자바 스프링 프레임워크

가로선

# 2022년 6월 3일

# 스프링 프레임워크는 DI, AOP, MVC, JDBC를 제공한다.

* DI란 외부에서 두 객체 간의 관계를 결정해주는 디자인 패턴으로, 인터페이스를 사이에 둬서 클래스 레벨에서는 의존관계가 고정되지 않도록 하고 런타임 시에 관계를 다이나믹하게 주입하여 유연성을 확보하고 결합도를 낮출 수 있게 해준다.
* AOP는 Aspect Oriented Programming의 약자로 관점 지향 프로그래밍이라고 불린다. 관점 지향은 쉽게 말해 어떤 로직을 기준으로 핵심적인 관점, 부가적인 관점으로 나누어서 보고 그 관점을 기준으로 각각 모듈화하겠다는 것이다. 여기서 모듈화란 어떤 공통된 로직이나 기능을 하나의 단위로 묶는 것을 말한다.
* MVC 패턴은 이름에서도 알 수 있듯이 모델(Model), 뷰(View), 컨트롤러(Controller) 세 개의 컴포넌트로 이루어졌다. 각 컴포넌트는 고유한 역할을 수행한다.
* JDBC(Java Database Connectivity)는 [자바](https://ko.wikipedia.org/wiki/%EC%9E%90%EB%B0%94_(%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%B0%8D_%EC%96%B8%EC%96%B4))에서 [데이터베이스](https://ko.wikipedia.org/wiki/%EB%8D%B0%EC%9D%B4%ED%84%B0%EB%B2%A0%EC%9D%B4%EC%8A%A4)에 접속할 수 있도록 하는 [자바 API](https://ko.wikipedia.org/wiki/%EC%9E%90%EB%B0%94_API)이다. JDBC는 데이터베이스에서 자료를 쿼리하거나 업데이트하는 방법을 제공한다.

# 스프링 컨테이너

스프링에서 객체를 생성하고 조립하는 컨테이너(container)로, 컨테이너를 통해 생성된 객체를 빈(Bean)이라고 부른다.

xml에서 생성한 객체 여러개를 담고 있는 그릇 - 컨테이너(이 안의 객체를 빈이라고도 함)

# 

# 2022년 6월 5일

# Maven

Java 기반 프로젝트의 라이프사이클 관리를 위한 빌드 도구. 이에 따라 컴파일과 빌드를 동시에 수행, 테스트를 병행하거나 서버 측 Deploy 자원을 관리할 수 있는 환경을 제공한다. 또한 라이브러리 관리 기능도 내포하고 있다. Java로 개발하다 보면 다양한 라이브러리를 필요로 하게 되는데, settings.xml 또는 pom.xml 파일에 필요한 라이브러리만 적으면 Maven이 알아서 다운받고 설치해주고 경로까지 지정해준다.

### 장점

* 컴파일과 빌드를 동시에 수행할 수 있다.
* 서버의 Deploy 자원을 관리할 수 있는 환경을 제공한다.
* pom.xml 파일을 통해 관리하므로 개발, 유지보수 측면에서 오픈소스 라이브러리, 프로젝트 등 관리가 용이하다.
* IDE에 종속된 부분들을 제거할 수 있다.
* Maven Profile 기능을 통해 배포 설정 파일을 관리하고 배포 파일을 생성할 수 있다.

### 단점

* Maven에서 기본적으로 지원하지 않는 빌드 과정을 추가해야 하는 경우 상당한 고생이 따른다.
* 특정 플러그인이 설정이 약간만 달라도 해당 설정을 분리해서 중복 기술할 때가 발생한다. 불필요하게 설정이 길어지고 중복, 가독성 저하가 발생하여 유지보수성을 떨어뜨린다.
* 이와 같은 단점을 해결하기 위해, [Gradle](https://namu.wiki/w/Gradle)(그레이들)이라는 새로운 빌드 툴이 등장하였다. [Gradle](https://namu.wiki/w/Gradle)은 [안드로이드](https://namu.wiki/w/%EC%95%88%EB%93%9C%EB%A1%9C%EC%9D%B4%EB%93%9C(%EC%9A%B4%EC%98%81%EC%B2%B4%EC%A0%9C)) 애플리케이션의 기본 빌드 툴로 채택되었다.

# pom.xml

* 메이븐 설정파일

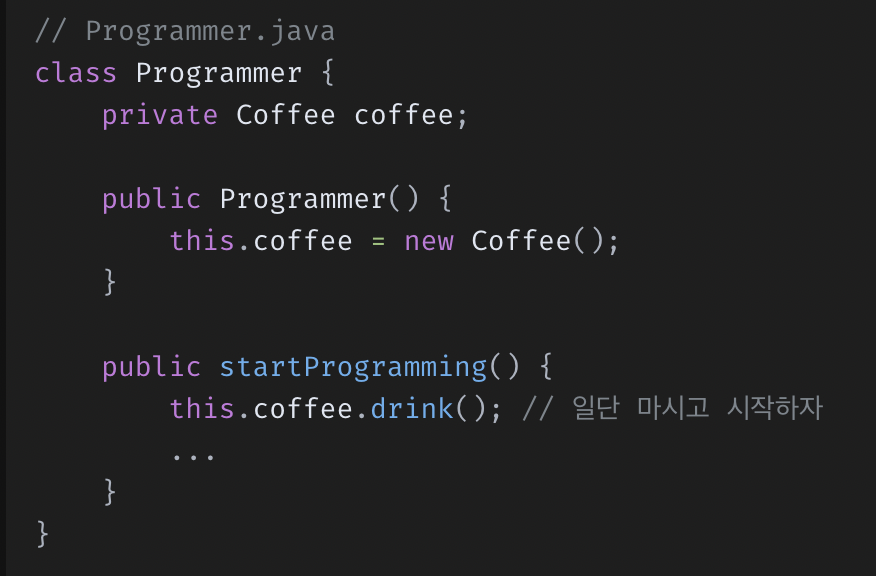
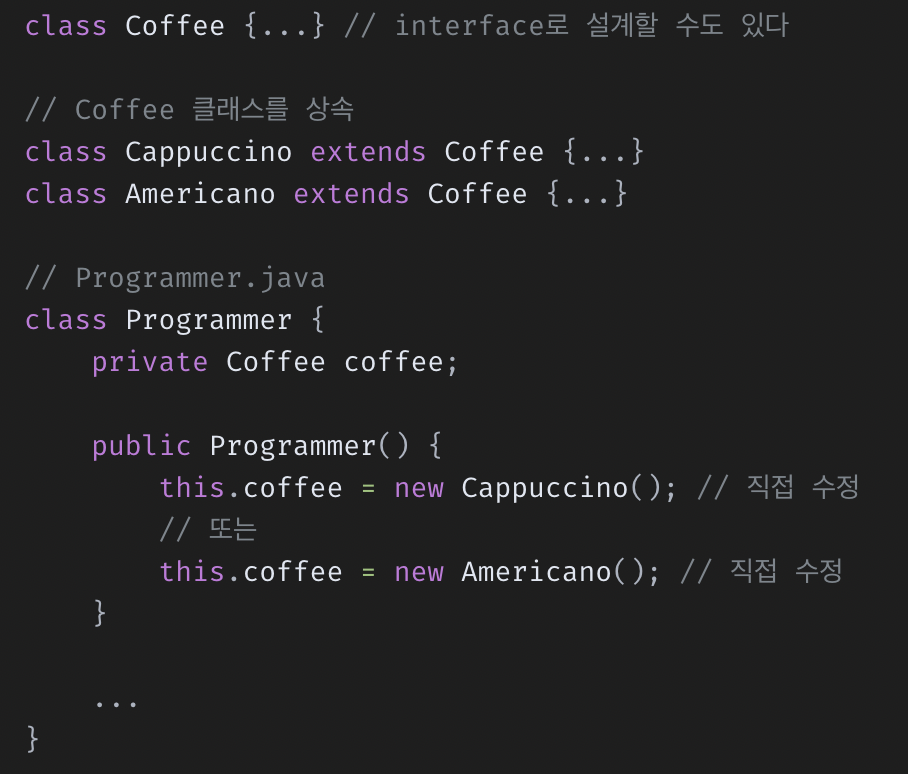
## 외부 xml 사용 방법

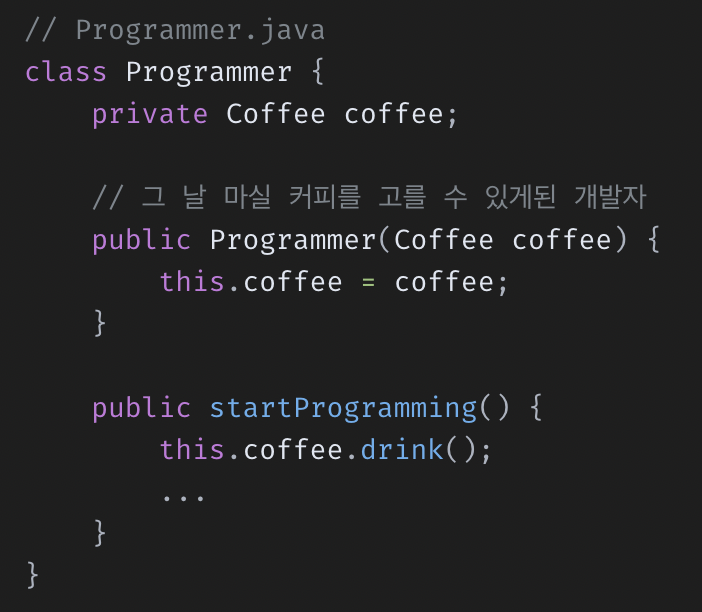
1. xml 파일에 bean 생성  
    <**bean** id=*"tWalk"* class=*"testPjt.TranspotationWalk"* />
2. GenericXmlApplicationContext를 사용해 xml 위치 명시  
    GenericXmlApplicationContext ctx =  
    new GenericXmlApplicationContext("classpath:applicationContext.xml");
3. getBean을 이용해 해당 객체 사용  
    TranspotationWalk transWalk = ctx.getBean("tWalk", TranspotationWalk.class);

=> 어노테이션으로 이용 가능

# DI(Dependency Injection)

## DI란?

* DI(의존성 주입)은 필요한 객체를 직접 생성하는 것이 아닌 외부로 부터 필요한 객체를 받아서 사용하는 것이다.

coffee 객체를 programmer에서 직접 생성 시 coffee 클래스가 변경 될때 마다 직접 바꿔줘야함 => 결합도가 높음

외부에서 받아오면 원하는 클래스를 유동적으로 변경 가능

## 스프링에서 DI

### 생성자를 이용한 의존성 주입

<bean id="studentDao" class="ems.member.dao.StudentDao" ></bean>

<bean id="registerService" class="ems.member.service.StudentRegisterService">

<constructor-arg ref="studentDao" ></constructor-arg>

</bean>

=> constructor-arg를 통해 registerService에 studentDao 의존성 주입

### setter을 이용한 의존성 주입

<bean id="dataBaseConnectionInfoDev" class="ems.member.DataBaseConnectionInfo">

<property name="jdbcUrl" value="jdbc:oracle:thin:@localhost:1521:xe" />

<property name="userId" value="scott" />

<property name="userPw" value="tiger" />

</bean>

### List 형태를 이용한 의존 객체 주입

<property name="developers">

<list>

<value>Cheney.</value><value>Eloy.</value> <value>Jasper.</value> <value>Dillon.</value> <value>Kian.</value>

</list>

</property>

### Map 형태를 이용한 의존 객체 주입

<property name="administrators">

<map>

<entry>

<key>

<value>Cheney</value> </key>

<value>cheney@springPjt.org</value>

</entry>

<entry>

<key>

<value>Jasper</value>

</key>

<value>jasper@springPjt.org</value>

</entry>

</map>

</property>

# 스프링 설정 파일 분리

배열을 이용해서 여러개 xml파일 삽입 가능

xmlArray = {“classpath:tx1.xml”, “classpath:tx2.xml};

GenericXmlApplicationContext ctx =  
 new GenericXmlApplicationContext(xmlArray);

# 

# bean의 범위

싱글톤: 자바처럼 new Class를 사용하는게 아니라 getBean을 이용해 호출하면 xml에 의해 이미 생성된 객체를 호출한다.

⇔

프로토타입: 자바처럼 새로운 객체를 계속 생성하려면 프로토타입으로 선언

<bean id=”dependencyBean” class="scope.ex.DependencyBean" scope=”prototype” >

# 생성자

: 생성자는 간단하게 얘기하면 인스턴스가 생성될 때 호출되는 '인스턴스 초기화 메소드'이다.

따라서 인스턴스 변수의 초기화 작업에 사용되고, 인스턴스 생성 시에 실행되어야 할 작업을 위해서 사용된다.

생성자가 하나도 없을 시 자동으로 생성된다.

## 매개변수가 있는 생성자

class Data2 {

int value;

Data2(int x) { // 매개변수가 있는 생성자.

value = x;

}

}

-> 기본 생성자가 없이 매개변수가 있는 생성자만 있으면 에러.

lass Data2 {

int value;

Data2() {} // 기본 생성자 필요

Data2(int x) { // 매개변수가 있는 생성자.

value = x;

}

}

단, 스프링에서 생성자에 @Autowired 사용 시 기본 생성자 불필요.

# 의존객체 자동 주입

xml에 annotation-config 설정

## @Autowired

객체의 타입이 일치하는 객체를 자동으로 주입한다.

생성자, 프로퍼티, 메소드 모두 사용 가능.

@Autowired(required=false)를 이용해 의존 객체가 없어도 예외 처리 가능

## @Resource

객체의 이름이 일치하는 객체를 자동으로 주입한다.

생성자에는 사용 불가.

## @Qualifier

같은 객체 중에서 사용할 Bean 설정.

@Autowired

@Qualifier(“userDao”)

## @Inject

required속성이 없는 Autowired와 동일