
public class AlmostEqualToConstraint : Constraint
{
    readonly int _expected;
    readonly double _expectedMin;
    readonly double _expectedMax;
    readonly int _percentageTolerance;


    public AlmostEqualToConstraint(int expected, int percentageTolerance)
    {
        _expected = expected;
        _expectedMin = expected * (1 - (double)percentageTolerance / 100);
        _expectedMax = expected * (1 + (double)percentageTolerance / 100);
        _percentageTolerance = percentageTolerance;
        Description = $"AlmostEqualTo {expected} with a tolerance of {percentageTolerance}%";
    }


    public override ConstraintResult ApplyTo<TActual>(TActual actual)
    {
        if (typeof(TActual) != typeof(int))
            return new ConstraintResult(this, actual, ConstraintStatus.Error);

        var actualInt = Convert.ToInt32(actual);

        if (_expectedMin <= actualInt && actualInt <= _expectedMax)
            return new ConstraintResult(this, actual, ConstraintStatus.Success);
        else
            return new ConstraintResult(this, actual, ConstraintStatus.Failure);
    }
}
```

Make new constraint usable in a fluent context

We're going to integrate the `AlmostEqualToConstraint` with the fluent NUnit interfaces, specifically the `Is` one. We'll need to extend the NUnit provided `Is` and use that throughout our code.

```
public class Is : NUnit.Framework.Is
{
    public static AlmostEqualToConstraint AlmostEqualTo(int expected, int percentageTolerance
```

```
= 5)
    {
        return new AlmostEqualToConstraint(expected, percentageTolerance);
    }
}
```

Read Write a custom constraint for the constraint model online: https://riptutorial.com/nunit/topic/9273/write-a-custom-constraint-for-the-constraint-model

Credits

S. No	Chapters	Contributors
1	Getting started with nunit	Community, forsvarir, kdtong, Old Fox, Pavel Yermalovich, Thulani Chivandikwa, Woodchipper
2	Attributes	kame, kdtong, Pavel Yermalovich, RJFalconer, Sibeesh Venu
3	Fluent Assertions	D.R., kdtong, mark_h, Paul Hicks
4	Test execution and lifecycle	forsvarir, Thulani Chivandikwa
5	Write a custom constraint for the constraint model	Horia Coman