

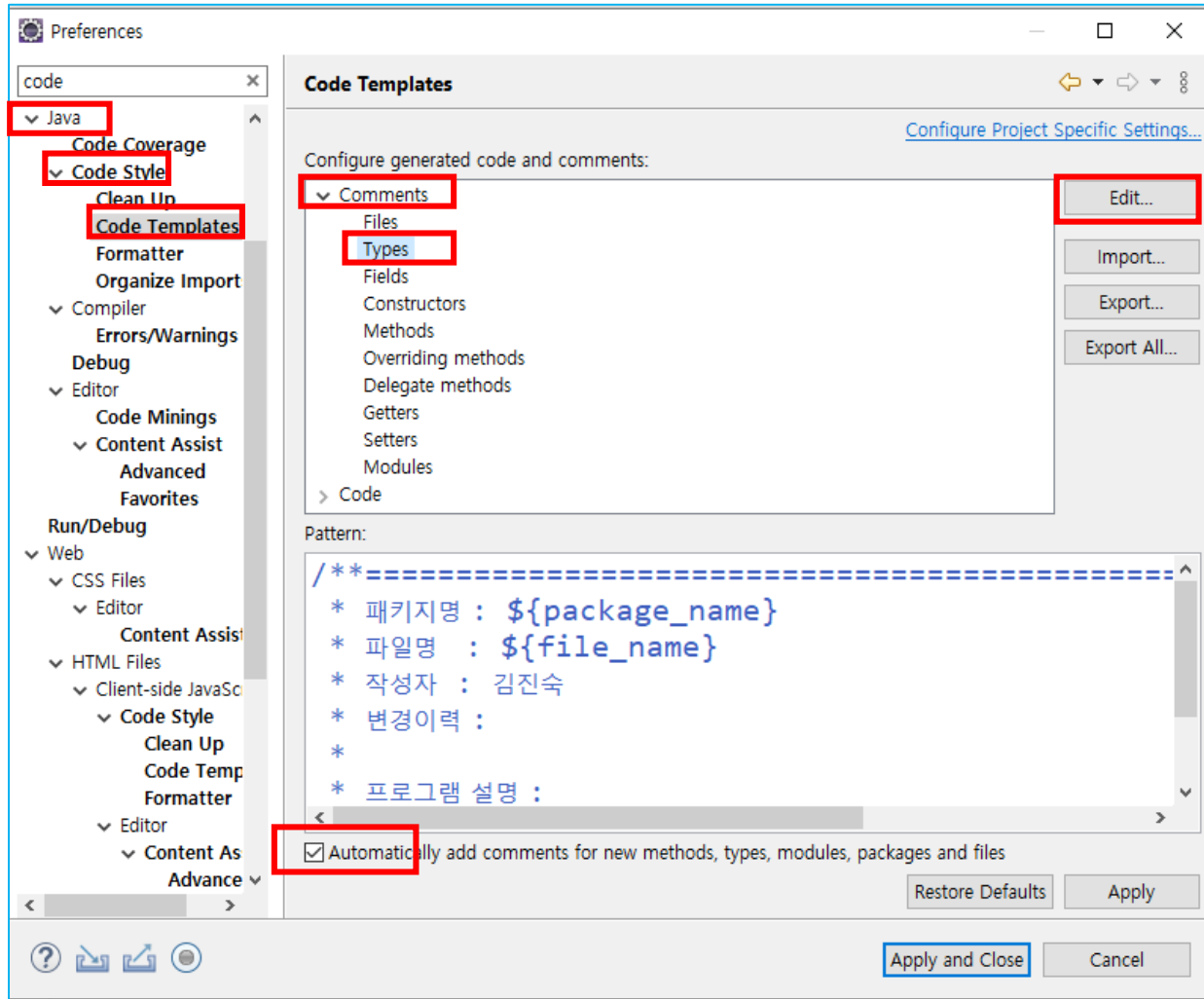
10장 DAO와 DTO (DBCP, Java Bean)

동의과학대학교
컴퓨터정보과
김진숙

내용

1. 테이블 작성
2. 설계도(Model 설계, View 설계)
3. 메인 화면
4. LoginDTO 객체 구현
5. LoginDAO 객체/JSP 다시 작성
 - insert 기능 구현
 - select 기능 구현
 - update 기능 구현
 - delete 기능 구현
6. 코드 개선하기 – 중복 코드 제거
7. 참고사항
 - Javabeen 만들기
 - Collection Framework
 - ArrayList객체
 - Generic

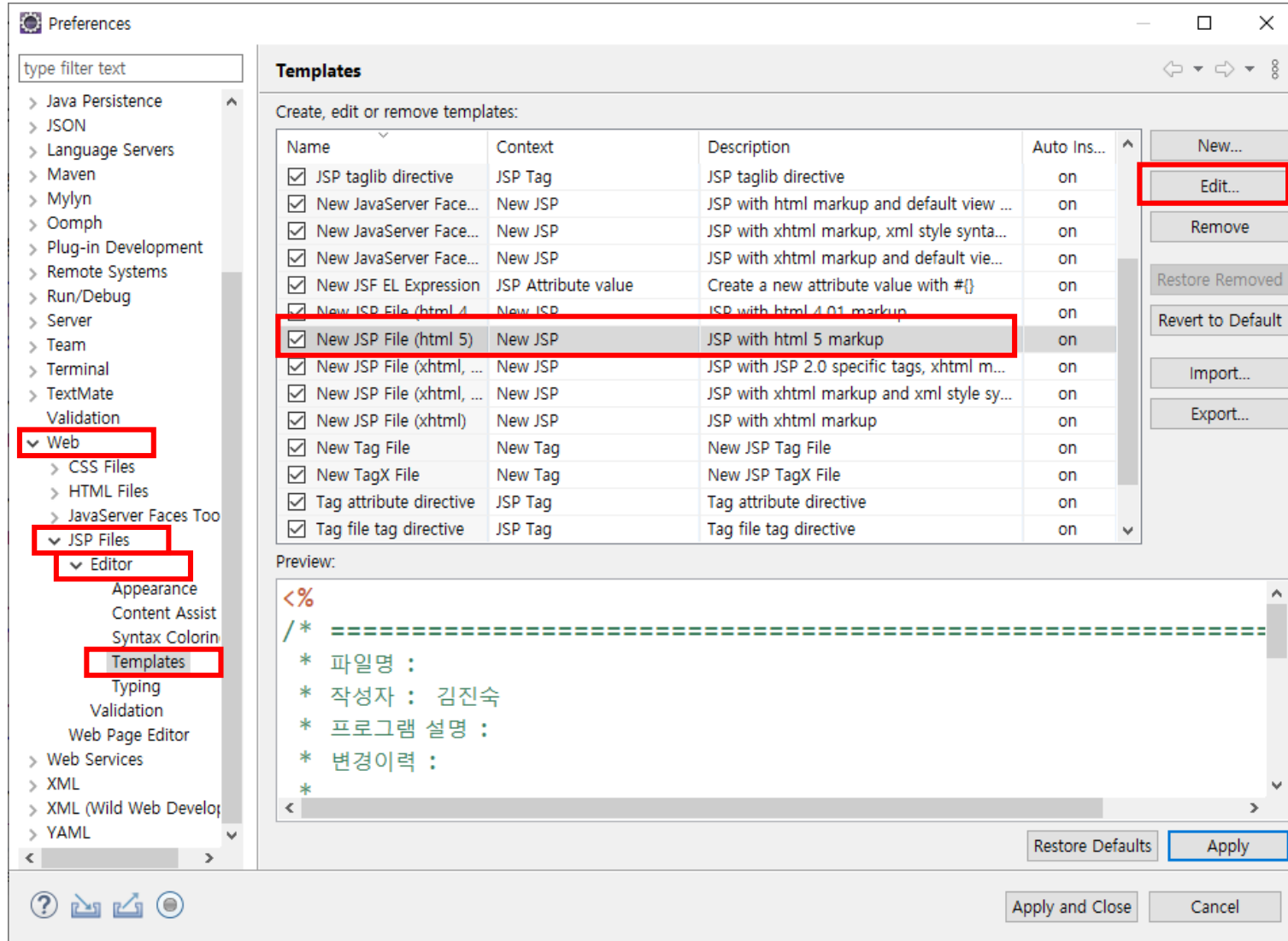
이클립스 자동주석 설정 - java



```
1 package csdit;  
2  
3 /** =====  
4  * 패키지명 : csdit  
5  * 파일명   : TestClass.java  
6  * 작성자   : 김진숙  
7  * 변경이력 :  
8  *  
9  * 프로그램 설명 :  
10 *  
11 * =====  
12  
13 public class TestClass {  
14  
15 }
```

이클립스의 주석 생성 기능 (Alt + Shift + J) 를
이용하여 작성

이클립스 자동주석 설정 - jsp



```
1= k%  
2 / * =====  
3 * 파일명 : file_name  
4 * 작성자 : 김진숙  
5 * 프로그램 설명 :  
6 * 변경이력 :  
7 *  
8 * =====*/  
9 %>  
10 <%@ page language="java" contentType="text/html; charset=UTF-8"  
11     pageEncoding="UTF-8"%>  
12 <!DOCTYPE html>  
13 <html>  
14 <head>  
15     <meta charset="UTF-8">  
16     <title>Insert title here</title>  
17 </head>  
18 <body>  
19  
20 </body>  
21 </html>
```

테이블 작성

- JDBC에서 사용한 테이블 사용(LOGIN)

```
CREATE TABLE login(  
  id VARCHAR(10),  
  name VARCHAR(20),  
  pwd VARCHAR(20)  
);
```

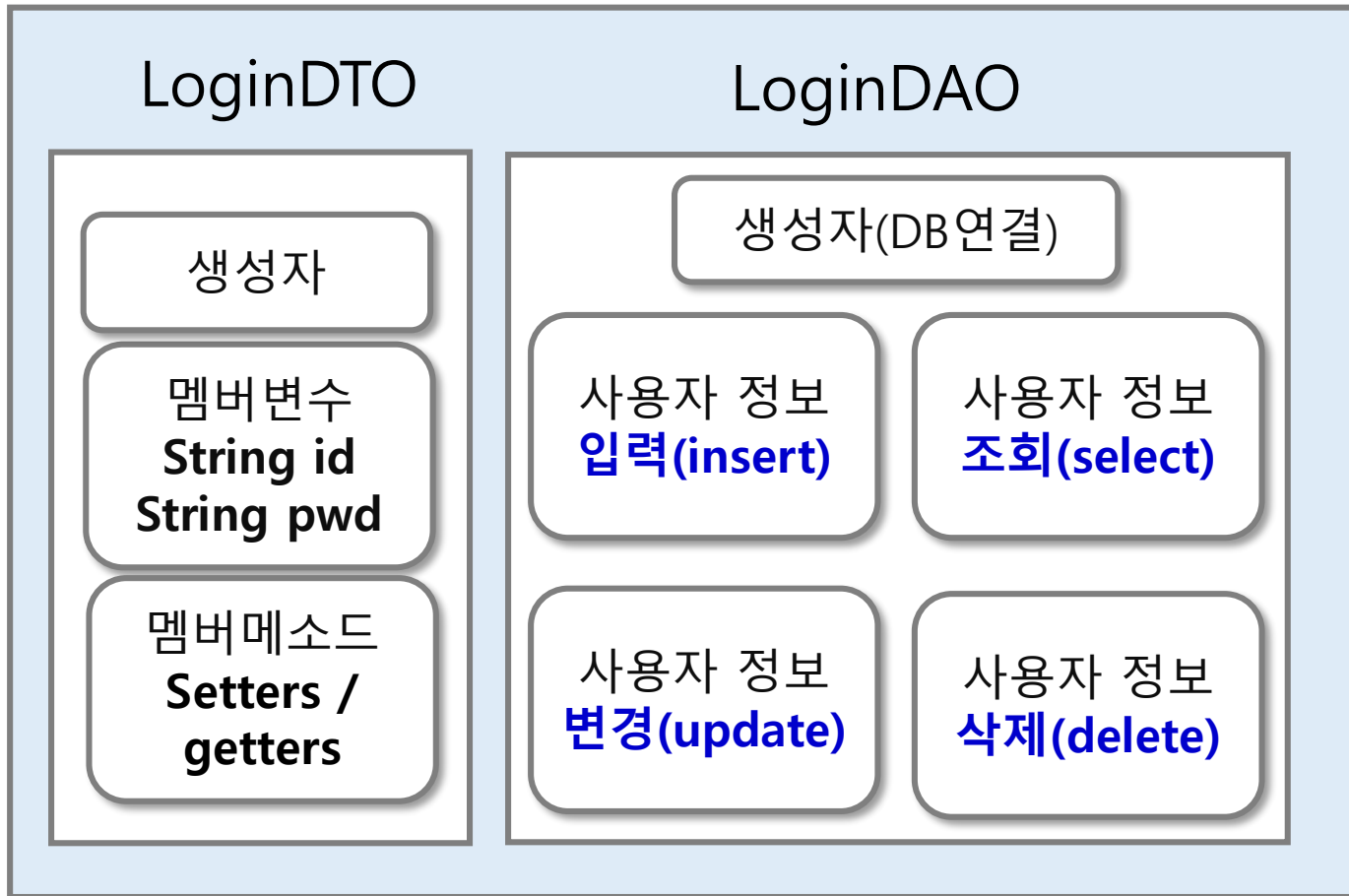
```
INSERT INTO login(id, name, pwd) VALUES ('gildong', '홍길동', '1111');  
INSERT INTO login(id, NAME, pwd) VALUES ('chunhyang', '성춘향', '2222');  
COMMIT;
```

DBCP 설정

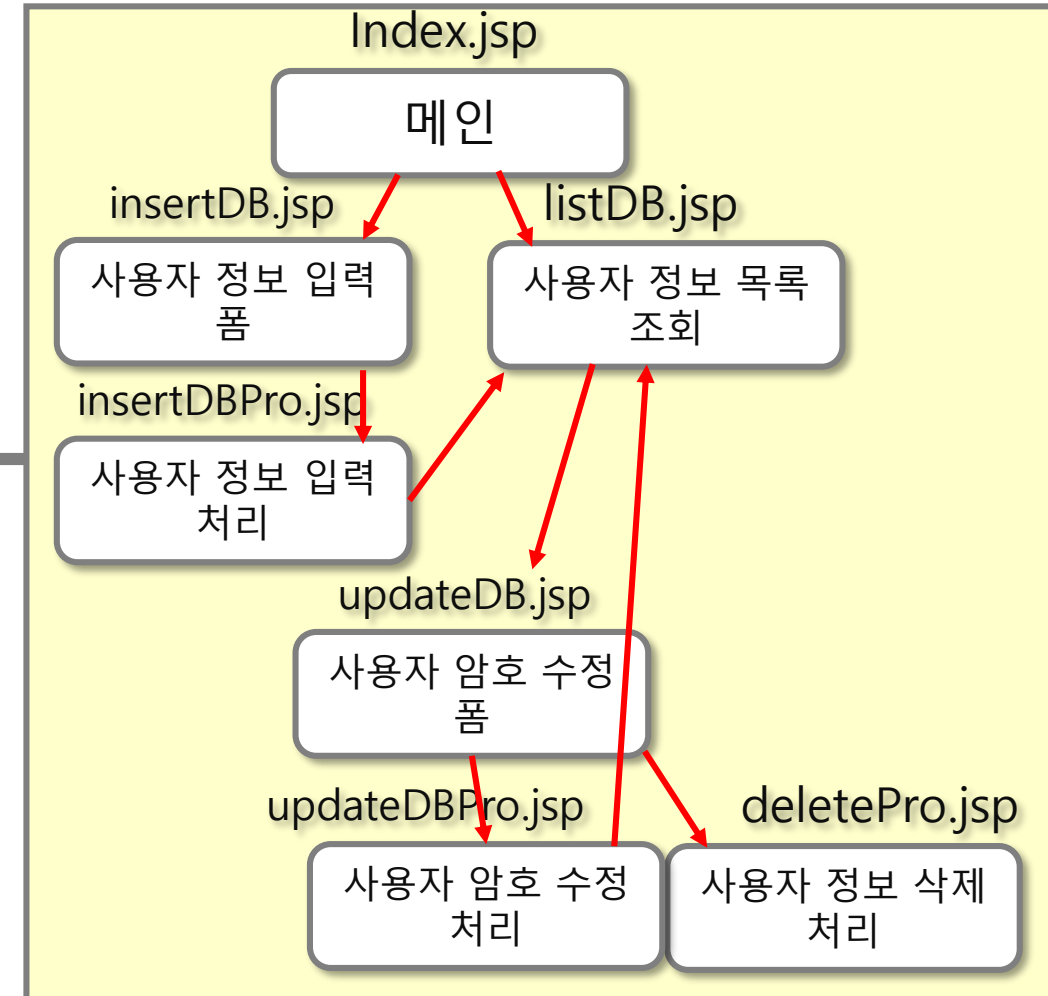
- /META-INF/context.xml

```
<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"  
>
```

Model 설계(Class)



View 설계(JSP)



메인

- index.jsp

사용자 관리

[사용자목록보기](#)[사용자 등록](#)

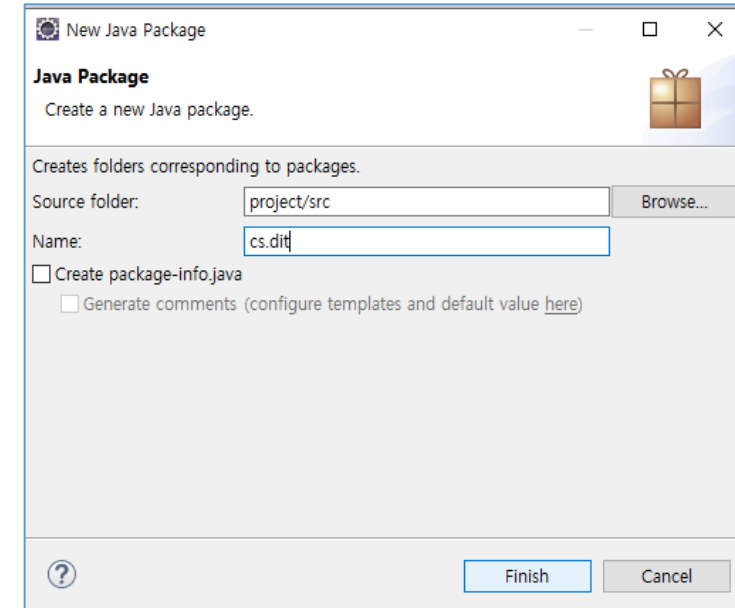
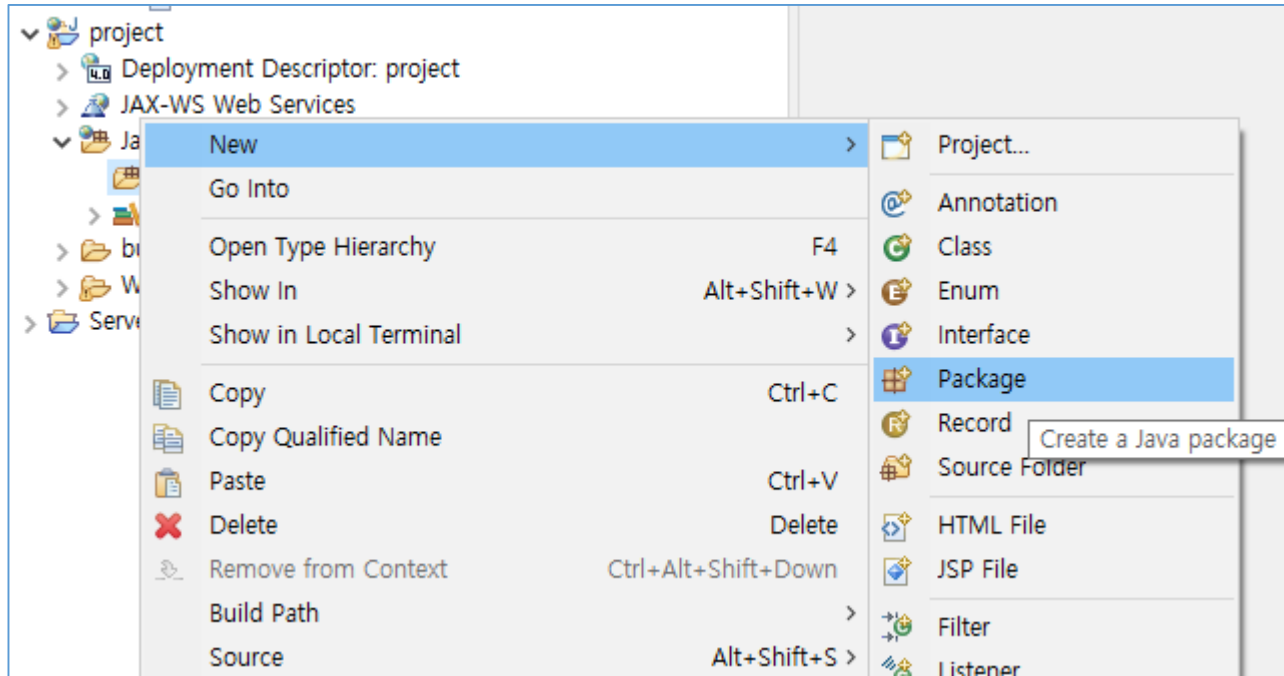
```
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2     pageEncoding="UTF-8"%>
3 <!DOCTYPE html>
4 <html>
5 <head>
6     <meta charset="UTF-8">
7     <title>사용자 관리</title>
8     <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css">
9     <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>
10    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.16.0/umd/popper.min.js"></script>
11    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js"></script>
12 </head>
13 <body>
14     <div class="container">
15         <h2>사용자 관리</h2>
16         <hr>
17         <input class="btn btn-primary" type="button" onclick="location.href='listDB.jsp'" value="사용자목록보기">
18         <input class="btn btn-primary" type="button" onclick="location.href='insertDB.jsp'" value="사용자 등록">
19     </div>
20 </body>
21 </html>
```


[실습] 이전 login 코드들을 DAO와 DTO로 변환

1. 패키지 생성
2. DTO(Data Transfer Object) 생성
 - login 테이블의 항목들로 DTO 작성
3. DAO(Data Access Object)생성
 - DBDP로 연동하여 커넥션 얻어오기 DAO 메소드로 처리
 - insertPro.jsp 코드를 DAO의 메소드로 처리
 - list.jsp 파일의 DB 관련 코드를 DAO에서 처리
 - updatePro.jsp 코드를 DAO의 메소드로 처리
 - delete.jsp 코드를 DAO 메소드로 처리

1. 패키지 생성

- 패키지 생성



2. DTO 생성

The screenshot illustrates the process of creating a DTO (Data Transfer Object) in an IDE. It shows the project structure, the context menu for creating a new class, the 'New Java Class' dialog, the resulting code for `LoginDto`, and the 'Generate Getters and Setters' dialog.

Project Structure: The project structure shows a package `cs.dit` under `src`. The `LoginDAO.java` file is selected, and the context menu is open, showing the option to create a new `Class`.

New Java Class Dialog: The dialog shows the source folder as `project/src`, the package as `cs.dit`, and the class name as `LoginDto`. The class is set to be `public` and inherit from `java.lang.Object`.

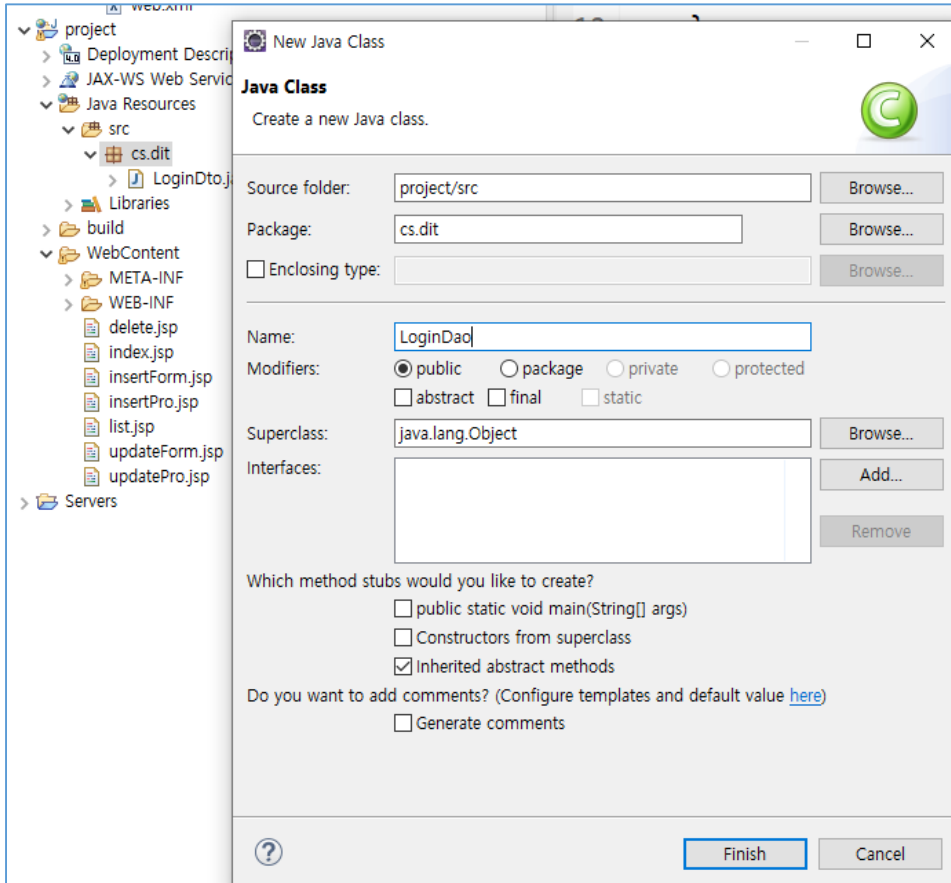
LoginDto.java Code: The code for `LoginDto` is shown, including the package declaration and the class definition with private fields `id`, `name`, and `pwd`.1 package cs.dit;
2
3 public class LoginDto {
4 private String id;
5 private String name;
6 private String pwd;
7
8
9 }
10

Generate Getters and Setters Dialog: The dialog shows the selected fields (`id`, `name`, `pwd`) and the option to generate getters and setters. The insertion point is set to 'After 'pwd'', and the sort by is 'Fields in getter/setter pairs'.

```
1 package cs.dit;
2
3 public class LoginDto {
4     private String id;
5     private String name;
6     private String pwd;
7     public String getId() {
8         return id;
9     }
10    public void setId(String id) {
11        this.id = id;
12    }
13    public String getName() {
14        return name;
15    }
16    public void setName(String name) {
17        this.name = name;
18    }
19    public String getPwd() {
20        return pwd;
21    }
22    public void setPwd(String pwd) {
23        this.pwd = pwd;
24    }
25
26    public LoginDto(String id, String name, String pwd) {
27        this.id = id;
28        this.name = name;
29        this.pwd = pwd;
30    }
31 }
```

3. DAO 작성

• DB 연동 코드 작성



API 정보 웹사이트

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

```
1  /**
4  package cs.dit;
5
6  import java.sql.Connection;
7  import java.sql.PreparedStatement;
8
9  import javax.naming.Context;
10 import javax.naming.InitialContext;
11 import javax.sql.DataSource;
12
13 /**=====
14  * 패키지명 : cs.dit
15  * 파일명 : LoginDao.java
16  * 작성자 : 김진숙
17  * 변경이력 :
18  * 2022-4-28/ 최초작성/ 김진숙
19  * 프로그램 설명 : Login 테이블의 내용과 연동하여 회원관리
20  *=====*/
21 public class LoginDao {
22
23     private Connection getConnection() throws Exception{
24         //1. JNDI를 이용하기 위한 객체 생성
25         Context initCtx = new InitialContext();
26
27         //2. 등록된 네이밍 서비스로부터 등록된 자원을 가져옴
28         Context envCtx = (Context) initCtx.lookup("java:comp/env");
29
30         //3. 자원들 중 원하는 jdbc/jskim 자원을 찾아내어 데이터소스를 가져옴
31         DataSource ds = (DataSource) envCtx.lookup("jdbc/jskim");
32
33         //4. 커넥션 얻어옴
34         Connection con = ds.getConnection();
35
36         return con;
37     }
38 }
```

[참조] Try-with-resources

- try에 자원 객체를 전달하면, try 코드 블록이 끝나면 자동으로 자원을 종료해주는 기능
- 따로 finally 블록이나 모든 catch 블록에 종료 처리를 하지 않아도 됨
- 이 때, try에 전달할 수 있는 자원은 AutoCloseable 인터페이스의 구현체로 한정됨(매뉴얼에서 해당 객체 확인 필요)
- AutoCloseable은 JDK1.7부터 추가된 인터페이스

```
try(Something1 s1 = new Something1();
    Something2 s2 = new Something2()) {

} catch(...) {
    ...
}
```

API 정보 웹사이트

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

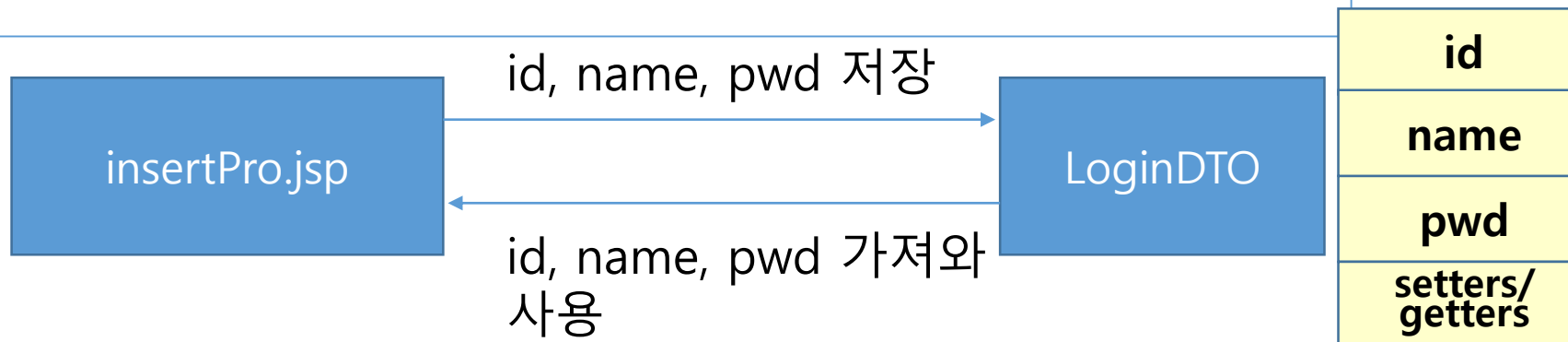
3. DAO 작성 – insert

- insert() 메소드 작성

파일들을 기능별로 쪼갤 때는 항상 입력, 출력 변수를 확인해야 함

```
39 public void insert(LoginDto dto) {  
40     String sql = "INSERT INTO login(ID, NAME, PWD) VALUES(?, ?, ?)";  
41  
42     try (  
43         Connection con = getConnection();  
44         PreparedStatement pstmt = con.prepareStatement(sql);  
45     )  
46     {  
47         pstmt.setString(1, dto.getId());  
48         pstmt.setString(2, dto.getName());  
49         pstmt.setString(3, dto.getPwd());  
50  
51         pstmt.executeUpdate();  
52     } catch (Exception e) {  
53         e.printStackTrace();  
54     }  
55 }
```

LoginDto 객체에서 데이터 가져오기



3. DAO 작성 – insert

- insertPro.jsp 다시 작성

```
1 <%@page import="cs.dit.LoginDao"%>
2 <%@page import="cs.dit.LoginDto"%>
3 <%@ page language="java" contentType="text/html; charset=UTF-8"
4     pageEncoding="UTF-8"
5 %>
6
7 <%request.setCharacterEncoding("utf-8");
8
9 String id = request.getParameter("id");
10 String name = request.getParameter("name");
11 String pwd = request.getParameter("pwd");
12
13 LoginDto dto = new LoginDto(id, name, pwd);
14 LoginDao dao = new LoginDao();
15
16 dao.insert(dto);
17
18 response.sendRedirect("list.jsp");
19 %>
```

LoginDto

id
name
pwd
Setters/ getters

LoginDto 객체에 데이터 저장

3. DAO 작성 - select

• list() 메소드 작성

```
public ArrayList<LoginDto> list(){
    String sql = "SELECT * FROM login";
    ArrayList<LoginDto> dtos = new ArrayList<LoginDto>();

    try (    Connection con = getConnection();
           Statement stmt = con.createStatement();
           ResultSet rs = stmt.executeQuery(sql);
        )
    {
        while(rs.next()) {
            LoginDto dto = new LoginDto();

            dto.setId(rs.getString("id"));
            dto.setName(rs.getString("name"));
            dto.setPwd(rs.getString("pwd"));

            dtos.add(dto);
        }

    } catch (Exception e) {
        e.printStackTrace();
    }

    return dtos;
}
```

dtos (ArrayList객체의 참조변수)

LoginDTO

ResultSet

GILSUN	GILDONG	KYUN
홍길순	홍길동	허균
1111	2222	3333
Setters/ getters	Setters/ getters	Setters/ getters

KYUN
허균
3333
Setters/ getters

id	name	pwd
GILSUN	홍길순	1111
GILDONG	홍길동	2222
KYUN	허균	3333

3. DAO 작성 - select

• list.jsp 다시 작성

```

7<%
8  //ArrayList<LoginDto> dtos = new LoginDao().list();
9  LoginDao dao = new LoginDao();
10 ArrayList<LoginDto> dtos = dao.list();
11 %>

```

```

32<% for(LoginDto dto : dtos) {
33  %>
34  <tr>
35    <td><a href="updateForm.jsp?id=<%=dto.getId() %>"><%=dto.getId() %></a></td>
36    <td><%=dto.getName() %></td>
37    <td><%=dto.getPwd() %></td>
38  </tr>
39  <%} %>

```

아이디	이름	비밀번호
yul	서이을	7654
chunhyang	성주희	4567
gildong	성주희	1111
minwoo	김민우	1111
sohee	성주희	99999

[홈으로](#)

id	name	pwd
GILSUN	홍길순	1111
GILDONG	홍길동	2222
KYUN	허균	3333

KYUN
허균
3333
Setters/ getters

dtos (ArrayList객체의 참조변수)

GILSUN	GILDONG	KYUN
홍길순	홍길동	허균
1111	2222	3333
Setters/ getters	Setters/ getters	Setters/ getters

0

1

2

3. DAO 작성 - update

- selectOne() 메소드 작성
 - Id 값으로 수정할 데이터를 검색

```
85 public LoginDto selectOne(String id) {  
86     String sql = "SELECT * FROM login WHERE id =?";  
87     LoginDto dto = new LoginDto();  
88  
89     try ( Connection con = getConnection();  
90           PreparedStatement pstmt = con.prepareStatement(sql);  
91         )  
92     { pstmt.setString(1, id);  
93  
94         try(ResultSet rs = pstmt.executeQuery();)  
95         {  
96             rs.next();  
97  
98             dto.setId(id);  
99             dto.setName(rs.getString("name"));  
100            dto.setPwd(rs.getString("pwd"));  
101  
102            }catch (Exception e) {  
103                e.printStackTrace();  
104            }  
105  
106        } catch (Exception e) {  
107            e.printStackTrace();  
108        }  
109        return dto;  
110    }  
111 }
```

3. DAO 작성 - update

• updateForm.jsp 다시 작성

```
1 <%@page import="cs.dit.LoginDao"%>
2 <%@page import="cs.dit.LoginDto"%>
3 <%@ page language="java" contentType="text/html; charset=UTF-8"
4     pageEncoding="UTF-8"
5     import="java.sql.*" %>
6 <% String id = request.getParameter("id");
7
8     LoginDto dto = new LoginDto();
9     LoginDao dao = new LoginDao();
10    out.println(id);
11
12    dto = dao.selectOne(id);
13 %>
```

```
24 <body>
25
26 <div class="container">
27     <br>
28     <h2 class="text-center font-weight-bold">사용자 정보 변경</h2>
29     <hr/>
30 <form action="updatePro.jsp" method="post">
31 <div class="form-group">
32     <label for="id">ID:</label>
33     <input type="text" class="form-control" id="id" name="id" value="<%=dto.getId()%>" readonly>
34 </div>
35 <div class="form-group">
36     <label for="name">NAME:</label>
37     <input type="text" class="form-control" id="name" name="name" value="<%=dto.getName()%>">
38 </div>
39 <div class="form-group">
40     <label for="pwd">PASSWORD:</label>
41     <input type="password" class="form-control" id="pwd" name="pwd" value="<%=dto.getPwd()%>">
42 </div>
43 <br>
44 <div class="text-center">
45     <input type="submit" value="변경" class="btn btn-secondary">
46     <input type="button" value="삭제" class="btn btn-secondary" onclick="location.href='delete.jsp?id=<%=id%>'">
47     <input type="button" value="목록" class="btn btn-secondary" onclick="location.href='list.jsp'">
48 </div>
49 </form>
50 </div>
51 </body>
```

3. DAO 작성 - update

- update() 메소드 작성

```
112 public void update(LoginDto dto) {  
113     String sql = "UPDATE login SET name = ?, pwd = ? WHERE id =?";  
114  
115     try (  
116         Connection con = getConnection();  
117         PreparedStatement pstmt = con.prepareStatement(sql);  
118     )  
119     {  
120         pstmt.setString(1, dto.getName());  
121         pstmt.setString(2, dto.getPwd());  
122         pstmt.setString(3, dto.getId());  
123  
124         pstmt.executeUpdate();  
125     } catch (Exception e) {  
126         e.printStackTrace();  
127     }  
128 }
```

3. DAO 작성 - update

- updatePro.jsp 다시 작성

```
1 <%@page import="cs.dit.LoginDao"%>
2 <%@page import="cs.dit.LoginDto"%>
3 <%@ page language="java" contentType="text/html; charset=UTF-8"
4     pageEncoding="UTF-8"
5     import ="java.sql.*" %>
6 <% request.setCharacterEncoding("utf-8");
7
8     String id = request.getParameter("id");
9     String name = request.getParameter("name");
10    String pwd = request.getParameter("pwd");
11
12    LoginDto dto = new LoginDto(id, name, pwd);
13    new LoginDao().update(dto);
14
15 %>
16
17 <script>
18     let ans = alert("변경되었습니다!");
19     if (!ans){
20         location.href='list.jsp';
21     }
22 </script>
```

3. DAO 작성 - delete

- delete() 메소드 작성

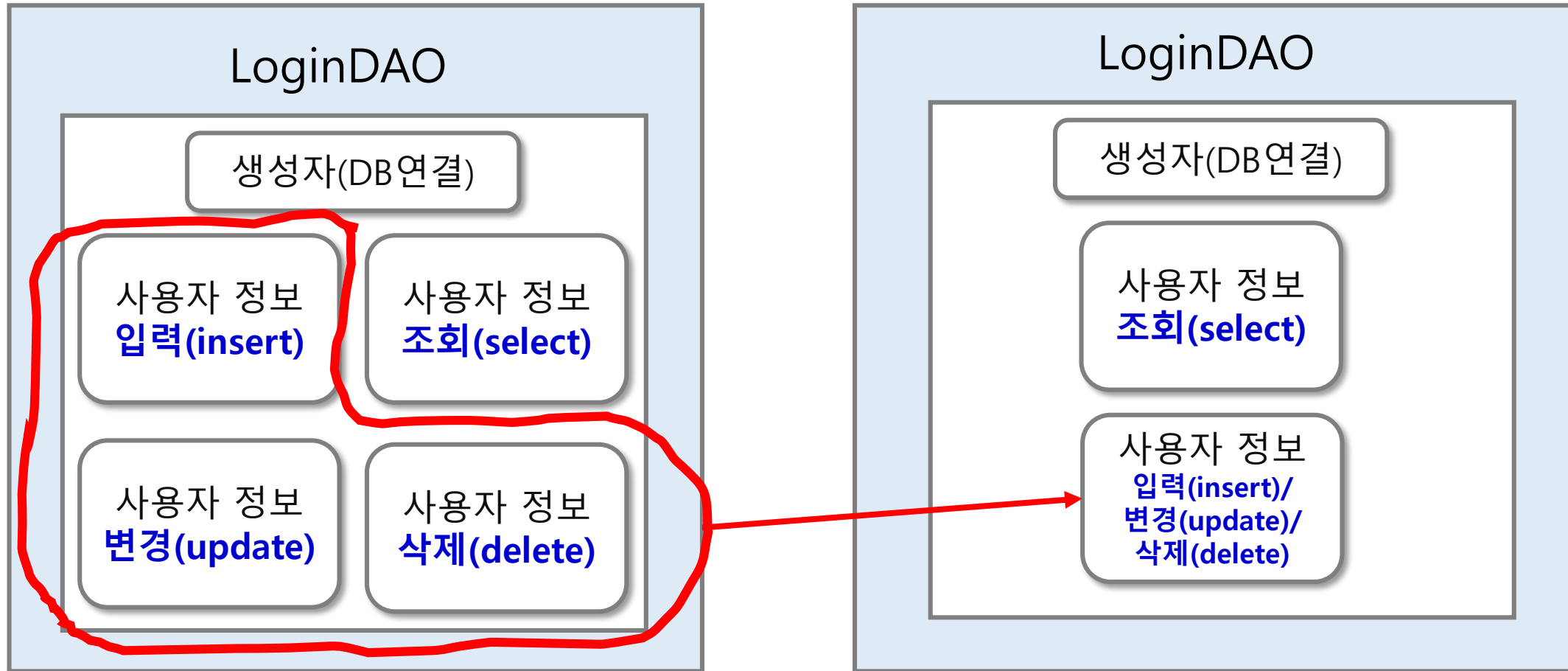
```
129 public void delete(String id) {  
130     String sql = "DELETE FROM login WHERE id =?";  
131  
132     try (  
133         Connection con = getConnection();  
134         PreparedStatement pstmt = con.prepareStatement(sql);  
135     )  
136     {  
137         pstmt.setString(1, id);  
138         pstmt.executeUpdate();  
139  
140     } catch (Exception e) {  
141         e.printStackTrace();  
142     }  
143 }
```

3. DAO 작성 - delete

- delete.jsp 다시 작성

```
1 <%@page import="cs.dit.LoginDao"%>
2 <%@ page language="java" contentType="text/html; charset=UTF-8"
3     pageEncoding="UTF-8"
4     import="java.sql.*" %>
5
6 <%
7     String id = request.getParameter("id");
8     new LoginDao().delete(id);
9
10 %>
11
12 <script>
13     let ans = alert("삭제되었습니다!");
14     if (!ans){
15         location.href='list.jsp';
16     }
17 </script>
```


코드 개선하기 - 중복 코드 제거



코드 개선하기(리팩토링) – 중복 코드 제거

- LoginDAO 수정

```
public void loginChange(LoginDto dto, String flag) {  
    PreparedStatement pstmt = null;  
  
    try(Connection con = getConnection();  
    ){  
        if(flag.equals("i")) {  
            String sql = "INSERT INTO login(ID, NAME, PWD) VALUES(?, ?, ?)";  
            pstmt = con.prepareStatement(sql);  
            pstmt.setString(1, dto.getId());  
            pstmt.setString(2, dto.getName());  
            pstmt.setString(3, dto.getPwd());  
        }else if(flag.equals("u")) {  
            String sql = "UPDATE login SET name = ?, pwd = ? WHERE id =?";  
            pstmt = con.prepareStatement(sql);  
            pstmt.setString(1, dto.getName());  
            pstmt.setString(2, dto.getPwd());  
            pstmt.setString(3, dto.getId());  
        }else if(flag.equals("d")) {  
            String sql = "DELETE FROM login WHERE id =?";  
            pstmt = con.prepareStatement(sql);  
            pstmt.setString(1, dto.getId());  
        }  
        pstmt.executeUpdate();  
    }  
    catch (Exception e) {  
        e.printStackTrace();  
    }  
    finally {  
        try {  
            if(pstmt!=null) pstmt.close();  
        } catch (SQLException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

- JSP – 처리 파일의 자바빈 호출 부분 수정

```
7 <%request.setCharacterEncoding("utf-8");  
8  
9 String id = request.getParameter("id");  
10 String name = request.getParameter("name");  
11 String pwd = request.getParameter("pwd");  
12  
13 LoginDto dto = new LoginDto(id, name, pwd);  
14 LoginDao dao = new LoginDao();  
15  
16 dao.loginChange(dto, "i");  
17  
18 response.sendRedirect("list.jsp");  
19 %>
```

"i" : insert
"u" : update
"d" : delete

참고사항

자바빈(JavaBean) 만들기

- 자바빈 작성 규칙
 - 자바빈은 **POJO**로 자바의 클래스를 만드는 것과 같은 규칙을 갖는다.

❖ 클래스 작성 순서

1. package 패키지명; //반드시 패키지 사용해야함. 기본 패키지에는 인식 안됨
2. import 패키지명을 포함한 라이브러리 클래스의 풀네임; //없으면 생략가능
3. public class 클래스명{ //인자값이 없는 기본생성자 있어야 함
.....
}

- 자바빈 저장 위치
 - 웹서버 : 웹어플리케이션폴더(컨텍스트)\WEB-INF\classes
 - 이클립스 : 프로젝트명-[Java Resources]-[src]

자바빈(JavaBean) 만들기

- 자바빈의 **클래스**를 선언

```
public class 클래스명{ }
```

- 자바 빈을 작성할 때는 접근제어의 강도가 가장 약한 **public** 주로 사용
 - 웹에서는 불특정 다수의 접근을 허용으로 누구나 접근할 수 있는 public사용
- 클래스명
 - 단어의 첫 글자는 대문자로 나머지는 소문자를 사용
 - 클래스명의 명명관습을 따름
 - 예 : public class **UtilClass**{ }

자바빈(JavaBean) 만들기

- 멤버 **변수** 선언

- 값을 저장하기 위한 필드로 접근제어자를 **private**로 선언
 - `private String userid;`

- 멤버 **메소드** 선언

- 멤버 변수에 접근하기 위한 **getXxx()**, **setXxx()** 메소드는 접근제어자를 **public**로 선언
- 멤버 변수에 값 **저장** : `setXxx()` 사용
- 멤버 변수 값 **읽기** : `getXxx()` 사용

```
Public void setId(String id){  
    this. id=id;  
}  
  
public String getId()  
    return id;  
}
```

[참고] 캡슐화(클래스)

- 캡슐화(Encapsulation)

- 정보 은닉(Information Hiding) 구현
- 객체가 무엇을 하는지 알고 **사용**할 뿐 객체의 내부는 알 필요 없음

- 캡슐화 구현 방법

- 누구나 접근할 수 있도록 **클래스**는 **public**으로 선언한다.
- 객체 내부 **변수**를 **private**으로 선언해 직접 접근할 수 없도록 한다.
- 객체 내부변수(멤버 변수)에 접근하려면 **public으로 선언된 setter/getter 메소드**를 사용한다.

객체지향프로그래밍 특성

- 상속성 : Inheritance(재사용)
- 캡슐화 : Encapsulation(정보은닉)
- 다형성 : Polymorphism(오버라이딩, 오버로딩)

자바빈(JavaBean) 만들기

- **setXxx()**메소드 작성방법

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

- 매개변수로부터 값을 받아와 저장 : **return type없음(void)** this는 자기 자신의 클래스를 가리키는 참조 객체
- **this**.id=id;
 - 넘어온 id 매개변수 값을 해당 객체의 멤버변수 id에 저장
 - 멤버변수명과 매개변수명이 같을 경우 구분을 위해 멤버변수에는 **this**가 붙는다.

- **getXxx()**메소드 작성방법

```
public String getId(){  
    return id;  
}
```

- 저장된 멤버변수를 사용하는 메소드이므로 매개변수가 필요 없음
- 저장된 값을 가져옴으로 **반드시 return type 을 지정**
 - 리턴 타입이 지정되면 해당 메소드의 마지막에 **return**문을 반드시 기술
→ **지정하지 않으면 에러 발생.**

자바빈(JavaBean) 만들기

- 클래스 작성
 - 자바빈 작성 위치
 - 이클립스
 - 프로젝트명-[Java Resources]-[src]

```
1 package lecture.ch10;
2
3 public class Test {
4     private String firstName;
5     private String lastName;
6
7     public String getFirstName() {
8         return firstName;
9     }
10    public void setFirstName(String firstName) {
11        this.firstName = firstName;
12    }
13    public String getLastName() {
14        return lastName;
15    }
16    public void setLastName(String lastName) {
17        this.lastName = lastName;
18    }
19 }
```

자바빈(JavaBean) 만들기

- Test 객체에 데이터를 저장하고 출력하기

```
1 <%@page import="lecture.ch10.Test"%>
2 <%@ page language="java" contentType="text/html; charset=UTF-8"
3   pageEncoding="UTF-8"%>
4 <%
5   //Test 객체에 데이터 입력
6   Test t = new Test();
7   t.setFirstName("gildong");
8   t.setLastName("홍길동");
9 %>
10 <!DOCTYPE html>
11 <html>
12 <head>
13   <meta charset="UTF-8">
14   <title>Test객체에 데이터 입력</title>
15 </head>
16 <body>
17   <%=t.getFirstName() %><br>
18   <%=t.getLastName() %>
19 </body>
20 </html>
```

Test 객체에 데이터 저장

Test 객체에 데이터 출력

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

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

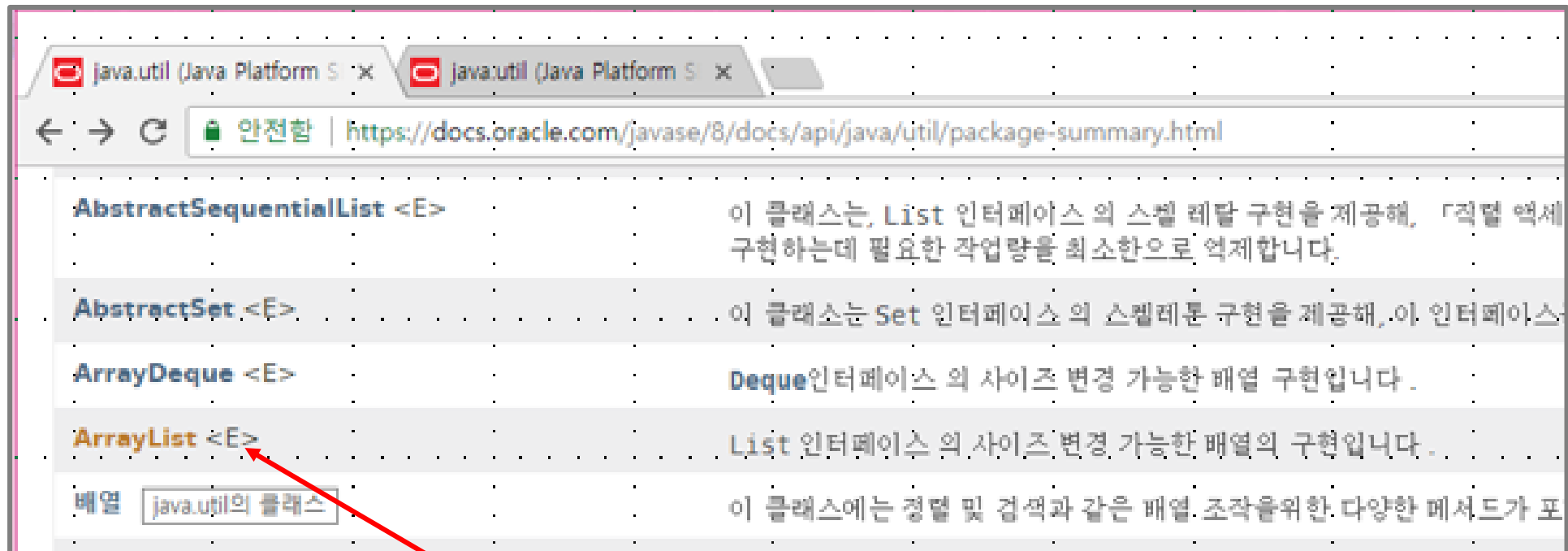
Array

- ✓ 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

Collection Framework

- 오라클 웹사이트의 문서 참조

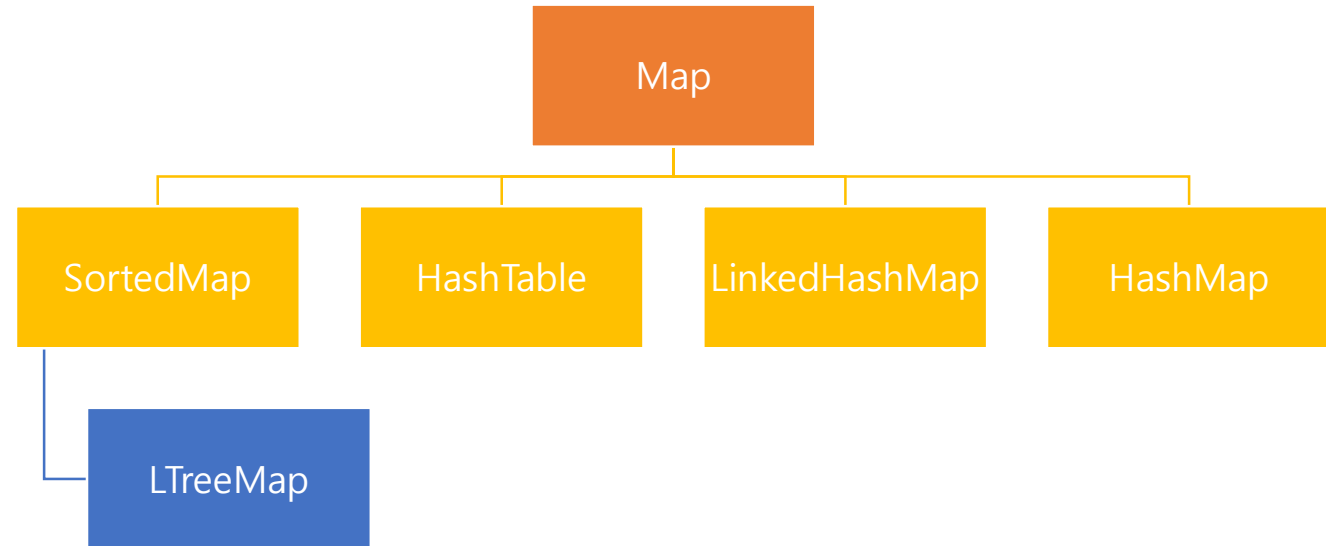
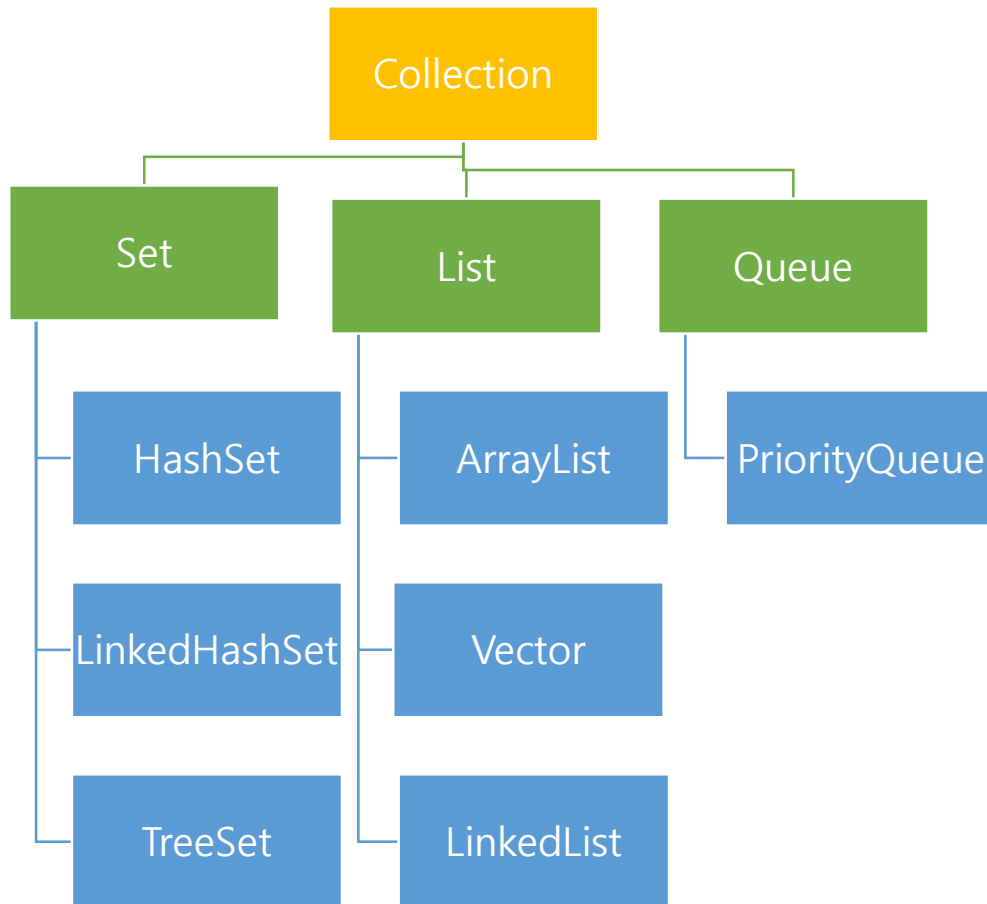
<https://docs.oracle.com/javase/8/docs/api/java/util/package-summary.html>



- 컬렉션은 Generic으로 정의됨

Collection Framework

- 다양한 상황에서 사용할 수 있는 Container 객체(데이터를 저장하는 자료 구조) <https://docs.oracle.com/javase/8/docs/api/java/util/package-summary.html>
- `java.util.*` 패키지에 존재



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	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 내부에서 사용할 데이터 타입을 **인스턴스를 생성할 때 확정**하는 기능
- 다양한 타입의 객체들을 다루는 메서드나 컬렉션 클래스에서 컴파일할 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 형변환되어 발생할 수 있는 오류를 줄여 줌
 - 타입체크와 형 변환을 생략으로 중복코드 제거(코드가 간결해 짐).
- Collection Framework을 사용하려면 Generic을 기본적으로 알아야 함

- 기존에는 다양한 종류의 타입을 다루는 메서드의 매개변수나 반환타입으로 Object의 참조 변수를 많이 사용했고, 그로 인해 형변환이 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

참조 사이트

- <http://bootstrapk.com/>