

Oracle 연동

1. JDBC

❖JDBC(Java Database Connectivity)

- ✓ 자바 프로그램과 데이터베이스에 대한 인터페이스로 데이터베이스에 접근하는 방법과 SQL 명령들을 전달해서 수행할 수 있도록 해주는 방법을 제공하는 라이브러리
- ✓ JDBC 인터페이스와 JDBC 드라이버로 구성
- ✓ 응용프로그램에서는 SQL문 만들어 JDBC Interface를 통해 전송하고 실제 구현 클래스인 JDBC 드라이버에서 DBMS에 접속을 시도하여 SQL문을 전송
- ✓ DBMS는 SQL 구문을 실행 한 후 결과를 JDBC Driver와 JDBC Interface에게 전달하고 이를 다시 응용프로그램으로 전달해서 SQL문의 결과를 사용
- ✓ JDBC의 역할은 Application과 DBMS의 Bridge 역할
- ✓ R 프로그래밍 언어에서도 이 방법을 통해서 데이터베이스 연동이 가능

❖JDBC 프로그래밍 방법

- ✓ 데이터베이스와 연결하는 드라이버 파일을 찾아서 로드
- ✓ 연결을 관리하는 Connection 인스턴스 생성
- ✓ sql 구문을 처리할 Statement, PreparedStatement, CallableStatement 인스턴스 생성
- ✓ 구문 실행 - select 구문 인 경우 ResultSet 인스턴스를 통한 sql 결과 처리
- ✓ 접속 종료

1. JDBC

❖ 드라이버 로드 및 데이터베이스 접속과 접속 종료

- ✓ `Class.forName(드라이버 이름);` => 드라이버 로드(`ClassNotFoundException`이 발생할 수 있는데 이 경우는 jdbc 연결 파일을 추가하지 않았거나 클래스 이름이 잘못된 경우)
- ✓ `Connection Connection변수명= DriverManager.getConnection(데이터베이스이름, "아이디","비밀번호");` => 접속(`SQLException` 이 발생할 수 있는데 이 때는 데이터베이스 이름이나 아이디 및 비밀번호 와 데이터베이스 활성화 여부 확인)
- ✓ `Connection변수명.close();` => 접속 종료

1. JDBC

❖ Oracle

- ✓ 드라이버 이름: oracle.jdbc.driver.OracleDriver
- ✓ 데이터베이스 이름: jdbc:oracle:thin:@ip:포트번호:데이터베이스이름
sid 대신에 ServiceName 으로 접속하는 경우에는 :데이터베이스이름 대신에 /서비스
이름 으로 작성
- ✓ ip는 로컬 컴퓨터이면 localhost
- ✓ 포트번호는 기본적으로 1521
- ✓ 데이터베이스 이름은 기본적으로 orcl(Standard Edition 이나 Enterprise Edition) 이나
xe(Express Edition)
- ✓ Oracle 홈페이지에서 JDBC 드라이버를 다운로드 받거나 설치된 폴더에서 찾아서
ojdbc6.jar 파일을 자바가 설치된 경로의 자바 설치폴더/ jre버전/lib/ext에 복사하거나
프로젝트의 build path에 드라이버 파일을 추가
- ✓ 오라클이 설치된 경우 드라이버는 C:\Wapp\W데이터베이스이름
Wproduct\W11.2.0\Wdbhome_1\Wjdbc\Wlib
- ✓ 오라클 홈페이지: <http://www.oracle.com/technetwork/database/enterprise-edition/jdbc-112010-090769.html>

실습(OracleConnect.java)

```
import java.sql.Connection;
import java.sql.DriverManager;
```

```
public class OracleConnect {
    static final String sid = "xe";
    static final String id = "scott";
    static final String pw = "tiger";
```

```
    public static void main(String[] args) {
```

```
        System.out.println("오라클 JDBC 드라이버 로딩중...");
```

```
        //오라클 데이터베이스 서버 접속
```

```
        Connection connection = null;
```

```
        try {
```

```
            // 접속할 데이터베이스의 URL을 만든다.
```

```
            String url = "jdbc:oracle:thin:@localhost:1521:" + sid;
```

```
            // 접속한다(Login)
```

```
            connection = DriverManager.getConnection(url, id, pw);
```

```
            System.out.println("접속 / 로그인 성공");
```

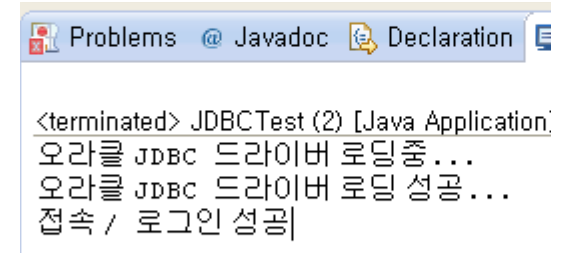
```
        }
```

```
        catch (Exception e) {
```

```
            System.out.println("접속 / 로그인 실패");
```

```
            System.out.println(e.getMessage());
```

```
        }
```



실습(OracleConnect.java)

```
//연결을 종료한다.
```

```
try {
```

```
    connection.close();
```

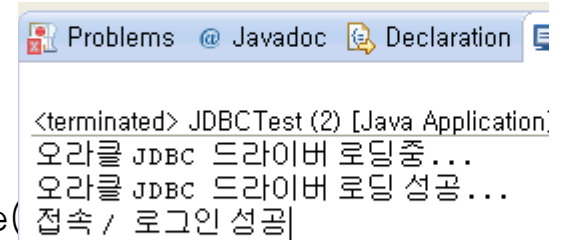
```
} catch (Exception e) {
```

```
    System.out.println(e.getMessage());
```

```
}
```

```
}
```

```
}
```



2. 접속 시 발생하는 예외

- ❖ No suitable driver found for jdbc:oracle:thin:@localhost:1521:xe: OJDBC드라이버에 해당하는 jar 파일이 Build Path에 제대로 추가됐는지 확인
- ❖ IO 오류: The Network Adapter could not establish the connection : 데이터베이스 접속 url을 확인하고 데이터베이스가 서비스 중인지 확인
- ❖ ORA-01017: invalid username/password; logon denied : 계정 정보를 확인

3. SQL 실행

❖ Connection 인스턴스의 트랜잭션 관리 메소드

- ✓ `setAutoCommit(true or false)`
- ✓ `rollback()`
- ✓ `commit()`
- ✓ 기본은 `autoCommit` 이며 직접 관리하기 위해서 `setAutoCommit`에 `false`를 전달 한 후 사용

3. SQL 실행

❖ SQL 실행 클래스

- ✓ Statement: 데이터베이스에 쿼리를 보내기 위한 클래스
 - Connection 인스턴스의 createStatement()로 생성
 - 이 클래스의 인스턴스는 Connection 인스턴스의 연결 정보를 가져와서 DB에 접근하므로 이 인스턴스를 사용하기 위해서는 Connection 인스턴스가 먼저 존재
 - insert, update, delete 구문실행: 영향 받은 행의 개수나 0을 리턴
 - ◆ 정수변수 = statement변수.executeUpdate(dml문장) ;
 - ◆ 리턴 값이 0이면 쿼리가 문법에는 맞지만 쿼리의 결과로 영향 받은 행이 없는 것이고 양수 값이면 영향 받은 행의 개수가 있는 것이므로 쿼리를 수행하고 결과로 변경된 행이 있는 것
 - select 구문 실행
 - ◆ ResultSet 변수명 = statement변수.executeQuery(select문장) ;
 - 종료: statement변수.close();

3. SQL 실행

❖ PreparedStatement

- ✓ SQL의 틀을 미리 정해 놓고, 나중에 값을 지정하는 방식
- ✓ PreparedStatement의 일반적 사용 - 값을 대입하는 곳에 ?를 설정하고 나중에 대입

```
PreparedStatement pstmt = conn.prepareStatement(  
    "insert into MEMBER (MEMBERID, NAME, EMAIL) values (?, ?, ?)");  
pstmt.setString(1, "ggangpae1"); // 첫번째 물음표의 값 지정  
pstmt.setString(2, "박문석"); // 두번째 물음표의 값 지정  
pstmt.executeUpdate();
```

- ✓ 쿼리 실행 관련 메소드
 - ResultSet executeQuery() - SELECT 쿼리를 실행할 때 사용되며 ResultSet을 결과값으로 리턴
 - int executeUpdate() - INSERT, UPDATE, DELETE 쿼리를 실행할 때 사용되며, 실행 결과 변경된 레코드의 개수를 리턴
- ✓ PreparedStatement의 장점
 - DBMS가 PreparedStatement와 관련된 쿼리 파싱 회수 감소
 - 작은 따옴표 등 값에 포함된 특수 문자의 처리가 수월
 - 문자열 연결에 따른 코드의 복잡함 감소

3. SQL 실행

?의 값 바인딩 메소드	설명
setString(int index, String x)	지정한 인덱스의 파라미터 값을 문자열 x로 지정
setInt(int index, int x)	지정한 인덱스의 파라미터 값을 int 값 x로 지정
setLong(int index, long x)	지정한 인덱스의 파라미터 값을 long 값 x로 지정
setDouble(int index, double x)	지정한 인덱스의 파라미터 값을 double 값 x로 지정
setFloat(int index, float x)	지정한 인덱스의 파라미터 값을 float 값 x로 지정
setTimestamp(int index, Timestamp x)	지정한 인덱스의 값을 SQL TIMESTAMP 타입을 나타내는 java.sql.Timestamp 타입으로 지정
setDate(int index, Date x)	지정한 인덱스의 값을 SQL DATE 타입을 나타내는 java.sql.Date 타입으로 지정
setTime(int index, Time x)	지정한 인덱스의 값을 SQL TIME 타입을 나타내는 java.sql.Time 타입으로 지정

3. SQL 실행

❖ CallableStatement

- ✓ 프로시저를 실행할 수 있는 클래스
- ✓ 인스턴스를 생성할 때 `connection인스턴스.getCall(call 프로시저이름(매개변수));` 또는 `conn.prepareCall(call 프로시저이름(?))`의 형태로 생성
- ✓ `executeQuery` 메소드로 프로시저를 실행합니다.

4. ResultSet

- ❖ close()
- ❖ getXXX(컬럼 인덱스): XXX 타입으로 인덱스에 해당하는 데이터 가져오기
- ❖ getXXX(컬럼 명): XXX 타입으로 컬럼 명에 해당하는 데이터 가져오기
- ❖ next(): 다음 행으로 진행(없으면 false를 리턴)
- ❖ getColumnCount()
- ❖ absolute(int row)
- ❖ beforeFirst()
- ❖ afterLast()
- ❖ first()
- ❖ last()
- ❖ previous()
- ❖ ResultSetMetaData getMetaData()

4. ResultSet

❖ ResultSetMetaData

- ✓ 열의 개수: `int getColumnCount()`
- ✓ 열에 대한 정보: `String getColumnName(int column)`
- ✓ 열의 제목을 출력하는 데 적절한 제목: `String getColumnLabel(int column)`
- ✓ 열을 문자열로 출력하기 위한 최대 문자 수: `int getColumnDisplaySize(int column)`
- ✓ 열의 SQL Type을 돌려줌: `int getColumnType(int column)`
- ✓ 열의 SQL TypeName을 돌려줌: `String getColumnName(int column)`

4. ResultSet

- ❖ ResultSet의 getXXX() 메소드
 - ✓ 해당 데이터 타입으로 열 값을 읽어옴
 - ✓ 매개변수로 열의 이름(문자열)이나 인덱스(정수)를 줄 수 있음
 - ✓ DB의 데이터 타입에 해당하는 자바 데이터 타입으로 데이터를 읽어야 함.
 - ✓ 모든 데이터 타입에 대해 getString() 메소드로 읽을 수 있음
- ❖ ResultSet에서 모든 데이터를 다 읽어들이기 후에는 close()를 호출하여 자원 해제
- ❖ 1개 데이터 읽기

```
if(rs.next()){  
    데이터 읽기  
}
```

- ❖ 여러 개의 데이터 읽기

```
if(rs.next()){  
    do{  
        데이터 읽기  
    }while(rs.next());  
}
```

실습

❖ Oracle에 접속해서 작업

✓ 시퀀스 생성

```
CREATE SEQUENCE id_sequence  
INCREMENT BY 1  
START WITH 1  
NOCYCLE;
```

✓ 테이블 생성

```
CREATE TABLE SAMPLE(  
    NUM NUMBER PRIMARY KEY,  
    NAME VARCHAR2(20));
```

✓ 데이터 추가:

```
INSERT INTO SAMPLE VALUES(id_sequence.nextval, 'IRIN');  
INSERT INTO SAMPLE VALUES(id_sequence.nextval, 'YEZI');  
INSERT INTO SAMPLE VALUES(id_sequence.nextval, 'SUZI');  
INSERT INTO SAMPLE VALUES(id_sequence.nextval, 'JENNIE');
```

✓ 데이터 확인 : SELECT * FROM SAMPLE;

✓ 작업 완료 : COMMIT;

실습(SELECT)

```
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.ResultSet;  
import java.sql.Statement;
```

1:IRIN
2:YEZI
3:SUZI
4:JENNIE

```
public class SelectMain {  
  
    static final String sid = "xe";  
    static final String id = "scott";  
    static final String pass = "tiger";  
  
    public static void main(String[] args) {  
  
        Connection connection = null;  
        Statement stmt = null;  
        ResultSet rs = null;  
  
        String query = "SELECT * FROM SAMPLE";
```

실습(SELECT)

```
try {  
    // 접속할 데이터베이스의 URL을 생성  
    String url = "jdbc:oracle:thin:@localhost:1521:" + sid;  
    // 접속  
    connection = DriverManager.getConnection(url, id, pass);  
    // sql을 실행 시킬 수 있는 Statement 인스턴스 생성  
    stmt = connection.createStatement();  
    // select 구문을 실행시키고 그 결과를 rs에 저장  
    rs = stmt.executeQuery(query);  
    // 데이터가 있다면  
    if (rs.next()) {  
        do {  
            // 첫번째 컬럼의 값은 정수로 가져오  
            System.out.println(rs.getInt(1) + ":" +  
                rs.getString(2));  
        } while (rs.next());  
    }  
    // 읽어온 데이터가 없다면  
    else {  
        System.out.println("읽은 데이터가 없습니다.");  
    }  
}
```

실습(SELECT)

```
        rs.close();  
        stmt.close();  
        connection.close();  
    } catch (Exception e) {  
        System.out.println("에러:" + e.getMessage());  
    }  
}  
}
```



실습(INSERT)

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
```

```
public class InsertMain {
```

```
    static final String sid = "xe";
    static final String id = "scott";
    static final String pass = "tiger";
```

```
    public static void main(String[] args) {
```

```
        Connection connection = null;
        Statement stmt = null;
        try {
```

```
            // 접속할 데이터베이스의 URL을 생성
```

```
            String url = "jdbc:oracle:thin:@localhost:1521:" + sid;
```

```
            // 접속
```

```
            connection = DriverManager.getConnection(url, id, pass);
```

```
            // sql을 실행 시킬 수 있는 Statement 인스턴스 생성
```

```
            stmt = connection.createStatement();
```

```
            String query =
```

```
                "INSERT INTO SAMPLE VALUES
```

```
                (id_sequence.nextval, 'TZUYU')";
```

실습(INSERT)

```
int result = stmt.executeUpdate(query);
if(result != 0)
    System.out.println("삽입성공");
else
    System.out.println("삽입실패");
stmt.close();
connection.close();
} catch (Exception e) {
    System.out.println("에러:" + e.getMessage());
}
}
```

실습(PreparedStatement)

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;

public class PreparedStatementMain {

    static final String sid = "xe";
    static final String id = "scott";
    static final String pass = "tiger";

    public static void main(String[] args) {

        Connection connection = null;
        PreparedStatement stmt = null;
        try {
            // 접속할 데이터베이스의 URL을 생성
            String url = "jdbc:oracle:thin:@localhost:1521:" + sid;
            // 접속
            connection = DriverManager.getConnection(url, id, pass);

            // sql을 실행 시킬 수 있는 Statement 인스턴스 생성
            stmt = connection.prepareStatement("INSERT INTO
SAMPLE VALUES(id_sequence.nextval,?)");
            stmt.setString(1, "TIFFANY");
```

실습(PreparedStatement)

```
int result = stmt.executeUpdate();
if (result != 0)
    System.out.println("삽입성공");
else
    System.out.println("삽입실패");
stmt.close();
connection.close();

} catch (Exception e) {
    System.out.println("에러:" + e.getMessage());
}

}
```

```
}
```

날짜와 시간

- ❖ 날짜만 저장할 때는 `java.sql.Date` 타입으로 `setDate` 함수를 이용해서 바인딩 할 수 있고 가져올 때는 `getDate`로 가져올 수 있음
- ❖ 시간만 저장할 때도 방법은 동일한데 `java.sql.Time` 클래스를 이용
- ❖ 날짜와 시간을 저장할 때 `java.sql.Timestamp` 클래스를 이용할 수도 있음
- ❖ 웹 프로그래밍이나 스마트 폰 프로그래밍이 아니라면 날짜를 입력받을 수 있는 방법이 없기 때문에 날짜 데이터를 만들려면 문자열로 입력받은 후 날짜 데이터로 변경해서 작업
- ❖ 날짜 나 시간 데이터를 문자열로 저장하기도 함

BLOB (Binary Large Object)

- ❖ 파일의 내용을 저장하기 위해서 사용하는 데이터 타입
- ❖ 파일을 생성해두고 이름을 저장해서 사용해도 되지만 파일 이름을 저장하게 되면 파일을 이동하거나 파일의 이름을 마음대로 변경할 수 없음
- ❖ BLOB는 스트림을 이용해서 저장하고 읽어옴
- ❖ 데이터베이스에 저장하기 위해서 BLOB를 바인딩 하기 위해서는 `setBinaryStream(인덱스 , FileInputStream fis, int length)` 메소드를 이용
- ❖ 데이터베이스에 저장된 blob 데이터를 읽어오기 위해서는 `InputStream getBinaryStream(인덱스 또는 컬럼이름)` 메소드를 이용해서 스트림을 받아서 처리

BLOB (Binary Large Object)

❖ Oracle에서 실습 테이블 생성:

```
CREATE TABLE FILESAMPLE(  
    NUM NUMBER(10) PRIMARY KEY,  
    FILENAME VARCHAR2(50) NOT NULL,  
    FILECONTENT BLOB NOT NULL,  
    UPLOADDATE DATE);
```

실습(BLOB 저장)

```
import java.io.File;
import java.io.FileInputStream;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.Timestamp;
import java.util.Calendar;

import javax.swing.JFileChooser;
import javax.swing.filechooser.FileNameExtensionFilter;

public class FileUploadMain {
    public static void main(String[] args) {
        Connection con = null;
        PreparedStatement pstmt = null;
        int rownum = -1;
```

실습(BLOB 저장)

```
try {  
    con =  
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe", "scott", "tiger");  
    // 파일을 선택하기 위한 대화상자  
    JFileChooser fc = new JFileChooser();  
    FileNameExtensionFilter filter = new  
FileNameExtensionFilter("이미지", "jpg","jpeg","gif","png");  
    fc.setFileFilter(filter);  
    // 대화상자를 화면에 출력하고 누른 버튼의  
    // 값을 result에 저장  
    int result = fc.showOpenDialog(null);  
    // 누른 버튼이 열기버튼이면  
    if (result == JFileChooser.APPROVE_OPTION) {  
        // 선택한 파일을 f에 저장  
        File f = fc.getSelectedFile();  
        String filename = f.getName();  
        FileInputStream fis = new FileInputStream(f);  
        // 현재 날짜 및 시간을 Date 인스턴스로 만들기  
        Calendar cal = Calendar.getInstance();  
        Timestamp today = new  
Timestamp(cal.getTimeInMillis());
```

실습(BLOB 저장)

```
        pstmt = con.prepareStatement("INSERT INTO
FILESAMPLE VALUES(" + "id_sequence.nextval, ?,?,?)");
        pstmt.setString(1, filename);
        // blob 바인딩
        pstmt.setBinaryStream(2, fis, (int) f.length());
        // 날짜 및 시간 바인딩
        pstmt.setTimestamp(3, today);

        rownum = pstmt.executeUpdate();
        if (rownum > 0)
            System.out.println("삽입 성공");
        pstmt.close();
        con.close();
    }
} catch (Exception e) {
    System.out.println(e.getMessage());
    e.printStackTrace();
}
}
```

실습(BLOB 읽기)

```
import java.io.FileOutputStream;
import java.io.InputStream;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Timestamp;

public class FileReadMain {
    public static void main(String[] args) {
        Connection con = null;
        PreparedStatement pstmt = null;
        String sql = null;
        ResultSet rs = null;
```

실습(BLOB 읽기)

```
try {
    con =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe", "scott", "tiger");
    // test2 테이블의 모든 데이터를 읽어오는 sql
    sql = "SELECT * FROM FILESAMPLE";
    pstmt = con.prepareStatement(sql);
    rs = pstmt.executeQuery();
    if (rs.next()) {
        do {
            // 숫자, 문자열, 날짜 읽어오기
            int num = rs.getInt("NUM");
            String filename = rs.getString("FILENAME");
            Timestamp uploadDate =

rs.getTimestamp("UPLOADDATE");

uploadDate);

            // BLOB 가져오기
            InputStream is =

            // 가져온 데이터를 filename에 기록하기
            FileOutputStream fos = new
FileOutputStream("./" + filename);
```

실습(BLOB 읽기)

```
while (true) {  
    byte[] b = new byte[1024];  
    int len = is.read(b);  
    if (len <= 0)  
        break;  
    fos.write(b, 0, len);  
}  
fos.close();  
  
} while (rs.next());  
} else {  
    System.out.println("데이터가 없습니다.");  
}  
} catch (Exception e) {  
    System.out.println(e.getMessage());  
    e.printStackTrace();  
}  
}  
}
```


Procedure

- ❖ 데이터베이스에서 자주 사용하는 sql을 미리 작성해두고 호출하는 개체
- ❖ 프로시저를 이용하면 코딩이나 성능 및 효율 면에서 이득을 취할 수 도 있음
- ❖ Oracle에서 작성

```
DROP SEQUENCE id_sequence;
```

```
CREATE SEQUENCE id_sequence INCREMENT BY 1 START WITH 1 NOCYCLE;
```

```
CREATE TABLE PROCSAMPLE(  
  NUM NUMBER(10) PRIMARY KEY,  
  MESSAGE VARCHAR2(200),  
  WRITEDATE DATE);
```

```
CREATE OR REPLACE PROCEDURE MyProc  
(P_MESSAGE IN PROCSAMPLE.MESSAGE%TYPE)  
  IS  
  BEGIN  
    INSERT INTO PROCSAMPLE  
      VALUES(id_sequence.nextval,P_MESSAGE, SYSDATE);  
  END;
```

실습(Procedure 호출)

```
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.util.Scanner;

public class ProcedureMain {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.print("메시지를 입력:");
        String message = "";
        try {
            message = in.nextLine();
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
        Connection conn = null;
        String url = "jdbc:oracle:thin:@localhost:1521:xe";
        String id = "scott";
        String pass = "tiger";
        String query1 = "{call MyProc(?)}";
```

실습(Procedure 호출)

```
try {  
    conn = DriverManager.getConnection(url, id, pass);  
    CallableStatement call = conn.prepareCall(query1);  
    call.setString(1, message);  
    call.executeQuery();  
    call.close();  
    conn.close();  
    in.close();  
    System.out.println("프로시저 실행 성공");  
} catch (Exception e) {  
    System.out.println("실패:" + e.getMessage());  
}  
}
```

DTO & DAO 패턴

- ❖ DTO(Data Transfer Object) 패턴: 데이터베이스와의 연동에 필요한 데이터를 나타내는 클래스를 별도로 생성해서 사용하는 패턴 - Variable Object(VO) 또는 Domain Class 라고도 함
- ❖ 생성방법
 - ✓ 테이블에서 저장할 컬럼들을 인스턴스 변수로 선언
 - ✓ 필요에 따라 생성자 재정의
 - ✓ Getter & Setter 를 생성
 - ✓ 디버깅을 위해서 toString()을 재정의
 - ✓ 인스턴스 단위 전송이 필요한 경우 Serializable 인터페이스 implements
- ❖ DAO(Data Access Object) 패턴: 데이터베이스와의 연동에 필요한 코드를 묶어서 별도의 클래스로 생성해서 사용하는 패턴으로 Singleton 패턴과 템플릿 메소드 패턴과 함께 사용하는 것을 권장

DTO & DAO 패턴

```
Markers Properties Servers Data Source Explorer Snippets Console
<terminated> GoodMain [Java Application] C:\Program Files\Java\jdk1.7.0_25\bin\javaw.exe (2013. 8. 28. 오전 8:00:45)

데이터베이스 출력
Good [code=001 , name=apple, manufacture=korea, price=1500]
Good [code=002 , name=watermelon, manufacture=korea, price=15000]
Good [code=003 , name=oriental melon, manufacture=korea, price=1000]
Good [code=004 , name=banana, manufacture=Philippines, price=500]
Good [code=005 , name=lemon, manufacture=korea, price=1500]
Good [code=006 , name=mango, manufacture=taiwan, price=700]

XML 출력
<Good><code>001 </code><name>apple</name><manufacture>korea</manufacture><price>1500</price></Good>
<Good><code>002 </code><name>watermelon</name><manufacture>korea</manufacture><price>15000</price></Good>
<Good><code>003 </code><name>oriental melon</name><manufacture>korea</manufacture><price>1000</price></Good>
<Good><code>004 </code><name>banana</name><manufacture>Philippines</manufacture><price>500</price></Good>
<Good><code>005 </code><name>lemon</name><manufacture>korea</manufacture><price>1500</price></Good>
<Good><code>006 </code><name>mango</name><manufacture>taiwan</manufacture><price>700</price></Good>
```

DTO & DAO 패턴

❖ 데이터베이스 테이블 작성

```
create table Goods (  
    code    char(5) not null,  
    name    varchar2(50) not null,  
    manufacture varchar2(20),  
    price   number(10) not null,  
    primary key(code)  
);
```

❖ 샘플 데이터 입력

```
insert into Goods values('001', 'apple', 'korea', 1500);  
insert into Goods values('002', 'watermelon', 'korea', 15000);  
insert into Goods values('003', 'oriental melon', 'korea', 1000);  
insert into Goods values('004', 'banana', 'philippines', 500);  
insert into Goods values('005', 'lemon', 'korea', 1500);  
insert into Goods values('006', 'mango', 'taiwan', 700);
```

```
commit;
```


```
select * from Goods;
```



DTO & DAO 패턴


- ❖ Goods 테이블의 데이터를 저장하기 위한 DTO 클래스인 Good 클래스를 생성

```
public class Good{  
    private String code;  
    private String name;  
    private String manufacture ;  
    private int price;  
  
    public Good() {  
        super();  
    }  
    public Good(String code, String name, String manufacture, int price) {  
        super();  
        this.code = code;  
        this.name = name;  
        this.manufacture = manufacture;  
        this.price = price;  
    }  
}
```



DTO & DAO 패턴

```
public String getCode() {  
    return code;  
}  
public void setCode(String code) {  
    this.code = code;  
}  
public String getName() {  
    return name;  
}  
public void setName(String name) {  
    this.name = name;  
}  
public String getManufacture() {  
    return manufacture;  
}  
public void setManufacture(String manufacture) {  
    this.manufacture = manufacture;  
}
```



DTO & DAO 패턴

```
public int getPrice() {  
    return price;  
}  
public void setPrice(int price) {  
    this.price = price;  
}  
@Override  
public String toString() {  
    return "Good [code=" + code + ", name=" + name + ", manufacture=" +  
        + manufacture + ", price=" + price + " ]";  
}  
}
```



DTO & DAO 패턴

❖ DAO 클래스의 메소드의 원형을 가진 GoodDAO 인터페이스를 생성

```
import java.util.List;
```

```
public interface GoodDAO {  
    //테이블의 모든 데이터를 가져오는 메소드  
    public List<Good> allSelect();  
  
    //기본키를 가지고 하나의 데이터를 가져오는 메소드  
    public Good getGood(String code);  
  
    //하나의 데이터를 삽입하는 메소드  
    public Boolean insertGood(Good good);  
  
    //하나의 데이터를 수정하는 메소드  
    public Boolean updateGood(Good good);  
  
    //하나의 데이터를 삭제하는 메소드  
    public Boolean deleteGood(String code);  
}
```



DTO & DAO 패턴

❖ DAO 메소드를 구현할 GoodDAOImpl 클래스를 생성

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.ArrayList;
import java.util.List;
```

```
public class GoodDAOImpl implements GoodDAO {
    // 싱글톤 클래스를 만드는 코드
    private GoodDAOImpl() {}

    private static GoodDAOImpl obj;

    public static GoodDAOImpl getInstance() {
        if (obj == null)
            obj = new GoodDAOImpl();
        return obj;
    }
}
```



DTO & DAO 패턴

- ❖ DAO 메소드를 구현할 GoodDAOImpl 클래스를 생성

// 데이터베이스 연동 메소드에서 사용할 변수 선언

```
private Connection con;
```

```
private PreparedStatement pstmt;
```

// 데이터베이스 연결 메소드

```
private boolean connect() {
```

```
    boolean result = false;
```

```
    try {
```

```
        con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe",
```

```
        "scott", "tiger");
```

```
        result = true;
```

```
    } catch (Exception e) {
```

```
        e.printStackTrace();
```

```
    }
```

```
    return result;
```

```
}
```



DTO & DAO 패턴

- ❖ DAO 메소드를 구현할 GoodDAOImpl 클래스를 생성

//con과 pstmt 연결 해제하는 메소드

```
private void close() {  
    try {  
        if (pstmt != null)  
            pstmt.close();  
        if (con != null)  
            con.close();  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```



DTO & DAO 패턴

- ❖ DAO 메소드를 구현할 GoodDAOImpl 클래스를 생성

// Goods 테이블의 모든 데이터를 읽어서 리턴하는 메소드

```
public List<Good> allSelect() {
```

```
    // 데이터를 저장해서 리턴할 인스턴스 생성
```

```
    List<Good> list = new ArrayList<Good>();
```

```
    // select 구문의 결과를 저장할 변수 생성
```

```
    ResultSet rs = null;
```

```
    try {
```

```
        // 연결에 성공하면
```

```
        if (connect()) {
```

```
            // SQL을 실행할 수 있는 인스턴스 생성
```

```
            pstmt = con.prepareStatement("select * from goods");
```

```
            // select 구문의 결과를 rs에 저장
```

```
            rs = pstmt.executeQuery();
```



DTO & DAO 패턴

- ❖ DAO 메소드를 구현할 GoodDAOImpl 클래스를 생성

// 검색된 데이터를 읽어서 list에 저장

```
if (rs.next()) {  
    do {
```

```
        Good good = new Good();
```

```
        good.setCode(rs.getString("code"));
```

```
        good.setName(rs.getString("name"));
```

```
        good.setManufacture(rs.getString("manufacture"));
```

```
        good.setPrice(rs.getInt("price"));
```

```
        list.add(good);
```

```
    } while (rs.next());
```

```
}
```

```
}
```

```
} catch (Exception e) {
```

```
    e.printStackTrace();
```

```
} finally {
```

```
    try {
```

```
        if (rs != null) rs.close();
```

```
        close();
```

```
    } catch (Exception e) {}
```

```
}
```

```
return list;
```

```
}
```

DTO & DAO 패턴

❖ DAO 메소드를 구현할 GoodDAOImpl 클래스를 생성

// code(Primary Key)를 받아서 goods 테이블에서 데이터를 검색한 후 리턴하는 메소드
// 데이터는 없거나 1개만 나올 수 있습니다.

// 없을 때는 null을 리턴하고 있을 때는 1개를 리턴

// 목록에서 제목을 눌렀을 때 상세보기를 하거나 회원정보 수정을 눌렀을 때 회원정보 보여주는 화면을 만들 때 사용

```
public Good getGood(String code) {
```

```
    // 검색된 정보를 저장해서 리턴하기 위한 변수
```

```
    Good good = null;
```

```
    // 검색된 데이터 정보를 저장할 변수
```

```
    ResultSet rs = null;
```

```
    try{
```

```
        //데이터베이스 연결
```

```
        if(connect()){
```

```
            //SQL 실행 인스턴스 생성
```

```
            pstmt = con.prepareStatement(
```

```
                "select * from goods where trim(code) = ?");
```

```
            //물음표에 데이터 바인딩
```

```
            pstmt.setString(1, code);
```

```
            //select 구문을 실행해서 결과를 rs에 저장
```

```
            rs = pstmt.executeQuery();
```


DTO & DAO 패턴

- ❖ DAO 메소드를 구현할 GoodDAOImpl 클래스를 생성

//검색된 데이터는 1개가 넘지 않습니다.

```
if(rs.next()){  
    good = new Good();  
    good.setCode(rs.getString("code").trim());  
    good.setName(rs.getString("name"));  
    good.setManufacture(rs.getString("manufacture"));  
    good.setPrice(rs.getInt("price"));  
}
```

```
}
```

```
}
```

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

```
}
```

```
finally{
```

```
    try{
```

```
        if(rs != null)
```

```
            rs.close();
```

```
        close();
```

```
    }catch(Exception e){
```

```
        e.printStackTrace();
```

```
    }
```

```
}
```

```
return good;
```

```
}
```

DTO & DAO 패턴

- ❖ DAO 메소드를 구현할 GoodDAOImpl 클래스를 생성

//하나의 데이터를 삽입하는 메소드

```
public Boolean insertGood(Good good){
    Boolean result = false;
    try{
        if(connect()){
            pstmt = con.prepareStatement(
                "insert into goods(" +
                "code, name, manufacture,price)" +
                " values(?,?,?,?)");
            pstmt.setString(1, good.getCode());
            pstmt.setString(2, good.getName());
            pstmt.setString(3, good.getManufacture());
            pstmt.setInt(4, good.getPrice());
            int rownum = pstmt.executeUpdate();
            if(rownum > 0)
                result = true;
        }
    }catch(Exception e){
        e.printStackTrace();
    }finally{
        close();
    }
    return result;
}
```

DTO & DAO 패턴

- ❖ DAO 메소드를 구현할 GoodDAOImpl 클래스를 생성

```
public Boolean updateGood(Good good) {  
    Boolean result = false;  
    try{  
        if(connect()){  
            pstmt = con.prepareStatement( "update goods " +  
            "set name = ?, manufacture = ?, price = ? "+  
            "where trim(code) = ?");  
  
            pstmt.setString(1, good.getName());  
            pstmt.setString(2, good.getManufacture());  
            pstmt.setInt(3, good.getPrice());  
            pstmt.setString(4, good.getCode());  
            int rownum = pstmt.executeUpdate();  
            if(rownum > 0)  
                result = true;  
        }  
    }catch(Exception e){  
        e.printStackTrace();  
    }finally{  
        close();  
    }  
    return result;  
}
```

DTO & DAO 패턴

- ❖ DAO 메소드를 구현할 GoodDAOImpl 클래스를 생성

@Override

```
public Boolean deleteGood(String code) {  
    Boolean result = false;  
    try{  
        if(connect()){  
            pstmt = con.prepareStatement(  
                "delete from goods " +  
                "where trim(code) = ?");  
  
            pstmt.setString(1, code);  
  
            int rownum = pstmt.executeUpdate();  
            if(rownum > 0)  
                result = true;  
        }  
    }catch(Exception e){  
        e.printStackTrace();  
    }finally{  
        close();  
    }  
    return result;  
}  
}
```

DTO & DAO 패턴

❖ Main 클래스를 생성


```
import java.util.List;
import java.util.Scanner;
import javax.swing.JOptionPane;

public class GoodMain {

    public static void main(String[] args) {
        //키보드 입력 객체 만들기
        Scanner sc = new Scanner(System.in);

        //각 열의 값을 저장할 변수
        String code = null;
        String name = null;
        String manufacture = null;
        String imsi = null;
        int price = 0;

        //데이터베이스 작업 결과를 저장할 변수
        Boolean result = null;
        Good good = null;
        List<Good> list = null;
```



DTO & DAO 패턴

❖ Main 클래스를 생성

```
//무한 루프
MainLoop : while(true) {
    //메뉴 출력과 입력 받기
    System.out.print("1.데이터 전체 가져오기 2.데이터 1개 가져오기 3.데이터 추가 4.
데이터 수정 5.데이터 삭제 6.프로그램 종료:");
    String menu = sc.nextLine();
    //데이터베이스 연동 인스턴스 만들기
    GoodDAO goodDao = GoodDAOImpl.getInstance();
    switch(menu) {
        case "1":
            list = goodDao.allSelect();
            for(Good temp : list) {
                System.out.println(temp);
            }
            break;
```

DTO & DAO 패턴

❖ Main 클래스를 생성

case "2":

```
System.out.print("조회할 데이터의 코드를 입력하세요:");  
code = sc.nextLine();  
good = goodDao.getGood(code);  
if(good == null) {  
    System.out.println("존재하지 않는 코드입니다.");  
}else {  
    System.out.println(good);  
}  
break;
```



DTO & DAO 패턴

❖ Main 클래스를 생성

case "3":

```
System.out.print("삽입할 데이터의 코드를 입력하세요:");
code = sc.nextLine();
good = goodDao.getGood(code);
if(good == null) {
```

```
    System.out.println("사용 가능한 코드입니다.");
```

```
    System.out.print("이름을 입력하세요:");
```

```
    name = sc.nextLine();
```

```
    System.out.print("원산지를 입력하세요:");
```

```
    manufacture = sc.nextLine();
```

```
    System.out.print("가격을 입력하세요:");
```

```
    imsi = sc.nextLine();
```

```
    try {
```

```
        price = Integer.parseInt(imsi);
```

```
        good = new Good(code, name, manufacture,
```

```
price);
```

```
        result = goodDao.insertGood(good);
```

```
        if(result == true) {
```

```
            System.out.println("데이터 삽입에 성
```

```
공하셨습니다.");
```

```
        }else {
```

```
            System.out.println("데이터 삽입에 실
```

```
패하셨습니다.");
```

```
        }
```

```
    }
```


DTO & DAO 패턴

❖ Main 클래스를 생성

다.");

```
        catch(Exception e) {  
            System.out.println("잘못된 가격을 입력하셨습니다.");  
        }  
    }else {  
        System.out.println("사용 불가능한 코드입니다.");  
    }  
    break;
```



DTO & DAO 패턴

❖ Main 클래스를 생성

case "4":

```
System.out.print("수정 할 데이터의 코드를 입력하세요:");  
code = sc.nextLine();  
good = goodDao.getGood(code);  
if(good != null) {
```

```
    System.out.println("수정 가능한 코드입니다.");
```

```
    System.out.print("이름을 입력하세요:");
```

```
    name = sc.nextLine();
```

```
    System.out.print("원산지를 입력하세요:");
```

```
    manufacture = sc.nextLine();
```

```
    System.out.print("가격을 입력하세요:");
```

```
    imsi = sc.nextLine();
```

```
    try {
```

```
        price = Integer.parseInt(imsi);
```

```
        good = new Good(code, name, manufacture,
```

```
price);
```

```
        result = goodDao.updateGood(good);
```

```
        if(result == true) {
```

```
            System.out.println("데이터 수정에 성
```

```
공하셨습니다.");
```

```
        }else {
```

```
            System.out.println("데이터 수정에 실
```

```
패하셨습니다.");
```

```
        }
```

```
    }
```

DTO & DAO 패턴

❖ Main 클래스를 생성

다.");

니다.");

```
        catch(Exception e) {  
            System.out.println("잘못된 가격을 입력하셨습니다.");  
        }  
    }else {  
        System.out.println("존재하지 않는 코드라서 수정할 수 없습니다.");  
    }  
    break;
```



DTO & DAO 패턴

❖ Main 클래스를 생성

case "5":

```
System.out.print("삭제할 데이터의 코드를 입력하세요:");
code = sc.nextLine();
good = goodDao.getGood(code);
if(good != null) {
    System.out.println("삭제 가능한 코드입니다.");
    int r = JOptionPane.showConfirmDialog(null, "정말로 삭제
    하시겠습니까?");

    if(r == JOptionPane.OK_OPTION) {
        result = goodDao.deleteGood(code);
        if(result == true) {
            System.out.println("삭제에 성공하셧
            습니다..");
        }else {
            System.out.println("삭제에 실패하셧
            습니다..");
        }
    }
} else {
    System.out.println("존재하지 않는 코드라서 삭제할 수 없습
    니다.");
}
break;
```

DTO & DAO 패턴

❖ Main 클래스를 생성

```
        case "6":
            break MainLoop;
        default :
            System.out.println("메뉴를 잘못 선택하셨습니다.");
            break;
    }
    System.out.println("엔터를 누르면 메인 메뉴로 이동합니다.");
    sc.nextLine();
}
System.out.println("프로그램을 종료합니다!!!!");
sc.close();
}
```

