2.10 Memory Testing with dotMemoryUnit



This section will guide you to:

* Create a Windows Class Library Project for running all tests in the test explorer

**Development Environment**

* Windows 10
* Visual Studio 2019 Community Version

This guide has ten subsections, namely:

2.10.1 Installing dotCover from ReSharper

2.10.2 Creating a Windows class library project for creating target classes to test

2.10.3 Creating a Windows class library project for running NUnit tests

2.10.4 Setting up NUnit as part of a Visual Studio project

2.10.5 Setting up NUnit3TestAdapter as part of a Visual Studio project

2.10.6 Setting up Moq as part of a Visual Studio project

2.10.7 Adding Test Fixture for performing various tests

2.10.8 Building the project

2.10.9 Running all the tests in Test Explorer

2.10.10 Pushing the code to your GitHub repositories

**Step 2.10.1:** Installing dotCover from ReSharper

* Go to <https://www.jetbrains.com/dotcover/download/#section=web-installer>
* Click on **Download.** This will download the web file for installing dotNet

**Step 2.10.2:** Creating a Windows class library project for creating target classes to test

* Open Visual Studio.
* From the top menu, click **File->New->Project**
* Select **(Class Library (.NET Framework)** from the displayed project types
* Click **Next**
* Name the **Project Name** as Phase4Section2.5 and click **Create**

**Step 2.10.3:** Creating a Windows class library project for running NUnit tests

* In **Solution Explorer,** right click the Solution item and click **Add->New Project**
* Select **(Class Library (.NET Framework)** from the displayed project types
* Click **Next**
* Name the **Project Name** as Phase4Section2.5.Tests and click **Create**

**Step 2.10.4:** Setting up NUnit as part of the project

* From the **Solution Explorer,** right click **Phase4Section2.5.Tests** and click **Manage Nuget Packages**
* Click on **Browse** tab and search for NUnit
* Click on the NUnit item and click **Install**

**Step 2.10.5:** Setting up NUnit3TestAdapter as part of the project

* From the **Solution Explorer,** right click on **Phase4Section2.3** and click **Manage Nuget Packages**
* Click on **Browse** tab and search for NUnit3TestAdapter
* Click on the NUnit3TestAdapter item and click **Install**

**Step 2.10.6:** Setting up Moq as part of the project

* From the **Solution Explorer,** right click **Phase4Section2.5.Tests** and click **Manage Nuget Packages**
* Click on **Browse** tab and search for Moq
* Click on the Moq item and click **Install**

**Step 2.10.7:** Adding Test Fixture for performing various tests

* From the **Solution Explorer,** expand **Phase4Section2.5.Tests** and double click **Class1.cs**
* Add the following code:

**using** System;

**using** System.Collections.Generic;

**using** System.IO;

**using** System.Linq;

**using** System.Text;

**using** System.Threading.Tasks;

**using** Moq;

**using** NUnit.Framework;

**namespace** Phase4Section2.\_5.Tests

{

[TestFixture]

**public** **class** Class1

{

[Test]

**public** **void** BasicAssertions()

{

**int** total = 100, marks1 = 60, marks2 = 75;

**string** name = **null**;

Assert.That(marks1, Is.Not.EqualTo(marks2));

Assert.That(marks1, Is.LessThan(marks2));

Assert.That(marks2, Is.InRange(50, 75));

Assert.That(name, Is.Null);

}

[Test]

**public** **void** Warnings()

{

**int** total = 100, marks1 = 60, marks2 = 75;

**string** name = **null**;

Warn.If(marks1 > 100);

Warn.If(name == **null**);

Warn.Unless(marks1 + marks2 < 200);

Assert.Warn("This is a warning message");

}

[Test]

**public** **void** ArrangeActAssert()

{

**var** calc = **new** Calculator();

**var** answer = calc.add(5, 19);

Assert.That(answer, Is.EqualTo(24));

}

[Test]

**public** **void** MultipleAssertions()

{

**int** total = 100, marks1 = 60, marks2 = 75;

**string** name = **null**;

Assert.Multiple(() =>

{

Assert.That(marks1, Is.Not.EqualTo(marks2));

Assert.That(marks1, Is.LessThan(marks2));

Assert.That(marks2, Is.InRange(50, 75));

});

}

[Test]

**public** **void** Exceptions()

{

**var** calc = **new** Calculator();

Assert.Throws<InvalidOperationException>(() => calc.addStrings("aaa", "Bbb"));

}

}

}

* From the **Solution Explorer,** expand **Phase4Section2.5.Tests** and right click **Add->New Class** name it as **Calculator.cs**
* Add the following code in **Calculator.cs**:

**using** System;

**using** System.Collections.Generic;

**using** System.Linq;

**using** System.Text;

**using** System.Threading.Tasks;

**namespace** Phase4Section2.\_5

{

**public** **interface** ICalculator

{

**int** add(**int** x, **int** y);

**int** addStrings(**string** x, **string** y);

}

**public** **class** Calculator:ICalculator

{

**public** **int** add(**int** x, **int** y)

{

**return** x + y;

}

**public** **int** addStrings(**string** x, **string** y)

{

**int** a = 0, b = 0;

Int32.TryParse(x, **out** a);

Int32.TryParse(y, **out** b);

**if** (a == 0 || b == 0)

**throw** **new** InvalidOperationException("String values are not valid integers");

**return** a + b;

}

}

**Step 2.10.8:** Building the project

* From the top menu, choose **Build->Build Solution**
* If any compile errors are shown, fix them as required

**Step 2.10.9:** Running all the tests in Test Explorer

* From the top menu, choose **Test->Windows->Test Explorer**
* In Test Explorer, click on **Run All**
* This will execute the tests and show the results in Test Explorer

**Step 2.10.10:** Pushing the code to your GitHub repositories

Open your command prompt and navigate to the folder where you have created your files.

cd <folder path>

Initialize your repository using the following command:

git init

Add all the files to your git repository using the following command:

git add .

Commit the changes using the following command:

git commit -m “Changes have been committed.”

Push the files to the folder you created initially using the following command:

git push -u origin master