Dynamic Link Libraries (DLLs) and Class Libraries in C#

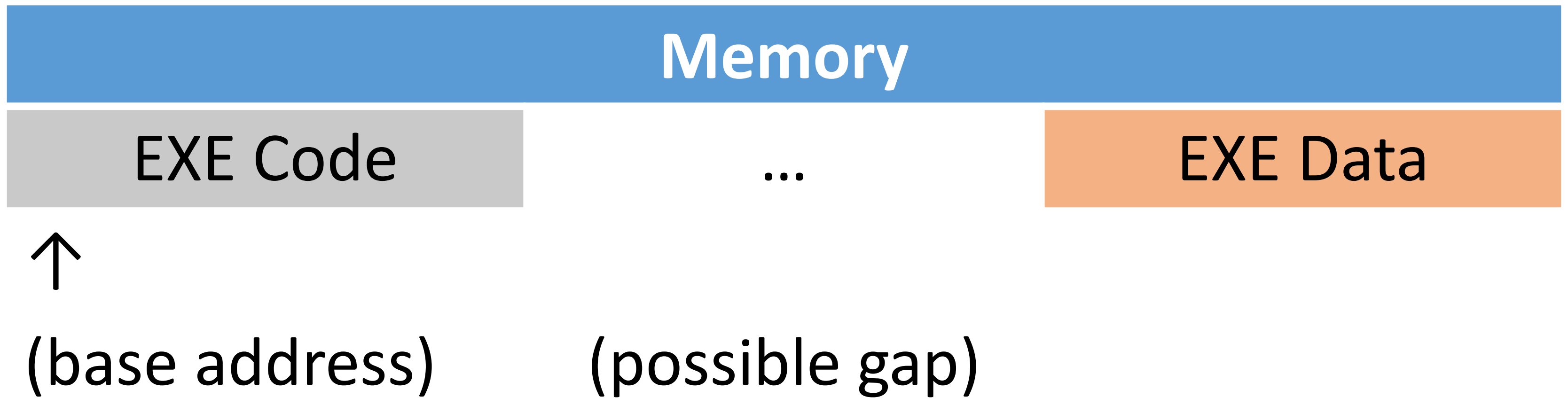
Cpt S 321

Washington State University

Executable Files

- To understand DLLs, you first need to understand the basics of what happens when you build an executable file and run it
- The executable file produced by compiling your code consists of **code** (obviously) and **data** (values of hard-coded variables; image, text, and other files that are embedded within the executable)
  - Check the "**bin**" folder of one of your projects
- The operating system needs to load both the code and data into memory when the <u>application is run</u>

OS Loads EXE into Memory

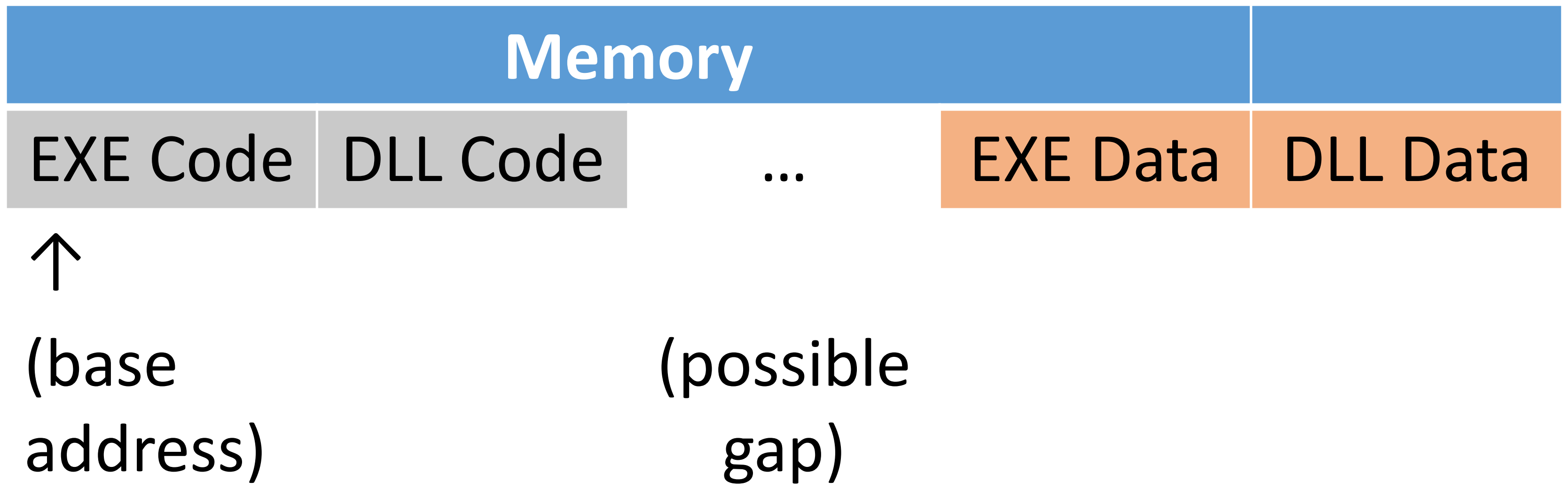


The OS loads the code and data into memory. While the code is likely packed into one sequential block and the data is packed into one sequential block, the two are not necessarily back-to-back in memory (they might or might not be, it’s the operating system’s choice).

DLLs

- DLLs are much like executable files in that they contain **code** and **data**
- The key difference is that an executable can be executed on its own and the DLL cannot
- The DLL gets loaded by a running application and then the code and data from the DLL is accessible to that application

OS Loads DLL into Memory



The OS loads the DLL into memory. It’s an executable that makes the call to load the DLL at runtime, so the OS appends the code and data onto the code and data segments of application memory, respectively. The **library** is **linked** in **dynamically** at runtime to the application’s memory space.

The Short Story on DLLs

- Exist on all major operating systems in some form (just called *dynamic libraries* or sometimes *shared libraries* in Linux and Mac OS)
- Provide a way to <u>extend application functionality at runtime</u> (plug-ins / add-ins)
- Provide access to <u>shared functionality for multiple applications</u>
  - Applications “talk” to the operating system, for example, when they want to create a Window
  - The OS creates the window and manages certain aspects of it, such as sending input events to it when it has focus
  - Applications access this functionality through an operating system DLL (this happens automatically in WinForms applications)

The C/C++ Way of Using DLLs

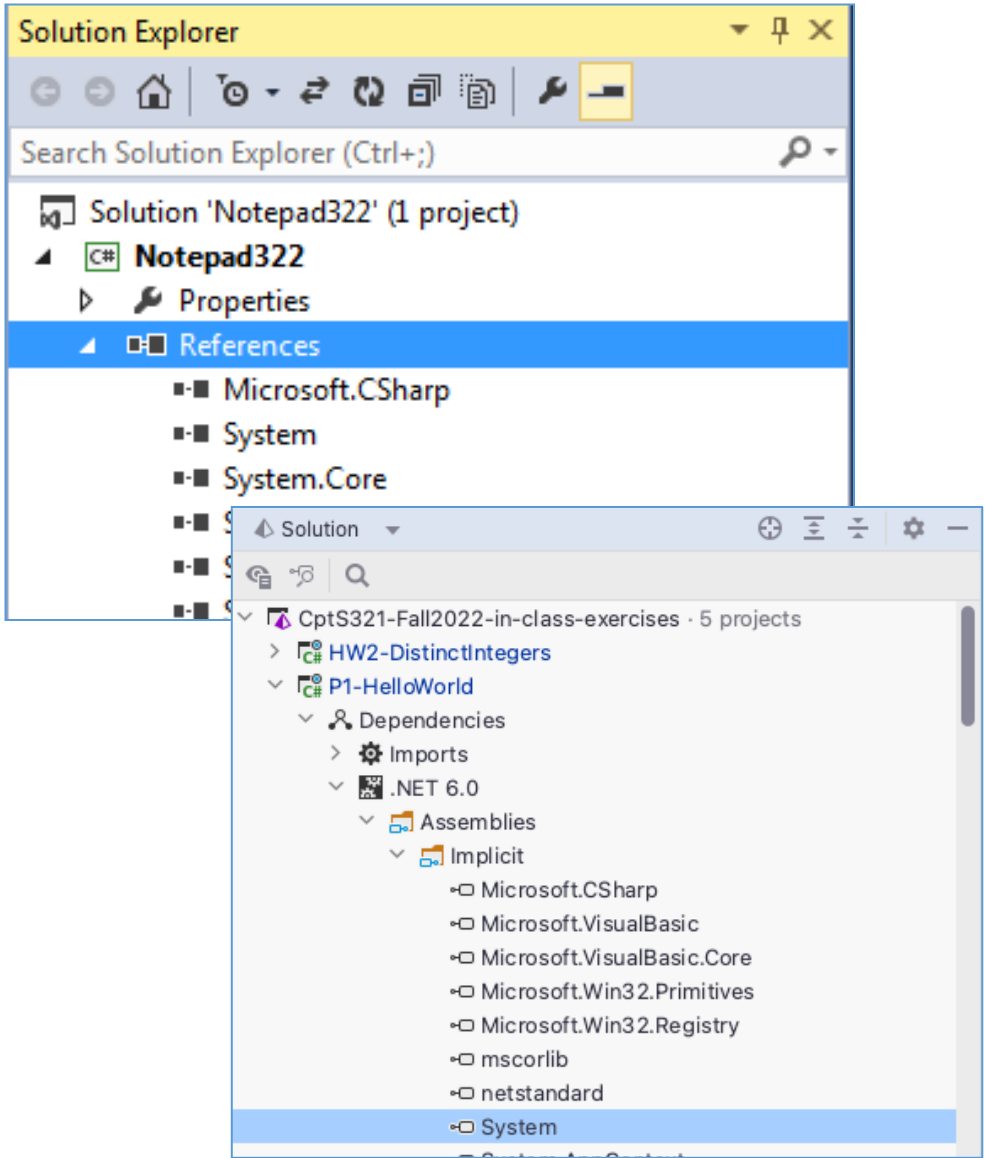
- Open the DLL at runtime
  - “LoadLibrary” function in Windows API
  - “dlopen” in Mac OS and Linux
- Get the address of an “exported function” in that DLL
  - “GetProcAdress” in Windows API
  - “dlsym” in Mac OS and Linux
- Cast a function pointer to the returned value and call the function when desired
  - Requires the application to know the function signature (type and number of parameters)
  - The interface between the application and the DLL is fragile and primitive. You only get an address from the DLL and then have to do the work to gather all exported functionality and call it as needed.
  - Can statically-link against DLLs and somewhat simplify some of this, but it’s still not as clean as what C# offers

The C# DLLs: Class Libraries

- You can make class library projects in C#, which compile to .DLL files
- These are exactly what they sound like: a library of C# classes declared in the DLL code. These classes can be used by applications just as if they were classes declared in the application’s code.

The C# DLLs: Class Libraries

- The “References” or “Dependencies” section in the solution explorer contains references to class libraries
- As soon as you’ve referenced one of these in your project, you’ll have access to all the <u>public</u> classes declared within that library



Solution Explorer

Search Solution Explorer (Ctrl+;)

```
Solution 'Notepad322' (1 project)
  Notepad322
    Properties
    References
      Microsoft.CSharp
      System
      System.Core
```

Solution

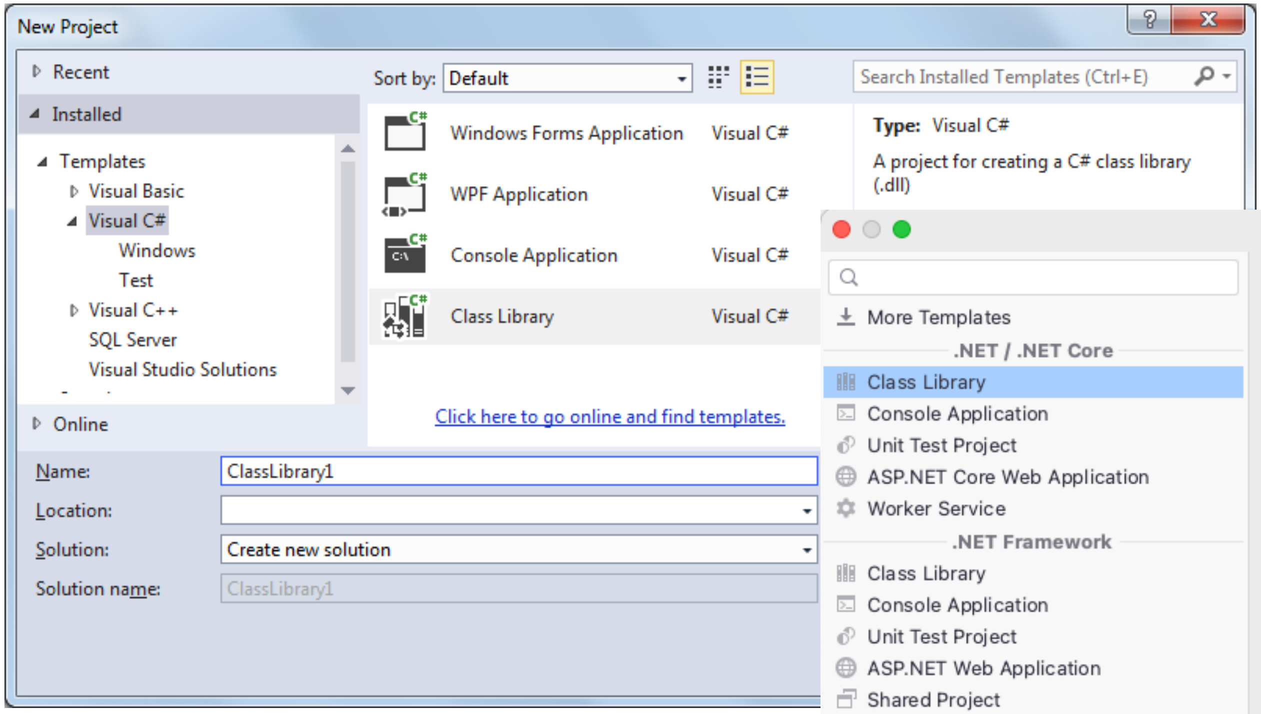
```
CptS321-Fall2022-in-class-exercises · 5 projects
  HW2-DistinctIntegers
  P1-HelloWorld
    Dependencies
      Imports
      .NET 6.0
        Assemblies
          Implicit
            Microsoft.CSharp
            Microsoft.VisualBasic
            Microsoft.VisualBasic.Core
            Microsoft.Win32.Primitives
            Microsoft.Win32.Registry
            mscorlib
            netstandard
            System
```

The C# DLLs: Class Libraries

- The other way to utilize a class library is to load it via code. This is the way you need to do it if you're designing something like a plug-in/add-in framework for your application. The add-in will be written by some other person after you've released your software, so you can't possibly reference it in the solution explorer before compiling.
- The Assembly class in the System.Reflection namespace represents a DLL that you can load at runtime
  - Use Assembly.LoadFile (a static method that returns an Assembly object) to load the DLL at runtime
- After loading it you need a way to actually use classes/structs that are publically declared within the library. You use a powerful language mechanism called "reflection" to do this, which we will talk about at a later date.

Building a Class Library

- As mentioned earlier, there is a class library project type



New Project

Recent
Installed
Templates
Visual Basic
Visual C#
Windows
Test
Visual C++
SQL Server
Visual Studio Solutions
Online

Sort by: Default

Search Installed Templates (Ctrl+E)

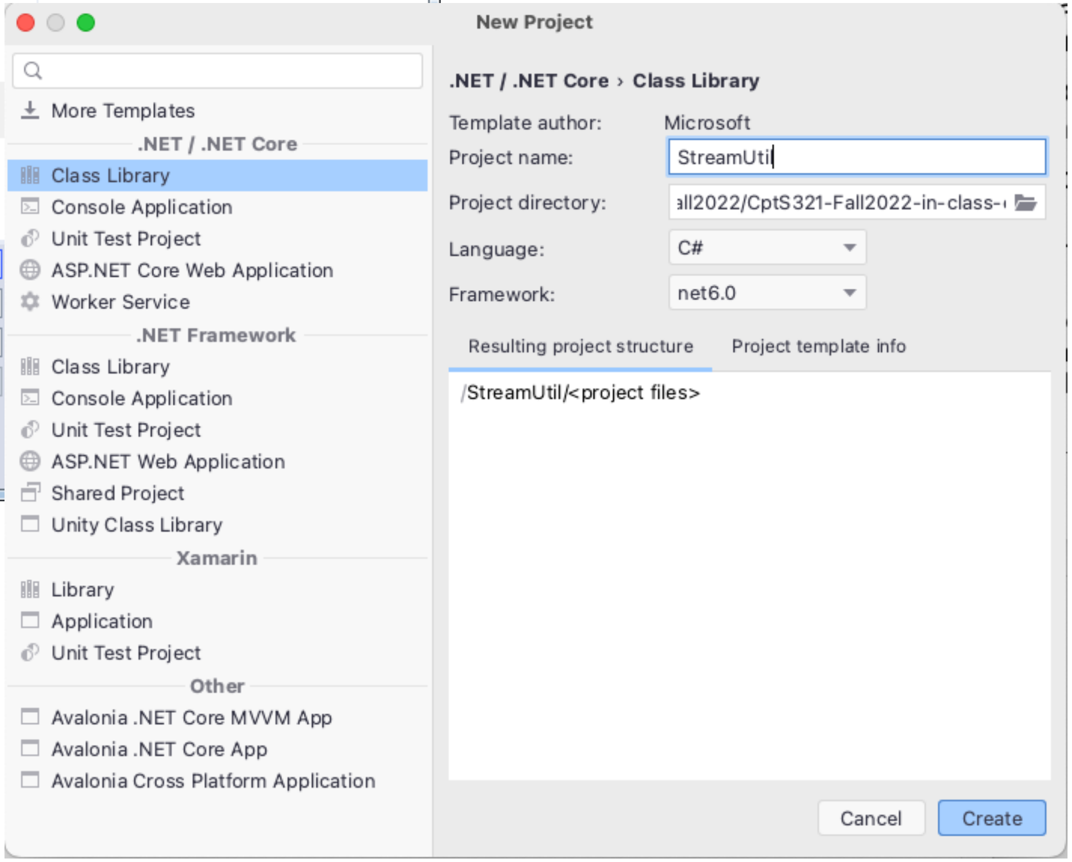
Windows Forms Application	Visual C#
WPF Application	Visual C#
Console Application	Visual C#
Class Library	Visual C#

Click here to go online and find templates.

Type: Visual C#

A project for creating a C# class library (.dll)

Name:	ClassLibrary1
Location:	
Solution:	Create new solution
Solution name:	ClassLibrary1



New Project

More Templates

.NET / .NET Core
- Class Library
- Console Application
- Unit Test Project
- ASP.NET Core Web Application
- Worker Service

.NET Framework
- Class Library
- Console Application
- Unit Test Project
- ASP.NET Web Application
- Shared Project
- Unity Class Library

Xamarin
- Library
- Application
- Unit Test Project

Other
- Avalonia .NET Core MVVM App
- Avalonia .NET Core App
- Avalonia Cross Platform Application

.NET / .NET Core › Class Library

Template author:	Microsoft
Project name:	StreamUtil
Project directory:	all2022/CptS321-Fall2022-in-class-
Language:	C#
Framework:	net6.0

Resulting project structure | Project template info

/StreamUtil/<project files>

Cancel | Create

Building a Class Library

- **COHESION**: Put whatever code you want in the class library – as long as it is cohesive!
- **REUSABILITY**: It's a way to put shared functionality in one spot and then easily have multiple applications use it
- **INFORMATION HIDING**:
  - Remember that only publically declared classes, structures and enumerations can be seen and used by projects that reference the class library
  - You can declare non-public classes to be used internally within the library, but you can't have any public type with a public function that returns one of these

Building a Class Library

- How do we declare a class within the class library that can be used by all other classes in the library, but is not visible by projects that reference the class library?
- You can’t declare the class as private or protected because then other classes wouldn’t be able to see it. You will actually get a compiler error if declaring a non-nested private or protected class.
- You can’t declare it as public because then, while other classes in the class library will be able to see and use it, so will anything that references the class library.

C# “internal” Access Modifier

- Remember that “**internal**” access modifier? We mentioned earlier that it is a new one (when comparing with C++)

- Now is when we see why it exists: internal means it is public within the class library, but is not visible to projects that reference the library. It’s public within the DLL only.

Time for coding!

- Remember our last coding exercise? Let’s transform the code into a class library to promote reusability:

1. Create a Class Library project in which you extract the functionality that you want to make reusable. I would call it **StreamUtil**.
2. Create another project in which you would use the previously created library. I called mine **DLLwithStreamsDemo**.
3. We need to connect the Demo to the library

   - Option A: Typically, we add the **.dll** of the library to the demo project and this is what we want to do for this exercise:
     - VS: Right click on the project -> add reference -> browse the .dll file
     - Avalonia: Right click on the project -> Add -> Add Reference.. And then click on ”Add from…”-> browse the folder and select the .dll file

   - Option B: **Not for this exercises but …** if you are developing the library at the same time as your demo project, it is more practical to add reference to the library project during the development the way we did for previous exercises by right click on the project -> add project reference -> select the StreamUtil project.