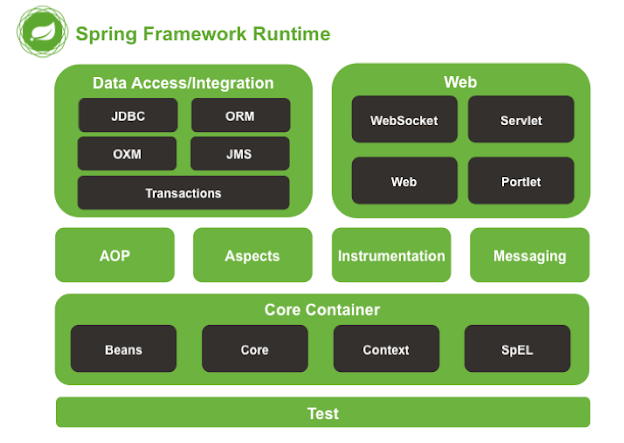
# Tổng quan về Spring

1. Được xây dựng trên 2 nguyên tắc design: Dependency Injection (DI) và Aspect Oriented Programming (AOP). Sử dụng mô hình POJO (Plain Old Java Object) giúp không phải làm việc với EJB, ứng dụng, luồng, cấu hình, …
2. Kiến trúc:



* Core Container: bào gồm các module spring core, beans, context, EL (Expression Language)
* Spring Bean: cung cấp tính năng IoC (Inversion of Control) và DI (Dependency Injection)
* Spring Context: hỗ trợ đa ngôn ngữ
* Spring Expression Language: Mở rộng từ JSP
* AOP, Aspects, Instrumentation và Messaging: hỗ trợ lập trình hướng khía cạnh
* Data Access / Integration: Giao tiếp vs CSDL
* Spring ORM (Object Relational Mapping)

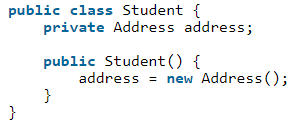
1. Các dự án:

* Spring MVC: xây dựng nền tảng web
* Spring Security: Cung cấp cơ chế bảo mật và phân quyền
* Spring Boot: chạy ứng dụng nhanh chóng
* Spring Batch: Hỗ trợ tạo lich trình và tiến trình các công việc theo mẻ
* Spring Social: Kết nối API thứ 3 (Fb, Gg, …)

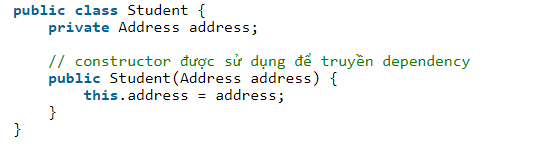
# Spring Core

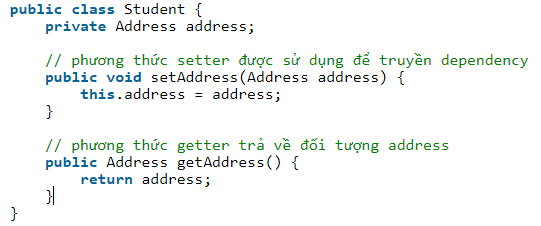
1. Dependency Injection: Loại bỏ sự phụ thuộc giữa các code

VD: Không dùng DI thì address phụ thuộc vào Student



VD: Dùng DI thì address có thể thay đổi thông qua Contructor và setter





1. IoC Container trong Spring

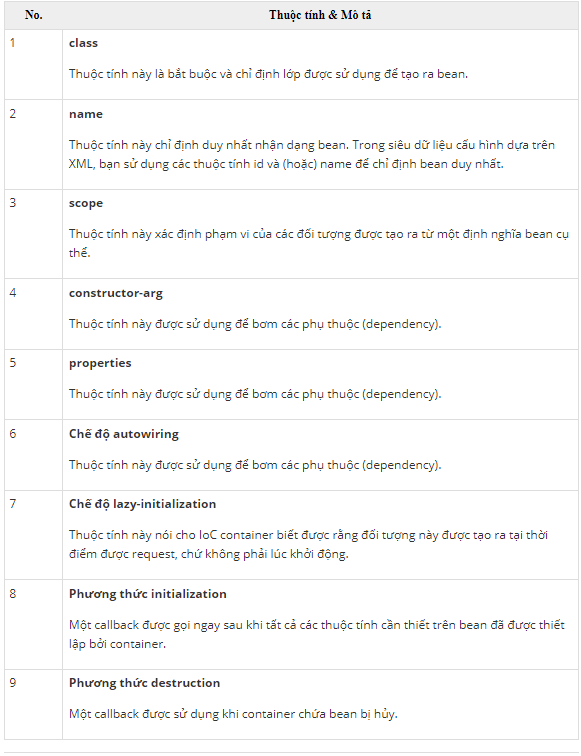
* Quản lí vòng đời, cấu hình, hủy **đối tượng** (Spring bean) bằng cách sử dụng DI
* Có 2 loại:

+ BeanFactory

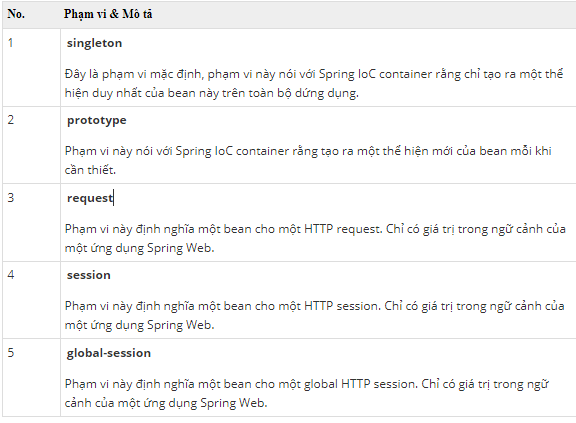
 + ApplicationContext (BeanFactory interface cha của ApplicationContext)



1. Thuộc tính của bean:

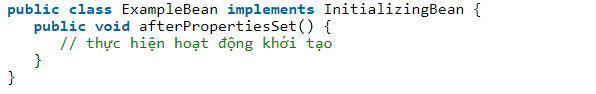


Giá trị scope:

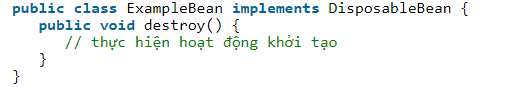


1. Vòng đời 1 Bean

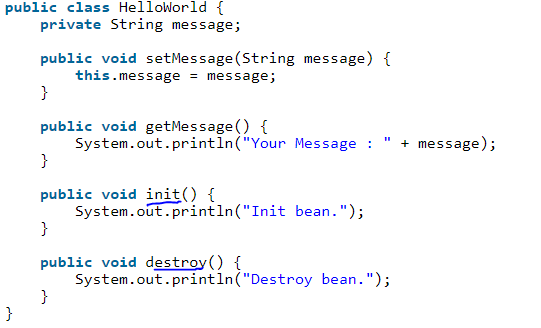
* Action cho khởi tạo:

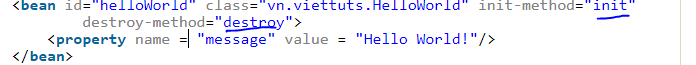


* Action cho hủy:

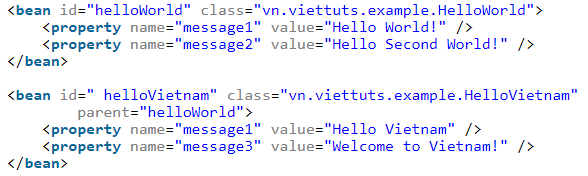


* Cách khác:

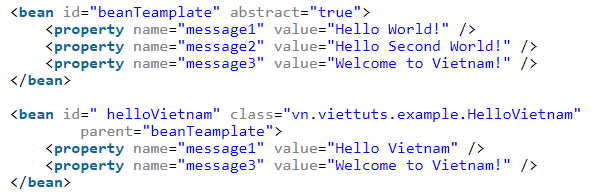


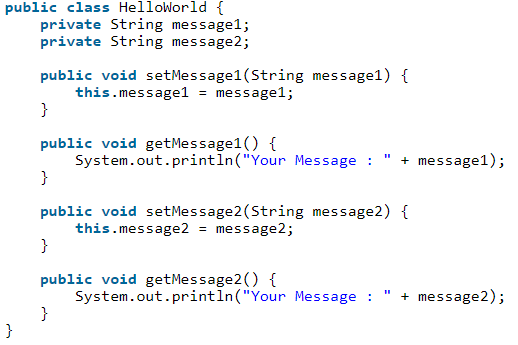


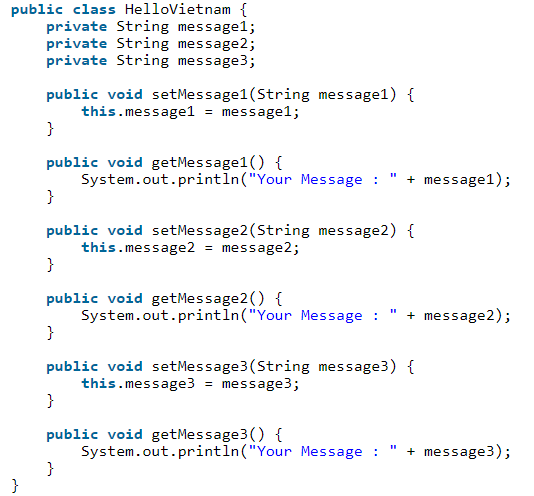
1. Kế thừa thuộc tính trong Bean:

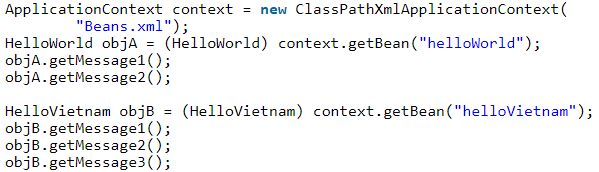


Cách 2 trong Bean:

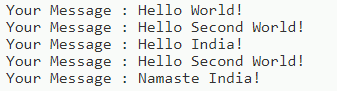




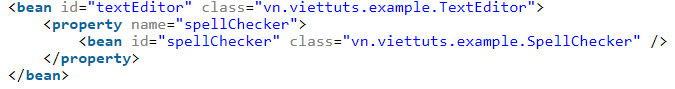
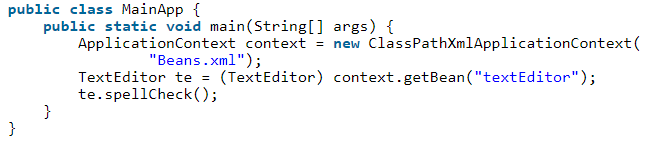
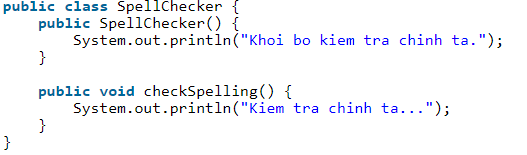
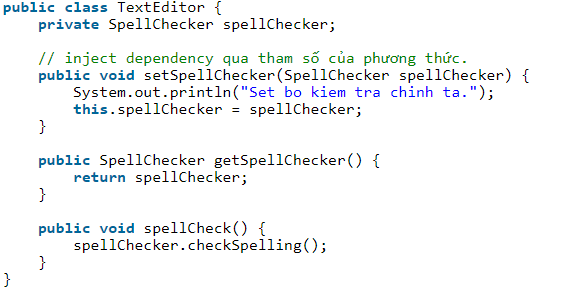




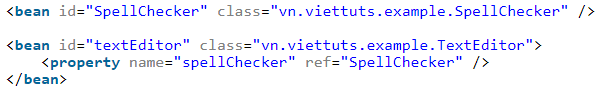
Kết quả:



1. Inner Bean:



Hoặc:



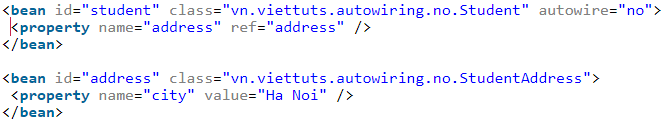
1. Injecting Collection:

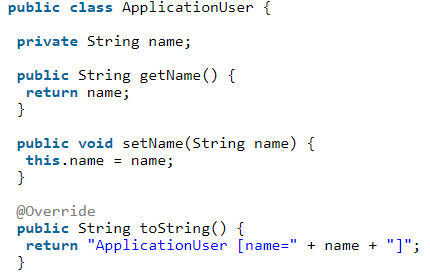


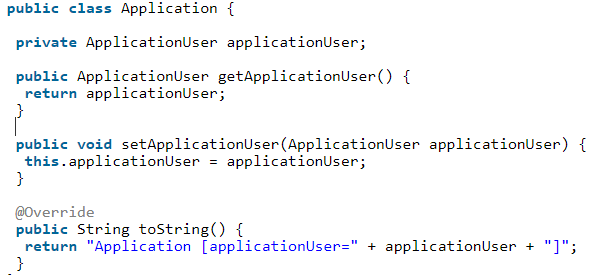
Tham chiếu:



1. Bean Auto Wiring

* autowire=”no”: Mặc định
* autowire=” byName”: Tự động liên kết đến





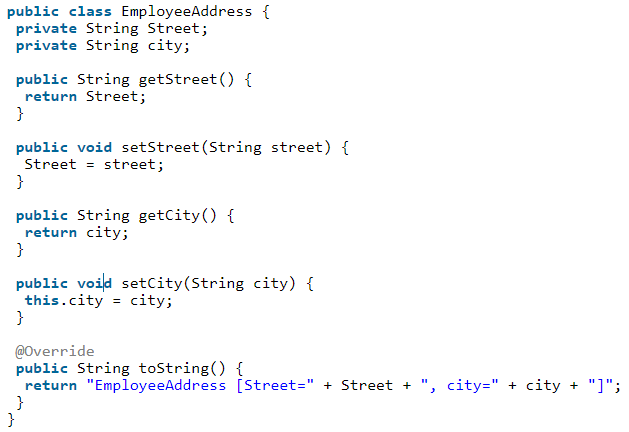
 <bean id="application" **class**="vn.viettuts.autowiring.byname.Application" autowire="byName" />

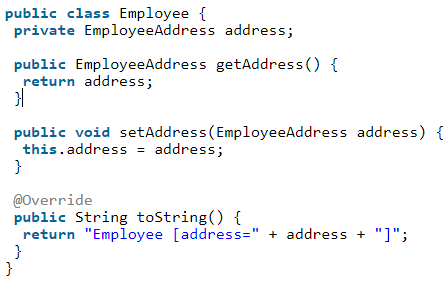
 <bean id="applicationUser" **class**="vn.viettuts.autowiring.byname.ApplicationUser">

  <property name="name" value="superUser" />

 </bean>

* autuwire=”byType”: Ghép dựa vào kiểu dữ liệu





<**bean** id="employee" class="vn.viettuts.autowiring.bytype.Employee" autowire="byType"/>

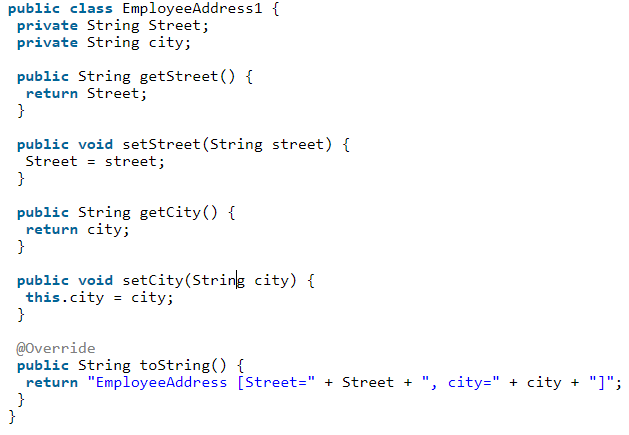
    <**bean** id="employeeAddress" class="vn.viettuts.autowiring.bytype.EmployeeAddress">

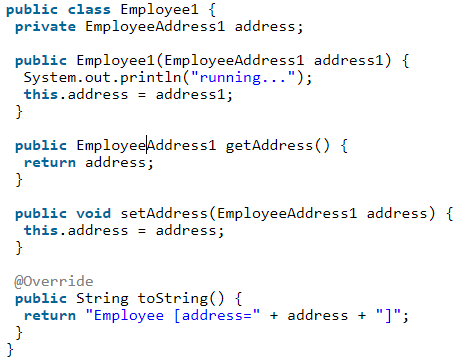
        <**property** name="street" value="333/33 Tran A"/>

        <**property** name="city" value="Ha Noi"/>

    </**bean**>

* autowire=” constructor”:





    <bean id="employee" **class**="vn.viettuts.autowiring.constructor.Employee1" autowire="constructor"/>

    <bean id="address1" **class**="vn.viettuts.autowiring.constructor.EmployeeAddress1" >

        <property name="street" value="444/44 Tran B"/>

        <property name="city" value="Ha Noi"/>

    </bean>

1. Muốn sử dụng annotation cần

Trong Beans.xml cần: <context:annotation-config/>

<?xml version = "1.0" encoding = "UTF-8"?>

<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-3.0.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-3.0.xsd">

<context:annotation-config/>

<!-- Definition for student bean -->

<bean id = "student" class = "com.tutorialspoint.Student">

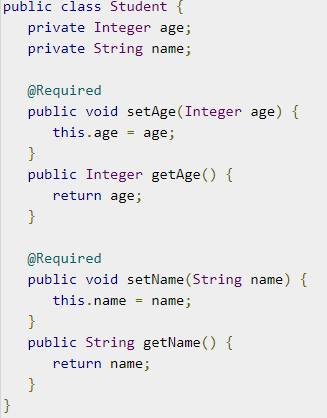
<property name = "name" value = "Zara" />

<property name = "age" value = "11"/>

</bean>

</beans>

1. @Required: Bắt buộc phải có Property đó

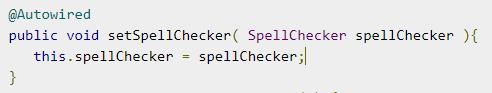


1. @Autowired:

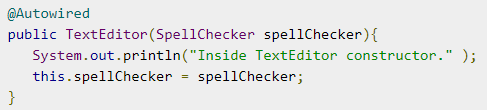
* ByName:



* ByType:

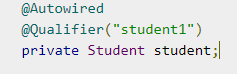


* ByConstructor:

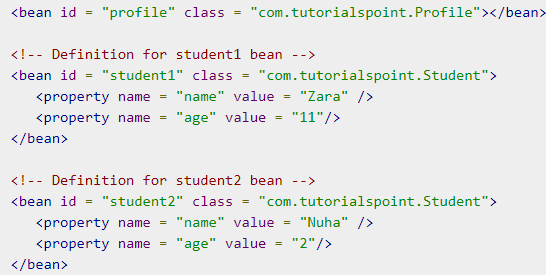


1. @Qualifier: Lựa chọn giá trị cho bean

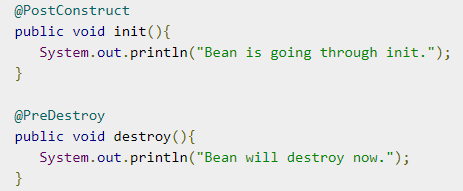
* Trong Profile



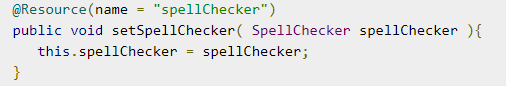
* Trong xml



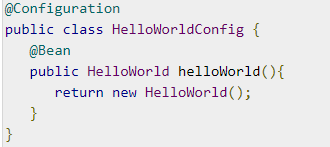
1. @PostConstruct & @PreDestroy:



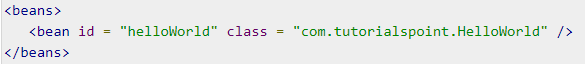
1. @Resource(name = “spellChecker”): Giống Autowire byName



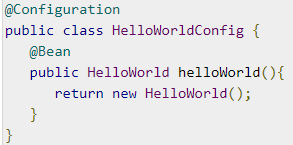
1. @Configuration & @Bean

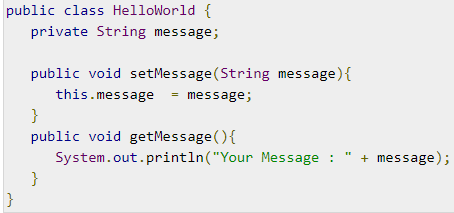


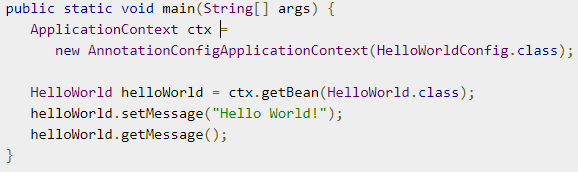
Giống như:



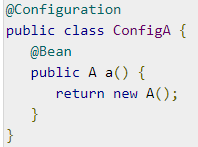
VD:

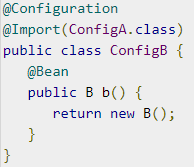


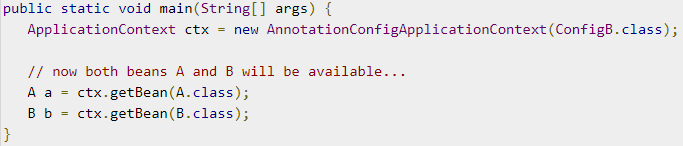




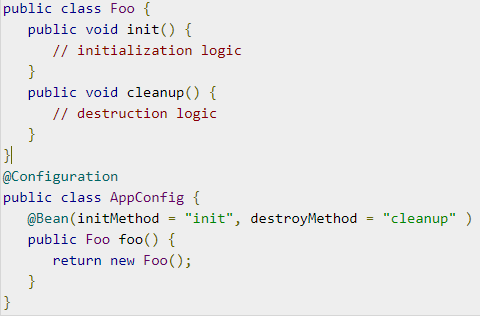
1. @Import: load bean vào một bean khác







1. Destroy and Init:

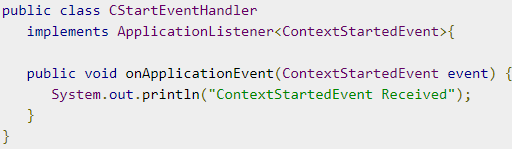


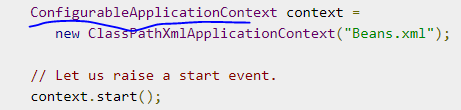
1. @Scope:

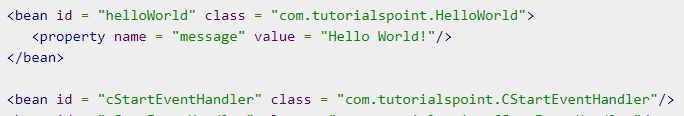


1. EventHandleing: Sự kiện khi thực hiện các hoạt động và context là **ConfigurableApplicationContext**

* **ContextRefreshedEvent:** Khi init bean hoặc dùng refresh() của context
* **ContextStartedEvent:** khi dùng start() của context và có thể restart (gọi lại start() một lần nữa)
* **ContextStoppedEvent:** khi dùng stop() của context, tạm thời dừng và có thể restart
* **ContextClosedEvent:** khi dùng close() của context, và không thể refresh hoặc restart
* **RequestHandledEvent:** khi có http request
* **VD:**







1. AOP

* Là kĩ thuật lập trình tách riêng logic
* Các khái niệm:
* Join point: các điểm xử lí, thực thi
* Advice: action tại join point gồm:

+ Before advice: chạy trước khi method thực hiện

+ After returning advice: chạy sau khi method trả về kết quả

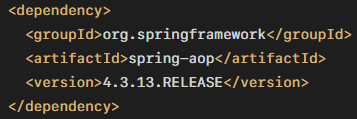
+ After throwing advice: chạy sau khi method ném ra một exception

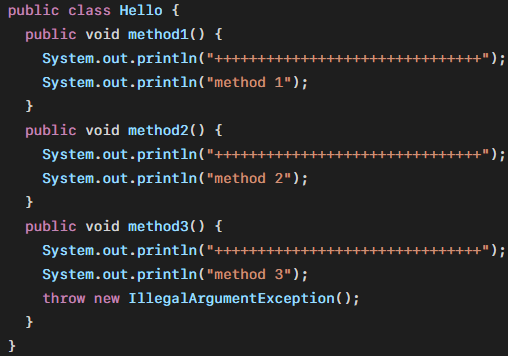
+ Around advice: Chạy khi method thực thi (gồm cả 3 loại advice trên)

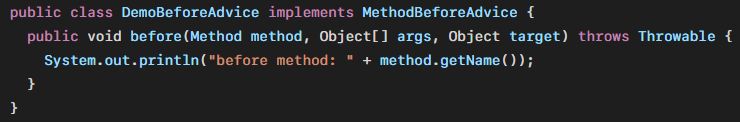
* Pointcut: là expression language giúp khớp nói với joint point
* Introduction: Cho phép introduce các new interface tới bất kì object advice nào
* Target Object: Object sẽ được adviced
* Interceptor: một aspect chỉ có duy nhất một advice
* AOP Proxy: cài đặt các Aspect
* Weaving: tiến trình nói các aspect với các object, types tạo nên adviced object

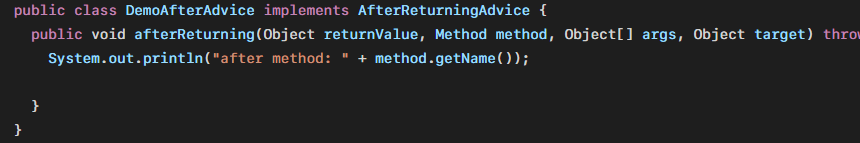
VD:

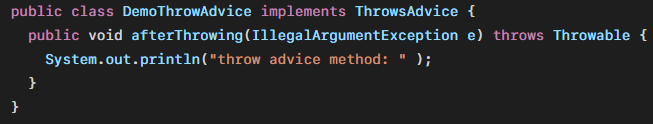
* Cài vào maven

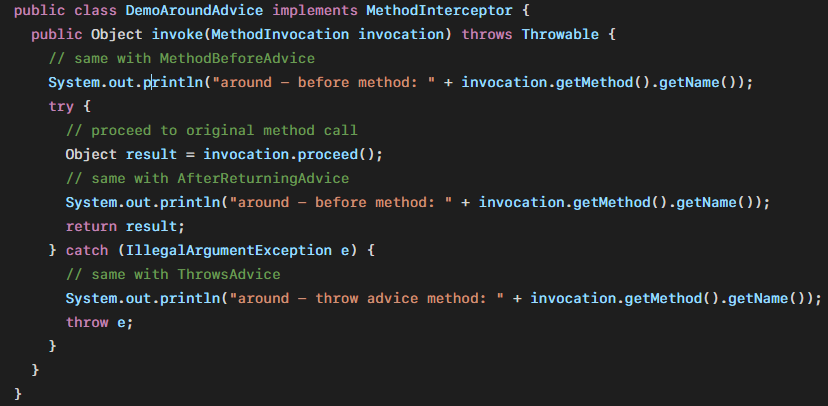


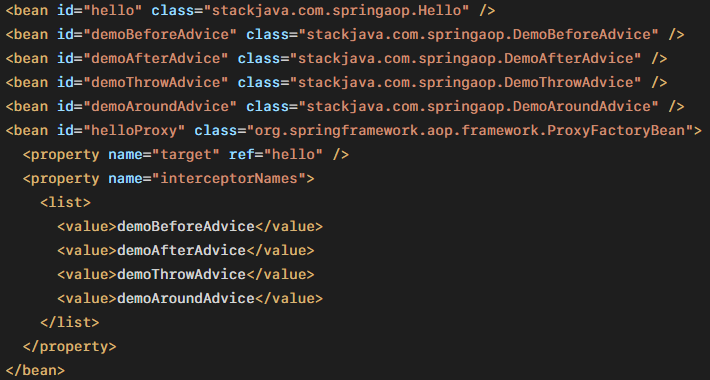






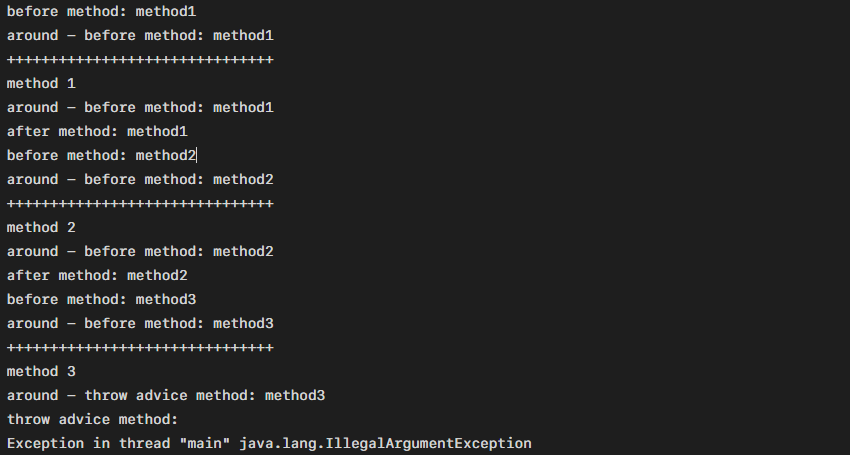








KQ:



1. JDBC Framework:

* Cài đặt driver:

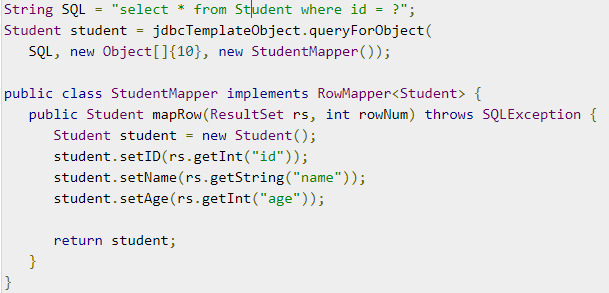


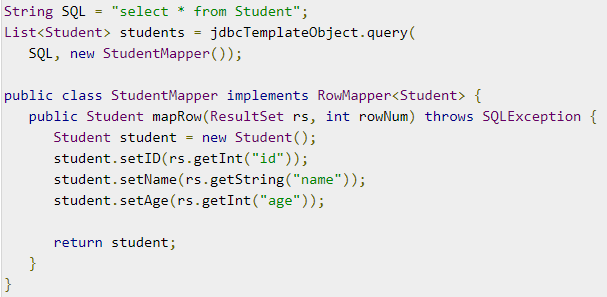
* SQL statement:











* Insert:



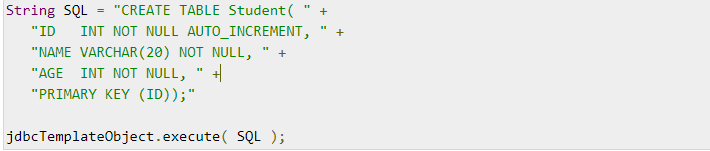
* Update:



* Delete:

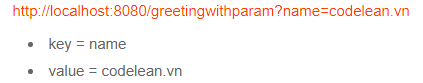


* DDL Statements:



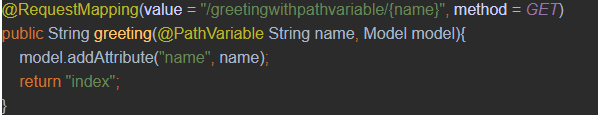
# Spring MVC

1. @RequestParam: Nhận key, value



1. @PathVariable: chỉ nhận value





# Spring Boot

1. **@RestController** = **@Controller** + **@ReponseBody**

=> Định nghĩa controller vs response là **JSON**

1. **@PostMapping** (value = “/abc”) = **@RequestMapping** (value = “/abc”, RequestMethod.POST)

* Định nghĩa phương thức POST với url là /abc, tương tự ta dùng cho các method khác

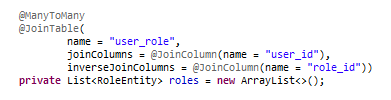
1. **JDBC** có nhược điểm lớn là mỗi **Table** người ta phải build riêng, vd: select \* from user, select \* from product => chỉ khác nhau cái tên nhưng phải viết thành 2 câu lệnh **SQL** khác nhau => Sinh ra **JPA** để tối ưu việc đó
2. Hệ thống sử dụng **SQL Server** tầm 10 năm đổi sang **MySQL** => gần như câu lệnh SQL phải sửa đổi lại hết => **JPA** hỗ trợ câu lệnh **HQL** (Hibernate query language) sử dụng **Entity** cho các class ra đời => PR
3. **@Entity**: Định nghĩa **Entity** trong JPA (**import** javax.persistence.Entity;)

**@Table**: Định nghĩa **Table** trong JPA (**import** javax.persistence.Table;)

1. **@Id** = NOT NULL + PRIMARY KEY => chỉ có method get
2. **@GeneratedValue** (strategy = GenerationType.IDENTITY): tự tăng giá trị
3. **@Column** (name = “field”): Định nghĩa tên trường (nếu không có annotation này thì nó không tạo field này, còn nếu có mà name trong thì tự chuyển (createBy => create\_by trong database)
4. **@MappedSuperclass**: Kế thường các Annotation từ class cha
5. **@ManyToMany, @OneToMany, @ManyToOne, @OneToOne:** Quan hệ giữa các bảng

VD: Many – Many

* Trong class UserEntity:



* Trong class RoleEntity:



VD: One – Many

* Trong NewEntity:



* Trong CategoryEntity:

