과정 진행 - Refactoring

과정 시간표 - 리팩토링

	20일 (월)	21일 (화)	22일 (수)	23일 (목)
09:00		9::00 ~ 11:30 실습 #2	09:00 ~ 11:00 - Refactoring Technique	9:00 ~ 12:00 리팩토링 평가
10:00				
11:00		11:30 ~ 12:00 실습 #2 리뷰	11:00 ~ 12:00 실습#4	
12:00	점심 식사	점심 식사	점심 식사	점심 식사
13:00	13:00 ~ 16:30 - 과정 소개 - 리팩토링 개요 - Code Smell - Long Parameter List,.. - Divergent Change	13:00 ~ 14:00 - Loops - Messaging Chains	13:00 ~ 14:00 실습#4	
14:00		14:00 ~ 17:00 실습 #3	14:00~15:00 실습 #4 리뷰 실습 #5 안내	
15:00			15:00~17:30 실습 #5	
16:00				
17:00	16:30 ~ 17:30 실습 #1 17:30 일과 마무리	17:00 실습 #3 리뷰 일과 마무리	17:30 일과 마무리	

* 과정 평가 : 실습(#1~#5) 80점 + 구현평가 120점 = 200 점

코드리뷰에 대한 설문결과

인식 부족

"자료를 만들어야 해서 형식적으로 코드 리뷰를 진행할 때가 많아요"
"업무 우선순위에서 가장 밀려서 코드리뷰가 진행되지 못하고 있습니다"

역량 부족

"리뷰할때 어디에 중점을 둬야 할지 모르겠어요..."
"아는만큼 보인다는데 리뷰할때 마다 한계가 느껴집니다."

책임감 부족

"상급자에게 코드리뷰를 강제하고 코드품질의 책임을 떠넘기는것 같아요"
"셀프(Self) 리뷰도 안하고 바로 리뷰 요청을 하는 경우가 많아요"

소통 부족

"코드리뷰를 받으면 왠지 평가받는것 같아서 부담스러워요"
"누가 제 코드를 보고 뭐라고 하는게 기분 나빠요"

코드리뷰의 목적

- **Better code quality** (ex.- readability, maintainability, ..)
- Finding defects
- Learning/Knowledge transfer
- Increase sense of mutual responsibility
- Finding better solutions
- Complying to QA guidelines

코드의 유형

- 코드의 유형 (=리뷰 대상의 유형)
  - 개발중인 코드 (자신, 팀, 다른팀)
  - 재사용 되는 코드
    - 과거에 개발한 코드 (자신, 팀, 다른팀)
    - **오픈**소스 코드

- 코드의 독자
  - 현재의 본인, 팀원, 다른 팀원
  - 미래의 본인, 팀원, 다른 팀원
    - 다른 배경에서 재사용될 수도 있음 *- ex. 아리안4 일부 -> 아리안5*

코드리뷰와 리팩토링

"**Refactoring** is

a change made to the internal structure of software

to make it **easier to understand**

and **cheaper to modify**

without changing its observable behavior." - Martin Fowler -

- **코드리뷰의 가이드 역할**
  - Self-Inspection(혹은 TDD-Cycle의 Refactor step),
  - 잠재적인 문제에 대한 리뷰 comment
  - 심도 있는 review결과 도출 등

리팩토링 vs. 성능

- 고객이 수용하고 있는 S/W를 리팩토링 한다면, 성능이 나빠지지 않을까?
  - 대부분의 S/W는 "Soft Real-Time" System
    - (dead line 혹은 reasonable 한 time-limit이 존재)
  - **유지 보수하기 쉽게 만든 후 -> 최적화를 통하여 성능을 맞춤**
    - 대부분의 S/W는 전체 코드의 극히 **일부**에서 대부분 시간을 소비
    - 좋은 설계의 경우, 최적화하기가 더 쉽고, 성능을 더 세밀하게 분석할 수 있음
      - Profiler가 지정해 주는 코드의 범위가 좁아짐 -> 지정된 범위에 대해 집중적으로 최적화
      - *Cf) "성능개선을 위한 90%의 시간은 낭비였다" (통계)*
  - 현업의 이미 최적화된 코드라면?
    - 이미 운용되고 있는 최적화된 코드도, **유지보수가 힘들다면** 리팩토링이 필요
      - (성능이 중요시되는 부분도 유지보수의 대상)
    - 리팩토링 전후의 성능을 비교하여, 리팩토링의 수준/방법을 결정하는 것이 자연스러울 수 있음
      - (Profiling)

실습 #2 : Supermarket

- 실습 목적 : Refactoring as a series of small changes

- Repository : Supermarket
  - **개인실습** Repository 이름 : TDD 실습용 repo./branch 그대로 사용
  - TDD 과정에서 작성한 테스트 코드를 활용하여, 리팩토링

- 요구사항 요약
  - 새로운 OfferType을 추가하기 쉽도록, ShoppingCart의 handleOffers() 리팩토링을 시작
  - 전체적인 일련의 리팩토링의 전반부 단계인, 코드를 정리하는 리팩토링
    - 관련기법) Slide-Statements. Extract-Variable, Inline-Variable, ...

실습 #3 : GildedRose

- 실습 목적 : Replace Type-code with subclasses

- Repository : GildedRose
  - **개인실습** Repository 이름 : TDD 실습용 repo./branch 그대로 사용
  - TDD 과정에서 작성한 테스트 코드를 활용하여, 리팩토링

- 요구사항 본문

  여관 주인장 Allison 은 매출을 올리기 위해 Food & Beverage 를 판매하려고 합니다.
  Allison 은 새로운 F&B item 들의 quality 를 계산하는 로직을 추가하려고 하였습니다.

  Allison 이 **새로운 F&B item 을 쉽게 추가할 수 있도록** 리팩토링 해주세요.

- 중첩된 if 구조 개선, type-code를 다형성으로 변경

실습 결과

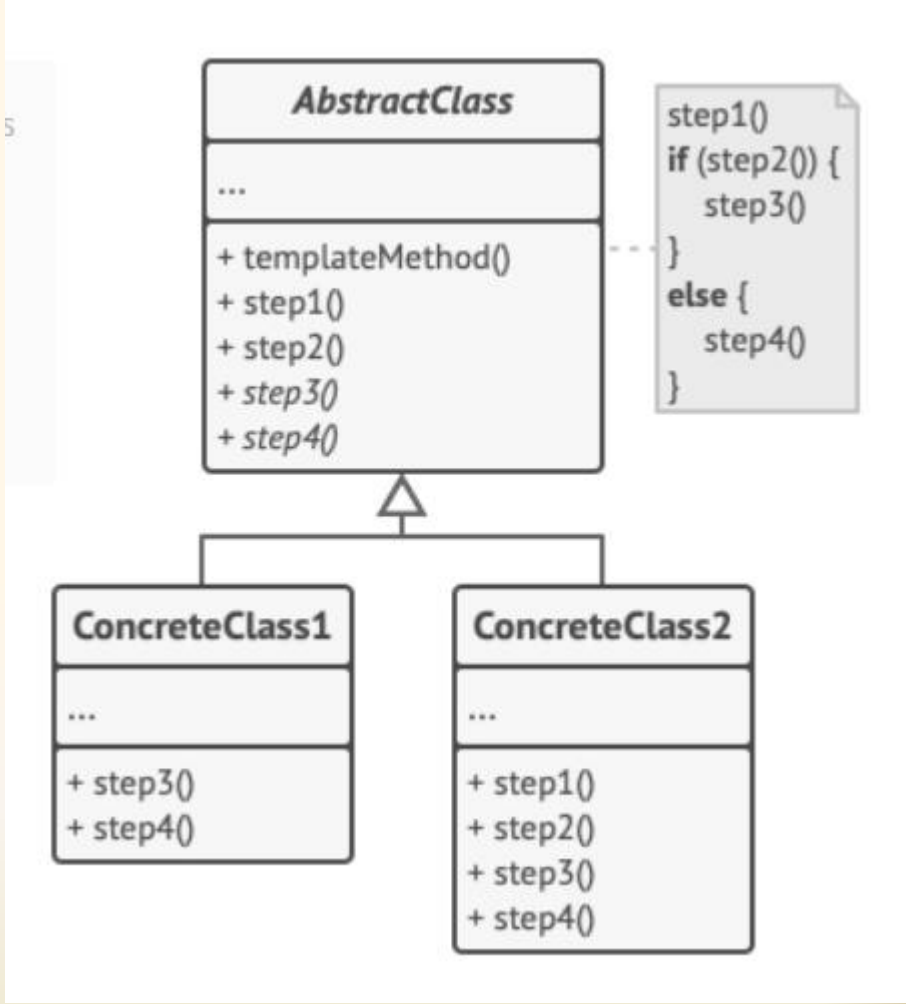
- GildedRose.updateQuality() 를 정리
  - if문의 조건이 false인 것들을 true기준으로 리팩토링
  - 중첩된 if문에서, item 이름을 비교하는 구문이 가장 상위에 위치하도록 조정
  - 최상위 if문을 각각의 item별로 분리

  - Quality를 50까지 증가시키는 구문을 std::min()을 사용하여 수정
  - items[i].quality = items[i].quality - items[i].quality 을 0으로 수정
- For loop아래 로직을 item으로 이동
- item을 상속받는 특수 item을 위한 class를 생성하여 updateQuality()를 각각 구현

Template Method 패턴 (1/2)

- **Intent**
  - A behavioral design pattern that defines the skeleton of an algorithm in the superclass but lets subclasses override specific steps of the algorithm without changing its structure.

- **활용**
  - 한 계층 구조내의
    서브클래스들에 구현된 비슷한 method 사이의
    코드 중복을 줄이거나 제거하는 데 도움이 됨



Template Method 패턴 (2/2)

- 절차
  - 두 함수를 일반화 (포함된 단위작업을 method로 추출하여, 원래의 함수를 Template화)
  - 일반화된 함수를 클래스로 독립 (Command Object, 슈퍼클래스)
    - (Replace Method With Method Object)
  - Template함수에서 호출하는 하위 함수를 서브클래스로 이동
  - 주어진 상속 구조 내의 서브클래스 사이에, 동일한(유사한) method가 있는지 확인
    - (공통 method vs. 재정의 필요한 method)
  - 공통 method에 대해 pull-up method (슈퍼클래스로 이동)
  - 서브클래스간 비공통 method에 대해 signature를 일치시킴
  - 비공통 method의 abstract method를 정의 (슈퍼클래스)

실습 #4 : Supermarket

- 실습 목적 : Template Method 패턴을 통한 리팩토링

- Repository : Supermarket
  - **개인실습** Repository 이름 : TDD 실습용 repo./branch **그대로** 사용
  - TDD 과정에서 작성한 테스트 코드를 활용하여, 리팩토링

- 요구사항 요약
  - Html printer 기능 추가됨(HtmlReceiptPrintr.txt 활용)
  - Html printer 기능 테스트 코드 수정 또는 추가 필요.
  - ReceiptPrinter클래스의 printReceipt()/printHtmlReceipt() 에 **Template Method패턴을 적용**하여 리팩토링
  - Algo.구조의 중복 제거

실습 #5 : Video Store

- 실습 제목 : Replace Type-Code with subclasses(delegate)
- & Template Method 패턴을 이용한 리팩토링

- 실습용 Remote Repository
  - **소스** Repository 이름 (**제공**)
    - https://github.ecodesamsung.com/Best-Reviewer-3-2/**RF_VIDEOSTORE.git**
  - **개인실습** Repository 이름
    - **Organization내 개인별 Repo. - RF_VIDEOSTORE_00** (개인번호 01~99)

실습 #5 설명 : 요구사항

- The sample program is very simple. It is a program to calculate and print a statement of a customer's charges at a video store. The program is told which movies a customer rented and for how long. It then calculates the charges, which depend on **how long** the movie is rented, and identifies **the type movie**. There are three kinds of movies: **regular, children's, and new releases**. In addition to **calculating charges,** the statement also computes **frequent renter points**, which vary depending on whether the film is a new release.

- **Further Requirements**
  - System owner wants to **add a type of movie** in order to vary rental charge and frequent renter points. It requires **"bestseller"** which is charged for 5 dollars per a day for first 2 days, after then 7.5 dollars per a day. bestseller type is quite expensive, it rewards 5 points per a day.

  - System owner **would like to change a type of movie** by certain policy.
    For instance, a film can be 'bestseller' from 'new release' if it become popular.
    The policy has not been determined so far, but we need to consider the possibility.

  - System Owner has a plan to extend his business on web.
    Not only printing customer statement on paper, but also **printing it as a html format.**
    Please refer below code.

실습 #5 설명 : 요구사항

- Video-Rental Program
  : 비디오의 종류(Regular, Children, New-Release)에 따라 고객의 요금, 적립금을 계산 및 출력

- 요구 사항
  - "bestseller" movie type 추가
    rental charge : 첫 이틀간은 하루에 5 달러, 그 이후는 7.5 달러
    frequent renter points : 하루에 5 포인트
  - movie type 변경 가능하게 수정
    - 예시) New-Release 가 Regular 로 변경
  - html format 프린트 기능 추가

실습 #4 설명 : To-do 1/2

- 테스트 코드 작성
  - Movie type 별 요금과 적립금 계산 확인 테스트 코드 작성
  - 프린트 기능 테스트 코드 추가

- 코드를 잘 이해하기 위한 리팩토링
  - Movie type별 계산 코드의 리팩토링
  - 새로운 movie type 추가 하기 쉽게 리팩토링
  - Movie type변경이 쉽게 리팩토링
    - Customer.statement() 리팩토링 – split loop, replace conditional with polymorphism or delegation
  - 프린트 로직 리팩토링

실습 #4 설명 : To–do 2/2

- "bestseller" movie type 추가
  - 추가될 movie type 에 대한 테스트 코드 추가
    - rental charge : 첫 이틀간은 하루에 5 달러, 그 이후는 하루에 7.5 달러
    - frequent renter points : 하루에 5 포인트
  - "bestseller" 타입 추가 구현
  - 리팩토링(타입 변경이 가능한 구조로 리팩토링)

- html format 프린트 기능 추가
  - html format 프린트 기능 테스트 코드 추가
  - html format 프린트 기능 추가 구현(Readme.md 참조)
    - htmlStatement()
  - 일반 출력과 html 출력의 중복 제거를 위한 리팩토링
    - Template Method pattern 기법 적용

실습 결과 (1/3) – Video Rental

- Best Seller 추가
  - Best Seller 를 단순 추가하기 위해서는 statement() 안에 priceCode 로 분기되어 있는 부분에 로직 추가하면 될 것.
  - 하지만 statement() 의 loop 에서는, total amount 와 적립금 총합, statement 를 출력하는 로직이 혼재되어 있어 리팩토링이 필요
  - 코드를 단순 추가하고 리팩토링할 경우, 리팩토링할 코드의 규모가 커지게 되어, 기능 추가하기 전 리팩토링 하기로 결정

  - Split loop 를 위해
    - While loop 를 for loop 로 수정하여 쉽게 iterator 돌리기 위해 Vector 를 List<> 로 수정

    - 지역변수 frequentRenterPointes, thisAmount, totalAmount 는 loop 안에서 값이 갱신이되고, statement 를 출력하는데 사용됨.
    - thisAmount 를 계산하는 구문과 값을 참조하는 구문 (totalAmount 계산 및 statement 출력) 을 나누기 위해, replace temp with query 를 적용하여 thisAmount 를 계산하는 구문을 함수로 추출하고, totalAmount 계산과 statement 출력시 해당 function 을 호출하도록 수정

    - Loop 를 3개로 나눔 (total amount 계산, frequentRenterPoints 계산, statement 구성)
    - total amount 계산, frequentRenterPoints 계산 을 function 으로 빼고, statement 구성시 호출하도록 수정.

실습 결과 (2/3) – Video Rental

- Best Seller 추가
  - Amount 계산시 switch 문을 제거하기 위해,
    - amountFor() 함수를 적절한 위치로 이동시키기 위해 Movie 로 이동
    - Replace conditional with polymorphism 을 위해 Replace Type code with subclasses 를 적용하는데, Movie 의 type 은 변경될 수 있으므로, delegate 방식을 사용하기로 결정
    - MovieType 을 객체를 생성하고, amount() 함수를 MovieType 으로 이동.
    - 각 type 별로 subclass 를 만들어 amount 를 계산하는 로직 구현
  - Type code 를 subclass 로 대체하기 위해 type code 를 참조하는 부분 제거
    - FrequencyRenterPoints 를 구하는 부분을 MovieType 으로 옮기고, NewRelease 에서 분기 처리
  - Type code 를 subclass 로 대체
  - Test code 와 함께 BestSeller 추가

실습 결과 (3/3) – Video Rental

- HtmlStatement 추가
  - htmlStatement() 를 현재 구조에 맞춰 테스트 코드와 함께 추가.
  - statement() 와 htmlStatement() 의 공통부 재활용을 위해 template method pattern 적용 결정
  - statement() 를 replace method with command(replace method with method object) 를 적용하여 객체로 추출
  - Statement.invoke() 에서 htmlStatement() 와 동일한 뼈대 부분은 그대로 두고, string 을 result 에 할당하는 가변적인 부분을 method 로 추출
  - Statement 를 상속하는 PaperStatement 객체를 생성하고, 추출한 method 들을 PaperStatement 로 옮기고, Statement 에서는 해당 method 들을 abstract method 로 선언.
  - htmlStatement() 도 replace method with command 적용하여 객체로 추출하고,
    Statement 를 상속받고 abstract method 에 가변부를 구현.

Code Smell을 찾는 Tip

- Object Calisthenics : 9 steps to better software design today, by Jeff Bay
  1. One level of indentation per method
  2. Don't use the ELSE keyword
  3. Wrap all primitives and Strings
  4. First class collections
  5. One dot per line
  6. Don't abbreviate
  7. Keep all entities small
  8. No classes with more than two instance variables
  9. No getters/setters/properties

- "Rather than construction,
  Software is more like Gardening - it is more **organic** then concrete"

- Pragmatic Programmer -

More Information

- Martin Fowler, Refactoring Improving the Design of Existing Code 2nd Ed., 2020
- Joshua Kerievsky, 패턴을 활용한 리팩토링, 2005
- Michael Feathers, 레거시 코드 활용전략, 2004
- Scott Ambler, Pramod Sadalage, 리팩토링 데이터베이스, 2006
- 오즈 모리하루, C와 C++게임코드로 알아보는 코딩의 기술, 2016

리팩토링 Kata를 구할 수 있는 곳..

- https://kata-log.rocks/refactoring, https://github.com/Pragmatists
  - Tennis Kata (1화일, 코드정리 짧음)
  - …

THANK YOU.