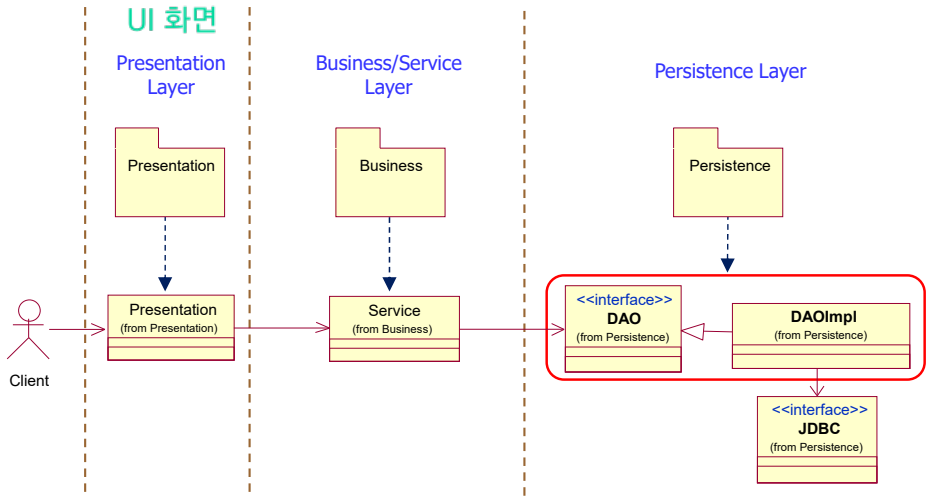
# 8. DAO, DTO

# 3-tier Application Architecture

- 3-계층 구조






# DAO Pattern

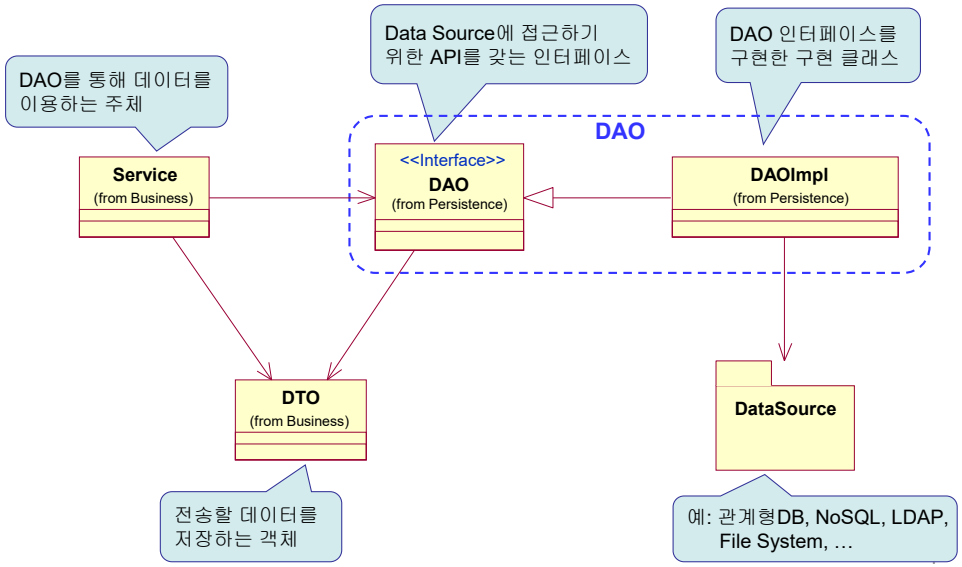
- 영속성 계층(Persistence Layer)
  - 애플리케이션의 객체 및 데이터를 영속적으로 저장 관리하는 계층
  - Database, File System, LDAP Server 등 데이터 저장소(Data Source) 이용
  - 비즈니스 계층에서 영속성 계층에 접근하기 위해 이용할 수 있는 구성 요소 필요
    - 데이터 저장소의 유형에 관계없이 이용할 수 있는 **공통의 API** 제공
    - **데이터 저장소에 독립적인** 애플리케이션 개발 가능

- Data Access Object(DAO)
  - 데이터 저장소를 공통의 인터페이스를 통해 이용할 수 있도록 해주는 구성 요소 (interface 및 구현 class)
  - 데이터 저장소에 대한 접근 및 이용 방법을 추상화하고 캡슐화 함



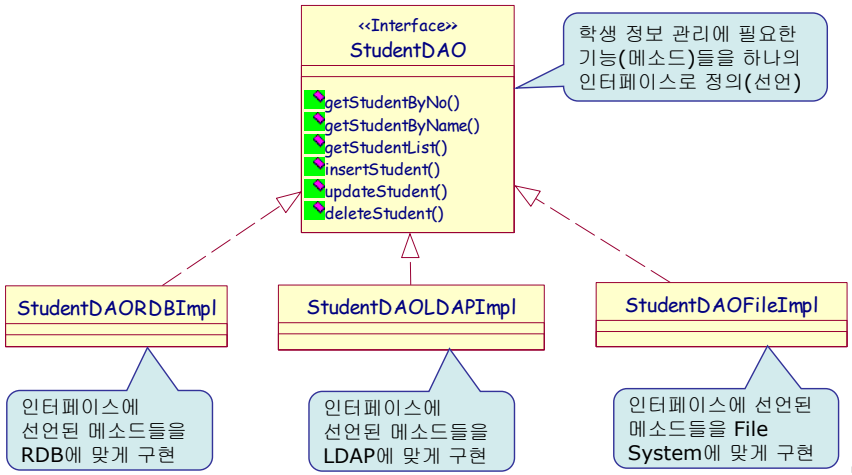
# DAO Pattern

- 구성 요소



# DAO Pattern

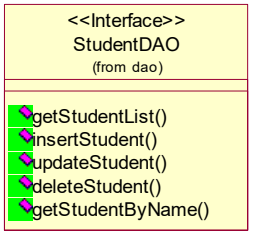
- 학생 정보 관리를 위한 DAO의 예
  - 하나의 인터페이스에 대해 여러 구현 클래스들을 정의해서 사용 가능






# DAO Pattern

- DAO 인터페이스 정의
  - Data Source에 대해 수행할 작업 선정
    - 데이터 삽입, 수정, 삭제, 검색
  - 예: 학생정보 관리
    - 전체 학생들의 정보를 검색
    - 새로운 학생의 정보 추가
    - 특정 학생의 정보 수정
    - 특정 학생의 정보 삭제
    - 이름으로 학생 정보 검색
    - 학번으로 학생 정보 검색 등





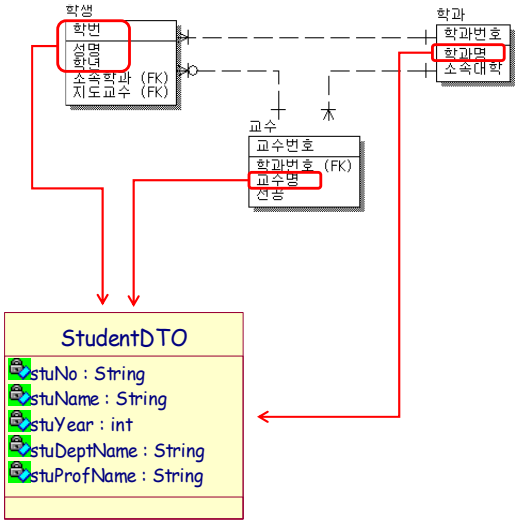

# DTO Pattern

- Data Transfer Object(DTO)
  - Java 객체들 사이의 메소드 호출에 필요한 데이터들을 포함하는 객체
    - 여러 데이터들을 하나의 DTO로 묶어 다른 객체나 sub-system에 전송함
    - Business Layer와 Persistence Layer 사이의 데이터 전송뿐만 아니라, Presentation Layer와 Business Layer 간의 데이터 전송에도 이용 가능
  - JavaBeans/POJO(Plain Old Java Object)로 구현
    - 데이터를 저장하기 위한 필드들과 이에 대한 getter/setter method 정의
    - 일반적으로 business method는 포함하지 않음
    - 네트워크를 통한 전송이나 파일 시스템 저장을 가능하게 하기 위해 직렬화(serialization)가 가능하도록 java.io.Serializable interface를 구현



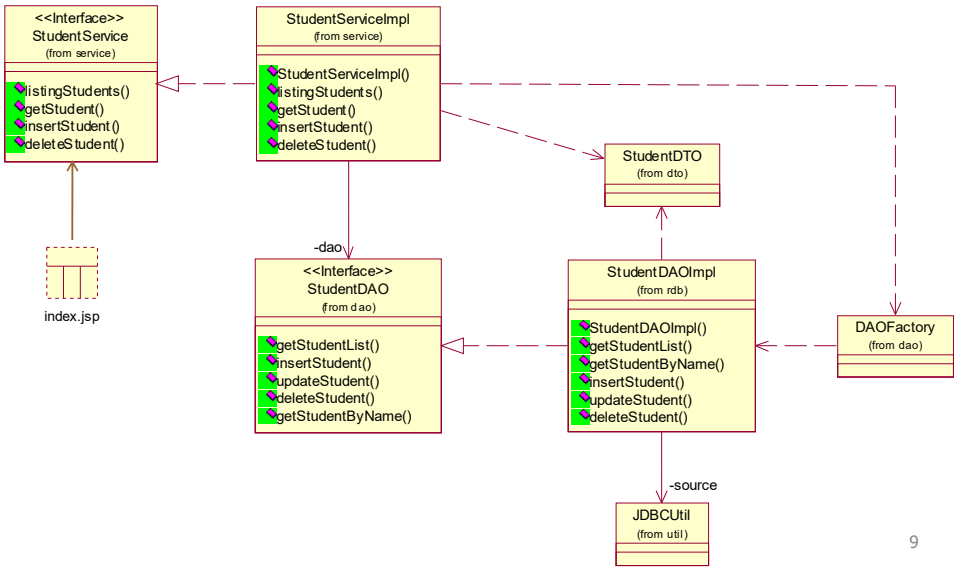
# DTO Pattern

- 예: 학생정보 DTO
  - 관련 업무에 필요한 데이터들로 구성
    - 학번
    - 성명
    - 학년
    - 소속학과(번호)
    - 지도교수(번호)
    - 학과명
    - 지도교수명
  - 데이터들은 일반적으로 여러 테이블에 나뉘어 저장/관리됨
  - → 하나의 DTO는 여러 테이블에 저장된 데이터를 포함 가능



# DAO & DTO 사용 예: 학생정보 관리 시스템

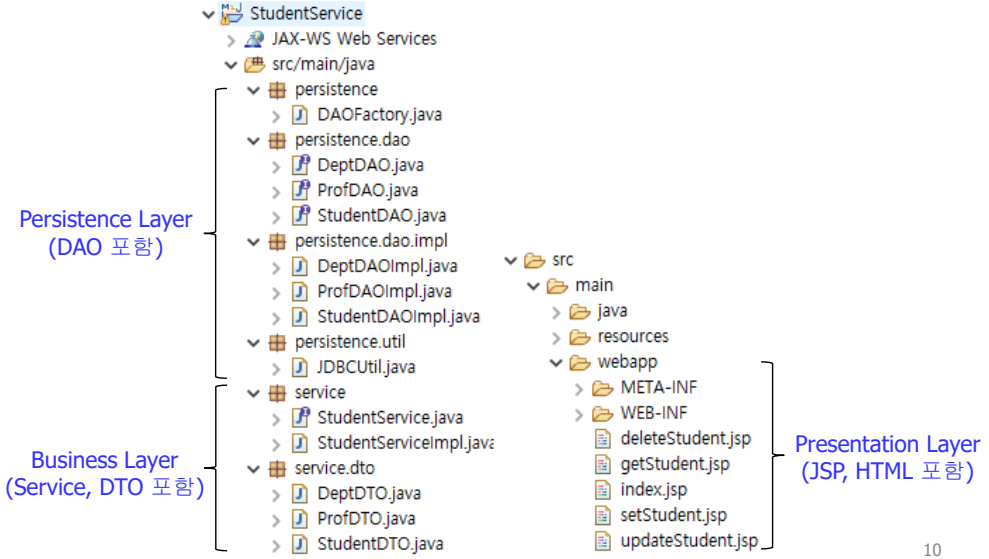
- Class diagram(일부)






# DAO & DTO 사용 예: 학생정보 관리 시스템

- Project 구조






# DAO & DTO 사용 예: Service interface

- **StudentService**
  - 학생정보 서비스를 위한 Façade Interface

```
import student.dto.StudentDTO;

public interface StudentService {
    public List<StudentDTO> ListingStudents();          // 전체 학생정보 리스트 반환
    public StudentDTO getStudent(String stuNo);         // 학번에 해당하는 학생 정보 반환
    public int insertStudent(StudentDTO stu);           // 학생정보 추가
    public int updateStudent(StudentDTO stu);           // 학생정보 갱신
    public int deleteStudent(String stuNo);             // 학생정보 삭제
    public List<StudentDTO> searchStudents(String name);   // 이름으로 검색
    …
}
```



# DAO & DTO 사용 예: Service class

- **StudentServiceImpl**
  - 학생정보 서비스를 위한 Façade Class

```
import java.util.List;
…
public class StudentServiceImpl implements StudentService {
    private StudentDAO dao = null;                  // interface type으로 정의

    public StudentServiceImpl() {
        DAOFactory factory = new DAOFactory();
        dao = factory.getStudentDAO();              // StudentDAOImpl 객체 반환
    }
    public List<StudentDTO> ListingStudents() {
        return dao.getStudentList();
    }
    public StudentDTO getStudent(String stuNo)  throws … {
        StudentDTO student = dao.getStudentByNo(stuNo);
        if (student == null)
            throw new UserNotFoundException(stuNo + "는 존재하지 않는 학번");
        return student;
    }
    …
}
```

# DAO & DTO 사용 예: DTO class

- **StudentDTO**

```
public class StudentDTO {
    private String stuNo = null;                    // 학번 (PK)
    private String stuName = null;                  // 성명
    private String pwd = null;                      // 암호
    private String stuPhoneNo = null;               // 학생 연락처
    private String year = null;                     // 학년
    private String profNo = null;                   // 지도교수번호 (FK)
    private String deptNo = null;                   // 학과 번호 (FK)
    private String deptName = null;                 // 학과명 (from DEPT table)
    private String profName = null;                 // 지도교수명 (from PROF table)

    public String getProfName() {                   // getter
        return profName;
    }
    public void setProfName(String profName) {      // setter
        this.profName = profName;
    }
    …
}
```



# DAO & DTO 사용 예: DAO factory class

- **DAOFactory**
  - DAO를 구현한 클래스가 여러 개 있을 경우, 각 클래스의 객체(instance)를 생성하여 반환하는 역할을 수행하는 Factory Class
  - DAO를 사용하는 클래스에서 특정 DAO 구현 클래스와 밀접하게 결합되는 것을 막기 위한 용도

```
import persistence.dao.*;
import persistence.dao.impl.*;

public class DAOFactory {
    public StudentDAO getStudentDAO() {
        return new StudentDAOImpl();
    }
    public DeptDAO getDeptDAO() { return new DeptDAOImpl(); }
    public ProfDAO getProfDAO() { return new ProfDAOImpl(); }
}
```

인터페이스 타입의 객체로 리턴



# DAO & DTO 사용 예: DAO interface

- **StudentDAO**
  - 학생정보 DAO를 위한 Interface 정의

```
import java.util.List;
import service.dto.StudentDTO;

public interface StudentDAO {
    public List<StudentDTO> getStudentList();
    public StudentDTO getStudentByNo(String stuNo);
    public List<StudentDTO> getStudentsByName(String name);
    public int insertStudent(StudentDTO stu);
    public int updateStudent(StudentDTO stu);
    public int deleteStudent(String stuNo);
}
```



# DAO & DTO 사용 예: DAO class

- **StudentDAOImpl**
  - 학생정보 DAO 구현 클래스
    - JDBC API 기반 (JDBCUtil 클래스 활용)

```
import java.util.List;
import java.util.ArrayList;
import java.sql.*;
import service.dto.*;
import persistence.DAOFactory;
import persistence.dao.*;
import persistence.util.JDBCUtil;

public class StudentDAOImpl implements StudentDAO {
    private JDBCUtil jdbcUtil = null;           // JDBCUtil 객체 참조 변수 선언

    public  StudentDAOImpl() {                  // 생성자
        jdbcUtil = new JDBCUtil();              // JDBCUtil 객체 생성 및 이용
    }

    // 학생 기본 정보를 포함하는 SELECT 절
    private static String query = "SELECT STUDENT.STU_NO, " +
                                  "STUDENT.STU_NAME, " +
                                  "STUDENT.PASSWORD AS STU_PASSWD, " +
                                  "STUDENT.STU_PHONE_NO, " +
                                  "STUDENT.YEAR AS STU_YEAR ";
```

# DAO & DTO 사용 예: DAO class

```
public List<StudentDTO> getStudentList() {
    String allQuery = query + ", " + "STUDENT.P_NO AS PROF_NO, "+
                                     "STUDENT.D_NO AS DEPT_NO " +
                                     "FROM STUDENT ORDER BY STU_NO ASC";
    jdbcUtil.setSqlAndParameters(allQuery, null);          // JDBCUtil 객체에 질의문 설정
    try {
        ResultSet rs = jdbcUtil.executeQuery();            // 질의 실행
        List<StudentDTO> list = new ArrayList<StudentDTO>();  // DTO들을 담기 위한 리스트 생성
        while (rs.next()) {
            StudentDTO dto = new StudentDTO();       // StudentDTO 객체 생성 후 검색 결과 저장
            dto.setStuNo(rs.getString("STU_NO"));
            dto.setStuName(rs.getString("STU_NAME"));
            …
            list.add(dto);                  // 리스트에 DTO 객체 저장
        }
        return list;                        // 학생정보를 저장한 DTO 객체들의 리스트를 반환
    } catch (Exception ex) {
        ex.printStackTrace();
    } finally {
        jdbcUtil.close();                   // ResultSet, PreparedStatement, Connection 객체 반환
    }  return null;
}
```

connenction없을



# DAO & DTO 사용 예: DAO class

```
public StudentDTO getStudentByNo(String stuNo) {
    String searchQuery = query + ", " + "PROFESSOR.P_NAME AS PROF_NAME, "+
                                        "DEPT.D_NAME AS DEPT_NAME " +
                                        "FROM STUDENT, PROFESSOR, DEPT " +
                                        "WHERE STUDENT.STU_NO = ? AND " +
                                            "STUDENT.P_NO = PROFESSOR.P_NO AND " +
                                            "STUDENT.D_NO = DEPT.D_NO ";
    Object[] param = new Object[] { stuNo };
    jdbcUtil.setSqlAndParameters(searchQuery, param);   // JDBCUtil에 질의 및 매개변수 설정
    try {
        ResultSet rs = jdbcUtil.executeQuery();      // 질의 실행
        StudentDTO stu = null;
        if (rs.next()) {                             // 질의 결과 존재
            stu = new StudentDTO();                  // StudentDTO 객체를 생성하여 학생 정보 저장
            stu.setStuNo(rs.getString("STU_NO"));
            stu.setStuName(rs.getString("STU_NAME"));
            …
            stu.setProfName(rs.getString("PROF_NAME"));
            stu.setDept(rs.getString("DEPT_NAME"));
        }
        return stu;                                  // 학생 정보를 담고 있는 StudentDTO 객체 반환
    } catch (Exception ex) { …
    } finally {  jdbcUtil.close(); }  return null;
}
```



# DAO & DTO 사용 예: DAO class

```
public int insertStudent(StudentDTO stu) {
    int result = 0;
    String insert = "INSERT INTO STUDENT (STU_NO, STU_NAME, PASSWORD, " +
                    "STU_PHONE_NO, YEAR, P_NO, D_NO) VALUES (?, ?, ?, ?, ?, ?, ?) ";
    DAOFactory factory = new DAOFactory();
    ProfDAO profDAO = factory.getProfDAO();      // factory를 통해 ProfDAO 객체 획득
    ProfDTO profDTO = profDAO.getProfByName(stu.getProfName());
    String profNo = profDTO.getProfNo();         // 교수번호를 구함
    if (profNo == null) { System.out.println("해당 교수가 없습니다."); return 0; }

    DeptDAO deptDAO = factory.getDeptDAO();    // factory를 통해 DeptDAO 객체 획득
    DeptDTO deptDTO = deptDAO.getDeptByName(stu.getDept());
    String deptNo = deptDTO.getDeptNo();         // 학과번호를 구함
    if (deptNo == null) { System.out.println("해당 학과가 없습니다."); return 0; }

    Object[] param = new Object[] {stu.getStuNo(), stu.getStuName(), stu.getPwd(),
                                   stu.getStuPhoneNo(), stu.getYear(), profNo, deptNo};
    jdbcUtil.setSqlAndParameters(insert, param);  // JDBCUtil에 insert문과 매개변수 설정
    try {
        result = jdbcUtil.executeUpdate();          // insert 문 실행
    } catch (Exception ex) {
        jdbcUtil.rollback(); ex.printStackTrace();  // rollback 실행
    } finally {
        jdbcUtil.commit();                          // commit 실행
        jdbcUtil.close();                           // ResultSet, PreparedStatement 등 자원 반환
    }   return result;          }
```



# DAO & DTO 사용 예: JDBC utility class

- **JDBCUtil**
  - JDBC API 사용 시 코드의 중복성과 복잡성을 제거하기 위한 utility class

```
import java.io.IOException;
import java.io.InputStream;
import java.util.Properties;
import javax.sql.DataSource;
import java.sql.*;
import org.apache.commons.dbcp2.BasicDataSource;     // Apache Commons DBCP2 라이브러리 이용

public class JDBCUtil {
    private String sql = null;                  // 실행할 query 문 저장
    private Object[] parameters = null;         // PreparedStatement 의 매개변수 값을 저장하는 배열
    private static DataSource ds = null;        // DBCP DataSource 객체 저장 및 유지
    private static Connection conn = null;      // Connection 객체 저장 및 유지
    private PreparedStatement pstmt = null;
    private CallableStatement cstmt = null;
    private ResultSet rs = null;
    private static Properties prop = new Properties();

    {   // static initializer
        // property 파일에 저장된 DB 접속 정보를 Properties 객체로 읽어 들임 (예외 처리 생략)
        InputStream input
            = getClass().getResourceAsStream("/dbinfo.properties");  // DB 접속 정보 설정 파일
        prop.load(input);
    }
```

# DAO & DTO 사용 예: JDBC utility class

```
private static void initJDBCUtil() {          // DBCP 초기화
    try {
        if (ds == null) {                     // DBCP 설정
            BasicDataSource bds = new BasicDataSource();
            bds.setDriverClassName(prop.getProperty("db.driver"));
            bds.setUrl(prop.getProperty("db.url"));
            bds.setUsername(prop.getProperty("db.username"));
            bds.setPassword(prop.getProperty("db.password"));
            ds = bds;
        }
    } catch (Exception ex) { ex.printStackTrace(); }
}

public JDBCUtil() {  initJDBCUtil();  }      // 기본 생성자
…

public void setSql(String sql) {              // sql 필드 setter
    this.sql = sql;
}

public void setParameters(Object[] parameters) {    // parameters 필드 setter
    this.parameters = parameters;
}
```



# DAO & DTO 사용 예: JDBC utility class

```
public void setSqlAndParameters(String sql, Object[] parameters) {  // sql 및 매개변수 저장
    this.sql = sql;
    this.parameters = parameters;
}

private int getParameterSize() {                    // 매개변수의 개수를 반환
    return parameters == null ? 0 : parameters.length;
}

private Object getParameter(int index)       // 매개변수 배열에서 특정 매개변수를 찾아 반환
        throws Exception {
    if (index >= getParameterSize())
        throw new Exception("INDEX 값이 파라미터의 갯수보다 많습니다.");
    return parameters[index];
}

private PreparedStatement getPreparedStatement() throws SQLException {
    if (conn == null) {                             // Connection이 없을 경우
        conn = ds.getConnection();                  // 새로운 Connection 획득
        conn.setAutoCommit(false);                  // DML문에 대한 자동 commit 기능을 해제함
    }                                               // Connection이 있을 경우 재사용
    if (pstmt != null) pstmt.close();          // 기존 PreparedStatement 객체가 존재할 경우 해제
    pstmt = conn.prepareStatement(sql);             // 새로운 PreparedStatement 객체 생성
    return pstmt;
}
```



# DAO & DTO 사용 예: JDBC utility class

```
public ResultSet executeQuery() {                   // PreparedStatement를 통해 질의 실행
    try {
        pstmt = getPreparedStatement();             // PreparedStatement 획득
        for (int i = 0; i < getParameterSize(); i++) {
            pstmt.setObject(i + 1, getParameter(i));  // 매개변수 존재 시 각각 설정
        }
        rs = pstmt.executeQuery();                  // 질의 실행 후 ResultSet 객체 반환
        return rs;
    } catch (Exception ex) { ex.printStackTrace(); }
    return null;
}

public int executeUpdate()                          // PreparedStatement를 통해 DML문 실행
        throws SQLException, Exception {
    pstmt = getPreparedStatement();
    for (int i = 0; i < getParameterSize(); i++) {
        if (getParameter(i) == null) {
            pstmt.setString(i + 1, null);
        } else {
            pstmt.setObject(i + 1, getParameter(i));    // 매개변수 설정
        }
    }
    return pstmt.executeUpdate();                   // Insert/Delete/Update 실행 후 count 값 반환
}
```



# DAO & DTO 사용 예: JDBC utility class

```
public void close() {                               // 자원 해제
    if (rs != null) {
        try { rs.close();  rs = null;
        } catch (SQLException ex) { ex.printStackTrace(); }
    }
    if (pstmt != null) {
        try { pstmt.close();    pstmt = null;
        } catch (SQLException ex) { ex.printStackTrace(); }
    }
    if (conn != null) {
        try { conn.close();   conn = null;
        } catch (SQLException ex) { ex.printStackTrace(); }
    }
}

public void commit() {                              // 트랜잭션 commit 수행
    try { conn.commit();
    } catch (SQLException ex) { ex.printStackTrace(); }
}

public void rollback() {                            // 트랜잭션 rollback 수행
    try { conn.rollback();
    } catch (SQLException ex) { ex.printStackTrace(); }
}
```

# DAO & DTO 사용 예: JSP

- index.jsp

```
<%@ page contentType="text/html; charset=utf-8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ page import="java.util.List, java.util.Iterator, service.*, service.dto.StudentDTO" %>
<html><head></head><body>
<%
    StudentService studentSvc = new StudentServiceImpl();  // 학생정보 서비스 객체 생성
    List<StudentDTO> list = studentSvc.ListingStudents();    // 전체 학생들의 정보 검색
    request.setAttribute("studentList", list);
%>
<h2>학생 목록</h2>
<c:forEach var="student" items="${studentList}">
    ${student.stuNo}      ${student.stuName}  
    <a href="<c:url value='getStudent.jsp'>
        <c:param name='stuName' value='${student.stuName}'/>
        </c:url>">조회</a>  
    <a href="<c:url value='deleteStudent.jsp'>
        <c:param name='stuNo' value='${student.stuNo}'/>
        </c:url>">삭제</a><br>
</c:forEach>
<br>
```



# DAO & DTO 사용 예: JSP

- index.jsp (contd.)

```
<h2>등록</h2>
<form method="post" action="setStudent.jsp">
학번:    <input type="text" name="stuNo" value="20150003" /><br>
이름:    <input type="text" name="stuName" value="Jain" /><br>
비밀번호: <input type="text" name="pwd" value="1111" /><br>
연락처: <input type="text" name="stuPhoneNo" value="010-3456-7890" /><br>
학년:    <input type="text" name="year" value="3" /><br>
학과:    <input type="text" name="dept" value="Computer" /><br>
지도교수: <input type="text" name="profName" value="Andy" /><br>
<input type="submit" value="등록" />
</form>
<br>
<h2>변경</h2>
<form method="post" action="updateStudent.jsp">
학번:    <input type="text" name="stuNo" value="20150003" /><br>
연락처: <input type="text" name="stuPhoneNo" value="010-1111-2222" /><br>
학년:    <input type="text" name="year" value="4" /><br>
<input type="submit" value="변경" />
</form>
```



# DAO & DTO 사용 예: JSP

- getStudent.jsp

```
<%@ page language="java" contentType="text/html; charset=utf-8" %>
<%@ page import="service.*, service.dto.StudentDTO" %>
<jsp:useBean id="stu" class="service.dto.StudentDTO" scope="page" />
<jsp:setProperty name="stu" property="*" />        ← DTO 객체를 통해 이름이 전달됨
<html>
<head><title>Insert title here</title></head>
<body>
<%
    StudentService studentSvc = new StudentServiceImpl();   // StudentService 객체 생성
    StudentDTO student = studentSvc.getStudent(stu.getStuName());  // 학생 정보 검색
    if (student != null ) {
        out.print("학번: " + student.getStuNo()+ "<br>");
        out.print("이름: " + student.getStuName()+ "<br>");
        out.print("학년: " + student.getYear()+ "<br>");
        out.print("학과: " + student.getDept() + "<br>");
        out.print("전화번호: " + student.getStuPhoneNo()+ "<br>");
        out.print("지도교수: " + student.getProfName()+ "<br>");
    }
%>
</body></html>
```



# DAO & DTO 사용 예: JSP

- setStudent.jsp

```
<%@ page language="java" contentType="text/html; charset=utf-8" %>
<%@ page import="service.*, service.dto.StudentDTO" %>
<jsp:useBean id="stu" class="service.dto.StudentDTO" scope="page" />
<jsp:setProperty name="stu" property="*" />      ← DTO를 통해 학생정보가 전달됨
<html>
<head>…</head>
<body>
<%
    StudentService studentSvc = new StudentServiceImpl();  // StudentService 객체 생성
    int result = studentSvc.insertStudent(stu);                    // 학생 정보 삽입
    System.out.println(result + " 개의 학생정보가 추가되었습니다.");
    response.sendRedirect("index.jsp");
%>
</body>
</html>
```

# DAO & DTO 사용 예: JSP

- updateStudent.jsp

```
<%@ page language="java" contentType="text/html; charset=utf-8" %>
<%@ page import="service.*, service.dto.StudentDTO" %>
<jsp:useBean id="stu" class="service.dto.StudentDTO" scope="page" />
<jsp:setProperty name="stu" property="*" />      ← DTO를 통해 학생정보가 전달됨
<html>
<head>…</head>
<body>
<%
    StudentService studentService = new StudentServiceImpl();
    int result = studentService.updateStudent(stu);                    // 학생 정보 갱신
    System.out.println(result + " 개의 학생정보가 수정되었습니다.");
    response.sendRedirect("index.jsp");
%>
</body>
</html>
```



# DAO & DTO 사용 예: JSP

- deleteStudent.jsp

```
<%@ page language="java" contentType="text/html; charset=utf-8"%>
<%@ page import="service.*, service.dto.StudentDTO" %>
<jsp:useBean id="stu" class="service.dto.StudentDTO" scope="page" />
<jsp:setProperty name="stu" property="*" />      ← DTO를 통해 학번이 전달됨
<html>
<head>…</head>
<body>
<%
    StudentService studentService = new StudentServiceImpl();
    int result = studentService.deleteStudent(stu.getStuNo());    // 학생 정보 삭제
    System.out.println(result + " 개의 학생정보가 삭제되었습니다.");
    response.sendRedirect("index.jsp");
%>
</body>
</html>
```