**Hands-On: Stage 3 - NUnit and Mocking Framework - Day 2 – Handson 1**

You have been given a source project called CollectionsLib that deals with set of collection objects. Write test methods for the below scenarios. Make sure that your tests pass. You may modify the collection values in the source project in order to make the test passed. Use the appropriate assert functionalities.

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).

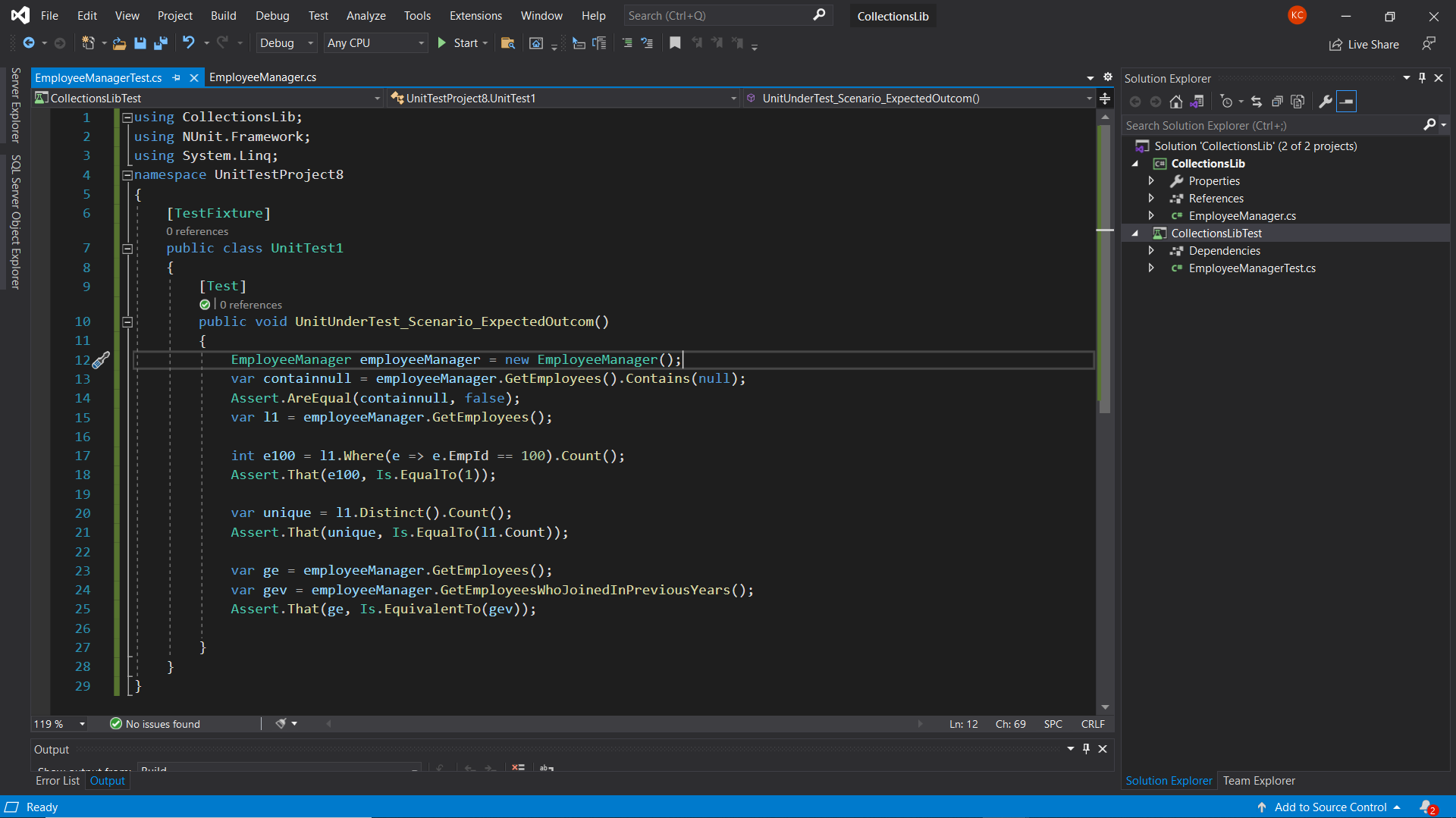
5) Write the suggested test methods.

6) Run your tests.

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

8) Rerun the test.

9) Observe the test result.

**PROJECT CODE:**

**IMPLEMENTATION:**

**EmployeeManager.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CollectionsLib

{

    public class Employee

    {

        public int EmpId { get; set; }

        public string EmpName { get; set; }

        public double Salary { get; set; }

        public DateTime DOJ { get; set; }

    }

    public class EmployeeManager

    {

        private static readonly List<Employee> employees;

        static EmployeeManager()

        {

            employees = new List<Employee>

            {

                new Employee { EmpId=100, EmpName="John",DOJ=DateTime.Now.AddYears(-5),Salary=30000},

                new Employee { EmpId=101, EmpName="Mary",DOJ=DateTime.Now.AddYears(-2),Salary=10000},

                new Employee { EmpId=102, EmpName="Steve",DOJ=DateTime.Now.AddYears(-2),Salary=10000},

                new Employee { EmpId=103, EmpName="Allen",DOJ=DateTime.Now.AddYears(-7),Salary=50000},

            };

        }

        public List<Employee> GetEmployees()

        {

            return employees;

        }

        public List<Employee> GetEmployeesWhoJoinedInPreviousYears()

        {

            return employees.FindAll(x=>x.DOJ<DateTime.Now);

        }

    }

}

**EmployeeManagerTest.cs**

using CollectionsLib;

using NUnit.Framework;

using System.Linq;

namespace UnitTestProject8

{

    [TestFixture]

    public class UnitTest1

    {

        [Test]

        public void UnitUnderTest\_Scenario\_ExpectedOutcom()

        {

            EmployeeManager employeeManager = new EmployeeManager();

            var containnull = employeeManager.GetEmployees().Contains(null);

            Assert.AreEqual(containnull, false);

            var l1 = employeeManager.GetEmployees();

            int e100 = l1.Where(e => e.EmpId == 100).Count();

            Assert.That(e100, Is.EqualTo(1));

            var unique = l1.Distinct().Count();

            Assert.That(unique, Is.EqualTo(l1.Count));

            var ge = employeeManager.GetEmployees();

            var gev = employeeManager.GetEmployeesWhoJoinedInPreviousYears();

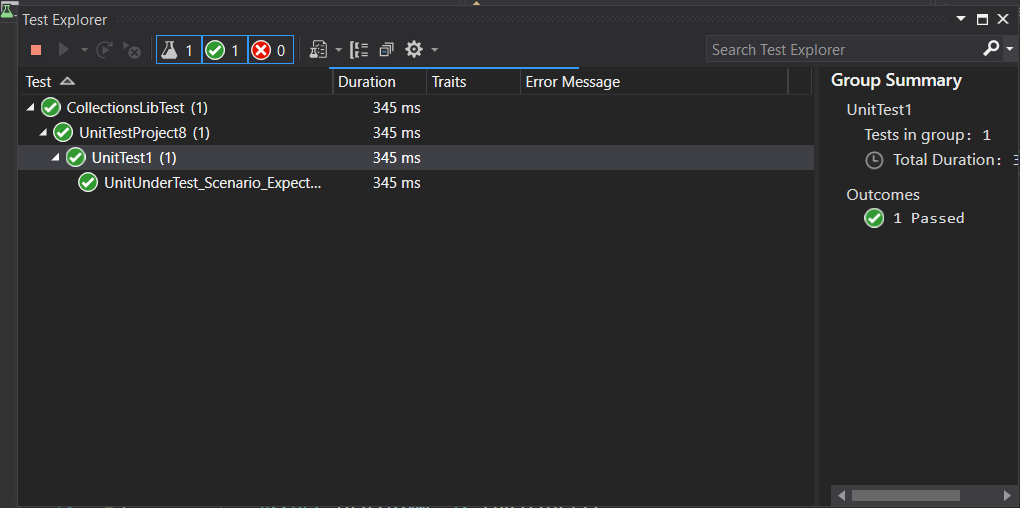
            Assert.That(ge, Is.EquivalentTo(gev));

        }

    }

}

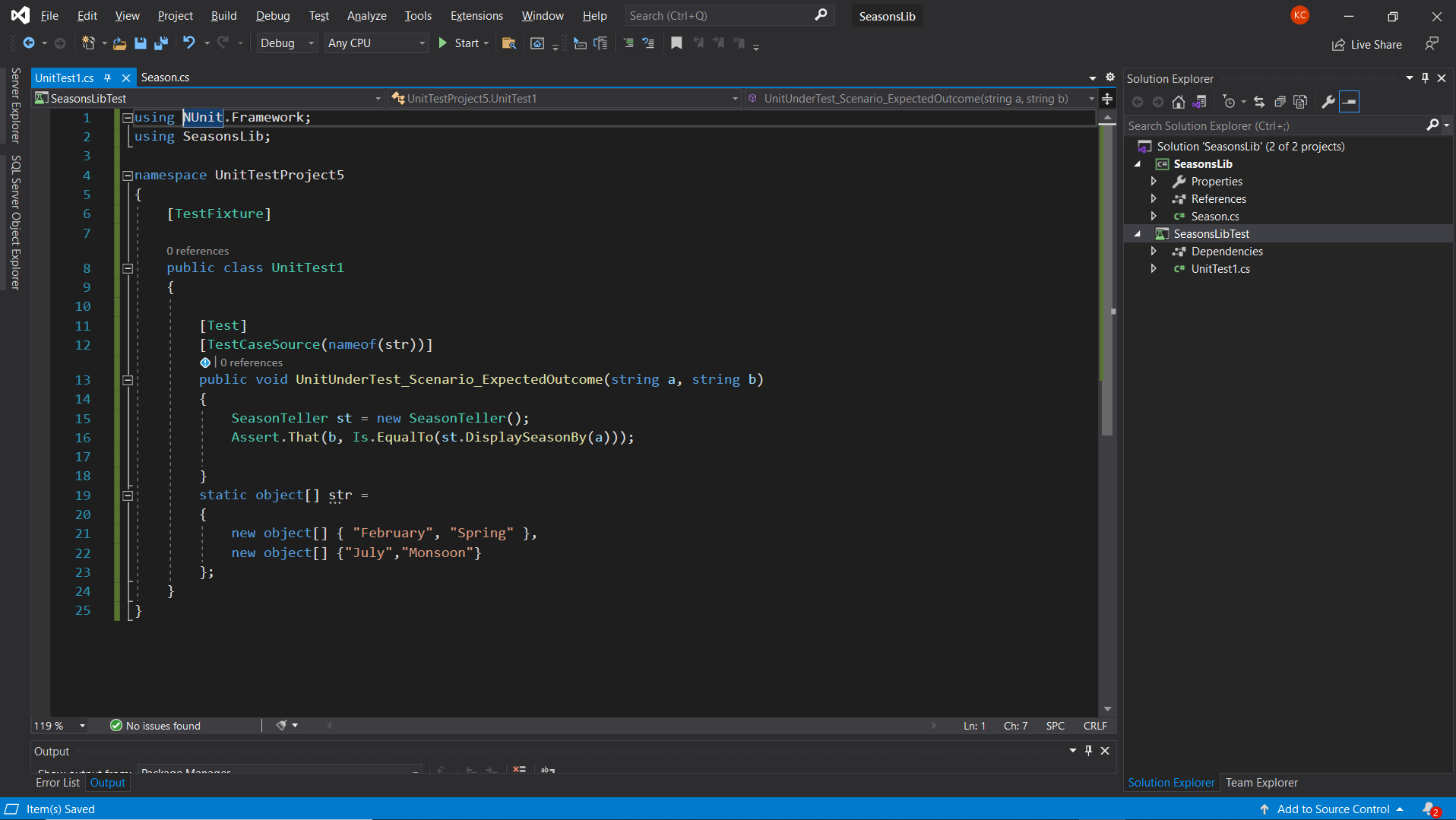
**OUTPUT:**



**Hands-On: Stage 3 - NUnit and Mocking Framework - Day 2 – Handson 2**

Create a Unit Test Project using NUnit Framework for the following requirement. Click here to download the source project FourSeasonsLib.

Try both straight forward and alternate ways of working with the TestCaseSource attribute for the above scenario.

**PROJECT CODE:**

**IMPLEMENTATION:**

**SeasonTeller.cs**

using System;

namespace SeasonsLib

{

    public class SeasonTeller

    {

        public string DisplaySeasonBy(string monthName)

        {

            string seasonName;

            if (monthName.Equals("February", StringComparison.OrdinalIgnoreCase) || monthName.Equals("March", StringComparison.OrdinalIgnoreCase))

            {

                seasonName = "Spring";

            }

            else if (monthName.Equals("April", StringComparison.OrdinalIgnoreCase) || monthName.Equals("May", StringComparison.OrdinalIgnoreCase) || monthName.Equals("June", StringComparison.OrdinalIgnoreCase))

            {

                seasonName = "Summer";

            }

            else if (monthName.Equals("July", StringComparison.OrdinalIgnoreCase) || monthName.Equals("August", StringComparison.OrdinalIgnoreCase) || monthName.Equals("September", StringComparison.OrdinalIgnoreCase))

            {

                seasonName = "Monsoon";

            }

            else if (monthName.Equals("October", StringComparison.OrdinalIgnoreCase) || monthName.Equals("November", StringComparison.OrdinalIgnoreCase))

            {

                seasonName = "Autumn";

            }

            else if (monthName.Equals("December", StringComparison.OrdinalIgnoreCase) || monthName.Equals("January", StringComparison.OrdinalIgnoreCase))

            {

                seasonName = "Winter";

            }

            else

            {

                return "Invalid Season";

            }

            return seasonName;

        }

    }

}

**SUT.cs**

using NUnit.Framework;

using SeasonsLib;

namespace UnitTestProject5

{

    [TestFixture]

    public class SUT

    {

        [Test]

        [TestCaseSource(nameof(str))]

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

        {

            SeasonTeller st = new SeasonTeller();

            Assert.That(b, Is.EqualTo(st.DisplaySeasonBy(a)));

        }

        static object[] str =

        {

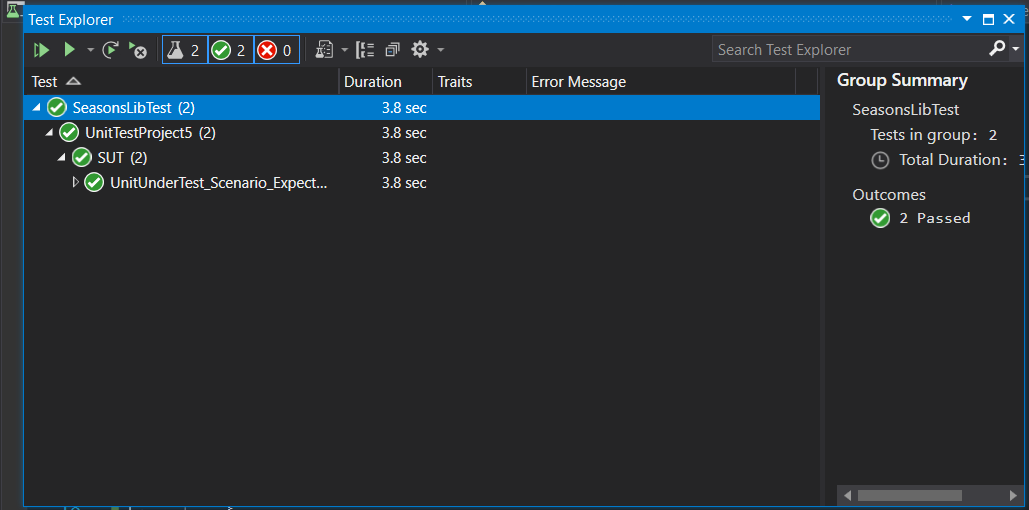
            new object[] { "February", "Spring" },

            new object[] {"July","Monsoon"}

        };

    }

}

**OUTPUT**

**Hands-On: Stage 3 - NUnit and Mocking Framework - Day 2 – Handson 3**

The LeapYearCalculatorLib application tells the user whether the entered year is a leap year or not. Also it checks the input value to make sure that the data given is a valid year.

· If the given year is a Leap Year, the program will output 1. If it’s not, then the program will result 0.

· Any value between 1753 and 9999 (both inclusive) will be a valid year. Violation of this rule will result -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).

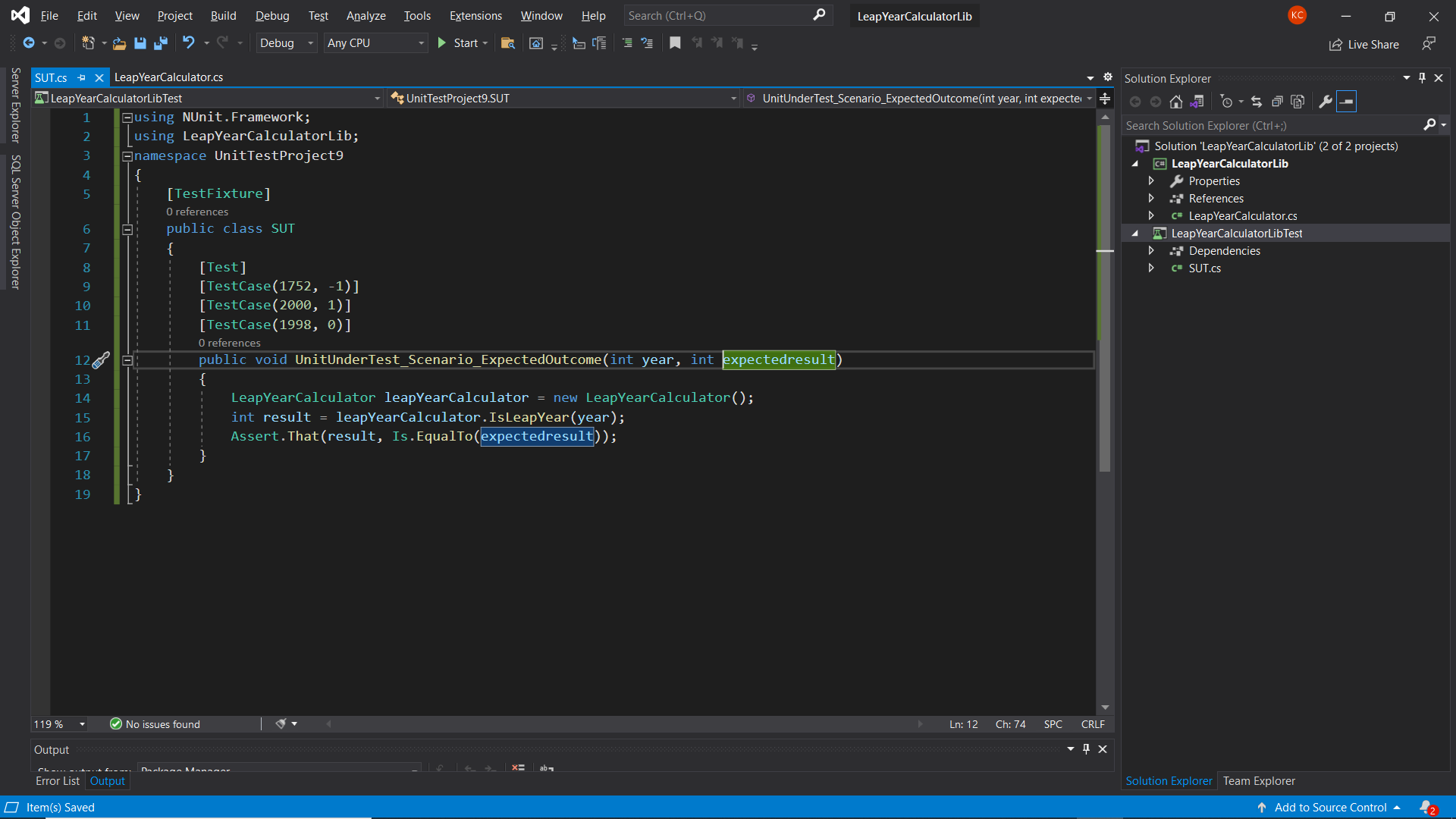
5) Write the suggested test methods.

6) Run your tests.

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

8) Rerun the test.

9) Observe the test result.

**PROJECT CODE:**

**IMPLEMENTATION:**

**LeapYearCalculator.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace LeapYearCalculatorLib

{

    public class LeapYearCalculator

    {

        public int IsLeapYear(int year)

        {

            int output;

            if (year<1753||year>9999)

            {

                return -1;

            }

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

                output = 1;

            else

                output = 0;

            return output;

        }

    }

}

**SUT.cs**

using NUnit.Framework;

using LeapYearCalculatorLib;

namespace UnitTestProject9

{

    [TestFixture]

    public class SUT

    {

        [Test]

        [TestCase(1752, -1)]

        [TestCase(2000, 1)]

        [TestCase(1998, 0)]

        public void UnitUnderTest\_Scenario\_ExpectedOutcome(int year, int expectedresult)

        {

            LeapYearCalculator leapYearCalculator = new LeapYearCalculator();

            int result = leapYearCalculator.IsLeapYear(year);

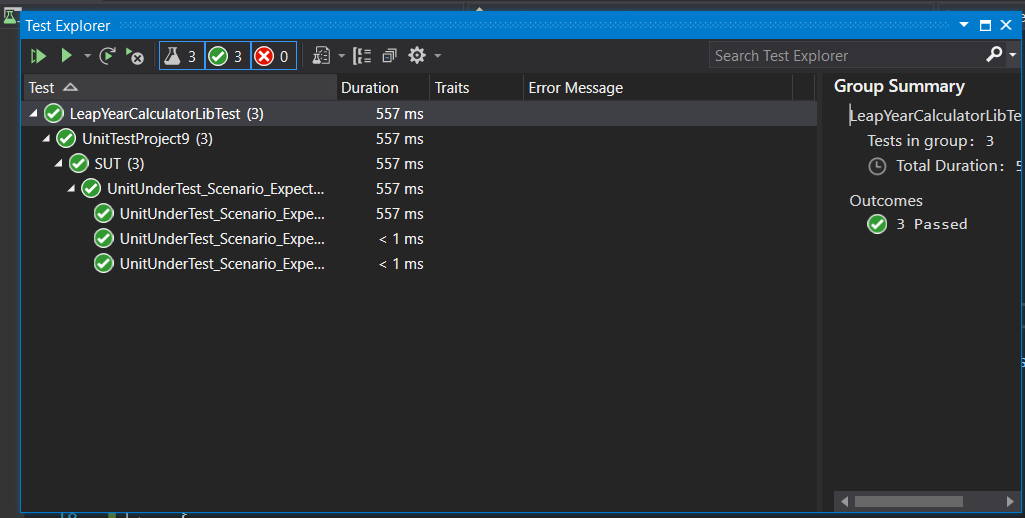
            Assert.That(result, Is.EqualTo(expectedresult));

        }

    }

}

**OUTPUT:**



**Hands-On: Stage 3 - NUnit and Mocking Framework - Day 2 – Handson 4**

Following is the application logic of the given UserManagerLib source project.

User creation will be successful under the below given condition

· 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.

· While writing test cases for the above program, you need to ensure that you are handling all types of exception that may raise during the CreateUser method call. Also, write test method for happy path in the function.

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).

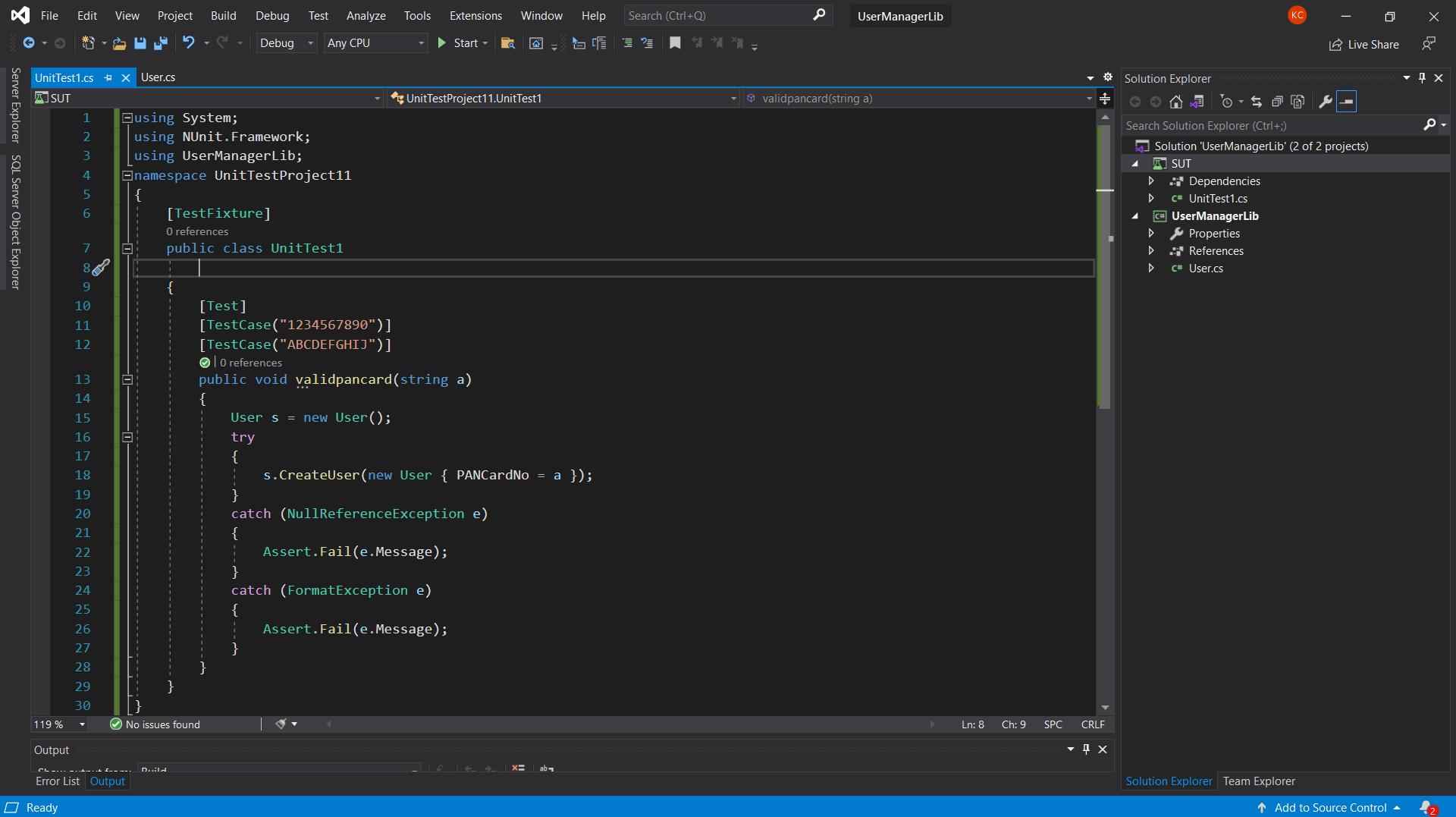
5) Write the suggested test methods.

6) Run your tests.

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

8) Rerun the test.

9) Observe the test result.

**PROJECT CODE:**

**IMPLEMENTATION:**

**User.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace UserManagerLib

{

    public class User

    {

        public Guid Id { get; set; }

        public string FirstName { get; set; }

        public string LastName { get; set; }

        public string EmailId { get; set; }

        public string PANCardNo

        {

            get;

            set;

        }

        public string ValidatePANCardNumber(string panCard)

        {

            if (string.IsNullOrEmpty(panCard))

            {

                throw new NullReferenceException("Invalid Pan Card Number");

            }

            else if (panCard.Length != 10)

            {

                throw new FormatException("Pan Card Number Should contain only 10 characters");

            }

            else

            {

                return "Valid";

            }

        }

        public void CreateUser(User user)

        {

            if (ValidatePANCardNumber(user.PANCardNo).Equals("Valid"))

            {

                //Do something

            }

        }

    }

}

**UnitTest1.cs**

using System;

using NUnit.Framework;

using UserManagerLib;

namespace UnitTestProject11

{

    [TestFixture]

    public class UnitTest1

    {

        [Test]

        [TestCase("1234567890")]

        [TestCase("ABCDEFGHIJ")]

        public void validpancard(string a)

        {

            User s = new User();

            try

            {

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

            }

            catch (NullReferenceException e)

            {

                Assert.Fail(e.Message);

            }

            catch (FormatException e)

            {

                Assert.Fail(e.Message);

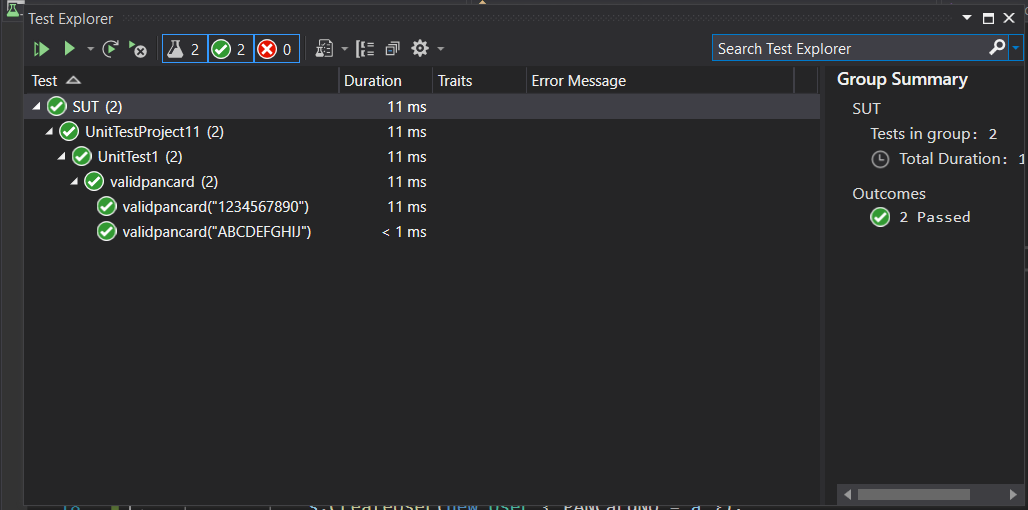
            }

        }

    }

}

**OUTPUT:**



**Hands-On: Stage 3 - NUnit and Mocking Framework - Day 2 – Handson 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).

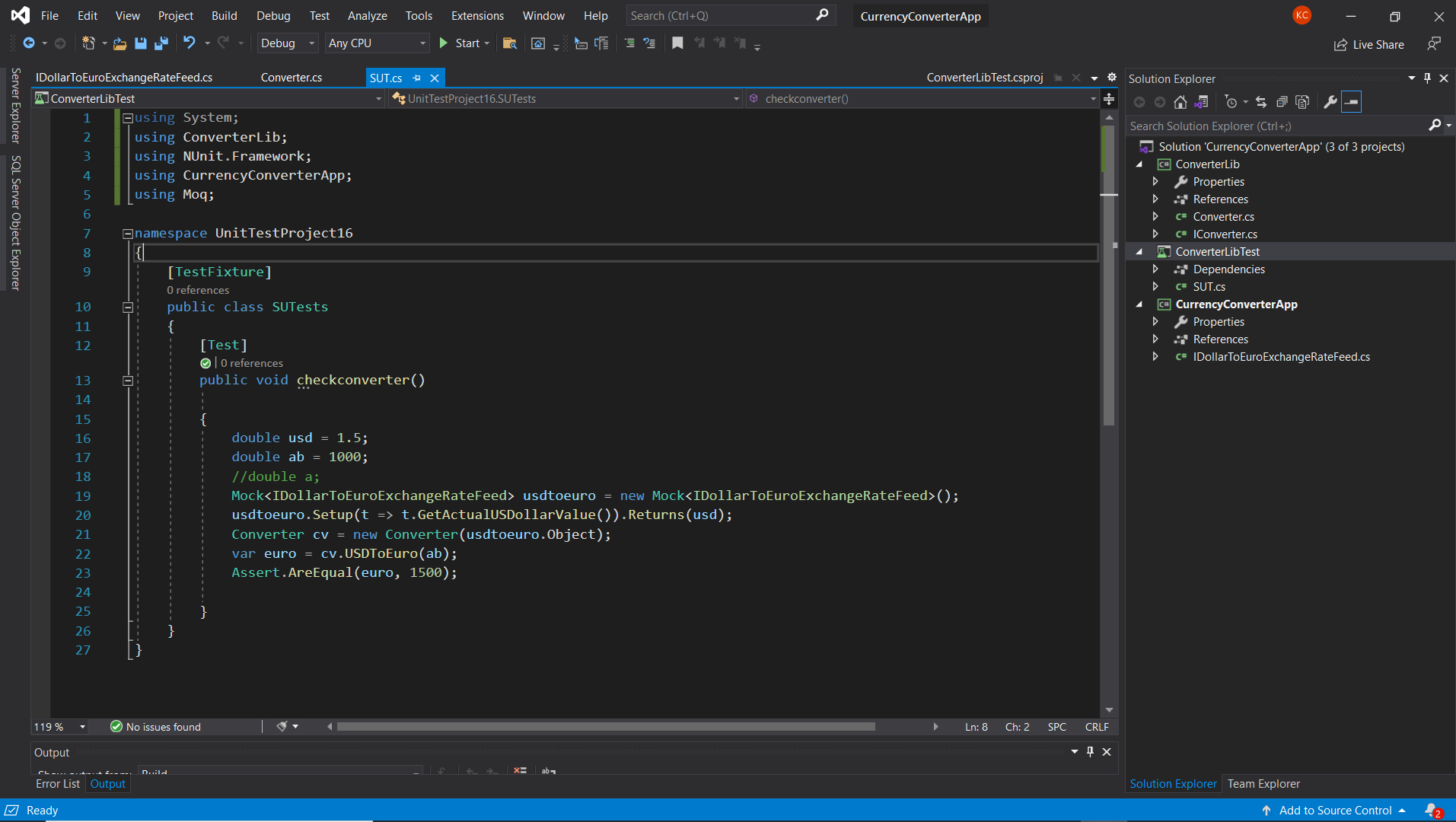
5) Write the suggested test methods.

6) Run your tests.

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

8) Rerun the test.

9) Observe the test result.

**PROJECT CODE:**

**IMPLEMENTATION:**

**Converter.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using CurrencyConverterApp;

namespace ConverterLib

{

    public class Converter : IConverter

    {

        IDollarToEuroExchangeRateFeed \_exchangeRateFeed;

        public Converter(IDollarToEuroExchangeRateFeed exchangeRateFeed)

        {

            \_exchangeRateFeed = exchangeRateFeed;

        }

        public double CelsiusToKelvin(double celsius)

        {

            return celsius + 273.15;

        }

        public double KilogramToPound(double kilogram)

        {

            return kilogram \* 2.205;

        }

        public double KilometerToMile(double kilometer)

        {

            return kilometer / 1.609;

        }

        public double LiterToGallon(double liter)

        {

            return liter / 3.785;

        }

        public double USDToEuro(double dollar)

        {

            return dollar \* \_exchangeRateFeed.GetActualUSDollarValue();

        }

    }

}

**IDollarToEuroExchangeRateFeed.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CurrencyConverterApp

{

    public interface IDollarToEuroExchangeRateFeed

    {

        double GetActualUSDollarValue();

    }

}

**SUT.cs**

using System;

using ConverterLib;

using NUnit.Framework;

using CurrencyConverterApp;

using Moq;

namespace UnitTestProject16

{

    [TestFixture]

    public class SUTests

    {

        [Test]

        public void checkconverter()

        {

            double usd = 1.5;

            double ab = 1000;

            //double a;

            Mock<IDollarToEuroExchangeRateFeed> usdtoeuro = new Mock<IDollarToEuroExchangeRateFeed>();

            usdtoeuro.Setup(t => t.GetActualUSDollarValue()).Returns(usd);

            Converter cv = new Converter(usdtoeuro.Object);

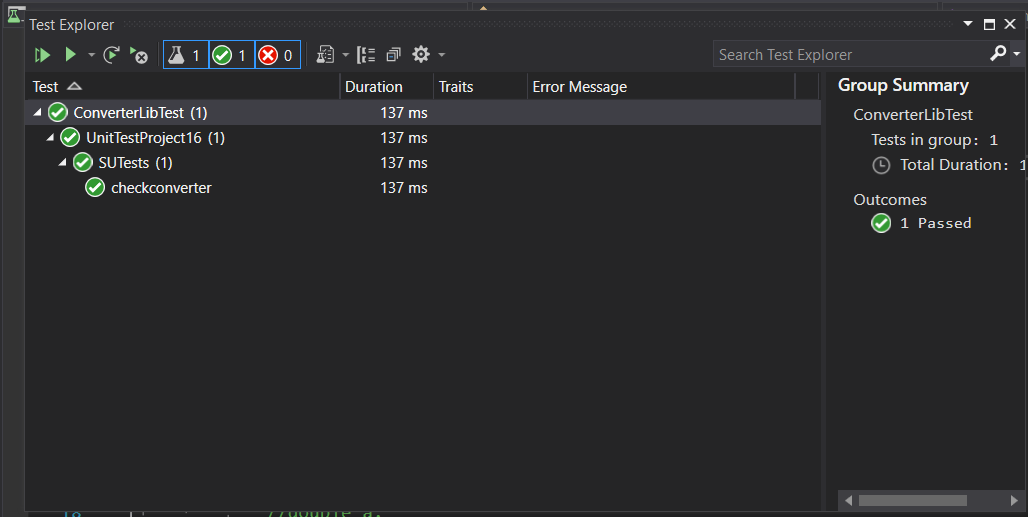
            var euro = cv.USDToEuro(ab);

            Assert.AreEqual(euro, 1500);

        }

    }

}

**OUTPUT:**