Library and dll

Framework Design Guidelines

- Naming Guidelines
  Provides guidelines for naming assemblies, namespaces, types, and members in class libraries.

- Type Design Guidelines
  Provides guidelines for using static and abstract classes, interfaces, enumerations, structures, and other types.

- Member Design Guidelines
  Provides guidelines for designing and using properties, methods, constructors, fields, events, operators, and parameters.

- Designing for Extensibility
  Discusses extensibility mechanisms such as subclassing, using events, virtual members, and callbacks, and explains how to choose the mechanisms that best meet your framework's requirements.

- Design Guidelines for Exceptions
  Describes design guidelines for designing, throwing, and catching exceptions.

- Usage Guidelines
  Describes guidelines for using common types such as arrays, attributes, and collections, supporting serialization, and overloading equality operators.

- Common Design Patterns

  Provides guidelines for choosing and implementing dependency properties and the dispose pattern.

Names of Namespaces

The goal when naming namespaces is creating sufficient clarity for the programmer using the framework to immediately know what the content of the namespace is likely to be.

The following template specifies the general rule for naming namespaces

```
<Company>.(<Product>|<Technology>)[.<Feature>][.<Subnamespace>]
```

Namespace Do and Don’t

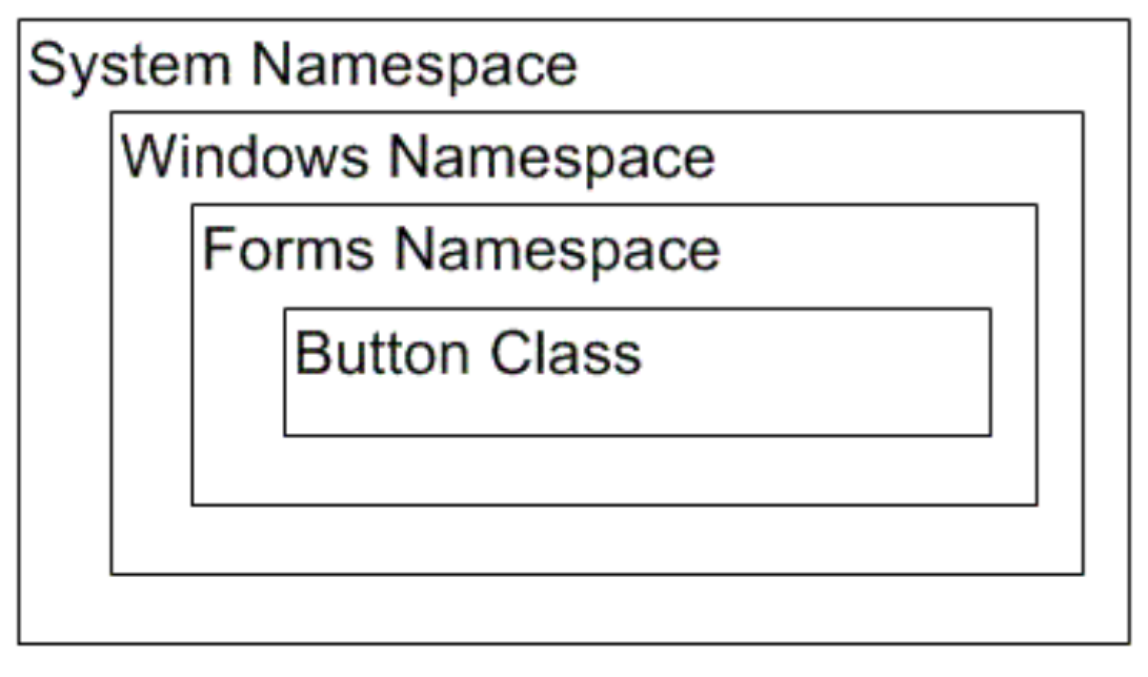
- DO prefix namespace names with a company name to prevent namespaces from different companies from having the same name.
- ✓ DO use a stable, version-independent product name at the second level of a namespace name.
- X DO NOT use organizational hierarchies as the basis for names in namespace hierarchies, because group names within corporations tend to be short-lived. Organize the hierarchy of namespaces around groups of related technologies.

- ✓ DO use PascalCasing, and separate namespace components with periods (e.g., Microsoft.Office.PowerPoint). If your brand employs nontraditional casing, you should follow the casing defined by your brand, even if it deviates from normal namespace casing.

- X DO NOT use the same name for a namespace and a type in that namespace.

Namespace example

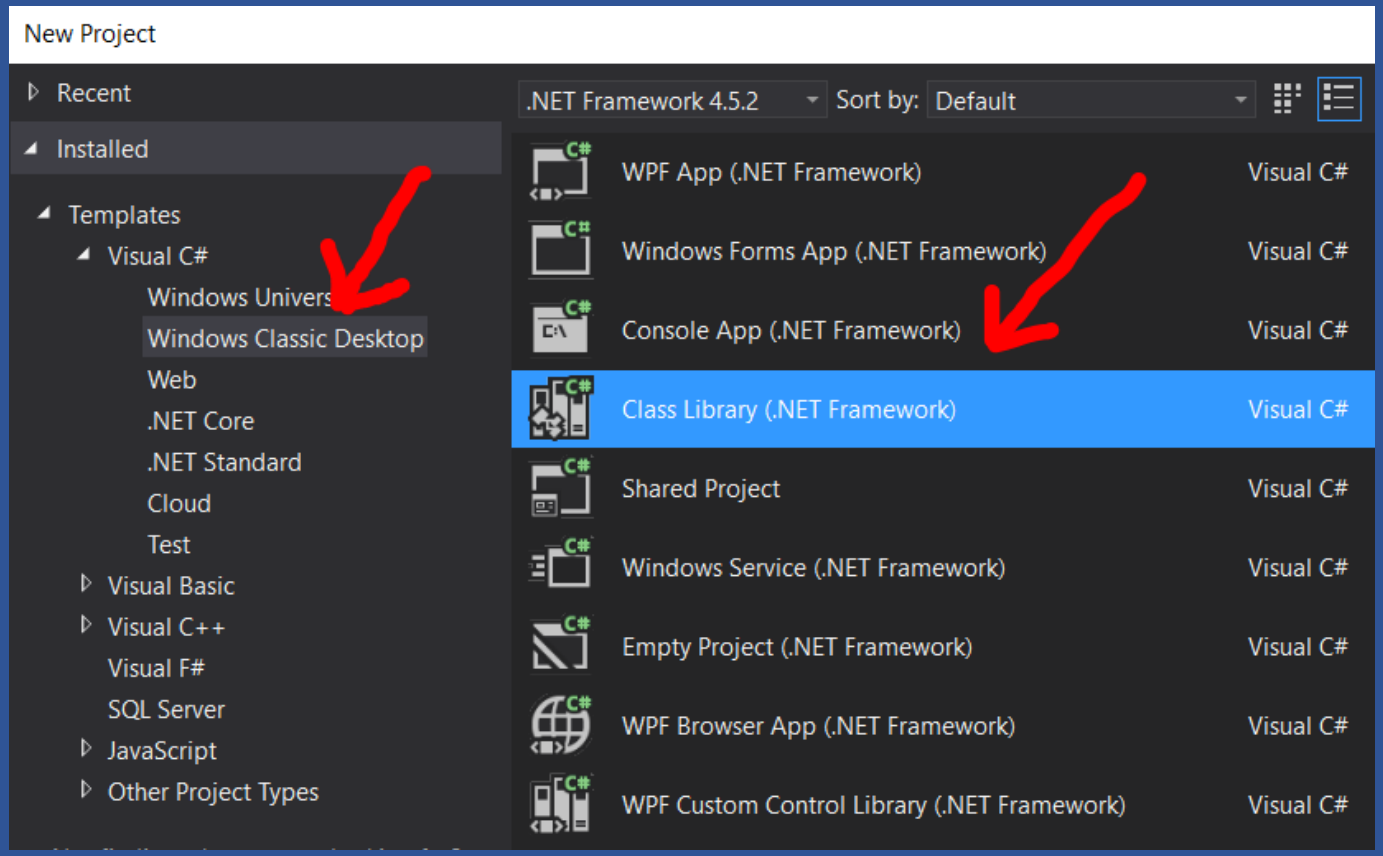
System.Windows.Forms.Button



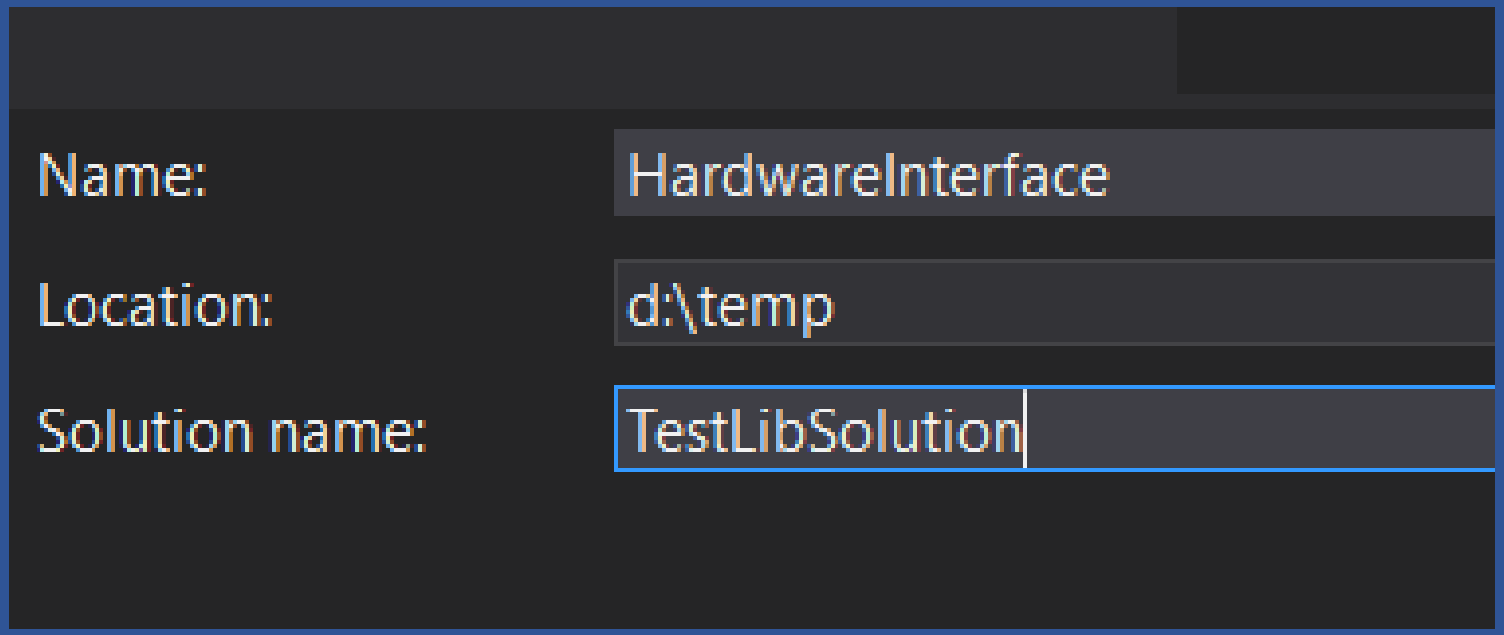
The Assembly Manifest

- The assembly name and version
- The culture or language the assembly supports (not required in all assemblies)
- The public key for any strong name assigned to the assembly (not required in all assemblies)
- A list of files in the assembly with hash information
- Information on exported types
- Information on referenced assemblies

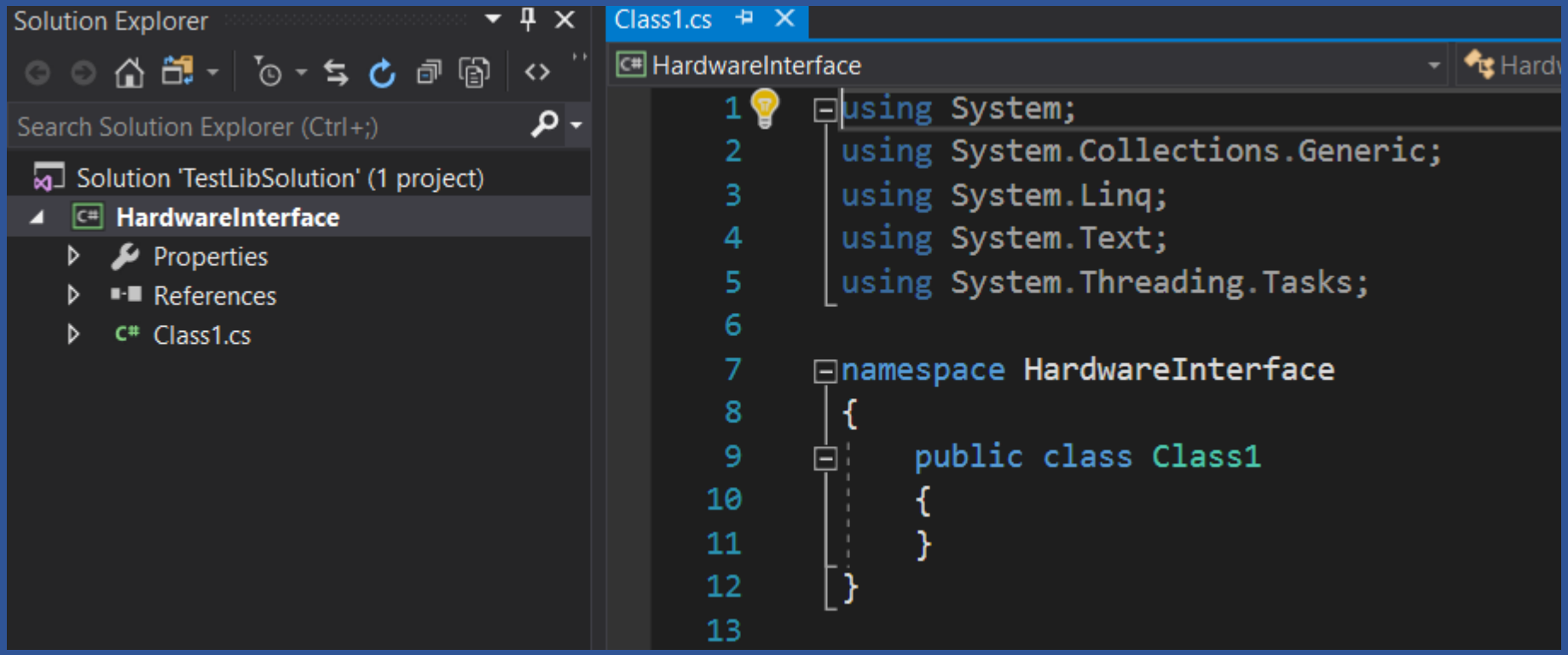
Create Class Library



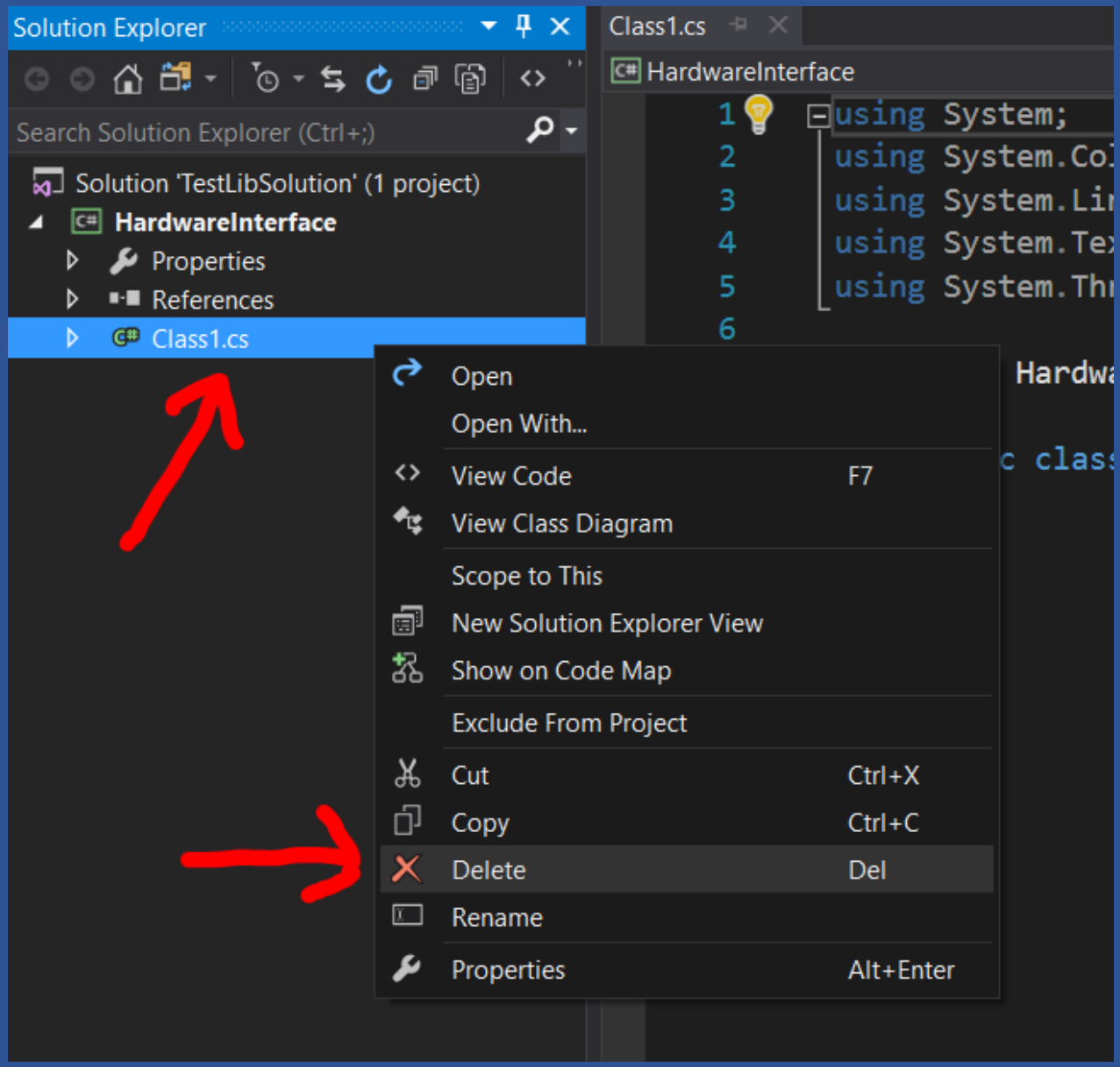
Set Project Name / Location / Solution Name



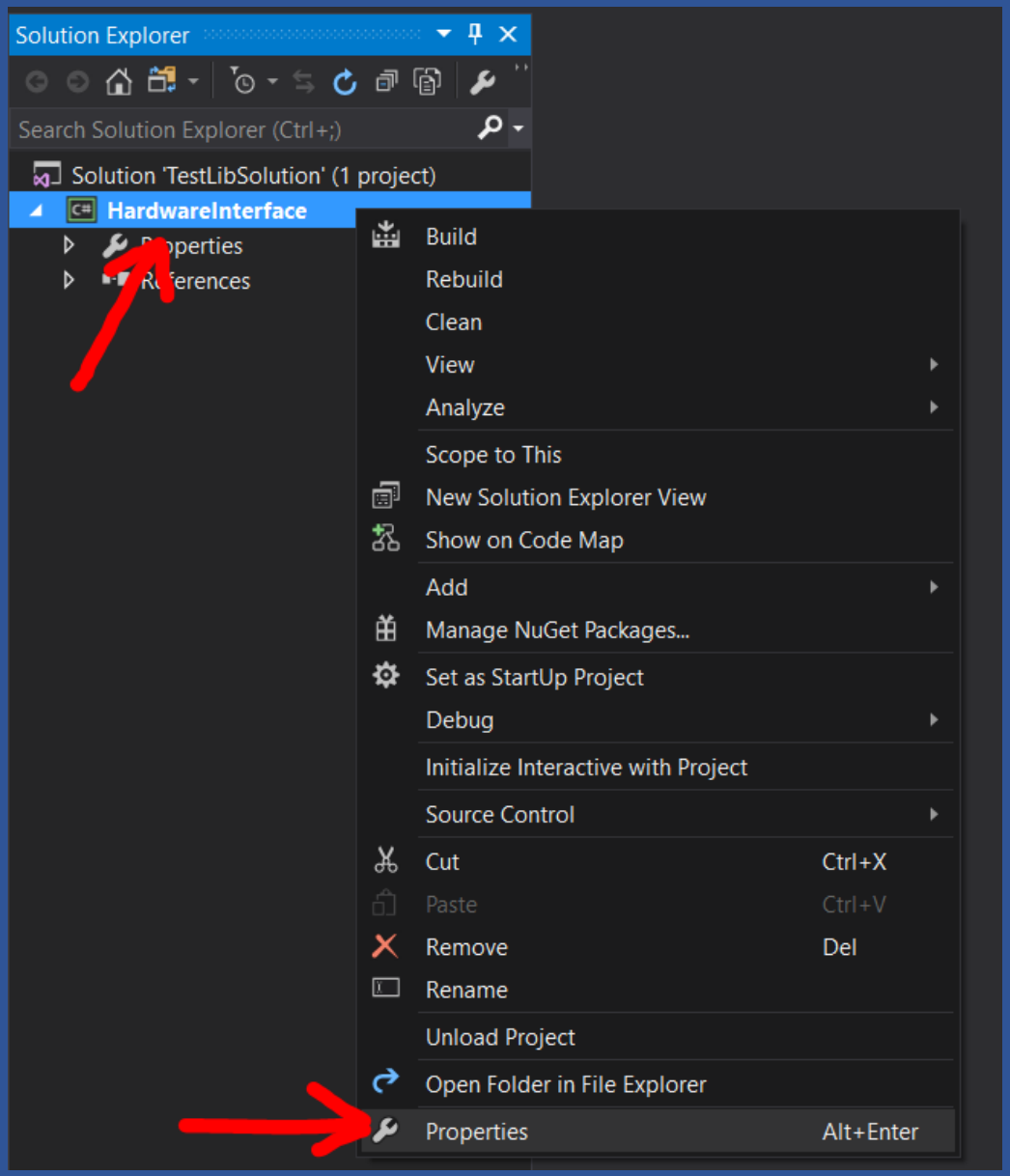
Empty Class library project



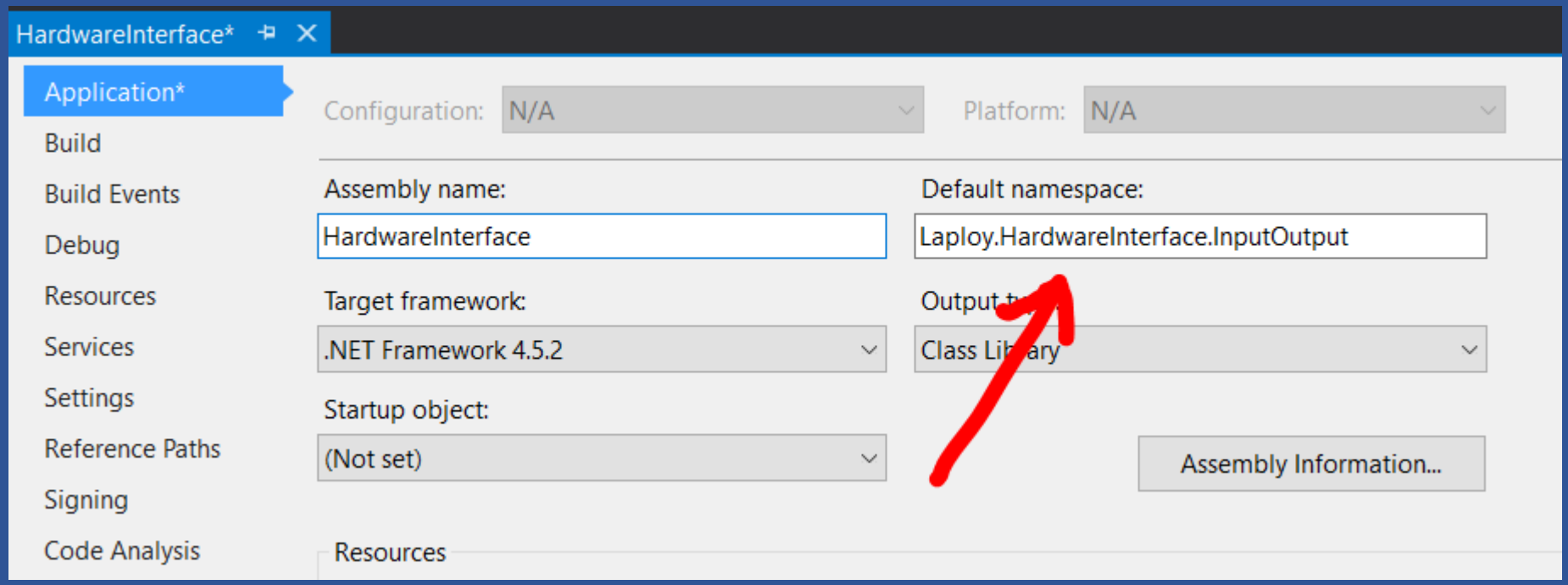
Delete Class1.cs



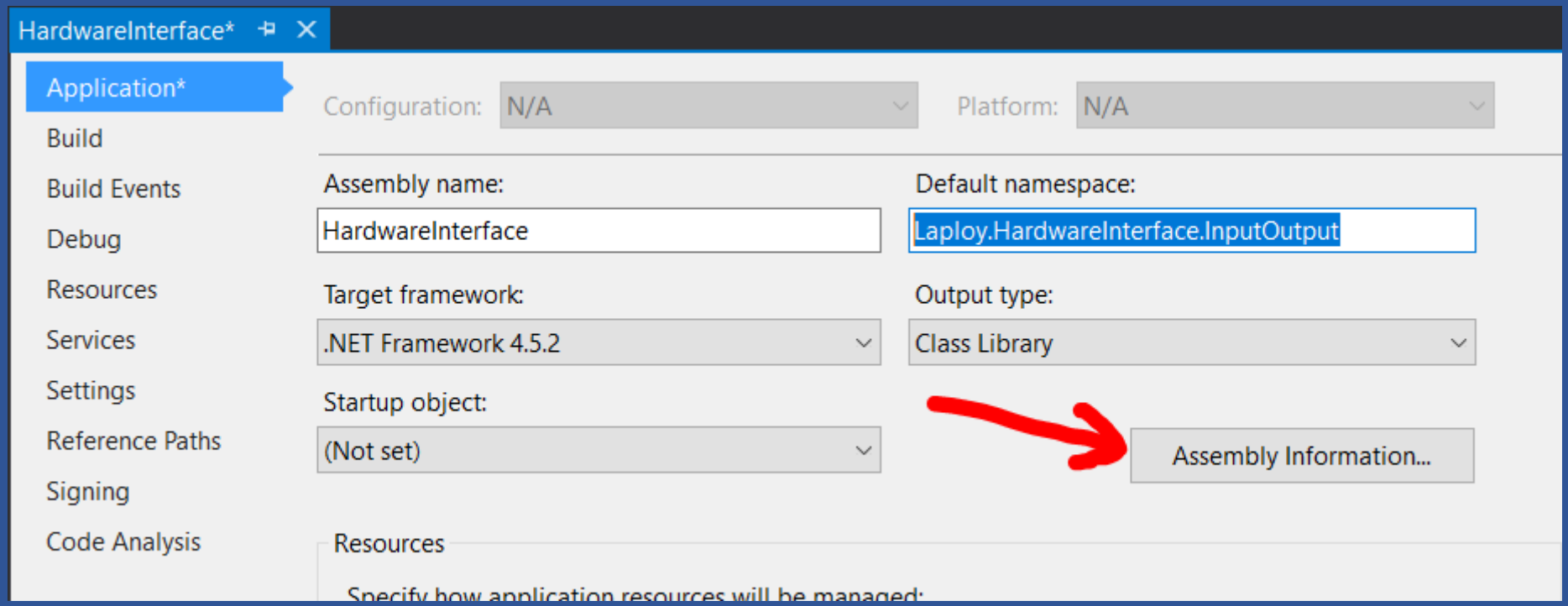
- Right-Click at the project's name
- Properties



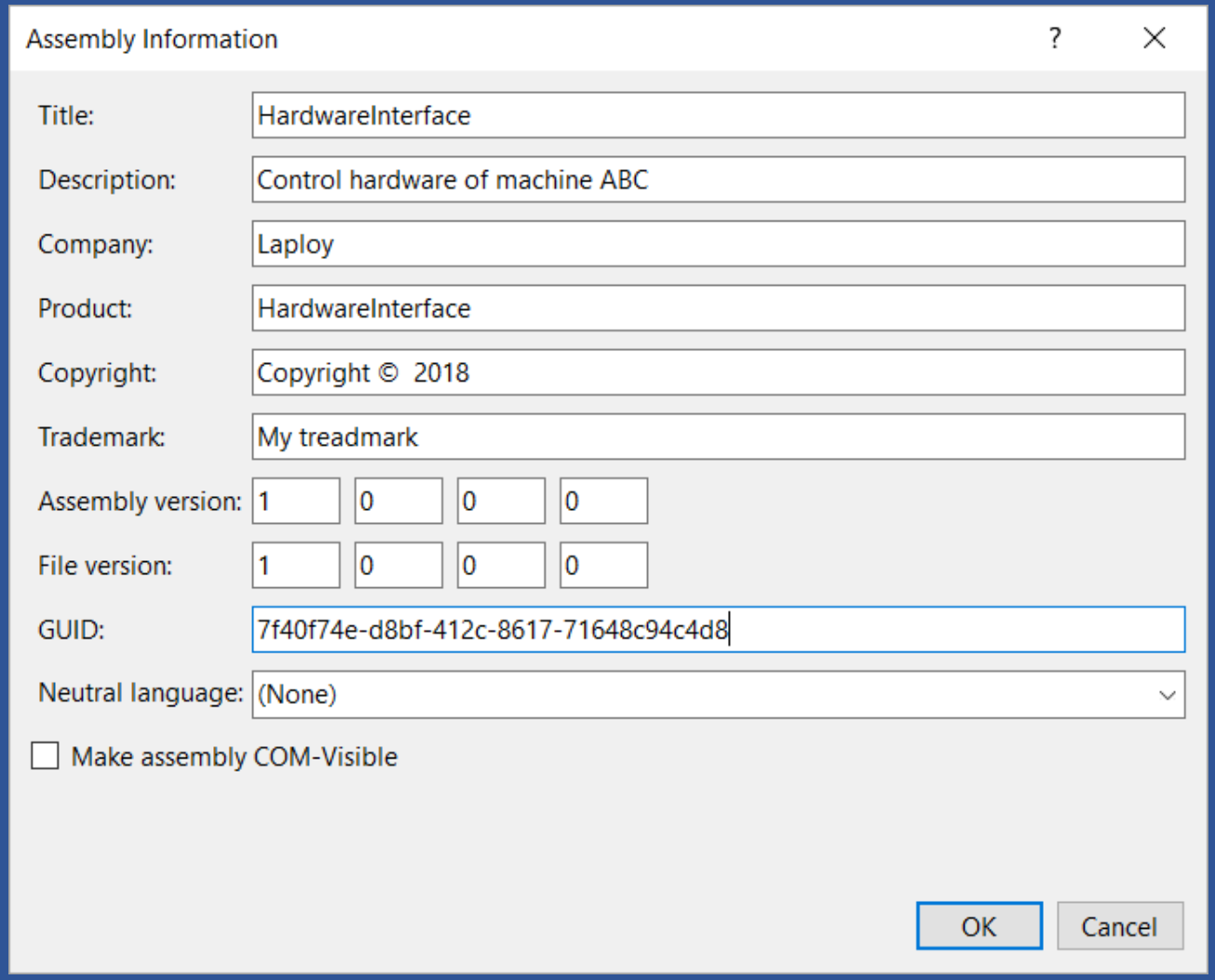
Change default namespace to Laploy.HardwareInterface.InputOutput



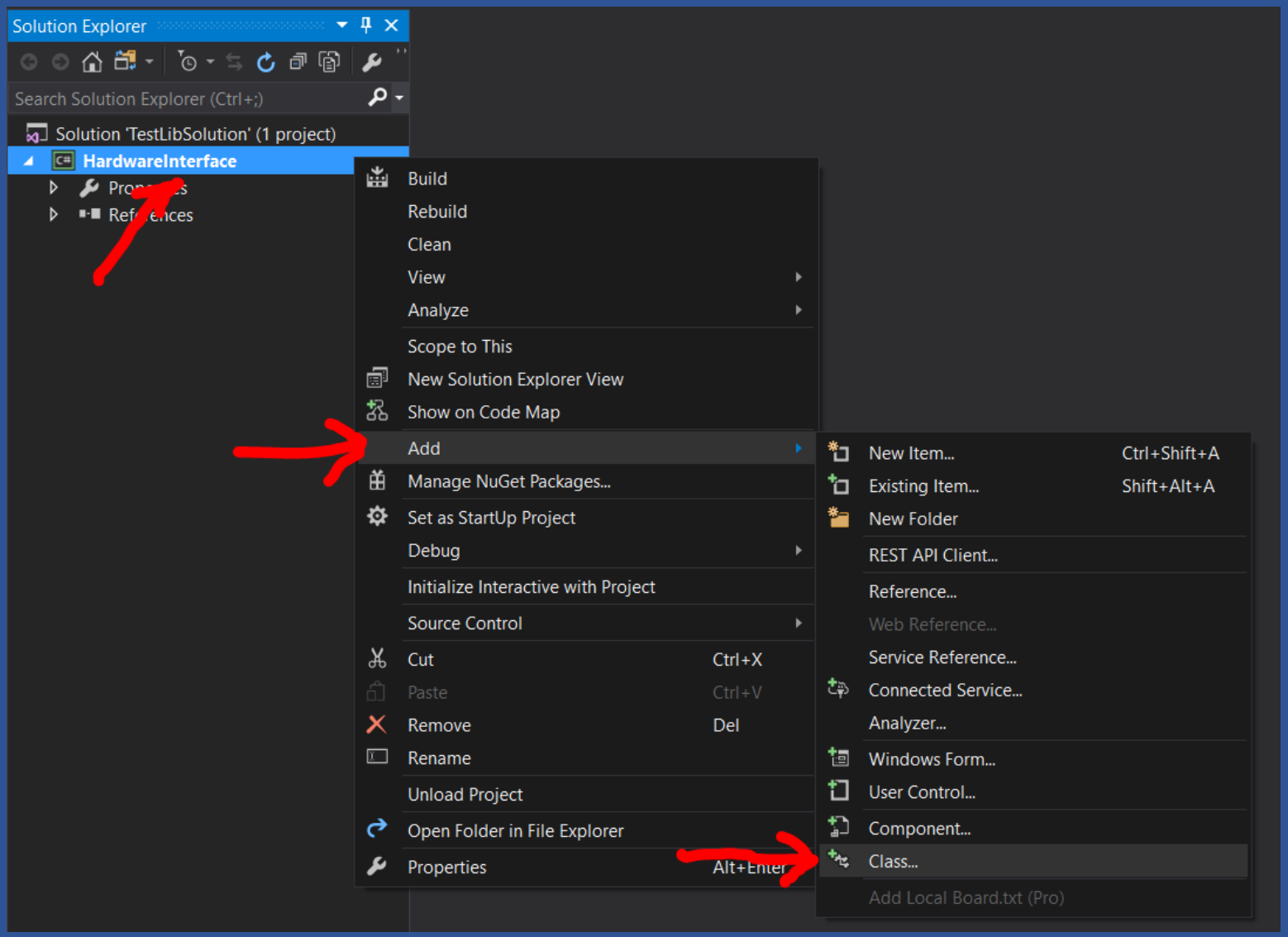
Click Assembly Information... button



Set Assembly Information



Add new class to project



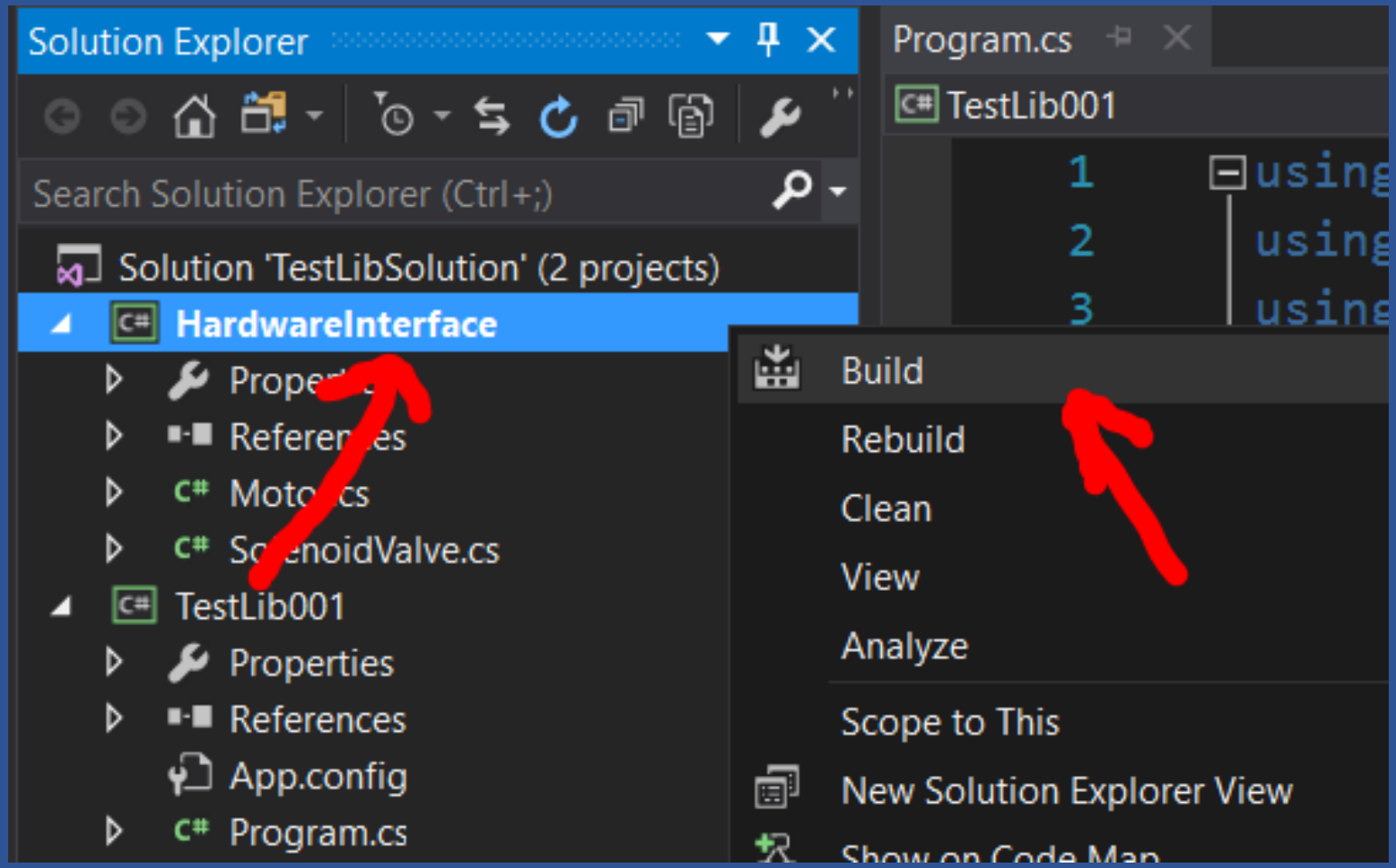
Add code to class SolenoidValve

```
3   using System;
4
5   namespace Laploy.HardwareInterface.InputOutput
6   {
7       public class SolenoidValve
8       {
9           public int Id { get; set; }
10          public string Name { get; set; }
11          public void TurnOn()
12          {
13              Console.WriteLine($"Solinoide {Name} is turned-on");
14          }
15          public void TurnOff()
16          {
17              Console.WriteLine($"Solinoide {Name} is turned-off");
18          }
19      }
20  }
```

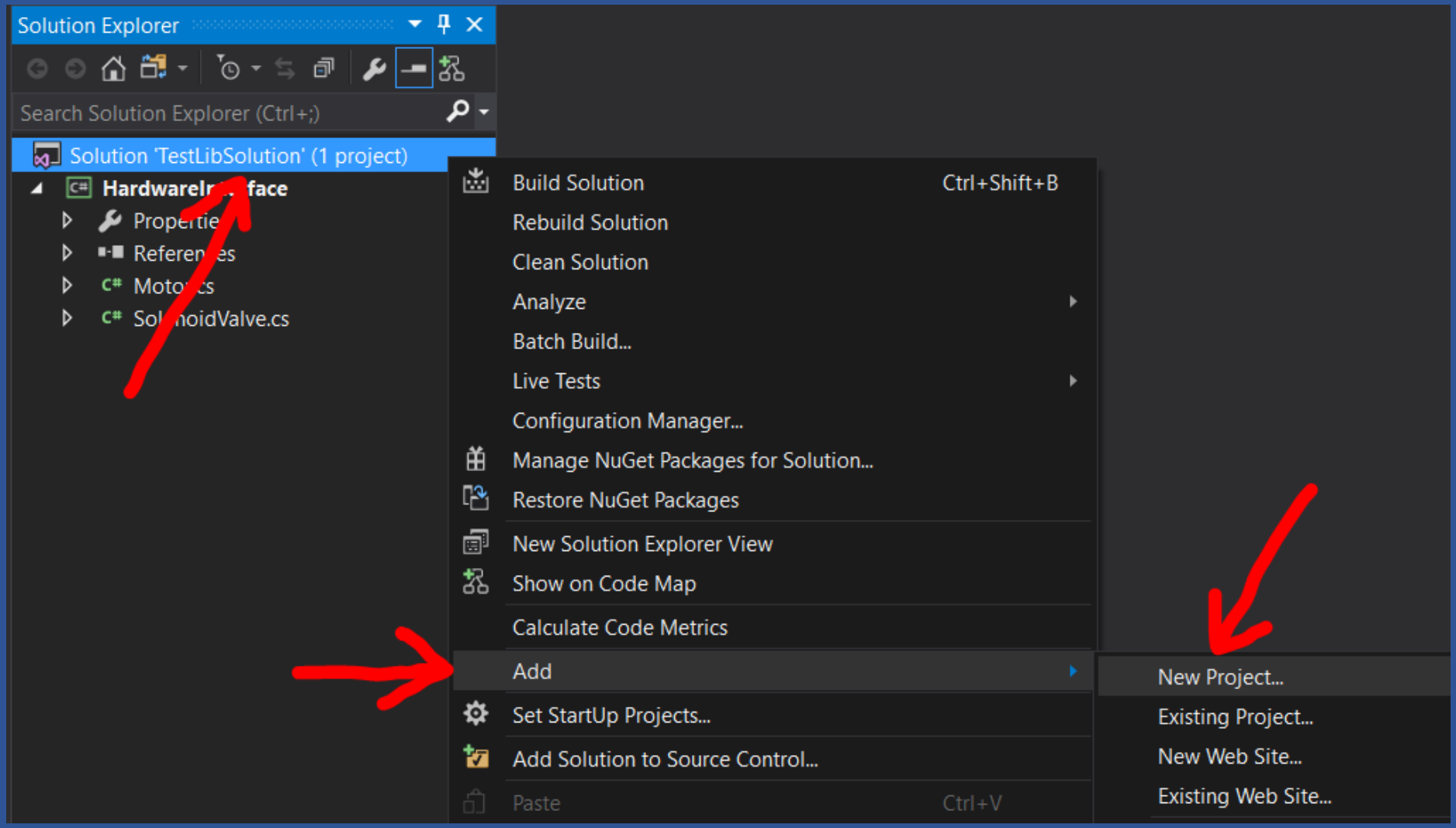
Add class Motor and add code

```
3     using System;
4
5     namespace Laploy.HardwareInterface.InputOutput
6     {
7         public class Motor
8         {
9             public int Id { get; set; }
10            public string Name { get; set; }
11            public int port { get; set; }
12
13            public void Start()
14            {
15                Console.WriteLine($"Motor {Id} port {port} started");
16            }
17            public void Stop()
18            {
19                Console.WriteLine($"Motor {Id} port {port} stoped");
20            }
21        }
22    }
```

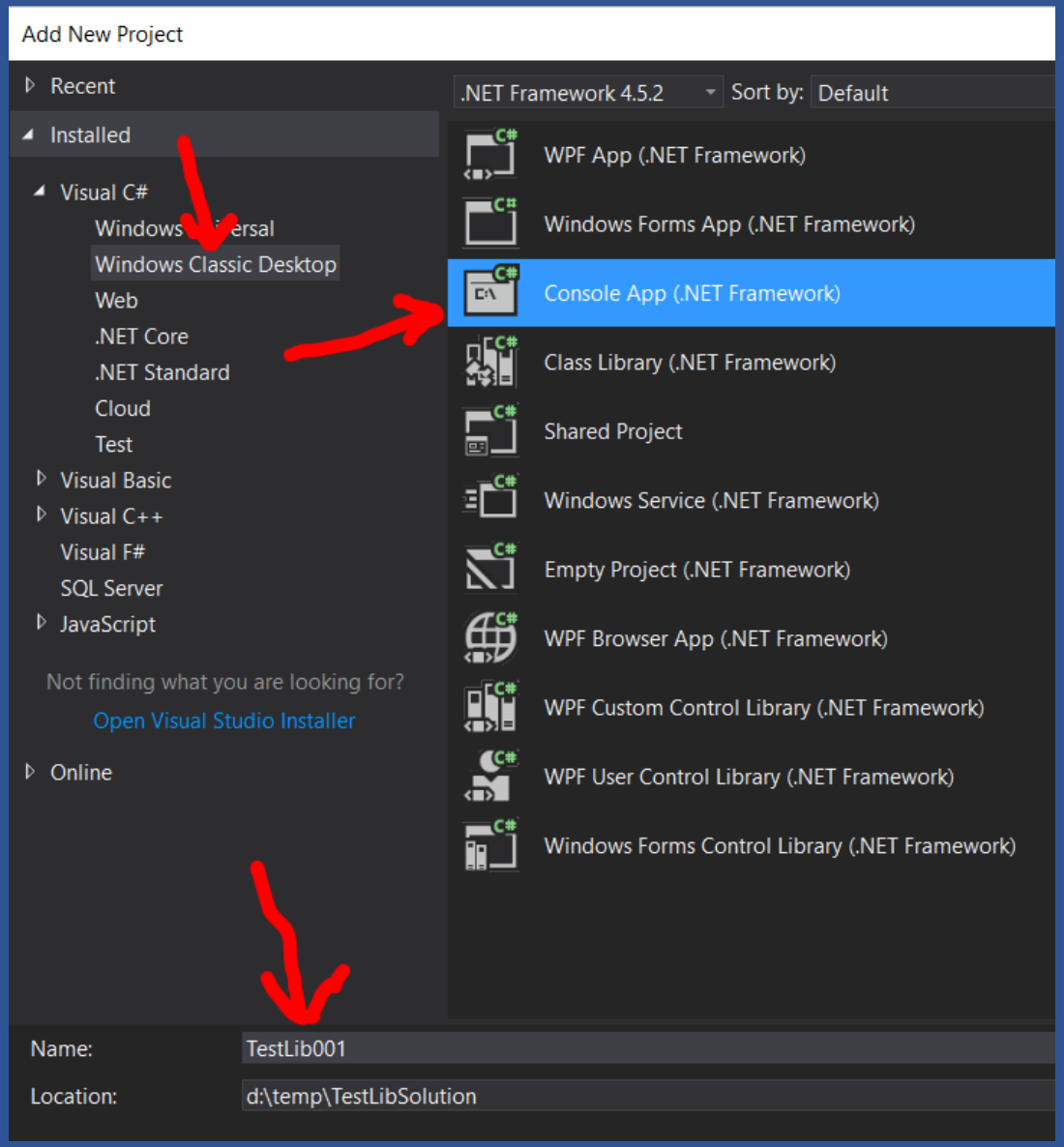
Build project HardwareInterface



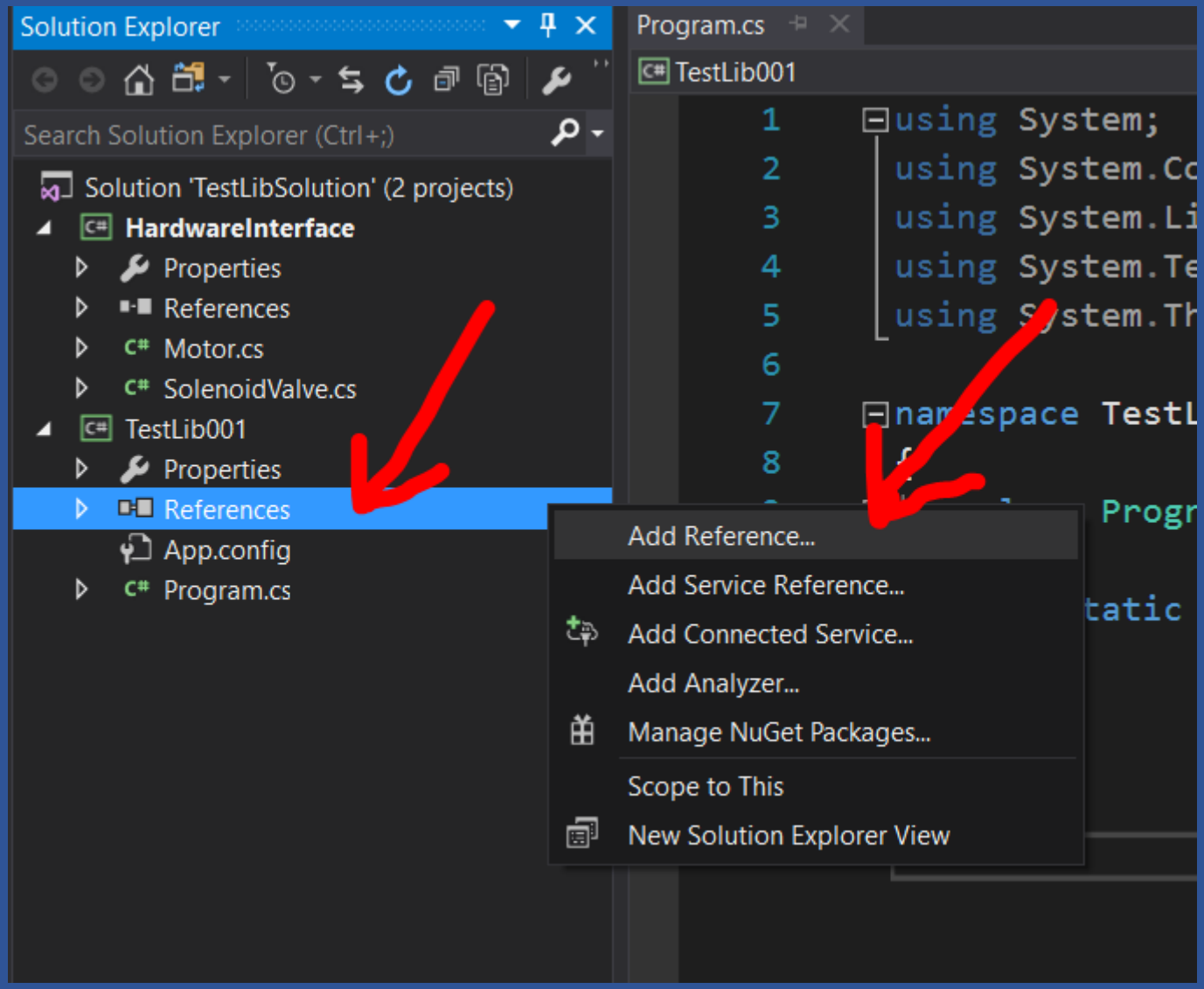
Add new project



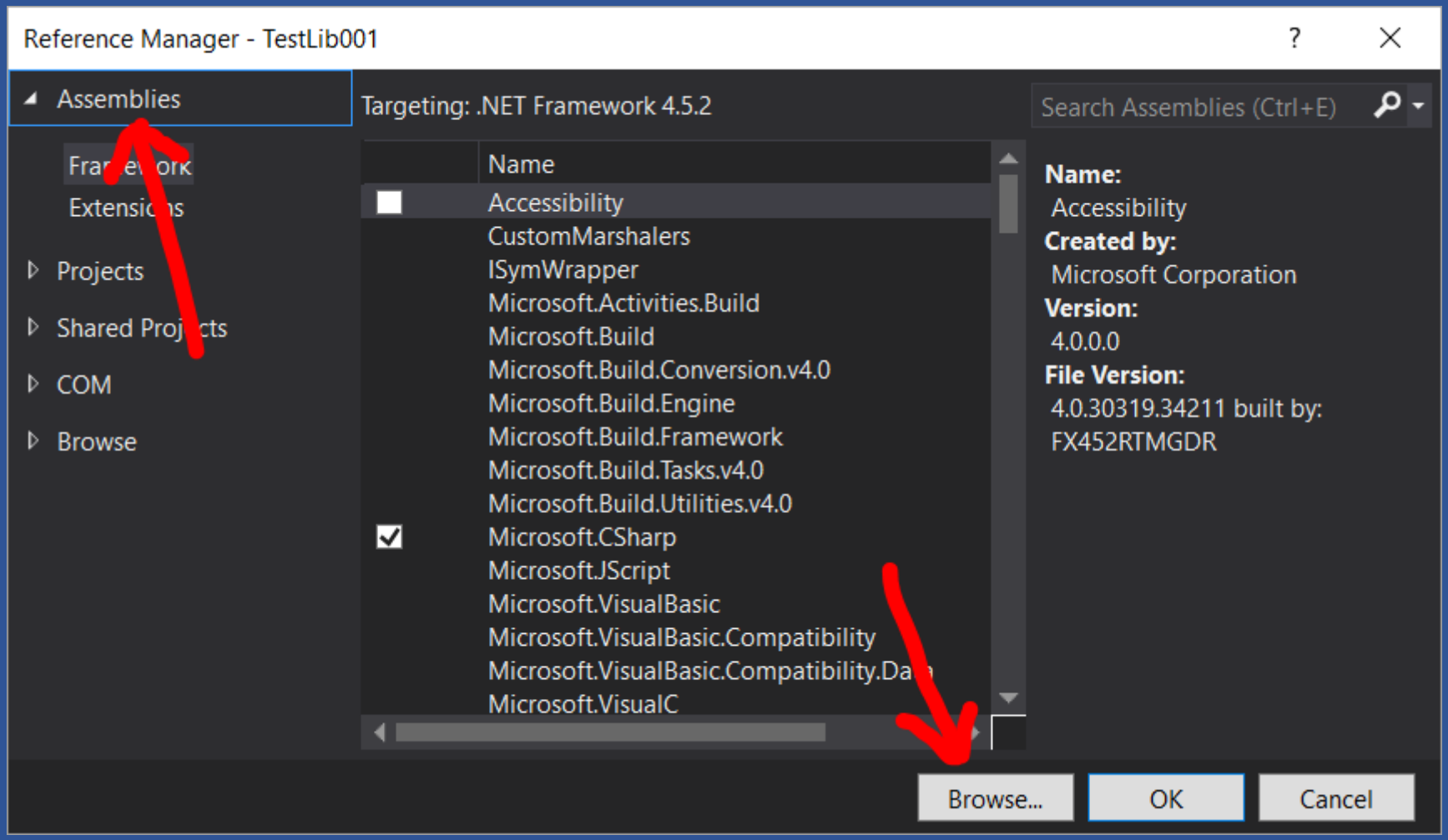
Console App / TestLib001



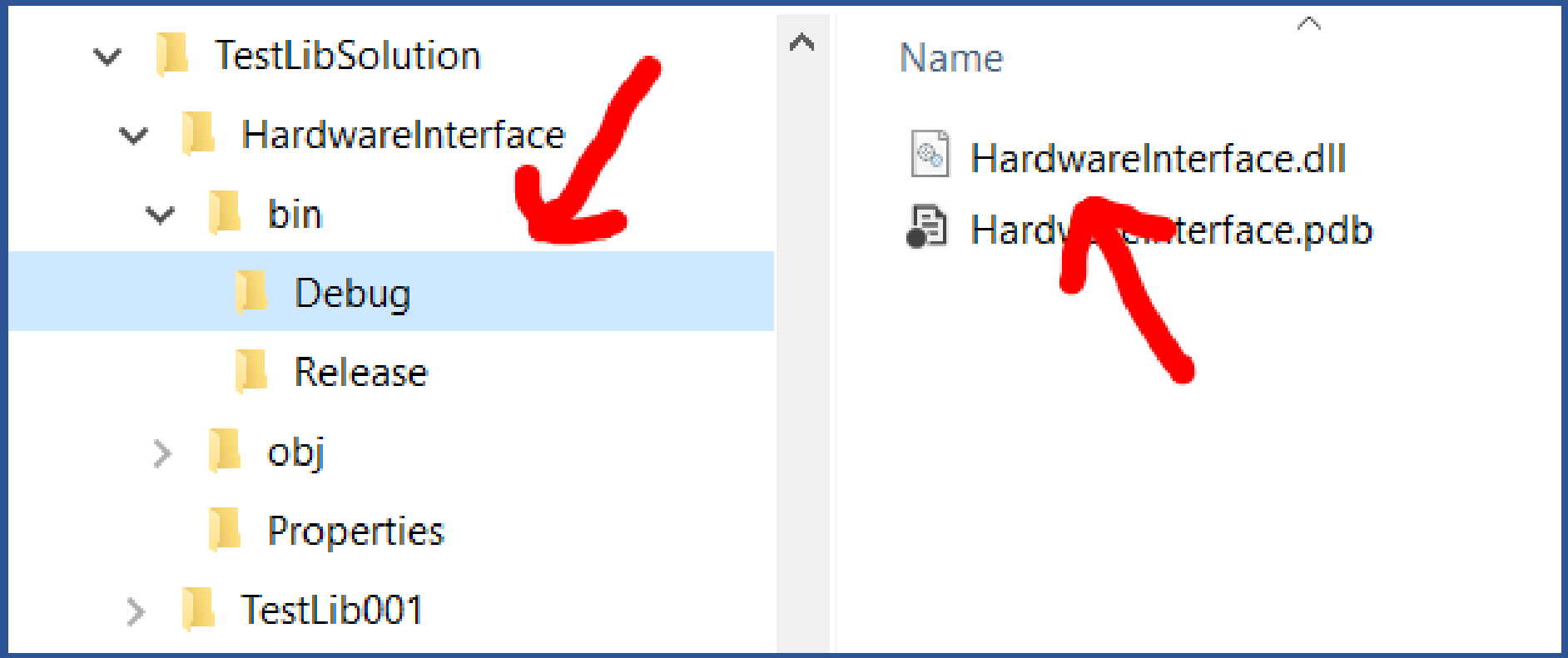
Add Reference



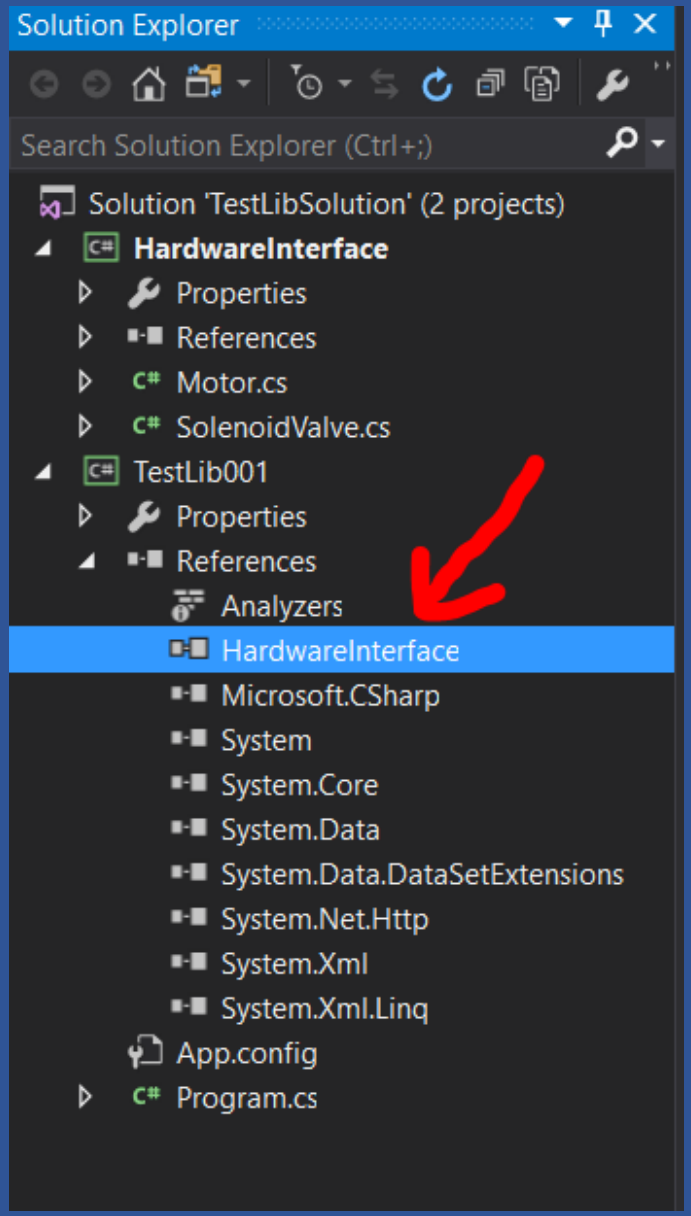
Add Assemblies / Browse...



Add Hardwareinterface.dll in TestLibSolution / HardwareInterface / bin / Debug



HardwareInterface appeared in the References list



Add using to TestLib001 / program.cs

```
using Laploy.HardwareInterface.InputOutput;
```

Add method ValveTest() to TestLib001 / program.cs

```
15      static void ValveTest()
16      {
17          SolenoidValve myValve1 = new SolenoidValve();
18          SolenoidValve myValve2 = new SolenoidValve();
19          myValve1.Id = 12;
20          myValve1.Name = "Front Valve";
21          myValve2.Id = 23;
22          myValve2.Name = "Rear Valve";
23
24          myValve1.TurnOn();
25          myValve2.TurnOn();
26
27          myValve1.TurnOff();
28          myValve2.TurnOff();
29      }
```

Add method MotorTest() to TestLib001 / program.cs

```
30        static void MotorTest()
31        {
32            Motor myMotorA = new Motor();
33            myMotorA.Id = 1;
34            myMotorA.port = 123;
35            myMotorA.Start();
36            myMotorA.Stop();
37
38            Motor myMotorB = new Motor();
39            myMotorB.Id = 2;
40            myMotorB.port = 9000;
41            myMotorB.Start();
42            myMotorB.Stop();
43        }
```

Add code to Main()

```
4   using Laploy.HardwareInterface.InputOutput;
5
6   namespace TestLib001
7   {
8       class Program
9       {
10          static void Main(string[] args)
11          {
12              ValveTest();
13              MotorTest();
14          }
15          static void ValveTest()...
30          static void MotorTest()...
44      }
45  }
```

Build and Run

C:\WINDOWS\system32\cmd.exe

```
Solinoide Front Valve is turned-on
Solinoide Rear Valve is turned-on
Solinoide Front Valve is turned-off
Solinoide Rear Valve is turned-off
Motor 1 port 123 started
Motor 1 port 123 stoped
Motor 2 port 9000 started
Motor 2 port 9000 stoped
Press any key to continue . . .
```

Exercise

1. Create new Solution “TestLibSolution2”
2. Create Library Project “HardwareSensors”
3. Add Class “TemperatureSensor”
4. Add auto-properties Id, Name, Tem
5. Add constructor to initiate properties values
6. Create Consol App “TestLib002”
7. Add reference of HardwareSensor assembly
8. Write code to test TemperatureSensor