Assembly trong C#, thư viện class, NuGet

Assembly trong C# (và .NET) là kết quả biên dịch code của mỗi project. Mỗi project đến lượt mình lại có thể tham chiếu (reference) tới các assembly khác để sử dụng các class (và kiểu dữ liệu khác) định nghĩa trong assembly đó. Nói cách khác, "chương trình" C# (và .NET) chính là một assembly được tạo ra bằng cách "ghép nối" nhiều assembly khác. Assembly của .NET có đuôi exe hoặc dll tương tự như các file chương trình hoặc thư viện của windows nhưng bản chất của chúng rất khác biệt.

Bài học này sẽ giới thiệu chi tiết về assembly trong C#, bao gồm quá trình xây dựng, sử dụng (tham chiếu) assembly của bạn và của bên thứ ba. Ngoài ra, trong bài này bạn cũng sẽ làm quen với NuGet – hệ thống quản lý gói thư viện dành cho các nền tảng của Microsoft.

NỘI DUNG CỦA BÀI [Ẩn]

1. Assembly trong C#
   1.1. Khái niệm và phân loại assembly trong C# .NET
   1.2. Vai trò và đặc điểm của assembly trong .NET
2. Sử dụng assembly có sẵn
   2.1. Sử dụng assembly của .NET
   2.2. Sử dụng assembly của bên thứ ba
3. Tự tạo và sử dụng class library trong solution
   3.1. Class Library project
   3.2. Tham chiếu project tới thư viện vừa tạo
   3.3. Một số lưu ý khi tạo thư viện class
4. Sử dụng thư viện từ NuGet, NuGet Packages Manager
   4.1. Giới thiệu về NuGet
   4.2. Cài đặt thư viện NewtonSoft.Json với NuGet Manager
   4.3. Cài đặt gói NuGet sử dụng website kết hợp Package Manager Console
   4.4. Một số lưu ý khi sử dụng các gói thư viện từ NuGet
5. Kết luận

Assembly trong C#

Khái niệm và phân loại assembly trong C# .NET

Trong quá trình học lập trình C# cơ bản từ đầu đến giờ bạn đã xây dựng khá nhiều project. Mỗi project này đều thuộc loại Console App – loại ứng dụng tự chạy độc lập với giao diện dòng lệnh. Trong C# (và .NET) chương trình ứng dụng này được gọi là một assembly.

Ngoài Console App, C# cũng có thể tạo ra nhiều loại project khác, tương ứng với các loại ứng dụng riêng. Kết quả biên dịch mọi project của C# compiler đều được gọi chung là assembly.

> Tên gọi assembly cũng dùng chung cho tất cả kết quả dịch các project viết bằng ngôn ngữ .NET khác như VB.NET, C++ CLI.

Các assembly này chủ yếu chia làm hai loại: một loại có thể chạy độc lập như một chương trình bình thường; một loại chỉ có thể được sử dụng bởi project khác.

Ví dụ, assembly của Console App mà bạn quen thuộc, của Windows Forms, của Windows Presentation Foundation đều thuộc loại thứ nhất. Đặc thù của loại assembly này là có mặt phương thức **static void Main()** với vai trò entry point – điểm khởi đầu của hoạt động của chương trình. Loại assembly này thường có đuôi **exe**, giống như các file chạy thông thường của windows.

Loại assembly thứ hai không chứa static void Main, do đó không thể tự mình chạy như một chương trình độc lập. Loại assembly này được gọi là *class library* (thư viện lớp) hoặc *code library*. Loại assembly này được sử dụng bởi các project khác hoặc chương trình khác, ví dụ, chạy trong IIS – chương trình máy chủ web của Microsoft, hoặc chạy trong hệ thống (ở dạng system service). Loại assembly này có đuôi là **dll** (dynamic link library), tương tự như các file thư viện hệ thống của windows.

Trên thực tế, loại assembly thứ hai rất đông đảo. Chúng tạo ra cả hệ thống thư viện class của .NET. Từ đầu đến giờ thực ra bạn đã liên tục sử dụng mscorlib.dll – một assembly thuộc loại này.

> C# compiler tự động tham chiếu tới thư viện mscorlib.dll trong tất cả các project. Thư viện này chứa tất cả các định nghĩa kiểu cơ sở của .NET.

Về bản chất, exe hay dll assembly không có gì khác biệt nhau. Nghĩa là hoàn toàn có thể coi exe assembly là một class library và tham chiếu tới nó từ project khác.

Assembly chứa mã CIL (Common Intermediate Language) – mã của ngôn ngữ trung gian không phụ thuộc platform. Ở giai đoạn thực thi, mã CIL được JIT (Just-in-time) compiler dịch tiếp thành mã đặc trưng của platform để thực thi. Nói một cách đơn giản, assembly cũng có thể coi là file mã nguồn viết bằng ngôn ngữ CIL. Bạn không cần quan tâm đến ngôn ngữ này làm gì. C# compiler luôn giúp bạn dịch mã nguồn chương trình về CIL rồi.

Vai trò và đặc điểm của assembly trong .NET

Thứ nhất, assembly trong .NET cho phép tái sử dụng code độc lập với ngôn ngữ lập trình.

Để dễ hiểu, hãy hình dung thế này. Nếu bạn xây dựng một thư viện class qua một project trên C#. Bạn có thể sử dụng thư viện này trong project viết bằng bất kỳ ngôn ngữ .NET nào, như VB.NET, F#. Ở chiều ngược lại cũng đúng. Một thư viện class viết trên VB.NET (cũng là một assembly) có thể dễ dàng sử dụng trong một project C#.

Không chỉ đơn thuần là sử dụng class trong thư viện, bạn cũng có thể mở rộng class trong thư viện (dù là viết bằng ngôn ngữ .NET khác) thông qua cơ chế kế thừa.

Một interface được xây dựng trong thư viện viết bằng F# cũng có thể được thực thi bởi class viết trong C#.

Thứ hai, assembly tạo thêm một mức độ quản lý nữa đối với các kiểu dữ liệu, bên cạnh namespace. Hãy hình dung bạn xây dựng hai thư viện class riêng, giả sử đặt tên là MyCars.dll và YourCars.dll. Trong cả hai thư viện này có cùng namespace CarLibrary. Trong mỗi namespace này đều xây dựng class SportsCar. Đối với .NET, đây là hai kiểu dữ liệu khác nhau.

Thứ ba, cần phân biệt giữa private và shared assembly. Assembly trong .NET có thể được triển khai theo kiểu "private" hoặc "shared". Private assembly nằm trong cùng thư mục (hoặc thư mục con) của chương trình sử dụng nó và chỉ dành riêng cho chương trình đó. Shared assembly được triển khai để nhiều chương trình trên máy tính có thể dùng sử dụng. Shared assembly nằm trong một thư mục đặc biệt có tên gọi là Global Assembly Cache (GAC).

Thứ tư, mỗi assembly là một đơn vị độc lập có số phiên bản riêng. Một assembly được gán một số phiên bản bao gồm 4 phần theo mẫu <major>.<minor>.<build>.<revision>. Nếu bạn không tự mình cung cấp số phiên bản, assembly sẽ được tự động gán số phiên bản là 1.0.0.0.

Giá trị phiên bản cho phép nhiều phiên bản shared của cùng một assembly cùng tồn tại trên cùng một máy. CLR đảm bảo sẽ load phiên bản phù hợp của assembly cho client.

Mỗi assembly có một file cấu hình dạng xml riêng. File cấu hình này cho phép chỉ định vị trí đặt assembly, phiên bản sẽ được tải cho client, v.v..

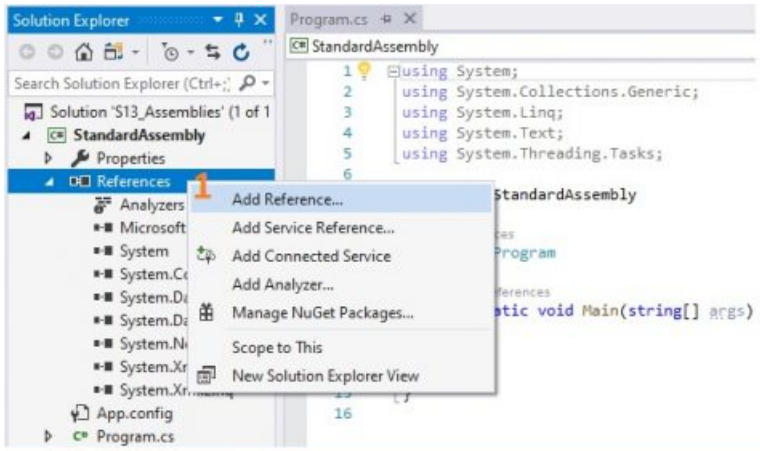
Sử dụng assembly có sẵn

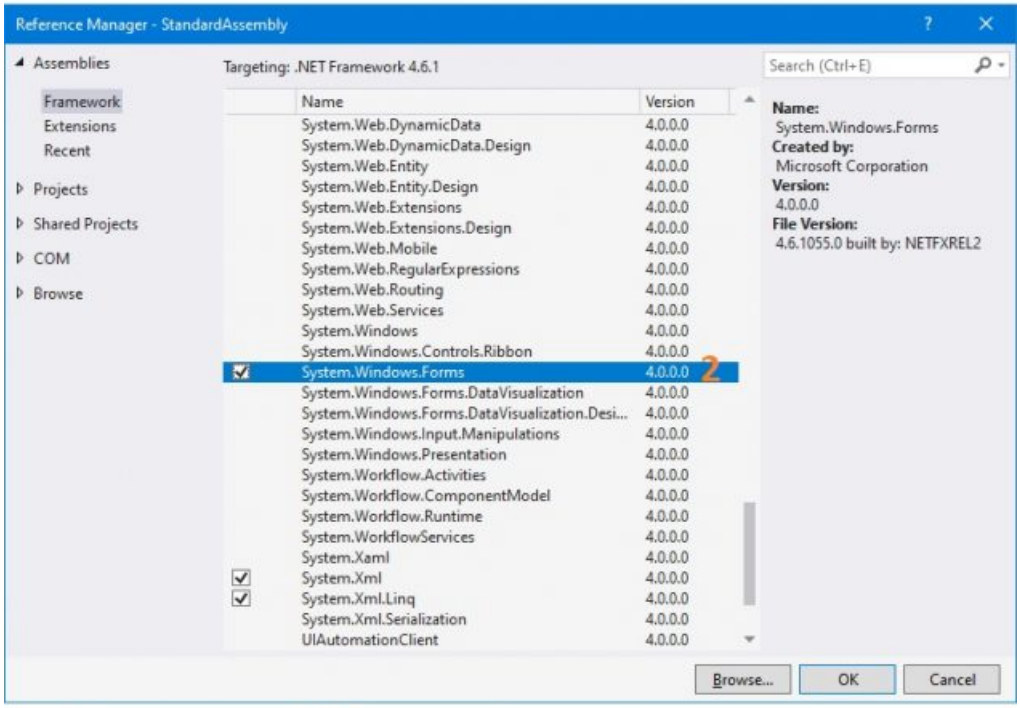
Việc sử dụng assembly trong C# project rất đơn giản. Bạn có thể dễ dàng tạo ra và sử dụng assembly của riêng mình cũng như sử dụng các assembly sẵn có (của .NET hoặc của bên thứ ba). Trong phần này chúng ta xem xét cách sử dụng assembly có sẵn. Phần tiếp theo sẽ hướng dẫn cách tự tạo và sử dụng assembly của riêng mình.

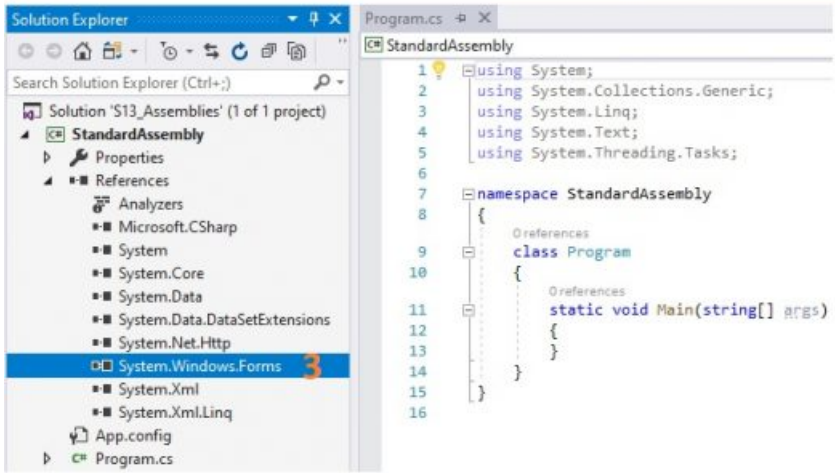
Sử dụng assembly của .NET

Để minh họa, chúng ta sẽ sử dụng class MessageBox của thư viện System.Windows.Forms. Class này cho phép chương trình hiện ra các hộp thoại thông báo.

Thực hiện theo các bước dưới đây để tham chiếu project tới thư viện System.Windows.Forms:







Viết vài dòng code vào Main():

```
using System;

namespace StandardAssembly
{
    class Program
    {
        static void Main(string[] args)
        {
```

```
            while (true)
            {
                Console.Write("Enter a message: ");
                var message = Console.ReadLine();

                System.Windows.Forms.MessageBox.Show(message,
                    "Message",
                    System.Windows.Forms.MessageBoxButtons.OK,
                    System.Windows.Forms.MessageBoxIcon.Information);
            }
        }
    }
}
```

Dịch và chạy thử chương trình để xem kết quả. Bạn nhập một dòng thông báo, chương trình sẽ hiện ra hộp thoại với thông báo đó.

Như bạn đã thấy, việc tham chiếu và sử dụng thư viện class chuẩn của .NET rất đơn giản.

Lưu ý rằng, nếu bạn mở thư mục bin/Debug (nơi chứa assembly chương trình), bạn sẽ không thấy file thư viện đâu. System.Windows.Forms.dll là một **shared assembly** của .NET Framework. Nó sẽ không được copy vào thư mục của chương trình.

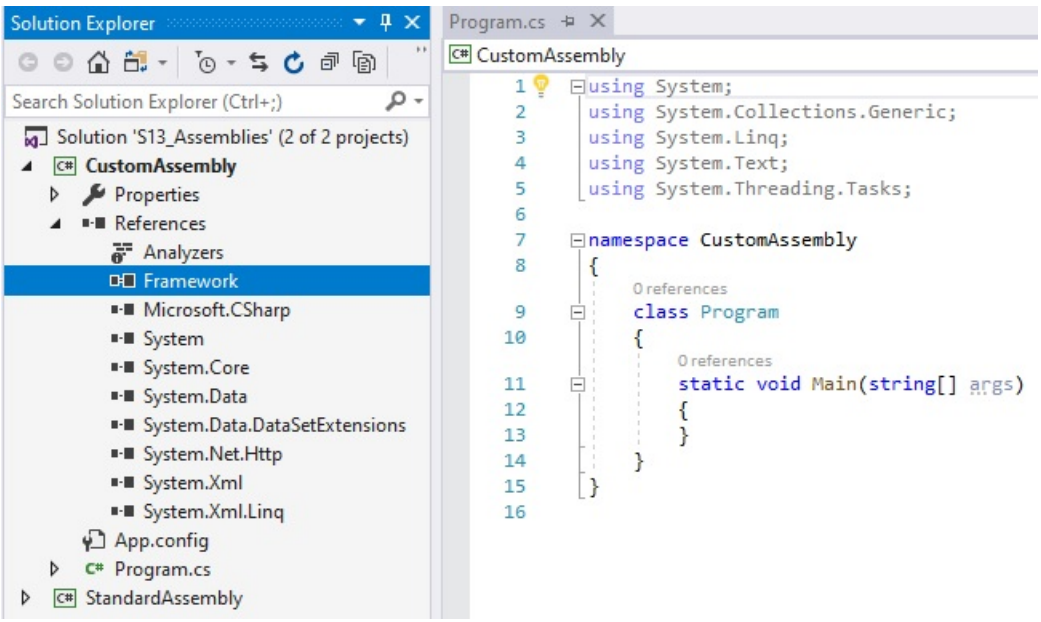
Sử dụng assembly của bên thứ ba

Đây là tình huống bạn tải từ đâu đó về một file thư viện dll và bạn muốn dùng nó trong project của mình. Để thực hiện ví dụ này, bạn có thể sử dụng một bộ thư viện mà Tự học ICT đã hướng dẫn làm trong loạt bài xây dựng thư viện hỗ trợ ứng dụng console.

Bạn tải file thư viện từ đường link này.

> Lưu ý: tránh sử dụng các thư viện không rõ nguồn gốc.

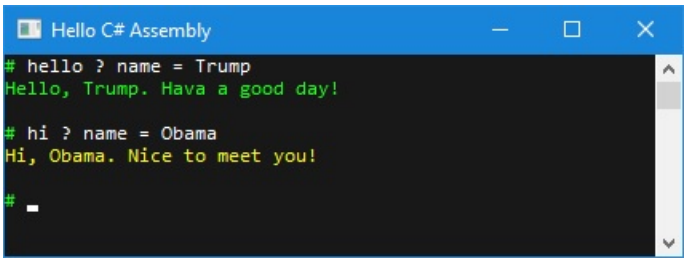
Thực hiện lại các thao tác như trên. Tuy nhiên ở bước 2 click vào nút **Browse** để tìm tới file dll vừa tải về.



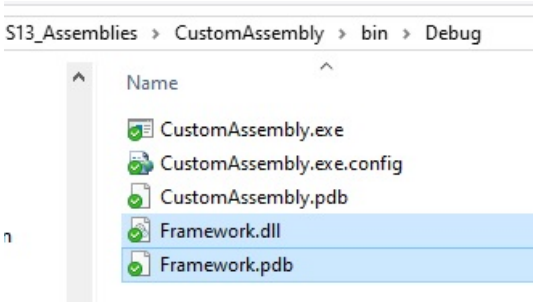
Viết vài dòng code sử dụng thư viện vừa rồi:

```
using System;

namespace CustomAssembly
{
    class Program
    {
        static void Main(string[] args)
        {
            var app = new Framework.Application
            {
                Title = "Hello C# Assembly",
                Prompt = "# ",
                Config = Config
            };

            app.Run();
        }

        static void Config()
        {
            Framework.Router.Instance.Register(
                route: "hello",
                action: (p) => Framework.ViewHelper.WriteLine($"Hello, {p["name"]}. Have
                );

            Framework.Router.Instance.Register(
                route: "hi",
                action: (p) => Framework.ViewHelper.WriteLine($"Hi, {p["name"]}. Nice to
                );
        }
    }
}
```

Chạy chương trình và nhập thử hai lệnh `hello ? name = Trump` và `hi ? name = Obama` để xem kết quả.



Nếu mở thư mục bin/Debug (nơi chứa exe assembly chương trình) bạn sẽ thấy file thư viện Framework.dll cùng nằm ở đây:



Framework.dll là một **private assembly**, chỉ được sử dụng bởi ứng dụng.

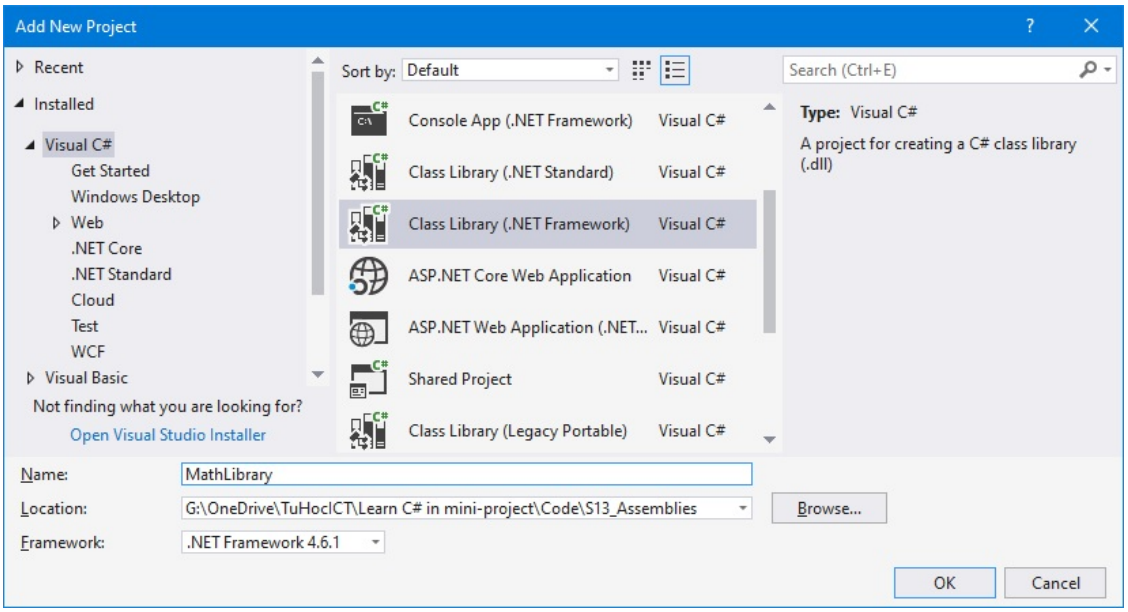
> Đây là bộ thư viện rất hữu ích nếu bạn phải viết các chương trình thực sự với giao diện console. Tự học ICT đã hướng dẫn chi tiết cách xây dựng thư viện này trong chuỗi bài học thư viện cho ứng dụng console.

Tự tạo và sử dụng class library trong solution

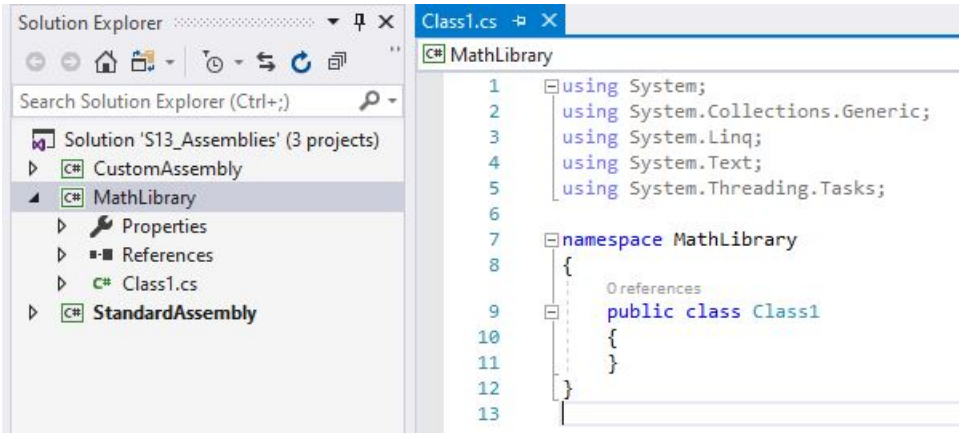
Để hiểu rõ hơn về assembly trong C# và .NET, hãy cũng tự xây dựng một class library, triển khai và sử dụng nó trong project. Class library này sẽ tổng hợp lại một số class mà bạn đã xây dựng trong các bài học trước. Do đó, bạn không cần tự gõ lại code mà chỉ cần copy code đã có sẵn.

Class Library project

Việc tạo ra các class library của riêng mình cũng rất đơn giản. Bạn hãy thêm một project mới vào solution. Tuy nhiên, ở cửa sổ Add New Project bạn chọn **Class Library** (.NET Framework) thay cho Console App (.NET Framework) mà bạn quen thuộc:



Sự khác biệt so với Console App project là không hề có Program hay Main. Bạn có một file code Class1.cs tạo sẵn:



Xóa bỏ file code này, thêm file code mới Complex.cs dành cho lớp Complex mà bạn đã từng xây dựng trong bài học về struct. Chúng ta chỉnh sửa Complex một chút để nó thành class.

```
using System;

namespace MathLibrary
```

```
{
    /// <summary>
    /// Class biểu diễn số phức
    /// </summary>
    public class Complex
    {
        public double Real; // trường thực
        public double Imaginary; // trường ảo

        public Complex()
        {

        }

        /// <summary>
        /// Hàm tạo
        /// </summary>
        /// <param name="r">phần thực</param>
        public Complex(double r)
        {
            Real = r;
            Imaginary = 0;
        }

        /// <summary>
        /// Hàm tạo
        /// </summary>
        /// <param name="r">phần thực</param>
        /// <param name="i">phần ảo</param>
        public Complex(double r, double i)
        {
            Real = r;
            Imaginary = i;
        }

        /// <summary>
        /// Chuyển chuỗi hợp lệ thành giá trị của Real và Imaginery
        /// </summary>
        /// <param name="value"></param>
        public void Parse(string value)
        {
            var temp = value.Trim();
            if (temp.EndsWith("i") || temp.EndsWith("I"))
            {
                temp = temp.TrimEnd('i', 'I');
                var tokens = temp.Split(new[] { '+', '-' }, 2);
                Real = double.Parse(tokens[0]);
                Imaginary = double.Parse(tokens[1]);
            }
            else
            {
                Real = double.Parse(temp);
            }
        }

        /// <summary>
        /// Chuyển chuỗi hợp lệ thành giá trị của Real và Imaginery
        /// </summary>
        /// <param name="value"></param>
        /// <returns></returns>
        public static Complex FromString(string value)
        {
            var temp = new Complex();
            temp.Parse(value);
            return temp;
        }

        /// <summary>
        /// Đặc tính, trả về module của số phức
        /// </summary>
        public double Modulus => Math.Sqrt(Real * Real + Imaginary * Imaginary);

        /// <summary>
        /// Ghi đè phép toán +
        /// </summary>
        /// <param name="a"></param>
        /// <param name="b"></param>
        /// <returns></returns>
        public static Complex operator +(Complex a, Complex b)
        {
            return new Complex(a.Real + b.Real, a.Imaginary + b.Imaginary);
```

```
        }

        /// <summary>
        /// Ghi đè phép toán -
        /// </summary>
        /// <param name="a"></param>
        /// <param name="b"></param>
        /// <returns></returns>
        public static Complex operator -(Complex a, Complex b)
        {
            return new Complex(a.Real - b.Real, a.Imaginary - b.Imaginary);
        }

        /// <summary>
        /// Ghi đè phương thức ToString() của object
        /// </summary>
        /// <returns></returns>
        public override string ToString()
        {
            if (Imaginary == 0)
            {
                return Real.ToString();
            }

            return $"{Real} {(Imaginary > 0 ? '+' : '-')} {Math.Abs(Imaginary)}i";
        }
    }
}
```

> Thực ra trong class library có thể chứa bất kỳ định nghĩa kiểu nào, không nhất thiết phải là class. Bạn có thể thoải mái định nghĩa các kiểu khác như enum, struct, interface, delegate. Class library không có gì khác biệt với Console App mà bạn đã quen thuộc.

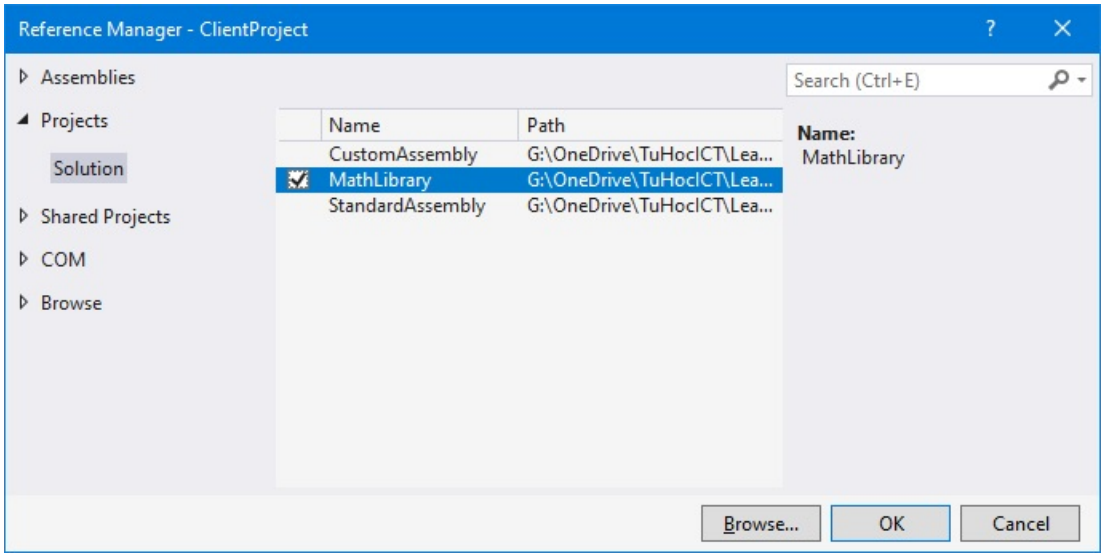
Để ý rằng bạn phải khai báo class Complex với từ khóa public

```
public class Complex
{
    ...
```

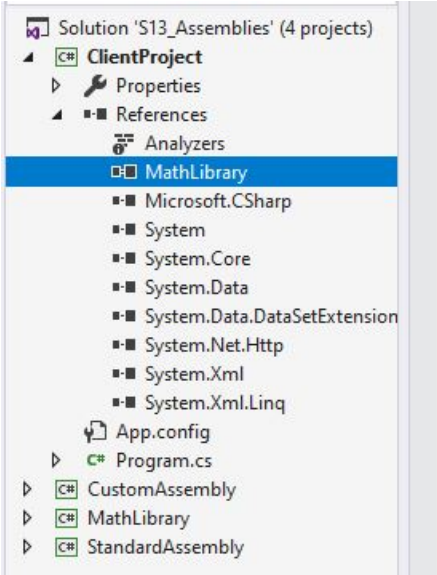
Nếu thiếu từ khóa này, lớp Complex chỉ có thể sử dụng trong nội bộ project MathLibrary. Khi đó, Complex sẽ không có ý nghĩa gì với project sử dụng thư viện này vì không ai nhìn thấy nó.

Sau khi viết code, hãy bấm tổ hợp Ctrl + Shift + B để build solution. Nếu không build solution, bạn sẽ chưa thể sử dụng thư viện này trong project khác.

Tham chiếu project tới thư viện vừa tạo



Nếu tham chiếu thành công, assembly MathLibrary sẽ xuất hiện trong danh sách References của ClientProject.



Giờ bạn có thể sử dụng Complex trong client code như trước đây:

```
1.  using System;
2.  using MathLibrary;
3.
4.  namespace ClientProject
5.  {
6.      using static Console;
7.      class Program
8.      {
9.          static void Main(string[] args)
10.         {
11.             Title = "Complex number";
12.
13.             // khai báo và khởi tạo biến a thuộc kiểu Complex
14.             var a = new Complex(1, 2);
15.             WriteLine($"a = {a}");
16.             // sử dụng đặc tính Modulus của Complex
17.             WriteLine($"|a| = {a.Modulus}");
18.
19.             // gọi phương thức Parse
20.             a.Parse("10-2i");
21.             WriteLine($"a = {a}");
22.
23.             // gọi phương thức tĩnh FromString
24.             var b = Complex.FromString("5 + 3i");
```

```
            WriteLine($"b = {b}");

            // thực hiện phép cộng trên số phức
            WriteLine($"a + b = { a + b}");

            ReadKey();
        }
    }
}
```

Lưu ý, ở khối using đầu file bạn có lệnh `using MathLibrary;` để sử dụng tên ngắn gọn của lớp Complex, thay vì phải sử dụng tên đầy đủ MathLibrary.Complex. MathLibrary là namespace nơi khai báo lớp Complex.

Lợi thế rất lớn của việc sử dụng class library trong solution nằm ở chỗ, nếu bạn thay đổi code của thư viện, sau khi build solution, client project sẽ tự cập nhật bản dịch mới của thư viện. Tức là, client project sẽ luôn sử dụng bản dịch cập nhật của thư viện mà không cần tham chiếu lại.

Nếu bạn bỏ từ khóa public trước khai báo class Complex (chuyển nó thành internal), client code sẽ không nhìn thấy class này nữa. Nghĩa là bạn không thể sử dụng Complex khi nó được khai báo là internal.

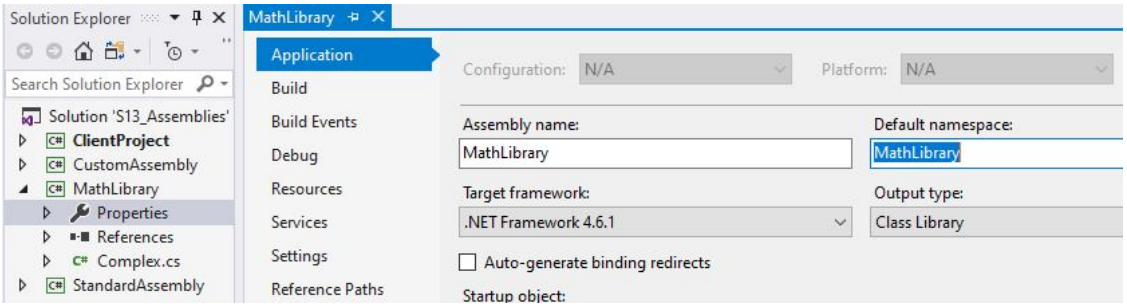
Một số lưu ý khi tạo thư viện class

Class Library project không có gì khác biệt với Console App project. Tuy nhiên, vì bạn xây dựng thư viện thường là để người khác sử dụng, có một số điều nên lưu ý.

Default namespace

Khi tạo project mới, Visual studio sẽ sử dụng tên project làm default namespace. Mỗi khi bạn tạo file code mới, default namespace sẽ được sử dụng ngay cho file đó.

Bạn có thể thay đổi giá trị này trong mục Properties của project như sau:



Xml documentation

Bạn đã biết documentation comment và vai trò của nó đối với Visual studio Intellisense. Nếu bạn tạo class library và sử dụng thẳng trong project, documentation comment sẽ được Intellisense tiếp tục sử dụng trong client project. Nghĩa là khi trỏ vào tên class/method/interface v.v, comment này sẽ hiển thị để bạn hiểu đó là cái gì.

Nếu bạn dịch và cung cấp thư viện dll này cho người khác sử dụng, documentation comment sẽ không xuất hiện trong client project. Để client project bất kỳ có thể sử dụng

documentation comment bạn đã viết trong thư viện, bạn mở node **Properties** như trên nhưng chuyển sang mục **Build**, click chọn mục "**XML documentation file**".

Khi build project, Visual studio sẽ tự động đưa các comment này vào một file xml. Nếu bạn cung cấp file xml này cùng với file thư viện dll, client project sẽ tiếp tục hiển thị các documentation comment.

> Khi sử dụng thư viện Framework.dll ở phần trên bạn có thể đã để ý thấy điều này.

Nested namespace

Khi có số lượng lớn các kiểu (class, struct, enum, interface, delegate) định nghĩa trong class library, bạn nên xem xét phân chia chúng theo chức năng nhiệm vụ. Điều này giúp đơn giản hóa việc sử dụng chúng trong client project.

Bạn đã biết quy ước phân chia code thành file trong project. Tuy nhiên, điều này chỉ ảnh hưởng đến việc viết và quản lý code của chính project đó. Đối với người sử dụng thư viện (đã biên dịch thành dll), họ chỉ nhìn thấy cách quản lý class theo namespace.

Do vậy, bạn nên phân chia class (và các định nghĩa kiểu khác) vào các **nested namespace** nếu cần thiết.

Lấy ví dụ thư viện MathLibrary bên trên. Bạn đã định nghĩa kiểu Complex. Nếu bạn tiếp tục định nghĩa các kiểu dữ liệu phức tạp khác như Vector, Matrix. Bạn cũng đồng thời định nghĩa class Math để thực hiện các hàm tính toán. Như vậy, các class này có thể chia làm hai nhóm: nhóm kiểu dữ liệu và nhóm thực hiện chức năng.

Để người sử dụng thư viện dễ dàng hiểu và tìm đến class theo nhu cầu, bạn có thể điều chỉnh namespace của các class Complex, Vector, Matrix thành:

```
namespace MathLibrary.Types
{
    public class Complex
    {
```

```
namespace MathLibrary.Types
{
    public class Vector
    {
```

Khi này, trong client project bạn có thể sử dụng các class này theo tên đầy đủ là `MathLibrary.Types.Complex`, `MathLibrary.Types.Vector`, v.v.. Hoặc sử dụng tên ngắn gọn nếu bổ sung `using MathLibrary.Types;` ở khối using.

Sử dụng thư viện từ NuGet, NuGet Packages Manager

Giới thiệu về NuGet

NuGet là hệ thống quản lý gói phần mềm mã mở miễn phí thiết kế cho các nền tảng phát triển ứng dụng của Microsoft từ 2010. Hiện nay NuGet đã trở thành một hệ sinh thái lớn chứa nhiều loại công cụ và dịch vụ. Trên NuGet hiện có khoảng hơn 130 nghìn gói thư viện với 1,4 triệu phiên bản và 1,3 tỉ lượt download.

Lập trình viên .NET có thể dễ dàng tìm, sử dụng và cung cấp các gói thư viện thông qua NuGet.

Trước đây NuGet thường được cài đặt như một ứng dụng mở rộng của Visual Studio. NuGet được cài đặt mặc định trên Visual Studio từ phiên bản 2012.

Chúng ta có thể sử dụng theo một số cách khác nhau:

1. sử dụng ứng dụng giao diện đồ họa NuGet Package Manager,
2. sử dụng giao diện dòng lệnh Package Manager Console,
3. cài đặt tự động với các file mã kịch bản.

Cách đơn giản nhất để tìm và cài đặt các gói thư viện từ NuGet là sử dụng tiện ích mở rộng NuGet Package Manager.

Cài đặt thư viện NewtonSoft.Json với NuGet Manager

Để minh họa việc tải và cài đặt gói thư viện từ NuGet, chúng ta sẽ cùng cài NewtonSoft.Json.

NewtonSoft.Json là bộ thư viện này cho phép chuyển đổi một object của C# thành một chuỗi ký tự định dạng theo quy ước của JSON (JavaScript Object Notation) cũng như chuyển đổi ngược chuỗi JSON về object của C#. Đây là một trong những bộ thư viện có lượt download lớn nhất trên NuGet.

> Quá trình chuyển đổi này có tên gọi là *serialization* (từ object về JSON) và *deserialization* (từ JSON về object). Bạn sẽ học về serialization, bao gồm xml, json và binary serialization trong một bài khác.

Bạn có thể sử dụng bộ thư viện này để tự lưu thông tin cấu hình cho ứng dụng, hoặc lưu trữ dữ liệu đơn giản thay cho sử dụng một cơ sở dữ liệu.

Bước 1. Mở giao diện quản lý các gói thư viện NuGet

Click phải vào References, chọn Manage NuGet Packages (xem hình dưới đây).

Mở giao diện Manage NuGet Packages

Bước 2. Chọn cài gói thư viện

Trong ô tìm kiếm ở tab Browse gõ *newtonsoft*, chọn gói NewtonSoft.Json và ấn Install.

Giao diện Manage NuGet Packages

Sau lệnh này, Visual Studio sẽ tải gói thư viện này về và cài đặt lên project tương ứng (trong trường hợp này là BookMan.ConsoleApp).

Kiểm tra kết quả

Sau khi cài đặt thành công bộ thư viện này, trong danh sách References sẽ xuất hiện thêm một mục "NewtonSoft.Json". Trong cấu trúc dự án sẽ xuất hiện thêm file "packages.config" chứa thông tin về các gói thư viện được cài đặt đặt thêm.

Sau khi dịch chương trình (Ctrl + Shift + B) thành công, trong thư mục BinDebug của dự án sẽ xuất hiện file thư việc NewtonSoft.Json.dll.

Khi triển khai ứng dụng cho người dùng cuối, file thư viện này cũng phải đi cùng file chương trình.

File thư viện Newtonsoft.json.dll sau khi cài đặt

Cài đặt gói NuGet sử dụng website kết hợp Package Manager Console

Cách thứ hai là sử dụng dịch vụ tìm kiếm trên website https://www.nuget.org/packages để tìm gói thư viện phù hợp. Sau đó copy dòng lệnh paste vào Package Manager Console.

Giao diện tìm kiếm thư viện Newtonsoft.Json trên website

Nếu không nhìn thấy tab Package Manager Console, chọn View => Other Windows => Package Manager Console, hoặc Tools => NuGet Package Manager => Package Manager Console.

Giao diện dòng lệnh của Package Manager Console

Khi sử dụng Package Manager Console lưu ý chọn tham số "Default project" là project mình cần cài đặt gói thư viện.

Một số lưu ý khi sử dụng các gói thư viện từ NuGet

Khi sử dụng các gói thư viện trên NuGet cần lưu ý xem xét kỹ sự phụ thuộc (dependency) của gói thư viện cần dùng với các gói thư viện khác.

Lý do là nhiều thư viện trên NuGet sử dụng lẫn nhau, cũng như được xây dựng cho các phiên bản .NET khác nhau.

Khi cài đặt một thư viện mà nó phụ thuộc vào các thư viện khác, các thư viện kia cũng phải được cài đặt theo và phải cài đặt phiên bản mà thư viện chính có thể sử dụng được.

Nếu các gói thư viện có phiên bản mới, NuGet cũng cho phép cập nhật phiên bản đang cài đặt trong dự án lên phiên bản mới. Tuy nhiên, cũng giống như khi cài đặt, phải lưu ý sự phụ thuộc giữa các thư viện trước khi quyết định nâng cấp.

Kết luận

Bài học rất dài này đã cung cấp cho bạn đầy đủ kiến thức cần thiết để hiểu và làm việc với assembly trong .NET. Các nội dung của bài không khó tuy nhiên bạn cần để ý khi sử dụng.

Một lời khuyên là đừng ngần ngại tách project lớn thành các project nhỏ với class library assembly trong đó. Nó sẽ giúp bạn quản lý code tốt hơn. Đặc biệt, nếu bạn có các class cần tái sử dụng qua nhiều project, hãy đặt nó vào một class library.

> + Nếu bạn thấy site hữu ích, trước khi rời đi hãy **giúp đỡ** site bằng một hành động nhỏ để site có thể phát triển và phục vụ bạn tốt hơn.
> + Nếu bạn thấy bài viết hữu ích, hãy giúp **chia sẻ** tới mọi người.
> + Nếu có thắc mắc hoặc cần trao đổi thêm, mời bạn viết trong phần **thảo luận** cuối trang.
> Cảm ơn bạn!