

MVC 패턴으로 변환

동의과학대학교 컴퓨터정보과
김진숙

프로그램 설치하기 – 사전 준비

- 사전 준비
 - 하나의 폴더 준비(폴더명: WEBPROJECT)
 - 이 폴더에 다음과 같이 필요한 파일 설치

WEBPROJECT		
이름	수정한 날짜	유형
apache-tomcat-9.0.65 ← 서버	2022-07-14 오후 4:28	파일 폴더
eclipse ← IDE(2022-06)	2022-06-09 오후 12:08	파일 폴더
jdk-17 ← 자바(jdk 17)	2021-08-06 오전 7:35	파일 폴더
jskim-workspace	2022-09-06 오후 10:32	파일 폴더

← 각자의 이름으로 워크스페이스폴더 작성(예 : honggildong)

프로그램 설치하기 - jdk

- OpenJdk설치하기

- <https://jdk.java.net/java-se-ri/17>

Java	버전	지원종료시기	지원비용	유지보수관리
Eclipse Temurin JDK(Adoptium JDK)	8u322(LTS)	2026년 5월	무료	자동설치대응
	11.0.14(LTS)	2024년 10월	무료	자동설치대응
	17.0.12(LTS)	미정	무료	자동설치대응
OpenJDK	17.0.12(LTS)	2031년 9월	무료	수동설치
OracleJRE	8u321(LTS)	2030년 12월	무료(일부 제외)	자동설치대응
OracleJDK	8u321(LTS)	2030년 12월	유료	
	11.0.14(LTS)	2026년 9월	유료	
	17.0.12(LTS)	2029년 9월	무료	자동설치대응

- 환경변수 설정

- JAVA_HOME
 - Path

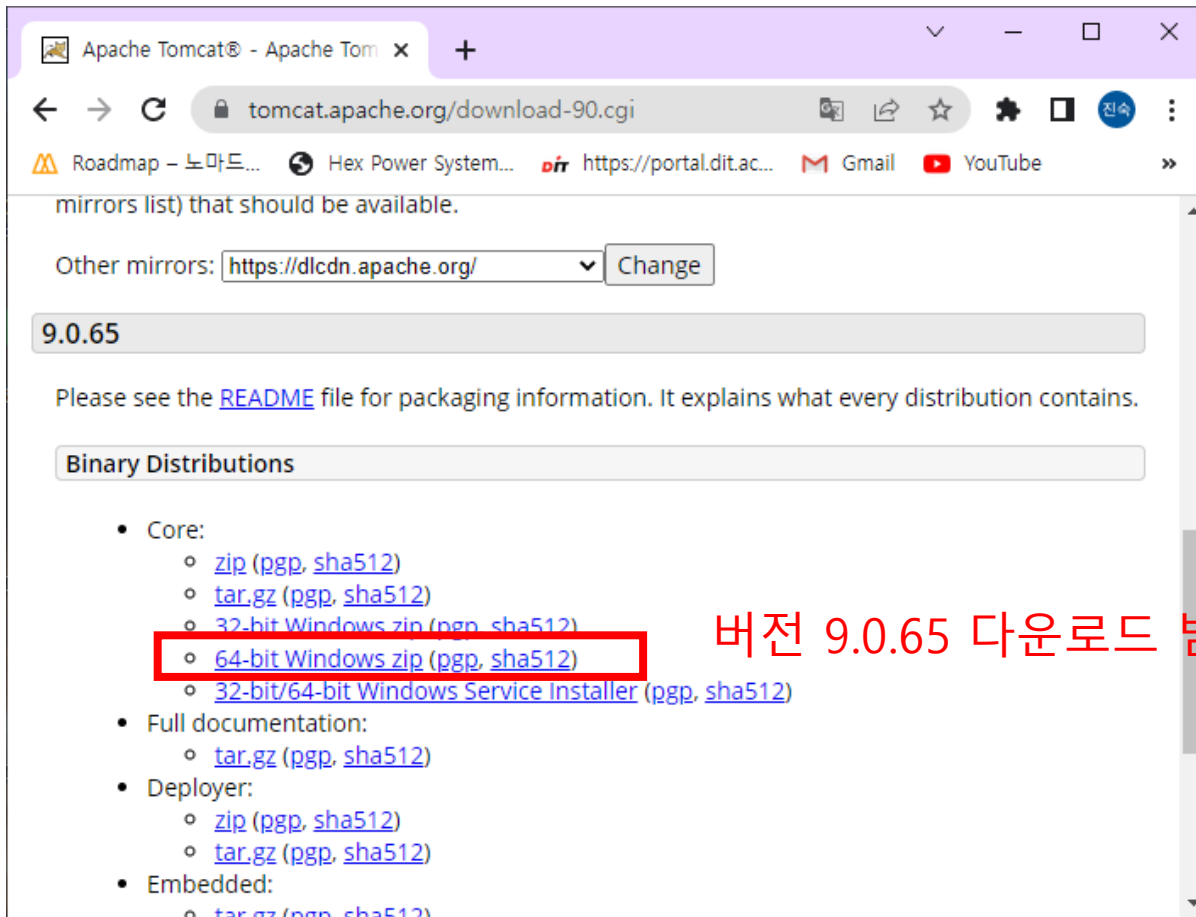
Java SE 18 = 62 (0x3E hex),
Java SE 17 = 61 (0x3D hex),
Java SE 16 = 60 (0x3C hex),
Java SE 15 = 59 (0x3B hex),
Java SE 14 = 58 (0x3A hex),
Java SE 13 = 57 (0x39 hex),
Java SE 12 = 56 (0x38 hex),
Java SE 11 = 55 (0x37 hex),
Java SE 10 = 54 (0x36 hex),^[4]
Java SE 9 = 53 (0x35 hex),^[5]
Java SE 8 = 52 (0x34 hex),
Java SE 7 = 51 (0x33 hex),
Java SE 6.0 = 50 (0x32 hex),
Java SE 5.0 = 49 (0x31 hex),
JDK 1.4 = 48 (0x30 hex),
JDK 1.3 = 47 (0x2F hex),
JDK 1.2 = 46 (0x2E hex),
JDK 1.1 = 45 (0x2D hex).

[출처]

https://en.wikipedia.org/wiki/Java_class_file

프로그램 설치하기 – apache tomcat

- tomcat 설치하기
 - <https://tomcat.apache.org/>

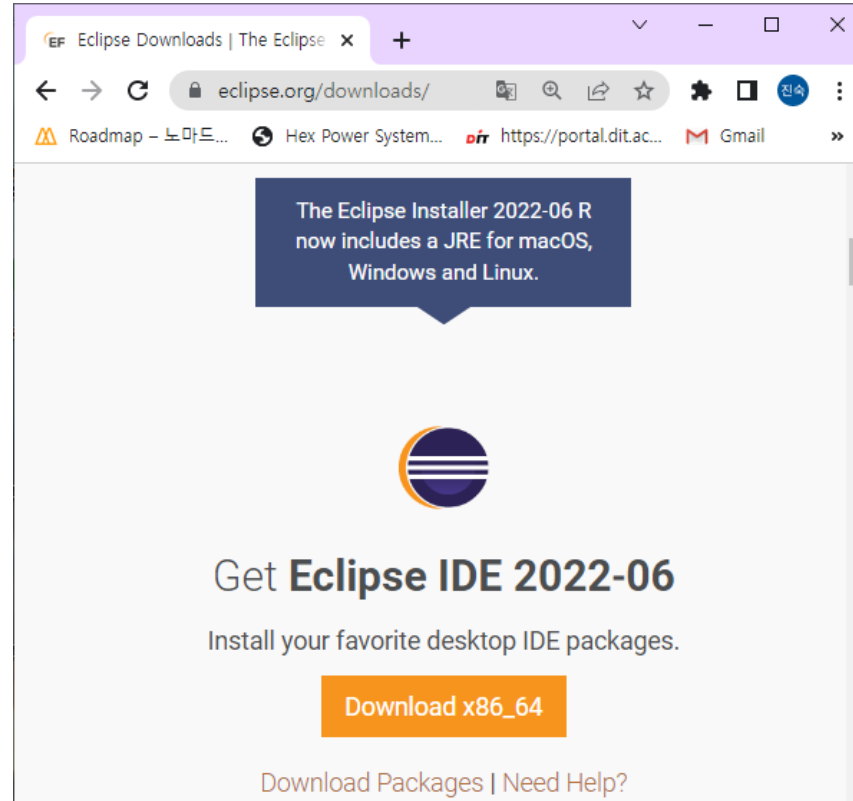


버전 9.0.65 다운로드 받기

- 환경변수 설정
CATALINA_HOME

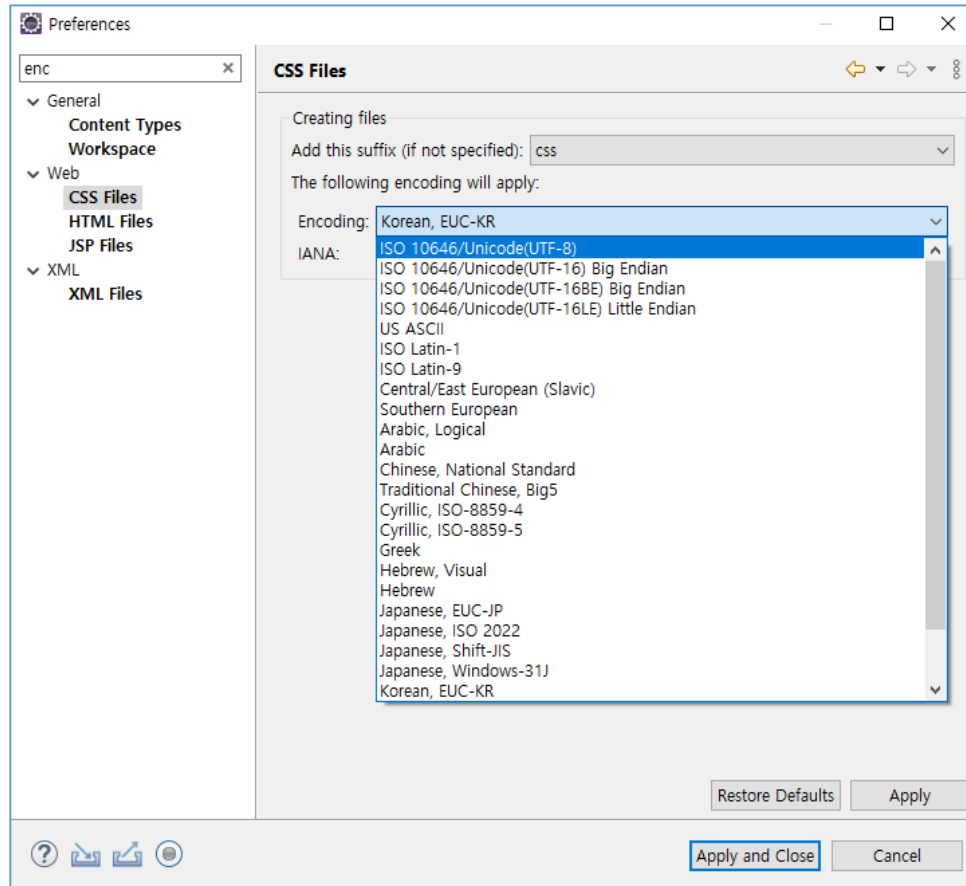
프로그램 설치하기 - eclipse

- Eclipse 설치하기
 - <https://www.eclipse.org/>
 - 최신버전 2022-06 설치
 - 보안 관련한 메시지에 확인하기



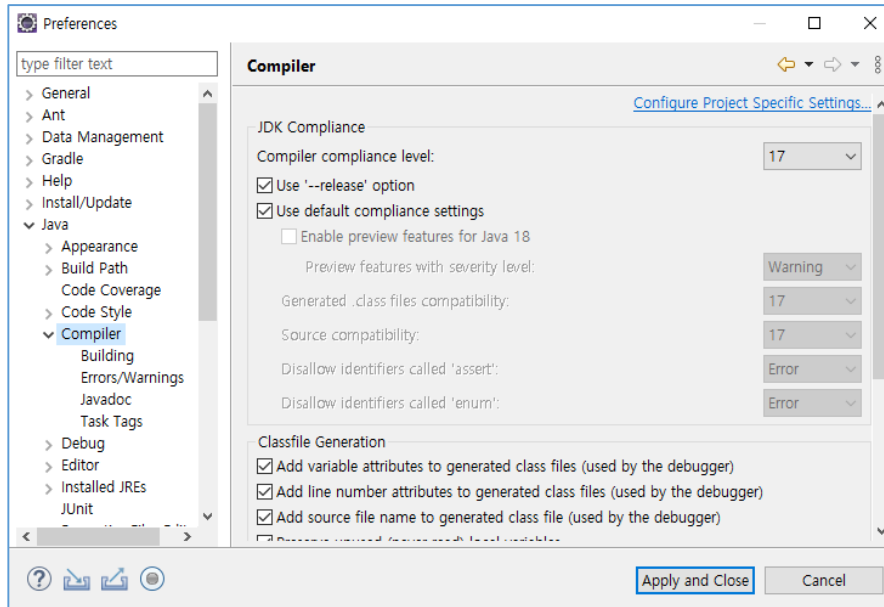
프로그램 설정하기 - eclipse

- 인코딩 설정(window - preferences)
 - Utf-8로 인코딩 방식 변경

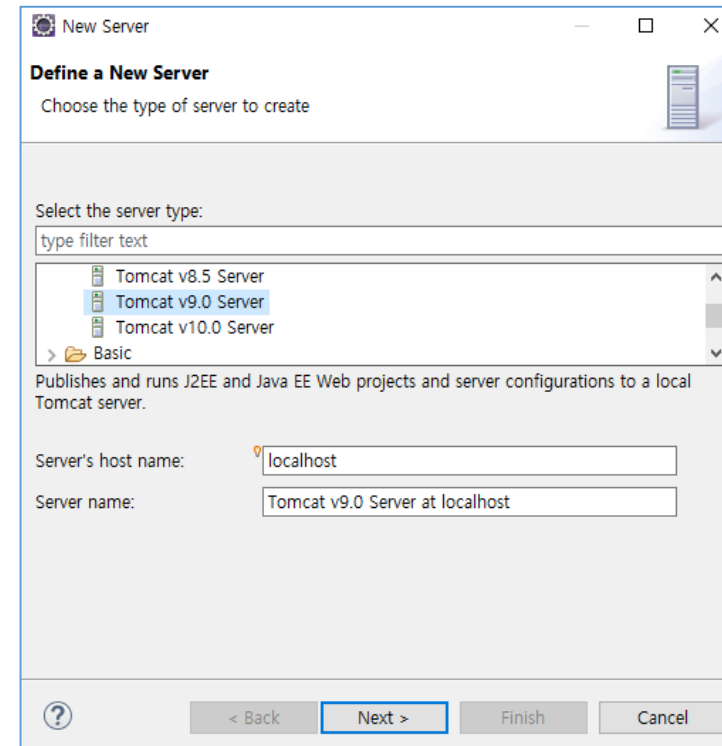


프로그램 설정하기 - eclipse

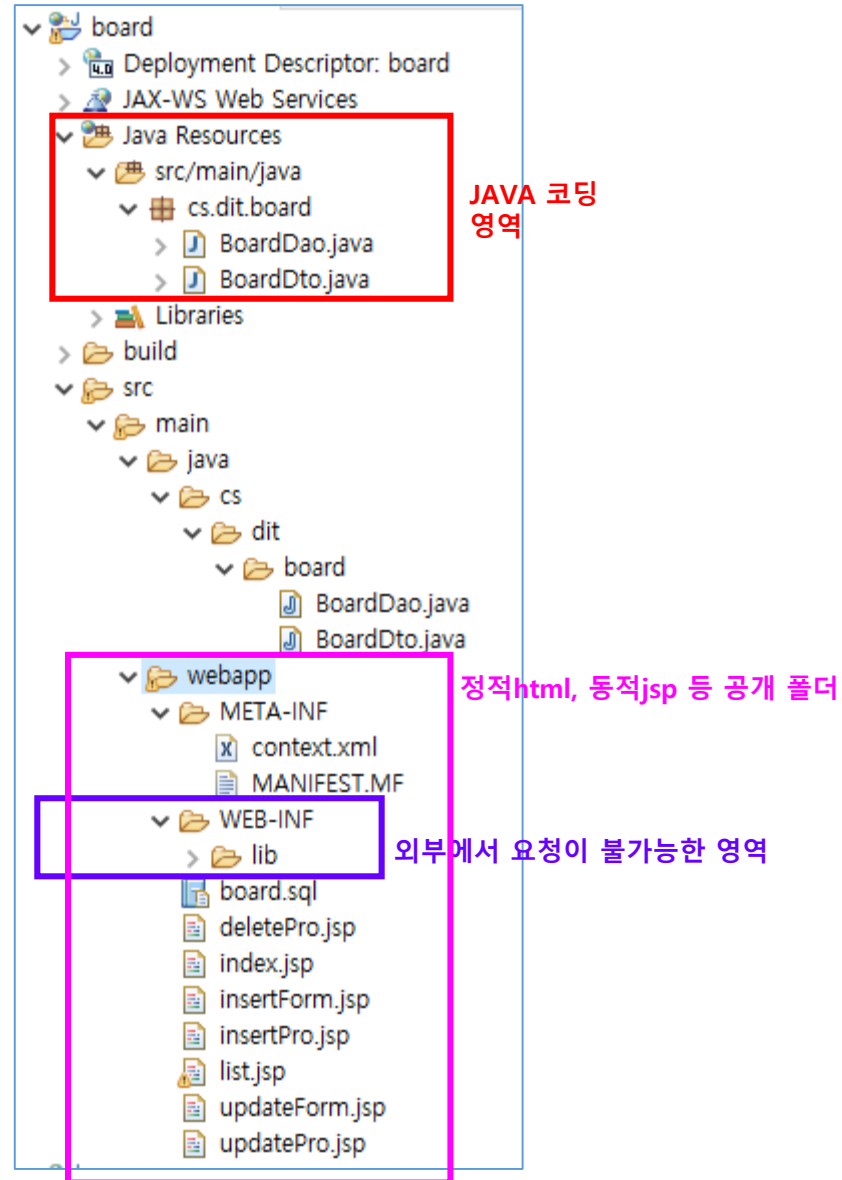
- Java 버전 확인(windows- preferences)



- 서버 설정

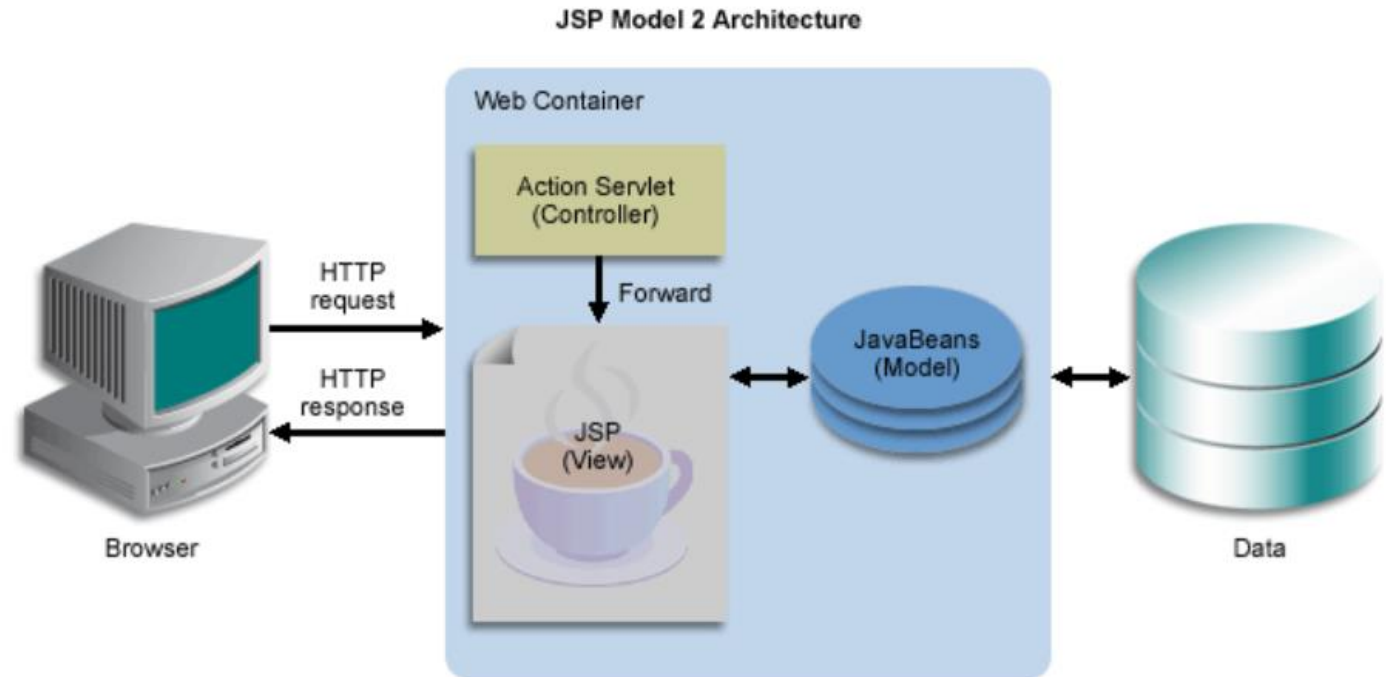


이클립스 자바 웹 디렉토리 구조



MVC 패턴

- 컨트롤러와 뷰가 물리적으로 분리된 방식
- model2 구조는 MVC 패턴을 웹 개발에 도입한 구조
- 구성요소
 - 서블릿(Controller)
 - JSP(View)
 - 자바빈(Model)

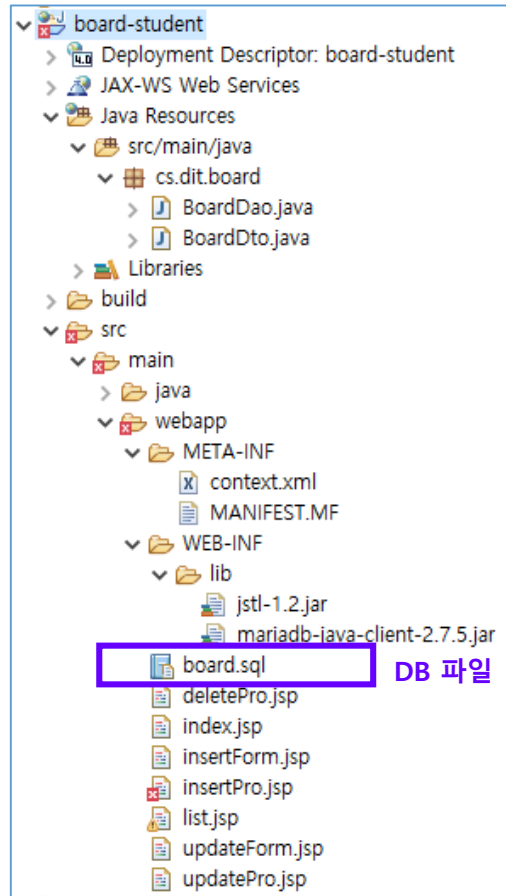


MVC 구성 요소

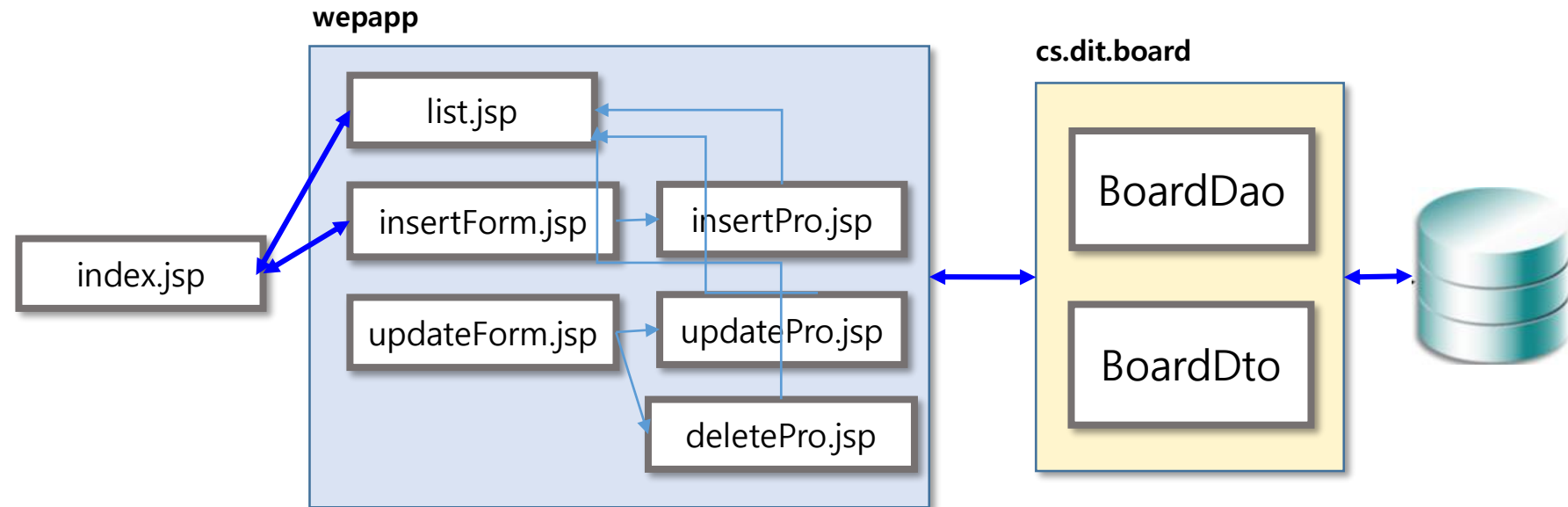
MVC 패턴	Model2	설명
Model	자바빈 POJO(Plain Old Java Object)	<ol style="list-style-type: none">1. 컨트롤러로부터 처리를 요청 받는다.2. 비즈니스 로직(DB연동 등)을 수행한다.3. 수행결과를 컨트롤러에 전달한다.
View	JSP	<ul style="list-style-type: none">• 클라이언트에 출력되는 화면 담당• 요청 결과 출력 뿐 아니라 컨트롤러에 요청을 보내기도 함• request, session객체에 저장된 정보를 토대로 화면 출력
Controller	Servlet	<ol style="list-style-type: none">1. 웹브라우저의 요청을 받는다.2. 웹브라우저의 요청을 분석한다.3. 분석된 요구를 바탕으로 필요한 로직을 처리하는 Model를 호출한다.4. 결과 출력 View인 JSP 페이지에 forward, redirect 로 출력을 위임한다.

사전 준비

1. board-student.war 파일 import 하기



[파일 구조]



사전 준비

2. DB 설치 : mariadb 설치

[05장. MariaDB\(MySQL\) 사용법과 필수 쿼리-22.pptx](#)

```
CREATE TABLE BOARD(  
  BCODE INT AUTO_INCREMENT PRIMARY KEY,  
  SUBJECT VARCHAR(100),  
  CONTENT TEXT,  
  WRITER VARCHAR(50),  
  REGDATE DATE  
);  
COMMIT;
```

```
INSERT INTO board(subject, content, writer, regdate) VALUES('안녕1', '반가워요1', 'gildong', SYSDATE());  
INSERT INTO board(subject, content, writer, regdate) VALUES('안녕2', '반가워요2', 'gildong2', SYSDATE());  
INSERT INTO board(subject, content, writer, regdate) VALUES('안녕3', '반가워요3', 'gildong3', SYSDATE());  
INSERT INTO board(subject, content, writer, regdate) VALUES('안녕4', '반가워요4', 'gildong4', SYSDATE());  
COMMIT;
```

사전 준비

3. DBCP API 준비

```
<!-- DBCP 설정 -->  
<Resource name = "jdbc/jskim"  
auth = "Container"  
type="javax.sql.DataSource"  
driverClassName = "org.mariadb.jdbc.Driver"  
username="jinsook"  
password="1111"  
url="jdbc:mariadb://localhost:3306/jinsookdb"  
maxWait = "5000"  
>
```

DBCP를 찾는 이름 설정.
자신의 이름으로 변경할 것

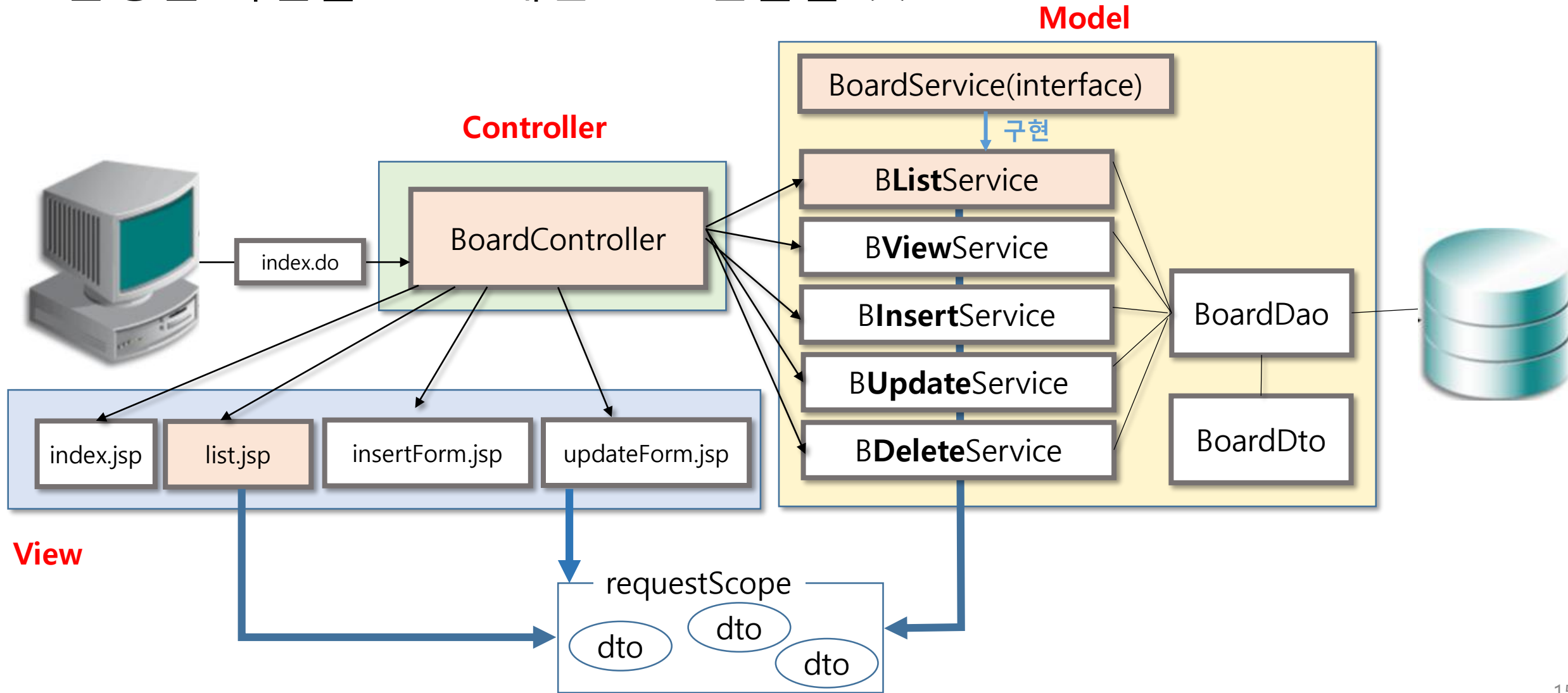
각자의 데이터베이스 설정으로 변경할 것

실습 1 – jsp 프로젝트 완성

- 전체 파일에 8개의 코드 문제를 해결할 것

실습 2 - MVC 변환

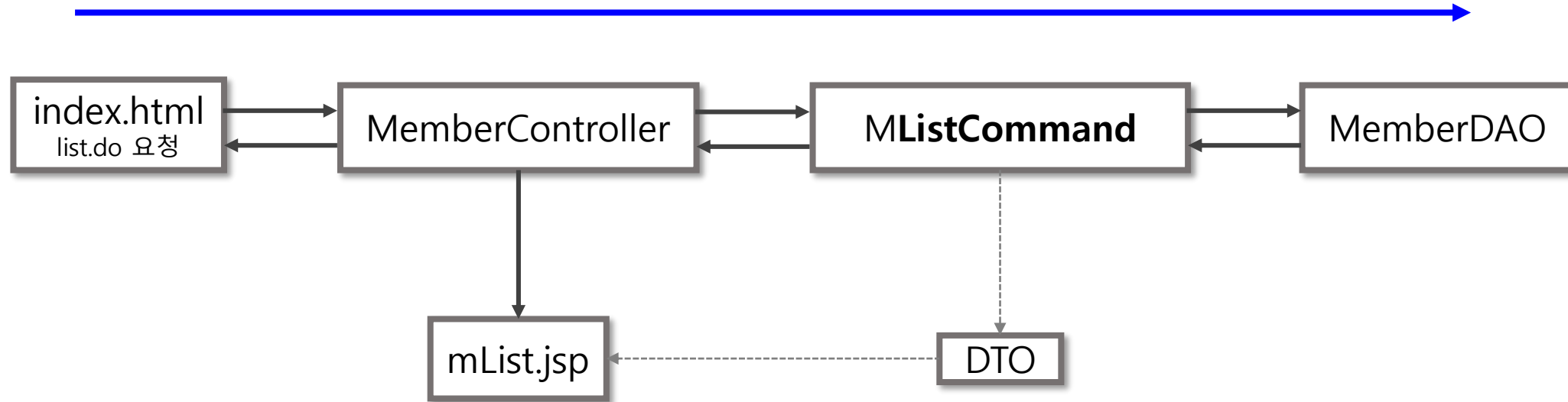
- 완성된 파일을 MVC 패턴으로 변환할 것



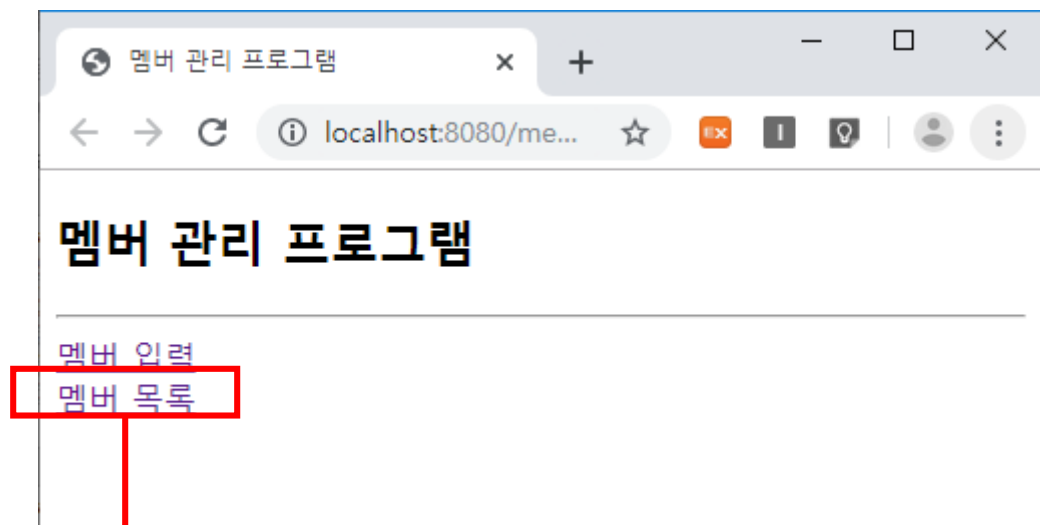
이전 자료 : member 관리

1. 멤버 목록 보기

구현 순서/ 테스트 순서



멤버 목록 보기 - 실행 화면



id	name	email	joinDate
jin	이요신	jins@gmail.com	2019. 11. 5
lee	이순신	lee@gmail.com	2019. 11. 5
song	이주신	song@gmail.com	2019. 11. 5

[홈으로](#) [멤버 등록](#)

멤버 목록 보기 - 구현 순서

중요!!!!

매 단계마다 프로그램을 실행시켜
디버깅을 하면서 구현하도록 한다.

1. 첫 페이지를 작성한다.
 - 파일명 : **index.html**
2. 데이터를 저장하는 DTO 클래스를 작성한다. - **Model**
 - 파일명 : **MemberDTO.java**
3. 클라이언트의 요청을 분석, 필요한 로직 처리를 하는 클래스를 호출하고 그 결과에 따라 이동할 페이지를 제어하는 서블릿 클래스를 작성한다. - **Controller**
 - 파일명 : **MemberController.java**
4. 클라이언트 요청에 따라 로직처리를 하는 Service클래스를 작성한다. - **Model**
 - 파일명 : **MListCommand.java** (MCommand 인터페이스를 구현)
5. DBMS에서 레코드셋을 검색하여 DTO에 저장하는 DAO클래스 메소드(list())를 작성한다. - **Model**
 - 파일명 : **MemberDAO.java**
6. 해당 결과를 화면 출력할 JSP 페이지를 작성한다. - **View**
 - 파일명 : **mList.jsp**

index.html

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1">
6   <title>멤버 관리 프로그램</title>
7 </head>
8 <body>
9   <h2>멤버 관리 프로그램</h2>
10  <hr>
11  <a href="insertForm.do">멤버 입력</a><br>
12  <a href="list.do">멤버 목록</a><br>
13 </body>
14 </html>
```

MemberDTO

```
1 package cs.dit.dto;
2
3 import java.sql.Date;
4
5 public class MemberDTO{
6
7     private String id;
8     private String pwd;
9     private String name;
10    private String email;
11    private Date joinDate;
12
13    public String getId() {
14        return id;
15    }
16    public void setId(String id) {
17        this.id = id;
18    }
19    public String getPwd() {
20        return pwd;
21    }
22    public void setPwd(String pwd) {
23        this.pwd = pwd;
24    }
25    public String getName() {
26        return name;
```

```
27    }
28    public void setName(String name) {
29        this.name = name;
30    }
31    public String getEmail() {
32        return email;
33    }
34    public void setEmail(String email) {
35        this.email = email;
36    }
37    public Date getJoinDate() {
38        return joinDate;
39    }
40    public void setJoinDate(Date joinDate) {
41        this.joinDate = joinDate;
42    }
43 }
```

MemberController

- 클라이언트 요청을 받고 분석하여 해당 로직을 처리

```
1 package cs.dit.controller;
2
3 import java.io.IOException;
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19 @WebServlet("*.do")
20 public class MemberController extends HttpServlet {
21     private static final long serialVersionUID = 1L;
22
23
24     protected void doHandle(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
25         request.setCharacterEncoding("utf-8");
26         String viewPage = null;
27         MCommand command = null;
28
29         String uri = request.getRequestURI(); //uri : /member-mvc/insert.do
30         String com = uri.substring(uri.lastIndexOf("/") + 1, uri.lastIndexOf(".do")); //command : insert
31
32         if (com != null && com.trim().equals("list")) {
33             command = new MListCommand();
34             command.execute(request, response);
35             viewPage = "mList.jsp";
36
37             RequestDispatcher rd = request.getRequestDispatcher(viewPage);
38             rd.forward(request, response);
39         }
40
41     }
42
43     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
44         doHandle(request, response);
45     }
46
47     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
48         doHandle(request, response);
49     }
50
51 }
```

MCommand 인터페이스

- 인터페이스(Interface) 역할은?
 - 인터페이스로 협업자들이 서로 메소드 규약을 만들어서 공유하면 각자의 구현하는 방식에 덜 영향을 받으면서 애플리케이션을 구축할 수 있음
 - 한 객체가 인터페이스를 구현하여 사용할 때 그 객체는 인터페이스의 메소드들을 반드시 구현해야 함(컴파일 오류)
 - 인터페이스는 달리 몸통을 갖춘 일반 메서드 또는 멤버 변수를 구성원으로 가질 수 없음
- 인터페이스의 특징
 - 하나의 클래스가 여러 개의 인터페이스 구현 가능
 - 인터페이스도 상속 가능
 - 인터페이스의 멤버는 반드시 public

MCommand 인터페이스

- 접미어를 Command를 가지고 있는 객체들은 다음의 인터페이스를 구현
- 반드시 execute 메소드를 구현하여야 한다.

```
1 package cs.dit.command;
2
3 import java.io.IOException;
4
5 import javax.servlet.ServletException;
6 import javax.servlet.http.HttpServletRequest;
7 import javax.servlet.http.HttpServletResponse;
8
9 public interface MCommand {
10     public void execute(HttpServletRequest request, HttpServletResponse response)
11         throws ServletException, IOException;
12 }
```


MListCommand

```
1 package cs.dit.command;
2
3+ import java.io.IOException;
12
13 public class MListCommand implements MCommand{
14
15-     @Override
16     public void execute(HttpServletRequest request, HttpServletResponse response) throws
17         //1. DB 작업을 위해 DB처리를 하고 있는 MemberDAO 객체를 생성한다.
18         MemberDAO dao = new MemberDAO();
19
20         //2. DB에서 가져온 ArrayList 내의 MemberDTO 객체들을 dtos 변수에 저장한다.
21         ArrayList<MemberDTO> dtos = dao.list();
22
23         //3. dtos를 request scope에 저장하여 View가 화면출력할 수 있도록 준비한다.
24         request.setAttribute("dtos", dtos);
25     }
26 }
```

MemberDAO

- 생성자에서는 DBCP에서 DataSource를 가져와 Connection 객체생성 준비
- Connection 객체 해제

```
1 package cs.dit.dao;
2
3 import java.sql.Connection;
15
16 public class MemberDAO {
17     private DataSource ds;
18     private Connection con;
19     private PreparedStatement pstmt;
20     private ResultSet rs;
21
22     //생성자에서 jdbc/mvc 객체를 찾아 DataSource 로 받는다.
23     public MemberDAO() {
24         try {
25             Context initContext = new InitialContext();
26             Context envContext = (Context) initContext.lookup("java:/comp/env");
27             ds = (DataSource)envContext.lookup("jdbc/mvc");
28
29             }catch(Exception e) {
30                 e.printStackTrace();
31             }
32         }
33     //Connection 해제를 위한 메소드
34     public void close() {
35         try {
36             if(con !=null) {
37                 con.close();
38                 con=null;
39             }
40             }catch(SQLException e) {
41                 e.printStackTrace();
42             }
43     }
```

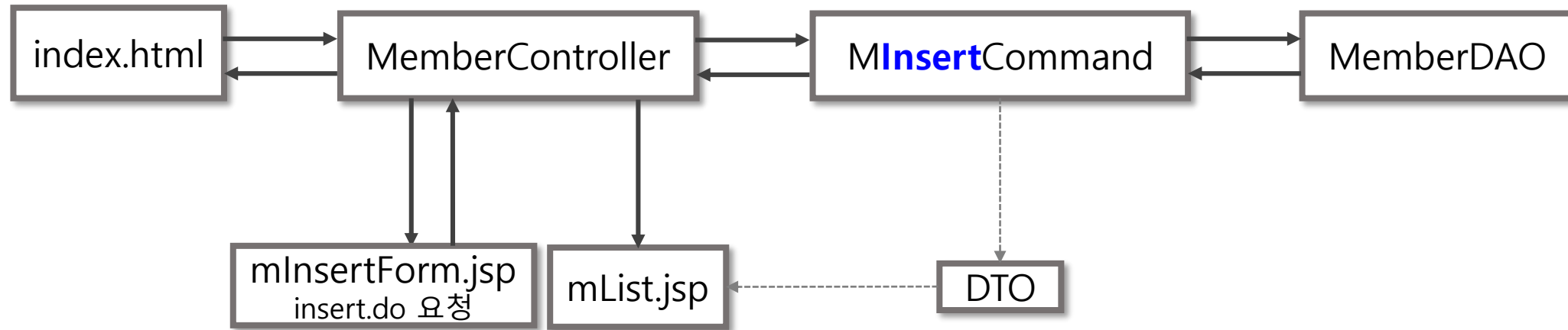
MemberDAO – list() 메소드

```
45 // 전체 멤버 목록보기
46 public ArrayList<MemberDTO> list(){
47     String sql = "select * from member";
48
49     ArrayList<MemberDTO> dtos = new ArrayList<MemberDTO>(); //DB처리 결과를 MemberDTO에 담아 ArrayList로 만들기 위해
50     try {
51         con = ds.getConnection();
52         pstmt = con.prepareStatement(sql);
53         rs = pstmt.executeQuery();
54
55         while (rs.next()) { //DB결과를 ResultSet에서 한행씩 추출하여 MemberDTO로 만든다.
56             MemberDTO dto = new MemberDTO();
57             dto.setId(rs.getString("id"));
58             dto.setPwd(rs.getString("pwd"));
59             dto.setName(rs.getString("name"));
60             dto.setEmail(rs.getString("email"));
61             dto.setJoinDate(rs.getDate("joinDate"));
62             dtos.add(dto); //MemberDTO객체를 ArrayList에 추가한다.
63         }
64         rs.close(); pstmt.close();
65     } catch (SQLException e) {
66         e.printStackTrace();
67     } finally {
68         close();
69     }
70     return dtos;
71 }
```

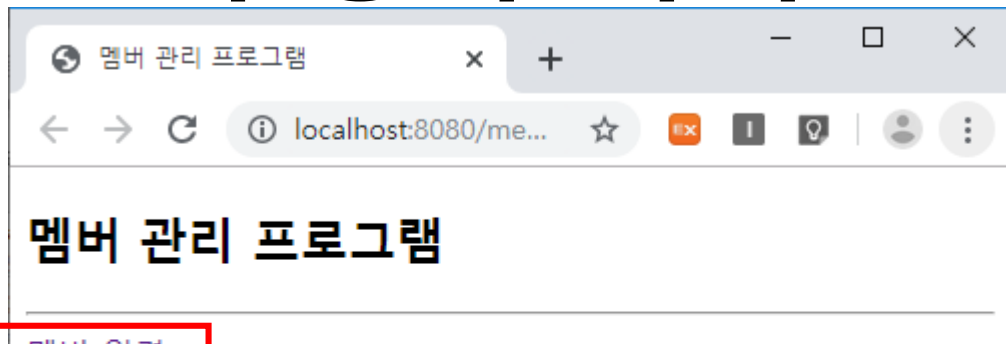
mList.jsp

```
1 <%@ page language="java"
2   pageEncoding="UTF-8"
3   contentType="text/html; charset=UTF-8"
4   %>
5 <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
6 <%@ taglib prefix="fm" uri="http://java.sun.com/jsp/jstl/fmt" %>
7 <%@ page import="java.util.*" %>
8 <!DOCTYPE html>
9 <html>
10 <head>
11   <meta charset="UTF-8" />
12   <meta name="viewport" content="width=device-width, initial-scale=1" />
13   <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" />
14   <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js" />
15   <title>게시판</title>
16 </head>
17 <body>
18   <div class="container">
19     <h2>멤버 목록</h2>
20     <br/>
21     <table class="table table-striped table-hover">
22       <tr>
23         <th>id</th>
24         <th>name</th>
25         <th>email</th>
26         <th>joinDate</th>
27       </tr>
28       <c:forEach var="dto" items="${dtos}">
29         <tr>
30           <td>${dto.id}</td>
31           <td>${dto.name}</td>
32           <td>${dto.email}</td>
33           <td><fmt:formatDate value="${dto.joinDate}" /></td>
34         </tr>
35       </c:forEach>
36     </table>
37     <input type="button" value="홈으로" onclick="location.href='index.html'" />
38     <input type="button" value="멤버 등록" onclick="location.href='insertForm.do'" />
39   </div>
40 </body>
41 </html>
```

2. 멤버 등록 하기



멤버 등록 하기 - 실행 화면



멤버 입력
멤버 목록

회원가입

아이디 :
byung

비밀번호 :
....

이름 :
이병헌

이메일 :
byung@gmail.com

가입하기 다시입력 홈으로

멤버 목록

id	name	email	joinDate
jin	이요신	jins@gmail.com	2019. 11. 5
lee	이순신	lee@gmail.com	2019. 11. 5
song	이주성	song@gmail.com	2019. 10. 30
byung	이병헌	byung@gmail.com	2019. 11. 6

홈으로 멤버 등록

멤버 등록 하기 - 구현 순서

중요!!!!

매 단계마다 프로그램을 실행시켜
디버깅을 하면서 구현하도록 한다.

1. 멤버 등록 페이지를 작성한다.
 - 파일명 : **mInsertForm.jsp**
2. 컨트롤러에 멤버등록을 위한 코드를 추가한다. - **Controller**
 - 파일명 : **MemberController.java**
3. 클라이언트가 보내온 매개변수들을 DTO의 형태로 만들고 DB로직을 호출하는 Command클래스를 작성한다. - **Model**
 - 파일명 : **MInsertCommand.java** (MCommand 인터페이스를 구현)
4. DTO에 담겨온 데이터를 DBMS에 저장처리를 하는 DAO클래스의 메소드(insert())를 작성한다. - **Model**
 - 파일명 : **MemberDAO.java**

mInsertForm.jsp

```
1 <%@ page language="java" contentType="text/html" %>
2   pageEncoding="UTF-8"%>
3 <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
4 <%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
5
6 <!DOCTYPE html>
7 <html>
8 <head>
9   <meta charset="UTF-8">
10  <meta name="viewport" content="width=device-width, initial-scale=1">
11  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css">
12  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js">
13  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js">
14  <title>회원가입창</title>
15 </head>
```

```
16 <body>
17 <div class="container">
18   <h2>회원가입</h2>
19   <form method="post" action="insert.do" class="form-horizontal">
20     <div class="form-group">
21       <div class="col-sm-4">
22         아이디 : <input type="text" name="id" class="form-control">
23       </div>
24     </div>
25     <div class="form-group">
26       <div class="col-sm-4">
27         비밀번호 : <input type="password" name="pwd" class="form-control">
28       </div>
29     </div>
30     <div class="form-group">
31       <div class="col-sm-4">
32         이름 : <input type="text" name="name" class="form-control">
33       </div>
34     </div>
35     <div class="form-group">
36       <div class="col-sm-4">
37         이메일 : <input type="text" name="email" class="form-control">
38       </div>
39     </div>
40     <input type="submit" value="가입하기">
41     <input type="reset" value="다시입력">
42     <input type="button" value="홈으로" onclick="location.href='index.html'">
43     <br><br>
44   </form>
45 </div>
46 </body>
47 </html>
```


MemberController 코드 추가

1. 멤버등록폼 호출 코드

```
36         }else if(com !=null && com.trim().equals("insertForm")) {  
37             viewPager = "mInsertForm.jsp";  
38         }
```

2. 멤버등록 처리 로직 호출 코드

- 멤버 등록처리 후에는 목록보기로 이동하도록 한다.

```
39         }else if(com !=null && com.trim().equals("insert")) {  
40             command = new MInsertCommand();  
41             command.execute(request, response);  
42             viewPager = "list.do";  
43         }
```

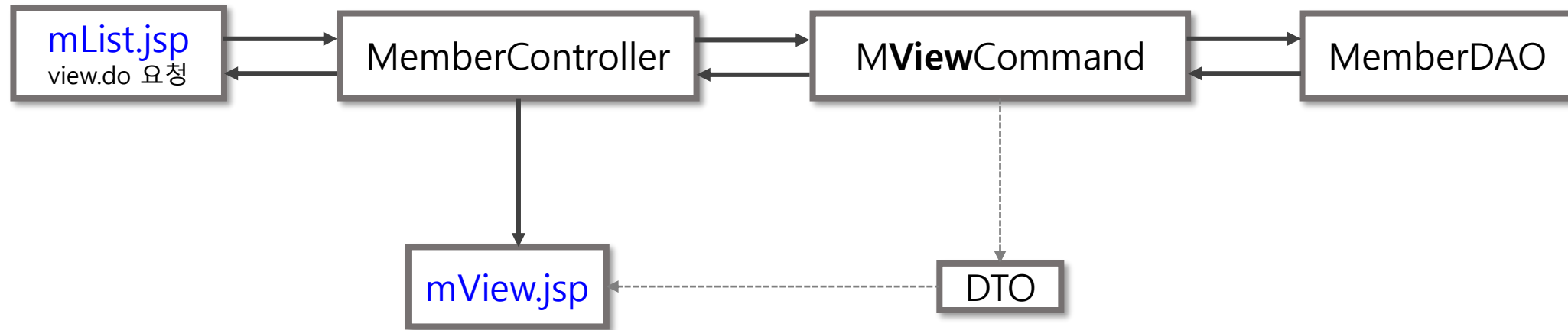
MInsertCommand

```
1 package cs.dit.command;
2
3 import java.io.IOException;
11
12 public class MInsertCommand implements MCommand{
13
14     @Override
15     public void execute(HttpServletRequest request, HttpServletResponse response) throws Service
16
17         MemberDTO dto = new MemberDTO();    //DB에 데이터를 저장하기 위해 DTO 객체 생성
18
19         dto.setId(request.getParameter("id"));    //DTO에 폼에서 전달된 데이터를 저장
20         dto.setPwd(request.getParameter("pwd"));
21         dto.setName(request.getParameter("name"));
22         dto.setEmail(request.getParameter("email"));
23
24         MemberDAO dao = new MemberDAO();
25         dao.insert(dto);    //DB에 DTO객체를 저장하기 위한 메소드 insert 호출
26     }
27 }
```

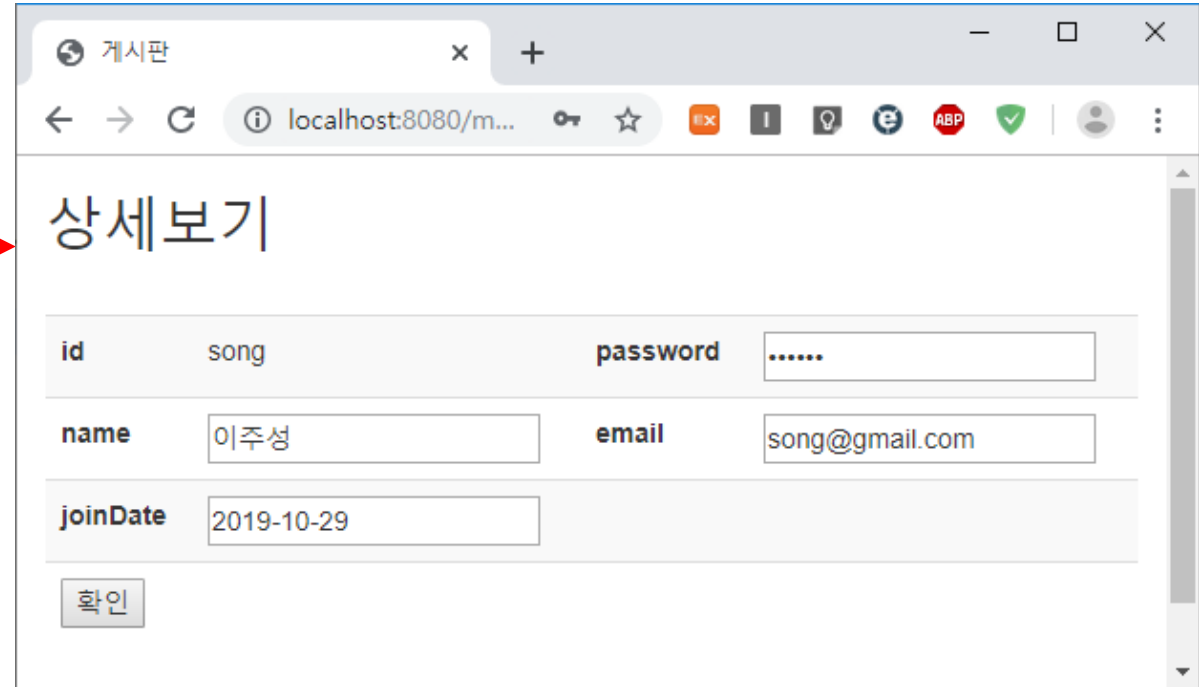
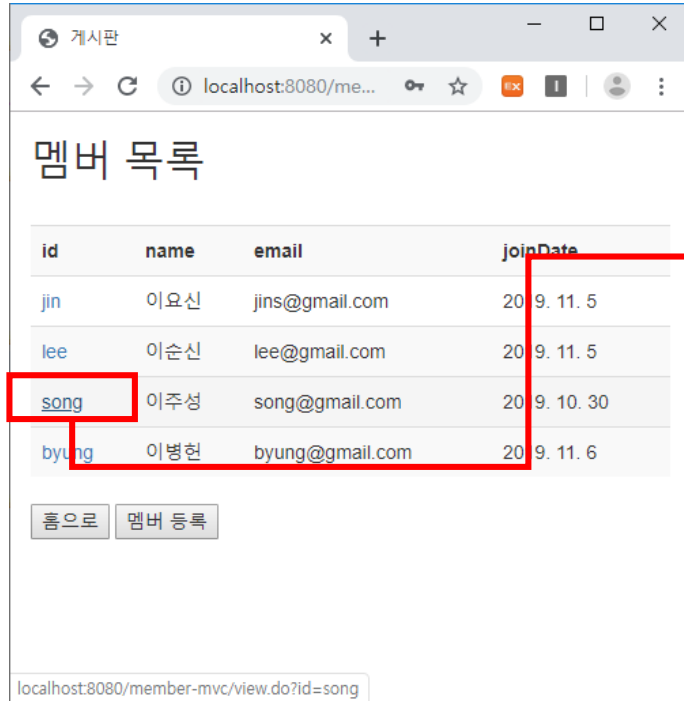
MemberDAO – insert()

```
101 // 멤버 추가하기
102 public void insert(MemberDTO dto) {
103     String sql = "insert into member(id, pwd, name, email, joinDate) values(?,?,?,?, SYSDATE)";
104
105     try {
106         con = ds.getConnection(); //Connection객체 CP에서 얻어오기
107         pstmt = con.prepareStatement(sql); //Connection객체를 통해 SQL문 준비
108         pstmt.setString(1, dto.getId()); //SQL문과 데이터 바인딩
109         pstmt.setString(2, dto.getPwd());
110         pstmt.setString(3, dto.getName());
111         pstmt.setString(4, dto.getEmail());
112
113         pstmt.executeUpdate(); //SQL을 수행하고 결과 반환 : 결과는 입력이 된 행 갯수
114
115         pstmt.close();
116     } catch (SQLException ex) {
117
118     }
119
120 }
```

3. 멤버 상세 보기



멤버 상세 보기 - 실행 화면



중요!!!!매 단계마다 프로그램을 실행시켜
디버깅을 하면서 구현하도록 한다.

멤버 상세 보기 - 구현 순서

1. id에 링크를 걸어 상세 보기로 이동할 수 있도록 변경한다. - **View**
 - 파일명 : **mList.jsp**
2. 컨트롤러에 멤버상세보기를 위한 코드를 추가한다. - **Controller**
 - 파일명 : **MemberController.java**
3. 클라이언트가 보내온 매개변수들을 DTO의 형태로 만들고 DB로직을 호출하는 Command클래스를 작성한다. - **Model**
 - 파일명 : **MViewCommand.java** (MCommand 인터페이스를 구현)
4. 전달된 id값으로 DB검색하는 메소드(view())를 작성한다. - **Model**
 - 파일명 : **MemberDAO.java**
5. 링크로 전달된 id값으로 멤버 상세 보기 페이지를 작성한다. - **View**
 - 파일명 : **mView.jsp**

mList.jsp 변경

```
16<body>
17<div class="container">
18  <h2>멤버 목록</h2>
19  <br/>
20  <table class="table table-striped table-hover">
21    <tr>
22      <th>id</th>
23      <th>name</th>
24      <th>email</th>
25      <th>joinDate</th>
26    </tr>
27    <c:forEach var="dto" items = "${dtos}">
28      <tr>
29        <td><a href="view.do?id=${dto.id}">${dto.id}</a></td>
30        <td>${dto.name}</td>
31        <td>${dto.email}</td>
32        <td><fmt:formatDate value="${dto.joinDate}"/></td>
33      </tr>
34    </c:forEach>
35  </table>
36  <input type="button" value = "홈으로" onclick = "location.href='index.html'">
37  <input type="button" value = "멤버 등록" onclick = "location.href='insertForm.do'">
38 </div>
39 </body>
40 </html>
```

MemberController 코드 추가

- 멤버 상세 보기에 해당하는 코드 추가

```
43         }else if(com !=null && com.trim().equals("view")) {  
44             command = new MViewCommand();  
45             command.execute(request, response);  
46             viewPage = "mView.jsp";  
47         }
```


MViewCommand

```
1 package cs.dit.command;
2
3 import java.io.IOException;
4
11
12 public class MViewCommand implements MCommand{
13
14     @Override
15     public void execute(HttpServletRequest request, HttpServletResponse response) throws ServletException
16         request.setCharacterEncoding("utf-8");
17
18         String id = request.getParameter("id");//링크가 걸려있는 id를 클릭하면 매개변수로 전달 받음
19
20         MemberDAO dao = new MemberDAO();
21
22         MemberDTO dto = dao.view(id); //상세보기를 위해 선택한 id로 DB에서 데이터 추출하여 DTO에 담는다.
23
24         request.setAttribute("dto", dto);//DTO를 view에서 데이터를 접근할 수 있도록 Request scope에 저장
25     }
26
27 }
```

MemberDAO – view()

```
73 //멤버 상세 보기
74 public MemberDTO view(String id) {
75     String sql = "select pwd, name, email, joinDate from member where id=?";
76     MemberDTO dto = new MemberDTO();
77     try {
78         con = ds.getConnection();
79         pstmt = con.prepareStatement(sql);
80
81         pstmt.setString(1, id);
82         rs = pstmt.executeQuery();
83
84         if(rs.next()) { //상세보기를 위한 한 레코드셋을 DTO에 저장
85             dto.setId(id);
86             dto.setPwd(rs.getString("pwd"));
87             dto.setName(rs.getString("name"));
88             dto.setEmail(rs.getString("email"));
89             dto.setJoinDate(rs.getDate("joinDate"));
90         }
91
92         rs.close();
93         pstmt.close();
94     } catch (SQLException e) {
95         e.printStackTrace();
96     } finally {
97         close();
98     }
99     return dto; //DTO객체에 데이터를 담아서 반환
100 }
```

mView.jsp

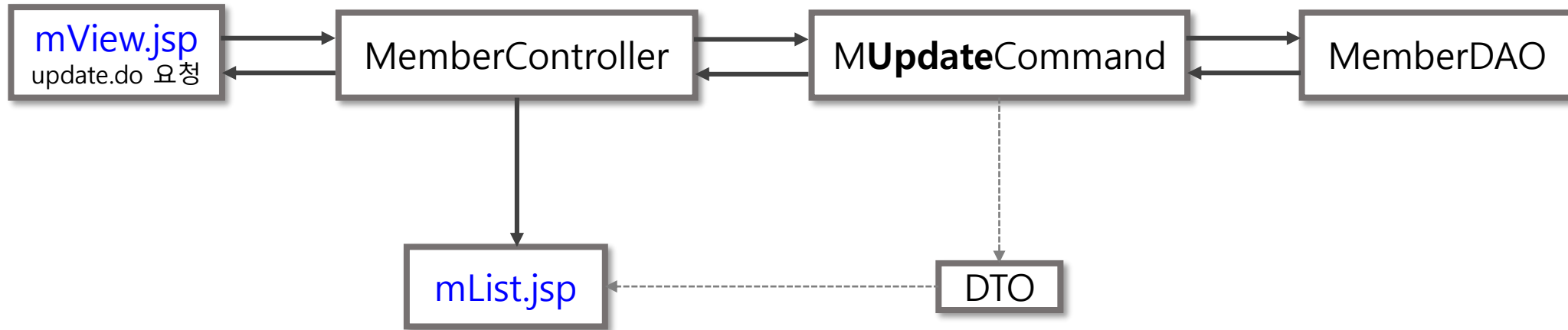
```
18<body>
19<div class="container">
20  <h2>상세보기</h2>
21  <br/>
22  <form action="" method="post">
23
24    <table class="table table-striped table-hover">
25      <tr>
26        <th>id</th><td>${dto.id}</td>
27        <th>password</th><td><input type="password" value="${dto.pwd}" name="pwd"></td>
28      </tr>
29      <tr>
30        <th>name</th><td><input type="text" value="${dto.name}" name="name"></td>
31        <th>email</th><td><input type="text" value="${dto.email}" name="email"></td>
32      </tr>
33      <tr>
34        <th>joinDate</th><td colspan="3"><input type="text" value="${dto.joinDate}" name="joinDate"></td>
35      </tr>
36      <tr>
37        <td colspan="4">
38          <input type="submit" value = "확인" >
39        </td>
40      </tr>
41    </table><br><br>
42  </form>
43 </div>
44 </body>
```

상세보기

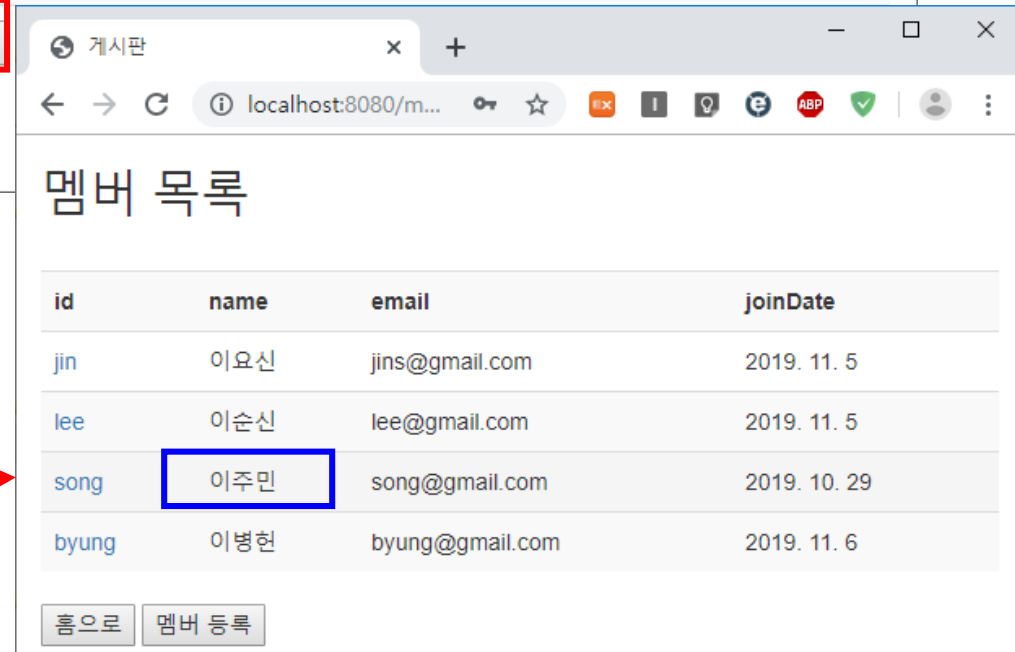
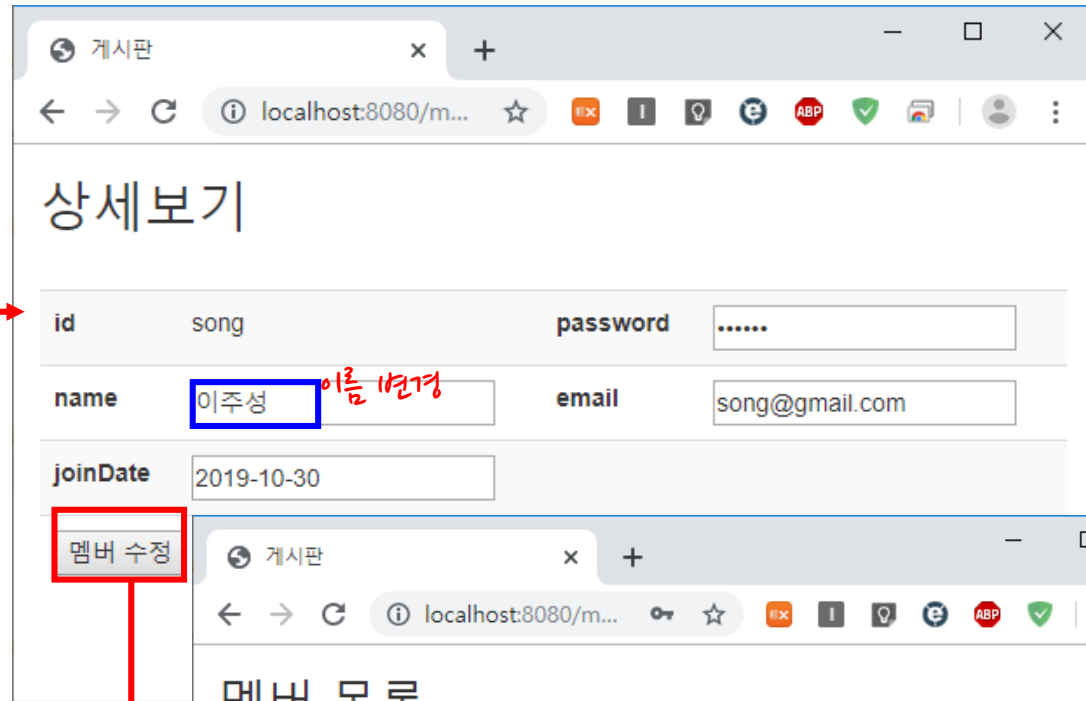
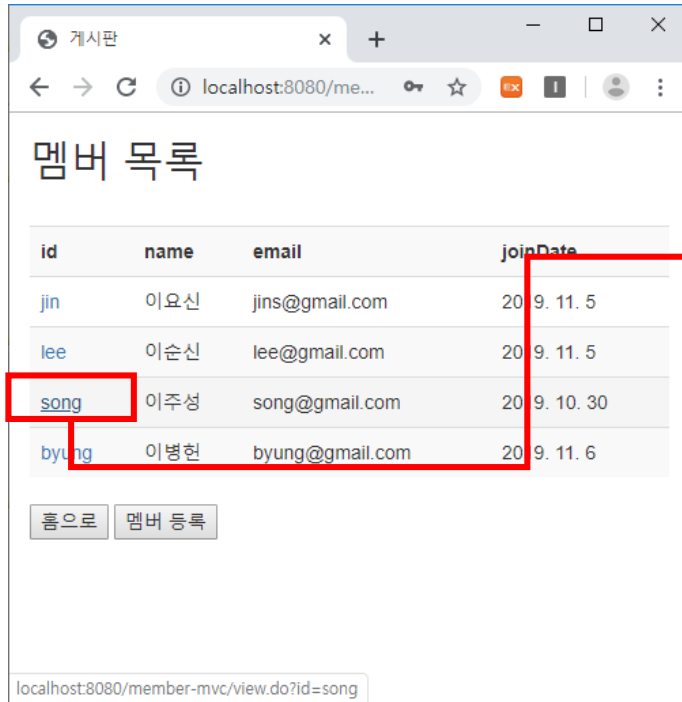
id	password	<input type="password"/>
name	<input type="text"/>	email <input type="text"/>
joinDate	<input type="text"/>	
<input type="button" value="확인"/>		

편집가능한 상태가 됨

4. 멤버 정보 수정 하기



멤버 정보 수정 하기 - 실행화면



멤버 정보 수정 하기 - 구현 순서

1. 변경된 정보를 넣고 확인을 누르면 수정 처리가 될 수 있도록 코드 변경하기. - **View**
 - 파일명 : **mView.jsp**
2. 컨트롤러에 멤버상세보기를 위한 코드를 추가한다. - **Controller**
 - 파일명 : **MemberController.java**
3. 클라이언트가 보내온 매개변수들을 DTO의 형태로 만들고 DB로직을 호출하는 Command클래스를 작성한다. - **Model**
 - 파일명 : **MViewCommand.java** (MCommand 인터페이스를 구현)
4. DTO에 담겨 온 수정 데이터를 DB에서 변경하도록 하는 메소드 (update())를 작성한다. - **Model**
 - 파일명 : **MemberDAO.java**

mView.jsp 확인

```
18 <body>
19 <div class="container">
20   <h2>상세보기</h2>
21   <br/>
22   <form action="update.do" method="post">
23     <input type="hidden" name="id" value="${dto.id}">
24     <table class="table table-striped table-hover">
25       <tr>
26         <th>id</th><td>${dto.id}</td>
27         <th>password</th><td><input type="password" value="${dto.pwd}" name="pwd"></td>
28       </tr>
29       <tr>
30         <th>name</th><td><input type="text" value="${dto.name}" name="name"></td>
31         <th>email</th><td><input type="text" value="${dto.email}" name="email"></td>
32       </tr>
33       <tr>
34         <th>joinDate</th><td colspan="3"><input type="text" value="${dto.joinDate}" name="joinDate"></td>
35       </tr>
36       <tr>
37         <td colspan="4">
38           <input type="submit" value = "멤버 수정" >
39         </td>
40       </tr>
41     </table><br><br>
42   </form>
43 </div>
44 </body>
```

멤버변수를 화면에 표시하지 않고 다음 페이지로 전달하는 방법



MemberController 코드 추가

- 멤버 상세 보기에 해당하는 코드 추가

```
48         else if(com !=null && com.trim().equals("update")){  
49             command = new MUpdateCommand();  
50             command.execute(request, response);  
51             viewPage = "list.do";  
52         }
```

변경된 게시글을 확인하기 위해서는 리스트를 보는 것이 가장 확실하므로
리스트를 호출하도록 한다.

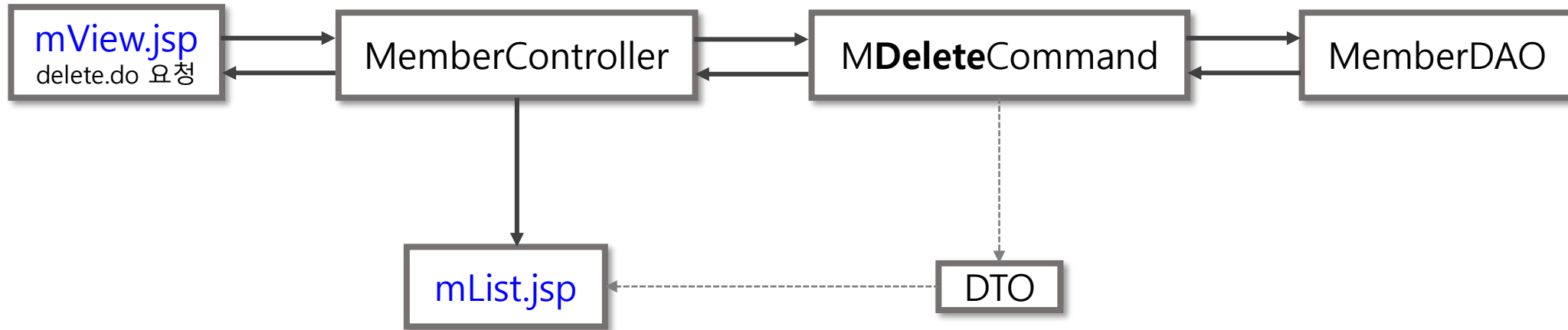
MUpdateCommand

```
1 package cs.dit.command;
2
3 import java.io.IOException;
12
13 public class MUpdateCommand implements MCommand {
14
15     @Override
16     public void execute(HttpServletRequest request, HttpServletResponse response) throws ServletException {
17         request.setCharacterEncoding("utf-8"); //한글 처리
18
19         MemberDTO dto = new MemberDTO();
20
21         dto.setId(request.getParameter("id"));
22         dto.setPwd(request.getParameter("pwd"));
23         dto.setName(request.getParameter("name"));
24         dto.setEmail(request.getParameter("email"));
25         dto.setJoinDate(Date.valueOf(request.getParameter("joinDate"))); //문자열로 받은 매개변수를 Date형으로 변환
26
27         MemberDAO dao = new MemberDAO();
28
29         dao.update(dto); //DB에 변경된 데이터 업데이트
30     }
31 }
```

MemberDAO – update()

```
128 // 멤버 정보 수정하기
129 public boolean update(MemberDTO dto) {
130     String sql = "update member set name=?, pwd=?, email=?, joinDate=? where id=?";
131     boolean check = false;
132     try {
133         con = ds.getConnection();
134         pstmt = con.prepareStatement(sql);
135         pstmt.setString(1, dto.getName());
136         pstmt.setString(2, dto.getPwd());
137         pstmt.setString(3, dto.getEmail());
138         pstmt.setDate(4, dto.getJoinDate());
139         pstmt.setString(5, dto.getId());
140
141         int x = pstmt.executeUpdate();
142
143         if(x<1) {
144             System.out.println("정상적으로 저장되지 않았습니다.");
145         }else {
146             check=true;
147         }
148         pstmt.close();
149     }catch(SQLException ex) {
150         System.out.println("SQL insert 오류 : " + ex.getMessage());
151         check = false;
152     }
153     return check;
154 }
```

4. 멤버 삭제 하기



멤버 삭제 하기 - 실행화면

게시판

localhost:8080/me...

멤버 목록

id	name	email	joinDate
jin	이요신	jins@gmail.com	2019. 11. 5
lee	이순신	lee@gmail.com	2019. 11. 5
song	이주민	song@gmail.com	2019. 10. 29
byung	이병헌	byung@gmail.com	2019. 11. 6

홈으로 멤버 등록

게시판

localhost:8080/m...

상세보기

id	song	password
name	이주민	email	song@gmail.com
joinDate	2019-10-29		

멤버 수정 멤버 삭제 멤버 목록 멤버 등록

게시판

localhost:8080/m...

멤버 목록

id	name	email	joinDate
jin	이요신	jins@gmail.com	2019. 11. 5
lee	이순신	lee@gmail.com	2019. 11. 5
byung	이병헌	byung@gmail.com	2019. 11. 6

홈으로 멤버 등록

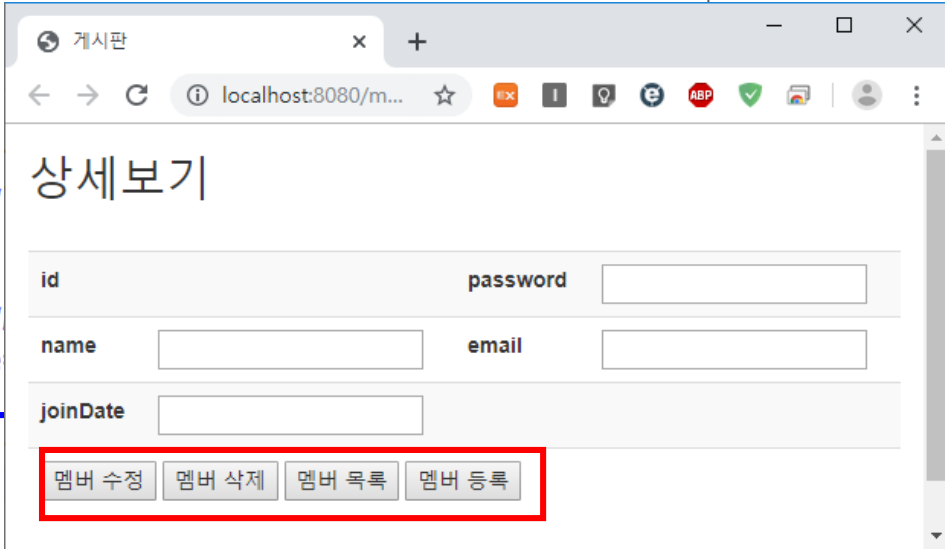
멤버 삭제 하기 - 구현 순서

1. 삭제버튼과 사용자 편리성을 이해 목록/등록 버튼을 추가하도록 코드를 변경한다. - **View**
 - 파일명 : **mView.jsp**
2. 컨트롤러에 멤버 삭제를 위한 코드를 추가한다. - **Controller**
 - 파일명 : **MemberController.java**
3. 클라이언트가 보내온 매개변수들을 DTO의 형태로 만들고 DB로직을 호출하는 Command클래스를 작성한다. - **Model**
 - 파일명 : **MDeleteCommand.java** (Mcommand 인터페이스를 구현)
4. DTO에 담겨 온 수정 데이터를 DB에서 삭제하도록 하는 메소드 (delete())를 작성한다. - **Model**
 - 파일명 : **MemberDAO.java**

mView.jsp 확인

```
18<body>
19<div class="container">
20  <h2>상세보기</h2>
21  <br/>
22  <form action="update.do" method="post">
23    <input type="hidden" name="id" value="${dto.id}">
24    <table class="table table-striped table-hover">
25      <tr>
26        <th>id</th><td>${dto.id}</td>
27        <th>password</th><td><input type="password" value="${dto.pwd}">
28      </tr>
29      <tr>
30        <th>name</th><td><input type="text" value="${dto.name}" name="
31        <th>email</th><td><input type="text" value="${dto.email}" name=
32      </tr>
33      <tr>
34        <td colspan="4">
35          <input type="submit" value = "멤버 수정" >
36          <input type="button" value = "멤버 삭제" onclick = "location.href='delete.do?id=${dto.id}'">
37          <input type="button" value = "멤버 목록" onclick = "location.href='list.do'">
38          <input type="button" value = "멤버 등록" onclick = "location.href='insertForm.do'">
39        </td>
40      </tr>
41    </table><br><br>
42  </form>
43</div>
44</body>
```

멤버번호를 화면에 표시하지 않고 다음 페이지로 전달하는 방법



MemberController 코드 추가

- 멤버 삭제에 해당하는 코드 추가

```
53         else if(com !=null && com.trim().equals("delete")){  
54             command = new MDeleteCommand();  
55             command.execute(request, response);  
56             viewPager = "list.do";  
57         }
```

게시글 삭제를 확인하기 위해 리스트 보기로 이동

MdeleteCommand

```
1 package cs.dit.command;
2
3 import java.io.IOException;
11
12 public class MDeleteCommand implements MCommand{
13
14     @Override
15     public void execute(HttpServletRequest request, HttpServletResponse response) throws ServletException {
16         request.setCharacterEncoding("utf-8");
17         String id = request.getParameter("id");
18
19         MemberDAO dao = new MemberDAO();
20
21         dao.delete(id);
22     }
23 }
```


MemberDAO – delete()

```
155 // 멤버 삭제 하기
156 public boolean delete(String id) {
157     String sql = "delete from member where id=?";
158     boolean check = false;
159     try {
160         con = ds.getConnection();
161         pstmt = con.prepareStatement(sql);
162         pstmt.setString(1, id);
163
164         int x = pstmt.executeUpdate();
165
166         if(x<1) {
167             System.out.println("정상적으로 삭제되지 않았습니다.");
168         }else {
169             check=true;
170         }
171         pstmt.close();
172     }catch(SQLException ex) {
173         System.out.println("SQL insert 오류 : " + ex.getLocalizedMessage());
174         check = false;
175     }
176     return check;
177 }
```

참고 자료

Collection Framework

- 배열의 단점을 보완한 데이터 군을 저장하는 클래스들을 표준화한 설계이다.
- 다수의 데이터를 쉽게 처리할 수 있는 방법을 제공하는 클래스들로 구성된다.

인터페이스	특징	구현 클래스
Set	데이터의 중복을 허용하지 않고 순서는 유지하지 않는 집합	HashSet, TreeSet 등
List	데이터의 중복을 허용하고 순서가 있는 데이터 구조 가짐	ArrayList, LinkedList, Vector 등
Map	Key와 value의 쌍으로 이루어진 데이터 집합 Key는 중복을 허용하지 않고 값은 중복 허용 순서는 유지되지 않음	HashMap, TreeMap, Hashtable 등

Array

- ✓ 관련된 데이터를 하나의 변수에 묶어 일괄적으로 관리하기 위한 데이터 구조
- ✓ 단점 :
 - 인덱스에 따라 값을 유지하므로 요소가 삭제되어도 빈자리가 남게 됨
 - 배열 크기 변경 불가능
 - 기능이 없음

- 컬렉션(collection) : 다수의 데이터, 데이터 그룹을 의미
- 프레임워크(framework) : 표준화, 정형화된 체계적인 프로그래밍 방식
- 컬렉션 클래스(collection class) : 다수의 데이터를 저장할 수 있는 클래스

Collection Framework

- 오라클 웹사이트의 문서 참조

<https://docs.oracle.com/javase/8/docs/api/java/util/package-summary.html>

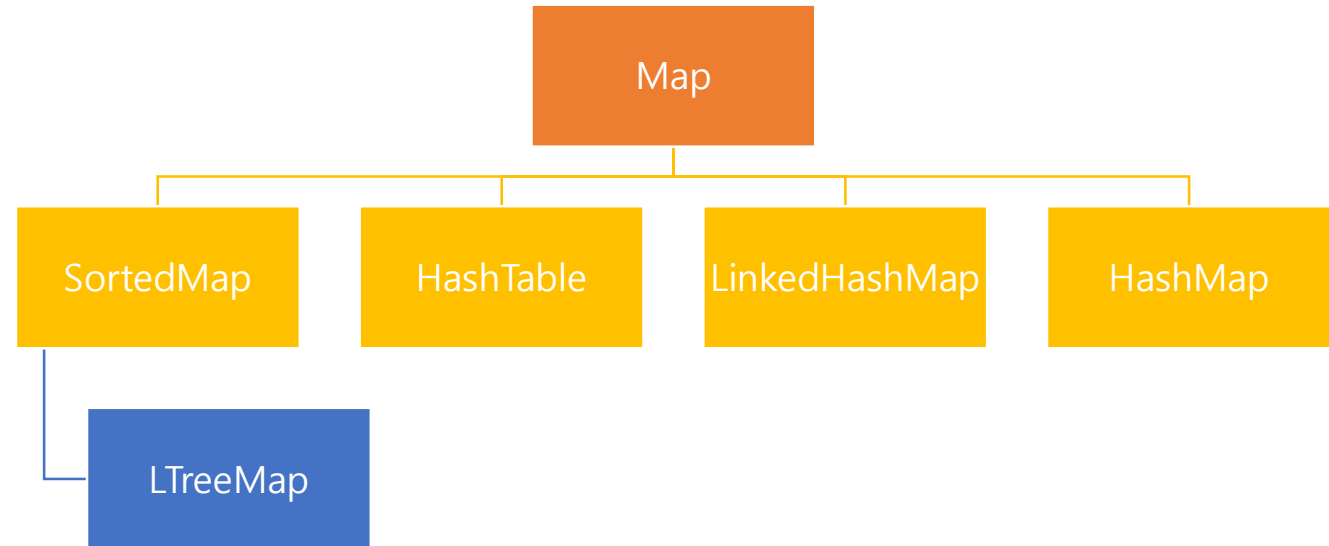
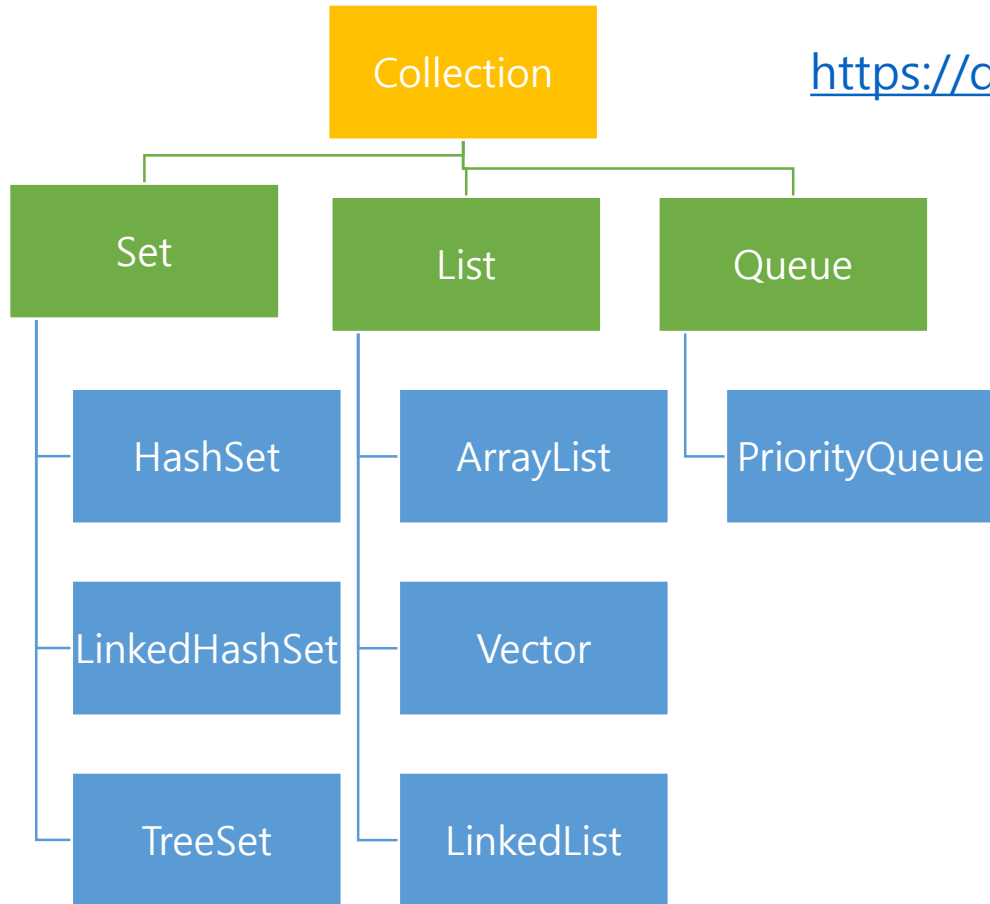


- 컬렉션은 Generic으로 정의됨

Collection Framework

- 다양한 상황에서 사용할 수 있는 Container 객체(데이터를 저장하는 자료구조)
- `java.util.*` 패키지에 존재

<https://docs.oracle.com/javase/8/docs/api/java/util/package-summary.html>



ArrayList 객체

- Array와 List의 특징으로 만들어짐

- 객체생성

```
ArrayList<E> alist = new ArrayList<E>();
```

- 주요 메소드

반환형	메소드	기능
boolean	add (E e)	지정된 요소를 리스트의 끝에 추가
void	add (int index, E e)	지정된 요소를 리스트의 지정된 위치에 삽입
void	clear ()	리스트로부터 모든 요소 삭제
boolean	contains (Object o)	리스트에 지정된 요소가 있는 경우는 true 반환
E	get (int index)	리스트 내의 지정된 위치의 요소 반환
boolean	isEmpty ()	리스트에 요소가 없는 경우는 true 반환
E	remove (int index)	리스트 내의 지정된 위치에 있는 요소 삭제
boolean	remove (Object o)	지정된 요소의 최초의 출현을 리스트로부터 삭제
int	size ()	리스트 내의 요소 수 반환

ArrayList 객체

```
12 <%  
13 //ArrayList 생성하기  
14 ArrayList<String> alist = new ArrayList<String>();  
15  
16 //ArrayList에 데이터 추가하기  
17 alist.add("즐거운");  
18 alist.add("금요일");  
19 alist.add("밤입니다.");  
20  
21 //ArrayList의 데이터 조회하기  
22 for(int i=0; i<alist.size(); i++){  
23     out.println(alist.get(i) + "<br>");  
24 }  
25 %>
```

제네릭(Generic)

- 클래스 내부에 사용할 데이터 타입을 **인스턴스를 생성 시 확정**
- 다양한 타입의 객체들을 다루는 메서드나 컬렉션 클래스에서 컴파일 할 때 타입 체크
- 제네릭 사용 이유
 - 데이터 타입의 안정성(Safety)를 위해 사용
 - 컴파일러가 실행 전 타입 에러 검출
 - 의도하지 않은 타입의 객체를 저장하는 것을 막고, 저장된 객체를 꺼내 올 때 원래의 타입과 다른 타입으로 형 변환되어 발생할 수 있는 오류를 줄여 줌
 - 타입 체크와 형 변환을 생략으로 중복 코드 제거(코드가 간결해 짐).
- Collection Framework을 사용하려면 Generic을 기본적으로 알아야 함

- 기존에는 다양한 종류의 타입을 다루는 메서드의 매개변수나 반환 타입으로 Object의 참조 변수를 많이 사용했고, 그로 인해 형 변환이 불가피했지만, 이젠 Generic으로 Object타입 대신 원하는 타입 (**객체타입만 가능**)을 지정
- 타입을 지정하지 않으면 Object 타입으로 간주

제네릭(Generic)

```
11 <%!  
12 public static class Test<E>{  
13     private E foo ;  
14     public void setFoo(E foo){  
15         this.foo = foo;  
16     }  
17     public E getFoo() { return foo; }  
18 }  
19 %>  
20 <h1>Hello World!</h1>  
21 <%  
22 Test<String> ts1 = new Test<String>();  
23 ts1.setFoo("즐거운 토요일");  
24 out.println(ts1.getFoo());  
25 out.println("<br>");  
26  
27 Test<Integer> ts2 = new Test<Integer>();  
28 ts2.setFoo(new Integer(1234));  
29 out.println(ts2.getFoo());  
30 %>
```

Hello World!

즐거운 토요일
1234