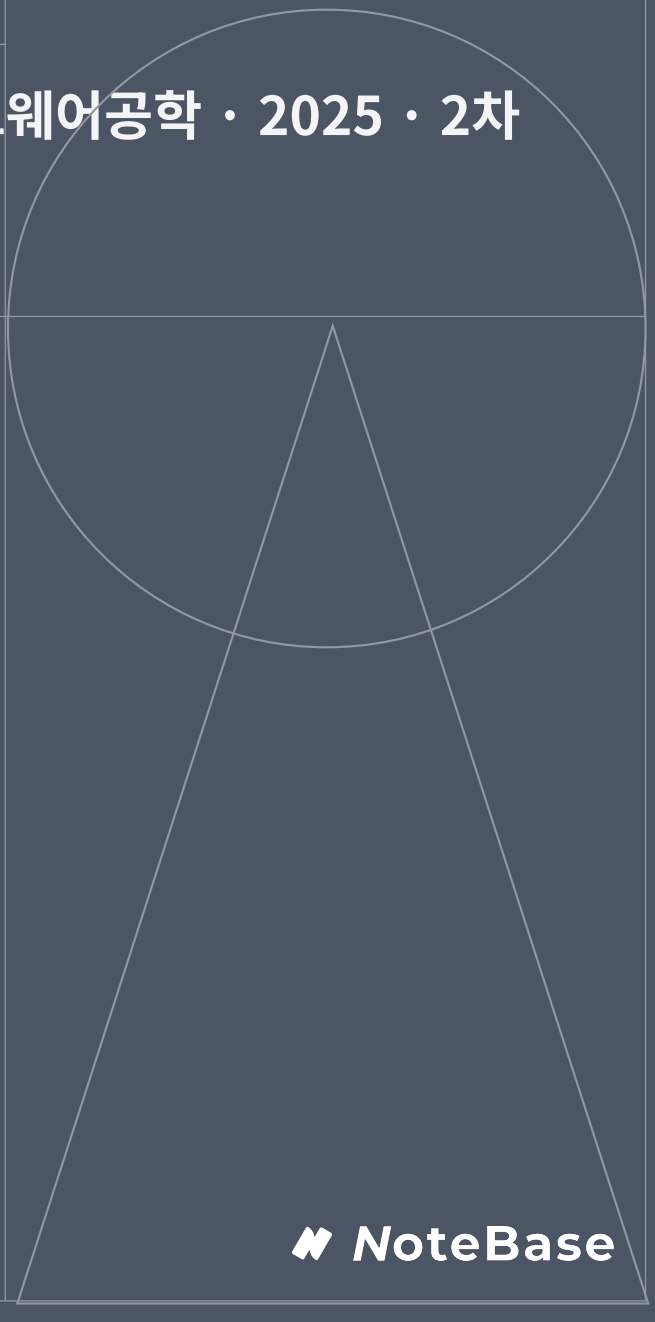
기출OX 해설집

공무원/국가직/7급 · 소프트웨어공학 · 2025 · 2차



NoteBase

기출OX 해설집

공무원/국가직/7급 · 소프트웨어공학 · 2025 · 2차

공무원/국가직/7급 · 소프트웨어공학 · 2025 · 2차

문 1. 다음 중 응집도(cohesion)가 가장 높은 것은?

① 절차적(procedural) 응집도

② 시간적(temporal) 응집도

③ 기능적(functional) 응집도

④ 논리적(logical) 응집도

정답 3번

출제 포인트

이 문제는 모듈 응집도의 상대적 강도를 비교하여 가장 높은 응집도를 고를 수 있는지를 묻는다.

핵심 개념

응집도는 한 모듈 내부 요소들이 하나의 목적에 얼마나 밀접하게 관련되는지를 뜻한다. 기능적 응집도는 모듈 전체가 하나의 명확한 기능만 수행하는 상태이다. 절차적 응집도는 처리 순서가 비슷한 작업을 묶은 상태이다. 시간적 응집도는 같은 시점에 실행되는 작업을 묶은 상태이다. 논리적 응집도는 유사한 성격의 기능을 한 모듈에 넣고 상황에 따라 선택하는 상태이다.

NoteBase

선지별 해설

1번 X: 절차적 응집도는 실행 순서가 관련된 처리들을 한 모듈로 묶은 경우이다. 순서가 연결될 뿐 단일 기능만 수행하는 구조는 아니므로 가장 높은 응집도가 아니다. 틀린 표현은 '가장 높은'이다. 올바르게는 '절차적 응집도는 기능적 응집도보다 낮다.'이다.
2번 X: 시간적 응집도는 초기화나 종료 처리처럼 같은 시점에 함께 수행되는 작업을 묶은 경우이다. 수행 시점만 같을 뿐 목적이 하나로 수렴하지 않을 수 있으므로 높은 응집도가 아니다. 틀린 표현은 '가장 높은'이다. 올바르게는 '시간적 응집도는 기능적 응집도보다 낮다.'이다.
3번 O: 기능적 응집도는 모듈의 모든 요소가 하나의 기능 수행만을 위해 결합된 경우이다. 단일 목적성과 책임이 가장 분명하므로 보기들 중 응집도가 가장 높다.
4번 X: 논리적 응집도는 입력 처리, 출력 처리처럼 성격이 비슷한 여러 기능을 한 모듈에 모아 두고 조건에 따라 선택하는 경우이다. 비슷한 종류라는 공통점은 있지만 실제 기능은 여러 개이므로 가장 높은 응집도가 아니다. 틀린 표현은 '가장 높은'이다. 올바르게는 '논리적 응집도는 관련 기능을 묶은 수준으로 기능적 응집도보다 낮다.'이다.

오답 포인트

①의 절차적과 ④의 논리적은 어느 정도 체계적으로 묶인 것처럼 보여 헷갈리기 쉽다. 그러나 응집도의 핵심 기준은 순서성이나 유사성이 아니라 모듈이 하나의 기능만 수행하는지 여부이다.

다시 나오면 이렇게 판단

응집도 문제는 먼저 모듈이 단일 기능만 수행하는지를 본다. 하나의 기능이면 기능적 응집도이고, 수행 순서로 묶이면 절차적, 같은 시점으로 묶이면 시간적, 유사 기능의 집합이면 논리적 응집도이다.

암기 포인트

가장 높은 응집도는 하나의 기능만 남는 기능적 응집도이다.

NoteBase

문 2. 다음은 웹사이트 회원가입 양식에 나이를 입력하는 텍스트 박스의 요구사항이다. 경곗값 분석 기법을 이용해 테스트 케이스를 작성할 경우, 입력값의 집합으로 옳은 것은?

회원의 나이를 입력하는 텍스트 박스에는 0세 이상, 120세 이하의 값만 입력해야 한다.

① 0, 60, 120

② -10, 0, 120, 130

③ 0, 1, 2, 118, 119, 120

④ -1, 0, 1, 119, 120, 121

정답 4번

출제 포인트

이 문제는 경곗값 분석에서 최소값과 최대값의 바로 바깥값과 바로 안쪽값까지 함께 고르는 기준을 묻는 문제이다.

핵심 개념

경곗값 분석은 입력 범위의 경계에서 오류가 잘 발생한다는 점을 이용해 최소값과 최대값 주변의 값을 집중적으로 시험하는 기법이다. 이 문제의 허용 범위는 0 이상 120 이하이므로 핵심 경계는 0과 120이고, 대표 테스트 값은 -1, 0, 1, 119, 120, 121이다. 단순 대표값만 고르는 것은 동등 분할에 가깝고, 경곗값 분석은 경계의 안팎을 함께 본다는 점이 다르다.

NoteBase

선지별 해설

① X
0, 60, 120은 최소 경계와 최대 경계, 중간값만 고른 집합이다. 틀린 부분은 경계 바로 바깥값과 바로 안쪽값 검사가 빠졌다는 점이다. 경곗값 분석으로는 -1, 0, 1, 119, 120, 121처럼 경계 주변을 함께 넣어야 한다.

② X
-10, 0, 120, 130은 경계 밖 값을 포함하지만 경계에 바로 인접한 값이 아니다. 틀린 부분은 -10과 130처럼 경계에서 멀리 떨어진 값을 대표 경곗값처럼 취급한 점이다. 실제로는 최소 바로 바깥값 -1과 최대 바로 바깥값 121, 그리고 바로 안쪽값 1과 119를 써야 한다.

③ X
0, 1, 2, 118, 119, 120은 모두 허용 범위 안의 값만 포함한다. 틀린 부분은 유효 범위 밖 경계값 검사가 없다는 점이다. 실제로는 잘못된 입력 검증을 위해 -1과 121도 포함해야 한다.

④ O
-1, 0, 1, 119, 120, 121은 최소 경계의 바깥값, 경계값, 안쪽값과 최대 경계의 안쪽값, 경계값, 바깥값을 모두 포함한다. 경곗값 분석의 전형적인 테스트 집합에 해당한다.

오답 포인트

경곗값 분석을 중간값이나 대표값을 고르는 방식으로 착각하면 ①처럼 60을 넣거나 ②처럼 멀리 떨어진 -10, 130을 넣게 된다. 또 유효값만 확인하면 된다고 생각하면 ③처럼 -1과 121을 빠뜨리게 된다.

다시 나오면 이렇게 판단

허용 범위가 주어지면 먼저 최소값과 최대값을 찾고, 그다음 최소-1, 최소, 최소+1, 최대-1, 최대, 최대+1 순서로 경계 주변 6개 값을 점검하면 된다.

암기 포인트

경곗값 분석은 min-1, min, min+1, max-1, max, max+1이다.

NoteBase

문 3. 다음 소프트웨어 개발 방법론과 모델링 도구를 바르게 연결한 것은?

(가) 정보공학 방법론
(나) 구조적 방법론
(다) 객체지향 방법론
A. 유스케이스 다이어그램(Usecase Diagram)
B. E-R 다이어그램(E-R Diagram)
C. 자료 흐름도(Data Flow Diagram)

① (가): A, (나): B, (다): C
② (가): A, (나): C, (다): B
③ (가): B, (나): C, (다): A
④ (가): C, (나): B, (다): A

정답 3번

출제 포인트

이 문제는 소프트웨어 개발 방법론별 대표 모델링 도구를 연결하는 문제이다.

핵심 정리

정보공학 방법론은 데이터 구조를 중심으로 분석하므로 E-R 다이어그램과 연결된다.
구조적 방법론은 기능과 처리 흐름을 분해하여 표현하므로 자료 흐름도와 연결된다.
객체지향 방법론은 사용자와 시스템 기능의 상호작용을 표현하므로 유스케이스 다이어그램과 연결된다.

NoteBase

선지별 해설

① X
(가)를 A로 연결한 부분이 틀렸다. 유스케이스 다이어그램은 객체지향 방법론의 요구사항 표현 도구이므로 정보공학 방법론과 맞지 않는다.
(나)를 B로 연결한 부분도 틀렸다. E-R 다이어그램은 구조적 방법론이 아니라 정보공학 방법론의 대표 모델링 도구이다.
(다)를 C로 연결한 부분도 틀렸다. 자료 흐름도는 객체지향이 아니라 구조적 방법론의 대표 도구이다.
틀린 표현 → 올바르게는 (가) B, (나) C, (다) A이다.

② X
(나)를 C로 연결한 부분은 맞다. 구조적 방법론은 자료 흐름도로 기능과 데이터 이동을 표현한다.
하지만 (가)를 A로 연결한 부분이 틀렸다. 정보공학 방법론은 E-R 다이어그램과 연결된다.
또한 (다)를 B로 연결한 부분도 틀렸다. 객체지향 방법론은 유스케이스 다이어그램과 연결된다.
틀린 표현 → 올바르게는 (가) B, (나) C, (다) A이다.

③ O
(가) 정보공학 방법론-B E-R 다이어그램의 연결이 맞다. 정보공학 방법론은 데이터 중심 분석과 데이터 모델링을 중시한다.
(나) 구조적 방법론-C 자료 흐름도의 연결이 맞다. 구조적 방법론은 기능 분해와 처리 흐름 표현에 자료 흐름도를 사용한다.
(다) 객체지향 방법론-A 유스케이스 다이어그램의 연결이 맞다. 객체지향 방법론은 행위자와 시스템 기능의 상호작용을 유스케이스로 표현한다.

④ X
(다)를 A로 연결한 부분은 맞다. 객체지향 방법론은 유스케이스 다이어그램과 연결된다.
하지만 (가)를 C로 연결한 부분이 틀렸다. 자료 흐름도는 정보공학 방법론이 아니라 구조적 방법론의 도구이다.
또한 (나)를 B로 연결한 부분도 틀렸다. E-R 다이어그램은 구조적 방법론이 아니라 정보공학 방법론의 도구이다.
틀린 표현 → 올바르게는 (가) B, (나) C, (다) A이다.

오답 포인트

E-R 다이어그램과 자료 흐름도를 모두 전통적 분석 도구로 묶어서 혼동하기 쉽다. 이 문제의 함정은 데이터 중심 도구인 E-R 다이어그램을 구조적 방법론으로, 기능 흐름 중심 도구인 자료 흐름도를 정보공학이나 객체지향 쪽으로 바꾸어 놓는 데 있다.

NoteBase

다시 나오면 이렇게 판단

방법론 문제는 도구가 무엇을 표현하는지부터 본다. 데이터 구조를 그리면 정보공학, 처리 흐름을 그리면 구조적, 사용자와 시스템 상호작용을 그리면 객체지향으로 판단된다.

암기 포인트

정보공학=E-R, 구조적=DFD, 객체지향=유스케이스이다.

NoteBase

문 4. 다음 코드에 대해 조건 커버리지를 100% 만족하는 테스트 케이스 집합은? (단, 단축연산(short-circuit evaluation)은 수행하지 않는다)

```
if(a > 5 && b == 0)
    printf("첫 번째 조건 충족\n");
if(a == 3 || b < 0)
    printf("두 번째 조건 충족\n");
```

① {(a = 2, b = 0), (a = 2, b = 1)}

② {(a = 3, b = 0), (a = 4, b = −2)}

③ {(a = 6, b = 0), (a = 2, b = 1)}

④ {(a = 6, b = 0), (a = 3, b = −1)}

정답 4번

출제 포인트

이 문제는 복합 조건문에서 단축연산을 하지 않을 때 조건 커버리지 100%의 의미를 실제 테스트 입력 집합에 적용할 수 있는지를 묻는 문제이다.

사례 정리

코드의 원자 조건은 `a > 5`, `b == 0`, `a == 3`, `b < 0`의 네 개이다.
단축연산을 하지 않으므로 `&&`, `||`의 좌우 조건이 모두 평가된다.
100% 조건 커버리지는 네 원자 조건 각각이 전체 테스트 집합에서 참과 거짓을 모두 한 번 이상 가져야 성립한다.

핵심 개념

조건 커버리지는 복합식 전체의 참·거짓이 아니라 그 안의 각 원자 조건이 참과 거짓을 모두 경험했는지를 보는 기준이다.
이 문제의 적용 요건은 `a > 5`, `b == 0`, `a == 3`, `b < 0` 각각에 대해 T와 F가 모두 나타나는 것이다.

적용 기준

이 사례에서는 각 선택지의 두 입력쌍을 코드에 대입해 네 원자 조건의 T/F를 모두 적어 보면 된다.
복합 조건문이 한 번 참이고 한 번 거짓인지만 보는 것은 결정 커버리지 판단이고, 조건 커버리지는 원자 조건별 T/F 이력을 본다.

NoteBase

선지별 해설

1번: X. `(a=2, b=0)`, `(a=2, b=1)`에서는 `a > 5`가 두 번 모두 거짓이고 `a == 3`도 두 번 모두 거짓이며 `b < 0`도 두 번 모두 거짓이다. 이 집합을 100% 조건 커버리지로 보는 판단이 틀렸고, 올바르게는 여러 원자 조건이 참을 한 번도 갖지 못한 집합이다.
2번: X. `(a=3, b=0)`, `(a=4, b=-2)`에서는 `a > 5`가 두 번 모두 거짓이다. `b == 0`, `a == 3`, `b < 0`은 일부 T/F가 바뀌지만 `a > 5`가 참을 만들지 못하므로 100% 조건 커버리지가 아니다. 올바르게는 `a > 5`의 참 사례가 추가되어야 한다.
3번: X. `(a=6, b=0)`, `(a=2, b=1)`에서는 `a > 5`, `b == 0`은 T/F를 모두 갖지만 `a == 3`은 두 번 모두 거짓이고 `b < 0`도 두 번 모두 거짓이다. 이 집합이 충분하다는 판단이 틀렸고, 올바르게는 두 번째 `if`의 각 원자 조건도 참 사례를 가져야 한다.
4번: O. `(a=6, b=0)`에서는 `a > 5`와 `b == 0`이 참이고 `a == 3`, `b < 0`은 거짓이다. `(a=3, b=-1)`에서는 `a > 5`, `b == 0`이 거짓이고 `a == 3`, `b < 0`이 참이다. 따라서 네 원자 조건 모두가 전체 집합에서 참과 거짓을 한 번씩 가져 100% 조건 커버리지를 만족한다.

오답 포인트

복합식 전체가 한 번 참이고 한 번 거짓이면 충분하다고 보면 결정 커버리지와 조건 커버리지를 혼동하게 된다. 또 단축연산을 하지 않는다는 문구를 놓치면 `b == 0`, `b < 0`의 평가 자체를 빠뜨려 판단이 흔들린다.

다시 나오면 이렇게 판단

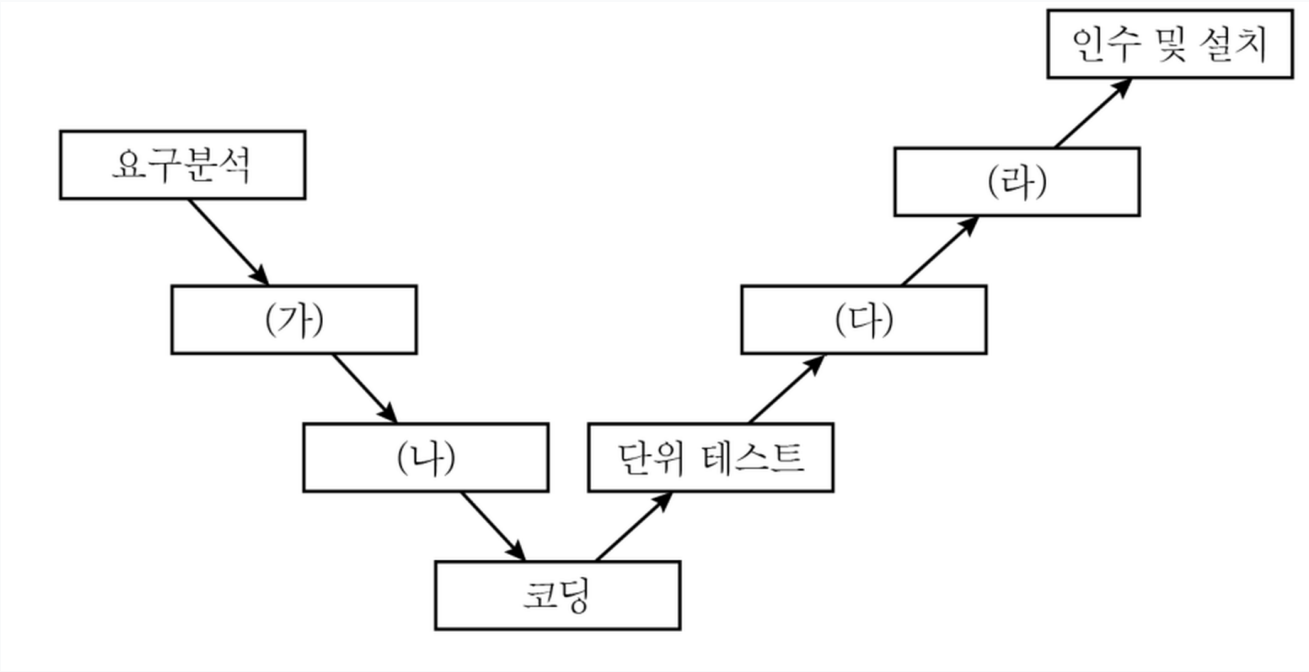
먼저 복합 조건을 원자 조건으로 분해한다. 다음으로 각 테스트 데이터마다 각 원자 조건의 T/F를 적는다.
마지막으로 원자 조건마다 T 한 번, F 한 번이 모두 있는지만 확인하면 된다.

암기 포인트

조건 커버리지는 원자 조건마다 T 한 번, F 한 번이다.

NoteBase

문 5. V 모델의 단계에서 (가) ~ (라)에 들어갈 내용을 바르게 연결한 것은?



A. 모듈 설계B. 시스템 테스트C. 아키텍처 설계D. 통합 테스트

① (가): A, (나): B, (다): C, (라): D

② (가): A, (나): C, (다): B, (라): D

③ (가): C, (나): A, (다): B, (라): D

④ (가): C, (나): A, (다): D, (라): B

정답 4번

출제 포인트

V 모델에서 왼쪽의 분석·설계 단계와 오른쪽의 대응 테스트 단계를 정확한 순서로 연결하는 문제이다.

NoteBase

선지별 해설

① X. 틀린 표현은 '(가)=A, (나)=B, (다)=C'이다. 올바르게는 '(가)=C, (나)=A, (다)=D'이다. 요구분석 다음은 모듈 설계가 아니라 아키텍처 설계이고, 그다음이 모듈 설계이다. 또한 단위 테스트 다음 단계는 아키텍처 설계와 대응하는 통합 테스트이다.
② X. 틀린 표현은 '(가)=A, (다)=B'이다. 올바르게는 '(가)=C, (다)=D'이다. 왼쪽에서 요구분석 다음은 아키텍처 설계가 와야 하고, 오른쪽에서 단위 테스트 다음은 시스템 테스트가 아니라 통합 테스트가 와야 한다.
③ X. 왼쪽의 '(가)=C, (나)=A'는 맞지만 오른쪽 테스트 단계가 뒤바뀌었다. 틀린 표현은 '(다)=B, (라)=D'이다. 올바르게는 '(다)=D, (라)=B'이다. 단위 테스트 다음은 통합 테스트이고, 그다음이 시스템 테스트이다.
④ O. '(가)=C, (나)=A, (다)=D, (라)=B'는 맞다. 요구분석 다음은 아키텍처 설계, 그다음은 모듈 설계이며, 코딩 이후에는 단위 테스트 다음으로 통합 테스트, 시스템 테스트, 인수 및 설치가 이어진다.

오답 포인트

아키텍처 설계와 모듈 설계의 순서를 뒤집거나 통합 테스트와 시스템 테스트를 바꾸면 오답이 된다. 특히 ③은 왼쪽 설계 단계는 맞지만 오른쪽 테스트 단계만 바꿔 놓은 함정이다.

다시 나오면 이렇게 판단

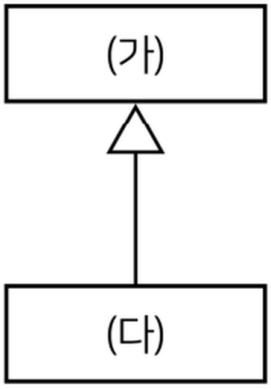
코딩을 기준점으로 왼쪽은 요구분석→아키텍처 설계→모듈 설계로 내려가고, 오른쪽은 단위 테스트→통합 테스트→시스템 테스트→인수 및 설치로 올라간다고 놓으면 빈칸을 바로 대응시킬 수 있다.

암기 포인트

V 모델은 설계는 큰 구조에서 작은 구조로 내려가고, 테스트는 작은 단위에서 큰 시스템으로 올라간다.

NoteBase

문 6. 다음은 클래스 다이어그램을 Java 코드로 구현한 것이다. (가) ~ (다)에 들어갈 내용을 바르게 연결한 것은?




```
class Lecture extends (나) {
    public Lecture() {}
}
class Study {
    public Study() {}
}
```

① (가): Study, (나): Study, (다): Lecture

② (가): Study, (나): Lecture, (다): Lecture

③ (가): Lecture, (나): Study, (다): Study

④ (가): Lecture, (나): Lecture, (다): Study

정답 1번

출제 포인트

이 문제는 UML 클래스 다이어그램의 일반화(상속) 화살표 방향과 Java의 extends 관계를 연결하는 문제이다.

선지별 해설

① O: (가)=Study, (나)=Study, (다)=Lecture의 연결이 맞다. 다이어그램에서 부모는 위의 (가)이고 자식은 아래의 (다)이므로 코드의 자식 클래스 Lecture는 (다)에, 부모 클래스 Study는 (가)와 extends 뒤의 (나)에 들어간다.

② X: (나)를 Lecture로 둔 부분이 틀리다. extends 뒤에는 부모 클래스가 와야 하므로 'class Lecture extends Lecture'가 아니라 'class Lecture extends Study'가 되어야 한다.

③ X: (가)를 Lecture로 둔 부분이 틀리다. 화살표가 가리키는 위쪽 클래스는 부모이므로 '(가)=Lecture'가 아니라 '(가)=Study'가 되어야 한다. 또한 아래 자식 클래스인 (다)도 Study가 아니라 Lecture가 되어야 한다.

④ X: (가)를 Lecture로 둔 부분과 (다)를 Study로 둔 부분이 모두 틀리다. 부모와 자식의 위치를 뒤바꾼 오류이므로 '(가)=Study, (다)=Lecture'로 고쳐야 하며, extends 뒤의 (나)도 Study가 맞다.

오답 포인트

빈 삼각형 화살표의 방향을 연관관계 화살표처럼 읽으면 부모와 자식을 거꾸로 판단하게 된다. 또한 Java 코드에서 extends 뒤에 오는 이름이 자식이라고 착각하면 ②처럼 오답이 된다.

NoteBase

다시 나오면 이렇게 판단

UML 일반화 화살표에서는 삼각형이 향하는 쪽을 먼저 부모 클래스로 잡고, Java 코드에서는 extends 앞이 자식 클래스, extends 뒤가 부모 클래스라고 대응하면 된다.

암기 포인트

UML 삼각형 끝은 부모, Java extends 뒤는 부모이다.

NoteBase

문 7. 다음 파이썬 코드에서 문장 커버리지를 100% 만족하는 (가) ~ (다)에 들어갈 경로를 바르게 연결한 것은? (단, 단축연산은 수행하지 않는다)

```
(1)  a = int(input('종료 숫자? '))
(2)  sum = 0
(3)  i = 1
(4)  while i <= a:
(5)      if i % 2 == 0:
(6)          sum += i
(7)      i += 1
(8)  print('{}까지 짝수합은 {}'.format(a, sum))
```

(1) – (2) – (3) – (4) – (5) – (가) – (나) – (다) – (8)

① (가): (6), (나): (7), (다): (5)

② (가): (6), (나): (7), (다): (4)

③ (가): (7), (나): (4), (다): (6)

④ (가): (7), (나): (4), (다): (7)

정답 2번

출제 포인트

이 문제는 while 반복문과 if 분기의 제어 흐름에서 문장 커버리지 100%를 만들려면 참 분기 문장과 반복 조건 재검사 위치를 정확히 연결할 수 있는지를 묻는다.

올바른 순서

1. 5 → 6 — if 조건이 참이면 짝수 누적문이 실행된다.
2. 6 → 7 — 누적 후 반복 변수 증가문으로 이동한다.
3. 7 → 4 — 반복 본문이 끝나면 while 조건을 다시 검사한다.

왜 이 순서: 6은 5의 참 분기에서만 실행되고, 7은 반복 본문의 공통 후속 문장이며, 다음 반복 여부와 종료 여부는 4에서 다시 결정된다.

NoteBase

선지별 해설

① X: (가)=6, (나)=7까지는 맞지만 (다)=5로 두면 7 다음에 while 조건 4를 다시 거치지 않는 잘못된 흐름이 된다. 올바르게는 (다)는 4이다.
② O: 5의 참 분기에서는 6이 실행되고, 이어서 7에서 i를 증가시킨 뒤 4로 돌아가 while 조건을 다시 검사한다. 예를 들어 a=2이면 전체 실행 중 6이 한 번 이상 실행되어 모든 문장을 방문할 수 있다.
③ X: (가)=7로 두면 5의 참 분기에서만 실행되는 6이 빠져 문장 커버리지가 100%가 되지 않는다. 또한 4 다음에 바로 6으로 가는 흐름도 성립하지 않는다. 올바르게는 (가)=6, (나)=7, (다)=4이다.
④ X: (가)=7이면 6이 빠져 문장 커버리지가 깨지고, (다)=7이면 7 뒤에 다시 7로 가는 흐름이 되어 반복 조건 재검사 4가 사라진다. 올바르게는 (가)=6, (나)=7, (다)=4이다.

오답 포인트

문장 커버리지는 조건식의 참 분기에서만 실행되는 6을 반드시 한 번 포함해야 한다는 점과, 반복문 본문이 끝난 뒤에는 5가 아니라 4에서 while 조건을 다시 검사한다는 점을 놓치기 쉽다.

다시 나오면 이렇게 판단

커버리지 문제는 분기문에서만 실행되는 문장이 무엇인지 먼저 찾고, 그 문장 실행 뒤 반복문의 공통 후속 문장과 조건 재검사 위치가 어디인지 순서대로 확인하면 된다.

암기 포인트

if 참 분기 실행은 6, 반복 본문 종료는 7, 다음 반복 판단은 4 순서이다.

NoteBase

문 8. 객체 지향 설계 원칙에 대한 설명으로 옳지 않은 것은?

① 단일 책임(single-responsibility)의 원칙: 클래스를 변경해야 하는 이유는 오직 하나여야 한다.

② 개방 폐쇄(open-closed)의 원칙: 모듈은 수정에는 열려 있어야 하고 확장에는 닫혀 있어야 한다.

③ 리스코프 대체(Liskov substitution)의 원칙: 상위 클래스의 객체는 자신의 하위 클래스로 대체할 수 있어야 한다.

④ 의존 관계 역전(dependency inversion)의 원칙: 클라이언트는 구체적인 클래스가 아닌 추상 클래스에 의존해야 한다.

정답 2번

출제 포인트

이 문제는 객체 지향 설계의 SOLID 원칙 중 SRP, OCP, LSP, DIP의 정확한 정의를 구별하는 문제이다.

핵심 개념

SRP는 클래스가 변경되는 이유를 하나만 갖도록 책임을 분리하는 원칙이다. OCP는 기존 코드를 자주 고치지 않고 기능을 확장할 수 있도록 수정에는 닫히고 확장에는 열려 있어야 한다는 원칙이다. LSP는 하위 타입이 상위 타입을 대체해도 프로그램의 올바름이 깨지지 않아야 한다는 원칙이다. DIP는 구체 클래스보다 추상화에 의존하도록 하여 결합도를 낮추는 원칙이다.

선지별 해설

① O: 단일 책임 원칙은 클래스를 변경해야 하는 이유가 하나여야 한다는 뜻이므로 옳다.
② X: 틀린 부분은 "모듈은 수정에는 열려 있고 확장에는 닫혀 있어야 한다"는 표현이다. 개방-폐쇄 원칙은 실제로는 "수정에는 닫혀 있고 확장에는 열려 있어야 한다"가 맞다.
③ O: 리스코프 대체 원칙은 상위 클래스 타입 자리에 하위 클래스 객체를 넣어도 동일한 규약이 유지되어야 한다는 뜻이므로 옳다.
④ O: 의존 관계 역전 원칙은 클라이언트가 구체 구현이 아니라 인터페이스나 추상 클래스 같은 추상화에 의존해야 한다는 뜻이므로 옳다.

오답 포인트

개방-폐쇄 원칙의 "개방"과 "폐쇄"를 일상어 의미로 읽으면 방향을 거꾸로 기억하기 쉽다. 이 문항은 ②에서 수정과 확장의 대상을 뒤집어 적어 둔 점이 함정이다.

다시 나오면 이렇게 판단

SOLID 문항은 각 약어가 답하는 질문으로 판별하면 된다. SRP는 "왜 바꾸는가", OCP는 "어디까지 고치지 않고 늘리는가", LSP는 "대체해도 되는가", DIP는 "무엇에 의존하는가"를 보면 판단할 수 있다.

NoteBase

암기 포인트

OCP는 확장에는 열리고 수정에는 닫힌다. DIP는 구현이 아니라 추상화에 의존한다.

NoteBase

문 9. (가)에 해당하는 시암 치댐버(Shyam R. Chidamber)와 크리스 캐머러(Chris F. Kemerer)의 객체지향 설계 척도는?

(가) = $C_{m1} + C_{m2} + \cdots + C_{mn}$ 단, $C_{mi}(i = 1, 2, \ldots, n)$는 클래스 C에 있는 메소드의 복잡도이다.

① NOC(Number Of Children)

② WMC(Weighted Methods per Class)

③ CBO(Coupling Between Object classes)

④ LCOM(Lack of Cohesion in Methods)

정답 2번

출제 포인트

이 문제는 CK 객체지향 설계 척도에서 메서드 복잡도의 합으로 정의되는 WMC와 다른 척도들의 정의를 구별하는 문제이다.

핵심 개념

WMC는 한 클래스에 속한 각 메서드의 복잡도를 모두 더한 값이다. 키워드는 '메서드 복잡도 합'이다.
NOC는 한 클래스의 직접 자식 클래스 수이다. 키워드는 '상속받는 하위 클래스 수'이다.
CBO는 한 클래스가 다른 클래스와 결합된 정도이다. 키워드는 '클래스 간 결합도'이다.
LCOM은 클래스 내 메서드들이 속성을 얼마나 공유하지 못하는지를 나타내는 응집도 결핍 척도이다. 키워드는 '메서드 간 응집 부족'이다.

NoteBase

선지별 해설

① X
NOC는 Number Of Children으로, 어떤 클래스의 직계 하위 클래스 개수를 세는 척도이다. 제시된 식처럼 메서드 복잡도를 더하는 척도가 아니다. 틀린 표현은 'NOC'이다. 올바르게는 '직계 자식 클래스 수'를 나타낸다.

② O
WMC는 Weighted Methods per Class로, 클래스에 포함된 각 메서드의 복잡도 $C_{m1}, C_{m2}, \ldots, C_{mn}$를 합한 값이다. 제시된 식과 정확히 일치한다.

③ X
CBO는 Coupling Between Object classes로, 한 클래스가 다른 클래스와 얼마나 연결되어 있는지를 나타낸다. 메서드 복잡도 합과는 무관하다. 틀린 표현은 'CBO'이다. 올바르게는 '클래스 간 결합도'를 나타낸다.

④ X
LCOM은 Lack of Cohesion in Methods로, 클래스 내부 메서드들이 공통 속성을 얼마나 공유하지 못하는지를 본다. 제시된 식은 응집 부족이 아니라 메서드 복잡도 가중합이다. 틀린 표현은 'LCOM'이다. 올바르게는 '메서드 간 응집 부족 정도'를 나타낸다.

오답 포인트

모든 보기가 CK 객체지향 설계 척도의 약어라서 약어만 보고 고르면 헷갈리기 쉽다. 이 문항의 함정은 '클래스 관련 척도'를 한 덩어리로 기억하는 데 있다. 식에 메서드별 복잡도의 합이 보이면 상속 수, 결합도, 응집도 척도를 지우고 WMC부터 떠올려야 한다.

다시 나오면 이렇게 판단

식이 메서드별 복잡도를 더하는 구조이면 WMC로 판단한다. 하위 클래스 수를 세면 NOC, 다른 클래스와의 연결 정도를 보면 CBO, 클래스 내부 메서드의 응집 부족을 보면 LCOM으로 구분한다.

암기 포인트

WMC=메서드 복잡도 합, NOC=자식 수, CBO=클래스 간 결합도, LCOM=메서드 응집 부족이다.

NoteBase

문 10. 시험 운영 중인 보안관제 시스템이 다음 표와 같이 운영되면서 정상 가동 중에 고장이 발생하였을 때, 시스템의 가용성(availability)은?

	가동1	고장1	가동2	고장2	가동3	고장3	가동4
시간(h)	12	4	9	2	10	3	5

① 0.2

② 0.25

③ 0.8

④ 0.9

정답 3번

출제 포인트

이 문제는 가용성 공식을 이용해 총 운영시간 대비 정상 가동시간의 비율을 구하는 문제이다.

조건 정리

주어진 값은 가동시간 12시간, 9시간, 10시간, 5시간과 고장시간 4시간, 2시간, 3시간이다.
구해야 할 값은 시스템의 가용성이다.
정상 가동 중 고장이 여러 번 발생했으므로 전체 가동시간과 전체 고장시간을 합산해 계산한다.

사용할 공식

가용성은 총 가동시간총 가동시간+총 고장시간이다.
이 공식을 쓰는 이유는 문제에서 일정 기간 동안 반복된 가동 구간과 고장 구간을 모두 제시했기 때문이다.

계산 과정

총 가동시간은 $12 + 9 + 10 + 5 = 36$시간이다.
총 고장시간은 $4 + 2 + 3 = 9$시간이다.
총 운영시간은 $36 + 9 = 45$시간이다.
따라서 가용성은 $\frac{36}{45} = 0.8$이다.

NoteBase

선지별 해설

① X. 0.2는 $\frac{9}{45}$로 계산한 비가동률에 해당한다. 이 문항에서 묻는 값은 가용성이므로 0.2가 아니라 0.8이 맞다.
② X. 0.25는 $\frac{9}{36}$처럼 고장시간을 가동시간으로 나눈 값이다. 가용성은 고장시간을 분자로 두지 않으므로 0.25가 아니라 $\frac{36}{45} = 0.8$로 고쳐야 한다.
③ O. 총 가동시간 36시간을 총 운영시간 45시간으로 나누면 0.8이므로 정답이다.
④ X. 0.9는 제시된 가동시간과 고장시간을 모두 반영한 계산 결과가 아니다. 주어진 표 전체를 합산하면 가용성은 0.9가 아니라 0.8이다.

오답 포인트

가용성과 비가동률을 뒤집어 계산하거나, 총 운영시간 대신 가동시간만 분모로 두는 실수가 자주 나온다. 이 문항의 0.2와 0.25가 그 함정을 그대로 반영한 값이다.

다시 나오면 이렇게 판단

가용성을 묻는 계산형 문제는 먼저 모든 가동 구간을 합산하고, 모든 고장 구간을 합산한 뒤, 총 가동시간총 가동시간+총 고장시간 순서로 판단한다.

암기 포인트

가용성은 가동시간 비율이고 비가동률은 고장시간 비율이므로 분자 자리를 바꾸지 않는 것이 핵심이다.

NoteBase

문 11. 다음과 같이 Java 코드를 변경하였을 때 적용된 기법은?

```
void select(int num)
{
  System.out.println("1. 한국어, 2. 영어");
  if(num == 1)
     System.out.println("한국어");
  else
     System.out.println("영어");
}
```



```
void select(int num)
{
  System.out.println("1. 한국어, 2. 영어");
  lang(num);
}
void lang(int i)
{
  if(i == 1)
     System.out.println("한국어");
  else
     System.out.println("영어");
}
```

① 리팩토링(refactoring)

② 재사용(reuse)

③ 순공학(forward engineering)

④ 역공학(reverse engineering)

정답 1번

출제 포인트

이 문제는 기존 Java 코드의 외부 동작은 유지한 채 내부 구조만 바꾼 사례를 보고 해당 소프트웨어 공학 기법을 판별할 수 있는지를 묻는다.

사례 정리

기존 select 메서드 안에 있던 조건 분기와 출력 로직을 별도 메서드 lang(int i)로 분리하였다.
select는 메뉴 출력 후 lang(num)을 호출하도록 바뀌었고, num 값에 따라 한국어 또는 영어를 출력하는 기능 자체는 그대로이다.
판단에 중요한 조건은 기능 결과의 변화 없이 코드 구조만 더 분리되고 읽기 쉬워졌다는 점이다.

NoteBase

핵심 개념

리팩토링은 소프트웨어의 외부 동작은 유지하면서 내부 구조를 개선하는 기법이다.
이 사례의 핵심은 메서드 추출을 통해 책임을 분리하고 가독성과 유지보수성을 높였다는 점이다.

적용 기준

이 사례에서는 기능 추가나 산출물 복원이 아니라 메서드 추출 여부를 본다.
if-else 로직을 select에서 lang으로 옮겼을 뿐 출력 결과는 같으므로 리팩토링에 해당한다.

선지별 해설

① O: 기존 메서드 내부 로직을 별도 메서드로 추출하여 구조를 개선했고, 실행 결과는 그대로 유지되므로 리팩토링이다.
② X: 재사용은 이미 만들어진 코드나 컴포넌트를 여러 곳에서 다시 활용하는 것이다. 이 사례의 핵심은 '적용된 기법이 재사용'이라는 점이 아니라 '구조 개선을 위한 코드 재구성'이라는 점이다. 올바르게는 기존 동작을 유지한 리팩토링이다.
③ X: 순공학은 요구사항이나 설계 모델에서 실제 프로그램을 구현해 가는 과정이다. 이 사례는 새로운 시스템을 설계에서 구현한 것이 아니라 기존 코드를 더 나은 구조로 고친 것이다. 올바르게는 기존 코드의 내부 구조 개선인 리팩토링이다.
④ X: 역공학은 기존 코드나 시스템으로부터 설계, 명세, 모델을 추출해 내는 과정이다. 이 사례는 코드 분석 결과를 상위 산출물로 복원한 것이 아니라 코드를 직접 수정한 것이다. 올바르게는 코드 수정 방식의 리팩토링이다.

오답 포인트

메서드를 따로 뽑아 냈다는 겉모습 때문에 ② 재사용과 혼동하기 쉽다. 그러나 이 문제의 판단 기준은 여러 곳에서 다시 쓰였는지가 아니라 기능은 유지한 채 내부 구조를 개선했는지이다.

다시 나오면 이렇게 판단

코드 변경 문제는 먼저 실행 결과가 달라졌는지 본다. 결과는 같고 메서드 분리, 이름 개선, 중복 제거처럼 내부 구조만 좋아졌다면 리팩토링으로 판단한다.

암기 포인트

리팩토링은 기능 유지, 구조 개선이다. 메서드 추출은 대표적인 리팩토링 기법이다.

NoteBase

문 12. 다음과 같은 조건으로 소프트웨어 개발 프로젝트를 수행할 때, 계약 기간 내에 프로젝트를 성공적으로 수행하기 위해 필요한 최소 기간은?

프로젝트 명: 재고관리 시스템 개발
추정 소스 코드 라인 수: 65,000 LOC
개발자 1인당 월평균 생산성(LOC/MM): 초급자 1,000, 중급자 1,500
현재 투입 가능 개발자 수(명): 초급자 4, 중급자 2

① 7개월
② 8개월
③ 9개월
④ 10개월

정답 4번

출제 포인트

이 문제는 개발자별 월 생산성을 합산해 전체 개발 기간을 계산하고, 최소 계약 기간이므로 올림 처리하는지를 묻는 문제이다.

조건 정리

주어진 값: 추정 소스 코드 라인 수는 65,000 LOC이다.
주어진 값: 초급자 4명의 생산성은 1인당 월 1,000 LOC이다.
주어진 값: 중급자 2명의 생산성은 1인당 월 1,500 LOC이다.
구해야 할 값: 계약 기간 내 성공적으로 완료하기 위한 최소 개월 수이다.
주의할 조건: 전체 기간은 팀의 월 총생산성으로 계산하며, 소수점 개월은 완료 보장을 위해 올림 처리한다.

사용할 공식

사용할 식은 전체 기간 = 총 작업량 / 팀의 월 총생산성이다.
이 식을 쓰는 이유: 문제에서 전체 LOC와 개발자별 월 생산성이 주어졌으므로, 팀이 한 달에 처리할 수 있는 총 LOC를 구한 뒤 전체 작업량을 나누면 필요한 개월 수를 계산할 수 있다.

NoteBase

계산 과정

1. 초급자 4명의 월 생산성은 4 × 1,000 = 4,000 LOC이다.
2. 중급자 2명의 월 생산성은 2 × 1,500 = 3,000 LOC이다.
3. 팀의 월 총생산성은 4,000 + 3,000 = 7,000 LOC이다.
4. 필요한 기간은 65,000 ÷ 7,000 ≈ 9.29개월이다.
5. 9개월이면 7,000 × 9 = 63,000 LOC로 부족하고, 10개월이면 7,000 × 10 = 70,000 LOC로 완료 가능하다.

선지별 해설

① X: 7개월이면 7,000 × 7 = 49,000 LOC만 개발할 수 있어 65,000 LOC에 한참 못 미친다. 틀린 표현은 "7개월이면 완료 가능"이며, 올바르게는 "7개월로는 완료할 수 없다"이다.
② X: 8개월이면 7,000 × 8 = 56,000 LOC만 개발할 수 있어 여전히 부족하다. 틀린 표현은 "8개월이 최소 기간"이며, 올바르게는 "8개월로는 완료할 수 없다"이다.
③ X: 9개월이면 7,000 × 9 = 63,000 LOC로 2,000 LOC가 부족하다. 계산값 9.29를 버림하면 나오는 오답이다. 틀린 표현은 "9개월이면 성공적으로 수행 가능"이며, 올바르게는 "9개월은 부족하고 10개월이 필요하다"이다.
④ O: 10개월이면 7,000 × 10 = 70,000 LOC를 개발할 수 있어 65,000 LOC를 충족한다. 계산 결과가 약 9.29개월이므로 최소 보장 기간은 10개월이다.

오답 포인트

계산값이 9.29개월이라고 해서 9개월로 처리하면 안 된다는 점이 함정이다. 이 문제는 평균 기간이 아니라 계약 기간 내 완료 가능한 최소 정수를 묻기 때문에 반드시 올림해야 한다.

다시 나오면 이렇게 판단

개발 기간 계산 문제는 전체 작업량을 팀의 월 총생산성으로 나눈 뒤, 완료 보장 여부를 확인해 정수 개월로 올림하는 순서로 판단한다.

암기 포인트

기간 문제는 총작업량 ÷ 월총생산성으로 구하고, 완료 조건이면 소수점 개월을 올림한다.

NoteBase

문 13. 다음 설명에 해당하는 소프트웨어 형상 관리 활동은?

◯ 형상 관리 계획서에 따라 변경 관리 프로세스가 진행되었는지 평가한다.
◯ 형상 항목의 변경이 요구사항에 맞게 이루어졌는지 확인한다.

① 형상 통제(configuration control)
② 형상 식별(configuration identification)
③ 형상 감사 및 검토(configuration audits and reviews)
④ 형상 상태 보고(configuration status accounting)

정답 3번

출제 포인트

이 문제는 소프트웨어 형상 관리 활동 중 형상 통제, 형상 식별, 형상 감사 및 검토, 형상 상태 보고의 역할 차이를 구별하는지를 묻는다.

핵심 개념

형상 통제는 형상 항목의 변경 요청을 심사하고 승인·반영하는 활동이다. 형상 식별은 형상 항목과 베이스라인을 정의하고 이름과 버전을 부여하는 활동이다. 형상 감사 및 검토는 변경이 절차대로 수행되었는지와 결과물이 요구사항에 맞는지를 검증하는 활동이다. 형상 상태 보고는 형상 항목과 변경 요청의 현재 상태를 기록하고 보고하는 활동이다.

선지별 해설

1번 X: 형상 통제는 변경 요청을 접수하고 영향 분석, 승인, 반영 여부를 관리하는 활동이다. 제시문은 변경을 승인하는 단계가 아니라 변경 관리 프로세스가 계획서대로 진행되었는지와 변경 결과가 요구사항에 맞는지를 평가하는 단계이다. 틀린 표현은 활동을 형상 통제로 본 점이다. 올바르게는 변경 승인·통제가 아니라 변경 결과와 절차의 적합성을 확인하는 형상 감사 및 검토이다.
2번 X: 형상 식별은 어떤 산출물을 형상 항목으로 둘지 정하고 식별자, 버전, 베이스라인을 정의하는 활동이다. 제시문에는 항목을 식별하거나 명명하는 내용이 없다. 틀린 표현은 변경 확인 활동을 형상 식별로 본 점이다. 올바르게는 식별이 아니라 변경 절차 준수 여부와 요구사항 적합성을 점검하는 형상 감사 및 검토이다.
3번 O: 형상 감사 및 검토는 형상 관리 계획서에 따라 변경 관리 프로세스가 수행되었는지 확인하고, 변경된 형상 항목이 요구사항에 맞게 수정되었는지를 검증하는 활동이다. 제시문의 두 문장이 바로 기능적·물리적 감사의 성격을 설명한다.
4번 X: 형상 상태 보고는 형상 항목의 버전, 변경 요청 처리 현황, 베이스라인 상태를 기록하고 이해관계자에게 보고하는 활동이다. 제시문은 상태를 기록·보고하는 것이 아니라 변경의 적절성과 절차 준수 여부를 평가한다. 틀린 표현은 검증 활동을 상태 보고로 본 점이다. 올바르게는 보고가 아니라 형상 감사 및 검토이다.

NoteBase

오답 포인트

변경이라는 단어만 보고 1번 형상 통제로 가기 쉽다. 그러나 통제는 변경을 승인하고 관리하는 활동이고, 제시문처럼 계획서 준수 여부와 요구사항 충족 여부를 사후적으로 확인하는 활동은 감사 및 검토이다.

다시 나오면 이렇게 판단

형상 관리 문제는 먼저 동사가 무엇인지 본다. 항목을 정하면 식별, 변경을 승인·반영하면 통제, 상태를 기록하면 상태 보고, 절차 준수와 결과 적합성을 확인하면 감사 및 검토이다.

암기 포인트

식별은 정하기, 통제는 바꾸기, 상태 보고는 기록하기, 감사 및 검토는 맞게 했는지 확인하기이다.

NoteBase

문 14. 프로토타입 모델에 대한 설명으로 옳지 않은 것은?

① 프로토타입을 통해 새로운 요구사항을 발견할 수 있다.

② 시스템 개발 최종 종료 단계에서만 오류를 탐지할 수 있다.

③ 사용자가 초기 프로토타입을 최종 완성된 소프트웨어로 오인할 수 있다.

④ 개발자 입장에서 프로토타이핑 과정을 관리·통제하기 어렵다.

정답 2번

출제 포인트

이 문제는 프로토타입 모델의 목적과 장단점을 정확히 구별할 수 있는지를 묻는다.

핵심 개념

프로토타입 모델은 초기 시제품을 빠르게 만들어 사용자와 확인하면서 요구사항을 구체화하는 개발 방식이다. 핵심은 완성품을 한 번에 만드는 것이 아니라 조기 피드백으로 요구사항과 오류를 미리 드러내는 데 있다.

선지별 해설

1번 O: 프로토타입을 사용하면 사용자가 시제품을 보며 빠뜨린 기능이나 추가 요구를 말할 수 있으므로 새로운 요구사항을 발견할 수 있다.
2번 X: 틀린 부분은 "최종 종료 단계에서만"이다. 프로토타입 모델은 개발 초기에 시제품을 통해 요구사항 오류나 설계상의 문제를 조기에 발견하는 데 강점이 있다. 올바르게는 "시스템 개발 초기부터 반복 검토를 통해 오류를 탐지할 수 있다"이다.
3번 O: 초기 프로토타입이 화면이나 기능 형태를 갖추고 있으면 사용자가 이를 최종 완성 소프트웨어로 오인할 수 있다. 이는 프로토타입 모델의 대표적인 단점이다.
4번 O: 프로토타입은 사용자 피드백에 따라 반복 수정이 많이 일어나므로 개발자 입장에서는 범위와 일정을 관리·통제하기 어려울 수 있다.

오답 포인트

프로토타입의 장점을 "최종 결과물을 빨리 만든다"로 오해하면 2번을 놓치기 쉽다. 이 모델의 강점은 최종 단계 검사가 아니라 초기 요구사항 확인과 조기 오류 발견에 있다.

다시 나오면 이렇게 판단

프로토타입 관련 선지는 먼저 초기 시제품, 사용자 피드백, 요구사항 명확화, 반복 수정이라는 키워드로 판단한다. 조기 검증 기능을 부정하거나 완성품과 동일시하는 표현이 나오면 오답 가능성이 크다.

NoteBase

암기 포인트

프로토타입은 먼저 보여 주고 빨리 고친다. 장점은 요구사항 발견과 조기 오류 탐지이고, 단점은 완성품 오인과 관리 통제의 어려움이다.

NoteBase

문 15. SPICE(ISO/IEC 15504) 모델에서 정의한 프로세스 성숙도 수준에 속하지 않는 것은?

① 수행(performed) 단계

② 관리(managed) 단계

③ 정의(defined) 단계

④ 최적화(optimizing) 단계

정답 3번

출제 포인트

이 문제는 SPICE(ISO/IEC 15504)가 정의한 프로세스 수행능력 수준의 명칭과 CMM/CMMI의 성숙도 수준 명칭을 구별하는 문제이다.

핵심 개념

SPICE의 프로세스 수준은 0 불완전, 1 수행(Performed), 2 관리(Managed), 3 확립(Established), 4 예측(Predictable), 5 최적화(Optimizing)로 구성된다. CMM/CMMI의 성숙도 수준에는 관리(Managed), 정의(Defined), 정량적 관리(Quantitatively Managed), 최적화(Optimizing) 같은 명칭이 쓰인다. 따라서 SPICE에서 3수준은 defined가 아니라 established라는 점이 핵심이다.

선지별 해설

① O: 수행(performed)은 SPICE 프로세스 수행능력 1수준의 명칭이다. 프로세스가 목적을 달성하며 수행되는 단계에 해당한다.
② O: 관리(managed)는 SPICE 2수준의 명칭이다. 계획과 모니터링 아래 프로세스가 관리되는 단계이다.
③ X: 틀린 부분은 "정의(defined) 단계"라는 표현이다. SPICE의 3수준 명칭은 정의가 아니라 확립(established)이다. "defined"는 CMM/CMMI의 성숙도 수준 3단계에서 쓰이는 표현이다.
④ O: 최적화(optimizing)는 SPICE 5수준의 명칭이다. 지속적 개선이 이루어지는 단계이다.

오답 포인트

SPICE와 CMM/CMMI는 둘 다 프로세스 성숙을 다루므로 단계 이름이 비슷하게 느껴진다. 특히 ③의 "defined"는 매우 익숙한 용어라 SPICE 3수준으로 착각하기 쉽지만, SPICE에서는 같은 자리에 "established"를 사용한다.

다시 나오면 이렇게 판단

프로세스 평가 모델 문제가 나오면 먼저 어떤 모델인지 확인한 뒤 단계 명칭을 대조한다. SPICE는 performed-managed-established-predictable-optimizing의 흐름으로, CMM/CMMI는 managed-defined-quantitatively managed-optimizing의 흐름으로 구분하면 된다.

NoteBase

암기 포인트

SPICE 3수준은 defined가 아니라 established이다.

NoteBase

문 16. 다음 Java 코드의 실행 결과는?

```
class Car {
    public void type() {
        System.out.println("Car");
    }
}

class Truck extends Car {
    public void type() {
        System.out.println("Truck");
    }
}

class Campingcar extends Truck {
    public void type() {
        super.type();
        System.out.println("Camping car");
    }
}

public class Main {
    public static void main(String[] args) {
        Car c = new Campingcar();
        c.type();
    }
}
```

① Car

② Camping car

③ Car
Truck

④ Truck
Camping car

정답 4번

출제 포인트

이 문제는 오버라이딩된 메서드의 동적 바인딩과 `super` 호출이 어느 부모 구현을 실행하는지를 묻는 문제이다.

핵심 개념

오버라이딩은 자식 클래스가 부모의 메서드를 같은 시그니처로 재정의하는 것이다. 동적 바인딩은 참조변수 타입이 아니라 실제 생성된 객체 타입을 기준으로 실행 메서드가 결정되는 것이다. `super.type()`은 조상 전체를 순서대로 호출하는 문법이 아니라 바로 부모 클래스인 `Truck`의 `type()` 구현을 직접 호출하는 것이다.

NoteBase

선지별 해설

① X: `c`의 참조형은 `Car`이지만 실제 객체는 `Campingcar`이므로 `c.type()`은 동적 바인딩에 의해 `Campingcar.type()`이 실행된다. 틀린 부분은 출력이 `Car` 하나만 나온다는 점이다. 올바르게는 `Truck`이 먼저 출력되고 그다음 `Camping car`가 출력된다.
② X: `Campingcar.type()` 안에는 `System.out.println("Camping car");`만 있는 것이 아니라 그보다 먼저 `super.type();`이 있다. 틀린 부분은 `super.type()` 호출을 누락한 점이다. 올바르게는 먼저 `Truck`이 출력된 뒤 `Camping car`가 출력된다.
③ X: `Campingcar`에서의 `super.type()`은 바로 부모인 `Truck.type()`을 호출한다. `Truck.type()` 안에는 다시 `super.type()` 호출이 없으므로 `Car.type()`은 실행되지 않는다. 틀린 부분은 `Car`가 먼저 출력된다고 본 점이다. 올바르게는 `Car`가 아니라 `Truck`이 먼저 출력된다.
④ O: `c.type()`은 실제 객체인 `Campingcar`의 `type()`을 실행하고, 그 안의 `super.type()`이 `Truck.type()`을 호출하여 `Truck`을 출력한다. 이어서 `Campingcar.type()`의 다음 줄이 실행되어 `Camping car`가 출력된다.

오답 포인트

참조변수 선언형이 `Car`라는 이유로 `Car.type()`이 실행된다고 오해하기 쉽다. 이 문항의 함정은 메서드 호출은 실제 객체 타입으로 결정된다는 점과 `super`가 최상위 부모까지 자동으로 올라가는 호출이 아니라 직계 부모 구현만 가리킨다는 점이다.

다시 나오면 이렇게 판단

상속 코드의 실행 결과를 묻는 문제는 먼저 실제 생성 객체가 무엇인지 확인하고, 그 객체에서 오버라이딩된 메서드가 있는지 본다. 그다음 메서드 내부에 `super` 호출이 있으면 직계 부모 구현만 추가로 따라가고, 그 부모 메서드 안에 다시 `super`가 있는지 별도로 확인하면 된다.

암기 포인트

메서드 호출은 실제 객체로 결정되고, `super`는 직계 부모 구현만 호출한다.

NoteBase

문 17. 순환 복잡도(cyclomatic complexity)가 4인 Java 코드는?

①
```
int max(int x, int y) {
   int m = 0;
   if(x > y)
      m = x;
   else
      m = y;
   return m;
}
```

②
```
int bonus(int base) {
   int x = 0;
   if(base > 2000)
      x = 3000;
   else if(base > 1000)
      x = 2000;
   else
      x = 1000;
   return x;
}
```

③
```
int calculate(int x, int y) {
   while(x != y) {
      if(x > y)
         x -= y;
      else
         y -= x;
   }
   return x;
}
```

④
```
int penalty(int speed) {
   int money = 0;
   if(speed > 200) {
      if(speed < 150)
         money = 50;
      else {
         money = 30;
         if(speed <= 100)
            money = 0;
      }
   }
   return money;
}
```

정답 4번

출제 포인트

이 문제는 순환 복잡도를 결정하는 분기문의 개수를 세어 코드별 독립 경로 수를 판단하는 문제이다.

핵심 개념

순환 복잡도는 제어 흐름 그래프에서 가능한 독립 경로의 수이며, 이 문항에서는 결정 지점 수 + 1로 계산한다. if, else if, while는 각각 결정 지점 1개로 세고, else 자체는 추가되지 않는다.

NoteBase

선지별 해설

① X: if가 1개뿐이므로 결정 지점은 1개이다. 따라서 순환 복잡도는 1 + 1 = 2이다. 틀린 표현은 이 코드가 순환 복잡도 4라는 판단이다. 올바르게는 순환 복잡도 2인 코드이다.
② X: if와 else if가 각각 결정 지점이므로 총 2개이다. 따라서 순환 복잡도는 2 + 1 = 3이다. 틀린 표현은 이 코드가 순환 복잡도 4라는 판단이다. 올바르게는 순환 복잡도 3인 코드이다.
③ X: while가 1개, 내부 if가 1개이므로 결정 지점은 총 2개이다. 따라서 순환 복잡도는 2 + 1 = 3이다. 틀린 표현은 이 코드가 순환 복잡도 4라는 판단이다. 올바르게는 순환 복잡도 3인 코드이다.
④ O: 바깥 if(speed > 200), 안쪽 if(speed < 150), else 블록 안의 if(speed <= 100)까지 결정 지점이 3개이다. 따라서 순환 복잡도는 3 + 1 = 4이다.

오답 포인트

else를 별도 분기로 잘못 세거나, 중첩 if를 하나로 뭉뚱그려 세면 오답이 된다. ④에서 if(speed > 200) 블록 안의 if(speed < 150)은 speed가 200보다 큰 동시에 150보다 작을 수 없으므로 실제로는 절대 실행되지 않는 도달 불가(dead code) 분기이다. 그러나 순환 복잡도는 실행 시 조건이 참이 될 수 있는지를 따지지 않고, 소스코드에 분기문(결정 지점)이 존재하면 그 자체로 1개로 센다. 따라서 도달 불가한 분기라도 if문이 적혀 있으면 결정 지점에 포함되어, ④의 결정 지점은 if 3개로 순환 복잡도 4가 된다. 조건식의 논리적 의미가 아니라 제어 분기의 개수로 계산한다는 점이 함정이다.

다시 나오면 이렇게 판단

코드 문제에서 순환 복잡도는 먼저 if, else if, while, for, case 같은 결정 지점을 표시한 뒤 그 개수에 1을 더해 판단한다. else는 단독으로 세지 않고, 중첩 분기는 각각 따로 센다.

암기 포인트

순환 복잡도는 결정 지점 수 + 1이며, else는 제외되고 중첩 if는 각각 더해진다.

NoteBase

문 18. 다음 파이썬 코드에 적용한 디자인 패턴은?

```
class Greeting(object):
    def say(self):
        kor = Korea()
        kor.kor_hi()

        jan = Japan()
        jan.jan_hi()

        chn = China()
        chn.ch_hi()

class Korea(object):
    def kor_hi(self):
        print("안녕하세요\n")

class Japan(object):
    def jan_hi(self):
        print("곤니찌와\n")

class China(object):
    def ch_hi(self):
        print("니하오마\n")

class Hello(object):
    def sayhello(self):
        sh = Greeting()
        sh.say()

insa = Hello()
insa.sayhello()
```

① 브릿지(bridge)

② 어댑터(adapter)

③ 퍼사드(facade)

④ 컴포지트(composite)

정답 3번

출제 포인트

이 문제는 클라이언트가 여러 하위 객체를 직접 다루지 않고 하나의 단순한 창구를 통해 기능을 쓰는 구조가 퍼사드 패턴인지 판별하는 문제이다.

핵심 개념

퍼사드는 복잡한 하위 시스템을 감추고 단순한 인터페이스를 제공하는 패턴이다. 어댑터는 맞지 않는 인터페이스를 변환하는 패턴이고, 브리지는 추상화와 구현을 분리해 독립적으로 확장하는 패턴이며, 컴포지트는 부분과 전체를 트리 구조로 동일하게 다루는 패턴이다.

NoteBase

선지별 해설

① X: 브리지는 추상화 계층과 구현 계층을 분리하고 둘을 조합하는 구조가 있어야 한다. 틀린 부분은 이 코드를 브리지로 보는 판단이다. 올바르게는 `Greeting`이 `Korea`, `Japan`, `China` 호출을 한곳에 모아 단순화한 구조이다.
② X: 어댑터는 기존 객체의 호환되지 않는 인터페이스를 다른 목표 인터페이스로 바꿔 준다. 틀린 부분은 인터페이스 변환이 일어난다고 보는 점이다. 올바르게는 이 코드는 `kor_hi`, `jan_hi`, `ch_hi`를 다른 이름으로 바꾸지 않고 단지 접근을 감춘다.
③ O: `Hello`는 `Greeting`만 사용하고 내부의 `Korea`, `Japan`, `China`를 직접 알 필요가 없다. `Greeting.say()`가 여러 하위 클래스를 순서대로 호출하는 단순한 진입점을 제공하므로 퍼사드 패턴에 해당한다.
④ X: 컴포지트는 객체를 부분-전체의 트리로 묶어 리프와 복합 객체를 같은 방식으로 다루는 패턴이다. 틀린 부분은 이 코드에 트리 구조와 재귀적 합성이 있다고 보는 점이다. 올바르게는 이 코드는 계층적 합성이 아니라 여러 하위 시스템 호출의 단순화이다.

오답 포인트

클래스가 여러 개 나온다는 이유만으로 브리지나 컴포지트로 오인하기 쉽다. 이 문항은 객체 수가 아니라 `Hello`가 실제로 무엇을 직접 알고 호출하는지를 보면 풀린다. `Hello`는 세 나라 클래스의 메서드를 직접 호출하지 않고 `Greeting` 하나만 호출하므로 핵심은 인터페이스 단순화이다.

다시 나오면 이렇게 판단

클라이언트 앞에 놓인 클래스가 복잡한 여러 객체를 대신 호출해 주면 퍼사드로 판단한다. 인터페이스를 변환하면 어댑터, 추상화와 구현을 분리하면 브리지, 부분과 전체를 같은 방식으로 다루면 컴포지트로 구분한다.

암기 포인트

퍼사드는 복잡한 내부를 숨기고 한 개의 쉬운 창구를 제공하는 패턴이다.

NoteBase

문 19. 다음 파이썬 코드에 해당하는 결합도(coupling)는?

```
a = 0
b = 0

def getinput():
    c = int(input("숫자 입력? "))
    return c

def getadd():
    c = a + b
    return c

def main():
    global a
    global b
    a = getinput()
    b = getinput()
    c = getadd()
    print("{} + {} = {}".format(a, b, c))

main()
```

① 스탬프(stamp) 결합도

② 제어(control) 결합도

③ 공통(common) 결합도

④ 내용(content) 결합도

정답 3번

출제 포인트

이 문제는 함수들이 전역 데이터를 함께 사용할 때의 결합도를 구별할 수 있는지를 묻는다.

핵심 개념

공통 결합도는 여러 모듈이 동일한 전역 변수나 공용 데이터를 함께 참조하는 형태이다.
스탬프 결합도는 배열·레코드·구조체처럼 복합 자료구조 전체를 전달하는 형태이다.
제어 결합도는 처리 흐름을 바꾸는 제어 정보나 플래그를 매개변수로 넘기는 형태이다.
내용 결합도는 한 모듈이 다른 모듈의 내부 데이터나 내부 로직을 직접 건드리는 형태이다.

NoteBase

선지별 해설

1번 X: 스탬프 결합도는 복합 자료구조 전체를 인수로 넘길 때 성립한다. 이 코드에는 구조체나 레코드 전달이 없다. 틀린 표현은 "스탬프 결합도"이고, 올바르게는 "전역 데이터를 공유하는 공통 결합도"이다.
2번 X: 제어 결합도는 호출하는 쪽이 제어 플래그를 넘겨 피호출 모듈의 처리 경로를 바꾸는 경우이다. 이 코드에는 분기 제어용 인수가 없다. 틀린 표현은 "제어 결합도"이고, 올바르게는 "전역 변수 a, b를 함께 쓰는 공통 결합도"이다.
3번 O: getadd()가 매개변수 없이 전역 변수 a와 b를 직접 참조하고, main()도 같은 전역 변수 a와 b를 갱신한다. 여러 함수가 동일한 전역 데이터를 공유하므로 공통 결합도이다.
4번 X: 내용 결합도는 한 모듈이 다른 모듈의 내부 구현이나 내부 데이터에 직접 접근하는 가장 강한 결합이다. 이 코드처럼 공용 전역 데이터를 함께 쓰는 것과는 다르다. 틀린 표현은 "내용 결합도"이고, 올바르게는 "공유 전역 변수에 의한 공통 결합도"이다.

오답 포인트

전역 변수를 사용한다는 사실만 보고 막연히 결합도가 높다고만 판단하면 내용 결합도와 혼동하기 쉽다. 이 문제의 핵심은 "다른 함수의 내부를 건드렸는가"가 아니라 "여러 함수가 같은 전역 데이터를 함께 썼는가"이다.

다시 나오면 이렇게 판단

매개변수로 무엇을 주고받는지보다 먼저 데이터 위치를 본다. 여러 모듈이 전역 변수·공용 영역을 함께 쓰면 공통 결합도이고, 제어용 값 전달이면 제어 결합도이며, 복합 자료구조 전달이면 스탬프 결합도이고, 다른 모듈 내부를 직접 건드리면 내용 결합도이다.

암기 포인트

전역 데이터 공유는 공통 결합도, 제어 플래그 전달은 제어 결합도, 내부 직접 접근은 내용 결합도이다.

NoteBase

문 20. 다음 세부 품질 요소를 모두 포함하는 ISO/IEC 9126의 소프트웨어 품질 특성은?

◯ 보안성
◯ 준수성
◯ 상호운용성
◯ 정확성
◯ 적합성

① 신뢰성(reliability)
② 기능성(functionality)
③ 사용성(usability)
④ 안정성(stability)

정답 2번

출제 포인트

ISO/IEC 9126의 품질특성과 세부 품질요소의 소속 관계를 구분하는 문제이다.

핵심 개념

기능성은 소프트웨어가 요구 기능을 제공하는 능력으로 적합성, 정확성, 상호운용성, 보안성, 준수성을 포함한다.
신뢰성은 주어진 조건에서 성능을 유지하는 능력으로 성숙성, 결함허용성, 회복성을 포함한다.
사용성은 사용자가 이해하고 배우고 운용하기 쉬운 정도를 뜻한다.
안정성은 ISO/IEC 9126에서 독립 품질특성이 아니라 유지보수성의 하위요소이다.

선지별 해설

① X. 신뢰성은 성숙성, 결함허용성, 회복성과 관련된다. 제시된 보안성, 준수성, 상호운용성, 정확성, 적합성은 신뢰성 요소가 아니다. 틀린 표현은 '신뢰성'이며 올바르게는 '기능성'이다.
② O. 기능성은 요구 기능 제공 능력을 뜻하며 적합성, 정확성, 상호운용성, 보안성, 준수성을 세부 품질요소로 둔다. 제시된 다섯 요소를 모두 포함하므로 옳다.
③ X. 사용성은 이해가능성, 학습성, 운용성 등 사용 편의와 관련된다. 제시된 다섯 요소는 사용성 요소가 아니다. 틀린 표현은 '사용성'이며 올바르게는 '기능성'이다.
④ X. 안정성은 ISO/IEC 9126에서 유지보수성의 하위요소로 다루어지며, 문제의 다섯 요소를 묶는 상위 품질특성이 아니다. 틀린 표현은 '안정성'이며 올바르게는 '기능성'이다.

NoteBase

오답 포인트

보안성을 별도 품질특성처럼 착각하거나, 안정성을 독립 특성으로 오인하면 틀리기 쉽다. 이 문항은 개별 용어가 아니라 다섯 세부 요소의 소속 묶음을 아는지를 묻는다.

다시 나오면 이렇게 판단

ISO/IEC 9126에서 적합성, 정확성, 상호운용성, 보안성, 준수성이 한 묶음으로 제시되면 기능성을 먼저 판단하면 된다. 안정성은 유지보수성의 하위요소인지 함께 구분하면 된다.

암기 포인트

기능성은 적합성, 정확성, 상호운용성, 보안성, 준수성의 묶음이다.

NoteBase

문 21. WBS(work breakdown structure)에 대한 설명으로 옳지 않은 것은?

① 프로젝트를 다수의 구성 요소로 분할하여 계층적으로 기술한 업무 구조이다.

② 프로젝트 수행 과정에서 요구되는 자원의 단위와 단가 정보를 작성한다.

③ 구성 요소를 산출물 중심으로 작성하거나 프로세스 중심으로 작성할 수 있다.

④ 프로젝트 전체 범위를 정의하고 프로젝트 계획과 관리를 위한 자료로 사용된다.

정답 2번

출제 포인트

이 문제는 WBS의 정의, 작성 기준, 활용 목적과 자원·비용 산정 자료를 구별하는 문제이다.

핵심 개념

WBS는 프로젝트 전체 범위를 달성하기 위해 수행할 업무나 산출물을 계층적으로 분해한 구조이다. 자원 단위나 단가 정보는 자원 계획이나 비용 산정 자료의 성격이며, 업무 분해 구조 자체를 뜻하지는 않는다.

선지별 해설

① O
프로젝트를 여러 하위 작업과 구성 요소로 나누어 계층 구조로 표현하는 것이 WBS의 기본 개념이므로 옳다.

② X
틀린 표현은 '자원의 단위와 단가 정보를 작성한다'이다. 자원의 단위와 단가 정보는 자원 계획이나 비용 산정 자료에서 다루며, WBS는 프로젝트 범위를 업무·산출물 단위로 계층 분해하여 정의하는 구조이다.

③ O
WBS는 산출물 중심으로도 작성할 수 있고 단계·활동 같은 프로세스 중심으로도 작성할 수 있으므로 옳다.
핵심은 프로젝트 범위를 빠짐없이 분해하는 방식이다.

④ O
WBS는 프로젝트 전체 범위를 명확히 정의하고 일정, 비용, 책임 배분 등 후속 계획과 통제를 위한 기준 자료로 쓰이므로 옳다.

오답 포인트

WBS도 프로젝트 계획 문서이므로 자원이나 비용 정보까지 포함한다고 섞어서 기억하기 쉽다. 그러나 WBS의 본질은 업무와 산출물의 계층 분해이고, 자원 수량·단위·단가는 별도의 자원·비용 산정 영역이라는 점이 함정이다.

NoteBase

다시 나오면 이렇게 판단

선지의 초점이 범위 정의, 업무 분해, 산출물 구조라면 WBS로 판단한다. 반대로 자원 수량, 단위, 단가, 예산 계산이 중심이면 WBS가 아니라 자원 계획이나 비용 산정 자료로 판단한다.

암기 포인트

WBS는 범위를 쪼개는 구조이고, 단가표는 비용을 계산하는 자료이다.

NoteBase

문 22. 기능점수(function point)에 대한 설명으로 옳지 않은 것은?

① 기능점수의 기준이 되는 소프트웨어 기능은 데이터 측면의 기능과 트랜잭션 측면의 기능으로 구분된다.

② 기능점수는 소프트웨어 개발 이전에 업무량을 측정할 수 있으며 소프트웨어 개발에 사용하는 언어와는 무관하다.

③ 기능점수 측정 대상 유형은 내부 논리파일, 내부 연계파일, 외부 입력, 외부 출력, 외부 조회이다.

④ 기능점수 방법은 소프트웨어 개발, 유지관리 및 재개발을 위한 비용을 산정하는 데 사용한다.

정답 3번

출제 포인트

이 문제는 기능점수의 측정 대상 5가지와 데이터 기능, 트랜잭션 기능의 구분을 정확히 아는지를 묻는 문제이다.

핵심 개념

데이터 기능은 애플리케이션이 보유하거나 참조하는 논리적 데이터 집합을 세는 기능이다. 결정 차이축은 데이터가 내부에서 유지되는지 외부 시스템에서 참조되는지이다. 키워드는 내부 논리파일, 외부 연계파일이다. 트랜잭션 기능은 데이터의 입력, 출력, 조회처럼 사용자가 시스템과 주고받는 처리 기능을 세는 기능이다. 결정 차이축은 데이터 저장 구조가 아니라 처리 행위인지 여부이다. 키워드는 외부 입력, 외부 출력, 외부 조회이다. 기능점수의 기본 측정 대상은 내부 논리파일, 외부 연계파일, 외부 입력, 외부 출력, 외부 조회의 5가지이다.

선지별 해설

① O: 기능점수는 데이터 기능과 트랜잭션 기능으로 나누어 측정한다. 데이터 기능에는 내부 논리파일과 외부 연계파일이 있고, 트랜잭션 기능에는 외부 입력, 외부 출력, 외부 조회가 있다.
② O: 기능점수는 소스코드 줄 수가 아니라 사용자 관점의 기능 수를 세는 방식이므로 개발 전에 추정할 수 있고 구현 언어에도 종속되지 않는다.
③ X: 틀린 부분은 "내부 연계파일"이다. 기능점수의 데이터 기능은 내부 논리파일(ILF)과 외부 연계파일(EIF)로 구분하므로 올바른 표현은 "내부 논리파일, 외부 연계파일, 외부 입력, 외부 출력, 외부 조회"이다.
④ O: 기능점수는 개발 규모 산정뿐 아니라 유지관리와 재개발의 노력 및 비용 추정에도 활용된다.

오답 포인트

③은 5가지 측정 대상의 목록을 거의 맞게 제시하고 한 단어만 바꾸어 놓은 함정이다. 수험생이 ILF와 EIF를 함께 외우지 않으면 "내부 논리파일"에 끌려 "내부 연계파일"도 맞는 표현처럼 오인하기 쉽다.

다시 나오면 이렇게 판단

기능점수 문제에서는 먼저 데이터 기능 2개와 트랜잭션 기능 3개를 나누어 본다. 데이터 기능 목록에서 ILF의 짝은 내부가 아니라 외부 연계파일(EIF)인지 확인하면 유사 용어 함정을 바로 걸러낼 수 있다.

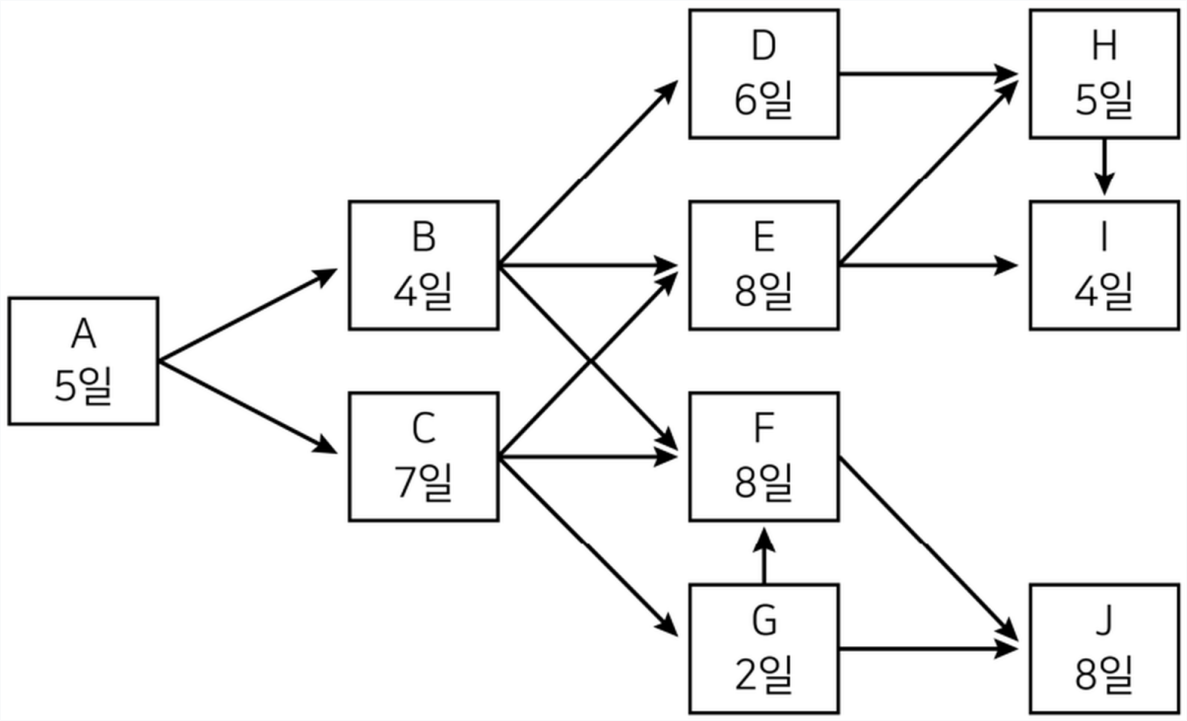
NoteBase

암기 포인트

FP 5유형은 ILF, EIF, EI, EO, EQ이며 데이터 2개와 트랜잭션 3개로 정리된다.

NoteBase

문 23. 다음 CPM 네트워크의 각 노드가 작업명과 작업에 소요되는 예상 기간을 나타낼 때, 프로젝트를 완성하는 데 예상되는 최단 기간은?



① 21일

② 24일

③ 28일

④ 30일

정답 4번

출제 포인트

이 문제는 CPM 네트워크에서 각 작업의 선행관계를 따라 가장 늦게 끝나는 완료 경로, 즉 임계경로의 총 소요기간을 구할 수 있는지를 묻는다.

조건 정리

작업 기간은 A 5일, B 4일, C 7일, D 6일, E 8일, F 8일, G 2일, H 5일, I 4일, J 8일이다.
선행관계는 A→B,C, B→D,E,F, C→E,F,G, D→H, E→H,I, G→F,J, F→J, H→I이다.
프로젝트 완료 시점은 마지막 작업들인 I와 J가 모두 끝나는 시점이므로 두 종료시점 중 큰 값을 본다.

NoteBase

사용할 공식

각 작업의 조기완료시각은 조기시작시각 + 작업기간이다.
합류 노드의 조기시작시각은 모든 선행 작업의 조기완료시각 중 최댓값이다.
프로젝트 완료기간은 종료 작업들의 조기완료시각 중 최댓값이다.
이 원리를 쓰는 이유는 CPM에서 후속 작업은 모든 선행 작업이 끝나야 시작할 수 있기 때문이다.

계산 과정

1. A는 5일에 끝난다.
2. B와 C는 A 후 시작하므로 B는 9일, C는 12일에 끝난다.
3. D는 B 후이므로 15일, G는 C 후이므로 14일에 끝난다.
4. E는 B와 C가 모두 끝나야 하므로 12일에 시작하여 20일에 끝난다.
5. F는 B, C, G가 모두 끝나야 하므로 시작시각은 max(9, 12, 14)=14일이고 22일에 끝난다.
6. H는 D와 E 후이므로 max(15, 20)=20일에 시작하여 25일에 끝난다.
7. I는 E와 H 후이므로 max(20, 25)=25일에 시작하여 29일에 끝난다.
8. J는 F와 G 후이므로 max(22, 14)=22일에 시작하여 30일에 끝난다.
9. 종료 작업은 I와 J이므로 프로젝트 완료기간은 max(29, 30)=30일이다.

선지 판단

① X: 21일은 불가능하다. E만 보아도 20일에 끝나고 H를 거치면 25일이 되므로 프로젝트 전체 완료기간이 21일일 수 없다. 틀린 표현은 완료기간을 21일로 본 점이며, 올바르게는 30일이다.
② X: 24일은 일부 경로만 단순 합산한 값이다. 실제로 I 경로도 29일, J 경로는 30일이므로 더 오래 걸린다. 틀린 표현은 완료기간을 24일로 본 점이며, 올바르게는 30일이다.
③ X: 28일은 F가 B와 C만 기다린다고 보고 G→F 선행관계를 놓치면 나오는 값이다. F는 G 완료 14일까지 기다려야 하므로 J는 30일에 끝난다. 틀린 표현은 완료기간을 28일로 본 점이며, 올바르게는 30일이다.
④ O: J의 조기완료시각이 30일이고 I의 조기완료시각은 29일이므로 프로젝트 전체 완료기간은 30일이다.

오답 포인트

합류 노드에서 선행 작업의 기간을 더하는 것이 아니라 선행 작업들의 완료시각 중 최댓값을 잡아야 한다는 점을 놓치기 쉽다. 특히 이 문항은 G→F 화살표를 빼먹으면 F 완료가 20일이 되어 ③ 28일을 고르기 쉽다.

다시 나오면 이렇게 판단

CPM 문제는 시작 노드부터 각 작업의 조기완료시각을 순서대로 적고, 합류 지점마다 선행 완료시각의 최댓값을 취한 뒤, 마지막 종료 노드들의 완료시각 중 최댓값을 정답으로 판단한다.

NoteBase

암기 포인트

CPM의 완료기간은 각 경로 합의 단순 비교가 아니라 합류 지점에서는 최댓값, 최종 완료도 종료 노드 최댓값으로 계산한다.

NoteBase

문 24. UML 설계의 4+1 View 모델에 대한 설명으로 옳은 것은?

① 논리적 관점은 시스템의 기능을 설명하기 위해 클래스 다이어그램이나 시퀀스 다이어그램을 사용한다.

② 개발 관점은 시스템이 사용자에게 제공하는 기능에 주목하며, 유스케이스 다이어그램이나 상태 다이어그램을 사용한다.

③ 물리적 관점은 시스템이 어떻게 개발되는지에 대한 관점이며, 클래스 다이어그램이나 유스케이스 다이어그램을 사용한다.

④ 프로세스 관점은 시스템에 필요한 하드웨어 환경을 나타내며, 배치 다이어그램이나 클래스 다이어그램을 사용한다.

정답 1번

출제 포인트

이 문제는 UML 4+1 View 모델에서 논리 관점, 개발 관점, 프로세스 관점, 물리적 관점, 유스케이스 관점이 각각 무엇을 설명하는지 구분하는 문제이다.

핵심 개념

논리 관점은 사용자가 요구하는 기능을 객체 구조와 상호작용으로 표현하는 관점이며 클래스 다이어그램, 시퀀스 다이어그램 등이 쓰인다.
개발 관점은 시스템을 개발자가 어떤 모듈과 컴포넌트로 조직하는지 보는 관점이며 컴포넌트 다이어그램, 패키지 다이어그램 등이 쓰인다.
프로세스 관점은 실행 시점의 프로세스, 동시성, 통신, 성능을 보는 관점이다.
물리적 관점은 소프트웨어가 어떤 하드웨어 노드에 배치되는지 보는 관점이며 배치 다이어그램이 대표적이다.
유스케이스 관점은 나머지 4개 관점을 시나리오로 묶어 시스템의 요구 기능을 설명하는 관점이다.

NoteBase

선지별 해설

① O: 논리적 관점은 시스템의 기능 요구를 객체 구조와 상호작용으로 표현하는 관점이므로 클래스 다이어그램이나 시퀀스 다이어그램으로 설명할 수 있다.
② X: 틀린 부분은 "개발 관점은 시스템이 사용자에게 제공하는 기능에 주목하며, 유스케이스 다이어그램이나 상태 다이어그램을 사용한다"는 표현이다. 개발 관점은 사용자 기능이 아니라 개발자가 시스템을 모듈과 컴포넌트로 어떻게 구성하는지를 본다. 올바르게는 "개발 관점은 소프트웨어의 모듈 구조와 구현 조직에 주목하며 컴포넌트 다이어그램이나 패키지 다이어그램을 사용한다"이다.
③ X: 틀린 부분은 "물리적 관점은 시스템이 어떻게 개발되는지에 대한 관점"과 "클래스 다이어그램이나 유스케이스 다이어그램"이라는 표현이다. 물리적 관점은 개발 절차가 아니라 시스템이 하드웨어 환경에 어떻게 배치되는지를 본다. 올바르게는 "물리적 관점은 시스템에 필요한 하드웨어 환경과 배치 구조를 나타내며 배치 다이어그램을 사용한다"이다.
④ X: 틀린 부분은 "프로세스 관점은 시스템에 필요한 하드웨어 환경을 나타내며"라는 표현이다. 하드웨어 환경은 물리적 관점의 내용이다. 프로세스 관점은 실행 중인 프로세스의 동시성, 통신, 성능을 다룬다. 올바르게는 "프로세스 관점은 시스템의 동적 동작과 프로세스 간 상호작용을 나타내며 시퀀스 다이어그램이나 활동 다이어그램 등으로 표현한다"이다.

오답 포인트

개발 관점과 유스케이스 관점, 프로세스 관점과 물리적 관점을 서로 뒤바꾸는 함정이 핵심이다. 특히 "사용자에게 제공하는 기능"은 유스케이스 또는 논리 관점 쪽 표현이고, "하드웨어 환경"은 물리적 관점의 키워드이다.

다시 나오면 이렇게 판단

4+1 View 문제는 먼저 기능, 구현 조직, 실행 동작, 하드웨어 배치, 시나리오를 각각 대응시키면 된다. 기능이면 논리 관점, 모듈 구조면 개발 관점, 동시성과 프로세스면 프로세스 관점, 하드웨어 노드면 물리적 관점, 사용자 시나리오면 유스케이스 관점으로 판단한다.

암기 포인트

논기개프물유로 묶어 기억하면 된다. 논리는 기능, 개발은 모듈, 프로세스는 동시성, 물리는 배치, 유스케이스는 시나리오이다.

NoteBase

문 25. 다음 코드에서 제시된 설계 원칙은?

```
class Gongmuwon {
    private String name;
    private String dept;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public void setDept(String dept) {
        this.dept = dept;
    }
}
```

① 캡슐화

② 다형성

③ 상속

④ 추상화

정답 1번

출제 포인트

이 문제는 객체지향 설계 원칙 중 캡슐화와 다형성·상속·추상화의 구분 기준을 묻는 문제이다.

핵심 개념

캡슐화: 데이터와 메서드를 객체 안에 묶고 내부 상태는 접근 제한자로 숨긴 뒤 정해진 메서드로만 다루게 하는 원칙이다. 결정 차이축은 접근 통제 여부이다. 키워드는 private, getter/setter이다.
다형성: 같은 메시지를 보내도 객체 타입에 따라 다른 동작이 나타나는 특성이다. 결정 차이축은 동일 인터페이스와 동작의 다양성이다. 키워드는 오버라이딩, 상위 타입 참조이다.
상속: 기존 클래스의 속성과 동작을 하위 클래스가 물려받아 재사용하는 관계이다. 결정 차이축은 클래스 간 계층 관계이다. 키워드는 extends, is-a 관계이다.
추상화: 불필요한 세부 구현을 감추고 본질적 속성만 표현하는 원칙이다. 결정 차이축은 핵심 개념의 단순화 여부이다. 키워드는 인터페이스, 추상 클래스이다.

NoteBase

선지별 해설

① O: name과 dept를 private으로 선언해 외부에서 직접 접근하지 못하게 하고, getName, setName, setDept 같은 public 메서드로만 접근하게 하므로 캡슐화이다.
② X: 다형성은 같은 메서드 호출이 객체에 따라 다르게 동작하는 특성인데, 이 코드에는 상속 관계나 오버라이딩이 없다. 틀린 표현은 "다형성"이다. 올바르게는 "private 필드와 public 접근 메서드를 통한 캡슐화"이다.
③ X: 상속은 상위 클래스의 속성과 메서드를 하위 클래스가 물려받는 구조인데, 이 코드에는 extends나 하위 클래스 정의가 없다. 틀린 표현은 "상속"이다. 올바르게는 "상속이 아니라 클래스 내부 데이터 은닉을 보여 주는 캡슐화"이다.
④ X: 추상화는 공통 본질만 뽑아 인터페이스나 추상 클래스로 표현하는 원칙인데, 이 코드는 추상 타입을 정의하지 않고 필드 접근을 통제하는 데 초점이 있다. 틀린 표현은 "추상화"이다. 올바르게는 "추상화가 아니라 private과 setter/getter를 이용한 캡슐화"이다.

오답 포인트

클래스로 현실 대상을 표현했다는 이유만으로 추상화로 보기 쉽고, 객체지향 4특성을 한꺼번에 떠올리면 private 선언의 의미를 놓치기 쉽다. 이 문항은 접근 제한자와 getter/setter가 보이면 먼저 캡슐화를 떠올리는지가 핵심이다.

다시 나오면 이렇게 판단

private, protected, public 같은 접근 제한자와 getter/setter처럼 외부 접근을 통제하는 장치가 보이면 캡슐화를 먼저 판단한다. 상속 관계가 있으면 상속, 같은 메시지에 객체별 다른 동작이 있으면 다형성, 핵심만 남긴 추상 타입이 있으면 추상화로 구분한다.

암기 포인트

private로 숨기고 메서드로 접근시키면 캡슐화이다.

NoteBase