# PHIẾU HỌC TẬP CHỦ ĐỘNG (PHT) - C# CĂN BẢN (TIẾP THEO)

**Môn học:** FIT4016: Thiết kế, Lập trình Back-End  
**Học phần:** C# Căn Bản (Chương 1 - Tổng Quan Ngôn Ngữ C#) - Phần 2

## PHIẾU HỌC TẬP C# [07] - DELEGATE, ANONYMOUS METHOD VÀ LAMBDA EXPRESSION

**Họ và tên:Trần Minh Quân**  
**Lớp: CNTT 18-01**  
**Video đã xem (Link):** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

### **1. Kiến thức cốt lõi**

#### **1.1 Delegate là gì?**

**Delegate:** Là một kiểu dữ liệu đại diện cho một hàm.

// Khai báo delegate

public delegate int MathOperation(int a, int b);

// Tạo hàm phù hợp

public static int Add(int a, int b)

{

return a + b;

}

public static int Multiply(int a, int b)

{

return a \* b;

}

// Sử dụng delegate

MathOperation operation = Add;

int result = operation(5, 3); // Gọi Add(5, 3) = 8

operation = Multiply;

result = operation(5, 3); // Gọi Multiply(5, 3) = 15

#### **1.2 Anonymous Method**

// Khai báo delegate

delegate int Calculate(int x, int y);

// Sử dụng anonymous method

Calculate calc = delegate(int x, int y)

{

return x + y;

};

int result = calc(10, 20); // 30

#### **1.3 Lambda Expression**

**Lambda:** Cách viết ngắn gọn hơn của anonymous method.

// Cú pháp: (tham số) => biểu thức

// Ví dụ 1: Một tham số

delegate int Double(int x);

Double doubleFunc = x => x \* 2;

Console.WriteLine(doubleFunc(5)); // 10

// Ví dụ 2: Hai tham số

delegate int Add(int a, int b);

Add addFunc = (a, b) => a + b;

Console.WriteLine(addFunc(5, 3)); // 8

// Ví dụ 3: Có body

Func<int, int, int> multiply = (a, b) =>

{

int result = a \* b;

return result;

};

Console.WriteLine(multiply(5, 3)); // 15

// Ví dụ 4: Không tham số

Action greet = () => Console.WriteLine("Xin chào!");

greet(); // In: Xin chào!

// Ví dụ 5: Với LINQ

List<int> numbers = new() { 1, 2, 3, 4, 5 };

var evens = numbers.Where(n => n % 2 == 0).ToList(); // [2, 4]

var squares = numbers.Select(n => n \* n).ToList(); // [1, 4, 9, 16, 25]

#### **1.4 Delegate vs Action vs Func**

| **Loại** | **Khai báo** | **Trả về** | **Ví dụ** |
| --- | --- | --- | --- |
| **Delegate tự định nghĩa** | delegate int MyDelegate(int); | Tùy ý | MyDelegate f = x => x \* 2; |
| \*\*Action\*\* | Action<int> action; | void | Action<int> print = x => Console.WriteLine(x); |
| **Func<T, TResult>** | Func<int, int> func; | TResult | Func<int, int> square = x => x \* x; |
| \*\*Predicate\*\* | Predicate<int> check; | bool | Predicate<int> isEven = n => n % 2 == 0; |

### **2. Ví dụ thực hành (Bắt buộc)**

**Kịch bản:** Xử lý danh sách sinh viên với các hàm lambda.

#### **2.1 Code Khởi đầu:**

using System;

using System.Collections.Generic;

using System.Linq;

class Program

{

// TODO 1: Khai báo delegate kiểm tra điều kiện

// delegate bool KiemTra(double diem);

// TODO 2: Khai báo delegate xử lý dữ liệu

// delegate void XuLy(string ten, double diem);

// TODO 3: Viết hàm LocSinhVien

// - Tham số: List<(string, double)>, Predicate

// - Lọc sinh viên theo điều kiện

// - Trả về: List<(string, double)> kết quả

// (Viết code của bạn tại đây)

// TODO 1: Khai báo delegate kiểm tra điều kiện (trả về bool)

    delegate bool KiemTra(double diem);

    // TODO 2: Khai báo delegate xử lý dữ liệu (trả về void - in ấn)

    delegate void XuLy(string ten, double diem);

    // TODO 3: Viết hàm LocSinhVien

    // Nhận vào danh sách và một hàm điều kiện (Predicate)

    static List<(string, double)> LocSinhVien(List<(string, double)> list, Predicate<(string, double)> condition)

    {

        // Cách viết ngắn gọn dùng LINQ

        return list.Where(x => condition(x)).ToList();

    }

// TODO 4: Viết hàm DuyetDanhSach

// - Tham số: List<(string, double)>, Action

// - Duyệt từng sinh viên, gọi action

// - Mục đích: in, tính toán, v.v.

// (Viết code của bạn tại đây)

    // TODO 4: Viết hàm DuyetDanhSach

    // Nhận vào danh sách và một hành động (Action) để thực thi cho từng phần tử

    static void DuyetDanhSach(List<(string, double)> list, Action<(string, double)> action)

    {

        foreach (var item in list)

        {

            action(item);

        }

    }

// TODO 5: Viết hàm TinhDiemTrungBinh

// - Tham số: List<double> (danh sách điểm)

// - Sử dụng lambda để cộng các điểm

// - Trả về: double (điểm TB)

// (Viết code của bạn tại đây)

// TODO 5: Viết hàm TinhDiemTrungBinh

    static double TinhDiemTrungBinh(List<double> dsDiem)

    {

        return dsDiem.Average();

    }

    static void Main()

    {

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

        Console.WriteLine("=== Delegate, Lambda, và LINQ ===\n");

        // Dữ liệu mẫu

        List<(string, double)> danhSach = new()

        {

            ("Nguyễn Văn A", 8.5),

            ("Trần Thị B", 7.2),

            ("Lê Văn C", 5.8),

            ("Phạm Thị D", 9.0),

            ("Hoàng Văn E", 6.5)

        };

static void Main()

{

Console.WriteLine("=== Delegate, Lambda, và LINQ ===\n");

// Dữ liệu: (Tên, Điểm)

List<(string, double)> danhSach = new()

{

("Nguyễn Văn A", 8.5),

("Trần Thị B", 7.2),

("Lê Văn C", 5.8),

("Phạm Thị D", 9.0),

("Hoàng Văn E", 6.5)

};

Console.WriteLine("=== Danh sách gốc ===");

// TODO 6: In toàn bộ danh sách

// Dùng foreach hoặc hàm DuyetDanhSach

foreach (var (ten, diem) in danhSach)

{

Console.WriteLine($"{ten,-20} │ {diem:F2}");

}

// TODO 7: Lọc sinh viên điểm >= 7.0

// Sử dụng lambda: n => n.Item2 >= 7.0

var gioi = /\* Gọi hàm LocSinhVien \*/;

Console.WriteLine("\n=== Sinh viên loại Giỏi (>= 7.0) ===");

foreach (var (ten, diem) in gioi)

{

Console.WriteLine($"{ten,-20} │ {diem:F2}");

}

// TODO 8: Lọc sinh viên điểm < 6.0

// Sử dụng lambda: n => n.Item2 < 6.0

var yeu = /\* Gọi hàm LocSinhVien \*/;

Console.WriteLine("\n=== Sinh viên loại Yếu (< 6.0) ===");

foreach (var (ten, diem) in yeu)

{

Console.WriteLine($"{ten,-20} │ {diem:F2}");

}

// TODO 9: Sắp xếp theo điểm giảm dần

// Sử dụng LINQ: .OrderByDescending(n => n.Item2)

var sapXep = /\* Thực hiện sắp xếp \*/;

Console.WriteLine("\n=== Xếp theo điểm giảm dần ===");

foreach (var (ten, diem) in sapXep)

{

Console.WriteLine($"{ten,-20} │ {diem:F2}");

}

// TODO 10: Lấy top 2 sinh viên

// Sử dụng LINQ: .Take(2)

var top2 = /\* Thực hiện \*/;

Console.WriteLine("\n=== Top 2 Sinh Viên ===");

foreach (var (ten, diem) in top2)

{

Console.WriteLine($"{ten,-20} │ {diem:F2}");

}

// TODO 11: Tính điểm trung bình

// Sử dụng lambda và Average()

double diemTB = /\* Gọi hàm hoặc LINQ \*/;

Console.WriteLine($"\n=== Điểm Trung Bình: {diemTB:F2} ===");

// TODO 12: Tìm sinh viên có điểm cao nhất

// Sử dụng LINQ: .MaxBy(n => n.Item2)

var topDiem = /\* Thực hiện \*/;

Console.WriteLine($"\n=== Sinh viên có điểm cao nhất: {topDiem.Item1} ({topDiem.Item2:F2}) ===");

}

}

#### **2.2 Ví dụ hoàn chỉnh**

**Hàm TODO 3: LocSinhVien**

public static List<(string, double)> LocSinhVien(

List<(string, double)> danhSach,

Predicate<(string, double)> condition)

{

List<(string, double)> ketqua = new();

foreach (var item in danhSach)

{

if (condition(item))

ketqua.Add(item);

}

return ketqua;

}

// Cách sử dụng:

var gioi = LocSinhVien(danhSach, sv => sv.Item2 >= 7.0);

**Hàm TODO 5: Tính TB với Lambda**

Func<List<double>, double> tinhTrungBinh = (diem) =>

{

double tong = 0;

foreach (double d in diem)

tong += d;

return tong / diem.Count;

};

// Hoặc dùng LINQ:

double diemTB = danhSach.Select(sv => sv.Item2).Average();

**Các LINQ Query thường dùng:**

// Lọc

var filtered = danhSach.Where(sv => sv.Item2 >= 7.0).ToList();

// Sắp xếp

var sorted = danhSach.OrderByDescending(sv => sv.Item2).ToList();

// Lấy n phần tử

var topN = danhSach.Take(2).ToList();

// Bỏ n phần tử

var skip = danhSach.Skip(2).ToList();

// Chuyển đổi

var names = danhSach.Select(sv => sv.Item1).ToList();

// Nhóm

var grouped = danhSach.GroupBy(sv => Math.Floor(sv.Item2)).ToList();

### **3. Yêu cầu Bằng chứng (Proof of Work)**

#### **A. Code đã hoàn thiện**

// (Dán code đầy đủ của bạn)

using System;

using System.Collections.Generic;

using System.Linq;

class Program

{

    // TODO 1: Khai báo delegate kiểm tra điều kiện (trả về bool)

    delegate bool KiemTra(double diem);

    // TODO 2: Khai báo delegate xử lý dữ liệu (trả về void - in ấn)

    delegate void XuLy(string ten, double diem);

    // TODO 3: Viết hàm LocSinhVien

    // Nhận vào danh sách và một hàm điều kiện (Predicate)

    static List<(string, double)> LocSinhVien(List<(string, double)> list, Predicate<(string, double)> condition)

    {

        // Cách viết ngắn gọn dùng LINQ

        return list.Where(x => condition(x)).ToList();

    }

    // TODO 4: Viết hàm DuyetDanhSach

    // Nhận vào danh sách và một hành động (Action) để thực thi cho từng phần tử

    static void DuyetDanhSach(List<(string, double)> list, Action<(string, double)> action)

    {

        foreach (var item in list)

        {

            action(item);

        }

    }

    // TODO 5: Viết hàm TinhDiemTrungBinh

    static double TinhDiemTrungBinh(List<double> dsDiem)

    {

        return dsDiem.Average();

    }

    static void Main()

    {

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

        Console.WriteLine("=== Delegate, Lambda, và LINQ ===\n");

        // Dữ liệu mẫu

        List<(string, double)> danhSach = new()

        {

            ("Nguyễn Văn A", 8.5),

            ("Trần Thị B", 7.2),

            ("Lê Văn C", 5.8),

            ("Phạm Thị D", 9.0),

            ("Hoàng Văn E", 6.5)

        };

        Console.WriteLine("=== Danh sách gốc ===");

        // TODO 6: In toàn bộ danh sách dùng hàm DuyetDanhSach

        DuyetDanhSach(danhSach, sv => Console.WriteLine($"{sv.Item1,-20} | {sv.Item2:F2}"));

        // TODO 7: Lọc sinh viên điểm >= 7.0 (Loại Giỏi)

        var gioi = LocSinhVien(danhSach, n => n.Item2 >= 7.0);

        Console.WriteLine("\n=== Sinh viên loại Giỏi (>= 7.0) ===");

        foreach (var (ten, diem) in gioi)

        {

            Console.WriteLine($"{ten,-20} | {diem:F2}");

        }

        // TODO 8: Lọc sinh viên điểm < 6.0 (Loại Yếu)

        var yeu = LocSinhVien(danhSach, n => n.Item2 < 6.0);

        Console.WriteLine("\n=== Sinh viên loại Yếu (< 6.0) ===");

        foreach (var (ten, diem) in yeu)

        {

            Console.WriteLine($"{ten,-20} | {diem:F2}");

        }

        // TODO 9: Sắp xếp theo điểm giảm dần

        var sapXep = danhSach.OrderByDescending(n => n.Item2).ToList();

        Console.WriteLine("\n=== Xếp theo điểm giảm dần ===");

        foreach (var (ten, diem) in sapXep)

        {

            Console.WriteLine($"{ten,-20} | {diem:F2}");

        }

        // TODO 10: Lấy top 2 sinh viên

        var top2 = sapXep.Take(2).ToList();

        Console.WriteLine("\n=== Top 2 Sinh Viên ===");

        foreach (var (ten, diem) in top2)

        {

            Console.WriteLine($"{ten,-20} | {diem:F2}");

        }

        // TODO 11: Tính điểm trung bình (Dùng LINQ Average trực tiếp cho nhanh)

        double diemTB = danhSach.Average(n => n.Item2);

        Console.WriteLine($"\n=== Điểm Trung Bình: {diemTB:F2} ===");

        // TODO 12: Tìm sinh viên có điểm cao nhất

        var topDiem = danhSach.MaxBy(n => n.Item2);

        Console.WriteLine($"\n=== Sinh viên có điểm cao nhất: {topDiem.Item1} ({topDiem.Item2:F2}) ===");

        Console.ReadLine();

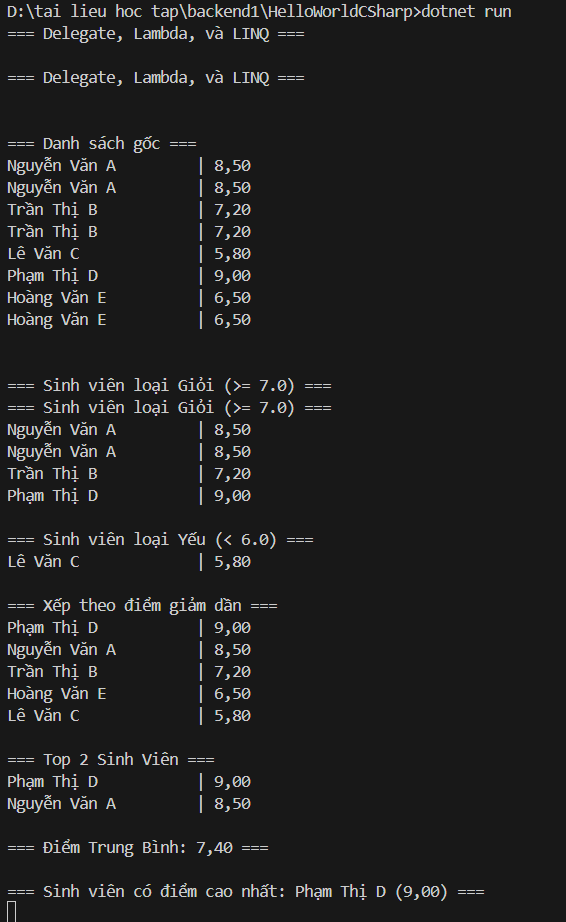
    }

}

#### **B. Ảnh chụp màn hình Kết quả**

* Danh sách gốc
* Lọc sinh viên giỏi
* Lọc sinh viên yếu
* Sắp xếp theo điểm
* Top 2 sinh viên
* Điểm trung bình
* Sinh viên có điểm cao nhất

(Dán ảnh chụp màn hình)



### **4. Câu hỏi Phản biện (Bắt buộc)**

**Một điều tôi chưa hiểu rõ:**

LINQ và Lambda Expression giúp code ngắn gọn và dễ đọc hơn rất nhiều. Tuy nhiên, em thắc mắc liệu việc lạm dụng LINQ có ảnh hưởng đến hiệu năng (performance) của ứng dụng so với việc sử dụng các vòng lặp for hoặc foreach truyền thống không? Đặc biệt là khi xử lý một danh sách dữ liệu cực lớn (hàng triệu bản ghi) thì nên ưu tiên sự ngắn gọn của LINQ hay tốc độ của vòng lặp?

## PHIẾU HỌC TẬP C# [08] - ASYNCHRONOUS PROGRAMMING (ASYNC/AWAIT)

**Họ và tên:** Trần Minh Quân  
**Lớp:** CNTT 18-01  
**Video đã xem (Link):** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

### **1. Kiến thức cốt lõi**

#### **1.1 Khái niệm Async/Await**

**Async/Await:** Cho phép chương trình chạy các tác vụ nặng mà không block (chặn) giao diện.

// Hàm bình thường (chặn)

public static void CongTac1()

{

System.Threading.Thread.Sleep(5000); // Chờ 5 giây

Console.WriteLine("Xong!");

// Trong 5 giây này, chương trình bị chặn, không làm gì được

}

// Hàm async (không chặn)

public static async Task CongTacAsync()

{

await Task.Delay(5000); // Chờ 5 giây

Console.WriteLine("Xong!");

// Trong 5 giây này, chương trình vẫn có thể làm việc khác

}

#### \*\*1.2 Task và Task\*\*

// Task: Công việc không trả về giá trị

public async Task XuLyDuLieu()

{

await Task.Delay(1000);

Console.WriteLine("Xử lý xong");

}

// Task<T>: Công việc trả về giá trị

public async Task<int> TinhToan()

{

await Task.Delay(1000);

return 10 + 20; // Trả về 30

}

// Sử dụng

await XuLyDuLieu(); // Chờ xong

int result = await TinhToan(); // Chờ xong, lấy kết quả

#### **1.3 Cú pháp async/await**

// Hàm async phải:

// 1. Khai báo với từ khóa async

// 2. Trả về Task hoặc Task<T>

// 3. Có ít nhất 1 await

public async Task<string> DocFile(string path)

{

// await: chờ tác vụ hoàn thành

string content = await File.ReadAllTextAsync(path);

return content;

}

// Main cũng có thể async (C# 7.1+)

static async Task Main(string[] args)

{

string content = await DocFile("file.txt");

Console.WriteLine(content);

}

#### **1.4 Xử lý lỗi trong async**

public async Task<int> DivideAsync(int a, int b)

{

try

{

if (b == 0)

throw new DivideByZeroException();

await Task.Delay(1000);

return a / b;

}

catch (DivideByZeroException ex)

{

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

return 0;

}

finally

{

Console.WriteLine("Xử lý xong");

}

}

#### **1.5 Bảng so sánh Sync vs Async**

| **Khía cạnh** | **Synchronous** | **Asynchronous** |
| --- | --- | --- |
| **Khai báo** | public void Method() | public async Task Method() |
| **Trả về** | Giá trị trực tiếp | Task / Task |
| **Chờ kết quả** | Gọi thường | await |
| **Khi nào dùng** | Tác vụ nhanh | Tác vụ chậm (I/O, mạng) |
| **Ưu điểm** | Đơn giản | Không chặn giao diện |
| **Nhược điểm** | Chặn giao diện | Phức tạp hơn |

### **2. Ví dụ thực hành (Bắt buộc)**

**Kịch bản:** Tạo chương trình tải dữ liệu từ nhiều nguồn mà không chặn giao diện.

#### **2.1 Code Khởi đầu:**

using System;

using System.Threading.Tasks;

using System.Collections.Generic;

class Program

{

// TODO 1: Viết hàm SimulateDownload

// - Tham số: string url, int delayMs

// - Trả về: Task<string> (dữ liệu tải về)

// - Mục đích: Giả lập tải dữ liệu từ URL

// - Sử dụng: await Task.Delay(delayMs);

// - Trả về: $"Dữ liệu từ {url}"

// (Viết code của bạn tại đây)

// TODO 1: Viết hàm SimulateDownload

    static async Task<string> SimulateDownload(string url, int delayMs)

    {

        await Task.Delay(delayMs); // Giả lập chờ tải mạng

        return $"Dữ liệu từ {url}";

    }

// TODO 2: Viết hàm TaiTuSinhVien

// - Tham số: string maSV

// - Trả về: Task<string> (tên sinh viên)

// - Mục đích: Lấy dữ liệu sinh viên (delay 2 giây)

// (Viết code của bạn tại đây)

 // TODO 2: Viết hàm TaiTuSinhVien (Giả lập delay 2s)

    static async Task<string> TaiTuSinhVien(string maSV)

    {

        await Task.Delay(2000);

        return "Nguyen Van A"; // Trả về tên giả định

    }

// TODO 3: Viết hàm TaiDiem

// - Tham số: string maSV

// - Trả về: Task<double> (điểm)

// - Mục đích: Lấy điểm (delay 1.5 giây)

// (Viết code của bạn tại đây)

// TODO 3: Viết hàm TaiDiem (Giả lập delay 1.5s)

    static async Task<double> TaiDiem(string maSV)

    {

        await Task.Delay(1500);

        return 8.5; // Trả về điểm giả định

    }

// TODO 4: Viết hàm TaiDonVi

// - Tham số: string maSV

// - Trả về: Task<string> (đơn vị)

// - Mục đích: Lấy đơn vị (delay 1 giây)

// (Viết code của bạn tại đây)

// TODO 4: Viết hàm TaiDonVi (Giả lập delay 1s)

    static async Task<string> TaiDonVi(string maSV)

    {

        await Task.Delay(1000);

        return "CNTT";

    }

// TODO 5: Viết hàm LayThongTinSinhVien

// - Tham số: string maSV

// - Trả về: Task<(string, double, string)>

// - Mục đích: Gọi 3 hàm async song song (parallel)

// - Sử dụng: Task.WhenAll() hoặc await từng cái

// (Viết code của bạn tại đây)

 // TODO 5: Viết hàm LayThongTinSinhVien (Chạy song song)

    static async Task<(string, double, string)> LayThongTinSinhVien(string maSV)

    {

        // Khởi chạy 3 tác vụ cùng lúc (không dùng await ngay ở đây)

        var taskTen = TaiTuSinhVien(maSV);

        var taskDiem = TaiDiem(maSV);

        var taskDonVi = TaiDonVi(maSV);

        // Đợi tất cả cùng hoàn thành (Song song)

        await Task.WhenAll(taskTen, taskDiem, taskDonVi);

        // Trả về kết quả khi tất cả đã xong

        return (await taskTen, await taskDiem, await taskDonVi);

    }

    static async Task Main()

    {

        Console.WriteLine("=== Asynchronous Programming ===\n");

static async Task Main()

{

Console.WriteLine("=== Asynchronous Programming ===\n");

// TODO 6: Gọi hàm SimulateDownload 3 lần

// Sử dụng async/await

Console.WriteLine("--- Tải tuần tự ---");

var start1 = DateTime.Now;

// (Viết code gọi 3 lần sequential)

Console.WriteLine($"Thời gian tuần tự: {(DateTime.Now - start1).TotalSeconds:F2}s");

// TODO 7: Gọi hàm SimulateDownload 3 lần song song

// Sử dụng Task.WhenAll()

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

var start2 = DateTime.Now;

// (Viết code gọi 3 lần parallel)

Console.WriteLine($"Thời gian song song: {(DateTime.Now - start2).TotalSeconds:F2}s");

// TODO 8: Gọi LayThongTinSinhVien

Console.WriteLine("\n--- Lấy thông tin sinh viên ---");

var (ten, diem, donvi) = await /\* Gọi hàm \*/;

Console.WriteLine($"Tên: {ten}");

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

Console.WriteLine($"Đơn vị: {donvi}");

}

}

#### **2.2 Ví dụ hoàn chỉnh**

**Hàm TODO 1: SimulateDownload**

public static async Task<string> SimulateDownload(string url, int delayMs)

{

Console.WriteLine($"🔄 Đang tải từ {url}...");

await Task.Delay(delayMs);

Console.WriteLine($"✓ Xong: {url}");

return $"Dữ liệu từ {url}";

}

**Hàm TODO 5: LayThongTinSinhVien (Tuần tự)**

public static async Task<(string, double, string)> LayThongTinSinhVien(string maSV)

{

string ten = await TaiTuSinhVien(maSV);

double diem = await TaiDiem(maSV);

string donvi = await TaiDonVi(maSV);

return (ten, diem, donvi);

}

**Hàm TODO 5: LayThongTinSinhVien (Song song)**

public static async Task<(string, double, string)> LayThongTinSinhVienParallel(string maSV)

{

// Khởi động 3 tác vụ cùng lúc

var tenTask = TaiTuSinhVien(maSV);

var diemTask = TaiDiem(maSV);

var donviTask = TaiDonVi(maSV);

// Chờ tất cả hoàn thành

await Task.WhenAll(tenTask, diemTask, donviTask);

// Lấy kết quả

string ten = await tenTask;

double diem = await diemTask;

string donvi = await donviTask;

return (ten, diem, donvi);

}

**TODO 6 & 7: Tải tuần tự vs Song song**

// Tuần tự (tổng ~4.5 giây)

Console.WriteLine("--- Tải tuần tự ---");

var start1 = DateTime.Now;

string data1 = await SimulateDownload("URL1", 1500);

string data2 = await SimulateDownload("URL2", 1500);

string data3 = await SimulateDownload("URL3", 1500);

Console.WriteLine($"Thời gian: {(DateTime.Now - start1).TotalSeconds:F2}s");

// Song song (tổng ~1.5 giây)

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

var start2 = DateTime.Now;

var tasks = new List<Task<string>>

{

SimulateDownload("URL1", 1500),

SimulateDownload("URL2", 1500),

SimulateDownload("URL3", 1500)

};

string[] results = await Task.WhenAll(tasks);

Console.WriteLine($"Thời gian: {(DateTime.Now - start2).TotalSeconds:F2}s");

### **3. Yêu cầu Bằng chứng (Proof of Work)**

#### **A. Code đã hoàn thiện**

// (Dán code đầy đủ của bạn)

using System;

using System.Threading.Tasks;

using System.Collections.Generic;

class Program

{

    // TODO 1: Viết hàm SimulateDownload

    static async Task<string> SimulateDownload(string url, int delayMs)

    {

        await Task.Delay(delayMs); // Giả lập chờ tải mạng

        return $"Dữ liệu từ {url}";

    }

    // TODO 2: Viết hàm TaiTuSinhVien (Giả lập delay 2s)

    static async Task<string> TaiTuSinhVien(string maSV)

    {

        await Task.Delay(2000);

        return "Nguyen Van A"; // Trả về tên giả định

    }

    // TODO 3: Viết hàm TaiDiem (Giả lập delay 1.5s)

    static async Task<double> TaiDiem(string maSV)

    {

        await Task.Delay(1500);

        return 8.5; // Trả về điểm giả định

    }

    // TODO 4: Viết hàm TaiDonVi (Giả lập delay 1s)

    static async Task<string> TaiDonVi(string maSV)

    {

        await Task.Delay(1000);

        return "CNTT";

    }

    // TODO 5: Viết hàm LayThongTinSinhVien (Chạy song song)

    static async Task<(string, double, string)> LayThongTinSinhVien(string maSV)

    {

        // Khởi chạy 3 tác vụ cùng lúc (không dùng await ngay ở đây)

        var taskTen = TaiTuSinhVien(maSV);

        var taskDiem = TaiDiem(maSV);

        var taskDonVi = TaiDonVi(maSV);

        // Đợi tất cả cùng hoàn thành (Song song)

        await Task.WhenAll(taskTen, taskDiem, taskDonVi);

        // Trả về kết quả khi tất cả đã xong

        return (await taskTen, await taskDiem, await taskDonVi);

    }

    static async Task Main()

    {

        Console.WriteLine("=== Asynchronous Programming ===\n");

        // TODO 6: Gọi hàm SimulateDownload 3 lần (Tuần tự)

        Console.WriteLine("--- Tài tuần tự ---");

        var start1 = DateTime.Now;

        // Dùng await từng dòng -> dòng này xong mới đến dòng kia

        await SimulateDownload("Site A", 1000);

        await SimulateDownload("Site B", 1000);

        await SimulateDownload("Site C", 1000);

        Console.WriteLine($"Thời gian tuần tự: {(DateTime.Now - start1).TotalSeconds:F2}s"); // Sẽ mất khoảng 3s

        // TODO 7: Gọi hàm SimulateDownload 3 lần song song

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

        var start2 = DateTime.Now;

        // Khởi tạo Task nhưng chưa await ngay

        var t1 = SimulateDownload("Site A", 1000);

        var t2 = SimulateDownload("Site B", 1000);

        var t3 = SimulateDownload("Site C", 1000);

        // Đợi cả 3 xong cùng lúc

        await Task.WhenAll(t1, t2, t3);

        Console.WriteLine($"Thời gian song song: {(DateTime.Now - start2).TotalSeconds:F2}s"); // Sẽ chỉ mất khoảng 1s

        // TODO 8: Gọi LayThongTinSinhVien

        Console.WriteLine("\n--- Lấy thông tin sinh viên ---");

        // Gọi hàm và hứng kết quả vào tuple

        var (ten, diem, donvi) = await LayThongTinSinhVien("SV001");

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

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

        Console.WriteLine($"Đơn vị: {donvi}");

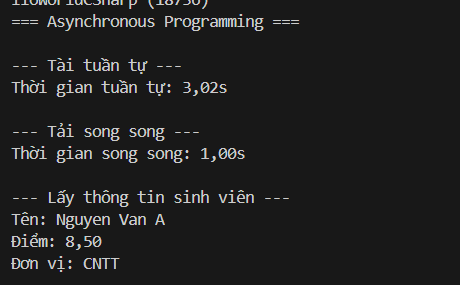
    }

}

#### **B. Ảnh chụp màn hình Kết quả**

* Tải dữ liệu tuần tự (thời gian)
* Tải dữ liệu song song (thời gian)
* Lấy thông tin sinh viên
* Hiển thị kết quả cuối cùng

(Dán ảnh chụp màn hình)



### **4. Câu hỏi Phản biện (Bắt buộc)**

**Một điều tôi chưa hiểu rõ:**

Khi sử dụng Task.WhenAll để chạy 3 tác vụ cùng lúc, nếu một trong các tác vụ bị lỗi (Exception), thì các tác vụ còn lại có tiếp tục chạy không và làm sao để bắt lỗi chính xác của tác vụ đó?

## 📋 TÓM TẮT CHƯƠNG 1 - C# CĂN BẢN

### **Những gì đã học (8 Phiếu)**

| **Phiếu** | **Nội dung** | **Kỹ năng** |
| --- | --- | --- |
| **[01]** | C# là gì, khái niệm cốt lõi | Hiểu ngôn ngữ, cài đặt môi trường |
| **[02]** | Cấu trúc điều khiển, vòng lặp | Kiểm soát luồng chương trình |
| **[03]** | Hàm, mảng | Tổ chức code, xử lý dữ liệu |
| **[04]** | OOP, Class, Object | Lập trình hướng đối tượng |
| **[05]** | Xử lý ngoại lệ | Bắt và xử lý lỗi |
| **[06]** | Collections (List, Dict, Stack) | Quản lý dữ liệu phức tạp |
| **[07]** | Delegate, Lambda, LINQ | Lập trình hàm, truy vấn dữ liệu |
| **[08]** | Async/Await | Lập trình bất đồng bộ |

### **Kiến thức kỳ vọng sau Chương 1**

✅ **Nền tảng C#:**

* Biến, kiểu dữ liệu, hằng
* Cấu trúc điều khiển (if, switch)
* Vòng lặp (for, while, foreach)
* Hàm (khai báo, gọi, tham số)
* Mảng (một chiều, nhiều chiều)

✅ **OOP trong C#:**

* Class, Object, Properties, Methods
* Constructor, Destructor
* Access Modifiers (public, private, protected)
* Encapsulation, Inheritance, Polymorphism

✅ **Collections & LINQ:**

* List, Dictionary, Stack, Queue
* Phương thức: Add, Remove, Find, Sort
* LINQ: Where, Select, OrderBy, Take

✅ **Xử lý lỗi:**

* try...catch...finally
* Exception handling
* Custom Exception

✅ **Delegate & Lambda:**

* Delegate, Anonymous Method
* Lambda Expression (=>)
* Action, Func, Predicate
* Ứng dụng trong LINQ

✅ **Async/Await:**

* Task, Task
* Async method, Await
* Parallel execution
* Error handling trong async

### **Kỹ năng thực hành**

✅ **Thành thạo:**

* Viết chương trình C# từ đơn giản đến phức tạp
* Sử dụng Visual Studio / VS Code
* Debug chương trình
* Đọc và hiểu code
* Giải quyết bài toán lập trình

✅ **Hiểu sâu:**

* Khi nào dùng cấu trúc nào
* Performance: Collections nào tốt nhất
* Async vs Sync
* Design patterns cơ bản

## 📚 TÀI LIỆU THAM KHẢO

**Bắt buộc đọc:**

* Freeman, Adam - "Pro ASP.NET Core 6" (Chương 1-3)
* Tài liệu Khoa cung cấp (Tiếng Việt)

**Tham khảo thêm:**

* Microsoft Docs: <https://docs.microsoft.com/en-us/dotnet/csharp/>
* C# Language Reference: <https://www.w3schools.com/cs/>
* LINQ Tutorial: <https://www.tutorialsteacher.com/linq>
* Async/Await: <https://docs.microsoft.com/en-us/dotnet/csharp/asynchronous-programming>

**Video học:**

* Microsoft Learn C#: <https://learn.microsoft.com/en-us/training/paths/csharp-first-steps/>
* YouTube: C# Tutorial for Beginners

**Chúc bạn hoàn thành xuất sắc Chương 1! 🚀**