Java SE + Java EE or Spring

MVC - DI - 느슨한 결합력과 인터페이스

트랜잭션 - AOP

인증과 권한 - Servlet Filter

결합력이 높아 문제

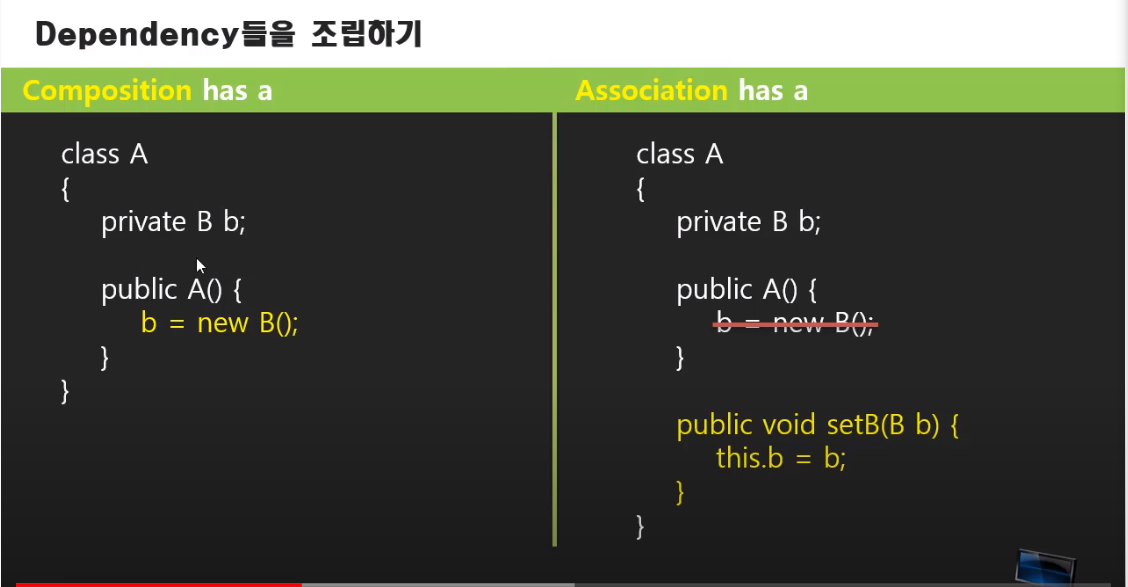
인터페이스 사용으로 결합력을 낮춰 개선

XML 등의 외부 설정을 통해 결합하는 방식(DI) - 새로운 객체로 변경하기

-> 이걸 Spring이 도와줌

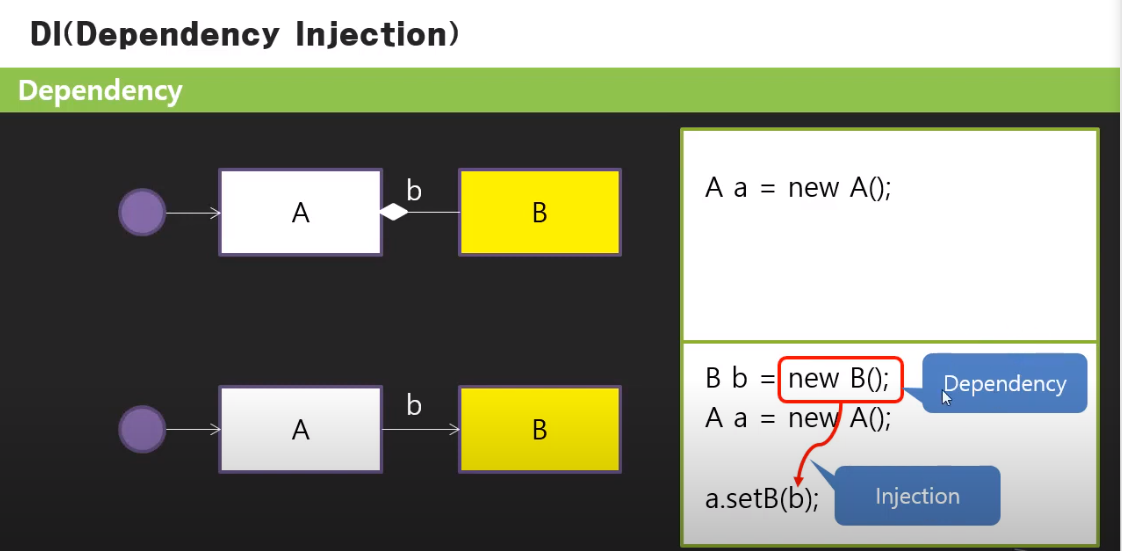
Dependency Injection / IoC Container

DI = 부품 조립



왼쪽 : 부품 일체형, 붙박이

오른쪽 : 부품 조립형, 갈아끼기 용이(기업형)

부품을 만들어서(Dependency) 꽂음(Injection)

방법 : Setter로 조립 / Constructor로 조립

조립을 도와주는 도구 = Spring

결과물만 바로 받을 수 있음

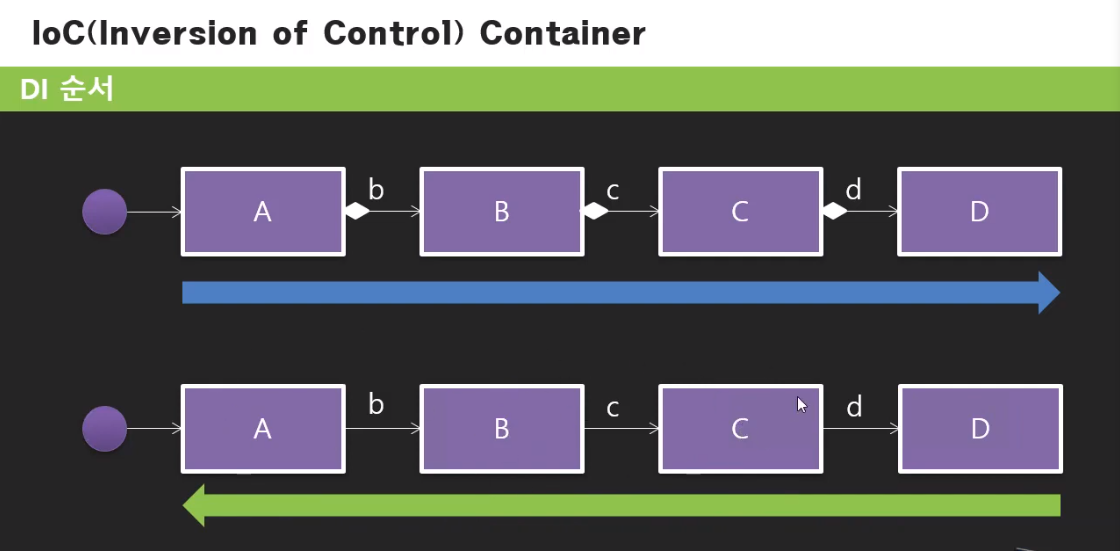
IoC 컨테이너

Spring에 주문서를 제공해서 만들도록 함(DI)

주문서의 예시 - XML, Annotation

주문서대로 만든 재료가 담긴 박스 = IoC 컨테이너(부품 컨테이너, DI 컨테이너)

작은 부품->큰 부품 순으로 만듦

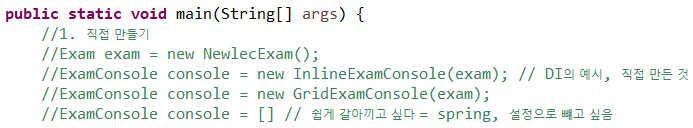


결합까지 역순으로 이뤄짐(Inversion of Control)

멋지게 단어 만듦

맨땅으로 부품을 만들어보기 직접 해보고 스프링을 도입해보기

Maven 강의 추후에 듣기



DI를 직접 만들어봄

이클립스 프로젝트 참조

Spring 프레임워크로 만들기 전 전초 작업

마켓플레이스 접근 시 오류는 무시해도 진행 가능

Help - Eclipse Marketplace 에서 Spring Tools 3 Add-On for Spring Tools 4 3.9.13.CI 설치

Src - {패키지 이름}.di - 오른쪽 클릭 - new - other - Spring Bean Configuration File - Next - 파일 이름 설정(ex. setting.xml)하여 관련 설정 파일 자동으로 추가

실제 DI를 위한 지시서 작성해보기

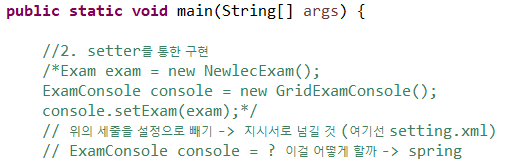
Class 이름명이 겹칠 수도 있으니 package명까지 기재 - package명은 중복을 피하기 위한 이름의 확장판

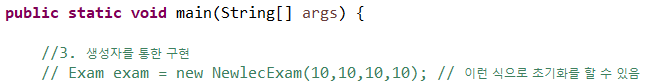
Id는 그 객체를 어떠한 이름으로 쓰는지

Class는 생성한 객체명

Xml을 활용해 지시했다라고 함

setting부도 구현 가능





규칙

- setExam -> (Property Name) ->exam // set 삭제 및 소문자 변경

타입에 따라 value형이냐 ref형이냐에 따라 다르게 넣음

실제 IoC 해보기

xml 파일의 위치에 따라 ApplicationContext 생성

Java Prj을 Maven Prj으로 바꾸기

프로젝트에서 오른쪽 마우스 클릭 - Configure - Convert to Maven Project 선택 - com.newlecture로 Group Id 변경(Maven 강의 듣기)

Dependencies - Dependencies 탭의 Add 클릭

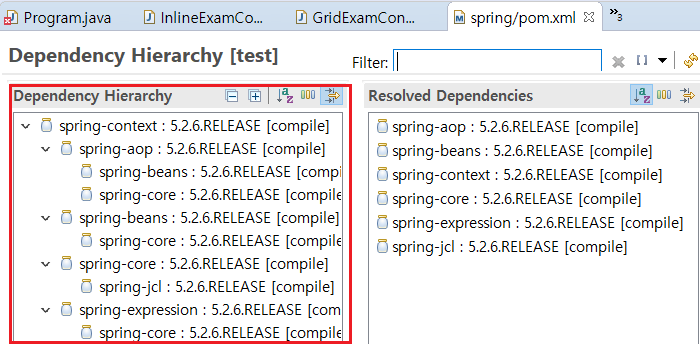
목록이 안 나오는건 당장은 Window - Show View - Others - Maven - Maven Repositories Open

Global Repositories - Central - 오른쪽 마우스 클릭 - Rebuild Index 클릭(시간이 너무 걸림)

Maven Repository 검색 후 mvnrepository.com 사이트 접속 - Spring Framework 검색 - Spring Context - 최신버전의 Maven 내용을 클릭하여 복사

Pom.xml의 version 태그 밑에 해당 내용 붙여넣기

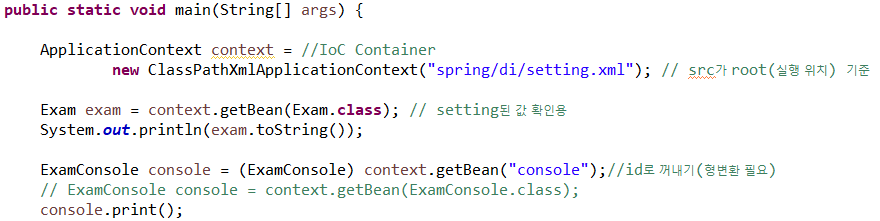
저장하면 필요한 것들을 가져오는 작업을 쭉 함



Dependencies Hierarchy를 확인해보면 필요한 Dependeny들을 모두 가져온 것을 볼 수 있음

IoC 컨테이너 형태(context)로 만들어뒀고 꺼내 쓸 때는 id로 가져올 수도 있고 class로도 가져올 수 있음

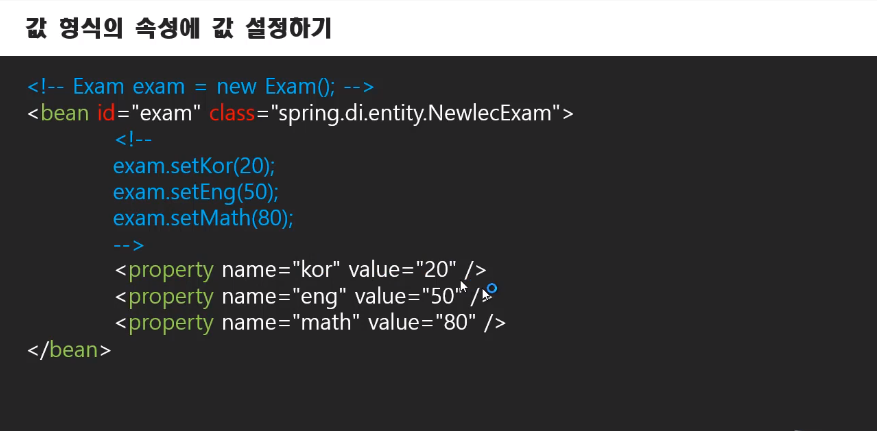
Class로 써서 뭉뚱그려 가져올 수 있으며 추가될 때는 구분하는 명시를 추가로 해주면 됨



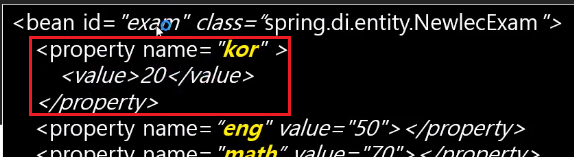
setting.xml 작성 시 ref나 value는 가져올 객체의 이름(id)을 넣어줌(=exam)

설정 파일만 수정하면 소스 코드 수정 없이 다른 형태의 함수 호출 가능

ref형식 말고 value 형식 DI해보기

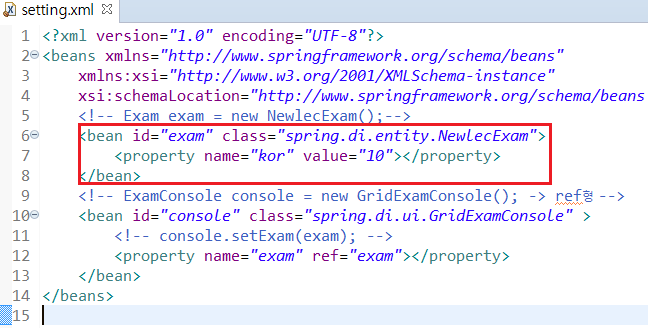


태그를 중첩해서도 설정 가능



Value 설정 이후 실행하면 오류 발생

값을 세팅할 수 있는 Setter를 만들지 않았기 때문 -> NewlecExam에서 setter 만들기

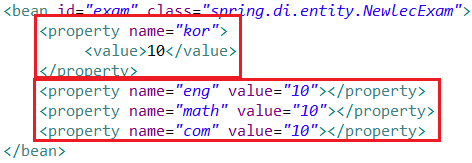


파일 내용 영역에서 마우스 오른쪽 클릭 - Source - Generating Getters and Setter... 클릭

규칙에 의해 kor 이름은 setKor를 자동 호출할 수 있게 됨

value 태그를 중첩해서 값을 세팅할 수도 있고 value 옵션을 통해 세팅할 수도 있다

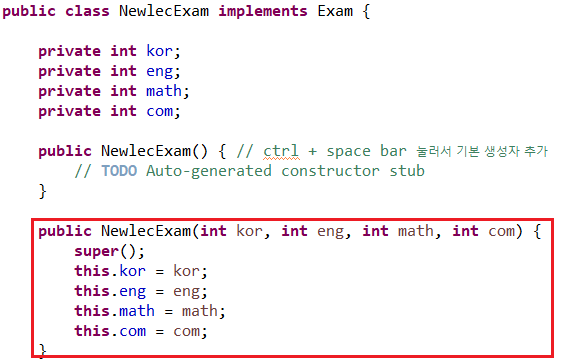
setter를 통한 DI 예시



생성자를 통해서 DI하기

Constructor-arg 속성 이용

오버로드 생성자 생성 : 소스 영역에서 오른쪽 마우스 클릭 - source - Generating Constructor with Fields, 원하는 필드들 선택하고 generate



자동 생성된 Super는 삭제

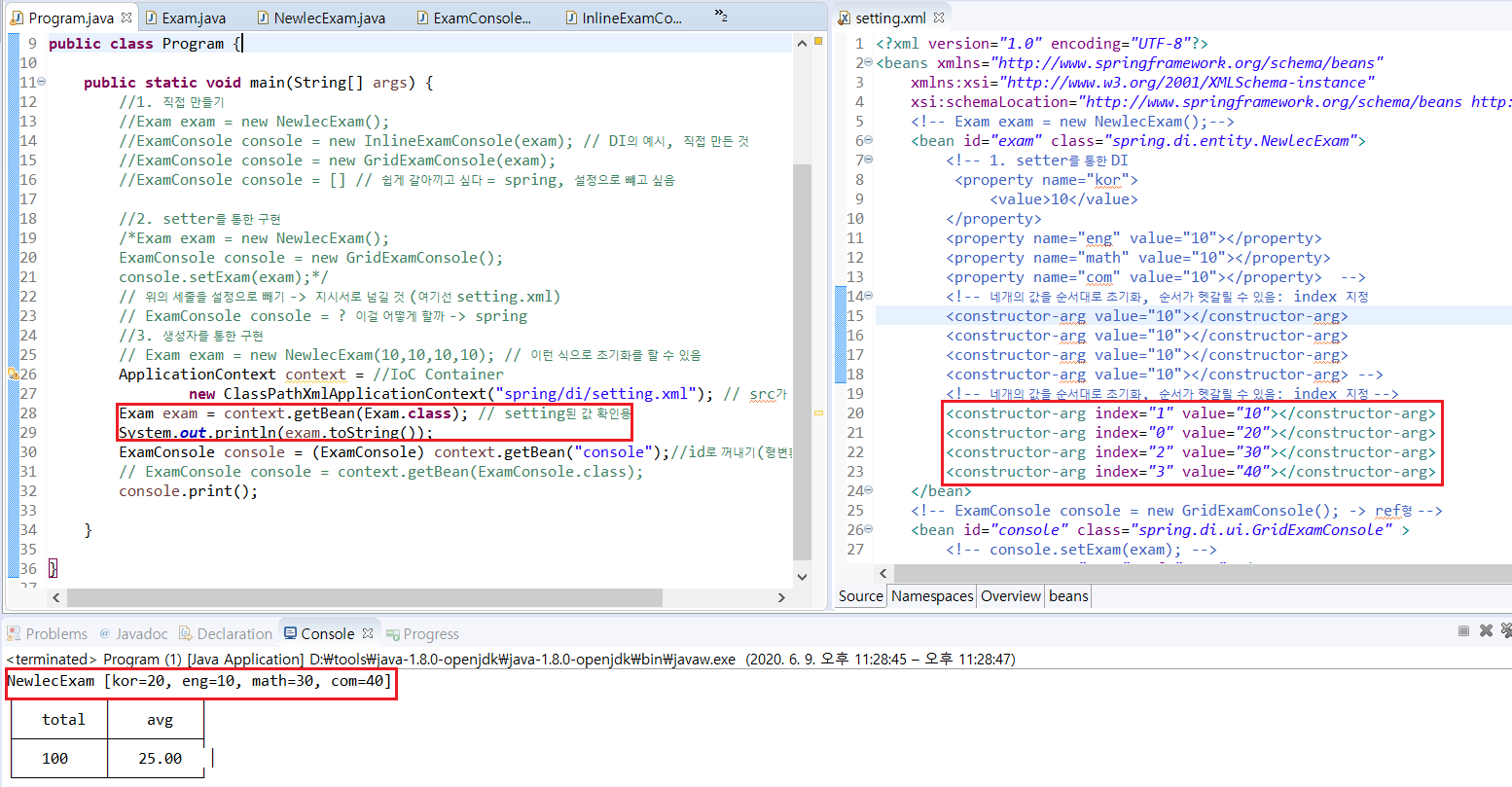
순서대로 입력하여 그대로 생성자를 통해 초기화할 수 있음

헷갈릴 수 있기 때문에 인덱스 지정도 가능

이를 확인하기 위해 toString 함수 구현

자동으로 만드는 방법 : 소스 영역에서 오른쪽 마우스 클릭 - Source - Generating toString() 클릭 - 원하는 Field 선택

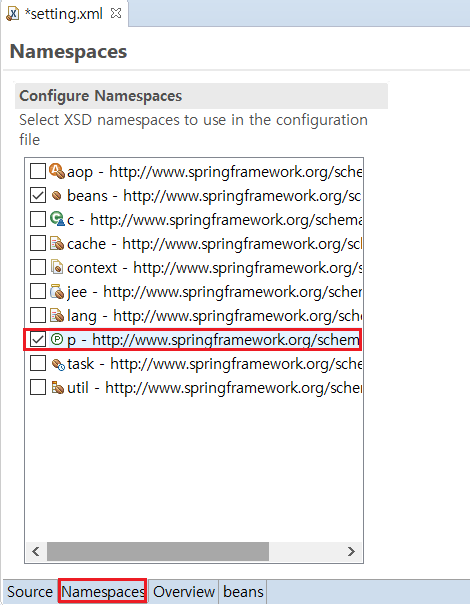
toString()을 활용해서 순서를 바꿔서 세팅한 것을 확인할 수 있음



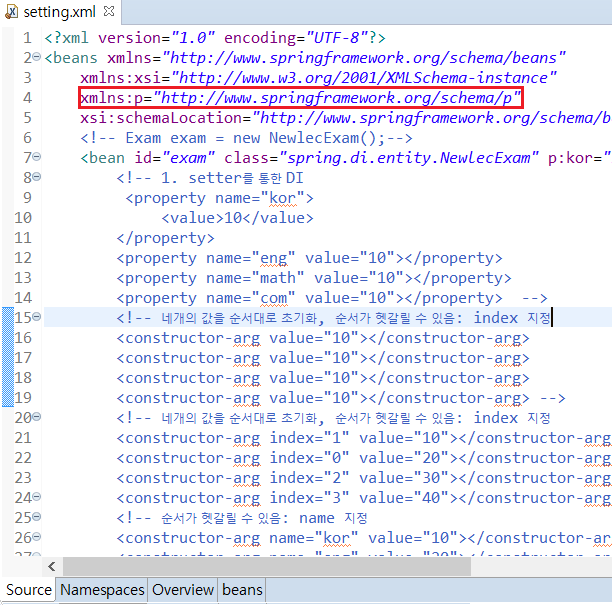
순서대로 넣기, Index 지정을 통해 넣기, name 지정을 통해 넣기, type 지정을 통해 넣기 정리



처리기를 통해 p 속성을 통해 단일 태그에 바로 초기화할 수 있게 하기 위해 namespaces영역에서 p의 체크박스 체크, 오류 메시지는 변경으로 인해 삭제 혹은 추가될 수 있다는 내용으로 그냥 확인 클릭



Namespace가 추가되고 p:kor쪽에 더 이상 오류 뜨지 않음



모듈이라는 단위, 같은 객체명을 가질 수 있으니 이를 구분하는 성과 같은 개념

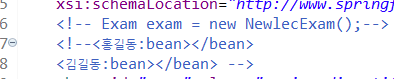
즉, 성이 Namespace, 확장된 이름

Ex. 같은 문서에 bean이 두 개일 수 있음, Namespace를 사용하면 자기가 처리하는 Namespace가 아니면 무시하고 지나감

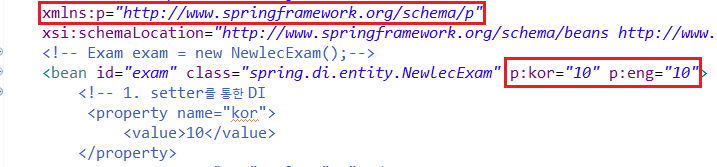
Namespace의 용도

1. 태그가 특정한 처리기에 의해서 처리될 수 있게 함

2. 태그 이름을 식별하기 위해 씀



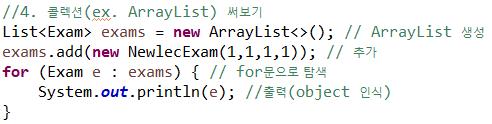
즉P가 담당한 태그이다라는 것을 추가한 것이라고 볼 수 있음



이런 것들을 Bean과 같은 태그 앞에도 쓸 수 있음 ex. p:bean

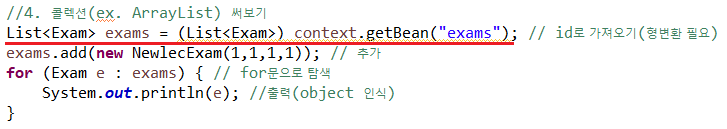
콜렉션 생성과 목록 DI

대표적인 콜렉션 : ArraList



이런 식으로 코드를 짤 수 있는데 setting.xml을 통해서도 만들어보자

Program.java



Setting.xml

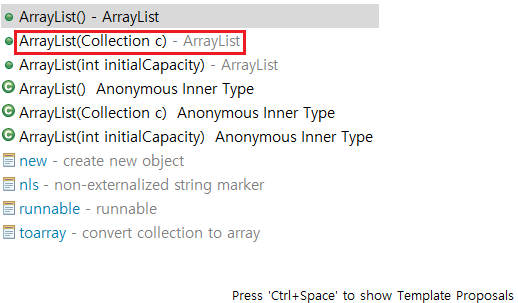


이와 같이 ArrayList 형식 역시 DI를 통해 생성할 수 있음

DI를 통해 콜렉션의 setter 구현

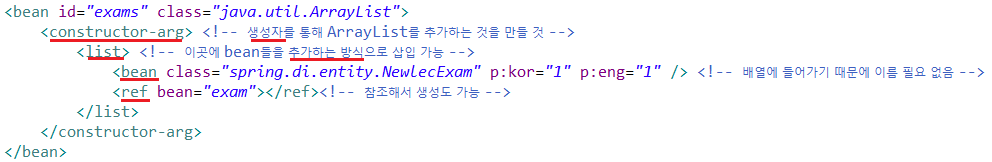
Add는 setter가 아니라 그걸로는 안됨

ArrayList의 사용법을 확인해보면 생성자에 collection을 대입하는 방식으로 객체를 추가할 수 있음

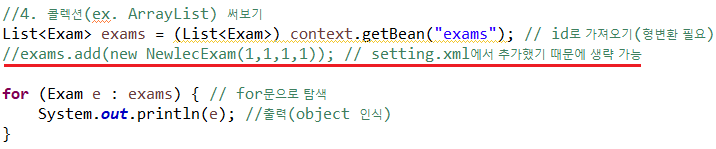


ArrayList 생성자의 사용법을 활용해 DI를 통해 객체들을 생성하고 삽입함

직접 bean들을 입력할 수도 있고 ref 태그를 통해 이미 만들어진 bean을 참조할 수 있음



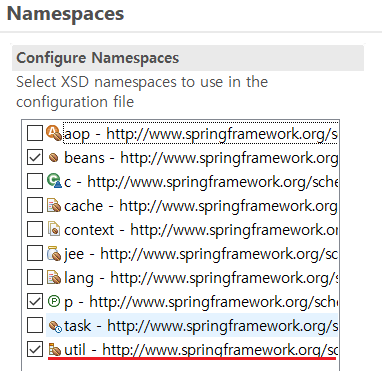
setting.xml에서 추가했기 때문에 생략 가능



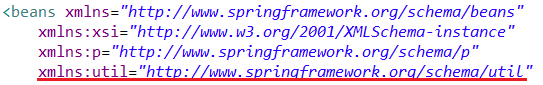
생성자를 통해서 List를 만드는 것이 아닌 별도의 객체로는 만들 수 없을까?

아래와 같은 방법으로 가능

Namespaces에서 util 항목을 체크



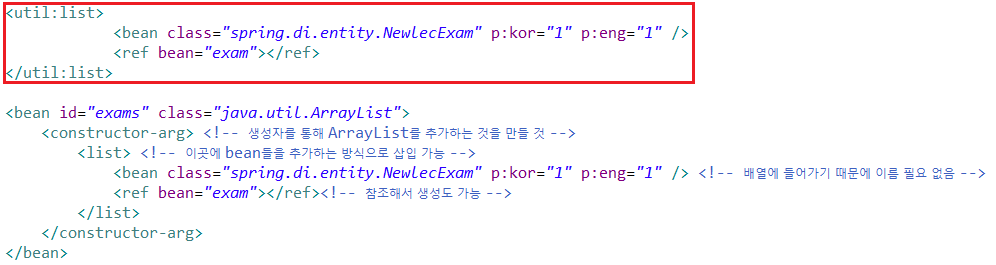
util관련 전처리기가 추가됨



이후 list라는 태그에 util이라는 전처리기 표시를 하여 따로 List 객체를 생성할 수 있음

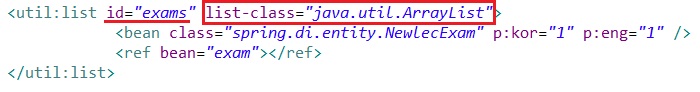
consturctor-arg 안에서 사용되는 태그와는 다름

다른 전처리기를 통해 처리되는 것임



이런식으로 컬렉션을 만들 수 있음

여러 클래스가 있으므로 그것을 지정해줘야 하며 이름도 필요함



이전 코드는 주석처리하여 확인하면 정상 작동됨

어노테이션을 이용할 때의 장점과 @Autowired를 이용한 DI

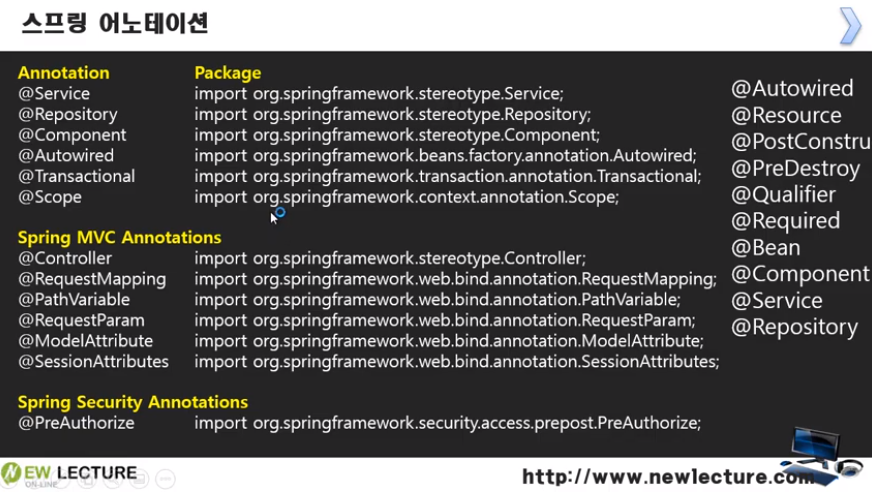
설정 파일을 지정하는 방법

1. 외부에 xml 파일 두기

2. 내부 코드에 Annotation(설정 정보)으로 심기 -> 자바 강의 듣기

DI하는 부분을 Annotation으로 바꿔볼 예정

스프링 관련 어노테이션들



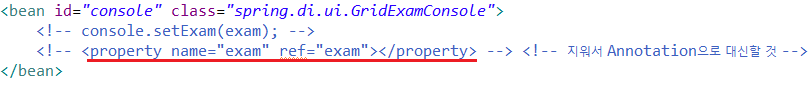
설정 파일을 수정하는 것도 귀찮아짐

Annotation을 사용했을 때 객체를 바꿔버리면 코드도, 설정도 바뀌게 함

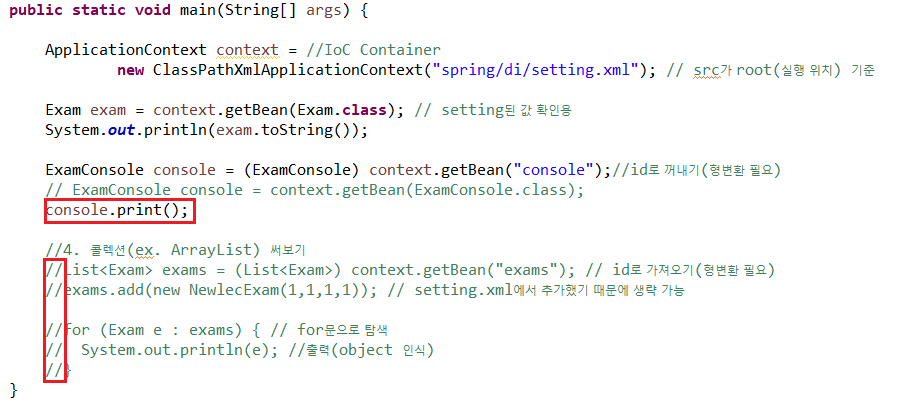
객체 생성은 다음 시간에 배울 것

객체에 객체를 DI하는 것을 일단 어노테이션으로 해볼 것(@Autowired)

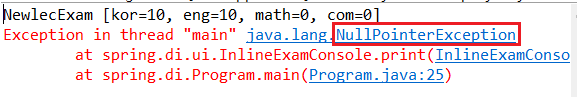
xml 파일쪽 주석 처리



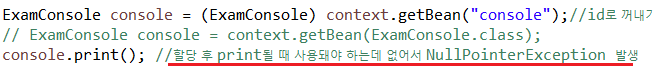
java 파일쪽 주석 처리



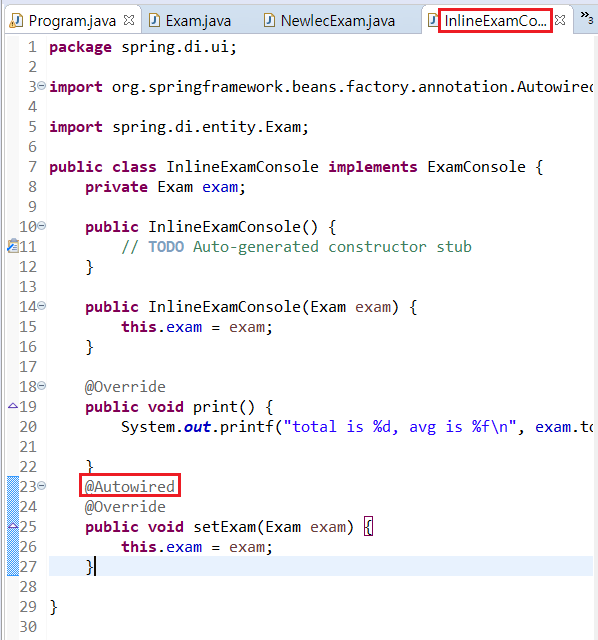
이대로 실행 시 NullPointerException 발생



실제로 DI가 안된 것을 확인한 것임



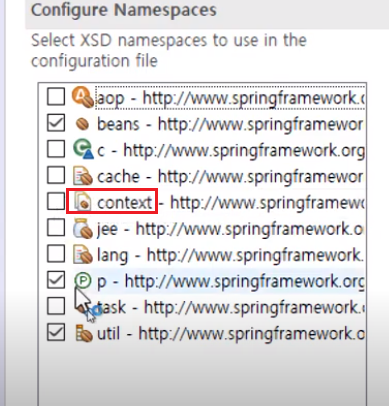
setter쪽에 Annotation을 추가함, 하지만 그래도 NullPointerException이 발생



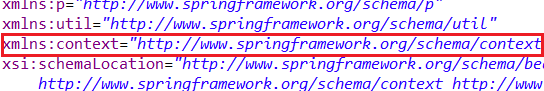
굳이 Class 안에 Autowired Annotation을 찾으려 하지 않기 때문임

지시서에 그걸 찾아달라는 것을 추가해야 함

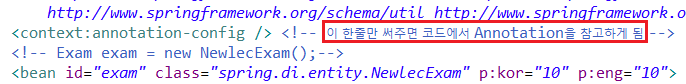
Namespaces에서 context 체크



xml 파일에 다음과 같은 처리기 추가됨

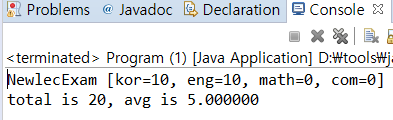


태그 추가



객체들이 Annotation을 갖고 있다는 뜻

정상 실행됨



xml 설정 파일에서 ref 속성을 통해 exam을 참고했던 것과 다르게 Annotation은 뭐를 근거로 바인딩을 한 것일까? -> 다음 시간

@Autowired와 @Qualifier

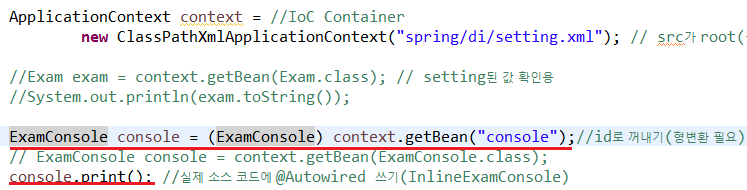
DI를 대신하는 Annotation

그 안을 들여다 봐라 -> Annotation 확인

Exam이라는 객체를 찾아서 자동으로 꽂아라라는 것

무슨 기준일까? 이름 or 객체 타입

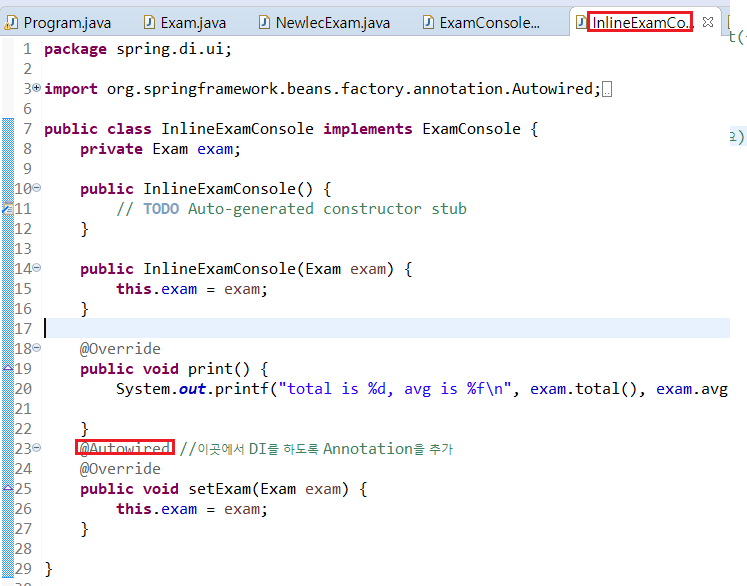
확인을 위해 방해가 되는 코드는 주석 처리하고 두 줄만 남겨둠



설정 파일에 따라 객체 생성 이후 그냥 지나가지 않고 객체 내부에 Annotation이 있는지 확인하게 됨

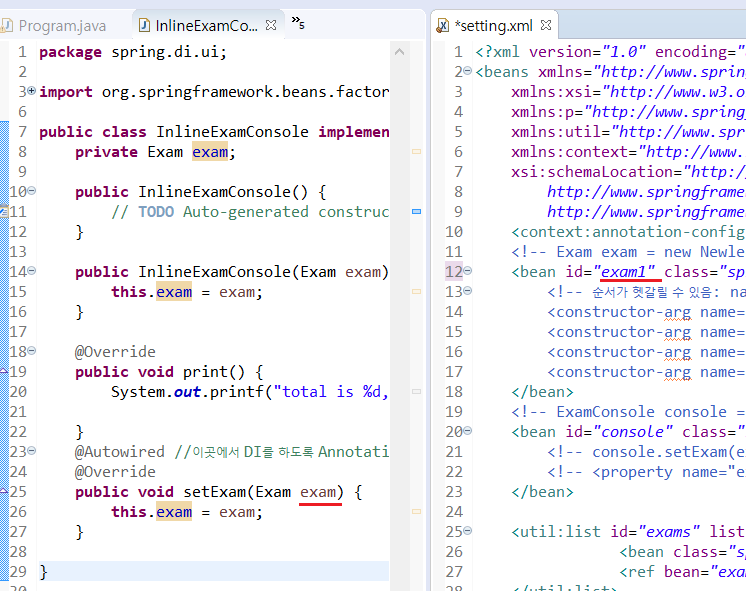


해당되는 함수에 @Autowired Annotation을 작성해줘 자동으로 DI를 할 수 있도록 함

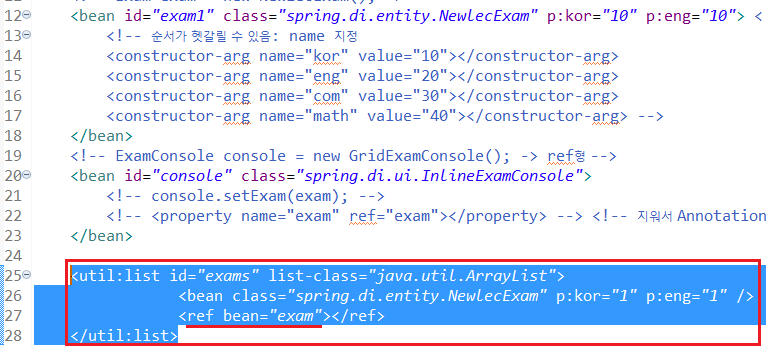


위의 상태로 실행하면 잘 동작하는 것 확인

변수명을 토대로 검사하는지 확인용

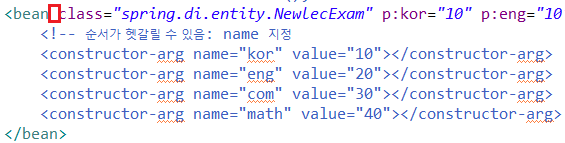


에러가 나면 안되지만 에러가 나게 됨, 참조를 하는 지시서의 내용이 있기 때문임, 해당 부분을 삭제하여 테스트



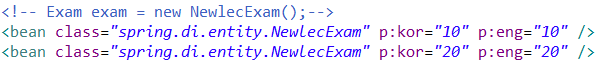
이후 확인해보면 정상 작동하는 것을 볼 수 있음 -> 이름이 아닌 객체 타입으로 구분한다는 결론

심지어 이름(id)을 아예 없어도 바인딩이 잘됨



Exam이라는 인터페이스와 부합/참조할 수 있는 객체를 자동으로 찾아와 바인딩을 해주는 것임

아래와 같이 같은 객체를 생성하려고 하려면 만족할 수 있는 디펜던시를 찾다가 두개가 나와 되지 않게 됨



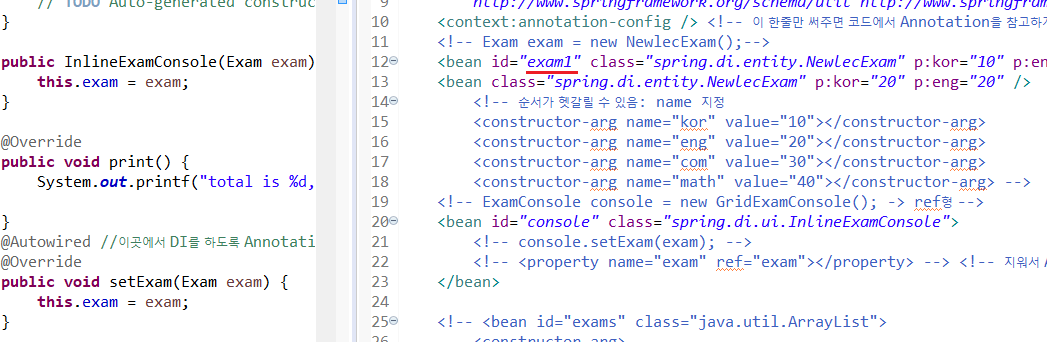
#0과 #1 객체가 있어 못해먹겠다는 에러가 발생함

그럼 이름을 지정해주면 어떨까?



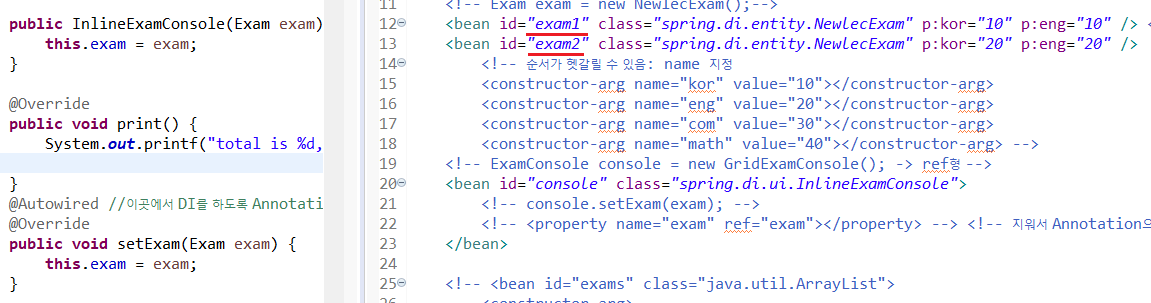
모호한 객체에 위와 같이 이름으로 구분해주면 정상 동작함

이름을 다시 구분하지 못하는 것으로 바꾸면 에러가 나는 것을 확인할 수 있음



구분할 때 먼저 객체의 형식으로 구분하고 그 구분이 모호하면 이름(변수명)을 통해 구분 짓는다

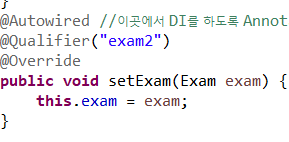
따라서 이 상태에서도 오류가 나기 마련임, 객체는 동일하기 때문 -> @Qualifier Annotation을 통해 해결



객체가 같을 때 어떤 아이를 쓸꺼다라는 것을 지정, exam1에 대한 결과가 나옴(total 20)



exam2로 바꾸면 total 40으로 나옴



Autowired가 자동으로 DI해주는 좋은 것인데 기본적으로 어떤 것을 통해 구분하는지 알아봤다. 자료형식이 기본 이름으로 구분 되는데 함수의 변수명으론 구분하는 게 한계가 있으니 Qualifier로 구분한다