Java Web Programming 입문

(Servlet/JSP)

28일차

오늘의 키워드

- 데이터베이스 (Database)
- JDBC (Java Database Connectivity)
- 오라클 (Oracle) 설치
  - JDBC Driver 설치
- Database Programming
  - Connection
  - Factory
  - Pooling

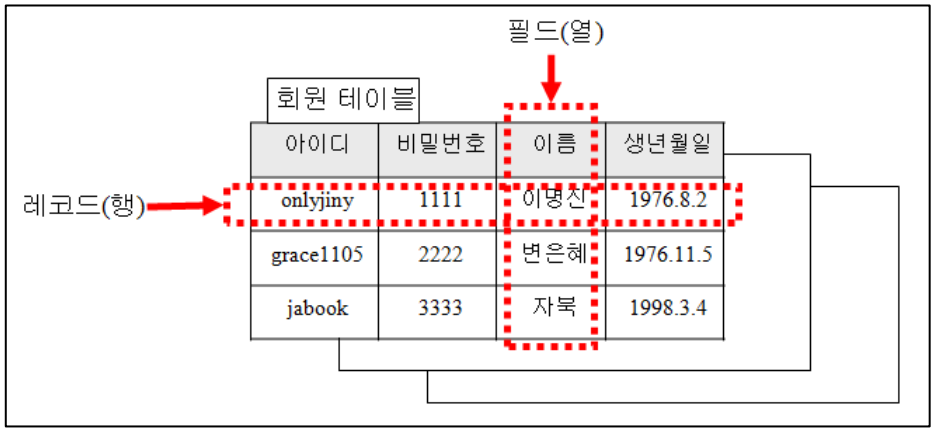
데이터베이스 (Database)

- 데이터베이스 (Database)
  - 하나 이상의 자료가 서로 연관되어 저장된 Data 집합
  - 목적
    - Data를 구조적으로 저장함으로써
    - 자료의 저장과 검색 및 갱신의 효율을 높이는 것

- DBMS (Database Management System)
  - Database를 효율적으로 관리하기 위한 System
    - Data 저장, 추출 등의 작업을 쉽고 빠르게 할 수 있도록
    - 도와주는 Software
  - Data의 저장, 검색
  - 다중 사용자를 위한 동시 실행 제어
  - 이전 상태로의 복귀, 보안 등등의 기능을 가짐
  - 대표적인 DBMS
    - Oracle (Oracle), MySQL, MS-SQL(MS), DB2(IBM), etc…

데이터베이스 (Database)

- 데이터베이스 (Database)의 종류
  - 계층형 데이터베이스 (Hierarchical Databse, HDB)
  - 객체지향 데이터베이스 (Object-Oriented Database, OODB)
  - 관계형 데이터베이스 (Relational Database, RDB)
    - Data를 행(Row)과 열(Column)의 관계로 표현
    - 표와 같은 형태



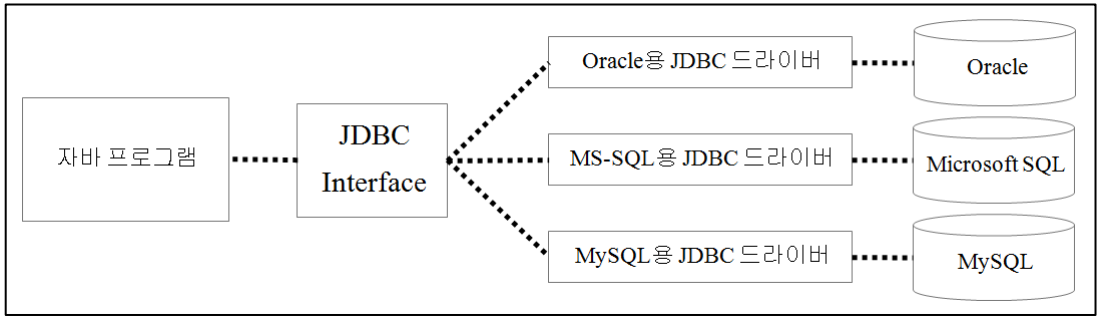
JDBC (Java Database Connectivity)

- JDBC (Java Database Connectivity)
  - Java Program과 Database를 연결시키는 통로
  - Java를 이용해서 Database Programming을 하기 위한
  - Java 표준 Database Interface

  - 왜?
    - Database 회사들이 많으니까…
    - 회사마다 각각의 Database를 만드니까…
    - 즉, Java를 이용하여 Database를 사용하는 방법도 다 다름
    - 따라서 Java쪽에 표준이 되는 Interface를 만들어 제공

JDBC (Java Database Connectivity)

- 역할



  - Java 응용 프로그램은 JDBC Interface를 통해 Database에 접근
    - Java 프로그램 내에서 Database 질의문(SQL)을 실행하기 위한 Java API를 제공
  - JDBC Interface는
    - SQL (Structured Query Language)을 DBMS에 전달
    - 결과를 받아서 Java 응용 프로그램으로 돌려줌

JDBC (Java Database Connectivity)

- JDBC Driver
  - Database 회사들은…
    - JDBC Interface를 제공받은 후,
    - 자신들의 Database에 맞는 기능을 구현
  - Programmer 들은…
    - 내부를 알지 못하더라도 Up-Casting 원리에 따라
    - JDBC 표준 Interface만 알면
    - Database를 조작할 수 있는 Program 작성이 가능
  - 즉, JDBC Driver는…
    - Database 회사들이 JDBC Interface를 상속받아
    - 각 Database에 맞게 기능을 구현해놓은 Class의 집합

JDBC (Java Database Connectivity)

- 특징
  - 관계형 데이터베이스 접근을 위한 표준 Library
  - 표준 Interface를 통해 여러 Database에 동시에 접근 가능
  - Java 언어의 사용으로 Platform 및 Database에 독립적
  - Database에 SQL문을 질의하여
    - 그 결과를 응용프로그램으로 돌려주는 API

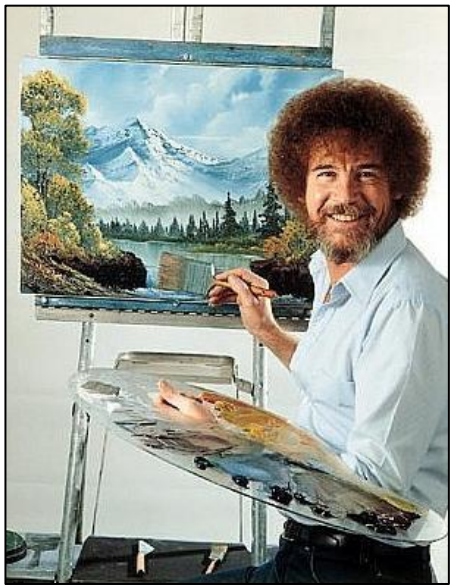
- SQL (Structured Query Language)
  - Database에 연결한 후,
    - Database로부터 Data를 삽입, 수정, 삭제, 검색하기 위한
      - 표준화된 명령문
    - SQL 구문을 실행하면
      - DBMS가 해당 명령문에 대한 처리 결과를 돌려줌
      - 그래서 질의어(Query Language)라고 부른다.
    - Database용 표준 질의언어

오라클 (Oracle) 설치

- 오라클 (Oracle) 설치

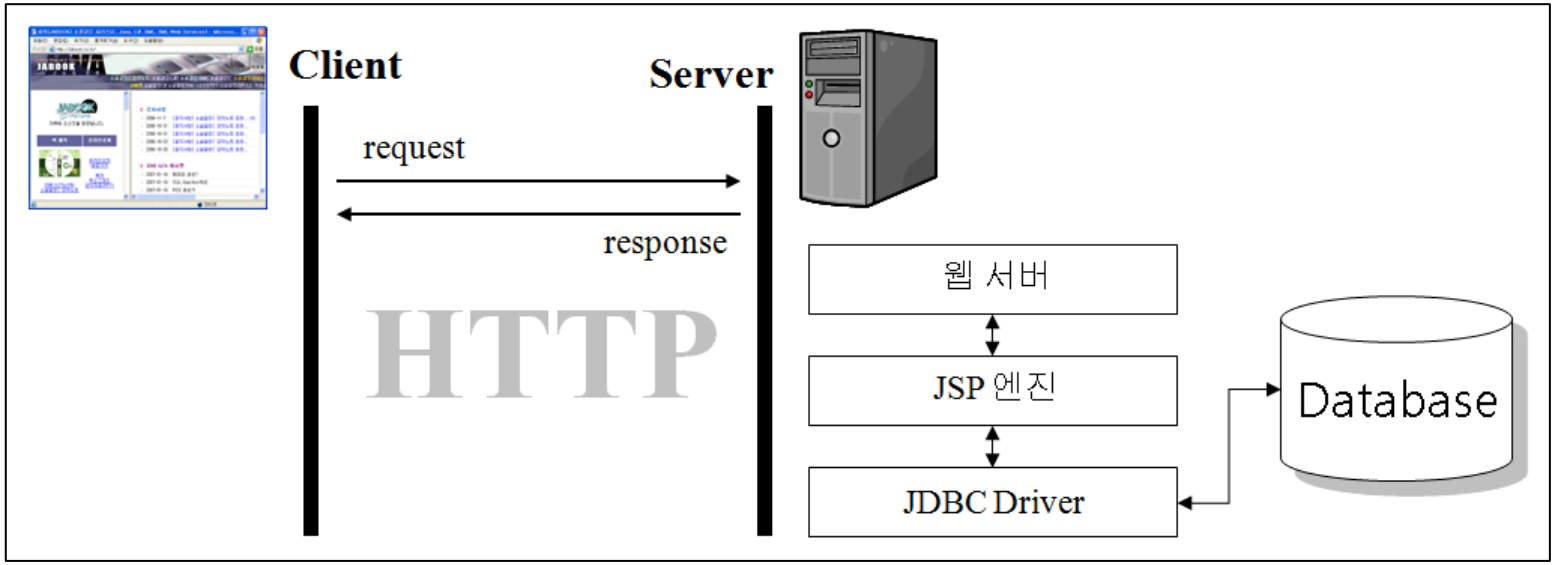
오라클 (Oracle) 설치

- JDBC Driver 설치
  - Oracle 설치경로/jdbc/lib/ojdbc6.jar
    - java 1.6에 최적화된 버전
    - java 1.5라면 ojdbc5.jar
    - 보통 ojdbc4.jar도 많이 씀
  - Tomcat 설치경로/lib/ 복사
  - Java 설치경로/lib/ext/ 복사
  - Eclipse 내 Project의 Library 폴더에 복사



Database Programming

- JDBC와 JSP의 연동 방식



Database Programming

- Database Programming 절차
  - java.sql Package Import
  - JDBC Driver Loading
  - 연결 URL 정의 및 DB Connection 생성
  - Connection Object로부터 Statement 생성
  - SQL 실행
  - SQL 반환 결과 처리
  - 연결 닫음

Database Programming

- Database Programming 절차
  - java.sql Package Import
    - JDBC를 이용하기 위해 java.sql Package를 import

```
<%@page import="java.sql.*"%>
```

  - JDBC Driver Loading
    - Oracle을 위한 JDBC Driver를
      - Class.forName() Method를 이용하여 Loading

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

  - 연결 URL 정의 및 DB Connection 생성
    - JDBC Driver를 관리하는 DriverManager로부터 Connection Obejct를 생성
    - getConnection() Method를 사용
      - Parameter
        - Database Server 접속 URL
        - Database에 접근이 허용된 계정, 비밀번호

```
String      url      = "jdbc:oracle:thin:@localhost:1521:ORCL";
String      id       = "scott";
String      password = "tiger";
Connection  conn     = DriverManager.getConnection(url, id, password);
```

Database Programming

- Database Programming 절차
  - Connection Object로부터 Statement 생성
    - Connection Object로부터 Statement Object 생성
    - Statement Object
      - Database에 질의
      - 작업 명령을 전단하는데 사용

```
Statement stmt = conn.createStatement();
```

  - SQL 실행
    - 생성된 Statement Object를 통해 SQL 구문 실행
    - Database로 전송
    - executeQuery() Method의 결과는 ResultSet Object
      - 질의의 결과를 처리

```
String sql = "select empno, ename, sal from emp";

ResultSet rs = stmt.executeQuery(sql);
```

Database Programming

- Database Programming 절차
  - SQL 반환 결과 처리
    - 질의의 결과를 가지고 있는 ResultSet Object를 이용하여
      - HTML Page로 출력

```
while(rs.next()) {
    out.print( rs.getInt(1)    + ", " +
               rs.getString(2) + ", " +
               rs.getDouble(3) + "<br>" );
}
```

  - 연결 닫음
    - Connection Interface의 close() Method를 사용하여
      - Database로부터 가져온
      - ResultSet, Statement, Connection Object 들을 순서대로 닫아줌

```
rs.close();
stmt.close();
conn.close();
```

Database Programming

- simpleJDBC.jsp

```
<%@page contentType="text/html; charset=euc-kr"%>
<%@page import="java.sql.*"%>
<html><head><title>Database Programming</title></head>
    <body><h3>간단한 데이터베이스 프로그래밍</h3>
        <%  Class.forName("oracle.jdbc.driver.OracleDriver");
            out.print("JDBC Driver 로딩 완료<br>");
            String     url      = "jdbc:oracle:thin:@localhost:1521:ORCL";
            String     id       = "scott";
            String     password = "tiger";
            Connection conn     = DriverManager.getConnection(url, id, password);
            out.print("연결 URL 정의 및 DB와 Connection 생성 완료<br>");
            Statement stmt = conn.createStatement();
            String sql = "CREATE TABLE TestEmp(                    " +
                         "  empno NUMBER(10) NOT NULL PRIMARY KEY, " +
                         "  ename VARCHAR2(20),                    " +
                         "  sal   NUMBER(20,2)                     ";
            stmt.executeUpdate(sql);
            out.print("TestEmp 테이블 생성 완료<br>");
            stmt = conn.createStatement();
            sql = "Insert into TestEmp select empno, ename, sal from emp";
            stmt.executeUpdate(sql);
            out.print("데이터 삽입 완료<br>");
            stmt = conn.createStatement();
            sql = "select empno, ename, sal from TestEmp";
            ResultSet rs = stmt.executeQuery(sql);
            out.print("데이터 쿼리 완료<br>");
            while(rs.next()) {
                out.print( rs.getInt(1)    + ", " + rs.getString(2) + ", " +
                           rs.getDouble(3) + "<br>" );
            }
            rs.close();
            stmt.close();
            conn.close();
            out.print("연결 종료<br>");     %>
</body></html>
```

Database Programming

- Connection Interface
  - Database에 접근하기 위한 Object를 생성하는 Interface
  - Connection Interface를 통해 한번 연결이 이루어지면
    - 모든 작업들은 이 연결을 이용하여 수행
    - 작업 후 반드시 close() Method로 연결된 Connection을 닫아주어야함
      - 해주지 않으면 Server의 Resource를 계속 차지하고 있음

  - Connection Object의 생성 방법
    - Database에 연결할때는
      - 먼저 Class 클래스의 forName() Method를 이용하여 해당 Database의 Driver를 Loading
      - JDBC Driver Loading
        - Oracle

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

        - MS-SQL

```
Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");
```

        - MySQL

```
Class.forName("com.mysql.jdbc.Driver");
```

Database Programming

- Connection Interface
  - java.sql.DriverManager Class의 getConnection() Method를 사용
    - Java Virtual Machine에 Loading 되어진 Driver들 중
      - 매개변수로 주어진 Database URL을 인식하는 Driver를 검색
    - Driver를 찾아내면 DriverManager Class는 그 Driver로부터
      - Database에 대한 접속을 얻어내는 Connection Object를 생성

```
Connection conn = DriverManager.getConnection(url);
```

    - 순수 Java Code라면 Exception 처리를 반드시 해주어야 한다
      - ClassNotFoundExcpetion
        - Driver Loading과 관련된 Exception
      - SQLException
        - Database 접속에 대한 Excpetion
      - JSP에서는 모든 예외를 자동으로 처리
        - 그래도 Java Coding 시에는 반드시 필요함을 기억해두자

    - 다 사용한 Connection Object는 연결을 종료시켜 준다

```
conn.close();
```

Database Programming

- Connection Interface
  - DBConnection.jsp

```
<%@page contentType="text/html; charset=euc-kr"%>
<%@page import="java.sql.*"%>
<html>
    <head>
        <title>Database Programming</title>
    </head>
    <body>
        <h3>데이터베이스 연결</h3>
        <%
            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection conn = DriverManager.getConnection(
                                                "jdbc:oracle:thin:@localhost:1521:ORCL",
                                                "scott",
                                                "tiger"                                    );
            out.print("<p>데이터베이스 연결 성공</p>");
            DatabaseMetaData dm = conn.getMetaData();
            out.print("<b>JDBC 이름</b>: " + dm.getDriverName() + "<br>");
            out.print("<b>JDBC 버전</b>: " + dm.getDriverVersion() + "<br>");
            out.print("<b>DBMS URL</b>: " + dm.getURL() + "<br>");
            out.print("<b>사용자 이름</b>: " + dm.getUserName() + "<br>");
            conn.close();
            out.print("<p>연결 종료</p>");
        %>
    </body>
</html>
```

Database Programming

- Connection Interface
  - Connection Object 생성

```
String     url      = "jdbc:oracle:thin:@localhost:1521:ORCL";
String     id       = "scott";
String     password = "tiger";
Connection conn     = DriverManager.getConnection(url, id, password);
```

```
String url = "jdbc:mysql://localhost:3306/Test?user=root&password=1053";
Connection conn = DriverManager.getConnection(url);
```

  - DatabaseMetaData Object
    - Database의 연결과 관련된 정보를 담고 있는 Object
    - Connection Object로부터 얻을 수 있음

```
DatabaseMetaData dm = conn.getMetaData();
out.print("<b>JDBC 이름</b>: " + dm.getDriverName() + "<br>");
out.print("<b>JDBC 버전</b>: " + dm.getDriverVersion() + "<br>");
out.print("<b>DBMS URL</b>: " + dm.getURL() + "<br>");
out.print("<b>사용자 이름</b>: " + dm.getUserName() + "<br>");
```

Database Programming

- Statement Interface
  - SQL문을 실행, 그에 대한 결과값을 가져오기 위해 사용
  - Statement Object 생성
    - Connection Interface의 createStatement() Method 사용

```
Statement stmt = conn.createStatement();
```

    - Statetment에서 Query를 실행할 때 사용하는 Method
      - executeQuery()
        - 결과 값(ResultSet) 있음
        - 결과값이 존재하는 SELECT 문을 실행할 때 사용

```
ResultSet rs = stmt.executeQuery(String sql);
```

      - executeUpdate()
        - 결과 값(ResultSet) 없음
        - CREATE, INSERT, UPDATE, DELETE 등의 결과값이 없는 SQL문 실행 시 사용

```
stmt.executeUpdate(String sql);
```

  - Statement Object 종료
    - 다 사용한 Statement Object는 다음과 같이 종료

```
stmt.close();
```

Database Programming

- Statement Interface
  - UseStatement.jsp

```jsp
<%@page contentType="text/html; charset=euc-kr"%>
<%@page import="java.sql.*"%>
<html>
    <head>
        <title>Database Programming</title>
    </head>
    <body>
        <h3>Statement 질의문 실행</h3>
        <%
            Class.forName("oracle.jdbc.driver.OracleDriver");
            String     url      = "jdbc:oracle:thin:@localhost:1521:ORCL";
            String     id       = "scott";
            String     password = "tiger";
            Connection conn     = DriverManager.getConnection(url, id, password);

            Statement  stmt     = conn.createStatement();
            String     sql      = "Insert into TestEmp value (9999, TestEname, 3000)";
            stmt.executeUpdate(sql);
            out.print("executeUpdate() 메소드<br>");

            stmt = conn.createStatement();
            sql  = "select * from TestEmp";
            ResultSet rs = stmt.executeQuery(sql);
            out.print("executeQuery() 메소드<br>");
            stmt.close();
            conn.close();
        %>
    </body>
</html>
```

Database Programming

- ResultSet Interface
  - exceuteQuery() Method의 반환 결과가 저장된 Interface
  - 조회 결과를 가리키는 Cursor를 관리
  - 결과값을 받을 때는 Data Type을 맞춰주어야 한다
    - ResultSet이 가지고 있는 Data들은 SQL Data Type을 가짐
    - Java에서 사용하는 Data Type으로 변형되어야 함
  - 사용
    - Query 작성 후 executeQuery() Method로 받아온 결과를
      - ResultSet Object로 얻어냄

```
ResultSet rs = stmt.executeQuery("select empno, ename, sal from TestEmp");
```

    - Result에 저장된 Data 값 출력을 위해 while문 사용
      - Data Type에 맞는 get~ Method를 이용하여 값을 출력
        - Parameter은 필드의 번호, 이름을 사용

```
while(rs.next()) {
    out.print( rs.getInt(1)    + ", " + rs.getString(2) + ", " + rs.getDouble(3));
}
```

    - 사용했던 ResultSet Object 종료

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

Database Programming

- ResultSet Interface
  - PrintResultSet.jsp

```
<%@page contentType="text/html; charset=euc-kr"%>
<%@page import="java.sql.*"%>
<html><head><title>Database Programming</title></head>
    <body>
        <h3>ResultSet 출력</h3>
        <%  Class.forName("oracle.jdbc.driver.OracleDriver");
            String     url      = "jdbc:oracle:thin:@localhost:1521:ORCL";
            String     id       = "scott";
            String     password = "tiger";
            Connection conn     = DriverManager.getConnection(url, id, password);

            Statement stmt = conn.createStatement();
            ccccccccsql  = "select * from TestEmp";
            ResultSet rs = stmt.executeQuery(sql); %>
        <table border="1" cellspacing="0" cellpadding="3">
        <tr><td align="center" bgcolor="#EEEEFF">empno</td>
            <td align="center" bgcolor="#EEEEFF">ename</td>
            <td align="center" bgcolor="#EEEEFF">sal</td></tr>
        <%  while(rs.next()) {
                out.print("<tr>");
                out.print("<td align=\"center\">" + rs.getInt(1) + "</td>");
                out.print("<td align=\"center\">" + rs.getString(2) + "</td>");
                out.print("<td align=\"center\">" + rs.getDouble(3) + "</td>");
                out.print("</tr>");
            } %>
        </table>
        <% rs.close();      stmt.close();   conn.close(); %>
</body></html>
```

Database Programming

- PreparedStatement Interface
  - 반복되는 SQL문을 수행해야 할 때 사용
  - SQL 문을 사전 Compile 하므로 수행 속도가 빠름
  - 위치 표시자를 사용하여 값을 연속적으로 바꿔줄 수 있음
    - 동일한 형식의 SQL 구문을 여러 번 반복 수행 시 편리
  - 작동 원리
    - SQL 문을 먼저 Compile하여 PreparedStatement에 저장
      - 미완성 상태
    - 이후에 SQL 문을 완성할 값을 받기 위해 물음표(?)를 사용
      - 위치 표시자
    - 위치 번호와 위치표시자에 들어갈 값을 Parameter로 가짐
    - SQL 문 내에 모든 위치표시자에 값을 할당
      - set~ Method 를 사용
    - executeUpdate() Method를 사용하여 Query 수행

Database Programming

- PreparedStatement Interface
  - PreparedStatement Object 사용
    - 생성
      - 물음표가 앞으로 넣어줄 값의 자리

```
String sql = "Insert into TestEmp (empno, ename, sal) value (?, ?, ?)";
PreparedStatement pstmt = conn.prepareStatement(sql);
```

    - 값 입력
      - set~ Method는 SQL Data Type과 호환되는 Type 으로 사용

```
pstmt.setInt    (1, 9998        );
pstmt.setString(2, "TestName2");
pstmt.setDouble(3, 3200.15     );
```

    - Query 실행

```
pstmt.executeUpdate();
```

    - PreparedStatement Object 종료

```
pstmt.close();
```

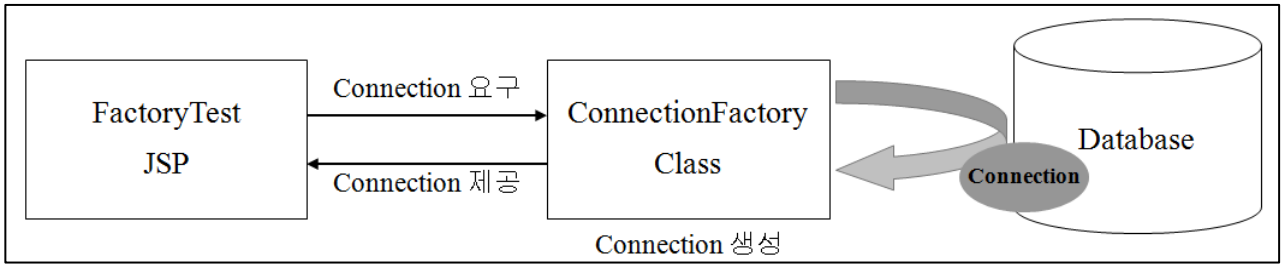
Database Programming

- PreparedStatement Interface
  - UsePreparedStatement.jsp

```jsp
<%@page contentType="text/html; charset=euc-kr"%>
<%@page import="java.sql.*"%>
<html>
    <head>
        <title>Database Programming</title>
    </head>
    <body>
        <h3>PreparedStatement 질의문 실행</h3>
        <%
            Class.forName("oracle.jdbc.driver.OracleDriver");
            String     url      = "jdbc:oracle:thin:@localhost:1521:ORCL";
            String     id       = "scott";
            String     password = "tiger";
            Connection conn     = DriverManager.getConnection(url, id, password);

            String sql = "Insert into TestEmp (empno, ename, sal) value (?, ?, ?)";
            PreparedStatement pstmt = conn.prepareStatement(sql);
            pstmt.setInt(1, 24);
            pstmt.setString(2, "Grace");

            pstmt.executeUpdate();
            pstmt.setInt(1, 44);
            pstmt.setString(2, "Cavin");
            pstmt.setDouble(3, 3200);
            pstmt.executeUpdate();

            out.print("레코드 삽입 완료<br>");
            pstmt.close();
            conn.close();
        %>
    </body>
</html>
```

Database Programming

- Factory 기법
  - 은…
    - Database와의 Connection을 위한 Factory Class 활용
    - Factory Class를 이용하여 Connection 생성



  - 동작과정
    - JSP Page에서 Connection이 필요할 경우 Factory Class에 Connection을 요청
    - Factory Class는 Connection을 생성해서 JSP Page로 전달

  - 장점
    - Code의 중복성을 줄이고 간결한 Coding이 가능
    - 재사용성을 높임

Database Programming

- Factory 기법
  - ConnectionFactory.java

```
import java.sql.*;

public class ConnectionFactory {
    private static ConnectionFactory connectionFactory = new ConnectionFactory();
    private ConnectionFactory(){};
    public static ConnectionFactory getDefaultFactory(){
        if(connectionFactory == null){
            connectionFactory = new ConnectionFactory();
        }
        return connectionFactory;
    }

    public Connection createConnection() {
        Connection connection = null;
        try {
            Class.forName("oracle.jdbc.driver.OracleDriver");
        } catch(ClassNotFoundException cnfe) {
            System.out.println(cnfe);
        }
        String     url      = "jdbc:oracle:thin:@localhost:1521:ORCL";
        String     id       = "scott";
        String     password = "tiger";
        try {
            connection = DriverManager.getConnection(url, id, password);
        } catch(SQLException sqle) {
            System.out.println(sqle);
        }
        return connection;
    }
}
```

Database Programming

- Factory 기법
  - getDefaultFactory() Method
    - 하나의 Object만을 생성해서 사용
    - ConnectionFactory Object를 얻기 위해서는
      - 반드시 getDefaultFactory() Metod 이용
    - static 특성으로 인해 Memory에 하나의 Object만 존재

```
public class ConnectionFactory {

    private static ConnectionFactory connectionFactory = new ConnectionFactory();

    public static ConnectionFactory getDefaultFactory(){
        connectionFactory = new ConnectionFactory();
        return connectionFactory;
    }
}
```

  - createConnection() Method
    - Connection을 생성하고 Connection을 돌려줌
    - JSP Page로부터 createConnection() Method가 호출되면
      - Database와의 Connection을 생성한후 Connection을 돌려줌

```
public Connection createConnection() {
    Connection connection = null;
    Class.forName("oracle.jdbc.driver.OracleDriver");
    connection = DriverManager.getConnection(url, user, password);
    return connection;
}
```

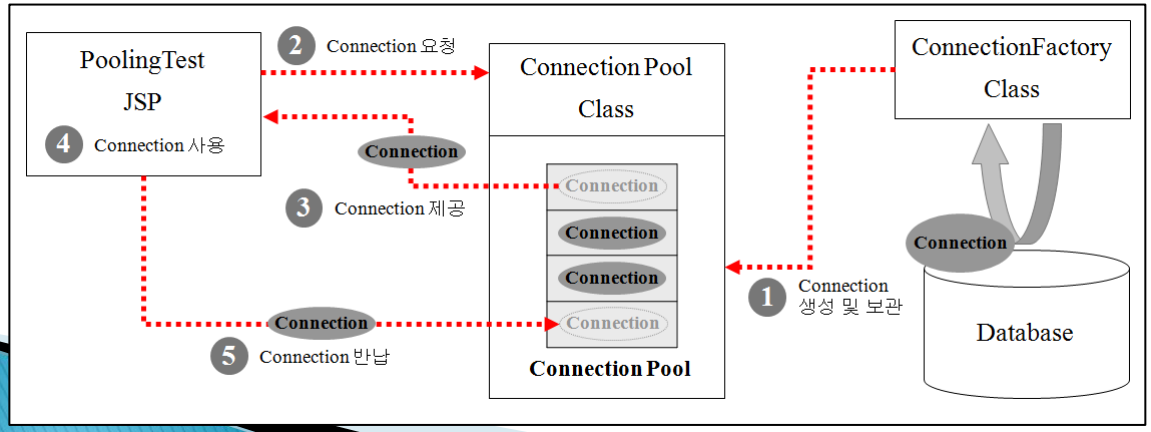
Database Programming

- Factory 기법
  - FactoryTest.jsp

```
<%@page contentType="text/html; charset=euc-kr"%>
<%@page import="java.sql.*%>
<html><head><title>Database Programming</title></head>
    <body>
        <h3>Factory 기법</h3>
        <%  ConnectionFactory connectionFactory = ConnectionFactory.getDefaultFactory();
            Connection        conn              = connectionFactory.createConnection();
            Statement         stmt              = conn.createStatement();
            String            sql               = "select * from TestEmp";
            ResultSet         rs                = stmt.executeQuery(sql);    %>
        <table border="1" cellspacing="0" cellpadding="3">
            <tr>
                <td align="center" bgcolor="#EEEEFF">empno</td>
                <td align="center" bgcolor="#EEEEFF">ename</td>
                <td align="center" bgcolor="#EEEEFF">sal</td>
            </tr>
        <%  while(rs.next()) {
                out.print("<tr>"                                                    );
                out.print("     <td align=\"center\">" + rs.getInt(1)    + "</td>" );
                out.print("     <td align=\"center\">" + rs.getString(2) + "</td>" );
                out.print("     <td align=\"center\">" + rs.getDouble(3) + "</td>" );
                out.print("</tr>"                                                   );
            }   %>
        </table>
        <%  rs.close();
            stmt.close();
            conn.close();    %>
</body></html>
```

Database Programming

- Pooling 기법
  - 은…
    - 미리 Database의 Connection을 여러 개 만들어서 저장
    - 사용자가 필요로 할 때마다 꺼내서 사용하고 다시 집어 넣는 방식
      - 원래는 필요할 때마다 Connection 생성, 사용 후 종료
    - Connection Pool이라는 Connection 관리자가
      - 미리 Connection을 만들어 놓고 필요할 때마다 빌려주는 방식
      - 연결과 해제를 관리
  - 동작 방식
    - Connection Pool은 미리 일정 개수의 Connection 을 생성
    - Database 작업이 필요한 JSP 파일은 Connection Pool에게 Connection을 요청
    - Connection 사용
    - 다 쓴 Connection은 다시 Connection Pool에게 반납



Database Programming

- Pooling 기법
  - 장점
    - 미리 Connection을 생성해 놓으므로 Connection 생성 시간을 계속해서 소비하지 않음
      - Connection Class는 Object화 될 때, 일반적인 다른 Class보다 수초정도의 시간이 걸림
      - Database가 원격지에 있다면 Network 상황에 따라 더 지연될 수도 있음
      - JSP Page가 빈번하게 Database Connection을 생성한다면, 시간적인 Overhead는 전체 System을 느리게 하는 원인이 됨
      - 일정 수의 Connection을 미리 만들어 놓고, Connection Pool을 통해 재사용 함으로써 상당한 속도 향상을 기대할 수 있음
    - 자원 공유 및 Connection Object 제어
      - 미리 생성된 Connection Object를 재사용하므로 효과적으로 System 자원 사용
      - 운용 가능한 수만큼의 Connection Object를 생성해서 사용하므로 System의 안정적인 운영이 가능

Database Programming

- Pooling 기법
  - 구현
    - Connection을 담아둘 Buffer 생성

```
private Vector buffer = new Vector();
```

    - 초기 Connection 생성
      - Connection Pool이 초기화되면 미리 일정한 개수의 Connection을 만들어 둠

```
public synchronized static void initConnectionPool() {
    Vector temp = ConnectionPool.getConnectionPool().getConnectionPoolBuffer();
    ConnectionFactory connectionFactory = ConnectionFactory.getDefaultFactory();
    for(int i = 0; i < MAX_CONNECTION; i++){
        Connection connection = connectionFactory.createConnection();
        temp.addElement(connection);
    }
}

public Vector getConnectionPoolBuffer() {    return this.buffer; }
```

      - Connection Pool이 가지고 있는 Buffer를 얻은 후
      - ConnectionFactory Object를 이용해서 MAX_CONNECTION에 지정된 수만큼의 Connection을 Vector에 삽입

Database Programming

- Pooling 기법
  - 구현
    - Connection 요청
      - Connection이 담긴 Vector로부터 하나의 Connection을 빼낸 후
      - getConnection의 반환값으로 return
      - 동기화 처리를 위한 Method에 synchronized 처리
      - 만약 Connection Pool의 Vector에 Connection이 없다면 wait()
      - Connection이 있다면 while문을 빠져 나와 Vector로부터 Connection을 빼낸 후 return

```
public synchronized Connection getConnection(){
    ConnectionInfo connectionInfo = null;
    while(buffer.size() == 0){
        this.wait();
    }
    connectionInfo = (ConnectionInfo)this.buffer.remove(0);
    return connectionInfo.connection;
}
```

    - Connection 반환
      - Parameter를 통해서 들어오는 Connection을
      - 다시 Connection Pool의 Vector에 삽입
        - wait()가 걸려있는 사용자를 풀어주기 위해
        - notifyAll()을 호출
        - 동기화 보장을 위해 Method에 sysnchronized 처리

```
public synchronized void releaseConnection(Connection Connection){
    this.buffer.addElement(new ConnectionInfo(Connection, System.currentTimeMillis()));
    this.notifyAll();
}
```

Database Programming

- Pooling 기법
  - ConnectionPool.java

```
public class ConnectionPool {
    private static int MAX_CONNECTION = 5;
    private Vector buffer = new Vector();
    private int wait_count = 0;
    private static ConnectionPool connectionPool = new ConnectionPool();
    static {
        try {
            initConnectionPool();
        } catch (SQLException e) {
            System.out.println("---Connection Create Error---");
            e.printStackTrace();
        } catch (ClassNotFoundException e2) {
            System.out.println("---Driver Calss Not Found Error--");
            e2.printStackTrace();
        }
    }

    private ConnectionPool() {      }

    public synchronized static void initConnectionPool() throws SQLException,
            ClassNotFoundException {
        ConnectionPool.getConnectionPool().destroyConnectionPool();
        Vector temp = ConnectionPool.getConnectionPool().getConnectionPoolBuffer();
        ConnectionFactory connectionFactory = ConnectionFactory.getDefaultFactory();
        for (int i = 0; i < MAX_CONNECTION; i++) {
            Connection connection = connectionFactory.createConnection();
            temp.addElement(new ConnectionInfo(connection, System.currentTimeMillis()));
            System.out.println("New Connection Created.." + connection);
        }
    }

    public synchronized static void destroyConnectionPool() {
        Vector temp = ConnectionPool.getConnectionPool().getConnectionPoolBuffer();
        int t = temp.size();
        System.out.println("버퍼의 크기는 : " + t);
        for (int i = 0; i < t; i++) {
            ConnectionInfo connectionInfo = (ConnectionInfo) temp.remove(0);
            if (connectionInfo.connection != null) {
                try {
                    connectionInfo.connection.close();
                    System.out.println("Connnection Closed.."+ connectionInfo.connection);
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
        }
    }
```

```
    public static ConnectionPool getConnectionPool() {
        if (connectionPool == null) {
            connectionPool = new ConnectionPool();
        }
        return connectionPool;
    }

    public synchronized Connection getConnection() {
        ConnectionInfo connectionInfo = null;
        if (wait_count > MAX_CONNECTION) {
            return null;
        }
        try {
            while (buffer.size() == 0) {
                wait_count++;
                this.wait();
                wait_count--;
            }
            connectionInfo = (ConnectionInfo) this.buffer.elementAt(0);
            long interval = System.currentTimeMillis() - connectionInfo.time;
            if (interval > 1000 * 60 * 30) {
                try {
                    System.out.print((interval / 1000)+ "초를 경과 Connection Close");
                    connectionInfo.connection.close();
                } catch (SQLException e1) {
                    e1.printStackTrace();
                    System.out.println("Connection Close Err");
                }
                ConnectionFactory connectionFactory = ConnectionFactory.getDefaultFactory();
                connectionInfo.connection = connectionFactory.createConnection();
                System.out.println(new java.util.Date().toString() + " Connection Open");
            }
        } catch (InterruptedException e2) {
            e2.printStackTrace();
        } finally {
            connectionInfo = (ConnectionInfo) this.buffer.remove(0);
        }
        return connectionInfo.connection;
    }

    public synchronized void releaseConnection(Connection Connection) {
        this.buffer.addElement(new ConnectionInfo(Connection, System.currentTimeMillis()));
        this.notifyAll();
    }

    public Vector getConnectionPoolBuffer() {
        return this.buffer;
    }
}
```

Database Programming

- Pooling 기법
  - 사용
    - PoolingTest.jsp

```
<%@page contentType="text/html; charset=euc-kr"%>
<%@page import="java.sql.*"%>
<html>
    <head>
        <title>Database Programming</title>
    </head>
    <body>
        <h3>Pooling 기법</h3>
        <%
            ConnectionPool pool = ConnectionPool.getConnectionPool();
            Connection     conn = pool.getConnection();
            Statement      stmt = conn.createStatement();
            String         sql  = "select * from TestEmp";
            ResultSet      rs   = stmt.executeQuery(sql);
        %>
        <table border="1" cellspacing="0" cellpadding="3">
            <tr>
                <td align="center" bgcolor="#EEEEFF">empno</td>
                <td align="center" bgcolor="#EEEEFF">ename</td>
                <td align="center" bgcolor="#EEEEFF">sal</td>
            </tr>
        <%
            while(rs.next()) {
                out.print("<tr>"                                                    );
                out.print("     <td align=\"center\">" + rs.getInt(1)    + "</td>" );
                out.print("     <td align=\"center\">" + rs.getString(2) + "</td>" );
                out.print("     <td align=\"center\">" + rs.getDouble(3) + "</td>" );
                out.print("</tr>"                                                   );
            }
        %>
        </table>
        <%
            pool.releaseConnection(conn);
        %>
    </body>
</html>
```

Database Programming

- Pooling 기법
  - 사용
    - Connection Object 생성

```
ConnectionPool pool = ConnectionPool.getConnectionPool();
Connection conn = pool.getConnection();
```

      - 이후 사용법은 일반 Connection 사용과 동일

    - Connection 반납

```
pool.releaseConnection(conn);
```

- Database Programming 은 맨날 똑같다.
  - Database 연결을 위해 Connection 얻기
  - 얻어온 Connection을 사용하여 Statement 생성
  - 생성된 Statement 에 SQL 문을 담아 Database 전송
  - Database는 전송된 SQL문을 수행 후 결과를 반환

- Connection Pool은 우리가 직접 작성할 필요도 없다.
  - Factory, Pooling 기법도 작성할 수준까지는 필요 없다.

- 단, 동작 원리는 반드시 파악하도록 하자.

오늘 숙제

- **집에서 해보자!**
- ConnectionPool.java 등 예제파일 분석
  - 드디어 분석 Season!