## 

## 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: Là bản thiết kế hoặc khuôn mẫu logic.

Đối tượng: Là thể hiện cụ thể (instance) của lớp, tồn tại trong bộ nhớ.

Khác biệt: Lớp chỉ định nghĩa, đối tượng là sản phẩm được tạo ra từ định nghĩa đó.

1. Constructor là gì? Tại sao cần constructor?

**A. Constructor là gì?**

Constructor (Hàm tạo) là một **phương thức đặc biệt** trong một lớp có các đặc điểm sau:

* Tên **phải trùng với tên của lớp**.
* **Không có kiểu trả về** (kể cả void).
* Tự động được gọi **ngay lập tức** khi một đối tượng của lớp đó được **khởi tạo** (new ClassName(...)).

**B. Tại sao cần Constructor?**

Constructor đóng vai trò thiết yếu để đảm bảo đối tượng được khởi tạo ở trạng thái **hợp lệ và sẵn sàng sử dụng**. Các lý do chính bao gồm:

* **Khởi tạo Giá trị:** Gán giá trị ban đầu cho các thuộc tính (Properties) của đối tượng (ví dụ: gán tên, tuổi, ID ngay khi tạo một đối tượng Sinh viên).
* **Thực hiện Logic Cần thiết:** Chạy bất kỳ logic thiết lập nào phải xảy ra trước khi đối tượng được sử dụng (ví dụ: kết nối cơ sở dữ liệu, đọc tệp cấu hình).
* **Ép buộc Tính hợp lệ:** Đảm bảo rằng đối tượng không thể tồn tại mà không có dữ liệu quan trọng (ví dụ: buộc phải truyền tên và mật khẩu khi tạo đối tượng User).

1. private, public, protected khác gì?

public: Dành cho giao diện, những gì cần được tiếp xúc (Exposed).

private: Dành cho các chi tiết nội bộ cần được che giấu (Hidden), cốt lõi của tính Đóng gói.

protected: Dành cho các thành phần được chia sẻ một cách an toàn giữa lớp cha và các lớp con trong hệ thống Kế thừa.

1. Kế thừa có tác dụng gì?

Là gì: Lớp con thừa hưởng thuộc tính và phương thức từ lớp cha.

Tác dụng:

Tái sử dụng mã (Code Reusability).

Thiết lập mối quan hệ "Is-A" (Ví dụ: Chó là Động vật).

Là nền tảng cho tính Đa hình

### 📝 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]:**

CODE 5.1

using System;

namespace BasicClass

{

    // Tạo lớp Student

    public class Student

    {

        // Thuộc tính

        public int StudentId { get; set; }

        public string Name { get; set; } = string.Empty; // FIX CS8618

        public double GPA { get; set; }

        // Phương thức in thông tin

        public void Display()

        {

            Console.WriteLine("--- Thông tin Sinh viên ---");

            Console.WriteLine($"Mã số: {StudentId}");

            Console.WriteLine($"Tên: {Name}");

            Console.WriteLine($"GPA: {GPA:F2}");

        }

    }

    class Program

    {

        static void Main(string[] args)

        {

            Student student1 = new Student();

            Student student2 = new Student();

            student1.StudentId = 1001;

            student1.Name = "Nguyễn Văn A";

            student1.GPA = 3.5;

            student2.StudentId = 1002;

            student2.Name = "Trần Thị B";

            student2.GPA = 3.951;

            student1.Display();

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

            student2.Display();

        }

    }

}

A screenshot of a computer program

AI-generated content may be incorrect.

CODE 5.2

using System;

namespace Constructor

{

    // TODO: Tạo lớp Product với Constructor (Đã cập nhật thêm Description và Quantity)

    public class Product

    {

        public int ProductId { get; set; }

        public string ProductName { get; set; }

        public decimal Price { get; set; }

        // Thuộc tính mới

        public string Description { get; set; }

        public int Quantity { get; set; }

        // Constructor nhận thêm tham số: description, quantity

        public Product(int productId, string productName, decimal price, string description, int quantity)

        {

            ProductId = productId;

            ProductName = productName;

            Price = price;

            Description = description; // Gán giá trị mới

            Quantity = quantity;       // Gán giá trị mới

            Console.WriteLine($"Sản phẩm '{ProductName}' (ID: {ProductId}) đã được tạo hoàn tất.");

        }

        public void Display()

        {

            Console.WriteLine($"--- Thông tin Sản phẩm Chi tiết ---");

            Console.WriteLine($"ID: {ProductId}");

            Console.WriteLine($"Tên: {ProductName}");

            Console.WriteLine($"Giá: {Price:C}");

            Console.WriteLine($"Mô tả: {Description}"); // In thông tin mới

            Console.WriteLine($"Tồn kho: {Quantity} đơn vị"); // In thông tin mới

        }

    }

    class Program

    {

        static void Main(string[] args)

        {

            // TODO: Tạo đối tượng Product dùng Constructor với các tham số mới

            Product productB = new Product(

                2,

                "Laptop LOQ Pro",

                255990000.00m,

                "Màn hình 144hz 16.7 inch, Chip Intel 10 Gen 13 , RAM 80GB, RTX 8060 20GB .",

                50

            );

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

            // TODO: In thông tin sản phẩm

            productB.Display();

        }

    }

}

A screenshot of a computer program

AI-generated content may be incorrect.

CODE 5.3

using System;

namespace Encapsulation

{

    // TODO: Tạo lớp BankAccount

    public class BankAccount

    {

        // - Private field: \_balance (số dư)

        private decimal \_balance;

        // Constructor để khởi tạo số dư ban đầu

        public BankAccount(decimal initialBalance)

        {

            if (initialBalance >= 0)

            {

                \_balance = initialBalance;

            }

            else

            {

                Console.WriteLine("Số dư ban đầu không hợp lệ. Đặt về 0.");

                \_balance = 0;

            }

        }

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

        public decimal Balance

        {

            get { return \_balance; }

            // Không có 'set' để chỉ cho phép đọc từ bên ngoài

        }

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

        public void Deposit(decimal amount)

        {

            if (amount > 0)

            {

                \_balance += amount;

                Console.WriteLine($"Gửi thành công: {amount:C}. Số dư mới: {Balance:C}");

            }

            else

            {

                Console.WriteLine("Lỗi: Số tiền gửi phải lớn hơn 0.");

            }

        }

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

        public bool Withdraw(decimal amount)

        {

            if (amount <= 0)

            {

                Console.WriteLine("Lỗi: Số tiền rút phải lớn hơn 0.");

                return false;

            }

            if (amount > \_balance)

            {

                Console.WriteLine($"Rút thất bại: Không đủ số dư ({amount:C} > {Balance:C}).");

                return false;

            }

            \_balance -= amount;

            Console.WriteLine($"Rút thành công: {amount:C}. Số dư mới: {Balance:C}");

            return true;

        }

    }

    class Program

    {

        static void Main(string[] args)

        {

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

            BankAccount account = new BankAccount(1000000); // 1,000,000 VNĐ

            Console.WriteLine($"Số dư ban đầu: {account.Balance:C}");

            account.Deposit(500000);   // Gửi 500,000

            account.Withdraw(200000);  // Rút 200,000

            account.Withdraw(2000000); // Thử rút quá số dư

            Console.WriteLine($"\nSố dư cuối cùng: {account.Balance:C}");

        }

    }

}

A screenshot of a computer program

AI-generated content may be incorrect.

CODE 5.4

using System;

namespace Inheritance

{

    // Lớp cha

    public class Animal

    {

        // FIX CS8618

        public string Name { get; set; } = string.Empty;

        // Phương thức virtual

        public virtual void MakeSound()

        {

            Console.WriteLine($"{Name} (Animal): Animal makes a sound");

        }

    }

    // Lớp con

    public class Dog : Animal

    {

        public override void MakeSound()

        {

            Console.WriteLine($"{Name} (Dog): Woof! Woof!");

        }

    }

    class Program

    {

        static void Main(string[] args)

        {

            Animal genericAnimal = new Animal() { Name = "Generic" };

            Dog myDog = new Dog() { Name = "Spike" };

            Console.WriteLine("Gọi MakeSound() của Animal:");

            genericAnimal.MakeSound();

            Console.WriteLine("\nGọi MakeSound() của Dog (đã bị ghi đè):");

            myDog.MakeSound();

        }

    }

}

A screenshot of a computer program

AI-generated content may be incorrect.

* 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?**

**Trả lời: Encapsulation giúp che giấu dữ liệu, bảo vệ dữ liệu khỏi truy cập sai, kiểm soát cách thay đổi dữ liệu và giúp chương trình dễ bảo trì, an toàn hơn.**

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

**Trả lời: Inheritance dùng khi có quan hệ “là một” (IS-A), ví dụ Dog là Animal.**

**Composition dùng khi có quan hệ “có một” (HAS-A), linh hoạt hơn và dễ mở rộng.**

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

Trả lời: Polymorphism cho phép **gọi cùng một phương thức nhưng hành vi khác nhau**, giúp code **linh hoạt, dễ mở rộng và giảm lặp code**.

### 💫 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

A screen shot of a computer

AI-generated content may be incorrect.

Chụp screenshot các chức năng: thêm, xóa, cập nhật, hiển thị

\*Thêm Sinh Viên

A black screen with white text

AI-generated content may be incorrect.

\*Xóa Sinh Viên

A black screen with white text

AI-generated content may be incorrect.

\*Cập Nhật Điểm Của Sinh Viên

A screen shot of a computer

AI-generated content may be incorrect.

\*Hiển Thị Danh Sách Sinh Viên

A screenshot of a computer

AI-generated content may be incorrect.

* Chụp screenshot khi có lỗi (input sai), kiểm tra xử lý lỗi

\*Khi Xóa Sinh Viên Chưa Được Nhập Dữ Liệu Vào

A black screen with white text

AI-generated content may be incorrect.

\*Khi Nhập Dữ Liệu Sai

A screenshot of a computer

AI-generated content may be incorrect.

* File source code hoàn chỉnh

using System;

using System.Linq;

namespace StudentManagementSystem

{

    // Custom Exception

    public class InvalidStudentDataException : Exception

    {

        public InvalidStudentDataException(string message) : base(message) { }

    }

    // ================= STUDENT =================

    public class Student

    {

        public string StudentId { get; set; }

        public string Name { get; set; }

        public double Score { get; set; }

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

        {

            if (string.IsNullOrWhiteSpace(id))

                throw new InvalidStudentDataException("Mã sinh viên không được rỗng.");

            if (string.IsNullOrWhiteSpace(name))

                throw new InvalidStudentDataException("Tên sinh viên không được rỗng.");

            if (score < 0 || score > 10)

                throw new InvalidStudentDataException("Điểm phải từ 0 đến 10.");

            StudentId = id;

            Name = name;

            Score = score;

        }

        public void Display()

        {

            Console.WriteLine($"ID: {StudentId} | Tên: {Name} | Điểm: {Score:F2}");

        }

    }

    // ================= STUDENT MANAGER =================

    public class StudentManager

    {

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

        private int count = 0;

        private int FindStudentIndex(string id)

        {

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

            {

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

                    return i;

            }

            return -1;

        }

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

        {

            if (count >= 50)

                throw new InvalidOperationException("Danh sách đã đầy.");

            if (FindStudentIndex(id) != -1)

                throw new InvalidOperationException("ID sinh viên đã tồn tại.");

            students[count++] = new Student(id, name, score);

            Console.WriteLine("✅ Thêm sinh viên thành công.");

        }

        public void RemoveStudent(string id)

        {

            int index = FindStudentIndex(id);

            if (index == -1)

                throw new ArgumentException("Không tìm thấy sinh viên.");

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

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

            students[count - 1] = null!;

            count--;

            Console.WriteLine("✅ Xóa sinh viên thành công.");

        }

        public void UpdateScore(string id, double newScore)

        {

            int index = FindStudentIndex(id);

            if (index == -1)

                throw new ArgumentException("Không tìm thấy sinh viên.");

            if (newScore < 0 || newScore > 10)

                throw new InvalidStudentDataException("Điểm phải từ 0 đến 10.");

            students[index].Score = newScore;

            Console.WriteLine("✅ Cập nhật điểm thành công.");

        }

        public double GetAverageScore()

        {

            if (count == 0) return 0;

            return students.Take(count).Average(s => s.Score);

        }

        public double GetMaxScore()

        {

            if (count == 0)

                throw new InvalidOperationException("Danh sách trống.");

            return students.Take(count).Max(s => s.Score);

        }

        public double GetMinScore()

        {

            if (count == 0)

                throw new InvalidOperationException("Danh sách trống.");

            return students.Take(count).Min(s => s.Score);

        }

        public Student? FindStudentById(string id)

        {

            int index = FindStudentIndex(id);

            return index != -1 ? students[index] : null;

        }

        public Student GetStudentWithMaxScore()

        {

            double max = GetMaxScore();

            return students.Take(count).First(s => s.Score == max);

        }

        public Student GetStudentWithMinScore()

        {

            double min = GetMinScore();

            return students.Take(count).First(s => s.Score == min);

        }

        public void DisplayAllStudents()

        {

            if (count == 0)

            {

                Console.WriteLine("Danh sách sinh viên trống.");

                return;

            }

            Console.WriteLine("\n--- DANH SÁCH SINH VIÊN ---");

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

            {

                Console.Write($"{i + 1}. ");

                students[i].Display();

            }

        }

    }

    // ================= PROGRAM =================

    class Program

    {

        static void Main()

        {

            StudentManager manager = new StudentManager();

            manager.AddStudent("SV001", "Nguyễn Văn A", 8.5);

            manager.AddStudent("SV002", "Trần Thị B", 7.2);

            manager.AddStudent("SV003", "Lê Văn C", 9.8);

            bool running = true;

            while (running)

            {

                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. Điểm trung bình");

                Console.WriteLine("6. Điểm cao nhất / thấp nhất");

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

                Console.WriteLine("0. Thoát");

                Console.Write("Chọn: ");

                string choiceInput = Console.ReadLine() ?? "";

                try

                {

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

                        continue;

                    switch (choice)

                    {

                        case 1:

                            HandleAddStudent(manager);

                            break;

                        case 2:

                            HandleRemoveStudent(manager);

                            break;

                        case 3:

                            HandleUpdateScore(manager);

                            break;

                        case 4:

                            manager.DisplayAllStudents();

                            break;

                        case 5:

                            Console.WriteLine($"Điểm TB: {manager.GetAverageScore():F2}");

                            break;

                        case 6:

                            HandleFindMinMaxScore(manager);

                            break;

                        case 7:

                            HandleFindStudent(manager);

                            break;

                        case 0:

                            running = false;

                            break;

                    }

                }

                catch (Exception ex)

                {

                    Console.WriteLine($"❌ Lỗi: {ex.Message}");

                }

            }

        }

        static void HandleAddStudent(StudentManager manager)

        {

            Console.Write("ID: ");

            string id = Console.ReadLine() ?? "";

            Console.Write("Tên: ");

            string name = Console.ReadLine() ?? "";

            Console.Write("Điểm: ");

            string scoreInput = Console.ReadLine() ?? "";

            if (!double.TryParse(scoreInput, out double score))

                throw new InvalidStudentDataException("Điểm không hợp lệ.");

            manager.AddStudent(id, name, score);

        }

        static void HandleRemoveStudent(StudentManager manager)

        {

            Console.Write("Nhập ID: ");

            string id = Console.ReadLine() ?? "";

            manager.RemoveStudent(id);

        }

        static void HandleUpdateScore(StudentManager manager)

        {

            Console.Write("Nhập ID: ");

            string id = Console.ReadLine() ?? "";

            Console.Write("Điểm mới: ");

            string scoreInput = Console.ReadLine() ?? "";

            if (!double.TryParse(scoreInput, out double score))

                throw new InvalidStudentDataException("Điểm không hợp lệ.");

            manager.UpdateScore(id, score);

        }

        static void HandleFindMinMaxScore(StudentManager manager)

        {

            Console.WriteLine($"Max: {manager.GetMaxScore():F2}");

            Console.WriteLine($"Min: {manager.GetMinScore():F2}");

            Student maxStudent = manager.GetStudentWithMaxScore();

            Student minStudent = manager.GetStudentWithMinScore();

            Console.WriteLine($"SV cao nhất: {maxStudent.Name} ({maxStudent.StudentId})");

            Console.WriteLine($"SV thấp nhất: {minStudent.Name} ({minStudent.StudentId})");

        }

        static void HandleFindStudent(StudentManager manager)

        {

            Console.Write("Nhập ID: ");

            string id = Console.ReadLine() ?? "";

            Student? student = manager.FindStudentById(id);

            if (student != null)

                student.Display();

            else

                Console.WriteLine("Không tìm thấy sinh viên.");

        }

    }

}

### 🤔 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?**

**Trả Lời:**

**Kiểm tra dữ liệu đầu vào (validation).**

**Nếu sai → thông báo lỗi và yêu cầu nhập lại.**

**Tránh để chương trình bị lỗi hoặc cho ra kết quả sai.**

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

Trả Lời:

Có thể dùng List thay vì array không? Ưu điểm gì?

Có, nên dùng List.

Ưu điểm của List so với array:

+ Tự động thay đổi kích thước.

+ Dễ thêm / xóa / sửa phần tử.

+ Nhiều hàm hỗ trợ sẵn (Add, Remove, Find…).

+ Code linh hoạt và dễ bảo trì hơn.

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

**Trả Lời:**

**Ghi dữ liệu ra file (txt, csv, json…).**

**Khi chạy lại chương trình → đọc file lên.**

### 💫 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% |