1. **Lý do cần thiết kế theo design pattern Dependency Injection**

* Nguyên tắc thiết kế SOLID (Dependency inversion)

Các module cấp cao không nên phụ thuộc vào các modules cấp thấp (mối quan hệ has-a).

* Cả 2 nên phụ thuộc vào abstraction.

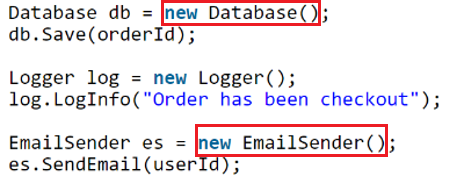
Interface (abstraction) không nên phụ thuộc vào chi tiết, mà ngược lại.

* Các class giao tiếp với nhau thông qua interface, không phải thông qua implementation.



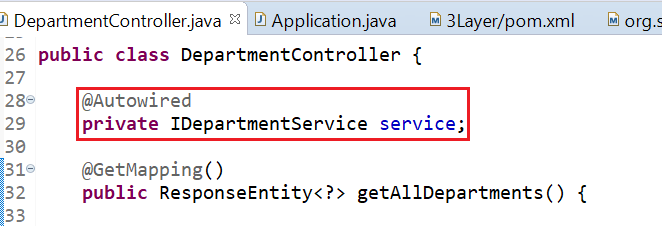
* Nếu các module cấp cao gọi các module cấp thấp 🡺 Module cấp cao sẽ phụ thuộc và module cấp thấp 🡺 tạo ra các dependency

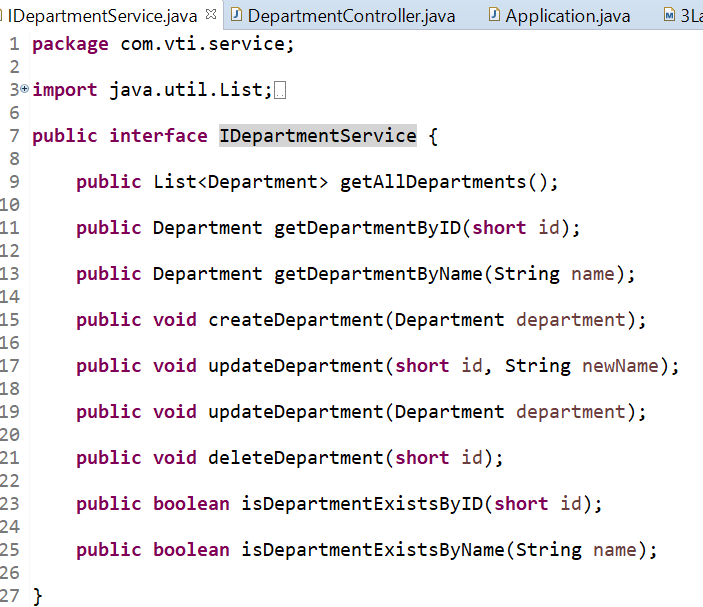
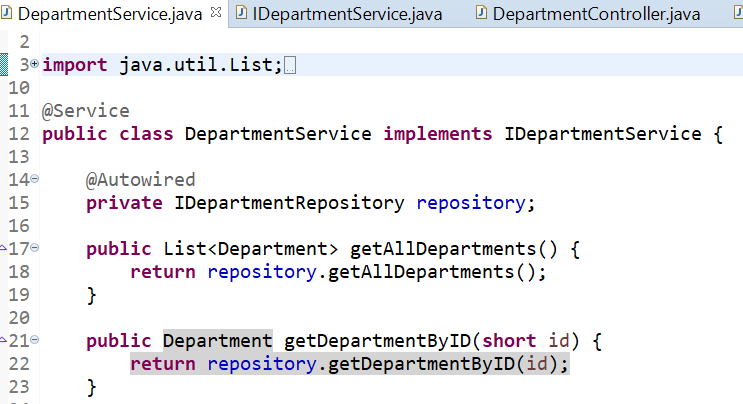
Khi module cấp thấp thay đổi 🡺 ảnh hưởng tới module cấp cao



*Object Cart phụ thuộc vào các Object Database, Logger, EmailSender*

* Nên thiết kế theo Dependency Injection
* Các module sẽ phụ thuộc vào 1 interface không đổi

**

1. **Dependency Injection (DI)**

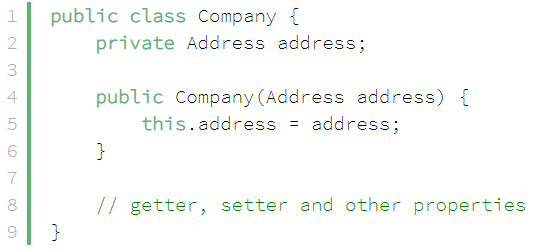
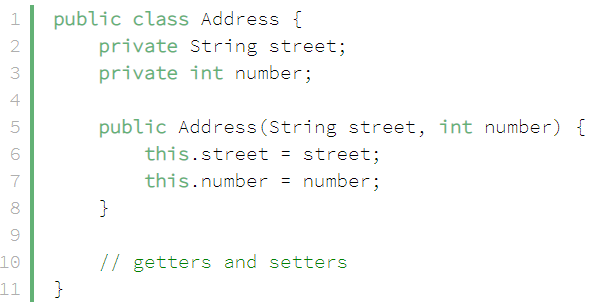
* Các module phụ thuộc (dependency) sẽ được inject vào module cấp cao
* DI được dùng để làm giảm sự phụ thuộc giữa các module

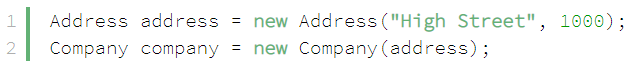
🡺 Dễ dàng hơn trong việc thay đổi module, bảo trì code và testing.

* Có 3 dạng DI
* **Constructor Injection** (thông dụng nhất)

Các dependency sẽ được DI container truyền vào (inject vào) 1 class thông qua constructor của class đó.

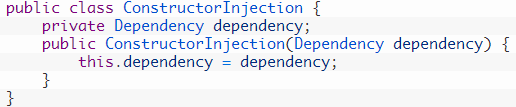
VD1:



*init object*

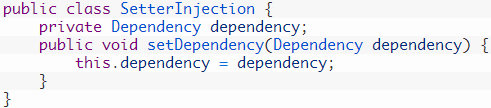
VD2:



* **Setter Injection**

Các dependency sẽ được truyền vào 1 class thông qua các hàm Setter.

VD: Tương tự constructor



* **Interface Injection**

Class cần inject sẽ implement 1 interface.

Interface này chứa 1 hàm tên Inject.

Container sẽ injection dependency vào 1 class thông qua việc gọi hàm Inject của interface đó.

1. **Cách thiết kế**

* Các module không giao tiếp trực tiếp với nhau, thông qua interface.
  + Module cấp thấp sẽ implement các interface
  + Module cấp cao sẽ gọi các interface
* Việc khởi tạo các Module cấp thấp sẽ được
  + Inject vào thông qua Constructor hoặc setter
  + Do DI Container thực hiện

Có thể tự cài đặt DI Container hoặc dùng library Unity, NInject, StructureMap,…

(Ta không gọi operator new để khởi tạo instance mà instance sẽ được truyền từ ngoài vào)

* Module nào gắn với interface nào sẽ được config trong code (hoặc trong file xml)
* **Thiết kế DI Container**
  + Lưu trữ các Interface (key), Module(value) tương ứng
  + Khi cài đặt 1 Module, container sẽ tìm Constructor của module đó
    - Nếu constructor không có tham số(Module không có dependency)

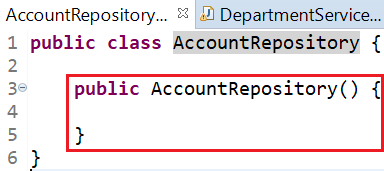
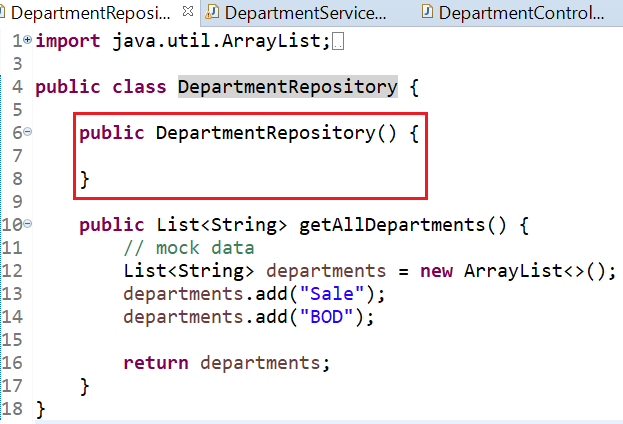
Container sẽ gọi constructor này để khởi tạo Module

* + - Nếu constructor có tham số (Module có dependency)

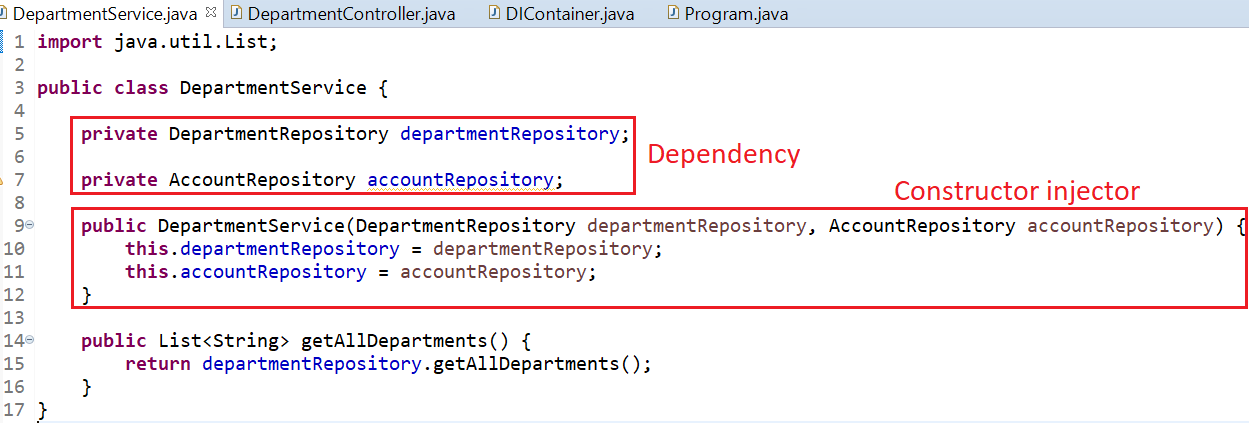
Container sẽ khởi tạo các tham số này và gán chúng vào constructor

🡺 đây là quá trình **injection**

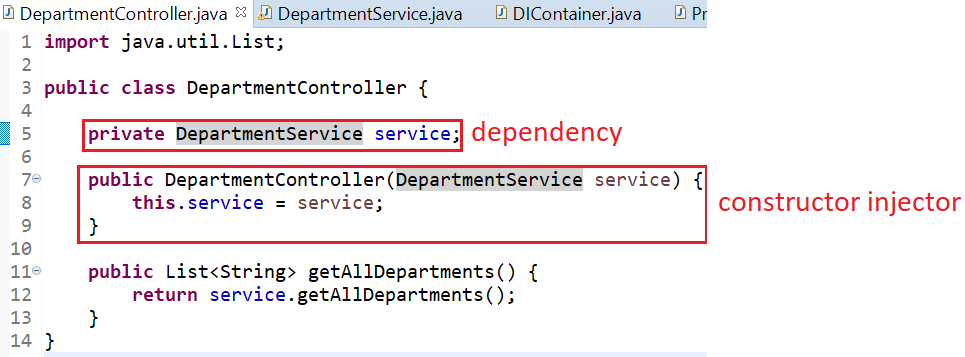
VD1: trong demo folder

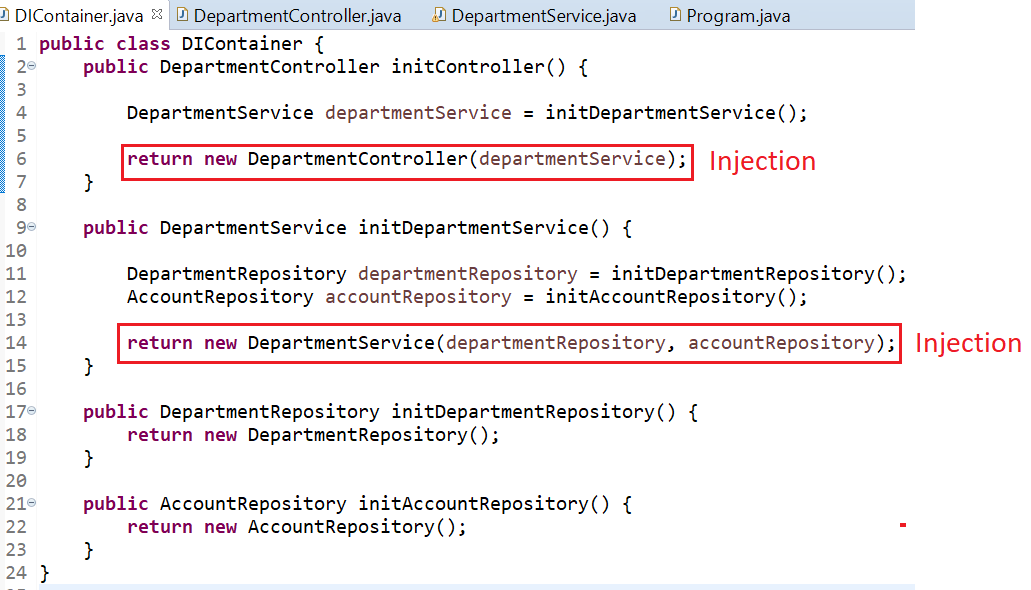
*Repository*



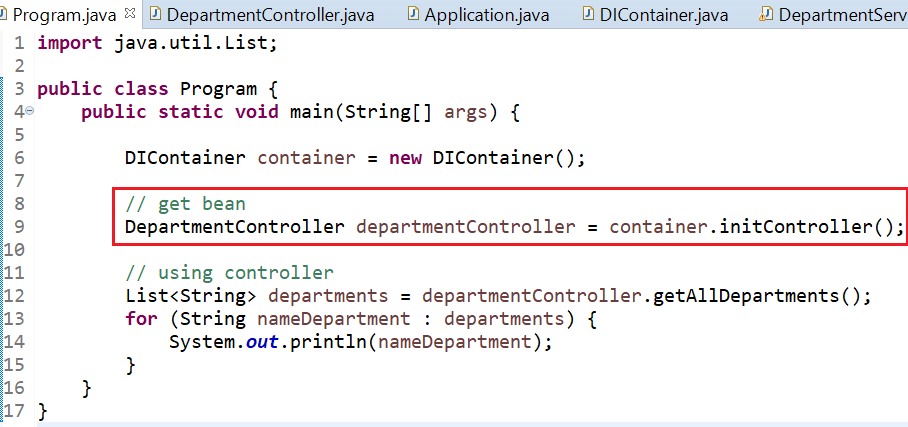
*Service*

**

*Controller*

**

*DI Container*



*Using*

VD2:

<https://toidicodedao.com/2015/11/12/dependency-injection-va-inversion-of-control-phan-3-di-container-ap-dung-di-vao-asp-net-mvc/>

1. **Ưu điểm và khuyến điểm**

|  |  |
| --- | --- |
| **ƯU ĐIỂM** | **KHUYẾT ĐIỂM** |
| Giảm sự kết dính giữa các module | Khái niệm DI khá khó hiểu, các developer mới sẽ gặp khó khăn khi học |
| Code dễ bảo trì, dễ thay thế module | Sử dụng interface nên đôi khi sẽ khó debug, do không biết chính xác module nào được gọi |
| Rất dễ test và viết Unit Test | Các object được khởi tạo toàn bộ ngay từ đầu, có thể làm giảm performance |
| Dễ dàng thấy quan hệ giữa các module (Vì các dependecy đều được inject vào constructor) | Làm tăng độ phức tạp của code |

* **Reference**

<https://toidicodedao.com/2015/11/03/dependency-injection-va-inversion-of-control-phan-1-dinh-nghia/>

<https://www.baeldung.com/inversion-control-and-dependency-injection-in-spring>

<https://stackjava.com/design-pattern/dependency-injection-di-la-gi.html>

<https://huongdanjava.com/vi/hieu-ve-dependency-injection.html>