05. 객체 지향 설계 5원칙 - SOLID

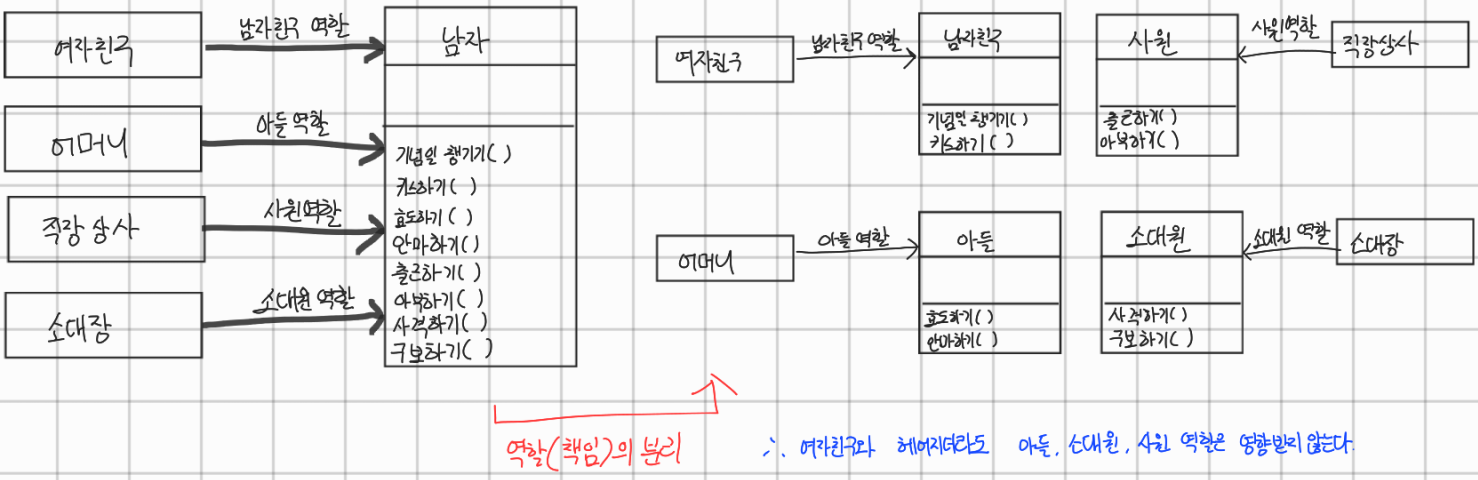
1. SRP(Single Responsibility Rrinciple): 단일 책임 원칙
2. OCP(Open Closed Principle) : 개발 폐쇄 원칙
3. LSP(Liskov Subsitution Principle) : 리스코프 치환 원칙
4. ISP(Interface Segregation Principle) : 인터페이스 분리 원칙
5. DIP(Dependency Inversion Principle) : 의존 역전 원칙

-> 응집도는 높이고 결합도는 낮춰라

- 결합도는 모듈간의 상호 의존 정도로서 결합도가 낮으면 모듈 간의 상호 의존성이 줄어들어 객체의 재사용이나 수정, 유지보수가 용이하다
- 응집도는 하나의 모듈 내부에 존재하는 구성 요소들의 기능적 관련성으로 응딥도가 높은 모듈은 하나의 책임에 집중하고 독립성이 높아져 재사용이나 기능의 수정, 유지보수가 용이하다

SRP - 단일 책임 원칙

> 어떤 클래스를 변경해야 하는 이유는 오직 하나뿐이어야 한다




```java
class 강아지 {
  final static Boolean 수컷 = true;
  final static Boolean 암컷 = false;
  Boolean 성별;

  void 소변보다() {
    if(this.성별 == 수컷) {
      // 한쪽 다리를 들고 소변을 본다
    } else {
      // 뒷다리 두 개를 굽혀 앉은 자세로 소변을 본다
    }
  }
}
```

```java
abstract class 강아지 {
  abstract void 소변보다()
}

class 수컷강아지 extends 강아지 {
  void 소변보다() {
    // 한쪽 다리를 들고 소변을 본다
  }
}

class 암컷강아지 extends 강아지 {
  void 소변보다() {
    // 뒷다리 두 개를 굽혀 앉은 자세로 소변을 본다
  }
}
```

소변보다() 메서드가 분기처리 되어있음을 볼 수 있다
하나의 메서드가 두 가지 행위를 구현하려고 한 모습이다
-> 단일 책임 원칙을 위배하고 있는 것이다

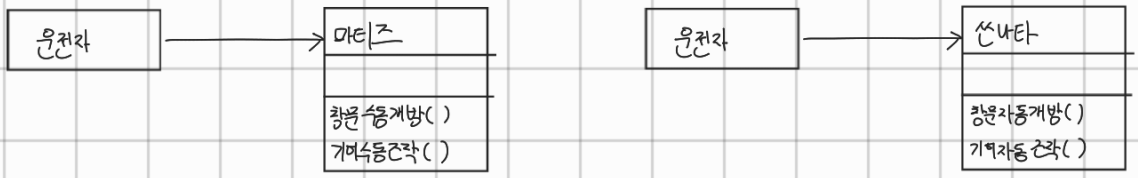
```java
class 사람 {
  String 군번;
  ...
}

...

사람 로미오 = new 사람();
사람 줄리엣 = new 사람();

줄리엣.군번 = "1573042009";
```

→ 줄리엣이 군번 속성을 갖고 값을 할당 받는 것을 제재할 방법이 없다.

OCP - 개방 폐쇄 원칙

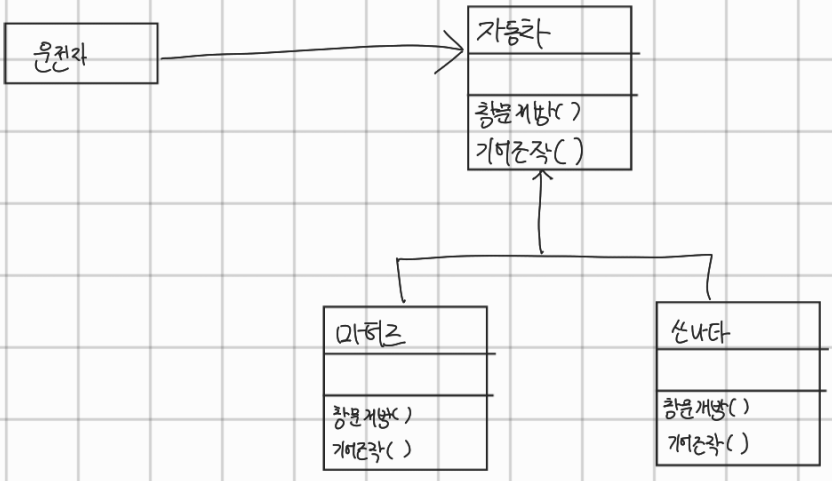
〉 소프트웨어 엔티티(클래스, 함수, 모듈 등)는 확장에 대해서는 열려 있어야 하지만 변경에 대해서는 닫혀 있어야 한다

〉 자신의 확장에는 열려 있고, 주변의 변화에 대해서는 닫혀있어야 한다



⇒ 자동차의 동작이 변경되어 운전자 행동에도 영향이 생김.

운전자는 자동차의 변화에 열려있어야 한다 = 영향을 받으면 안된다.



LSP - 리스코프 치환 원칙

〉 서브 타입은 언제나 자신의 기반 타입(base type)으로 교체할 수 있어야 한다

- 상속은 조직도나 계층도가 아닌 분류도가 돼야 한다
  * 하위 클래스 is a kind of 상위 클래스 - 하위 분류는 상위 분류의 한 종류이다
  * 구현 클래스 is able to 인터페이스 - 구현 분류는 인터페이스 할 수 있어야한다
    -〉인터페이스 할 수 있어야 한다
      1. AutoCloseable - 자동으로 닫힐 수 있어야한다
      2. Appendasble - 덧붙일 수 있어야한다
      3. Cloneable - 복제할 수 있어야한다
      4. Runnable - 실행할 수 있어야한다

아버지 춘향이 = new 딸();

-〉 딸을 생성해서 이름을 춘향이라고 한 것까지는 좋은데 아빠의 역할을 맡기고 있다

동물 뽀로로 = new 펭귄();

-〉 펭귄을 생성해 뽀로로라 이름을 짓고 동물의 역할을 맡긴다

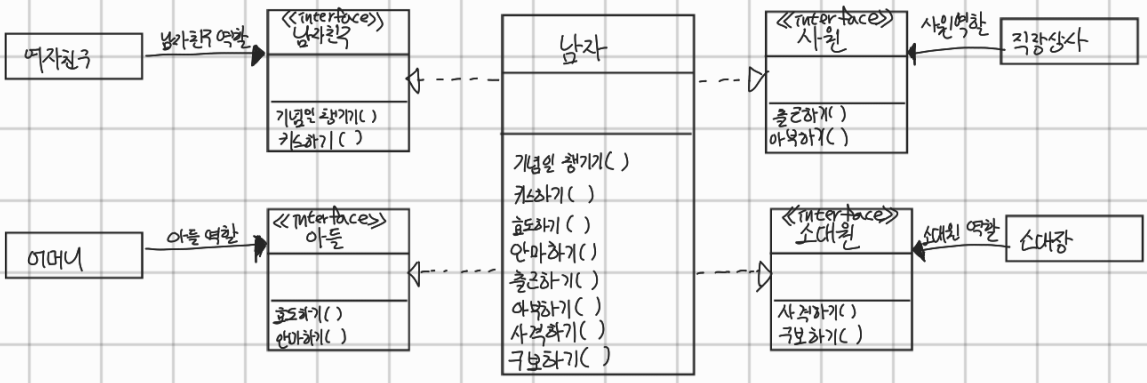
=〉 리스코프 치환 법칙을 만족한다

따라서,

〉 하위 클래스의 인스턴스는 상위형 객체 참조 변수에 대입해 상위 클래스의 인스턴스 역할을 하는 데 문제가 없어야 한다

ISP - 인터페이스 분리 원칙

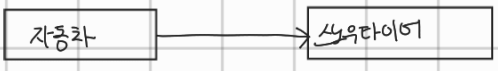
〉 클라이언트는 자신이 사용하지 않는 메서드에 의존 관계를 맺으면 안된다



결론적으로 단일 책임 원칙(SRP)과 인터페이스 분할 원칙은 같은 문제에 대한 두 가지 다른 해결책이라고 볼 수 있다
프로젝트 요구사항과 설계자의 취향에 따라 단일 책임 원칙이나 인터페이스 분할 원칙 중 하나를 선택해서 설계할 수 있다
하지만 특별한 경우가 아니라면 단일 책임 원칙을 적용하는 것이 더 좋은 해결책이라고 할 수 있다

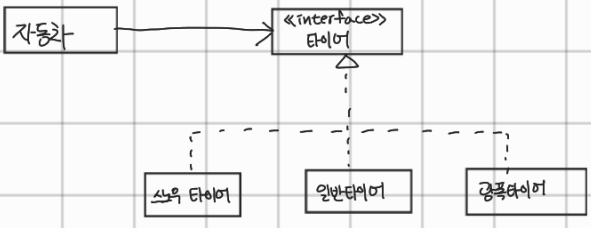
DIP - 의존 역전 원칙

〉 고차원의 모듈은 저차원 모듈에 의존하면 안 된다
이 두모듈 모두 다른 추상화된 것에 의존해야 한다
〉 추상화된 것은 구체적인 것에 의존하면 안된다
구체적인 것이 추상화된 것에 의존해야 한다
〉 자주 변경되는 구체(Concrete) 클래스에 의존하지 마라



↳ 자동차가 스노우타이어에 의존한다
스노우타이어는 계절이 바뀌면 교체해야 하고
이것에 자동차는 영향을 받는다.

자신보다 더 자주 변경되는 스노우타이어에 의존하게끔
프로그램하기 쉬웠다는 냄새를 풍기고 있다.



↳ 이전에는 스노우 타이어에 의존하는 것이 있었는데
타이어 인터페이스에 의존하게 되었다.

이처럼 자신보다 변하기 쉬운 것에 의존하던 것을
추상화된 인터페이스나 상위클래스를 두어
변하기 쉬운 것의 변화에 영향을 받지 않게하는 것

〉 자신보다 변하기 쉬운 것에 의존하지 마라