مَا لَا يَسَعُ
المُبَرْمِجَ جَهْلَهُ

CH01 - OOP

CH01 - OOP

EP.01 : Inheritance vs Composition

لو أنت فاكر إن كل ما تشوف علاقة is-a تستخدم Inheritance...
يبقى في مشكلة عندك .

Abdalrhman Ismail
Backend .NET Trainee

أغلبنا لما بدأ فهنا الـ Inheritance غلط

أول ما حد يتعلم OOP يقوم عامل :

Simple Example

```
1  class Animal
2  {
3      public void Eat() => Console.WriteLine("Eating...");
4  }
5
6  class Dog : Animal
7  {
8      public void Bark() => Console.WriteLine("Barking...");
9  }
```

"ويبدأ يعمل شجرة نسب مالهاش أي لازمة
ويحس إنه كده كتب كود عبقري "

بس الحقيقة ؟

إن الـ Inheritance ممكن تعتبره سلاح ذو حدين فعلا
لو استخدمته غلط ، هتدفع الثمن في كل سطر بعده.

Abdalrhman Ismail
Backend .NET Trainee

و السؤال هنا ؟

هل الـ Inheritance دايمًا الحل ؟

ولا في حاجة أقوى اسمها Composition ؟

بس خلينا نقول أولًا يعني إيه Inheritance أصلًا ؟

الـ Inheritance هي إن : Class يرث سلوك وخصائص

من Class تاني وبنقول عليها علاقة is-a ← Dog is an Animal

لكن المشكلة إن العلاقة دي صلبة جدًا , لو عدلت في الأب كل

الأبناء يتأثروا متعرفش تخلط بين سلوكين لنفس الشيء

وده بداية مفهوم مهم اسمه : **Tight Coupling**

يعني الكلاسات مربوطة ببعض ربط قوي أي تعديل في الـ Base Class =

ripple effect في المشروع كله و ده بيقلل من الـ :

Flexibility	Maintainability	Scalability

Abdalrhman Ismail
Backend .NET Trainee

طيب الحل ؟

Favor Composition over Inheritance

و دي قاعدة أساسية في تصميم الـ Software

يعني بدل ما تقول: Dog is an Animal

قــول : Dog has a behavior

مثال مهم يوضح مشكلة الـ Inheritance

Violation of the Liskov Substitution Principle (LSP)

```
1  class Bird
2  {
3      public void Fly()
4      {
5          Console.WriteLine("Flying...");
6      }
7  }
8
9  class Ostrich : Bird
10 {
11     // !! والنعامة مبتطيرش
12 }
```

AbdalRhman Ismail
Backend .NET Trainee

مبدأ Liskov Substitution Principle (LSP)

والمبدأ ده علاقته بالمشكلة اللي معانا إننا كسرنا الـ Logic
و ده بالظبط اللي بيتعارض معاه ...
بس خلينا نقول الاول , المبدأ ده بيقول ايه ؟
ببساطة إن أي كلاس ابن لازم ينفع يحل مكان الأب
من غير ما يبوّظ البرنامج ... بمعنى اخر :

هل النعامة ينفع تتحط مكان الطائر؟
هنلاقي الإجابة : لا ...
يبقى الـ Inheritance هنا كقرار ديزاين غلط .
وده يورينا إن الـ Inheritance مش بس"Educational Concept"
دي "Already Engineering Responsibility"

Abdalrhman Ismail
Backend .NET Trainee

طب نعملها صح إزاي؟ (Composition)

فهنيجي هنا نعيد ديزاين المثال اللي فات :

دلوقتي : العصفور يطير ✔

النعامة ميبقاش عندها IFlyable ✔

مفيش كسر للـ Logic ✔

مفيش توريث غصب ✔

و ده بقى يحقق مبدأين مهمين من SOLID

فهنعمل ربط سريع بـ SOLID الاول :

Open/Closed Principle (OCP) (1)

نقدر نزود سلوك جديد , من غير ما نعدل في الكود القديم
بس نضيف Implementation جديدة للـ Interface

Dependency Inversion (2) Principle (DIP)

Composition In C#

```
1  public interface IFlyable
2  {
3      void Fly();
4  }
5
6  public class FlyBehavior : IFlyable
7  {
8      public void Fly()
9      {
10         Console.WriteLine("Flying...");
11     }
12 }
13
14 public class Bird
15 {
16     private readonly IFlyable _flyBehavior;
17
18     public Bird(IFlyable flyBehavior)
19     {
20         _flyBehavior = flyBehavior;
21     }
22
23     public void PerformFly()
24     {
25         _flyBehavior?.Fly();
26     }
27 }
```

Dependency Inversion Principle (DIP)

```
1  private readonly IFlyable _flyBehavior;
```

لاحظ إن : الكلاس بيعتمد على abstraction مش على implementation

وده يخلي النظام : Highly flexible | Easy to modify | Testable

طب هل الـ Inheritance ⟵ Anti-Pattern ؟

الإجابة على السؤال ده ممكن نقول :مش دايمًا لكن في حالات كتير.. آه

الـ Inheritance تبقى غلط لما :

يبقى Inheritance هنا مش Best Practice ...

لو العلاقة مش منطقية 100%

أو الـ behavior مش ثابت و في احتمال إن requirements تتغير قدام شوية ...

مقارنة سريعة

Inheritance	Composition
علاقة قوية	علاقة مرنة
Tight Coupling	Loose Coupling
صعب التعديل	سهل التوسع
يكسر LSP أحيانًا	يحافظ عليه
مناسب للعلاقات الثابتة	مناسب للسلوكيات المتغيرة

Abdalrhman Ismail
Backend .NET Trainee

مثال لو بتبني API بـ .NET

عندك مثلاً :

- Notification Service
- Payment System
- Authentication Provider
- Logging Mechanism

لو استخدمت inheritance غلط هتلاقي نفسك كل ما Client يطلب تعديل فهتعدل في Base Class وكل النظام يتأثر

لكن لو استخدمت Composition + Interfaces :

- تضيف Payment جديد من غير ما تلمس القديم
- تفصل Business Logic عن Infrastructure
- تعمل Mock في Unit Tests بسهولة
- تحقق Clean Architecture

وده الفرق بين : Junior بيكتب كود يشتغل
و Engineer بيبني نظام يعيش

Abdalrhman Ismail
Backend .NET Trainee

كلمة أخيرة

انا عبدالرحمن اسماعيل
طالب بالفرقة الثانية كلية الحاسبات
و المعلومات ، مهتم بالـ : Software
Development , و حاليا في مسار الـ
Web Development مع الـ Ps ,
مجرد شخص بيسعى و بيحاول يعمل اللي عليه و يستمر .

لو وصلت لحد هنا فـ أشكرك جدًا
و نتقابل إن شاء لله في بوست جديد من السلسلة ♡

Abdalrhman Ismail
Backend .NET Trainee