

Java Programming II Lab 7



교 과 명 : 자바프로그래밍 2

담당교수명 : 박경신 교수님

학 과 : 컴퓨터공학과

학 번 : 32185010

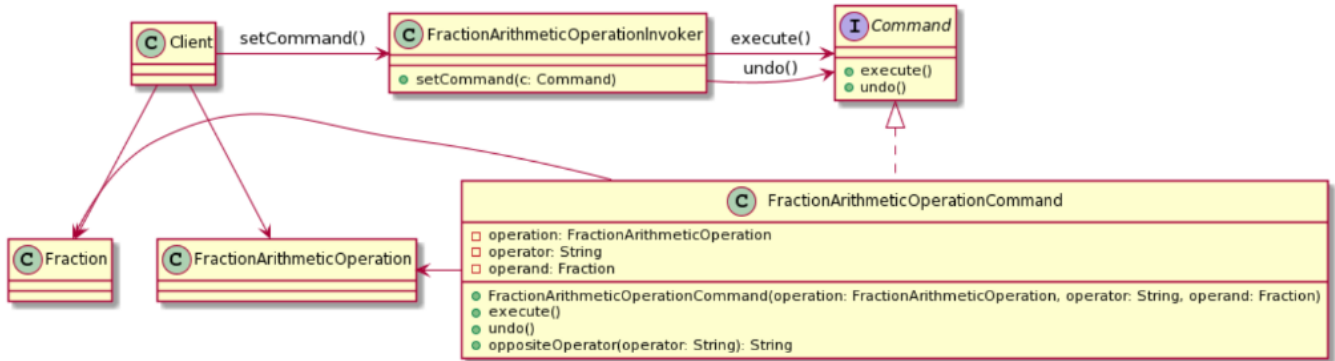
성 명 : 홍찬희

제 출 일 : 2021. 11. 03



Lab 7

커맨드패턴 (Command Pattern) 이란, 실행되어질 기능들을 캡슐화를 통해 여러기능을 사용할 수 있는 재사용성 높은 클래스를 설계하는 패턴이다. 기본적으로 Command, ConcreteCommand, Invoker, Receiver로 구성되는데 Invoker와 Receiver의 의존성을 제거함으로써 기능이 바뀌어도 클래스명을 수정하여 호출할 필요없게 해준다.



해당 프로젝트에서는 Command는 Command, ConcreteCommand는 FractionArithmeticOperationCommand, Invoker는 FractionArithmeticOperationInvoker, Receiver는 FractionArithmeticOperation으로 구성되어있다.

```
// receiver class
FractionArithmeticOperation operation = new FractionArithmeticOperation();
// invoker class
FractionArithmeticOperationInvoker invoker = new FractionArithmeticOperationInvoker();
```

Client 메인클래스의 동작을 살펴보면, FractionArithmeticOperation (Receiver) 생성을 통해 operation의 value를 0/1 즉, 0으로 설정한다. 그다음, FractionArithmeticOperationInvoker (Invoker) 생성을하여 호출을 명령을 할 클래스를 만든다.

```
invoker.setCommand(new FractionArithmeticOperationCommand(operation, operator: "+", a));
invoker.execute(); // 0/1 + 5/4 = 5/4
```

FractionArithmeticOperationCommand에 어떤 연산을 할지 매개변수로 넣어 객체를 생성해준뒤 setCommand 메소드를 통해 생성된 invoker에 실행할 기능을 넣어준다. setCommand된 invoker를 execute() 메소드를 동작시켜주면,

```
// 현재 Command의 동작을 실행시키는 메소드
public void execute(){
    stack1.push(command); // 현재 Command를 stack1 스택에 push 해준다.
    this.command.execute(); // 현재 Command의 execute 메소드를 실행시킨다.
}
```

현재 Command가 stack1에 저장되고, Command의 execute() 메소드를 실행시킨다.

```
if(this.operator.equals("+")){ // operator가 '+'면 분수의 덧셈연산
    this.operation.value = FractionArithmeticOperation.plus(operation.value, operand);
    System.out.print(" + ");
}
```

execute() 메소드 동작의 일부분을 보면, 현재 Command의 operator가 +,-,*,/ 인지 확인하고 연산자에 맞는 분수의 연산을 실행하고 그 값을 Fraction 형태로 반환한다. 반환받은 Fraction(결과값) 은 this.operation.value에 다시 저장한다.

```
invoker.undo(); // 1/3 * 13/7 = 13/21
invoker.undo(); // 13/21 / 4/7 = 13/12
```

undo의 동작은 간단하다. undo는 마지막에 했던 동작의 반대되는 행동을하는 메소드이므로, 마지막 동작이 '+' 였다면 '-'를, '*'였다면 '/'를 해주면 된다.


```
@Override
public void undo() {
    this.operator = oppositeOperator(this.operator); // 현재 operator의 반대되는 operator 받아오기
    this.execute(); // operator가 반대인것 빼고는 계산과정은 같기때문에 this.execute 메소드 실행
}
```

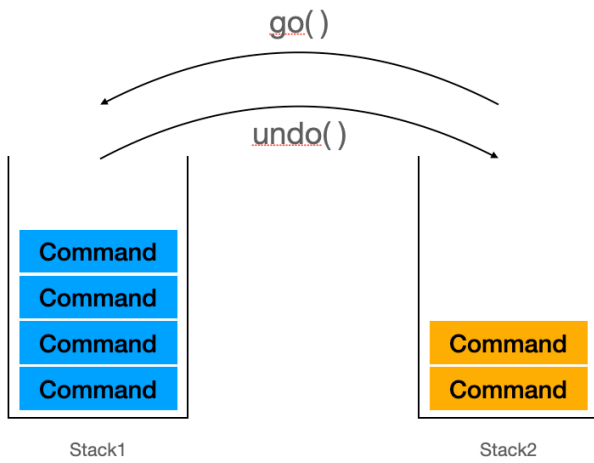
따라서, 현재의 operator를 반대되는 operator로 바꿔준뒤, 똑같이 execute() 메소드를 실행시켜주면 이전 값 이 연산된다.

[실행결과]

```
a=5/4
b=1/2
c=2/3
d=4/7
e=13/7
0/1 + 5/4 = 5/4
5/4 + 1/2 = 7/4
7/4 - 2/3 = 13/12
13/12 * 4/7 = 13/21
13/21 / 13/7 = 1/3
1/3 * 13/7 = 13/21
13/21 / 4/7 = 13/12
13/12 + 2/3 = 7/4
7/4 - 1/2 = 5/4
5/4 - 5/4 = 0/1
no undo command in the stack!
```

[추가코드]

문서파일의  과 같은 기능을 만들었다. undo() 메소드는 뒤로가기 같은 느낌을 받아서 뒤로갔을때 다시 앞으로 올 수 있는 go() 메소드를 만들었다.



stack1과 stack2를 위 그림과 같이 유지하여, `undo()` 메소드가 실행되면 `pop()` 되는 `Command`가 stack2로 push 되고, 다시 `go()` 메소드가 실행되면, stack2에서 `pop()` 되는 `Command`가 stack1으로 push되도록 구현했다.