**DAY 1**

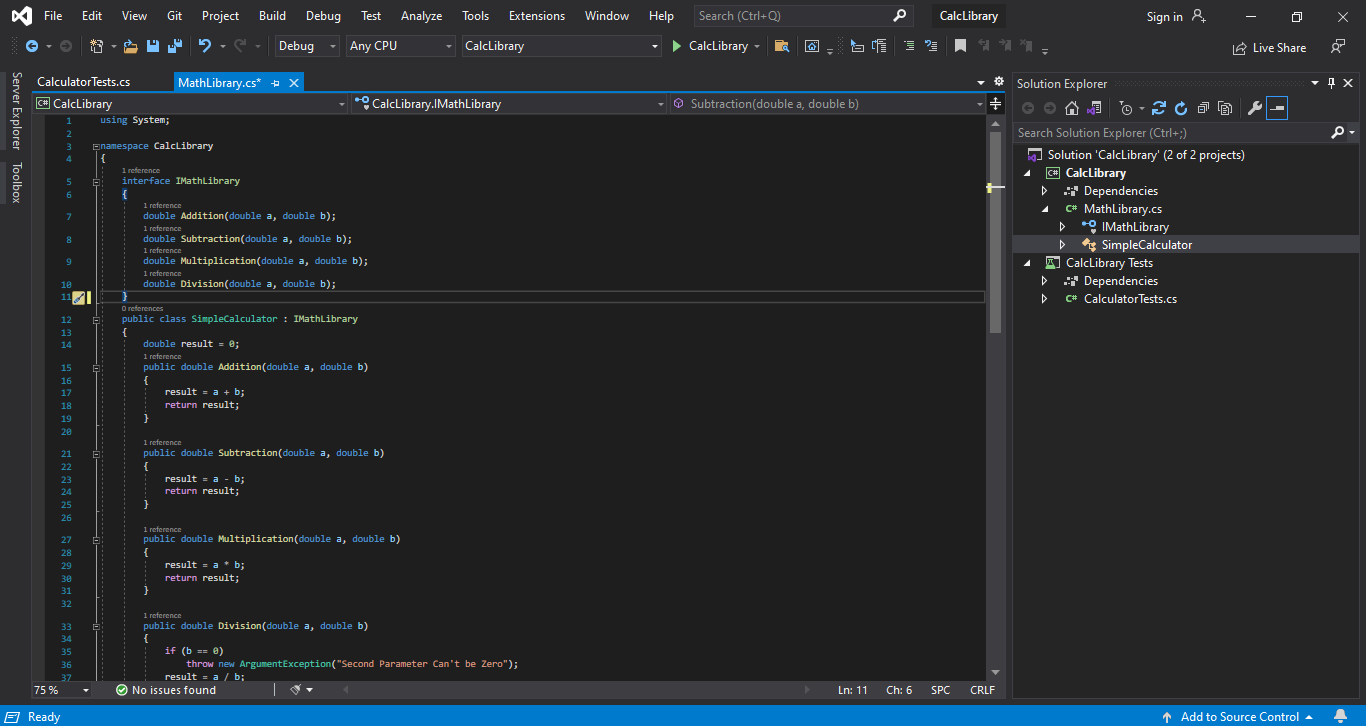
**NUNIT HANDSON**

**Hands On 1:**

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.

**Implementation**

****

**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;

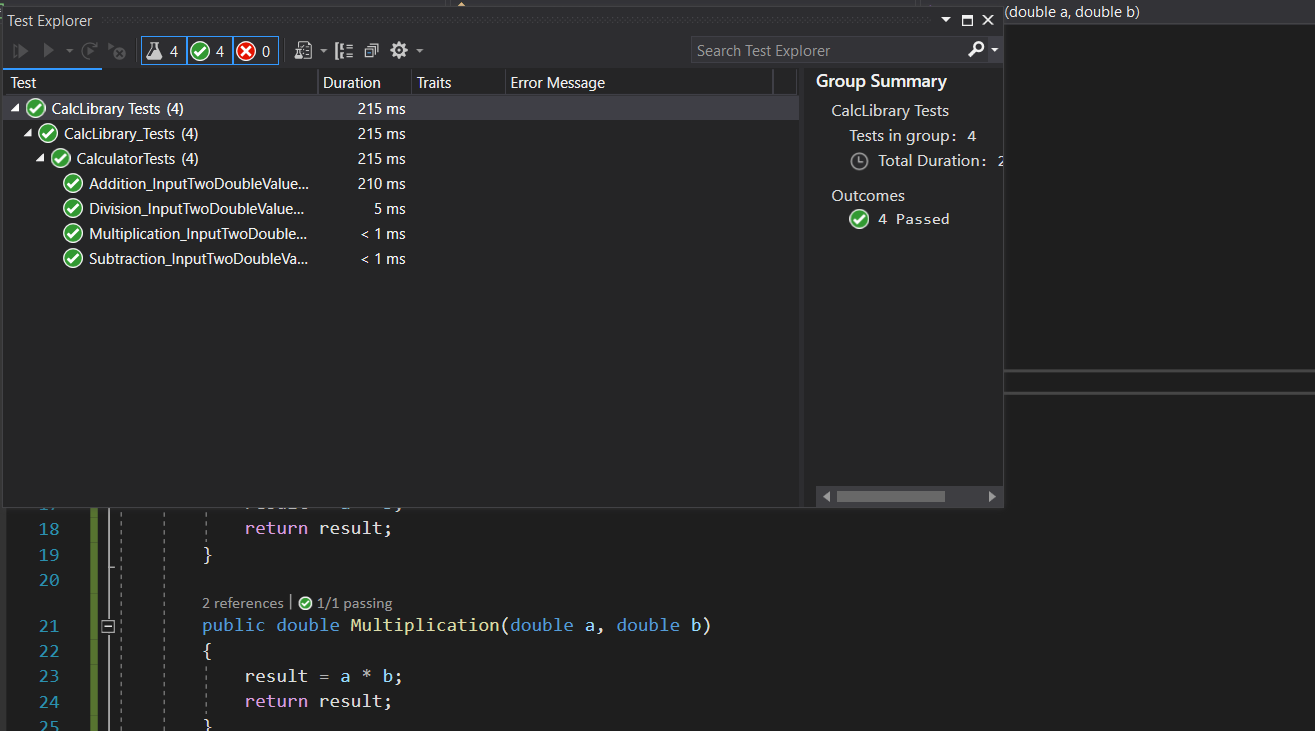
double Result = mathLibrary.Division(20, 2);

Assert.AreEqual(expectedResult, Result);

}

}

}

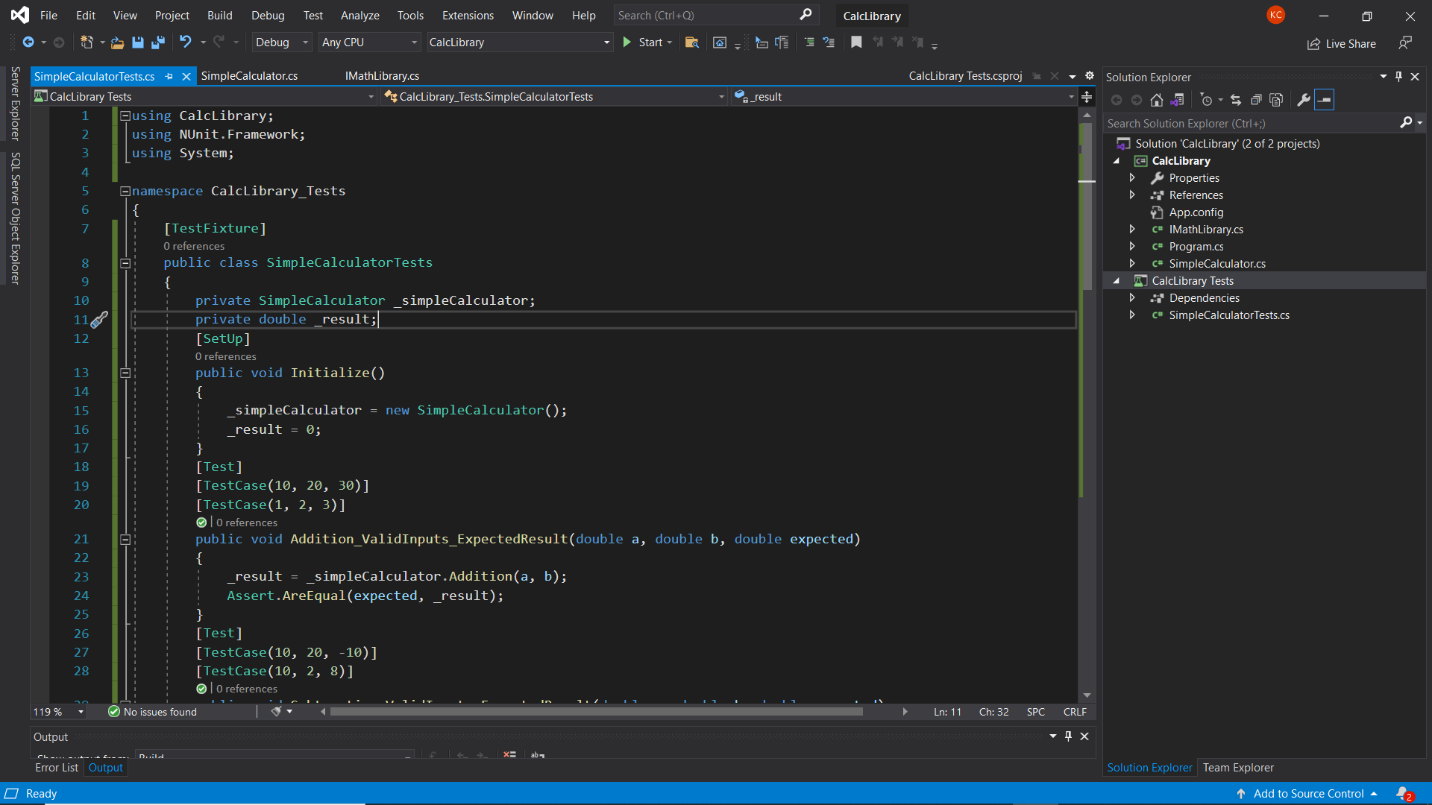
**Output**

**Hands On 2:**

Steps listed below to write the NUnit test cases for the application.

* 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 resultsCreate 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.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.

**Implementation**

****

**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();

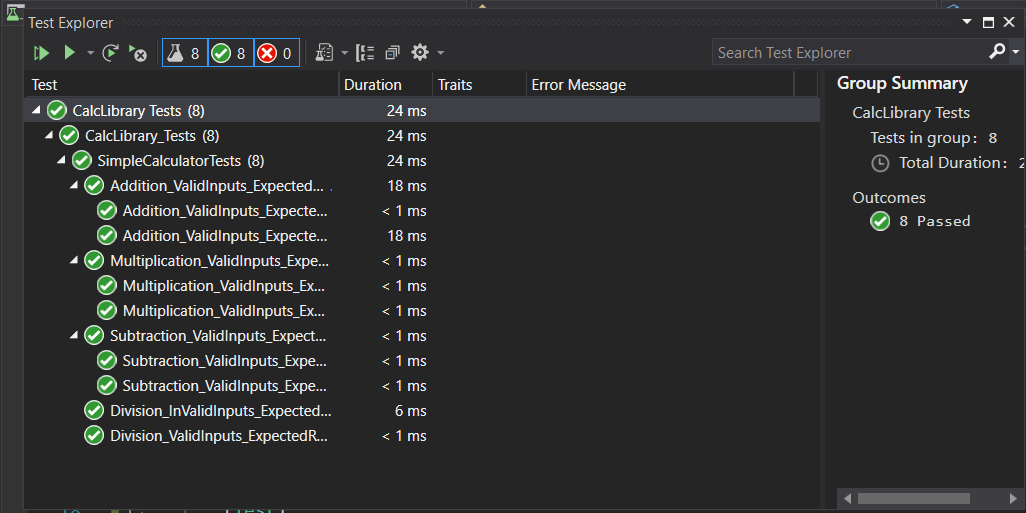
result = simpleCalculator.GetResult;

}

}

}

**Output**

****

**Hands On 3:**

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

* Create a Class Library project in the same solution which is provided and name it as suggested.
* Rename the class file name (<SUT>Tests.cs).Add the assembly reference of the UtilLib project to the test project.
* Additionally, add the reference of both NUnit and NUnit3TestAdapter in the test project using NuGet Package Manager (NPM).
* Write the suggested test methods.Run your tests.
* Break the test by modifying the source project functionality.
* Rerun the test. Observe the test result

**Implementation**

using Microsoft.VisualStudio.TestTools.UnitTesting;

using System;

using NUnit.Framework;

using nunithandson3;

using Assert = NUnit.Framework.Assert;

namespace UnitTestProject3

{

[TestFixture]

public class UnitTest1 {

UrlHostNameParser u;

[SetUp]

public void SetUp()

{

u = new UrlHostNameParser();

}

[TearDown]

public void Dispose()

{

u = null;

}

[TestCase]

public void Testcase1()

{

string act = u.ParseHostName("https://Facebook.com");

string exp = "Facebook.com";

Assert.That(act, Is.EqualTo(exp));

}

[TestCase]

public void Testcase2()

{

string act = u.ParseHostName("http://twitter.com");

string exp = "twitter.com";

Assert.That(act, Is.EqualTo(exp));

}

[TestCase]

public void Testcase3()

{

var ex = Assert.Throws<FormatException>(() => u.ParseHostName("https12://gmail.com"));

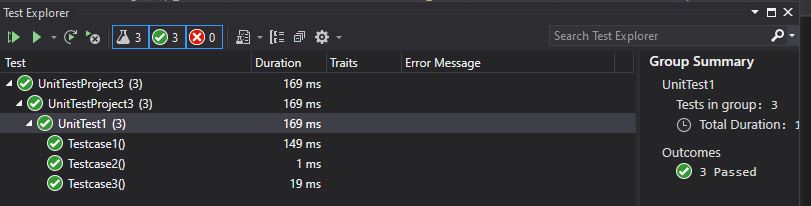
Assert.That(ex.Message, Is.EqualTo("Url is not in correct format"));

}

}

}

**Output**

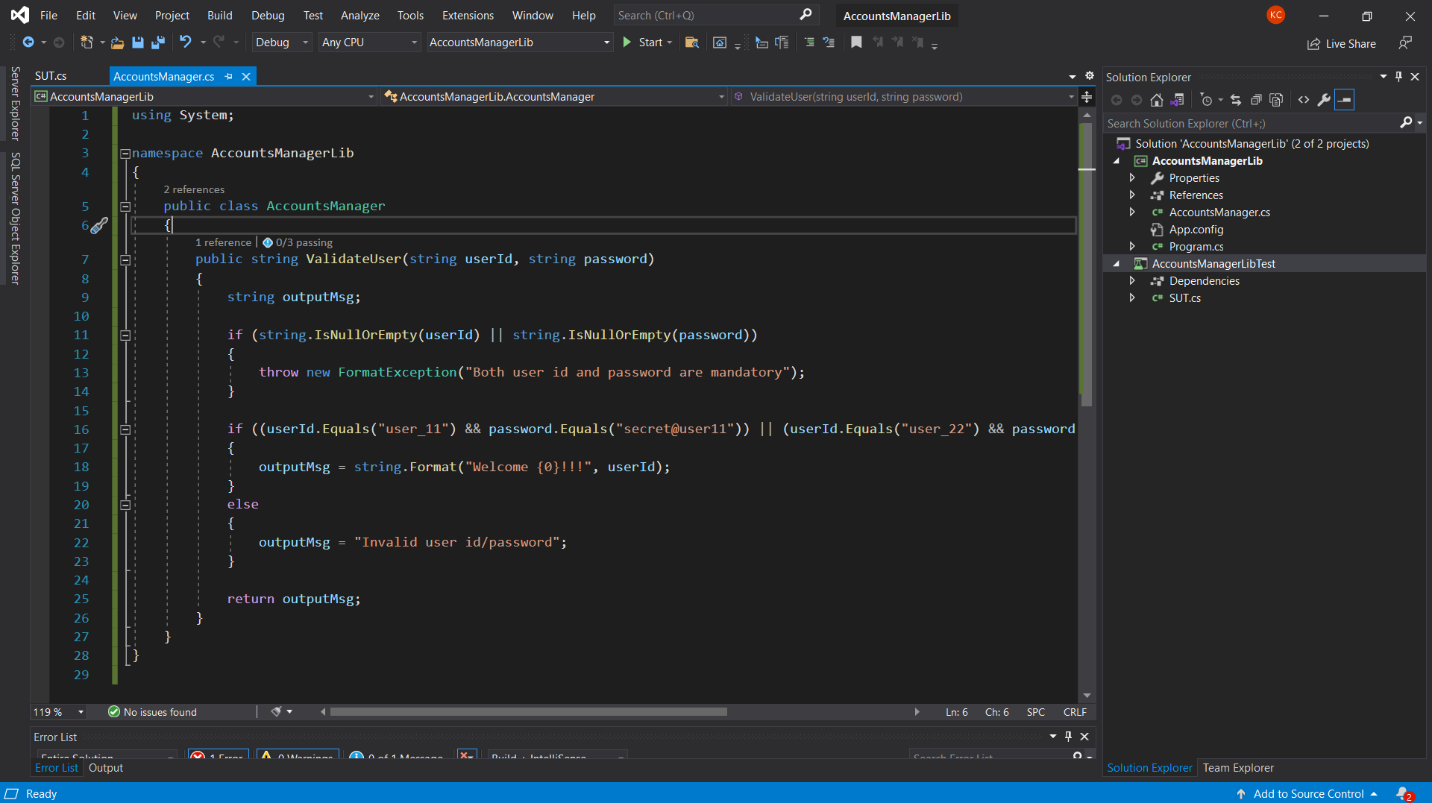
****

**Hands On 4:**

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

* Create a Class Library project in the same solution which is provided and name it as suggested.
* Rename the class file name (<SUT>Tests.cs). Add the assembly reference of the UtilLib project to the test project.
* Additionally add the reference of both NUnit and NUnit3TestAdapter in the test project using NuGet Package Manager (NPM).Write the suggested test methods.
* Run your tests.Break the test by modifying the source project functionality.
* Rerun the test.Observe the test result.

**Implementation**



using Microsoft.VisualStudio.TestTools.UnitTesting;

using System;

using NUnit.Framework;

using nunithandson3;

using Assert = NUnit.Framework.Assert;

namespace UnitTestProject3

{

[TestFixture]

public class UnitTest1 {

UrlHostNameParser u;

[SetUp]

public void SetUp()

{

u = new UrlHostNameParser();

}

[TearDown]

public void Dispose()

{

u = null;

}

[TestCase]

public void LoginTestcase1()

{

string exp = "Invalid user id/password";

string act = u.ValidateUser("user\_22", "secret@user12s");

Assert.That(act, Is.EqualTo(exp));

}

[TestCase]

public void LoginTestcase2()

{

string act = u.ValidateUser("user\_11", "secret@user11");

string exp = string.Format("Invalid user id/password");

Assert.That(act, Is.EqualTo(exp));

}

[TestCase]

public void LoginTestcase3()

{

string act = u.ValidateUser("user\_12", "secret@user12");

string exp = string.Format("Welcome user\_12!!!");

Assert.That(act, Is.EqualTo(exp));

}

[TestCase]

public void LoginTestcase4()

{

string exp = "Invalid user id/password";

string act = u.ValidateUser("user\_23", "secret@user12s");

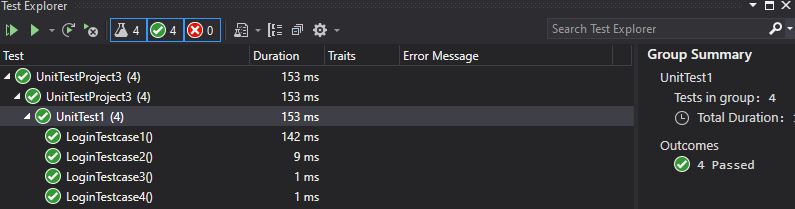
Assert.That(act, Is.EqualTo(exp));

}

}

}

**Output**

****