**Cognizant Week 2 Assignment – NUnit Hands-On**

1. **Tested Library: AccountsManagerLib**

**Code: AccountsManagerTests.cs**

**using NUnit.Framework;**

**using AccountsManagerLib;**

**using System;**

**namespace NUnitMoqHandsOn**

**{**

**[TestFixture]**

**public class AccountsManagerTests**

**{**

**private AccountsManager \_manager;**

**[SetUp]**

**public void Setup()**

**{**

**\_manager = new AccountsManager();**

**}**

**[Test]**

**public void ValidateUser\_WithCorrectCredentials\_ReturnsWelcomeMessage()**

**{**

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

**Assert.AreEqual("Welcome user\_11!!!", result);**

**}**

**[Test]**

**public void ValidateUser\_WithIncorrectCredentials\_ReturnsInvalidMessage()**

**{**

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

**Assert.AreEqual("Invalid user id/password", result);**

**}**

**[Test]**

**public void ValidateUser\_WithEmptyUsername\_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\_WithEmptyPassword\_ThrowsFormatException()**

**{**

**var ex = Assert.Throws<FormatException>(() => \_manager.ValidateUser("user\_11", ""));**

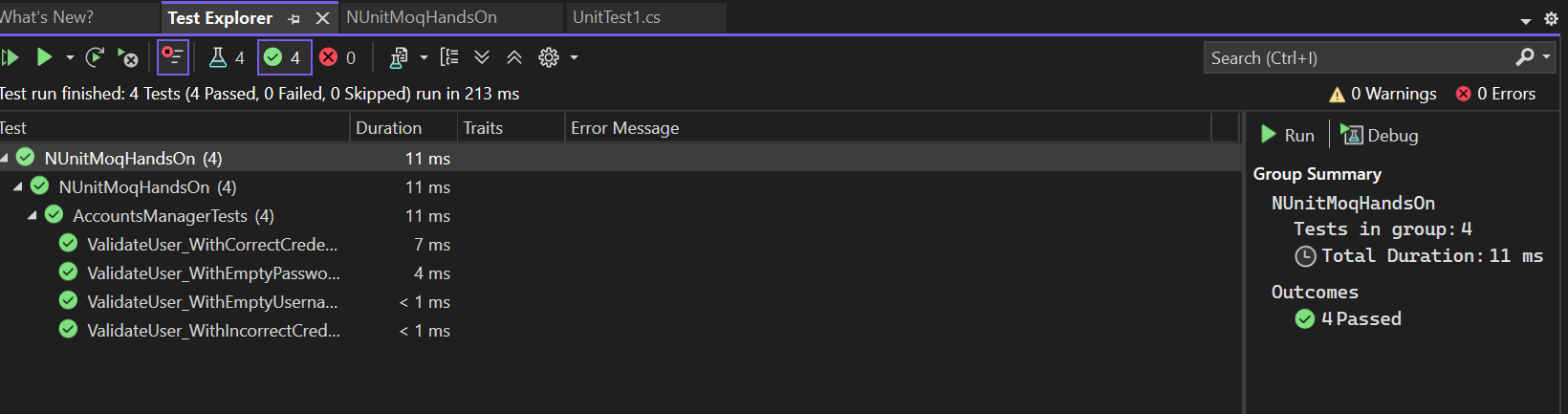
**Assert.That(ex.Message, Is.EqualTo("Both user id and password are mandatory"));**

**}**

**}**

**}**

**Output:**

****

1. **Tested Library: CalcLibrary**

****Code:** SimpleCalculatorTests.cs**

**using NUnit.Framework;**

**using CalcLibrary;**

**using System;**

**namespace NUnitMoqHandsOn**

**{**

**[TestFixture]**

**public class SimpleCalculatorTests**

**{**

**private SimpleCalculator calc;**

**[SetUp]**

**public void Setup()**

**{**

**calc = new SimpleCalculator();**

**}**

**[Test]**

**public void Addition\_WithTwoNumbers\_ReturnsCorrectSum()**

**{**

**double result = calc.Addition(10, 5);**

**Assert.AreEqual(15, result);**

**}**

**[Test]**

**public void Subtraction\_WithTwoNumbers\_ReturnsCorrectDifference()**

**{**

**double result = calc.Subtraction(10, 3);**

**Assert.AreEqual(7, result);**

**}**

**[Test]**

**public void Multiplication\_WithTwoNumbers\_ReturnsCorrectProduct()**

**{**

**double result = calc.Multiplication(4, 3);**

**Assert.AreEqual(12, result);**

**}**

**[Test]**

**public void Division\_WithValidNumbers\_ReturnsCorrectQuotient()**

**{**

**double result = calc.Division(10, 2);**

**Assert.AreEqual(5, result);**

**}**

**[Test]**

**public void Division\_ByZero\_ThrowsArgumentException()**

**{**

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

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

**}**

**[Test]**

**public void AllClear\_ResetsResultToZero()**

**{**

**calc.Addition(10, 5);**

**calc.AllClear();**

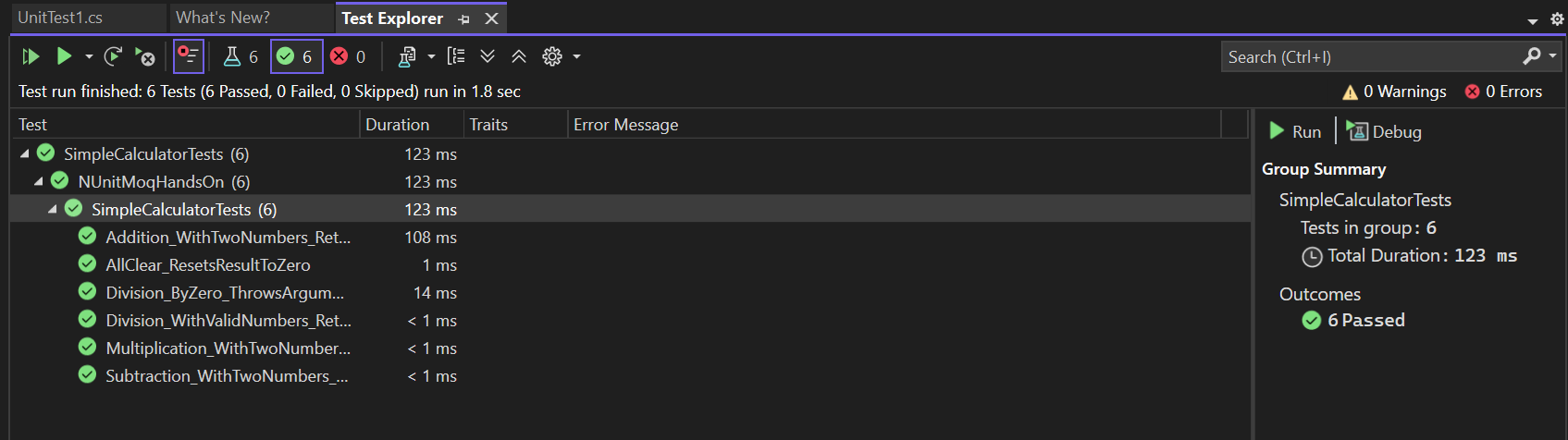
**Assert.AreEqual(0, calc.GetResult);**

**}**

**}**

**}**

**Output:**

****

1. **Tested Library: CollectionsLib**

****Code:** EmployeeManagerTests.cs**

using NUnit.Framework;

using CollectionsLib;

using System;

using System.Collections.Generic;

namespace NUnitMoqHandsOn

{

[TestFixture]

public class EmployeeManagerTests

{

private EmployeeManager manager;

[SetUp]

public void Setup()

{

manager = new EmployeeManager();

}

[Test]

public void GetEmployees\_ReturnsAllEmployees()

{

List<Employee> employees = manager.GetEmployees();

Assert.AreEqual(4, employees.Count);

}

[Test]

public void GetEmployeesWhoJoinedInPreviousYears\_ReturnsEmployeesWithDOJInPast()

{

var result = manager.GetEmployeesWhoJoinedInPreviousYears();

// All employees have DOJ in the past, so result count should be 4

Assert.AreEqual(4, result.Count);

foreach (var emp in result)

{

Assert.Less(emp.DOJ, DateTime.Now);

}

}

[Test]

public void GetEmployees\_ContainsExpectedEmployeeData()

{

var employees = manager.GetEmployees();

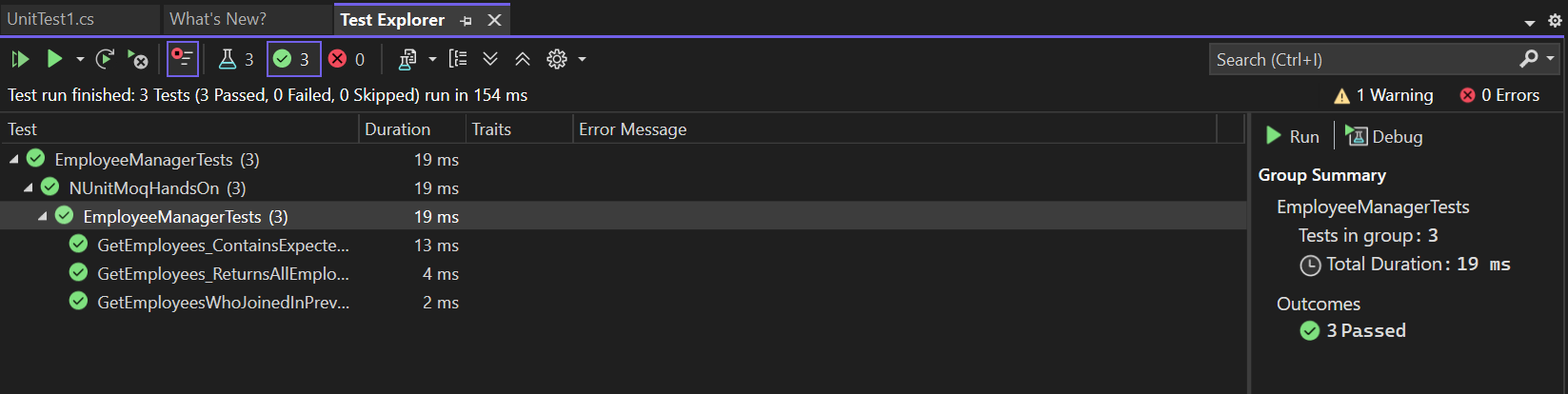
Assert.IsTrue(employees.Exists(e => e.EmpId == 100 && e.EmpName == "John"));

}

}

}

**Output:**



1. **Tested Library: UtilLib**

****Code:** UrlHostNameParserTests.cs**

using NUnit.Framework;

using System;

using UtilLib;

namespace UtilLibTests

{

[TestFixture]

public class UrlHostNameParserTests

{

private UrlHostNameParser \_parser;

[SetUp]

public void Setup()

{

\_parser = new UrlHostNameParser();

}

[Test]

public void ParseHostName\_ValidHttpUrl\_ReturnsHostName()

{

string url = "http://www.example.com/path";

string result = \_parser.ParseHostName(url);

Assert.AreEqual("www.example.com", result);

}

[Test]

public void ParseHostName\_ValidHttpsUrl\_ReturnsHostName()

{

string url = "https://sub.domain.org/page";

string result = \_parser.ParseHostName(url);

Assert.AreEqual("sub.domain.org", result);

}

[Test]

public void ParseHostName\_InvalidProtocol\_ThrowsFormatException()

{

string url = "ftp://example.com";

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

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

}

[Test]

public void ParseHostName\_EmptyString\_ThrowsIndexOutOfRangeException()

{

Assert.Throws<IndexOutOfRangeException>(() => \_parser.ParseHostName(""));

}

[Test]

public void ParseHostName\_NullInput\_ThrowsNullReferenceException()

{

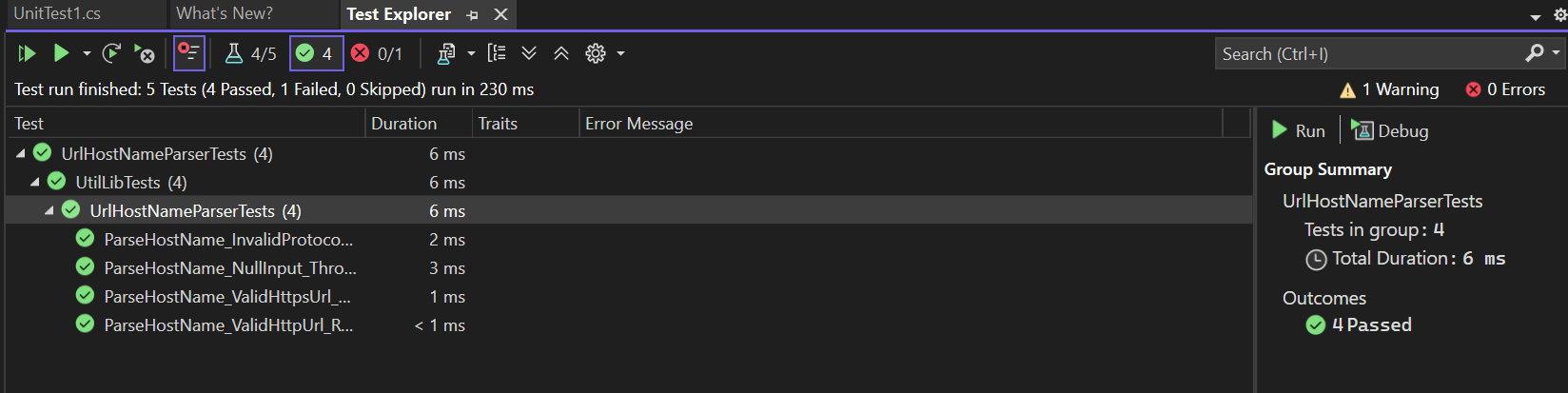
Assert.Throws<NullReferenceException>(() => \_parser.ParseHostName(null));

}

}

}

**Output:**



1. **Tested Library: UserManagerLib**

****Code:** UserTests.cs**

using NUnit.Framework;

using System;

using UserManagerLib;

namespace UserManagerLibTests

{

[TestFixture]

public class UserTests

{

private User \_user;

[SetUp]

public void Setup()

{

\_user = new User();

}

[Test]

public void ValidatePANCardNumber\_ValidPAN\_ReturnsValid()

{

string pan = "ABCDE1234F";

string result = \_user.ValidatePANCardNumber(pan);

Assert.AreEqual("Valid", result);

}

[Test]

public void ValidatePANCardNumber\_NullPAN\_ThrowsNullReferenceException()

{

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

}

[Test]

public void ValidatePANCardNumber\_EmptyPAN\_ThrowsNullReferenceException()

{

Assert.Throws<NullReferenceException>(() => \_user.ValidatePANCardNumber(""));

}

[Test]

public void ValidatePANCardNumber\_InvalidLengthPAN\_ThrowsFormatException()

{

string pan = "ABCDE123"; // less than 10 chars

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

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

}

[Test]

public void CreateUser\_WithValidPAN\_DoesNotThrow()

{

var user = new User

{

Id = Guid.NewGuid(),

FirstName = "Poushali",

LastName = "Saha",

EmailId = "poushali@example.com",

PANCardNo = "ABCDE1234F"

};

Assert.DoesNotThrow(() => \_user.CreateUser(user));

}

[Test]

public void CreateUser\_WithInvalidPAN\_ThrowsFormatException()

{

var user = new User

{

Id = Guid.NewGuid(),

FirstName = "Poushali",

LastName = "Saha",

EmailId = "poushali@example.com",

PANCardNo = "123" // Invalid PAN

};

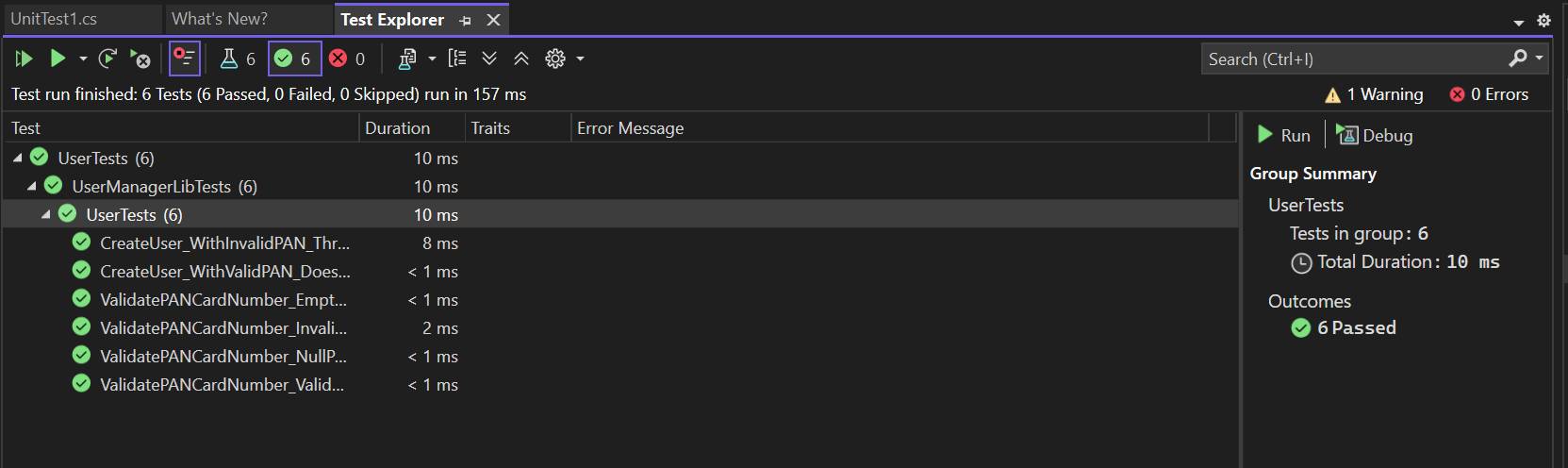
Assert.Throws<FormatException>(() => \_user.CreateUser(user));

}

}

}

**Output:**



1. **Tested Library: ConverterLib**

****Code:** ConverterTest.cs**

using NUnit.Framework;

using ConverterLib;

namespace ConverterLibTests

{

[TestFixture]

public class ConverterTest

{

private Converter \_converter;

[SetUp]

public void Init()

{

\_converter = new Converter(null); // Pass null, not testing USDToEuro now

}

[Test]

public void CelsiusToKelvin\_ReturnsCorrectValue()

{

double result = \_converter.CelsiusToKelvin(0);

Assert.AreEqual(273.15, result, 0.001);

}

[Test]

public void KilogramToPound\_ReturnsCorrectValue()

{

double result = \_converter.KilogramToPound(10);

Assert.AreEqual(22.05, result, 0.001);

}

[Test]

public void KilometerToMile\_ReturnsCorrectValue()

{

double result = \_converter.KilometerToMile(16.09);

Assert.AreEqual(10, result, 0.01);

}

[Test]

public void LiterToGallon\_ReturnsCorrectValue()

{

double result = \_converter.LiterToGallon(3.785);

Assert.AreEqual(1, result, 0.01);

}

}

}

1. **Tested Library: FourSeasonsLib**

****Code:** SeasonTellerTests.cs**

using NUnit.Framework;

using SeasonsLib;

namespace FourSeasonsLibTests

{

[TestFixture]

public class SeasonTellerTests

{

private SeasonTeller \_seasonTeller;

[SetUp]

public void Setup()

{

\_seasonTeller = new SeasonTeller();

}

[Test]

public void DisplaySeasonBy\_February\_ReturnsSpring()

{

string result = \_seasonTeller.DisplaySeasonBy("February");

Assert.AreEqual("Spring", result);

}

[Test]

public void DisplaySeasonBy\_April\_ReturnsSummer()

{

string result = \_seasonTeller.DisplaySeasonBy("April");

Assert.AreEqual("Summer", result);

}

[Test]

public void DisplaySeasonBy\_July\_ReturnsMonsoon()

{

string result = \_seasonTeller.DisplaySeasonBy("July");

Assert.AreEqual("Monsoon", result);

}

[Test]

public void DisplaySeasonBy\_October\_ReturnsAutumn()

{

string result = \_seasonTeller.DisplaySeasonBy("October");

Assert.AreEqual("Autumn", result);

}

[Test]

public void DisplaySeasonBy\_January\_ReturnsWinter()

{

string result = \_seasonTeller.DisplaySeasonBy("January");

Assert.AreEqual("Winter", result);

}

[Test]

public void DisplaySeasonBy\_InvalidMonth\_ReturnsInvalidSeason()

{

string result = \_seasonTeller.DisplaySeasonBy("HelloMonth");

Assert.AreEqual("Invalid Season", result);

}

[Test]

public void DisplaySeasonBy\_MixedCaseMonth\_ReturnsCorrectSeason()

{

string result = \_seasonTeller.DisplaySeasonBy("mArch");

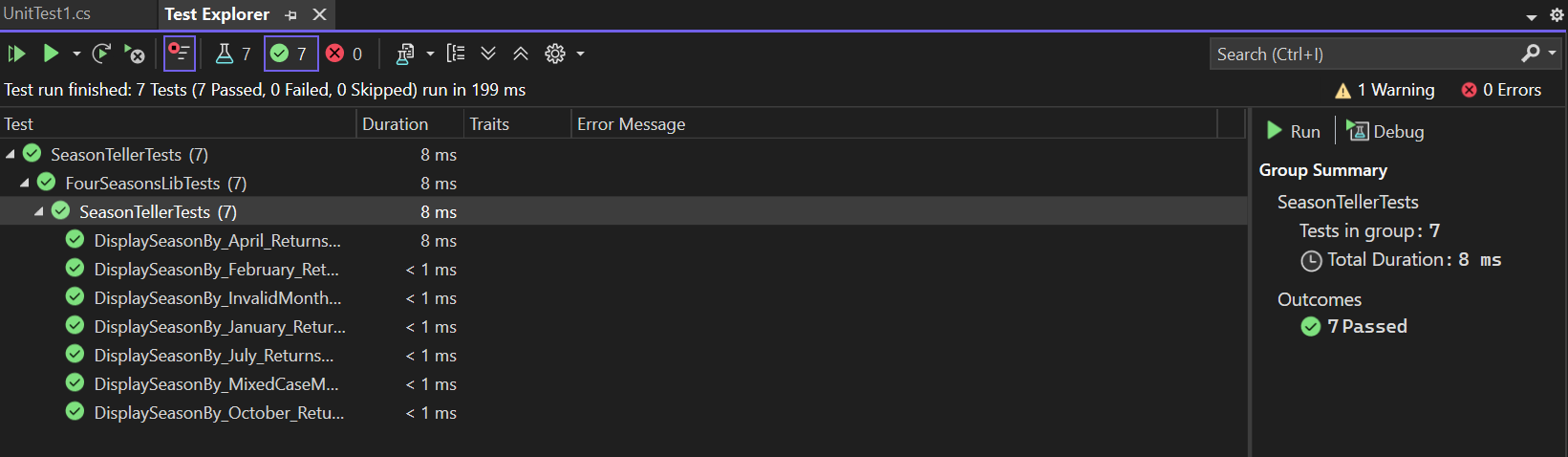
Assert.AreEqual("Spring", result);

}

}

}

**Output:**

****

1. **Tested Library: LeapYearCalculatorLib**

****Code:** LeapYearCalculatorTests.cs**

using NUnit.Framework;

using LeapYearCalculatorLib;

namespace LeapYearCalculatorLibTests

{

[TestFixture]

public class LeapYearCalculatorTests

{

private LeapYearCalculator \_calculator;

[SetUp]

public void Setup()

{

\_calculator = new LeapYearCalculator();

}

[Test]

public void IsLeapYear\_2000\_Returns1()

{

int result = \_calculator.IsLeapYear(2000);

Assert.AreEqual(1, result); // divisible by 400

}

[Test]

public void IsLeapYear\_2024\_Returns1()

{

int result = \_calculator.IsLeapYear(2024);

Assert.AreEqual(1, result); // divisible by 4 and not 100

}

[Test]

public void IsLeapYear\_1900\_Returns0()

{

int result = \_calculator.IsLeapYear(1900);

Assert.AreEqual(0, result); // divisible by 100 but not by 400

}

[Test]

public void IsLeapYear\_2023\_Returns0()

{

int result = \_calculator.IsLeapYear(2023);

Assert.AreEqual(0, result); // not divisible by 4

}

[Test]

public void IsLeapYear\_1700\_ReturnsMinus1()

{

int result = \_calculator.IsLeapYear(1700);

Assert.AreEqual(-1, result); // year < 1753

}

[Test]

public void IsLeapYear\_10000\_ReturnsMinus1()

{

int result = \_calculator.IsLeapYear(10000);

Assert.AreEqual(-1, result); // year > 9999

}

}

}

**Output:**

