토비의 스프링 3.1[ 스프링의 이해와 원리 ]

작성자: 이원태

[개요]

스프링은 객체지향 설계(object oriented design)와 구현에 관해 특정한 모델과 기법을 억지로 강요하지 않지만 오브젝트를 어떻게 효과적으로 설계하고, 구현하고, 사용하고, 이를 개선 해나갈 것인가에 대한 명쾌한 기준을 마련해준다.

1. 초난감 DAO

개요 : 관심사가 한곳에 집중되며 중복 코드 존재를 제거 하자

* 장점: 없다.
* 단점: 중복 코드 존재, 모든 변경과 발전이 한 번에 한 가지 관심사항에 집중
* 문제: 변경 요청 시 필요한 작업을 최소화하고 그 변경이 다른 곳에 문제를 일으키지 않게 할 수 있는가?
* 해결방안: 관심사의 분리(Separation of concerns) 와 중복된 코드를 메소드로 추출한다.
  + 변화가 한 번에 한가지 관심에 집중되어서 일어난다면, 우리가 준비해야 할 일은 관심이 같은 것끼리 모으고, 관심이 다른 것은 따로 떨어져 있게 하는 것
  + 객체지향에 적용한다면) 관심이 같은 것끼리 하나의 객체 안으로 하고, 관심이 다른 것은 따로 떨어져서 서로 영향을 주지 않도록 분리하는 것

* 실습 부분)
* 초난감 User DAO -> 관심사 분리 + 중복 코드의 메소드 추출한 DAO
* 개요: User DAO에는 여러 관심사가 존재하며 add()와 get()에 중복된 코드(스파게티 코드) 제거하자

[User DAO 관심사항’s]

* 1. DB와 연결을 위한 Connection을 어떻게 가져올지 관심사
  2. Param으로 넘어온 사용자 정보를 Statement에 바인딩 시키고, Statement에 담긴 SQL을 DB를 통해 실행시키는 관심사
  3. 작업이 끝나면 사용한 리소스인 Statement와 Connection 오브젝트를 닫아줘서 소중한 공유 리소스를 시스템으로 돌려주는 관심사
* 문제: 첫번째 관심사인 DB Connection을 가져오는 코드는 다른 관심사와 섞여서 add()에 있으며 동일 코드가 get()에도 중복되어 있다.
* 해결방안: 관심사 분리 + 중복 코드의 메소드 추출
  + Connection을 가져오는 중복된 코드를 분리하여 독립적인 getConnection()로 만든다.
* wrap up)
  + 여러 메소드에 중복돼서 등장하는 특정 관심사항이 담긴 초난감 DAO 클래스에서

+ 리팩토링\* 중 메소드 추출(extract method)기법을 통해 별도의 메소드로 분리

* + DB종류, 접속 방법이 변경돼도 그 관심이 집중되는 부분의 코드만 수정하면 된다.
  + 관심이 다른 코드가 있는 메소드에는 영향을 주지 않는다.

#add()에 대한 검증 주의사항

- main() 메소드를 이용한 테스트를 여러 번 실행 할 경우 테이블의 기본키id 값이 중복되므로 사전에 User 테이블의 사용자 정보를 모두 삭제 해야 한다.

1. 관심사 분리와 중복 코드의 메소드 추출한 DAO

개요 : 변화에 대응하는 수준이 아닌 변화를 반기는 DAO 만들자

장점 : 관심사 분리 와 중복 코드의 메소드 추출하여 관심이 다른 코드가 있는 메소드에는 영향을 주지 않는다.

단점 : 소스코드를 제공해주지 않고 사용자가 원하는 DB Connection 생성 방식으로 변경하고 싶을 경우 독자적으로 만들수 가 없다.

문제 : UserDao소스코드를 N사와 D사에 제공해주지 않고도 고객 스스로 원하는 DB컨넥션 생성 방식을 적용해가면서 UserDao를 사용할 수 있을까?

해결방안 : UserDAO에서 메소드 구현 코드를 제거하고 getConnection()을 추상메소드로 만들어 상속을 통해 서브클래스로 확장 한다.

wrap up)

UserDAO의 소스코드를 제공해서 수정하지 않아도 상속을 통해 getConnection() 메소드를 원하는 방식으로

확장한 후 UserDAO의 기능과 함께 사용할 수 있다.

실습 ) 메소드 추출한 DAO -> 상속을 통해 서브클래스로 분리한 DAO

개요 : 기존에 같은 클래스에 다른 메소드로 분리됐던 DB Connection 관심을 상속을 통해 자유롭게 서브클래스로 확장하자

문제 : 소스코드 공개하지 않을 경우 고객이 원하는 방식으로 코드변경 및 독자적 메소드를 만들 수 있을까?

해결방안 : 상속을 통해 확장 시킨다.

1. 슈퍼클래스인 UserDAO를 abstract 키워드를 붙쳐서 추상 클래스로 만든다.
2. 기존 관심사 메소드의 구현 코드를 제거한 후 abstract 키워드를 붙쳐서 추상 메소드로 만든다.
3. 추상 클래스이자 슈퍼클래스인 UserDAO를 상속 받아(extends) 서브클래스에서 슈퍼클래스의 추상 메소드를 정의하여 확장시킨다.

wrap up)

클래스 계층구조를 통해 두개의 관심인 1.데이터 가공 관심을 담고있는 UserDAO 2. DB연결 관심을 담고있는 NUserDAO와 DUserDAO가 독립적으로 분리되어 손쉽게 확장 할 수 있다.

* 템플릿 메소드 패턴 : 슈퍼 클래스기능의 일부를 추상 메소드나 오버라이딩이 가능한 protected 메소드 등으로 만든 뒤 서브클래스에서 필요에 맞게 구현해 사용하는 디자인 패턴\*
* 팩토리 메소드 패턴 : UserDAO의 getConnection()메소드는 Connection타입 오브젝트를 생성 한다는 기능을 정의해 놓은 추상 메소드이며 서브 클래스에서 구체적인 오브젝트 생성 방법을 결정하게 하는 디자인 패턴\*

1. 상속을 통해 서브클래스로 분리한 DAO

개요 : 상속의 단점을 없애고 두 개의 관심사를 독립시키면서 손쉽게 확장할 수 있는 클래스의 분리 방법을 알아보자

장점 : 소스코드를 제공해주지 않고 상속을 통해 사용자가 원하는 DB Connection 생성 방식으로 변경하고 싶을 경우

독립적으로 만들 수 있다.

단점 :

1. 슈퍼클래스가 다른 목적을 위해 상속을 사용하고 있다면 후에 다른 목적으로 상속(다중상속)을 적용하기 힘들다.
2. 서브 클래스는 슈퍼클래스의 기능을 직접 사용할 수 있으므로 슈퍼클래스 내부 변경 시 모든 서브클래스 수정 및 개발 해야 하는 문제와 반대로 이런 변화를 주지 않기 위해 슈퍼클래스에 제약을 가해야 하는 단점이 있다.
3. 확장된 서브 클래스의 메소드를 다른 슈퍼클래스에 적용 할 수 없다. 만약에 다른 슈퍼클래스들이 계속 만들어진다면 상속을 통해서 만들어진 서브클래스 구현 메소드가 매 슈퍼클래스마다 중복돼는 단점이 있다.

ex)

확장된 기능인 DB컨넥션을 생성하는 코드를 다른 dao 클래스에 적용 할 수 없어 UserDAO 외의 DAO 클래스들이 계속 만들어진다면 그때는 상속을 통해서 만들어진 getConnection()의 구현 코드가 매 DAO 클래스마다 중복된다.

해결방안 : 상속 관계가 아닌 완전히 독립적인 클래스로 만든다.

실습 ) 상속을 통해 서브클래스로 분리한 DAO -> 클래스 분리한 DAO

개요 ) 상속의 단점을 없애고 두 개의 관심사를 독립 시켜 확장 할 수 있도록 독립적인 클래스를 만들자

해결방안 )

1. 기존 UserDAO에서 상속 키워드인 abstract 제거한다.
2. UserDAO 생성자애서 new 키워드를 사용해 SimpleConnectionMaker 클래스의 오브젝트를 만들어두고, 이를 add(), get()메소드에서 사용한다.
3. DB 커넥션 생성 기능을 독립시킨 SimpleConnectionMaker 클래스에 DB 생성 기능을 그 안에 넣는다.

1. 클래스 분리 DAO

개요 : 클래스 분리로 생긴 종속적 문제를 인터페이스를 통해 해결하자

장점 : 두 개의 관심사를 독립시키면서 손쉽게 확장할 수 있고 두 관심사에 대한 결합도가 낮아 코드 수정이 편하다.

단점 : UserDAO의 코드가 SImpleConnectionmaker 클래스에 종속되어 있기 때문에 UserDAO 코드 변경없이 DBConnection 변경 불가능하다.

1. SImpleConnectionmaker 의 메소드인 makeNewConnection()을 사용하고 있지만 만약에 D사에서 openConnection()이라는 메소드 이름을 사용했다면 UserDAO안의 코드를 다 수정 해야 한다.

2. UserDAO가 DB커넥션을 가져오는 클래스와 메소드에 종속적이기 때문에 확장이 힘들다.

(종속된 부분 : SImpleConnectionmaker. makeNewConnection();)

해결방안 : 인터페이스를 통해 두 개의 클래스가 서로 연결되어 있지 않도록 중간에 추상적인\* 연결고리를 만들어 준다.

wrap up : DB커넥션 인터페이스를 사용하는 UserDAO 입장에서는 어떤 클래스로 만들어졌는지 상관 없이 메소드 사용하여 확장이 편리해졌다

실습 ) 클래스 분리 DAO -> 인터페이스 도입 DAO

개요 : 인터페이스를 이용하여 db커넥션을 제공하는 클래스에 대한 구체적인 정보를 제거 하자

해결방안 )

1. ConnectionMaker 인터페이스에 DB커넥션으르 가져오는 메소드(makerConnection)를 정의 한다.
2. ConnectionMaker 인터페이스를 구현한 클래스를 만들고 DB컨넥션을 가져오는 메소드(makerConnection)를 작성한다.
3. UserDAO의 add(),get() 메소드와 필드에 ConnectionMaker라는 인터페이스와 makerConnection메소드를 사용한다.

wrap up :

1. 기존 클래스 분리 할 경우 소스 코드 공개 없이 확장이 불가능 했지만 interface를 이용하여 원하는 방식대로 확장이 가능해 졌다.

2. DB커넥션 클래스를 확장하여도 UserDAO의 코드를 고칠 일 없다.

5. 인터페이스 분리 DAO

개요 : DB커넥션에 대한 종속은 제거 했지만 UserDAO에 어떤 클래스의 오브젝트를 사용할지 결정하는 생성자 코드를 제거 하자

장점 : DB커넥션에 대해 확장이 가능하며 UserDao의 코드를 고칠 일이 없다.

단점 : UserDao에 어떤 ConnectionMaker 구현 클래스를 사용 할지를 결정하는 코드가 남아있어 UserDao 변경 없이 DB컨넥션 기능의 확장이 자유롭지 못하다. (new DConnectionMaker(); 존재)

해결방안 : 관계 설정 책임의 분리

1. UserDao와 ConnectionMaker의 특정 구현 클래스 사이의 관계를 설정해주는 관심을 UserDao의 클라이언트 오브젝트에서 분리 시켜준다.

(UserDao와 ConnectionMaker사이의 관계 설정 관심 : connectionMaker = new DConnectionMaker(); )

2. UserDao오브젝트와 ConnctionMaker 오브젝트 사이에 관계를 설정 해준다.

3. 관계 설정은 런타임 시에 한쪽이 다른 오브젝트의 레퍼런스를 갖고 있는 방식으로 만든다.

(main() : connectionMaker = new DConnectionMaker(); ('사용'관계)

4.UserDao에서 connecitonMaker와의 관계 설정을 만들지 않고 생성자 파라미터나 메소드 파라미터를 이용해 외부에서 받아온다.

Public UserDao(ConectionMaker connectionMaker){

This.connectionMaker = connectionMaker;

}

5. 파라미터로 제공받은 connectionMaker오브젝트는 인터페이스에 정의된 메소드를 이용한다.

다형성 ) 코드에서는 특정 클래스를 전혀 알지 못하더라도 해당 클래스가 구현한 인터페이스를 사용 했다면, 그 클래스의 오브젝트를 인터페이스 타입으로 받아서 사용 할 수 있다.

wrap up :

1. UserDao 클래스의 main() 메소드가 UserDao 클라이언트 이다.

2. UserDao와 DConnectionMaker 오브젝트사이에 관계를 만들어서 사용한다. (=클라이언트의 책임)

3. 클라이언트(main)는 ConnectionMaker의 구현 클래스를 선택하고, 선택한 클래스의 오브젝트를 생성해서 UserDao와 연결한다.

ConnectionMaker connectionmaker = new DConnectionMaker(); // connectionmaker의 구현 클래스 선택+오브젝스생성

userDao dao = new UserDao(connectionMaker); // userDao와 연결

4. userDao의 생성자를 수정해서 ConnectionMaker의 오브젝트를 전달 받을 수 있도록 파라미터 추가 한다.

\* 기존 userDao에는 어떤 connectionMaker 구현 클래스를 사용 할지 결정하는 코드가 있었지만 그것을 클라이언트에게 넘겨서 제거했다.

=> UserDao의 변경 없이도 자유롭게 DBConection 기능을 확장하여 UserDao에서 사용할 수 있다.

[최종]

 [ConnectionMaker 인터페이스]

public interface ConnectionMaker{

public Connection makeConnection() throws ClassNotFoundException,SQLException;

}

[ConnectionMaker 구현 클래스]

public class DConnectionMaker implements ConnectionMaker{

public Connection makerConnection() throws ClassNotFoundException,SQLException{

//D사의 독자적인 방법으로 connectiond을 생성하는 코드

}

[UserDao 클라스]

public class UserDao{

private ConnectionMaker connectionMaker;

public UserDao(ConnectionMaker connectionMaker){

this.connectionMaker = connectionMaker;

 }

public void add(User user) throws ClassNotFoundException, SQLException{

Connection c = connecionMaker.makeConnection();

}

public void get(User user) throws ClassNotFoundException, SQLException{

Connection c = connecionMaker.makeConnection();

}

public static void main(String[] args) throws ClassNotFoundException, SQLException

{

ConnectionMaker connectionMaker = new DConnectionMaker();

UserDao dao = new UserDao(connectionMaker) ;

….

}

 }

1.UserDao가 실제로 사용할 DConnectionMaker 오브젝트 생성한다.

2.UserDao 오브젝트를 생성하면서 생성자를 통해 DConnectionMaker 오브젝트를 전달한다.

(생성자는 ConnectionMaker 인터페이스 타입으로 전달 받기 때문에 ConnectionMaker인터페이스를 구현 했다면 상관 없다 )

3. 필요한 오브젝트 생성하고 관계를 맺어주는 준비 작업은 끝났다.

4. 테스트 할 경우 UserDao의 add 와 get 메소드를 사용하며 이때 UserDao는 생성자를 통해 전달 받아서 인스턴스 변수에 저장해둔

DconnectionMaker 오브젝트의 makeConnection()메소드를 호출해서 DB 컨넥션을 생성해 사용 한다.

=> UserDao와 ConnectionMaker 클래스를 분리하고 서로 영향을 주지 않으면서도 필요에 따라 자유롭게 확장이 가능한 구조이다.

[원칙과 패턴 - 걔방 폐쇄 원칙 ( open-closed principle) ]

: 깔끔한 설계를 위해 적용 가능한 객체지향 설계 원칙 중 하나다.

* > 클래스나 모듈은 확장에는 열려 있어야 하고 변경에는 받혀 있어야 한다.
* > 인터페이스를 사용해 확장 기능을 정의한 대부분의 API는 개방폐쇄원칙을 따른다.

(Ex: UserDao 는 DB 연결 방법이라는 기능을 확장하는데 열려 있고 이런 변화에 UserDao 자신의 핵심 기능을 구현한 코드는 영향을 받지 않고 유지 할 수 있으므로 변경에는 닫혀있다. )

객체지향 설계 원칙(SOLID) : 객체지향의 특징을 잘 살릴 수 있는 설계의 특징 + 로머트 마틴이 정리함

* 개방 폐쇄 원칙(o) : 높은 응집도와 낮은 결합도(high coherence and low coupling) 라는 소프트웨어 개발의 고전적인 원리로도 설명이 가능하다.
  + 높은 응집도 : 변화가 일어날 때 해당 모듈에서 변하는 부분이 큰 것
    - ex) 초난감 DAO에서 ConnectionMaker인터페이스를 이용해 DB연결 기능을 독립시킨 경우, ConnectionMaker 구현 클래스를 새로 만들던지, DconnectionMaker를 수정 하더라도 UserDao 등 다른 클래스의 기능에 영향을 주기 않는다는 사실에 응집도가 높아진 이유다.
  + 낮은 결합도 : 책임과 관심사가 다른 오브젝트 또는 모듈과는 느슨하게 연결된 형태를 유지하는 것이 바람직하다.
    - 하나의 변경이 발생할 때 다른 모듈과 객체로 변경에 대한 요구가 전파되지 않는 상태를 말한다.

ex)

1. ConnectionMaker 인터페이스의 도입으로 인해 DB 연결 기능을 구현한 클래스가 바뀌더라도 DAO의 코드는 변경될 필요가 없다.
2. ConnectionMaker의 클래스를 결정하는 책임을 DAO의 클라이언트로 분리한 덕분에 사용할 ConnectionMaker 구현 클래스가 바뀌어도, DAO클래스의 코드를 수정할 필요가 없다.

= > UserDao와 ConnectionMaker 는 각각 자신의 책임에 대한 응집도가 높고 둘의 사이는 인터페이스로 인해 서로 알 필요가 없는 느슨한 구조로 결합도가 낮다.

* 원칙 : 예외는 있겠지만 대부분의 상황에 잘 들어 맞는 가이드라인
* 디자인 패턴 : 특별한 상황에서 발생하는 문제에 대한 좀더 구체적인 솔루션
* 객체지향 설계 원칙 : 일반적인 상황에서 적용 가능한 설계 기준
* SRP(s) : 단일 책임 원칙
* OCP(o) : 개방 폐쇄 원칙
* LSP(l) 리스코프 지환 원칙
* ISP(i) : 인터페이스 분리 원칙
* DIP(d) : 의존관계 역전 원칙

[전략 패턴]

* 개선한 UserDaoTest - UserDao - ConnectionMaker 구조를 디자인 패턴의 시각으로 전략 패턴에 해당한다.
* 자신의 기능 맥락(context)에서 필요에 따라 변경이 필요한 알고리즘을 인터페이스를 통해 통째로 외부로 분리시키고(ConnectoinMaker) 이를 구현한 구체적인 알고리즘 클래스를 (DConnectionMaker) 필요에 따라 바꿔서 사용할 수 있게 하는 디자인 패턴
* 전력 패턴은 UserDaoTest와 같은 클라리언트의 필요성에 대해서도 잘 설명 하고 있다.
  + 컨텍스트(UserDao)를 사용하는 클라리언트(UserDaoTest)는 컨텍스트(UserDao)가 사용할 전략(DConnectionMaker)을 컨텍스트(UserDao)의 생성자 등을 통해 제공해주게 일반적이다.

\*DAO(data access object) : DB사용해 데이터를 조회하거나 조작하는 기능을 전담하도록 만든 오브젝트를 말한다.

\*관심사 : 컴퓨터 프로그램 코드에 영향을 미치는 정보의 집합

\*객체지향 기술 : 가상의 추상 체계를 효과적으로 구성할 수 있고, 이를 자유롭고 편리하게 변경, 발전, 확장 시킬 수 있다.

=> wrap up) 변화에 효과적으로 대처할 수 있다.s

\*스파게티 코드 : 컴퓨터 프로그램의 소스 코드가 복잡하게 얽힌 모습을 스파게티의 면발에 비유 + 정상적으로 작동하지만, 사람이 코드를 읽으면서 그 코드의 작동을 파악하기는 어렵다

\*리펙토링 : 기존의 코드를 외부의 동작방식에는 변화 없이 내부 구조를 변경해서 재구성하는 작업 또는 기술을 말한다.

* 효과 : 나쁜 냄새(중복 코드) 코드 내부의 설계가 개선되어 유지보수 용이, 코드의 품질 향상

\*추상 메소드 : 메소드 코드는 없지만 메소드 자체는 존재 + abstract로 구성

\*디자인 패턴 : SW 설계 시 특정 사항에서 자주 만나는 문제를 객체지향적인 설계로 해결하기 위해 적용 할 수 있는 확장성 추구와 재사용이 가능한 상속 과 오브젝트 합성 솔루션

\*추상화 : 어떤 것들의 공통적인 성격을 뽑아내어 이를 따로 분리해내는 작업 s