# PHIẾU HỌC TẬP C# MODERN [01] - ASYNC/AWAIT - VÉ BẮT BUỘC CHO BACKEND

## Họ và tên: Mai Trọng Thế

## 

## Lớp:CNTT 18-09

## Video đã xem (Link): https://www.youtube.com/watch?v=OZLCVcCB5Gg

## 1. Kiến thức cốt lõi (Tại sao Backend phải dùng Async?)

Vì khi dùng Async nó sẽ giúp hệ thống xử lý nhiều request cùng lúc đưa lại cho ta sự tối ưu về hiệu quả, đặc biệt khi ta chạy dotnet run chờ trả lại data thì dùng async đã giúp chuẩn bị cho các bước sau.

### 1. Async/Await là gì?

* Async/Await là một **pattern** (mẫu lập trình) trong C# để viết code bất đồng bộ (ví dụ: không block thread khi chờ I/O).
* Từ **async** có nghĩa là: hàm bất đồng bộ
* Từ **await** có nghĩa là: Chờ bất đồng bộ cho một tác vụ hoàn thành mà không block thread

### 2. Vấn đề của Synchronous (❌ Cách cũ):

Hãy nhìn vào ví dụ sau:

public List<Product> GetProducts() {

var products = \_db.Products.ToList(); // 2 giây

return products;

}

* Nếu có 1000 requests đến cùng lúc, server cần bao nhiêu threads?
  + A. 100 threads
  + B. 1000 threads
  + C. Tùy mục đích
* Vấn đề là: 1000 threads
* Giải thích: khi ta có 1 request thì sẽ có 1 threads xử lý và khi chạy thì threads sẽ bị delay để đợi database trả dữ liệu và trong khoảng thời gian đó thì threads sẽ không làm được gì và như vậy khi có 1000 request sẽ có 1000 threads xử lý làm cho việc đó làm sever bị treo theo đó là việc chiếm ram, cpu và cạn kiệt threads
* (Nên giải thích tại sao server không thể phục vụ 1000 requests cùng lúc)

### 3. Giải pháp của Asynchronous (✅ Cách mới):

Hãy nhìn vào ví dụ sau:

public async Task<List<Product>> GetProductsAsync() {

var products = await \_db.Products.ToListAsync(); // 2 giây

return products;

}

**Quy tắc Vàng Async/Await:**

* Hàm async phải trả về: Task hoặc Task(T)
* Tên hàm async thường kết thúc bằng: Async
* Phải dùng await trước các hàm async

### 4. Lợi ích chính của Async/Await:

* ✅ Thread được giải phóng trong lúc chờ I/O
* ✅ Có thể xử lý nhiều requests với ít threads hơn
* ✅ Hiệu suất cao, ít dễ crash

## 2. Ví dụ thực hành (Viết hàm async đầu tiên)

*Yêu cầu: Viết code C# (không copy-paste) theo đúng các TODO dưới đây. Sau đó, trình bày lý do tại sao cách viết này tốt hơn.*

### Bài tập 1: Viết hàm GetProductsAsync() đơn giản

**TODO 1:** Khai báo một hàm async có tên là GetProductsAsync() trả về Task<List<Product>>

// Hãy viết hàm của bạn ở đây:

public async Task<List<Product>> GetProductsAsync()

{

await Task.Delay(1000);

return new List<Product>

{

new Product { Id = 1, Name = "Laptop", Price = 1500 },

new Product { Id = 2, Name = "Mouse", Price = 25 },

new Product { Id = 3, Name = "Keyboard", Price = 45 }

};

}

**Giải thích:** Tại sao hàm này tốt hơn cách viết synchronous?

* Không block luồng chính
* Hiệu năng tốt hơn
* Quan trọng với UI/API/Web

### Bài tập 2: Viết hàm GetUserByIdAsync() với await

**TODO 2:** Viết một hàm async nhận vào int userId và trả về Task<User?>. Bên trong hàm, dùng await để gọi \_db.Users.FindAsync(userId)

public async Task<User?> GetUserByIdAsync(int userId)

{

return await \_db.Users.FindAsync(userId);

}

### Bài tập 3: Gọi hàm async từ Controller

**TODO 3:** Hãy điền vào chỗ trống trong đoạn code Controller sau:

[ApiController]

[Route("api/[controller]")]

public class ProductsController : ControllerBase

{

private readonly AppDbContext \_db;

public ProductsController(AppDbContext db)

{

\_db = db;

}

[HttpGet]

public async Task<IActionResult> GetProducts()

{

var products = await \_db.Products.ToListAsync();

return Ok(products);

}

}

## 3. Câu hỏi bắt buộc (Sai lầm thường gặp)

### Lỗi 1: Dùng .Result

Hãy giải thích tại sao đoạn code sau là **SAI**:

var products = GetProductsAsync().Result; // ❌ Deadlock!

**Trả lời:**

* Vấn đề là: .Result chặn (block) thread hiện tại và chờ Task hoàn thành
* Giải pháp là dùng: await GetProductsAsync() thay vì .Result

### Lỗi 2: Quên await

Hãy giải thích tại sao đoạn code sau là **SAI**:

public async Task<List<Product>> GetProductsAsync() {

var products = \_db.Products.ToListAsync(); // ❌ Lỗi!

return products;

}

**Trả lời:**

* Vấn đề là: ToListAsync() trả về Task<List<Product>> nhưng code lại gán trực tiếp cho biến và trả về như List<Product>, gây sai kiểu dữ liệu và chưa chờ Task hoàn thành.
* Để sửa lỗi, bạn cần: await trước .ToListAsync()

## 4. Thực hành tình huống thực tế

*Hãy trả lời các câu hỏi sau dựa trên hiểu biết của bạn:*

### Câu hỏi 1: Khi nào nên dùng async/await?

* A. Khi làm việc với Database (EF Core, .ToListAsync(), .FindAsync())
* B. Khi gọi External API (HttpClient.GetStringAsync())
* C. Khi đợi file (File.ReadAsync(), StreamReader.ReadLineAsync())
* D. Tất cả các trường hợp trên

**Trả lời:** D vì:

async/await nên dùng cho I/O-bound operations, tức là các thao tác phải chờ đợi:

* A. Database (EF Core)  
   → Chờ DB trả dữ liệu → không nên block thread
* B. External API  
   → Chờ network response → async giúp server xử lý request khác
* C. File / Stream  
   → Chờ đọc/ghi file → async giúp UI/Web không bị đơ

### Câu hỏi 2: Có thể viết hàm async mà không cần await bên trong không?

public async Task<string> GetNameAsync() {

return "Hùng"; // Không có await

}

**Trả lời:** Yes, vì:

* Compiler sẽ tự động bọc giá trị trả về thành Task.FromResult("Hùng")
* Code vẫn chạy đúng, không lỗi

### Câu hỏi 3: Thuộc tính IAsyncEnumerablelà gì?

**Trả lời:**

* IAsyncEnumerablecho phép bạn: Duyệt dữ liệu bất đồng bộ từng phần tử, sử dụng await foreach, không cần tải toàn bộ dữ liệu vào bộ nhớ cùng lúc.

## 5. Một điều tôi chưa hiểu rõ hoặc muốn hỏi thêm

**(Bắt buộc phải đặt ít nhất 1 câu hỏi)**

Ví dụ câu hỏi tốt:

* "Em gọi GetProductsAsync() nhưng quên await, thì code sẽ chạy không? Hay là báo lỗi?"
* "Khi nào em dùng async/await, khi nào không?"
* "Sự khác biệt giữa Taskvà Task là gì?"

**Câu hỏi của tôi:**

Nếu em gọi một hàm async nhưng quên dùng await, ví dụ:

var products = GetProductsAsync();

thì chương trình sẽ chạy như thế nào?

* Có bị lỗi compile không?
* Khi nào Task đó thực sự được thực thi?
* Điều gì sẽ xảy ra nếu em truy cập products.Result sau đó?

# PHIẾU HỌC TẬP C# MODERN [02] - LINQ - CÔNG CỤ QUYỀN LỰC XỬ LÝ DỮ LIỆU

## Họ và tên: Mai Trọng Thế

## Lớp: CNTT 18 - 09

## Video đã xem (Link): https://www.youtube.com/watch?v=fAynoHX4PRM

## 1. Kiến thức cốt lõi (Giải phẫu LINQ)

*Sau khi xem video, hãy điền vào các chỗ trống dưới đây:*

### 1. LINQ là gì?

* LINQ = Language Integrated Query
* LINQ giống như một **công cụ tìm kiếm** cho Collection, giúp bạn:
  + **Truy vấn dữ liệu** từ Collection (List, Array, Dictionary, …)
  + **Lọc (filter)** dữ liệu theo điều kiện
  + **Sắp xếp (sort)** dữ liệu
  + **Nhóm (group)** dữ liệu
  + **Chọn (select)** dữ liệu cần thiết

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

### 2. So sánh: Vòng lặp vs LINQ

**Cách cũ (Vòng lặp for/foreach):**

List<Product> expensive = new List<Product>();

foreach (var p in products) {

if (p.Price > 1000) {

expensive.Add(p);

}

}

**Cách mới (LINQ):**

var expensive = products.Where(p => p.Price > 1000).ToList();

* Cách mới có ít dòng code hơn cách cũ
* Cách mới dễ dễ đọc hơn cách cũ

### 3. Các hàm LINQ quan trọng - Phần 1:

| **Hàm LINQ** | **Mục đích sử dụng (Dùng để làm gì?)** |
| --- | --- |
| **Where()** | (Tự điền: Dùng để) lọc dữ liệu theo điều kiện |
| **Select()** | (Tự điền: Dùng để) chọn hoặc chuyển đổi dữ liệu sang dạng khác |
| **OrderBy() / OrderByDescending()** | (Tự điền: Dùng để) sắp xếp dữ liệu tăng dần / giảm dần |
| **Take()** | (Tự điền: Dùng để) lấy N phần tử đầu tiên |
| **Skip()** | (Tự điền: Dùng để) bỏ qua N phần tử đầu tiên |

### 4. Các hàm LINQ quan trọng - Phần 2:

| **Hàm LINQ** | **Mục đích sử dụng** |
| --- | --- |
| **Count()** | (Tự điền: Dùng để) đếm số phần tử |
| **Any()** | (Tự điền: Dùng để) kiểm tra có ít nhất một phần tử thỏa điều kiện hay không |
| **All()** | (Tự điền: Dùng để) kiểm tra tất cả phần tử có thỏa điều kiện hay không |
| **First() / FirstOrDefault()** | (Tự điền: Dùng để) lấy phần tử đầu tiên (hoặc null/giá trị mặc định nếu không có) |
| **GroupBy()** | (Tự điền: Dùng để) nhóm dữ liệu theo một thuộc tính |

## 2. Ví dụ thực hành (Viết LINQ đầu tiên)

*Yêu cầu: Cho danh sách sản phẩm như sau, hãy viết LINQ để trả lời các câu hỏi.*

var products = new List<Product> {

new Product { Id = 1, Name = "Laptop", Price = 1500, CategoryId = 1, Stock = 10 },

new Product { Id = 2, Name = "Mouse", Price = 25, CategoryId = 2, Stock = 50 },

new Product { Id = 3, Name = "Keyboard", Price = 75, CategoryId = 2, Stock = 0 },

new Product { Id = 4, Name = "Monitor", Price = 300, CategoryId = 1, Stock = 5 }

};

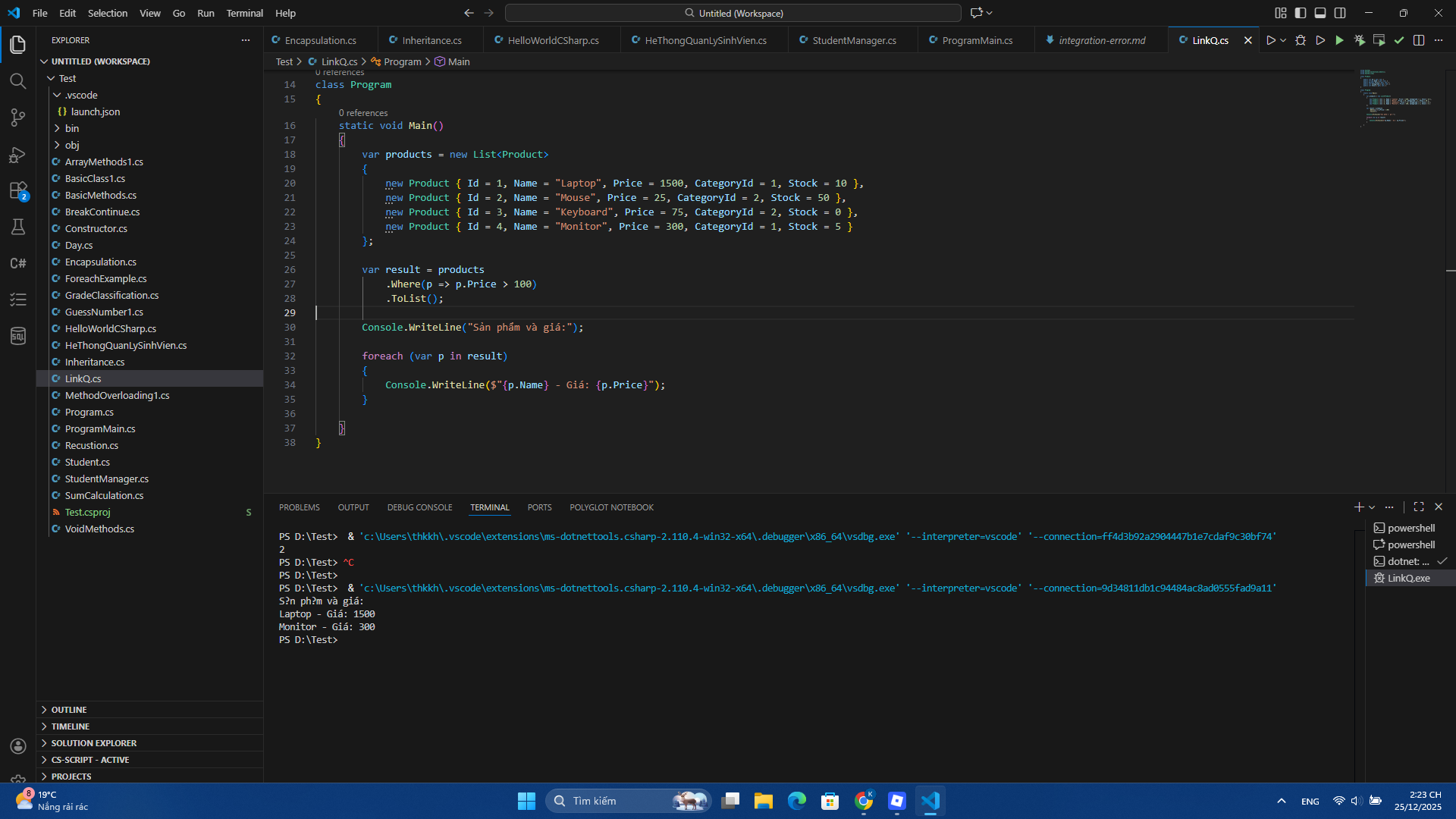
### Bài tập 1: Lấy tất cả sản phẩm có giá > 100

**Yêu cầu:** Viết LINQ sử dụng Where()

var result = products

.Where(p => p.Price > 100)

.ToList();

**Kết quả dự kiến:** 

### Bài tập 2: Lấy chỉ tên và giá của tất cả sản phẩm

**Yêu cầu:** Viết LINQ sử dụng Select()

var result = products

.Select(p => new

{

p.Name,

p.Price

})

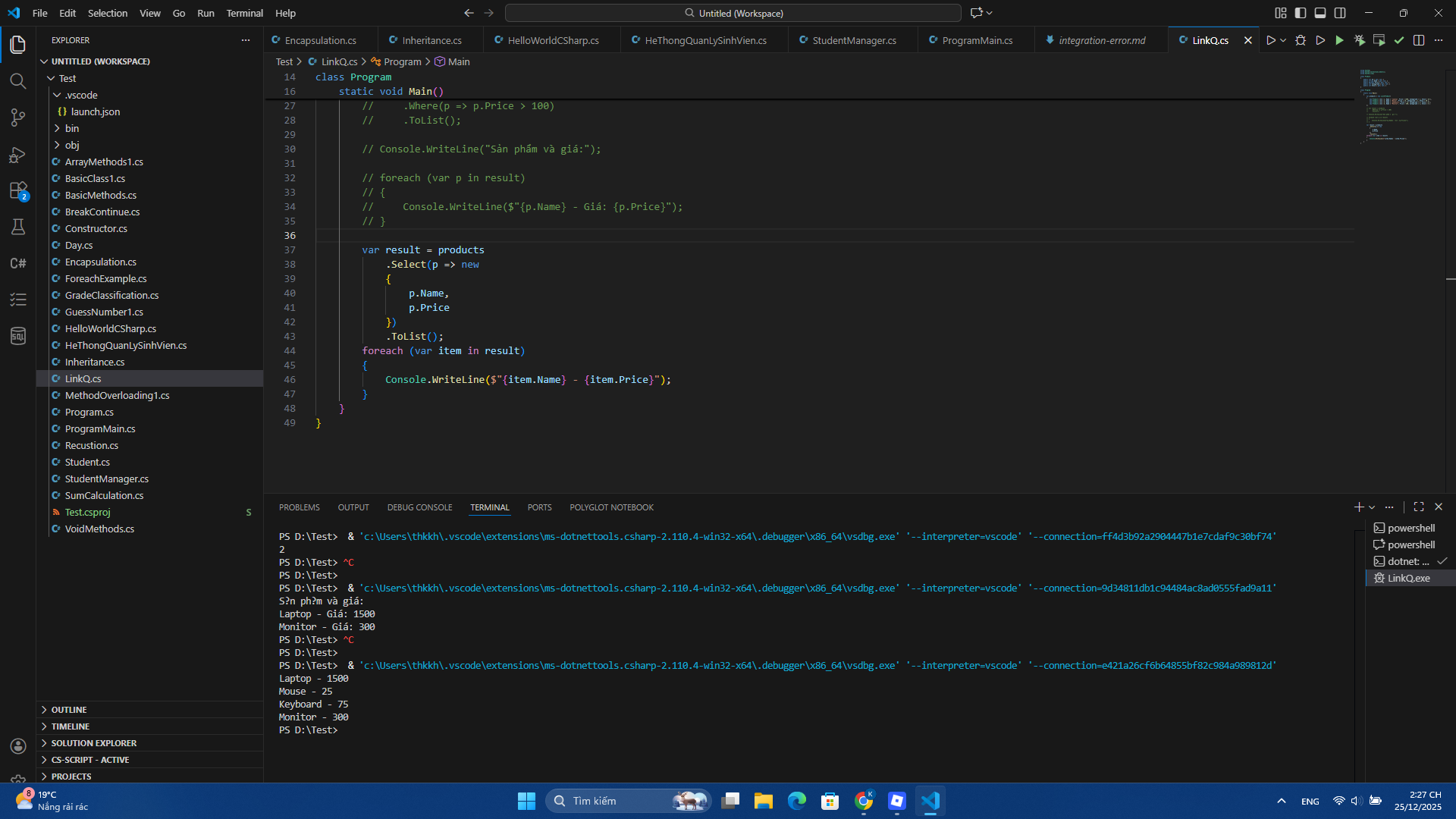
.ToList();

foreach (var item in result)

{

Console.WriteLine($"{item.Name} - {item.Price}");

}

**Kết quả dự kiến:** 

### Bài tập 3: Lấy 2 sản phẩm đắt tiền nhất

**Yêu cầu:** Sắp xếp giảm dần theo giá, rồi lấy 2 sản phẩm đầu tiên

var result = products

.OrderByDescending(p => p.Price)

.Take(2)

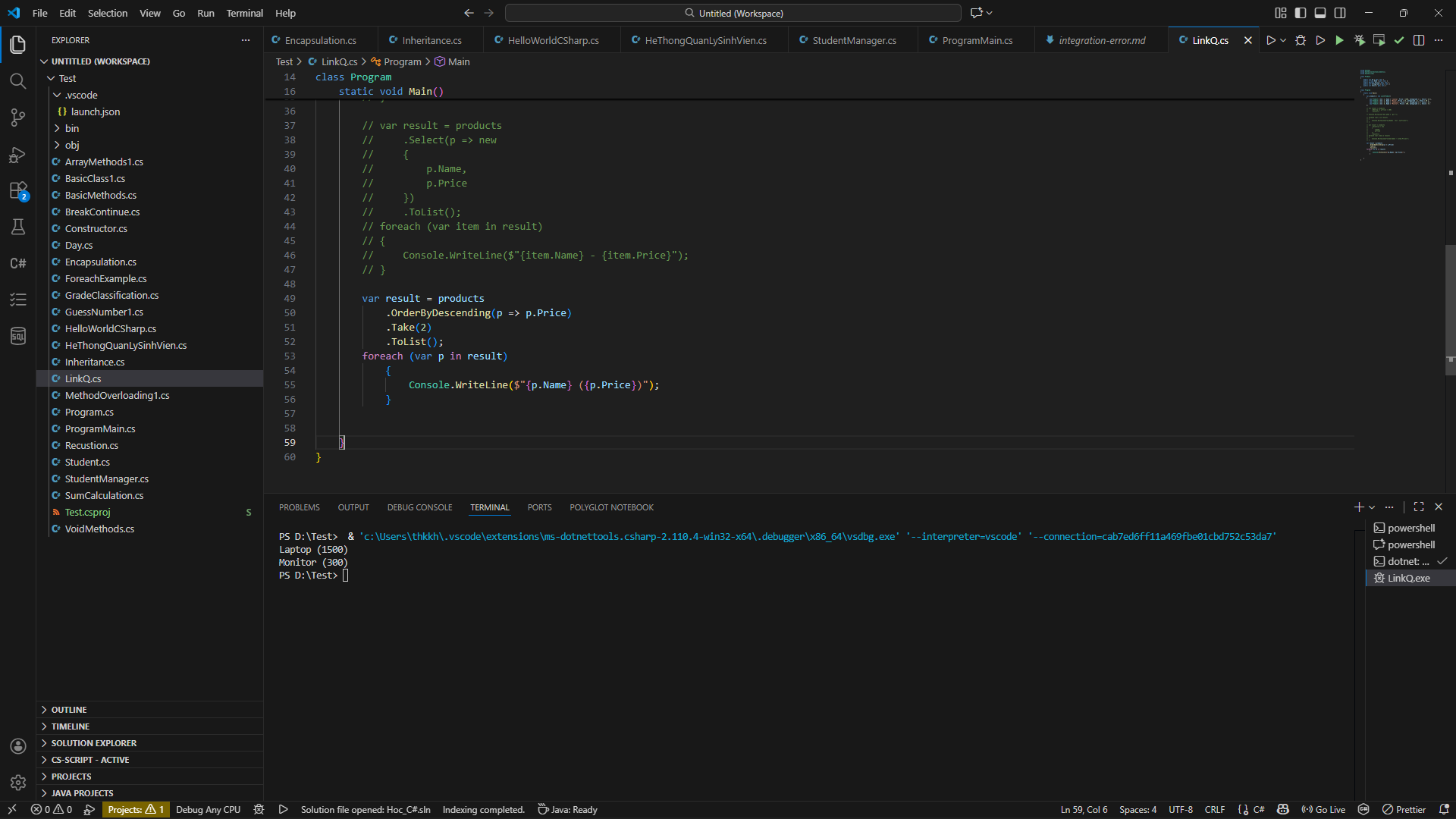
.ToList();

foreach (var p in result)

{

Console.WriteLine($"{p.Name} ({p.Price})");

}

**Kết quả dự kiến:** 

### Bài tập 4: Phân trang - Lấy trang thứ 2, mỗi trang 2 sản phẩm

**Yêu cầu:** Sắp xếp theo Id, dùng Skip() và Take() để lấy trang 2

int pageNumber = 2;

int pageSize = 2;

var result = products

.OrderBy(p => p.Id)

.Skip((pageNumber - 1) \* pageSize)

.Take(pageSize)

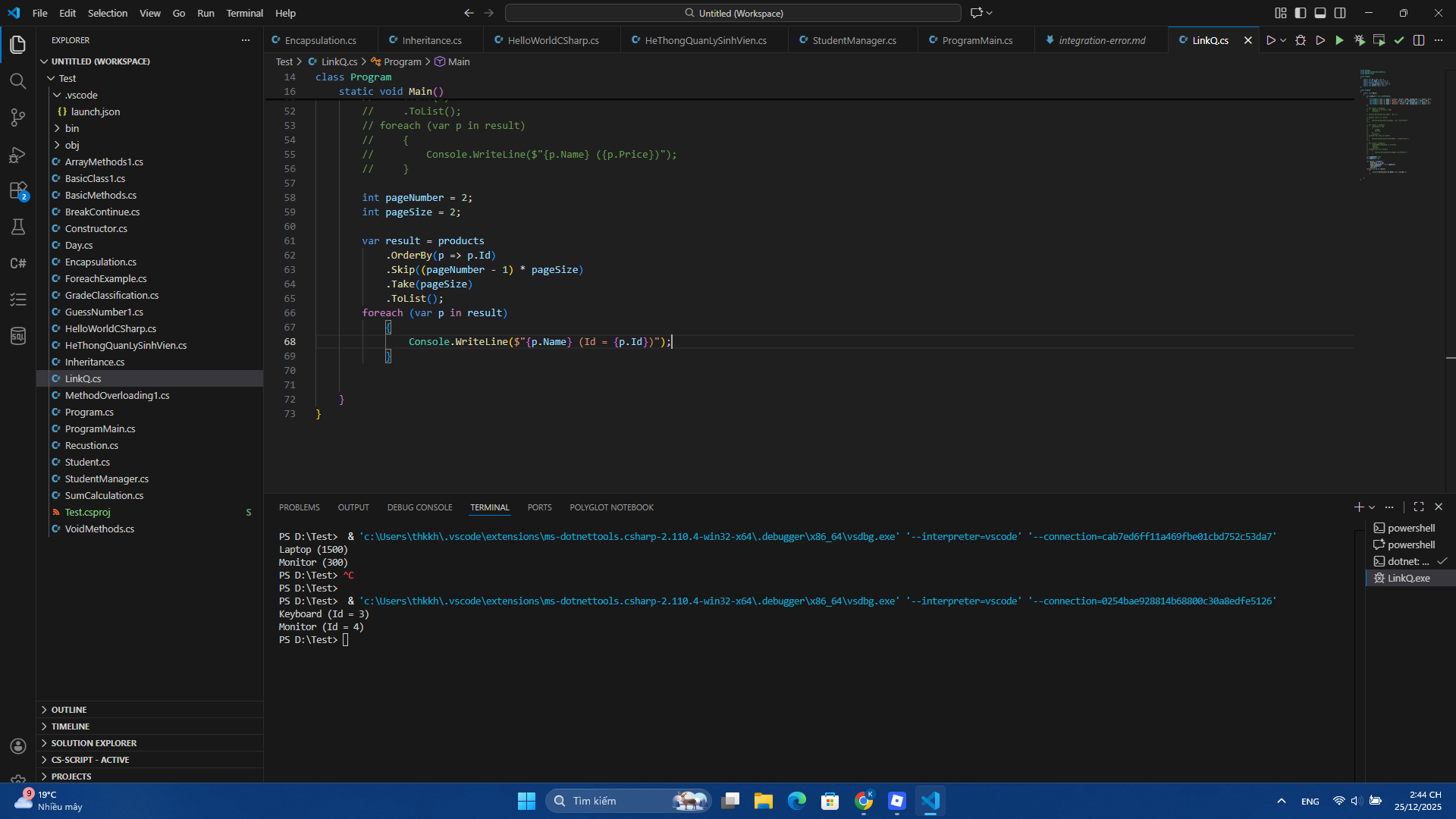
.ToList();

foreach (var p in result)

{

Console.WriteLine($"{p.Name} (Id = {p.Id})");

}

**Kết quả dự kiến:** 

### Bài tập 5: Đếm và kiểm tra

**TODO 1:** Đếm số sản phẩm còn hàng (Stock > 0)

int countInStock = products.Count(p => p.Stock > 0);

**TODO 2:** Kiểm tra có sản phẩm nào giá < 50 không?

bool hasCheapProduct = products.Any(p => p.Price < 50);

**TODO 3:** Kiểm tra có phải tất cả sản phẩm đều có giá > 0 không?

bool allValidPrice = products.All(p => p.Price > 0);

* Kết quả: 

## 3. Câu hỏi bắt buộc (GroupBy & Join)

### Câu hỏi 1: GroupBy - Nhóm dữ liệu

Hãy viết LINQ để **nhóm sản phẩm theo CategoryId** và lấy:

* Mã category
* Số lượng sản phẩm trong category đó
* Tổng giá trị hàng hóa (Price × Stock)

var groupResult = products

.GroupBy(p => p.CategoryId)

.Select(g => new

{

CategoryId = g.Key,

ProductCount = g.Count(),

TotalValue = g.Sum(p => p.Price \* p.Stock)

})

.ToList();

foreach (var item in groupResult)

{

Console.WriteLine(

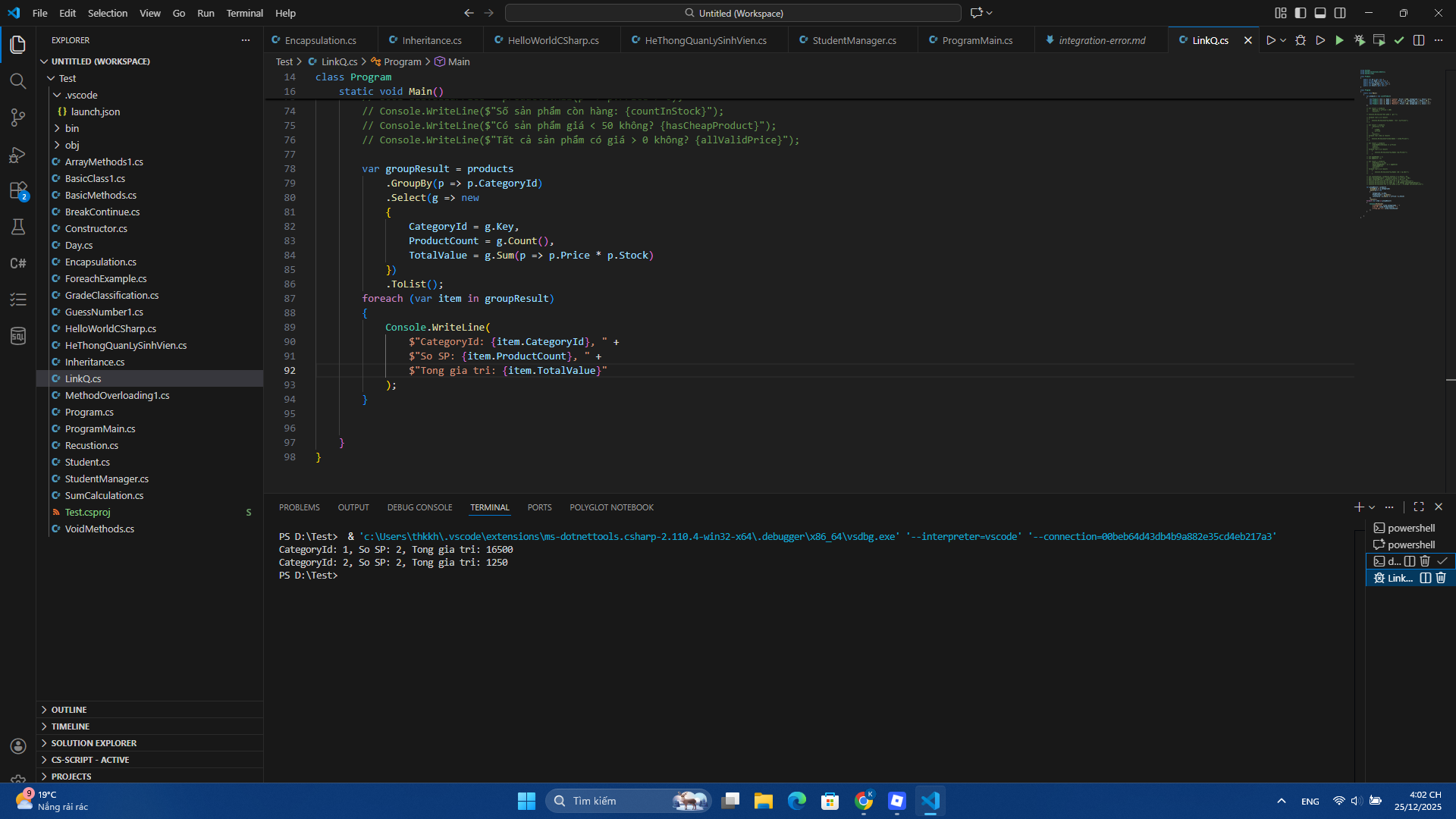
$"CategoryId: {item.CategoryId}, " +

$"Số SP: {item.ProductCount}, " +

$"Tổng giá trị: {item.TotalValue}"

);

}

* Kết quả: 

### Câu hỏi 2: Join - Kết nối 2 bảng

Giả sử bạn có một bảng Categories:

var categories = new List<Category> {

new Category { Id = 1, Name = "Điện tử" },

new Category { Id = 2, Name = "Phụ kiện" }

};

Hãy viết LINQ để **JOIN** products với categories để lấy:

* Tên sản phẩm
* Tên category
* Giá

// Hãy viết code của bạn ở đây:

var joinResult = products

.Join(

categories,

p => p.CategoryId,

c => c.Id,

(p, c) => new

{

ProductName = p.Name,

CategoryName = c.Name,

p.Price

}

)

.ToList();

foreach (var item in joinResult)

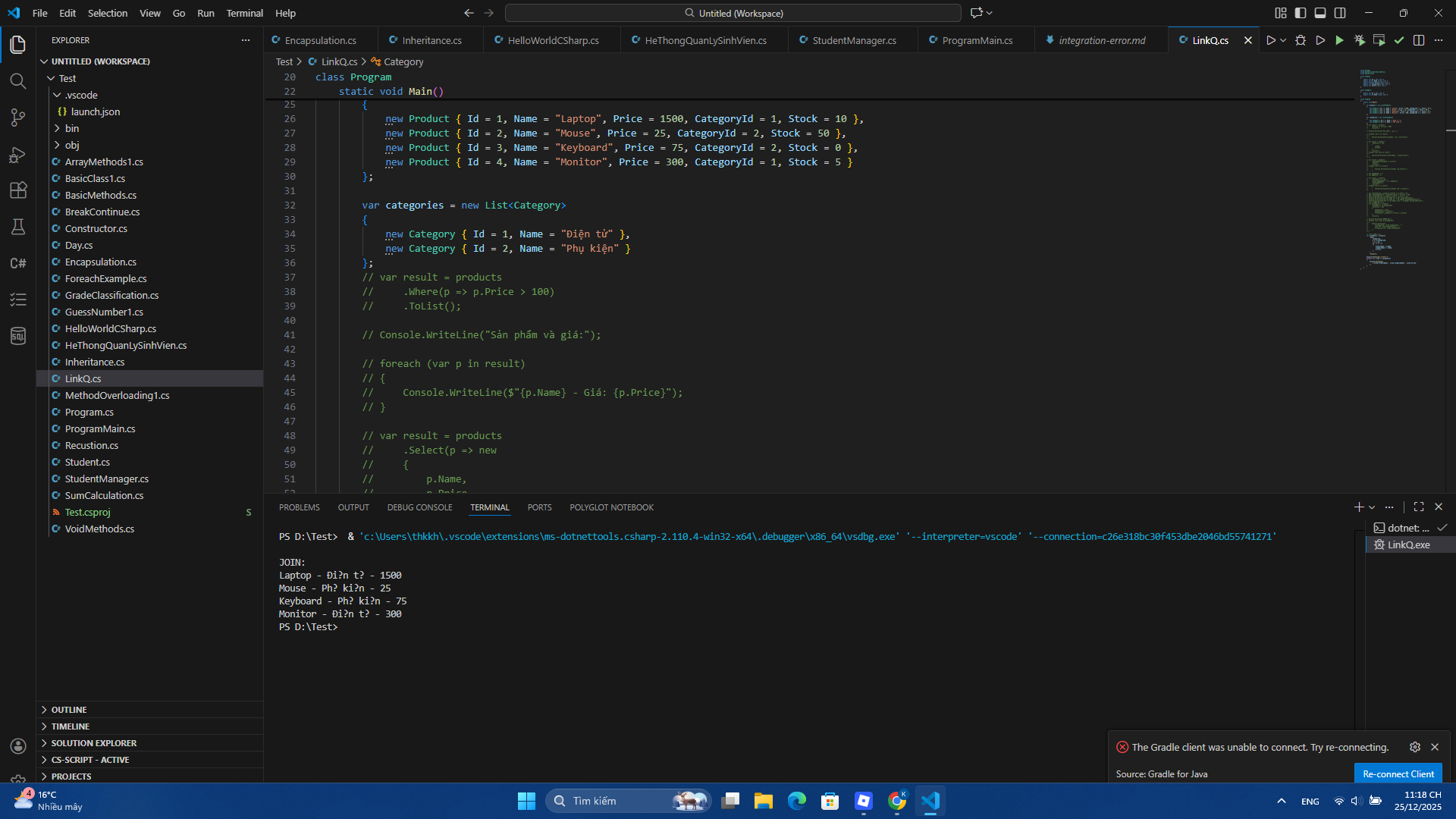
{

Console.WriteLine(

$"{item.ProductName} - {item.CategoryName} - {item.Price}"

);

}

* Kết quả: 

## 4. Thực hành Entity Framework Core (LINQ với Database)

*Hãy trả lời các câu hỏi về LINQ trong Entity Framework Core:*

### Câu hỏi 1: Điểm khác nhau giữa LINQ to Objects vs LINQ to Entity Framework

| **Loại** | **Nơi xử lý** | **Ví dụ** |
| --- | --- | --- |
| LINQ to Objects | Trong bộ nhớ (RAM – CLR) | var result = list.Where(x => x.Id > 5).ToList(); |
| LINQ to Entity Framework | |  | | --- | | Database (SQL Server, MySQL, …) | | var result = \_db.Products.Where(x => x.Id > 5).ToListAsync(); |

### Câu hỏi 2: Khi nào cần gọi .ToListAsync()?

**Trả lời:**

* Chỉ cần gọi .ToListAsync() khi truy vấn Database (LINQ to Entity Framework) (ví dụ: gọi Database)
* Không cần gọi khi LINQ to Objects (dữ liệu đã nằm trong RAM) (ví dụ: LINQ to Objects)

### Câu hỏi 3: So sánh hiệu suất

Cách A:

var products = await \_db.Products.ToListAsync(); // Lấy tất cả

var expensive = products.Where(p => p.Price > 1000).ToList();

Cách B:

var expensive = await \_db.Products.Where(p => p.Price > 1000).ToListAsync();

**Câu hỏi:** Cách nào tốt hơn và tại sao?

**Trả lời:** B tốt hơn vì: EF Core sẽ dịch LINQ thành SQL và lọc dữ liệu ngay trong database, giúp giảm dữ liệu truyền về và tối ưu hiệu suất

## 5. Một điều tôi chưa hiểu rõ hoặc muốn hỏi thêm

**(Bắt buộc phải đặt ít nhất 1 câu hỏi)**

Ví dụ câu hỏi tốt:

* "Sự khác biệt giữa First() và FirstOrDefault() là gì?"
* "Khi nào em dùng GroupBy, khi nào dùng Join?"
* "Em có thể viết LINQ dài 10 dòng được không? Hay là cứ phải đơn giản?"

**Câu hỏi của tôi:**

Sự khác nhau giữa IQueryable và IEnumerable là gì, và khi làm việc với Entity Framework Core thì nên dùng loại nào để tối ưu hiệu suất?

# 

# 

# 

# 

# PHIẾU HỌC TẬP C# MODERN [03] - NULLABLE REFERENCE TYPES, RECORDS & PATTERN MATCHING

## Họ và tên: Mai Trọng Thế

## Lớp: CNTT 18 - 09

## Video đã xem (Link): https://www.youtube.com/watch?v=NDweaZZZcbc&t=269s

## 1. Kiến thức cốt lõi (Tránh Null, Dùng Records, Viết Clean Code)

*Sau khi xem video, hãy điền vào các chỗ trống dưới đây:*

### 1. Vấn đề với Null - NullReferenceException

Hãy nhìn vào ví dụ sau:

public string GetUserName(int userId) {

var user = \_db.Users.Find(userId);

return user.Name; // 💥 Nếu user = null, sẽ crash!

}

* Lỗi này được gọi là: NullReferenceException
* Nó xảy ra khi: chương trình cố gắng truy cập thuộc tính hoặc phương thức của một đối tượng đang có giá trị null
* Để tránh, chúng ta phải: kiểm tra null trước khi sử dụng đối tượng (null check) hoặc dùng các kỹ thuật an toàn với null

### 2. Nullable Reference Types (C# 8.0+)

**Bật tính năng trong .csproj:**

<Nullable>enable</Nullable>

Với tính năng này bật:

* Dấu ? trong string? có nghĩa là: Biến tham chiếu này được phép nhận giá trị null
* Dấu ? trong user?.Name (safe navigation operator) có nghĩa là: Chỉ truy cập Name nếu user không phải null

### 3. So sánh Records vs Class

| **Đặc điểm** | **Class** | **Record** |
| --- | --- | --- |
| **Mutable** (có thể sửa) | Có (mutable) | Mặc định là immutable |
| **Dùng cho** | Entities, Services, Business logic | DTOs, Data transfer, Read-only models |
| **Ví dụ** | Entities, Services | DTOs, Data transfer |

### 4. Pattern Matching - Switch Expression

**Cách cũ (Switch statement):**

string message;

switch (quantity) {

case 1: message = "No discount"; break;

case 2: case 3: message = "10%"; break;

default: message = "20%"; break;

}

**Cách mới (Switch expression - C# 8+):**

string message = quantity switch {

1 => "No discount",

>= 2 and <= 3 => "10%",

> 3 => "20%",

\_ => "Invalid"

};

* Cách mới ngắn gọn dòng code hơn
* Cách mới rõ ràng / dễ đọc hơn dễ đọc hơn
* \_ trong switch expression có nghĩa là: trường hợp mặc định (default) – khớp với mọi giá trị còn lại không thỏa các pattern phía trên

## 2. Ví dụ thực hành (Viết Code an toàn hơn)

*Yêu cầu: Viết code C# theo các TODO dưới đây.*

### Bài tập 1: Dùng Nullable Reference Types

**TODO 1:** Viết một hàm nhận vào int userId và trả về string? (có thể null):

public string? GetUserName(int userId)

{

var user = \_db.Users.Find(userId);

if (user == null)

return null;

return user.Name;

}

**TODO 2:** Sửa hàm trên bằng **safe navigation operator** ?.:

public string? GetUserNameSafe(int userId) {

var user = \_db.Users.Find(userId);

return user?.Name;

}

### Bài tập 2: Viết Record đơn giản

**TODO 1:** Viết một record có tên ProductDTO với 3 thuộc tính: Id (int), Name (string), Price (decimal)

public record ProductDTO(int Id, string Name, decimal Price);

**TODO 2:** Tạo một instance của ProductDTO:

// Hãy tạo instance:

var product = new ProductDTO(1, "Laptop", 1000m);

**TODO 3:** Tạo một bản copy với giá sửa đổi (dùng with expression):

// Tạo bản copy với giá giảm 10%:

var discounted = product with { Price = product.Price \* 0.9m };

### Bài tập 3: Viết Switch Expression

**Yêu cầu:** Viết một hàm nhận vào loại sản phẩm (string) và trả về mã thuế (int)

Ví dụ:

* "Electronics" → 10
* "Food" → 5
* "Clothing" → 8
* Khác → 0

// Hãy viết hàm của bạn:

public int GetTaxRate(string productType)

{

return productType switch

{

"Electronics" => 10,

"Food" => 5,

"Clothing" => 8,

\_ => 0

};

}

## 3. Câu hỏi bắt buộc

### Câu hỏi 1: Nullable Reference Types

Giả sử bạn viết code sau:

string name = \_db.Users.Find(userId)?.Name; // Có lỗi gì không?

**Trả lời:** Có lỗi vì: \_db.Users.Find(userId)?.Name có thể trả về null, trong khi biến name được khai báo là string (non-nullable).

**Cách sửa:**

string? name = \_db.Users.Find(userId)?.Name;

### Câu hỏi 2: Record vs Class - Khi nào dùng cái nào?

**Trường hợp A:** Tạo một DTO để truyền dữ liệu từ API sang Frontend

* Dùng Record (Record / Class)

**Trường hợp B:** Tạo một Entity để đại diện cho một bảng trong Database

* Dùng Class (Record / Class)

**Trường hợp C:** Tạo một Service để quản lý logic

* Dùng Class (Record / Class)

### Câu hỏi 3: Switch Expression

Nhìn vào code sau:

int score = 85;

string grade = score switch {

>= 90 => "A",

>= 80 => "B",

>= 70 => "C",

\_ => "F"

};

**Câu hỏi:** Giá trị của grade là bao nhiêu?

* A. "B"
* B. "C"
* C. "F"
* D. Báo lỗi

**Trả lời:** A vì:

* score = 85
* Điều kiện được kiểm tra từ trên xuống
* 85 không ≥ 90 → bỏ qua
* 85 ≥ 80 → khớp pattern thứ hai

## 4. Thực hành tình huống thực tế

### Tình huống 1: Xử lý Null khi gọi API

public async Task<string> GetCityNameAsync(int cityId) {

var city = await \_db.Cities.FindAsync(cityId);

return city.Name; // Có thể crash!

}

**Yêu cầu:** Sửa hàm trên để **an toàn**:

public async Task<string?> GetCityNameAsync(int cityId)

{

var city = await \_db.Cities.FindAsync(cityId);

if (city is null)

return null;

return city.Name;

}

### Tình huống 2: Viết API Response DTO

Giả sử bạn muốn trả về một response chứa thông tin sản phẩm và thông tin category.

**Yêu cầu:** Viết 2 records:

* CategoryInfo (Id, Name)
* ProductResponse (Id, Name, Price, CategoryInfo)

public record CategoryInfo(int Id, string Name);

public record ProductResponse(

int Id,

string Name,

decimal Price,

CategoryInfo Category

);

## 5. Một điều tôi chưa hiểu rõ hoặc muốn hỏi thêm

**(Bắt buộc phải đặt ít nhất 1 câu hỏi)**

Ví dụ câu hỏi tốt:

* "Nếu em quên dấu ? trong string? thì sao? Code vẫn chạy không?"
* "Record bất biến có nghĩa là gì? Em không thể sửa giá trị được sao?"
* "Khi nào em dùng switch expression, khi nào dùng if-else?"

**Câu hỏi của tôi:**

Nếu API luôn đảm bảo dữ liệu tồn tại (ví dụ: luôn có User, City, Product), thì em có nên dùng string? và null-check nữa không, hay nên throw exception để code rõ ràng hơn? Khi đi làm thực tế thì team thường chọn cách nào và vì sao?

**Chúc mừng bạn đã hoàn thành Phiếu Học Tập C# Modern!** 🎓