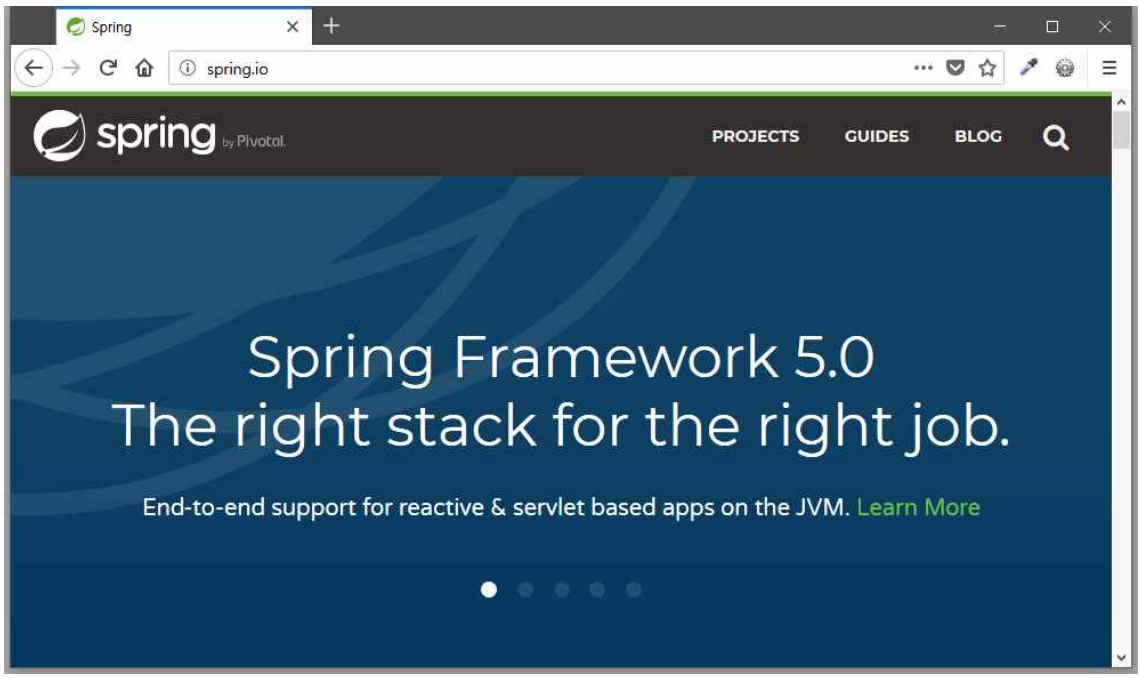
■ Spring 프레임워크



1) Application Framework : 거대한 애플리케이션 개발 프레임워크

2) DI(IoC) Framework : 제어역전(의존성 주입) 프레임워크

3) AOP Framework : 관점지향 프레임워크

4) Data Access Framework : 데이터 연결 프레임워크

5) Transaction management Framework : 트랜젝션 관리

6) MVC Framework : MVC 패턴(MODEL2) 적용 프레임워크

7) 그 외 다양한 기능

■ 스프링 프레임워크의 탄생배경

1) 웹사이트가 점점 커지면서 엔터프라이즈급의 서비스가 필요하게 됨

2) 자바진영에서는 EJB가 엔터프라이즈급 서비스로 각광을 받게 됨

  - 세션빈에서 Transaction 관리가 용이함

  - 로깅, 분산처리, 보안등

3) 하지만 EJB는 개발시 여러 가지 제약이 존재함(배보다 배꼽이 더 큼)

- EJB스펙에 정의된 인터페이스에 따라 코드를 작성하므로 기존에 작성된 POJO를 변경해야 함
- 컨테이너에 배포를 해야만 테스트가 가능해 개발속도가 저하됨
- 배우기 어렵고, 설정해야 할 부분이 많음

4) Rod Johnson이 'Expert One-on-One J2EE Development without EJB' 라는 저서에서 EJB를 사용하지 않고 엔터프라이즈 어플리케이션을 개발하는 방법을 소개함(스프링의 모태가 됨)
- AOP나 DI같은 새로운 프로그래밍 방법론으로 가능
- POJO로 선언적인 프로그래밍 모델이 가능해 짐

POJO란? 1. Plain Old Java Object의 약자임 2. Plain의 뜻 : component interface를 상속받지 않는 특징(특정 프레임워크에 종속되지 않는) 3. Old : EJB 이전의 자바 클래스를 의미함
Java Bean이란? 원래 비주얼 툴(UI)에서 조작가능한 컴포넌트를 말했으나 웹 기반 방식으로 바뀌면서 자바빈은 다음 두가지 관례를 따라 만들어진 오브젝트를 말함 1. Default 생성자 2. 프로퍼티(멤버필드)와 setter / getter

2. 스프링 프레임워크의 취지

■ 기술 자체보다 좋은 설계가 중요함

■ 인터페이스를 통한 느슨한 결합은 자바 빈의 훌륭한 모델임

■ 코드는 테스트하기 쉽고 편해야 함

■ EJB의 복잡도에서 벗어나야 함

느슨한 결합이란? 두 객체가 느슨하게 결합되어 있다는 것이란, 그 둘이 상호작용을 하긴 하지만, 서로에 대해 서로 잘 모른다는 뜻이다. 느슨하게 결합되는 디자인을 사용하면 변경사항이 생겨도 무난히 처리할 수 있는 유연한 객체지향 시스템을 구축할 수 있다.

3. 스프링 프레임워크의 목표

■ 엔터프라이즈 서비스를 쉽게 구축

■ 의존성 주입(Dependency Injection)을 통한 유연한 프레임워크 구현

■ 관점지향 프로그래밍(Aspect oriented Programming) 지원

■ Application의 완전한 이식성 제공

■ 반복적인 코드의 제거

■ 생산성 향상

4. 스프링의 특징

■ 스프링은 자바객체를 담고 있는 경량의 컨테이너다. 이들 자바객체의 생성 소멸과 같은 라이프사이클을 관리한다.(여기서 경량이라고 말하는 것은 무조건 용량이 작다는 뜻이 아니라 불필요한 복잡함을 제거했다는 뜻이다.)

■ 오픈소스 프레임워크이다.

■ 스프링은 의존성 주입(Dependency Injection)을 지원한다.

■ 스프링은 관점지향 프로그래밍(Aspect Oriented Programming)을 지원한다.

■ 스프링은 POJO(Plain Old Java Object)를 지원한다.

■ 스프링은 트랜잭션 처리를 위한 일관된 방법을 제공하며, 설정파일을 통해 트랜잭션 관련정보를 선언적으로 정의할 수 있다.

■ 스프링은 다양한 ORM(Object-Relation Mapping)툴과의 연동을 지원한다.

■ 스프링은 Enterprise Application개발에 필요한 다양한 API를 지원한다.

> 컨테이너란?
> 컨테이너란 객체를 생성하고 소멸하는 등의 라이프사이클을 관리하는 것을 의미한다.

■ POJO(Plain Old Java Object)

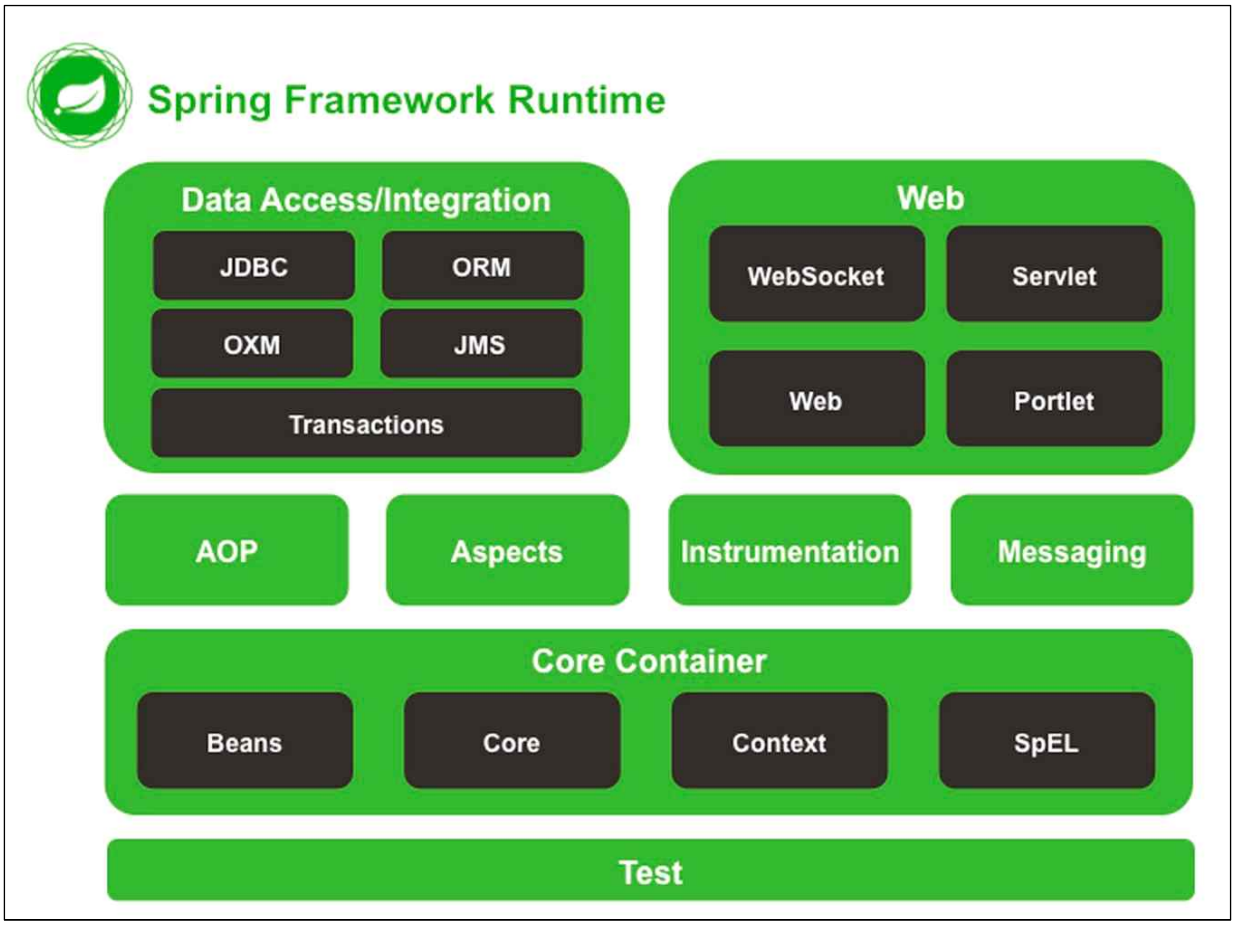
1) 프레임워크가 요구하는 특정규약에 종속적이지 않음

(우리가 만든 MVC는 Action을 무조건 구현해야 했음)

2) 특정 환경에 종속되지 않음

(우리가 만든 MVC는 HttpServlet, HttpSession등 웹환경에 종속)

5. 스프링 모듈 설명



■ Spring Core

- 스프링의 가장 기본적인 기능 수행(DI)
- 모든 스프링 애플리케이션 기반인 Bean Factory를 포함하고 있음
- Bean Factory는 DI를 통해 설정과 의존성을 실제 코드와 분리하는 팩토리패턴 구현객체임

■ Spring Context

- 국제화메시지, 애플리케이션 생명주기, 이벤트, 유효성 검증등을 지원함
- 이메일, jndi접근, ejb 연계, 스케쥴링 등의 엔터프라이즈 서비스를 지원

■ Spring AOP

- 관점지향 프로그래밍 지향
- 메타데이터의 지원을 통해 Aspect를 적용할지 알려주는 annotation을 추가할 수 있음
- 사용하거나 배우기가 어려움

■ Spring ORM

- jdbc 상위에서 ORM 프레임워크와 연동가능하게 함
- 연동된 ORM 프레임워크의 트랜잭션도 관리가 가능

■ Spring DAO

- jdbc로 작업할 때 연결취득, 명령, 실행, 결과집합처리 연결 끊기등의 반복적인 코드를 추상화함
- 데이터베이스 접근 코드를 간결하게 함
- 트랜잭션 서비스를 제공
- 다른 ORM보다 코딩하기 불편하여 거의 사용하지 않음

■ Spring Web

- 웹 기반 어플리케이션에 적합한 Context를 제공함

- 파일업로드 등의 multipart 요청처리나 요청파라미터를 빈즈에 바인딩하는 등의 웹 관련 작업을 지원

■ Spring Web MVC

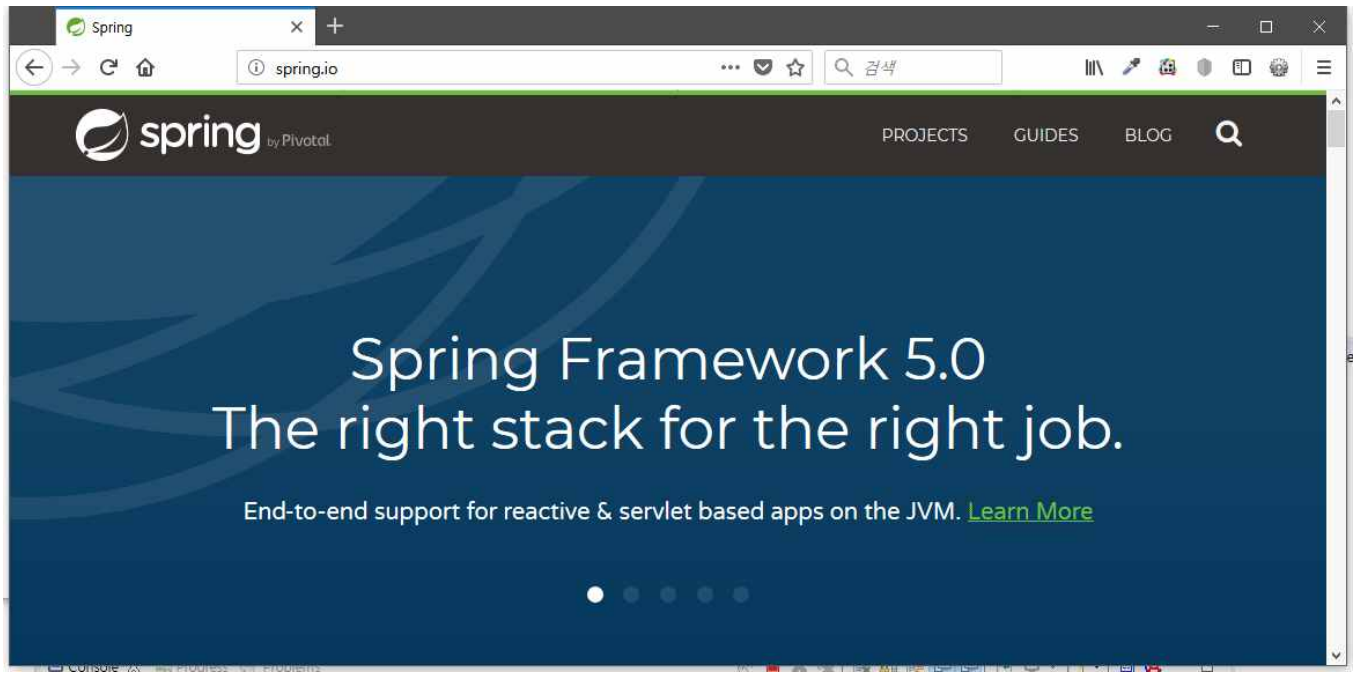
- 웹 어플리케이션 구축을 위한 모델-뷰-컨트롤러 프레임워크 제공
- 사용시 요청 파라미터를 선언적인 방법으로 비즈니스 객체에 바인딩 가능

■ 그 외 JMX, JCA, JMS, portlet MVC, remoting 등

> 팩토리패턴이란?
> 객체를 생성하기 위한 인터페이스를 정의하는데, 어떤 클래스 인스턴스를 만들지는 서브클래스에서 결정하게 만든다. 구성 요소 별로 '객체의 집합'을 생성해야 할 때 유용하다. 이 패턴을 사용하여 상황에 알맞은 객체를 생성할 수 있다.

6. 스프링 프레임워크 구현을 위한 환경설정

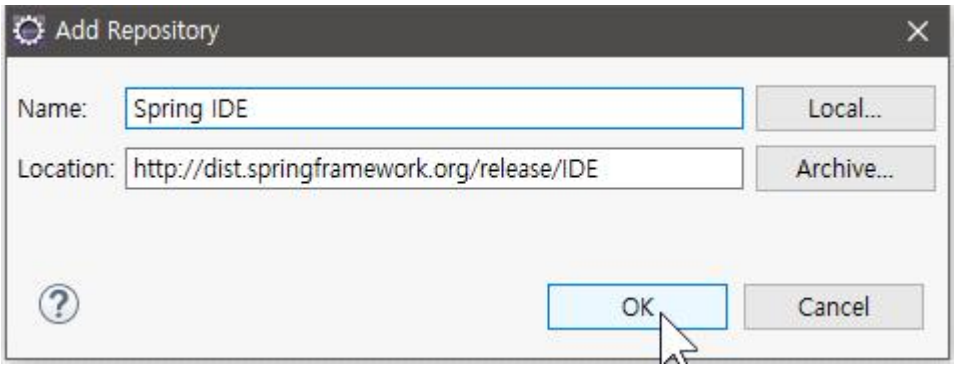
■ spring.io

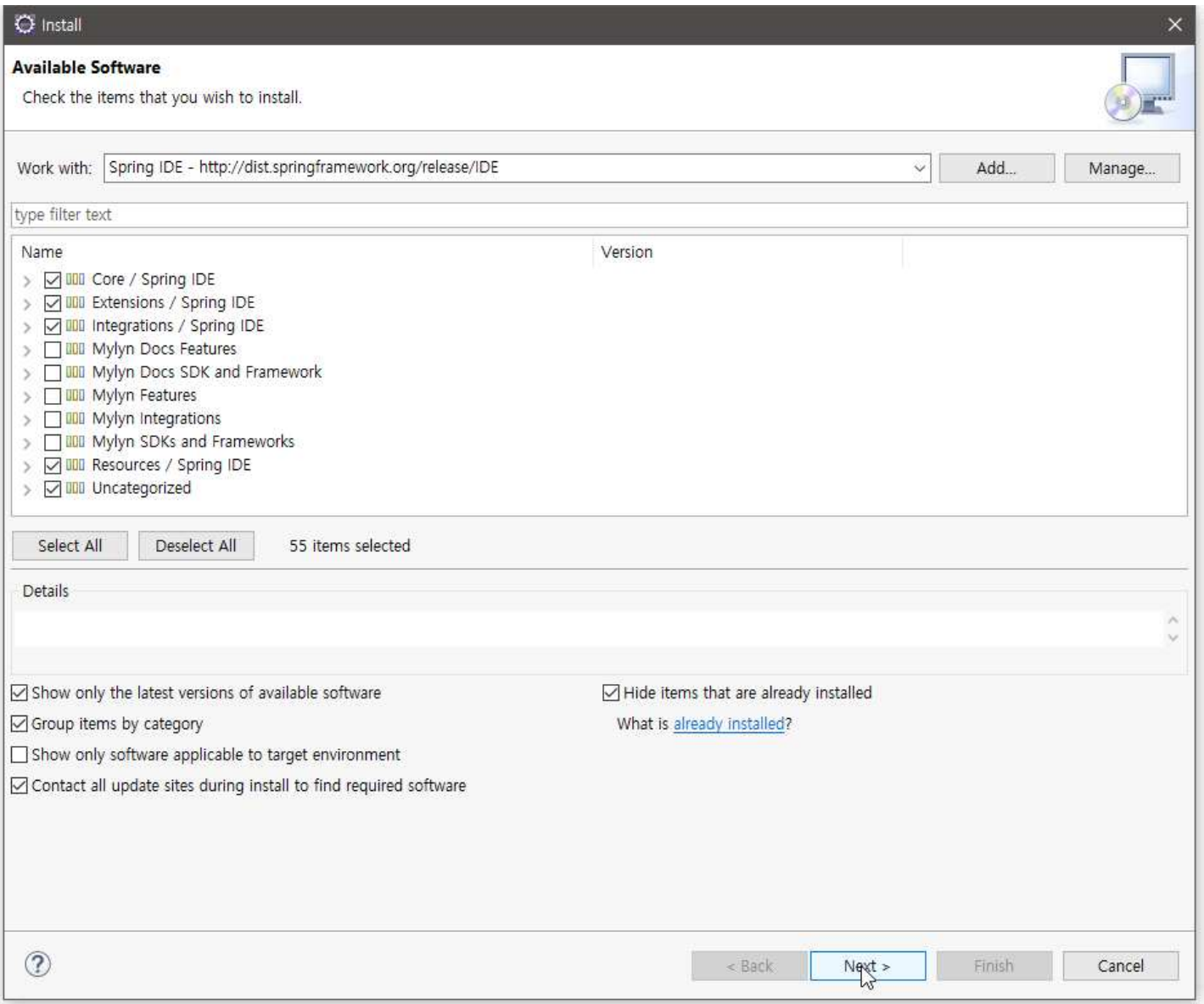


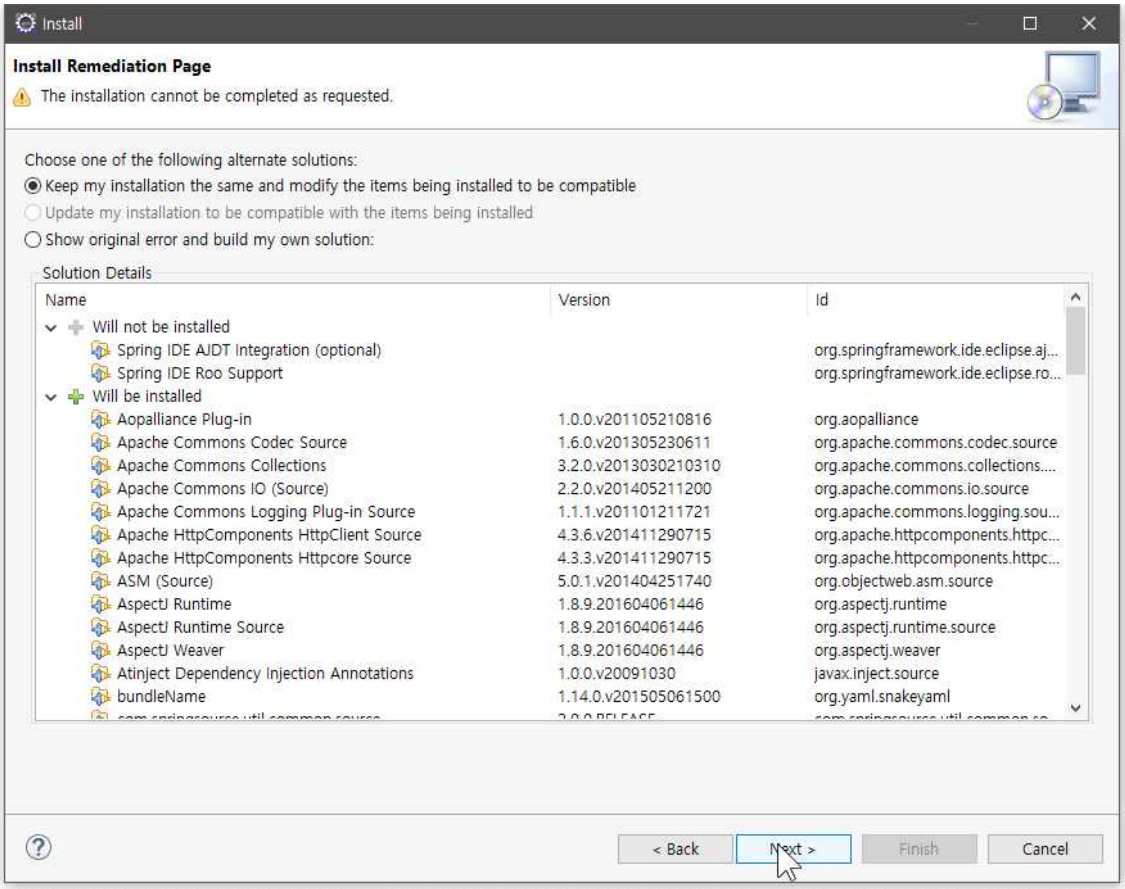
- download가 없음

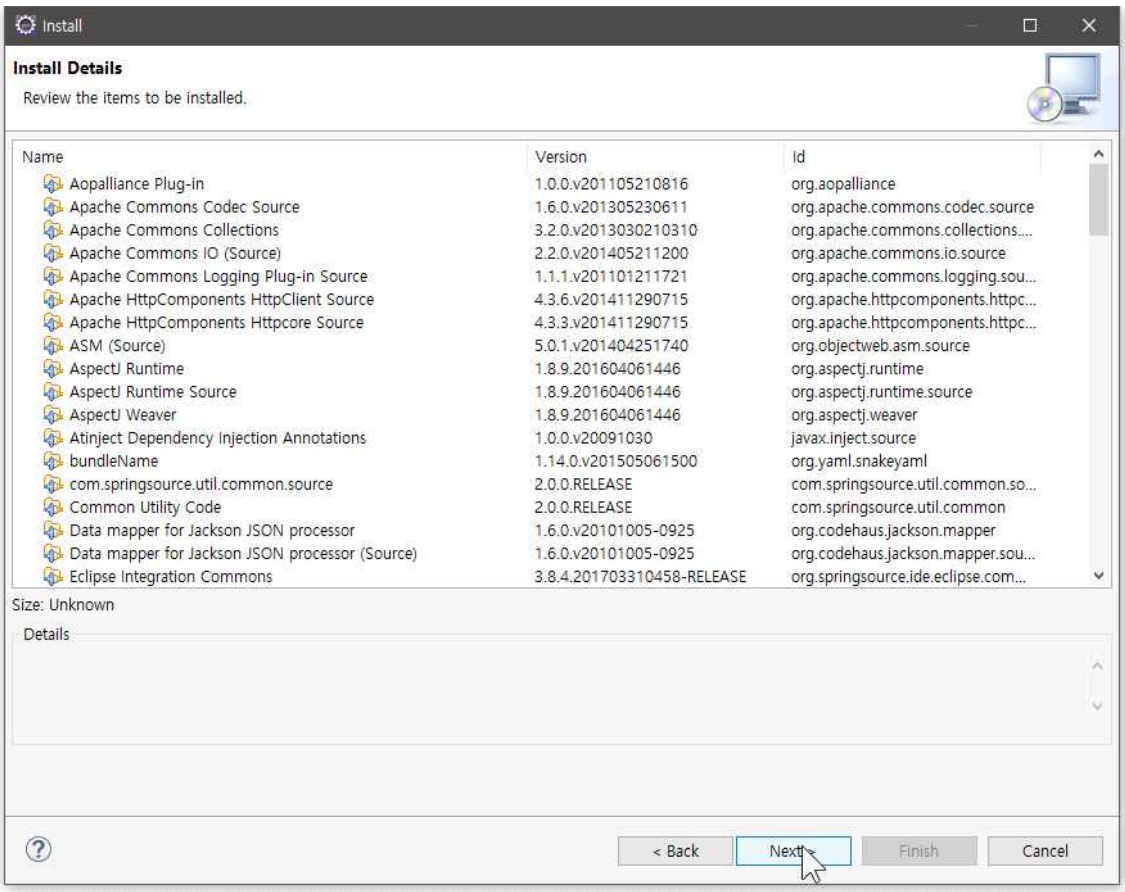
■ Spring IDE 플러그인 설치

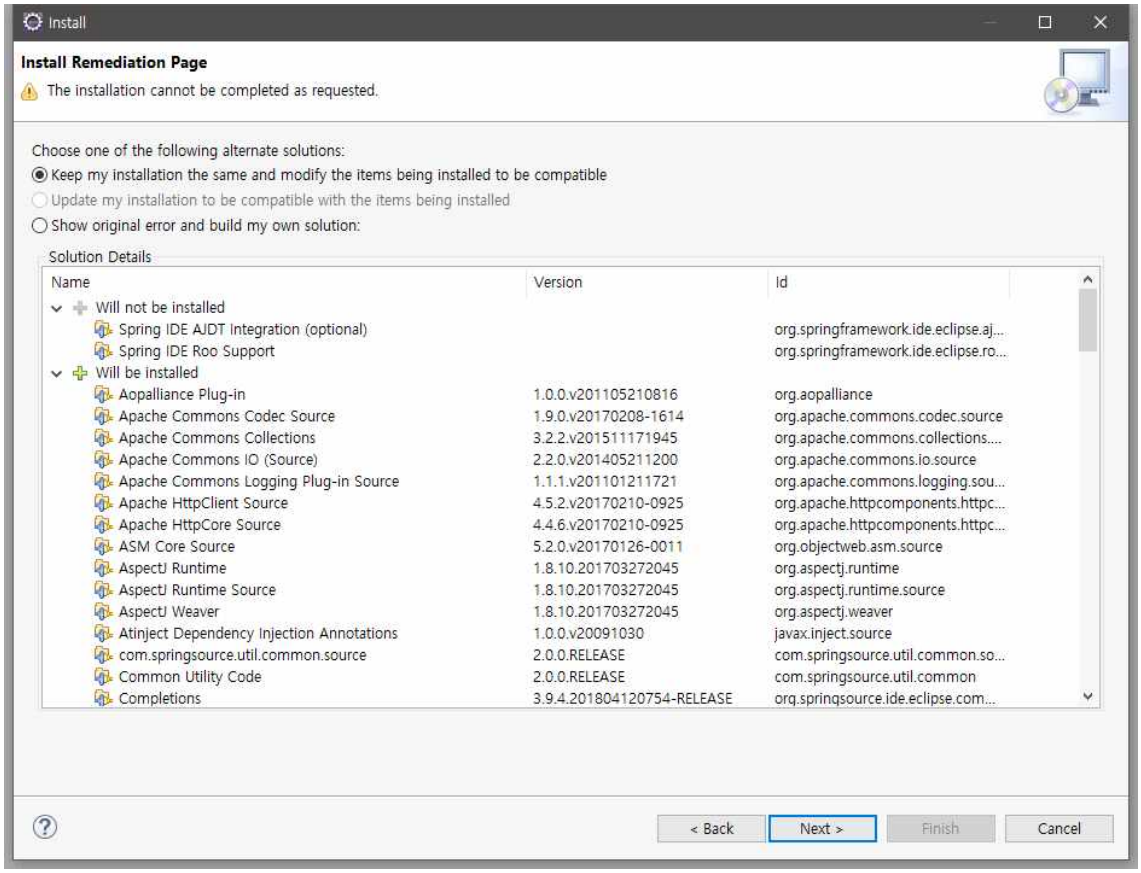
1) http://dist.springframework.org/release/IDE

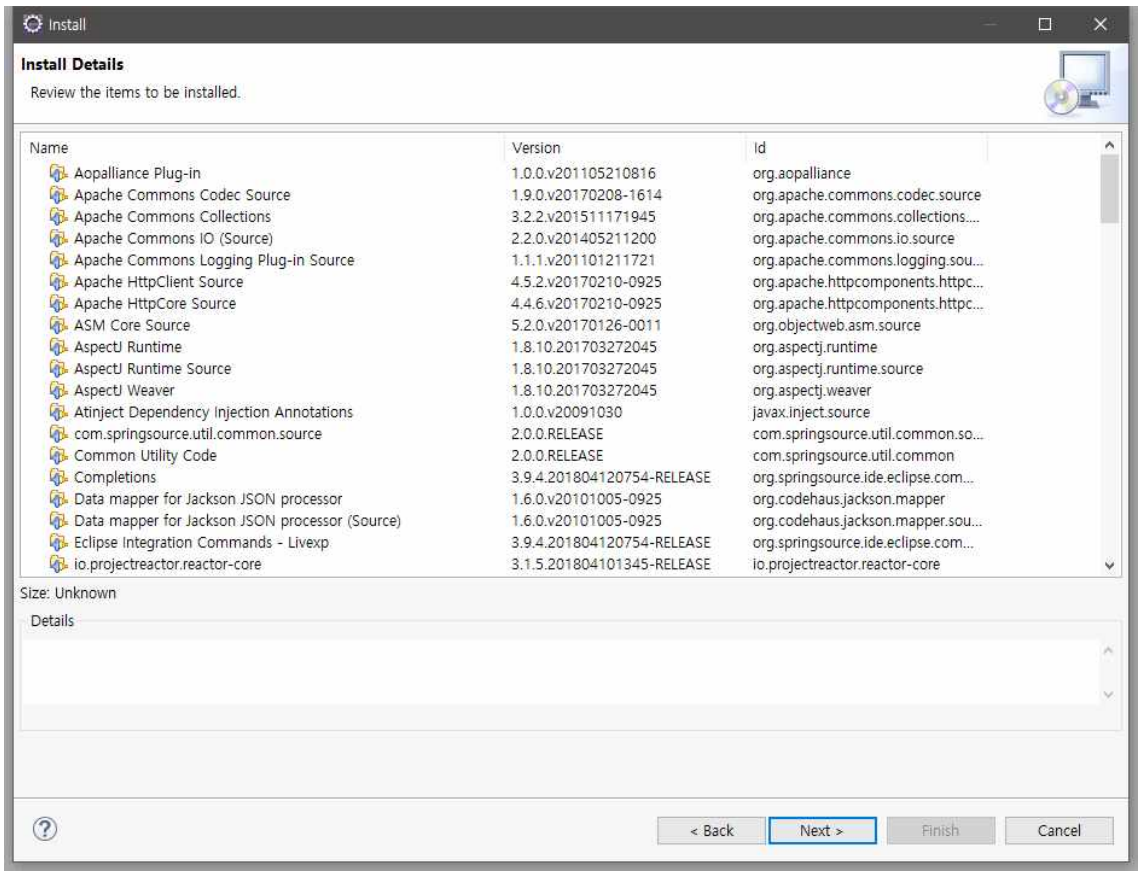


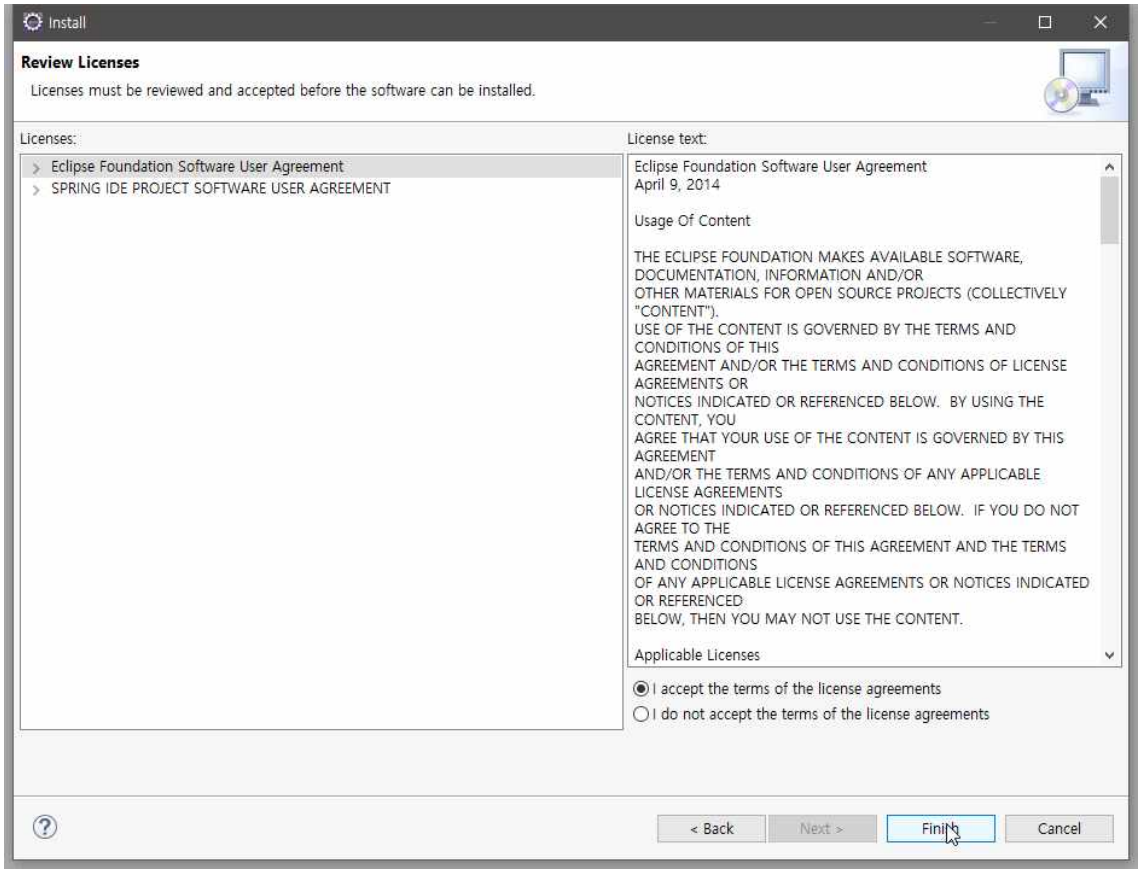



Install
Install Remediation Page
The installation cannot be completed as requested.
Choose one of the following alternate solutions:
Keep my installation the same and modify the items being installed to be compatible
Update my installation to be compatible with the items being installed
Show original error and build my own solution:
Solution Details
Name
Version
Id
Will not be installed
Spring IDE AJDT Integration (optional)
org.springframework.ide.eclipse.aj...
Spring IDE Roo Support
org.springframework.ide.eclipse.ro...
Will be installed
Aopalliance Plug-in
1.0.0.v201105210816
org.aopalliance
Apache Commons Codec Source
1.6.0.v201305230611
org.apache.commons.codec.source
Apache Commons Collections
3.2.0.v2013030210310
org.apache.commons.collections....
Apache Commons IO (Source)
2.2.0.v201405211200
org.apache.commons.io.source
Apache Commons Logging Plug-in Source
1.1.1.v201101211721
org.apache.commons.logging.sou...
Apache HttpComponents HttpClient Source
4.3.6.v201411290715
org.apache.httpcomponents.httpc...
Apache HttpComponents Httpcore Source
4.3.3.v201411290715
org.apache.httpcomponents.httpc...
ASM (Source)
5.0.1.v201404251740
org.objectweb.asm.source
AspectJ Runtime
1.8.9.201604061446
org.aspectj.runtime
AspectJ Runtime Source
1.8.9.201604061446
org.aspectj.runtime.source
AspectJ Weaver
1.8.9.201604061446
org.aspectj.weaver
Atinject Dependency Injection Annotations
1.0.0.v20091030
javax.inject.source
bundleName
1.14.0.v201505061500
org.yaml.snakeyaml
< Back
Next >
Finish
Cancel


Install
Install Details
Review the items to be installed.
Name
Version
Id
Aopalliance Plug-in
1.0.0.v201105210816
org.aopalliance
Apache Commons Codec Source
1.6.0.v201305230611
org.apache.commons.codec.source
Apache Commons Collections
3.2.0.v2013030210310
org.apache.commons.collections....
Apache Commons IO (Source)
2.2.0.v201405211200
org.apache.commons.io.source
Apache Commons Logging Plug-in Source
1.1.1.v201101211721
org.apache.commons.logging.sou...
Apache HttpComponents HttpClient Source
4.3.6.v201411290715
org.apache.httpcomponents.httpc...
Apache HttpComponents Httpcore Source
4.3.3.v201411290715
org.apache.httpcomponents.httpc...
ASM (Source)
5.0.1.v201404251740
org.objectweb.asm.source
AspectJ Runtime
1.8.9.201604061446
org.aspectj.runtime
AspectJ Runtime Source
1.8.9.201604061446
org.aspectj.runtime.source
AspectJ Weaver
1.8.9.201604061446
org.aspectj.weaver
Atinject Dependency Injection Annotations
1.0.0.v20091030
javax.inject.source
bundleName
1.14.0.v201505061500
org.yaml.snakeyaml
com.springsource.util.common.source
2.0.0.RELEASE
com.springsource.util.common.so...
Common Utility Code
2.0.0.RELEASE
com.springsource.util.common
Data mapper for Jackson JSON processor
1.6.0.v20101005-0925
org.codehaus.jackson.mapper
Data mapper for Jackson JSON processor (Source)
1.6.0.v20101005-0925
org.codehaus.jackson.mapper.sou...
Eclipse Integration Commons
3.8.4.201703310458-RELEASE
org.springsource.ide.eclipse.com...
Size: Unknown
Details
< Back
Next >
Finish
Cancel


Install
Install Remediation Page
The installation cannot be completed as requested.
Choose one of the following alternate solutions:
Keep my installation the same and modify the items being installed to be compatible
Update my installation to be compatible with the items being installed
Show original error and build my own solution:
Solution Details
Name
Version
Id
Will not be installed
Spring IDE AJDT Integration (optional)
org.springframework.ide.eclipse.aj...
Spring IDE Roo Support
org.springframework.ide.eclipse.ro...
Will be installed
Aopalliance Plug-in
1.0.0.v201105210816
org.aopalliance
Apache Commons Codec Source
1.9.0.v20170208-1614
org.apache.commons.codec.source
Apache Commons Collections
3.2.2.v201511171945
org.apache.commons.collections....
Apache Commons IO (Source)
2.2.0.v201405211200
org.apache.commons.io.source
Apache Commons Logging Plug-in Source
1.1.1.v201101211721
org.apache.commons.logging.sou...
Apache HttpClient Source
4.5.2.v20170210-0925
org.apache.httpcomponents.httpc...
Apache HttpCore Source
4.4.6.v20170210-0925
org.apache.httpcomponents.httpc...
ASM Core Source
5.2.0.v20170126-0011
org.objectweb.asm.source
AspectJ Runtime
1.8.10.201703272045
org.aspectj.runtime
AspectJ Runtime Source
1.8.10.201703272045
org.aspectj.runtime.source
AspectJ Weaver
1.8.10.201703272045
org.aspectj.weaver
Atinject Dependency Injection Annotations
1.0.0.v20091030
javax.inject.source
com.springsource.util.common.source
2.0.0.RELEASE
com.springsource.util.common.so...
Common Utility Code
2.0.0.RELEASE
com.springsource.util.common
Completions
3.9.4.201804120754-RELEASE
org.springsource.ide.eclipse.com...
< Back
Next >
Finish
Cancel


Install
Install Details
Review the items to be installed.
Name
Version
Id
Aopalliance Plug-in
1.0.0.v201105210816
org.aopalliance
Apache Commons Codec Source
1.9.0.v20170208-1614
org.apache.commons.codec.source
Apache Commons Collections
3.2.2.v201511171945
org.apache.commons.collections....
Apache Commons IO (Source)
2.2.0.v201405211200
org.apache.commons.io.source
Apache Commons Logging Plug-in Source
1.1.1.v201101211721
org.apache.commons.logging.sou...
Apache HttpClient Source
4.5.2.v20170210-0925
org.apache.httpcomponents.httpc...
Apache HttpCore Source
4.4.6.v20170210-0925
org.apache.httpcomponents.httpc...
ASM Core Source
5.2.0.v20170126-0011
org.objectweb.asm.source
AspectJ Runtime
1.8.10.201703272045
org.aspectj.runtime
AspectJ Runtime Source
1.8.10.201703272045
org.aspectj.runtime.source
AspectJ Weaver
1.8.10.201703272045
org.aspectj.weaver
Atinject Dependency Injection Annotations
1.0.0.v20091030
javax.inject.source
com.springsource.util.common.source
2.0.0.RELEASE
com.springsource.util.common.so...
Common Utility Code
2.0.0.RELEASE
com.springsource.util.common
Completions
3.9.4.201804120754-RELEASE
org.springsource.ide.eclipse.com...
Data mapper for Jackson JSON processor
1.6.0.v20101005-0925
org.codehaus.jackson.mapper
Data mapper for Jackson JSON processor (Source)
1.6.0.v20101005-0925
org.codehaus.jackson.mapper.sou...
Eclipse Integration Commands - Livexp
3.9.4.201804120754-RELEASE
org.springsource.ide.eclipse.com...
io.projectreactor.reactor-core
3.1.5.201804101345-RELEASE
io.projectreactor.reactor-core
Size: Unknown
Details
< Back
Next >
Finish
Cancel



■ Users 테이블

Column Name	Data Type	Nullable	Default	Primary Key
ID	VARCHAR2(20)	No	-	1
NAME	VARCHAR2(30)	No	-	-
PASSWORD	VARCHAR2(20)	No	-	-
				1 - 3

■ User VO

```
package user.domain;

public class User {

	private String id,name,password;

	public String getId() {
		return id;
	}

	public void setId(String id) {
		this.id = id;
```

```
        }

        public String getName() {
                return name;
        }

        public void setName(String name) {
                this.name = name;
        }

        public String getPassword() {
                return password;
        }

        public void setPassword(String password) {
                this.password = password;
        }

}
```

■ UserDAO

```
package user.dao;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import user.domain.User;

public class UserDAO {

        public void add(User user) throws ClassNotFoundException,SQLException {
                Class.forName("oracle.jdbc.OracleDriver");

                Connection c =

DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","test","1111");

                PreparedStatement ps =
                                c.prepareStatement("INSERT       INTO       users(id,name,password)
VALUES(?,?,?)");

                ps.setString(1,user.getId());
                ps.setString(2,user.getName());
```

```
                ps.setString(3,user.getPassword());

                ps.executeUpdate();

                ps.close();
                c.close();

        }

        public User get(String id) throws ClassNotFoundException,SQLException {
                Class.forName("oracle.jdbc.OracleDriver");

                Connection c =

DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","test","1111");

                PreparedStatement ps = c.prepareStatement("SELECT * FROM users WHERE id = ?");

                ps.setString(1,id);

                ResultSet rs = ps.executeQuery();

                rs.next();

                User user = new User();

                user.setId(rs.getString(1));
                user.setName(rs.getString(2));
                user.setPassword(rs.getString(3));

                rs.close();
                ps.close();
                c.close();

                return user;

        }
}
```

■ Test코드

```
package user.test;

import java.sql.SQLException;
import user.dao.UserDAO;
import user.domain.User;
```

```
public class TestApp {

	public static void main(String[] args) throws ClassNotFoundException,SQLException {
		UserDAO dao = new UserDAO();

		User user = new User();
		user.setId("kimpilgu");
		user.setName("김필구");
		user.setPassword("1234");

		dao.add(user);

		System.out.println("등록 성공");

		User user2 = dao.get(user.getId());

		System.out.println(user2.getId());
		System.out.println(user2.getName());

		System.out.println("조회 성공");
	}
}
```

■ 리펙토링

1) 기존의 코드를 외부의 동작방식에는 변화 없이 내부 구조를 변경하여 재구성하는 작업 또는 기술을 말함

2) 리펙토링을 하면 코드 내부의 설계가 개선되어 코드를 이해하기가 더 편해지고, 변화에 효율적으로 대응할 수 있음

■ 관심사의 분리(Separation of Concerns)

> 관심이 같은 것끼리 하나의 객체 안에, 또는 친한 객체로 모이게 하고, 관심이 다른 것은 가능한 따로 떨어져서 서로 영향을 주지 않도록 분리하는 것

- MyBatis 프레임워크도 관심사의 분리로 SQL구문만 따로 모아두는 mapper XML

이 존재함

■ 높은 응집도

1) 변화가 일어날 때 해당 모듈에서 변화하는 부분이 큰 것

2) Connection 얻어오는 부분이 변경되면 모든 DAO에 가서 변경해야 함

3) 하지만 인터페이스로 만들 경우 수정하는 코드가 적음

■ 낮은 결합도

1) 하나의 오브젝트의 변경이 일어날 때의 관계를 맺고 있는 다른 오브젝트에게 변화를 요구하는 정도

2) 결합도가 낮아야 유지보수가 편리함

■ 리펙토링의 시작

1) 문제점 : 커넥션 얻는 코드의 중복

해결책 : 메서드로의 분리

```
private Connection getConnection() throws ClassNotFoundException,
                        SQLException {

                Class.forName("oracle.jdbc.OracleDriver");
                String url = "jdbc:oracle:thin:@localhost:1521:xe";
                return DriverManager.getConnection(url, "test", "1111");

        }

        public void add(User user) throws ClassNotFoundException, SQLException {

                Connection c = getConnection();

중략...
```

2) 또다른 문제점 : 각 DAO안에 getConnection() 메서드 중복

해결책 : 새로운 클래스 생성

```
package user.dao2;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class ConnectionMaker {

	public Connection getConnection() throws ClassNotFoundException,
			SQLException {

		Class.forName("oracle.jdbc.OracleDriver");
		String url = "jdbc:oracle:thin:@localhost:1521:xe";
		return DriverManager.getConnection(url, "test", "1111");
	}
}
```

```
	public void add(User user) throws ClassNotFoundException, SQLException {

		Connection c = new ConnectionMaker().getConnection();

중략...
```

3) Connection이 변경되었을때 수정이 불가능

해결책 : 인터페이스로 선언후 구현

```
package user.dao2;

import java.sql.Connection;
import java.sql.SQLException;

public interface ConnectionMaker {

	public Connection getConnection() throws ClassNotFoundException,
			SQLException ;

}
```

```
package user.dao2;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class OracleConnectionMaker implements ConnectionMaker {
```

```
        public Connection getConnection() throws ClassNotFoundException,
                        SQLException {
                Class.forName("oracle.jdbc.OracleDriver");
                String url = "jdbc:oracle:thin:@localhost:1521:xe";
                return DriverManager.getConnection(url, "test", "1111");
        }
}
```

```
        private ConnectionMaker maker;

        public UserDAO() {
                maker = new OracleConnectionMaker();
        }

        public void add(User user) throws ClassNotFoundException, SQLException {

                Connection c = maker.getConnection();
```

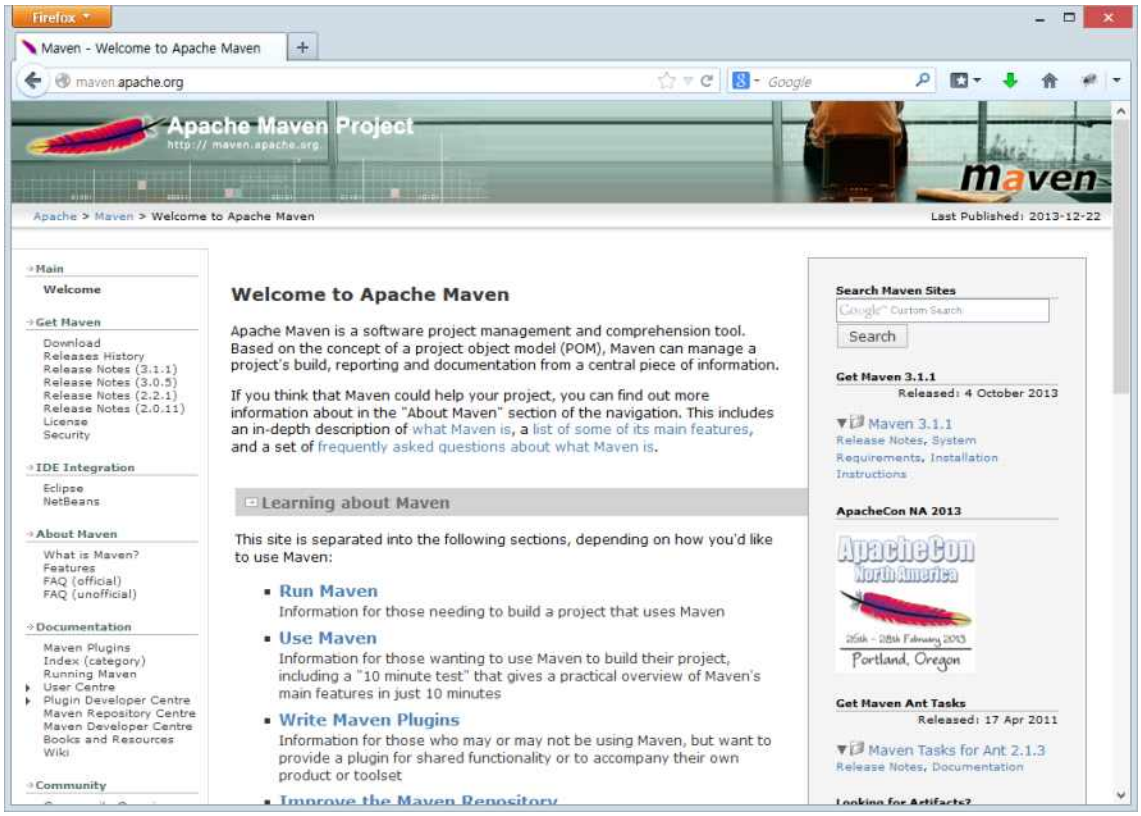
4) UserDAO와 OracleConnectionMaker는 여전히 결합되어 있음

해결책 : 외부에서 ConnectionMaker를 주입해줌

```
        private ConnectionMaker maker;

        public UserDAO(ConnectionMaker maker) {
                this.maker = maker;
        }
```

```
       public static void main(String[] args) throws ClassNotFoundException,SQLException {
                UserDAO dao = new UserDAO(new OracleConnectionMaker());

                User user = new User();
                user.setId("kimpilgu");
                user.setName("김필구");
                user.setPassword("1234");

                dao.add(user);
```

■ Maven 사용



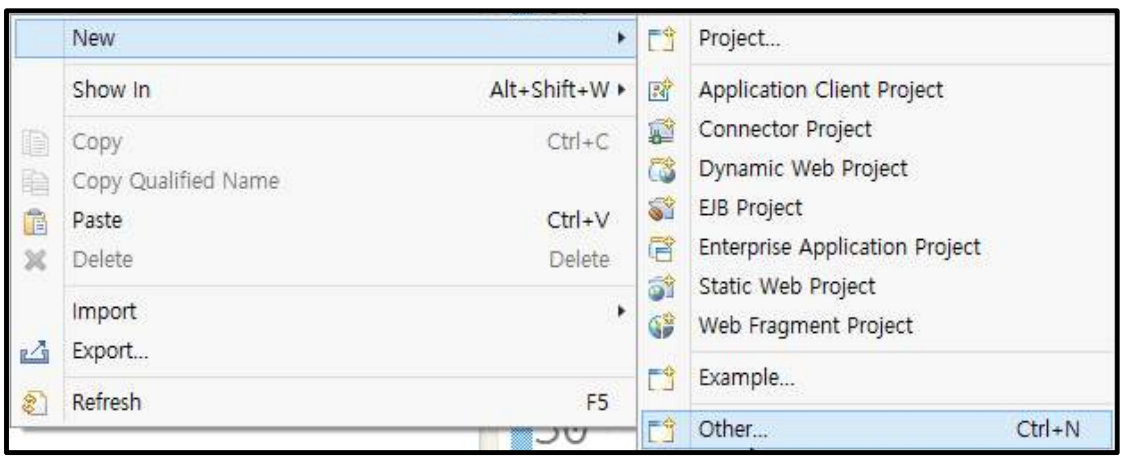
1) 빌드 / 문서화 / 리포팅

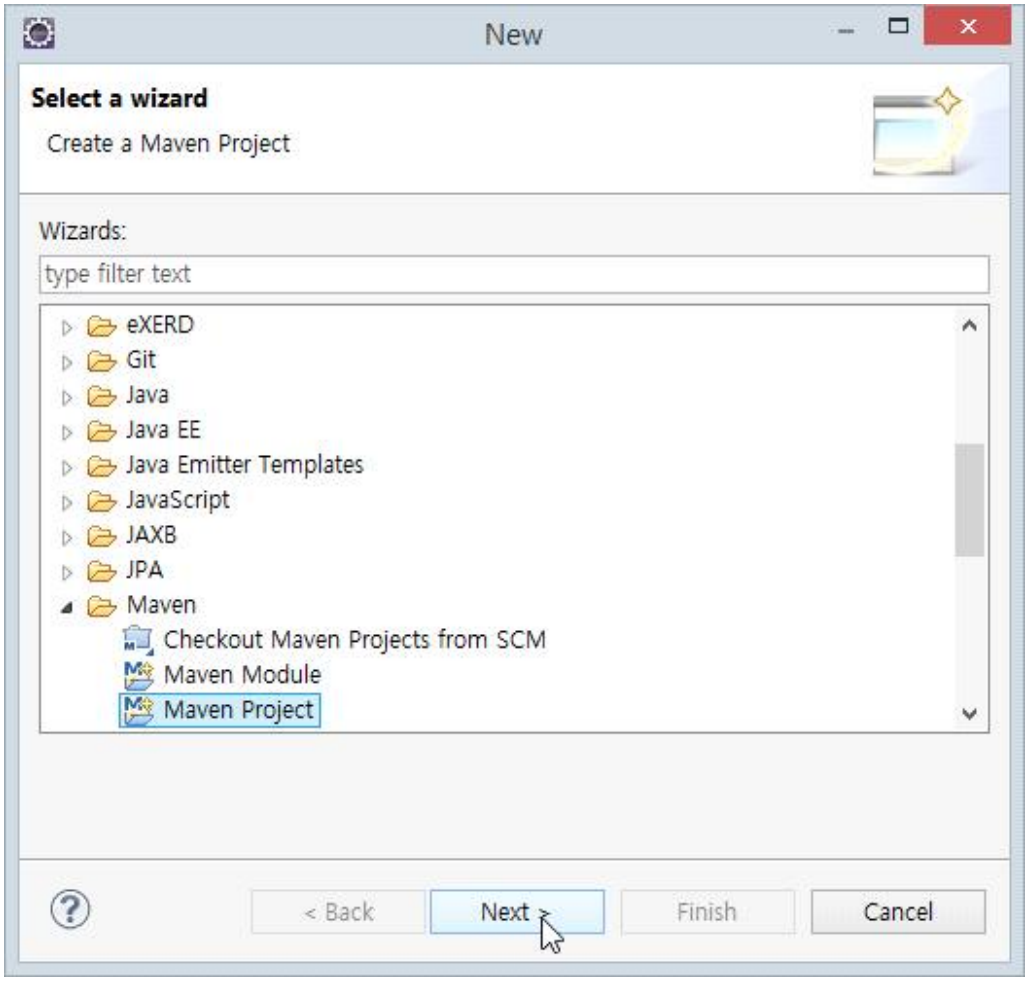
2) 의존관계

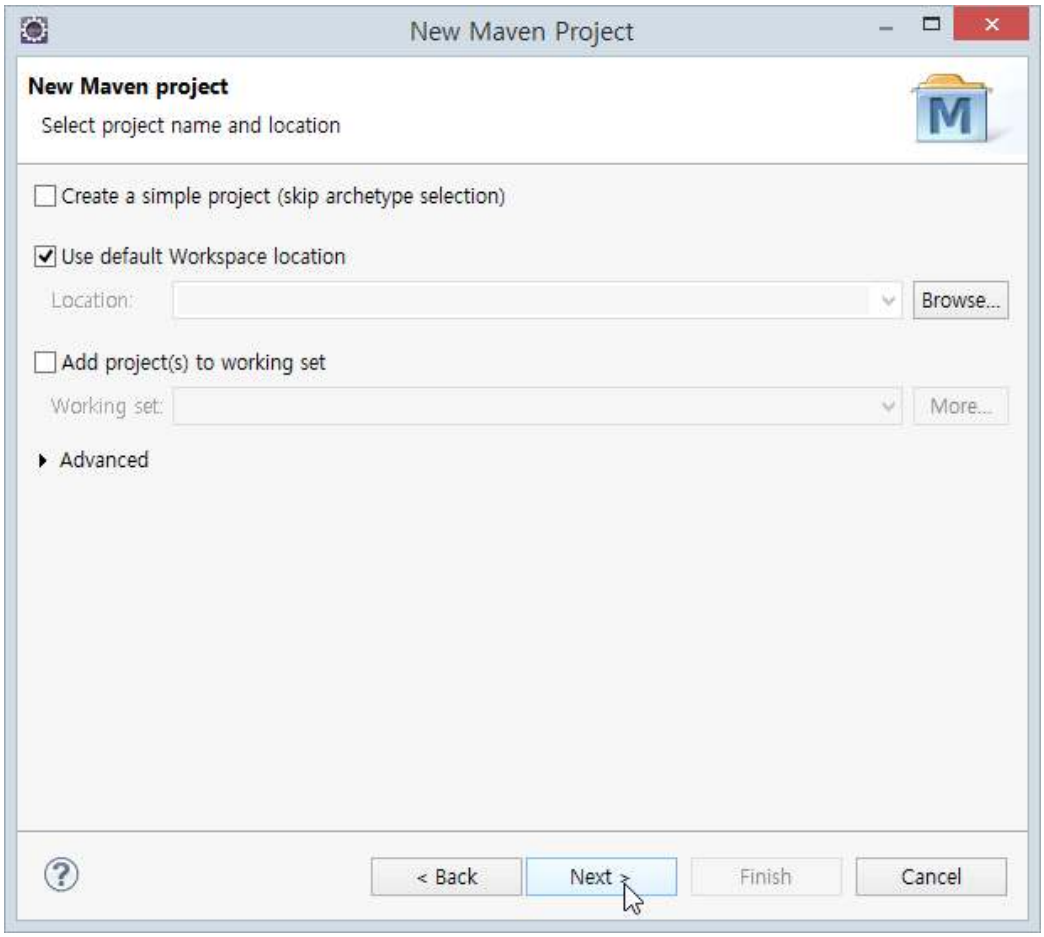
3) 소스코드 관리 / 배포

4) 이클립스 기능에 추가되어 있음

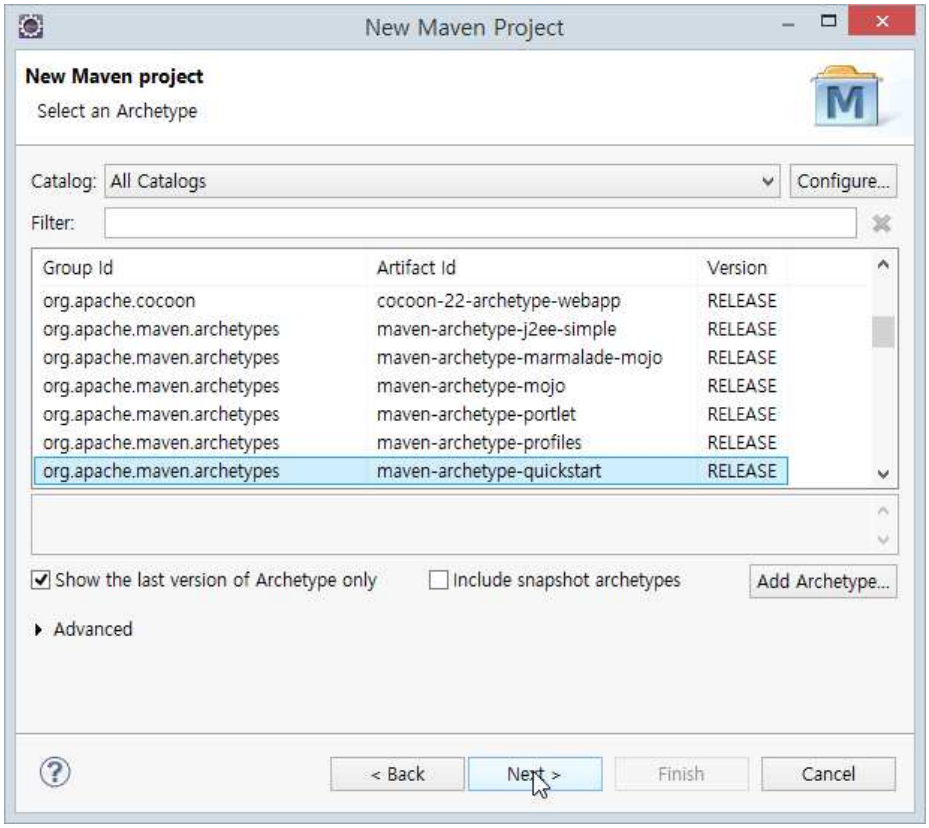
■ Maven 프로젝트 만들기



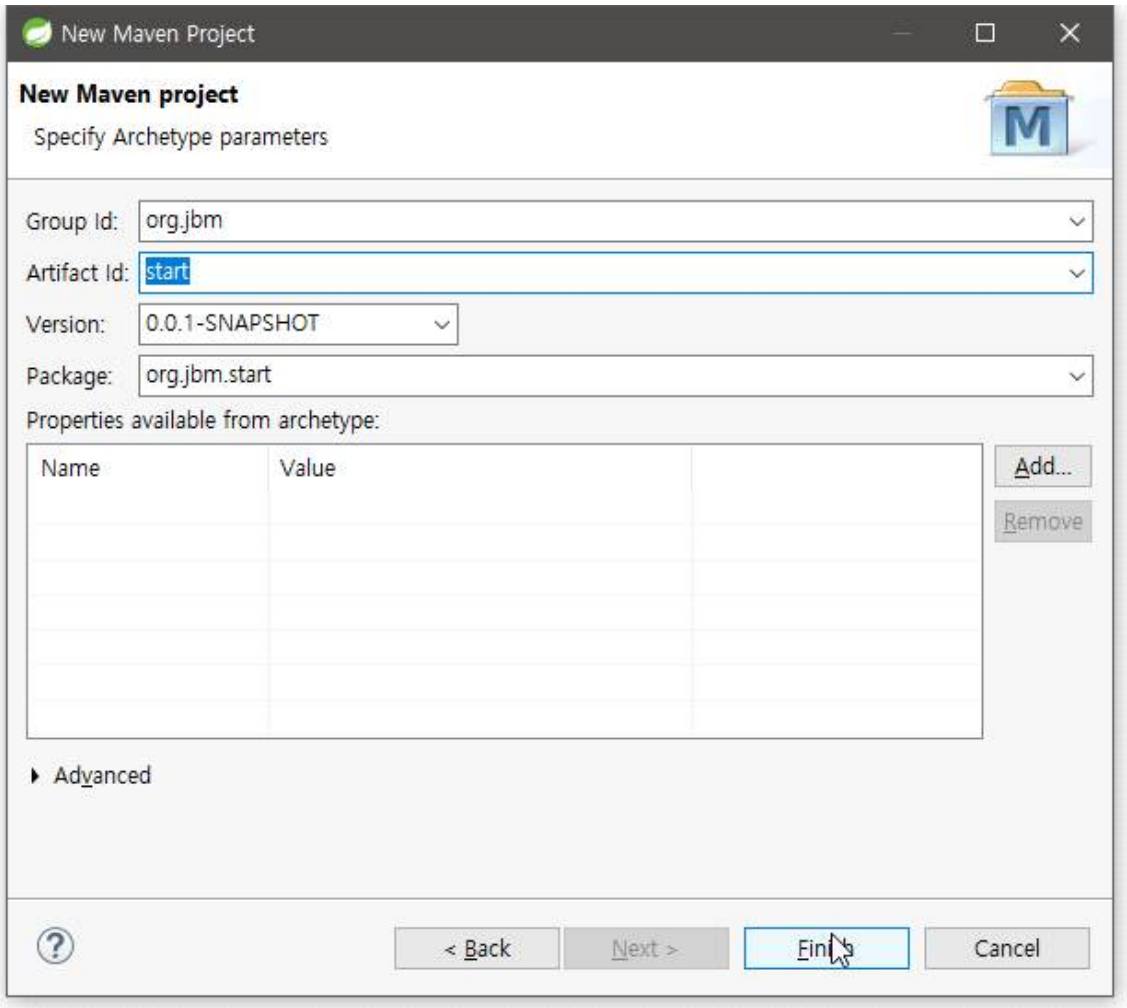

New
Select a wizard
Create a Maven Project
Wizards:
type filter text
eXERD
Git
Java
Java EE
Java Emitter Templates
JavaScript
JAXB
JPA
Maven
Checkout Maven Projects from SCM
Maven Module
Maven Project
< Back
Next >
Finish
Cancel


New Maven Project
New Maven project
Select project name and location
Create a simple project (skip archetype selection)
Use default Workspace location
Location:
Browse...
Add project(s) to working set
Working set:
More...
Advanced
< Back
Next >
Finish
Cancel

■ 기본으로 선택



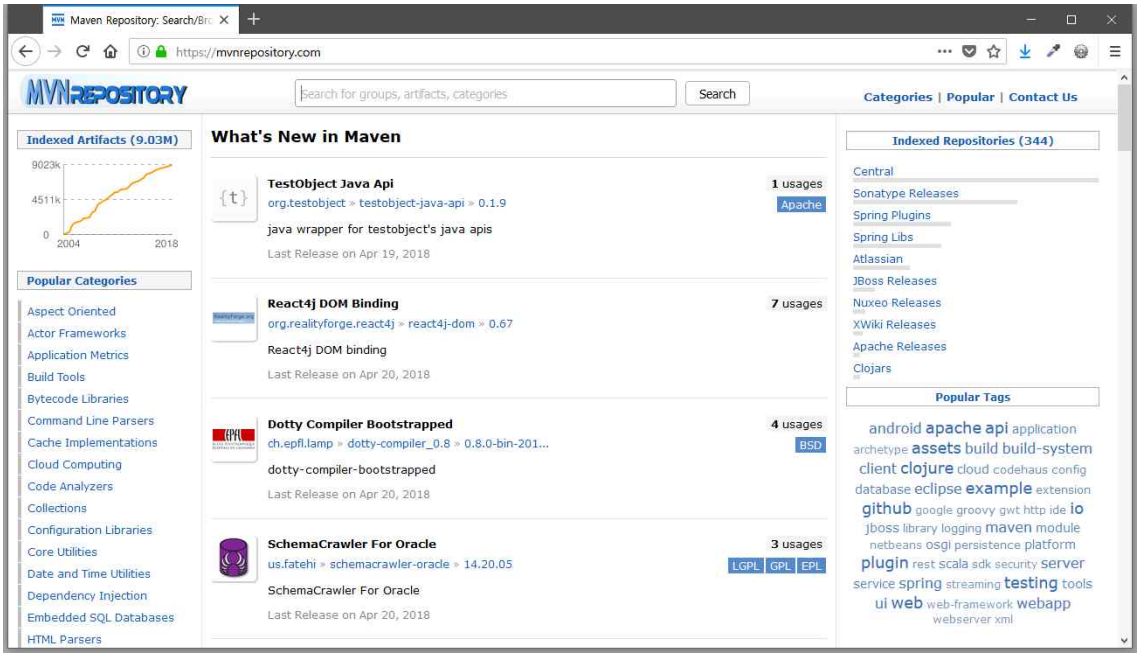
■ group id : 자신의 회사 / artifact id : 현재 프로젝트



■ Project Object Model (pom.xml)

1) 메이븐에서 중요한 역할

2) 의존성 등 프로젝트에 대한 설정을 할 수 있음



- http://mvnrepository.com/ 에서 검색 가능

3) 우리가 의존성을 추가하면 알아서 저장소(repository)에서 다운로드해줌

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>

        <groupId>org.jbm</groupId>
        <artifactId>start</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <packaging>jar</packaging>

        <name>start</name>
        <url>http://maven.apache.org</url>

        <properties>
                <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        </properties>

        <repositories>
                <repository>
                        <id>oracle</id>
```

```
                        <name>ORACLE JDBC Repository</name>
                        <url>https://maven.atlassian.com/3rdparty/</url>
                </repository>
        </repositories>

        <dependencies>
                <dependency>
                        <groupId>com.oracle</groupId>
                        <artifactId>ojdbc6</artifactId>
                        <version>12.1.0.1-atlassian-hosted</version>
                </dependency>
        </dependencies>
</project>
```

■ Spring의 기본 라이브러리 가져오기

```
<dependencies>
        <dependency>
                <groupId>org.springframework</groupId>
                <artifactId>spring-context</artifactId>
                <version>5.0.5.RELEASE</version>
        </dependency>
        <dependency>
                <groupId>com.oracle</groupId>
                <artifactId>ojdbc6</artifactId>
                <version>11.2.0.3</version>
        </dependency>
</dependencies>
```

7. 스프링 프레임워크와 DI(Dependency Injection)

■ 의존성(Dependency)

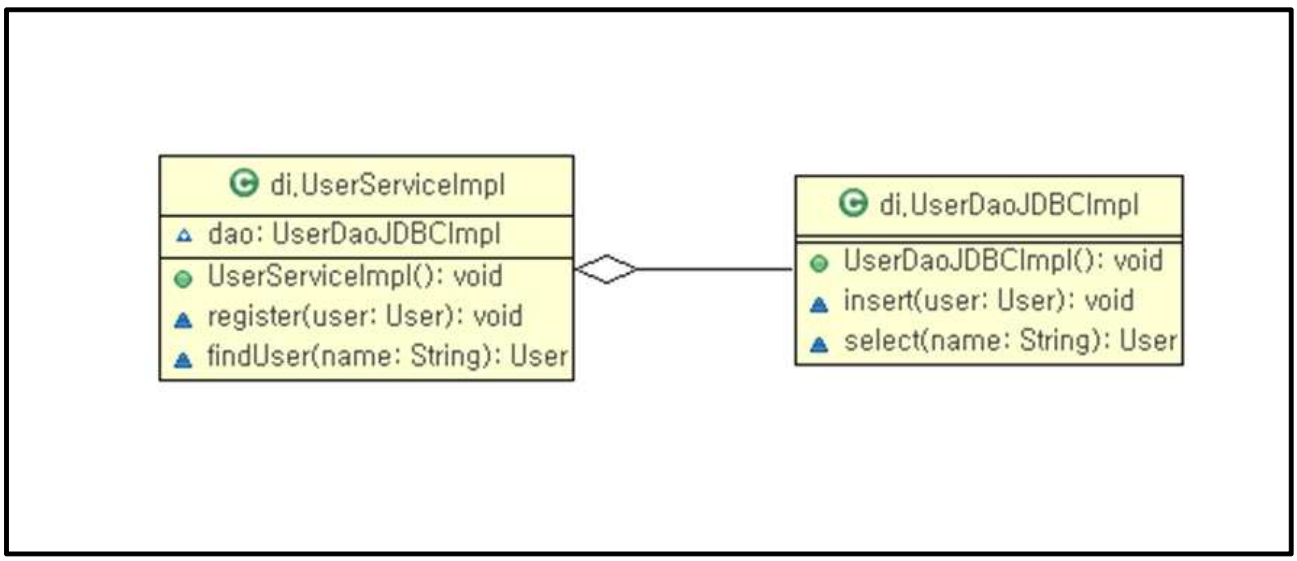
▮ 비즈니스 로직을 수행하기 위해서는 둘 이상의 클래스가 사용되는데, 각 객체는 협업할 객체의 참조를 취득해야할 책임이 있는데, 이것이 의존성이다.

▮ 객체간의 결합도가 높으면 테스트하기 어려운 코드가 만들어진다.

■ 의존성 주입 : 객체들은 객체의 생성 시점에 spring container로부터 의존성을 부여 받게 된다. 즉, 의존하는 객체를 주입받게 된다.

▮ 클래스가 구현 클래스에 의존하는 경우

- 클래스와 클래스간의 결합도가 높다.



- 자신이 사용할 DAO객체를 직접 new라는 키워드를 사용하여 생성하여 사용할 경우 의존성이 높음
- 프로젝트 스펙이 바뀌거나, 다른 DAO를 참조해야 할 때 코드의 수정이 불가피함

문제)

```
package ex1;

public class HelloApp {

	public static void main(String[] args) {

		________ bean = ____  ____________;

		bean._________("Spring");
	}
}
```

결과

```
Hello Spring!
```

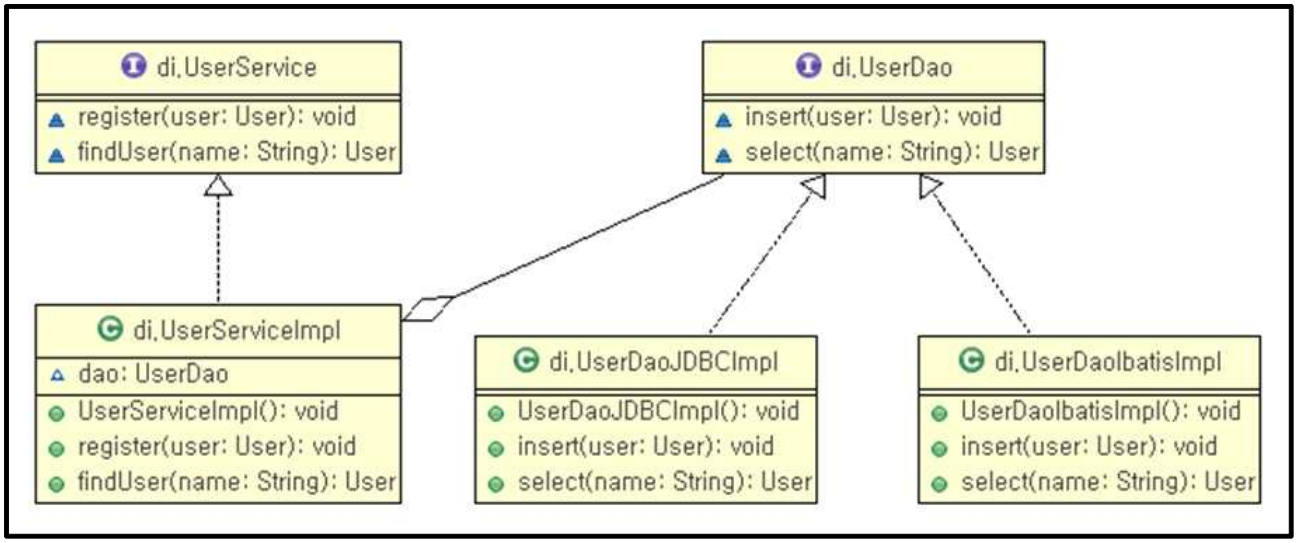
1번 예제

```
package ex1;

public class HelloApp {

}
```

```
package ex1;

public class MessageBean {

	public void sayHello(String name) {
		System.out.println("Hello!! "+name+"!");
	}

}
```

■ 클래스가 인터페이스에 의존하는 경우

- 클래스간의 결합도가 낮아진다.



- 인터페이스를 사용하여 의존성은 조금 낮아졌으나 여전히 의존성이 높음

2번 예제

인터페이스 작성

```
package ex2.bean;

public interface MessageBean {

	public void sayHello(String name);

}
```

구현클래스 작성

```
package ex2.bean;

public class MessageBeanImplKr implements MessageBean{

	@Override
	public void sayHello(String name) {
		System.out.println("안녕하세요 " + name+"!");
	}

}
```

```
package ex2.bean;

public class MessageBeanImplEn implements MessageBean{

	@Override
	public void sayHello(String name) {
		System.out.println("Hello "+ name+"!");
	}
```

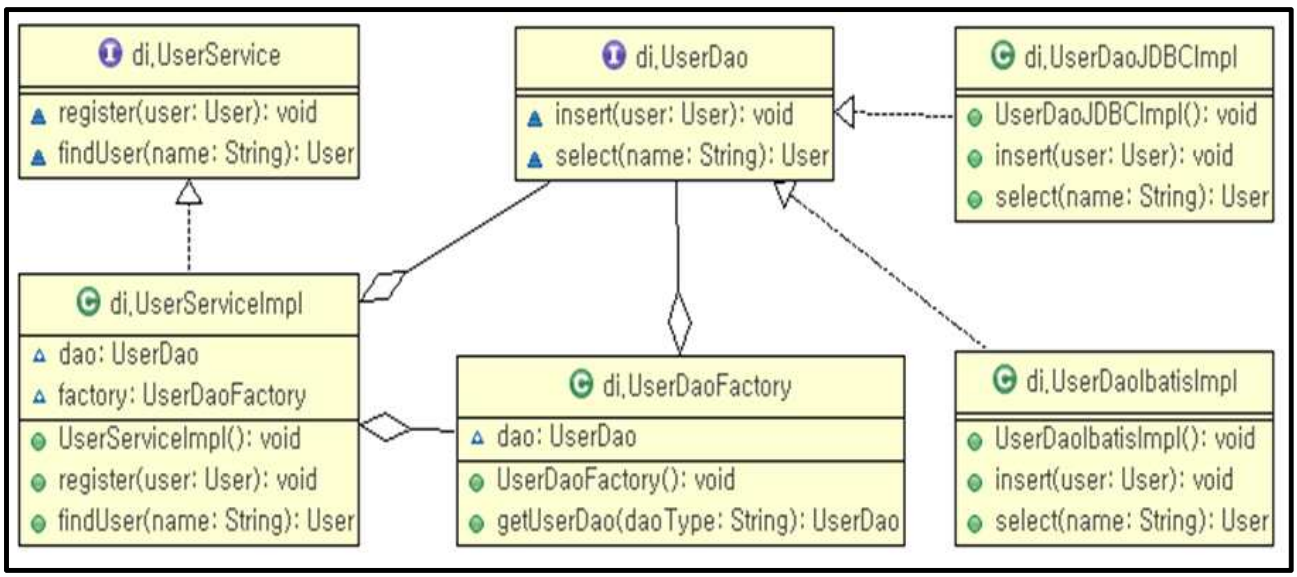
```
}
```

실행 클래스 작성

```
package ex2;

import ex2.bean.MessageBean;
import ex2.bean.MessageBeanImplEn;
import ex2.bean.MessageBeanImplKr;

public class HelloApp {

	public static void main(String[] args) {
		MessageBean bean = new MessageBeanImplEn();
		bean.sayHello("Spring");
	}

}
```

▮ factory pattern을 사용한 경우

- 구현클래스를 변경하더라도 소스 코드의 변화 없음



- DAO와 의존성은 낮아졌으나 Factory클래스와의 의존성은 여전히 높음

3번 예제

인터페이스 작성

```
package ex2.bean;
```

```
public interface MessageBean {

	public void sayHello(String name);

}
```

구현클래스 작성

```
package ex2.bean;

public class MessageBeanImplKr implements MessageBean{

	@Override
	public void sayHello(String name) {
		System.out.println("안녕하세요 " + name+"!");
	}

}
```

```
package ex2.bean;

public class MessageBeanImplEn implements MessageBean{

	@Override
	public void sayHello(String name) {
		System.out.println("Hello "+ name+"!");
	}

}
```

팩토리 작성

```
package ex3.factory;

import bean.MessageBean;
import bean.MessageBeanImplEn;
import bean.MessageBeanImplKr;


public class MessageBeanFactory {

	public static MessageBean getMessageBean(String country) {

		MessageBean bean = null;

		if(country.equals("kr")) {
			bean = new MessageBeanImplKr();
		}else if(country.equals("en")) {
			bean = new MessageBeanImplEn();
```

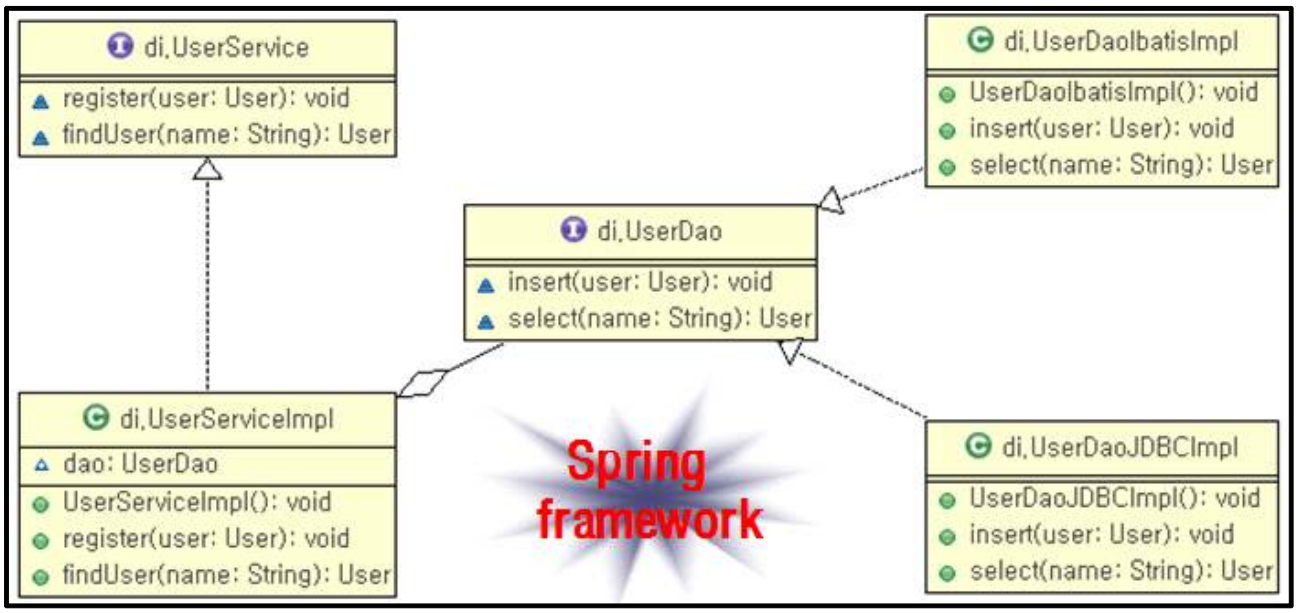
```
                }

                return bean;

        }

}
```

실행 클래스 작성

```
public class HelloApp {

        public static void main(String[] args) {

                MessageBean bean = MessageBeanFactory.getMessageBean("en");

                bean.sayHello("Spring");


        }
}
```

▮ spring DI 사용한 경우

- spring container가 객체 사이의 의존관계를 조립한다.



- 객체 외부의 조립기가 각 객체의 의존관계를 설정해 줌
- 이를 DI패턴이라 함

4번 예제

설정파일

```
<?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="messageBean" class="bean.MessageBeanImplEn"/>
</beans>
```

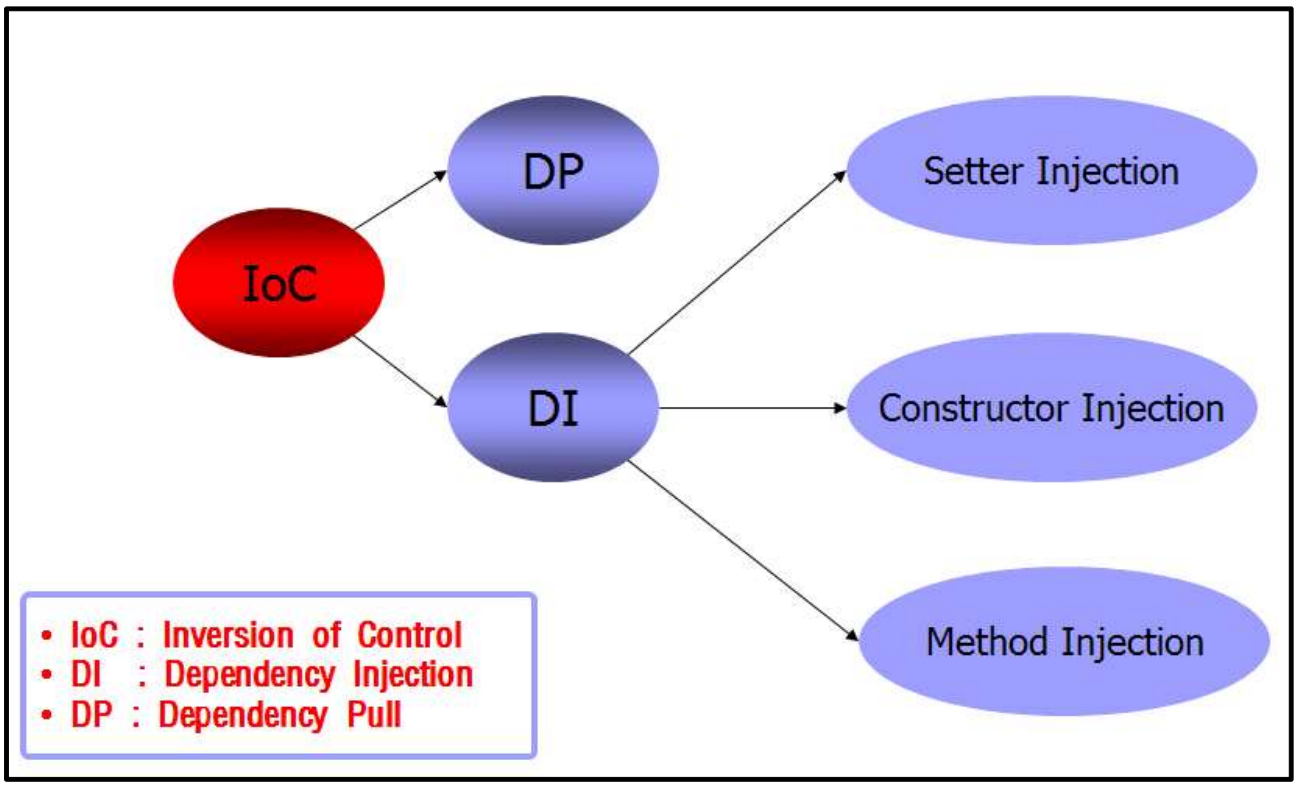
실행 클래스 작성

```
package ex4;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import bean.MessageBean;

public class HelloApp {

        public static void main(String[] args) {
                ApplicationContext context =
                          new ClassPathXmlApplicationContext("ex4/beans.xml");
                MessageBean bean = (MessageBean)context.getBean("messageBean");

                bean.sayHello("Spring");

        }

}
```

■ 제어역행(IoC : Inversion of Control)

각 객체는 협업할 객체의 참조를 취득해야하는 책임이 있는데, 제어역행은 의존하는 객체를 역행적으로 취득하는 것이다. IoC는 한 객체가 의존성을 가지는 다른 객체의 참조를 취득하는 방법에 대한 책임의 역행이라는 의미를 가지고 있다. IoC를 적용하면 객체들은 어떤 존재에 의해 객체를 생성할 때 의존성을 가지는 객체를 주입받게 된다.

※ 의존성을 가지는 객체를 참조하는 법



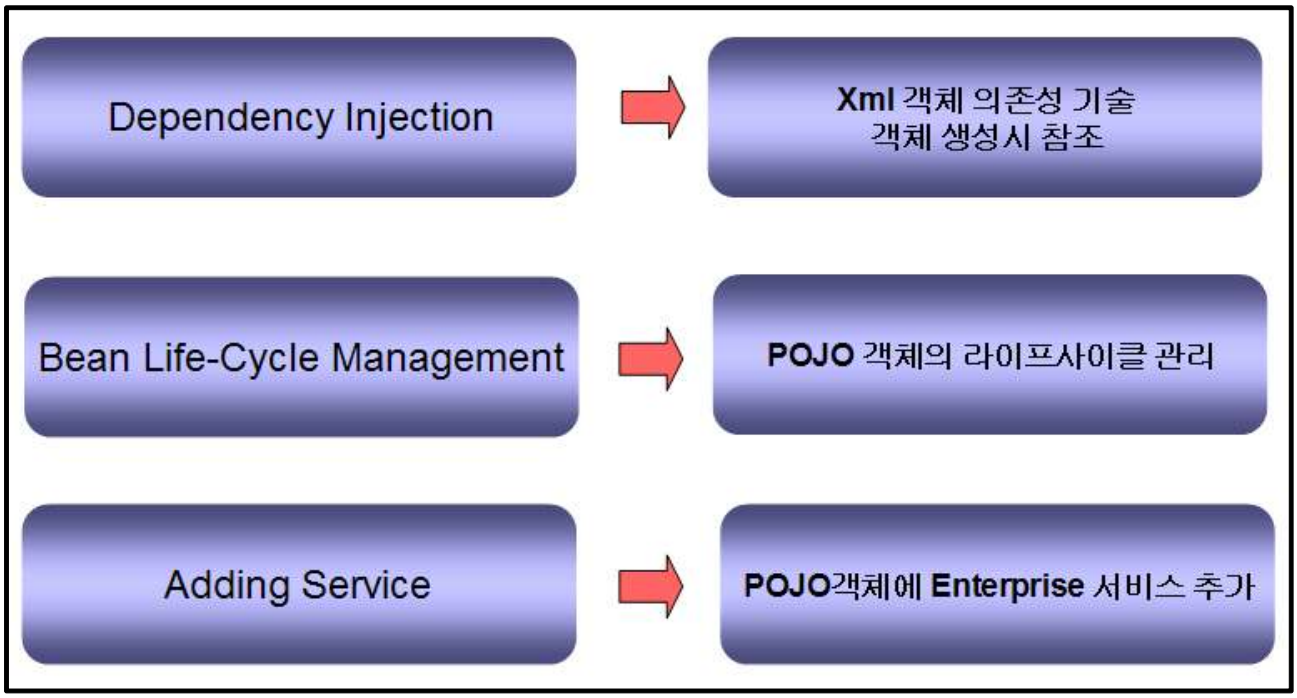
IoC(Inversion of Control) 란?

제어 역행화는 말 그대로 제어가 일반적인 흐름을 따르지 않고, 반대로 흘러간다는 뜻으로 각각의 프로그램이 가지고 있던 구현 객체에 대한 정보가 이젠 프레임워크에서 관리되는 것임. 현재는 DI라는 용어를 더 많이 사용함

의존성이란?

보통 비즈니스 로직을 수행하기 위해 둘 이상의 클래스를 사용한다. 전통적으로 각 객체는 협업할 객체의 참조를 취득해야 하는 책임이 있다. 이것이 의존성이다. 객체 간의 결합도가 높으면 테스트하기 어려운 코드를 만들어 낸다.

■ spring framework에서의 DI



8. 스프링 프레임워크의 DI 방법

■ 특정 interface를 통한 의존성 설정

- 거의 사용하지 않음

■ Setter를 통한 의존성 설정

- 클래스

```
public class ServiceImpl implements Service {

        private int timeout;

        private AccountDao dao;


        public void setTimeout(int timeout) {

                this.timeout = timeout;

        }

        public void setDao(AccountDao dao) {

                this.dao = dao;

        }

}
```

- xml문서 설정

```
<bean id="service" class="service.ServiceImpl">
     <property name="timeout"><value>30</value></property>
     <property name="dao" ref="accountDao"></property>
</bean>
```

■ Constructor를 통한 의존성 설정

- 클래스

```
public class ServiceImpl implements Service {
        private int timeout;
        private AccountDao dao;

        public ServiceImpl(int timeout, AccountDao dao) {
                this.timeout = timeout;
                this.dao = dao
        }

        //Business methods from service
}
```

- xml문서 설정

```
<bean id="service" class="service.ServiceImpl">
     <constructor-arg><value>30</value></constructor-arg>
     <constructor-arg ref="accountDao"></constructor-arg>
</bean>
```

9. xml 설정과 Wiring

■ DTD와 xml 스키마, 2가지 방법을 이용하여 설정파일을 작성할 수 있음

- dtd를 이용

```
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
       "http://www.springframework.org/dtd/spring-beans-2.0.dtd">
```

- xml스키마를 이용

```
<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">
```

■ 빈의 생성 및 사용

- 컨텍스트 파일의 루트 요소는 <beans>

- 빈을 static 메서드를 이용해 객체를 생성해야 하는 경우

```
       <bean id="daoIbatis" class="di.dao.UserDaoIbatisImpl" factory-method="getInstance"/>
```

- null값을 설정할 수 있음

■ 의존 관계 설정

- 생성자 방식(<constructor-arg>태그 사용)

```
       <bean id="daoIbatis" class="di.dao.UserDaoIbatisImpl">
         <constructor-arg><value>1</value></constructor-arg>
       </bean>
```

- 다른 빈을 참조하는 경우 <ref>태그나 <constructor-arg>의 ref속성을 이용

```
        <bean id="daoIbatis" class="di.dao.UserDaoIbatisImpl">
          <constructor-arg><ref bean="articleDao" /></constructor-arg>
        </bean>
```

■ Setter를 이용한 방식(<property>태그 사용)

- 기본자료형과 String의 경우는 <value>태그나 <property>의 value속성을 이용

```
   <bean id="daoIbatis" class="di.dao.UserDaoIbatisImpl">
         <property><value>1</value></property>
   </bean>
```

- 다른 빈을 참조하는 경우 <ref>태그나 <constructor-arg>의 ref속성을 이용

```
        <bean id="daoIbatis" class="di.dao.UserDaoIbatisImpl">
            <property><ref bean="articleDao" /></property>
        </bean>
```

- xml 네임스페이스를 이용한 프로퍼티 설정(접두어 : p 사용)

```
<?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:p="http://www.springframework.org/schema/p"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans.xsd">

        <bean id="steampackMarine" class="starcraft.Marine"
                p:no="2"
                p:hp="40"
                p:command-ref="steampackAttack"
                init-method="init"
        />
</beans>
```

p:프로퍼티이름(빈의 멤버필드), p:프로퍼티이름-ref를 사용하면 편리함

■ 빈 객체의 범위 지정

- <bean> 태그의 scope 속성 값

이 름	설명
singleton	스프링 컨테이너에 한 개의 빈 객체만 존재함(기본값)
prototype	빈을 사용할 때 마다 객체를 생성한다.
request	http 요청마다 빈 객체를 생성한다.(WebApplicationContext에서만 가능)
session	http 세션마다 빈 객체를 생성한다.(WebApplicationContext에서만 가능)
global-session	global http 세션에 빈 객체를 생성한다.(포틀릿을 지원하는 context에서만 가능)

■ Collection Wiring

- List 타입

```
<bean id=“listBean” class=“example.ListBean”>
   <property name=“listPro”>
     <list><ref bean=“someBean”/></list>
   </property>
</bean>
```

- Set 타입

```
<bean id=“setBean” class=“example.SetBean”>
    <property name=“setPro”>
      <set><ref bean=“someBean”/></set>
    </property>
</bean>
```

- Map 타입

```
<bean id=“mapBean” class=“example.MapBean”>
     <property name=“mapPro”>
         <map>
         <entry key=”key1”><value>value1</value></entry>
         <entry key=“key2”><ref bean=“someBean”/></entry>
         </map>
     </property>
</bean>
```

■ 의존관계 자동 설정

- <bean> 태그의 autowire 속성으로 설정

- autowire 타입

타입종류	설명
byName	프로퍼티의 이름과 같은 이름을 갖는 빈 객체를 설정한다.
byType	프로퍼티의 타입과 같은 타입을 갖는 빈 객체를 설정한다.
constructor	생성자 파라미터 타입과 같은 갖는 빈 객체를 생성자에 전달한다.
autodetect	construtor 방식을 먼저 적용하고, byType방식을 이용하여 적용한다.

- 특별한 경우가 아니라면 사용하지 않아야 함