객체 지향 설계(Object Oriented Design)의 정수 SOLID (로버트 C. 마틴)

SRP(Single Responsibility Principle): 단일 책임 원칙

어떤 클래스를 변경해야 하는 이유는 오직 하나뿐이어야 한다. - 추상화

너무 많은 역할 X

(잘된 경우보다 잘못된 경우를 살펴보는 것이 이해하는 데 좋다.)

이를 잘 지키지 못하면 if문을 남발하게 된다. ex) [토지], 면적, 건물데이터베이스 테이블이 원칙을 지키지 않아서 발생한 경우이지만

테이블 설계할 때도 단일 책임 원칙을 고려해야 했다. p180

OCP(Open Closed Principle): 개방 폐쇄 원칙

소프트웨어 엔티티(클래스, 모듈, 함수 등)는 확장에 대해서는 열려 잇어야 하지만 변경에 대해서는 닫혀 있어야 한다.

= 자신의 확장에는 열려 있고, 주변의 변화에 대해서는 닫혀 있어야 한다.

(다양한 곳에서 이야기되고 있어 딱 꼬집어서 예를 들기가 쉽지 않다. 책에서는 여러 예를 들었다. 포인트는 인터페이스)

다양한 자동차가 생긴다고 하는 것은 자동차 입장에서는 자신의 확장에 개방돼 있는 것이고, 운전자 입장에서는 주변의 변화에 폐쇄돼 있는 것이다. P.184 그림 확인.

ex) JDBC 자바 애플리케이션은 JDBC 인터페이스라고 하는 완충 장치로 인해 변화에 영향을 받지 않는다.(주변의 변화에 닫혀 있다.)

데이터베이스를 교체한다는 것은 데이터베이스가 자신의 확장에는 열려 있다는 것이다.(p.184)

-->개방 폐쇄 원칙을 따르지 않는다고 해서 객체 지향 프로그램을 구현하는 것이 불가능한 것은 아니지만 개방 폐쇄 원칙을 무시하고 프로그램을 작성하면 객체 지향 프로그래밍의 가장 큰 장점인 유연성, 재사용성, 유지보수성 등을 얻을 수 없다. 따라서 객체 지향 프로그래밍에서 개방 폐쇄 원칙은 반드시 지켜야 할 원칙이다.(스프링 프레임워크 또한 개방 폐쇄 원칙의 좋은 예이다.)

LSP(Liskov Substitution Principle): 리스코프 치환 원칙

서브 타입은 언제나 자신의 기반 타입(base type)으로 교체할 수 있어야 한다. - 객체 지향의 상속이라는 특성을 올바르게 사용하면 자연스럽게 얻게 되는 것.

=하위 클래스의 인스턴스는 상위형 객체 참조 변수에 대입해 상위클래스의 인스턴스 역할을 하는 데 문제가 없어야 한다.

=하위 클래스 is a kind of 상위 클래스

means 상속X 확장O

(리스코프 치환 원칙은 말로 기억하기보다는 그림을 통해 기억하는 것이 편하다. 계층도/조직도(X), 분류도(O) p.188

ex) 아버지 춘향이 = new딸()

딱 봐도 이상하지 않은가? 딸을 하나 낳아서 이름을 춘향이라 한 것까지는 좋은데 아빠의 역할을 맡기고 있다.

리스코프 치환 원칙은 객체 지향의 상속이라는 특성을 올바르게 활용하면 자연스럽게 얻게 되는 것이다.

ISP(Interface Segregation Principle): 인터페이스 분리 원칙

클라이언트는 자신이 사용하지 않는 메서드에 의존 관계를 맺으면 안 된다.

-->단일 책임 원칙과 인터페이스 분할 원칙 은 같은 문제에 대한 두 가지 다른 해결 책이라고 볼 수 있다.

하지만 특별한 경우가 아니라면 단일 책임 원칙을 적용 하는 것이 더 좋은 해결책이라고 할 수 있다.

인터페이스 분할 원칙을 이야기할 때 항상 함께 등장하는 원칙 중 하나로 인터페이스 최소주의 원칙이라는 것이 있다.

DIP(Dependency Inversion Principle): 의존 역전 원칙

고차원 모듈은 저차원 모듈에 의존하면 안 된다. 이 두 모듈 모두 다른 추상화된 것에 의존해야 한다.

추상화된 것은 구체적인 것에 의존하면 안 된다. 구체적인 것이 추상화된 것에 의존해야 한다.

자주 변경되는 구체(concrete)클래스에 의존하지 마라. -로버트 C.마틴

ex) 자동차 --> 스노우타이어 , 자동차와 스노우타이어 사이에는 의존 관계까 있다.

public class SnowTire{

public String runSomething() {

return "스노우 타이어 장착";

}

}public class Car {

public static void main(String[] args) {

SnowTire tire = new SnowTire();

System.out.println(tire.runSomething());

}

}

(의존 역전) 자동차 --> 타이어(추상interface) <-- 스노우타이어, 일반타이어, 광폭타이어

설명) 스토우 타이어는 계절이 바뀌면 일반 타이어로 교체해야 한다. 이런 경우 스노우타이어를 일반타이어로 교체할 때

자동차는 그 영향에 노출돼 있음을 알 수 있다. 바로 자동차 자신보다 더 자주 변하는 스노우타이어에 의존하기에 부서지기 쉬움이라는

나쁜 냄새를 풍기고 있는 것이다.

기존에는 스노우타이어가 그 무엇에도 의존하지 않는 클래스였는데, 추상적인 것인 타이어 인터페이스에 의존하게 됐다. 바로 의존 방향이 역전된 것이다.

(개방 폐쇄 원칙과 그 맥락을 같이 하기도 한다.)

SOLID에 관한 더 많은 정보(p177~178)

UML 실전에서는 이것만 쓴다 6장

ZDnet "객체 지향 SW 설계 원칙" 기사 참고.

--> 응집도는 높이고(High Cohesion), 결합도는 낮추라(Loose Coupling)는 고전 원칙을 객체 지향의 관점에서 재정립한 것.(하늘에서 뚝 떨어진 것이 아님)

http://toby.epril.com/?p=727

결합도(coupling)는 ‘클래스간의 서로 다른 책임들이 얽혀 있는 상호 의존도의 정도

결합도가 낮으면 모듈 간의 상호 의존서잉 줄어들어 객체의 재사용이나 수정, 유지보수가 용이하다.

응집도(cohesion)는 ‘하나의 클래스가 하나의 기능(책임)을 온전히 순도 높게 담당하고 있는 정도

응집도가 높은 모듈은 하나의 책임에 집중하고 독립성이 높아져 재사용이나 기능의 수정, 유지보수가 용이하다.

스무살 때는 나이먹는 것이 좋았다. 서른에 오니 나이 먹는 것이 싫다.

복습하면서 느낀 점은 게시판 구현은 툴사용에 불과한 것은 아닐까 란 생각을 했다.

핵심은 객체지향의 설계원칙과 스프링의 사상을 읽히는 데 중점을 두어야 하는 것은 아닐까.

(ㄴㄴ 모든 원칙이 다 적용 되는 거야.(게시판 구현은 지금까지 니가 공부했던 모든 것을 적용하는거야.)

책은 제목처럼 자바 객체지향의 원리에 대해서 다룬 뒤에 스프링에 대한 약간의 기능 구현을 한다.

이 책이 좋았던 것은 자바 문법에 맞추어 코딩하기 급급했던 나에게 프로그래밍에 대한 새로운 시각을 갖을 수 있도록 한 점이다.

책의 전반부는 객체지향 프로그래밍에 대해 올바른 이해를 위한 설명이 주를 이룬다. 후반부에는 객체지향을 바탕으로 스프링 프레임워크에 대한 설명을 한다.(p.31)

객체지향을 설명할 때 좋았던 점.

기존의 설명의 잘못된 점과 새로운 이해 ex) 상속

자바 언어에 대한 설명 ex)다중 상속p.117 -->인터페이스

그럼으로서 자바를 더 잘 이해할 수 있게 된다.(C# C++과 비교를 하면서)

객체지향을 비유를 통해 명쾌한 설명

도구(객체지향) 도구를 올바르게 사용(설계 원칙) 레시피(디자인 패턴) 비유가 이것들을 이해하기 아주 명쾌했다.

구체적인 내용은 책을 통해 확인하도록 하자.(저작권 보호)