**Skill:-NUnit and Moq**

**Filename:-1. NUnit-Handson**

**Name:- DOCUMENT 1\_NUnit-Handson**

**Question**

**TestFixture & Test**

Please download the application available [here](https://cognizantonline.sharepoint.com/:u:/r/sites/GTP-Solutions/Gencsharepath/Shared%20Documents/Internship2020/FSE/DotNet/02%20-%20NUnit,%20C%23%204.5,%20ASP.Net%20Core/Handson/CalcLibrary.zip?csf=1&e=aLxB66). 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**

**Code For CalculatorTests.cs in CalcLibrary.cs**

using NUnit.Framework;

using CalcLibrary;

using System;

namespace CalcLibrary.Tests

{

[TestFixture] // Marks class for testing

public class CalculatorTests

{

private IMathLibrary calculator;

[SetUp] // Runs before each test

public void Setup()

{

calculator = new SimpleCalculator(); // Loosely coupled - use interface

}

[TearDown] // Runs after each test

public void Teardown()

{

((SimpleCalculator)calculator).AllClear();

}

[Test] // Single test

public void Addition\_ShouldReturnCorrectResult()

{

var result = calculator.Addition(10, 5);

Assert.That(result, Is.EqualTo(15));

}

[Test] // Test with multiple inputs

[TestCase(2, 3, 5)]

[TestCase(-1, 1, 0)]

[TestCase(0, 0, 0)]

[TestCase(-5, -3, -8)]

public void Addition\_WithMultipleInputs\_ShouldReturnCorrectSum(double a, double b, double expected)

{

var result = calculator.Addition(a, b);

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

}

[Test]

[TestCase(10, 5, 5)]

public void Subtraction\_ShouldReturnCorrectResult(double a, double b, double expected)

{

var result = calculator.Subtraction(a, b);

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

}

[Test]

[TestCase(4, 5, 20)]

public void Multiplication\_ShouldReturnCorrectResult(double a, double b, double expected)

{

var result = calculator.Multiplication(a, b);

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

}

[Test]

[TestCase(10, 2, 5)]

public void Division\_ShouldReturnCorrectResult(double a, double b, double expected)

{

var result = calculator.Division(a, b);

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

}

[Test]

public void Division\_ByZero\_ShouldThrowException()

{

Assert.Throws<ArgumentException>(() => calculator.Division(10, 0));

}

[Test]

[Ignore("This test is not implemented yet")]

public void Percentage\_ShouldReturnCorrectResult()

{

// Placeholder test

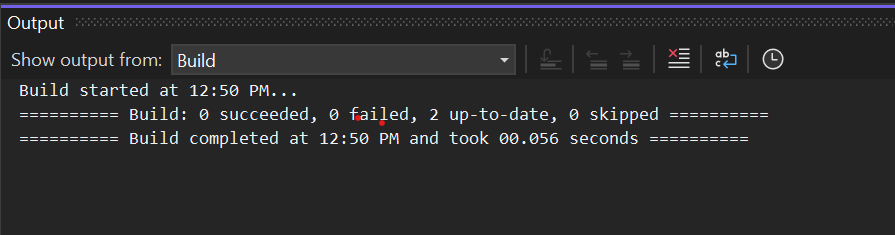
}

}

}

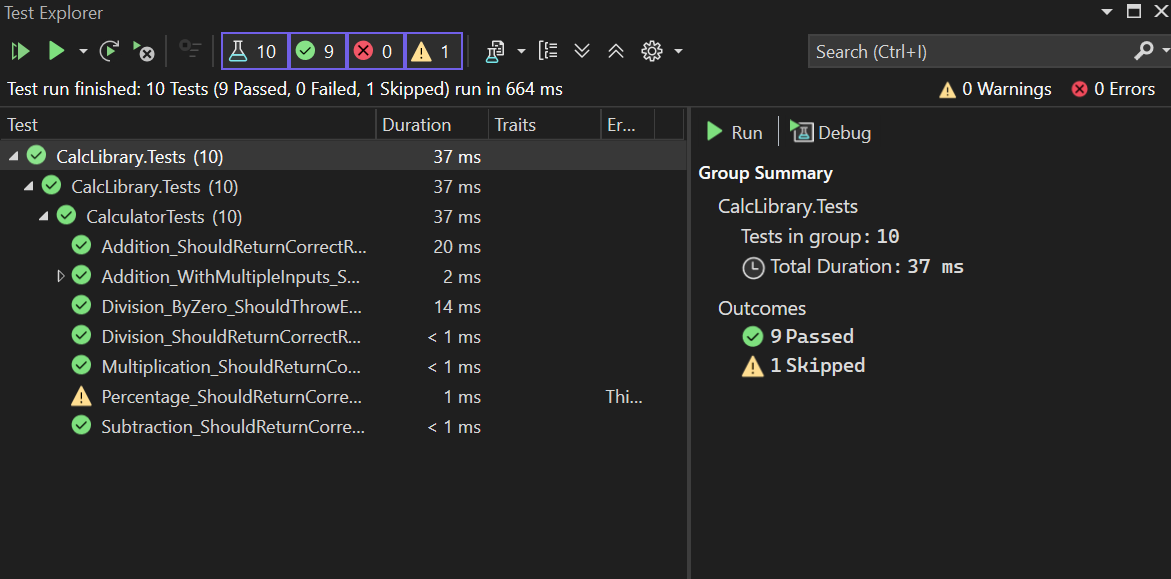
**Finally Building of Project**

**OUTPUT:-**



**Passing Test Case**

**OUTPUT:-**



**Name:- DOCUMENT 2\_NUnit-Handson**

**Question**

**Parameterized test cases**

1. Create test case to verify the subtraction feature of the calculator with various input types.

* Create test cases with ‘TestCase’ attribute to send in input parameters and the expected result.
* Add more than 1 ‘TestCase’ attributes to check various combinations for subtractions.
* Use Assert.Equal to check the actual and expected results

1. Create a test case to verify the multiplication concepts of calculator

* Create test cases with ‘TestCase’ attribute to send in input parameters and the expected result.
* Add more than 1 ‘TestCase’ attributes to check various combinations for subtractions.
* Use Assert.Equal to check the actual and expected results

1. Create a test case to verify the division logic of the calculator

* Create test cases with ‘TestCase’ attribute to send in input parameters and the expected result.
* Add more than 1 ‘TestCase’ attributes to check various combinations for subtractions.
* Use Assert.Equal to check the actual and expected results
* In one of the inputs, provide the divisor value to be 0
  + Use Try Catch block to catch the ArgumentException
  + Use Assert.Fail to notify the user that the test case has failed. Give the message “Division by zero” in the Assert.Fail, which will be notified to the user. This message will be seen in the test explorer.

**Test void methods**

In the MathLibrary class there is a property “GetResult”. The result of every operation is stored in a variable ‘result’. This value is accessed by the property.

The class also has a method “AllClear” that sets the value of the result variable to 0.

* Create a test method ‘TestAddAndClear’
* Invoke the Addition method of the math class library
* Verify if the expected and Actual results match using Assert.AreEqual
* Invoke the ‘AllClear’ method
* Use Assert.AreEqual to check if the result is 0 or not

**Solution**

**Code For CalculatorTests.cs in CalcLibrary.cs**

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 TearDown()

{

calculator.AllClear();

}

// 1. PARAMETERIZED TEST CASES - Subtraction

[Test]

[TestCase(10, 5, 5)]

[TestCase(5, 10, -5)]

[TestCase(0, 0, 0)]

[TestCase(-5, -3, -2)]

public void Subtraction\_TestCases(double a, double b, double expected)

{

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

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

}

// 2. PARAMETERIZED TEST CASES - Multiplication

[Test]

[TestCase(2, 3, 6)]

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

[TestCase(0, 100, 0)]

[TestCase(7.5, 2, 15)]

public void Multiplication\_TestCases(double a, double b, double expected)

{

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

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

}

// 3. PARAMETERIZED TEST CASES - Division

[Test]

[TestCase(10, 2, 5)]

[TestCase(9, 3, 3)]

[TestCase(100, 5, 20)]

public void Division\_TestCases(double a, double b, double expected)

{

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

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

}

// 4. DIVISION BY ZERO - EXCEPTION TESTING

[Test]

public void Division\_ByZero\_ShouldThrowArgumentException()

{

try

{

double result = calculator.Division(10, 0);

Assert.Fail("Division by zero"); // This line will fail the test if no exception is thrown

}

catch (ArgumentException ex)

{

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

}

}

// 5. VOID METHOD TESTING

[Test]

public void TestAddAndClear()

{

calculator.Addition(10, 5);

Assert.That(calculator.GetResult, Is.EqualTo(15));

calculator.AllClear();

Assert.That(calculator.GetResult, Is.EqualTo(0));

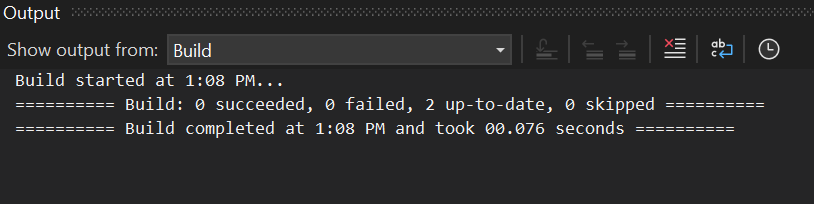
}

}

}

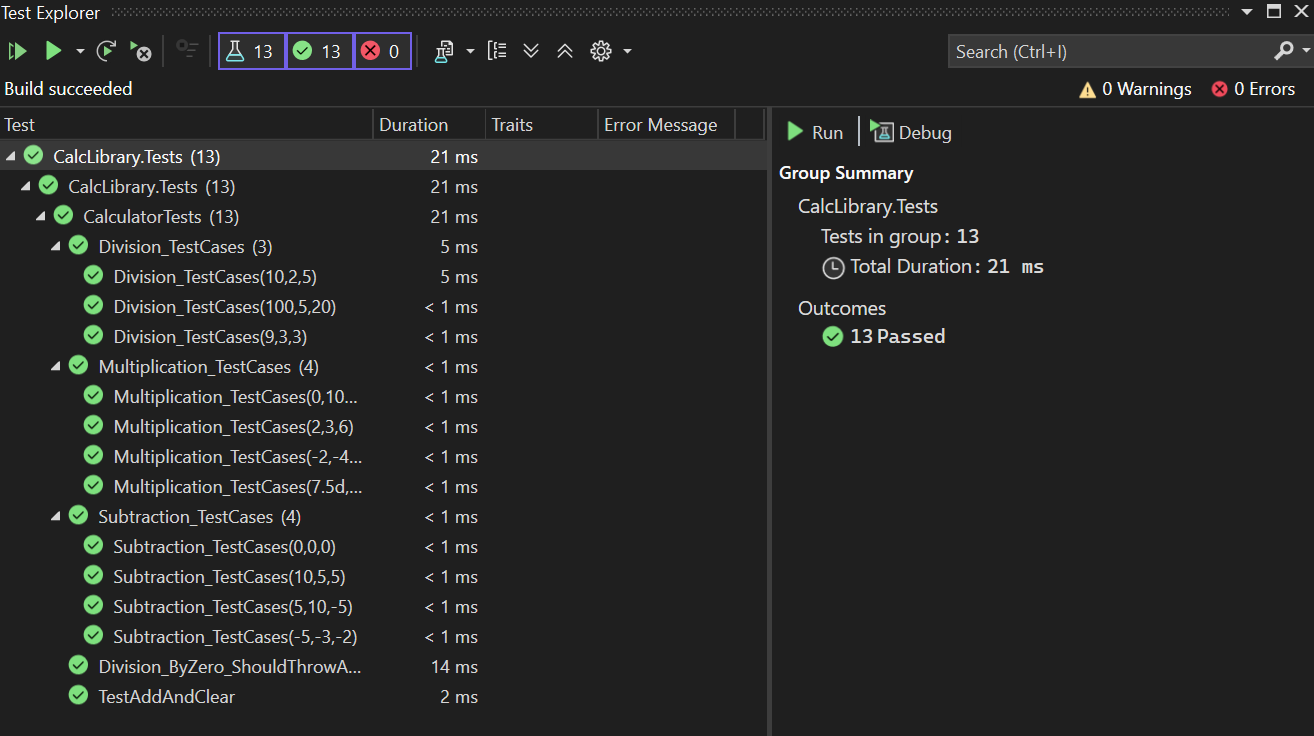
**Finally Building of Project**

**OUTPUT:-**



**Passing Test Case**

**OUTPUT:-**



**Name:- DOCUMENT 3\_NUnit-Handson**

**Question**

# Objectives

* This lab will help you become skilled at writing automated unit tests using the NUnit framework.
* Explain & demonstrate various NUnit custom attributes to identify tests.

Create a unit test project using NUnit for the given **UtilLib** project. Click [here](https://cognizantonline.sharepoint.com/:u:/r/sites/GTP-Solutions/Gencsharepath/Shared%20Documents/Internship2020/FSE/DotNet/02%20-%20NUnit,%20C%23%204.5,%20ASP.Net%20Core/Handson/UtilLib.zip?csf=1&web=1&e=RApwRV) to download the source project.

The functionality called **ParseHostName** which is defined in the **UrlHostNameParser** class. It parses the host name from the URL using certain logic. Write all possible test methods for the given functionality to make sure that it returns the expected result under various circumstances.

Since the function is having two execution paths, you need to write at least two test methods.

**Recommendations:**

Test Project Name:*<ClassLib\_Project>.Tests*

Test Class Name: *<SUT>Tests*

Test Method Name:  *UnitUnderTest\_Scenario\_ExpectedOutcome*

**Note:**

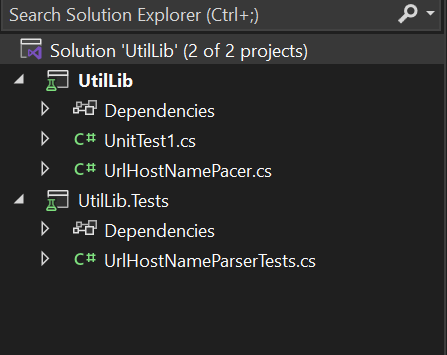
* *Enforce the Single Assertion Rule*
* *Use Assert.That()*

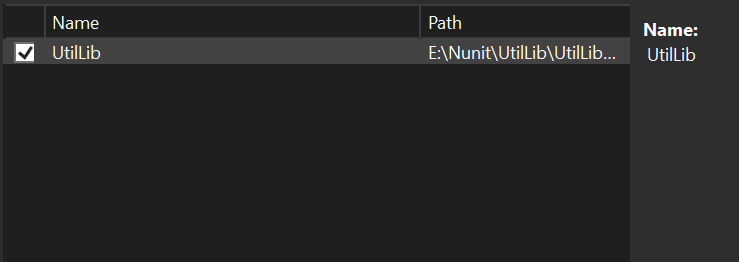
**Steps to perform**

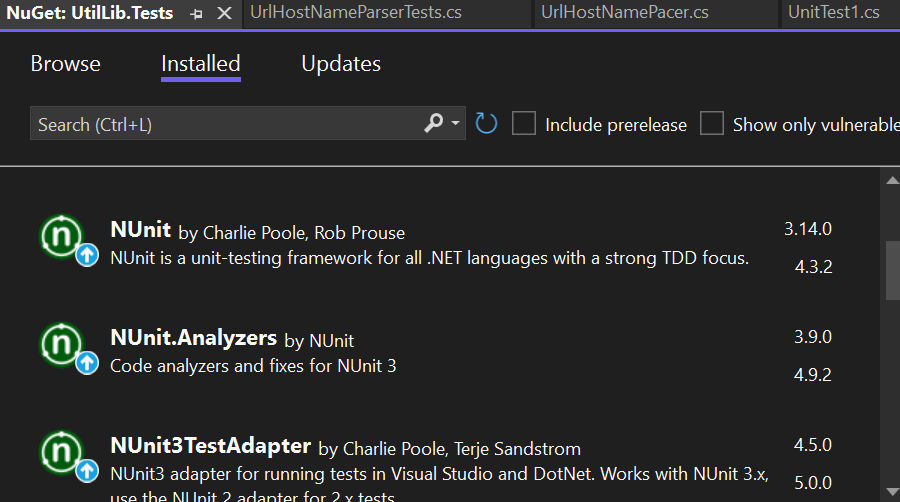
1. Create a Class Library project in the same solution which is provided and name it as suggested.
2. Rename the class file name (<SUT>Tests.cs).
3. Add the assembly reference of the UtilLib project to the test project.
4. Additionally add the reference of both NUnit and NUnit3TestAdapter in the test project using NuGet Package Manager (NPM).
5. Write the suggested test methods.
6. Run your tests.
7. Break the test by modifying the source project functionality.
8. Rerun the test.
9. Observe the test result.

**Solution**

**Upto 6 number we have done rename the project and add reference and install packages Screenshot are below**







**Code for UrlHostNamePacer.cs in UtilLib**

namespace UtilLib

{

public class UrlHostNameParser

{

public string ParseHostName(string url)

{

try

{

var uri = new System.Uri(url);

return uri.Host;

}

catch

{

return string.Empty;

}

}

}

}

**Code for UrlHostNameParserTests.cs in UtilLib.Tests**

using NUnit.Framework;

using UtilLib;

namespace UtilLib.Tests

{

[TestFixture]

public class UrlHostNameParserTests

{

private UrlHostNameParser parser;

[SetUp]

public void SetUp()

{

parser = new UrlHostNameParser();

}

[Test]

public void ParseHostName\_ValidUrlWithHttps\_ReturnsHostName()

{

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

string expected = "www.example.com";

string result = parser.ParseHostName(url);

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

}

[Test]

public void ParseHostName\_ValidUrlWithHttp\_ReturnsHostName()

{

string url = "http://subdomain.testsite.org/home";

string expected = "subdomain.testsite.org";

string result = parser.ParseHostName(url);

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

}

[Test]

public void ParseHostName\_InvalidUrl\_ReturnsEmptyString()

{

string url = "not a valid url";

string expected = string.Empty;

string result = parser.ParseHostName(url);

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

}

[Test]

public void ParseHostName\_EmptyString\_ReturnsEmptyString()

{

string url = "";

string expected = string.Empty;

string result = parser.ParseHostName(url);

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

}

[Test]

public void ParseHostName\_NullInput\_ReturnsEmptyString()

{

string url = null;

string expected = string.Empty;

string result = parser.ParseHostName(url);

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

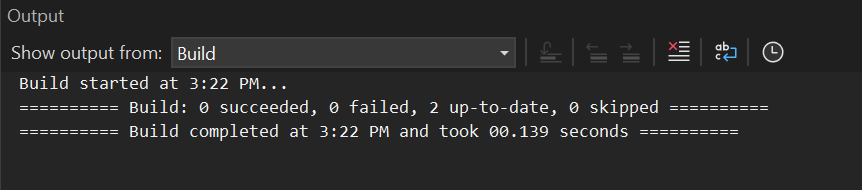
}

}

}

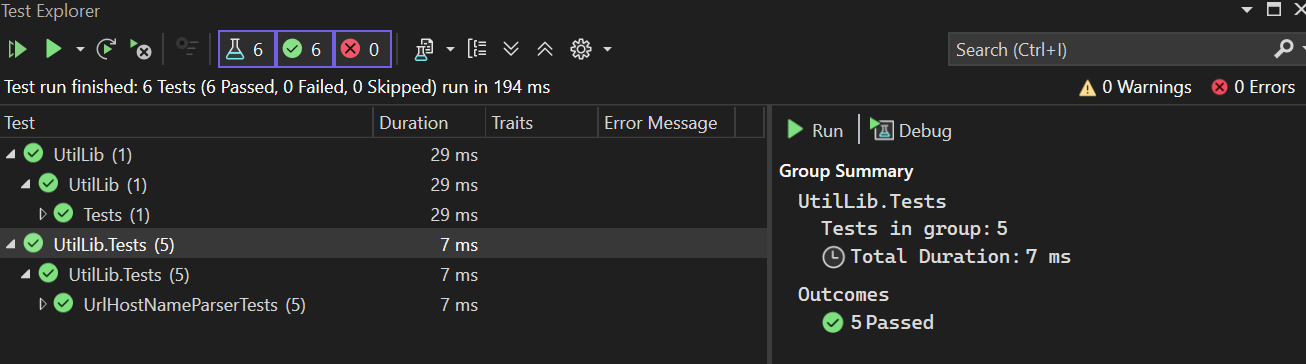
**Building of Project Upto Step 6**

**OUTPUT:-**



**Passing Test Case**

**OUTPUT:-**



**After 7 we are** modifying the source project functionality and repating same thing that is build up and tests…

**Code for UrlHostNamePacer.cs in UtilLib**

namespace UtilLib

{

public class UrlHostNameParser

{

public string ParseHostName(string url)

{

return "broken.host.com";

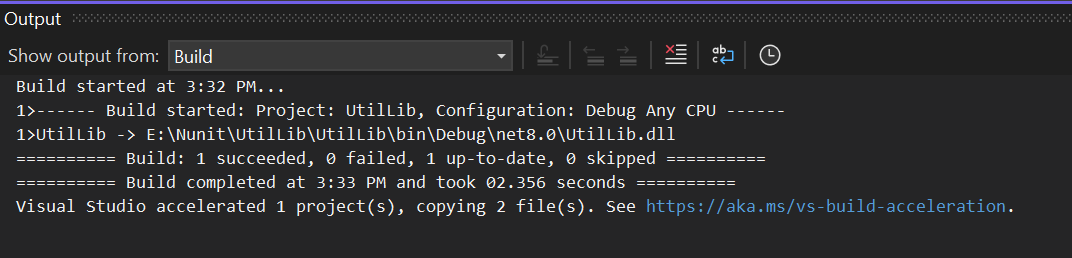
}

}

}

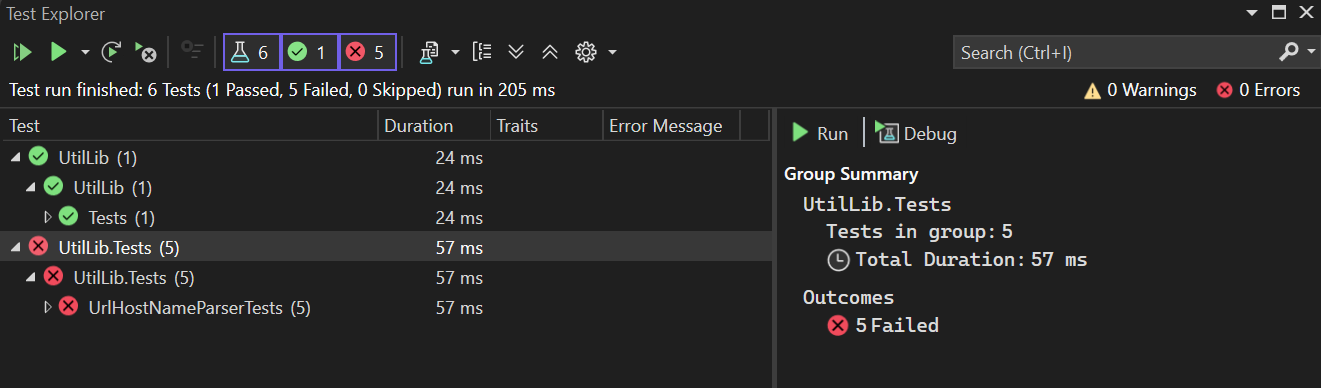
**Building of Project After Step 6**

**OUTPUT:-**



**Passing Test Case**

**OUTPUT:-**



**Name:- DOCUMENT 4\_NUnit-Handson**

**Solution:-**

**Code for AccountsManagerLib**

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 Login\_ValidCredentialsUser11\_ReturnsWelcomeMessage()

{

string result = manager.Login("user\_11", "secret@user11");

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

}

[Test]

public void Login\_ValidCredentialsUser22\_ReturnsWelcomeMessage()

{

string result = manager.Login("user\_22", "secret@user22");

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

}

[Test]

public void Login\_InvalidCredentials\_ReturnsInvalidMessage()

{

string result = manager.Login("user\_11", "wrongpass");

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

}

[Test]

public void Login\_MissingUserId\_ThrowsArgumentException()

{

var ex = Assert.Throws<ArgumentException>(() => manager.Login("", "password"));

Assert.That(ex.Message, Is.EqualTo("User ID and password are required"));

}

[Test]

public void Login\_MissingPassword\_ThrowsArgumentException()

{

var ex = Assert.Throws<ArgumentException>(() => manager.Login("user\_11", ""));

Assert.That(ex.Message, Is.EqualTo("User ID and password are required"));

}

}

}

**Code for AccountsManagerTests.cs in 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 Login\_ValidCredentialsUser11\_ReturnsWelcomeMessage()

{

string result = manager.Login("user\_11", "secret@user11");

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

}

[Test]

public void Login\_ValidCredentialsUser22\_ReturnsWelcomeMessage()

{

string result = manager.Login("user\_22", "secret@user22");

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

}

[Test]

public void Login\_InvalidCredentials\_ReturnsInvalidMessage()

{

string result = manager.Login("user\_11", "wrongpass");

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

}

[Test]

public void Login\_MissingUserId\_ThrowsArgumentException()

{

var ex = Assert.Throws<ArgumentException>(() => manager.Login("", "password"));

Assert.That(ex.Message, Is.EqualTo("User ID and password are required"));

}

[Test]

public void Login\_MissingPassword\_ThrowsArgumentException()

{

var ex = Assert.Throws<ArgumentException>(() => manager.Login("user\_11", ""));

Assert.That(ex.Message, Is.EqualTo("User ID and password are required"));

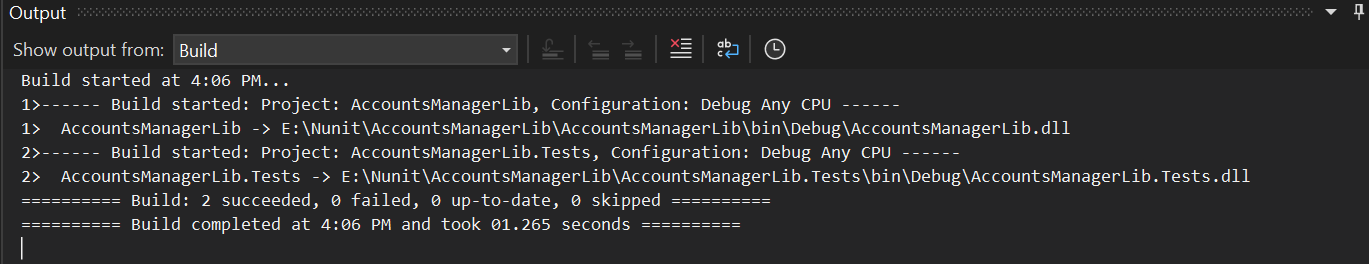
}

}

}

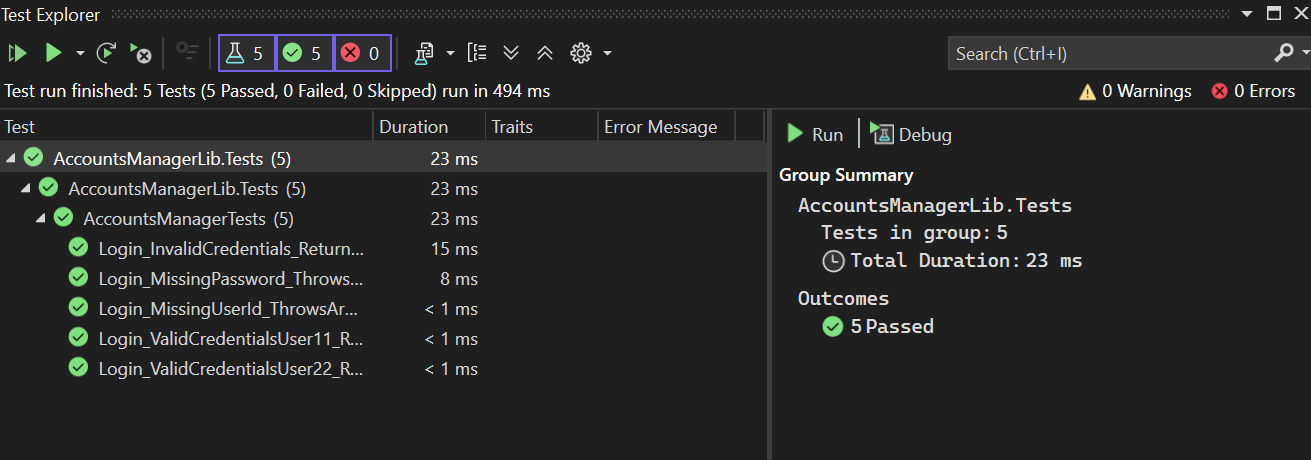
**Building of Project Before Step 6**

**OUTPUT:-**



**Passing Test Case**

**OUTPUT:-**



***Break a Function & Re-Test***

**Code for AccountsManagerLib**

using System;

namespace AccountsManagerLib

{

public class AccountsManager

{

public string Login(string userId, string password)

{

if (string.IsNullOrEmpty(userId) || string.IsNullOrEmpty(password))

throw new ArgumentException("User ID and password are required");

// 🔴 Introduced Bug: Changed Welcome to Hello (invalid return string)

if ((userId == "user\_11" && password == "secret@user11") ||

(userId == "user\_22" && password == "secret@user22"))

{

return $"Hello {userId}!!!"; // ❌ This will break the test

}

return "Invalid user id/password";

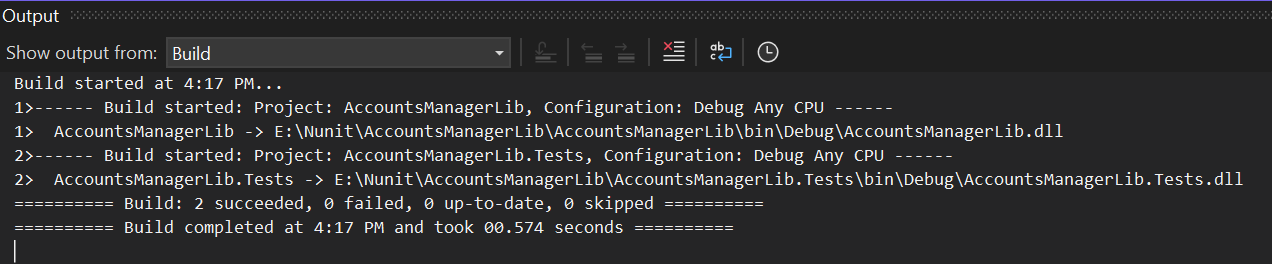
}

}

}

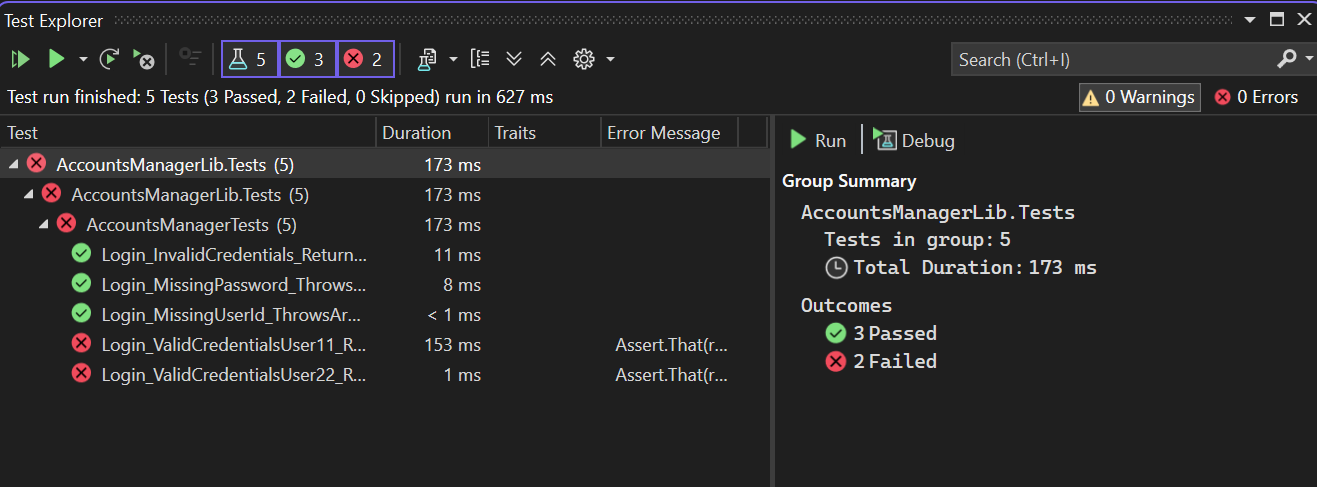
**Building of Project After Step 6**

**OUTPUT:-**



**Passing Test Case**

**OUTPUT:-**



**Name:- DOCUMENT 5\_NUnit-Handson**

**Solution:-**

**Code for Employee.cs in CollectionsLib**

namespace CollectionsLib

{

public class Employee

{

public int Id { get; set; }

public string Name { get; set; }

public override bool Equals(object obj)

{

if (obj is Employee other)

{

return Id == other.Id;

}

return false;

}

public override int GetHashCode()

{

return Id.GetHashCode();

}

}

}

**Code for EmployeeManager.cs**

using System.Collections.Generic;

namespace CollectionsLib

{

public class EmployeeManager

{

public List<Employee> GetEmployees()

{

return new List<Employee>

{

new Employee { Id = 100, Name = "Alice" },

new Employee { Id = 101, Name = "Bob" },

new Employee { Id = 102, Name = "Charlie" }

};

}

public List<Employee> GetEmployeesWhoJoinedInPreviousYears()

{

return new List<Employee>

{

new Employee { Id = 100, Name = "Alice" },

new Employee { Id = 101, Name = "Bob" },

new Employee { Id = 102, Name = "Charlie" }

};

}

}

}

**Code for CollectionsLib.Tests in EmployeeManagerTests.cs**

using CollectionsLib;

using NUnit.Framework;

using NUnit.Framework.Legacy;

using System.Collections.Generic;

using System.Linq;

namespace CollectionsLib.Tests

{

[TestFixture]

public class EmployeeManagerTests

{

private EmployeeManager manager;

[SetUp]

public void Setup()

{

manager = new EmployeeManager();

}

[Test]

public void GetEmployees\_NoNullsInCollection\_ShouldPass()

{

var employees = manager.GetEmployees();

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

}

[Test]

public void GetEmployees\_ContainsEmployeeWithId100\_ShouldPass()

{

var employees = manager.GetEmployees();

var contains = employees.Any(e => e.Id == 100);

Assert.That(contains, Is.True); // Constraint Model

}

[Test]

public void GetEmployees\_UniqueEmployeesById\_ShouldPass()

{

var employees = manager.GetEmployees();

var distinctCount = employees.Select(e => e.Id).Distinct().Count();

Assert.That(distinctCount, Is.EqualTo(employees.Count)); // Constraint Model

}

[Test]

public void GetEmployeesAndPreviousEmployees\_AreEqual\_CollectionsShouldPass()

{

var current = manager.GetEmployees();

var previous = manager.GetEmployeesWhoJoinedInPreviousYears();

// Classic Model

CollectionAssert.AreEqual(current, previous);

// Constraint Model

Assert.That(current, Is.EqualTo(previous));

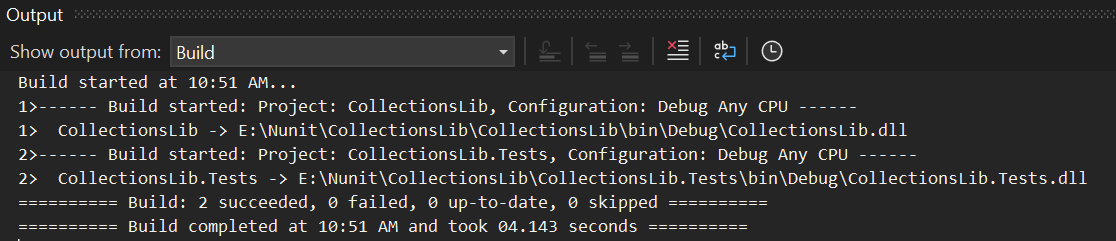
}

}

}

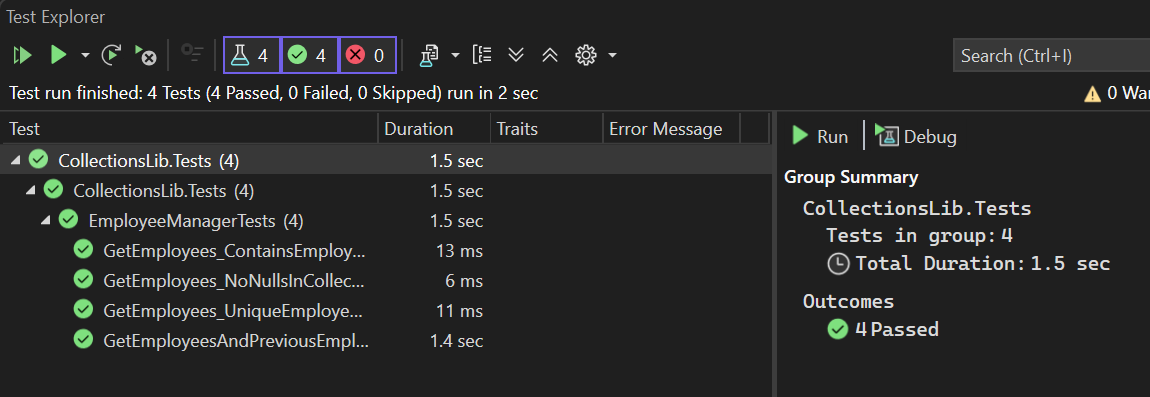
**Building of Project Upto Step 6**

**OUTPUT:-**



**Passing Test Case**

**OUTPUT:-**



***Break a Function & Re-Test***

**Code for CollectionsLib.Tests in EmployeeManager.cs**

using System.Collections.Generic;

namespace CollectionsLib

{

public class EmployeeManager

{

public List<Employee> GetEmployees()

{

// ❌ Duplicate Id (101) – This will break the uniqueness test

return new List<Employee>

{

new Employee { Id = 100, Name = "Alice" },

new Employee { Id = 101, Name = "Bob" },

new Employee { Id = 101, Name = "Eve" } // Duplicate ID to break test

};

}

public List<Employee> GetEmployeesWhoJoinedInPreviousYears()

{

return new List<Employee>

{

new Employee { Id = 100, Name = "Alice" },

new Employee { Id = 101, Name = "Bob" },

new Employee { Id = 102, Name = "Charlie" }

};

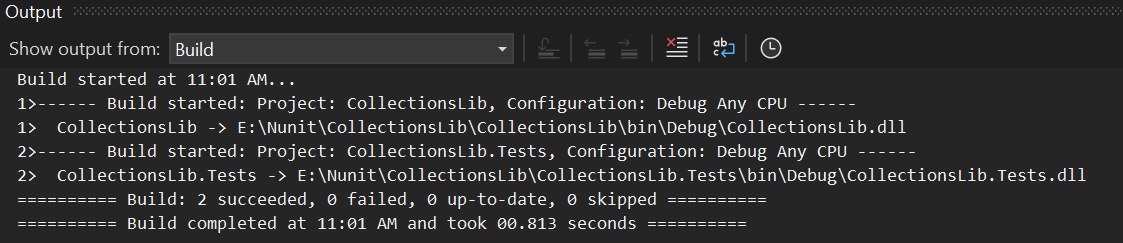
}

}

}

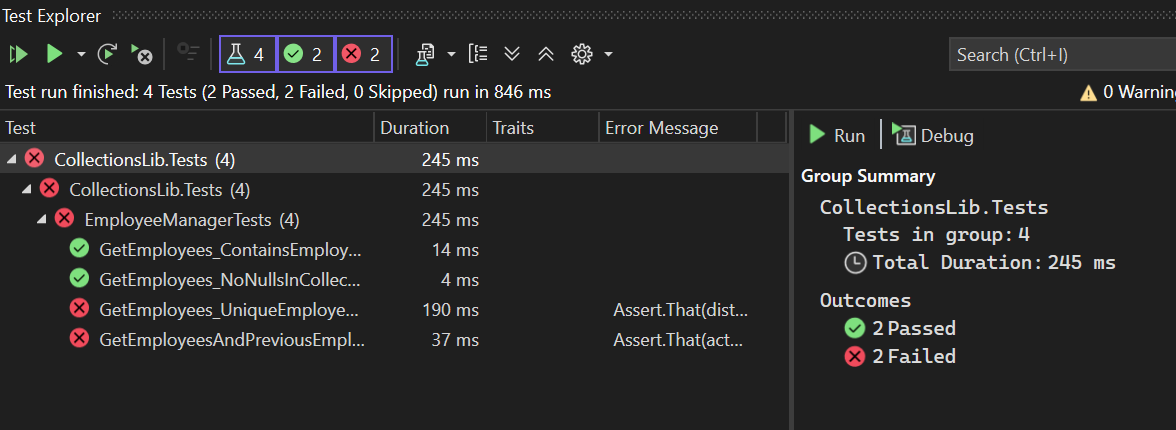
**Building of Project After Step 6**

**OUTPUT:-**



**Passing Test Case**

**OUTPUT:-**



**Name:- DOCUMENT 6\_NUnit-Handson**

**Solution:-**

**Code for SeasonFinder.cs in FourSeasonsLib**

using System;

namespace FourSeasonsLib

{

public class SeasonFinder

{

public string GetSeason(string month)

{

switch (month.ToLower())

{

case "february":

case "march":

return "Spring";

case "april":

case "may":

case "june":

return "Summer";

case "july":

case "august":

return "Monsoon";

case "september":

case "october":

case "november":

return "Autumn";

case "december":

case "january":

return "Winter";

default:

throw new ArgumentException("Invalid month");

}

}

}

}

**Code for SeasonFinderTests.csin FourSeasonsLib.Tests**

using NUnit.Framework;

using FourSeasonsLib;

using System.Collections;

namespace FourSeasonsLib.Tests

{

[TestFixture]

public class SeasonFinderTests

{

private SeasonFinder seasonFinder;

[SetUp]

public void Setup()

{

seasonFinder = new SeasonFinder();

}

// External source for TestCaseSource

public static IEnumerable MonthToSeasonCases

{

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("Autumn");

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");

}

}

[Test, TestCaseSource(nameof(MonthToSeasonCases))]

public string GetSeason\_MonthName\_ReturnsExpectedSeason(string month)

{

return seasonFinder.GetSeason(month);

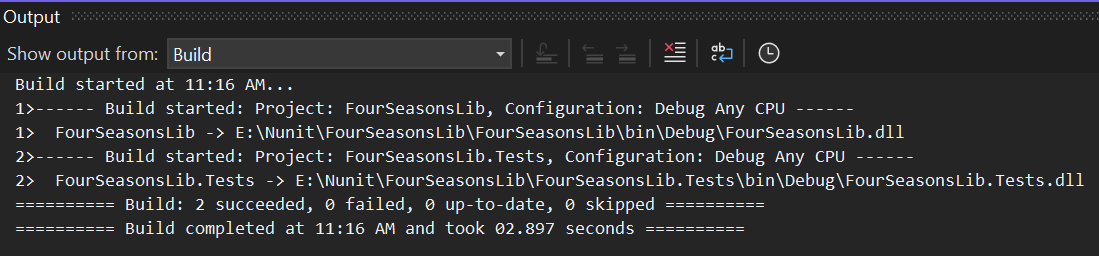
}

}

}

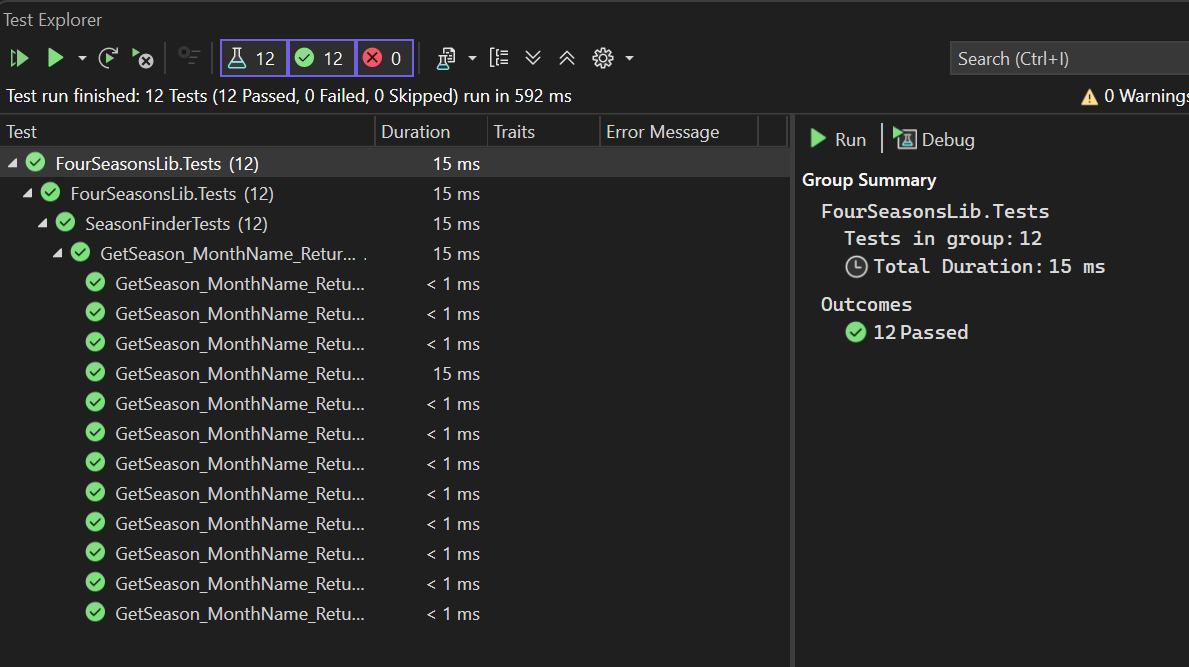
**Building of Project Upto Step 6**

**OUTPUT:-**



**Passing Test Case**

**OUTPUT:-**



***Break a Function & Re-Test***

**Code for SeasonFinder.cs in FourSeasonsLib**

using System;

namespace FourSeasonsLib

{

public class SeasonFinder

{

public string GetSeason(string month)

{

switch (month.ToLower())

{

case "february":

case "march":

return "Summer"; // ❌ Wrong logic

case "april":

case "may":

case "june":

return "Summer";

case "july":

case "august":

return "Monsoon";

case "september":

case "october":

case "november":

return "Autumn";

case "december":

case "january":

return "Winter";

default:

throw new ArgumentException("Invalid month");

}

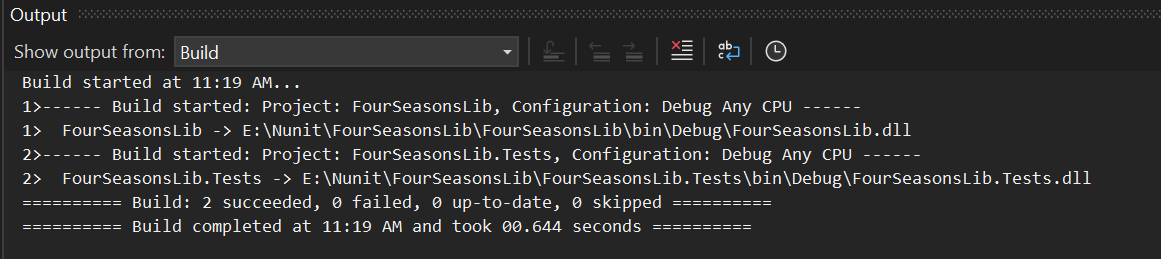
}

}

}

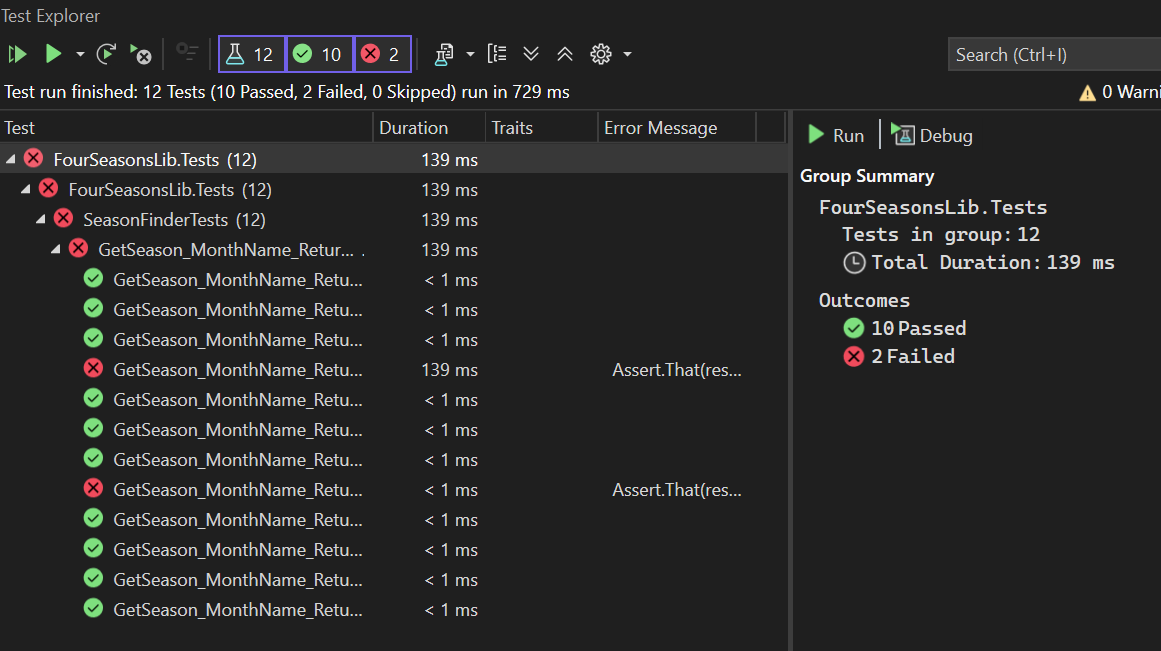
**Building of Project After Step 6**

**OUTPUT:-**



**Passing Test Case**

**OUTPUT:-**



**Name:- DOCUMENT 7\_NUnit-Handson**

**Solution:-**

**Code for LeapYearChecker.cs in LeapYearCalculatorLib**

namespace LeapYearCalculatorLib

{

public class LeapYearChecker

{

public int IsLeapYear(int year)

{

if (year < 1753 || year > 9999)

return -1;

if ((year % 4 == 0 && year % 100 != 0) || (year % 400 == 0))

return 1;

return 0;

}

}

}

**Code for LeapYearCheckerTests.cs in LeapYearCalculatorLib.Tests**

using NUnit.Framework;

using LeapYearCalculatorLib;

namespace LeapYearCalculatorLib.Tests

{

[TestFixture]

public class LeapYearCheckerTests

{

private LeapYearChecker checker;

[SetUp]

public void Setup()

{

checker = new LeapYearChecker();

}

// Valid Leap Years

[TestCase(2000, ExpectedResult = 1)]

[TestCase(2024, ExpectedResult = 1)]

[TestCase(2400, ExpectedResult = 1)]

public int IsLeapYear\_LeapYears\_ReturnsOne(int year)

{

return checker.IsLeapYear(year);

}

// Valid Non-Leap Years

[TestCase(1700, ExpectedResult = -1)] // Invalid

[TestCase(1900, ExpectedResult = 0)]

[TestCase(2023, ExpectedResult = 0)]

public int IsLeapYear\_NonLeapYears\_ReturnsZeroOrMinusOne(int year)

{

return checker.IsLeapYear(year);

}

// Invalid Years

[TestCase(1000, ExpectedResult = -1)]

[TestCase(1500, ExpectedResult = -1)]

[TestCase(10000, ExpectedResult = -1)]

public int IsLeapYear\_InvalidYears\_ReturnsMinusOne(int year)

{

return checker.IsLeapYear(year);

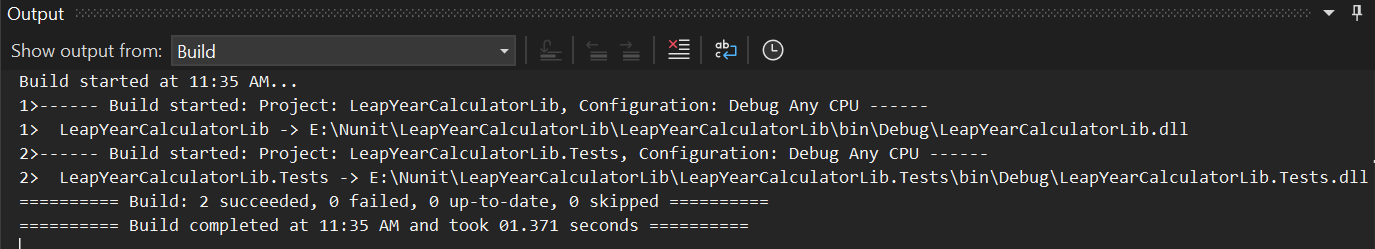
}

}

}

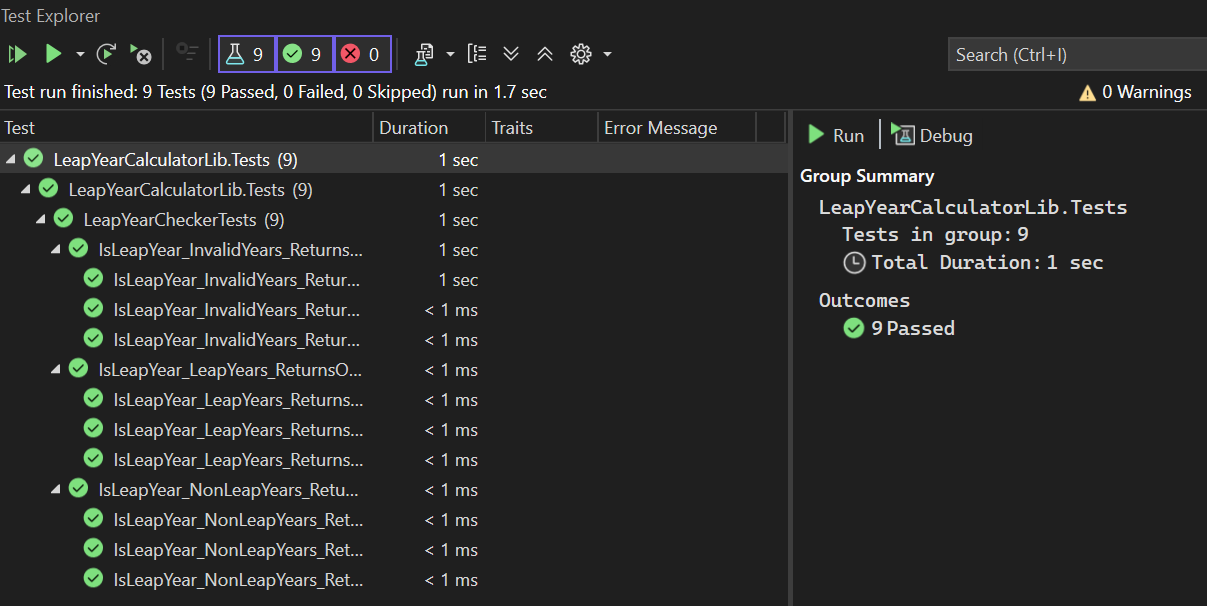
**Building of Project Upto Step 6**

**OUTPUT:-**



**Passing Test Case**

**OUTPUT:-**



***Break a Function & Re-Test***

**Code for LeapYearChecker.cs in LeapYearCalculatorLib**

namespace LeapYearCalculatorLib

{

public class LeapYearChecker

{

public int IsLeapYear(int year)

{

if (year < 1753 || year > 9999)

return -1;

// ❌ BROKEN: Removed leap year logic

return 0;

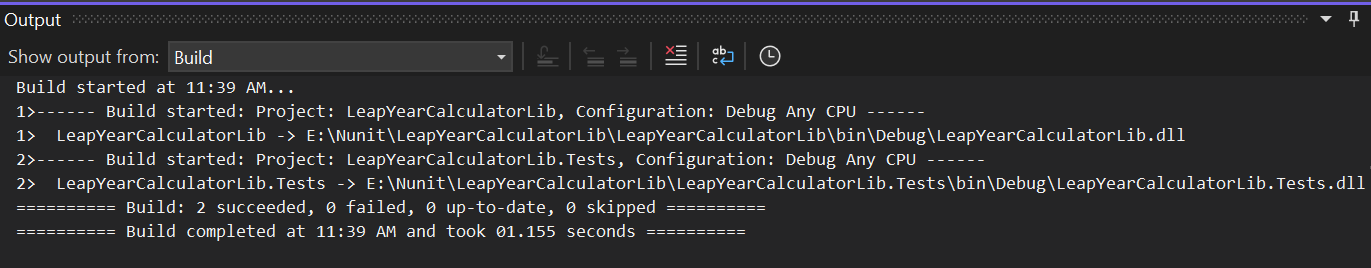
}

}

}

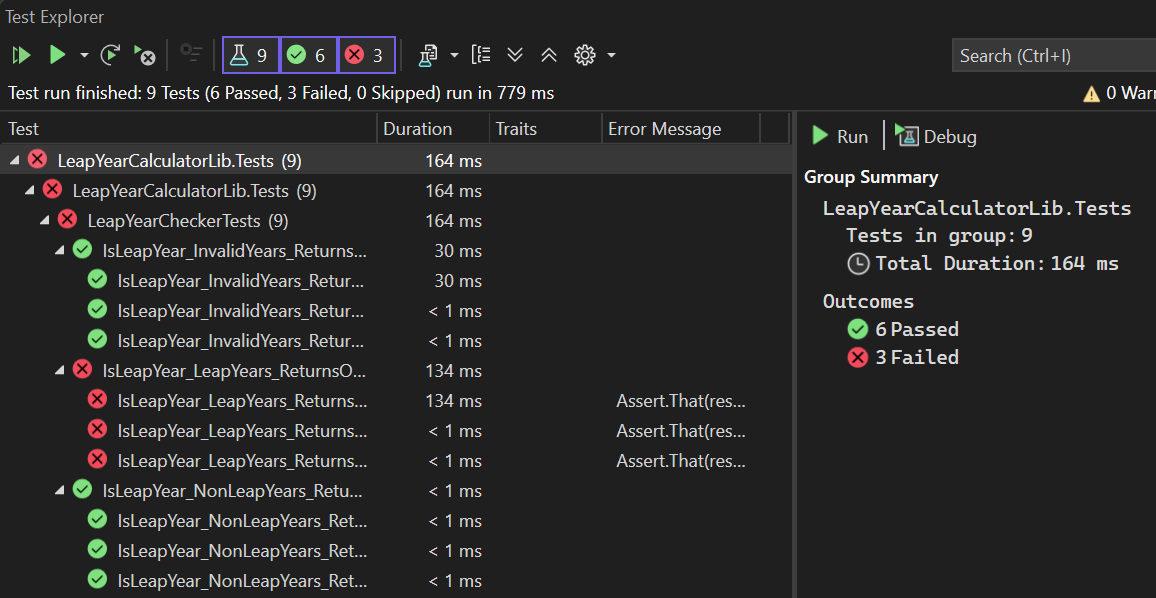
**Building of Project After Step 6**

**OUTPUT:-**



**Passing Test Case**

**OUTPUT:-**



**Name:- DOCUMENT 8\_NUnit-Handson**

**Solution:-**

**Code for UserManager.cs in UserManagerLib**

using System;

namespace UserManagerLib

{

public class UserManager

{

public string PANCardNo { get; set; }

public string CreateUser(string panCardNo)

{

if (string.IsNullOrEmpty(panCardNo))

throw new NullReferenceException("PAN card number is required");

if (panCardNo.Length != 10)

throw new FormatException("PAN card number must be exactly 10 characters");

this.PANCardNo = panCardNo;

return "User created successfully";

}

}

}

**Code for UserManagerTests.cs in UserManagerLib.Tests**

using NUnit.Framework;

using UserManagerLib;

using System;

namespace UserManagerLib.Tests

{

[TestFixture]

public class UserManagerTests

{

private UserManager manager;

[SetUp]

public void Setup()

{

manager = new UserManager();

}

[Test]

public void CreateUser\_ValidPANCard\_ReturnsSuccessMessage()

{

string result = manager.CreateUser("ABCDE1234Z");

Assert.That(result, Is.EqualTo("User created successfully"));

}

[Test]

public void CreateUser\_EmptyPANCard\_ThrowsNullReferenceException()

{

var ex = Assert.Throws<NullReferenceException>(() => manager.CreateUser(""));

Assert.That(ex.Message, Is.EqualTo("PAN card number is required"));

}

[Test]

public void CreateUser\_NullPANCard\_ThrowsNullReferenceException()

{

var ex = Assert.Throws<NullReferenceException>(() => manager.CreateUser(null));

Assert.That(ex.Message, Is.EqualTo("PAN card number is required"));

}

[Test]

public void CreateUser\_InvalidLengthPANCard\_ThrowsFormatException()

{

var ex = Assert.Throws<FormatException>(() => manager.CreateUser("AB123"));

Assert.That(ex.Message, Is.EqualTo("PAN card number must be exactly 10 characters"));

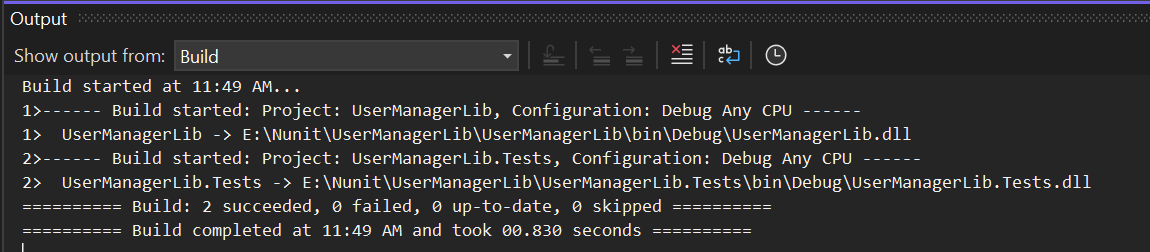
}

}

}

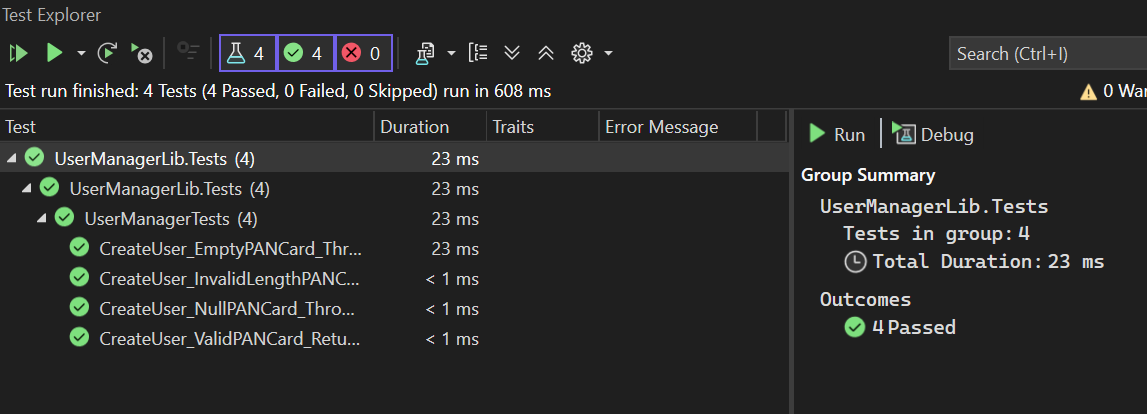
**Building of Project Upto Step 6**

**OUTPUT:-**



**Passing Test Case**

**OUTPUT:-**



***Break a Function & Re-Test***

**Code for UserManager.cs in UserManagerLib**

using System;

namespace UserManagerLib

{

public class UserManager

{

public string PANCardNo { get; set; }

public string CreateUser(string panCardNo)

{

if (string.IsNullOrEmpty(panCardNo))

throw new FormatException("Wrong input"); // ❌ wrong exception type

if (panCardNo.Length != 10)

throw new FormatException("PAN card number must be exactly 10 characters");

this.PANCardNo = panCardNo;

return "User created successfully";

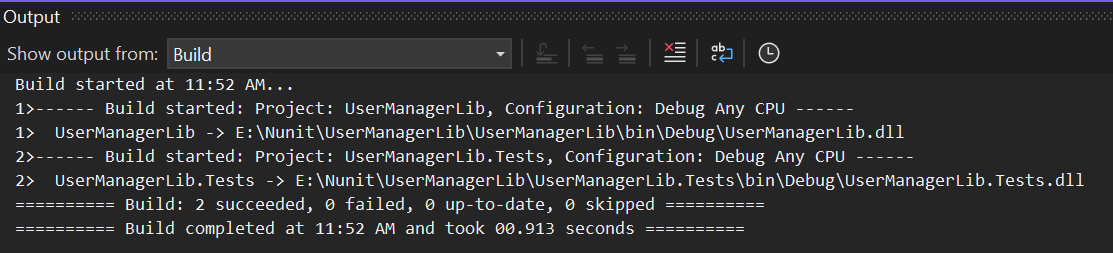
}

}

}

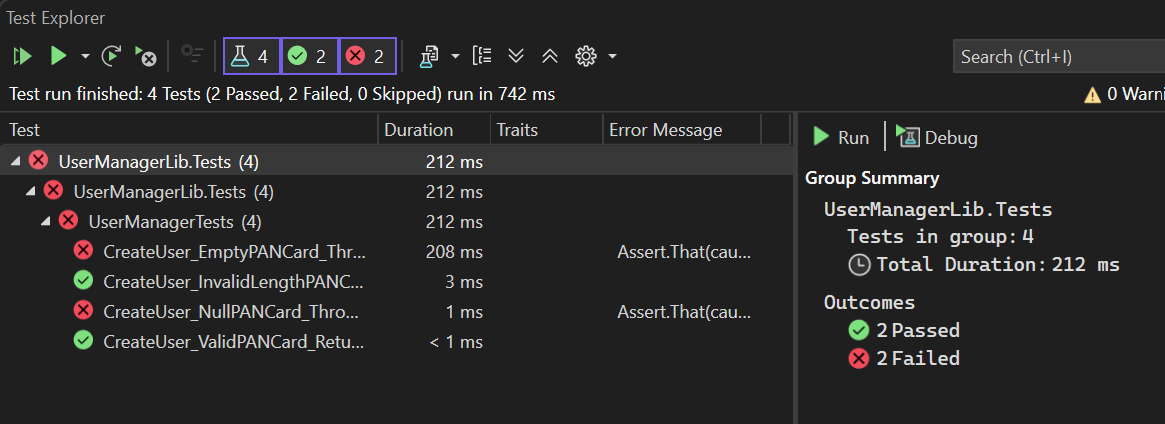
**Building of Project After Step 6**

**OUTPUT:-**



**Passing Test Case**

**OUTPUT:-**



**Name:- DOCUMENT 9\_NUnit-Handson**

**Solution:-**

**Code for IDollarToEuroExchangeRateFeed.cs in ConverterLib**

namespace ConverterLib

{

public interface IDollarToEuroExchangeRateFeed

{

double GetTheExchangeRate();

}

}

**Code for Converter.cs in ConverterLib**

namespace ConverterLib

{

public class Converter

{

private readonly IDollarToEuroExchangeRateFeed rateFeed;

public Converter(IDollarToEuroExchangeRateFeed feed)

{

this.rateFeed = feed;

}

public double USDToEuro(double amount)

{

return amount \* rateFeed.GetTheExchangeRate();

}

}

}

**Code for ConverterTests.cs in ConverterLib.Tests**

using ConverterLib;

using Moq;

using NUnit.Framework;

using System;

namespace ConverterLib.Tests

{

[TestFixture]

public class ConverterTests

{

private Mock<IDollarToEuroExchangeRateFeed> mockFeed;

private Converter converter;

[SetUp]

public void Setup()

{

mockFeed = new Mock<IDollarToEuroExchangeRateFeed>();

}

[Test]

public void USDToEuro\_ValidInputWithFixedRate\_ReturnsCorrectConversion()

{

// Arrange

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

converter = new Converter(mockFeed.Object);

// Act

double result = converter.USDToEuro(100);

// Assert

Assert.That(result, Is.EqualTo(85));

}

[Test]

public void USDToEuro\_ZeroInput\_ReturnsZero()

{

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

converter = new Converter(mockFeed.Object);

double result = converter.USDToEuro(0);

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

}

[Test]

public void USDToEuro\_NegativeInput\_ReturnsNegativeEuro()

{

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

converter = new Converter(mockFeed.Object);

double result = converter.USDToEuro(-50);

Assert.That(result, Is.EqualTo(-42.5));

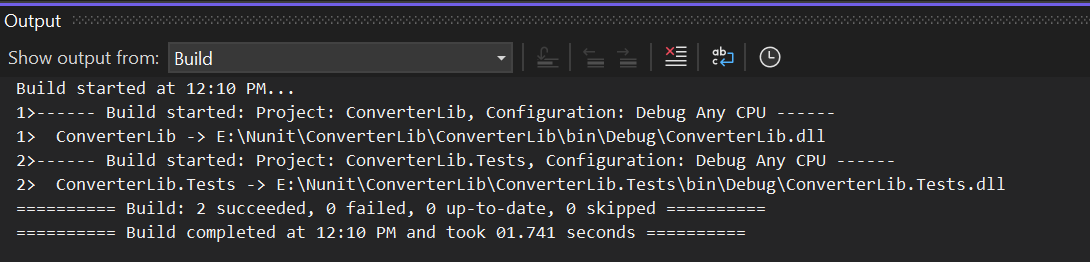
}

}

}

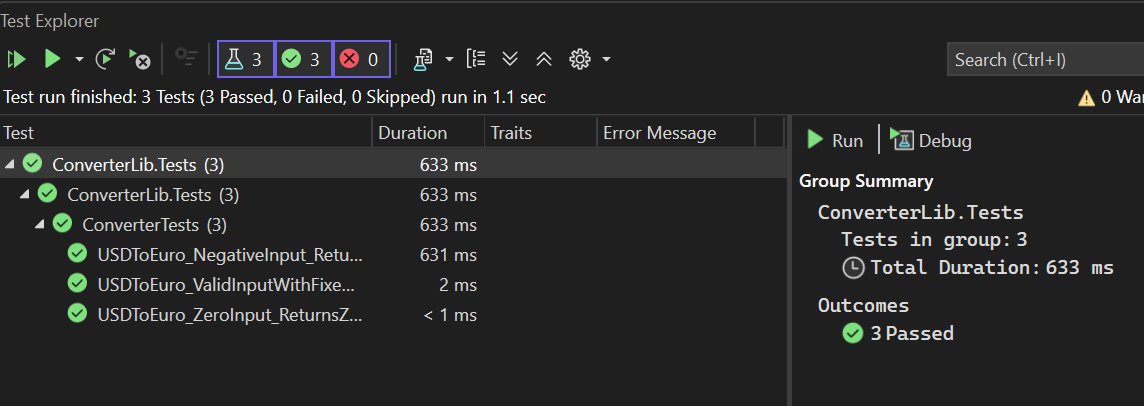
**Building of Project Upto Step 6**

**OUTPUT:-**



**Passing Test Case**

**OUTPUT:-**



***Break a Function & Re-Test***

**Code for Converter.cs in ConverterLib**

namespace ConverterLib

{

public class Converter

{

private readonly IDollarToEuroExchangeRateFeed rateFeed;

public Converter(IDollarToEuroExchangeRateFeed feed)

{

this.rateFeed = feed;

}

// ❌ Broken logic for testing purposes (Step 7)

public double USDToEuro(double amount)

{

// Wrong formula: should multiply, not add

return amount + rateFeed.GetTheExchangeRate();

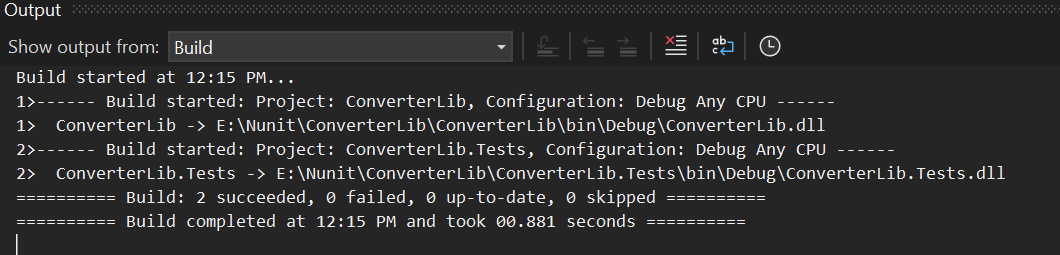
}

}

}

**Building of Project After Step 6**

**OUTPUT:-**



**Passing Test Case**

**OUTPUT:-**

