**C# OOP**

1. **Architecture of .NET applications**

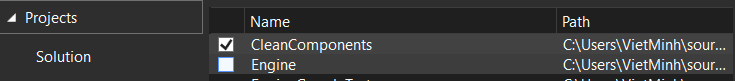
* Solutions 🡪 contains multiple projects
* Project 🡪 develops one or more Assemblies.
* Assembly (.dll or .exe) 🡪 container for related namespaces
  + .dll is a library, cannot be executed by itself
  + .exe is a executable that reserve threads and run
  + When compiling, the compiler builds one or more Assemblies.
* Namespace 🡪 Container for related classes, structs, interfaces, enums, and delegates

(in C++, we #include libraries containing multiple namespaces)

* Class (and enum…): Data (Attributes) & Methods (Functions)

1. **Connecting Projects (References & Dependencies)**

Since every types developed are namespaces, classes, etc (OOP), we can easily “pass” them to other files

* Right click the “Reference” (or “Dependencies”) tab then select “Add Reference”
* Choose the project to be referenced from and check the box
* A screenshot of a computer

  Description automatically generated with medium confidenceYou can then use all types defined in that project with “using”

Difference between References and Dependencies: Reference is old framework, Dependency is new. However, they can be represented differently.

* If A **refers** to B, A “uses” a namespace/type from B.
* If A **depends** on B, B needs to be built before A (not necessarily a reference, e.g., artifacts produced by the build of another project)
* Sets up Build Order

A screenshot of a computer

Description automatically generated with medium confidence

1. **C# Program**

A black background with white text

Description automatically generated with low confidence

* **using directive** 🡪 includes other **namespaces/types**
  + using System (basic utility classes)
  + using System.Collections.Generic (lists & collections…)
  + using System.Threading.Tasks (multithreading)
  + using namespace from referenced projects/.dll files
* You can also create an alias for a namespace/type with a *using <alias> = <class\_directive>*
* Everything in C# are classes, even including the program (*internal class Program*)
* main function is within the Program class:

static void Main (string[] args)

1. **\*\*\* Access Modifiers \*\*\* - For class members**

The following seven accessibility levels can be specified using the access modifiers:

|  |  |
| --- | --- |
| public | Access isn't restricted. |
| protected | Access is limited to the containing class or types derived from the containing class. |
| internal | Access is limited to the current assembly. |
| protected internal | Access is limited to the current assembly or types derived from the containing class. |
| private | Access is limited to the containing type. |
| private protected | Access is limited to the containing class or types derived from the containing class within the current assembly. |
| file | The declared type is only visible in the current source file. File scoped types are generally used for source generators. |

1. **\*\*\* C# Types \*\*\***

|  |  |
| --- | --- |
| Class | Contains: Variables, Methods, Inheritance, Polymorphism |
| Struct | Contains: Variables, Methods (lighter than Class) |
| Interface | **Contains: only methods & properties.** Any class or struct that implements an Interface must provide an implementation of all members defined in the interface.   * Interface has no member data, no constructors…. Abstract class have all * A class can implement more than one interface but extend only one class |
| Enum | Assign integers to related string ‘values’ 🡪 Protects what’s being passed into a method, or assigned to a variable (cannot use random values & break the code) |
| Delegates | Uses: function pointer, foundation for events  Every event in .NET must be based on a delegate. |

1. **Properties**

A member of the class that help you read, write, or compute value of a private field 🡪 Promote safety

A screen shot of a computer

Description automatically generated with medium confidenceCommon scenario: having a **private backing field** (private member)

A picture containing text, screenshot, font

Description automatically generated

A picture containing text, font, screenshot, number

Description automatically generated\*\*If the property simply get and set without logic a private field, we can have a **short form**

🡪 Backing field is created anonymously.

1. **where (generic type T constraint)**

The **where** clause in a generic definition specifies **constraints on the types that are used as arguments** for type parameters. Constraints can specify interfaces, base classes, or require a generic type to be a reference, value, or unmanaged type. They declare capabilities that the type argument must have, and must be placed after any declared base class or implemented interfaces.

e.g., requiring that T implements the IComparable<T> interface

public class AGenericClass<T> where T : IComparable<T> { }