**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

using NUnit.Framework;

using CalcLibrary;

using System;

namespace UnitTests

{

[TestFixture]

public class SimpleCalculatorTests

{

private SimpleCalculator \_calculator;

[SetUp]

public void SetUp()

{

\_calculator = new SimpleCalculator();

}

[TearDown]

public void TearDown()

{

\_calculator = null; // Clean up resources

}

[Test]

[TestCase(2, 3, 5)]

[TestCase(-1, 1, 0)]

[TestCase(0, 0, 0)]

public void Addition\_ValidInputs\_ReturnsExpectedResult(double a, double b, double expectedResult)

{

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

Assert.That(result, Is.EqualTo(expectedResult), $"Addition of {a} and {b} should be {expectedResult}");

}

[Test]

[TestCase(5, 3, 2)]

[TestCase(-1, -1, 0)]

[TestCase(0, 1, -1)]

public void Subtraction\_ValidInputs\_ReturnsExpectedResult(double a, double b, double expectedResult)

{

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

Assert.That(result, Is.EqualTo(expectedResult), $"Subtraction of {a} and {b} should be {expectedResult}");

}

[Test]

[TestCase(2, 3, 6)]

[TestCase(-2, 3, -6)]

[TestCase(0, 5, 0)]

public void Multiplication\_ValidInputs\_ReturnsExpectedResult(double a, double b, double expectedResult)

{

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

Assert.That(result, Is.EqualTo(expectedResult), $"Multiplication of {a} and {b} should be {expectedResult}");

}

[Test]

[TestCase(6, 3, 2)]

[TestCase(-6, 3, -2)]

[TestCase(0, 1, 0)]

public void Division\_ValidInputs\_ReturnsExpectedResult(double a, double b, double expectedResult)

{

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

Assert.That(result, Is.EqualTo(expectedResult), $"Division of {a} by {b} should be {expectedResult}");

}

[Test]

public void Division\_ByZero\_ThrowsArgumentException()

{

Assert.Throws<ArgumentException>(() => \_calculator.Division(5, 0), "Second Parameter Can't be Zero");

}

[Test]

public void AllClear\_ResetsResultToZero()

{

// Perform some operations

\_calculator.Addition(5, 10);

Assert.That(\_calculator.GetResult, Is.EqualTo(15));

// Reset

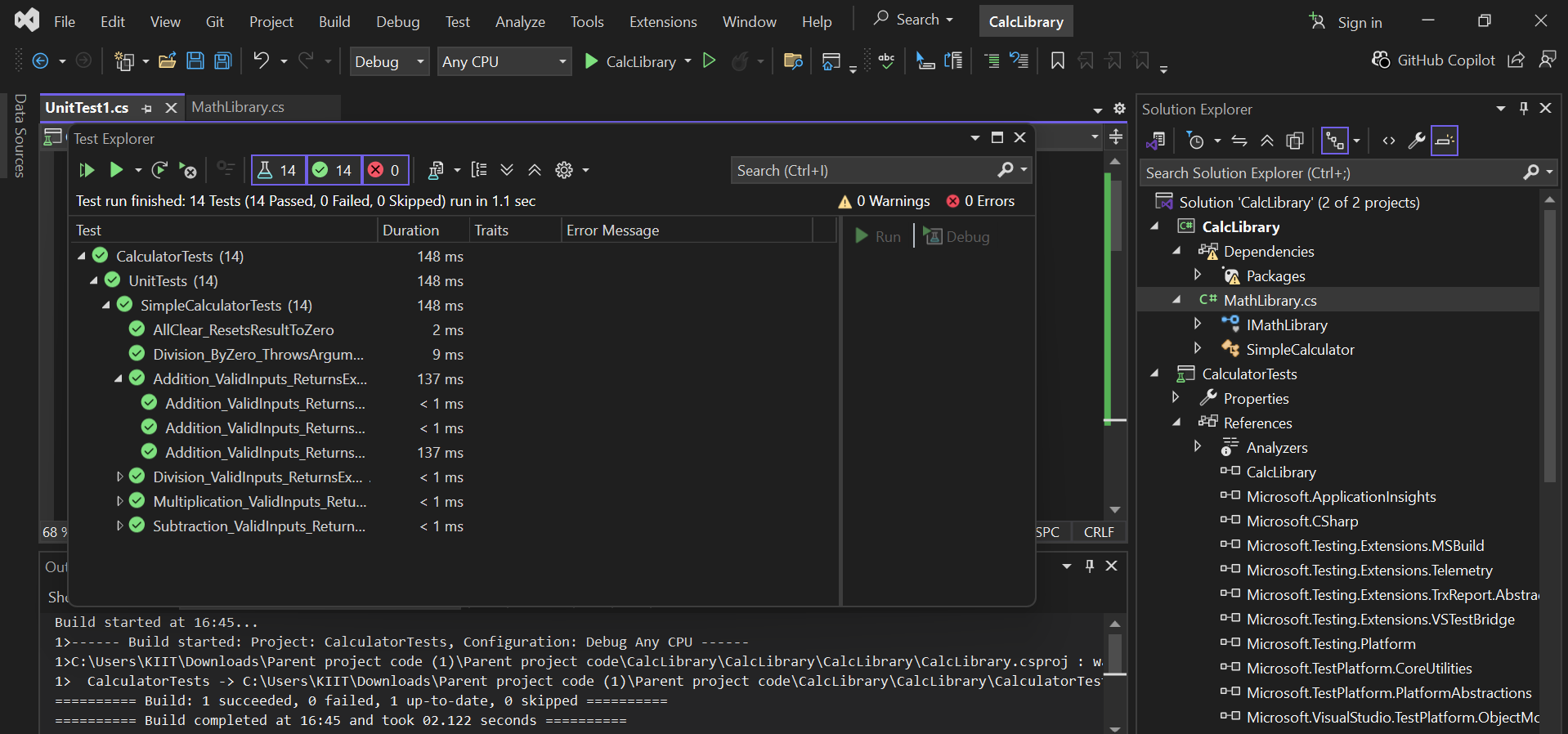
\_calculator.AllClear();

Assert.That(\_calculator.GetResult, Is.EqualTo(0), "AllClear should reset the result to 0");

}

}

}



**MOQ HANDSON**

**TASK1**

**MailSender code**

using System.Net;

using System.Net.Mail;

namespace CustomerCommLib

{

public interface IMailSender

{

bool SendMail(string toAddress, string message);

}

public class MailSender : IMailSender

{

public bool SendMail(string toAddress, string message)

{

try

{

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;

}

catch

{

return false;

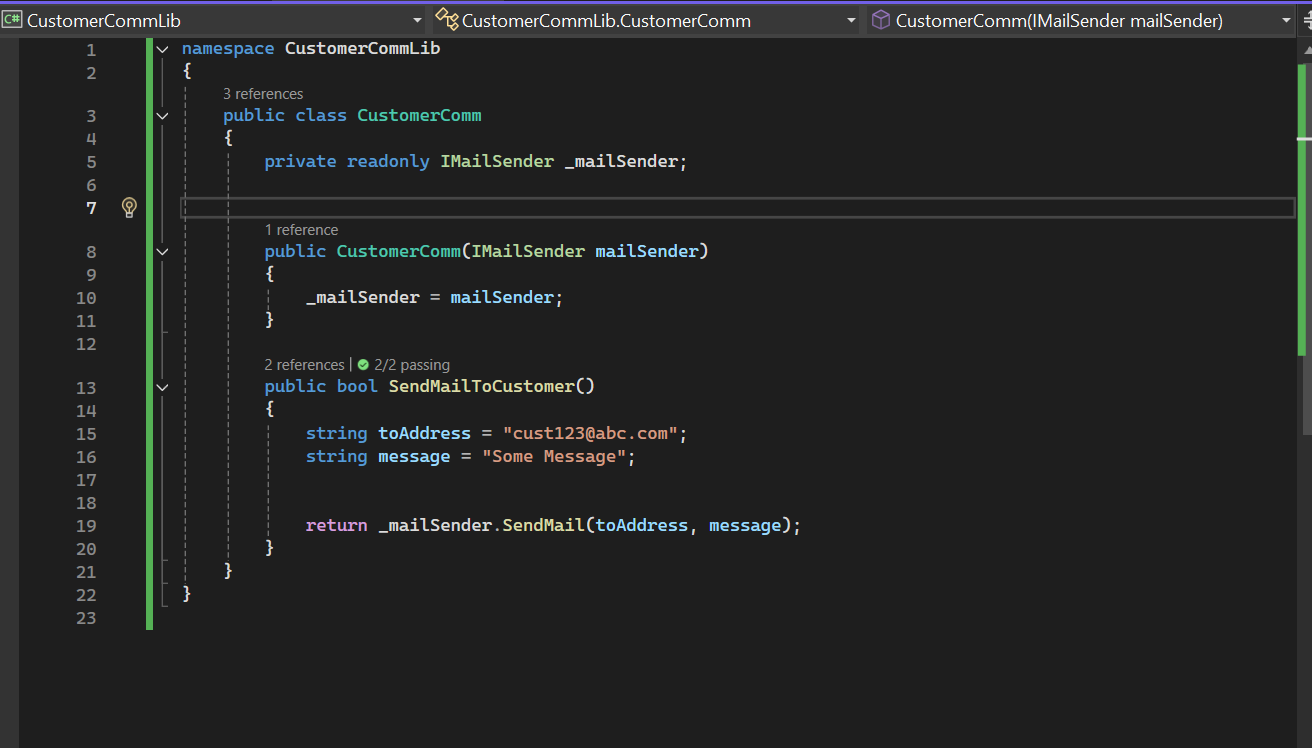
}

}

}

}

CustomerCommLib code

****

**CustomerCommTest code**

using NUnit.Framework;

using Moq;

using CustomerCommLib;

namespace CustomerCommTests

{

[TestFixture]

public class CustomerCommTests

{

private Mock<IMailSender> \_mailSenderMock;

private CustomerComm \_customerComm;

[SetUp]

public void SetUp()

{

\_mailSenderMock = new Mock<IMailSender>();

\_customerComm = new CustomerComm(\_mailSenderMock.Object);

}

[Test]

public void SendMailToCustomer\_WhenCalled\_ReturnsTrue()

{

\_mailSenderMock.Setup(ms => ms.SendMail(It.IsAny<string>(), It.IsAny<string>())).Returns(true);

var result = \_customerComm.SendMailToCustomer();

Assert.That(result, Is.True);

\_mailSenderMock.Verify(ms => ms.SendMail("cust123@abc.com", "Some Message"), Times.Once);

}

[Test]

public void SendMailToCustomer\_WhenMailFails\_ReturnsFalse()

{

\_mailSenderMock.Setup(ms => ms.SendMail(It.IsAny<string>(), It.IsAny<string>())).Returns(false);

var result = \_customerComm.SendMailToCustomer();

Assert.That(result, Is.False);

}

}

}

