1. How would you modify your interface, for future version (new features) if many clients are already using it? I mean do they not affected with compile error, since not all methods/properties implemented??

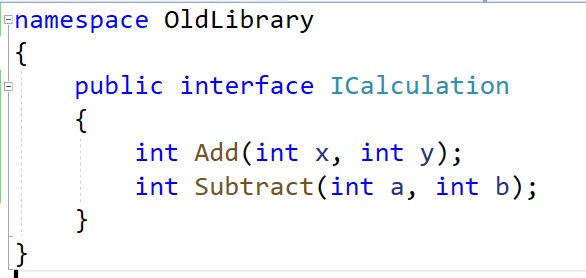
Answer: You can use the extension method for interface too. If client is using old library and old interface code, that client will not be impacted by errors when you extend the interface, as you will do it through another library. If the client do not use the library with extension methods for that interface, then simple the client will not be able to use new features/methods of the interface.

The beauty is, whichever class implements that old interface, Client application will be getting the extension methods for the class, when the client uses the library with new extension methods of the interface as well as the class which implements the old interface. You need not change anything in the class implementing the old interface. It’s just the client application who uses the class, implementing the old interface, has to add reference to the new library with extension methods for the interface (and that means for the class too).

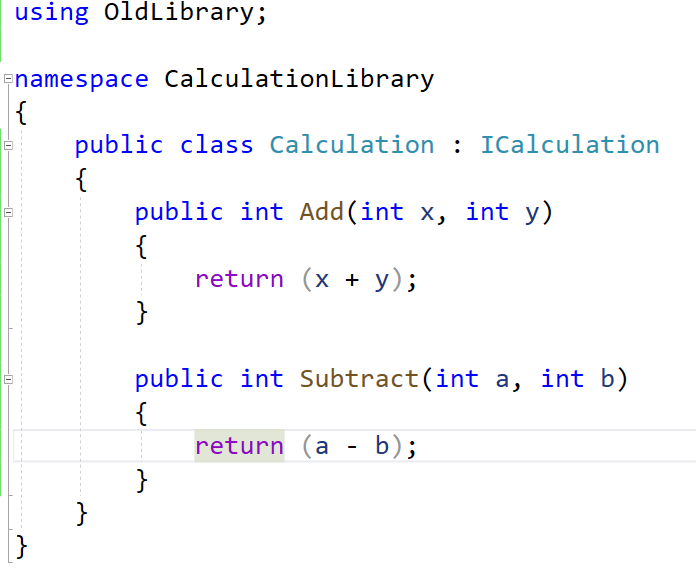
Following is an example. The code has been uploaded for your reference in the following link:

<https://github.com/joydip-git/siemens_ta_22ndMarch2021/tree/main/codes/samiullachikkodidoubts/InterfaceExtendingMechaism>

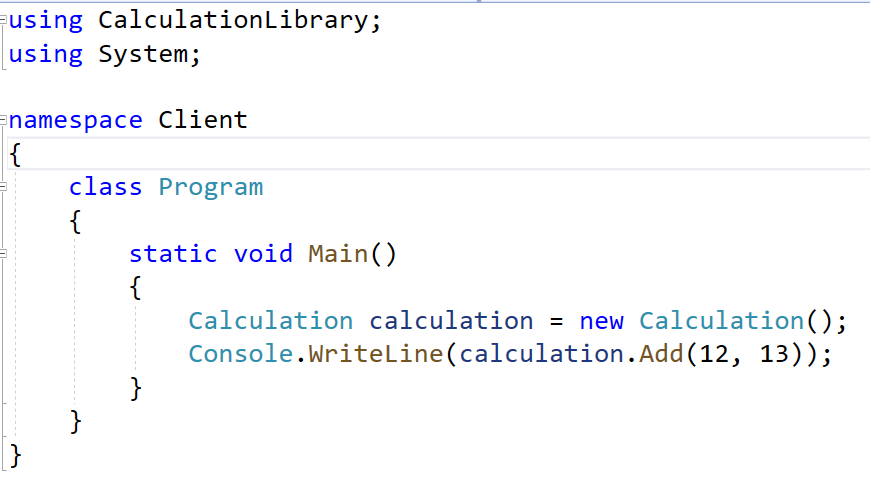
Say I have an old interface (ICalculation) in a library (OldLibrary) with just two methods: Add and Subtract.



This interface is implemented in a class, named, Calculation in a library (CalculationLibrary).

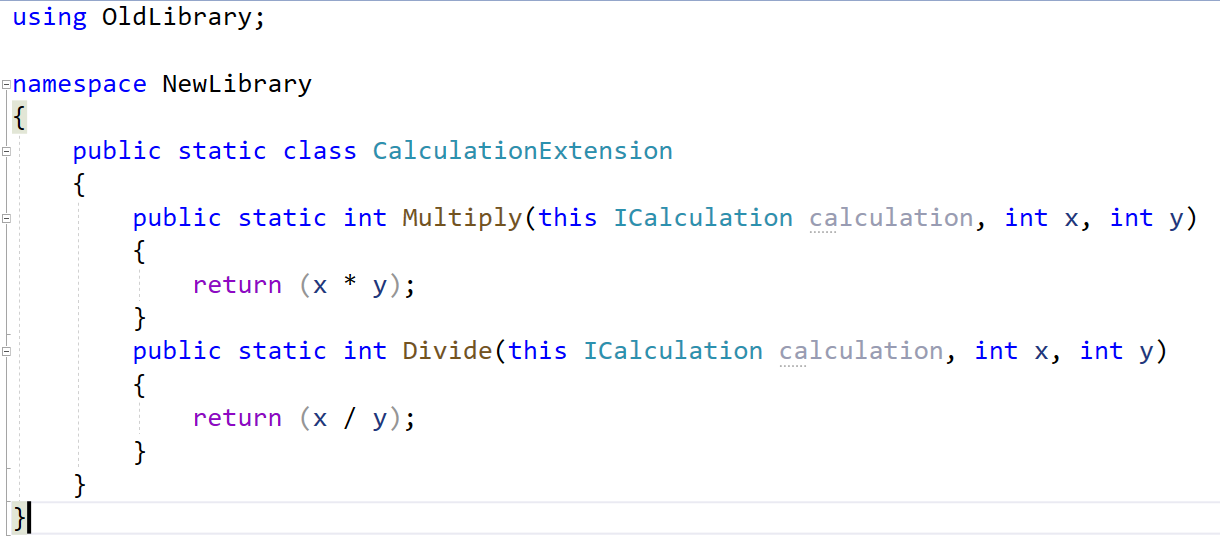


The console application, named, Client adds reference to the Calculation Library and creates object of Calculation class and calls Add or Subtract methods.

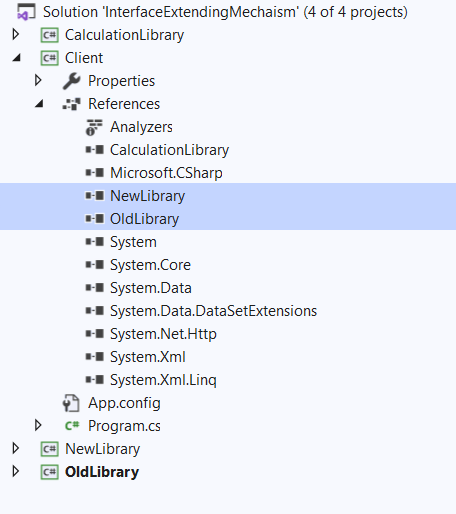


Now, you want to extend the interface.

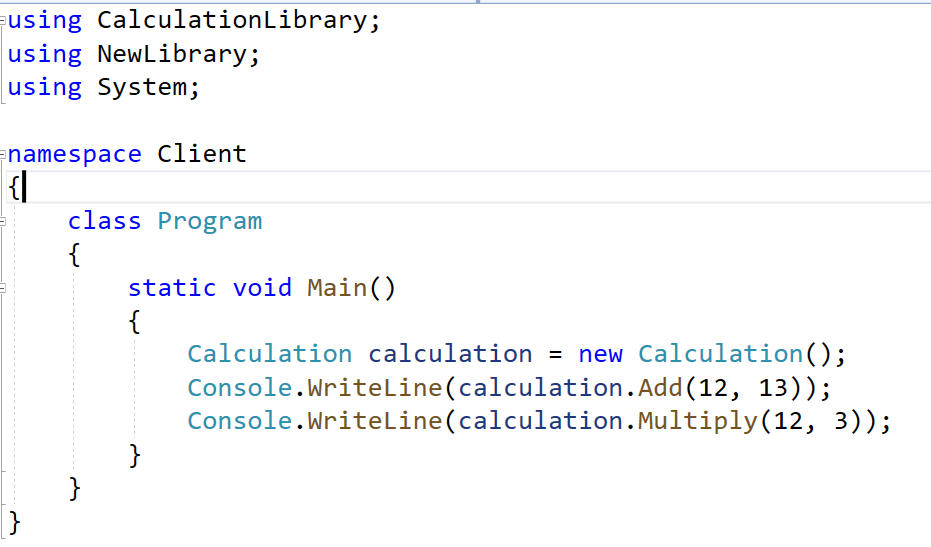
1. Create another library (NewLibrary) with a static class, named, CalculatiionExtension, with two static methods: Multiply and Divide



1. Now, in the console application (Client), add reference to the OldLibrary (containing ICalculation interface) and NewLibrary both (containing CalculationExtension class)



1. Use the namespace “using NewLibrary ” in Console application’s Program.cs file. Go ahead, and you will be able to call Multiply and Divide extension methods on Calculation object created in Main method.



1. Can you please explain once again this below problem?

Suppose I made Employee an Abstract class or Interface type and 2 concrete Class (Hr, Developer) like we did during the course. And in client code if I create an object of a base class type –

Employee employee = new Developer( - - - - )

Employee.ProjectName ?

I cannot access some properties if am upcasting it to base class, these properties were unique for concrete classes. How to address this issue, other than down casting it to again concrete type? Because I have a lot of classes. .

Answer:

1. You can put those methods/properties in base as virtual or abstract and override them in child class, but that might not be the ideal solution as you don’t want project name in the base class (employee), since it’s exclusively should be member of child class ONLY.
2. Hence, there is no other way rather than down-casting, how much irritating or unfortunate that might be.
3. Now let’s think about the code that you have written. You have created an object of Developer/Hr and tried to up-cast it. The question here is WHY? Until and unless a situation demands we try not to use up-casting when we create an object of child class and try to store the reference in a variable. So the following code should be:

Developer dev = new Developer( - - - - )

dev.ProjectName 🡪 you can access

Generally we do up-casting, when we are sure that there are certain members that can be accessed even if up-casting is done.