Refactoring 01

책의 1장은 간단한 예제를 놓고 천천히 리팩토링하는 내용을 다룹니다.
개인적으로 Test 와 OOP 에 대해 잘 설명해 놓은듯해서 책 읽는 시간(`주말`)이 아깝지 않았습니다.

1장에 나오는 리팩토링 기법들은 아래와 같습니다.

```
1. extract method
2. rename variable
3. move method
4. replace temp with query
5. replace conditional with Polymorphism
```

이 내용 모두를 정리하기는 애매할 듯하여 개인적으로 인상깊었던 3,5 을 정리하도록 하겠습니다.

> 코드 내용이 대다수여서 모두 정리하기는 의미가 없을듯 합니다. :D

Base

예제는 간단한 `비디오 대여 시스템` 입니다.
`Customer` , `Rental` , `Video` 3개의 객체가 존재하며 각 객체들은 서로에게 상호작용 합니다

> 리팩토링에 대한 예제이기에, 통으로 묶인 코드를 점진적으로 분리하고, 고쳐나갑니다.

extract and move the method

메서드 추출이 필요한 이유는 아래와같은 예제로 설명 가능합니다.

```
class Customer {
    String name
    Vector rentals = new Vector();

    Customer(String name) {
        this.name = name
    }

    void addRental(Rental rental) {
        rentals.add(rental);
    }

    String statement() {
        double totalAmount = 0
        int frequentRenterPoints = 0
        Enumeration rentals = rentals.elements()
        String result = getName() + "고객님의 대여 기록 \n"
```

```
        while (rentals.hasMoreElements()) {
            Rental each = (Rental) rentals.nextElement()

            // 비디오 종류별 대여료 계산
            double thisAmount = each.getCharge(each)
            // 적립 포인트 1 증가
            frequentRenterPoints++

            // 최신물을 이틀 이상 대여하면 보너스 포인트 지급
            if ((each.getMovie().getPriceCode() == Movie.NEW_RELEASE) && each.getDa
                frequentRenterPoints++
            }

            // 이번에 대여하는 비디오 정보와 대여료 출력
            result += "/t" + each.getMovie().getTitle() + "\t" +
                    String.valueOf(thisAmount) + "\n"

            // 현재까지 누적된 총 대여료
            totalAmount += thisAmount
        }

        result += "적립 포인트: " + String.valueOf(frequentRenterPoints)
    }
}
```

Clean code 에서는 함수는 짧을 수록 이해 및 유지보수하기 쉽다고 말합니다.

> 모니터에 함수가 담겨야한다 -> 5줄(?) 이하 여야한다.

5줄은 극단적으로 들릴 수 있겠지만, 그만큼 함수는 작을 수록 좋다는 의미입니다.
그런 뜻에서 위 함수는 너무나 많은 책임을 들고 있고, 때문에 유지보수또한 어려워집니다.

때문에 책에서는 기능(?) 단위로 함수를 분리합니다.

1. point 함수 분리

statement 함수에는 `포인트` / `가격` / `출력` 의 기능을 수행합니다.
간단하게 포인트를 구하는 로직을 함수로 분리하죠.

```
frequentRenterPoints++

// 최신물을 이틀 이상 대여하면 보너스 포인트 지급
if ((each.getMovie().getPriceCode() == Movie.NEW_RELEASE) && each.getDaysRented() >
    frequentRenterPoints++
}
```

`frequentRenterPoints` 는 추후 사용자 내역을 출력할 때, 사용하는 변수로 분리된다고하도 큰 문제가 없을듯합니다 :D

```
String statement(){
  ...

    result += "적립 포인트: " + String.valueOf(getPoints(each))
}

int getPoints(Rental each) {
        // 최신물을 이틀 이상 대여하면 보너스 포인트 지급
       if ((each.getMovie().getPriceCode() == Movie.NEW_RELEASE) && each.getDaysRen
           return 2
       }
       return 1;
 }
```

위 처럼 사용할 경우, statement에 선언되었던 임시변수(`frequentRenterPoints` -phase4)도 삭제되며 함수의 길이가 확실히 줄어들게 됩니다.

> while 문이 2배로 늘어난다는 이슈가 있을 수 있지만, 이 부분은 성능분석을 할때 신경을 쓰라고 말합니다.

2. point 함수 이동

point 부분을 추출하고 보니, 묘한 부분이 보입니다.

```
if ((each.getMovie().getPriceCode() == Movie.NEW_RELEASE) && each.getDaysRented() >
    return 2
}
```

위 함수는 `Customer` 내부에 있는 코드로 캡슐화가 덜 되어있는 코드입니다.

`Don't Ask, Tell` 법칙으로 알 수 있듯, 객체는 요청하는게 아니라 단지 메시지를 통해서만 동작해야합니다.
그런 의미에서 위 코드는 Movie 객체가 할일을 하고 있습니다.

위 코드를 Rental 로 옮기고 Movie에 별도의 메시지를 만들어 통신해야합니다.

> Rental

```
int getPoints() {
    return movie.getPoints(this.daysRented);
}
```

> Movile

```
int getPoints(int dayRental) {
    return price.getPoints(dayRental)
}
```

위 처럼 사용 코드를 수정하 추후 point를 구하는 로직이 바뀌어도 Customer 나 rental에는 전혀 영향을 주지 않는 코드가 됩니다.

마치며

책의 1 장은 이후 배울 내용들에 대한 요약본으로 느껴집니다.
간단한 예제로 코드를 수정하는 법을 알려주네요 :D
시간이 되신다면 직접 쳐보는 걸 추천드립니다 ~!