Name-Sangram Mandal

Superset Id-6363848

Week-2(Handson-Exercise)

1. Nunit Handson

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

**Code :**

using NUnit.Framework;

using CalcLibrary;

using System;

namespace CalcLibraryTests

{

    [TestFixture]

    public class CalculatorTests

    {

        private SimpleCalculator calculator;

        [SetUp]

        public void Setup()

        {

            calculator = new SimpleCalculator();

        }

        [TearDown]

        public void Teardown()

        {

            calculator.AllClear();

        }

        [Test]

        [TestCase(2, 3, 5)]

        [TestCase(-1, -1, -2)]

        [TestCase(0, 0, 0)]

        public void Add\_ShouldReturnExpectedSum(double x, double y, double expected)

        {

            var result = calculator.Addition(x, y);

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

        }

        [Test]

        [TestCase(10, 5, 5)]

        [TestCase(0, 0, 0)]

        public void Subtract\_ShouldReturnCorrectDifference(double x, double y, double expected)

        {

            var result = calculator.Subtraction(x, y);

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

        }

        [Test]

        [TestCase(2, 3, 6)]

        [TestCase(-2, 3, -6)]

        public void Multiply\_ShouldReturnCorrectProduct(double x, double y, double expected)

        {

            var result = calculator.Multiplication(x, y);

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

        }

        [Test]

        [TestCase(10, 2, 5)]

        [TestCase(9, 3, 3)]

        public void Divide\_WithValidInput\_ShouldReturnQuotient(double x, double y, double expected)

        {

            var result = calculator.Division(x, y);

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

        }

        [Test]

        public void Divide\_ByZero\_ShouldThrowException()

        {

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

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

        }

        [Test]

        [Ignore("Test case intentionally disabled")]

        public void ThisTestIsIgnored()

        {

            Assert.Fail("This should not run");

        }

        [Test]

        public void Result\_ShouldMatchLastOperationValue()

        {

            calculator.Addition(5, 5);

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

        }

        [Test]

        public void Reset\_ShouldClearResultToZero()

        {

            calculator.Multiplication(4, 2);

            calculator.AllClear();

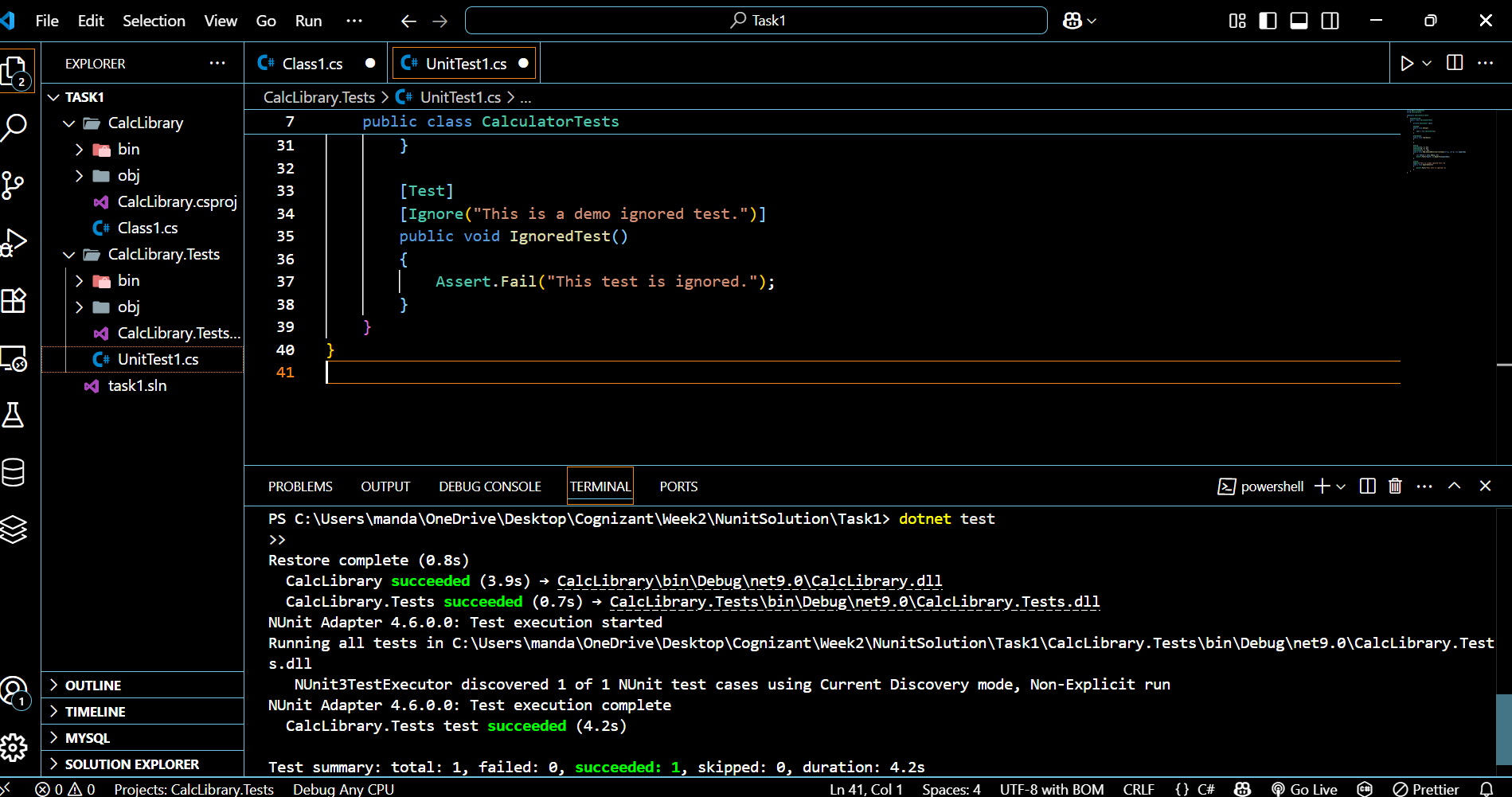
            Assert.That(calculator.GetResult, Is.EqualTo(0));

        }

    }

}

Output :



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

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

using NUnit.Framework;

using CalcLibrary;

using System;

namespace CalcLibraryTests

{

    [TestFixture]

    public class CalculatorTests

    {

        private SimpleCalculator calculator;

        [SetUp]

        public void SetUp()

        {

            calculator = new SimpleCalculator();

        }

        [TearDown]

        public void TearDown()

        {

            calculator.AllClear();

        }

        [Test]

        [TestCase(2, 3, 5)]

        [TestCase(-1, -1, -2)]

        [TestCase(0, 0, 0)]

        public void Add\_WithVariousInputs\_ReturnsCorrectSum(double x, double y, double expected)

        {

            var actual = calculator.Addition(x, y);

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

        }

        [Test]

        [TestCase(10, 5, 5)]

        [TestCase(-4, -6, 2)]

        [TestCase(0, 0, 0)]

        public void Subtract\_ValidInputs\_ReturnsExpectedResult(double x, double y, double expected)

        {

            var actual = calculator.Subtraction(x, y);

            Assert.AreEqual(expected, actual);

        }

        [Test]

        [TestCase(2, 3, 6)]

        [TestCase(-2, 4, -8)]

        [TestCase(0, 100, 0)]

        public void Multiply\_UsingTestCases\_ReturnsCorrectProduct(double x, double y, double expected)

        {

            var actual = calculator.Multiplication(x, y);

            Assert.AreEqual(expected, actual);

        }

        [Test]

        [TestCase(12, 3, 4)]

        [TestCase(7, 7, 1)]

        public void Divide\_WithValidNumbers\_ReturnsQuotient(double x, double y, double expected)

        {

            var actual = calculator.Division(x, y);

            Assert.AreEqual(expected, actual);

        }

        [Test]

        public void Divide\_WhenDivisorIsZero\_ThrowsArgumentException()

        {

            try

            {

                calculator.Division(5, 0);

                Assert.Fail("Division by zero");             }

            catch (ArgumentException ex)

            {

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

            }

            catch (Exception)

            {

                Assert.Fail("Unexpected exception type was thrown.");

            }

        }

        [Test]

        public void AddAndReset\_ResultIsClearedToZero()

        {

            calculator.Addition(4, 6);

            Assert.AreEqual(10, calculator.GetResult);

            calculator.AllClear();

            Assert.AreEqual(0, calculator.GetResult);

        }

        [Test]

        [Ignore("Skipping test intentionally for demonstration")]

        public void Ignored\_TestExample()

        {

            Assert.AreEqual(5, 99);

        }

        [Test]

        public void TriggerAssertionException\_CatchAndPass()

        {

            try

            {

                Assert.AreEqual(100, calculator.Subtraction(4, 2));

            }

            catch (AssertionException ex)

            {

                TestContext.WriteLine("Expected assertion failure: " + ex.Message);

                Assert.Pass();

            }

            Assert.Fail("Expected AssertionException was not thrown.");

        }

    }

}

Output :

