2. 메서드 추출과 인라인 메서드

[예제 1] 리팩토링 : 메서드 추출(extract method)

- **메서드 추출**은 **코드 조각**을 고유한 메서드로 변환하는 것이다.(2번이상 여러번 사용되는 경우)
- 코드 조각을 빼내어, 코드 목적이 잘 드러나도록 **직관적 이름의 메서드를 생성**한다.

〈리팩토링 전〉

```
class Test{
  final static double PI = 3.141592;
  public static void main(String args[]){
    double radius = 100;
    double area = PI * radius * radius;     //원의 면적을 구하는 코드(코드 조각)
    System.out.println(area);
  }
}
```

〈원의 면적을 구하는 경우〉

↓

↓ 리팩토링 : 메서드 추출, 메서드 cal_area() 추출

↓

〈리팩토링 후〉

```
class Test{
  final static double PI = 3.141592;
  public static void main(String args[]){
    double radius = 100;
    System.out.println(cal_area(radius));
  }
  static double cal_area(double radius){    //추출된 메서드(메서드 이름을 잘 지어야 함)
    double area = PI * radius * radius;
    return area;                            //메서드는 하나의 값을 반환하는 것을 권장
  }
}
```

// 메서드 추출 이점

- 코드 중복을 줄인다. 신속, 정확하게 가독성을 향상시킨다.
- 추출된 메서드 자체가 좋은 문서화가 된다.(이해하기 쉬운 이름으로 된 메서드)
- 메서드가 작게 쪼개져 있으면, 재사용될 가능성이 높아진다.

[예제 2] 리팩토링 : 인라인 메서드(inline method)

- 먼저, 인라인 메서드(inline method)는 메서드 추출(extract method)의 **반대**이다.
- 인라인 메서드는 메서드 호출 위치에 메서드 내용을 직접 삽입한다.**(해당 메서드는 삭제)**

〈리팩토링 전〉

```
public void mymethod(int n)
{
  if (isEven(n))                          //메서드 isEven(n) 호출
    System.out.println("짝수");
  else
    System.out.println("홀수");
}
boolean isEven(int n)                     //메서드 isEven(n) 정의 - 메서드 기능이 단순
{
  return n % 2 == 0 ? true : false;
}
```

〈짝수/홀수 판별하는 경우〉

↓

↓ 리팩토링 : 메서드 내용 직접 삽입, 메서드 isEven()은 삭제

↓

〈리팩토링 후〉

```
public void mymethod(int n)
{
  if (n % 2 == 0)                         //메서드 isEven(n) 위치에 메서드 기능 삽입
    System.out.println("짝수");
  else
    System.out.println("홀수");
}
```

- **인라인 메서드**는 매소드 기능이 **지나치게 단순**한 경우에 적용(오직 한번만 사용되는 경우)
- **인라인 메서드**는 **매소드 기능이 단순**하여 메서드 이름만큼이나 명확할 때에 적용한다.

- **인라인 메서드**는 매소드가 잘못 추출되었을 때 새로운 리팩토링을 위해 사용하기도 한다.

기출문제 분석

1. 〈보기〉와 같이 프로그램을 변경하는 작업으로 가장 옳은 것은? [2018년 서울 7급]

```
----<변경 전>----------------------------------------------------------------------------
public class Painter {
  public double calculatePaintRequired(double row, double col, double paintPerUnit) {
    double area = row * col;
    double paintRequired = area/paintPerUnit;
    return paintRequired;
  }
}
----<변경 후>----------------------------------------------------------------------------
public class Painter {
  public double calculatePaintRequired(double row, double col, double paintPerUnit) {
    double area = calculateArea(row, col);
    double paintRequired = area/paintPerUnit;
    return paintRequired;
  }
  private double calculateArea(double row, double col) {
    double area = row * col;
    return area;
  }
}
-----------------------------------------------------------------------------------------
```

① 재사용(reuse)　② 재공학(reengineering)
③ 역공학(reverse engineering)　④ 리팩토링(refactoring)

♤ **리팩토링 - 메서드 추출**

• 변경된 내용은 **면적**을 구하는 메서드 calculateArea(row, col)를 생성하였다.(가독성)

// 리팩토링하는 목적

• 소프트웨어 디자인을 개선시킨다.
• 소프트웨어 가독성을 높이고 유지보수를 편하게 한다.
• 기능은 그대로 두면서 내부 논리구조를 바꾸고 개선하는 유지보수 행위이다.

정답 : ④

2. 다음은 리팩토링 전후의 코드 변화이다. 적용된 리팩토링 기법에 해당하는 것은? [2019년 국가 7급]

```
-----------------------------------------------
public void myMethod(int n) {
  if (isEven(n)) System.out.println("Even");
  else           System.out.println("Odd");
}
private boolean isEven(int number) {
  return number % 2 ==  0;
}
-----------------------------------------------
```

```
-----------------------------------------------
public void myMethod(int n) {
  if (n % 2 == 0) System.out.println("Even");
  else            System.out.println("Odd");
}
-----------------------------------------------
```

① 메서드 은폐(hide method)
② 메서드 추출(extract method)
③ 메서드 내용 직접 삽입(inline method)
④ 메서드를 매개변수로 전환(parameterize method)

♤ **리팩토링 : 메서드 내용 직접 삽입(inline method)**

• 메서드 기능이 매우 단순하여 메서드 이름만 보아도 쉽게 판단될 때
• 메서드를 호출하는 위치에 메서드의 기능을 넣고, 해당 메서드는 삭제한다.

```
// 짝수 / 홀수 판별하는 경우
public void mymethod(int n) {
  if (isEven(n)) System.out.println("짝수");
  else           System.out.println("홀수");
}
boolean isEven(int n) { return n % 2 == 0 ? true : false; }
```

↓ 리팩토링 : 메서드 내용 직접 삽입, 메서드 isEven()은 삭제

```
public void mymethod(int n) {
  if (n % 2 == 0) System.out.println("짝수");
  else            System.out.println("홀수");
}
```

정답 : ③

3. 다음 코드 변환에 대한 리팩토링 기법에 맞는 최적의 작업은? [2020년 국가 7급]

```
-----------------------------------------
void printOwing(double amount) {
   printBanner();
   System.out.println("name:"+_name);
   System.out.println("amount:"+_amount);
}
-----------------------------------------
                ⇓
-----------------------------------------
void printOwing(double amount) {
   printBanner();
   printDetails(amount);
}
void printDetails(double amount) {
   System.out.println("name:"+_name);
   System.out.println("amount:"+_amount);
}
-----------------------------------------
```

① 메서드 이동 작업이다.
② 메서드 추출 작업이다.
③ 인터페이스 추출 작업이다.
④ 템플릿 메서드 형성 작업이다.

♤ **메서드 추출(extract method)**

- **메서드 추출**은 **코드 조각**을 고유한 메서드로 변환하는 것이다.
- 코드 조각을 빼내어, 코드 목적이 잘 드러나도록 **직관적 이름의 메서드를 생성**한다.

```
-----------------------------------------
   System.out.println("name:"+_name);
   System.out.println("amount:"+_amount);
-----------------------------------------
          ⇓ 메서드 추출
-------------------------------------------------------------------------------------
void printDetails(double amount) {      //추출된 메서드(메서드 이름을 잘 지어야 함)
   System.out.println("name:"+_name);
   System.out.println("amount:"+_amount);
}
-------------------------------------------------------------------------------------
```

정답 : ②

4. 학사관리시스템에서 파악된 다음 코드 악취를 해결하기 위한 리팩토링 기법은? [2023년 국가 7급]

> ○ enrol() 메서드의 코드가 4,000라인이 넘어 가독성이 낮다.
> ○ study()와 credit() 메서드는 각각 200라인 정도지만, 두 메서드는 100라인 정도가 중복되어 있다.

① 메서드 추출(extract method)　② 위임 숨기기(hide delegate)
③ 중개자 제거(remove middle man)　④ 클래스 인라인화(inline class)

♤ **메서드 추출**

- **메서드 추출**은 **코드 조각**을 고유한 메서드로 변환하는 것이다.
- 코드 조각을 빼내어, 코드 목적이 잘 드러나도록 **직관적 이름의 메서드를 생성**한다.
- **메서드 추출**은 코드 **중복**을 줄인다.
- 추출된 메서드 자체가 좋은 문서화가 된다.(이해하기 쉬운 이름으로 된 메서드)
- 신속하게, 정확하게 재사용성과 **가독성**을 **향상**시킨다.
- 메서드가 작게 쪼개져 있으면, 재사용될 가능성이 높아진다.
- enrol() 메서드의 코드가 4,000라인이 넘어 **가독성**이 낮으므로 메서드가 작게 쪼개야 한다.

정답 : ①