**IoC và DI**

1. **Khái niệm**

Trong Java, IoC (Inversion of Control) và DI (Dependency Injection) là hai khái niệm quan trọng trong lập trình hướng đối tượng, đặc biệt là khi xây dựng các ứng dụng theo kiến trúc Spring Framework.

* **IoC (Inversion of Control)**

**Inversion of Control** là **nguyên tắc** trong đó việc quản lý đối tượng và vòng đời của chúng được chuyển giao cho **một container** hoặc **framework** thay vì do chính mã chương trình quản lý.

Thay vì tạo và quản lý đối tượng theo cách thủ công, **IoC Container** sẽ tạo và cung cấp các đối tượng theo nhu cầu của ứng dụng. Điều này giúp tách biệt việc **khởi tạo** đối tượng khỏi **sử dụng** nó, giúp mã dễ bảo trì hơn.

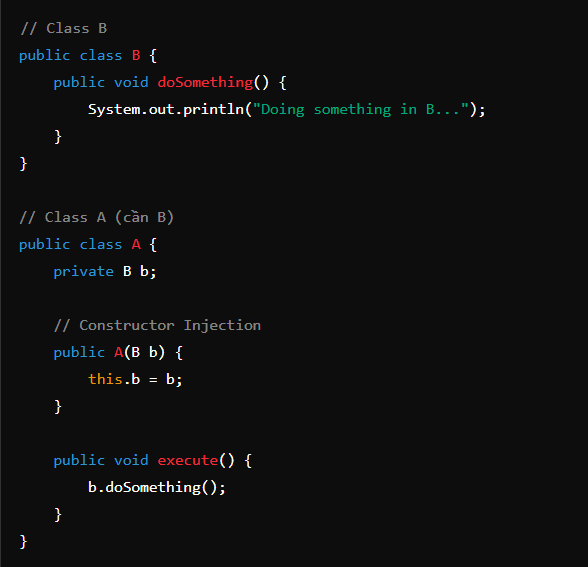
**Ví dụ:** Nếu bạn cần đối tượng của class Student, thay vì gọi new Student () để tạo đối tượng, **IoC container** (như Spring) sẽ cung cấp đối tượng đó cho bạn khi cần hay nói cách khác IoC khi mình cần dùng class Student không cần phải gọi New nữa mà chỉ gần gọi getBean. IoC sẽ giúp chúng ta tạo ra class (đối tượng) và quản lí chúng

**Vậy tại sao lại cần dùng IoC** : sau này chúng ta sẽ làm dự án rất lớn và trong đó sẽ có rất nhiều class (đối tượng ) nếu chúng ta cứ New() việc này sẽ rất tốn thời gian và gây cho hệ thống của chúng ta bị chậm

## ****DI (Dependency Injection)****

**Dependency Injection** là **kỹ thuật** để thực hiện IoC. Nó là quá trình **"tiêm" các phụ thuộc** cần thiết vào đối tượng từ bên ngoài Trong Java, DI thường được sử dụng bằng cách **tiêm qua constructor, setter**, hoặc **field injection**.

**Ví dụ giúp hiểu hơn về công dụng của DI:** Giả sử class A cần sử dụng class B. Thay vì A tự tạo một đối tượng của B, chúng ta sẽ **tiêm** đối tượng của B vào A thông qua constructor hoặc setter.



**IoC** là nguyên tắc chuyển giao quyền kiểm soát đối tượng cho một container hoặc framework.

**DI** là cách thức thực hiện IoC, bằng cách tiêm các phụ thuộc vào đối tượng thay vì để nó tự khởi tạo.

# **Cách làm**

* **Tạo IoC**

B0: Đầu tiên chúng ta cần chuẩn bị **dependencies** trong pom (chuyển sang maven project)

|  |
| --- |
| <!-- https://mvnrepository.com/artifact/org.springframework/spring-beans -->  <!-- https://mvnrepository.com/artifact/org.springframework/spring-beans -->  <!-- https://mvnrepository.com/artifact/org.springframework/spring-core --> |

B1: Chúng ta chỉ cần **tạo class** như bình thường .Ví dụ class SinhVien

Chúng ta có 2 cách để chuyển nó thành IoC đó là viết code trong xml và dùng annotation . Đầu tiên mình sẽ viết code trong xml

B2: Tạo file **applicationContext.xml** trong package **src/main/java** (lưu ý phải đúng chỗ này và giống tên)

B3 : Vào link bên dưới để lấy beans

<https://docs.spring.io/spring-framework/docs/4.2.x/spring-framework-reference/html/xsd-configuration.html>

|  |
| --- |
| <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns:context=*"http://www.springframework.org/schema/context"*  xsi:schemaLocation=*"*  *http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans.xsd*  *http://www.springframework.org/schema/context*  *http://www.springframework.org/schema/context/spring-context.xsd"*>  //vietcode ở đây  </beans> |

B4 : Viết code vào xml

|  |
| --- |
| <bean id=*"****hs****"* class=*"duong dan toi class SinhVien "*>  </bean> |

B5: Test : chúng ta sẽ tạo hàm main để test. Trước khi test cần khai báo context , sau đó chỉ cần gọi getBean để khởi tạo đối tượng

|  |
| --- |
| **public** **static** **void** main(String[] args) {  // **TODO** Auto-generated method stub  ClassPathXmlApplicationContext context = **new** ClassPathXmlApplicationContext("applicationContext.xml"); //tên xml      SinhVien sinhvien = context.getBean("***hs***", SinhVien.**class**);    context.close();  } |

Tóm lại bạn thấy đó như ban đầu nếu muốn khởi tạo sinh viên chúng ta sẽ dùng new SinhVien() còn bây giờ sẽ khởi tạo bằng getBean.

Vậy còn cách dùng annotation thì sao. Anotation sẽ đơn giản hơn nhiều và chúng ta sẽ không cần phải code nhiều.Cách làm như sau :

B0, B1, B2, B3 : tương tự

B4 thay vì viết code chúng ta chỉ cần thêm

* @component trên đầu class SinhVien : nếu để như vậy id bean mặc đình là tên class
* @component(“tenMoi”) thì id bean sẽ là “tenMoi”

B5 Test

|  |
| --- |
| **public** **static** **void** main(String[] args) {  // **TODO** Auto-generated method stub  ClassPathXmlApplicationContext context = **new** ClassPathXmlApplicationContext("applicationContext.xml"); //tên xml      SinhVien sinhvien = context.getBean("***sinhVien***", SinhVien.**class**);  //thay vì truyền vào id là hs .chúng ta sẽ truyền vào tên class.Lưu ý chữ đầu viết thường    context.close();  } |

* **Tạo DI nếu ở trên dùng IoC thông qua viết code trong xml**

**Có 3 cách**

Cách 1 : Dùng contrustor (sử dụng nhiều)

Cách 2 : Dùng setter (ít hơn chút)

Cách 3 : Dùng interface (hiếm)

**Tôi sẽ hướng dẫn 2 cách đầu :Đầu tiên sẽ là cách 1 dùng constructor**

**Lưu ý :** trước khi dùng DI bạn phải cấu hình IoC trước như phía trên

Ví dụ đề : Tôi có

* Class SinhVien implement class ConNguoi có phương thức là (getTuoi)
* Class NhanVien implement class CanBo có phương thức là (getTen)

Làm sao để class **SinhVien** có thể lấy được phương thức **getTen** nhưng không được phép phụ thuộc gì tới NhanVien. Đó chính là DI

B1: Chuyển thành IoC

B2 : Trong class SinhVien chúng ta sẽ tạo ra biến

private CanBo canBo; (lấy phương thức thông qua interface của NhanVien)

B3: Tạo constructor SinhVien đầy đủ tham số .Sau đó chỉ cần viết phương thức mới có sử dụng phương thức của class NhanVien.

B4: Viết code trong xml

|  |
| --- |
| <bean id=*"nv"* class=*"duong dan toi class NhanVien "*>  </bean>  <bean id=*"sv"* class=*"duong dan toi class SinhVien "*>  <constructor-arg ref=*"nv"* /> (câu lệnh này là chúng ta đang tiêm cho class SinhVien có thể lấy được phương thức của class NhanVien)  </bean> |

B5: Test như bình thường

**Tôi sẽ hướng dẫn 2 cách đầu :Sau đó sẽ là cách 2 dùng constructor**

B1: Chuyển thành IoC

B2 : Trong class SinhVien chúng ta sẽ tạo ra biến

private CanBo canBo; (lấy phương thức thông qua interface của NhanVien)

B3: Tạo phương thức setCanBo.Sau đó chỉ cần viết phương thức mới có sử dụng phương thức của class NhanVien.

B4: Viết code trong xml .Lưu ý class SinhVien có bao nhiêu biến có bấy nhiêu property

|  |
| --- |
| <bean id=*"nv"* class=*"duong dan toi class NhanVien "*>  </bean>  < <bean id=*"sv"* class=*" duong dan toi class SinhVien "*>  <property name=*"canBo"* ref=*"nv"* /> câu lệnh này là chúng ta đang tiêm cho class SinhVien có thể lấy được phương thức của class NhanVien)  <property name=*"email"* value=*"nguyenphan1122k3@gmail.com"*/>  <property name=*"team"* value=*"Criket team"*/>  </bean> |

B5: Test như bình thường

* **Tạo DI nếu ở trên dùng IoC thông qua annotation**

**Có 3 cách**

Cách 1 : Dùng contrustor (sử dụng nhiều)

Cách 2 : Dùng setter (ít hơn chút)

Cách 3 : Dùng interface (hiếm)

B1, B2, B3 : TƯƠNG TỰ

B4: Thay vì viết code ta chỉ cần gắn @Autowired phía trên constuctor

B5: Test như bình thường

Nhưng ví dụ có quá nhiều class implement CanBo thì sao. Lúc đó SinhVien sẽ lấy phương thức getTen() của ai

@Qualifier(“tenBean của class muốn lấy phương thức getTen()”) dùng phía trên của private CanBo canBo