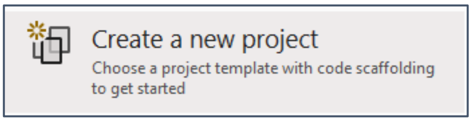
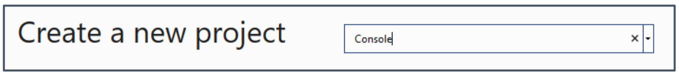
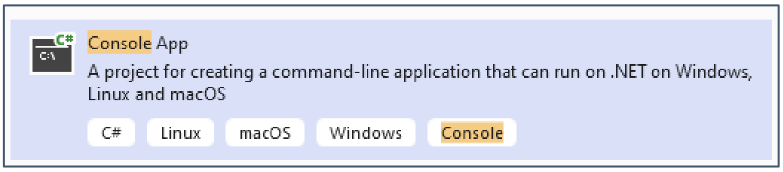
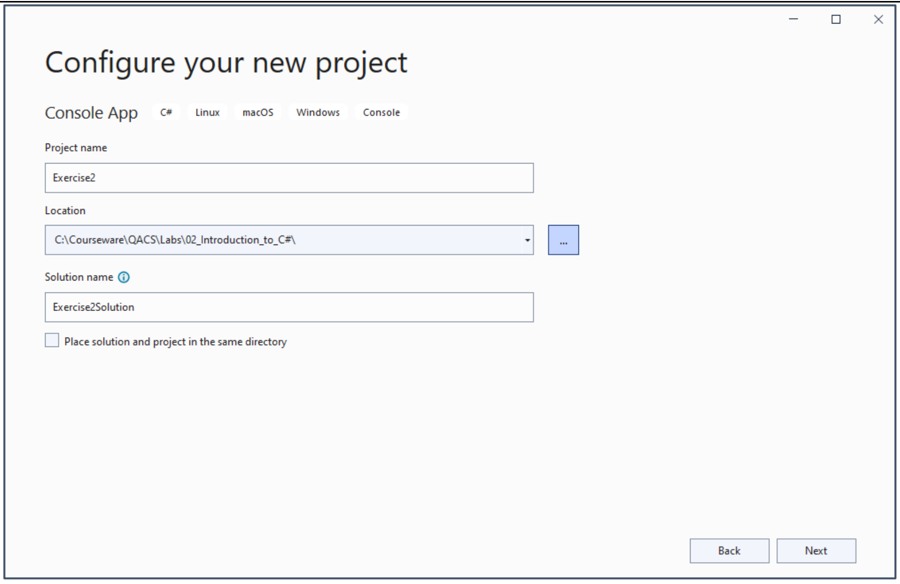
Introduction to C#

The objective of this exercise is to get you started using C# and the Visual Studio IDE and introduce you to simple debugging. You will also work with a test project and write some simple tests.

1	Start **Microsoft Visual Studio 2022**
2	Choose '**Create a new project'**  Create a new project Choose a project template with code scaffolding to get started
3	In the Search box, type **Console**  Create a new project Console
4	Select **Console App** and choose **Next**  Console App A project for creating a command-line application that can run on .NET on Windows, Linux and macOS C# Linux macOS Windows Console
5	Name the *Project* **Exercise2** and the *Solution* **Exercise2Solution**. Save the files in **C:\Courseware\QACS\Labs\02_Introduction_to_C#**



Configure your new project

Console App C# Linux macOS Windows Console

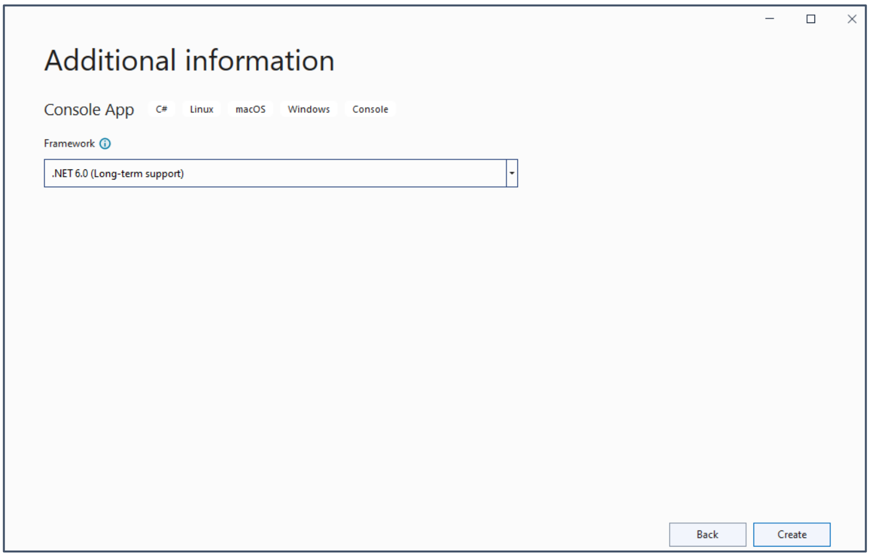
Project name: Exercise2

Location: C:\Courseware\QACS\Labs\02_Introduction_to_C#\

Solution name: Exercise2Solution

□ Place solution and project in the same directory

6 Ensure **.NET 6.0 (Long-term support)** is selected in Additional information and click **Create**

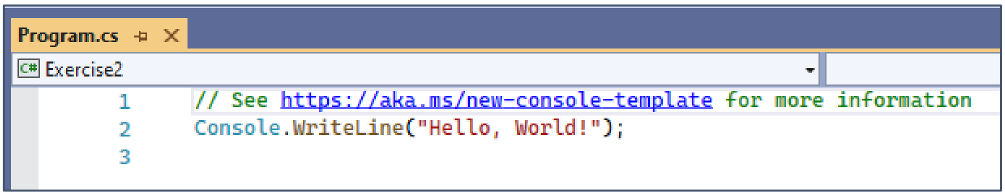


Additional information

Console App C# Linux macOS Windows Console

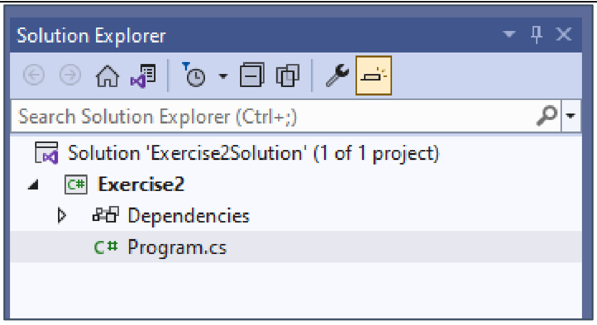
Framework: .NET 6.0 (Long-term support)

7 You will see a code editor window for a file called **Program.cs** containing the following code:

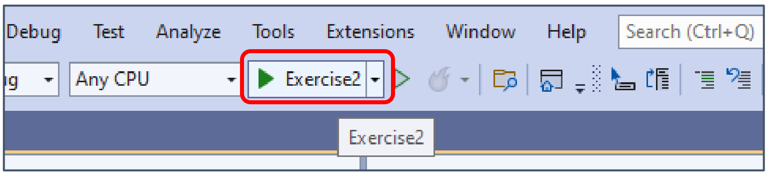


```
// See https://aka.ms/new-console-template for more information
Console.WriteLine("Hello, World!");

```

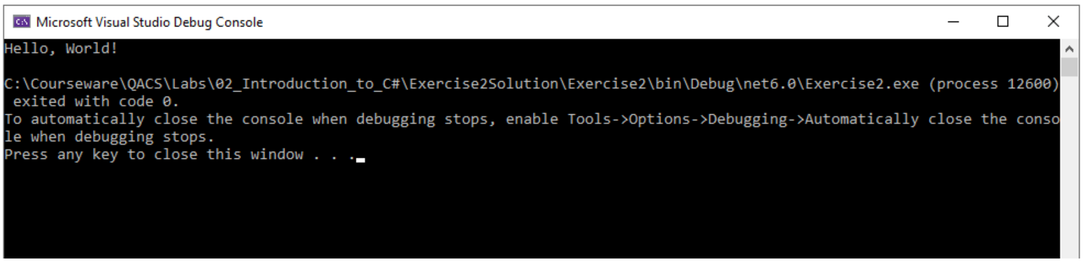
8 You can see the Solution Explorer window with **Program.cs** being tracked as the active file.


Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'Exercise2Solution' (1 of 1 project)
Exercise2
Dependencies
Program.cs

9 From the toolbar, click the green arrow to **Start with Debugging**:



10 Observe the output of the program and press any key to close the console window.

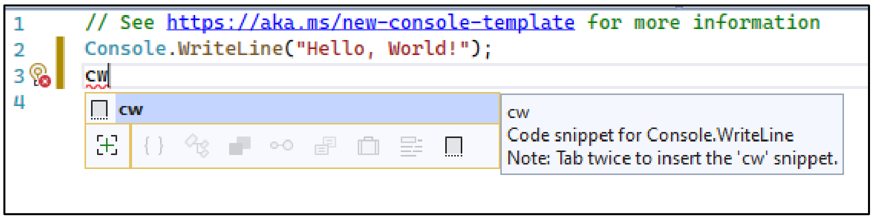


```
Hello, World!

C:\Courseware\QACS\Labs\02_Introduction_to_C#\Exercise2Solution\Exercise2\bin\Debug\net6.0\Exercise2.exe (process 12600)
 exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . ._
```

11 Add a line of code at line 3 as follows. Type:

cw

Notice it is a recognised code snippet for **Console.WriteLine**:

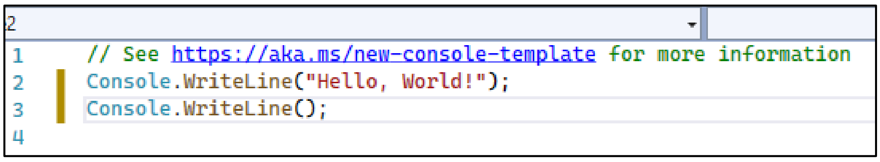


```
1  // See https://aka.ms/new-console-template for more information
2  Console.WriteLine("Hello, World!");
3  cw
4
```

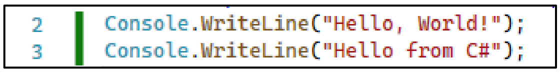
cw

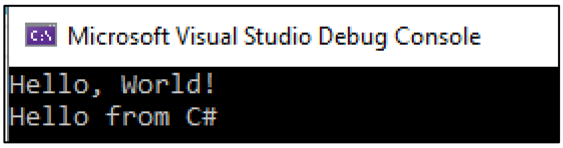
Code snippet for Console.WriteLine
Note: Tab twice to insert the 'cw' snippet.

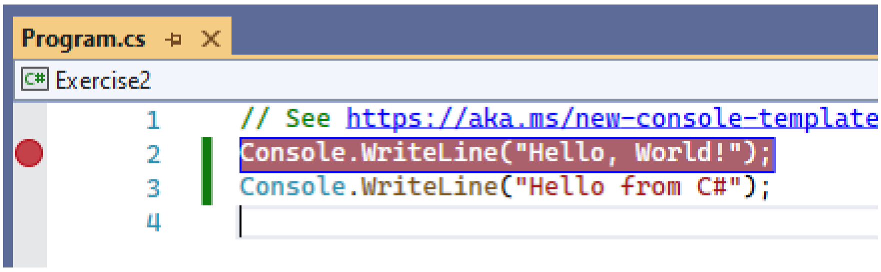
Tab twice to insert the code:

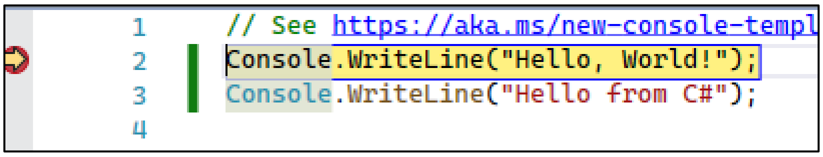


```
1  // See https://aka.ms/new-console-template for more information
2  Console.WriteLine("Hello, World!");
3  Console.WriteLine();
4
```

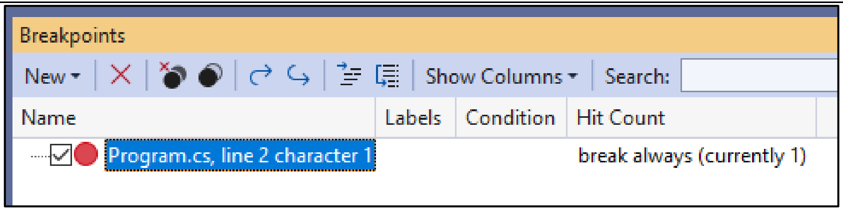
12	Add the text string **"Hello from C#"** as a parameter to the WriteLine method.
	
	`2 Console.WriteLine("Hello, World!");`
	`3 Console.WriteLine("Hello from C#");`

13	Run the program again, this time using the keyboard shortcut **F5**
	
	
	`Hello, World!`
	`Hello from C#`
	Press any key to quit the program.

14	You will now do some very simple debugging.
	Click in the margin well to the left of **line 2** to set a breakpoint:
	
	
	`1 // See https://aka.ms/new-console-template`
	`2 Console.WriteLine("Hello, World!");`
	`3 Console.WriteLine("Hello from C#");`
	`4`

15	Debug the program with **F5** and notice how the application is now paused on your breakpoint:
	
	`1 // See https://aka.ms/new-console-templ`
	`2 Console.WriteLine("Hello, World!");`
	`3 Console.WriteLine("Hello from C#");`
	`4`
	Observe the changes to the Visual Studio layout.
	Numerous debug windows are now open including Autos, Locals, Watch 1, Call Stack, and Breakpoints.

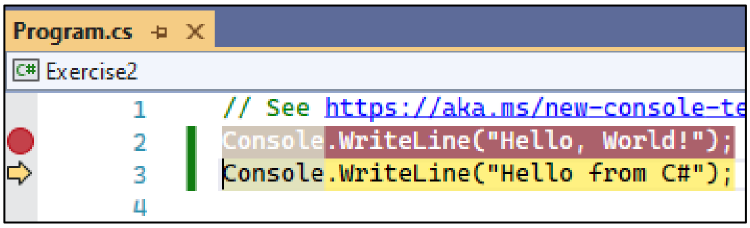
16	Open the **Breakpoints** window and observe the **Hit Count** value is set to **"break always (currently 1)"**



Name	Labels	Condition	Hit Count
☑ ● Program.cs, line 2 character 1			break always (currently 1)

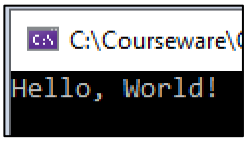
17 You will Step Into the next line of code which will run the line that is currently highlighted.

Press **F11**.



```
1   // See https://aka.ms/new-console-te
2   Console.WriteLine("Hello, World!");
3   Console.WriteLine("Hello from C#");
4
```

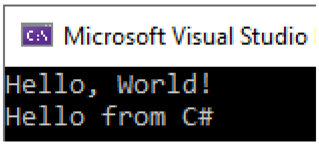
Look at the **Console** output window. You can see line 2 has run:



```
Hello, World!
```

Press **F11** again.

The program ends because the last line of code has been run successfully. You can see the complete result in the output console:

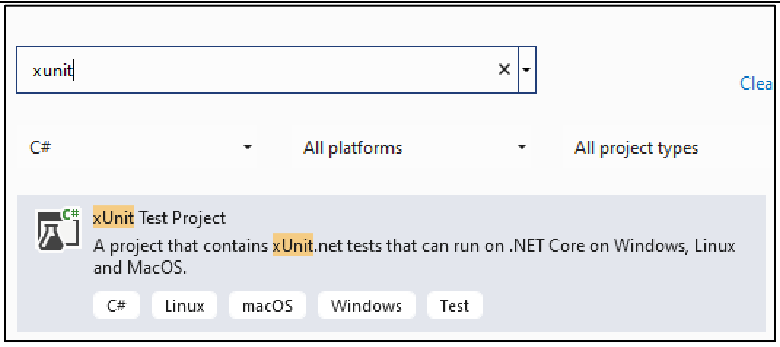


```
Hello, World!
Hello from C#
```

18 Some of the exercises on the course use tests to validate code behaviour so you will now add a test project to your solution.

Right-click the **Exercise2Solution** and choose **Add -> New Project**

In the search box type **xunit**.



xunit

C# · All platforms · All project types

xUnit Test Project

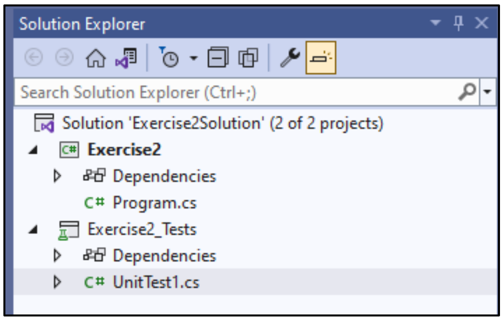
A project that contains xUnit.net tests that can run on .NET Core on Windows, Linux and MacOS.

Select **xUnit Test Project** and click **Next**.

Name the project **Exercise2_Tests** and click **Next**.

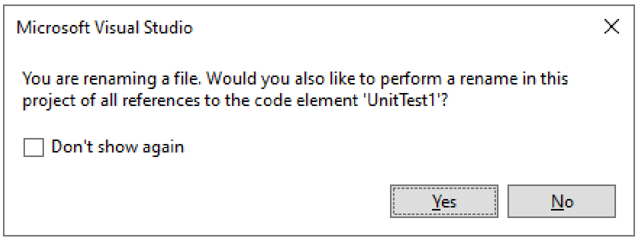
Ensure **.NET 6.0 (Long-term support)** is selected and click **Create**.

19 Your **Solution Explorer** window now contains one solution with two projects:



```
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'Exercise2Solution' (2 of 2 projects)
◢ Exercise2
  ▷ Dependencies
    Program.cs
◢ Exercise2_Tests
  ▷ Dependencies
  ▷ UnitTest1.cs
```

20 Rename **UnitTest1.cs** (by right-clicking on the file name) to **SimpleTests.cs** and select **YES** when the following prompt displays:

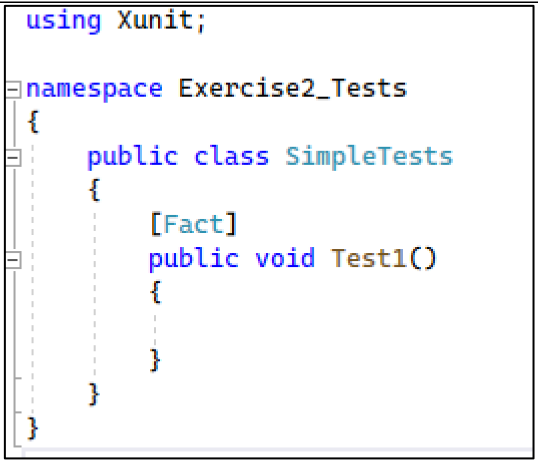


Microsoft Visual Studio

You are renaming a file. Would you also like to perform a rename in this project of all references to the code element 'UnitTest1'?

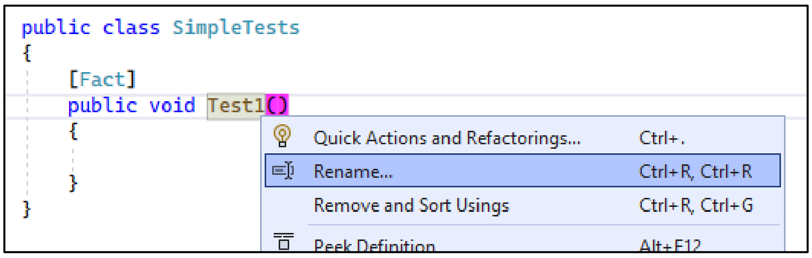
□ Don't show again

21 Your **SimpleTests.cs** file contains the following starter code:



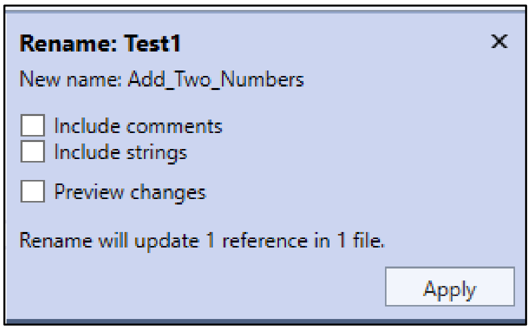
```
using Xunit;

namespace Exercise2_Tests
{
    public class SimpleTests
    {
        [Fact]
        public void Test1()
        {

        }
    }
}
```

22 Rename **Test1** to **Add_Two_Numbers**:



```
public class SimpleTests
{
    [Fact]
    public void Test1()
    {

    }
}
```

Quick Actions and Refactorings... Ctrl+.
Rename... Ctrl+R, Ctrl+R
Remove and Sort Usings Ctrl+R, Ctrl+G
Peek Definition Alt+F12



Rename: Test1 ✕

New name: Add_Two_Numbers

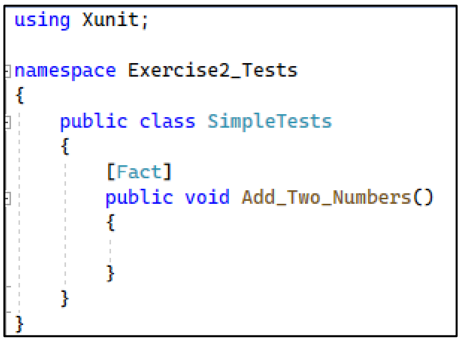
□ Include comments
□ Include strings

□ Preview changes

Rename will update 1 reference in 1 file.

Apply

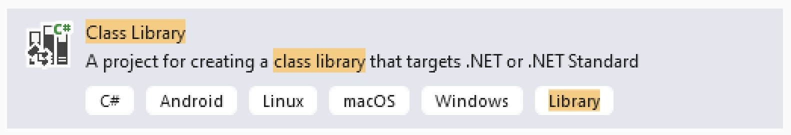
Click **Apply**.



```
using Xunit;

namespace Exercise2_Tests
{
    public class SimpleTests
    {
        [Fact]
        public void Add_Two_Numbers()
        {

        }
    }
}
```

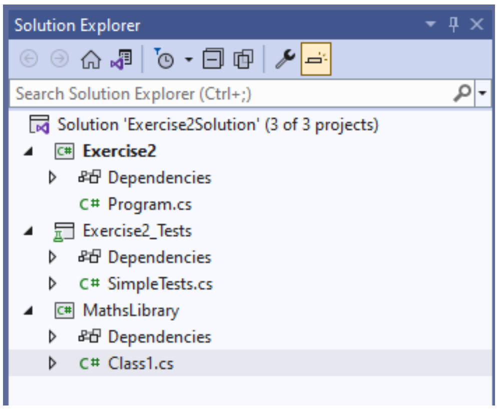
23 You are going to write some simple tests for a Calculator. This calculator is going to be created in a new project of type Class Library.

Add a new **class library** project to the solution called **MathsLibrary**.



Class Library
A project for creating a class library that targets .NET or .NET Standard

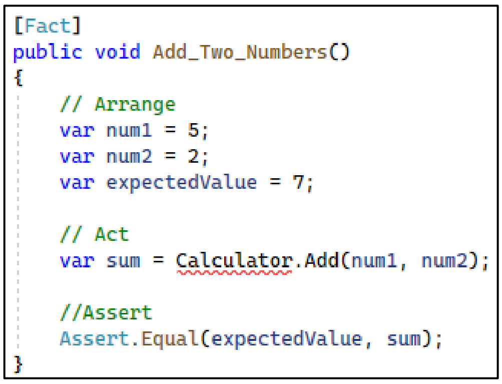
Solution Explorer should now look as follows:



```
Solution 'Exercise2Solution' (3 of 3 projects)
  Exercise2
    Dependencies
    Program.cs
  Exercise2_Tests
    Dependencies
    SimpleTests.cs
  MathsLibrary
    Dependencies
    Class1.cs
```

24 Delete the file **Class1.cs**.

25 In **SimpleTests.cs** add the following code to **Add_Two_Numbers**:



```
[Fact]
public void Add_Two_Numbers()
{
    // Arrange
    var num1 = 5;
    var num2 = 2;
    var expectedValue = 7;

    // Act
    var sum = Calculator.Add(num1, num2);

    //Assert
    Assert.Equal(expectedValue, sum);
}
```

This test code uses the standard testing pattern called the triple A pattern: *Arrange, Act, Assert.*

Arrange is for setting up items you need for the test.

Act is for carrying out the action you are testing.

Assert is for confirming the acted upon code behaves as expected.

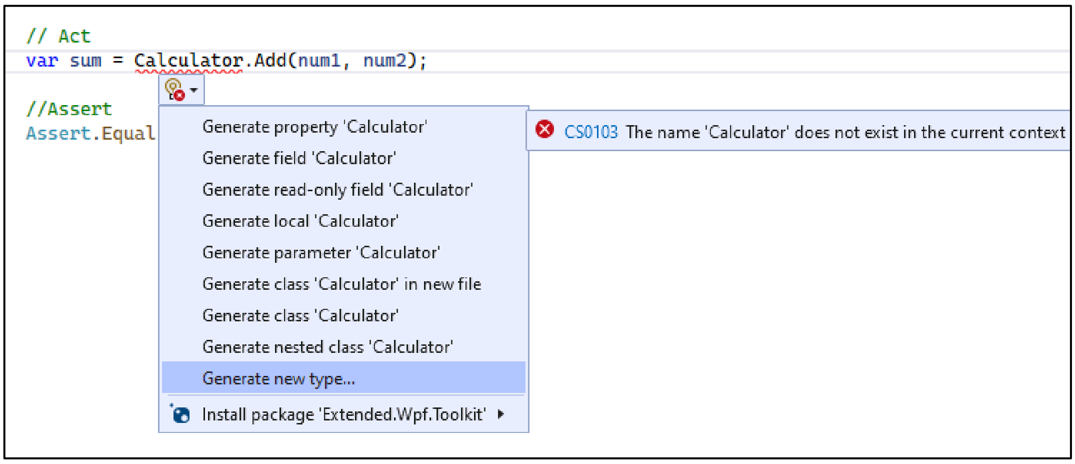
26 The arrange phase creates three variables: **num1**, **num2**, and **expectedValue**.

The act phase calls an **Add** method on a **Calculator**, passing in **num1** and **num2** as parameters and assigning the result to a variable called **sum**.

The assert phase checks whether the **expectedValue** and the **sum** values are equal.

The Calculator type does not exist so you will use Visual Studio to help you create it.

Press **Ctrl+.** (Ctrl+dot) on **Calculator** to see the available options:



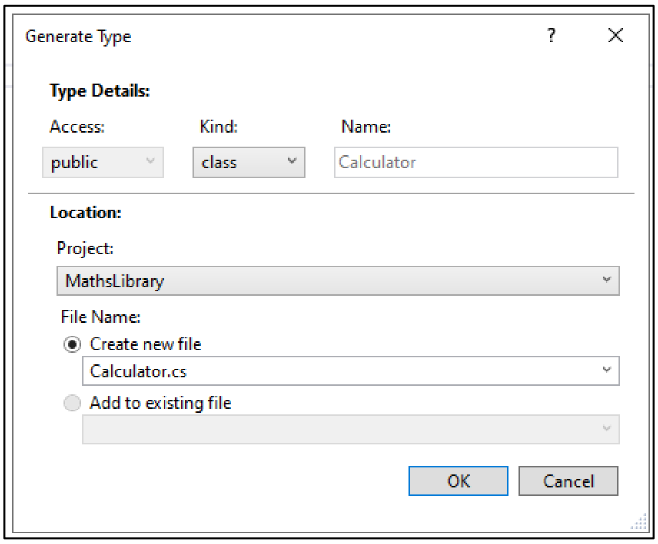
```
// Act
var sum = Calculator.Add(num1, num2);

//Assert
Assert.Equal
```

CS0103 The name 'Calculator' does not exist in the current context

Generate property 'Calculator'
Generate field 'Calculator'
Generate read-only field 'Calculator'
Generate local 'Calculator'
Generate parameter 'Calculator'
Generate class 'Calculator' in new file
Generate class 'Calculator'
Generate nested class 'Calculator'
Generate new type...
Install package 'Extended.Wpf.Toolkit'

You want Calculator to be created in your **MathsLibrary** project rather than locally within the Test project so choose '**Generate new type...**'

27 In the dialog box, ensure a **public class** will be created and change the project to **MathsLibrary** and **Create new file**:



Generate Type

Type Details:

Access: public

Kind: class

Name: Calculator

Location:

Project: MathsLibrary

File Name:

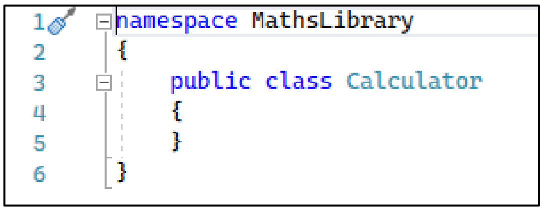
● Create new file

Calculator.cs

○ Add to existing file

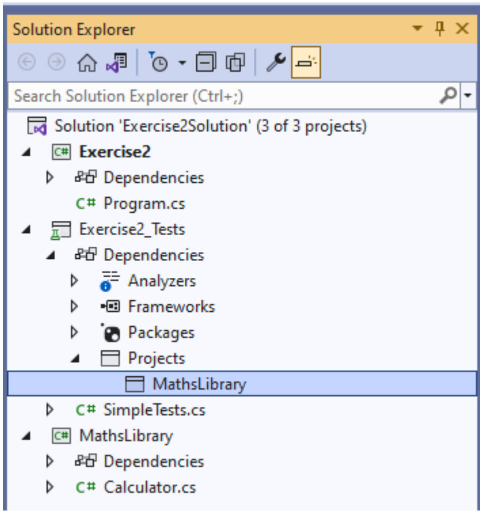
Click **OK**.

28 Visual Studio has created a new class (a kind of type) called **Calculator** in **MathsLibrary**:



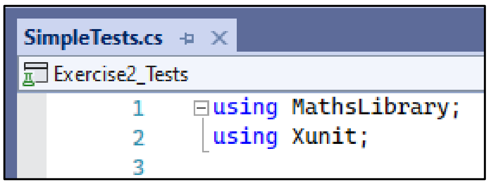
```
namespace MathsLibrary
{
    public class Calculator
    {
    }
}
```

Visual Studio has also added a *reference* to the **MathsLibrary** project:



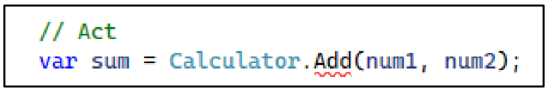
```
Solution 'Exercise2Solution' (3 of 3 projects)
◢ Exercise2
    ▷ Dependencies
      Program.cs
◢ Exercise2_Tests
    ◢ Dependencies
        ▷ Analyzers
        ▷ Frameworks
        ▷ Packages
        ◢ Projects
            MathsLibrary
    ▷ SimpleTests.cs
◢ MathsLibrary
    ▷ Dependencies
    ▷ Calculator.cs
```

It has also imported the **MathsLibrary** *namespace* into your test project:



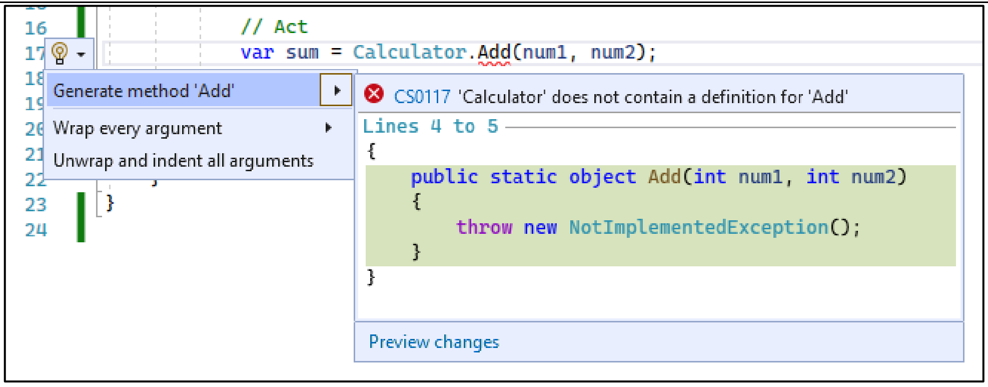
```
using MathsLibrary;
using Xunit;
```

29 The act phase of the test code now recognises the **Calculator** type but displays an error because **Calculator** does not contain a definition for **Add**:



```
// Act
var sum = Calculator.Add(num1, num2);
```

30 Use **Ctrl+dot** on **Add** to generate the method:

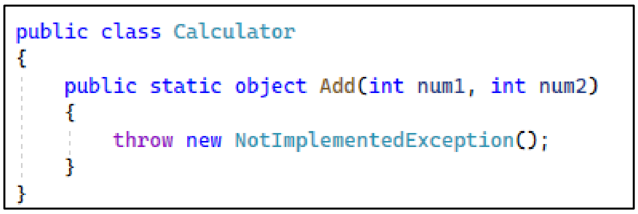


```
// Act
var sum = Calculator.Add(num1, num2);
```

CS0117 'Calculator' does not contain a definition for 'Add'

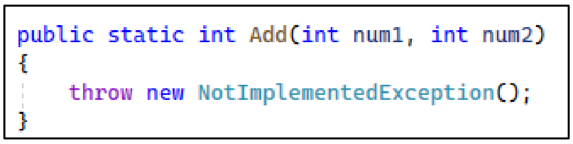
```
Lines 4 to 5
{
    public static object Add(int num1, int num2)
    {
        throw new NotImplementedException();
    }
}
```

31 **Calculator.cs** now contains an **Add** method:



```
public class Calculator
{
    public static object Add(int num1, int num2)
    {
        throw new NotImplementedException();
    }
}
```

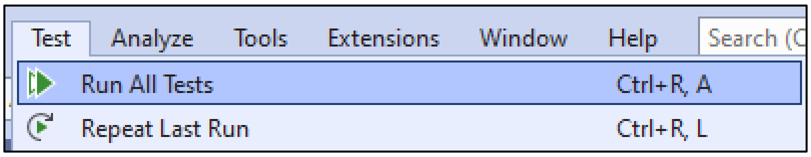
32 You want your **Add** method to return whole numbers so change the word **object** to **int**.



```
public static int Add(int num1, int num2)
{
    throw new NotImplementedException();
}
```

33 You will run the test and observe the outcome.

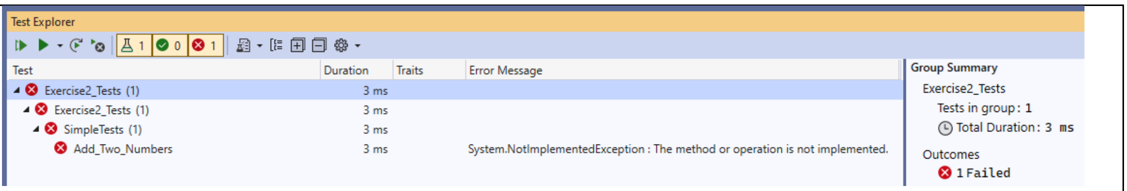
Test -> Run All Tests.



34 Ensure the Test Explorer window is visible:

Test -> Test Explorer.

Expand the Test until you see the **Add_Two_Numbers** failed test (it appears in red) alongside the error message: **The method operation is not implemented.**



Test Explorer

Test | Duration | Traits | Error Message

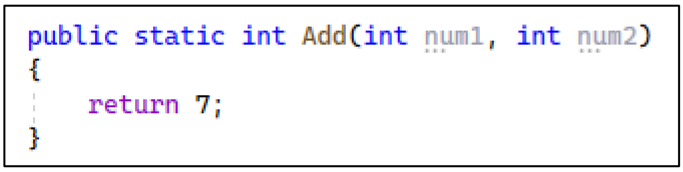
Exercise2_Tests (1) 3 ms
Exercise2_Tests (1) 3 ms
SimpleTests (1) 3 ms
Add_Two_Numbers 3 ms System.NotImplementedException : The method or operation is not implemented.

Group Summary
Exercise2_Tests
Tests in group: 1
Total Duration: 3 ms
Outcomes
1 Failed

35 You will edit the **Add** method code to ensure the method is implemented and confirm that the test passes.

Delete the line of code: **throw new NotImplementedException;**

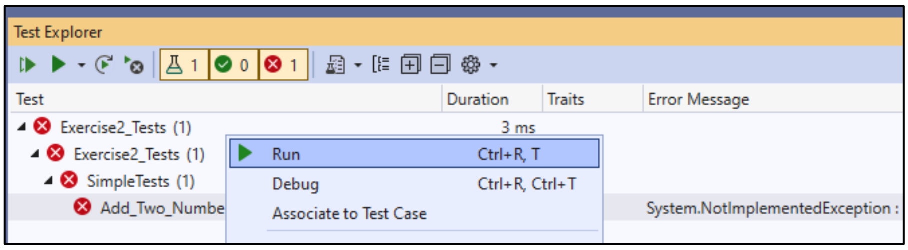
Replace the code with: **return 7;**



```
public static int Add(int num1, int num2)
{
    return 7;
}
```

This is hard-coding the expected value, which allows you to confirm the test is working correctly.

36 Right-click the failed test in **Test Explorer** and select **Run**.

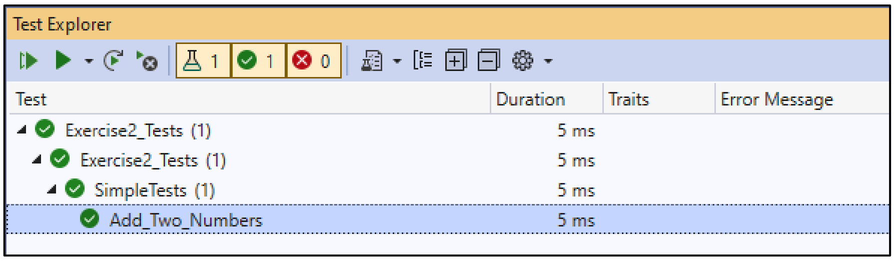


Test Explorer

Test | Duration | Traits | Error Message

Exercise2_Tests (1) 3 ms
Exercise2_Tests (1)
SimpleTests (1)
Add_Two_Numbe System.NotImplementedException :

The test should now pass:



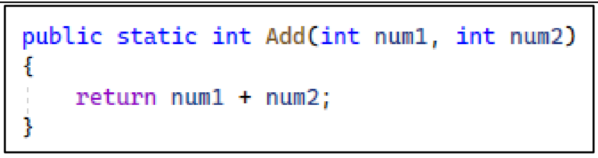
Test Explorer

Test | Duration | Traits | Error Message

Exercise2_Tests (1) 5 ms
Exercise2_Tests (1) 5 ms
SimpleTests (1) 5 ms
Add_Two_Numbers 5 ms

37 The final step is to refactor the code within the method to perform the calculation so that additional tests adding different integers will also pass.

Edit the **Add** method as follows:

<table>
<tr><th></th><th>
<pre>
public static int Add(int num1, int num2)
{
    return num1 + num2;
}
</pre></th></tr>
<tr><td>38</td><td>Re-run the test to ensure it continues to pass.

The process that you just followed is called Test-Driven Development (TDD). It follows a three-stage approach referred to as <i>red-green-refactor,</i> whereby you write a test before implementing the code. You ensure the test fails. This is the red stage. This is to guard against any false positives. You then write enough implementation to get the test to pass. This is the green stage. You then refactor the code to improve the implementation, ensuring the tests still pass.</td></tr>
<tr><td>39</td><td>If you have time, write a test for a <b>Subtract</b> method, then use Visual Studio to help build the implementation. Use <b>Test Explorer</b> to run your tests.</td></tr>
<tr><td>40</td><td>Solutions are provided in the <b>End</b> folder for your reference.</td></tr>
</table>