JAVA 프레임워크
SPRING 예제 (2)

장안대학교
인터넷정보통신과

고객관리 미니 프로젝트개요

테이블 구성

열(D): 이름 BCARD

PK	이름	데이터 유형	크기	널이 아님	기본값
🔑	BNO	NUMBER		☑	
	BNAME	VARCHAR2	50	☑	
	PHONE	VARCHAR2	15	☑	
	DESCRIPTION	VARCHAR2	1000	☑	

메시지 :

[신규]

Sample No	Title	Phone	Description	수정	삭제
1	양옥경	010-1234-4321	태진아 부인	수정	삭제
2	이현상	010-2234-2321	저현상 동생	수정	삭제
3	김수현	010-3234-3321	드라마 작가	수정	삭제
4	박상현	010-4234-4421	정치가	수정	삭제
5	강양원	010-5234-5421	장안대학교수	수정	삭제

[Submit]

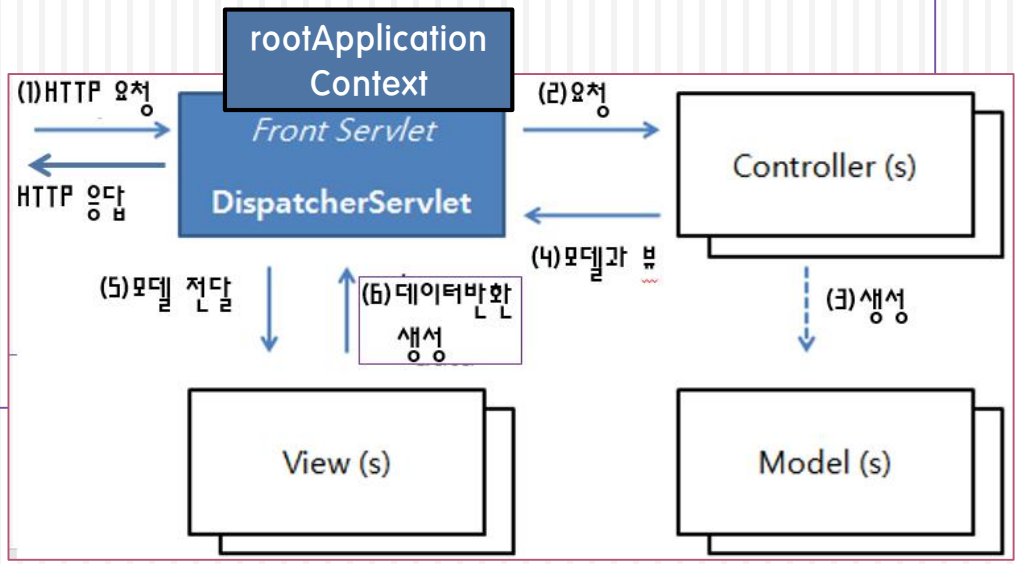
이름	
전화	
내용	

[Submit]

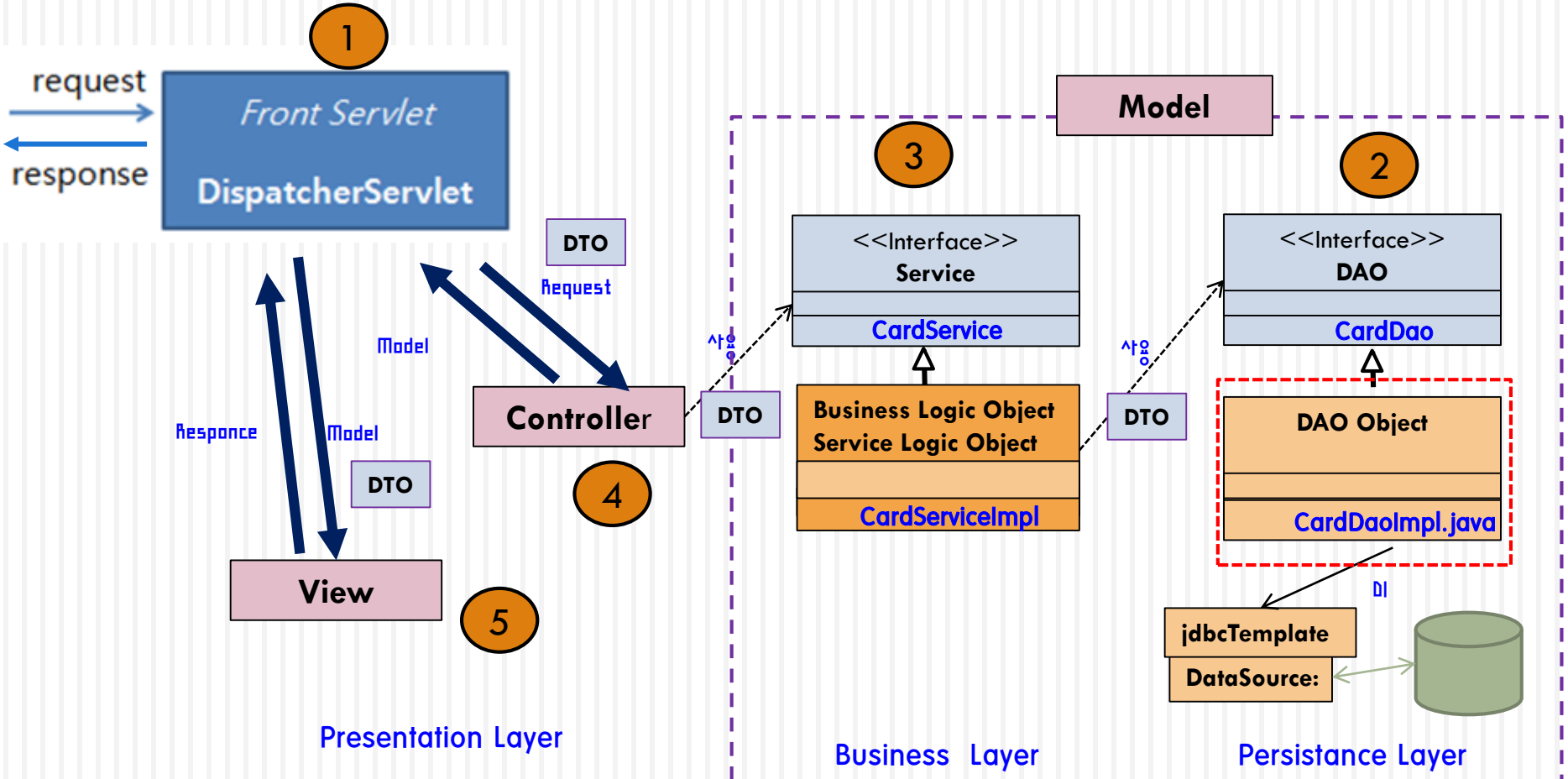
이름	양옥경
전화	010-1234-4321
내용	태진아 부인

Maven project 실행 순서

1. maven Project 생성
2. POM 설정- repository, spring_jdbc, hibernate-validator 등
3. web.xml 설정- 한글, 멀티미디어파일 처리 부분 설정
4. DTO 클래스 만들기
5. root-context.xml ➔ dataSource 객체 설정
6. Dao – 인터페이스, 구현클래스 생성➔ root-context.xml 수정
7. Service – 인터페이스, 구현클래스 생성➔root-context.xml 수정
8. Controller ➔root-context.xml 수정
9. Servlet-context.xml 설정
10. View 생성
11. Test

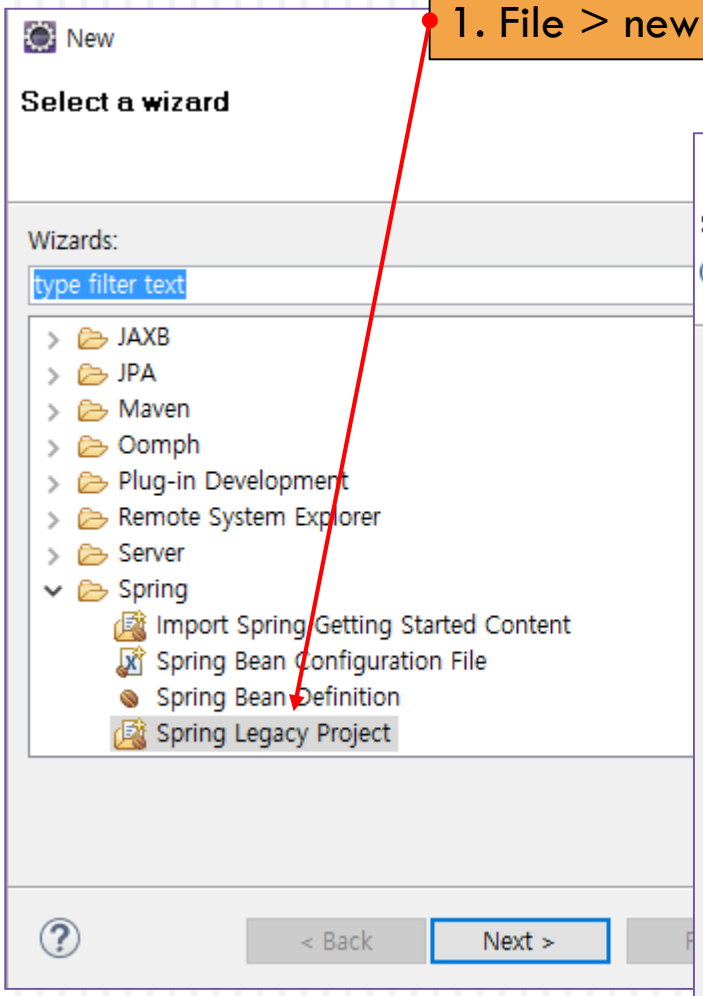


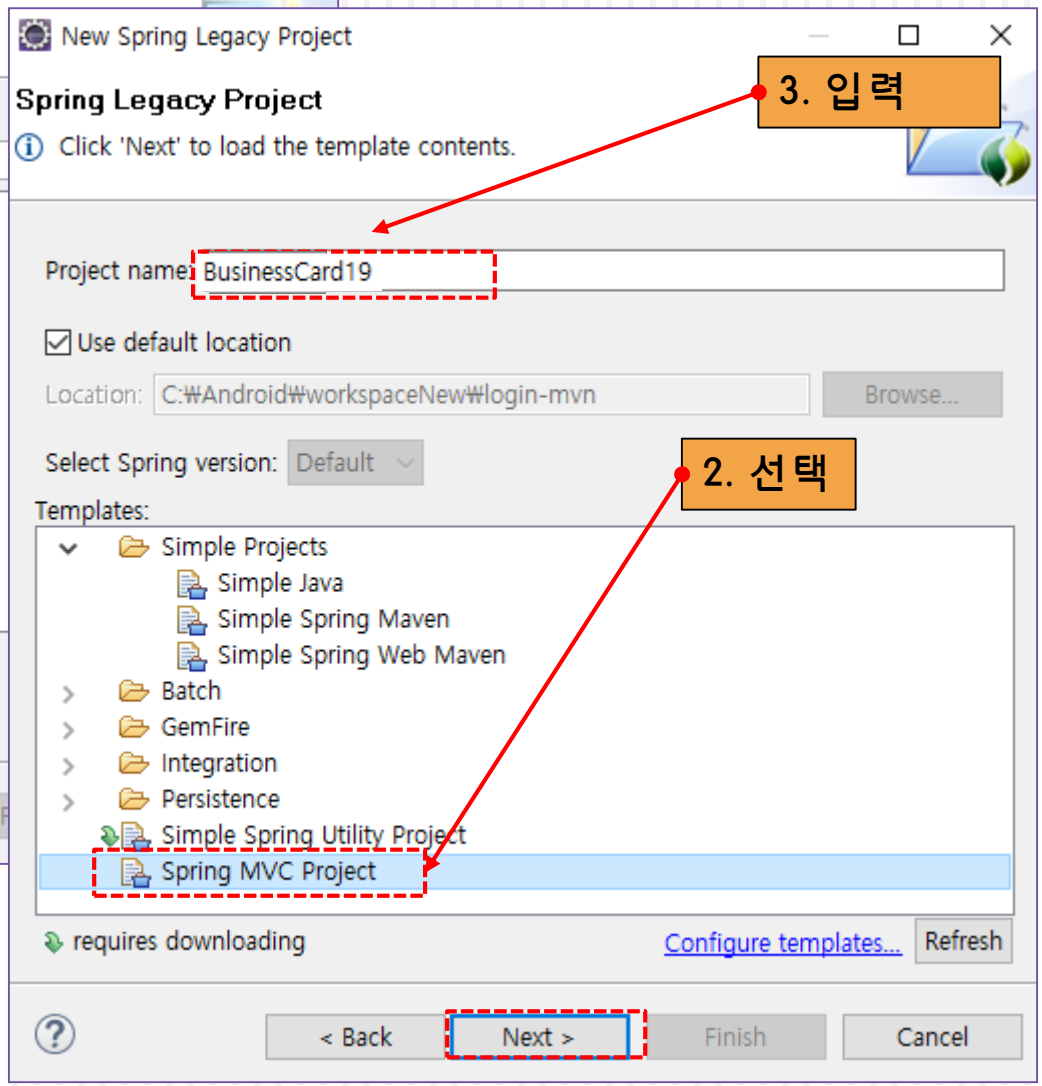
DAO design 패턴 설계 구현



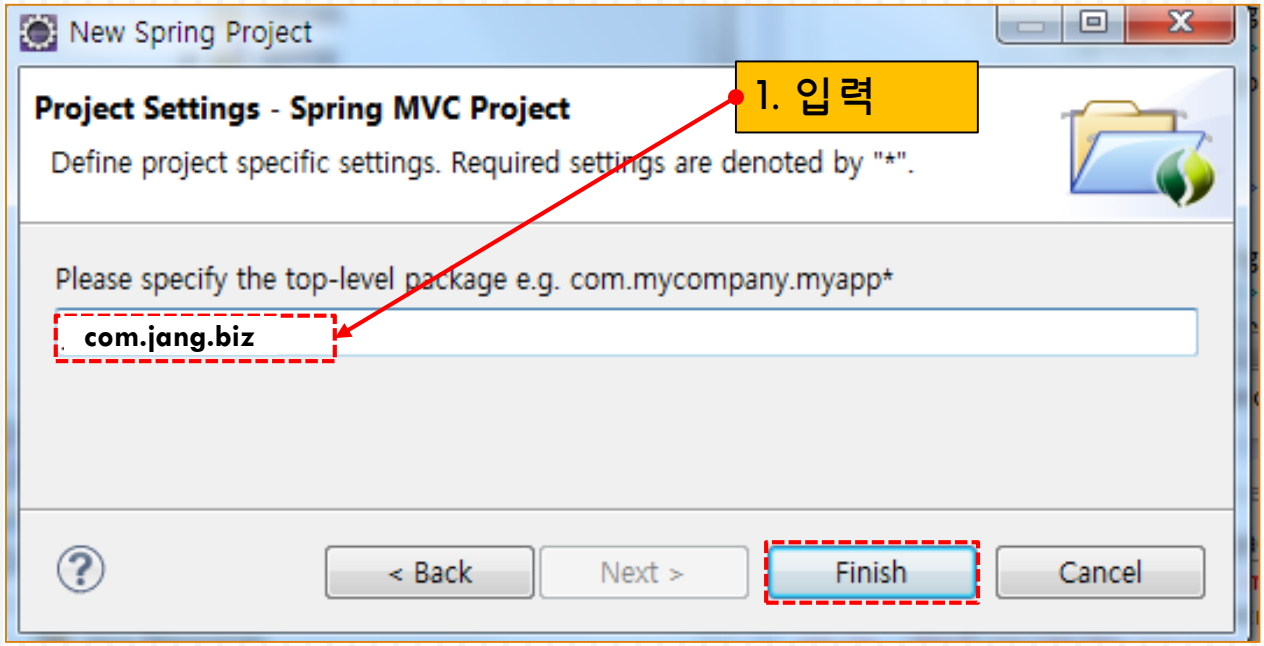
Maven project 구성(1)

1. File > new> other > Spring > Spring Legacy project > next >





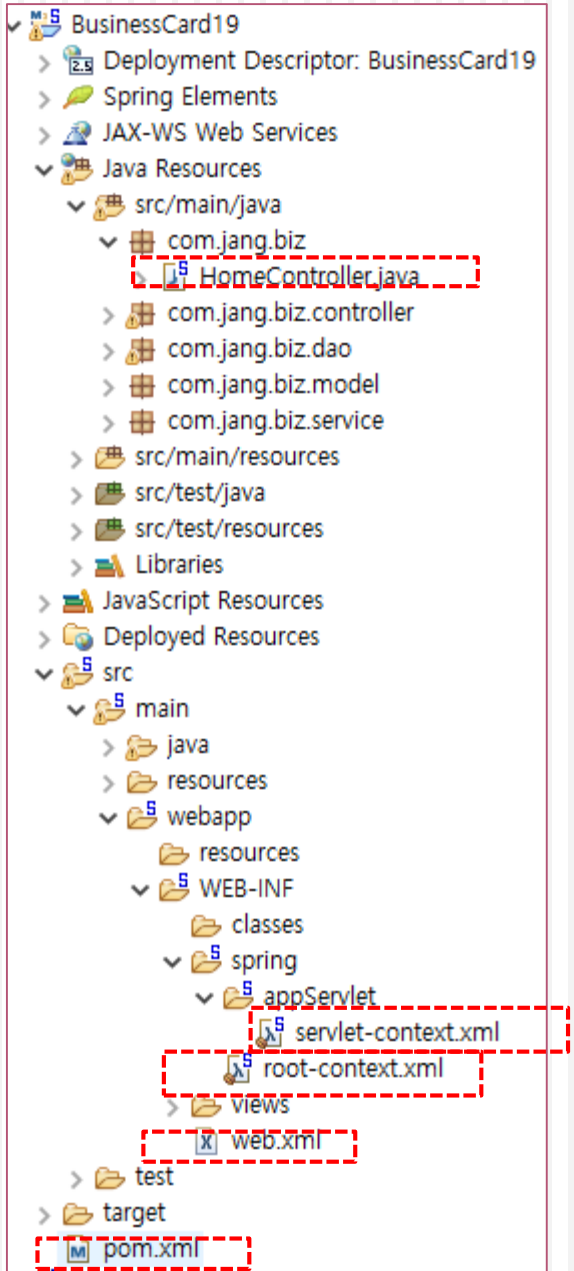
Maven project 구성



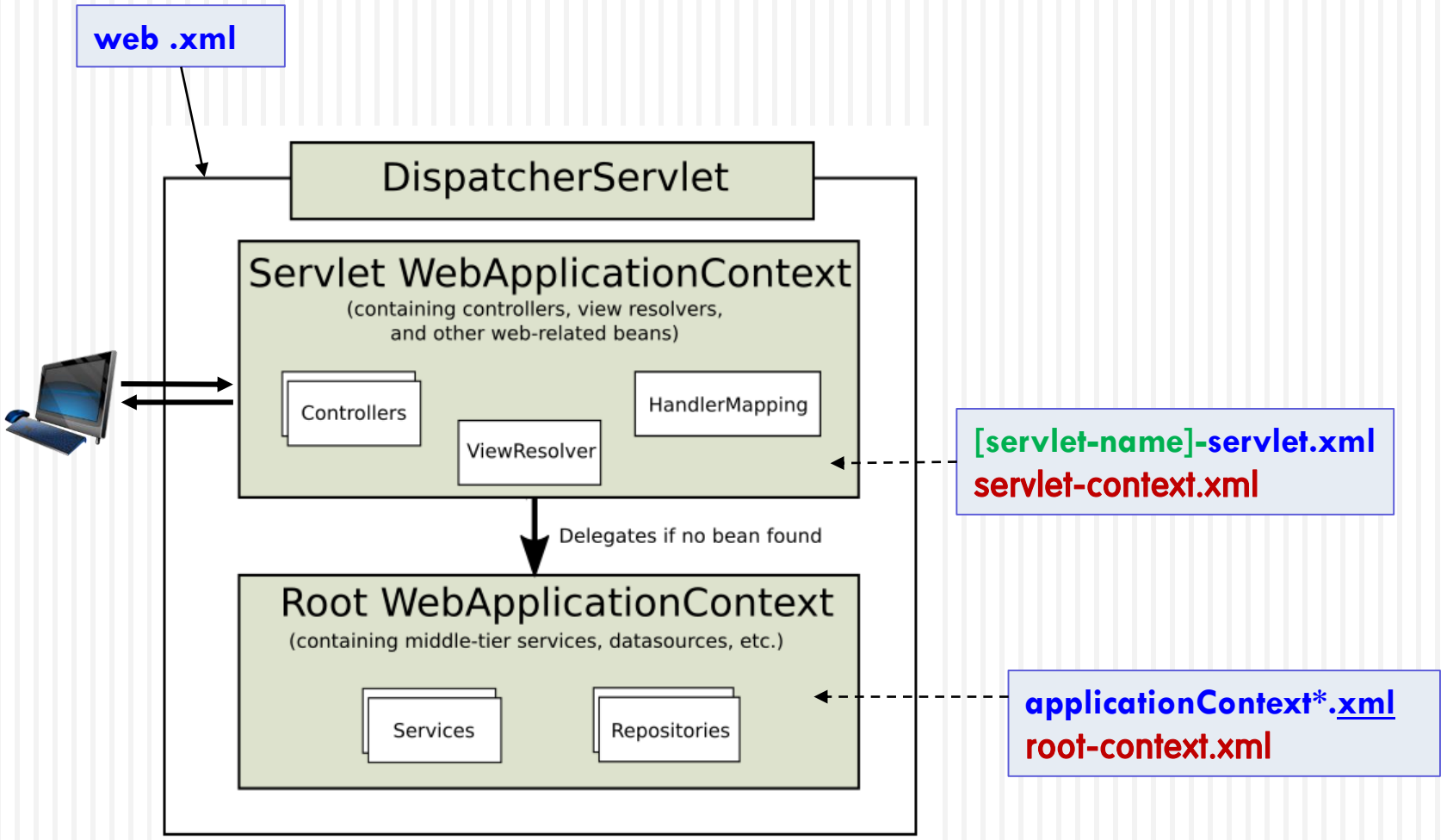
Maven project 구성(2)

1. 기본 구성화일 생성

2. Hello world 기본 실행 확인

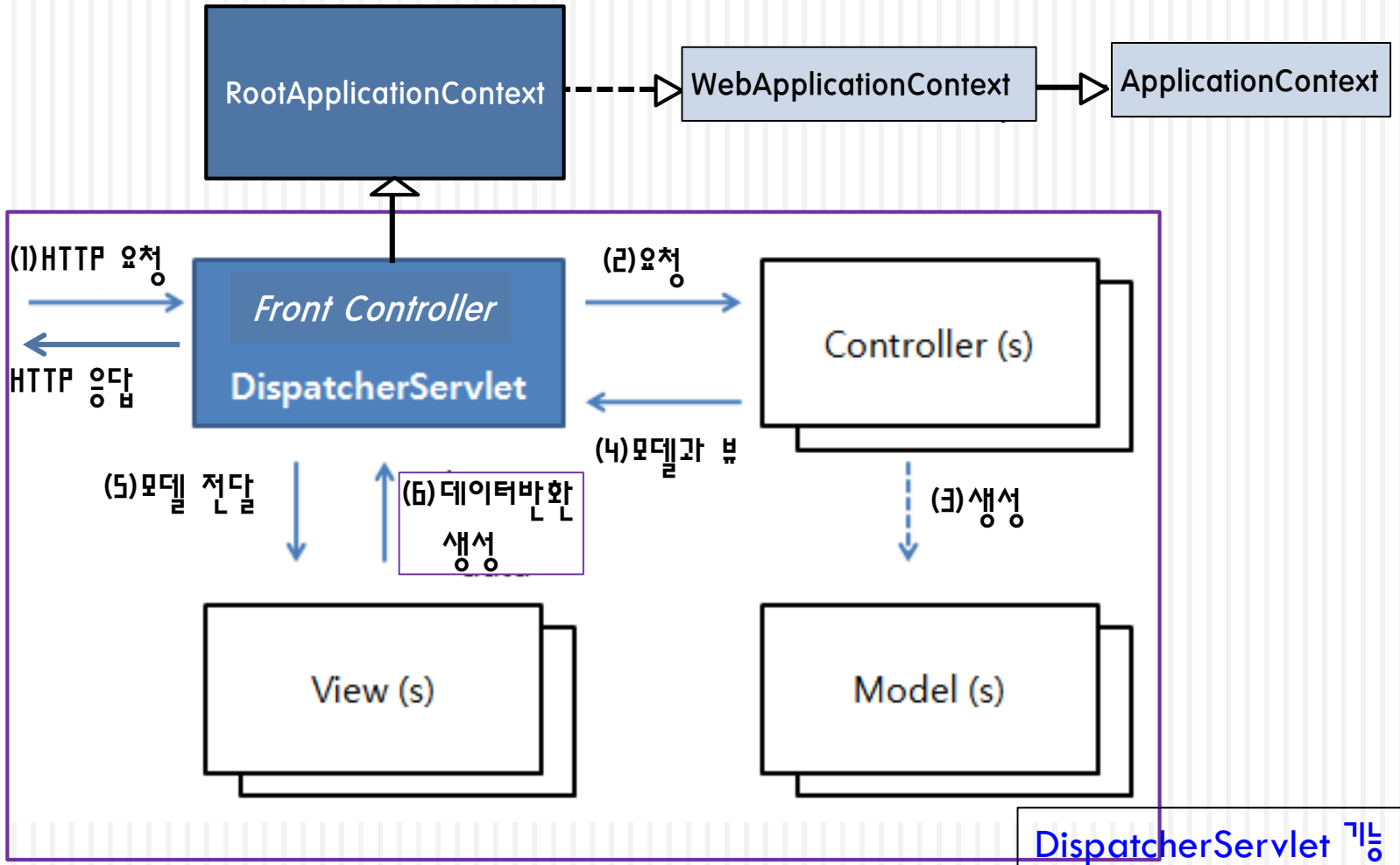


Web IoC 컨테이너 구성방법



DispatcherServlet 과 MVC 아키텍처

- DispatcherServlet은 bean을 생성하고, Client 요청을 적절한 bean에 연결하여 실행하며, 처리결과를 특정 view에 할당하여 Client 요청에 응답한다.



DispatcherServlet 기능

- bean 생성
- HandlerMapping
- ViewResolver

POM 구성

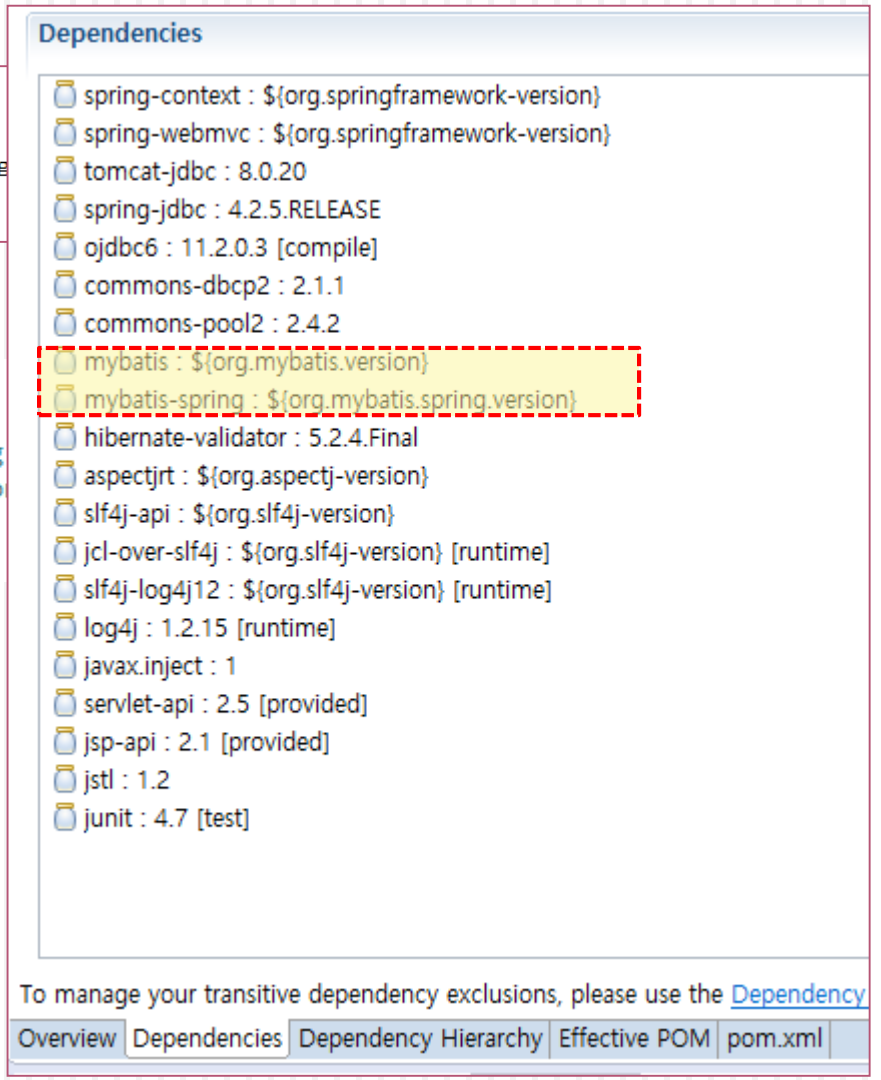
- **Repository 정의 -ojdbC6**

```
<repositories>
    <repository>
        <id>codelds</id>
        <url>https://code.lds.org/nexus/content/g
    </repository>
</repositories>
```

- **properties 정의**

```
<properties>
    <java-version>12</java-version>
    <org.springframework-version>5.0.8.RELEASE</org
    <org.aspectj-version>1.6.10</org.aspectj-versio
    <org.slf4j-version>1.6.6</org.slf4j-version>
</properties>
```

- **dependencies 정의**
  - spring-jdbc
  - ojdbc6
  - commons-dbcp2,commons-pool2
  - hibernate-validator 구성
- **plugin 설정**

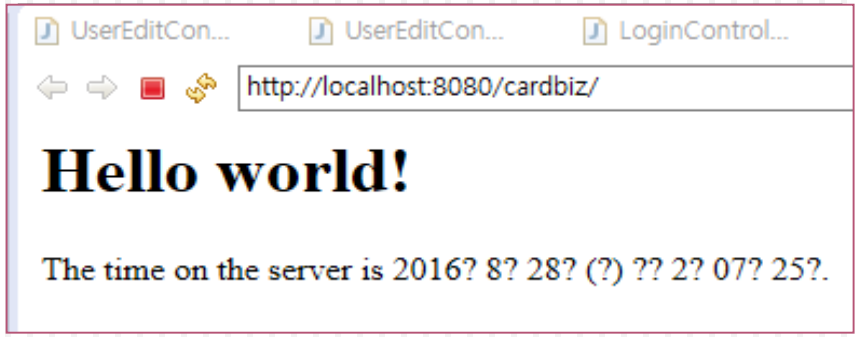


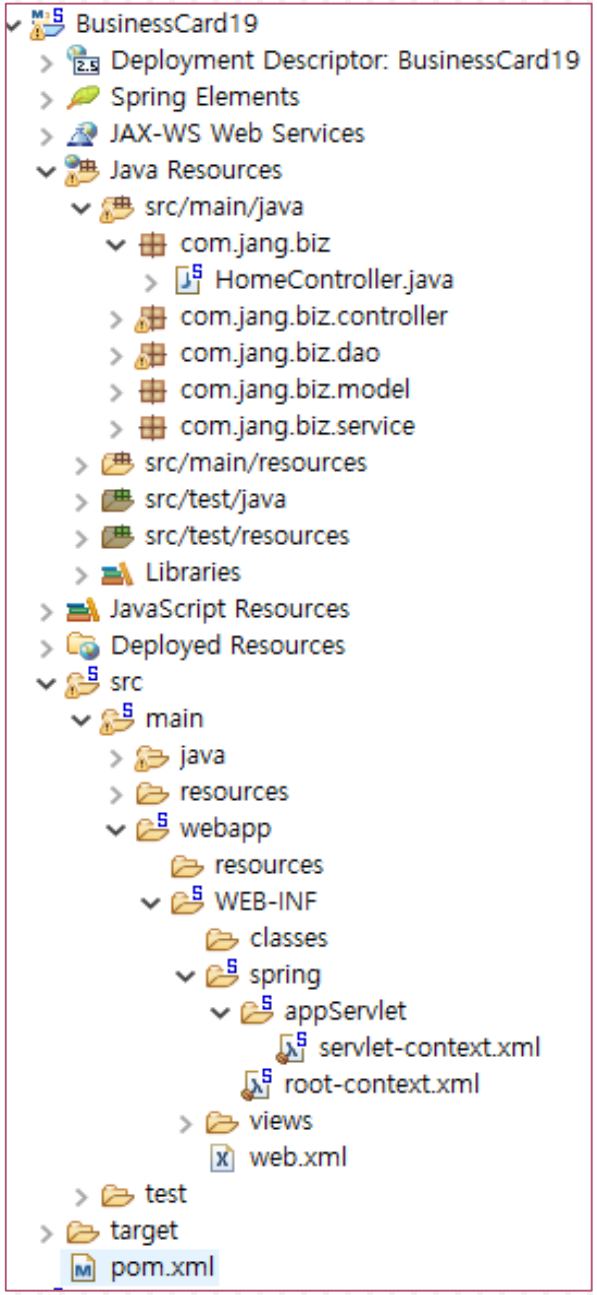
Dependency 추가 –spring-jdbc, validator

```
<!-- Spring JDBC Support -->
<dependency>
	<groupId>org.springframework</groupId>
	<artifactId>spring-jdbc</artifactId>
	<version>${org.springframework-version}</version>
</dependency>
<dependency>
	<groupId>com.oracle</groupId>
	<artifactId>ojdbc6</artifactId>
	<version>11.2.0.3</version>
	<scope>compile</scope>
</dependency>
<dependency>
	<groupId>org.apache.commons</groupId>
	<artifactId>commons-dbcp2</artifactId>
	<version>2.1.1</version>
</dependency>
<dependency>
	<groupId>org.apache.commons</groupId>
	<artifactId>commons-pool2</artifactId>
	<version>2.4.2</version>
</dependency>
<!-- Hibernate Validator -->
<dependency>
	<groupId>org.hibernate</groupId>
	<artifactId>hibernate-validator</artifactId>
	<version>6.0.12.Final</version>
</dependency>
```

M2Eclipse 프로젝트 Layout 변경

- POM 변경 내용 적용
  - Project 우 클릭 > Maven > Update Maven Project > OK
- 구동테스트



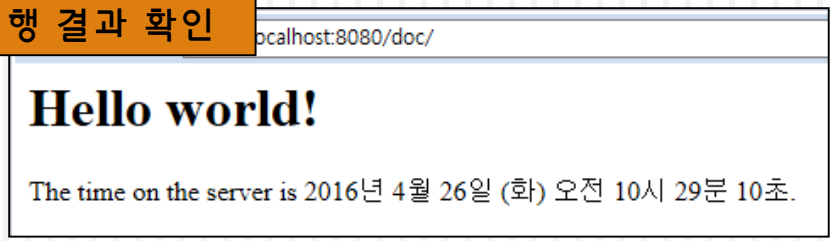


Web.xml 추가 -한글처리

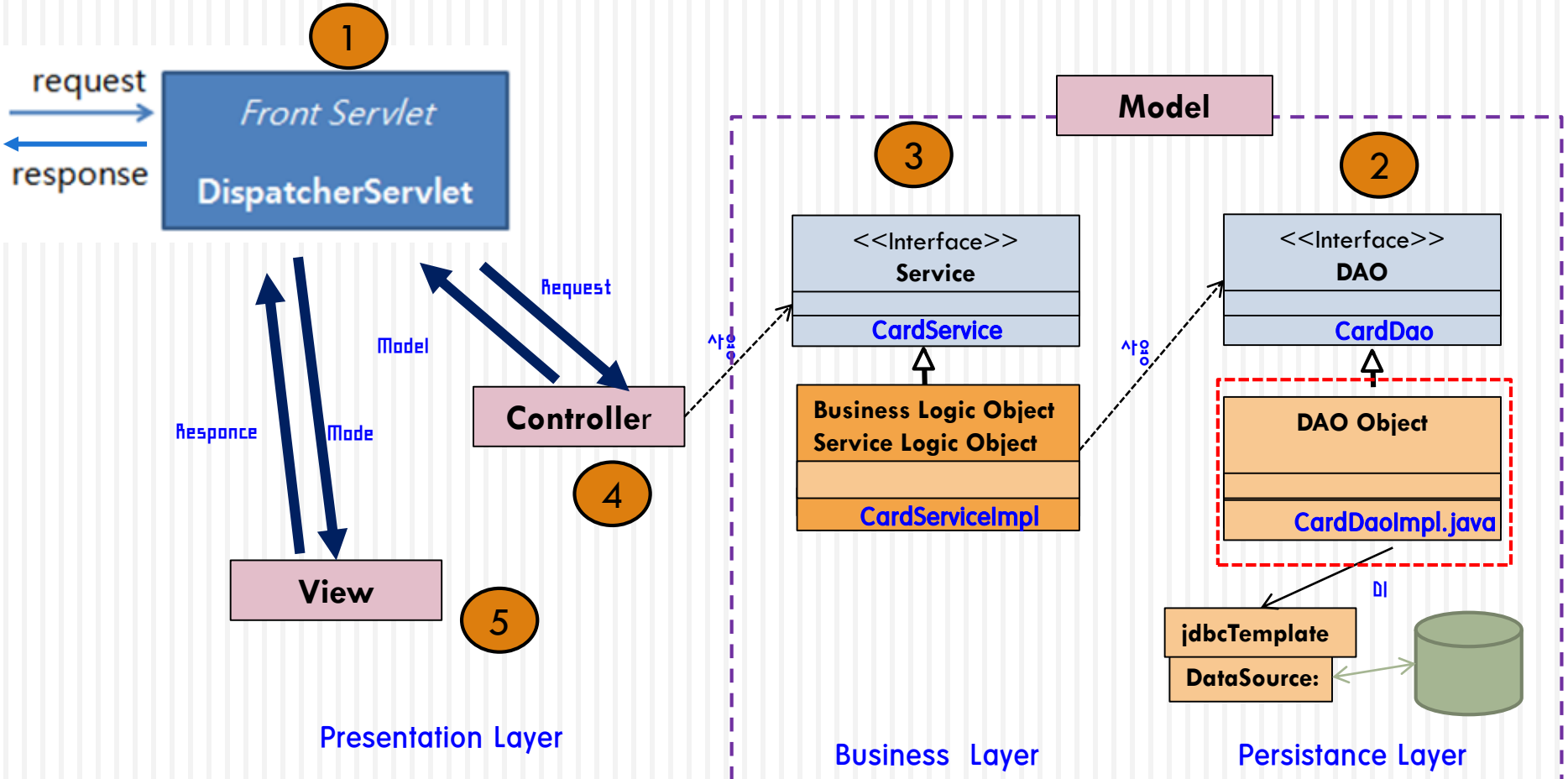
```
<!-- utf-8 filter -->
 <filter>
    <filter-name>encodingFilter</filter-name>
    <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
    <init-param>
       <param-name>encoding</param-name>
       <param-value>UTF-8</param-value>
    </init-param>
    <!-- forceEncoding true 값을 주지 않을 경우 강제 인코딩 하지 않는다. -->
    <init-param>
       <param-name>forceEncoding</param-name>
       <param-value>true</param-value>
    </init-param>
 </filter>
 <filter-mapping>
    <filter-name>encodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
 </filter-mapping>
```

1. <context-param> 위에 추가

2. 실행 결과 확인

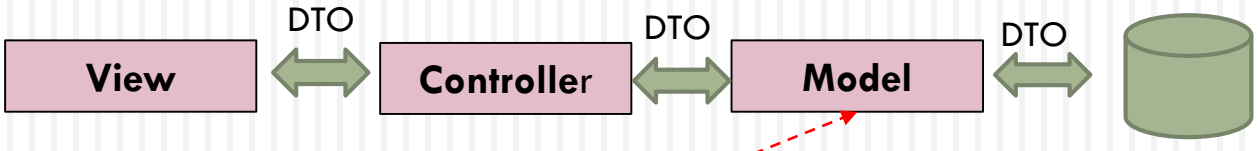


DAO design 패턴 설계 구현



DTO 객체와 DAO 객체

- DTO(Data Transfer Object)
  - VO(Value Object)
  - 계층간 데이터 교환을 위한 자바빈즈(객체)
  - 계층사이를 이동하는 데이터는 대부분 DB의 테이블 항목의 내용을 그대로 전달하므로 테이블 항목과 같은 저장 객체를 만들어 사용하면 데이터 이동이 간단해진다.

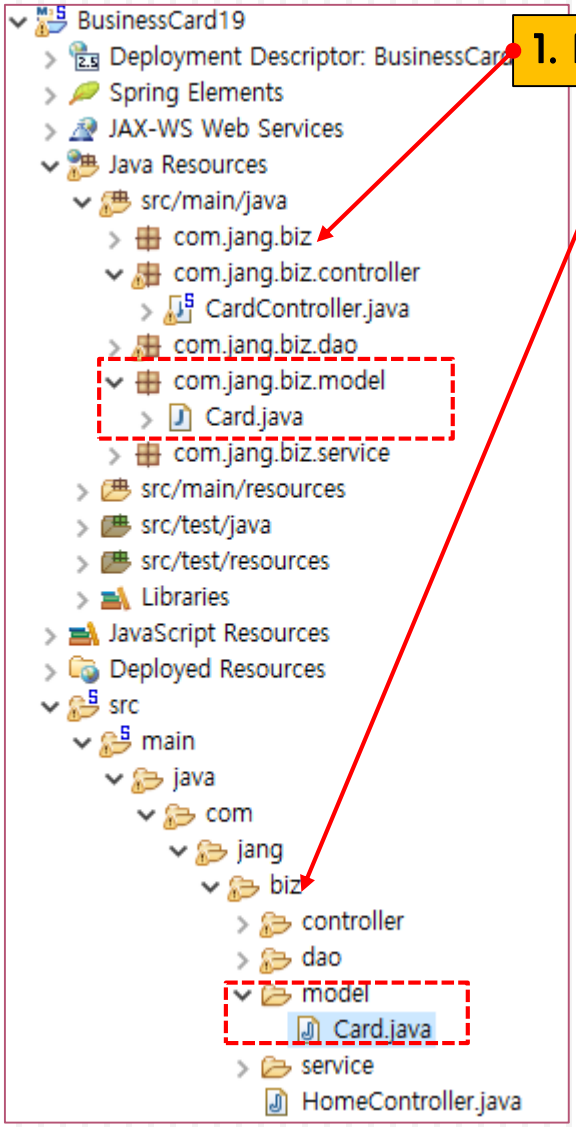


- DAO(Data Access Object)
  - DB를 사용해 데이터를 조회하거나 조작하는 기능을 전담하는 객체
  - 일반적으로 SQL 문을 사용하여 DB 조작
  - Persistence 계층 :데이터베이스(영구저장소)에 data를 CRUD하는 계층

전달객체(Dto) 구성 - Card.java

열(D): 이름 BCARD

PK	이름	데이터 유형	크기
	BNO	NUMBER	
	BNAME	VARCHAR2	50
	PHONE	VARCHAR2	15
	DESCRIPTION	VARCHAR2	1000



1. New>class

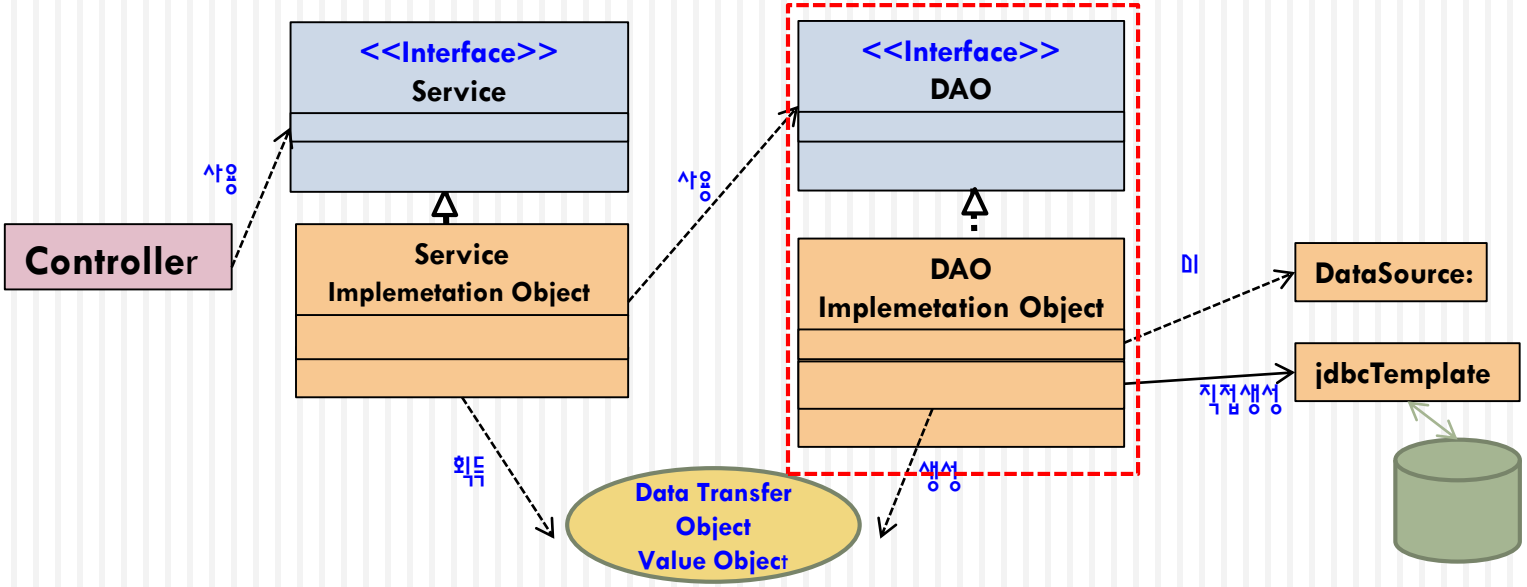
```java
package com.jang.biz.model;

import javax.validation.constraints.NotEmpty;

public class Card {

    private int bno;
    @NotEmpty(message="이름이 입력되지 않았습니다.!")
    private String bname;
    @NotEmpty(message="전화번호가 입력되지 않았습니다.!")
    private String phone;
    @NotEmpty(message="설명이 입력되지 않았습니다.!")
    private String description;
```

2. source>Generate gatter&setter

DAO 클래스 작성

- 일반적으로 DAO 클래스는 하나의 테이블에 CRUD 작업을 수행하며 다양한 형태의 검색 메서드를 구성할 수 있다.
  - 조회(query)의 경우만 반환값이 있다.
  - Update() ,insert().delete() 메서드는 영향을 받은 레코드의 개수를 반환한다.

- DI(의존관계 주입) 구현을 위하여 Interface 를 먼저 작성하고 이를 상속받아 DAO를 구현한다. (Strategy pattern)



CardDao interface 객체 구성

```
package com.jang.biz.dao;

import java.util.List;

public interface CardDao {

    List<Card> getCardList();

    Card getCard(int bno);

    int addCard(Card card);

    int updateCard(Card card);

    int deleteCard(int bno);

}
```

메시지 :

[신규]

번호	이름	전화	기록	수정	삭제
1	양현정34	010-1234-5678	현장감독	수정	삭제
2	김현상	010-2123-9867	백수대왕	수정	삭제
3	강양욱	010-2132-2234	영화 감독	수정	삭제
4	양99	010-1234-5678	998876	수정	삭제
5	김수민121	010-2345-6543	998876	수정	삭제

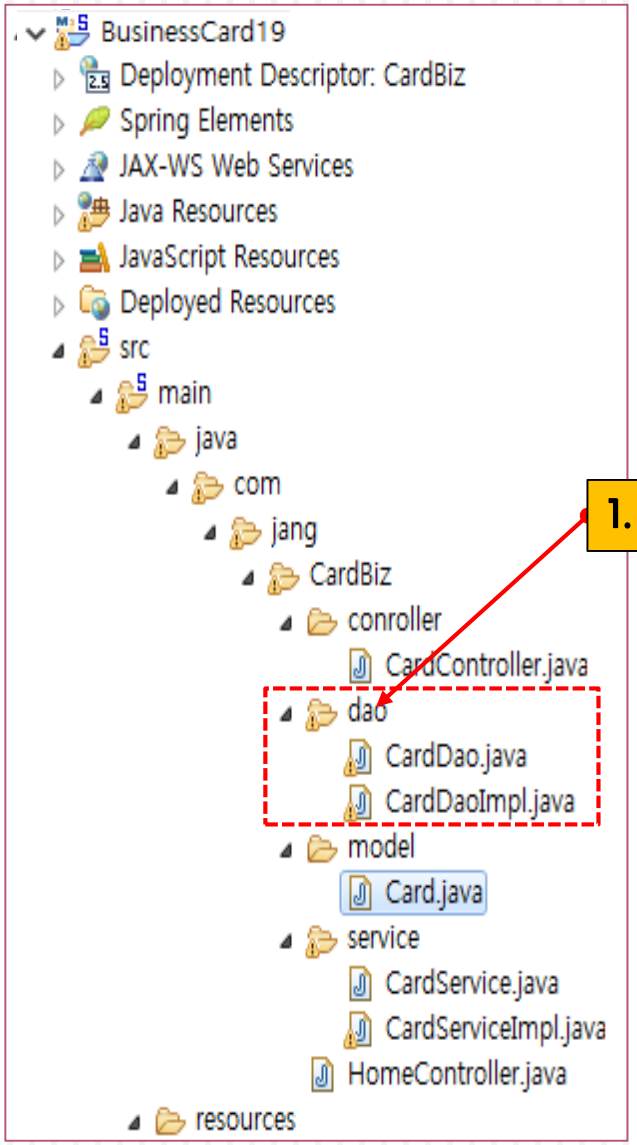
[Submit]

이름	양옥경
전화	010-1234-4321
내용	태진아 부인

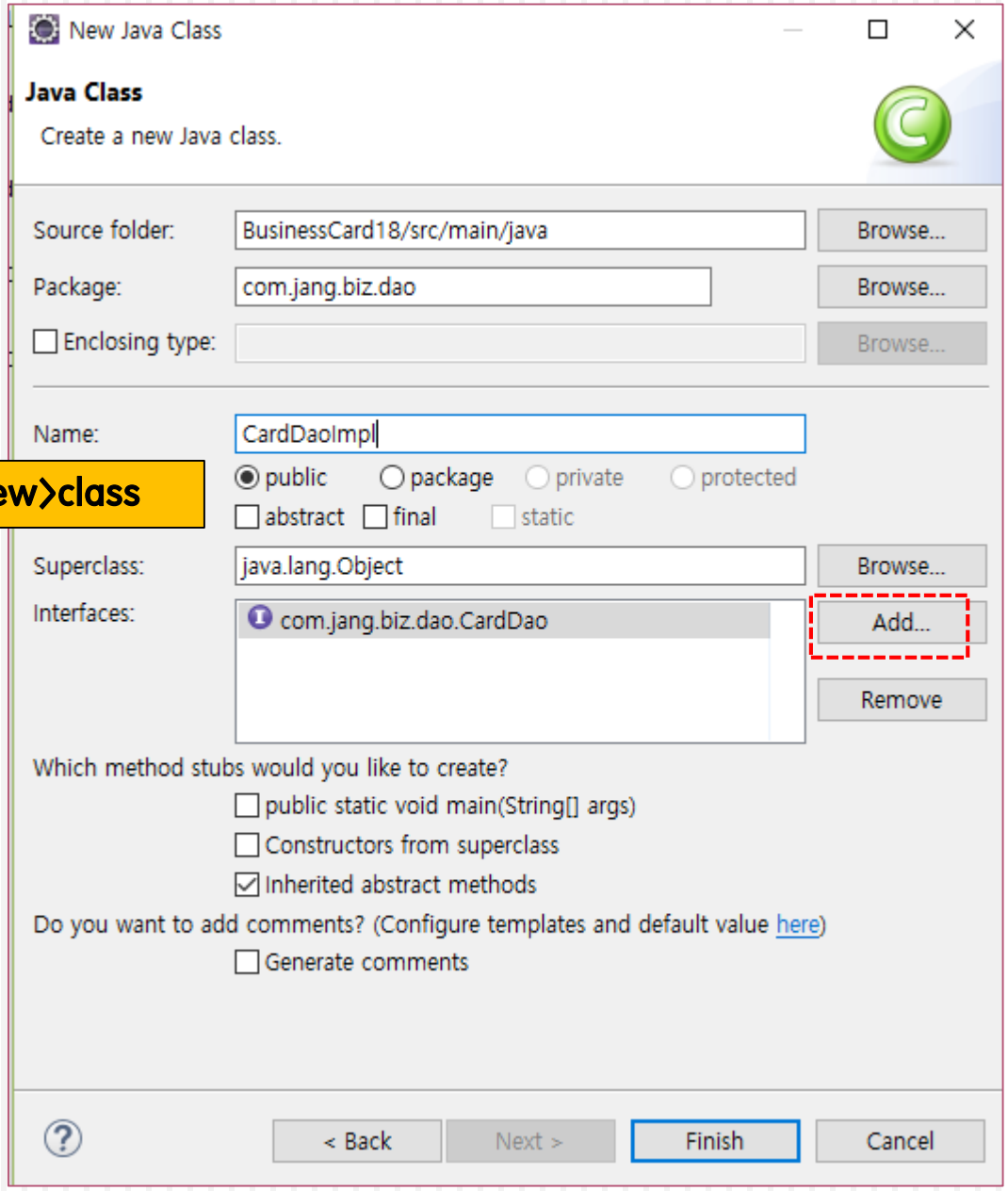
메서드 명칭과 매개변수, 수행후 반환 받고자 하는 데이터 형에 주의

JdbcTemplate.**update()** 메소드는 쿼리 실행 결과로 변경된 행의 개수를 리턴한다

CardDaoImpl 구현 클래스 작성



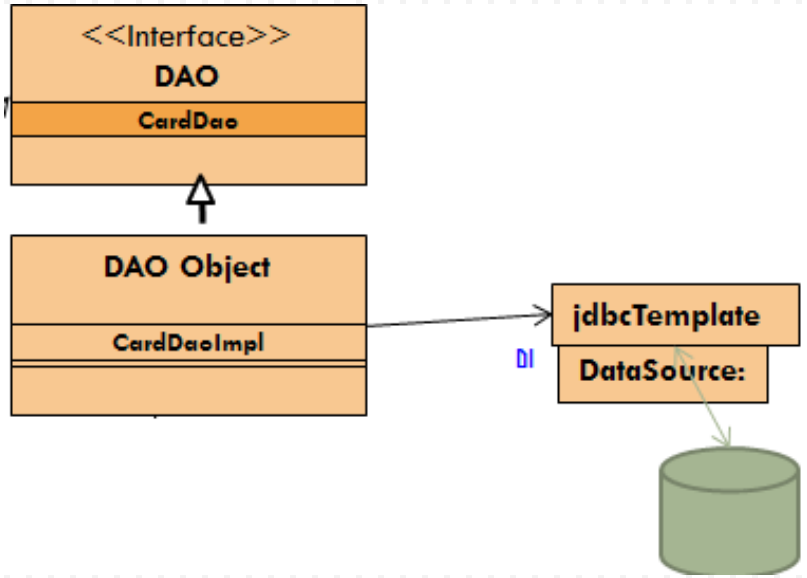
1. New>class



CardDaoImpl 구현 클래스 작성

```java
public class CardImpl   implements CardDao {

    @Override
    public List<Card> getCardList() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public Card getCard(int bno) {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public int addCard(Card card) {
        // TODO Auto-generated method stub
        return 0;
    }

    @Override
    public int updateCard(Card card) {
        // TODO Auto-generated method stub
        return 0;
    }

    @Override
    public int deleteCard(int bno) {
        // TODO Auto-generated method stub
        return 0;
    }

}
```

- Jdbc를 사용하기 쉽게 JDBC를 한 단계 더 추상화 시킨 JDBC API를 제공한다.
  - jdbcTemplete, RowMapper, SqlParameterSource 클래스 제공
  - 개발속도 향상
  - 에러발생 가능성 감소
- 개발자가 직접 해결 해야하는 connection, 명령객체 생성, SQL 실행 ,SQL Exception 처리와 같은 작업을 프레임워크가 담당



Java 의 자료구조

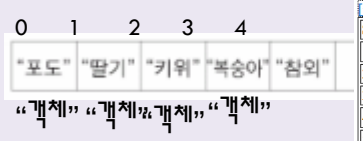
배열 (Array)

같은 타입의 여러 변수를 하나의 묶음으로 다루는 것. 정적인 크기 결정
선언 : int[] arr = new int[5]
사용 : arr[0] ~ arr[4]

List (interface)

Aarray list : 동적인 크기 결정, 인덱스를 가지고 있으며, 모든 타입의 데이터를 담을 수 있는 구조(제네릭을 사용)
검색이 빠르다. 데이터 추가 삭제시 성능 떨어짐,

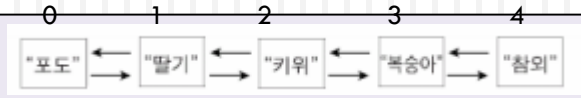
ArrayList <String> myList = new ArrayList <String>( );
List <string> myList = new ArrayList <string>( );



메시지 :
[신규]

Sample No	Title	Phone	Description	수정	삭제
1	양옥경	010-1234-4321	태진아 부인	수정	삭제
2	이현상	010-2234-2321	저현상 동생	수정	삭제
3	김수현	010-3234-3321	드라마 작가	수정	삭제
4	박상현	010-4234-4421	정치가	수정	삭제
5	강양원	010-5234-5421	장안대학교수	수정	삭제

linkedlist : 인덱스 없음, 추가 삭제 빠르나 차례 대로 검색- 느림



List<String> myList = new LinkedList<String>();

사용: myList.add("F"), myList.remove(2), myList.get(2); myList.set(2,"a") , myList.size, int index1 = list.indexOf("사과");

Vector : 공유자원과 복수의 사용자가 사용하는 스레드 동기화 프로그램에 사용- 동기화 처리 때문에 속도가 느려짐

Map (interface)

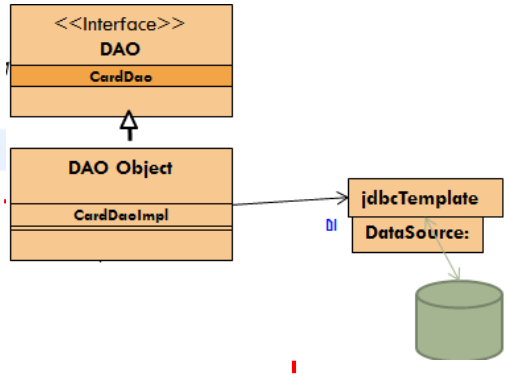
HashMap : 키와 데이터 값의 한쌍으로 묶어서 관리하며 키의 중복을 허용하지 않는다0.
HashMap <String, String> map = **new** HashMap <String, String>();
Map <String, String> map = new HashMap <String, String>();

key	value	
people	사람	"객체"
baseball	야구	"객체"

map.put("people", "사람"); map.put("baseball", "야구");
map.get("people"), map.remove("people"), map.size()

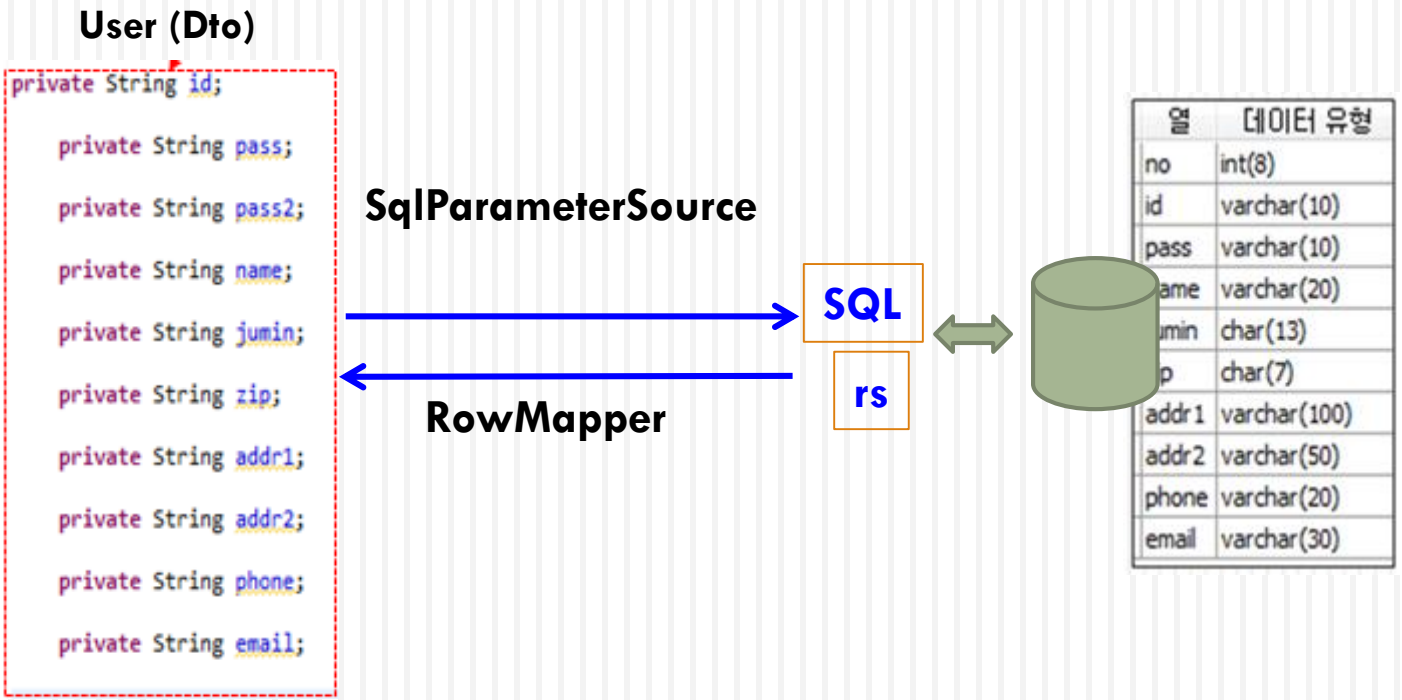
CardDaoImpl 구현 클래스 작성

```java
package com.jang.CardBiz.dao;

import java.util.List;

public class CardDaoImpl implements CardDao {

    private JdbcTemplate jdbcTemplate;
    private NamedParameterJdbcTemplate jdbcTemplate2;

    public void setDataSource(DataSource dataSource) {
        this.jdbcTemplate = new JdbcTemplate(dataSource);
        this.jdbcTemplate2 = new NamedParameterJdbcTemplate(dataSource);
    }
```

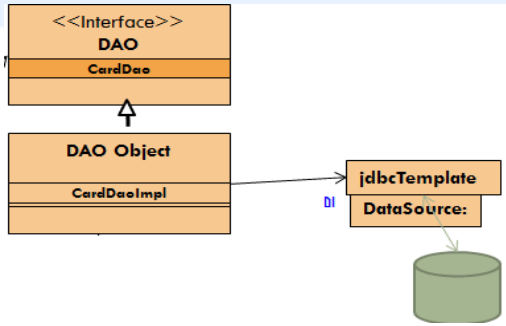



```java
    @Override
    public List<Card> getCardList() {
        String SQL= "SELECT bno,bname,phone,description FROM  bcard order by bno asc ";
        RowMapper mapper = new BeanPropertyRowMapper <Card>(Card.class);
        List<Card> cList = (List)this.jdbcTemplate.query(SQL, mapper);
        return  cList ;
    }
```

SqlParameterSource와 RowMapper



CardDaoImpl 구현 클래스 작성(2 : 읽기 메서드)




```java
public class CardDaoImpl implements CardDao {

    private JdbcTemplate jdbcTemplate;
    private NamedParameterJdbcTemplate jdbcTemplate2;

    public void setDataSource(DataSource dataSource) {
        this.jdbcTemplate = new JdbcTemplate(dataSource);
        this.jdbcTemplate2 = new NamedParameterJdbcTemplate(dataSource);
    }

    @Override
    public List<Card> getCardList() {
        String SQL= "SELECT bno,bname,phone,description FROM  bcard order by bno asc ";
        RowMapper<Card> mapper = new BeanPropertyRowMapper <Card>(Card.class);
        List<Card> cList = (List) this.jdbcTemplate.query(SQL, mapper);
        return  cList ;
    }

    @Override
    public Card getCard(int bno) {
        String SQL= "SELECT bno,bname,phone,description FROM  bcard WHERE  bno = ?";
        RowMapper <Card> mapper = new BeanPropertyRowMapper<Card>(Card.class);
        return this.jdbcTemplate.queryForObject(SQL, mapper, bno);
    }
```

CardDaoImpl 구현 클래스 작성(3)

```
@Override
public int  addCard(Card card) {
    int MaxNo= (int)this.jdbcTemplate.queryForObject("select max(bno)+1 from bcard",Integer.class);
    card.setBno(MaxNo);
    String SQL="INSERT INTO bcard (bno,bname,phone,description) VALUES (:bno,:bname,:phone,:description)";
    SqlParameterSource parameterSource = new BeanPropertySqlParameterSource(card);
    return this.jdbcTemplate2.update(SQL, parameterSource);
}

@Override
public int updateCard(Card card) {
    String SQL="UPDATE bcard SET bname = :bname, phone = :phone, description = :description WHERE  bno  = :bno";
    SqlParameterSource parameterSource = new BeanPropertySqlParameterSource(card);
    return this.jdbcTemplate2.update(SQL, parameterSource);
}

@Override
public int deleteCard(int bno) {
    return this.jdbcTemplate.update("DELETE FROM bcard WHERE  bno = ?", bno);
}
```

root-context.xml에 dataSource 와 cardDao 객체 생성

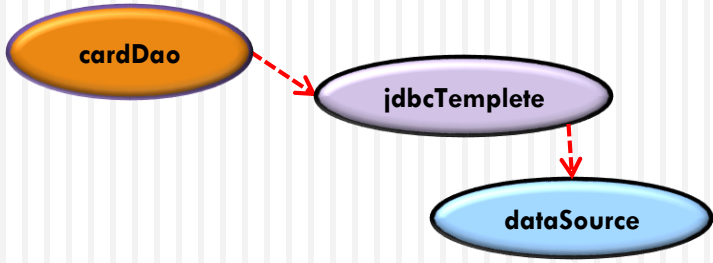
```
<!-- Data Source  -->
<bean id="dataSource" class="org.apache.commons.dbcp2.BasicDataSource" >
    <property name="driverClassName">
        <value>oracle.jdbc.driver.OracleDriver</value>
    </property>
    <property name="url">
        <value>jdbc:oracle:thin:@168.126.146.45:1521:orcl</value>
    </property>
    <property name="username">
        <value>jjin</value>
    </property>
    <property name="password">
        <value>jjinpang</value>
    </property>
    <property name="maxTotal">
        <value>2</value>
    </property>
    <property name="maxIdle">
        <value>5</value>
    </property>
    <property name="maxWaitMillis">
        <value>10000</value>
    </property>
</bean>
```

```
<!-- CardDao -->
 <bean id="cardDao" class="com.jang.biz.dao.CardDaoImpl" >
    <property name="dataSource">
        <ref bean="dataSource" />
    </property>
 </bean>
```

xml 메터설정 정보와 setxxx()메서드 호출

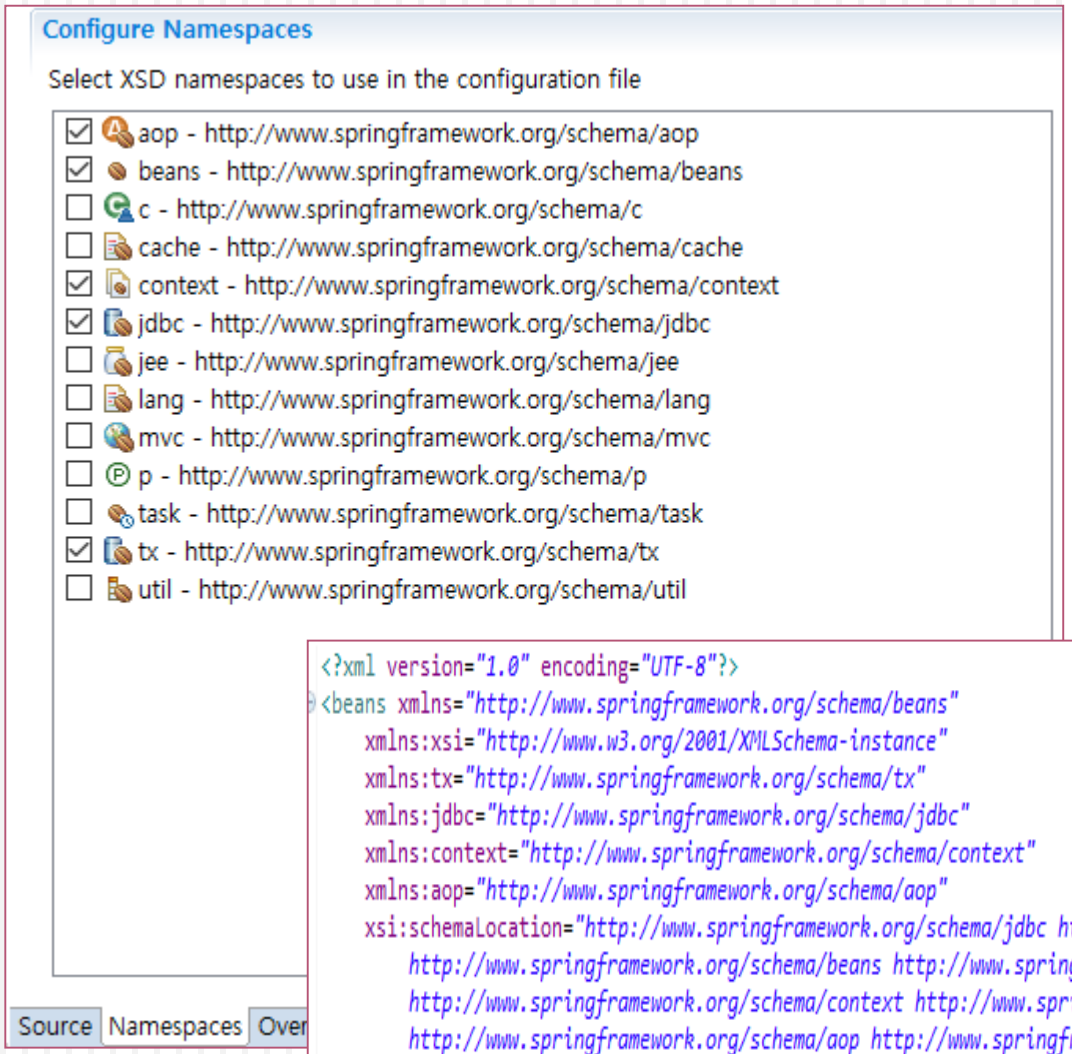
- 애플리케이션 시작시 *CardDaoImpl 클래스를 이용하여 cardDao* 객체를 생성한다.
- *cardDao* 객체 생성시 setDataSource()메서드를 실행하며 dataSource 객체를 파라메터(매개변수)로 전달한다.

```
<!-- CardDao -->
 <bean id="cardDao" class="com.jang.biz.dao.CardDaoImpl" >
    <property name="dataSource">
        <ref bean="dataSource" />
    </property>
 </bean>
```



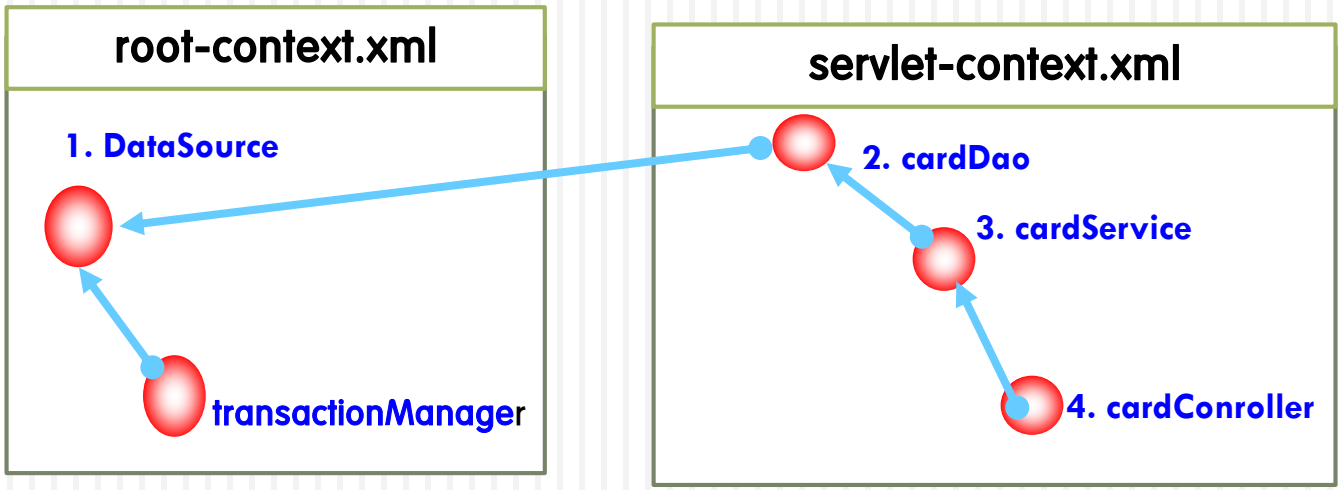

```
public class CardDaoImpl implements CardDao {

    private JdbcTemplate jdbcTemplate;
    private NamedParameterJdbcTemplate jdbcTemplate2;

    public void setDataSource(DataSource dataSource) {
        this.jdbcTemplate = new JdbcTemplate(dataSource);
        this.jdbcTemplate2 = new NamedParameterJdbcTemplate(dataSource);
    }
```

root-context.xml 파일에 Namespace 추가




```
<?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:tx="http://www.springframework.org/schema/tx"
    xmlns:jdbc="http://www.springframework.org/schema/jdbc"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xsi:schemaLocation="http://www.springframework.org/schema/jdbc http://www.springframework.org/schema/jdbc/spring-jdbc-4.3.xsd
        http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.3.xsd
        http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.3.xsd
        http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.3.xsd">
```

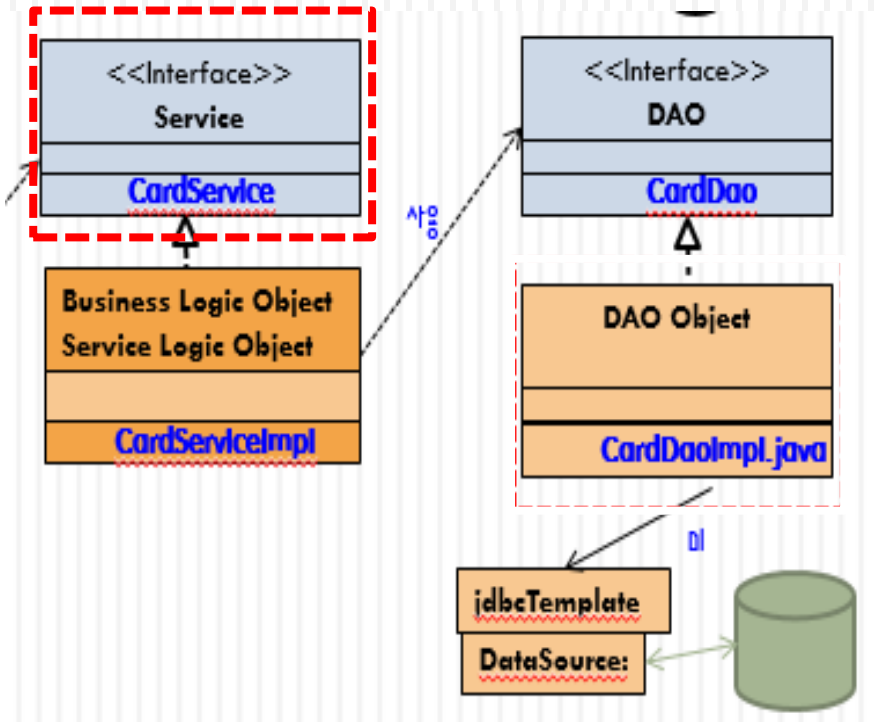
컨테이너와 의존관계 설정



cardService 인터페이스 생성

1. 우클릭 > New > interface
2. 컨트롤러와 Dao 사이에서 Dao의 메서드를 호출하여 결과를 받아 비즈니스 로직을 처리하여 결과를 컨트롤러에 반환하는 메서드 결정
   (메서드 이름, 매개변수, 수행후 반환 받고자 하는 데이터 형)

```java
package com.jang.biz.service;

import java.util.List;

public interface CardService {

    List<Card> getCardList();

    Card getCards(int bno);

    int addCards(Card card);

    int updateCards(Card card);

    int deleteCards(int bno);

}
```

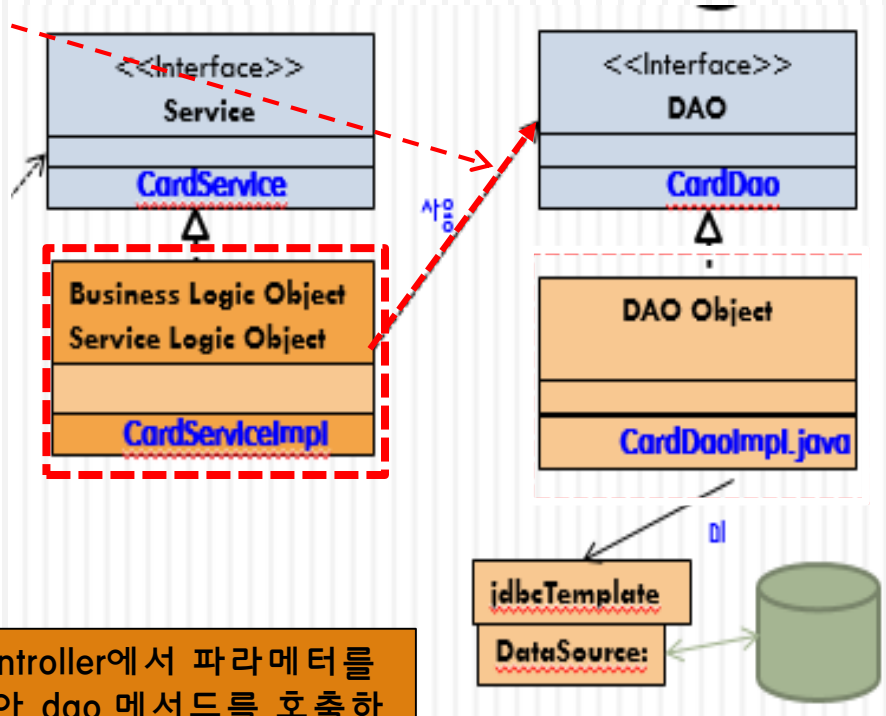


cardService 구현 클래스 생성

1. 우클릭 > New > class
2. 여기서는 컨트롤러와 Dao 사이에서 Dao의 메서드를 호출하여 컨트롤러와 연결하는 단순 메서드 작성 (매개변수, 반환값 결정)

```java
@Service(value = "cardService")
public class CardServiceImpl implements CardService {

    @Resource(name = "cardDao")
    private CardDao cardDao;

    @Override
    public List<Card> getCardList() {
        return this.cardDao.getCardList();
    }

    @Override
    public Card getCards(int bno) {
        return this.cardDao.getCard(bno);
    }

    @Override
    public int addCards(Card card) {
        return this.cardDao.addCard(card);
    }

    @Override
    public int updateCards(Card card) {
        return this.cardDao.updateCard(card);
    }

    @Override
    public int deleteCards(int bno) {
        return  this.cardDao.deleteCard(bno);
    }

}
```

cardDaoImpl 객체는 외부 root-context.xml 파일에서 생성하여 주입된다.(DI)

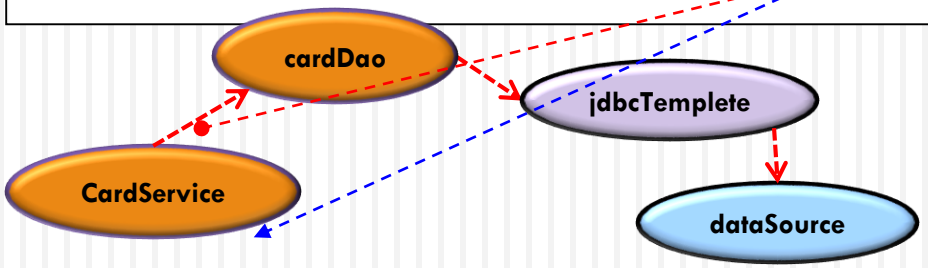


Controller에서 파라메터를 받아 dao 메서드를 호출하고 처리 결과 return

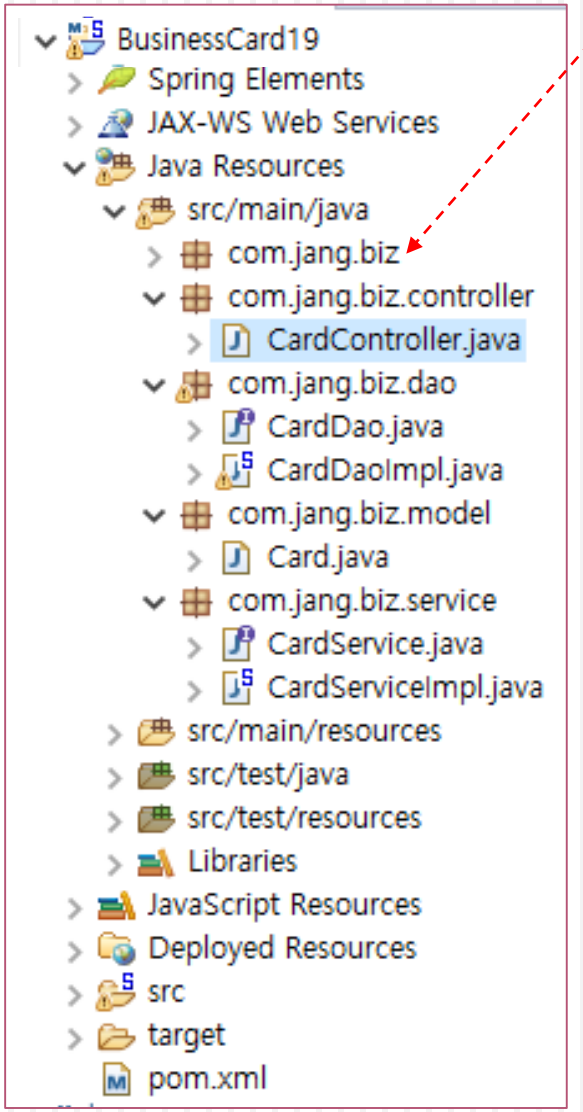
servlet-context.xml 설정

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="http://www.springframework.org/schema/mvc"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:beans="http://www.springframework.org/schema/beans"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc.xsd
        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">

    <!-- DispatcherServlet Context: defines this servlet's request-processing infrastructure -->

    <!-- Enables the Spring MVC @Controller programming model -->
    <annotation-driven />

    <!-- Handles HTTP GET requests for /resources/** by efficiently serving up static resources in the ${webappRoot}/resource
    <resources mapping="/resources/**" location="/resources/" />

    <!-- Resolves views selected for rendering by @Controllers to .jsp resources in the /WEB-INF/views directory -->
    <beans:bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <beans:property name="prefix" value="/WEB-INF/views/" />
        <beans:property name="suffix" value=".jsp" />
    </beans:bean>

    <context:component-scan base-package="com.jang.cardbiz" />


</beans:beans>
```

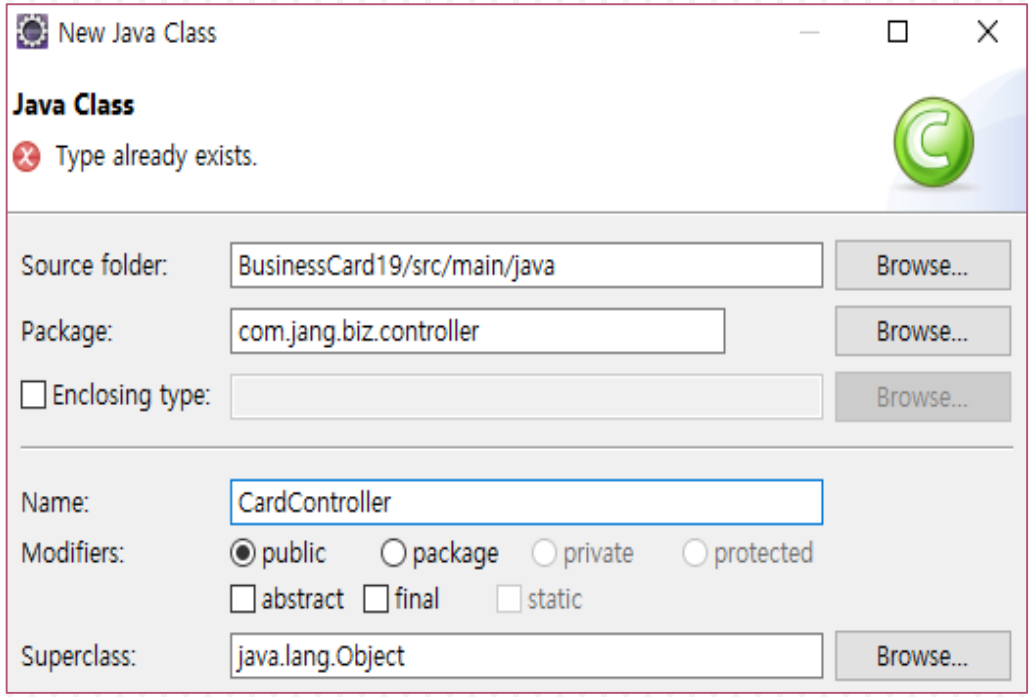
```java
@Service(value = "cardService")
public class CardServiceImpl implements CardService {


    @Resource(name = "cardDao")
    private CardDao cardDao;

    @Override
    public List<Card> getCardList() {
        return this.cardDao.getCardList();
    }
```



Controller 작성 (1)

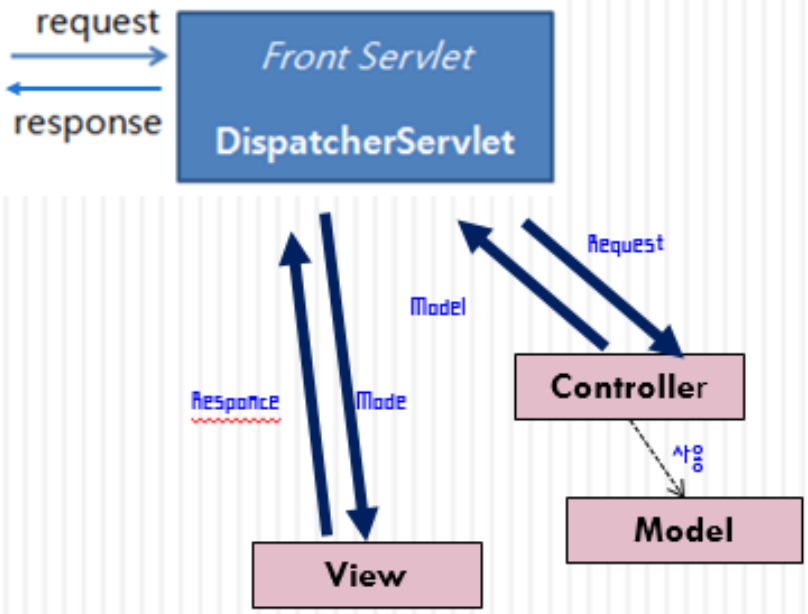


1. 우클릭 > new > class 생성

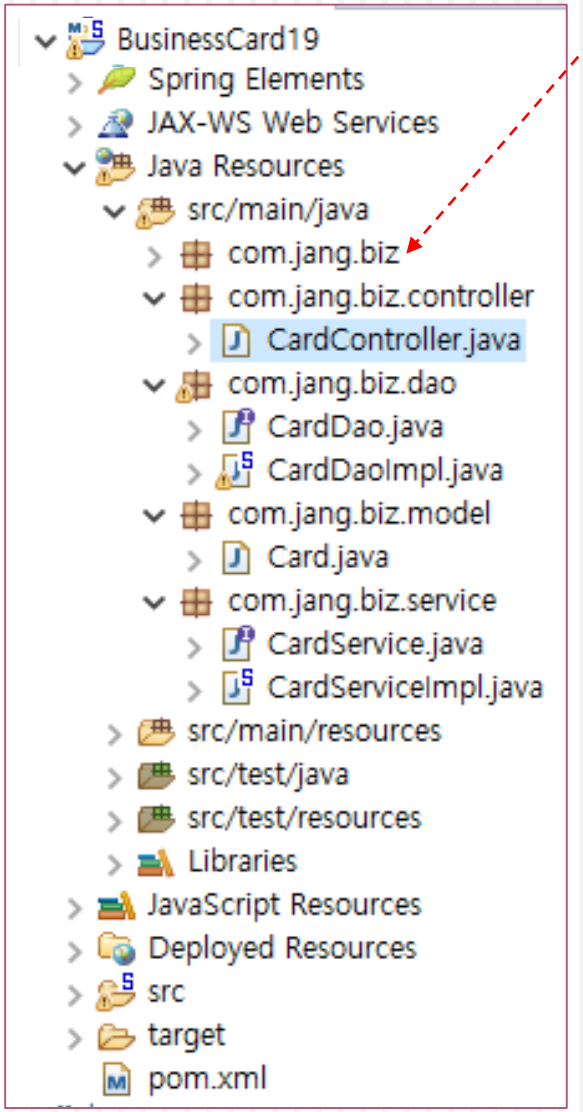


Controller

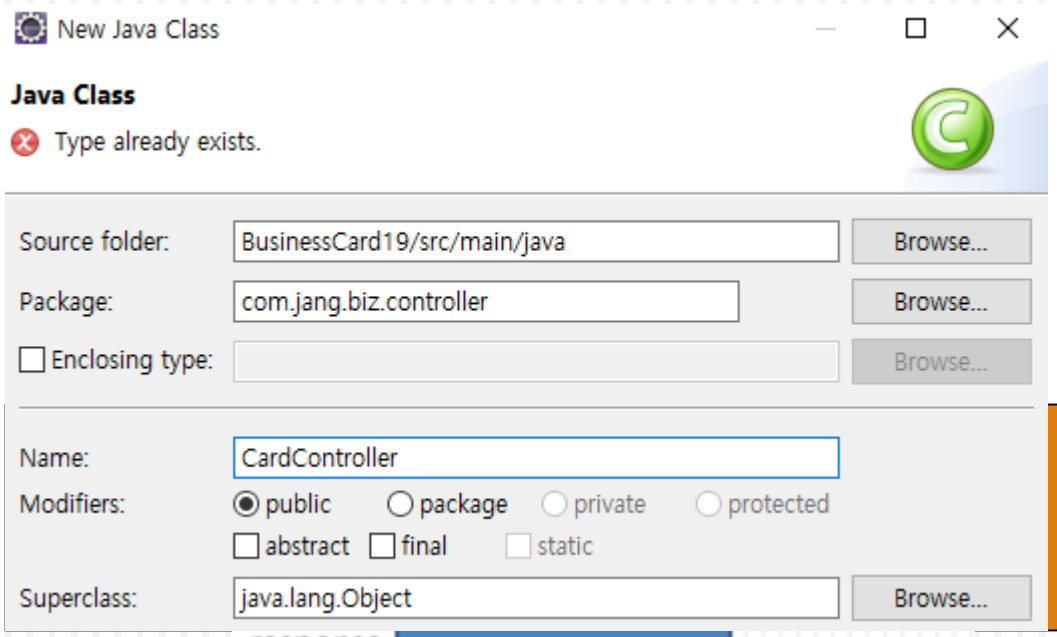
- 사용자 요청 (URL)을 처리할 메서드를 작성한다.
  - Handler mapping 메서드
  - service 객체와 의존관계 설정하고 메서드를 호출하여 요청 데이터 처리
- View 결정과 처리된 데이터 전달(model 객체)

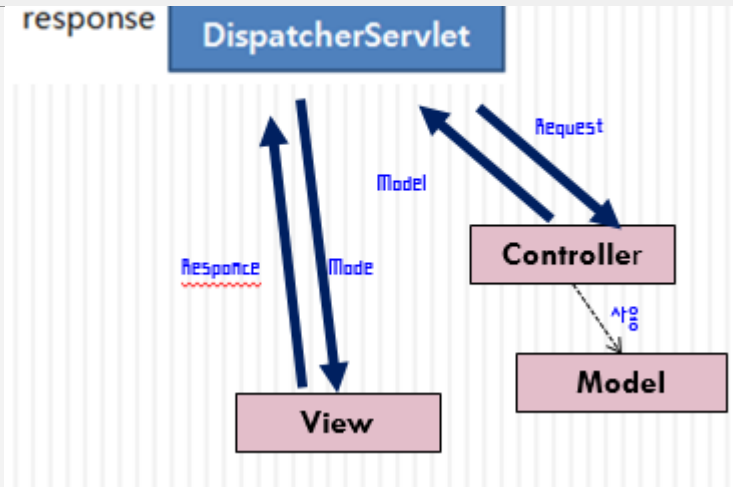


Controller 작성 (1)



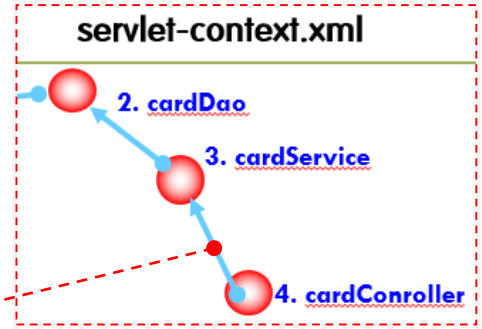
1. 우클릭 > new > class 생성



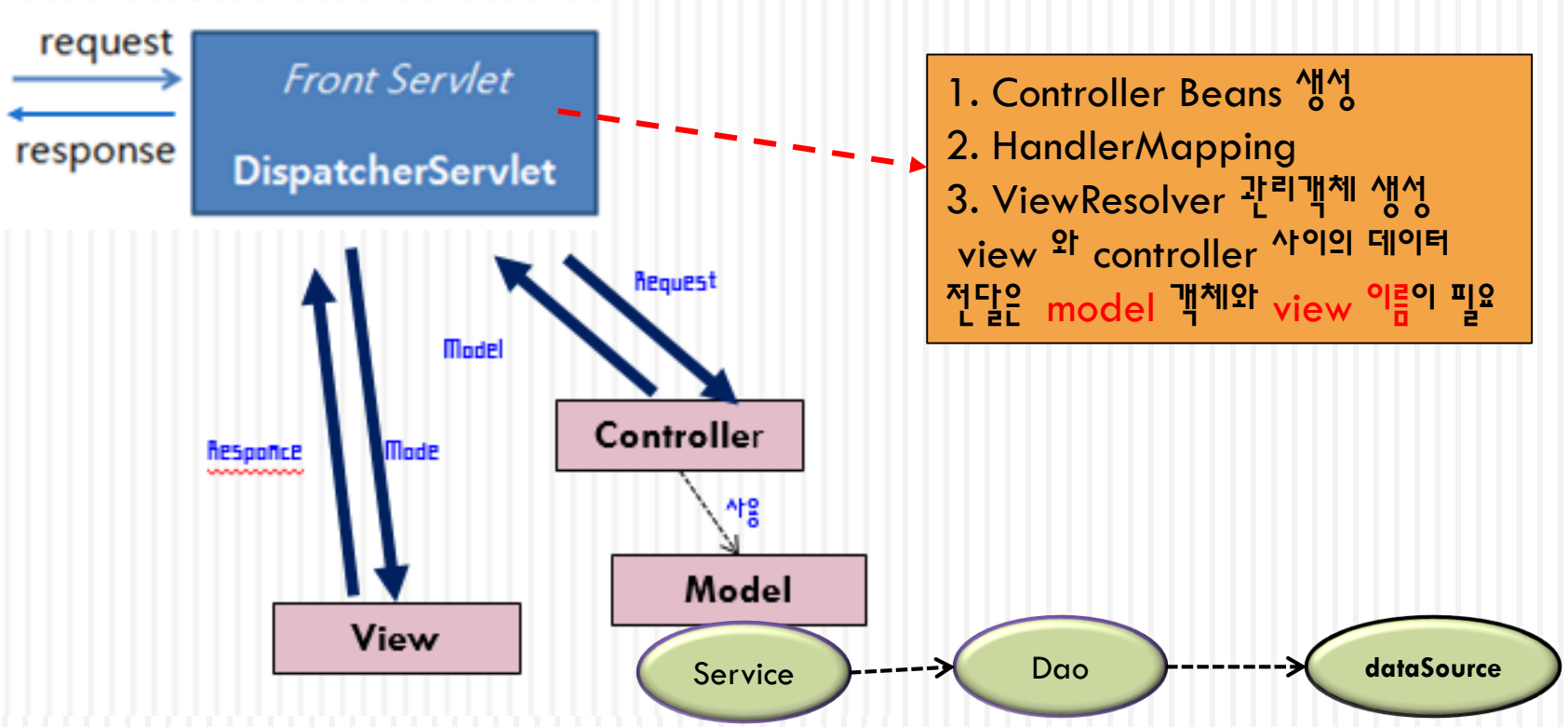


Controller 작성 (1)

```java
package com.jang.biz.controller;

import java.util.List;


@Controller(value = "cardController")//<context:component-scan>
public class CardController {

    @Resource(name = "cardService")
    private CardService cardService;

    @RequestMapping(value = "/list", method = RequestMethod.GET)
    public String list(Model model) throws Exception {

        List<Card> list = this.cardService.getCardList();
        model.addAttribute("list", list);

        return "CardList";
    }
```



Controller

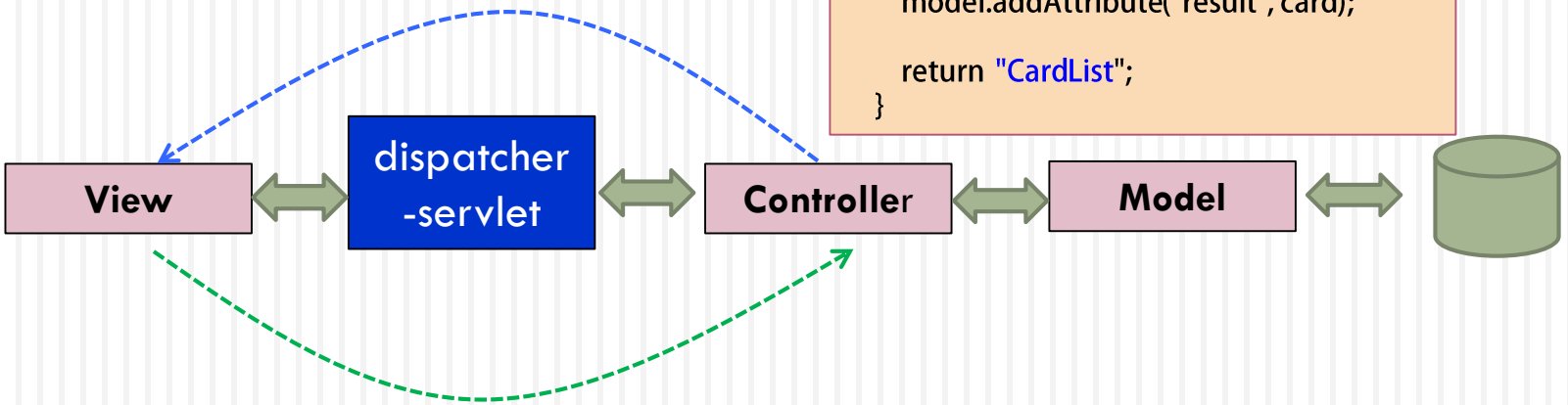

request
response
Front Servlet
DispatcherServlet
1. Controller Beans 생성
2. HandlerMapping
3. ViewResolver 관리객체 생성
view 와 controller 사이의 데이터
전달은 model 객체와 view 이름이 필요
Request
Model
Responce
Mode
Controller
사용
Model
View
Service
Dao
dataSource

Controller & View 사이 데이터 전달

```
public ModelAndView  list( ModelMap model ) throws Exception {
    model.addAttribute("list", list);
    return new ModelAndView("CardList", model);.
}
```

```
public  String list( ModelMap model ) {
    model.addAttribute("result", card);
    return "CardList";
}
```

ModelAndView 객체
Model, Map 객체



Card DTO 객체

과거에는 ModelAndView 객체에 view 이름과 model 객체를 담아 return 했으나
최근에는 view 이름만 return하면 model 은 자동으로 전송

View 설계

메시지 :

[신규]

Sample No	Title	Phone	Description	수정	삭제
1	양옥경	010-1234-4321	태진아 부인	수정	삭제
2	이현상	010-2234-2321	저현상 동생	수정	삭제
3	김수현	010-3234-3321	드라마 작가	수정	삭제
4	박상현	010-4234-4421	정치가	수정	삭제
5	강양원	010-5234-5421	장안대학교수	수정	삭제

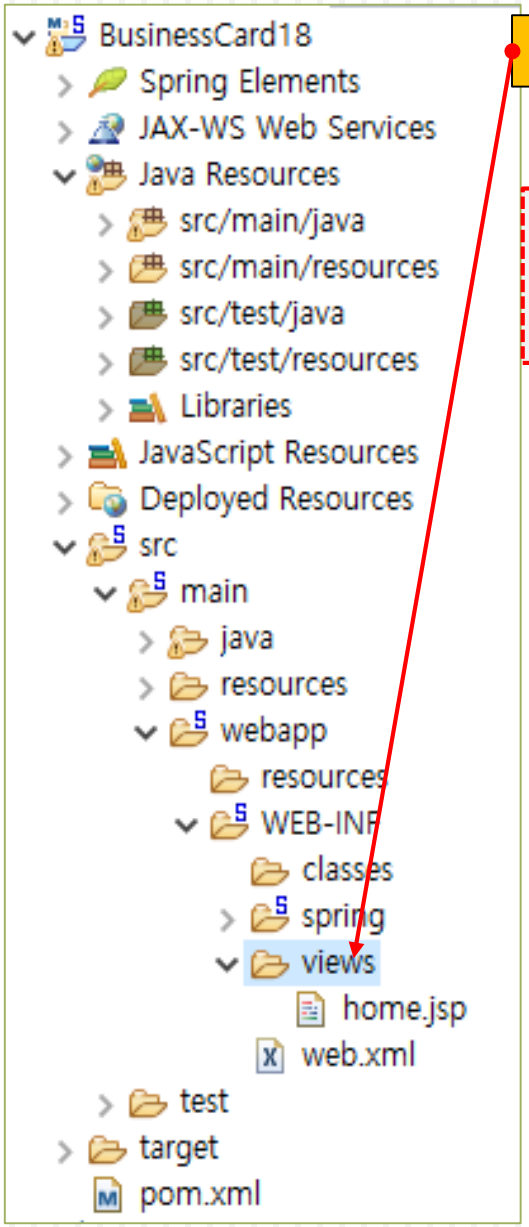
[Submit]

이름	
전화	
내용	

[Submit]

이름	양옥경
전화	010-1234-4321
내용	태진아 부인

CardList.jsp 생성 (1)



1. 우클릭 > New > jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"  pageEncoding="UTF-8"%>
<%@ taglib prefix="c"      uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags"%>

<!DOCTYPE html>
<html>
<head>
<meta charset="UTP-8">
<title>Card List</title>
</head>
<body>

</body>
</html>
```

저장객체 접근과 처리에 필요한 표준 사용자 정의 JSP 태그 라이브러리 (JSTL) 설정

JSTL(JavaServer Pages Standard Tag Library)
•JSTL은 사용자 정의 표준태그라이브러리)
•JSTL의 주요 강점 중 하나는 서블릿 컨텍스트에 저장된 데이타 같은 애플리케이션 데이타를 액세스 및 조작하는 쉬운 방법을 제공하는 간단한 EL을 사용할 수 있게함
•<%= sum %> ➔ ${sum}

CardList.jsp

```jsp
<%@ page language="java" contentType="text/html; charset=UTF-8"  pageEncoding="UTF-8"%>
<%@ taglib prefix="c"       uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="form"    uri="http://www.springframework.org/tags/form" %>

<!DOCTYPE html>
<html>
<head>
<meta charset="UTP-8">
<title>Card List</title>

<script type="text/javascript">

function fnCmdNew() {
    document.cardForm.bno.value = 0;
    document.cardForm.action = 'toForm.do';
    document.cardForm.submit();
}

function fnCmdEdit(sampleNo) {
    document.cardForm.bno.value = sampleNo;
    document.cardForm.action = 'toForm.do';
    document.cardForm.submit();
}

function fnCmdDelete(sampleNo) {
    document.cardForm.bno.value = sampleNo;
    document.cardForm.action = 'delete.do';
    document.cardForm.submit();
}
```

JSTL(JavaServer Pages Standard Tag Library)

- JSTL은 사용자 정의 표준태그라이브러리)
- JSTL의 주요 장점 중 하나는 서블릿 컨텍스트에 저장된 데이타 같은 애플리케이션 데이타를 액세스 및 조작하는 쉬운 방법을 제공하는 간단한 EL을 사용할 수 있게함

메시지 :

신규

Sample No	Title	Phone	Description	수정	삭제
1	양옥경	010-1234-4321	태진아 부인	수정	삭제
2	이현상	010-2234-2321	저현상 동생	수정	삭제
3	김수현	010-3234-3321	드라마 작가	수정	삭제
4	박상현	010-4234-4421	정치가	수정	삭제
5	강양원	010-5234-5421	장안대학교수	수정	삭제

CardList.jsp

```jsp
<body>
메시지 : ${message}<br />
<a href="#" onclick="fnCmdNew()">[신규]</a>

<form:form id="cardForm" name="cardForm" method="post">
    <input type="hidden" name="bno" />
    <input type="hidden" name="bno" />
    <table border="1">
        <thead>
            <tr>
                <th>번호</th>
                <th>이름</th>
                <th>전화</th>
                <th>기록</th>
                <th>수정</th>
                <th>삭제</th>
            </tr>
        </thead>
        <tbody>
            <c:forEach var="result" items="${list}" varStatus="status">
                <tr>
                    <td>${result.bno}</td>
                    <td>${result.bname}</td>
                    <td>${result.phone}</td>
                    <td>${result.description}</td>
                    <td><a href="#" onclick="fnCmdEdit('${result.bno}')">수정</a></td>
                    <td><a href="#" onclick="fnCmdDelete('${result.bno}')">삭제</a></td>
                </tr>
            </c:forEach>
        </tbody>
    </table>
</form:form>
</body>
</html>
```

```java
@RequestMapping(value = "/addsave", method = RequestMethod.POST)
public String add(@ModelAttribute("sampleVo") CardDto cardDto,
                  RedirectAttributes redirectAttributes) {
    this.cardService.add(cardDto);
    redirectAttributes.addFlashAttribute("message", "추가되었습니다.");

    return "redirect:list";
}
```

```java
@RequestMapping(value = "/list", method = RequestMethod.GET)
public String list(ModelMap model) throws Exception {

    List<CardDto> list = this.cardService.getAll();
    model.addAttribute("list", list);

    return "CardList";
}
```

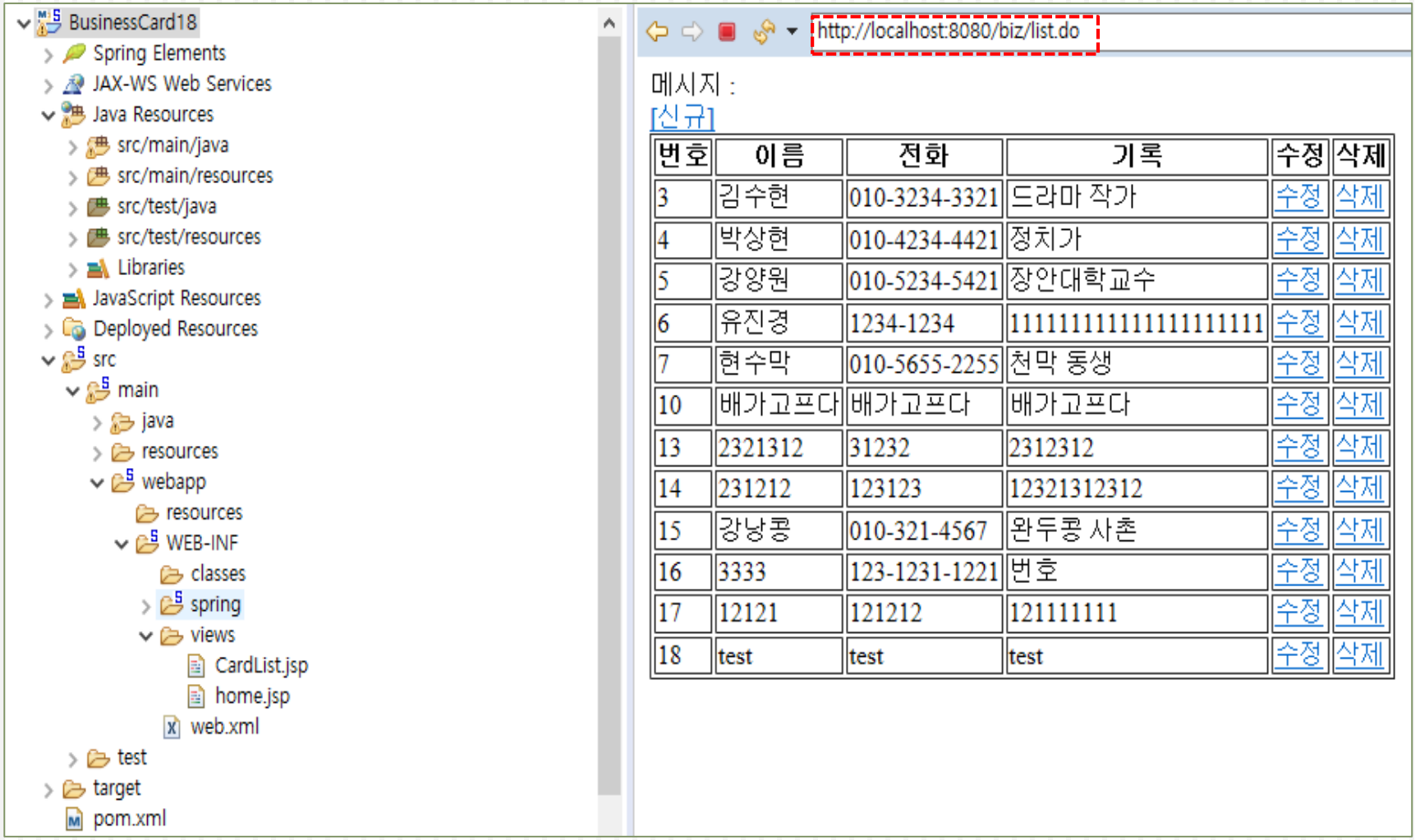
1	양옥경	010-1234-4321	태진아 부인	수정	삭제
2	이현상	010-2234-2321	저현상 동생	수정	삭제
3	김수현	010-3234-3321	드라마 작가	수정	삭제
4	박상현	010-4234-4421	정치가	수정	삭제
5	강양원	010-5234-5421	장안대학교수	수정	삭제

이벤트 처리함수 작성(4)

```
<head>
<meta charset="UTP-8">
<title>Card List</title>
<script type="text/javascript">

function fnCmdNew() {
    document.cardForm.bno.value = 0;
    document.cardForm.action = 'form.do';
    document.cardForm.submit();
}

function fnCmdEdit(sampleNo) {
    document.cardForm.bno.value = sampleNo;
    document.cardForm.action = 'form.do';
    document.cardForm.submit();
}

function fnCmdDelete(sampleNo) {
    document.cardForm.bno.value = sampleNo;
    document.cardForm.action = 'delete.do';
    document.cardForm.submit();
}

</script>
</head>
```

```
메시지 : ${message}<br />
<a href="#" onclick="fnCmdNew()">[신규]</a>

<form:form id="cardForm" name="cardForm" method="post">
    <input type="hidden" name="bno" />

    <table border="1">
        <c:forEach var="result" items="${list}" varStatus="status">
            <tr>
                <td>${result.bno}</td>
                <td>${result.bname}</td>
                <td>${result.phone}</td>
                <td>${result.description}</td>
                <td><a href="#" onclick="fnCmdEdit('${result.bno}')">수정</a
                <td><a href="#" onclick="fnCmdDelete('${result.bno}')">삭제<
            </tr>
        </c:forEach>
    </tbody>
</table>
```

중간 실행 및 결과 확인



번호	이름	전화	기록	수정	삭제
3	김수현	010-3234-3321	드라마 작가	수정	삭제
4	박상현	010-4234-4421	정치가	수정	삭제
5	강양원	010-5234-5421	장안대학교수	수정	삭제
6	유진경	1234-1234	11111111111111111111	수정	삭제
7	현수막	010-5655-2255	천막 동생	수정	삭제
10	배가고프다	배가고프다	배가고프다	수정	삭제
13	2321312	31232	2312312	수정	삭제
14	231212	123123	12321312312	수정	삭제
15	강낭콩	010-321-4567	완두콩 사촌	수정	삭제
16	3333	123-1231-1221	번호	수정	삭제
17	12121	121212	121111111	수정	삭제
18	test	test	test	수정	삭제

Controller 작성 (2)

```
@RequestMapping(value = "/toForm")
public String form(@RequestParam(value = "bno", required = false, defaultValue="0") int bno, Model model)
        throws Exception {

    if (bno > 0) {  //수정

        Card  card = this.cardService.getCards(bno);
        model.addAttribute("commandUrl", "editSave");
        model.addAttribute("card", card);

    } else {        //신규
        model.addAttribute("commandUrl", "addSave");
        model.addAttribute("card", new Card() );
    }

    return "CardView";
}
```

메시지 :

[신규]

Sample No	Title	Phone	Description	수정	삭제
1	양옥경	010-1234-4321	태진아 부인	수정	삭제
2	이현상	010-2234-2321	저현상 동생	수정	삭제
3	김수현	010-3234-3321	드라마 작가	수정	삭제
4	박상현	010-4234-4421	정치가	수정	삭제
5	강양원	010-5234-5421	장안대학교수	수정	삭제

Redirect : **rediect:${원하는 페이지주소}** : 쿼리스트링으로 넘겨주기 때문에 모두 보인다..
RedirectAttributes을 이용하면 사용자에게 보여주 싶지 않은 데이타는 노출하지 않고 넘겨 줄 수 있다.

[Submit]

이름	양옥경
전화	010-1234-4321
내용	태진아 부인

[Submit]

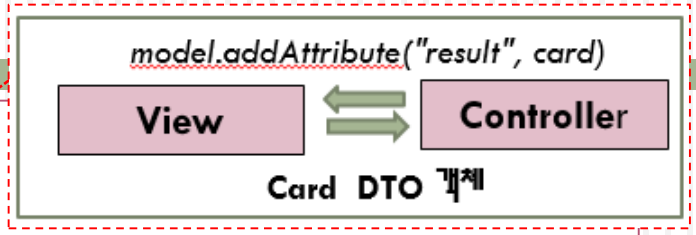
이름	
전화	
내용	

CardView.jsp

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Card View</title>

<script type="text/javascript">
function fnCmdSave() {
    document.cardView.action = '${commandUrl}';
    if (cardView.bname.value.length < 1){
        alert("이름을 입력하세요.");
        cardView.bname.focus();
        return false;
    }
    if (cardView.phone.value.length < 1){
        alert("전화번호를 입력하세요.");
        cardView.phone.focus();
        return false;
    }
    if (cardView.description.value.length < 1){
        alert("내용을 입력하세요.");
        cardView.description.focus();
        return false;
    }

    if (document.cardView.bno.value==""){
        document.cardView.bno.value=0;
    }

    document.cardView.submit();
}
</script>

</head>
```

```
if (bno > 0) {   //수정
    Card  card = this.cardService.getCards(bno);
    model.addAttribute("commandUrl", "editSave");
    model.addAttribute("card", card);
} else {         //신규
    model.addAttribute("commandUrl", "addSave");
    model.addAttribute("card", new Card() );
}
```

CardView.jsp(2)




```
<body>

<a href="#" onclick="fnCmdSave()">[Submit]</a>

<form:form  modelAttribute="card" name="cardView" method="post">
    <input type="hidden" id="bno" name="bno" value="${card.bno}" />

    <table border="1">
        <tr>
            <th>이름</th>
            <td><form:input path="bname" size="50" value="${card.bname}" />
            <span class="fieldError"><form:errors path="bname" /></span>
            </td>
        </tr>
         <tr>
            <th>전화</th>
            <td><form:input path="phone" size="50" value="${card.phone}" />
            <span class="fieldError"><form:errors path="phone" /></span>
            </td>
        </tr>
        <tr>
            <th>내용</th>
            <td><form:input path="description" size="50" value="${card.description}" />
            <span class="fieldError"><form:errors path="description" /></span>
            </td>
        </tr>
    </table>
</form:form>

</body>
</html>
```

[Submit]

이름	양옥경
전화	010-1234-4321
내용	태진아 부인

실행 및 확인

메시지 :

[신규]

번호	이름	전화	기록	수정	삭제
3	김수현	010-3234-3321	드라마 작가	수정	삭제
4	박상현	010-4234-4421	정치가	수정	삭제
5	강양원	010-5234-5421	장안대학교수	수정	삭제
6	유진경	1234-1234	11111111111111111111	수정	삭제
7	현수막	010-5655-2255	천막 동생	수정	삭제
10	배가고프다	배가고프다	배가고프다	수정	삭제
13	2321312	31232	2312312	수정	삭제
14	231212	123123	12321312312	수정	삭제
15	강낭콩	010-321-4567	완두콩 사촌	수정	삭제
16	3333	123-1231-1221	번호	수정	삭제
17	12121	121212	121111111	수정	삭제
18	test	test	test	수정	삭제

[Submit]

이름	
전화	
내용	

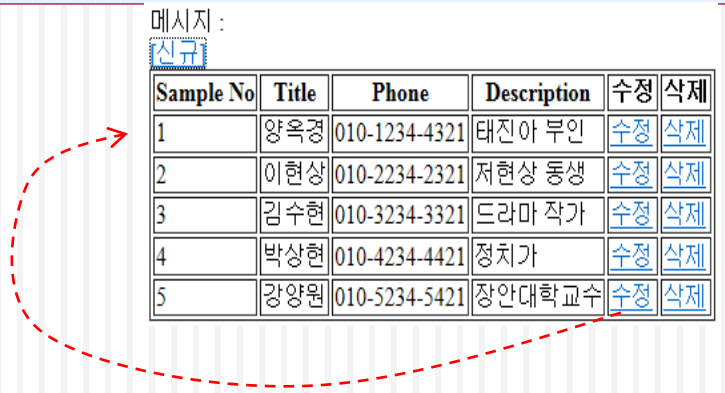
[Submit]

이름	김수현
전화	010-3234-3321
내용	드라마 작가

Controller 작성 (3)

```
@RequestMapping(value = "/addSave", method = RequestMethod.POST)
public String add( @Valid Card card, BindingResult result, RedirectAttributes rea) throws Exception {

    if (result.hasErrors()) {
        rea.addAllAttributes(result.getModel());
        return "redirect:list";
    }

    if (this.cardService.addCards(card) == 1) {
        rea.addFlashAttribute("message", "신규 등록되었습니다.");
        return "redirect:list";
    }
    else {
        rea.addFlashAttribute("message", "신규 등록에 실패하였습니다.  확인 후 다시 시도 하여주십시오");
        return "redirect:list";
    }

}
```

Spring controller 에서 특정 url로 redirect



Controller 작성 (4)

```
@RequestMapping(value = "/editSave", method = RequestMethod.POST)
public String update(@ModelAttribute("card") @Valid Card card, BindingResult result, RedirectAttributes rea) throws Exception  {

    if (result.hasErrors()) {
        rea.addAllAttributes(result.getModel());
        return "redirect:list";
    }

    if (this.cardService.updateCards(card)==1) {
        rea.addFlashAttribute("message", "수정 완료되었습니다.");
        return "redirect:list";
    }
    else {
        rea.addFlashAttribute("message", "수정 작업을 실패하였습니다. 확인 후 다시 시도 하여주십시오");
        return "redirect:list";
    }

}

@RequestMapping(value = "/delete", method = RequestMethod.POST)
public String delete(@RequestParam(value = "bno", required = false) int bno, RedirectAttributes rea) throws Exception  {

    if(this.cardService.deleteCards(bno)==1){
        rea.addFlashAttribute("message", "삭제되었습니다.");
        return "redirect:list";
    }
    else {
        rea.addFlashAttribute("message", "삭제하지 못하였습니다. 확인 후 다시 시도 하여주십시오.");
        return "redirect:list";
    }
}
```

실행 및 최종 결과 확인

메시지 : 신규등록 되었습니다.

[신규]

번호	이름	전화	기록	수정	삭제
3	김수현	010-3234-3333	인기 드라마 작가	수정	삭제
4	박상현	010-4234-4421	정치가	수정	삭제
5	강양원	010-5234-5421	장안대학교수	수정	삭제
6	유진경	1234-1234	11111111111111111111	수정	삭제
7	현수막	010-5655-2255	천막 동생	수정	삭제
10	배가고프다	배가고프다	배가고프다	수정	삭제
13	2321312	31232	2312312	수정	삭제
14	231212	123123	12321312312	수정	삭제
15	강낭콩	010-321-4567	완두콩 사촌	수정	삭제
16	3333	123-1231-1221	번호	수정	삭제
17	12121	121212	121111111	수정	삭제
18	현장감	0101-1111-2345	현장감독	수정	삭제

메시지 :

[신규]

Sample No	Title	Phone	Description	수정	삭제
1	양옥경	010-1234-4321	태진아 부인	수정	삭제
2	이현상	010-2234-2321	저현상 동생	수정	삭제
3	김수현	010-3234-3321	드라마 작가	수정	삭제
4	박상현	010-4234-4421	정치가	수정	삭제
5	강양원	010-5234-5421	장안대학교수	수정	삭제

@Service , @Repository를 이용한 bean 등록

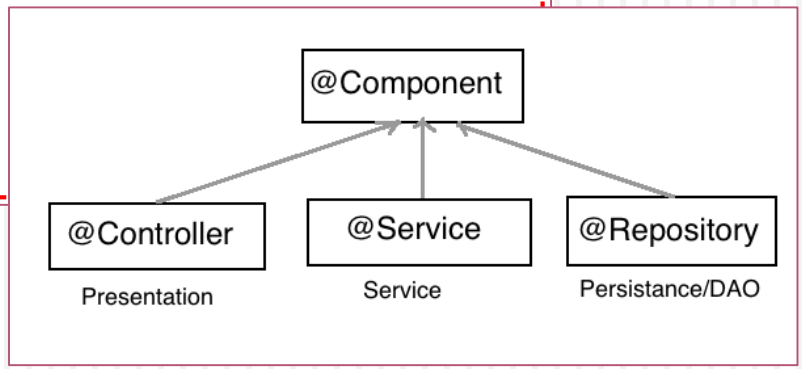
- Service클래스는 @Service, DAO 클래스는 @Repository 를 사용하면 <context:component-scan base-package="com.jang.biz" /> 에 의하여 컨테이너에 자동 bean으로 등록된다.

```
@Service(value = "cardService")
public class CardServiceImpl implements CardService {
```

- 이 경우 root-context.xml 에서 cardService bean을 생성할 필요가 없다.

root-context.xml

```
<!-- CardDao -->
 <bean id="cardDao" class="com.jang.biz.dao.CardDaoImpl" >
    <property name="dataSource">
        <ref bean="dataSource" />
    </property>
 </bean>

<!-- cardService bean 생성
 <bean id="cardService" class="com.jang.biz.service.CardServiceImpl" >
    <property name="cardDao">
        <ref bean="cardDao" />
    </property>
 </bean>
 -->
```



@Repository를 이용한 bean 등록

```xml
<!-- CardDao -->
<bean id="cardDao" class="com.jang.biz.dao.CardDaoImpl" >
    <property name="dataSource">
        <ref bean="dataSource" />
    </property>
</bean>
```

XML 기반 설정 메타데이터

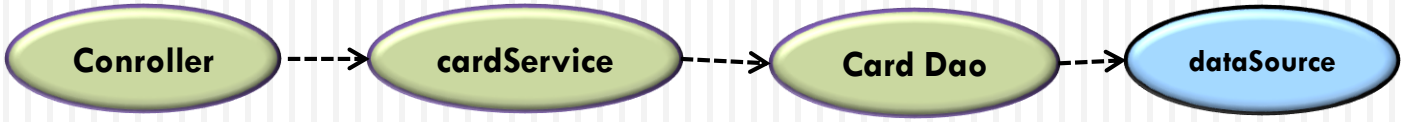
```java
public class CardDaoImpl implements CardDao {

    private JdbcTemplate jdbcTemplate;
    private NamedParameterJdbcTemplate jdbcTemplate2;

    public void setDataSource(DataSource dataSource) {
        this.jdbcTemplate = new JdbcTemplate(dataSource);
        this.jdbcTemplate2 = new NamedParameterJdbcTemplate(dataSource);
    }
```

애노테이션 기반 설정 설정 메타데이터

```java
@Repository(value="cardDao")
public class CardDaoImpl implements CardDao {

    private JdbcTemplate jdbcTemplate;
    private NamedParameterJdbcTemplate jdbcTemplate2;

    @Autowired
    public void setDataSource(DataSource dataSource) {
        this.jdbcTemplate = new JdbcTemplate(dataSource);
        this.jdbcTemplate2 = new NamedParameterJdbcTemplate(dataSource);
    }
```

@Resource, @Autowired를 이용한 DI 설정

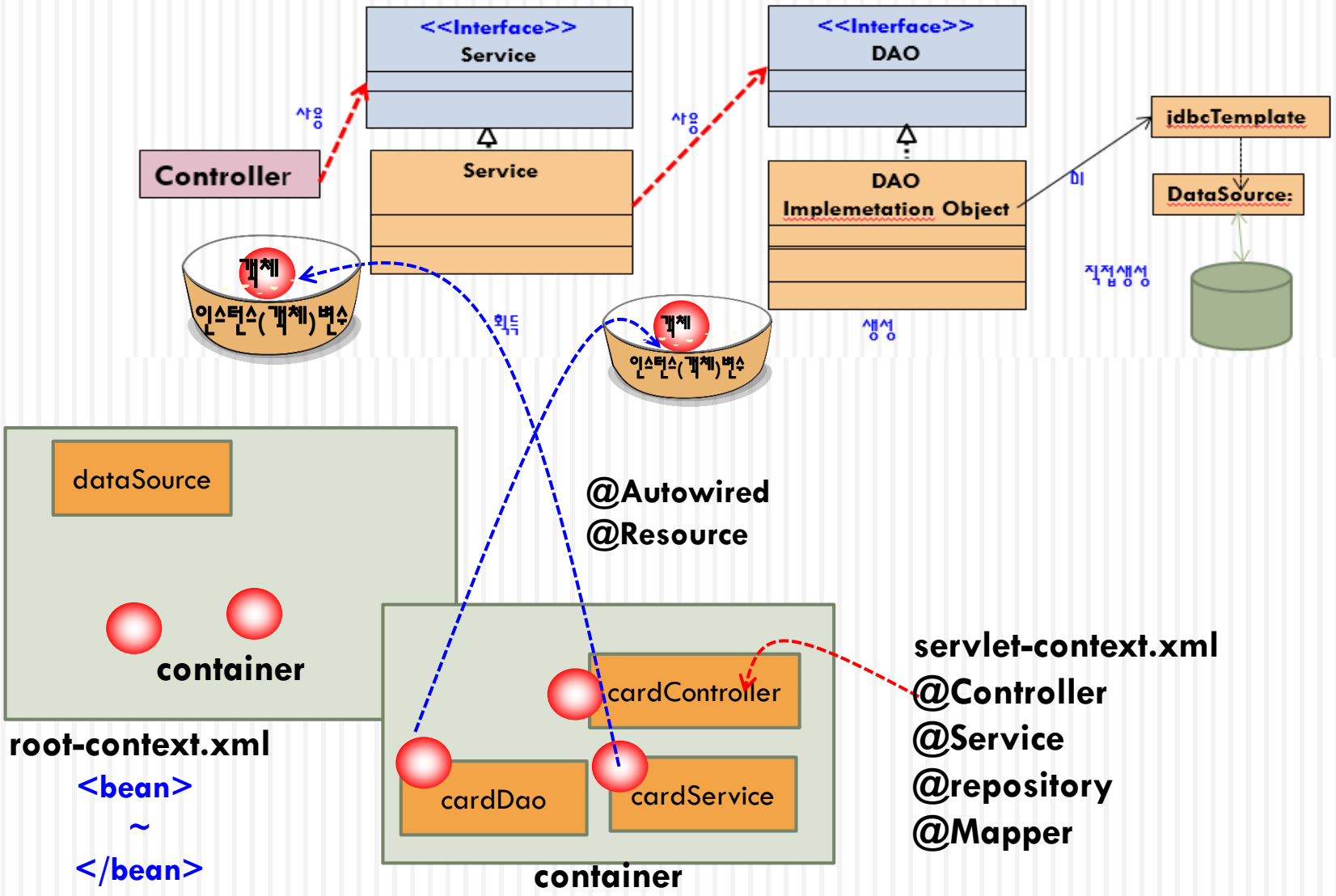
```java
@Controller(value = "cardController")
public class CardController {

    @Resource(name = "cardService")
    private CardService cardService;
```




```java
@Service(value = "cardService")
public class CardServiceImpl implements CardService {

    /*
    private CardDao cardDao;

    public void setCardDao(CardDao cardDaoImpl){
        this.cardDao = cardDaoImpl;
    }
    */

    @Resource(name = "cardDao")
    private CardDao cardDao;
```

Mybatis를 이용한 고객관리 구현

IoC 컨테이너와 객체 사용



Java 와 spring의 객체 사용

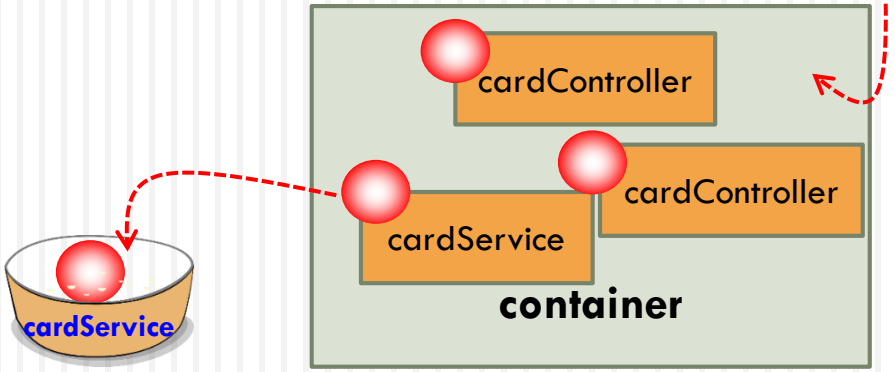
일반적인 java 프로그램 : =실행 도중 객체를 생성하여 영구적으로 연결

```
int   a =  10 ;
CardService  cardService  =  new CardService();
```

스프링 : 시작시 컨테이너에 객체를 생성해 놓고 필요할 때 일시적으로 연결 해서 사용

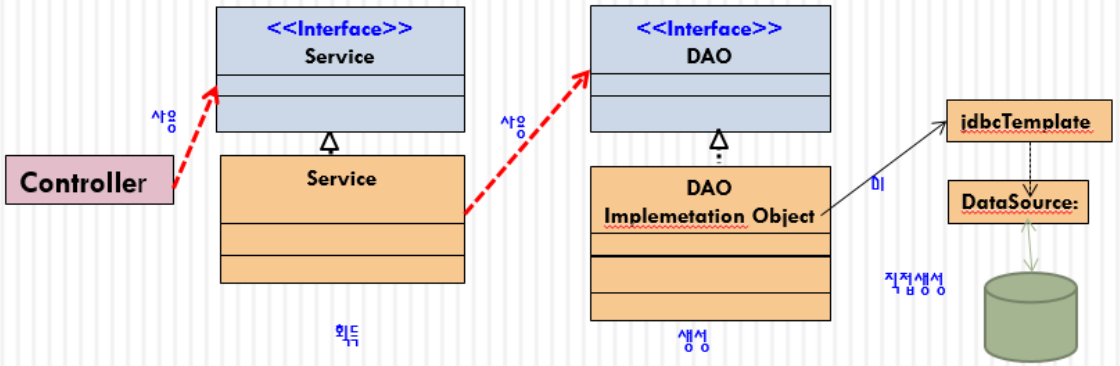
```
@Resource(value= 'cardService');
Private CardService  cardService

@Autowired
Private CardService  cardService
```

servlet-context.xml
@Controller
@Service
@repository
@Mapper



의존 관계 주입

```
public class CardController {

    private  int         a = 10
    private CardService  cardService = new CardServiceImpl() ;
    ------------------------------------------------------

    private CardService  cardService;
    public void setCardService(CardService cardServiceImpl)
    {
        this.cardService = cardServiceImpl;
    }

    ------------------------------------------------------
    @Resource(name = "cardServiceImpl")
    private CardService cardService; //@Autowired :

}
```



MyBatis Dao 객체 개발 순서

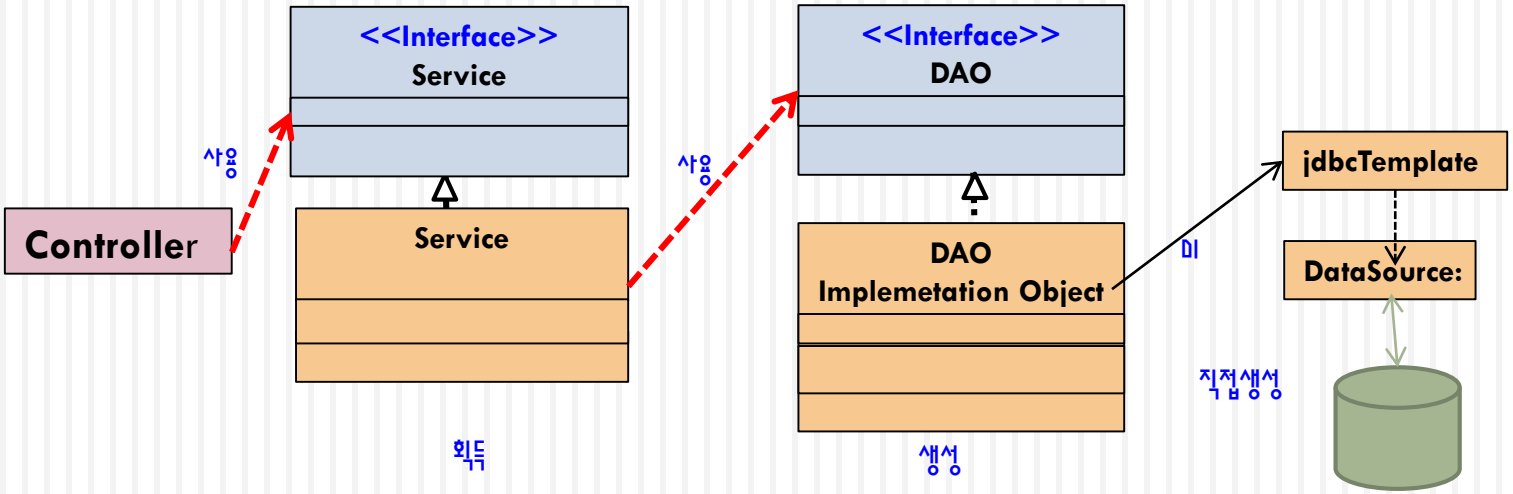
1. Dao(MAPPER) 인터페이스 or 클래스 설계
   - 메서드(함수)명, 입력(매개변수:parameter) ,출력(반환값)

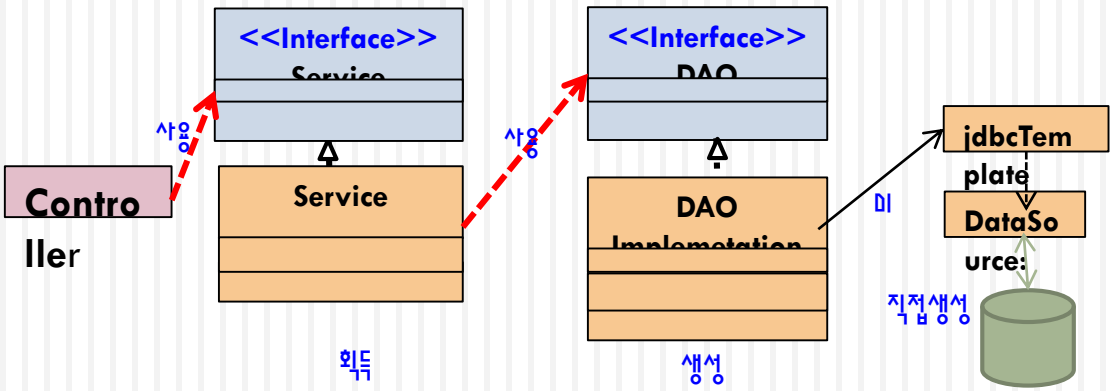
2. 메서드에 필요한 sql 문을 XML 파일로 작성
   - Namespace : 연동되는 인터페이스(클래스) java 파일 경로 확인
   - <XML 태그명, id=메서드명, resultType:반환값 , prameterType:매개변수>
     sql 문 ~
     </ XML 태그명 >

3. sqlSessionFactory 와 sqlSession 객체 생성 (root-context.xml)
   - xml 파일 경로 설정
   - 연계되는 인터페이스(클래스) java 파일 경로 확인

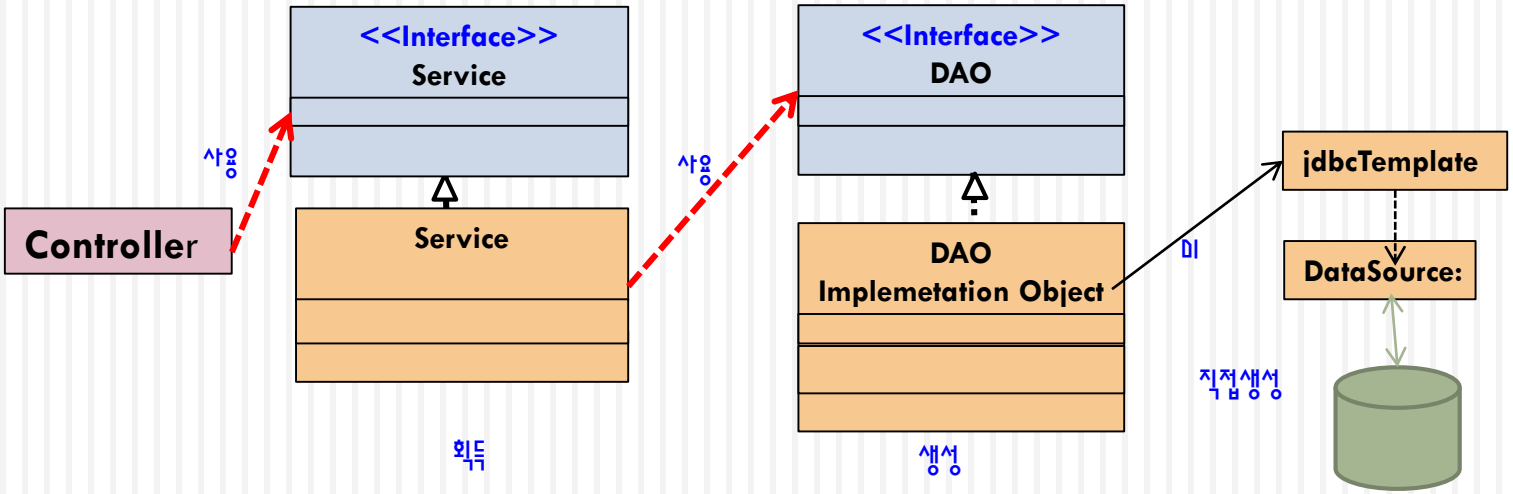
4. Dao(MAPPER) 클래스에서 sqlSession 객체 @Autowired
   - 각 메서드에서 필요한 sqlSession 객체의 메서드 작성

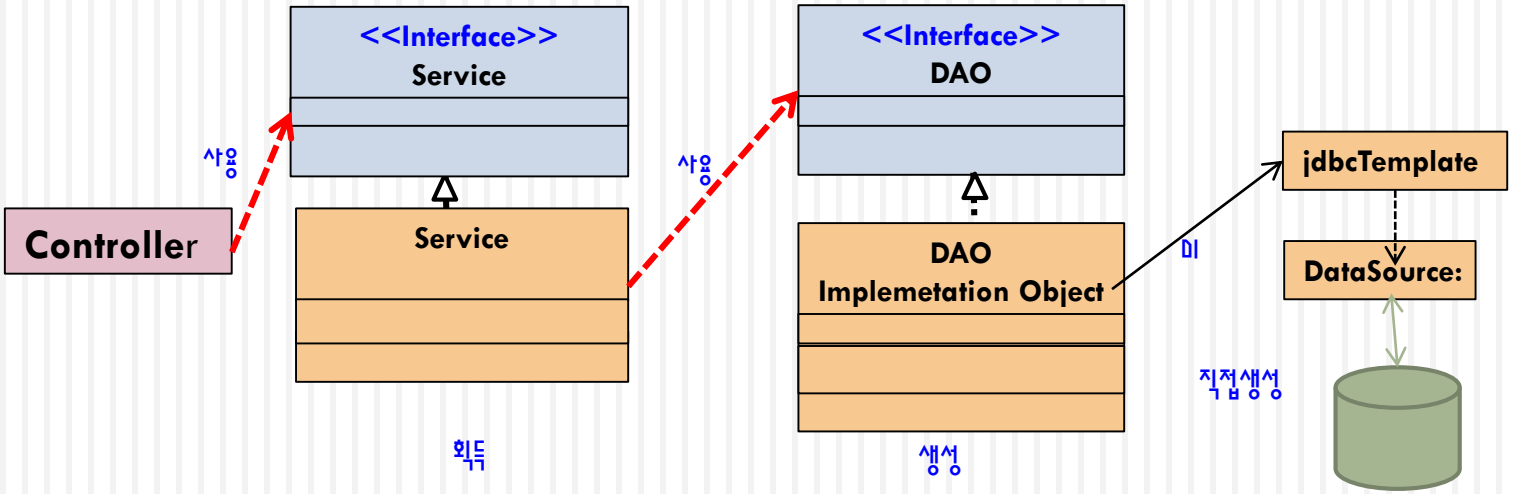
인터페이스 개념


<<Interface>>
Service
<<Interface>>
DAO
Controller
Service
DAO
Implemetation Object
jdbcTemplate
DataSource:
사용
사용
의
직접생성
획득
생성




```
private JdbcTemplate jdbcTemplate;
private NamedParameterJdbcTemplate jdbcTemplate2;

    public void setDataSource(DataSource dataSource) {
        this.jdbcTemplate = new JdbcTemplate(dataSource);
        this.jdbcTemplate2 = new NamedParameterJdbcTemplate(dataSource);
    }
```


<<Interface>>
Service
<<Interface>>
DAO
Controller
Service
DAO
Implemetation Object
jdbcTemplate
DataSource:
사용
사용
의
직접생성
획득
생성


<<Interface>>
Service
<<Interface>>
DAO
Controller
Service
DAO
Implemetation Object
jdbcTemplate
DataSource:
사용
사용
의
직접생성
획득
생성

```java
@Controller(value = "cardController
public class CardController {

@Resource(name = "cardService")

private CardService cardService; //@Autowired :


    @RequestMapping(value = "/list", method = RequestMethod.GET)//
    public String list(ModelMap model) throws Exception { //

        List<Card> list = this.cardService.listCards();
        model.addAttribute("list", list);

        return "CardList"; //��ȯ���  ��ʅ���  ������  view ��Ī, Map
������ model��  ����
    }
```