**916217-G Pavan Kumar**

**Stage3-NUnit Day2 hands-on-1**

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.

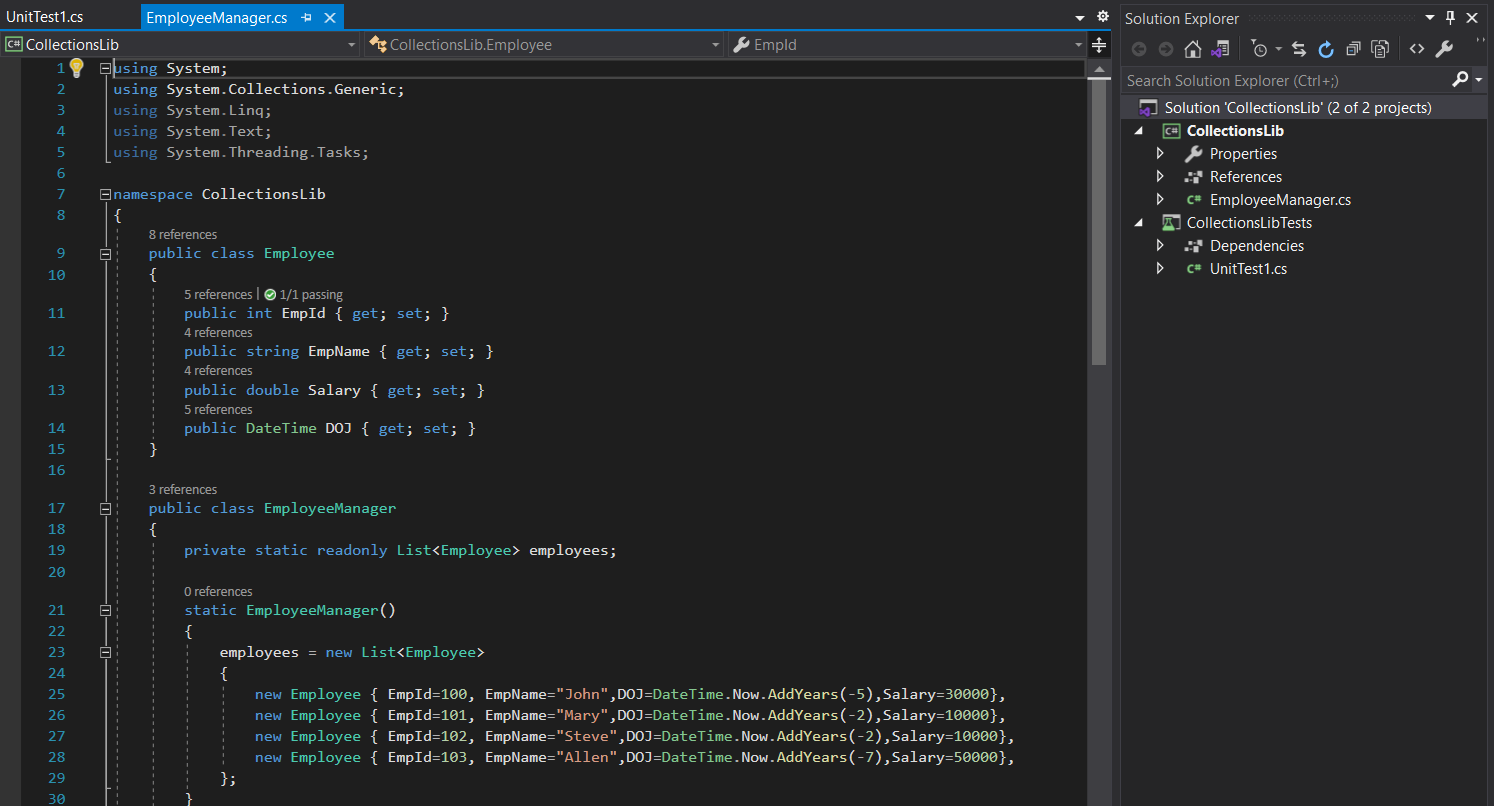
6) Run your tests.

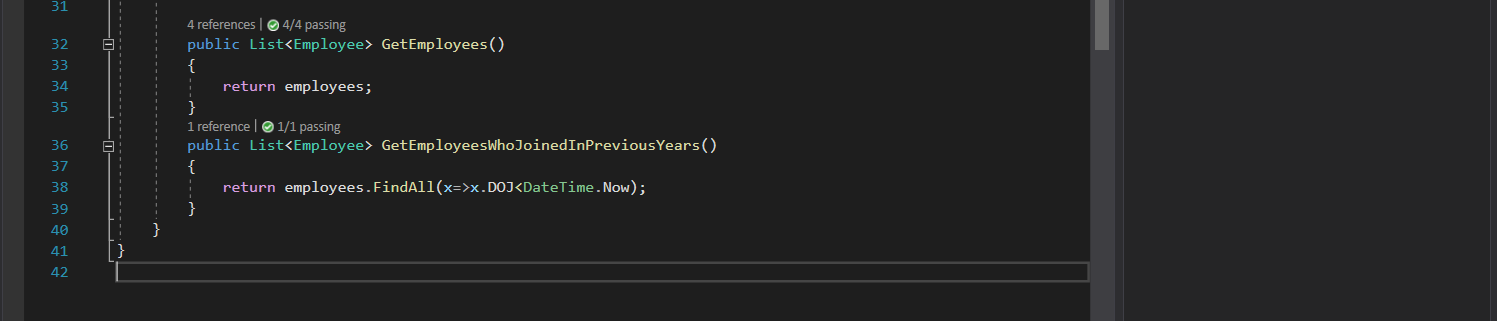
7) Break the test by modifying the source project functionality.

8) Rerun the test.

9) Observe the test result.

**Given code**





**Implementation**

**Collectionstests.cs**

using NUnit.Framework;

using CollectionsLib;

namespace CollectionsLibTests

{

[TestFixture]

public class SUT

{

EmployeeManager emp;

[SetUp]

public void SetUp()

{

emp = new EmployeeManager();

}

[TearDown]

public void Teardown()

{

emp = null;

}

[Test]

public void UnitUnderTest\_Scenario\_ExpectedOutcome()

{

var result = emp.GetEmployees().Contains(null);

Assert.IsFalse(result);

}

[Test]

public void UnitUnderTest\_Scenario\_ExpectedOutcome1()

{

var result = emp.GetEmployees().Find(t => t.EmpId == 100);

Assert.NotNull(result);

}

[Test]

public void UnitUnderTest\_Scenario\_ExpectedOutcome2()

{

var result = emp.GetEmployees();

Assert.That(result, Is.Unique);

}

[Test]

public void UnitUnderTest\_Scenario\_ExpectedOutcome3()

{

var result = emp.GetEmployees();

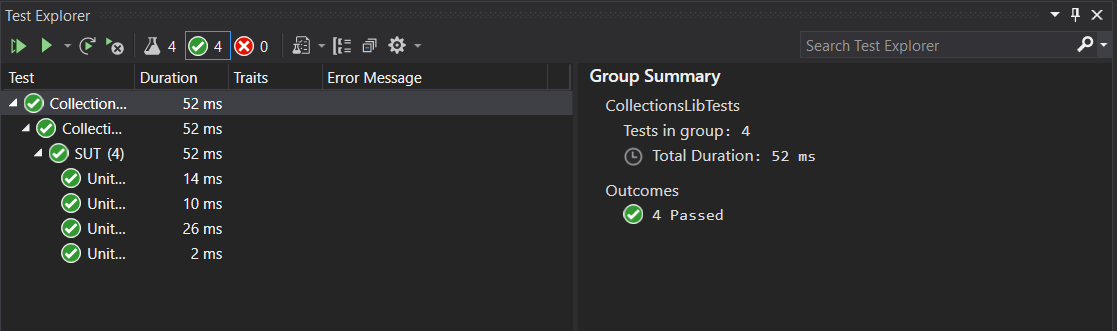
var result1 = emp.GetEmployeesWhoJoinedInPreviousYears();

Assert.That(result, Is.EqualTo(result1));

}

}}

Output



**Stage3-NUnit Day2 hands-on-2**

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 ConverterLib project to the test project.

4) Additionally add the reference of NUnit, NUnit3TestAdapter and Moq in the test project using NuGet Package Manager (NPM).

5) Write the suggested test methods.

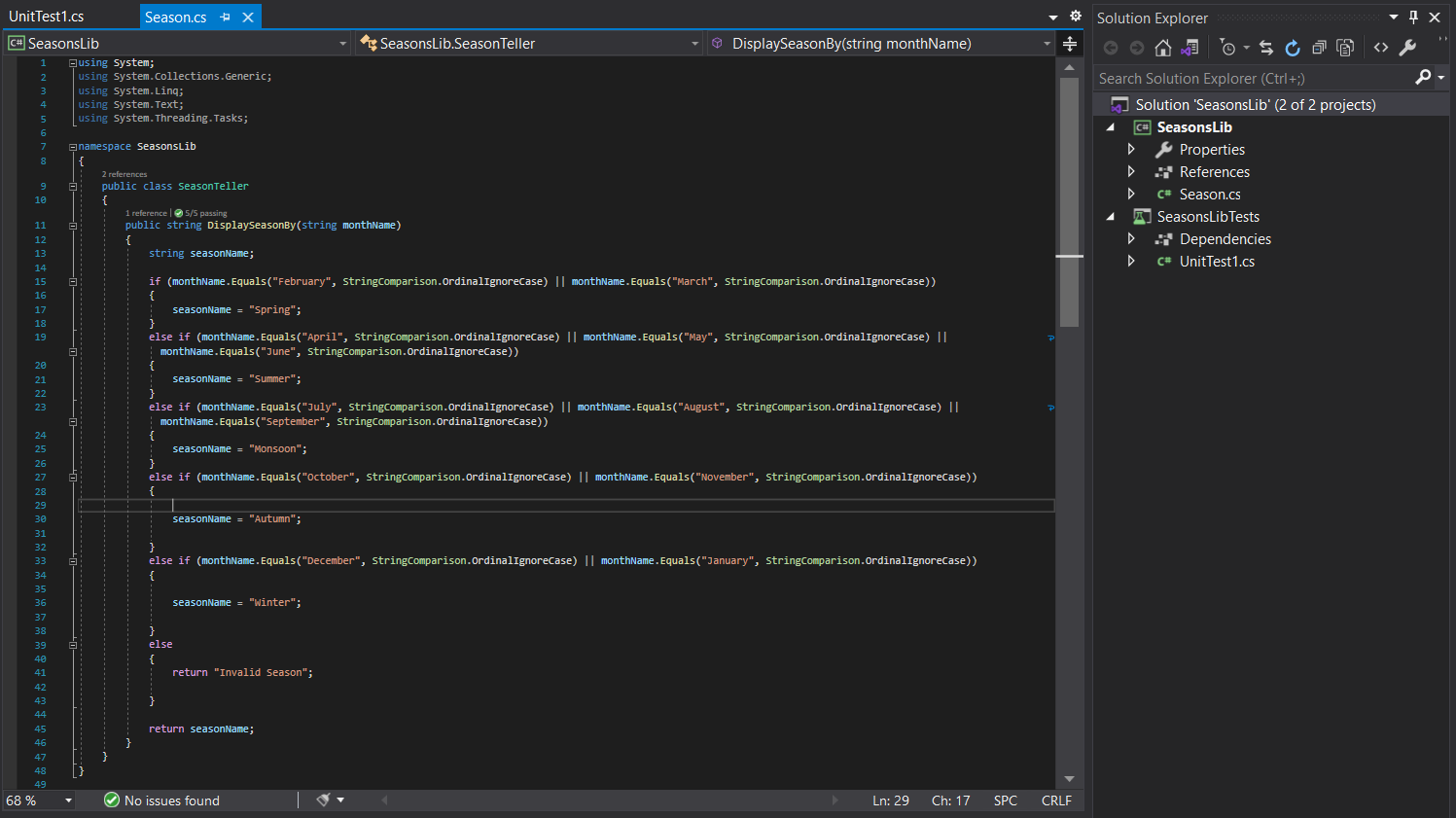
6) Run your tests.

7) Break the test by modifying the source project functionality.

8) Rerun the test.

9) Observe the test result.

**Given code**



**Implementation**

**SeasonsLibTests.cs**

using NUnit.Framework;

using SeasonsLib;

namespace SeasonsLibTests

{

public class SUT

{

SeasonTeller season;

[SetUp]

public void Setup()d

{

season = new SeasonTeller();

}

[TearDown]

public void Teardown()

{

season = null;

}

[Test]

[TestCase("February", "Spring")]

[TestCase("april", "Summer")]

[TestCase("July", "Monsoon")]

[TestCase("december", "Winter")]

[TestCase("ecember", "Invalid Season")]

public void UnitUnderTest\_Scenario\_ExpectedOutcome(string a,string expected)

{

string result = season.DisplaySeasonBy(a);

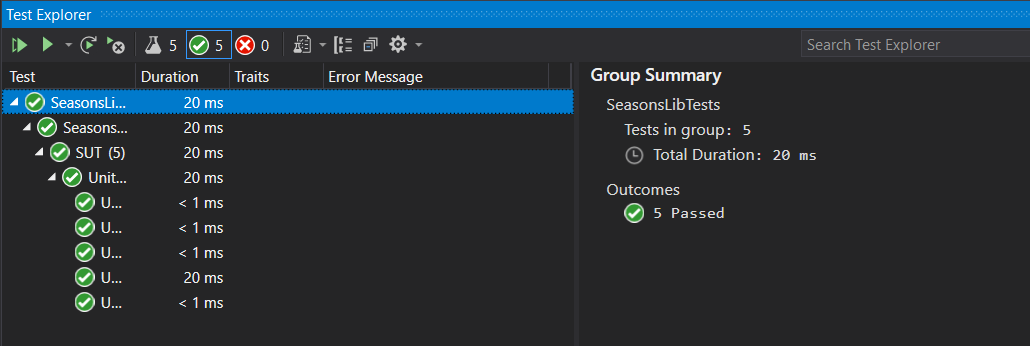
Assert.That(expected, Is.EqualTo(result));

}

}

}

**Output**



**Stage3-NUnit Day2 hands-on-3**

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.

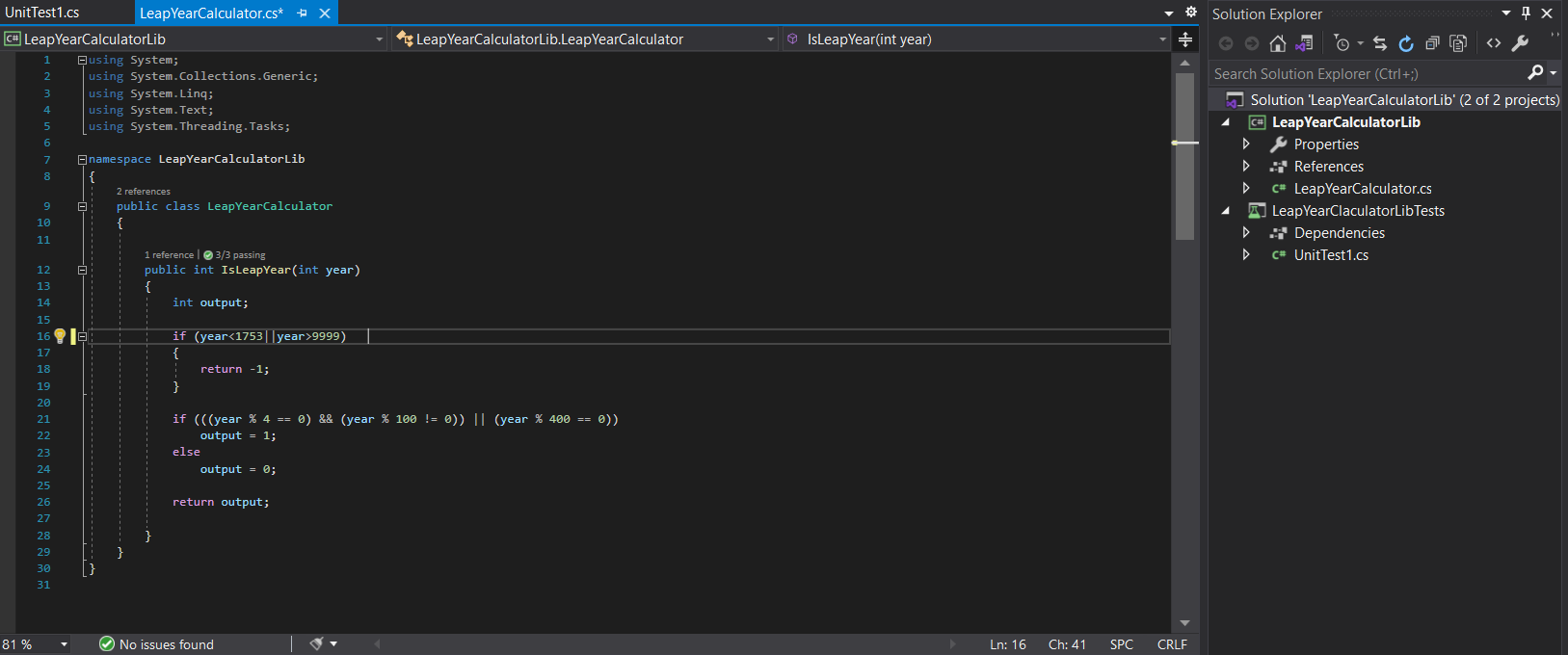
6) Run your tests.

7) Break the test by modifying the source project functionality.

8) Rerun the test.

9) Observe the test result

**Given code.**



**Implementation**

**Leapyeartest.cs**

using NUnit.Framework;

using LeapYearCalculatorLib;

namespace LeapYearClaculatorLibTests

{

public class SUT

{

LeapYearCalculator year;

[SetUp]

public void Setup()

{

year = new LeapYearCalculator();

}

[Test]

[TestCase(1740,-1)]

[TestCase(2000, 1)]

[TestCase(2002, 0)]

public void UnitUnderTest\_Scenario\_ExpectedOutcome(int a,int excepected)

{

int result = year.IsLeapYear(a);

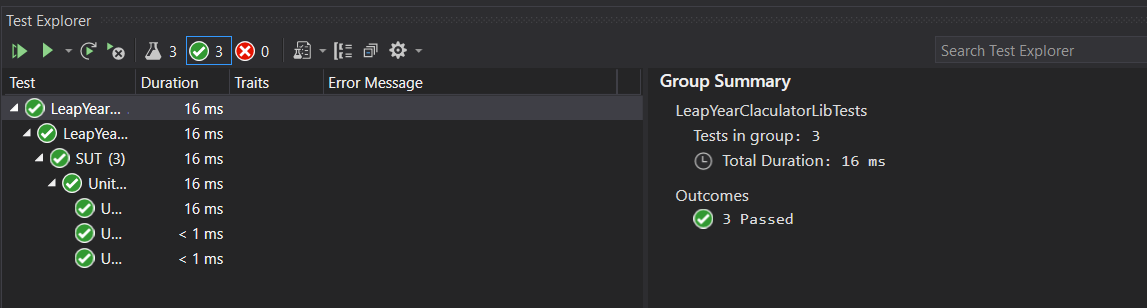
Assert.That(result,Is.EqualTo(excepected));

}

}

}

**Output**



**Stage3-NUnit Day2 hands-on-4**

PANCardNo property reads only 10 characters length value from the user. It is a mandatory property while creating the user.

Following exceptions may occur while creating the user.

o NullReferenceException- If the input value is empty or null

o FormatException-If the input string does not meet the length criteria.

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.

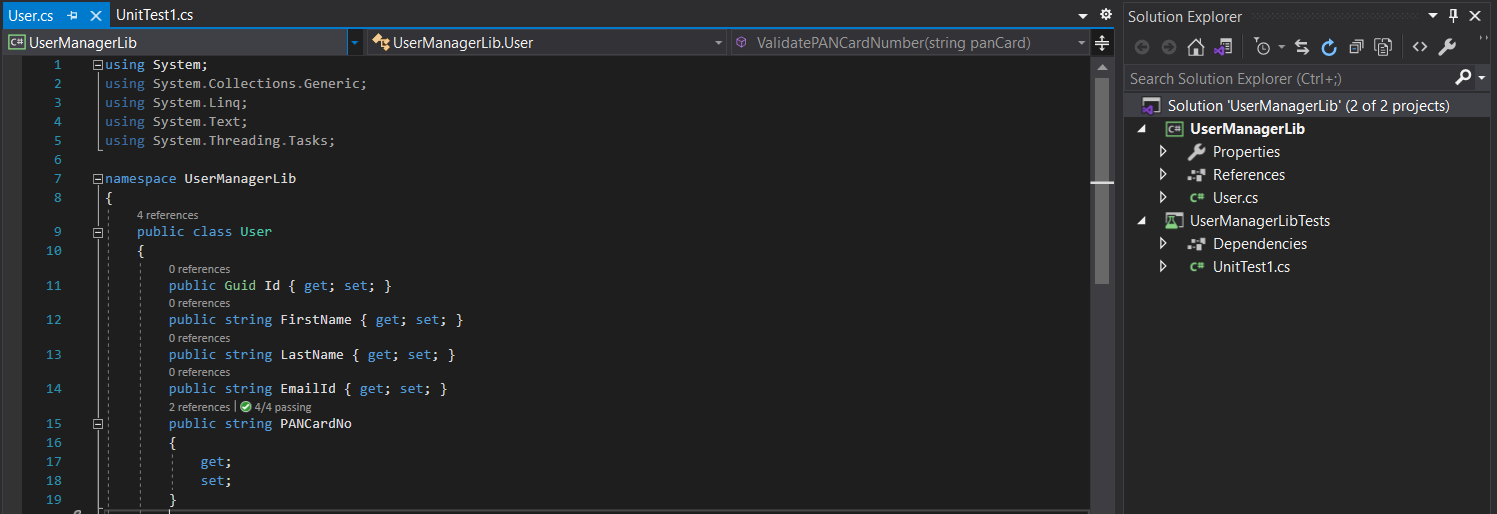
6) Run your tests.

7) Break the test by modifying the source project functionality.

8) Rerun the test.

9) Observe the test result.

**Given Code**





**Implemenatation**

**UserManagerLibTests**

using NUnit.Framework;

using UserManagerLib;

using System;

namespace UserManagerLibTests

{

public class Tests

{

User user;

[SetUp]

public void Setup()

{

user = new User();

}

[Test]

[TestCase("CYGPK00189")]

[TestCase("ABCDEFGHIJ")]

[TestCase("")]

[TestCase("HCGYJIL8")]

public void validpancard(string a)

{

try

{

user.CreateUser(new User { PANCardNo = a });

}

catch (NullReferenceException e)

{

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

}

catch (FormatException e)

{

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

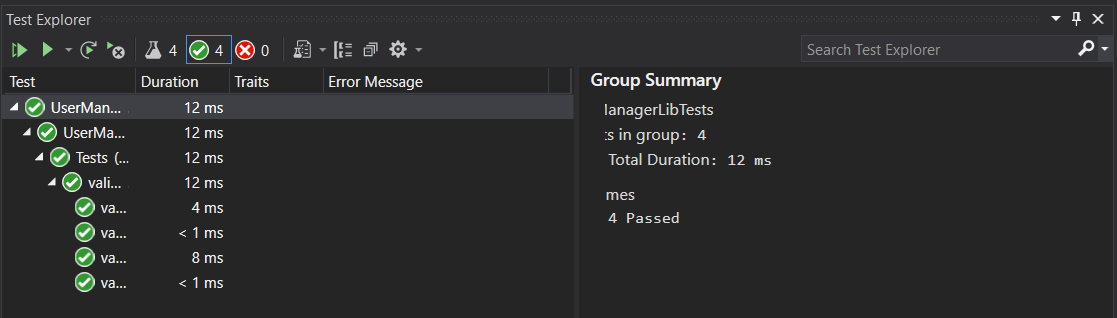
}

}

}

}

**Output**



**Stage3-NUnit Day2 hands-on-5**

One of the functionalities called USDToEuro which is defined in the Converter class should be your primary target while unit testing. It takes the US dollar as an input and convert it to Euro with the help of an external service, IDollarToEuroExchangeRateFeed. Since your application requires this functionality and the same can’t be tested while unit testing because you may not have a grip on the logic behind that service. On top of that, this particular functionality might have tested before it’s made available.

Use Moq framework in order to bypass the functionality which is defined in the IDollarToEuroExchangeRateFeed service.

Write test methods for the given functionalities to make sure that it returns the expected result under various circumstances.

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 ConverterLib project to the test project.

4) Additionally, add the reference of NUnit, NUnit3TestAdapter and Moq in the test project using NuGet Package Manager (NPM).

5) Write the suggested test methods.

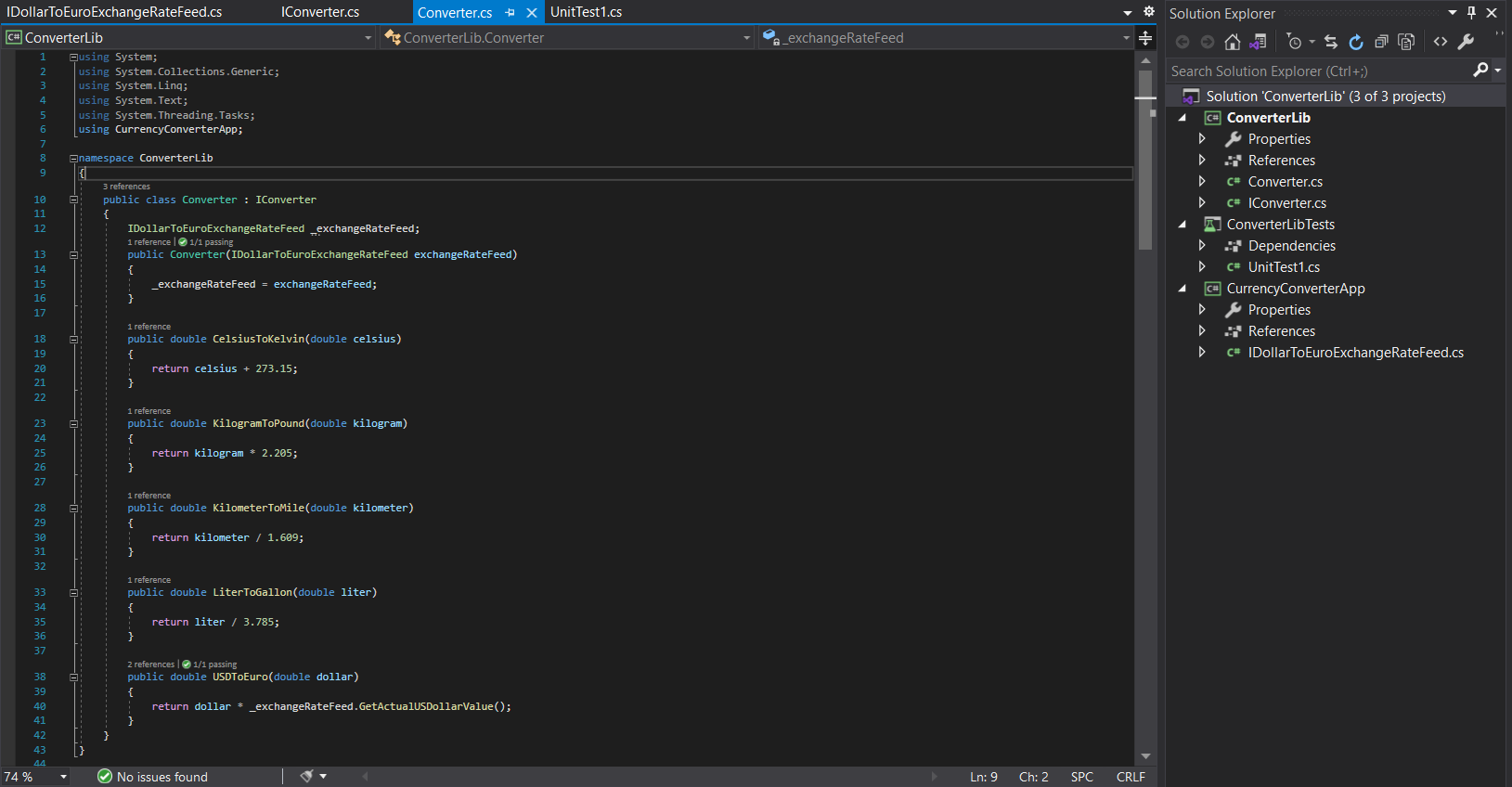
6) Run your tests.

7) Break the test by modifying the source project functionality.

8) Rerun the test.

9) Observe the test result.

**Given Code**



**Implemenatation**

**UnitTests.cs**

using NUnit.Framework;

using Moq;

using System;

using ConverterLib;

using CurrencyConverterApp;

namespace ConverterLibTests

{

public class SUT

{

Mock<IDollarToEuroExchangeRateFeed> excahnge;

[SetUp]

public void Setup()

{

excahnge = new Mock<IDollarToEuroExchangeRateFeed>();

}

[Test]

public void UnitUnderTest\_Scenario\_ExpectedOutcome()

{

double original = 100;

double add =1000;

excahnge.Setup(t => t.GetActualUSDollarValue()).Returns(original);

Converter convert = new Converter(excahnge.Object);

var euro = convert.USDToEuro(add);

Assert.That(euro, Is.EqualTo(100000.0d));

}

}

}

**output**

