

전략파트너개발그룹

# 쉽게 익히는 클리코드

이번주 컨텐츠는?  
~냄새. 뒤엀킨 변경

\*리팩.  
단계  
조개기

\*리팩.  
함수  
옮기기

\*리팩.  
클래스  
추출하기



## ~냄새. 뒤엎킨 변경

### Divergent Chagne

- 소프트웨어는 변경에 **유연하게(soft)** 대처할 수 있어야 한다.
- 어떤 한 모듈이(함수 또는 클래스) 여러가지 이유로 다양하게 변경되어야 하는 상황
  - 예) 새로운 결제 방식을 도입하거나, DB를 변경할 때 동일한 클래스에 여러 메소드를 수정해야 하는 경우
- 서로 **다른 문제는 서로 다른 모듈에서 해결**해야 한다.
  - 모듈의 **책임이 분리**되어 있을수록 해당 문맥을 더 잘 이해할 수 있으며 다른 문제는 신경쓰지 않아도 된다.



## ~냄새. 뒤엎킨 변경

### 사용할 수 있는 리팩토링 기술

- "단계 쪼개기(Split Phase)"를 사용해 서로 다른 문맥의 코드를 분리할 수 있다.
- "함수 옮기기(Move Function)"를 사용해 적절한 모듈로 함수를 옮길 수 있다.
- 여러가지 일이 하나의 함수에 모여 있다면 "함수 추출하기(Extract Function)"를 사용할 수 있다.
- 모듈이 클래스 단위라면 "클래스 추출하기(Extract Class)"를 사용해 별도의 클래스로 분리할 수 있다.

■ 주황색 리팩토링 방법은 이번호에서 확인해주세요

## \*리팩토링. 단계 쪼개기

### Split Phase

- 1 서로 다른 일을 하는 코드를 각기 다른 모듈로 분리한다.
  - SRP(Single Responsibility Principle) 만족시키며, 어떤 것이 변경이 필요할 때 관련 있는 것만 신경쓸 수 있다.
- 2 여러 일을 하는 함수의 처리 과정을 각기 다른 단계로 구분할 수 있다.
  - 예) 전처리 > 주요 작업 > 후처리
  - 예) 컴파일러 : 텍스트 읽어오기 > 실행 가능한 형태로 변경
- 3 서로 다른 데이터를 사용한다면 단계를 나누는데 있어 중요한 단서가 될 수 있다.
- 4 중간 데이터(Intermediate Data)를 만들어 단계를 구분하고 매개변수를 줄이는데 활용할 수 있다.

## \*리팩토링. 단계 쪼개기

### Split Phase

```
String[] orderData = orderString.split(/\s+/);  
int productPrice = priceList[orderData[0].split("-")[1]];  
int orderPrice = parseInt(orderData[1]) * productPrice;
```



```
Order orderRecord = parseOrder(order);  
int orderPrice = price(orderRecord, priceList);  
  
Order parseOrder(aString) {  
    String[] values = aString.split(/\s+/);  
    return new Order(values[0].split("-")[1].parseInt(values[1]);  
}  
  
int price(Order order, int[] priceList) {  
    return order.quantity * priceList[order.productID];  
}
```

## \*리팩토링. 함수 옮기기

### Move Function

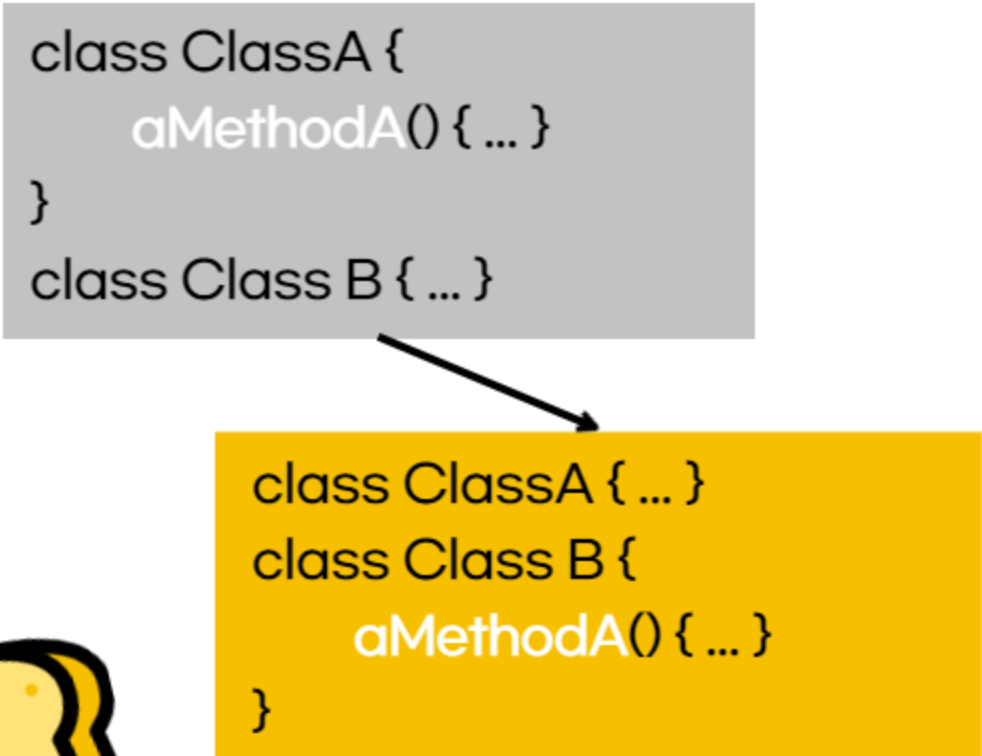
관련있는 함수나 필드가 항상 고정적인 것은 아니기 때문에 때에 따라 옮겨야 할 필요가 있다.

#### 함수를 옮겨야 하는 경우

- 해당 함수가 다른 문맥(클래스)에 있는 데이터(필드)를 더 많이 참조하는 경우
- 해당 함수를 다른 클라이언트(클래스)에서도 필요로 하는 경우

함수를 옮겨갈 새로운 문맥(클래스)이 필요한 경우에는 "**여러 함수를 클래스로 묶기 (Combine Functions into Class)**" 또는 "**클래스 추출하기(Extract Class)**"를 사용한다.

```
class ClassA {  
    aMethodA() { ... }  
}  
class Class B { ... }
```



```
class ClassA { ... }  
class Class B {  
    aMethodA() { ... }  
}
```



## \*리팩토링. 클래스 추출하기

### Extract Class

클래스가 **다루는 책임(Responsibility)**이 많아  
질수록 클래스가 점차 커진다.

**하위 클래스**를 만들어 책임을 분산시킬 수도 있  
다.

#### 클래스를 쪼개는 기준

- 데이터나 메소드 중 일부가 매우 밀접한  
관련이 있는 경우
- 일부 데이터가 대부분 같이 바뀌는 경우
- 데이터 또는 메소드 중 일부를 삭제한다  
면 어떻게 될 것인가?
  - 문제가 없는 경우 추출



## \*리팩토링. 클래스 추출하기

### Extract Class



```
class Person {  
    getName() { ... }  
    getOfficeAreaCode() { ... }  
    getOfficeNumber() { ... }  
}
```



```
class Person {  
    private TelephoneNumber teleNum;  
    getName() { ... }  
    getOfficeAreaCode() { return teleNum.getAreaCode(); }  
    getOfficeNumber() { return teleNum.getNumber(); }  
}  
  
class TelephoneNumber {  
    getAreaCode() { ... }  
    getNumber() { ... }  
}
```





클린코드 꿀팁을 놓치고 있다면?

클린코드 **팁**  
구독해요!

전략파트너개발그룹 - 양지용

