## 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 (Comparison Operators) được sử dụng để so sánh hai giá trị và luôn trả về một giá trị Boolean (true hoặc false).

| **Toán tử** | **Ý nghĩa** | **Ví dụ (a=5, b=10)** | **Kết quả** |
| --- | --- | --- | --- |
| **==** | Bằng | $a == b$ | false |
| **!=** | Khác | $a != b$ | true |
| **>** | Lớn hơn | $a > b$ | false |
| **<** | Bé hơn | $a < b$ | true |
| **>=** | Lớn hơn hoặc bằng | $a >= 5$ | true |
| **<=** | Bé hơn hoặc bằng | $a <= 10$ | true |

1. Toán tử logic là gì? (&&, ||, !)

Toán tử logic (Logical Operators) được dùng để kết hợp các điều kiện Boolean lại với nhau, và cũng trả về kết quả là một giá trị Boolean (true hoặc false).

| **Toán tử** | **Ý nghĩa** | **Ví dụ (Giả sử C1​ là true, C2​ là `false$)** | **Kết quả** |
| --- | --- | --- | --- |
| **&&** | VÀ (AND) | $C\_1$ && $C\_2$ | false |
| **||** | HOẶC (OR) | $C\_1$ || $ $C$ | true |
| **!** | PHỦ ĐỊNH (NOT) | $!C\_1$ | false |

Ví dụ: Muốn kiểm tra $x$ có nằm trong khoảng $(0, 10)$ hay không, ta dùng:

if (x > 0 && x < 10)

1. Sự khác biệt giữa for và foreach là gì?

Cả hai đều là vòng lặp, nhưng mục đích sử dụng có sự khác biệt rõ rệt:

| **Tính năng** | **for** | **foreach** |
| --- | --- | --- |
| **Mục đích chính** | Lặp lại hành động với số lần xác định trước, hoặc lặp dựa trên một điều kiện đếm/kiểm tra. | Lặp qua từng phần tử của một bộ sưu tập (mảng, danh sách, v.v.). |
| **Cách thức lặp** | Dùng một biến đếm (index) để truy cập phần tử (ví dụ: array[i]). | Truy cập trực tiếp giá trị của từng phần tử trong bộ sưu tập. |
| **Phạm vi** | Có thể lặp lùi, lặp tiến, lặp bước nhảy, và thay đổi giá trị của biến đếm/index và các phần tử. | Thường chỉ lặp xuôi, không thể thay đổi giá trị của phần tử đang lặp trong bộ sưu tập (thường là chỉ đọc). |
| **Khi dùng** | Khi cần biết vị trí của phần tử, hoặc khi cần lặp theo một logic không phải là duyệt tuần tự hết các phần tử (vd: lặp 5 lần). | Khi chỉ cần giá trị của từng phần tử trong bộ sưu tập và duyệt tất cả các phần tử. |

*Ví dụ:*

* **for**: for (int i = 0; i < array.Length; i++) { Console.WriteLine(array[i]); }
* **foreach**: foreach (var item in array) { Console.WriteLine(item); }

1. Khi nào dùng break? Khi nào dùng continue?

Đây là hai từ khóa quan trọng để kiểm soát luồng của vòng lặp:

**break**

* Mục đích: Thoát hẳn khỏi vòng lặp (hoặc khối switch) đang chứa nó.
* Khi dùng: Khi đã tìm thấy kết quả mong muốn, hoặc gặp một điều kiện cần phải dừng ngay lập tức toàn bộ quá trình lặp mà không cần xét đến các lần lặp tiếp theo.
* *Ví dụ:* Khi tìm một số trong mảng, thấy rồi thì break để tiết kiệm thời gian.

**continue**

* Mục đích: Bỏ qua phần mã còn lại của lần lặp hiện tại và chuyển sang lần lặp kế tiếp của vòng lặp.
* Khi dùng: Khi gặp một điều kiện mà bạn muốn bỏ qua việc xử lý cho trường hợp đó, nhưng vẫn muốn tiếp tục vòng lặp với các phần tử/lần lặp khác.
* *Ví dụ:* Khi xử lý dữ liệu, nếu gặp giá trị null hoặc giá trị không hợp lệ, ta dùng continue để bỏ qua giá trị đó và chuyển sang giá trị tiếp theo.

**Tóm lại:**

* break $\rightarrow$ Dừng vòng lặp. (Dừng hẳn)
* continue $\rightarrow$ Dừng lần lặp hiện tại. (Chuyển sang lần lặp mới)

### 📝 VÍ DỤ THỰC HÀNH

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

using System;

namespace GradeClassification

{

class Program

{

static void Main(string[] args)

{

// TODO: Nhập điểm (giả sử: 75)

int score = 75; // Thay đổi giá trị để test khác

// TODO: Phân loại điểm

// 90-100: A (Xuất sắc)

// 80-89: B (Khá)

// 70-79: C (Trung bình)

// 60-69: D (Yếu)

// < 60: F (Không đạt)

 if (score >= 90)

            {

                grade = "A (Xuất sắc)";

            }

            else if (score >= 80)

            {

                grade = "B (Khá)";

            }

            else if (score >= 70)

            {

                grade = "C (Trung bình)";

            }

            else if (score >= 60)

            {

                grade = "D (Yếu)";

            }

            else

            {

                grade = "F (Không đạt)";

            }

            // TODO: In kết quả

            Console.WriteLine($"Điểm số: {score}");

            Console.WriteLine($"Xếp loại: {grade}");

        }

    }

}

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

using System;

namespace DayOfWeek

{

    class Program

    {

        static void Main(string[] args)

        {

            int day = 17;

            string dayName = "";

            switch (day)

            {

                case 1:

                    dayName = "Thứ Hai";

                    break;

                case 2:

                    dayName = "Thứ Ba";

                    break;

                case 3:

                    dayName = "Thứ Tư";

                    break;

                case 4:

                    dayName = "Thứ Năm";

                    break;

                case 5:

                    dayName = "Thứ Sáu";

                    break;

                case 6:

                    dayName = "Thứ Bảy";

                    break;

                case 7:

                    dayName = "Chủ Nhật";

                    break;

                default:

                    dayName = "Ngày không hợp lệ";

                    break;

            }

            Console.WriteLine($"Số ngày {day} là: {dayName}");

        }

    }

}

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

using System;

namespace SumCalculation

{

    class Program

    {

        static void Main(string[] args)

        {

            int sum = 0;

            for (int i = 1; i <= 100; i++)

            {

                sum += i;

            }

            Console.WriteLine($"Tổng các số từ 1 đến 100: {sum}");

        }

    }

}

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

using System;

namespace GuessNumber

{

    class Program

    {

        static void Main(string[] args)

        {

            // TODO: Tạo chương trình đoán số

            int secretNumber = 50;

            int guess;

            int attempts = 0;

            Console.WriteLine("Chào mừng đến với trò chơi Đoán Số! Hãy đoán một số.");

            int[] simulatedGuesses = { 1, 10, 20, 30, 40, 45, 48, 49, 50 };

            int guessIndex = 0;

            do

            {

                guess = simulatedGuesses[guessIndex++];

                attempts++;

                Console.Write($"Lần đoán thứ {attempts} ({guess}): ");

                if (guess < secretNumber)

                {

                    Console.WriteLine("Quá thấp.");

                }

                else if (guess > secretNumber)

                {

                    Console.WriteLine("Quá cao.");

                }

                else

                {

                    Console.WriteLine("Chính xác! Bạn đã đoán đúng.");

                }

            } while (guess != secretNumber);

            Console.WriteLine($"Bạn đã đoán đúng sau {attempts} lần.");

        }

    }

}

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

using System;

namespace ForeachExample

{

    class Program

    {

        static void Main(string[] args)

        {

            // TODO: Tạo mảng tên các bạn

            string[] friends = { "A", "B", "C", "D", "E" };

            // TODO: In danh sách bạn bè

            int index = 1;

            foreach (string friend in friends)

            {

                Console.WriteLine($"{index}. {friend}");

                index++;

            }

        }

    }

}

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

using System;

namespace BreakContinue

{

    class Program

    {

        static void Main(string[] args)

        {

            // TODO: In các số lẻ từ 1 đến 20 (dùng continue)

            Console.Write("Các số lẻ từ 1 đến 20: ");

            for (int i = 1; i <= 20; i++)

            {

                if (i % 2 == 0)

                {

                    continue;

                }

                Console.Write(i + " ");

            }

            Console.WriteLine("\n---");

            // TODO: Tìm số 7 trong mảng [2, 5, 7, 1, 9, 7, 3]

           int[] numbers = { 7, 2, 5, 1 };

            int target = 7;

            for (int i = 0; i < numbers.Length; i++)

            {

                if (numbers[i] == target)

                {

                    Console.WriteLine($"Tìm thấy số {target} tại vị trí {i}");

                    break;

                }

            }

        }

    }

}

**📸 CHỨNG THỰC [3.1 -> 3.6]:**

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

bài 3.1:



Bài 3.2

 

Bài 3.3

 

Bài 3.4

A black background with white text

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Bài 3.5

A screenshot of a computer screen

AI-generated content may be incorrect.

A black background with white letters

AI-generated content may be incorrect.

Bài 3.6

A number on a black background

AI-generated content may be incorrect.

A number on a black background

AI-generated content may be incorrect.

### 🤔 CÂU HỎI PHẢN BIỆN

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

**🔹 Lý do chính:**

**for gọn – rõ – dễ kiểm soát số lần lặp**

Cấu trúc for gom **3 thứ quan trọng vào 1 chỗ**:

for (khởi\_tạo; điều\_kiện; cập\_nhật)

**Ví dụ:**

for (int i = 0; i < 5; i++) {

printf("%d\n", i);

}

Nhìn vào là biết:

* Bắt đầu từ đâu (i = 0)
* Lặp bao nhiêu lần (i < 5)
* Tăng thế nào (i++)

**Với while:**

int i = 0;

while (i < 5) {

printf("%d\n", i);

i++;

}

Dễ:

* **Quên tăng i++ → vòng lặp vô hạn**
* Điều kiện, biến lặp bị **tách rời → khó đọc**

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

Cách 1: Dùng **biến cờ (flag)** – phổ biến & an toàn

bool stop = false;

for (int i = 0; i < 5 && !stop; i++) {

for (int j = 0; j < 5; j++) {

if (i == 2 && j == 3) {

stop = true;

break; // thoát vòng trong

}

}

}

break chỉ thoát **1 vòng**, nên cần **flag** để vòng ngoài biết dừng.

Cách 2: Dùng break **có nhãn (label)** (Java / C#)

outer:

for (int i = 0; i < 5; i++) {

for (int j = 0; j < 5; j++) {

if (i == 2 && j == 3) {

break outer;

}

}

}

Thoát **ngay lập tức cả 2 vòng**  
 Dùng ít thôi, vì dễ rối code.

Cách 3: Dùng return (khi ở trong hàm)

void test() {

for (int i = 0; i < 5; i++) {

for (int j = 0; j < 5; j++) {

if (i == 2 && j == 3) {

return; // thoát luôn hàm

}

}

}

}

Thoát **tất cả vòng lặp + hàm**

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

**Khác nhau cốt lõi:**

| **while** | **do-while** |
| --- | --- |
| Kiểm tra điều kiện trước | Kiểm tra điều kiện sau |
| Có thể không chạy lần nào | Luôn chạy ít nhất 1 lần |

**Ví dụ while:**

int x = 5;

while (x < 3) {

printf("Hello");

}

Không in gì cả

**Ví dụ do-while:**

int x = 5;

do {

printf("Hello");

} while (x < 3);

In **Hello 1 lần**

**Khi nào nên dùng do-while?**

* Menu chương trình
* Nhập dữ liệu ít nhất 1 lần
* Xác nhận người dùng

**Ví dụ menu:**

int choice;

do {

printf("1. Play\n2. Exit\n");

scanf("%d", &choice);

} while (choice != 2);

Luôn hiện menu ít nhất 1 lần – rất hợp lý

### 💫 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?)

**Tái sử dụng code**  
→ Viết 1 lần, dùng nhiều nơi

**Code gọn gàng, dễ đọc**  
→ Chia chương trình lớn thành các phần nhỏ

**Dễ bảo trì & sửa lỗi**  
→ Lỗi ở đâu sửa đúng phương thức đó

**Tăng tính tổ chức & logic**  
→ Mỗi phương thức làm **1 nhiệm vụ rõ rang**

Không cần viết lại phép cộng nhiều lần

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

| **Tham số (Parameter)** | **Đối số (Argument)** |
| --- | --- |
| Khai báo trong phương thức | Giá trị truyền vào |
| Là biến | Là dữ liệu thực |
| Chưa có giá trị cụ thể | Có giá trị cụ thể |

### Ví dụ:

static int Add(int a, int b) // a, b là tham số

{

return a + b;

}

Add(5, 3); // 5, 3 là đối số

**Tham số = cái nhận**  
**Đối số = cái đưa vào**

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

Phải có return:

Phương thức **có kiểu trả về KHÁC** void

static int Sum(int a, int b)

{

return a + b; // BẮT BUỘC

}

Không return → lỗi biên dịch

Không cần return:

Phương thức có kiểu trả về void

static void ShowMessage()

{

Console.WriteLine("Hello");

}

void = chỉ làm việc, **không trả dữ liệu**

1. Nạp chồng phương thức dựa vào điều gì?

Dựa vào:

1. Số lượng tham số
2. Kiểu dữ liệu tham số
3. Thứ tự kiểu tham số

KHÔNG dựa vào:

* Tên biến
* Kiểu trả về

### 📝 VÍ DỤ THỰC HÀNH

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

using System;

namespace BasicMethods

{

class Program

{

// TODO: Viết phương thức tính tổng 2 số

// Tên: Add

// Tham số: int a, int b

// Trả về: int

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// TODO: Viết phương thức tính tích 2 số

// Tên: Multiply

// Tham số: double x, double y

// Trả về: double

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

static void Main(string[] args)

{

// TODO: Gọi phương thức Add và in kết quả

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// TODO: Gọi phương thức Multiply và in kết quả

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

}

}

}

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

using System;

namespace VoidMethods

{

class Program

{

// TODO: Viết phương thức in hộp trang trí

// Tên: PrintBox

// Tham số: string text

// Trả về: void

// Ví dụ: PrintBox("Hello") in ra:

// \*\*\*\*\*\*\*\*\*\*\*

// \* Hello \*

// \*\*\*\*\*\*\*\*\*\*\*

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

static void Main(string[] args)

{

// TODO: Gọi phương thức PrintBox với các giá trị khác nhau

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

}

}

}

**TODO 4.3: [PHƯƠNG THỨC VỚI MẢNG]**

using System;

namespace ArrayMethods

{

class Program

{

// TODO: Viết phương thức tính tổng các phần tử trong mảng

// Tên: SumArray

// Tham số: int[] numbers

// Trả về: int

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// TODO: Viết phương thức tìm số lớn nhất trong mảng

// Tên: FindMax

// Tham số: int[] numbers

// Trả về: int

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

static void Main(string[] args)

{

int[] scores = { 85, 92, 78, 90, 88 };

// TODO: Gọi SumArray và in kết quả

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// TODO: Gọi FindMax và in kết quả

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

}

}

}

**TODO 4.4: [NẠP CHỒNG PHƯƠNG THỨC]**

using System;

namespace MethodOverloading

{

class Program

{

// TODO: Viết phương thức Print có 2 phiên bản

// Phiên bản 1: Print(int x) - in một số

// Phiên bản 2: Print(string text) - in một chuỗi

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// TODO: Viết phương thức Add có 2 phiên bản

// Phiên bản 1: Add(int a, int b) - cộng 2 số nguyên

// Phiên bản 2: Add(double a, double b) - cộng 2 số thực

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

static void Main(string[] args)

{

// TODO: Gọi Print với int

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// TODO: Gọi Print với string

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// TODO: Gọi Add(int, int)

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// TODO: Gọi Add(double, double)

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

}

}

}

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

using System;

namespace Recursion

{

class Program

{

// TODO: Viết phương thức tính giai thừa

// Tên: Factorial

// Tham số: int n

// Trả về: long

// Công thức: n! = n \* (n-1)!

// Điều kiện dừng: 0! = 1

// Gợi ý: Gọi lại chính nó với n-1

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

static void Main(string[] args)

{

// TODO: Tính 5! bằng Factorial(5) và in kết quả

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// TODO: Tính 10! bằng Factorial(10) và in kết quả

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

}

}

}

**📸 CHỨNG THỰC [4.1 -> 4.5]:**

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

Bài 4.1

A screenshot of a computer

Description automatically generated A computer screen shot of a black screen

Description automatically generated

Bài 4.2

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

4.3

A computer screen shot of a black screen

Description automatically generatedA screenshot of a computer

Description automatically generated

4.4

A screen shot of a black and white screen

Description automatically generated

using System;

namespace MethodOverloading

{

class Program

{

// TODO: Viết phương thức Print có 2 phiên bản

// Phiên bản 1: Print(int x)

// Logic: In ra số nguyên kèm chú thích.

static void Print(int x)

{

Console.WriteLine($"In số nguyên: {x}");

}

// Phiên bản 2: Print(string text)

// Logic: In ra chuỗi ký tự kèm chú thích.

static void Print(string text)

{

Console.WriteLine($"In chuỗi: {text}");

}

// TODO: Viết phương thức Add có 2 phiên bản

// Phiên bản 1: Add(int a, int b)

// Logic: Cộng 2 số nguyên.

static int Add(int a, int b)

{

return a + b;

}

// Phiên bản 2: Add(double a, double b)

// Logic: Cộng 2 số thực.

static double Add(double a, double b)

{

return a + b;

}

static void Main(string[] args)

{

// TODO: Gọi Print với int

Print(100);

// TODO: Gọi Print với string

Print("Hello Overloading");

// TODO: Gọi Add(int, int)

int intSum = Add(5, 10);

Console.WriteLine($"Tổng int: {intSum}");

// TODO: Gọi Add(double, double)

double doubleSum = Add(2.5, 3.7);

Console.WriteLine($"Tổng double: {doubleSum}");

Console.ReadKey();

}

}

}

4..5

A screenshot of a computer

Description automatically generated

using System;

namespace Recursion

{

class Program

{

// TODO: Viết phương thức tính giai thừa

// Logic:

// 1. Trường hợp cơ sở (Điểm dừng): Nếu n bằng 0 hoặc 1, trả về 1.

// 2. Trường hợp đệ quy: Trả về n nhân với Giai thừa của (n-1).

// Ví dụ: Factorial(3) = 3 \* Factorial(2) -> 3 \* 2 \* Factorial(1) -> 3 \* 2 \* 1 = 6

static long Factorial(int n)

{

if (n == 0 || n == 1)

{

return 1;

}

return n \* Factorial(n - 1);

}

static void Main(string[] args)

{

// TODO: Tính 5! bằng Factorial(5) và in kết quả

long f5 = Factorial(5);

Console.WriteLine($"5! = {f5}"); // Kết quả: 120

// TODO: Tính 10! bằng Factorial(10) và in kết quả

long f10 = Factorial(10);

Console.WriteLine($"10! = {f10}"); // Kết quả: 3628800

Console.ReadKey();

}

}

}

### 🤔 CÂU HỎI PHẢN BIỆN

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 không nên trong nhiều trường hợp.

Lý do cần nạp chồng:

1. Giữ ý nghĩa thống nhất
   * Cùng 1 hành động → cùng 1 tên
2. Code dễ đọc & dễ nhớ
3. Tránh đặt tên lan man
4. Giống cách các thư viện chuẩn hoạt động

Nạp chồng giúp code gọn, logic rõ ràng và dễ mở rộng  
→ Dùng tên khác nhau *được*, nhưng nạp chồng tốt hơn

1. **Đệ quy có ưu và nhược điểm gì?**

**Ưu điểm:**

1. Code ngắn gọn, dễ hiểu
2. Diễn tả tốt các bài toán có cấu trúc lặp lại
   * Giai thừa
   * Dãy Fibonacci
   * Cây, thư mục, đồ thị

**Nhược điểm:**

1. Tốn bộ nhớ (stack)
2. Dễ gây Stack Overflow
3. Chạy chậm hơn vòng lặp
4. Khó debug khi đệ quy sâu
   * Khi nào nên dùng đệ quy?

Khi:

* Bài toán có thể chia nhỏ giống nhau
* Độ sâu không quá lớn
* Cần code dễ hiểu hơn hiệu năng

Ví dụ:

* Duyệt cây thư mục
* Thuật toán chia để trị (QuickSort, MergeSort)
* DFS
  + Khi nào nên dùng vòng lặp?

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

**NÊN NGẮN**

**Lý do:**

1. Dễ đọc – dễ hiểu
2. Dễ test & sửa lỗi
3. Tái sử dụng tốt hơn
4. Tuân theo nguyên tắc SRP  
   (Single Responsibility Principle)

**Nguyên tắc thực tế:**

* 5 – 20 dòng là đẹp
* Chỉ làm 1 nhiệm vụ duy nhất
* Nếu phải cuộn chuột → quá dài

**Phương thức dài gây:**

* Khó hiểu logic
* Khó bảo trì
* Dễ lỗi dây chuyền

**TỔNG KẾT CUỐI**

* Nạp chồng giúp giữ tên phương thức thống nhất
* Đệ quy đẹp, dễ hiểu nhưng tốn tài nguyên
* Vòng lặp nhanh, an toàn, dùng cho dữ liệu lớn
* Phương thức nên ngắn, mỗi cái làm 1 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?

| **Lớp (Class)** | **Đối tượng (Object)** |
| --- | --- |
| Bản thiết kế | Thể hiện cụ thể |
| Chưa tồn tại thật | Có tồn tại trong bộ nhớ |
| Định nghĩa thuộc tính & phương thức | Có giá trị dữ liệu cụ thể |

Class = khuôn  
Object = sản phẩm từ khuôn

1. Constructor là gì? Tại sao cần constructor?

**** Là **phương thức đặc biệt**

** Trùng tên với lớp**

** Tự động chạy** khi tạo đối tượng

** Không có kiểu trả về**

**Tại sao cần constructor?**

Khởi tạo giá trị ban đầu

Đảm bảo đối tượng luôn ở trạng thái hợp lệ

Giảm lỗi quên gán dữ liệu

Code gọn & an toàn hơn

1. private, public, protected khác gì?

| **Từ khóa** | **Truy cập được từ** |
| --- | --- |
| public | Mọi nơi |
| private | Chỉ trong chính lớp đó |
| protected | Trong lớp đó và lớp con |

1. Kế thừa có tác dụng gì?

Tác dụng:

1. Tái sử dụng code
2. Giảm trùng lặp
3. Mở rộng chức năng lớp cha
4. Tạo quan hệ cha – con rõ ràn

### 📝 VÍ DỤ THỰC HÀNH

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

using System;

namespace BasicClass

{

// TODO: Tạo lớp Student

// Thuộc tính: StudentId, Name, GPA

// Phương thức: Display() (in thông tin)

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

class Program

{

static void Main(string[] args)

{

// TODO: Tạo 2 đối tượng Student

// TODO: Gán giá trị thuộc tính

// TODO: Gọi phương thức Display()

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

}

}

}

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

using System;

namespace Constructor

{

// TODO: Tạo lớp Product với Constructor

// Constructor nhận tham số: productId, productName, price

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

class Program

{

static void Main(string[] args)

{

// TODO: Tạo đối tượng Product dùng Constructor

// TODO: In thông tin sản phẩm

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

}

}

}

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

using System;

namespace Encapsulation

{

// TODO: Tạo lớp BankAccount

// - Private field: \_balance (số dư)

// - Public property: Balance (chỉ đọc, phòng trừy cập trái phép)

// - Public method: Deposit(amount) - gửi tiền

// - Public method: Withdraw(amount) - rút tiền (kiểm tra đủ tiền không)

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

class Program

{

static void Main(string[] args)

{

// TODO: Tạo tài khoản và thử các thao tác

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

}

}

}

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

using System;

namespace Inheritance

{

// TODO: Tạo lớp Animal (cha)

// Thuộc tính: Name

// Phương thức: MakeSound() (in ra "Animal makes a sound")

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// TODO: Tạo lớp Dog (con) kế thừa từ Animal

// Override phương thức MakeSound() (in ra "Woof! Woof!")

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

class Program

{

static void Main(string[] args)

{

// TODO: Tạo đối tượng Animal và Dog

// TODO: Gọi MakeSound() của cả hai

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

}

}

}

**📸 CHỨNG THỰC [5.1 -> 5.4]:**

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

Bài 5.1

A screenshot of a computer

Description automatically generated

Bài 5.2

A screenshot of a computer

Description automatically generated

Bài 5.3

A screenshot of a computer

Description automatically generated

Bài 5.4

A screenshot of a computer

Description automatically generated

### 🤔 CÂU HỎI PHẢN BIỆN

1. **Tại sao encapsulation (che giấu dữ liệu) lại quan trọng?**

**1. Bảo vệ dữ liệu**

* Ngăn sửa trực tiếp dữ liệu quan trọng
* Tránh trạng thái đối tượng bị sai

**2. Kiểm soát hành vi**

* Mọi thay đổi dữ liệu đều đi qua logic kiểm tra

**3. Giảm phụ thuộc (Low Coupling)**

* Bên ngoài **không cần biết chi tiết bên trong**
* Dễ sửa code mà không ảnh hưởng nơi khác

**4. Dễ bảo trì & mở rộng**

* Thay đổi cách xử lý nội bộ không làm hỏng chương trình

**Kết luận:**

**Encapsulation giúp chương trình an toàn, ổn định và dễ bảo trì hơn**

1. **Kế thừa vs. Composition - khi nào dùng cái nào?**

| **Kế thừa** | **Composition** |
| --- | --- |
| Quan hệ **is-a** | Quan hệ **has-a** |
| Lớp con là 1 dạng của lớp cha | Đối tượng chứa đối tượng khác |
| Gắn chặt (tight coupling) | Linh hoạt hơn |

**Khi nên dùng kế thừa:**

* Quan hệ cha – con rõ ràng
* Lớp con thực sự là một dạng của lớp cha
* Ít thay đổi cấu trúc

Khi nên dùng composition:

* Cần linh hoạt
* Dễ thay đổi thành phần
* Tránh phụ thuộc chặt chẽ

Nguyên tắc hiện đại: *Favor Composition over Inheritance*

**Kết luận:**

Kế thừa dùng khi is-a  
Composition dùng khi has-a và muốn linh hoạt

1. **Lợi ích của polymorphism (đa hình) là gì?**

**1. Code linh hoạt & mở rộng**

* Thêm lớp mới không cần sửa code cũ

**2. Giảm if-else**

Không đa hình:

if (type == "Dog") ...

else if (type == "Cat") ...

Có đa hình:

animal.Speak();

**3. Tuân theo nguyên tắc OCP**

* Open for extension
* Closed for modification

**4. Code sạch & chuyên nghiệp**

* Đúng chuẩn OOP
* Dễ bảo trì

**Kết luận:**

Polymorphism giúp chương trình mở rộng dễ, ít sửa code, giảm rối logic

### 💫 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]**

// Lớp Student chứa thông tin và điểm của 1 sinh viên

public class Student

{

public string StudentId { get; set; }

public string Name { get; set; }

public double Score { get; set; }

// Constructor

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

{

// TODO: Khởi tạo các thuộc tính với validation

// - StudentId không được rỗng

// - Name không được rỗng

// - Score phải từ 0 đến 10

}

// Phương thức in thông tin

public void Display()

{

// TODO: In ra "ID: [StudentId] | Tên: [Name] | Điểm: [Score]"

}

}

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

public class StudentManager

{

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

private int count = 0; // Số lượng sinh viên hiện tại

// TODO: Phương thức AddStudent(string id, string name, double score)

// Thêm sinh viên mới, kiểm tra trùng lặp

// TODO: Phương thức RemoveStudent(string id)

// Xóa sinh viên theo ID

// TODO: Phương thức UpdateScore(string id, double newScore)

// Cập nhật điểm

// TODO: Phương thức GetAverageScore()

// Tính điểm trung bình

// TODO: Phương thức GetMaxScore()

// Tìm điểm cao nhất

// TODO: Phương thức FindStudentById(string id)

// Trả về đối tượng Student hoặc null

// TODO: Phương thức DisplayAllStudents()

// In danh sách tất cả sinh viên

}

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

using System;

namespace StudentManagementSystem

{

class Program

{

static void Main(string[] args)

{

StudentManager manager = new StudentManager();

bool running = true;

while (running)

{

// TODO: In menu

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

Console.WriteLine("1. Thêm sinh viên");

Console.WriteLine("2. Xóa sinh viên");

Console.WriteLine("3. Cập nhật điểm");

Console.WriteLine("4. In danh sách");

Console.WriteLine("5. Tính điểm trung bình");

Console.WriteLine("6. Tìm điểm cao nhất");

Console.WriteLine("7. Tìm sinh viên");

Console.WriteLine("0. Thoát");

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

// TODO: Nhận lựa chọn từ người dùng

// TODO: Dùng switch xử lý từng lựa chọn

// TODO: Thêm try-catch để xử lý lỗi

}

}

}

}

**📸 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

A computer screen shot of a black screen

Description automatically generated A computer screen shot of a black screen

Description automatically generated A computer screen shot of a black screen

Description automatically generated A computer screen shot of a black screen

Description automatically generated A screenshot of a computer

Description automatically generated

### 🤔 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?**

KHÔNG nên để chương trình crash

KHÔNG tin dữ liệu người dùng

**Cách xử lý chuẩn:**

**1. Kiểm tra điều kiện (Validation)**

int score;

do

{

Console.Write("Nhập điểm (0–10): ");

} while (!int.TryParse(Console.ReadLine(), out score) || score < 0 || score > 10);

Bắt:

* Nhập chữ
* Nhập ngoài phạm vi (15, -3)

**2. Thông báo lỗi rõ ràng**

Console.WriteLine("Điểm phải từ 0 đến 10!");

**3. Không cho lưu dữ liệu sai**

Chỉ xử lý khi dữ liệu **hợp lệ**

**Kết luận:**

Luôn validate dữ liệu đầu vào để chương trình an toàn & ổn định

1. \*\*Có thể dùng Listthay vì array không? Ưu điểm gì?\*\*

| **Array** | **List<T>** |
| --- | --- |
| Kích thước cố định | Kích thước **động** |
| Khó thêm/xóa | Thêm/xóa **dễ** |
| Ít phương thức | Nhiều method tiện lợi |
| Phải tự quản lý | Tự quản lý bộ nhớ |

**Ưu điểm của List:**

1. Không cần biết trước số phần tử
2. Code ngắn & rõ
3. Dễ mở rộng
4. Ít lỗi hơn array
5. **Nếu muốn lưu dữ liệu xuống file sau khi chương trình thoát, làm sao?**

### Cách đơn giản nhất: **ghi file text**

using System.IO;

File.WriteAllText("data.txt", "Nguyen Van A - 8");

Ghi nhiều dòng:

List<string> data = new List<string>

{

"An - 8",

"Binh - 9"

};

File.WriteAllLines("data.txt", data);

### 💫 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% |