스프링 개요와 제어의 역전

스프링등장 배경

1. 스프링의 역사

2004년 **03**월	**1.0**	**DIxAOP**컨테이너 시작**, Bean**정의 파일시대
2004년 **09**월	**1.1**	**Bean**정의파일 간략화
2005년 **05**월	**1.2**	**Bean**정의 파일 간략화 보완
2006년 **10**월	**2.0**	**Bean**정의파일이 **DTD**로부터 **XML**스키마형식으로 변경 어노테이션의 등장 **JPA**와 스크립트언어의 지원과 다기능화
2007년 **11**월	**2.5**	어노테이션 강화
2009년 **12**월	**3.0**	어노테이션 추가 강화했으나 대기업중심으로 **Bean**정의 선호
2011년	**3.1**	클라우드 시대에 대응**,** 스프링에 캐시기능 등장
2014년	**4.0**	스프링 **4** 시작

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

3. EJB의 등장과 쇠퇴

- 분산처리와 트랜잭션의 융합컴포넌트
  분산된 **EJB**컴포넌트를 마치 하나의 데이터베이스만 있는 것처럼 제어

- **EJB**란 원래 분산환경을 위한 컴포넌트로써 등장했고
  JSP, Servlet과 같은 웹어플리케이션 기술은 아니었음

- **EJB**는 웹 어플리케이션용 재사용 가능한 컴포넌트나
  SQL기술이 필요없는 **DB**프레임워크처럼 사용

- **JSP**와 **Servlet**으로 프레젠테이션을 구현하고
  비즈니스 로직은 **EJB**로 구현하는 것을 웹 어플리케이년의 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

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) DI: Dependency Injection지원

설정파일이나 어노테이션을 통해 객체간의 의존 관계를 설정하므로 의존하고 있는 객체를 직접 생성하거나 검색할 필요가 없음

3) IoC : Inversion of Control(제어의 역전**)**

- 프로그램의 제어 흐름 구조가 바뀌는 것
- 일반적인 프로그램은 시작하는 시점에서 다음에 사용할 오브젝트를 결정하고 생성해서 호출하고 사용함. 라이브러리처럼 어플리케이션에서 필요할 때 호출하여 사용함
- 제의의 역전은 제어권한을 다른 대상에게 위임하고 모든 오브젝트는 위임 받은 특별한 오브젝트에 의해 결정되고 만들어짐, 프레임 워크처럼 개발한 클래스를 등록해 주고 프레임워크에 의해서 어플리케이션이 사용되도록 함

4) AOP(Aspect Oriented Programming)지원

- 모든 부가기능과 적용 대상의 조합을 통해 여러 오브젝트에 산재해 있는 공통적인 기능을 쉽게 개발하고 관리하는 기능

5) POJO(Plain Old Java)지원 – 특정한 인터페이스를 구현 하거나 클래스를 상속 받지 않아도 됨, 즉 기존에 작성한 코드를 수정 없이 사용 가능함

6) **transaction**을 처리하기 위한 일관된 방법 제공 : **JDBC**를 사용하든 **JTA**를 사용하든 또는 컨테이너가 제공하는 트랜잭션을 사용하든 설전파일을 통하여 트랙젝션 정보를 입력하기 때문에 **transaction** 구현에 관계없이 동일한 코드를 여러 환경에서 사용할 수 있음

7) 연속성과 관련된 다양한 **API**지원 – **JDBC, iBATIS,** 하이버네이트, **JPA** 등 데이터베이스 처리라이브러리 연동 지원

8) 다양한 **API**연동 지원 – **JMS,** 메일, 스케줄링

3. 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**

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

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 비지니스 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.
IoC를 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

3개발환경

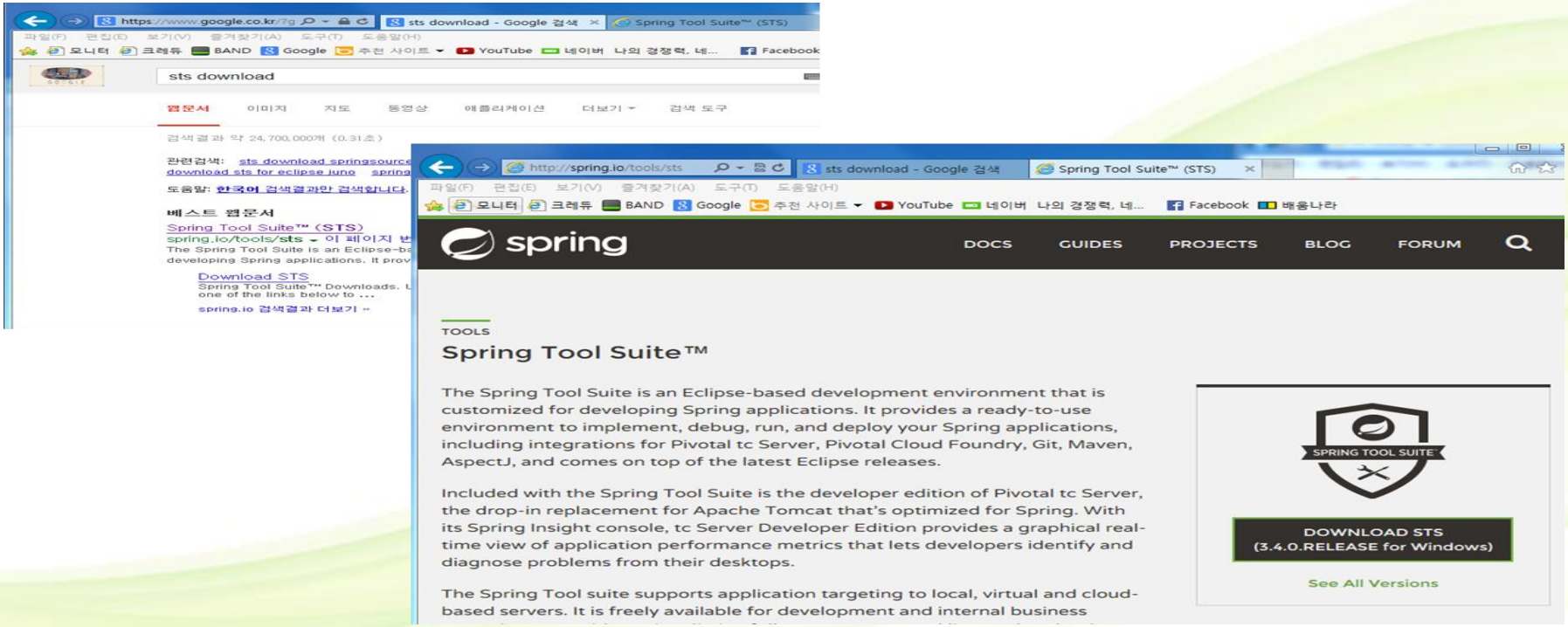
1. STS란?

- 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.

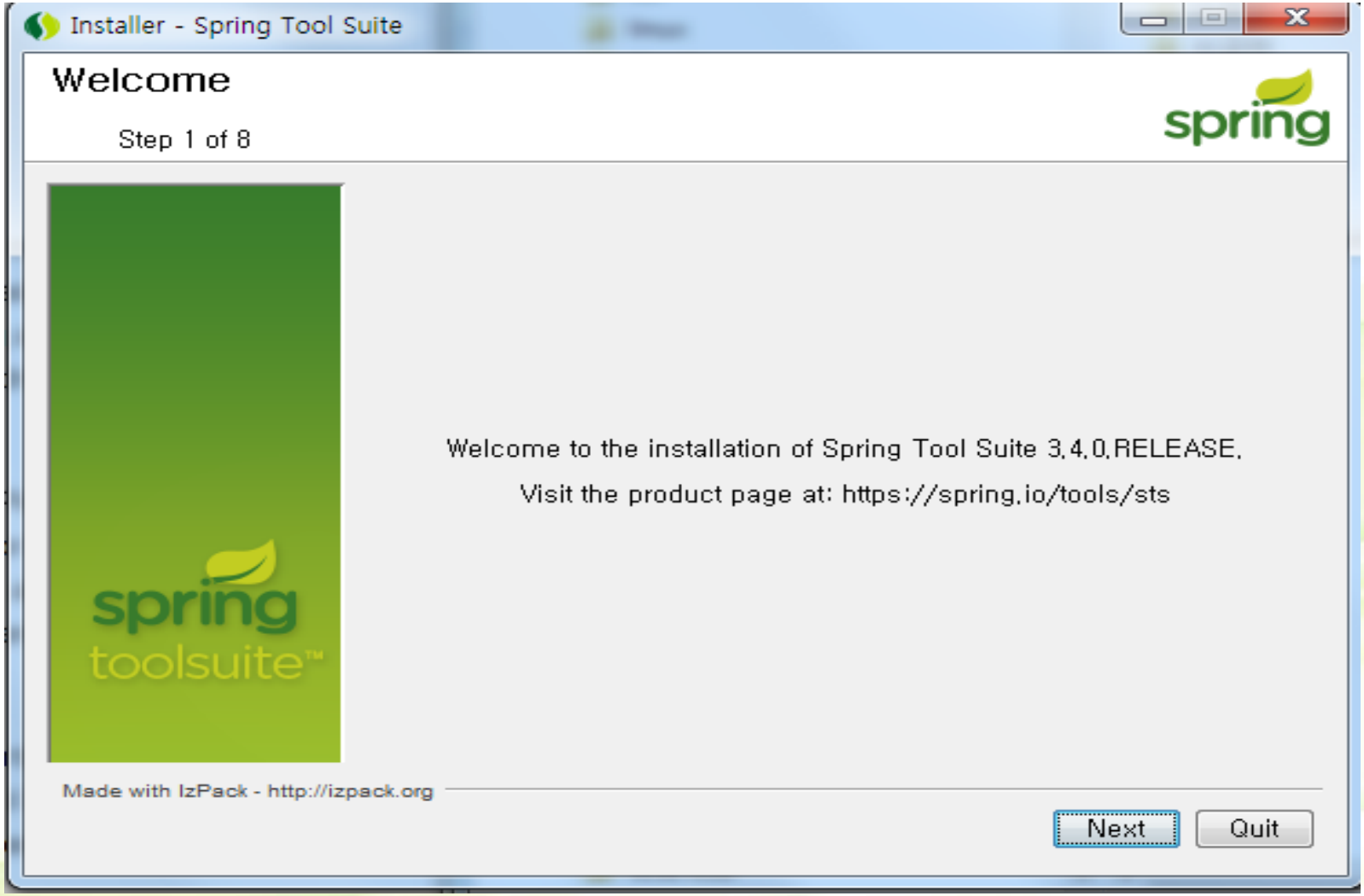
- **DownLoad : google**에서 **sts download**조회

http://spring.io/tools/sts

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

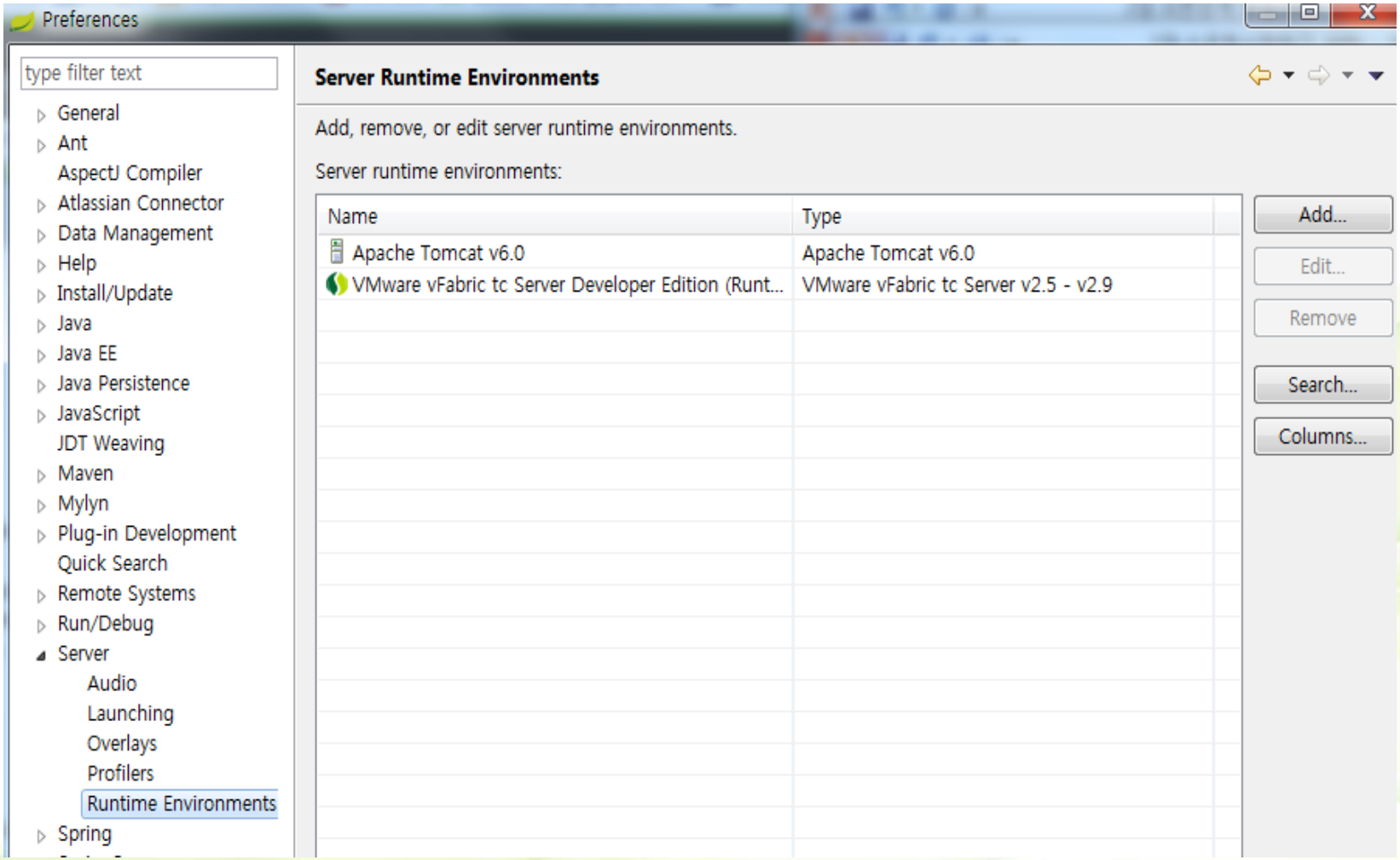


스프링 프레임워크**(Spring Framework)** 플러그인 및 실제 라이브러리를 설치, 설정



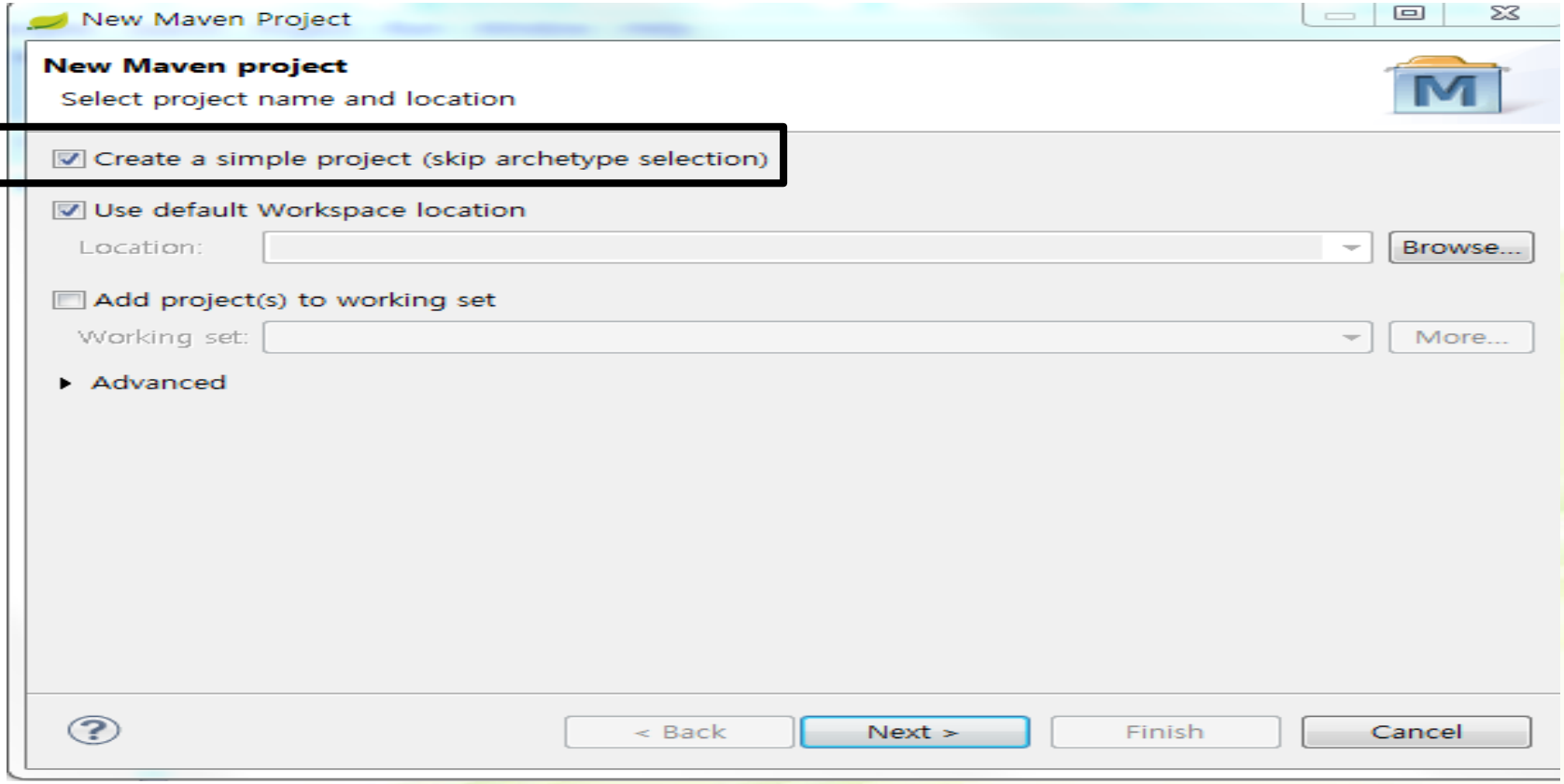
톰캣설정

Window- Preferences
Server – Runtime Environments를 선택



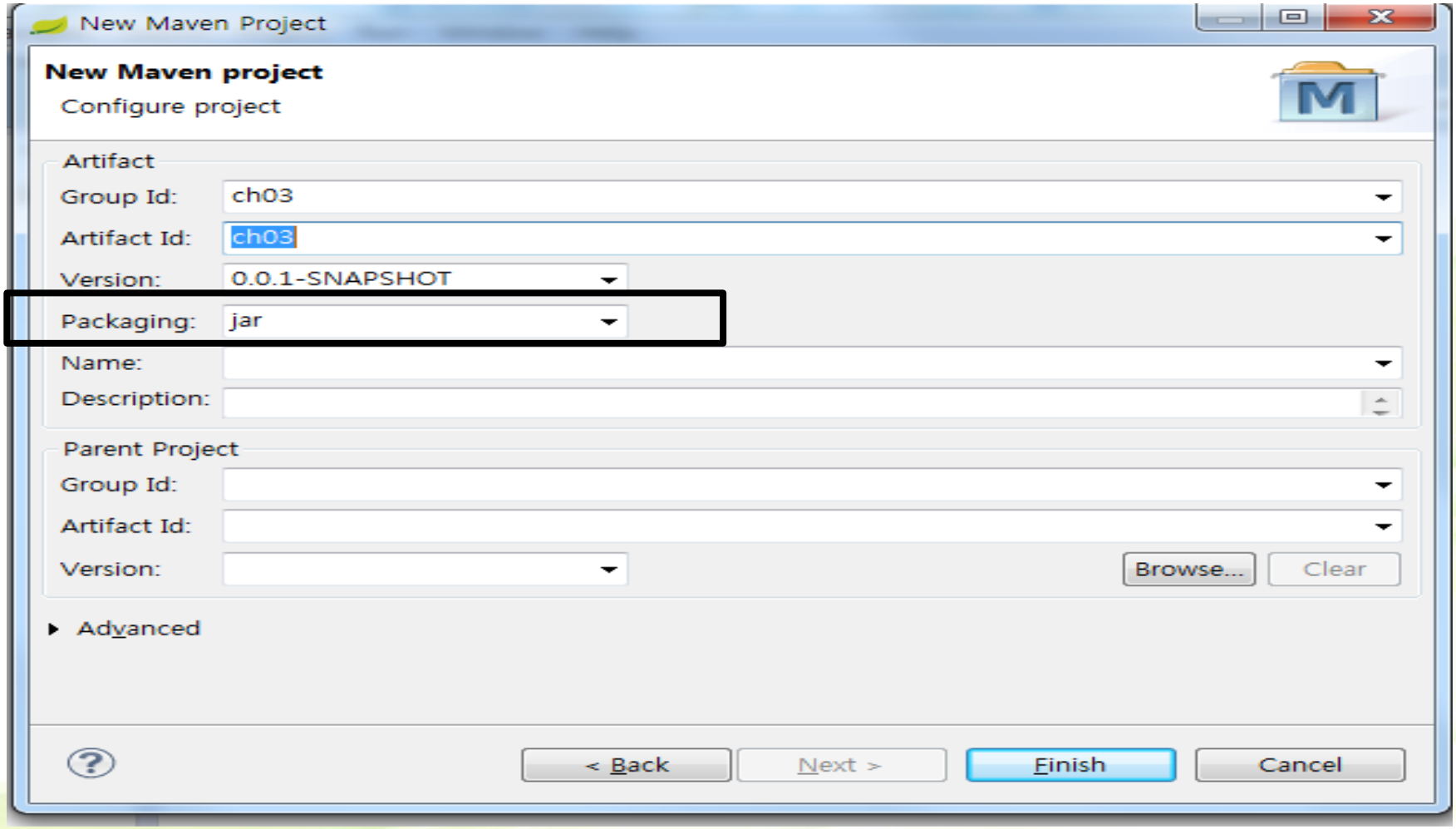
프로젝트 만들기

File – new – maven Project
create a simple project에 체크하고 **next**



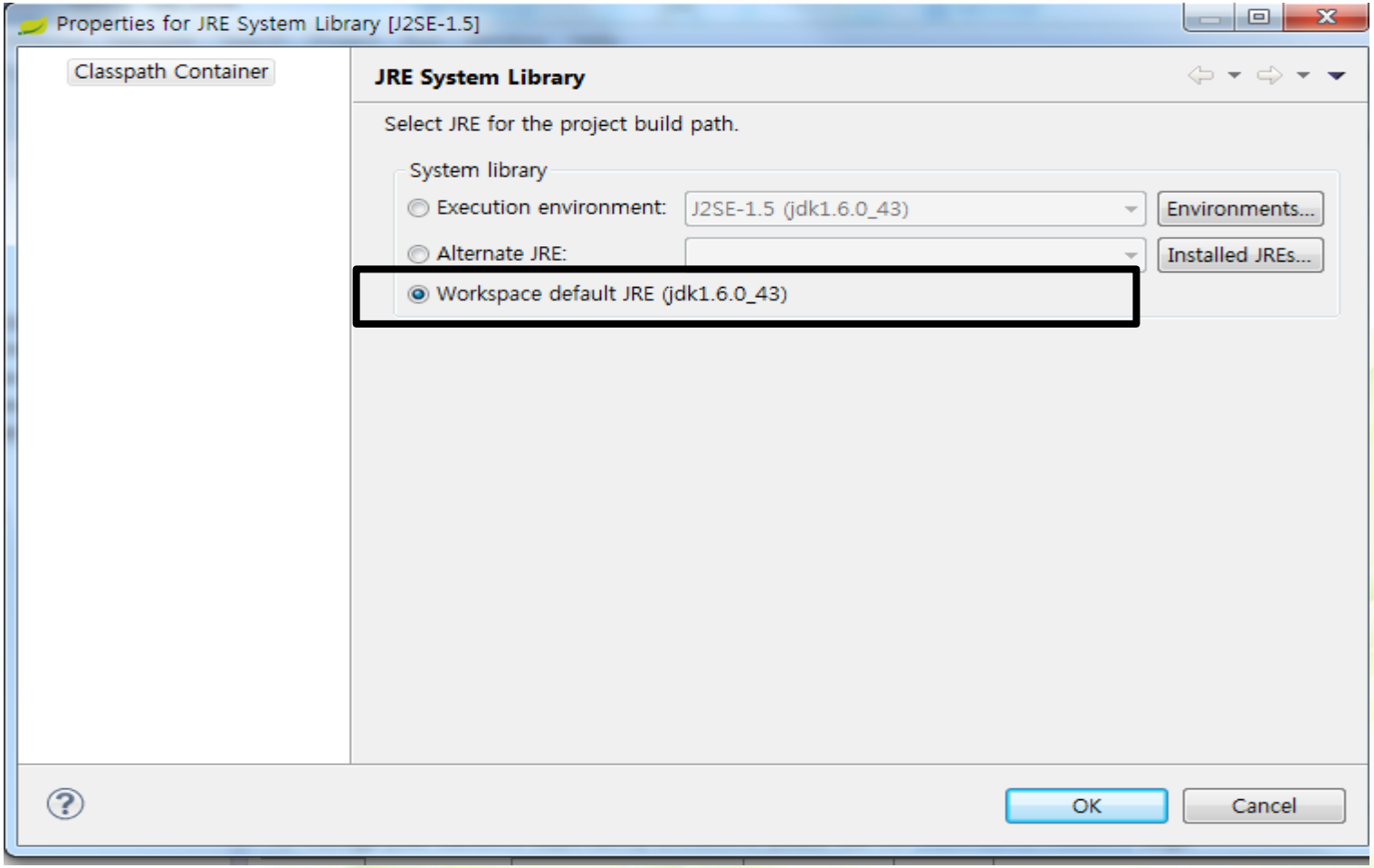
ID만들기

Group ID, Artifact ID를 입력하고
Packaging에 **web**일 때는 **war** 그밖에는 **jar**선택



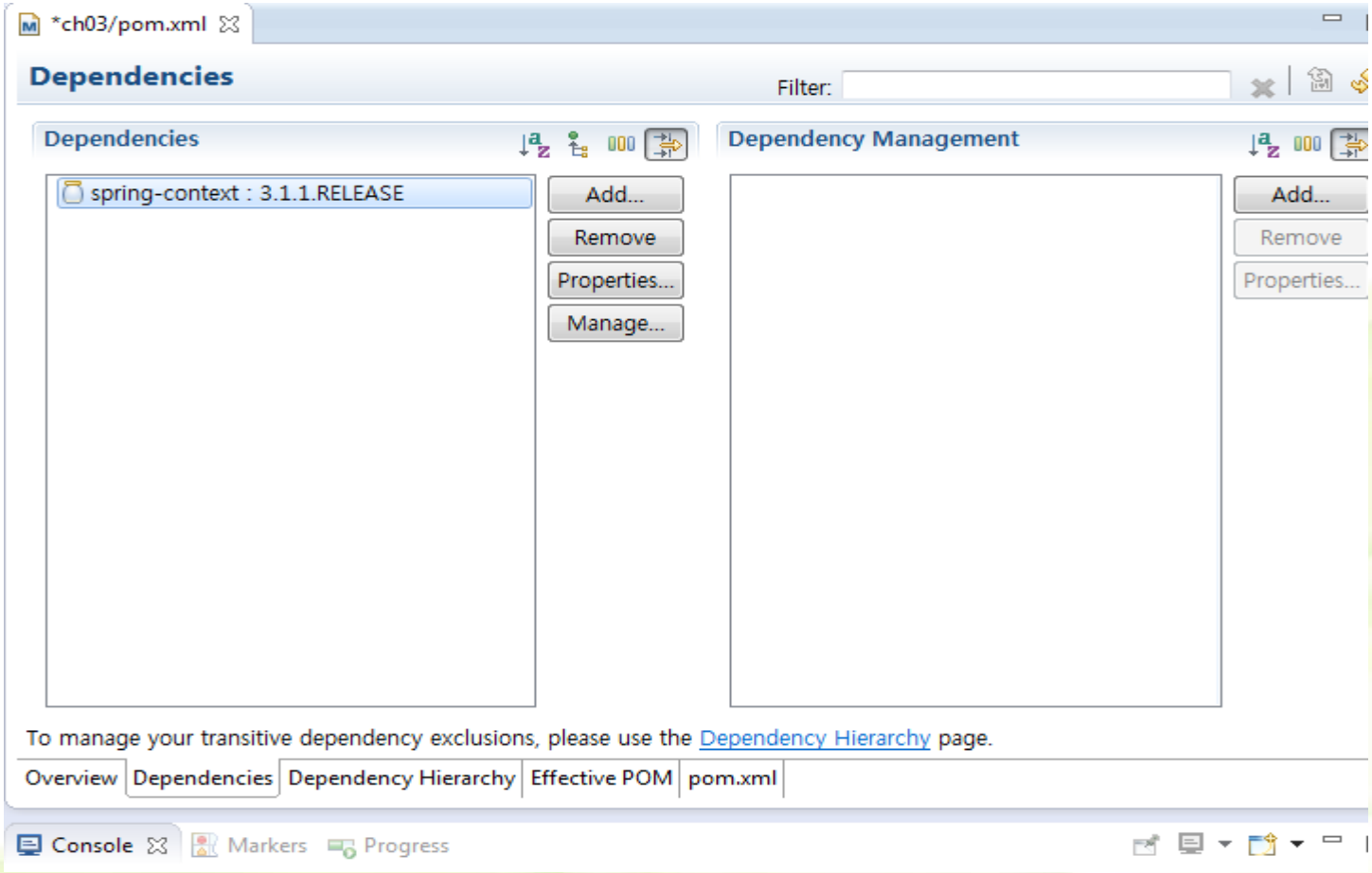
JDK설정

JRE System Library를 선택하고 우측 버튼을 클릭하여 **Propeties**를 선택한 후에 **workspace default**선택



의존 라이브러리 설정

Poem.xml에서 **dependencies**탭을 열고 **add**를 선택한후
Group ID : org.springframework
Artifact ID : spring-context
Version : 3.1.1.RELEASE
Scope : compile

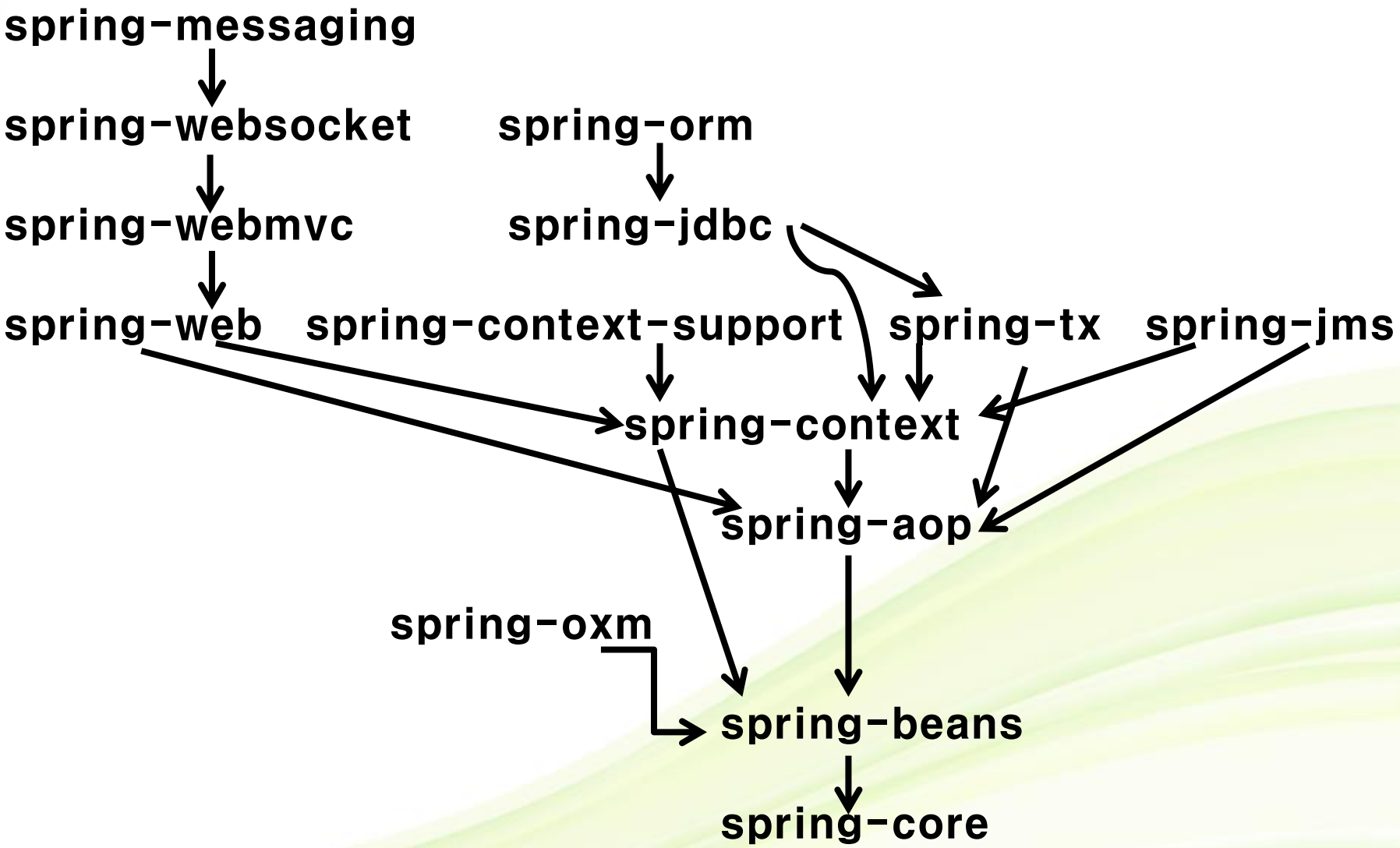


스프링의 주요 모듈 목록

모듈 명	설 명
core	DI기능을 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 AplicationContext 인터페이슬 통해 구현
context.support	Ehcache, 메일, 스케줄링, UI의 Velocity 지원 기능을 재공한다
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.

모듈 명	설 명
Transaction	AOP을 이용한 선언적 트랜잭션 관리 및 코드를 이용한 트랜잭션 관리 기능을 제공한다
Jdbc	JDBC프로그랭을 뤼한 추상 레이어를 제공한다. IDBC 탬플릿을 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 TestING를 이용한 스프링 컴퍼넌트의 테스트를 지원한다
instrument	Instrumentation 지언 클래스를 제공한다
instrument.tomcat	톰갯서버를 위한 instrumentation지원클래스를 제공한다
asm	ASM 라이브러리를 재패키징한 모듈

스프링 주요 모듈의존 관계



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 설정값입력
- **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 ?

1. Maven 소개

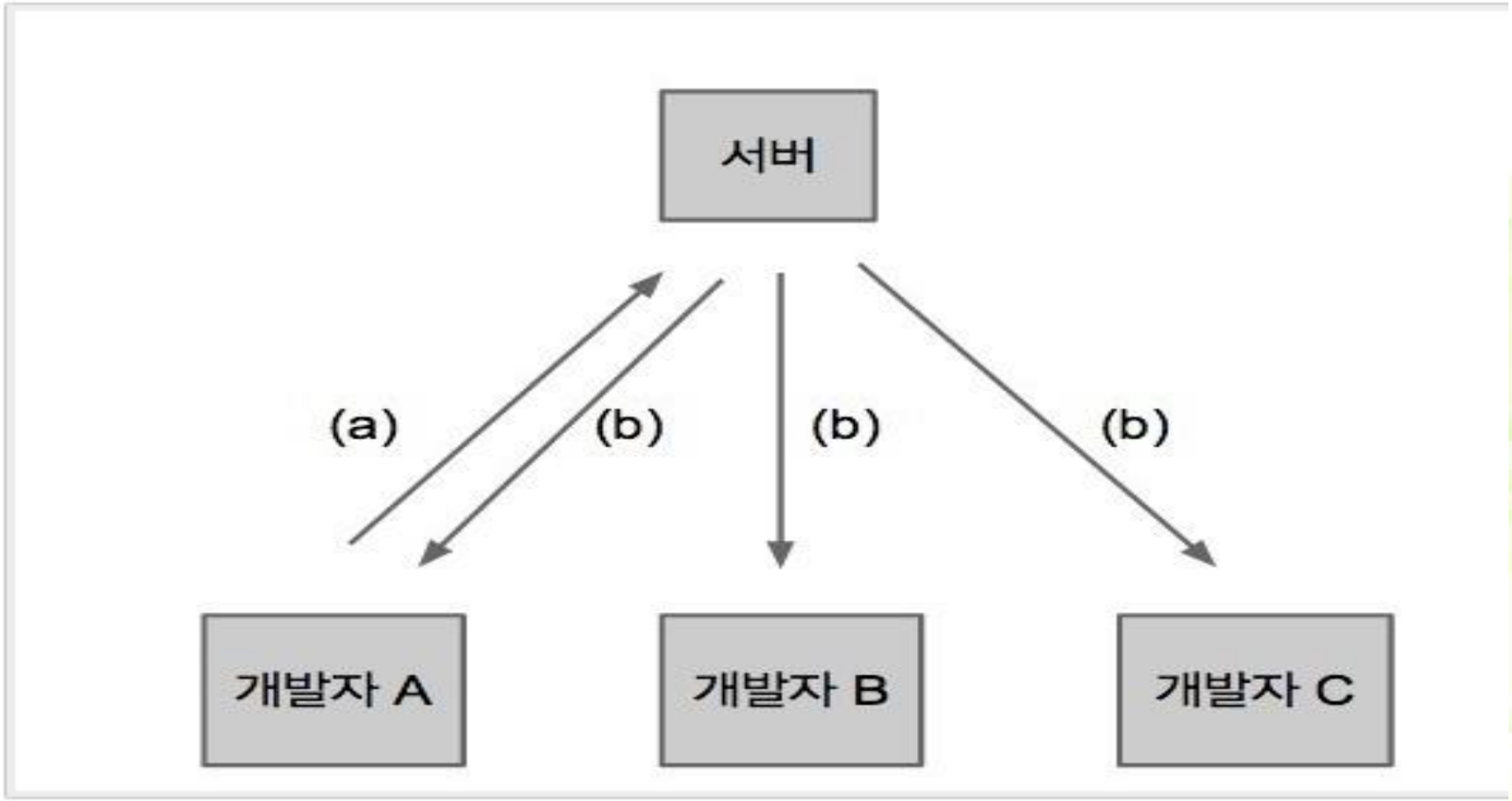
이전에는 자바 프로젝트를 시작할 때 제일 먼저하는 작업이 이클립스를 띄우고, 자바 프로젝트를 만들어 라이브러리를 lib폴더에 추가하는 것이었다. 이러한 과정에 들어가기 전에 아래와 같은 고민이 생겼다.

> 1.프로젝트가 빌드에 필요한 것이 무엇인가?
> 2.다운로드 받을 필요가 있는 라이브러리가 어떤 것인가?
> 3.다운로드 받은 것을 어디에 넣어야 하는가?
> 4.빌드에서 어떤 goal을 실행할 수 있는가?

이런 고민 때문에 잘될때는 새로운 프로젝트의 빌드를 해결하는데 최소 몇분이 걸리고, 잘 되지 않을 때는 프로젝트의 빌드가 어려워서 새로운 개발자가 소스를 수정하고 프로젝트를 컴파일하는 지점에 도달하는데 수시간이 걸리는 현상이 발생하였다.

2. Maven의 동작원리

이제 Maven의 동작원리에 대해 간단히 알아보자.



1.개발자 A가 "embian"이라는 라이브러리를 만들었다.

1.1 이 라이브러리는 완벽하지 않고, 다른기능을 추가해야 할수 있다.

1.2 그래서 여러 개발자가 필요시 이 라이브러리를 사용해야 한다

2. 라이브러리를 만들고 난후 Maven툴을 이용하여 빌드 한 후 서버에 올린다.(a)

3. 개발자 B와 개발자C가 "embian"라이브러리가 필요한 경우 프로젝트마다 설정되어 있는 "pom.xml"이라는 파일을 설정하면 Maven툴이 알아서 서버에 접속하여 "embian"라이브러리를 다운받아 자신의 프로젝트에 자동으로 추가한다.(b)

4.필요에 따라 "pom.xml" 파일을 설정하면 Maven툴은 주기적으로 서버와 통신하여 최신버전의 "embian"라이브러리를 다운받게 할 수 있다. 개발자 A 혹은 다른 개발자가 "embian"라이브러리를 수정하여 서버에 업로드 할 수 있기 때문이다. (a)

5."embian"라이브러리를 사용하는 "embian2"라는 라이브러리가 있을 경우 Maven툴은 알아서(의존성을 체크하여) "embian2"와 "embian"라이브러리를 다운받는다.

6. Maven툴은 여러 플러그인을 hudson이나 Ant,Nexus등의 유명툴과 연동할수 있다.

3.정리

Maven을 소개하고 동작원리까지 알아보았으니 이제 정리하면서 다시 한번 말해 Maven은 무엇일까?

그 답은 여러분의 관점에 달렸다.

대부분의 Maven사용자들은 Maven을 소스 코드로부터 배포 가능한 산출물을 만드는데 사용되는 도구인 "build tool"이라고 부를 것이고, 빌드 담당자와 프로젝트 매니저들은 Maven을 프로젝트 관리 툴로서 더 종합적인 무언가 라고도 할것이다.

Maven과 같은 프로젝트 관리 도구는 빌드 도구에서 발견되는 기능의 상위 개념을 제공한다. 빌드 기능을 제공하는 것 이외에, Maven은 레포트를 실행하고, 웹 사이트를 만들며, 작업 팀의 멤버 간의 커뮤니케이션을 가능하게 해준다.

다음은 Apache Maven의 공식적인 정의이다.

Maven은 프로젝트 객체 모델 (Project Object Model), 표준의 집합, 프로젝트 라이프사이클, 의존성 관리 시스템, 라이프사이클에 정의된 단계에서 플러그인 목표를 실행하는 논리를 포함하는 프로젝트 관리 도구이다.

여러분이 Maven을 사용할 때 잘 정의된 프로젝트 객체 모델을 사용한 프로젝트를 기술하며, Maven이 그 다음에 공유된(혹은 사용자가 만든) 플러그인들로부터 cross-cutting 로직을 적용할 수 있다. Maven이 프로젝트 관리 도구라는 사실에 두려움을 느끼겠지만 만약 당신이 빌드 도구를 찾는다면 Maven이 그 역할을 수행할 것이다.

Maven이 성공할 수 있었던 핵심 근거는 소프트웨어를 빌드하는데 공통 인터페이스를 정의했다는 것이다.아파치 Wicket과 같은 프로젝트가 Maven 사용하는 것을 살펴보면 소스를 체크아웃 받고 부담없이 mvn install을 사용해서 빌드를 한다.

IoC(Inversion of Control)

❖일반적인 프로그램에서의 제어는 오브젝트를 직접 생성하고 그 오브젝트를 이용해서 메소드를 호출하는 형태입니다.

❖제어의 역전이란 앞에서의 코드처럼 직접 객체를 생성하지 않고 다른 곳에서 생성해준 객체를 가지고 메소드를 호출합니다.

❖실제 자신이 객체를 생성하는 것이 아닌 형태인데 JSP에서 서블릿은 메인 메소드가 없고 우리가 직접 생성자를 호출한 적도 없지만 서블릿을 사용을 했는데 이는 서블릿에 대한 제어 권한을 가진 컨테이너가 적절한 시점에 서블릿 클래스의 오브젝트를 생성하고 그 안의 메소드를 호출하는 형태가 되는데 이도 제어의 역전입니다.

❖프레임워크나 dbcp를 이용한 데이터베이스 활용도 제어의 역전을 활용한 예로 객체 생성 시 객체의 생성자를 직접 호출하지 않을 것입니다.

IoC(Inversion of Control)

❖스프링에서는 스프링이 제어권을 가지고 직접 만들고 관계를 설정하는 오브젝트를 Bean이라고 합니다.

❖오브젝트 단위의 애플리케이션 컴포넌트를 빈이라고 합니다.

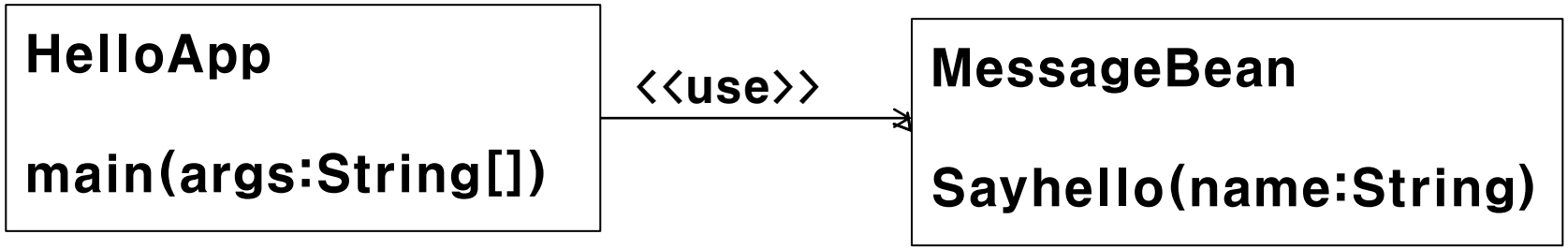
❖스프링 빈은 스프링 컨테이너가 생성과 관계설정, 사용 등을 제어해주는 제어의 역전이 적용된 오브젝트를 가리키는 말입니다.

❖이렇게 빈의 생성과 관계설정 같은 제어를 담당하는 IoC 오브젝트를 빈 팩토리(bean factory)라고 합니다.

❖빈 팩토리는 주로 application context를 주로 사용합니다.

❖Application context는 평범한 자바 코드로 객체를 생성하는 것이 아니고 설정정보를 담고 있는 무엇인가를 가져와서 이를 활용한 IoC 엔진입니다.

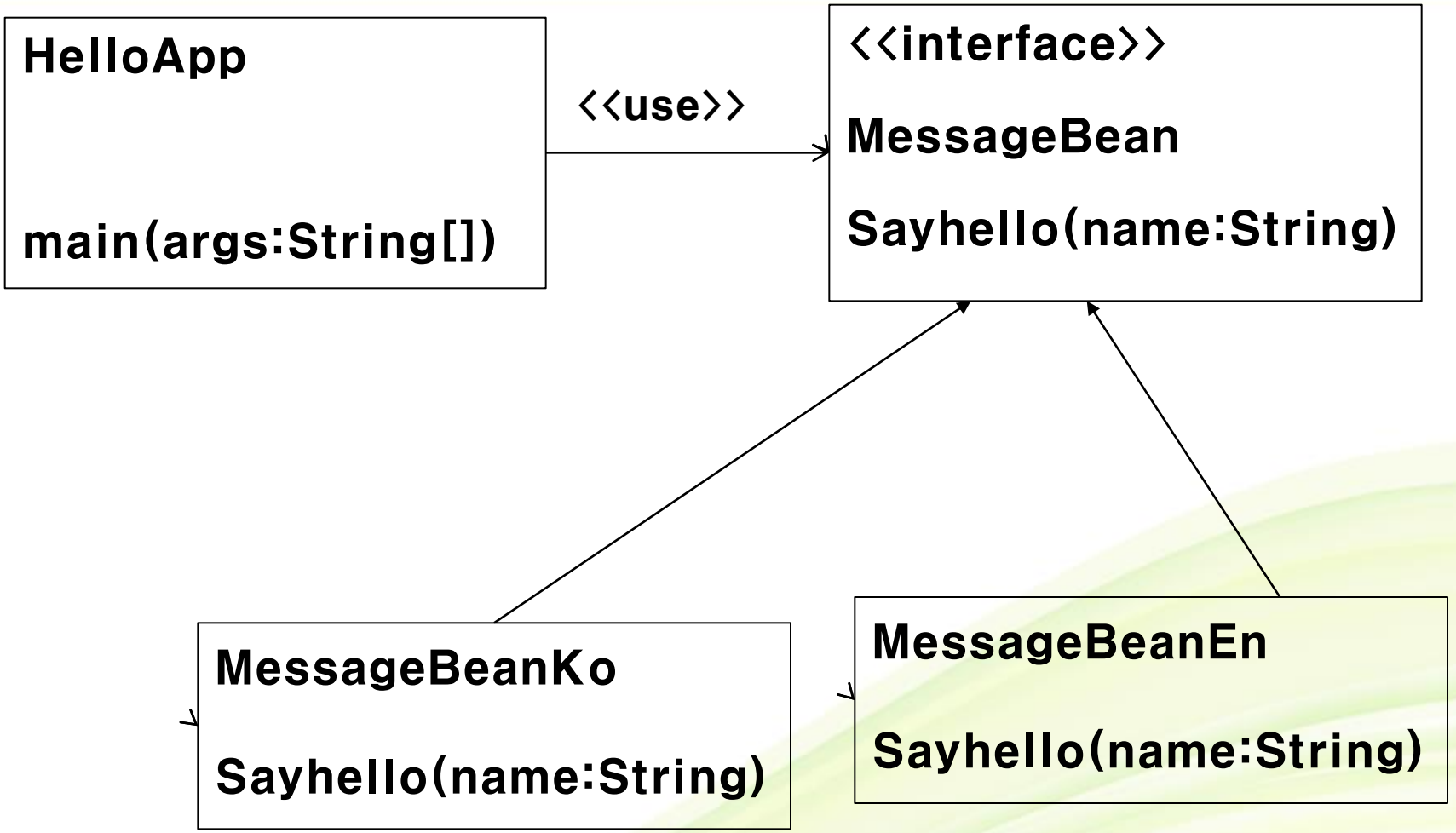
Sample01




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

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

Sample02



클래스간의 의존성을 약하게 하기 위해서 인터페이스 사용

결합도, 응집도

세 개의 예제를 통한 스프링의 기본 기능과 구조 이해하기

(2) 인터페이스 적용한 샘플 어플리케이션 분석하기(sample2)

◈ MessageBean.java

```
package sample2;
public interface MessageBean {
 void sayHello(String name);
}
```

◈ MessageBeanEn.java

```
package sample2;
public class MessageBeanEn implements MessageBean {
 @Override
 public void sayHello(String name) {
    System.out.println("Hello," +name+ "!");
 }
}
```

세 개의 예제를 통한 스프링의 기본 기능과 구조 이해하기

(2) 인터페이스 적용한 샘플 어플리케이션 분석하기(sample2)

MessageBeanKo.java

```
package sample2;
public class MessageBeanKo implements MessageBean {
 @Override
 public void sayHello(String name) {
    System.out.println("안녕하세요!" +name+ "씨");
 }
}
```

HelloApp.java

```
package sample2;
public class HelloApp {
 public static void main(String[ ] args) {
    MessageBean bean = new MessageBeanEn( );
    //MessageBean bean = new MessageBeanKo( );
    bean.sayHello( "Spring" );
 }
}
```

세 개의 예제를 통한 스프링의 기본 기능과 구조 이해하기

(2) 인터페이스 적용한 샘플 어플리케이션 분석하기(sample2)

```
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

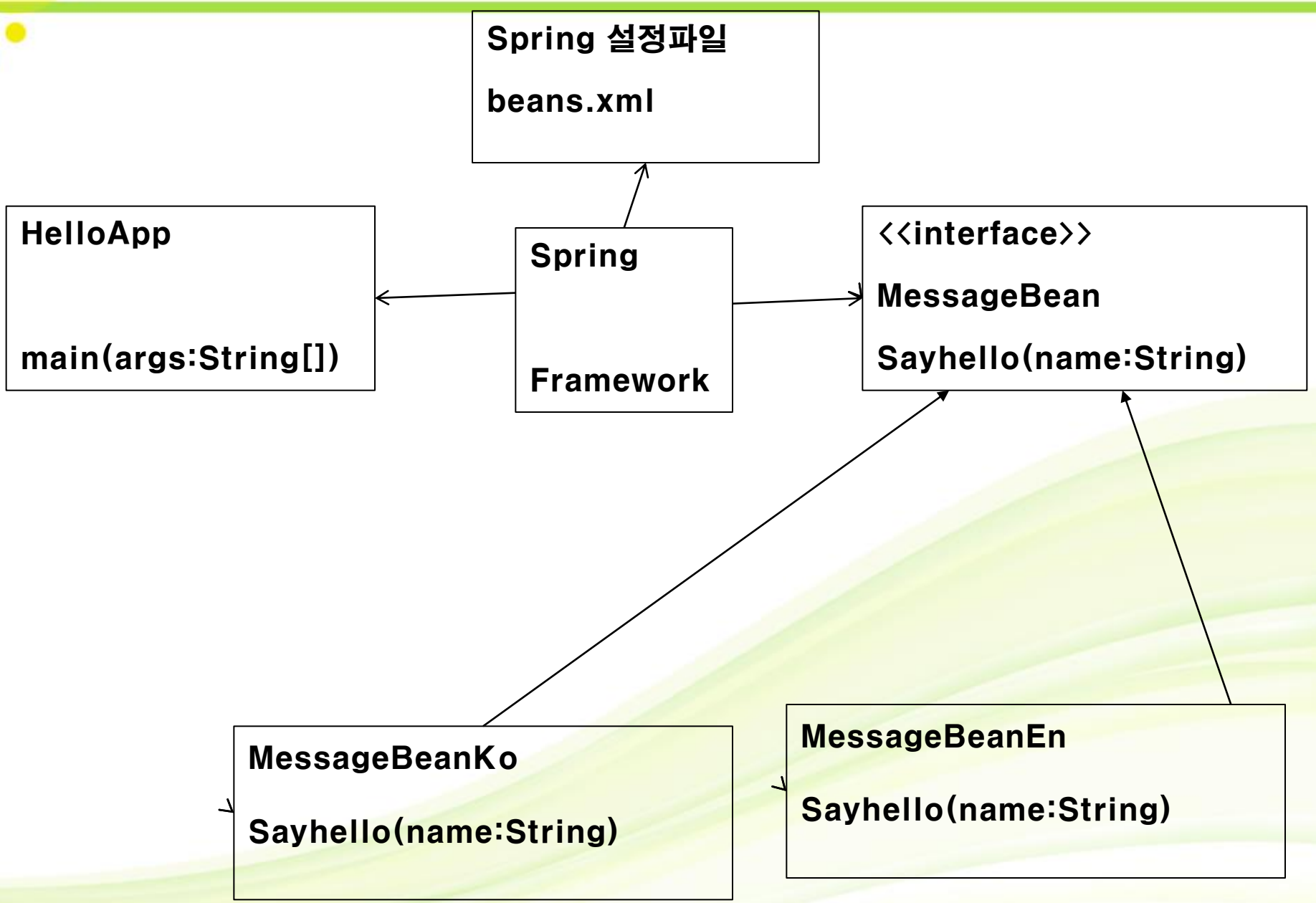
BeanFactory 인터페이스

org.springframework.beans.factory.BeanFactory 인터페이스는 빈 객체를 관리하고, 각 빈 객체간의 의존 관계를 설정해 설정해 주는 기능을 제공하는 가장 단순한 컨터이너이다. 구현 클래스로는 **org.springframework.beans.xml.XmlBeanFactory**클래스이다
XmlBeanFactory는 외부자원으로부터 설정정보를 와 빈 객체를 생성한다.
스프링은 **org.springframework.core.io.Resource** 인터페이스를 사용하여 다양한 종류의 자원을 동일한 방식으로 표현할 수 있도록 하고 있으며 **Resource**를 이용하여 **XmlBeanFactory**에 생성 정보를 전달할 수 있다.

```
Resource resource = new FileSystemResource( "applicationContext.xml");
BeanFactory factory = new XmlBeanFactory(resource );
MessageBean bean = (MessageBean)factory.getBean("messageBean");
```

FileSystemResource : 파일시스템의 특정 파일로부터 정보를 읽어온다
InputStreamReource : **input Stream**으로부터 정보를 읽어 온다
ClassPathResource ; 클래스 패스에 있는 자원으로부터 정보를 읽어온다
UrlResource : 특정 **URL**로부터 정보를 읽어 온다
ServletContextResource : 웹 어풀리케이션의 루트 디렉톨;를 기준으로 지정한 경로에 위치한 자원으로부터 정보를 읽어 온다

SAMPLE03


Spring 설정파일
beans.xml
HelloApp
main(args:String[])
Spring
Framework
<<interface>>
MessageBean
Sayhello(name:String)
MessageBeanKo
Sayhello(name:String)
MessageBeanEn
Sayhello(name:String)

세 개의 예제를 통한 스프링의 기본 기능과 구조 이해하기

(3) 스프링을 이용한 샘플 어플리케이션 분석하기(sample3)

```
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")); */
            ApplicationContext bf =
                        new FileSystemXmlApplicationContext("bean01.xml");
            // MessageBean mb = bf.getBean("message",MessageBean.class);
            // MessageBean mb = (MessageBean)bf.getBean("message");
            MessageBean mb = (MessageBean)bf.getBean("k");
            mb.sayHello("홍길동");
    }
}
```

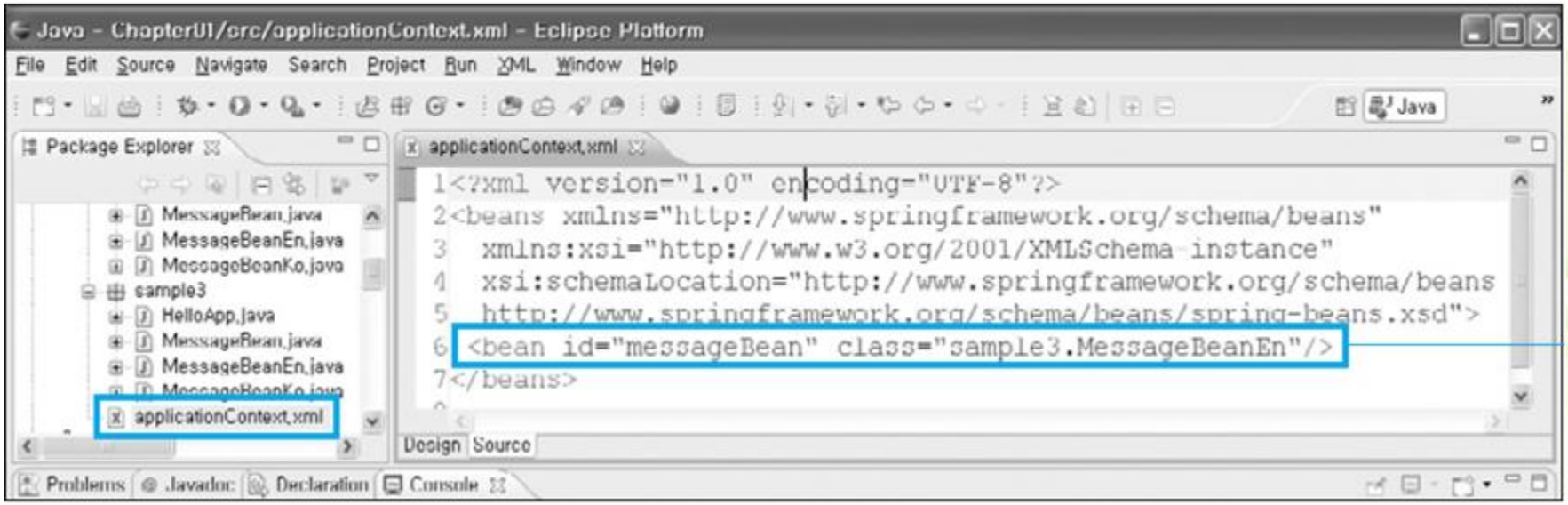
XmlBeanFactory : 빈을 생성하고 설정 및 관리하는 역할

세 개의 예제를 통한 스프링의 기본 기능과 구조 이해하기

(3) 스프링을 이용한 샘플 어플리케이션 분석하기(sample3)

beans.xml

```
<bean id = "messageBean" class = "sample3.MessageBeanEn" />
```



설정 파일 : **beans.xml(**임으로 지정**)**

Id : Bean붙이는 식별자
name : id의 별칭, 복수 가능
class : bean클래스의 완전한 클래스 이름
parent : bean정의를 상속
abstract : bean클래스의 추상클래스 여부 **false**
singleton : bean이 싱글톤으로 리턴하는 여부 **true**
lazy-init : bean의 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 : bean초기화 실행 시킬 메서드
destroy-method : bean컨테이너 종료할 때 실행시킬 메서드

```
<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

3.1 스프링의 장점

필요한 인스턴스를 스프링에서 미리 생성해 준다.

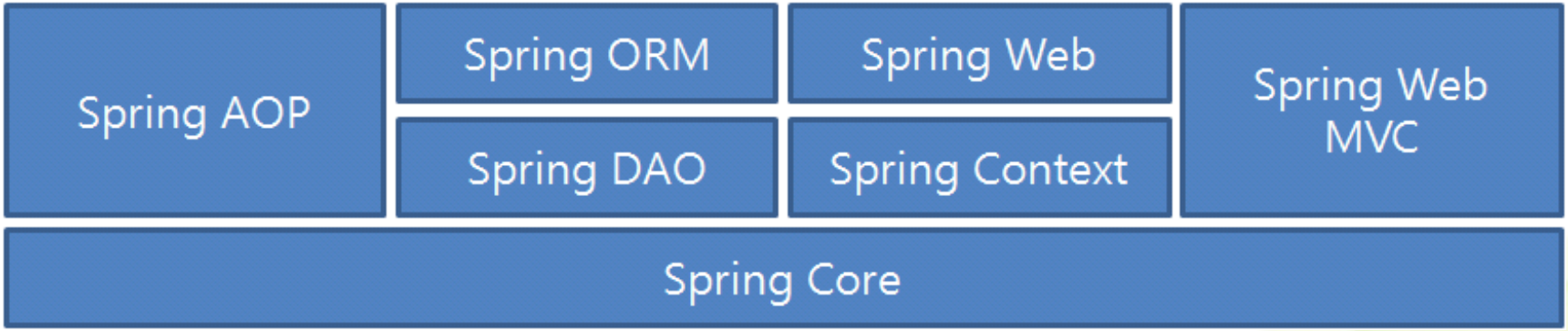
클래스 사이의 결합(loosely coupled)을 느슨하게 할 수 있어 클래스 간의 의존 관계가 약해진다.

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

3.2 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.

3.3 스프링을 구성하는 모듈



의존 관계 주입
(DI : Dependancy Injection)

DI에 대해서

- DI는 「Dependency Injection」의 약어로서 인터페이스를 이용하여 컴포넌트화를 실현하는 것
- 의존성 : 어떤 클래스가 자신의 임무를 하기 위해 필요한 값이나 사용할 다른 클래스와의 관계
- 주입 : 어떤 클래스의 인스턴스에 대해 의존성을 설정 하는 것

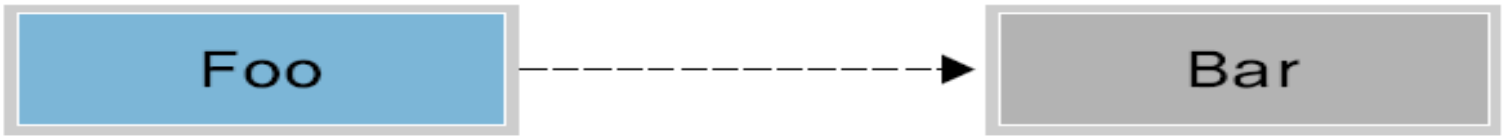
- 따라서 DI는 객체간의 의존 관계를 객체 자신이 아닌 외부 조립기가 수행해 준다는 개념, 스프링에서는 설정파일과 어노테이션을 이용하여 객체간의 의존관계 설정하는 기능 제공

- 의존관계 주입컨테이너는 어떤 클래스가 필요로 하는 값이나 인스턴스를 생성, 취득하고 그 클래스의 인스턴스에 대하여 설정함
- 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

'Constructor Injection' –생성자를 통해서 의존 관계 연결

◈ Foo.java

```
public class Foo{
  private Bar bar;
  public Foo(Bar bar){
    this.bar = bar;
  }
}
```

◈ applicationContext.xml

```
〈bean id = "foo" class = "Foo" 〉
  〈constructor-arg〉
    〈ref bean = "bar" /〉
  〈/constructor-arg〉
〈/bean〉
〈bean id = "bar" class = "Bar" /〉
```

스프링의 설정에서 생성된 Bar 인스턴스를 <bean> 요소의 자식 요소인 <constructor-arg> 요소를 연결

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= "Bean**이름**" />**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 Injection' 이란 클래스 사이의 의존 관계를 연결시키기 위해서 setter 메소드를 이용하는 방법을 말한다.

◈ Foo.java

```
public class Foo{
  private Bar bar;
  public setBar(Bar bar){
    this.bar = bar;
  }
}
```

```
<bean id = "foo" class = "Foo" >
  <property name = "bar" ref = "bar" > </property>  ..... 1
</bean>

<bean id = "bar" class = "Bar" />  ..... 2
```

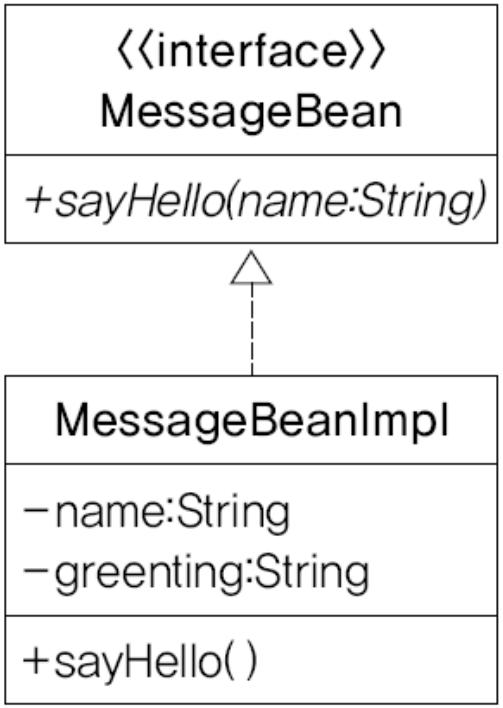
ApplicatonContext인터페이스와 WebApplicationContext 인터페이스

org.springframework.context.ApplicationContext인터페이스는 **BeanFactory** 인터페이스를 상속받은 하우 인터페이스로서 **BeanFactory**가 제공하는 빈 관리기능이 외에 빈 객체 라이프싸이클, 파일과 같은 자원 처리 추상화, 메시지 지원 및 국제화 지원, 이벤트지원, **XML**스키마 확장을 통한 편리한 설정 등 추가적인 기능을 제공하고 있다.

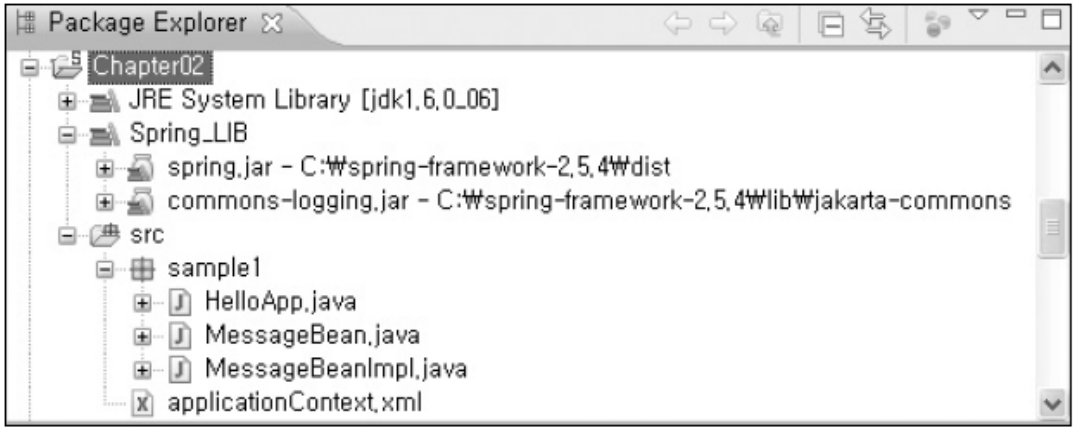
Org.springframework.web.context.WebApplicationConrtext 인터페이스는 웹 어플리케이션에 위한 **ApplicationContext**로서 하나의 웹 어플리케이션 마다 한 개 이상의 **WebApplicationContext**를 가질 수 있다

. **ClassPathXmlApplicationContext** 클래스 패스에 위치한 **XML**파일로부터 설정정보를 로딩한다
. **FileSystemXmlApplicationContext** 파일 시스템에 위치한 **XML**파일로부터 설정정보를 로딩한다
. **XmlWebApplicationContext** 웹 어플리케이션에 위치한 **XML** 파일로부터 설정정보를 로딩한다

DI 패턴 이해를 위한 예제



▲ 클래스 그림



▲ 파일 구성

```
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();
}
```

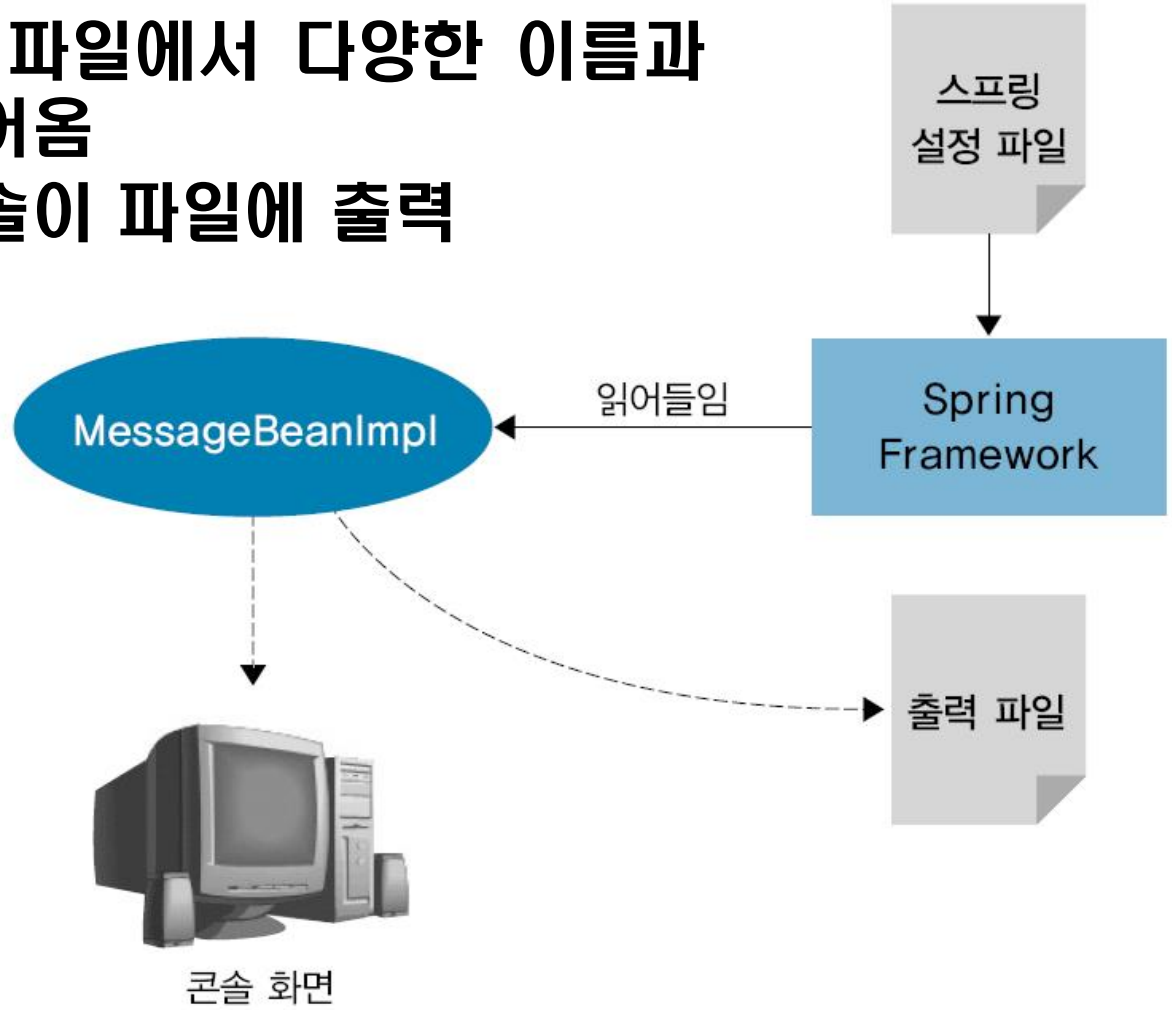
```
<?xml version="1.0" encoding="UTF-8" ?>

<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="sample.MessageBeanImpl" >
      <constructor-arg>
         <value>Spring</value>
      </constructor-arg>

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

</beans>
```

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

```
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;
  }
}
```

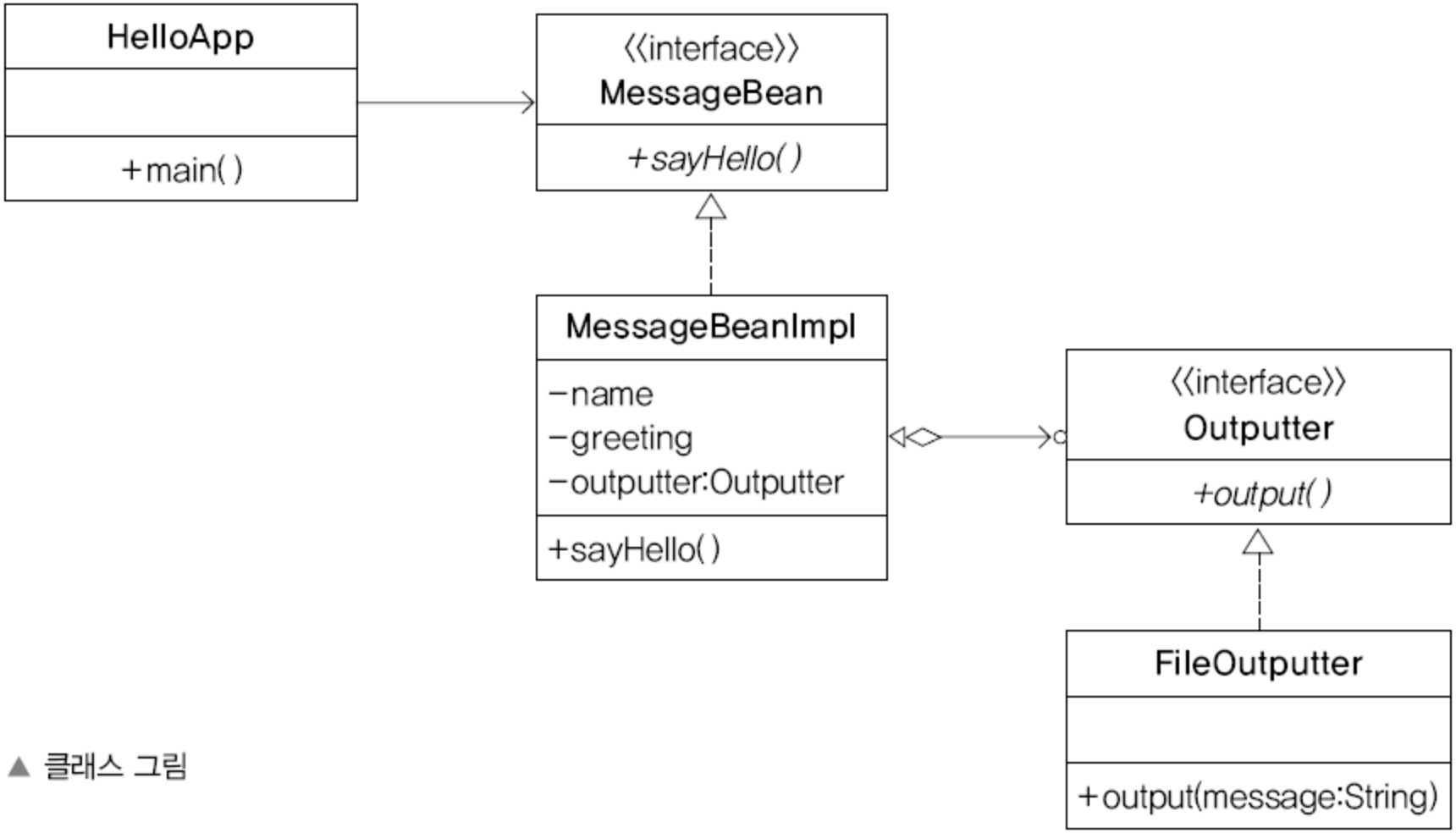
DI 패턴 이해를 위한 예제

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

```
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();
}
```

```
<?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>
```

```
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;
    }
}
```

```
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;
    }
}
```

설정 파일의 핵심

다른 빈을 참조하는 경우 **<ref>**요소를 이용하고 **local** 속성에 참조하는 **bean**이름을 지정합니다

```
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 HelloApp {

   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();
   }
}
```

```
package sample1;
public interface MessageBean {
        void sayHello();
}

package sample1;
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();
        }
    }
}
```

```
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;
    }
}
```

```
<?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>
```

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/schema/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);
    }
}
```

```
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();
        }
}
```

```
package sample;
import org.springframework.beans.factory.BeanFactory;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import sample.di.business.domain.Product;
import sample.di.business.service.ProductService;

public class ProductSampleRun {
    public static void main(String[] args) {
        ProductSampleRun productSampleRun = new ProductSampleRun();
        productSampleRun.execute();
    }
    public void execute() {
        BeanFactory ctx = new ClassPathXmlApplicationContext(
                "/sample/config/applicationContext.xml");
        ProductService productService = ctx.getBean(ProductService.class);
        Product product = productService.getProduct();
        System.out.println(product);
    }
}
```

어노테이션을 사용한 DI

- 스프링은 크게 Bean정의 파일을 사용한 DI와 어노티션이용한 것이 있다. 앞의 것이 Bean을 이용한 것임

ProductSampleRun.java

```
package an;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import an.di.business.domain.Product;
import an.di.business.service.ProductService;
public class ProductSampleRun {
    public static void main(String[] args) {
        ProductSampleRun productSampleRun = new ProductSampleRun();
        productSampleRun.execute();
    }
    public void execute() {
        ApplicationContext ctx = new ClassPathXmlApplicationContext(
                "/an/config/applicationContext.xml");
        ProductService productService = ctx.getBean(ProductService.class);
        Product product = productService.getProduct();
        System.out.println(product);
    }
}
```

```
package an.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 + "]";
    }
}
```

1. 어노테이션 설명

DI컨테이너는 **@Autowired**가 붙은 인스턴스 변수에 대입할 수 있는 클래스를 **@Component**가 붙은 클래스에서 찾아내 그 인스턴스를 인젝션해준다
따라서 **setter**메소드를 준비할 필요가 없다
@Componet가 붙은 클래스가 여러 개 있어도 형이 다르면 **@Autowired**다 붙은 인스턴스변수에 인젝션하지 않는다. 이렇게 하는 방법을 **byType**라고 한다

2. 주요스키마

. bean 스키마 : **Bean(**컴포넌트**)**설정
. conext : Bean검색과 어노테이션 설정
. jee : JNDI의 **lookup** 및 **EJB**의 **lookup**설정
. util : 정의와 프로퍼티 파일을 불러오는 등의 유틸리티 기능 설정
. lang : 스크립트 언어를 이용할 경우의 설정
. aop : AOP설정
. tx : 트랜잭션 설정
. mvc : Spring mvc설정

3. 태그

<context:annotation-config />@Autowired, @Resource를 이용할 때 선언
<context:component-scan base-package=” 패키지이름,…**” />**
@Component, @Service등의 컴포넌트를 이용할 때 선언

4. 인테베이스에 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. @Component확장

@Service : 서비스용 어노테이션으로 트랜젝션관리
아직 사용하지 않음

@Repository : 데이터 억세스층의 **DAO**어너테이션

ProductDao.java

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

ProductDaoImpl.java

```
package an.di.dataaccess;
import org.springframework.stereotype.Component;
import an.di.business.domain.Product;
import an.di.business.service.ProductDao;
@Component
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 an.di.business.service;
import an.di.business.domain.Product;
public interface ProductService {
   Product getProduct();
}
```

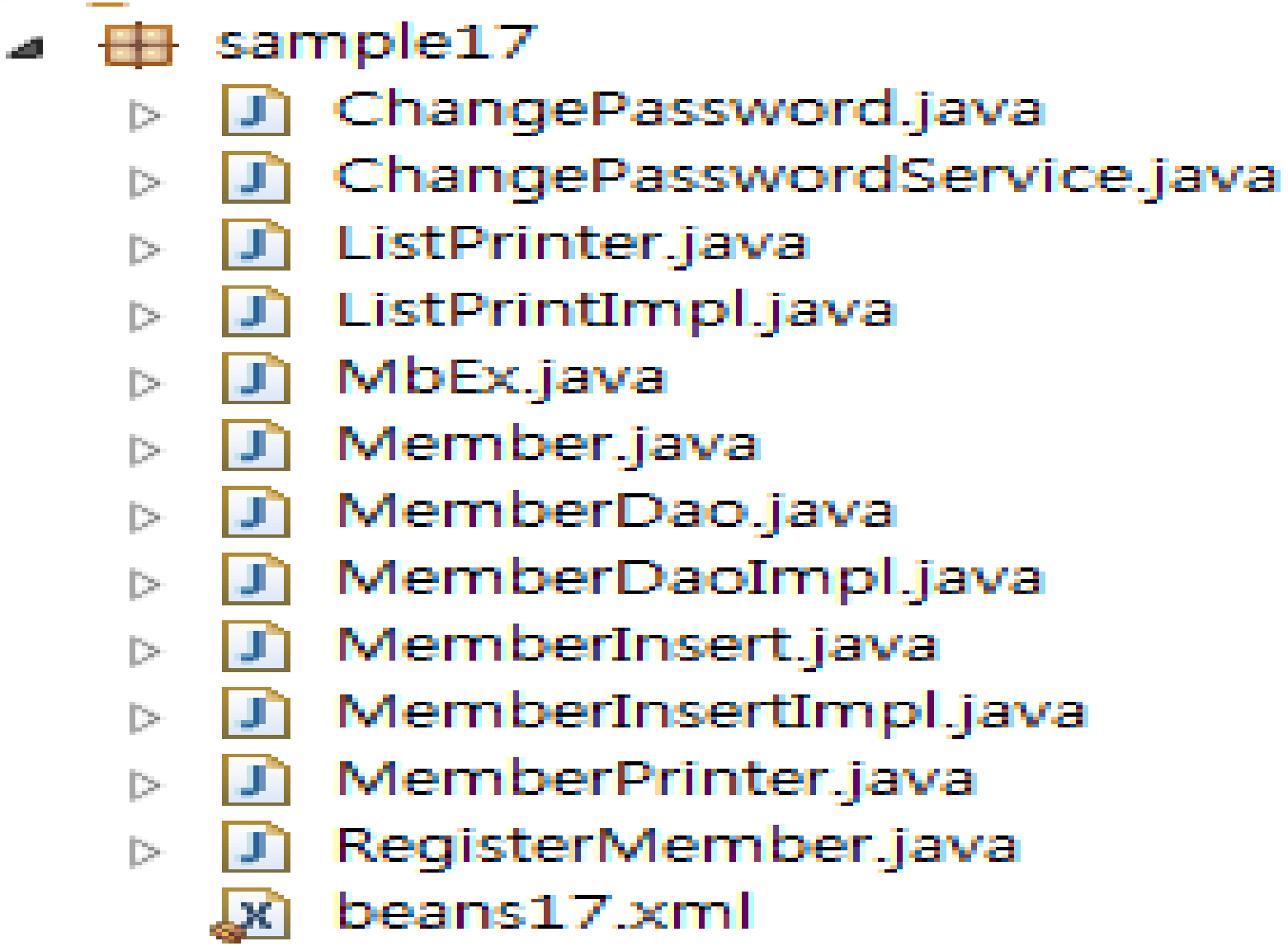
ProductServiceImpl.java

```
package an.di.business.service;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import an.di.business.domain.Product;
@Component
public class ProductServiceImpl implements ProductService {
    @Autowired
    private ProductDao productDao;
    public Product getProduct() {
        return productDao.findProduct();
    }
}
```

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-3.1.xsd
     http://www.springframework.org/schema/context
     http://www.springframework.org/schema/context/spring-context-3.1.xsd">
  <context:annotation-config />
  <context:component-scan base-package="an.di.business.*" />
  <context:component-scan base-package="an.di.dataaccess" />
</beans>
```

회원가입 입력프로젝트


sample17
ChangePassword.java
ChangePasswordService.java
ListPrinter.java
ListPrintImpl.java
MbEx.java
Member.java
MemberDao.java
MemberDaoImpl.java
MemberInsert.java
MemberInsertImpl.java
MemberPrinter.java
RegisterMember.java
beans17.xml

Member.java

```
package spring1;
import java.util.Date;
public class Member {
private Long id;                    private String email;  private String password;
private String name;                private Date registerDate;
    public Member(String email, String password, String name,
                        Date registerDate) {
            this.email = email;             this.password = password;
            this.name = name;               this.registerDate = registerDate;
    }
    void setId(Long id) {  this.id = id;  }
    public Long getId() {   return id;   }
    public String getEmail() {return email;}
    public String getPassword() {   return password;  }
    public String getName() {   return name;  }
    public Date getRegisterDate() {   return registerDate;   }
    public void changePassword(String oldPassword, String newPassword) {
          if (!password.equals(oldPassword))
                throw new IdPasswordNotMatchingException();
          this.password = newPassword;
    }
}
```

```
package sample17;
import java.util.Collection;
public interface MemberDao {
    Member selectByEmail(String email);
    int insert(Member member);
    void update(Member member);
    Collection<Member> selectAll();
}

package sample17;

public interface ChangePassword {
    public int  changePassword(String email,
            String oldPwd, String newPwd);
}

package sample17;
public interface MemberInsert {
     int insert(RegisterMember req);
}
```

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();
    }
}
```

```
package sample17;
public class ChangePasswordService implements ChangePassword{
    private MemberDao memberDao;

    public ChangePasswordService(MemberDao memberDao) {
        this.memberDao = memberDao;
    }

    public int  changePassword(String email, String oldPwd, String newPwd) {
        int result = 0;
        Member member = memberDao.selectByEmail(email);
        if (member == null) {
            System.out.println("없는 멤버입니다");
        } else {
            result = member.changePassword(oldPwd, newPwd);
            memberDao.update(member);
        }
        return result;
     }
}
```

RegisterMember.java

```
package spring1;
public class RegisterMember {
    private String email;
    private String password;
    private String confirmPassword;
    private String name;

    public String getEmail() {  return email;  }
    public void setEmail(String email) {  this.email = email;   }
    public String getPassword() {   return password;   }
    public void setPassword(String password) {  this.password = password;   }
    public String getConfirmPassword() {   return confirmPassword;   }
    public void setConfirmPassword(String confirmPassword) {
        this.confirmPassword = confirmPassword;
    }
    public String getName() {   return name;  }
    public void setName(String name) {   this.name = name;  }

    public boolean isPasswordEqualToConfirmPassword() {
        return password.equals(confirmPassword);
   }
}
```

MemberInsertImpl.java

```
package spring1;
import java.util.Date;
public class MemberInsertImpl implements MemberInsert{
    private MemberDao md;
    public void setMd(MemberDao md) {
        this.md = md;
   }
   public int insert(RegisterMember req) {
      int result = 0;
      Member member = md.selectByEmail(req.getEmail());
      if (member!=null) {
          System.out.println("이미 있는 이메일입니다");
          return result;
       }  else {
         Member nMem = new Member(req.getEmail(),
         req.getPassword(),req.getName(),new Date());
         result = md.insert(nMem);
         return result;
      }
   }
}
```

ChangePasswordService.java

```
package sample17;
public class ChangePasswordService implements ChangePassword{
    private MemberDao memberDao;

    public ChangePasswordService(MemberDao memberDao) {
          this.memberDao = memberDao;
    }
    public int  changePassword(String email, String oldPwd, String newPwd) {
          int result = 0;
          Member member = memberDao.selectByEmail(email);
          if (member == null) {
               System.out.println("없는 멤버입니다");
          }    else {
               result = member.changePassword(oldPwd, newPwd);
               memberDao.update(member);
          }
          return result;
    }
}
```

MemberPrinter.java

```
package sample17;
public class MemberPrinter {
    public void print(Member member) {
        System.out.printf("회원정보 이름 : %s, "+
          "이메일 : %s, ID : %d, 등록일 : %TF\n",
          member.getName(),member.getEmail(),
          member.getId(),member.getRegisterDate());
     }
}
```

ListPrinter.java

```
package sample17;
public interface ListPrinter {
    void printAll();
}
```

ListPrintImpl.java

```
package sample17;
import java.util.*;
public class ListPrintImpl implements ListPrinter {
    private MemberDao md;
    private MemberPrinter mp;

    public void setMd(MemberDao md) {this.md = md;}
    public void setMp(MemberPrinter mp) {this.mp = mp;}

    public void printAll() {
       Collection<Member> list = md.selectAll();
       for (Member mem : list) {
          mp.print(mem);
     }
   }
}
```

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();
}
```

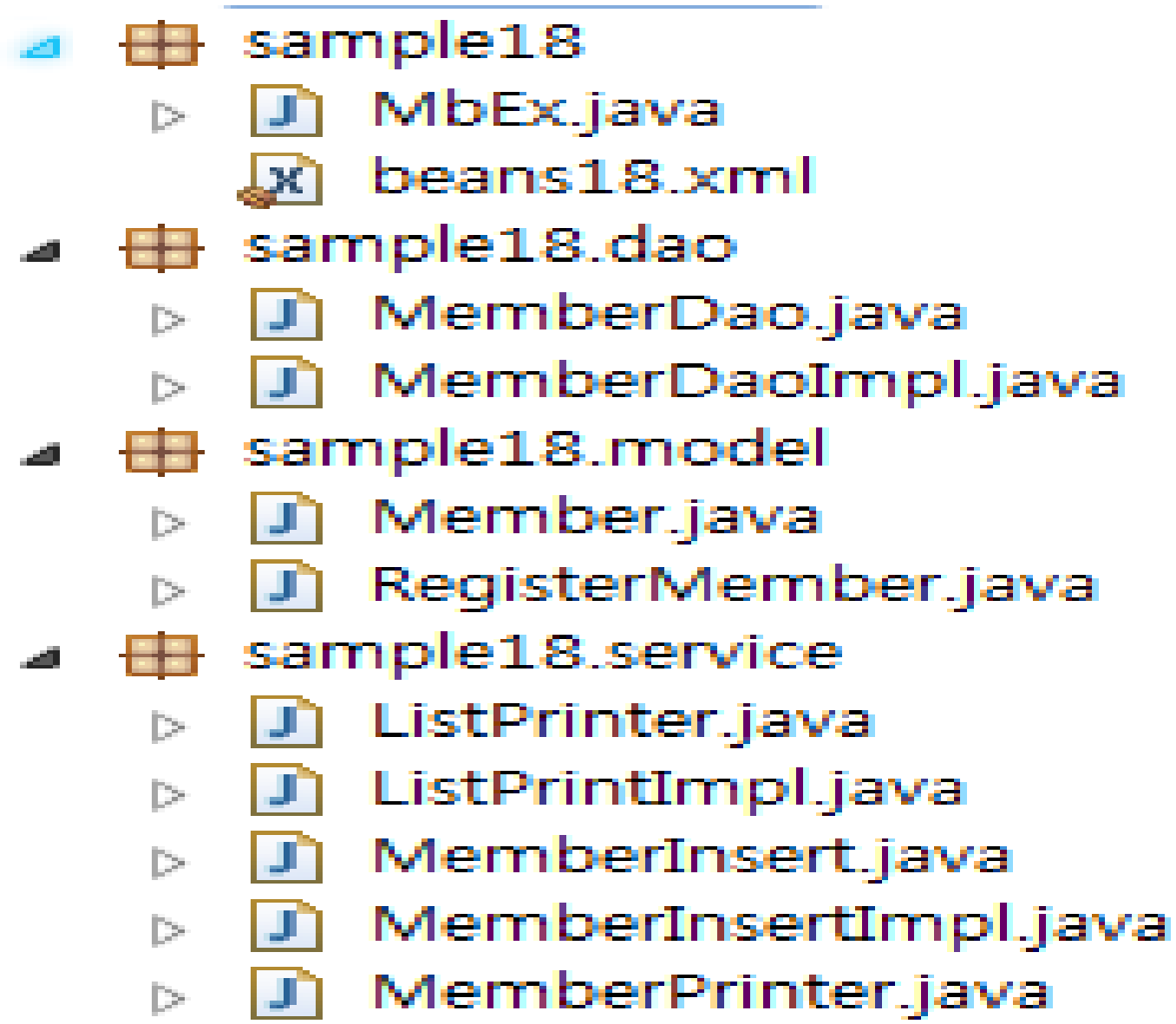
```
static void processInsert(String[] str) {
      if (str.length !=5) {
            help();   return;
       }
      RegisterMember req = new RegisterMember();
      req.setEmail(str[1]);
      req.setName(str[2]);
      req.setPassword(str[3]);
      req.setConfirmPassword(str[4]);
      if (!req.passConfirm()) {
            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>
```

회원가입 입력프로젝트(spring DI이용 : 생성자 방식)



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>
```

JavaConfig.java (자바코드를 이용한 설정)

```
package spring5;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
@Configurationpublic
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");
    }
}
```