NUNIT AND MOQ)

using System;

using NUnit.Framework;

using CalcLibrary;

namespace CalcLibrary.Tests

{

    [TestFixture]

    public class CalculatorTests

    {

        private SimpleCalculator calculator;

        [SetUp]

        public void Setup()

        {

            // Initialize calculator before each test

            calculator = new SimpleCalculator();

            Console.WriteLine("Test setup completed - Calculator initialized");

        }

        [TearDown]

        public void TearDown()

        {

            // Cleanup after each test

            calculator.AllClear();

            Console.WriteLine("Test cleanup completed - Calculator cleared");

        }

        [Test]

        public void Addition\_ValidInputs\_ReturnsCorrectSum()

        {

            // Arrange

            double a = 5.0;

            double b = 3.0;

            double expected = 8.0;

            // Act

            double actual = calculator.Addition(a, b);

            // Assert

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

        }

        [TestCase(1, 2, 3)]

        [TestCase(10, 5, 15)]

        [TestCase(-1, -2, -3)]

        [TestCase(0, 0, 0)]

        [TestCase(1.5, 2.5, 4.0)]

        public void Addition\_ParameterizedTests\_ReturnsCorrectSum(double a, double b, double expected)

        {

            // Act

            double actual = calculator.Addition(a, b);

            // Assert

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

        }

        [Test]

        public void Subtraction\_ValidInputs\_ReturnsCorrectDifference()

        {

            // Arrange

            double a = 10.0;

            double b = 4.0;

            double expected = 6.0;

            // Act

            double actual = calculator.Subtraction(a, b);

            // Assert

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

        }

        [TestCase(10, 5, 5)]

        [TestCase(0, 5, -5)]

        [TestCase(-3, -2, -1)]

        [TestCase(7.5, 2.5, 5.0)]

        public void Subtraction\_ParameterizedTests\_ReturnsCorrectDifference(double a, double b, double expected)

        {

            // Act

            double actual = calculator.Subtraction(a, b);

            // Assert

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

        }

        [Test]

        public void Multiplication\_ValidInputs\_ReturnsCorrectProduct()

        {

            // Arrange

            double a = 4.0;

            double b = 5.0;

            double expected = 20.0;

            // Act

            double actual = calculator.Multiplication(a, b);

            // Assert

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

        }

        [TestCase(2, 3, 6)]

        [TestCase(0, 5, 0)]

        [TestCase(-2, 3, -6)]

        [TestCase(2.5, 4, 10.0)]

        public void Multiplication\_ParameterizedTests\_ReturnsCorrectProduct(double a, double b, double expected)

        {

            // Act

            double actual = calculator.Multiplication(a, b);

            // Assert

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

        }

        [Test]

        public void Division\_ValidInputs\_ReturnsCorrectQuotient()

        {

            // Arrange

            double a = 10.0;

            double b = 2.0;

            double expected = 5.0;

            // Act

            double actual = calculator.Division(a, b);

            // Assert

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

        }

        [TestCase(10, 2, 5)]

        [TestCase(15, 3, 5)]

        [TestCase(-6, 2, -3)]

        [TestCase(7.5, 2.5, 3.0)]

        public void Division\_ParameterizedTests\_ReturnsCorrectQuotient(double a, double b, double expected)

        {

            // Act

            double actual = calculator.Division(a, b);

            // Assert

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

        }

        [Test]

        public void Division\_ByZero\_ThrowsArgumentException()

        {

            // Arrange

            double a = 10.0;

            double b = 0.0;

            // Act & Assert

            Assert.That(() => calculator.Division(a, b),

                       Throws.TypeOf<ArgumentException>()

                       .With.Message.EqualTo("Second Parameter Can't be Zero"));

        }

        [Test]

        public void GetResult\_AfterAddition\_ReturnsLastResult()

        {

            // Arrange

            double a = 3.0;

            double b = 4.0;

            double expected = 7.0;

            // Act

            calculator.Addition(a, b);

            double actual = calculator.GetResult;

            // Assert

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

        }

        [Test]

        public void AllClear\_ResetsResult\_ToZero()

        {

            // Arrange

            calculator.Addition(5, 3); // Result should be 8

            // Act

            calculator.AllClear();

            double actual = calculator.GetResult;

            // Assert

            Assert.That(actual, Is.EqualTo(0));

        }

        [Test]

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

        public void IgnoredTest\_WillNotRun()

        {

            // This test will be skipped during execution

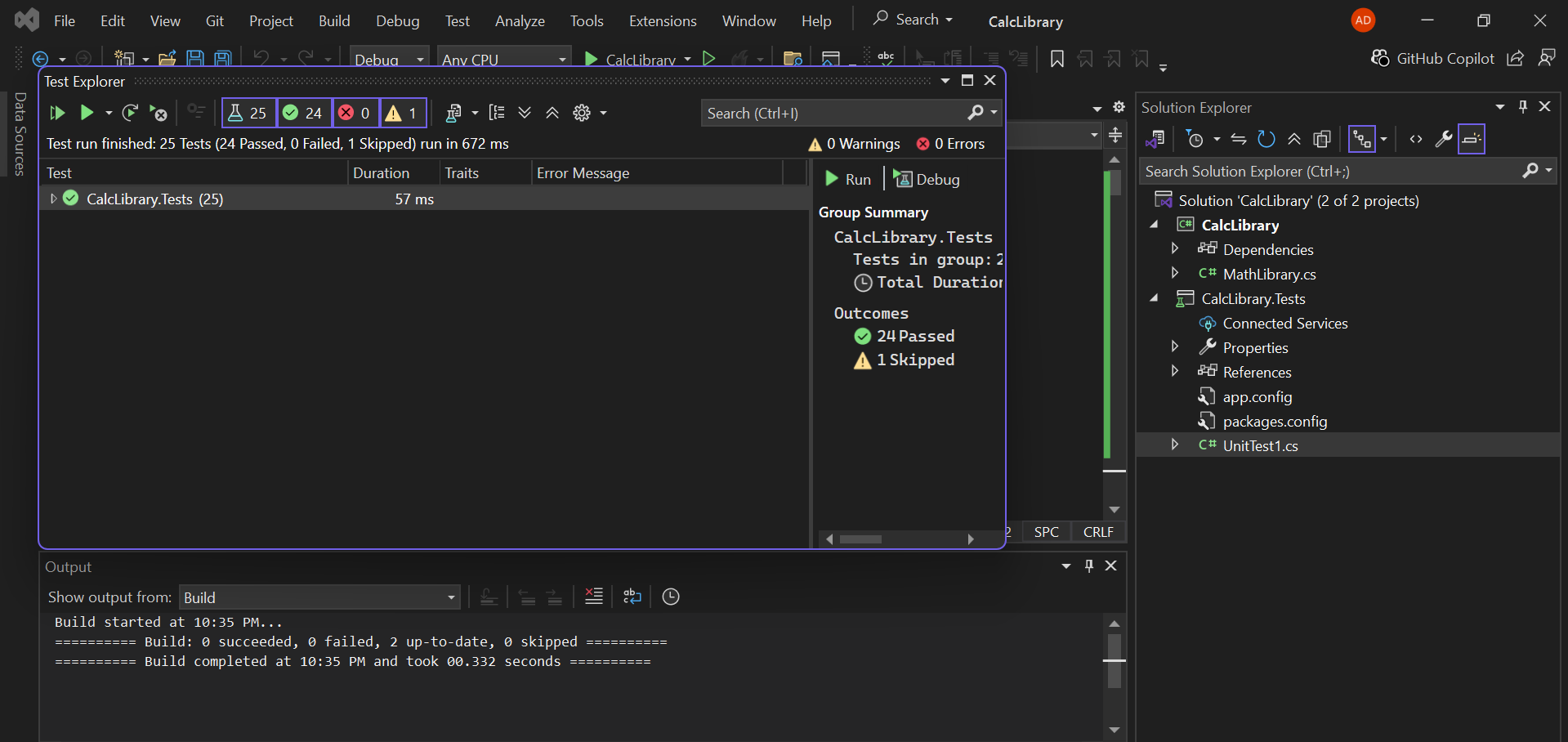
            Assert.Fail("This test should not run");

        }

    }

}

OUTPUT)



Q2)

using NUnit.Framework;

using Moq;

using CustomerCommLib;

namespace CustomerCommLib.Tests

{

    [TestFixture]

    public class CustomerCommTests

    {

        private Mock<IMailSender> \_mockMailSender;

        private CustomerComm \_customerComm;

        [SetUp]

        public void SetUp()

        {

            // Arrange - Create mock object

            \_mockMailSender = new Mock<IMailSender>();

            // Inject the mock object into CustomerComm

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

        }

        [Test]

        public void SendMailToCustomer\_ShouldCallSendMail\_WithCorrectParameters()

        {

            // Arrange - Set up mock behavior

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

                          .Returns(true);

            // Act - Call the method under test

            bool result = \_customerComm.SendMailToCustomer();

            // Assert - Verify the method was called with correct parameters

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

            Assert.IsTrue(result);

        }

        [Test]

        public void SendMailToCustomer\_ShouldReturnTrue\_WhenMailSentSuccessfully()

        {

            // Arrange

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

                          .Returns(true);

            // Act

            bool result = \_customerComm.SendMailToCustomer();

            // Assert

            Assert.IsTrue(result);

        }

        [Test]

        public void SendMailToCustomer\_ShouldCallSendMailOnce()

        {

            // Arrange

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

                          .Returns(true);

            // Act

            \_customerComm.SendMailToCustomer();

            // Assert - Verify SendMail was called exactly once

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

        }

}

}

Output  
