1일차

AOP

Aspect Oriented Programming

Add functionality to objects

Declaratively

- Logging, security, transactions, etc…

Spring projects

단순 프레임워크를 돕는 애드온이라고 봐도 된다.

Inversion of Control

The approach of outsourcing the construction and management of objects

Spring Container

Primary functions

Create and manage objects

Inject object’s dependencies

Spring 개발 절차

Spring Beans를 설정한 후

Spring 컨테이너 생성

스프링 컨테이너에서 Beans를 얻는다?

**스프링 프레임워크는 뒤에서 직접 코드를 만든다.**

Constructor Injection 개발절차

1. Defendency interface와 class 를 정의한다.

2. class의 injections을 위한 Constructor를 만든다.

3. Spring config file에 dependency injection을 수정한다.

**Question**

Why do we specify the Coach interface in getBean()?

For example:

Coach theCoach = context.getBean("myCoach", Coach.class);

**Answer**

Behaves the same as getBean(String), but provides a measure of type safety by throwing a BeanNotOfRequiredTypeException if the bean is not of the required type. This means that ClassCastException can't be thrown on casting the result correctly, as can happen with getBean(String).

<https://www.udemy.com/course/spring-hibernate-tutorial/learn/lecture/5215498#questions/3582594>

위에 링크가 시사하는 것 : 같은 property를 가진 파일은 동시에 로딩될 수 없다.

**Q. Setter injection vs Constructor injection**

Constructor-based or setter-based DI?

Since you can mix constructor-based and setter-based DI, it is a good rule of thumb to use constructors for mandatory dependencies and setter methods or configuration methods for optional dependencies. Note that use of the [@Required](https://docs.spring.io/spring/docs/current/spring-framework-reference/core.html#beans-required-annotation) annotation on a setter method can be used to make the property be a required dependency; however, constructor injection with programmatic validation of arguments is preferable.

The Spring team generally advocates constructor injection, as it lets you implement application components as immutable objects and ensures that required dependencies are not null. Furthermore, constructor-injected components are always returned to the client (calling) code in a fully initialized state. As a side note, a large number of constructor arguments is a bad code smell, implying that the class likely has too many responsibilities and should be refactored to better address proper separation of concerns.

Setter injection should primarily only be used for optional dependencies that can be assigned reasonable default values within the class. Otherwise, not-null checks must be performed everywhere the code uses the dependency. One benefit of setter injection is that setter methods make objects of that class amenable to reconfiguration or re-injection later. Management through [JMX MBeans](https://docs.spring.io/spring/docs/current/spring-framework-reference/integration.html#jmx) is therefore a compelling use case for setter injection.

Use the DI style that makes the most sense for a particular class. Sometimes, when dealing with third-party classes for which you do not have the source, the choice is made for you. For example, if a third-party class does not expose any setter methods, then constructor injection may be the only available form of DI.

Bean’s lifecycle : scope

명시적인 스코프가 없으면 singleton으로 가진다.

Singleton

즉, 우리가 코드상으로 getBean을 하더라도 똑 같은 레퍼런스 bean을 return한다.

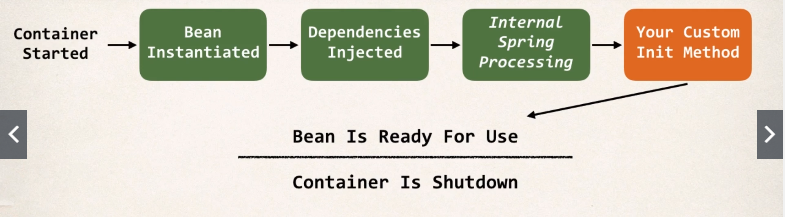
Prototype

매번 계속해서 새로운 class object reference를 return한다.

남은 타입은 웹서버를 위한 것이라 다음 시간ㅇ,,

Spring bean == Java Object

Bean lifecycle



Bean의 생명주기 중 파과와 탄생에서 custom code를 삽입 가능하다.

**Special Note about init and destroy Method Signatures**

When using XML configuration, I want to provide additional details regarding the method signatures of the init-method  and destroy-method .

메소드의 modifier는 아무 상관없음.

리턴 타입도 상관 없으나 return한 값을 확인하지 못하기 때문에 보통 “void”타입을 사용

이름 상관 없음.

Argument는 어떤 값도 받지 못하기 때문에 반드시 없어야 한다!