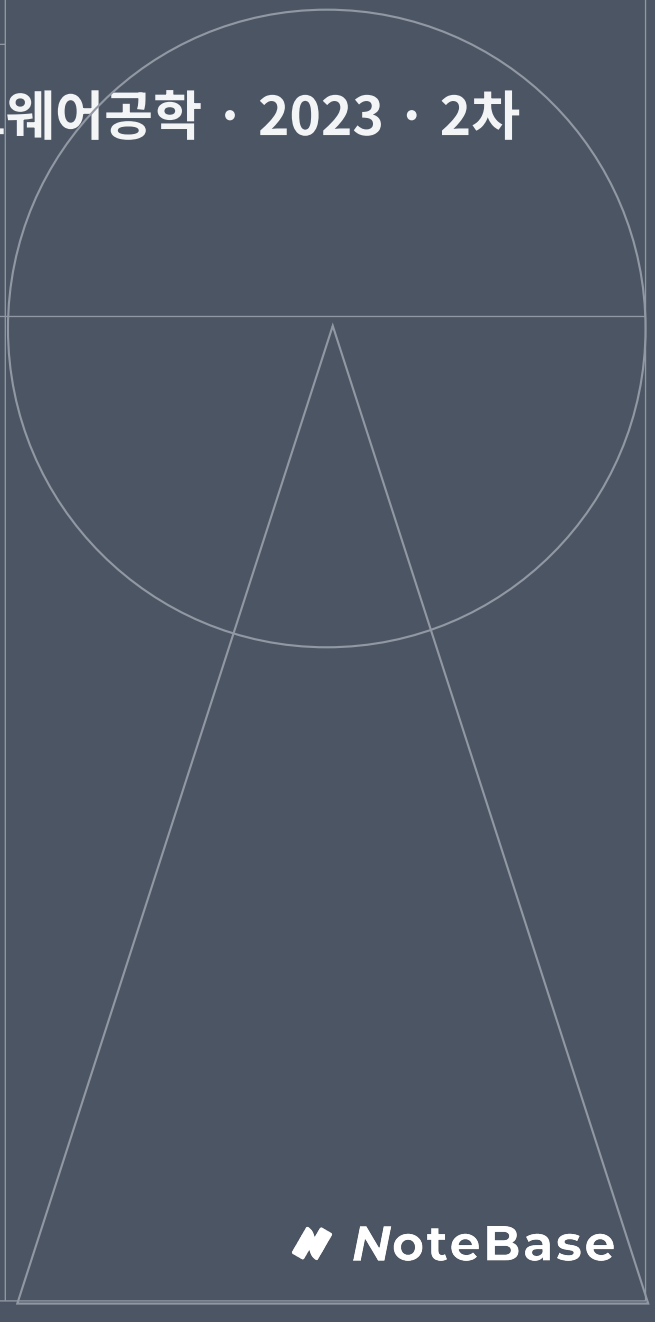
# 기출OX 해설집

공무원/국가직/7급 · 소프트웨어공학 · 2023 · 2차



NoteBase

# 기출OX 해설집

## 공무원/국가직/7급 · 소프트웨어공학 · 2023 · 2차

**공무원/국가직/7급 · 소프트웨어공학 · 2023 · 2차**

**문 1. 애자일 방법론에 대한 설명으로 옳지 않은 것은?**

① 요구사항의 변화가 적은 대규모 시스템 개발에 적합하다.

② 포괄적인 문서보다 실행 가능한 소프트웨어를 중시한다.

③ 프로세스와 도구보다 개인과의 상호 소통을 중시한다.

④ 익스트림 프로그래밍(XP), 동적 시스템 개발 방법론(DSDM) 등이 있다.

**정답 1번**

**출제 포인트**

이 문제는 애자일 방법론의 핵심 가치와 적용 상황을 전통적 계획 주도 개발과 구별할 수 있는지를 묻는 문제이다.

**핵심 개념**

애자일은 변화에 빠르게 대응하도록 짧은 반복 개발과 지속적인 협업을 중시하는 개발 접근이다. 애자일은 실행 가능한 소프트웨어와 개인 간 상호작용을 중시하고, 요구사항 변화가 잦은 환경에 더 잘 맞는다. 반대로 요구사항이 안정적이고 문서·절차 중심 통제가 강한 대규모 개발은 전통적 계획 주도 방법론의 성격이 더 강하다.

**선지별 해설**

① X: 틀린 부분은 "요구사항의 변화가 적은 대규모 시스템 개발에 적합하다"는 표현이다. 애자일은 요구사항 변화에 유연하게 대응하는 데 강점이 있으므로, 올바르게는 "요구사항 변화가 잦고 사용자 피드백을 자주 반영해야 하는 개발에 적합하다"이다.

② O: 애자일 선언은 포괄적인 문서보다 실행 가능한 소프트웨어를 더 가치 있게 본다. 문서를 완전히 배제한다는 뜻이 아니라 실제로 동작하는 결과물을 우선한다는 뜻이다.

③ O: 애자일 선언은 프로세스와 도구보다 개인과 상호작용을 더 가치 있게 본다. 팀원 간 소통과 협업이 빠른 피드백과 변경 대응의 핵심이기 때문이다.

④ O: XP와 DSDM은 대표적인 애자일 방법론에 해당한다. 둘 다 반복 개발과 변화 대응을 중시하는 애자일 계열의 예시이다.

NoteBase

### 오답 포인트

애자일을 단순히 빠르게 만드는 방법으로만 외우면 적용 환경을 거꾸로 판단하기 쉽다. 이 문항은 ①에서 "변화가 적은", "대규모"라는 표현을 넣어 전통적 계획 주도 개발의 특징을 애자일에 뒤섞어 놓은 점이 함정이다.

### 다시 나오면 이렇게 판단

애자일 관련 선지는 먼저 가치 축을 본다. 문서보다 실행 결과, 도구보다 사람, 고정 계획보다 변화 대응을 말하면 애자일 쪽이다. 반대로 변화가 적고 절차·문서·사전 계획 중심의 안정적 대규모 개발을 강조하면 전통적 방법론 쪽으로 판단한다.

### 암기 포인트

애자일은 변화 대응형, 전통적 방법론은 계획 고정형이라는 대비로 기억한다.

NoteBase

**문 2. 객체지향 설계 원칙에 대한 설명으로 옳지 않은 것은?**

① 확장에는 열려 있어야 하고 변경에는 닫혀 있어야 한다.

② 상위 클래스의 객체는 하위 클래스의 객체로 교체될 수 있어야 한다.

③ 클라이언트는 추상 클래스나 인터페이스가 아닌 구체(concrete) 클래스에 의존해야 한다.

④ 하나의 범용 인터페이스보다는 여러 개의 특화된 인터페이스로 나누어야 한다.

**정답 3번**

**출제 포인트**

이 문제는 객체지향 설계 원칙인 OCP, LSP, DIP, ISP의 핵심 의미를 구별하는 문제이다.

**핵심 개념**

OCP는 기능 확장에는 열려 있고 기존 코드 수정에는 닫혀 있어야 한다는 원칙이다.
LSP는 상위 타입을 사용하는 곳에서 하위 타입으로 바꾸어도 프로그램의 의미가 깨지지 않아야 한다는 원칙이다.
DIP는 클라이언트가 구체 클래스보다 추상화인 인터페이스나 추상 클래스에 의존해야 한다는 원칙이다.
ISP는 하나의 큰 인터페이스보다 역할별로 분리된 작은 인터페이스 여러 개가 바람직하다는 원칙이다.

NoteBase

### 선지별 해설

① O
확장에는 열려 있고 변경에는 닫혀 있어야 한다는 설명은 개방-폐쇄 원칙(OCP)을 그대로 설명한 내용이므로 옳다.

② O
상위 클래스형으로 다루던 객체를 하위 클래스 객체로 바꾸어도 정상 동작해야 한다는 설명은 리스코프 치환 원칙(LSP)에 해당하므로 옳다.

③ X
틀린 부분은 "구체(concrete) 클래스에 의존해야 한다"이다.
의존 역전 원칙(DIP)에서는 클라이언트가 구체 클래스가 아니라 추상 클래스나 인터페이스 같은 추상화에 의존해야 한다.
따라서 올바른 표현은 "클라이언트는 구체 클래스가 아닌 추상 클래스나 인터페이스에 의존해야 한다"이다.

④ O
하나의 범용 인터페이스보다 여러 개의 특화된 인터페이스로 나누어야 한다는 설명은 인터페이스 분리 원칙(ISP)이므로 옳다.

### 오답 포인트

③은 DIP의 방향을 거꾸로 바꾸어 놓은 선지이다. "추상화에 의존"과 "구체 클래스에 의존"을 바꾸어 내면 익숙한 용어만 보고 맞는 문장처럼 보이기 쉽다.

### 다시 나오면 이렇게 판단

선지에 의존 대상이 나오면 먼저 추상화인지 구체 클래스인지 확인한다. 객체지향 설계 원칙 문제는 각 원칙의 핵심 키워드 하나만 정확히 대입해도 빠르게 판단할 수 있다.

### 암기 포인트

OCP는 확장 개방·수정 폐쇄, LSP는 치환 가능, DIP는 추상화 의존, ISP는 인터페이스 분리이다.

NoteBase

**문 3. 요구사항 수집 기법에 대한 설명으로 옳지 않은 것은?**

① 인터뷰는 사용자 또는 관계자들과의 직접적인 대화를 통하여 요구사항을 추출한다.

② 워크스루는 업무에서 사용하고 있는 다양한 종류의 문서를 분석하는 기법이다.

③ JAD(Joint Application Development)는 프로젝트 관리자, 사용자, 개발자 등이 모여 상호 토론하는 방법이다.

④ 관찰은 개발될 소프트웨어의 사용자가 수행하는 활동을 살펴보는 기법이다.

**정답 2번**

**출제 포인트**

이 문제는 요구사항 수집 기법의 정의를 구분하여 인터뷰, 워크스루, JAD, 관찰의 차이를 정확히 아는지를 묻는 문제이다.

**핵심 개념**

인터뷰는 사용자나 이해관계자와 직접 대화하여 요구를 끌어내는 기법이다. 워크스루는 참여자들이 모여 업무 흐름, 시나리오, 산출물을 함께 따라가며 이해를 맞추는 검토 성격의 기법이다. JAD는 관리자, 사용자, 개발자가 함께 참여하는 집중 워크숍 방식의 요구사항 정의 기법이다. 관찰은 사용자가 실제 업무를 수행하는 모습을 살펴 숨은 요구와 작업 절차를 파악하는 기법이다.

**선지별 해설**

① O: 인터뷰는 사용자 또는 관련자와의 직접적인 질의응답을 통해 요구사항을 수집하는 대표적 기법이므로 옳다.
② X: 틀린 부분은 "업무에서 사용하고 있는 다양한 종류의 문서를 분석하는 기법"이라는 설명이다. 이는 워크스루가 아니라 문서 분석에 해당한다. 워크스루는 참여자들이 모여 절차나 산출물을 함께 검토하며 요구를 구체화하는 방식이다.
③ O: JAD는 프로젝트 관리자, 사용자, 개발자 등 여러 이해관계자가 함께 모여 집중적으로 토론하며 요구사항을 정리하는 방법이므로 옳다.
④ O: 관찰은 사용자가 실제로 수행하는 활동과 작업 흐름을 살펴 요구사항을 파악하는 기법이므로 옳다.

**오답 포인트**

②는 워크스루와 문서 분석을 바꿔 놓은 선지이다. 요구사항 수집 기법 문제에서는 "문서를 분석한다"는 표현이 나오면 문서 분석을 먼저 떠올려야 하며, 워크스루는 사람들의 공동 검토라는 점이 핵심이다.

NoteBase

**다시 나오면 이렇게 판단**

선지의 핵심 동사를 먼저 본다. 직접 묻고 답하면 인터뷰, 함께 모여 토론하면 JAD, 실제 작업을 지켜보면 관찰, 기존 문서를 읽고 분석하면 문서 분석으로 판단하면 된다.

**암기 포인트**

인터뷰는 대화, JAD는 합동 워크숍, 관찰은 현장 확인, 문서 분석은 문서 검토이다.

NoteBase

**문 4. (가), (나)에 해당하는 용어를 바르게 연결한 것은?**

(가) 모듈과 모듈 사이의 관계에서 관련 정도를 나타내는 용어이다. 모듈 간에는 관련이 적을수록 상호 의존성이 줄어들어 모듈의 독립성이 높아지고 독립성이 높아지면 모듈 간에 영향이 적어져 좋은 설계가 된다.
(나) 모듈 내부에 존재하는 구성 요소 사이의 밀접한 정도를 나타내는 용어이다. 밀접한 정도가 높을수록 필요한 구성 요소만 모여 있고, 낮을수록 서로 관련성이 적은 구성 요소들이 모여 있다.

① (가): 결합도(coupling), (나): 응집도(cohesion)
② (가): 결합도(coupling), (나): 성숙성(maturity)
③ (가): 응집도(cohesion), (나): 결합도(coupling)
④ (가): 응집도(cohesion), (나): 성숙성(maturity)

**정답 1번**

**출제 포인트**

이 문제는 모듈 간의 관련 정도를 뜻하는 결합도와 모듈 내부 요소들의 밀접 정도를 뜻하는 응집도를 구별할 수 있는지를 묻는다.

**핵심 개념**

결합도: 모듈과 모듈 사이의 의존 관계가 얼마나 강한지를 나타내는 정도이다. 키워드는 모듈 간 관계, 의존성, 낮을수록 좋은 설계이다.
응집도: 하나의 모듈 내부 요소들이 하나의 기능으로 얼마나 긴밀하게 모여 있는지를 나타내는 정도이다. 키워드는 모듈 내부 집중도, 높을수록 좋은 설계이다.
성숙성: 소프트웨어가 오랜 사용과 결함 수정 과정을 거치며 안정적으로 운용될 가능성을 뜻하는 품질 특성이다.

**선지별 해설**

1번: O. (가)는 모듈과 모듈 사이의 관련 정도를 설명하므로 결합도이고, 관련이 적을수록 독립성이 높아져 좋은 설계가 된다. (나)는 모듈 내부 구성 요소 사이의 밀접한 정도를 설명하므로 응집도이다.
2번: X. 틀린 부분은 (나)를 성숙성으로 본 표현이다. (나)는 품질 특성인 성숙성이 아니라 모듈 내부 요소의 관련성을 뜻하는 응집도이다.
3번: X. 틀린 부분은 (가)를 응집도로, (나)를 결합도로 뒤바꾼 표현이다. 올바르게는 (가)는 모듈 간 관계이므로 결합도이고, (나)는 모듈 내부의 집중도이므로 응집도이다.
4번: X. 틀린 부분은 (가)를 응집도로, (나)를 성숙성으로 본 표현이다. 올바르게는 (가)는 결합도이고, (나)는 성숙성이 아니라 응집도이다.

NoteBase

### 오답 포인트

출제자는 모듈 간과 모듈 내부라는 기준을 바꾸어 헷갈리게 만든다. 특히 응집도와 결합도는 둘 다 모듈의 품질과 관련되어 이름이 비슷해 보이지만, 판단 축이 외부 관계인지 내부 집중도인지가 다르다.

### 다시 나오면 이렇게 판단

설명 문장에 모듈과 모듈 사이, 의존성, 독립성이 나오면 결합도로 본다. 모듈 내부 구성 요소, 하나의 기능, 밀접한 정도가 나오면 응집도로 본다.

### 암기 포인트

결합도는 모듈 밖 관계, 응집도는 모듈 안 집중도이다.

NoteBase

**문 5. 컴포넌트 기반 개발에 대한 설명으로 옳지 않은 것은?**

① 컴포넌트는 정적으로 바인딩되며 실행시간에는 인터페이스를 통해 접근할 수 없다.

② 재사용 가능한 컴포넌트를 기반으로 소프트웨어를 개발하는 방법론이다.

③ 컴포넌트는 독립적이고 교체 가능한 부품이다.

④ 컴포넌트는 인터페이스를 통해 다른 컴포넌트와 연결된다.

**정답 1번**

**출제 포인트**

이 문제는 컴포넌트 기반 개발에서 컴포넌트의 재사용성, 독립성, 인터페이스 기반 결합이라는 핵심 특징을 구별하는 문제이다.

**핵심 개념**

컴포넌트: 명확한 인터페이스를 제공하며 독립적으로 배포·교체할 수 있는 재사용 가능한 소프트웨어 단위이다. / 결정 차이축: 독립적 배포와 인터페이스 기반 조합이 가능한가이다. / 키워드: 재사용, 독립성, 교체 가능성, 인터페이스

컴포넌트 기반 개발: 이미 만들어진 컴포넌트를 조합하여 시스템을 구축하는 개발 방식이다. / 결정 차이축: 새로 전부 구현하는 것이 아니라 검증된 부품을 조립하는가이다. / 키워드: 조립, 재사용, 생산성

NoteBase

**선지별 해설**

① X
- 틀린 부분은 "정적으로 바인딩되며"와 "실행시간에는 인터페이스를 통해 접근할 수 없다"이다.
- 컴포넌트는 인터페이스를 통해 상호작용하며, 구현부를 숨긴 채 교체·연결될 수 있는 것이 핵심이다.
- 실제로는 실행 시에도 인터페이스를 통해 사용될 수 있고, 필요에 따라 동적 바인딩이나 교체가 가능한 구조를 지향한다.

② O
- 재사용 가능한 컴포넌트를 기반으로 시스템을 조립해 소프트웨어를 개발하는 방식이라는 설명으로, 컴포넌트 기반 개발의 정의에 해당한다.

③ O
- 컴포넌트는 독립적으로 배포할 수 있고 다른 부품으로 대체할 수 있어야 하므로, 독립적이고 교체 가능한 부품이라는 설명이 맞다.

④ O
- 컴포넌트 간 결합은 내부 구현이 아니라 공개된 인터페이스를 통해 이루어지므로, 다른 컴포넌트와 인터페이스로 연결된다는 설명이 맞다.

**오답 포인트**

①은 바인딩 방식을 "정적"으로 단정하고 인터페이스 접근까지 부정해 컴포넌트의 핵심 속성을 뒤집어 놓은 선지이다. 컴포넌트와 일반 모듈을 비슷하게 보고 내부 구현에 직접 결합된다고 생각하면 이 함정에 걸리기 쉽다.

**다시 나오면 이렇게 판단**

컴포넌트 관련 선지는 먼저 인터페이스 제공 여부, 독립 배포 가능 여부, 교체 가능 여부를 본다. 이 세 요소를 부정하면 대체로 틀린 선지이고, 재사용과 조합을 강조하면 옳은 선지일 가능성이 높다.

**암기 포인트**

컴포넌트는 인터페이스로 연결되고 독립적으로 교체되는 재사용 부품이다.

NoteBase

**문 6. V 모델에 대한 설명으로 옳은 것만을 모두 고르면?**

ㄱ. V 모델은 인수 테스트를 포함한다.
ㄴ. 폭포수 모델의 변형으로 테스트 단계를 추가 확장한 것이다.
ㄷ. 폭포수 모델이 산출물 중심이라면, V 모델은 각 개발 단계를 검증하는 데 초점을 둔다.

① ㄱ, ㄴ
② ㄱ, ㄷ
③ ㄴ, ㄷ
④ ㄱ, ㄴ, ㄷ

**정답 4번**

**출제 포인트**

이 문제는 V 모델이 폭포수 모델을 어떻게 확장했는지와 각 개발 단계에 대응하는 테스트·검증 관점을 묻는 문제이다.

**핵심 개념**

폭포수 모델은 분석, 설계, 구현을 순차적으로 진행하는 산출물 중심 모델이다.
V 모델은 이 흐름을 유지하면서 각 개발 단계에 대응하는 테스트 단계를 연결해 검증과 확인을 강화한 모델이다.
요구사항에는 인수 테스트, 상위 설계에는 시스템 테스트, 하위 설계에는 통합 테스트, 구현에는 단위 테스트가 대응된다는 점이 핵심이다.

NoteBase

**선지별 해설**

① X
- ㄱ은 옳고 ㄴ도 옳지만 ㄷ이 빠져 있다.
- ㄷ도 옳다. 폭포수 모델이 문서와 산출물의 순차 생산에 무게를 둔다면, V 모델은 각 단계 산출물을 대응 테스트 단계로 검증하는 데 초점을 둔다.
- 틀린 표현은 "ㄱ, ㄴ"만 고른 점이다. 올바르게는 "ㄱ, ㄴ, ㄷ"이다.

② X
- ㄱ은 옳고 ㄷ도 옳지만 ㄴ이 빠져 있다.
- ㄴ도 옳다. V 모델은 폭포수 모델의 변형으로, 개발 단계 옆에 대응 테스트 단계를 추가해 확장한 형태이다.
- 틀린 표현은 "ㄱ, ㄷ"만 고른 점이다. 올바르게는 "ㄱ, ㄴ, ㄷ"이다.

③ X
- ㄴ과 ㄷ은 옳지만 ㄱ이 빠져 있다.
- ㄱ도 옳다. V 모델에는 단위 테스트, 통합 테스트, 시스템 테스트, 인수 테스트가 포함되므로 인수 테스트를 포함한다는 설명이 맞다.
- 틀린 표현은 "ㄴ, ㄷ"만 고른 점이다. 올바르게는 "ㄱ, ㄴ, ㄷ"이다.

④ O
- ㄱ은 옳다. V 모델은 구현 이후의 테스트를 개발 단계와 대응시켜 인수 테스트까지 포함하는 생명주기 모델이다.
- ㄴ은 옳다. 폭포수 모델의 직선적 개발 흐름에 테스트·검증 관점을 추가한 변형이 V 모델이다.
- ㄷ은 옳다. V 모델의 핵심은 각 개발 단계의 결과물을 대응 테스트로 검증하는 데 있다.

**오답 포인트**

수험생이 ㄱ을 단위 테스트와 혼동해 인수 테스트를 제외하거나, ㄷ을 너무 추상적으로 읽고 일반적인 순차 개발 설명으로 넘기면 일부 조합만 고르기 쉽다. 이 문항은 개별 진술의 진위를 먼저 판단한 뒤 조합을 고르는 문제이다.

**다시 나오면 이렇게 판단**

V 모델 문제가 나오면 먼저 폭포수 모델의 확장인지, 각 개발 단계와 테스트 단계가 1:1로 대응하는지, 인수 테스트까지 포함하는지를 확인한다. 그다음 각 진술의 O/X를 확정한 뒤 조합 보기를 고르면 된다.

**암기 포인트**

V 모델은 폭포수 + 단계별 검증 모델이다. 왼쪽은 개발, 오른쪽은 대응 테스트이며 끝에는 인수 테스트가 놓인다.

NoteBase

**문 7. CMMI(Capability Maturity Model Integration)의 단계에 대한 설명으로 옳지 않은 것은?**

① 관리(managed) - 프로젝트 진행 비용, 일정, 기능에 대한 추적이 가능한 단계

② 정의(defined) - 소프트웨어 개발과정이 임시방편이거나 무질서하여 이를 정의해야 하는 단계

③ 최적화(optimizing) - 정량적인 피드백과 기술을 통해 소프트웨어 개발 프로세스가 지속적으로 개선되는 단계

④ 정량적 관리(quantitatively managed) - 소프트웨어 프로세스와 제품의 품질에 대해 정량적 측정이 이루어지는 단계

**정답 2번**

**출제 포인트**

이 문제는 CMMI 성숙도 단계별 핵심 특징을 구분할 수 있는지를 묻는 문제이다.

**핵심 개념**

Initial은 개발 과정이 임시방편적이고 무질서한 상태인 단계이다.
Managed는 프로젝트 단위에서 요구사항, 일정, 비용, 산출물을 계획하고 추적할 수 있는 단계이다.
Defined는 조직 차원의 표준 프로세스가 문서화되고 정립된 단계이다.
Quantitatively Managed는 프로세스와 제품 품질을 정량적으로 측정하고 관리하는 단계이다.
Optimizing은 정량적 피드백을 바탕으로 프로세스를 지속적으로 개선하는 단계이다.
이 문제의 핵심 구분축은 임시방편적 상태는 Initial, 표준화된 정의는 Defined라는 점이다.

NoteBase

**선지별 해설**

① O
Managed 단계는 프로젝트 진행 비용, 일정, 기능을 계획하고 추적할 수 있는 수준이므로 설명이 타당하다.

② X
틀린 부분은 "소프트웨어 개발과정이 임시방편이거나 무질서하여 이를 정의해야 하는 단계"라는 표현이다.
이 설명은 Defined가 아니라 Initial 단계의 특징이다.
올바르게는 Defined 단계는 조직 차원의 표준 소프트웨어 프로세스가 문서화되고 정립되어 일관되게 사용하는 단계이다.

③ O
Optimizing 단계는 정량적 피드백과 혁신 기술을 활용해 프로세스를 지속적으로 개선하는 단계이므로 설명이 맞다.

④ O
Quantitatively Managed 단계는 소프트웨어 프로세스와 제품 품질을 정량적으로 측정하고 관리하는 단계이므로 설명이 맞다.

**오답 포인트**

Defined와 Initial을 혼동하면 ②를 맞는 설명으로 오인하기 쉽다. "정의해야 하는 상태"는 아직 프로세스가 정립되지 않은 Initial의 특징이고, Defined는 이미 표준 프로세스가 정의된 상태라는 점이 함정이다.

**다시 나오면 이렇게 판단**

CMMI 단계 문제는 먼저 단계명을 낮은 수준에서 높은 수준으로 배열한 뒤 각 단계의 핵심 키워드를 붙여 판단한다. Initial은 무질서, Managed는 프로젝트 추적, Defined는 조직 표준화, Quantitatively Managed는 정량 관리, Optimizing은 지속 개선으로 구분된다.

**암기 포인트**

Initial은 무질서, Managed는 추적, Defined는 표준화, Quantitatively Managed는 정량화, Optimizing은 개선이다.

NoteBase

**문 8. 다음 설명에 해당하는 아키텍처 패턴은?**

◯ 분리된(decoupled) 컴포넌트들이 원격 서비스를 호출해 상호작용하는 시스템을 구조화할 때 유용하다.
◯ OMG(Object Management Group)의 CORBA는 이 아키텍처의 대표적인 사례이다.

① MVC(Model-View-Controller)
② 계층(layered)
③ 브로커(broker)
④ 파이프 필터(pipe-and-filter)

**정답 3번**

**출제 포인트**

이 문제는 분산 환경에서 원격 서비스 호출을 중개하는 브로커 패턴의 정의와 대표 사례를 식별하는 문제이다.

**핵심 개념**

브로커 패턴은 분산된 클라이언트와 서버 사이에 중개자 역할의 브로커를 두고 원격 서비스 호출을 연결하는 아키텍처이다.
MVC는 모델, 뷰, 컨트롤러로 화면 처리와 업무 로직의 책임을 분리하는 패턴이다.
계층 패턴은 시스템을 역할별 계층으로 나누어 인접 계층 중심으로 상호작용하게 하는 구조이다.
파이프-필터 패턴은 데이터를 여러 처리 단계로 흘려보내며 순차적으로 변환하는 구조이다.

**선지별 해설**

① X: MVC는 사용자 인터페이스 구조를 분리하는 패턴이지 분리된 컴포넌트의 원격 서비스 호출을 중개하는 구조가 아니다. 틀린 부분은 브로커 패턴 설명에 MVC를 대응시킨 점이다. 올바르게는 MVC는 모델-뷰-컨트롤러의 역할 분리를 통해 화면과 로직의 결합을 낮추는 패턴이다.
② X: 계층 패턴은 프레젠테이션층, 비즈니스층, 데이터층처럼 수직 계층을 나누는 구조이다. 틀린 부분은 CORBA와 같은 원격 객체 중개 구조를 계층 패턴으로 본 점이다. 올바르게는 계층 패턴의 대표 예는 OSI 참조 모델이나 전형적인 3계층 시스템이다.
③ O: 브로커 패턴은 분리된 컴포넌트가 브로커를 통해 원격 서비스를 호출하며 상호작용하도록 만든다. CORBA는 ORB가 객체 요청을 중개하므로 브로커 패턴의 대표 사례에 해당한다.
④ X: 파이프-필터 패턴은 각 필터가 입력 데이터를 가공해 다음 필터로 넘기는 데이터 흐름 중심 구조이다. 틀린 부분은 원격 호출 중개 구조를 파이프-필터에 연결한 점이다. 올바르게는 파이프-필터는 컴파일러 단계 처리처럼 연속된 변환 파이프라인을 구성하는 패턴이다.

NoteBase

### 오답 포인트

MVC, 계층, 파이프-필터는 모두 대표적인 아키텍처 패턴이라 이름만 보고 고르기 쉽다. 그러나 발문의 핵심 표현은 "원격 서비스 호출"과 "CORBA"이므로 분산 객체 중개 여부를 먼저 잡아야 한다.

### 다시 나오면 이렇게 판단

선지나 지문에 CORBA, ORB, 원격 객체, 서비스 중개가 보이면 브로커 패턴을 먼저 떠올린다. 화면 책임 분리이면 MVC, 계층 분할이면 계층 패턴, 데이터 처리 단계 연결이면 파이프-필터로 구분한다.

### 암기 포인트

CORBA-ORB-원격 중개는 브로커 패턴의 핵심 연결고리이다.

NoteBase

**문 9. K 대학은 외국인 신입생의 증가에 대응하기 위해 다음과 같이 정보 시스템을 유지보수하려고 한다. (가), (나)에 해당하는 유지보수 유형을 바르게 연결한 것은?**

(가) DBMS가 다국어 지원이 가능한 제품으로 변경되어 이와 관련된 프로그램을 수정하고자 한다.
(나) 기존 비대면 수업 등록 기능에서 발생한 오류 사항을 수정하고자 한다.

① 수정(corrective), 예방(preventive)
② 예방(preventive), 수정(corrective)
③ 완전(perfective), 적응(adaptive)
④ 적응(adaptive), 수정(corrective)

**정답 4번**

**출제 포인트**

이 문제는 소프트웨어 유지보수의 네 유형인 수정, 적응, 완전, 예방 유지보수의 구분 기준을 묻는 문제이다.

**핵심 개념**

수정 유지보수는 이미 발견된 오류를 고치는 유지보수이다.
적응 유지보수는 운영 환경, 플랫폼, DBMS, 제도처럼 외부 환경이 바뀌어 그 변화에 맞추는 유지보수이다.
완전 유지보수는 성능 향상, 사용성 개선, 기능 개선처럼 품질이나 효용을 높이는 유지보수이다.
예방 유지보수는 아직 장애로 드러나지 않은 잠재 문제를 미리 제거해 미래 오류를 줄이는 유지보수이다.

NoteBase

**선지별 해설**

① X
(가)를 수정으로 본 점이 틀리다. (가)는 오류 수정이 아니라 DBMS가 다국어 지원 제품으로 변경된 외부 환경 변화에 맞추는 경우이므로 적응 유지보수이다.
(나)를 예방으로 본 점도 틀리다. 이미 비대면 수업 등록 기능에서 발생한 오류를 고치는 것이므로 수정 유지보수이다.

② X
(가)를 예방으로 본 점이 틀리다. 예방 유지보수는 잠재 결함을 미리 없애는 활동이고, (가)는 변경된 DBMS 환경에 맞춰 프로그램을 수정하는 적응 유지보수이다.
(나)는 기존 기능에서 발생한 오류를 수정하는 것이므로 수정 유지보수라는 점은 맞다. 다만 (가) 때문에 전체 선지는 틀리다.

③ X
(가)를 완전으로 본 점이 틀리다. 완전 유지보수는 성능이나 편의성, 기능 개선이 중심인데, (가)는 외부 DBMS 환경 변화 대응이므로 적응 유지보수이다.
(나)를 적응으로 본 점도 틀리다. (나)는 환경 변화 대응이 아니라 이미 발생한 오류 수정이므로 수정 유지보수이다.

④ O
(가)는 DBMS 변경이라는 환경 변화에 맞추는 것이므로 적응 유지보수이다.
(나)는 기존 비대면 수업 등록 기능의 오류를 고치는 것이므로 수정 유지보수이다.

**오답 포인트**

(가)에서 다국어 지원이 추가된다는 표현만 보고 기능 향상처럼 읽으면 완전 유지보수로 오인하기 쉽다. 그러나 판단 기준은 기능이 좋아졌는지가 아니라 변경 원인이 외부 환경 변화인지 여부이다. (나)도 시스템 안정성을 높인다고 해서 예방으로 보기 쉬우나, 이미 발생한 오류를 고치는 순간 수정 유지보수로 본다.

**다시 나오면 이렇게 판단**

유지보수 유형은 변경의 목적과 원인으로 구분한다. 이미 발생한 오류를 고치면 수정, 외부 환경 변화에 맞추면 적응, 성능·기능을 더 좋게 만들면 완전, 미래 장애를 막으려고 미리 손보면 예방으로 판단한다.

**암기 포인트**

오류가 이미 났으면 수정, 환경이 바뀌었으면 적응, 더 좋게 만들면 완전, 미리 막으면 예방이다.

NoteBase

**문 10. 프로그램의 내부 구조를 참조하지 않고, 요구사항 명세서를 기반으로 입력 데이터를 선정한 후 출력 결과를 확인하는 테스트 기법에 해당하지 않는 것은?**

① 경곗값 분석(boundary value analysis)

② 동등분할(equivalence partitioning)

③ 원인-결과 그래프(cause-effect graph)

④ 자료 흐름(data flow)

**정답 4번**

**출제 포인트**

이 문제는 블랙박스 테스트 기법과 화이트박스 테스트 기법의 구분을 묻는 문제이다.

**핵심 개념**

블랙박스 테스트는 프로그램의 내부 구조를 보지 않고 요구사항과 입력·출력 관계를 기준으로 테스트 케이스를 설계하는 기법이다.
화이트박스 테스트는 프로그램 내부 논리, 제어 흐름, 데이터 흐름처럼 코드 구조를 기준으로 경로와 사용 관계를 점검하는 기법이다.
경곗값 분석, 동등분할, 원인-결과 그래프는 블랙박스 테스트 기법이고, 자료 흐름 테스트는 화이트박스 테스트 기법이다.

NoteBase

**선지별 해설**

① O
경곗값 분석은 입력값의 최소값, 최대값, 경계 바로 안팎 값을 중심으로 결함을 찾는 기법이므로 요구사항 기반 블랙박스 테스트에 해당한다.

② O
동등분할은 같은 방식으로 처리될 입력 집합을 동치 클래스로 나누어 대표값을 뽑는 기법이므로 내부 코드가 아니라 입력 조건을 기준으로 하는 블랙박스 테스트이다.

③ O
원인-결과 그래프는 입력 조건과 출력 결과의 논리 관계를 그래프로 정리해 테스트 케이스를 도출하는 기법이므로 명세 기반 블랙박스 테스트에 해당한다.

④ X
자료 흐름 테스트는 변수의 정의, 사용, 소멸 같은 데이터 흐름 경로를 코드 내부에서 추적하는 기법이므로 블랙박스 테스트에 해당하지 않는다.
틀린 표현은 블랙박스 테스트 기법으로 보는 부분이다.
올바르게는 자료 흐름 테스트는 화이트박스 테스트 기법이다.

**오답 포인트**

원인-결과 그래프와 자료 흐름은 모두 '흐름'이나 '관계'를 다루는 말처럼 보여 혼동하기 쉽다. 그러나 원인-결과 그래프는 입력 조건과 결과의 논리 관계를 명세 수준에서 다루고, 자료 흐름은 코드 내부 변수의 정의-사용 경로를 다룬다는 점이 핵심 차이이다.

**다시 나오면 이렇게 판단**

선지가 입력 조건, 출력 결과, 명세, 동치 클래스, 경계값처럼 외부 동작 기준이면 블랙박스로 판단할 수 있다.
반대로 제어 흐름, 자료 흐름, 경로, 조건 분기처럼 코드 내부 구조 기준이면 화이트박스로 판단할 수 있다.

**암기 포인트**

명세와 입출력은 블랙박스, 경로와 자료 흐름은 화이트박스이다.

NoteBase

**문 11. 비기능적 요구사항에 해당하는 것은?**

① 전자레인지 시스템은 쿠킹 중일 때 남은 쿠킹 시간을 계산하여 표시한다.

② 프로그램은 400MB 이내의 메모리를 사용해야 한다.

③ 입력한 문서의 음절, 단어, 문장, 줄, 문단, 문장당 단어의 수를 분석하여 표 형태로 보여 줄 수 있어야 한다.

④ 예약 시스템은 특정일에 예약한 고객의 리스트를 출력한다.

**정답 2번**

**출제 포인트**

이 문제는 기능적 요구사항과 비기능적 요구사항을 구분할 수 있는지를 묻는다.

**핵심 개념**

기능적 요구사항은 시스템이 무엇을 해야 하는지를 정의한 요구사항이다.
비기능적 요구사항은 시스템이 어느 수준의 성능, 자원 사용, 품질 속성으로 동작해야 하는지를 정의한 요구사항이다.
이 문항의 판단축은 기능의 존재 여부가 아니라 품질 제약이나 성능 기준의 제시 여부이다.

**선지별 해설**

1번 X: 전자레인지가 남은 쿠킹 시간을 계산하여 표시하는 것은 시스템이 제공해야 할 기능이다. 틀린 부분은 이를 비기능적 요구사항으로 볼 수 있다는 해석이다. 올바르게는 기능적 요구사항이다.
2번 O: 프로그램이 400MB 이내의 메모리를 사용해야 한다는 것은 자원 사용량에 대한 제약이다. 이는 시스템이 어떤 품질 수준으로 동작해야 하는지를 정한 비기능적 요구사항이다.
3번 X: 문서의 음절, 단어, 문장, 줄, 문단, 문장당 단어 수를 분석하여 표 형태로 보여 주는 것은 시스템이 수행할 처리 기능을 설명한다. 틀린 부분은 이를 비기능적 요구사항으로 분류하는 점이다. 올바르게는 기능적 요구사항이다.
4번 X: 특정일에 예약한 고객 리스트를 출력하는 것은 예약 시스템의 출력 기능이다. 틀린 부분은 이를 비기능적 요구사항으로 분류하는 점이다. 올바르게는 기능적 요구사항이다.

**오답 포인트**

기능적 요구사항과 비기능적 요구사항을 구분할 때는 동작 자체인지, 그 동작이 지켜야 할 품질 제약인지부터 본다. 1번, 3번, 4번은 모두 시스템이 수행해야 할 기능이고, 2번만 메모리 사용량이라는 품질 제약을 제시한다.

**다시 나오면 이렇게 판단**

선지가 시스템이 무엇을 한다를 말하면 기능적 요구사항이고, 얼마나 빠르게 처리한다, 얼마나 적은 자원을 쓴다, 얼마나 안정적으로 동작한다처럼 품질 기준을 말하면 비기능적 요구사항이다.

NoteBase

**암기 포인트**

무엇을 한다는 기능적 요구사항이고, 어느 수준으로 동작해야 한다는 비기능적 요구사항이다.

NoteBase

**문 12. 요구사항 검증 항목에 대한 설명으로 옳지 않은 것은?**

① 완전성 - 사용자 요구사항이 빠짐없이 완전히 반영되었는가?

② 일관성 - 요구사항 간의 모순이나 충돌이 존재하는가?

③ 추적성 - 개발 결과와 요구사항 일치를 검증할 수 있는가?

④ 명확성 - 서술된 명세서의 내용이 모호하지 않고 간결하고 명쾌한가?

**정답 3번**

**출제 포인트**

이 문제는 요구사항 검증 항목의 의미를 구분하고, 특히 추적성과 검증가능성을 혼동하지 않는지를 묻는 문제이다.

**핵심 개념**

완전성은 필요한 사용자 요구사항이 빠짐없이 포함되었는지를 보는 항목이다.
일관성은 요구사항들 사이에 모순이나 충돌이 없는지를 보는 항목이다.
추적성은 각 요구사항의 출처와 후속 산출물 사이의 연결 관계를 추적할 수 있는지를 보는 항목이다.
명확성은 요구사항 표현이 모호하지 않고 이해하기 쉽게 서술되었는지를 보는 항목이다.
검증가능성은 구현 결과가 요구사항에 맞는지 시험이나 점검으로 확인할 수 있는지를 보는 항목이다.

**선지별 해설**

① O - 완전성은 사용자 요구사항이 누락 없이 반영되었는지를 확인하는 기준이므로 옳은 설명이다.
② O - 일관성은 요구사항 상호 간에 모순이나 충돌이 없는지를 확인하는 기준이므로 옳은 설명이다.
③ X - 틀린 부분은 '추적성'에 '개발 결과와 요구사항 일치 검증'을 연결한 표현이다. 이는 추적성이 아니라 검증가능성에 대한 설명이다. 추적성은 요구사항의 출처를 거슬러 올라가거나 설계·구현·테스트와의 연결을 따라갈 수 있는 성질로 고쳐야 한다.
④ O - 명확성은 명세 내용이 모호하지 않고 간결하며 이해 가능하게 표현되었는지를 보는 항목이므로 옳은 설명이다.

**오답 포인트**

③은 '추적성'과 '검증가능성'을 바꿔 놓은 전형적인 함정이다. 수험생이 '검증'이라는 단어만 보고 맞다고 판단하기 쉽지만, 추적성의 핵심 키워드는 출처와 연계 추적이며 시험 가능성은 검증가능성의 영역이다.

**다시 나오면 이렇게 판단**

요구사항 품질 속성 문제에서는 먼저 핵심 키워드를 대응시키면 된다. 누락 여부는 완전성, 모순 여부는 일관성, 출처와 연결 관계는 추적성, 시험으로 확인 가능 여부는 검증가능성, 표현의 모호성 여부는 명확성으로 판단한다.

NoteBase

**암기 포인트**

추적성은 출처와 연계, 검증가능성은 시험과 확인이다.

NoteBase

**문 13. 학사관리시스템에서 파악된 다음 코드 악취를 해결하기 위한 리팩토링 기법은?**

> ○ enrol() 메서드의 코드가 4,000라인이 넘어 가독성이 낮다.
> ○ study()와 credit() 메서드는 각각 200라인 정도지만, 두 메서드는 100라인 정도가 중복되어 있다.

① 메서드 추출(extract method)
② 위임 숨기기(hide delegate)
③ 중개자 제거(remove middle man)
④ 클래스 인라인화(inline class)

**정답 1번**

**출제 포인트**

이 문제는 코드 악취의 유형을 보고 그 악취를 직접 줄이는 리팩토링 기법을 연결할 수 있는지를 묻는 문제이다.

**핵심 개념**

메서드 추출은 긴 메서드의 일부 로직이나 여러 곳에 반복되는 공통 로직을 별도 메서드로 뽑아 가독성과 재사용성을 높이는 기법이다.
위임 숨기기는 클라이언트가 간접 객체를 직접 탐색하지 않도록 위임 관계를 감추는 기법이다.
중개자 제거는 단순 전달만 하는 불필요한 위임 메서드를 없애 직접 호출하게 만드는 기법이다.
클래스 인라인화는 책임이 거의 없는 작은 클래스를 다른 클래스로 합쳐 구조를 단순화하는 기법이다.

NoteBase

### 선지별 해설

① O: enrol()처럼 지나치게 긴 메서드는 여러 의미 단위로 쪼개 별도 메서드로 추출하면 읽기 쉬워진다. study()와 credit()의 100라인 중복도 공통 부분을 하나의 메서드로 추출해 두 메서드가 호출하게 만들 수 있으므로 두 악취를 함께 줄일 수 있다.
② X: 틀린 부분은 "위임 숨기기"이다. 위임 숨기기는 객체 탐색 경로를 감추는 데 쓰이며, 긴 메서드나 중복 코드 제거와 직접 연결되지 않는다. 올바르게는 긴 메서드와 중복 코드에는 메서드 추출을 적용한다.
③ X: 틀린 부분은 "중개자 제거"이다. 중개자 제거는 의미 없이 전달만 하는 메서드가 너무 많을 때 쓰는 기법이다. 이 문항의 문제는 전달 단계가 아니라 메서드 길이와 중복이므로 올바르게는 공통 로직을 추출해 메서드를 분해한다.
④ X: 틀린 부분은 "클래스 인라인화"이다. 클래스 인라인화는 책임이 거의 없는 클래스를 합칠 때 쓰는 기법이다. 메서드가 너무 길거나 두 메서드에 동일 코드가 반복되는 문제를 바로 해결하지 못한다. 올바르게는 긴 메서드와 중복 구간을 별도 메서드로 추출한다.

### 오답 포인트

수험생은 "리팩토링 기법 이름"만 보고 막연히 구조 단순화와 관련 있어 보이는 ③이나 ④를 고르기 쉽다. 그러나 문항의 핵심 단서는 "4,000라인이 넘어 가독성이 낮다"와 "100라인 정도가 중복되어 있다"이며, 둘 다 메서드 단위 분해와 공통 로직 추출로 해결되는 악취라는 점이 함정 포인트이다.

### 다시 나오면 이렇게 판단

문장에서 긴 메서드, 중복 코드, 가독성 저하가 보이면 먼저 메서드 추출을 떠올린다. 반대로 위임 숨기기와 중개자 제거는 객체 간 호출 경로 문제, 클래스 인라인화는 클래스 책임 배치 문제인지부터 구분하면 된다.

### 암기 포인트

긴 메서드와 중복 코드는 메서드 추출, 위임 문제는 hide delegate와 remove middle man, 작은 클래스 정리는 inline class이다.

NoteBase

**문 14. 나선형(spiral) 모델에서 수행하는 활동과 가장 거리가 먼 것은?**

① 개발(develop)

② 리스크 식별 및 해결(identify, resolve risks)

③ 목표 정의(determine objectives)

④ 유지보수(maintenance)

**정답 4번**

**출제 포인트**

이 문제는 나선형 모델의 반복 주기에서 수행하는 핵심 활동과 개발 완료 후의 운영 단계 활동을 구별하는 문제이다.

**핵심 개념**

나선형 모델은 반복을 돌면서 목표 정의, 리스크 식별 및 해결, 개발과 검증, 다음 단계 계획을 수행하는 위험 주도형 개발 모델이다.
유지보수는 소프트웨어가 배포된 뒤 운영 과정에서 발생하는 수정과 개선 활동으로, 나선형 모델의 기본 4대 활동에 직접 포함되는 항목은 아니다.

NoteBase

**선지별 해설**

① X
개발은 나선형 모델의 핵심 활동 중 하나이다. 나선형 모델은 각 반복에서 제품을 개발하고 검증한다.
틀린 표현은 '가장 거리가 먼 것'에 ①을 넣는 판단이다. 올바르게는 개발은 나선형 모델에 포함되는 활동이다.

② X
리스크 식별 및 해결은 나선형 모델을 대표하는 핵심 활동이다. 나선형 모델은 위험 분석과 대응을 중심으로 반복을 진행한다.
틀린 표현은 '가장 거리가 먼 것'에 ②를 넣는 판단이다. 올바르게는 리스크 식별 및 해결은 나선형 모델의 중심 활동이다.

③ X
목표 정의도 나선형 모델의 기본 활동에 해당한다. 각 반복의 시작에서 목표, 대안, 제약을 정한다.
틀린 표현은 '가장 거리가 먼 것'에 ③을 넣는 판단이다. 올바르게는 목표 정의는 나선형 모델의 출발 활동이다.

④ O
유지보수는 배포 이후의 운영 단계에서 주로 다루는 활동이다. 고전적인 나선형 모델의 4대 활동은 목표 정의, 리스크 식별 및 해결, 개발, 계획이므로 유지보수는 가장 거리가 먼 선택지이다.

**오답 포인트**

나선형 모델을 반복 개발이라는 큰 말로만 외우면 ① 개발만 떠올리고, ② 리스크 관리나 ③ 목표 정의를 놓치기 쉽다. 이 문항의 함정은 나선형 모델의 4대 활동을 정확히 기억하지 못하면 유지보수와 개발 활동을 같은 층위로 혼동하게 되는 점이다.

**다시 나오면 이렇게 판단**

개발 생명주기 모델 문제에서는 먼저 해당 모델의 반복 내부 활동인지, 배포 후 운영 단계 활동인지를 구분하면 된다. 목표 정의, 위험 분석, 개발, 계획처럼 반복 내부에서 수행되는 활동이면 나선형 모델과 가깝고, 유지보수처럼 운영 이후 활동이면 거리가 멀다.

**암기 포인트**

나선형 모델은 목표-위험-개발-계획의 반복이고, 유지보수는 배포 후 활동이다.

NoteBase

**문 15. 다음 설명에 해당하는 테스트는?**

시스템을 변경한 후 변경된 부분이 기존 시스템에 부작용을 일으키지 않는지를 조사하기 위해 이미 사용한 테스트 케이스를 재사용하여 실행하는 테스트

① 베타(beta) 테스트
② 스트레스(stress) 테스트
③ 신뢰성(reliability) 테스트
④ 회귀(regression) 테스트

**정답 4번**

**출제 포인트**

시스템 변경 후 기존 기능에 부작용이 없는지 확인하는 회귀 테스트의 정의를 묻는 문제이다.

**핵심 개념**

베타 테스트는 개발이 끝난 제품을 실제 사용자 환경에서 사용자가 시험해 보는 테스트이다.
스트레스 테스트는 시스템에 한계 이상의 부하를 주어 장애 대응과 안정성을 확인하는 테스트이다.
신뢰성 테스트는 일정 조건에서 시스템이 고장 없이 요구 기능을 지속적으로 수행하는지를 확인하는 테스트이다.
회귀 테스트는 프로그램을 수정하거나 변경한 뒤 기존 기능에 예상치 못한 영향이 없는지 기존 테스트 케이스를 다시 실행해 확인하는 테스트이다.

**선지별 해설**

① X: 틀린 표현은 변경 후 기존 시스템의 부작용 점검을 베타 테스트로 본 부분이다. 베타 테스트는 실제 사용자에게 사용하게 하여 실사용 환경의 문제를 찾는 테스트이다.
② X: 틀린 표현은 이미 사용한 테스트 케이스를 재사용해 변경 영향 여부를 확인하는 절차를 스트레스 테스트로 본 부분이다. 스트레스 테스트는 과도한 부하나 극한 조건에서 시스템의 한계와 안정성을 확인하는 테스트이다.
③ X: 틀린 표현은 변경 후 부작용 확인 절차를 신뢰성 테스트로 본 부분이다. 신뢰성 테스트는 일정 시간이나 조건에서 고장 없이 동작하는지를 검증하는 테스트이다.
④ O: 시스템 변경 후 변경된 부분이 기존 기능에 부작용을 일으키지 않는지 확인하기 위해 기존 테스트 케이스를 다시 실행하는 것은 회귀 테스트의 정의에 정확히 해당한다.

**오답 포인트**

변경 후 다시 돌린다는 표현만 보고 일반적인 재시험이나 안정성 시험으로 넓게 해석하면 오답이 된다. 이 문항의 핵심 단어는 '이미 사용한 테스트 케이스 재사용'과 '기존 시스템에 부작용'이다.

NoteBase

**다시 나오면 이렇게 판단**

수정·배포 이후 기존 테스트 케이스를 다시 실행해 영향 범위를 확인하면 회귀 테스트로 판단한다. 반대로 실사용자 참여는 베타 테스트, 과부하 부여는 스트레스 테스트, 장시간 무고장 확인은 신뢰성 테스트로 구분된다.

**암기 포인트**

회귀 테스트는 변경 후 기존 기능 재검증, 베타는 사용자 시험, 스트레스는 극한 부하, 신뢰성은 무고장 지속성이다.

NoteBase

**문 16. 스크럼(Scrum)에 대한 설명으로 옳지 않은 것은?**

① 매일 15분 정도의 짧은 일일 스크럼 회의를 한다.

② 스프린트(sprint)는 인덱스 카드를 이용하여 기능을 간략히 서술한 것이다.

③ 제품 백로그(product backlog)는 개발할 제품에 대한 요구사항 목록을 포함한다.

④ 소프트웨어를 반복적으로 개발하면서 사용자의 요구사항을 지속적으로 반영한다.

**정답 2번**

**출제 포인트**

이 문제는 스크럼의 핵심 구성요소인 스프린트, 제품 백로그, 일일 스크럼의 의미를 구분하는 문제이다.

**핵심 개념**

스크럼은 짧은 반복 주기인 스프린트를 통해 제품을 점진적으로 개발하는 애자일 프레임워크이다.
스프린트는 일정한 기간 안에 개발 목표를 수행하는 반복 주기이다.
제품 백로그는 제품에 필요한 요구사항과 기능 목록을 우선순위에 따라 정리한 것이다.
인덱스 카드에 기능을 간략히 적는 표현은 스프린트가 아니라 사용자 스토리나 백로그 항목의 설명 방식에 가깝다.

**선지별 해설**

① O: 일일 스크럼은 매일 약 15분 내외로 진행하는 짧은 회의이므로 맞는 설명이다.
② X: 틀린 부분은 '스프린트는 인덱스 카드를 이용하여 기능을 간략히 서술한 것'이라는 표현이다. 스프린트는 기능 설명 카드가 아니라 정해진 기간의 반복 개발 주기이다. 인덱스 카드로 기능을 간략히 적는 것은 사용자 스토리나 백로그 항목 설명에 해당한다.
③ O: 제품 백로그는 개발할 제품에 대한 요구사항 목록을 관리하는 저장소이므로 맞는 설명이다.
④ O: 스크럼은 반복적으로 소프트웨어를 개발하면서 사용자 요구사항을 지속적으로 반영하는 방식이므로 맞는 설명이다.

**오답 포인트**

스프린트와 사용자 스토리를 서로 바꿔 놓으면 헷갈리기 쉽다. 스프린트는 시간 단위의 반복 주기이고, 기능을 짧게 서술하는 것은 백로그 항목 또는 사용자 스토리라는 점이 핵심 함정이다.

**다시 나오면 이렇게 판단**

스크럼 선지를 보면 먼저 그것이 기간을 뜻하는지, 요구사항 목록을 뜻하는지, 회의 방식을 뜻하는지 구분하면 된다. 기간이면 스프린트, 목록이면 제품 백로그, 매일 15분 회의이면 일일 스크럼으로 판단한다.

NoteBase

**암기 포인트**

스프린트는 반복 기간, 백로그는 요구사항 목록, 일일 스크럼은 15분 회의이다.

NoteBase

**문 17. 다음 자바 코드에 적용된 디자인 패턴은?**

```
public class DesignManager {
      private DesignManager() { }
      private static DesignManager _instance = null;
      public synchronized
            static DesignManager Instance() {
            if (_instance == null)
                  _instance = new DesignManager();
            return _instance;
      }
}
```

① 데코레이터(decorator)

② 반복자(iterator)

③ 싱글톤(singleton)

④ 옵서버(observer)

**정답 3번**

**출제 포인트**

이 문제는 싱글톤, 데코레이터, 반복자, 옵서버의 핵심 구조를 구분할 수 있는지를 묻는 문제이다.

**핵심 개념**

싱글톤은 클래스의 인스턴스를 하나만 생성하고 전역에서 그 하나를 공유하는 패턴이다. 키워드는 private 생성자, static 인스턴스, 접근 메서드이다.
데코레이터는 기존 객체를 감싸서 기능을 동적으로 덧붙이는 패턴이다. 키워드는 wrapping, 동일 인터페이스, 기능 확장이다.
반복자는 집합 내부 표현을 노출하지 않고 원소를 순차 접근하게 하는 패턴이다. 키워드는 순회, next, hasNext이다.
옵서버는 상태 변화가 생기면 구독자들에게 통지하는 패턴이다. 키워드는 발행-구독, notify, 다수 객체 연계이다.

NoteBase

**선지별 해설**

① X: 데코레이터는 객체를 감싸 기능을 추가하는 구조가 있어야 하는데, 제시 코드에는 다른 객체를 포함하거나 감싸는 필드가 없다. 틀린 부분은 기능 확장용 래핑 구조로 보는 점이다. 올바르게는 private 생성자와 static 단일 인스턴스로 하나만 생성하는 싱글톤 구조이다.
② X: 반복자는 컬렉션 원소를 순회하는 hasNext(), next() 같은 접근 인터페이스가 핵심인데, 제시 코드에는 순회 대상도 순회 메서드도 없다. 틀린 부분은 순차 접근 제어 구조로 보는 점이다. 올바르게는 인스턴스 생성 수를 하나로 제한하는 싱글톤이다.
③ O: 생성자가 private이라 외부에서 직접 객체를 만들 수 없고, static 필드에 유일한 인스턴스를 저장한 뒤 접근 메서드에서 같은 객체를 반환한다. synchronized는 동시 접근 시 단일 인스턴스 생성을 보조하는 장치이므로 싱글톤의 전형적 형태이다.
④ X: 옵서버는 subject와 observer 사이의 등록, 통지, 갱신 구조가 있어야 하는데, 제시 코드에는 이벤트 통지나 구독자 목록이 없다. 틀린 부분은 상태 변화 통지 구조로 보는 점이다. 올바르게는 단 하나의 객체를 공유하는 싱글톤이다.

**오답 포인트**

private 생성자만 보고 단순한 객체 생성 제어로 읽거나, synchronized만 보고 동기화 기법으로만 읽으면 패턴 분류를 놓치기 쉽다. 이 문제의 핵심은 생성 제한, static 단일 인스턴스 저장, 동일 객체 반환이 한 세트로 나타난다는 점이다.

**다시 나오면 이렇게 판단**

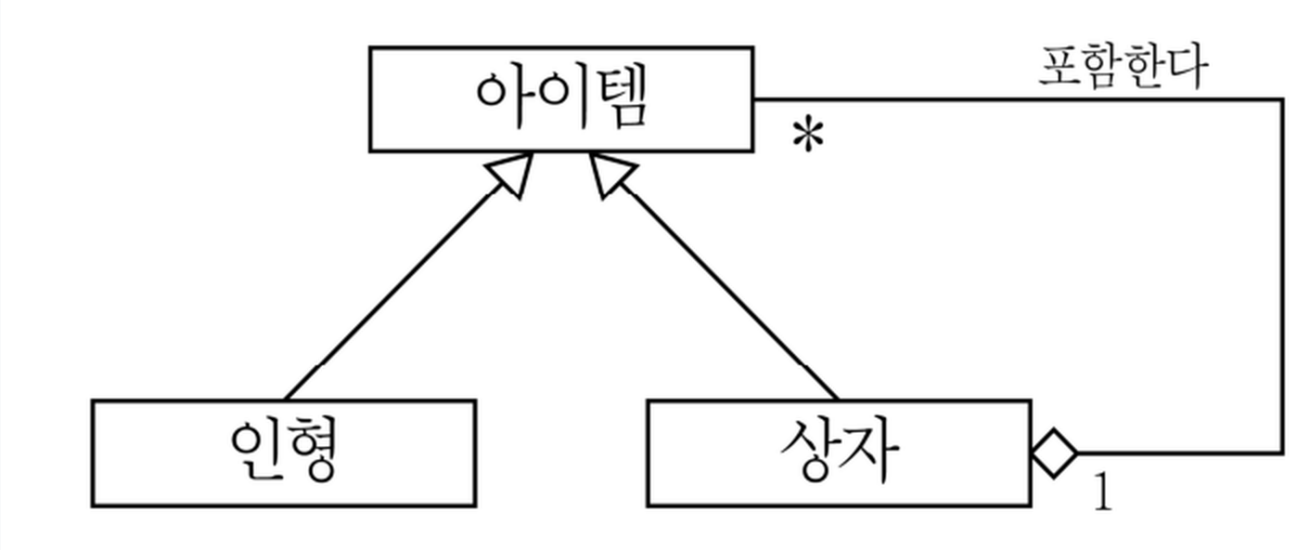
생성자가 private인지, 클래스 내부에 static 인스턴스 저장소가 있는지, 접근 메서드가 항상 그 하나를 반환하는지를 먼저 본다. 이 세 요소가 함께 보이면 싱글톤으로 판단한다.

**암기 포인트**

private 생성자 + static 단일 객체 + 공용 접근 메서드이면 싱글톤이다.

NoteBase

**문 18. 다음 클래스 다이어그램에 대한 설명으로 옳지 않은 것은?**




① 인형은 상자를 포함할 수 없다.

② 상자와 인형은 아이템의 종류이다.

③ 상자는 인형 또는 상자를 하나 이상 포함해야 한다.

④ 상자가 인형을 포함한 상자를 포함할 수 있다.

**정답 3번**

**출제 포인트**

이 문제는 UML 클래스 다이어그램에서 일반화 관계와 집합관계의 다중성을 함께 읽어 상자가 무엇을 몇 개까지 포함할 수 있는지를 판단하는 문제이다.

NoteBase

**선지별 해설**

① O
인형은 아이템의 하위 클래스이지만, 포함 관계의 집합자 쪽은 상자이다.
따라서 이 다이어그램에서 상자가 아이템을 포함하며, 인형이 상자를 포함하는 관계는 정의되어 있지 않다.

② O
인형과 상자에서 아이템으로 향하는 빈 삼각형 화살표는 일반화 관계이다.
따라서 상자와 인형은 모두 아이템의 종류이다.

③ X
상자와 아이템 사이의 다중성 *는 0개 이상을 뜻한다.
틀린 표현은 '하나 이상'이며, 올바르게는 '상자는 인형 또는 상자를 0개 이상 포함할 수 있다'이다.

④ O
상자도 아이템의 하위 클래스이므로 상자는 포함 대상인 아이템에 해당한다.
따라서 한 상자가 다른 상자를 포함하고, 그 안의 상자가 다시 인형을 포함하는 구조가 가능하다.

**오답 포인트**

③의 '하나 이상'이 함정이다. UML 다중성에서 *는 0개 이상을 뜻하므로 1개 이상으로 읽으면 안 된다. 또한 상자가 아이템의 하위 클래스라는 점을 놓치면 ④도 틀리기 쉽다.

**다시 나오면 이렇게 판단**

빈 삼각형은 상속, 빈 마름모는 집합관계, 끝에 적힌 기호는 다중성으로 읽는다. *는 0개 이상, 1은 정확히 1개로 해석하면 선지의 참거짓을 바로 가를 수 있다.

**암기 포인트**

UML에서 *는 0..*이고, 하위 클래스는 상위 클래스 자리에 포함될 수 있다.

NoteBase

**문 19. 기능 점수(function point)에 대한 설명으로 옳지 않은 것은?**

① 2022년 SW사업 대가산정 가이드에서 기능 점수 산정의 정통법은 소프트웨어 기능을 도출한 후 각 기능에 평균 복잡도를 적용한다.

② 계획 단계뿐 아니라 분석, 설계, 구현 단계에서도 사용할 수 있어 단계가 진행됨에 따라 더 정확한 기능 점수를 측정할 수 있다.

③ 구현 기술, 구현 언어, 개발 도구, 개발자의 능력에 상관없이 소프트웨어의 규모 측정에 일관성을 제공한다.

④ 소프트웨어의 기능을 정량화해 개발 비용 산정에 활용한다.

**정답 1번**

**출제 포인트**

이 문제는 기능점수 정통법과 ITO 기능점수 방식의 차이, 그리고 기능점수 방식의 일반적 특징을 구분하는 문제이다.

**핵심 개념**

기능점수 정통법은 사용자가 요구한 기능을 데이터기능과 트랜잭션기능으로 나누고, 각 기능의 복잡도를 세부 기준으로 판정해 기능점수를 산정하는 방식이다.
ITO 기능점수 방식은 개선관리 서비스에서 고객이 인지하는 산출물 기준의 기능 개수에 정통법의 평균가중치를 적용해 규모를 재는 간략화된 방식이다.
기능점수 방식 자체는 구현 방법이 아니라 사용자가 요구한 기능을 기준으로 소프트웨어 규모를 측정하는 사용자 관점의 정량 기법이다.

**선지별 해설**

① X: 틀린 부분은 '정통법은 소프트웨어 기능을 도출한 후 각 기능에 평균 복잡도를 적용한다'는 표현이다. 2022 SW사업 대가산정 가이드에서 고객이 인지하는 산출물 기준의 기능 개수에 정통법의 평균가중치를 적용하는 것은 ITO 기능점수 방식이다. 올바르게는 정통법은 각 기능의 세부 복잡도를 판정해 가중치를 적용하고, 평균가중치를 쓰는 방식은 ITO 기능점수 방식이라고 보아야 한다.
② O: 기능점수는 개발 이전에도 업무량을 측정할 수 있고, 개발은 물론 기획과 운영 등 전 수명주기에서 측정 가능하므로 계획 단계에만 한정되지 않는다. 단계가 구체화될수록 기능 식별이 선명해져 측정 정확도도 높아질 수 있다.
③ O: 기능점수는 '어떻게 구현했는가'가 아니라 '사용자가 어떤 기능을 요구했는가'를 기준으로 측정한다. 따라서 구현 기술, 구현 언어, 개발 도구, 개발자 숙련도 같은 공급자 측 요소와 무관하게 규모를 일관성 있게 측정하는 취지에 부합한다.
④ O: 기능점수는 사용자 관점의 기능 규모를 정량화하는 방법이므로 소프트웨어 개발과 유지관리의 비용 및 자원 소요 산정에 활용된다. 개발 비용 산정의 입력값으로 쓰인다는 설명이 맞다.

NoteBase

**오답 포인트**

①은 '평균 복잡도'라는 표현 때문에 그럴듯해 보이지만, 평균가중치를 적용하는 것은 정통법이 아니라 ITO 기능점수 방식이다. 정통법, 간이화 방식, ITO 방식의 문구를 섞어 놓으면 혼동하기 쉽다.

**다시 나오면 이렇게 판단**

기능점수 문제에서는 먼저 기준 축을 본다. 사용자 요구 기능을 세부 복잡도로 직접 판정하면 정통법이고, 산출물 기준 기능 개수에 평균가중치를 씌우면 ITO 같은 간략화 방식으로 판단할 수 있다. 이어서 기능점수의 일반 특징은 사용자 관점, 구현기술 독립성, 비용 산정 활용으로 정리하면 된다.

**암기 포인트**

정통법은 세부 복잡도 판정, ITO 방식은 평균가중치 적용, 기능점수는 사용자 기능 기준 규모 측정이다.

NoteBase

**문 20. 다음 자바 코드를 클래스 다이어그램으로 바르게 표현한 것은?**

```
class Student {
      private String name;
      protected String department;
      private static int nextAvailID = 1;
      protected void passed(Course theCourse) {...}
      public static void getNextAvailID() {...}
}
```

① 

| Student |
| --- |
| +name: String<br>#department: String<br>-nextAvailID: int = 1 |
| +passed(theCourse: Course)<br>+getNextAvailID() |

② 

| Student |
| --- |
| -name: String<br>#department: String<br>-nextAvailID: int = 1 |
| #passed(theCourse: Course)<br>+getNextAvailID() |

③ 

| Student |
| --- |
| -name: String<br>-department: String<br>-nextAvailID: int = 1 |
| #passed(theCourse: Course)<br>+getNextAvailID() |

④ 

| Student |
| --- |
| -name: String<br>#department: String<br>-nextAvailID: int = 1 |
| #passed(theCourse: Course)<br>+getNextAvailID() |

**정답 2번**

NoteBase

**출제 포인트**

이 문제는 자바 코드의 접근 제어자와 static 여부를 UML 클래스 다이어그램 기호로 정확히 옮길 수 있는지를 묻는 문제이다.

**선지별 해설**

1번: X. `name`은 `private`이므로 UML에서 `-name: String`이어야 하는데, 1번은 `+name: String`으로 표시하여 공개 범위를 잘못 나타냈다. 또한 `passed`는 `protected`이므로 `#passed(theCourse: Course)`가 되어야 하는데, 1번은 `+passed(theCourse: Course)`로 표시하였다.
2번: O. `name`은 private이므로 `-`, `department`는 protected이므로 `#`로 표시하는 것이 맞다. `nextAvailID`와 `getNextAvailID()`는 `static`이므로 밑줄로 표시하는 것도 맞고, 각각 private와 public 접근 수준도 코드와 일치한다. `passed` 역시 protected 메서드이므로 `#passed(theCourse: Course)`가 맞다.
3번: X. `department`는 `protected`인데 3번은 `-department: String`으로 표시하였다. 틀린 표현은 `-department`이고, 올바르게는 `#department: String`이다.
4번: X. `nextAvailID` 필드와 `getNextAvailID()` 메서드는 모두 `static`인데 4번은 밑줄이 없어 정적 멤버 표현이 빠져 있다. 틀린 표현은 정적 표기의 누락이고, 올바르게는 `-nextAvailID: int = 1`과 `+getNextAvailID()`를 밑줄로 표시해야 한다.

**오답 포인트**

접근 제어자 기호와 static 표기를 따로 보지 않고 이름만 맞추면 틀리기 쉽다. 특히 `protected`를 `private`로, `static` 멤버의 밑줄 표시를 일반 멤버처럼 지나치면 3번이나 4번으로 흔들리게 된다.

**다시 나오면 이렇게 판단**

클래스 다이어그램 문제는 멤버마다 접근 제어자와 static 여부를 한 줄씩 대응시켜 판단한다. `private=-`, `protected=#`, `public=+`를 먼저 대입하고, `static`이 붙은 필드와 메서드는 밑줄이 있는지 추가로 확인하면 된다.

**암기 포인트**

UML 클래스 다이어그램은 `- private`, `# protected`, `+ public`, 밑줄은 `static`이다.

NoteBase

**문 21. 다음 소스 코드에 대해 조건(condition) 커버리지와 분기(branch) 커버리지를 각각 100 % 만족하는 테스트 케이스 집합은? (단, 단축 연산(short-circuit evaluation)은 수행하지 않는다)**

```
if (x >= 0 || y < 4)
    z = x + y;
else
    z = x - y;
```

① {(x = -1, y = 2), (x = 1, y = 6)}

② {(x = -1, y = 2), (x = 2, y = 2)}

③ {(x = 1, y = 6), (x = -2, y = 6)}

④ {(x = 2, y = 2), (x = -2, y = 6)}

**정답 4번**

**출제 포인트**

이 문제는 조건 커버리지와 분기 커버리지의 차이를 이해하고, 주어진 테스트 케이스 집합이 두 기준을 모두 만족하는지 판정하는 문제이다.

**핵심 개념**

조건 커버리지는 복합 조건을 이루는 각 단일 조건이 참과 거짓을 모두 한 번 이상 갖는지를 보는 기준이다.
분기 커버리지는 전체 if 조건식의 결과가 참 분기와 거짓 분기를 모두 한 번 이상 수행하는지를 보는 기준이다.
이 문항에서는 단축 연산을 하지 않으므로 $x \geq 0$와 $y < 4$를 각각 독립된 조건으로 보고, 전체 조건식 $(x \geq 0)||(y < 4)$의 참·거짓도 함께 확인해야 한다.

NoteBase

### 선지별 해설

① X
첫 번째 케이스 ($x = -1, y = 2$)는 $x \geq 0$이 거짓, $y < 4$가 참이므로 전체 조건식은 참이다. 두 번째 케이스 ($x = 1, y = 6$)는 $x \geq 0$이 참, $y < 4$가 거짓이므로 전체 조건식은 역시 참이다.
따라서 각 단일 조건은 참과 거짓을 모두 가져 조건 커버리지는 만족하지만, 전체 조건식이 한 번도 거짓이 되지 않아 else 분기를 수행하지 못한다.
틀린 판단은 두 기준을 모두 만족한다고 보는 점이다. 올바르게는 조건 커버리지만 100%이고 분기 커버리지는 100%가 아니다.

② X
첫 번째 케이스 ($x = -1, y = 2$)는 거짓/참, 두 번째 케이스 ($x = 2, y = 2$)는 참/참이다. 이 경우 $y < 4$는 두 번 모두 참이므로 조건 커버리지부터 충족하지 못한다.
또한 전체 조건식도 두 경우 모두 참이어서 else 분기가 실행되지 않는다.
틀린 판단은 두 기준을 만족한다고 보는 점이다. 올바르게는 조건 커버리지와 분기 커버리지를 모두 만족하지 못한다.

③ X
첫 번째 케이스 ($x = 1, y = 6$)는 참/거짓이어서 전체 조건식이 참이고, 두 번째 케이스 ($x = -2, y = 6$)는 거짓/거짓이어서 전체 조건식이 거짓이다. 따라서 분기 커버리지는 만족한다.
하지만 $y < 4$는 두 번 모두 거짓이므로 이 단일 조건이 참을 한 번도 갖지 못해 조건 커버리지는 만족하지 못한다.
틀린 판단은 두 기준을 모두 만족한다고 보는 점이다. 올바르게는 분기 커버리지만 100%이고 조건 커버리지는 100%가 아니다.

④ O
첫 번째 케이스 ($x = 2, y = 2$)는 $x \geq 0$이 참이고 $y < 4$도 참이어서 전체 조건식이 참이다. 두 번째 케이스 ($x = -2, y = 6$)는 $x \geq 0$이 거짓이고 $y < 4$도 거짓이어서 전체 조건식이 거짓이다.
따라서 $x \geq 0$는 참과 거짓을 모두 갖고, $y < 4$도 참과 거짓을 모두 갖는다. 동시에 전체 조건식도 참과 거짓을 모두 만들어 if와 else 분기를 모두 실행하므로 조건 커버리지와 분기 커버리지를 각각 100% 만족한다.

### 오답 포인트

가장 흔한 착각은 각 단일 조건의 참·거짓만 확인하고 전체 조건식의 참·거짓을 따로 보지 않는 점이다. ①은 조건 커버리지는 만족하지만 전체 조건식이 모두 참이라 else 분기가 실행되지 않는다. ③은 분기 커버리지는 만족하지만 $y < 4$가 한 번도 참이 되지 않아 조건 커버리지가 부족하다.

NoteBase

### 다시 나오면 이렇게 판단

먼저 각 테스트 케이스에서 $x \geq 0$, $y < 4$, 전체 조건식의 결과를 표처럼 나열해 본다. 그다음 단일 조건마다 참·거짓이 모두 나왔는지로 조건 커버리지를 판단하고, 전체 조건식이 참과 거짓을 모두 만들었는지로 분기 커버리지를 따로 판단하면 된다.

### 암기 포인트

조건 커버리지는 각 원자 조건의 참·거짓, 분기 커버리지는 전체 조건식의 참·거짓이다.

NoteBase

**문 22. 다음 설명에 해당하는 UML 다이어그램은?**

◯ 객체가 수행하는 작업의 흐름을 표현한다.
◯ 비즈니스 프로세스와 같은 고수준의 흐름을 표현한다.
◯ 병행 처리를 표현하기 위해 포크와 조인을 이용한다.

① 배치(deployment) 다이어그램
② 액티비티(activity) 다이어그램
③ 컴포넌트(component) 다이어그램
④ 패키지(package) 다이어그램

**정답 2번**

**출제 포인트**

이 문제는 액티비티 다이어그램의 용도와 이를 배치, 컴포넌트, 패키지 다이어그램과 구별하는 기준을 묻는다.

**핵심 개념**

액티비티 다이어그램: 작업과 제어 흐름을 표현하는 행위 중심 다이어그램이다 / 결정 차이축: 무엇의 흐름을 그리는가 / 키워드: 업무 흐름, 비즈니스 프로세스, 포크, 조인
배치 다이어그램: 소프트웨어 산출물이 하드웨어 노드에 어떻게 배치되는지를 표현한다 / 결정 차이축: 어디에 배치되는가 / 키워드: 노드, 실행 환경
컴포넌트 다이어그램: 시스템을 이루는 컴포넌트와 인터페이스의 구조적 관계를 표현한다 / 결정 차이축: 어떤 구성요소가 어떻게 연결되는가 / 키워드: 컴포넌트, 인터페이스
패키지 다이어그램: 모델 요소를 묶는 패키지와 패키지 간 의존관계를 표현한다 / 결정 차이축: 어떻게 그룹화되는가 / 키워드: 패키지, 의존관계

NoteBase

### 선지별 해설

1번 X: 배치 다이어그램은 소프트웨어나 컴포넌트가 노드에 배치되는 물리적 구조를 표현한다. 틀린 부분은 다이어그램 이름을 배치로 본 점이다. 실제로는 객체나 업무의 작업 흐름, 병행 처리, 포크와 조인을 나타내므로 액티비티 다이어그램이다.
2번 O: 액티비티 다이어그램은 비즈니스 프로세스처럼 고수준의 흐름을 표현할 수 있다. 작업 순서와 분기, 병행 처리까지 나타내므로 제시된 설명과 일치한다.
3번 X: 컴포넌트 다이어그램은 구현 단위인 컴포넌트와 그 인터페이스 관계를 표현한다. 틀린 부분은 병행 처리를 포크와 조인으로 표현한다는 설명을 컴포넌트 다이어그램에 연결한 점이다. 실제로 포크와 조인은 액티비티 다이어그램의 표기이다.
4번 X: 패키지 다이어그램은 관련 요소를 묶고 패키지 사이의 의존관계를 정리하는 데 쓰인다. 틀린 부분은 흐름 표현용 다이어그램을 패키지 다이어그램으로 본 점이다. 실제로 흐름과 병행 제어를 표현하는 것은 액티비티 다이어그램이다.

### 오답 포인트

배치, 컴포넌트, 패키지 다이어그램은 모두 구조를 보는 정적 관점이고, 액티비티 다이어그램은 흐름을 보는 동적 관점이라는 점을 섞으면 틀린다. 특히 3번째 설명의 포크와 조인은 액티비티 다이어그램의 대표 키워드이다.

### 다시 나오면 이렇게 판단

선지 설명에 흐름, 작업 순서, 비즈니스 프로세스, 분기, 포크, 조인 같은 말이 나오면 먼저 액티비티 다이어그램을 떠올린다. 반대로 노드 배치는 배치, 구성요소와 인터페이스는 컴포넌트, 그룹화와 의존관계는 패키지로 판단한다.

### 암기 포인트

액티비티는 흐름, 배치는 노드, 컴포넌트는 구성요소, 패키지는 묶음이다.

NoteBase

**문 23. 작업 분할 구조도(WBS, Work Breakdown Structure)에 대한 설명으로 옳지 않은 것은?**

① 프로젝트에서 수행해야 하는 모든 작업을 포함해야 한다.

② 작업 프로세스 중심으로 작성할 수도 있고, 산출물 위주로 작성할 수도 있다.

③ 작업 패키지(work package)는 계층 구조 상위에 있는 여러 작업을 묶은 관리 단위이다.

④ 프로젝트 목표를 달성하고 결과물을 산출하기 위해 수행해야 할 작업을 계층적으로 분할한 것이다.

**정답 3번**

**출제 포인트**

이 문제는 WBS의 기본 원칙과 작업 패키지의 위치를 구별하는지를 묻는 문제이다.

**핵심 개념**

WBS는 프로젝트 목표 달성과 결과물 산출에 필요한 전체 작업을 계층적으로 분할한 구조이다.
작업 패키지는 WBS 최하위 수준에서 일정·원가·책임을 배정할 수 있을 정도로 구체화된 관리 단위이다.
WBS는 산출물 중심으로도 만들 수 있고, 단계·프로세스 중심으로도 만들 수 있다.

**선지별 해설**

1번 O: WBS는 프로젝트에서 수행해야 할 전체 범위를 빠짐없이 포함해야 한다. 이는 WBS의 완전성 원칙에 해당한다.
2번 O: WBS는 작업 프로세스 중심으로 구성할 수도 있고, 산출물 중심으로 구성할 수도 있다. 분해 기준은 하나로 고정되지 않는다.
3번 X: 틀린 부분은 "계층 구조 상위"이다. 작업 패키지는 상위 관리 단위가 아니라 WBS의 최하위 수준 관리 단위이다. 올바르게는 "작업 패키지는 계층 구조 최하위에서 일정과 원가를 관리할 수 있도록 정의한 작업 단위이다."라고 본다.
4번 O: WBS는 프로젝트 목표 달성과 결과물 산출을 위해 필요한 작업을 계층적으로 분할한 구조라는 설명이므로 옳다.

**오답 포인트**

작업 패키지를 단순히 "여러 작업을 묶은 덩어리"로만 기억하면 상위 묶음이라고 착각하기 쉽다. 이 문항은 작업 패키지의 핵심이 "상위 묶음"이 아니라 "최하위 관리 가능 단위"라는 점을 묻는다.

**다시 나오면 이렇게 판단**

WBS 관련 선지는 먼저 전체 범위를 빠짐없이 포함하는지, 다음으로 작업 패키지가 최하위 관리 단위인지 확인하면 빠르게 판단할 수 있다. "상위"라는 표현이 나오면 작업 패키지와 충돌하는지 먼저 본다.

NoteBase

**암기 포인트**

WBS는 전체 작업의 계층 분할이고, 작업 패키지는 최하위 관리 단위이다.

NoteBase

**문 24. 클래스 B 객체를 생성하면서 전달한 인자의 값을 상속을 이용하여 화면에 출력하기 위한 C++ 코드 중 오류가 있는 코드의 라인 번호만을 모두 고르면? (단, 밑줄 친 라인 번호의 코드 이외에는 오류가 없다)**

| 라인 번호 | C++ 코드 |
|---|---|
| 1 | #include<iostream> |
| 2 | using namespace std; |
| 3 | |
| 4 | class A { |
| 5 | private: |
| 6 | int size; |
| 7 | public: |
| 8 | A(int size) { size = size; }; //오류 확인 |
| 9 | virtual void print() { cout << size; }; |
| 10 | }; |
| 11 | |
| 12 | class B : public A { |
| 13 | public: |
| 14 | B(int size) : A(size) { }; |
| 15 | void print() { cout << size; }; //오류 확인 |
| 16 | }; |
| 17 | |
| 18 | int main() { |
| 19 | A *b = new B(10); |
| 20 | b.print(); //오류 확인 |
| 21 | return 0; |
| 22 | } |

① 15

② 8, 15

③ 8, 20

④ 8, 15, 20

**정답 4번**

### 출제 포인트

이 문제는 상속된 클래스의 생성자 초기화, private 멤버 접근, 객체 포인터의 멤버 호출 방식을 코드에 적용해 오류 라인을 판별할 수 있는지를 묻는다.

### 사례 정리

A 클래스에는 private 멤버 size와 이를 출력하는 virtual print()가 있다.
B 클래스는 A를 public 상속하며 생성자에서 A(size)를 호출한다.
main에서는 A *b = new B(10);으로 객체를 생성한 뒤 전달한 값이 화면에 출력되기를 기대한다.

NoteBase

### 핵심 개념

생성자에서 멤버와 매개변수 이름이 같으면 this 포인터나 초기화 목록으로 구분해야 실제 멤버가 초기화된다. private 멤버는 선언한 클래스 내부에서만 직접 접근할 수 있으므로 자식 클래스 B에서 A::size를 직접 사용할 수 없다.
포인터가 가리키는 객체의 멤버 함수는 .이 아니라 ->로 호출한다.

### 적용 기준

각 밑줄 라인이 전달값 10을 정상적으로 저장하고 출력하는 흐름을 깨는지 확인하면 된다. 저장 단계에서는 멤버 초기화가 실제로 되었는지, 상속 단계에서는 접근 제한이 맞는지, 호출 단계에서는 포인터 문법이 맞는지를 차례로 본다.

### 선지별 해설

1번 X: 15번만 고르면 부족하다. 15번은 B::print()에서 A의 private 멤버 size를 직접 출력하므로 접근 위반 오류가 맞다. 그러나 8번도 생성자에서 size = size;로 자기 자신에게 대입해 멤버 size가 초기화되지 않는 오류이고, 20번도 포인터 b에 대해 b.print()를 사용한 문법 오류이다. 올바르게는 8, 15, 20이다.
2번 X: 8번과 15번은 오류가 맞지만 20번이 빠져 있다. 20번은 A *b가 포인터인데 . 연산자로 멤버를 호출했으므로 잘못이다. 올바르게는 b->print();로 써야 한다.
3번 X: 8번과 20번은 오류가 맞지만 15번이 빠져 있다. 15번의 cout << size;는 B가 상속받았더라도 private 멤버 size에 직접 접근할 수 없으므로 컴파일되지 않는다. 올바르게는 A에서 protected로 바꾸거나, A의 public/protected 접근 함수를 통해 출력해야 한다.
4번 O: 8번, 15번, 20번은 모두 오류이다. 8번은 멤버 초기화 실패, 15번은 private 멤버 직접 접근, 20번은 포인터 멤버 호출 문법 오류이므로 이 조합이 맞다.

### 오답 포인트

8번을 문법 오류가 아니라는 이유로 빼기 쉽고, 15번은 상속받았으니 접근 가능하다고 착각하기 쉽다. 또한 19번에서 b가 포인터라는 점을 놓치면 20번의 . 사용을 지나치기 쉽다.

### 다시 나오면 이렇게 판단

객체지향 코드 오류 문제는 생성자 초기화, 멤버 접근 제어, 호출 대상의 자료형을 순서대로 본다. 변수명이 같으면 실제 멤버가 바뀌는지 확인하고, 자식 클래스에서는 private 직접 접근이 안 된다는 점을 먼저 적용하며, 포인터 변수이면 멤버 호출을 ->로 처리한다.

### 암기 포인트

같은 이름 대입은 멤버 초기화가 아닐 수 있고, private은 자식도 직접 접근하지 못하며, 포인터 호출은 ->이다.

NoteBase

**문 25. 다음은 프로젝트 P의 계획 단계에서 수립한 작업, 소요기간, 각 작업의 선행작업이다. 프로젝트 수행 중 개발자들이 독감에 걸려 작업 T5와 T7이 각각 7일, 5일이 지연되어 프로젝트 최소 완료시간이 40일로 되었다면 (가)에 들어갈 작업 소요기간(일)은? (단, T5와 T7을 제외한 다른 작업은 계획대로 수행되었다)**

| 작업 | 소요기간(일) | 선행작업 |
|---|---|---|
| T1 | 7 | – |
| T2 | 8 | T1 |
| T3 | 3 | T1 |
| T4 | 5 | T2, T3 |
| T5 | 6 | T3 |
| T6 | 10 | T5 |
| T7 | (가) | T4 |
| T8 | 5 | T6, T7 |

① 8

② 10

③ 13

④ 15

**정답 2번**

**출제 포인트**

이 문제는 선행작업 관계에서 지연 후의 주공정을 다시 계산하여 T7의 원래 소요기간을 역산하는 문제이다.

**조건 정리**

주어진 값: T5는 원래 6일인데 7일 지연되어 13일이 된다. T7은 원래 (가)일인데 5일 지연되어 (가)+5일이 된다.
구해야 할 값: T7의 원래 소요기간이다.
주의할 조건: T8은 T6와 T7이 모두 끝나야 시작하므로 두 경로 중 더 늦는 경로가 전체 완료시간을 결정한다.

NoteBase

### 사용할 공식

프로젝트 최소 완료시간은 말단 작업까지의 가능한 경로들 중 최장 경로 길이이다.
이 공식을 쓰는 이유: 이 문제는 작업망의 선행관계가 주어졌고, 지연된 뒤의 전체 완료시간 40일이 최장 경로로 결정되기 때문이다.

### 계산 과정

1. T1 종료 시점은 7일이다.
2. T2는 15일, T3는 10일에 끝난다. 따라서 T4는 15일에 시작해 20일에 끝난다.
3. T5는 지연으로 13일이 되어 10일에 시작해 23일에 끝나고, T6는 33일에 끝난다.
4. T7은 20일에 시작해 20+((가)+5)=25+(가)일에 끝난다.
5. T8은 T6와 T7이 모두 끝난 뒤 시작하므로 시작 시점은 (33,25+(가))이고, 완료 시점은 (33,25+(가))+5=40이다.
6. 따라서 (38,30+(가))=40이고, 38보다 커져야 40이 되므로 30+(가)=40이다. 따라서 (가)=10이다.

### 선지별 해설

① X: '(가)=8일'이면 지연 후 T7은 13일이므로 T7 종료는 33일, T6 종료도 33일이다. 그러면 T8은 33일에 시작해 38일에 끝난다. 틀린 표현은 '(가)=8일'이며, 올바르게는 '(가)=10일'이다.
② O: '(가)=10일'이면 지연 후 T7은 15일, T7 종료는 35일이다. T8은 $\max(33, 35) = 35$일에 시작해 40일에 끝나므로 조건과 일치한다.
③ X: '(가)=13일'이면 지연 후 T7은 18일, T7 종료는 38일이다. T8은 38일에 시작해 43일에 끝난다. 틀린 표현은 '(가)=13일'이며, 올바르게는 '(가)=10일'이다.
④ X: '(가)=15일'이면 지연 후 T7은 20일, T7 종료는 40일이다. T8은 40일에 시작해 45일에 끝난다. 틀린 표현은 '(가)=15일'이며, 올바르게는 '(가)=10일'이다.

### 오답 포인트

T5의 지연 7일과 T7의 지연 5일을 단순히 더하거나, T8이 T6와 T7 중 하나만 끝나면 시작된다고 착각하면 오답이 나온다. 이 문항의 함정은 마지막 작업 T8의 시작 시점이 두 선행작업의 완료시점 중 최댓값이라는 점이다.

### 다시 나오면 이렇게 판단

작업망 계산 문제는 각 작업의 가장 이른 완료시점을 먼저 적고, 병합 노드에서는 선행작업 완료시점의 최댓값을 취해 최장 경로를 다시 계산하면 된다. 지연이 생기면 지연된 작업의 완료시점부터 말단까지 다시 연결해 본다.

### 암기 포인트

병합 작업의 시작시점은 선행작업 완료시점의 합이 아니라 최댓값이며, 프로젝트 완료시간은 최장 경로가 결정한다.

NoteBase