

2021-04-19

🕒 Created

2021년 4월 19일 오전 11:27

☰ Tags

비어 있음

※ 이클립스 초기 설정

▶ [첨부파일]

■ JDBC(Java DataBase Connectivity) 개념

■ JDBC 사전설정 및 주의사항

■ JDBC 프로그래밍 절차

■ 이클립스(Eclipse) 관련 주요사항

■ 이클립스에서 톰캣서버 추가 안될 ...

■ 이클립스(Eclipse) 주요 단축키

★ 자주 사용하는 단축키

○ Alt 주요 조합

○ Ctrl 주요 조합

○ Function Key 주요 기능

○ 디버깅 관련 주요 기능

○ 문장 자동 기능(템플릿) 사용 시 ...

○ 에디터 변환

○ 실행

○ 문자열 찾기

○ 기타 단축키

○ 자바와 오라클 연동하기

[Eclipse]

- JDBC01

DBConn.java

Test001

Test002

[SQL]

JDBC01_scott

※ 이클립스 초기 설정

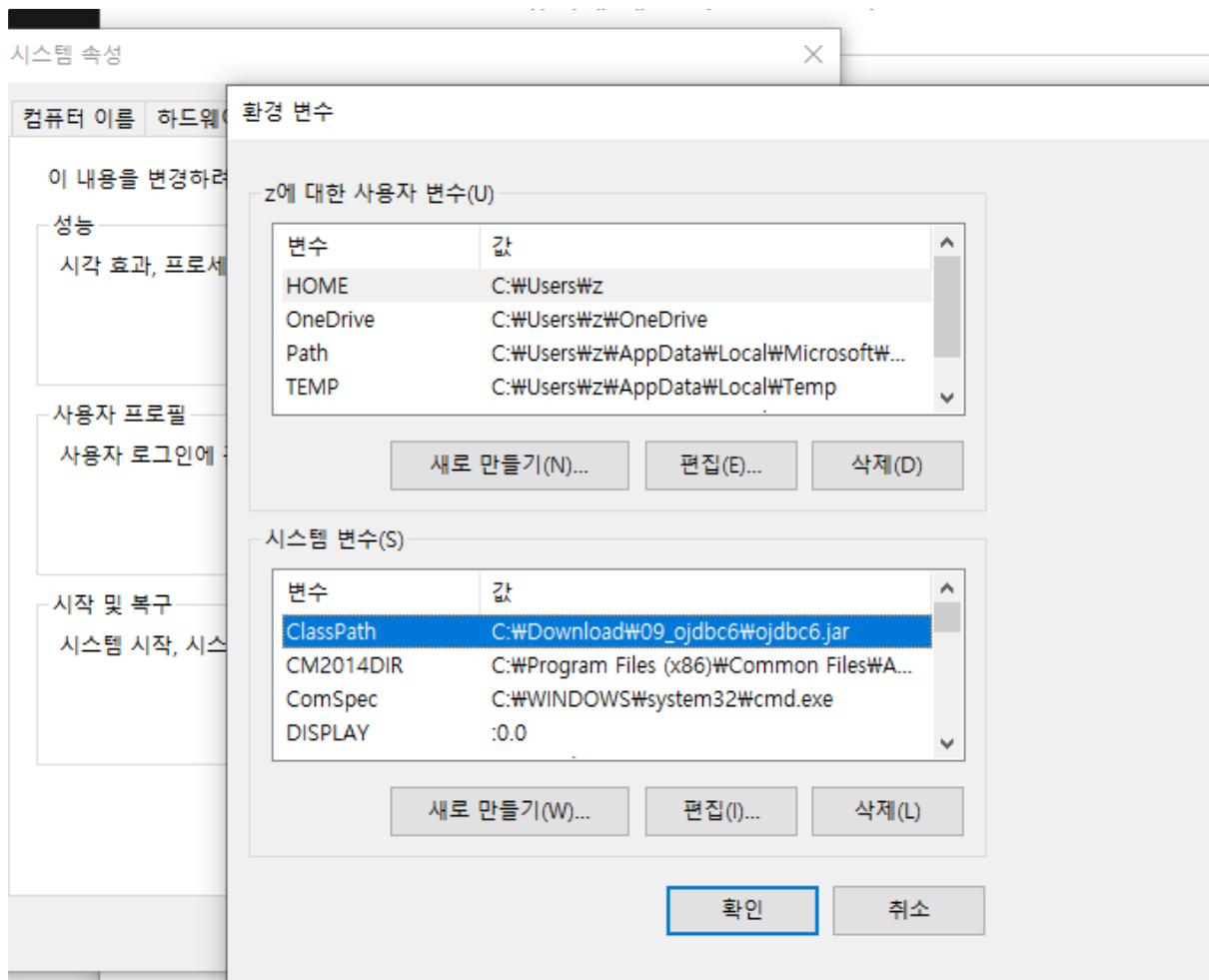
[폴더 이동]

이클립스 vol1 압축 풀고 c 드라이브에 eclipse 폴더 만들어서
eclipse 폴더 안에 bin 내용물 바로 들어있을 수 있게

tomcat 도 마찬가지로 c 드라이브로 옮기기

→ 폴더안에 바로 리소스 폴더(파일들) 있을수 있게 하라는 뜻!















[JDBC 설정]



이름(N):

값(V):

렉터리 찾아보기(D)... 파일 찾아보기(F)...

<< 로컬 디스크 (C:) > Program Files > Java > jdk1.8.0_281 > jre > lib > ext			
이름	수정된 날짜	유형	
 access-bridge-64.jar	2021-02-02 오전 10:09	ALZ	
 cldrdata.jar	2021-02-02 오전 10:09	ALZ	
 dnsns.jar	2021-02-02 오전 10:09	ALZ	
 jaccess.jar	2021-02-02 오전 10:09	ALZ	
 jfxrt.jar	2021-02-02 오전 10:09	ALZ	
 localedata.jar	2021-02-02 오전 10:09	ALZ	
 meta-index	2021-02-02 오전 10:09	파일	
 nashorn.jar	2021-02-02 오전 10:09	ALZ	
 ojdbc6.jar	2021-04-19 오전 11:48	ALZ	
 sunec.jar	2021-02-02 오전 10:09	ALZ	
 sunjce_provider.jar	2021-02-02 오전 10:09	ALZ	
 sunmscapi.jar	2021-02-02 오전 10:09	ALZ	
 sunpkcs11.jar	2021-02-02 오전 10:09	ALZ	
 zipfs.jar	2021-02-02 오전 10:09	ALZ	

↑ > 내 PC > 로컬 디스크 (C:) > Program Files > Java > jre1.8.0_281 > lib > ext			
	이름	수정한 날짜	유형
 바면 드 oad tudy 처 로 특 받은 파일 총평	access-bridge-64.jar	2021-02-02 오전 10:10	ALZip JA
	cldrdata.jar	2021-02-02 오전 10:10	ALZip JA
	dnsns.jar	2021-02-02 오전 10:10	ALZip JA
	jaccess.jar	2021-02-02 오전 10:10	ALZip JA
	jfxrt.jar	2021-02-02 오전 10:10	ALZip JA
	localedata.jar	2021-02-02 오전 10:10	ALZip JA
	meta-index	2021-02-02 오전 10:10	파일
	nashorn.jar	2021-02-02 오전 10:10	ALZip JA
	ojdbc6.jar	2021-04-19 오전 11:48	ALZip JA
	sunec.jar	2021-02-02 오전 10:10	ALZip JA
	sunjce_provider.jar	2021-02-02 오전 10:10	ALZip JA
	sunmscapi.jar	2021-02-02 오전 10:10	ALZip JA
	sunpkcs11.jar	2021-02-02 오전 10:10	ALZip JA
	zipfs.jar	2021-02-02 오전 10:10	ALZip JA

[이클립스 프로그램 설정]

- 이클립스 실행 → window → Preferences → General → Appearance → Colors and fonts
→ 제일아래 text font → 나눔고딕코딩으로 변경하고 Apply
 - General → Content Types → Text → Default encoding → UTF-8 넣고 UPDATE
 - General → Editor → Text Editors → show line numbers 체크 확인
 - General → Web Browser → Use external web browser → Chrome 체크 → Apply
 - Java → Code Style → Formatter → Active profile → new → Profile name :
classF_standard_JDBC → ok
대화창 뜨면 Brace positions → 맨 위 체크박스 클릭 → next line 변경
→ 제일 아래 Lambda body 만 Same line 으로 변경 → Apply → ok
 - Java → Compiler → 1.8인지 확인
 - 검색창에 encoding → Workspace → 좌측 하단에 Text file encoding → Other → UTF-8 → apply
- ⇒ 여기까지 완료 되었다면 Apply and Close

[이클립스에서 왼쪽 Project workspace 에서]

Create a Dynamic Web project

project name : JDBC01

new runtime → 톰캣 8.5

browser → c드라이브에 들어있는 아파치 톰캣 폴더 선택

→ next → next → finish

[패키지 / class 파일 생성]

- java resources → src → new → package → name : com.util
 - com.util (우클릭) → new → class → name : DBconn
 - java resources → src → new → package → name : com.test
-

■ JDBC(Java DataBase Connectivity) 개념

1. JDBC(Java DataBase Connectivity)는 자바 프로그램이 DBMS에 일관된 방식으로 접근할 수 있도록 API 를 제공하는 자바 클래스들의 모임으로 다음과 같은 특징을 가진다.

1. JDBC 는 함수 호출용 SQL 인터페이스
2. JDBC 는 ANSI SQL-92 표준을 지원
3. JDBC 는 공통된 SQL 인터페이스를 바탕
4. JDBC 는 익히고 사용하기 쉽다.

⇒ JDBC 란 데이터베이스에 연결 및 작업을 하기 위한 JAVA 의 표준 인터페이스이다.

2. JDBC 구성

1. 응용 프로그램
 - a. 데이터베이스에 연결을 요청
 - b. 데이터베이스에 SQL 문을 전송
 - c. SQL 문의 처리 결과 요청
 - d. 오류가 발생하는 경우에 오류 처리
 - e. 트랜잭션을 제어
 - f. 연결 종료
2. 드라이버 매니저
 - a. 데이터베이스에 맞는 드라이버 검색
 - b. JDBC 초기화를 위한 작업 수행
3. 드라이버
 - a. 데이터베이스에 연결
 - b. 데이터베이스에 SQL 문을 전달
 - c. 응용 프로그램에 검색 결과를 전달
 - d. 필요한 경우 커서를 조작
 - e. 필요한 경우 트랜잭션을 시작
4. DBMS
 - a. 데이터가 저장되어 있는 장소

3. System Architecture

JDBC API 는 2-tier 와 3-tier 를 모두 지원한다.



※ tier

일련의 유사한 객체가 나열된 상태에서 계층 또는 열을 나타낸다.
프로그램의 일부가 여러 객체에 나뉘어 존재할 수 있으며
그 계층 또한 네트워크 상의 다른 컴퓨터에 위치할 수 있다.

1. 2-tier

- a. 자바 애플릿이나 애플리케이션이 JDBC 를 이용하여 DBMS에 직접 접근
- b. 가장 대표적인 C/S 구조
- c. 프로그램이 간단하다는 장점
- d. 보안, 로드밸런싱, 확장성(오브젝트 재사용) 등의 문제점
- e. 2-tier 디자인이 적합한 경우
 - 애플리케이션이 하나의 데이터베이스만을 사용
 - 데이터베이스 엔진이 하나의 CPU에서 동작
 - 데이터베이스가 계속 거의 같은 크기로 유지
 - 사용자 기반이 같은 크기로 유지
 - 요구가 확정되어 변화 가능성이 극히 적거나 없는 경우
 - 애플리케이션을 종료한 후에도 최소한의 지속성을 요구하는 경우

2. 3-tier

- a. 자바 애플릿이나 애플리케이션이 DBMS에 직접 접근하는 것이 아니라
중간에 미들웨어(미들티어)를 거쳐 데이터베이스에 접근
- b. 데이터베이스와의 연동 부분을 분리시킴으로써
Presentation Layer 가 데이터베이스 저장 방법에 신경을 쓰지 않아도 된다.
- c. 클라이언트는 단지 미들티어를 참조
- d. 미들티어 서버는 DBMS와 같이 특정한 작업만 수행하는 최종 서버와
통신을 하여 결과를 얻은 후 이 결과를 클라이언트에 전달
- e. 2-tier 모델보다 안정적이고 유연하며 보안이 강화
 - tier 1 : 사용자 인터페이스를 담당하는 클라이언트
 - tier 2 : http, 코바 등을 지원하는 응용 처리 서버
 - tier 3 : DBMS와 같이 사용자가 최종적으로 원하는 기능을 수행할 서버

4. JDBC Driver 유형

1. Type 1

- JDBC-ODBC Driver

- 특징

가. 데이터베이스를 연동하기 위해 브릿지 기술을 사용

나. ODBC API로의 게이트웨이를 제공하여

실제로는 ODBC 의 API를 구현함으로써 데이터베이스를 연동

다. 브릿지 솔루션은 보통 클라이언트에 소프트웨어가 설치될 것을 요구

라. JDBC-ODBC Driver 는 ODBC Driver 가 풍부하기 때문에

거의 대부분 데이터베이스 시스템에서 사용할 수 있으며,

JDBC-ODBC Driver 를 사용하는 클라이언트에

사전에 ODBC Driver 가 설치되어 있는 경우 매우 유용하게 사용할 수 있다.

마. 속도와 관련한 가장 큰 문제

JDBC 를 통해 호출된 명령이 다시 ODBC 를 통해 나가야 하기 때문에

두 개의 브릿지를 거치며, 이로 인해 빠른 속도를 기대하기 어렵다.

빠른 성능을 요구하는 애플리케이션의 경우에는

Type 1. JDBC-ODBC 브릿지는 적당하지 않다.

바. JDBC-ODBC 브릿지를 사용하는 시스템에는

반드시 해당 데이터베이스에 연결하기 위한 ODBC Driver 가 설치되어야 하며,

이 단점은 애플릿에서 JDBC 를 사용하여 프로그래밍 해야 하는 경우 많은 문제가 된다.

애플릿을 사용하여 JDBC-ODBC 를 사용할 경우,

애플릿을 다운 받은 클라이언트에 미리 해당 ODBC Driver 가 설치되어 있어야 하기 때문에 배포 등에 많은 문제가 발생하게 된다.

2. Type 2

- Native-API / Partly Java Driver(사용 일부 자바)

- 특징

가. 각각의 데이터베이스 제조업체들이제공한 C 혹은 C++ 메소드를 자바 코드가 호출하는 방식

-> 벤더라고 불림 -> 약속되어있는 자바 코드만 불러온다.

나. 부분적으로 자바 드라이버인 원사 API 라고 일컬어짐

다. 데이터베이스와 연결되는 부분이 Native Code 로 구현되어 있는 만큼 JDBC-ODBC 브릿지에 비해 빠른 속도를 제공한다.

라. JDBC Driver 를 사용하고자 하는 각각의 클라이언트에

DBMS Vender 의 데이터베이스 라이브러리가 로드되어야 하기 때문에

인터넷이나 CS 환경에서는 사용하기 적합하지 않다.
또한 Type 3, 4 드라이버에 비해 낮은 성능

3. Type 3

- Net-Protocol / All-Java Driver(순수 자바)
- 클라이언트에서 일반적인 Network API 를 이용해 보낸 정보를 서버가 Database 에 독점적 형태로 변환하는 방식
- 특징
 - 가. 클라이언트에 존재하는 JDBC Driver 는 소켓을 사용하여 서버에 존재하는 미들웨어(Middleware) 애플리케이션에 연결
 - 나. 애플리케이션은 클라이언트의 요청을 사용하고자 하는 데이터베이스에 독점적인 API로 전환
 - 다. 하나의 드라이버로 여러 개의 데이터베이스를 연동
 - 라. 클라이언트에 소프트웨어를 설치할 필요가 없음

4. Type 4 → 변환이 필요없다. 보통 Type 4 를 쓰게된다.

- Native-Protocol / All-Java Driver(순수 자바)
- Database Engine 에 사용되는 Network Protocol 을 Java Socket 으로 직접 Database 에 교신하는 방식
- 특징
 - 가. 가장 직접적인 순수 자바 솔루션
 - 나. 거의 대부분 데이터베이스의 제조업체가 제공한다.
 - 다. ODBC 나 NativeLib 형태로 request 를 변환하지 않기 때문에 Performance 가 매우 좋다.
 - 또한, 특별하게 Driver 나 Lib, Middleware 등을 설치할 필요가 없기 때문에 배포 등이 매우 용이하다.

5. Oracle JDBC Driver

1. Type 4

오라클 『Thin Driver』라고 불리운다.

자바로 작성된 Net8의 TCP/IP 버전의 자체적인 실행을 포함하며
플랫폼에 독립적이고 실행 시간에 브라우저로 다운로드 된다.
그리고 서버 측에는 TCP/IP Listener 가 필요하며,
연결 스트링은 TNSNAMES 엔트리가 아닌 TCL/IP 주소와 포트번호이다.

URL Format → `jdbc:oracle:thin:@[host]:[port]:[database]`

→ host 에 기록하게 될건 오라클 서버 아이디이다.

→ 프로그램이 연동되는 계정 이름

→ port 은 오라클 포트번호를 기록

→ database 설정해놓은 접속해야하는 오라클서버의 si 서버
지금은 express edition 이니까 xe로 공통

6. JDBC API

JDBC API 는 데이터베이스의 데이터를 액세스 할 수 있도록 제공되는
표준 자바 API 클래스와 인터페이스로 구성되며
다음의 기능을 실행하기 위한 클래스와 인터페이스를 제공한다.

- JAVA 프로그램에서 데이터베이스 서버에 접속
- SQL문을 구성, 데이터베이스 서버에서 실행
- 데이터베이스 서버가 처리한 결과 가져오기
- 데이터베이스의 정보, 처리 결과에 대한 정보 등을 가져오기

※ 『`java.sql.*`』 패키지 → 이것을 주로 쓰게 될 것. 오라클과 자바 연동

- JAVA Application 으로부터 Database 를 조작하는 API 는

JDK의 코어(core) API로 `java.sql` 패키지에 설정되어 있다.

- JDBC 는 데이터베이스에 접속하기 위해 한 개의 클래스(`java.sql.DriverManager`)와

두 개의 인터페이스(`java.sql.Driver` 와 `java.sql.Connection`)를 사용한다.

■ JDBC 사전설정 및 주의사항

○ 해당 워크스테이션에 오라클이 설치된 경우(대면 수업 학습 환경)

시스템의 classpath를 다음과 같이 생성 및 변경

- (11g Express Edition 설치 경로가 c:\oraclexe 인 경우...)

.;C:\oraclexe\app\oracle\product\11.x.x\server\jdbc\lib\ojdbc6.jar -> 여기에선

ojdbc6.jar 이 있기때문에 복사해서 그대로 사용하여도 무방하다.
- (10g Enterprise Edition / Standard Edition 설치 경로가 c:\oracle 인 경우...)

.;C:\oracle\product\10.x.x\db_x\jdbc\lib\ojdbc14.jar

○ 해당 워크스테이션에 오라클이 설치되어 있지 않은 경우(비대면 수업 학습 환경 / 실무 환경)

- Oracle 용 JDBC 드라이버를 다운로드 해야 한다.

가. 다운로드 경로 및 절차

- ① <<http://www.oracle.com/database/technologies/appdev/jdbc.html>> 이동
- ② 해당 페이지에서 스크롤을 살짝 내려 Get Started 항목의 JDBC Download 클릭

→ <<https://www.oracle.com/database/technologies/appdev/jdbc-downloads.html>>

downloads.html>

※ 현재 우리에게 적합한 버전의 ojdbc6.jar 파일 없음
→ 별도 배포

나. 파일명

『ojdbc6.jar』 또는 『ojdbc8.jar』 또는 『ojdbc14.jar』 파일

다. 시스템의 classpath 에 변수값으로 ojdbc.jar(ojdbc14.jar)

파일 추가(경로 포함)

→ 캡처 참고

※ 공통

- JDBC 드라이버 설치를 위한 ojdbc6.jar 파일을

다음의 경로에 복사&붙여넣기

-> 이클립스를 사용하지 않더라도 jdbc를 사용하게끔 하기 위해
 - C:\Program Files\Java\jdk1.8.0_281\jre\lib\ext
 - C:\Program Files\Java\jre1.8.0_281\lib\ext

※ JSP 실습을 위한 추가 설정

- 웹 서버에서 실질적으로 Oracle 용 JDBC 드라이버를 찾는 경로는 『아파치톰캣루트\lib』이므로 ojdbc6.jar 파일을 이 경로에 복사&붙여넣기 한다.
- C:\apache-tomcat-8.5.54\lib
- 아파치 톰캣 다운로드 경로 → <http://tomcat.apache.org/>

■ JDBC 프로그래밍 절차

1. 드라이버 로딩
Class.forName()
→ Oracle Driver 를 Java 에서 사용하기 위해 드라이버를 JVM에 로딩하는 과정.
2. 커넥션 할당받기
DriverManager.getConnection()
3. 쿼리문 전송을 위한 작업 객체 할당받기
Statement 또는 PreparedStatement 할당받기
conn.createStatement() 또는 **conn.prepareStatement()**
4. 작업 객체를 활용하여 쿼리문 전송
○ DML(insert, update, delete)문인 경우
int updateCount = stmt.executeUpdate(sql)
→ 영향받은 레코드 수(적용된 행의 갯수) 반환
○ select 문인 경우
ResultSet rs = stmt.executeQuery(sql)
→ 결과 집합의 형태로 ResultSet 반환
5. (select 구문의 경우)
ResultSet 의 논리적 커서 이동을 통해
각 컬럼의 데이터를 바인딩 해 온다.
boolean b = rs.next();
→ 커서 이동.
커서가 위치한 지점에 레코드가 존재하면 true 반환, 없으면 false 를 반환.
커서는 가장 선두 첫 번째 레코드의 직전에 위치하고 있다가
『next()』가 호출되면 진행한다.

6. 사용을 마친 리소스 반납

`rs.close();` → **ResultSet** 사용했을 경우
`stmt.close();` → **Statement** 사용했을 경우
`DBConn.close();`
(null 체크하여 `close()` 해 주는 것을 권장.
finally 블록에서 구현하는 것을 권장.)

■ 이클립스(Eclipse) 관련 주요사항

○ 이클립스 다운로드 사이트

<http://www.eclipse.org/downloads>

write by KIM HO JIN

최종 수정 2020-07-24

○ 이클립스 한글 패치(권장하지 않음)

<http://www.eclipse.org/babel/downloads.php>

- Babel Language Packs 항목에서 이클립스 버전 클릭
- BabelLanguagePack-eclipse-ko_xxxxxx.zip
파일 다운로드
- 압축을 해제하여 이클립스 디렉토리에 덮어쓰기 한다.

○ 이클립스(Eclipse) 개요

이클립스는 프로젝트 기반으로 소스를 작성하고 실행하는 방식이다.
간단한 자바 소스를 작성하고, 터미널에서 컴파일 및 실행하는 정도로
실무에서 요구하는 응용 프로그램을 제작할 수 없다.

다른 개발자들이 제공하는 많은 라이브러리를 참조하고 공조하면서
솔루션 차원에서 작업해야 다양하고 세세한 기능을 구현하는 프로그램을 만들 수 있
다.

이러한 배경에서 이클립스와 같은 통합개발도구가 탄생하게 된 것이다.
이와같은 시대적 요구에 따라 이클립스는
프로젝트 단위로 소스를 관리하고 개발하며 패키징하는 방식을 사용하고 있다.

이클립스가 처음 실행될 때 워크스페이스(Workspace)를 결정하는 안내 창이 나타난다.

이때 입력하는 디렉터리 경로는...

기존에 존재하지 않는 디렉터리일 경우

"OK" 버튼을 클릭하면 입력한 디렉터리가 자동으로 생성되면서 새로운 워크스페이스를 할당하게 된다.

Workspace Launcher 창 아래에 있는

『Use this as default and do not ask again』을 체크하면

이 워크스페이스를 기본 워크스페이스로 지정하고

다음부터는 이 대화상자가 나타나지 않는다.

이 대화상자는 나중에 워크스페이스를 변경하는 메뉴에서 호출할 수 있기 때문에 이 체크상자를 체크해도 문제가 발생하지 않는다.

다만, 많은 워크스페이스로 작업을 해야 할 경우

현재 내가 실행한 이클립스가 어느 워크스페이스를 작업장으로 하고 있는지

확인할 필요가 있을 경우가 많이 있다.

이 경우 이 체크를 해제하여 이클립스를 실행할 때마다 확인함으로써

워크스페이스를 혼동하는 실수를 하지 않도록

경고 창의 기능으로 활용하는 것도 경험상 권장할 사항이다.

○ 이클립스가 실행되지 않는 경우

이클립스의 루트 경로의 『eclipse.ini』파일에 -vm 부분을 추가한다.

[eclipse.ini 파일]

-showsplsh

org.eclipse.platform

--launcher.XXMaxPermSize

256M

-framework

plugins\org.eclipse.osgi_xxx.jar

-vm

[JDK 7 설치경로]\bin\javaw.exe

-vmargs

-Dosgi.requiredJavaVersion=1.7

-Xms40m

-Xmx512m

○ JSP / HTML 한글 인코딩을 UTF-8로 변경

- Windows > Preferences... > Web > Jsp Files
- Windows > Preferences... > Web > HTML Files

우측 화면의 Encoding 에 "ISO 10646/Unicode(UTF-8)"로 변경 후 [ok] 버튼 클릭

○ 자바 스크립트 파일 한글 인코딩을 UTF-8로 변경

- Windows > Preferences... > General > Contents type ...

우측 화면의 Text-Javascript 의 Default encoding 에 "utf-8"로 변경 후 [update] 버튼 클릭

○ 워크스페이스의 문자셋을 UTF-8로 변경

(자바 파일, JSP, JavaScript 등 모든 문서의 내용을 UTF-8로 변경)

- Windows > Preferences > General > Workspace 의 Text file encoding 을 [Other-UTF-8]로 변경

○ 주석의 맞춤법 검사를 해주는 옵션 해제 (성능 저하)

- Windows > Preferences > General > Editors > TextEditors Spelling > Enable spell checking 옵션 해제

○ 외부에 있는 압축 파일 불러오기 (import)

1. Package Explorer 에서 프로젝트를 선택하고 [Import...]
2. Import 뷰에서 'Zip file'을 선택
3. [Browser...] 를 이용하여 선택할 압축 파일이 있는 디렉토리 찾기
4. 압축 파일(jar)을 선택하고 [열기] 버튼을 클릭
※ 필요한 것만 선택할 수 있고, 모든 파일을 선택할 수 있다.

○ 외부로 압축 파일 내보내기 (export)

1. Package Explorer 에서 프로젝트를 선택하고 [Export...]
2. Export 대화상자에서 'JAR file'을 선택하고 [Next]
3. 압축할 파일을 선택하고 [JAR file:]에 보낼 위치와 이름을 입력
 - 라이브러리식 jar 압축 파일을 만들려면 [finish]
 - 어플리케이션식 jar 압축 파일을 작성할 경우에는 [Next]
 1. JAR Export 대화상자에서 JAR Packing Option 은 기본사항을 사용
 2. 메인 클래스(main 메소드가 있는 클래스)를 선택
[Main class:]를 [Browser] 를 통하여 선택
 3. 압축 파일을 만들 것인지를 확인한 다음 [Yes]

○ 이클립스에서 JAVA 한글 도움말 설정

1. 이클립스 실행
2. 왼쪽 Package Explorer -> JRE System Library -> rt.jar -> 오른쪽 마우스 클릭 -> Properties 선택
3. Java Source Attachment -> Location path 의 내용을 지운다.

4. Javadoc Location -> Browse

<http://java.sun.com/javase/6/docs/api/> 를

<http://xrath.com/javase/ko/6/docs/ko/api/index.html> 로 변경

○ 자바 프로젝트의 생성

『New Java Project』 창이 나타나면 『Project name』 항목에 프로젝트명 입력
프로젝트명은 자바 Naming Rule 에 따라 영문과 숫자를 이용하여 입력
생성할 프로젝트에 대한 저장소를 『Location』 항목에서 확인하고
JRE 항목에 이 프로젝트에서 사용할 JRE 또는 JDK 버전을 설정

『Project layout』 항목은 자바 소스 파일과 컴파일 후 만들어지는 클래스 파일을
같은 폴더에 둘 것인지 분리할 것인지를 설정하는 프로젝트 생성 옵션이다.
오래전 자바 초창기에는 터미널에서 수동으로 일일이 컴파일해야 했기 때문에
소스 파일과 클래스 파일을 같은 폴더에 놓고 사용하는 습관이 생기게 되었다.

하지만 최근에는 이클립스와 같은 고차원의 개발 도구를 사용하면서
소스 파일과 배포본 클래스 파일을 완전히 분리하여 배포본의 버전 관리를 하는 것이
표준으로 되어가고 있다.

이클립스에는 프로젝트는 하나의 그룹으로 묶는 "Working Sets"라는 프로젝트 그룹
개념이 있다.

프로젝트를 생성할 때에도 Working Sets 설정 항목이 나타나는데
나중에 원하는 Working Sets에 프로젝트를 설정할 수 있기 때문에
굳이 지금 당장 설정하지 않고 넘어가도 무방하다.

이클립스의 Working Sets에 대해 익숙해지면
프로젝트를 생성할 때 어느 Working Sets을 설정해야 하는지 자연스럽게 알게 될 것
이다.

"New Java Project" 창이 "Java Settings"과정으로 전환되었다.

이 화면에서는 자바 프로젝트가 사용할 소스 폴더와 컴파일 결과 폴더
그리고 기타 프로젝트에 대한 설정 및 참조 라이브러리 등을 추가로 설정할 수 있다.

이 모든 설정들은 나중에 프로젝트 속성에서 다시 변경 설정할 수 있기 때문에
초반부터 일일이 살펴볼 필요는 없다.

단지, JavaProject1이라는 프로젝트는
src 폴더를 소스 폴더로 정의하고, bin 폴더를 결과 폴더로 정의하고 있다는
정도로 이해하고 "Finish"버튼을 클릭하면 된다.

프로젝트 생성을 완료할 때 "Open Associated Perspective?"라는 확인 창이 나타나게
된다.

이는 자바 프로젝트에 대한 분할영역 화면(Perspective)을 처음 열기 때문이다.
이 창에서 "Yes"버튼을 클릭하여 지금 생성한 프로젝트에 적합한 분할영역 화면이 나
타나게 한다.

Java Perspective 분할영역 화면이 나타나면서 JavaProject1이라는 자바 프로젝트가 나타난다.

이클립스는 분할영역 화면 왼쪽에 있는 Package Explorer 창 또는 Project Explorer 창을 통해

작업할 프로젝트와 그 안에 들어있는 소스 파일들을 탐색할 수 있다.

■ 이클립스에서 톰캣서버 추가 안될 때 (Server Name 이 빈칸으로 나올 때)






1. 이클립스를 종료한다.(Close Eclipse)
2. 다음 경로의 두 개 파일을 삭제한다.
`{workspace-directory}/.metadata/.plugins/org.eclipse.core.runtime/.settings`
 - ① org.eclipse.wst.server.core.prefs
 - ② org.eclipse.jst.server.tomcat.core.prefs
3. 이클립스를 다시 실행시킨다.(Restart Eclipse)

■ 이클립스(Eclipse) 주요 단축키

write by KIM HO JIN

최종 수정 2020-07-24

★ 자주 사용하는 단축키

1. 주요 단축키 보기
 Ctrl + Shift + L
2. 블록 단위 주석 처리(/* */)
 -  블록 지정한 후에 Ctrl + Shift + /
3. 블록 단위 주석 해제
 -  블록 지정한 후에 Ctrl + Shift + \
4. 라인 단위 주석 처리(//) 또는 제거
 Ctrl + /
5. 자동 완성 기능. 어휘의 자동완성(Content Assistance)
 Ctrl + Spacebar

6. 라인 이동 : 커서가 있는 라인의 모든 글자가 방향키에 따라 이동
▶ Alt + 방향키(위,아래)
7. 라인 삭제 : 커서가 있는 라인을 삭제
▶ Ctrl + D
8. 에러 픽스 : 에러난 곳에 대해 해결 방법을 제시함
▶ 에러 발생 지점(빨간밑줄)에서 Ctrl + 1
9. Undo / Redo
▶ Ctrl + Z / Ctrl + Y
10. System.out.println(); 생성
▶ syso 입력 후 Ctrl + Spacebar
11. 들여쓰기 자동 수정
→ 커서가 있는 라인의 들여쓰기를 자동으로 맞춰준다.
블럭을 지정하고 실행 시 블럭 내에서 자동 들여쓰기
▶ Ctrl + I
12. Getter / Setter 자동 생성
▶ Shift + Alt + S R
13. 디버깅 시작
▶ F11
14. 상속 구조 클래스 보기(메소드 등)
클래스명을 선택하고 누르면 해당 클래스의 Hierarchy 를 볼 수 있다.
▶ F4
15. 변수 및 메소드 변경 : 같은 이름 모두 바꾸기(변경할 변수/메소드/클래스에서 단축키를 누르고 변경 후에 엔터를 누르면 이름이 모두 변경)
▶ Alt + Shift + R
16. 에디터 화면 넓게 / 좁게
▶ Ctrl + M
17. 열 복사 모드로 전환 / 해제
▶ Alt + Shift + A
18. 저장 및 컴파일
▶ Ctrl + S
19. 열린 파일 모두 닫기
▶ Ctrl + Shift + F4
20. 열린 파일 모두 닫기
▶ Ctrl + Shift + W

21. 창 닫기

▶ Ctrl + W

○ Alt 주요 조합

▶ Alt + Shift + J ⇒ 설정해 둔 기본주석 부여(JavaDoc 주석)

▶ Alt + Shift + R ⇒ 변수 및 메소드 변경

→ 변경할 변수 에서 단축키를 누르고

변경 후에 엔터를 누르면 변수명이 모두 변경

▶ Alt + Shift + 방향키 ⇒ 블록 선택하기

▶ Alt + ←→(좌/우) ⇒ 뷰 화면의 탭에 열린 페이지 이동(이전과 이후)

▶ Alt + ↑↓(상/하) ⇒ 커서가 있는 줄을 위 아래로 이동하여 자리 바꾸기

→ 커서가 있는 줄의 모든 글자가 방향키에 따라 위아래로 이동

○ Ctrl 주요 조합

▶ CTRL + 휠 ⇒ 페이지 단위 이동

▶ CTRL + 객체클릭 ⇒ 클래스나 메소드 혹은 멤버를 정의한 곳으로 이동(Open Declaration)

▷ F3

▶ Ctrl + 1 ⇒ Quick Fix(자동 완성 기능의 업그레이드형)

- 구현하지 않은 메소드 추가

- 로컬 변수 이름 바꾸기

- Assignment 입력

- 행 둘러싸기 (if/where/for이나 블록으로 둘러 싸려면 해당영역을 선택하고 Ctrl + 1)

▶ Ctrl + D ⇒ 한 라인 삭제 (커서가 있는 줄을 삭제)

▶ Ctrl + E ⇒ 뷰 화면의 탭에 열린 페이지 이동

▶ Ctrl + L ⇒ 특정 라인으로 이동 (대화창에 숫자 입력 → 원하는 라인으로 이동)

▶ Ctrl + I ⇒ 자동 들여쓰기 수정

→ 커서가 있는 줄의 들여쓰기를 자동으로 맞춰준다.

블록을 지정하고 실행 시 블록내에서 자동 들여쓰기

- ▶ Ctrl + K ⇒ 문자열 찾기 (찾고자 하는 문자열을 블록으로 설정한 후...)
- ▷ Ctrl + Shift + K ⇒ 역순으로 찾기
- ▷ Ctrl + J ⇒ 입력하며 실시간으로 찾기
- ▷ Ctrl + Shift + J ⇒ 입력하며 실시간으로 역순으로 찾기
- ▷ Ctrl + F ⇒ 기본으로 찾기
- ▷ Ctrl + H ⇒ Find 및 Replace
- ▶ Ctrl + M ⇒ 전체화면 토글 (선택한 화면 전체보기)
→ 에디터 화면 넓게/좁게
- ▶ Ctrl + O ⇒ 현재 편집 화면의 아웃라인 (메소드 리스트 확인, 메소드나 필드 이동 가능)
- ▶ Ctrl + Q ⇒ 마지막 편집위치로 가기
- ▶ Ctrl + T ⇒ 계층 구조 보기(★★★)
→ 하이어라키 팝업 창 띄우기(인터페이스 구현 클래스간 이동시 편리)
- ▶ Ctrl + W ⇒ 창 닫기 (선택 소스 닫기)
- ▷ Ctrl + Shift + W ⇒ 열린 파일 모두 닫기
- ▷ Ctrl + Shift + F4 ⇒ 열린 파일 모두 닫기
- ▶ Ctrl + , or . ⇒ 이전 또는 다음 Annotation(Error, Warning, Bookmark)으로 이동
- ▶ Ctrl + / ⇒ 주석 처리 (여러줄 블록 처리 가능)
- ▷ Ctrl + Shift + / ⇒ JavaDoc 주석 추가
- ▶ Ctrl + Alt + ↑↓(상/하) ⇒ 한줄(블록) 복사
- ▶ Ctrl + Shift + E ⇒ Switch to Editor (탭에 열려있는 Editor 이동)
- ▶ Ctrl + Shift + F ⇒ 자동 줄맞춤. 알아서 정렬. (소스 정리)
- ▶ Ctrl + Shift + G ⇒ 클래스의 메소드나 필드를 Reference하고 있는 곳으로 이동
- ▷ 반대 : F3 (Reference하는 클래스로 이동)
- ▶ Ctrl + Shift + L ⇒ 단축키 보기 (Window > Preference > General > Keys 메뉴에서 확인 가능)
- ▷ Ctrl + Shift + L + L ⇒ 단축키 지정
- ▶ Ctrl + Shift + M : 캐럿이 위치한 대상에 필요한 특정클래스 import
- ▶ Ctrl + Shift + O ⇒ 자동 import 갱신 처리 (사용하지 않는 Class는 삭제)
- ▶ Ctrl + Shift + R ⇒ Open Resource (파일 이름으로 소스찾기)
- ▶ Ctrl + Shift + U ⇒ 소스 내에서 선택한 변수, 클래스, 메소드 등을 사용하는 라인 찾기
- ▶ Ctrl + Shift + ↑↓(상/하) ⇒ 다음/이전 메소드 단위로 이동

- ▶ Ctrl + Shift + Spacebar ⇒ 메소드 파라미터 힌트 (메소드에 입력해야 하는 파라미터 정보가 표시된다.)
- ▶ Ctrl + F3 ⇒ 클래스 아웃라인
- ▶ Ctrl + F6 ⇒ View 화면의 탭에 열린 페이지 이동 (재정의 하는게 편리)
→ ULTRAEDIT나 EDITPLUS 의 Ctrl + TAB 과 같은 기능
- ▶ Ctrl + F7 ⇒ View 간 화면 전환
- ▶ Ctrl + F8 ⇒ Perspective 간 화면 전환
- ▶ Ctrl + F11 ⇒ 바로 전에 실행했던 클래스 실행
- ▶ Ctrl + PageDown ⇒ 뷰 화면의 탭에 열린 페이지 이동
- ▶ Ctrl + 2 + R : Rename(리팩토링)

○ Function Key 주요 기능

- ▶ F2 ⇒ 에러의 원인에 대한 힌트 제공(에러 라인 빨간줄에 커서를 위치시킨 후 ...)
- ▶ F3 ⇒ Java 편집기에서 Reference하는 클래스의 자바파일로 이동 / 메소드 정의부로 이동
 - ▷ Ctrl + 클릭
 - ▷ 반대 : Ctrl + Shift + G (클래스의 메소드나 필드를 Reference하고 있는 곳으로 이동)
- ▶ F4 ⇒ 해당 클래스의 Hierarchy (클래스명을 선택하고 누르면 해당 클래스의 Hierarchy(계층)를 볼 수 있다.)
- ▶ F11 ⇒ 디버깅 시작
- ▶ F12 ⇒ Editor로 포커스

○ 디버깅 관련 주요 기능

- ▶ F11 ⇒ 디버깅 시작 (디버그 모드로 실행)
- ▶ F5 ⇒ Step Into
(현재의 명령문이 호출되는 메소드 속으로 진행하여, 그 첫 문장을 실행하기 전에 멈춘다.
하지만, 자바 라이브러리 클래스 수준까지 들어가므로 단계필터 사용을 체크(Shift + F5)를 하면
필터를 설정한 클래스에 대해서는 Step Over 기능과 같은 기능을 수행한다.)

▶ F6 ⇒ Step Over

(현재의 명령문을 실행하고 다음 명령문 직전에 다시 멈춘다.)

▶ F8 ⇒ Resume

(멈추어 있던 스레드를 다시 계속 실행한다.)

▶ Ctrl + Shift + B ⇒ 현재커서위치에 Break point 설정 또는 해제

※ Display view(표시)

: Window > Show View > Other > Debug > Display 를 선택하여
소스상에서 필요한 부분을 선택해서 실행시켜 볼 수 있다.

○ 문장 자동 기능(템플릿) 사용 시 키워드

▶ sysout ⇒ System.out.println()

▶ try ⇒ try~catch문 완성.

▶ for ⇒ for문 완성 배열에 관련된 변수가 조건문 알아서 들어감.

▶ do ⇒ do~while문 완성.

▶ while ⇒ while문 완성.

※ 문장 자동 완성 템플릿을 수정하거나 추가하려면

Window > Preference > Java > Editor > Templates 에서 할 수 있다.

○ 에디터 변환

▶ 에디터가 여러 파일을 열어서 작업중일때

Ctrl + F6 키를 누르면 여러파일명이 나오고

F6키를 계속 누르면 아래로 → 키보드 누르면 화면이 고정되어 선택할수있음

▷ Ctrl + Shift + F6 키를 누르면 위로 커서가 움직인다.

▶ Alt + 방향키 ⇒ 소스코드 네비게이션(순서대로 나타나고 닫은창도 버퍼안 있으면 살아남)

▶ Ctrl + 마우스커서(혹은 F3) ⇒ 클래스나 메소드 혹은 멤버를 상세하게 검색하고자 할때 (f3은 부모로 바로날라감)

▶ Ctrl + F7 ⇒ 뷰(View)간 네비게이션 화면 전환

▶ Ctrl + F8 ⇒ 모드(Perspective)창 간의 네비게이션 화면 전환

▶ Ctrl + E ⇒ 뷰(View) 화면의 탭에 열린 페이지 이동

▷ Ctrl + F6 ⇒ 뷰(View) 화면의 탭에 열린 페이지 이동

▶ F12 : 에디터로 포커스 위치

○ 실행

▶ Ctrl + F11 ⇒ 바로 전에 실행했던 클래스 실행.

▶ Shift + Alt + X ⇒ 실행에 관련 된 단축키 나옴 (단축키보기)

→ (A ⇒ 애플릿실행, R ⇒ 서버실행, J ⇒ 어플리케이션실행)

○ 문자열 찾기

▶ Ctrl + K ⇒ 찾고자 하는 문자열을 블록으로 설정한 후 키를 누른다.

▷ Ctrl + Shift + K ⇒ 역으로 찾고자 하는 문자열을 찾아감.

▶ Ctrl + J ⇒ 입력하면서 찾을 수 있음.

▷ Ctrl + Shift + J ⇒ 입력하면서 거꾸로 찾아갈 수 있음.

▶ Ctrl + F ⇒ 기본적으로 찾기

▶ Ctrl + H : Find 및 Replace

○ 기타 단축키

▶ Ctrl + 3 ⇒ 기술검색해서 사용하기.

▶ Ctrl + Alt + J ⇒ 소스 한줄로 바꾸기 (// 주석삭제필수 , 자동줄맞춤과 함께 사용하여 한 줄로 찾아바꾸기 활용)

▶ Ctrl + Alt + L ⇒ 소문자 형태로 바꾸기

▶ Ctrl + Alt + K ⇒ test_aaa 형태를 testAaa 형태로 바꾸기 (마켓에서 anyedit 플러그인 설치)

▶ 블록지정 + Shift + Alt + Z ⇒ 관련된 여러 가지 기능 문들 나옴.(괜찮은 기능)

▶ Ctrl + O ⇒ 해당 소스의 메소드 리스트를 확인

▶ Alt + Shift + up,down ⇒ {} 단위로 선택됨 up하면할수록 더넓은범위 ex) if -> 메소드, 클래스

▶ Alt + Shift + S → R ⇒ Getter/setter 자동생성

▶ Ctrl + Shift + T ⇒ 자바소스찾기 (jar파일안에있는소스들 볼수있음)

▶ Ctrl + Alt + H ⇒ 메소드나 클래스 이름을 블록하고 누르면 메서드를 사용하는 모든 소스를 보여준다(라인까지상세하게)

▶ Alt + Shift + L ⇒ 선택한코드를 지역변수화시켜 자동등록 (Kjjj.getTest = 111; → Kjjj1 = kjjj kjjj1.getTest= 1111; 이런식)

▶ Alt + Shift + M ⇒ 블록된 소스를 매서드화시켜서 빼내고 그걸 자동으로 호출하는 소스까지만들어준다~

▶ Ctrl + Alt + 방향키 ⇒ 아래나 위로 하면 그 라인이 복사&붙여넣기 된다.

▶ Windows - Preferences - Java - Editor - Syntax Coloring - java

- Keyword 'return' , Keywords excluding 'return' = (blue)

- Classes - (red)

▶ Windows - Preferences - General - Appearance - Colors and Fonts - Basic - Text Font → Tahoma, Verdana, 굵게, 크기12로 조정

○ 자바와 오라클 연동하기

[Eclipse]

- *JDBC01*

DBConn.java


```

package com.util; import java.sql.Connection;
/*=====*/ /* ※ 싱글톤(Singleton) 디자인 패턴을 이
용한 Database 연결 객체 생성 전용 클래스 → DB 연결 과정이 가장 부하가 크기 때
문에 한 번 연결된(생성된) 객체를 계속 사용하는 것이 좋다. */ import
java.sql.DriverManager; public class DBConn { // 변수 선언 private static
Connection dbConn; //-- 자동 null 초기화 // 메소드 정의 → 연결 public
static Connection getConnection() //-- Connection 타입을 반환 { // 한 번
연결된 객체를 계속 사용 // 즉, 연결되지 않은 경우에만 연결을 시도하겠다는 의미
// → 싱글톤(디자인 패턴 = 공식,설계) if (dbConn == null) { try { String
url="jdbc:oracle:thin:@211.238.142.162:1521:xe"; //--
『211.238.142.162:1521』는 오라클 서버 ip 주소를 기재하는 부분 // 원격지의
오라클이 아니라 로컬의 오라클 서버일 경우는 // 『localhost』이나
『127.0.0.1』과 같이 loop back address 로 기재하는 것도 가능 // 『1521』은
오라클 리스너 기본 Port Number // 『xe』는 오라클 SID(Express Edition 은
xe) String user = "scott"; //-- 오라클 사용자 계정 이름 String pwd =
"tiger"; //-- 오라클 사용자 계정 암호
Class.forName("oracle.jdbc.driver.OracleDriver"); //-- OracleDriver 클래스
에 대한 객체 생성 dbConn = DriverManager.getConnection(url, user, pwd);
//-- 오라클 서버 실제 연결 // 갖고 있는 인자값(매개변수)은 오라클주소, 계정
명, 패스워드 } catch(Exception e) // ClassNotFoundException, SQLException
{ System.out.println(e.toString()); //-- 오라클 서버 연결 실패 시 오류 메세
지 출력 부분 } } return dbConn; //-- 구성된 연결 객체 반환 } //
getConnection() 메소드의 오버로딩 → 연결 public static Connection
getConnection(String url, String user, String pwd) { if (dbConn == null)
{ try { Class.forName("oracle.jdbc.driver.OracleDriver"); dbConn =
DriverManager.getConnection(url, user, pwd); } catch (Exception e) {
System.out.println(e.toString()); } } return dbConn; } // 메소드 정의 → 연
결 종료 public static void close() { // dbConn 변수(멤버 변수)는 //
Database 가 연결된 상태일 경우 Connection 을 갖는다. // 연결되지 않은 상태라
면... null 을 갖는다. if (dbConn != null) { try { // 연결 객체의 isClosed()
메소드를 통해 연결 상태 확인 //-- 연결이 닫혀있는 경우 true 반환 // 연결이 닫
혀있지 않은 경우 false 반환 if (!dbConn.isClosed()) dbConn.close(); //-- 연
결 객체의 close() 메소드를 통해 연결 종료 } catch (Exception e) {
System.out.println(e.toString()); } } dbConn = null; //-- 연결 객체 초기화
} }

```

Test001

```

/*===== Test001.java - main() 메소드
를 포함하는 테스트 클래스 =====*/
package com.test; import java.sql.Connection; import com.util.DBconn;
public class Test001 { public static void main(String[] args) {
Connection conn = DBconn.getConnection(); // ※ DB 연결 과정이 가장 부하가
크기 때문에 // 한 번 연결된 객체를 계속 사용할 수 있도록 Singleton 패턴 적용
if (conn != null) { System.out.println("데이터베이스 연결 성공~~!!"); }
DBconn.close(); //-- close() 메소드 호출을 통해 연결 종료 } }

```

Test002

```

/*===== Test002.java - main() 메소드
를 포함하는 테스트 클래스 - 데이터베이스 연결 - 데이터 입력
=====*/ package com.test; import
java.sql.Connection; import java.sql.Statement; import com.util.DBconn;
public class Test002 { public static void main(String[] args) {
Connection conn = DBconn.getConnection(); if (conn == null) {
System.out.println("데이터베이스 연결 실패~!!!"); System.exit(0); }
//System.out.println("데이터베이스 연결 성공~!!!"); try { // 작업 객체 준비
Statement stmt = conn.createStatement(); // 쿼리문 준비 String sql =
"INSERT INTO TBL_MEMBER(SID, NAME, TEL) VALUES (2, '김서현', '010-2222-
2222')"; //-- 주의. 쿼리문 끝에 『;』 붙이지 않는다. //-- 주의. 자바에서 실행
한 DML 구문은 내부적으로 자동 commit 된다. //-- 주의. 오라클에서 트랜잭션 처
리가 끝나지 않으면 (commit) // 데이터 액션 처리가 이루어지지 않는다. int
result = stmt.executeUpdate(sql); if (result > 0) { System.out.println("데
이터 입력 성공~!!!"); } else { System.out.println("데이터 입력 실패ㅠ_ㅠ");
} } catch (Exception e) { System.out.println(e.toString()); }
DBconn.close(); } }

```

- ⚠ • 쿼리문 끝에 『;』 붙이지 않는다.
- 자바에서 실행한 DML 구문은 내부적으로 자동 commit 된다.
- 오라클에서 트랜잭션 처리가 끝나지 않으면 (commit) 데이터 액션 처리가 이루어지지 않는다.

[SQL]

JDBC01_scott

```
SELECT USER FROM DUAL; ----> SCOTT DROP TABLE TBL_MEMBER; ----> Table
TBL_MEMBER이(가) 삭제되었습니다. --o 실습 테이블 생성 CREATE TABLE
TBL_MEMBER ( SID NUMBER , NAME VARCHAR2(30) , TEL VARCHAR2(60) ,
CONSTRAINT MEMBER_SID_PK PRIMARY KEY(SID) ); ----> Table TBL_MEMBER이(가)
생성되었습니다. --o 샘플 데이터 입력 INSERT INTO TBL_MEMBER(SID, NAME, TEL)
VALUES (1, '김가영', '010-1111-1111'); ----> 1 행 이(가) 삽입되었습니다.
SELECT * FROM TBL_MEMBER; ----> 1 김가영 010-1111-1111 --o 커밋 COMMIT; --
==> 커밋 완료. --o 자바에서 Test002 클래스 실행 후 다시 확인 SELECT * FROM
TBL_MEMBER; ----> /* 1 김가영 010-1111-1111 2 김서현 010-2222-2222 */
```