INTCDE21ID008

STAGE-3

916483 – Naveen S

Day 1 – NUnit

Hands-On 1:

Follow the steps listed below to write the NUnit test cases for the application.

1) Create a Unit test project(.NET Framework) in the solution provided.

2) Add the CalcLibrary project as reference

3) Create a class “CalculatorTests” to write all the test cases for the methods in the solution

4) Use the ‘TestFixture’, ‘SetUp’ and ‘TearDown’ attributes, to declare, initialize and cleanup activities respectively

5) Create a Test method to check the addition functionality

6) Use the ‘TestCase’ attribute to send the inputs and the expected result

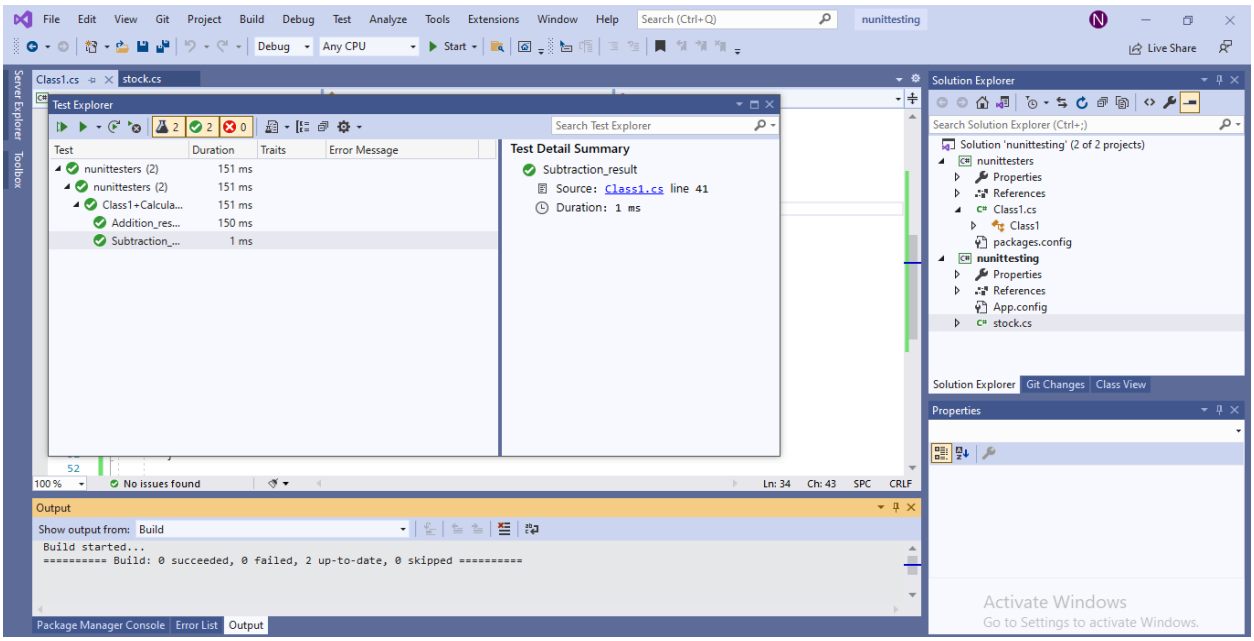
7) Use Assert.That to check the actual and expected result match

IMPLEMENTATION:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using NUnit.Framework;
using nunittesting;

namespace nunittesters
{
    [TestFixture]
    public class Class1

    {
        [TestFixture]
        public class CalculatorTests
        {
```

```
            stock st;
            double Result;
            [SetUp]
            public void SetUp()
            {
                st = new stock();
                Result = 8;
            }
            [TearDown]
            public void TearDown()
            {
                st = null;
            }
            [Test]
            public void Addition_result()
            {
                double expectedResult = 12;

                Result = st.Add(2);

                Assert.AreEqual(expectedResult, Result);
            }
            [Test]
            public void Subtraction_result()
            {

                double expectedResult = 5;

                Result = st.sub(5);

                Assert.AreEqual(expectedResult, Result);
            }

        }

    }
}
```

OUTPUT:



Hands-On 2:

Follow the steps listed below to write the NUnit test cases for the application.

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

2) Create test cases with ‘TestCase’ attribute to send in input parameters and the expected result.

3) Add more than 1 ‘TestCase’ attributes to check various combinations for subtractions.

4) Use Assert.Equal to check the actual and expected results

5) Create a test case to verify the multiplication concepts of calculator

6) Create test cases with ‘TestCase’ attribute to send in input parameters and the expected result.

7) Add more than 1 ‘TestCase’ attributes to check various combinations for subtractions.

8) Use Assert.Equal to check the actual and expected results

9) Create a test case to verify the division logic of the calculator

10) Create test cases with ‘TestCase’ attribute to send in input parameters and the expected result.

11) Add more than 1 ‘TestCase’ attributes to check various combinations for subtractions.

12) Use Assert.Equal to check the actual and expected results

13) In one of the inputs, provide the divisor value to be 0

14) Use Try Catch block to catch the ArgumentException

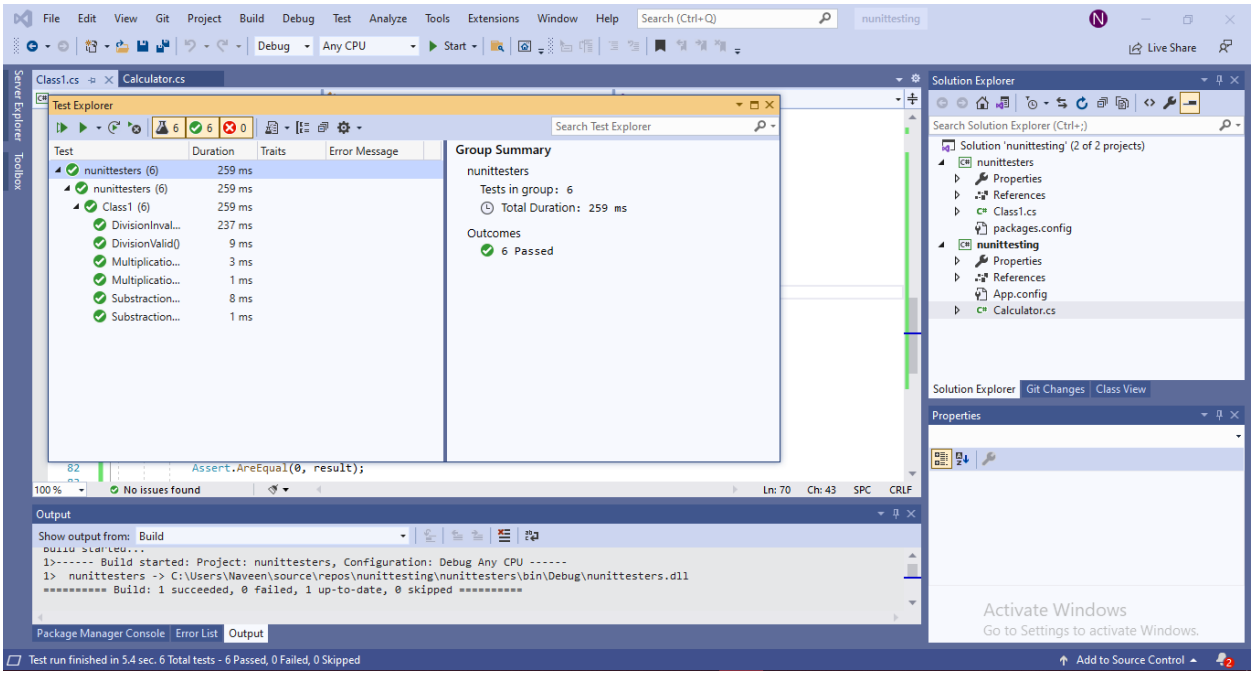
15) Use Assert.Fail to notify the user that the test case has failed. Give the message “Division by zero” in the Assert.Fail, which will be notified to the user. This message will be seen in the test explorer.

IMPLEMENTATION:

```
using System;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using NUnit.Framework;
using nunittesting;

namespace nunittesters
{
    [TestFixture]
    class Class1
    {
        Calculator cal;
        private double result;
        [SetUp]
        public void SetUp()
        {
            cal = new Calculator();
        }
        [TearDown]
        public void TearDown()
        {
            cal = null;
        }
        [TestCase]
        public void SubstractionBig_to_small()
        {
            double ares = cal.Subtraction(30, 20);
            double eres = 10;
            Assert.That(ares, Is.EqualTo(eres));
```

```
        }
        [TestCase]
        public void SubstractionSmall_to_big()
        {
            double ares = cal.Subtraction(20, 30);
            double eres = -10;
            Assert.That(ares, Is.EqualTo(eres));
        }
        [TestCase]
        public void Multiplication()
        {
            double ares = cal.Multiplication(5, 4);
            double eres = 20;
            Assert.That(ares, Is.EqualTo(eres));
        }
        [TestCase]
        public void Multiplication2()
        {
            double ares = cal.Multiplication(4, 4);
            double eres = 16;
            Assert.That(ares, Is.EqualTo(eres));
        }
        [TestCase]
        public void DivisionValid()
        {
            double ares = cal.Division(10, 10);
            Assert.That(1, Is.EqualTo(ares));
        }
        [TestCase]
        public void DivisionInvalid()
        {
            try
            {
                double f = cal.Division(10, 0);

            }
            catch (Exception v)
            {
                Assert.AreEqual("Attempted to divide by zero.", v.Message);

            }
        }
        [TearDown]

        public void CleanUp()

        {
            cal.AllClear();
            result = cal.GetResult();
            Console.WriteLine(result);
            Assert.AreEqual(0, result);

        }
    }

}
```

OUTPUT:



Hands-On 3:

Follow the steps listed below to write the NUnit test cases for the application.

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.

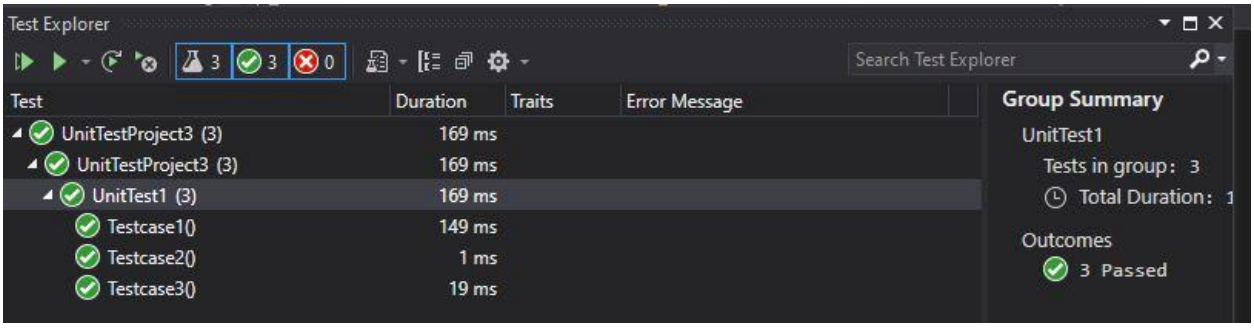
8) Rerun the test.

9) Observe the test result.

IMPLEMENTATION:

```
using System;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using NUnit.Framework;
using nunittesting;

namespace nunittesters
{
    [TestFixture]
    class Class1
    {
        UrlParser u;

        [SetUp]
        public void SetUp()
        {
            u = new UrlParser();
        }
        [TearDown]
        public void Dispose()
        {
            u = null;
        }

        [TestCase]
        public void Testcase1()
        {
            string act = url.ParseHostName("https://google.com");
            string exp = "Facebook.com";
            Assert.That(act, Is.EqualTo(exp));
        }
        [TestCase]
        public void Testcase2()
        {

            string act = url.ParseHostName("http://e-resume3.000webhostap.com");
            string exp = "twitter.com";
            Assert.That(act, Is.EqualTo(exp));
        }
        [TestCase]
        public void Testcase3()
        {
            var ex = Assert.Throws<FormatException>(() =>
url.ParseHostName("https12://gmail.com"));
            Assert.That(ex.Message, Is.EqualTo("Url is not in correct format"));
        }
    }
}
```

OUTPUT:



Hands-On 4:

Follow the steps listed below to write the NUnit test cases for the application.

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.

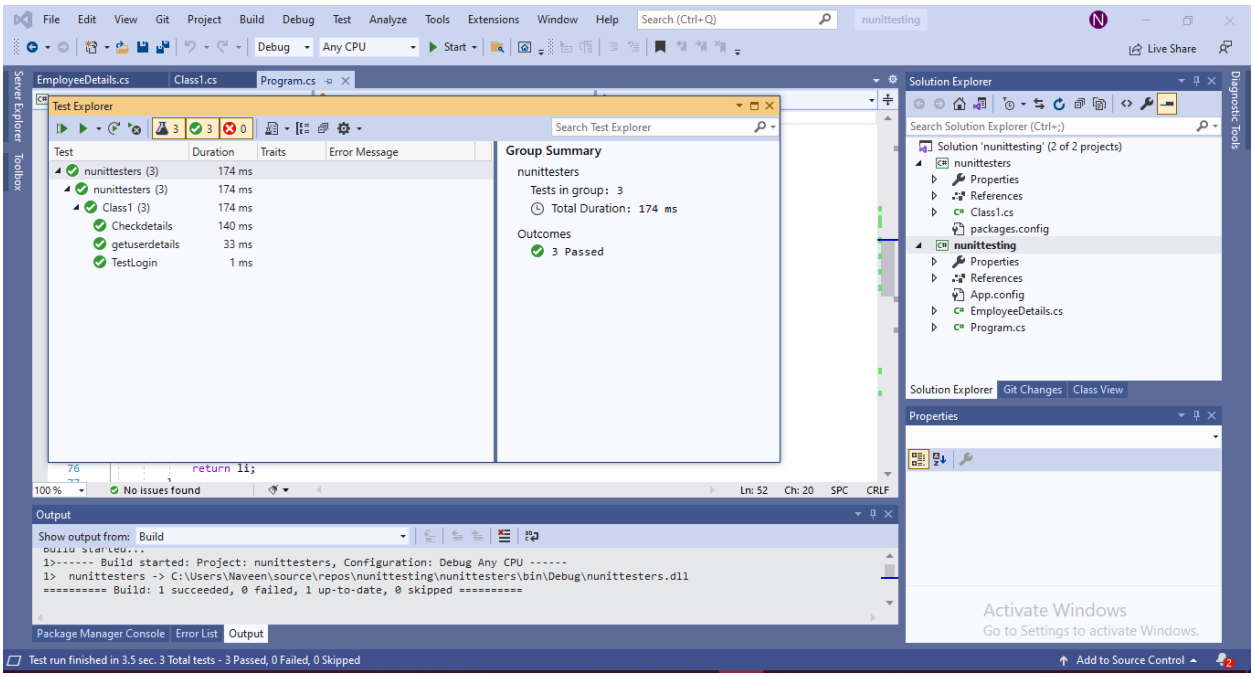
9) Observe the test result.

IMPLEMENTATION:

```
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Text;
using System.Threading.Tasks;
using NUnit.Framework;
using nunittesting;


namespace nunittesters
{
    [TestFixture]
    public class Class1
    {
            List<EmployeeDetails> li;
            [Test]
            public void Checkdetails()
            {
                Program pobj = new Program();
                li = pobj.AllUsers();
                foreach (var x in li)
                {
                    Assert.IsNotNull(x.id);
                    Assert.IsNotNull(x.Name);
                    Assert.IsNotNull(x.salary);
                    Assert.IsNotNull(x.Gender);
                }
            }
            [Test]
            public void TestLogin()
            {
                Program pobj = new Program();
                string x = pobj.Login("Ajit", "1234");
                string y = pobj.Login("", "");
            string n = pobj.Login("Naveen", "Hai");
                string z = pobj.Login("Admin", "Admin");
                Assert.AreEqual("Userid or password could not be Empty.", y);
                Assert.AreEqual("Incorrect UserId or Password.", x);
                Assert.AreEqual("Welcome Admin.", z);
                Assert.AreEqual("Welcome developer", n);
            }
            [Test]
            public void getuserdetails()
            {
                Program pobj = new Program();
                var p = pobj.getDetails(100);
                foreach (var x in p)
                {
                    Assert.AreEqual(x.id, 100);
                    Assert.AreEqual(x.Name, "Bharat");
                }
            }

        }
    }
```

OUTPUT:


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
nunittesting
Debug
Any CPU
Start
Live Share
EmployeeDetails.cs
Class1.cs
Program.cs
Test Explorer
Search Test Explorer
Test
Duration
Traits
Error Message
nunittesters (3) 174 ms
nunittesters (3) 174 ms
Class1 (3) 174 ms
Checkdetails 140 ms
getuserdetails 33 ms
TestLogin 1 ms
Group Summary
nunittesters
Tests in group: 3
Total Duration: 174 ms
Outcomes
3 Passed
return 11;
No issues found
Ln: 52 Ch: 20 SPC CRLF
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'nunittesting' (2 of 2 projects)
nunittesters
Properties
References
Class1.cs
packages.config
nunittesting
Properties
References
App.config
EmployeeDetails.cs
Program.cs
Solution Explorer
Git Changes
Class View
Properties
Output
Show output from: Build
1>------ Build started: Project: nunittesters, Configuration: Debug Any CPU ------
1> nunittesters -> C:\Users\Naveen\source\repos\nunittesting\nunittesters\bin\Debug\nunittesters.dll
========== Build: 1 succeeded, 0 failed, 1 up-to-date, 0 skipped ==========
Package Manager Console
Error List
Output
Activate Windows
Go to Settings to activate Windows.
Test run finished in 3.5 sec. 3 Total tests - 3 Passed, 0 Failed, 0 Skipped
Add to Source Control