1. **Nunit-Handson**

Follow the steps listed below to write the NUnit test cases for the application.

• Create a Unit test project(.Net Framework) in the solution provided.

• Add the CalcLibrary project as reference

• Create a class “CalculatorTests” to write all the test cases for the methods in the solution

• Use the ‘TestFixture’, ‘SetUp’ and ‘TearDown’ attributes, to declare, initialize and cleanup activities respectively

• Create a Test method to check the addition functionality

• Use the ‘TestCase’ attribute to send the inputs and the expected result

• Use Assert.That to check the actual and expected result match

**Solution:**

* I created the Unit test project file with the name “CalculatorTest” in Vs code with .NET CLI.

**dotnet new nunit -n CalculatorTests**

**cd CalculatorTests**

* And after that I added the CalcLibrary project as reference.

**dotnet add reference "C:\Users\mouli\OneDrive\Desktop\weekly learnings\week2\Parent project code\CalcLibrary\CalcLibrary\CalcLibrary.csproj"**

* Created a class “CalculatorTests” to write all the test cases for the methods in the solution.

using NUnit.Framework;

using CalcLibrary;

namespace CalculatorTests

{

[TestFixture]

public class CalculatorTests

{

private SimpleCalculator calculator;

[SetUp]

public void Init()

{

calculator = new SimpleCalculator();

}

[TearDown]

public void Cleanup()

{

calculator.AllClear();

}

[Test]

[TestCase(2, 3, 5)]

[TestCase(10, 15, 25)]

[TestCase(-5, 5, 0)]

public void Addition\_ReturnsCorrectResult(double a, double b, double expected)

{

double result = calculator.Addition(a, b);

Assert.That(result, Is.EqualTo(expected));

}

}

}

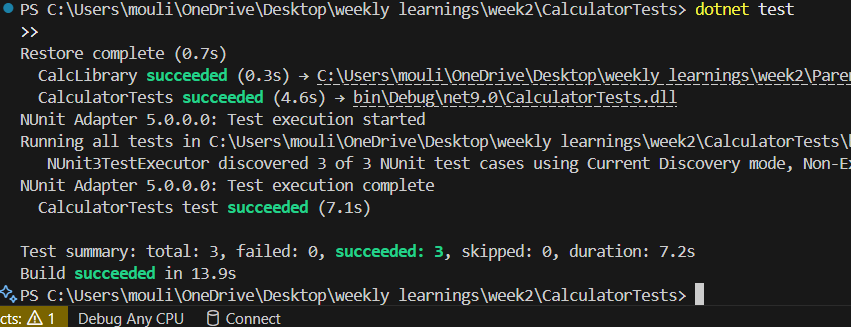
* After than I added the 3 Packages those are:

**PS C:\Users\mouli\OneDrive\Desktop\weekly learnings\week2\CalculatorTests> dotnet add package NUnit**

**>> dotnet add package NUnit3TestAdapter**

**>> dotnet add package Microsoft.NET.Test.Sdk**

**Result:**

****

1. **Nunit-Handson**

**Note: from this solution I used visual studios community 2022.**

Parameterized test cases

1. Create test case to verify the subtraction feature of the calculator with various input types.

• Create test cases with ‘TestCase’ attribute to send in input parameters and the expected result.

• Add more than 1 ‘TestCase’ attributes to check various combinations for subtractions.

• Use Assert.Equal to check the actual and expected results

2. Create a test case to verify the multiplication concepts of calculator

• Create test cases with ‘TestCase’ attribute to send in input parameters and the expected result.

• Add more than 1 ‘TestCase’ attributes to check various combinations for subtractions.

• Use Assert.Equal to check the actual and expected results

3. Create a test case to verify the division logic of the calculator

• Create test cases with ‘TestCase’ attribute to send in input parameters and the expected result.

• Add more than 1 ‘TestCase’ attributes to check various combinations for subtractions.

• Use Assert.Equal to check the actual and expected results

• In one of the inputs, provide the divisor value to be 0

• Use Try Catch block to catch the ArgumentException

• Use Assert.Fail to notify the user that the test case has failed. Give the message “Division by zero” in the Assert.Fail, which will be notified to the user. This message will be seen in the test explorer.

Test void methods

In the MathLibrary class there is a property “GetResult”. The result of every operation is stored in a variable ‘result’. This value is accessed by the property.

The class also has a method “AllClear” that sets the value of the result variable to 0.

• Create a test method ‘TestAddAndClear’

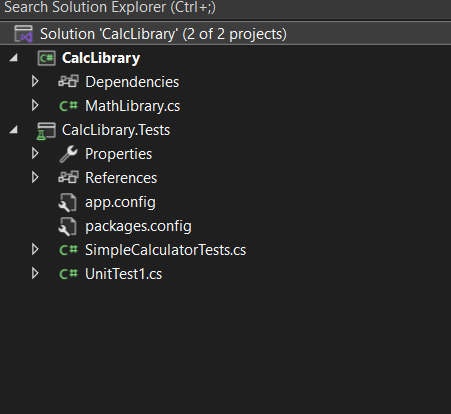
• Invoke the Addition method of the math class library

• Verify if the expected and Actual results match using Assert.AreEqual

• Invoke the ‘AllClear’ method

• Use Assert.AreEqual to check if the result is 0 or not

**Solution:**

****

using NUnit.Framework;

using CalcLibrary;

using System;

namespace CalcLibrary.Tests

{

[TestFixture]

public class SimpleCalculatorTests

{

private SimpleCalculator calculator;

[SetUp]

public void Setup()

{

calculator = new SimpleCalculator();

}

[TearDown]

public void Cleanup()

{

calculator.AllClear();

}

//Subtraction Test (Parameterized)

[TestCase(10, 5, 5)]

[TestCase(-5, -3, -2)]

[TestCase(5.5, 2.2, 3.3)]

[TestCase(0, 0, 0)]

public void Subtraction\_ParameterizedTests(double a, double b, double expected)

{

var result = calculator.Subtraction(a, b);

Assert.That(result, Is.EqualTo(expected));

}

//Multiplication Test (Parameterized)

[TestCase(2, 3, 6)]

[TestCase(-2, -2, 4)]

[TestCase(0, 100, 0)]

[TestCase(1.5, 2.0, 3.0)]

public void Multiplication\_ParameterizedTests(double a, double b, double expected)

{

var result = calculator.Multiplication(a, b);

Assert.That(result, Is.EqualTo(expected));

}

//Division Test (Parameterized + Exception Case)

[TestCase(10, 2, 5)]

[TestCase(9, 3, 3)]

[TestCase(7.5, 2.5, 3.0)]

public void Division\_ParameterizedTests(double a, double b, double expected)

{

var result = calculator.Division(a, b);

Assert.That(result, Is.EqualTo(expected));

}

//Division By Zero Test

[Test]

public void Division\_ByZero\_ShouldThrowArgumentException()

{

try

{

calculator.Division(5, 0);

Assert.Fail("Division by zero"); //This will show up in Test Explorer if exception is not thrown

}

catch (ArgumentException ex)

{

Assert.That(ex.Message, Is.EqualTo("Second Parameter Can't be Zero"));

}

}

//Test Void Method: AllClear() and GetResult Property

[Test]

public void TestAddAndClear()

{

double result = calculator.Addition(12, 8);

Assert.That(calculator.GetResult, Is.EqualTo(20), "Addition did not give expected result.");

calculator.AllClear(); //Call the void method

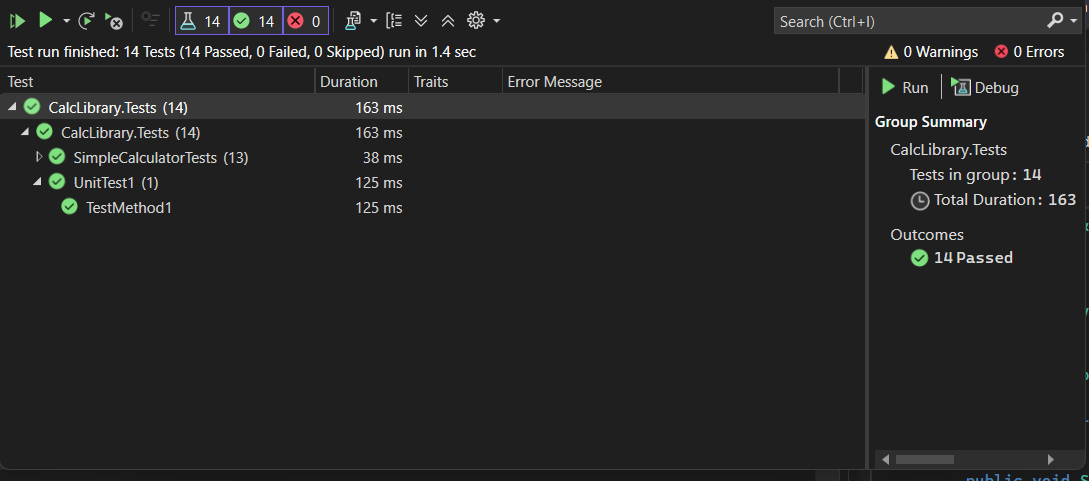
Assert.That(calculator.GetResult, Is.EqualTo(0), "AllClear did not reset the result to 0.");

}

}

}

**RESULT:**

****

1. **Nunit-Handson**

Create a unit test project using NUnit for the given UtilLib project. Click here to download the source project.

The functionality is called ParseHostName which is defined in the UrlHostNameParser class. It parses the host name from the URL using certain logic. Write all possible test methods for the given functionality to make sure that it returns the expected result under various circumstances.

Since the function has two execution paths, you need to write at least two test methods.

Recommendations:

Test Project Name: <ClassLib\_Project>.Tests

Test Class Name: <SUT>Tests

Test Method Name: UnitUnderTest\_Scenario\_ExpectedOutcome

Note:

• Enforce the Single Assertion Rule

• Use Assert.That()

Steps to perform

1) Create a Class Library project in the same solution which is provided and name it as suggested.

2) Rename the class file name (<SUT>Tests.cs).

3) Add the assembly reference of the UtilLib project to the test project.

4) Additionally add the reference of both NUnit and NUnit3TestAdapter in the test project using NuGet Package Manager (NPM).

5) Write the suggested test methods.

6) Run your tests.

7) Break the test by modifying the source project functionality.

8) Rerun the test.

9) Observe the test result.

**Solution:**

using NUnit.Framework;

using UtilLib;

using System;

namespace UtilLib.Tests

{

[TestFixture]

public class UrlHostNameParserTests

{

private UrlHostNameParser parser;

[SetUp]

public void Setup()

{

parser = new UrlHostNameParser();

}

//Test: Valid HTTP URL

[Test]

public void ParseHostName\_HttpUrl\_ReturnsHostName()

{

string url = "http://www.example.com/page";

string expected = "www.example.com";

string result = parser.ParseHostName(url);

Assert.That(result, Is.EqualTo(expected));

}

//Test: Valid Https url

[Test]

public void ParseHostName\_HttpsUrl\_ReturnsHostName()

{

string url = "https://mail.google.com/inbox";

string expected = "mail.google.com";

string result = parser.ParseHostName(url);

Assert.That(result, Is.EqualTo(expected));

}

//Test: Invalid protocol (ftp)

[Test]

public void ParseHostName\_FtpUrl\_ThrowsFormatException()

{

string url = "ftp://files.server.com/resource";

var ex = Assert.Throws<FormatException>(() => parser.ParseHostName(url));

Assert.That(ex.Message, Is.EqualTo("Url is not in correct format"));

}

//Test: Garbage string (no protocol)

[Test]

public void ParseHostName\_NoProtocol\_ThrowsFormatException()

{

string url = "example.com/page";

var ex = Assert.Throws<FormatException>(() => parser.ParseHostName(url));

Assert.That(ex.Message, Is.EqualTo("Url is not in correct format"));

}

//Optional: Edge case - URL with port number

[Test]

public void ParseHostName\_UrlWithPort\_IgnoresPortNumber()

{

string url = "https://www.example.com:8080/page";

string expected = "www.example.com"; //You may change logic to handle this if needed

string result = parser.ParseHostName(url);

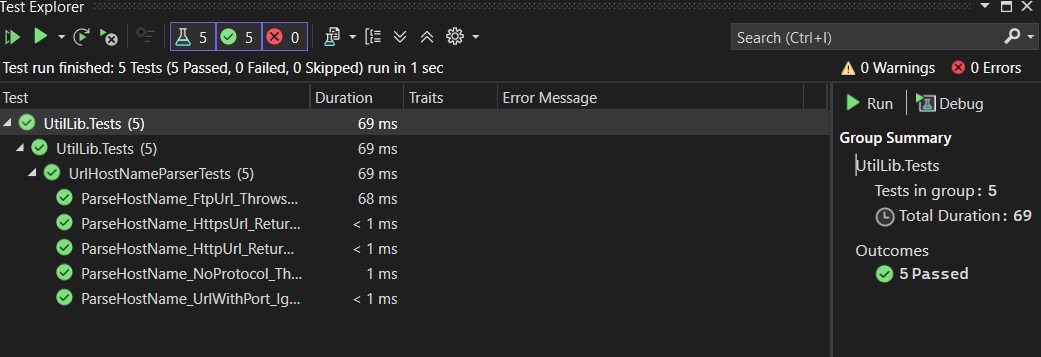
Assert.That(result, Is.EqualTo(expected));

}

}

}

**Result:**

****