SOLID

객체 지향 설계 5원칙

장정훈

SOLID?

- SRP(Single Responsibility Principle) – 단일 책임 원칙
- OCP(Open Closed Principle) – 개방 폐쇄 원칙
- LSP(Liskov Substitution Principle) – 리스코프 치환 원칙
- ISP(Interface Segregation Principle) – 인터페이스 분리 원칙
- DIP(Dependency Inversion Principle) – 의존 역전 원칙

SOLID

- 응집도는 높이고 결합도는 낮추라는 고전 원칙을 객체 지향의 관점에서 재정립한 것이라 할 수 있음.

- 이렇게 설계 된 소프트웨어는 재사용이 많아지고 수정이 최소화 되기 때문에 유지 보수가 용이.

SRP – 단일 책임 원칙

- "어떤 클래스를 변경해야 하는 이유는 오직 하나뿐이어야 한다."

개발자
- Spring 백엔드 개발()
- JSP 개발()
- Vue.JS 개발()
- WAS 설정()
- DB 설계()
- DB 유지보수()

백엔드 개발자
- Spring 백엔드 개발()

서버사이드 개발자
- JSP 개발()

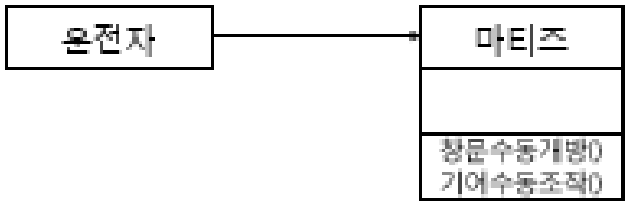
클라이언트사이드 개발자
- Vue.JS 개발()

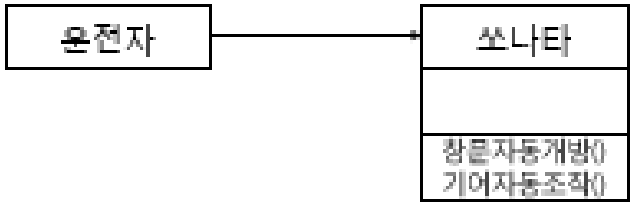
인프라 개발자
- WAS 설정()

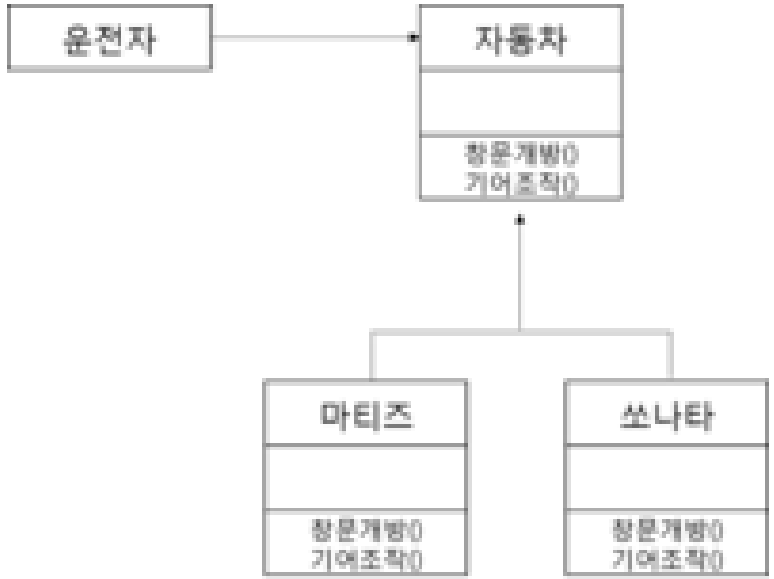
DBA
- DB 설계()
- DB 유지보수()

OCP – 개방 폐쇄 원칙

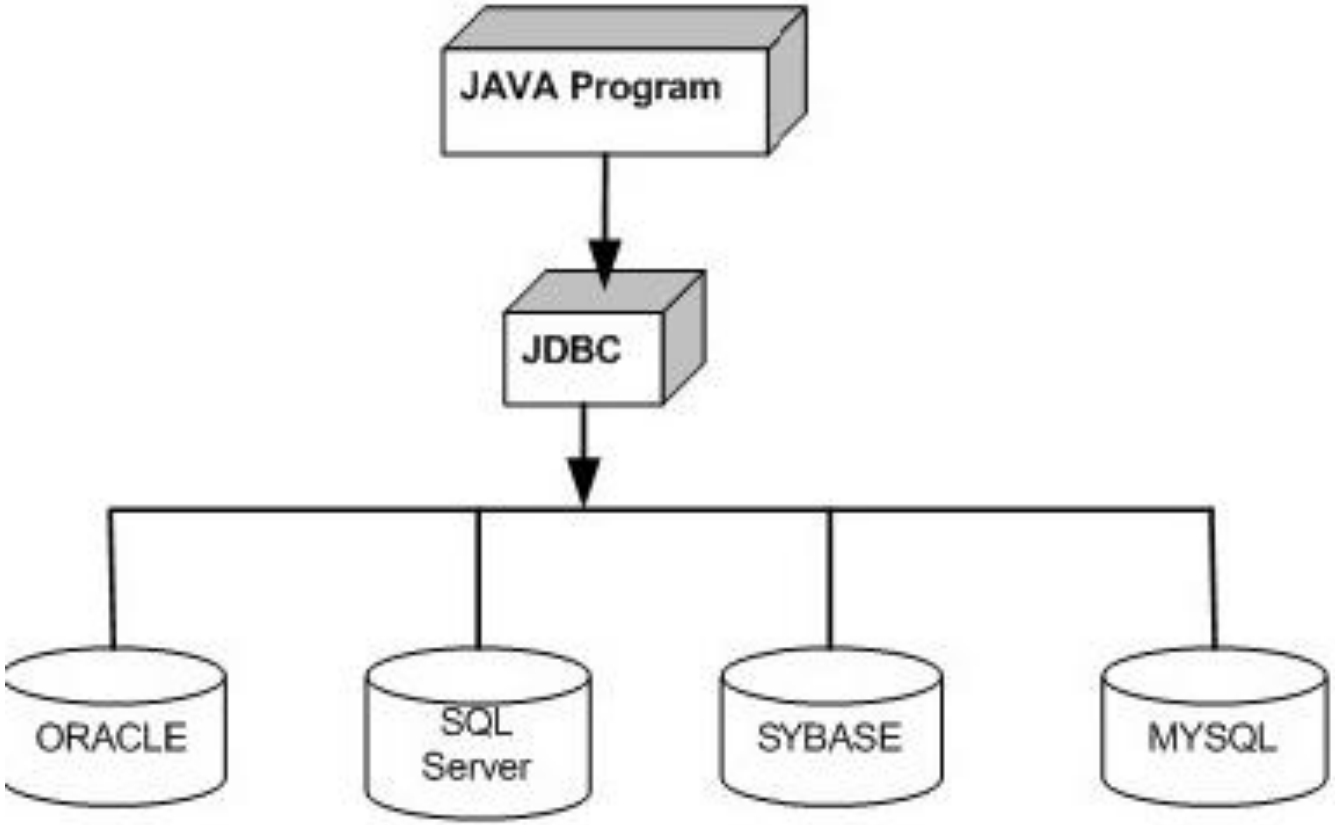
- "소프트웨어 엔티티(클래스, 모듈, 함수 등)는 확장에 대해서는 열려 있어야 하지만 변경에 대해서는 닫혀 있어야 한다."







OCP – 개방 폐쇄 원칙



LSP – 리스코프 치환 원칙

- “서브 타입은 언제나 자신의 기반 타입으로 교체할 수 있어야 한다.”

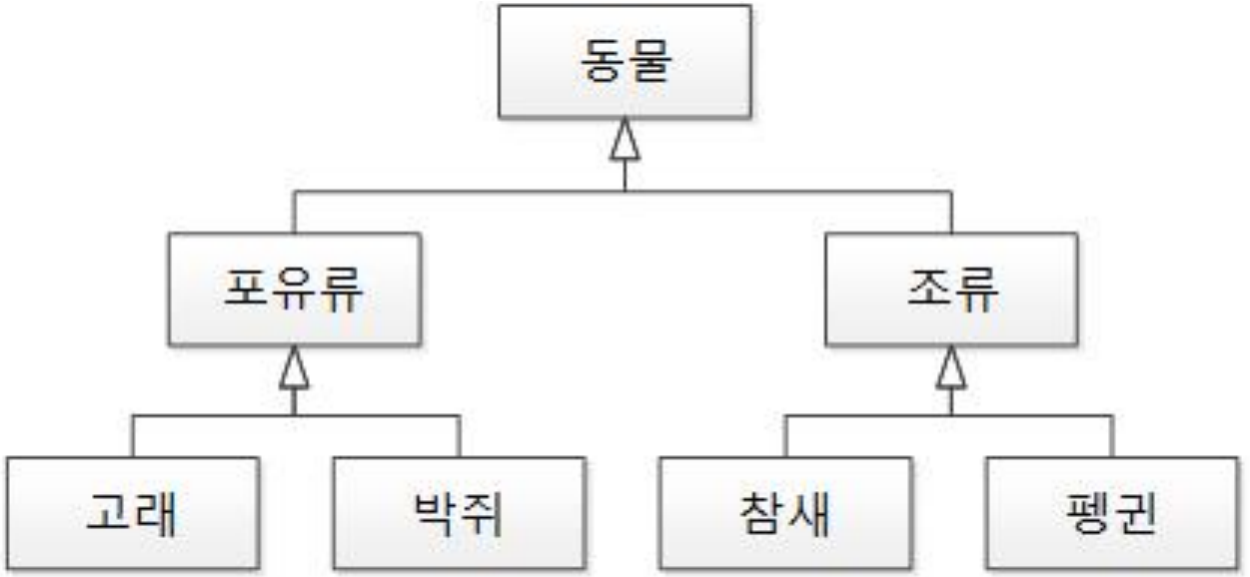
- 하위 클래스 is a kind of 상위 클래스
  - 하위 분류는 상위 분류의 한 종류이다.

- 구현 클래스 is able to 인터페이스
  - 구현 분류는 인터페이스할 수 있어야 한다.

LSP – 리스코프 치환 원칙

리스코프 치환 원칙 위배 예시

LSP – 리스코프 치환 원칙



리스코프 치환 원칙 위배 예시

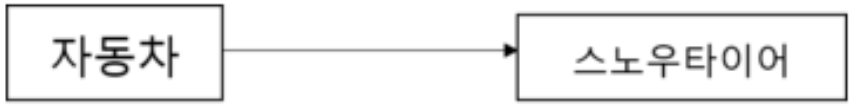
ISP – 인터페이스 분리 원칙

- “클라이언트는 자신이 사용하지 않는 메소드에 의존 관계를 맺으면 안 된다.”

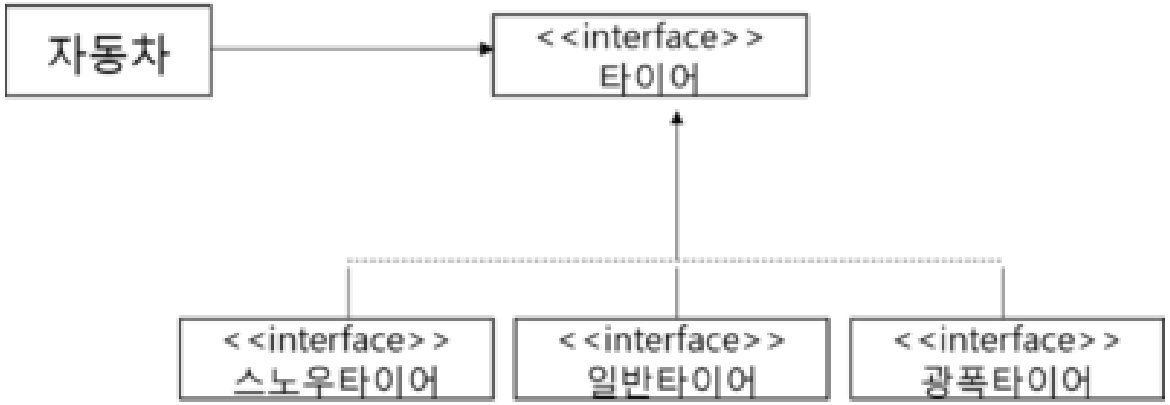
- 단일 책임 원칙(SRP)과 인터페이스 분할 원칙(ISP)은 같은 문제에 대한 두 가지 다른 해결책

DIP – 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"



의존 역전 원칙 적용 전
(자주 변경되는 구체 클래스에 의존)



의존 역전 원칙 적용 후

- SRP(단일 책임 원칙)
  - 어떤 클래스를 변경해야 하는 이유는 오직 하나뿐이어야 한다.
- OCP(개발 폐쇄 원칙)
  - 자신의 확장에는 열려 있고, 주변의 변화에 대해서는 닫혀 있어야 한다.
- LSP(리스코프 치환 원칙)
  - 서브 타입은 언제나 자신의 기반 타입으로 교체할 수 있어야 한다.
- ISP(인터페이스 분리 원칙)
  - 클라이언트는 자신이 이용하지 않는 메서드에 의존 관계를 맺으면 안 된다.
- DIP(의존 역전 원칙)
  - 자신보다 변하기 쉬운 것에 의존하지 마라