**1. NUnit-Handson**

**TestFixture & Test**

**SimpleCalculatorTests.cs**

using NUnit.Framework;

using CalcLibrary;

using System;

namespace CalcTests

{

[TestFixture]

public class SimpleCalculatorTests

{

private SimpleCalculator \_calc;

[SetUp]

public void Setup()

{

\_calc = new SimpleCalculator();

}

[TearDown]

public void Teardown()

{

\_calc.AllClear();

}

[TestCase(2, 3, 5)]

[TestCase(-1, 1, 0)]

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

{

var result = \_calc.Addition(a, b);

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

}

[TestCase(5, 3, 2)]

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

{

var result = \_calc.Subtraction(a, b);

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

}

[TestCase(4, 2, 8)]

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

{

var result = \_calc.Multiplication(a, b);

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

}

[TestCase(10, 2, 5)]

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

{

var result = \_calc.Division(a, b);

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

}

[Test]

public void Division\_ByZero\_ThrowsException()

{

var ex = Assert.Throws<ArgumentException>(() => \_calc.Division(5, 0));

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

}

[Test]

public void GetResult\_ReturnsCorrectValueAfterOperation()

{

\_calc.Addition(2, 3);

Assert.That(\_calc.GetResult, Is.EqualTo(5));

}

[Test]

public void AllClear\_SetsResultToZero()

{

\_calc.Addition(10, 20);

\_calc.AllClear();

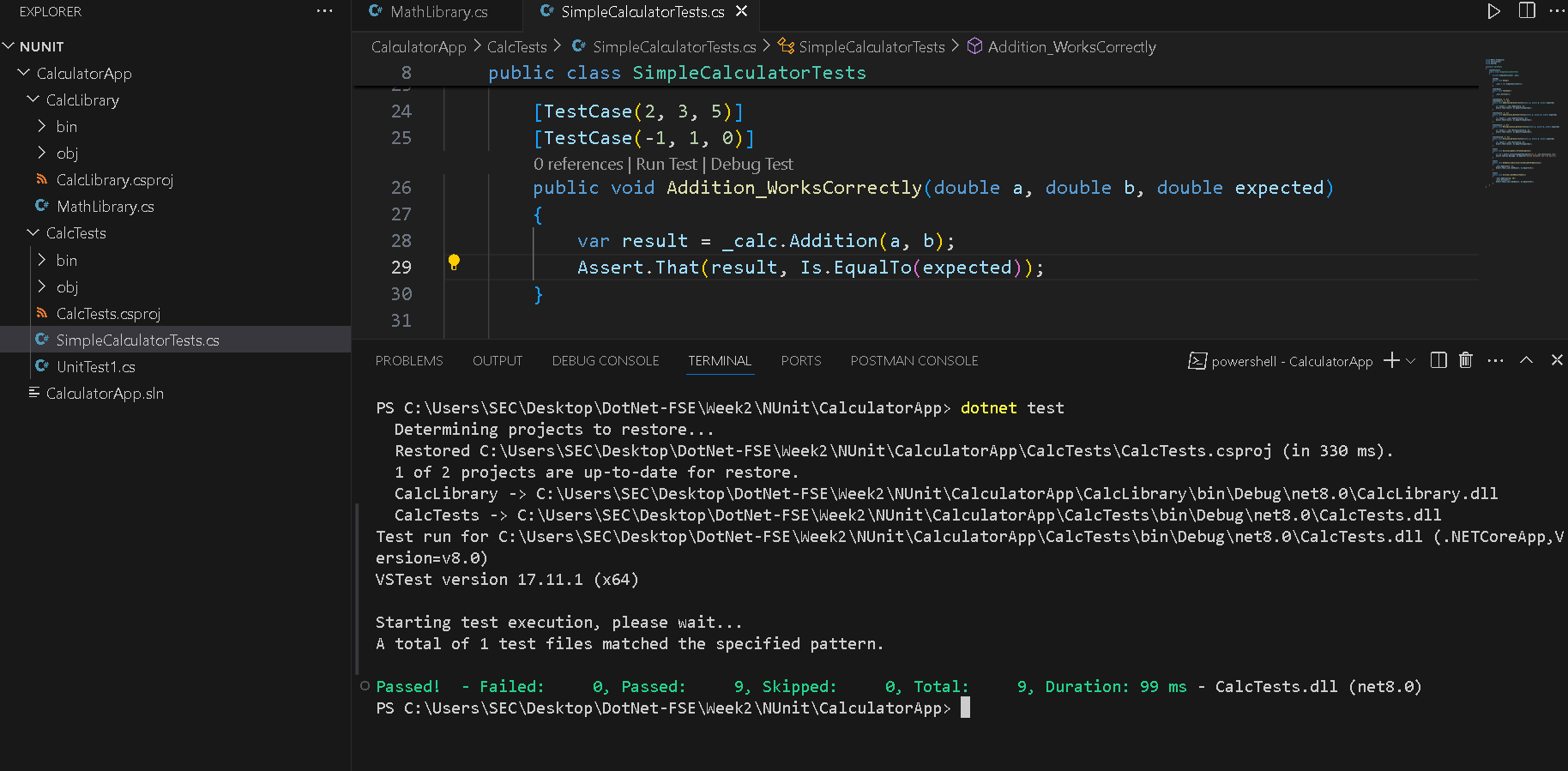
Assert.That(\_calc.GetResult, Is.EqualTo(0));

}

}

}

**OUTPUT:**

****

|  |
| --- |

**2.Moq-Handson:**

| **1. Write Testable Code with Moq** |
| --- |

**· Create a Class Library (Language C#) project using Visual Studio IDE, and name it as CustomerCommLib.**

**· Rename the default Class1 class name as MailSender.**

**· Include the following namespaces with ‘using’ directive.**

**o System.Net**

**o System.Net.Mail**

**In the above code we injected the dependency (IMailSender) through constructor of CustomerComm class so that we can pass the mock object of the dependency wherever it is necessary.**

**We have successfully created a class that's written in such a way that we can run a unit test against it and an exception won't be thrown. We achieve this by mocking the call to IMailSender.SendMail() and adding a mocked return value of true to it.**

**MailSender,cs**

using System.Net;

using System.Net.Mail;

namespace CustomerCommLib

{

public class MailSender : IMailSender

{

public bool SendMail(string toAddress, string message)

{

MailMessage mail = new MailMessage();

SmtpClient smtpServer = new SmtpClient("smtp.gmail.com");

mail.From = new MailAddress("your\_email\_address@gmail.com");

mail.To.Add(toAddress);

mail.Subject = "Test Mail";

mail.Body = message;

smtpServer.Port = 587;

smtpServer.Credentials = new NetworkCredential("username", "password");

smtpServer.EnableSsl = true;

smtpServer.Send(mail);

return true;

}

}

}

**CustomerCommunicatorTests.cs**

using NUnit.Framework;

using Moq;

using CustomerCommLib;

namespace CustomerComm.Tests

{

public class CustomerCommunicatorTests

{

[Test]

public void SendMailToCustomer\_ReturnsTrue\_WhenMailSent()

{

// Arrange

var mockMailSender = new Mock<IMailSender>();

mockMailSender.Setup(x => x.SendMail(It.IsAny<string>(), It.IsAny<string>())).Returns(true);

var customerComm = new CustomerCommunicator(mockMailSender.Object);

// Act

var result = customerComm.SendMailToCustomer();

// Assert

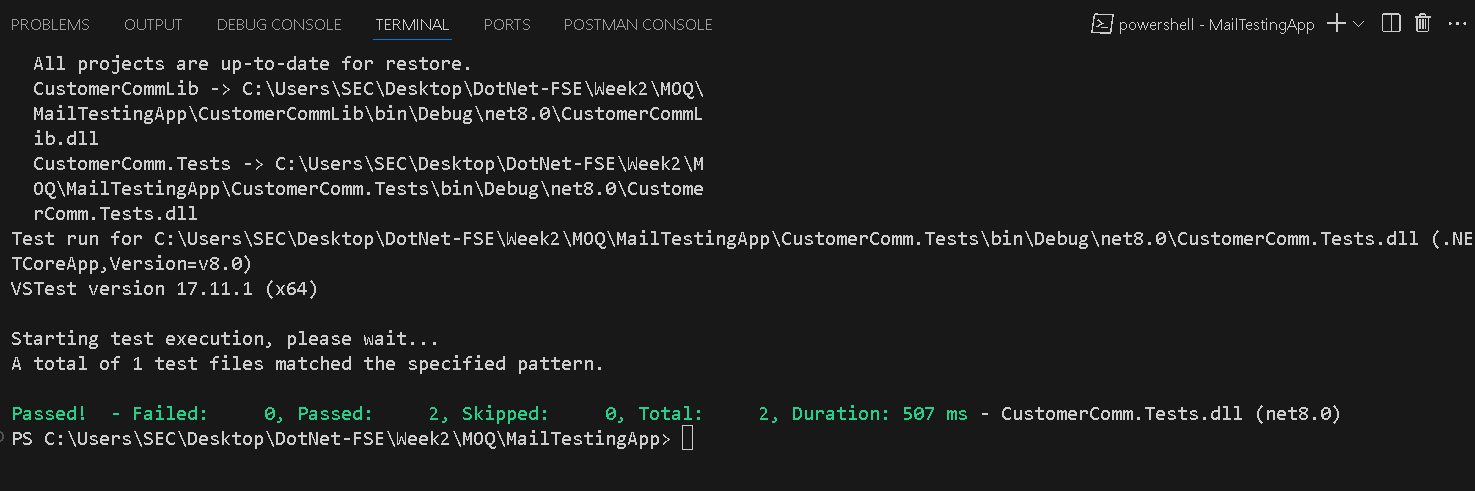
Assert.That(result, Is.True); // or Assert.IsTrue(result);

}

}

}

**OUTPUT:**

****