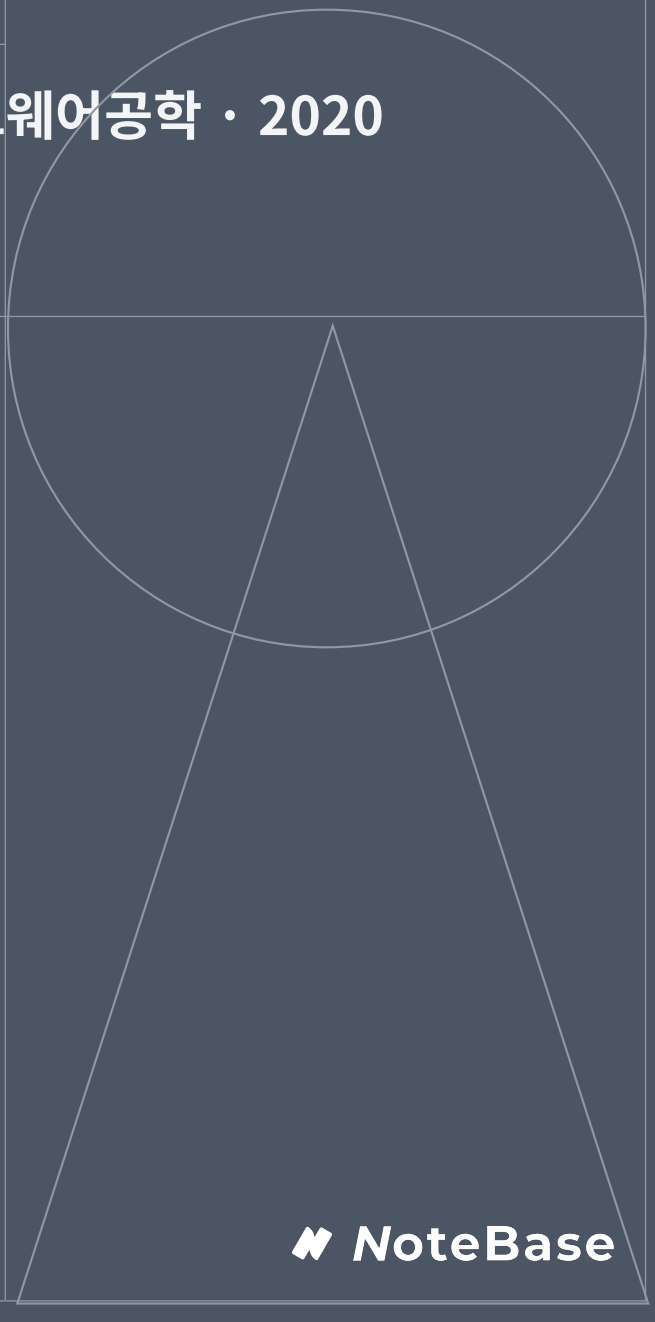
기출OX 해설집

공무원/국가직/7급 · 소프트웨어공학 · 2020



NoteBase

기출OX 해설집

공무원/국가직/7급 · 소프트웨어공학 · 2020

공무원/국가직/7급 · 소프트웨어공학 · 2020

문 1. 다음 설명에 해당하는 소프트웨어 개발 프로세스 방법은?

◯ 설계가 변경되어도 이를 잘 수용할 수 있도록 짧게 반복하면서 소프트웨어를 개발하는 방법
◯ 반복적이고 진화적인 프로세스와 유사하지만, 경험의 축적으로 나온 모델
◯ 소프트웨어 개발이 인력 집약적이라는 데 관심을 두고 사람과 팀 구성에 초점을 맞춤

① RAD 모델(Rapid Application Development Model)
② 반복/점증 모델(Iterative/Incremental Model)
③ 나선형 모델(Spiral Model)
④ 애자일 프로세스 모델(Agile Process Model)

정답 4번

출제 포인트

이 문제는 애자일 프로세스 모델과 반복/점증 모델, RAD 모델, 나선형 모델의 핵심 차이를 구분하는 문제이다.

핵심 개념

애자일 프로세스 모델은 짧은 반복 주기로 개발하면서 변화하는 요구를 수용하고, 사람과 상호작용, 팀 협업을 중시하는 개발 방법이다.
반복/점증 모델은 기능을 나누어 반복적으로 개발하고 점진적으로 완성도를 높이는 모델이지만, 사람과 팀 구성을 핵심 가치로 전면에 두는 모델은 아니다.
RAD 모델은 프로토타이핑, CASE 도구, 재사용 등을 활용해 개발 기간을 단축하는 신속 개발 모델이다.
나선형 모델은 반복 개발에 위험 분석과 위험 관리를 결합한 모델이다.

NoteBase

선지별 해설

① X
제시된 설명은 RAD 모델이 아니라 애자일 프로세스 모델에 해당한다. RAD 모델은 빠른 개발과 프로토타이핑이 핵심이며, 설계 변경 수용과 사람·팀 중심이라는 설명이 핵심 특징은 아니다.
틀린 표현은 모델 선택 자체이다. 올바르게는 RAD 모델이 아니라 애자일 프로세스 모델이다.

② X
반복/점증 모델은 짧은 반복과 점진적 산출물 제공이라는 점에서는 유사하지만, 제시문처럼 경험의 축적으로 정립되었고 사람과 팀 구성에 초점을 맞춘다는 점까지 포함하면 애자일 프로세스 모델을 가리킨다.
틀린 표현은 모델 선택 자체이다. 올바르게는 반복/점증 모델이 아니라 애자일 프로세스 모델이다.

③ X
나선형 모델은 반복성과 점진성을 가지지만 핵심은 위험 분석과 위험 관리이다. 제시문에는 위험 분석이 없고, 오히려 변화 수용과 팀 중심 개발이 강조되어 있으므로 맞지 않는다.
틀린 표현은 모델 선택 자체이다. 올바르게는 나선형 모델이 아니라 애자일 프로세스 모델이다.

④ O
애자일 프로세스 모델은 짧은 주기의 반복 개발로 변화하는 요구를 수용하고, 개인과 상호작용, 팀 협업을 중시한다. 제시문의 세 설명을 가장 직접적으로 만족하는 선택지이다.

오답 포인트

반복 개발이라는 공통점 때문에 ② 반복/점증 모델과 ④ 애자일 프로세스 모델을 혼동하기 쉽다. 이 문항은 반복 여부 자체보다 변화 수용, 경험 축적, 사람과 팀 중심이라는 애자일의 고유 키워드를 잡는지가 핵심이다.

다시 나오면 이렇게 판단

짧은 반복, 요구 변화 수용, 사람과 팀 중심이 함께 나오면 애자일 프로세스 모델로 판단한다. 빠른 개발과 프로토타이핑이면 RAD, 위험 분석이 나오면 나선형, 기능을 점진적으로 추가하는 데 초점이 있으면 반복/점증 모델로 구분한다.

암기 포인트

변화 수용·사람 중심은 애자일, 신속 개발은 RAD, 위험 분석은 나선형, 점진 완성은 반복/점증이다.

NoteBase

문 2. 클래스(Class) 다이어그램에 대한 설명으로 옳지 않은 것은?

① 개별 사례들을 보고 공통점을 찾아 인식하는 확장형(extensional)이 있다.

② 일반적인 개념의 속성, 동작들을 안에서 파악하여 적용하는 내재형(intentional)이 있다.

③ 오퍼레이션이나 처리 과정이 수행되는 동안 일어나는 일들을 단계적으로 표현하고자 할 때 사용하는 다이어그램이다.

④ 클래스 다이어그램은 도메인 개념과 속성 및 관계를 나타내는 중요한 모델이다.

정답 3번

출제 포인트

이 문제는 클래스 다이어그램의 정적 모델 성격과 클래스 식별 방식인 확장형, 내재형 개념을 구별하는 문제이다.

핵심 개념

클래스 다이어그램은 시스템이나 도메인의 클래스, 속성, 연산, 관계를 표현하는 정적 구조 모델이다.
확장형은 여러 개별 사례를 관찰해 공통 속성을 추출하여 클래스를 인식하는 방식이다.
내재형은 개념이 본래 가지는 속성과 동작을 분석해 클래스를 정의하는 방식이다.
동작의 시간 흐름이나 처리 순서를 단계적으로 나타내는 것은 클래스 다이어그램이 아니라 활동 다이어그램이나 상호작용 다이어그램 같은 동적 모델에 가깝다.

선지별 해설

① O. 개별 객체나 사례를 보고 공통점을 묶어 클래스를 도출하는 방식이 확장형이다. 사례에서 공통 속성을 뽑아 일반화한다는 설명과 맞는다.
② O. 일반적 개념이 가진 속성과 동작을 내부적으로 분석해 클래스를 정의하는 방식이 내재형이다. 개념의 본질적 특성을 보고 적용한다는 설명과 맞는다.
③ X. 틀린 부분은 "오퍼레이션이나 처리 과정이 수행되는 동안 일어나는 일들을 단계적으로 표현"한다는 표현이다. 이런 시간 흐름과 절차 표현은 클래스 다이어그램이 아니라 활동 다이어그램, 순차 다이어그램 등 동적 다이어그램의 역할이다. 올바르게는 클래스 다이어그램은 클래스와 그 속성, 연산, 관계 같은 정적 구조를 표현한다.
④ O. 클래스 다이어그램은 도메인 개념과 속성 및 관계를 나타내는 대표적인 모델이다. 분석 단계에서 문제 영역의 구조를 파악하는 데 핵심적으로 쓰인다.

오답 포인트

③은 UML 다이어그램의 정적 모델과 동적 모델을 섞어 놓은 선지이다. 클래스 다이어그램에 연산이라는 단어가 들어가므로 처리 순서까지 표현한다고 오해하기 쉽지만, 연산 목록을 적는 것과 실행 과정을 시간 순서로 표현하는 것은 다르다.

NoteBase

다시 나오면 이렇게 판단

다이어그램 문제는 먼저 정적 구조를 그리는지, 시간 흐름과 절차를 그리는지부터 구분하면 된다. 클래스, 속성, 관계를 묻으면 클래스 다이어그램이고, 단계적 수행 과정이나 메시지 흐름을 묻으면 다른 동적 다이어그램으로 판단한다.

암기 포인트

클래스 다이어그램은 구조, 활동·순차 다이어그램은 흐름이라는 축으로 구분한다.

NoteBase

문 3. 사용자 인터페이스를 설계할 때 고려 사항이 아닌 것은?

① 소프트웨어 내부 모듈 사이의 인터페이스를 정의하거나 외부 시스템 사이의 인터페이스를 정의한다.

② 사용자가 어떤 액션을 취하였는지 아니면 오류가 발생했는지 알지 못할 때는 시스템이 적절히 지원하는 것은 중요하다.

③ 사용자 인터페이스는 일관성이 있어야 한다.

④ 사용자 인터페이스는 사용자로부터의 입력이 최소화되도록 해야 한다.

정답 1번

출제 포인트

이 문제는 사용자 인터페이스 설계 원칙과 시스템 내부·외부 인터페이스 정의를 구분할 수 있는지를 묻는 문제이다.

핵심 개념

사용자 인터페이스는 사용자와 시스템이 상호작용하는 화면, 입력, 피드백 방식을 설계하는 것이다.
시스템 인터페이스는 소프트웨어 내부 모듈 간 또는 외부 시스템 간의 연결 규칙을 정의하는 것이다.
사용자 인터페이스 설계의 대표 원칙은 일관성 유지, 사용자 입력 최소화, 적절한 피드백 제공이다.

NoteBase

선지별 해설

① O
소프트웨어 내부 모듈 사이 또는 외부 시스템 사이의 인터페이스 정의는 시스템 설계나 아키텍처 설계의 내용이다. 사용자와 시스템의 상호작용을 다루는 사용자 인터페이스 설계 고려 사항과는 직접적인 범주가 다르므로 정답이다.

② X
사용자가 어떤 동작을 했는지, 오류가 발생했는지 알 수 없을 때 시스템이 적절한 피드백과 안내를 제공하는 것은 사용자 인터페이스 설계의 핵심 원칙이다.
틀린 판단은 이 내용을 '고려 사항이 아니다'라고 보는 부분이다. 올바르게는 사용자 인터페이스 설계 시 반드시 고려해야 하는 사항이다.

③ X
일관성은 메뉴 구성, 용어, 버튼 동작, 화면 흐름을 예측 가능하게 만들어 사용 오류를 줄인다. 따라서 사용자 인터페이스는 일관성이 있어야 한다.
틀린 판단은 '일관성은 고려 사항이 아니다'라고 보는 부분이다. 올바르게는 일관성은 대표적인 사용자 인터페이스 설계 원칙이다.

④ X
사용자 입력을 최소화하면 입력 부담과 오류 가능성이 줄어든다. 기본값 제공, 자동 완성, 선택형 입력 등이 이에 해당한다.
틀린 판단은 '사용자 입력 최소화는 고려 사항이 아니다'라고 보는 부분이다. 올바르게는 사용자 인터페이스는 사용자 입력이 최소화되도록 설계하는 것이 바람직하다.

오답 포인트

인터페이스라는 단어만 보고 ①을 사용자 인터페이스로 오인하기 쉽다. 그러나 ①은 사용자와의 상호작용이 아니라 모듈 간, 시스템 간 연결을 설명하므로 범주가 다르다.

다시 나오면 이렇게 판단

발문이 사용자 인터페이스 설계 원칙을 묻는다면 먼저 대상이 '사용자와 시스템의 상호작용'인지 확인한다. 모듈 간 연결, 외부 시스템 연동처럼 시스템 구조를 설명하면 사용자 인터페이스가 아니라는 기준으로 판단한다.

암기 포인트

UI는 사용자와의 상호작용, 시스템 인터페이스는 모듈·시스템 간 연결이다.

NoteBase

문 4. 소프트웨어 개발 단계에 따른 테스트의 순서는?

ㄱ. 통합 테스트
ㄴ. 인수 테스트
ㄷ. 단위 테스트
ㄹ. 시스템 테스트

① ㄷ - ㄱ - ㄴ - ㄹ
② ㄷ - ㄱ - ㄹ - ㄴ
③ ㄷ - ㄴ - ㄱ - ㄹ
④ ㄷ - ㄹ - ㄱ - ㄴ

정답 2번

출제 포인트

이 문제는 소프트웨어 개발 단계에 따른 테스트 수행 순서를 묻는 문제이다.

올바른 순서

1. 단위 테스트 - 개별 모듈이나 함수가 설계대로 동작하는지 확인하는 단계이다.
2. 통합 테스트 - 단위 테스트를 마친 모듈들을 결합하여 인터페이스와 상호작용을 검증하는 단계이다.
3. 시스템 테스트 - 통합된 전체 시스템이 요구사항을 만족하는지 검증하는 단계이다.
4. 인수 테스트 - 사용자나 고객이 실제 업무 관점에서 최종 수용 여부를 판단하는 단계이다.

왜 이 순서: 개별 모듈 검증이 끝나야 모듈 결합을 볼 수 있고, 전체 시스템 검증이 끝나야 사용자 수용 여부를 판단할 수 있다.

선지별 해설

① X: ㄴ과 ㄹ의 순서가 틀렸다. 인수 테스트는 시스템 테스트 뒤에 수행된다. 틀린 표현은 '단위 - 통합 - 인수 - 시스템'이고, 올바르게는 '단위 - 통합 - 시스템 - 인수'이다.
② O: ㄷ - ㄱ - ㄹ - ㄴ은 단위 테스트 후 통합 테스트, 그다음 시스템 테스트, 마지막 인수 테스트의 표준 흐름과 일치한다.
③ X: 단위 테스트 다음에 바로 인수 테스트가 오는 배열이 틀렸다. 인수 테스트는 통합 테스트와 시스템 테스트를 모두 거친 뒤에 수행된다. 틀린 표현은 '단위 - 인수 - 통합 - 시스템'이고, 올바르게는 '단위 - 통합 - 시스템 - 인수'이다.
④ X: 시스템 테스트와 통합 테스트의 선후가 바뀌었다. 전체 시스템 시험은 모듈 간 통합이 끝난 뒤에 가능하다. 틀린 표현은 '단위 - 시스템 - 통합 - 인수'이고, 올바르게는 '단위 - 통합 - 시스템 - 인수'이다.

NoteBase

오답 포인트

통합 테스트와 시스템 테스트, 시스템 테스트와 인수 테스트를 비슷한 최종 점검 단계로 묶어 외우면 순서를 바꾸기 쉽다. 이 문제의 함정은 '누가 검증하는가'와 '무엇을 대상으로 검증하는가'를 구분하지 않으면 ①이나 ④로 흔들린다는 점이다.

다시 나오면 이렇게 판단

테스트 순서 문제는 검증 대상을 작은 단위에서 큰 단위로 확장해 판단하면 된다. 모듈 단위 확인 후 모듈 결합을 보고, 그다음 전체 시스템을 검증한 뒤, 마지막에 사용자가 수용 여부를 판단하는 흐름이다.

암기 포인트

단통시인이다. 단위 → 통합 → 시스템 → 인수의 흐름으로 기억한다.

NoteBase

문 5. ISO의 소프트웨어 프로세스 평가를 위한 국제 표준인 SPICE에 대한 설명이다. 이에 해당하는 프로세스 범주는?

시스템과 소프트웨어 제품을 개발하는 모든 프로세스, 즉 요구사항 분석(명세화), 설계, 구현, 테스트 등이 이 범주에 속한다.

① 조직 프로세스(Organization Process)

② 공학 프로세스(Engineering Process)

③ 고객-공급 프로세스(Customer-Supplier Process)

④ 지원 프로세스(Support Process)

정답 2번

출제 포인트

이 문제는 SPICE의 프로세스 범주 중 제품 개발 활동을 담당하는 공학 프로세스를 다른 범주와 구별하는 문제이다.

핵심 개념

공학 프로세스는 시스템과 소프트웨어 제품을 실제로 개발하는 범주로서 요구사항 분석, 설계, 구현, 통합, 테스트 같은 개발 활동이 들어간다.
조직 프로세스는 조직 차원의 관리, 개선, 인력, 인프라 같은 기반을 다루는 범주이다.
고객-공급 프로세스는 획득, 공급, 계약, 납품처럼 고객과 공급자 사이의 거래와 제공 관계를 다루는 범주이다.
지원 프로세스는 문서화, 형상관리, 품질보증, 검증, 확인, 감사처럼 개발을 보조하는 공통 지원 활동을 다루는 범주이다.

NoteBase

선지별 해설

① X: 조직 프로세스는 조직 운영과 개선을 위한 범주이지 제품 개발 단계 자체를 뜻하지 않는다. 틀린 부분은 '요구사항 분석, 설계, 구현, 테스트'를 조직 프로세스로 보는 점이다. 올바르게는 이런 활동은 공학 프로세스에 속한다.
② O: 제시문은 시스템과 소프트웨어 제품을 개발하는 모든 프로세스를 말하고 있고, 예시로 요구사항 분석, 설계, 구현, 테스트를 들고 있으므로 공학 프로세스에 해당한다.
③ X: 고객-공급 프로세스는 고객 요구의 획득, 공급 계약, 납품과 같은 외부 거래 관계를 다루는 범주이다. 틀린 부분은 개발 내부 활동을 고객-공급 프로세스로 본 점이다. 올바르게는 고객-공급 프로세스라면 획득이나 공급 같은 계약·제공 활동이 제시되어야 한다.
④ X: 지원 프로세스는 개발을 돕는 공통 활동 범주이지 요구사항 분석부터 구현·테스트까지의 본개발 흐름을 뜻하지 않는다. 틀린 부분은 제품 개발 활동 전체를 지원 프로세스로 본 점이다. 올바르게는 지원 프로세스라면 형상관리, 품질보증, 검증, 확인, 감사 같은 활동이 제시되어야 한다.

오답 포인트

수험생은 제시문에 '테스트'가 들어 있으면 검증·확인과 비슷하다고 보고 지원 프로세스로 오인하기 쉽다.
그러나 여기서는 요구사항 분석, 설계, 구현, 테스트를 한 묶음의 개발 흐름으로 제시하므로 공학 프로세스로 판단해야 한다.

다시 나오면 이렇게 판단

설명에 요구분석, 설계, 구현, 통합, 테스트처럼 제품을 직접 만드는 단계가 나오면 공학 프로세스로 본다. 계약, 획득, 납품이면 고객-공급 프로세스이고, 형상관리·품질보증·검증이면 지원 프로세스이며,
개선·교육·인프라면 조직 프로세스로 구분한다.

암기 포인트

만드는 일은 공학, 거래는 고객-공급, 돕는 일은 지원, 조직 기반은 조직 프로세스이다.

NoteBase

문 6. 다음은 UML2.0 다이어그램에 대한 내용이다. <보기 1>과 <보기 2>를 바르게 연결한 것은?

<보기 1>
ㄱ. 시스템의 컴파일 시 계층적 구조를 기술하는 다이어그램이다.
ㄴ. 시퀀스 다이어그램과 액티비티 다이어그램을 혼합한 다이어그램이다.
ㄷ. 하나의 클래스 실행 시 내부 구조를 상세하게 표현하고자 할 때 사용하는 복합 구조를 표현하는 다이어그램이다.
<보기 2>
A. 인터랙션 오버뷰 다이어그램(Interaction Overview Diagram)
B. 컴포지트 스트럭처 다이어그램(Composite Structure Diagram)
C. 패키지 다이어그램(Package Diagram)

① ㄱ: A, ㄴ: B, ㄷ: C
② ㄱ: C, ㄴ: A, ㄷ: B
③ ㄱ: A, ㄴ: C, ㄷ: B
④ ㄱ: C, ㄴ: B, ㄷ: A

정답 2번

출제 포인트

이 문제는 UML 2.0 다이어그램의 정의를 보고 패키지 다이어그램, 인터랙션 오버뷰 다이어그램, 컴포지트 스트럭처 다이어그램을 정확히 구별할 수 있는지를 묻는다.

핵심 개념

패키지 다이어그램은 시스템을 패키지 단위로 묶어 컴파일 시점의 계층적 구조와 의존관계를 표현하는 다이어그램이다.
인터랙션 오버뷰 다이어그램은 제어 흐름은 액티비티처럼 표현하면서 각 노드에 상호작용을 배치해 시퀀스 다이어그램과 액티비티 다이어그램의 성격을 함께 쓰는 다이어그램이다.
컴포지트 스트럭처 다이어그램은 클래스나 컴포넌트 내부의 구조, 역할, 연결을 보여 주어 실행 시 내부 협력 구조를 상세히 표현하는 다이어그램이다.

NoteBase

선지별 해설

1번: X. ㄱ을 A로, ㄴ을 B로, ㄷ을 C로 연결한 점이 틀리다. ㄱ의 "시스템의 컴파일 시 계층적 구조"는 패키지 다이어그램이므로 C여야 한다. ㄴ의 "시퀀스 다이어그램과 액티비티 다이어그램을 혼합"은 인터랙션 오버뷰 다이어그램이므로 A여야 한다. ㄷ의 "클래스 실행 시 내부 구조를 상세히 표현"은 컴포지트 스트럭처 다이어그램이므로 B여야 한다.
2번: O. ㄱ을 C, ㄴ을 A, ㄷ을 B로 연결했으므로 각 설명과 UML 다이어그램 명칭이 정확히 대응한다.
3번: X. ㄷ을 B로 연결한 점만 맞고 ㄱ과 ㄴ의 연결이 틀리다. ㄱ은 A가 아니라 C여야 하고, ㄴ은 C가 아니라 A여야 한다.
4번: X. ㄱ을 C로 연결한 점은 맞지만 ㄴ과 ㄷ의 연결이 바뀌었다. ㄴ은 B가 아니라 A여야 하고, ㄷ은 A가 아니라 B여야 한다.

오답 포인트

패키지 다이어그램과 컴포지트 스트럭처 다이어그램을 둘 다 구조를 나타내는 그림으로만 외우면 ㄱ과 ㄷ을 바꾸기 쉽다. 또 인터랙션 오버뷰 다이어그램은 이름에 sequence가 직접 들어 있지 않아 ㄴ을 다른 구조 다이어그램으로 오인하기 쉽다.

다시 나오면 이렇게 판단

설명에 "컴파일 시 계층 구조"가 나오면 패키지 다이어그램, "시퀀스와 액티비티의 결합"이 나오면 인터랙션 오버뷰 다이어그램, "클래스 내부의 실행 구조"가 나오면 컴포지트 스트럭처 다이어그램으로 연결하면 된다.

암기 포인트

패키지는 정적 계층 구조, 인터랙션 오버뷰는 시퀀스+액티비티, 컴포지트 스트럭처는 클래스 내부 실행 구조이다.

NoteBase

문 7. 다음 파이썬(Python) 코드에 적용된 디자인 패턴은?

```
from abc import ABCMeta, abstractmethod
class Order(metaclass=ABCMeta):
   @abstractmethod
   def execute(self):
      pass
class BuyStockOrder(Order):
   def __init__(self, stock):
      self.stock = stock
   def execute(self):
      self.stock.buy()
class SellStockOrder(Order):
   def __init__(self, stock):
      self.stock = stock
   def execute(self):
      self.stock.sell()
class StockTrade:
   def buy(self):
      print("You will buy stocks")
   def sell(self):
      print("You will sell stocks")
class Agent:
   def __init__(self):
      self.__orderQueue = []
   def placeOrder(self, order):
      self.__orderQueue.append(order)
      order.execute()
if __name__ == '__main__':
   #Client
   stock = StockTrade()
   buyStock = BuyStockOrder(stock)
   sellStock = SellStockOrder(stock)
   #Invoker
   agent = Agent()
   agent.placeOrder(buyStock)
   agent.placeOrder(sellStock)
```

① Interpreter Pattern

② Strategy Pattern

③ Memento Pattern

④ Command Pattern

정답 4번

출제 포인트

이 문제는 요청을 객체로 캡슐화하는 커맨드 패턴과 다른 행위 패턴의 차이를 묻는 문제이다.

핵심 개념

커맨드 패턴은 실행할 요청 자체를 객체로 만들어 호출자와 실제 수행 대상을 분리하는 패턴이다. 이 코드에서 `Order`는 Command 인터페이스, `BuyStockOrder`와 `SellStockOrder`는 Concrete Command, `StockTrade`는 Receiver, `Agent`는 Invoker에 해당한다.
인터프리터 패턴은 문법 규칙을 클래스로 표현하고 `interpret()` 같은 연산으로 문장을 해석하는 패턴이다.
전략 패턴은 동일한 목적의 알고리즘들을 교체 가능하게 캡슐화하는 패턴이다.
메멘토 패턴은 객체의 내부 상태를 저장해 두었다가 나중에 복원하는 패턴이다.

NoteBase

선지별 해설

① X
인터프리터 패턴이라면 문법 규칙, 비터미널/터미널 표현식, 해석 연산이 중심이 된다. 이 코드는 문자열이나 문장을 해석하지 않고 `execute()`로 매수·매도 요청을 실행한다.
틀린 표현은 `Interpreter Pattern`이다. 올바르게는 요청 객체를 실행하는 `Command Pattern`이다.

② X
전략 패턴이라면 동일한 문제를 푸는 여러 알고리즘을 교체하는 구조가 나와야 한다. 이 코드는 알고리즘 선택이 아니라 매수·매도 요청을 객체로 만들어 `Agent`가 호출하도록 분리한 구조이다.
틀린 표현은 `Strategy Pattern`이다. 올바르게는 요청 캡슐화와 호출자-수신자 분리가 핵심인 `Command Pattern`이다.

③ X
메멘토 패턴이라면 객체 상태를 저장하는 Memento와 이를 관리하는 Caretaker가 나타나야 한다. 이 코드에는 상태 저장·복원 로직이 없고 즉시 실행되는 주문 객체만 있다.
틀린 표현은 `Memento Pattern`이다. 올바르게는 `execute()`를 가진 주문 객체가 동작을 캡슐화한 `Command Pattern`이다.

④ O
`BuyStockOrder`, `SellStockOrder`가 각각 매수·매도 요청을 객체로 캡슐화하고, `Agent`가 이를 받아 실행한다. 실제 작업은 `StockTrade`의 `buy()`, `sell()`이 수행하므로 전형적인 커맨드 패턴 구조이다.

오답 포인트

`execute()` 메서드가 있다는 점만 보고 전략 패턴으로 오인하기 쉽다. 그러나 전략 패턴은 알고리즘 교체가 핵심이고, 이 문제는 요청 자체를 객체로 만든 뒤 Invoker가 호출한다는 점이 핵심이다. 상태 저장 객체가 없으므로 메멘토 패턴도 아니다.

다시 나오면 이렇게 판단

추상 명령 인터페이스가 있고 구체 명령 객체가 Receiver를 들고 있으며 Invoker가 그 명령을 받아 실행하면 커맨드 패턴으로 판단한다. 반대로 해석 규칙이면 인터프리터, 알고리즘 교체면 전략, 상태 저장·복원이면 메멘토로 구분한다.

암기 포인트

커맨드 패턴은 요청을 객체로 만든다. Invoker-Agent, Receiver-StockTrade, Command-Order 구조가 보이면 커맨드 패턴이다.

NoteBase

문 8. 비기능적 요구사항에 해당하는 것은?

① 항공편, 탑승객, 예약을 입력하는 방법을 결정해야 한다.

② 여행사와 고객이 데이터베이스에 접근할 때 어떤 정보를 얻을 수 있는지 결정해야 한다.

③ 자주 탑승하는 고객을 서비스하기 위해 시스템을 확장할 수 있도록 설계해야 한다.

④ 요금 계산 방법을 결정해야 한다.

정답 3번

출제 포인트

이 문제는 기능적 요구사항과 비기능적 요구사항을 구별하는 문제이다. 핵심은 시스템이 무엇을 해야 하는지를 묻는지, 아니면 어떤 품질과 제약을 만족해야 하는지를 묻는지 판별하는 데 있다.

핵심 개념

기능적 요구사항은 시스템이 제공해야 하는 기능, 처리, 입력, 출력, 계산 규칙을 정의한 것이다.
비기능적 요구사항은 성능, 신뢰성, 사용성, 보안성, 확장성처럼 시스템이 갖추어야 할 품질 속성이나 제약을 정의한 것이다.
판단축은 무엇을 하느냐이면 기능적 요구사항이고, 얼마나 잘 동작하고 얼마나 쉽게 확장되느냐이면 비기능적 요구사항이다.

NoteBase

선지별 해설

1번 X
'항공편, 탑승객, 예약을 입력하는 방법'은 시스템의 입력 기능과 인터페이스를 정하는 내용이다.
틀린 표현은 '비기능적 요구사항'에 해당한다고 보는 점이다.
올바르게는 시스템이 어떤 데이터를 어떻게 입력받을지를 정하는 기능적 요구사항이다.

2번 X
'여행사와 고객이 데이터베이스에 접근할 때 어떤 정보를 얻을 수 있는지'는 사용자별 조회 정보와 접근 범위를 정하는 내용이다.
틀린 표현은 이를 품질 속성으로 보는 점이다.
올바르게는 누가 어떤 정보를 조회할 수 있는지를 정의하는 기능적 요구사항이다.

3번 O
'자주 탑승하는 고객을 서비스하기 위해 시스템을 확장할 수 있도록 설계'는 확장성을 요구하는 내용이다.
확장성은 시스템이 미래 변화나 기능 추가를 수용할 수 있는 품질 속성이므로 비기능적 요구사항에 해당한다.

4번 X
'요금 계산 방법'은 시스템이 수행해야 할 업무 규칙과 처리 로직을 정하는 내용이다.
틀린 표현은 이를 비기능적 요구사항으로 보는 점이다.
올바르게는 요금을 어떤 기준으로 계산할지를 정의하는 기능적 요구사항이다.

오답 포인트

입력 방법, 조회 정보, 계산 방법처럼 업무 처리 내용을 묻는 문장을 비기능적 요구사항으로 착각하기 쉽다. 특히 2번은 '데이터베이스에 접근'이라는 표현 때문에 보안성으로 오해할 수 있지만, 이 선지는 실제로 어떤 정보를 제공할지를 묻고 있으므로 기능적 요구사항이다.

다시 나오면 이렇게 판단

선지가 시스템이 해야 할 일, 처리할 데이터, 계산 규칙, 제공할 정보를 말하면 기능적 요구사항이다. 선지가 성능, 보안성, 신뢰성, 사용성, 확장성처럼 시스템의 품질이나 제약을 말하면 비기능적 요구사항이다.

암기 포인트

무엇을 한다는 기능적 요구사항이고, 얼마나 잘 버티고 얼마나 쉽게 늘어나는가는 비기능적 요구사항이다.
확장성은 대표적인 비기능 키워드이다.

NoteBase

문 9. 형상 관리 요소 및 절차에 대한 설명으로 옳지 않은 것은?

① 형상 요소는 새 프로젝트가 시작될 때 파악한다.

② 소프트웨어 형상에 대한 정보를 추적하고 유지하기 위해 형상의 상태를 보관 및 관리한다.

③ 형상은 대상 시스템을 분석하여 높은 수준의 추상화된 정보를 추출하는 작업이다.

④ 시스템 형상의 일관성과 개발자 사이의 협력을 확인하기 위해 형상 항목에 대한 변경을 관리한다.

정답 3번

출제 포인트

이 문제는 형상관리의 핵심 절차와 역공학의 개념을 구별하는 문제이다.

핵심 개념

형상관리는 소프트웨어 산출물을 형상 항목으로 식별하고, 변경과 버전 상태를 통제·기록하여 일관성을 유지하는 활동이다. 결정 차이축은 변경 관리와 상태 추적 여부이다. 키워드는 형상 식별, 형상 통제, 형상 상태 보고이다.
역공학은 기존 시스템을 분석하여 상위 수준의 설계나 명세 정보를 추출하는 활동이다. 결정 차이축은 기존 시스템 분석을 통한 추상화 정보 추출 여부이다. 키워드는 분석, 추출, 추상화이다.

선지별 해설

① O: 형상 식별은 프로젝트 초기에 형상 항목을 정하고 기준선을 설정하는 절차이므로 맞다.
② O: 형상 상태 보고는 형상 항목의 버전, 변경 이력, 처리 상태를 기록하고 추적하는 활동이므로 맞다.
③ X: 틀린 부분은 "대상 시스템을 분석하여 높은 수준의 추상화된 정보를 추출"이라는 표현이다. 실제로는 역공학에 대한 설명이다. 형상관리는 형상 항목의 버전과 변경 사항을 식별하고 통제하며 상태를 기록하는 활동이다.
④ O: 형상 통제는 형상 항목의 변경을 심사하고 승인하며 반영 과정을 관리하여 시스템의 일관성과 개발자 간 협업을 유지하는 절차이므로 맞다.

오답 포인트

③은 형상관리 절차를 묻는 문항에 역공학 정의를 끼워 넣은 선지이다. "추상화된 정보를 추출"이라는 표현이 보이면 형상관리가 아니라 역공학으로 판단해야 한다. ②의 상태 보고와 ④의 변경 통제를 같은 절차로 뭉뚱그리면 오답이 된다.

다시 나오면 이렇게 판단

"버전·변경·상태 추적"이 중심이면 형상관리이고, "기존 시스템 분석·상위 정보 추출"이 중심이면 역공학으로 판단한다.

NoteBase

암기 포인트

형상관리는 변경을 관리하고, 역공학은 상위 정보를 추출한다.

NoteBase

문 10. 다음 설명에 해당하는 스크럼(Scrum) 관련 활동은?

스프린트가 끝나는 시점이나 일정 주기로 수행한다. 이 활동을 통하여 프로젝트를 진행하는 과정에서 드러난 좋았던 점, 여러 가지 문제나 미진한 점 등을 도출한다. 이 활동을 통해 이미 설정된 프로세스로만 프로젝트를 진행하지 않고 프로세스를 지속적으로 개선하여 변화하는 비즈니스 환경에 보다 능동적으로 적응할 수 있도록 한다.

① 스프린트 회고

② 스프린트 리뷰

③ 일일 스크럼 미팅

④ 릴리스 계획

정답 1번

출제 포인트

이 문제는 스크럼 이벤트 중 팀의 업무 방식과 협업 절차를 점검하고 개선하는 활동이 무엇인지 묻는 문제이다.

핵심 개념

스프린트 회고는 스프린트가 끝날 때 팀이 진행 과정의 잘된 점, 문제점, 개선점을 돌아보며 다음 스프린트의 작업 방식을 개선하는 활동이다.
스프린트 리뷰는 이해관계자와 함께 결과물 증분을 검토하고 제품 백로그를 조정하는 활동이다.
일일 스크럼 미팅은 매일 짧게 진행 상황과 장애 요소를 공유하며 당일 협업을 조정하는 활동이다.
릴리즈 계획은 배포 시점과 범위를 계획하는 활동으로, 팀 프로세스 회고와는 목적이 다르다.

선지별 해설

① O: 스프린트 종료 시점에 주기적으로 수행하고, 좋았던 점·문제점·미진한 점을 도출해 프로세스를 지속적으로 개선한다는 설명은 스프린트 회고의 정의와 일치한다.
② X: 틀린 부분은 결과물 검토 활동처럼 본 점이다. 스프린트 리뷰는 팀의 일하는 방식을 개선하는 회의가 아니라 이해관계자와 함께 증분을 검토하고 피드백을 반영하는 활동이다.
③ X: 틀린 부분은 수행 시점과 목적이다. 일일 스크럼 미팅은 스프린트 종료 시점이 아니라 매일 수행하며, 회고가 아니라 당일 업무 조율과 장애 요소 공유가 목적이다.
④ X: 틀린 부분은 활동 대상이다. 릴리즈 계획은 배포 일정과 범위를 계획하는 활동이지, 스프린트 수행 과정의 장단점과 개선점을 도출하는 회고 활동이 아니다.

NoteBase

오답 포인트

스프린트 리뷰와 스프린트 회고를 모두 스프린트 종료 무렵에 한다는 이유로 혼동하기 쉽다. 그러나 리뷰는 결과물과 이해관계자 피드백이 중심이고, 회고는 팀 프로세스 개선이 중심이라는 점이 결정적 차이이다.

다시 나오면 이렇게 판단

설명에 잘된 점, 문제점, 개선점, 프로세스 개선, 다음 스프린트에 반영과 같은 표현이 있으면 스프린트 회고로 판단할 수 있다. 결과물 시연과 피드백이면 리뷰이고, 매일 진행 상황 공유이면 일일 스크럼 미팅이다.

암기 포인트

리뷰는 결과물 점검, 회고는 작업 방식 개선이다.

NoteBase

문 11. 객체지향 설계가 갖는 특징으로 옳지 않은 것은?

① 객체지향 설계에서 중요한 것은 시스템을 구성하는 객체와 속성, 연산을 정의하는 것이다.

② 객체지향 설계는 하나의 커다란 작업을 여러 개의 작은 작업으로 분할하고, 분할된 각각의 소작업을 함수(모듈)로 구현하는 것이다.

③ 객체지향 설계에서는 주어진 객체의 특성을 분석하여 공통된 특징을 갖는 슈퍼클래스를 생성하는 추상화 기법을 통해 객체의 설계 비용과 시간을 줄일 수 있다.

④ 객체지향 설계에서는 캡슐화를 통해 객체의 세부내용 변경에 의해 발생될 수 있는 오류의 파급을 줄일 수 있다.

정답 2번

출제 포인트

이 문제는 객체지향 설계의 핵심 특징과 기능 분해 중심의 구조적 설계를 구별할 수 있는지를 묻는 문제이다.

핵심 개념

객체지향 설계는 객체를 중심으로 속성과 연산을 함께 묶고 객체 간 관계와 책임을 설계하는 방법이다.
구조적 설계는 큰 기능을 작은 기능으로 분해하여 함수나 모듈 단위로 구현하는 방법이다.
추상화는 공통 속성을 뽑아 상위 개념으로 일반화하는 것이고, 캡슐화는 객체 내부 구현을 감추어 변경의 영향을 줄이는 것이다.

NoteBase

선지별 해설

① O
시스템을 구성할 객체와 그 객체의 속성, 연산을 정의하는 것은 객체지향 설계의 기본이다. 객체는 데이터와 행위를 함께 다루는 단위이므로 이 설명은 옳다.

② X
틀린 부분은 "하나의 커다란 작업을 여러 개의 작은 작업으로 분할"하고 "함수(모듈)로 구현"한다는 표현이다. 이는 객체지향 설계가 아니라 기능 분해 중심의 구조적 설계 설명이다. 객체지향 설계는 큰 작업을 함수로 쪼개는 대신 객체의 책임과 상호작용을 기준으로 클래스와 객체를 설계한다.

③ O
여러 객체의 공통 특성을 뽑아 슈퍼클래스로 일반화하는 것은 추상화와 일반화에 해당한다. 이런 방식은 중복을 줄이고 재사용성을 높여 설계 비용과 시간을 줄일 수 있으므로 옳다.

④ O
캡슐화는 객체의 내부 구현을 외부에 직접 노출하지 않는 것이다. 내부 세부내용이 바뀌어도 외부와의 인터페이스가 유지되면 오류의 파급 범위를 줄일 수 있으므로 옳다.

오답 포인트

객체지향 설계도 모듈을 사용하므로 ②를 맞는 말처럼 보기 쉽다. 그러나 이 문항의 핵심은 분해 기준이다. 함수·작업 분해가 중심이면 구조적 설계이고, 객체·책임·상호작용이 중심이면 객체지향 설계이다.

다시 나오면 이렇게 판단

선지에 함수, 모듈, 작업 분할, 절차 분해가 나오면 먼저 구조적 설계 설명인지 의심하면 된다. 반대로 객체, 클래스, 속성, 연산, 캡슐화, 추상화가 나오면 객체지향 설계의 특징으로 판단할 수 있다.

암기 포인트

객체지향은 객체 중심, 구조적 설계는 기능 중심이다.

NoteBase

문 12. 소프트웨어 설계 시 고려해야 하는 결합도가 낮은 것부터 순서대로 바르게 나열한 것은?

① 스탬프 결합도 - 자료 결합도 - 제어 결합도 - 공통 결합도 - 내용 결합도

② 자료 결합도 - 스탬프 결합도 - 제어 결합도 - 내용 결합도 - 공통 결합도

③ 스탬프 결합도 - 자료 결합도 - 제어 결합도 - 내용 결합도 - 공통 결합도

④ 자료 결합도 - 스탬프 결합도 - 제어 결합도 - 공통 결합도 - 내용 결합도

정답 4번

출제 포인트

결합도의 개념과 약한 결합에서 강한 결합으로 이어지는 배열 기준을 묻는 문제이다.

올바른 순서

1. 자료 결합도: 필요한 데이터만 매개변수로 전달하는 결합이다.
2. 스탬프 결합도: 배열, 레코드, 구조체처럼 자료구조 전체를 전달하는 결합이다.
3. 제어 결합도: 제어 플래그나 분기 정보를 넘겨 다른 모듈의 처리 흐름에 영향을 주는 결합이다.
4. 공통 결합도: 전역 변수나 공용 데이터를 여러 모듈이 함께 사용하는 결합이다.
5. 내용 결합도: 한 모듈이 다른 모듈의 내부 데이터나 처리 내용을 직접 참조하거나 수정하는 가장 강한 결합이다.

왜 이 순서인가: 필요한 값만 주고받을수록 결합이 약하고, 공유 상태나 내부 구현에 깊게 의존할수록 결합이 강해진다.

선지별 해설

① X: 틀린 부분은 "스탬프 결합도 - 자료 결합도"이다. 자료 결합도는 필요한 값만 전달하므로 스탬프 결합도보다 결합이 약하다. 올바르게는 "자료 결합도 - 스탬프 결합도 - 제어 결합도 - 공통 결합도 - 내용 결합도"이다.

② X: 틀린 부분은 "내용 결합도 - 공통 결합도"이다. 내용 결합도는 다른 모듈 내부를 직접 건드리는 가장 강한 결합이므로 공통 결합도보다 뒤에 와야 한다. 올바르게는 "자료 결합도 - 스탬프 결합도 - 제어 결합도 - 공통 결합도 - 내용 결합도"이다.

③ X: 틀린 부분은 앞의 "스탬프 결합도 - 자료 결합도"와 뒤의 "내용 결합도 - 공통 결합도"이다. 자료 결합도와 스탬프 결합도의 순서, 공통 결합도와 내용 결합도의 순서가 모두 뒤바뀌었다. 올바르게는 "자료 결합도 - 스탬프 결합도 - 제어 결합도 - 공통 결합도 - 내용 결합도"이다.

④ O: 자료 결합도는 가장 약하고, 그다음이 스탬프 결합도, 제어 결합도, 공통 결합도, 내용 결합도 순이므로 낮은 것부터 나열한 순서가 맞다.

NoteBase

오답 포인트

자료 결합도와 스탬프 결합도는 모두 데이터를 주고받는다는 점 때문에 앞부분 순서를 바꾸기 쉽고, 공통 결합도와 내용 결합도는 모두 강한 결합이라 끝부분도 자주 혼동한다. 필요한 값만 전달하면 자료, 자료구조 전체를 넘기면 스탬프, 전역 데이터를 함께 쓰면 공통, 내부를 직접 건드리면 내용으로 구분된다.

다시 나오면 이렇게 판단

결합도 문제는 무엇을 넘기고 어디까지 의존하는지를 본다. 필요한 값만 전달하면 가장 약하고, 자료구조 전체 전달이면 그다음, 제어정보 전달이면 더 강하며, 전역공유와 내부직접참조는 가장 뒤에 둔다. 이 기준으로 자료-스탬프-제어-공통-내용 순서를 판단할 수 있다.

암기 포인트

자료 < 스탬프 < 제어 < 공통 < 내용 순서이며, 값만 전달할수록 약하고 내부를 건드릴수록 강하다.

NoteBase

문 13. 소프트웨어 개발 단계를 시간의 흐름에 따라 네 개의 범주(도입, 상세, 구축, 이행)로 나누고, 각 범주에는 요구사항 도출부터 설계, 구현, 평가까지의 개발 생명주기가 포함되어 있는 방법론은?

① XP(eXtreme Programming)

② UP(Unified Process)

③ CMM(Capability Maturity Model)

④ SPICE(Software Process Improvement and Capability dEtermination)

정답 2번

출제 포인트

이 문제는 UP의 4단계 생명주기와 XP, CMM, SPICE의 성격 차이를 구분하는 문제이다.

핵심 개념

UP는 도입, 상세, 구축, 이행의 4개 범주로 개발을 진행하는 반복적 개발 프로세스이다. 각 범주 안에서도 요구사항, 설계, 구현, 테스트 같은 활동이 반복된다.
XP는 짧은 반복과 고객 참여, 테스트 중심 개발을 강조하는 애자일 방법론이다.
CMM은 소프트웨어 조직의 프로세스 성숙도를 단계별로 평가하는 모델이다.
SPICE는 소프트웨어 프로세스의 능력과 개선 수준을 평가하는 국제 표준 기반 모델이다.

선지별 해설

① X: XP는 eXtreme Programming으로, 짧은 반복 주기와 페어 프로그래밍, 지속적 통합 등을 중시하는 애자일 방법론이다. 틀린 부분은 UP의 4단계 생명주기를 XP에 대응시킨 점이다. 올바르게는 XP가 아니라 UP가 도입-상세-구축-이행의 단계 구성을 가진다.
② O: UP는 Unified Process로, 시간을 따라 도입, 상세, 구축, 이행의 4개 범주로 나누고 각 범주 안에서 요구사항 도출, 설계, 구현, 평가 활동을 반복적으로 수행한다. 문제의 설명과 일치한다.
③ X: CMM은 Capability Maturity Model로, 개발 생명주기 자체를 설명하는 방법론이 아니라 조직의 소프트웨어 프로세스 성숙도를 평가하는 모델이다. 틀린 부분은 개발 단계 모델처럼 본 점이다. 올바르게는 CMM은 성숙도 평가 모델이다.
④ X: SPICE는 Software Process Improvement and Capability dEtermination으로, 프로세스 개선과 능력 평가를 위한 표준이다. 틀린 부분은 개발 생명주기 방법론으로 본 점이다. 올바르게는 SPICE는 프로세스 평가·개선 모델이다.

오답 포인트

UP, XP는 개발 방법론이고 CMM, SPICE는 프로세스 평가 모델이라는 축을 놓치면 헷갈리기 쉽다. 특히 ①처럼 XP를 반복 개발이라는 이유만으로 UP의 4단계 구조와 혼동하는 함정이 자주 나온다.

NoteBase

다시 나오면 이렇게 판단

선지에서 도입, 상세, 구축, 이행이라는 4단계가 보이면 먼저 UP를 떠올리면 된다. 반대로 성숙도, 능력 평가, 프로세스 개선이 핵심어로 나오면 CMM이나 SPICE 쪽으로 판단하면 된다.

암기 포인트

UP는 4단계 개발 프로세스, XP는 애자일 실천, CMM·SPICE는 프로세스 평가 모델이다.

NoteBase

문 14. 빈칸에 들어갈 내용을 순서대로 바르게 나열한 것은?

하나의 모듈을 테스트할 때 상위나 하위 모듈이 개발이 안 된 경우가 있다. 상위나 하위 모듈이 개발될 때까지 기다릴 수 없어 가상의 상위나 하위 모듈을 만들어 사용한다. 상위 모듈 역할을 하는 가상의 모듈을 ( )(이)라 하고 그 역할은 테스트할 모듈을 ( )하는 것이다. 반대로 하위 역할을 하는 모듈을 ( )(이)라 하고 인자를 통해 받은 값을 가지고 수행한 후 그 결과를 테스트할 모듈에 넘겨주는 역할을 한다.

① 스텁(stub) - 구현 - 드라이버(driver)
② 드라이버(driver) - 구현 - 스텁(stub)
③ 스텁(stub) - 호출 - 드라이버(driver)
④ 드라이버(driver) - 호출 - 스텁(stub)

정답 4번

출제 포인트

이 문제는 단위 테스트에서 상위 모듈 대역인 드라이버와 하위 모듈 대역인 스텁의 역할, 그리고 드라이버가 테스트 대상 모듈을 호출한다는 점을 구분하는 문제이다.

핵심 개념

드라이버(driver)는 아직 준비되지 않은 상위 모듈 대신 테스트 대상 모듈을 호출하는 가상 모듈이다.
스텁(stub)은 아직 준비되지 않은 하위 모듈 대신 미리 정한 값을 돌려주는 가상 모듈이다.
단위 테스트에서는 상위가 없으면 드라이버를 두고, 하위가 없으면 스텁을 둔다.

선지별 해설

1번 X: 첫 빈칸이 틀렸다. 상위 모듈 역할을 하는 가상 모듈은 스텁이 아니라 드라이버이다. 또 드라이버의 역할은 테스트할 모듈을 구현하는 것이 아니라 호출하는 것이다.
2번 X: 첫 번째와 세 번째 빈칸은 맞지만 두 번째 빈칸이 틀렸다. 드라이버는 테스트할 모듈을 구현하지 않고 호출한다. 따라서 '구현'이 아니라 '호출'이 들어가야 한다.
3번 X: 두 번째 빈칸은 맞지만 첫 번째와 세 번째 빈칸이 서로 바뀌었다. 상위 모듈 역할은 드라이버이고, 하위 모듈 역할은 스텁이다.
4번 O: 상위 모듈 역할을 하는 가상 모듈은 드라이버이고, 드라이버는 테스트할 모듈을 호출한다. 하위 모듈 역할을 하는 가상 모듈은 스텁이므로 제시한 순서가 맞다.

오답 포인트

드라이버와 스텁을 둘 다 가짜 모듈로만 기억하면 방향이 섞인다. 상위 대신 아래로 호출하는 쪽이 드라이버이고, 아래 대신 결과를 되돌려주는 쪽이 스텁이라는 방향을 놓치면 2번이나 3번을 고르기 쉽다.

NoteBase

다시 나오면 이렇게 판단

통합 관계의 방향을 먼저 본다. 테스트 대상 모듈을 누가 불러 주는지 보면 상위 대역인 드라이버를 찾을 수 있고, 테스트 대상이 누구의 결과를 받아 오는지 보면 하위 대역인 스텁을 찾을 수 있다.

암기 포인트

드라이버는 위에서 불러 주는 모듈이고, 스텁은 아래에서 값을 돌려주는 모듈이다.

NoteBase

문 15. 다음 코드 변환에 대한 리팩토링 기법에 맞는 최적의 작업은?

```
Void printOwing(double amount) {
  printBanner();

  System.out.println("name:"+_name);
  System.out.println("amount:"+_amount);
}
```



```
Void printOwing(double amount) {
  printBanner();
  printDetails(amount);
}

Void printDetails(double amount) {
  System.out.println("name:"+_name);
  System.out.println("amount:"+_amount);
 }
```

① 메소드 이동 작업이다.

② 메소드 추출 작업이다.

③ 인터페이스 추출 작업이다.

④ 템플릿 메소드 형성 작업이다.

정답 2번

출제 포인트

코드 블록을 별도 메소드로 분리하는 메소드 추출과, 다른 리팩토링 기법들의 목적 차이를 구별하는 문제이다.

핵심 개념

메소드 추출은 한 메소드 안의 일부 문장을 별도 메소드로 분리하고 원래 위치를 그 호출문으로 바꾸는 리팩토링이다.
메소드 이동은 기능 전체의 소속을 더 적절한 다른 클래스나 객체로 옮기는 리팩토링이다.
인터페이스 추출은 여러 구현이 공유할 연산 계약을 인터페이스로 뽑아내는 리팩토링이다.
템플릿 메소드 형성은 상위 클래스에 알고리즘의 골격을 두고 하위 클래스가 일부 단계를 재정의하게 하는 구조화 기법이다.

NoteBase

선지별 해설

① X: 틀린 부분은 "메소드 이동"이다. 제시된 변환은 같은 문맥의 출력 코드 일부를 `printDetails(amount)`라는 새 메소드로 분리한 것이다. 메소드 이동은 `printOwning`이나 관련 기능을 다른 클래스 쪽으로 옮길 때의 설명이므로 올바르게는 메소드 추출 작업이다.
② O: `printOwning` 내부의 세부 출력 문장을 별도 메소드 `printDetails(amount)`로 뽑아내고, 원래 자리를 그 호출문으로 대체했다. 이는 긴 메소드의 일부를 잘라 이름 붙인 새 메소드로 만드는 메소드 추출의 전형적인 형태이다.
③ X: 틀린 부분은 "인터페이스 추출"이다. 인터페이스 추출은 공통 동작의 선언을 인터페이스로 분리하는 작업이다. 제시된 코드에는 인터페이스 선언이나 다형성 계약 추출이 없으므로 올바르게는 메소드 추출 작업이다.
④ X: 틀린 부분은 "템플릿 메소드 형성"이다. 템플릿 메소드는 상속 구조에서 상위 클래스가 처리 순서의 골격을 정하고 하위 클래스가 세부 단계를 바꾸게 하는 방식이다. 제시된 변환에는 상속 구조나 단계 재정의가 없으므로 올바르게는 메소드 추출 작업이다.

오답 포인트

코드를 "분리했다"는 표면만 보고 메소드 이동이나 인터페이스 추출로 오인하기 쉽다. 이 문항은 분리된 대상이 다른 클래스도, 인터페이스도, 상속 골격도 아니라 기존 메소드 내부 문장 묶음이라는 점을 잡아내는지가 핵심 함정이다.

다시 나오면 이렇게 판단

기존 메소드 내부의 몇 줄을 새 이름의 메소드로 빼고 원래 코드를 그 호출문으로 바꾸면 메소드 추출로 판단한다. 다른 클래스 쪽으로 책임이 옮겨지면 메소드 이동, 공통 계약이 생기면 인터페이스 추출, 상속 골격이 생기면 템플릿 메소드 형성으로 구분한다.

암기 포인트

문장 묶음을 새 메소드로 빼면 메소드 추출, 기능 소속을 옮기면 메소드 이동이다.

NoteBase

문 16. McCall의 소프트웨어 품질 요소에 대한 설명으로 옳지 않은 것은?

① 사용성(Usability) - 사용자가 쉽게 이해하고 사용이 용이하며 흥미를 느끼는 정도

② 정확성(Correctness) - 기능적인 요구사항 만족과 고객 목적 이행 정도

③ 이식성(Portability) - 다른 시스템과 정보 교류의 연동 및 통합 정도

④ 신뢰성(Reliability) - 필요한 정확성으로 기능을 수행하는 정도

정답 3번

출제 포인트

이 문제는 McCall의 소프트웨어 품질 요소 중 사용성, 정확성, 이식성, 신뢰성, 상호운용성의 정의를 정확히 구분하는지를 묻는 문제이다.

핵심 개념

사용성은 사용자가 시스템을 배우고 이해하며 사용하기 쉬운 정도이다.
정확성은 소프트웨어가 요구기능을 충족하고 고객 목적을 제대로 달성하는 정도이다.
이식성은 소프트웨어를 다른 하드웨어나 운영환경으로 쉽게 옮겨 사용할 수 있는 정도이다.
상호운용성은 다른 시스템과 정보를 주고받으며 연동·통합할 수 있는 정도이다.
신뢰성은 요구된 기능을 필요한 정확성과 안정성으로 수행하는 정도이다.

선지별 해설

① O: 사용성은 이해와 학습이 쉽고 사용이 편리한 정도를 뜻하므로 설명이 맞다.
② O: 정확성은 기능적 요구사항 만족과 고객 목적 이행 정도를 뜻하므로 설명이 맞다.
③ X: 틀린 부분은 "이식성(Portability)"이다. 다른 시스템과 정보 교류의 연동 및 통합 정도는 이식성이 아니라 상호운용성(Interoperability)이다. 이식성은 다른 환경으로 옮겨도 쉽게 동작하는 성질이다.
④ O: 신뢰성은 필요한 정확성으로 의도한 기능을 수행하는 정도를 뜻하므로 설명이 맞다.

오답 포인트

이식성과 상호운용성을 모두 "다른 시스템과 관련된 성질"로만 외우면 ③에 걸리기 쉽다. 이식성은 환경 이동, 상호운용성은 시스템 간 연동이라는 차이로 구분된다.

다시 나오면 이렇게 판단

선지에 다른 시스템과의 정보 교환, 연동, 통합이 나오면 상호운용성을 먼저 떠올린다. 반대로 다른 운영체제나 하드웨어로 옮겨도 사용 가능한지를 묻으면 이식성으로 판단한다.

NoteBase

암기 포인트

이식성은 옮겨 가는 성질, 상호운용성은 서로 연결되는 성질이다.

NoteBase

문 17. 객체지향의 개념에서 다형성(Polymorphism)의 주요 설명으로 옳지 않은 것은?

① 속성과 관련된 오퍼레이션을 클래스 안에 묶어서 하나로 취급한다.

② 하나의 인터페이스에 메소드를 데이터 타입(Data Type) 및 파라미터 수를 변경하여 재정의가 가능하다.

③ 하나의 인터페이스를 일관성 있게 사용 중심에서 제공할 수 있다.

④ 오버로딩(Overloading), 오버라이딩(Overriding)을 이용한 재사용성을 높일 수 있다.

정답 1번

출제 포인트

이 문제는 객체지향의 핵심 개념 중 다형성과 캡슐화를 구분할 수 있는지를 묻는 문제이다.

핵심 개념

다형성은 하나의 인터페이스나 메시지에 대해 객체나 상황에 따라 서로 다르게 동작하게 하는 성질이다.
캡슐화는 데이터와 그 데이터를 다루는 연산을 하나로 묶어 외부에는 필요한 인터페이스만 노출하는 성질이다.
오버로딩은 같은 이름의 메서드를 매개변수형이나 개수를 달리해 여러 방식으로 정의하는 것이고, 오버라이딩은 상속 관계에서 상위 클래스 메서드를 하위 클래스가 재정의하는 것이다.

선지별 해설

① X
속성과 관련된 오퍼레이션을 클래스 안에 묶어서 하나로 취급한다는 설명은 다형성이 아니라 캡슐화의 설명이다.
틀린 표현은 "다형성의 주요 설명"이다.
올바르게는 "속성과 관련된 오퍼레이션을 클래스 안에 묶어서 하나로 취급하는 것은 캡슐화이다"가 된다.

② O
같은 이름의 메서드를 데이터 타입이나 파라미터 수를 달리해 여러 형태로 정의하는 것은 메서드 오버로딩으로, 다형성의 대표적 구현 형태 중 하나이다.

③ O
하나의 인터페이스를 중심으로 서로 다른 객체를 일관성 있게 사용할 수 있게 하는 것이 다형성의 핵심이다.
같은 메시지를 보내더라도 실제 동작은 객체별로 다를 수 있다.

④ O
오버로딩과 오버라이딩은 다형성을 구현하는 대표 수단이다.
같은 이름의 연산을 다양한 상황에 맞게 재사용할 수 있으므로 코드의 확장성과 재사용성을 높일 수 있다.

NoteBase

오답 포인트

객체지향의 기본 개념인 캡슐화와 다형성을 섞어 놓은 점이 함정이다. ①은 "속성과 연산을 묶는다"라는 표현 때문에 그럴듯해 보이지만, 이 문장은 다형성이 아니라 캡슐화의 정의에 가깝다.

다시 나오면 이렇게 판단

선지에 "묶어서 숨기고 하나로 관리한다"가 나오면 먼저 캡슐화를 떠올리고, "같은 인터페이스로 다르게 동작한다"가 나오면 다형성을 떠올리면 된다. 정의의 중심축이 결합과 은닉인지, 동일 인터페이스의 다양한 동작인지로 구분하면 된다.

암기 포인트

캡슐화는 묶고 숨기는 것이고, 다형성은 같은 이름으로 다르게 동작하는 것이다.

NoteBase

문 18. T. McCabe의 순환 복잡도(Cyclomatic Complexity)에 대한 설명으로 옳지 않은 것은?

① 사이클로메틱 수는 각 모듈에 대한 제어도(fan-out)를 이용하여 측정한다.

② 사이클로메틱 수는 코드 전체에서 독립적인 경로의 수를 선형적으로 측정한다.

③ 사이클로메틱 수는 그래프 이론을 기반으로 하여, 코드를 동등한 제어 흐름 그래프로 변환한 다음, 메트릭을 결정하기 위한 그래프의 속성을 이용하여 계산한다.

④ 원시코드의 구조적인 복잡성을 알아내는 측도이다.

정답 1번

출제 포인트

이 문제는 맥케이브의 순환 복잡도가 무엇을 기준으로 계산되며 무엇을 의미하는지 구별할 수 있는지를 묻는다.

핵심 개념

순환 복잡도는 제어 흐름 그래프의 독립 경로 수를 바탕으로 프로그램의 논리적 복잡성을 수치화한 척도이다. fan-out은 한 모듈이 호출하거나 제어를 넘기는 모듈 수를 뜻하는 개념으로, 순환 복잡도의 계산 기준이 아니다.

선지별 해설

1번(X): 틀린 부분은 "제어도(fan-out)를 이용하여 측정한다"는 표현이다. 순환 복잡도는 fan-out으로 재는 척도가 아니라 제어 흐름 그래프에서의 독립 경로 수나 $V(G) = E - N + 2$ 같은 그래프 속성으로 계산한다.
2번(O): 순환 복잡도는 코드 전체에서의 선형 독립 경로 수를 나타내므로 구조적 복잡성과 필요한 테스트 경로 수를 판단하는 기준이 된다.
3번(O): 순환 복잡도는 그래프 이론에 기반하며, 코드를 동등한 제어 흐름 그래프로 변환한 뒤 노드와 간선의 관계 같은 그래프 속성으로 계산한다.
4번(O): 순환 복잡도는 원시코드의 제어 구조가 얼마나 복잡한지를 보여 주는 대표적인 구조적 복잡도 척도이다.

오답 포인트

fan-out과 순환 복잡도를 같은 계열의 척도로 혼동하면 1번에 흔들린다. fan-out은 모듈 간 호출 확산 정도이고, 순환 복잡도는 분기에서 생기는 독립 경로 수라는 점이 핵심 차이이다.

다시 나오면 이렇게 판단

선지에 독립 경로, 제어 흐름 그래프, 테스트 경로 수가 나오면 순환 복잡도를 떠올리면 된다. 반대로 모듈 호출 수나 인터페이스 결합 정도가 나오면 fan-in, fan-out 계열인지 먼저 구분하면 된다.

암기 포인트

순환 복잡도는 독립 경로 수이고, fan-out은 모듈 호출의 퍼짐 정도이다.

NoteBase

문 19. 〈표 1〉과 같이 기능항목에 대한 복잡도가 주어지고, 입력은 10개, 출력은 5개, 질의는 8개, 파일은 28개, 응용인터페이스는 4개이며, 복잡도는 단순이다. 개발팀의 생산성은 주당 60 기능점수로 구현된다. 〈표 2〉를 참조하여 프로젝트를 수행하기 위한 노력을 추정한 값은?

<표 1>

	항목	복잡도		
		단순	보통	복잡
1	입력(트랜잭션)	3	4	6
2	출력(화면/양식)	4	5	7
3	질의	2	4	6
4	파일	5	10	15
5	응용 인터페이스	4	7	10

<표 2>

특성	처리복잡도	비고
데이터 통신	5	0: 영향 없음
분산데이터 처리	2	1: 약간의 영향
처리속도	0	2: 어느 정도 영향
트랜잭션 처리	2	3: 보통 정도의 영향
하드웨어 이용	3	4: 중요한 영향
온라인 데이터 입력	0	5: 심각한 영향
온라인 갱신 정도	0	
사용자 편의성	2	
계산의 복잡성	2	
재사용성	2	
설치의 용이함	2	
운영의 용이함	0	
이식성	3	
유지 보수성	2	

① 3.30 persons-week

② 3.31 persons-week

③ 3.32 persons-week

④ 3.33 persons-week

정답 4번

출제 포인트

이 문제는 기능점수법에서 미조정 기능점수와 보정계수를 구한 뒤 생산성으로 노력 값을 환산하는 문제이다.

조건 정리

주어진 값은 입력 10개, 출력 5개, 질의 8개, 파일 28개, 응용인터페이스 4개이며 모두 단순이다.
단순 가중치는 입력 3, 출력 4, 질의 2, 파일 5, 응용인터페이스 4이다.
표 2의 총처리복잡도 합은 25이고, 생산성은 1 persons-week당 60 기능점수이다.

NoteBase

사용할 공식

미조정 기능점수는 각 기능항목 수와 가중치의 곱의 합으로 계산한다.
보정계수는 $0.65 + 0.01 \times \sum F_i$이다.
노력은 보정 기능점수를 생산성으로 나눈 값이다.
이 문제는 기능점수법으로 규모를 보정한 뒤 생산성으로 effort를 환산하는 상황이므로 이 공식을 사용한다.

계산 과정

1. 미조정 기능점수는 $10 \times 3 + 5 \times 4 + 8 \times 2 + 28 \times 5 + 4 \times 4 = 222$이다.
2. 총처리복잡도는 $5 + 2 + 0 + 2 + 3 + 0 + 0 + 2 + 2 + 2 + 2 + 0 + 3 + 2 = 25$이므로 보정계수는 $0.65 + 0.01 \times 25 = 0.90$이다.
3. 보정 기능점수는 $222 \times 0.90 = 199.8$이다.
4. 노력은 $199.8 \div 60 = 3.33$ persons-week이다.

선지 판단

1번 X: 계산 결과는 3.33 persons-week이므로 3.30 persons-week는 맞지 않는다. 틀린 표현은 '3.30 persons-week'이며 올바르게는 '3.33 persons-week'이다. 이 값은 미조정 기능점수 합계나 보정계수를 작게 잡을 때 나오는 오답이다.
2번 X: 계산 결과와 0.02 persons-week 차이가 있으므로 틀리다. 틀린 표현은 '3.31 persons-week'이며 올바르게는 '3.33 persons-week'이다. 총처리복잡도 합계나 중간 반올림을 잘못하면 이런 값이 나온다.
3번 X: 계산 결과와 0.01 persons-week 차이가 있어도 정답값은 아니다. 틀린 표현은 '3.32 persons-week'이며 올바르게는 '3.33 persons-week'이다. 보정 기능점수 199.8을 정확히 60으로 나누어야 한다.
4번 O: 미조정 기능점수 222, 보정계수 0.90, 보정 기능점수 199.8이므로 노력은 3.33 persons-week가 된다.

오답 포인트

질의의 단순 가중치를 2가 아니라 다른 값으로 보거나, 표 2의 총처리복잡도 합을 25가 아닌 값으로 더하면 오답이 된다. 마지막 나눗셈에서 조기 반올림을 하면 3.31 또는 3.32처럼 비슷한 오답에 끌릴 수 있다.

다시 나오면 이렇게 판단

기능점수 계산형은 항목별 가중치로 미조정 기능점수를 먼저 구하고, 총처리복잡도 합으로 보정계수를 만든 뒤, 보정 기능점수를 생산성으로 나누는 순서로 판단한다.

암기 포인트

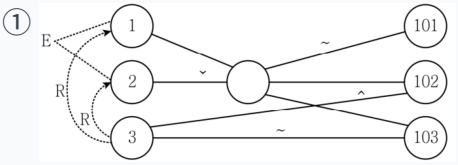
기능점수법은 미조정 기능점수 → 보정계수 → 보정 기능점수 → effort 순서로 계산한다.

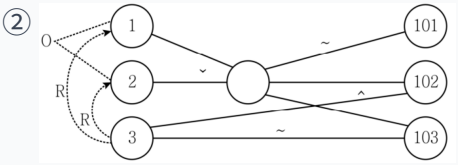
NoteBase

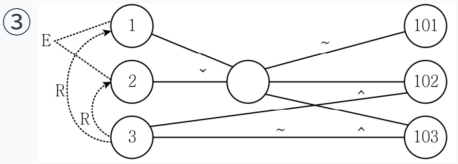
문 20. 원인-결과 그래프 기법으로 테스트 케이스를 구하고자 한다. 문제에 대한 원인과 결과가 다음과 같을 때, 제한조건을 포함한 그래프는?

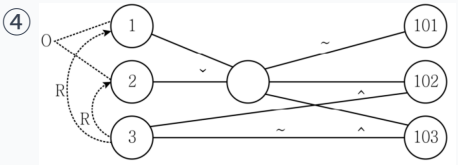
◯ 첫 번째 열이 A 또는 B로 시작하고, 두 번째 열이 #이면 '출입가능'을 출력한다.
◯ 첫 번째 열이 A 또는 B로 시작하지 않으면 '출입금지'로 출력한다.
◯ 첫 번째 열이 A 또는 B로 시작하고, 두 번째 열이 #이 아니면 '비밀번호 오류'를 출력한다.
◯ A는 부장, B는 사원을 의미한다.

원인	결과
1: 첫 번째 열이 A	101: '출입금지' 출력
2: 첫 번째 열이 B	102: '출입가능' 출력
3: 두 번째 열이 #	103: '비밀번호 오류' 출력

① 

② 

③ 

④ 

정답 3번

출제 포인트

이 문제는 원인-결과 그래프에서 배타 제약과 논리 결합을 함께 읽어 101=≠g(12), 102=(12)3, 103=(12)≠g3를 정확히 표현했는지를 판단하는 문제이다.

NoteBase

선지별 해설

① X. 1과 2를 E로 둔 점은 맞지만 103이 (12)≠g3로 완성되지 않았다. 틀린 표현은 103에 3의 부정 조건이 빠진 점이고, 올바르게는 첫 번째 열이 A 또는 B이면서 두 번째 열이 #이 아닐 때만 103으로 가야 한다.
② X. 1과 2 사이를 O로 둔 점이 틀리다. 첫 번째 열이 A도 B도 아닐 수 있으므로 1과 2는 정확히 하나가 반드시 참인 관계가 아니라 서로 동시에 참일 수 없는 배타 관계이다. 올바르게는 O가 아니라 E이어야 하며, 그래야 A 또는 B가 아닌 경우가 101로 연결된다.
③ O. 1과 2를 E로 두어 A와 B가 동시에 올 수 없음을 반영했고, 102를 (12)3, 103을 (12)≠g3, 101을 ≠g(12)로 표현했다. 제시된 조건과 제한조건을 모두 만족한다.
④ X. 103을 (12)≠g3로 표현한 점은 맞지만 1과 2를 O로 둔 점이 틀리다. 틀린 표현은 첫 번째 열이 반드시 A 또는 B라고 강제한 부분이고, 올바르게는 1과 2를 E로 두어 A 또는 B가 아닌 경우도 허용해야 한다.

오답 포인트

이 문항의 함정은 E와 O를 혼동하거나, 103에서 두 번째 열이 #이 아니라는 부정 조건을 빠뜨리는 데 있다. 1과 2는 동시에 참일 수 없지만 둘 다 거짓일 수는 있으므로 O가 아니라 E이다.

다시 나오면 이렇게 판단

먼저 원인들 사이의 제약이 배타인지 필수인지부터 구분한다. 그다음 첫 번째 열이 A 또는 B인지로 12를 만들고, 두 번째 열이 #이면 102, #이 아니면 103, A 또는 B가 아니면 101로 분기하면 된다.

암기 포인트

A/B 관계는 E이고, 결과 분기는 #→102, ≠g#→103, ≠g(A B)→101이다.

NoteBase