

# C# for HAAS F1

## Exercise Workbook



## Lab 01: Introduction to Git and GitHub

Use these instructions to familiarize yourself with GitHub (if you have not used GitHub previously)

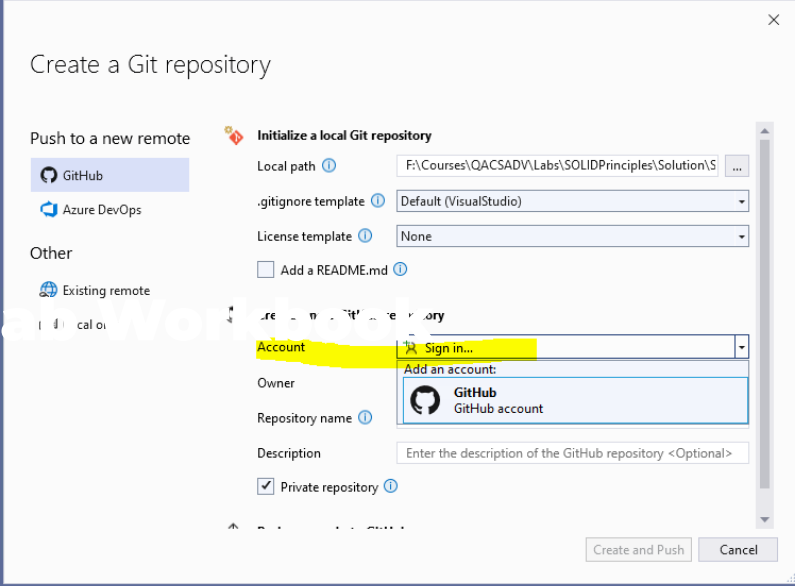
### Objective

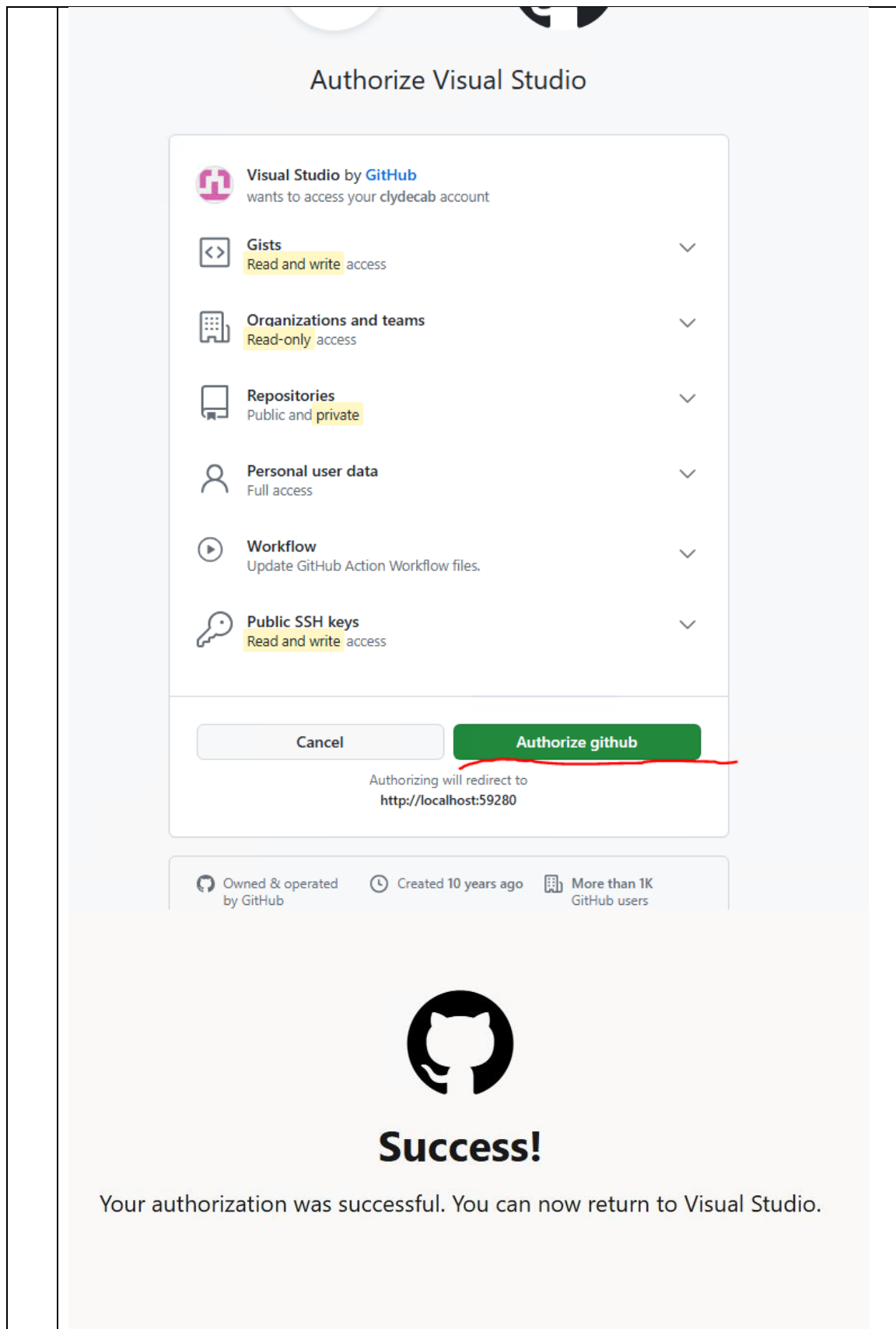
Gain an understanding of how GitHub can be used to work on projects in a collaborative environment. You are encouraged to use GitHub as a repository for **ALL** the code you work on as part of this course. Using it means you are less likely to be impacted on a virtual machine timing out at some point. This is especially true of LOD machines and less so if you are using GoToMyPC. However, **ALL** VM's will be reset at the course's conclusion so **backing things up is a really good idea**.

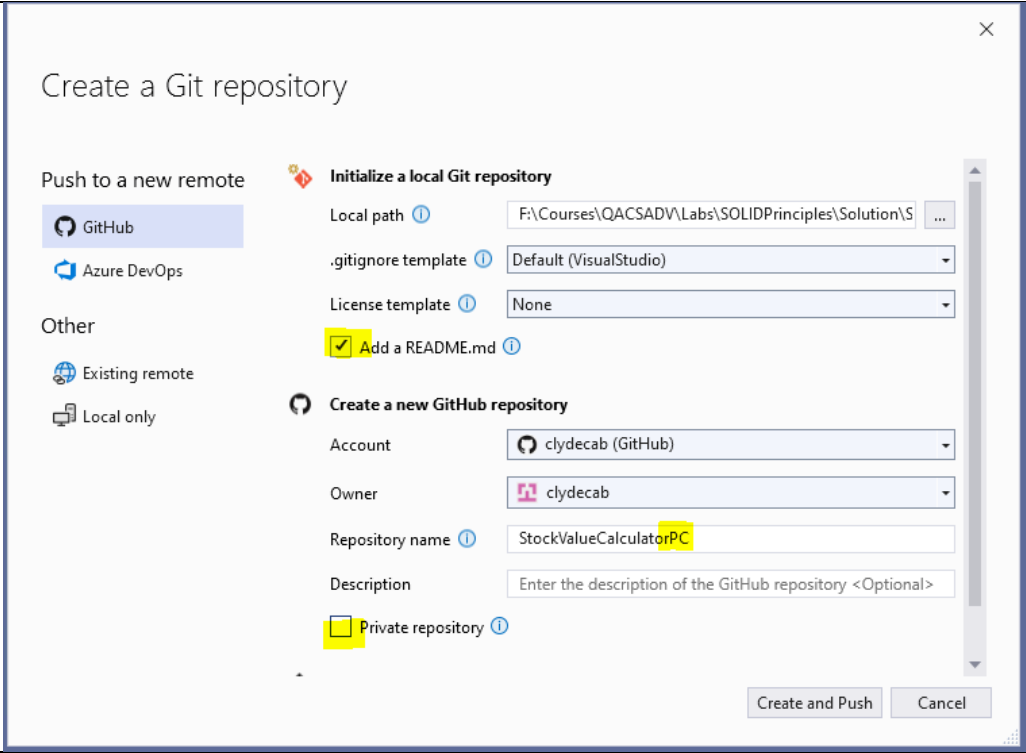
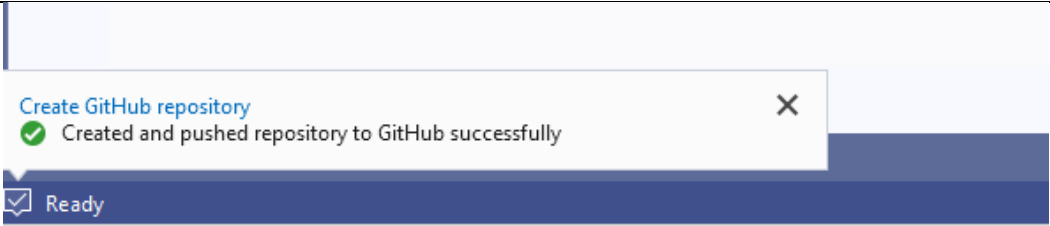
### Requirements

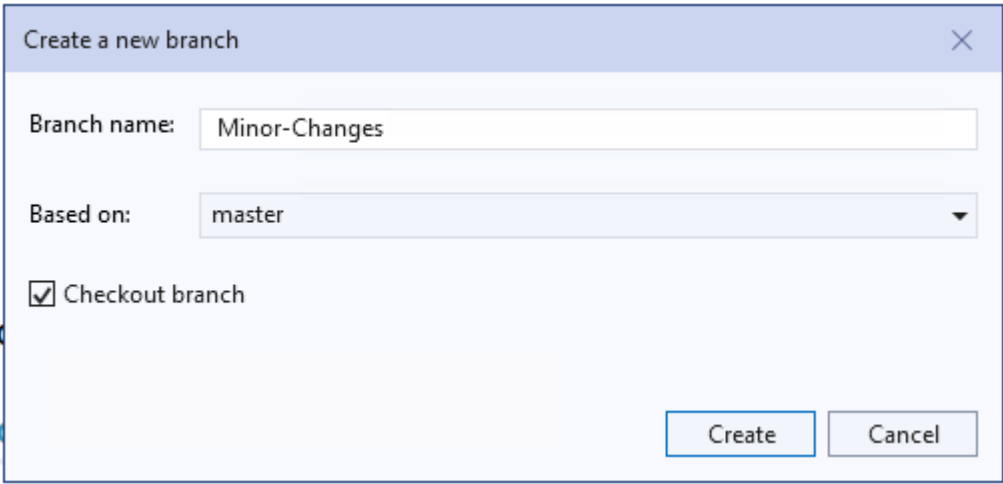
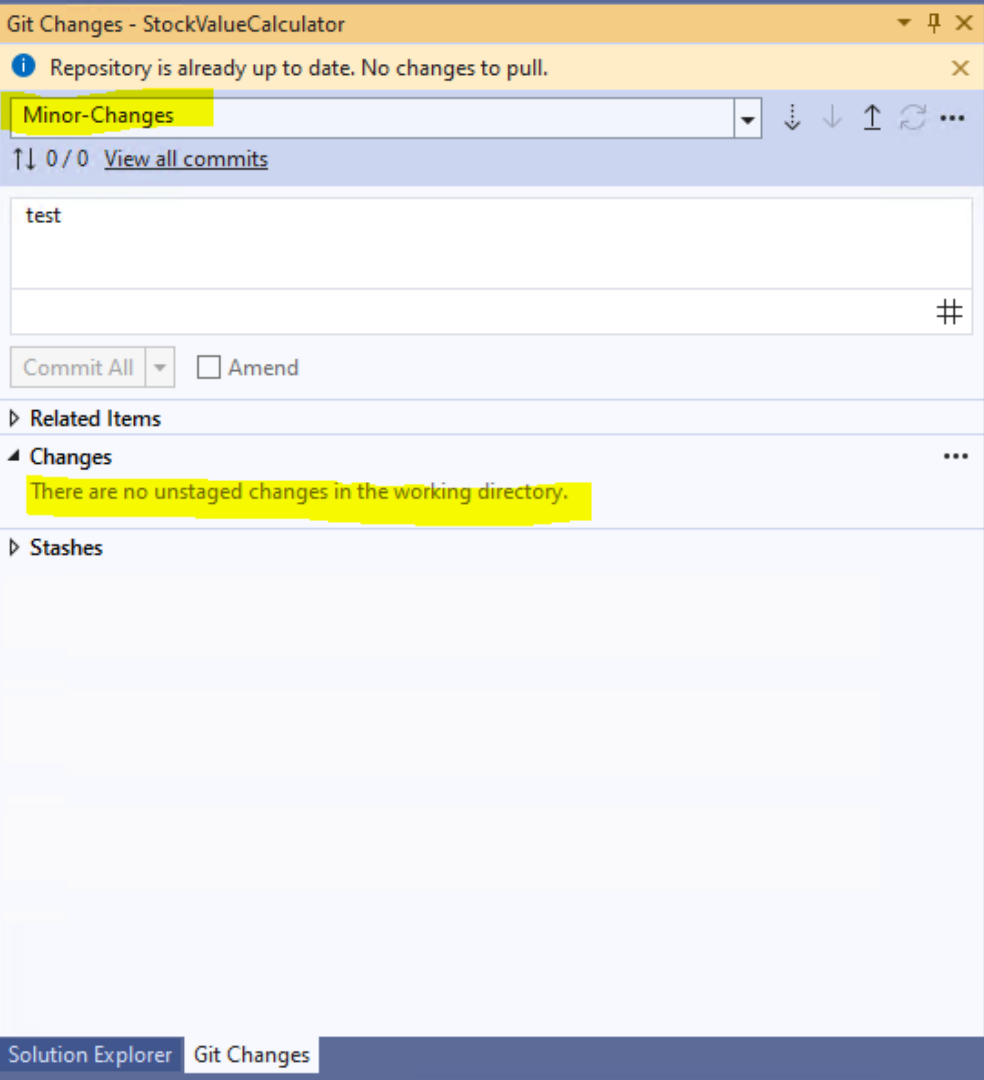
Create a GitHub account (if necessary) and then follow a the “hello world intro to GitHub” on the GitHub portal. Then use Visual Studio in conjunction with Github to manage versioning.

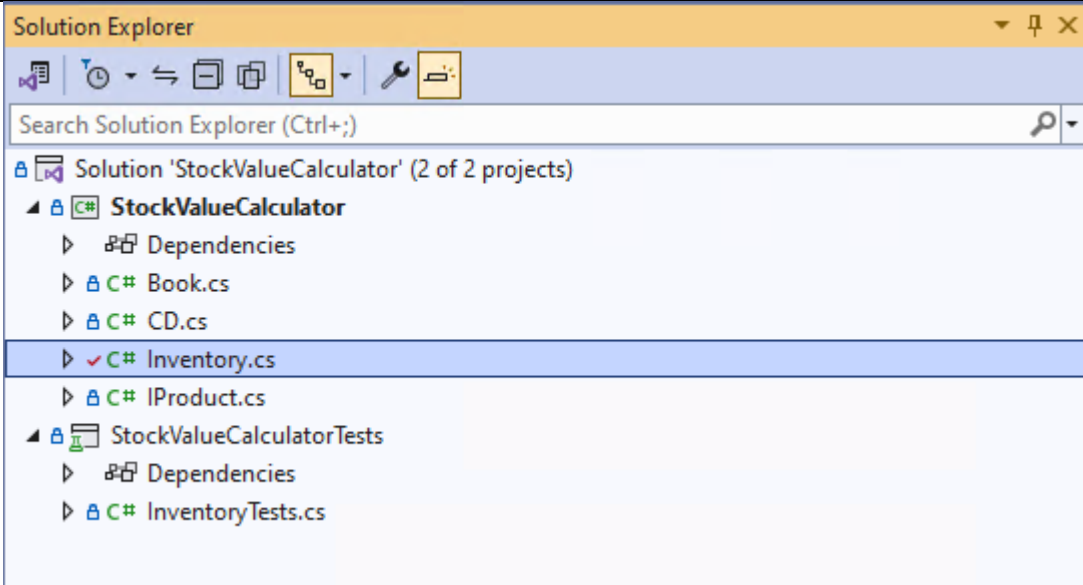
### Steps to Complete

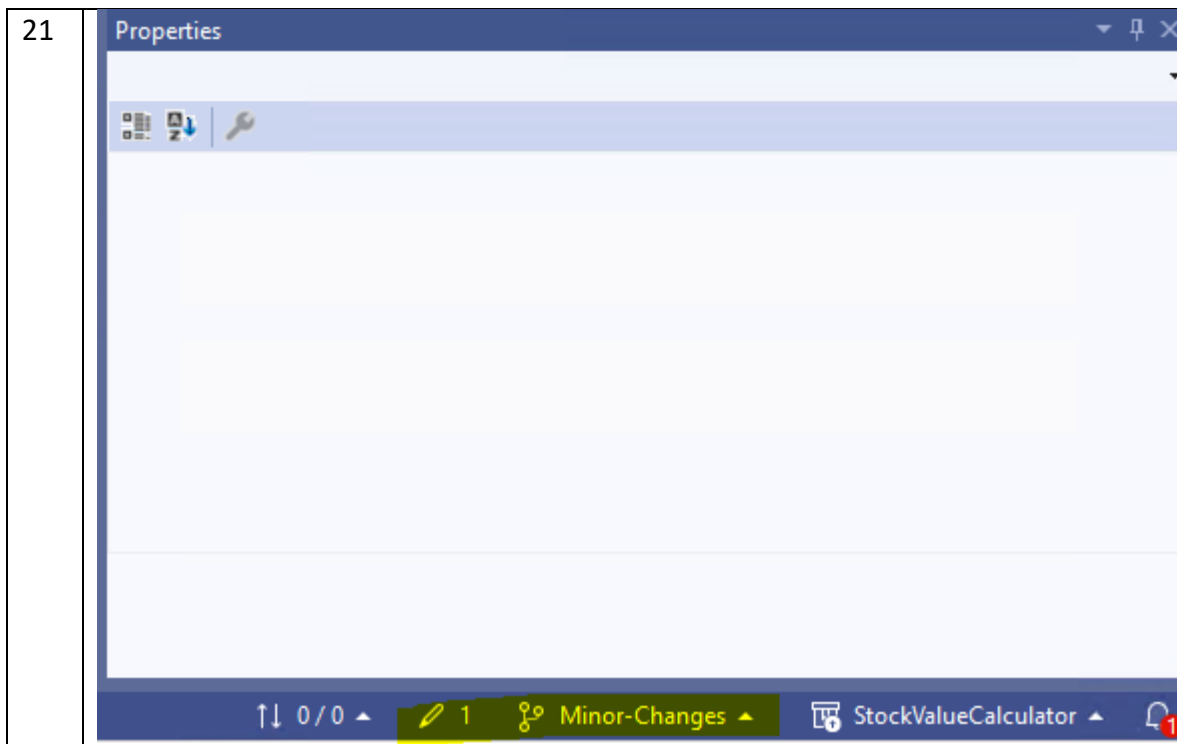
1	<b>Create your GitHub account</b> <a href="https://docs.github.com/en/get-started/start-your-journey/creating-an-account-on-github">https://docs.github.com/en/get-started/start-your-journey/creating-an-account-on-github</a>
2	<b>Create a repository and use it to merge changes into the main branch from a another</b> <a href="https://docs.github.com/en/get-started/start-your-journey/hello-world">https://docs.github.com/en/get-started/start-your-journey/hello-world</a>
3	<b>Using GitHub with Visual Studio</b> <ol style="list-style-type: none"> <li>Using Visual Studio Open the Solution called StockValueCalculator.sln located in Labs\00 Introduction to Git and GitHub\Begin folder</li> <li>Under the Git item on the main menu select Create Git Repository</li> <li>Sign in to GitHub using the account you used earlier.</li> </ol>
4	



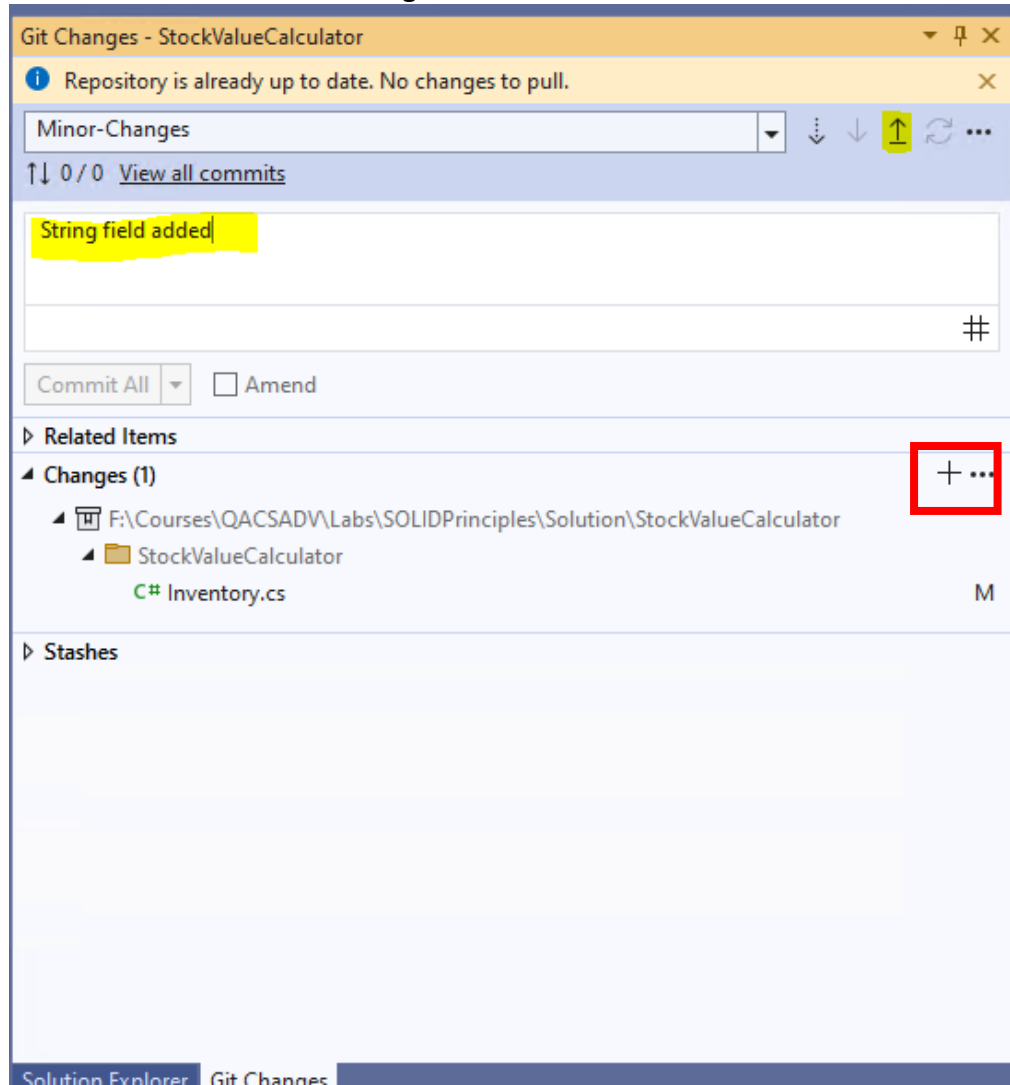
5	Add your initials to the end of the auto-generated repository name ("PC" in the example below), deselect the Private repository option and select the Add a README.md
6	
7	Then click on the Create and Push button
8	Your project has now been uploaded to GitHub and you also have a local copy(clone) on the local file system.
9	
10	Now make a branch to support updates to the current project. Click on the Git item in the main menu of Visual Studio and select the <b>New Branch</b> Item. Name the branch <b>Minor-Changes</b> and confirm that the <b>Checkout branch</b> option is selected

11	
12	Click on create, and if prompted for further input, accept the defaults and continue.
13	The <b>Git Changes</b> window should show that there are currently no changes deployed to the branch named <b>Minor-Changes</b>
14	
15	Open the Inventory.cs file and declare a private field of type string named S1 and initialise this to have a default value of "test"

16	<pre> public class Inventory {     private List&lt;IProduct&gt; products = new List&lt;IProduct&gt;();     private string s1 = "test";      public void AddProduct(IProduct product)     { </pre>
17	Notice that the modified file has a red tick associated with it in the Solution Explorer window
18	
19	Open the Git Changes tab and <b>double click</b> on the Inventory.cs file. The diff view appears highlighting the change (Additional line) added to the source code
20	Additionally, the status bar at displayed at the bottom of the Visual Studio window displays 1 change to the file associated with the branch named Minor-Changes

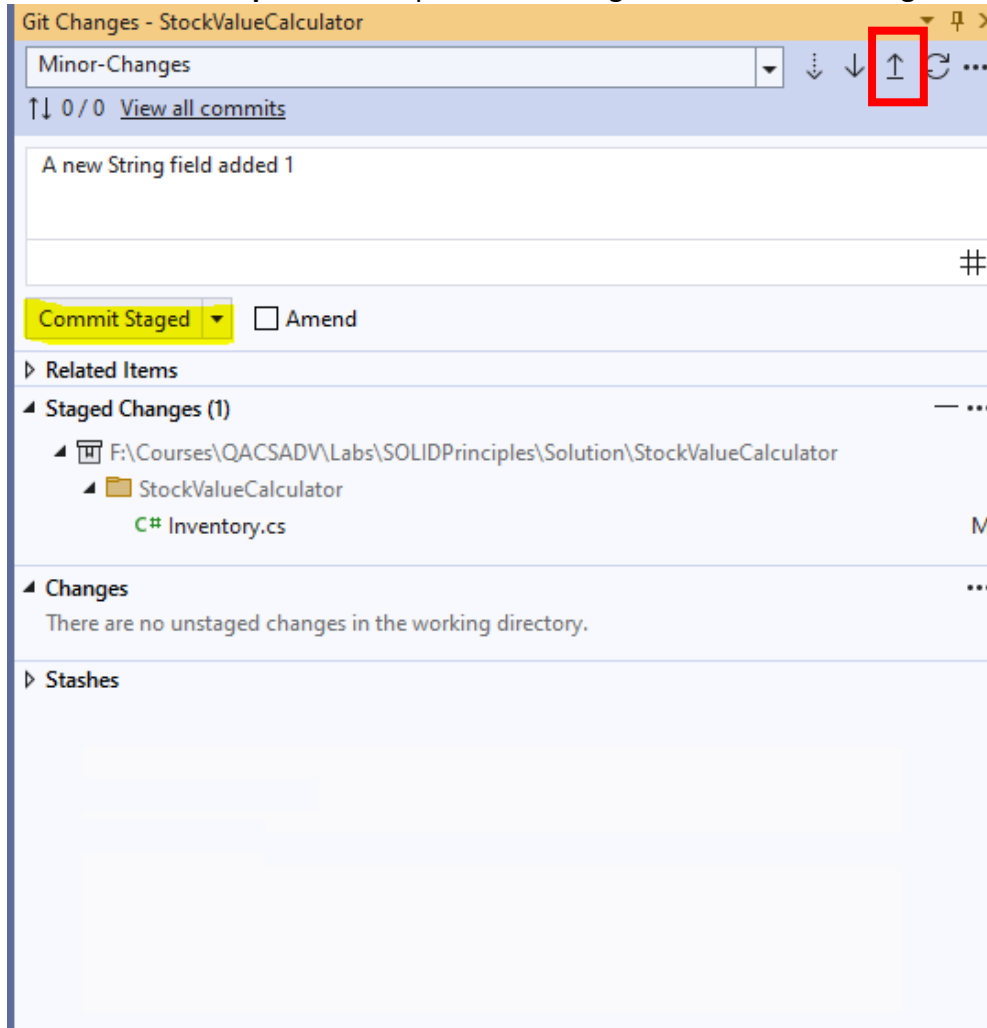


- 22 Now we will commit the changes to the GitHub Repo. In solution explorer right click on the Inventory.cs file select the Git menu item and then select the Commit or Stash item. Enter a description in the input text field, such as “string field added”. and click on the + Stage All button

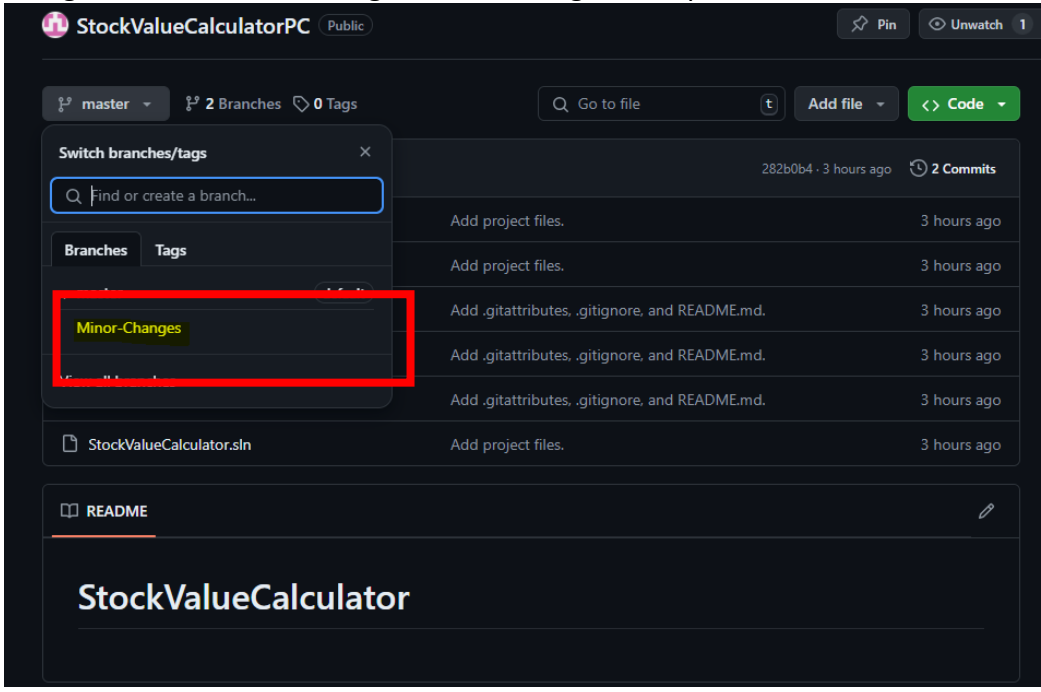




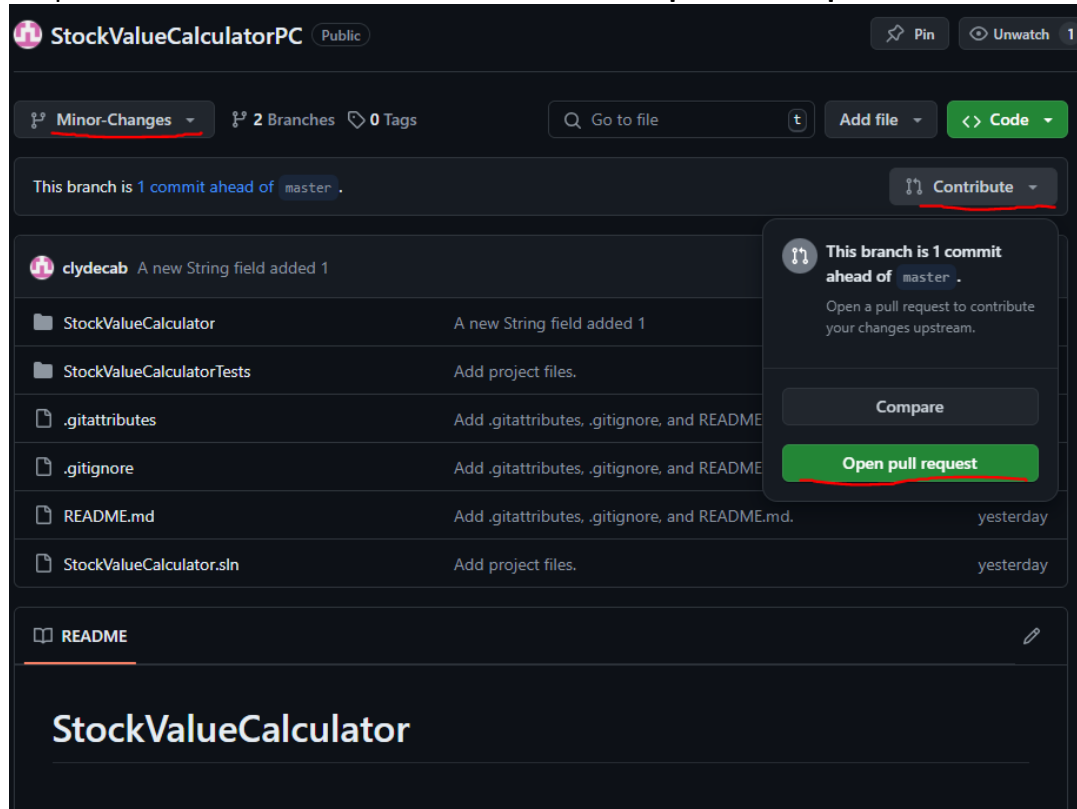
- 23 The UI should now display as shown below. Click on the **Commit Staged** button. Then click on the **up arrow** to upload the changes to the Minor-Changes Branch



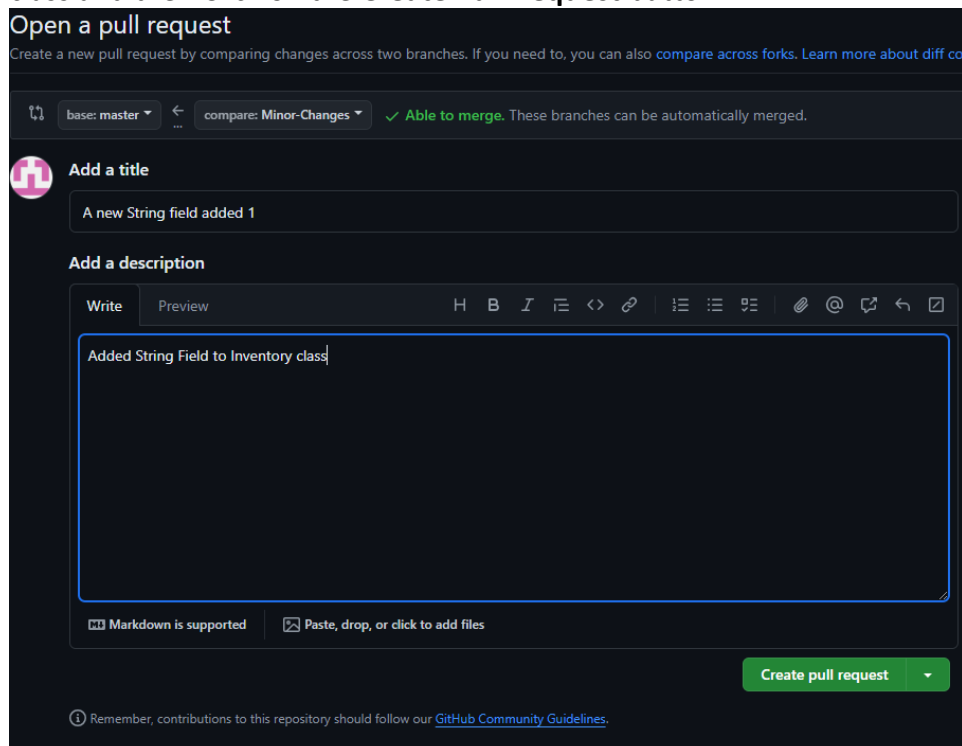
- 24 Check that the changes have been pushed to the repo by using a browser to navigate to <https://github.com/> and select the StockValueCalculatorXX repo.

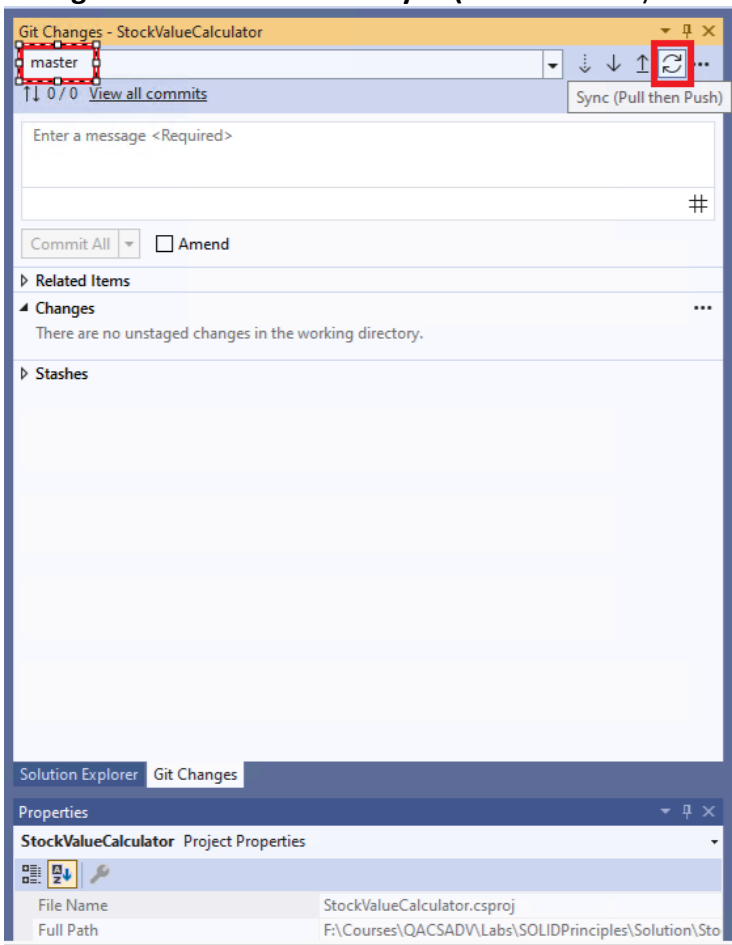
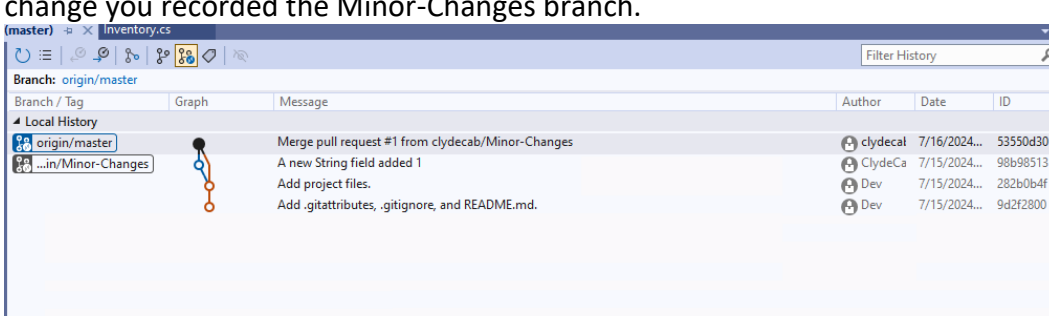
25	<p>Navigate to the Minor-Changes branch using the drop down menu</p>  <p>The screenshot shows the GitHub interface for the repository 'StockValueCalculatorPC'. At the top, there's a header with the repository name, 'Public' status, and buttons for 'Pin' and 'Unwatch'. Below this, a navigation bar shows the current branch 'master', the number of branches '2', and tags '0'. A search bar 'Go to file' and buttons 'Add file' and 'Code' are also present. A 'Switch branches/tags' dropdown menu is open, displaying a search bar 'Find or create a branch...' and a list of branches. The 'Minor-Changes' branch is highlighted with a red rectangular box. Below the branches list, there's a section for 'Commits' showing a list of recent commits with their messages and timestamps. At the bottom, there's a 'README' section with the title 'StockValueCalculator'.</p>
26	<p>Open the StockValue calculator folder that contains the Inventory.cs file. You should see the modifications have been pushed to the branch named Minor-Changes. The master branch should still have the original version. We will now merge the changes recorded in the Minor-Changes branch with the master branch</p>

- 27 In the GitHub Portal ensure the Minor-Changes branch is selected in the dropdown. Then from the Contribute menu click **Open Pull Request**



- 28 In the form that appears add a description e.g. Added String Field to Inventory class and then Click on the **Create Pull Request** button.



29	On the next page click on the <b>Merge pull request</b> and then <b>Confirm merge</b> buttons in order to commit the changes with the master branch
30	Go back to Visual Studio select the <b>master</b> branch from the dropdown in the <b>Git Changes</b> tab and click on the <b>Sync (Push and Pull)</b> button
	
31	The graph displaying the updates to the master branch will now include the change you recorded the Minor-Changes branch.
	
32	Feel free to use GitHub when working on other exercises in the labs that follow this one.

## 02 REVIEW - Object-Oriented Programming (OOP) Principles

The objective of these exercises is to consolidate your understanding of object-oriented programming (OOP) concepts and to experience the run time behaviours of reference and value types. The second exercise aims to start authoring classes with instance methods and data.

### Part 1 – Calling pre-supplied methods

1	Open the existing pre-supplied Visual Studio solution (.sln) called 'MethodCallingPractice'. It can be found in folder labs\02a Object_Oriented_Programming_Concepts\Begin\Types.																		
2	Open the file Program.cs. Inside the Main method you will see two commented out lines that make calls to a couple of test methods. The 2 test methods contain code that creates and manipulates objects of type Account (a reference type) and int (a value type).																		
3	<p>Open the file Account.cs. You will see the implementation of a reference type (a class) that represents a bank account object.</p> <p>In Program.cs, expand the "Testing Ref Type behaviour" region and examine the TestRefType method and the two PassAccountByxxxx methods. You will note that the balances of the 2 accounts are being displayed 4 times.</p> <p>Create a table like the one below, either with a pen and paper or in NoteBook.exe and by just reading the code, write down what you think the Balance will be when read from the variables ac1 and ac2 at these four points in the code.</p> <table><tr><td></td><td>ac1.Balance</td><td>ac2.Balance</td></tr><tr><td>1st</td><td>_____</td><td></td></tr><tr><td>_____ 2nd</td><td>_____</td><td></td></tr><tr><td>_____ 3rd</td><td>_____</td><td></td></tr><tr><td>_____ 4th</td><td>_____</td><td></td></tr><tr><td>_____</td><td></td><td></td></tr></table>		ac1.Balance	ac2.Balance	1st	_____		_____ 2nd	_____		_____ 3rd	_____		_____ 4th	_____		_____		
	ac1.Balance	ac2.Balance																	
1st	_____																		
_____ 2nd	_____																		
_____ 3rd	_____																		
_____ 4th	_____																		
_____																			
4	Uncomment the line in the Main function that calls the TestRefType method and run the program checking how many of the values you got correct.																		
5	If you answers differed from those shown, use the Visual Studio debugger to step through the code. If you are unsure about the results, please ask your instructor to explain them.																		

### Working with value types

In this section of the lab, you will see how value types work, and how they differ in their behaviour from reference types.

6	In C# the <code>int</code> data type (alias for <code>System.Int32</code> ) is not a class it is a value type and thus will exhibit value type behaviour.															
7	Comment out the call to <code>TestRefType</code> in the <b>Main</b> method of the <b>Program</b> class.															
8	Uncomment the call to <code>TestValueType</code> in the <b>Main</b> method of the <b>Program</b> class.															
9	<p>Examine the method <code>TestValueType</code> and the two <b><code>PassIntByxxxx</code></b> methods. You will note that the ints are being displayed 4 times. <b>Just by reading the code</b>, write down in the table below what you think the value will be when read from the variables <code>num1</code> and <code>num2</code> at these four points in the code.</p> <table><thead><tr><th></th><th>num1</th><th>num2</th></tr></thead><tbody><tr><td>1st</td><td>_____</td><td>_____</td></tr><tr><td>2nd</td><td>_____</td><td>_____</td></tr><tr><td>3rd</td><td>_____</td><td>_____</td></tr><tr><td>4th</td><td>_____</td><td>_____</td></tr></tbody></table>		num1	num2	1st	_____	_____	2nd	_____	_____	3rd	_____	_____	4th	_____	_____
	num1	num2														
1st	_____	_____														
2nd	_____	_____														
3rd	_____	_____														
4th	_____	_____														
10	<p>Run the program and check how many of the values you got correct.</p> <p>If your answers differed from those shown, use the Visual Studio debugger to step through the code. If you are unsure about the results, please ask your instructor to explain them</p>															
11	Now comment out the calls to <code>TestRefType</code> and <code>TestValueType</code> and uncomment the calls to <b><code>TestIntArray</code></b> and <b><code>TestAccountArray</code></b> .															
12	Read the code being executed by each method, there are 2 simple questions to be answered, choose answers. Run the code and see if you were correct.															

### Part 2 – OPTIONAL - Creating a simple Account class

In this next exercise you will simultaneously author class `Account` whilst writing code in class `Program` that uses your new `Account` data type and the functionality it exposes.

1	Open the pre-supplied solution <b><code>ClientExe.sln</code></b> . You will find it in <code>Labs\02a_Object_Oriented_Programming_Concepts\Begin</code> .
2	Open class <code>Program</code> . Expand the <code>Main</code> method.  Type ' <code>Acc</code> ' – there is no data type (class) called <code>Account</code> (yet).
3	Right click Solution ' <b><code>ClientExe</code></b> ' in <code>SolutionExplorer</code> and choose <code>Add/New Project</code> .  In the ' <code>Add New Project</code> ' dialog choose ' <code>Class Library</code> ' name it ' <code>Finance</code> ' and ensure it is placed in the correct <code>\Begin</code> folder.
4	In the project ' <code>Finance</code> ' you now have a file called <b><code>Class1.cs</code></b> containing the code of a class called ' <code>Class1</code> '. Rename the FILE in <code>Solution Explorer</code> to <b><code>Account.cs</code></b> and you will be prompted to rename the class as well.
5	You now have public class <code>Account { }</code> .
6	Inside <code>Main</code> type ' <code>Acc</code> ' again – still no sign of data type <code>Account</code> .

7	In the ClientExe area of Solution Explorer right-click 'Dependencies. Choose 'Add Project Reference', from the Projects tab choose 'Finance' and click 'OK'.
8	Inside Main type 'Acc' again – still no sign of data type Account. But if you type 'Finance'<dot> you can see the Finance.Account data type. Add a 'using Finance' clause at the top of the file and you can now type 'Acc' and see that 'Account' is a known data type.
9	Inside Main declare 2 variables of type Account <pre>Account ac1, ac2;</pre>
10	Now type these 2 statements which show 2 different but useless objects being created. <pre>ac1 = new Account(); ac2 = new Account();</pre>
11	Type a 4th statement and run your code and see that the result is false. <pre>Console.WriteLine(ac1 == ac2); // these are DIFFERENT objs</pre>
12	Keep the 1st statement and DELETE the other 3.
13	You are going to add some functionality to the Account class. Declare 3 private fields inside the class noting the lowercase. <pre>private string holder; private decimal balance; // will default to 0.00 private string accNo; // can contain alphanumeric chars</pre>
14	Use the 'ctor' code snippet to author a 'default constructor'.  A constructor is a special method that runs when an object is being instantiated from the class. Constructor methods can be configured to take parameters (just like ordinary functions). <b>We will look at constructors more fully in the next session.</b>  Give it 2 parameters (name and balance) and 2 statements. The statements should store the parameters in the instance variables (giving the object some initial state) you will need to use the word 'this' once: <pre>public Account(string name, decimal balance) {     holder = name;     this.balance = balance; }</pre>
15	Returning to Main, the 2 statements that create the instances of the Account class will be showing syntax errors. This is because the constructors each need to be given a person's name and a starting balance. Edit the code as shown below. <pre>ac1 = new Account("Fred", 100); ac2 = new Account("Susy", 200);</pre>
16	Each account now has its own 'holder' and 'balance'. Each account also has its own 'accNo' but it has defaulted to null for each of them.

	<p>We would like these 'Student Accounts' to each have an 'accNo' in the format 'SA-nnnn' where nnnn is an 'auto-incremented' sequential number padded to 4 digits with zeros. i.e. SA-0001, SA-0002 for Fred and Susy respectively.</p> <p>Declare a 4th field in Account as follows</p> <pre>private int nxtAccNo; // defaults to 0</pre> <p>Add this next statement to the bottom of the constructor method. Will this work do you think? Will the account numbers be SA-0001 and SA-0002 for 'Fred' and 'Susy' or will they both have SA-0001?</p> <pre>accNo = "SA-" + (++nxtAccNo).ToString().PadLeft(4, '0');</pre> <p>Well, they will both be SA-0001 because each account object will get its own version of nxtAccNo starting at zero.</p> <p>Solve the problem now by marking nxtAccNo as static this means there is only one copy of this numeric field shared between all the instances (but each instance will have its own accNo in format 'SA-nnnn')</p> <pre>private static int nxtAccNo; // defaults to 0 and is shared</pre>
17	<p>Let's prove that it has worked.</p> <p>Open class Account and author a method as follows (it won't compile yet)</p> <pre>public string GetDetails() { }</pre>
18	<p>We would like this method to return a tab-separated string containing the account number, the holder's name and the account balance. Easy to do via concatenation, but here is a perfect opportunity to see again how string interpolation works.</p> <p>Insert this single statement into the method.</p> <pre>return \$"{accNo}\t{holder}\t{balance}";</pre>
19	<p>Return to Main and after creating the 2 Account objects display the details of both objects on the Console. You really should not need to look at the code fragment below to see how to do this.</p> <pre>Console.WriteLine(ac1.GetDetails()); Console.WriteLine(ac2.GetDetails());</pre> <p>To achieve this you could have typed no more than 'cw' Tab Tab an 'a' a &lt;dot&gt; a 'g' then '()'. If you typed more than that then you may want to try it again.</p> <p>Run the code. You should now be seeing output showing 'Fred' and 'Susy's correct account details with unique account numbers SA-0001 and SA-0002.</p>
20	<p>Now add further functionality to the Account class as follows</p> <pre>public void Deposit(decimal amt) {     balance += amt; } public bool withdraw(decimal amt) {</pre>



	<pre>bool result = false; return result;           // just to make it compile }</pre>
21	<p>The Deposit method is straightforward – it will work, there is no upper balance.</p> <p>Withdraw however is a bool method as very soon you will want to ‘listen’ to see if a withdrawal has been successful.</p> <p>Here is the algorithm. If the amount that is being withdrawn is less than or equal to the balance on the account then the balance should be adjusted downwards and result set to true. Implement that code now.</p> <pre>public bool withdraw(decimal amt) {     bool result = false;     if (?? ? ??) {         ?? ? ?           // change the balance         ?? ? ?           // ensure method returns true     }     return result;       // to make it compile }</pre> <p>This would work fine if there was no concept of an overdraft. But these are ‘Student Accounts’ – there is a £500 overdraft limit and it is the same for all Accounts so this should be held in a static (belonging to the class) variable.</p> <p>Declare a static decimal overdraftLimit with a value of 500.</p> <p>Also adjust the Withdraw(decimal amt) method to allow a student to withdraw as long as they do not go beyond their overdraft limit as opposed to below 0.</p>
22	<p>Perform a Deposit into ‘ac1’ and a successful Withdraw from ‘ac2’.</p> <p>Make these method calls before the display of the details. Check the methods have produced the correct values.</p> <p>Now change the parameter passed to Withdraw() so that it would take the student beyond their overdraft limit and check the balance does not change.</p> <p>You probably coded your Withdraw method as follows?</p> <pre>public bool withdraw(decimal amt) {     bool result = false;     if (amt &lt;= (balance + overdraftLimit)) {         balance -= amt;           // change the balance         result = true;           // ensure method returns true     }     return result; }</pre>
23	<p>It works fine but it is not obvious that overdraftLimit is ‘shared’ between all instances.</p> <p>Not clear that it belongs to ‘Account’ rather than ‘this’.</p>

	<p>Make the statement clearer perhaps by changing it to.</p> <pre>if (amt &lt;= (this.balance + Account.overdraftLimit)) {</pre>
24	<p>Now we wish to enable transfers between accounts so that one student could do a little internet banking and 'loan' money to a friend.</p> <p>It could be written as an instance method and if it was its signature might be</p> <pre>public bool Transfer(decimal amt, Account to) {     ... calls to this.Withdraw and to.Deposit }</pre> <p>The code that would invoke it might look like this:</p> <pre>ac1.Transfer(100M, ac2); // 100 from ac1 to ac2</pre> <p>Alternatively, it could be written as a class (static) method with this signature.</p> <pre>public static bool Transfer(Account from, Account to,                            decimal amt) { }</pre> <p>This latter version is perhaps more 'appropriate' for a method which by definition affects 2 accounts, but it is a matter of personal preference.</p> <p>Write this method signature now, it will not compile yet of course.</p>
25	<p>Now for the implementation.</p> <p>Because it is boolean it should declare a boolean result variable and set it to false (just being pessimistic).</p> <p>The final statement should return the result.</p> <p>In the main body of the method you should do a Withdraw from the 'from' account and only if that is a successful Withdraw() should it then do a Deposit() into the 'to' Account.</p> <p>After the Withdraw &amp; Deposit (or neither) operations have completed you should always display the result of attempting the transfer in the format</p> <p>"Transfer Successful: Yes"</p> <p>or</p> <p>"Transfer Successful: No"</p> <p>You should be able to implement that very easily without looking at the code block below that shows you the final code.</p> <p>Have an attempt now. It is a good chance to test yourself out on whether you understood how the conditional operator ( ?: ) works as you should use it to produce the "Yes" or "No" strings.</p> <pre>public static bool Transfer(Account from, Account to,</pre>

	<pre>                                 decimal amt) {     bool result = false;     if (from.Withdraw(amt))     {         to.Deposit(amt);         result = true;     }      Console.WriteLine(         \$"Transfer Successful: {(result ? "YES" : "NO")}");     return result; } </pre>
26	<p>Now go back to Main() and perform a Transfer of a modest sum from 'ac1' to 'ac2' or the other way round, depends which student 'Fred' or 'Susy' you think has blown their 'terms' allowance during 'freshers week'.</p> <p>Remember the method you are calling is static not instance.</p>
27	<p>Before we wrap up note that although we can display the 'Details' of any account object we do not have the ability to simply find out the balance on the account or in fact the name of the holder of an account.</p> <p>Author and code now a simple public decimal GetBalance() method and a public string GetHolder() method. They will be needed in the challenges that follow and in the next chapter's lab.</p>

### If You Have Time: Coding Challenge 1

28	<p>Add the 2 account references 'ac1' and 'ac2' to a List of Account objects, add a 3rd and a 4th if you want.</p> <p>Author a method (in Program) called ProcessAccounts with the following signature.</p> <pre> public static void ProcessAccounts(List&lt;Account&gt; accs) { } </pre> <p>This method should loop through the accounts it receives and add a £10 bonus (call Deposit()) to each of them.</p> <p>Call the method from Main and after getting control back loop through the array yourself displaying the account holders name and the account balance to check that they have all gone up by £10.</p>
----	---

### If You Have Time: Coding Challenge 2

29	<p>Declare and create a List of string called names exactly as follows</p> <pre> List&lt;string&gt; names = new List&lt;string&gt; {"Ann", "Anne", "Annie", "Anneka", "Annabel"}; </pre>
30	<p>Declare and create a List of Account objects called studentAccs</p>

	<p>Declare and create an instance of class Random, it has a useful Next() method that you soon will use repeatedly.</p>
31	<p>Write a simple for loop that runs 'the right number of times' that creates an account for each of the names and adds the object to the studentAccs collection.</p> <p>The account holders name should be the 'next' name from the names array.</p> <p>Use instance method Next() of the Random object you created earlier to generate a random number of £ in the range 10-99 inclusive to be used as opening balance.</p> <pre>for(int i = 0; i &lt; ???; i++) {     ???.??(new ??(??,??)); }</pre>
32	<p>Write a foreach loop that steps through the list of accounts displaying the details of each account.</p> <p>Run the code.</p> <p>The 5 names should now have an account each with a balance in the range £10 - 99 inclusive.</p> <p>Now you are going to write code in a loop so that a Transfer happens from each account holder to the 'next' (5 transfers in total).</p> <p>i.e. it will transfer money from 'Ann's account to 'Anne's account.</p> <p>From 'Anne's account to 'Annie's account etc and lastly (the tricky bit) from the final person 'Annabel' to the first person 'Ann's account.</p> <p>The amount of money that each person transfers out is the length of their own name.</p> <p>It is easy to write code to determine the length of the account holders name as class String has a 'Length' property.</p> <p>So Ann will give £3 to Anne. Anne will give £4 to Annie. Annie will give £5 to Anneka... finally Annabel will give £7 to Ann.</p> <p>Each person's balance will drop by £1 except Ann whose balance will go up by £4 (£3 out and £7 in).</p> <p>Write this code now without hard coding any names or amounts.</p>

**Before moving on don't forget to commit and push your work to GitHub.**

## 02d REVIEW Quiz Questions

A review of the OOP concepts presented in the Foundations course

### Q1

Which of the following best defines Object - Oriented Programming?

- A. A programming paradigm based on functions and procedures.
- B. A programming style that focuses on using classes and objects to design applications.
- C. A style of programming focused entirely on data manipulation using loops and conditionals.
- D. A method of coding without any use of predefined libraries.

**QA**

### Q2

Which of the following is NOT a principle of OOP?

- A. Encapsulation
- B. Polymorphism
- C. Abstraction
- D. Iteration

**QA**

**Q3**

What does the keyword `this` refer to in C#?

- A. The base class of the current object.
- B. The current instance of the class.
- C. A static instance of the class.
- D. A reference to the class's namespace.

**QA**

6

**Q4**

How is method overloading achieved in C#?

- A. By using different return types for methods with the same name.
- B. By using different access modifiers for methods with the same name.
- C. By defining multiple methods with the same name but different numbers of parameters and/or different parameter data types.
- D. By defining methods in base and derived classes.

**QA**

7

## Q5

What is the correct syntax to create an object of a class named Person in C#?

Select all that apply

- A. `Person obj = new();`
- B. `Person obj = Person();`
- C. `Person obj = new Person();`
- D. `Person obj = new object Person();`

QA

8

## Q6

Which of the following statements about constructors in C# is TRUE?

- A. Constructors can have a return type.
- B. A class can only have one constructor.
- C. Constructors can be overloaded.
- D. Constructors cannot be parameterized.

QA

9

**Q7**

What is the purpose of a property in C#?

- A. To provide a way to encapsulate data fields.
- B. To define functions within a class.
- C. To enforce method overloading.
- D. To initialize objects automatically.

**QA**

10

**Q8**

Which of the following is TRUE about auto-implemented properties in C#?

- A. They require explicit implementation of getter and setter methods.
- B. They automatically create a private, backing field for the property.
- C. They cannot have an initial value.
- D. They can only be used in interfaces.

**QA**

11



## Q9

How can you create a read-only property in C#?

- A. By defining only a set accessor.
- B. By defining only a get accessor.
- C. By using the readonly keyword.
- D. By defining both get and set accessors with private access.

QA

12

## Q10

What is the output of the following code?

```
class Sample {  
    public int MyProperty { get; private set; } =  
    10;  
  
    public Sample() {  
        MyProperty = 20;  
    }  
}  
  
static void Main() {  
    Sample obj = new Sample();  
    Console.WriteLine(obj.MyProperty);  
}
```

- A. 10
- B. 20
- C. Error: Cannot modify property
- D. Undefined behavior

QA

13

## Q11

Which of the following demonstrates a property with a custom backing field?

- A.**
- ```
private int _value;
public int value {
    get { return _value; }
    set { _value = value; }
}
```
- B.**
- ```
public int value { get; set; }
```
- C.**
- ```
private int _value;

public int value {
    get => _value;
    set => _value = value;
}
```
- D.** Both A and C

## Q12

How would you define a property in C# that ensures a value cannot exceed a maximum limit?

- A. By using a custom get accessor.
- B. By adding validation logic in the set accessor.
- C. By using a readonly keyword with the property.
- D. By defining a custom exception for invalid values.

**Q13**

What is the purpose of a default constructor in C#?

- A. To provide default values to all fields.
- B. To allow object creation without passing arguments.
- C. To ensure all methods are initialized automatically.
- D. To override the base class's constructor.

QA

16

**Q14**

Which of the following is TRUE about constructor chaining in C#?

- A. It is achieved by invoking another constructor of the same class using `base()`.
- B. It is achieved by invoking another constructor of the same class using `this()`.
- C. It is only possible in derived classes.
- D. It allows invoking constructors of multiple classes simultaneously.

QA

17

**Q15**

What will happen if no constructor is defined in a C# class?

- A. The class cannot be instantiated.
- B. The class is treated as an abstract class.
- C. The compiler provides a default constructor with no parameters.
- D. An error is thrown at runtime.

**Q16**

What is the use of the static constructor in C#?

- A. To initialize instance members of the class.
- B. To override the behaviour of a base class constructor.
- C. To provide a default constructor when none is defined.
- D. To initialize static members of the class.

## Q17

Given the following code, what is the data type of the query variable?

```
string[] fruit = { "grapes",  
    "apples", "pears", "cherries"  
};  
var query =  
    from f in fruit  
    where f.Length > 5  
    select f;
```

- A. IQueryable<string>
- B. string
- C. IEnumerable<string>
- D. IEnumerable<int>

## Q18

Given a Car class has Make, Model and Registration properties and the following code, how would you filter the collection to return all cars that are Fords in reverse alphabetical sequence of their registrations?

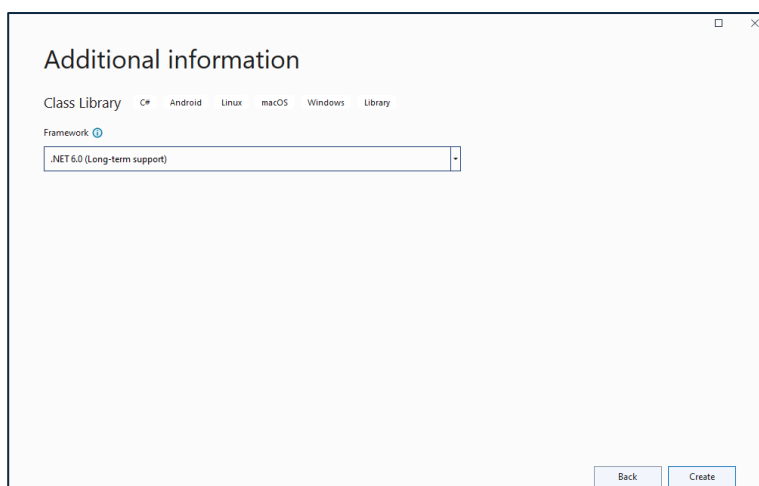
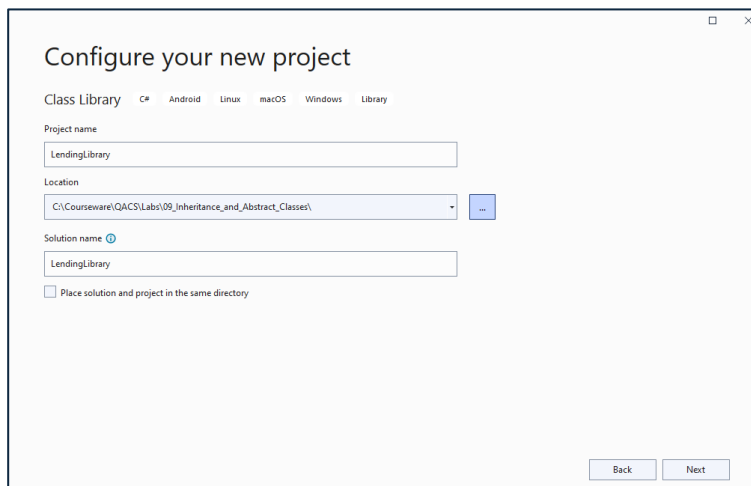
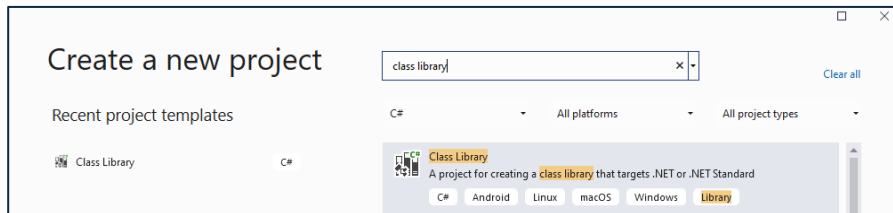
```
List<Car> cars = new(){  
    new Car("Ford", "Mondeo", "AB12 ABC"),  
    new Car("AUDI", "TT", "AU01 TT0"),  
    new Car("Ford", "Fiesta", "FF10 ILF"),  
    new Car("Ford", "Fiesta", "JB00 OOT") };
```

- A. IEnumerable<Car> fords = from car in cars  
 where car.Make == "Ford"  
 orderbydescending car.Registration  
 select car;
- B. IEnumerable<Car> fords = cars  
 .Where(c => c.Make == "Ford")  
 .OrderByDescending(c => c.Registration);
- C. IEnumerable<Car> fords = from cars  
 where car.Make == "Ford"  
 orderby car.Registration descending  
 select car;
- D. IEnumerable<Car> fords = cars  
 .Where(c => c.Make == "Ford")  
 .OrderByDescending(c => c.Registration)  
 .Select(c => new {c.Make, c.Model,  
 c.Registration})

## 03 Inheritance and Abstract Classes

The objective of this exercise is to consolidate your understanding of inheritance by building an inheritance hierarchy.

- 1 Locate the '**Labs\03\_Inheritance\_and\_Abstract\_Classes\Begin**' folder and, inside it, create a new Visual Studio project of type Class Library called **LendingLibrary**.



Delete **Class1**

Add to this solution an XUnit Test project called **TestProject**

- 2 In the Test project, replace the starter test with the 'Create' test as found in the 'Labs\03\_Inheritance\_and\_Abstract\_Classes\Assets' folder (copy just the first part of the file, up to the end of the test).

```
[Fact]
public void Create()
{
    Library library = new Library();
    Member greta = library.Add(name: "Greta Thunberg", age: 15);
    Member donald = donald = library.Add(name: "Donald Trump", age: 73);

    Assert.Equal(2, library.NumberOfMembers);
    Assert.Equal(1, greta.MembershipNumber);
    Assert.Equal(2, donald.MembershipNumber);
}
```

- 3 Create **Library** and **Member** classes in the **LendingLibrary** project, then copy in the code listed just after the 'Create' test.

```
namespace LendingLibrary
{
    public class Library
    {
        Dictionary<int, Member> members = new Dictionary<int, Member>();

        public int NumberOfMembers => members.Keys.Count;

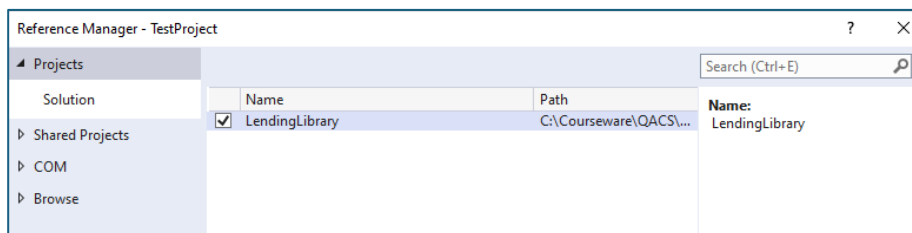
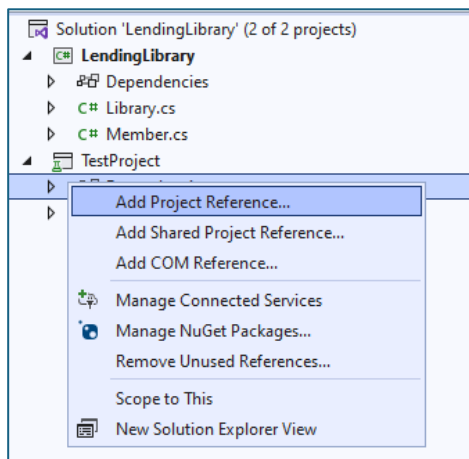
        int GetNextFreeMembershipNumber()
        {
            return (members.Keys.Count == 0) ? 1 : members.Keys.Max() + 1;
        }

        public Member Add(string name, int age)
        {
            Member member = new Member(name, age, GetNextFreeMembershipNumber());
            members.Add(member.MembershipNumber, member);
            return member;
        }
    }
}
```

```
namespace LendingLibrary
{
    public class Member
    {
        public string Name { get; }
        public int MembershipNumber { get; }
        public int Age { get; }

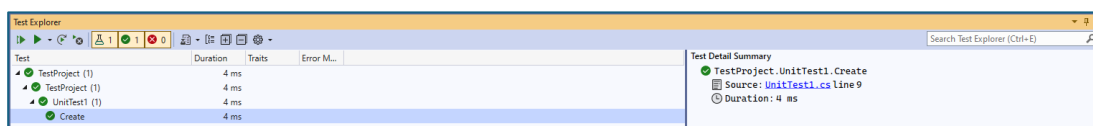
        public Member(string name, int age, int membershipNumber)
        {
            this.Name = name;
            this.Age = age;
            this.MembershipNumber = membershipNumber;
        }
    }
}
```

In the test project, add a project reference to the **LendingLibrary** project.



Add a using statement 'using LendingLibrary;' to **UnitTest1.cs**

Run the **Create** test and confirm it passes.





|   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|---|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 4 | <p>We are going to need the <b>library</b> instance and <b>greta</b> and <b>donald</b> member objects in further tests, so to avoid duplication of code, move the declarations of these to the class level and initialise them in the constructor.</p> <p><b>Note:</b> If you get stuck, we've shown it in the Assets folder.</p>                                                                                                                                                                                                              |
| 5 | <p>Later on, we're going to need some additional properties of <b>Member</b>, so add these in and get Visual Studio to create the properties.</p> <pre> public UnitTest1() {     library = new Library();     greta = library.Add(name: "Greta Thunberg", age: 15);     greta.Street = "Queen Street";     greta.City = "Stockholm";     greta.OutstandingFines = 25M;     donald = library.Add(name: "Donald Trump", age: 73);     donald.Street = "Trump Tower";     donald.City = "New York";     donald.OutstandingFines = 2500M; } </pre> |
| 6 | <p>In this library we have different rules for borrowing a book, dependent on whether the member is over 16 or not.</p> <p>We are going to need this enum. Add it to your <b>LendingLibrary</b>, in a file called <b>BookCategory.cs</b></p> <pre> namespace LendingLibrary {     public enum BookCategory     {         Children,         Adult     } } </pre>                                                                                                                                                                                |
|   | <p>We will need a <b>Book</b> class. Add the class and the properties and get Visual Studio to generate the constructor:</p>                                                                                                                                                                                                                                                                                                                                                                                                                   |

```
public class Book
{
    public string Title { get; }
    public BookCategory Category { get; }
    public int BookCode { get; }

    public Book(string title, BookCategory category, int bookCode)
    {
        Title = title;
        Category = category;
        BookCode = bookCode;
    }
}
```

We will also need some Book code in the **Library** class:

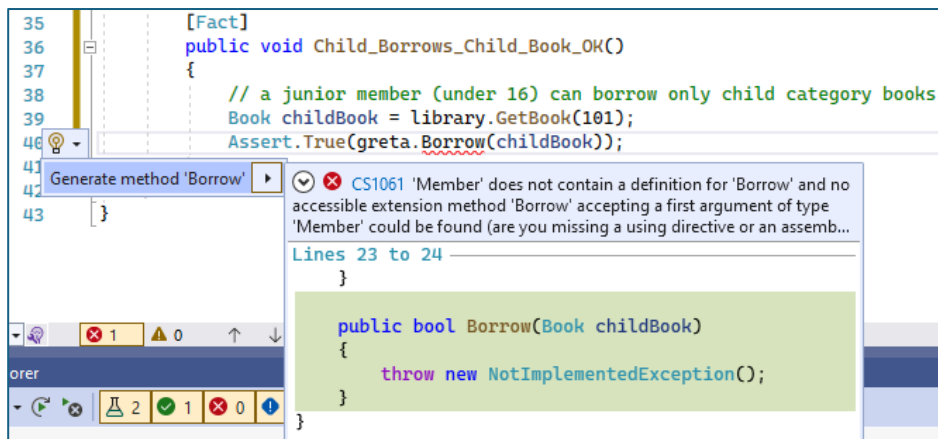
```
Dictionary<int, Book> books = new Dictionary<int, Book>();

public Book GetBook(int code)
{
    return books[code];
}

public Library()
{
    books.Add(100, new Book("Walls have ears", BookCategory.Adult, 100));
    books.Add(101, new Book("Noddy goes to Toytown", BookCategory.Children, 101));
}
```

- 7 In the TestProject, add in the **Child\_Borrows\_Child\_Book\_OK** test from the **Assets** folder.

Get Visual Studio to resolve the **Borrow()** method:



Populate the method like this:

```
public bool Borrow(Book book)
{
    return true;
}
```

|   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|---|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|   | Run the Test and confirm it passes.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 8 | <p>Add in the test <b>Child_Borrows_Adult_Book_Fails</b>.</p> <pre>[Fact] public void Child_Borrows_Adult_Book_Fails() {     // a junior member (under 16) can borrow only child category books     Book adultBook = library.GetBook(100);     Assert.False(greta.Borrow(adultBook)); }</pre> <p>Run this Test. It fails because it is currently hard-coded to return true.</p> <p>Now we need to modify Borrow:</p> <pre>public bool Borrow(Book book) {     return book.Category == BookCategory.Children; }</pre> <p>Re-run the test. It should now pass.</p> |
| 9 | <p>Add the test <b>Adult_Can_Borrow_Any_Book</b>.</p> <pre>[Fact] public void Adult_Can_Borrow_Any_Book() {     // an adult member (over 16) can borrow any book     Book adultBook = library.GetBook(100);     Book childBook = library.GetBook(101);     Assert.True(donald.Borrow(adultBook));     Assert.True(donald.Borrow(childBook)); }</pre> <p>Run this Test. It fails because the <b>Borrow()</b> method only returns true for children's books.</p> <p>Modify the <b>Borrow</b> code again:</p>                                                       |

```
public bool Borrow(Book book)
{
    if (Age >= 16)
    {
        return true;
    }
    else
    {
        return (book.Category == BookCategory.Children);
    }
}
```

Run the Tests. All tests should now pass.

- 10 In this library, fines are handled differently for juniors and adults. Juniors must provide a CashFund; Adults must provide BankTransfer details.

Add in the two 'Fines' tests:

```
[Fact]
public void Child_Pays_Fine_From_Cash_Fund()
{
    greta.CashFund = 20M;
    greta.PayFine(7M);
    Assert.Equal(13M, greta.CashFund);
}

[Fact]
public void Adult_Pays_Fine_By_Bank_Transfer()
{
    donald.SetupBankTransferLimit(20M);
    donald.PayFine(7M);
    Assert.Equal(13M, donald.BankTransferAvailable);
}
```

Add this to your **Member** class:

```
public decimal CashFund { get; set; }

public void PayFine(decimal fine)
{
    if (Age < 16)
    {
        CashFund -= fine;
    }
    else
    {
        BankTransferAvailable -= fine;
    }
}

public decimal BankTransferAvailable { get; private set; }
public void SetupBankTransferLimit(decimal amount)
{
    BankTransferAvailable += amount;
}
```

|  |                         |
|--|-------------------------|
|  | Confirm all tests pass. |
|--|-------------------------|

### Where we are so far

OK – it works.

But can you see how we are constantly changing working code as we discover more about the junior and adult rules that apply to this library.

In a real project, this means we are *constantly* breaking working code.

Now we will switch to **Inheritance** to see if this helps the situation.

|    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|----|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 11 | <p>We are going to refactor the <b>Member</b> class. In order to compare versions, we will do this:</p> <ol style="list-style-type: none"> <li>1) Copy+Paste <b>Member.cs</b></li> <li>2) Rename the original to <b>Member1.cs</b>. When it asks you if you want Visual Studio to perform a rename, answer <b>No</b>.</li> <li>3) Rename the copy to <b>Member2.cs</b>.</li> <li>4) In Member1.cs, press <b>Ctrl+A Ctrl+K Ctrl+C</b> to comment out the entire class.</li> <li>5) Create two folders in LendingLibrary: <ol style="list-style-type: none"> <li>a. 01 Without Inheritance</li> <li>b. 02 With Inheritance</li> </ol> </li> <li>6) And move Member1.cs to <b>01 Without Inheritance</b> and Member2.cs to <b>02 With Inheritance</b>.</li> </ol> <p>Your project now only has one uncommented Member class in Member2.cs.</p> <p>All tests will pass as no code has been changed.</p> |
| 12 | <p>In the 02 With Inheritance folder, add two new classes: <b>JuniorMember</b> and <b>AdultMember</b>.</p> <p>Adjust their namespaces to be just <b>LendingLibrary</b>.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 13 | <p>Get both of these subclasses to derive from the <b>Member</b> base class.</p> <p>Implement a constructor that passes all parameters to the base class constructor:</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |

```
namespace LendingLibrary
{
    public class JuniorMember : Member
    {
        public JuniorMember(string name, int age, int membershipNumber) :
            base(name, age, membershipNumber)
        {
        }
    }
}
```

Do the same for **AdultMember**.

- 14 We need to modify **Library.Add** to create either a *Junior* or an *Adult* member. Change Library.Add() to this:

```
public Member Add(string name, int age)
{
    Member member;
    if (age < 16)
    {
        member = new JuniorMember(name, age, GetNextFreeMembershipNumber());
    }
    else
    {
        member = new AdultMember(name, age, GetNextFreeMembershipNumber());
    }
    members.Add(member.MembershipNumber, member);
    return member;
}
```

All tests should pass.

- 15 Actually, we want to disallow creating 'Member' objects – clients should be forced to create either Junior or Adult members.

Make the Member class **abstract**.

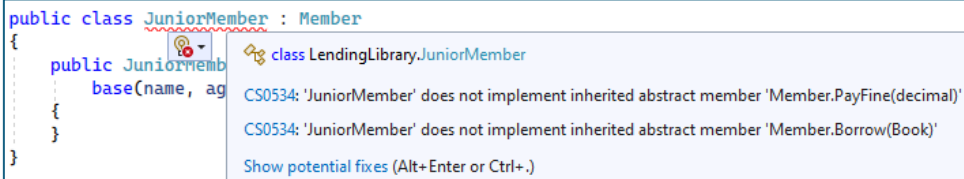
- 16 In Member2.cs, copy Borrow(), CashFund, PayFine(), BankTransferAvailable and SetUpBankTransferLimit() into notepad, deleting them from Member.

- 17 In Member, insert these two abstract methods:

```
public abstract bool Borrow(Book book);
public abstract void PayFine(decimal fine);
```

These abstract methods replace the concrete versions we just deleted.

18 Go to **JuniorMember**. You will see a red squiggly.



```
public class JuniorMember : Member
{
    public JuniorMember(
        base(name, ag
    {
    }
}
```

class LendingLibrary.JuniorMember

CS0534: 'JuniorMember' does not implement inherited abstract member 'Member.PayFine(decimal)'

CS0534: 'JuniorMember' does not implement inherited abstract member 'Member.Borrow(Book)'

Show potential fixes (Alt+Enter or Ctrl+.)

Using **Ctrl+dot**, implement the Abstract class. This will put in the signatures of the methods defined in Member:

```
public override bool Borrow(Book book)
{
    throw new NotImplementedException();
}

public override void PayFine(decimal fine)
{
    throw new NotImplementedException();
}
```

19 Inside each of these members, copy the relevant code from that which you stored in notepad that is specific just to *Junior* members

Repeat for AdultMember.

- You will no longer need the 'if' statement because you are removing the code that doesn't apply
- You will need to paste in the **CashFund** property for the JuniorMember, and the **BankTransferAvailable** and associated **SetUpBankTransferLimit()** method for the AdultMember.

20 If you now go back to the tests, you can see that it doesn't know that greta is a JuniorMember and Donald is an AdultMamber:

```
public class UnitTest1
{
    Library library;
    Member greta;
    Member donald;
```

There are two ways of fixing this:

- 1) Make greta a Junior and donald an Adult.
- 2) Find a form of word that works for both such that the client software is unaware as to whether they are Junior or Adult.

|    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|----|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|    | We'll do both...                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 21 | <p>Make these changes:</p> <pre> Library library; JuniorMember greta; AdultMember donald;  public UnitTest1() {     library = new Library();     greta =(JuniorMember) library.Add(name: "Greta Thunberg", age: 15);     greta.Street = "Queen Street";     greta.City = "Stockholm";     greta.OutstandingFines = 25M;      donald = donald = (AdultMember)library.Add(name: "Donald Trump", age: 73);     donald.Street = "Trump Tower";     donald.City = "New York";     donald.OutstandingFines = 2500M; } </pre> |
| 22 | The tests will pass, however, this is not a great solution because the client code is now acutely aware of the subclasses, meaning if a new subclass is invented, for example, <i>StudentMember</i> , you will have to modify the client code.                                                                                                                                                                                                                                                                         |
| 23 | Use Ctrl+z (Undo) to remove the changes you made in step 21.                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 24 | <p>A better way of looking at it is:</p> <p><i>'Is there some form of words that could operate at the Member level (i.e., for every type of Member) that makes sense to both Junior and Adult members and could be interpreted correctly by both of them?'</i></p> <p>i.e., the same intent but they have different implementations?</p> <p>How about <b>SetFineLimit()</b> and <b>GetFineCredit()</b> ?</p>                                                                                                           |
| 25 | <p>Make these changes:</p> <p>TestProject</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |



```
[Fact]
public void Child_Pays_Fine_From_Cash_Fund()
{
    greta.SetFineLimit(20M);
    greta.PayFine(7M);
    Assert.Equal(13M, greta.GetFineCredit());
}

[Fact]
public void Adult_Pays_Fine_By_Bank_Transfer()
{
    donald.SetFineLimit(20M);
    donald.PayFine(7M);
    Assert.Equal(13M, donald.GetFineCredit());
}
```

Member:

```
public abstract void SetFineLimit(decimal amount);
public abstract decimal GetFineCredit();
```

When adding methods to JuniorMember and AdultMember, you should use **ctrl-dot** on the red squiggly to create the method signatures for you, then delete the **NotImplementedException** and replace it with the relevant code for the specific class.

JuniorMember:

```
private decimal CashFund { get; set; } // now private

public override void SetFineLimit(decimal amount)
{
    CashFund = amount;
}

public override decimal GetFineCredit()
{
    return CashFund;
}
```

AdultMember:

```
private decimal BankTransferAvailable { get; set; } // now private

public override void SetFineLimit(decimal amount)
{
    SetupBankTransferLimit(amount);
}

public override decimal GetFineCredit()
{
    return BankTransferAvailable;
}
```

All tests should pass.

## State Pattern

*If you don't have time:*

Then the moral of this section is 'Always ask the question – can a subtype morph into another subtype?'. For example, can a dog become a cat, keeping the original mammally bits? If the answer is 'No', as is the case with dogs and cats, then inheritance (as we've done so far in this lab) is fine.

But often one type can morph into another. In our case, a JuniorMember can become an AdultMember when they turn 16. Therefore, the statement should be:

*A LibraryMember has a MembershipType (and a Junior MembershipType is a type of MembershipType)*

rather than:

*A LibraryMember is a Junior Member*

*If you have time, then carry on:*

**Note:** If this section is difficult to appreciate, don't worry. It's something you should, in time, be aware of.

Unfortunately, even though the solution we've just developed looks really elegant as there are almost no 'if' statements in there and all the concerns are separated, there is still a problem.

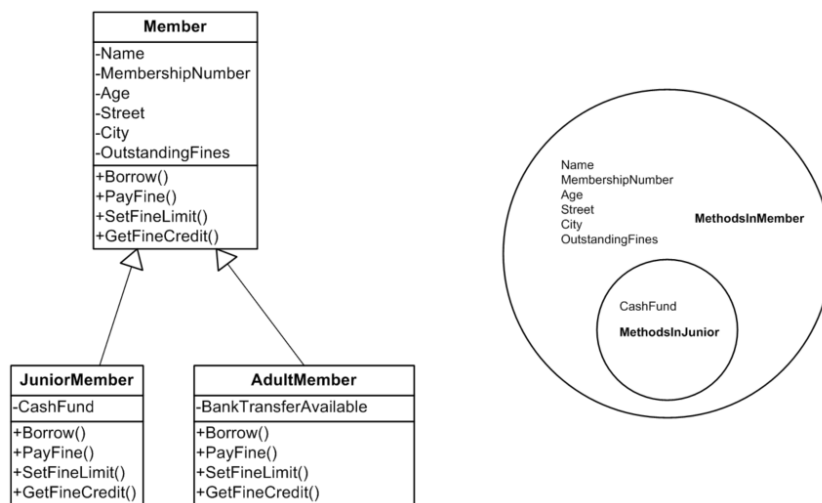
What happens when Greta turns 16?

Easy, you say – you just create her as an AdultMember

The rules of inheritance are that you cannot morph one type into another, so you have to destroy the original JuniorMember and create a brand new AdultMember. This presents some problems:

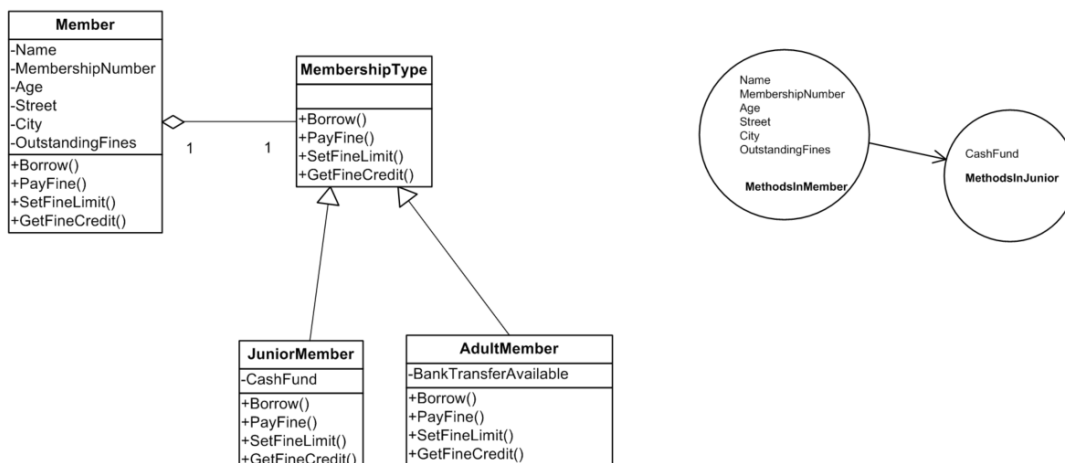
1. You have to transfer all the data (name, street, city, etc.) - what if some of that data is private?
2. Greta will feature in various collections. She would need to be removed from these and the new Greta would need to be inserted silently (probably breaking the rules we have established).

Our current class diagram and a representative object is shown below:



It's like we have one object built from two recipes – the **Member** recipe and the **JuniorMember** recipe. And when we destroy this object, we not only throw away the **CashFund** and **MethodsInJunior** (which is fine), we also throw away all the data and methods in the **Member** bit of the object.

What we need is the following. We need two objects such that we only throw away the fields and methods specific to being a Junior. For example, a **Member** has a **MembershipType** rather than a **Member** is a **Junior** from birth to death.



If you have plenty of time and feel very confident, then go ahead and have a go at changing your solution to be the **State Pattern**.

However, if it's been a long lab already, it's best to open the solution (**End2**), go to the **View** menu and select **Task List**. We have already made the required changes, and we have also annotated the few changes that were needed

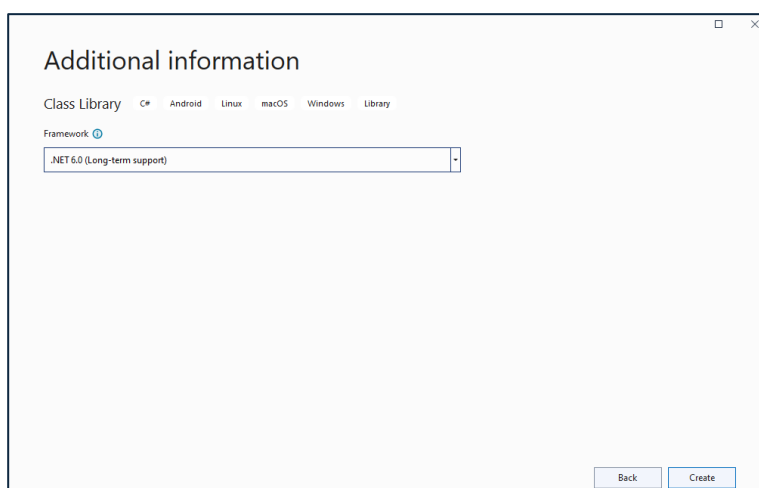
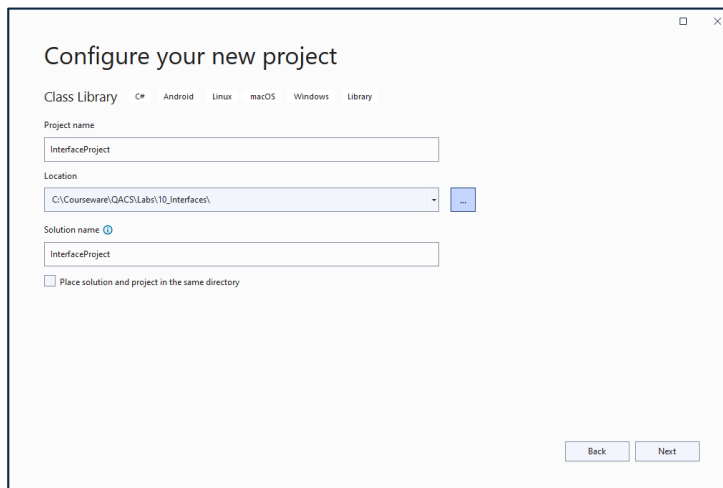
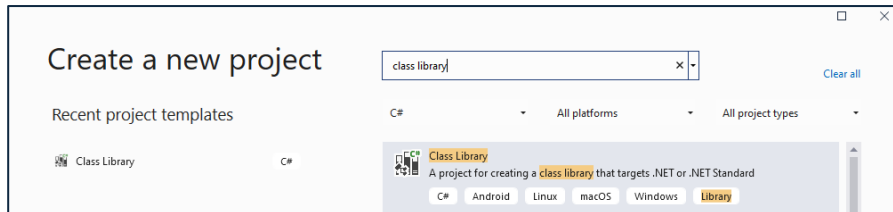
**Before moving on don't forget to commit and push your work to GitHub.**

## 04 Interfaces

The objective of this exercise is to consolidate your understanding of interfaces.

1

Locate the **'..\Labs\04\_Interfaces\Begin'** folder and, inside it, create a new Visual Studio project of type Class Library called **InterfaceProject**.



Delete **Class1**.

Add to this solution an XUnit Test project called **TestProject**.

|   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|---|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 2 | <p>We have provided a <b>Person</b> class in the Assets folder. Drag this file onto your <b>InterfaceProject</b>.</p> <pre>namespace InterfaceProject {     public class Person     {         public string Name { get; }         public string Address { get; }         public DateTime Dob { get; }          public Person(string name, string address, DateTime dob)         {             Name = name;             Address = address;             Dob = dob;         }     } }</pre> |
| 3 | <p>Replace the provided Test1() with the two tests in <b>IEquatable.txt</b> from the <b>'..\Labs\04_Interfaces\Assets'</b> folder and resolve the red squiggles.</p>                                                                                                                                                                                                                                                                                                                     |
| 4 | <p>Run the tests. The first test fails, the second test passes by accident.</p> <p>The problem is that the compiler doesn't know how to figure out if two Persons are the same person. For example, if your bank account has a balance of £100 and so does mine – does that mean they are the same account?</p>                                                                                                                                                                          |
| 5 | <p>Make Person implement the interface <b>IEquatable&lt;Person&gt;</b></p> <p>This will make you implement the <b>Equals</b> method.</p> <p>As far as we are concerned, if the Name, Address, and DateOfBirth are the same, it is the same person. Put in this code and ensure both tests now pass.</p>                                                                                                                                                                                  |

```
public bool Equals(Person? other)
{
    return (
        Name == other.Name &&
        Address == other.Address &&
        Dob == other.Dob
    );
}
```

- 6 You will now compare some people, the Spice Girls, by putting them in order based on their DateOfBirth.

We'll put these in DateOfBirth order.

Copy in the test in the Assets folder called **Compare\_People** and resolve the red squiggles.

```
[Fact]
public void Compare_People()
{
    List<Person> spiceGirls = new List<Person>() {
        new Person("Baby", "Finchley", dob:new DateTime(1976, 1, 21) ),
        new Person("Posh", "Harlow", dob:new DateTime(1974, 4, 17) ),
        new Person("Ginger", "Watford", dob:new DateTime(1972, 8, 6) ),
    };

    spiceGirls.Sort();
    Assert.Equal("Ginger", spiceGirls[0].Name);
    Assert.Equal("Posh", spiceGirls[1].Name);
    Assert.Equal("Baby", spiceGirls[2].Name);
}
```

Run the test. It will fail. Look at the error message:

|                                      |       |                                                                                   |
|--------------------------------------|-------|-----------------------------------------------------------------------------------|
| UnitTest1 (3)                        | 13 ms |                                                                                   |
| Compare_People                       | 3 ms  | System.InvalidOperationException : Failed to compare two elements in the array... |
| IEquatable_Are_These_The_Same        | 6 ms  |                                                                                   |
| IEquatable_Are_These_The_Same_People | 4 ms  |                                                                                   |

System.InvalidOperationException : Failed to compare two elements in the array.  
 ---- System.ArgumentException : At least one object must implement IComparable.

It fails because it does not know how to sort Persons into order because they are not comparable.

- 7 Implement the interface **IComparable<Person>**

Notice the **CompareTo** method returns an int.

We want to rank Person by **DateOfBirth**, so:

|   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|---|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|   | <p>CompareTo should return <b>+1</b> if DoB is greater than the compared to object's' DoB.</p> <p><b>-1</b> if smaller</p> <p>Otherwise return <b>0</b>.</p> <p>Enter the code and run the test to ensure it now passes.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 8 | <p>In the Assets folder is a class <b>AssassinatedPresident</b>. Drag this onto your <b>InterfaceProject</b>.</p> <p>Add the test <b>Compare_Presidents</b>.</p> <pre>[Fact] public void Compare_Presidents() {     List&lt;AssassinatedPresident&gt; assassinations = new List&lt;AssassinatedPresident&gt;() {         new AssassinatedPresident("Kennedy", "Dallas", dob:new DateTime(1917, 5, 29), assassinated:new DateTime(1963, 11, 22)),         new AssassinatedPresident("Lincoln", "Ford Theatre", dob:new DateTime(1809, 2, 12), assassinated:new DateTime(1865, 4, 15)),         new AssassinatedPresident("Perceval", "Houses of Parliament", dob:new DateTime(1762, 11, 1), assassinated: new DateTime(1812, 5, 11)),     };      assassinations.Sort();     Assert.Equal("Lincoln", assassinations[0].Name);     Assert.Equal("Perceval", assassinations[1].Name);     Assert.Equal("Kennedy", assassinations[2].Name); }</pre> <p>Now implement <b>Comparable</b> for <b>AssassinatedPresident</b> where we want the sort order to be by the <i>month</i> in which they were <i>assassinated</i>.</p> <p>Run the Test and confirm it passes.</p> |
| 9 | <p>You will now experiment with cloning objects.</p> <p>Copy in the <b>Clone_A_Spice_Girl</b> test.</p> <p>The interface you need is <b>ICloneable</b>. Make Person implement the interface <b>ICloneable</b>.</p> <p>Conveniently, there is a protected method on the Object class which does this for us:</p> <pre>public object Clone() {     return this.MemberwiseClone(); }</pre> <p>But you'll get a red squiggly in the test.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |



This is because **Clone** method returns an object, so we have to cast the result to **Person** in the test:

```
[Fact]
public void Clone_A_Spice_Girl()
{
    Person baby = new Person("Baby", "Finchley", dob: new DateTime(1976, 1, 21));
    Person babyClone = (Person)baby.Clone();
    Assert.Equal(baby, babyClone);
}
```

Re-run the test. It should now pass.

10 This works, but it's not ideal.

It's a pity that the framework does not offer a generic version of **ICloneable**.

Have a go at creating your own generic **ICloneable<T>** interface.

**Hint:** Do an F12 on the generic interfaces **IEquatable** and **IComparable** to see how they are created.

Implement your generic version of **ICloneable<T>** in **Person** instead of the non-generic **ICloneable**. You can perform the explicit cast in the implemented **Clone** method.

Your test should no longer require the explicit cast and should be as follows:

```
[Fact]
public void Clone_A_Spice_Girl()
{
    Person baby = new Person("Baby", "Finchley", dob: new DateTime(1976, 1, 21));
    Person babyClone = baby.Clone();
    Assert.Equal(baby, babyClone);
}
```

Run the Tests. All tests should now pass.

11 You will now explore another interface **ITaxRules** and see how it is implemented in different classes.

Into your **TestProject**, copy the **Tax Tests** from the **Assets** folder:

```
[Fact]
public void Evaluate_UK_Tax()
{
    Product product = new Product(50M, new UKTaxRules(true));
    Assert.Equal(18.75M, product.GetTotalTax());
}

[Fact]
public void Evaluate_US_Tax()
{
    Product product = new Product(50M, new USTaxRules());
    Assert.Equal(7.5M, product.GetTotalTax());
}
```

We have provided all the code for you.

Copy the **Product** folder from the **Assets** folder and drop it onto your **InterfaceProject**.

Have a look at the **Product** class. See that it is passed in an **ITaxRules** in its constructor but it doesn't know or care what sort of tax rules they, are as long as they implement **ITaxRules**.

Review the code and confirm all tests pass.

## Explicit interfaces

Because one class can implement many interfaces, it's possible that there could be a clash of method name. In the example coming up, we create an

**AmphibiousVehicle** (this is the DUKW – pronounced Duck), and it was actually the US Army coding for such a vehicle.

D=1942

U=Utility

K=all-wheel drive

W=2 powered rear axles

This remarkable vehicle is truly a water vehicle and truly a land vehicle. In other technologies we might describe this by using multiple inheritance. .NET outlaws multiple inheritance, so we would achieve this by implementing multiple interfaces.

|   |                                                                                                                                                                                                                                                                                                                                                             |
|---|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1 | <p>Drag the folder <b>DUKW</b> onto your <b>InterfaceProject</b> and have a look at the three files.</p> <p>In particular, notice that both interfaces have a <b>Brake</b> method.</p> <p>A land vehicle brakes by squeezing the disc pads.</p> <p>A water vehicle brakes by putting the propeller into reverse.</p> <p>Therefore, we need two methods.</p> |
| 2 | <p>Open the <b>AmphibiousVehicle</b> class and implement the interface <b>ILandVehicle</b>.</p> <p>Build the project to confirm there are no errors.</p> <p>You will see it gives you one method and both interfaces are satisfied.</p> <p>However, it gives you no chance to have two methods.</p> <p>Delete the <b>Brake</b> method.</p>                  |
| 3 | <p>Now implement the <b>IWaterVehicle</b> interface <i>explicitly</i> by placing the cursor on <b>IWaterVehicle</b> in <b>AmphibiousVehicle</b> and using <b>Ctrl+dot</b> -&gt; <b>Implement all members explicitly</b>.</p>                                                                                                                                |

|   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|---|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|   | <p>Delete the <b>throw NotImplementedException</b>.</p> <p>Now place the cursor on <b>ILandVehicle</b> and use <b>Ctrl+dot</b> -&gt; <b>Implement interface</b>.</p> <p>You get the Brake method.</p> <p>Delete the <b>throw NotImplementedException</b>.</p>                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 4 | <p>You will now use a test to call the implicitly implemented brake method and the explicitly implemented break method.</p> <p>Add this test to <b>TestProject</b>:</p> <pre>[Fact] public void Explicit_Interfaces() {     AmphibiousVehicle av = new AmphibiousVehicle();     av.Brake();     ((IWaterVehicle)av).Brake(); }</pre> <p>Add a breakpoint:</p> <pre>[Fact] public void Explicit_Interfaces() {     AmphibiousVehicle av = new AmphibiousVehicle();     av.Brake();     ((IWaterVehicle)av).Brake(); }</pre> <p>Right-click in the test and choose <b>Debug Tests</b>. Step into the code using and <b>Step Into (F11)</b> to confirm that both brake methods are called.</p> |

•

## Lab 05: SOLID Principles

### Objective

Your goal is to refactor code so that adheres to SOLID principles by implementing an inventory system for a bookstore.

### Overview

You are provided with a "Starter" program that manages an inventory system for a bookstore. The system allows adding books and CDs to the inventory and calculating the total stock value. The original code violates multiple SOLID principles. Your task is to refactor this code to make it more modular, maintainable, and extensible.

### GITHUB

Before starting on the lab please think seriously about using GitHub as a repository for the code.

### Steps

Open the StockValueCalculator solution in Visual Studio (located in Labs\05 SOLIDPrinciples\Begin). It contains two projects one called StockValueCalculator that contains functioning code that keeps track of a collection of products and their collective value and the other called StockValueCalculatorTests that hosts a single unit test that validates the functionality of the StockValueCalculator project.

Run the test to ensure the code functions correctly.

### Problems in the Starter Code:

**Single Responsibility Principle:** The Product class is managing products without distinction between types.

**Open/Closed Principle:** Adding a new product type (like magazines or DVDs) would require modifying existing methods and possibly the Product class.

**Dependency Inversion Principle:** There is a direct dependency on low-level module details (like product type checks) in the inventory management.

### Requirements

Refactor the Starter code so that it adheres to SOLID principles such that:

- Each product type (Book, CD) has its class that handles its specific attributes. (Single Responsibility Principle)
- The addition of new product types is straightforward and will not require changes to the existing code. (You can achieve this by creating a new class that implements an IProduct interface). (Open/Closed Principle)
- The Inventory class works with the IProduct interface, not concrete classes, which will decouple the code and make it more flexible. (Dependency Inversion Principle).

### GITHUB

**Before moving on don't forget to commit and push your work to GitHub.**

## Lab 06: Coding Design Patterns

### Exercise: Implement a Vehicle Management System using Design Patterns

#### Objective

Your goal is to implement the functionality described below using the specified design patterns

#### Overview

You are tasked with designing and implementing a vehicle management system in C#. This system will allow users to create, manage, and monitor different types of vehicles such as cars, lorries, and motorcycles. To ensure the application is scalable, maintainable, and well-organized, you should employ the following three design patterns: **Factory**, **Composite**, and **Observer**.

#### GITHUB

Before starting on the lab please think seriously about using GitHub as a repository for the code.

#### Steps

In this lab you are provided with a starter project that contains a set of relevant code files each of which contain sets of comments that suggest what functionality the classes/interfaces should provide.

Open the starter project (located in Labs\06 Design Patterns\Begin) and implement the following requirements:

#### Requirements

##### 1. Vehicle Creation (Factory Pattern)

- Implement a **VehicleFactory** class that creates vehicles like cars, lorries, and motorcycles. This factory will facilitate object creation and can be extended in the future to include more vehicle types without modifying the client code.
- Each vehicle should be derived from a common interface or abstract class, e.g., **IVehicle**.

##### 2. Managing Fleets of Vehicles (Composite Pattern)

- Use the Composite pattern to treat individual vehicles and groups of vehicles uniformly.
- Implement a **VehicleGroup** class that can contain individual vehicles or other groups of vehicles. This class should also implement the **IVehicle** interface.

##### 3. Monitoring Vehicle Status (Observer Pattern)

- Implement an Observer pattern where a **VehicleMonitor** class (which displays vehicle statuses) observes changes in the vehicles' properties or compositions (like adding or removing a vehicle in a **VehicleGroup** or starting or stopping a vehicle's engine).
- Whenever a vehicle's status is updated, the monitor should automatically update to reflect changes.

#### Steps to Complete

##### 2. Define IVehicle Interface

- Define common operations like **DisplayStatus**, **StartEngine**, and **StopEngine**.
- Include methods for adding and removing vehicles from groups.
- The interface should also define a property named **Owner** of type string

##### 3. Implement Concrete Vehicles

- Create classes like **Car**, **Lorry**, and **Motorcycle** that implement the **IVehicle** interface.
- For each class add a constructor that has a single string parameter. Use the parameter value passed to initialise the **Owner** property

#### 4. Create Vehicle Factory

- Implement the **VehicleFactory** with methods to create different vehicles based on input parameters, such as **CreateVehicle("Car")**.

#### 5. Implement VehicleGroup

- This class should implement **IVehicle** and contain a list of **IVehicle** objects. It should delegate calls to its contained vehicles (e.g., it displays its status by asking each contained vehicle to display its own status).
- In the **VehicleGroup** class add a constructor that accepts a string parameter, which is then used to initialise the **Owner** property.

#### 6. Implement VehicleMonitor and Observer Logic

- Create a **VehicleMonitor** class that is notified when any of its vehicles change (e.g. "Car engine started" or "Lorry engine stopped") or when the status of a vehicle within a group changes.
- Implement an interface like **IVehicleChangedObserver** with a method **Update** that **VehicleMonitor** will implement. Vehicles will notify the **VehicleMonitor** through this interface when they change.

#### 7. Test Your Application

- Write a simple main program to demonstrate creating vehicles via the factory, adding them to the monitor, grouping vehicles using **VehicleGroup**, and starting and stopping vehicle engines to see the monitor update.
- Alternatively, or if you have time, create a set of unit tests for your application.

## GITHUB

Before moving on don't forget to commit and push your work to GitHub.

## Enhancement (If you have time)

Try to enhance the Vehicle Management System by incorporating the Command Pattern. This pattern will provide a flexible and extendable way to encapsulate all details of operations performed on vehicles, such as starting or stopping engines, into command objects. This will also allow for easier tracking of operations (useful for undo/redo functionalities in more complex applications) and can organize the commands into a queue or a history log.

Steps you will need to take:

#### 1. Define a Command Interface.

- Give the interface a name of  **ICommand**.
- Get the interface to support **Execute** and **Undo** methods. Both methods should be void and take no parameters.

#### 2. Implement StartEngineCommand and StopEngineCommand classes.

- Create concrete command classes for starting and stopping the vehicle engines.
- Make the classes implement **ICommand**.
- Each class's constructor should be passed an **IVehicle** object that the **Execute** and **Undo** methods should use to invoke the **StartEngine** and **StopEngine** methods.

#### 3. Create a CommandInvoker class. Implement an invoker class that can execute commands and optionally manage a history of commands for undo operations. It is suggested you do this by

defining and instantiating a `Stack<ICommand>` collection within the class. Then implement the following methods:

- **ExecuteCommand** that takes an **ICommand** object as a parameter, invokes its **Execute** method and then pushes the command onto the Stack.
  - **UndoLastCommand** that takes an **ICommand** object as a parameter, checks to ensure the stack isn't empty and if not pops the last command off the stack and invokes its **Undo** method.
4. **Integrate the Commands into the Main Program.** Modify the main program to use commands for vehicle operations.
- Declare and instantiate a **CommandInvoker** object.
  - Create some **StartEngineCommand** and **StopEngineCommand** objects passing appropriate parameters to the constructor.
  - Call the invoker object's **ExecuteCommand** method a number of times passing in the **ICommand** objects you've just created.
  - Call the invoker object's **UndoLastCommand** method.
  - Check the programs output to ensure the code is working as expected.

## GITHUB

Before moving on don't forget to commit and push your work to GitHub.



## 07 JSON, CSV and Streams

### Working with JSON data

|   |                                                                                                                                                                                                          |
|---|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1 | Create a console app                                                                                                                                                                                     |
| 2 | Add a class to the project that is suitable for holding film information (film_id, title, synopsis, director, release_date ).                                                                            |
| 3 | Deserialize a JSON file containing films into a List<Film> object. Use the JSON listed below on the yellow background to create the file content. There's a file called Films.json in the Assets folder. |
| 4 | Display the film information on a console screen.                                                                                                                                                        |
| 5 | Add another film to the list.                                                                                                                                                                            |
| 6 | Serialize the list back out to the JSON file.                                                                                                                                                            |
| 7 | If you have time, duplicate the code but use dynamic / anonymous types rather than purpose-built classes.                                                                                                |

### Reading and Writing CSV files

|    |                                                                                                                                                                             |
|----|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 8  | Write code to read the CSV file "streaming_movies.csv" into a list using a custom class suitable for the data within each row. You will find the file in the Assets folder. |
| 9  | Filter the list to get just movies on the Netflix platform, and sort it by movie title.                                                                                     |
| 10 | Write the filtered list of movies to a new CSV file, but only including the columns: Title, ReleaseYear, and Revenue.                                                       |

### Streams

|    |                                                                                                                                                                                                                                                                                                                                          |
|----|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 11 | Open the Visual Studio solution called Streams.sln located in the Begin folder.                                                                                                                                                                                                                                                          |
| 12 | Review the code that already exists. You will notice there are functions called CompressIntoByteArray, DecompressByteArrayIntoString, EncryptString and DecryptString. Three of these functions contain ready written (and functioning code) but the DecompressByteArrayIntoString currently does nothing with the passed in byte array. |
| 13 | Using the CompressToByteArray function as a guide try to add appropriate code to the DecompressByteArrayIntoString to make the decompression process work.                                                                                                                                                                               |

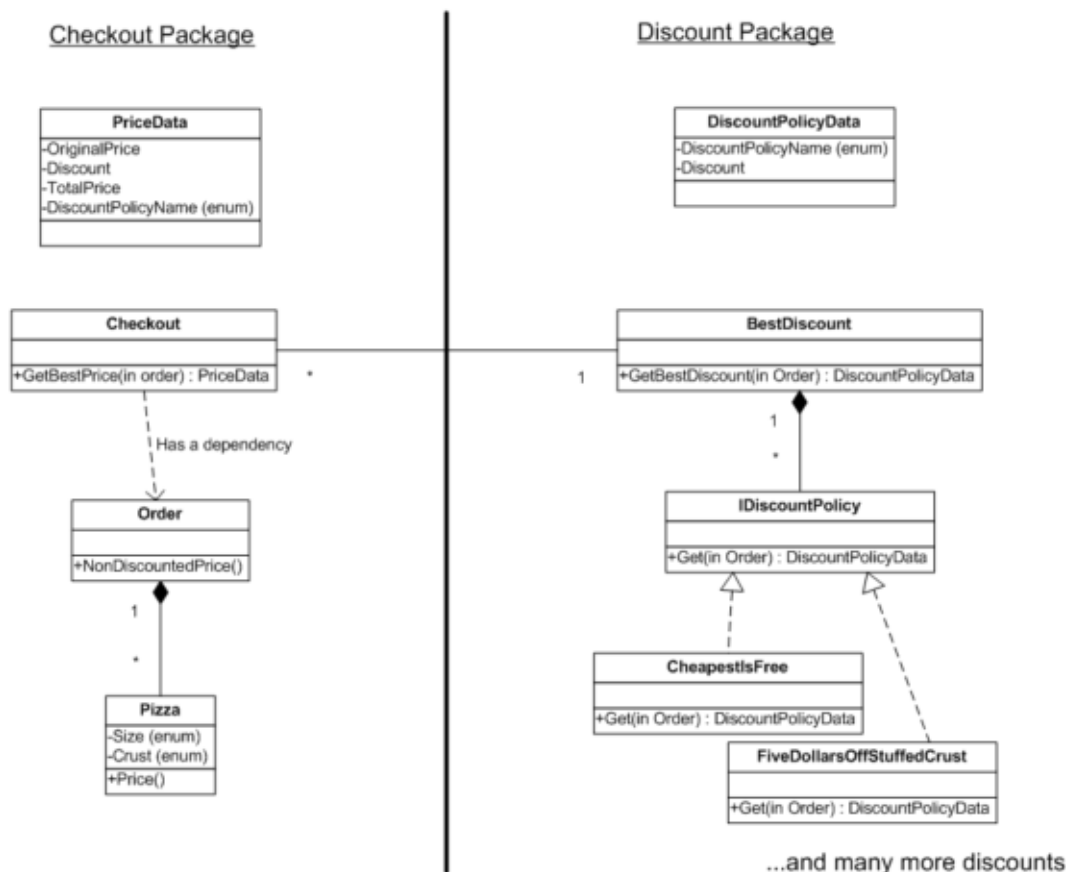
Before moving on don't forget to commit and push your work to GitHub.

## 08 Delegates and Lambdas

The objective of this exercise is to consolidate your understanding of delegates and lambdas.

### Scenario

We have the following classes and interfaces being used by a Pizza Ordering application.



An **Order** may contain one or many **Pizzas**.

**Checkout** needs to know the best discount to apply. We have a **BestDiscount** class to do this for us, which has a list of all available discounts. It passes in the order to each discount to see which one gets the best discount. It needs to return both the discount price and the name of the discount with said price. It does this by packaging the result into a **DiscountPolicyData** object and returning that to the Checkout.

The Checkout needs to know the price and name of the applied best discount, as well as what that discount was, the original price, hence the total to pay.

This exercise will take around 30 minutes.

|   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|---|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1 | Open the '..\Labs\08_Delegates_and_Lambdas\Pizza' project and compile it.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 2 | <p>To familiarise yourself with the problem, have a look at these classes:</p> <p><b>Pizza</b></p> <p>Notice that it has a Price property and it calculates the price from the size and the crust. To make it easier for you to follow the discounts, we have suffixed the size and crust with the price (so Small is actually Small_10 because its \$10). We wouldn't do this in real life because it would be hard to maintain if the prices change, but for our purposes, it makes the tests easier to understand.</p> <p><b>Order</b></p> <p>Order is a List of pizzas. Notice how it sums the prices of each pizza to get the non-discounted total.</p> <p><b>Checkout</b></p> <p>The GetBestPrice is passed the order (which is a List of pizzas). It passes this off to the bestDiscount object to work it out.</p> <p><b>BestDiscount</b></p> <p>Has a list of IDiscountPolicies. To get the best discount, it asks each discount policy what its discount would be for this order.</p> <p>Just as an aside here, notice how we have an abstract class BestDiscount that is subclassed by various strategies (Weekday, Weekend, etc.) where we can apply different rules in different circumstances.</p> <p><b>DiscountPolicyData</b></p> <p>In BestDiscount, it compares these to see which gives the best discount. We have to implement IComparable.</p> <p>Finally, the policies themselves e.g., <b>CheapestIsFree</b>.</p> <p>There is no discount here if the order consists only of one pizza.</p> <p>If it's more than one, it loops round and remembers the cheapest.</p> |

|   |                                                                                                                                                                                                                                               |
|---|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 3 | Now look at the tests in <b>CheckoutTests</b> . We've written a few tests that return the discount, and it's here where the <code>Small_10</code> , <code>Thin_4</code> etc. will help you see that it really has selected the best discount. |
| 4 | Run all tests and confirm they all pass.                                                                                                                                                                                                      |

As you can see, we can use Interfaces to achieve the decoupling we need – the `BestDiscount` class has no knowledge of the individual discount policies. It does have knowledge of `IDiscountPolicy` and all discount policies implement that.

But it is a bit clumsy:

For each discount policy, we must create a new class that implements `IDiscountPolicy`. That class has just one method which must be called 'Get'.

In this instance, we could have a neater syntax that doesn't care about the name of the discount policy method and just cares about the signature.

Look at the signature of the 'Get' method:

```
public interface IDiscountPolicy
{
    DiscountPolicyData Get(Order order);
}
```

We would describe this as a method that takes one **Order** as a parameter and returns something of type **DiscountPolicyData**.

We can use the **Func<T, TReturns>** delegate for this:

```
Func<Order, DiscountPolicyData> discountPolicy;
```

We refer to it as a generic delegate because you can provide any type. Note that, for `Func<>`, the last type is always the return type.

|   |                                                                                                                                                                             |
|---|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 5 | Delete the file <b>IDiscountPolicy.cs</b>                                                                                                                                   |
| 6 | In the Discounts folder, create a <b>public static class DiscountPolicies</b> . This will contain all our policies.<br><br>Change the namespace to just <b>PizzaProject</b> |

For each of the policies in the Policies folder, copy the **Get()** method and paste it into the class DiscountPolicies, replacing 'Get' with the name of the original class. Make the method static.

For example, the method **CheapestIsFree.Get** becomes:

```
public static class DiscountPolicies
{
    public static DiscountPolicyData CheapestIsFree(Order order)
    {
        System.Diagnostics.Debug.WriteLine("CheapestIsFree");
        var pizzas = order.Pizzas;
        if (pizzas.Count < 2)
        {
            return new DiscountPolicyData(DiscountPolicyName.None, 0M);
        }

        // Loop round all pizzas in the order and get the one with the minimum price
        decimal minPrice = decimal.MaxValue;
        foreach (Pizza pizza in pizzas)
        {
            if (pizza.Price < minPrice)
            {
                minPrice = pizza.Price;
            }
        }
        return new DiscountPolicyData(DiscountPolicyName.Cheapest_Is_Free, minPrice);
    }
}
```

Repeat for all discount policies.

7 Delete the folder **Policies** and the three files within it

8 If you compile now, it will fail because **BestDiscount** still relies on the now deleted **IDiscountPolicy**. Change this to:

```
protected List<Func<Order, DiscountPolicyData>> policies
{
    get; private set;
}
= new List<Func<Order, DiscountPolicyData>>();
```

9 Also, in BestDiscount, we now don't have classes like CheapestIsFree.

```
policies.Add(new CheapestIsFree());
```

But we do have methods that satisfy the signature of Func<Order,DiscountPolicyData>>

Change it to:

```
policies.Add(DiscountPolicies.CheapestIsFree);
```

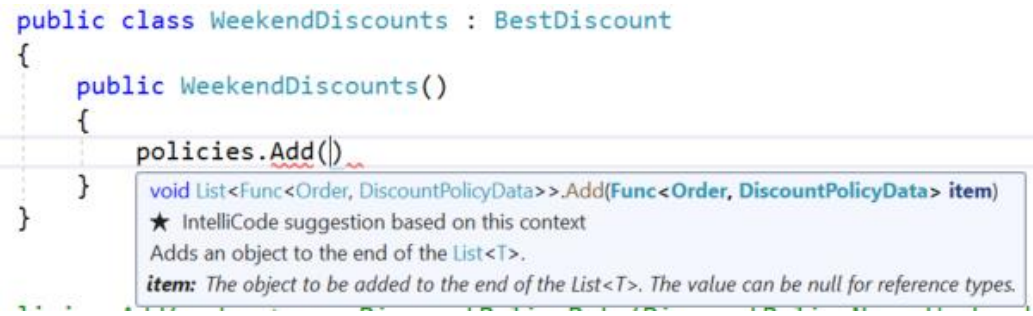
|    |                                                                                                                                                                                                                                                                                  |
|----|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 10 | <p>Repeat for the other two policies.</p> <p>To make the code even simpler, put in a using statement:</p> <pre>using static PizzaProject.DiscountPolicies;</pre>                                                                                                                 |
| 11 | <p>We are then left with one problem:</p> <pre>foreach (IDiscountPolicy policy in policies)</pre> <p>Resolve this as follows:</p> <pre>foreach (Func&lt;Order, DiscountPolicyData&gt; policy in policies) {     DiscountPolicyData thisOrdersPolicyData = policy(order); }</pre> |
|    | Run the tests. They should all pass.                                                                                                                                                                                                                                             |

Have a think about what you've just done.

You had an interface with one method and wherever you define such a method it must be packaged into a class and you will probably never re-use either the class or the method. You have replaced this with a pointer to a method of the required signature.

Now we have delegates in play, we can use lambda expressions.

|    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|----|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 12 | <p>Add this test to your <b>CheckoutTests</b>.</p> <pre>[Fact] public void Weekend_Ten_Percent_Off() {     checkout = new Checkout(new WeekendDiscounts());     order.Pizzas.Add(new Pizza(Size.Small_10, Crust.Regular_2));      PriceData priceData = checkout.GetBestPrice(order);      Assert.Equal(10.8M, priceData.TotalPrice);     Assert.Equal(DiscountPolicyName.Weekend_10_Percent_Off, priceData.DiscountPolicyName); }</pre> <p>We want a new discount policy to operate on weekends where you get 10% off.</p> |
|----|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|----|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 13 | <p>Add this to the <b>DiscountPolicyName</b> enum:</p> <pre>Weekend_10_Percent_Off</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 14 | <p>In the BestDiscount file, there is a class WeekendDiscounts.</p> <p>Add a constructor. Within the constructor type the code:</p> <pre>policies.Add(</pre> <p>Have a look at the IntelliSense:</p>  <pre>public class WeekendDiscounts : BestDiscount {     public WeekendDiscounts()     {         policies.Add()     } }</pre> <p>You will see it is expecting a parameter of type Func that takes an Order and returns a DiscountPolicyData.</p> <p>That means we can either point to a method of this signature (which is what we have done in the WeekdayDiscounts constructor) or we could put in a lambda expression.</p> |
| 15 | <p>Put in this lambda:</p> <pre>policies.Add(order=&gt; new     DiscountPolicyData(DiscountPolicyName.Weekend_10_Percent_Off,         order.NonDiscountedPrice * 0.1M) );</pre> <p>Run the <b>Weekend_Ten_Percent_Off</b> test and confirm it passes.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                             |

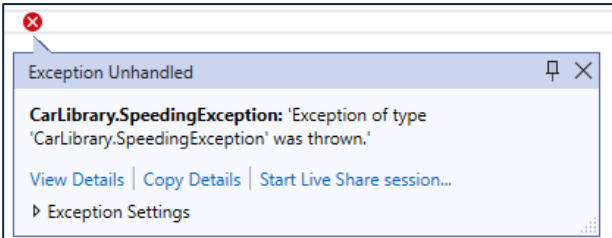
Before moving on don't forget to commit and push your work to GitHub.



## 09 Exception Handling

The objective of this exercise is to consolidate your understanding of exception handling and creating and throwing custom exceptions.

|   |                                                                                                                                                                                                                                                                                                                             |
|---|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1 | Open the <b>CarLibrary</b> solution in:<br><br><b>'..\Labs\09_Exception_Handling\Begin'</b>                                                                                                                                                                                                                                 |
| 2 | Comment out the existing code in <b>Program.cs</b>                                                                                                                                                                                                                                                                          |
| 3 | Open <b>Car.cs</b> and override the <b>ToString</b> method to display detailed car information:<br><br><pre>return \$"Car Make is {Make}, Model is {Model}, Colour is {Colour}, Speed is {Speed} MPH";</pre>                                                                                                                |
| 4 | Add a new auto-implemented property called <b>RoadSpeedLimit</b> .                                                                                                                                                                                                                                                          |
| 5 | You will change the logic of the Speed setter to account for the road's speed limit and whether or not the value that you are setting is a legal driving speed for the current road. <ul style="list-style-type: none"> <li>• If it is, set the value</li> <li>• If it is not, you will raise a custom exception</li> </ul> |
| 6 | In a separate file in the class library project, create an exception class called <b>SpeedingException</b> .<br><br>Don't forget to use inheritance.                                                                                                                                                                        |
| 7 | Within the new custom exception class, create an auto-implemented property called <b>ExcessSpeed</b> and set this value within the constructor.                                                                                                                                                                             |
| 8 | In <b>Car.cs</b> , if the car is not travelling at a legal speed, throw a new instance of <b>SpeedingException</b> , ensuring you pass in the excess speed value.                                                                                                                                                           |
| 9 | In <b>Program.cs</b> , create two new car instances: <b>slowCar</b> and <b>fastCar</b>                                                                                                                                                                                                                                      |

|    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|----|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 10 | <p>Set the following values for <b>slowCar</b>:</p> <pre>Car slowCar = new Car("Renault", "Clio"); slowCar.Colour = "Black";  slowCar.RegistrationNumber = "CLIO 1"; slowCar.RoadSpeedLimit = 30; slowCar.Speed = 30; Console.WriteLine(slowCar.ToString());</pre>                                                                                                                                                                                                                                                                                                     |
| 11 | <p>Set the following values for <b>fastCar</b>:</p> <pre>Car fastCar = new Car("BMW", "M5"); fastCar.Colour = "Silver";  fastCar.RegistrationNumber = "FAST 1"; fastCar.RoadSpeedLimit = 70; fastCar.Speed = 80; Console.WriteLine(fastCar.ToString());</pre>                                                                                                                                                                                                                                                                                                          |
| 12 | <p>Compile and run your application.</p> <p>You should see an unhandled exception:</p>                                                                                                                                                                                                                                                                                                                                                                                              |
| 13 | <p>Uncomment the existing code in <b>Program.cs</b>.</p> <p>Set the <b>RoadSpeedLimit</b> to <b>50</b> for Car <b>c2</b>.</p> <p>Wrap the code in this file with <b>try...catch...finally</b> blocks to handle the exceptions that are thrown.</p> <p>Add a catch block for <b>Exception</b> as well as for <b>SpeedingException</b>.</p> <p>Utilise the properties of the exception class to display useful messages to the console:</p> <pre>Console.WriteLine(\$"A speeding exception occurred. The car is travelling {ex.ExcessSpeed} MPH above the limit");</pre> |
| 14 | <p>Compile and run your application.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |

|    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|----|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|    | Observe the exceptions that are thrown and caught.                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 15 | <p>Add a property to store the <i>Car</i> instance within the <b>SpeedingException</b> and use this to access information that can be output to the console to help identify the Car that is speeding:</p> <pre> catch (SpeedingException ex) {     Console.WriteLine(\$"A speeding exception occurred. The car is travelling {ex.ExcessSpeed} MPH above the limit");     Console.WriteLine(\$"A speeding exception occurred. Car {ex.Car.RegistrationNumber} is travelling {ex.ExcessSpeed} MPH above the limit"); } </pre> |

### If you have time

|    |                                                                                                                                                        |
|----|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| 16 | Create a list of valid colours within <b>Car.cs</b> and create a custom <b>InvalidColourException</b> that is thrown if the colour is not in the list. |
| 17 | Observe how this exception is caught by the generic Exception event handler.                                                                           |
| 18 | Add a custom catch block to handle this specific type of exception.                                                                                    |
| 19 | A suggested solution is provided in the <b>End</b> folder for your reference.                                                                          |

**Before moving on don't forget to commit and push your work to GitHub.**

## Lab 10: A Quick Tour Around ASP.NET Core Web API MVC

### Objective

In this exercise we give you a quick tour around the fundamentals of ASP.NET Core Web API MVC.

### Overview

You start out by creating a basic ASP.NET MVC API Core project and then explore how to go about adding a controller and giving it a set of Actions. You will explore how C# method overloading causes issues that can be overcome by adding routing via `HttpGet` attributes. You will test the applications by using the built in Swagger capabilities and also take a look at using Postman as an alternative.

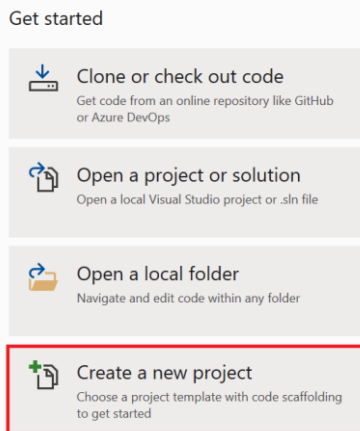
This exercise will take around 30 minutes.

### GITHUB

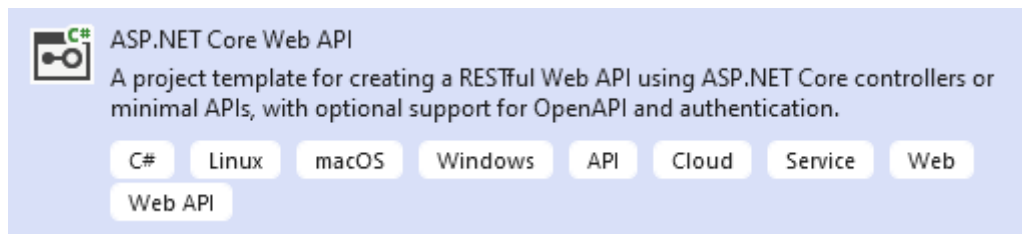
Before starting on the lab please think seriously about using GitHub as a repository for the code.

1 There is no starter for this project. Instead, we will create a new Web Application:

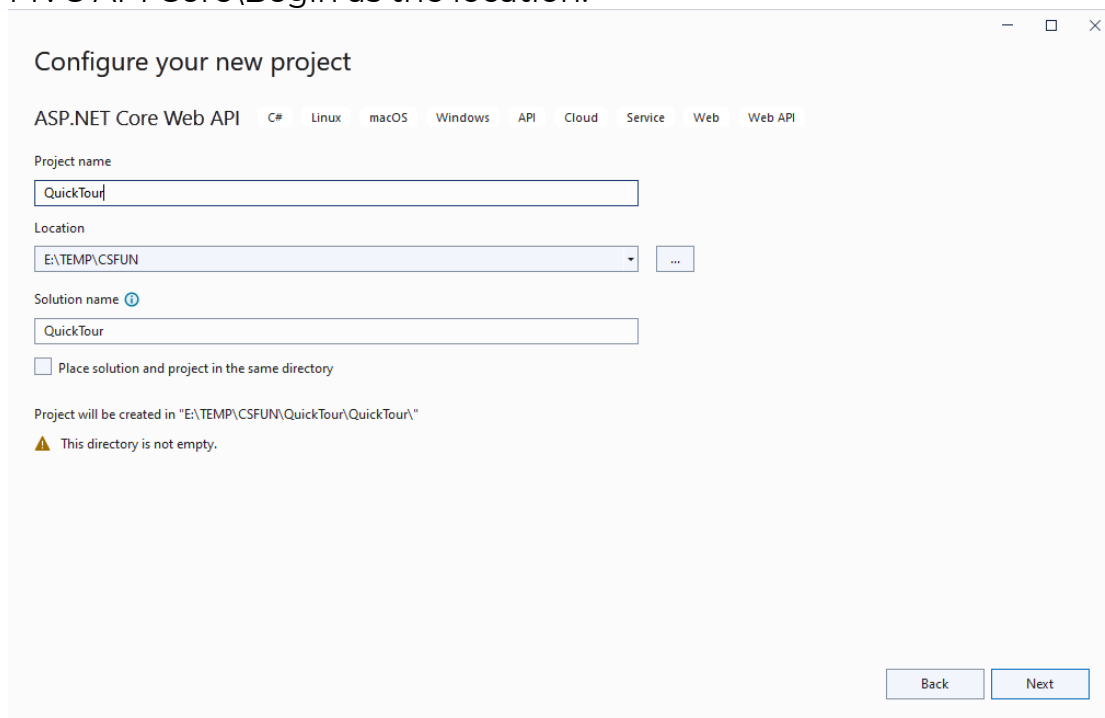
In Visual Studio, Select Create a new project



Search for “Web Core” and select ASP.NET Core Web API Application



Name the project **QuickTour** specifying Labs\10 Introduction to ASP.NET MVC API Core\Begin as the location:

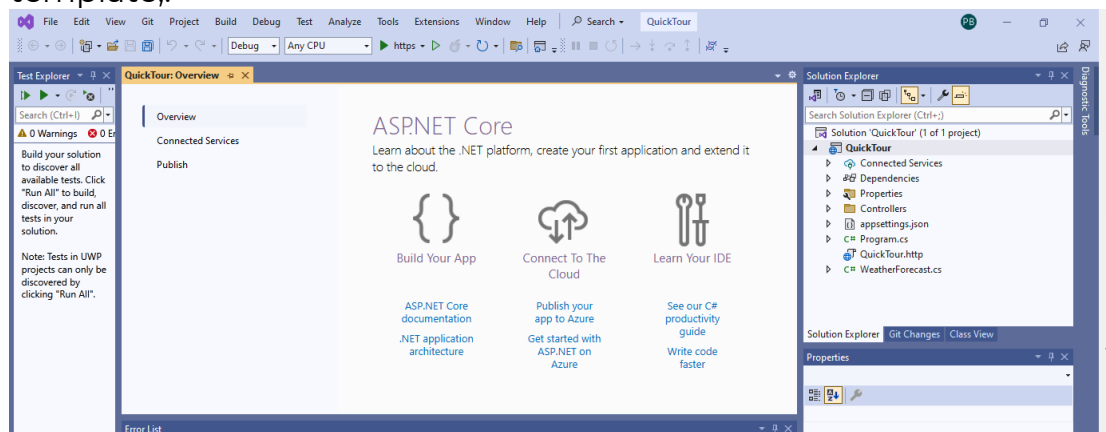


Select Next

Ensure the settings are as specified in the next screenshot and press Create

The screenshot shows the 'Additional information' dialog box in Visual Studio. The 'Framework' is set to '.NET 8.0 (Long Term Support)'. The 'Authentication type' is set to 'None'. The 'Configure for HTTPS' checkbox is checked. The 'Enable Docker' checkbox is unchecked. The 'Docker OS' is set to 'Linux'. The 'Enable OpenAPI support' checkbox is checked. The 'Do not use top-level statements' checkbox is unchecked. The 'Use controllers' checkbox is checked. At the bottom right, there are 'Back' and 'Create' buttons.

Visual Studio creates a new MVC API project, based on the default project template,.

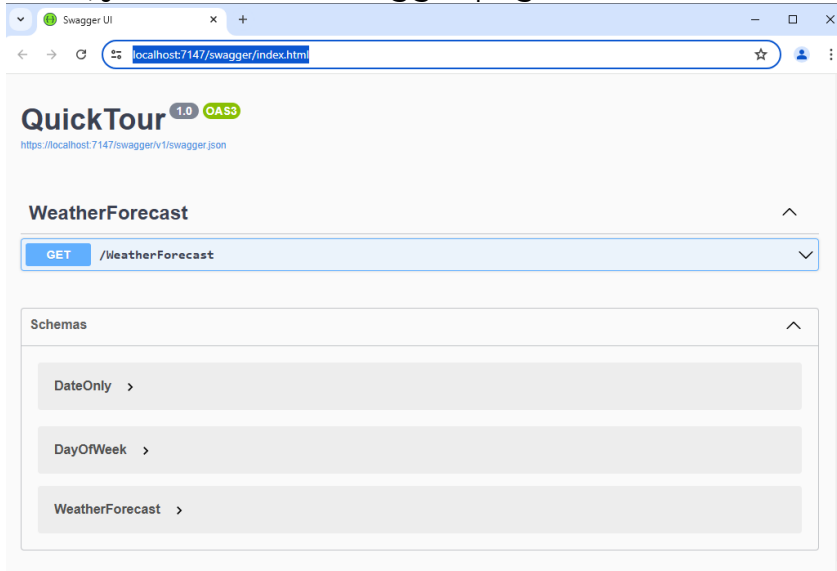


2

To make sure it's a runnable website, press F5.

You may be asked to accept a certificate the very first time you run an MVC application in development mode. If so, you should accept the certificate.

Then, you will see a Swagger page in a web browser:



Feel free to follow the prompts and invoke the WeatherForecast functionality. You should see something like the following.

**QuickTour 1.0 OA 19**  
<https://localhost:7147/swagger/v1/swagger.json>

### WeatherForecast

GET /WeatherForecast

Parameters

No parameters

Execute Clear

Responses

Curl

```
curl -X 'GET' \
  'https://localhost:7147/WeatherForecast' \
  -H 'accept: text/plain'
```

Request URL

```
https://localhost:7147/WeatherForecast
```

Server response

Code Details

200

```
{
  "data": "2024-07-16",
  "temperatureC": -10,
  "temperatureF": 10,
  "summary": "Chilly"
},
{
  "data": "2024-07-17",
  "temperatureC": 20,
  "temperatureF": 70,
  "summary": "Hot"
},
{
  "data": "2024-07-18",
  "temperatureC": 30,
  "temperatureF": 87,
  "summary": "Cool"
},
{
  "data": "2024-07-19",
  "temperatureC": 40,
  "temperatureF": 100,
  "summary": "Scorching"
},
{
  "data": "2024-07-20",
  "temperatureC": 50,
  "temperatureF": 120,
  "summary": "Hot"
}
```

Content-type: application/json; charset=utf-8  
date: Mon, 15 Jul 2024 15:00:45 GMT  
server: Kestrel

Response

Code Description Links

200 Success No links

Media type

text/plain

Content Accept Header

Example Value | Schema

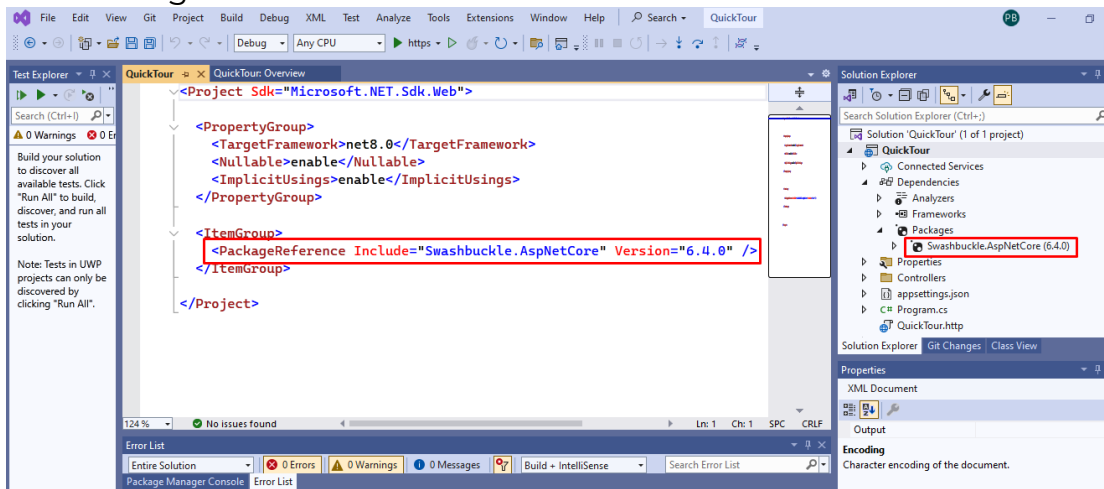
```
{
  "data": {
    "year": 0,
    "month": 0,
    "day": 0,
    "dayOfWeek": 0,
    "dayOfYear": 0,
    "dayNumber": 0
  },
  "temperatureC": 0,
  "temperatureF": 0,
  "summary": "string"
}
```

The data highlighted in the red box (above) shows some randomly generated data that is supposed to forecast what the weather will be like over the next 5 days.

Close the browser.



- 3 Expand the Dependencies > Packages folder. Also, right-click the project > Edit Project File. Note that the (meta) packages listed here match those in the Packages folder



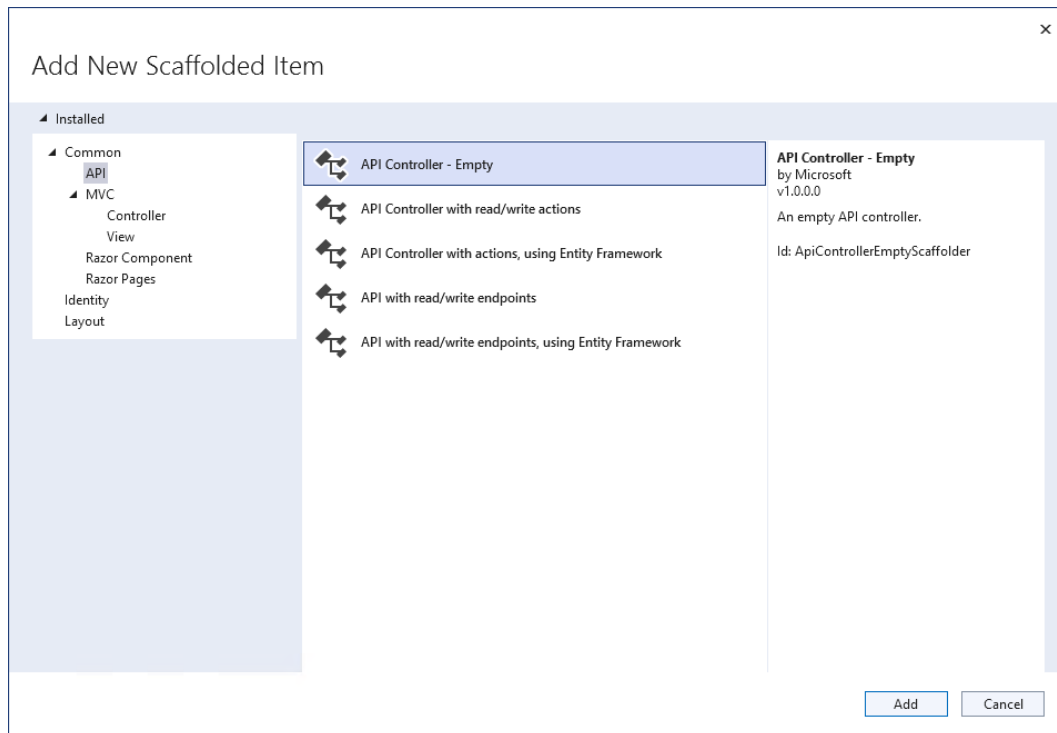
- 4 Rather than working with the existing "Weather" logic we will learn more about ASP.NET MVC API by adding additional code to the site. To get started we are going to need some data. To flesh it out a bit we will imagine we're building an on-line shop, so let's use that as the topic. Right-click on the QuickTour project in the Solution Explorer window and select Add | New Folder giving it the name Models. Add a C# class called Product.cs. Populate it like this:

```
public class Product
{
    public int ProductId { get; set; }
    public string Name { get; set; }
}
```

Note: we have added an Id because we would intend to store this in a database eventually.

The recommendation is to use `<className>Id` as it fits in with EntityFramework (discussed later) conventions somewhat better than just 'Id'

- 5 Right-click on the Controllers folder and select Add | Controller... Then, in the Add New Scaffold Item dialog box select MVC Controller – Empty and click Add.



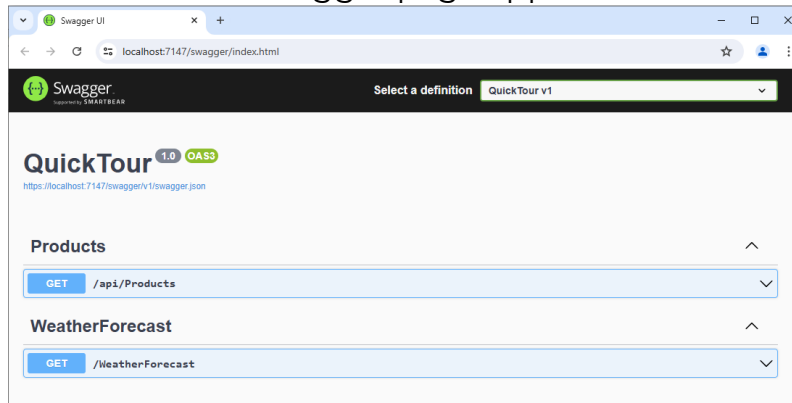
In the Add New Item dialog select the **API Controller – Empty** option.

Name the controller 'ProductsController' and press "Add".

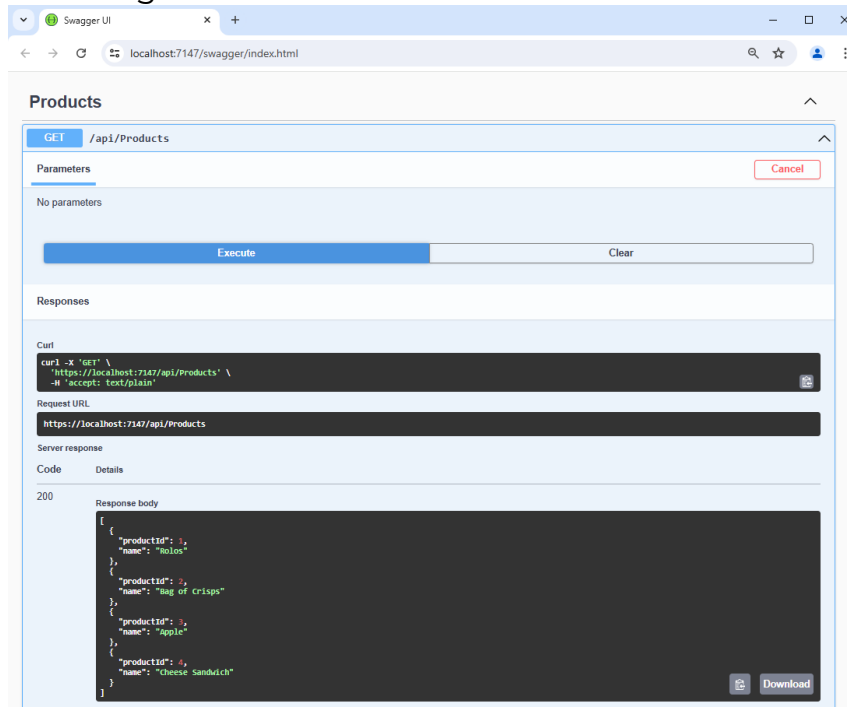
- 6 You should now see an empty class that is decorated with two attributes `[Route("api/[controller]")]` and `[ApiController]`.
- The `Route` attribute is specifying a template that dictates the part of the URL that directs the request to the controller. The `[controller]` section tells the run-time to replace it with the name of the controller (in this case `Products`) and would be analogous to `[Route("api/Products")]`. The benefit coming should the developer ever change the controller class name. If this attribute is omitted, then routing is based on method level routing.
- The `[ApiController]` attribute can be applied to controller classes to enable some API-specific behaviours such as making attribute routing a mandatory requirement (i.e. use of the `Route` attribute (see above).
- Add a new method to the class called `Products` that returns an `IEnumerable<Product>`.
- Decorate the method with an `HttpGet` attribute. Note this attribute isn't strictly necessary but Swagger uses it to determine how the method is to be used (Get, Post, Put, etc..) and will display an error if the attribute is missing.
- Public methods in a controller are called Actions – this is the `Products()` Action.
- Edit the method so it generates a number of `Product` objects adding them to a `List`. Then get the method to return the list:

```
[ApiController]
[Route("api/[controller]")]
public class ProductsController : ControllerBase
{
    [HttpGet]
    public IEnumerable<Product> Products()
    {
        List<Product> products = new List<Product>();
        products.Add(new Product { ProductId = 1, Name = "Rolos" });
        products.Add(new Product { ProductId = 2, Name = "Bag of
Crisps" });
        products.Add(new Product { ProductId = 3, Name = "Apple" });
        products.Add(new Product { ProductId = 4, Name = "Cheese
Sandwich" });
        return products;
    }
}
```

## 7 F5. Ensure the Swagger page appears



Test drive the Get `/api/Products` option by pressing the drop-down arrow and clicking the Try it out button and then press execute. You should see the following:

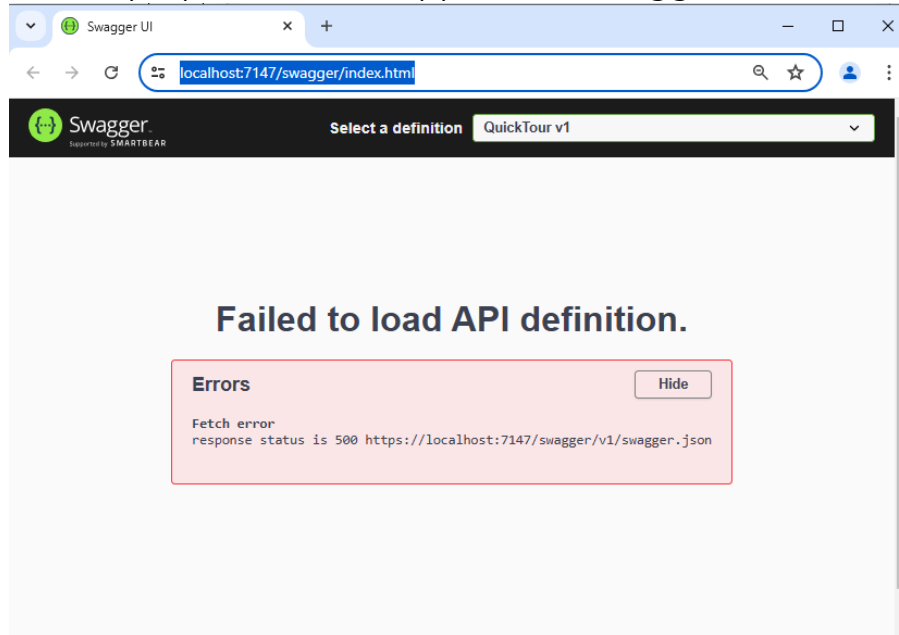


## 8 Let's add a second end-point (Action) to the controller. To keep things brief we'll get it to do the same thing as the Products method but return the list in alphabetical order of product name.

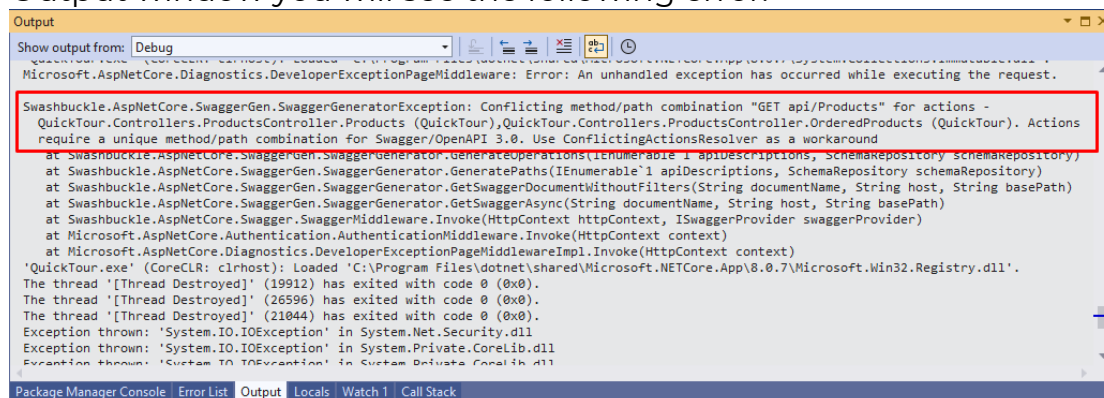
Add the following code to the Products controller:

```
[HttpGet]
public IEnumerable<Product> OrderedProducts()
{
    List<Product> products = new List<Product>();
    products.Add(new Product { ProductId = 1, Name = "Rolos" });
    products.Add(new Product { ProductId = 2, Name = "Bag of Crisps" });
    products.Add(new Product { ProductId = 3, Name = "Apple" });
    products.Add(new Product { ProductId = 4, Name = "Cheese Sandwich" });
    return products.OrderBy(p => p.Name).ToList();
}
```

9 F5 and prepare to be disappointed. Swagger will be unhappy:



10 The reason for the error is a little strange. If you dig into Visual Studio's Output window you will see the following error:



It's complaining about a conflicting method/path for `QuickTour.Controllers.ProductsController.Products` and `QuickTour.Controllers.ProductsController.OrderedProducts`

And further states Actions require a unique method/path combination for Swagger. You'd be forgiven for thinking that the two methods do have unique method/path combinations given their different names. However, it's not just Swagger that's complaining. If you were to call the method from an external client as a real API call, you'd get a similar error. Weirdly, in spite of the different method names, the runtime is upset because the signatures of the two methods are identical (i.e. neither take any parameters) and are therefore deemed to be the same!

- 11 The workaround is to extend the `HttpGet` attributes to specify an extension to the controller's template ("`api/[controller]`"):

```
[HttpGet("ProductsDetail")]
public IEnumerable<Product> ProductsDetail()
{
    List<Product> products = new List<Product>();
    products.Add(new Product { ProductId = 1, Name = "Rolos" });
    products.Add(new Product { ProductId = 2, Name = "Bag of Crisps" });
    products.Add(new Product { ProductId = 3, Name = "Apple" });
    products.Add(new Product { ProductId = 4, Name = "Cheese Sandwich" });
    return products;
}

[HttpGet("OrderedProducts")]
public IEnumerable<Product> OrderedProducts()
{
    List<Product> products = new List<Product>();
    products.Add(new Product { ProductId = 1, Name = "Rolos" });
    products.Add(new Product { ProductId = 2, Name = "Bag of Crisps" });
    products.Add(new Product { ProductId = 3, Name = "Apple" });
    products.Add(new Product { ProductId = 4, Name = "Cheese Sandwich" });
    return products.OrderBy(p => p.Name).ToList();
}
```

Note, it is perfectly OK to give the template the same value as the Action name. In the above example the two URL's needed to invoke the methods are:

<https://localhost:7147/api/Products/ProductsDetail>  
<https://localhost:7147/api/Products/OrderedProducts>

It would be perfectly OK to alter the attributes to the following:

```
[HttpGet("Products")]...
[HttpGet("Ordered")]...
```

And then the respective URL's would be:

<https://localhost:7147/api/Products/Products>  
<https://localhost:7147/api/Products/Ordered>

Test drive the app with both of the suggested changes and ensure everything works as expected.

12 Let's now add an additional Action that takes a parameter.

```
[HttpGet("Products/{id}")]  
public Product ProductsDetail(int id)  
{  
    Product p = new Product { ProductId = id, Name = "Rolos" };  
    return p;  
}
```

13 Notice we now have two overloaded methods (same name different signature which means the C# compiler will be happy and the new Action's `HttpGet` attribute starts off the same as its sister but is extended to include `"/{id}"`. The Squiggly braces indicate the URL will have a piece of data at its end (e.g. `api/Products/Products/12`) which will be passed on to the `int id` parameter specified in the method signature.

14 Run the code.  
When testing in Swagger you will be given the opportunity to enter a value for the id into a Parameters text box:

The screenshot shows the Swagger UI interface in a web browser. The URL bar indicates the endpoint is `localhost:7147/swagger/index.html`. The interface displays a GET endpoint for `/api/Products/Products/{id}`. Under the 'Parameters' tab, there is a table with columns 'Name' and 'Description'. A parameter named 'id' is listed, marked as 'required' and 'integer(\$int32) (path)'. A text input field next to 'id' is highlighted with a red box. Below the parameters, there is an 'Execute' button. The 'Responses' section shows a table with columns 'Code', 'Description', and 'Links'. A response with code '200' and description 'Success' is listed, with 'No links' shown. At the bottom, there is a 'Media type' dropdown menu set to 'text/plain'.

Obviously, the code, as it stands, will ignore the value and always return Rolos.

|    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|----|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 15 | <p>It is absolutely fine to decorate an Action with more than one route (<code>HttpGet</code> attribute]. Try decorating the appropriate Actions with the following additional routes. Think carefully which method should get which attribute and what the corresponding URLs would look like:</p> <pre>[HttpGet("")] [HttpGet("{id}")] [HttpGet("ProductDetail/{id}")]</pre>                                                                                                                                                                                                                                                                                                                                                                                               |
| 16 | <p>What do you think would happen if we removed the controller level <code>[Route("api/[controller]")]</code> attribute?</p> <p>Think about it, make a prediction and then launch the app to see if you were right.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 17 | <p>Add another Action with the same <code>ProductsDetail</code> name that takes the name of a product as a string parameter and returns an associated <code>Product</code> (again fake the search in the same way we did when passing the id).</p> <p>Decorate the new Action with the same <code>HttpGet</code> attributes as with the id approach but replace "id" with "name" wherever it occurs.</p>                                                                                                                                                                                                                                                                                                                                                                     |
| 18 | <p>The C# compiler should be happy because whilst there are now three methods that each have the same name it can distinguish between them because of the parameter types (no parameters, int and string).</p> <p>Launch the app and use Swagger to test the new Actions...</p> <p>Unfortunately, the method calls (both the ones that use the new string parameter and also the ones that used to work that used an int parameter) fail with the following message:</p> <pre>Microsoft.AspNetCore.Routing.Matching.AmbiguousMatchException: The request matched multiple endpoints.</pre> <p>Even though we've satisfied the C# compiler the ASP.NET Route selector logic can't distinguish between the two types of parameter, because they both appear to be strings!</p> |

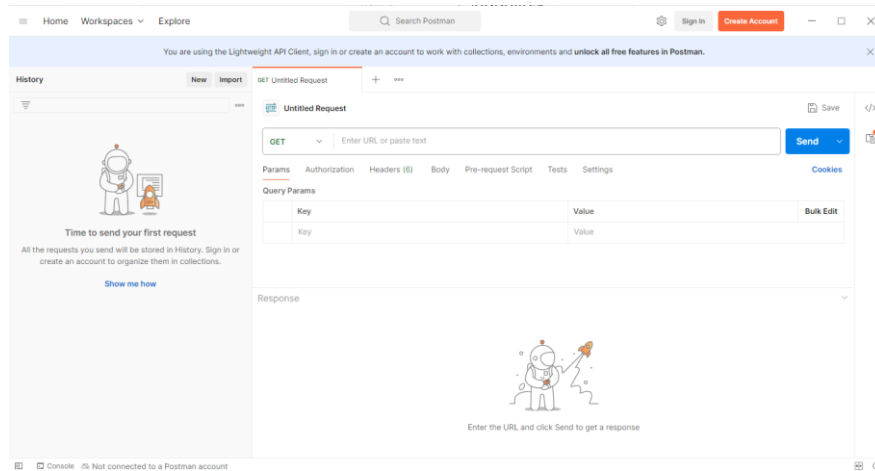


|    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|----|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 19 | <p>The solution to the problem is to spell out the parameter type inside the <code>HttpGet</code> routing template as follows:</p> <pre>[HttpGet("ProductDetail/{id:int}")] [HttpGet("Products/{id:int}")] [HttpGet("{id:int}")] public Product ProductsDetail(int id) {     Product p = new Product { ProductId = id, Name = "Rolos" };     return p; }</pre> <p>There's no need to do the same for the "name" parameters because the default is to treat them as strings.</p> |
| 20 | <p>An alternative to Swagger.</p> <p>Swagger is OK as an an-hoc way of testing your Actions but if you want a set of slightly more permanent tests that save you from having to continually type the same things into the various Swagger text boxes you should consider using a tool like Postman.</p>                                                                                                                                                                         |

21 Launch the app in Visual Studio.

Start Postman from the Windows Start menu.

You should see a window for an untitled GET request that is prompting for a URL.



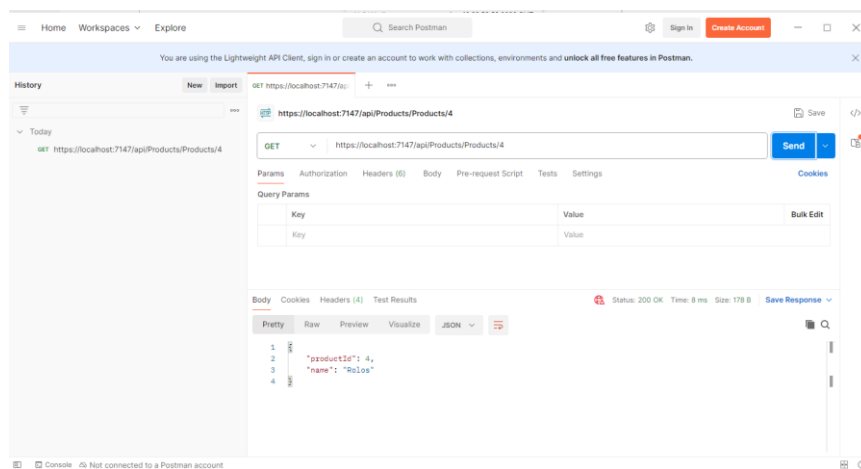
Enter the following as the URL:

<https://localhost:7147/api/Products/Products/4>

Press the blue Send button

Note you may receive an "SSL error: unable to verify the first certificate" warning message. It's OK to take the suggested option of disabling SSL verification.

Look at the response in the bottom half of the screen and confirm it is what you expect.



To create a new request, you can press the plus "+" button towards the top centre of the screen. You will also be able to create Post, Put and Delete requests by dropping the down arrow that is located to the right of the word GET (and to the left of the test URL).

Note, if you are prepared to register a set of credentials with Postman (it's free to do this) then you will be able to permanently save your requests (by pressing the Save button to the upper right of the screen).

It is definitely worth getting familiar with a tool like Postman it could transform your life!

## GITHUB

Before moving on don't forget to commit and push your work to GitHub.

### If You Have Time

|   |                                                                                                                         |
|---|-------------------------------------------------------------------------------------------------------------------------|
| 1 | Use Postman to create a set of test calls that exercise all the possible URLs that your Quick Tour project supports.    |
| 2 | Add additional Actions that take multiple parameters                                                                    |
| 3 | Rework the code in Actions that return a single Product so that it picks a matching value from a collection of Products |
| 4 | Refactor the code so it contains no duplicate logic                                                                     |

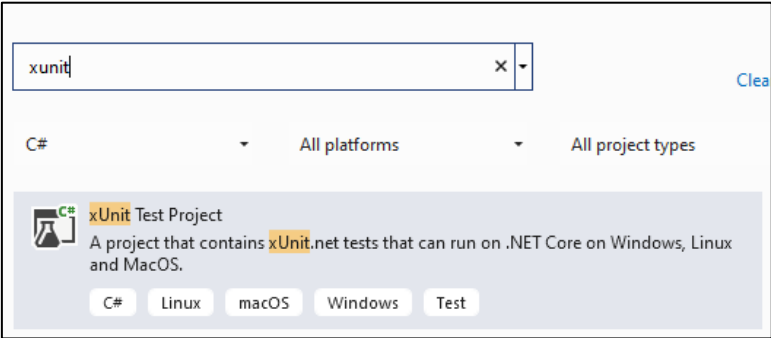
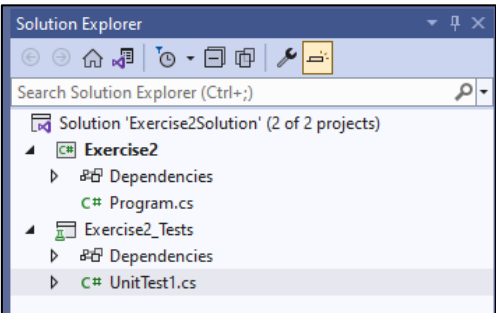
## GITHUB

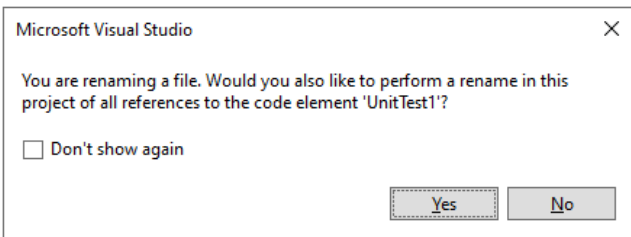
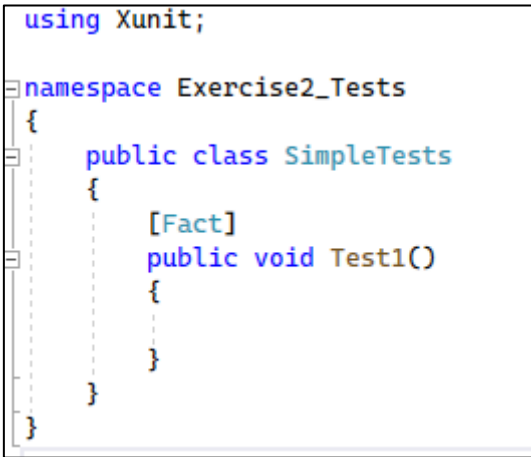
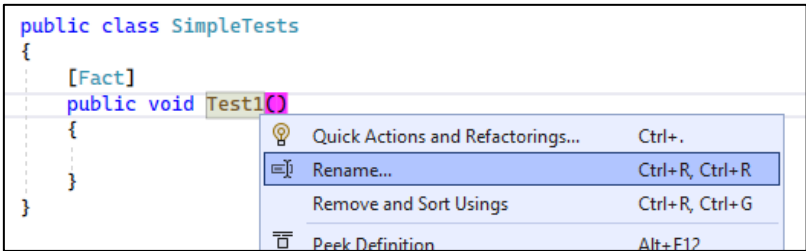
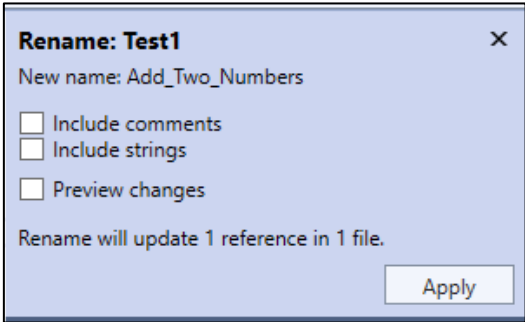
Before moving on don't forget to commit and push your work to GitHub.

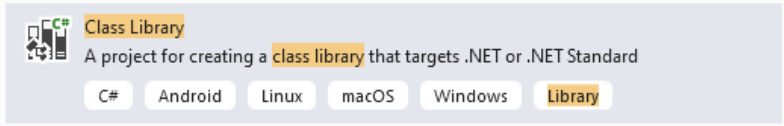
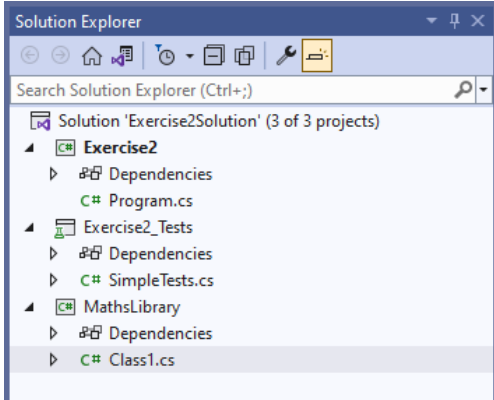
## 11 Unit Testing

The objective of this exercise is to consolidate your understanding of Unit Testing in C#.

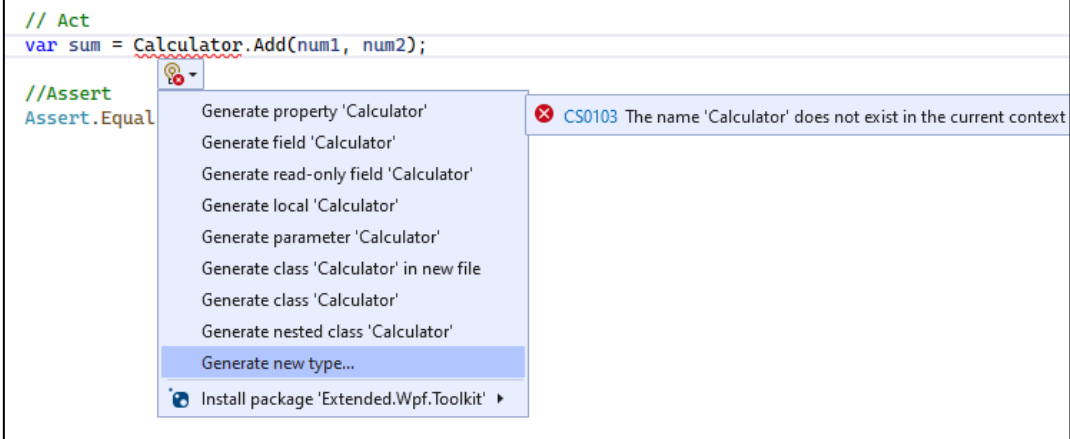
### Part 1 Some Basic Testing

|   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|---|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1 | <p>Create a new project in Visual Studio called <b>Exercise11Tests</b>.</p> <p>In the search box type <b>xunit</b>.</p>  <p>Select <b>xUnit Test Project</b> and click <b>Next</b>.</p> <p>Name the project <b>Exercise11_Tests</b> and locate it in Labs\11 Unit Testing\Begin.</p> <p>Click <b>Next</b>.</p> <p>Ensure <b>.NET 8.0 (Long-term support)</b> is selected and click <b>Create</b>.</p> |
| 2 | <p>Your <b>Solution Explorer</b> window now contains one solution with two projects:</p>                                                                                                                                                                                                                                                                                                            |
| 3 | <p>Rename <b>UnitTest1.cs</b> (by right-clicking on the file name) to <b>SimpleTests.cs</b> and select <b>YES</b> when the following prompt displays:</p>                                                                                                                                                                                                                                                                                                                              |

|   |                                                                                                                                                                                                                                                                                                                            |
|---|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|   |  <p>Microsoft Visual Studio</p> <p>You are renaming a file. Would you also like to perform a rename in this project of all references to the code element 'UnitTest1'?</p> <p><input type="checkbox"/> Don't show again</p> <p>Yes No</p> |
| 4 | <p>Your <b>SimpleTests.cs</b> file contains the following starter code:</p>                                                                                                                                                              |
| 5 | <p>Rename <b>Test1</b> to <b>Add_Two_Numbers</b>:</p>   <p>Click <b>Apply</b>.</p>                                                                  |

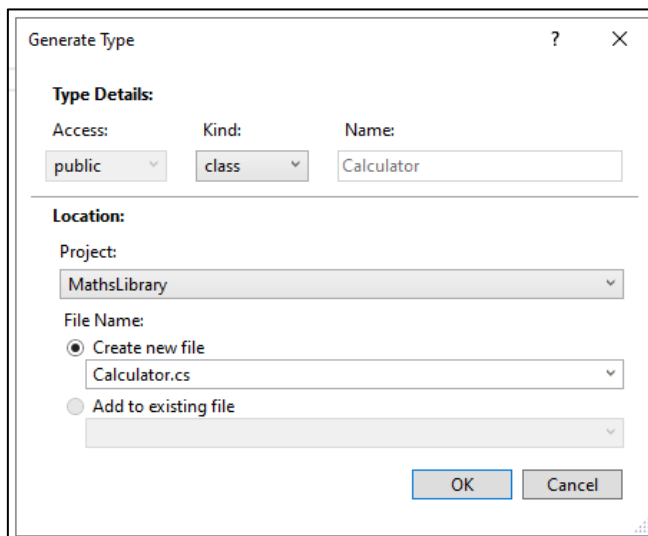
|   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|---|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|   | <pre> using Xunit;  namespace Exercise2_Tests {     public class SimpleTests     {         [Fact]         public void Add_Two_Numbers()         {         }     } } </pre>                                                                                                                                                                                                                                                                                            |
| 6 | <p>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.</p> <p>Add a new <b>class library</b> project to the solution called <b>MathsLibrary</b>.</p>  <p>Solution Explorer should now look as follows:</p>  |
| 7 | Delete the file <b>Class1.cs</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 8 | In <b>SimpleTests.cs</b> add the following code to <b>Add_Two_Numbers</b> :                                                                                                                                                                                                                                                                                                                                                                                           |

|    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|    | <pre>[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); }</pre>                                                                                                                                                                                                                                                                                                                                               |
| 9  | <p>This test code uses the standard testing pattern called the triple A pattern: <i>Arrange</i>, <i>Act</i>, <i>Assert</i>.</p> <p><i>Arrange</i> is for setting up items you need for the test.</p> <p><i>Act</i> is for carrying out the action you are testing.</p> <p><i>Assert</i> is for confirming the acted upon code behaves as expected.</p>                                                                                                                                                                                                                                    |
| 10 | <p>The arrange phase creates three variables: <b>num1</b>, <b>num2</b>, and <b>expectedValue</b>.</p> <p>The act phase calls an <b>Add</b> method on a <b>Calculator</b>, passing in <b>num1</b> and <b>num2</b> as parameters and assigning the result to a variable called <b>sum</b>.</p> <p>The assert phase checks whether the <b>expectedValue</b> and the <b>sum</b> values are equal.</p> <p>The Calculator type does not exist so you will use Visual Studio to help you create it.</p> <p>Press <b>Ctrl+.</b> (Ctrl+dot) on <b>Calculator</b> to see the available options:</p> |



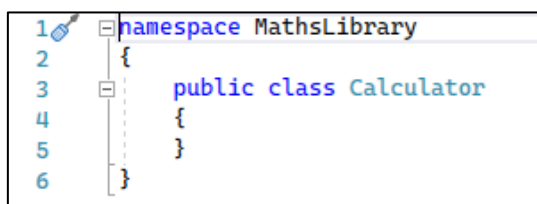
You want `Calculator` to be created in your **MathsLibrary** project rather than locally within the Test project so choose **'Generate new type...'**

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



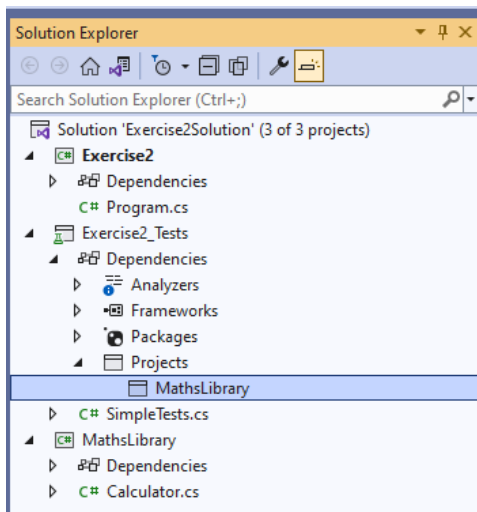
Click **OK**.

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

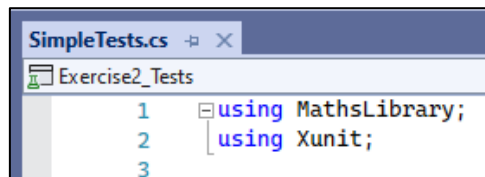




Visual Studio has also added a *reference* to the **MathsLibrary** project:



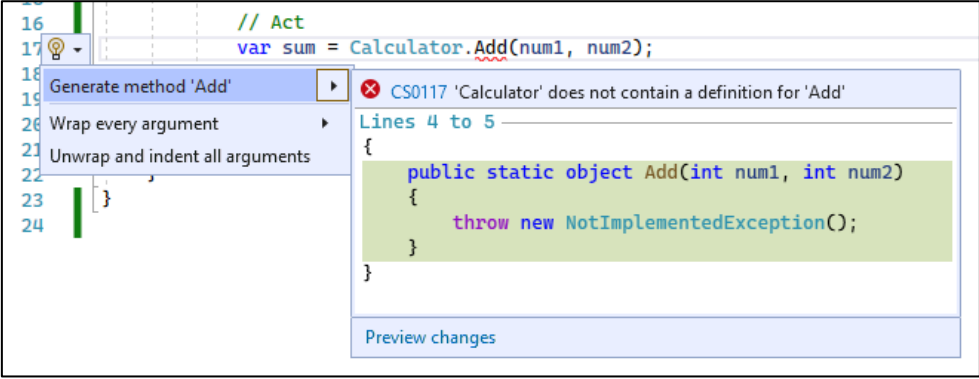
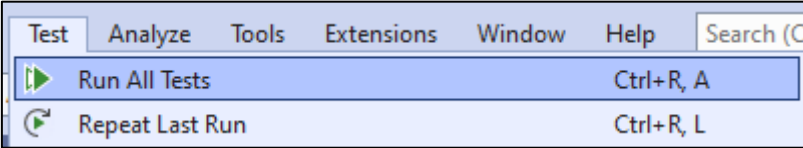
It has also imported the **MathsLibrary** *namespace* into your test project:

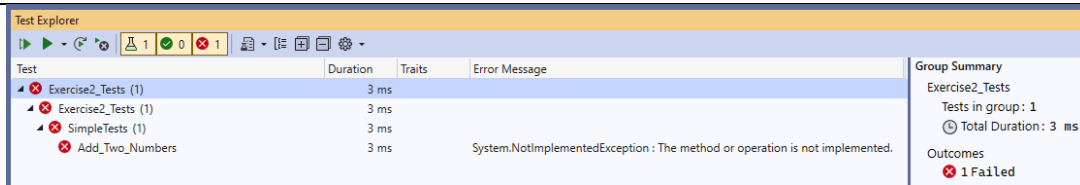


- 13 The act phase of the test code now recognises the **Calculator** type but displays an error because **Calculator** does not contain a definition for **Add**:

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

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

|    |                                                                                                                                                                                                                                                                         |
|----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|    |                                                                                                                                                                                       |
| 15 | <p><b>Calculator.cs</b> now contains an <b>Add</b> method:</p> <pre>public class Calculator {     public static object Add(int num1, int num2)     {         throw new NotImplementedException();     } }</pre>                                                         |
| 16 | <p>You want your <b>Add</b> method to return whole numbers so change the word <b>object</b> to <b>int</b>.</p> <pre>public static int Add(int num1, int num2) {     throw new NotImplementedException(); }</pre>                                                        |
| 17 | <p>You will run the test and observe the outcome.</p> <p><b>Test -&gt; Run All Tests.</b></p>                                                                                       |
| 18 | <p>Ensure the Test Explorer window is visible:</p> <p><b>Test -&gt; Test Explorer.</b></p> <p>Expand the Test until you see the <b>Add_Two_Numbers</b> failed test (it appears in red) alongside the error message: <b>The method operation is not implemented.</b></p> |



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

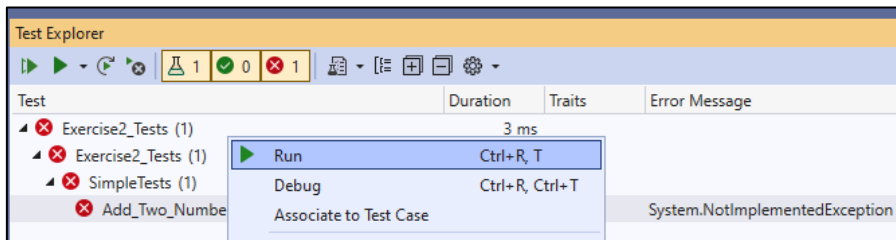
Delete the line of code: **throw new NotImplementedException;**

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

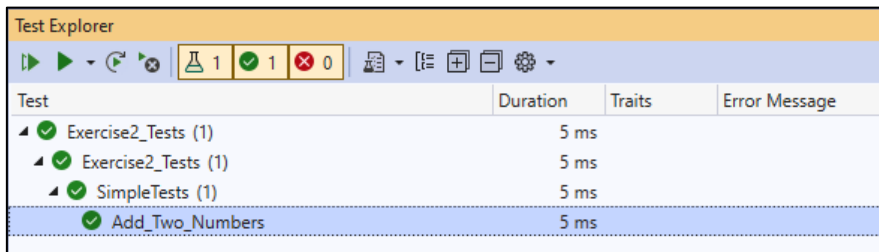
```
public static int Add(int num1, int num2)
{
    return 7;
}
```

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

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



The test should now pass:



- 21 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.

|    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|----|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|    | <p>Edit the <b>Add</b> method as follows:</p> <pre>public static int Add(int num1, int num2) {     return num1 + num2; }</pre>                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 22 | <p>Re-run the test to ensure it continues to pass.</p> <p>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.</p> |
| 23 | <p>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.</p>                                                                                                                                                                                                                                                                                                                                                                                     |
| 24 | <p>Solutions are provided in the <b>End</b> folder for your reference.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |

## Part 2: Using Mocks

|   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|---|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1 | <p>Open the <b>MovieSelector</b> solution in:</p> <p><b>‘..\Labs\11_Unit_Testing\Begin\MovieSelectorLab’</b></p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 2 | <p>Take some time to review the code that allows ready written project that allows users to see what movies are showing based on the current time and day.</p> <p>Note the <b>MovieDecider</b> class relies on two dependencies, a time service (<b>ITimeService</b>) that Gets the <b>TimeOfDay</b> as a <b>TimeSpan</b> object and Database context object that implements <b>IMoviesContext</b>. Instances of both of these objects need to be passed to the <b>MovieDecider</b> class's constructor. We need create tests that will serve up movies that reflect a specified time of day (no matter what the actual current time is). We also don't want to be accessing the underlying database when running our tests.</p> |

|    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|----|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 3  | Add a new <b>xUnit</b> project to the solution called <b>MovieSelectorTestProject</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 4  | <p>Your job now is to create a set of unit tests that ensure the correct movies are served up at different times of day and on different days (based on the number of days on from the current date). A look at the <b>Program.cs</b> file will give you an idea of What is going on.</p> <p><b>Note:</b> The <b>MovieDecider</b> class's constructor is overloaded and one version takes <b>ITimeService</b> and <b>IMoviesContext</b> objects as its parameters. This means the class has 2 dependencies both of which are based on interfaces which could be faked.</p> |
| 5  | Rename the <b>UnitTest1.cs</b> file as <b>MovieDeciderTests.cs</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 6  | The assets folder contains a file called <b>MovieCollectionMaker.txt</b> . Open this now and copy the code it contains into the <b>MovieDeciderTests</b> class. Note it creates a collection of <b>Movie</b> objects called movies.                                                                                                                                                                                                                                                                                                                                        |
| 7  | Use <b>NuGet</b> to install the <b>FakeItEasy</b> package.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 8  | Add a new Test method called " <b>MorningMoviesTest</b> ". We want this test to ensure that only movies that are running in the morning are returned.                                                                                                                                                                                                                                                                                                                                                                                                                      |
| 9  | <p>Declare an <b>ITimeService</b> variable called <b>timeService</b>. Make it equal to:</p> <pre>A.Fake&lt;ITimeService&gt;();</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| 10 | <p>Add code such that the fake service anticipates a call to the <b>timeService</b>'s <b>GetTimeOfDay</b> method and returns a new <b>TimeSpan</b> of 9:30 am:</p> <pre>A.CallTo(() =&gt; timeService.GetTimeOfDay())     .Returns(new TimeSpan(9, 30, 0));</pre>                                                                                                                                                                                                                                                                                                          |
| 11 | <p>Declare an <b>IMoviesContext</b> object called <b>moviesContext</b> and make it equal to:</p> <pre>A.Fake&lt;IMoviesContext&gt;();</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                |

|    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|----|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 12 | Add code such that the fake service anticipates a call to the <code>moviesContext</code> object's <code>GetMovies</code> method and returns the <code>movies</code> collection:                                                                                                                                                                                                                                                                                                                                                                              |
| 13 | Create a <code>MovieDecider</code> object called <code>movieDecider</code> passing the <code>timeService</code> and <code>moviesContext</code> objects to its constructor.                                                                                                                                                                                                                                                                                                                                                                                   |
| 14 | Declare a <code>List&lt;Movie&gt;</code> variable called <code>filteredMovies</code> and make it equal to the result of calling <code>movieDecider.FilterMovies(1)</code> (where the "1" indicates to return the movies that are happening tomorrow.                                                                                                                                                                                                                                                                                                         |
| 15 | Create some <code>Assert</code> statements that ensure the <code>filteredMovie</code> collection contains the expected data: <ul style="list-style-type: none"><li>• The filtered collection should contain a count of 2 movies</li><li>• The first <code>Movie</code> in the collection should return "Alien is showing at 11:00" when it's <code>ToString()</code> method is invoked.</li><li>• The second <code>Movie</code> in the collection should return "Skyfall is showing at 10:06" when it's <code>ToString()</code> method is invoked.</li></ul> |
| 16 | Add some additional test methods that ensure afternoon, evening and night movies are being catered for correctly.                                                                                                                                                                                                                                                                                                                                                                                                                                            |

#### If you have time

|    |                                                                                                                                                                                                                                                                                                                        |
|----|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 17 | Create a set of additional tests that ensure the <code>TimeOfDayDecider</code> class's <code>GetTimeOfDay</code> method is working correctly. It is meant to return "morning", "afternoon", "evening" or "night" depending on the time of day. Create tests that ensure each of the words is being correctly returned. |
|----|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

**Before moving on don't forget to commit and push your work to GitHub.**



## APPENDIX

A set of additional labs that may be of interest

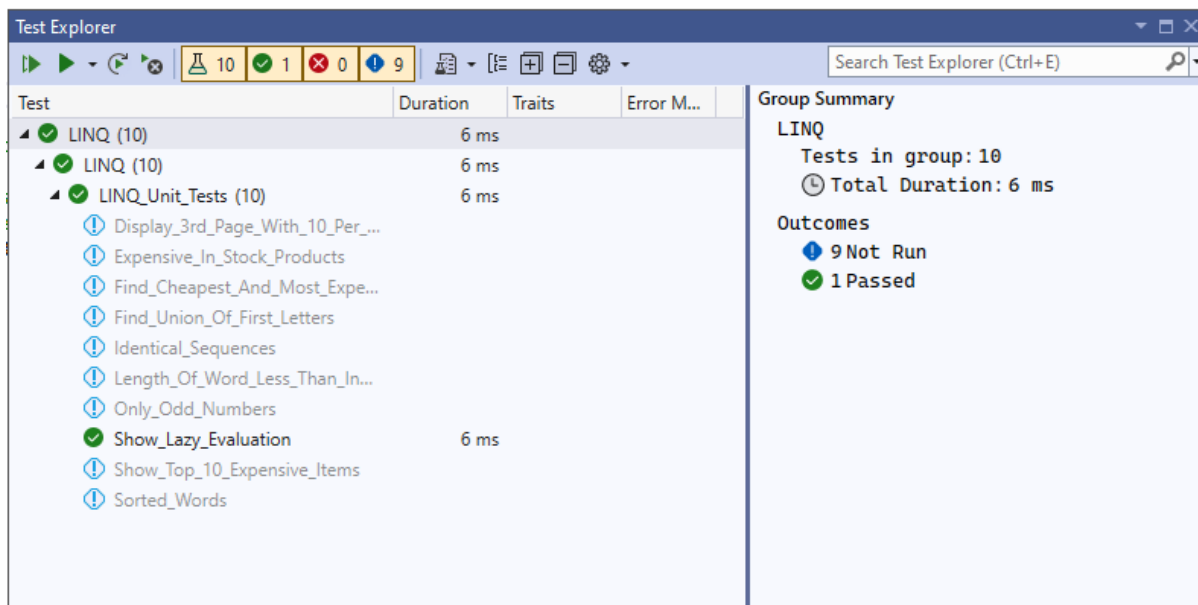


## 02b REVIEW - OPTIONAL Language Integrated Query (LINQ)

The objective of this exercise is to give you a wide view of the capabilities of Language Integrated Query (LINQ).

### Writing LINQ Queries

1. Open ‘..\Labs\07\_LINQ\Begin\LINQ\_Solution.sln’
2. Open the **LINQ\_Unit\_Tests.cs** file and read the instructions at the top of the file.
3. Each test shows an imperative way of solving a problem. At the end of each test are clues as to how you might get started solving the same problem with LINQ. Have a go solving the problem with LINQ.
4. After writing the LINQ equivalent code, run each unit test by right-clicking within the unit test and choosing Run Tests. Ensure each test still passes.
5. As you progress, do a visual comparison of how much briefer the declarative LINQ solutions are compared to the provided non-LINQ solutions.



**Before moving on don't forget to commit and push your work to GitHub.**

## 02c REVIEW - OPTIONAL Properties and Constructors

The objective of this exercise is to consolidate your understanding of C# properties and constructors.

### The Car Library and Console Projects

|   |                                                                                                                                                                                                                                                                                                                                                     |
|---|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1 | Create a new Class Library project called <b>CarLibrary</b> .<br><br>Rename <b>Class1.cs</b> to <b>Car.cs</b>                                                                                                                                                                                                                                       |
| 2 | Add a Console Application project to the Solution called <b>CarConsole</b> .<br><br>Set <b>CarConsole</b> as the start-up project.<br><br>In <b>CarConsole</b> , add a project reference to the <b>CarLibrary</b> project.                                                                                                                          |
| 3 | In <b>Car.cs</b> , create a property of type <i>int</i> called <b>Speed</b> with a <i>backing field</i> <b>speed</b> .<br><br>Validate that the speed set is above zero but under 100.                                                                                                                                                              |
| 4 | Add an auto-implemented property of type <i>string</i> called <b>RegistrationNumber</b> .                                                                                                                                                                                                                                                           |
| 5 | Add a calculated expression bodied property called <b>SpeedInKilometres</b> of type <i>double</i> .<br><br>To calculate the speed in kilometres, multiply the speed by 1.609344                                                                                                                                                                     |
| 6 | Add string properties for <b>Make</b> , <b>Model</b> , and <b>Colour</b> .                                                                                                                                                                                                                                                                          |
| 7 | In <b>CarConsole</b> , in <b>Program.cs</b> :<br><br>Delete the line of code that outputs 'Hello, World!'<br><br>Instantiate a car object, <b>c1</b> .<br><br>Issue a using directive to bring the <b>CarLibrary</b> namespace into scope.<br><br>Write the name of the instance to the console:<br><br><code>Console.WriteLine(nameof(c1));</code> |

|    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|----|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|    | <p>Build and run the console application to confirm the object can be successfully instantiated.</p> <p>Set the make of <b>c1</b> to be <b>'Ford'</b>.</p> <p>Write the <i>make</i> of c1 to the console.</p> <p>Write the <i>model</i> of c1 to the console.</p> <p>What value is displayed?</p>                                                                                                                                                                                             |
| 8  | <p>In the <b>Car</b> class, create a constructor that accepts a <i>make</i> and a <i>model</i> only.</p> <p>Initialise these values within the constructor.</p>                                                                                                                                                                                                                                                                                                                               |
| 9  | <p>In <b>CarConsole</b>:</p> <p>Re-run the app. Does it build successfully?</p>                                                                                                                                                                                                                                                                                                                                                                                                               |
| 10 | <p>Create a <i>parameterless</i> constructor</p> <p>Set the make and model to be <b>Unknown</b> and the colour to be <b>Black</b>.</p> <p>Confirm the console app builds and runs successfully.</p> <p>What value is displayed for the model?</p>                                                                                                                                                                                                                                             |
| 11 | <p>In <b>CarConsole</b>:</p> <p>Instantiate a car object, <b>c2</b>, using the overloaded constructor. The make is <b>Audi</b>, the model is <b>TT</b>;</p> <p>Write the make and model of <b>c2</b> to the console.</p> <p>Set the colour property to <b>Red</b>.</p> <p>Write c2's colour property to the console.</p> <p>Set the speed of <b>c2</b> to <b>30 miles per hour</b>.</p> <p>Display the speed in the console in both <i>miles per hour</i> and <i>kilometres per hour</i>.</p> |

|    |                                                                                                                                                                                                                                                                                                                                                                      |
|----|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 12 | <p>In <b>CarConsole</b>:</p> <p>Instantiate a car object, <b>c3</b>, using the overloaded constructor (<b>BMW, X5</b>) and an object initialiser that sets the colour to <b>Grey</b> and the registration number to <b>ABC 123</b>.</p> <p>Write the property values of <b>c3</b> to the console.</p>                                                                |
| 13 | <p>In <b>Car.cs</b>, chain the parameterless constructor to the overloaded constructor, passing <b>Unknown Make</b> and <b>Unknown Model</b> as the parameters.</p> <p>In the body of the parameterless constructor, remove the make and model and set the colour to be <b>White</b>.</p> <p>Confirm the console application still builds and runs successfully.</p> |
| 14 | <p>In <b>CarConsole</b>:</p> <p>Instantiate a car object, <b>c4</b>, using the parameterless constructor.</p> <p>Write the property values of <b>c4</b> to the console.</p> <p>Confirm <b>c4</b> is an unknown make and model that is white with an empty registration number.</p>                                                                                   |

### If You Have Time - Coding Challenge: Enhance the Finance ClientExe Projects

|   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|---|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1 | <p>Either return to your solution to the previous exercise (07 OOP Concepts) or open the one in the Begin folder.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| 2 | <p>Rework the Account class so:</p> <ul style="list-style-type: none"> <li>The holder field is private with access being given to it via a public property called Holder. The property should be both Gettable and Settable with the following validation being applied: <ul style="list-style-type: none"> <li>If the passed in value is null the value of holder should be set to "*** Anon ***".</li> <li>If the passed in value is <b>not</b> more than 2 character the value of holder should be set to the passed in value surrounded by 3 sets of asterixis. E.G. If the passed in</li> </ul> </li> </ul> |

|   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|---|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|   | <p>value is "Mo" the value of holder should be set to "*** Mo ***".</p> <ul style="list-style-type: none"><li>• The private balance filed should be replaced by a gettable but <b>not</b> settable property called Balance.</li></ul> <p>You will need to rework other elements of code in both the Account and Program classes in order to make the code compile.</p> <p>Add code to the Program class that fully tests the new features.</p>                                   |
| 3 | <p>We want to be able to create an account object allowing the balance to default to 0. i.e. a single parameter constructor of string name only. Give the Account class this as a second constructor. Use constructor chaining (see relevant slides in the PowerPoint appendix) so the single parameter constructor calls the two-parameter constructor passing the name and a zero for the balance.</p> <p>Add code to the Program class that fully tests the new features.</p> |

**Before moving on don't forget to commit and push your work to GitHub.**



**QA.com**