

# What Is Spring?

Spring is the most popular application development framework for enterprise Java™. Millions of developers use Spring to create high performing, easily testable, reusable code without any lock-in.

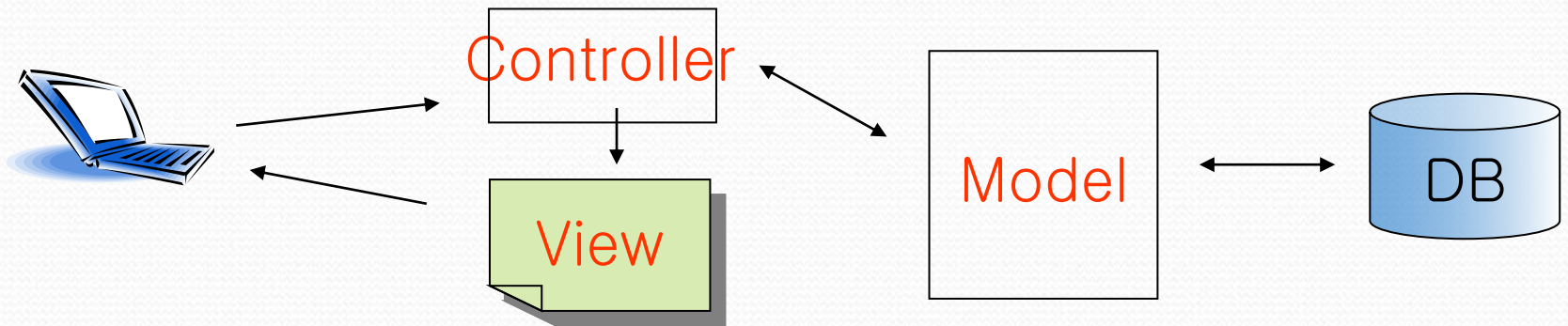
HBI PYK

## ■ MVC 디자인 패턴

### ■ 정의

- 전통적인 GUI 애플리케이션을 구현할 때 사용되는 디자인 패턴이다.
- 사용자의 입력을 받아서 처리하는 부분과, 결과를 사용자에게 보여주는 부분을 완전 분리하여 관리하는 형태의 설계기법이다.

### ■ MVC Architecture

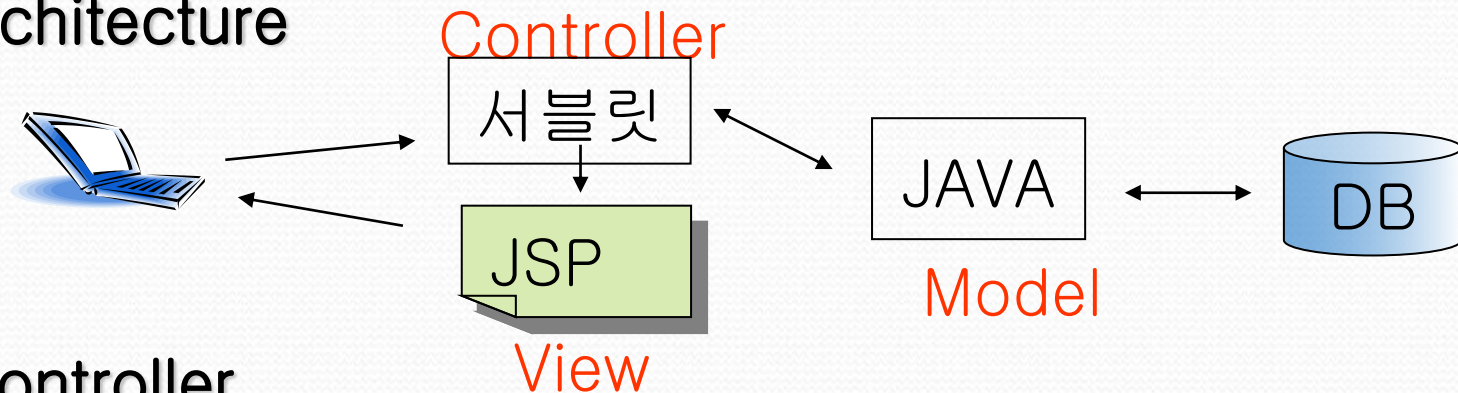


### ■ 특징

- 처리작업의 분리로 인해 유지보수와 확장이 용이하다.
- 각 컴포넌트의 재 사용성이 높아진다.
- 초기 비용비용이 많이 들지만 유지,보수가 효과적이다.
- 웹 애플리케이션을 구현할 때 일반적으로 많이 사용하는 패턴이다.

## ■ 웹 애플리케이션의 MVC 디자인 패턴

### ■ Architecture



### ■ Controller

- 서블릿이 Controller 역할을 담당한다.
- Controller 역할
  - 클라이언트의 요청을 분석한다.
  - 분석된 요구사항을 바탕으로 필요한 Model을 호출한다.
  - 처리결과를 보여주기 위한 뷰파일을 선택한다.

### ■ View

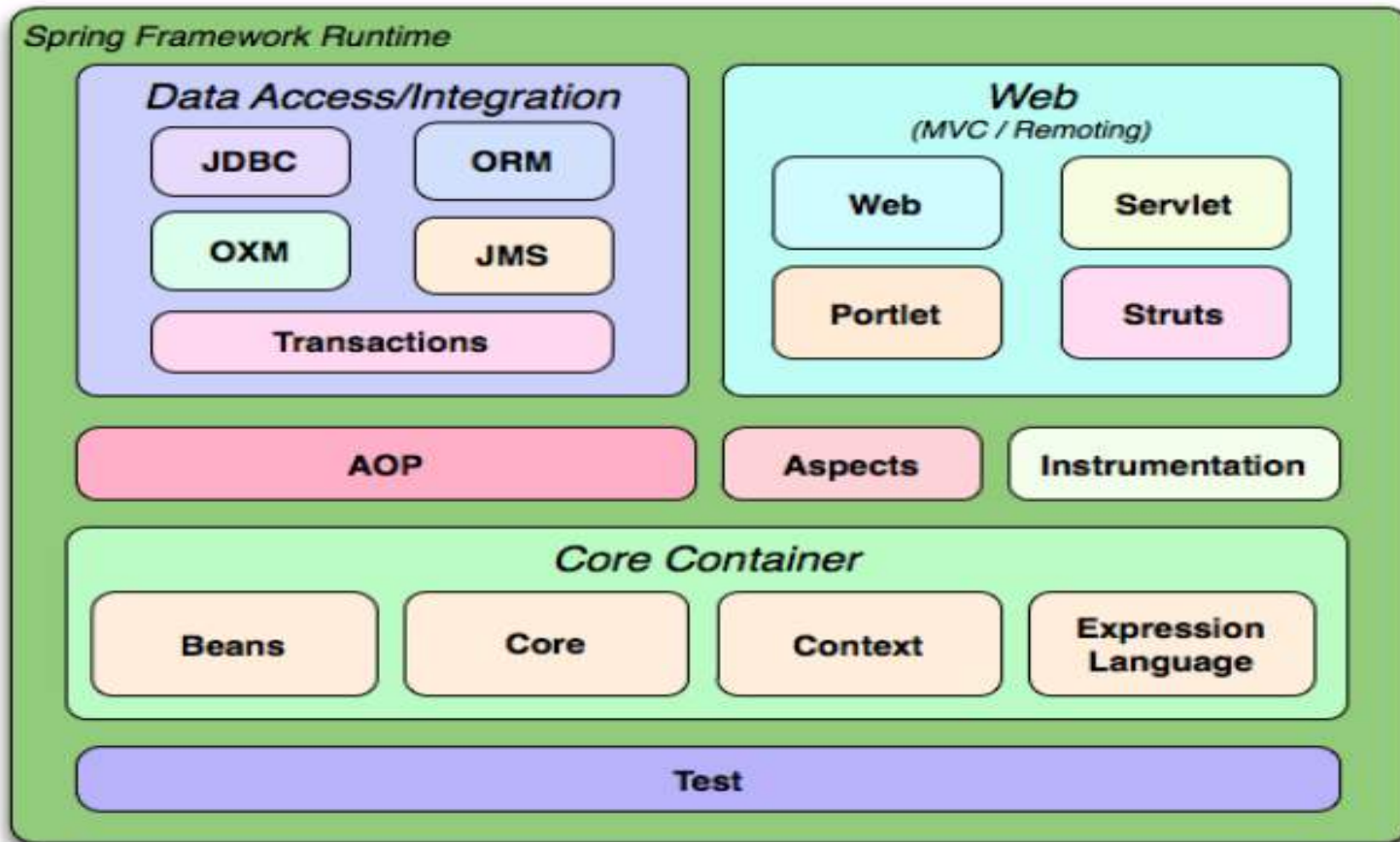
- 주로 JSP 가 View 역할을 담당한다.

### ■ Model

- DB연동과 같은 비즈니스 로직을 처리한다.

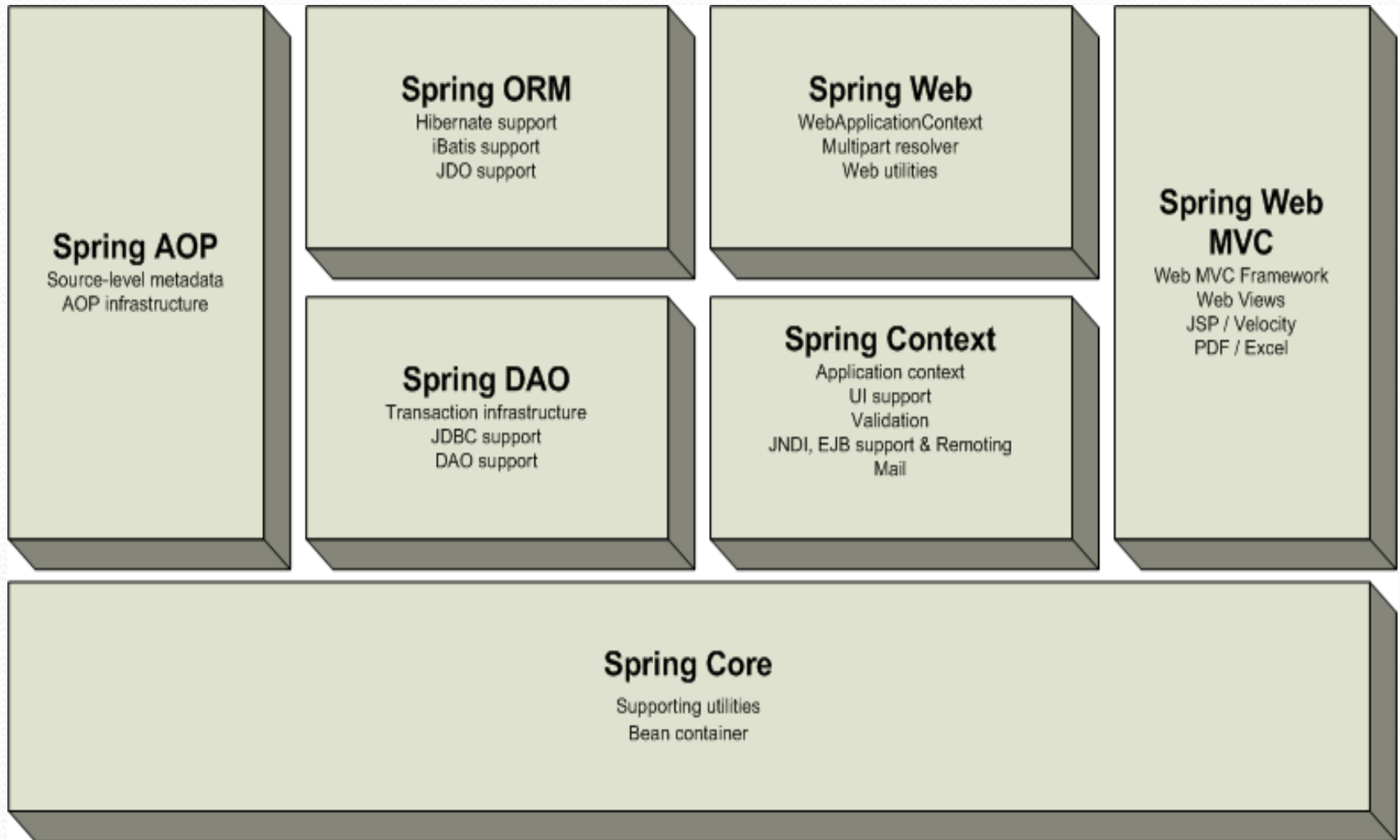


# Spring framework Overview

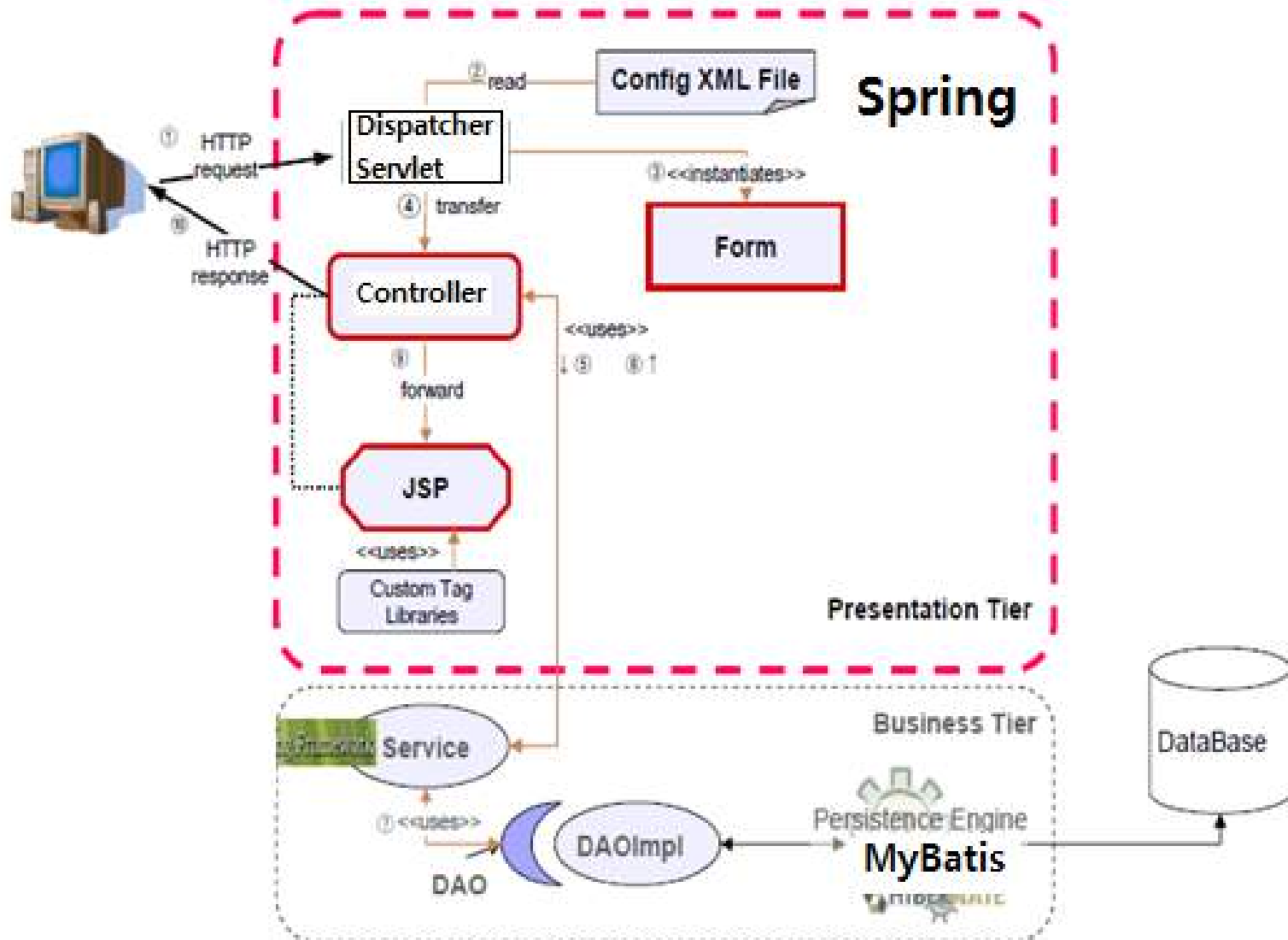


Overview of the Spring Framework

# SpringFramework의 구성 모듈



# Open Source Framework Architecture





# Package의 기능

- Core Package
  - Framework의 가장 기본적인 부분.
  - bean container를 기능적으로 관리하는 것을 허용하는 의존성 삽입(DI 혹은 IoC) 기능을 제공. 기본적으로 Spring의 모든 빈은 Singleton이지만 이를 제거한 Factory Pattern도 제공.
- Context Package
  - bean에 접근하기 위한 방법을 제공, Context 생성을 하기 위한 지원을 추가해준다.
- DAO Package
  - 반복적인 코드를 추상화하여 Database 접근 코드를 깨끗하고 간결하게 유지해준다.
  - 추가적으로 AOP모듈을 사용하여 Transaction Service를 제공한다.
- ORM Package
  - JDO, Hibernate, iBATIS를 포함하는 인기있는 ORM(object/relational mapping) API를 위한 통합 레이어를 제공한다. 이에 대해서도 Transaction 관리를 지원한다.
- AOP Package
  - AOP Programming을 위한 풍부한 기능을 제공하는 Package
- Web Package
  - 파일업로드를 위한 multipart 요청의 처리, servlet listener를 사용한 context init, applicationContext와 같은 web기반 통합기능들을 제공, Webwork나 Struts에 대한 지원도 포함한다.
- Web MVC Package
  - Web application을 위한 Model-View-Controller 구현 물을 제공한다.

# Spring Framework 기본 개념

## (1) POJO (Plain Old Java Object)

다른 클래스를 상속받아서 만들어야 하는 클래스가 아닌 순수한 형태의 자바 클래스

## (2) IoC(Inversion of Control)

제어의 역전이라는 뜻으로 개발자가 직접 객체를 언제 생성하고 없앨지 결정하는 것이 아니라 컨테이너에게 맡긴다는 뜻이다.

POJO 객체의 생성에서 생명주기의 관리까지를 IoC Container 에 담당시킴으로써(XML 이용) 개발에 있어서 편의성과 재사용성의 극대화를 추구하는 개념

## (3) AOP (Aspect Oriented Programming)

관점 지향 프로그래밍이란 뜻으로 기존의 객체지향언어에 의해 추구된 모듈화에 따라 많아진 중복된 코드라든지 공통된 처리에 관련한 것들을 관점으로 뽑아내어 공통으로 처리하는 방식



# IoC (Inversion of Control)

- Bean 의 관리

- 1) Spring 에서 BeanFactory가 기본 IoC 컨테이너이고, 보다 향상된 형태로 ApplicationContext라는 컨테이너도 지원하고 있다.
- 2) IoC 컨테이너에 의해서 Bean 의 생명주기가 관리된다.
- 3) 이렇게 관리되는 Bean 은 POJO 타입이고 싱글톤이다.
- 4) XML 설정을 통해서 각각의 Bean 들을 묶어주는데, 이를 wiring라고 한다.
- 5) 묶인 Bean 을 원하는 곳에 적용하는 것을 DI(의존성 주입) 이라고 부른다.

# Spring 실습 환경 설정 및 예제 실행 방법

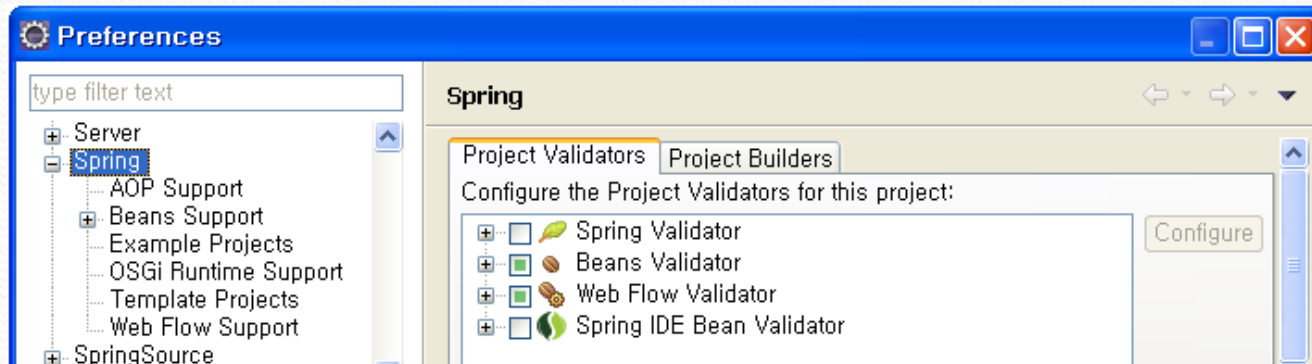
## 방법1 :

- 1) java jdk 설치
- 2) eclipse 설치
- 3) eclipse의 메뉴 중 help -> Eclipse Marketplace를 선택한 후 spring을 검색하여 STS를 다운 받는다.



다운 완료 후 이클립스 restart를 물으면 yes하면 된다.

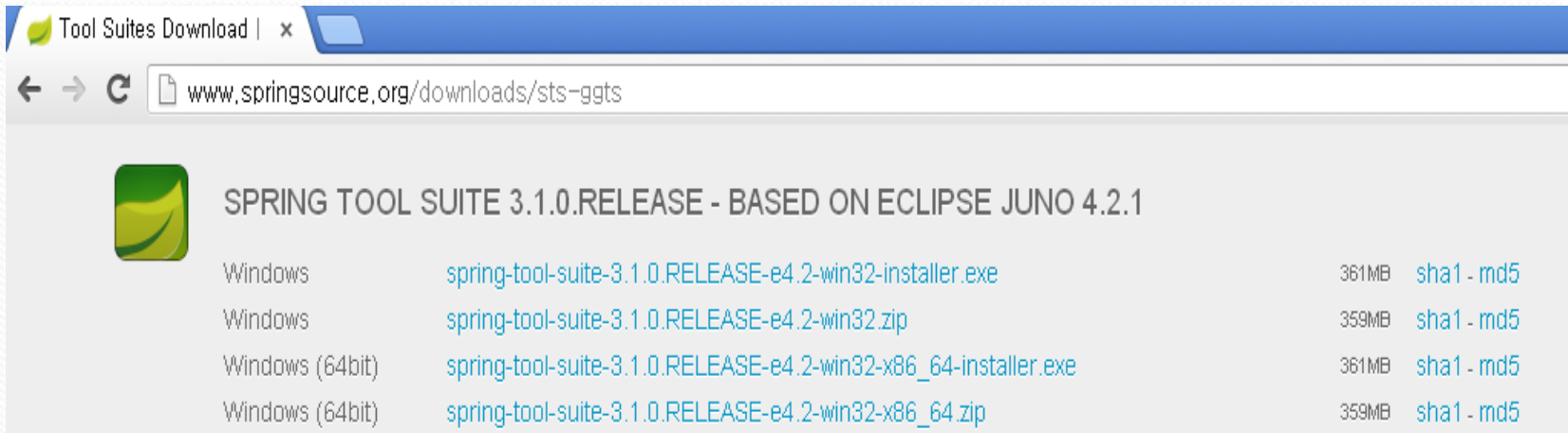
제대로 spring이 이클립스에 셋팅되면 아래와 같은 화면을 볼 수 있다.





## 방법2:

- 1) java jdk 설치
- 2) <http://www.springsource.org> 에서



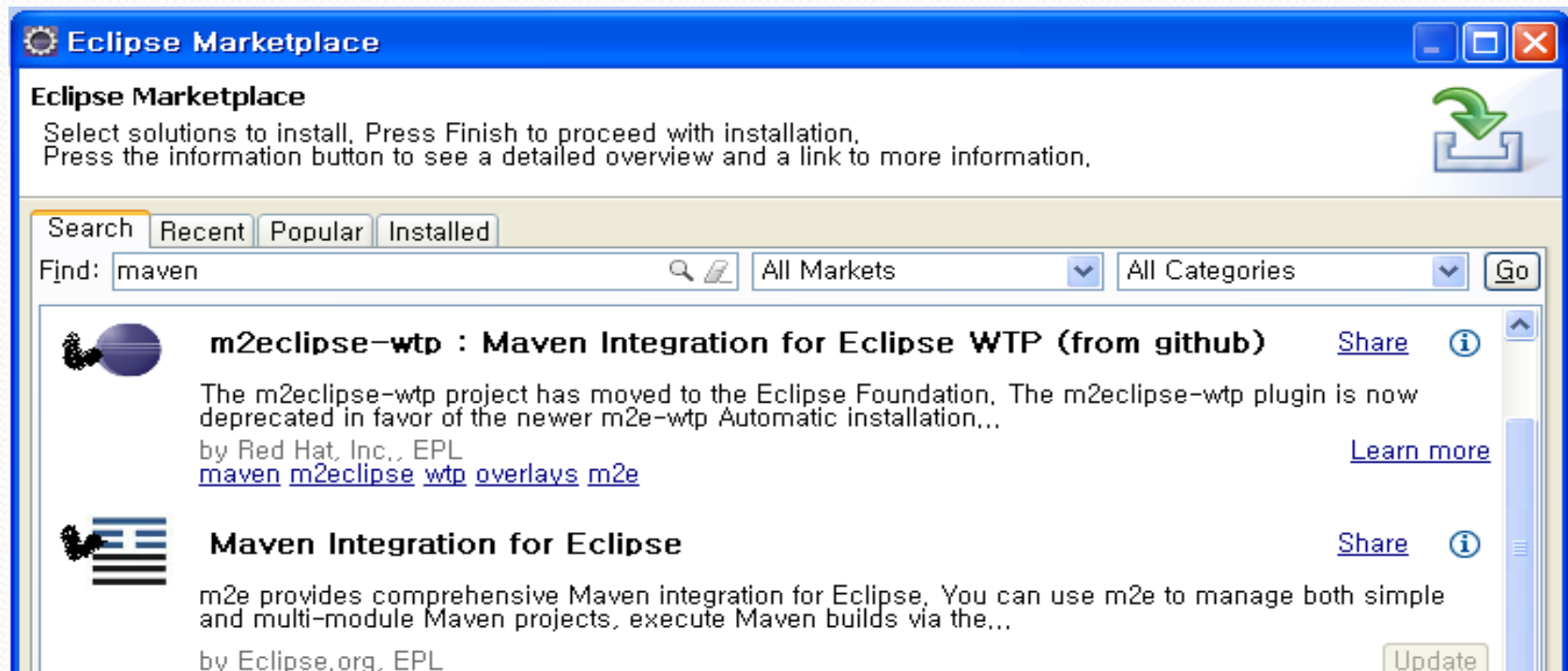
를 다운 받아 압축을 풀고 실행하면 이클립스가 포함된 스프링 실행환경이 완성된다.



# Maven 설치 후 사용하기

**Maven** 은 java project library를 관리하고 실행파일의 build를 책임지는 framework다.

**Maven framework**를 사용하려면 eclipse의 메뉴 중 help -> Eclipse Marketplace를 선택한 후 maven을 검색하여 다운 받는다.



## Spring의 DI란?

### 생성자 방식

- 의존할 객체를 통해 객체를 생성함으로써 의존관계를 형성하는 방법

#### 객체생성(생성자를 통한 초기화) 싱글톤 패턴

```
<bean id="userImpl" class="di.demo1.UserImpl">  
  <constructor-arg value="30"></constructor-arg>  
  <constructor-arg value="+"></constructor-arg>  
  <constructor-arg value="15"></constructor-arg>  
</bean>
```

Bean 태그를 활용하여 명시한다.

생성자의 인자 순서대로 나열

객체명 : userImpl

생성대상 : di.demo1.UserImpl 클래스에서  
인자가 3개인 생성자를 통해 생성

#### 객체 생성 및 의존관계 설정(생성자) 방법

```
<bean id="userService1" class="di.demo1.UserService">  
  <constructor-arg>-->  
    <ref bean="userImpl"/>  
  </constructor-arg>  
</bean>-->
```

위에서 생성된 객체 userImpl

객체명 : userService1

생성대상 : di.demo1.UserService 클래스에서  
userImpl의 객체를 통해 객체 생성



## 프로퍼티 방식

- setter 를 통해 값을 전달받아 객체를 생성한다.

객체생성(프로퍼티 설정 : 반드시 setter가 필요하다.)

Constructor가 없다.  
Default 생성자 호출

Bean 태그를 활용하여 명시한다

```
<bean id="userImpl" class="di.demo1.UserImpl">
  <property name="num1" value="15"></property>
  <property name="num2" value="20"></property>
  <property name="oper" value="*"></property>
</bean>
```

Property 태그를 통해 setter에 값을 전달

객체명 : userImpl

생성대상 : di.demo1.UserImpl 클래스에서  
Setter를 통해 지정된 값을 활용하여 객체를 생성

```
<bean id="userService1" class="di.demo1.UserService">
  <property name="user" ref="userImpl"/>
</bean>
```

Setter를 통해 값 전달

위에서 생성된 객체 userImpl

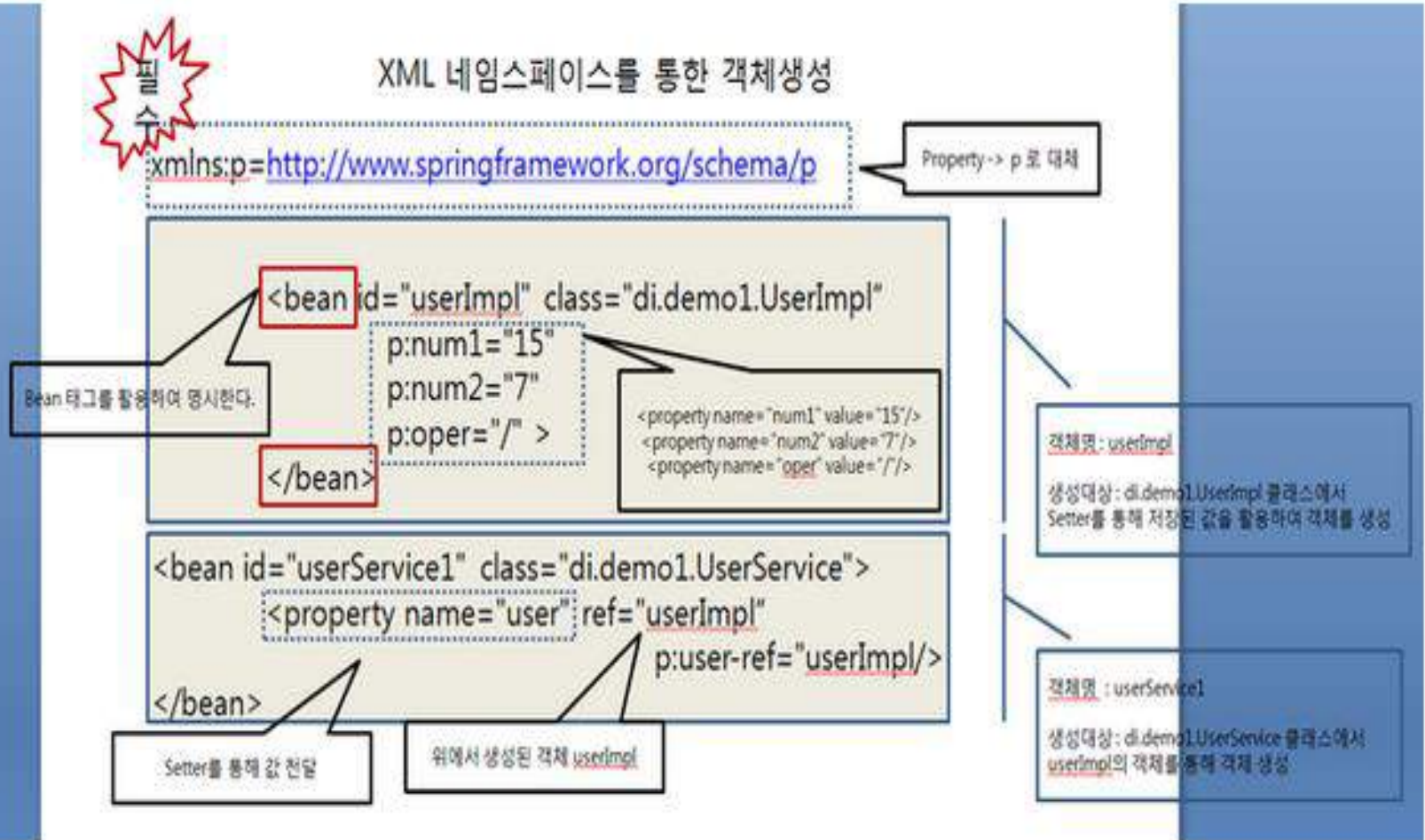
객체명 : userService1

생성대상 : di.demo1.UserService 클래스에서  
userImpl의 객체를 통해 객체 생성



## XML Name Space 방식

- 프로퍼티 방식에서 <property> 태그를 사용하지 않고 간단한 네이밍을 통해 <property> 태그를 대체한다.



# Bean XML 설정

- 1) scope 설정 : IoC Container 에 의해 관리되는 Bean의 기본 범위는 singleton이다.  
이걸 scope 속성으로 다르게 지정하는 것이 가능하다.
  - <1> singleton(기본값) : IoC Container 내에서 유일한 객체로 생성하는 범위
  - <2> prototype : 호출때마다 객체가 따로 생성되는 방식
  - <3> request : 요청시마다 생성
  - <4> session : 세션마다 하나씩 생성
- 2) 객체 생성 메소드 지정 : Singleton 팩토리 클래스에는 생성자가 private 이므로 따로 객체 생성 메소드를 지정시 factory-method 속성으로 메소드명을 지정해줄 수 있다.



# 스프링 라이프 사이클

## □ 특징:

- 스프링 컨테이너에 저장되는 빈 객체는 최소한 생성, 초기화, 소멸의 라이프 사이클을 갖게 됨.
- 스프링은 빈 객체의 생성, 초기화, 소멸뿐만 아니라 추가적인 단계를 제공하고 있으며, 이를 통해 라이프 사이클에 따른 빈 객체의 상태를 정교하게 제어.

## 스프링 라이프 사이클 순서 -----

- 1.빈 객체 생성
- 2.`BeanNameAware.setBeanName()` - 빈 이름 설정
- 3.`BeanFactoryAware.setBeanFactory()` - 빈 팩토리 설정
- 4.`BeanPostProcessor`의 초기화 전처리 - 초기화 전에 빈에 대한 처리
- 5.`InitializingBean.afterPropertiesSet()` - property설정(setter) 완료 후 실행
- 6.커스텀 `init-method` - 초기화 메서드
- 7.`BeanPostProcessor`의 초기화 후처리 - 초기화 후에 빈에 대한 처리
- 8.빈 객체 사용
- 9.`DisposableBean.destory()` - 객체가 소멸되기 전에 수행
- 10.커스텀 `destory-method` - 객체가 소멸되기 전에 사용자가 정의한 것 수행



# DI (Dependency Injection)

의존성 주입이란 뜻으로 객체를 생성해서 필요로 하는 곳에 넣어주는 것을 뜻한다.

- 1) Constructor Injection (생성자 주입)  
생성자에 필요 객체를 선언하고 멤버변수(Field)로써 객체를 사용하는 것을 말한다.  
객체 생성시 자동으로 IoC 컨테이너가 관련 객체를 주입시켜준다.
- 2) Setter Injection (세터 메소드 이용 주입)  
생성자 주입과 방식은 비슷하나 setter 메소드를 이용하는 점이 차이점이다.

# DI와 빈 객체를 위한 Annotation 설정

## 종류

<1> @Required : 필수 프로퍼티를 세팅할 때 사용

<2> @Autowired : 자동으로 똑같은 이름의 빈을 찾아서 주입해준다. (타입으로 찾음)

<3> @Qualifier("name") : bean 설정에 <qualifier> 태그를 이용한 수식어로 특정 빈 지정 시 사용

<4> @Resource(name="testBean") : 특정 bean 이름을 지정 (JDK6, JEE5 이상)

@Resource 어노테이션은 어플리케이션에서 필요로 하는 자원을 자동 연결할 때 사용된다. 프로퍼티 및 설정메서드 (Setter, Getter) 등에 적용시키며 스프링 설정파일에 등록되어있는 빈 객체의 name 속성을 통하여 자동으로 주입된다. (@Autowired 어노테이션이 타입으로 자동 주입을 하는 반면, @Resource 어노테이션은 name 속성을 통해 자동 주입을 실행한다)

<5> @PostConstruct, @PreDestroy : 초기화, 소멸 시 사용될 메소드 지정  
---- 이하는 <context:component-scan /> 태그로 사용 가능

<6> @Component : XML 설정없이 자동 빈 등록 @Component("name") 이름 지정 가능

<7> @Service : @Component 와 현재 다를 바가 없으나 차후 세부화 예정.

<8> @Repository : 퍼시스턴스 영역의 Exception 을 번역해주는 기능 제공

<9> @Controller : Spring MVC 사용시 Controller에 설정

<10> @Scope("prototype") : 싱글톤이 아닌 것으로 scope 지정 시 사용



# Bean 에서 특정 객체를 주입 받기 위한 설정

Bean에서 특정 객체가 필요할 경우 관련 인터페이스를 구현함으로써 객체를 자동으로 주입받을 수 있다.

- 1) BeanFactoryAware : BeanFactory 객체를 제공
- 2) BeanNameAware : Bean의 id값을 제공
- 3) ApplicationContextAware : ApplicationContext를 제공
- 4) MessageSourceAware
- 5) ApplicationEventPublisherAware
- 6) ResourceLoaderAware
- 7) BeanPostProcessor : Bean이 생성될 때 호출되는 메소드를 제공



# AOP (Aspect Oriented Programming)

- 1) Aspect : 관점이란 뜻으로 트랜잭션, 보안, 로깅 처럼 공통적으로 쓰이는 부분들에 개입하는 것을 모듈화한 것.
- 2) Join point : 클래스의 객체 생성 시점, 메소드 호출 시점, Exception 발생 시점 등 AOP가 개입되는 시점을 말한다.
- 3) Advice : 특정한 join point에서 interceptor 형태로써 다양한 기능을 제공하는 것을 뜻함.
  - <1> Before advice : 메소드 호출전의 기능 제공
  - <2> After returning advice : 모든 메소드 실행이 다 끝나고 난 뒤 기능 제공
  - <3> After throwing advice : 익셉션이 던져졌을 때 기능 제공
  - <4> After (finally) advice : 익셉션 발생이 됐던 안됐던 간에 기능 제공
  - <5> Around advice : 가장 강력한 advice 로써 메소드 시작부터 마지막까지 두루 기능을 제공하는 것이 가능
- 4) Pointcut : 정규 표현식이나 AspectJ의 문법 등으로 실제로 적용되는 join point 를 선언.
- 5) Weaving : 실제 aspect 객체와 대상 객체를 컴파일 시나, 로딩 시나, 실행 시에 합쳐주는 것을 뜻한다.
- 6) Target : 실제 advice를 받는 객체로써 대상을 뜻한다.

# Spring에서의 JDBC 처리

<http://static.springsource.org/spring/docs/3.0.x/api/> 참조

JdbcDaoSupport 클래스는 간결한 데이터베이스 소스를 지원하기 위해 **JdbcTemplate** 클래스를 지원하고 있다. JdbcTemplate 클래스는 JDBC 작업을 할 때 정형화된 코드를 내장하고 있기 때문에 반드시 필요한 것들만 전달해 주면 데이터베이스에 연결, PreparedStatement 객체 생성, SQL 실행 등을 담당해 준다.

**JdbcTemplate** 객체를 사용하기 위해서는 getJdbcTemplate()를 호출하면 된다

dataSource 속성을 가지고 있기 때문에 이 클래스를 설정파일에 등록할 때는 dataSource 속성에 dataSource 태그의 참조를 \* 입력해 주면 된다. 즉, **JdbcDaoSupport** 클래스는 JdbcTemplate 객체와 dataSource 객체를 모두 사용할 수 있는 객체가 된다. JdbcTemplate 을 사용하면 Connection, PreparedStatement, ResultSet 등에 대해 스프링이 제어를 해 주기 때문에 개발자는 좀더 간결한 데이터베이스 코드를 사용할 수 있게 된다.



# 스프링 MVC 란?

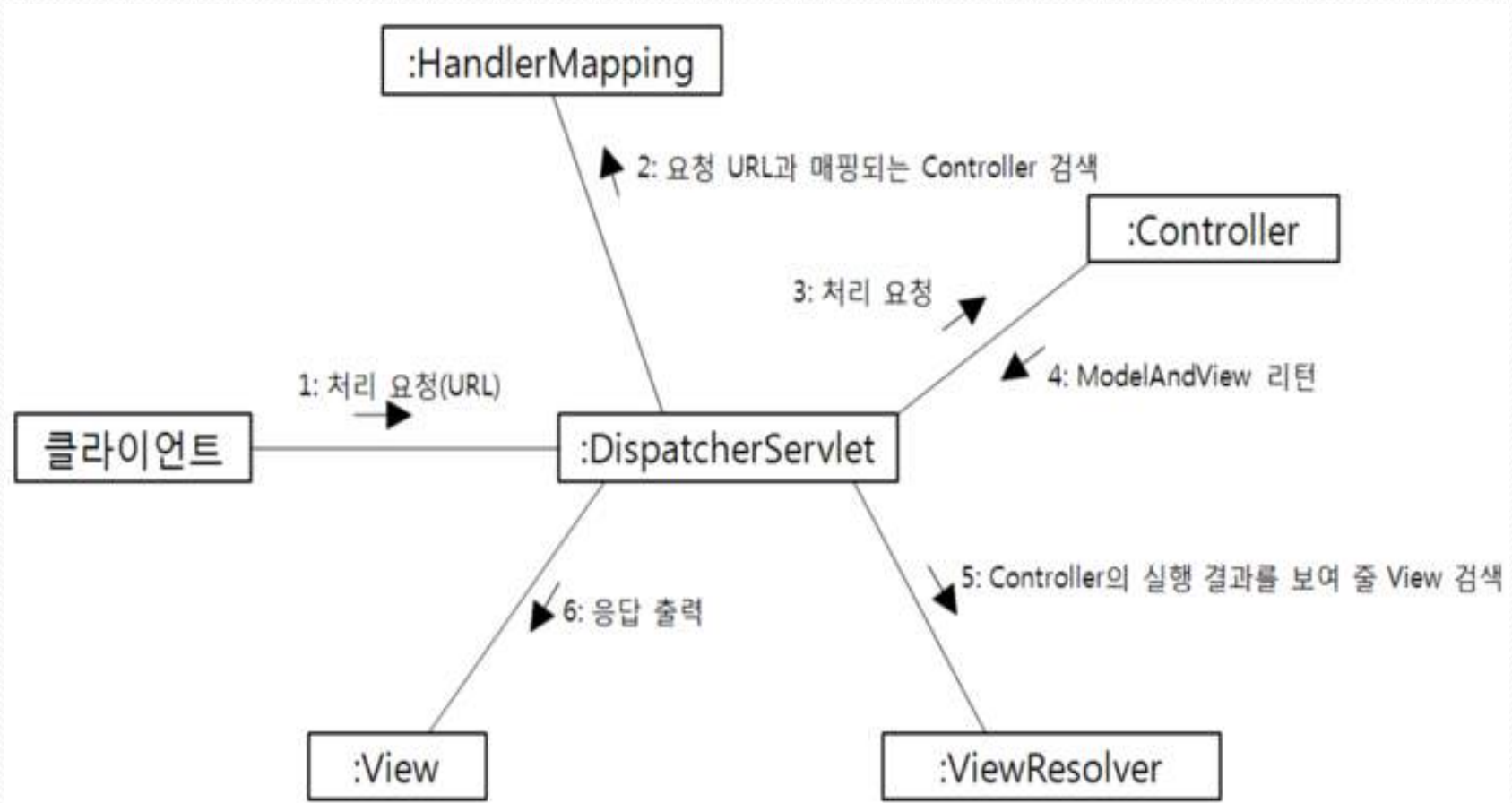
- 스프링이 직접 제공하는 서블릿 기반의 MVC 프레임워크이다. 스프링 서블릿 또는 스프링 MVC라고 부른다.
- 프론트 컨트롤러 역할을 하는 DispatcherServlet을 핵심 엔진으로 사용한다. 스프링이 제공하는 AOP, 트랜잭션 처리,
- DI 등의 기능을 그대로 사용하면서 MVC 패턴의 기반하여 웹 어플리케이션을 개발할 수 있다.



# 스프링 MVC의 구성 요소

- DispatcherServlet : 클라이언트의 요청을 전달받는다. Controller에게 클라이언트의 요청을 전달하고, Controller가 리턴한 결과값을 View에 전달하여 알맞은 응답을 생성하도록 한다.
- HandlerMapping : 클라이언트의 요청 URL을 어떤 Controller가 처리할지를 결정한다.
- Controller : 클라이언트의 요청을 처리한 뒤, 그 결과를 DispatcherServlet에 알려준다.
- ViewResolver : Controller의 처리 결과를 보여줄 View를 결정한다.
- View : Controller의 처리 결과를 보여줄 응답을 생성한다

# 구성 요소 간의 메시지 흐름





# 각 흐름을 간략하게 정리하면...

- 클라이언트의 요청이 DispatcherServlet에 전달된다.
- DispatcherServlet은 HandlerMapping을 사용하여 클라이언트의 요청이 전달될 Controller 객체를 구한다.
- DispatcherServlet은 Controller 객체의 handleRequest() 메소드를 호출하여 클라이언트의 요청을 처리한다.
- Controller.handleRequest() 메소드는 처리 결과 정보를 담은 ModelAndView 객체를 리턴한다.
- DispatcherServlet은 ViewResolver로부터 처리 결과를 보여줄 View를 구한다.
- View는 클라이언트에 전송할 응답을 생성한다.



# 기본적인 Spring MVC 설정 잡기

- 1) web.xml 에 DispatcherServlet 의 설정을 잡는다. (이 곳의 name이 name명-servlet.xml 파일이 설정파일이 된다)
- 2) HandlerMapping 을 이용 URL 과 Bean의 name 속성을 같은 것으로 설정한다.  

```
<bean id="beanName" class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping" />
```

참고) 만약 이 설정을 넣지 않으면 자동으로 이 HandlerMapping 이 적용된다.
- 3) Spring 설정 파일에 name에 URL 을 입력하고 class 에 Controller 를 지정한 뒤 작성한다.
- 4) Controller 에서는 ModelAndView 객체를 생성해서 직접 갈 곳을 지정해준다.

# HandlerMapping 구현 클래스

- 1) SimpleUrlHandlerMapping : 패턴과 컨트롤러 이름을 비교하여, URL 이 패턴에 매칭될 경우 지정 컨트롤러 사용
- 2) BeanNameUrlHandlerMapping(default) : URL 과 일치하는 bean 이름의 컨트롤러 사용
- 3) ControllerClassNameHandlerMapping : URL 과 매칭되는 클래스 이름을 갖는 빈을 컨트롤러 사용
- 4) DefaultAnnotationHandlerMapping : @RequestMapping 어노테이션을 이용 컨트롤러 사용

참고) web.xml에 <url-pattern> 을 '/path/\*' 같이 설정을 할 경우 전체 경로를 다 사용하고 싶으면 HandlerMapping 설정 시 <property name="alwaysUseFullPath" value="true" /> 를 넣어서 설정을 잡는다.

참고2) 복수개의 HandlerMapping 구현 시 property 로 order 를 넣어서 순서를 정해줄 수도 있다.



# Controller 구현 클래스

- 1) Controller, AbstractController : 단순 처리용
- 2) AbstractCommandController : 요청 파라미터를 객체에 자동으로 저장해 주고 검증기능 제공한다.
- 3) SimpleFormController : 폼을 출력하고 폼에 입력한 데이터를 처리할 때 사용
- 4) AbstractWizardFormController : 여러 페이지에 걸쳐 데이터 입력 시 사용
- 5) ParameterizableViewController, UrlFilenameViewController : 단순히 뷰로 바로 전달할 때 사용



# ViewResolver 구현 클래스

- 1) InternalResourceViewResolver(default)  
: JSP 나 tiles(1.x) 연동 위한 View 객체를 리턴
- 2) VelocityViewResolver  
: Velocity 연동을 위한 View 객체를 리턴
- 3) BeanNameViewResolver  
: 뷰 이름과 똑같은 이름의 Bean 객체를 View 로 이용
- 4) UrlBasedViewResolver  
: tiles 2.x 를 쓸 경우 View 로 Tiles viewname 을 이용

# Validator 와 Errors

- 1) Validator 인터페이스를 상속받은 클래스로써 `command(form)`에 들어오는 값들에 대한 검증이 가능하다
- 2) 주요 메소드 (Validator 인터페이스를 상속받는 클래스에서 구현하는 메소드)
  - <1> `boolean supports(Class clazz)` : 인자값으로 들어오는 클래스가 검증대상인지 확인하는 메소드
  - <2> `void validate(Object target, Errors errors)` : 실제 검증 메소드  
(타겟은 `command` 객체이다)
    - a. `ValidationUtils` 클래스를 통해서 데이터의 검증을 한다.  
혹은 `Errors` 를 이용해 에러를 저장한다.
    - b. 이렇게 저장된 오류는 jsp 화면에서 `<form:errors>` 를 이용해 오류를 보여주는 것이 가능하다.
    - c. `Errors` 에는 전체적인 오류를 저장하기 위한 `reject` 메소드와 특정 필드에 대한 오류를 저장하기 위한 `rejectValue` 메소드가 있다.
- 3) `command` 가 쓰이는 `Controller.xml` 설정에 프로퍼티로써 `validator` 를 적용시켜야 한다.
  - ex) `<property name="validator" ref="reboardValidator"></property>`



# Annotation 을 이용한 Controller 셋팅

1) XML 셋팅 <context:component-scan base-package="" />

2) 종류

<1> @Controller : 대상 컨트롤러 클래스의 위에 선언

<2> @RequestMapping : 처리할 주소(value) 및 HTTP METHOD 지정  
ex) @RequestMapping(value="/index.spring",  
method=RequestMethod.GET)

<3> @RequestParam("parameterName") : 요청 파라미터를 인자 값으로  
선언 시 사용

<4> @ModelAttribute("commandName") : 인자값에 선언 시 요청 파라미터  
들을 받는 커맨드 객체로 사용,  
메소드 위에 선언 시 뷰에 세팅될 커맨드객체로 사용

<5> @SessionAttributes("commandName") : 클래스 위에 선언하여 세션에  
커맨드 객체 저장

# Controller 클래스의 메소드 선언

- <1> 인자타입(parameter type)으로 사용 가능한 타입
- a. HttpServletRequest, HttpServletResponse, HttpSession
  - b. java.util.Locale : 현재 지역정보
  - c. InputStream, Reader : 요청 콘텐츠에 직접 접근
  - d. OutputStream, Writer : 응답 콘텐츠로 사용
  - e. @RequestParam 을 이용한 파라미터 명시
  - f. Map, ModelMap : 뷰에 전달할 데이터 저장용
  - g. Command Class : @ModelAttribute 를 사용시 이름이 지정되고, 지정하지 않으면 클래스명이 지정됨(앞글자 소문자)
  - h. Errors, BindingResult : 커맨드 클래스 바로 뒤에 위치해야 함.  
유효성 체크 위해 사용
  - i. SessionStatus : 폼 처리 완료 시 호출하기 위해 사용  
@SessionAttributes 와 같이 사용



## return type으로 사용 가능한 타입

- a. ModelAndView : 일반적인 컨트롤러 메소드의 리턴 객체
- b. Map : 뷰에 전달할 데이터의 집합체,  
view 이름은 요청 URL로 결정  
(RequestToViewNameTranslator)
- c. String : view 이름으로 처리
- d. void : 직접 응답 처리시

# Transaction

- (1) ACID : 트랜잭션에서 추구했던 기본 4요소
- (2) 프로그램적인 트랜잭션
- (3) 선언적인 트랜잭션