Question:
Objectives

- Explain the meaning of Unit testing and its difference on comparison with Functional testing
    - Smallest unit to test mocking dependencies
- List various types of testing
    - Unit testing, Functional testing, Automated testing, Performance testing
- Understand the benefit of automated testing
- Explain what is loosly coupled & testable design
    - Write code that is NOT dependent on the class for data.
- Write your first testing program to validate a calculator addition operation
    - TestFixture, Test
- Understand the need of [SetUp], [TearDown] & [Ignore] attributes.
- Explain the benefit of writing parameterised test cases.
    - TestCase

TestFixture & Test

Please download the application available here. This will be used to write Unit test cases

Follow the steps listed below to write the NUnit test cases for the application.

- Create a Unit test project(.Net Framework) in the solution provided.
- Add the CalcLibrary project as reference
- Create a class “CalculatorTests” to write all the test cases for the methods in the solution
- Use the ‘TestFixture’, ‘SetUp’ and ‘TearDown’ attributes, to declare, initialize and cleanup activities respectively
- Create a Test method to check the addition functionality
- Use the ‘TestCase’ attribute to send the inputs and the expected result
- Use Assert.That to check the actual and expected result match

SOLUTION:

Test code file:

```
using NUnit.Framework;
using CalcLibrary;
using System;

namespace CalcLibrary.Tests
{
    [TestFixture]
```

```
    public class CalculatorTests
    {
        private SimpleCalculator calculator;
        [SetUp]
        public void Setup()
        {
            calculator = new SimpleCalculator();
        }
        [TearDown]
        public void Cleanup()
        {
            calculator.AllClear();
        }
        [Test]
        [TestCase(5, 3, 8)]
        [TestCase(-2, 10, 8)]
        [TestCase(0, 0, 0)]
        public void TestAddition(double a, double b, double expected)
        {
            double result = calculator.Addition(a, b);
            Assert.That(result, Is.EqualTo(expected));
        }
        [Test]
        [TestCase(10, 4, 6)]
        [TestCase(5, -3, 8)]
        public void TestSubtraction(double a, double b, double expected)
        {
            double result = calculator.Subtraction(a, b);
            Assert.That(result, Is.EqualTo(expected));
        }
        [Test]
        [TestCase(3, 4, 12)]
        [TestCase(7, 0, 0)]
        public void TestMultiplication(double a, double b, double expected)
        {
            double result = calculator.Multiplication(a, b);
            Assert.That(result, Is.EqualTo(expected));
        }
        [Test]
        [TestCase(20, 5, 4)]
        public void TestDivision(double a, double b, double expected)
        {
            double result = calculator.Division(a, b);
            Assert.That(result, Is.EqualTo(expected));
        }
        [Test]
        public void TestDivisionByZero()
        {
            Assert.Throws<ArgumentException>(() => calculator.Division(10, 0));
        }
    }
}
```

Output:

```
PS C:\Users\KIIT\OneDrive\Desktop\NUnit handson\CalcLibrary\CalcLibrary.Tests> dotnet test
Restore complete (0.7s)
  CalcLibrary succeeded (0.3s) → C:\Users\KIIT\OneDrive\Desktop\NUnit handson\CalcLibrary\CalcLibrary\bin\Debug\netstandard2.0\CalcLibrary.dll
  CalcLibrary.Tests succeeded (0.4s) → bin\Debug\net9.0\CalcLibrary.Tests.dll
NUnit Adapter 5.0.0.0: Test execution started
Running all tests in C:\Users\KIIT\OneDrive\Desktop\NUnit handson\CalcLibrary\CalcLibrary.Tests\bin\Debug\net9.0\CalcLibrary.Tests.dll
   NUnit3TestExecutor discovered 9 of 9 NUnit test cases using Current Discovery mode, Non-Explicit run
NUnit Adapter 5.0.0.0: Test execution complete
  CalcLibrary.Tests test succeeded (1.9s)

Test summary: total: 9, failed: 0, succeeded: 9, skipped: 0, duration: 1.8s
Build succeeded in 4.3s
PS C:\Users\KIIT\OneDrive\Desktop\NUnit handson\CalcLibrary\CalcLibrary.Tests>
```