**API Development with .Net Core API 3.1**

* + 1. **Basic knowledge (1 day)** 
       1. S.O.L.I.D Principles

1. Single responsibility principle:

Mỗi lớp chỉ nên chịu trách nhiệm về một nhiệm vụ cụ thể nào đó mà thôi.

1. Open/closed principle

Không được sửa đổi một Class có sẵn, nhưng có thể mở rộng bằng kế thừa.

1. Liskov substitution principle

Các đối tượng (instance) kiểu class con có thể thay thế các đối tượng kiểu class cha mà không gây ra lỗi.

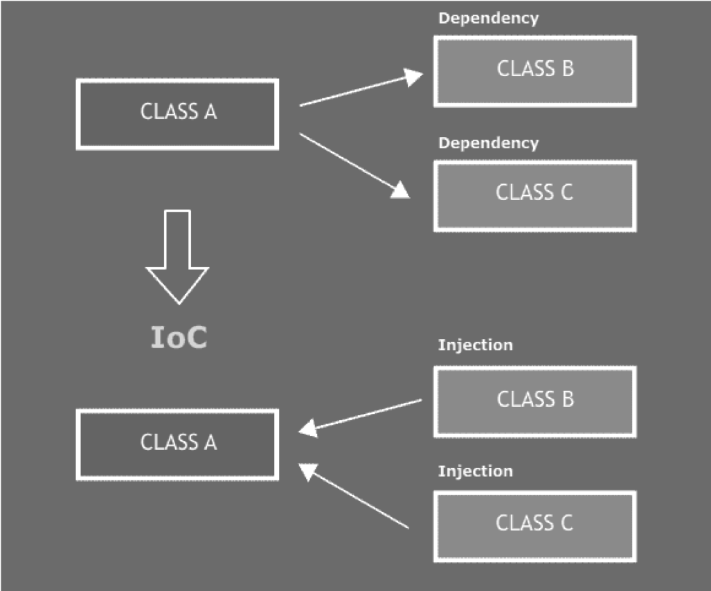
1. Interface segregation principle

Thay vì dùng 1 interface lớn, ta nên tách thành nhiều interface nhỏ, với nhiều mục đích cụ thể.

1. Dependency inversion principle

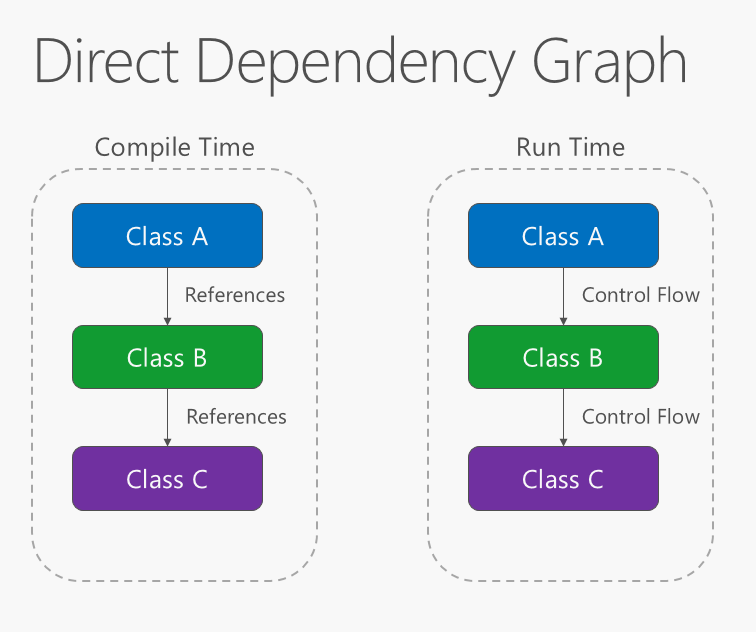
* Các module cấp cao không nên phụ thuộc vào các modules cấp thấp. Cả 2 nên phụ thuộc vào abstraction.
* Interface (abstraction) không nên phụ thuộc vào chi tiết, mà ngược lại (Các class giao tiếp với nhau thông qua interface (abstraction), không phải thông qua implementation.)
  + - 1. OOP Principles
  1. Abstraction (Tính trừu tượng)
* Trừu tượng có nghĩ là tổng quát hóa một cái gì đó lên, không cần chú ý chi tiết bên trong. Nó không màng đến chi tiết bên trong là gì và người ta vẫn hiểu nó mỗi khi nghe về nó.
* Trừu tượng được thể hiện qua interface và abstract class
  1. Encapsulation (Tính bao đóng)
* Các dữ liệu và phương thức có liên quan với nhau được đóng gói thành các lớp để tiện cho việc quản lý và sử dụng. Tức là mỗi lớp được xây dựng để thực hiện một nhóm chức năng đặc trưng của riêng lớp đó.
* Ngoài ra, đóng gói còn để che giấu một số thông tin và chi tiết cài đặt nội bộ để bên ngoài không thể nhìn thấy.
* Tính đóng gói được thể hiện thông qua các modifier access và property
  1. Inheritance (Tính kế thừa)
* Nó cho phép xây dựng một lớp mới dựa trên các định nghĩa của lớp đã có.
* Tái sử dụng mã nguồn 1 cách tối ưu, tận dụng được mã nguồn.
  1. Polymophirsm (Tính đa hình)
* Tính đa hình là một hành động có thể được thực hiện bằng nhiều cách khác nhau.
* Tính đa hình gồm: Đa hình run time và đa hình compile time
  + - 1. Inversion of Control – IoC

IoC -> đảo ngược sự điều khiển

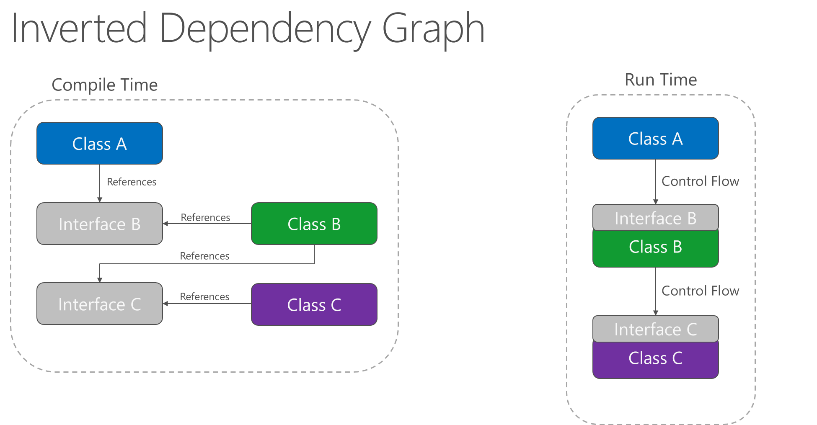


* Ở mô hình không IoC, Class A khi cần chủ động tạo ra đối tượng lớp Class B và Class C (nó nắm quyền khởi tạo, điều khiển)
* Với mô hình IoC thì class A không tự khởi tạo cũng không chịu trách nhiệm quản lý Class B, Class C. Nó nhận được hai dependency này từ bên ngoài thông qua một cơ chế nào đó (như bằng setter, bằng tham số hàm tạo lớp A, bằng gán thuộc tính ...)

Thiết kế truyền thống - tham chiếu trực tiếp đến Dependency



Thiết kế theo cách đảo ngược phụ thuộc Inverse Dependency

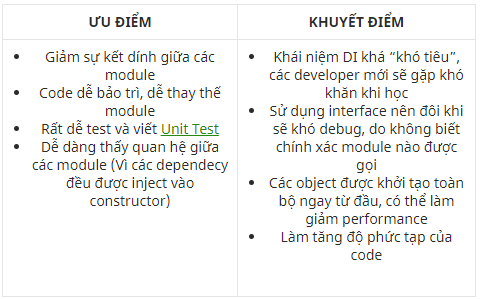


* Ở đây class A thay vì phụ thuộc trực tiếp vào class B thì nó sẽ phụ thuộc vào interface B rồi từ đó class B sẽ được tiêm vào interface B

**Các kiểu Dependency Injection**

* Inject thông qua phương thức khởi tạo: thông qua hàm khởi tạo
* Inject thông qua setter: thuộc tính của lớp
* Inject thông qua các Interface: Interface có chứa các phương thức Setter

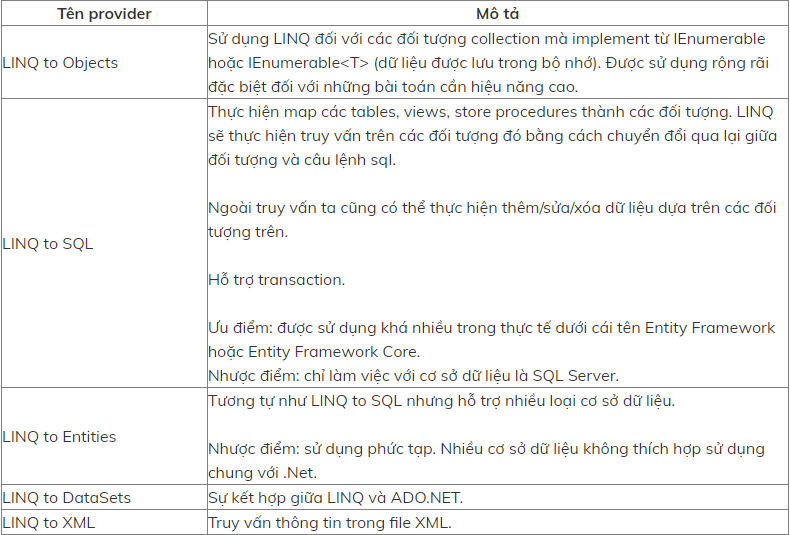
**Ưu điểm và khuyết điểm của DI**



* + - 1. LinQ

**Thành phần cấu thành LINQ**

* Nguồn dữ liệu.
* Tạo câu lệnh truy vấn.
* Thực thi truy vấn để lấy kết quả.
  1. Nguồn dữ liệu



* 1. Tạo câu lệnh truy vấn.
* Câu truy vấn: [LINQ Query Syntax (tutorialsteacher.com)](https://www.tutorialsteacher.com/linq/linq-query-syntax)
* Phương thức truy vấn: [LINQ - Query Operators - Tutorialspoint](https://www.tutorialspoint.com/linq/linq_query_operators.htm)

1. Thực thi truy vấn để lấy kết quả.

Có 2 kịch bản để thực thi truy vấn:

* Trì hoãn thực thi (Deferred Execution): có thể dùng for/foreach để lấy dữ liệu
* Thực thi ngay lập tức (Immediate Execution): *dùng những toán tử chuyển đổi như ToList, ToArray, ToDictionary... hoặc những toán tử thành phần như First, FirstOrDefault, Last, LastOrDefault...*