# 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’** |
| 3 | In the Search box, type **Console** |
| 4 | Select **Console App**and choose **Next** |
| 5 | Name the *Project* **Exercise2** and the *Solution* **Exercise2Solution**. Save the files in {**installFolder}\Labs\02\_Introduction\_to\_C#** |
| 6 | Ensure **.NET 6.0 (Long-term support)** is selected in Additional information and click **Create** |
| 7 | You will see a code editor window for a file called **Program.cs** containing the following code: |
| 8 | You can see the Solution Explorer window with **Program.cs** being tracked as the active file. |
| 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. |
| 11 | Add a line of code at line 3 as follows. Type:  cw  Notice it is a recognised code snippet for **Console.WriteLine**:    **Tab twice** to insert the code: |
| 12 | Add the text string **“Hello from C#”** as a parameter to the WriteLine method. |
| 13 | Run the program again, this time using the keyboard shortcut **F5**    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: |
| 15 | Debug the program with **F5** and notice how the application is now paused on your breakpoint:    Observe thechanges 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)”** |
| 17 | You will Step Into the next line of code which will run the line that is currently highlighted.  Press **F11**.    Look at the **Console** output window. You can see line 2 has run:    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: |
| 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**.    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: |
| 20 | Rename **UnitTest1.cs** (by right-clicking on the file name) to **SimpleTests.cs** and select **YES** when the following prompt displays: |
| 21 | Your **SimpleTests.cs**file contains the following starter code: |
| 22 | Rename **Test1** to **Add\_Two\_Numbers**:      Click **Apply**. |
| 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**.    Solution Explorer should now look as follows: |
| 24 | Delete the file **Class1.cs**. |
| 25 | In **SimpleTests.cs** add the following code to **Add\_Two\_Numbers**: |
|  | 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:    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 **publicclass** will be created and change the project to **MathsLibrary** and **Create new file**:    Click **OK**. |
| 28 | Visual Studio has created a new class (a kind of type) called **Calculator** in **MathsLibrary**:    Visual Studio has also added a *reference* to the **MathsLibrary** project:    It has also imported the **MathsLibrary***namespace* into your test project: |
| 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**: |
| 30 | Use **Ctrl+dot** on **Add** to generate the method: |
| 31 | **Calculator.cs** now contains an **Add** method: |
| 32 | You want your **Add** method to return whole numbers so change the word **object** to **int**. |
| 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.** |
| 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;**    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**.    The test should now pass: |
| 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: |
| 38 | 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 *red-green-refactor,* 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. |
| 39 | If you have time, write a test for a **Subtract** method, then use Visual Studio to help build the implementation. Use **Test Explorer** to run your tests. |
| 40 | Solutions are provided in the **End** folder for your reference. |