**Bạc Anh Tuấn**

|  |  |  |
| --- | --- | --- |
| Ngày sinh: 06/02/2004  Trường: ĐH Công Nghệ GTVT  Ngành: Công Nghệ Thông Tin  SĐT: 0965152243  Profile: facebook.com/anhtuanmc.24/ | Địa chỉ: Kiến Hưng, Hà Đông  Email: [anhtuan60204@gmail.com](mailto:anhtuan60204@gmail.com)  Link github báo cáo: https://github.com/AnhTuan2310/Intern-Report  Ngày vào làm: 27/09/2025 |  |

**Thu hoạch học tập ngày 29/10/2025**

# Front End (28/10/2025):

## HTML/CSS

### Các thẻ cơ bản

|  |  |
| --- | --- |
| <article> | Đại diện cho một nội dung độc lập (bài báo, bài đăng blog,...) |
| <aside> | Dành cho nội dung liên quan bên lề (sidebar, quảng cáo) |
| <details> | Tạo ra một tiện ích có thể thu gọn và mở rộng |
| <figcaption> | Dùng để nhóm 1 ảnh, sơ đồ <figure> với chú thích của nó <figcaption> |
| <footer> | Định nghĩa phần chân trang, thư mục (Thường là thông tin liên hệ, bản quyền,...) |
| <header> | Định nghĩa đầu trang hoặc của một mục (Thanh điều hướng, Logo, ...) |
| <main> | Đại diện cho nội dung chính, duy nhất của tài liệu |
| <mark> | Làm nổi bật văn bản trong một đoạn văn |
| <nav> | Dành riêng cho các liên kết điều hướng chính (Menu) |
| <section> | Nhóm các nội dung có liên quan đến nhau (1 Chương, 1 tab,...) |
| <summary> | Tương tự <details> |
| <time> | Định nghĩa ngày tháng, thời gian |

### Các layout cơ bản

#### CSS Flexbox (Flexible box):

Đây là hệ thống layout 1 chiều (hoặc hàng, hoặc cột).

Hoàn hảo cho: Căn chỉnh các mục trong menu (header), sắp xếp các thẻ (card) trong một hàng, căn giữa nội dung theo chiều dọc và chiều ngang.

#### CSS Grid (Lưới):

Đây là hệ thống layout 2 chiều (cả hàng và cột).

Hoàn hảo cho: Bố cục tổng thể của toàn bộ trang web (chia cột sidebar, main content), tạo các thư viện ảnh (gallery) phức tạp.

#### Mô hình hộp (The Box Model):

Một khái niệm cơ bản: Mọi phần tử HTML đều là một cái "hộp".

Bao gồm 4 phần: Content (nội dung), Padding (vùng đệm), Border (viền), và Margin (lề). Hiểu rõ điều này là bắt buộc để căn chỉnh layout.

#### Positioning (Định vị):

Sử dụng thuộc tính position (với các giá trị static, relative, absolute, fixed, sticky).

Giúp bạn đặt một phần tử ở một vị trí chính xác trên trang, hoặc "ghim" nó lại khi cuộn.

#### Float:

Cách cũ để cho văn bản "chảy" xung quanh một hình ảnh.

Ngày nay, không nên dùng float để xây dựng layout chính của trang (dùng Flexbox hoặc Grid).

### Responsive

#### Media Queries (@media):

Đây là công cụ quan trọng nhất. Nó cho phép bạn áp dụng các quy tắc CSS khác nhau dựa trên điều kiện của thiết bị

*(Chẳng hạn: "nếu chiều rộng màn hình nhỏ hơn 768px, hãy cho menu hiển thị thành dạng nút bấm").*

Ví dụ: @media (max-width: 600px) { ... }

#### Viewport Meta Tag:

Một dòng bắt buộc phải có trong thẻ <head> của HTML.

Nó bảo trình duyệt di động: "Hãy hiển thị trang web với chiều rộng bằng chiều rộng của thiết bị và không phóng to".

<meta name="viewport" content="width=device-width, initial- scale=1.0">

#### Fluid Grids (Lưới linh hoạt):

Sử dụng Flexbox/Grid kết hợp với các đơn vị tương đối (như %) thay vì các đơn vị cố định (như px) để chiều rộng các cột tự co giãn theo màn hình.

#### Flexible Units (Đơn vị linh hoạt):

Ưu tiên dùng các đơn vị như %, vw (chiều rộng viewport), vh (chiều cao viewport), rem, em (cho kích thước chữ) thay vì px.

#### Flexible Media (Hình ảnh/Video linh hoạt):

Một kỹ thuật CSS đơn giản để đảm bảo hình ảnh không bao giờ bị "tràn" ra ngoài màn hình:

img, video { max-width: 100%; height: auto; }

#### Mobile-First (Ưu tiên di động):

Đây là một chiến lược thiết kế: Xây dựng layout cho di động trước, sau đó dùng Media Queries để thêm các yếu tố phức tạp hơn cho màn hình lớn hơn.

## JS ES6:

### 1. Biến (Variables):

Trong phiên bản cũ, biến hay dùng var nhưng var hay bị lỗi là tràn ra ngoài block của chính nó, ví dụ khai báo trong for nhưng ngoài for lại gọi lên được

Phiên bản mới hiện nay bổ sung let để giải quyết vấn đề đó, ví dụ let trong for thì ngoài for kh gọi được

const thì phải gán giá trị và kh thể thay đổi giá trị trong quá trình làm trừ đối với Obj và Array thì có thể thay đổi data bên trong được nhưng kh ghi đè cho cấu trúc cũ được

Ví dụ:

const LIST\_DATA = {id = "5"}thì có thể set LIST\_DATA.id = "6"chứ không thể LIST\_DATA = {id = "7"}

### 2. Hàm (Method)

Tham số Mặc định (Default Parameters): Cho phép gán giá trị mặc định cho tham số nếu nó không được truyền vào.

Ví dụ:

function chaoES6(ten = 'Bạn') {

console.log(`Chào ${ten}`);

}

chaoES6(); // "Chào Bạn"

chaoES6("TEN"); // "Chào TEN"

Tham số Còn lại (Rest Parameters): Dùng ... để gom tất cả các tham số còn lại vào một mảng.

Ví dụ:

function tinhTong(...so) {

// 'so' là một mảng, chẳng hạn [1, 2, 3, 4]

return so.reduce((total, num) => total + num, 0);

}

console.log(tinhTong(1, 2, 3, 4)); // 10

### 3. Arrow Func (Hàm mũi tên):

Đây là cú pháp viết hàm ngắn gọn và giải quyết từ khóa this

Ví dụ:

Cũ:

var binhPhuong = function(x){

return x\*x;

}

Mới:

const binhPhuong = (x) =>{

return x\*x;

}

Đặc biệt nếu chỉ có 1 tham số và cần 1 return thì có thể viết:

const binhPhuong = x => x\*x;

Lexical:

Ví dụ:

ES5 (Phải dùng 'self' vì 'this' trong setTimeout là 'window')

function DongHo() {

this.giay = 0;

var self = this;

setInterval(function() {

self.giay++;

console.log(self.giay);

}, 1000);

}

ES6 (Arrow function sử dụng luôn 'this' của DongHoES6)

function DongHoES6() {

this.giay = 0;

setInterval(() => {

this.giay++;

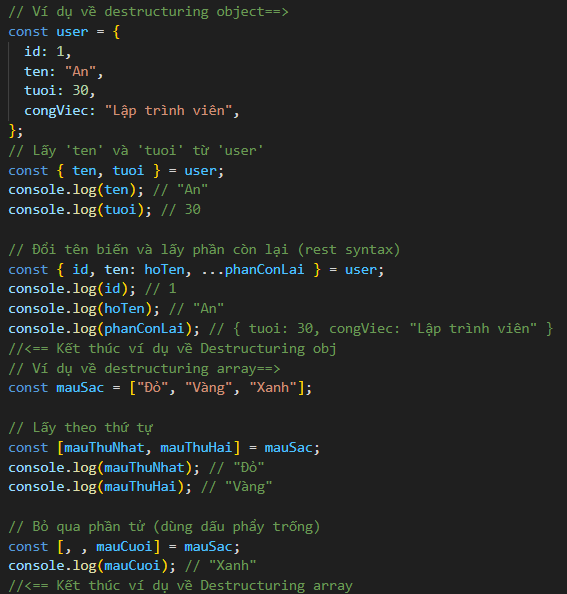
console.log(this.giay);

}, 1000);

}

### 4. Destructuring: Phá vỡ cấu trúc

Đây là cú pháp giúp bóc tách phần tử từ Objs/ Arrays cho phép bạn "bóc tách" các giá trị từ Objects hoặc Arrays vào các biến riêng lẻ. Giúp code ngắn gọn và dễ đọc hơn rất nhiều.



### 5. Module

Module hiểu đơn giản là một file JS độc lập, có thể xuất (export) các biến/hàm/lớp… ra ngoài để file khác nhập (import) vào dùng.

Tác dụng: chia nhỏ code, tái sử dụng, tránh trùng tên, và dễ bảo trì.

### 6. Export và Import:

Export:

Named Export: Dùng khi muốn export nhiều thứ trong cùng file.

Default Export: Dùng khi file chỉ có một giá trị chính cần export

Import

Named Import: Import những Named Export phải sử dụng {}

Default Import: Import từ những Default Export không cần sử dụng {}



# Back End (29/10/2025):

## Giới thiệu .NET

.NET là **nền tảng phát triển phần mềm đa ngôn ngữ** của Microsoft.

Cho phép viết ứng dụng **web, desktop, mobile, cloud, IoT, game**...

Ngôn ngữ phổ biến nhất dùng với .NET là **C#** (ngoài ra còn có F#, VB.NET).

### .NET 8 SDK

|  |  |
| --- | --- |
| **Thành phần có trong .NET 8 SDK** | **Công dụng, mô tả** |
| SDK (Software Development Kit) | Gồm compiler, thư viện, công cụ CLI để **phát triển và build** ứng dụng .NET. |
| Runtime | Là môi trường **chạy ứng dụng .NET** (giống như JVM với Java). Nó chứa CLR (Common Language Runtime) để quản lý bộ nhớ, GC, JIT, exception,... |

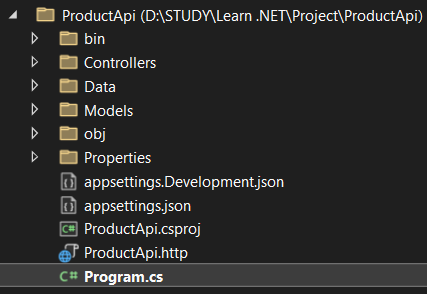
### .NET CLI

Đây là công cụ dòng lệnh để quản lý project .NET.

Một số lệnh cơ bản:

|  |  |
| --- | --- |
| **Lệnh** | **Chức năng** |
| dotnet new console -n HelloWorld | Tạo project console mới |
| dotnet build | Biên dịch Project |
| dotnet run | Chạy ứng dụng |
| dotnet new webapi -n NewApi | Tạo Project WebApi mới |
| dotnet sln add <project> | Thêm Project vào Solution |
| dotnet clean | Dọn file build |

### Cấu trúc Solution/Project



Trong đó:

bin/ → File build ra (binary)

obj/ → File tạm trong lúc build

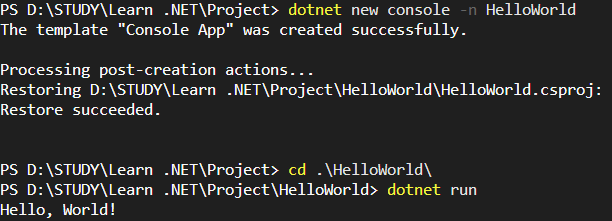
Program.cs → File code chính (entry point)

ProductApi.csproj → File cấu hình project (.NET SDK, version,...)

### Tạo project HelloWorld

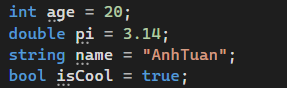
Cú pháp:

* dotnet new console -n HelloWorld : Tạo 1 console project tên HelloWorld
* cd .\HelloWorld\ : Di chuyển đến folder Project mới tạo
* dotnet run : Chạy chương trình trong Project



### C# cơ bản (Bài tập)

#### Biến & Kiểu dữ liệu



int : Kiểu nguyên

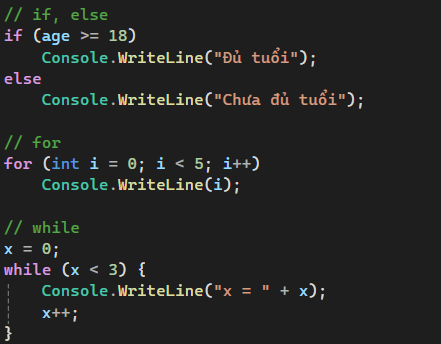
double/float : Số thực

bool : Đúng/sai

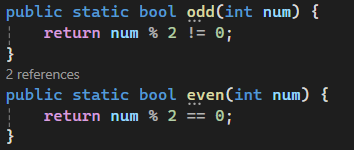
string : Chuỗi

char : 1 ký tự

#### Cấu trúc điều khiển & vòng lặp



#### Hàm

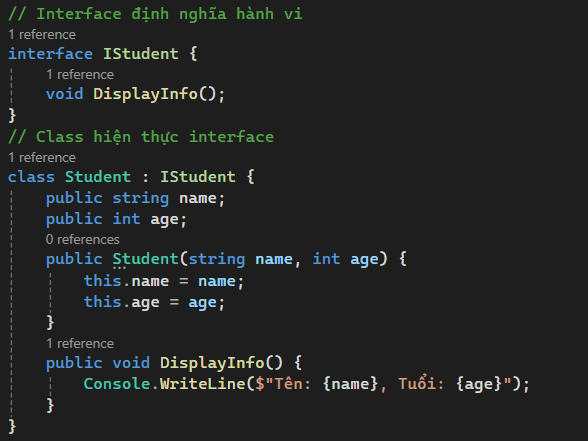


*Hàm kiểm tra điều kiện lẻ, chẵn của 1 tham số truyền vào*

#### Class và Interface

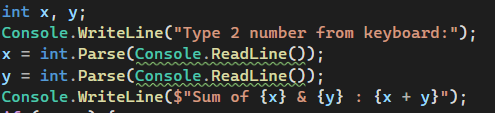
Class và Interface khác gì nhau:

|  |  |  |
| --- | --- | --- |
| **Tiêu chí đánh giá** | **Class** | **Interface** |
| Mục đích | Dùng để **định nghĩa cấu trúc và hành vi** của đối tượng (bao gồm dữ liệu + hành động). | Dùng để **định nghĩa một bộ hành vi** (hợp đồng) mà class khác phải tuân theo. |
| Thành phần chứa được | Có thể chứa **thuộc tính (fields), phương thức (methods), constructor, destructor, property, event**, v.v. | Chỉ chứa **khai báo** phương thức, property, event (từ C# 8 trở đi có thể có default implementation). |
| Triển khai (Implementation) | Class có **code thực thi** bên trong phương thức. | Interface **chỉ mô tả** có cái gì, chứ **không định nghĩa** làm sao. |
| Kế thừa | Một class chỉ **kế thừa duy nhất** từ **một class cha**, nhưng có thể **triển khai nhiều interface**. | Interface có thể **kế thừa nhiều interface khác**. |
| Tạo đối tượng | Có thể tạo object từ class  (new MyClass() ). | Không thể tạo object trực tiếp từ interface. |
| Từ khóa sử dụng | class | interface |
| Ví dụ sử dụng | Cài đặt logic, mô hình hóa thực thể như Student, Car, UserService. | Định nghĩa quy tắc chung cho nhiều class, ví dụ IStudent, IDrawable, IComparable. |



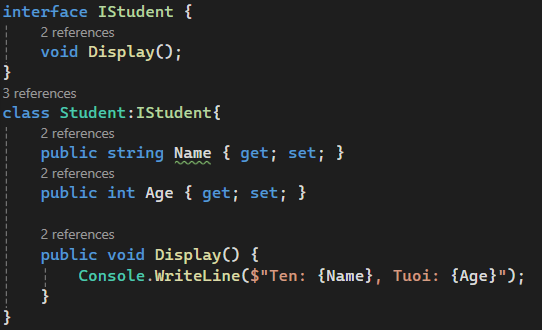
*Ví dụ về Interface và Class*

#### Bài tập tính tổng

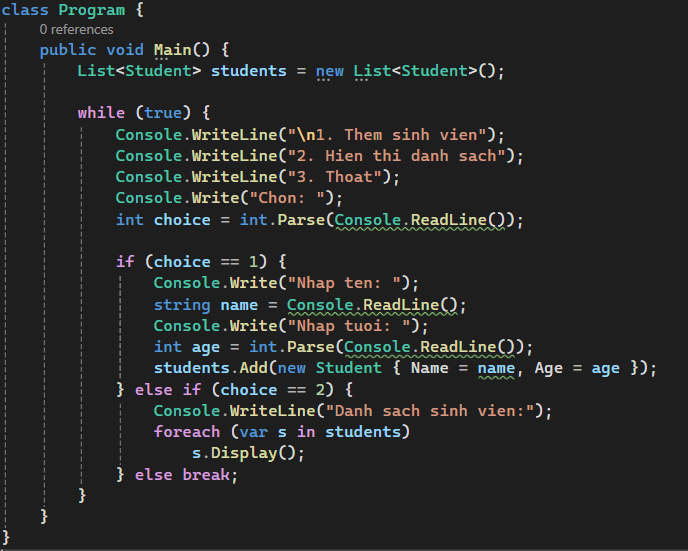


*Khai báo 2 biến thuộc int, nhập liệu từ bàn phím và in ra màn hình kết quả tính tổng*

#### Bài tập quản lý sinh viên



*Khai báo Interface và Class*



*Các thao tác cơ bản*

## OOP – Lập trình hướng đối tượng

### Khái niệm

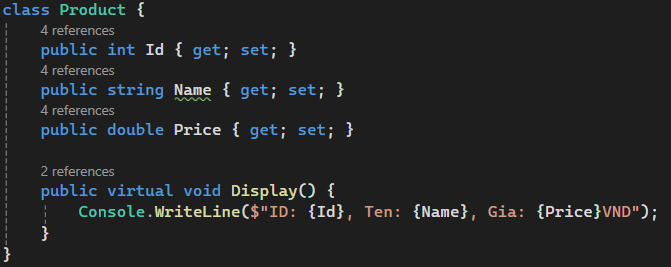
OOP (Object-Oriented Programming) là mô hình lập trình mô phỏng thế giới thực bằng đối tượng (object) – mỗi đối tượng có thuộc tính (property) và hành vi (method).

### 4 tính chất chính của OOP

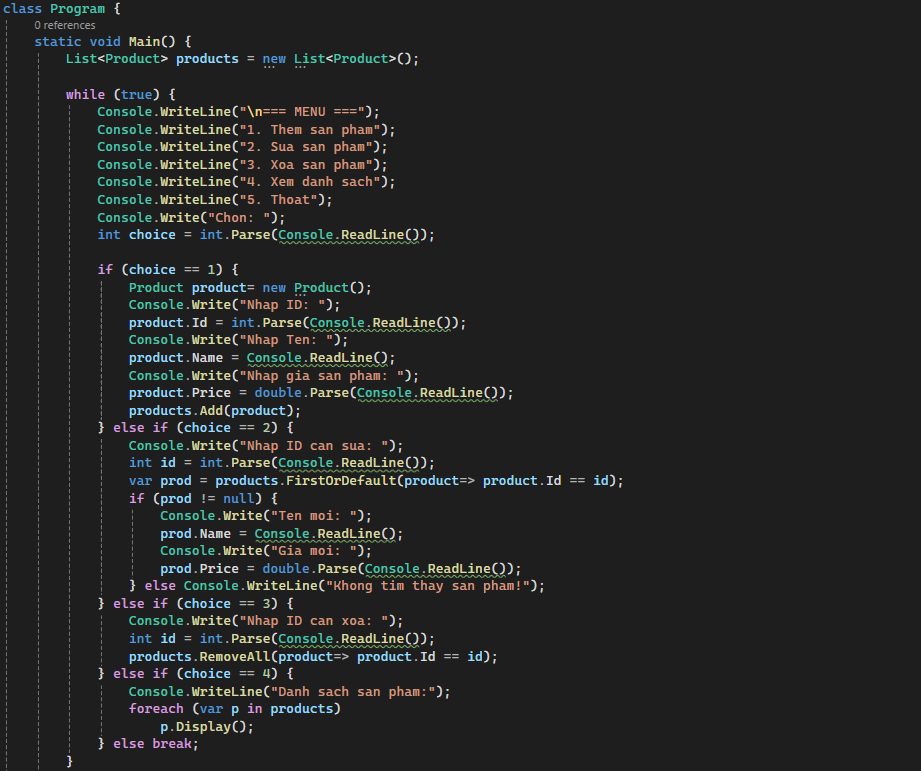
|  |  |  |
| --- | --- | --- |
| Tính chất | Mô tả | Ví dụ |
| Encapsulation (Đóng gói) | Ẩn giấu dữ liệu, chỉ cho phép truy cập thông qua các phương thức được cung cấp. | Dùng private cho biến, public cho hàm getter/setter. |
| Inheritance (Kế thừa) | Cho phép class con **kế thừa** thuộc tính và phương thức của class cha để tái sử dụng code. | class Car : Vehicle |
| Polymorphism (Đa hình) | Một hành động có thể có **nhiều cách thực hiện** khác nhau. | Phương thức Draw() của Circle, Square có cách dùng khác nhau. |
| Abstraction (Trừu tượng) | Chỉ giữ lại **đặc điểm cốt lõi**, ẩn chi tiết cài đặt. Dùng abstract class hoặc interface. | abstract class Animal có Speak(); mỗi loài định nghĩa khác nhau. |

### Bài Tập

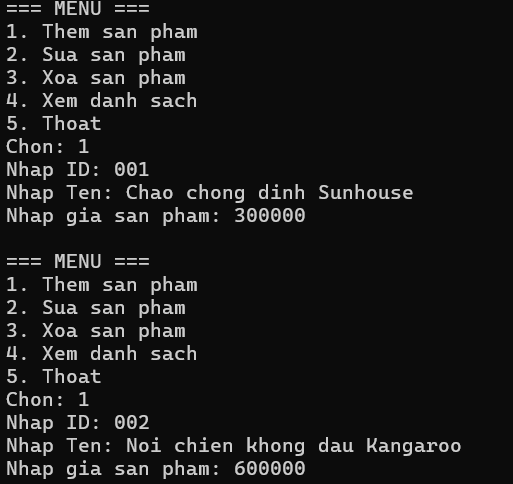
#### Bài tập Quản lý sản phẩm



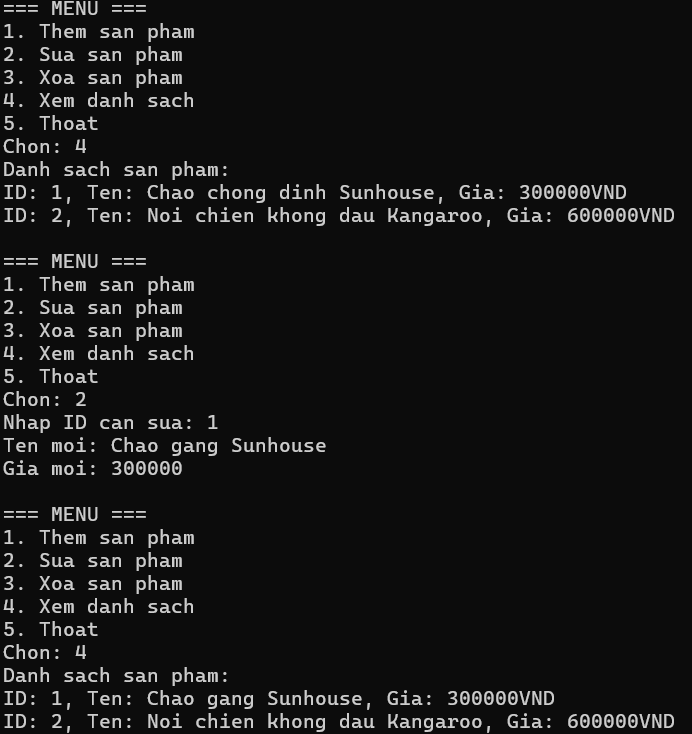
*Khai báo Class Product*



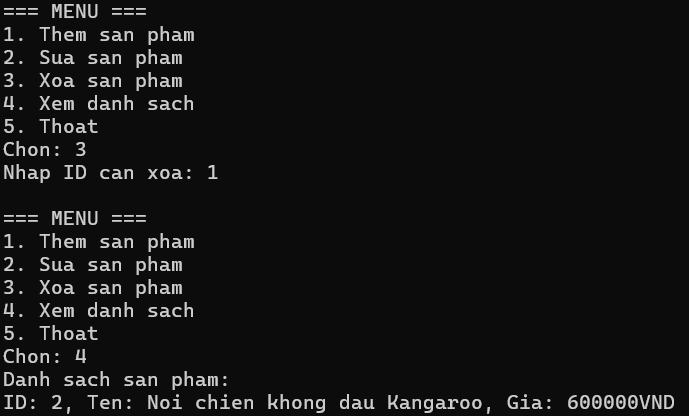
*Tạo menu và các thao tác*

**

*Thao tác Thêm sản phẩm*

**

*Thao tác Hiển thị danh sách, sửa sản phẩm*

**

*Xóa sản phẩm*

## Collection & LINQ

### List

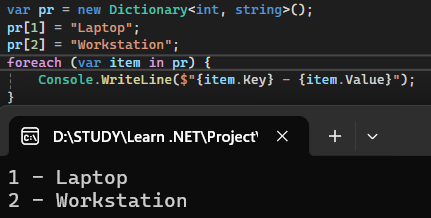
List<T> : Danh sách có thứ tự, dễ thêm/xóa

Ví dụ: List<int> nums = new List<int>();

### Dictionary

Dictionary<TKey, TValue> : Lưu cặp key – value

Ví dụ: Dictionary<int, string> dictionary = new()



### LINQ (Language Integrated Query)

#### Khái niệm

**LINQ** là cú pháp **truy vấn dữ liệu tích hợp vào C#**, giúp ta làm việc với dữ liệu từ nhiều nguồn (List, Array, Database, XML, JSON, …) mà không cần vòng lặp thủ công.

Ví dụ:

var result = from x in list

where x.Price > 1000

select x.Name;

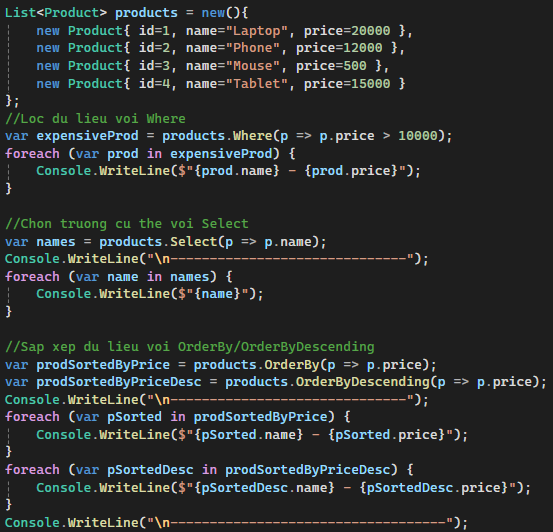
#### Cú pháp

1. Query Syntax: Cú pháp giống SQL

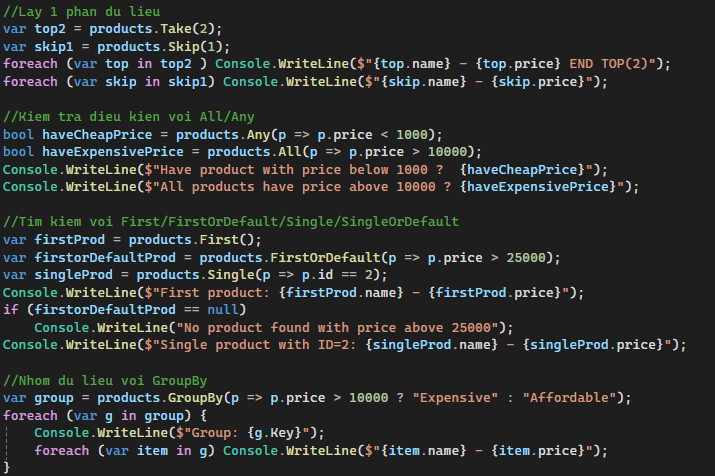
Ví dụ: from … where … select …

1. Method Syntax: Phổ biến hơn, dễ kết hợp hơn

Ví dụ: list.Where(...).Select(...)



*Thao tác với Where, Select, OrderBy/OrderByDescending*

**

*Lấy 1 phần dữ liệu, kiểm tra điều kiện, tìm kiếm, nhóm dữ liệu*

## Exception & Logging

### Exception

#### Khái niệm

Exception trong .NET là các sự kiện lỗi bất thường xảy ra trong quá trình thực thi chương trình và được biểu diễn dưới dạng các đối tượng kế thừa từ lớp cơ sở System.Exception. Khi một ngoại lệ xảy ra, nó có thể làm dừng chương trình ngay lập tức nếu không được xử lý thông qua cơ chế try-catch. Exception giúp chương trình kiểm soát các tình huống lỗi, từ lỗi logic đến lỗi hệ thống, và cung cấp thông tin chi tiết về nguyên nhân và vị trí xảy ra lỗi.

#### Các khái niệm chính

Ngoại lệ (Exception):

Là các đối tượng đại diện cho lỗi trong quá trình chạy, được tạo ra khi có sự cố không mong muốn, ví dụ: chia cho 0, truy cập một file không tồn tại, hoặc lỗi kết nối mạng.

Hệ thống try-catch: Cung cấp cơ chế để xử lý ngoại lệ một cách có chủ động.

try: Khối mã chứa các câu lệnh có khả năng gây ra ngoại lệ.

catch: Khối mã sẽ được thực thi nếu một ngoại lệ tương ứng được ném ra từ khối try.

System.Exception:

Lớp cơ sở cho tất cả các loại ngoại lệ trong .NET, chứa đựng thông tin về lỗi như thông báo lỗi, nguyên nhân, và vị trí xảy ra lỗi.

"Ném" (throw) và "bắt" (catch) ngoại lệ:

Khi một lỗi xảy ra, một đối tượng ngoại lệ sẽ được "ném" (throw).

Nếu không có khối catch nào xử lý ngoại lệ đó, nó sẽ được "truyền" lên ngăn xếp cuộc gọi cho đến khi một khối catch phù hợp được tìm thấy hoặc chương trình kết thúc.

#### Cấu trúc

Try-catch block :  
 try{

// Code có thể gây lỗi

}catch (Exception e){

// Xử lý lỗi tại đây

}finally{

// Code luôn chạy dù có lỗi hay không

}

### Logging

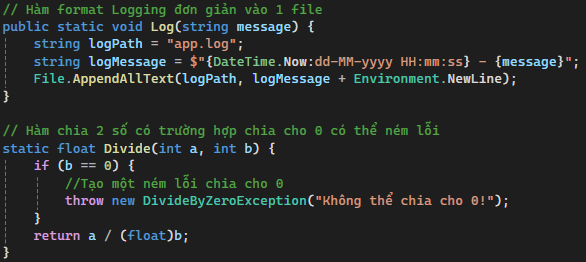
#### Khái niệm

Ở mức đơn giản nhất, logging là quá trình ghi lại các sự kiện xảy ra trong ứng dụng của bạn trong quá trình thực thi. Những sự kiện này có thể là:

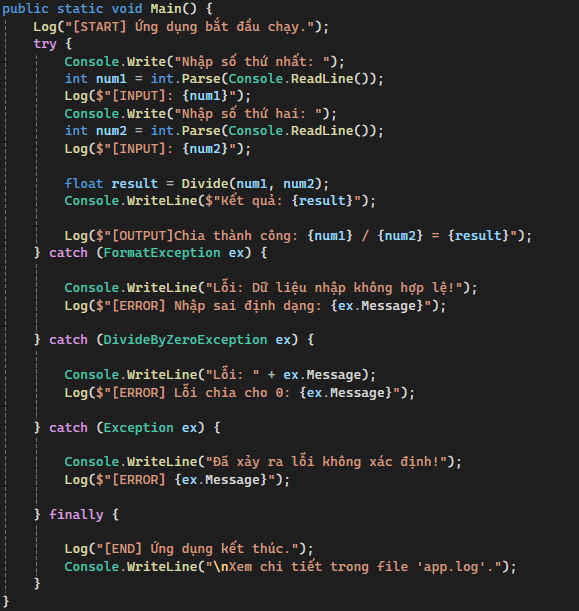
* + - Ứng dụng bắt đầu hoặc kết thúc.
    - Một request HTTP được xử lý.
    - Dữ liệu được đọc hoặc ghi vào cơ sở dữ liệu (như khi làm việc với Entity Framework Core hay các loại database khác).
    - Một tác vụ nền được thực hiện.
    - Một lỗi hoặc ngoại lệ xảy ra.
    - Thay đổi trạng thái quan trọng.

## Ví dụ:

#### Demo ví dụ với trường hợp chia cho 0

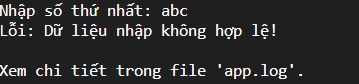


*Thiết lập hàm tạo log và hàm chia kết hợp throw*



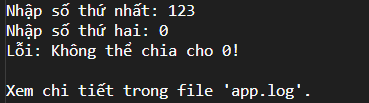
*Lồng Logging vào trong khối try-catch*

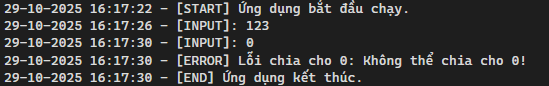
Kiểm tra với dữ liệu thực tế

**

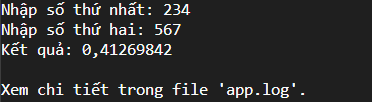
**

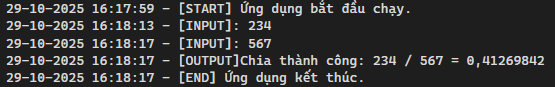
*Trường hợp sai định dạng*

**

**

*Trường hợp chia cho 0*

**

**

*Trường hợp bình thường*