# **Name- Sayandeep Dey (SupersetID:** **6363427)**

# **WEEK – 2 (Handson- Exercises)**

1. **Nunit Testing:**

**Exercise 1:** **CalculatorTests:**

**Code: (In CalculatorTests.cs(Test/CalcLibraryTest):**

using NUnit.Framework;

using CalcLibrary;

using System;

namespace CalcLibraryTests

{

    // TestFixture marks this class as a container for unit tests

    [TestFixture]

    public class CalculatorTests

    {

        private SimpleCalculator calc;

        // SetUp runs before each test method

        [SetUp]

        public void Init()

        {

            calc = new SimpleCalculator(); // Loose coupling: no external dependency

        }

        // TearDown runs after each test method

        [TearDown]

        public void Cleanup()

        {

            calc.AllClear(); // Reset state after each test

        }

        //  1. Addition with multiple parameterized test cases

        [Test]

        [TestCase(2, 3, 5)]

        [TestCase(-1, -1, -2)]

        [TestCase(0, 0, 0)]

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

        {

            Assert.That(calc.Addition(a, b), Is.EqualTo(expected));

        }

        //  2. Subtraction test

        [Test]

        [TestCase(10, 5, 5)]

        [TestCase(0, 0, 0)]

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

        {

            Assert.That(calc.Subtraction(a, b), Is.EqualTo(expected));

        }

        //  3. Multiplication test

        [Test]

        [TestCase(2, 3, 6)]

        [TestCase(-2, 3, -6)]

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

        {

            Assert.That(calc.Multiplication(a, b), Is.EqualTo(expected));

        }

        //  4. Division test with valid input

        [Test]

        [TestCase(10, 2, 5)]

        [TestCase(9, 3, 3)]

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

        {

            Assert.That(calc.Division(a, b), Is.EqualTo(expected));

        }

        //  5. Division by zero test

        [Test]

        public void Division\_ByZero\_ThrowsArgumentException()

        {

            var ex = Assert.Throws<ArgumentException>(() => calc.Division(10, 0));

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

        }

        //  6. [Ignore] attribute example

        [Test]

        [Ignore("This test is ignored for demonstration purposes")]

        public void IgnoredTest()

        {

            Assert.That(1 + 1, Is.EqualTo(3)); // This won't run

        }

        //  7. Internal state testing via GetResult property

        [Test]

        public void GetResult\_ShouldReturnLastComputedValue()

        {

            calc.Addition(5, 5);

            Assert.That(calc.GetResult, Is.EqualTo(10));

        }

        [Test]

        public void AllClear\_ShouldResetResultToZero()

        {

            calc.Multiplication(4, 2);

            calc.AllClear();

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

        }

    }

}

**In CalcLibraryTests.csproj file:**

<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>

    <TargetFramework>net8.0</TargetFramework>

    <IsPackable>false</IsPackable>

  </PropertyGroup>

  <ItemGroup>

    <ProjectReference Include="..\..\CalcLibrary\CalcLibrary.csproj" />

  </ItemGroup>

  <ItemGroup>

    <PackageReference Include="Microsoft.NET.Test.Sdk" Version="17.9.0" />

    <PackageReference Include="NUnit" Version="3.13.3" />

    <PackageReference Include="NUnit3TestAdapter" Version="4.5.0" />

    <PackageReference Include="coverlet.collector" Version="3.2.0" />

  </ItemGroup>

</Project>

**Output:** **A computer screen shot of a program

AI-generated content may be incorrect.**

**Exercise 2:** **CalculatorTests(Nunit Testing):**

**Code: (In CalculatorTests.cs(Test/CalcLibraryTest):**

**Code:**

using NUnit.Framework;

using CalcLibrary;

using System;

namespace CalcLibraryTests

{

    [TestFixture]

    public class CalculatorTests

    {

        private SimpleCalculator calc;

        [SetUp]

        public void Init()

        {

            calc = new SimpleCalculator(); // Loosely coupled: no external dependency

        }

        [TearDown]

        public void Cleanup()

        {

            calc.AllClear(); // Reset after each test

        }

        // 🔹 Addition using Assert.That

        [Test]

        [TestCase(2, 3, 5)]

        [TestCase(-1, -1, -2)]

        [TestCase(0, 0, 0)]

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

        {

            Assert.That(calc.Addition(a, b), Is.EqualTo(expected));

        }

        // 🔹 Subtraction using Assert.AreEqual

        [Test]

        [TestCase(10, 5, 5)]

        [TestCase(0, 0, 0)]

        [TestCase(-5, -5, 0)]

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

        {

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

            Assert.AreEqual(expected, result);

        }

        // 🔹 Multiplication using Assert.AreEqual

        [Test]

        [TestCase(2, 3, 6)]

        [TestCase(-2, 3, -6)]

        [TestCase(0, 10, 0)]

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

        {

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

            Assert.AreEqual(expected, result);

        }

        // 🔹 Division using Assert.AreEqual and try-catch

        [Test]

        [TestCase(10, 2, 5)]

        [TestCase(9, 3, 3)]

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

        {

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

            Assert.AreEqual(expected, result);

        }

        [Test]

        public void Division\_ByZero\_ShouldThrow()

        {

            try

            {

                calc.Division(10, 0);

                Assert.Fail("Division by zero"); // This will show in test explorer if exception not thrown

            }

            catch (ArgumentException ex)

            {

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

            }

            catch (Exception)

            {

                Assert.Fail("Unexpected exception type");

            }

        }

        // 🔹 Void method testing

        [Test]

        public void TestAddAndClear()

        {

            calc.Addition(7, 3);

            Assert.AreEqual(10, calc.GetResult);

            calc.AllClear();

            Assert.AreEqual(0, calc.GetResult);

        }

        // 🔹 Ignored test

        [Test]

        [Ignore("This test is ignored for demonstration purposes")]

        public void IgnoredTest()

        {

            Assert.AreEqual(1 + 1, 3); // Should never be executed

        }

        // 🔹 AssertionException demonstration

        [Test]

        public void AssertionExceptionExample()

        {

            try

            {

                Assert.AreEqual(5, calc.Subtraction(3, 2)); // Intentionally wrong

            }

            catch (AssertionException ex)

            {

                TestContext.WriteLine("Caught AssertionException: " + ex.Message);

                Assert.Pass(); // Confirm we expected this failure

            }

            Assert.Fail("Test did not throw AssertionException as expected");

        }

        // 🔸🔒 Private Method Testing — Why not?

        /\*

         \* Private methods should not be tested directly because:

         \* - They are implementation details.

         \* - Unit tests should only verify public behavior.

         \* - Testing private methods tightly couples tests to internal changes.

         \*/

        // 🔸🧪 Mocking (Moq framework) — usage explained

        /\*

         \* Mocking is used to simulate external dependencies like:

         \* - Databases

         \* - APIs

         \* - File systems

         \*

         \* Moq is a popular .NET mocking framework.

         \* Example: Mock<IDatabase> mockDb = new Mock<IDatabase>();

         \* Then inject it into your class under test.

         \*/

    }

}

**Output:**

**A screen shot of a computer

AI-generated content may be incorrect.**

**Exercise: 2(MOQ Handson Excercies):**

**Code: (Task-1): Project File name: CustomerCommLib**

**In CustomerCommLib.cs :**

namespace CustomerCommLib

{

    public class CustomerComm

    {

        private readonly IMailSender \_mailSender;

        public CustomerComm(IMailSender mailSender)

        {

            \_mailSender = mailSender;

        }

      public bool SendMailToCustomer(string to, string message)

{

    return \_mailSender.SendMail(to, message);

}

    }

}

**In IMailSender.cs:**

namespace CustomerCommLib

{

    public interface IMailSender

    {

        bool SendMail(string toAddress, string message);

    }

}

**In 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("sayandeeepdey1511@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;

        }

    }

}

**Project File name: CustomerCommLib.Tests:**

**In CustomerCommLibTests.cs:**

using NUnit.Framework;

using Moq;

using CustomerCommLib;

namespace CustomerCommLib.Tests

{

    [TestFixture]

    public class CustomerCommTests

    {

       [Test]

public void SendMailToCustomer\_ShouldReturnTrue\_WhenMocked()

{

    var mockSender = new Mock<IMailSender>();

    mockSender.Setup(m => m.SendMail("sayandeepdey1511@gmail.com", "hello tester")).Returns(true);

    var comm = new CustomerComm(mockSender.Object);

    var result = comm.SendMailToCustomer("sayandeepdey1511@gmail.com", "hello tester");

    Assert.IsTrue(result);

    mockSender.Verify(m => m.SendMail("sayandeepdey1511@gmail.com", "hello tester"), Times.Once);

}

    }

}

**In CustomerCommLibTests.csproj:**

<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>

    <TargetFramework>net8.0</TargetFramework>

    <IsPackable>false</IsPackable>

  </PropertyGroup>

  <ItemGroup>

    <ProjectReference Include="..\CustomerCommLib\CustomerCommLib.csproj" />

    <PackageReference Include="Microsoft.NET.Test.Sdk" Version="17.9.0" />

    <PackageReference Include="Moq" Version="4.20.70" />

    <PackageReference Include="NUnit" Version="3.13.3" />

    <PackageReference Include="NUnit3TestAdapter" Version="4.5.0" />

  </ItemGroup>

</Project>

**Output:**

**A screen shot of a computer

AI-generated content may be incorrect.**

**Task- 2:**

**Code:**

**In CustomerCommLib.cs :**

namespace CustomerCommLib

{

    public class CustomerComm

    {

        private readonly IMailSender \_mailSender;

        public CustomerComm(IMailSender mailSender)

        {

            \_mailSender = mailSender;

        }

        public bool SendMailToCustomer()

        {

            return \_mailSender.SendMail("sayandeepdey1511@gmail.com", "Some Message");

        }

    }

}

**In IMailSender.cs:**

namespace CustomerCommLib

{

    public interface IMailSender

    {

        bool SendMail(string toAddress, string message);

    }

}

**In 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("sayandeepdey1511@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;

        }

    }

}

**In CustomerCommTests.cs:**

using NUnit.Framework;

using Moq;

using CustomerCommLib;

namespace CustomerCommLib.Tests

{

    [TestFixture]

    public class CustomerCommTests

    {

        private Mock<IMailSender> \_mockMailSender;

        private CustomerComm \_customerComm;

        [OneTimeSetUp]

        public void Init()

        {

            \_mockMailSender = new Mock<IMailSender>();

            // Mock setup: any string for both parameters → return true

            \_mockMailSender

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

                .Returns(true);

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

        }

        [TestCase]

        public void SendMailToCustomer\_ShouldReturnTrue\_WhenMocked()

        {

            var result = \_customerComm.SendMailToCustomer();

            Assert.That(result, Is.True);

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

        }

    }

}

**Output:**

**A screenshot of a computer program

AI-generated content may be incorrect.**