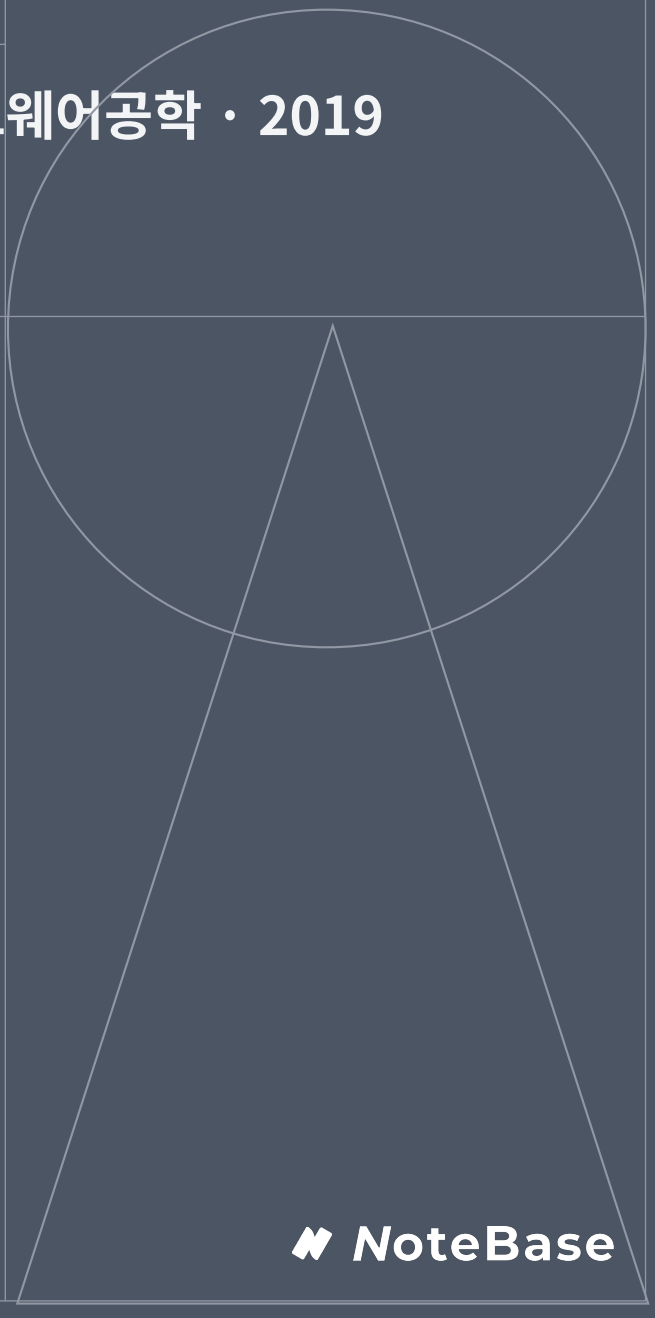
기출OX 해설집

공무원/국가직/7급 · 소프트웨어공학 · 2019



NoteBase

기출OX 해설집

공무원/국가직/7급 · 소프트웨어공학 · 2019

공무원/국가직/7급 · 소프트웨어공학 · 2019

문 1. 애자일 선언문은 애자일 방법론이 추구하고 있는 가치를 요약하고 있다. 애자일 선언문의 내용으로 옳은 것은?

① 포괄적인 문서보다는 제대로 동작하는 소프트웨어에 더 가치를 둔다.

② 고객과의 협력보다는 계약 협상에 더 가치를 둔다.

③ 변화에 대응하는 것보다는 계획을 따르는 것에 더 가치를 둔다.

④ 개인과 상호작용보다는 도구나 프로세스에 더 가치를 둔다.

정답 1번

출제 포인트

이 문제는 애자일 선언문의 네 가지 가치가 무엇인지와 서로 대비되는 표현을 정확히 구별할 수 있는지를 묻는 문제이다.

핵심 개념

애자일 선언문은 소프트웨어 개발에서 무엇을 더 우선하는지 제시한 가치 선언이다.
애자일은 포괄적인 문서보다 제대로 동작하는 소프트웨어를, 계약 협상보다 고객과의 협력을, 계획을 따르는 것보다 변화에 대응하는 것을, 도구와 프로세스보다 개인과 상호작용을 더 가치 있게 본다.

선지별 해설

1번: O. 애자일 선언문의 핵심 가치 가운데 하나가 포괄적인 문서보다 제대로 동작하는 소프트웨어를 더 중시한다는 내용이므로 옳다.
2번: X. 틀린 부분은 "고객과의 협력보다는 계약 협상에 더 가치를 둔다"는 표현이다. 애자일은 계약 협상보다 고객과의 협력을 더 가치 있게 본다.
3번: X. 틀린 부분은 "변화에 대응하는 것보다는 계획을 따르는 것에 더 가치를 둔다"는 표현이다. 애자일은 계획을 따르는 것보다 변화에 대응하는 것을 더 가치 있게 본다.
4번: X. 틀린 부분은 "개인과 상호작용보다는 도구나 프로세스에 더 가치를 둔다"는 표현이다. 애자일은 도구와 프로세스보다 개인과 상호작용을 더 가치 있게 본다.

NoteBase

오답 포인트

이 문항의 함정은 애자일 선언문의 네 가치쌍을 서로 반대로 뒤집어 놓은 데 있다. 특히 2번, 3번, 4번은 익숙한 용어를 그대로 쓰면서 우선순위만 반대로 바꾸었기 때문에 암기 방향이 흐리면 쉽게 틀린다.

다시 나오면 이렇게 판단

애자일 관련 선지는 두 요소를 대비시키는 구조인지 먼저 본다. 그런 다음 각 가치쌍에서 어느 쪽을 더 우선하는지만 확인하면 정오를 빠르게 판단할 수 있다.

암기 포인트

애자일은 문서보다 소프트웨어, 계약보다 협력, 계획보다 변화 대응, 도구보다 사람을 더 중시한다.

NoteBase

문 2. 다음에서 설명하는 아키텍처는?

◯ 시스템은 여러 개의 실행 모듈 또는 컴포넌트로 구성되며 각 컴포넌트는 독립된 기능을 제공한다.
◯ 시스템의 서비스는 컴포넌트들의 기능을 조합하여 제공된다.
◯ 컴포넌트가 상호 작용하는 데이터베이스는 다른 컴포넌트의 데이터베이스와 독립적이다.
◯ 각 컴포넌트의 실패는 격리되며 일부가 실패해도 전체 시스템이 중단되지 않는다.
◯ 각 컴포넌트를 독립적으로 개발하고 배포할 수 있으므로 지속적 통합과 지속적 배포가 가능하다.

① 피어-투-피어(peer-to-peer) 아키텍처
② 저장소(repository) 아키텍처
③ 마스터-슬레이브(master-slave) 아키텍처
④ 마이크로서비스(microservice) 아키텍처

정답 4번

출제 포인트

이 문제는 마이크로서비스 아키텍처의 핵심 특징과 다른 소프트웨어 아키텍처의 구분 기준을 묻는 문제이다.

핵심 개념

마이크로서비스 아키텍처는 시스템을 작은 서비스 단위로 분해하고, 각 서비스를 독립적으로 개발·배포·운영하는 구조이다. 핵심 키워드는 서비스 조합, 서비스별 독립 배포, 장애 격리, 서비스별 데이터 관리이다.
피어-투-피어 아키텍처는 중앙 지배 노드 없이 동등한 노드들이 직접 자원을 주고받는 구조이다.
저장소 아키텍처는 여러 컴포넌트가 하나의 중앙 저장소를 공유하는 구조이다.
마스터-슬레이브 아키텍처는 마스터가 작업을 분배하고 슬레이브가 이를 수행하는 통제 구조이다.

NoteBase

선지별 해설

① X: 피어-투-피어 아키텍처는 동등한 노드 간 직접 연결과 자원 공유가 핵심이지, 독립 서비스들을 조합해 비즈니스 기능을 제공하는 구조가 아니다. 틀린 표현은 '각 컴포넌트는 독립된 기능을 제공하고 이를 조합해 서비스를 제공한다'에 해당하는 성격을 피어-투-피어에 대응시키는 점이다. 올바르게는 '동등한 노드들이 중앙 서버 없이 직접 통신하고 자원을 공유한다'이다.
② X: 저장소 아키텍처는 여러 서브시스템이 공통 저장소를 함께 사용하는 구조이므로, 제시문처럼 컴포넌트별 데이터베이스가 서로 독립적인 구조와 반대이다. 틀린 표현은 '데이터베이스는 다른 컴포넌트의 데이터베이스와 독립적이다'에 대응하지 못한다는 점이다. 올바르게는 '중앙 저장소를 여러 컴포넌트가 공유한다'이다.
③ X: 마스터-슬레이브 아키텍처는 마스터가 슬레이브를 제어하고 역할을 분배하는 구조이므로, 각 컴포넌트를 독립적으로 배포하고 장애를 격리하는 구조와 다르다. 틀린 표현은 '각 컴포넌트를 독립적으로 개발하고 배포할 수 있다'이다. 올바르게는 '마스터가 슬레이브를 관리하며 종속적으로 작업을 수행한다'이다.
④ O: 제시문은 독립 서비스 구성, 서비스 조합, 서비스별 독립 데이터베이스, 장애 격리, 독립 배포와 CI/CD 적합성을 설명하므로 마이크로서비스 아키텍처의 특징과 일치한다.

오답 포인트

출제자는 '독립된 컴포넌트', '데이터베이스 분리', '장애 격리', '독립 배포'라는 표현을 묶어 마이크로서비스를 떠올릴 수 있는지를 물었다. 특히 저장소 아키텍처는 '데이터베이스'라는 단어만 보고 헷갈리기 쉬우나, 핵심은 데이터베이스의 존재 자체가 아니라 중앙 저장소 공유 여부이다.

다시 나오면 이렇게 판단

서비스를 작게 나누어 조합하고, 각 서비스가 독립 배포·독립 데이터 저장·장애 격리를 가진다면 마이크로서비스로 판단한다. 중앙 저장소를 함께 쓰면 저장소 아키텍처이고, 중앙 제어자가 작업을 나누면 마스터-슬레이브이며, 동등한 노드끼리 직접 연결되면 피어-투-피어이다.

암기 포인트

마이크로서비스는 서비스별 독립 배포·독립 데이터·장애 격리이고, 저장소는 중앙 저장소 공유이며, 마스터-슬레이브는 중앙 제어이고, 피어-투-피어는 동등 노드 연결이다.

NoteBase

문 3. 다음은 무엇에 사용되는 도구인가?

JUnit
Mockito
JMeter

① 버전 관리
② 프로젝트 관리
③ 소프트웨어 설계
④ 소프트웨어 테스팅

정답 4번

출제 포인트

이 문제는 대표적인 테스트 도구인 JUnit, Mockito, JMeter의 공통 용도를 보고 소프트웨어 테스트 범주를 식별하는 문제이다.

핵심 개념

버전 관리는 소스 변경 이력을 관리하는 활동이며 Git, SVN 같은 도구가 대표적이다.
프로젝트 관리는 일정, 자원, 진척을 관리하는 활동이며 Jira, MS Project 같은 도구가 대표적이다.
소프트웨어 설계는 구조와 모듈, 인터페이스를 설계하는 활동이며 UML 모델링 도구 등이 여기에 가깝다.
소프트웨어 테스트는 기능 검증, 모의 객체 기반 단위 테스트, 성능·부하 측정을 수행하는 활동이며 JUnit, Mockito, JMeter가 모두 이 범주에 속한다.

선지별 해설

① X: 버전 관리는 형상과 변경 이력을 다루는 범주이다. 제시된 JUnit, Mockito, JMeter는 소스 버전을 관리하지 않는다. 틀린 표현은 "버전 관리"이며, 올바르게는 "소프트웨어 테스트"이다.
② X: 프로젝트 관리는 일정과 인력, 작업 흐름을 통제하는 범주이다. 제시된 도구들은 일정 관리 도구가 아니라 테스트 수행과 검증을 위한 도구들이다. 틀린 표현은 "프로젝트 관리"이며, 올바르게는 "소프트웨어 테스트"이다.
③ X: 소프트웨어 설계는 아키텍처, 클래스 구조, 인터페이스를 정의하는 범주이다. JUnit은 단위 테스트 프레임워크이고, Mockito는 목 객체를 활용한 테스트 지원 도구이며, JMeter는 성능·부하 테스트 도구이다. 틀린 표현은 "소프트웨어 설계"이며, 올바르게는 "소프트웨어 테스트"이다.
④ O: JUnit은 단위 테스트, Mockito는 목 객체 기반 테스트, JMeter는 성능·부하 테스트에 쓰인다. 세 도구의 공통 용도는 모두 소프트웨어 테스트이다.

NoteBase

오답 포인트

JMeter를 개발 보조 도구 정도로만 기억하면 공통 범주를 놓치기 쉽다. 이 문항은 각 도구의 세부 역할이 달라도 모두 테스트 활동에 속한다는 점을 묻는다.

다시 나오면 이렇게 판단

도구 이름이 여러 개 제시되면 먼저 각 도구의 대표 용도를 한 단어로 환원해 공통분모를 찾는다. 단위 테스트, 목 객체 테스트, 성능 테스트처럼 종류가 달라도 상위 범주가 모두 테스트이면 정답은 소프트웨어 테스트이다.

암기 포인트

JUnit-단위 테스트, Mockito-목 객체 테스트, JMeter-성능·부하 테스트이다.

NoteBase

문 4. CMMI 프로세스 성숙도 수준(maturity level) 중 관리(Managed) 단계의 특징으로만 묶은 것은?

ㄱ. 기본적인 프로젝트 관리 프로세스가 정의되어 비용, 일정, 기능 등을 추적할 수 있다.
ㄴ. 새로운 프로젝트에 대한 계획과 관리가 이전의 성공한 프로젝트에 근거하여 이루어진다.
ㄷ. 조직의 소프트웨어 프로세스를 전담하는 소프트웨어공학 프로세스 그룹이 있다.
ㄹ. 프로세스 개선을 지속적으로 추진하여 프로세스 능력 수준을 높인다.

① ㄱ, ㄴ
② ㄴ, ㄷ
③ ㄱ, ㄴ, ㄹ
④ ㄴ, ㄷ, ㄹ

정답 1번

출제 포인트

CMMI 성숙도 2단계 Managed와 3단계 Defined, 5단계 Optimizing의 구분을 묻는 문제이다.

핵심 개념

Managed는 기본 프로젝트 관리 프로세스를 두고 비용, 일정, 기능을 계획하고 추적하는 단계이다.
Defined는 조직 표준 프로세스를 문서화하고 전사적으로 정착시키는 단계이다.
Quantitatively Managed는 측정치로 프로세스를 정량 관리하는 단계이다.
Optimizing는 지속적 프로세스 개선으로 프로세스 능력을 높이는 단계이다.

선지별 해설

1번 O: ㄱ은 비용, 일정, 기능 등을 추적할 수 있다는 내용이므로 Managed의 핵심 특징이다. ㄴ은 이전의 성공한 프로젝트 경험을 바탕으로 새 프로젝트를 계획하고 관리한다는 내용이므로 Managed의 반복 가능하고 관리된 특성에 해당한다. 따라서 ㄱ, ㄴ만 묶은 1번이 맞다.
2번 X: ㄴ은 Managed가 맞지만 ㄷ은 조직의 소프트웨어 프로세스를 전담하는 그룹을 둔다는 내용이므로 Defined 단계 특징이다. 틀린 표현은 'ㄴ, ㄷ'이다. 올바르게는 'ㄱ, ㄴ'이다.
3번 X: ㄱ, ㄴ은 Managed가 맞지만 ㄹ은 지속적으로 프로세스를 개선하여 능력 수준을 높인다는 내용이므로 Optimizing 단계 특징이다. 틀린 표현은 'ㄱ, ㄴ, ㄹ'이다. 올바르게는 'ㄱ, ㄴ'이다.
4번 X: ㄴ만 Managed이고 ㄷ은 Defined, ㄹ은 Optimizing에 해당한다. 틀린 표현은 'ㄴ, ㄷ, ㄹ'이다. 올바르게는 'ㄱ, ㄴ'이다.

NoteBase

오답 포인트

ㄷ의 '조직의 소프트웨어 프로세스를 전담하는 그룹'은 조직 표준화 키워드이므로 Defined로 가야 하고, ㄹ의 '지속적으로 추진'은 지속 개선 키워드이므로 Optimizing로 가야 한다. '관리'라는 말만 보고 조직 차원 활동이나 개선 활동까지 모두 Managed로 묶으면 오답이 된다.

다시 나오면 이렇게 판단

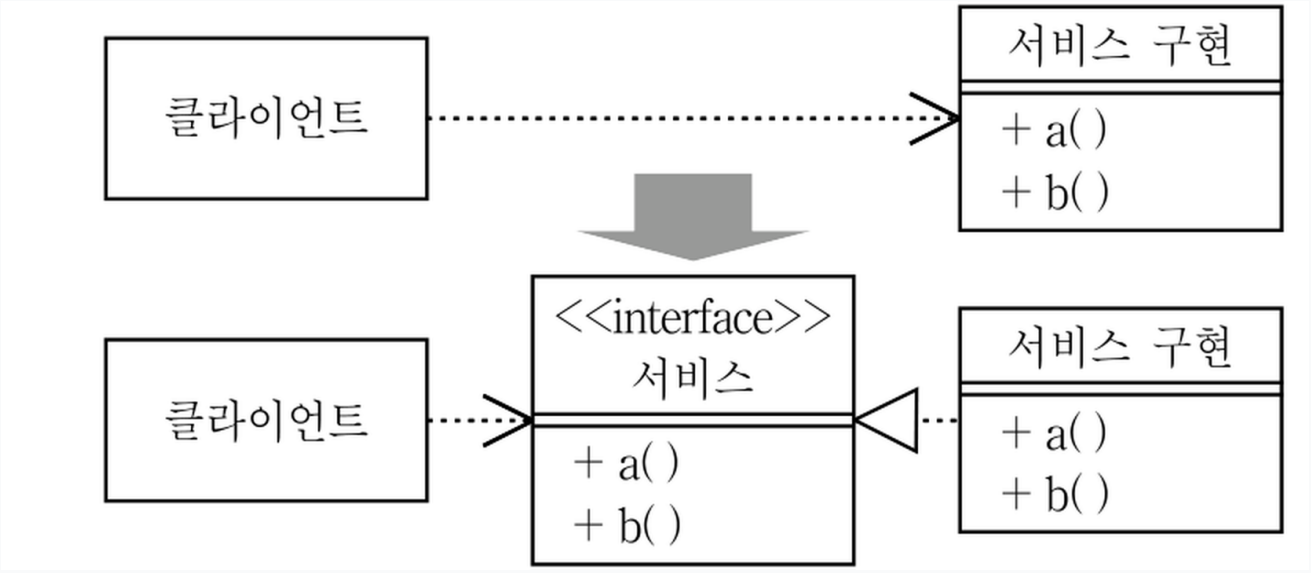
프로젝트 단위에서 계획, 추적, 통제를 하는 내용이면 2단계 Managed로 본다. 조직 표준 프로세스 정착이면 3단계 Defined, 측정치 기반 정량 관리이면 4단계 Quantitatively Managed, 지속적 개선이면 5단계 Optimizing로 판단한다.

암기 포인트

Managed는 프로젝트 계획·추적 단계, Defined는 조직 표준화 단계, Optimizing는 지속 개선 단계이다.

NoteBase

문 5. 그림과 같이 서비스 구현 클래스의 a(), b() 연산을 사용하는 클라이언트 클래스가 서비스 구현 클래스에 직접 의존하는 관계에서 클라이언트 클래스가 서비스 인터페이스에 의존하고 서비스 구현 클래스는 서비스 인터페이스를 구현하는 것으로 설계를 변경하였다. 다음 중 이와 가장 관련이 깊은 SOLID 설계 원칙은?



① 단일 책임 원칙(Single Responsibility Principle)

② 리스코프 치환 원칙(Liskov Substitution Principle)

③ 의존관계 역전 원칙(Dependency Inversion Principle)

④ 인터페이스 분리 원칙(Interface Segregation Principle)

정답 3번

출제 포인트

이 문제는 도식에서 클라이언트가 구체 구현 클래스가 아니라 서비스 인터페이스에 의존하도록 바뀐 구조를 보고, 고수준 모듈과 저수준 모듈이 모두 추상화에 의존하게 만드는 원칙을 묻는 문제이다.

NoteBase

선지별 해설

① X
단일 책임 원칙은 클래스가 변경될 이유를 하나만 가져야 한다는 원칙이다. 이 도식은 책임 수를 나누는 문제가 아니라 의존 대상을 구체 클래스에서 인터페이스로 바꾸는 문제이다. 틀린 표현은 "단일 책임"이며, 올바르게는 "클라이언트와 구현 클래스의 의존을 추상화로 옮기는 원칙"이다.

② X
리스코프 치환 원칙은 하위 타입이 상위 타입을 대체해도 프로그램의 의미가 깨지지 않아야 한다는 원칙이다. 그림은 구현 클래스가 인터페이스를 구현한다는 점은 보이지만, 핵심은 치환 가능성 검증이 아니라 의존 방향 변경이다. 틀린 표현은 "치환"이며, 올바르게는 "구체 구현에 대한 직접 의존을 추상화 의존으로 바꾸는 것"이다.

③ O
의존관계 역전 원칙은 고수준 모듈인 클라이언트와 저수준 모듈인 서비스 구현이 모두 추상화인 서비스 인터페이스에 의존해야 한다는 원칙이다. 그림에서도 클라이언트의 의존 대상이 구체 서비스 구현에서 서비스 인터페이스로 바뀌고, 구현 클래스가 그 인터페이스를 구현하므로 의존 방향이 추상화 중심으로 재구성된다.

④ X
인터페이스 분리 원칙은 클라이언트가 사용하지 않는 메서드에 의존하지 않도록 인터페이스를 잘게 분리하는 원칙이다. 이 문제의 인터페이스는 a(), b()를 그대로 제공하며, 클라이언트가 불필요한 기능에 끌려 들어가는 상황을 묻지 않는다. 틀린 표현은 "분리"이며, 올바르게는 "구체 클래스 직접 의존을 인터페이스 의존으로 역전하는 것"이다.

오답 포인트

인터페이스가 등장하면 곧바로 인터페이스 분리 원칙으로 오인하기 쉽다. 그러나 이 문제의 핵심은 인터페이스를 몇 개로 나누었는지가 아니라, 클라이언트가 구체 구현이 아닌 추상화에 의존하도록 바뀌었다는 점이다. ②도 구현 클래스가 인터페이스를 구현하므로 맞아 보이지만, 치환 가능성 자체를 검증하는 문항은 아니다.

다시 나오면 이렇게 판단

도식 문제에서 먼저 본 것은 의존 화살표가 어디를 향하는지이다. 클라이언트가 구체 클래스 대신 인터페이스나 추상 클래스에 의존하고, 구현체가 그 추상화를 따르는 구조이면 의존관계 역전 원칙을 우선 판단할 수 있다. 반대로 인터페이스를 역할별로 쪼개는지가 보이면 인터페이스 분리 원칙을 검토한다.

암기 포인트

DIP는 고수준과 저수준이 모두 구체가 아니라 추상화에 의존하는 구조이다.

NoteBase

문 6. 어떤 웹 서비스 시스템은 다음과 같은 특징을 가지고 있다. 이 시스템과 관련하여 ISO/IEC 9126 품질 특성 중에서 개선할 필요가 있는 것은?

온라인/오프라인 도움말을 제공하지 않는다.
시스템이 제공하는 기능을 메뉴명으로 이해하기 어렵다.
모든 웹 페이지에서 홈페이지로 바로 가는 '홈 버튼'이 제공되지 않아 이전 페이지로 이동하는 '뒤로 가기 버튼'을 이용하여 여러 단계를 거쳐 홈페이지로 갈 수밖에 없다.

① 효율성(efficiency)
② 사용성(usability)
③ 이식성(portability)
④ 유지보수성(maintainability)

정답 2번

출제 포인트

주어진 웹 서비스의 불편 사항을 ISO/IEC 9126 품질 특성에 대응시켜 어떤 품질 속성이 부족한지 판단하는 문제이다.

사례 정리

- 온라인/오프라인 도움말이 없다.
- 메뉴명만으로 기능을 이해하기 어렵다.
- 홈 버튼이 없어 이전 페이지를 여러 단계 거쳐 돌아가야 한다.
- 판단에 중요한 조건은 사용자 입장에서 이해하기 쉽고 조작하기 쉬운지 여부이다.

핵심 개념

- 사용성은 사용자가 시스템의 기능을 이해하고 배우고 조작하기 쉬운 정도를 뜻한다.
- ISO/IEC 9126의 사용성에는 이해가능성, 학습성, 운용성과 같은 요소가 포함된다.
- 효율성은 응답시간이나 자원 사용의 적절성, 이식성은 다른 환경으로 옮겨 사용할 수 있는 정도, 유지보수성은 수정과 시험의 용이성을 뜻한다.

적용 기준

이 사례에서는 속도나 자원 사용량이 아니라 도움말 부재, 이해하기 어려운 메뉴명, 불편한 이동 구조가 문제이다. 핵심은 사용자가 기능을 이해하고 원하는 화면으로 쉽게 이동할 수 있는지이다.

NoteBase

선지별 해설

① X. 효율성은 처리 성능, 응답시간, 자원 사용과 관련된 속성이다. 제시문에는 느린 처리나 과도한 자원 소모가 아니라 사용자가 쓰기 불편한 문제가 제시되어 있다. 틀린 표현은 '효율성'이다. 올바르게는 '사용성'이다.
② O. 도움말이 없고 메뉴명이 이해하기 어려우며 홈으로 바로 가지 못하는 것은 이해가능성과 운용성이 떨어진다는 뜻이므로 사용성 문제이다.
③ X. 이식성은 운영체제나 환경이 바뀌어도 쉽게 이전하거나 설치할 수 있는 정도를 말한다. 제시문에는 다른 플랫폼으로의 이전 문제는 없다. 틀린 표현은 '이식성'이다. 올바르게는 '사용성'이다.
④ X. 유지보수성은 오류 수정, 기능 변경, 시험의 용이성과 관련된다. 제시문은 개발자 관점의 수정 용이성이 아니라 사용자 관점의 이용 불편을 설명한다. 틀린 표현은 '유지보수성'이다. 올바르게는 '사용성'이다.

오답 포인트

웹 페이지를 여러 단계 거쳐 이동해야 한다는 문장을 보고 시간을 많이 쓴다는 이유로 효율성으로 오해하기 쉽다. 그러나 이 문항의 초점은 처리 속도가 아니라 탐색과 조작의 편의성이다.

다시 나오면 이렇게 판단

품질 특성 문제에서는 먼저 불편의 주체가 사용자인지 개발자인지부터 구분한다. 도움말, 메뉴 이해도, 화면 이동 편의는 사용성으로 판단하고, 속도와 자원은 효율성, 수정 용이는 유지보수성, 환경 이전은 이식성으로 판단한다.

암기 포인트

도움말, 메뉴 이해, 화면 이동이 불편하면 사용성 문제이다.

NoteBase

문 7. 스크럼(Scrum)의 제품 백로그(product backlog)에 대한 설명으로 옳지 않은 것은?

① 제품 백로그에 있는 업무 목록은 프로젝트를 수행하는 동안 수정되고 정제된다.

② 제품 백로그의 업무 중 높은 우선순위를 갖는 항목부터 개발한다.

③ 제품 백로그에 있는 업무의 우선순위를 결정한 후에는 변경하지 않는다.

④ 제품 책임자(product owner)가 제품 백로그를 관리한다.

정답 3번

출제 포인트

이 문제는 스크럼의 제품 백로그가 고정 목록이 아니라 우선순위와 내용이 계속 조정되는 동적인 목록이라는 점을 묻는 문제이다.

핵심 개념

제품 백로그: 제품에 필요한 기능, 개선, 수정 사항을 우선순위에 따라 정리한 목록이다. 프로젝트가 진행되는 동안 계속 추가, 수정, 재정렬된다.
우선순위: 비즈니스 가치와 필요도에 따라 높은 항목부터 먼저 개발한다.
제품 책임자: 제품 백로그의 내용과 순서를 관리하고 우선순위를 결정하는 책임을 진다.

선지별 해설

① O: 제품 백로그는 프로젝트 수행 중에도 계속 정제되고 수정된다. 제품 요구가 바뀌면 항목의 내용과 순서도 함께 바뀔 수 있으므로 옳은 설명이다.
② O: 제품 백로그 항목은 우선순위가 높은 것부터 구현하는 것이 원칙이다. 가치가 큰 항목을 먼저 개발해 점진적으로 제품을 완성하므로 옳은 설명이다.
③ X: 틀린 부분은 "우선순위를 결정한 후에는 변경하지 않는다"이다. 제품 백로그의 우선순위는 시장 상황, 요구 변화, 피드백에 따라 계속 바뀔 수 있다. 올바르게는 "제품 백로그에 있는 업무의 우선순위는 필요에 따라 변경된다"이다.
④ O: 제품 책임자(product owner)는 제품 백로그를 관리하는 역할을 맡는다. 항목의 우선순위를 조정하고 무엇을 먼저 개발할지 결정하므로 옳은 설명이다.

오답 포인트

애자일이라고 해서 우선순위가 없다는 뜻이 아니라, 우선순위가 고정되지 않고 계속 재조정된다는 뜻이다. ③은 "우선순위가 있다"는 말과 "우선순위가 영구히 고정된다"는 말을 섞어 함정을 만든 선택지이다.

NoteBase

다시 나오면 이렇게 판단

스크럼의 백로그 관련 선지는 먼저 관리 주체가 제품 책임자인지, 다음으로 목록이 살아 있는 동적 목록인지, 마지막으로 높은 우선순위부터 개발하는지의 세 기준으로 판단하면 된다.

암기 포인트

제품 백로그는 제품 책임자가 관리하는 우선순위 목록이며, 우선순위는 고정이 아니라 계속 조정된다.

NoteBase

문 8. 개발자 A는 <명세>에 따라 <코드>를 작성한 후 테스팅을 수행하였다. A는 100% 문장 커버리지를 달성하면서 동시에 프로그램의 오류를 발견할 수 있었다. A가 사용한 테스트 입력은? (단, 단축연산(short-circuit evaluation)은 수행하지 않는다)

<명 세>
두 정수를 입력 받아 두 정수 중 적어도 하나가 음수이면 두 정수의 곱을 반환하고 그렇지 않다면 두 정수의 합을 반환한다.

<코 드>

```
int foo(int v1, int v2) {
    int v3 = v1*v2;
    if (v1 >= 0 || v2 >= 0)
        v3 = v1+v2;
    return v3;
}
```

① (v1 = -2, v2 = 2)
② (v1 = 2, v2 = 0)
③ (v1 = -2, v2 = -2)
④ (v1 = 0, v2 = 0)

정답 1번

출제 포인트

이 문제는 문장 커버리지를 100%로 만들려면 if 본문이 실행되어야 한다는 점과, 명세와 코드의 분기 조건이 서로 다르다는 점을 함께 적용하는 문제이다.

사례 정리

명세에서는 두 정수 중 하나라도 음수이면 곱을 반환하고, 둘 다 음수가 아니면 합을 반환한다.
코드에서는 먼저 곱을 저장한 뒤 `v1 >= 0 || v2 >= 0`이면 합으로 덮어쓴다.
따라서 코드는 두 수가 모두 음수일 때만 곱을 반환하므로 명세와 조건이 다르다.

NoteBase

핵심 개념

문장 커버리지는 모든 실행문이 한 번 이상 수행되는지를 보는 기준이다.
이 문항에서는 `v3 = v1 + v2;` 문장까지 실행되어야 100% 문장 커버리지가 된다.

적용 기준

오류를 찾으려면 명세의 반환값과 코드의 반환값이 달라져야 한다.
100% 문장 커버리지를 만족하려면 if 조건이 참이어서 본문이 실행되어야 한다.
따라서 한 입력이 이 두 조건을 동시에 만족하는지를 보면 된다.

선지별 해설

1번 O: 입력이 (-2, 2)이면 명세는 음수가 하나라도 있으므로 곱인 -4를 반환해야 한다. 그러나 코드는 `v2 >= 0`가 참이어서 if 본문을 실행하고 합인 0을 반환하므로 오류가 드러난다. 또한 if 본문까지 실행되므로 100% 문장 커버리지가 된다.
2번 X: 입력이 (2, 0)이면 명세도 합 2를 반환하고 코드도 if 본문을 실행해 합 2를 반환하므로 오류가 드러나지 않는다. 틀린 판단은 '이 입력으로 오류를 발견할 수 있다'는 점이고, 올바르게는 '100% 문장 커버리지는 달성하지만 오류는 발견하지 못한다'이다.
3번 X: 입력이 (-2, -2)이면 명세도 곱 4를 반환하고 코드도 if 조건이 거짓이어서 곱 4를 반환하므로 오류가 드러나지 않는다. 또한 if 본문이 실행되지 않아 100% 문장 커버리지가 아니다. 틀린 판단은 '이 입력이 정답이 될 수 있다'는 점이고, 올바르게는 '오류도 드러나지 않고 문장 커버리지도 100%가 아니다'이다.
4번 X: 입력이 (0, 0)이면 명세도 합 0을 반환하고 코드도 if 본문을 실행해 합 0을 반환하므로 오류가 드러나지 않는다. 틀린 판단은 '이 입력으로 오류를 발견할 수 있다'는 점이고, 올바르게는 '100% 문장 커버리지는 달성하지만 오류는 발견하지 못한다'이다.

오답 포인트

문장 커버리지를 분기 커버리지처럼 오해하면 3번을 고르기 쉽다. 이 문제의 함정은 '오류를 드러내는 입력'과 'if 본문을 실제로 실행하는 입력'을 동시에 만족해야 한다는 점이다. 한 수만 음수인 경우에만 명세와 코드의 차이가 드러난다.

다시 나오면 이렇게 판단

커버리지 문제는 먼저 반드시 실행되어야 하는 문장을 찾고, 그다음 명세의 분기 조건과 코드의 분기 조건을 비교한다. 이후 '그 문장을 실행시키는가'와 '명세 결과와 코드 결과가 다른가'를 동시에 만족하는 입력만 정답으로 남기면 된다.

암기 포인트

명세는 '하나라도 음수이면 곱'이고 코드는 '하나라도 0 이상이면 합'이므로, 한 수만 음수인 입력이 오류를 가장 잘 드러낸다.

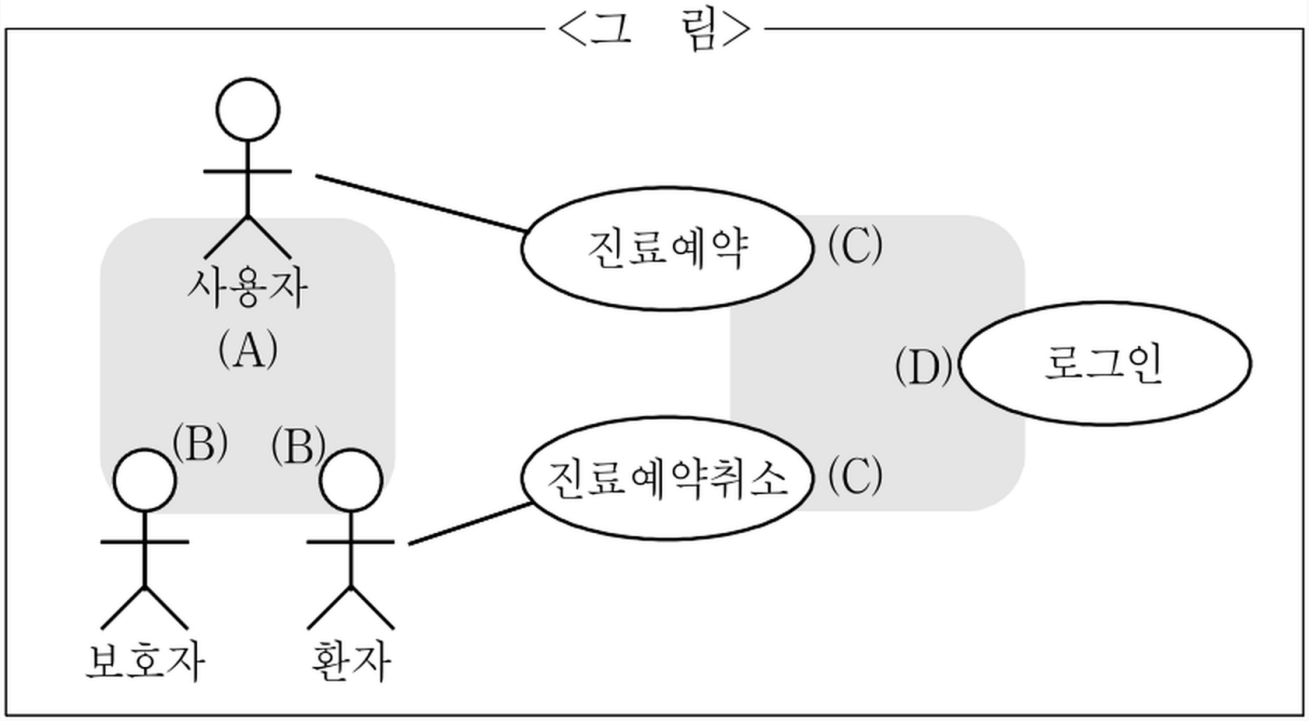
NoteBase

문 9. <그림>은 <보기>에서 기술하고 있는 병원 진료 시스템의 일부 명세에 해당하는 유스케이스 다이어그램이다. <그림>에서 '사용자'와 '보호자', '사용자'와 '환자' 사이에 들어갈 관계와 '진료예약'과 '로그인', '진료예약취소'와 '로그인' 사이에 들어갈 관계로 옳은 것은? (단, A, B, C, D는 다이어그램의 구성 요소가 아니며 관계의 방향을 지시한다)

<보기>

병원 진료 시스템 사용자로 환자와 보호자가 있다. 환자는 진료예약이나 진료예약취소를 모두 할 수 있는 반면에 보호자는 진료예약만이 가능하다. 또한 진료예약이나 진료예약취소를 하기 위해서는 반드시 로그인 절차가 필요하다.

<그 림>



① (A) ◁—— (B), (C) ···<<include>>···▷ (D)

② (A) ◁—— (B), (C) ···<<extend>>···◁ (D)

③ (A) ——▷ (B), (C) ···<<include>>···▷ (D)

④ (A) ——▷ (B), (C) ···<<extend>>···◁ (D)

정답 1번

출제 포인트

이 문제는 유스케이스 다이어그램에서 액터 일반화의 방향과 필수 공통 기능에 쓰는 `<<include>>` 관계를 도식에 맞게 배치할 수 있는지를 묻는 문제이다.

NoteBase

선지별 해설

① O: 보호자와 환자는 모두 사용자의 하위 액터이므로 액터 일반화가 들어간다. UML 일반화는 하위 액터에서 상위 액터로 빈 삼각형 화살표가 향하므로 (B)에서 (A)로 향하는 방향이 맞다. 또한 진료예약과 진료예약취소를 하려면 반드시 로그인이 필요하므로 두 유스케이스는 로그인 유스케이스를 포함한다. 따라서 (C)에서 (D)로 향하는 `<<include>>` 관계가 맞다.
② X: 액터 일반화 방향은 맞지만 로그인 관계가 틀렸다. 문제의 조건은 로그인 절차가 반드시 필요하다는 것이므로 `<<extend>>`가 아니라 `<<include>>`이어야 한다. 올바르게는 진료예약과 진료예약취소에서 로그인으로 향하는 `<<include>>` 관계이다.
③ X: 로그인 관계는 `<<include>>`로 맞지만 액터 일반화 방향이 반대이다. 사용자(A)가 보호자·환자(B)의 특수화가 아니라 보호자와 환자가 사용자의 특수화이다. 올바르게는 보호자와 환자에서 사용자(A)로 향하는 일반화 관계이다.
④ X: 두 관계가 모두 틀렸다. 액터 쪽은 사용자에서 보호자·환자로 향하지 않고 보호자와 환자에서 사용자로 향해야 한다. 로그인 쪽도 선택적으로 붙는 `<<extend>>`가 아니라 필수 공통 절차를 뜻하는 `<<include>>`가 들어가야 한다.

오답 포인트

유스케이스 다이어그램에서는 문장 의미만 따라가면 화살표 방향을 거꾸로 두기 쉽다. `사용자로 환자와 보호자가 있다`는 문장을 보고 사용자에서 하위 액터로 화살표를 그리기 쉽고, `로그인 후 예약`이라는 표현만 보고 `<<extend>>`로 오해하기 쉽다. 이 문항의 함정은 선택지 2에서 관계 종류를 바꾸고, 선택지 3에서 일반화 방향만 뒤집어 부분적으로 맞는 그림처럼 보이게 만든 점이다.

다시 나오면 이렇게 판단

명세에서 어떤 행위자가 상위 개념의 종류로 제시되면 하위 액터에서 상위 액터로 일반화를 둔다. 또 어떤 기능을 수행할 때 공통 절차가 반드시 선행되면 기본 유스케이스에서 공통 유스케이스로 `<<include>>`를 둔다. 선택적으로 추가되는 부가 기능일 때만 `<<extend>>`를 쓴다.

암기 포인트

하위 액터의 빈 삼각형은 상위 액터를 향하고, 반드시 수행되는 공통 절차는 `<<include>>`이다.

NoteBase

문 10. 다음은 어떤 시스템의 유지보수를 위한 요구사항의 일부이다. 아래 요구사항에 따라 수행한 유지보수 활동이 기존 기능에 영향을 끼쳤는지 알아보기 위해 수행하는 테스팅은?

Req-01	윈도우 환경에서 동작하는 시스템을 리눅스 환경에서도 동작하도록 한다.
Req-02	SMS 문자 발송 기능의 오류를 수정한다.
Req-03	논리 흐름을 보다 이해하기 쉽도록 코드 구조를 개선한다.

① 회귀 테스팅(regression testing)

② 사용성 테스팅(usability testing)

③ 성능 테스팅(performance testing)

④ 보안성 테스팅(security testing)

정답 1번

출제 포인트

이 문제는 유지보수 수행 뒤 기존 기능이 깨지지 않았는지 확인하는 테스트가 무엇인지 구별하는 문제이다.

핵심 개념

회귀 테스트는 수정·이식·구조개선 뒤 기존에 되던 기능이 여전히 정상 동작하는지 다시 확인하는 테스트이다.
사용성 테스트는 사용자가 시스템을 쉽게 배우고 사용할 수 있는지를 평가하는 테스트이다.
성능 테스트는 응답시간, 처리량, 자원 사용량 같은 성능 특성을 측정하는 테스트이다.
보안성 테스트는 취약점, 인증·인가, 데이터 보호 같은 보안 요구 충족 여부를 점검하는 테스트이다.

NoteBase

선지별 해설

1번 O: 유지보수 요구사항에는 운영체제 이식, 오류 수정, 코드 구조 개선이 포함되어 있다. 이런 변경 뒤 기존 기능에 영향이 생겼는지 확인하는 대표 테스트가 회귀 테스트이다.
2번 X: 틀린 부분은 "기존 기능에 영향을 끼쳤는지 알아보기 위해" 사용성 테스트를 수행한다고 보는 점이다. 사용성 테스트는 화면 구성, 조작 편의성, 학습 용이성을 평가하는 테스트이며, 이 문맥에서는 회귀 테스트로 고쳐야 한다.
3번 X: 틀린 부분은 "기존 기능 영향 확인"을 성능 테스트와 연결한 점이다. 성능 테스트는 속도, 처리량, 부하 대응을 본다. 기존 기능이 수정 때문에 오동작하는지 확인하는 테스트는 회귀 테스트이다.
4번 X: 틀린 부분은 유지보수 후 기존 기능 영향 여부를 보안성 테스트로 판단한 점이다. 보안성 테스트는 취약점과 보호기능 검증이 목적이다. 이 문맥의 올바른 표현은 회귀 테스트를 수행한다는 것이다.

오답 포인트

문제에 오류 수정과 코드 개선이 함께 제시되어 있어 유지보수 유형을 묻는다고 오해하기 쉽다. 그러나 발문 핵심은 유지보수 종류가 아니라 "기존 기능에 영향을 끼쳤는지"라는 영향 확인 문구이다.

다시 나오면 이렇게 판단

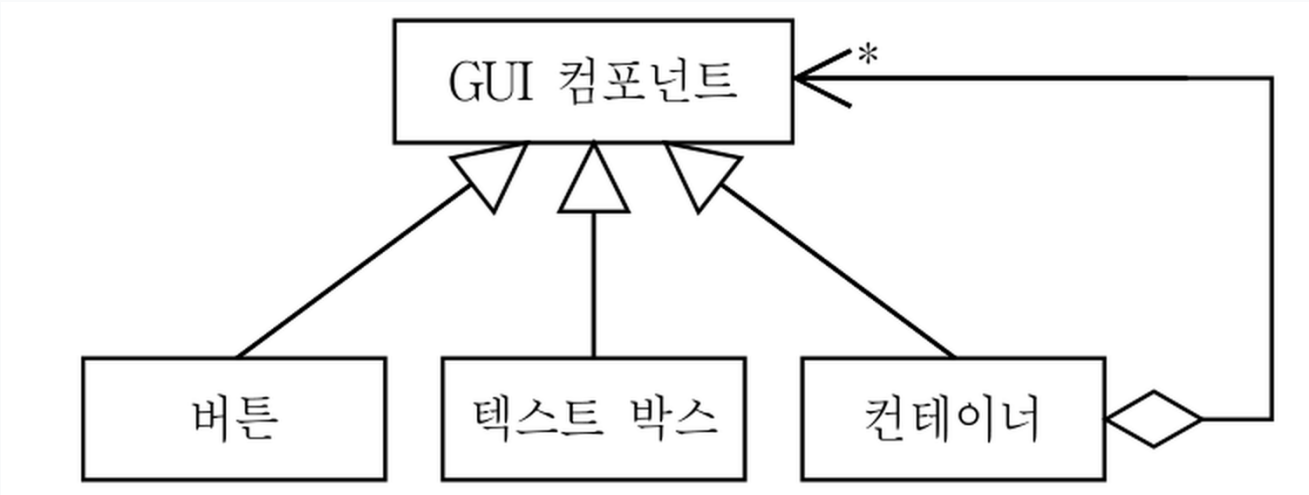
프로그램을 수정·이식·리팩터링한 뒤 예전 기능이 그대로 되는지 재확인하는 상황이면 먼저 회귀 테스트를 떠올린다. 편의성, 속도, 보안이 아니라 변경의 부작용 점검인지부터 구분하면 된다.

암기 포인트

변경 후 기존 기능 재확인은 회귀 테스트, 편의성은 사용성, 속도는 성능, 취약점은 보안성이다.

NoteBase

문 11. 다음 클래스 다이어그램으로 표현한 설계에서 사용한 디자인 패턴은?



① Adapter 패턴

② Composite 패턴

③ Observer 패턴

④ Factory Method 패턴

정답 2번

출제 포인트

이 문제는 공통 추상 컴포넌트를 두고 버튼, 텍스트 박스 같은 리프와 컨테이너가 같은 인터페이스를 공유하면서 컨테이너가 다시 GUI 컴포넌트를 포함하는 계층 구조를 보고 Composite 패턴을 식별하는 문제이다.

NoteBase

선지별 해설

① X: Adapter 패턴은 호환되지 않는 인터페이스를 변환해 기존 클래스를 다른 인터페이스로 사용할 수 있게 하는 구조이다. 이 그림에는 인터페이스 변환용 어댑터가 없고, 틀린 표현은 "서로 다른 인터페이스 연결"이다. 올바르게는 "부분-전체 계층을 동일한 인터페이스로 다루는 구조"이다.
② O: GUI 컴포넌트를 공통 상위 타입으로 두고 버튼, 텍스트 박스, 컨테이너가 모두 이를 상속한다. 특히 컨테이너가 GUI 컴포넌트를 다시 여러 개 포함하므로 리프와 복합 객체를 동일하게 다루는 Composite 패턴에 해당한다.
③ X: Observer 패턴은 주체와 관찰자 사이의 일대다 의존 관계에서 상태 변화 통지를 보내는 구조이다. 이 그림에는 통지, 구독, 갱신 관계가 없고, 틀린 표현은 "변화 통지 관계"이다. 올바르게는 "재귀적 포함 관계를 가진 트리 구조"이다.
④ X: Factory Method 패턴은 객체 생성 책임을 상위 클래스의 팩토리 메서드와 하위 클래스의 구체 생성으로 분리하는 생성 패턴이다. 이 그림은 생성 과정이 아니라 객체 구성 구조를 나타내며, 틀린 표현은 "객체 생성 방식"이다. 올바르게는 "객체들을 조합해 전체-부분 구조를 표현하는 방식"이다.

오답 포인트

컨테이너도 GUI 컴포넌트의 한 종류이면서 동시에 다른 GUI 컴포넌트를 포함한다는 점을 놓치면 단순 상속 구조나 일반 연관 관계로 오해하기 쉽다. 이 문항의 함정은 상속 화살표보다 오른쪽의 집합 관계를 함께 읽어야 Composite가 드러난다는 점이다.

다시 나오면 이렇게 판단

공통 추상 타입 아래에 리프 객체와 복합 객체가 함께 매달리고, 복합 객체가 다시 같은 추상 타입의 객체들을 자식으로 가지면 Composite 패턴으로 판단한다.

암기 포인트

Composite는 "컨테이너도 컴포넌트이고 컴포넌트를 담는다"로 기억한다.

NoteBase

문 12. 어떤 소프트웨어의 고장 간 평균 시간(MTBF; Mean Time Between Failures)이 2,500시간이고 평균 수리 시간(MTTR; Mean Time To Repair)이 100시간일 때 평균 가동 시간(MTTF; Mean Time To Failure)과 가용성(availability)은? (단, 소수점 이하에서 반올림한다)

① 2,400시간, 92%

② 2,400시간, 96%

③ 2,600시간, 92%

④ 2,600시간, 96%

정답 2번

출제 포인트

이 문제는 MTBF와 MTTR의 관계를 이용해 MTTF를 구하고, 그 값으로 가용성을 계산하는 문제이다.

조건 정리

주어진 값: MTBF는 2,500시간, MTTR은 100시간이다.
구해야 할 값: MTTF와 가용성이다.
주의할 조건: MTBF는 고장까지의 가동시간과 수리시간을 합한 값이므로 MTBF = MTTF + MTTR 관계를 먼저 써야 한다.

사용할 공식

공식은 $MTBF = MTTF + MTTR$, $Availability = \frac{MTTF}{MTTF + MTTR}$이다.
이 공식을 쓰는 이유: 수리 가능한 시스템의 한 주기는 정상 가동시간과 수리시간으로 이루어지므로, 평균 고장 간 시간은 두 시간을 합한 값이고 가용성은 전체 주기 중 실제로 가동되는 시간의 비율이기 때문이다.

계산 과정

1. $MTTF = MTBF - MTTR = 2500 - 100 = 2400$이므로 평균 가동 시간은 2,400시간이다.
2. 가용성은 $\frac{MTTF}{MTTF + MTTR} = \frac{2400}{2400 + 100} = \frac{2400}{2500}$이다.
3. $\frac{2400}{2500} = 0.96$이므로 가용성은 96%이다.

NoteBase

선지별 해설

① X: MTTF 2,400시간은 맞지만 가용성 92%가 틀리다. 계산 결과 가용성은 96%이다. 틀린 표현은 "92%"이고, 올바르게는 "96%"이다.
② O: $MTTF = 2400$시간, 가용성은 $\frac{2400}{2500} = 0.96$이므로 96%가 맞다.
③ X: "2,600시간"이 틀리다. MTTF는 MTBF에서 MTTR을 빼서 구하므로 2,500 + 100이 아니라 2,500 - 100 = 2,400시간이다. 가용성도 이에 따라 92%가 아니라 96%이다. 올바르게는 "2,400시간, 96%"이다.
④ X: 가용성 96% 자체는 정답 계산과 일치하지만, "2,600시간"이 틀려 전체 선지는 오답이다. MTTF는 수리시간을 더하는 값이 아니라 빼는 값이므로 올바르게는 "2,400시간, 96%"이다.

오답 포인트

MTBF를 MTTF와 같은 뜻으로 착각하거나, MTTF를 구할 때 MTTR을 빼지 않고 더하는 실수가 자주 나온다. 또 가용성을 구할 때 분모를 MTBF로 보지 않고 임의의 비율로 계산하면 92% 같은 오답이 나온다.

다시 나오면 이렇게 판단

신뢰도 계산형 문제는 먼저 주어진 약어의 관계를 식으로 정리하고, 합인지 차인지 확인한 뒤, 구한 값을 전체 주기 대비 비율식에 대입해 판단한다.

암기 포인트

수리 가능 시스템은 $MTBF = MTTF + MTTR$, 가용성은 $\frac{MTTF}{MTBF}$로 정리된다.

NoteBase

문 13. 형상관리의 형상 제어(configuration control) 활동에서 수행하는 작업으로만 묶은 것은?

ㄱ. 형상 항목과 형상 식별자 선정
ㄴ. 변경 요청사항에 대한 심사 및 변경 실시
ㄷ. 변경 내용을 확인하고 베이스라인 수립
ㄹ. 형상 관리 계획서대로 형상관리가 진행되고 있는지 검증

① ㄱ, ㄴ
② ㄴ, ㄷ
③ ㄴ, ㄹ
④ ㄷ, ㄹ

정답 2번

출제 포인트

이 문제는 형상관리 활동 중 형상 식별, 형상 제어, 형상 감사의 역할 차이를 구분하는 문제이다.

핵심 개념

형상 식별은 형상 항목이 무엇인지 정하고 각 항목의 식별자와 기준선을 정의하는 활동이다.
형상 제어는 변경 요구를 심사하고 승인된 변경을 반영하여 기준선의 일관성을 유지하는 활동이다.
형상 감사는 형상관리 계획과 실제 결과가 일치하는지 검증하는 활동이다.

NoteBase

선지별 해설

① X
- ①은 ㄱ, ㄴ을 묶었는데 ㄱ이 형상 제어가 아니라 형상 식별에 해당하므로 틀리다.
- ㄱ의 '형상 항목과 형상 식별자 설정'은 변경 통제가 아니라 무엇을 관리 대상으로 삼을지 정하는 식별 활동이다.
- ㄴ의 '변경 요청사항에 대한 심사 및 변경 실시'는 형상 제어가 맞다.
- 올바르게는 형상 제어 활동만 묶으면 ㄴ, ㄷ이다.

② O
- ②는 ㄴ, ㄷ을 묶은 선택지이며 둘 다 형상 제어 활동에 해당하므로 옳다.
- ㄴ은 변경 요청을 검토하고 실제 변경을 반영하는 절차이므로 전형적인 형상 제어이다.
- ㄷ은 변경 내용을 확인한 뒤 그 결과를 기준선에 반영해 베이스라인을 수립하는 내용이므로, 변경을 기준선에 통제된 상태로 반영하는 형상 제어로 본다.

③ X
- ③은 ㄴ은 맞지만 ㄹ이 틀려서 오답이다.
- ㄹ의 '형상 관리 계획서대로 형상관리가 진행되고 있는지 검증'은 변경 통제가 아니라 형상 감사에 해당한다.
- 올바르게는 ㄹ 대신 ㄷ이 들어가야 형상 제어 활동이 된다.

④ X
- ④는 ㄷ은 맞지만 ㄹ이 틀려서 오답이다.
- 틀린 부분은 ㄹ을 형상 제어로 본 점이다.
- 실제로는 ㄹ은 계획 준수 여부를 확인하는 감사 활동이고, 형상 제어만 묶으면 ㄴ, ㄷ이다.

오답 포인트

형상 항목 선정과 식별자 부여는 형상관리 전체에서 중요하므로 통제 활동처럼 보이기 쉽다. 그러나 이는 변경을 다루는 단계가 아니라 관리 대상을 정의하는 형상 식별이다. 또 '베이스라인 수립'이라는 말만 보고 식별로만 외우면 ㄷ을 놓치기 쉬운데, 이 문항에서는 변경 확인 후 기준선에 반영하는 맥락이므로 형상 제어로 판단한다.

다시 나오면 이렇게 판단

문장이 관리 대상의 이름, 식별자, 초기 기준선을 정하면 형상 식별이다. 변경 요청을 심사하거나 승인된 변경을 반영하면 형상 제어이다. 계획대로 수행되었는지 점검하면 형상 감사이다.

암기 포인트

식별은 무엇을 관리할지 정하는 일이고, 제어는 변경을 다루는 일이며, 감사는 제대로 했는지 확인하는 일이다.

NoteBase

문 14. 다음은 리팩토링 전후의 코드 변화이다. 적용된 리팩토링 기법에 해당하는 것은?

```
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 메소드 은폐(Hide Method)

② 메소드 추출(Extract Method)

③ 메소드 내용 직접 삽입(Inline Method)

④ 메소드를 매개변수로 전환(Parameterize Method)

정답 3번

출제 포인트

이 문제는 Extract Method와 Inline Method의 방향 차이를 구별하는 문제이다.

핵심 개념

Extract Method는 기존 코드 일부를 별도 메소드로 분리해 호출하게 만드는 리팩토링이다.
Inline Method는 별도 메소드 호출을 없애고 그 내용을 호출 위치에 직접 넣는 리팩토링이다.
Hide Method는 메소드의 가시성을 낮춰 외부 호출을 막는 리팩토링이다.
Parameterize Method는 비슷한 여러 메소드를 매개변수를 받는 하나의 메소드로 일반화하는 리팩토링이다.

NoteBase

선지별 해설

① X: Hide Method는 메소드를 숨기는 기법이지, 보조 메소드의 본문을 호출부에 직접 넣는 기법이 아니다. 틀린 표현은 '메소드 은폐'이다. 올바르게는 메소드의 접근 범위를 줄여 외부에서 직접 쓰지 못하게 하는 리팩토링이다.
② X: Extract Method는 한 메소드 안의 코드 일부를 새 메소드로 뽑아내는 경우에 해당한다. 이 문항은 반대로 isEven(n) 호출을 제거하고 그 내용을 myMethod 안에 넣었으므로 추출이 아니다. 틀린 표현은 '메소드 추출'이다. 올바르게는 분리된 메소드 내용을 호출 위치에 합쳐 넣은 것이다.
③ O: isEven(n) 호출이 사라지고 number % 2 == 0 조건식이 직접 들어갔다. 별도 메소드의 내용을 호출부로 대체했으므로 Inline Method에 해당한다.
④ X: Parameterize Method는 여러 유사 메소드를 하나로 묶으면서 차이를 매개변수로 처리하는 기법이다. 이 문항에는 새로운 매개변수화가 없고 메소드 수도 통합되지 않았다. 틀린 표현은 '메소드를 매개변수로 전환'이다. 올바르게는 메소드 호출을 제거하고 본문을 직접 삽입한 것이다.

오답 포인트

Extract Method와 Inline Method를 반대 방향의 짝으로 보지 않으면 ②와 ③이 헷갈린다. 이 문항의 함정은 isEven이라는 별도 메소드가 있었기 때문에 추출과 관련 있다고 착각하게 만드는 점이다.

다시 나오면 이렇게 판단

변화 전후를 보고 새 메소드가 생겼으면 Extract Method이고, 기존 호출이 사라지면서 메소드 본문이 직접 들어오면 Inline Method이다. 가시성 변경이면 Hide Method이고, 여러 메소드를 하나로 일반화하면 Parameterize Method이다.

암기 포인트

추출은 밖으로 빼는 것이고, 인라인은 안으로 접어 넣는 것이다.

NoteBase

문 15. 소프트웨어 테스팅 문서에 관한 국제 표준은?

① ISO/IEC 9899

② IEEE 828

③ IEEE 829

④ IEEE 1042

정답 3번

출제 포인트

이 문제는 소프트웨어 테스트 문서 표준과 다른 소프트웨어 공학 표준 번호를 구분하는 문제이다.

핵심 정리

- ISO/IEC 9899: C 프로그래밍 언어 표준이다.
- IEEE 828: 소프트웨어 형상관리 계획과 형상관리 절차를 다루는 표준이다.
- IEEE 829: 소프트웨어 테스트 문서의 종류와 작성 내용을 다루는 표준이다.
- IEEE 1042: 소프트웨어 형상관리에 관한 가이드이다.

선지별 해설

① X. ISO/IEC 9899는 소프트웨어 테스트 문서 표준이 아니라 C 언어 표준이다. 틀린 표현은 "소프트웨어 테스트 문서"이며, 올바르게는 "C 프로그래밍 언어 국제 표준"이다.
② X. IEEE 828은 테스트 문서 표준이 아니라 소프트웨어 형상관리 표준이다. 틀린 표현은 "소프트웨어 테스팅 문서"이며, 올바르게는 "소프트웨어 형상관리 계획 및 관리 절차 관련 표준"이다.
③ O. IEEE 829는 Software Test Documentation 표준으로 테스트 계획서, 테스트 설계 명세서, 테스트 결과 보고서 등 테스트 문서 체계를 다룬다.
④ X. IEEE 1042는 테스트 문서 표준이 아니라 소프트웨어 형상관리 가이드이다. 틀린 표현은 "소프트웨어 테스팅 문서"이며, 올바르게는 "소프트웨어 형상관리 지침"이다.

오답 포인트

IEEE 828, IEEE 829, IEEE 1042처럼 모두 소프트웨어 공학 관련 번호여서 숫자만 보고 혼동하기 쉽다. 특히 828과 1042는 모두 형상관리 계열인데, 829만 테스트 문서라는 점을 놓치면 오답이 된다.

다시 나오면 이렇게 판단

국제 표준 번호를 묻는 문제는 먼저 핵심 키워드를 잡아 연결하면 된다. "829 = 테스트 문서", "828·1042 = 형상관리", "9899 = C 언어"처럼 분야를 먼저 나누면 정답을 빠르게 고를 수 있다.

NoteBase

암기 포인트

IEEE 829는 테스트 문서, IEEE 828과 1042는 형상관리, ISO/IEC 9899는 C 언어 표준이다.

NoteBase

문 16. 다음은 두 정수를 입력받아 큰 값을 출력하는 프로그램이다. 이 프로그램에서 caller 함수와 max 함수 간의 결합도와 max 함수의 응집도를 바르게 나열한 것은?

```
void caller(int x, int y) {
    int maxVal = max(x, y);
    printf("%d", maxVal);
}

int max(int v1, int v2) {
    if (v1 > v2) return v1;
    else return v2;
}
```

① 제어(control) 결합도, 논리적(logical) 응집도
② 제어(control) 결합도, 기능적(functional) 응집도
③ 데이터(data) 결합도, 논리적(logical) 응집도
④ 데이터(data) 결합도, 기능적(functional) 응집도

정답 4번

출제 포인트

이 문제는 모듈 간 결합도에서 데이터 결합도와 제어 결합도를 구별하고, 단일 기능만 수행하는 함수의 기능적 응집도를 판별할 수 있는지를 묻는다.

핵심 개념

데이터 결합도는 모듈 사이에 필요한 데이터만 매개변수로 주고받는 결합이다.
제어 결합도는 처리 흐름을 바꾸는 제어 정보나 플래그를 넘겨 다른 모듈의 내부 동작을 좌우하는 결합이다.
기능적 응집도는 모듈의 모든 요소가 하나의 명확한 기능 수행만을 위해 모인 상태이다.
논리적 응집도는 유사한 성격의 기능들을 한 모듈에 묶어 두고, 호출 시점의 조건이나 선택값에 따라 그중 하나를 수행하는 상태이다.

NoteBase

선지별 해설

1번: X. caller는 max에 정수 x, y만 전달하므로 결합도는 제어 결합도가 아니라 데이터 결합도이다. 제어 결합도라면 수행 경로를 바꾸는 제어값을 넘겨야 한다. 또 max 함수는 비교 후 큰 값을 반환하는 한 가지 기능만 수행하므로 논리적 응집도가 아니라 기능적 응집도이다.
2번: X. max 함수의 응집도를 기능적 응집도로 본 점은 맞지만, caller와 max 사이 결합도를 제어 결합도로 본 점이 틀리다. 틀린 표현은 '제어(control) 결합도'이고, 올바르게는 '데이터(data) 결합도'이다.
3번: X. caller와 max 사이에는 정수 데이터만 오가므로 데이터 결합도라는 판단은 맞다. 그러나 max 함수는 여러 유사 기능 중 하나를 선택 수행하는 구조가 아니므로 논리적 응집도가 아니다. 틀린 표현은 '논리적(logical) 응집도'이고, 올바르게는 '기능적(functional) 응집도'이다.
4번: O. caller는 max에 필요한 입력 데이터 x, y만 전달하므로 데이터 결합도이다. max 함수는 두 수를 비교해 큰 값을 반환하는 하나의 명확한 기능만 수행하므로 기능적 응집도이다.

오답 포인트

매개변수가 전달된다는 사실만 보고 제어 결합도로 오해하기 쉽다. 그러나 전달되는 값이 단순 데이터인지, 처리 흐름을 지시하는 제어 정보인지부터 구분해야 한다. 또 if문이 있다고 해서 논리적 응집도가 되는 것이 아니라, 모듈 전체가 여러 기능 중 하나를 선택 수행하는지 여부를 봐야 한다.

다시 나오면 이렇게 판단

함수 간 관계는 먼저 전달 인자의 성격을 본다. 필요한 값만 넘기면 데이터 결합도이고, 동작 선택용 플래그를 넘기면 제어 결합도이다. 함수 내부 응집도는 그 함수가 결국 하나의 목적만 수행하는지, 여러 유사 기능을 조건에 따라 골라 수행하는지를 기준으로 판단한다.

암기 포인트

값만 주고받으면 데이터 결합도, 흐름을 지시하면 제어 결합도이다. 한 함수가 한 일만 하면 기능적 응집도이다.

NoteBase

문 17. 다음 용어에 대한 설명으로 옳은 것은?

① 사용자 스토리(User Story)는 개발자 관점에서 프로젝트를 통해 구현하고 싶은 기능을 상세하게 표현한 알고리즘이다.

② 베이스라인(Baseline)은 공식적인 변경절차 없이 언제라도 개발자가 변경할 수 있는 상태에 있는 산출물의 집합이다.

③ 기능 점수(Function Point)는 시스템을 구현한 기술에 의존적이고 개발자에 의해 식별되는 기능에 기반하여 시스템의 크기를 측정하는 척도이다.

④ 순환 복잡도(Cyclomatic Complexity)는 원시 코드의 복잡도를 정량적으로 평가하는 척도이며 기본 경로(basis path) 테스팅과 밀접한 관련이 있다.

정답 4번

출제 포인트

이 문제는 사용자 스토리, 베이스라인, 기능 점수, 순환 복잡도의 정의와 적용 관점을 정확히 구별하는지를 묻는 문제이다.

핵심 개념

사용자 스토리는 사용자가 얻고 싶은 가치나 기능을 사용자 관점에서 짧게 서술한 요구사항이다.
베이스라인은 공식적으로 확정되어 이후 변경 시 형상통제 절차를 거쳐야 하는 산출물 집합이다.
기능 점수는 구현 기술과 무관하게 사용자에게 보이는 기능을 기준으로 소프트웨어 규모를 측정하는 척도이다.
순환 복잡도는 프로그램 제어 흐름의 독립 경로 수를 바탕으로 코드의 논리적 복잡도를 정량화하는 척도이다.

선지별 해설

① X: 틀린 부분은 '개발자 관점', '상세하게 표현', '알고리즘'이다. 사용자 스토리는 개발자가 아니라 사용자 관점에서 필요한 기능과 가치를 간결하게 적는 요구사항 표현이며 알고리즘이 아니다.
② X: 틀린 부분은 '공식적인 변경절차 없이 언제라도 개발자가 변경할 수 있는 상태'이다. 베이스라인은 한번 기준선으로 확정되면 임의 변경이 아니라 공식적인 변경통제 절차를 통해서만 수정되는 산출물 집합이다.
③ X: 틀린 부분은 '구현한 기술에 의존적'과 '개발자에 의해 식별되는 기능'이다. 기능 점수는 프로그래밍 언어나 구현 기술에 독립적이며, 개발자 내부 관점이 아니라 사용자에게 제공되는 기능을 기준으로 규모를 측정한다.
④ O: 순환 복잡도는 McCabe가 제안한 척도로, 제어 흐름 그래프의 독립 경로 수와 관련되어 원시 코드의 복잡도를 정량적으로 평가한다. 따라서 기본 경로 테스팅과 밀접한 관련이 있다는 설명이 맞다.

NoteBase

오답 포인트

이 문항은 각 용어의 '관점'과 '변경 가능성'을 뒤섞어 놓은 점이 함정이다. ①은 사용자 스토리를 개발자 산출물처럼 바꾸고, ②는 베이스라인의 핵심인 형상통제를 반대로 뒤집고, ③은 기능 점수의 장점인 기술 독립성을 기술 의존성으로 바꾸어 혼동을 유도한다.

다시 나오면 이렇게 판단

용어 문제는 먼저 누가 기준인지부터 본다. 사용자 스토리는 사용자 관점, 기능 점수는 사용자 기능 관점, 베이스라인은 형상통제 관점, 순환 복잡도는 제어 흐름과 테스트 경로 관점으로 연결하면 정오 판단이 빨라진다.

암기 포인트

사용자 스토리는 사용자 요구, 베이스라인은 변경통제, 기능 점수는 기술 독립 규모, 순환 복잡도는 독립 경로 수이다.

NoteBase

문 18. <명세>는 어떤 과목의 통과 여부를 결정하는 프로그램에 대한 명세이다. <코드>의 프로그램은 <명세>에 따라 작성하였지만 오류가 있다. <코드>의 오류를 검출할 수 있는 테스팅 기법과 테스트 입력을 바르게 짝 지은 것은?

<명세>
입력 점수가 70보다 크거나 같으면 통과이고 그렇지 않으면 통과하지 못한다. 점수는 0이상 100이하 범위를 갖는 정수형이다. 프로그램의 반환 값이 0이면 통과, 1이면 통과하지 못함, -1이면 입력이 범위를 벗어났음을 나타낸다.

<코 드>

```
int passOrNot(int score) {
     if ((score > 100) || (score < 0)) return -1;
     if (score > 70) return 0;
     else return 1;
}
```

① 경계 값 분석 기법, 70
② 경계 값 분석 기법, 100
③ 동등 분할 기법, 50
④ 동등 분할 기법, 80

정답 1번

출제 포인트

이 문제는 경계값 분석과 동등 분할의 차이, 그리고 비교 연산자 오류를 가장 잘 드러내는 테스트 입력이 무엇인지를 묻는 문제이다.

핵심 개념

경계값 분석은 분기 기준값이나 입력 범위의 경계 자체를 시험하는 기법이다. 이 문제의 핵심 경계는 합격 기준인 70이다.
동등 분할은 같은 결과를 내는 입력 집합에서 대표값 하나를 뽑아 시험하는 기법이다. 50은 불합격 구간의 대표값이고 80은 합격 구간의 대표값이지만, 둘 다 경계 자체는 아니다.
이 문제의 오류는 `score > 70`으로 작성되어 70점을 통과로 처리하지 못한다는 점이다. 따라서 대표값보다 경계값 70을 직접 넣어야 오류가 드러난다.

NoteBase

선지별 해설

① O
경계값 분석 기법과 입력 70의 조합이 맞다. 명세는 70 이상이면 통과인데 코드에는 `score > 70`이 들어 있어 70점일 때 0이 아니라 1을 반환한다. 경계값 70을 넣으면 명세와 구현의 불일치가 바로 드러난다.

② X
100은 입력 범위의 상한 경계이기는 하지만, 이 코드의 실제 오류를 드러내는 값은 아니다. 100을 넣으면 범위 검사도 통과하고 `score > 70`도 참이므로 0을 반환해 명세와 일치한다. 틀린 표현은 `100`이고, 올바르게는 `경계값 분석 기법, 70`이다.

③ X
50은 동등 분할에서 불합격 구간을 대표하는 값이지만 경계값이 아니다. 50을 넣으면 명세대로 불합격이므로 오류가 검출되지 않는다. 틀린 표현은 `동등 분할 기법, 50`이고, 올바르게는 `경계값 분석 기법, 70`이다.

④ X
80은 동등 분할에서 통과 구간을 대표하는 값이지만 역시 경계값이 아니다. 80을 넣으면 명세대로 통과하므로 `>`와 `>=`의 차이가 드러나지 않는다. 틀린 표현은 `동등 분할 기법, 80`이고, 올바르게는 `경계값 분석 기법, 70`이다.

오답 포인트

학생이 0과 100만 보고 범위의 경계만 떠올리면 ②로 흔들리기 쉽다. 그러나 이 문제의 버그는 입력 범위 경계가 아니라 합격 판정 경계 70에서 발생한다. 또 50이나 80처럼 각 구간의 대표값은 동등 분할의 예시는 되지만, 비교 연산자의 포함 여부 오류는 잘 드러내지 못한다.

다시 나오면 이렇게 판단

명세에 `이상`, `이하`, `초과`, `미만` 같은 포함 조건이 보이면 먼저 분기 기준값을 찾는다. 구현 코드의 비교 연산자가 그 조건과 정확히 일치하는지 확인하고, 불일치 가능성이 있으면 대표값보다 기준값 자체를 넣는 경계값 분석을 우선 적용한다.

암기 포인트

동등 분할은 구간 대표값을 본다. 경계값 분석은 기준값 그 자체를 본다. `>= 70` 조건 오류는 70에서 잡힌다.

NoteBase

문 19. 유스케이스 분석 기법에서는 유스케이스를 바탕으로 세 가지 유형의 클래스를 도출한다. 다음 중에서 이에 해당하지 않는 클래스 유형은?

① 제어(control) 클래스

② 경계(boundary) 클래스

③ 싱글톤(singleton) 클래스

④ 엔티티(entity) 클래스

정답 3번

출제 포인트

이 문제는 유스케이스 분석에서 도출하는 분석 클래스의 3유형인 경계 클래스, 제어 클래스, 엔티티 클래스를 구별하는 문제이다.

핵심 개념

경계 클래스는 액터와 시스템의 접점을 담당하는 인터페이스 성격의 클래스이다. 키워드는 입력, 출력, 화면, 외부 연동이다.
제어 클래스는 유스케이스의 흐름과 처리 절차를 조정하는 클래스이다. 키워드는 이벤트 제어, 업무 흐름, 로직 조정이다.
엔티티 클래스는 시스템이 장기적으로 관리할 업무 데이터와 정보를 표현하는 클래스이다. 키워드는 저장 데이터, 업무 객체, 영속 정보이다.
싱글톤은 인스턴스를 하나만 생성하도록 제한하는 설계 패턴이다. 분석 클래스의 3유형이 아니라 구현 단계에서 쓰는 생성 패턴이다.

NoteBase

선지별 해설

① O
제어 클래스는 유스케이스 수행 순서를 조정하고 경계 클래스와 엔티티 클래스 사이의 처리를 통제하므로 유스케이스 분석에서 도출되는 클래스 유형에 해당한다.

② O
경계 클래스는 사용자나 외부 시스템과 맞닿는 경계를 담당하므로 유스케이스 분석에서 식별되는 대표적인 클래스 유형이다.

③ X
틀린 부분은 "싱글톤 클래스"라는 표현이다.
싱글톤은 분석 클래스 유형이 아니라 객체를 하나만 생성하도록 하는 설계 패턴이다. 올바르게는 유스케이스 분석의 3유형은 제어 클래스, 경계 클래스, 엔티티 클래스이다.

④ O
엔티티 클래스는 업무상 지속적으로 보관해야 할 데이터와 그 속성을 표현하므로 유스케이스 분석에서 도출되는 클래스 유형에 해당한다.

오답 포인트

싱글톤은 UML이나 객체지향 문맥에서 자주 보이는 용어이므로 클래스 유형처럼 보이기 쉽다. 그러나 이 문항은 설계 패턴이 아니라 유스케이스 분석 단계의 클래스 분류를 묻고 있으므로 분석 클래스 3유형만 떠올려야 한다.

다시 나오면 이렇게 판단

유스케이스 분석 클래스 문제는 먼저 사용자 접점이면 경계, 흐름 조정이면 제어, 업무 데이터 보관이면 엔티티로 분류한다. 보기 중 싱글톤, 팩토리 같은 패턴 용어가 나오면 분석 클래스가 아니라 설계 패턴인지부터 구분하면 된다.

암기 포인트

유스케이스 분석 3유형은 경계-제어-엔티티이며, 싱글톤은 클래스 유형이 아니라 설계 패턴이다.

NoteBase

문 20. 다음 Java 코드의 MyEmployeeService, EmployeeService, EmployeeDao, Employee 클래스를 UML 클래스 다이어그램으로 표현하였을 때 클래스 다이어그램에 나타나지 않는 관계는? (단, MyEmployeeService, EmployeeService, EmployeeDao, Employee 클래스는 이 외 다른 클래스와 아무 관계가 없다)

```
class MyEmployeeService extends EmployeeService{
     private EmployeeDao employeeDao;
     public void setEmployeeDao(EmployeeDao dao){
          this.employeeDao = dao;
     }
     public Employee getEmployee(String id){
          return employeeDao.get(id);
     }
     public void addEmployee(Employee emp){
          employeeDao.insert(emp);
     }
}
class EmployeeService { ... }
class EmployeeDao { ... }
class Employee { ... }
```

① 연관(association)

② 의존(dependency)

③ 일반화(generalization)

④ 실체화(realization)

정답 4번

출제 포인트

코드의 속성 선언, 메서드 시그니처, 상속 키워드를 보고 UML 클래스 다이어그램의 관계를 구분하는 문제이다.

핵심 개념

연관은 한 클래스가 다른 클래스의 객체를 속성으로 보유하는 관계이다.
의존은 매개변수, 반환형, 지역 사용처럼 다른 클래스를 사용하여 영향을 받는 관계이다.
일반화는 `extends`처럼 상위 클래스를 상속하는 관계이다.
실체화는 인터페이스나 추상 명세를 실제 클래스로 구현하는 관계이다.

NoteBase

선지별 해설

① O
`private EmployeeDao employeeDao;`처럼 MyEmployeeService가 EmployeeDao 객체를 멤버 필드로 보유하므로 연관 관계가 나타난다.

② O
`setEmployeeDao(EmployeeDao dao)`의 매개변수, `getEmployee(...)`의 반환형, `addEmployee(Employee emp)`의 매개변수처럼 다른 클래스를 사용하므로 의존 관계가 나타난다.

③ O
`class MyEmployeeService extends EmployeeService`가 상속 선언이므로 일반화 관계가 나타난다.

④ X
실체화는 인터페이스를 `implements`하거나 추상 명세를 구현할 때 나타나는 관계이다. 이 코드에는 인터페이스 구현이 없고 `extends`에 의한 일반화만 있다. 틀린 표현은 '실체화가 나타난다'이며, 올바르게는 '실체화는 나타나지 않는다'이다.

오답 포인트

일반화와 실체화는 모두 삼각형 화살표를 써서 헷갈리기 쉽다. 그러나 `extends`는 클래스 상속이므로 일반화이고, 실체화는 보통 인터페이스 구현이나 추상 명세의 구체화일 때만 성립한다.

다시 나오면 이렇게 판단

필드로 보유하면 연관, 매개변수나 반환형으로 쓰면 의존, `extends`이면 일반화, `implements`이면 실체화로 판단하면 된다.

암기 포인트

필드 보유는 연관, 시그니처 사용은 의존, `extends`는 일반화, `implements`는 실체화이다.

NoteBase