**Hands-On: Stage 3 - NUnit Day 1 - Handson 1**

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

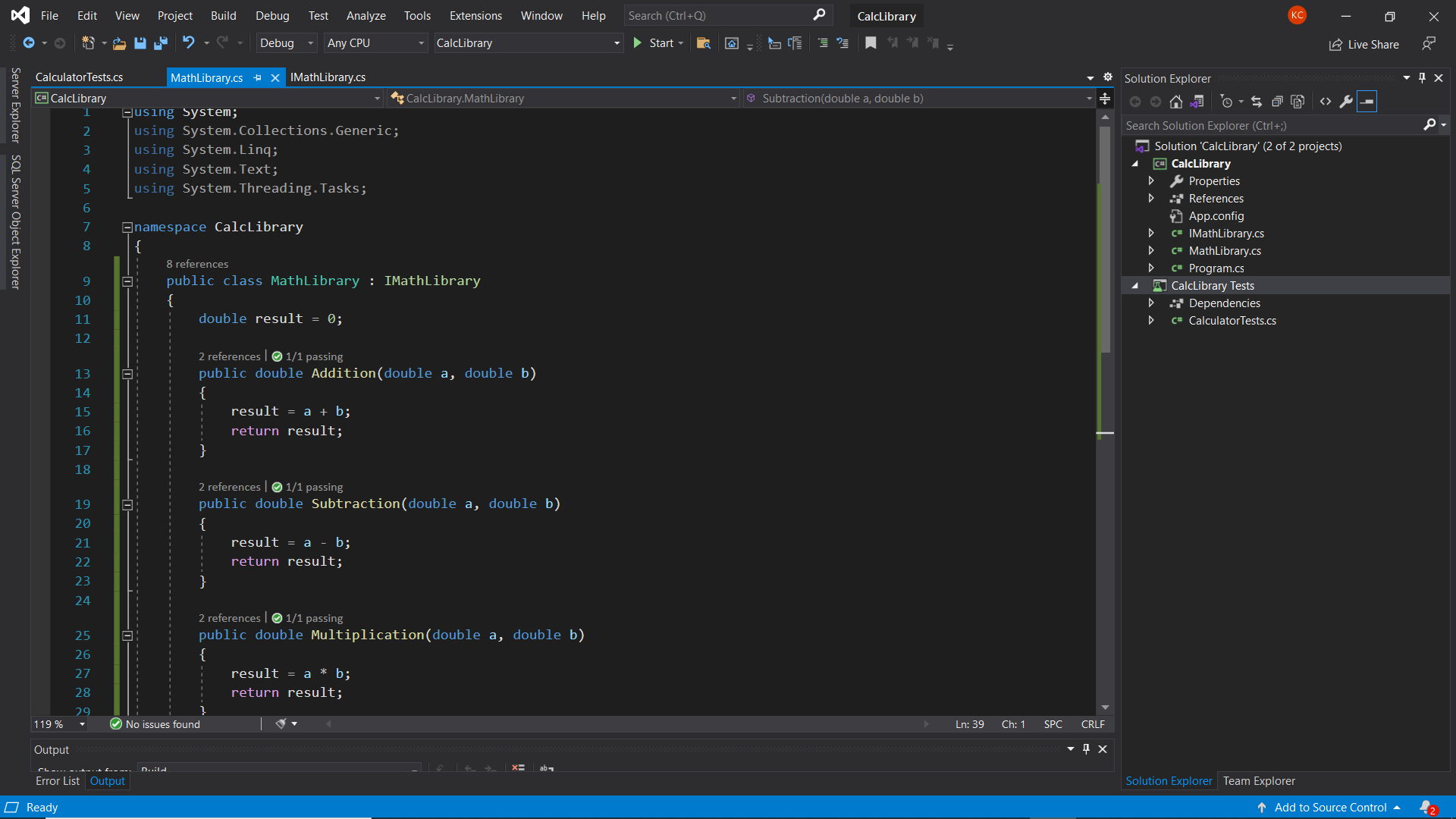
· Create a class “CalculatorTests” to write all the test cases for the methods in the solution

· 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

**PROJECT CODE:**

**IMPLEMENTATION:**

**MathLibrary.cs**

using System;

namespace CalcLibrary

{

    public class MathLibrary : IMathLibrary

    {

        double result = 0;

        public double Addition(double a, double b)

        {

            result = a + b;

            return result;

        }

        public double Subtraction(double a, double b)

        {

            result = a - b;

            return result;

        }

        public double Multiplication(double a, double b)

        {

            result = a \* b;

            return result;

        }

        public double Division(double a, double b)

        {

            if (b == 0)

                throw new ArgumentException("Second Parameter Can't be Zero");

            result = a / b;

            return result;

        }

    }

}

**IMathLibrary.cs**

namespace CalcLibrary

{

    interface IMathLibrary

    {

        double Addition(double a, double b);

        double Subtraction(double a, double b);

        double Multiplication(double a, double b);

        double Division(double a, double b);

    }

}

**CalcLibraryTests.cs**

using CalcLibrary;

using NUnit.Framework;

namespace CalcLibrary\_Tests

{

    [TestFixture]

    public class CalculatorTests

    {

        [Test]

        public void Addition\_InputTwoDoubleValues\_ReturnsSum()

        {

            MathLibrary mathLibrary = new MathLibrary();

            //Arrange

            double expectedResult = 4;

            //Act

            double Result = mathLibrary.Addition(2, 2);

            //Assert

            Assert.AreEqual(expectedResult, Result);

        }

        [Test]

        public void Subtraction\_InputTwoDoubleValues\_ReturnsDifference()

        {

            MathLibrary mathLibrary = new MathLibrary();

            double expectedResult = 5;

            double Result = mathLibrary.Subtraction(10, 5);

            Assert.AreEqual(expectedResult, Result);

        }

        [Test]

        public void Multiplication\_InputTwoDoubleValues\_ReturnsProduct()

        {

            MathLibrary mathLibrary = new MathLibrary();

            double expectedResult = 16;

            double Result = mathLibrary.Multiplication(8, 2);

            Assert.AreEqual(expectedResult, Result);

        }

        [Test]

        public void Division\_InputTwoDoubleValues\_ReturnsQuotient()

        {

            MathLibrary mathLibrary = new MathLibrary();

            double expectedResult = 10;

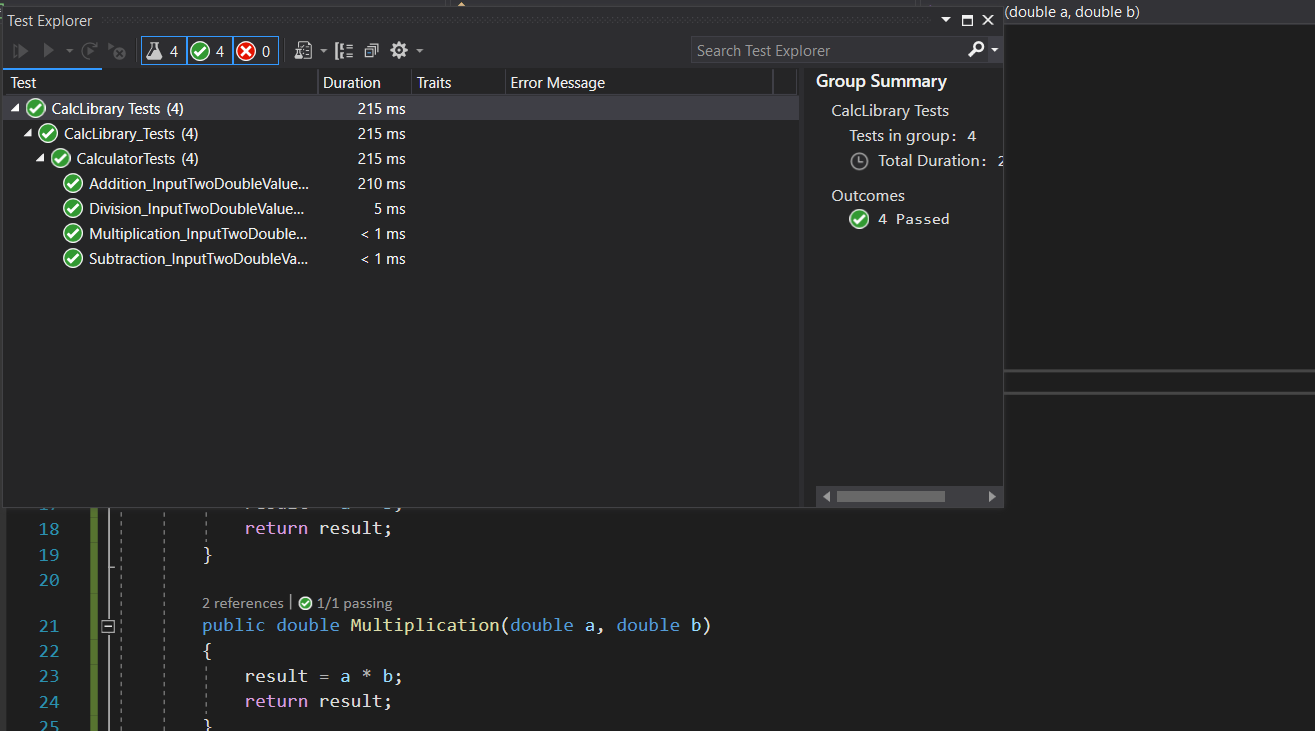
            double Result = mathLibrary.Division(20, 2);

            Assert.AreEqual(expectedResult, Result);

        }

    }

}

**OUTPUT:**

**Hands-On: Stage 3 - NUnit Day 1 – Handson 2**

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’

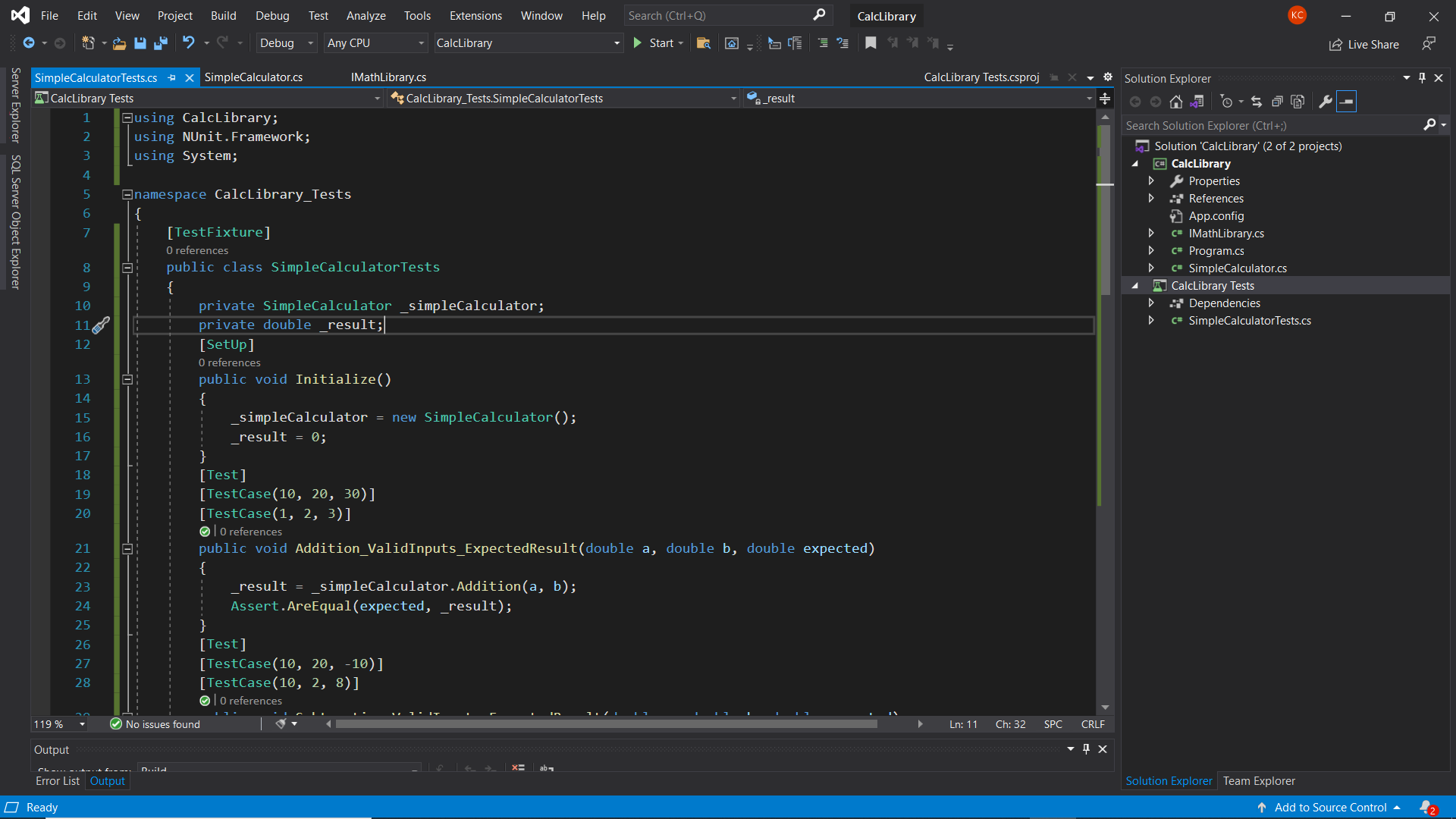
· Invoke the Addition method of the math class library

· 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

**PROJECT CODE:**



**IMPLEMENTATION:**

**SimpleCalculator.cs**

using System;

namespace CalcLibrary

{

    public class SimpleCalculator : IMathLibrary

    {

        double result = 0;

        public double Addition(double a, double b)

        {

            result = a + b;

            return result;

        }

        public double Subtraction(double a, double b)

        {

            result = a - b;

            return result;

        }

        public double Multiplication(double a, double b)

        {

            result = a \* b;

            return result;

        }

        public double Division(double a, double b)

        {

            if (b == 0)

                throw new ArgumentException("Second Parameter Can't be Zero");

            result = a / b;

            return result;

        }

        public void AllClear()

        {

            result = 0;

        }

        public double GetResult

        {

            get { return result; }

        }

    }

}

**IMathLibrary.cs**

namespace CalcLibrary

{

    interface IMathLibrary

    {

        double Addition(double a, double b);

        double Subtraction(double a, double b);

        double Multiplication(double a, double b);

        double Division(double a, double b);

    }

}

**SimpleCalculatorTests.cs**

using CalcLibrary;

using NUnit.Framework;

using System;

namespace CalcLibrary\_Tests

{

    [TestFixture]

    public class SimpleCalculatorTests

    {

        private SimpleCalculator \_simpleCalculator;

        private double \_result;

        [SetUp]

        public void Initialize()

        {

            \_simpleCalculator = new SimpleCalculator();

            \_result = 0;

        }

        [Test]

        [TestCase(10, 20, 30)]

        [TestCase(1, 2, 3)]

        public void Addition\_ValidInputs\_ExpectedResult(double a, double b, double expected)

        {

            \_result = \_simpleCalculator.Addition(a, b);

            Assert.AreEqual(expected, \_result);

        }

        [Test]

        [TestCase(10, 20, -10)]

        [TestCase(10, 2, 8)]

        public void Subtraction\_ValidInputs\_ExpectedResult(double a, double b, double expected)

        {

            \_result = \_simpleCalculator.Subtraction(a, b);

            Assert.AreEqual(expected, \_result);

        }

        [Test]

        [TestCase(10, 20, 200)]

        [TestCase(10, 0, 0)]

        public void Multiplication\_ValidInputs\_ExpectedResult(double a, double b, double expected)

        {

            \_result = \_simpleCalculator.Multiplication(a, b);

            Assert.AreEqual(expected, \_result);

        }

        [Test]

        [TestCase(10, 2, 5)]

        public void Division\_ValidInputs\_ExpectedResult(double a, double b, double expected)

        {

            \_result = \_simpleCalculator.Division(a, b);

            Assert.AreEqual(expected, \_result);

        }

        [Test]

        [TestCase(10, 0)]

        public void Division\_InValidInputs\_ExpectedException(double a, double b)

        {

            Assert.Throws<ArgumentException>(() => \_simpleCalculator.Division(a, b));

        }

        [TearDown]

        public void CleanUp()

        {

            \_simpleCalculator.AllClear();

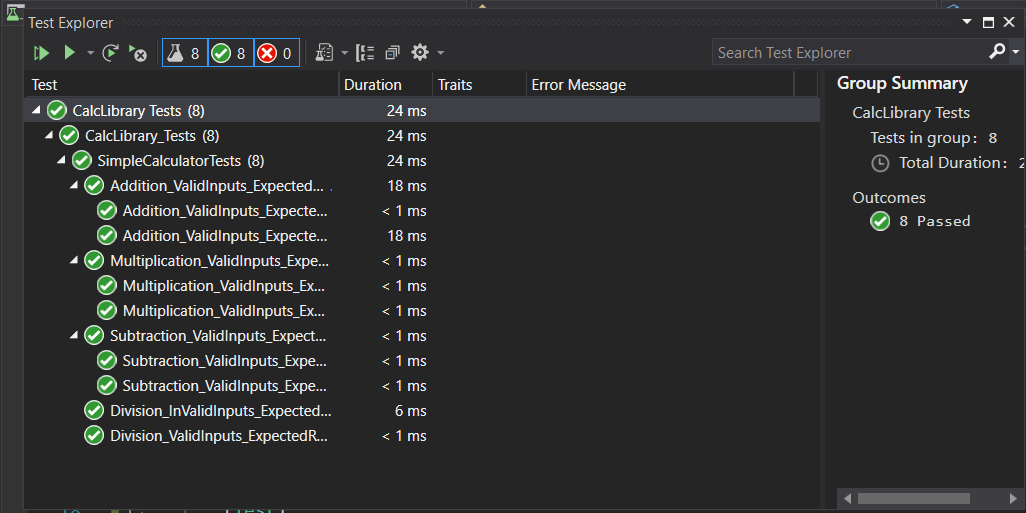
            \_result = \_simpleCalculator.GetResult;

        }

    }

}

**OUTPUT:**



**Hands-On: Stage 3 - NUnit Day 1 – Handson 3**

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).

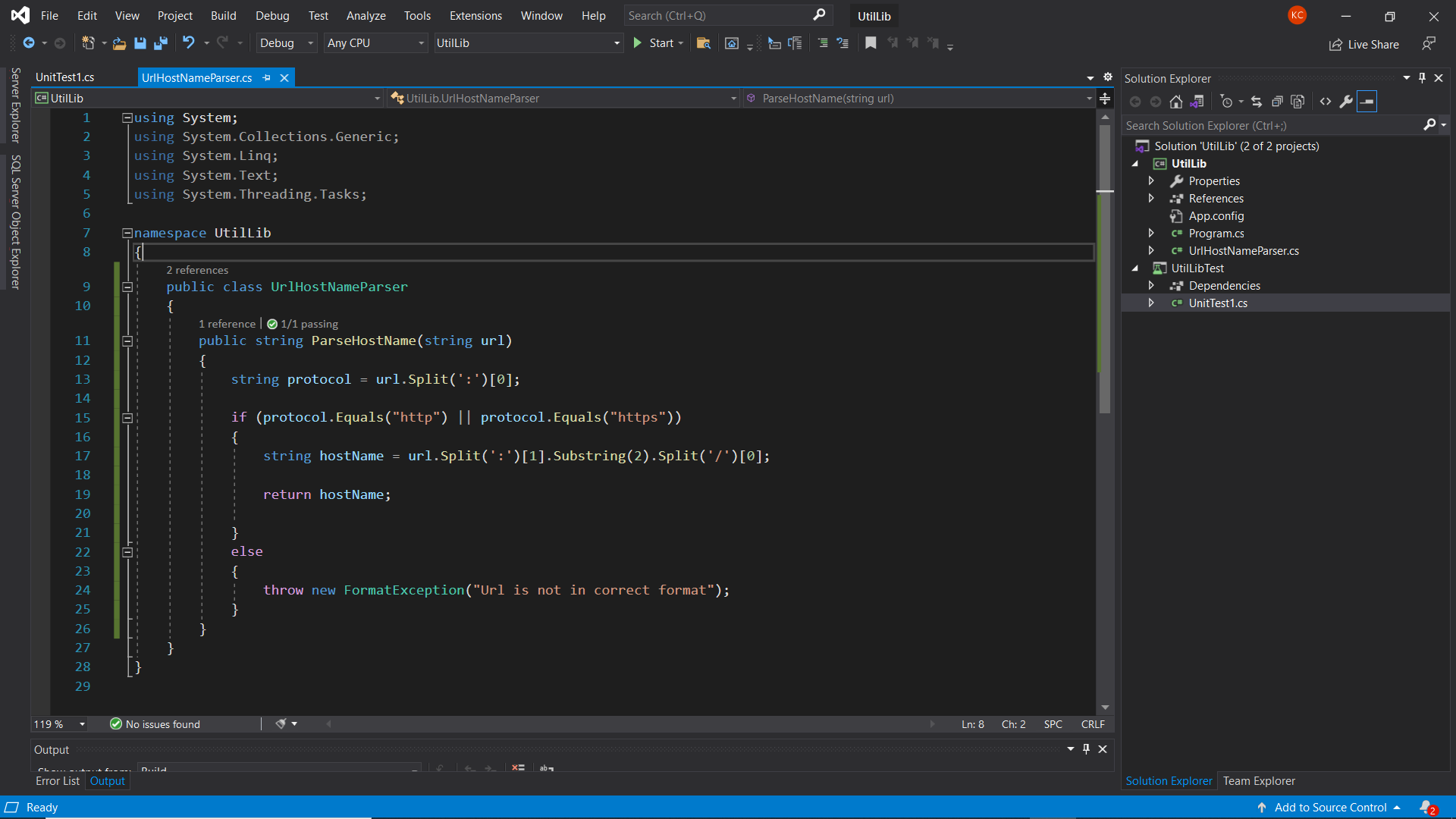
5) Write the suggested test methods.

6) Run your tests.

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

8) Rerun the test.

9) Observe the test result.

**PROJECT CODE:**

**IMPLEMENTATION:**

**UrlHostNameParser.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace UtilLib

{

    public class UrlHostNameParser

    {

        public string ParseHostName(string url)

        {

            string protocol = url.Split(':')[0];

            if (protocol.Equals("http") || protocol.Equals("https"))

            {

                string hostName = url.Split(':')[1].Substring(2).Split('/')[0];

                return hostName;

            }

            else

            {

                throw new FormatException("Url is not in correct format");

            }

        }

    }

}

**SUT.cs**

using NUnit.Framework;

using UtilLib;

namespace UnitTestProject1

{

    [TestFixture]

    public class SUT

    {

        [Test]

        public void UnitUnderTest\_Scenario\_ExpectedOutcome()

        {

            UrlHostNameParser uln = new UrlHostNameParser();

            string result = uln.ParseHostName("https://www.tutorialpoint.com");

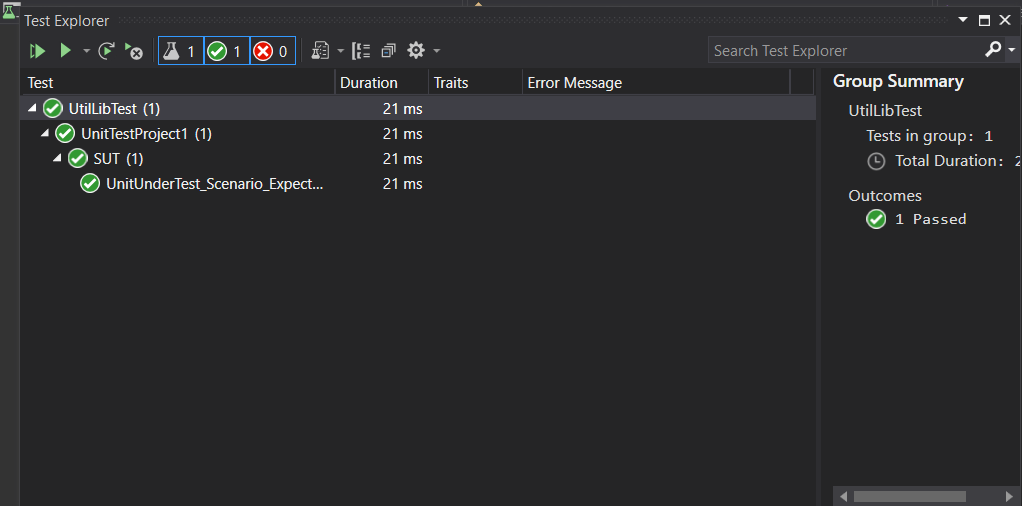
            Assert.That(result, Is.EqualTo("www.tutorialpoint.com"));

        }

    }

}

**OUTPUT**



**Hands-On: Stage 3 - NUnit Day 1 – Handson 4**

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).

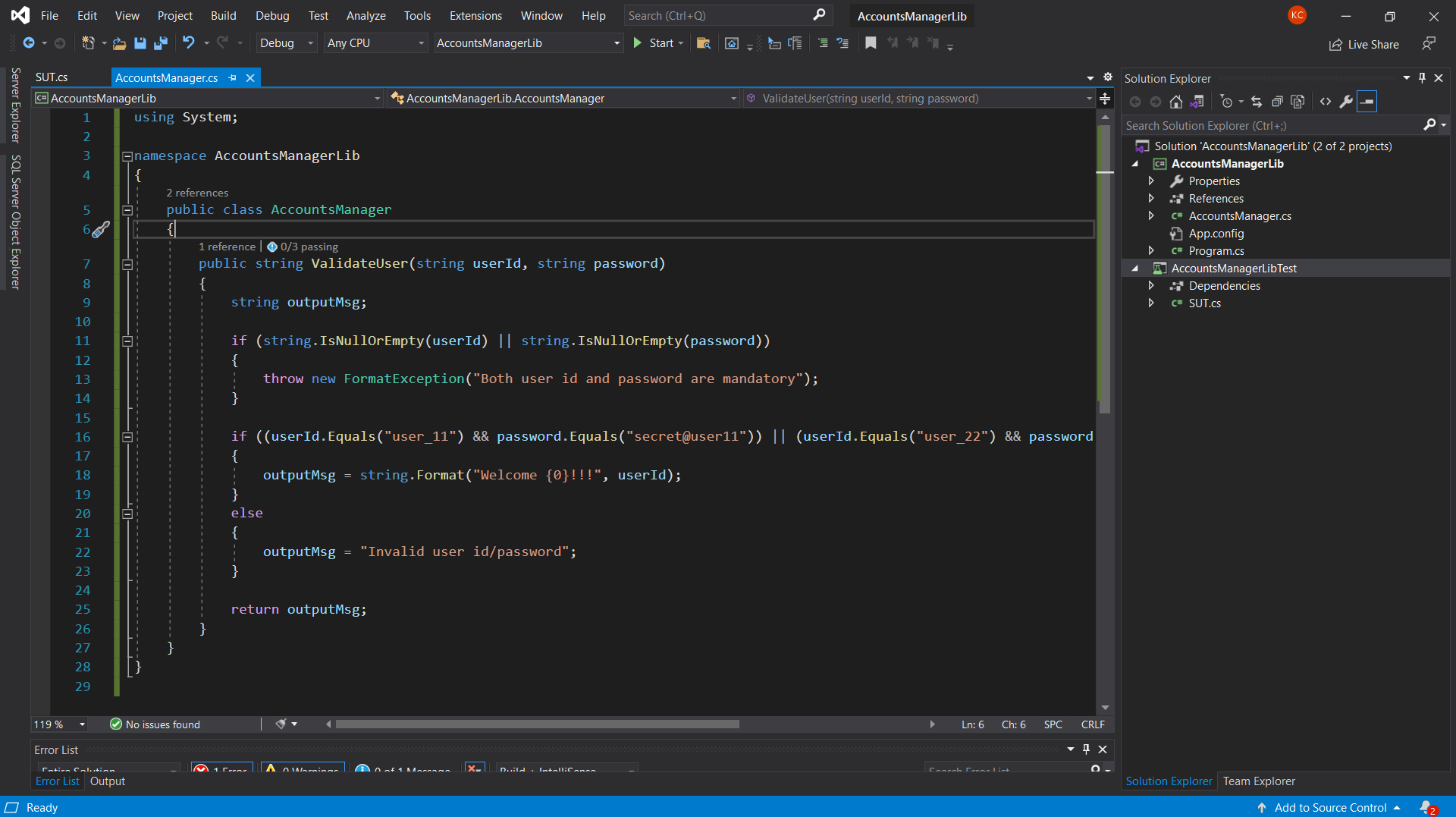
5) Write the suggested test methods.

6) Run your tests.

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

8) Rerun the test.

9) Observe the test result.

**PROJECT CODE:**

**IMPLEMENTATION:**

**AccountsManager.cs**

using System;

namespace AccountsManagerLib

{

    public class AccountsManager

    {

        public string ValidateUser(string userId, string password)

        {

            string outputMsg;

            if (string.IsNullOrEmpty(userId) || string.IsNullOrEmpty(password))

            {

                throw new FormatException("Both user id and password are mandatory");

            }

            if ((userId.Equals("user\_11") && password.Equals("secret@user11")) || (userId.Equals("user\_22") && password.Equals("secret@user22")))

            {

                outputMsg = string.Format("Welcome {0}!!!", userId);

            }

            else

            {

                outputMsg = "Invalid user id/password";

            }

            return outputMsg;

        }

    }

}

**SUT.cs**

using NUnit.Framework;

using AccountsManagerLib;

namespace AccountsManagerLibTest

{

    [TestFixture]

    public class SUT

    {

        [Test]

        [TestCase("user\_11", "secret@user11", "Welcome user\_11!!!")]

        [TestCase("user\_22", "secret@user22", "Welcome user\_22!!!")]

        [TestCase("user\_11", "secret@user22", "Invalid user id/password")]

        public void UnitUnderTest\_Scenario\_ExpectedOutcome1(string id, string password, string message)

        {

            AccountsManager ac = new AccountsManager();

            string result = ac.ValidateUser(id, password);

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

        }

    }

**OUTPUT**

