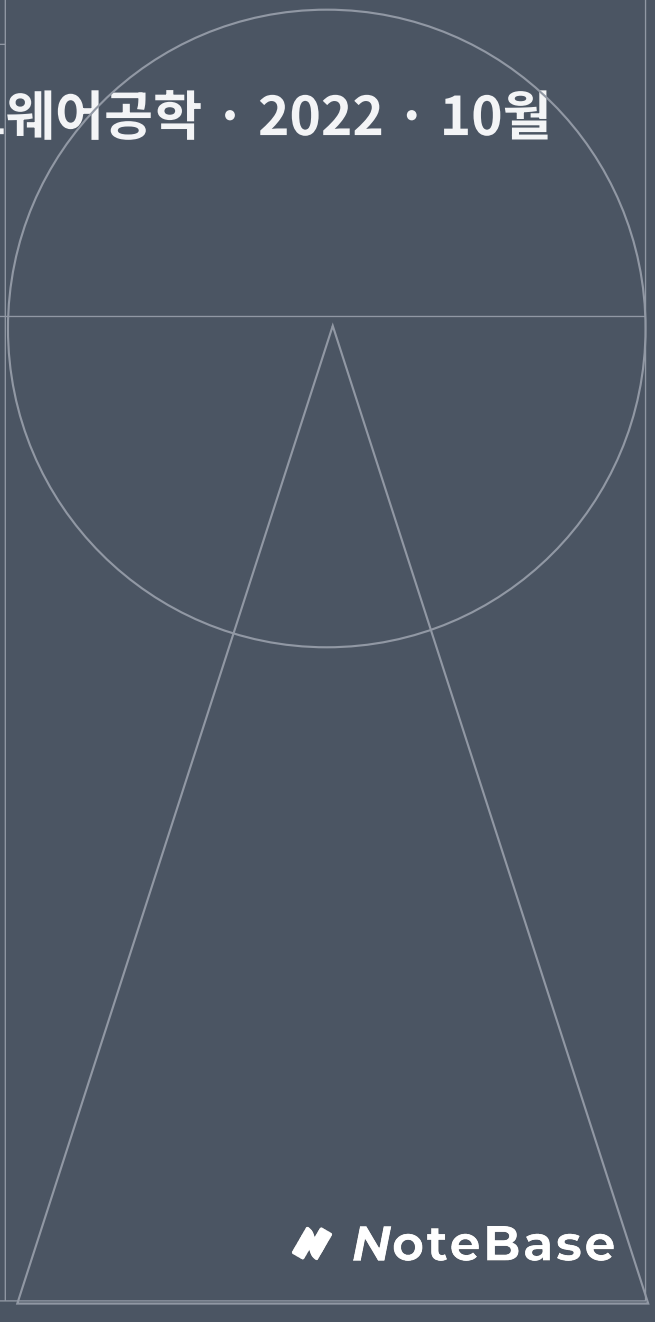
기출OX 해설집

공무원/서울시/7급 · 소프트웨어공학 · 2022 · 10월



NoteBase

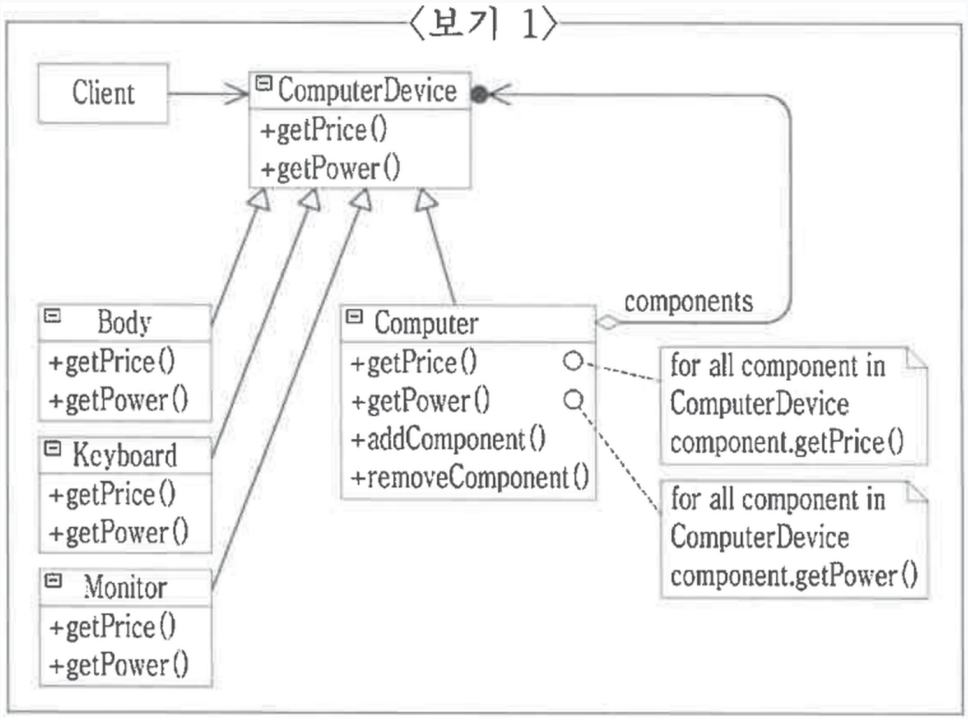
기출OX 해설집

공무원/서울시/7급 · 소프트웨어공학 · 2022 · 10월

공무원/서울시/7급 · 소프트웨어공학 · 2022 · 10월

문 1. <보기 1>은 컴포지트 디자인 패턴(composite design pattern)을 사용하여 컴퓨터의 가격 또는 전력 소모량을 계산할 목적으로 구성한 프로그램의 클래스 다이어그램이다. <보기 2>는 새로운 부품 VR 헤드셋을 컴퓨터에 추가하기 위해 이 프로그램을 컴포지트 디자인 패턴에 따라 수정한 때, 최종 프로그램에 대한 설명이다. <보기 2>에서 옳지 않은 것을 모두 고른 것은?

<보기 1>



<보기 2>

ㄱ. Body 클래스가 변경되었다.

ㄴ. Computer 클래스가 변경되었다.

ㄷ. ComputerDevice 클래스가 변경되었다.

① ㄱ

② ㄱ, ㄴ

③ ㄴ, ㄷ

④ ㄱ, ㄴ, ㄷ

NoteBase

정답 4번

출제 포인트

컴포지트 패턴에서 새로운 Leaf 객체를 추가할 때 기존 Component, Composite, 다른 Leaf를 수정하지 않는 확장 방식을 묻는 문제이다.

선지별 해설

① X: ㄱ만 고른 선택지이다. VR 헤드셋은 Body와 같은 새 Leaf 클래스로 추가하면 되므로 기존 Body 클래스는 변경되지 않는다. 틀린 표현은 "ㄱ만 옳지 않다"이고, 올바르게는 "ㄱ, ㄴ, ㄷ이 모두 옳지 않다"이다.
② X: ㄱ, ㄴ만 고른 선택지이다. Computer는 이미 ComputerDevice 타입의 components를 추가·삭제하고 순회 계산하는 구조이므로 VRHeadset도 같은 타입으로 넣으면 된다. 기존 Computer 클래스 수정은 필요 없다. 틀린 표현은 "ㄱ, ㄴ만 옳지 않다"이고, 올바르게는 "ㄱ, ㄴ, ㄷ이 모두 옳지 않다"이다.
③ X: ㄴ, ㄷ만 고른 선택지이다. ComputerDevice는 가격과 전력 계산을 위한 공통 연산을 이미 제공하므로 새 부품 추가 때문에 인터페이스 자체를 바꿀 이유가 없다. Body도 바뀌지 않으므로 틀린 표현은 "ㄴ, ㄷ만 옳지 않다"이고, 올바르게는 "ㄱ, ㄴ, ㄷ이 모두 옳지 않다"이다.
④ O: ㄱ, ㄴ, ㄷ을 모두 고른 선택지이다. 컴포지트 패턴에서는 새 부품을 추가할 때 기존 Leaf인 Body, Composite인 Computer, Component인 ComputerDevice를 수정하지 않고, 같은 인터페이스를 구현하는 VRHeadset 클래스를 새로 추가하는 방식으로 확장한다. 따라서 ㄱ, ㄴ, ㄷ은 모두 옳지 않다.

오답 포인트

새 부품이 들어오면 Composite나 Component도 함께 고쳐야 한다고 오해하기 쉽다. 그러나 이 도식은 Computer가 이미 ComputerDevice 컬렉션을 순회하도록 설계되어 있으므로, 새 부품은 같은 인터페이스를 구현하는 Leaf 추가로 끝난다.

다시 나오면 이렇게 판단

컴포지트 패턴 도식에서 공통 Component 인터페이스와 Composite의 자식 컬렉션이 이미 있으면, 새 부품 추가는 기존 클래스 변경인지 새 Leaf 추가인지를 먼저 본다. 메서드 시그니처와 자식 관리 방식이 그대로면 기존 Component, Composite, Leaf는 변경되지 않는다고 판단한다.

암기 포인트

Composite는 새 부품이 생겨도 기존 Component·Composite·기존 Leaf를 고치지 않고 새 Leaf를 추가하는 구조이다.

NoteBase

문 2. <보기>에서 설명하는 UP(Unified Process)의 단계 (가)~(다)를 옳게 짝지은 것은?

<보기>
(가) 소프트웨어 시스템을 개발 환경에서 사용자 환경으로 옮긴다.
(나) 핵심 아키텍처를 구축하고 대부분의 요구사항을 명확히 정의한다.
(다) 주요 요구사항들을 나열하고 전체적으로 10% 정도의 유스케이스(use case)를 상세히 작성한다.

① (가): 전이(transition), (나): 정련(elaboration), (다): 도입(inception)
② (가): 도입(inception), (나): 구축(construction), (다): 정련(elaboration)
③ (가): 전이(transition), (나): 구축(construction), (다): 정련(elaboration)
④ (가): 구축(construction), (나): 전이(transition), (다): 도입(inception)

정답 1번

출제 포인트

이 문제는 UP의 4단계인 도입, 정련, 구축, 전이의 목적을 구분하는 문제이다.

핵심 개념

도입은 프로젝트 범위와 주요 요구사항을 정리하고 전체 유스케이스 중 일부만 상세화하는 단계이다.
정련은 핵심 아키텍처를 확립하고 대부분의 요구사항을 구체화하는 단계이다.
구축은 구현과 테스트를 통해 실행 가능한 제품을 완성하는 단계이다.
전이는 개발된 시스템을 실제 사용자 환경으로 배포하는 단계이다.

NoteBase

선지별 해설

① O
(가)는 개발 환경에서 사용자 환경으로 시스템을 옮기는 배포 단계이므로 전이이다.
(나)는 핵심 아키텍처를 구축하고 대부분의 요구사항을 명확히 하는 단계이므로 정련이다.
(다)는 주요 요구사항을 나열하고 약 10% 정도의 유스케이스만 상세히 작성하는 초기 단계이므로 도입이다.
② X
틀린 부분은 (가)를 도입, (나)를 구축, (다)를 정련으로 대응한 점이다.
올바르게는 (가) 전이, (나) 정련, (다) 도입이다.
③ X
틀린 부분은 (나)를 구축, (다)를 정련으로 대응한 점이다.
올바르게는 (나) 정련, (다) 도입이다. 구축은 아키텍처 확립이 아니라 구현과 테스트가 중심인 단계이다.
④ X
틀린 부분은 (가)를 구축, (나)를 전이로 대응한 점이다.
올바르게는 (가) 전이, (나) 정련, (다) 도입이다.

오답 포인트

정련과 구축을 혼동하면 아키텍처 확립 단계와 실제 구현 단계를 뒤바꾸기 쉽다. 또한 도입은 초기 범위 설정과 일부 유스케이스 상세화 단계라는 점을 놓치면 (다)를 정련으로 오인하기 쉽다.

다시 나오면 이렇게 판단

배포와 사용자 환경 이전이 보이면 전이, 핵심 아키텍처와 대부분 요구사항 확정이 보이면 정련, 초기 범위 설정과 일부 유스케이스 상세화가 보이면 도입, 실제 구현과 테스트가 보이면 구축으로 판단한다.

암기 포인트

도입은 범위와 일부 유스케이스, 정련은 아키텍처와 대부분 요구사항, 구축은 구현, 전이는 사용자 배포이다.

NoteBase

문 3. 프로젝트 계획을 세울 때 적용하는 기법들을 순서대로 바르게 나열한 것은?

① CPM - Gantt Chart - WBS

② WBS - CPM - Gantt Chart

③ CPM - WBS - Gantt Chart

④ WBS - Gantt Chart - CPM

정답 2번

출제 포인트

이 문제는 프로젝트 계획 수립에서 WBS, CPM, Gantt Chart를 어떤 흐름으로 적용하는지를 묻는 문제이다.

올바른 순서

1. WBS - 프로젝트 범위와 산출물을 작업 단위로 분해한다.
2. CPM - 분해된 작업들의 선후관계와 주공정을 분석한다.
3. Gantt Chart - 분석된 작업 일정과 기간을 막대 형태로 시각화한다.

왜 이 순서: CPM은 WBS로 작업 목록이 먼저 정의되어야 적용할 수 있고, Gantt Chart는 작업 기간과 선후관계가 정리된 뒤에 일정표로 표현할 수 있다.

선지별 해설

1번 X: 틀린 부분은 CPM을 WBS보다 먼저 둔 점이다. CPM은 작업 분해 결과와 선후관계 대상 작업이 있어야 계산되므로 WBS 뒤에 와야 한다. 올바르게는 WBS - CPM - Gantt Chart이다.

2번 O: WBS로 작업을 세분화한 뒤 CPM으로 작업 간 의존관계와 주공정을 구하고, 마지막에 Gantt Chart로 일정표를 표현하므로 순서가 맞다.

3번 X: 틀린 부분은 CPM을 WBS보다 먼저 둔 점이다. 작업을 정의하기 전에는 주공정을 분석할 수 없다. 올바르게는 WBS를 먼저 수행한 뒤 CPM을 적용하고 마지막에 Gantt Chart를 작성한다.

4번 X: 틀린 부분은 Gantt Chart를 CPM보다 먼저 둔 점이다. Gantt Chart는 일정 표현 도구이고, CPM은 그 전에 필요한 선후관계 분석 기법이다. 올바르게는 WBS - CPM - Gantt Chart이다.

오답 포인트

WBS, CPM, Gantt Chart를 모두 일정관리 기법으로만 묶어 외우면 순서를 헷갈리기 쉽다. WBS는 작업 분해, CPM은 분석, Gantt Chart는 시각화라는 역할 차이를 구분해야 한다.

다시 나오면 이렇게 판단

프로젝트 계획 절차 문제는 먼저 작업을 정의하는 단계가 무엇인지 찾고, 그다음 선후관계를 분석하는 단계, 마지막으로 결과를 일정표로 표현하는 단계를 순서대로 배치하면 된다.

NoteBase

암기 포인트

작업 분해는 WBS, 경로 분석은 CPM, 일정 표현은 Gantt Chart이다.

NoteBase

문 4. 소프트웨어 요구사항 명세서(Software Requirement Specification)의 항목에 해당하지 않는 것은?

① 사용자 요구사항 정의
② 시스템 아키텍처
③ 인터페이스 요구사항
④ 소프트웨어 개발 일정

정답 4번

출제 포인트

이 문제는 소프트웨어 요구사항 명세서에 들어가는 요구사항·구조 기술과 프로젝트 관리 항목을 구분할 수 있는지를 묻는다.

핵심 개념

소프트웨어 요구사항 명세서(SRS)는 소프트웨어가 무엇을 해야 하는지와 어떤 제약을 가지는지를 문서화한 것이다. 판단 기준은 제품의 기능·인터페이스·구조를 설명하는 항목인지, 아니면 개발 일정·인력·공정처럼 프로젝트 수행 계획을 설명하는 항목인지이다. 사용자 요구사항 정의, 시스템 아키텍처, 인터페이스 요구사항은 SRS 쪽이고, 소프트웨어 개발 일정은 프로젝트 계획 쪽이다.

선지별 해설

1번 O: 사용자 요구사항 정의는 사용자가 필요로 하는 기능과 제약을 기술하는 내용이므로 SRS에 포함된다.
2번 O: 시스템 아키텍처는 시스템의 상위 구조와 구성 요소 관계를 설명하는 항목으로 요구사항 명세 문서의 구성 요소로 볼 수 있다.
3번 O: 인터페이스 요구사항은 사용자 인터페이스나 외부 시스템과의 연결 조건을 명세하는 요구사항이므로 SRS의 핵심 항목이다.
4번 X: 틀린 부분은 "소프트웨어 개발 일정"이다. 개발 일정은 마일스톤, 기간, 공정 계획을 다루는 프로젝트 관리 산출물이다. 올바르게는 SRS 항목이 아니라 프로젝트 계획서나 개발 계획에 포함된다고 보아야 한다.

오답 포인트

시스템 아키텍처를 설계 산출물로만 외우면 2번을 제외하고 싶어지지만, 이 문제는 SRS 문서의 항목 구성을 묻고 있다. 반대로 4번의 개발 일정은 문서 목차처럼 보이더라도 제품 요구사항이 아니라 프로젝트 관리 정보라는 점이 함정이다.

NoteBase

다시 나오면 이렇게 판단

선지가 소프트웨어가 제공해야 할 기능, 외부와의 인터페이스, 시스템의 상위 구조를 설명하면 SRS 항목으로 판단한다. 선지가 언제까지 무엇을 만들지 같은 일정·공정·자원 배분을 말하면 프로젝트 계획 항목으로 판단한다.

암기 포인트

SRS는 기능·인터페이스·구조를 적는 문서이고, 일정은 프로젝트 계획에 적는 문서이다.

NoteBase

문 5. 기능적 요구사항(functional requirement)과 비기능적 요구사항(non-functional requirement)에 대한 설명으로 가장 옳지 않은 것은?

① 비기능적 요구사항은 기능적 요구사항보다 구현해야 하는 우선순위가 낮은 기능에 대한 요구사항을 포함한다.

② 동일한 소프트웨어에 대해서 여러 종류의 사용자가 존재할 수 있으며, 사용자 종류에 따라 별도의 기능적 요구사항을 가질 수 있다.

③ 기능적 요구사항은 소프트웨어 시스템을 통해 사용자가 얻고자 하는 서비스에 대한 조건으로, 사용자 입력에 따른 기대 출력에 대한 조건으로 표현할 수 있다.

④ 보안 요구사항, 성능 요구사항은 비기능적 요구사항이다.

정답 1번

출제 포인트

이 문제는 기능적 요구사항과 비기능적 요구사항의 구분, 특히 비기능적 요구사항을 낮은 우선순위 기능으로 오해하는지를 묻는 문제이다.

핵심 개념

기능적 요구사항은 시스템이 제공해야 하는 기능, 서비스, 입력에 대한 처리와 출력 등 행위를 설명하는 요구사항이다.
비기능적 요구사항은 시스템이 그 기능을 어떤 품질 수준과 제약 조건 아래에서 제공해야 하는지를 설명하는 요구사항이다.
구별 축은 '무엇을 하는가'와 '어떤 품질·제약으로 하는가'이다.

선지별 해설

① X. 비기능적 요구사항은 기능적 요구사항보다 우선순위가 낮은 기능 요구사항이 아니다. 성능, 보안, 신뢰성, 사용성, 확장성처럼 시스템 전체나 기능 수행 방식에 적용되는 품질 속성과 제약 조건이다. 틀린 표현은 '우선순위가 낮은 기능에 대한 요구사항'이다. 올바르게는 '기능이 충족해야 할 품질 속성이나 제약 조건에 대한 요구사항'이다.
② O. 동일한 소프트웨어라도 관리자, 일반 사용자, 외부 연계 시스템 사용자처럼 사용자 유형이 다르면 필요한 기능이 달라질 수 있다. 따라서 사용자 종류별로 별도의 기능적 요구사항을 가질 수 있다.
③ O. 기능적 요구사항은 사용자가 시스템을 통해 얻으려는 서비스와 시스템의 행위를 설명한다. 입력에 대해 어떤 처리와 기대 출력을 제공해야 하는지로 표현할 수 있으므로 기능적 요구사항의 설명으로 옳다.
④ O. 보안 요구사항과 성능 요구사항은 기능 자체의 목록이라기보다 기능 수행 시 지켜야 할 품질 속성이다. 따라서 비기능적 요구사항에 해당한다.

NoteBase

오답 포인트

출제자가 노린 함정은 '비기능'이라는 말을 '덜 중요한 기능'이나 '나중에 구현할 기능'으로 착각하게 만드는 ①의 표현이다. 비기능적 요구사항은 우선순위 분류가 아니라 품질 속성과 제약 조건의 분류이다.

다시 나오면 이렇게 판단

요구사항이 로그인, 조회, 저장, 출력처럼 시스템이 해야 할 행위를 말하면 기능적 요구사항으로 판단한다.
요구사항이 응답시간, 처리량, 보안, 신뢰성, 사용성처럼 기능의 품질이나 제약을 말하면 비기능적 요구사항으로 판단한다.

암기 포인트

기능적 요구사항은 '무엇을 할 것인가'이고, 비기능적 요구사항은 '얼마나 잘, 어떤 조건에서 할 것인가'이다.

NoteBase

문 6. 추상화와 구현을 분리하여, 실행시간에 동적으로 인터페이스의 다른 구현들을 사용할 수 있는 디자인 패턴은?

① adapter 패턴

② bridge 패턴

③ proxy 패턴

④ facade 패턴

정답 2번

출제 포인트

이 문제는 추상화와 구현을 분리하여 둘을 독립적으로 확장하고 실행시간에 구현 객체를 교체할 수 있게 하는 Bridge 패턴의 핵심 개념을 묻는 문제이다.

핵심 개념

Bridge 패턴은 추상화 계층과 구현 계층을 분리하여 두 축을 독립적으로 변화시킬 수 있게 하는 패턴이다. 키워드는 추상화-구현 분리와 런타임 구현 교체이다.
Adapter 패턴은 기존 인터페이스가 맞지 않는 클래스를 원하는 인터페이스로 변환하는 패턴이다. 키워드는 인터페이스 변환이다.
Proxy 패턴은 실제 객체에 대한 대리자를 두어 접근 제어, 지연 로딩, 부가 기능을 제공하는 패턴이다. 키워드는 대리 접근이다.
Facade 패턴은 복잡한 서브시스템에 대한 단순한 통합 인터페이스를 제공하는 패턴이다. 키워드는 단순화된 창구이다.

선지별 해설

① X: adapter 패턴은 호환되지 않는 인터페이스를 변환하는 데 목적이 있다. 틀린 부분은 문제의 핵심인 "추상화와 구현을 분리"에 해당하지 않는 점이다. 올바르게는 "기존 인터페이스를 다른 인터페이스로 맞춰 연결하는 패턴"이다.
② O: bridge 패턴은 추상화와 구현을 분리하여 각각 독립적으로 확장할 수 있고, 구현 객체를 실행시간에 바꿔 끼울 수 있으므로 문제의 설명과 일치한다.
③ X: proxy 패턴은 실제 객체 대신 대리 객체를 두어 접근을 통제하거나 부가 기능을 붙이는 데 목적이 있다. 틀린 부분은 "다른 구현들을 사용할 수 있는 구조"로 본 점이다. 올바르게는 "대리 객체를 통한 접근 제어 패턴"이다.
④ X: facade 패턴은 복잡한 서브시스템을 감추고 단순한 인터페이스를 제공하는 패턴이다. 틀린 부분은 이를 추상화와 구현의 분리 구조로 본 점이다. 올바르게는 "서브시스템 사용을 쉽게 만드는 단순화 인터페이스 패턴"이다.

NoteBase

오답 포인트

adapter, bridge, facade는 모두 구조 패턴이어서 이름만 보고 혼동하기 쉽다. 특히 adapter를 "연결해 준다"는 이미지로 bridge와 섞어 오답 처리하기 쉽지만, adapter는 인터페이스 변환이고 bridge는 추상화와 구현의 분리라는 구조 차이가 있다.

다시 나오면 이렇게 판단

문장에서 "추상화와 구현의 분리", "독립적 확장", "실행시간 구현 교체"가 보이면 bridge를 우선 판단한다. "호환되지 않는 인터페이스 연결"이면 adapter, "대리 접근"이면 proxy, "복잡한 기능을 감춘 단순 창구"이면 facade로 구분된다.

암기 포인트

Bridge는 분리, Adapter는 변환, Proxy는 대리, Facade는 단순화이다.

NoteBase

문 7. <보기>는 MyIF 인터페이스를 정의하고 이를 MyImpl 클래스로 구현한 자바 코드와 이들의 관계를 UML로 표현한 클래스 다이어그램이다. (가), (나)에 들어갈 내용과 (다)에 들어갈 그림을 옳게 짝지은 것은?

<보기>

- 코드

```
(가) MyIF { … }
Class MyImpl (나) MyIF { … }
```

- UML 표기

(다)

① abstract extends



② interface extends



③ interface implements



④ interface implements



정답 3번

출제 포인트

자바에서 인터페이스를 선언하는 키워드와 클래스로 인터페이스를 구현하는 키워드, 그리고 이를 UML realization 관계로 읽는지를 묻는 문제이다.

NoteBase

선지별 해설

① X: (가)에 abstract가 들어가면 추상 클래스 선언이 되므로 인터페이스 선언이 아니다. MyIF는 interface로 선언해야 하고, 클래스가 인터페이스를 받을 때도 extends가 아니라 implements를 사용한다. 그림 역시 구현 관계의 속 빈 삼각형이 아니라 단순 점선 화살표 형태이므로 UML 구현 표기와 맞지 않는다. 틀린 표현은 'abstract, extends'이며 올바르게는 'interface, implements'이다.
② X: (가)의 interface는 맞지만 (나)의 extends가 틀리다. 클래스가 인터페이스를 구현할 때는 class MyImpl implements MyIF 형태가 되어야 한다. UML도 실선과 속 빈 삼각형은 일반화, 즉 상속 관계를 뜻하므로 인터페이스 구현 관계와 다르다. 틀린 표현은 'extends'와 실선 일반화 표기이며 올바르게는 'implements'와 점선 realization 표기이다.
③ O: interface MyIF { ... }, class MyImpl implements MyIF { ... }가 자바 문법에 맞는다. UML에서도 MyImpl에서 MyIF 쪽으로 향하는 점선과 속 빈 삼각형은 인터페이스 구현, 즉 realization 관계를 뜻하므로 코드와 그림이 정확히 대응한다.
④ X: (가)의 interface와 (나)의 implements는 맞지만 UML 화살표 방향이 반대이다. 구현 관계의 삼각형은 구현 클래스가 아니라 인터페이스 쪽을 향해야 한다. 틀린 표현은 화살표 방향이며 올바르게는 MyImpl에서 출발해 MyIF를 향하는 점선 속 빈 삼각형이어야 한다.

오답 포인트

인터페이스도 상위 타입이라는 이유로 extends를 떠올리거나, UML에서 상속과 구현을 모두 삼각형으로만 외워 실선과 점선을 구분하지 못하면 틀리기 쉽다. 특히 ④처럼 코드 빈칸은 맞고 화살표 방향만 뒤집은 선택지가 대표적인 함정이다.

다시 나오면 이렇게 판단

코드에서는 인터페이스 선언이면 interface, 클래스의 인터페이스 수용이면 implements인지 먼저 본다. UML에서는 실선 삼각형이면 상속, 점선 일반 화살표면 의존, 점선 속 빈 삼각형이면 인터페이스 구현으로 판단하면 된다.

암기 포인트

인터페이스는 interface, 구현은 implements, UML 구현 관계는 점선 삼각형이 인터페이스를 향한다.

NoteBase

문 8. 블랙박스 테스트 기법에 대한 설명으로 가장 옳은 것은?

① 경곗값 분석: 테스트 대상의 입출력값을 특정 클래스로 분할한 후 각각의 클래스로부터 대표값을 추출하여 테스트 케이스를 생성하고 테스트 데이터의 범주를 식별한다.

② 인과 그래프: 입력값 조합의 경우의 수를 줄이기 위해 두 입력값의 조합을 통해 테스트 케이스를 도출하는 기법으로, 모든 경우의 수를 테스트하지 않고 2개 요소의 모든 조합을 확인할 수 있도록 테스트 케이스를 생성한다.

③ 페어와이즈 조합: 입력 데이터 간 관계가 출력에 영향을 미치는 상황을 체계적으로 분석하여 테스트 케이스를 생성한다.

④ 결정 테이블: 의사 결정의 여러 조합을 정하는 행위나 결과를 식별함으로써 시스템의 예상되는 행위를 파악하여 테스트 케이스를 생성한다.

정답 4번

출제 포인트

이 문제는 블랙박스 테스트 기법인 등가분할, 경곗값 분석, 인과 그래프, 페어와이즈 조합, 결정 테이블의 핵심 정의와 적용 목적을 구분하는 문제이다.

핵심 개념

경곗값 분석은 입력 범위의 경계와 그 주변 값을 중점적으로 시험하는 기법이다. 등가분할은 입력값을 같은 성질의 클래스로 나눈 뒤 각 클래스의 대표값을 뽑는 기법이다. 인과 그래프는 입력 조건과 출력 결과의 논리 관계를 그래프로 표현해 테스트 케이스를 도출하는 기법이다. 페어와이즈 조합은 여러 입력 요소 중 두 요소씩의 모든 조합이 한 번 이상 나타나도록 케이스를 줄이는 기법이다. 결정 테이블은 조건 조합과 그에 따른 행위 또는 결과를 표 형태로 정리해 테스트 케이스를 도출하는 기법이다.

NoteBase

선지별 해설

① X. 선지의 설명은 입력값을 특정 클래스로 분할하고 각 클래스의 대표값을 뽑는 방식이므로 등가분할에 해당한다. 틀린 부분은 "경곗값 분석"이다. 올바르게는 "등가분할은 테스트 대상의 입출력값을 특정 클래스로 분할한 후 각 클래스의 대표값을 추출하여 테스트 케이스를 생성한다"이다.
② X. 선지의 설명은 두 입력값의 조합을 통해 경우의 수를 줄이면서 2개 요소의 모든 조합을 확인하는 방식이므로 페어와이즈 조합에 해당한다. 틀린 부분은 "인과 그래프"이다. 올바르게는 "페어와이즈 조합은 모든 경우를 다 시험하지 않고도 두 요소의 모든 조합이 포함되도록 테스트 케이스를 생성한다"이다.
③ X. 선지의 설명은 입력 데이터 간 관계가 출력에 영향을 미치는 상황을 분석하는 방식이므로 인과 그래프에 해당한다. 틀린 부분은 "페어와이즈 조합"이다. 올바르게는 "인과 그래프는 입력 조건과 출력 결과의 인과 관계를 체계적으로 분석하여 테스트 케이스를 생성한다"이다.
④ O. 결정 테이블은 여러 조건 조합과 그에 대응하는 행위나 결과를 정리하여 시스템의 예상 동작을 확인하는 기법이다. 선지는 조건 조합과 결과를 식별해 테스트 케이스를 만든다는 점을 정확히 설명하므로 옳다.

오답 포인트

이 문항의 함정은 기법 이름만 바꾸어 붙인 데 있다. ①의 "특정 클래스로 분할"은 등가분할의 표현이고, ②의 "두 입력값의 조합"은 페어와이즈 조합의 표현이며, ③의 "입력과 출력의 관계 분석"은 인과 그래프의 표현이다. 정의의 핵심 키워드를 못 잡으면 모두 그럴듯하게 보이기 쉽다.

다시 나오면 이렇게 판단

선지를 읽을 때 먼저 핵심 키워드를 잡으면 된다. 경계값이 나오면 경곗값 분석, 클래스 분할이 나오면 등가분할, 입력과 출력의 논리 관계가 나오면 인과 그래프, 두 요소 조합 최소화가 나오면 페어와이즈 조합, 조건 조합과 결과 정리가 나오면 결정 테이블로 판단한다.

암기 포인트

경계는 경곗값 분석, 클래스는 등가분할, 인과는 인과 그래프, 두 개씩은 페어와이즈, 조건-행동표는 결정 테이블이다.

NoteBase

문 9. 린(Lean) 소프트웨어 개발의 7가지 원칙에 해당하는 것을 <보기>에서 모두 고른 것은?

<보기>
ㄱ. 낭비를 제거하라
ㄴ. 학습을 확대하라
ㄷ. 최대한 빨리 배포하라
ㄹ. 최대한 빨리 결정하라
ㅁ. 팀장에게 권한을 집중시켜라

① ㄱ, ㄴ, ㄷ
② ㄱ, ㄷ, ㄹ
③ ㄴ, ㄹ, ㅁ
④ ㄱ, ㄴ, ㄷ, ㄹ

정답 1번

출제 포인트

이 문제는 린 소프트웨어 개발의 7가지 원칙을 정확히 구별하는지를 묻는 문제이다.

핵심 개념

린 소프트웨어 개발은 제조업의 린 사고를 소프트웨어에 적용한 개발 원칙이다.
낭비 제거는 불필요한 기능, 대기, 재작업처럼 가치가 없는 활동을 줄이는 원칙이다.
학습 확대는 반복적 개발과 피드백을 통해 지식을 빠르게 축적하는 원칙이다.
빠른 배포는 Deliver as fast as possible에 해당하며 고객 피드백을 앞당기는 원칙이다.
의사결정은 가능한 한 늦게, 충분한 정보가 모였을 때 내리는 것이 원칙이다.
권한은 팀장에게 집중하는 것이 아니라 팀 전체에 위임하여 자율성을 높이는 것이 원칙이다.

NoteBase

선지별 해설

① O
- ㄱ 낭비를 제거하라, ㄴ 학습을 확대하라, ㄷ 최대한 빨리 배포하라는 모두 린 소프트웨어 개발의 원칙에 해당한다.
- 보기 안에 틀린 진술이 포함되지 않으므로 ①이 정답이다.

② X
- ②에는 ㄹ 최대한 빨리 결정하라가 포함되어 있어 틀리다.
- 틀린 표현은 최대한 빨리 결정하라이다.
- 올바르게는 가능한 한 늦게 결정하라 또는 충분한 정보를 얻은 뒤 결정하라이다.

③ X
- ③에는 ㄹ과 ㅁ이 포함되어 있어 틀리다.
- ㄹ의 틀린 표현은 최대한 빨리 결정하라이다. 올바르게는 가능한 한 늦게 결정하라이다.
- ㅁ의 틀린 표현은 팀장에게 권한을 집중시켜라이다. 올바르게는 팀에 권한을 부여하라 또는 팀을 empowered 상태로 두어라이다.

④ X
- ④도 ㄹ이 포함되어 있어 틀리다.
- 틀린 표현은 최대한 빨리 결정하라이다.
- 올바르게는 가능한 한 늦게 결정하라이다.

오답 포인트

린 개발은 빠른 흐름을 중시하므로 ㄷ의 빠른 배포와 ㄹ의 빠른 결정이 비슷하게 보이는 함정이 있다. 그러나 린은 배포는 빠르게 하되 의사결정은 정보를 더 확보할 때까지 늦추는 쪽에 가깝다. 또 ㅁ처럼 리더에게 권한을 모으는 방식은 전통적 통제와 가깝고 린의 팀 권한 위임 원칙과 반대이다.

다시 나오면 이렇게 판단

린 원칙 문제는 제거할 것, 늘릴 것, 늦출 것, 빠르게 할 것을 구분하면 풀린다. 낭비는 제거, 학습은 확대, 배포는 빠르게, 결정은 늦게, 권한은 팀에 부여라고 대응시키면 된다.

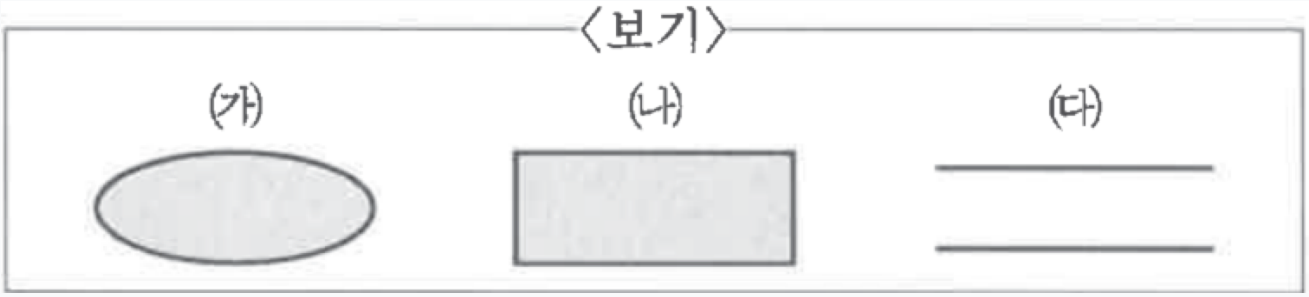
암기 포인트

린 개발은 낭비 제거, 학습 확대, 빠른 배포, 늦은 결정, 팀 권한 부여로 정리된다.

NoteBase

문 10. <보기>는 데이터 흐름도에서 사용하는 기호들이다. (가)~(다) 기호의 의미를 옳게 짝지은 것은?

<보기>



① (가): 프로세스, (나): 데이터 저장소, (다): 외부 개체

② (가): 프로세스, (나): 외부 개체, (다): 데이터 저장소

③ (가): 외부 개체, (나): 데이터 저장소, (다): 프로세스

④ (가): 외부 개체, (나): 프로세스, (다): 데이터 저장소

정답 2번

출제 포인트

이 문제는 데이터 흐름도에서 프로세스, 외부 개체, 데이터 저장소의 기본 기호를 구별할 수 있는지를 묻는다.

선지별 해설

1번: X. (가)를 프로세스로 본 것은 맞지만 (나)를 데이터 저장소로, (다)를 외부 개체로 본 부분이 틀리다. 사각형은 데이터 저장소가 아니라 외부 개체이고, 평행한 두 줄은 외부 개체가 아니라 데이터 저장소이다.
2번: O. (가)=프로세스, (나)=외부 개체, (다)=데이터 저장소의 대응이 데이터 흐름도 기본 기호와 정확히 일치한다.
3번: X. (가)를 외부 개체로 본 부분이 틀리다. 타원형은 외부 개체가 아니라 프로세스이다. 또한 (다)를 프로세스로 본 부분도 틀리다. 평행한 두 줄은 프로세스가 아니라 데이터 저장소이다.
4번: X. (가)를 외부 개체로 본 부분이 틀리다. 타원형은 외부 개체가 아니라 프로세스이다. 또한 (나)를 프로세스로 본 부분도 틀리다. 사각형은 프로세스가 아니라 외부 개체이다.

오답 포인트

사각형과 평행한 두 줄을 모두 저장 관련 기호처럼 받아들이면 (나)와 (다)를 뒤바꾸기 쉽다. DFD에서는 닫힌 사각형은 외부 개체이고, 두 줄 기호가 데이터 저장소라는 점이 함정이다.

NoteBase

다시 나오면 이렇게 판단

기호의 외곽 형태를 먼저 본다. 두 줄이면 데이터 저장소, 닫힌 사각형이면 외부 개체, 원형·타원형이면 프로세스로 판단한다.

암기 포인트

DFD는 버블형=프로세스, 박스형=외부 개체, 평행선형=데이터 저장소이다.

NoteBase

문 11. 관점지향 프로그래밍(Aspect Oriented Programming)에서 횡단 관심 모듈의 기능이 삽입되어 동작할 수 있는 실행 가능한 특정 위치를 의미하는 용어는?

① 어드바이스(advice)

② 위빙(weaving)

③ 조인 포인트(join point)

④ 포인트 컷(point-cut)

정답 3번

출제 포인트

이 문제는 AOP의 핵심 용어인 advice, weaving, join point, pointcut의 정의를 구분하는 문제이다.

핵심 개념

조인 포인트는 프로그램 실행 과정에서 부가 기능이 들어갈 수 있는 구체적 실행 지점이다.
포인트컷은 여러 조인 포인트 중 실제로 적용할 대상을 골라내는 조건식 또는 규칙이다.
어드바이스는 조인 포인트에서 실행되는 횡단 관심 기능 자체이다.
위빙은 어드바이스와 핵심 모듈을 결합하여 동작하게 만드는 과정이다.

선지별 해설

① X: 어드바이스는 횡단 관심을 구현한 실제 부가 기능 코드이다. 틀린 부분은 '실행 가능한 특정 위치'라는 표현이다. 올바르게는 조인 포인트에서 실행되는 동작 또는 처리 로직이다.
② X: 위빙은 aspect를 기존 프로그램 코드에 결합하는 과정이다. 틀린 부분은 '특정 위치'라는 표현이다. 올바르게는 횡단 관심 모듈을 핵심 관심 모듈에 삽입해 통합하는 절차이다.
③ O: 조인 포인트는 메서드 호출, 메서드 실행, 예외 처리 같은 프로그램의 실행 지점을 뜻하므로 문제의 설명과 일치한다.
④ X: 포인트컷은 조인 포인트 자체가 아니라 어떤 조인 포인트에 어드바이스를 적용할지 선택하는 표현식이다. 틀린 부분은 위치 그 자체로 본 점이다. 올바르게는 조인 포인트를 선별하는 규칙이다.

오답 포인트

조인 포인트와 포인트컷을 같은 의미로 섞어 외우면 틀리기 쉽다. 이 문항은 '실행 가능한 특정 위치'라는 표현으로 조인 포인트를 묻고 있는데, 많은 수험생이 그 위치를 고르는 규칙인 포인트컷으로 착각한다.

다시 나오면 이렇게 판단

문장에서 '위치'를 묻으면 조인 포인트를 먼저 떠올리면 된다. 그 위치를 고르는 조건이면 포인트컷, 그 위치에서 실행되는 코드면 어드바이스, 둘을 결합하는 절차면 위빙으로 판단한다.

NoteBase

암기 포인트

위치는 조인 포인트, 조건은 포인트컷, 코드는 어드바이스, 결합은 위빙이다.

NoteBase

문 12. 상태 디자인 패턴(state design pattern)을 적용하여 전등(Light)의 상태를 관리하는 상태 클래스들을 <보기 1>과 같이 Java 프로그램으로 구현하였다. <보기 2>와 같이 취침등(SLEEP) 상태를 추가하여 전등의 동작을 변경하고자 한다. 상태 디자인 패턴에 따라 작성한 최종 코드에 대한 설명으로 가장 옳지 않은 것은?

〈보기 1〉

REQ-1) 전등이 꺼진 상태에서 on 버튼을 누르면 켜진다.
REQ-2) 전등이 켜진 상태에서 off 버튼을 누르면 꺼진다.

```
public class State {
   public void on_button(Light light) {}
   public void off_button(Light light) {}
}

public class ON implements State {
   public void off_button(Light light) {
        light.setState(new OFF());
   }
}

public class OFF implements State {
   public void on_button(Light light) {
        light.setState(new ON());
   }
}
```

<보기 2>

REQ-3) 전등이 켜진 상태에서 on 버튼을 누르면 취침등 상태가 된다.

REQ-4) 취침등 상태에서 on 버튼을 누르면 다시 전등이 켜진다.

REQ-5) 취침등 상태에서 off 버튼을 누르면 꺼진다.

① ON 상태 클래스가 변경되었다.

② OFF 상태 클래스가 변경되었다.

③ SLEEP 상태 클래스가 추가되었다.

④ Light 클래스는 변경되지 않았다.

정답 2번

출제 포인트

이 문제는 상태 패턴에서 새로운 상태와 전이 요구사항이 추가될 때 어떤 상태 클래스와 컨텍스트 클래스가 실제로 변경되는지를 묻는 문제이다.

NoteBase

사례 정리

기존 요구사항에서는 OFF 상태에서 on 버튼을 누르면 ON으로, ON 상태에서 off 버튼을 누르면 OFF로 전이된다.
추가 요구사항에서는 ON 상태에서 on 버튼을 누르면 SLEEP으로, SLEEP 상태에서 on 버튼을 누르면 ON으로, SLEEP 상태에서 off 버튼을 누르면 OFF로 전이된다.
판단에 중요한 조건은 새로 생긴 전이의 출발 상태가 ON과 SLEEP뿐이며 OFF에서의 동작 요구는 추가되지 않았다는 점이다.

핵심 개념

상태 패턴은 상태별 행동을 각 상태 클래스에 캡슐화하고, 상태 전이가 바뀌면 해당 전이를 담당하는 상태 클래스만 수정하거나 새 상태 클래스를 추가하는 구조이다.
적용 요건은 컨텍스트인 Light가 현재 상태 객체에 처리를 위임하고, 각 상태 클래스가 자신의 입력 이벤트에 대한 전이만 책임지는 것이다.

적용 기준

이 사례에서는 새 요구사항이 어느 상태에서 어떤 버튼 입력을 받을 때 발생하는지 기준으로 변경 대상을 판단해야 한다.
핵심은 새 전이의 출발 상태가 ON인지, SLEEP인지, OFF인지를 따지는 것이다.

NoteBase

선지별 해설

① O
ON 상태에서는 기존에 on_button 동작이 정의되지 않았지만 추가 요구사항으로 on 버튼 입력 시 SLEEP으로 전이해야 하므로 ON 상태 클래스는 변경된다.

② X
OFF 상태에는 새 요구사항이 추가되지 않았다. 기존 요구사항인 OFF에서 on 버튼을 누르면 ON으로 전이하는 동작은 그대로 유지된다. 틀린 표현은 "OFF 상태 클래스가 변경되었다"이고, 올바르게는 "OFF 상태 클래스는 변경되지 않았다"이다.

③ O
새로운 상태인 SLEEP이 추가되었으므로 SLEEP 상태 클래스가 새로 만들어져야 한다. 이 클래스는 on_button에서 ON으로, off_button에서 OFF로 전이하는 동작을 가진다.

④ O
상태 패턴에서는 Light가 상태 객체에 처리를 위임하므로, 상태 종류가 늘어나더라도 상태 인터페이스와 위임 구조가 유지되면 Light 클래스 자체는 보통 변경되지 않는다. 이 사례에서도 새로운 전이 로직은 상태 클래스들 안에서 처리된다.

오답 포인트

헷갈리기 쉬운 지점은 "새 상태가 추가되었으니 기존 상태를 전부 고쳐야 한다"고 막연히 생각하는 것이다. 이 문제는 도착 상태가 아니라 출발 상태를 기준으로 변경 범위를 따져야 하며, OFF에서는 새 이벤트 처리 요구가 없으므로 OFF 클래스는 그대로이다.

다시 나오면 이렇게 판단

상태 패턴 문제는 먼저 추가된 전이를 "어느 상태에서 어떤 입력을 받는가"로 쪼개서 본다. 그다음 새 전이의 출발 상태에 해당하는 클래스만 수정 대상으로 보고, 새 상태가 등장하면 그 상태 클래스를 추가하며, 컨텍스트가 단순 위임 구조이면 보통 변경되지 않는다고 판단할 수 있다.

암기 포인트

상태 패턴은 전이의 출발 상태 클래스가 바뀌고, 새 상태가 생기면 상태 클래스를 추가한다.

NoteBase

문 13. 익스트림 프로그래밍(XP)에서 좋은 사용자 스토리가 되기 위한 특성으로 가장 옳지 않은 것은?

① 각 사용자 스토리의 크기 또는 작업 소요 시간을 추정할 수 있어야 한다.

② 스토리는 테스트가 가능하도록 작성되어야 한다.

③ 스토리 간에 의존성이 강화되어야 한다.

④ 좋은 스토리는 그 크기가 작은 편이다.

정답 3번

출제 포인트

이 문제는 XP의 좋은 사용자 스토리 기준인 독립성, 테스트 가능성, 추정 가능성, 작은 크기를 구별하는 문제이다.

핵심 개념

좋은 사용자 스토리는 사용자 가치가 드러나면서도 개발 단위로 잘게 나뉜 요구사항이다. 대표 기준은 INVEST로, 서로 독립적이고(Independent), 협의 가능하며(Negotiable), 가치가 있고(Valuable), 작업량 추정이 가능하며(Estimable), 충분히 작고(Small), 테스트 가능해야 한다(Testable).

선지별 해설

① O
각 사용자 스토리의 크기나 작업 소요 시간을 대략적으로 추정할 수 있어야 반복 계획과 우선순위 조정이 가능하다. 이는 Estimable 조건에 해당한다.

② O
스토리는 완료 여부를 확인할 수 있도록 테스트 가능한 형태여야 한다. 이는 Testable 조건에 해당한다.

③ X
틀린 부분은 "스토리 간에 의존성이 강화되어야 한다"는 표현이다. 좋은 사용자 스토리는 서로 강하게 얽히는 것이 아니라 가능한 한 독립적이어야 한다. 올바르게는 "스토리 간 의존성은 최소화되어야 한다"이다.

④ O
좋은 스토리는 한 반복 주기 안에서 구현하고 검증하기 쉽도록 작은 편이어야 한다. 이는 Small 조건에 해당한다.

오답 포인트

③은 여러 스토리가 연결되어 있으면 체계적이라고 오해하기 쉽다는 점을 노린 선지이다. 그러나 XP에서는 의존성이 커질수록 우선순위 변경, 분할 개발, 반복 계획이 어려워지므로 좋은 스토리의 방향과 반대가 된다.

NoteBase

다시 나오면 이렇게 판단

사용자 스토리 문제는 INVEST를 기준으로 판단하면 된다. 추정 가능, 테스트 가능, 작음은 좋은 특성이고, 스토리 간 결합과 의존성 증가는 나쁜 특성으로 본다.

암기 포인트

좋은 사용자 스토리는 독립적이고 작으며 추정 가능하고 테스트 가능하다.

NoteBase

문 14. 모듈화 설계에 대한 설명으로 가장 옳지 않은 것은?

① 모듈화가 잘 된 소프트웨어일수록 재사용 가능성이 높다.

② 모듈 사이의 불필요한 상호 교류를 최소화함으로써 결합력(coupling)을 낮출 수 있다.

③ 모듈 사이의 기능적 관련성이 강할수록 응집력(cohesion)이 높다.

④ 기능 응집력(functional cohesion)이 높은 모듈에서는 모든 요소가 하나의 기능 구현을 위해 구성되어 있다.

정답 3번

출제 포인트

이 문제는 모듈화 설계의 핵심 기준인 응집력과 결합력의 의미를 구분할 수 있는지를 묻는 문제이다.

핵심 개념

응집력은 한 모듈 내부의 요소들이 하나의 기능을 위해 얼마나 밀접하게 관련되는지를 나타내는 정도이다.
결합력은 서로 다른 모듈 사이가 얼마나 강하게 의존하고 상호작용하는지를 나타내는 정도이다.
좋은 모듈화는 모듈 내부는 높은 응집력, 모듈 사이는 낮은 결합력을 갖게 하여 재사용성과 유지보수성을 높인다.

선지별 해설

① O
모듈화가 잘 되면 기능이 분리되고 인터페이스가 명확해져 필요한 부분만 독립적으로 재사용하기 쉬워진다.
따라서 재사용 가능성이 높아진다는 설명은 옳다.

② O
결합력은 모듈 간 의존 정도이므로, 불필요한 상호 교류를 줄이면 모듈 사이 영향이 감소하여 결합력을 낮출 수 있다. 좋은 모듈 설계의 방향과 일치한다.

③ X
틀린 부분은 "모듈 사이의 기능적 관련성"이다.
응집력은 모듈 사이의 관계가 아니라 모듈 내부 구성 요소들 사이의 관련성을 뜻한다.
틀린 표현 → 올바르게는 "모듈 내부 요소들의 기능적 관련성이 강할수록 응집력(cohesion)이 높다"이다.

④ O
기능 응집력은 가장 바람직한 응집 형태로, 모듈을 이루는 모든 요소가 하나의 기능만을 수행하도록 긴밀하게 구성된 상태를 뜻한다. 따라서 설명이 옳다.

NoteBase

오답 포인트

응집력과 결합력을 모두 '관련성'으로 외우면 ③처럼 모듈 내부 개념과 모듈 사이 개념을 뒤바꾸기 쉽다. 이 문항의 함정은 '모듈 사이의'라는 표현을 응집력에 붙여 개념 축을 바꾼 데 있다.

다시 나오면 이렇게 판단

선지에 '모듈 내부인가, 모듈 사이인가'를 먼저 표시하면 판단이 빨라진다. 내부 요소들의 집중도는 응집력, 모듈 간 의존도는 결합력으로 보면 된다. 그다음 좋은 설계는 높은 응집력과 낮은 결합력이라는 원칙에 맞는지 확인하면 된다.

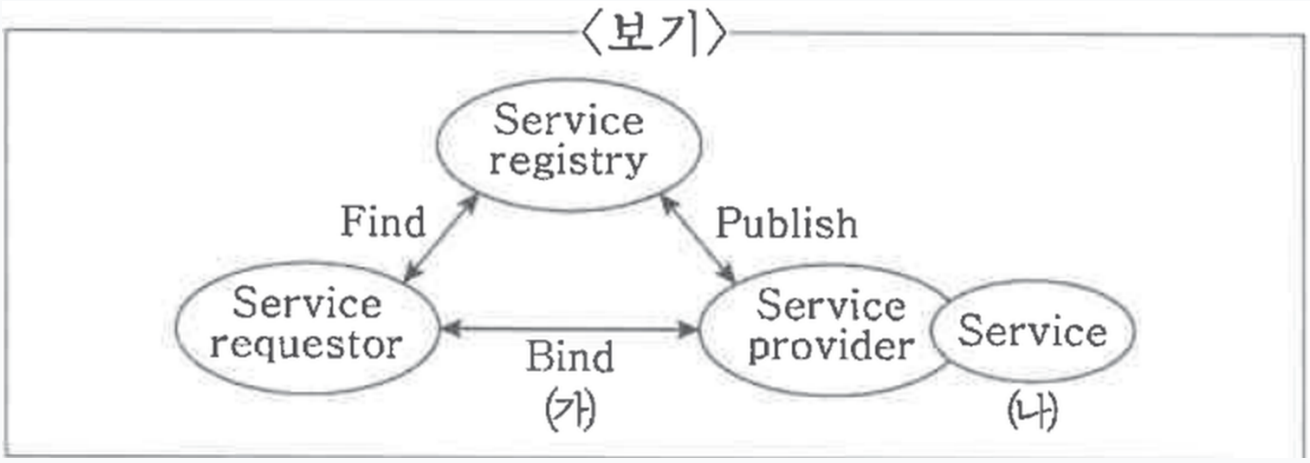
암기 포인트

응집력은 모듈 안, 결합력은 모듈 밖이라는 대비로 기억하면 된다.

NoteBase

문 15. <보기>는 웹 기반의 서비스 지향 아키텍처의 구성으로 (가)는 메시지를 교환하기 위한 프로토콜이고, (나)는 서비스 정의 언어이다. (가), (나)에 알맞은 내용을 순서대로 나열한 것은?

<보기>



① (가): SOAP, (나): UDDI

② (가): WSDL, (나): UDDI

③ (가): UDDI, (나): SOAP

④ (가): SOAP, (나): WSDL

정답 4번

출제 포인트

웹 서비스 지향 아키텍처에서 SOAP, WSDL, UDDI의 역할을 도식과 역할 키워드로 구분하는 문제이다.

NoteBase

선지별 해설

① X
- (가)를 SOAP으로 본 점은 맞지만, 틀린 표현은 '(나) UDDI'이다.
- UDDI는 서비스 레지스트리에 서비스를 Publish/Find할 때 쓰는 등록·검색 규격이다. (나)는 서비스 정의 언어이므로 올바르게는 '(나) WSDL'이다.

② X
- 틀린 표현은 '(가) WSDL'과 '(나) UDDI'이다.
- WSDL은 서비스 인터페이스와 호출 형식을 기술하는 언어이지 메시지 교환 프로토콜이 아니다. (가)는 올바르게는 'SOAP'이고, (나)는 올바르게는 'WSDL'이다.

③ X
- 틀린 표현은 '(가) UDDI'와 '(나) SOAP'이다.
- UDDI는 서비스 등록·검색용이고 SOAP은 메시지 교환용이므로 서로 역할이 다르다. 올바르게는 '(가) SOAP, (나) WSDL'이다.

④ O
- (가)를 SOAP, (나)를 WSDL로 둔 배열이 맞다.
- SOAP은 서비스 요청자와 제공자 사이에서 메시지를 교환하는 프로토콜이고, WSDL은 서비스의 연산, 메시지 형식, 접근 방법을 기술하는 서비스 정의 언어이다.

오답 포인트

SOAP, WSDL, UDDI를 모두 웹 서비스 관련 용어로만 외우면 역할이 섞인다. 이 문항은 그림에 Service registry가 보인다는 이유로 UDDI를 (나)에 넣거나, WSDL을 프로토콜로 착각하는 지점을 노린 문제이다.

다시 나오면 이렇게 판단

문장 속 역할 키워드를 먼저 대응시키면 된다. 등록·검색이면 UDDI, 서비스 설명이면 WSDL, 실제 메시지 교환이면 SOAP으로 판단한다.

암기 포인트

UDDI는 찾기, WSDL은 설명서, SOAP은 호출 메시지라는 대응으로 기억한다.

NoteBase

문 16. 소프트웨어 프로세스 모델 중 폭포수형 모델(waterfall model)이 제시하는 개발 단계의 선후 관계에서 먼저 실행되어야 하는 단계와 나중에 실행해야 할 단계로 가장 옳지 않은 것은? (단, 화살표 왼편의 단계가 오른쪽보다 먼저 실행해야 함을 나타내며, 그 중간에 다른 단계가 있을 수 있다.)

① 기능 요구사항 분석 → 설계

② 설계 → 시스템 시험

③ 통합 → 단위 시험

④ 코딩 → 배포, 설치 및 운영

정답 3번

출제 포인트

이 문제는 폭포수형 모델의 개발 단계가 어떤 선후관계로 이어지는지를 묻는 문제이다.

올바른 순서

기능 요구사항 분석 → 설계 → 코딩 → 단위 시험 → 통합 시험 → 시스템 시험 → 배포, 설치 및 운영의 흐름으로 본다.
왜 이 순서인가: 앞 단계의 산출물이 뒤 단계의 입력이 되기 때문이다. 요구사항이 정리되어야 설계를 할 수 있고, 코딩이 끝나야 단위 시험을 하며, 단위 시험을 거친 모듈을 묶어 통합 시험을 하고, 그 다음 전체 시스템 수준의 시험과 배포 단계로 진행된다.

선지별 해설

① O: 기능 요구사항 분석은 무엇을 만들어야 하는지 확정하는 단계이고, 설계는 그 요구를 어떻게 구현할지 구조화하는 단계이므로 분석이 설계보다 먼저이다.
② O: 시스템 시험은 통합된 전체 시스템을 검증하는 단계이므로 설계 이후에 코딩, 단위 시험, 통합 시험 등을 거쳐 나중에 수행된다.
③ X: 틀린 부분은 "통합 → 단위 시험"이다. 단위 시험은 개별 모듈 수준에서 먼저 수행되고, 그 결과를 바탕으로 여러 모듈을 묶어 통합 시험을 한다. 올바르게는 "단위 시험 → 통합 시험"이다.
④ O: 코딩이 되어 있어야 실제 소프트웨어를 배포하고 설치하여 운영할 수 있으므로 코딩이 배포, 설치 및 운영보다 먼저이다.

오답 포인트

단위 시험과 통합 시험의 순서를 뒤집으면 틀린다. 개별 모듈을 먼저 확인한 뒤 모듈 간 결합을 검증한다는 계층을 놓치면 ③을 맞는 문장처럼 보기 쉽다.

NoteBase

다시 나오면 이렇게 판단

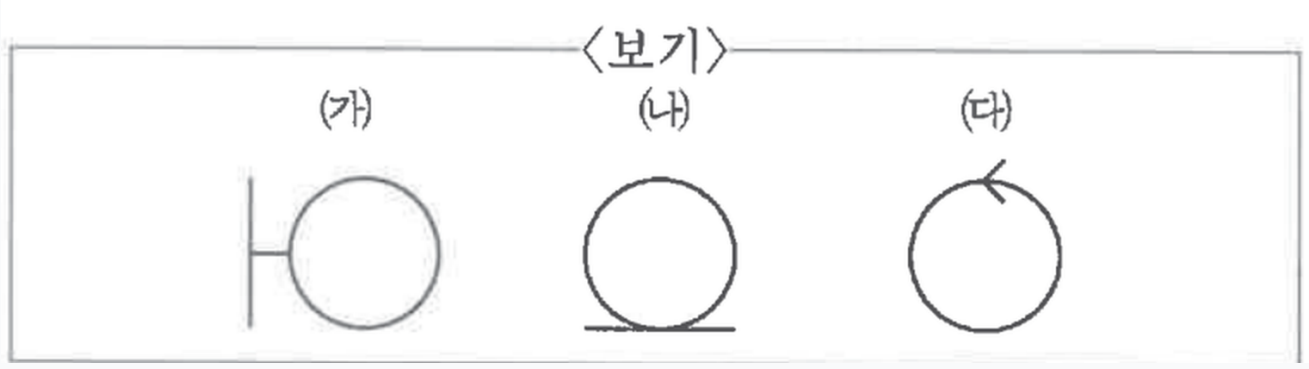
폭포수형 순서 문제는 산출물의 의존관계로 판단한다. 요구사항이 설계의 입력인지, 구현물이 시험의 입력인지, 개별 검증 뒤에 전체 검증이 오는지를 보면 선후관계를 빠르게 가를 수 있다.

암기 포인트

폭포수형은 분석 → 설계 → 구현 → 단위 시험 → 통합 시험 → 시스템 시험 → 운영의 직선 흐름이다.

NoteBase

문 17. <보기>는 UML 스테레오타입(stereotype)이 적용된 클래스를 표현한 것이다. (가)~(다)에 해당하는 스테레오타입을 옳게 짝지은 것은?



① (가): boundary, (나): control, (다): entity

② (가): boundary, (나): entity, (다): control

③ (가): control, (나): entity, (다): boundary

④ (가): control, (나): boundary, (다): entity

정답 2번

출제 포인트

이 문제는 UML 분석 모델의 BCE 스테레오타입 기호에서 시각적 특징을 보고 boundary, entity, control을 구분하는 문제이다.

NoteBase

선지별 해설

① X
- (가)를 boundary로 본 것은 맞지만, (나)를 control로 본 부분이 틀렸다. (나)는 원 아래 밑줄이 있으므로 entity이다.
- (다)를 entity로 본 부분도 틀렸다. (다)는 원 위쪽 제어 표식이 있으므로 control이다.
- 틀린 표현은 '(나)=control, (다)=entity'이고, 올바르게는 '(나)=entity, (다)=control'이다.

② O
- (가)=boundary, (나)=entity, (다)=control의 대응이 도식의 시각적 특징과 정확히 일치한다.
- boundary는 경계, entity는 데이터, control은 제어 흐름이라는 역할 구분까지 맞는다.

③ X
- (가)를 control로 본 부분이 틀렸다. (가)는 위쪽 제어 표식이 아니라 왼쪽 경계선이 있으므로 boundary이다.
- (다)를 boundary로 본 부분도 틀렸다. (다)는 왼쪽 경계선이 없고 위쪽 제어 표식이 있으므로 control이다.
- 틀린 표현은 '(가)=control, (다)=boundary'이고, 올바르게는 '(가)=boundary, (다)=control'이다.

④ X
- (가)를 control로 본 부분이 틀렸다. (가)는 boundary이다.
- (나)를 boundary로 본 부분도 틀렸다. (나)는 entity이다.
- (다)=entity라고 본 부분도 틀렸다. (다)는 control이다.
- 틀린 표현은 '(가)=control, (나)=boundary, (다)=entity'이고, 올바르게는 '(가)=boundary, (나)=entity, (다)=control'이다.

오답 포인트

원 모양이 모두 비슷하므로 위치별 보조 표식을 놓치면 boundary와 entity, control을 뒤섞기 쉽다. 특히 boundary의 왼쪽 세로선과 entity의 아래 밑줄을 비슷한 장식으로 보면 ①, ③, ④처럼 잘못 고르기 쉽다.

다시 나오면 이렇게 판단

먼저 원의 어느 위치에 보조 표식이 붙는지 본다. 왼쪽 세로선이면 boundary, 아래 밑줄이면 entity, 위쪽 제어 표식이면 control로 바로 대응시키면 된다.

암기 포인트

BCE 기호는 왼쪽 boundary, 아래 entity, 위 control로 외운다.

NoteBase

문 18. 객체지향 품질 척도의 설명으로 옳은 것을 <보기>에서 모두 고른 것은?

<보기>
ㄱ. 객체 클래스 사이의 결합(CBO, Coupling Between Object Classes): 해당 클래스가 의존하고 있는 클래스의 개수로, CBO가 클수록 재사용이 어렵다.
ㄴ. 클래스당 가중 메소드(WMC, Weighted Method per Class): 클래스의 메소드 개수에 그 클래스의 메소드가 호출하는 메소드의 개수를 더한 값으로, WMC가 클수록 책임이 커져 분리할 필요성이 있다.
ㄷ. 메소드의 응집 결핍(LCOM, Lack of Cohesion in Methods): 해당 클래스의 속성을 공유하지 않는 메소드 쌍의 수로, LCOM이 크면 클래스 응집도가 떨어진다.
ㄹ. 상속 트리의 깊이(DIT, Depth of Inheritance): 상속 트리의 루트로부터 해당 클래스까지 가장 깊은 상속 경로이다.

① ㄴ, ㄹ
② ㄱ, ㄴ, ㄷ
③ ㄱ, ㄷ, ㄹ
④ ㄴ, ㄷ, ㄹ

정답 3번

출제 포인트

객체지향 품질 척도인 CBO, WMC, LCOM, DIT의 정의를 정확히 구분하는 문제이다.

핵심 개념

CBO는 한 클래스가 다른 클래스들과 얼마나 강하게 결합되어 있는지를 보는 척도이다.
WMC는 클래스 안의 메서드들이 가진 복잡도 가중치의 합이며, 단순화한 경우 메서드 수로 본다.
LCOM은 메서드들이 같은 속성을 공유하지 못하는 정도를 나타내는 척도이며 값이 클수록 응집도가 낮다.
DIT는 상속 트리의 루트에서 해당 클래스까지의 최대 상속 깊이를 뜻한다.

NoteBase

선지별 해설

① X
ㄴ이 틀리고 ㄹ만 옳으므로 조합 전체가 틀리다. ㄴ의 틀린 부분은 "클래스의 메소드 개수에 그 클래스의 메소드가 호출하는 메소드의 개수를 더한 값"이라는 정의이다. 실제로는 WMC는 클래스 메서드들의 복잡도 가중치 합이며, 단순화하면 메서드 수로 본다. ㄹ의 DIT 설명은 상속 루트부터 해당 클래스까지의 가장 깊은 경로라는 뜻으로 옳다.

② X
ㄱ과 ㄷ은 옳지만 ㄴ이 틀리므로 조합 전체가 틀리다. ㄱ의 CBO는 클래스 간 결합 수를 뜻하고 값이 클수록 재사용과 유지보수가 어려워지므로 옳다. ㄴ은 WMC를 호출 메서드 수까지 더하는 값으로 설명한 점이 틀렸고, 올바르게는 메서드 복잡도 가중치의 합이다. ㄷ의 LCOM은 속성을 공유하지 않는 메서드 쌍이 많을수록 커지고, 값이 클수록 응집도가 낮아진다는 설명이 맞다.

③ O
ㄱ, ㄷ, ㄹ만 묶은 조합이므로 옳다. ㄱ은 CBO의 정의와 해석이 맞고, ㄷ은 LCOM이 클수록 응집도가 떨어진다는 점이 맞으며, ㄹ은 DIT를 상속 깊이로 설명한 내용이 맞다.

④ X
ㄷ과 ㄹ은 옳지만 ㄴ이 틀리므로 조합 전체가 틀리다. ㄴ의 잘못된 표현은 WMC를 호출하는 메서드 개수까지 합산하는 척도로 본 점이다. 올바르게는 클래스 내부 메서드들의 복잡도 가중치 합으로 이해해야 한다.

오답 포인트

WMC를 메서드 호출 관계 수와 연결해 외우면 ㄴ을 맞는 설명으로 오인하기 쉽다. 이 문항은 척도의 이름이 가리키는 대상이 클래스 간 관계인지, 클래스 내부 메서드 복잡도인지, 응집도인지, 상속 깊이인지를 구분하는지를 묻는다.

다시 나오면 이렇게 판단

척도 이름이 가리키는 대상을 먼저 대응시키면 빠르게 판단할 수 있다. CBO는 클래스 간 결합, WMC는 클래스 내부 메서드 가중치, LCOM은 응집 결핍, DIT는 상속 깊이로 연결하면 조합 선지를 바로 걸러낼 수 있다.

암기 포인트

CBO는 클래스 간 결합, WMC는 메서드 가중치 합, LCOM은 응집도 부족, DIT는 상속 깊이이다.

NoteBase

문 19. 리팩토링(refactoring)과 관련한 설명으로 가장 옳지 않은 것은?

① 리팩토링을 통해 성능 개선이 이루어지기도 한다.

② 소프트웨어에 대한 가독성과 이해도를 높일 수 있다.

③ 내부 구조의 변경 없이 외부 인터페이스와 기능을 변경할 수 있다.

④ 리팩토링이 필요한 코드의 징후를 코드 스멜(code smell)이라고 한다.

정답 3번

출제 포인트

이 문제는 리팩토링의 핵심 정의와 코드 스멜의 의미를 정확히 구별하는지를 묻는다.

핵심 개념

리팩토링은 소프트웨어의 외부에서 보이는 동작은 유지한 채 내부 구조를 개선하는 작업이다. 코드 스멜은 당장 오류는 아니지만 구조 개선이 필요할 가능성을 알려 주는 징후이다.

선지별 해설

1번 O: 리팩토링의 주목적은 성능 개선이 아니라 구조 개선이지만, 중복 제거나 비효율 코드 정리 과정에서 성능 개선이 함께 이루어지기도 한다.
2번 O: 메서드 분리, 이름 개선, 중복 제거 등은 코드를 읽기 쉽게 만들므로 소프트웨어의 가독성과 이해도를 높일 수 있다.
3번 X: 틀린 부분은 "내부 구조의 변경 없이 외부 인터페이스와 기능을 변경"이라는 표현이다. 리팩토링은 반대로 외부 인터페이스와 관찰 가능한 기능은 유지하고 내부 구조를 변경하여 품질을 높이는 작업이다.
4번 O: 코드 스멜은 리팩토링이 필요할 수 있음을 보여 주는 나쁜 징후를 뜻하는 용어이므로 옳은 설명이다.

오답 포인트

3번은 리팩토링의 정의를 정반대로 바꾸어 놓은 선지이다. 수험생이 내부 구조 변경과 외부 기능 유지의 방향을 뒤집어 기억하면 쉽게 틀린다.

다시 나오면 이렇게 판단

리팩토링 문제가 나오면 먼저 외부 동작 유지 여부를 본다. 그다음 내부 구조 개선인지, 외부 기능 변경인지 구분하면 정답을 빠르게 가를 수 있다.

암기 포인트

리팩토링은 밖은 그대로 두고 안을 고치는 작업이고, 코드 스멜은 그 작업이 필요하다는 신호이다.

NoteBase

문 20. <보기>와 관련한 소프트웨어 아키텍처로 가장 옳은 것은?

<보기>
사물인터넷의 한 가지 응용 시스템인 홈 오토메이션은 허락 없이 출입구 또는 창문이 열리는 상황을 센서가 감지하면 자동으로 알람 경보를 내거나 문자 메시지를 거주자에게 전달한다.

① 계층형 아키텍처(layered architecture)
② 이벤트 기반 아키텍처(event-based architecture)
③ 파이프 필터 아키텍처(pipe and filter architecture)
④ 저장소 아키텍처(repository architecture)

정답 2번

출제 포인트

이 문제는 사건이 발생하면 센서가 이를 감지하고 후속 동작을 즉시 트리거하는 구조가 어떤 아키텍처인지 묻는 문제이다.

핵심 개념

계층형 아키텍처는 기능을 계층으로 나누어 상위 계층이 하위 계층의 서비스를 사용하는 구조이다.
이벤트 기반 아키텍처는 특정 이벤트의 발생을 감지하면 그 이벤트를 구독한 구성요소가 비동기적으로 반응하는 구조이다.
파이프 필터 아키텍처는 입력 데이터가 여러 처리 단계를 순차적으로 통과하며 변환되는 구조이다.
저장소 아키텍처는 여러 구성요소가 중앙 저장소를 공유하며 데이터를 읽고 쓰는 구조이다.

NoteBase

선지별 해설

① X
계층형 아키텍처는 프레젠테이션 계층, 비즈니스 계층, 데이터 계층처럼 책임을 층으로 나누는 데 초점이 있다. 보기의 핵심은 층 분리가 아니라 센서가 상황을 감지했을 때 경보나 문자 발송이 즉시 이어지는 사건 반응 구조이다.
틀린 표현은 "계층형 아키텍처"이다. 올바르게는 이벤트 발생에 따라 후속 기능이 작동하는 이벤트 기반 아키텍처이다.

② O
이벤트 기반 아키텍처는 센서가 출입구나 창문 열림이라는 이벤트를 감지하면 경보 시스템이나 메시지 전송 기능이 그 이벤트에 반응하도록 구성한다. 홈 오토메이션의 감지-통지 흐름과 가장 직접적으로 맞는다.

③ X
파이프 필터 아키텍처는 데이터를 여러 필터가 순차 가공하는 배치형 또는 스트림 처리 흐름에 적합하다. 보기는 데이터 변환 단계의 연쇄보다 특정 사건에 대한 알림 반응이 핵심이다.
틀린 표현은 "파이프 필터 아키텍처"이다. 올바르게는 센서 이벤트를 받아 관련 서비스가 반응하는 이벤트 기반 아키텍처이다.

④ X
저장소 아키텍처는 중앙 저장소를 중심으로 여러 컴포넌트가 공통 데이터를 공유하는 구조이다. 보기에는 중앙 저장소 공유가 핵심으로 드러나지 않고, 무단 개방 감지라는 사건이 발생했을 때 경보와 문자 발송이 트리거되는 점이 핵심이다.
틀린 표현은 "저장소 아키텍처"이다. 올바르게는 사건 감지와 후속 반응을 중심으로 한 이벤트 기반 아키텍처이다.

오답 포인트

센서, 문자 메시지, 홈 오토메이션 같은 단어만 보고 IoT 시스템 전체 구조를 막연히 떠올리면 저장소형이나 계층형으로 흔들리기 쉽다. 이 문항은 시스템의 일반적 구성보다 "감지되면 바로 반응한다"는 사건 중심 표현이 무엇인지 잡아내는지가 핵심 함정이다.

다시 나오면 이렇게 판단

문장에 감지, 발생, 통지, 트리거, 반응처럼 사건 발생 후 후속 기능이 연쇄적으로 작동하는 표현이 나오면 먼저 이벤트 기반 아키텍처를 떠올린다. 반대로 계층 분리인지, 순차 데이터 처리인지, 중앙 저장소 공유인지가 핵심이면 다른 아키텍처를 검토한다.

NoteBase

암기 포인트

이벤트 기반 아키텍처는 "이벤트 발생 -> 감지 -> 반응" 구조이고, 파이프 필터는 "입력 -> 단계별 변환" 구조이다.

NoteBase