1. 코딩과 리펙토링

1. 코딩(coding)

구현은 원시코드를 작성하는 것이다. 이를 코딩(coding)이라고도 한다.
코딩은 상세설계에 기술된 내용과 일치되어야 한다.
원시코드와 소프트웨어 품질은 매우 밀접하다.

〈바람직한 프로그램 코드〉
- 원시코드는 간결하고, 명료하게 작성해야 한다.
- 원시코드 작성에서 임시변수의 사용은 피하는 것이 좋다.
- 빠른 프로그램보다 먼저 명료한 프로그램을 작성하는 것이 좋다.
- 원시코드 작성단계에서 발생한 설계변경은 설계문서에 반영해야 한다.
- 다양한 오류 처리를 위해서 예외처리를 효과적으로 사용하는 것이 필요하다.

// 구조적 프로그래밍
- 구조적 프로그래밍은 **goto 문**을 가능한 사용하지 않고 프로그래밍하는 것이다.
- 구조적 프로그래밍은 **순차적, 선택적, 반복적**으로 프로그래밍하는 것이다.
- 대부분의 현대 프로그래밍 언어는 '구조적 언어'의 문장을 제공하고 있다.
- '구조적 언어'의 문장을 사용하면 손쉽게 하향식 프로그래밍을 구현할 수 있다.
- '구조적 언어' 문장 : if, while, repeat, for, case문

코딩 오류는 프로그래머가 부딪치는 현실이다. 오류 제거에 많은 시간이 소요된다.

〈코딩 오류 종류〉
메모리 누수, 인덱스 오류, 별칭 오류, 버퍼 오류, 수식 오류, 자료형(type) 오류 등

- 메모리 누수 현상은 쓰레기 회수를 자동으로 처리하지 않는 C/C++에서 자주 발생된다.

2 http://cafe.daum.net/pass365(홍재연)

2. 리팩토링(refactoring)

리팩토링은 '소프트웨어 기능(결과)은 변경되지 않도록 하면서
원시코드의 구조를 합리적으로 재조정' 하는 것이다.

〈리팩토링 목적〉

- 리팩토링은 소프트웨어 **디자인을 개선**시킨다.
- 리팩토링은 소프트웨어 가독성을 높이고 유지보수를 편하게 한다.
- 리팩토링은 버그 찾는데 도움이 되며, 프로그램을 빨리 작성할 수 있도록 한다.
- 리팩토링은 기능은 그대로 두면서 내부 논리구조를 바꾸고 개선하는 유지보수 행위이다.
- 리팩토링(refactoring)은 **버그를 없애**거나 **새로운 기능을 추가하는 행위는 아니다.**
- 리팩토링은 소프트웨어 **원래의 기능은 그대로 유지**된다.

// 코드 스멜(code smell)

- 코드 스멜은 원시코드에서 **심각한 문제**를 일으킬 가능성이 있는 코드의 증상이다.
- 코드 스멜은 1990년대 켄트 벡과 워즈위키에 의해 고안된 것이다.
- 무엇이 코드 스멜인지 아닌지의 여부를 결정하는 것은 **주관적인 것**이다.
  → 언어, 개발자, 개발 방법에 따라 다양하다.
- 코드 스멜 확인 자동 도구 : 체크스타일, PMD, 파인드벅스 등

// 일반적인 코드 스멜

- 읽기 어려운 코드
- 과도하게 긴 식별자
- 과도하게 짧은 식별자
- 과도한 데이터의 반환
- 긴 메서드(long method)
- 임시 필드(temporary field)
- 중복된 로직을 가진 프로그램
- 복잡한 조건문이 포함된 프로그램
- 긴 파라미터 리스트(long parameter list)
- 추측에 근거한 일반화(speculative generality)
- 병렬적인 상속 구조(parallel inheritance hierarchies)
- 불필요한 설명문(comment) : 간결성을 확보한 코드는 설명문이 필요 없다.

탐구

리팩토링 종류

메서드 정리	• 메서드 추출 extract method • 메서드 내용 직접 삽입 inline method • 임시변수 내용 직접 삽입 inline temp • 임시변수 분리 split temporary variable • 직관적 임시변수 사용 introduce explaining variable • 임시변수를 메서드 호출로 전환 replace temp with query • 매개변수로의 값 대입 제거 remove assignments to parameters • 메서드를 메서드 객체로 전환 replace method with method object
객체 간의 기능 이동	• 필드 이동 move field • 메서드 이동 move method • 대리 객체 은폐 hide delegate • 클래스 추출 extract class • 클래스 내용 직접 삽입 inline class • 과잉 중개 메서드 제거 remove middle man • 외래 클래스에 메서드 추가 introduce foreign method
데이터 체계화	• 값을 참조로 전환 change value to reference • 참조를 값으로 전환 change reference to value • 배열을 객체로 전환 replace array with object • 필드 자체 캡슐화 self encapsulate field • 데이터 값을 객체로 전환 replace data value with object • 관측 데이터 복제 duplicate observed data • 필드 캡슐화 encapsulate field • 컬렉션 캡슐화 encapsulate collection • 분류 부호를 클래스로 전환 replace type code with class • 레코드를 데이터 클래스로 전환 replace record with data class • 분류 부호를 하위클래스로 전환 replace type code with subclasses • 하위클래스를 필드로 전환 replace subclass with fields
조건문 간결화	• 주장 넣기 introduce assertion • 여러 겹의 조건문을 감시 절로 전환 • 조건문 쪼개기 decompose conditional • 중복 조건식 통합 consolidate conditional expression • 조건문을 재정의로 전환 replace conditional with polymorphism • 조건문의 공통 실행 코드 빼내기 consolidate duplicate conditional fragments • 제어 플래그 제거 remove control flag • null 검사를 널 객체에 위임 introduce null object

// 리팩토링 종류 계속

메서드 호출 단순화	• 매개변수 추가 add parameter • 메서드명 변경 rename method • 매개변수 제거 remove parameter • 메서드를 매개변수로 전환 parameterize method • 매개변수를 메서드로 전환 replace parameter with explicit methods • 상태 변경 메서드와 값 반환 메서드를 분리 separate query from modifier • 객체를 통째로 전달 preserve whole object • 매개변수 세트를 객체로 전환 introduce parameter object • 매개변수 세트를 메서드로 전환 replace parameter with method • 쓰기 메서드 제거 remove setting method • 메서드 은폐 hide method • 생성자를 팩토리 메서드로 전환 replace constructor with factory method • 하향 타입 변환을 캡슐화 encapsulate downcast • 에러 부호를 예외 통지로 교체 replace error code with exception • 예외처리를 테스트로 교체 replace exception with test
일반화 처리	• 필드 상향 pull up field • 필드 하향 push down field • 메서드 상향 pull up method • 메서드 하향 push down method • 생성자 내용 상향 pull up constructor body • 인터페이스 추출 extract interface • 자식클래스 추출 extract subclass • 부모클래스 추출 extract superclass • 계층 병합 collapse hierarchy • 템플릿 메서드 형성 form template method • 상속을 위임으로 전환 replace inheritance with delegation • 위임을 상속으로 전환 replace delegation with inheritance

기출문제 분석

1. 바람직한 프로그램 코드에 대한 설명으로 옳지 않은 것은? [2009년 국가 7급]

① 다양한 오류 처리를 위한 간결한 구조를 위해서 예외처리를 효과적으로 사용하는 것이 필요하다.
② 소스코드 작성단계에서 발생한 설계변경은 설계문서에 반영할 필요는 없다.
③ 임시변수의 사용은 피하는 것이 좋다.
④ 빠른 프로그램보다 먼저 명료한 프로그램을 작성하는 것이 좋다.

♤ **바람직한 프로그램 코드**

- 소스코드 작성단계에서 발생한 설계변경은 설계문서에 반영할 필요는 없다.(×)
  → 설계변경은 설계문서에 당연히 반영되어야 한다.
  → 설계문서는 지속적으로 현실에 맞게 수정되어야 한다.

정답 : ②

2. 리팩토링(refactoring)에 대한 설명으로 옳은 것으로만 묶은 것은? [2012년 전산 7급]

가. 소프트웨어의 디자인을 개선시킨다.
나. 기능을 지속적으로 추가하는 작업이다.
다. 소프트웨어를 이해하기 쉽게 만들고, 프로그램을 빨리 작성하게 도와준다.
라. 겉으로 보이는 동작의 변화없이 내부구조를 변경하는 것이다.
마. 버그를 수정하는 작업이다.

---

① 가, 나, 다　　② 가, 다, 라
③ 나, 다, 마　　④ 나, 라, 마

♤ **리팩토링**

- 소프트웨어 디자인을 개선시킨다.
- 소프트웨어 가독성을 높이고 유지보수를 편하게 한다.
- 버그 찾는데 도움이 되며, 프로그램을 빨리 작성할 수 있도록 한다.
- 리팩토링(refactoring)은 버그를 없애거나 새로운 기능을 추가하는 행위는 아니다.

정답 : ②

3. 리팩토링에 대한 설명으로 옳지 않은 것은? [2017년 국가 7급]

① 리팩토링의 대상은 읽기 어려운 코드, 중복된 로직의 코드, 복잡한 조건문이 있는 코드 등이 대표적인 것이다.
② 리팩토링은 실행 중인 프로그램의 기능 변경이 수반되어야 한다.
③ 리팩토링을 통해서 프로그램의 이해가 쉬워진다.
④ 리팩토링은 결함을 찾는 데 도움을 준다.

♤ **리팩토링(refactoring)**

• 리팩토링은 실행 중인 프로그램의 기능 변경이 수반되어야 한다.(×)
→ 리팩토링은 새로운 기능을 추가하는 행위는 아니다.
→ 기존의 소프트웨어에 대해 새로운 기능 추가나 개선은 유지보수 단계에서 실시한다.

• 소프트웨어 공학에서 리팩토링은 '소프트웨어 기능(결과)은 변경되지 않도록 하면서
• 원시코드의 구조를 합리적으로 재조정' 하는 것이다. 소프트웨어 원래의 기능은 유지된다.

정답 : ②

4. 리팩토링의 대상이 되는 코드 스멜(code smell)에 해당하지 않는 것은? [2013년 국가 7급]

① 읽기 어려운 프로그램
② 중복된 로직을 가진 프로그램
③ 외부에서 보이는 기능을 변경해야 하는 프로그램
④ 복잡한 조건문이 포함된 프로그램

♤ **리팩토링의 대상이 되는 코드 스멜(code smell) – 코드 냄새**

• 외부에서 보이는 기능을 변경해야 하는 프로그램(×)
→ 외부에서 보이는 기능으로는 원시코드 구조를 알 수 없으므로
→ 리팩토링 대상이 아니다.

• 리팩토링(refactoring)은 버그를 없애거나 새로운 기능을 추가하는 행위는 아니다.

정답 : ③