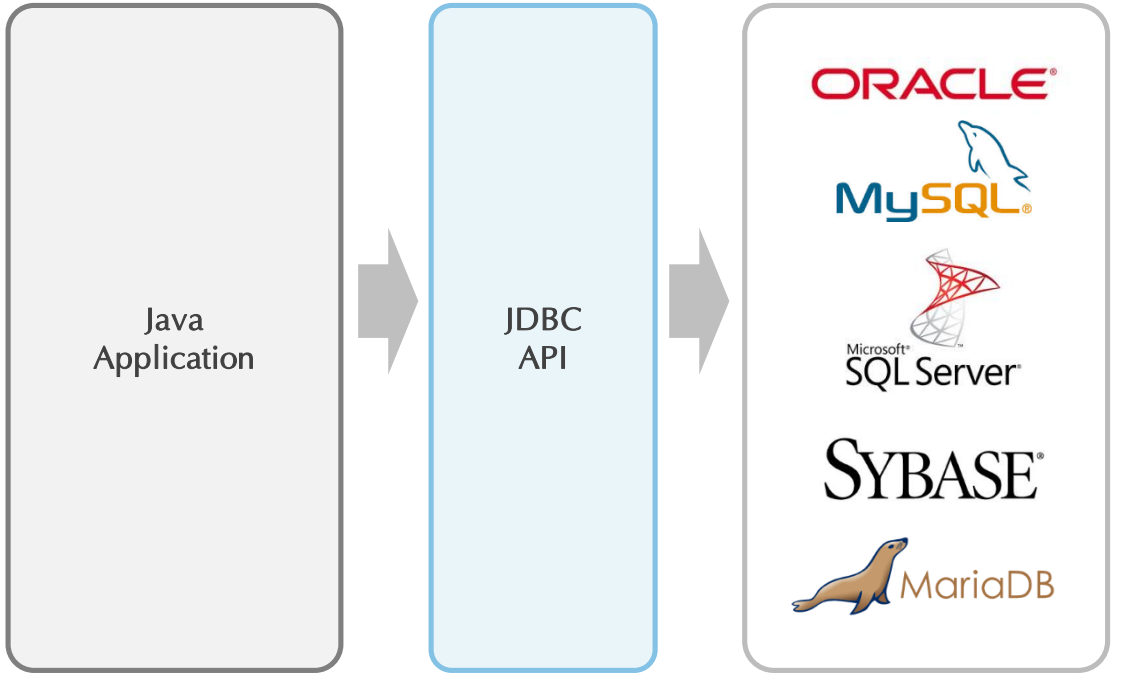
JDBC (ver3.4.0)

목차(Table of Contents)

1. JDBC 소개

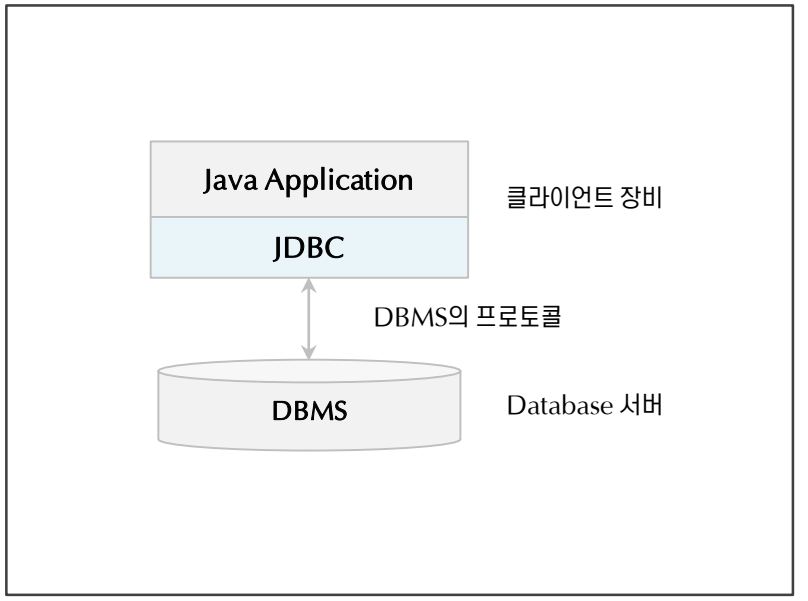
1.1 JDBC 소개

✓ 데이터베이스는 여러 시스템 또는 사람들이 공유할 목적으로 통합 관리하는 정보의 집합입니다.
✓ 데이터베이스는 데이터베이스 관리 시스템을 통해 운영되며 일반적으로 데이터베이스를 DBMS라고 부르기도 합니다.
✓ JDBC는 데이터베이스 연결을 지원하는 자바 API입니다.
✓ JDBC는 데이터베이스에서 데이터를 저장, 조회, 변경, 삭제하는 방법을 제공합니다.

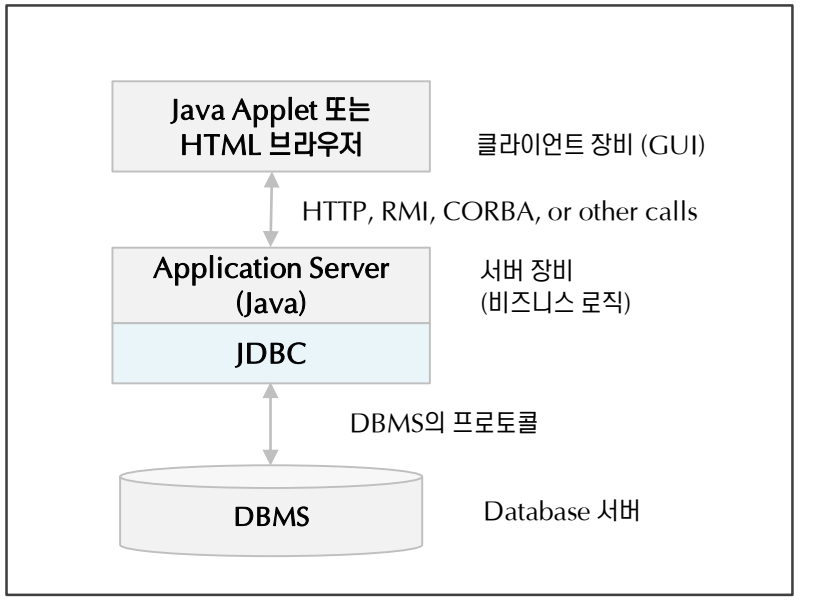


1.2 JDBC 아키텍처

✓ SQL은 관계형 데이터베이스 관리시스템에서 데이터를 관리에 사용하는 질의 언어입니다.
✓ DBC는 자바 애플리케이션에서 SQL을 이용하여 DBMS에 데이터를 입력하고 출력하는 역할을 담당합니다.
✓ JDBC API는 Two-tier와 Three-Tier모델에서 데이터 접근을 지원합니다.



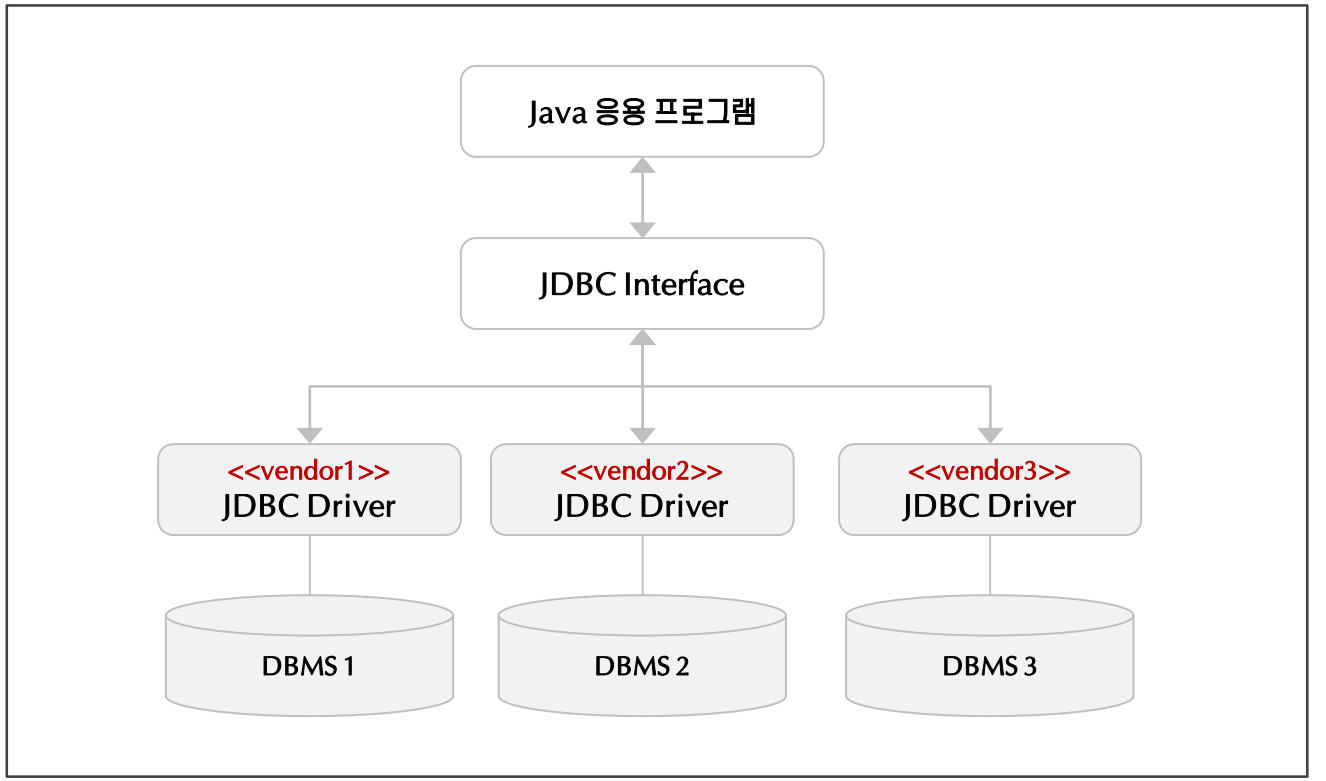
데이터 접근을 위한 2-티어 아키텍처



데이터 접근을 위한 3-티어 아키텍처

1.3 표준 인터페이스

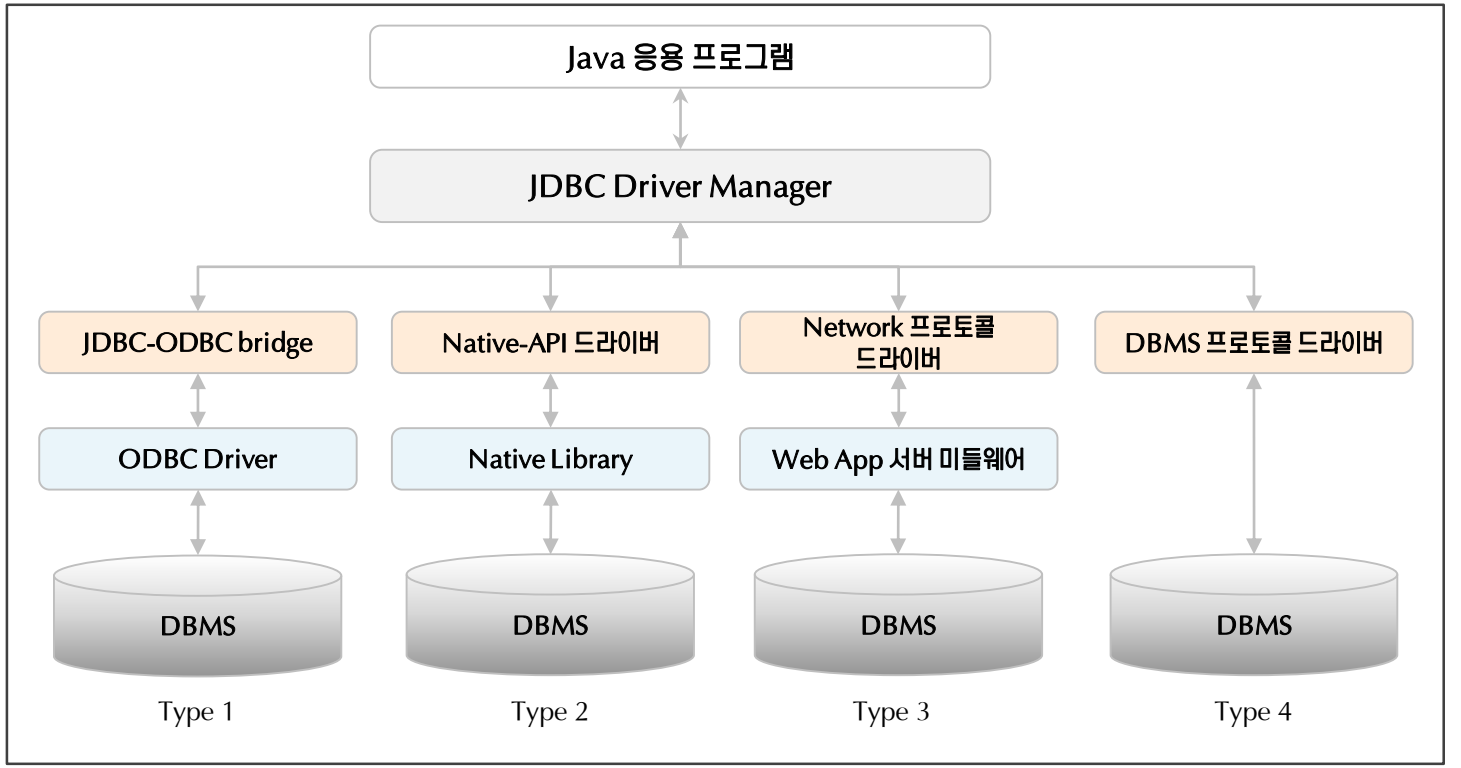
- ✓ 자바에서는 직접적으로 데이터베이스의 프로토콜을 사용하는 대신 JDBC 인터페이스를 사용 합니다.
- ✓ JDBC 인터페이스는 자바와 데이터베이스를 연결하기 위한 기능을 정의합니다.
- ✓ 데이터베이스 제조사는 JDBC 인터페이스를 자사 제품에 맞도록 구현합니다.
- ✓ 다양한 벤더들 때문에 자바에서는 JDBC 인터페이스 정의만 제공합니다.



JDBC 인터페이스와 DBMS의 관계

1.4 JDBC 드라이버

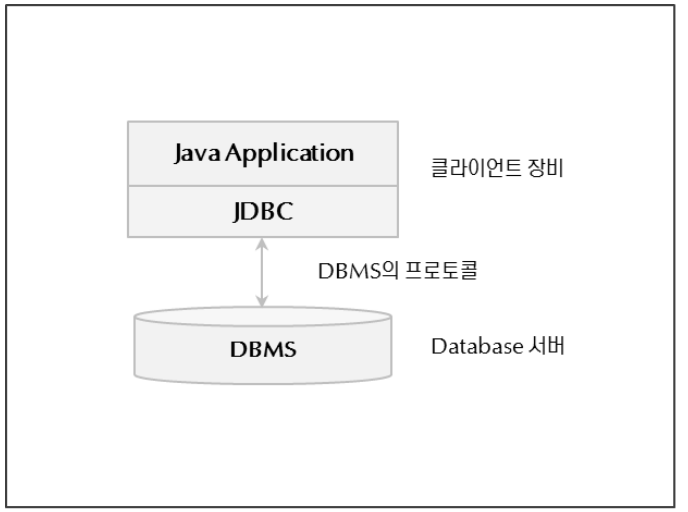
✓ JDBC 표준 인터페이스를 구현한 소프트웨어를 “JDBC 드라이버”라 합니다.
✓ JDBC 드라이버는 ODBC의 설계사상을 이어 받아 개발되었습니다.
✓ JDBC 드라이버는 “JDBC-ODBC 브리지”, “Native API”, “네트워크 프로토콜”, “DBMS 프로토콜’’로 구분합니다.
✓ 이 네 가지 중에 DBMS 프로토콜 드라이버를 주로 사용합니다.



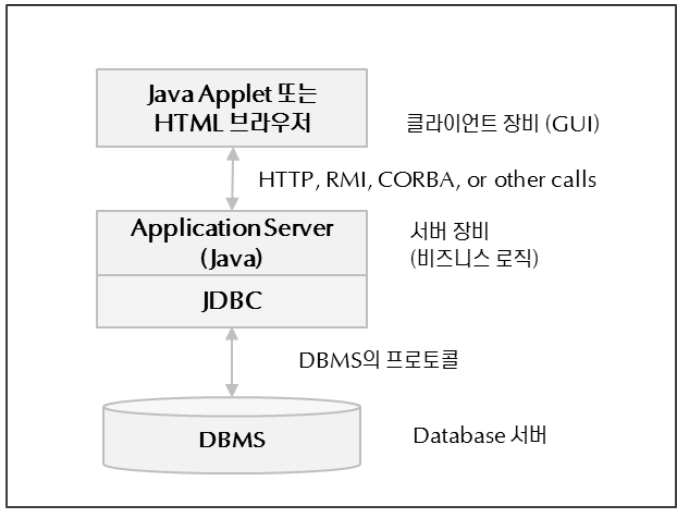
JDBC 드라이버의 종류

1.5 요약

✓ 데이터베이스는 여러 시스템 또는 사람들이 공유하고 사용할 목적으로 통합 관리되는 정보의 집합입니다.
✓ JDBC는 자바 애플리케이션에서 SQL을 이용하여 DBMS에 데이터를 입력하고 출력하는 역할을 담당합니다.
✓ 자바에서는 직접적으로 데이터베이스의 프로토콜을 사용하는 대신 JDBC 인터페이스를 사용 합니다.
✓ JDBC 드라이버는 JDBC 표준 인터페이스를 데이터베이스 벤더들이 자사 DB에 맞추어 구현한 소프트웨어입니다.



데이터 접근을 위한 2-티어 아키텍처



데이터 접근을 위한 3-티어 아키텍처

2. JDBC 프로그래밍

2.1 데이터 추가

- ✓ DriverManager.getConnection(url, user, password)을 사용하여 데이터베이스에 접속합니다.
- ✓ 데이터 추가는 PreparedStatement객체의 executeUpdate() 메소드를 사용합니다.
- ✓ setXXX() 메소드를 사용해서 파라미터 값을 설정합니다.
- ✓ executeUpdate()는 반영된 레코드 수를 반환합니다.

```
 // DB 연결객체 생성
Connection conn = DriverManager.getConnection(url, user, password);

// Statement 객체 생성 및 파라미터 세팅
String sql = "INSERT INTO lecture(lecNo, name, instructor) VALUES (?,?,?)";
PreparedStatement pstmt = conn.prepareStatement(sql);
pstmt.setString(1, "5");
pstmt.setString(2, "JavaScript");
pstmt.setString(3, "김현재");

// SQL 실행
int count = pstmt.executeUpdate();
System.out.println(count + "개의 레코드가 생성되었습니다.");

// 자원반납
pstmt.close();
conn.close();
```

DB에서 결과 확인

```
MariaDB [namootest]> select * from lecture;
+-------+-------------+------------+
| lecNo | name        | instructor |
+-------+-------------+------------+
| 1     | SQL기초     | 박석우     |
| 2     | JDBC        | 김현재     |
| 3     | SWT&JFace   | 한성구     |
| 4     | Servlet&JSP | 박지인     |
| 5     | JavaScript  | 김현재     |
+-------+-------------+------------+
5 rows in set (0.00 sec)

MariaDB [namootest]>
```

2.2 데이터 조회

- ✓ DriverManager.getConnection(url, user, password)를 통해서 데이터베이스와 접속합니다.
- ✓ 데이터 조회는 PreparedStatement객체의 executeQuery() 메소드를 사용합니다.
- ✓ executeQuery() 메소드는 조회결과를 ResultSet으로 반환합니다.
- ✓ ResultSet을 통해서 결과를 확인합니다.

```
 // DB 연결객체 생성
Connection conn = DriverManager.getConnection(url, user, password);

// PreparedStatement 생성
PreparedStatement pstmt = conn.prepareStatement(
    "SELECT name, instructor FROM lecture");

// 파라미터 세팅 및 SQL 실행
ResultSet result = pstmt.executeQuery();
while (result.next()) {
    System.out.println("강좌명:" + result.getString(1) +", 강사:" + result.getString(2));
}
result.close();

// 자원반납
pstmt.close();
conn.close();
```

실행결과

```
강좌명:SQL기초, 강사:박석우
강좌명:JDBC, 강사:김현재
강좌명:SWT&JFace, 강사:한성구
강좌명:Servlet&JSP, 강사:박지인
```

2.3 데이터 수정

- ✓ DriverManager.getConnection(url, user, password)를 통해서 데이터베이스와 접속합니다.
- ✓ 데이터 수정은 PreparedStatement객체의 executeUpdate() 메소드를 사용합니다.
- ✓ setXXX() 메소드를 사용해서 파라미터 값을 설정합니다.
- ✓ executeUpdate()는 반영된 레코드 수를 반환합니다.

```
 // DB 연결객체 생성
Connection conn = DriverManager.getConnection(url, user, password);

// Statement 객체 생성 및 파라미터 세팅
String sql = "UPDATE lecture SET instructor = ? WHERE lecNo = ?";
PreparedStatement pstmt = conn.prepareStatement(sql);
pstmt.setString(1, "최은정");
pstmt.setString(2, "5");

// SQL 실행
int count = pstmt.executeUpdate();
System.out.println(count + "개의 레코드가 갱신되었습니다.");

// 자원반납
pstmt.close();
conn.close();
```

DB에서 결과 확인

```
MariaDB [namootest]> select * from lecture;
+-------+-------------+------------+
| lecNo | name        | instructor |
+-------+-------------+------------+
| 1     | SQL기초     | 박석우     |
| 2     | JDBC        | 김현재     |
| 3     | SWT&JFace   | 한성구     |
| 4     | Servlet&JSP | 박지인     |
| 5     | JavaScript  | 최은정     |
+-------+-------------+------------+
5 rows in set (0.00 sec)

MariaDB [namootest]>
```

2.4 데이터 삭제

- ✓ DriverManager.getConnection(url, user, password)를 통해서 데이터베이스와 접속합니다.
- ✓ 데이터 삭제는 PreparedStatement객체의 executeUpdate() 메소드를 사용합니다.
- ✓ setXXX() 메소드를 사용해서 파라미터 값을 설정합니다.
- ✓ executeUpdate()는 반영된 레코드 수를 반환합니다.

```
 // DB 연결객체 생성
Connection conn = DriverManager.getConnection(url, user, password);

// Statement 객체 생성 및 파라미터 세팅
String sql = "DELETE FROM lecture WHERE lecNo = ?";
PreparedStatement pstmt = conn.prepareStatement(sql);
pstmt.setString(1, "5");

// SQL 실행
int count = pstmt.executeUpdate();
System.out.println(count + "개의 레코드가 삭제되었습니다.");

// 자원반납
pstmt.close();
conn.close();
```

DB에서 결과 확인

```
MariaDB [namootest]> select * from lecture;
+-------+-------------+------------+
| lecNo | name        | instructor |
+-------+-------------+------------+
| 1     | SQL기초     | 박석우     |
| 2     | JDBC        | 김현재     |
| 3     | SWT&JFace   | 한성구     |
| 4     | Servlet&JSP | 박지인     |
+-------+-------------+------------+
4 rows in set (0.00 sec)

MariaDB [namootest]>
```

2.5 요약

- ✓ JDBC 인터페이스는 데이터베이스에 연결하고, 데이터 입출력을 위한 기능을 제공합니다.
- ✓ DriverManager.getConnection()은 데이터베이스와 접속하는 메소드입니다.
- ✓ 데이터의 추가, 수정, 삭제는 PreparedStatement객체의 executeUpdate() 메소드를 사용합니다.
- ✓ 데이터 조회는 PreparedStatement객체의 executeQuery() 메소드를 사용합니다.

3. JDBC 이해

3.1 JDBC 드라이버 로드

- ✓ DB와의 접속을 위해서 먼저 드라이버를 설정합니다.
- ✓ Class.forName("jdbc.Driver")방법으로 드라이버를 설정할 수 있습니다.
- ✓ Class.forName은 드라이버를 동적으로 로딩합니다.
- ✓ 드라이버가 로딩되었으면 DriverManager.getConnection() 메소드를 통해서 DB와 접속합니다.

```
public final class Driver implements java.sql.Driver {
    ...
    static {
        try {
            DriverManager.registerDriver(new Driver());
        } catch (SQLException e) {
            throw new RuntimeException("Could not register driver", e);
        }
    }
```

DBMS에서 제공한 드라이버 클래스

개발자가 작성할 코드

```
try {
    //드라이버 로딩 -> 드라이버 매니저 등록
    Class.forName("org.mariadb.jdbc.Driver");
}
catch (ClassNotFoundException e) {
    e.printStackTrace();
}
```

MariaDB

드라이버가 로드될 때 실행되면서
DriverManager에 자동으로 등록 됩니다.

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

Oracle

3.2 Connection

- ✓ Connection은 자바 애플리케이션과 데이터베이스를 연결하는 객체입니다.
- ✓ 서버주소(localhost), 포트번호(3306), 사용자 ID, 비밀번호, 데이터베이스 명으로 데이터베이스에 접속합니다.
- ✓ DriverManager를 통해서 Connection 객체를 획득합니다.
- ✓ API를 모두 사용한 후에는 close() 메소드를 호출하여 자원은 반납합니다.

```
String url = "jdbc:mysql://localhost:3306/namootest"; //MariaDB
//String url = "jdbc:oracle:thin:@localhost:1521:XE“; //Oracle
String user = "namoouser";
String password = "namoouser";

//드라이버 로딩 -> 드라이버 매니저 등록
Class.forName("org.mariadb.jdbc.Driver"); //MariaDB
//Class.forName("oracle.jdbc.driver.OracleDriver"); //Oracle

// DB 연결객체 생성
Connection conn = DriverManager.getConnection(url, user, password);

// JDBC API 수행
// ...

// DB연결객체 자원반납
conn.close();
```

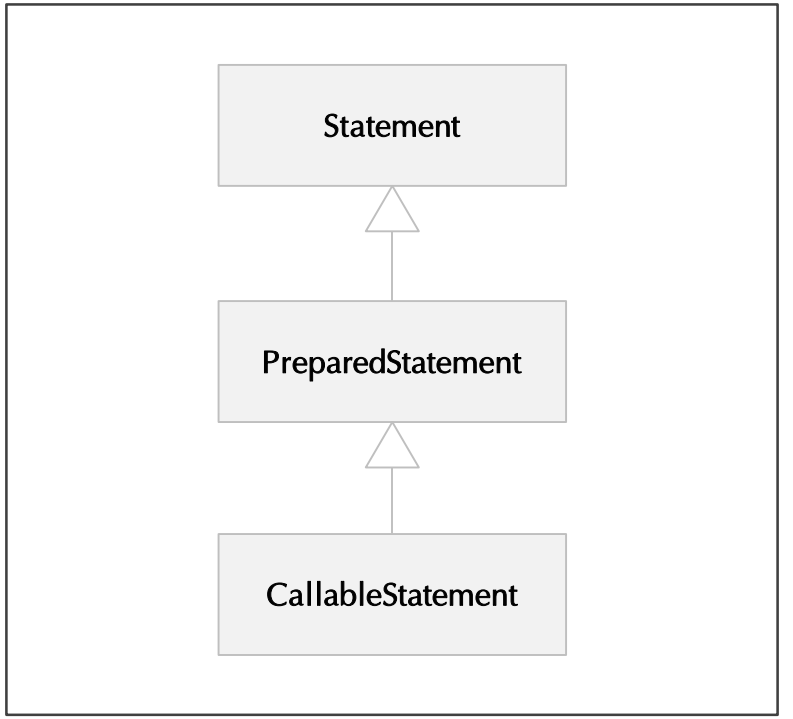
3.3 Statement

- ✓ SQL을 만들고 수행한 후 결과 값을 제공하며 Connection.createStatement() 메소드로 객체를 생성합니다.
- ✓ execute(String sql)은 질의문(select)이나 갱신문(insert, update, delete) 모두 사용 가능합니다.
- ✓ executeQuery(String sql)는 select 실행 시 사용하고, ResultSet을 객체로 반환합니다.
- ✓ executeUpdate(String sql)는 insert, update 또는 delete 실행 시 사용됩니다. 결과는 반영된 레코드 개수입니다.

```
// JDBC API 수행
Statement stmt = conn.createStatement();
stmt.executeQuery("SELECT * FROM lecture WHERE instructor='김현재'");
// ...

// 자원반납
stmt.close();
conn.close();
```

3.4 PreparedStatement

- ✓ PreparedStatement 인터페이스는 Statement 인터페이스를 상속합니다.
- ✓ PreparedStatement 인터페이스는 Statement 인터페이스의 단점을 개선합니다.
- ✓ CallableStatement 인터페이스는 PreparedStatement 인터페이스를 상속합니다.
- ✓ CallableStatement 인터페이스를 사용해서 SQL StoredProcedure를 수행합니다.



PreparedStatement 클래스의 상속계층

3.5 Statement와 PreparedStatement 비교

- ✓ createStatement()를 통해서 생성된 statement 객체는 재사용할 수 없습니다.
- ✓ prepareStatement()를 통해서 생성된 statement 객체는 다시 사용할 수 있습니다.
- ✓ PreparedStatement의 재사용은 쿼리의 변경되는 데이터 값만 setXXX() 메소드로 변경합니다.
- ✓ PreparedStatement를 사용할 경우 SQL의 변경되는 데이터는 '?'로 작성합니다.

Statement를 사용한 예

```
// Statement 생성 및 SQL 실행
Statement stmt = conn.createStatement();
ResultSet result = stmt.executeQuery(
   "SELECT name FROM lecture WHERE instructor='김현재'");

while (result.next()) {
   System.out.println("강좌명:" + result.getString(1));
}
result.close();
stmt.close();
// Statement 생성 및 SQL 실행
stmt = conn.createStatement();
result = stmt.executeQuery(
   "SELECT name FROM lecture WHERE instructor='박석우'");

while (result.next()) {
   System.out.println("강좌명:" + result.getString(1));
}
result.close();
//
stmt.close();
conn.close();
```

PreparedStatement를 사용한 예

```
// PreparedStatement 객체 생성
PreparedStatement pstmt = conn.prepareStatement(
    "SELECT name FROM lecture WHERE instructor = ?");

// 파라미터 세팅 및 SQL 실행
pstmt.setString(1, "김현재");
ResultSet result = pstmt.executeQuery();
while (result.next()) {
    System.out.println("강좌명:" + result.getString(1));
}
result.close();
// 파라미터 세팅 및 SQL 실행 (statement객체 재사용)
pstmt.setString(1, "박석우");
result = pstmt.executeQuery();
while (result.next()) {
    System.out.println("강좌명:" + result.getString(1));
}
result.close();

//
pstmt.close();
conn.close();
```

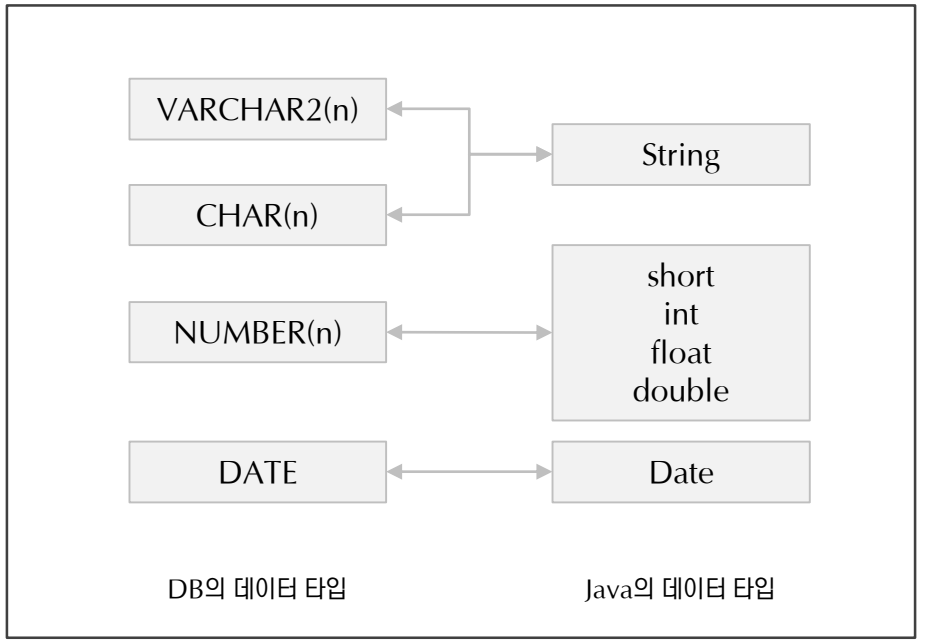
3.6 ResultSet

- ✓ ResultSet은 SQL 쿼리 실행 결과를 다루는 인터페이스로써 Statememt의 메소드 실행으로 얻을 수 있습니다.
- ✓ ResultSet.next()는 결과를 얻을 수 있는 행으로 이동하면서 결과의 획득 여부를 boolean 타입으로 반환합니다.
- ✓ ResultSet.getString(int n)은 현재 행에서 n번째 컬럼의 데이터를 String 값으로 반환합니다.
- ✓ ResultSet.getString(String columnName)은 현재 행에서 columnName으로 조회된 데이터를 반환합니다.

```
// Statement 생성 및 SQL 수행
Statement stmt = conn.createStatement();
ResultSet result = stmt.executeQuery(
   "SELECT name FROM lecture WHERE instructor='김현재'");

while (result.next()) {
   System.out.println("강좌명:" + result.getString(1));
}
result.close();
stmt.close();
```

3.7 데이터 타입의 변환

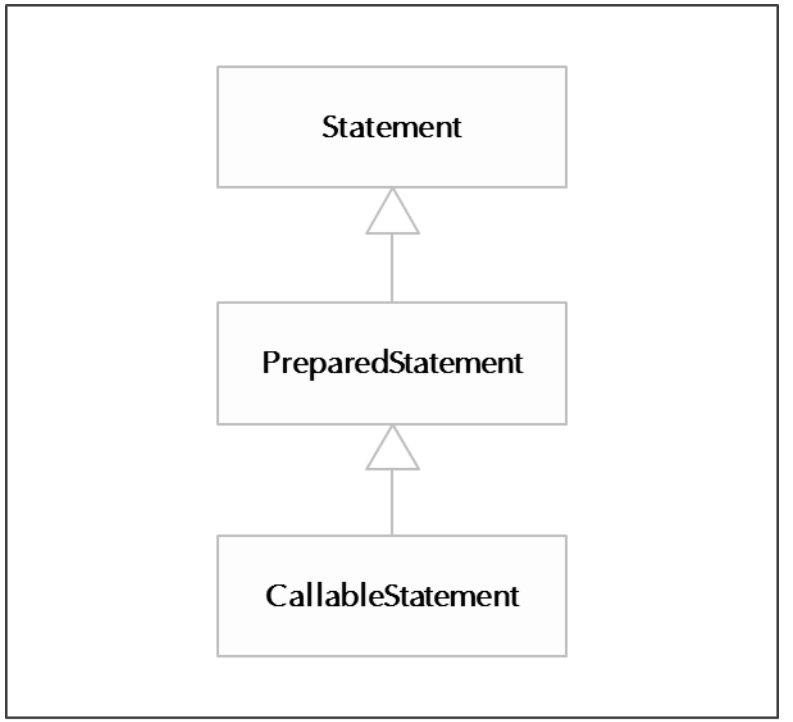
- ✓ 모든 프로그래밍 언어는 자체적인 데이터 타입을 가지고 있으며 데이터베이스도 고유의 데이터 타입을 가지고 있습니다.
- ✓ 자바와 DBMS는 데이터 타입이 서로 다르므로 변환작업이 필요합니다.
- ✓ 자바 타입을 데이터베이스 타입으로 변환할 때는 PreparedStatement의 setXXX 메소드를 사용합니다.
- ✓ 데이터베이스 타입을 자바 타입으로 변환할 때는 ResultSet 객체의 getXXX 메소드를 이용합니다.



데이터베이스와 Java 간의 데이터 타입 변환

3.8 요약

- ✓ JDBC는 데이터베이스를 사용하는 객체를 제공합니다. 사용이 끝난 후에는 close()를 통해 자원을 반납해야 합니다.
- ✓ Connection 객체는 자바 애플리케이션과 데이터베이스를 연결하는 객체입니다.
- ✓ Statement 객체는 SQL을 만들고, 수행결과를 제공하는 객체로, 하위 클래스에는 PrepareStatement 등이 있습니다.
- ✓ ResultSet 객체는 SQL 쿼리 실행 결과를 다루는 인터페이스로, Statement 메소드 실행을 통해 얻을 수 있습니다.

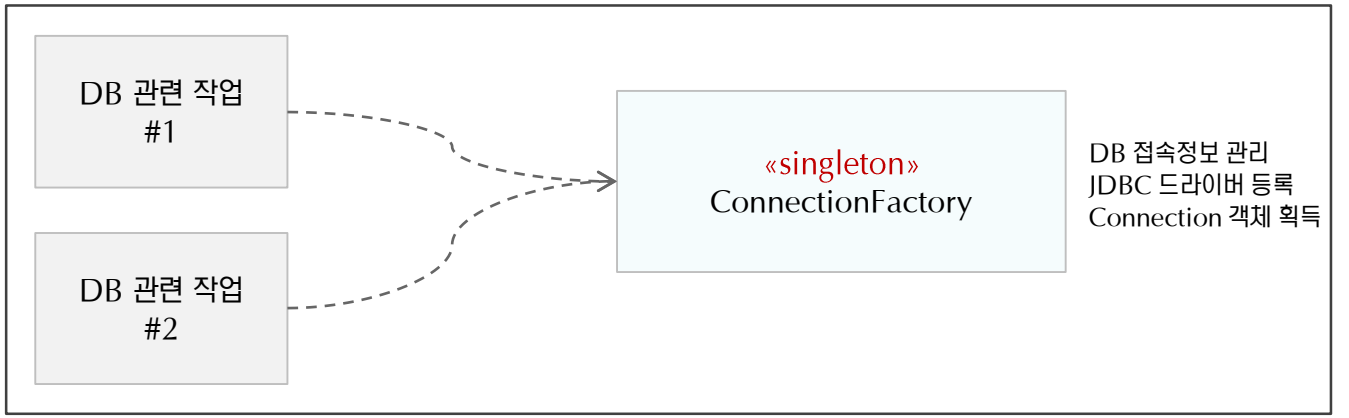


PreparedStatement 클래스의 상속계층

4. JDBC 이해 II

4.1 Connection Factory (1/4) – 개요

- ✓ JDBC는 DriverManager를 통해 Connection 객체를 획득합니다.
- ✓ 여러 곳에서 DB 관련 처리를 하는 경우 DB접속관련 코드 중복이 발생하여 유지보수를 어렵게 합니다.
- ✓ Connection Factory를 통해서 DB 접속관련 코드를 공통으로 사용하면 중복 코드가 발생하지 않습니다.
- ✓ Connection Factory를 사용하면 미래 DB 변경에 대해서도 수정이 편리합니다.



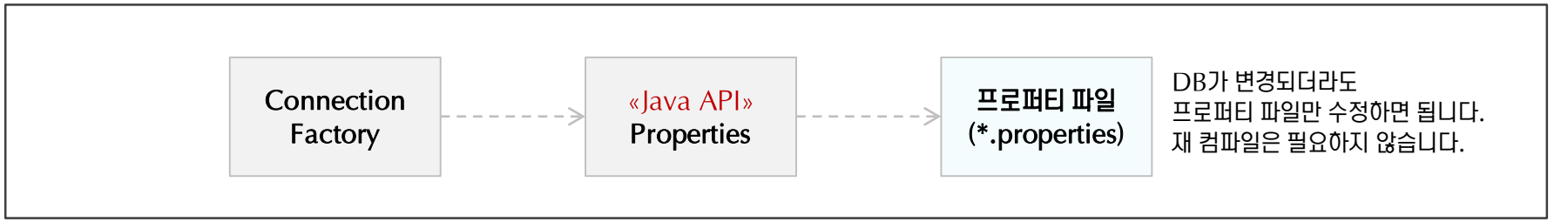
4.1 Connection Factory (2/4) – 구현

- ✓ ConnectionFactory 생성자를 통해서 드라이버를 동적으로 로딩합니다.
- ✓ ConnectionFactory는 재사용이 가능하므로 메모리에 한번만 생성합니다.
- ✓ DBMS 접속정보를 프로퍼티에 저장하면 미래에 DB가 변동되더라도 재 컴파일이 필요없습니다.
- ✓ createConnection() 메소드는 신규 Connection 객체를 반환합니다.

```
public class ConnectionFactory {

    private static final String DRIVER_NAME = "org.mariadb.jdbc.Driver"; //MariaDB
    //parivate static final String DRVIVER_NAME = "oracle.jdbc.driver.OracleDriver"; //Oracle
    private static final String URL = "jdbc:mysql://localhost:3306/namootest"; //MariaDB
    //parivate static final String URL = "jdbc:oracle:thin:@localhost:1521:XE"; //Oracle
    private static final String USERNAME = "namoouser";
    private static final String PASSWORD = "namoouser";
    private static ConnectionFactory instance = new ConnectionFactory();

    private ConnectionFactory() {
        try {
            Class.forName(DRIVER_NAME);
        } catch (ClassNotFoundException e) { }
    }
    public static ConnectionFactory getInstance() {
        return instance;
    }

    public Connection createConnection() throws SQLException {
        //
        return DriverManager.getConnection(URL, USERNAME, PASSWORD);
    }
}
```

4.1 Connection Factory (3/4) – DB 접속정보 관리 (1/2)

- ✓ DB 접속정보를 클래스 내에서 관리할 경우 DB가 변경되면 클래스를 수정해서 다시 컴파일 합니다.
- ✓ "Properties"(프로퍼티)는 키-값 형태로 데이터를 저장하는 객체입니다.
- ✓ 프로퍼티 값은 별도의 파일로 분리하여 관리 할 수 있습니다.
- ✓ DB접속정보를 Properties 파일로 만들어 관리하면 DB접속정보가 변경되더라도 재 컴파일이 필요없습니다.




```
private void loadProperties() {
    //
    Properties prop = new Properties();
    InputStream is = this.getClass().getResourceAsStream("jdbc.properties");
    try {
        prop.load(is);
        this.driver = prop.getProperty("database.driver");
        this.url = prop.getProperty("database.url");
        this.username = prop.getProperty("database.username");
        this.password = prop.getProperty("database.password");
    } catch (IOException e) {
        //
        e.printStackTrace();
    }
}
```

4.1 Connection Factory (4/4) – DB 접속정보 관리 (2/2)

- ✓ 프로퍼티를 적용해서 Connection Factory를 개선할 수 있습니다.
- ✓ 자바에 코딩된 DB접속 정보를 jdbc.properties 파일로 옮겼습니다.
- ✓ jdbc.properties에 저장된 DB접속 정보를 읽기 위해 Properties 객체를 사용합니다.
- ✓ 개선된 ConnectionFactory는 재 컴파일이 필요없습니다.

```
public class ConnectionFactory {
    //
    private static ConnectionFactory instance =
                    new ConnectionFactory();

    private String driver;
    private String url;
    private String username;
    private String password;

    private ConnectionFactory() {
        //
        loadProperties();

        try {
            Class.forName(driver);
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
```

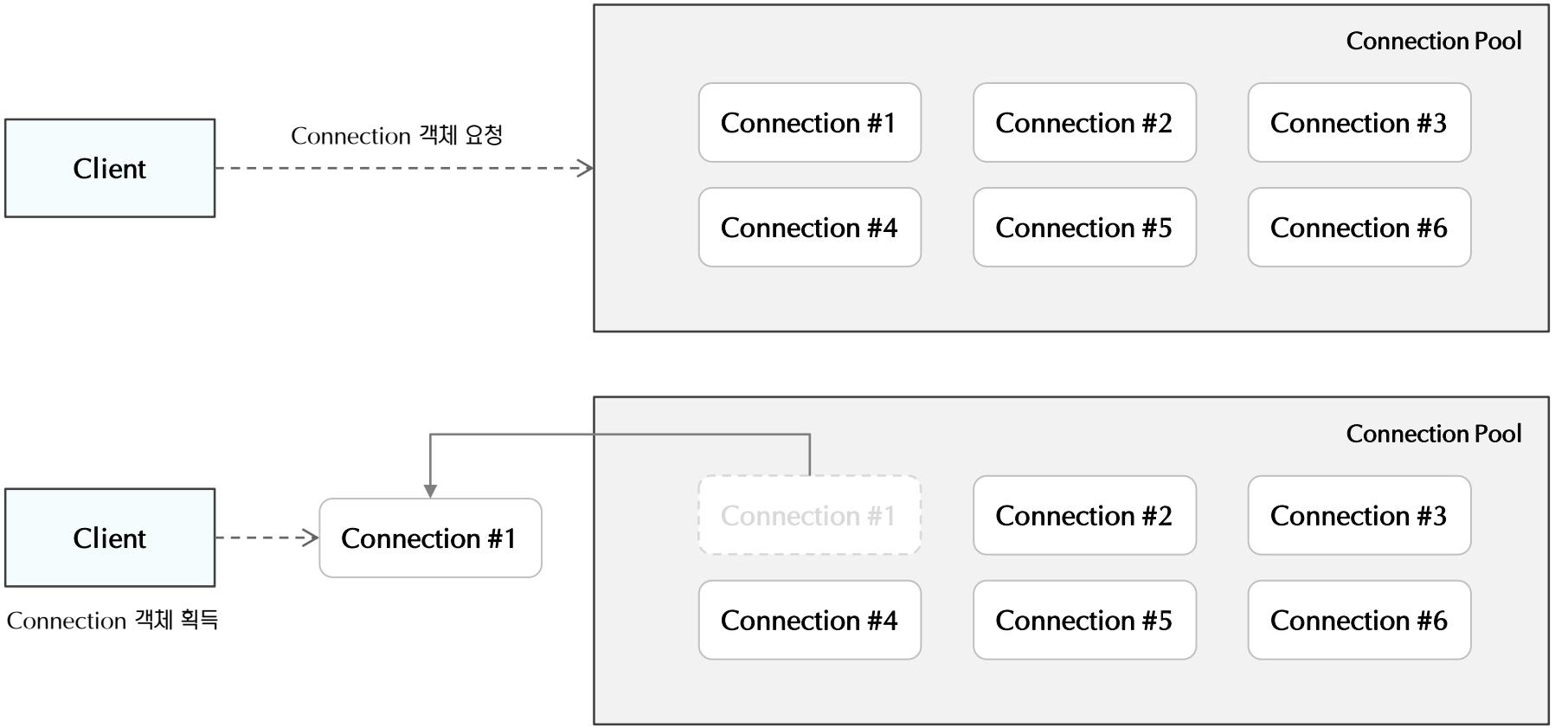
jdbc.properties

```
database.driver=org.mariadb.jdbc.Driver
database.url=jdbc:mysql://localhost:3306/namootest
database.username=namoouser
database.password=namoouser
```

```
    private void loadProperties() {
        //
        Properties prop = new Properties();
        InputStream is = this.getClass()
                        .getResourceAsStream("jdbc.properties");
        try {
            prop.load(is);
            this.driver = prop.getProperty("database.driver");
            this.url = prop.getProperty("database.url");
            this.username = prop.getProperty("database.username");
            this.password = prop.getProperty("database.password");
        } catch (IOException e) {
            //
            e.printStackTrace();
        }
    }

    public static ConnectionFactory getInstance() {
        //
        return instance;
    }

    public Connection createConnection() throws SQLException {
        //
        return DriverManager.getConnection(url, username, password);
    }
}
```

4.2 Connection Pool (1/2) – 개요

- ✓ DB 프로그래밍에서 Connection 객체를 획득하는데 걸리는 시간을 줄일 수 있다면 응답 시간을 단축 할 수 있습니다.
- ✓ Connection Pooling 기법은 Connection 객체를 요청할 때 마다 새롭게 생성하지 않고 재사용합니다.
- ✓ Connection Pool에 일정 수의 Connection 객체를 미리 준비해 두고 요청 시 미리 준비된 객체를 재사용합니다.
- ✓ 사용이 완료된 Connection객체는 다시 Pool로 반환합니다.



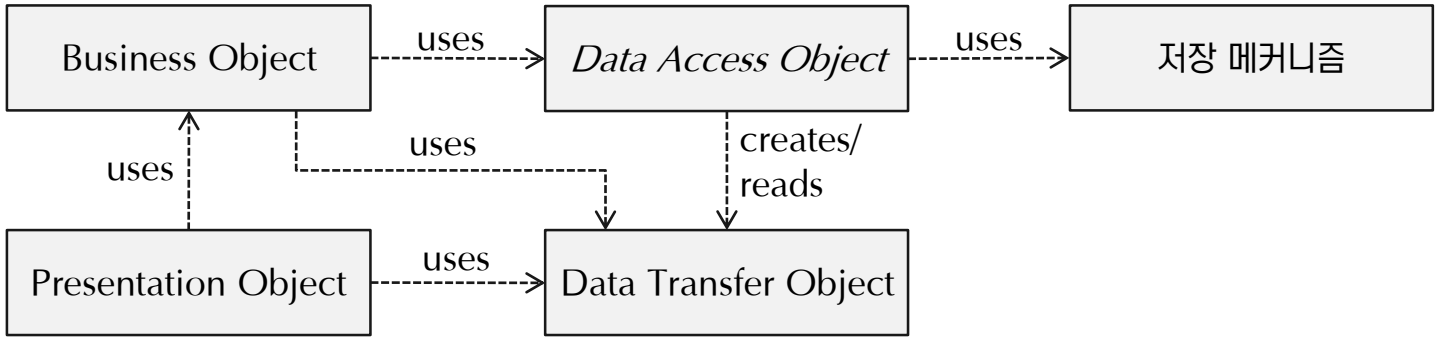
4.2 Connection Pool (2/2) – Apache Commons DBCP

✓ Apache의 Commons DBCP는 데이터베이스 Connection Pool 을 지원하기 위한 오픈 소스 프로젝트 입니다.
✓ DBCP를 사용하기 위해서는 http://commons.apache.org/ 에서 필요한 라이브러리를 다운받습니다.
✓ 필요한 파일은 “commons-dbcp”, “commons-pool”, “commons-logging” 3가지 입니다.

```
public class DbConnection {
    private static DbConnection instance = new DbConnection();

    private DataSource dataSource;
    private DbConnection() {
        //
        BasicDataSource ds = new BasicDataSource();
        ds.setDriverClassName("org.mariadb.jdbc.Driver"); //MariaDB
        //ds.setDriverClassName("oracle.jdbc.driver.OracleDriver"); //Oracle
        ds.setUsername("namoouser");
        ds.setPassword("namoouser");
        ds.setUrl("jdbc:mysql://localhost:3306/namootest"); //MariaDB
        //ds.setUrl("jdbc:oracle:thin:@localhost:1521:XE"); //Oracle

        ds.setMaxTotal(100); // 최대 커넥션 갯수
        ds.setMaxIdle(10);   // Idle 상태에 풀이 소유한 최대 커넥션 개수
        ds.setInitialSize(10); // 풀의 초기 커넥션 개수
        ds.setMaxWaitMillis(1000); // 커넥션이 존재하지 않을 때 커넥션 획득에 대기할 시간
        dataSource = ds;
    }
    public static Connection getConnection() throws SQLException {
        return instance.dataSource.getConnection();
    }
}
```

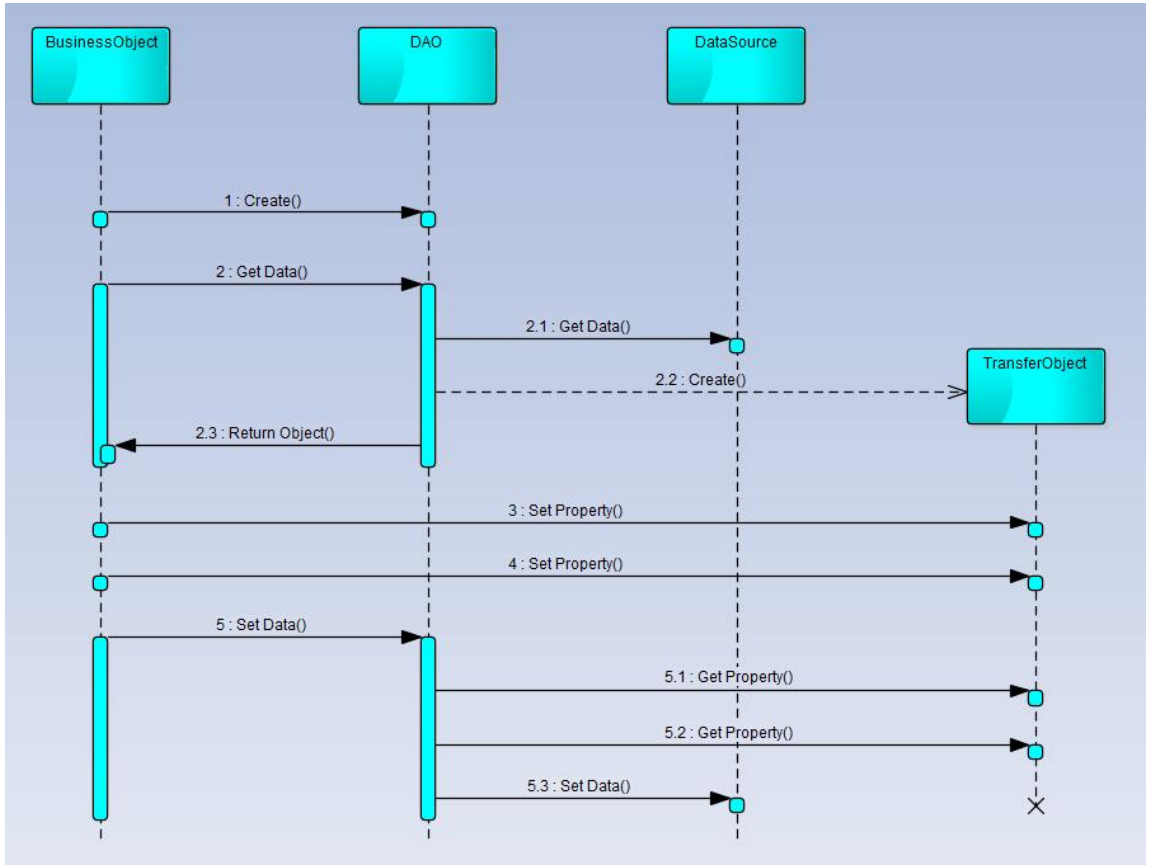
4.3 데이터 접근 패턴 (1/3)

- ✓ Data Access Object(DAO) 패턴은 DB에 접근하는 코드와 데이터를 활용하는 코드를 분리합니다.
- ✓ DAO 객체는 DB 접근에 대한 책임을 갖습니다.
- ✓ DAO 객체는 Data Transfer Object(DTO)를 통해서 다른 객체들과 소통합니다.
- ✓ Data Access Object(DAO)의 도입은 비즈니스 로직과 자원 티어를 분리합니다.



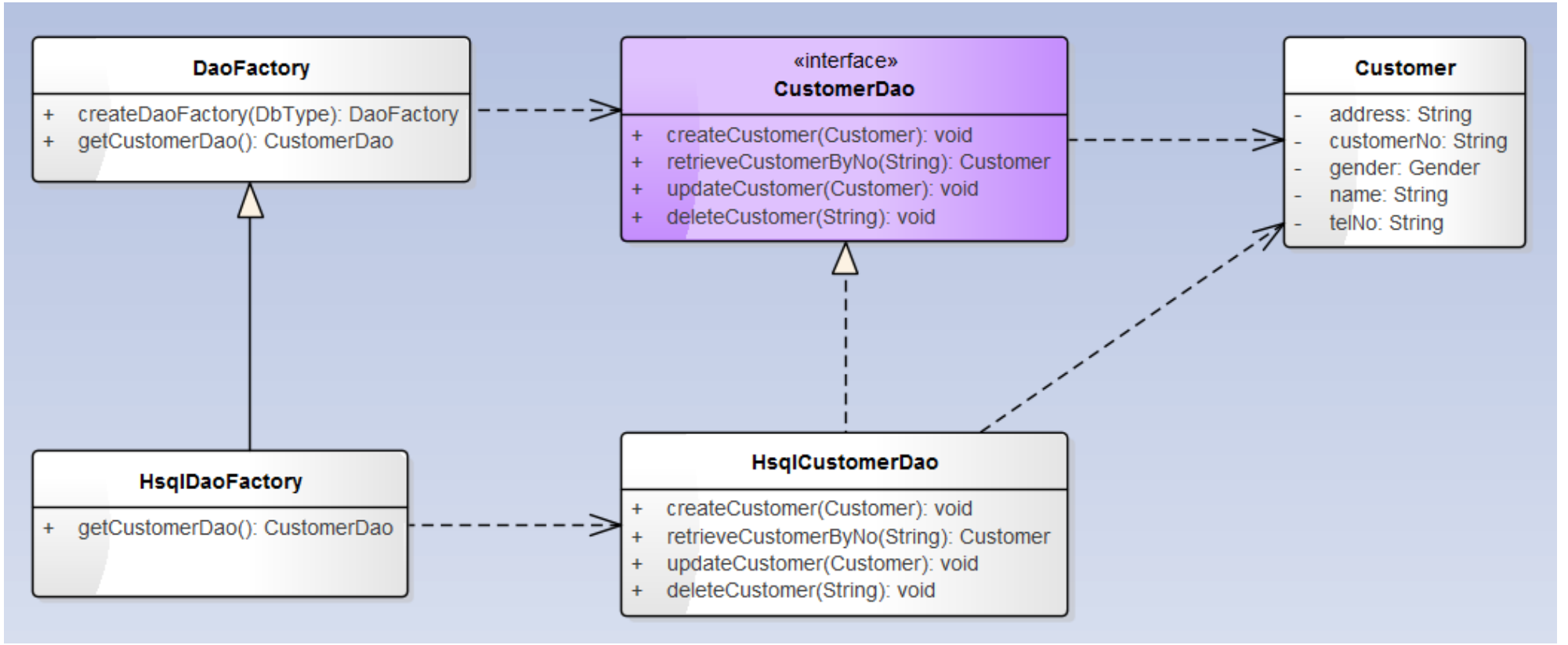
4.3 데이터 접근 패턴 (2/3)

- ✓ DAO는 비즈니스 컴포넌트가 외부데이터를 사용하도록 하기 위해 객체 캡슐화를 제공합니다.
- ✓ EJB 엔티티 빈과 달리 원격 객체가 아니며 데이터에 대한 getter/setter를 제외한 비즈니스 메소드는 포함하지 않습니다.
- ✓ 비즈니스 로직은 별도로 위치하고 독립적으로 테스트가 가능합니다.
- ✓ 비즈니스 로직은 저장 레이어가 변경되어도 재사용이 가능합니다.



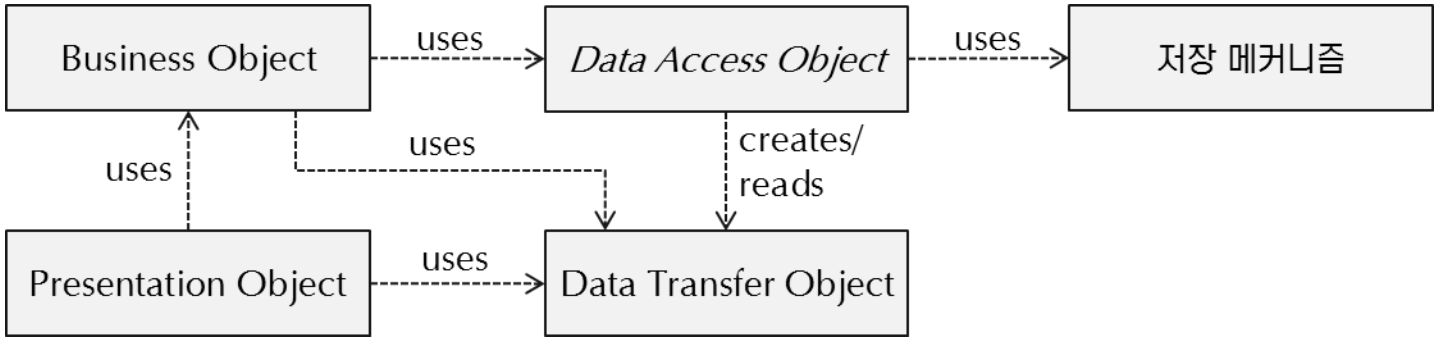
4.3 데이터 접근 패턴 (3/3)

- ✓ CustomerDao는 사용자 정보를 관리하는 DAO 입니다.
- ✓ HsqlCustomerDao는 메모리에 데이터를 저장하는 CustomerDao 구현체 입니다.
- ✓ Customer는 데이터 클라이언트와 주고 받는 메시지 입니다.
- ✓ 데이터 클라이언트는 CustomerDao와 독립된 비즈니스 로직을 구성할 수 있습니다.



4.4 요약

- ✓ ConnectionFactory 를 구현하여 반복적으로 나타나는 DB접속관련 코드를 한 곳에서 관리합니다.
- ✓ Connection Pool 기법은 연결 객체를 요청할 때 마다 새롭게 생성하지 않고 재사용하는 방법으로 성능을 개선합니다.
- ✓ Apache Commons DBCP를 이용하면 데이터베이스 Connection Pool 을 편리하게 프로젝트에 적용할 수 있습니다.
- ✓ Data Access Object(DAO) 패턴은 DB에 접근하는 로직과 데이터를 활용하는 업무 로직을 분리합니다.

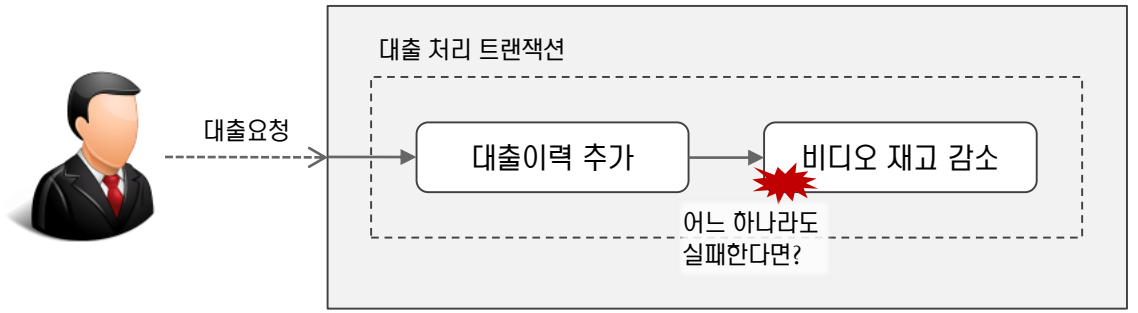


5. 트랜잭션 처리

5.1 트랜잭션 개념

- ✓ 트랜잭션은 여러 개의 관련 있는 작업들을 하나의 처리단위로 묶어 놓은 것을 의미합니다.
- ✓ 하나의 트랜잭션은 모든 데이터가 정상적으로 반영되거나 반영되지 않음을 보장합니다.
- ✓ 애플리케이션에서 한번의 요청이 여러 테이블의 데이터들을 동시에 수정한다면 하나의 트랜잭션으로 처리되어야 합니다.
- ✓ 트랜잭션이 적용되지 않으면 애플리케이션의 데이터를 신뢰 할 수 없습니다.

비디오 대출 처리



- 대출이 일어날 경우
  1) 대출 테이블에 대출 이력에 대한 행이 추가됨과 동시에
  2) 영화 테이블의 재고 수량과 수정일자 값이 갱신되어야 합니다.
- 만약 두 작업 중 하나라도 이루어지지 않는다면 데이터의 무결성이 보장이 안됩니다.
  따라서 두 작업은 하나의 Transaction으로 처리되어야 합니다.

[대출 테이블]

고객ID	영화등록번호	대여일자	대여비
C0001	M0001	2015-03-03	500
C0002	M0002	2015-02-26	500
C0003	M0003	2015-03-02	1000
C0003	**M0003**	**2015-03-09**	**1000**

① 대출 테이블에 새로운 행이 추가됨

[영화 테이블]

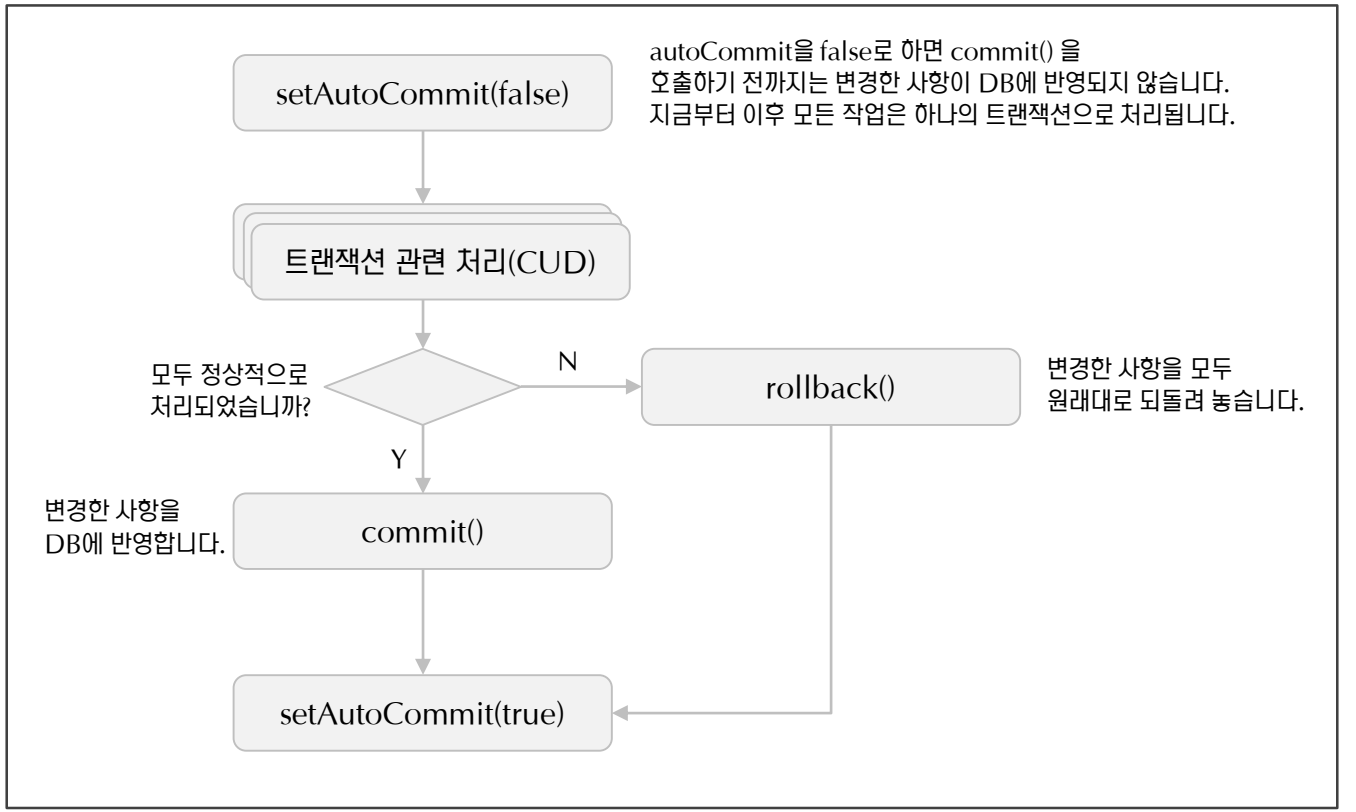
영화등록번호	영화제목	재고	수정일자
M0001	변호인	10	2015-03-03
M0002	또 하나의 약속	8	2015-02-26
M0003	어벤져스 2	5	2015-03-02

② 재고량 감소 (5 → 4)

② 수정일자 갱신 (→ 2015-03-09)

5.2 JDBC 트랜잭션 관리

- ✓ JDBC의 Connection 인터페이스는 트랜잭션 관리를 위한 메소드를 제공합니다.
- ✓ 여러 Statement를 한번에 반영함으로써 트랜잭션을 보장합니다.
- ✓ Connection.commit() 또는 Connection.rollback() 등으로 수행된 Statement를 반영하거나 취소합니다.
- ✓ 트랜잭션을 관리하기 위해서는 Connection.setAutoCommit(false)를 수행합니다.(기본값은 true입니다.)



JDBC를 이용한 트랜잭션 처리과정

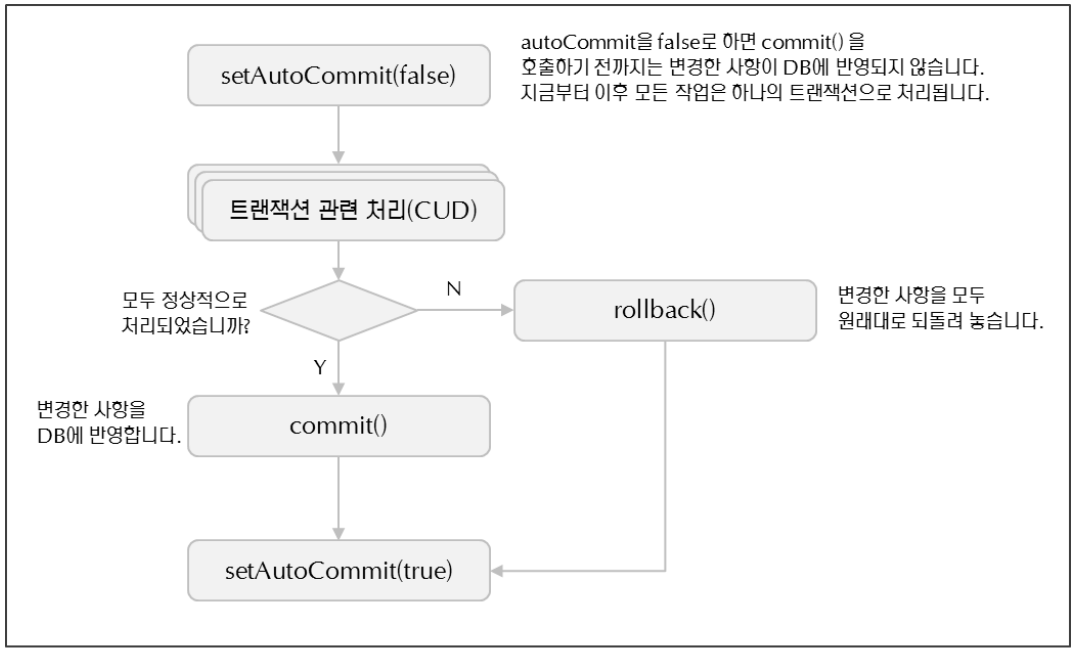
5.3 트랜잭션 예외처리

- ✓ JDBC를 사용해서 DB와 연동할 때 접속실패, SQL오류 등의 예외가 발생할 수 있습니다.
- ✓ 예외는 SQLException과 이를 상속받은 SQLWarning, DataTruncation, BatchUpdateException등이 있습니다.
- ✓ 트랜잭션 처리 중 예외가 발생하는 경우 catch절을 통해서 rollback()을 수행하도록 합니다.
- ✓ 예외처리가 적절히 수행되지 않는다면 데이터의 무결성을 보장 할 수 없습니다.

```
try {
    // DB 연결객체 생성
    conn = DriverManager.getConnection(url, user, password);
    // 트랜잭션 시작
    conn.setAutoCommit(false);

    // 트랜잭션 관련 처리1
    String sql = "INSERT INTO ... VALUES (?,?,?)";
    pstmt = conn.prepareStatement(sql);

    // 트랜잭션 관련 처리2
    // ...
    conn.commit();
} catch (SQLException e) {
    //
    e.printStackTrace();
    try {
        conn.rollback();
    } catch (SQLException e1) { }
} finally {
    try {
        conn.setAutoCommit(true);
        pstmt.close();
        conn.close();
    } catch (SQLException e) { }
}
```

5.4 요약

- ✓ 트랜잭션은 여러 개의 관련있는 작업들을 하나의 처리단위로 묶어 놓은 것을 의미합니다.
- ✓ JDBC 는 여러 Statement 를 한번에 반영함으로써 트랜잭션을 보장합니다.
- ✓ JDBC Connection 인터페이스는 commit()과 rollback()과 같은 트랜잭션 관리를 위한 메소드를 제공합니다.
- ✓ 트랜잭션이 필요한 DB 관련 작업에서는 데이터 무결성을 보장하기 위하여 트랜잭션 예외처리가 필요합니다.



JDBC를 이용한 트랜잭션 처리과정

토론

✓ 질문과 대답

✓ 토론