*객체지향 설계와 패턴*  
**LAB ASSIGNMENT #12: Refactoring**

## Weight 10% 마감 6월 14일(월)

**목적: 리팩토링**

문제 1. 다음 프로그램의 코드 스멜은 무엇인지 지적하고 리팩토링하라.

(1)

public class Matcher {

public Matcher() {}

public boolean match(int[] expected, int[] actual,

int clipLimit, int delta)

{

// Clip "too-large" values

for (int i = 0; i < actual.length; i++)

if (actual[i] > clipLimit)

actual[i] = clipLimit;

// Check for length differences

if (actual.length != expected.length)

return false;

// Check that each entry within expected +/- delta

for (int i = 0; i < actual.length; i++)

if (Math.abs(expected[i] - actual[i]) > delta)

return false;

return true;

}

}

public class Matcher {  
 public Matcher() {  
 }  
  
 public boolean match(int[] expected, int[] actual, int clipLimit, int delta) {  
  
  
 if (actual.length != expected.length) {  
 return false;  
 }  
  
  
 *// 하나로 합친 부분* for (int i = 0; i < actual.length; i++) {  
 if (actual[i] > clipLimit) { *// 이곳으로 옮김* actual[i] = clipLimit;  
 } else if (Math.*abs*(expected[i] - actual[i]) > delta) {  
 return false;  
  
 }  
 }  
  
 return true;  
 }  
}

중복된 코드

for(int i = 0; i<actual.length; i++)가 중복되어 하나로 통합했다.

(2) 다음 프로그램으로 리팩토링 한 코드를 테스트하고 결과 화면을 캡처하라.

import org.junit.Test;

import static org.junit.Assert.assertTrue;

public class MatcherTest {

@Test

public void testMatch() {

Matcher matcher = new Matcher();

int[] expected = new int[] {10, 50, 30, 98};

int clipLimit = 100;

int delta = 5;

int[] actual = new int[] {12, 55, 25, 110};

assertTrue(matcher.match(expected, actual, clipLimit, delta));

actual = new int[] {10, 60, 30, 98};

assertTrue(!matcher.match(expected, actual, clipLimit, delta));

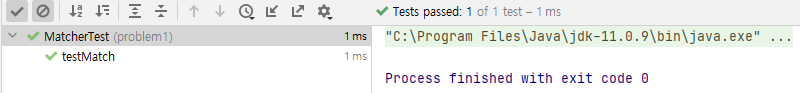
actual = new int[] {10, 50, 30};

assertTrue(!matcher.match(expected, actual, clipLimit, delta));

}

}

<테스트 결과화면>



**문제 2. Long Method**

첨부된 machine.zip 안의 코드를 열어 다음 사항에 답하라.

(1) Report.java 안의 코드 블록 4 개를 찾아서 어디서부터 어디인지 설명하고 리팩토링하여 함수로 추출하라.

블록1 : out.write("FACTORY REPORT\n");

블록2

Iterator<Machine> line = machines.iterator();  
while (line.hasNext()) {  
 Machine machine = (Machine) line.next();  
 out.write("Machine " + machine.name());  
  
 if (machine.bin() != null)  
 out.write(" bin=" + machine.bin());  
  
 out.write("\n");  
}  
out.write("\n");

블록3

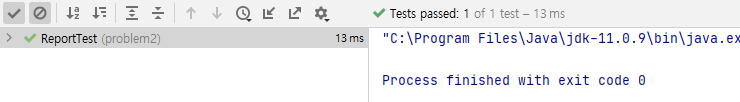
out.write("Robot");  
if (robot.location() != null)  
 out.write(" location=" + robot.location().name());  
  
if (robot.bin() != null)  
 out.write(" bin=" + robot.bin());  
  
out.write("\n");

블록4 : out.write("========\n");

(2) report() 메소드를 4개의 문장으로 리팩토링하라(Extract Method 방법을 사용할 것).

public class Report {  
 public static void writeHeader(Writer out) throws IOException{  
 out.write("FACTORY REPORT\n");  
 }  
  
 public static void writeMachines(Writer out, List<Machine> machines) throws IOException{  
 for (Machine machine : machines) {  
 machine.write(out);  
 }  
 out.write("\n");  
 }  
  
 public static void writeRobot(Writer out, Robot robot) throws IOException {  
 robot.write(out);  
 }  
  
 public static void writeFooter(Writer out) throws IOException {  
 out.write("========\n");  
 }  
  
 public static void report(Writer out, List<Machine> machines, Robot robot)  
 throws IOException   
 {  
  
 *writeHeader*(out);  
 *writeMachines*(out, machines);  
 *writeRobot*(out, robot);  
 *writeFooter*(out);  
  
 }  
}

<테스트 결과화면>



**문제 3. 논리 구조**

if (!((score > 700) ||

((income >= 40000) && (income <= 100000)

&& authorized && (score > 500)) ||

(income > 100000)))

reject();

else

accept();

(1) 위 논리를 DeMorgan의 법칙을 적용하여 가능한 간단하게 표현하라.

if((score <= 700)

    && ((income < 40000) || (income > 100000) || !authorized || (score <= 500))

    && (income <= 100000))

(2) 변수 이름을 다시 잘 정하여 조건을 다시 써보라.

boolean middleIncome = (income >= 40000) && (income <= 100000);

boolean highIncome = (income > 100000);

boolean highScore = (score > 700);

boolean midScore = (score > 500);

if(!(highScore || (middleIncome && authorized && midScore) || highIncome ))

(3) if 와 else 부분을 뒤집고 여러 개의 if 문장으로 나누어 쓰라(여러 곳에 accept() 문장이 나오도록).

if(score > 700)

    accept();

else if(( (income > 40000) &&(income < 100000)) && authorized && (score > 500))

    accept();

else if(income > 100000)

    accept();

else

    reject();

(4) 조건을 계산하는 메소드를 따로 추출하여 조건 식을 다시 써라.

boolean acceptable(int income, int score, boolean authorized){

    if(score > 700 || income> 10000)

        return true;

    if ( (income > 40000) &&(income < 100000) && authorized &&(score>500))

        return true;

    return false;

}

if(acceptable(income, score, authorized))

    accept();

else

    reject();

(5) 어떤 것이 더 간단하고 명확한가? 두 가지를 혼합할 수 있는가?

(4)가 가장 명확하고 간단하다고 생각한다. 다만 변수를 새로 지정한다면 더 보기좋다. 즉, (2), (4)를 혼합한 것이 가장 간결하다.

boolean acceptable(int income, int score, boolean authorized){

    boolean middleIncome = (income >= 40000) && (income <= 100000);

    boolean highIncome = (income > 100000);

    boolean highScore = (score > 700);

    boolean midScore = (score > 500);

    if(highScore || highIncome)

        return true;

    if ( middleIncome && authorized && midScore))

        return true;

    return false;

}

if(acceptable(income, score, authorized))

    accept();

else

    reject();

**문제 4. Template Method**

다음 코드를 보고 물음에 답하라.

public abstract class Party { }

public class Person extends Party {

private String firstName;

private String lastName;

private Date dob;

private String nationality;

public void printNameAndDetails() {

System.out.println("Name: " + firstName + " " + lastName);

System.out.println("DOB: " + dob.toString() + ", Nationality: " + nationality);

}

}

public class Company extends Party {

private String name;

private String companyType;

private Date incorporated;

public void PrintNameAndDetails() {

System.out.println("Name: " + name + " " + companyType);

System.out.println("Incorporated: " + incorporated.toString());

}

}

(1) 코드 클론이 있는가? 있다면 표시하라.

public void PrintNameAndDetails(){ … }

(2) Template method를 이용하여 코드 클론을 제거하라.

public abstract class Party {  
 public void printNameAndDetails() {  
 printName();  
 printDetails();  
 }  
  
 public abstract void printName();  
  
 public abstract void printDetails();  
}

import java.util.Date;  
  
public class Company extends Party{  
 private String name;  
 private String companyType;  
 private Date incorporated;  
  
  
 @Override  
 public void printName() {  
 System.*out*.println("Name: " + name + " " + companyType);  
 }  
  
 @Override  
 public void printDetails() {  
 System.*out*.println("Incorporated: " + incorporated.toString());  
 }  
}

public class Person extends Party {  
 private String firstName;  
 private String lastName;  
 private Date dob;  
 private String nationality;  
  
 @Override  
 public void printName() {  
 System.*out*.println("Name: " + firstName + " " + lastName);  
 }  
  
 @Override  
 public void printDetails() {  
 System.*out*.println("DOB: " + dob.toString() + ", Nationality: " + nationality);  
  
 }  
}

**문제 5. Bad Smells 찾기**

첨부한 파일 <PlanGame.zip>에 있는 프로그램의 Bad Smell을 찾는 문제이다. Planning Game은 다음과 같은 기능을 가지고 있다.

“고객은 새 카드를 만들고, 기존 카드를 분할하고, 기존 카드를 삭제하고, 계획에 대한 간단한 분석을 얻을 수 있다. 플레이어는 카드에 비용 견적을 입력하고 실행 속도 견적을 업데이트 할 수 있다. 또한 누구나 카드에 입력하거나 테두리를 드래그하여 이동할 수 있다. 또한 각 작업 후에 업데이트되는 요약 정보가 화면 하단에 표시된다.”

이 프로그램의 코드는 Table (버튼을 포함한 전체 애플리케이션), Background(카드가 놓인 위치) 및 Card (인덱스 카드를 나타냄)의 세 가지 객체로 구성된다.

이 코드에서 파악되는 모든 Bad smell을 찾아 위치와 이유를 설명하라. 리팩토링 카탈로그의 목록을 참조하라.

**긴 메서드**

Table 클래스 안에 addActionListener등이 포함된 메소드들이 너무 길다. 시작과 끝을 모르겠다.

Card클래스의 생성자가 너무 길다

**큰 클래스**

Table클래스와 Card클래스 모두 가지고 있는 책임이 너무 많다.

**중복된 코드**

패널에 버튼을 추가하는 코드가 약간만 다르고 거의 비슷함

Table과 Card 모두 border color를 설정함

Card 클래스에서 title과 body를 추가하는 방식이 비슷함

**발산하는 변경**

GUI와 addActionListener로 추가한 action들을 독립적으로 변경할 수 없음

**shotgun surgery**

card의 경계 색깔을 바꾸는 코드가 두 객체에 존재함

**Switch 문장**

rotateCost()메소드에서 costLabel에 대한 if-else if문이 많음. OCP를 준수해서 상태패턴을 쓰는게 좋아보임

**제출물**

제출하여야 할 것:

1. 각 문제에 대한 답을 문제 아래 빈칸에 기술하라.

2. 구현한 결과:

\* 원시코드

\* 실행 한 후의 결과 화면

**평가 기준**

과제 #12다음과 같은 기준으로 평가할 것임:

1. 주어진 질문에 대한 답의 정확성

2. 구현의 정확성

3. 리팩토링 결과물에서 볼 수 있는 리팩토링에 대한 이해 수준