

Spring Framework

- 스프링 프레임워크 소개

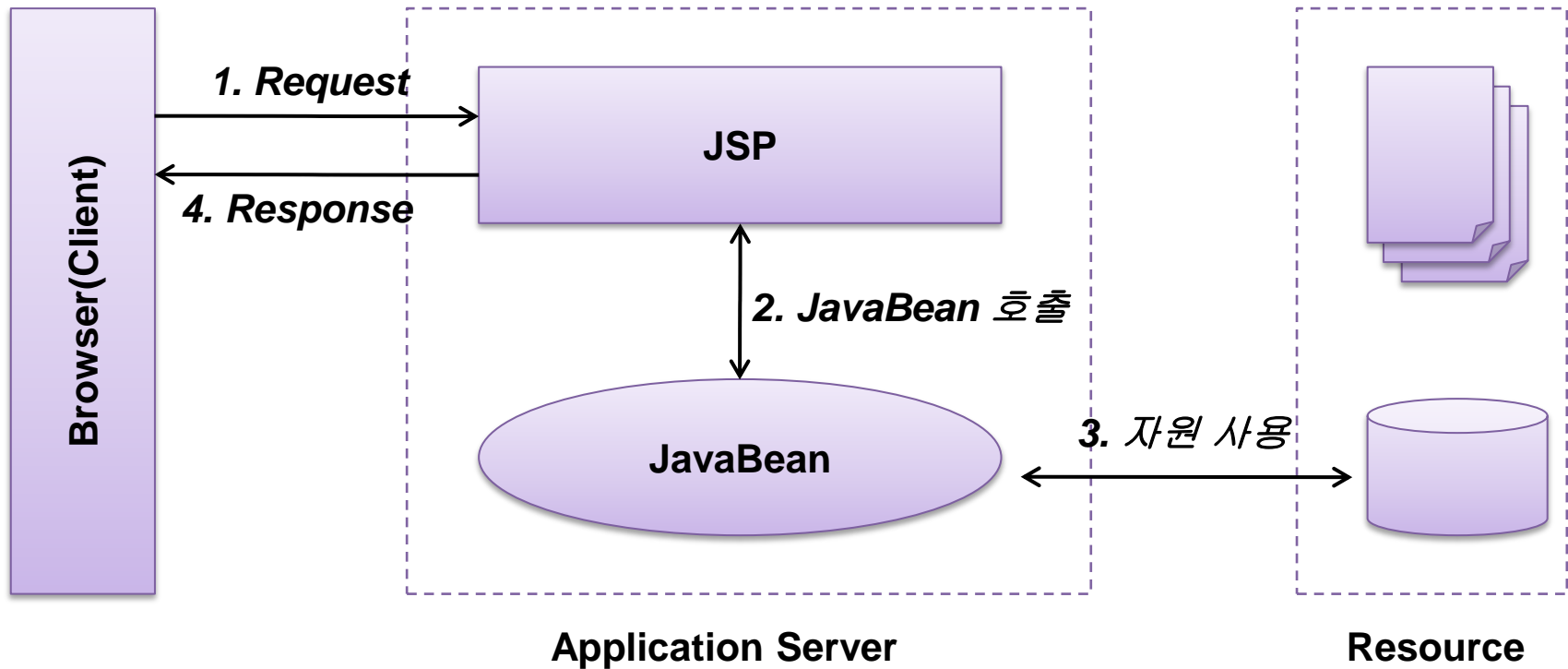
■ CONTENTS

- Web Application 설계방식
- 프레임워크란
- 스프링 프레임워크란
- 스프링프레임워크 설치와 모듈 구성
- DeppenDency Injection과 스프링프레임워크

■ Web Application 설계방식

• 모델1 개요

- JSP 만 이용하여 개발하는 경우
- JSP + Java Bean을 이용하여 개발하는 경우
- Model2의 Controller 개념이 모호



■ Web Application 설계방식

- **모델1의 장단점**

- **장점**

- 개발속도가 빠름
 - 개발자의 기술적인 숙련도가 낮아도 배우기 쉬워 빠르게 적용 가능

- **단점**

- JSP 페이지에서 프레젠테이션 로직과 비즈니스 로직이 혼재되어 복잡
 - 로직의 혼재로 인해 개발자와 디자이너의 작업 분리가 어려움
 - JSP 코드의 복잡도로 인해 유지보수가 어려워짐

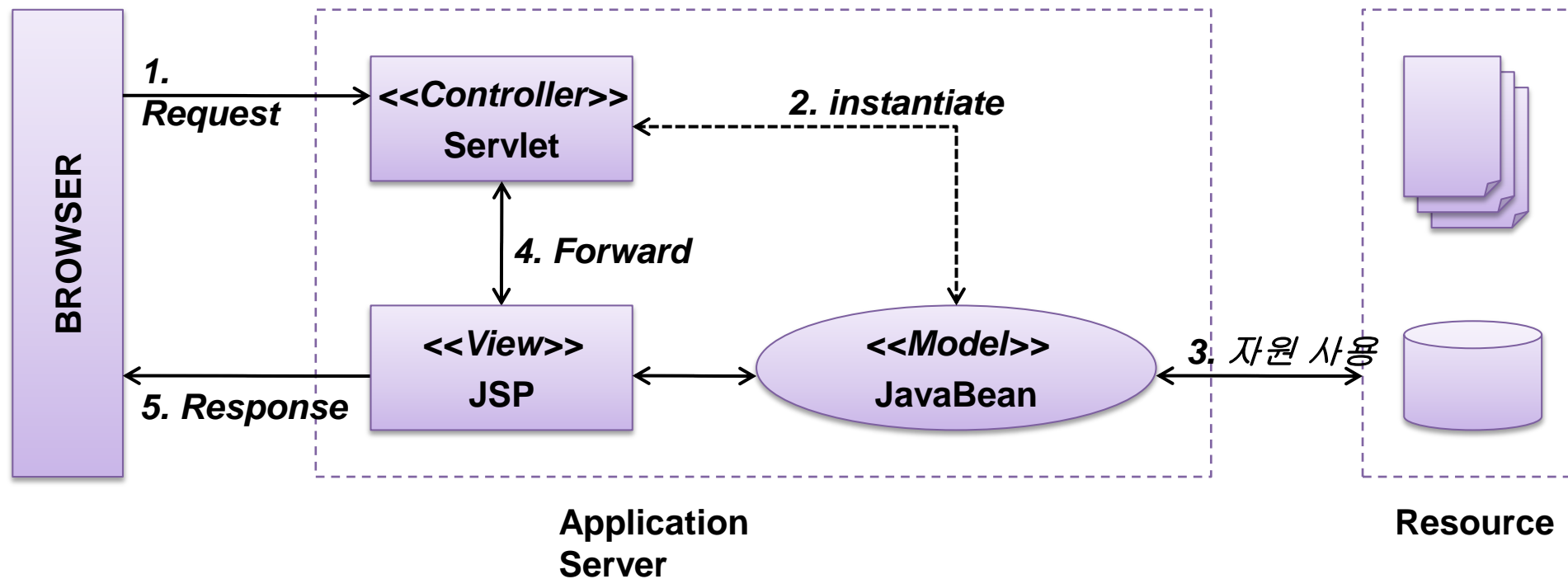
- **웹 애플리케이션이 복잡해지고 사용자 요구가 증가함에 따라 새로운 개발방식을 요구 → 모델2**

■ Web Application 설계방식

- **모델2 개요**

- GUI 개발모델인 MVC를 웹 애플리케이션에 적용한 방식
- **Application의 역할을 Model – View – Controller로 분리**
- **Model: Business Logic 담당**
 - Java Bean으로 구현
 - Business Service(Manager)
 - Business Logic의 workflow 관리
 - DAO (Data Access Object) : DB 연동해 Business Logic 처리
- **View: Client에게 응답을 처리한다.**
 - JSP로 구현
- **Controller: 클라이언트의 요청을 받아 Model과 View사이에서 이벤트 흐름 제어**
 - Servlet으로 구현
 - Client의 요청을 받아 Client가 보낸 Data를 읽고 검사
 - Model에게 Business Logic을 요청
 - Model의 처리 결과에 맞는 View에게 응답 요청

■ Web Application 설계방식



■ Web Application 설계방식

- **모델2의 장단점**

- **장점**

- 비즈니스 로직과 프리젠테이션의 분리로 인해 어플리케이션이 명료해지며 유지보수와 확장이 용이함
 - 디자이너와 개발자의 작업을 분리해 줌

- **단점**

- 개발 초기에 아키텍처 디자인을 위한 시간의 소요로 개발 기간이 늘어남
 - MVC 구조에 대한 개발자들의 이해가 필요함

■ 프레임워크

- 프레임워크

- 뼈대 혹은 틀
- 소프트웨어 관점 : 아키텍처에 해당하는 골격 코드

- '아키텍처' , '골격'

- 애플리케이션을 개발할 때 가장 중요한 것이 애플리케이션의 구조를 결정하는 것이 아키텍처인데 이 아키텍처에 해당하는 골격 코드를 프레임워크가 제공한다.

예) 컵을 만든다.

A와 B는 자유롭게 컵을 만든다.

A의 퇴사, B가 A가 만든 컵을 고쳐야 한다....

■ 프레임워크

- **기존 애플리케이션 개발 과정의 문제점**

- 시스템 개발과정에서 대부분의 개발자들은 산출물에 입각해서 개발을 하므로 아키텍처의 일관성이 잘 유지 되지만 유지보수 과정에서는 개발자의 경험에 의존하는 경우가 많다.

- **프레임워크는 이러한 문제를 근본적으로 해결**

- 애플리케이션 개발에 기본이 되는 뼈대나 틀을 제공
개발자에게 모든 것을 위임하는 것이 아니라 애플리케이션의 기본 아키텍처는 프레임워크가 제공,
그 뼈대에 살을 붙이는 작업만 개발자가 하는 것이다.

■ 프레임워크

- **프레임워크의 장점**

- 1. 빠른 구현 시간**

개발자는 비즈니스 영역만 구현하면 됨. 제한된 시간에 많은 기능을 구현할 수 있다.

- 2. 쉬운 관리**

유지보수에 들어가는 인력과 기간을 줄일 수 있다.

- 3. 개발자의 역량 획일화**

숙련된 개발자와 초급개발자의 코드가 비슷해진다.

관리자 입장에서 개발 인력을 더 효율적으로 구성할 수 있다.

- 4. 검증된 아키텍처의 재사용과 일관성 유지**

아키텍처에 관한 고민이나 검증 없이 애플리케이션을 개발한다.

유지보수 과정에서 아키텍처가 왜곡되거나 변형되지 않는다.

■ 스프링 프레임워크란

- 로드존슨이 2004년에 만든 오픈 소스 프레임워크.
- 스프링 프레임워크가 등장하기 전까지는 자바 기반의 엔터프라이즈 애플리케이션은 대부분 EJB(Enterprise Java Beans)로 개발되었다.
- **EJB의 문제점**
 - 복잡하고, 고가의 비용발생, 많은 시간과 노력이 필요.
스팩의 복잡함, 학습에 많은 시간이 필요, 유지보수 역시 복잡함, 설치를 위해 WAS(Web Application Sever)가 필요
: JEUS, Weblogic, WebSpere 등, 수 천만원의 고가 장비.
 - 다양한 디자인 패턴을 이해하고 적용해야 함.
- 스프링 프레임워크는 이미 많은 디자인 패턴이 적용되어 배포되는 프레임워크이기 때문에 많은 디자인 패턴을 사용하는 것과 같다.

■ 스프링 프레임워크란

- 프레임 워크가 구현하고 있는 디자인 패턴

1) Proxy 패턴

스프링에서는 AOP(Aspect Oriented Programming)에 기반해 수많은 로깅 처리, 트랜잭션 처리를 하고 있다. Spring의 AOP는 CGLIB이라는 런타임 프록시 생성 라이브러리를 사용해, AOP를 위한 프록시 객체를 생성한다. xml과 AspectJ 스크립트를 사용해 대상객체들에 대한 프록시를 생성하므로 코드로 적을 수는 없다.

2) Singleton 패턴

스프링의 XML 및 configuration 파일안에 설정된 Bean들은 기본적으로 싱글톤 패턴이다.

3) Template Method 패턴

jdbcTemplate을 사용할 때 사용하는 jdbcTemplate, jmx 사용시에 사용되는 jmxTemplate, jpa를 사용할 때 사용하는 jpaTemplate 등의 클래스는 템플릿 메소드 패턴 방식으로 구현되어 있다. 템플릿 메소드 패턴이란 핵심 로직 처리 부분은 부모 클래스에 대부분 구현되어 있고, 상속받은 자식 클래스는 최종적인 부분만 각기 다르게 구현하는 방식이다.

jdbcTemplate 는 커넥션 완전히 정리하기, 결과반복, 예외처리, 트랜잭션 처리 등의 복잡하고 세부적인 사항들을 스프링 내부에서 알아서 처리해주는 코드가 미리 들어가 있고, 또한 다양한 generic 을 지원한다. 최종적인 db binding(SELECT, UPDATE, INSERT, DELETE 등의 쿼리를 전달하는 부분)을 jdbcTemplate이 템플릿이 호출한다. 물론 db는 oracle 혹은 mysql 어느 것이라도 될 수 있다.

4) Dependency Injection

프레임워크(주로 서블릿 컨테이너)에서 사용자가 사용하는 객체를 생성하고 관리해주는 방식이다.

■ 스프링 프레임워크란

- 프레임 워크가 구현하고 있는 디자인 패턴

5) MVC(Model View Controller) 패턴

Spring MVC에서는 유저가 작성한 Controller가 하나의 웹서버에 의존적인 서블릿 방식으로 작성하고 동작하는 것이 아니라, POJO방식으로 구현된다. *pojo : 상속하거나 구현하는 클래스가 아닌 클래스

1. Url Controller에 대한 테스트가 더욱 독립적이고 간편해지게 된다.
2. Controller의 역할을 논리적인 view의 이름 (예를 들어 home.jsp) 으로 한정해주고, 실제 UI구성은 View Resolver (JSP나 Velocity같은 템플릿 처리엔진)이 처리하기 때문에, 컨트롤러의 독립성과 재 사용성을 높여주게 된다.

6) Front Controller

스프링은 DispatcherServlet 이라는 Front Controller방식을 채택하고 있다.

이 Front Controller는 모든 User Controller 바로 앞에서 들어오는 모든 요청(request)을 처리해서 User Controller로 전달해 주고, User Controller로부터 결과를 받는 역할을 한다.

DispatcherServlet 에서 예외처리나 쿠키처리, 브라우저로부터의 파일 수신 등 복잡하고 까다로운 역할을 하고, 유저는 순수 로직에만 집중할 수 있게 도와준다.

7) View Helper

스프링은 매우 많은 자체 JSP tag들과 Velocity macro등을 가지고 있다. 그렇기 때문에 JSP나 Velocity 로 서버쪽 UI를 구성한경우, 스프링기반 컨트롤러와 자연스럽게 연동할수 있다. Apache Tile와 SiteMesh등이 대표적이다.

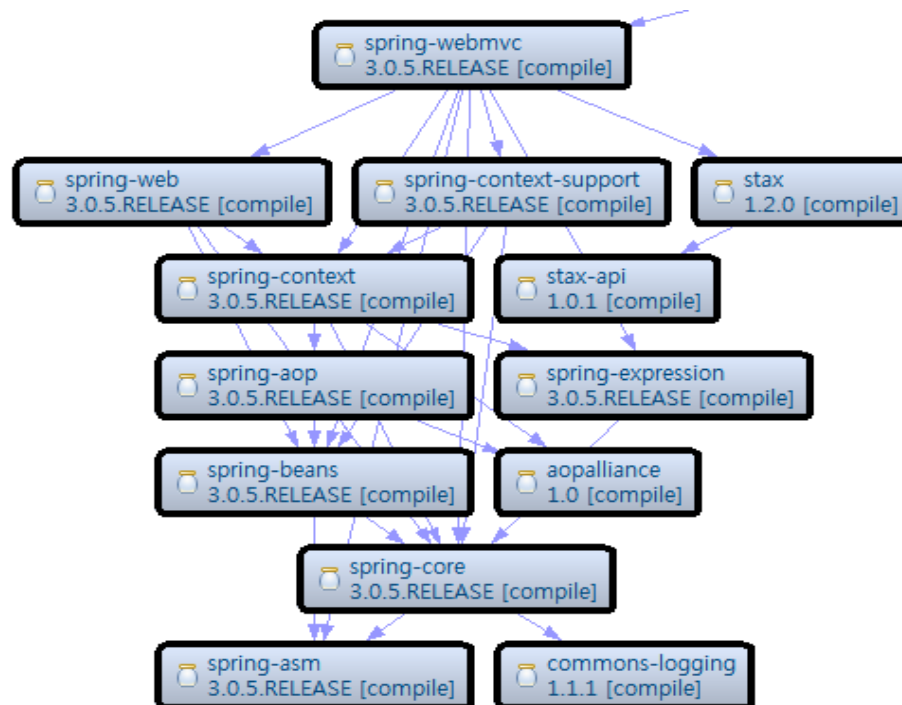
■ 스프링 프레임워크의 주요 특징

- 스프링은 **경량의 프레임워크**이다.
 - 자바 객체를 담고 있는 컨테이너. **자바객체의 생성, 소멸과 같은 라이프 사이클을 관리한다.**
- 스프링은 **DI(dependency Injection)패턴**을 지원한다.
 - 설정파일을 통해서 의존관계를 설정할 수 있다.
- 스프링은 **AOP(Aspect Oriented Programming)**를 지원한다.
 - 트랙잭션이나 로깅, 보안과 같은 공통 기능을 분리해서 각각의 모듈에 적용할 수 있다.
- 스프링은 **POJO(Plain Old Java Object)**를 지원한다.
 - 특정 인터페이스나 클래스를 상속받지 않은 자바 객체를 스프링 컨테이너가 저장하고 있다.
- **트랜잭션 처리를 위한 일관된 방법을 제공한다.**
- **영속성과 관련된 다양한 API를 제공한다.**
 - **JDBC, iBatis(myBatis), Hibernate, JPA**등과 같은 프레임워크와의 연동을 지원한다.
- 자체적으로 **MVC 프레임워크**를 제공
 - 스프링만 가지고 MVC 기반의 웹 어플리케이션을 개발

■ 스프링 프레임워크의 주요 특징

• 경량

- 스프링은 크기 측면에서 EJB에 비해 상대적으로 매우 가볍다.
- 여러 개의 모듈로 구성, 모듈은 하나 이상의 JAR 파일로 구성 한다.
- JAR 파일만 있으면 개발과 실행이 가능하다.
- 스프링을 이용해서 만든 애플리케이션의 배포 역시 매우 빠르고 쉽다.
- POJO - 특별한 규칙 없이 단순한 객체 (상속이나 포함의 조건이 없음.)



■ 스프링 프레임워크의 주요 특징

- **Plain Old Java Object[POJO] 지원**

- 특정 규약 및 환경에 종속적이지 않은 평범한 일반 자바 클래스 의미
- Spring 개발에 POJO 클래스를 활용할 수 있다는 건 특정 구현 기술에 종속적이지 않다는 의미
- 개발 후의 테스트 시에도 DB와 특정 서버 없이도 테스트를 할 수 있기 때문에 개발이 빨라진다는 장점이 있음
- Tomcat 과 같은 서블릿 컨테이너의 경우 서블릿 생성이나 필터 등을 생성 할 때에는 반드시 특정 클래스를 상속하거나 인터페이스를 구현해야합니다.

■ 스프링 프레임워크의 주요 특징

- **Dependency Injection[DI] 지원**
 - 객체는 의존하고 있는 객체를 코드 상에서 직접 생성하거나 검색할 필요가 없음
 - DI는 스프링 컨테이너가 지원하는 핵심 개념 중 하나
 - 스프링은 설정 파일이나 어노테이션을 이용하여 객체 간의 의존 관계를 설정할 수 있다.
- **DI 지원 : 의존성주입(= Ioc (Inversion of Control : 제어역행))**
- **객체간의 의존관계를 객체 자신이 아닌 외부의 조립기가 수행**
 - Spring은 설정 파일(xml)이나 애노테이션을 통해서 객체간의 의존 관계를 설정 할 수 있도록 함

■ 스프링 프레임워크의 주요 특징

• 제어의 역행

- 프로그램 개발에서 낮은 결합도(의존관계)와 높은 응집도를 고려해서 개발.
- 제어의 역행(Inversion of Control, IoC)을 통해 애플리케이션을 구성하는 객체간의 결합, 즉 낮은 결합도를 유지한다.

객체를 직접 생성하는 것이 아니라 설정을 통해 주입 해준다.

즉 참조변수에 객체를 생성하는 것이 아니라 객체를 따로 생성해서 참조변수를 넣어주는 것이다.

- IOC가 적용되면 객체 생성을 자바 코드에서 직접 처리하는 것이 아니라 스프링 컨테이너가 대신 처리, 객체와 객체 사이의 의존관계 역시 컨테이너가 처리한다.
- 따라서 자바 코드에서 의존관계가 명시적으로 처리되지 않기 때문에 결합도가 떨어져서 유지보수가 편리해진다.

■ Dependency Injection과 스프링프레임워크

```
class A{  
    B b = new B();    //B의 instance 생성  
    public void print(){  
        b.a();  
    }  
}
```



```
class B{  
    public void a(){  
        System.out.println("B.a() print....");  
    }  
}
```

- **A has a B. (A는 B에 의존한다.)**

- 의존하는 객체를 취할 때 사용하는 클래스(A)측에서 호출. 이를 제어순행이라고 함

- **DI는 의존하는 객체에 대한 취득(획득)의 책임이 사용하는 class에 있지 않고 스프링 컨테이너가 제공(주입)해 준다.**

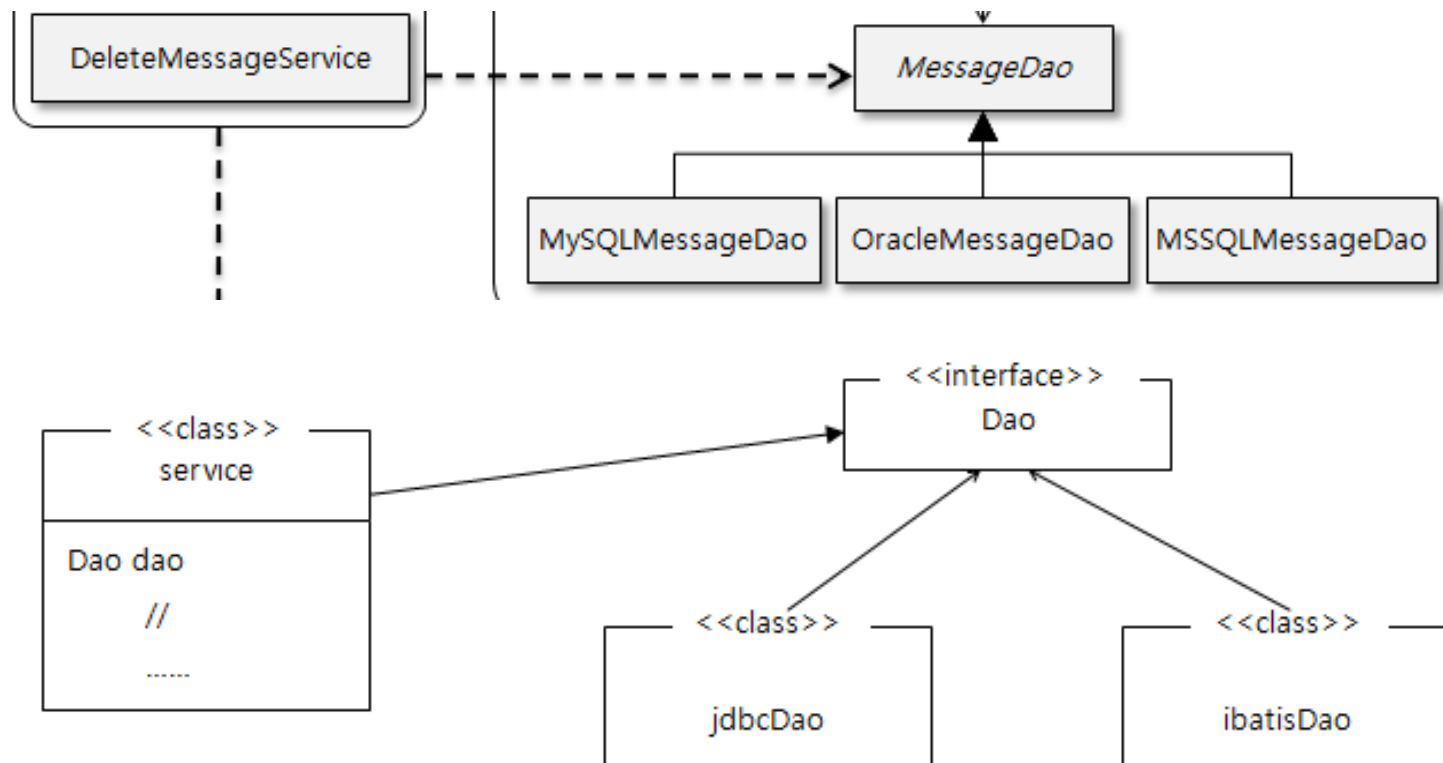
- 즉, A class에서 B class의 메소드를 사용하기 위해서는 B b = new B();를 통해 B class의 instance인 b를 생성해야 하는데,

스프링의 DI는 A class를 생성할 때 B class의 instance(객체)를 생성하여 주입까지 해주는 기능을 제공.

그러므로 개발자는 A class에서 B b = new B();라는 식의 코드를 작성할 필요가 없이 스프링 컨테이너가 만들어 놓은 instance(bean)를 가지고 와서 사용만 하면 됨. (xml 설정을 통해)

■ Dependency Injection과 스프링프레임워크

- 결합도를 낮추는 구조



■ Dependency Injection과 스프링프레임워크

- DAO Interface를 통한 의존성 낮춤

B class의 메소드 이름이 바뀌면 A class의 메소드 이름도 바뀌어야 한다.(의존성이 높음).

이는 내가 의존하는 class가 바뀌면 그것을 사용하는 모든 class가 전부 수정해야 한다는 문제점이 있다.

이런 결합도가 높은 코드를 위해 interface 기반으로 만들어 결합도를 낮추어야 한다.

```
dao = new jdbcDao();
```

```
dao = new ibatisDao();
```

이런 식의 구조는 의존성을 낮출 수 있다.

스프링의 경우 컨테이너가 **빈 팩토리 역할(객체를 주입해주는 역할)**로 설정파일에서만 필요한 설정을 해주면 기존코드 변경하지 않고도 dao의 값(jdbc인지, ibatis인지)을 가져와 사용할 수 있다.

xml(설정파일)을 통해 객체의 의존성을 설정할 수 있기 때문이다.

즉, 코드의 변경 없이 xml의 설정만으로 개발자가 원하는 객체의 주입으로 바꿀 수 있다.

■ 스프링 프레임워크의 주요 특징

- 트랜잭션 처리를 위한 일관된 방법 제공
 - **JDBC API** 및 **JTA**를 사용하거나 컨테이너가 제공하는 트랜잭션을 사용, 설정 파일을 통해 트랜잭션 관련 정보를 관리하기 때문에 특정 트랜잭션 구현 방법에 상관없이 동일한 코드를 여러 환경에서 사용 가능
 - 선언적인 트랜잭션을 지원하여 코드를 수정하지 않고도 트랜잭션을 적용 및 변경 가능

■ 스프링 프레임워크의 주요 특징

- 컨테이너

- 컨테이너는 특정 객체의 생성과 관리를 담당하고 운용에 필요한 다양한 기능을 제공
- 컨테이너는 일반적으로 서버 안에 포함되어 배포 및 구동된다.
- 대표 컨테이너 'Servlet 컨테이너', 'EJB 컨테이너'가 있다.
- 'Servlet 컨테이너' - TOMCAT
- 스프링도 컨테이너.

■ 스프링 프레임워크의 주요 특징

- **Enterprise Application에서 필요로 하는 기능 제공**
 - Spring은 단순한 툴과 기본적인 개발 환경만으로도 Enterprise 개발에서 필요로 하는 주요한 기능을 갖춘 애플리케이션 개발에 적합
 - 예 : 고비용을 요하는 WAS(Web Application Server)를 사용하지 않고도, WAS에서 지원해주는 트랜잭션 관리 및 보안, 객체 pooling과 같은 Enterprise 개발의 고급 기술들도 **Spring Framework를 통해서 단순한 작업 및 설정만으로 동일한 수준의 기능들을 사용할 수 있게 됨**

■ 스프링 프레임워크의 장점

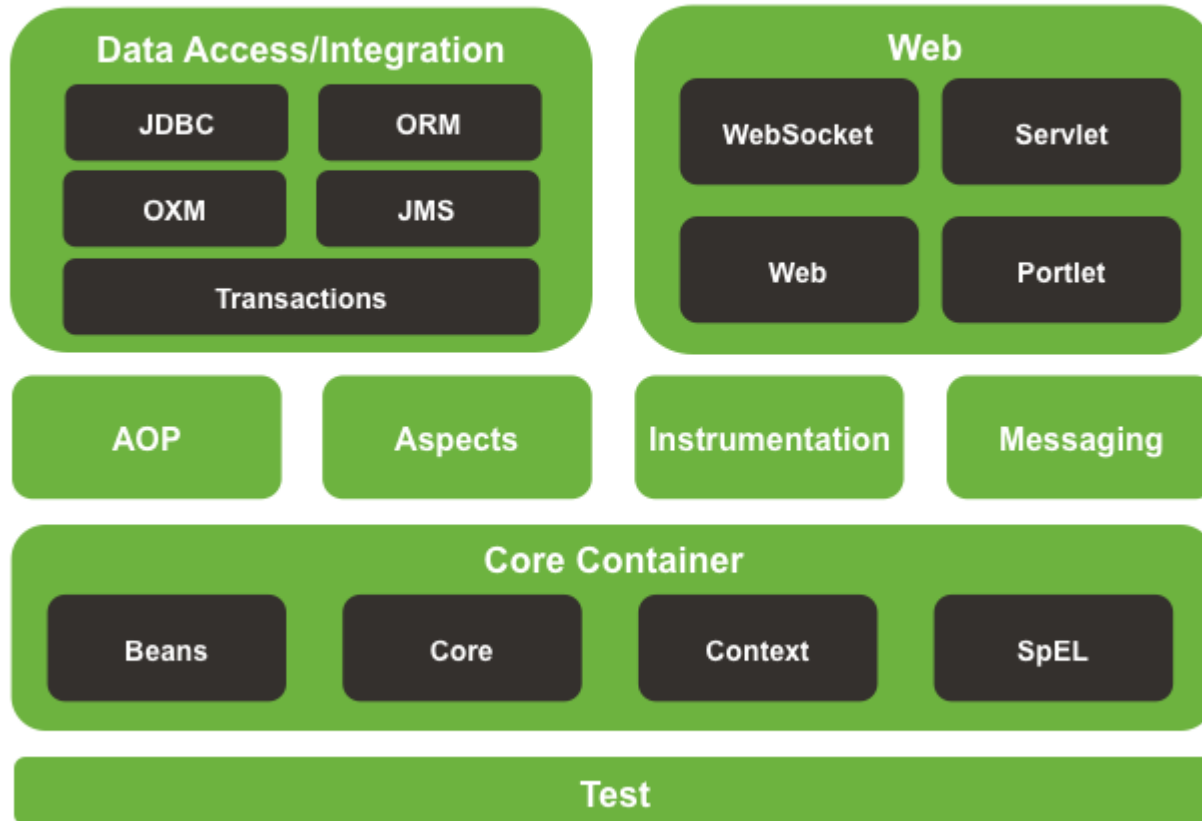
- **개발자들이 개발하고자 하는 애플리케이션 로직 개발에만 집중할 수 있음**
 - 기술에 대한 접근 방식이 일관성이 없거나, 특정 환경에 종속적이지 않음
 - 따라서 실행 로직의 기능이 변경되는 것이 아니라 서버 등의 실행 환경이 바뀌고 적용되는 조건이 바뀐다 해도 코드까지 수정할 필요가 없음
- **개발이 단순해짐**
 - Spring의 의존 관계, 트랜잭션등의 설정 방법에 대한 지식을 습득한 후에는 설정 적용 기술만으로도 Enterprise 개발의 기술적인 복잡함과 그에 따른 수고를 제거 가능
- **POJO 방식의 기술 사용이 가능**
 - 특정 규약 및 환경에 종속되지 않은 일반 자바 클래스를 지원하므로 컨테이너에 의존적인 코드를 추가하지 않아도 애플리케이션을 개발 할 수 있음
 - 개발후의 테스트도 쉽고 빠르게 할 수 있음

■ 스프링 프레임워크의 모듈

- Spring4.0은 20여개의 모듈로 구성



Spring Framework Runtime



■ 스프링 프레임워크의 설치와 모듈구성

- **MAVEN 빌드도구를 이용하기**
 - 의존 관계의 library들도 자동 다운로드

<http://repo.springsource.org/release>

<https://mvnrepository.com/>

■ STS 다운로드

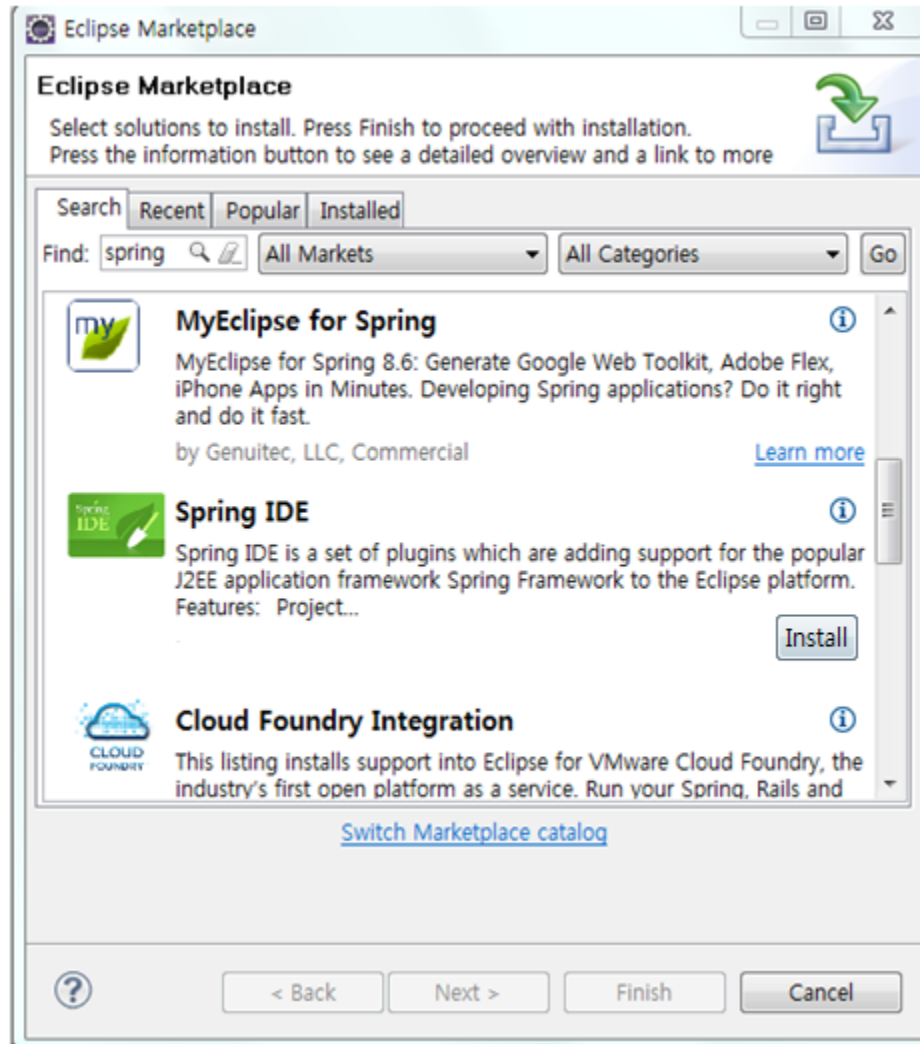
- <http://spring.io/tools>

- Spring Tool Suite는 Spring 응용 프로그램 개발을 위해 사용자 정의 된 Eclipse 기반 개발 환경입니다.

Pivotal tc Server, Pivotal Cloud Foundry, Git, **Maven**, AspectJ 등의 통합을 포함하여 Spring 애플리케이션을 구현, 디버깅, 실행 및 배포 할 수 있는 환경을 제공합니다.

■ 스프링 프레임워크의 설치와 모듈구성

- Spring IDE 설치



■ 스프링 프레임워크의 설치와 모듈구성

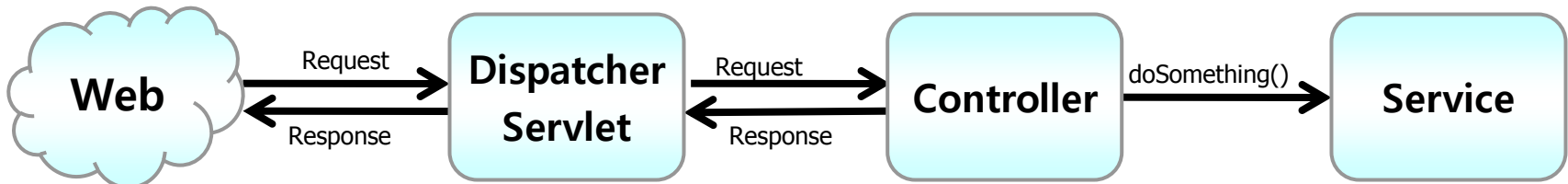
- Spring IDE 설치



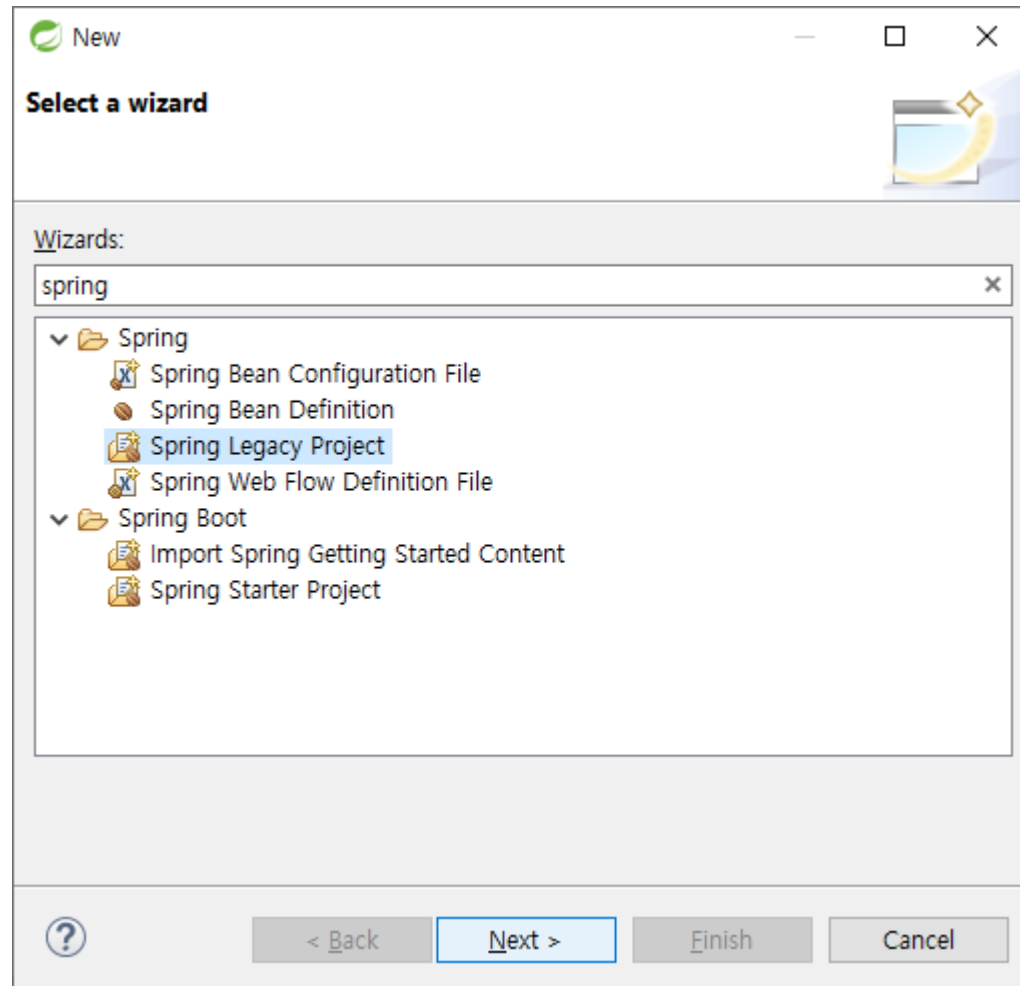
■ 스프링 프레임워크의 설치와 모듈구성

- Spring Project 를 위한 구성
 - Spring Framework library
 - 설정 메타정보 파일
 - Spring은 설정 메타데이터 정보를 필요로 함
 - 이 설정 메타데이터는 Spring 컨테이너에 객체 생성 및 관계 설정 내용을 XML 또는 properties[프로퍼티] 파일, 소스코드 애노테이션과 같은 외부 리소스로 작성
 - 자바 소스들
 - Spring 빈
 - Spring 빈 사용 클래스들
 - 이외의 자바 클래스들

- Spring을 이용한 웹 어플리케이션 작성
 1. Spring 기본 환경 설정
 2. 컨트롤러 작성
 3. 컨텍스트 설정 파일에 컨트롤러 설정
([ServletName]-servlet.xml)
 4. 컨트롤러와 JSP의 연결 위해 View Resolver 설정
 5. JSP 코드 작성
 6. 실행



- Project 생성



- Project 생성

New Spring Legacy Project

Spring Legacy Project

Click 'Next' to load the template contents.

Project name:

☒ Use default location

Location: [Browse...](#)

Select Spring version:

Templates:

- Simple Projects
 - Simple Java
 - Simple Spring Maven
 - Simple Spring Web Maven
- Batch
- GemFire
- Integration
- Persistence
- Simple Spring Utility Project
- Spring MVC Project**

requires downloading [Configure templates...](#) [Refresh](#)

Description:
A new Spring MVC web application development project

URL: <https://dist.springsource.com/release/STS/help/org.springframework.templates.mvc-3.2.2.zip>

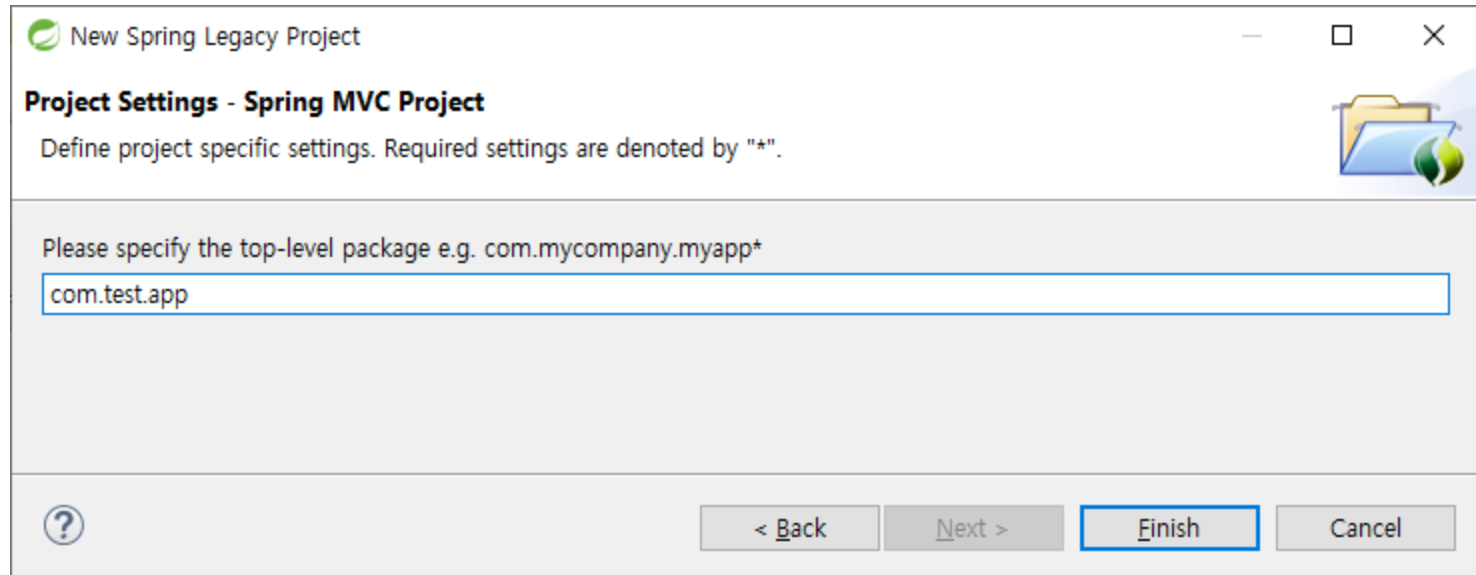
Working sets

☐ Add project to working sets [New...](#)

Working sets: [Select...](#)

[? < Back](#) [Next >](#) [Finish](#) [Cancel](#)

- Project 생성



New Spring Legacy Project

Project Settings - Spring MVC Project

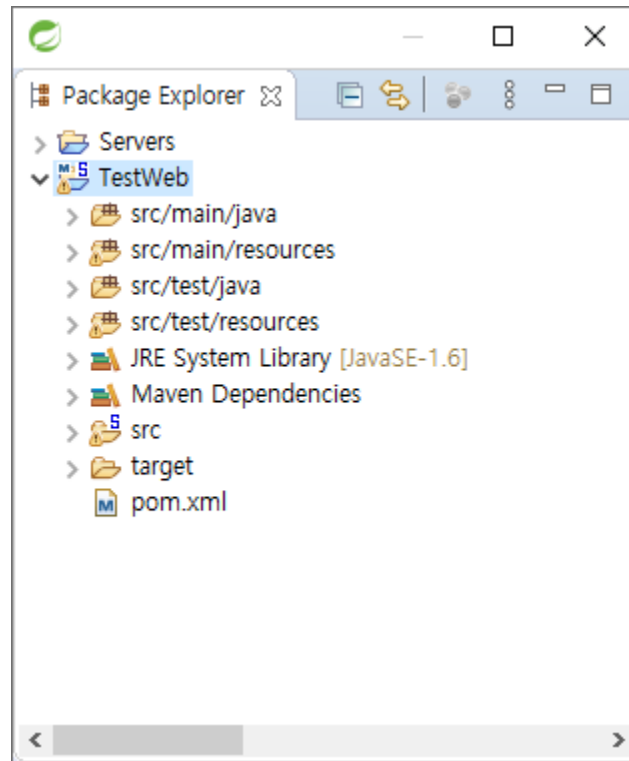
Define project specific settings. Required settings are denoted by "*".

Please specify the top-level package e.g. com.mycompany.myapp*

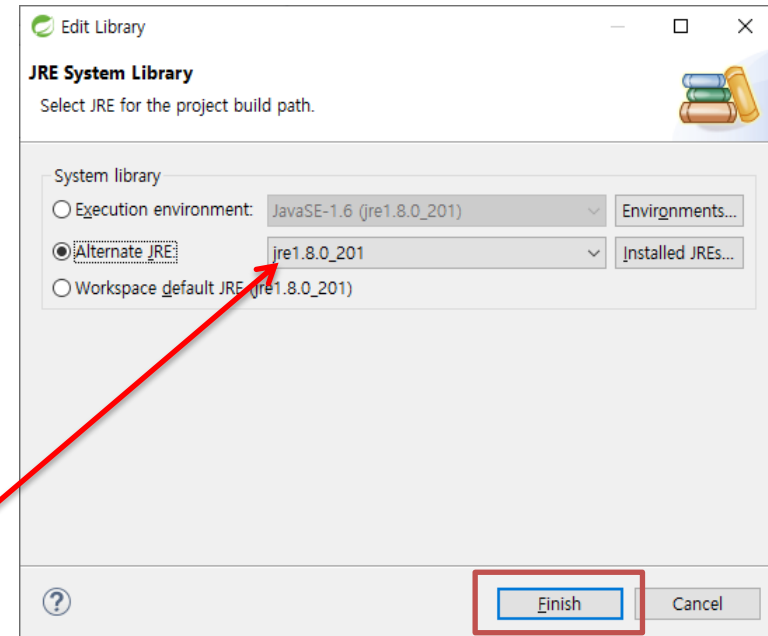
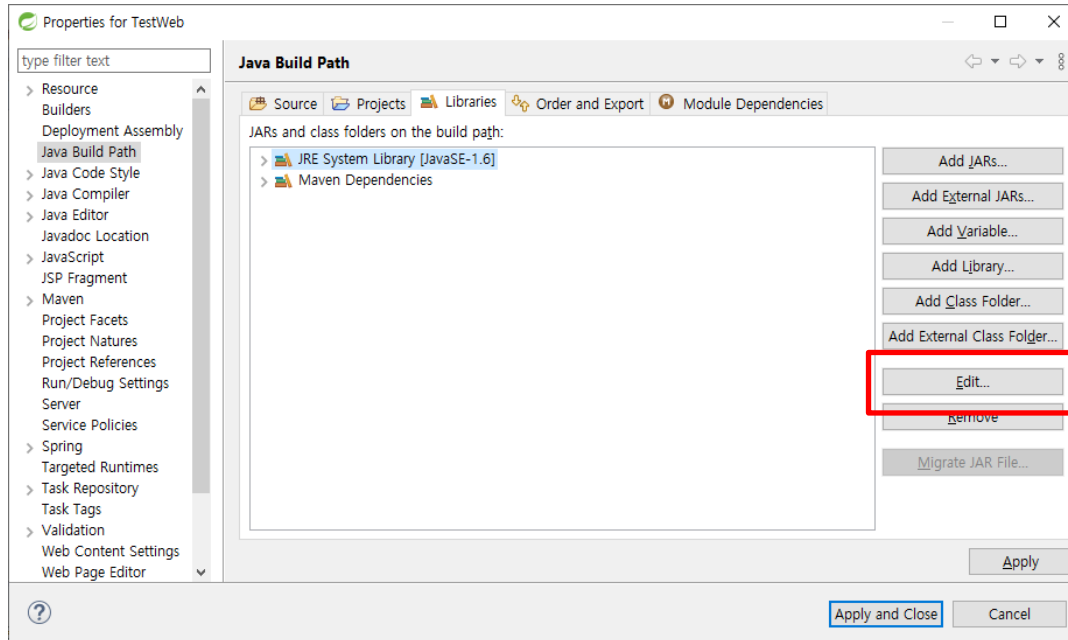
com.test.app

? < Back Next > Finish Cancel

- Project 생성



- Project 생성



- Project 생성

Properties for TestWeb

type filter text

- > Resource
- Builders
- Deployment Assembly
- Java Build Path
- > Java Code Style
- > Java Compiler
- > Java Editor
- Javadoc Location
- JavaScript
- JSP Fragment
- > Maven
- Project Facets**
- Project Natures
- Project References
- Run/Debug Settings
- Server
- Service Policies
- > Spring
- Targeted Runtimes
- > Task Repository
- Task Tags
- > Validation
- Web Content Settings
- Web Page Editor
- Web Project Settings
- WikiText
- > XDoclet

Project Facets

Configuration: <custom> [Save As... Delete]

| Project Facet | Version |
|---|------------|
| > <input type="checkbox"/> Axis2 Web Services | |
| <input type="checkbox"/> Cloud Foundry Standalone Application | 1.0 |
| <input type="checkbox"/> CXF 2.x Web Services | 1.0 |
| <input checked="" type="checkbox"/> Dynamic Web Module | 2.5 |
| <input checked="" type="checkbox"/> Java | 1.6 |
| <input type="checkbox"/> JavaScript | 1.0 |
| <input type="checkbox"/> JavaServer Faces | 2.3 |
| <input type="checkbox"/> JAX-RS (REST Web Services) | 1.1 |
| <input type="checkbox"/> WebDoclet (XDoclet) | 1.2.3 |

Details | **Runtimes**

Select a project facet on the left to view information about it.

Dynamic Web Module : 3.1로 변경
Java : 1.8 로 변경

[Revert] **[Apply]**

[Apply and Close] [Cancel]

- Project 생성

The screenshot shows the Eclipse IDE interface. On the left, the Package Explorer displays the project structure for 'TestWeb', including 'src/main/java', 'src/main/resources', 'src/test/java', 'src/test/resources', 'JRE System Library [JavaSE-1.6]', 'Maven Dependencies', 'src', 'target', and 'pom.xml'. The 'pom.xml' file is highlighted with a red box. An arrow points from this box to the main editor window. The main editor window displays the 'TestWeb/pom.xml' file content. The XML content is as follows:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
4   <modelVersion>4.0.0</modelVersion>
5   <groupId>com.test</groupId>
6   <artifactId>app</artifactId>
7   <name>TestWeb</name>
8   <packaging>war</packaging>
9   <version>1.0.0-BUILD-SNAPSHOT</version>
10  <properties>
11    <java-version>1.6</java-version>
12    <org.springframework-version>3.1.1.RELEASE</org.springframework-version>
13    <org.aspectj-version>1.6.10</org.aspectj-version>
14    <org.slf4j-version>1.6.6</org.slf4j-version>
15  </properties>
16  <dependencies>
17    <!-- Spring -->
```

Red boxes highlight the 'pom.xml' file in the Package Explorer, the 'java-version' property in the pom.xml, and the 'springframework-version' property in the pom.xml. Arrows point from these boxes to the text below.

Java 버전 : 1.8 로 변경
spring 버전은 차후 변경

• Project 생성

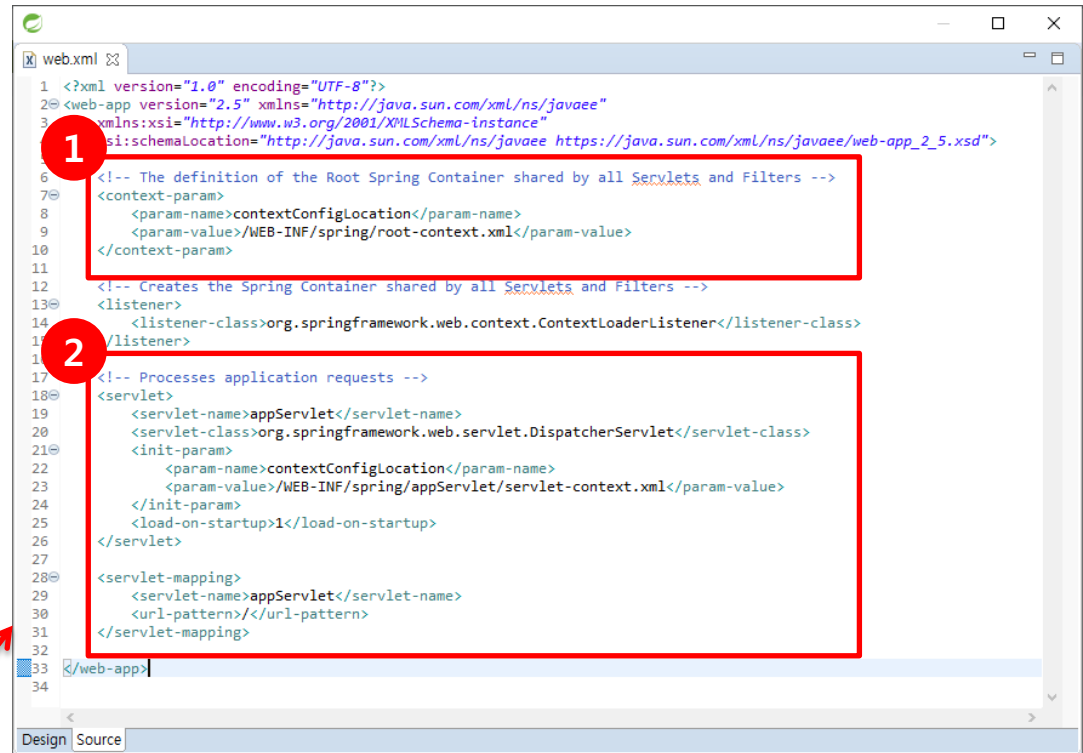
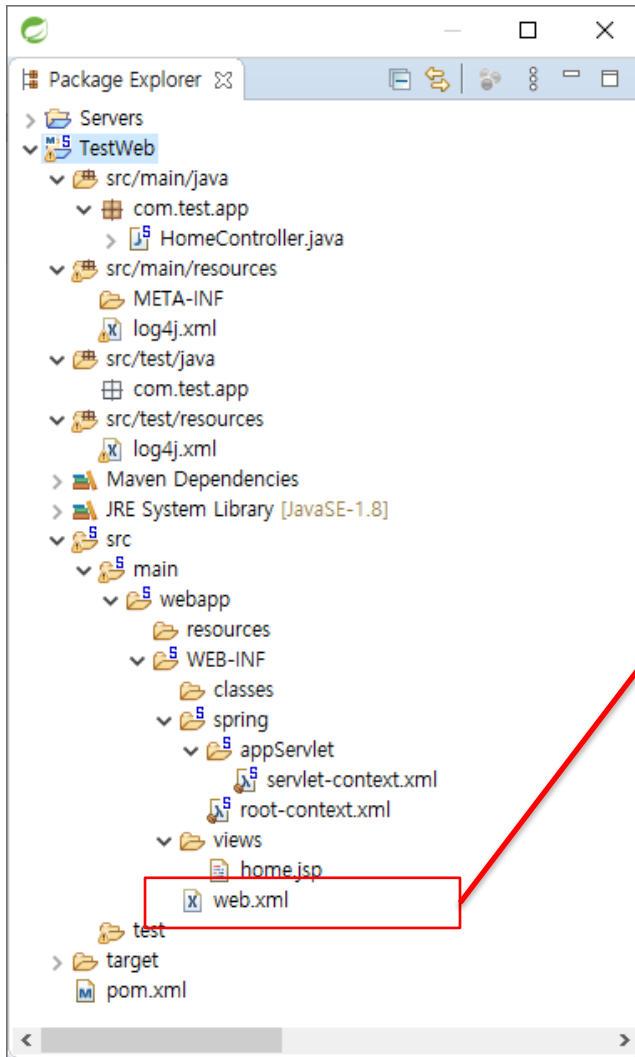
The screenshot shows the Package Explorer for a project named 'TestWeb'. The structure is as follows:

- src/main/java**
 - com.test.app
 - HomeController.java
- src/main/resources**
 - META-INF
 - log4j.xml
- src/test/java**
 - com.test.app
- src/test/resources**
 - log4j.xml
- Maven Dependencies**
- JRE System Library [JavaSE-1.8]**
- src**
 - main
 - webapp
 - resources
 - WEB-INF
 - classes
 - spring
 - appServlet
 - servlet-context.xml
 - root-context.xml
 - views
 - home.jsp
 - test
 - target
 - pom.xml

Annotations with arrows pointing to specific parts of the structure:

- src/main/java**: java 파일 소스가 존재하는 위치
- src/main/resources**: java 소스에서 사용할 자원의 위치
- src/test/java**: 단위 테스트(junit)을 사용할 java 소스코드 파일들의 위치
- src/test/resources**: 단위 테스트(junit)을 사용할 자원 파일들의 위치
- resources** (under webapp): 웹 서비스에 사용할 정적 자원들을 저장하는 폴더 : css, js, html, image
- WEB-INF**: spring Framework 설정 파일
- appServlet**: @MVC 에서 사용할 view (jsp) 파일
- views**: maven 빌드 도구의 설정 파일
- target/pom.xml**: maven 빌드 도구의 설정 파일

- Project 생성

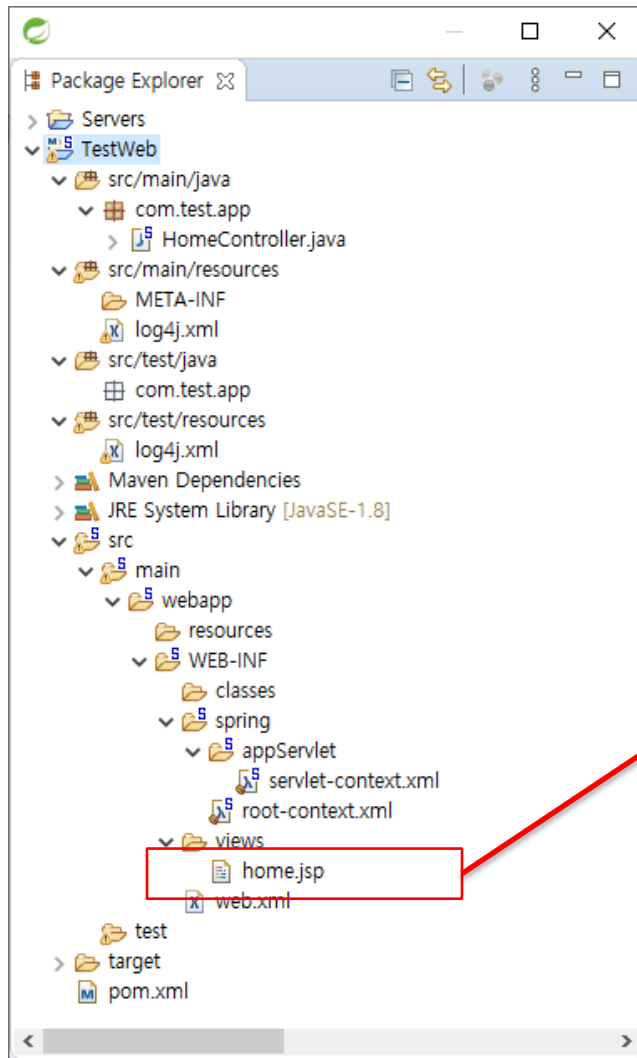


1 Spring Framework 설정 파일

2 Spring Framework DispatcherServlet front controller Servlet 설정

HelloWorld

- Project 생성



- Project 생성

