의존과 DI, DIP

Agenda

- Dependency
- Dependency Injection
- Dependency Inversion Principle

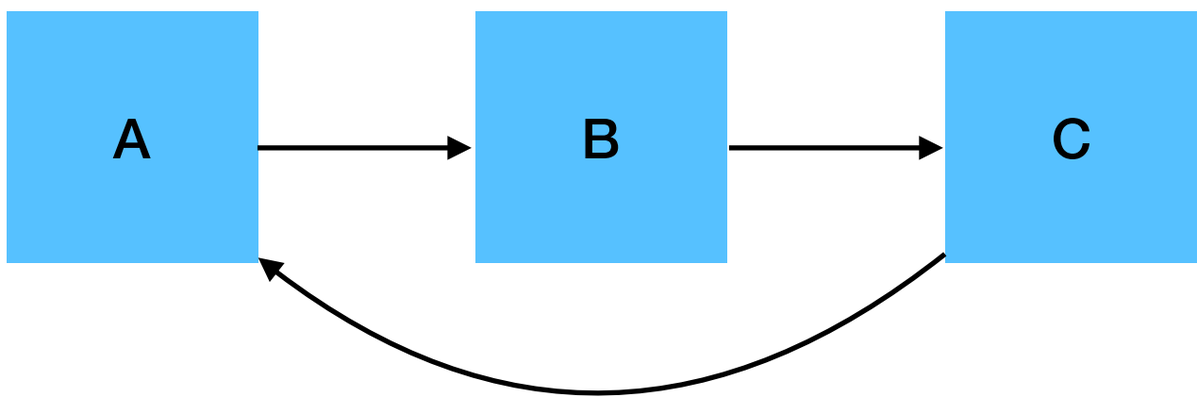
Dependency

- 의존(Dependency)이란?
  - 기능 구현을 위해 다른 구성 요소를 사용하는 것
  - ex) 객체 생성, 메소드 호출, 데이터 사용

```
public class Foo{
    private Bar bar;
    public Foo() {
        bar = new Bar();
    }
}
```

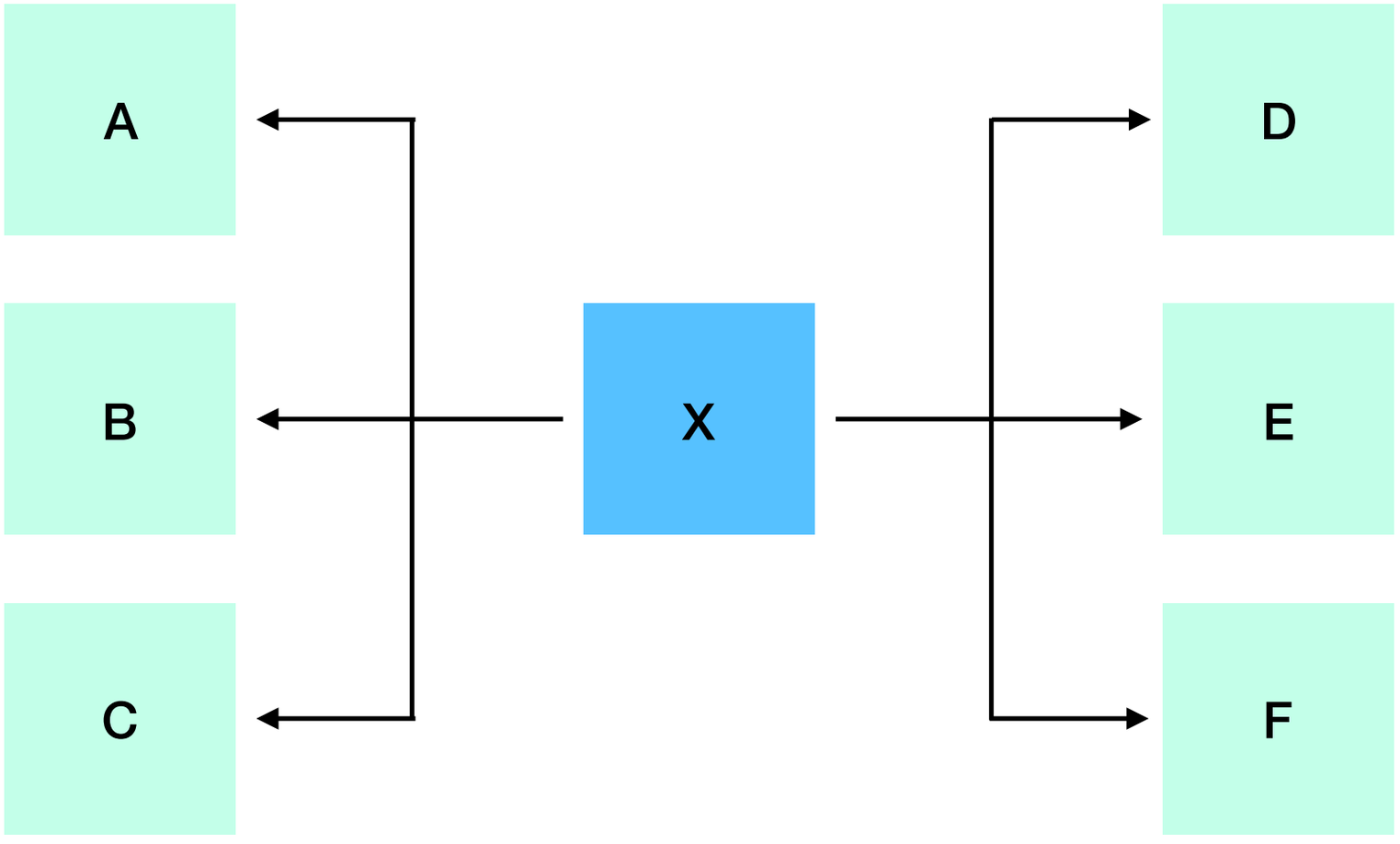
Dependency

- 의존은 변경이 전파될 가능성을 의미함
- 의존하는 대상이 바뀌면 나도 바뀔 가능성이 큼
- 순환 의존



의존하는 대상이 많다면?

의존하는 대상은 적을 수록 좋다. 변경될 가능성이 적기 때문



의존하는 대상이 많다면?

1. 기능이 많을 경우

```
public class UserService {
    public void regist(ReqReq reqReq) { … }
    public void changePw(ChangeReq chgReq) { … }
    public void blockUser(String id, String reason) { … }
}
```

각 기능마다 의존하는 대상이 다를 수 있음

한 기능 변경이 다른 기능에 영향을 줄 수 있음

한 클래스에서 제공하는 기능이 많으면 기능 별로 분리를 고려해봐야한다

의존하는 대상이 많다면?

```
public class UserService {
    public void regist(ReqReq reqReq) { … }
    public void changePw(ChangeReq chgReq) { … }
    public void blockUser(String id, String reason) { … }
}
```

↓

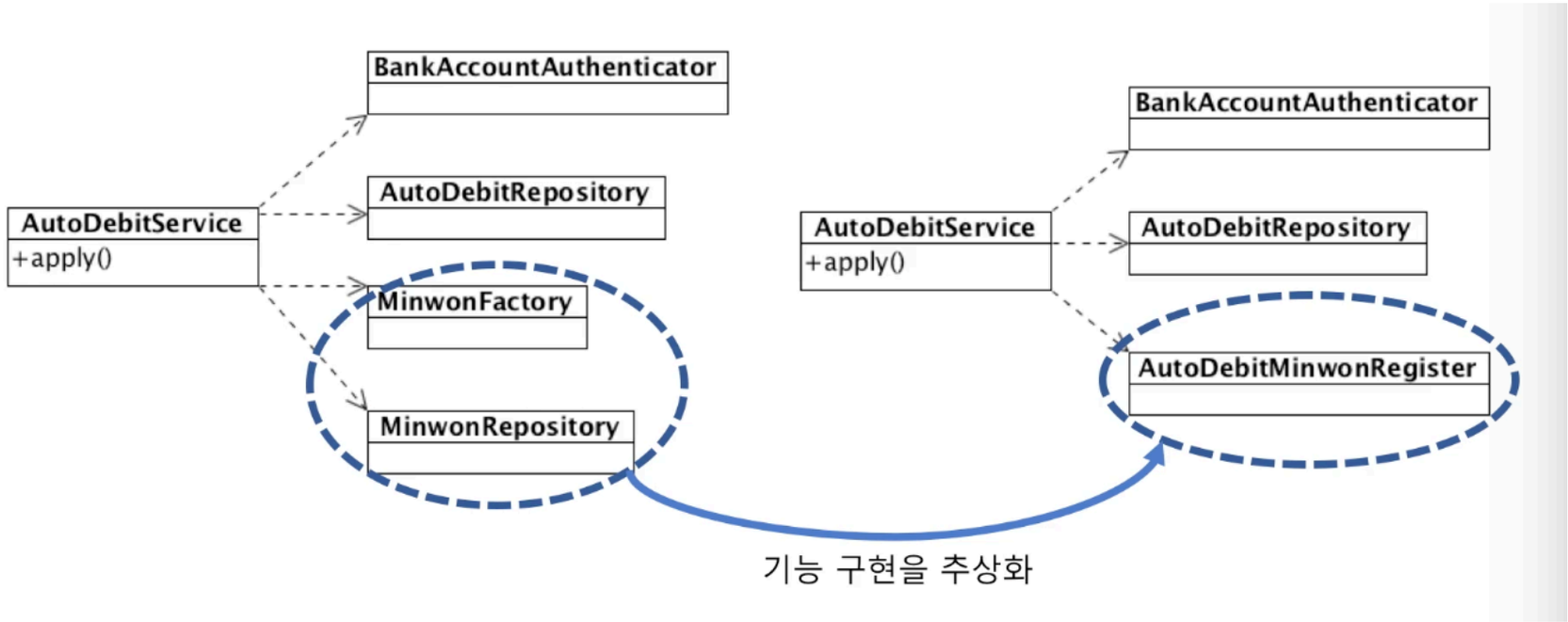
```
public class UserRegistService {
    public void regist(ReqReq reqReq) { }
}

public class ChagePwService {
    public void changePw(ChangeReq chgReq){ }
}

public class UserBlockService {
    public void blockUser(String id, String reason) { }
}
```

의존하는 대상이 많다면?

2. 묶어보기



몇 가지 의존 대상을 단일 기능으로 묶어서 생각해보면 의존 대상을 줄일 수 있다

의존 대상 객체를 직접 생성하면?

의존하는 객체를 직접 생성할 경우 클래스가 바뀌면 의존하는 코드도 변경된다

의존 대상 객체를 직접 생성하지 않는 방법은?

Dependency Injection

Dependency Injection

내부가 아니라 외부에서 의존 대상을 넣어주는 것

의존하는 대상을 직접 생성하지 않고, 생성자, 메소드를 이용해서 전달받는 방식

Dependency Injection

```java
public class ScheduleService {
    private UserRepository repository;
    private Calculator cal;

    public ScheduleService(UserRepository repository) {
        this.repository = repository;
    }
    public void setCalculator(Calculator cal) {
        this.cal = cal;
    }
}
```

```java
// 초기화 코드

UserRepository userRepo = new DbUserRespository();
Calculator cal = new Calculator();

ScheduleService schSvc = new ScheduleService(userRepo);
schSvc.setCalculator(cal);
```

프로그램을 시작하는 메인 메소드에서 의존 객체를 생성하고 주입을 할 수 있지만
보통 조립기를 이용하여 객체를 생성하고 의존 주입을 처리함

(ex. Spring Framwork)

Dependency Inversion Principle

이 원칙을 따르면, **상위 계층(정책 결정)**이 **하위 계층(세부 사항)**에 의존하는 전통적인 의존 관계를 반전(역전)시킴으로써 **상위 계층이 하위 계층의 구현으로부터 독립**되게 할 수 있다. 이 원칙은 다음과 같은 내용을 담고 있다.

첫째, 상위 모듈은 하위 모듈에 의존해서는 안된다. 상위 모듈과 하위 모듈 모두 추상화에 의존해야 한다.

둘째, 추상화는 세부 사항에 의존해서는 안된다. 세부사항이 추상화에 의존해야 한다.

- 위키백과 -

Dependency Inversion Principle

- 고수준 모듈
    - 어떤 의미있는 단일 기능을 제공하는 모듈
    - 고수준 모듈은 상위 수준의 정책을 구현
- 저수준 모듈
    - 고수준 모듈의 기능을 구현하기 위해 필요한 하위 기능의 실제 구현

고수준 모듈, 저수준 모듈 예

수정한 도면 이미지를 NAS 에 저장하고, 측정 정보를 DB 테이블에 저장하고,
수정 의뢰 정보를 DB 에 저장하는 기능

- 고수준
  - 도면 이미지를 저장
  - 측정 정보를 저장
  - 도면 수정 의뢰를 함
- 저수준
  - NAS 에 이미지 저장
  - MEAS_INFO 테이블에 저장
  - BP_MOD_REQ 테이블에 저장

고수준 모듈은 **기능의 정책을 제공**, 저수준 모듈을 **하위 기능의 구현을 제공**

고수준이 저수준에 직접 의존하면?

```
public class MeasureService {
    public void measure(MesureReq req) {
        File file = req.getFile();
        nasStorage.save(file);
    }

    jdbcTemplate.update("insert into MEAS_INFO ...");
    jdbc.Template.update("insert into BP_MOD_REQ ...");
}
```

고수준이 저수준에 직접 의존하면?

요구 사항이 변경 되었다!!

```
public class MeasureService {
    public void measure(MesureReq req) {
        File file = req.getFile();
        nasStorage.save(file);
    }

    jdbcTemplate.update("insert into MEAS_INFO ...");
    jdbc.Template.update("insert into BP_MOD_REQ ...");
}
```

```
public class MeasureService {
    public void measure(MesureReq req) {
        File file = req.getFile();
        s3storage.upload(file);
    }

    jdbcTemplate.update("insert into MEAS_INFO ...");
    rabitmq.convertAndSend(…);
}
```

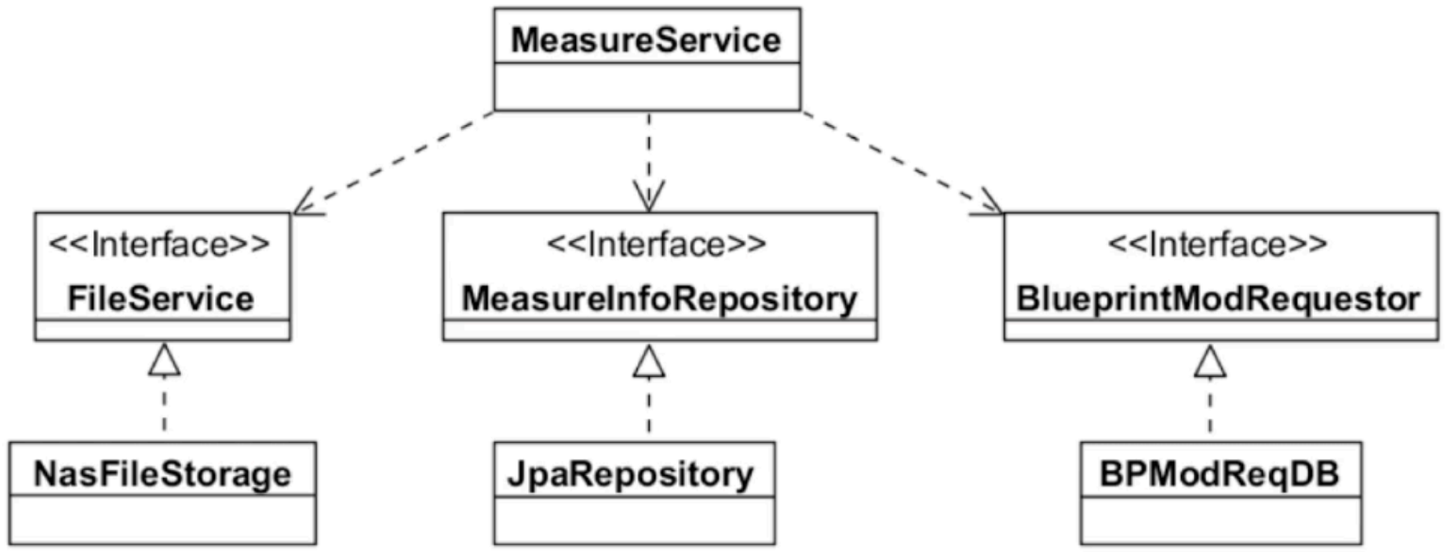
Dependency Inversion Principle

이렇게 고수준 모듈이 저수준 모듈에 의존하면서 발생하는 문제를 방지하기 위한 것이 DIP !

고수준 모듈이 저수준 모듈에 의존하면 안되며

저수준 모듈이 고수준 모듈에 정의한 추상타입에 의존해야한다

Dependency Inversion Principle

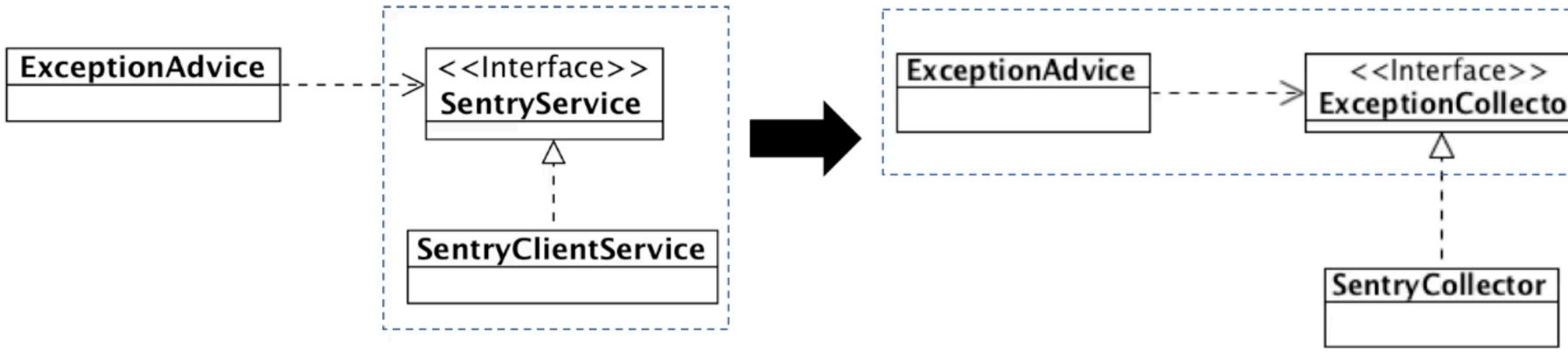


고수준 모듈이 저수준 모듈에 의존하는 것이 아니라 저수준 모듈이 고수준 모듈에 의존하도록

의존의 방향을 역전시키라는 원칙이 DIP !!

Dependency Inversion Principle

DIP 에서 하위 기능에 대한 추상타입을 도출할 때는 고수준 입장에서 추상화해야한다



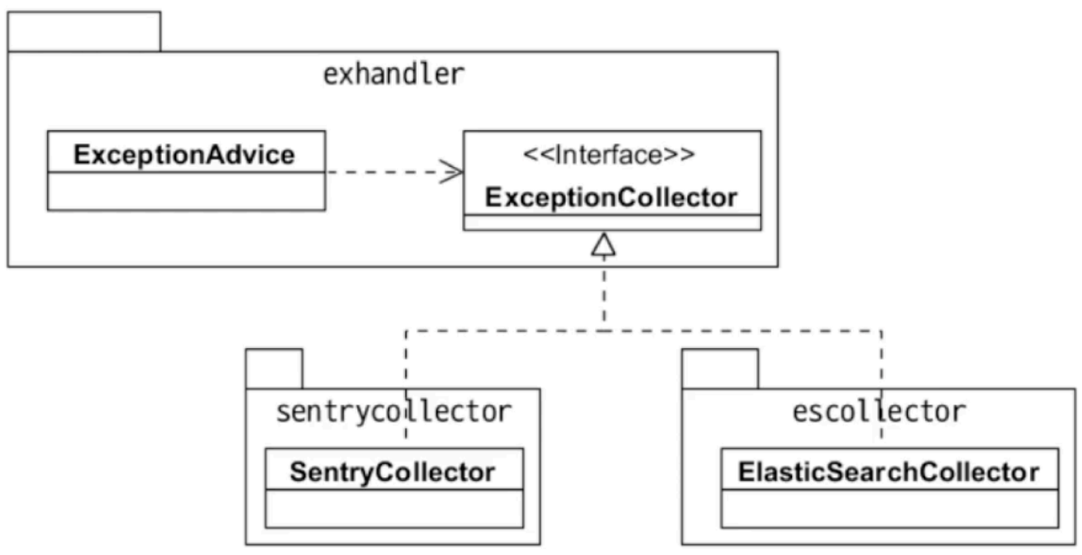
저수준 입장에서 추상화를 하면 Sentry 가 중심이 된다

Dependency Inversion Principle

DIP 는 유연함을 높여준다

고수준 모듈의 변경을 최소화하면서 저수준 모듈의 변경 유연함을 높임

만약 exception 을 Sentry 가 아니라 ElasticSearch 로 보내고 싶다고 하자



고수준 모듈이 바뀌지 않으면서 저수준 모듈의 구현을 바꿀 수 있는 유연함이 생김

연습

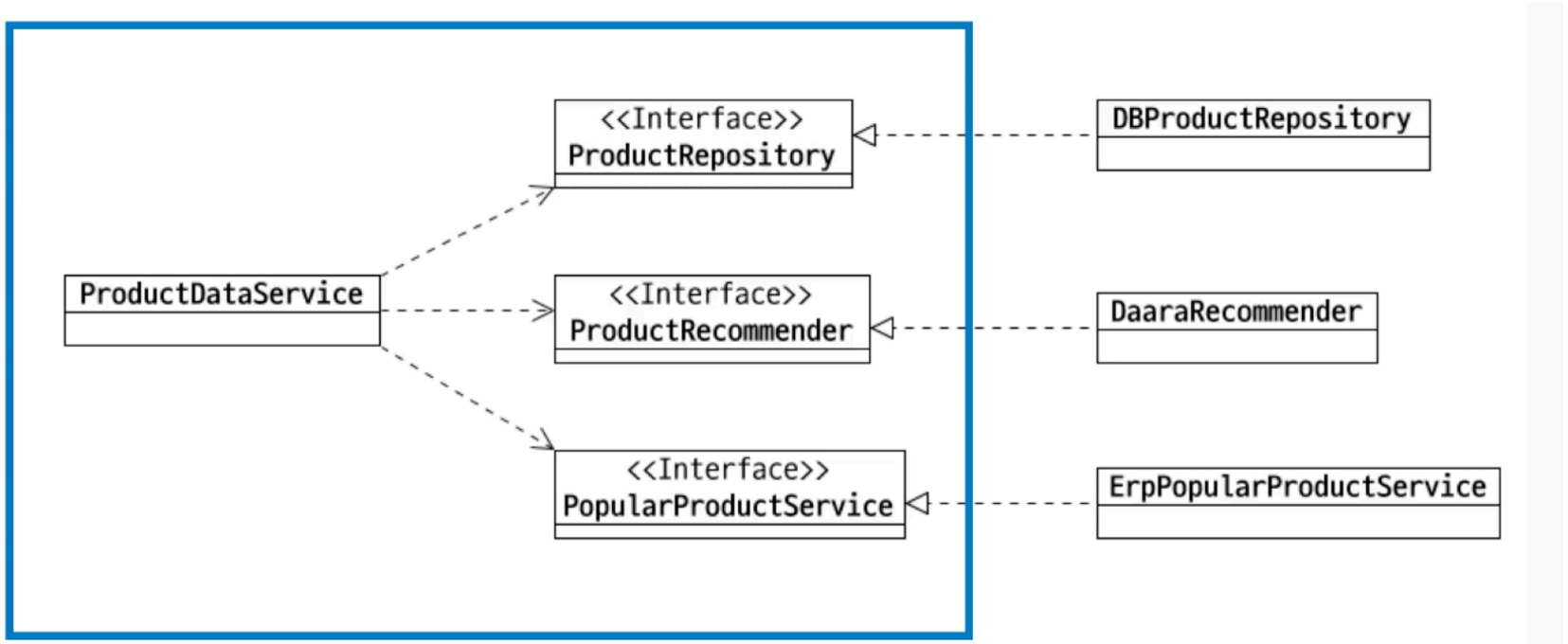
- 상품 상세 정보와 추천 상품 목록 제공 기능
  - 상품 번호를 이용해서 상품 DB에서 상세 정보를 구함
  - Foo API 를 이용해서 추천 상품 5개를 구함
  - 추천 상품이 5개 미만이면 같은 분류에 속한 상품 중 최근 한달 판매가 많은 인기 상품을 ERP 에서 구해 5개를 채움

연습

- 고수준
    - 상품 번호로 상품 상세 정보 구함
    - 추천 상품 5개 구함
        - 인기 상품 구함
- 저수준
    - DB 에서 상세 정보 구함
    - Foo API 에서 상품 5개 구함
    - 같은 분류에 속한 상품에서 최근 한달 판매가 많은 상품 ERP 에서 구함

연습

이렇게 분리한 구조를 설계에 반영해보자. 다음과 같이 설계를 할 수 있다



연습

처음부터 DIP 를 잘 따르는 구조로 설계하는 것은 힘들다.

DIP 만 잘해도 좋은 설계가 될 가능성이 매우 높다.

고수준과 저수준 관점에서 기능을 분리하고 설계하는 연습을 많이 해보자.

Reference

- 인프런 객체지향 프로그래밍 입문 - 의존과 DI, DIP
  - https://www.inflearn.com/course/%EA%B0%9D%EC%B2%B4-%EC%A7%80%ED%96%A5-%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%B0%8D-%EC%9E%85%EB%AC%B8/dashboard
- Dependency Injection 이란?
  - https://medium.com/@jang.wangsu/di-dependency-injection-%EC%9D%B4%EB%9E%80-1b12fdefec4f
- 세 가지 DI 컨테이너로 향하는 저녁 산책
  - http://www.nextree.co.kr/p11247/
- Inversion Of Control(IOC) Container 란?
  - https://medium.com/@jang.wangsu/di-inversion-of-control-container-%EB%9E%80-12ecd70ac7ea
- Dependency Injection
  - https://www.youtube.com/watch?v=IKD2-MAkXyQ