

소프트웨어공학

2018년도 국가공무원 5급(기술) 공개경쟁채용 제2차시험

응시번호 :                      성명 :

제 1 문. 소스 코드 리팩토링(refactoring)에 관한 다음 물음에 답하시오.        (총 20점)

- 1) 리팩토링의 개념을 기술하고, 리팩토링을 통해 얻을 수 있는 장점 2가지를 설명하시오. (6점)
- 2) 소프트웨어 개발 프로젝트의 소스 코드를 살펴보는 과정에서 C1 클래스에 속한 f1() 메소드의 코드가 3,000 라인이 넘어 가독성이 떨어지고, 각각 200 라인 정도의 길이를 가지는 f2() 메소드와 f3() 메소드의 코드 사이에 100 라인 정도가 중복되어 있음을 알게 되었다. f1()이 가지는 코드 악취(bad smell)와 f2(), f3()가 가지는 코드 악취를 제시하고, 각각에 대해 적용 가능한 리팩토링 방법을 설명하시오. (6점)
- 3) 다음은 소프트웨어 개발 프로젝트에서 작성한 API 코드 중 일부이다. 소스 코드 구현 및 유지보수 관점에서 코드가 가지고 있는 문제점을 설명하고, 이를 해결하는 데 적절한 리팩토링 방법을 설명하시오. (8점)

```
...
bool addUser(int id, int age, int year, int month, int
day, string name, string addr, string phone, string
memo);
bool modifyUser(int id, int age, int day, int month, int
year, string memo, string name, string phone, string
addr);
...
```

제 2 문. 다음 C 프로그램을 주어진 테스트 케이스를 이용하여 테스트 하고자 한다.  
물음에 답하시오. (단, 단축연산(short-circuit evaluation)은 수행하지 않는다)  
(총 30점)

줄 번호	소스 코드	테스트 케이스
1	void foo(int x, int y) {	T = {t1:<x = -3, y = 3>, t2:<x = 2, y = -5>}
2	if (x > 0 && y < 3)	
3	x = y + 3;	
4	else	
5	x = y - 3;	
6	if (x <= 0    y > 0)	
7	z = 2 * x;	
8	else	
9	z = x * x;	
10	printf("x = %d\n", x);	
	printf("z = %d\n", z);	
	}	

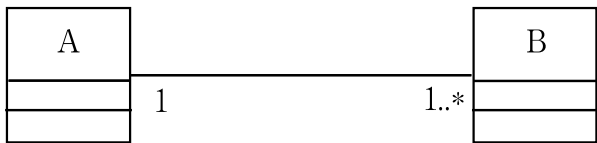
- 1) 문장 커버리지(statement coverage)를 계산하고, 근거를 제시하시오. (8점)
- 2) 조건 커버리지(condition coverage)를 계산하고, 근거를 제시하시오. 만일 조건 커버리지 값이 1이 아니라면, 조건 커버리지 값이 1이 되도록 테스트 케이스를 추가하시오. (14점)
- 3) 결정 커버리지(decision coverage)를 계산하고, 근거를 제시하시오. (8점)

제 3 문. 디자인 패턴(Design Pattern)은 생성 패턴, 구조 패턴, 행위 패턴으로 분류될 수 있다. 다음 물음에 답하시오. (총 30점)

- 1) 생성 패턴의 정의를 기술하고, Factory Method 패턴과 Singleton 패턴을 설명하시오. (10점)
- 2) 구조 패턴의 정의를 기술하고, Adapter 패턴과 Facade 패턴을 설명하시오. (10점)
- 3) 행위 패턴의 정의를 기술하고, Command 패턴과 Visitor 패턴을 설명하시오. (10점)

제 4 문. UML 클래스 다이어그램에 관한 다음 물음에 답하시오. (총 20점)

- 1) 다음 클래스 A, B 간 연관(association)의 다중성(multiplicity)에 대하여 설명하시오. (4점)



- 2) 다음 Java 클래스들의 선언 부분을 하나의 클래스 다이어그램으로 표현하시오. (단, HashMap을 템플릿 클래스가 아닌 Graph와 Node 간의 연관으로 표현하고 그 의미를 나타내야 한다. HashMap의 키는 nodeId이다) (16점)

```

abstract class Graph {
    private HashMap<Integer, Node> nodes; // nodeId:Integer
    private Edge edges[];
    abstract public int getCost();
    ...
}

class TaskGraph extends Graph {
    @Override
    public int getCost() { ... }
    ...
}

abstract class Node {
    private int nodeId;
    protected Edge predecessor;
    protected Edge successor;
    abstract public int getCost();
    ...
}

class TaskNode extends Node {
    @Override
    public int getCost() { ... }
    ...
}

class Edge {
    private Node predecessorNode;
    private Node successorNode;
    ...
}
  
```