Assemblies in .NET

Objectives

- Overview Assemblies in .NET
- Explain components in .NET Assemblies: Manifest, Metadata, CIL and Resources
- Explain Role of .NET Assemblies
- Explain types .NET Assemblies : Static and Dynamic
- Explain and demo about view CIL Code assemblies by ildasm tool
- Explain demo about dumpbin tool
- Demo create Assemblies and consume with C# Console Application

What is the .NET Assemblies

- Assemblies form the fundamental units of deployment, version control, reuse, activation scoping, and security permissions for .NET-based applications
- An assembly is a collection of types and resources that are built to work together and form a logical unit of functionality
- Assemblies take the form of executable (.exe) or dynamic link library (.dll) files, and are the building blocks of .NET applications. They provide the common language runtime with the information it needs to be aware of type implementations
- In .NET(.NET 5), we can build an assembly from one or more source code files. This allows larger projects to be planned so that several developers can work on separate source code files or modules, which are combined to create a single assembly

Assemblies Properties

- Assemblies are implemented as **.exe(Windows OS)** or **.dll** files
- A .NET assembly can be **static** or **dynamic** and can be single-module or multimodule
- **Static assemblies** are stored on disk in portable executable (PE) files. Static assemblies can include interfaces, classes, and resources like bitmaps, JPEG files, and other resource files
- **Dynamic assemblies** are created dynamically at runtime and by using a specialized API of the .NET Core BCL called the Reflection Emit API, which is part of Reflection Services. A dynamic .NET assembly is created and executed directly in memory and it can be saved in a storage device

Assemblies Properties

- Assemblies are only loaded into memory if they are required. If they aren't used, they aren't loaded. This means that assemblies can be an efficient way to manage resources in larger projects
- We can programmatically obtain information about an assembly by using Reflection. We can load an assembly just to inspect it by using the MetadataLoadContext class and the *Assembly.ReflectionOnlyLoad* or *Assembly.ReflectionOnlyLoadFrom methods*

Role of .NET Assemblies

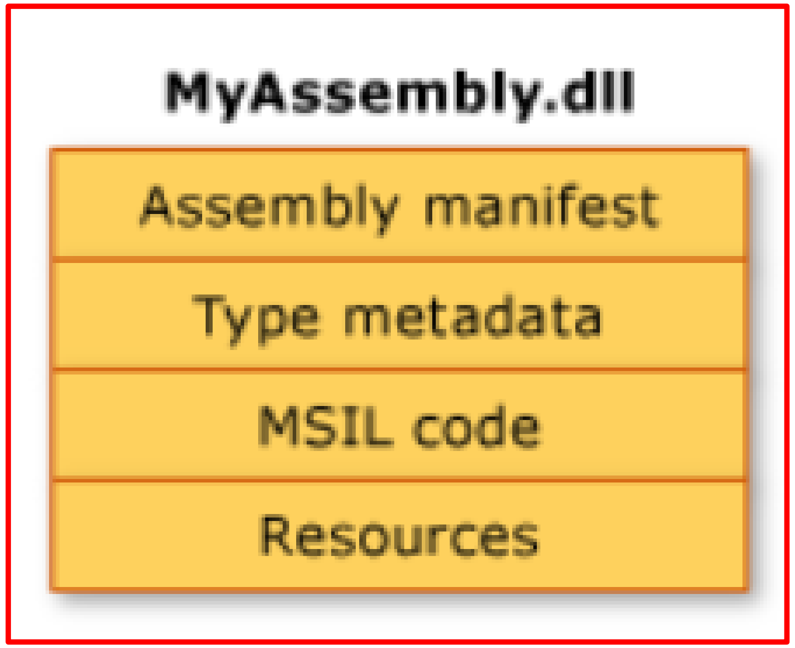
- **Assemblies Promote Code Reuse**
  - A code library (a class library) is a *.dll that contains types intended to be used by external applications and allows us to reuse types in a language-independent manner
- **Assemblies Establish a Type Boundary**
  - Every type's identity includes the name of the assembly in which it resides. A type called “*MyType”* that is loaded in the scope of one assembly is not the same as a type called “*MyType*” that is loaded in the scope of another assembly

Role of .NET Assemblies

- **Assemblies Are Versionable Units**
  - Assemblies are assigned a four-part numerical version number of the form **<major>.<minor>.<build>.<revision>** that allows multiple versions of the same assembly to coexist in harmony on a single machine
  - All types and resources in the same assembly are versioned as a unit
- **Assemblies Are Self-Describing**
  - Assemblies are regarded as self-describing, in part because they record in the assembly’s manifest every external assembly they must have access to in order to function correctly
  - In addition to manifest data, an assembly contains metadata that describes the composition (member names, implemented interfaces, base classes, constructors, etc.) of every contained type

The Format of a .NET Assembly

- A .NET assembly(*.dll or *.exe) consists of the following elements:
  - An operating system (e.g. Windows) file header
  - A CLR file header
  - **An assembly manifest**
  - **Type metadata**
  - **CIL code**
  - ***Optional embedded resources***

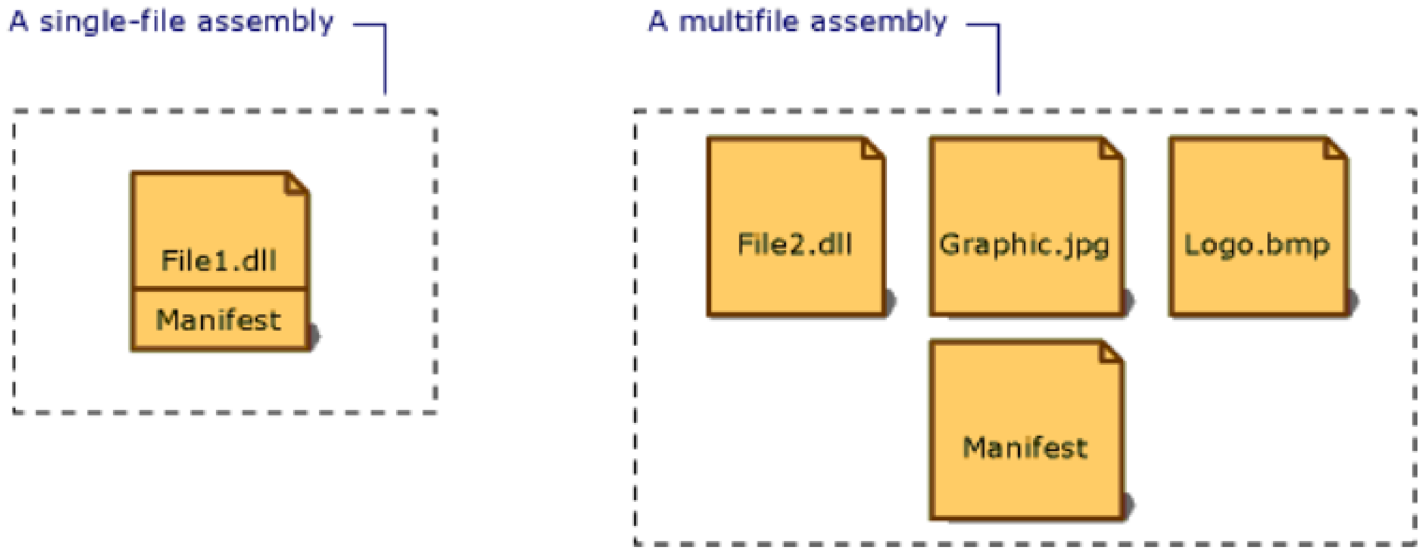


The Format of a .NET Assembly

- **Type metadata:** An assembly contains metadata that completely describes the format of the contained types, as well as the format of external types referenced by this assembly. The .NET Core runtime uses this metadata to resolve the location of types (and their members) within the binary, lay out(arrange) types in memory, and facilitate remote method invocations
- **CIL Code:** At its core, an assembly contains CIL code that at runtime, the internal CIL is compiled on the fly using a just-in-time (JIT) compiler, according to the platform- and CPU-specific instructions
- **Optional embedded resources**: contains any number of embedded resources, such as application icons, image files, sound clips, string tables or localized resources

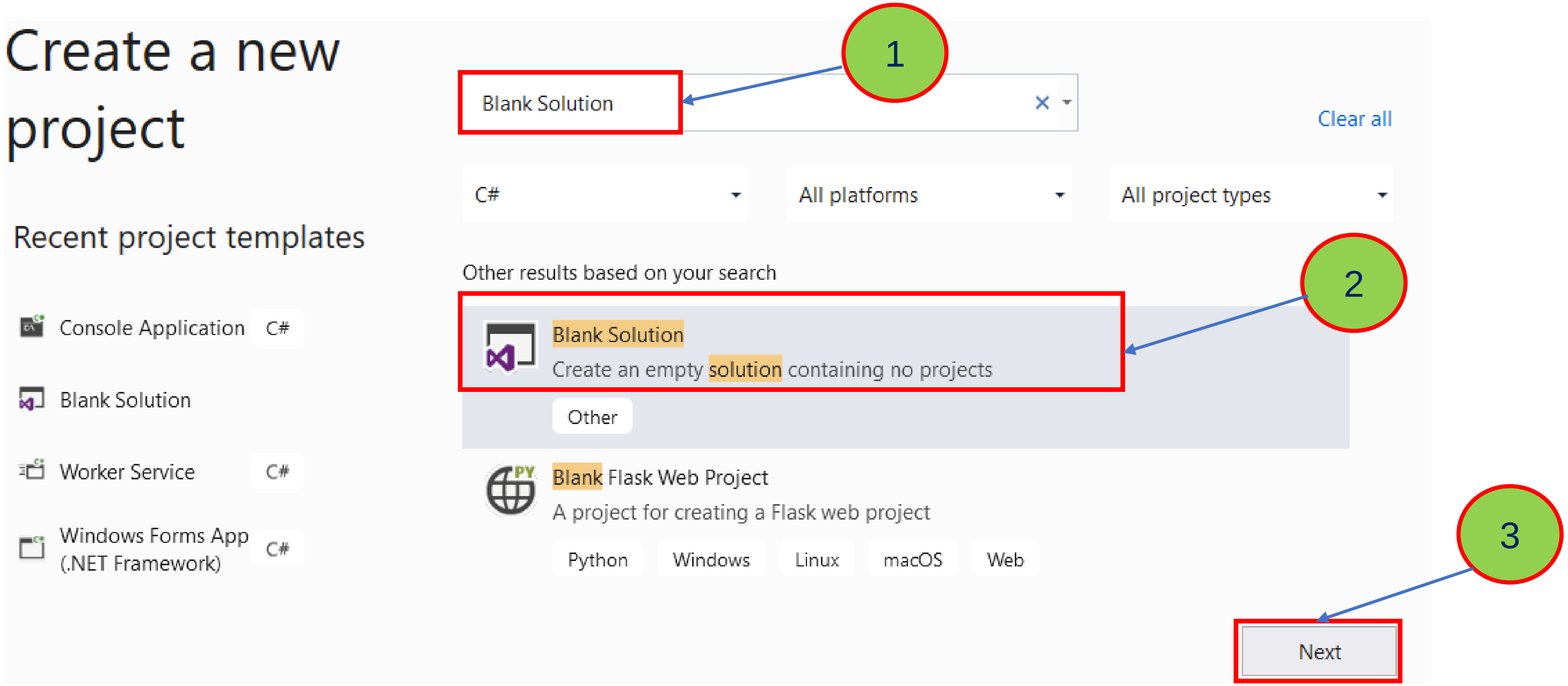
The Format of a .NET Assembly

- **An assembly manifest:**
  - An assembly manifest contains all the metadata needed to specify the assembly's version requirements and security identity, and all metadata needed to define the scope of the assembly and resolve references to resources and classes
  - The assembly manifest can be stored in either a PE file (an .exe or .dll) with Microsoft intermediate language (MSIL) code or in a standalone PE file that contains only assembly manifest information



Create .NET Assemblies Demonstration

1. Create a Blank Solution to include all projects(Class Library and Console Application) by open the Visual Studio .NET application, choosing File | New Project | Blank Solution | Next



Create a new project

Recent project templates

Console Application C#

Blank Solution

Worker Service C#

Windows Forms App (.NET Framework) C#

Blank Solution

Clear all

C#

All platforms

All project types

Other results based on your search

Blank Solution

Create an empty solution containing no projects

Other

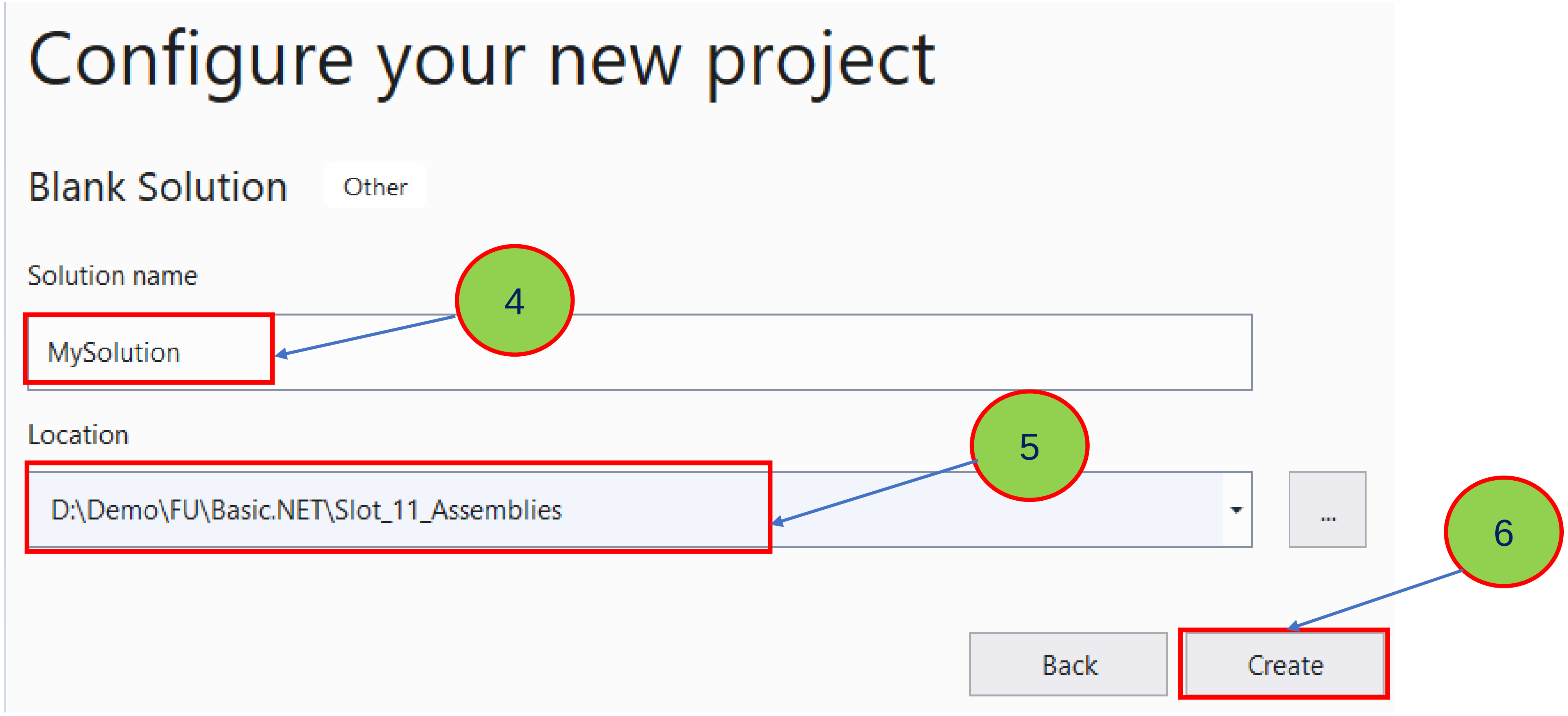
Blank Flask Web Project

A project for creating a Flask web project

Python Windows Linux macOS Web

Next

2. Fill out **Solution name**: MySolution and **Location** then click **Next**



Configure your new project

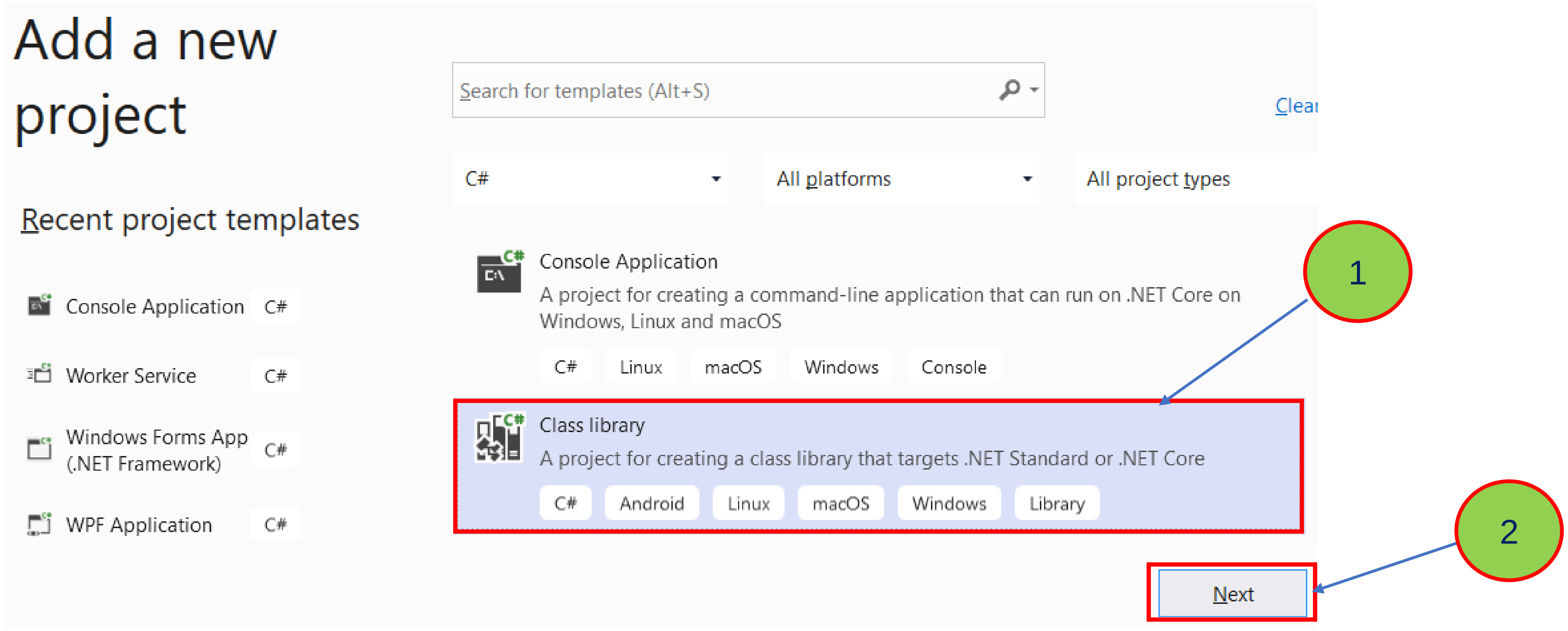
Blank Solution Other

Solution name: MySolution

Location: D:\Demo\FU\Basic.NET\Slot_11_Assemblies

3. Add to the **MySolution** a Class Library project to create the assemblies

- From the File menu | Add | New Project, on the Add New Project dialog, select "Class Library" then Next



Add a new project

Search for templates (Alt+S)

Clear

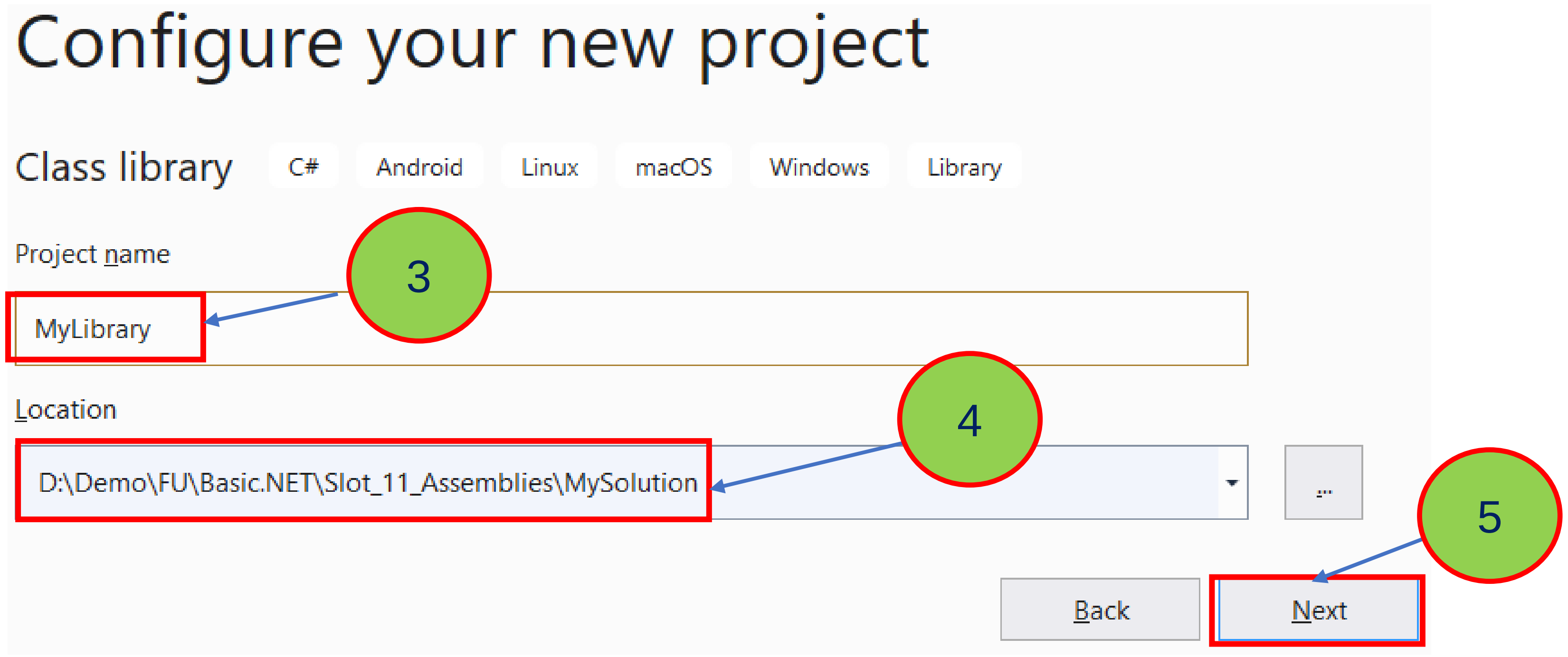
C# | All platforms | All project types

Recent project templates

Console Application	C#
Worker Service	C#
Windows Forms App (.NET Framework)	C#
WPF Application	C#

Console Application
A project for creating a command-line application that can run on .NET Core on Windows, Linux and macOS
C# Linux macOS Windows Console

Class library
A project for creating a class library that targets .NET Standard or .NET Core
C# Android Linux macOS Windows Library



Configure your new project

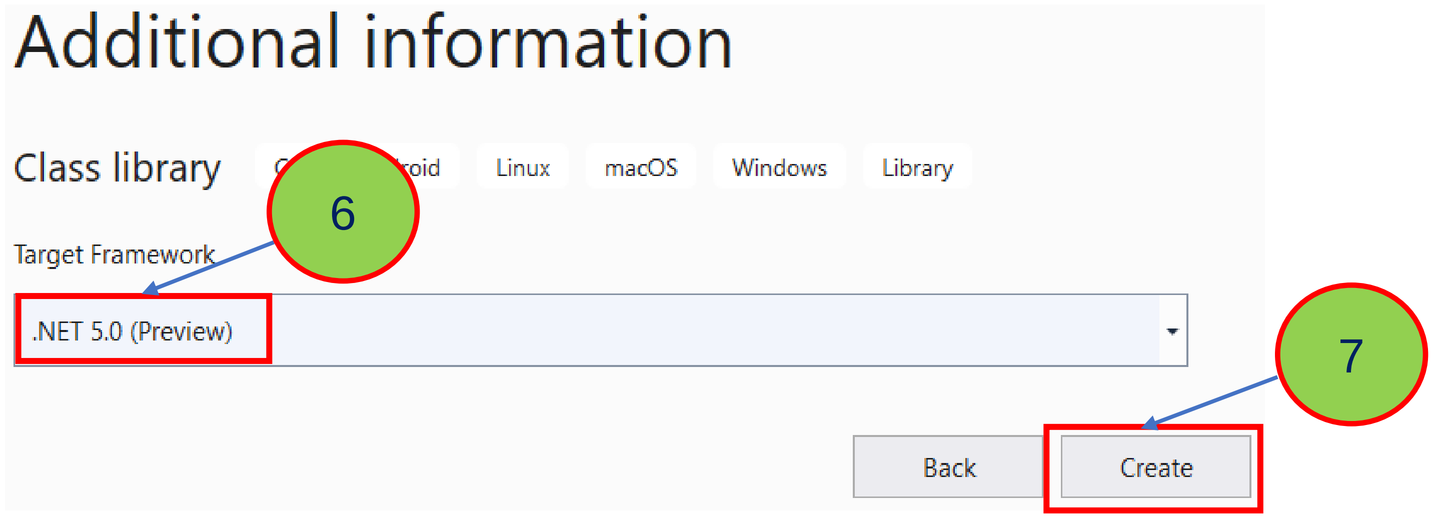
Class library C# Android Linux macOS Windows Library

Project name

MyLibrary

Location

D:\Demo\FU\Basic.NET\Slot_11_Assemblies\MySolution



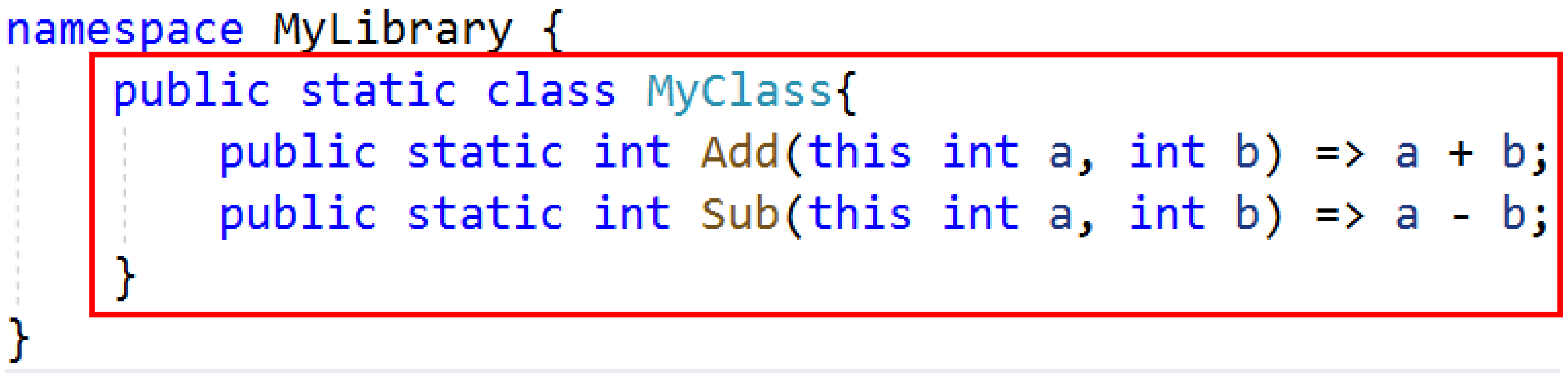
Additional information

Class library C# Android Linux macOS Windows Library

Target Framework

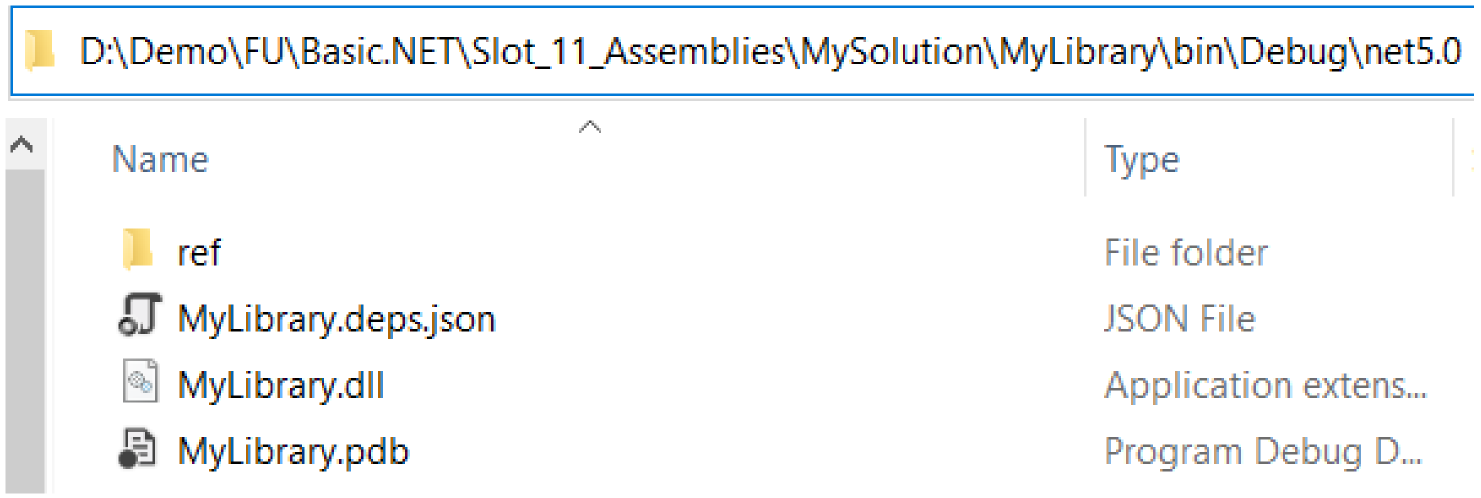
.NET 5.0 (Preview)

4. On the **MyLibrary** project, rename **Class1.cs** to **MyClass.cs** then write the codes as follows :



```
namespace MyLibrary {
    public static class MyClass{
        public static int Add(this int a, int b) => a + b;
        public static int Sub(this int a, int b) => a - b;
    }
}
```

5. Right-click on the **MyLibrary** project, select **Build** to compile to **MyLibrary.dll**

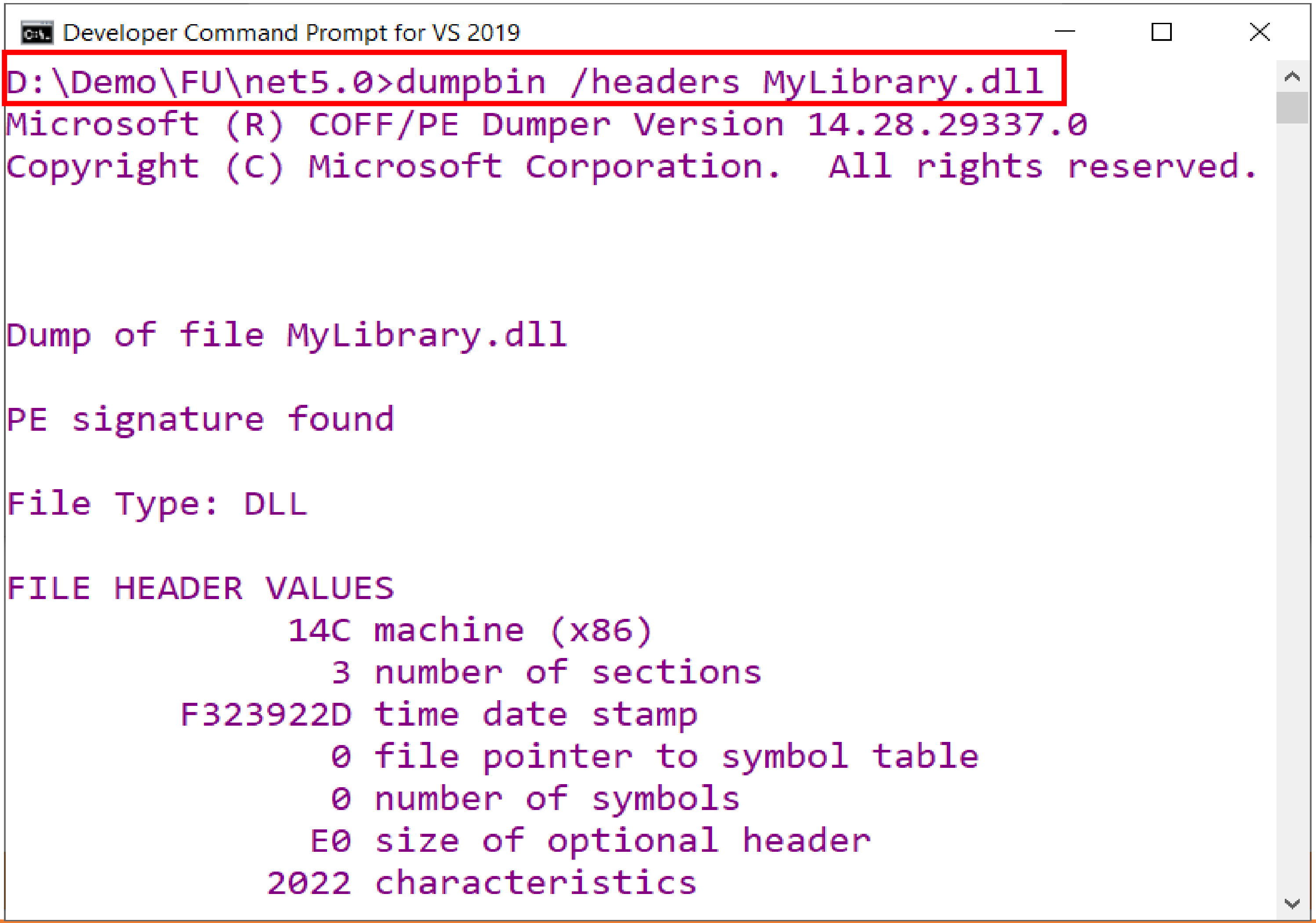


D:\Demo\FU\Basic.NET\Slot_11_Assemblies\MySolution\MyLibrary\bin\Debug\net5.0

Name	Type
ref	File folder
MyLibrary.deps.json	JSON File
MyLibrary.dll	Application extens...
MyLibrary.pdb	Program Debug D...

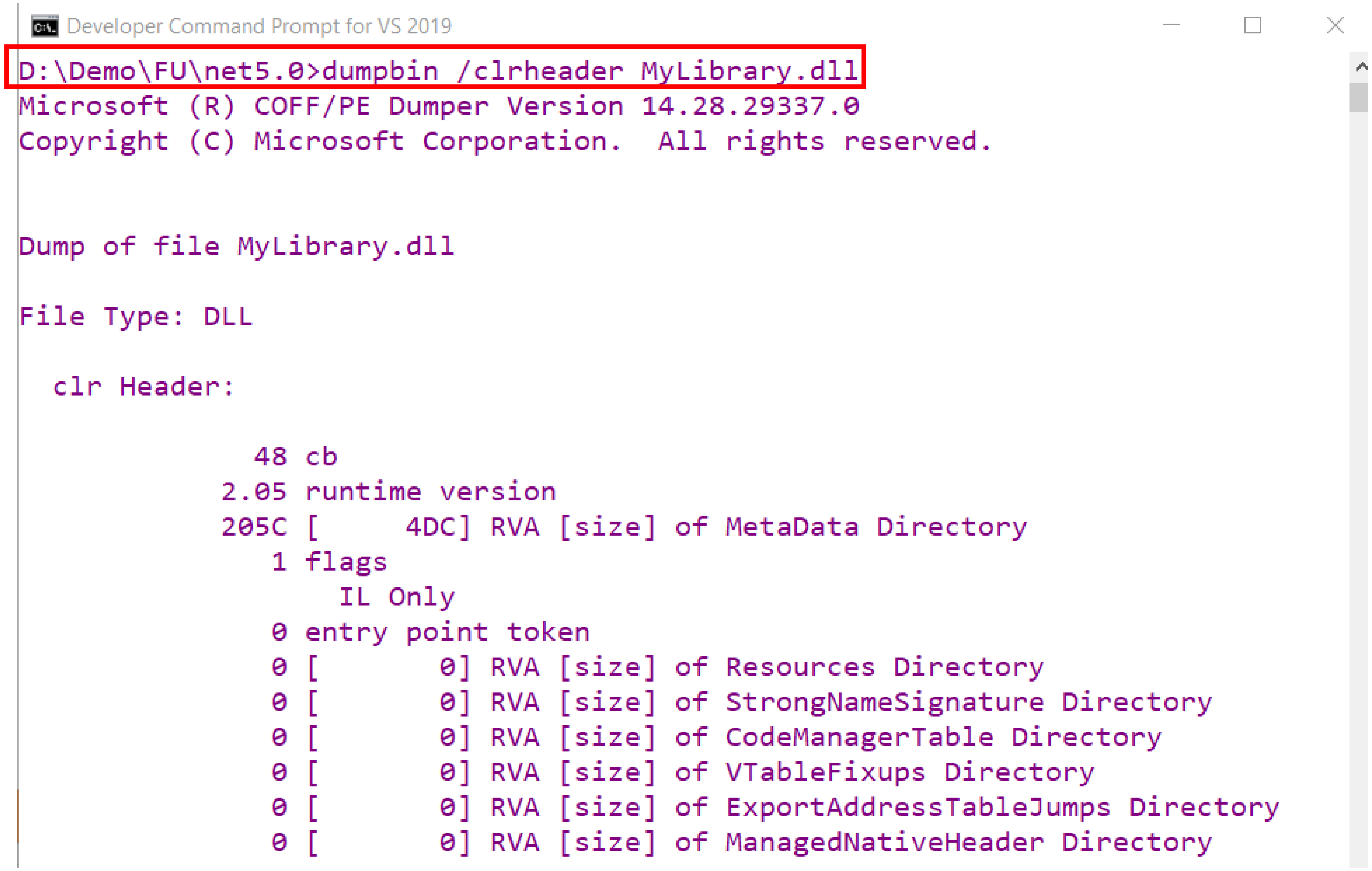
6. To view **Operating system file header** in the **MyLibrary.dll**

- Open **Developer Command Prompt for VS 2019,** use **dumpbin** command



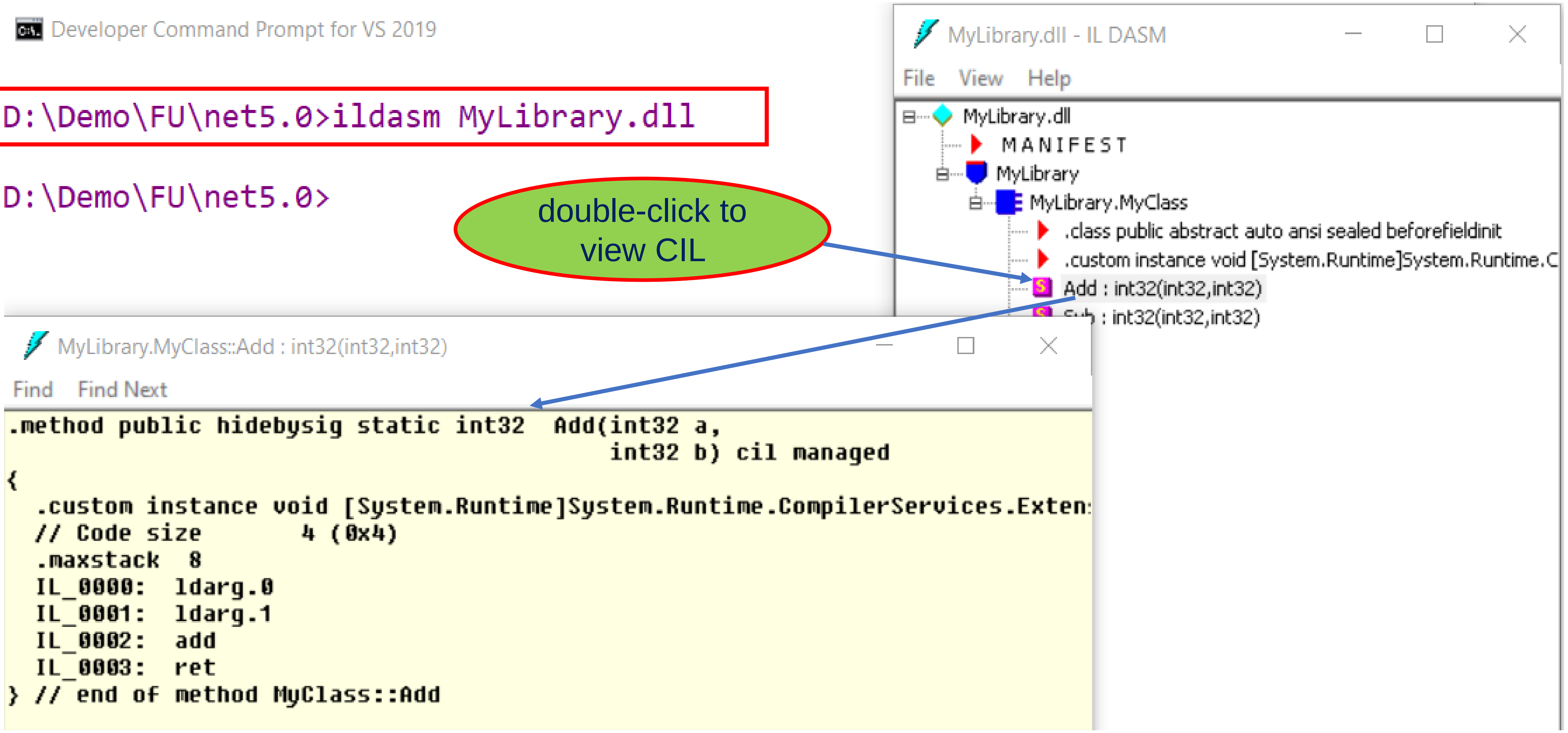
```
D:\Demo\FU\net5.0>dumpbin /headers MyLibrary.dll
Microsoft (R) COFF/PE Dumper Version 14.28.29337.0
Copyright (C) Microsoft Corporation.  All rights reserved.



Dump of file MyLibrary.dll

PE signature found

File Type: DLL

FILE HEADER VALUES
             14C machine (x86)
               3 number of sections
        F323922D time date stamp
               0 file pointer to symbol table
               0 number of symbols
              E0 size of optional header
            2022 characteristics
```

7. To view **CLR file header** in the **MyLibrary.dll**



```
D:\Demo\FU\net5.0>dumpbin /clrheader MyLibrary.dll
Microsoft (R) COFF/PE Dumper Version 14.28.29337.0
Copyright (C) Microsoft Corporation.  All rights reserved.


Dump of file MyLibrary.dll

File Type: DLL

  clr Header:

              48 cb
            2.05 runtime version
            205C [     4DC] RVA [size] of MetaData Directory
               1 flags
                   IL Only
               0 entry point token
               0 [       0] RVA [size] of Resources Directory
               0 [       0] RVA [size] of StrongNameSignature Directory
               0 [       0] RVA [size] of CodeManagerTable Directory
               0 [       0] RVA [size] of VTableFixups Directory
               0 [       0] RVA [size] of ExportAddressTableJumps Directory
               0 [       0] RVA [size] of ManagedNativeHeader Directory
```

8. To view **CIL Code** in the **MyLibrary.dll,** use **ildasm** tool

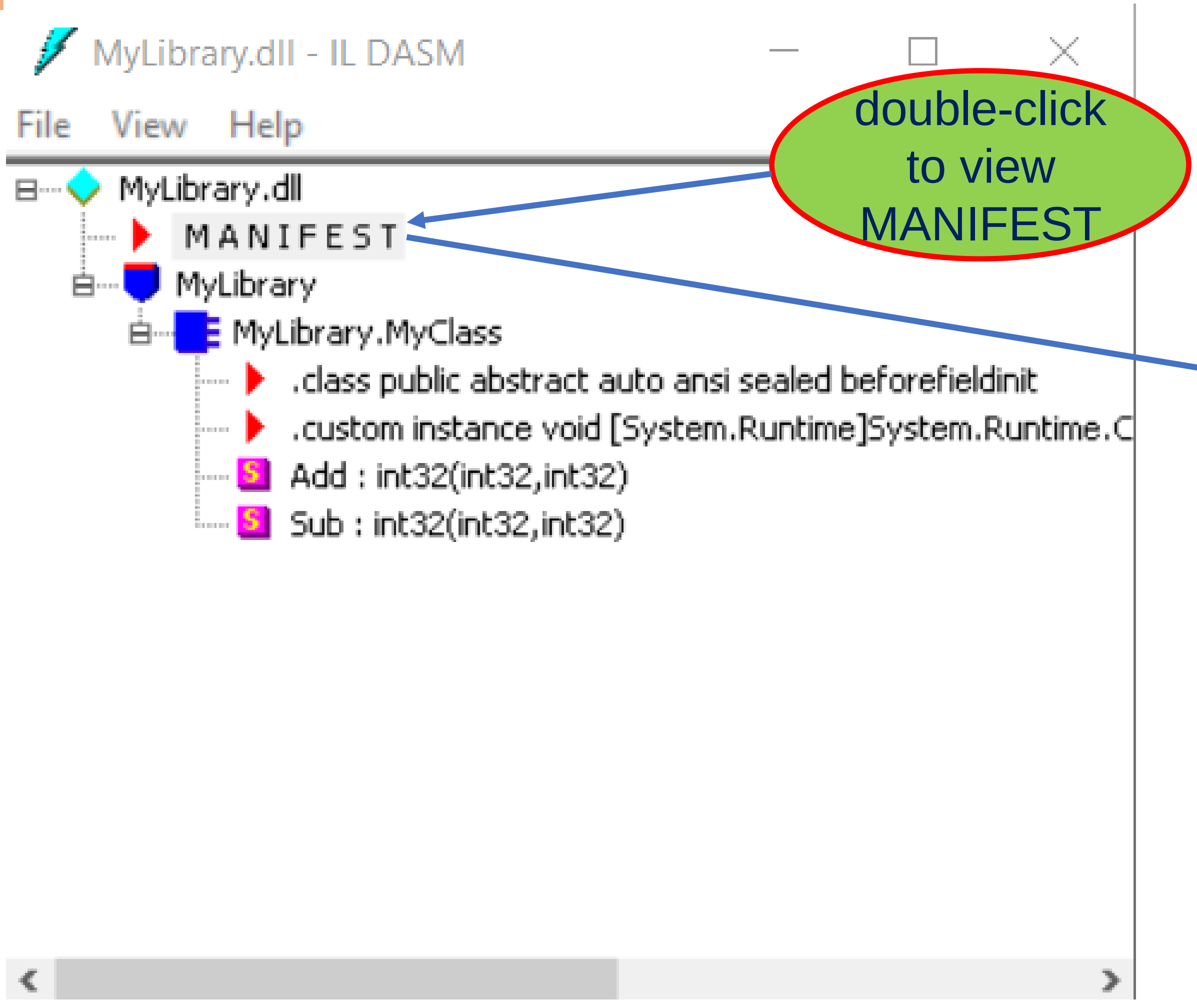

Developer Command Prompt for VS 2019
MyLibrary.dll - IL DASM
File View Help
double-click to view CIL
MyLibrary.MyClass::Add : int32(int32,int32)
Find Find Next

```
D:\Demo\FU\net5.0>ildasm MyLibrary.dll

D:\Demo\FU\net5.0>
```

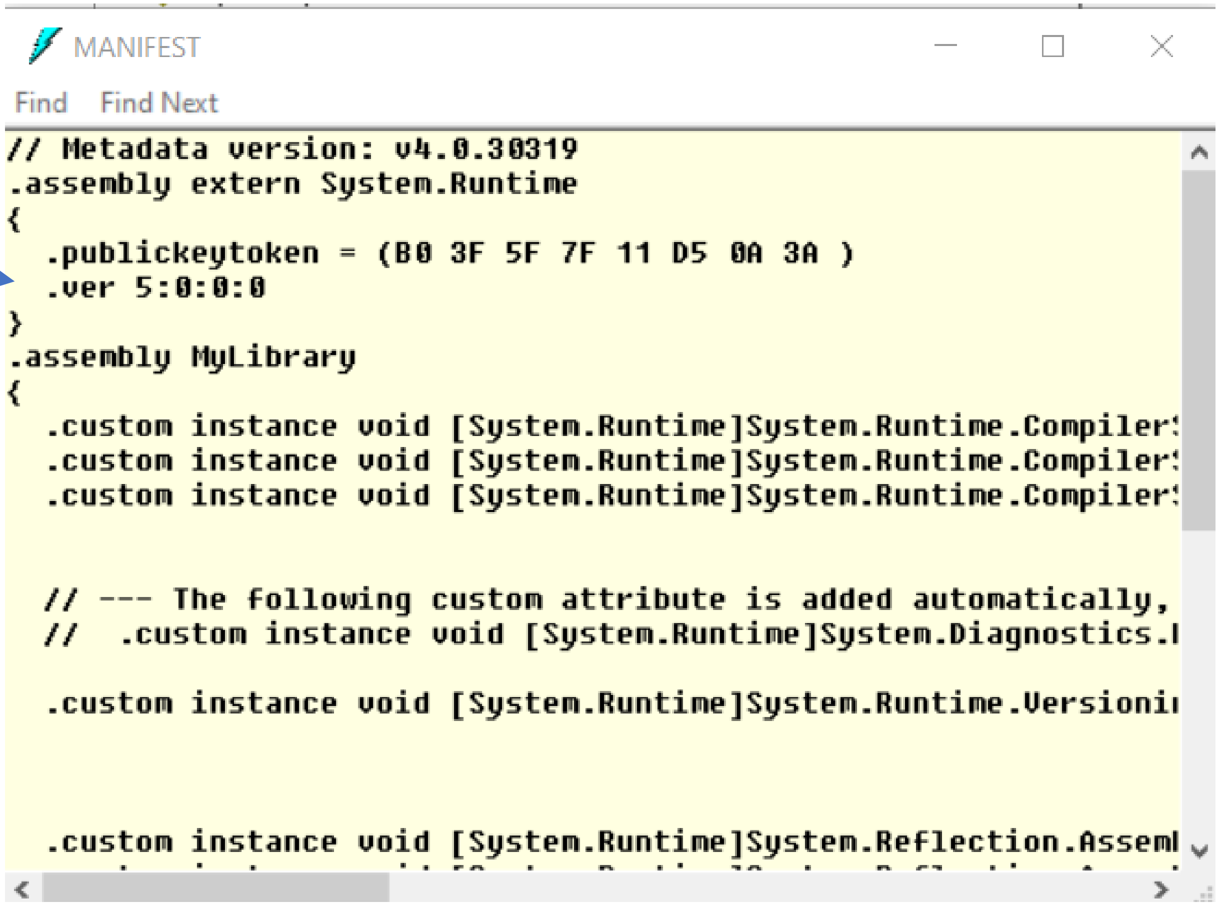
```
MyLibrary.dll
    MANIFEST
    MyLibrary
        MyLibrary.MyClass
            .class public abstract auto ansi sealed beforefieldinit
            .custom instance void [System.Runtime]System.Runtime.C
            Add : int32(int32,int32)
            Sub : int32(int32,int32)
```

```
.method public hidebysig static int32  Add(int32 a,
                                           int32 b) cil managed
{
  .custom instance void [System.Runtime]System.Runtime.CompilerServices.Exten
  // Code size       4 (0x4)
  .maxstack  8
  IL_0000:  ldarg.0
  IL_0001:  ldarg.1
  IL_0002:  add
  IL_0003:  ret
} // end of method MyClass::Add
```

9. View **MANIFEST** in the **MyLibrary.dll**

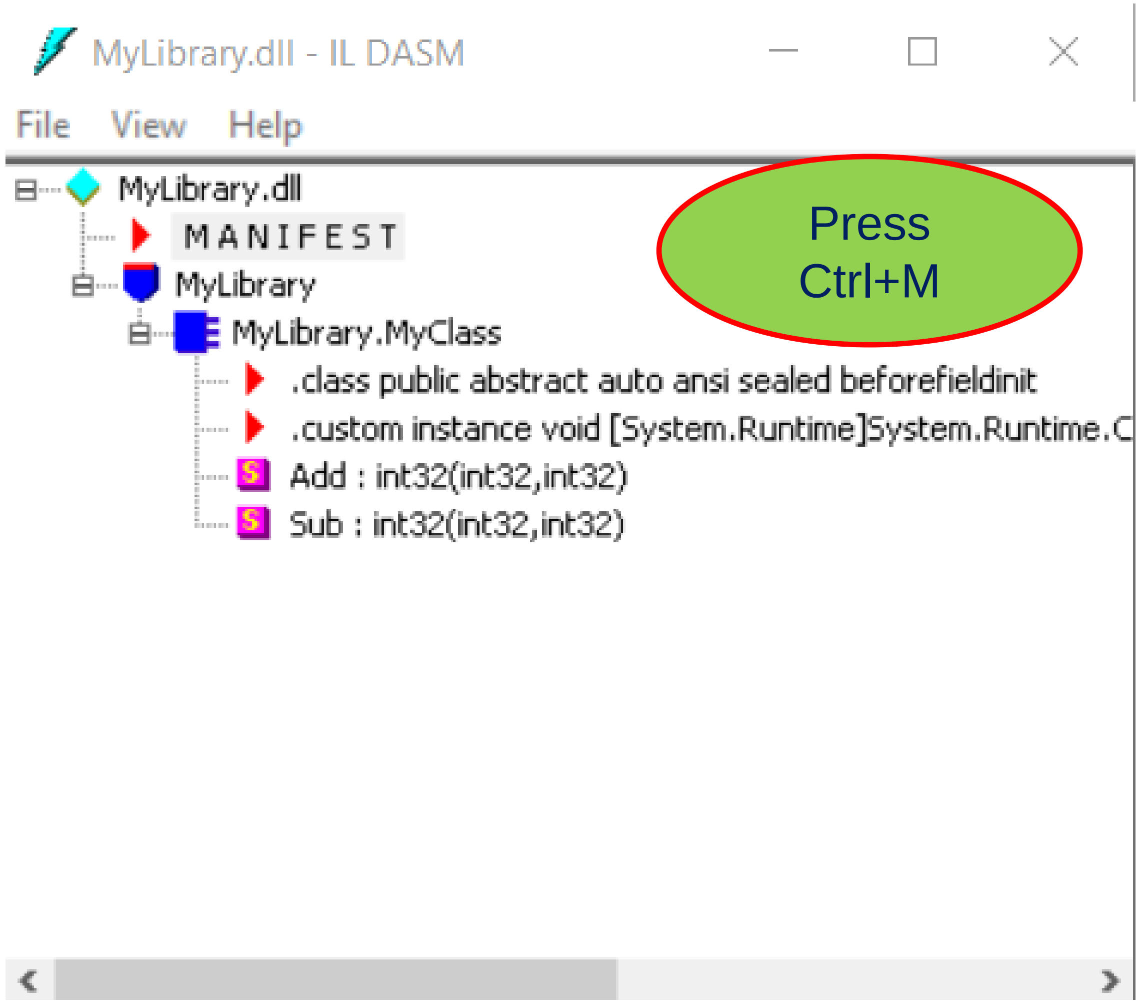


```
MyLibrary.dll
  MANIFEST
  MyLibrary
    MyLibrary.MyClass
      .class public abstract auto ansi sealed beforefieldinit
      .custom instance void [System.Runtime]System.Runtime.C
      Add : int32(int32,int32)
      Sub : int32(int32,int32)
```

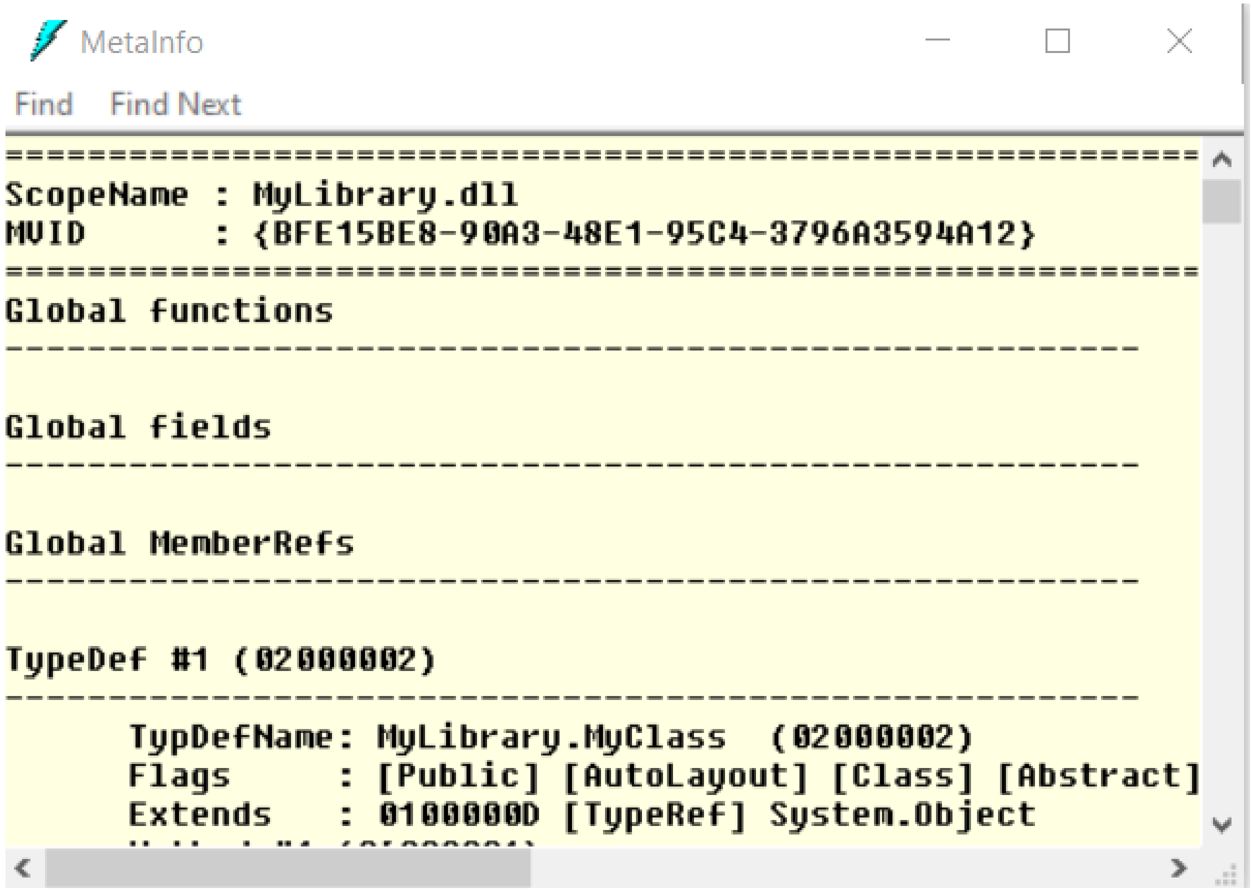


```
// Metadata version: v4.0.30319
.assembly extern System.Runtime
{
  .publickeytoken = (B0 3F 5F 7F 11 D5 0A 3A )
  .ver 5:0:0:0
}
.assembly MyLibrary
{
  .custom instance void [System.Runtime]System.Runtime.Compiler
  .custom instance void [System.Runtime]System.Runtime.Compiler
  .custom instance void [System.Runtime]System.Runtime.Compiler


  // --- The following custom attribute is added automatically,
  //  .custom instance void [System.Runtime]System.Diagnostics.I

  .custom instance void [System.Runtime]System.Runtime.Versioni



  .custom instance void [System.Runtime]System.Reflection.Assem
```

10. View **Metadata** in the **MyLibrary.dll**



```
MyLibrary.dll
    MANIFEST
    MyLibrary
        MyLibrary.MyClass
            .class public abstract auto ansi sealed beforefieldinit
            .custom instance void [System.Runtime]System.Runtime.C
            Add : int32(int32,int32)
            Sub : int32(int32,int32)
```



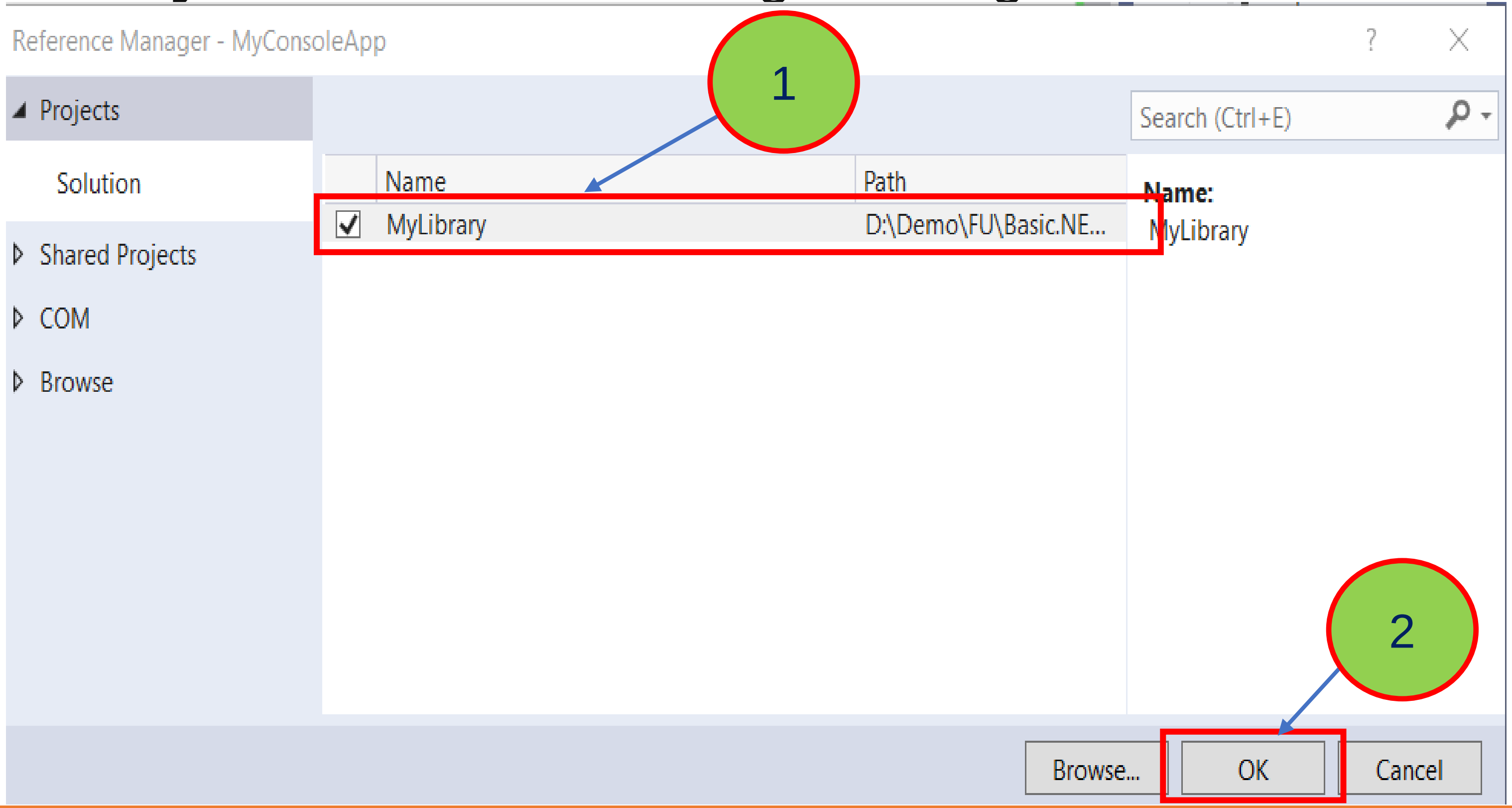
```
===========================================================
ScopeName : MyLibrary.dll
MVID      : {BFE15BE8-90A3-48E1-95C4-3796A3594A12}
===========================================================
Global functions
-------------------------------------------------------

Global fields
-------------------------------------------------------

Global MemberRefs
-------------------------------------------------------

TypeDef #1 (02000002)
-------------------------------------------------------
      TypDefName: MyLibrary.MyClass  (02000002)
      Flags     : [Public] [AutoLayout] [Class] [Abstract]
      Extends   : 0100000D [TypeRef] System.Object
```

TRƯỜNG ĐẠI HỌC FPT

Consuming Assemblies with C# Console Application Demonstration

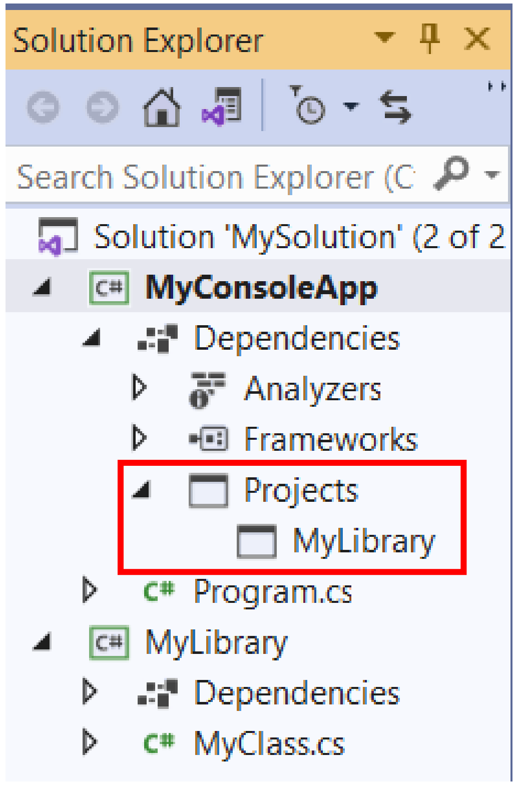
1. Add to **MySolution** a C# Console application named **MyConsoleApp.** After creating a **MyConsoleApp** project, right-click this project and select **Set as Startup Project**
2. Reference **MyLibrary** to the **MyConsoleApp** application

- Right-click on the **MyConsoleApp** project, and choose Add | Project Reference | select **MyLibrary** on Reference Manager dialog as follows:



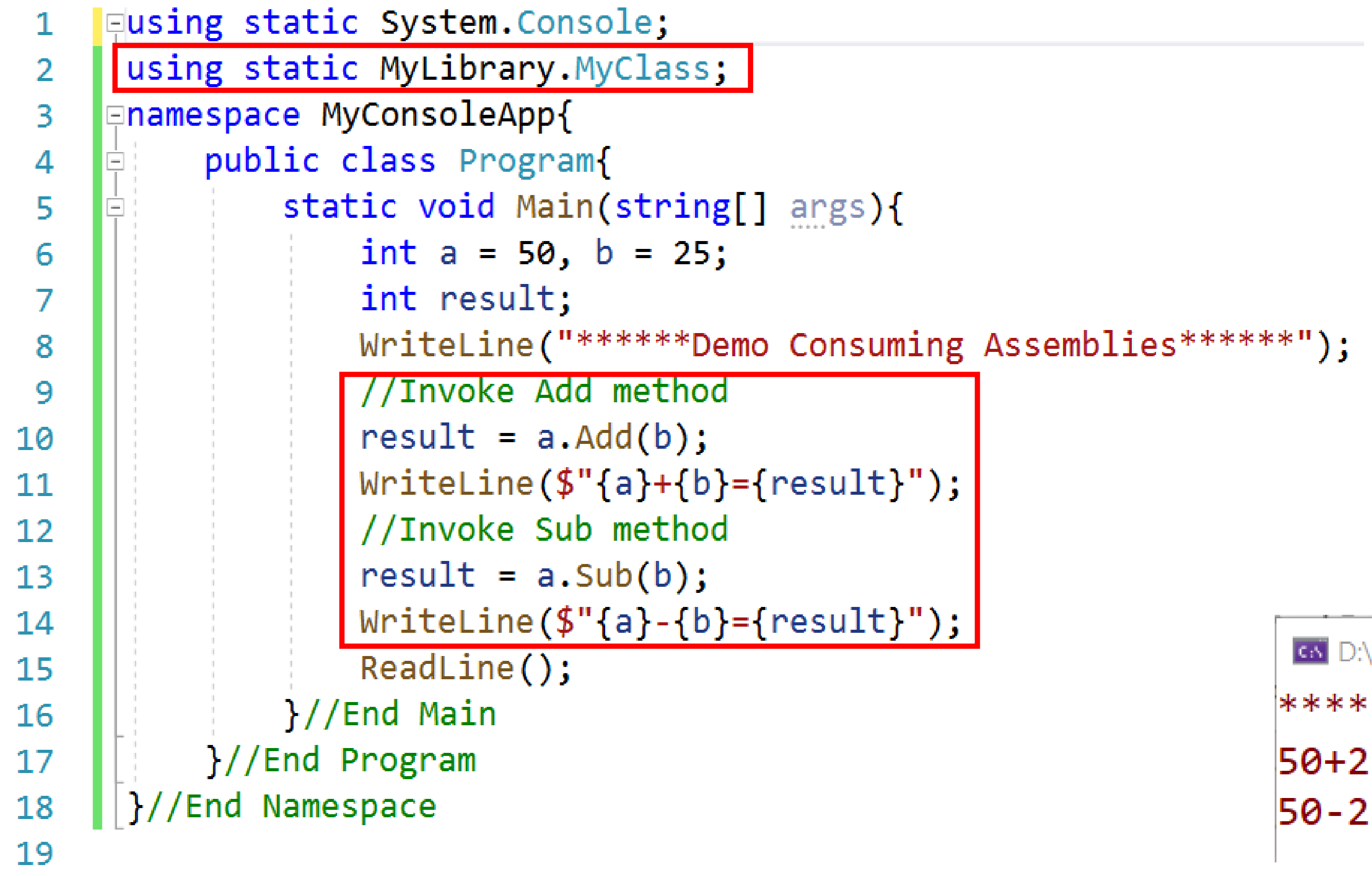
	Name	Path
☑	MyLibrary	D:\Demo\FU\Basic.NE...

Name:
MyLibrary

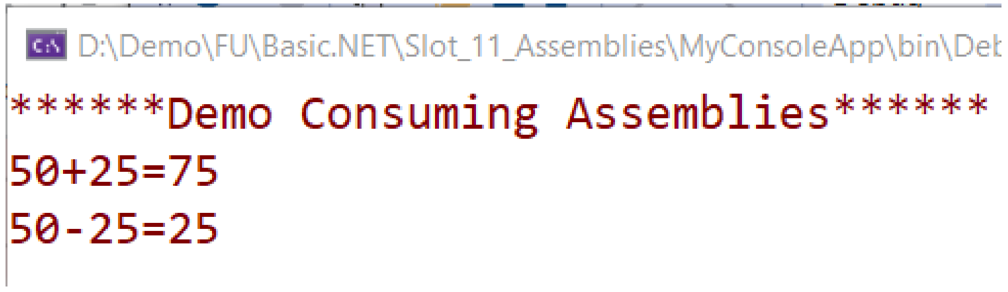


TRƯỜNG ĐẠI HỌC FPT

3. In the **MyConsoleApp** project , write the codes in **Program.cs** then **Run**



```
using static System.Console;
using static MyLibrary.MyClass;
namespace MyConsoleApp{
    public class Program{
        static void Main(string[] args){
            int a = 50, b = 25;
            int result;
            WriteLine("******Demo Consuming Assemblies******");
            //Invoke Add method
            result = a.Add(b);
            WriteLine($"{a}+{b}={result}");
            //Invoke Sub method
            result = a.Sub(b);
            WriteLine($"{a}-{b}={result}");
            ReadLine();
        }//End Main
    }//End Program
}//End Namespace
```



```
D:\Demo\FU\Basic.NET\Slot_11_Assemblies\MyConsoleApp\bin\Det
******Demo Consuming Assemblies******
50+25=75
50-25=25
```

Summary

- Concepts were introduced:
  - What is the Assemblies in .NET?
  - Explain components in .NET Assemblies: Manifest, Metadata, CIL and Resources
  - Explain Role of .NET Assemblies
  - Explain types .NET Assemblies : Static and Dynamic
  - Explain and demo about view CIL Code assemblies by ildasm tool
  - Explain demo about dumpbin tool
  - Demo create Assemblies and consume with C# Console Application