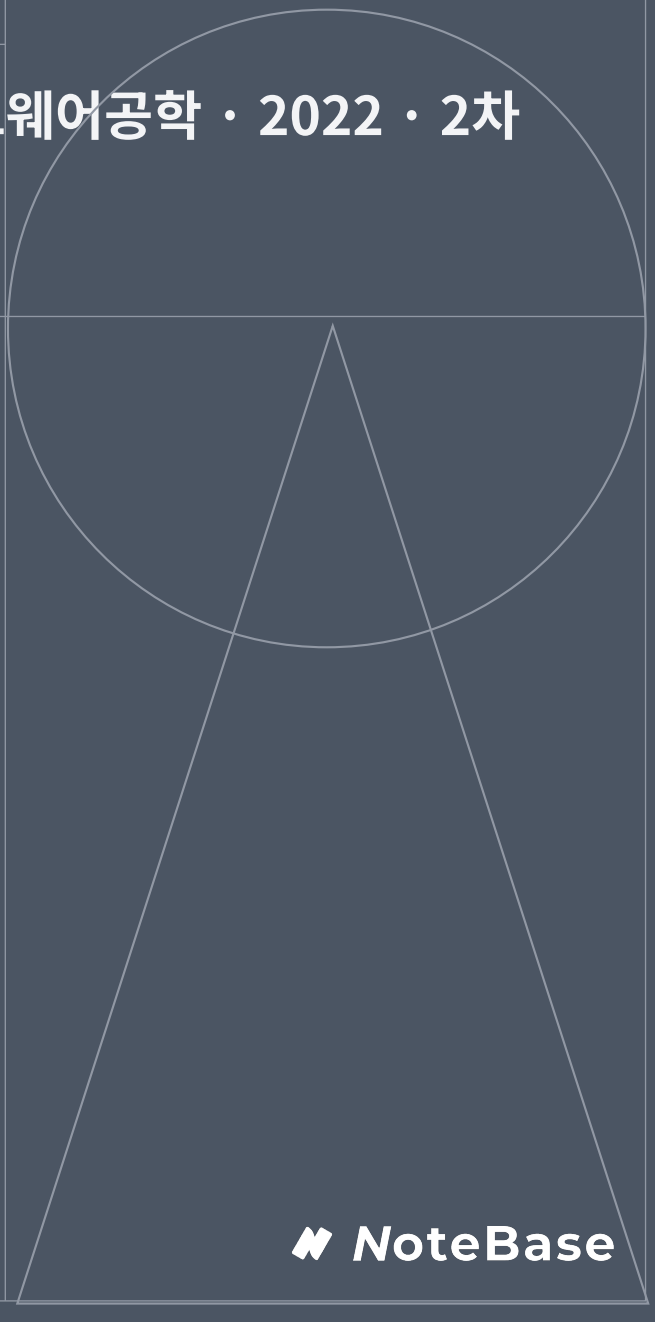
기출OX 해설집

공무원/국가직/7급 · 소프트웨어공학 · 2022 · 2차



NoteBase

기출OX 해설집

공무원/국가직/7급 · 소프트웨어공학 · 2022 · 2차

공무원/국가직/7급 · 소프트웨어공학 · 2022 · 2차

문 1. 소프트웨어 프로젝트에서 수행 작업에 대한 설명으로 옳지 않은 것은?

① 프로그래밍은 소프트웨어 공학의 모든 작업을 통합하는 작업이다.

② 설계 작업은 요구사항을 어떻게 구현할지 결정하는 과정으로 서브 시스템 분할, 상세 설계, 인터페이스 설계 및 자료 저장 방식 등의 작업으로 구성된다.

③ 모델링 작업은 시스템의 기능, 동작 및 구성을 추상화 형태로 표현해 가는 과정으로 사용 사례 모델링, 정 · 동적 모델링이 있다.

④ 테스트 작업은 개별 코드를 테스트하기 위한 단위 테스트, 이를 연결하여 제대로 작동하는지 확인하는 통합 테스트 및 빌드(build)한 시스템의 기능이 원하는 대로 작동하는지 확인하는 시스템 테스트 작업 등이 있다.

정답 3번

출제 포인트

이 문제는 소프트웨어 프로젝트의 수행 작업 중 설계, 모델링, 프로그래밍, 테스트의 역할과 특히 모델링의 분류 체계를 구분하는지를 묻는 문제이다.

핵심 개념

설계는 요구사항을 실제 시스템 구조와 인터페이스로 구체화하는 작업이다.
모델링은 시스템을 이해하고 설계하기 위해 기능, 구조, 동작을 추상화하여 표현하는 작업이다.
모델링의 대표 분류는 기능 모델링, 정적 모델링, 동적 모델링이다.
사용 사례 모델링은 기능 모델링의 대표 기법이고, 정적 모델링은 구조를, 동적 모델링은 행위와 상태 변화를 표현한다.
프로그래밍은 분석·설계 결과를 코드로 구현하여 실행 가능한 시스템으로 통합하는 작업이다.
테스트는 단위, 통합, 시스템 등 여러 수준에서 소프트웨어가 요구대로 동작하는지 검증하는 작업이다.

NoteBase

선지별 해설

① O
프로그래밍은 분석과 설계에서 정의된 내용을 실제 코드로 구현하여 하나의 실행 가능한 소프트웨어로 묶는 작업이므로 통합적 성격의 설명으로 볼 수 있다.

② O
설계는 요구사항을 어떤 구조와 인터페이스, 자료 저장 방식으로 실현할지 정하는 단계이므로 서브시스템 분할, 상세 설계, 인터페이스 설계 등의 설명이 맞다.

③ X
틀린 표현은 "사용 사례 모델링, 정·동적 모델링"이다.
모델링의 병렬 분류는 기능 모델링, 정적 모델링, 동적 모델링이다.
사용 사례 모델링은 기능 모델링의 대표적인 방법이지 정적·동적 모델링과 같은 수준의 분류명으로 병렬 나열하는 표현은 부정확하다.
올바르게는 "모델링 작업은 시스템의 기능, 구조, 동작을 추상화하여 표현하는 과정으로 기능 모델링, 정적 모델링, 동적 모델링이 있다."라고 정리한다.

④ O
단위 테스트는 개별 코드 단위를 검증하고, 통합 테스트는 연결된 모듈 간 상호작용을 확인하며, 시스템 테스트는 빌드된 전체 시스템이 요구 기능을 만족하는지 확인하므로 설명이 타당하다.

오답 포인트

③은 내용이 전부 틀린 것이 아니라 모델링의 분류 축을 미세하게 바꿔 놓은 선지이다. 수험생은 사용 사례 모델링이 익숙하다는 이유로 그대로 맞다고 보기 쉬우나, 시험에서는 기능 모델링-정적 모델링-동적 모델링의 분류 체계를 정확히 묻는다.

다시 나오면 이렇게 판단

소프트웨어공학에서 작업이나 모델의 종류를 묻는 선지는 먼저 분류 기준이 같은 수준인지 확인하면 된다. 기능 모델링의 예시와 모델링의 대분류를 섞어 쓰면 오답일 가능성이 높다.

암기 포인트

모델링은 기능-정적-동적의 3축이고, 사용 사례는 기능 모델링에 속한다.

NoteBase

문 2. 다음 설명에 해당하는 프로젝트 범위 관리 프로세스는?

◯ 주요 프로젝트 인도물을 더 작고 관리 가능한 구성 요소로 나누는 작업이다.
◯ WBS(Work Breakdown Structure)를 도구로 사용하고, WBS 체계에 따른 계층 구조와 최하위 항목인 작업 패키지가 결과로 산출된다.

① 범위 기획
② 범위 정의
③ 범위 검증
④ 범위 변경 통제

정답 2번

출제 포인트

이 문제는 프로젝트 범위 관리 프로세스 중 WBS를 이용해 인도물을 작업 패키지까지 세분화하는 범위 정의를 구별하는 문제이다.

핵심 개념

범위 기획은 범위를 어떻게 정의하고 검증하고 통제할지 관리 방식을 정하는 단계이다.
범위 정의는 주요 인도물을 더 작고 관리 가능한 요소로 분해하여 WBS와 작업 패키지를 만드는 단계이다.
범위 검증은 완성된 인도물이나 범위를 이해관계자가 공식적으로 수용하는 단계이다.
범위 변경 통제는 승인된 범위 기준선의 변경을 심사하고 반영하는 단계이다.

선지별 해설

① X: 범위 기획은 범위관리계획을 수립하는 단계이므로 인도물을 WBS로 세분화하는 작업과 다르다. 틀린 부분은 범위 기획으로 본 점이다. 올바르게는 범위를 작업 패키지까지 분해하는 활동은 범위 정의이다.
② O: 제시문은 주요 프로젝트 인도물을 더 작고 관리 가능한 구성요소로 나누고, WBS의 최하위 산출물인 작업 패키지를 만든다고 했으므로 범위 정의에 해당한다.
③ X: 범위 검증은 만들어진 결과물이나 범위를 고객 또는 발주자가 수용하는 절차이다. 틀린 부분은 세분화 작업을 검증 단계로 본 점이다. 올바르게는 검증은 분해가 아니라 인수와 확인이다.
④ X: 범위 변경 통제는 범위 변경 요청을 평가하고 기준선 변경 여부를 관리하는 절차이다. 틀린 부분은 WBS 작성과 작업 패키지 도출을 변경 통제로 본 점이다. 올바르게는 변경 통제는 승인된 범위의 수정 관리이다.

오답 포인트

범위 정의와 범위 기획을 혼동하기 쉽다. 기획은 관리 절차를 정하는 메타 단계이고, 정의는 인도물을 실제로 분해해 WBS를 만드는 실행 단계라는 차이를 놓치면 ①로 흔들린다.

NoteBase

다시 나오면 이렇게 판단

선지나 지문에 WBS, 분해, 계층 구조, 작업 패키지가 나오면 먼저 범위 정의를 떠올린다. 반대로 수용, 승인, 변경 요청, 기준선 관리가 나오면 각각 범위 검증이나 범위 변경 통제로 분리한다.

암기 포인트

WBS와 작업 패키지는 범위 정의, 수용은 범위 검증, 변경 요청 평가는 범위 변경 통제이다.

NoteBase

문 3. 다음 설명에 해당하는 것으로 옳은 것은?

◯ 프로젝트 진행의 중요한 상태를 정의한다.
◯ 프로젝트나 프로덕트가 특정 상태에 이르렀는지 나타낸다.
◯ 계속되는 개발 또는 유지보수 작업의 기준이 된다.
◯ 형상 항목에 대한 변경을 제어하는 메커니즘이다.

① VCS(Version Control System)
② Baseline
③ CMMI(Capability Maturity Model Integration)
④ CI(Continuous Integration)

정답 2번

출제 포인트

이 문제는 형상관리에서 Baseline의 의미와 VCS, CMMI, CI의 역할을 구별할 수 있는지를 묻는다.

핵심 개념

Baseline은 특정 시점의 산출물 상태를 공식 기준선으로 확정한 것으로, 이후 개발·유지보수와 변경 통제의 기준이 된다.
VCS는 파일과 형상 항목의 버전 이력을 저장·추적하는 시스템이다.
CMMI는 조직의 개발 프로세스 성숙도를 평가하고 개선하는 모델이다.
CI는 코드를 자주 통합하고 빌드·테스트를 자동화하는 개발 방식이다.

선지별 해설

1번 X: VCS는 버전 변경 이력과 병합 상태를 관리하는 시스템이다. 틀린 부분은 설명 상자의 내용을 VCS의 정의로 본 점이다. 올바르게는 특정 상태를 공식 기준으로 삼아 이후 변경을 통제하는 개념은 Baseline이다.
2번 O: Baseline은 프로젝트나 프로덕트가 특정 상태에 도달했음을 표시하는 기준선이다. 한 번 기준선으로 설정되면 이후 개발과 유지보수의 출발점이 되고, 변경은 통제 절차를 거쳐 관리된다.
3번 X: CMMI는 개발 조직의 프로세스 성숙도와 개선 수준을 다루는 모델이다. 틀린 부분은 상태 기준선의 정의를 CMMI에 대응시킨 점이다. 올바르게는 CMMI는 프로세스 개선 모델이고, 문제의 설명은 Baseline이다.
4번 X: CI는 개발 결과물을 자주 통합하여 빌드와 테스트를 지속적으로 수행하는 방식이다. 틀린 부분은 변경 통제 기준선을 CI로 본 점이다. 올바르게는 지속적 통합은 통합 자동화 개념이고, 형상 항목의 공식 기준 상태는 Baseline이다.

NoteBase

오답 포인트

VCS와 Baseline이 모두 형상관리 영역에 있으므로 혼동하기 쉽다. VCS는 변경 이력을 관리하는 도구·체계이고, Baseline은 그 변경을 비교하고 승인할 때 기준이 되는 확정 상태라는 점이 차이이다.

다시 나오면 이렇게 판단

선지나 설명에서 기준선, 특정 시점의 확정 상태, 이후 변경 통제의 기준이라는 표현이 나오면 Baseline을 먼저 떠올리면 된다. 버전 기록이면 VCS, 프로세스 성숙도면 CMMI, 자동 통합이면 CI로 구분된다.

암기 포인트

Baseline은 기준선, VCS는 버전관리, CMMI는 성숙도 모델, CI는 지속적 통합이다.

NoteBase

문 4. 소프트웨어의 품질 속성에 대한 설명으로 옳지 않은 것은?

① 신뢰성 - 요구된 기능을 실행하여 정확하고 일관성 있는 결과를 생성하는 속성

② 효율성 - 요구된 기능을 최소의 시간과 자원을 사용하여 결과를 생성하는 속성

③ 모듈성 - 컴포넌트의 통합이나 조정을 쉽게 만드는 속성

④ 상호운용성 - 소프트웨어가 여러 운영 환경 및 플랫폼에서 실행될 수 있도록 변형이 가능한 속성

정답 4번

출제 포인트

이 문제는 소프트웨어 품질 속성의 정의를 구분할 수 있는지를 묻는 문제이다.

핵심 개념

신뢰성은 요구된 기능을 주어진 조건에서 정확하고 일관되게 수행하는 성질이다.
효율성은 성능 결과를 내는 데 필요한 시간과 자원을 얼마나 적게 쓰는지를 보는 성질이다.
모듈성은 시스템을 독립된 컴포넌트로 나누어 조합과 변경의 영향을 줄이기 쉽게 하는 성질이다.
상호운용성은 다른 시스템이나 소프트웨어와 정보를 교환하고 그 정보를 함께 활용할 수 있는 성질이다.
여러 운영 환경과 플랫폼에 맞게 옮기거나 바꾸기 쉬운 성질은 상호운용성이 아니라 이식성이다.

선지별 해설

1번 O: 신뢰성은 요구된 기능이 정확하고 일관된 결과를 내는 성질이므로 설명이 맞다.
2번 O: 효율성은 최소한의 시간과 자원으로 요구된 결과를 내는 성질이므로 설명이 맞다.
3번 O: 모듈성은 컴포넌트를 나누어 통합·조정·변경을 쉽게 하고 다른 부분에 미치는 영향을 줄이는 성질이므로 취지가 맞다.
4번 X: 틀린 표현은 "여러 운영 환경 및 플랫폼에서 실행될 수 있도록 변형이 가능한"이다. 이 내용은 상호운용성이 아니라 이식성의 설명이다. 올바르게는 "상호운용성은 다른 시스템이나 소프트웨어와 정보를 교환하고 그 교환된 정보를 함께 활용할 수 있는 속성"이라고 고쳐야 한다.

오답 포인트

상호운용성과 이식성을 혼동하면 틀리기 쉽다. 다른 시스템과의 정보 교환을 말하면 상호운용성이고, 운영 환경이나 플랫폼을 바꾸어도 실행되게 만드는 성질을 말하면 이식성이다.

다시 나오면 이렇게 판단

선지에 "다른 시스템과 정보 교환"이 나오면 상호운용성으로 본다. "여러 운영 환경·플랫폼에서 실행"이나 "변형 가능"이 나오면 이식성으로 판단한다.

NoteBase

암기 포인트

상호운용성은 연결과 교환, 이식성은 환경 변경과 이동이다.

NoteBase

문 5. 유스케이스 명세서를 구성하는 항목에 대한 설명으로 옳지 않은 것은?

① 개요는 유스케이스가 제공하는 기능을 나타낸 것으로, 그 기능을 사용하는 사용자 액터까지 표현해야 한다.

② 이벤트 흐름(시나리오)은 유스케이스 기능에 대한 액터와 시스템 간의 대화로, 문자나 순차 다이어그램을 사용해 표현한다.

③ 기능 요구사항은 성능, 신뢰성, 보안성, 가용성 등 여러 요구 품질 요소를 나타낸다.

④ 후행 조건은 유스케이스의 수행 완료 후에 만족되어야 하는 조건으로 상태나 수치의 바뀐 모습을 명확하고 구체적으로 기술해야 한다.

정답 3번

출제 포인트

이 문제는 유스케이스 명세서의 구성 항목 중 기능 요구사항과 비기능 요구사항의 차이, 그리고 시나리오·후행 조건의 의미를 구별할 수 있는지를 묻는다.

핵심 개념

기능 요구사항은 시스템이 무엇을 해야 하는지를 나타내는 요구사항이다.
비기능 요구사항은 성능, 신뢰성, 보안성, 가용성처럼 시스템이 어느 수준으로 동작해야 하는지를 나타내는 품질 요구사항이다.
이벤트 흐름은 액터와 시스템이 기능을 수행하는 과정의 상호작용 순서를 기술한 것이다.
후행 조건은 유스케이스 수행이 끝난 뒤 시스템 상태가 어떻게 바뀌어 있어야 하는지를 적는 것이다.

선지별 해설

1번: O. 개요는 해당 유스케이스가 제공하는 기능을 간단히 설명하는 항목이고, 그 기능과 관련된 주요 액터를 함께 드러내는 것이 자연스럽다. 어떤 기능을 누구가 사용하는지까지 보여야 유스케이스의 범위가 분명해진다.
2번: O. 이벤트 흐름, 즉 시나리오는 액터와 시스템 사이에 어떤 상호작용이 어떤 순서로 일어나는지를 기술하는 부분이다. 이를 문장으로 쓰거나 순차 다이어그램처럼 흐름이 드러나는 방식으로 표현할 수 있다.
3번: X. 틀린 부분은 "기능 요구사항은 성능, 신뢰성, 보안성, 가용성 등 여러 요구 품질 요소를 나타낸다"는 표현이다. 성능, 신뢰성, 보안성, 가용성은 기능 요구사항이 아니라 비기능 요구사항 또는 품질 요구사항에 해당한다. 올바르게는 "기능 요구사항은 시스템이 제공해야 할 기능을 나타내고, 성능·신뢰성·보안성·가용성 등은 비기능 요구사항을 나타낸다"가 된다.
4번: O. 후행 조건은 유스케이스가 정상적으로 끝난 뒤 만족되어야 하는 결과 상태를 적는 항목이다. 실행 후 어떤 데이터나 상태가 어떻게 바뀌었는지를 구체적으로 적어야 검증 가능한 명세가 된다.

NoteBase

오답 포인트

3번은 요구사항 분류를 바꿔치기한 선지이다. 수험생이 성능, 보안성 같은 품질 속성을 시스템 기능의 일부로 막연히 묶어 기억하면 기능 요구사항으로 착각하기 쉽다.

다시 나오면 이렇게 판단

선지에 성능, 신뢰성, 보안성, 가용성 같은 단어가 나오면 먼저 비기능 요구사항인지부터 본다. 반대로 시스템이 무엇을 해 주는지, 액터와 어떤 상호작용을 하는지를 설명하면 기능 요구사항이나 시나리오 항목으로 판단한다.

암기 포인트

기능 요구사항은 무엇을 하는가, 비기능 요구사항은 얼마나 잘해야 하는가를 묻는다고 기억하면 된다.

NoteBase

문 6. ISO/IEC 9126 모델에 대한 품질 특성 요소로 잘못 짝지은 것은?

① 신뢰성 요소 - 성숙성, 결함허용성, 회복성

② 유지보수성 요소 - 분석성, 변경성, 안정성

③ 기능성 요소 - 적용성, 설치용이성, 부합성

④ 사용성 요소 - 이해성, 학습용이성, 운영성

정답 3번

출제 포인트

이 문제는 ISO/IEC 9126의 품질 특성과 각 하위 특성의 대응 관계를 정확히 구분하는지를 묻는 문제이다.

핵심 개념

기능성은 소프트웨어가 요구 기능을 정확하고 안전하게 제공하는 성질로 적합성, 정확성, 상호운용성, 보안성 등이 핵심이다.
신뢰성은 장애 상황에서도 정상 수준의 서비스를 유지하고 복구하는 성질로 성숙성, 결함허용성, 회복성이 핵심이다.
유지보수성은 수정과 분석이 쉬운 정도를 뜻하며 분석성, 변경성, 안정성, 시험성이 핵심이다.
사용성은 사용자가 이해하고 배우고 운용하기 쉬운 정도를 뜻하며 이해성, 학습용이성, 운용성이 핵심이다.
이식성은 다른 환경으로 옮겨 적용하기 쉬운 정도를 뜻하며 적용성, 설치용이성, 공존성, 대체성이 핵심이다.

선지별 해설

① O: 신뢰성의 대표 하위 특성은 성숙성, 결함허용성, 회복성이다. 장애 발생 가능성을 줄이고 장애 후 복구 가능성을 보는 묶음이므로 맞다.
② O: 유지보수성의 하위 특성에는 분석성, 변경성, 안정성이 포함된다. 수정 원인 파악과 변경의 용이성, 변경 후 영향의 안정성을 묻는 구성이라 맞다.
③ X: 틀린 부분은 기능성 요소를 적용성, 설치용이성, 부합성으로 제시한 점이다. 적용성과 설치용이성은 기능성이 아니라 이식성의 하위 특성이다. 기능성은 적합성, 정확성, 상호운용성, 보안성, 기능성 준수성으로 이해하는 것이 옳다.
④ O: 사용성의 하위 특성으로 이해성, 학습용이성, 운용성을 묶는 것은 맞다. 사용자가 시스템을 이해하고 배우고 실제로 조작하기 쉬운지를 보는 항목들이다.

오답 포인트

③은 기능성과 이식성의 하위 특성을 서로 바꿔 놓은 함정이다. 설치용이성이라는 표현이 기능 제공과 관련 있어 보이지만 실제로는 다른 환경에 배치하고 옮기는 이식성 범주에 속한다.

NoteBase

다시 나오면 이렇게 판단

품질 특성 문제는 먼저 상위 범주가 무엇을 평가하는지 한 줄로 정의한 뒤 하위 특성을 연결하면 판단이 빨라진다. 기능성은 기능 제공, 신뢰성은 장애 대응, 사용성은 사용 편의, 유지보수성은 수정 용이, 이식성은 환경 이동으로 구분하면 된다.

암기 포인트

적용성·설치용이성은 기능성이 아니라 이식성이다.

NoteBase

문 7. 유스케이스 다이어그램에 대한 설명으로 옳지 않은 것은?

① 시스템 외부의 액터들은 서로 관계를 형성하지 않는다.

② 시스템과 상호작용하는 외부 시스템은 액터로 표현한다.

③ 유스케이스들의 공통 호출 서비스는 <<include>> 관계로 유스케이스를 구성한다.

④ 유스케이스들의 선택적 서비스는 <<extend>> 관계로 유스케이스를 구성한다.

정답 1번

출제 포인트

이 문제는 유스케이스 다이어그램에서 액터 간 관계와 <<include>>, <<extend>>의 의미를 구별할 수 있는지를 묻는다.

핵심 개념

액터는 시스템과 상호작용하는 외부 역할이며, 액터 사이에도 일반화 관계를 둘 수 있다.
<<include>>는 여러 유스케이스가 공통으로 수행하는 필수 기능을 분리해 재사용하는 관계이다.
<<extend>>는 기본 유스케이스에 조건부로 덧붙는 선택적 기능을 나타내는 관계이다.

선지별 해설

1번: X. 틀린 표현은 '서로 관계를 형성하지 않는다'이다. 액터는 서로 완전히 독립적인 존재가 아니며 상위 액터와 하위 액터처럼 일반화 관계를 형성할 수 있다.
2번: O. 사람뿐 아니라 시스템과 데이터를 주고받는 외부 시스템도 유스케이스 다이어그램에서는 액터로 표현한다.
3번: O. 여러 유스케이스가 공통으로 호출하는 기능은 기본 유스케이스가 반드시 포함하는 공통 기능이므로 <<include>> 관계로 분리한다.
4번: O. 특정 조건에서만 추가되는 선택적 기능은 기본 유스케이스를 확장하는 성격이므로 <<extend>> 관계로 표현한다.

오답 포인트

액터를 사람 그림으로만 외우면 2번을 놓치기 쉽고, 액터 사이에는 관계가 없다고 단정하면 1번에 걸린다. 또한 공통 기능과 선택 기능을 모두 포함 관계로 뭉뚱그리면 3번과 4번을 구분하지 못한다.

다시 나오면 이렇게 판단

유스케이스 문제에서는 먼저 액터가 사람만이 아니라 외부 역할 전체인지 확인하고, 기능이 반드시 공통 수행되면 <<include>>, 조건부로 덧붙으면 <<extend>>로 판단한다.

NoteBase

암기 포인트

액터도 일반화 가능, 공통 필수는 <<include>>, 선택 확장은 <<extend>>이다.

NoteBase

문 8. MVC(Model-View-Controller) 패턴에 대한 설명으로 옳지 않은 것은?

① 뷰는 사용자에게 처리 로직의 결과를 보여준다.

② 컨트롤러는 사용자의 요청을 받아 모델에 보낸다.

③ 모델은 데이터를 관리하며 변경이 일어나면 뷰에게 통지한다.

④ 사용자 인터페이스가 변경되면 모델에 영향을 준다.

정답 4번

출제 포인트

이 문제는 MVC 패턴에서 Model, View, Controller의 역할 분리와 의존 방향을 정확히 구분하는지를 묻는다.

핵심 개념

Model은 데이터와 업무 로직을 관리하는 구성요소이다.
View는 Model의 상태나 처리 결과를 사용자에게 보여 주는 화면 계층이다.
Controller는 사용자 입력을 받아 해석하고 필요한 처리를 Model에 전달하는 중재 계층이다.
MVC의 핵심은 사용자 인터페이스와 업무 로직을 분리하여 View 변경이 Model에 직접 영향을 주지 않게 하는 구조이다.

선지별 해설

1번 O: View는 처리 결과와 데이터를 사용자에게 표시하는 역할을 한다. 사용자에게 결과를 보여 준다는 설명이므로 옳다.
2번 O: Controller는 사용자의 요청이나 입력을 받아 적절한 Model 기능을 호출한다. 요청을 받아 Model에 보낸다는 설명이므로 옳다.
3번 O: Model은 데이터를 관리하고 상태 변화가 생기면 그 변경 사실을 View에 반영할 수 있도록 알린다. 데이터 관리와 변경 통지라는 Model의 역할을 설명한 것이므로 옳다.
4번 X: 틀린 부분은 '사용자 인터페이스가 변경되면 모델에 영향을 준다'는 표현이다. 올바르게는 사용자 인터페이스인 View가 변경되어도 Model은 직접 영향을 받지 않도록 분리하는 것이 MVC의 목적이다.

오답 포인트

View와 Model이 서로 관련된다는 이유만으로 UI 변경이 곧 Model 변경이라고 오해하기 쉽다. 이 문항은 '관련 있음'과 '직접 영향 받음'을 구분하지 못하면 4번을 놓치게 만든다.

다시 나오면 이렇게 판단

MVC 문제는 먼저 입력은 Controller, 데이터와 로직은 Model, 출력은 View로 역할을 배치한 뒤, UI 변경이 업무 로직까지 흔드는 설명인지 확인하면 된다.

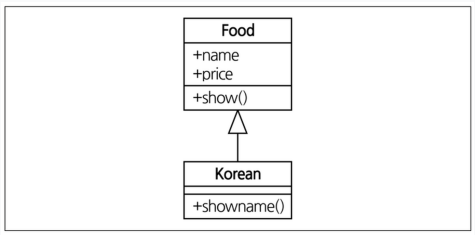
NoteBase

암기 포인트

MVC는 입력은 Controller, 처리와 데이터는 Model, 화면은 View이며, View가 바뀌어도 Model은 독립적이다.

NoteBase

문 9. 다음 클래스 다이어그램을 파이썬 코드로 표현한 것이다. (가) ~ (라)에 들어갈 내용으로 옳은 것은?




```
class (가):
  def __init__(self, name, price):
    self.name = name
    self.price = price

  def (나)(self):
    print("{}'s price = {}".format(self.name,
          self.price))

class (다)((가)):
  def __init__(self, name, price):
    Food.__init__(self, name, price)

  def (라)(self):
    print("{}".format(self.name))
```

① (가): Food, (나): show, (다): Korean, (라): showname

② (가): Korean, (나): showname, (다): Food, (라): show

③ (가): Food, (나): showname, (다): Korean, (라): show

④ (가): Korean, (나): show, (다): Food, (라): showname

정답 1번

출제 포인트

이 문제는 클래스 다이어그램의 상속 방향과 각 클래스에 선언된 속성·메서드를 파이썬 클래스 선언으로 정확히 옮길 수 있는지를 묻는 문제이다.

NoteBase

선지별 해설

① O
(가)는 상위 클래스 이름이므로 Food가 들어가야 한다. Food 클래스에는 name, price 속성과 show() 메서드가 있으므로 (나)는 show가 맞다. 빈 삼각형 화살표가 Food 쪽을 가리키므로 Korean이 Food를 상속하므로 (다)는 Korean이 맞다. Korean 클래스에만 있는 메서드는 showname()이므로 (라)는 showname이 맞다.

② X
(가)를 Korean으로 두면 상위 클래스가 Korean이 되어 다이어그램의 상속 방향과 반대가 된다. 또한 (나)를 showname으로 두면 Food 클래스에 showname()이 선언된 꼴이 되어 다이어그램과 맞지 않는다. 틀린 표현은 "(가)=Korean, (나)=showname, (다)=Food, (라)=show"이고, 올바르게는 "(가)=Food, (나)=show, (다)=Korean, (라)=showname"이다.

③ X
(가)=Food와 (다)=Korean은 맞지만, Food 클래스의 메서드를 (나)=showname으로 둔 부분이 틀리다. Food에는 show()가 있고 Korean에만 showname()이 있다. 틀린 표현은 "(나)=showname, (라)=show"이고, 올바르게는 "(나)=show, (라)=showname"이다.

④ X
(나)=show는 맞지만 (가)를 Korean으로 두는 순간 Korean이 부모 클래스가 되어 상속 관계가 뒤집힌다. 이어서 (다)=Food도 함께 틀리게 된다. 틀린 표현은 "(가)=Korean, (다)=Food"이고, 올바르게는 "(가)=Food, (다)=Korean"이다.

오답 포인트

빈 삼각형 화살표의 방향을 반대로 읽으면 부모와 자식 클래스가 뒤바뀐다. 또한 메서드 이름만 보고 show와 showname을 바꾸기 쉬운데, show는 Food 박스에 있고 showname은 Korean 박스에 있다는 위치 차이가 핵심 함정이다.

다시 나오면 이렇게 판단

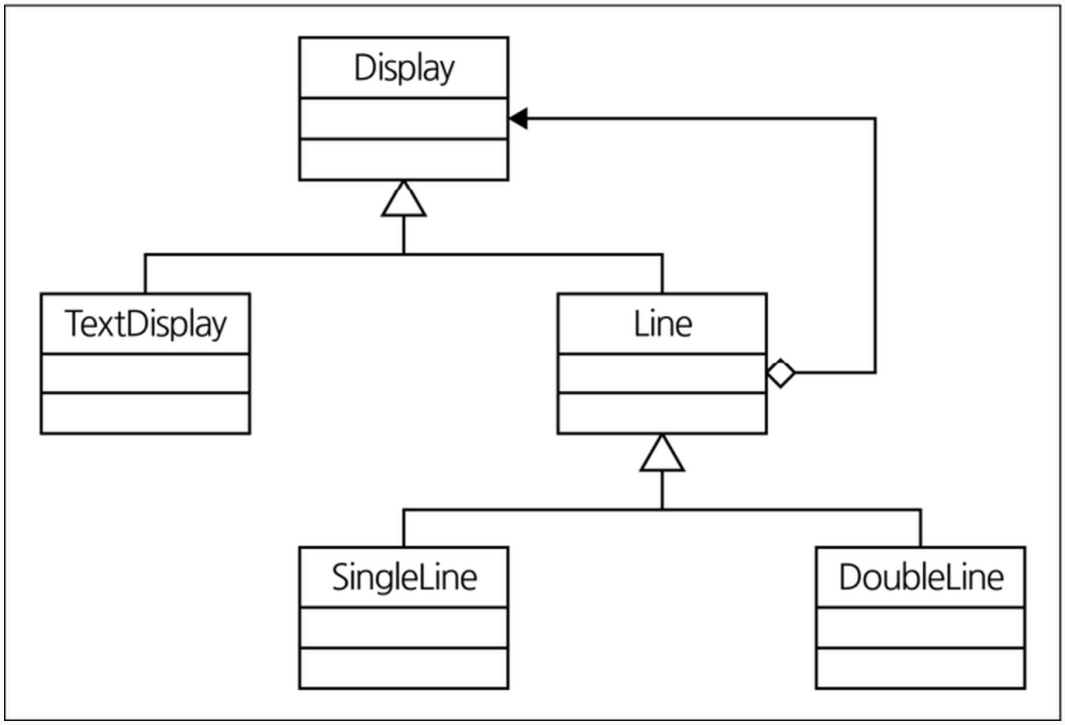
상속 문제는 먼저 빈 삼각형 화살표가 가리키는 쪽을 부모 클래스로 확정한 뒤, 각 클래스 박스에 적힌 속성과 메서드를 같은 이름으로 코드의 클래스 본문에 배치하면 된다.

암기 포인트

UML 빈 삼각형 화살표의 끝은 부모 클래스이고, 박스 안 메서드 이름은 그 클래스의 메서드이다.

NoteBase

문 10. 다음 클래스 다이어그램과 관련이 있는 디자인 패턴은?



① 데코레이터(Decorator) 패턴

② 메멘토(Memento) 패턴

③ 어댑터(Adapter) 패턴

④ 빌더(Builder) 패턴

정답 1번

출제 포인트

이 문제는 클래스 다이어그램에서 공통 추상 클래스인 Display를 상속하면서 내부에 다시 Display를 참조해 기능을 덧붙이는 구조를 보고 데코레이터 패턴을 식별할 수 있는지를 묻는다.

NoteBase

선지별 해설

1번 O: Display가 공통 Component 역할을 하고, TextDisplay가 ConcreteComponent 역할을 하며, Line이 Display를 상속하면서 내부에 Display를 보유하므로 Decorator의 전형적 구조이다. SingleLine과 DoubleLine은 구체 데코레이터로서 원래 객체에 선 장식 기능을 추가한다.
2번 X: 메멘토 패턴은 객체의 상태를 저장했다가 복원하기 위한 패턴으로 Originator, Memento, Caretaker의 관계가 중심이다. 이 그림처럼 같은 추상 타입을 상속하면서 내부에 동일 타입 객체를 감싸는 구조가 아니다. 틀린 표현은 메멘토로 보는 점이며, 올바르게는 상태 저장용 객체 관계가 보여야 메멘토 패턴이다.
3번 X: 어댑터 패턴은 호환되지 않는 기존 클래스를 Target 인터페이스에 맞게 변환하는 구조가 핵심이다. 이 그림에는 기존 Adaptee를 변환하는 어댑터가 아니라 동일한 Display 계열 객체를 감싸 기능을 확장하는 구조가 나타난다. 틀린 표현은 어댑터로 보는 점이며, 올바르게는 Target-Adapter-Adaptee의 변환 관계가 보여야 어댑터 패턴이다.
4번 X: 빌더 패턴은 복잡한 객체 생성 과정을 Builder, ConcreteBuilder, Director로 분리하는 생성 패턴이다. 이 그림은 객체 생성 절차 분리가 아니라 기존 객체에 기능을 덧붙이는 상속+위임 구조이다. 틀린 표현은 빌더로 보는 점이며, 올바르게는 생성 단계 분리 구조가 보여야 빌더 패턴이다.

오답 포인트

상속만 보고 패턴을 고르면 어댑터나 빌더처럼 넓게 오인하기 쉽다. 이 문항의 함정은 Line이 Display를 상속하는 점보다 내부에 다시 Display를 보유한다는 점에 있으며, 같은 타입을 감싸 기능을 누적 추가하면 데코레이터 패턴이다.

다시 나오면 이렇게 판단

먼저 공통 추상 타입이 있는지 보고, 다음으로 어떤 클래스가 그 추상 타입을 상속하면서 내부 필드로 같은 추상 타입 객체를 참조하는지를 확인한다. 이 두 조건이 함께 보이면 기존 객체를 감싸 기능을 확장하는 데코레이터 패턴으로 판단할 수 있다.

암기 포인트

데코레이터 패턴은 같은 추상 타입을 상속하고 같은 추상 타입 객체를 내부에 들고 기능을 덧붙이는 구조이다.

NoteBase

문 11. 화이트박스 테스트만을 모두 고르면?

ㄱ. 제어 흐름 테스팅
ㄴ. 경로 테스팅
ㄷ. 상태기반 테스팅
ㄹ. 원인 결과 그래프

① ㄱ, ㄴ
② ㄱ, ㄷ
③ ㄴ, ㄷ
④ ㄴ, ㄹ

정답 1번

출제 포인트

이 문제는 화이트박스 테스트 기법과 블랙박스 테스트 기법을 구분하는 문제이다.

핵심 개념

화이트박스 테스트는 소스코드의 내부 구조와 제어 흐름을 기준으로 테스트 케이스를 설계하는 기법이다.
블랙박스 테스트는 내부 구현을 보지 않고 요구사항, 입력 조건, 출력 결과, 상태 전이를 기준으로 테스트하는 기법이다.
제어 흐름 테스팅과 경로 테스팅은 코드의 분기와 실행 경로를 확인하므로 화이트박스에 속한다.
상태기반 테스팅과 원인 결과 그래프는 명세와 상태 변화, 입력-출력 관계를 다루므로 블랙박스에 속한다.

선지별 해설

① O: ㄱ 제어 흐름 테스팅과 ㄴ 경로 테스팅은 모두 프로그램 내부 로직의 분기와 실행 경로를 확인하는 대표적 화이트박스 테스트이다.
② X: ㄱ은 화이트박스이지만 ㄷ 상태기반 테스팅은 시스템의 상태와 이벤트 전이를 기준으로 검증하는 블랙박스 테스트이다. 틀린 표현은 "상태기반 테스팅을 화이트박스에 포함한 부분"이며, 올바르게는 "ㄱ만 화이트박스이고 ㄷ은 블랙박스이다"이다.
③ X: ㄴ 경로 테스팅은 화이트박스이지만 ㄷ 상태기반 테스팅은 블랙박스이다. 틀린 표현은 "경로 테스팅과 상태기반 테스팅을 같은 화이트박스 범주로 묶은 부분"이며, 올바르게는 "경로 테스팅은 화이트박스, 상태기반 테스팅은 블랙박스이다"이다.
④ X: ㄴ 경로 테스팅은 화이트박스이지만 ㄹ 원인 결과 그래프는 입력 조건과 결과의 논리 관계를 분석해 테스트 케이스를 만드는 블랙박스 기법이다. 틀린 표현은 "원인 결과 그래프를 화이트박스에 포함한 부분"이며, 올바르게는 "ㄴ만 화이트박스이고 ㄹ은 블랙박스이다"이다.

NoteBase

오답 포인트

상태기반 테스팅과 원인 결과 그래프는 용어만 보면 구조 분석처럼 느껴져 화이트박스로 착각하기 쉽다. 그러나 화이트박스의 판단 기준은 코드 내부 구조를 직접 보느냐이고, 상태와 입력-출력 관계를 기준으로 하면 블랙박스이다.

다시 나오면 이렇게 판단

테스트 기법이 코드의 분기, 문장, 경로, 데이터 흐름처럼 내부 구현을 직접 따라가면 화이트박스이다. 반대로 요구사항, 상태 전이, 동치 분할, 경계값, 원인 결과 관계처럼 외부 명세를 기준으로 하면 블랙박스이다.

암기 포인트

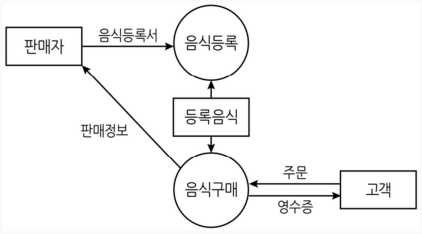
화이트박스는 코드 구조, 블랙박스는 명세와 상태이다.

NoteBase

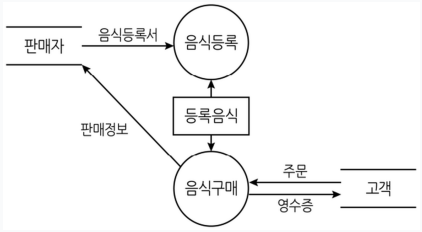
문 12. 다음 Yourdon/DeMarco 표기법에 따라서 자료 흐름도로 표현하였을 때 옳은 것은?

판매자가 음식을 등록하여 등록음식 데이터베이스에 저장한 후에, 고객이 음식을 주문하면, 고객에게 영수증이 발급되고, 판매자에게 판매정보가 전달된다.

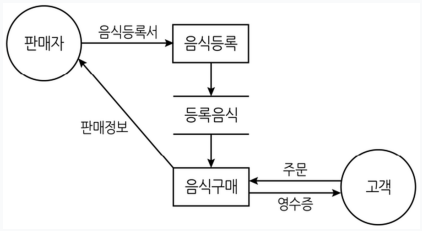
①



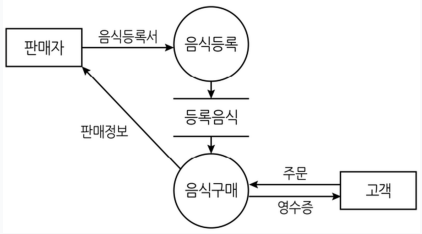
②



③



④



정답 4번

출제 포인트

이 문제는 Yourdon/DeMarco 자료 흐름도에서 외부 개체, 처리, 자료저장소, 자료 흐름의 표기법을 정확히 구분하는지 묻는 문제이다.

NoteBase

선지별 해설

① X
- 판매자와 고객을 사각형으로, 음식등록과 음식구매를 원으로 표시한 점은 맞지만 등록음식을 사각형으로 그려 자료저장소가 아니라 외부 개체처럼 표현하였다.
- 틀린 표현은 등록음식의 도형이다. 올바르게는 등록음식 데이터베이스는 평행선 형태의 자료저장소로 표시되어야 한다.

② X
- 음식등록과 음식구매를 원으로 둔 점은 맞지만 판매자와 고객을 평행선으로 그려 외부 개체가 아니라 자료저장소처럼 표현하였다.
- 틀린 표현은 판매자와 고객의 도형이다. 올바르게는 외부 개체인 판매자와 고객은 사각형으로 표시되어야 한다.

③ X
- 등록음식을 평행선으로 표시한 점은 맞지만 판매자와 고객을 원으로, 음식구매를 사각형으로 그려 외부 개체와 처리의 표기법을 서로 바꾸었다.
- 틀린 표현은 판매자, 고객, 음식구매의 도형이다. 올바르게는 판매자와 고객은 사각형, 음식구매는 원으로 표시되어야 한다.

④ O
- 판매자와 고객을 사각형인 외부 개체로, 음식등록과 음식구매를 원인 처리로, 등록음식을 평행선인 자료저장소로 올바르게 표현하였다.
- 음식목록, 주문, 영수증, 판매정보의 흐름도 각 주체 사이에 자연스럽게 연결되어 있어 Yourdon/DeMarco 표기법에 맞는 자료 흐름도이다.

오답 포인트

이 문항의 함정은 업무 내용보다 도형의 의미를 정확히 외웠는지를 묻는 데 있다. 특히 등록음식 데이터베이스를 사각형으로 그리거나, 판매자·고객 같은 외부 개체를 자료저장소 표기와 혼동하면 쉽게 오답이 된다.

다시 나오면 이렇게 판단

자료 흐름도 문제는 먼저 사각형이 외부 개체, 원이 처리, 평행선이 자료저장소라는 기본 도형 의미를 확인한 뒤 각 데이터 흐름 화살표가 그 의미에 맞게 연결되는지를 보면 빠르게 판단할 수 있다.

암기 포인트

Yourdon/DeMarco는 사각형은 외부 개체, 원은 처리, 평행선은 자료저장소로 구분한다.

NoteBase

문 13. 객체지향 프로그램의 품질 측정 방법으로 옳지 않은 것은?

① 클래스 당 가중치를 갖는 메소드
② 사이클로메틱 복잡성
③ 상속 트리의 깊이
④ 메소드의 응집 결핍

정답 2번

출제 포인트

이 문제는 객체지향 전용 품질 척도와 일반적인 프로그램 복잡도 척도를 구별할 수 있는지를 묻는다.

핵심 개념

클래스 당 가중치를 갖는 메소드(WMC)는 클래스가 가진 메소드 수와 복잡도를 함께 보는 객체지향 척도이다.
상속 트리의 깊이(DIT)는 상속 계층이 얼마나 깊은지를 보는 객체지향 척도이다.
메소드의 응집 결핍(LCOM)은 한 클래스 내부 메소드들이 얼마나 느슨하게 관련되는지를 보는 객체지향 척도이다.
사이클로매틱 복잡성은 제어 흐름의 독립 경로 수를 기준으로 모듈 복잡도를 재는 일반 소프트웨어 복잡도 척도이다.

선지별 해설

1번 O: 클래스 당 가중치를 갖는 메소드는 WMC를 뜻한다. 클래스에 메소드가 많거나 각 메소드 복잡도가 높을수록 유지보수 부담이 커지는지를 보는 대표적 객체지향 품질 척도이다.
2번 X: 사이클로매틱 복잡성은 객체지향 품질 측정법이라기보다 제어 흐름 그래프의 분기 수를 바탕으로 독립 경로 수를 재는 일반 모듈 복잡도 척도이다. 틀린 표현은 객체지향 프로그램의 품질 측정 방법이라는 점이며, 올바르게는 전통적인 프로그램 복잡도 측정 방법이다.
3번 O: 상속 트리의 깊이는 DIT를 뜻한다. 상속 구조가 깊어질수록 재사용 가능성은 커질 수 있지만 이해와 테스트 부담도 함께 커지므로 객체지향 설계 품질을 보는 척도로 사용된다.
4번 O: 메소드의 응집 결핍은 LCOM을 뜻한다. 같은 클래스의 메소드들이 동일한 속성을 얼마나 함께 사용하는지를 봄으로써 클래스 응집도를 평가하는 객체지향 품질 척도이다.

오답 포인트

사이클로매틱 복잡성도 품질과 관련된 수치이므로 객체지향 척도로 착각하기 쉽다. 이 문항은 C-K 계열의 객체지향 메트릭과 일반 모듈 복잡도 메트릭을 섞어 놓은 점이 함정이다.

NoteBase

다시 나오면 이렇게 판단

선지가 클래스, 상속, 응집, 결합도처럼 객체 구조 자체를 재면 객체지향 메트릭으로 본다. 반대로 분기 수, 경로 수처럼 제어 흐름 중심이면 일반 프로그램 복잡도 척도로 판단한다.

암기 포인트

WMC, DIT, LCOM은 객체지향 메트릭이고 사이클로매틱 복잡성은 제어 흐름 복잡도 메트릭이다.

NoteBase

문 14. 소프트웨어의 요구 사항 수집 및 기술에 대한 설명으로 옳지 않은 것은?

① 사용자의 요구 사항을 구체적으로 파악하기 위해 소프트웨어의 일부 기능을 빠르게 구현한 프로토타입을 이용한다.

② 소프트웨어 사용 관련자들이 브레인스토밍을 통해 다양한 요구 사항을 도출한다.

③ 현재 시스템의 문제를 파악하기 위해 관련 직원으로부터 시스템의 문제점을 파악하고 자료를 수집한다.

④ 수집한 요구 사항은 프로그래밍 언어를 이용해서 구체적으로 기술한다.

정답 4번

출제 포인트

이 문제는 요구 사항 수집 기법과 요구 사항 기술 방법을 구분할 수 있는지를 묻는 문제이다.

핵심 개념

요구 사항 수집은 사용자와 이해관계자로부터 필요한 기능과 제약을 끌어내는 활동이다. 키워드는 인터뷰, 브레인스토밍, 프로토타이핑, 현행 업무 조사이다.
요구 사항 기술은 수집된 요구를 문서로 명확하게 표현하는 활동이다. 키워드는 자연어 명세, 구조적 명세, 모델, UML이다.
프로그래밍 언어는 구현 수단이지 요구 사항 자체를 기술하는 기본 수단은 아니다.

NoteBase

선지별 해설

① O
프로토타입은 소프트웨어의 일부 기능을 빠르게 구현해 사용자의 반응을 확인하면서 요구 사항을 구체화하는 기법이다. 요구 사항이 불명확할 때 효과적인 수집 기법이다.

② O
브레인스토밍은 여러 이해관계자가 자유롭게 의견을 제시하면서 다양한 요구 사항 후보를 도출하는 대표적 요구 사항 수집 기법이다.

③ O
현재 시스템의 문제를 파악하기 위해 관련 직원 인터뷰, 현장 관찰, 자료 수집을 수행하는 것은 현행 시스템 분석과 사실 조사에 해당한다. 요구 사항 수집의 기초 자료를 얻는 올바른 방법이다.

④ X
틀린 부분은 "프로그래밍 언어를 이용해서"이다. 수집한 요구 사항은 구현 코드가 아니라 자연어, 구조적 명세, 다이어그램과 같은 요구 사항 명세 방법으로 기술한다. 프로그래밍 언어는 설계 이후 구현 단계의 표현 수단에 가깝다.

오답 포인트

요구 사항을 '구체적으로 적는다'는 표현 때문에 곧바로 코드 수준 기술로 연결하기 쉽다. 그러나 요구 사항 분석 단계의 구체화와 구현 단계의 코딩은 다른 활동이므로, ④는 기술 대상과 표현 수단을 뒤바꾼 선지이다.

다시 나오면 이렇게 판단

선지가 인터뷰, 브레인스토밍, 프로토타입, 현행 조사처럼 요구를 끌어내는 활동을 말하면 수집 단계로 본다. 반대로 자연어 명세, 모델, 다이어그램처럼 수집된 요구를 문서화하는 표현을 말하면 기술 단계로 본다. 코드 작성이 나오면 구현 단계인지 먼저 의심하게 된다.

암기 포인트

요구 사항은 수집 후 명세하고, 코드는 그다음 구현 단계에서 작성된다.

NoteBase

문 15. 소프트웨어 생명주기에 대한 설명으로 옳지 않은 것은?

① 소프트웨어 개발방법론의 바탕이 된다.

② 일반적으로 정의단계, 개발단계, 유지보수 단계로 구분된다.

③ 프로젝트 비용 산정과 개발 계획을 수립할 수 있는 기본 골격이 된다.

④ 개발단계에서 시간과 비용이 가장 많이 든다.

정답 4번

출제 포인트

이 문제는 소프트웨어 생명주기의 전체 단계와 각 단계의 역할, 특히 유지보수 단계의 비중을 구별할 수 있는지를 묻는다.

핵심 개념

소프트웨어 생명주기는 소프트웨어의 계획, 개발, 운영, 폐기에 이르는 전 과정을 단계로 나눈 기본 틀이다.
개발단계는 요구사항을 바탕으로 설계, 구현, 시험을 수행해 제품을 만드는 단계이다.
유지보수단계는 운영 중 발생하는 오류 수정, 기능 개선, 환경 변화 대응을 수행하는 단계이며 누적 시간과 비용 비중이 가장 큰 경우가 많다.

선지별 해설

1번: O. 소프트웨어 생명주기는 개발 활동을 어떤 순서와 범위로 관리할지 정하는 기본 골격이므로 다양한 소프트웨어 개발방법론의 바탕이 된다.
2번: O. 소프트웨어 생명주기는 크게 정의단계, 개발단계, 유지보수 단계로 구분하는 설명이 대표적이다.
3번: O. 생명주기를 단계별로 나누면 각 단계에 필요한 인력, 기간, 산출물을 예측할 수 있으므로 프로젝트 비용 산정과 개발 계획 수립의 기준이 된다.
4번: X. 틀린 부분은 '개발단계에서 시간과 비용이 가장 많이 든다'는 표현이다. 일반적으로 가장 큰 누적 시간과 비용이 드는 단계는 유지보수 단계이다.

오답 포인트

개발단계에서 설계, 구현, 시험이 집중되므로 가장 많은 자원이 든다고 단정하기 쉽다. 그러나 생명주기 전체 관점에서는 운영 이후의 수정, 개선, 적응 작업이 장기간 반복되어 유지보수 비용이 더 크게 누적된다.

다시 나오면 이렇게 판단

생명주기 문제는 먼저 전체 단계를 정의, 개발, 유지보수로 나눈 뒤 각 단계의 대표 기능을 연결해 판단한다.
비용과 기간의 최대 비중을 묻는 표현이 나오면 유지보수 단계를 우선 떠올리면 된다.

NoteBase

암기 포인트

생명주기는 전체 틀이고, 비용의 무게중심은 개발단계가 아니라 유지보수 단계에 있다.

NoteBase

문 16. 평가 점수에 따른 성적을 아래 표와 같이 부여하는 소프트웨어를 제작하여 경곗값 분석 기법으로 테스트할 때, 테스트 케이스는 몇 개인가? (단, 평가 점수는 정수이다)

평가 점수	등급
90점 이상부터 100점 이하	A
80점 이상부터 90점 미만	B
70점 이상부터 80점 미만	C
60점 이상부터 70점 미만	D
0점 이상부터 60점 미만	F

① 10

② 15

③ 18

④ 20

정답 3번

출제 포인트

이 문제는 정수 입력에 대해 경곗값 분석을 적용할 때 경계점마다 직전값, 경계값, 직후값을 잡아 전체 테스트 케이스 수를 계산하는 문제이다.

조건 정리

주어진 값:
- 점수 구간은 0점 이상 100점 이하의 정수이다.
- 등급 경계는 0, 60, 70, 80, 90, 100이다.

구해야 할 값:
- 경곗값 분석으로 필요한 테스트 케이스 수이다.

주의할 조건:
- 점수가 정수이므로 경계의 바로 아래와 바로 위는 각각 1 차이 나는 정수로 잡는다.

NoteBase

사용할 공식

경곗값 분석에서는 각 경계점 b마다 $b-1$, b, $b+1$을 시험한다.
이 공식을 쓰는 이유:
이 문제의 오류 가능성은 등급이 바뀌는 경계에서 가장 크므로 각 경계의 바로 전, 경계 자체, 바로 다음 값을 확인해야 하기 때문이다.

계산 과정

1. 경계점을 정리하면 0, 60, 70, 80, 90, 100의 6개이다.
2. 각 경계점마다 시험값은 $b-1$, b, $b+1$의 3개이다.
3. 따라서 전체 테스트 케이스 수는 $6 \times 3 = 18$개이다.
4. 실제 값으로 쓰면 -1, 0, 1, 59, 60, 61, 69, 70, 71, 79, 80, 81, 89, 90, 91, 99, 100, 101의 18개이다.

선지별 해설

① X
10개는 일부 경계만 세거나 각 등급 구간을 대표값으로만 잡은 값이다. 올바르게는 경계점 6개 각각에 대해 3개씩 시험하므로 18개이다.

② X
15개는 경계점 수는 보았지만 일부 경계의 직전값 또는 직후값을 빠뜨린 계산이다. 올바르게는 0, 60, 70, 80, 90, 100의 모든 경계에서 $b-1$, b, $b+1$을 확인해야 하므로 18개이다.

③ O
경계점 6개에 대해 각 3개 값씩 시험하면 $6 \times 3 = 18$개이므로 맞다.

④ X
20개는 경계 개수를 중복 계산한 값이다. 경곗값 분석은 등급 수를 기계적으로 곱하는 것이 아니라 실제 경계점 개수 6개를 기준으로 세므로 올바르게는 18개이다.

오답 포인트

구간의 개수 5개만 보고 계산하거나, 경계점은 찾았지만 양끝 경계 0과 100을 빼먹으면 오답이 된다. 이 문제의 함정은 등급 개수와 경계점 개수를 혼동하는 데 있다.

다시 나오면 이렇게 판단

경곗값 분석 문제는 먼저 입력 범위의 실제 경계점을 모두 적고, 각 경계마다 직전값, 경계값, 직후값을 붙여서 개수를 센다. 구간 수가 아니라 경계점 수를 기준으로 판단한다.

NoteBase

암기 포인트

경곗값 분석은 경계마다 3개씩 본다. 이 문제는 경계 6개이므로 18개이다.

NoteBase

문 17. 다음 디자인 패턴들을 GoF(Gang of Fours) 패턴 분류에 따라 구분하였을 때, 생성 패턴, 구조 패턴, 행위 패턴의 개수를 순서대로 바르게 나열한 것은?

◯ Abstract Factory ◯ Adapter ◯ Bridge
◯ Builder ◯ Composite ◯ Decorator
◯ Iterator ◯ Mediator ◯ Observer
◯ Prototype ◯ Proxy ◯ Singleton
◯ Strategy ◯ Visitor

① 생성 패턴: 3, 구조 패턴: 5, 행위 패턴: 6

② 생성 패턴: 4, 구조 패턴: 5, 행위 패턴: 5

③ 생성 패턴: 5, 구조 패턴: 3, 행위 패턴: 6

④ 생성 패턴: 6, 구조 패턴: 4, 행위 패턴: 4

정답 2번

출제 포인트

이 문제는 GoF 디자인 패턴을 생성 패턴, 구조 패턴, 행위 패턴으로 정확히 분류한 뒤 각 개수를 세는 문제이다.

핵심 개념

생성 패턴은 객체를 어떻게 생성할지를 다루는 패턴이다. 구조 패턴은 클래스와 객체를 어떻게 조합해 구조를 만드는지를 다루는 패턴이다. 행위 패턴은 객체 사이의 책임 분담과 상호작용 방식을 다루는 패턴이다.
제시된 목록을 분류하면 생성 패턴은 Abstract Factory, Builder, Prototype, Singleton의 4개이다. 구조 패턴은 Adapter, Bridge, Composite, Decorator, Proxy의 5개이다. 행위 패턴은 Iterator, Mediator, Observer, Strategy, Visitor의 5개이다.

선지별 해설

① X: 틀린 부분은 "행위 패턴 6"이다. 제시된 행위 패턴은 Iterator, Mediator, Observer, Strategy, Visitor로 5개뿐이다. 올바르게는 4, 5, 5이다.
② O: 생성 4개, 구조 5개, 행위 5개로 분류가 정확하다. 생성 패턴에는 Factory Method가 없으므로 생성 개수는 5가 아니라 4이다.
③ X: 틀린 부분은 "생성 패턴 5"와 "구조 패턴 3"이다. 생성 패턴은 4개이고 구조 패턴은 5개이다. 올바르게는 4, 5, 5이다.
④ X: 틀린 부분은 "생성 패턴 6"과 "구조 패턴 4"이다. 생성 패턴은 4개, 구조 패턴은 5개이므로 수치가 모두 어긋난다. 올바르게는 4, 5, 5이다.

NoteBase

오답 포인트

Factory Method와 Singleton의 위치를 혼동하거나, Strategy와 Bridge처럼 이름이 익숙한 패턴을 기능이 아니라 느낌으로 분류하면 개수가 쉽게 틀어진다. 이 문제는 패턴 이름을 외운 정도가 아니라 분류 축을 정확히 알고 있는지를 확인하는 함정형 문항이다.

다시 나오면 이렇게 판단

패턴 문제는 먼저 각 패턴을 하나씩 생성, 구조, 행위 중 어디에 속하는지 표시한 뒤 마지막에 개수를 센다. 이름이 비슷해 보여도 생성은 객체 생성, 구조는 조합 구조, 행위는 상호작용이라는 기준으로만 판단하면 된다.

암기 포인트

생성은 만들기, 구조는 엮기, 행위는 상호작용이며 이 문제의 개수는 4, 5, 5이다.

NoteBase

문 18. SPICE(Software Process Improvement and Capability dEtermination) 모델은 소프트웨어 개발 프로세스를 구성하는 각 프로세스 영역의 달성 수준(또는 능력)을 측정하는 레벨을 제시한다. 레벨과 그 레벨의 측정 관점을 짝 지은 것으로 옳지 않은 것은?

① 수행단계 - 해당 프로세스의 목적은 달성하지만 계획되거나 추적되지 않는다.

② 관리단계 - 표준으로 정의된 프로세스를 사용하여 계획되고 관리한다.

③ 예측단계 - 표준 프로세스 능력을 정량적으로 측정, 이해, 관리할 수 있어 품질의 정량적 관리가 가능하다.

④ 최적화 단계 - 프로세스가 지속적으로 개선되고 최적화된다.

정답 2번

출제 포인트

이 문제는 SPICE 프로세스 능력수준 가운데 관리단계와 확립단계의 구분 기준을 묻는 문제이다.

핵심 개념

수행단계는 프로세스 목적을 달성하지만 수행 방식이 아직 체계적으로 관리되지는 않는 수준이다.
관리단계는 수행된 프로세스를 계획하고 모니터링하며 작업산출물을 관리하는 수준이다.
확립단계는 조직의 표준 프로세스를 기반으로 수행 프로세스를 정의하여 사용하는 수준이다.
예측단계는 프로세스 성과를 정량적으로 측정하고 통제하는 수준이다.
최적화단계는 정량적 이해를 바탕으로 프로세스를 지속적으로 개선하는 수준이다.

선지별 해설

① O. 수행단계는 프로세스가 목적은 달성하지만 계획·추적·표준화가 충분히 자리 잡지 않은 상태이므로 설명이 맞다.
② X. 틀린 부분은 "표준으로 정의된 프로세스를 사용하여"이다. 관리단계의 핵심은 계획과 모니터링, 작업산출물 관리이며, 조직 표준 프로세스를 사용한다는 설명은 확립단계에 해당한다.
③ O. 예측단계는 프로세스 능력을 정량적으로 측정하고 이해하여 품질과 성과를 통계적으로 관리할 수 있는 수준이므로 설명이 맞다.
④ O. 최적화단계는 정량적 관리 결과를 바탕으로 프로세스를 지속적으로 개선하는 수준이므로 설명이 맞다.

오답 포인트

관리단계와 확립단계가 가장 자주 뒤섞인다. 관리단계는 계획·추적·산출물 관리가 핵심이고, 확립단계는 조직 표준 프로세스 사용이 핵심이라는 점을 구분하면 ②의 오류가 바로 보인다.

NoteBase

다시 나오면 이렇게 판단

SPICE 수준 문제는 먼저 선택지의 핵심 동사를 본다. 계획·모니터링이면 관리단계, 표준 프로세스 정의·사용이면 확립단계, 정량 측정이면 예측단계, 지속적 개선이면 최적화단계로 판단된다.

암기 포인트

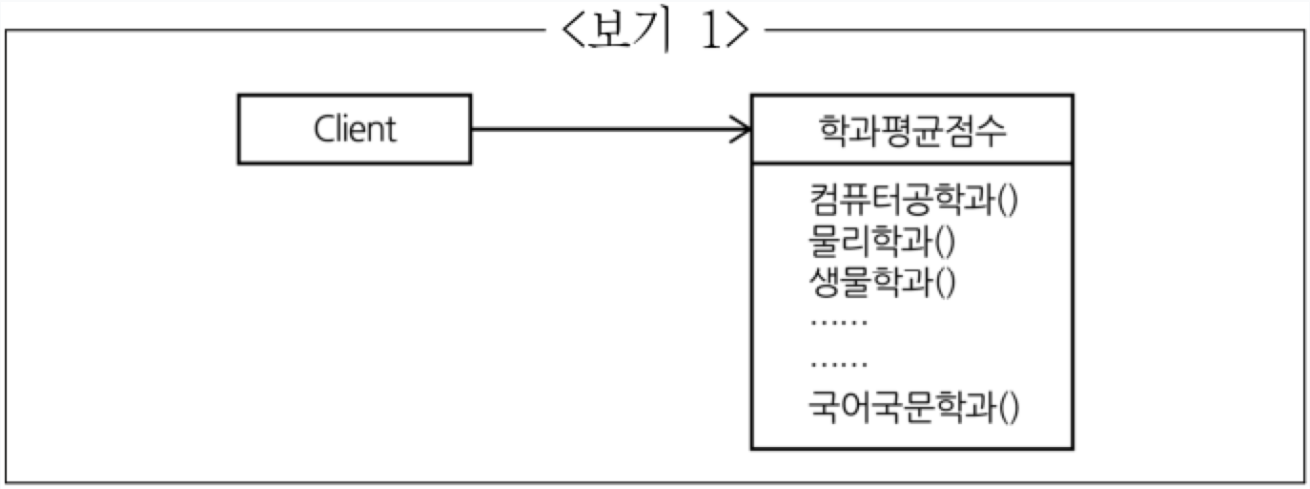
관리는 계획·통제, 확립은 표준화, 예측은 정량화, 최적화는 지속개선이다.

NoteBase

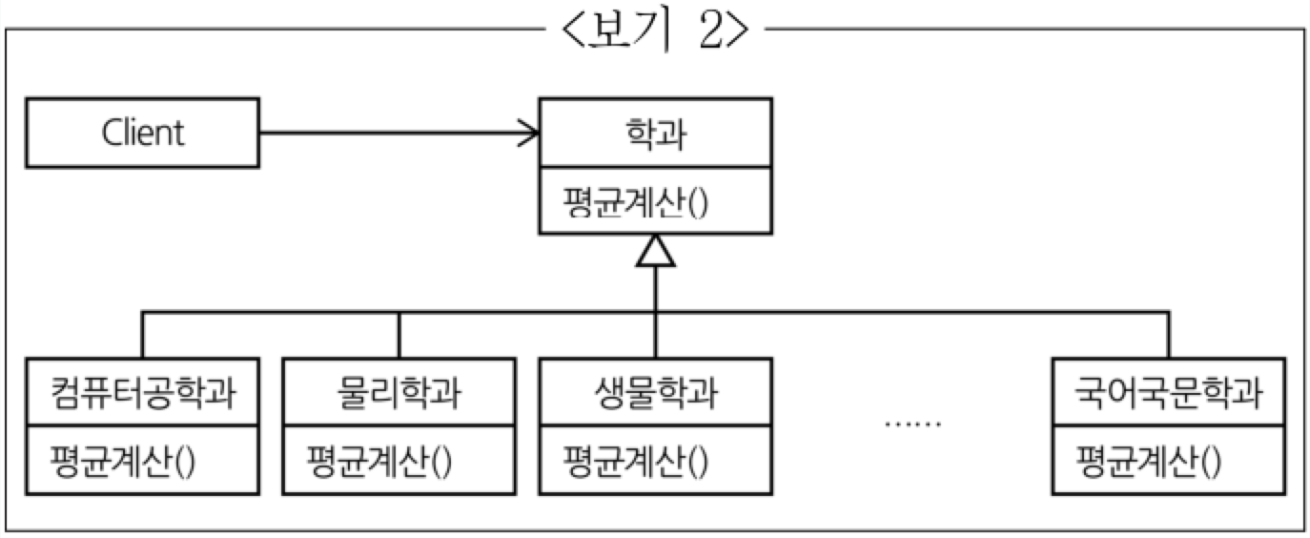
문 19. 다음 클래스 다이어그램은 객체지향 설계에서 클래스를 설계할 때의 원칙 가운데 하나를 적용한 사례이다. 설계 원칙으로 옳은 것은?

'학과평균점수' 클래스의 역할은 한 대학의 학과별 기말고사 평균 점수를 계산하기 위한 것이다. <보기 1>은 클래스 설계의 해당 원칙을 적용하기 전의 다이어그램이고, <보기 2>는 해당 원칙을 적용한 이후의 다이어그램이다.

<보기 1>



<보기 2>



① 개방 폐쇄의 원칙

② 의존 관계 역전의 원칙

③ 인터페이스 분리의 원칙

④ 단일 책임 원칙

정답 1번

NoteBase

출제 포인트

이 문제는 도식에서 학과별 평균 계산 기능을 상속 구조로 분리하여 새 학과가 추가되어도 기존 상위 구조를 고치지 않고 확장으로 대응하는 점을 보고 개방 폐쇄의 원칙을 판단하는 문제이다.

선지별 해설

① O
- <보기 1>에서는 학과평균점수 클래스 안에 컴퓨터공학과, 물리학과, 생물학과 같은 학과별 기능이 직접 들어 있다.
- <보기 2>에서는 상위 개념인 학과에 평균계산()을 두고 각 학과가 이를 구현하는 구조로 바뀌어 새 학과가 생겨도 기존 상위 구조를 수정하지 않고 클래스를 추가하여 확장할 수 있다.
- 이는 수정에는 닫히고 확장에는 열려 있어야 한다는 개방 폐쇄의 원칙에 맞다.

② X
- 의존 관계 역전의 원칙은 상위 모듈과 하위 모듈이 모두 구체 구현이 아니라 추상화에 의존하게 만드는 데 초점이 있다.
- 이 도식에도 추상화 요소는 보이지만, 문항이 강조하는 변화의 핵심은 새로운 학과 추가 시 기존 클래스를 계속 고치지 않도록 구조를 바꾼 점이다.
- 틀린 표현은 '의존 관계 역전의 원칙'이다. 올바르게는 '개방 폐쇄의 원칙'이다.

③ X
- 인터페이스 분리의 원칙은 클라이언트가 자신이 쓰지 않는 메서드에 의존하지 않도록 작은 인터페이스로 나누는 데 초점이 있다.
- 이 그림은 하나의 거대한 인터페이스를 여러 개로 쪼개는 상황이 아니라, 학과별 구현을 하위 클래스로 확장 가능하게 만든 상황이다.
- 틀린 표현은 '인터페이스 분리의 원칙'이다. 올바르게는 '개방 폐쇄의 원칙'이다.

④ X
- 단일 책임 원칙은 클래스가 하나의 변경 이유만 가지도록 책임을 분리하는 데 초점이 있다.
- 이 그림에서 직접 드러나는 핵심은 책임 수의 분리보다 새 학과 추가에 따른 변경을 상속 확장으로 흡수한 점이다.
- 틀린 표현은 '단일 책임 원칙'이다. 올바르게는 '개방 폐쇄의 원칙'이다.

오답 포인트

②와 혼동하기 쉽다. <보기 2>에 추상 클래스 학과가 등장하므로 의존 관계 역전처럼 보이지만, 정답을 가르는 기준은 추상화의 존재 자체가 아니라 새로운 학과가 생길 때 기존 코드를 수정하는지, 클래스를 추가하여 확장하는지이다.

NoteBase

다시 나오면 이렇게 판단

기존 클래스 안에 유형별 분기나 개별 기능이 계속 늘어나던 구조를 상위 추상화와 하위 확장 구조로 바꾸어 새 항목 추가를 클래스 추가로 처리하면 개방 폐쇄의 원칙으로 판단한다. 추상화가 보이더라도 초점이 의존 방향인지, 확장 방식인지부터 구분하면 된다.

암기 포인트

새 기능 추가를 기존 코드 수정이 아니라 새 클래스 확장으로 처리하면 개방 폐쇄의 원칙이다.

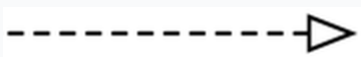
NoteBase

문 20. 다음 Java 코드에서 생략된 부분을 고려하지 않고 명시된 부분만으로 UML 클래스 다이어그램을 작성하였을 때, 다이어그램에 나타나지 않는 관계는?

```
class Person { ... }
class Work { ... }
interface PublicService {
     public void doWork(Work work);
}
class Official extends Person implements Publ
icService {
     public void sayHello() { ... }
     public void doWork(Work work) { ... }
     ...
}
```

① 

② 

③ 

④ 

정답 2번

출제 포인트

이 문제는 Java 코드의 상속, 인터페이스 구현, 매개변수 사용을 UML 관계로 옮겼을 때 일반화, 실체화, 의존 관계는 나타나지만 집합 관계는 나타나지 않는다는 점을 판단하는 문제이다.

NoteBase

선지별 해설

① X
- 점선 화살표는 의존 관계를 뜻한다. `doWork(Work work)`처럼 `Work`가 메서드 매개변수로 사용되므로 `PublicService`와 `Official`은 `Work`에 의존한다.
- 틀린 판단은 이 관계가 나타나지 않는다고 본 점이다. 올바르게는 코드에 의존 관계가 나타난다.

② O
- 빈 마름모가 있는 실선은 집합 관계를 뜻한다. 집합은 객체가 다른 객체를 속성이나 필드로 보유하는 구조에서 드러난다.
- 제시된 코드에는 `Official`이나 다른 클래스가 `Work`를 멤버로 포함하는 부분이 없으므로 집합 관계는 나타나지 않는다.

③ X
- 점선과 빈 삼각형은 실체화 관계를 뜻한다. `Official extends Person implements PublicService`에서 `implements PublicService`가 바로 인터페이스 실체화이다.
- 틀린 판단은 이 관계가 나타나지 않는다고 본 점이다. 올바르게는 `Official`과 `PublicService` 사이에 실체화 관계가 나타난다.

④ X
- 실선과 빈 삼각형은 일반화 관계를 뜻한다. `Official extends Person`이므로 `Official`은 `Person`을 상속한다.
- 틀린 판단은 이 관계가 나타나지 않는다고 본 점이다. 올바르게는 `Official`과 `Person` 사이에 일반화 관계가 나타난다.

오답 포인트

매개변수로 한 번 사용하는 관계와 멤버로 계속 보유하는 관계를 혼동하면 ①과 ②를 바꾸기 쉽다. UML에서 메서드 인자 사용은 의존 관계이고, 집합 관계는 필드나 속성 보유가 있을 때 나타난다.

다시 나오면 이렇게 판단

코드에서 `extends`는 일반화, `implements`는 실체화로 먼저 찾고, 메서드 매개변수에만 등장하면 의존으로 본다. 객체를 필드로 가지고 있는지까지 확인한 뒤에 집합이나 합성을 판단한다.

암기 포인트

`extends`는 일반화, `implements`는 실체화, 매개변수 사용은 의존, 필드 보유가 있어야 집합이다.

NoteBase

문 21. 화이트박스 테스트 커버리지에 대한 설명으로 옳은 것은?

① 문장 커버리지는 입력 데이터가 미리 정의한 유형에 적합한지를 검증하는 방법이다.

② 동일한 프로그램에 대해 분기 커버리지를 달성하면 조건 커버리지를 달성한다.

③ 동일한 프로그램에 대해 조건 커버리지를 달성하면 분기 커버리지를 달성한다.

④ 동일한 프로그램에 대해 수정조건분기 커버리지(Modified Condition Decision Coverage, MCDC)를 달성하면 조건 커버리지를 달성한다.

정답 4번

출제 포인트

이 문제는 문장 커버리지, 분기 커버리지, 조건 커버리지, 수정조건분기 커버리지의 정의와 포함 관계를 구별할 수 있는지를 묻는다.

핵심 개념

문장 커버리지는 프로그램의 각 실행 문장이 한 번 이상 수행되는지를 보는 기준이다.
분기 커버리지는 각 결정문의 결과가 참과 거짓을 모두 한 번 이상 갖는지를 보는 기준이다.
조건 커버리지는 결정문 안의 각 개별 조건이 참과 거짓을 모두 한 번 이상 갖는지를 보는 기준이다.
수정조건분기 커버리지는 각 개별 조건이 전체 결정 결과에 독립적으로 영향을 주는지를 확인하는 기준으로, 각 조건의 참과 거짓이 모두 나타나야 한다.

선지별 해설

1번: X. 문장 커버리지는 입력 데이터가 미리 정의한 유형에 적합한지를 검증하는 방법이 아니다. 틀린 표현은 "입력 데이터가 미리 정의한 유형에 적합한지"이다. 문장 커버리지는 각 실행 문장이 실제로 수행되었는지를 확인하는 구조 기반 테스트 기준이다.
2번: X. 분기 커버리지를 달성해도 조건 커버리지를 반드시 달성하는 것은 아니다. 틀린 표현은 "분기 커버리지를 달성하면 조건 커버리지를 달성한다"이다. 복합 조건문에서는 결정 결과의 참과 거짓만 모두 나오고 각 개별 조건의 참과 거짓은 모두 나오지 않을 수 있다.
3번: X. 조건 커버리지를 달성해도 분기 커버리지를 반드시 달성하는 것은 아니다. 틀린 표현은 "조건 커버리지를 달성하면 분기 커버리지를 달성한다"이다. 각 개별 조건이 참과 거짓을 모두 가져도 전체 결정 결과가 한쪽으로만 나와 분기 결과를 모두 만들지 못할 수 있다.
4번: O. 수정조건분기 커버리지는 각 조건이 독립적으로 결정 결과에 영향을 주는지를 보여야 하므로 각 개별 조건의 참과 거짓이 모두 포함된다. 따라서 조건 커버리지를 달성한다는 설명이 옳다.

NoteBase

오답 포인트

분기 커버리지와 조건 커버리지를 서로 포함 관계로 단순화하면 틀리기 쉽다. 이 둘은 복합 조건문에서 서로 독립적일 수 있고, MCDC가 그보다 더 강한 기준이라는 점을 놓치면 2번과 3번을 혼동하게 된다.

다시 나오면 이렇게 판단

커버리지 문제는 먼저 무엇을 세는 기준인지 구분하면 된다. 문장은 실행 문장, 분기는 결정 결과, 조건은 개별 불리언 조건, MCDC는 개별 조건의 독립적 영향 여부를 본다는 순서로 대입하면 정답을 판단할 수 있다.

암기 포인트

문장=실행문, 분기=결정 결과, 조건=개별 조건, MCDC=개별 조건의 독립적 영향이다.

NoteBase

문 22. 소프트웨어 개발 프로젝트 수행을 위한 설명으로 옳은 것은?

① 소프트웨어 기능의 정확성을 검증하기 위해 기능 점수 방법을 적용한다.

② 소프트웨어 개발 요구 사항의 변경에 효과적으로 대응하기 위해 폭포수 모형을 적용한다.

③ 소프트웨어 개발 프로젝트의 비용을 예측하기 위해 COCOMO 방법을 적용한다.

④ 소프트웨어 개발 프로젝트 일정을 계획하기 위해 CMMI 방법을 적용한다.

정답 3번

출제 포인트

이 문제는 기능 점수, 폭포수 모형, COCOMO, CMMI의 쓰임을 구별할 수 있는지를 묻는다.

핵심 개념

기능 점수는 사용자에게 제공되는 기능을 기준으로 소프트웨어 규모를 측정하는 방법이다.
폭포수 모형은 요구분석, 설계, 구현, 시험을 순차적으로 진행하는 전통적 개발 모형이다.
COCOMO는 소프트웨어 규모를 바탕으로 노력과 비용을 추정하는 비용 산정 모형이다.
CMMI는 조직의 소프트웨어 프로세스 성숙도를 평가하고 개선하는 모델이다.

선지별 해설

1번: X. 틀린 부분은 "기능의 정확성을 검증하기 위해 기능 점수 방법"이다. 기능 점수 방법은 테스트 기법이 아니라 기능 규모 산정 기법이므로 정확성 검증에 쓰이지 않는다. 올바르게는 "소프트웨어 규모를 측정하거나 개발 비용을 추정하기 위해 기능 점수 방법을 적용한다"이다.
2번: X. 틀린 부분은 "요구 사항의 변경에 효과적으로 대응하기 위해 폭포수 모형"이다. 폭포수 모형은 단계가 고정되어 있어 요구 변경 대응이 어렵다. 올바르게는 "요구 사항 변경에 효과적으로 대응하려면 반복적 개발이나 애자일 접근이 더 적합하다"이다.
3번: O. COCOMO는 소프트웨어 규모를 기준으로 개발 노력과 비용을 예측하는 대표적 비용 산정 모형이므로 프로젝트 비용 예측 설명으로 옳다.
4번: X. 틀린 부분은 "일정을 계획하기 위해 CMMI 방법"이다. CMMI는 일정 계획 도구가 아니라 프로세스 성숙도 개선 모델이다. 올바르게는 "프로젝트 일정을 계획할 때는 WBS, PERT, CPM 같은 일정 관리 기법을 활용한다"이다.

오답 포인트

규모 측정, 비용 산정, 개발 모형, 프로세스 성숙도 모델을 모두 "프로젝트 관리 기법"으로 뭉뚱그리면 틀리기 쉽다. 특히 기능 점수와 COCOMO는 모두 산정과 관련되어 함께 헷갈리기 쉽고, 폭포수와 CMMI는 개발 절차와 품질 개선이라는 점에서 역할이 섞이기 쉽다.

NoteBase

다시 나오면 이렇게 판단

선지에 나온 용어가 무엇을 측정하거나 관리하는 도구인지를 먼저 본다. 규모를 재는지, 비용을 추정하는지, 개발 절차를 설명하는지, 조직 프로세스를 개선하는지로 나누면 정답을 빠르게 가를 수 있다.

암기 포인트

기능 점수는 규모, COCOMO는 비용, 폭포수는 순차 개발, CMMI는 프로세스 성숙도이다.

NoteBase

문 23. 다음 입력 인자가 가질 수 있는 값들이 <표 1>과 같이 주어져 있을 때, 페어와이즈(pairwise) 테스트를 수행하고자 한다. 페어와이즈 테스트를 위한 인자 조합이 <표 2>와 같다고 할 때, (가)~(라)에 들어갈 내용으로 옳은 것은?

<표 1>

	인자		
	A	B	C
값	A1	B1	C1
	A2	B2	C2
	A3		

<표 2>

A	B	C
A1	B1	C1
A1	B2	C2
A2	(가)	C1
A2	(나)	C2
A3	(다)	C1
A3	(라)	C2

① (가): B1, (나): B2, (다): B1, (라): B2

② (가): B1, (나): B1, (다): B2, (라): B2

③ (가): B1, (나): B2, (다): B2, (라): B1

④ (가): B2, (나): B2, (다): B1, (라): B1

정답 3번

NoteBase

출제 포인트

이 문제는 페어와이즈 테스트에서 A-B, A-C, B-C의 모든 값쌍이 한 번 이상 나타나도록 남은 B값을 채울 수 있는지를 묻는 문제이다.

자료 읽는 순서

1. 인자쌍 A-B, A-C, B-C에서 만들어져야 하는 전체 조합 수를 먼저 정리한다.
2. 표 2에서 A와 C의 배치만으로 A-C 쌍이 이미 모두 충족되는지 확인한다.
3. 남은 B값이 A-B의 누락쌍과 B-C의 누락쌍을 동시에 메우는지 본다.

선지별 해설

1번: X. (가)=B1, (나)=B2, (다)=B1, (라)=B2이면 A-B의 6개 쌍은 채워지지만 B-C가 B1-C1과 B2-C2만 반복된다. 그래서 B1-C2와 B2-C1이 빠져 페어와이즈 조건을 만족하지 못한다. 올바르게는 남은 네 행에서 B1-C2와 B2-C1이 모두 한 번씩 나오도록 (가)=B1, (나)=B2, (다)=B2, (라)=B1로 교차 배치되어야 한다.
2번: X. (가)=B1, (나)=B1, (다)=B2, (라)=B2이면 B-C의 네 쌍은 모두 나오지만 A2는 B1과만, A3는 B2와만 연결된다. 그래서 A2-B2와 A3-B1이 빠진다. 올바르게는 A2와 A3가 각각 B1, B2를 한 번씩 가져야 하므로 (가)=B1, (나)=B2, (다)=B2, (라)=B1이어야 한다.
3번: O. (가)=B1, (나)=B2, (다)=B2, (라)=B1이면 A2는 B1, B2를 모두 가지고 A3도 B2, B1을 모두 가진다. 따라서 A-B의 6개 쌍이 전부 충족된다. 또 B-C는 B1-C1, B2-C2가 앞의 두 행에 있고, B2-C1과 B1-C2가 뒤의 네 행에서 추가되어 4개 쌍이 모두 완성된다. A-C는 표 2의 행 배치만으로 이미 A1, A2, A3 각각에 대해 C1과 C2가 모두 나타나므로 충족된다.
4번: X. (가)=B2, (나)=B2, (다)=B1, (라)=B1이면 A2는 B2와만, A3는 B1과만 연결된다. 그래서 A2-B1과 A3-B2가 빠진다. B-C의 네 쌍이 보이더라도 A-B 완전조합이 아니므로 페어와이즈 조건을 만족하지 못한다. 올바르게는 A2와 A3의 B값이 서로 교차되게 배치되어 (가)=B1, (나)=B2, (다)=B2, (라)=B1이 되어야 한다.

오답 포인트

B-C 쌍 네 개만 채워지면 충분하다고 보면 2번이나 4번을 고르기 쉽다. 페어와이즈는 모든 인자쌍에 대해 전체 조합이 한 번 이상 나와야 하므로 A-B, A-C, B-C를 각각 따로 점검해야 한다.

다시 나오면 이렇게 판단

페어와이즈 문제는 인자 두 개씩 묶어 필요한 전체 쌍을 먼저 적는다. 그다음 이미 충족된 쌍과 누락된 쌍을 나누어 보고, 남은 칸이 누락쌍을 모두 메우는지 확인하면 된다.

암기 포인트

페어와이즈는 전체 경우의 수를 다 만드는 기법이 아니라 모든 인자쌍을 한 번 이상 덮는 기법이다.

NoteBase

문 24. 소프트웨어의 총 개발 기간은 16개월이며, 처음 6명의 개발 인력이 100% 투입되었다. 개발지연으로 11개월 시작 시점에 3명 중 2명은 100%, 1명은 50%를 추가 투입하여 개발이 완료되었다. 이 소프트웨어 개발에 투입된 총 MM(Man-Month)은?

① 111 MM

② 112 MM

③ 113 MM

④ 114 MM

정답 1번

출제 포인트

이 문제는 개발 인력의 투입률과 투입 기간을 구분하여 총 투입 공수 *MM*를 계산할 수 있는지를 묻는 문제이다.

조건 정리

주어진 값: 처음 6명은 100% 투입, 총 개발 기간은 16개월이다.
추가 조건: 11개월 시작 시점부터 3명이 추가되고, 그중 2명은 100%, 1명은 50% 투입이다.
구해야 할 값: 전체 개발에 투입된 총 *MM*이다.
주의할 조건: 11개월 시작 시점은 11개월차의 시작이므로 추가 인력의 투입 기간은 11, 12, 13, 14, 15, 16개월의 6개월이다.

사용할 공식

총 *MM*는 각 인력의 투입률을 반영한 인원 수에 투입 개월 수를 곱해 모두 더한 값이다.
총 MM = Σ(투입인원 × 투입률 × 투입기간)
이 공식을 쓰는 이유는 사람 수만 세면 안 되고, 50% 투입처럼 부분 투입 인력도 공수로 환산해야 하기 때문이다.

계산 과정

처음 6명의 기본 투입 공수는 $6 \times 16 = 96$*MM*이다.
추가 인력의 투입 공수는 $(2 \times 1.0 + 1 \times 0.5) \times 6 = 2.5 \times 6 = 15$*MM*이다.
따라서 총 투입 공수는 $96 + 15 = 111$*MM*이다.

NoteBase

선지 판단

1번: O. 계산 결과가 111*MM*이므로 맞다.
2번: X. 112*MM*는 추가 투입 공수를 15가 아닌 16으로 잘못 본 값이다. 올바르게는 추가 인력 2.5명이 6개월 투입되므로 15*MM*를 더해야 한다.
3번: X. 113*MM*는 추가 인력의 투입률이나 기간을 과대 계산한 값이다. 올바르게는 1명을 100%가 아니라 50%로 반영하여 총합을 111*MM*로 계산해야 한다.
4번: X. 114*MM*는 추가된 3명을 모두 100%로 보고 $3 \times 6 = 18$*MM*를 더할 때 나오는 값이다. 올바르게는 3명 전원이 아니라 2명만 100%이고 1명은 50%이다.

오답 포인트

가장 흔한 함정은 11개월 시작 시점을 남은 5개월로 잘못 보거나, 50% 투입 1명을 100% 한 명처럼 계산하는 점이다. 이 문제는 기간 해석과 투입률 반영을 동시에 정확히 해야 한다.

다시 나오면 이렇게 판단

인력 공수 문제는 먼저 구간별로 기간을 나누고, 다음으로 각 구간의 실질 투입 인원수를 투입률까지 반영해 계산한 뒤, 마지막에 구간 공수를 모두 합산하는 방식으로 판단한다.

암기 포인트

총 *MM*는 사람 수의 단순 합이 아니라 투입률을 반영한 인원 × 기간의 합이다.

NoteBase

문 25. 다음은 모듈을 구성하는 코드의 일부이다. 두 함수 A()와 B() 사이에 형성되는 결합으로 옳은 것은? (단, ...은 생략된 코드이며, 해당 코드는 고려하지 않는다)

```
int A(){
    int a;
     ...
    B(a);
     ...
}
int B(int x){
    ...
    if(x > 0)
        /*  send data   */
    else
        /*  receive data */
}
```

① 내용 결합

② 자료 결합

③ 스탬프 결합

④ 제어 결합

정답 4번

출제 포인트

이 문제는 모듈 간 결합도에서 자료 결합과 제어 결합을 구별할 수 있는지를 묻는다.

핵심 개념

자료 결합은 한 모듈이 다른 모듈에 처리 대상 데이터만 전달하는 결합이다.
제어 결합은 한 모듈이 다른 모듈의 내부 처리 흐름을 바꾸는 제어 정보까지 전달하는 결합이다.
스탬프 결합은 필요한 일부만 써도 레코드나 구조체 같은 복합 자료구조 전체를 넘기는 결합이다.
내용 결합은 한 모듈이 다른 모듈의 내부 데이터나 내부 로직을 직접 참조하거나 수정하는 결합이다.

NoteBase

선지별 해설

1번 X: 틀린 표현은 "내용 결합"이다. A()는 B()의 내부 변수나 내부 코드를 직접 참조하지 않고 단지 인자 x를 넘긴다. 내용 결합이 아니라 함수 호출로 값을 전달하는 형태이다.
2번 X: 틀린 표현은 "자료 결합"이다. 자료 결합은 전달한 값이 단순 처리 데이터일 때 성립한다. 이 코드의 x는 B() 내부에서 if(x > 0) 조건을 결정하여 send와 receive 중 어느 경로를 탈지 정하므로, 단순 데이터가 아니라 제어 정보 역할까지 한다. 올바르게는 제어 결합이다.
3번 X: 틀린 표현은 "스탬프 결합"이다. 스탬프 결합은 배열, 레코드, 구조체처럼 묶인 자료구조 전체를 넘길 때 성립한다. 여기서는 정수형 변수 a 하나만 인자로 전달하므로 스탬프 결합이 아니다. 올바르게는 제어 결합이다.
4번 O: A()가 넘긴 값 a는 B()의 내부 분기 조건 x > 0을 결정한다. 전달된 인자가 B()의 처리 흐름 자체를 바꾸므로 두 함수 사이의 결합은 제어 결합이다.

오답 포인트

인자를 하나만 넘기면 곧바로 자료 결합으로 처리하기 쉽지만, 이 문제의 핵심은 인자의 개수가 아니라 그 인자의 역할이다. 값이 계산 대상 데이터인지, 내부 분기를 지시하는 제어 정보인지를 구별하지 못하면 2번으로 오답 처리하기 쉽다.

다시 나오면 이렇게 판단

모듈 간 전달값이 상대 모듈의 연산 대상인지 먼저 본다. 그다음 그 값이 if, case, 반복 여부처럼 내부 흐름을 바꾸는지 확인한다. 흐름을 바꾸면 제어 결합이고, 처리 데이터만 넘기면 자료 결합이다.

암기 포인트

데이터만 넘기면 자료 결합, 분기까지 지시하면 제어 결합이다.

NoteBase