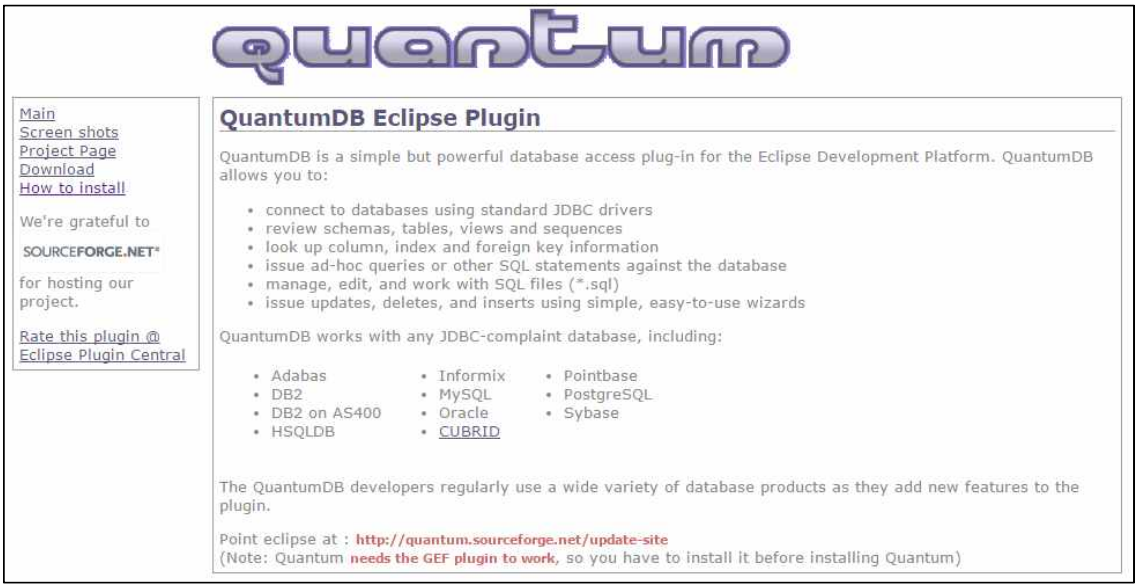
■ Quantum DB 플러그인의 설치

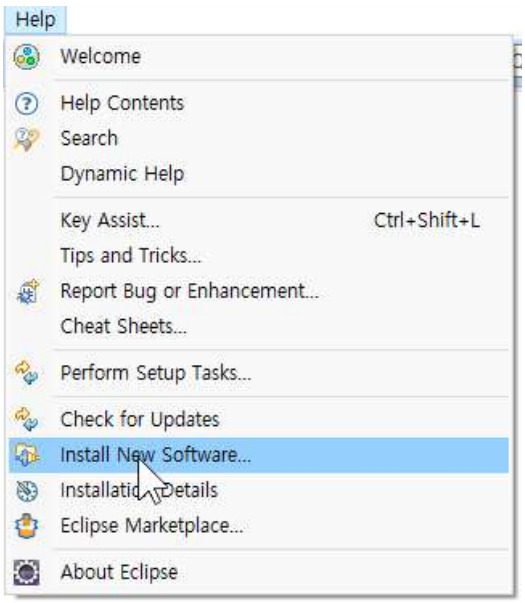
1) http://quantum.sourceforge.net/



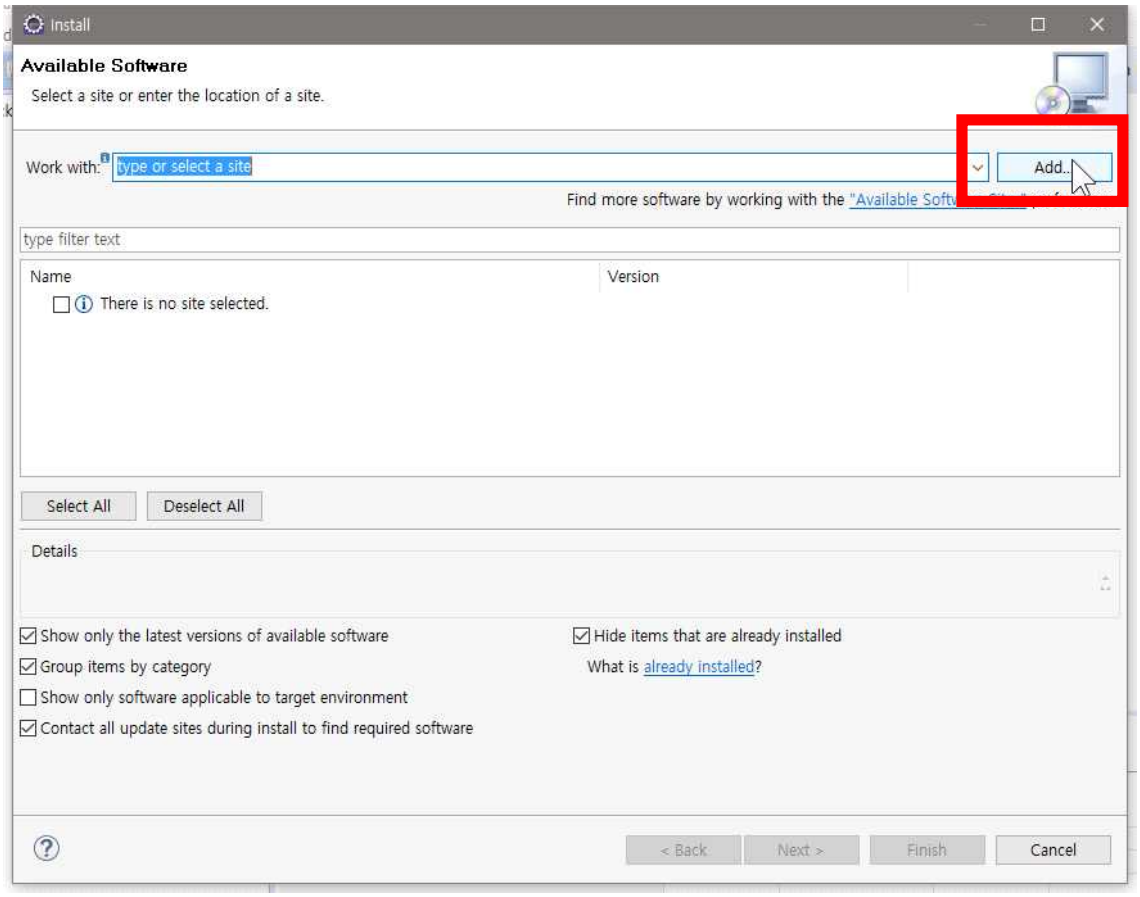
- How to install로 이동

- 설치 주소 : http://quantum.sourceforge.net/update-site

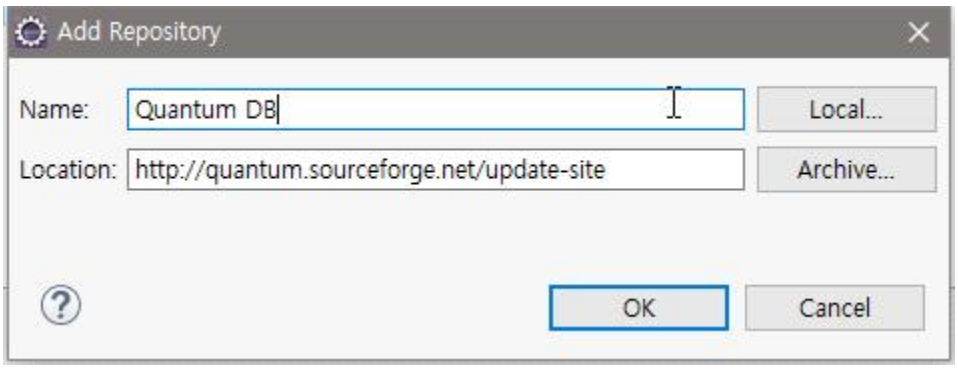
2) 메뉴 -> Help -> Install New Software



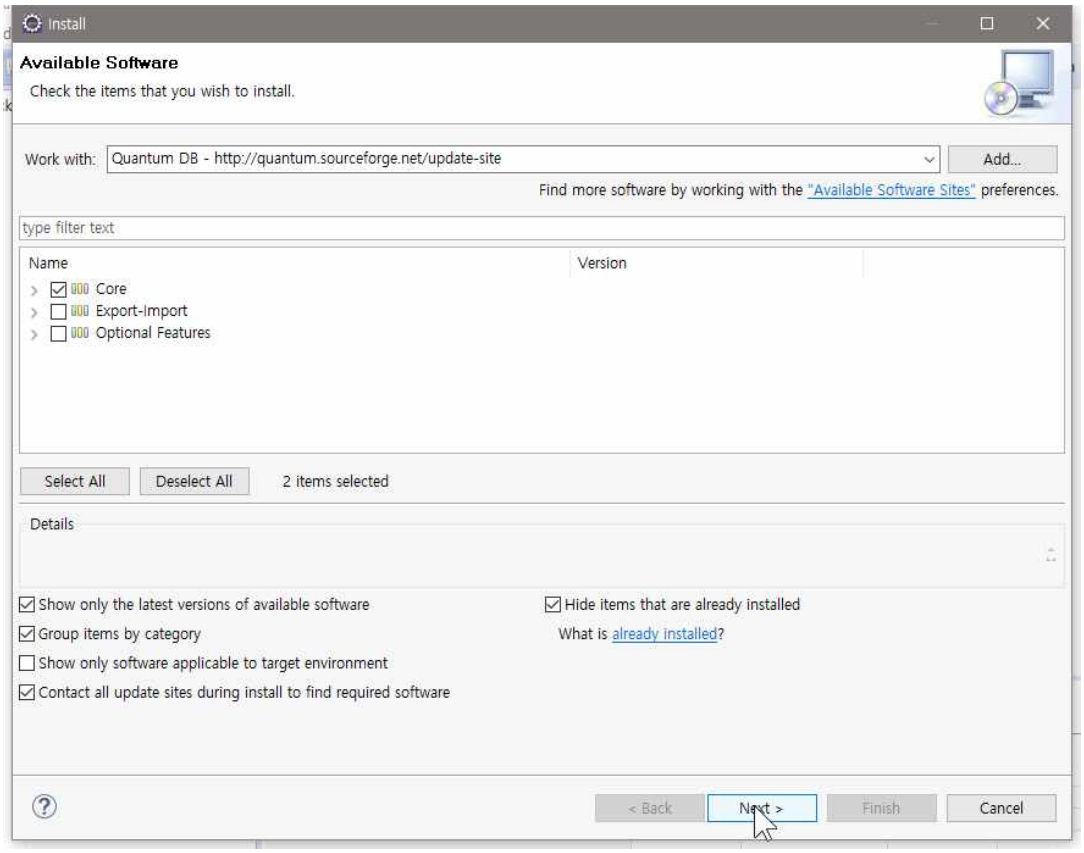
3) add 클릭



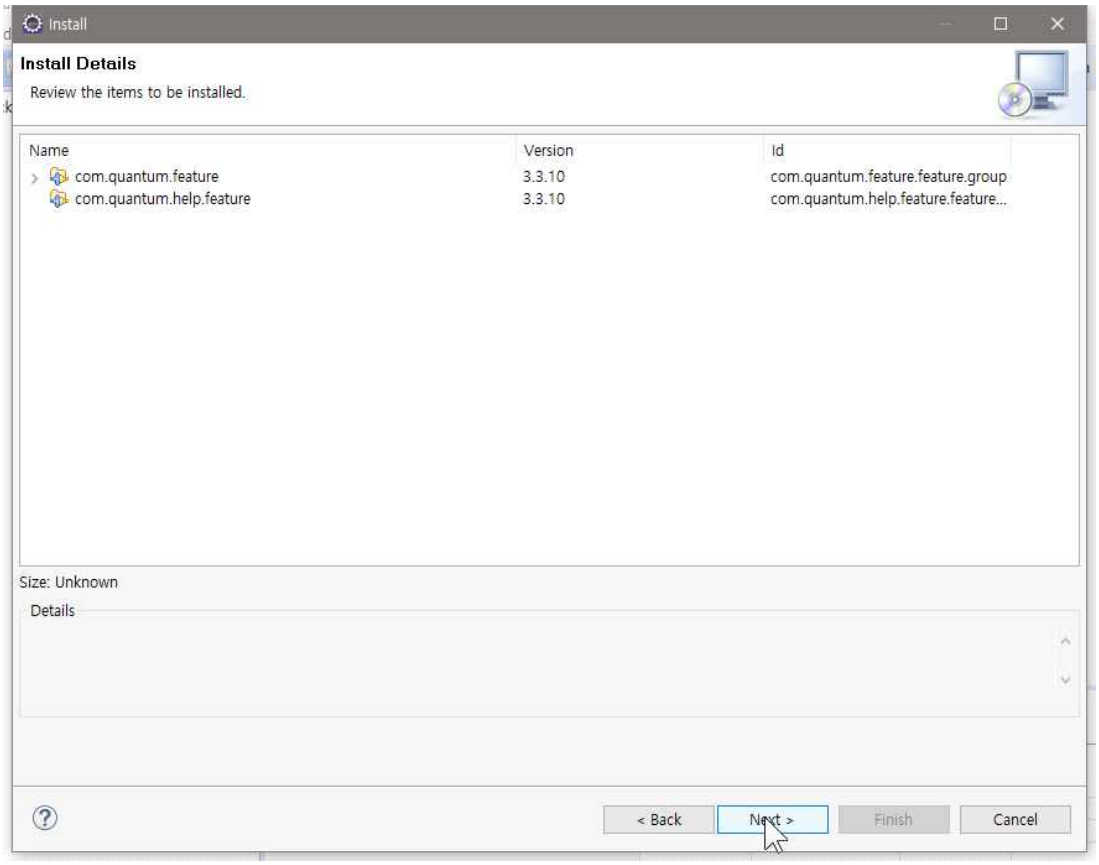
4) 위의 주소 입력



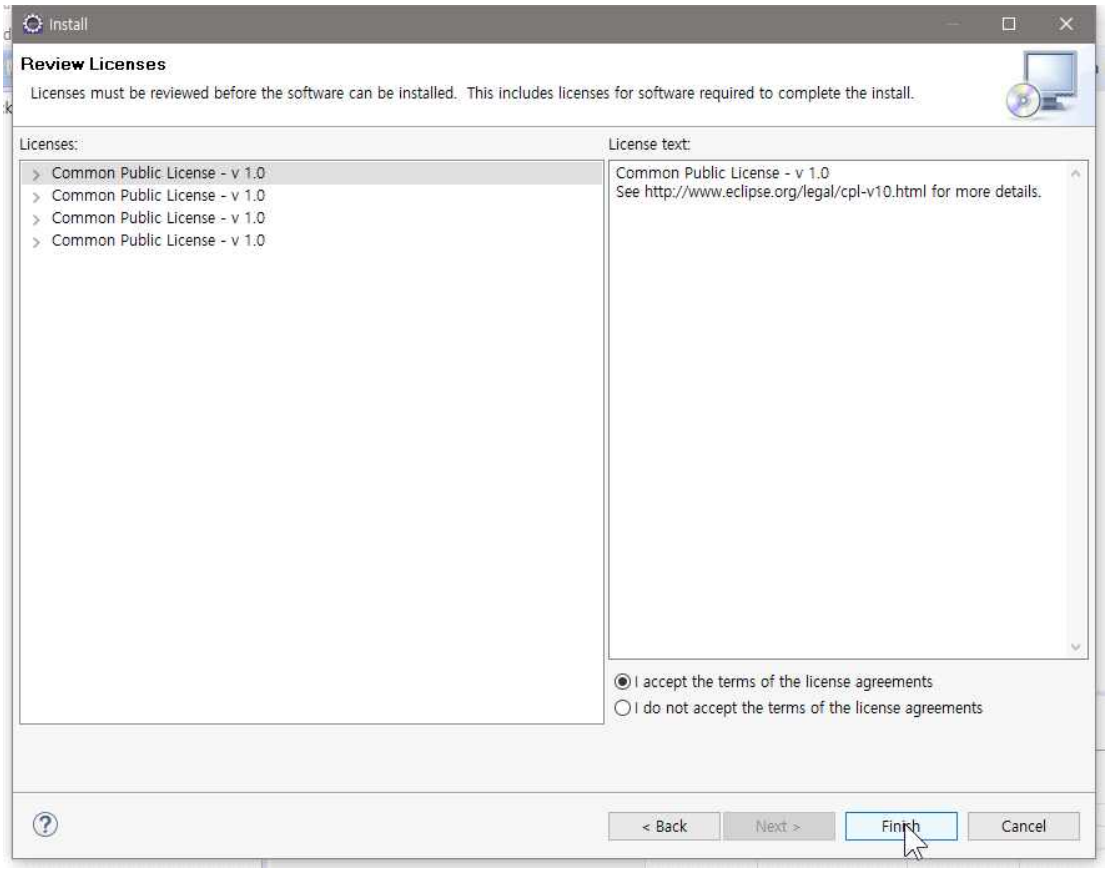
5) core 선택



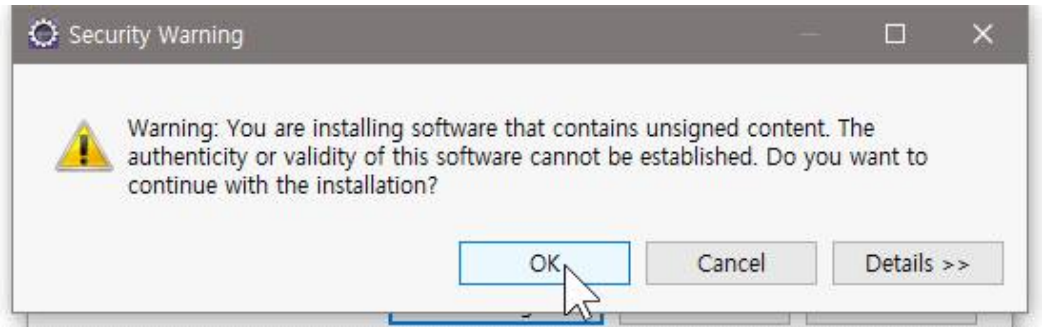
6) next 선택



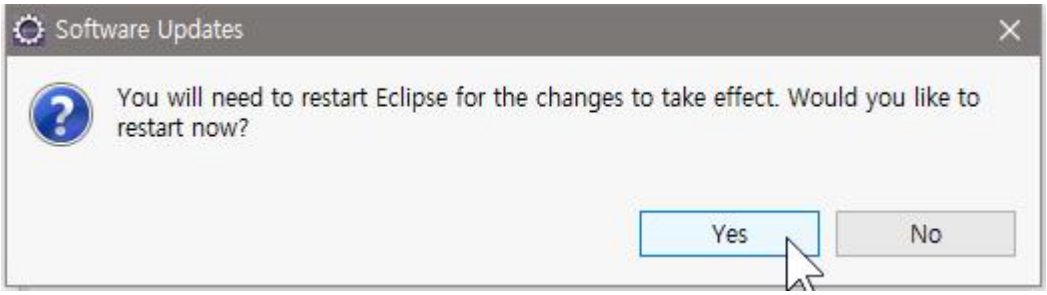
7) 라이센스 수락 및 finish 선택



8) ok 선택



9) yes 선택



1. 컬랙션(Collection) 클래스

■ 다수의 데이터를 쉽게 처리할 수 있는 표준화된 방법을 제공하는 클래스들

■ 핵심 인터페이스

종 류	중복 허용여부	인덱스
Set	X	없음
List	O	있음
Map(key와 value)	key : X, value : O	없음

종 류	특 징
List	순서가 있는 데이터의 집합 데이터의 중복을 허용 (쉽게 생각하면 무한배열)
	ArrayList , LinkedList, Vector 등
Set	순서를 유지하지 않는 데이터의 집합 데이터의 중복을 허용하지 않음
	HashSet, TreeSet 등
Map	키(key)와 값(value) 쌍으로 이루어진 데이터의 집합 순서는 유지되지 않음 키는 중복허용하지 않음 값은 중복 허용
	HashMap, TreeMap, HashTable 등

■ Set

예제

```
package ex;

import java.util.HashSet;

public class Ex01 {
    public static void main(String[] args) {
        HashSet hs = new HashSet();

        //SET은 중복값을 허용하지 않습니다.
        //즉, 일반 객체라면 주소값이 같다면 중복 허용 안함
        //String, Wrapper클래스들은 내용이 같다면 중복 허용하지 않습니다.
        hs.add("TEST");
        hs.add("TEST");
        hs.add(new String("TEST"));
        hs.add(4);

        Ex01 e = new Ex01();

        hs.add(e);
        hs.add(e);
        hs.add(new Ex01());

        System.out.println("hs의 길이는 ? : " + hs.size());

        System.out.println("hs는? : " + hs.toString());


    }
}
```

■ Map

예제

```
package ex;

import java.util.Hashtable;

public class Ex02 {
    public static void main(String[] args) {
        Hashtable ht = new Hashtable();

        ht.put("1", "TEST");
        ht.put("2", "Test");
        ht.put("3", "TESt");
        ht.put("4", "TEST");

        System.out.println("ht의 사이즈는? : " + ht.size());

        System.out.println("ht는? : " + ht);


    }
}
```

■ List

예제

```
import java.util.ArrayList;

public class Ex03 {
    public static void main(String[] args) {

        ArrayList list = new ArrayList();

        System.out.println("list의 사이즈는? : " + list.size());

        list.add("TEST1");
        list.add("TEST2");
        list.add("TEST3");
        list.add(new String("TEST4"));

        System.out.println("list의 사이즈는? : " + list.size());

        int size = list.size();

        for(int i = 0 ; i < size ; i++) {

            System.out.println((list.get(i)));
        }

        for(Object obj :list) {
            String str = (String)obj;
            System.out.println(str);
        }

    }
}
```

2. 제네릭(Generic)

■ 제네릭이 필요한 이유(5.0 이전의 컬랙션 클래스 사용)

예제 1 : C:\Java_Study\Generic\src\generic\GenericPrev.java

```
package generic;

import java.util.Vector;

public class GenericPrev {
    public static void main(String[] args) {
        Vector v = new Vector();

        v.add("TEST");
        v.add("TEST2");
        v.add("TEST3");

        for(Object obj : v) {
            String msg = (String)obj;
            GenericPrev.print(msg);
        }
    }

    public static void print(String s) {
        System.out.println("s의 값 : " + s);
    }
}
```

■ 제네릭을 사용한 경우

예제 2 : C:\Java_Study\Generic\src\generic\GenericPrev.java

```
package generic;

import java.util.Vector;

public class Generic5 {
    public static void main(String[] args) {
        Vector<String> v = new Vector<String>();

        v.add("TEST");
        v.add("TEST2");
        v.add("TEST3");
```

```
        for(String s : v) {
            GenericPrev.print(s);
        }
    }

    public static void print(String s) {
        System.out.println("s의 값 : " + s);
    }
}
```

■ toString() Overriding

예제 3 : C:\Java_Study\Generic\src\generic\GenericPrev.java

```
package generic;

import java.util.Vector;

public class Ge {
    public static void main(String[] args) {
        Vector v = new Vector();

        v.add(new String("TEST"));
        v.add("TEST2");
        v.add("TEST3");

        for(Object obj : v) {
            System.out.println(obj);
        }
    }
}
```

3. 랜덤 숫자 출력

■ Random 클래스의 사용

```
Random r1 = new Random();
int index = r1.nextInt(10)

0에서 9까지 정수중의 한 수를 반환함
```

■ Math.random() 메서드의 사용

```
int r = (int) (Math.random()*9);

0에서 9까지 정수중의 한 수를 반환함
```

예제 1

```
package random;

import java.util.Random;

public class RandomEx02 {
    public static void main(String[] args) {
        Random r = new Random();
        for(int i = 0; i < 20 ; i++) {
            System.out.println("Random 클래스 이용 : "+(r.nextInt(9)+1));
        }

        for(int j = 0 ; j < 20 ; j++) {
            System.out.println("Math.random() 이용 : " + ((int)(Math.random()*9)+1));
        }
    }
}
```

1. 프로그램 오류

■ 프로그램이 실행 중 어떤 원인에 의해서 오작동을 하거나 비정상적으로 종료되는 경우 오류라고 함

■ 컴파일 에러 : 컴파일시 발생하는 에러

■ 런타임 에러 : 실행도중에 발생하는 에러

■ 컴파일러는 문법적인 오류만 인식할 수 있음

■ 자바에서는 런타임에러를 에러(Error)와 예외(Exception)으로 구분함

오류구분	설 명
예외(Exception)	프로그램 코드에 의해서 수습될 수 있는 다소 미약한 오류
오류(Error)	프로그램 코드에 의해서 수습될 수 없는 심각한 오류

2. 예외(Exception)가 일어나는 상황

■ 정수를 0으로 나누는 경우

■ 배열의 index값이 음수 값을 가지거나, 크기를 벗어나는 경우

■ 부적절한 형변환

■ 입출력시 interrupt가 나타나는 경우

■ 입출력하기 위한 파일이 존재하지 않는 경우

■ 메서드 호출시

3. 예외처리의 목적

■ 예외의 발생으로 인한 실행 중인 프로그램의 갑작스런 비정상 종료를 막고, 정상적인 실행상태를 유지할 수 있도록 하는 것

4. 예외처리구문(try ~ catch)

```
try{
     //예외가 발생할 가능성이 있는 코드(문장)

    }catch(예외타입1 매개변수1 ){

      //예외발생시 처리할 코드 (예외가 발생할 때만 실행된다.)

    }catch(예외타입2 매개변수2 ){

      //예외발생시 처리할 코드

    } finally{
      //예외에 상관없이 실행 할 코드
    }
```

■ 익셉션이 발생하는 경우

예제 1

```
class ExceptionEx01 {
      public static void main(String args[]) {
            int number = 50;
            int result = 0;

            for(int i=0; i < 10; i++) {
                  result = number / (int)(Math.random() * 5);
                  System.out.println(result);
            }
      }
}
```

1) 문법상의 오류가 없기 때문에 컴파일시 에러 발생하지 않음

2) 실행도중 정수를 0으로 나눌경우가 생길 수 있고 익셉션 발생

6/2=3 -> 3*2=6

x/0=y -> y*0=x

■ 익셉션 처리

예제 2

```
class ExceptionEx02 {
	public static void main(String args[]) {
		int number = 50;
		int result = 0;

		for(int i=0; i < 10; i++) {
			try {
				result = number / (int)(Math.random() * 5);
				System.out.println(result);
			} catch (ArithmeticException e)	{
				// ArithmeticException이 발생하면 수행된다.
				System.out.println("Exception 발생");
			} // try-catch의 마지막
		} // for의 마지막
	}
}
```

■ 익셉션 구문 실행 순서

예제 3

```
class ExceptionEx03 {
	public static void main(String args[]) {
			System.out.println(1);
			System.out.println(2);
			try {
				System.out.println(3);
				System.out.println(4);
			} catch (Exception e)	{
				System.out.println(5);
			} // try-catch의 마지막
			System.out.println(6);
	}	// main메서드의 마지막
}
```

■ 익셉션 구문 실행 순서2

예제 4

```
class ExceptionEx04 {
	public static void main(String args[]) {
			System.out.println(1);
			System.out.println(2);
			try {
				System.out.println(3);
				System.out.println(0/0); // ArithmeticException을 발생시킨다.
				System.out.println(4);  // 실행되지 않는다.
			} catch (ArithmeticException ae)	{
				System.out.println(5);
			}	// try-catch의 마지막
			System.out.println(6);
	}	// main메서드의 마지막
}
```

5. Runtime 익셉션과 그 외의 익셉션

■ runtime 익셉션은 컴파일시 에러 발생하지 않음

■ 그 외 익셉션은 컴파일시 에러 발생함

예제 5

```
class ExceptionEx05{
	public static void main(String[] args)
	{
		throw new Exception();		// Exception을 강제로 발생시킨다.
	}
}
```

■ 예외 강제 발생

```
throw new 발생시킬 예외객체 생성자 / throw 예외객체
```

예제 6

```
class ExceptionEx06 {
	public static void main(String[] args)
	{
		throw new RuntimeException();	// RuntimeException을 강제로 발생시킨다.
	}
}
```

■ 예외처리 순서

예제 7

```
class ExceptionEx07 {
	public static void main(String args[]) {
			System.out.println(1);
			System.out.println(2);
			try {
				System.out.println(3);
				System.out.println(0/0); // ArithmeticException을 발생시킨다.
				System.out.println(4);  // 실행되지 않는다.
			} catch (Exception e)	{// ArithmeticException대신 Exception을 사용.
				System.out.println(5);
			}	// try-catch의 마지막
			System.out.println(6);
	}	// main메서드의 마지막
}
```

예제 8

```
class ExceptionEx08 {
	public static void main(String args[]) {
			System.out.println(1);
			System.out.println(2);
			try {
				System.out.println(3);
				System.out.println(0/0);	// ArithmeticException을 발생시킨다.
				System.out.println(4);	// 실행되지 않는다.
			} catch (ArithmeticException ae)	{
				if (ae instanceof ArithmeticException)
					System.out.println("true");
				System.out.println("ArithmeticException");
			} catch (Exception e)	{
				System.out.println("Exception");
			}	// try-catch의 마지막
			System.out.println(6);
	}	// main메서드의 마지막
}
```

■ finally를 포함한 처리순서

예제 9

```
class ExceptionEx09 {
      int[] ss;
    public ExceptionEx09() {

        ss = new int[3];// 속성(멤버필드) 초기화

    }

    public void prog(){

       for(int i=0; i<=ss.length; i++){
           System.out.println("for문의 시작 " + i + "번째");
           try {
           System.out.println(ss[i]);

           } catch (Exception e) {
               System.out.println("Exception 발생" + e);
              return;
           //}
               //return;
         }finally{
              System.out.println("finally 영역");
         }
            System.out.println("for문의 끝" + i + "번째");
       }
    }

    public static void main(String[] args) {
        ExceptionEx09 ref = new ExceptionEx09();
        ref.prog();
        System.out.println("프로그램 끝!");
    }
}
```

■ Exception 클래스의 주요 메서드

메서드명	설 명
printStackTrace()	예외 발생 당시의 호출스택(Call Stack)에 있었던 메서드의 정보와 예외 메세지를 화면에 출력
getMessage()	발생한 예외클래스의 인스턴스에 저장된 예외메세지를 얻을 수 있음

6. throws 예약어

```
[접근제한] 반환형 메서드명([인자1, 인자2...]) throws 예외클래스1, 예외클래스2....
```

■ 자신을 호출한 메서드로 익셉션 처리를 위임함

예제 10

```
public class ExceptionEx10 {
    private static void test() throws Exception {
        System.out.println(6/0);
    }

    public static void main(String[] args) {
        try {
            test(); //try catch문으로 예외처리 해야함
        } catch (Exception e) {
            System.out.println("예외 발생");
        }
    }
}
```

15. JDBC란?

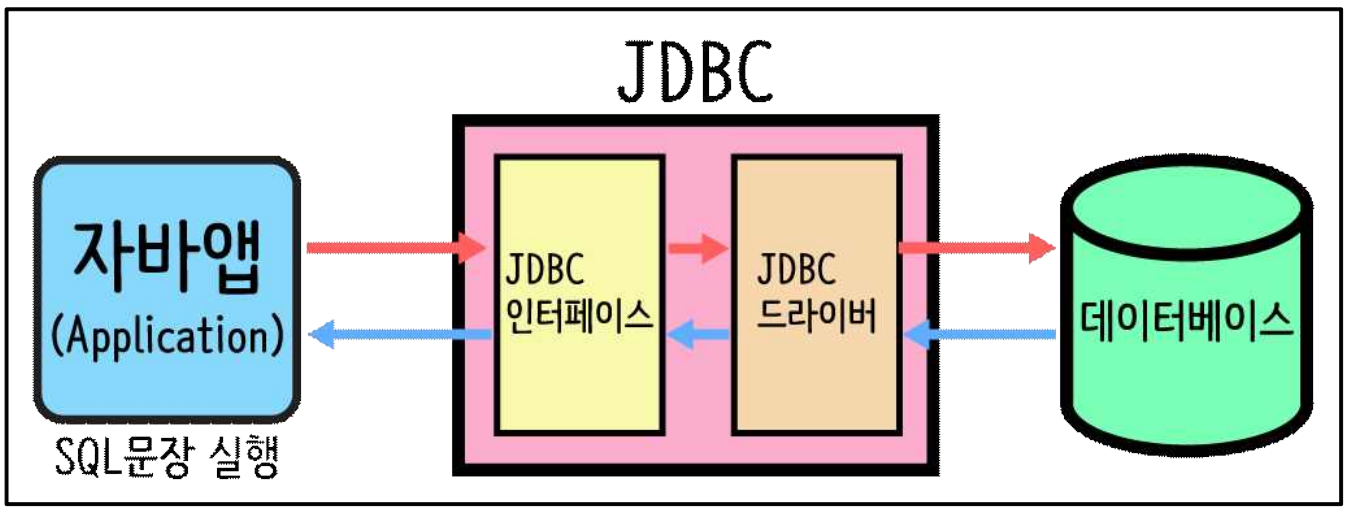
■ JDBC(Java Database Connectitivy)를 이용하여 데이터베이스에 접근하여 각종 SQL문을 수행할 수 있도록 제공하는 API를 말한다.

■ DBMS의 종류가 다양하고, DBMS마다 구조와 특정이 다르기 때문에 특정 DBMS에 맞게 API를 개발하는 것은 어렵다.

자바에서는 모든 DBMS에서 공통적으로 사용 할 수 있는 인터페이스와 클래스로 구성된 JDBC를 개발하고, 실제 각 DBMS에 적합한 구현클래스는 각 DBMS의 벤더(bender)에게 구현토록 하도록 했으며, 각 벤더에서 구현한 구현클래스들 JDBC드라이버라고 한다.

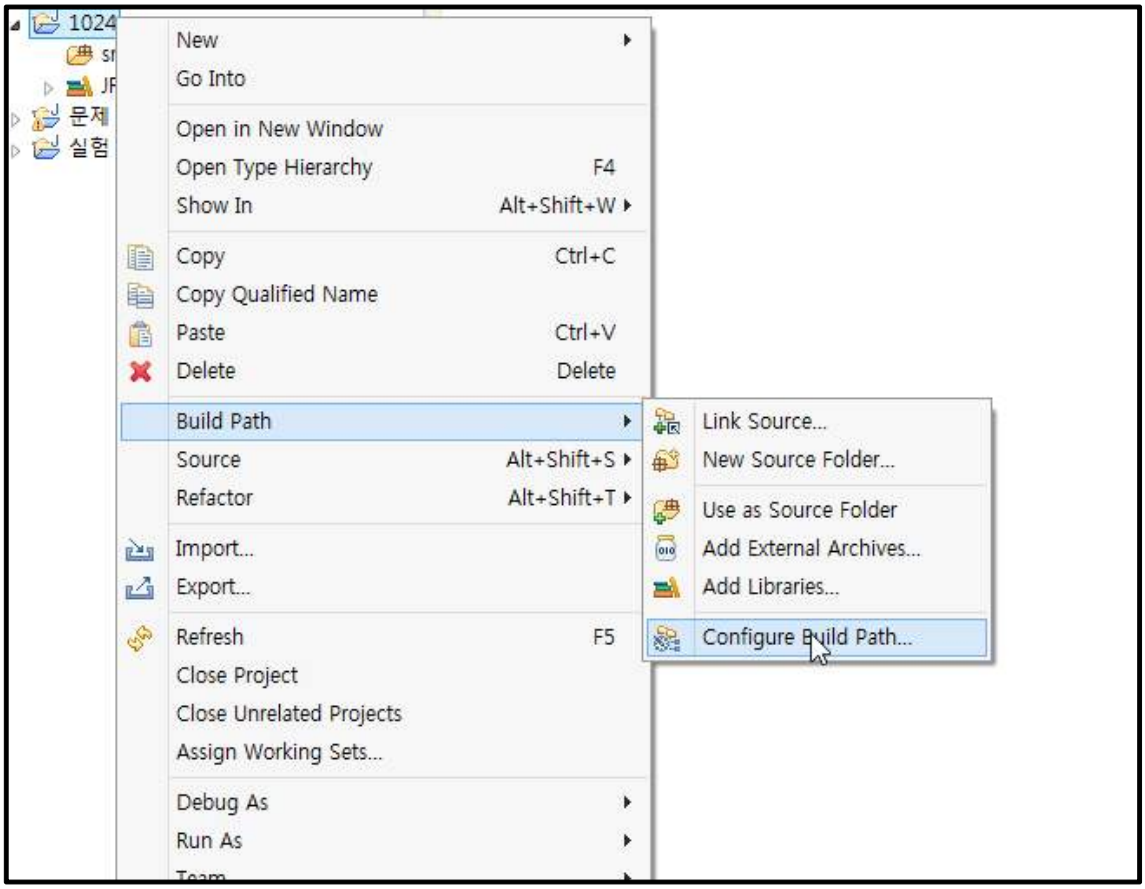
■ 자바 개발자는 특정 DBMS를 선택하고, JDBC 코딩을 위해 DBMS에 적합한 JDBC 드라이브를 선택하기만 하면 DBMS의 종류에 상관없이 동일한 방법으로 데이터베이스 관련 작업을 수행할 수 있다.

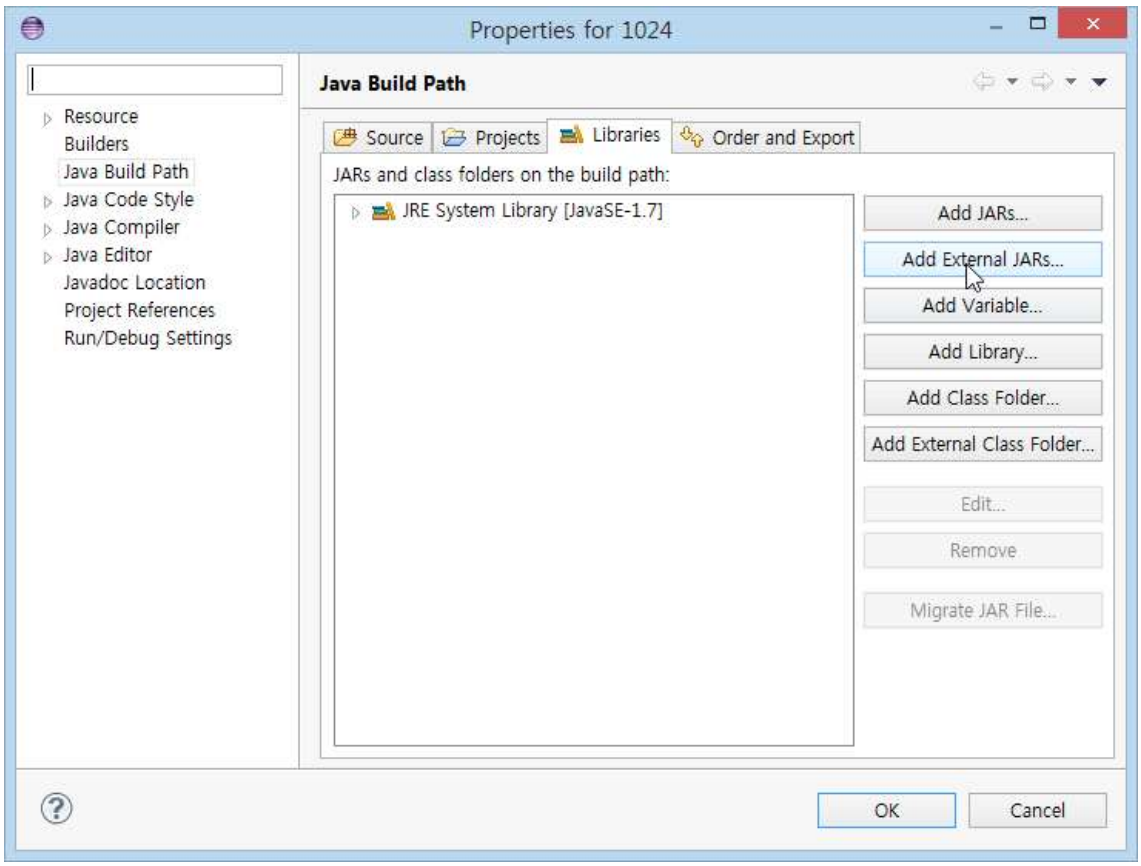
JDBC의 구조

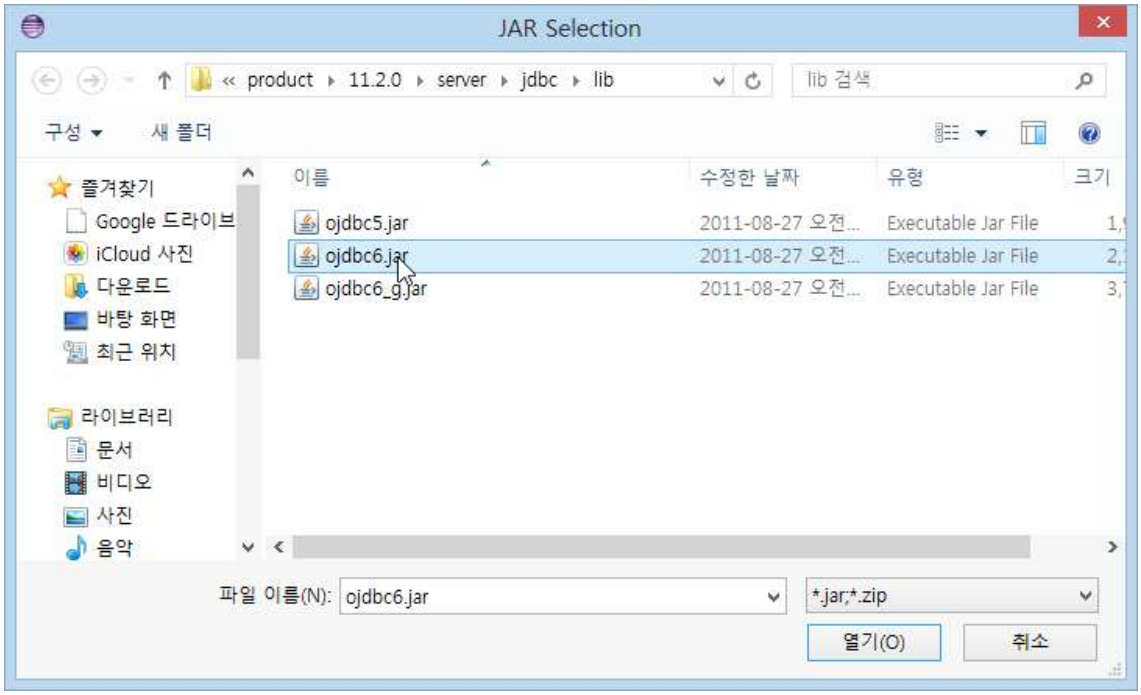


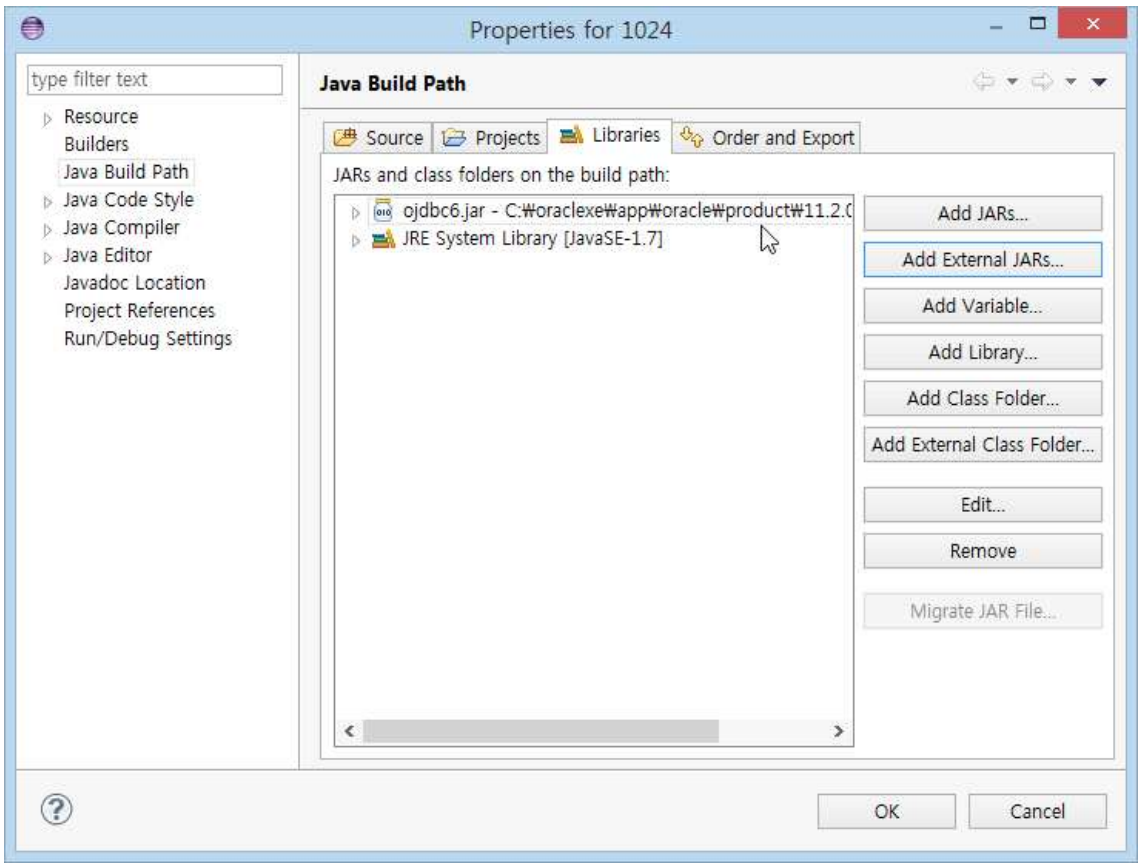
JDBC 코딩하기

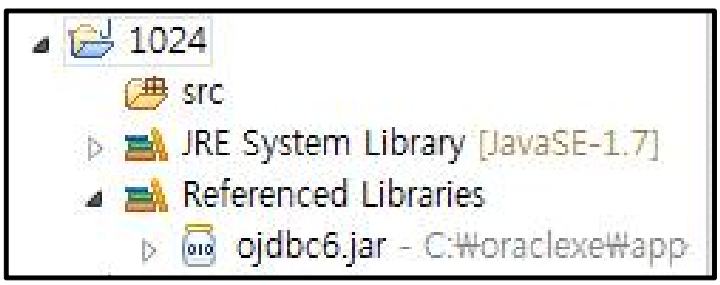
■ JDBC 드라이브 등록하기






JAR Selection
« product ▸ 11.2.0 ▸ server ▸ jdbc ▸ lib
lib 검색
구성
새 폴더
즐겨찾기
Google 드라이브
iCloud 사진
다운로드
바탕 화면
최근 위치
라이브러리
문서
비디오
사진
음악
이름
수정한 날짜
유형
크기
ojdbc5.jar
ojdbc6.jar
ojdbc6_g.jar
2011-08-27 오전...
Executable Jar File
파일 이름(N): ojdbc6.jar
.jar;.zip
열기(O)
취소


Properties for 1024
type filter text
Resource
Builders
Java Build Path
Java Code Style
Java Compiler
Java Editor
Javadoc Location
Project References
Run/Debug Settings
Java Build Path
Source
Projects
Libraries
Order and Export
JARs and class folders on the build path:
ojdbc6.jar - C:₩oraclexe₩app₩oracle₩product₩11.2.0
JRE System Library [JavaSE-1.7]
Add JARs...
Add External JARs...
Add Variable...
Add Library...
Add Class Folder...
Add External Class Folder...
Edit...
Remove
Migrate JAR File...
OK
Cancel


1024
src
JRE System Library [JavaSE-1.7]
Referenced Libraries
ojdbc6.jar - C:₩oraclexe₩app

오라클 드라이버 경로 :

C:\oraclexe\app\oracle\product\11.2.0\server\jdbc\lib

■ JDBC API의 주요 클래스

▮Connection : 응용프로그램과 DB사이의 연결을 담당하는 객체

▮Statement : SQL을 DB에 전송하고 실행결과를 반환한다.

▮ResultSet : select 쿼리문의 실행인 결과집합을 가지고 있으며, 결과집합내의 현재 row를 가르키는 Cursor를 가지고 있다.

■ JDBC 코딩의 절차

▮JDBC 드라이브를 메모리에 로드하기

▮Connection 객체 얻기

▮Statement 객체 얻기

▮SQL 실행하기

▮SQL 실행결과 처리하기

JDBC 코딩예제 (insert 문)

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

public class JDBCDemo1 {

    public static void main(String[] args) {

        Connection con = null;
        Statement stmt = null;

        try{
            //jdbc 드라이브 메모리에 로드하기
            Class.forName("oracle.jdbc.driver.OracleDriver");
```

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

"test", “1111”);

            //Statement 객체 얻기
            stmt = con.createStatement();

            //SQL 작성하기
            StringBuffer sql = new StringBuffer();
            sql.append("insert into department ");
            sql.append("values (203, '제어계측공학과', 200, '7호관')");

            //쿼리 실행시키기
            int result = stmt.executeUpdate(sql.toString());

            System.out.println(result + " 개의 행이 추가되었습니다");

        }catch(ClassNotFoundException e){
            e.printStackTrace();
        }catch(SQLException e){
            e.printStackTrace();
        }finally{
            //사용한 자원 반납 처리
            try{if(stmt != null)stmt.close();}catch(SQLException e){}
            try{if(con != null)con.close();}catch(SQLException e){}
        }
    }
}
```

DriverManager와 Connection

■ DriverManager 클래스는 static 메소드인 getConnection()메소드를 이용해서 DB와 연결된 Connection 객체를 반환한다.

■ getConnection(String url, String user, String password)

url : "oracle:jdbc:thin:@ip:port:oracle_sid"

user : 오라클의 사용자 명

password : 사용자 비밀번호

■ Connection 객체는 DB와의 연결을 담당하는 객체다. Connection객체를 이용해서 SQL을 전송•실행할 수 있는 Statement객체를 얻을 수 있다.

■Connection의 주요 메소드

반환형	매소드	설명
void	close()	Connection객체를 해제한다.
	commit()	트랜잭션으로 설정된 모든 자원을 db에 반영한다.
Statement	createStatement()	SQL을 전송할 수 있는 Statement 객체를 반환한다.
PreparedStatement	prepareStatement(String sql)	SQL을 전송할 수 있는 PreparedStatement 객체를 반환한다.
resultSet	executeQuery(String sql)	SQL이 select문인 경우 사용한다. 수행결과로 ResultSet을 반환한다.
void	rollback()	현재 트랜잭션에 설정내의 모든 작업을 되돌린다.
	rollback(Savepointsavepoint)	Savepoint설정이후의 모든 작업을 되돌린다.
void	setSavepoint(String name)	현재의 트랜잭션내에 savepoint를 설정한다.
void	setAutoCommit(boolean value)	auto-commit기능을 설정한다.

Statement

■ SQL을 DB로 전송하고 실행결과를 반환한다.

■ Statement의 주요 메소드

반환형	매소드	설명
void	addBatch(String sql)	Statement객체에 SQL 문을 추가한다. SQL의 일괄처리 할 때 사용된다.
	clearBatch()	Statement객체의 모든 SQL문을 비운다.
	close()	Statement 객체를 해제한다.
boolean	execute(String sql)	SQL문을 수행하고, 수행결과가 ResulSet객체인 경우 true를 반환하고, 갱신된 숫자를 반환하면 false를 반환한다.
int[]	executeBatch()	Statement객체에 추가된 모든 SQL 문을 일괄처리한다. 일괄처리된 각각의 SQL문에 대한 결과값을 int[]로 반환하다.
resultSet	executeQuery(String sql)	SQL이 select문인 경우 사용한다. 수행결과로 ResultSet을 반환한다.
int	executeUpdate(String sql)	SQL문이 insert, delete, update, create, drop인 경우 사용한다.
ResultSet	getResultSet()	ResultSet객체를 반환한다.

JDBC 예제 (update 문)

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

public class JDBCDemo2 {

	public static void main(String[] args) {

		StringBuffer sql = new StringBuffer();
		sql.append("update department ");
		sql.append("set dname = '생명공학과' ");
		sql.append("where deptno = 209 ");

		Connection con = null;
```

```
		Statement stmt = null;

		try{
			Class.forName("oracle.jdbc.driver.OracleDriver");

			con
			   =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl",
					"scott", "tiger");

			stmt = con.createStatement();

			    //update문은 executeUpdate()를 사용한다.
			int i = stmt.executeUpdate(sql.toString());
			System.out.println(i + " 개의 행이 변경되었습니다.");

		}catch(ClassNotFoundException e){
			e.printStackTrace();
		}catch(SQLException e){
			e.printStackTrace();
		}finally{
			//Connection, Statement 자원반납처리...
			try{
				if(stmt != null)stmt.close();
			}catch(SQLException e){}
			try{
				if(con != null)con.close();
			}catch(SQLException e){}
		}
	}
}
```

JDBC 예제 (delete 문)

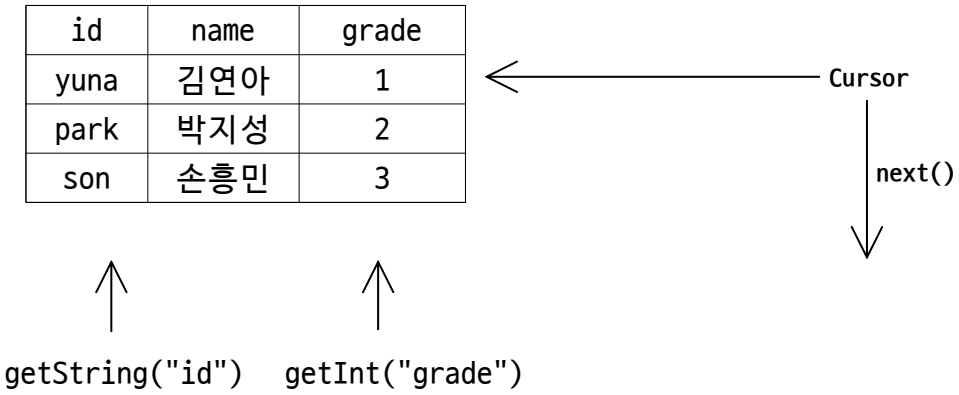
```
public class JDBCDemo3 {

    public static void main(String[] args){
        StringBuffer sql = new StringBuffer();
        sql.append("delete department ");
        sql.append("where dname = '제어계측공학과' ");

        Connection con = null;
        Statement stmt = null;
        try{

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

                    con
                       =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl",
                                    "scott", "tiger");

                    stmt = con.createStatement();

            //update문ㅇ
                    int i = stmt.executeUpdate(sql.toString());
                    System.out.println(i + " 개의 행이 삭제되었습니다.");

        }catch(ClassNotFoundException e){
            e.printStackTrace();
        }catch(SQLException e){
            e.printStackTrace();
        }finally{
            //사용한 자원 반납 처리
            try{if(stmt != null)stmt.close();}catch(SQLException e){}
            try{if(con != null)con.close();}catch(SQLException e){}
        }
    }
}
```

ResultSet

■ select 쿼리의 수행결과로 얻어진 결과집합을 추상화한 것이다.

■ ResultSet은 결과집합에서 data를 가져올 수 있는 다양한 메소드와 Cursor를 포함하고 있다.

■ ResultSet의 구조



반환형	매소드	설명
boolean	first()	ResultSet에서 커서를 첫번째 row로 이동한다. row가 존재하면 true, 존재하지 않으면 false를 반환한다.
	last()	ResultSet에서 커서를 마지막 row로 이동한다. row가 존재하면 true, 존재하지 않으면 false를 반환한다.
	next()	ResultSet에서 커서를 다음 row로 이동한다. row가 존재하면 true, 존재하지 않으면 false를 반환한다.
	previous()	ResultSet에서 커서를 이전 row로 이동한다. row가 존재하면 true, 존재하지 않으면 false를 반환한다.
void	close	ResultSet 객체를 해제한다.
int	getInt(int columnindex)	ResultSet객체의 커서가 가르키는 현재 row에서 columnIndex에 해당하는 int값을 반환한다.
	getInt(String columnName)	ResultSet객체의 커서가 가르키는 현재 row에서 columnName에 해당하는 int값을 반환한다.
String ResultSet	getString(int columnIndex)	ResultSet객체의 커서가 가르키는 현재 row에서 columnIndex에 해당하는 String값을 반환한다.
	getString(String columnName)	ResultSet객체의 커서가 가르키는 현재 row에서 columnName에 해당하는 String값을 반환한다.

■ ResultSet의 주요 메소드

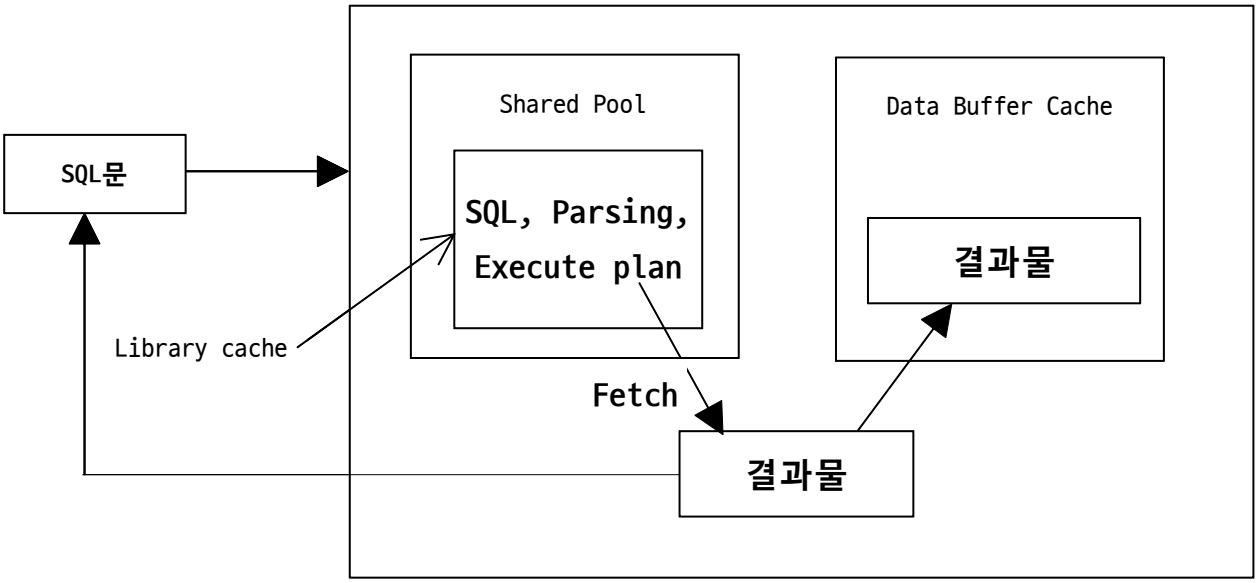
※ResultSet내에는 오라클의 자료형마다. 두 개씩의 getXXX()메소드가 존재

JDBC 예제 (select 문)

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


public class JDBCDemo4 {
    public static void main(String[] args){

        StringBuffer sql = new StringBuffer();
             sql.append("select deptno, dname, college, loc ");
             sql.append("from department ");

             Connection con = null;
             Statement stmt = null;
             //select쿼리의 수행결과집합과 결과집합에서 데이타를
             //추출할 수 있는 메소드를 가지고 있는 객체
             ResultSet rs = null;

             try{
                    Class.forName("oracle.jdbc.driver.OracleDriver");
                    //thin Type jdbc 드라이버 : 자체 네트웍 protocol가지고 있는
                    //jdbc 드라이버...
                    con
                       =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl",
                                                              "scott", "tiger");
                    stmt = con.createStatement();

                    //실행하고자하는 sql이 select 쿼리인 경우
                    //쿼리실행의 결과집합를 리턴 받을 수 있는
                    //executeQuery()를 사용한다.
                    rs = stmt.executeQuery(sql.toString());
```

```
                    //resultset객체로부터 데이타 추출하기
                    //resultset의 next()메소드는 커서를 다음행으로 이동시킴
                    //이동된 위치에 row가 존재하면 true, 존재하지 않으면
                    //false를 반환한다.

                    //rs.next()가 참인 동안, 즉 결과집합에 행이 존재하는 동안
                    //ResultSet으로부터 데이타 추출....
                    while(rs.next()){
                          //커서가 위치한 row의 각각의 column에서 값 추출하기
                          //resultset의 getXXX(컬럼위치), getXXX(컬럼이름)
                          //을 사용해서 column의 값 추출
                          int i = rs.getInt(1); //rs.getInt("deptno");
                          String s1 = rs.getString("dname");
                          int j = rs.getInt("colleage");
                          String s2 = rs.getString("loc");

                          System.out.println(i + "\t" + s1 + "\t" + j + "\t" +
s2);

                    }

             }catch(ClassNotFoundException e){
                    e.printStackTrace();
             }catch(SQLException e){
                    e.printStackTrace();
             }finally{
                    try{if(rs != null)rs.close();}catch(SQLException e){}
                    try{if(stmt != null)stmt.close();}catch(SQLException e){}
                    try{if(con != null)con.close();}catch(SQLException e){}
             }

       }
}
```

■ 오라클의 메모리 구조 및 SQL문 실행 절차



▮ 오라클로 전송된 SQL문은 Parsing -> execute plan -> fetch의 작업을 한 이후에 SQL문의 수행결과를 Data Buffer Cache에 저장한다.

▮ 똑같은 SQL문이 전송되면 Library cache에 저장된 SQL의 Parsing 결과와 execute plan의 그대로 사용하게 됨으로 수행속도를 향상 시킬 수 있다.

■ PreparedStatement는 SQL의 형태는 동일하나 조건이나 변수 값이 다른 문장을 바인딩변수를 사용해서 변수 처리함으로써 항상 동일한 SQL문을 동일하게 처리하게 할 수 있다.

■ PreparedStatement객체의 생성 및 바인딩 변수의 사용

```
//바인딩변수는 실제값으로 대체될 부분에 사용한다.
String sql = "insert into department values(?, ?, ?, ?)";
PreparedStatement pstmt = con.prepareStatement(sql);
//바인딩변수의 개수만큼 순서대로 해당 변수와 대체될 값을 지정해준다.
pstmt.setInt(1, 203);
pstmt.setString(2, "생명공학과");
pstmt.setInt(3, 200);
pstmt.setString(4, "6호관");
```

※바인딩변수는 컬럼명에는 절대 사용할 수 없다.

■ PreparedStatement는 PreparedStatement의 바인딩변수에 값을 지정해주는 setXXX()메소드를 제공한다.

ConneUtil 클래스 (Connection을 반환하는 메소드를 제공하는 클래스)

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

public class ConnUtil {

	static{
		try{
			Class.forName("oracle.jdbc.driver.OracleDriver");
		}catch(ClassNotFoundException e){
			e.printStackTrace();
		}
	}

	public static Connection getCon()throws SQLException{

		return

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

"scott", "tiger");

	}
}
```

PreparedStatement 예제 (select 문)

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class JDBCDemo8 {

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

```
            StringBuffer sql = new StringBuffer();
            sql.append("select a.name, a.profno, a.position, b.dname ");
            sql.append("from professor a, department b ");
            sql.append("where a.deptno = b.deptno ");
            sql.append("and a.deptno = ? ");

            Connection con = null;
            PreparedStatement pstmt = null;
            ResultSet rs = null;

            try{
                  con = ConnUtil.getCon();
                  pstmt = con.prepareStatement(sql.toString());

                  //?(바인딩변수)에 대체할 실제값 지정
                  pstmt.setInt(1, 101);

                  //쿼리 실행시켜서 결과집합 얻기
                  rs = pstmt.executeQuery();

                  //결과집합에서 값 추출하기
                  while(rs.next()){
                        System.out.print(rs.getString("name") + "\t");
                        System.out.print(rs.getInt("profno") + "\t");
                        System.out.print(rs.getString("name") + "\t");
                        System.out.println(rs.getString("position") + "\t");
                  }
            }catch(SQLException e){
                  e.printStackTrace();
            }finally{
                  try{if(pstmt != null)pstmt.close();}catch(SQLException e){}
                  try{if(con != null)con.close();}catch(SQLException e){}
            }

      }
}
```

PreparedStatement 예제 (update 문)

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

public class JDBCDemo7 {

	public static void main(String[] args) {

		StringBuffer sql = new StringBuffer();
		sql.append("update professor ");
		sql.append("set sal = ? " );
		sql.append("where name = ? ");

		Connection con = null;
		PreparedStatement pstmt = null;

		try{
			con = ConnUtil.getCon();

			pstmt = con.prepareStatement(sql.toString());
			pstmt.setInt(1, 500);
			pstmt.setString(2, "홍길동");

			int i =  pstmt.executeUpdate();

			System.out.println(i + "개의 행이 변경되었습니다.");


		}catch(SQLException e){
			e.printStackTrace();
		}finally{
			try{if(pstmt != null)pstmt.close();}catch(SQLException e){}
			try{if(con != null)con.close();}catch(SQLException e){}
		}
	}
}
```

PreparedStatement 예제 (insert 문)

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

public class JDBCDemo6 {

	public static void main(String[] args) {

		//preparedStatement를 사용하는 경우
		//sql쿼리에서 실제값으로 대체될 부분을 ?로 처리한다.
		//?는 db에서 sql실행시에 실제값으로 대체된다.
		StringBuffer sql = new StringBuffer();
		sql.append("insert into professor ");
		sql.append("values(?, ?, ?, ?, ?, sysdate, ?, ?)");

		Connection con = null;
		PreparedStatement pstmt = null;

		try{
			con = ConnUtil.getCon();

			//PreparedStatement 객체 얻기
			pstmt = con.prepareStatement(sql.toString());

			//파라미터 셋팅
			//쿼리의 ?(바인딩 변수)에 대체될 실제값 지정하기
			pstmt.setInt(1, 9920);
			pstmt.setString(2, "홍길동");
			pstmt.setString(3, "gildong");
			pstmt.setString(4, "교수");
			pstmt.setInt(5, 450);
			pstmt.setInt(6, 40);
			pstmt.setInt(7, 201);

			//쿼리 실행
			int i = pstmt.executeUpdate();
			System.out.println(i + " 개의 행이 추가되었습니다.");

		}catch(SQLException e){
```

```
                    e.printStackTrace();
             }finally{
                    try{if(pstmt != null)pstmt.close();}catch(SQLException e){}
                    try{if(con != null)con.close();}catch(SQLException e){}
             }
      }
}
```

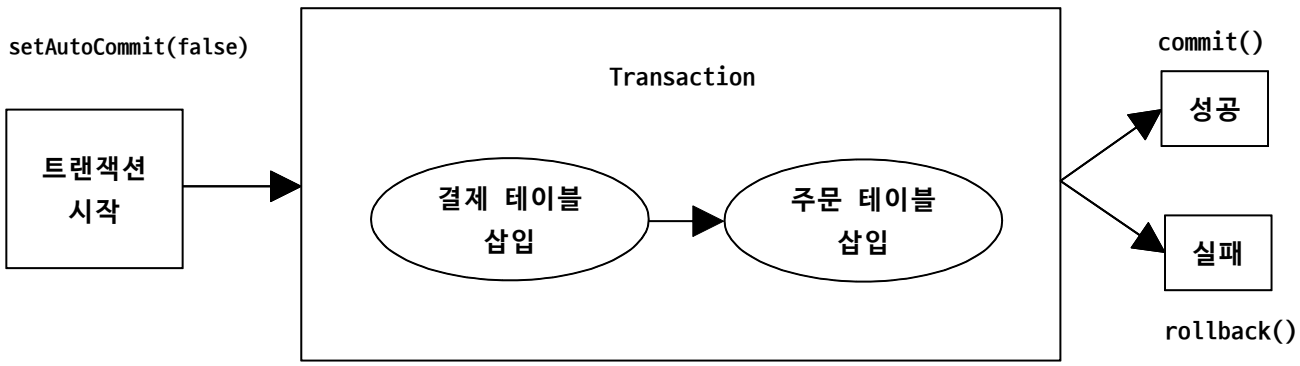
Transaction

■ Transaction 여러 개의 작업을 하나의 논리적인 작업 단위로 묶어 주는 것을 말한다. 하나의 작업 단위내의 전체 작업들이 모두 올바르게 수행되거나 , 또는 전체 작업이 모두 수행되지 않도록 한다.

■ Connection의 Transaction 관련 메소드

반환형	매소드	설명
void	commit()	트랜잭션으로 설정된 모든 자원을 db에 반영한다.
void	rollback()	현재 트랜잭션에 설정내의 모든 작업을 되돌린다.
	rollback(Savepoint savepoint)	Savepoint설정이후의 모든 작업을 되돌린다.
void	setSavepoint(String name)	현재의 트랜잭션내에 savepoint를 설정한다.
void	setAutoCommit(boolean value)	auto-commit기능을 설정한다.

■ 응용 프로그램에서의 Transaction 처리



Transaction 예제 1

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

import ex2.ConnUtil;

public class JDBCDemo17 {

	public static void main(String[] args) {

		//transaction : 논리적인 작업단위
		//insert, delete, update등의 작업들을
		//하나의 논리적인 작업단위로 묶어서
		//쿼리실행시 모든 작업이 정상처리된경우에는
		//commit을 실행해서 db에 반영하고,
		//쿼리실행중 하나라도 정상처리되지 않은 경우
		//rollback을 실행해서 작업단위내의 모든 작업을 취소한다.

		StringBuffer sql1 = new StringBuffer();
		sql1.append("insert into department ");
		sql1.append("values(?,?,?,?) ");

		StringBuffer sql2 = new StringBuffer();
		sql2.append("update department ");
		sql2.append("set dname = ?, loc = ? ");
		sql2.append("where deptno = ?");

		Connection con = null;
		PreparedStatement pstmt = null;

		try{
			////////////   transaction 시작  ////////////////////
			//하나의 논리적인 작업단위 시작
			con = ConnUtil.getCon();

			//autoCommit기능 비활성화 시키기
			con.setAutoCommit(false);
```

```
                //-------------- 1번 작업 시작 -------------------//
                pstmt = con.prepareStatement(sql1.toString());
                pstmt.setInt(1, 255);
                pstmt.setString(2, "핵물리학과");
                pstmt.setInt(3, 200);
                pstmt.setString(4, "9호관");

                pstmt.executeUpdate();
                //-------------- 1번 작업 종료 -------------------//

                //-------------- 2번 작업 시작 -------------------//
                pstmt = con.prepareStatement(sql2.toString());
                pstmt.setString(1, "생명공학과");
                pstmt.setString(2, "8호관");
                pstmt.setInt(3, 255);

                pstmt.executeUpdate();
                //-------------- 2번 작업 종료 -------------------//

                //쿼리가 정상적으로 실행된 경우
                //db에 반영
                con.commit();
                System.out.println("db에 반영됨.........");

            }catch(SQLException e){
                try {
                    con.rollback();
                    System.out.println("db 반영이 취소됨.........");
                } catch (SQLException e1) {
                    e1.printStackTrace();
                }
            }finally{
                try{if(pstmt != null)pstmt.close();}catch(SQLException e){}
                try{if(con != null)con.close();}catch(SQLException e){}
                ////////////////// transaction 종료   ////////////////////
            }
        }
}
```