Interface Segregation Principle

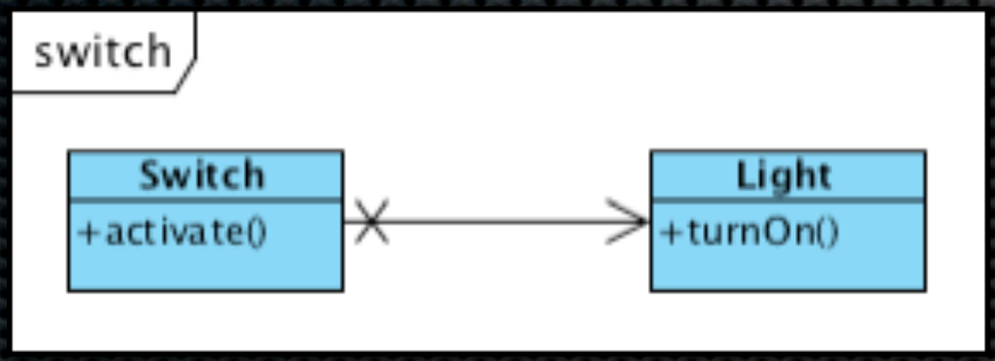
Daum Corp.
백명석

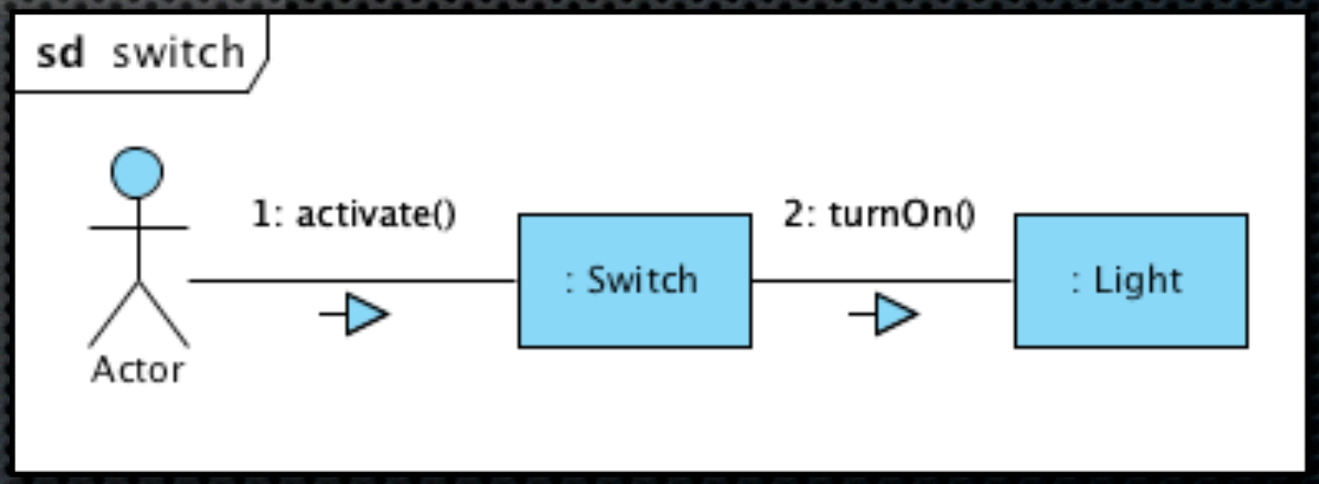
Interface Segregation Principle

- Don't depend on things that you don't need
- 사용하지 않지만 의존성을 가지고 있다면
  - 그 인터페이스가 변경되면 재컴파일/빌드/배포됨
  - 독립적인 개발/배포가 불가하다는 의미
- SRP와도 연관
  - 한 기능에 변경이 발생하면 다른 기능을 사용하는 클라이언트들에도 영향을 미침
- 사용하는 기능만 제공하도록 인터페이스를 분리함으로써 한 기능에 대한 변경의 여파를 최소화
- 클라이언트 입장에서 인터페이스를 분리하라는 원칙

Switch 예제

- switch가 activate되면 light를 turn on하는 SW를 설계하라.





Switch 예제

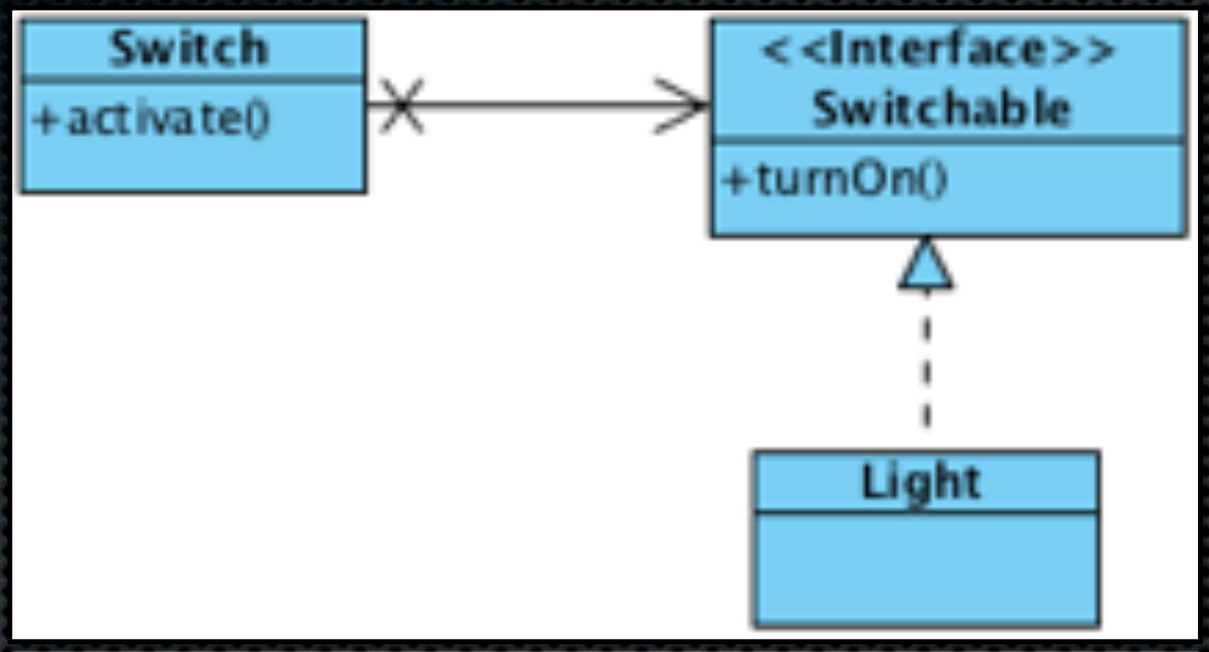
```
public class Switch {
    private Light light;

    public Switch(Light light) {
        this.light = light;
    }

    public void activate() {
        light.turnOn();
    }
}
```

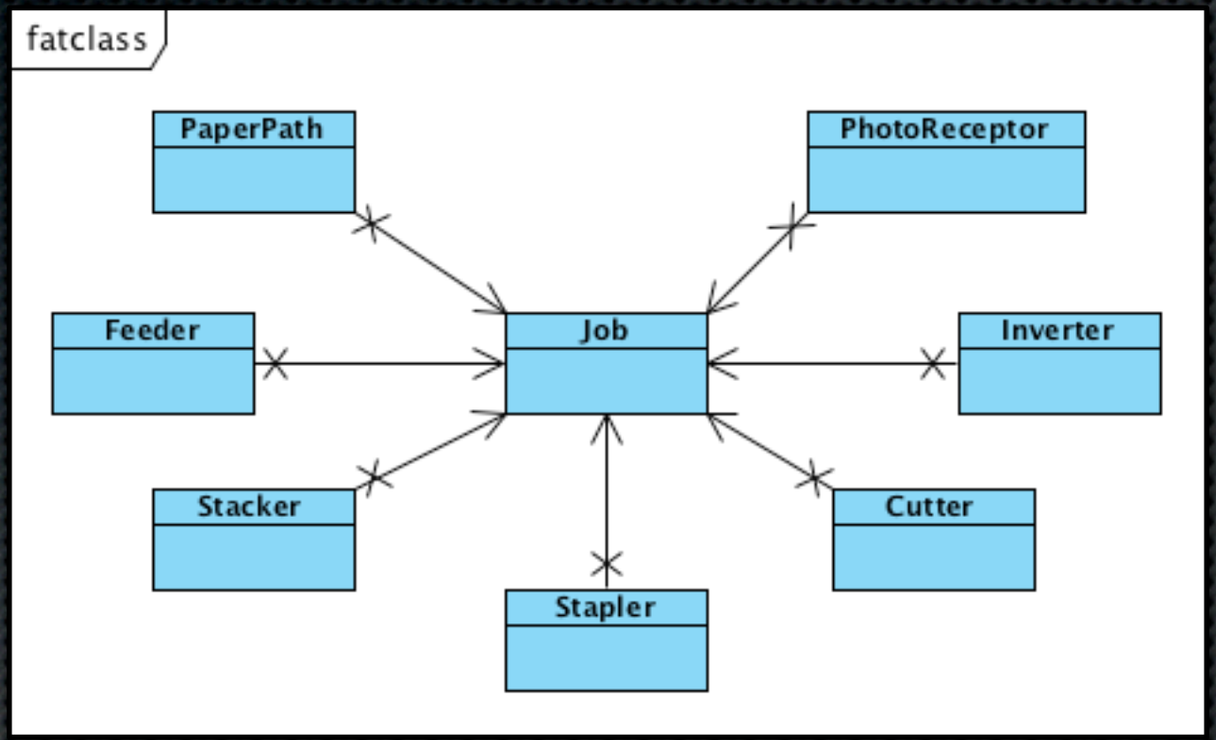
- 문제점
  - Switch가 Light에 의존적임.
  - Switch는 Light 뿐 아니라 Fan, Motor 등도 turnOn할 수 있다.
  - Switch는 Light에 대해서 알면 안된다.

Switch 예제



- 해결책
  - Switch는 Light에 대한 의존성을 갖지 않는다.
  - Switch와 Switchable이 같은 패키지/배포 단위
  - Interface(Switchable)는
    - 클라이언트(Switch)에 속한다
    - 구현체(Light)와는 관련이 없다
    - 그러므로 인터페이스의 이름은 클라이언트와 연관된 것이어야

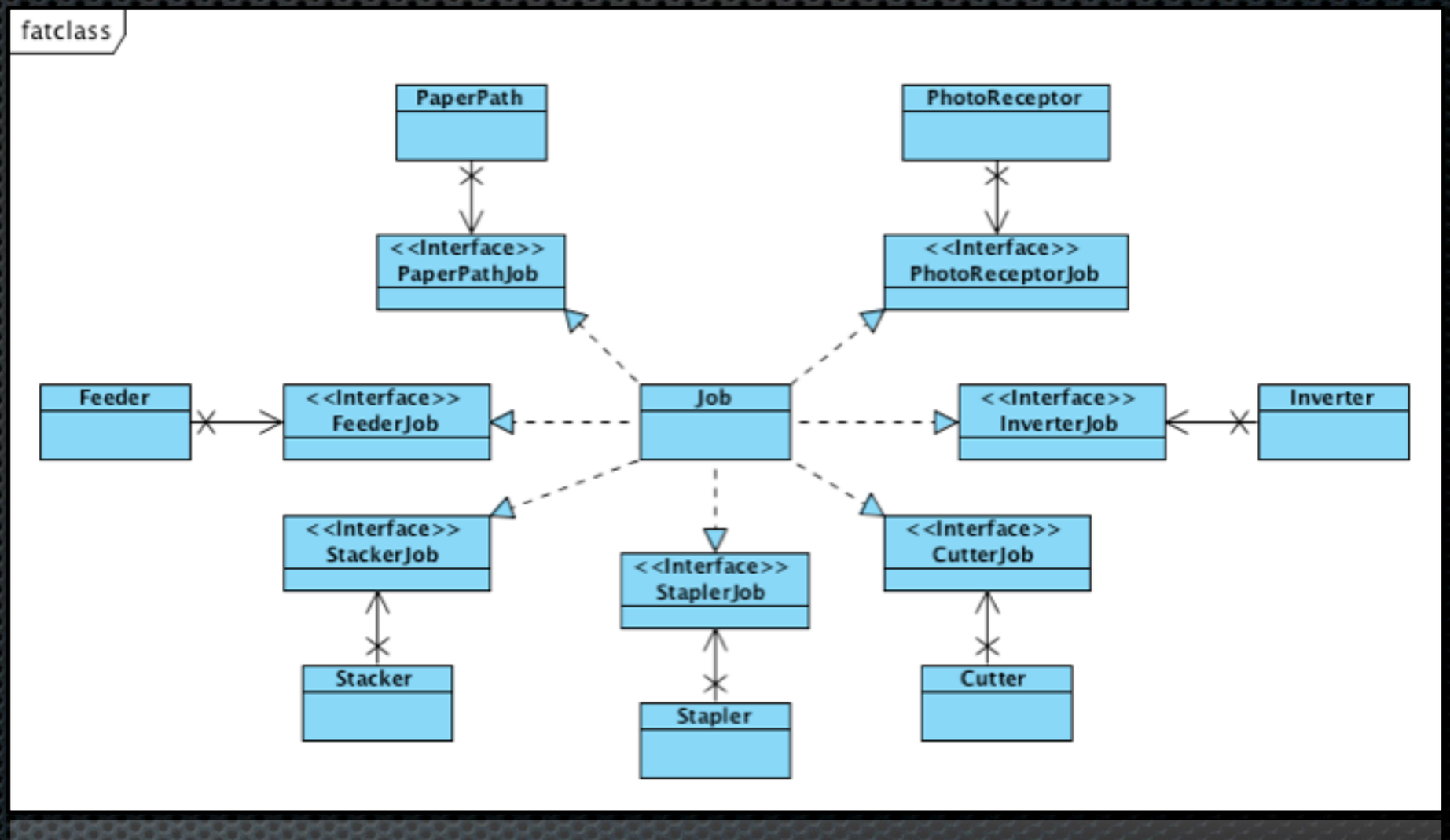
Fat class 예제



- Job 클래스가 많은 일을 하고 있어서 많은 Fan In
- 각 서브 시스템은 서로 다른 이유로 Job에 의존
- 문제점
  - rebuild에 많은 시간 소요
  - 독립적인 배포/개발 불가

Fat class 예제

- 해결책
  - One Interface for a sub system
  - 어떤 인터페이스에 변경이 발생하면
    - Job 클래스와 해당 인터페이스를 사용하는 서브 시스템만 rebuild



Fat class를 만다면

- Interface를 생성하여 Fat Class를 클라언트로부터 isolate시켜야함
- Fat Class에서 다수의 Interface를 구현
- Interface는 구현체보다는 클라이언트와 논리적으로 결합되므로 클라이언트가 호출하는 메소드만 Interface에 정의되었다는 것을 확신할 수 있음.

–ISP 준수

Fat class를 만다면

-