

# 스프링

강사 : 강병준

# EJB의 등장과 쇠퇴

## 1. EJB의 등장과 쇠퇴

- 분산처리와 트랜잭션의 융합컴포넌트  
분산된 **EJB**컴포넌트를 마치 하나의 데이터베이스만 있는 것처럼 제어
- **EJB**란 원래 분산환경을 위한 컴포넌트로서 등장했고  
**JSP, Servlet**과 같은 웹어플리케이션 기술은 아니었음
- **EJB**는 웹 어플리케이션용 재사용 가능한 컴포넌트나  
**SQL**기술이 필요없는 **DB**프레임워크처럼 사용
- **JSP**와 **Servlet**으로 프레젠테이션을 구현하고  
비즈니스 로직은 **EJB**로 구현하는 것을 웹 어플리케이션의 권장 설계
- **EJB**는 본래 분산용이지만 대부분은 웹은 로컬을 쓰므로  
개발과 테스트가 복잡하게 됨
- 따라서 **1990**년 말에 스프링이 등장하여 발전됨
- **Java EE**로 단장한 **EJB3**부터는 스프링+하이버네이트와  
비슷한 사양으로 등장했으나 이미 사용자층은 스프링으로 이동한 상태임

# 스프링등장 배경

## 2. 스프링의 역사

- 복잡함에 반기를 들어서 나온 프레임워크
- 플랫폼 호환성 문제 해결
- 스프링은 **JavaEE**가 제공하는 다수의 기능을 지원하고 있기 때문에 **JavaEE**를 대체
- 스프링은 **IoC(제어의 역전)**을 지원
- 스프링은 **DI(dependency Injection)**패턴을 지원
- 스프링은 **AOP(Aspect Oriented Programming)**를 지원
- 스프링은 **POJO(Plain Old Java Object)**를 지원
- 트랜잭션 처리를 위한 일관된 방법을 제공
- 영속성과 관련된 다양한 **API**를 제공
- **MyBatis, Hibernate, JPA**등과 같은 프레임워크와의 연동을 지원
- 편리한 **MVC** 구조를 가지고 있고 **WAS**에 종속되지 않는 개발 환경
- **2.5**버전에서 어노테이션을 이용한 설정 지원
- **3.0**에서는 **Java** 클래스를 이용한 설정 지원
- **4.0**에서는 모바일과 웹 환경에서 **REST** 방식의 **Controller** 지원

## 2. 스프링의 사용

- 자바로 큰 어플리케이션을 만들 때에 사용하는 프레임워크
- 구현 뿐 아니라 설계도 도움이 되는 프레임워크
- 스프링은 **Rod Johnson**을 중심으로 개발 **Java/Java EE** 프레임 워크

# 스프링 특징

1. 스프링 – 엔터프라이즈 어플리케이션에서 필요로 하는 기능을 제공하는 프레임 워크, **EJB**가 테스트가 힘들고 개발속도가 늦어서 대체 수단으로 등장
2. 특징
  - 1) 경량 컨테이너, 자바의 생성 소멸과 같은 라이프사이클 관리
  - 2) **POJO(Plain Old Java)**
  - 3) **transaction**을 처리하기 위한 일관된 방법 제공 : **JDBC**를 사용하든 **JTA**를 사용하든 또는 컨테이너가 제공하는 트랜잭션을 사용하든 설정파일을 통하여 트랜잭션 정보를 입력
    - ※ . 자바 트랜잭션 **API(Java Transaction API, JTA)**는 **XA** 리소스(예, 데이터베이스) 간의 분산 트랜잭션을 처리하는 자바 **API**
  - 4) 연속성과 관련된 다양한 **API**지원 – **JDBC**, **iBATIS**, 하이버네이트, **JPA** 등 데이터베이스 처리라이브러리 연동 지원
  - 5) 다양한 **API**연동 지원 – **JMS**, 메일, 스케줄링
  - 6) **IoC : Inversion of Control**(제어의 역전)
  - 7) **DI: Dependency Injection**지원
    - 설정파일이나 어노테이션을 통해 객체간의 의존 관계를 설정하므로 의존하고 있는 객체를 직접 생성하거나 검색할 필요가 없음
  - 8) **AOP(Aspect Oriented Programming)**지원
    - 모든 부가기능과 적용 대상의 조합을 통해 여러 오브젝트에 산재해 있는 공통적인 기능을 쉽게 개발하고 관리하는 기능

# 스프링 특징

1. 사전에 JDK SE 1.6 버전 이상 설치
2. 스프링 프로젝트 IDE
  - 1) Eclipse에 라이브러리 직접 추가 – 스프링 라이브러리 다운로드
  - 2) <https://maven.springframework.org/release/org/springframework/spring/>
  - 3) Eclipse Plugin 이용
  - 4) IntelliJ 사용
  - 5) 프레임워크 전용 Eclipse 이용: 전자정부프레임워크
  - 6) STS(Spring Tool Suite – Spring 전용 Eclipse) 사용  
<http://spring.io/tools>: 최신 버전  
<http://spring.io/tools/sts/all>: 모든 버전
3. 스프링 프로젝트 종류
  - 1) Spring Boot를 이용(Spring Starter Project): 최대한 간단하게 실행하고 배포가 가능한 수준의 애플리케이션을 만들 때 사용하는데 WAS 설정없이 실행이 가능하기 때문에 테스트 하기에 편리하지만 기존의 웹 프로젝트 설정과 다른 방법으로 사용하며 JSP 설정은 별도로 수행
  - 2) Spring Template Project 이용(Spring Project): WAS를 사용하거나 이전에 Spring Project를 만들어본 경우 사용하는 방식인데 WAS로 인한 리소스 소모가 심하다는 단점이 있지만 기존 프로젝트들이 이 방식으로 많이 만들어져 있음

# 티어와 레이어

## 1) 티어 : 물리적인 층

- 클라이언트층(**PC**, 스마트 폰)
- 중간층(애플리케이션 서버)
- **EIS**층 (**DB**, 레거시 시스템)

## 2) 레이어 : 논리적인 층

- 프레젠테이션층 : 사용자 인터페이스와 컨트롤러
- 비즈니스로직 층 : **service**층 업무처리 대상
- 데이터엑세스 층 : **Dao**

## Spring 주요기능

- **DI/AOP**컨테이너
- 스프링 **MVC**, 스프링 웹플로우
- 스프링**JDBC**
- 웹 인티그레이션 기능
- **ORM** 인티그레이션 기능

# 개발환경

## 1. STS란?

- 이클립스에서 스프링을 편하게 쓰기 위해 스프링 **IDE** 프로젝트를 만들었고 로드 존슨이 설립한 스프링 지원회사인 스프링소스(**SpringSource**)에서 스프링소스툴 스위트(**STS, SpringSource Tools Suite**)를 만들었다. 한때 유료제품이었지만 지금은 무료.

- **Download** : google에서 sts download조회

<https://spring.io/tools/>

<http://docs.spring.io/spring/docs/current/javadoc-api/>

spring®

Why Spring ▾

Learn ▾

Projects ▾

Training

Support

Community ▾



## Spring Tools | 4

Spring Tools 4 is the next generation of Spring tooling for your favorite coding environment. Largely rebuilt from scratch, it provides world-class support for developing Spring-based enterprise applications, whether you prefer Eclipse, Visual Studio Code, or Thelia IDE.

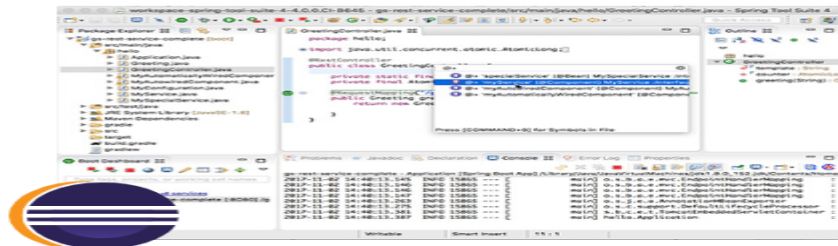
## Spring Tools 4 for Eclipse

The all-new Spring Tool Suite 4.  
Free. Open source.

LINUX 64-BIT

MACOS 64-BIT

WINDOWS 64-BIT





# 개발환경

**Sts 3.대 버전 다운로드**

**<https://github.com/spring-projects/toolsuite-distribution/wiki/Spring-Tool-Suite-3>**

**full distribution on Eclipse 4.16**

**[https://download.springsource.com/release/STS/3.9.15.RELEASE/dist/e4.16/spring-tool-suite-3.9.15.RELEASE-e4.16.0-win32-x86\\_64.zip](https://download.springsource.com/release/STS/3.9.15.RELEASE/dist/e4.16/spring-tool-suite-3.9.15.RELEASE-e4.16.0-win32-x86_64.zip)**

**[https://download.springsource.com/release/STS/3.9.15.RELEASE/dist/e4.16/spring-tool-suite-3.9.15.RELEASE-e4.16.0-macosx-cocoa-x86\\_64.dmg](https://download.springsource.com/release/STS/3.9.15.RELEASE/dist/e4.16/spring-tool-suite-3.9.15.RELEASE-e4.16.0-macosx-cocoa-x86_64.dmg)**

**[https://download.springsource.com/release/STS/3.9.15.RELEASE/dist/e4.16/spring-tool-suite-3.9.15.RELEASE-e4.16.0-linux-gtk-x86\\_64.tar.gz](https://download.springsource.com/release/STS/3.9.15.RELEASE/dist/e4.16/spring-tool-suite-3.9.15.RELEASE-e4.16.0-linux-gtk-x86_64.tar.gz)**

**p2 repositories**



## STS 4.대 이상

**spring-tool-suite-4-4.5.0.RELEASE-e4.14.0-win32.win32.x86\_64.self-extracting.jar**

압축을 풀어보면 다음과 같습니다.

| 이름                  | 수정한 날짜         | 유형         | 크기        |
|---------------------|----------------|------------|-----------|
| META-INF            | 2020-01-13 ... | 파일 폴더      |           |
| ui                  | 2020-01-13 ... | 파일 폴더      |           |
| contents.zip        | 2019-12-17 ... | 압축(ZIP) 파일 | 389,396KB |
| SelfExtractor.class | 2019-10-17 ... | CLASS 파일   | 8KB       |

contents.zip 을 한번 더 압축해제하면 원하는 결과물이 나옵니다.

| 이름                       | 수정한 날짜           | 유형                | 크기    |
|--------------------------|------------------|-------------------|-------|
| configuration            | 2020-01-13 오전... | 파일 폴더             |       |
| dropins                  | 2019-12-17 오전... | 파일 폴더             |       |
| features                 | 2020-01-13 오전... | 파일 폴더             |       |
| META-INF                 | 2020-01-13 오전... | 파일 폴더             |       |
| p2                       | 2020-01-13 오전... | 파일 폴더             |       |
| plugins                  | 2020-01-13 오전... | 파일 폴더             |       |
| readme                   | 2020-01-13 오전... | 파일 폴더             |       |
| .eclipseproduct          | 2019-12-10 오전... | ECLIPSEPRODUCT... | 1KB   |
| artifacts.xml            | 2019-12-17 오전... | XML 문서            | 176KB |
| eclipsesec.exe           | 2019-12-17 오전... | 응용 프로그램           | 127KB |
| license.txt              | 2019-12-17 오전... | 텍스트 문서            | 12KB  |
| open-source-licenses.txt | 2019-12-17 오전... | 텍스트 문서            | 932KB |
| SpringToolSuite4.exe     | 2019-12-17 오전... | 응용 프로그램           | 416KB |
| SpringToolSuite4.ini     | 2019-12-17 오전... | 구성 설정             | 1KB   |

# STS 3 설치

## Help – Eclipse marketplace sts조회 Spting Tools 3 설치

### Eclipse Marketplace

Select solutions to install. Press Install Now to proceed with installation.  
Press the "more info" link to learn more about a solution.



Search

Recent


Popular

Favorites

Installed


Giving IoT an Edge


Find:



**Spring Tools 3 Add-On for Spring Tools 4 3.9.12.CI**  
Spring Tools 3 Add-On for Spring Tools 4 Attention: This add-on pack provides additional components from the previous Spring Tools 3 generation to be installed... [more info](#)  
by [Pivotal](#), EPL  
[J2EE](#) [spring](#) [Spring IDE](#) [Cloud](#) [jee](#) ...  


★ 213


 Installs: 284K (8,085 last month)



**Spring Tools 3 (Standalone Edition) 3.9.11.RELEASE**  
Spring Tools 3 (Standalone Edition) The Spring Tools 3 contain the previous generation Spring tooling for Eclipse, mostly focused on working with Spring apps... [more info](#)  
by [Pivotal](#), EPL  
[J2EE](#) [spring](#) [Spring IDE](#) [Cloud](#) [jee](#) ...  

★ 28

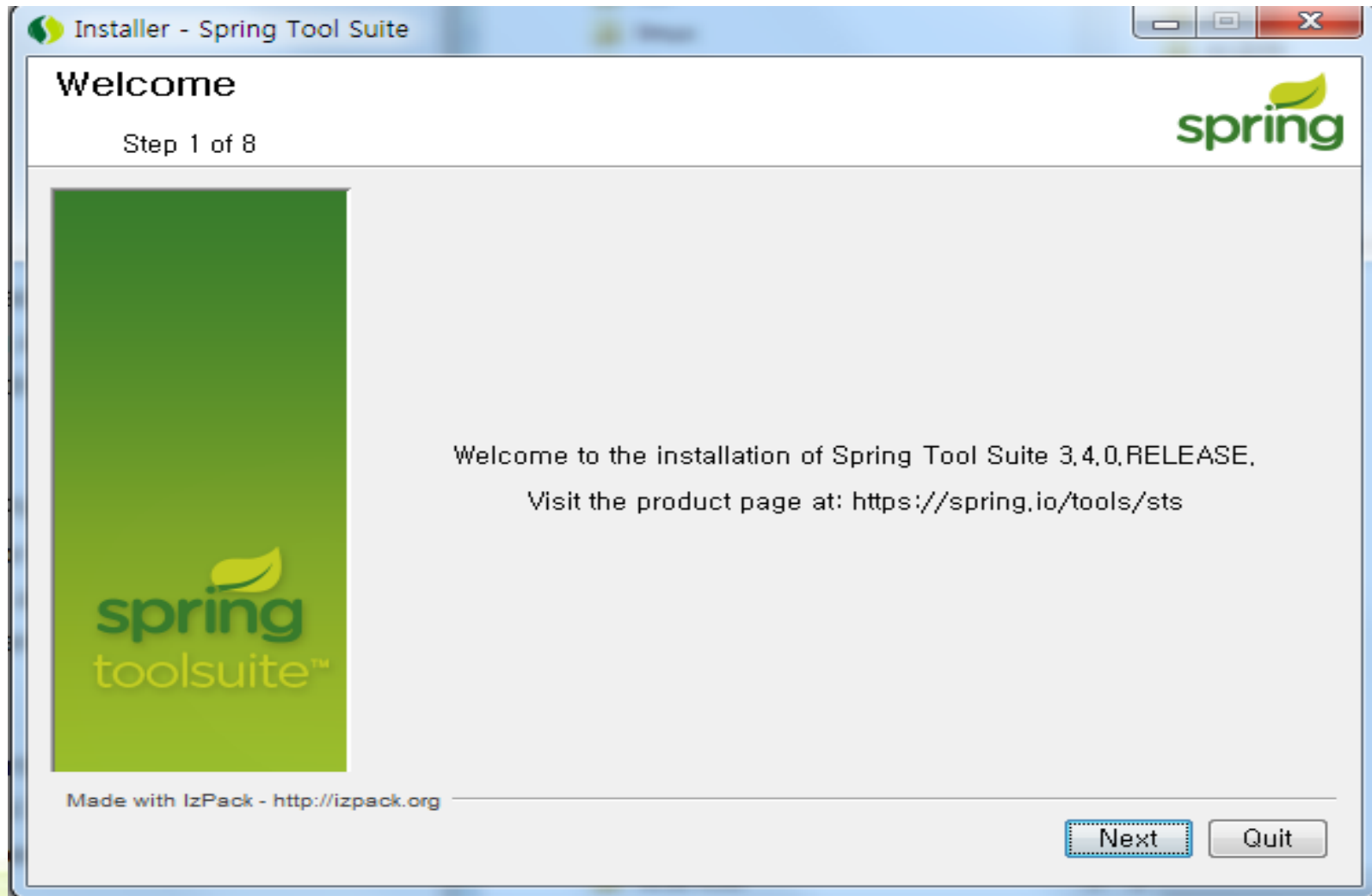
 Installs: 39.8K (7,653 last month)



**Spring Tools 4 (aka Spring Tool Suite 4) 4.5.1.RELEASE**  
Spring Tools 4 is the next generation of Spring Boot tooling for your favorite coding environment. Largely rebuilt from scratch, it provides world-class support... [more info](#)  
by [Pivotal](#), EPL  
[spring](#) [Spring IDE](#) [Cloud](#) [Spring Tool Suite](#) [STS](#) ...

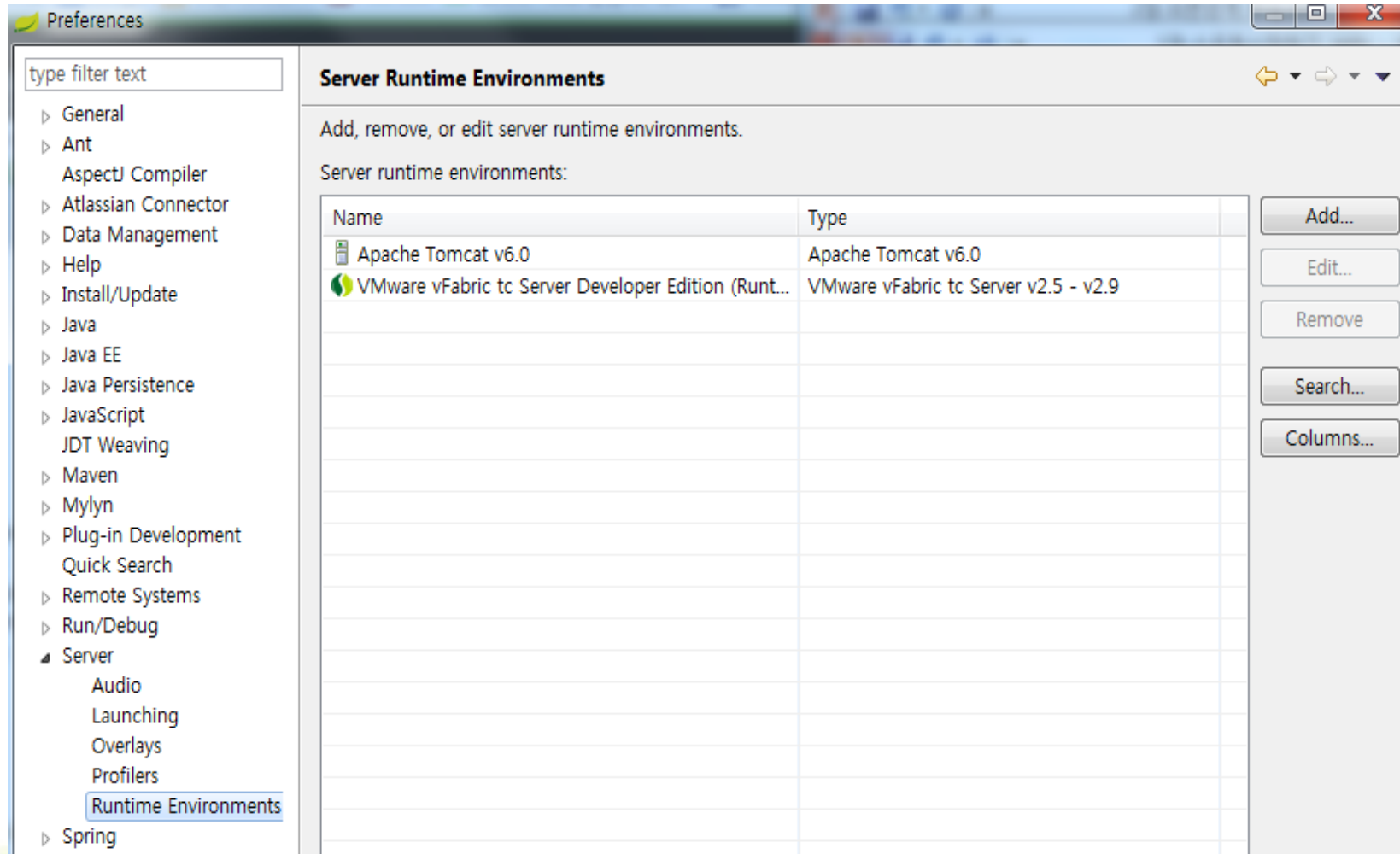
# 개발환경

스프링 프레임워크(**Spring Framework**) 플러그인 및 실제 라이브러리를 설치, 설정



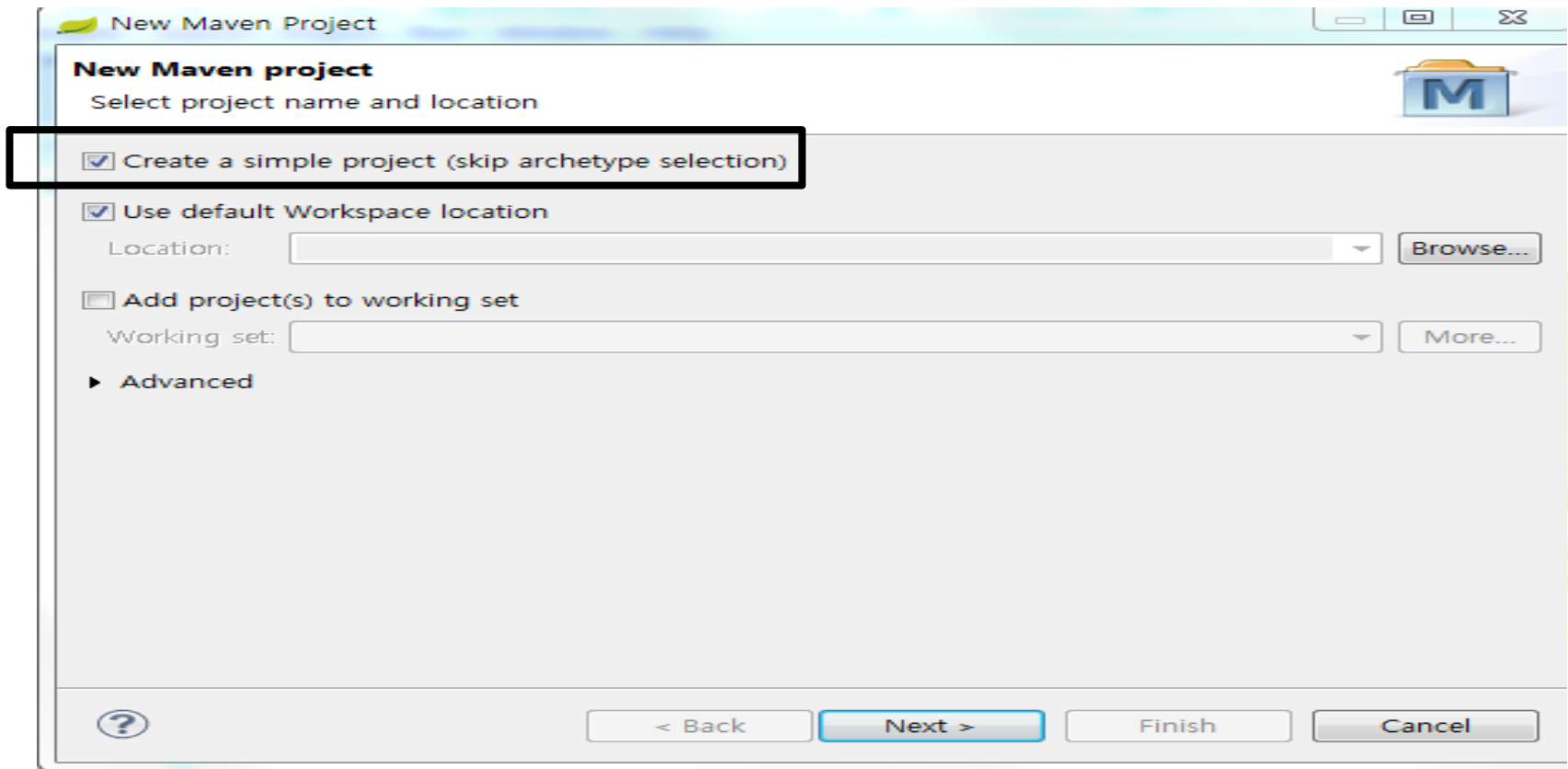
## Window- Preferences

Server – Runtime Environments를 선택



# 프로젝트 만들기

**File – new – maven Project**  
**create a simple project**에 체크하고 **next**



# ID만들기

Group ID, Artifact ID를 입력하고  
Packaging에 web일 때는 war 그밖에는 jar선택

New Maven Project

**New Maven project**  
Configure project

Artifact

Group Id: ch03

Artifact Id: ch03

Version: 0.0.1-SNAPSHOT

Packaging: jar

Name:

Description:

Parent Project

Group Id:

Artifact Id:

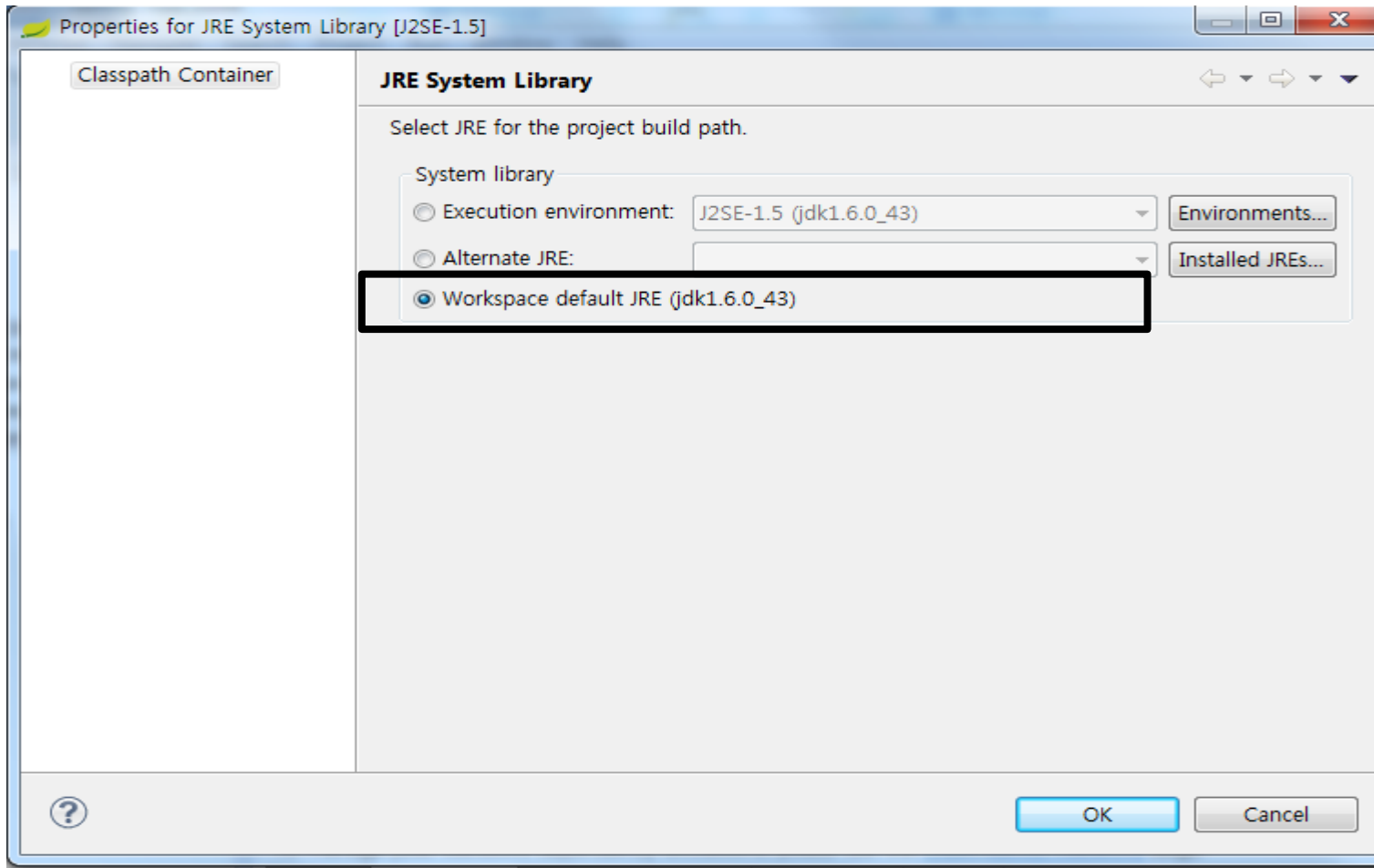
Version:

Browse... Clear

Advanced

? < Back Next > Finish Cancel

**JRE System Library**를 선택하고 우측 버튼을 클릭하여 **Properties**를 선택한 후에 **workspace default** 선택





# 의존 라이브러리 설정

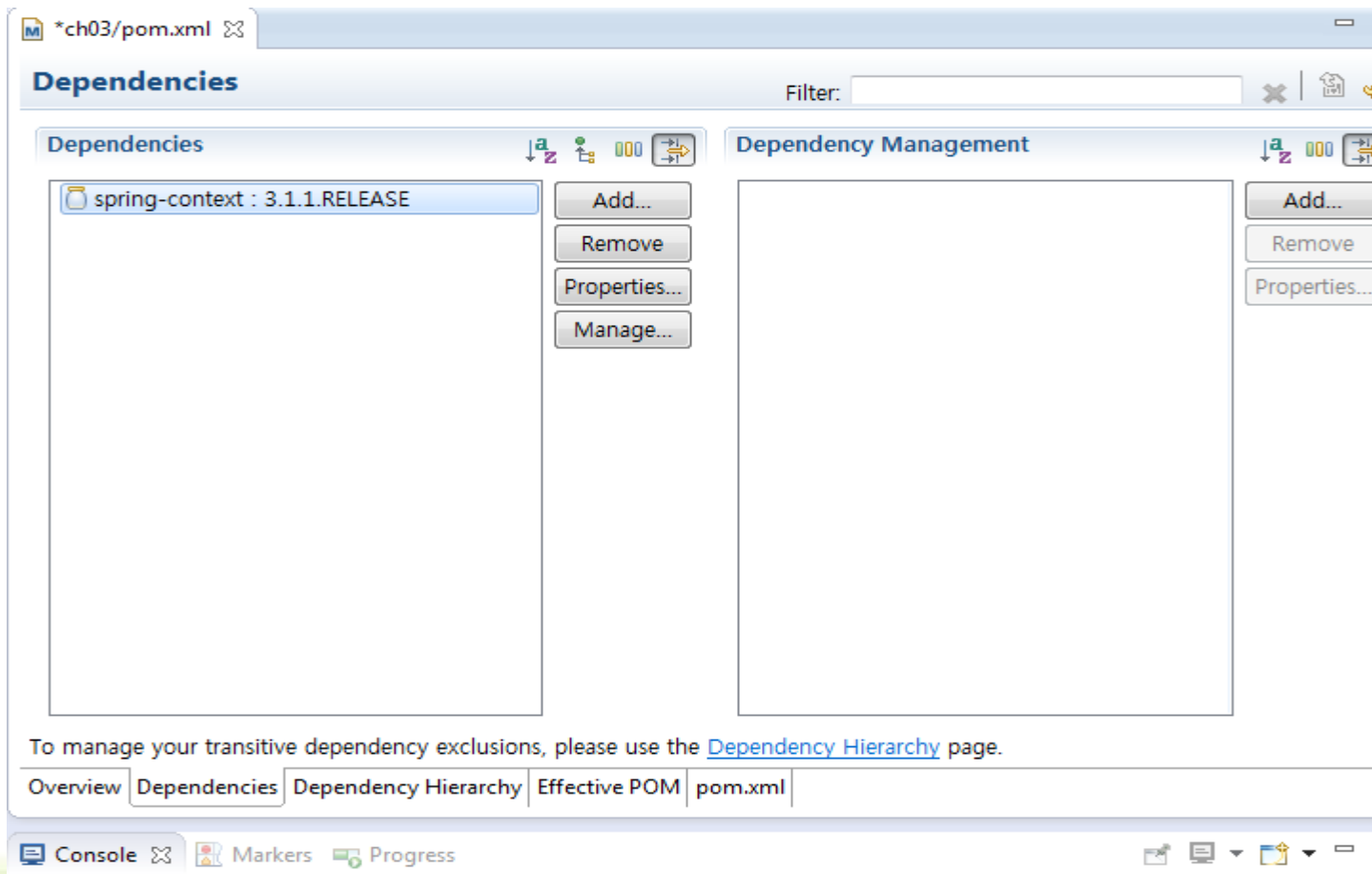
Poem.xml에서 **dependencies** 탭을 열고 **add**를 선택한 후

**Group ID : org.springframework**

**Artifact ID : spring-context**

**Version : 4.1.1.RELEASE**

**Scope : compile**



# 스프링의 주요 모듈 목록

| 모듈 명            | 설 명  |
|-----------------|--|
| core            | 미기능을 비롯한 프레임워크의 기반을 제공   |
| beans           | BeanFactory 인터페이스를 통해 구현된다   |
| expression      | 객체에 접근하고 객체를 조작하기 위한 표현언어를 제공한다. JSP에 규정된 통합EL을 확장한다   |
| context         | spring-code와 spring-beans모듈을 확장해서 국제화, 이벤트처리, 리소스 로딩, 서블릿컨테이너를 위한 컨텍스트 생성 등의 기능을 추가 제공<br>AplicationContext 인터페이스를 통해 구현 |
| context.support | Ehcache, 메일, 스케줄링, UI의 Velocity 지원 기능을 제공한다  |
| aop             | AOP Alliance에 호환되는 AOP 구현을 제공한다  |
| aspects         | AspectJ와의 통합을 제공한다   |
| web             | 파일 업로드, Locale 처리 등 웹을 위한 통합, 원격지원 중 웹 관련 기능제공   |
| web.servlet     | 스프링 MVC를 제공, JSP, Velocity에 대한 뷰 연동을 지원한다  |
| web.struts      | 스프링과 스트러츠 연동 기능을 제공한다  |
| web.portlet     | 포틀릿 환경에서 사용되는 MVC구현을 제공한다.   |

# 스프링의 주요 모듈 목록

| 모듈 명              | 설 명  |
|-------------------|--|
| Transaction       | AOP을 이용한 선언적 트랜잭션 관리 및 코드를 이용한 트랜잭션 관리 기능을 제공한다  |
| Jdbc              | JDBC프로그래밍을 위한 추상 레이어를 제공한다. JDBC 템플릿을 제공함으로써 간결한 코드를 JDBC프로그래밍을 할 수 있도록 돕는다                |
| orm               | 하이버네이트, JPA, IBATIS, MYBATIS, JDO 등 ORM API를 위한 통합 레이어를 제공한다. 스프링이 제공하는 트랜잭션 관리와의 연동을 지원한다 |
| oxm               | 객체와의 XML 사이의 매핑을 처리하기 위한 추상 레이어를 제공한다. JAXB, Castor, XMLBeans, JIBX, Xstream과의 연동을 지원한다    |
| jms               | JMS의 메시지를 생성하고 수신하는 기능을 제공한다   |
| test              | Junit이나 TestNG를 이용한 스프링 컴퍼넌트의 테스트를 지원한다  |
| instrument        | Instrumentation 지원 클래스를 제공한다   |
| instrument.tomcat | 톰캣서버를 위한 instrumentation지원클래스를 제공한다  |
| asm               | ASM 라이브러리를 재패키징한 모듈  |

# 스프링 주요 모듈의존 관계

spring-messaging

spring-websocket

spring-webmvc

spring-web

spring-orm

spring-jdbc

spring-context-support

spring-tx

spring-jms

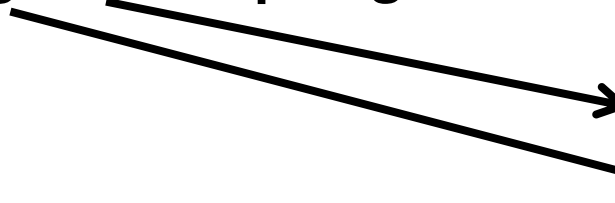
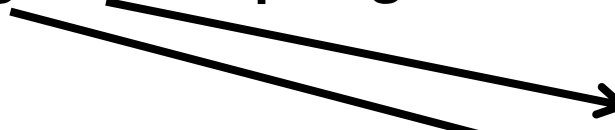
spring-context

spring-aop

spring-oxm

spring-beans

spring-core



# 새 프로젝트 작성

- File -> New -> Project -> **Maven Project** 선택하고 **Next**
- **Group ID : ch01**
- **Artifact ID : ch01-01**
- **Package ;** 웹 어플리케이션은 **war** 아니면 **jar** 선택

## 의존라이브러리 설정

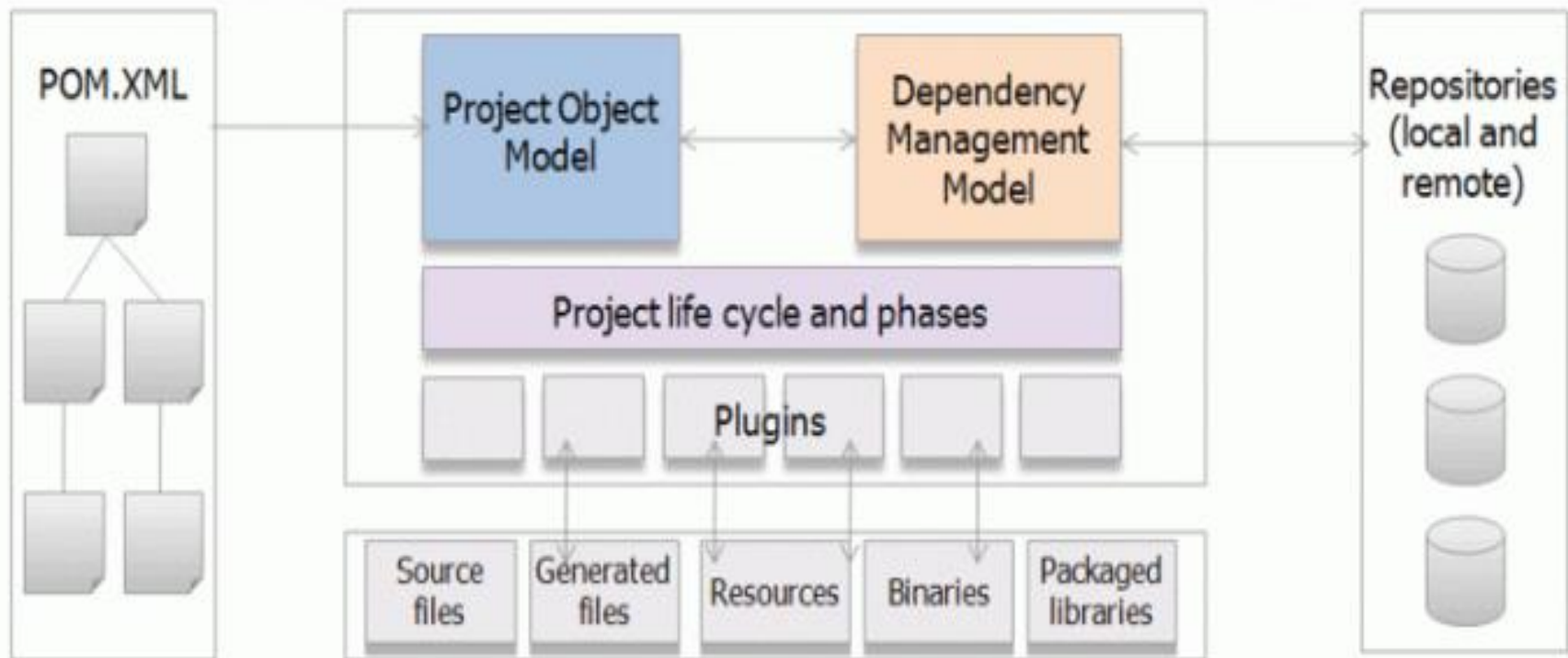
- **pom.xml**을 열고 **dependencies** 탭을 연다
- **Add**를 선택하고 설정값 입력
- **Group ID : org.springframework**
- **Artifact ID : spring-context**
- **Version : 4.1.1.RELEASE**
- **Scope : compile**

## 프로젝트 폴더구성

- **src/main/java** : 어플리케이션 자바소스
- **src/main/resource** : 애플리케이션의 설정파일. 메시지 리소스 파일클래스 경로상에 배치하는 것
- **src/test/java** : 테스트 프로그램의 자바소스 파일
- **src/test/resource** : 테스트프로그램의 설정파일. 메시지 리소스 파일클래스 경로상에 배치하는 것

# Maven이란 ?

- pom.xml 파일을 이용해서 관련된 jar 파일(라이브러리 파일, 모듈)을 다운하고 관리
- 프로젝트를 관리하기 위한 툴(라이브러리뿐만 컴파일, 테스트, 패키지 등 다양한 기능을 제공)
- 기본적으로 사용하는 이클립스에 Maven project가 포함되어 있다.



(출처 : 표준프레임워크 포털 - Maven 아키텍처 )

# Spring IoC 정의

마틴 파울러는 **2004**년의 글에서 제어의 어떤 측면이 역행되는 것인지에 대한 의문을 제기했다. 그는 의존하는 객체를 역행적으로 취득하는 것이라는 결론을 내렸다. 그는 그와 같은 정의에 기초하여 제어 역행이라는 용어에 좀더 참신한 ‘의존성 주입(**dependency injection**)’이라는 이름을 지어줬다.

모든 어플리케이션은 비즈니스 로직을 수행하기 위해 서로 협업하는 둘 또는 그 이상의 클래스들로 이뤄진다. 전통적으로 각 객체는 협업할 객체의 참조를 취득해야 하는 책임이 있다. 이것이 의존성이다. 이는 결합도가 높으며 테스트하기 어려운 코드를 만들어 낸다.

**IoC**를 적용함으로써 객체들은 시스템 내의 각 객체를 조정하는 어떤 외부의 존재에 의해 생성 시점에서 의존성을 부여 받는다. 즉 의존성이 객체로 주입(**inject**)된다는 말이다. 따라서 **IoC**는 한 객체가 협업해야 하는 다른 객체의 참조를 취득하는 방법에 대한 책임의 역행이라는 의미를 갖는다.

## 설계원칙

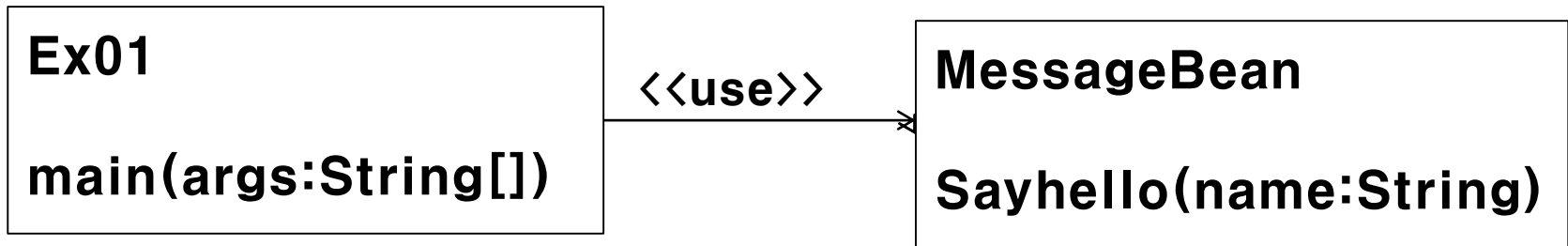
자주 변경되는 구상클래스(**Concrete class**)에 의존하지 마라  
어떤 클래스를 상속받아야 한다면, 기반 클래스를 추상 클래스로 만들어라  
인터페이스를 만들어서 이 인터페이스에 의존하라

→ **DIP(Dependency Inversion Principle)**

좋은 제품 : 강한 응고도 약한 결합도



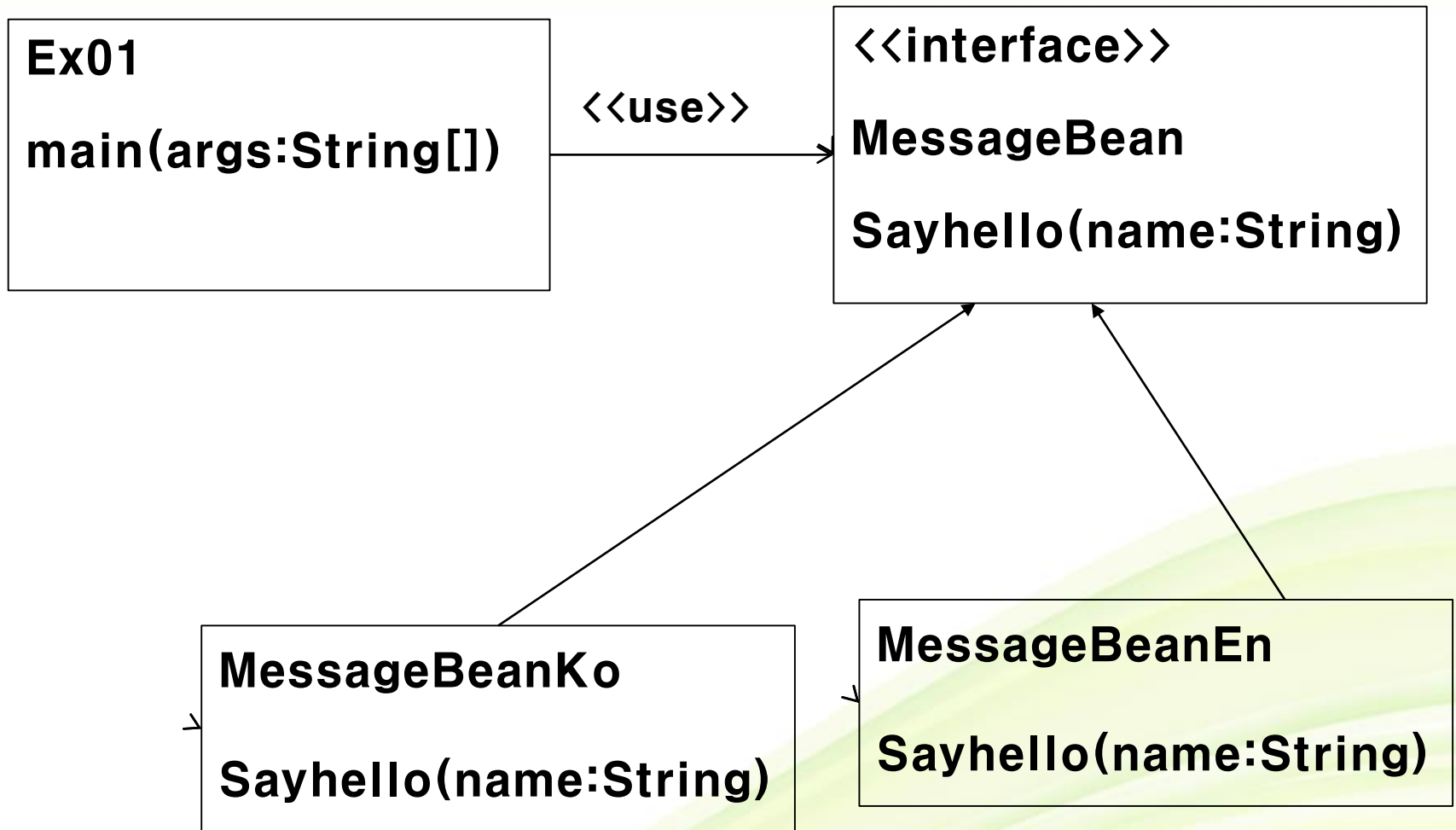
# 스프링의 기본 기능과 구조



```
package sample01;
public class MessageBean {
    public void sayHello(String name) {
        System.out.println("Hello, " + name + "!");
    }
}
```

```
package sample01;
public class Ex01 {
    public static void main(String[] args) {
        MessageBean bean = new MessageBean();
        bean.sayHello("Spring");
    }
}
```

# 스프링의 기본 기능과 구조



클래스간의 의존성을 약하게 하기 위해서 인터페이스 사용  
결합도, 응집도

# 스프링의 기본 기능과 구조

```
package samp2;  
public interface MessageBean {  
    public void sayHello(String name);  
}
```

```
package samp2;  
Public class MessageBeanEn implements MessageBean {  
    public void sayHello(String name) {  
        System.out.println(name+" !! Hello");  
    }  
}
```

# 스프링의 기본 기능과 구조

```
package samp2;
public class MessageBeanKr implements MessageBean {
    public void sayHello(String name) {
        System.out.println(name+" !! 안녕하세요");
    }
}
```

```
package samp2;
public class Ex01 {
    public static void main(String[] args) {
        // MessageBean mb = new MessageBeanEn();
        MessageBean mb = new MessageBeanKr();
        mb.sayHello("spring");
    }
}
```

# 스프링의 기본 기능과 구조

```
MessageBean bean = new MessageBeanEn( );  
Bean.sayHello( "Spring" );
```

수정하지 않아도 되는 코드

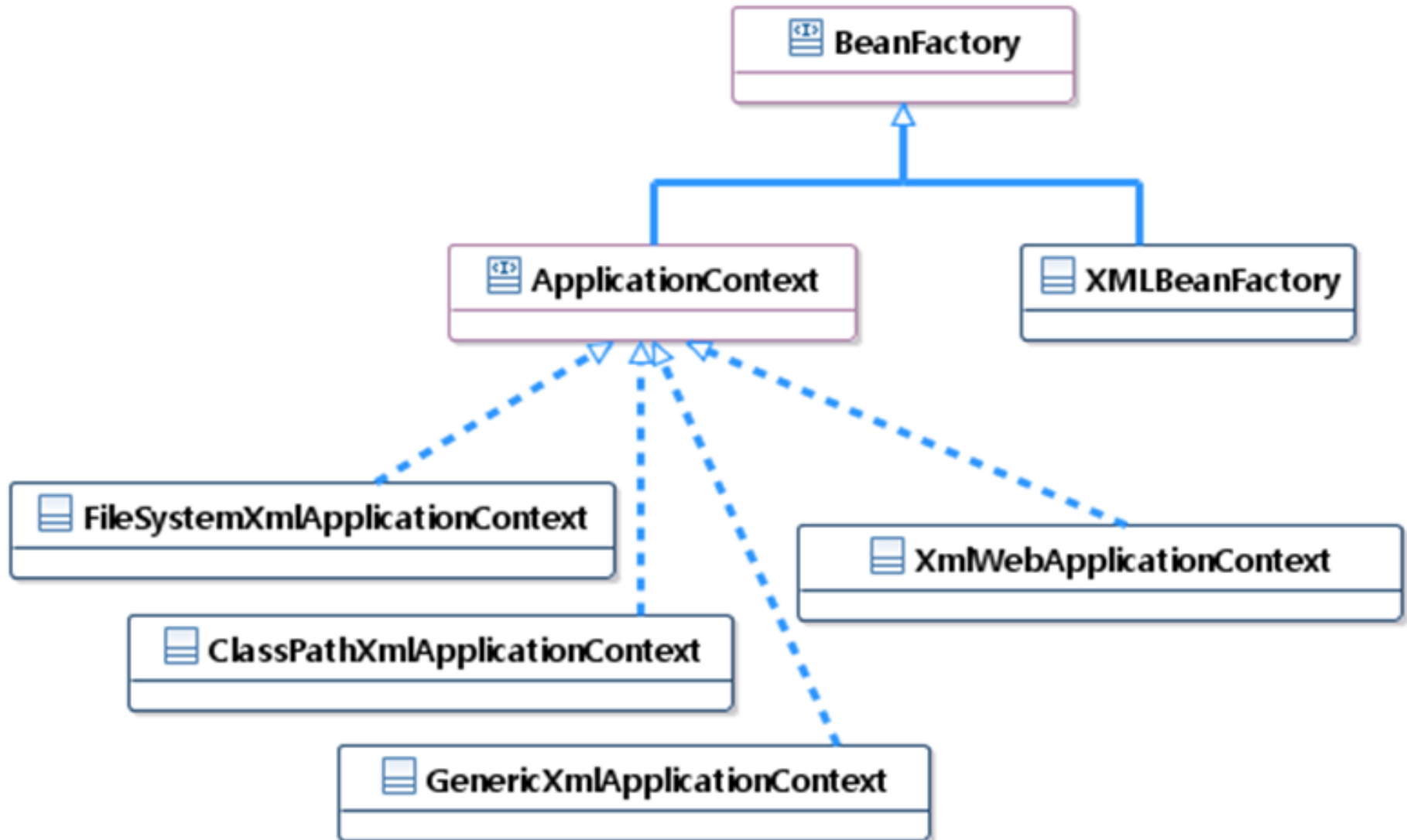
```
MessageBean bean = new MessageBeanKo( );  
Bean.sayHello( "Spring" );
```

아직은 수정해야 하는 코드가 존재

수정하지 않아도 되는 코드

# XML을 이용한 DI - 스프링 컨텍스트 상속도

스프링의 컨텍스트 클래스들은 객체를 생성하고 관리하는 컨테이너의 기능을 함



# BeanFactory 인터페이스

1. **bean** 객체를 관리하고 각 **bean** 객체간의 의존 관계를 설정해주는 기능을 제공하는 가장 단순한 인터페이스로 이 인터페이스를 **implements** 한 클래스로는 **XmlBeanFactory** 클래스가 존재
2. 생성자에 **Resource** 인터페이스를 구현한 클래스의 객체를 대입해서 객체 생성
3. **Resource** 인터페이스를 구현한 클래스
  - 1) **FileSystemResource**: 파일 시스템의 파일로부터 생성
  - 2) **InputStreamResource**: **InputStream** 이용
  - 3) **ClassPathResource**: 클래스 패스에서 읽어옵니다.
  - 4) **UrlResource**: **URL**에서 읽어옵니다.
  - 5) **ServletContextResource**: 웹 애플리케이션의 루트 디렉토리에서 읽어옵니다.
4. **Resource** 인터페이스를 **implements** 한 클래스의 객체를 이용해서 객체 생성 후 **getBean()**을 호출해서 설정된 **bean**의 **id**를 넘겨주거나 여기에 추가로 리턴되는 클래스 이름을 넘겨서 **bean** 객체를 생성합니다.
5. **BeanFactory** 인터페이스만 구현한 클래스는 단순히 컨테이너에서 객체를 생성하고 **DI**를 처리해주는 기능만을 제공



# ApplicationContext

1. 스프링에서 다양한 부가 기능(웹 개발, 메시지 처리..) 을 사용하기 위해서는 **BeanFactory** 인터페이스 대신에 여러 기능이 추가된 **ApplicationContext** 인터페이스를 이용합니다.

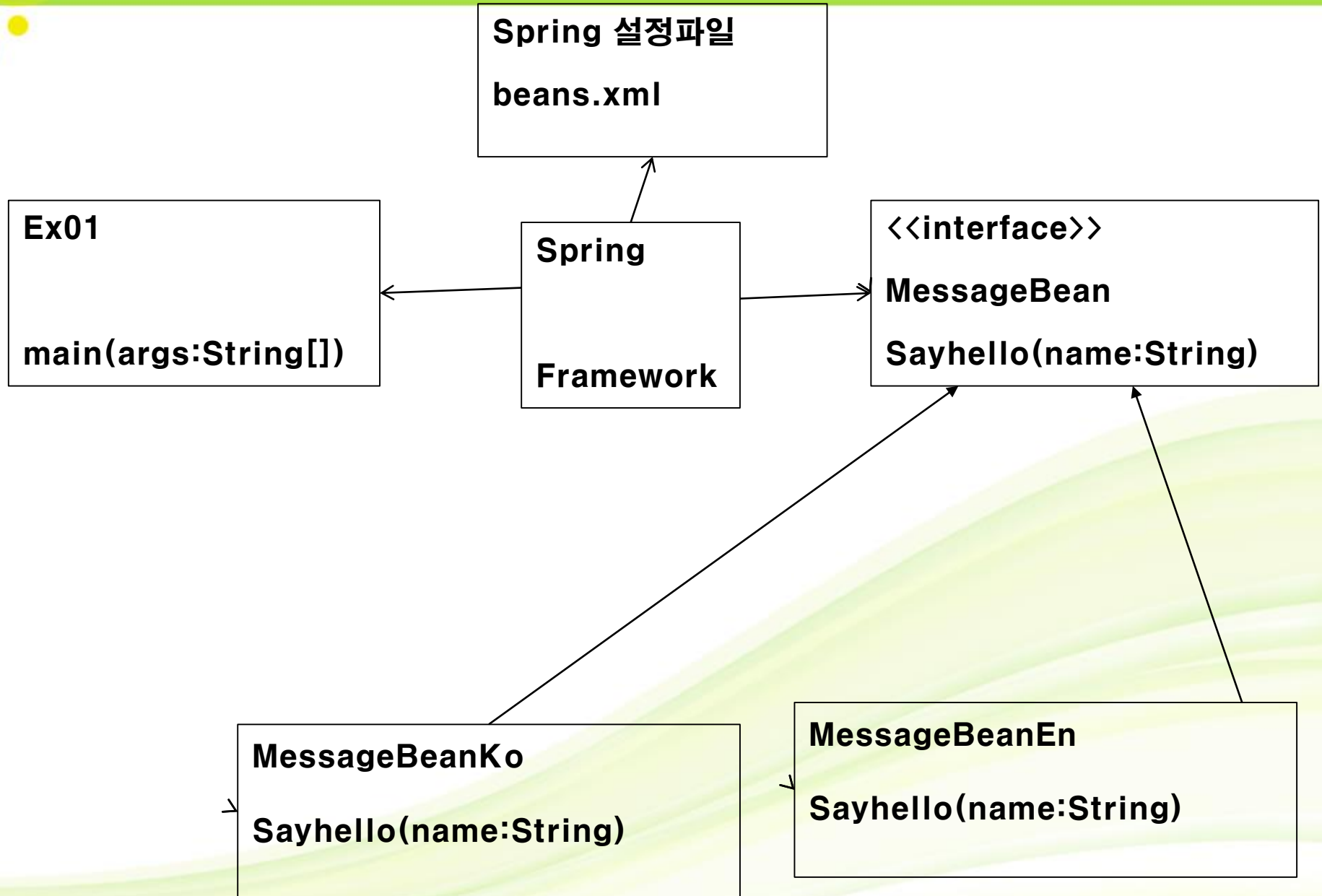
## 2. ApplicationContext

- 1) **BeanFactory** 인터페이스를 상속한 인터페이스로 빈 객체 라이프 사이클, 파일과 같은 자원 처리 추상화, 메시지 지원 및 국제화 지원, 이벤트 지원, **xml** 스키마 확장 등의 추가적인 기능을 제공하는 인터페이스
- 2) **AnnotationConfigApplicationContext**, **GenericXmlApplicationContext** 클래스가 **ApplicationContext** 인터페이스를 implements

## 3. WebApplicationContext

- 1) **Web** 프로젝트에서 사용가능 한 **ApplicationContext** 인터페이스 **ClassPathXmlApplicationContext**, **FileSystemXmlApplicationContext**, **XmlWebApplicationContext**, **AnnotationConfigWebApplicationContext**, 클래스가 implements
- 2) 클래스를 이용하는 경우는 거의 없고 웹 애플리케이션에서 **web.xml** 파일의 설정을 통해 **XmlWebApplicationContext** 객체를 생성해서 사용하는 경우가 많음

# 스프링의 기본 기능과 구조



# 스프링의 기본 기능과 구조

```
package sample03;
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
import org.springframework.core.io.FileSystemResource;
public class Ex01 {
    public static void main(String[] args) {
        /* BeanFactory bf =
        new XmlBeanFactory(new FileSystemResource("bean01.xml")); */
        AbstractApplicationContext ac
        // ApplicationContext ac =
            new FileSystemXmlApplicationContext("bean01.xml");
        // MessageBean mb = bf.getBean("message",MessageBean.class);
        // MessageBean mb = (MessageBean)ac.getBean("message");
        MessageBean mb = (MessageBean)bf.getBean("k");
        mb.sayHello("홍길동");
    }
}
```

**XmlBeanFactory** : 빈을 생성하고 설정 및 관리하는 역할

# 스프링의 기본 기능과 구조

## (3) 스프링을 이용한 샘플 어플리케이션 분석하기(sample3)

beans.xml

```
<?xml version="1.0" encoding="UTF-8"?>  
<beans xmlns="http://www.springframework.org/schema/beans"  
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
  xsi:schemaLocation="http://www.springframework.org/schema/beans  
    http://www.springframework.org/schema/beans/spring-beans.xsd">  
  <bean id="mb" class="samp3.MessageBeanKr"></bean>  
</beans>
```

# 설정 파일 beans.Xml

**id : Bean** 붙이는 식별자

**name : id**의 별칭, 복수 가능

**class : bean** 클래스의 완전한 클래스 이름

**parent : bean** 정의를 상속

**abstract : bean** 클래스의 추상클래스 여부 **false**

**singleton : bean**이 싱글톤으로 리턴하는 여부 **true**

**lazy-init : bean**의 로딩을 지연시킬 지 여부 **default**

**autowire** : 오토와이어 설정 **default**

**dependency-check** 위존 관계 확인 방법 **default**

**depends-on** : 이 **bean**이 의존할 **bean** 이름 먼저 초기화 보장

**init-methid** : **bean** 초기화 실행 시킬 메서드

**destroy-method** : **bean** 컨테이너 종료할 때 실행시킬 메서드

# 설정 파일 beans.Xml

```
<bean id= "messageBean" name= "a b c"  
class= "Sample3.MessageBeanEn" />  
<bean id= "messageBean" name= "a,b,c"  
class= "Sample3.MessageBeanEn" />  
<bean id= "messageBean" name= "a;b;c"  
class= "Sample3.MessageBeanEn" />  
<bean id= "messageBean" name= "a b,c"  
class= "Sample3.MessageBeanEn" />
```

```
MessageBean bean = factory.getBean("messageBean",  
    MessageBean.class);  
MessageBean bean = factory.getBean( "a",  
    MessageBean.class);  
MessageBean bean = factory.getBean( "b",  
    MessageBean.class);  
MessageBean bean = factory.getBean( "c",  
    MessageBean.class);
```

# 스프링의 특성

## - 스프링의 장점

필요한 인스턴스를 스프링에서 미리 생성해 준다.

클래스 사이의 결합(loosely coupled)을 느슨하게 할 수 있어 클래스 간의 의존 관계가 약해진다.

인스턴스를 스프링이 맡음 - 객체 변경할 때 스프링 파일 설정 변경으로 해결

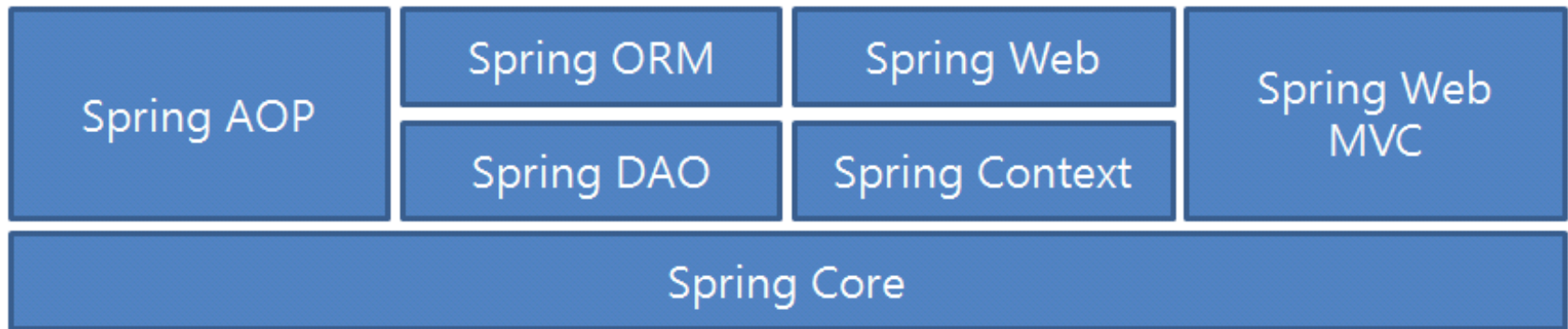
## - 스프링의 특징

- 스프링은 '어플리케이션 프레임워크'로 불리며, 웹 어플리케이션은 물론, 위에서 살펴 본 예제에서처럼 콘솔 어플리케이션이나 스윙과 같은 GUI 어플리케이션 등 어떤 어플리케이션에도 적용 가능한 프레임워크이다.
- 스프링은 EJB와 같이 복잡한 순서를 거치지 않아도 간단하게 이용할 수 있기 때문에 '경량(Lightweight) 컨테이너'라고도 부른다.
- 스프링은 Dependency Injection(DI)과 Aspect Oriented Programming(AOP)을 가장 중점적인 기술로 사용지만, 이외에도 여러 가지 기능을 제공하고 있다.



# 스프링의 특성

## - 스프링을 구성하는 모듈





# 의존 관계 주입

## [DI : Dependancy Injection]

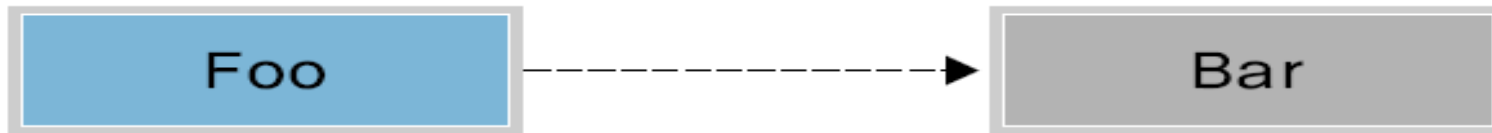
# DI에 대해서

1. DI는 「Dependency Injection」의 약어로서 인터페이스를 이용하여 컴포넌트화를 실현하는 것
2. 의존성 : 어떤 클래스가 자신의 임무를 하기 위해 필요한 값이나 사용할 다른 클래스와의 관계
3. 주입 : 어떤 클래스의 인스턴스에 대해 의존성을 설정 하는 것
4. 따라서 DI는 객체간의 의존 관계를 객체 자신이 아닌 외부 조립기가 수행해 준다는 개념, 스프링에서는 설정파일과 어노테이션을 이용하여 객체간의 의존관계 설정하는 기능 제공
5. 의존관계 주입컨테이너는 어떤 클래스가 필요로 하는 값이나 인스턴스를 생성, 취득하고 그 클래스의 인스턴스에 대하여 설정함
6. 즉 개발자가 필요로 하는 인스턴스를 생성, 취득하는 코드를 만들지 않아도 되므로 결합의존성이 낮아짐
  - Constructor Injection(생성자 주입)
  - Setter Injection(설정 메서드 통한 주입)

# DI에 대해서

## ◆ Foo.java

```
public class Foo{  
    private Bar bar;  
}
```



- **Foo**클래스가 **Bar**클래스에 의존하고 있고, **Bar**에 대한 인스턴스 참조는 의존 관계
- 생성자를 통한 주입이란 생성자를 사용해 의존관계 주입

# Constructor Injection(생성자를 이용하는 경우)

## ❖ 생성자를 이용하는 경우

<bean id= “ “ class = “ ” >

<constructor-arg value = “값” >

</bean>

✓<constructor-arg> 생성자의 매개변수로 **value**에 타입을 설정하지 않으면 **String** 타입으로 처리하며 **String**이 아니면 가장 가까운 데이터 타입을 찾아가게 되며 **type**을 직접 문자열 형태로 기재하는 것도 가능

✓**Value** 대신 **ref**를 이용해서 다른 **bean** 태그의 **id**나 **name**을 사용하는 것이 가능

✓**Ref bean= “bean 이름”** 또는 **ref local= “bean 이름”** 으로 기재해야 합니다.

✓**construct-arg**는 여러 개 사용이 가능

✓값이 **null**이면 **value** 대신에 <null />을 입력

# 전달인자가 두 개인 생성자

**<constructor-arg>**와 **<property>**요소

속성      설명

**index** 생성자의 몇 번째 인수에 값을 넘길 것인가 지정

**type** 생성자의 어떤 데이터 타입인 인수에 값을 넘길 것인지

**ref** 자식 요소 **<ref bean= “Bean이름” />**대신 사용

**value** 자식요소**<value>**값**</value>**대신 사용

---

```
public Foo(int a, String b)
{ }
```

---

```
<bean id= “foo” class= “Foo” >
<constructor-arg index= “0” >
<value>25</value>
</ constructor-arg>
<constructor-arg index= “1” value= “Hello” />
</bean>
```

---

```
<bean id= “foo” class= “Foo” >
<constructor-arg type= “int” value= “25” />
<constructor-arg type= “java.lang.String” value= “Hello” />
</bean>
```

# Setter Injection

## ❖ setter 메소드를 이용하는 경우

```
<bean id= “ “ class = “ ” >
```

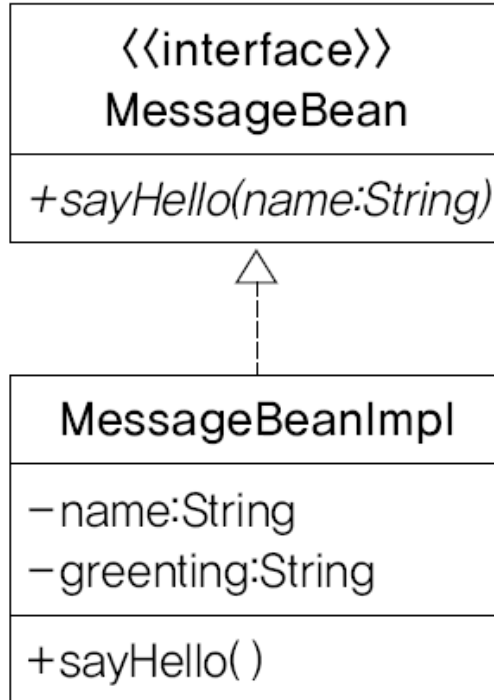
```
  <property name= “setter 메소드에서 set을 제외한 부분” >
```

```
    <value 또는 ref> 값 </value>
```

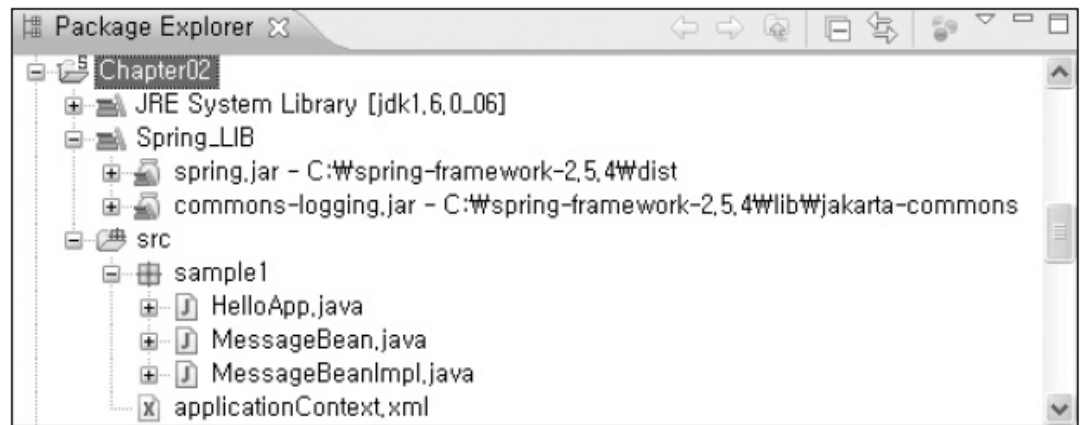
```
</bean>
```

**property name**은 변수 이름이 아니고 **setter** 메소드에서 **set**을 제외한 부분의 첫 글자만 소문자로 변경한 문자열입니다.

# 미 패턴 이해를 위한 예제



▲ 클래스 그림



▲ 파일 구성



# 미 패턴 이해를 위한 예제

```
package sample;
public class HelloApp {
    public static void main(String[] args) {
        BeanFactory factory = new XmlBeanFactory(new
FileSystemResource("applicationContext.xml"));
        MessageBean bean =
        (MessageBean)factory.getBean("messageBean");
        bean.sayHello();
    }
}
```

```
ApplicationContext ac =
    new FileSystemXmlApplicationContext("beans02.xml");
ApplicationContext ac =
new GenericXmlApplicationContext("classpath:beans02.xml");
```

```
package sample;
```

```
public interface MessageBean {
    void sayHello();
}
```

# 미 패턴 이해를 위한 예제

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

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/b
eans
http://www.springframework.org/schema/beans/spring-beans-
3.0.xsd">
```

```
  <bean id="messageBean" class="sample.MessageBeanImpl" >
```

```
    <constructor-arg>
```

```
      <value>Spring</value>
```

```
    </constructor-arg>
```

```
    <property name="greeting">
```

```
      <value>Hello, </value>
```

```
    </property>
```

```
  </bean>
```

```
</beans>
```

```
※. <import resource="classpath:beans02.xml"/>
```

# DI 패턴 이해를 위한 예제

```
package sample;
public class MessageBeanImpl implements MessageBean {
    private String name;
    private String greeting;
    @Override
    public void sayHello() {
        System.out.println(greeting + name + "!");
    }

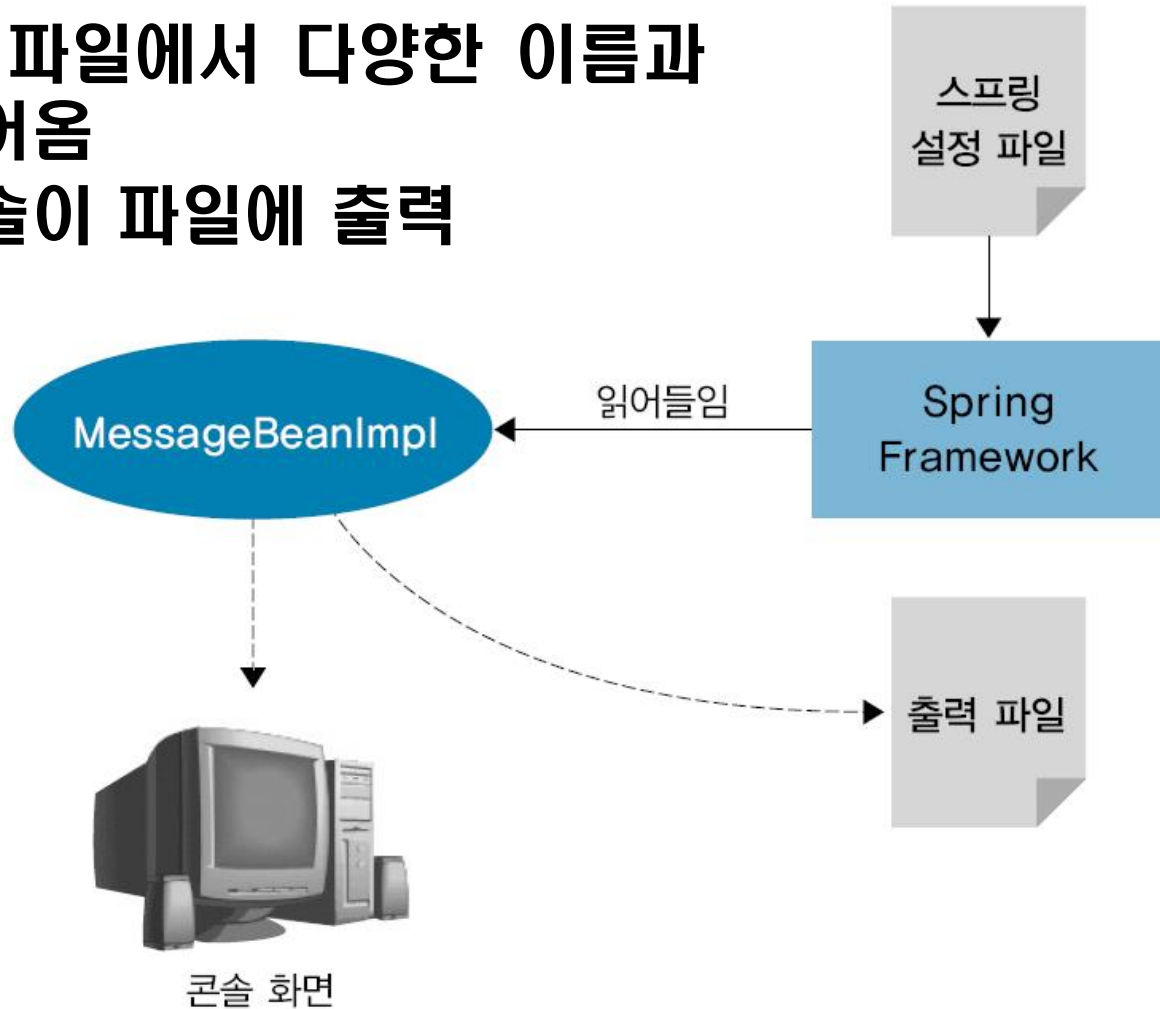
    public MessageBeanImpl(String name) {
        this.name = name;
    }

    public String getGreeting() {
        return greeting;
    }

    public void setGreeting(String greeting) {
        this.greeting = greeting;
    }
}
```

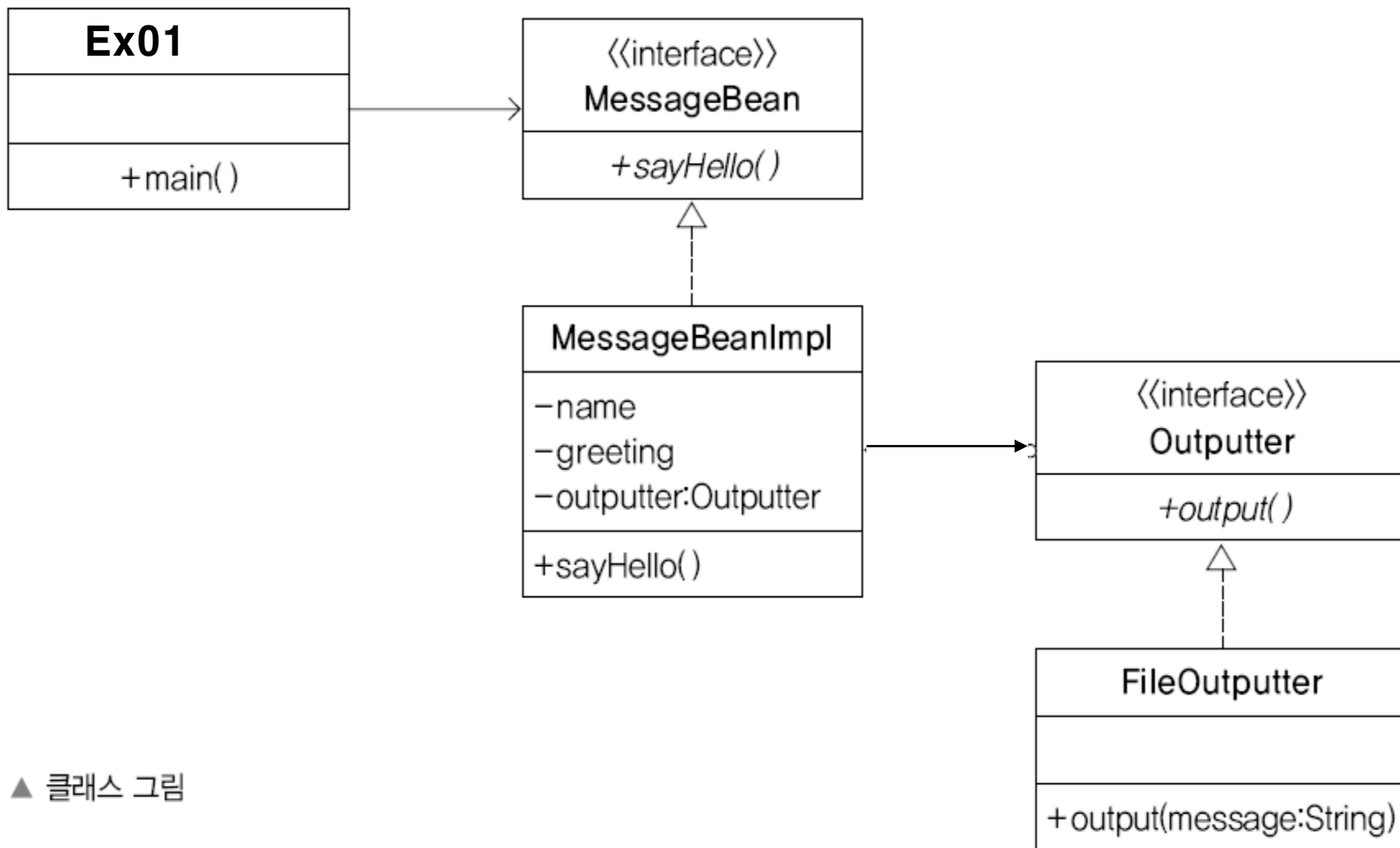
# 다양한 장치로 메시지를 출력하는 빈

스프링 설정 파일에서 다양한 이름과  
인사말을 얻어옴  
메시지를 콘솔이 파일에 출력



▲ 예제 개요 그림

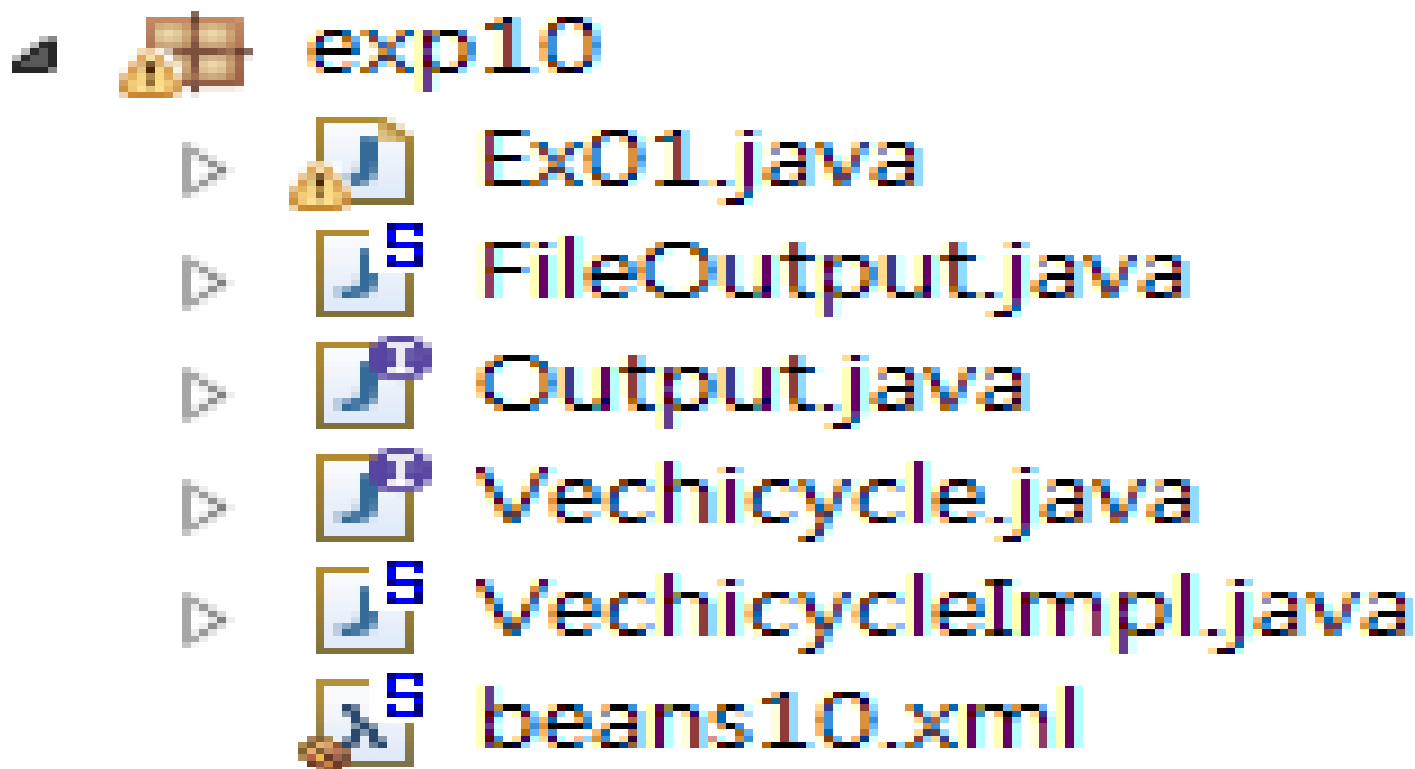
# 다양한 장치로 메시지를 출력하는 빈



▲ 클래스 그림

# 다양한 장치로 메시지를 출력하는 빈

## ▼ 파일 구성



# 다양한 장치로 메시지를 출력하는 빈

```
package sample1;  
import org.springframework.beans.factory.BeanFactory;  
import org.springframework.beans.factory.xml.XmlBeanFactory;  
import org.springframework.core.io.FileSystemResource;
```

```
public class HelloApp {  
    public static void main(String[] args) {  
        BeanFactory factory = new XmlBeanFactory(new  
FileSystemResource("beans.xml"));  
        MessageBean bean =  
(MessageBean)factory.getBean("messageBean");  
        bean.sayHello();  
    }  
}
```

```
package sample1;
```

```
public interface MessageBean {  
    void sayHello();  
}
```

# 다양한 장치로 메시지를 출력하는 빈

```
<?xml version="1.0" encoding="UTF-8" ?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
    <bean id="messageBean" class="sample1.MessageBeanImpl" >
        <constructor-arg>
            <value>Spring</value>
        </constructor-arg>

        <property name="greeting">
            <value>Hello, </value>
        </property>

        <property name="outputter">
            <ref local="outputter" />
        </property>
    </bean>
    <bean id="outputter" class="sample1.FileOutputter">
        <property name="filePath">
            <value>out.txt</value>
        </property>
    </bean>
</beans>
```



# 다양한 장치로 메시지를 출력하는 빈

```
package sample1;
import java.io.IOException;
public class MessageBeanImpl implements MessageBean {
    private String name;
    private String greeting;
    private Outputter outputter;
    public MessageBeanImpl(String name) {
        this.name = name;
    }
    public void setGreeting(String greeting) {
        this.greeting = greeting;
    }
    public void sayHello() {
        String message = greeting + name + "!";
        System.out.println(message);
        try {
            outputter.output(message);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    public void setOutputter(Outputter outputter) {
        this.outputter = outputter;
    }
}
```

# 다양한 장치로 메시지를 출력하는 빈

```
package sample1;  
import java.io.IOException;  
public interface Outputter {  
    public void output(String message) throws IOException;  
}
```

```
package sample1;
```

```
import java.io.*;  
public class FileOutputter implements Outputter {  
    private String filePath;  
    public void output(String message) throws IOException {  
        FileWriter out = new FileWriter(filePath);  
        out.write(message);  
        out.close();  
    }  
    public void setFilePath(String filePath) {  
        this.filePath = filePath;  
    }  
}
```

# 다양한 장치로 메시지를 출력하는 빈

```
package sample1;
```

```
import org.springframework.beans.factory.BeanFactory;  
import org.springframework.beans.factory.xml.XmlBeanFactory;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.FileSystemXmlApplicationContext;  
import org.springframework.core.io.FileSystemResource;
```

```
public class Ex01 {
```

```
    public static void main(String[] args) {
```

```
        // BeanFactory factory = new XmlBeanFactory(new  
            FileSystemResource("beans2.xml"));  
        ApplicationContext factory = new  
            FileSystemXmlApplicationContext("beans2.xml");  
        MessageBean bean = (MessageBean)factory.getBean("messageBean");  
        bean.sayHello();
```

```
    }
```

```
}
```

# 다양한 장치로 메시지를 출력하는 빈

```
public interface MessageBean {  
    void sayHello();  
}
```

```
import java.io.IOException;  
import org.springframework.beans.factory.annotation.Autowired;  
public class MessageBeanImpl implements MessageBean {  
    private String name;  
    private String greeting;  
    @Autowired  
    private Outputter outputter;  
    public MessageBeanImpl(String name) {    this.name = name;    }  
    public void setGreeting(String greeting) { this.greeting = greeting; }  
  
    public void sayHello() {  
        String message = greeting + name + "!";  
        System.out.println(message);  
        try {  
            outputter.output(message);  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

# 다양한 장치로 메시지를 출력하는 빈

```
package sample1;  
import java.io.IOException;  
public interface Outputter {  
    public void output(String message) throws IOException;  
}
```

```
package sample1;  
import java.io.*;
```

```
public class FileOutputter implements Outputter {  
    private String filePath;  
  
    public void output(String message) throws IOException {  
        FileWriter out = new FileWriter(filePath);  
        out.write(message);  
        out.close();  
    }  
  
    public void setFilePath(String filePath) {  
        this.filePath = filePath;  
    }  
}
```

# 다양한 장치로 메시지를 출력하는 빈

```
<?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 />
  <bean id="messageBean" class="sample1.MessageBeanImpl" >
    <constructor-arg value="Spring"/>
    <property name="greeting" value="Hello,"/>
  </bean>
  <bean id="outputter" class="sample1.FileOutputter">
    <property name="filePath">
      <value>kk.txt</value>
    </property>
  </bean>
</beans>
```

# xml을 이용한 bean 생성

- 스프링 빈 설정 파일을 만들고 `<bean id= “아이디” class= “빈의 클래스의 전체 경로” ></bean>`을 이용해서 bean 생성 코드를 작성
- `GenericXmlApplicationContext` 객체를 생성하고 `getBean` 메소드 호출
- `ApplicationContext`는 `IoC` 컨테이너이면서 `Singleton`을 저장하고 관리하는 `Singleton` 레지스트리
- 특별한 설정을 하지 않으면 스프링에서 생성한 빈 오브젝트는 전부 `Singleton` 형태로 생성되고 매개변수가 없는 생성자(디폴트 생성자)를 이용해서 생성
- 싱글톤으로 생성하는 이유는 스프링이 구동되는 환경이 서버 환경일 가능성이 높기 때문
- 서버환경에서 클라이언트의 요청이 올 때마다 각 로직을 담당하는 오브젝트를 새로 만들게 되면 너무나 많은 오브젝트를 생성하게 되어서 성능이 저하되기 때문입니다.
- 스프링의 빈 오브젝트는 기본적으로 `Singleton`의 `scope`를 갖지만 요청이 올 때마다 생성해주는 `prototype scope`도 있고 `request`나 `session scope`도 존재합니다.



# Bean 정의 파일로 DI

- **BeanFactory**는 실행시 건네지는 **Bean**정의 파일을 바탕으로 인스턴스를 생성하고 인젝션을 처리  
**applicationContext.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.springframework.org/schem
a/beans http://www.springframework.org/schema/beans/spring-
beans.xsd">
  <bean id="productService"
        class="sample.di.business.service.ProductServiceImpl"
        autowire="byType" />
  <bean id="productDao"
        class="sample.di.dataaccess.ProductDaoImpl" />
</beans>
```



```
package sample.di.business.domain;
public class Product {
    private String name;    private int price;
    public Product(String name, int price) {
        this.name = name;    this.price = price;
    }
    public String getName() { return name;    }
    public int getPrice() {    return price;    }
    public String toString() {
        return "Product [name=" + name + ", price=" + price + "]";
    }
}
```

```
package sample.di.business.service;
import sample.di.business.domain.Product;
public interface ProductDao {
    Product getProduct();
}
```

```
package sample.di.dataaccess;
import sample.di.business.domain.Product;
import sample.di.business.service.ProductDao;
public class ProductDaoImpl implements ProductDao {
    public Product getProduct() {
        // Dao답게 제품명과 가격을 가진 Product를 검색한 것처럼 반환한다.
        return new Product("호치키스", 100);
    }
}
```

# Bean 정의 파일로 DI

```
package sample.di.business.service;  
import sample.di.business.domain.Product;  
public interface ProductService {  
    Product getProduct();  
}
```

```
package sample.di.business.service;  
import sample.di.business.domain.Product;  
public class ProductServiceImpl implements ProductService {  
    private ProductDao productDao;  
    public void setProductDao(ProductDao productDao) {  
        this.productDao = productDao;  
    }  
    public Product getProduct() {  
        return productDao.getProduct();  
    }  
}
```

# Bean 정의 파일로 DI

```
package samp11;
```

```
import org.springframework.context.support.AbstractApplicationContext;  
import org.springframework.context.support.GenericXmlApplicationContext;
```

```
public class Ex01 {  
    public static void main(String[] args) {  
        AbstractApplicationContext ac = new  
  
        GenericXmlApplicationContext("classpath:beans11.xml");  
        ProductService ps = ac.getBean(ProductService.class);  
        Product product = ps.getProduct();  
        System.out.println(product);  
        ac.close();  
    }  
}
```

# 어노테이션을 사용한 DI

- 스프링은 크게 **Bean**정의 파일을 사용한 **DI**와 어노테이션이  
용한 것이 있다. 앞의 것이 **Bean**을 이용한 것임

## ProductSampleRun.java

```
package samp12;
import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.GenericXmlApplicationContext;
public class Ex01 {
    public static void main(String[] args) {
        AbstractApplicationContext ac = new
            GenericXmlApplicationContext("/samp12/beans12.xml");
        ProductService ps = ac.getBean(ProductService.class);
        Product product = ps.getProduct();
        System.out.println(product);
        ac.close();
    }
}
```

# 어노테이션을 사용한 DI

```
package samp12;  
public class Product {  
    private String name;  
    private int price;  
  
    public Product(String name, int price) {  
        this.name = name; this.price = price;  
    }  
  
    public String toString() {  
        return "제품[이름:"+name+", 가격:"+price+"]";  
    }  
  
}
```

# Annotation 을 이용한 DI - 의존성 주입

## @Autowired

타입을 기준으로 의존성을 주입,  
같은 타입 빈이 두 개 이상 있을 경우 변수이름으로 빈을 찾음  
Spring 아노테이션

## @Qualifier

빈의 이름으로 의존성 주입  
@Autowired와 같이 사용  
Spring 아노테이션

## @Resource

name 속성을 이용하여 빈의 이름을 직접 지정  
JavaSE의 아노테이션

## @Inject

@Autowired 아노테이션을 사용하는 것과 같다  
JavaSE의 아노테이션

# 어노테이션을 사용한 DI

## 1. 어노테이션 설명

**DI**컨테이너는 **@Autowired**가 붙은 인스턴스 변수에 대입할 수 있는 클래스를 **@Component**가 붙은 클래스에서 찾아내 그 인스턴스를 인젝션해준다

따라서 **setter**메소드를 준비할 필요가 없다

**@Component**가 붙은 클래스가 여러 개 있어도 형이 다르면 **@Autowired**가 붙은 인스턴스변수에 인젝션하지 않는다. 이렇게 하는 방법을 **byType**라고 한다

## 2. 주요스키마

- . **bean** 스키마 : **Bean**(컴포넌트)설정
- . **context** : **Bean**검색과 어노테이션 설정
- . **jee** : **JNDI**의 **lookup** 및 **EJB**의 **lookup**설정
- . **util** : 정의와 프로퍼티 파일을 불러오는 등의 유틸리티 기능 설정
- . **lang** : 스크립트 언어를 이용할 경우의 설정
- . **aop** : **AOP**설정
- . **tx** : 트랜잭션 설정
- . **mvc** : **Spring mvc**설정

## 3. 태그

**<context:annotation-config />** **@Autowired**, **@Resource**를 이용할 때 선언

**<context:component-scan base-package="**패키지이름,**..." />**

**@Component**, **@Service**등의 컴포넌트를 이용할 때 선언

# 어노테이션을 사용한 DI

## 4. 인테페이스에 구현클래스가 두개인 경우

1) **@Autowired**와 병행해서 **@Qualifier**를 하는 방법

**@Autowired**

**@Qualifier( “productDao” )**

**private ProductDao productDao**

-----  
**@Component( “productDao” )**

**public class ProductDaoImpl implements ProductDao {**

**...**

**}**

2) **context:component-scan**이용

## 5. @Component 확장

**@Service** : 서비스용 어노테이션으로 트랜잭션관리  
아직 사용하지 않음

**@Repository** : 데이터 액세스층의 **DAO**어노테이션



# ProductDao.java

```
package samp12;  
public interface ProductDao {  
    Product getProduct(String name);  
}
```

## ProductDaoImpl.java

```
package samp12;  
import org.springframework.stereotype.Component;  
import org.springframework.stereotype.Repository;  
/* @Component */  
@Repository  
public class ProductDaoImpl implements ProductDao {  
    // Dao이지만 간단히 하고자 RDB에는 액세스하지 않는다.  
    public Product findProduct() {  
        // Dao답게 제품명과 가격을 가진 Product를 검색한 것처럼 반환한다.  
        return new Product("호치키스", 100);  
    }  
}
```

## ProductService.java

```
package samp12;  
public interface ProductService {  
    Product getProduct();  
}
```

## ProductServiceImpl.java

```
package samp12;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Component;  
import org.springframework.stereotype.Service;  
/* @Component */  
@Service  
public class ProductServiceImpl implements ProductService {  
    @Autowired  
    private ProductDao pd;  
    public Product getProduct() {  
        return pd.getProduct("산딸기");  
    }  
}
```

# an.config/applicationContext.xml

```
<?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.xsd
                           http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.1.xsd">
    <context:component-scan base-
package="samp12"></context:component-scan>
</beans>
```

# 회원가입 입력프로젝트

- exp15
  - Ex01.java
  - Ex02.java
  - JavConfig.java
  - beans15.xml
- exp15.dao
  - MemberDao.java
  - MemberDaoImpl.java
- exp15.model
  - Member.java
  - RegisterMember.java
- exp15.service
  - MemberService.java
  - MemberServiceImpl.java

# Member.java

```
public class Member {  
    private int id;          private String pass;  
    private String email;    private String name;  
    private Date reg_date;  
    public Member(String pass, String email, String name,  
                   Date reg_date) {  
        this.pass = pass; this.name = name;  
        this.email = email; this.reg_date = reg_date;  
    }  
    public int getId() {    return id;    }  
    public void setId(int id) {    this.id = id;    }  
    public String getPass() {    return pass;    }  
    public String getEmail() {    return email;    }  
    public String getName() {    return name;    }  
    public Date getReg_date() {    return reg_date;    }  
    public String toString() {  
        return "아이디:"+id+", 암호:"+pass+", 이름:"+name+  
            ", 이메일:"+email+", 등록일:"+reg_date;  
    }  
}
```

# RegisterMember.java

```
public class RegisterMember {  
    private String pass;private String confirmPassword;  
    private String name;private String email;  
  
    public String getPass() {return pass;}  
    public void setPass(String pass) {this.pass = pass;}  
    public String getConfirmPass() {return confirmPassword;}  
    public void setConfirmPass(String confirmPassword) {  
        this.confirmPass = confirmPassword;  
    }  
    public String getName() {return name;}  
    public void setName(String name) {this.name = name;}  
    public String getEmail() {return email;}  
    public void setEmail(String email) {this.email = email;}  
    public boolean passCheck() {  
        return pass.equals(confirmPass);  
    }  
}
```

# 회원가입 입력프로젝트

```
import java.util.Collection;  
public interface MemberDao {  
    Member selectByEmail(String email);  
    void insert(Member member);  
    void update(Member member);  
    Collection<Member> selectAll();  
}
```

```
import java.util.Collection;  
import java.util.List;  
public interface MemberService {  
    int insert(RegisterMember rm);  
    Member select(String string);  
    Collection<Member> list();  
    int delete(String string);  
    int update(RegisterMember rm);  
}
```

## MemberDao.java

```
package spring1;
import java.util.Collection;
import java.util.HashMap;
import java.util.Map;
public class MemberDao implements MemberDao {
    private static long nextId = 0;
    private Map<String, Member> map = new HashMap<String, Member>();
    public Member selectByEmail(String email) {
        return map.get(email);
    }
    public void insert(Member member) {
        member.setId(++nextId);
        map.put(member.getEmail(), member);
    }
    public void update(Member member) {
        map.put(member.getEmail(), member);
    }
    public Collection<Member> selectAll() {
        return map.values();
    }
}
```



# 회원가입 입력프로젝트

**MemberServiceImpl.java**

```
public class MemberServiceImpl implements MemberService {  
    private MemberDao md;  
    public void setMd(MemberDao md) {  
        this.md = md;  
    }  
    public int insert(RegisterMember rm) {  
        int result = 0;  
        Member member = md.selectByEmail(rm.getEmail());  
        if (member == null) {  
            member = new Member(rm.getPass(),rm.getEmail(),  
                rm.getName(), new Date());  
            md.insert(member);  
            result = 1;  
        } else { System.out.println("이미 데이터가 있습니다");}  
            return result;  
        }  
        public Member select(String email) {  
            return md.selectByEmail(email);  
        }  
        public Collection<Member> list() {  
            return md.list();  
        }
```

# 회원가입 입력프로젝트

```
public int delete(String email) {  
    int result = 0; //데이터가 이미 있는지 확인  
    Member member = md.selectByEmail(email);  
    if (member != null) {  
        md.delete(email);  
        result = 1;  
    } else { System.out.println("없는데 우짜 삭제하니");}  
    return result;  
}  
  
public int update(RegisterMember rm) {  
    int result = 0; //데이터가 이미 있는지 확인  
    Member member = md.selectByEmail(rm.getEmail());  
    if (member != null) {  
        member.setPass(rm.getPass());  
        member.setName(rm.getName());  
        md.update(member);  
        result = 1;  
    } else { System.out.println("없는데 우짜 고치니 ? 헐");}  
    return result;  
}  
}
```

# MbEx.java

```
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import java.util.*;
public class MbEx {
    private static ApplicationContext ac = null;
    public static void main(String[] args) {
        ac = new ClassPathXmlApplicationContext("/sample17/beans17.xml");
        Scanner sc = new Scanner(System.in);
        while(true) {
            System.out.println("명령어를 입력하세요");
            String command = sc.nextLine();
            if (command.equalsIgnoreCase("exit")) {
                System.out.println("종료합니다"); break;
            } else if (command.equals("list")) { processList(); continue;
            } else if (command.startsWith("new ")) {
                processInsert(command.split(" ")); continue;
            } else if (command.startsWith("change ")) {
                processChangeCommand(command.split(" "));
                continue;
            }
            help();
        }
        sc.close();
    }
}
```

# 회원가입 입력프로젝트

```
static void processInsert(String[] str) {  
    if (str.length != 5) {  
        help(); return;  
    }  
    RegisterMember req = new RegisterMember();  
    req.setEmail(str[1]);  
    req.setName(str[2]);  
    req.setPassword(str[3]);  
    req.setConfirmPassword(str[4]);  
    if (!req.passConfirm()) {  
        System.out.println("암호와 암호확인이 다릅니다");  
        return;  
    }  
    MemberInsert mi = (MemberInsert) ac.getBean("mi");  
    int result = mi.insert(req);  
    if (result > 0)  
        System.out.println("등록 완료");  
}
```

```
static void processChangeCommand(String[] arg) {
    if (arg.length != 4) {
        help();
        return;
    }
    ChangePassword changePwdSvc =
        ac.getBean("changePwdSvc", ChangePassword.class);
    int result = changePwdSvc.changePassword(arg[1], arg[2], arg[3]);

    if (result > 0) System.out.println("암호를 변경했습니다.\n");
}
```

```
static void help() {
    System.out.println();
    System.out.println("잘못된 명령입니다.");
    System.out.println("명령어 사용법:");
    System.out.println("new 이메일 이름 암호 암호확인");
    System.out.println("change 이메일 현재비번 변경비번");
    System.out.println("list");
    System.out.println("exit");
    System.out.println();
}
}
```

## Beans17.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-
beans.xsd">
  <bean id="md" class="sample17.MemberDaoImpl"></bean>
  <bean id="mp" class="sample17.MemberPrinter"></bean>
  <bean id="lp" class="sample17.ListPrintImpl">
    <property name="md" ref="md"></property>
    <property name="mp" ref="mp"></property>
  </bean>
  <bean id="mi" class="sample17.MemberInsertImpl">
    <property name="md" ref="md"></property>
  </bean>
</beans>
```

## **beans18.xml**

```
<?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.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">

<context:component-scan base-package="sample18.*">
</context:component-scan>

</beans>
```



# 자바코드를 이용한 설정

- **Annotation**을 이용한 **Factory** 클래스 생성
  - 클래스 상단에 **@Configuration**을 추가
  - **Factory** 메소드의 상단에 **@Bean**을 추가
- **Annotation**을 이용한 **Factory** 클래스로 빈 생성
  - **AnnotationConfigApplicationContext** 클래스의 객체를 생성한 후 **getBean**을 호출하면 됩니다.



# JavaConfig.java

```
package spring5;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
@Configuration
public class JavaConfig {
    @Bean
    public MemberDao memberDao() {
        return new MemberDao();
    }
    @Bean
    public MemberRegisterService memberRegSvc() {
        return new MemberRegisterService(memberDao());
    }
    @Bean
    public MemberPrinter printer() {        return new MemberPrinter();    }
    @Bean
    public MemberInfoPrinter infoPrinter() {
        MemberInfoPrinter infoPrinter = new MemberInfoPrinter();
        infoPrinter.setMemberDao(memberDao());
        infoPrinter.setPrinter(printer());
        return infoPrinter;
    }
}
```

# Main.java

```
package spring5;import org.springframework.context.ApplicationContext;
import
org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class Main {
    public static void main(String[] args) {
        ApplicationContext ctx =
            new AnnotationConfigApplicationContext(JavaConfig.class);
        MemberRegisterService regSvc =
            ctx.getBean("memberRegSvc", MemberRegisterService.class);
        MemberInfoPrinter infoPrinter =
            ctx.getBean("infoPrinter", MemberInfoPrinter.class);

        RegisterRequest regReq = new RegisterRequest();
        regReq.setEmail("kbj010@hanmail.net");
        regReq.setName("주니");
        regReq.setPassword("1234");
        regReq.setConfirmPassword("1234");

        regSvc.regist(regReq);
        infoPrinter.printMemberInfo("kbj010@hanmail.net");
    }
}
```

# XML을 이용한 DI - 컬렉션 타입 의존성 설정

| 태그      | 타입                   |
|---------|----------------------|
| <list>  | java.util.List 또는 배열 |
| <set>   | java.util.Set        |
| <map>   | java.util.Map        |
| <props> | java.util.Properties |

# XML을 이용한 DI – List 타입 의존성 설정

```
package list;  
import java.util.List;  
public class CollectionBean {  
    private List<String> addressList;  
    public void setAddressList(List<String> addressList) {  
        this.addressList = addressList;  
    }  
    public List<String> getAddressList() {    return addressList;    }  
}
```

```
<bean id="collectionBean" class="list.CollectionBean">  
    <property name="addressList">  
        <list>  
            <value>서울 강남구 역삼동</value>  
            <value>서울 성동구 행당동</value>  
        </list>  
    </property>  
</bean>
```

# XML을 이용한 DI – List 타입 의존성 설정

```
package list;
import java.util.List;
import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class Ex01 {
    public static void main(String[] args) {
        AbstractApplicationContext ac =
            new
ClassPathXmlApplicationContext("/list/applicationContext.xml");
        CollectionBean bean = ac.getBean(CollectionBean.class);
        List<String> addressList = bean.getAddressList();
        for (String addr : addressList) {
            System.out.println(addr);
        }
        ac.close();
    }
}
```

# XML을 이용한 DI – Set 타입 의존성 설정

```
package set;
import java.util.Set;
public class CollectionBean {
    private Set<String> addressSet;
    public Set<String> getAddressSet() { return addressSet; }
    public void setAddressSet(Set<String> addressSet) {
        this.addressSet = addressSet;
    }
}
```

```
<bean id="collectionBean" class="set.CollectionBean">
  <property name="addressSet">
    <list>
      <value>서울 강남구 역삼동</value>
      <value>서울 성동구 행당동</value>
      <value>서울 성동구 행당동</value>
    </list>
  </property>
</bean>
```

# XML을 이용한 DI – Set 타입 의존성 설정

```
package set;
import java.util.Set;
import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class Ex01 {
    public static void main(String[] args) {
        AbstractApplicationContext ac =
            new
        ClassPathXmlApplicationContext("/set/applicationContext.xml");
        CollectionBean bean = ac.getBean(CollectionBean.class);
        Set<String> addressSet = bean.getAddressSet();
        for (String addr : addressSet) {
            System.out.println(addr);
        }
        ac.close();
    }
}
```

# XML을 이용한 DI – Map 타입 의존성 설정

```
package map;
import java.util.Map;
public class CollectionBean {
    private Map<String, String> addressMap;
    public Map<String, String> getAddressMap() { return addressMap; }
    public void setAddressMap(Map<String, String> addressMap) {
        this.addressMap = addressMap;
    }
}
```

```
<bean id="collectionBean" class="map.CollectionBean">
    <property name="addressMap">
        <map>
            <entry>
                <key><value>고길동</value></key>
                <value>서울 강남구 역삼동</value>
            </entry>
            <entry>
                <key><value>마이클</value></key>
                <value>서울 성동구 행당동</value>
            </entry>
        </map>
    </property>
</bean>
```



# XML을 이용한 DI – Properties 타입 의존성 설정

```
package properties;
import java.util.Properties;
public class CollectionBean {
    private Properties addressPro;
    public Properties getAddressPro() {
        return addressPro;
    }
    public void setAddressPro(Properties addressPro) {
        this.addressPro = addressPro;
    }
}
```

```
<bean id="collectionBean" class="properties.CollectionBean">
    <property name="addressPro">
        <props>
            <prop key="고길동">역삼동</prop>
            <prop key="나야나">관악구</prop>
        </props>
    </property>
</bean>
```

# Spring API



<https://docs.spring.io/spring-framework/docs/4.3.28.RELEASE/javadoc-api/>

