NUnit Testing

1. NUnit-Handson (CalcLibrary )

The testing code for CalculatorTests:-

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CalcLibrary.test

{

[TestFixture]

public class CalculatorTests

{

private SimpleCalculator \_calculator;

[SetUp]

public void Setup()

{

\_calculator = new SimpleCalculator();

}

[TearDown]

public void Cleanup()

{

\_calculator = null;

}

[Test]

[TestCase(2, 3, 5)]

[TestCase(-1, 1, 0)]

[TestCase(0, 0, 0)]

public void Add\_WhenCalled\_ReturnsExpectedSum(int a, int b, int expected)

{

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

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

}

[Test]

[Ignore("Temporarily skipping this test")]

public void ThisTestIsIgnored()

{

Assert.Fail("You should not see this test run.");

}

}

}

The testing code for DivisionTestAndExceptionHandling:-

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CalcLibrary.test

{

[TestFixture]

public class DivisionTestAndExceptionHandling

{

private SimpleCalculator \_calculator;

[SetUp]

public void Setup()

{

\_calculator = new SimpleCalculator();

}

[TearDown]

public void Cleanup()

{

\_calculator = null;

}

[Test]

[TestCase(10, 2, 5)]

[TestCase(9, 3, 3)]

public void Divide\_WhenCalled\_ReturnsExpectedResult(int a, int b, int expected)

{

var result = \_calculator.Division(a, b);

Assert.AreEqual(expected, result);

}

[Test]

public void Divide\_ByZero\_ThrowsException()

{

try

{

var result = \_calculator.Division(10, 0);

Assert.Fail("Division by zero"); // If no exception is thrown

}

catch (ArgumentException ex)

{

Assert.That(ex.Message, Is.EqualTo("Second Parameter Can't be Zero")); // or your exact message

}

}

}

}

1. NUnit-Handson (CalcLibrary )

The testing code for GetResultAndAllclearTest:-

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CalcLibrary.test

{

[TestFixture]

public class GetResultAndAllclearTest

{

private SimpleCalculator \_calculator;

[SetUp]

public void Setup()

{

\_calculator = new SimpleCalculator();

}

[Test]

public void TestAddAndClear()

{

// Step 1: Add two numbers

\_calculator.Addition(10, 5);

// Step 2: Verify result is correct

Assert.AreEqual(15, \_calculator.GetResult, "Addition result mismatch");

// Step 3: Clear the result

\_calculator.AllClear();

// Step 4: Verify result is reset to 0

Assert.AreEqual(0, \_calculator.GetResult, "Result was not cleared correctly");

}

}

}

The testing code for MultiplicationTest:-

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CalcLibrary.test

{

[TestFixture]

public class MultiplicationTest

{

private SimpleCalculator \_calculator;

[SetUp]

public void Setup()

{

\_calculator = new SimpleCalculator();

}

[TearDown]

public void Cleanup()

{

\_calculator = null;

}

[Test]

[TestCase(2, 3, 6)]

[TestCase(-2, 3, -6)]

[TestCase(0, 5, 0)]

public void Multiply\_WhenCalled\_ReturnsExpectedResult(int a, int b, int expected)

{

var result = \_calculator.Multiplication(a,b);

Assert.AreEqual(expected, result);

}

[Test]

[Ignore("Temporarily skipping this test")]

public void ThisTestIsIgnored()

{

Assert.Fail("You should not see this test run.");

}

}

}

The testing code for SubstractionTest:-

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CalcLibrary.test

{

[TestFixture]

public class SubstractionTest

{

private SimpleCalculator \_calculator;

[SetUp]

public void Setup()

{

\_calculator = new SimpleCalculator();

}

[TearDown]

public void Cleanup()

{

\_calculator = null;

}

[Test]

[TestCase(10, 5, 5)]

[TestCase(0, 0, 0)]

[TestCase(20, 30, -10)]

public void Subtract\_WhenCalled\_ReturnsExpectedResult(int a, int b, int expected)

{

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

Assert.AreEqual(expected, result);

}

[Test]

[Ignore("Temporarily skipping this test")]

public void ThisTestIsIgnored()

{

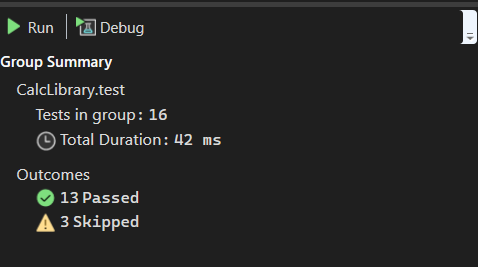
Assert.Fail("You should not see this test run.");

}

}

}

The output:-



1. NUnit-Handson (UtilLib)

The testing code for UrlHostNameParserTests:-

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace UtilLib.Test

{

[TestFixture]

public class UrlHostNameParserTests

{

private UrlHostNameParser \_parser;

[SetUp]

public void Setup()

{

\_parser = new UrlHostNameParser();

}

[Test]

public void ParseHostName\_HttpsUrl\_ReturnsHostName()

{

// Arrange

string input = "https://www.example.com/page";

string expected = "www.example.com";

// Act

string result = \_parser.ParseHostName(input);

// Assert

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

}

[Test]

public void ParseHostName\_HttpUrl\_ReturnsHostName()

{

string input = "http://test.domain.org/home";

string expected = "test.domain.org";

string result = \_parser.ParseHostName(input);

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

}

[Test]

public void ParseHostName\_UnsupportedProtocol\_ThrowsFormatException()

{

string input = "ftp://files.example.com/download";

var ex = Assert.Throws<FormatException>(() => \_parser.ParseHostName(input));

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

}

[Test]

public void ParseHostName\_NullInput\_ThrowsNullReferenceException()

{

string input = null;

Assert.Throws<NullReferenceException>(() => \_parser.ParseHostName(input));

}

[Test]

public void ParseHostName\_EmptyInput\_ThrowsIndexOutOfRangeException()

{

string input = "";

var ex = Assert.Throws<FormatException>(() => \_parser.ParseHostName(input));

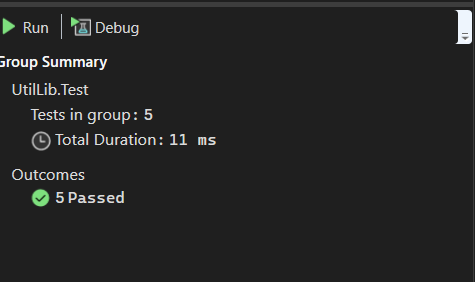
Assert.That(ex.Message, Is.EqualTo("Url is not in correct format"));

}

}

}

The output:-



1. NUnit-Handson (AccountsManagerTests)

The testing code for AccountsManagerTests:-

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AccountsManagerLib.Test

{

[TestFixture]

public class AccountsManagerTests

{

private AccountsManager \_manager;

[SetUp]

public void SetUp()

{

\_manager = new AccountsManager();

}

[Test]

public void ValidateUser\_ValidUser11Credentials\_ReturnsWelcomeMessage()

{

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

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

}

[Test]

public void ValidateUser\_ValidUser22Credentials\_ReturnsWelcomeMessage()

{

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

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

}

[Test]

public void ValidateUser\_InvalidCredentials\_ReturnsInvalidMessage()

{

string result = \_manager.ValidateUser("user\_11", "wrongpass");

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

}

[Test]

public void ValidateUser\_EmptyUserId\_ThrowsFormatException()

{

var ex = Assert.Throws<FormatException>(() => \_manager.ValidateUser("", "anyPassword"));

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]

public void ValidateUser\_NullValues\_ThrowsFormatException()

{

var ex = Assert.Throws<FormatException>(() => \_manager.ValidateUser(null, null));

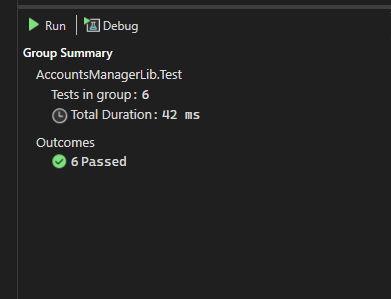
Assert.That(ex.Message, Is.EqualTo("Both user id and password are mandatory"));

}

}

}

The output:-



1. NUnit-Handson (CollectionsLib)

The testing for EmployeeManagerTests:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CollectionsLib.Test

{

[TestFixture]

public class EmployeeManagerTests

{

private EmployeeManager \_manager;

[SetUp]

public void Setup()

{

\_manager = new EmployeeManager();

}

[Test]

public void GetEmployees\_NoNullValues\_AllItemsNotNull()

{

var employees = \_manager.GetEmployees();

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

}

[Test]

public void GetEmployees\_EmployeeWithId100Exists\_ReturnsTrue()

{

var employees = \_manager.GetEmployees();

bool exists = employees.Any(e => e.EmpId == 100);

Assert.That(exists, Is.True);

}

[Test]

public void GetEmployees\_AllEmployeesHaveUniqueEmpIds\_ListIsUnique()

{

var employees = \_manager.GetEmployees();

var uniqueEmpIds = employees.Select(e => e.EmpId).Distinct().Count();

Assert.That(uniqueEmpIds, Is.EqualTo(employees.Count));

}

[Test]

public void GetEmployees\_And\_GetEmployeesWhoJoinedInPreviousYears\_AreEqual\_ClassicAssert()

{

var list1 = \_manager.GetEmployees();

var list2 = \_manager.GetEmployeesWhoJoinedInPreviousYears();

CollectionAssert.AreEqual(list1, list2); // classic model

}

[Test]

public void GetEmployees\_And\_GetEmployeesWhoJoinedInPreviousYears\_AreEqual\_ConstraintModel()

{

var list1 = \_manager.GetEmployees();

var list2 = \_manager.GetEmployeesWhoJoinedInPreviousYears();

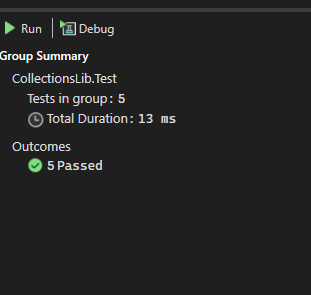
Assert.That(list1, Is.EqualTo(list2)); // constraint model

}

}

}

The output :-



1. NUnit-Handson (FourSeasonsLib)

The testing code for the SeasonTellerTests:-

using SeasonsLib;

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace SeasonLib.Test

{

[TestFixture]

public class SeasonTellerTests

{

private SeasonTeller \_seasonTeller;

[SetUp]

public void SetUp()

{

\_seasonTeller = new SeasonTeller();

}

// ✅ Test case source method

public static IEnumerable SeasonTestCases

{

get

{

yield return new TestCaseData("February").Returns("Spring");

yield return new TestCaseData("March").Returns("Spring");

yield return new TestCaseData("April").Returns("Summer");

yield return new TestCaseData("May").Returns("Summer");

yield return new TestCaseData("June").Returns("Summer");

yield return new TestCaseData("July").Returns("Monsoon");

yield return new TestCaseData("August").Returns("Monsoon");

yield return new TestCaseData("September").Returns("Monsoon");

yield return new TestCaseData("October").Returns("Autumn");

yield return new TestCaseData("November").Returns("Autumn");

yield return new TestCaseData("December").Returns("Winter");

yield return new TestCaseData("January").Returns("Winter");

yield return new TestCaseData("UnknownMonth").Returns("Invalid Season");

yield return new TestCaseData("").Returns("Invalid Season");

}

}

[Test, TestCaseSource(nameof(SeasonTestCases))]

public string DisplaySeasonBy\_ValidMonthName\_ReturnsExpectedSeason(string input)

{

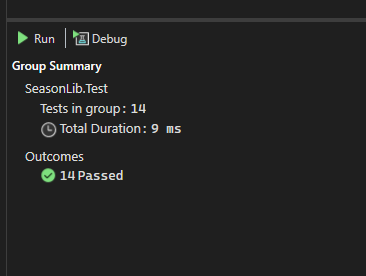
return \_seasonTeller.DisplaySeasonBy(input);

}

}

}

The output:-



1. NUnit-Handson (LeapYearCalculatorLib )

The testing code for the LeapYearCalculatorTests:-

namespace LeapYearCalculatorLib.Tests

{

[TestFixture]

public class LeapYearCalculatorTests

{

private LeapYearCalculator \_calculator;

[SetUp]

public void SetUp()

{

\_calculator = new LeapYearCalculator();

}

// Valid leap years → returns 1

[TestCase(2000, 1)]

[TestCase(2020, 1)]

[TestCase(2400, 1)]

// Valid non-leap years → returns 0

[TestCase(2019, 0)]

[TestCase(2100, 0)]

[TestCase(2023, 0)]

// Invalid years → returns -1

[TestCase(1700, -1)]

[TestCase(10000, -1)]

[TestCase(1500, -1)]

public void IsLeapYear\_YearInput\_ExpectedResult(int year, int expected)

{

int result = \_calculator.IsLeapYear(year);

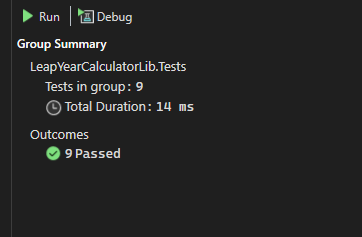
Assert.That(result, Is.EqualTo(expected));

}

}

}

The output:-



1. NUnit-Handson (UserManagerLib )

The testing code for the UserTests :-

namespace UserManagerLib.Tests

{

[TestFixture]

public class UserTests

{

private User \_user;

[SetUp]

public void SetUp()

{

\_user = new User();

}

[Test]

public void ValidatePANCardNumber\_ValidInput\_ReturnsValid()

{

string result = \_user.ValidatePANCardNumber("ABCDE1234F");

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

}

[Test]

public void ValidatePANCardNumber\_NullOrEmptyInput\_ThrowsNullReferenceException()

{

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

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

}

[Test]

public void ValidatePANCardNumber\_InvalidLength\_ThrowsFormatException()

{

var ex = Assert.Throws<FormatException>(() => \_user.ValidatePANCardNumber("ABC123"));

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

}

[Test]

public void CreateUser\_ValidPANCard\_UserCreatedSuccessfully()

{

var newUser = new User

{

FirstName = "John",

LastName = "Doe",

EmailId = "john.doe@example.com",

PANCardNo = "ABCDE1234F"

};

Assert.That(() => \_user.CreateUser(newUser), Throws.Nothing);

}

[Test]

public void CreateUser\_InvalidPANCard\_ThrowsFormatException()

{

var newUser = new User

{

FirstName = "John",

LastName = "Doe",

EmailId = "john.doe@example.com",

PANCardNo = "ABC"

};

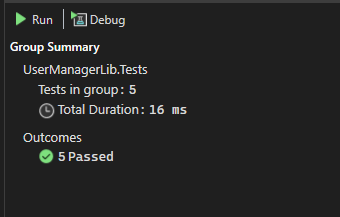
Assert.Throws<FormatException>(() => \_user.CreateUser(newUser));

}

}

}

The output:-



1. NUnit-Handson (ConverterLib )

The testing code for the ConverterTests:-

using NUnit.Framework;

using Moq;

using ConverterLib;

namespace ConverterLib.Tests

{

[TestFixture]

public class ConverterTests

{

[Test]

public void USDToEuro\_ValidDollarAmount\_ReturnsExpectedEuroValue()

{

// Arrange

var mockFeed = new Mock<IDollarToEuroExchangeRateFeed>();

mockFeed.Setup(x => x.GetExchangeRate()).Returns(0.85); // Mocked exchange rate

var converter = new Converter(mockFeed.Object);

double dollar = 100;

// Act

double result = converter.USDToEuro(dollar);

// Assert (Single Assertion Rule)

Assert.That(result, Is.EqualTo(85.0).Within(0.01));

}

[Test]

public void USDToEuro\_ZeroDollar\_ReturnsZero()

{

// Arrange

var mockFeed = new Mock<IDollarToEuroExchangeRateFeed>();

mockFeed.Setup(x => x.GetExchangeRate()).Returns(0.85);

var converter = new Converter(mockFeed.Object);

// Act

double result = converter.USDToEuro(0);

// Assert

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

}

[Test]

public void USDToEuro\_NegativeDollar\_ReturnsNegativeEuro()

{

// Arrange

var mockFeed = new Mock<IDollarToEuroExchangeRateFeed>();

mockFeed.Setup(x => x.GetExchangeRate()).Returns(0.75);

var converter = new Converter(mockFeed.Object);

// Act

double result = converter.USDToEuro(-40);

// Assert

Assert.That(result, Is.EqualTo(-30).Within(0.01));

}

}

}

The output:-

