**Nunit Sai Kumaravelu A.T.**

**916531**

**Hands on-1**

***TestFixture & Test Please download the application available here. 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***

*UnitTest1.cs*

*using Microsoft.VisualStudio.TestTools.UnitTesting;*

*using System;*

*using NUnit.Framework;*

*using Nunithandson1;*

*using Assert = NUnit.Framework.Assert;*

*namespace UnitTestProject1*

*{*

*[TestFixture]*

*public class UnitTest1*

*{*

*Program pro;*

*[SetUp]*

*public void SetUp()*

*{*

*pro = new Program();*

*}*

*[TearDown]*

*public void TearDown()*

*{*

*pro = null;*

*}*

*[TestCase]*

*public void add()*

*{*

*double ares = pro.Addition(10, 12);*

*double e = 22;*

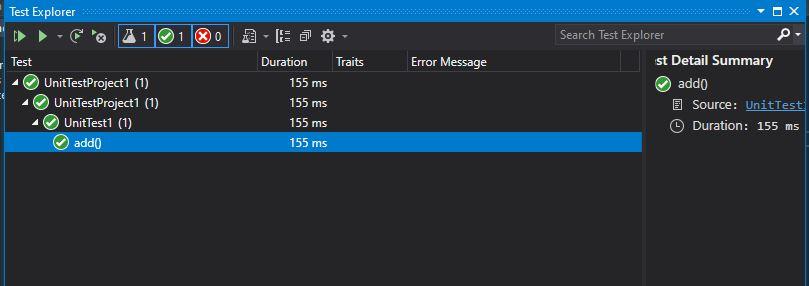
*Assert.That(ares,Is.EqualTo(e));*

*}*

*}*

*}*

***Result:***

******

**Hands on – 2**

***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***

***2. 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***

***3. 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.***

Handson2.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using NUnit.Framework;

using Nunithandson1;

namespace UnitTestProject1

{

[TestFixture]

class Handson2

{

Program pro;

[SetUp]

public void SetUp()

{

pro = new Program();

}

[TearDown]

public void TearDown()

{

pro = null;

}

[TestCase]

public void SubstractionTestcase1()

{

double a = pro.Subtraction(30, 20);

double e = 10;

Assert.That(a, Is.EqualTo(e));

}

[TestCase]

public void SubstractionTestcase2()

{

double a= pro.Subtraction(20, 30);

double e= -10;

Assert.That(a, Is.EqualTo(e));

}

[TestCase]

public void MultiplicationTestCase1()

{

double a = pro.Multiplication(5,4);

double e = 20;

Assert.That(a, Is.EqualTo(e));

}

[TestCase]

public void MultiplicationTestCase2()

{

double a = pro.Multiplication(0, 4);

double e =0;

Assert.That(a, Is.EqualTo(e));

}

[TestCase]

public void DivisionTestcase1()

{

double a = pro.Division(100, 10);

double e = 10;

Assert.That(a, Is.EqualTo(e));

}

[TestCase]

public void DivisionTestcase2()

{

var ero = Assert.Throws<ArgumentException>(() => pro.Division(100, 0));

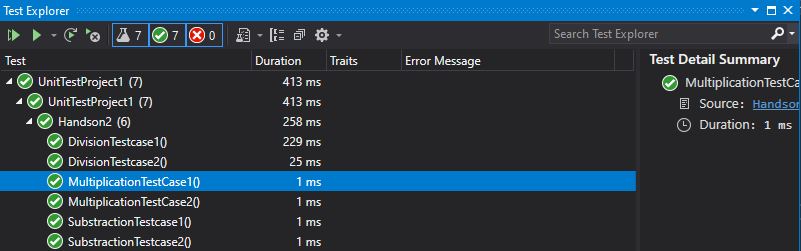
Assert.AreEqual("Second Parameter Can't be Zero", ero.Message);

}

}

}

**Result:**

****

**Hands on-3**

**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 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.**

UnitTest1.cs

using Microsoft.VisualStudio.TestTools.UnitTesting;

using System;

using NUnit.Framework;

using nunithandson3;

using Assert = NUnit.Framework.Assert;

namespace UnitTestProject3

{

[TestFixture]

public class UnitTest1 {

UrlHostNameParser u;

[SetUp]

public void SetUp()

{

u = new UrlHostNameParser();

}

[TearDown]

public void Dispose()

{

u = null;

}

[TestCase]

public void Testcase1()

{

string act = u.ParseHostName("https://Facebook.com");

string e = "Facebook.com";

Assert.That(act, Is.EqualTo(e));

}

[TestCase]

public void Testcase2()

{

string a = u.ParseHostName("http://twitter.com");

string e= "twitter.com";

Assert.That(a, Is.EqualTo(e));

}

[TestCase]

public void Testcase3()

{

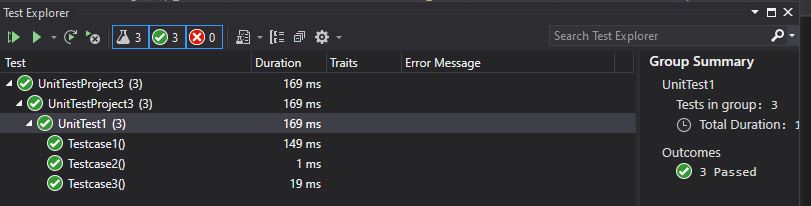
var e = Assert.Throws<FormatException>(() => u.ParseHostName("https12://gmail.com"));

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

}

}

}

****

**Hands on-4**

**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 Framework for the following requirement. Click here to download the source project.**

**You are given a user account module called AccountsManagerLib using which an employee can login to the EMS portal using his/her credentials. If he/she provides valid login credentials, then the function should return the following message.**

**“Welcome <user\_id>!!!”**

**In case user provides invalid details, the function should return the following error message.**

**“Invalid user id/password”**

**If user doesn’t provide either userid or password, the function should throw an ArgumentException.**

**Valid Credentials are the following**

**User Id Password**

**user\_11 secret@user11**

**user\_22 secret@user22**

**Recommendations:**

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

**Test Class Name: <SUT>Tests**

**Test Method Name: UnitUnderTest\_Scenario\_ExpectedOutcome**

**After writing the above test methods, run the tests and assert the results with that of the success/failure messages.**

**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.**

UnitTest1.cs

using Microsoft.VisualStudio.TestTools.UnitTesting;

using System;

using NUnit.Framework;

using nunithandson3;

using Assert = NUnit.Framework.Assert;

namespace UnitTestProject3

{

[TestFixture]

public class UnitTest1 {

UrlHostNameParser u;

[SetUp]

public void SetUp()

{

u = new UrlHostNameParser();

}

[TearDown]

public void Dispose()

{

u = null;

}

[TestCase]

public void LoginTestcase1()

{

string exp = "Invalid user id/password";

string act = u.ValidateUser("user\_22", "secret@user12s");

Assert.That(act, Is.EqualTo(exp));

}

[TestCase]

public void LoginTestcase2()

{

string act = u.ValidateUser("user\_11", "secret@user11");

string exp = string.Format("Invalid user id/password");

Assert.That(act, Is.EqualTo(exp));

}

[TestCase]

public void LoginTestcase3()

{

string act = u.ValidateUser("user\_12", "secret@user12");

string exp = string.Format("Welcome user\_12!!!");

Assert.That(act, Is.EqualTo(exp));

}

[TestCase]

public void LoginTestcase4()

{

string exp = "Invalid user id/password";

string act = u.ValidateUser("user\_23", "secret@user12s");

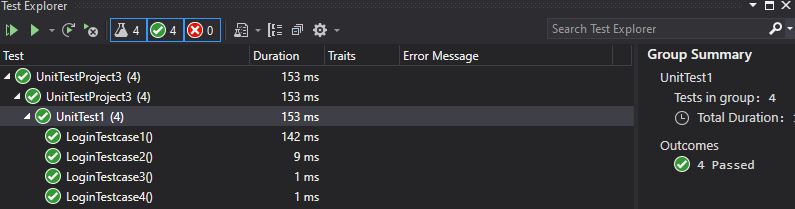
Assert.That(act, Is.EqualTo(exp));

}

}

}

**Result:**

****