**NUnit and Moq**

**NUnit-Handson**

**TestFixture & Test:**

Code:

using NUnit.Framework;

using System;

using CalcLibrary;

namespace CalcLibrary.Tests

{

[TestFixture]

public class SimpleCalculatorTests

{

private SimpleCalculator calc;

[SetUp]

public void Setup()

{

calc = new SimpleCalculator();

}

[TearDown]

public void TearDown()

{

calc.AllClear();

}

[Test]

[TestCase(3, 4, 7)]

[TestCase(-1, -1, -2)]

[TestCase(5.5, 2.5, 8)]

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

{

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

Assert.That(result, Is.EqualTo(expected).Within(0.0001));

}

[Test]

[TestCase(10, 4, 6)]

[TestCase(-2, -2, 0)]

[TestCase(5.5, 2.5, 3)]

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

{

double result = calc.Subtraction(a, b);

Assert.That(result, Is.EqualTo(expected).Within(0.0001));

}

[Test]

[TestCase(2, 3, 6)]

[TestCase(-1, 5, -5)]

[TestCase(2.5, 2, 5)]

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

{

double result = calc.Multiplication(a, b);

Assert.That(result, Is.EqualTo(expected).Within(0.0001));

}

[Test]

[TestCase(10, 2, 5)]

[TestCase(9, 3, 3)]

[TestCase(5.5, 2.2, 2.5)]

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

{

double result = calc.Division(a, b);

Assert.That(result, Is.EqualTo(expected).Within(0.0001));

}

[Test]

public void Division\_ByZero\_ThrowsArgumentException()

{

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

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

}

[Test]

public void AllClear\_SetsResultToZero()

{

calc.Addition(5, 10); // set some value

calc.AllClear();

Assert.That(calc.GetResult, Is.EqualTo(0));

}

[Test]

public void GetResult\_ReturnsLastComputedValue()

{

calc.Subtraction(10, 4);

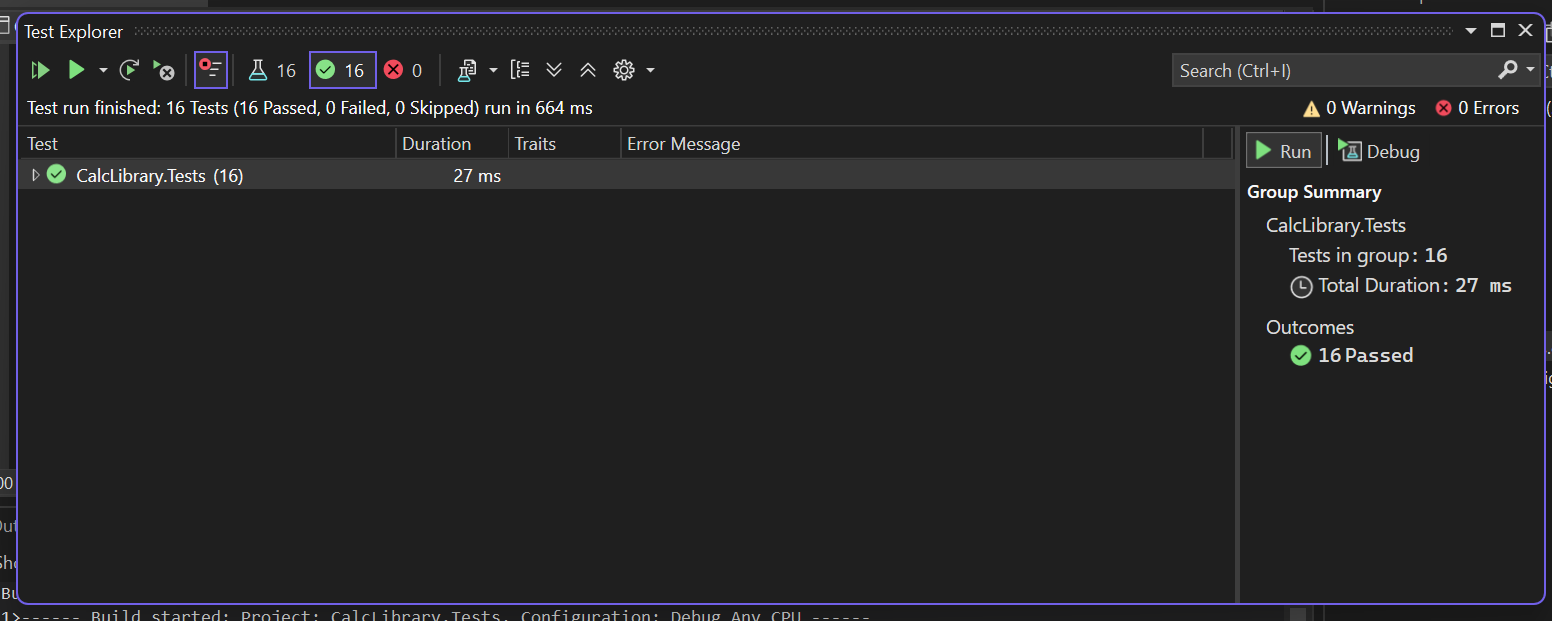
Assert.That(calc.GetResult, Is.EqualTo(6));

}

}

}

Output:



**Moq-Handson**

**Write Testable Code with Moq**

Task-1:

Code for IMailSender.cs:

namespace CustomerCommLib

{

public interface IMailSender

{

bool SendMail(string toAddress, string message);

}

}

Code for 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;

}

}

}

Code for CustomerComm:

namespace CustomerCommLib

{

public class CustomerComm

{

IMailSender \_mailSender;

public CustomerComm(IMailSender mailSender)

{

\_mailSender = mailSender;

}

public bool SendMailToCustomer()

{

\_mailSender.SendMail("cust123@abc.com", "Some Message");

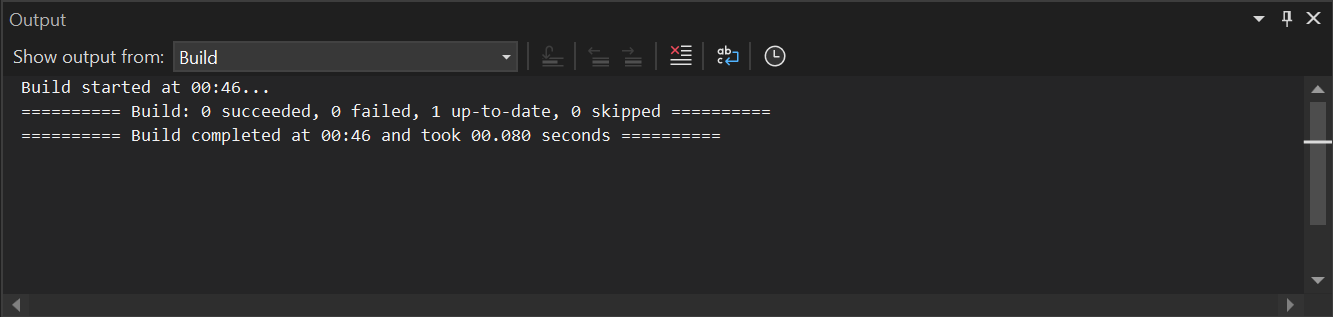
return true;

}

}

}

Output:



Task-2:

Code:

namespace CustomerComm.Tests

{

[TestFixture]

public class CustomerCommTests

{

private Mock<IMailSender> \_mockMailSender;

private CustomerCommLib.CustomerComm \_customerComm;

[OneTimeSetUp]

public void Init()

{

\_mockMailSender = new Mock<IMailSender>();

\_mockMailSender

.Setup(ms => ms.SendMail(It.IsAny<string>(), It.IsAny<string>()))

.Returns(true);

\_customerComm = new CustomerCommLib.CustomerComm(\_mockMailSender.Object);

}

[TestCase]

public void SendMailToCustomer\_ShouldReturnTrue()

{

var result = \_customerComm.SendMailToCustomer();

Assert.IsTrue(result);

\_mockMailSender.Verify(ms => ms.SendMail(It.IsAny<string>(), It.IsAny<string>()), Times.Once);

}

}

}

Output:

