

고급객체지향 프로그래밍 강의노트 #11

DAO Pattern

조용주

ycho@smu.ac.kr

다오 패턴 (DAO Pattern)

□ 목적

- 업무와 DBMS 를 분리하기 위해 사용됨
- 업무와 데이터 2 계층을 분리하고, 서로 그 상태를 동일하게 유지 (persistence)

□ 업무 : 고객, 주문, 유통 같은 업무 절차 및 규칙

□ 데이터 : 정보를 어떻게 저장할지, 예를 들어 고객 ID 는 몇 자리로 할 것인지, 숫자 또는 문자인지 등 물리적인 문제

디자인 패턴 요소

요소	설명
이름	다오 (DAO)
문제	DB 를 사용하는 방법이 변경되면 클라이언트의 수정이 많아진다
해결방안	사용 방법의 분리
결과	loose coupling, 확장성

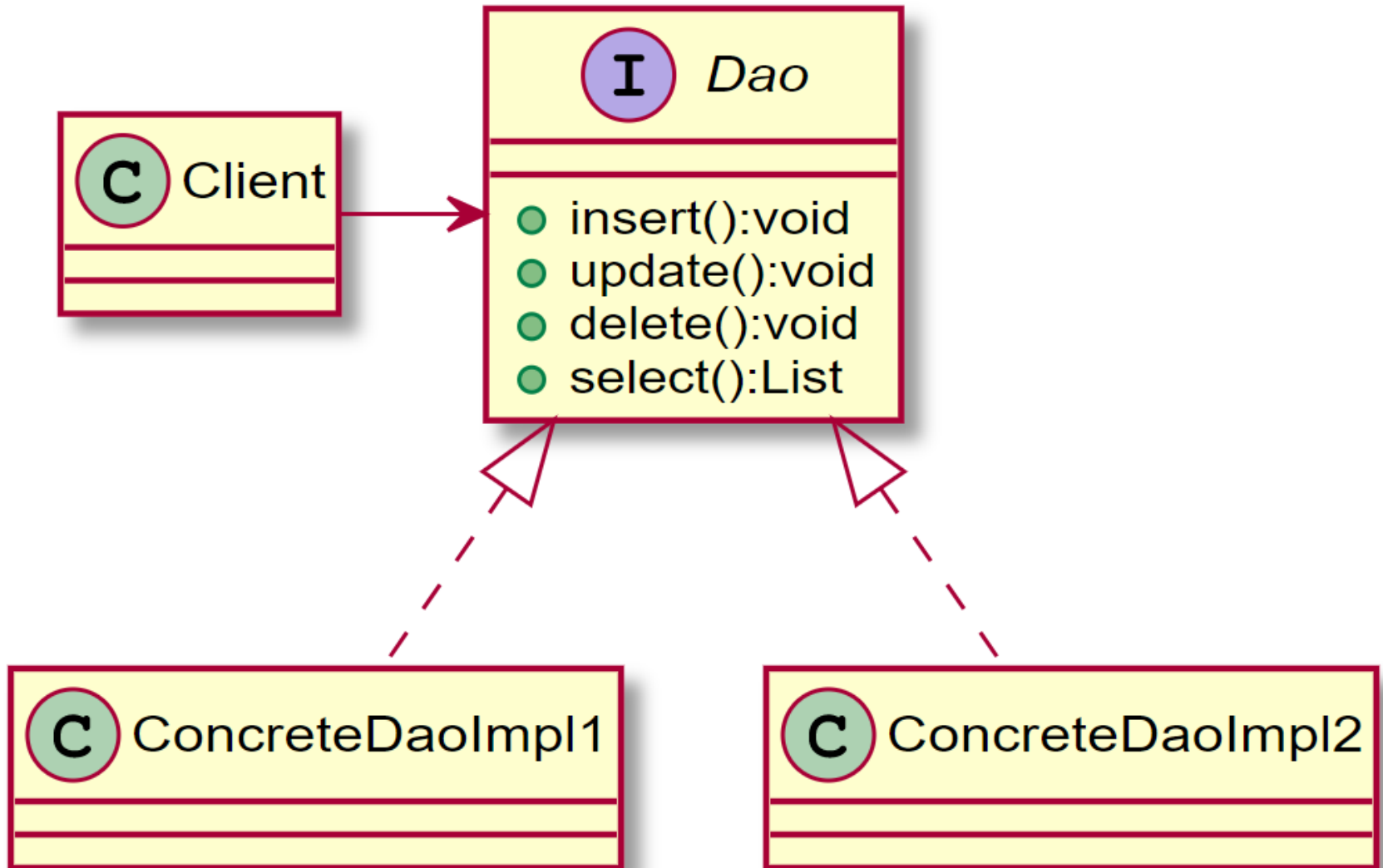
문제

- ❑ 데이터를 저장하는 방식이 다양함 (데이터베이스 , 파일 , XML, csv 파일 등)
- ❑ 사용하는 방법이 다르면 변경 부분이 많아짐 (예 : DBMS 에 따라 SQL 문이 다를 수 있음)
- ❑ 데이터를 저장하는 방식을 분리하여 , 변경 부분을 최소화 시킴

설계

역할	설계
Dao	모델에 대한 기본적인 CRUD 인터페이스
DaoImpl	Dao 인터페이스를 구현한 concrete class
Value Object (또는 ModelObject)	Dao 를 사용하여 데이터를 저장하는 단순 POJO

설계



사례 1 - 주소록

- CRUD (Create, Read, Update, Delete) 를 구현
- 주소록 관리
 - 사용자가 데이터베이스를 생성하고 데이터를 입력
 - 수정 요청
 - 데이터가 존재하는지 확인
 - 삭제 요청
 - 데이터가 존재하는지 확인

사례 1 - 주소록

□ 세 가지 버전으로 개발

■ AddressBookWithoutDao

- DAO 패턴을 사용하지 않고 직접 DB 를 조작
- 데이터베이스는 Sqlite 를 사용
- 데이터베이스 파일 이름 (MovieMedia.db)
- 테이블 구조 (Table name: MovieMedia)

■ AddressBookWithList

- DAO 패턴을 사용하지만 , DB 를 사용하는 것이 아니고 , ArrayList 를 이용해서 개발

■ AddressBookWithDao

- DAO 패턴과 DB 를 이용해서 개발

버전 1 – AddressBookWithoutDao

▣ JDBC 사용 방법

단계	설명
1	jar 파일 다운로드 및 classpath 지정
2	import java.sql.*
3	드라이버 로딩 (sqlite 는 필요없음)
4	DriverManager.getConnection()
5	Statement 를 사용해서 SQL 쿼리 사용 , ResultSet 으로 결과 받기
6	Close() 하기

버전 1 – AddressBookWithoutDao

- Sqlite jar 파일 다운로드 및 클래스 패스 지정
 - <https://bitbucket.org/xerial/sqlite-jdbc/downloads/>에서 최신 버전 다운로드
 - IDE 에 jar 파일 추가 혹은 커맨드 창에서 실행시킬 때에는 옵션으로 jar 파일을 클래스 path 에 지정
 - 예 : Main 클래스를 실행시킨다고 가정

```
java -classpath ".;./sqlite-jdbc-3.27.2.1.jar" Main
```

- IntelliJ 에 jar 파일 추가
 - File | Project Structure 선택
 - 왼쪽에서 Libraries 선택
 - + 버튼 누르고, jar 파일 찾아서 추가

버전 1 – AddressBookWithoutDao

- DriverManager.getConnection() 을 이용해서 유
파일 연결하고 SQL 쿼리를 실행시킬 수 있도록 St
atement 생성

```
Connection connection;  
Statement statement;  
  
// DB_FILE_NAME 은 데이터베이스가 들어 있는 파일 이름  
connection = DriverManager.getConnection(  
    "jdbc:sqlite:" + DB_FILE_NAME);  
  
statement = connection.createStatement();  
  
// set timeout to 30 sec.  
statement.setQueryTimeout(30);
```

버전 1 – AddressBookWithoutDao

□ 쿼리 실행

```
statement.execute("SQL Query");
```

□ 테이블 생성

이름	자료형	비고
ID	INTEGER	PRIMARY KEY
name	text	
address	text	

```
String table = " (ID INTEGER PRIMARY KEY  
AUTOINCREMENT, name text, address text)";  
statement.executeUpdate(  
    "DROP TABLE IF EXISTS " + DB_TABLE_NAME);  
statement.executeUpdate(  
    "CREATE TABLE " + DB_TABLE_NAME + table);
```

버전 1 – AddressBookWithoutDao

▣ 데이터 확인

```
ResultSet rs;  
rs = statement.executeQuery(  
    "SELECT * FROM persons");  
while (rs.next()) {  
    System.out.println("" + rs.getInt("ID")  
        + ", " + rs.getString("name") + ", "  
        + rs.getString("address"));  
}
```

버전 1 – AddressBookWithoutDao

▣ 전체 코드

```
import java.sql.*;
import java.util.Properties;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;

public class AddressBookWithoutDAO {
    final static String DB_FILE_NAME = "addressbook.db";
    final static String DB_TABLE_NAME = "persons";

    public static void main (String[] args){
        Connection connection = null;
        ResultSet rs = null;
        Statement statement = null;
    }
}
```

```
try {
    connection = DriverManager.getConnection(
        "jdbc:sqlite:" + DB_FILE_NAME);
    statement = connection.createStatement();
    statement.setQueryTimeout(30);

    final String table = " (ID INTEGER PRIMARY KEY
AUTOINCREMENT, name text, address text)";

    // create table
    statement.executeUpdate(
        "DROP TABLE IF EXISTS " + DB_TABLE_NAME);
    statement.executeUpdate(
        "CREATE TABLE " + DB_TABLE_NAME + table);

    System.out.println("--- inserting...");
    statement.execute("INSERT INTO persons(name, ad-
dress) VALUES('Seonyoung Kim','1 Hongji Dong')");
    statement.execute("INSERT INTO persons(name, ad-
dress) VALUES('Jangkwon Lee','2 Hongji Dong')");
```

```
System.out.println("--- finding all...");
rs = statement.executeQuery(
"SELECT * FROM persons WHERE id < 4 ORDER BY id");
while (rs.next()) {
    System.out.println(
        "" + rs.getInt("ID") + ", "
        + rs.getString("name") + ", "
        + rs.getString("address"));
}
```

```
System.out.println("--- updating...");
statement.execute(
    "UPDATE persons SET name = 'Sooyoung Lim'
WHERE ID = 1");
```

```
System.out.println("--- see if updated...");
rs = statement.executeQuery(
    "SELECT * FROM persons WHERE id == 1");
while (rs.next()) {
    System.out.println(rs.getInt("id") + ", "
        + rs.getString("name") + ", "
        + rs.getString("address"));
}
```


버전 1 – AddressBookWithoutDao

```
System.out.println("--- deleting...");
statement.execute(
    "DELETE FROM persons WHERE id = 1");

System.out.println("--- finding all after deleting...");
rs = statement.executeQuery(
    "SELECT * FROM persons WHERE id < 4 ORDER BY id");
while (rs.next()) {
    System.out.println(rs.getInt("id") + ", "
        + rs.getString("name") + ", "
        + rs.getString("address"));
}
}
catch (Exception e){
    e.printStackTrace();
}
```

버전 1 – AddressBookWithoutDao

```
finally {  
    try {  
        if (rs != null) { rs.close(); }  
        if (statement != null) { statement.close(); }  
        if (connection != null) { connection.close(); }  
    }  
    catch (Exception e) {  
        e.printStackTrace();  
    }  
}  
} // main  
} // class
```

버전 2 – AddressBookWithList

```
public class Person {  
    private static int counter = 1;  
    private int id;  
    private String name;  
    private String address;  
  
    Person(String name, String address){  
        this.name = name;  
        this.address = address;  
        id = counter;  
        counter++;  
    }  
  
    Person(int id, String name, String address) {  
        this.name = name;  
        this.address = address;  
        this.id = id;  
    }  
}
```

버전 2 – AddressBookWithList

```
public String toString() {  
    return "" + id + ", " + name + ", " + address;  
}  
  
public String getName() { return name; }  
  
public void setName(String name) {  
    this.name = name;  
}  
... // 다른 getter 와 setter for id and address  
}
```

버전 2 – AddressBookWithList

□ DAO 인터페이스

```
import java.util.List;

public interface PersonDao {
    public void insert(Person p);
    public List<Person> findAll();
    public Person findById(int id);
    public void update(Person p, int id);
    public void delete(int id);
    public void delete(Person p);
}
```

버전 2 – AddressBookWithList

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

public class PersonDaoImpl implements PersonDao {
    List<Person> persons;

    public PersonDaoImpl() {
        persons = new ArrayList<Person>();
    }

    public void insert(Person p) {
        persons.add(p);
    }

    public List<Person> findAll() {
        return persons;
    }
}
```

```
public Person findById(int id) {  
    int n = 0;  
    for (Person pi : persons) {  
        if (pi.getId() == id) {  
            break;  
        }  
        n++;  
    }  
    if (n >= persons.size()) {  
        return null;  
    }  
    return persons.get(n);  
}
```

```
public void update(Person p, int id) {  
    Person person = findById(id);  
    if (person != null) {  
        person.setName(p.getName());  
        person.setAddress(p.getAddress());  
    }  
}
```



버전 2 – AddressBookWithList

```
public void delete(int id) {  
    Person person = findById(id);  
    if (person != null) {  
        persons.remove(person);  
    }  
}  
  
public void delete(Person p) {  
    persons.remove(p);  
}  
}
```




버전 2 – AddressBookWithList

```
public class AddressBookWithList {  
    public static void main(String[] args) {  
        Person p;  
        PersonDao personDao = new PersonDaoImpl();  
  
        System.out.println("--- inserting...");  
        p = new Person("Seonyoung Kim", "1 Hongji Dong");  
        personDao.insert(p);  
        p = new Person("Jangkwon Lee", "2 Hongji Dong");  
        personDao.insert(p);  
  
        System.out.println("--- finding all...");  
        for (Person pi : personDao.findAll()) {  
            System.out.println("reading... " + pi);  
        }  
    }  
}
```

```
System.out.println("--- updating...");  
p = personDao.findAll().get(0);  
p.setName("Sooyoung Lim");  
personDao.update(p, p.getId());
```

```
System.out.println("--- see if updated...");  
p = personDao.findById(1);  
if (p != null) {  
    System.out.println(p);  
}
```

```
System.out.println("--- deleting...");  
personDao.delete(1); // or personDao.delete(p);
```

```
System.out.println("--- finding all after delet-  
ing...");  
for (Person pi : personDao.findAll()) {  
    System.out.println("reading... " + pi);  
}  
}  
}
```

설계

역할	설계
Dao	모델에 대한 기본적인 CRUD 인터페이스
DaoImpl	Dao 인터페이스를 구현한 concrete class
Value Object (또는 ModelObject)	Dao 를 사용하여 데이터를 저장하는 단순 POJO

설계

