**NUnit HandsOn**

1. **1 -)**
2. **Meaning of Unit Testing and Its Difference from Functional Testing**

**Unit Testing**:

* Unit testing is a type of software testing where individual components or functions of a software are tested in isolation.
* It focuses on the **smallest testable part** of the code like a method or function.
* Mocking is often used to **simulate dependencies**, so the unit being tested is isolated from external systems like databases or APIs.

**Functional Testing**:

* Functional testing is a **higher-level** test that validates the software system against the functional requirements/specifications.
* It tests **end-to-end scenarios** including integration between components.

1. **Types of Testing**

· **Unit Testing**: Tests individual methods or components.

· **Functional Testing**: Validates the system works as intended (business logic).

· **Automated Testing**: Uses tools or scripts to run tests automatically.

· **Performance Testing**: Checks system performance under load (e.g., response time, throughput).

1. **Understand the benefit of automated testing**

· **Speed**: Tests run faster than manual tests.

· **Repeatability**: Can run frequently and consistently.

· **Coverage**: Tests can cover a large codebase efficiently.

· **Early Bug Detection**: Catches regressions early in development.

· **Cost-Effective**: Saves time and resources in the long term.

1. **Explain what is loosly coupled & testable design**

**Loosely Coupled**:

* Components do **not heavily depend** on one another.
* We can **swap dependencies** (e.g., with mock objects) easily.

**Testable Design**:

* Code is structured in a way that makes it easy to test.
* **Interfaces**, **dependency injection**, and **mocking frameworks are used**.

1. **Write your first testing program to validate a calculator addition operation**

using NUnit.Framework;

[TestFixture]

public class CalculatorTests

{

[Test]

public void Add\_TwoNumbers\_ReturnsSum()

{

var calc = new SimpleCalculator();

double result = calc.Addition(3, 4);

Assert.AreEqual(7, result);

}

1. }**Understand the need of [SetUp], [TearDown] & [Ignore] attributes.**

**· [SetUp]:** Runs **before** each test. Good for initializing resources.

**· [TearDown]:** Runs **after** each test. Good for cleanup.

**· [Ignore]:** Temporarily skip a test (e.g., if the feature isn’t ready).

1. **Explain the benefit of writing parameterised test cases.**

· Reduces **code duplication**.

· Allows testing **multiple input-output combinations** easily.

· Increases **test coverage** with minimal code.

**Main Test for (CalcLibrary.Tests) :-**

using CalcLibrary;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using NUnit.Framework;

using System;

namespace CalcLibrary.Tests

{

[TestFixture]

public class CalculatorTests

{

private SimpleCalculator \_calculator;

[SetUp]

public void Setup()

{

\_calculator = new SimpleCalculator();

}

[TearDown]

public void TearDown()

{

\_calculator = null;

}

[TestCase(10, 5, 15)]

[TestCase(0, 0, 0)]

[TestCase(-3, -6, -9)]

[TestCase(8, 12, 20)]

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

{

double result = \_calculator.Addition(a, b);

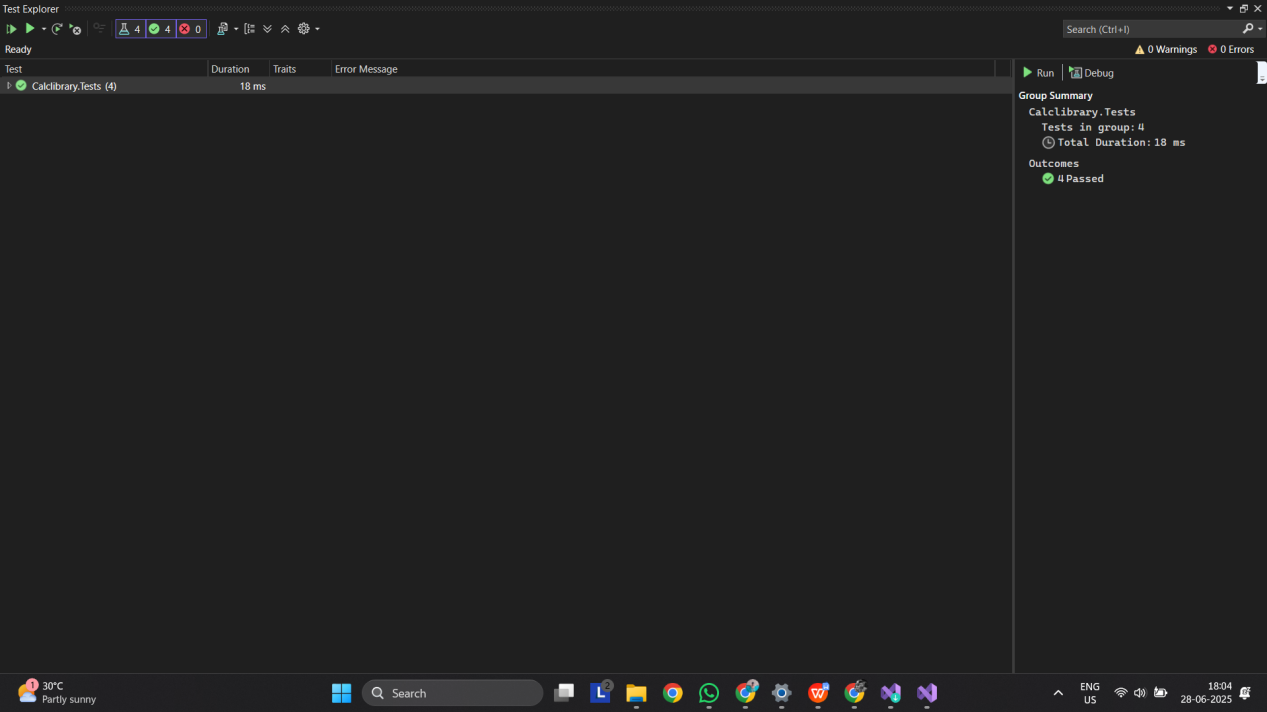
NUnit.Framework.Assert.That(result, Is.EqualTo(expected));

}

}

}

**Test Results:**



**Q. 2 -)**

## 1. Demonstration on Parameterized Test Cases Using an Example

### Implementation:

**Calculator Method (SimpleCalculator.cs)**

csharp

CopyEdit

public double Addition(double a, double b)

{

return a + b;

}**Test Case (CalculatorTests.cs)**

csharp

CopyEdit

[TestCase(3, 4, 7)]

[TestCase(10, 20, 30)]

[TestCase(-5, 5, 0)] public void Addition\_ValidInputs\_ReturnsCorrectResult(double a, double b, double expected)

{

double result = \_calculator.Addition(a, b);

Assert.AreEqual(expected, result);

}

**Explanation:**  
Parameterized tests reduce code duplication and allow testing of multiple input-output combinations efficiently. The [TestCase] attribute is used to run the same test logic with various inputs and expected outputs.

## 2. Demonstrate on Testing Methods That Return a Value

### Implementation:

**Subtraction Method (SimpleCalculator.cs)**

csharp

CopyEdit

public double Subtraction(double a, double b)

{

return a - b;

}

**Test Case (CalculatorTests.cs)**

csharp

CopyEdit

[TestCase(10, 5, 5)]

[TestCase(0, 0, 0)]

[TestCase(-3, -6, 3)]

[TestCase(8, 12, -4)] public void Subtraction\_ValidInputs\_ReturnsCorrectResult(double a, double b, double expected)

{

double result = \_calculator.Subtraction(a, b);

Assert.AreEqual(expected, result);

}

**Explanation:**  
This demonstrates how to test a return value using Assert.AreEqual.

## 3. Demonstrate on Testing Methods That Throw an Exception

### Implementation:

**Division Method (SimpleCalculator.cs)**

using System;

csharp

CopyEdit

public double Divide(double a, double b)

{

if (b == 0)

throw new ArgumentException("Cannot divide by zero");

return a / b;

}

**Test Case with Exception Handling (CalculatorTests.cs)**

csharp

CopyEdit

[Test]public void Divide\_ByZero\_ShouldThrowException()

{

try

{

\_calculator.Divide(10, 0);

Assert.Fail("Division by zero");

}

catch (ArgumentException ex)

{

Assert.That(ex.Message, Is.EqualTo("Cannot divide by zero"));

Assert.That(ex, Is.TypeOf<ArgumentException>());

}

**Explanation:**  
This demonstrates how to test error conditions using try-catch, Assert.Fail, and verifying the exception type and message.

## 4. Demonstrate on Testing Void Methods

### Implementation:

**Class Methods (SimpleCalculator.cs)**

csharp

CopyEdit

private double result;

public void AllClear()

{

result = 0;

}

public double AddAndStore(double a, double b)

{

result = a + b;

return result;

}

public double GetResult() => result;

**Test Case (CalculatorTests.cs)**

using static System.Net.Mime.MediaTypeNames;

csharp

CopyEdit

[Test]public void AddAndClear\_ResultShouldBeZeroAfterClear()

{

\_calculator.AddAndStore(5, 5);

Assert.AreEqual(10, \_calculator.GetResult());

\_calculator.AllClear();

Assert.AreEqual(0, \_calculator.GetResult());

}

**Explanation:**  
This test demonstrates how to validate the result of a method that changes internal state and has no return value.

## 5. Explain Why Testing a Private Method Is Not Beneficial

**Explanation:**  
Testing private methods is not a good practice because it breaks object-oriented principles such as encapsulation. Private methods are implementation details and should be tested indirectly via public methods. This avoids tight coupling between test logic and internal code structure and allows greater flexibility during refactoring.

## 6. Explain Mocking Framework and Its Usage

**Explanation:**  
A mocking framework like Moq is used to simulate the behavior of dependencies such as databases, web services, or file systems. This isolates the unit under test and allows testing of components independently from their environment.

**Interface (IMailSender.cs)**

using System;

using static System.Net.Mime.MediaTypeNames;

csharp

CopyEdit

public interface IMailSender

{

bool SendMail(string to, string message);

}

**Dependent Class (CustomerComm.cs)**

csharp

CopyEdit

public class CustomerComm

{

private readonly IMailSender \_mailSender;

public CustomerComm(IMailSender mailSender)

{

\_mailSender = mailSender;

}

public bool SendMailToCustomer()

{

return \_mailSender.SendMail("cust123@abc.com", "Hello!");

}

}

**Test Using Moq (CustomerCommTests.cs)**

csharp

CopyEdit

using Moq;

[Test] public void SendMailToCustomer\_ShouldReturnTrue\_WhenMailIsSent()

{

var mockSender = new Mock<IMailSender>();

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

var comm = new CustomerComm(mockSender.Object);

bool result = comm.SendMailToCustomer();

Assert.IsTrue(result);

}

**Explanation:**  
In this test, Moq is used to create a fake implementation of the IMailSender interface. This makes the test independent of the actual email sending logic.

**Main Test Code (CalcLibrary.Tests\_New) :-**

using NUnit.Framework;

using CalcLibrary;

using System;

namespace CalcLibrary.Tests\_New

{

[TestFixture]

public class MathLibraryTests

{

private MathLibrary \_math;

[SetUp]

public void Setup()

{

\_math = new MathLibrary();

}

[TestCase(10, 5, 5)]

[TestCase(0, 0, 0)]

[TestCase(-4, -4, 0)]

public void Test\_Subtract(int a, int b, int expected)

{

Assert.AreEqual(expected, \_math.Subtract(a, b));

}

[TestCase(2, 3, 6)]

[TestCase(-2, 4, -8)]

[TestCase(0, 9, 0)]

public void Test\_Multiply(int a, int b, int expected)

{

Assert.AreEqual(expected, \_math.Multiply(a, b));

}

[TestCase(10, 2, 5)]

[TestCase(9, 3, 3)]

public void Test\_Divide(int a, int b, int expected)

{

Assert.AreEqual(expected, \_math.Divide(a, b));

}

[Test]

public void Test\_Divide\_By\_Zero()

{

try

{

\_math.Divide(5, 0);

Assert.Fail("Division by zero");

}

catch (ArgumentException ex)

{

Assert.That(ex.Message, Is.EqualTo("Cannot divide by zero"));

}

}

[Test]

public void Test\_Add\_And\_Clear()

{

\_math.Add(7, 3);

Assert.AreEqual(10, \_math.GetResult);

\_math.AllClear();

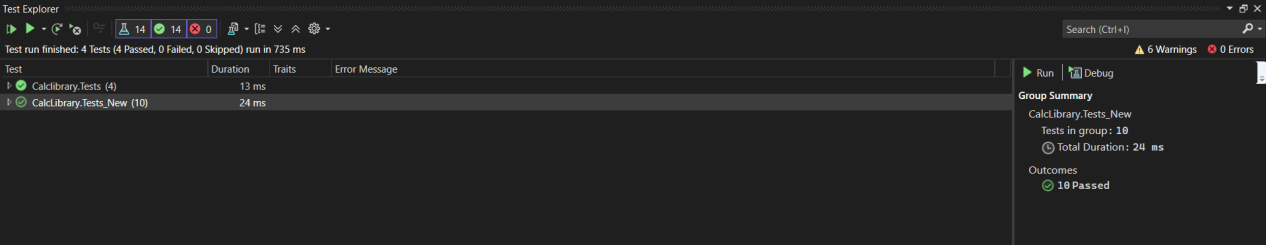
Assert.AreEqual(0, \_math.GetResult);

}

}

}

**Test Result :**



**Q . 3 -)**

**Main Test Code (UtilLib.Tests) :-**

using NUnit.Framework;

using UtilLib;

using System;

namespace UtilLib.Tests

{

public class UrlHostNameParserTests

{

private UrlHostNameParser parser;

[SetUp]

public void Setup()

{

parser = new UrlHostNameParser();

}

[TestCase("https://www.example.com/path", ExpectedResult = "www.example.com")]

[TestCase("https://google.com/search", ExpectedResult = "google.com")]

public string ParseHostName\_ShouldReturnCorrectHostName(string url)

{

return parser.ParseHostName(url);

}

[Test]

public void ParseHostName\_ShouldReturnHost()

{

string result = parser.ParseHostName("https://test.com/page");

Assert.AreEqual("test.com", result);

}

[Test]

public void DummyVoidMethod\_Test()

{

Assert.Pass("Dummy void test executed.");

}

[Test]

public void ParseHostName\_ShouldThrowExceptionForInvalidUrl()

{

try

{

parser.ParseHostName("invalid-url");

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

}

catch (FormatException ex)

{

Assert.That(ex.Message, Is.EqualTo("Url is not in correct format"));

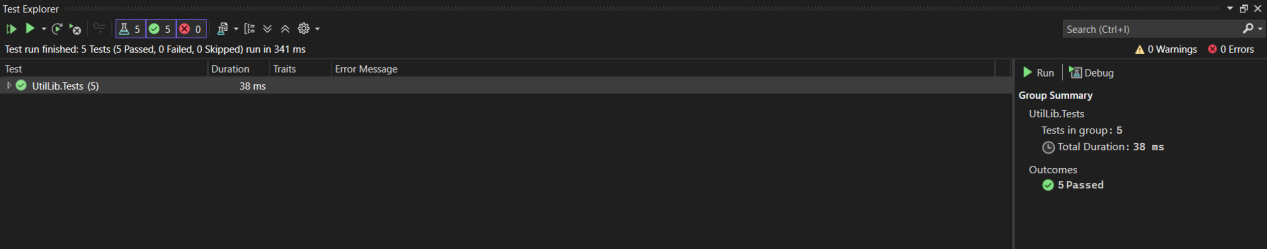
}

}

}

}

**Test Result :**



**Q . 4 -)**

**Main Test Code (AccountsManagerLib.Tests) :-**

using NUnit.Framework;

using AccountsManagerLib;

using System;

namespace AccountsManagerLib.Tests

{

[TestFixture]

public class AccountsManagerTests

{

private AccountsManager \_manager;

[SetUp]

public void Setup()

{

\_manager = new AccountsManager();

}

[Test]

public void ValidateUser\_ValidUser11Credentials\_ReturnsWelcomeMessage()

{

var result = \_manager.ValidateUser("user\_11", "secret@user11");

Assert.That(result, Is.EqualTo("Welcome user\_11!!!"));

}

[Test]

public void ValidateUser\_ValidUser22Credentials\_ReturnsWelcomeMessage()

{

var result = \_manager.ValidateUser("user\_22", "secret@user22");

Assert.That(result, Is.EqualTo("Welcome user\_22!!!"));

}

[Test]

public void ValidateUser\_InvalidCredentials\_ReturnsInvalidMessage()

{

var result = \_manager.ValidateUser("wrong\_user", "wrong\_pass");

Assert.That(result, Is.EqualTo("Invalid user id/password"));

}

[Test]

public void ValidateUser\_EmptyUserId\_ThrowsFormatException()

{

var ex = Assert.Throws<FormatException>(() =>

{

\_manager.ValidateUser("", "secret@user11");

});

Assert.That(ex.Message, Is.EqualTo("Both user id and password are mandatory"));

}

[Test]

public void ValidateUser\_EmptyPassword\_ThrowsFormatException()

{

var ex = Assert.Throws<FormatException>(() =>

{

\_manager.ValidateUser("user\_11", "");

});

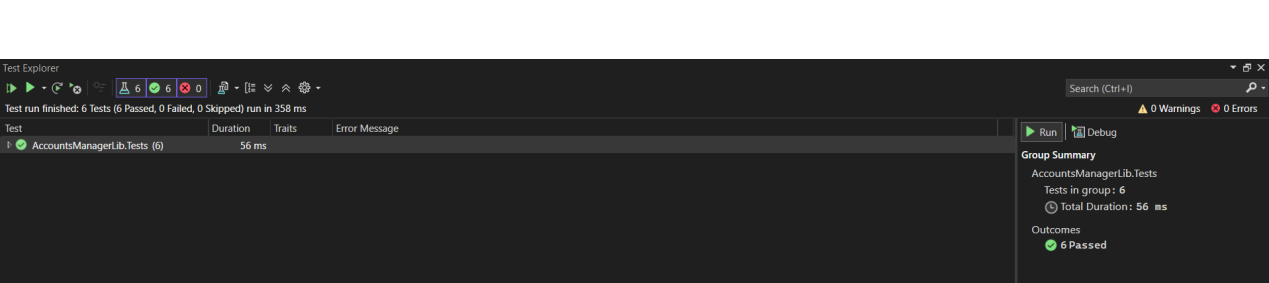
Assert.That(ex.Message, Is.EqualTo("Both user id and password are mandatory"));

}

}

}

**Test Result :**



**Q . 5 -)**

**Main Test Code (CollectionsLib.Test) :-**

using NUnit.Framework;

using CollectionsLib;

using System.Linq;

namespace CollectionsLib.Test

{

[TestFixture]

public class EmployeeManagerTests

{

private EmployeeManager \_manager;

[SetUp]

public void Setup()

{

\_manager = new EmployeeManager();

}

[Test]

public void GetEmployees\_ShouldNotContainNulls()

{

var employees = \_manager.GetEmployees();

Assert.That(employees, Is.All.Not.Null);

}

[Test]

public void GetEmployees\_ContainsEmployeeWithId100()

{

var employees = \_manager.GetEmployees();

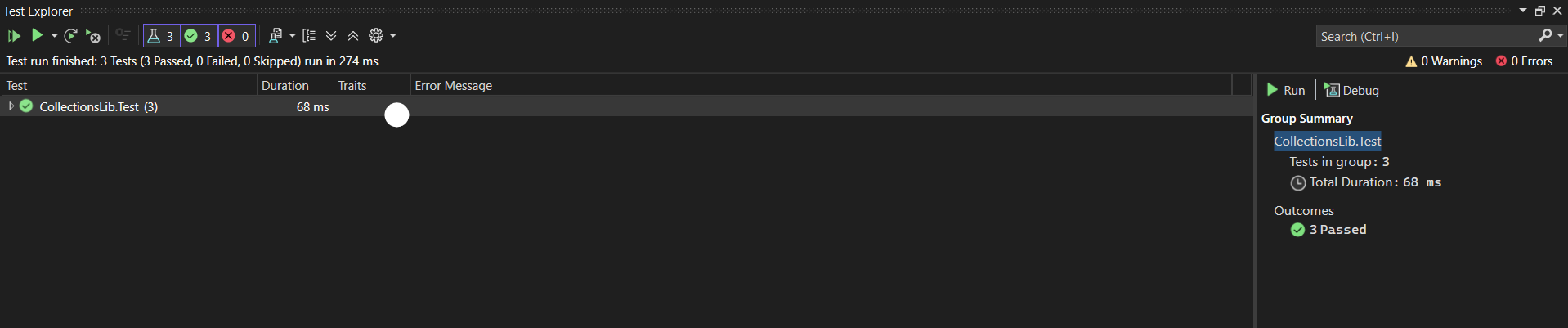
Assert.That(employees.Any(e => e.EmpId == 100), Is.True);

}

}

}

**Test Result :**





**Q . 6 -)**

**Main Test Code (FourSeasonsLib.Tests) :-**

using NUnit.Framework;

using FourSeasonsLib;

using System.Collections;

using System;

namespace FourSeasonsLib.Tests

{

[TestFixture]

public class SeasonDetectorTests

{

private SeasonDetector \_detector;

[SetUp]

public void Setup()

{

\_detector = new SeasonDetector();

}

[Test, TestCaseSource(nameof(MonthSeasonPairs))]

public void GetSeason\_ValidMonth\_ReturnsCorrectSeason(string month, string expectedSeason)

{

var result = \_detector.GetSeason(month);

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

}

[Test]

public void GetSeason\_InvalidMonth\_ThrowsArgumentException()

{

Assert.Throws<ArgumentException>(() => \_detector.GetSeason("hello"));

}

static IEnumerable MonthSeasonPairs

{

get

{

yield return new TestCaseData("February", "Spring");

yield return new TestCaseData("March", "Spring");

yield return new TestCaseData("April", "Summer");

yield return new TestCaseData("May", "Summer");

yield return new TestCaseData("June", "Summer");

yield return new TestCaseData("July", "Monsoon");

yield return new TestCaseData("August", "Monsoon");

yield return new TestCaseData("September", "Autumn");

yield return new TestCaseData("October", "Autumn");

yield return new TestCaseData("November", "Autumn");

yield return new TestCaseData("December", "Winter");

yield return new TestCaseData("January", "Winter");

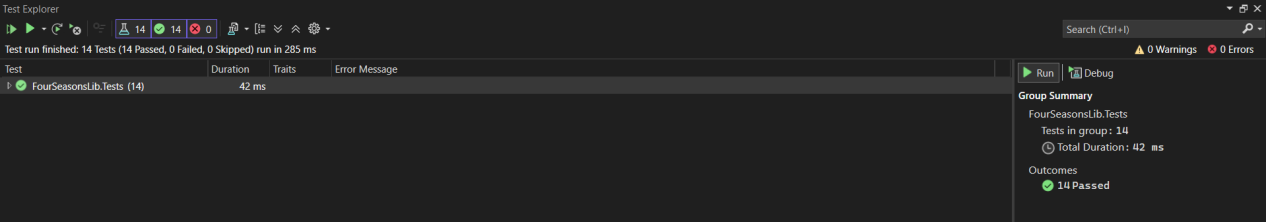
}

}

}

}

**Test Result :**



**Q . 7 -)**

**Main Test Code (LeapYearCalculatorLib.Tests) :-**

using NUnit.Framework;

using LeapYearCalculatorLib;

namespace LeapYearCalculatorLib.Tests

{

[TestFixture]

public class LeapYearCalculatorTests

{

private LeapYearCalculator \_calculator;

[SetUp]

public void Setup()

{

\_calculator = new LeapYearCalculator();

}

[TestCase(2000, 1)]

[TestCase(2016, 1)]

[TestCase(2400, 1)]

public void IsLeapYear\_LeapYears\_Returns1(int year, int expected)

{

var result = \_calculator.IsLeapYear(year);

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

}

[TestCase(1900, 0)]

[TestCase(2019, 0)]

[TestCase(2100, 0)]

public void IsLeapYear\_NonLeapYears\_Returns0(int year, int expected)

{

var result = \_calculator.IsLeapYear(year);

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

}

[TestCase(1500, -1)]

[TestCase(10000, -1)]

[TestCase(0, -1)]

public void IsLeapYear\_InvalidYears\_ReturnsMinus1(int year, int expected)

{

var result = \_calculator.IsLeapYear(year);

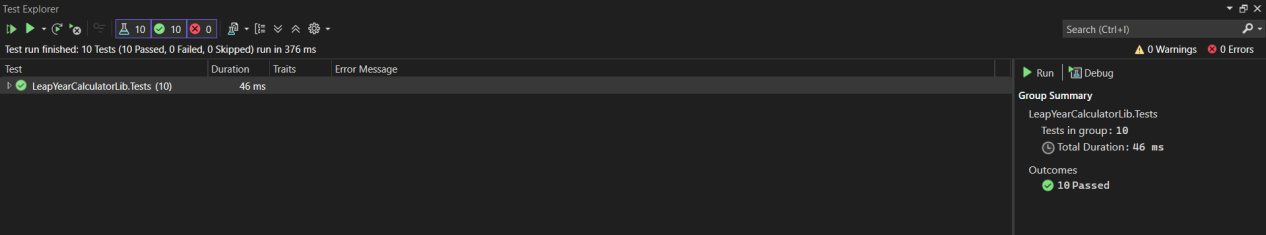
Assert.That(result, Is.EqualTo(expected));

}

}

}

**Test Result :**



**Q . 8 -)**

**Main Test Code (UserManagerLib.Tests) :-**

using NUnit.Framework;

using UserManagerLib;

using System;

using System.Collections;

namespace UserManagerLib.Tests

{

[TestFixture]

public class UserTests

{

private User \_user;

[SetUp]

public void Setup()

{

\_user = new User();

}

[Test, TestCaseSource(nameof(NullOrEmptyPANInputs))]

public void ValidatePANCardNumber\_NullOrEmpty\_ThrowsNullReferenceException(string input)

{

var ex = Assert.Throws<NullReferenceException>(() => \_user.ValidatePANCardNumber(input));

Assert.That(ex.Message, Is.EqualTo("Invalid Pan Card Number"));

}

[TestCase("ABCDE123")]

[TestCase("ABCDE123456")]

public void ValidatePANCardNumber\_InvalidLength\_ThrowsFormatException(string input)

{

var ex = Assert.Throws<FormatException>(() => \_user.ValidatePANCardNumber(input));

Assert.That(ex.Message, Is.EqualTo("Pan Card Number Should contain only 10 characters"));

}

[Test]

public void ValidatePANCardNumber\_ValidInput\_ReturnsValid()

{

var result = \_user.ValidatePANCardNumber("ABCDE1234Z");

Assert.That(result, Is.EqualTo("Valid"));

}

static IEnumerable NullOrEmptyPANInputs()

{

yield return new TestCaseData((string)null);

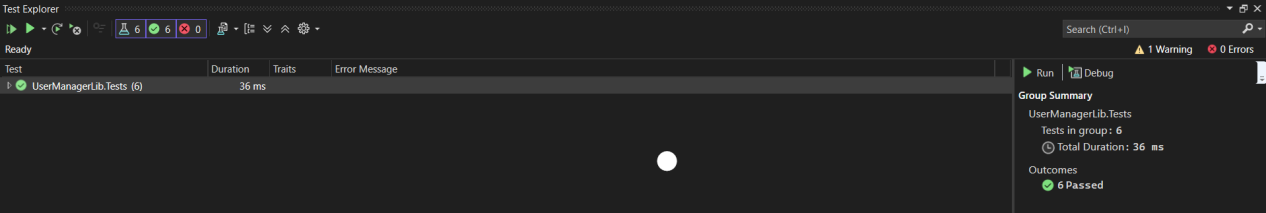
yield return new TestCaseData("");

}

}

}

**Test Result :**





**Q . 9 -)**

**Main Test Code (ConverterLib.Tests) :-**

using NUnit.Framework;

using ConverterLib;

namespace ConverterLib.Tests

{

[TestFixture]

public class ConverterTests

{

private Converter \_converter;

[SetUp]

public void Setup()

{

\_converter = new Converter();

}

[TestCase(0, 273.15)]

[TestCase(100, 373.15)]

[TestCase(-273.15, 0)]

public void CelsiusToKelvin\_ReturnsExpected(double celsius, double expectedKelvin)

{

var result = \_converter.CelsiusToKelvin(celsius);

Assert.That(result, Is.EqualTo(expectedKelvin).Within(0.01));

}

[TestCase(0, 0)]

[TestCase(1, 2.20462)]

[TestCase(10, 22.0462)]

public void KilogramToPound\_ReturnsExpected(double kg, double expectedPound)

{

var result = \_converter.KilogramToPound(kg);

Assert.That(result, Is.EqualTo(expectedPound).Within(0.01));

}

[TestCase(0, 0)]

[TestCase(1, 0.621371)]

[TestCase(5, 3.106855)]

public void KilometerToMile\_ReturnsExpected(double km, double expectedMile)

{

var result = \_converter.KilometerToMile(km);

Assert.That(result, Is.EqualTo(expectedMile).Within(0.01));

}

[TestCase(0, 0)]

[TestCase(1, 0.264172)]

[TestCase(10, 2.64172)]

public void LiterToGallon\_ReturnsExpected(double liter, double expectedGallon)

{

var result = \_converter.LiterToGallon(liter);

Assert.That(result, Is.EqualTo(expectedGallon).Within(0.01));

}

[TestCase(1, 0.85)]

[TestCase(10, 8.5)]

[TestCase(100, 85)]

public void USDToEuro\_ReturnsExpected(double usd, double expectedEuro)

{

var result = \_converter.USDToEuro(usd);

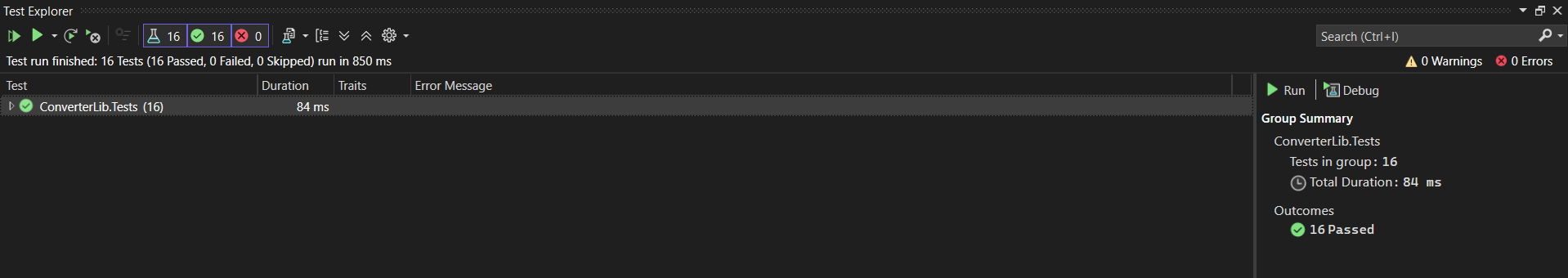
Assert.That(result, Is.EqualTo(expectedEuro).Within(0.01));

}

}

}

**Test Result :**



**\*\* Moq HandsOn \*\***

#### 1. ****File: IMailSender.cs****

namespace CustomerCommLib

{

public interface IMailSender

{

bool SendMail(string toAddress, string message);

}

}

#### 2. ****File: 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("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);

return true;

}

}

}

#### 3. ****File: CustomerComm.cs****

namespace CustomerCommLib

{

public class CustomerComm

{

private readonly IMailSender \_mailSender;

public CustomerComm(IMailSender mailSender)

{

\_mailSender = mailSender;

}

public bool SendMailToCustomer()

{

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

}

}

}

**Project 2: CustomerComm.Tests**

1. **File: CustomerCommTests.cs**

using NUnit.Framework;

using Moq;

using CustomerCommLib;

namespace CustomerComm.Tests

{

[TestFixture]

public class CustomerCommTests

{

private Mock<IMailSender> \_mockMailSender;

private CustomerCommLib.CustomerComm \_customerComm;

[OneTimeSetUp]

public void Init()

{

\_mockMailSender = new Mock<IMailSender>();

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

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

}

[Test]

public void SendMailToCustomer\_ShouldReturnTrue\_WhenMailIsSent()

{

bool result = \_customerComm.SendMailToCustomer();

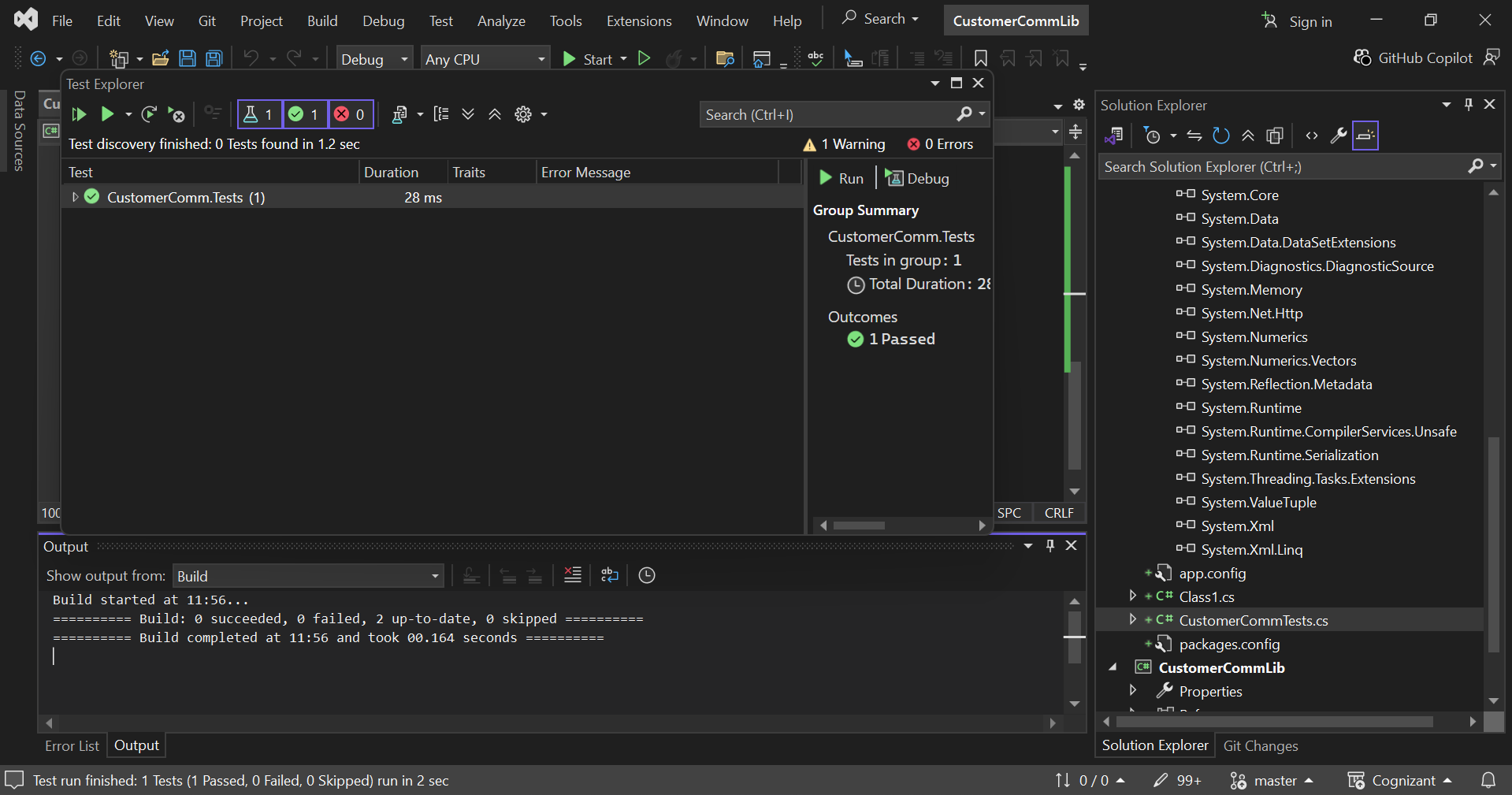
Assert.That(result, Is.True);

}

}

}

**Output**

****