N-Unit

**UNIT**In this IT world a unit refers to simply a smallest piece of code which takes an input ,does certain operation, and gives an output.

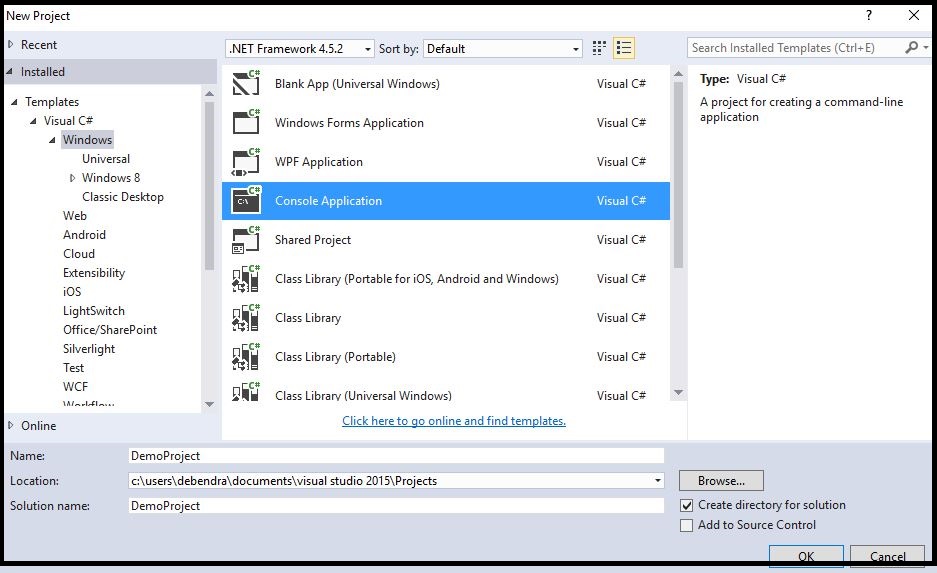
And testing this small piece of code is called Unit Testing. A lot of unit test frameworks are available for .Net nowadays, if we check in Visual Studio we have MSTest from Microsoft integrated in Visual Studio. Some 3rd party frameworks are:

* NUnit
* MbUnit

Out of all these Nunit is the most-used testing Framework.

What Is NUnit?

NUnit is a unit-testing framework for all .Net languages. NUnit is Open Source software and NUnit 3.0 is released under the MIT license. This framework is very easy to work with and has user friendly attributes for working



Write the following class in this project.

namespace DemoProject {

public class EmployeeDetails {

public int id { get; set; }

public string Name {get; set; }

public double salary { get; set; }

public string Geneder { get; set; }

}

}

Now in the class program write some conditions which need to be tested.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DemoProject {

public class Program {

public string Login(string UserId, string Password) {

if (string.IsNullOrEmpty(UserId) || string.IsNullOrEmpty(Password)) {

return "Userid or password could not be Empty.";

} else {

if (UserId == "Admin" && Password == "Admin") {

return "Welcome Admin.";

}

return "Incorrect UserId or Password.";

}

}

public List < EmployeeDetails > AllUsers() {

List < EmployeeDetails > li = new List < EmployeeDetails > ();

li.Add(new EmployeeDetails {

id = 100, Name = "Bharat", Geneder = "male", salary = 40000

});

li.Add(new EmployeeDetails {

id = 101, Name = "sunita", Geneder = "Female", salary = 50000

});

li.Add(new EmployeeDetails {

id = 103, Name = "Karan", Geneder = "male", salary = 40000

});

li.Add(new EmployeeDetails {

id = 104, Name = "Jeetu", Geneder = "male", salary = 23000

});

li.Add(new EmployeeDetails {

id = 105, Name = "Manasi", Geneder = "Female", salary = 80000

});

li.Add(new EmployeeDetails {

id = 106, Name = "Ranjit", Geneder = "male", salary = 670000

});

return li;

}

public List < EmployeeDetails > getDetails(int id) {

List < EmployeeDetails > li1 = new List < EmployeeDetails > ();

Program p = new Program();

var li = p.getUserDetails();

foreach(var x in li) {

if (x.id == id) {

li1.Add(x);

}

}

return li1;

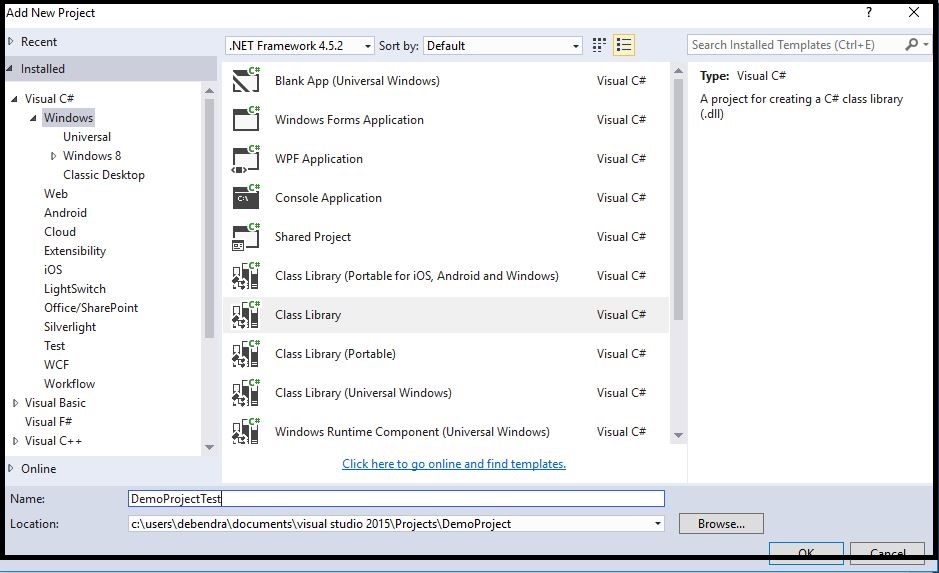
}

static void Main(string[] args) {}

}

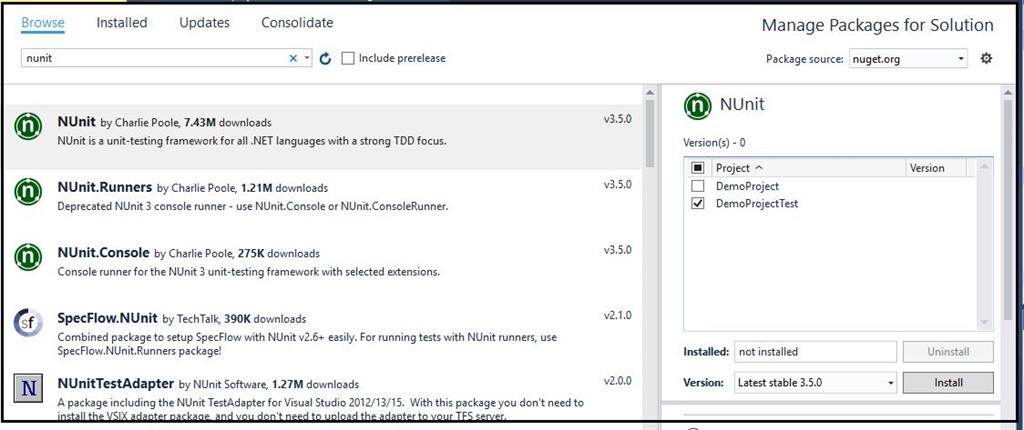
}

So here I have written 3 methods or we can say some requirements which we want to test by using Unit Test.



The Nunit framework does not require any specific project type, but most of the time people will add a class library to separate their code from their unit tests. You need to reference the nunit.framework.dll yourself. But if you use nuget this will be done for you automatically while installing NUnit.

So for using Nunit we need to go for Nuget Package Manager and download it.

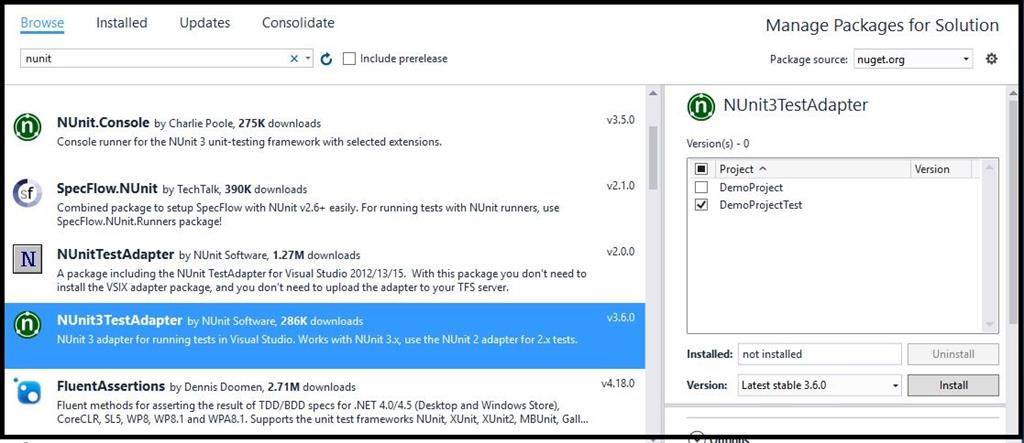


Now after installing the Nunit we need one more dll that needs to be installed in the project.

## NUnit Test Adapter for Visual Studio

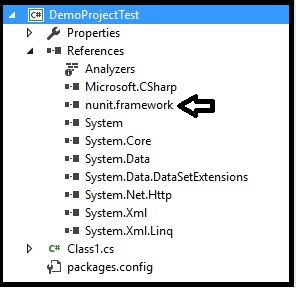
The NUnit Test Adapter allows you to run NUnit tests inside Visual Studio.

**Note :** If you do not add this component in your project you will not be able to find your tests in test Explorer.

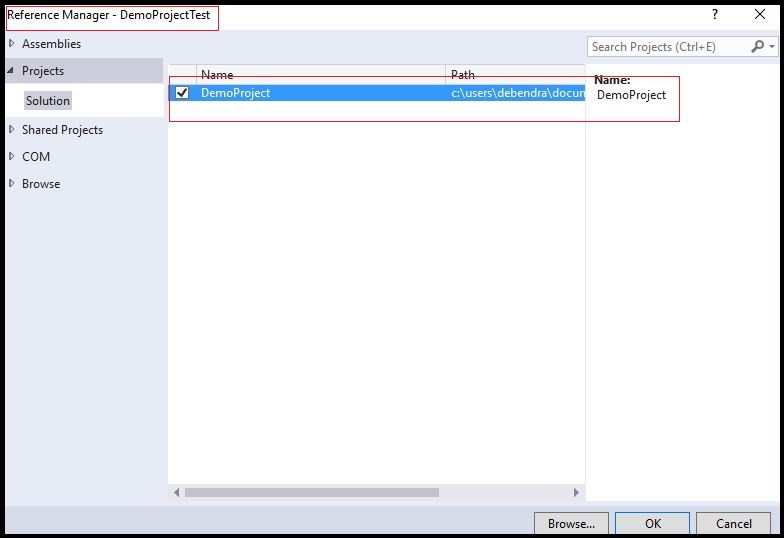


**Note**

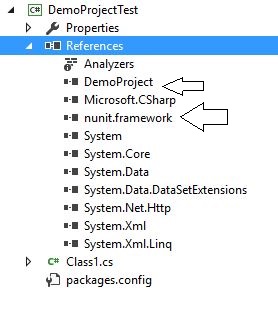
You need to install both the Libraries in the project where you are writing the test methods only.



Now our next job is to add the Demo Project dll in DemoProjectTest References.



Now it will add the following dll to the references.



So here is the test case I  have written to test our requirements.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using NUnit.Framework;

using DemoProject;

namespace DemoProjectTest {

[TestFixture]

public class DemoTests {

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.Geneder);

}

}

[Test]

public void TestLogin() {

Program pobj = new Program();

string x = pobj.Login("Ajit", "1234");

string y = pobj.Login("", "");

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);

}

[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");

}

} }

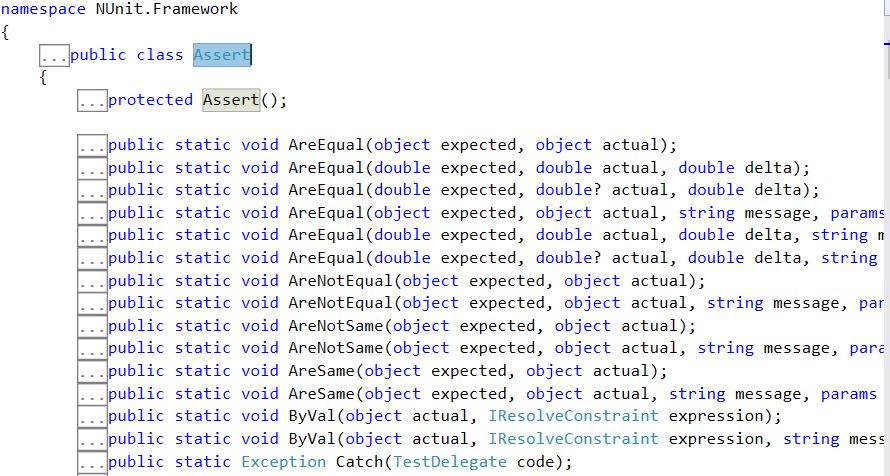
}

**TestFixture**

The class that is to be tested using Nunit should be decorated with TextFixture.

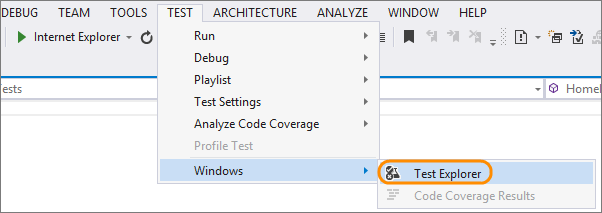
**Test**

This attribute identifies the method to be tested. If we do not write this attribute then we can't to identify the test in Testexplorer. We have an Assert class with the following methods for validating different conditions in the TestFixture.

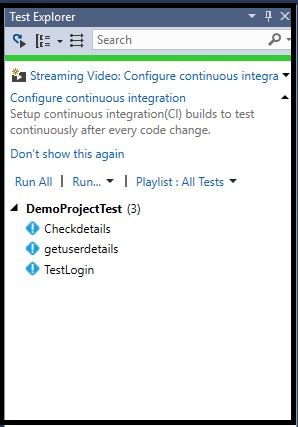


Each methods has different functions.

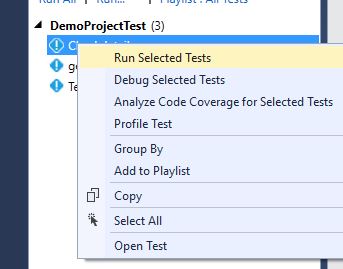
Now let's check our test. Go to the Visual Studio Test Explorer.



Now if we check the Left side of Visual Studio it will open a pane called Test Explorer with all the test scenarios as follows.

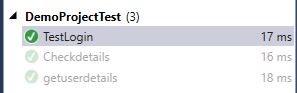


Now from here if we run a test individually just right click on it and run the test



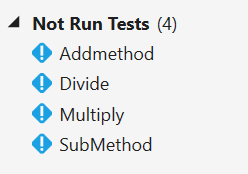
So in this way we can run the tests.

If the test Runs successfully it will show the following tick mark.

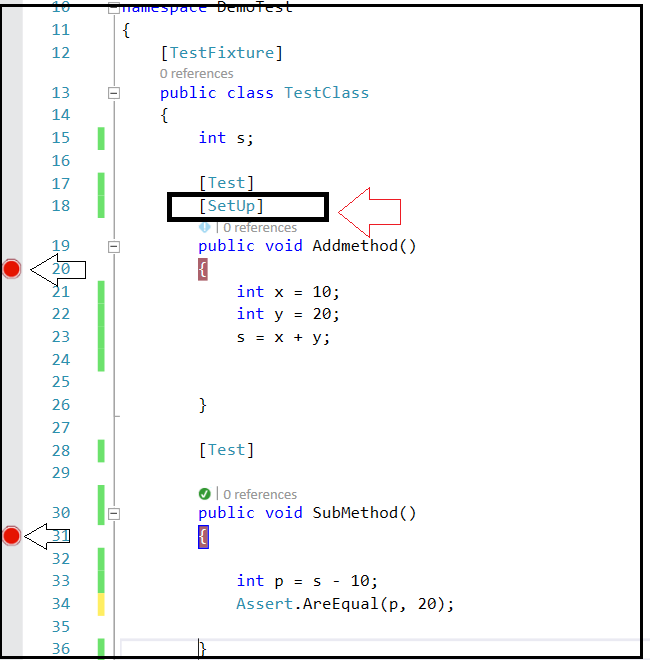


Beside these attribute we have some more attributes like,

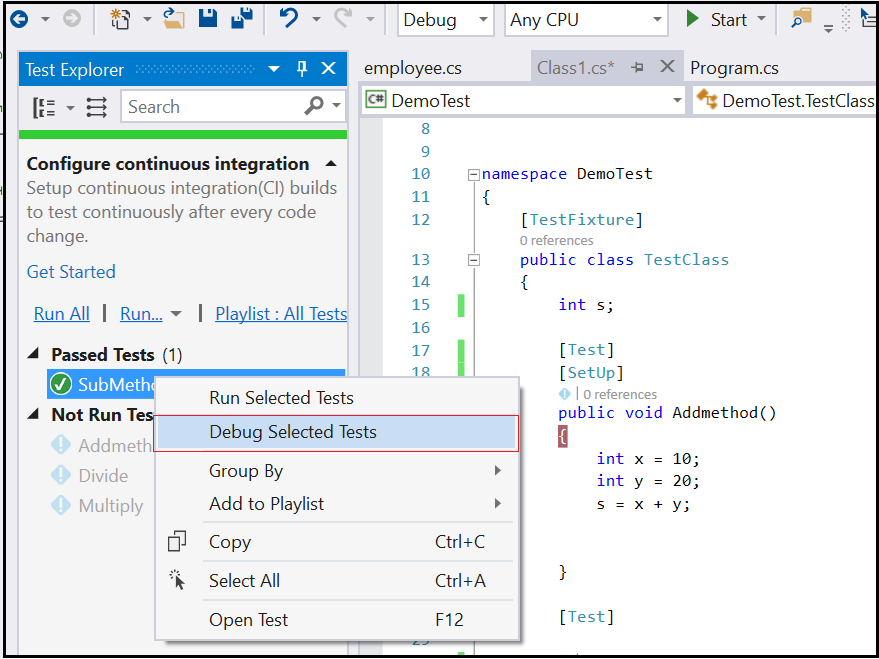
1. **SetUp**  
   This attribute is used when you want to execute a piece of code in each test case. It identifies a method to be executed each time before a TestMethod/Test is executed. Let's check this with an example. Here I have created a new project for testing. Here I have created four test methods.



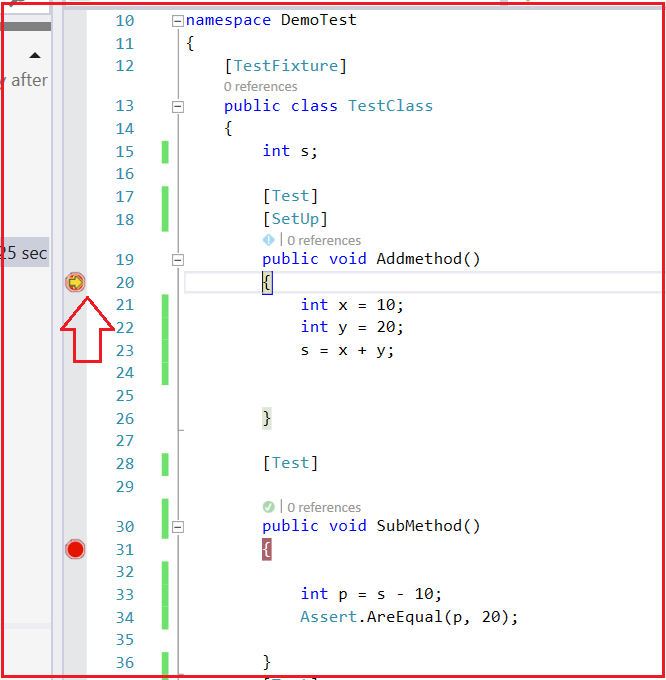
Now let's check the methods.



So here I have the methods where we have used **setUp** attribute. Let's try to debug and check **SubMethod**. For debugging just Right Click and Debug selected Test.

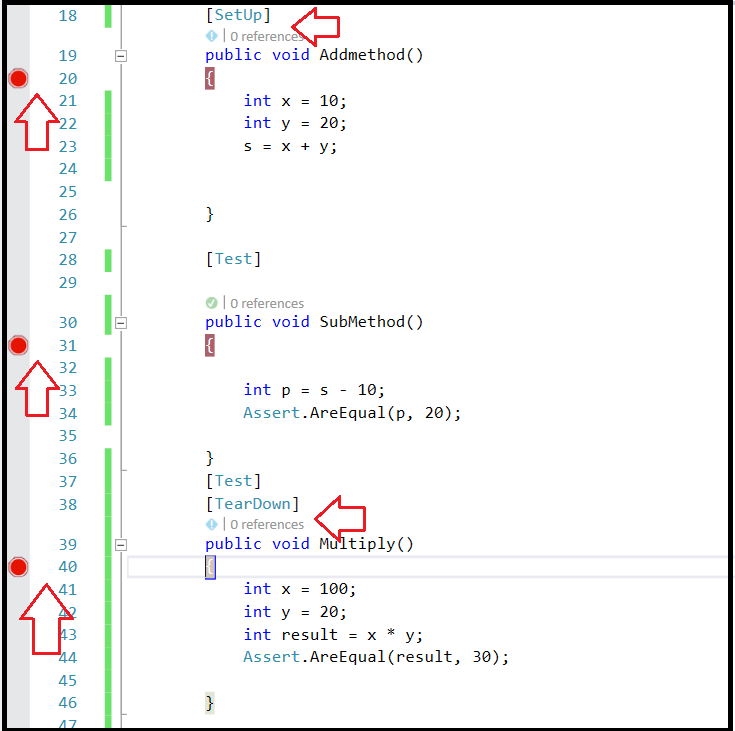


Now if we notice the sequence it will go to the **Addmethod** first like these.

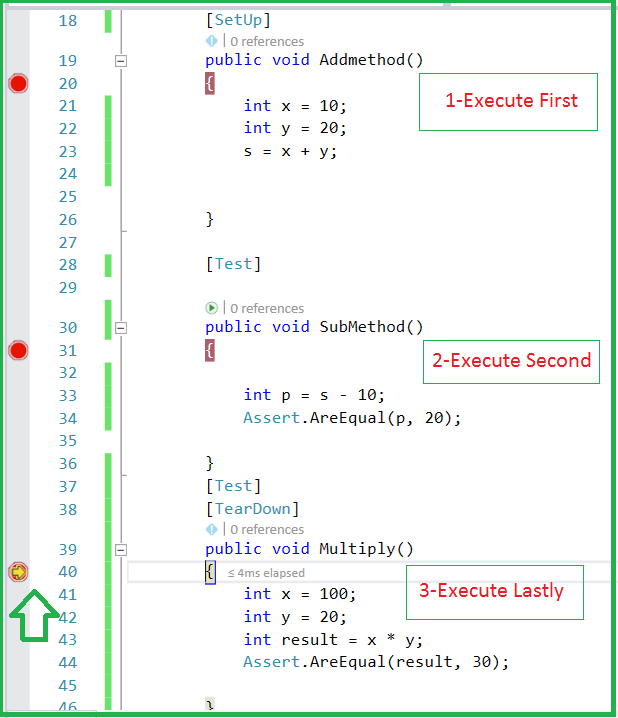


**Thus the method where the setup attribute is written will be executed first before any test method.**

1. **TearDown**  
   After completely executing each test if you want to execute a piece of code then you have to write this code under **TearDown** attribute. So let's check this again. So we have set breakpoints to check the sequence of execution.



If we check the sequence of execution the method that used TearDown attribute will be executed last after execution of the Test case.



**Note**

If two SetUp classes are there the class that was written first will be executed first and then after that, the class that is written next to it will execute. (Top to bottom approach).

Similarly, in tear down the order is reversed -- it will follow bottom to top approach.

So in this way, we can test our requirements using Nunit Testing.