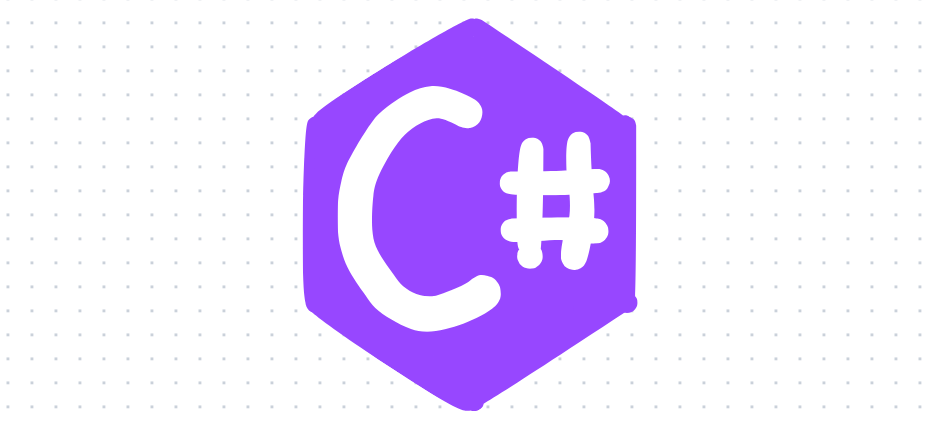
Nhật ký học C#

(từ đầu vì hiện tại chưa biết gì về cái này)



# I, Tổng quan về ngôn ngữ và Framework

## 1, Định nghĩa – giới thiệu

### C#

* C#: là **ngôn ngữ lập trình**.
* Do Microsoft phát triển trong khuôn khổ .Net Framework (lúc đầu là thế giờ đã được mở rộng).
* Một vài ứng dụng phổ biến:
  + Phát triển ứng dụng desktop: Windows Forms, WPF
  + Phát triển ứng dụng web: ASP.NET Core, Blazor
  + Phát triển game: Unity
  + Lập trình di động đa nền tảng: MAUI, Xamarin
  + Xây dựng hệ thống backend, API

*(cũng nhiều)*

* Tích hợp kha khá tính năng hiện đại: LINQ, async/await, kiểu nullable, pattern matching, và các extension methods, … chắc chắn sẽ nói nhưng là ở những phần sau.

### .NET Platform

* DotNet là một **nền tảng phát triển phần mềm (software development platform)**.
* Một “nền tảng phát triển phần mềm” là một định nghĩa rộng: bao gồm những thành phần như công cụ phát triển (trình biên dịch, thông dịch, IDE, công cụ kiểm thử, …), thư viện và API, Runtime Environment (môi trường thực thi cho code), và các plugins (khả năng mở rộng và kết hợp để đáp ứng nhu cầu thực tế).

*Định nghĩa này là môi trường rộng nhất mà t từng đọc.*

### Các phiên bản của .NET

#### .NET Framework (2002 – 2019)

* DotNet Framework cũng là một **nền tảng phát triển phần mềm** (cũ).
* Phiên bản cuối cùng của .NET Framework là 4.8 vào 2019.
* Chỉ hỗ trợ Windows.
* Mang tính năng truyền thống *(Windows Forms, WPF, WCF, ASP.NET truyền thống).*

#### .NET Core (2016 – 2020)

* DotNet Core cũng là một **nền tảng phát triển phần mềm.** Nhưng cũng cũ.
* Phiên bản cuối cùng của .NET Core là 3.1 vào 2019.
* Bắt đầu hỗ trợ đa nền tảng.
* **Bổ sung thêm tính năng *(ASP.NET Core, EF Core, SignalR, gRPC)*.**

#### .NET (2020 đến nay)

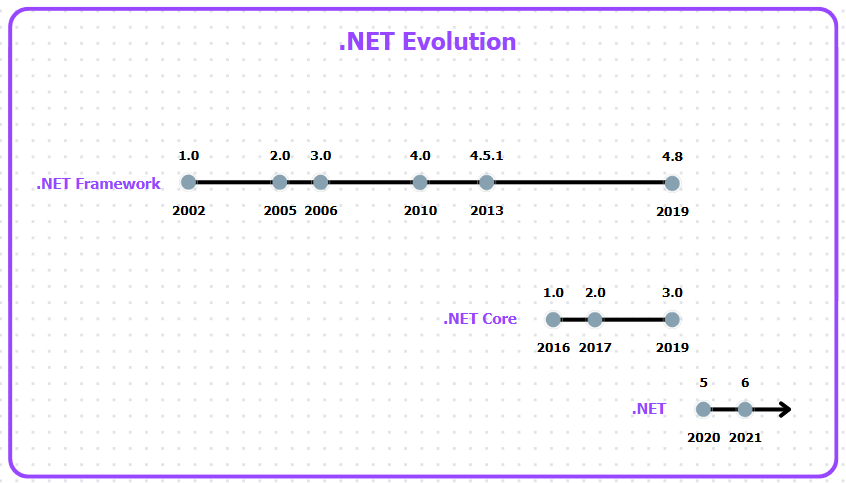
* DotNet cũng là **nền tảng phát triển phần mềm** (mới).
* Được coi như sự kết hợp và thay thế cho .NET Core và .NET Framework.
* Hỗ trợ thêm nhiều tính năng khác nhau *(nói sau đi nói giờ cũng có để ý được đâu?)*.



*Tại sao phải giới thiệu nhiều phiên bản như vậy? Cứ học theo cái mới nhất đi có phải đỡ lằng nhằng hơn không? Tại sao phải biết những thứ này trong khi chưa viết code?*

* Thì cũng định bỏ qua rồi đấy nhưng khi đọc tài liệu và gặp những thuật ngữ như trên, không thể nào tránh khỏi confused. Nên phải viết tử tế các khái niệm ngay từ đầu.

#### Bonus: biểu đồ thời gian các phiên bản và giai đoạn của từng nền tảng

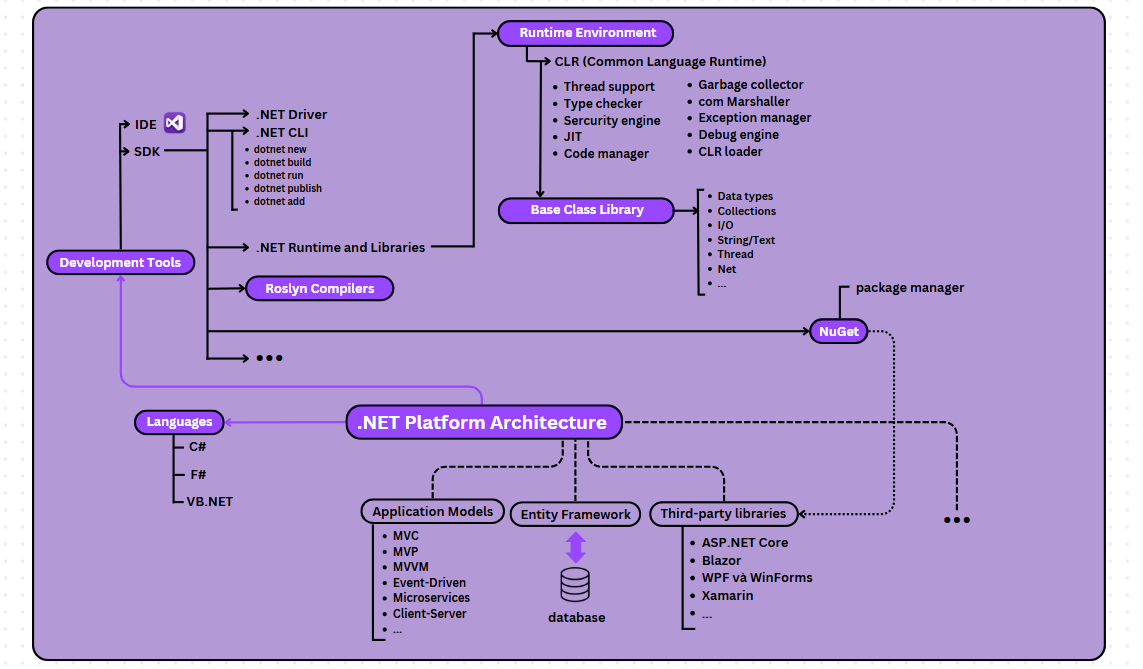


## 2, Cấu trúc cơ bản của một nền tảng .NET nói chung

Trước khi code thì tôi vẫn khá thắc mắc làm sao một chương trình C# chạy được. Thắc mắc có phần phức tạp và hơi ngu vào lúc này nhưng cũng nên biết qua thành phần của .NET để hiểu xem nó chạy như thế nào.



Một “nền tảng phát triển phần mềm”, tất nhiên cực kỳ phức tạp. Thử mổ xẻ một chút bên ngoài của nó:



Tôi đã lại vẽ lại khoảng 11 lần cái hình này.

Phức tạp thế nhờ.



Đầu tiên hiểu những nét màu tím là thành phần chính của chương trình cơ bản (biên dịch, chạy code). Còn nét đứt đoạn là thành phần thêm, phục vụ khi làm với những chương trình lớn và phức tạp hơn.

Bây giờ tìm hiểu qua về chúng nó xem sao.

Cấu trúc cơ bản chia ra thành 2 phần: Development Tool và Language (?). Không phải chúng không liên kết với nhau, mà là vì mục đích của 2 phần đó khác nhau: Development Tool có cấu trúc được xây dựng với mục đích chạy chương trình, còn Language liên quan đến cách xây dựng như thế nào để hỗ trợ được các ngôn ngữ như hình. Vậy chỉ nên tập trung nhiều vào Development Tool trong phần tìm hiểu tổng quát này.

### 1, Languages

* Languages được viết ở cấp độ lập trình viên.
* Ngôn ngữ trong .NET platform rất đa dạng, nhưng phổ biến nhất vẫn là **C#**. Nghĩa là .NET hỗ trợ nhiều ngôn ngữ ấy. Nhưng đang học C# thôi. Có thể để ý thêm phần **Razor** (mã HTML kết hợp với C# trong ASP.NET MVC hoặc Blazor).
* Chả cần phải biết nó đọc code C# như thế nào đâu :)) hỏi thế khác nào hỏi, tôi đang đọc tiếng Việt như nào?

### 2, Development Tools

* Nghĩa đen là **công cụ phát triển**.
* Là các phần mềm hoặc tiện ích giúp lập trình viên viết, gỡ lỗi, kiểm tra, triển khai mã nguồn trong quá trình phát triển phần mềm.
* Nó chỉ là một **khái niệm chung** cho các công cụ hỗ trợ lập trình thôi.

Trong bối cảnh tôi đang xét thì có thể nó gồm 2 phần chính là IDE và SDK. Các loại khác tính sau.

#### 2.1, IDE

* IDE phổ biến dành cho C# là Visual Studio. (Windows/macOS)
* Download: <https://visualstudio.microsoft.com/downloads/>

*IDE này khá nặng và có nhiều chức năng. Nhưng chưa để ý vội mấy chức năng sau, hiện tại đang học C# core.*

* Dạng project sẽ học trong phần cơ bản này: C# console app - project OOP (có thể được tổ chức dưới dạng MVC).

*Vì OOP đã học từ trước nên phần này sẽ lướt qua và tập trung vào khác biệt trong cú pháp/cấu trúc xây dựng ngôn ngữ và chương trình so với Java.*

Tự tìm hiểu phần cài đặt IDE.

#### 2.2, SDK

<https://learn.microsoft.com/vi-vn/dotnet/core/sdk>

#### Định nghĩa

* Software Development Kit.
* Là một **bộ công cụ phát triển phần mềm** cung cấp các công cụ, thư viện, tài liệu, và các thành phần cần thiết để các lập trình viên có thể phát triển ứng dụng hoặc phần mềm trên một nền tảng cụ thể.
* *Nghe cái này cũng giống khái niệm của Development Tool. Vậy chúng khác nhau cái gì?*

**

*SDK là bộ công cụ phát triển dành cho một nền tảng cụ thể, đóng vai trò như bộ máy chính để thực thi code, tương tự như động cơ của một chiếc ô tô. Còn Development Tool là cả chiếc ô tô, bao gồm cả những thiết bị hỗ trợ lập trình viên như IDE, Debugger, và cả SDK. Nói cách khác, Development Tool là một tập hợp bao gồm cả SDK, và một vài công cụ khác.*

* Theo docs và theo thực tế tôi thấy thì SDK gồm 3 thành phần nổi bật:.NET Driver, .NET CLI và .NET Runtime and Libraries. Ngoài ra có thể kể đến những thành phần khác, như NuGet (quản lý gói),…

##### 2.2.1, NET Driver và .NET CLI

* Trong SDK, **.NET Driver** là một **công cụ dòng lệnh** giúp lập trình viên tương tác với môi trường .NET.
* Còn **.NET CLI** (Command Line Interface) là **giao diện dòng lệnh của .NET**.
* **CLI** là giao diện để người dùng nhập lệnh. **Driver** là thành phần thực sự thực hiện các tác vụ dựa trên lệnh đó.
* Cụ thể hơn thì:
  + **CLI** tiếp nhận lệnh “dotnet + <yêu cầu>”.
  + **CLI** chuyển lệnh tới **Driver**.
  + **Driver** kiểm tra mã nguồn, biên dịch ứng dụng, tải runtime, và thực thi mã.
  + Kết quả (chạy thành công hoặc lỗi) được hiển thị qua **CLI**.

Tùy vào từng lệnh mà Driver sẽ sử dụng những tài nguyên và thành phần của SDK để chạy chương trình. Đây cũng là nơi khởi nguồn và tổng quát cho việc chạy chương trình.

(…)

*(Thôi chả nói nhiều nữa hết tg rồi)*

##### 2.2.2, Roslyn Compilers

[*https://github.com/dotnet/roslyn*](https://github.com/dotnet/roslyn)

* Là **bộ công cụ biên dịch (compiler platform).**
* Được phát triển bởi Microsoft, tích hợp sẵn trong .NET và hỗ trợ các ngôn ngữ lập trình chính của .NET như **C#** và **Visual Basic (.NET)**.
* Đây là thành phần trực tiếp tiếp xúc với đống code C# mình viết ra, chuyển đổi mã C# đó thành mã trung gian IL, rồi đẩy nó sang cho phần tiếp theo thực thi.

*Tại sao phải biên dịch thành IL à? Nói sau.*

* Roslyn bao gồm 2 thành phần chính là **Compiler** và **Compiler Platform APIs**.

###### Trình biên dịch (Compiler):

Bao gồm các trình biên dịch cụ thể cho:

* + - C# (**csc.exe**): Biên dịch mã nguồn C# thành mã trung gian IL (Intermediate Language).
    - VB.NET (vbc.exe): Biên dịch mã nguồn VB.NET thành mã IL.

Thành phần này xử lý các giai đoạn:

* + - Lexical Analysis: Phân tích mã nguồn thành các token.
    - Syntax Analysis: Xây dựng cây cú pháp (Syntax Tree).
    - Semantic Analysis: Xác minh các thông tin ngữ nghĩa như kiểu dữ liệu, phạm vi biến.
    - Code Generation: Chuyển đổi mã nguồn thành mã IL để chạy trên Common Language Runtime (CLR). (nói sau)

###### API Công cụ lập trình (Compiler Platform APIs):

Đây là thành phần mở rộng của Roslyn, cho phép sử dụng các tính năng của trình biên dịch không chỉ để biên dịch mà còn để phân tích, tạo, và chỉnh sửa mã. (gợi ý code và phát hiện lỗi chính tả code đấy).

*Sẽ không bàn về cấu trúc chi tiết của nó vì quá không cần thiết.*

Các thành phần chính và tính năng nổi bật của nó:

* + Syntax Tree APIs: Cho phép truy xuất và chỉnh sửa cây cú pháp của mã nguồn.
  + Semantic Model APIs: Cung cấp thông tin về kiểu dữ liệu, tham chiếu và ngữ nghĩa trong mã.
  + Code Generation APIs: Hỗ trợ tạo hoặc chỉnh sửa mã nguồn mới.
* Ứng dụng:
  + Tích hợp vào IDE để cung cấp IntelliSense – sửa lỗi cú pháp, và gợi ý mã.
  + Phát triển các công cụ refactor mã hoặc kiểm tra chất lượng mã tự động.
  + Tạo các công cụ tùy chỉnh phân tích mã (Code Analyzers).

Chỉ chú ý ở phần này là: Roslyn Compiler tương tác với mã nguồn C# và biên dịch nó ra mã trung gian IL chứ chưa liên quan gì tới phần chạy chương trình đó.

##### 2.2.3, .NET Runtime Environment

###### Định nghĩa

* Về tổng quát, Runtime Environment – môi trường chạy – là một **môi trường thực thi đặc biệt dành riêng** để chương trình code chạy trong đó.
* Nếu như thế áp vào cái đang học thì Runtime Environment trong trường hợp này là môi trường dùng để chạy các chương trình code bằng C#. Vậy cụ thể hơn thì môi trường này bao gồm những gì?



*Trước hết phải hiểu: Runtime Environment là một khái niệm rộng. Môi trường chạy ở đây có thể bao gồm luôn cả hệ điều hành, cả những gì cung cấp code cho chương trình (IDE),... trong một vài hoàn cảnh đối với những ngôn ngữ khác như Java, Runtime Environment chỉ là “nơi để code chạy”. Có nghĩa là nó không mang tính chất bao hàm một vài những thành phần bên trong (trong ảnh). Lúc đó có thể hiểu Runtime Environment là Runtime Execution Environment – môi trường thực thi mã.*

*Định nghĩa của mỗi nguồn có thể khác nhau, nhưng sau khi đọc n cái nguồn thì tôi thấy như này là hợp lý nhất.*



###### Thành phần của Runtime Environment

Không cần tìm hiểu quá sâu bây giờ đâu. Runtime Environment trong bối cảnh này, như trên nói, là môi trường thực thi mã, và thông thường nó là **CLR (Common Language Runtime)**.

Thế cái đó là gì?

<https://learn.microsoft.com/en-us/dotnet/standard/clr>

<https://www.geeksforgeeks.org/architecture-of-common-language-runtime-clr/>

Định nghĩa và chức năng chung của CLR

* Common Language Runtime.
* Theo docs thì nó mang chức năng cung cấp một nền tảng chạy thống nhất cho tất cả các ngôn ngữ lập trình được hỗ trợ trong .NET, như C#, VB.NET, F#, v.v. *(còn nền tảng đó ra sao tôi chịu)*
* Nó chịu trách nhiệm **chạy mã** và cung cấp các dịch vụ hỗ trợ để giúp lập trình viên phát triển ứng dụng dễ dàng hơn.
* Chạy mã ở đây không phải là mã C#. **CLR chạy mã IL (Immediate Language),** hay gọi là **Microsoft Intermediate Language (MSIL)** hoặc **Common Intermediate Language (CIL).**

*Đây là cái mã mà Roslyn vừa biên dịch ở pha trước.*

Trong CLR có rất nhiều thành phần, và cũng chỉ nên biết qua về chúng, nên tôi đã nhờ GPT giúp một phần:



**a, Thread Support (Hỗ trợ luồng)**

Đảm bảo thực hiện và quản lý các luồng trong ứng dụng, hỗ trợ lập trình đa luồng để tăng hiệu năng và quản lý tài nguyên.

**b, Type Checker (Kiểm tra kiểu dữ liệu)**

Đảm bảo rằng tất cả các thao tác và dữ liệu trong chương trình tuân theo các quy tắc kiểu dữ liệu, giúp phát hiện lỗi tại thời gian chạy.

**c, Security Engine (Cơ chế bảo mật)**

Quản lý quyền truy cập và bảo mật ứng dụng.

**d, JIT (Just-In-Time Compiler)**

Biên dịch mã Intermediate Language (IL) sang mã máy khi chương trình được chạy, đảm bảo hiệu suất tối ưu. Còn nó lấy IL ở đâu ra lát nói.

**e, Code Manager (Quản lý mã)**

Quản lý việc thực thi mã trong ứng dụng, bao gồm theo dõi mã đã được nạp và biên dịch.

**f, Garbage Collector (Bộ thu gom rác)**

Tự động quản lý bộ nhớ, giải phóng các tài nguyên không còn được sử dụng để tránh rò rỉ bộ nhớ.

**g, COM Marshaller**

Hỗ trợ tương tác giữa mã .NET và các thành phần COM (Component Object Model), đảm bảo chúng giao tiếp hiệu quả.

Hiểu đơn giản hơn, khi những phần mềm khác có liên kết với chương trình C# (như Excel chẳng hạn), mà nó sử dụng định dạng COM để cung cấp API *(định dạng COM - chuẩn của Microsoft để tạo ra các thành phần phần mềm (components) có thể tái sử dụng và giao tiếp với nhau, và chương trình C# không theo chuẩn này, chúng theo định dạng .NET)* thì com Marshaller đóng gói dữ liệu từ định dạng .NET sang định dạng COM và ngược lại. Mục đích để 2 chương trình có cấu tạo phần mềm khác nhau (.NET và COM) có thể giao tiếp.



**h, Exception Manager (Quản lý ngoại lệ)**

Quản lý và xử lý ngoại lệ trong ứng dụng, cung cấp cơ chế phát hiện và khôi phục lỗi.

**i, Debug Engine (Công cụ gỡ lỗi)**

Cung cấp các công cụ để gỡ lỗi mã nguồn, bao gồm theo dõi thực thi, điểm dừng (breakpoints), và kiểm tra trạng thái chương trình.

**j, CLR Loader**

Nạp và khởi động Common Language Runtime (CLR), quản lý việc tải các assembly cần thiết để chạy ứng dụng.

k, Base Class Library

Dịch theo nghĩa đen là những lớp thư viện cơ sở.

* Là một tập hợp các thư viện cốt lõi được cung cấp trong .NET Framework, .NET Core và các phiên bản .NET khác.
* BCL chứa các lớp và chức năng nền tảng, hỗ trợ các tác vụ phổ biến trong lập trình ứng dụng.
* Các thành phần trong BCL (chủ yếu là những thư viện base):
  + **Kiểu dữ liệu**: System.Object, System.String, System.Int32,...
  + **Collections**: List, Dictionary,... trong System.Collections.
  + **I/O**: Đọc/ghi tệp, luồng (System.IO).
  + **Chuỗi và văn bản**: StringBuilder, Encoding (System.Text).
  + **Xử lý lỗi**: System.Exception.
  + **Luồng và bất đồng bộ**: System.Threading, System.Threading.Tasks.
  + **Mạng**: System.Net, System.Net.Http.

*(Còn nhiều thứ để nói hơn nhưng thôi)*

Chắc quan tâm duy nhất là cái BCL vì sắp học tới nó.

##### 2.2.4, NuGet

###### 1, Định nghĩa

* Là trình quản lý gói (package manager).
* Nó giúp thêm, cập nhật và quản lý các thư viện bên ngoài (packages) trong dự án.
* NuGet được tích hợp trong **Visual Studio**, nhưng cũng có thể sử dụng độc lập thông qua **giao diện dòng lệnh Command Line Interface (CLI)**.

###### 2, Chức năng

Vì khá rõ ràng rồi không trừu tượng nữa nên nói nhanh:

**Quản lý gói:**

* Tìm, cài đặt, cập nhật, và gỡ bỏ các gói thư viện.
* Xử lý các phụ thuộc giữa các gói tự động.

**Khôi phục gói:**

* Đảm bảo tất cả các gói cần thiết được tải về khi build dự án.

**Tạo và phân phối gói:**

* Cho phép đóng gói thư viện và đẩy lên kho NuGet công khai hoặc riêng tư.

**Hỗ trợ kho lưu trữ:**

* Kết nối kho công khai (NuGet.org) hoặc kho riêng của tổ chức.

**Tích hợp công cụ:**

* Sử dụng qua CLI hoặc giao diện Visual Studio để quản lý gói.

*(Cái này hỗ trợ khá nhiều nhưng từ từ tìm hiểu tiếp)*

## 3, Những thành phần bổ sung

Như đã nói thì những thành phần bên trên mới chỉ là thành phần cơ bản để chạy chương trình Hello World. Để được một dự án trung bình thì cần thêm thư viện, v.v… Đó là những thành phần còn lại (hoặc là tôi mới chỉ biết đến đó) cúa một chương trình C#.

Giới thiệu 3 cái trong hình vẽ của t: Application Models (tương đương với cấu trúc mã nguồn – MVC các thứ), Entity Framework (tương đương với Hibernate), Third-party libraries (thư viện từ nguồn thứ 3).

##### 3.1, Application Models

* **Định nghĩa:** Các mô hình **tổ chức mã nguồn** để quản lý và phát triển ứng dụng.
* **Chức năng:** Tách biệt logic nghiệp vụ, giao diện, và xử lý yêu cầu. Hoặc là yêu cầu cụ thể của mô hình.
* Các mô hình phổ biến: MVC (và mấy cái tương tự như hình), Web API, Razor Pages, Blazor, …

##### 3.2, Entity Framework

* **Định nghĩa:** ORM framework trong .NET, giúp ánh xạ đối tượng trong mã nguồn với cơ sở dữ liệu.
* **Chức năng:** (cũng tương tự như Hibernate)
  + Tự động hóa các thao tác cơ sở dữ liệu (CRUD).
  + Quản lý phụ thuộc dữ liệu và quan hệ giữa các bảng.
  + Hỗ trợ truy vấn dữ liệu bằng LINQ (bên Hibernate là HQL).

##### 3.3, Third-party Libraries

* **Định nghĩa:** Các thư viện từ bên thứ ba, không phải do Microsoft cung cấp, thường được tải từ NuGet.
* **Chức năng:** Bổ sung hoặc thay thế các tính năng của framework gốc, giúp tăng năng suất và mở rộng khả năng của ứng dụng.

##### …

### Cách hoạt động và luồng chương trình cơ bản

Với cấu trúc như trên thì một đoạn mã C# đi vào máy sẽ được xử lý như thế nào?

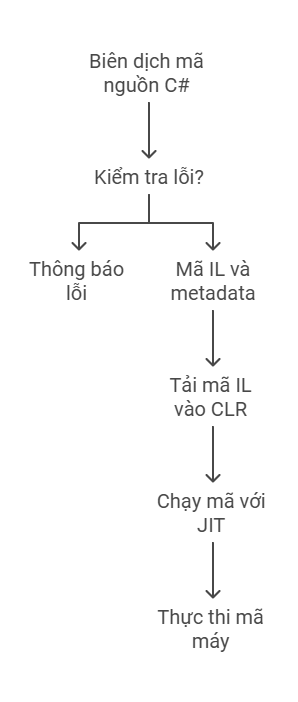
Về cơ bản gồm 2 bước: biên dịch và chạy chương trình.

#### Biên dịch (Compilation)

* **Mục tiêu:**  
  Chuyển mã nguồn C# (.cs) thành **mã trung gian (Intermediate Language - IL)** để có thể thực thi trên nền tảng .NET.
* **Quy trình:**
* **Biên dịch nhờ trình biên dịch C# (csc.exe):**
  + Mã nguồn được dịch sang mã IL (tệp .dll hoặc .exe).
  + Trong quá trình này, thông tin về cấu trúc chương trình (metadata) cũng được thêm vào.
* **Kiểm tra lỗi trong quá trình biên dịch:**
  + Trình biên dịch kiểm tra cú pháp và ngữ nghĩa của mã nguồn.
  + Nếu phát hiện lỗi, quá trình biên dịch sẽ dừng lại và báo lỗi.
* **Kết quả:**
  1. Tệp chứa mã IL và metadata, có thể được thực thi bởi .NET Runtime.

#### Chạy (Execution)

* **Mục tiêu:**  
  Thực thi chương trình bằng cách chuyển đổi mã IL thành mã máy, sau đó chạy trên CPU.
* **Quy trình:**
  1. **Tải mã IL vào CLR (Common Language Runtime):**
     + CLR tải chương trình và kiểm tra tính hợp lệ.
  2. **Chạy mã với JIT:**
     + CLR chuyển mã IL thành **mã máy (machine code)** tại thời điểm chạy.
     + Quá trình biên dịch nhờ JIT diễn ra từng phần (mỗi phương thức chỉ được dịch khi được gọi).
  3. **Thực thi mã máy:**
     + Mã máy được thực thi trực tiếp bởi CPU.



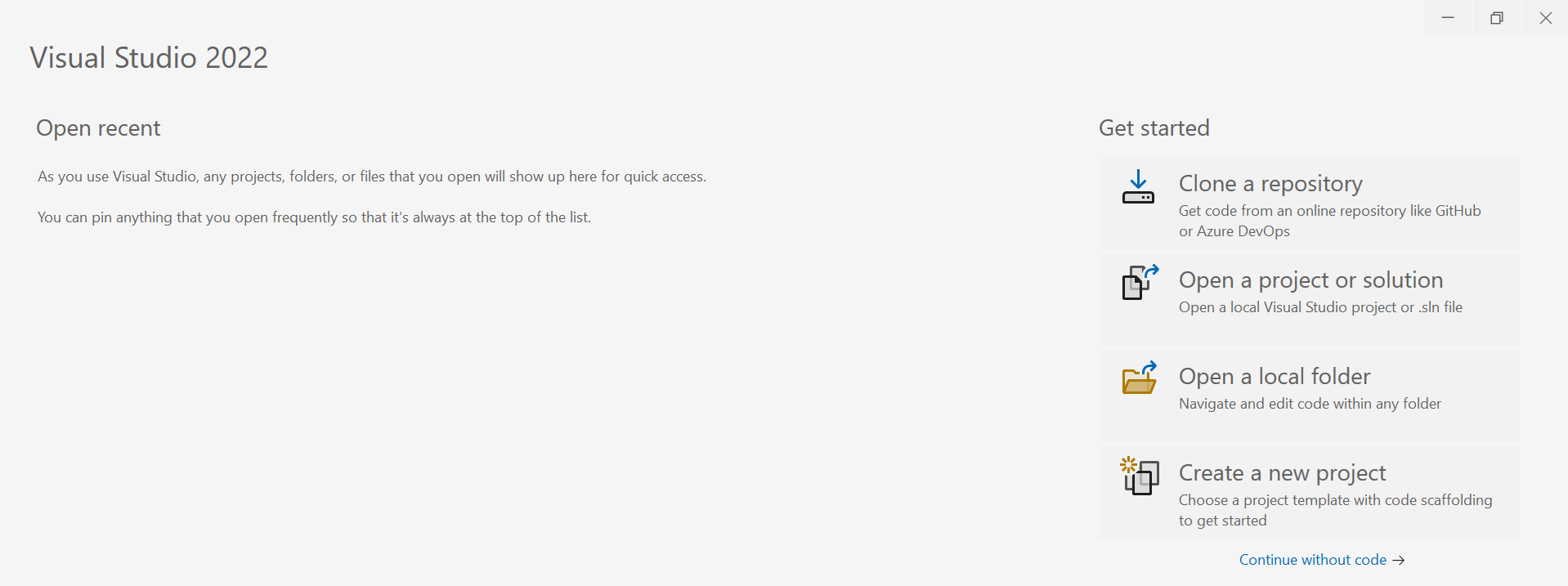
*(Đây là cái hình mô tả mà tôi nhờ AI vẽ. Cũng tạm nên lấy luôn)*

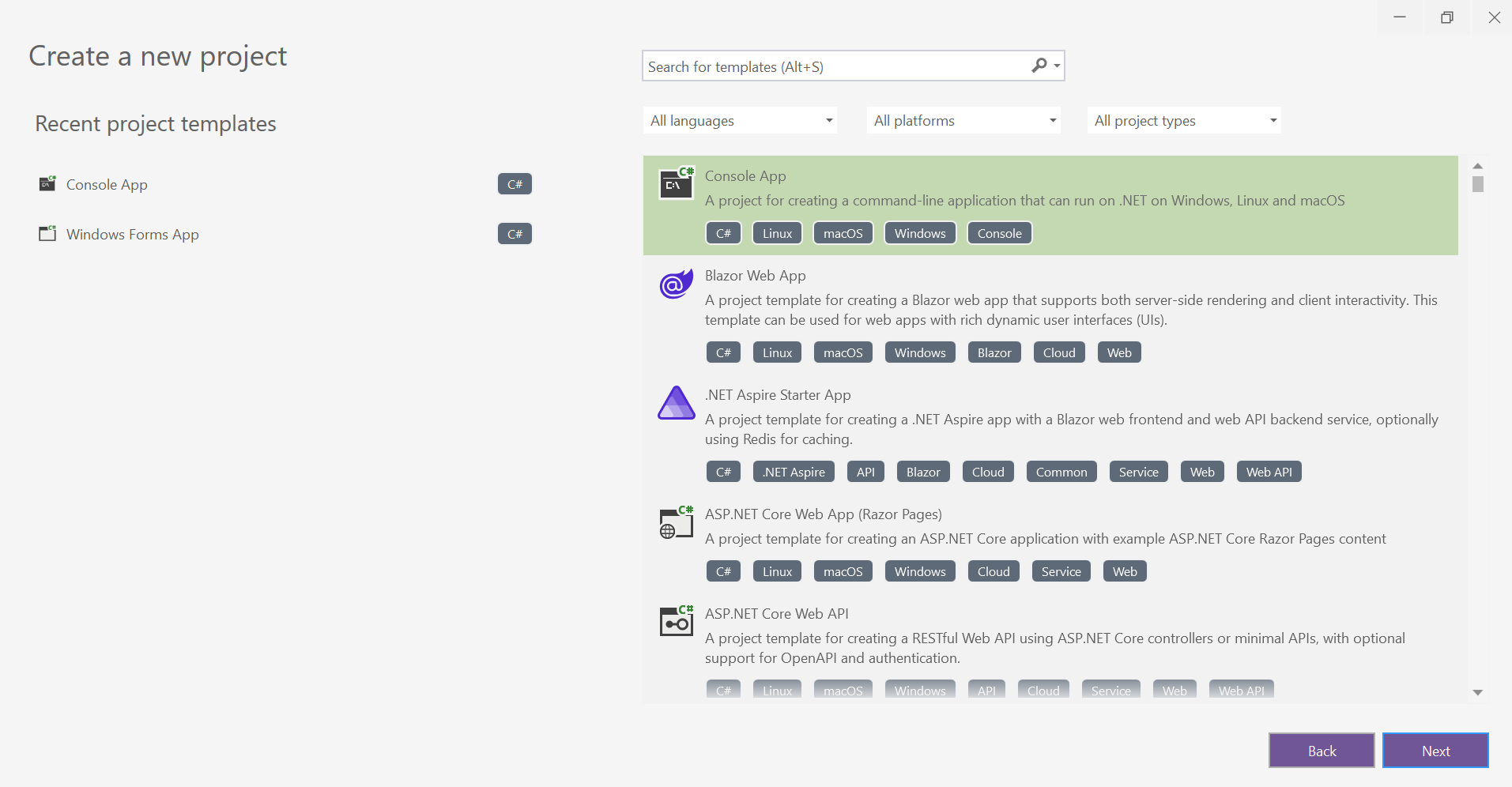
*Học lý thuyết đã quá nhiều rồi. Bây giờ cần đi vào code ngay một chút cho thay đổi không khí.*



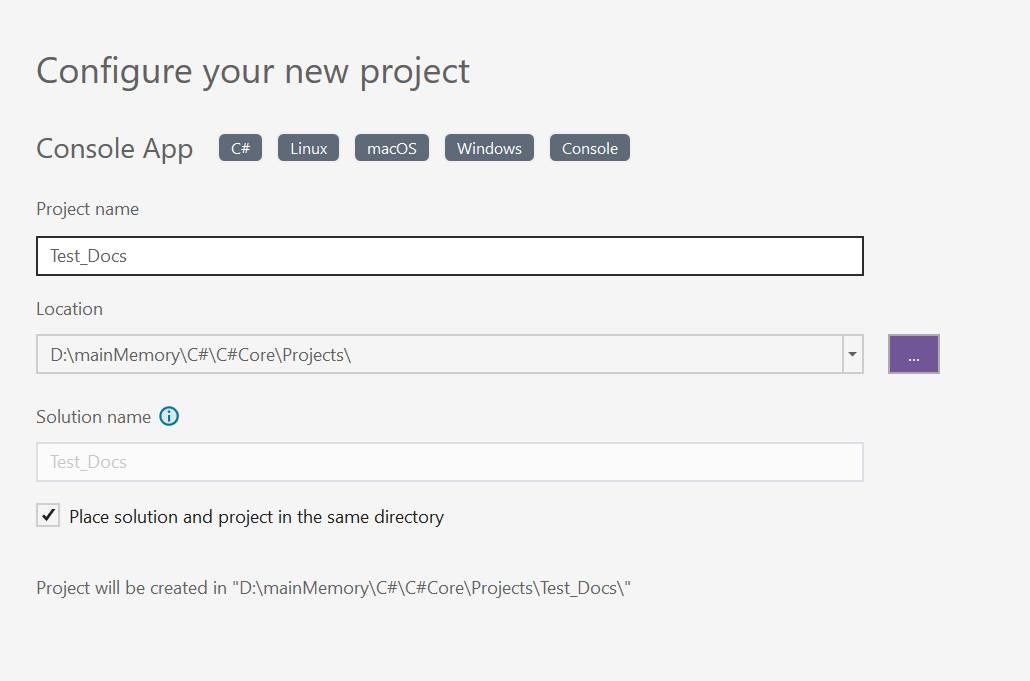
# II, Học Syntax Code

## 1, Làm quen với IDE

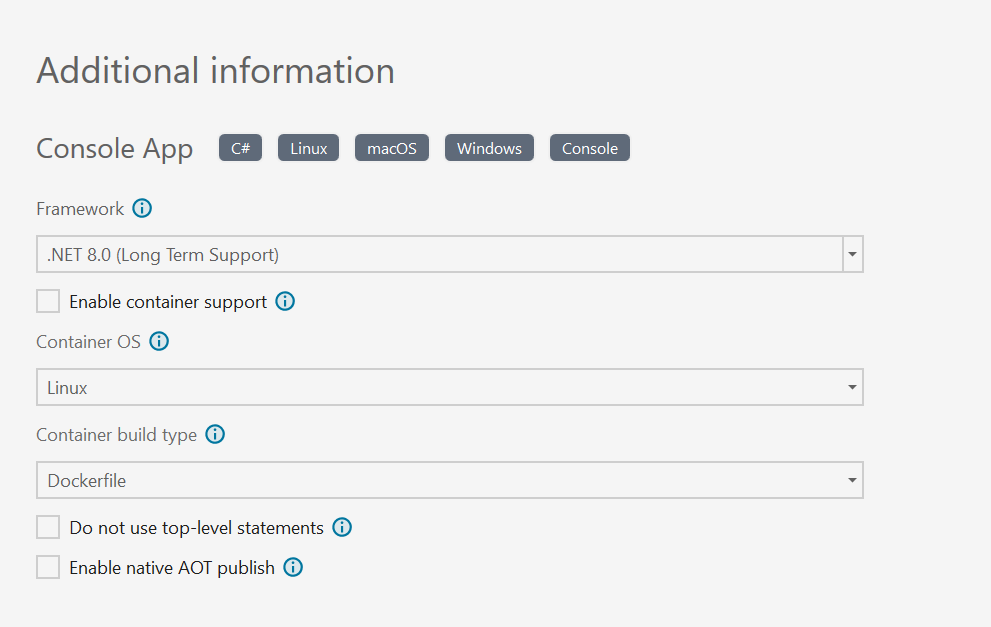
* Khi mới tải xong mở lên nó sẽ như này:  
  
* Giờ tạo thử project mới (console app):



* Đặt tên và vị trí:



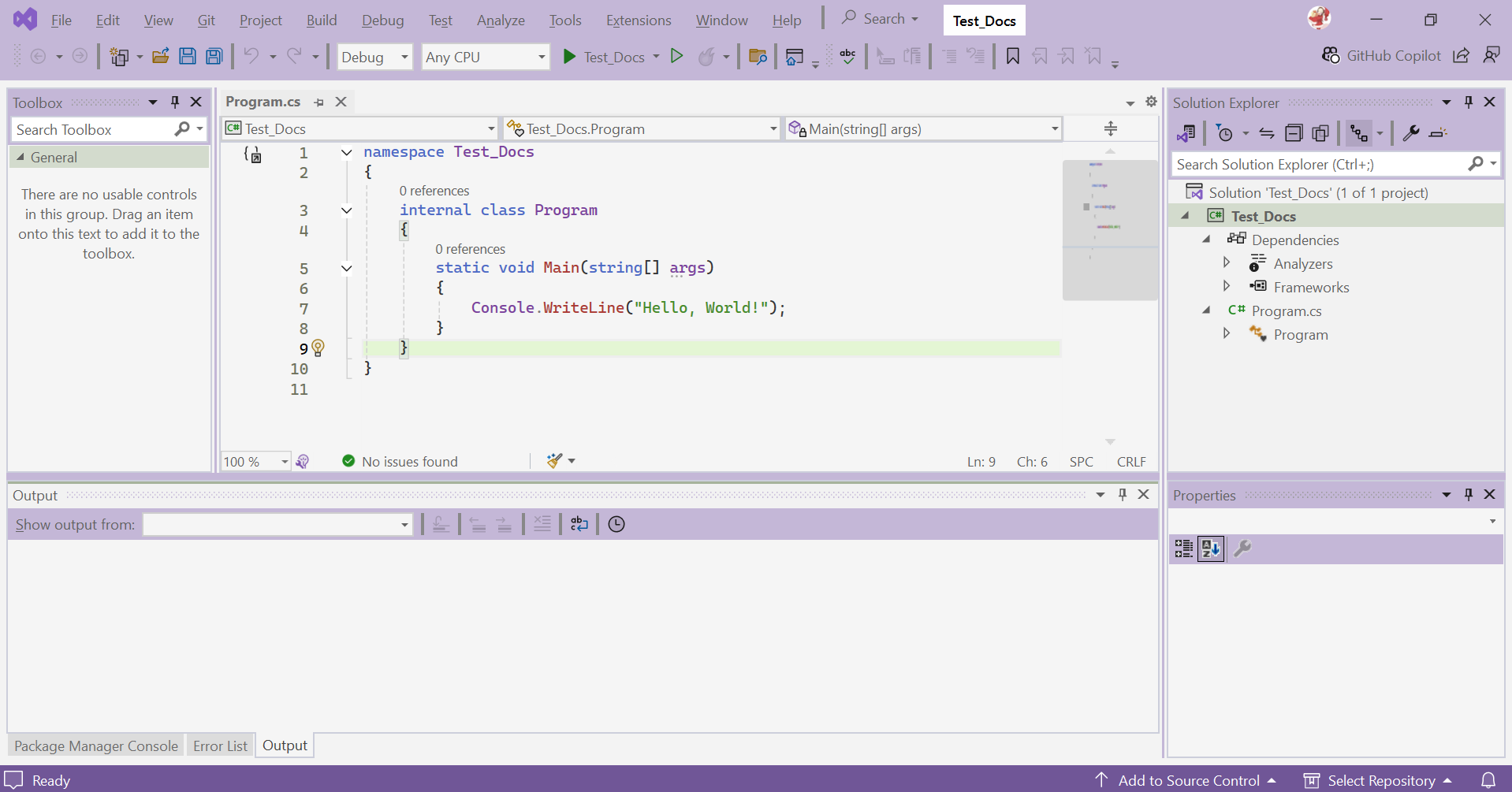
* Sau đó nó sẽ đưa ra cho mình vài lựa chọn cấu hình cho chương trình như sau:



Giải thích qua:

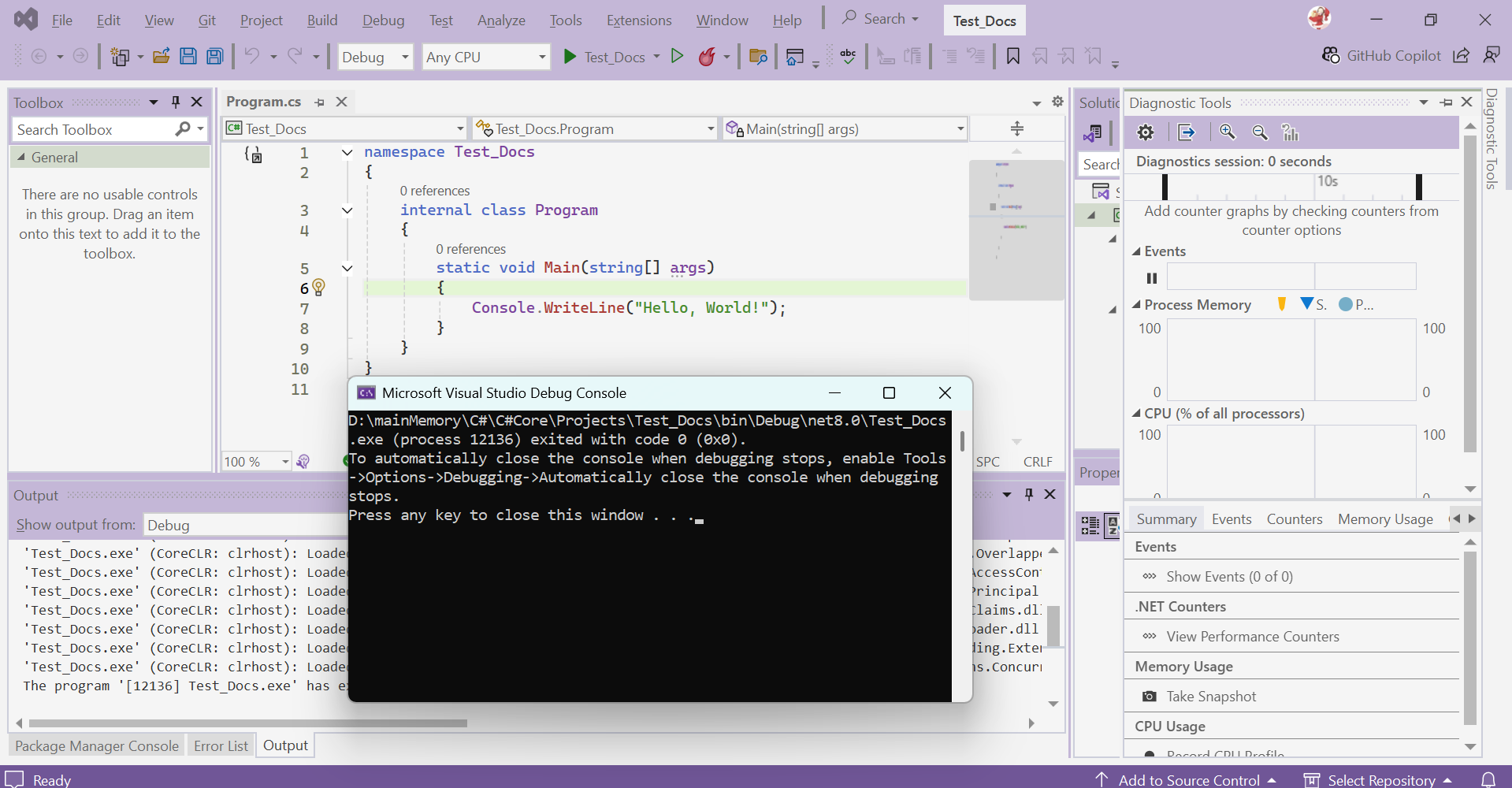
* **Framework (.NET 8.0 Long Term Support)**:
  + .NET 8.0 là phiên bản mới của nền tảng .NET với chế độ **Long Term Support (LTS)**. Điều này có nghĩa là phiên bản này sẽ được hỗ trợ lâu dài (ít nhất 3 năm), bao gồm các bản vá bảo mật và cập nhật ổn định.
  + Đây là lựa chọn tốt cho các dự án cần sự ổn định lâu dài.
* **Enable container support**:
  + Khi kích hoạt, dự án sẽ được cấu hình để hỗ trợ đóng gói dưới dạng **container** (ví dụ như Docker). Container hóa giúp ứng dụng có thể chạy đồng nhất trên nhiều môi trường khác nhau (Linux, Windows, macOS).
  + **Container OS (Linux)**:
    - Hệ điều hành **Linux** được chọn để chạy container.
    - Linux thường được ưu tiên trong các môi trường container do tính nhẹ, nhanh và phổ biến khi triển khai trên đám mây.
    - Mặc định VS sẽ chọn cái này làm container support. Nhưng mình cũng chưa cần biết lắm bao giờ học Docker thì có thể nói sau.
  + **Container build type (Dockerfile)**:
    - Lựa chọn này xác định cách thức xây dựng container.
* **Do not use top-level statements**:
  + Mặc định trong .NET 6+ khi tạo ứng dụng console, **Top-Level Statements** được sử dụng để viết mã ngắn gọn hơn (không cần khai báo Main method).
  + Nếu chọn tùy chọn này, dự án sẽ quay lại kiểu khai báo static void Main(string[] args) truyền thống. Điều này phù hợp với các lập trình viên thích viết code theo cách cũ.
* **Enable native AOT publish**:
  + **Native AOT (Ahead-of-Time)** giúp biên dịch mã .NET thành mã máy gốc (native code) trước khi ứng dụng chạy.
  + Điều này giúp: giảm kích thước tệp thực thi, tăng tốc độ khởi động ứng dụng, giảm các phụ thuộc vào runtime .NET.
  + Tuy nhiên, điều này có thể làm tăng thời gian build và không phải lúc nào cũng tương thích với tất cả thư viện.
* Theo những gì bên trên thì tôi sẽ chọn **.NET 8.0** và **Do not use top-level statements.**

Sau đó ấn tạo Project.

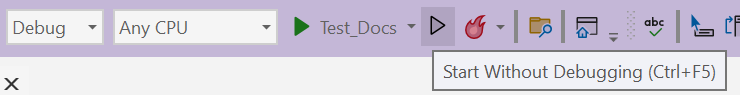


*Nhiều chức năng thế?*

Có cực kỳ nhiều chức năng ở đây, nhưng tôi không quan tâm nhiều lắm, vì chạy chương trình còn chưa nổi. Thế ấn thử vào nút Run xem?



Vậy là vừa vào chế độ Debug. Mà chương trình nhỏ, nên thôi ấn cái nút xanh nhạt nhạt bên phải xem.

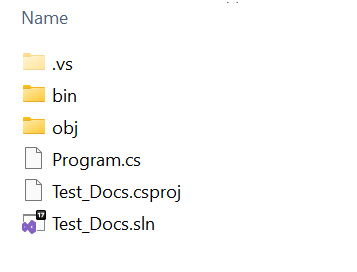


Đây là run without debug, cũng được, một phím tắt tốt để chạy chương trình khi chưa cần biết thêm cấu trúc IDE mà chỉ tập trung vào cú pháp.

Xong rồi. Bây giờ thì đi luôn vào một Project:

## 2, Cấu trúc một Project cơ bản

* Cấu trúc (trong Window Explorer) của một Project (Console App) như sau:



*Tất nhiên trong những chương trình lớn hơn và phức tạp hơn thì có cấu trúc khác, nhưng tôi cũng không đi sâu vào phần này ngay. (tôi đang muốn code lắm rồi)*

* + **Thư mục .vs**:
    - Đây là thư mục chứa các tệp cấu hình và thông tin tạm thời dành cho Visual Studio như cài đặt Debug, Cache, và User-specific settings.
    - Thường có thể bỏ qua trong hệ thống kiểm soát phiên bản (Git).
  + **Thư mục bin**:
    - Thư mục này chứa các tệp **đầu ra** (output) sau khi build dự án.
    - Các tệp .exe, .dll hoặc các tệp thực thi khác sẽ nằm trong thư mục con (**Debug** hoặc **Release**, tùy mode).
  + **Thư mục obj**:
    - Thư mục tạm thời được tạo trong quá trình biên dịch.
    - Các tệp trong thư mục này giúp hỗ trợ quá trình build, như tệp .dll trung gian, metadata hoặc file tạm khác.
  + **Program.cs**:
    - Đây là file mã nguồn chính của ứng dụng **C#**.
    - File này chứa điểm vào chính của chương trình, thường là **Main method**.
    - Có thể thay cái này bằng các folder, nhưng mà chắc là không cần.
  + **Test\_Docs.csproj**:
    - Đây là tệp cấu hình dự án **C#** (C# Project File).
    - Tệp này xác định thông tin dự án:
      * Target framework (ví dụ: .NET 8.0)
      * Các package hoặc thư viện được tham chiếu (NuGet packages)
      * Cấu hình build và publish.
  + **Test\_Docs.sln**:
* Đây là file **Solution** của Visual Studio.
* Tệp này chứa thông tin về các dự án bên trong solution, giúp Visual Studio quản lý nhiều dự án trong cùng một solution.

Thế phần solution là gì vậy?

### Solution

#### Định nghĩa

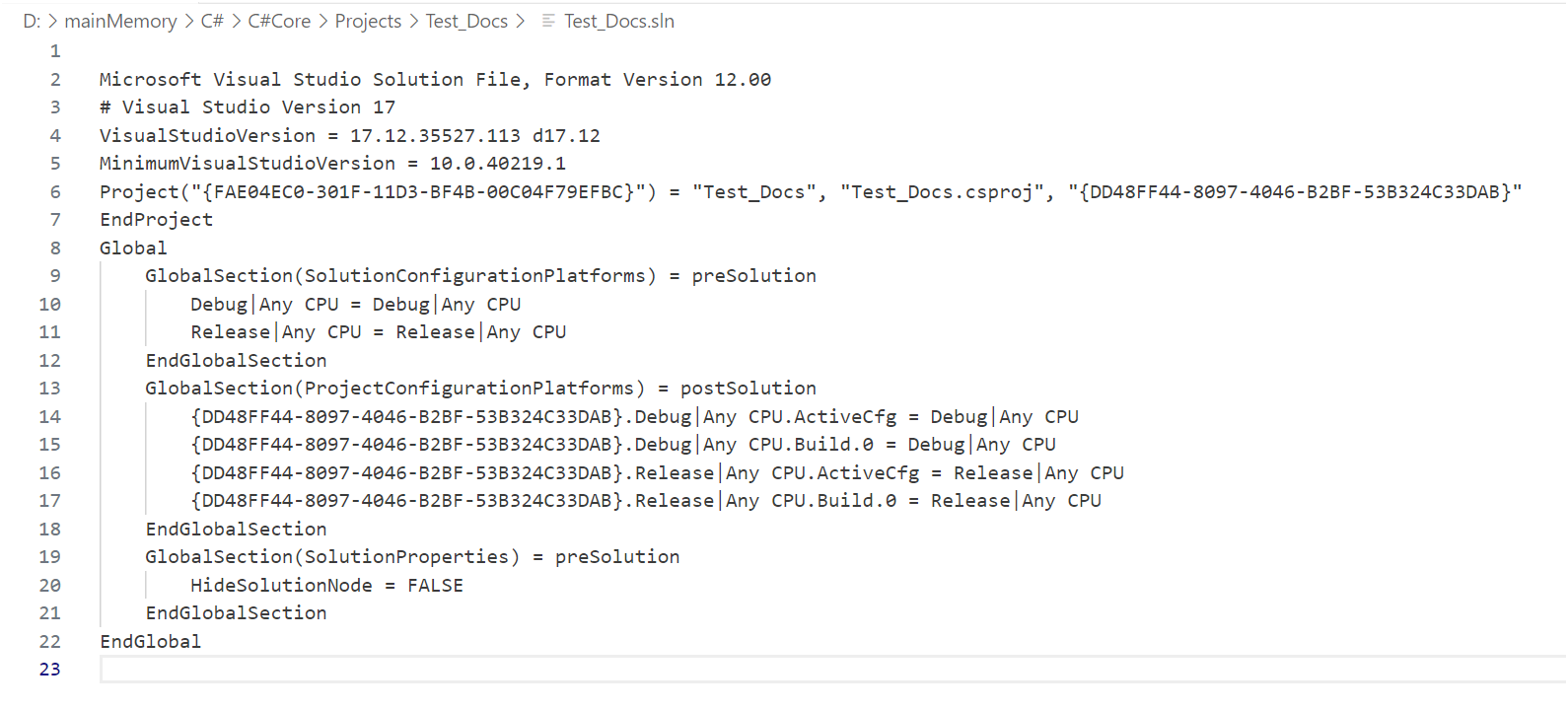
* Solution đơn giản là **container** chứa các project con.
* Nó không phải là một gói chứa tất cả các project, nó chỉ mang khả năng tham chiếu:

Một file .sln lưu **thông tin về cấu trúc và mối quan hệ giữa các project**. Nó **không chứa trực tiếp mã nguồn**, mà tham chiếu tới các file .csproj, .vbproj, v.v.

* Có chức năng quản lý các project liên quan với nhau trong cùng một môi trường. “Quản lý” ở đây có thể là điều phối mối quan hệ giữa các Projects (A có phụ thuộc B hay không, A dùng được chức năng gì từ B), cấu hình Build và Debug (để chạy một hay nhiều Project một lúc),

*Nghĩa là có thể có nhiều Project trong một Solution. Định nghĩa chương trình lớn nhất từ đây chuyển về Solution chứ không phải Project.*

Như một project mới tạo của tôi thì nó sẽ có file .sln như này:



Nhìn chung thì nó sẽ có 3 phần:

* + Dòng 1-7: Định nghĩa project
  + Dòng 9-12: Solution Configuration Platforms (Cấu hình Debug/Release cho toàn bộ Solution)
  + Dòng 13-18: Project Configuration Platforms (Cấu hình riêng cho từng Project trong Solution)
* Từ phần này chỉ cần rút ra là, với mục tiêu là học syntax với Console App thì cần để ý vào file cs để code là một, file exe/dll trong mục Debug/Release để chạy là hai, và nếu có thì thêm file .sln.

Bên trên thì tôi đã biết thế nào là file .exe và thế nào là .dll rồi. .exe là để chạy còn .dll là file thư viện. Nhưng tôi chưa rõ lắm, khi nào thì tạo ra .exe và khi nào tạo ra .dll? Và tại sao lại cần tới 2 loại file chứa IL, mục tiêu của nó khác nhau như thế nào?

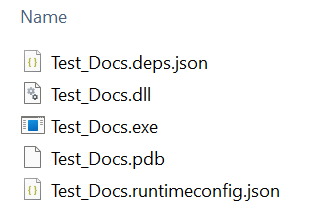


*(đáng lẽ phải viết phần này từ phần hoạt động nhưng nó over chỗ đó rồi nên để dưới đây viết)*

### File exe và dll

#### Vị trí trong project và tại sao lại ở đó

* Khi tôi ấn vào Run cái chương trình Hello World nó có sẵn trong Project tôi đã tạo, trong phần biên dịch ra tôi nhận được những file này trong thư mục Release:

**

*Nhìn vào đây thì có 2 câu hỏi: tại sao chỉ một chương trình code chỉ có mỗi in ra Hello World mà lại có cả file .dll và .exe? Và những file còn lại có chức năng gì?*

* Tôi không có câu trả lời cho câu hỏi thứ nhất, nên là tôi thử hỏi GPT:



* Mã IL được lưu trong dll để hỗ trợ **đa nền tảng.**
* File **exe** chỉ chứa **một stub** (đoạn mã khởi động nhỏ).
* Stub này sẽ gọi vào mã IL trong **dll** để chạy ứng dụng.

Thì ra vấn đề nằm ở cách hoạt động của phiên bản .NET 5.0+, nó sẽ sinh ra file exe như đầu mối kích hoạt và những logic code được xử lý ở phần file .dll.

Thêm nữa thì tôi không phải là người đầu tiên thắc mắc về vấn đề này, và thực sự Microsoft confirm:

<https://learn.microsoft.com/en-us/answers/questions/699745/combine-the-compiled-dll-with-exe>

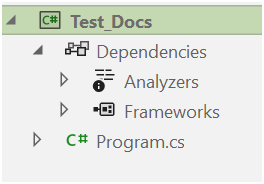
Còn về vấn đề các file đó có chức năng gì trong chương trình thì tiếp tục GPT (vì tôi thực sự chẳng biết gì cả):

* **Test\_Docs.dll**
* Đây là file chính chứa **Intermediate Language (IL)** của ứng dụng, cùng với **metadata** miêu tả cấu trúc mã.
* Nó được tạo ra sau khi build ứng dụng và có thể được sử dụng bởi nhiều file thực thi hoặc thư viện khác.
* **Test\_Docs.exe**
* Là file thực thi chính của ứng dụng, dùng để khởi động ứng dụng.
* Nó thường gọi file .dll để thực thi các chức năng cụ thể.
* **Test\_Docs.deps.json**
* File này liệt kê tất cả các **dependency** (gói, thư viện, hoặc framework) cần thiết khi ứng dụng chạy.
* Nó giúp runtime biết được vị trí và phiên bản của các gói phụ thuộc.
* **Test\_Docs.pdb**
* File này chứa các **debugging symbols**, hỗ trợ quá trình gỡ lỗi (debugging).
* Nó ánh xạ mã nguồn của ứng dụng với mã IL, giúp nhà phát triển xác định lỗi dễ dàng hơn.
* **Test\_Docs.runtimeconfig.json**
* File này chứa các thiết lập cấu hình runtime, như phiên bản .NET cần thiết để chạy ứng dụng.
* Nó đảm bảo rằng ứng dụng được chạy với môi trường phù hợp.
* Như vậy là đã khá rõ những thành phần của folder Projects rồi. Vào Visual Studio đi đợi lâu quá rồi.



### (Tiếp tục vào làm quen với Visual Studio)

Vào project trong Visual Studio thì tôi thấy cái này:



Có thể hiểu qua như sau:

* **Dependencies**: Chứa các phụ thuộc (nuget packages, SDK).
* **Analyzers**: Các công cụ để phân tích và đưa ra cảnh báo code.
* **Frameworks**: Hiển thị .NET SDK/framework đang dùng.
* **Program.cs**: File mã nguồn chính trong dự án, nơi chứa Main().

Vì nãy tôi chọn **Do not use top-level statements** (vẫn lựa chọn dùng hàm Main()) nên tôi vẫn viết theo Main() cho truyền thống.

## 3, Syntax

Syntax, syntax, syntax.



***Từ đây tôi sẽ viết phần cơ bản dựa theo web*** [***https://xuanthulab.net/***](https://xuanthulab.net/)***, đây là một nơi tốt để tiếp thu cái syntax này.***

### 1, Phạm vi tổ chức của chương trình C#

Trước khi học syntax cơ bản thì cần học syntax và cấu trúc của một mã nguồn. (đây là cấu trúc của ***mã nguồn***, không phải cấu trúc của ***Project***)

* Phạm vi tổ chức: là thành phần cơ bản để tổ chức mã nguồn.
* Nghĩa là các dòng code và logic được gói lại như thế nào khi gửi đến Complier.

Về tổ chức code sẽ có 5 phạm vi tổ chức (không phải Access Modifier):

#### 1, Field, Property và Method

* Trường, thuộc tính và phương thức là những phạm vi (chắc là) nhỏ nhất của mã nguồn.
* Field và Property là 2 cái khác nhau:
  + Field là biến lưu trữ dữ liệu trong class, struct, và ý nghĩa của nó chỉ dừng lại ở phạm vi bên trong class hay struct đó. Nó thường dùng cho những constant – hằng số. Nói chung thì nó là biến đơn giản, truy cập trực tiếp, ít kiểm soát.
  + Property là một khái niệm gắn liền với OOP. Nó không chỉ mang tính chất lưu trữ như Field mà còn cung cấp cơ chế cơ chế truy cập và thay đổi giá trị cho những thành phần ngoài class. Nếu như vậy là nó tương đương với getter/setter của Java?



*Thuộc tính trong C# có 2 loại: thuộc tính thường (non auto-implemented) và thuộc tính auto-implemented. Loại 2 cho phép tinh giảm cú pháp getter/setter đi, nghĩa là sẽ có getter/setter mặc định cho Property trong class (OOP). Cái này sẽ nói sau trong phần syntax OOP.*

* + Khai báo và gán giá trị cho một Field trong C# sẽ như sau:

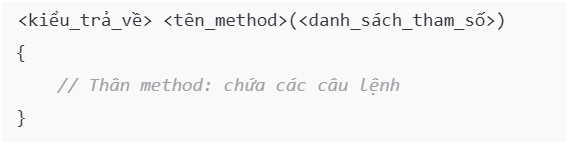


*(vẫn cần dấu chấm phẩy nhé)*

Khai báo Property sẽ được tính sau.

* Method – phương thức, một khối mã thực hiện một nhiệm vụ hoặc tính toán cụ thể. Một method có thể nhận tham số đầu vào và trả về giá trị. (như Java)

Cú pháp của nó cũng tương tự như Java:

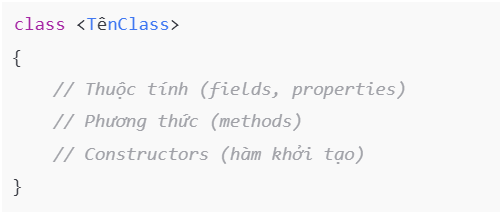


*Phần này chắc khỏi cần VD.*

#### 2, Class

* Lớp – là một kiểu dữ liệu người dùng định nghĩa, được sử dụng để tạo ra các đối tượng.
* Một class là khuôn mẫu để tổ chức dữ liệu (thuộc tính) và hành vi (phương thức).

Cú pháp tổng quát của class trông như sau:



Về cú pháp cụ thể thì nói sau trong phần OOP nhé. Cú pháp ảo ma phết.



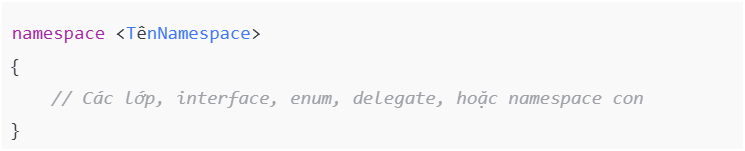
* Struct là một kiểu dữ liệu giá trị (value type) dùng để nhóm các thuộc tính và phương thức liên quan.
* Struct tương tự như **class**, nhưng thường được dùng để biểu diễn các đối tượng đơn giản và nhỏ, không cần tính năng nâng cao như kế thừa.
* “class” ở đây được dùng với nghĩa chung, nghĩa là với OOP nó bao gồm cả class, interface, abtract class, …

#### 3, Namespace

* Nghĩa đen là “không gian tên”. Nghĩa là:

Trong một chương trình lớn hoặc khi sử dụng nhiều thư viện, có thể xảy ra việc hai thành phần có cùng tên. Namespace tạo ra một không gian riêng biệt cho mỗi nhóm thành phần, giúp phân biệt chúng.

* Một vài tính chất của namespace:
  + **Tổ chức mã nguồn**: Nhóm các thành phần liên quan (class, interface, enum,...) lại với nhau.
  + **Tránh xung đột tên**: Các thành phần cùng tên trong namespace khác nhau không bị xung đột, cũng có nghĩa là không được trùng tên trong cùng một namespace.
  + **Cấu trúc phân cấp**: Hỗ trợ namespace lồng nhau để tổ chức rõ ràng. Namespace này có thể chứa namespace con.
  + **Using**: Truy cập các namespace bằng từ khóa “using”.
  + **Không bị ràng buộc**: Namespace chỉ là logic, không ảnh hưởng đến cách lưu trữ vật lý của file. Nghĩa là tên namespace thế nào cũng được, không phải giống tên folder như cách đặt tên package trong Java. Không biết cái này là tốt hơn hay kém hơn nữa nhưng mà tôi vẫn đang prefer cách làm bên Java.
* Cú pháp của namespace như sau:



* Nhìn thì có vẻ nó khá tương đồng với **package** của Java.
* Có một đặc điểm quan trọng khác với package là namespace không yêu cầu phụ thuộc tên thư mục vật lý.
* Điều đó suy ra namespace chỉ là không gian logic (mình và máy hiểu chứ không được thể hiện rõ trong folder) và kéo theo việc namespace không hỗ trợ kiểm soát truy cập qua Access Modifier.
* Một namespace bao gồm nhiều class, interface,… và cả namespace con nữa.

#### 4, Assembly

<https://learn.microsoft.com/en-us/dotnet/standard/assembly/>

* Từ đây trở đi các đơn vị không phải là một “phạm vi mã nguồn” như 3 cái trên nữa, bởi vì nó chứa mã trung gian.
* Đồng nghĩa với việc nó không có một cú pháp nào để nhận biết cụ thể.

Vậy nó là gì và phạm vi nó rộng tới đâu?

* Một Assembly – nghĩa đen là một “tập hợp đã được lắp ráp” – là **một đơn vị triển khai và thực thi**  cơ bản trong .NET. (đọc theo ngắt Space cho dễ nhìn).
* Một Assembly có thê hiểu là một tệp đã được biên dịch (thành file .exe hoặc .dll).
* Một Assembly có thể bao gồm nhiều namespace, class,… và thậm chí tài nguyên nhúng (hình ảnh, file, dữ liệu nhúng).



*Cực kỳ mù mờ trong khi viết phần này vì sự không rõ ràng trong khái niệm khi nói về phạm vi.*



Nếu thấy chưa được ổn lắm thì có thể coi phạm vi của nó tương đương với file .jar/war bên Java, nơi chứa các package và class.

(…)

#### 5, Solution

* Cái này cũng nói ở bên trên phần ***Solution*** rồi.
* Nó cũng là khái niệm nằm ngoài “phạm vi mã nguồn” như Assembly.
* Mỗi Solution có thể chứa nhiều Project, đồng nghĩa với việc nó **có thể chứa nhiều Assembly**.
* Có thể coi nó tương đương với một Workspace (tập hợp dự án) trong Java.
* Tương tự như Assembly thì Solution không có cú pháp cụ thể để nhận biết.

#### Tóm tắt và kết luận

* Có thể theo thứ tự bé => lớn như sau:

Field/Property/Method -> Class -> Namespace ->Assembly ->Solution

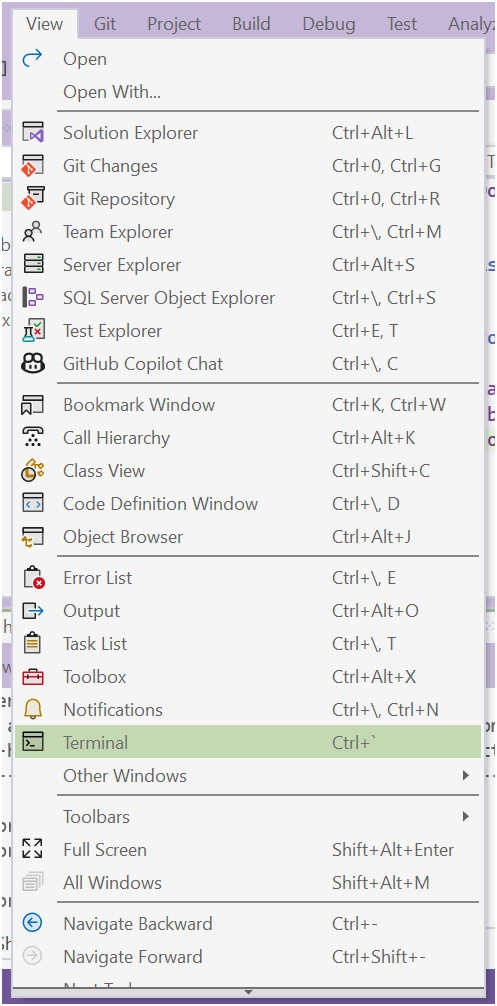
* Bảng so sánh chúng với đơn vị tương đương bên Java (cho dễ nhìn dễ nhớ):

| **Mức tổ chức (C#)** | **Vai trò** | **Vị trí tương đương trong Java** |
| --- | --- | --- |
| **Field** | Biến lưu trữ dữ liệu trong class. | **Field**: Biến trong class (khai báo trực tiếp trong lớp). |
| **Property** | Cách truy xuất và thiết lập dữ liệu an toàn, thông qua getter và setter. | **Getter/Setter**: Phương thức lấy và đặt giá trị (không có từ khóa riêng). |
| **Method** | Phương thức xử lý logic, được khai báo bên trong class. | **Method**: Hàm thành viên của class. |
| **Class** | Định nghĩa đối tượng, nhóm các thành phần như field, property, method. | **Class**: Tương tự, nhóm các thành phần liên quan. |
| **Struct** | Kiểu giá trị dùng để lưu trữ dữ liệu nhẹ, giống class nhưng không hỗ trợ kế thừa. | **Class** (với kiểu giá trị): Java không có struct. |
| **Namespace** | Gom nhóm nhiều class, tránh xung đột tên (không cần khớp cấu trúc thư mục). | **Package**: Gom nhóm class, bắt buộc khớp với cấu trúc thư mục vật lý. |
| **Assembly** | Tệp thực thi (.exe) hoặc thư viện (.dll) chứa nhiều namespace và class. | **JAR/WAR**: Tệp lưu trữ chứa các package và class. |
| **Solution** | Tập hợp các dự án (project) liên quan, quản lý nhiều assembly. | **Workspace/Project Group**: Tập hợp các dự án trong IDE (Eclipse, IntelliJ). |

Phần trên chỉ để code cho có hệ thống (về sau).

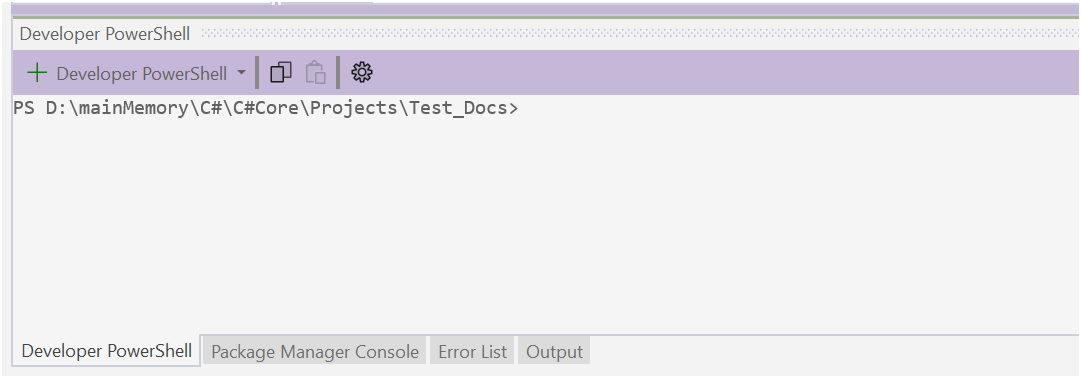
### 2, Run một chương trình với VS

* Như trên có cách ấn nút Run và Run Without Debug rồi, nhưng nếu thấy chưa đủ thì tôi còn làm thêm cách mở Terminal trong Visual Studio.
* Cụ thể:
  + Vào Views:

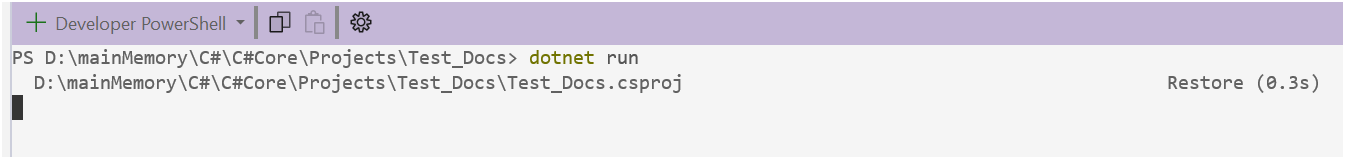


Có nhiều thứ để chọn nhưng hiện tại hãy chọn Terminal.

Sau đó ở phần Output sẽ hiện lên Terminal (Developer PowerShell):



Ở đây nếu gõ lệnh “dotnet run”, VS sẽ tìm đến file .csproj để chạy chương trình:



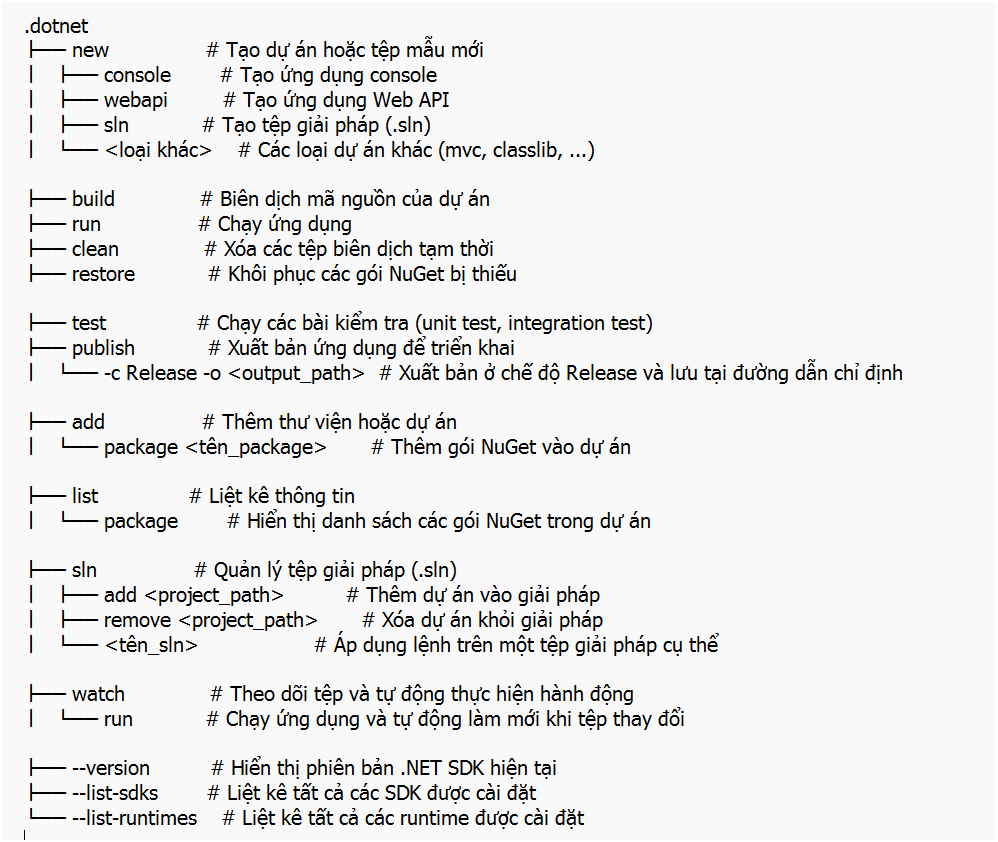
*(mà tại sao lại có thể chạy như vậy?)*



* Như vậy là có thể làm quen với các câu lệnh CLI.
* Cũng đặt ra câu hỏi một câu lệnh CLI sẽ hoạt động như thế nào để chạy chương trình? Quy trình trước khi quá trình Cách hoạt động và luồng chương trình cơ bản xảy ra?

#### Giới thiệu câu lệnh CLI

Để nói về tất cả câu lệnh CLI thì không phải mục tiêu bây giờ. Nhưng tôi có thể liệt kê ra vài lệnh cơ bản như sau:



Các lệnh khác đầy đủ hơn có thể xem ở đây:

<https://learn.microsoft.com/en-us/dotnet/core/tools/>

*(phần này mang tính chất giới thiệu nhiều hơn là hiểu)*

Yêu cầu ở đây chắc là nhớ một vài lệnh cơ bản để chạy chương trình.

#### Bonus: khi nhận câu lệnh “dotnet run”, chương trình sẽ làm gì?



Thì đây là quá trình:

| **Giai đoạn** | **Tệp/Thư mục sử dụng** | **Vai trò** |
| --- | --- | --- |
| **1. Xác định dự án** | - MyApp.csproj | Xác định thông tin dự án như framework, cấu hình biên dịch, và phụ thuộc. |
| **2. Khôi phục phụ thuộc** | - Thư mục obj/ | Lưu thông tin về các phụ thuộc (gói NuGet, tham số biên dịch). |
|  | - Cache NuGet (~/.nuget/packages hoặc %USERPROFILE%\.nuget\packages) | Lấy các gói NuGet cần thiết cho dự án. |
| **3. Biên dịch mã nguồn** | - Các tệp \*.cs trong thư mục dự án | Chứa logic của chương trình, được biên dịch thành tệp thực thi (.dll). |
|  | - Thư mục bin/<Configuration>/<Framework> | Lưu trữ tệp nhị phân đã biên dịch (MyApp.dll) và các tài nguyên khác. |
| **4. Chạy chương trình** | - Tệp thực thi (bin/<Configuration>/<Framework>/MyApp.dll) | Tệp nhị phân được chạy bằng runtime .NET. |
| **5. Sử dụng runtime**  **.NET** | - Runtime cài sẵn trên hệ thống | Cung cấp môi trường để thực thi tệp DLL. |



CHANGE WRITING STYLE

**(từ đoạn này trở đi tính chính xác và tính khai thác sẽ không được đảm bảo và sẽ hạ xuống mức thấp nhất)**

### 3, Top-level Statement

* Mỗi chương trình C# đều có điểm mồi là hàm Main. Nhưng bắt đầu từ C#9 không cần khai báo hàm này nữa (còn cứ khai báo như cũ cũng không sao).
* Toàn bộ những chỉ thị lệnh (mệnh đề, statement) không viết trong namespace nào, không viết trong lớp nào thì trình biên dịch sẽ tự động nhận biết đây là các mã lệnh của hàm Main và tự động sinh ra hàm Main.
* Có thể tách riêng phần Main thành một file dù vẫn không viết trong namespace hay class nào. Nếu cần nạp namespace thì vẫn dùng lệnh using.
* Các lệnh với aspnet cũng có thay đổi trong những phiên bản mới (học đến rồi nói sau).

### 4, Biến và kiểu dữ liệu trong C#

#### Biến

* Trong C#, một biến là một vùng nhớ được đặt tên, dùng để lưu trữ giá trị dữ liệu trong chương trình.
* Quy tắc đặt tên vẫn giống trong Java.
* Tránh trùng từ khóa trong C#:

|  |
| --- |
| **abstract** **as** **base** bool **break** byte **case** |
| **catch** char checked **class** **const** **continue** **decimal** |
| **default** **delegate** **do** **double** **else** **enum** **event** |
| **explicit** **extern** **false** **finally** **fixed** **float** **for** |
| **foreach** **goto** **if** **implicit** **in** **int** |
| **interface** **internal** **is** **lock** **long** **namespace** **new** |
| **null** **object** **operator** **out** **out** **override** **params** |
| **private** **protected** **public** **readonly** **ref** **return** **sbyte** |
| **sealed** **short** **sizeof** **stackalloc** **static** **string** **struct** |
| **switch** **this** **throw** **true** **try** **typeof** **uint** |
| **ulong** **unchecked** **unsafe** **ushort** **using** **virtual** **void** |
| **volatile** **while** |
|  |
| **add** **alias** **ascending** **descending** **dynamic** **from** **get** |
| **global** **group** **into** **join** **let** **orderby** |
| **partial** **remove** **select** **set** |

* Cú pháp khai báo biến: **datatype variable\_name**;

#### Kiểu dữ liệu

* **Ngoài cách khai báo thông thường thì C# có thể khai báo biến với từ khóa “var”.** 
  + - Khi khởi tạo với “var”, kiểu dữ liệu sẽ được tự động detect bằng giá trị của biến.
    - Khởi tạo một biến với “var” yêu cầu gán giá trị ngay khi khởi tạo. (“var a; a = 3;” là sai, “var a = 3;” là đúng).
* Muốn khai báo hằng số: dùng thêm từ khóa “const”.

VD: const string str = “abc”;

* Muốn dữ liệu kiểu string xuất ra terminal có dấu, sử dụng mã hóa UTF-8. Đặt câu lệnh sau vào đầu chương trình, cụ thể là trước khi in string ra Console.

**Console.OutputEncoding = System.Text.Encoding.UTF8;**



Phân loại kiểu dữ liệu trong C#:

[*https://www.geeksforgeeks.org/c-sharp-data-types/*](https://www.geeksforgeeks.org/c-sharp-data-types/)

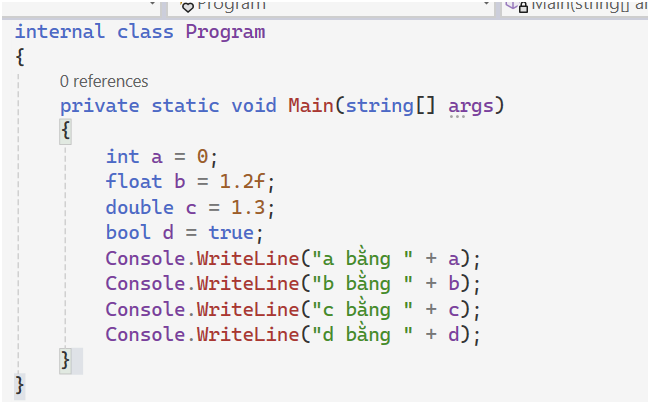
* Có 3 kiểu dữ liệu:
  + Value Type (kiểu giá trị)
  + Reference Type (kiểu tham chiếu)
  + Pointer Type (kiểu con trỏ)

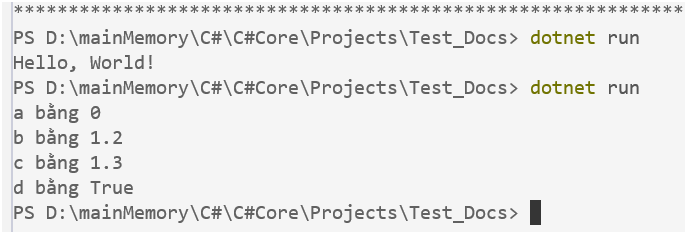
##### Value type

* Kiểu dữ liệu giá trị trong C# khác so với kiểu dữ liệu nguyên thủy (Primitive Type) trong Java. (VD, int trong C# khác cấu trúc so với int trong Java)
* Kiểu dữ liệu int, long, byte, … trong C# là những alias cho kiểu Struct dựng sẵn (do C# dựng) trong khi trong Java nó thực sự là 1 kiểu dữ liệu nguyên thủy.
* Điều này làm cho kiểu dữ liệu int,… trong C# có thêm một vài phương thức (do là Struct nên có thể mang theo method).
* Bảng tổng hợp các kiểu dữ liệu giá trị:

| **Alias** | **Type Name** | **Type** | **Size (bits)** | **Range** | **Default Value** |
| --- | --- | --- | --- | --- | --- |
| byte | System.Byte | Unsigned | 8 | 0 to 255 | 0 |
| sbyte | System.SByte | Signed | 8 | -128 to 127 | 0 |
| short | System.Int16 | Signed | 16 | -32,768 to 32,767 | 0 |
| ushort | System.UInt16 | Unsigned | 16 | 0 to 65,535 | 0 |
| int | System.Int32 | Signed | 32 | -2,147,483,648 to 2,147,483,647 | 0 |
| uint | System.UInt32 | Unsigned | 32 | 0 to 4,294,967,295 | 0 |
| long | System.Int64 | Signed | 64 | -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 | 0 |
| ulong | System.UInt64 | Unsigned | 64 | 0 to 18,446,744,073,709,551,615 | 0 |
| float | System.Single | Floating | 32 | ±1.5 × 10⁻⁴⁵ to ±3.4 × 10³⁸ | 0.0f |
| double | System.Double | Floating | 64 | ±5.0 × 10⁻³²⁴ to ±1.7 × 10³⁰⁸ | 0.0d |
| decimal | System.Decimal | Floating | 128 | ±1.0 × 10⁻²⁸ to ±7.9 × 10²⁸ | 0.0m |
| char | System.Char | Unicode | 16 | U+0000 to U+FFFF (Unicode 16-bit) | '\0' |
| bool | System.Boolean | Boolean | 8 (theoretically) | true or false | false |

Vd:





##### Reference type

* Kiểu dữ liệu tham chiếu sẽ chứa địa chỉ bộ nhớ có giá trị biến.
* Kiểu tham chiếu sẽ không lưu trữ giá trị biến trực tiếp trong bộ nhớ.
* Khi tạo một biến kiểu tham chiếu, chẳng hạn như một đối tượng hoặc một chuỗi, thực tế là đang lưu trữ một tham chiếu (hoặc con trỏ) đến vị trí trong bộ nhớ nơi dữ liệu được lưu giữ.
* Dữ liệu thực tế cho các loại tham chiếu được lưu trữ trên heap. Heap là một vùng bộ nhớ lớn được sử dụng để phân bổ bộ nhớ động. Các kiểu tham chiếu tích hợp là string, object.

###### string

* string (không viết hoa) là alias cho kiểu System.String.

###### object

* Tất cả những kiểu dữ liệu trong C# đều kế thừa từ Object.
* Class chính thức là System.Object.
* Khi một kiểu dữ liệu được đổi về Object thì được gọi là quá trình “boxing”
* Khi Object chuyển đổi thành một kiểu dữ liệu thì gọi là quá trình “un-boxing”

### Xuất dữ liệu ra màn hình Console

* Dùng class Console.

<https://learn.microsoft.com/en-us/dotnet/api/system.console?view=net-9.0>

* Class Console thể hiện các luồng đầu vào, đầu ra và lỗi tiêu chuẩn cho các ứng dụng Console. Lớp này không thể được kế thừa.
* Dưới đây là các phương thức nhập/xuất cơ bản của **System.Console** trong C#:

| **Phương thức** | **Mô tả** | **Ví dụ** |
| --- | --- | --- |
| **Write(string value)** | Ghi chuỗi văn bản ra console mà không xuống dòng. | Console.Write("Hello "); Console.Write("World!"); |
| **WriteLine(string value)** | Ghi chuỗi văn bản ra console và tự động xuống dòng. | Console.WriteLine("Hello, World!"); |
| **ReadLine()** | Đọc toàn bộ chuỗi ký tự nhập từ người dùng (kết thúc khi nhấn Enter). | string input = Console.ReadLine(); Console.WriteLine("You entered: " + input); |
| **ReadKey()** | Đọc một phím bấm từ người dùng mà không yêu cầu nhấn Enter. | Console.WriteLine("Press any key to continue..."); Console.ReadKey(); |
| **Read()** | Đọc một ký tự từ luồng đầu vào và trả về mã ASCII của ký tự đó. | int asciiCode = Console.Read(); Console.WriteLine("ASCII code: " + asciiCode); |
| **Clear()** | Xóa toàn bộ nội dung trong console. | Console.Clear(); |
| **SetCursorPosition(int left, int top)** | Đặt vị trí con trỏ đầu vào/xuất tại cột (left) và dòng (top). | Console.SetCursorPosition(10, 5); Console.Write("Text at (10, 5)"); |
| **Beep()** | Phát ra âm thanh bíp mặc định từ hệ thống. | Console.Beep(); |
| **Beep(int frequency, int duration)** | Phát ra âm thanh bíp với tần số và thời gian chỉ định (tần số: Hz, thời gian: milliseconds). | Console.Beep(1000, 500); |
| **WriteLine(object value)** | Ghi giá trị của đối tượng (sử dụng phương thức ToString()) ra console và xuống dòng. | Console.WriteLine(123); Console.WriteLine(DateTime.Now); |
| **Write(string format, params object[] args)** | Ghi chuỗi được định dạng ra console mà không xuống dòng. | Console.Write("Hello, {0}!", "World"); |
| **WriteLine(string format, params object[] args)** | Ghi chuỗi được định dạng ra console và tự động xuống dòng. | Console.WriteLine("Today is {0:D}", DateTime.Now); |
| **Flush()** | Xóa bộ đệm đầu ra của console (thường không cần sử dụng trong ứng dụng cơ bản). | Console.Out.Flush(); |