**NUnit Handson**

using System;

using NUnit.Framework;

namespace CalculatorSingleFileDemo

{

    // The class to be tested

    public class Calculator

    {

        public int Add(int a, int b) => a + b;

    }

    // The test class

    [TestFixture]

    public class CalculatorTests

    {

        private Calculator \_calculator;

        [SetUp]

        public void SetUp()

        {

            \_calculator = new Calculator();

        }

        [TearDown]

        public void TearDown()

        {

            \_calculator = null;

        }

        [Test]

        [TestCase(1, 2, 3)]

        [TestCase(-1, -1, -2)]

        [TestCase(0, 0, 0)]

        [TestCase(int.MaxValue, 0, int.MaxValue)]

        public void Add\_WhenCalled\_ReturnsSum(int a, int b, int expected)

        {

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

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

        }

        [Test, Ignore("Example of ignoring a test")]

        public void IgnoredTest()

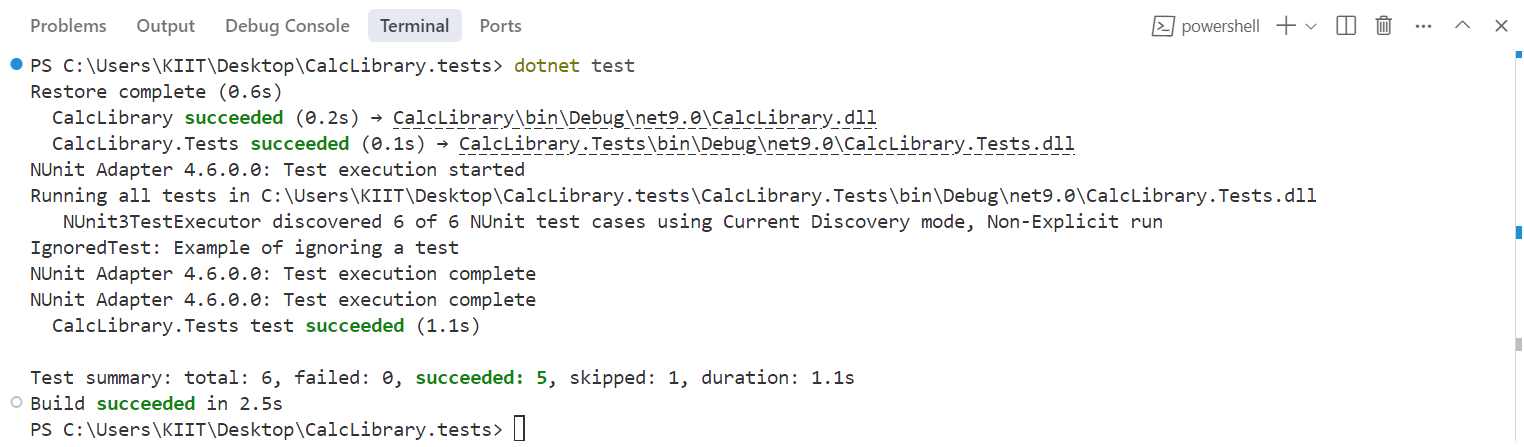
        {

            Assert.Fail("This test is ignored.");

        }

    }

}

****

**Moq Handson**

Write Testable Code with Moq

using System;

using System.Net;

using System.Net.Mail;

using NUnit.Framework;

using Moq;

namespace CustomerCommLib

{

    public interface IMailSender

    {

        bool SendMail(string toAddress, string message);

    }

    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); // Commented out for safety in testable scenario

            return true;

        }

    }

    public class CustomerComm

    {

        private IMailSender \_mailSender;

        public CustomerComm(IMailSender mailSender)

        {

            \_mailSender = mailSender;

        }

        public bool SendMailToCustomer()

        {

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

        }

    }

}

namespace CustomerComm.Tests

{

    using CustomerCommLib;

    [TestFixture]

    public class CustomerCommTests

    {

        private CustomerComm \_customerComm;

        private Mock<IMailSender> \_mockMailSender;

        [OneTimeSetUp]

        public void Setup()

        {

            \_mockMailSender = new Moq.Mock<IMailSender>();

            // Always return true for any two strings

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

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

        }

        [TestCase]

        public void SendMailToCustomer\_ShouldReturnTrue()

        {

            var result = \_customerComm.SendMailToCustomer();

            Assert.That(result, Is.True);

            TestContext.Progress.WriteLine("Test Passed: SendMailToCustomer\_ShouldReturnTrue");

        }

    }

}

