## PHT C# [03] - ĐIỀU KIỆN VÀ VÒNG LẶP

### 🎯 KIẾN THỨC CỐT LỐI

| **Cấu Trúc** | **Cú Pháp** | **Khi Dùng** |
| --- | --- | --- |
| **if** | if (condition) { } | Kiểm tra điều kiện đúng/sai |
| **if-else** | if (c1) { } else { } | Có 2 lựa chọn |
| **if-else if-else** | if (c1) { } else if (c2) { } else { } | Nhiều lựa chọn |
| **switch** | switch(x) { case 1: break; } | Kiểm tra giá trị cụ thể |
| **for** | for(i=0; i<n; i++) { } | Lặp với số lần xác định |
| **while** | while(condition) { } | Lặp cho đến khi điều kiện sai |
| **do-while** | do { } while(condition); | Lặp ít nhất 1 lần |
| **foreach** | foreach(var item in array) { } | Lặp qua mảng |
| **break** | Thoát vòng lặp/switch ngay | Dừng sớm |
| **continue** | Bỏ qua phần còn lại, lần lặp kế tiếp | Bỏ qua những trường hợp nhất định |

### 💡 KHÁM PHÁ KIẾN THỨC

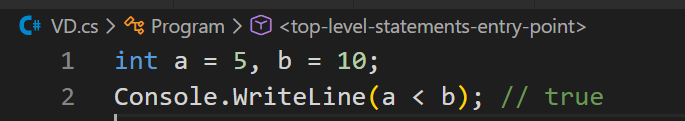
**Câu hỏi tự kiểm tra:**

1. Toán tử so sánh là gì? (==, !=, <, >, <=, >=)

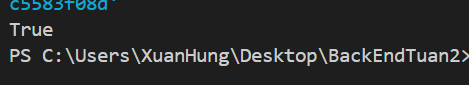
Toán tử so sánh dùng để so sánh hai giá trị và trả về **true** hoặc **false**:

* + == : bằng nhau
  + != : khác nhau
  + < : nhỏ hơn
  + > : lớn hơn
  + <= : nhỏ hơn hoặc bằng
  + >= : lớn hơn hoặc bằng

Ví dụ:



* + Kêt quả:

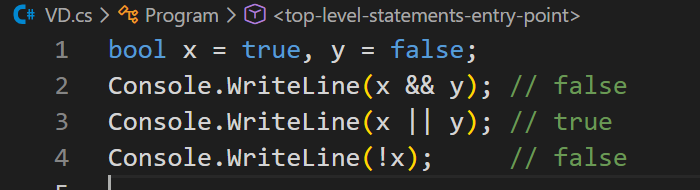


1. Toán tử logic là gì? (&&, ||, !)

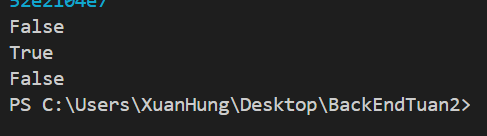
Toán tử logic dùng để kết hợp hoặc đảo giá trị **true/false**:

* + && : AND (đúng khi cả hai điều kiện đều đúng)
  + | : OR (đúng khi ít nhất một điều kiện đúng)
  + ! : NOT (đảo ngược giá trị logic)

Ví dụ:

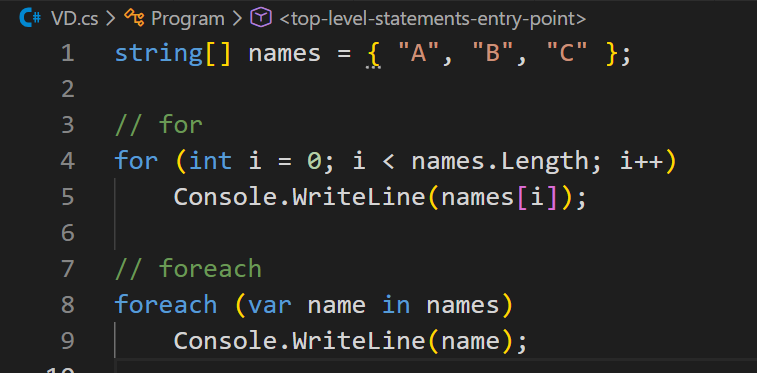


* + Kết quả:



1. Sự khác biệt giữa for và foreach là gì?
   * **for**: Lặp với số lần xác định, thường dùng khi biết chỉ số hoặc cần thao tác theo index.
   * **foreach**: Lặp qua từng phần tử trong một tập hợp (mảng, danh sách), không cần quan tâm đến chỉ số.

Ví dụ:

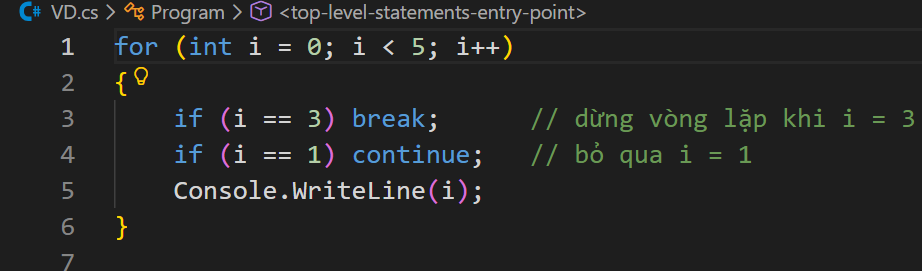


* + Kết quả:

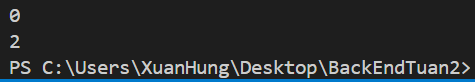


1. Khi nào dùng break? Khi nào dùng continue?
   * **break**: Dùng để thoát khỏi vòng lặp hoặc switch ngay lập tức.
   * **continue**: Dùng để bỏ qua phần còn lại của lần lặp hiện tại và chuyển sang lần lặp tiếp theo.

Ví dụ:

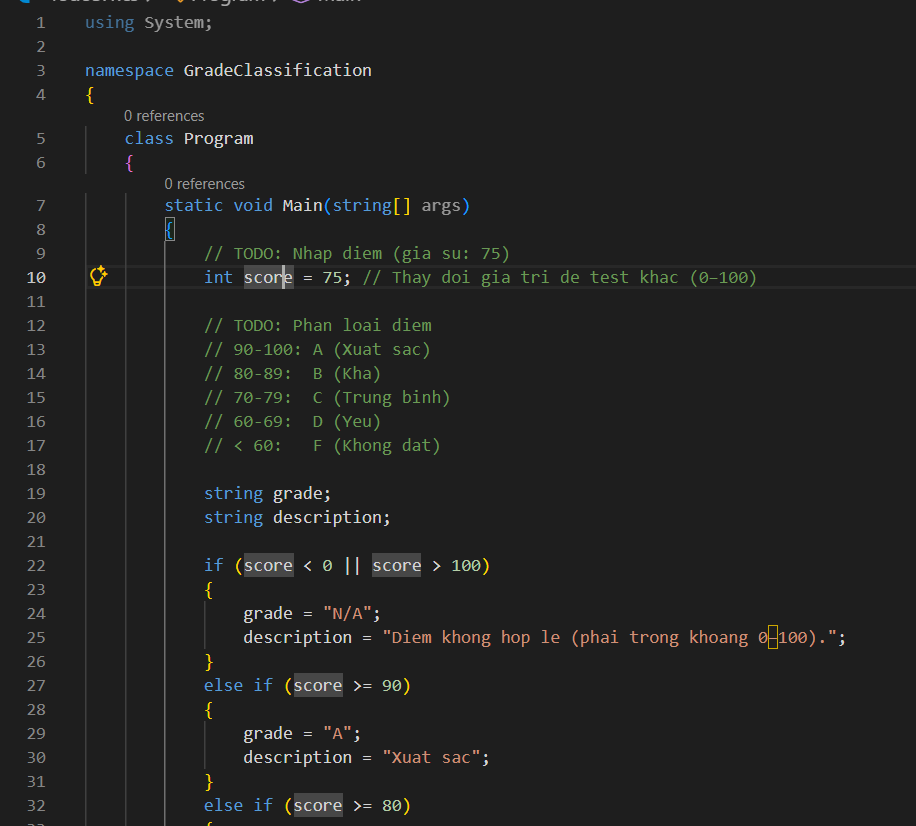


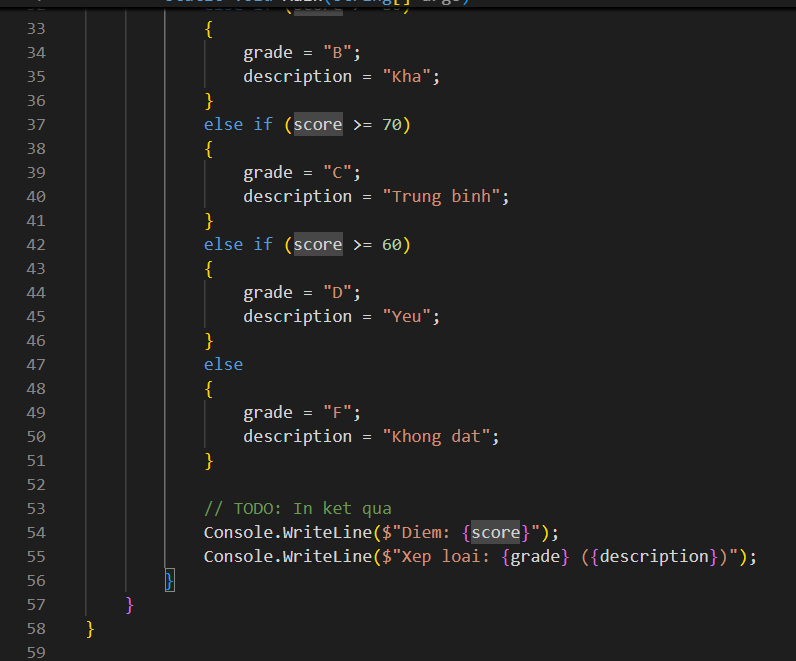
* + Kết quả:



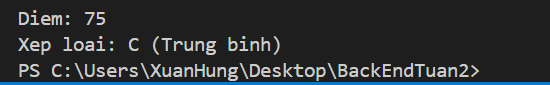
### 📝 VÍ DỤ THỰC HÀNH

**TODO 3.1: [PHÂN LOẠI ĐIỂM - IF/ELSE IF/ELSE]**

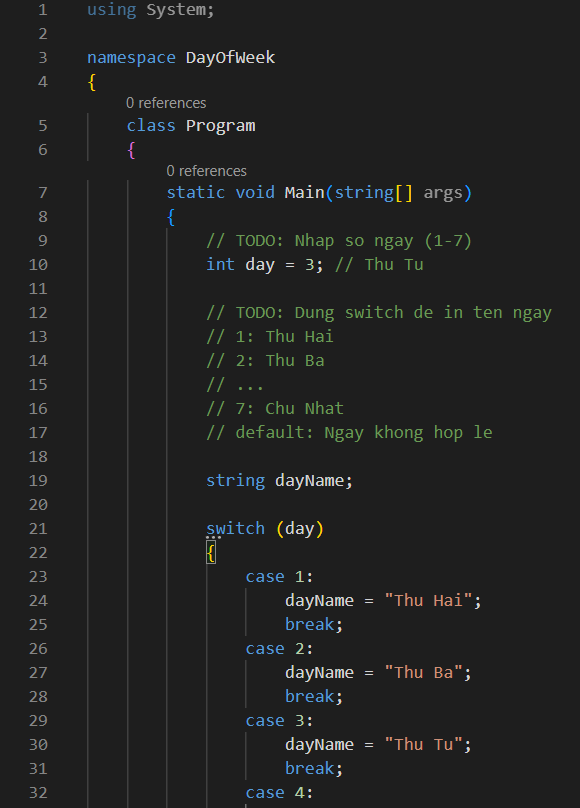


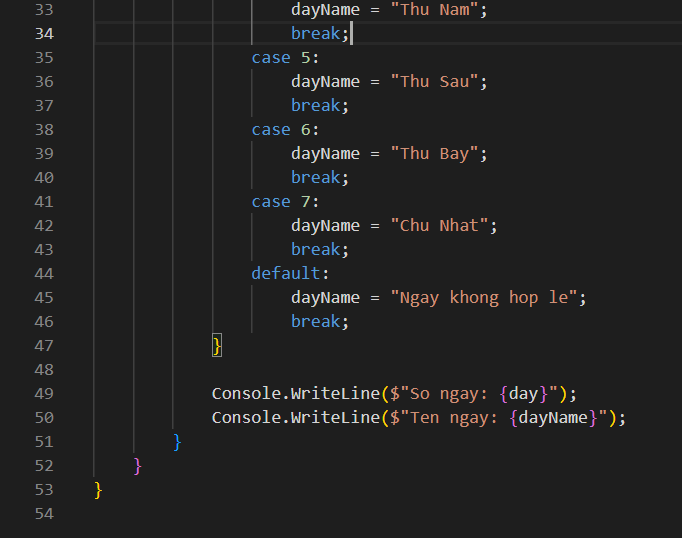


* + Kết quả:

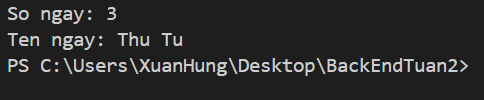


**TODO 3.2: [SWITCH - CHỌN NGÀY TRONG TUẦN]**

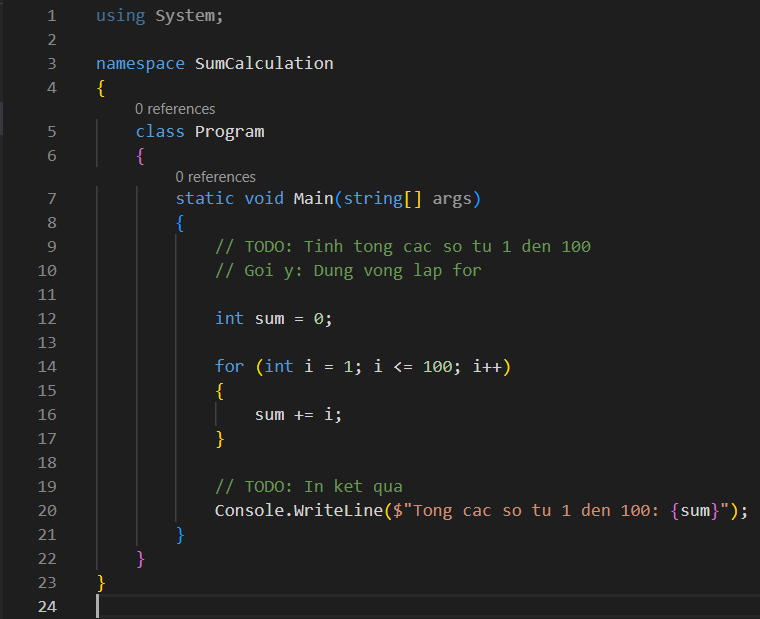




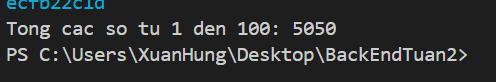
* + Kết quả:



**TODO 3.3: [VÒNG LẶP FOR - TÍNH TỔNG]**

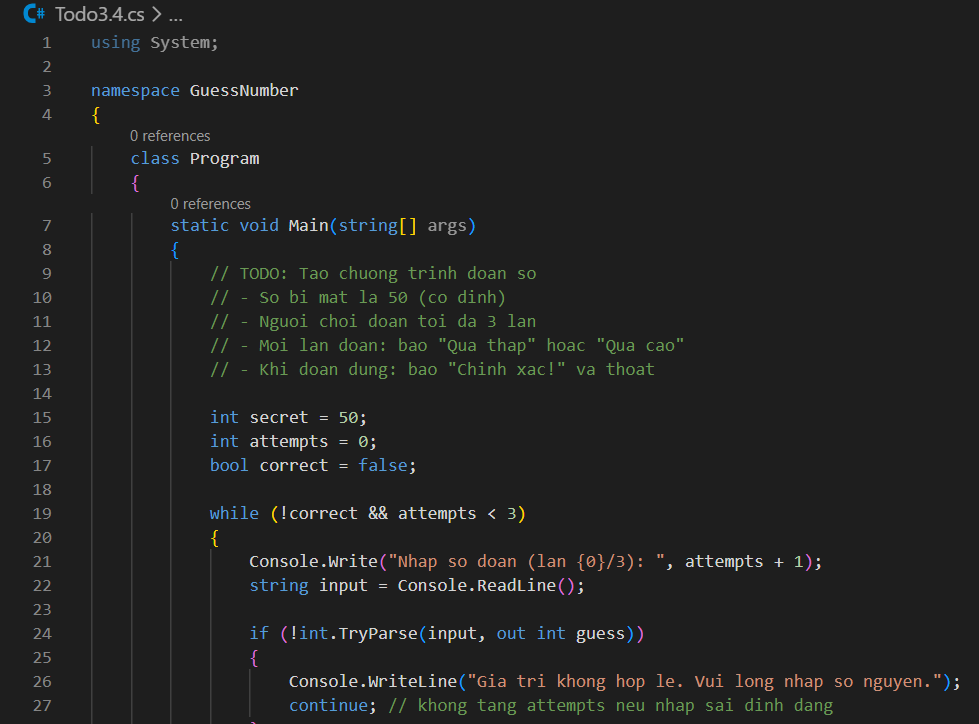


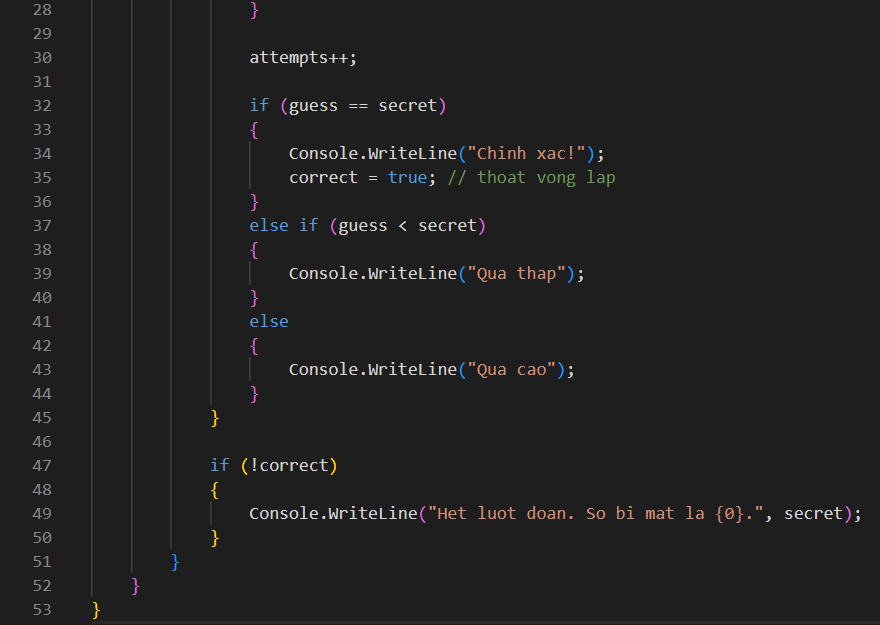
* + Kết quả:



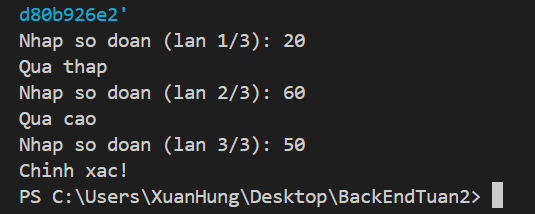
* + Giải thích:
* Khởi tạo sum = 0.
* Dùng vòng lặp for từ i = 1 đến i <= 100.
* Cộng dồn sum += i.
* In kết quả: **Tong cac so tu 1 den 100: 5050**.

**TODO 3.4: [VÒNG LẶP WHILE - ĐOÁN SỐ]**

**-** ****

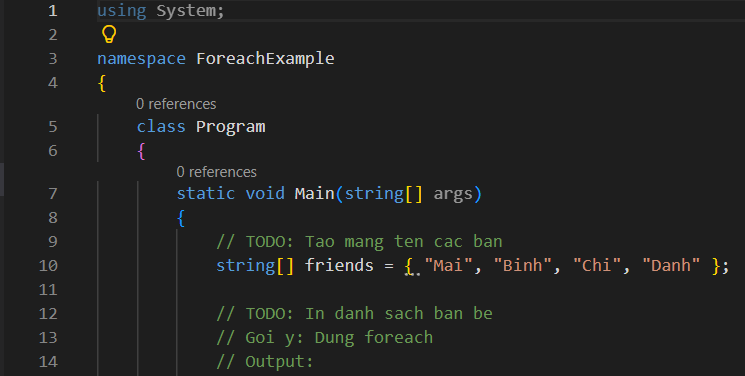


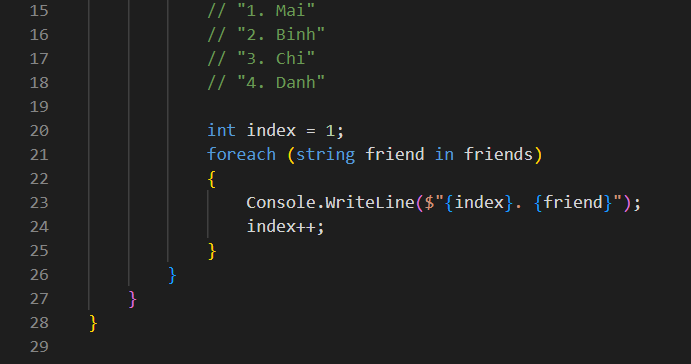
* + Kết quả



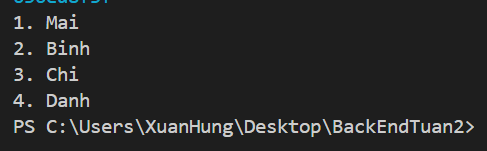
* + Giải thích:
* Vong while tiep tuc khi chua doan dung **va** chua vuot 3 lan.
* Dung int.TryParse de xu ly truong hop nguoi dung nhap khong phai so.
* Khi doan dung → in **"Chinh xac!"** va thoat vong lap.

**TODO 3.5: [FOREACH - IN DANH SÁCH]**



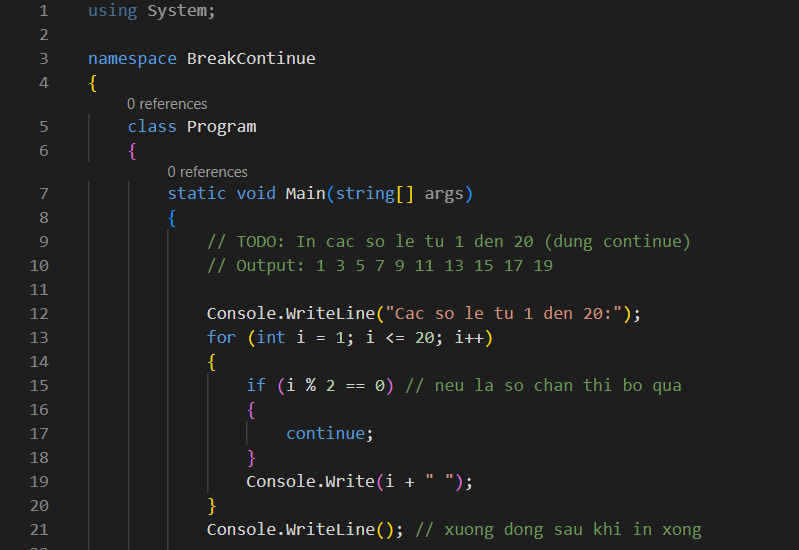


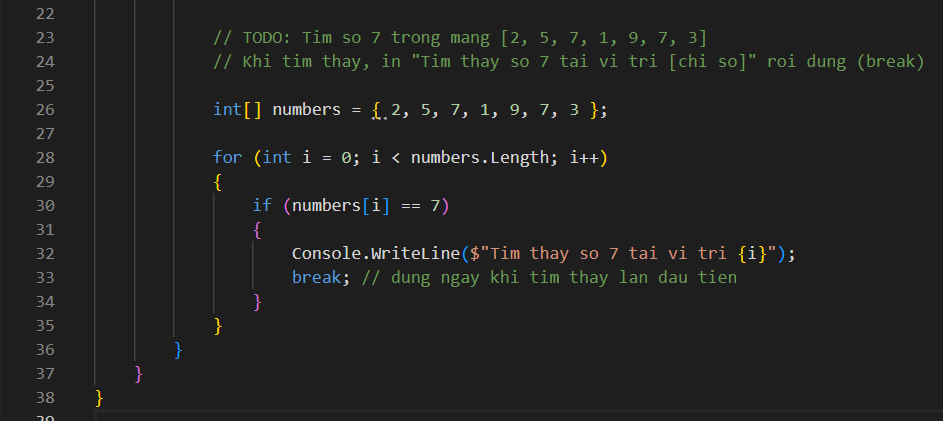
* + Kết quả:



* + Giải thích:
* Dùng foreach để duyệt qua từng phần tử trong mảng friends.
* Biến index bắt đầu từ 1 để đánh số thứ tự.
* In theo định dạng: "{index}. {friend}".

**TODO 3.6: [BREAK & CONTINUE - TÌM SỐ]**





* + Kết quả:



* + Giải thích:
* **continue**: Bo qua phan con lai cua vong lap khi gap so chan, chi in cac so le.
* **break**: Dung vong lap ngay khi tim thay **so 7** lan dau trong mang.

**📸 CHỨNG THỰC [3.1 -> 3.6]:**

* Chạy 6 chương trình và chụp screenshot output
* Thay đổi input để test các trường hợp khác nhau

### 🤔 CÂU HỎI PHẢN BIỆN

1. **Tại sao vòng lặp for lại được dùng nhiều hơn while trong lập trình?**
2. **Nếu bạn có vòng lặp for lồng trong for, làm cách nào để thoát cả hai vòng lặp cùng lúc?**
3. **So sánh while và do-while - khi nào nên dùng do-while?**

Bài làm

**1. Tại sao vòng lặp for lại được dùng nhiều hơn while trong lập trình?**

* **for** thường được dùng khi:
  + Biết trước số lần lặp (ví dụ: từ 1 đến 100).
  + Có biến đếm và điều kiện rõ ràng.
* **Ưu điểm**:
  + Cú pháp gọn: khai báo biến đếm, điều kiện, và bước tăng ngay trong một dòng.
  + Dễ đọc, dễ bảo trì.
* **while** phù hợp hơn khi:
  + Không biết trước số lần lặp (ví dụ: nhập cho đến khi đúng).
  + Điều kiện phụ thuộc vào dữ liệu động.

Vì đa số bài toán có số lần lặp xác định, nên for được dùng nhiều hơn.

**2. Nếu bạn có vòng lặp for lồng trong for, làm cách nào để thoát cả hai vòng lặp cùng lúc?**

Có 3 cách phổ biến:

* **Cách 1: Dùng biến cờ (flag)**  
  Đặt một biến bool để kiểm soát thoát khỏi vòng ngoài.
* **Cách 2: Dùng goto** (ít dùng, không khuyến khích vì khó bảo trì).
* **Cách 3: Dùng break có nhãn (label)** trong C#:

**3. So sánh while và do-while – khi nào nên dùng do-while?**

* **while**:
  + Kiểm tra điều kiện trước khi chạy.
  + Có thể không chạy lần nào nếu điều kiện sai ngay từ đầu.
* **do-while**:
  + Chạy ít nhất **một lần** trước khi kiểm tra điều kiện.
  + Dùng khi cần thực hiện hành động trước rồi mới kiểm tra (ví dụ: nhập dữ liệu, hiển thị menu ít nhất một lần).

### 💫 KẾT NỐI ĐÁNH GIÁ

**Mốc kiểm tra quan trọng:** Bạn phải **thành thạo if/else/switch** và **for/while/foreach** để học phần tiếp theo

## 

## PHT C# [04] - PHƯƠNG THỨC (FUNCTION/METHOD)

### 🎯 KIẾN THỨC CỐT LỐI

| **Khái Niệm** | **Ý Nghĩa** | **Ví Dụ** |
| --- | --- | --- |
| **Phương thức** | Khối code có tên, có thể tái sử dụng | static int Add(int a, int b) |
| **Tham số (Parameters)** | Biến đầu vào của phương thức | Add(int a, int b) → a, b là tham số |
| **Đối số (Arguments)** | Giá trị cụ thể truyền vào | Add(5, 3) → 5, 3 là đối số |
| **Kiểu trả về** | Kiểu dữ liệu phương thức trả về | int, string, void (không trả về) |
| **Return** | Câu lệnh trả về giá trị | return sum; |
| **Static** | Phương thức thuộc lớp, gọi trực tiếp | Program.Add(5, 3) |
| **Nạp chồng (Overloading)** | Nhiều phương thức cùng tên, tham số khác | Add(int, int) và Add(double, double) |
| **Đệ quy (Recursion)** | Phương thức gọi chính nó | Factorial(n) = n \* Factorial(n-1) |

### 💡 KHÁM PHÁ KIẾN THỨC

**Câu hỏi tự kiểm tra:**

1. Tại sao phải dùng phương thức? (Lợi ích của phương thức là gì?)
2. Sự khác biệt giữa tham số và đối số?
3. Phương thức nào phải có return? Phương thức nào không?
4. Nạp chồng phương thức dựa vào điều gì?

Bài làm

1. Tại sao phải dùng phương thức? (Lợi ích của phương thức là gì?)

* Tái sử dụng mã: Viết một lần, dùng nhiều lần → giảm lặp code.
* Dễ bảo trì: Khi cần sửa logic, chỉ sửa trong phương thức.
* Tổ chức chương trình rõ ràng: Chia nhỏ bài toán thành các khối chức năng.
* Tăng tính đọc hiểu: Code dễ hiểu hơn nhờ tên phương thức mô tả chức năng.

2. Sự khác biệt giữa tham số và đối số?

* Tham số (Parameter): Biến được khai báo trong định nghĩa phương thức, đóng vai trò nhận giá trị.
* Ví dụ: int Add(int a, int b) → a, b là tham số.
* Đối số (Argument): Giá trị thực tế truyền vào khi gọi phương thức.
* Ví dụ: Add(5, 3) → 5, 3 là đối số.

3. Phương thức nào phải có return? Phương thức nào không?

* Có return: Khi phương thức có kiểu trả về khác void (ví dụ: int, string).
* Ví dụ: int Add(int a, int b) { return a + b; }
* Không có return: Khi kiểu trả về là void (không trả về giá trị).
* Ví dụ: void PrintMessage() { Console.WriteLine("Hello"); }

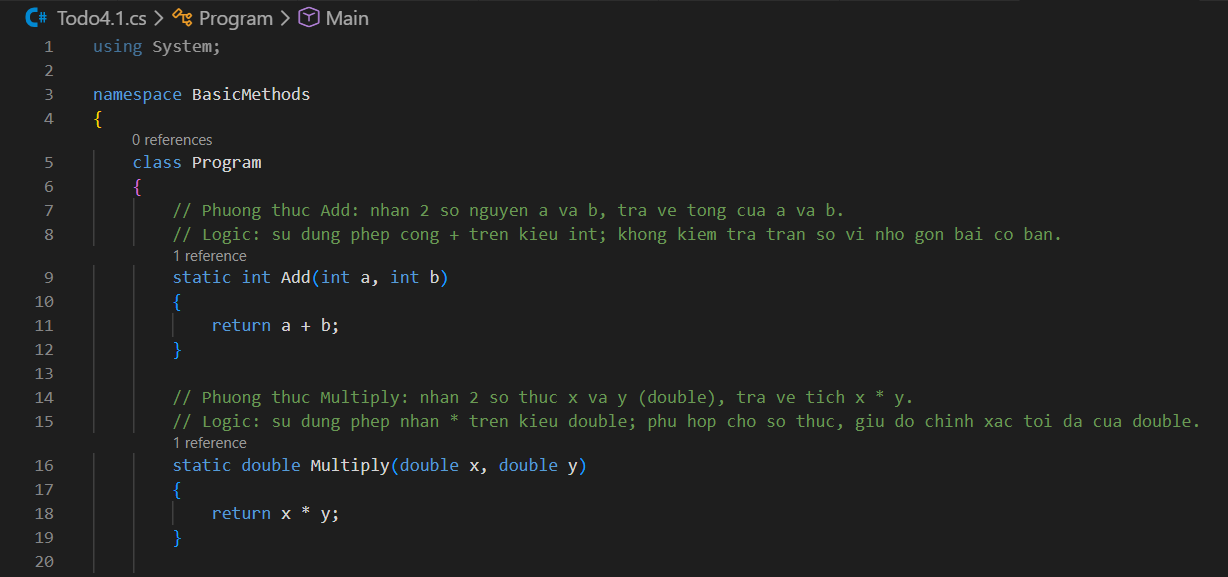
4. Nạp chồng phương thức dựa vào điều gì?

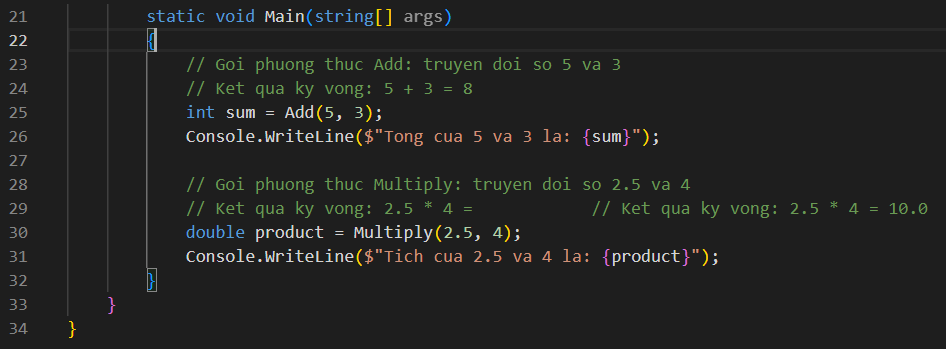
* Tên phương thức giống nhau nhưng khác:
* Số lượng tham số
* Kiểu dữ liệu tham số
* Thứ tự tham số
* Không phụ thuộc vào kiểu trả về (chỉ dựa vào tham số).

### 📝 VÍ DỤ THỰC HÀNH

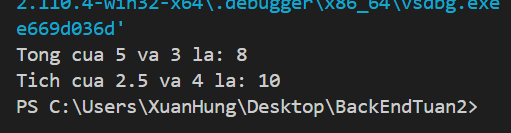
**TODO 4.1: [PHƯƠNG THỨC TÍNH TOÁN CỐ BẢN]**

* Đoạn code:



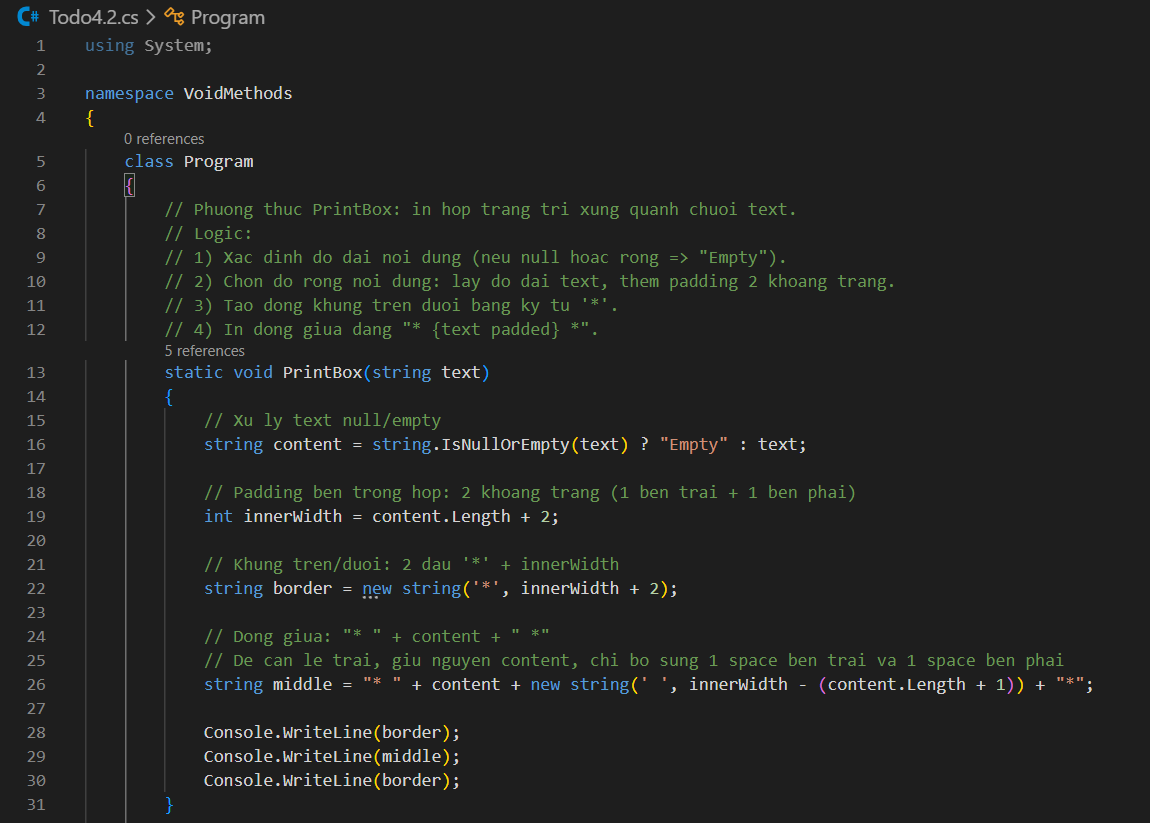


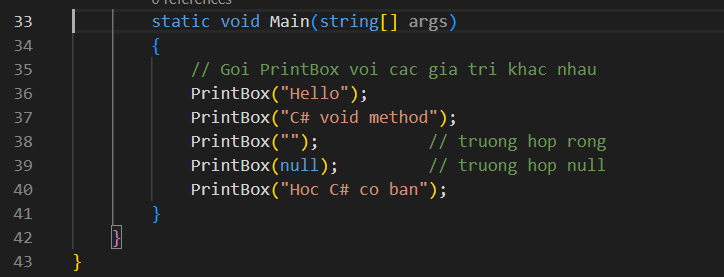
* Kết quả:



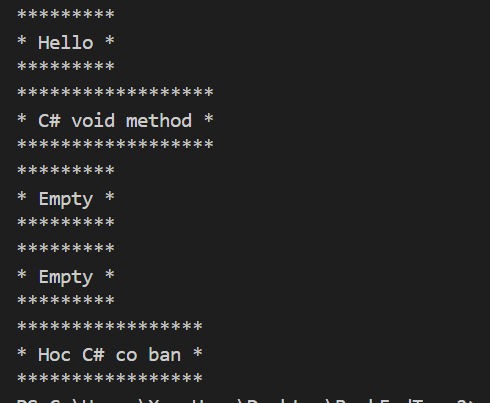
**TODO 4.2: [PHƯƠNG THỨC VOID - KHÔNG TRẢ VỀ]**

* Đoạn code:



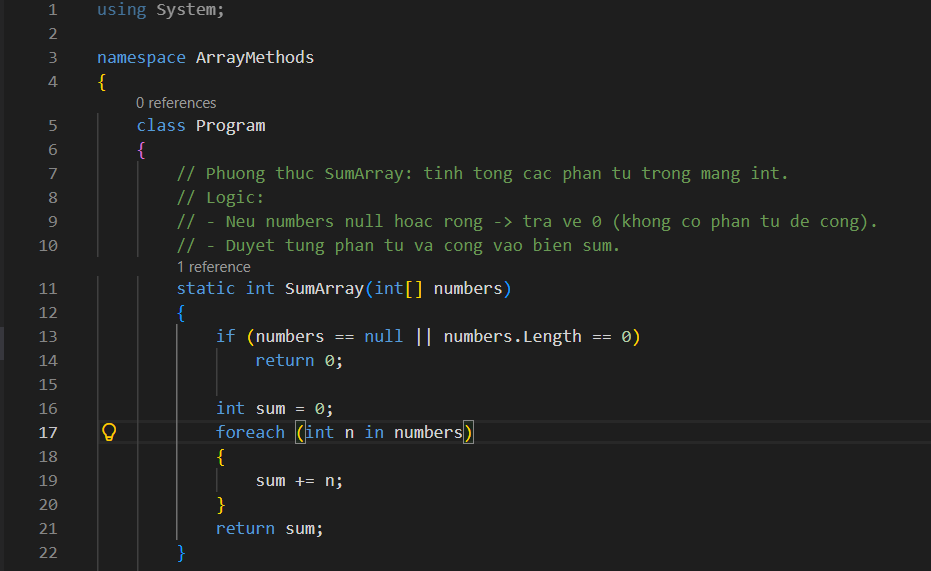


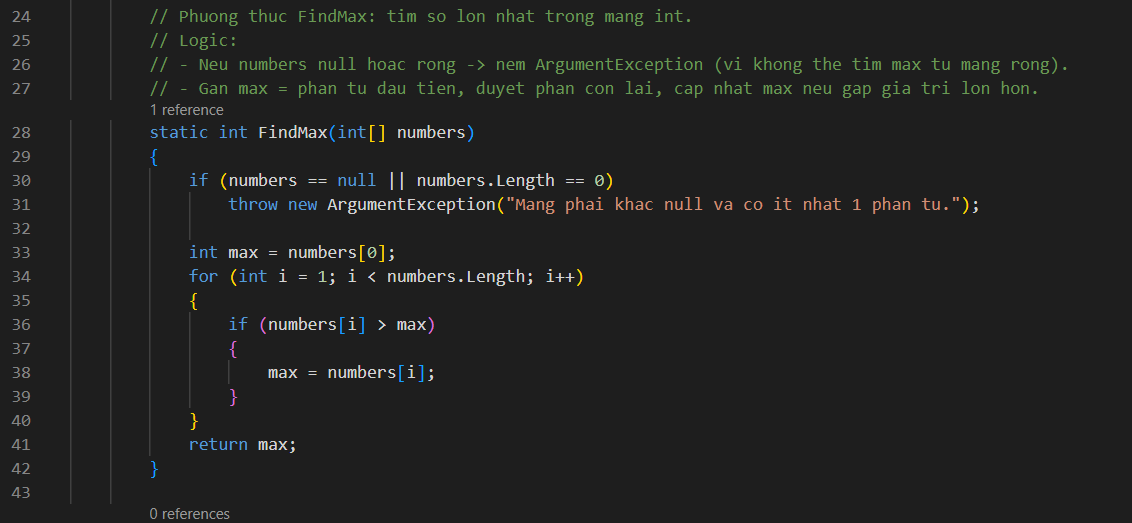
* Kết quả:

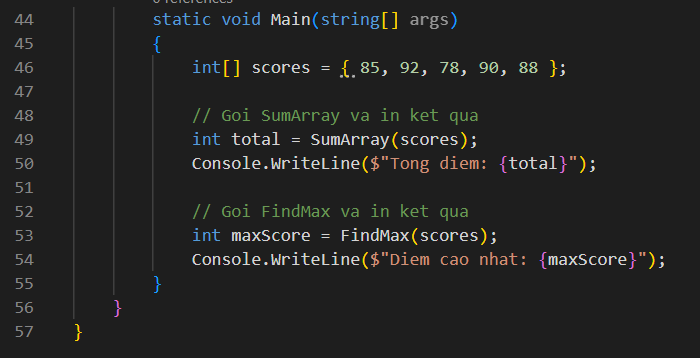


**TODO 4.3: [PHƯƠNG THỨC VỚI MẢNG]**

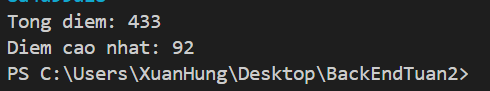
* Đoạn code:





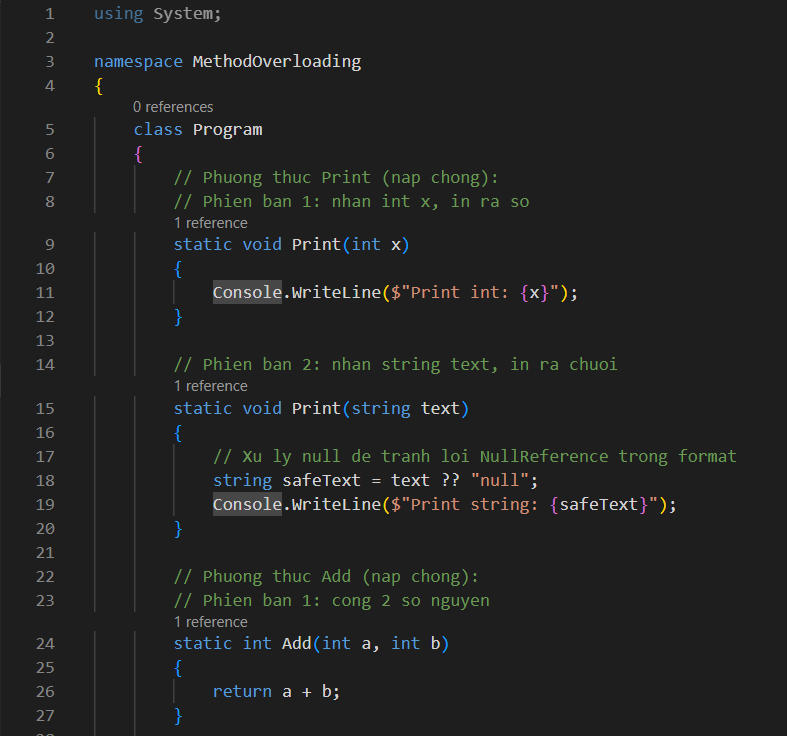


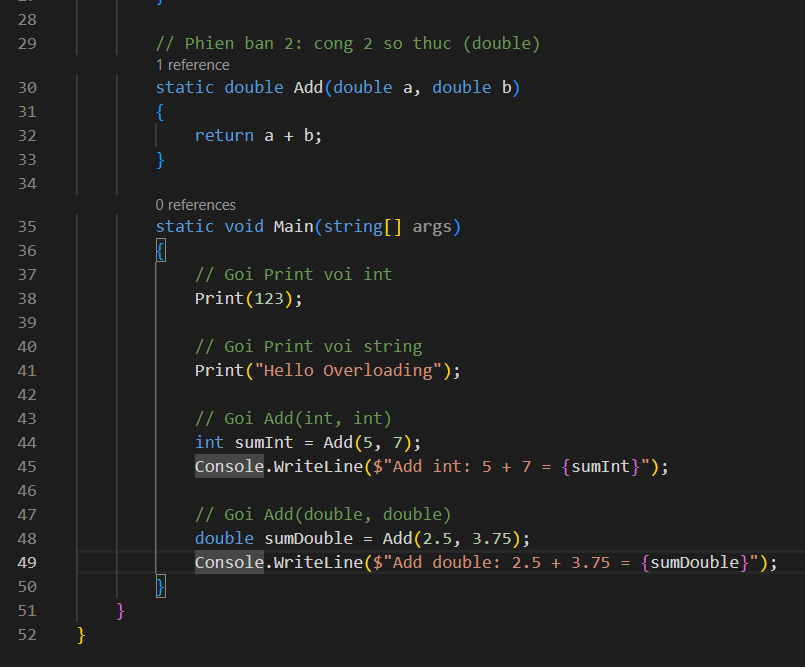
* Kết quả:



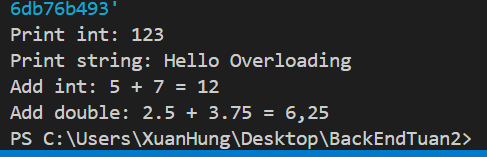
**TODO 4.4: [NẠP CHỒNG PHƯƠNG THỨC]**

* Đoạn code



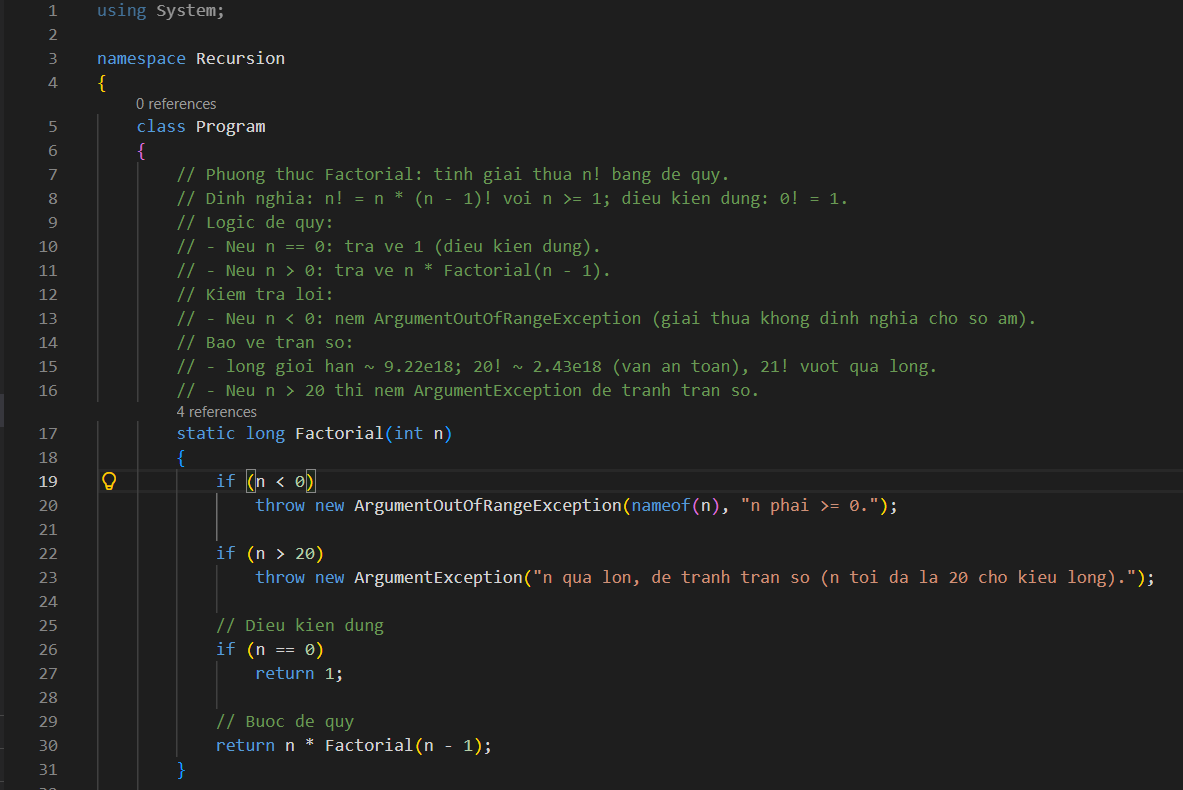


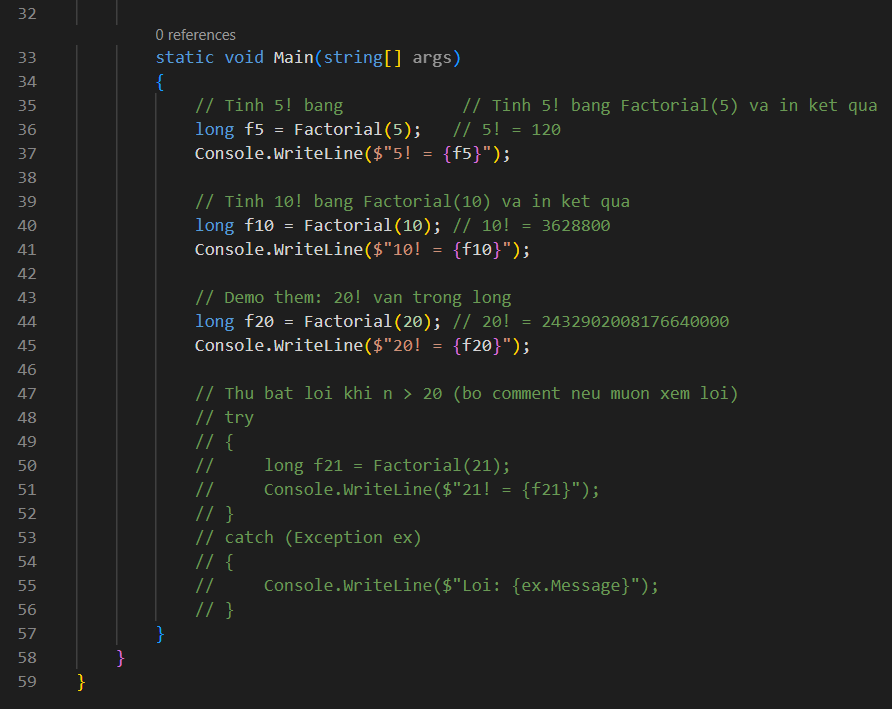
* Kết quả:



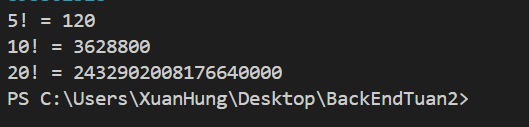
**TODO 4.5: [ĐỆ QUY - TÍNH GIAI THỪA]**

* Đoạn code:





* Kết quả:



**📸 CHỨNG THỰC [4.1 -> 4.5]:**

* Chạy từng chương trình và chụp output
* Viết comment giải thích logic phương thức

### 🤔 CÂU HỎI PHẢN BIỆN

1. **Tại sao cần nạp chồng phương thức? Không thể dùng tên khác nhau cho từng trường hợp không?**
2. **Đệ quy có ưu và nhược điểm gì?**
   * Khi nào nên dùng đệ quy?
   * Khi nào nên dùng vòng lặp?
3. **Phương thức nên ngắn hay dài? Tại sao?**

Bài làm

**1. Tại sao cần nạp chồng phương thức? Không thể dùng tên khác nhau cho từng trường hợp không?**

Có thể dùng tên khác nhau, nhưng:

* Nạp chồng (Overloading) giúp tăng tính nhất quán: cùng một hành động (ví dụ: Add) nhưng áp dụng cho nhiều kiểu dữ liệu.
* Dễ đọc và dễ nhớ: người dùng chỉ cần nhớ một tên phương thức cho cùng một chức năng.
* Giảm trùng lặp logic: nếu tên khác nhau (AddInt, AddDouble), code sẽ kém linh hoạt và khó bảo trì.

Ví dụ:

Add(5, 7);        // int

→ Cùng tên Add, nhưng xử lý kiểu dữ liệu khác nhau.

**2. Đệ quy có ưu và nhược điểm gì?**

Ưu điểm:

* Code ngắn gọn, dễ hiểu với bài toán có cấu trúc lặp tự nhiên (như cây, giai thừa, Fibonacci).
* Trực quan: biểu diễn bài toán theo định nghĩa toán học.

Nhược điểm:

* Hiệu năng thấp hơn vòng lặp: mỗi lần gọi đệ quy tạo một stack frame → tốn bộ nhớ.
* Nguy cơ tràn stack nếu không có điều kiện dừng hoặc n quá lớn.
* Khó debug hơn vòng lặp.

Khi nào nên dùng đệ quy?

* Bài toán có cấu trúc phân rã tự nhiên:
  + Duyệt cây, duyệt đồ thị.
  + Tính giai thừa, Fibonacci.
  + Thuật toán chia để trị (QuickSort, MergeSort).

Khi nào nên dùng vòng lặp?

* Khi cần hiệu năng cao, n lớn.
* Khi bài toán không yêu cầu phân rã tự nhiên.
* Khi muốn tránh tràn stack.

**3. Phương thức nên ngắn hay dài? Tại sao?**

* Nên ngắn (thường < 20 dòng):
  + Dễ đọc, dễ hiểu.
  + Dễ bảo trì và kiểm thử.
  + Giúp tái sử dụng logic.
* Nếu phương thức quá dài → nên chia nhỏ thành nhiều phương thức theo nguyên tắc Single Responsibility (mỗi phương thức chỉ làm một việc).

### 💫 KẾT NỐI ĐÁNH GIÁ

**Chuẩn bị tiếp theo:** Phiếu 05 sẽ dạy **lập trình hướng đối tượng (OOP)** - Đây sẽ là bước nhảy vọt lớn

## 

## PHT C# [05] - LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG (OOP)

### 🎯 KIẾN THỨC CỐT LỐI

| **Khái Niệm** | **Ý Nghĩa** | **Ví Dụ** |
| --- | --- | --- |
| **Lớp (Class)** | Bản thiết kế cho đối tượng | class Student { } |
| **Đối tượng (Object)** | Thể hiện cụ thể của lớp | Student sv1 = new Student(); |
| **Thuộc tính (Property)** | Dữ liệu của đối tượng | sv1.Name, sv1.Age |
| **Phương thức (Method)** | Hành động của đối tượng | sv1.Display(), sv1.Calculate() |
| **Constructor** | Phương thức khởi tạo đối tượng | Student(string name) { ... } |
| **Encapsulation** | Che giấu chi tiết, chỉ mở công khai cần thiết | private, public, protected |
| **Kế thừa (Inheritance)** | Lớp con thừa từ lớp cha | class Dog : Animal { } |
| **Đa hình (Polymorphism)** | Một hành động, nhiều cách thực thi | virtual và override |
| **Giao diện (Interface)** | Hợp đồng quy định phương thức bắt buộc | interface IMovable { } |

### 💡 KHÁM PHÁ KIẾN THỨC

**Câu hỏi tự kiểm tra:**

1. Sự khác biệt giữa lớp và đối tượng?
2. Constructor là gì? Tại sao cần constructor?
3. private, public, protected khác gì?
4. Kế thừa có tác dụng gì?

Bài làm

* + - 1. Sự khác biệt giữa lớp và đối tượng?
  + Class: là bản thiết kế (blueprint) cho đối tượng, định nghĩa đối tượng (property) và phương thức (method) mà đối tượng sẽ có.
  + VD:

Class student {

Public string name;

Public int Age;

Public void Display() {…}

}

* + Object: là thể hiện cụ thể tự một lớp, biểu hiện qua class.
  + VD:

Student sv1 = new Student();

Sv1.Name = “Hung”;

Sv1.Age = 19;

* + Điểm khác biệt chính:
* Class là khuôn mẫu, không chiếm bộ nhớ dữ liệu.
* Object là thực thể cụ thể, chiếm bộ nhớ và có giá trị cho các thuộc tính.

1. Constructor là gì? Tại sao cần constructor?
   * Constructor: là phương thức đặc biệt trong lớp, được gọi khi tạo đối tương, dùng để khởi tạo giá trị ban đầu cho thuộc tính.
   * VD:

Class student {

Public string Name;

Public student (string name) {

Name = name;

}

}

* + Cần sử dụng Constructer vì:
* Đảm bảo đối tượng được được khởi tạo với giá trị hợp lệ ngay từ đầu.
* Giúp code gọn hơn, tránh phải gán giá trị thủ công sau khi tạo đối tượng.

1. private, public, protected khác gì?
   * Publica: thành phần có thể truy cập từ bất cứ đâu (trong và ngoài lớp).
   * Private: chỉ truy cập được trong lớp chính.
   * Protected: truy cập được trong lớp đó và các lớp kế thừa.
   * VD:

Class student {

Public string name; //Ai cũng truy cập được

Private int age; //Chỉ trong student

Proteced string address; //Student và lớp con

1. Kế thừa có tác dụng gì?
   * Cho phép lớp con sử dụng thuộc tính và phương pháp của lớp cha, giúp:

* Tái sử dụng mã code(code reuse).
* Mở rộng chức năng mà không viết lại từ đầu.
* Hỗ trợ đa hình (polymorphism).
  + VD:

Class Animal {

Public void Eat () {…}

}

Class Dog {

Public void Bark () {…}

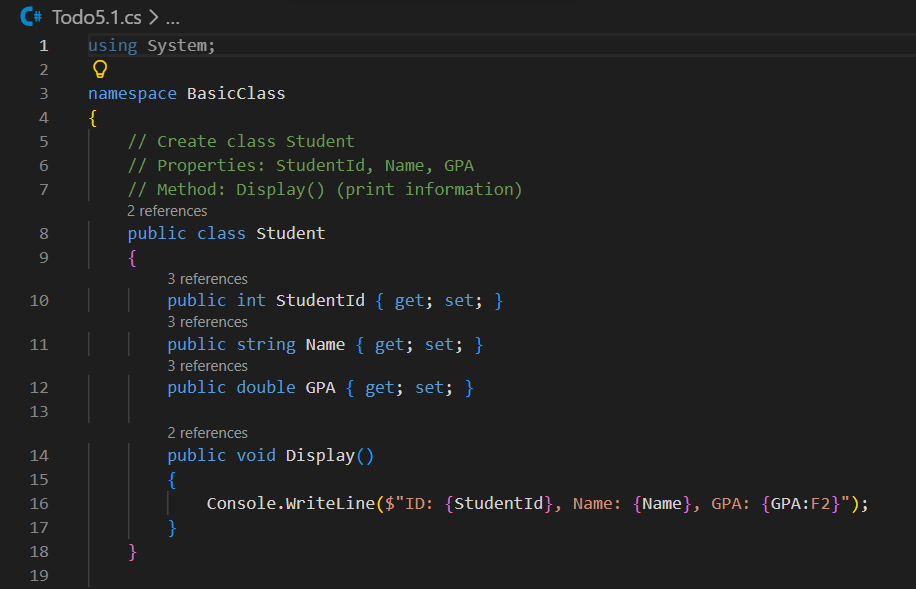
}

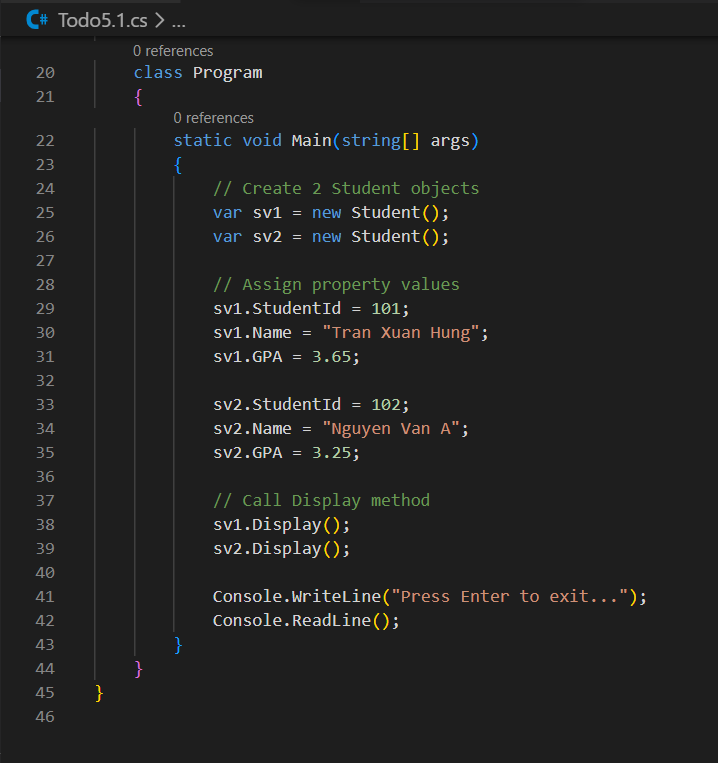
* + Dog kế thừa từ Animal nên có thể gọi Eat().

### 📝 VÍ DỤ THỰC HÀNH

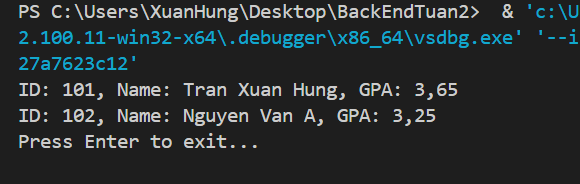
**TODO 5.1: [TẠO LỚP CƠ BẢN - STUDENT]**

* + **Mã Code:**





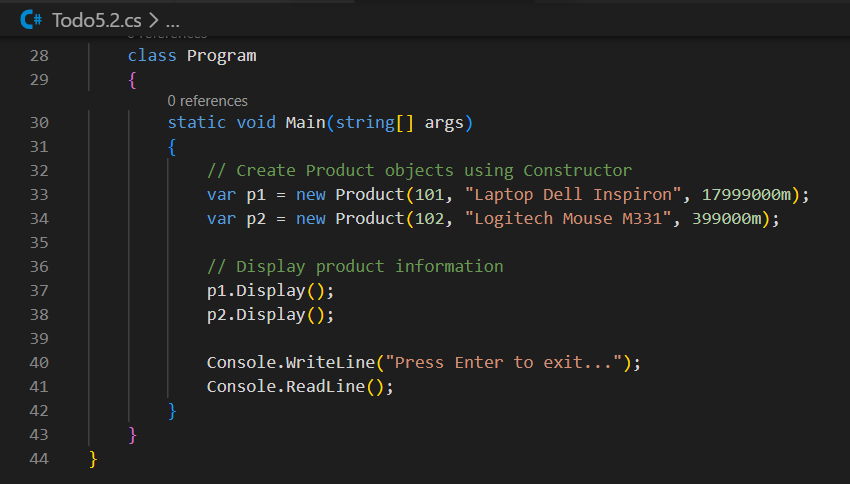
* + **Kết quả:**



**TODO 5.2: [CONSTRUCTOR - KHỞI TẠO DỮ LIỆU]**

* + **Mã code:**

****

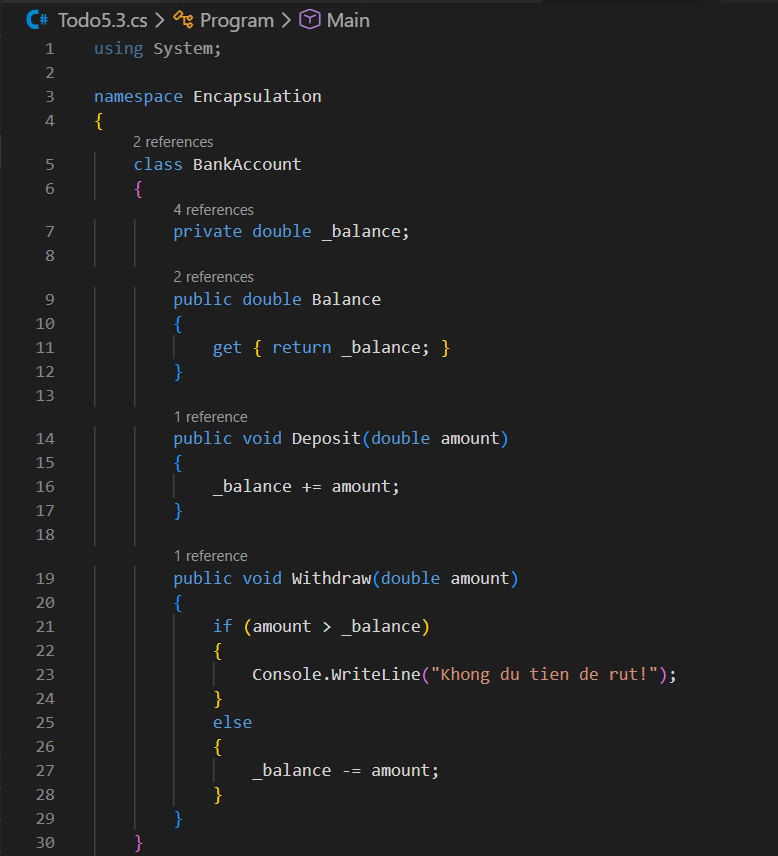
****

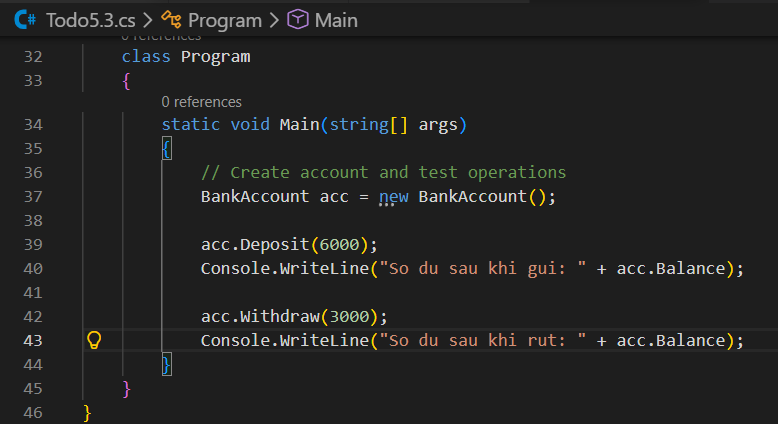
* + **Kết quả:**

****

**TODO 5.3: [ENCAPSULATION - CHE GIẤU DỮ LIỆU]**

* + **Mã code**



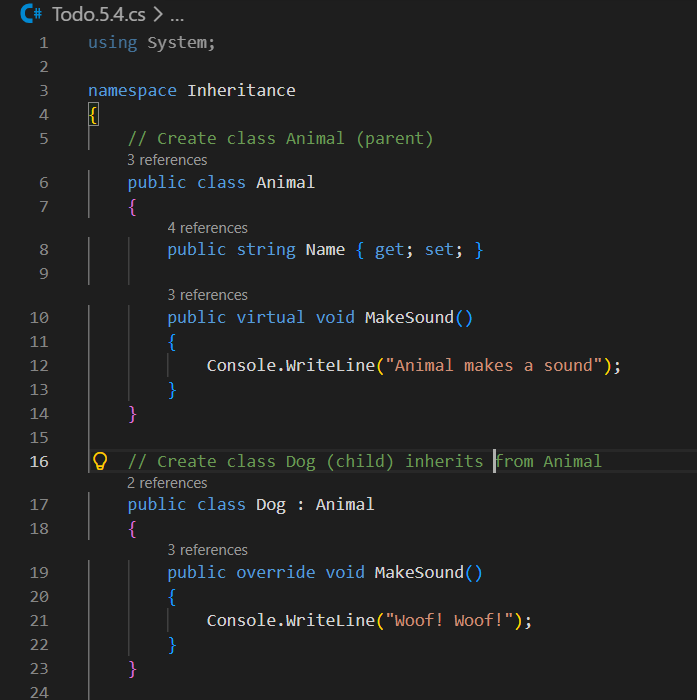


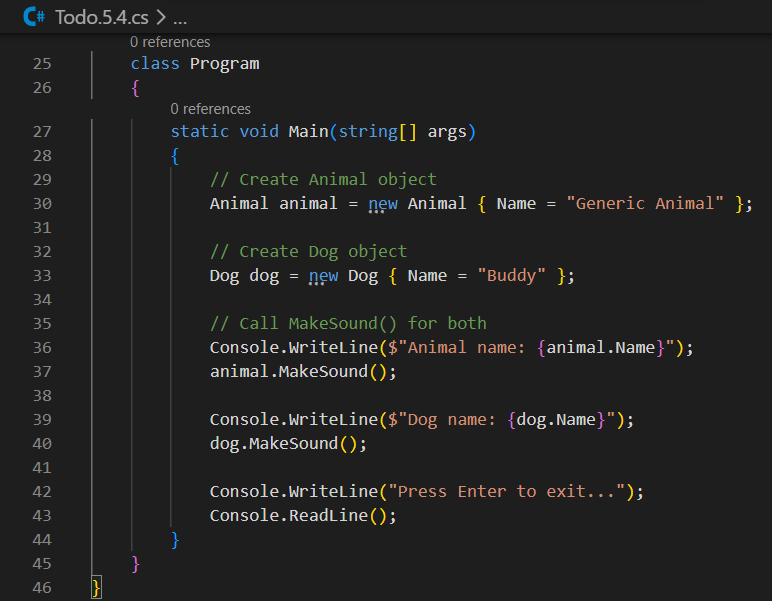
* + **Kết quả:**



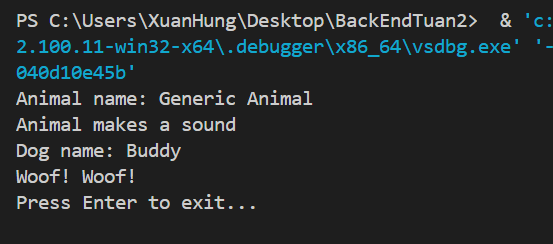
**TODO 5.4: [KẾ THỪA - ANIMAL & DOG]**

* + **Mã code:**

****

****

* + **Kết quả:**

****

**📸 CHỨNG THỰC [5.1 -> 5.4]:**

* Chạy từng chương trình, screenshot output
* Thay đổi dữ liệu để test chức năng

### 🤔 CÂU HỎI PHẢN BIỆN

1. **Tại sao encapsulation (che giấu dữ liệu) lại quan trọng?**
   * Bảo vệ dữ liệu: Ngăn việc truy cập và thay đổi trực tiếp từ bên ngoài, tránh lỗi hoặc hành vi không mong muốn.
   * Kiểm soát logic: Cho phép kiểm tra và xác thực dữ liệu thông qua getter/setter hoặc phương thức, ví dụ: không cho số dư âm.
   * Giảm phụ thuộc: Giúp thay đổi nội bộ lớp mà không ảnh hưởng đến mã bên ngoài (tăng tính bảo trì).
   * Tăng tính bảo mật: Dữ liệu nhạy cảm (như mật khẩu, số dư) không bị lộ hoặc chỉnh sửa tùy tiện.
2. **Kế thừa vs. Composition - khi nào dùng cái nào?**
   * Kế thừa (Inheritance):
     + Dùng khi có quan hệ “is-a” (là một loại của).
     + Ví dụ: Dog is-a Animal.
     + Cho phép tái sử dụng code từ lớp cha, hỗ trợ đa hình.
     + Nhược điểm: Ràng buộc chặt, dễ gây phụ thuộc và khó thay đổi nếu cấu trúc phức tạp.
   * Composition:
     + Dùng khi có quan hệ “has-a” (có một).
     + Ví dụ: Car has-a Engine.
     + Linh hoạt hơn, dễ thay đổi thành phần mà không ảnh hưởng toàn bộ hệ thống.
     + Ưu điểm: Giảm coupling, dễ bảo trì, dễ test.

Nguyên tắc:

* + Ưu tiên composition hơn kế thừa (theo SOLID: “Favor composition over inheritance”), trừ khi quan hệ thực sự là “is-a”

1. **Lợi ích của polymorphism (đa hình) là gì?**
   * Đơn giản hóa code: Cho phép gọi cùng một phương thức trên nhiều đối tượng khác nhau mà không cần biết kiểu cụ thể.
   * Mở rộng dễ dàng: Thêm lớp mới mà không sửa code cũ (tuân thủ Open/Closed Principle).
   * Tăng tính linh hoạt: Dễ triển khai các pattern như Strategy, Factory.

### 💫 KẾT NỐI ĐÁNH GIÁ

**Trọng yếu:** Phần này là **nền tảng OOP** - phải hiểu chắc để học tiếp

## PHT C# [06] - XỬ LÝ LỖI VÀ DỰ ÁN TỔNG HỢP

### 🎯 KIẾN THỨC CỐT LỐI

| **Khái Niệm** | **Ý Nghĩa** | **Ví Dụ** |
| --- | --- | --- |
| **Exception** | Lỗi xảy ra lúc chạy chương trình | DivideByZeroException |
| **try-catch** | Bắt lỗi và xử lý | try { } catch { } |
| **finally** | Code luôn chạy, dù có lỗi hay không | Đóng file, giải phóng tài nguyên |
| **Validation** | Kiểm tra dữ liệu đầu vào | Kiểm tra age > 0 |
| **throw** | Ném ra một exception | throw new Exception("Lỗi"); |

### 📝 DỰ ÁN TỔNG HỢP: HỆ THỐNG QUẢN LÝ SINH VIÊN

**Yêu cầu chức năng:**

✅ 1. Danh sách sinh viên (tối đa 50)

✅ 2. Thêm sinh viên mới (có validation)

✅ 3. Xóa sinh viên theo ID

✅ 4. Cập nhật điểm cho sinh viên

✅ 5. Tính điểm trung bình

✅ 6. Tìm điểm cao nhất / thấp nhất

✅ 7. Tìm sinh viên theo ID

✅ 8. Xử lý lỗi (try-catch)

**TODO 6.1: [TẠO LỚP STUDENT]**

using System;

namespace PHT06\_Project

{

    // CLASS STUDENT

    public class Student

    {

        public string StudentId { get; private set; }

        public string Name { get; private set; }

        public double Score { get; private set; }

        // Constructor with validation

        public Student(string id, string name, double score)

        {

            if (string.IsNullOrWhiteSpace(id))

                throw new ArgumentException("StudentId khong duoc rong.");

            if (string.IsNullOrWhiteSpace(name))

                throw new ArgumentException("Name khong duoc rong.");

            if (score < 0 || score > 10)

                throw new ArgumentOutOfRangeException(nameof(score), "Score phai tu 0 den 10.");

            StudentId = id.Trim();

            Name = name.Trim();

            Score = Math.Round(score, 2);

        }

        // Update score with validation

        public void UpdateScore(double newScore)

        {

            if (newScore < 0 || newScore > 10)

                throw new ArgumentOutOfRangeException(nameof(newScore), "Score phai tu 0 den 10.");

            Score = Math.Round(newScore, 2);

        }

        // Display student info

        public void Display()

        {

            Console.WriteLine($"ID: {StudentId} | Ten: {Name} | Diem: {Score:0.00}");

        }

    }

}

**TODO 6.2: [TẠO LỚP STUDENT MANAGER]**

using System;

namespace PHT06\_Project

{

    // LOP STUDENT MANAGER

    public class StudentManager

    {

        private Student[] students = new Student[50];

        private int count = 0; // So luong sinh vien hien tai

        // Them sinh vien moi, kiem tra trung lap

        public bool AddStudent(string id, string name, double score)

        {

            if (count >= students.Length)

                throw new InvalidOperationException("Danh sach da day (toi da 50 sinh vien).");

            if (FindIndexById(id) != -1)

                throw new InvalidOperationException($"Da ton tai sinh vien voi ID '{id}'.");

            var s = new Student(id, name, score); // co the nem exception tu validation

            students[count++] = s;

            return true;

        }

        // Xoa sinh vien theo ID

        public bool RemoveStudent(string id)

        {

            int idx = FindIndexById(id);

            if (idx == -1) return false;

            // Dich trai de lap cho trong

            for (int i = idx; i < count - 1; i++)

                students[i] = students[i + 1];

            students[count - 1] = null!;

            count--;

            return true;

        }

        // Cap nhat diem

        public bool UpdateScore(string id, double newScore)

        {

            Student s = FindStudentById(id);

            if (s == null) return false;

            s.UpdateScore(newScore); // co the nem exception neu ngoai [0..10]

            return true;

        }

        // Tinh diem trung binh

        public double GetAverageScore()

        {

            if (count == 0) return double.NaN;

            double sum = 0;

            for (int i = 0; i < count; i++)

                sum += students[i].Score;

            return Math.Round(sum / count, 2);

        }

        // Tim diem cao nhat

        public double GetMaxScore()

        {

            if (count == 0) return double.NaN;

            double max = double.MinValue;

            for (int i = 0; i < count; i++)

                if (students[i].Score > max) max = students[i].Score;

            return Math.Round(max, 2);

        }

        // Tim diem thap nhat

        public double GetMinScore()

        {

            if (count == 0) return double.NaN;

            double min = double.MaxValue;

            for (int i = 0; i < count; i++)

                if (students[i].Score < min) min = students[i].Score;

            return Math.Round(min, 2);

        }

        // Tim sinh vien theo ID

        public Student FindStudentById(string id)

        {

            int idx = FindIndexById(id);

            return idx == -1 ? null : students[idx];

        }

        // In danh sach tat ca sinh vien

        public void DisplayAllStudents()

        {

            if (count == 0)

            {

                Console.WriteLine(">> Danh sach trong.");

                return;

            }

            Console.WriteLine("\n-- DANH SACH SINH VIEN --");

            for (int i = 0; i < count; i++)

                students[i].Display();

        }

        // Helper: tim vi tri theo ID

        private int FindIndexById(string id)

        {

            for (int i = 0; i < count; i++)

            {

                if (students[i].StudentId.Equals(id, StringComparison.OrdinalIgnoreCase))

                    return i;

            }

            return -1;

        }

    }

}

**TODO 6.3: [MENU CHÍNH - MAIN PROGRAM]**

using System;

namespace PHT06\_Project

{

    class Program

    {

        static void Main(string[] args)

        {

            StudentManager manager = new StudentManager();

            bool running = true;

            while (running)

            {

                Console.WriteLine("\n========== MENU ==========");

                Console.WriteLine("1. Them sinh vien");

                Console.WriteLine("2. Xoa sinh vien");

                Console.WriteLine("3. Cap nhat diem");

                Console.WriteLine("4. In danh sach");

                Console.WriteLine("5. Tinh diem trung binh");

                Console.WriteLine("6. Tim diem cao nhat");

                Console.WriteLine("7. Tim sinh vien theo ID");

                Console.WriteLine("8. Tim diem thap nhat");

                Console.WriteLine("0. Thoat");

                Console.WriteLine("==========================");

                Console.Write(">> Chon chuc nang: ");

                string input = Console.ReadLine();

                if (!int.TryParse(input, out int choice))

                {

                    Console.WriteLine("!! Lua chon khong hop le. Vui long nhap so.");

                    continue;

                }

                try

                {

                    switch (choice)

                    {

                        case 1:

                            Console.WriteLine("\n-- Them sinh vien --");

                            string id = ReadNonEmpty("Nhap ID: ");

                            string name = ReadNonEmpty("Nhap ten: ");

                            double score = ReadScore("Nhap diem [0..10]: ");

                            if (manager.AddStudent(id, name, score))

                                Console.WriteLine(">> Them thanh cong.");

                            break;

                        case 2:

                            Console.WriteLine("\n-- Xoa sinh vien --");

                            string delId = ReadNonEmpty("Nhap ID can xoa: ");

                            bool removed = manager.RemoveStudent(delId);

                            Console.WriteLine(removed ? ">> Xoa thanh cong." : ">> Khong tim thay sinh vien.");

                            break;

                        case 3:

                            Console.WriteLine("\n-- Cap nhat diem --");

                            string updId = ReadNonEmpty("Nhap ID: ");

                            double newScore = ReadScore("Nhap diem moi [0..10]: ");

                            bool updated = manager.UpdateScore(updId, newScore);

                            Console.WriteLine(updated ? ">> Cap nhat thanh cong." : ">> Khong tim thay sinh vien.");

                            break;

                        case 4:

                            manager.DisplayAllStudents();

                            break;

                        case 5:

                            double avg = manager.GetAverageScore();

                            Console.WriteLine(double.IsNaN(avg)

                                ? ">> Chua co sinh vien de tinh diem trung binh."

                                : $">> Diem trung binh: {avg:0.00}");

                            break;

                        case 6:

                            double max = manager.GetMaxScore();

                            Console.WriteLine(double.IsNaN(max)

                                ? ">> Chua co sinh vien de tim diem cao nhat."

                                : $">> Diem cao nhat: {max:0.00}");

                            break;

                        case 7:

                            Console.WriteLine("\n-- Tim sinh vien theo ID --");

                            string findId = ReadNonEmpty("Nhap ID: ");

                            Student found = manager.FindStudentById(findId);

                            if (found == null)

                                Console.WriteLine(">> Khong tim thay sinh vien.");

                            else

                                found.Display();

                            break;

                        case 8:

                            double min = manager.GetMinScore();

                            Console.WriteLine(double.IsNaN(min)

                                ? ">> Chua co sinh vien de tim diem thap nhat."

                                : $">> Diem thap nhat: {min:0.00}");

                            break;

                        case 0:

                            running = false;

                            Console.WriteLine(">> Thoat chuong trinh. Tam biet!");

                            break;

                        default:

                            Console.WriteLine("!! Lua chon khong hop le. Vui long chon tu 0 den 8.");

                            break;

                    }

                }

                catch (Exception ex)

                {

                    Console.WriteLine($"!! Loi: {ex.Message}");

                }

            }

        }

        // Helper: doc chuoi khong rong

        static string ReadNonEmpty(string prompt)

        {

            while (true)

            {

                Console.Write(prompt);

                string s = Console.ReadLine();

                if (!string.IsNullOrWhiteSpace(s)) return s.Trim();

                Console.WriteLine("!! Khong duoc de trong. Vui long nhap lai.");

            }

        }

        // Helper: doc diem [0..10]

        static double ReadScore(string prompt)

        {

            while (true)

            {

                Console.Write(prompt);

                string s = Console.ReadLine();

                if (double.TryParse(s, out double val) && val >= 0 && val <= 10)

                    return val;

                Console.WriteLine("!! Diem phai la so tu 0 den 10. Vui long nhap lai.");

            }

        }

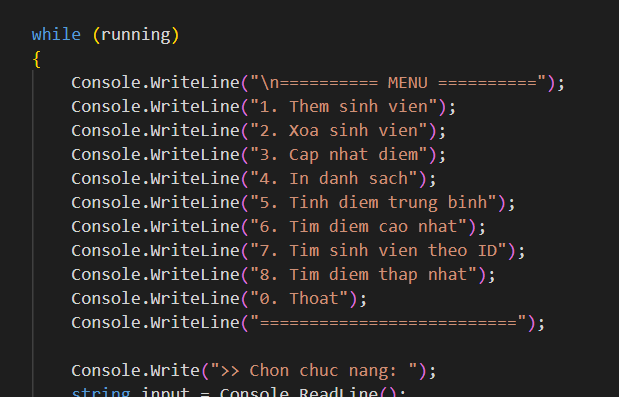
    }

}

**📸 CHỨNG THỰC DỰ ÁN:**

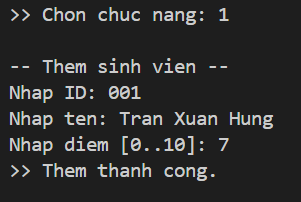
* Chụp screenshot menu
* Chụp screenshot các chức năng: thêm, xóa, cập nhật, hiển thị
* Chụp screenshot khi có lỗi (input sai), kiểm tra xử lý lỗi
* File source code hoàn chỉnh

Menu

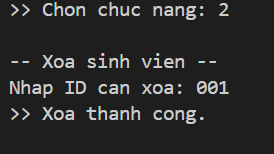


Các chức năng: thêm, xoá, cập nhật, hiển thị

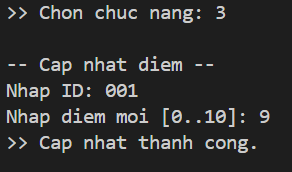
* Thêm sinh viên:



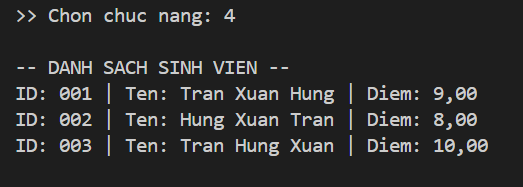
* Xoá sinh viên:



* Cập nhật sinh viên:

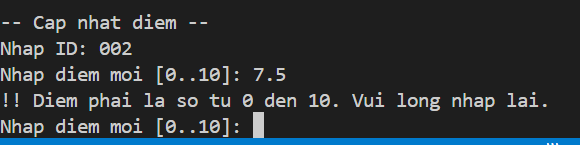


* Hiển thị sinh viên

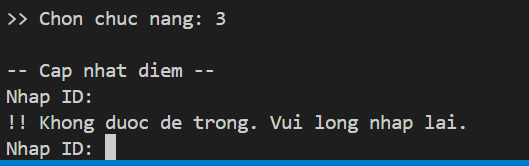


Khi có lỗi (input sai), kiểm tra xử lý lỗi

* Lỗi khi nhập số không phải số tự nhiên từ 0 đến 10



* Lỗi để trống dữ liệu



### 🤔 CÂU HỎI PHẢN BIỆN

1. **Nếu người dùng nhập dữ `liệu sai (ví dụ: điểm là 15), chương trình nên xử lý như thế nào?**
2. \*\*Có thể dùng Listthay vì array không? Ưu điểm gì?\*\*
3. **Nếu muốn lưu dữ liệu xuống file sau khi chương trình thoát, làm sao?**

Bài làm

**1. Nếu người dùng nhập dữ liệu sai (ví dụ: điểm là 15), chương trình nên xử lý thế nào?**

* **Hiện tại**: Hàm ReadScore() kiểm tra điểm phải từ 0 đến 10. Nếu sai, in thông báo và yêu cầu nhập lại.
* **Đơn giản**: Không cho phép lưu dữ liệu sai, bắt nhập lại cho đến khi đúng.

**2. Có thể dùng List<Student> thay vì mảng không? Ưu điểm gì?**

* **Có thể dùng List**.
* **Ưu điểm**:
  + Không giới hạn số lượng phần tử.
  + Có sẵn phương thức tiện lợi (Add, Remove, Find).
  + Dễ dùng LINQ để tính trung bình, tìm max/min.
* **Nhược điểm**: Tốn thêm chút bộ nhớ, nhưng không đáng kể.

**3. Nếu muốn lưu dữ liệu xuống file sau khi thoát, làm sao?**

* **Cách đơn giản**: Ghi ra file CSV hoặc JSON.
* **Ví dụ CSV**:

File.WriteAllText("data.csv", $"{id},{name},{score}");

Hiện thị thêm dòng

* **Đọc lại**: Dùng File.ReadAllLines() và Split(',').

### 💫 KẾT NỐI ĐÁNH GIÁ

**Bước tiếp theo:**

* Nếu hoàn thành được dự án này, bạn đã sẵn sàng cho **ASP.NET Core** Backend
* Phiếu tiếp theo sẽ dạy **advanced OOP** (interfaces, abstract class, LINQ)
* Tiếp theo là **xây dựng Web API** thực tế

## 🎓 HƯỚNG DẪN NỘP BÀI

### Cấu Trúc Thư Mục Nộp Bài:

StudentName\_CSHarp/

├── PHT01\_Introduction/

│ ├── HelloWorld.cs

│ └── HelloWorld\_Output.png

├── PHT02\_Variables/

│ ├── ShoppingCalculator.cs

│ ├── StudentGrades.cs

│ ├── UserProfile.cs

│ └── [screenshots]

├── PHT03\_Conditions/

│ ├── GradeClassification.cs

│ ├── DayOfWeek.cs

│ └── [screenshots]

├── PHT04\_Methods/

│ ├── BasicMethods.cs

│ └── [screenshots]

├── PHT05\_OOP/

│ ├── BasicClass.cs

│ └── [screenshots]

└── PHT06\_Project/

├── Student.cs

├── StudentManager.cs

├── Program.cs

└── [screenshots]

### Yêu Cầu Nộp Bài:

* ✅ **Hoàn thành tất cả TODO** trong mỗi phiếu
* ✅ **Screenshot chạy chương trình** cho mỗi bài
* ✅ **Comment code** rõ ràng
* ✅ **Nộp file .cs** + file .md (tài liệu)
* ✅ **Dự án 06** phải **chạy trơn tru**, **xử lý lỗi tốt**

### Tiêu Chí Đánh Giá:

| **Tiêu Chí** | **Điểm** |
| --- | --- |
| Hoàn thành 100% TODO | 40% |
| Code chạy đúng, không lỗi | 30% |
| Xử lý lỗi (try-catch, validation) | 15% |
| Comment, giải thích logic | 10% |
| Nộp đầy đủ file, cấu trúc rõ ràng | 5% |