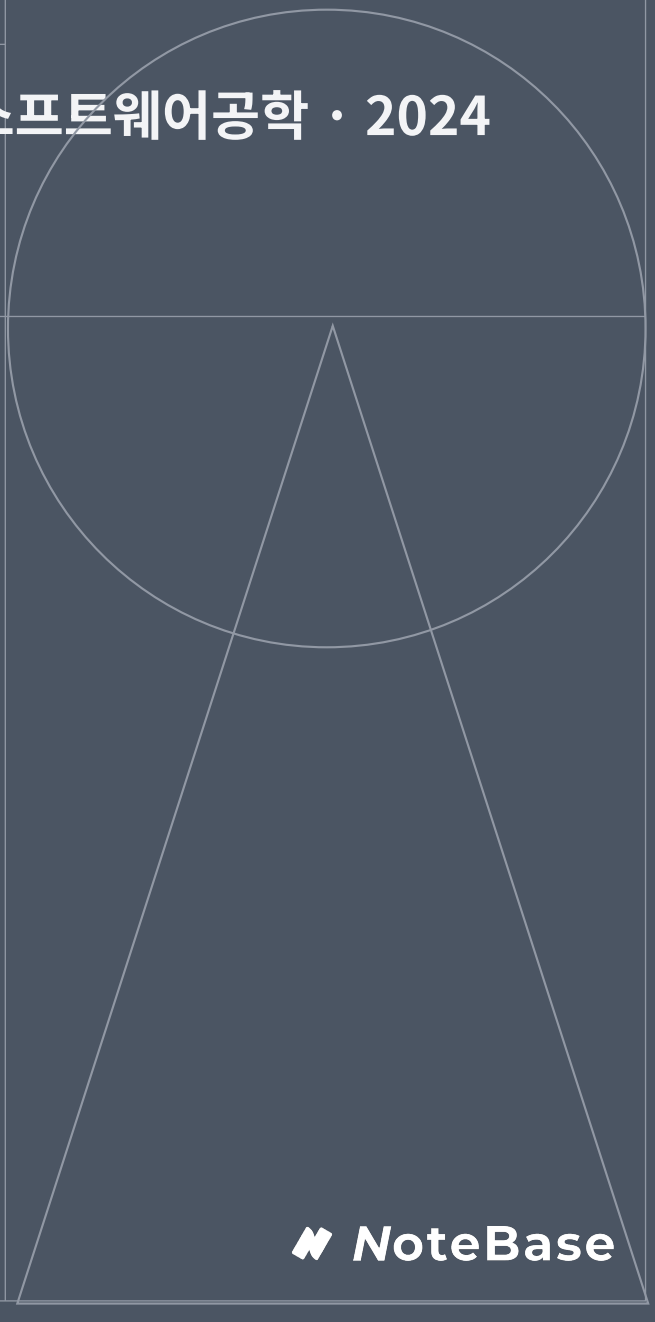
기출OX 해설집

공무원/경찰/공채(경위) · 소프트웨어공학 · 2024



NoteBase

기출OX 해설집

공무원/경찰/공채(경위) · 소프트웨어공학 · 2024

공무원/경찰/공채(경위) · 소프트웨어공학 · 2024

문 1. 프로젝트에 대한 고객 요구가 아래와 같을 때, 다음 보기 중 해당 프로젝트를 수행하기에 가장 적절한 소프트웨어 개발 프로세스 모델은?

가. 프로젝트 초반에 전체 시스템에 대한 모든 요구를 최초 계획단계에서 모두 확정하여 서브시스템으로 분할하고, 릴리즈 횟수와 시기를 정의해야 한다.
나. 이를 기반으로 전체 프로젝트 계획을 수립하여 '설계-구현/시험-설치/운영' 과정의 반복을 통해 각 서브시스템별로 단계적으로 릴리즈하는 형태로 진행하여, 전체 시스템을 확장하고 완성하여 최대한 신속히 작동하는 SW를 조기에 제공받고 싶다.
다. 이미 릴리즈된 서비스는 사용자로부터 제시된 새로운 아이디어나 의견을 받아 기 확정되어 있던 개발 범위에 추가 반영하여 개발하고, 함께 릴리즈함으로써 신규 시스템으로 인한 조직의 충격을 완화해야 한다.
라. 규모가 큰 개발 조직을 보유하고 있으니, 개발 기간 단축을 위해 분할된 서브시스템 간 병행 개발도 시도해 달라. 그러나, 대규모 프로젝트임을 고려하여 개발은 정확하게 정해진 표준 절차를 기반으로 수행해야 한다. 또한, 이전 릴리즈 된 SW와 개발해야 하는 서브시스템 간의 연계 및 통합에 대한 위험 부담, 디버깅 등을 통한 실시간 장애 관리 등을 고려할 때 우수한 프로젝트 관리자가 요구된다.

① Agile 프로세스 모델
② 점증적 프로세스 모델
③ 폭포수 프로세스 모델
④ Spiral 프로세스 모델

정답 2번

간단한 해설

전체 시스템을 서브시스템으로 나누고 **작동하는 기능을 단계적으로 릴리즈하여 누적·확장**하는 것이 점증적 모델의 핵심이다. 전체 요구와 일정을 미리 계획하더라도 부분별 개발·배포를 반복할 수 있다.

NoteBase

대응 관계 해설

점증적 프로세스 모델은 전체 시스템을 기능 단위로 나누고, 사용 가능한 부분을 차례로 개발·배포하여 기능을 누적시키는 모델이다. 지문의 요구들은 이 점증적 개발을 전체 계획과 표준 절차에 따라 수행하는 형태에 대응한다.

가. 전체 요구와 릴리즈 계획: 전체 요구를 먼저 파악해 서브시스템으로 분할하고, 각 부분을 언제 제공할지 정한다. 이는 **전체 범위를 계획하되 제품은 여러 차례 나누어 제공**하는 점증적 개발의 계획 방식에 해당한다. 요구를 초기에 확정한다는 단서만으로 전체 시스템을 한 번에 완성하는 폭포수 모델로 판단해서는 안 된다.

나. 단계적 릴리즈와 조기 제공: 각 서브시스템에 설계·구현·시험·설치 과정을 적용하고, 완성된 부분을 먼저 운영한다. 뒤의 릴리즈에서 기능을 더해 전체 시스템을 완성하므로 사용자는 전체 개발이 끝나기 전부터 작동하는 소프트웨어를 제공받을 수 있다.

다. 피드백과 점진적 도입: 먼저 배포한 부분을 사용하면서 얻은 의견을 변경 관리하여 후속 개발과 릴리즈에 반영한다. 기존 기능의 개선과 새 기능의 추가가 함께 이루어지고, 시스템을 한꺼번에 전환하지 않아 조직의 적응 부담도 줄일 수 있다. 사용자 의견을 수용한다는 특징만으로 애자일로 단정할 수는 없다.

라. 병행 개발과 통합 관리: 분할한 서브시스템은 의존 관계를 조정해 병행 개발할 수 있다. 다만 이미 운영 중인 기능과 새 기능의 인터페이스·통합·결함·릴리즈 일정을 함께 관리해야 하므로 프로젝트 관리 역량이 중요하다. 이 사례처럼 표준 절차를 따르는 대규모 조직에서도 점증적 개발을 적용할 수 있다.

따라서 모든 요구를 종합하면 ② 점증적 프로세스 모델이 가장 적절하다. 애자일은 짧은 주기의 협업과 변화 대응을 강조하지만, 이 사례의 결정적 단서는 사전 전체 계획과 기능의 단계적 누적 릴리즈이다. 폭포수 모델은 전체 개발 단계를 순차적으로 진행하는 방식이 중심이고, Spiral 모델은 반복 주기마다 위험 분석을 중심에 둔다는 점에서 지문의 핵심 구조와 다르다.

NoteBase

문 2. 클라이언트-서버 아키텍처의 구조와 개념에 대한 설명으로 적절하지 않은 것은?

① MOM(Message Oriented Middleware)은 일종의 소프트웨어 버스를 생성하여 레거시 시스템과 협력사의 외부 시스템을 통합할 수 있으며, 기본적으로 강력한 결합의 동기 기술로 메시지의 송신측과 수신측이 강하게 결합되어 있다.

② CORBA는 이기종 분산 환경에서 서로 다른 시스템 간에 상호작용을 무리없이 제공하기 위하여 ORB(Object Request Broker)에 광범위하게 사용되는 표준으로 OMG(Object Management Group)에서 개발되었다.

③ 애플리케이션 서버는 N-tier 아키텍처의 중간층에 위치하며 분산 통신, 보안, 트랜잭션 등을 제공하는 컴포넌트 기반의 서버 기술로, 클라이언트 레이어, 웹 레이어, 비즈니스 컴포넌트 레이어, 엔터프라이즈 정보 시스템 레이어로 구성된다.

④ 피어 투 피어(Peer to Peer) 아키텍처는 원칙적으로 클라이언트와 서버의 구분이 존재하지 않는 구조로, 네트워크 상의 개별 노드에서 계산이 실행되며 이를 기반으로 한 대표적인 사례로 블록체인 기술이 있다.

정답 1번

간단한 해설

①이 적절하지 않다. MOM의 기본 특징은 메시지를 매개로 한 **느슨한 결합과 비동기 통신**이다. 송신 측과 수신 측이 강하게 결합된 동기 기술이라는 설명은 이를 반대로 서술한 것이다.

NoteBase

선지별 해설

① (X) 메시지 지향 미들웨어(MOM)는 메시지 큐나 브로커를 매개로 시스템을 연결하며, 기본적으로 **비동기 통신과 느슨한 결합**을 지향한다. 송신 측은 수신 측의 즉각적인 처리를 기다리지 않고 메시지를 전달할 수 있다. 따라서 외부·레거시 시스템 통합에 활용된다는 부분은 맞지만 강한 결합의 동기 기술이라는 부분이 틀리다.

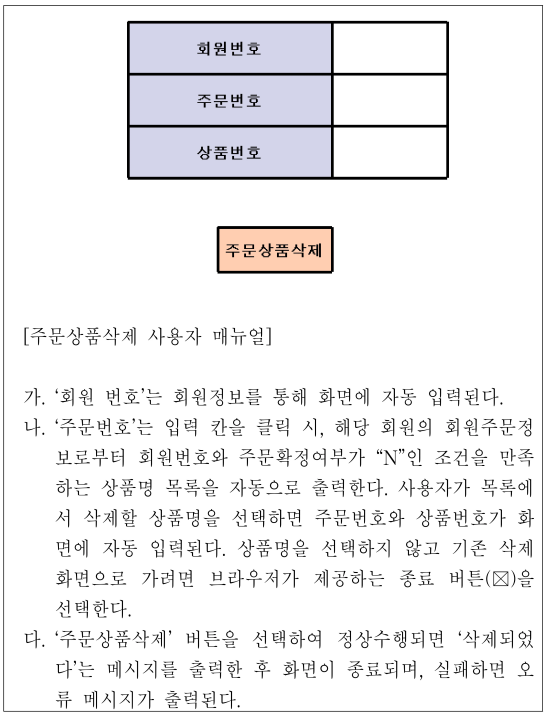
② (O) CORBA는 OMG가 정한 분산 객체 통신 표준이다. ORB는 객체 호출을 중개하여 서로 다른 운영체제나 구현 언어를 사용하는 시스템 사이에서도 상호작용할 수 있도록 한다. CORBA는 특정 언어나 단일 운영체제에만 종속된 기술이 아니다.

③ (O) 애플리케이션 서버는 다계층 구조의 중간층에서 비즈니스 컴포넌트를 실행하고 보안·트랜잭션·분산 통신 등의 공통 기능을 제공한다. 이를 이용하는 전체 시스템은 클라이언트, 웹, 비즈니스, 기업 정보 시스템(EIS) 계층으로 구분할 수 있다. 이 계층 구분은 전체 애플리케이션의 논리적 구조이며, 모든 계층이 애플리케이션 서버 한 대의 내부에 위치한다는 뜻은 아니다.

④ (O) P2P에서는 노드가 서비스의 요청자와 제공자 역할을 함께 수행할 수 있어 고정된 클라이언트·서버 구분을 두지 않는 것이 원칙이다. 각 노드가 작업과 정보를 분담하는 구조이며, 블록체인 네트워크는 이러한 P2P 방식으로 거래·블록 정보를 전파하는 대표적인 사례이다.

NoteBase

문 3. IFPUG(International Function Point Users Group)에서 정의한 기능 점수 산정 기준에 따라 업무 규모를 산정 중이다. 다음에 제시된 주문상품삭제 화면에서 사용자 매뉴얼에 따라 삭제를 수행할 때 DET(Data Element Type) 개수는? (단, 회원정보에서 회원번호를 관리한다. 주문은 회원만 가능하고, 회원이 주문 시 회원주문정보(주문번호, 주문일시, 상품번호, 상품명, 주문수량, 단가, 주문총액, 주문확정여부, 회원번호)에 저장된다. 그 외 모든 상황은 제외된다.)



① 6

② 7

③ 8

④ 9

정답 3번

간단한 해설

상품명 목록을 보여 주는 **조회 기능과 실제 삭제 기능을 구분**하여 DET를 센다. 목록 조회는 3개, 삭제는 5개이므로 합계는 8개이며, 삭제 성공·오류 메시지는 합쳐서 1개로 계산한다.

NoteBase

계산 과정

DET는 사용자가 식별할 수 있는 비반복적인 데이터 항목이다. 처리 기능의 DET는 애플리케이션 경계를 드나드는 데이터 항목에 기능 실행을 지시하는 제어 항목과 응답 메시지 제공 능력을 더하여 센다. 같은 기본프로세스 안의 반복 항목은 한 번만 세지만, 서로 다른 기본프로세스에서 쓰이는 항목은 각 프로세스에서 산정한다.

1. 산정할 기능 구분: 매뉴얼 나에서는 주문번호 입력 칸을 클릭하면 상품명 목록이 나오며, 사용자는 상품을 고르지 않고 조회를 끝낼 수도 있다. 따라서 이 목록 조회는 실제 삭제와 구분되는 **외부조회(EQ)**이다. 매뉴얼 다의 삭제는 회원주문정보를 변경하는 외부입력(EI)이다. 화면 하나에 두 기능이 있으므로 각각의 DET를 구한다.

2. 상품명 목록 조회: 조회 대상을 정하는 회원번호 1개, 출력되는 상품명 1개, 주문번호 입력 칸을 클릭하여 목록 조회를 실행하는 제어 항목 1개를 센다. 목록에 상품명이 여러 개 나와도 같은 데이터 유형이므로 상품명은 한 번만 센다. 조회의 DET는 1 + 1 + 1 = 3개이다. 주문확정여부 'N'은 내부 자료를 고르는 고정 조건이며, 사용자가 입력하거나 출력받는 항목이 아니므로 추가하지 않는다.

3. 주문상품 삭제: 삭제 화면의 회원번호·주문번호·상품번호가 각각 1개이므로 데이터 항목은 3개이다. **자동 입력되어도 화면에 표시되어 경계를 통과하는 항목**은 포함한다. 상품명은 별도 목록 조회에서 제시되는 항목이므로 삭제 기능에 다시 더하지 않는다. 여기에 '주문상품삭제' 실행 제어 1개와 처리 결과 응답 1개를 더하면 3 + 1 + 1 = 5개이다. '삭제되었다'는 성공 메시지와 오류 메시지는 같은 삭제 기능의 응답 능력이므로 합쳐서 1개로 센다.

4. 제외 항목과 합계: 브라우저가 제공하는 종료 버튼은 브라우저 자체의 기능이므로 애플리케이션의 업무 기능을 실행하는 제어 항목으로 세지 않는다. 회원주문정보에 저장된 주문일시·주문수량·단가·주문총액도 이 화면의 조회·삭제에서 경계를 드나드는 항목으로 제시되지 않았으므로 더하지 않는다. 회원번호는 서로 다른 두 기능에서 각각 산정하므로 전체 화면에서 한 번으로 합치지 않는다. 따라서 전체 DET는 조회 3개 + 삭제 5개 = 8개이며, ③의 값과 일치한다.

NoteBase

문 4. 다음 UML 2.0 다이어그램에 대한 설명으로 적절하지 않은 것은?

① 배치 다이어그램은 컴포넌트 간의 커넥션을 나타내는 것으로 연결된 컴포넌트의 구조를 나타낸다.

② 커뮤니케이션 다이어그램은 순차 다이어그램과 같은 내용을 표현하나 위임과 전달의 관계를 네트워크 형태로 더 명확하게 표현한다.

③ 인터랙션 오버뷰 다이어그램은 메시지 교환과 제어 흐름을 동시에 표현한다.

④ 패키지 다이어그램은 관련된 클래스를 패키지로 그룹화하여 표현함으로써 의존도를 낮추는데 도움이 된다.

정답 1번

간단한 해설

배치 다이어그램은 **실행환경의 노드와 소프트웨어의 배치 관계**를 나타낸다. 컴포넌트 간 연결과 구조를 중심으로 나타내는 것은 컴포넌트 다이어그램이다.

선지별 해설

① X: 배치 다이어그램은 서버·장치·실행환경 등의 **노드에 소프트웨어 아티팩트가 어떻게 배치되는지**와 노드 사이의 통신 경로를 나타낸다. 컴포넌트 간의 연결, 인터페이스와 의존관계 등 소프트웨어 구성요소의 구조는 컴포넌트 다이어그램의 주된 표현 대상이다. 두 다이어그램의 목적을 혼동한 ①이 적절하지 않다.

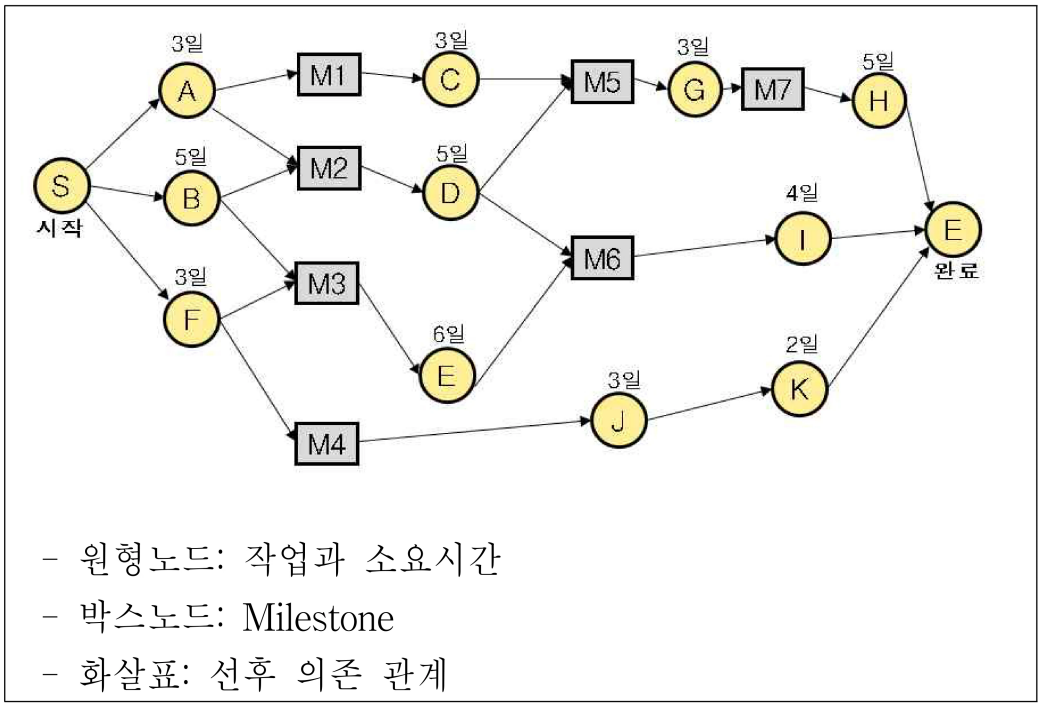
② O: 커뮤니케이션 다이어그램은 상호작용에 참여하는 객체와 그 사이의 링크를 네트워크 형태로 배치하고, 링크를 따라 전달되는 메시지를 표시한다. 순차 다이어그램과 같은 상호작용을 표현할 수 있지만 **누가 누구와 연결되어 메시지를 주고받는지**를 강조한다. 순차 다이어그램은 시간 순서를 강조하며, 커뮤니케이션 다이어그램에서는 메시지 번호 등으로 순서를 나타낸다.

③ O: 인터랙션 오버뷰 다이어그램은 개별 상호작용을 활동 다이어그램과 유사한 흐름 속에 배치한다. 메시지 교환을 다루는 상호작용을 포함하거나 참조하면서 분기·병합·병렬 실행 등 **상호작용 사이의 제어 흐름**을 함께 보여 준다. 따라서 개별 메시지 교환과 전체 진행 구조를 연결해 파악할 수 있다.

④ O: 패키지 다이어그램은 관련 클래스 등 모델 요소를 패키지로 묶고 패키지 사이의 의존관계를 표현한다. 이를 통해 기능별 경계와 불필요한 의존을 살펴 **패키지 간 결합을 줄이는 설계**에 도움을 줄 수 있다. 단순히 그림을 그리거나 클래스를 묶었다는 사실만으로 실제 의존성이 자동으로 사라지는 것은 아니다.

NoteBase

문 5. CPM 네트워크가 다음과 같을 때 프로젝트 소요기일은?



① 16일

② 8일

③ 18일

④ 20일

정답 3번

간단한 해설

정답은 ③ **18일**이다. 주공정 B → D → G → H의 소요기간은 5 + 5 + 3 + 5 = 18**일**이다. 병렬 작업을 전부 더하는 것이 아니라 완료까지 가장 오래 걸리는 경로를 기준으로 한다.

NoteBase

계산 과정

CPM에서는 선행 작업이 모두 끝나야 다음 작업을 시작할 수 있다. 따라서 합류 지점의 가장 빠른 도달 시각은 **선행 경로 완료 시각의 최댓값**이다. 박스 노드는 마일스톤이므로 별도의 작업기간을 더하지 않는다. 계산의 시작 시각은 0일로 둔다.

처음 작업 A, B, F는 각각 3일, 5일, 3일에 끝난다. 따라서 M1은 3일, M2는 max(3, 5) = 5일, M3는 max(5, 3) = 5일, M4는 3일에 도달한다.

C는 3 + 3 = 6일, D는 5 + 5 = 10일에 끝난다. 아래쪽의 6일 작업 E는 5 + 6 = 11일에 끝나고, J는 3 + 3 = 6일에 끝난다. 여기서 6일 작업 E와 그림 오른쪽의 완료 지점 E는 구별한다.

M5는 C와 D가 모두 끝난 max(6, 10) = 10일이므로 G는 10 + 3 = 13일, 이어서 H는 13 + 5 = 18일에 끝난다. M6는 D와 6일 작업 E가 모두 끝난 max(10, 11) = 11일이므로 I는 11 + 4 = 15일에 끝난다. K는 J 다음에 실행되어 6 + 2 = 8일에 끝난다.

최종 완료 지점은 H, I, K의 완료를 모두 기다리므로 프로젝트 소요기일은 max(18, 15, 8) = 18**일**이다. 주공정의 작업 순서는 B → D → G → H이며, 기간 합계도 5 + 5 + 3 + 5 = 18일이다. 따라서 정답은 ③이다.

NoteBase

문 6. 소프트웨어 구축 프로젝트가 8개월간 1억의 예산으로 착수되었다. 오늘은 4개월째 되는 마지막 날로 계획상으로 전체 공정의 60%를 완료했어야 했다. 그런데, 실제 작업 완료율은 52%이고, 지출된 비용은 5천만원이었다. 프로젝트에 대한 EVA(Earned Value Analysis) 결과 보고서의 내용으로 가장 적절하지 않은 것은?

① SV(Schedule Variance)는 -8,000,000원이다.

② CV(Cost Variance)는 2,000,000원이다.

③ SPI(Schedule Performance Index)는 0.86이다.

④ CPI(Cost Performance Index)는 1.2이다.

정답 4번

간단한 해설

비용성과지수 CPI는 **획득가치(EV)를 실제비용(AC)으로 나눈 값**이다. 실제 완료한 작업의 예산가치는 5,200만원이므로 CPI는 $\frac{5,200}{5,000} = 1.04$이며, 1.2가 아니다.

선지별 해설

총예산은 1억원이다. 계획가치 PV는 계획 공정률 60%에 해당하는 6,000만원, 획득가치 EV는 실제 완료율 52%에 해당하는 5,200만원, 실제비용 AC는 지출액인 5,000만원이다. 기간이 8개월 중 4개월 지났다고 계획 공정률을 50%로 바꾸지 않고, 문제에 제시된 **계획 공정률 60%**를 적용한다.

① O: 일정편차 SV = EV − PV = 5,200 − 6,000 = −800만원, 즉 −8,000,000원이다. 실제 완료한 작업의 예산가치가 계획보다 작으므로 일정이 지연되어 있다. 이 편차는 시간 단위가 아니라 금액 단위이다.

② O: 비용편차 CV = EV − AC = 5,200 − 5,000 = 200만원, 즉 2,000,000원이다. 완료한 작업의 예산가치보다 실제 투입비용이 적으므로 비용 측면에서는 유리한 상태이다.

③ O: 일정성과지수 $SPI = \frac{EV}{PV} = \frac{5,200}{6,000} = 0.8666\ldots$이다. 선지의 0.86은 **소수 둘째 자리까지 버린 표시**로 볼 수 있다. 소수 둘째 자리로 반올림하면 0.87이므로 0.86을 정확값이나 반올림값으로 이해해서는 안 된다. 지수가 1보다 작다는 점에서 계획보다 진도가 늦다.

④ X: 비용성과지수 $CPI = \frac{EV}{AC} = \frac{5,200}{5,000} = 1.04$이다. **1.2가 아니라 1.04**이며, 지출 1원당 예산가치 1.04원에 해당하는 작업을 수행했다는 뜻이다. $\frac{6,000}{5,000} = 1.2$는 PV를 AC로 나눈 값으로, CPI의 분자에 써야 할 EV를 잘못 사용한 결과이다.

NoteBase

문 7. 다음 중 GoF의 디자인 패턴에 속하지 않은 것은?

① 추상 팩토리(Abstract Factory)

② 단순 팩토리(Simple Factory)

③ 팩토리 메서드(Factory Method)

④ 템플릿 메서드(Template Method)

정답 2번

간단한 해설

정답은 ② **단순 팩토리**이다. 단순 팩토리는 객체 생성에 쓰이는 일반적인 기법이지만 GoF의 23개 패턴에는 포함되지 않는다. 추상 팩토리와 팩토리 메서드는 생성 패턴, 템플릿 메서드는 행위 패턴이다.

선지별 해설

① (O) 추상 팩토리는 서로 관련되거나 의존하는 객체들의 집합을 구체적인 클래스에 직접 의존하지 않고 생성하도록 하는 생성 패턴이다. 같은 계열의 객체를 일관되게 생성하는 데 초점을 두며 GoF의 23개 패턴에 포함된다.

② (X) 단순 팩토리는 입력값이나 조건에 따라 생성할 구체 클래스를 선택하는 별도의 생성 함수를 두는 방식이다. 널리 사용되는 객체 생성 관용구이지만 **GoF의 23개 디자인 패턴에는 독립된 항목으로 포함되지 않는다.** 이름이 비슷한 팩토리 메서드와 구별해야 한다.

③ (O) 팩토리 메서드는 객체 생성 인터페이스를 정의하되 실제로 어떤 구체 객체를 만들지는 하위 클래스가 결정하도록 하는 생성 패턴이다. 생성 결정을 하위 클래스에 위임한다는 점이 핵심이며 GoF 패턴에 포함된다.

④ (O) 템플릿 메서드는 상위 클래스에서 알고리즘의 전체 절차를 정하고 일부 단계를 하위 클래스가 구현하거나 재정의하도록 하는 행위 패턴이다. 알고리즘의 골격은 유지하면서 세부 동작을 바꾸며 GoF 패턴에 포함된다.

NoteBase

문 8. 요구분석(Requirement analysis) 단계를 올바른 순서로 나열한 것은?

① 요구 추출 - 도메인 분석 - 명세화 - 검토

② 도메인 분석 - 요구 추출 - 분석과 명세화 - 검토

③ 요구 추출 - 분석과 명세화 - 도메인 분석 - 검토

④ 도메인 분석 - 명세화 - 요구 추출 - 검토

정답 2번

간단한 해설

업무 분야를 이해한 뒤 요구를 수집해야 요구의 의미와 맥락을 제대로 파악할 수 있다. 수집한 요구를 분석해 명세로 만들고, 그 명세가 실제 필요와 일치하는지 검토한다.

올바른 순서

도메인 분석 → 요구 추출 → 분석과 명세화 → 검토

도메인 분석: 소프트웨어가 적용될 업무 분야의 용어, 업무 절차, 규칙, 이해관계자와 기존 시스템을 파악한다. **어떤 문제와 업무를 다루는지 이해**해야 적절한 질문을 만들고 사용자의 요구를 정확한 맥락에서 해석할 수 있다.

요구 추출: 도메인에 대한 이해를 바탕으로 인터뷰, 관찰, 문서 조사 등을 통해 사용자와 이해관계자가 필요로 하는 기능, 품질, 제약조건을 수집한다. 구체적인 요구를 확보해야 이후에 그 요구의 의미와 관계를 분석할 수 있다.

분석과 명세화: 추출한 요구의 중복·모순·누락과 실현 가능성을 분석하고, 필요한 조정과 우선순위 정리를 거쳐 명확한 문서나 모델로 표현한다. 요구를 수집하기 전에 명세부터 확정하는 것이 아니라, 확보한 요구를 개발과 검증의 기준으로 구체화하는 단계이다.

검토: 작성된 명세가 이해관계자의 의도를 제대로 반영하는지, 요구가 일관되고 완전하며 실현·검증 가능한지 확인한다. 명세가 있어야 그 내용을 체계적으로 검토할 수 있으며, 오류나 누락이 발견되면 앞 단계로 돌아가 보완할 수 있다. 따라서 이 순서는 기본적인 의존 관계를 나타내며 실제 활동의 반복을 금지하지 않는다.

NoteBase

문 9. 소프트웨어공학 지식 체계 (SWEBOK, SoftWare Engineering Body Of Knowledge)에서 정의된 S/W 관리 측면에 속하지 않은 것은?

① SW 형상관리

② SW 프로세스

③ SW 유지보수

④ SW 도구, 방법론

정답 3번

간단한 해설

SW 유지보수는 제공된 소프트웨어를 수정하고 개선하는 **제품 생명주기의 개발 활동**이다. 작업 전반의 절차·변경·지원 수단을 다루는 관리 측면과 구별한다.

선지별 해설

각 선지가 이 문항에서 구분하는 'S/W 관리 측면에 속한다'는 기준으로 판단한다.

① O: SW 형상관리는 코드·문서 등 관리 대상의 식별, 버전과 변경의 통제, 상태 기록 및 감사를 다룬다. 제품의 구성과 변경 이력을 일관되게 유지하는 **관리·통제 활동**이므로 이 문항의 관리 측면에 속한다.

② O: SW 프로세스는 개발과 유지보수 활동을 어떤 절차와 기준으로 수행할 것인지 다룬다. 프로세스의 정의·측정·평가·개선을 통해 **작업 수행 방식을 체계적으로 관리**하므로 관리 측면으로 분류한다.

③ X: SW 유지보수는 이미 제공된 소프트웨어의 결함을 수정하거나 변경된 환경에 적응시키고 성능·기능 등을 개선하는 활동이다. 이는 **제품 자체를 수정·발전시키는 생명주기 활동**으로서 이 문항의 개발 측면에 해당한다. 유지보수에도 계획·형상관리·품질관리가 필요하지만, 유지보수라는 지식영역 자체를 관리 측면으로 분류하는 것은 아니므로 ③이 요구한 답이다.

④ O: SW 도구와 방법론은 소프트웨어 활동을 체계적으로 수행하기 위한 기법과 지원 수단을 다룬다. 특정 제품의 한 기능을 직접 만드는 활동과 달리 여러 생명주기 활동에 공통으로 적용되는 **방법·도구의 선택과 활용 기반**이므로, 이 문항에서는 관리·지원 측면에 속한다.

NoteBase

문 10. 다음 중 형상변경 제어 단계에 수행되는 활동으로 적절하지 않은 것은?

① 형상 항목 변경 계획 수립

② 형상 항목 변경의 이유 파악

③ 형상 항목에 대한 검토(verification)와 확인(validation)

④ 형상 항목 변경 영향도 분석

정답 3번

간단한 해설

정답은 ③이다. 변경 계획 수립, 이유 파악, 영향도 분석은 변경제어에 해당한다. 형상 항목 자체의 검토와 확인은 주로 **형상감사·검증**의 성격이므로 구별한다.

선지별 해설

① (O) 변경제어는 변경 요청을 검토하고 승인된 변경을 질서 있게 반영하는 활동이다. 변경 대상, 범위, 일정, 담당자와 적용 절차 등을 정하는 형상 항목 변경 계획은 이 과정에 포함된다.

② (O) 변경의 필요성과 타당성을 판단하려면 왜 변경해야 하는지 파악해야 한다. 결함 수정, 요구사항 변경 등 변경 사유를 확인하는 것은 변경 요청의 평가와 승인 여부 판단을 위한 활동이다.

③ (X) 형상 항목이 요구사항과 기준선에 맞는지 검토하고 결과의 적합성을 확인하는 것은 주로 **형상감사·형상검증**의 성격에 해당한다. 이 문항은 변경의 필요성·영향·실행을 통제하는 기능과 산출물의 적합성을 검증하는 기능을 구별한다. 따라서 형상변경 제어 단계의 대표 활동으로는 가장 적절하지 않다. 다만 실제 절차에서 변경 반영 여부의 확인까지 변경제어와 연계될 수 있으므로, 변경제어에는 어떤 확인도 없다는 뜻은 아니다.

④ (O) 형상 항목 변경이 다른 모듈, 문서, 시험, 비용 및 일정에 미치는 영향을 분석해야 변경을 승인할지 판단할 수 있다. 따라서 **변경 영향도 분석**은 변경제어의 핵심 활동이다.

NoteBase

문 11. 다음 보기의 설명은 공연예약시스템을 위하여 제시된 업무 프로세스(가)에 대하여, 정보 은닉을 고려한 기초 설계 상태의 클래스 다이어그램 결과이다. 기초 설계가 가장 적절하지 않은 것은? (단, 회원정보관리, 공연티켓관리, 예약정보 클래스명은 각각 회원, 공연티켓, 예약정보로 하며, 클래스들에서 선언된 모든 속성과 메서드들의 이름은 나, 다와 같다. 그 외 모든 상황은 제외된다.)

가. 업무 프로세스	① 본 시스템을 통해 회원정보관리(등록, 수정, 삭제, 조회)와 공연티켓관리(등록, 수정, 삭제, 조회)를 수행한다. ② 회원은 예약정보를 통해 공연 티켓을 구매한다. 예약 후에 티켓을 구매할 수 있다. ③ 구매 전 예약정보 확인을 위해 회원ID와 티켓번호를 입력하여 조회를 수행한다.
나. 선언된 모든 속성명들	회원ID, 비밀번호, 성명, 스마트폰번호, 예약번호, 예약일자, 티켓번호, 공연장소, 공연회차, 좌석번호, 공연시작일시, 공연종료일시
다. 선언된 모든 메서드명들	회원등록, 회원수정, 회원삭제, 회원조회, 예약, 구매, 티켓등록, 티켓수정, 티켓삭제, 티켓조회

① 회원 클래스에 데이터 변수 총 4개(회원ID, 비밀번호, 성명, 스마트폰번호)가 Private(-) 유형으로, 메서드 총 4개(회원등록, 회원수정, 회원삭제, 회원조회)가 Public(+)으로 정의되었다.

② 예약정보 클래스에 데이터 변수 총 2개(예약번호, 예약일자)가 Private(-) 유형으로, 메서드 총 2개(예약, 구매)가 Public(+) 유형으로 정의되었다.

③ 공연티켓 클래스에 데이터 변수 총 6개(티켓번호, 공연장소, 공연회차, 좌석번호, 공연시작일시, 공연종료일시)가 Private(-) 유형으로, 메서드 총 4개(티켓등록, 티켓수정, 티켓삭제, 티켓조회)가 Public(+)으로 정의되었다.

④ 회원, 예약정보, 공연티켓 클래스의 관계는 아래와 같다.



정답 2번

간단한 해설

②가 가장 적절하지 않다. 회원ID와 티켓번호로 예약을 확인해야 하므로 예약정보는 **예약한 회원과 대상 티켓을 식별할 정보**도 반영해야 한다. 예약번호와 예약일자 두 속성만 제시한 설계는 이 요구를 충분히 담지 못한다.

NoteBase

선지별 해설

① (O) 회원ID·비밀번호·성명·스마트폰번호는 회원의 상태를 나타내는 속성이다. 이를 private(-)로 숨기고 회원등록·회원수정·회원삭제·회원조회 기능을 public(+) 메서드로 제공하면, 외부에서 속성을 직접 조작하지 않고 정해진 기능을 통해 회원정보를 관리하게 된다.

② (X) 업무 프로세스 가의 ③은 **회원ID와 티켓번호로 예약정보를 조회**하도록 요구한다. 따라서 예약정보에는 예약번호·예약일자뿐 아니라 어느 회원의 어느 티켓에 관한 예약인지를 식별할 정보가 연결되어야 한다. 제시된 기초 설계에서는 회원ID·티켓번호를 함께 반영해야 하므로 두 속성만으로 예약정보를 구성한 설명이 불충분하다. 문제는 private와 public의 구분이 아니라 예약과 회원·티켓을 식별하는 정보의 누락이다.

③ (O) 티켓번호·공연장소·공연회차·좌석번호·공연시작일시·공연종료일시는 공연티켓의 속성에 해당한다. 여섯 속성을 private(-)로 두고 티켓등록·티켓수정·티켓삭제·티켓조회 메서드를 public(+)으로 제공하는 것은 정보 은닉을 고려한 설계이다.

④ (O) 다중성은 연관선의 반대편 객체 하나에 대해 해당 끝의 객체가 몇 개 연결되는지를 나타낸다. 회원 쪽의 1과 예약정보 쪽의 0..*는 예약정보 하나가 회원 한 명에 속하고 회원 한 명은 예약정보를 없거나 여러 개 가질 수 있음을 뜻한다. 공연티켓 쪽의 1..*와 예약정보 쪽의 1은 예약정보 하나에 하나 이상의 티켓이 연결되고 각 티켓은 하나의 예약정보에 연결됨을 뜻한다. 화살표는 예약정보에서 회원과 공연티켓을 참조하는 방향을 나타낸다.

NoteBase

문 12. 시퀀스 다이어그램의 메시지 설계 방법에 대한 설명으로 적절하지 않은 것은?

① 객체생명선은 수직 방향의 점선으로 표시하며, 위에서 아래로의 시간 흐름을 의미한다.

② 객체명은 “객체명:클래스명” 형태로 정의하며, 클래스명만으로 정의할 수도 있다.

③ 재귀 메시지를 제외한 일반적인 메시지 호출은 수평 방향의 화살표로 표시하며, 화살표 위에는 송신 객체가 가지고 있는 함수명을 정의한다.

④ 수신 객체에게 요청한 함수가 완료될 때까지 기다리는 경우에는 동기 메시지로 호출하고, 수신 객체로부터 반환을 기다리지 않는 경우에는 비동기 메시지 호출로 설계한다.

정답 3번

간단한 해설

함수 호출 메시지에는 송신 객체의 함수가 아니라 **수신 객체에게 실행을 요청하는 함수**의 이름을 적는다. 송신 객체는 요청하는 쪽이고 수신 객체는 해당 연산을 수행하는 쪽이다.

선지별 해설

시퀀스 다이어그램은 상호작용에 참여하는 객체의 생명선과 객체 사이의 메시지를 시간 순서에 따라 표현한다. 함수 호출 메시지의 방향은 호출하는 객체에서 호출을 받는 객체로 향한다.

① O: 객체의 생명선은 일반적으로 객체 머리 부분에서 아래로 이어지는 수직 점선으로 표시한다. 위에서 아래로 사건의 시간 순서가 진행되며, 선의 길이가 실제 경과시간에 비례한다는 뜻은 아니다.

② O: 객체를 식별할 때 객체명:클래스명 형태를 사용한다. 객체명을 생략하고 클래스만 나타낼 수도 있으며, 예를 들어 :Order는 이름을 생략한 Order 클래스의 객체를 뜻한다. 클래스 자체와 그 클래스에 속한 객체를 혼동하지 않도록 콜론 표기에 유의한다.

③ X: 일반적인 객체 간 메시지는 송신 객체에서 수신 객체로 향하는 수평 화살표로 나타내지만, 함수 호출의 이름은 **수신 객체가 제공하는 연산**을 가리킨다. 예를 들어 주문 객체가 결제 객체에 pay()를 요청하면 pay()는 결제 객체가 수행할 연산이지, 요청을 보낸 주문 객체의 함수라는 뜻이 아니다. 자기 자신에게 보내는 메시지는 생명선으로 되돌아오는 형태로 표시한다.

④ O: 동기 호출에서는 송신 측이 호출한 연산의 완료와 제어의 반환을 기다린다. 비동기 호출에서는 완료 응답을 기다리며 송신 측의 실행을 멈추지 않고 다음 작업을 계속할 수 있다. 비동기 호출이라고 해서 수신 측의 처리가 즉시 끝나거나 이후 응답 메시지가 전혀 불가능하다는 뜻은 아니다.

NoteBase

문 13. 다음 프로그램에서 sender와 receiver 함수 간의 결합도와 receiver 함수의 응집도가 적절하게 나열된 것은?

```
void sender(void) {
    int num1, num2;
    int res;
   scanf("%d %d", &num1, &num2);
   res = receiver(num1, num2);
   printf("Result = %d\n", res);
}

int receiver(int a, int b) {
    if (a > b) {
      return (a * 3);
    } else {
      return (b * 2);
    }
}
```

① 제어 결합도, 기능적 응집도

② 자료 결합도, 기능적 응집도

③ 제어 결합도, 논리적 응집도

④ 자료 결합도, 논리적 응집도

정답 4번

간단한 해설

두 함수는 **계산에 쓰일 개별 정수 데이터**를 전달하므로 자료 결합도이다. receiver 내부는 비슷한 산술 처리 중 하나를 조건에 따라 선택하는 구조로 제시되어 논리적 응집도로 분류한다.

NoteBase

구성 요소별 해설

결합도는 모듈 사이의 의존 관계를, 응집도는 한 모듈 내부의 처리들이 묶인 방식을 나타낸다. 따라서 sender와 receiver 사이에 무엇이 전달되는지와 receiver 내부에서 어떤 처리를 수행하는지를 나누어 살펴야 한다.

sender와 receiver의 결합도: sender는 입력받은 두 정수 num1과 num2를 receiver의 매개변수 a와 b로 전달하고 결과를 돌려받는다. 전달되는 것은 계산에 쓰이는 **개별 데이터 값**이므로 자료 결합도이다. 호출자가 특정 동작을 지시하기 위한 별도의 제어 플래그나 처리 종류를 넘기는 제어 결합도와 구별된다. 전달된 데이터가 수신 함수의 조건식에 쓰인다는 이유만으로 제어 결합도가 되는 것은 아니다.

receiver의 응집도: 이 문항은 receiver 안에 묶인 두 산술 처리를 조건에 따라 선택하는 구조에 주목한다. a가 b보다 크면 a의 3배를 반환하고, 그렇지 않으면 b의 2배를 반환한다. 이처럼 논리적으로 비슷한 범주의 처리들을 한 모듈에 모아 두고 조건에 따라 하나를 선택하는 것으로 보아 논리적 응집도에 해당한다. 기능적 응집도는 모든 요소가 하나의 명확한 기능을 수행하기 위해 긴밀히 결합한 경우이다. 단순히 if문이 있다는 사실만으로 모든 함수의 응집도를 논리적이라고 분류하는 것은 아니다.

따라서 두 함수 간에는 자료 결합도, receiver 내부에는 논리적 응집도를 대응시키는 ④가 적절하다.

NoteBase

문 14. sort() 알고리즘을 상속받은 여러 순서정렬 알고리즘들이 다형성을 이용하여 확장될 수 있도록 설계함으로써 sort() 알고리즘을 수정하는 상황에서도 client 프로그램은 수정에 영향을 받지 않았다. 이것이 만족하고 있는 Martin의 객체지향 설계 원칙으로 가장 적절한 것은?

① 단일 책임의 원칙(Single Responsibility Principle)

② 인터페이스 분리의 원칙(Interface Segregation Principle)

③ 개방 폐쇄의 원칙(Open Closed Principle)

④ 리스코프 대체의 원칙(Liskov Substitution Principle)

정답 3번

간단한 해설

정답은 ③ **개방 폐쇄의 원칙**이다. 다형성으로 정렬 알고리즘은 **확장**할 수 있지만 기존 client 프로그램은 **수정하지 않아도 된다**는 점이 핵심이다.

개념 구분

개방 폐쇄의 원칙은 소프트웨어가 **확장에는 열려 있고 수정에는 닫혀 있어야 한다**는 원칙이다. 문제에서는 공통된 sort() 호출을 유지하면서 다형성으로 정렬 알고리즘을 확장한다. 정렬 구현이 달라져도 기존 client 프로그램을 고치지 않는다는 점이 핵심이므로 ③이다.

여기서 수정에 닫혀 있다는 말은 모든 코드를 영원히 수정하지 않는다는 뜻이 아니다. 예상한 종류의 기능 변경·확장을 새로운 구현으로 수용하여, 그 구현을 이용하는 안정된 기존 코드에 변경이 전파되지 않도록 설계한다는 뜻이다.

리스코프 대체의 원칙은 하위 유형의 객체를 상위 유형의 객체 대신 사용해도 프로그램이 기대하는 행동과 계약이 유지되어야 한다는 원칙이다. 다형성을 안전하게 사용하는 데 관련되지만, 이 문항은 대체 시 행동계약 위반 여부가 아니라 **기능 확장과 기존 코드 수정의 분리**를 강조한다.

단일 책임의 원칙은 한 모듈이 변경되어야 하는 이유를 하나의 책임에 집중시키는 것이고,

인터페이스 분리의 원칙은 클라이언트가 사용하지 않는 기능에 의존하지 않도록 인터페이스를 나누는 것이다. 제시된 정렬 알고리즘 확장 사례의 직접적인 초점은 이 둘이 아니다.

NoteBase

문 15. 어떤 모듈의 설계 품질을 측정한 결과 팬아웃(Fan-out) 지표의 값이 과도하게 커서 재사용이 어렵고, 다른 모듈의 수정에 영향을 크게 받고, 다른 모듈과의 상호작용으로 유지보수 복잡도가 높아질 수 있다는 진단을 받았다. 다음 중 진단된 문제를 해소하기 위하여 고려할 수 있는 가장 적절한 설계 패턴은?

① Facade 패턴

② Adapter 패턴

③ Bridge 패턴

④ Composite 패턴

정답 1번

간단한 해설

Facade 패턴은 복잡한 서브시스템에 단순한 통합 인터페이스를 제공한다. 여러 모듈을 직접 호출하던 의존을 Facade를 통한 접근으로 정리하여, 진단 대상 모듈의 팬아웃과 변경 영향을 줄일 수 있다.

핵심 풀이

문제의 진단: 팬아웃은 한 모듈이 호출하거나 의존하는 다른 모듈의 수를 나타내는 지표이다. 값이 과도하면 해당 모듈이 많은 외부 인터페이스와 상호작용을 알아야 하므로, 재사용 시 함께 고려할 대상이 늘어나고 외부 변경의 영향도 커질 수 있다. 이 사례의 핵심은 **여러 모듈에 대한 직접 의존을 줄이는 것**이다.

Facade의 적용: 복잡한 서브시스템 앞에 **단순하고 통합된 인터페이스**를 제공하여 클라이언트가 여러 내부 모듈을 일일이 호출하는 대신 Facade를 이용하게 한다. 내부 호출과 조정은 Facade 뒤로 숨기므로, 진단 대상 모듈이 알아야 할 외부 대상과 변경 접점이 줄어든다. 서브시스템의 복잡성이 없어지는 것이 아니라 접근 경계를 정리하는 것이므로, Facade 자체에 책임이 과도하게 몰리지 않도록 설계해야 한다.

다른 패턴과의 구별: Adapter는 서로 맞지 않는 인터페이스를 변환하여 연결하는 패턴이고,

Bridge는 추상화와 구현을 분리하여 각각 독립적으로 변경하도록 하는 패턴이다.

Composite는 부분과 전체를 트리 구조로 구성하여 같은 방식으로 다루게 한다.

이 사례에는 인터페이스 불일치, 두 계층의 독립적 확장, 부분-전체 구조보다 **여러 서브시스템 접근의 단순화**가 직접적인 해결 과제로 제시되어 있다.

따라서 과도한 팬아웃으로 생긴 직접 의존과 상호작용의 복잡성을 완화하기 위해 가장 적절하게 고려할 패턴은 ① Facade이다.

NoteBase

문 16. 제임스 럼바우의 객체지향 분석 모델링과 각 모델링에서 사용되는 다이어그램으로 적절한 것은?

① 객체 모델: 클래스, 동적 모델: 상태, 기능 모델: 데이터 흐름

② 객체 모델: 상태, 동적 모델: 데이터 흐름, 기능 모델: 클래스

③ 객체 모델: 데이터 흐름, 동적 모델: 클래스, 기능 모델: 상태

④ 객체 모델: 상태, 동적 모델: 클래스, 기능 모델: 데이터 흐름

정답 1번

간단한 해설

객체 모델은 **정적인 클래스 구조**, 동적 모델은 **사건에 따른 상태 변화**, 기능 모델은 **데이터의 처리와 흐름**을 나타낸다. 각각 클래스 다이어그램, 상태 다이어그램, 데이터 흐름 다이어그램을 연결한다.

대응 관계 해설

럼바우의 객체지향 분석은 시스템을 구조, 시간에 따른 행위, 데이터 변환이라는 세 관점으로 나누어 표현한다. 어떤 관점을 나타내는 모델인지에 따라 다이어그램을 연결한다.

객체 모델 → 클래스 다이어그램: 객체의 속성과 연산, 객체를 묶는 클래스 및 클래스 사이의 관계를 표현한다. 시스템을 이루는 요소와 그 연결이라는 **정적인 구조**에 초점을 둔다.

동적 모델 → 상태 다이어그램: 사건이 발생할 때 객체가 어느 상태에서 어느 상태로 바뀌는지를 표현한다. 단순한 데이터 이동이 아니라 **사건에 따른 상태 전이**를 다룬다.

기능 모델 → 데이터 흐름 다이어그램: 입력 데이터가 처리 과정을 거쳐 출력으로 변환되는 모습을 데이터 흐름, 처리, 저장소 등으로 표현한다. 클래스의 구조나 상태 변화의 시점보다 **입력에서 출력으로의 변환**에 초점을 둔다.

따라서 객체 모델은 클래스, 동적 모델은 상태, 기능 모델은 데이터 흐름으로 연결한 ①이 적절하다.

NoteBase

문 17. 객체지향 설계 메트릭(Metric)이 아닌 것은?

① 클래스당 가중 메서드(Weighted method per class)

② 싸이클로매틱 복잡도(Cyclomatic complexity)

③ 메서드 응집 결핍(Lack of cohesion in methods)

④ 상속 트리의 깊이(Depth of inheritance tree)

정답 2번

간단한 해설

정답은 ② **싸이클로매틱 복잡도**이다. 이는 제어 흐름의 독립 경로 수를 나타내는 일반적인 프로그램 복잡도 지표이다. WMC·LCOM·DIT는 클래스의 메서드, 응집도, 상속 구조를 평가하는 대표적인 객체지향 설계 메트릭이다.

선지별 해설

① (O) 클래스당 가중 메서드(WMC)는 클래스에 정의된 각 메서드의 복잡도 가중치를 합산한 값이다. 모든 메서드의 가중치를 1로 두면 메서드 수가 된다. 클래스의 구현 부담과 복잡도를 평가하는 객체지향 설계 메트릭이다.

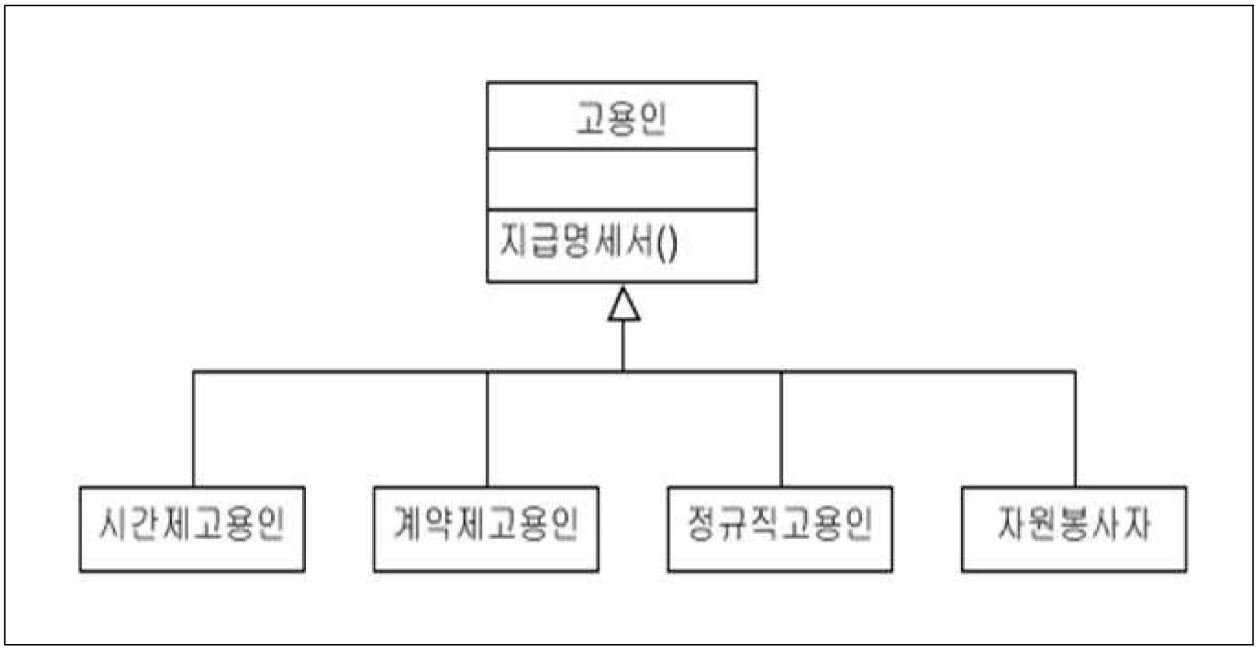
② (X) 싸이클로매틱 복잡도는 프로그램의 **제어 흐름에서 독립적인 경로의 수**를 측정하는 지표이다. 분기 구조의 복잡도와 필요한 테스트 경로를 평가하는 일반적인 프로그램 메트릭으로, 클래스·상속·응집도에 초점을 둔 객체지향 고유 설계 메트릭과 구별된다. 객체지향 프로그램의 개별 메서드에도 적용할 수 있지만, 그 사실이 이 지표를 객체지향 전용 메트릭으로 만들지는 않는다.

③ (O) 메서드 응집 결핍(LCOM)은 클래스의 메서드들이 공통 속성을 함께 사용하는 정도 등을 통해 응집도의 부족을 평가한다. 대표적인 정의에서 값이 클수록 메서드 간 관련성이 낮아 클래스 분리 필요성을 검토할 수 있다. 클래스 내부의 설계 품질을 다루는 객체지향 메트릭이다.

④ (O) 상속 트리의 깊이(DIT)는 클래스에서 상속 계층의 루트까지 이르는 경로의 깊이를 측정한다. 깊이가 커지면 상속받는 동작과 관계를 더 많이 고려해야 하므로 재사용 정도와 이해·변경의 복잡성을 살펴보는 데 활용된다. 상속이라는 객체지향 특성을 직접 반영하는 설계 메트릭이다.

NoteBase

문 18. 아래 클래스 다이어그램이 위반하고 있는 SOLID 원칙은?



① 단일 책임의 원칙(Single Responsibility Principle)

② 개방 폐쇄의 원칙(Open Closed Principle)

③ 리스코프 대체의 원칙(Liskov Substitution Principle)

④ 의존성 역전의 원칙(Dependency Inversion Principle)

정답 3번

간단한 해설

지급명세서()를 제공하는 고용인의 하위 타입에 무급 자원봉사자를 두면 **상위 타입의 급여 지급 관련 계약을 그대로 만족하기 어렵다**. 하위 타입으로 교체해도 상위 타입의 기대가 유지되어야 한다는 리스코프 대체 원칙의 문제이다.

NoteBase

도식 해설

다이어그램의 빈 삼각형 화살표는 일반화, 즉 상속 관계를 나타내며 상위 클래스인 '고용인'을 가리킨다. '고용인'에는 지급명세서()가 있고, 시간제고용인·계약제고용인·정규직고용인과 함께 자원봉사자도 그 하위 클래스로 연결되어 있다.

리스코프 대체 원칙은 **상위 타입이 사용되는 자리에 하위 타입을 넣어도 기대한 동작과 계약이 유지**되어야 한다는 원칙이다. 이 문항의 급여 지급 맥락에서 세 고용 유형은 지급명세서를 제공하는 역할과 부합하지만, 무급 자원봉사자는 같은 급여 지급 계약을 자연스럽게 만족하지 못한다. 자원봉사자를 대신 넣었을 때 지급명세서 기능을 거부하거나 의미 없는 동작으로 처리해야 한다면 상위 타입에 대한 사용자의 기대가 깨진다. 단순히 이름상 모두 일하는 사람이라는 공통점만으로 이 상속이 타당해지는 것은 아니다.

개선하려면 공통적인 인적 정보나 활동 역할과 급여 지급 기능을 분리하여, 급여를 받는 고용인만 지급명세서 기능을 제공하도록 계층이나 인터페이스를 구성할 수 있다. 다만 도식에는 메서드 구현이 제시되지 않았으므로 특정 예외 발생이나 반환값이 실제 구현되어 있다고 단정하지 않는다.

따라서 이 도식이 직접 드러내는 문제는 ③ 리스코프 대체 원칙 위반이다. 단일 책임 원칙은 변경 책임의 분리, 개방 폐쇄 원칙은 기존 코드 수정 없이 확장하는 구조, 의존성 역전 원칙은 구체 구현보다 추상에 의존하는 구조에 관한 것으로, 여기서 핵심인 하위 타입의 대체 가능성과 구별된다.

NoteBase

문 19. 아래 소스 코드에 대하여 문장(Statement) 커버리지, 조건(Condition) 커버리지, 결정(Decision) 커버리지, 조건-결정(Condition-Decision) 커버리지를 평가하고자 한다. 제시된 테스트 데이터를 기준으로 평가를 수행했을 때, 다음 보기의 설명이 적절하지 않은 것은? (단, Short-circuit evaluation은 수행하지 않는다.)

소스 코드	if (x >= -3 && y < 5)
	  x=x+y;
	else
	  x=x-y;

테스트 데이터	T = { (-5, -2), (1, 7) } 단, (a, b)는 (x값, y값)임

① 문장 커버리지는 66% 임

② 조건 커버리지는 50% 임

③ 결정 커버리지는 50% 임

④ 조건-결정 커버리지는 50% 임

정답 2번

간단한 해설

두 테스트에서 x ≥ −3은 거짓·참, y가 5보다 작다는 조건은 참·거짓으로 평가되므로 **각 기본조건의 참과 거짓이 모두 실행**된다. 따라서 조건 커버리지는 50%가 아니라 100%이며, 전체 AND 결정이 두 번 모두 거짓인 것과 구별해야 한다.

NoteBase

선지별 해설

기본조건을 A: $x \geq -3$, B: y가 5보다 작다는 조건이라 두면 전체 결정은 A AND B이다. 입력 (−5, −2)에서는 A가 거짓이고 B가 참이며, 입력 (1, 7)에서는 A가 참이고 B가 거짓이다. 두 경우 모두 전체 결정은 거짓이어서 x = x − y가 실행된다. **단락 평가를 하지 않으므로** 첫 번째 입력에서도 B의 참 결과가 실제 평가된다.

① O: 이 문항은 if의 조건 평가와 두 대입문을 문장 커버리지의 세 대상으로 센다. 두 테스트에서 조건 평가와 x = x − y는 실행되지만 x = x + y는 한 번도 실행되지 않는다. 따라서 $\frac{2}{3} \times 100 = 66.666\ldots\%$로, 선지의 66%는 소수 부분을 생략한 표시이다.

② X: 조건 커버리지는 전체 논리식이 아니라 **각 기본조건의 참·거짓**이 실행되었는지를 본다. A는 거짓과 참, B는 참과 거짓을 각각 한 번씩 경험하므로 네 조건 결과가 모두 충족된다. 조건 커버리지는 $\frac{4}{4} \times 100 = 100\%$이다.

③ O: 결정 커버리지는 전체 결정식의 참·거짓 결과를 기준으로 한다. A AND B는 두 테스트 모두 거짓이고 참인 경우는 없으므로, 두 결정 결과 중 하나만 실행되어 $\frac{1}{2} \times 100 = 50\%$이다.

④ O: 조건-결정 커버리지는 조건 커버리지와 결정 커버리지의 요구를 함께 충족해야 한다. 이 테스트는 기본조건의 참·거짓은 모두 충족하지만 전체 결정의 참 결과는 실행하지 못한다. 공식 채점에서는 이 선지의 50%를 인정한다. 이 수치는 **조건·결정 달성률 중 낮은 값을 적용한다는 전제**로 설명할 수 있다. 그 기준에서는 조건 100%와 결정 50% 중 낮은 값인 50%가 된다. 다만 이 산정 규칙은 문제에 명시되어 있지 않으며, 두 커버리지를 함께 충족해야 한다는 정의만으로 50%가 자동 도출되는 것은 아니다. 따라서 낮은 값을 취하는 방식을 조건-결정 커버리지의 보편적인 산정 공식으로 일반화해서는 안 된다.

NoteBase

문 20. 다음 중 리팩토링을 하는 이유로 적절하지 않은 것은?

① 소프트웨어 설계를 개선하려고

② 소프트웨어 이해를 쉽게 하려고

③ 추가 기능 구현 속도를 빠르게 하려고

④ 소프트웨어 인증을 받으려고

정답 4번

간단한 해설

정답은 ④이다. 리팩토링은 **외부 동작을 유지하면서 내부 구조를 개선**하여 설계와 이해도를 높이고 이후 기능 추가를 쉽게 하는 작업이다. 인증 취득은 별도의 적합성 평가 활동이다.

선지별 해설

① (O) 리팩토링은 **외부에서 관찰되는 동작을 유지하면서 내부 구조를 개선**하는 작업이다. 중복을 제거하고 책임을 적절히 분리하는 등 소프트웨어 설계를 개선하기 위해 수행한다.

② (O) 의미 있는 이름을 사용하고 복잡한 함수를 나누는 등의 구조 개선은 코드의 의도를 드러낸다. 따라서 개발자가 소프트웨어를 더 쉽게 이해하도록 하는 것은 리팩토링의 주요 목적이다.

③ (O) 구조가 명확하고 변경 지점이 잘 분리된 코드는 새로운 기능을 추가하기 쉽다. 리팩토링 자체가 기능을 추가하는 것은 아니지만, **후속 기능 구현과 변경에 드는 시간·노력을 줄이기 위한 준비**로 수행할 수 있다.

④ (X) 소프트웨어 인증은 정해진 기준에 대한 적합성을 평가받는 별도 활동이다. 인증 준비 중 코드 개선이 필요할 수는 있어도, **인증 취득 자체가 리팩토링의 고유한 목적은 아니다**. 리팩토링의 직접적인 목적은 내부 설계·이해도·변경 용이성의 개선이다.

NoteBase

문 21. 유지보수 및 운영 중인 소프트웨어의 MTBF(Mean Time Between Failures)가 3,000시간이고, MTTR(Mean Time To Repair)이 120시간으로 측정되었다. 해당 시스템의 가용성은? (단, 소수점 이하에서 반올림한다.)

① 92%

② 94%

③ 96%

④ 98%

정답 3번

간단한 해설

가용성은 **정상 운영시간을 운영시간과 복구시간의 합으로 나눈 비율**이다. $\frac{3,000}{3,000+120} \times 100 \approx 96.15\%$이므로 반올림한 가용성은 96%이다.

계산 과정

적용할 관계: 가용성은 운영과 복구를 반복하는 시스템에서 **전체 시간 중 정상적으로 사용할 수 있는 시간의 비율**이다. MTBF를 고장 사이의 평균 정상 운영시간, MTTR을 고장 후 평균 수리·복구시간으로 두면, 한 주기의 평균 전체 시간은 두 시간의 합이다. 따라서 가용성은 $A = \frac{MTBF}{MTBF + MTTR}$로 계산한다.

수치 대입: 두 값의 단위가 모두 시간이므로 별도 단위 변환은 필요 없다. 한 주기의 평균 전체 시간은 3,000 + 120 = 3,120시간이며, 정상 운영시간인 3,000시간을 분자로 둔다.
$A = \frac{3,000}{3,000+120} = \frac{3,000}{3,120} \approx 0.961538$

백분율과 반올림: 비율에 100을 곱하여 백분율로 바꾸면 $A \times 100 \approx 96.1538\%$이다. 문제의 지시대로 백분율의 소수점 이하를 반올림하면 **96%**이므로 ③과 일치한다.

NoteBase

문 22. 소프트웨어 품질 성숙도 수준 평가 체계인 CMMi와 SPICE(ISO/IEC 15504)에 대한 비교설명으로 적절하지 않은 것은?

① 일반적인 품질 성숙도 수준은 CMMi는 5단계, SPICE는 6단계로 구분한다.

② 성숙도 평가 시 다양한 영역으로 나누어 측정한 후 SPICE는 하나의 레벨로 조직의 성숙도 등급을 도출하지만, CMMi는 프로세스 영역별로 조직의 역량을 구분하여 성숙도 등급을 도출한다.

③ CMMi는 현재의 소프트웨어 개발 프로세스 개선을 통해 조직의 품질 성숙도를 향상하고자 하는 목적이 강한 반면, SPICE는 소프트웨어 조직이 제품을 생산하는 역량을 스스로 평가하거나 계약 시 사업자 평가를 위한 목적으로 많이 활용된다.

④ 조직 차원에서 프로세스가 최적화되어 프로젝트 목표를 만족하며, 지속적으로 프로세스 개선이 시도되고 성과가 관리된다면 CMMi와 SPICE 모두 품질 성숙도는 5등급으로 평가될 수 있다.

정답 2번

간단한 해설

②가 적절하지 않다. 대표적인 비교에서 CMMI의 단계적 표현은 **조직의 성숙도 수준**을, SPICE는 **프로세스별 능력 수준**을 평가한다. ②는 이 관계를 반대로 설명하였다.

선지별 해설

① (O) 이 문항에서 비교하는 전통적인 CMMI 단계적 표현은 성숙도를 1~5의 다섯 수준으로 구분한다. ISO/IEC 15504의 SPICE는 프로세스 능력을 0~5의 여섯 수준으로 평가한다. 시작 번호가 달라 단계 수는 다르지만 두 체계의 최고 수준 번호는 5이다.

② (X) **두 체계의 대표적인 평가 방식을 서로 바꾸어 서술하였다.** CMMI의 단계적 표현은 프로세스 영역들의 달성 상태를 종합하여 조직의 성숙도 수준을 제시한다. 반면 SPICE는 개별 프로세스별 능력 수준을 평가하여 프로파일을 만든다. 다만 CMMI에도 프로세스 영역별 능력을 다루는 연속적 표현이 있으므로, CMMI는 어떤 경우에도 개별 영역을 평가하지 않는다고 이해해서는 안 된다.

③ (O) CMMI는 프로세스 개선을 체계적으로 추진하여 조직의 성숙도를 높이는 데 활용된다. SPICE는 프로세스 개선뿐 아니라 자체 능력 진단과 공급자 선정 등에 필요한 프로세스 능력 판단에도 활용된다. 이는 두 체계의 대표적인 활용 방향을 비교한 것이며, 각각의 용도가 서로 완전히 배타적이라는 뜻은 아니다.

④ (O) 지속적 개선과 최적화는 두 체계 모두 최고 수준인 5와 연결된다. CMMI의 5수준은 최적화 단계이며, ISO/IEC 15504의 능력 5수준도 지속적인 프로세스 개선을 지향한다. 다만 실제 평가는 하위 수준을 포함한 요구사항의 충족 여부에 따라 이루어지므로, 개선 활동이 있다는 사실 하나만으로 자동으로 5수준이 되는 것은 아니다.

NoteBase

문 23. 다음에서 설명하는 객체지향 개념으로 적절한 것은?

가. 특정 항목에 대한 정보를 외부에 감출 수 있고, 필요에 따라 보이게 할 수도 있는 정보은닉을 실현하는 장치이다.
나. 사용자에게 해당 객체의 기능(서비스)과 사용법만 제공해 사용하기 쉽게 하고 내부는 함부로 변경할 수 없게 감추는 개념이다.
다. 소프트웨어 내부인 객체 내부에 서로 관련된 데이터와 그 데이터를 조작할 수 있는 메서드를 같이 포장하는 방법으로 안에 포함된 메서드만을 사용해 데이터 값을 변경할 수 있게 한다.

① 모듈화
② 상속성
③ 캡슐화
④ 다형성

정답 3번

간단한 해설

정답은 ③ **캡슐화**이다. 가·나·다는 모두 **데이터와 메서드를 묶고, 내부 구현을 감추며, 공개된 기능을 통해 접근하도록 통제**하는 특성을 설명한다.

구성 요소별 해설

가. 필요한 정보만 외부에 공개하고 내부 구현을 감추는 정보은닉을 실현한다는 설명이다. 캡슐화는 공개 범위를 통제하는 경계를 만들어 외부가 내부 표현에 직접 의존하지 않도록 한다.

나. 객체가 제공하는 기능과 사용법은 공개하되 내부 상태와 구현은 함부로 변경하지 못하게 한다는 설명이다. 이는 **공개된 인터페이스를 통한 접근과 내부 구현의 보호**라는 캡슐화의 측면이다.

다. 관련된 데이터와 그 데이터를 처리하는 메서드를 하나로 묶는다는 설명이다. 내부 데이터의 변경을 정해진 메서드를 거치도록 통제하면 객체가 상태의 일관성을 유지할 수 있다. **데이터와 메서드의 결합**은 캡슐화의 핵심이다.

따라서 가·나·다는 모두 ③ 캡슐화를 설명한다. 모듈화는 시스템을 독립적인 구성단위로 나누는 더 넓은 개념이고, 상속성은 기존 클래스의 특성을 이어받아 확장하는 개념이다. 다형성은 같은 인터페이스나 호출에 대해 실제 객체의 유형에 따라 서로 다른 동작을 제공하는 개념이므로 구별한다.

NoteBase

문 24. 다음에서 설명하는 소프트웨어 개발 방법론으로 적절한 것은?

가. 1980년대 4세대 언어로부터 진화되어 데이터 중심 응용 시스템 개발을 위해 사용되었다.
나. 데이터를 생성, 검색, 표시, 보고하는 데이터베이스 프로그래밍 언어, 인터페이스 생성기, 보고서 생성기 등의 도구를 이용한다.
다. 재사용이 가능한 프로그램 컴포넌트의 개발을 강조하며 빠른 개발을 위해 객체 기술을 효과적으로 활용한다.
라. 비즈니스 모델링, 데이터 모델링, 프로세스 모델링, 애플리케이션 생성, 테스팅과 인수인계 과정을 통해 수행한다.

① RAD(Rapid Application Development) 방법론
② CBD(Component-Based Development) 방법론
③ Prototype 방법론
④ Spiral 방법론

정답 1번

간단한 해설

제시된 비즈니스·데이터·프로세스 모델링과 애플리케이션 생성 과정은 **RAD의 개발 단계**에 해당한다. 생성 도구와 재사용을 활용해 데이터 중심 응용 시스템을 빠르게 개발한다는 설명도 이에 부합한다.

NoteBase

구성 요소별 해설

RAD는 도구를 통한 자동화, 사용자 참여, 재사용 등을 활용하여 응용 시스템의 개발기간을 단축하는 방법론이다. 제시된 네 설명을 함께 연결하여 방법론을 식별한다.

가. 4세대 언어와 데이터 중심 개발: 데이터 조회·처리와 화면 및 보고서 작성에 적합한 고수준 언어·도구의 발전을 활용한다. 데이터 중심 업무용 응용 시스템을 빠르게 만드는 RAD의 배경에 해당한다.

나. 생성 도구의 활용: 데이터베이스 프로그래밍 언어, 인터페이스 생성기, 보고서 생성기를 이용하여 개발자가 모든 코드를 직접 작성하는 부담을 줄인다. **도구에 의한 자동화**로 개발 속도를 높인다는 설명이다.

다. 재사용과 객체 기술: 이미 만들어진 프로그램 구성요소를 활용하거나 재사용할 수 있게 개발하여 중복 작업을 줄인다. 재사용은 CBD에도 중요하지만, 이 단서만으로 방법론을 결정하지 않고 다른 설명과 함께 판단해야 한다.

라. 모델링에서 생성·시험으로 이어지는 과정: 비즈니스 모델링으로 업무의 정보 흐름을 파악하고, 데이터 모델링으로 필요한 데이터를 정의하며, 프로세스 모델링으로 데이터 처리 과정을 정한다. 이를 바탕으로 애플리케이션을 생성하고 시험·인수인계를 수행하는 단계 구성이 RAD와 연결된다.

네 단서를 종합하면 RAD 방법론인 ①이 적절하다. CBD는 컴포넌트의 조립·재사용, 프로토타입 방법론은 시제품을 통한 요구 확인, 나선형 방법론은 반복 과정의 위험 분석을 중심으로 구별된다. 특히 **제시된 모델링·애플리케이션 생성 단계의 결합**이 RAD를 식별하는 결정적 근거이다.

NoteBase

문 25. 소프트웨어 유지보수 활동에 대한 설명으로 적절하지 않은 것은?

① 소프트웨어 재공학(Reengineering)은 추상화된 정보를 얻기 위한 정공학(Forward engineering)과 새로운 요구사항에 맞춰 구현하는 역공학(Reverse engineering) 작업을 포함한다.

② 소프트웨어 재공학은 소프트웨어 시스템의 개선 차원에서 활용되며, 소프트웨어의 유지보수와 소프트웨어 개발의 선택 시점에서 적용된다.

③ 역공학은 유지보수 향상에 도움을 주는 설계에 대한 충분한 이해를 목적으로 하며, 대상 시스템의 변경이 아닌 분석에 중점을 둔다.

④ 소프트웨어의 리스트럭처링(Restructuring)은 비구조적인 코드를 구조적인 코드로 변환시키는 것으로 변경 부분, 변경 빈도, 자원 소프트웨어에 대한 의존도를 고려하여 수행한다.

정답 1번

간단한 해설

구현을 분석해 설계 같은 추상 정보를 얻는 것은 **역공학**이고, 요구와 설계를 실제 구현으로 발전시키는 것은 **정공학**이다. 두 작업의 방향을 뒤바꾼 설명이 잘못되었다.

선지별 해설

① X: 정공학과 역공학의 설명이 서로 바뀌었다. 기존 구현을 분석하여 설계·구조 등 **더 추상적인 정보를 복원하는 것은 역공학**이고, 요구와 설계를 바탕으로 **실행 가능한 구현을 만드는 것은 정공학**이다. 재공학은 기존 시스템을 이해하고 재구성한 뒤 개선된 형태로 구현하는 과정에서 두 작업을 활용할 수 있다.

② O: 재공학은 기존 소프트웨어의 유용한 기능과 지식을 활용하면서 구조·품질·유지보수성 등을 개선하는 접근이다. 기존 시스템을 계속 수정해 유지할지, 새로 개발할지를 검토하는 상황에서 전면 재개발의 비용·위험과 기존 자산의 재사용 가능성을 비교하여 적용할 수 있다.

③ O: 역공학의 목적은 구현에서 상위 수준의 구조와 설계 정보를 추출해 기존 시스템을 이해하는 데 있다. 코드와 구성 요소의 관계를 분석하여 문서화가 부족한 시스템의 유지보수를 돕지만, 역공학 자체가 곧 소프트웨어의 기능이나 코드를 변경하는 작업은 아니다. 그 분석 결과를 이용한 변경은 이후 재구성이나 정공학 등의 작업에서 이루어질 수 있다.

④ O: 리스트럭처링은 외부에서 관찰되는 기능을 유지하면서 내부 표현이나 구조를 개선하는 작업이다. 비구조적인 제어 흐름을 구조화된 코드로 바꾸는 것이 대표적 사례이다. 변경 대상과 변경 빈도, 사용하는 자원·지원 소프트웨어에 대한 의존 관계를 검토해야 영향 범위를 파악하고 기능을 보존할 수 있다. 구조화의 대상은 코드뿐 아니라 데이터 등도 될 수 있다.

NoteBase

문 26. 소프트웨어 프로세스 성숙도 모델인 CMMi의 Level 3과 Level 5에 해당하는 단계별 프로세스가 적절하게 짝지어진 것은?

가. 요구사항 개발 (Requirements development)
나. 근본 원인 분석과 해결책 (Causal analysis and resolution)
다. 프로세스 및 제품 품질 보증 (Process and product quality assurance)
라. 공급업체 계약관리 (Supplier agreement and management)
마. 정량적 프로젝트 관리 (Quantitative project management)
바. 통합을 위한 조직의 환경 (Organizational environment for integration)
사. 요구사항 관리 (Requirements management)
아. 제품 통합 (Product integration)

① Level 3: 바, 사, 아, Level 5: 다
② Level 3: 라, 마, 바, Level 5: 다
③ Level 3: 다, 라, 사, Level 5: 나
④ Level 3: 가, 바, 아, Level 5: 나

정답 4번

간단한 해설

정답은 ④이다. 문항의 단계별 분류에서 Level 3에는 요구사항 개발·통합을 위한 조직의 환경·제품 통합인 **가·바·아**가, Level 5에는 근본 원인 분석과 해결책인 **나**가 해당한다.

NoteBase

대응 관계 해설

가. 요구사항 개발은 Level 3에 해당한다. 고객의 요구를 분석하여 제품·구성요소의 요구사항으로 구체화하는 영역이다. 이미 정해진 요구사항의 변경과 일관성을 다루는 요구사항 관리와 구별한다.

나. 근본 원인 분석과 해결책은 Level 5에 해당한다. 결함이나 문제의 원인을 분석하고 원인을 제거하여 재발을 방지하는 등 지속적인 개선을 추진하는 영역이다.

다. 프로세스 및 제품 품질 보증은 Level 2에 해당한다. 수행한 프로세스와 산출물이 정해진 기준을 따르는지 객관적으로 확인하고 부적합 사항을 해결하도록 지원한다. Level 5의 원인 분석과는 다르다.

라. 공급업체 계약관리는 Level 2에 해당한다. 외부 공급자로부터 제품이나 서비스를 획득할 때 계약을 수립하고 계약 이행을 관리하는 영역이다.

마. 정량적 프로젝트 관리는 Level 4에 해당한다. 정량적 목표와 측정 자료를 이용하여 프로젝트의 프로세스 성과를 관리한다. 표준 프로세스를 정의하는 Level 3보다 높은 정량적 관리 수준에 속한다.

바. 통합을 위한 조직의 환경은 문항이 사용하는 구 CMMI 분류에서 Level 3에 해당한다. 통합된 업무 수행을 지원하는 조직 환경과 협업 기반을 마련하는 영역이다.

사. 요구사항 관리는 Level 2에 해당한다. 요구사항에 대한 합의와 변경을 관리하고 프로젝트 계획·산출물과의 일관성을 유지한다. Level 3의 요구사항 개발과 이름을 혼동하지 않아야 한다.

아. 제품 통합은 Level 3에 해당한다. 제품 구성요소를 결합하고 인터페이스의 적합성을 확인하여 통합된 제품을 완성하는 영역이다.

따라서 **Level 3은 가·바·아, Level 5는 나**로 연결되므로 정답은 ④이다. ①·②는 Level 2인 다를 Level 5로 연결했고, ③은 Level 2인 다·라·사를 Level 3으로 연결하였다.

NoteBase

문 27. 두 모듈이 정보를 교환할 때 레코드나 배열과 같은 자료구조를 구성하는 데이터 중 하나의 데이터만을 필요로 하지만 레코드나 배열 전체가 넘어오는 경우에 해당하는 결합(Coupling)으로 적절한 것은?

① 데이터 결합(Data coupling)
② 공통 결합(Common coupling)
③ 스탬프 결합(Stamp coupling)
④ 내용 결합(Content coupling)

정답 3번

간단한 해설

일부 데이터만 필요한데 **레코드나 배열 전체를 전달**하면 스탬프 결합이다. 필요한 개별 값만 전달하는 데이터 결합과 달리, 받는 모듈이 복합 자료구조에도 의존한다.

개념 구분

핵심 단서: 호출받는 모듈이 실제로 필요한 것은 자료구조의 한 데이터뿐인데, 인터페이스로는 **레코드나 배열 전체**가 전달된다. 결합도는 모듈 사이의 의존 정도를 나타내므로, 필요한 값뿐 아니라 그것을 담은 자료구조에도 의존하게 된다는 점에 주목한다.

스탬프 결합: 복합 자료구조를 매개변수로 전달하고, 이를 받는 모듈이 그 구조의 일부만 사용하는 경우에 해당한다. 예를 들어 직원의 이름만 필요한 모듈에 이름·주소·급여를 포함한 직원 레코드 전체를 넘기는 경우이다. 받는 모듈이 **필요 이상의 자료구조를 알아야 한다**는 점에서 스탬프 결합으로 분류한다.

다른 결합과의 구별: 데이터 결합은 모듈에 필요한 개별 데이터 값을 매개변수로 전달하는 경우이다. 공통 결합은 여러 모듈이 전역 변수나 공통 데이터 영역을 공유하는 경우이고, 내용 결합은 한 모듈이 다른 모듈의 내부 데이터나 코드에 직접 접근하는 경우이다. 이 문제는 전역 영역 공유나 다른 모듈 내부 침범이 아니라, 전달하는 인자의 범위가 필요 이상으로 큰 사례이다.

따라서 일부 데이터만 필요하지만 자료구조 전체를 전달한다는 조건에 대응하는 것은 ③ 스탬프 결합이다.

NoteBase

문 28. 다음 코드 사례에 해당하는 코딩 오류로 가장 적절한 것은?

```
char* f(int n)
{
    char *op;
    if (n > 0)
        op = (char*) malloc(size);
    if (n == 1)
        return NULL;
    return(op);
}
```

① NULL의 사용

② 버퍼 오버플로(Overflow)

③ 중복된 프리(Free) 선언

④ 메모리 누수(Leak)

정답 4번

간단한 해설

정답은 ④ **메모리 누수**이다. $n = 1$이면 malloc으로 할당한 뒤 **해제 없이 NULL을 반환**하여, 할당된 메모리를 가리키던 주소를 잃게 된다.

NoteBase

핵심 풀이

$n = 1$인 실행 경로를 따라가면 첫 번째 조건 $n > 0$이 참이므로 malloc(size)가 실행된다. 메모리 할당에 성공했다고 가정하면 op는 새로 할당된 메모리 영역을 가리킨다.

이어서 두 번째 조건 $n = 1$도 참이므로 함수는 return NULL로 종료한다. 이때 **할당된 메모리를 free하지도 않고 그 주소를 호출자에게 반환하지도 않는다**. 지역 포인터 op가 사라지면서 해당 메모리에 접근해 해제할 경로를 잃으므로 ④ **메모리 누수**가 발생한다.

NULL을 반환하는 행위 자체는 오류가 아니므로 ①이 아니다. 코드에는 할당 범위를 넘어서 데이터를 쓰는 동작이 없어 버퍼 오버플로를 직접 보여주지 않으며, free 호출 자체가 없으므로 중복 해제도 아니다.

누수 방지를 위해서는 이 조기 반환 경로에서 더 이상 필요하지 않은 메모리를 해제하거나, 메모리를 할당하기 전에 해당 반환 조건을 처리하도록 흐름을 정리해야 한다. 별도로 $n \leq 0$이면 초기화되지 않은 op를 반환하는 문제도 있으므로 포인터 초기화와 입력·할당 결과 검사가 필요하다. 다만 보기에서 묻는 핵심 결함은 할당 후 조기 반환으로 인한 누수이다.

NoteBase

문 29. 호텔 예약 시스템을 개발할 때, 기능 요구사항으로 적절하지 않은 것은?

① 고객이 객실 타입, 숙박 인원 등 예약 관련 사항을 입력하고 수정할 수 있어야 한다.

② 호텔 직원이 고객의 예약정보를 등록하고, 조회하고, 수정할 수 있어야 한다.

③ 시스템의 날짜 표기는 표준에 따라 YYYY-MM-DD 형식으로 제공해야 한다.

④ 시스템 사용자 역할별로 정보의 범위를 명시해야 한다.

정답 3번

간단한 해설

날짜를 YYYY-MM-DD 형식으로 표시하라는 요구는 예약 업무의 동작이 아니라 **출력 형식과 표준 준수에 관한 제약**이다. 이 문항에서는 기능 요구사항과 구별되는 비기능 요구사항에 해당한다.

선지별 해설

기능 요구사항은 시스템이 제공할 서비스와 입력에 따른 처리, 조회·변경할 정보 등 무엇을 해야 하는지를 규정한다. 비기능 요구사항은 그 기능이 갖추어야 할 품질이나 설계·표현·준수 기준 등의 제약을 규정한다. 여기서는 구체적인 예약 서비스와 날짜 표기 형식의 제약을 구별한다.

① O: 고객이 객실 타입과 숙박 인원 등을 입력하고 수정하는 것은 예약정보를 생성·변경하는 업무 기능이다. 사용자의 입력에 대해 시스템이 수행해야 할 동작을 명시하므로 기능 요구사항에 해당한다.

② O: 호텔 직원에게 예약정보 등록·조회·수정 서비스를 제공한다는 요구이다. 수행 주체와 제공할 데이터 처리 기능이 구체적이므로 기능 요구사항이다.

③ X: 날짜를 표시하는 기능의 존재가 아니라 **표시 형식을 YYYY-MM-DD로 제한**하고 표준을 준수하도록 요구한다. 이 문항의 구분에서는 비기능적 제약에 해당한다. 예약을 등록하거나 날짜를 조회하는 서비스 자체에 관한 요구와는 구별된다.

④ O: 사용자 역할에 따라 접근·조회·처리할 수 있는 정보의 범위를 정하는 요구로 이해한다. 예를 들어 고객은 자신의 예약을, 담당 직원은 업무상 허용된 예약을 다루도록 하는 구체적인 기능 범위의 명세이다. 보안과 관련된 모든 요구가 자동으로 비기능인 것은 아니며, **역할별 접근을 허용하거나 제한하는 동작**은 기능으로 명세할 수 있다.

NoteBase

문 30. DevOps 프로세스 단계별 수행 활동에 대한 설명 중 적절한 것은?

① 운영 및 모니터링 단계에서는 사용자의 경험과 반응을 추적하고, 개발척도의 달성 여부를 확인한다.

② 배포 단계에서는 애플리케이션에 대한 비즈니스 가치와 요구사항을 정의하고, 배포 척도의 조정과 배포 계획의 조정을 수행한다.

③ 테스트 단계에서는 정의된 요구사항을 기반으로 소스 코드를 개발하며 개발자 입장에서의 단위테스트를 수행한다.

④ 빌드 단계에서는 배포된 애플리케이션을 설치하는 과정에서 발생한 문제를 복구하고, 필요 시 스토리지, 데이터베이스, 네트워크 등의 인프라를 구축한다.

정답 1번

간단한 해설

정답은 ①이다. 운영 및 모니터링은 사용자 반응과 서비스 지표를 추적하여 목표 달성 여부를 확인하고 개선에 반영한다. 나머지는 계획·개발·배포의 활동을 다른 단계와 혼동한 설명이다.

선지별 해설

① (O) 운영 및 모니터링에서는 서비스의 상태와 성능뿐 아니라 사용자의 이용 경험·반응도 관찰한다. 수집한 로그와 지표를 통해 설정한 목표와 척도의 달성 여부를 확인하고, 개선이 필요한 사항을 다음 개발·계획 활동에 반영한다.

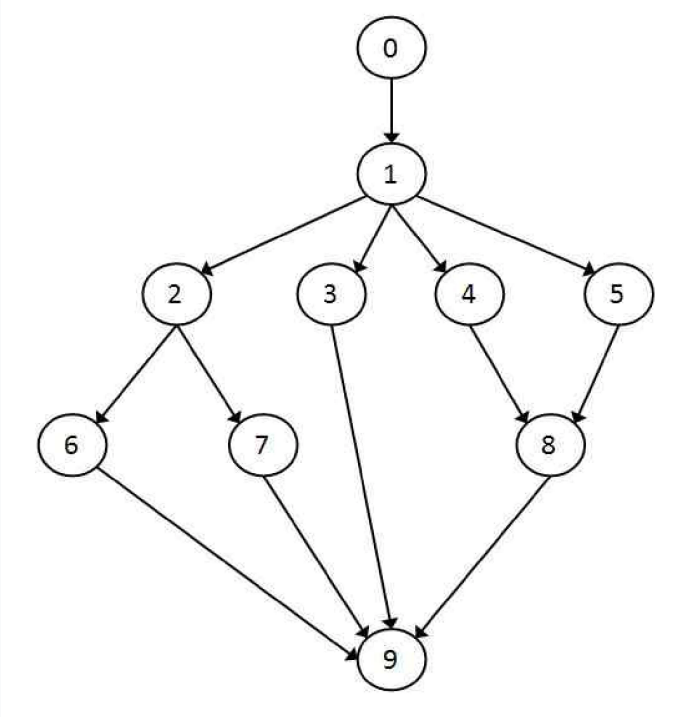
② (X) 애플리케이션이 제공할 비즈니스 가치와 요구사항을 정하고 목표 지표와 실행 계획을 조정하는 것은 주로 계획 단계의 활동이다. 배포 단계의 핵심은 준비된 소프트웨어를 대상 환경에 설치·적용하여 사용 가능하게 하는 것이므로, 요구사항 정의를 배포의 대표 활동으로 설명한 것은 적절하지 않다.

③ (X) 정의된 요구사항을 소스 코드로 구현하고 개발자가 단위 테스트로 동작을 확인하는 것은 개발·코딩 단계의 대표 활동이다. 테스트 단계의 중심은 구현된 결과가 기능·품질 요구사항을 충족하는지 검증하는 데 있다. 단위 테스트 자체가 테스트 활동인 것은 맞지만, 소스 코드 개발을 테스트 단계의 주된 활동으로 분류한 것이 잘못이다.

④ (X) 빌드는 소스 코드를 컴파일·결합하고 필요한 의존성을 반영하여 실행 파일이나 배포 가능한 패키지를 만드는 과정이다. 대상 환경의 인프라를 준비하거나 설치 실패를 복구하는 일은 배포·운영과 관련된 활동이므로 빌드와 구별해야 한다.

NoteBase

문 31. 아래 제시된 제어 흐름 그래프에서 McCabe의 순환 복잡도는?



① 3

② 4

③ 5

④ 6

정답 3번

간단한 해설

그림의 **노드는 10개, 간선은 13개**이고 하나로 연결되어 있다. 따라서 순환 복잡도는
$V(G) = E - N + 2 = 13 - 10 + 2 = 5$이다.

NoteBase

계산 과정

공식과 조건: McCabe의 순환 복잡도는 제어 흐름에서 선형 독립인 경로의 수를 나타낸다. 문제처럼 **하나로 연결된 제어 흐름 그래프**에서는 $V(G) = E - N + 2$를 사용한다. 여기서 E는 방향 간선의 수, N은 노드의 수이다.

노드와 간선 세기: 그림의 노드는 0부터 9까지이므로 N = 10이다. 간선은 0→1, 1→2, 1→3, 1→4, 1→5, 2→6, 2→7, 3→9, 4→8, 5→8, 6→9, 7→9, 8→9의 13개이므로 E = 13이다. 같은 도착 노드로 합쳐져도 출발 노드가 다른 화살표는 각각 별도의 간선으로 센다.

분기 수로 확인: 노드 1에는 출구가 4개이므로 독립 경로를 3개 늘리고, 노드 2에는 출구가 2개이므로 1개 늘린다. 기본 경로 1개에 더하면 $1 + (4 - 1) + (2 - 1) = 5$이다. **모든 분기가 이진 분기인 경우에만** 분기 노드 수에 1을 더하는 단순 공식을 그대로 쓸 수 있으므로, 이 그림에서 분기 노드 2개만 세어 3으로 계산하면 안 된다.

계산: $V(G) = 13 - 10 + 2 = 5$이다. 따라서 순환 복잡도는 5이며, ③과 일치한다.

NoteBase

문 32. 리팩토링이 필요하다고 판단되는 의심나는 상황 또는 증표를 부르는 용어는?

① 두 개의 모자

② 코드의 나쁜 냄새

③ F.I.R.S.T

④ TDD(Test Driven Development)

정답 2번

간단한 해설

정답은 ② **코드의 나쁜 냄새**이다. 리팩토링을 검토해야 할 **구조적 문제의 징후**를 뜻하며, 그 자체가 반드시 실행 오류라는 뜻은 아니다.

개념 구분

코드의 나쁜 냄새(Code Smell)는 내부 설계나 구조에 문제가 있을 가능성을 알려 주는 징후이다. 중복 코드, 지나치게 긴 메서드, 과도하게 많은 책임을 가진 클래스 등이 대표적이다. 이런 신호를 발견하면 리팩토링이 필요한지 살펴보므로 정답은 ②이다.

코드의 나쁜 냄새가 곧 실행 오류나 확정된 설계 결함이라는 뜻은 아니다. 코드가 정상 동작하더라도 유지보수하기 어려운 구조일 수 있으며, 냄새가 보여도 실제 개선 필요성은 맥락을 확인하여 판단해야 한다.

두 개의 모자는 새로운 기능을 추가하는 작업과 외부 동작을 유지하며 구조를 개선하는 리팩토링 작업을 구분하는 비유이다. 코드에서 발견되는 문제 징후의 명칭은 아니다.

F.I.R.S.T는 빠르고, 독립적이며, 반복 가능하고, 결과를 스스로 판정할 수 있으며, 적시에 작성되는 테스트를 강조하는 원칙이다.

TDD는 실패하는 테스트를 먼저 작성하고 통과하는 코드를 구현한 뒤 리팩토링하는 개발 방식이다. 둘 다 나쁜 냄새 자체를 가리키는 용어가 아니다.

NoteBase

문 33. 다음의 내용을 자료 사전으로 적절하게 표현한 것은?

학생정보는 ID, 성명, 성별, 생년월일, 나이로 구성되며, 생년월일과 나이는 둘 중 하나만 선택 가능하고, 성별은 생략할 수 있다.

① 학생정보 = ID + 성명 + (성별) + [생년월일|나이]

② 학생정보 = ID + 성명 + {성별} + [생년월일|나이]

③ 학생정보 = ID + 성명 + (성별) + {생년월일|나이}

④ 학생정보 = ID + 성명 + {성별} + {생년월일|나이}

정답 1번

간단한 해설

생략할 수 있는 성별은 **(성별)**, 둘 중 하나만 선택하는 생년월일과 나이는 **[생년월일|나이]**로 표현한다. 중괄호 { }는 생략이나 양자택일이 아니라 반복을 나타낸다.

핵심 풀이

자료 사전은 자료를 구성하는 요소와 결합·선택·반복 등의 관계를 기호로 정의한다. 이 문제는 성별의 생략 가능성과 생년월일·나이 중 하나를 선택한다는 조건을 서로 다른 기호로 표현해야 한다.

필수 요소의 결합: ID와 성명은 생략하거나 둘 중 하나만 고른다는 조건이 없으므로 모두 포함한다. 등호는 자료의 정의, +는 구성요소의 결합을 나타내므로 ID + 성명으로 쓴다.

성별의 생략: **소괄호 ( )는 선택적으로 포함할 수 있는 항목**을 뜻한다. 따라서 성별은 (성별)로 표현한다. 중괄호 { }는 반복을 뜻하므로 생략 가능성의 표기로 대신하지 않는다.

생년월일과 나이의 선택: **대괄호 안의 세로줄은 대안 중 하나의 선택**을 나타낸다. 둘 중 하나만 사용할 수 있으므로 [생년월일|나이]로 쓴다. {생년월일|나이}는 요구된 선택 관계를 올바르게 표현하지 못한다.

조건을 합치면 학생정보 = ID + 성명 + (성별) + [생년월일|나이]가 되며, 이는 ①과 일치한다.

NoteBase

문 34. 데이터 추상화에 대한 설명으로 적절하지 않은 것은?

① 데이터와 함께 데이터에 대한 조작도 함께 정의한다.

② 시스템의 복잡성을 줄이는 데 도움이 된다.

③ 서비스를 제공하는 객체의 집합으로 소프트웨어 시스템을 바라본다.

④ 함수 내부에 대한 상세 정보 없이 호출 방법과 무엇을 수행하는지에 집중한다.

정답 4번

간단한 해설

함수의 내부 절차 대신 호출 방법과 수행 내용에 집중하는 것은 **절차 추상화**이다. 데이터 추상화는 **데이터와 그 데이터에 허용되는 연산**을 함께 정의하면서 내부 표현을 감추는 데 초점을 둔다.

선지별 해설

① O: 데이터 추상화는 데이터를 단순한 저장 공간으로만 다루지 않고, 그 데이터에 허용되는 연산과 함께 하나의 추상적인 대상으로 정의한다. 예를 들어 스택은 내부 저장 방식보다 원소를 넣고 꺼내는 연산과 그 의미를 중심으로 이해한다. 사용자는 정해진 연산을 통해 데이터를 다룬다.

② O: 데이터의 내부 표현을 감추고 필요한 연산과 의미만 드러내면 사용자가 세부 구현을 모두 알아야 할 필요가 줄어든다. 그 결과 시스템을 이해할 때 다루어야 하는 정보가 줄고, 내부 표현의 변경이 다른 부분에 미치는 영향도 제한할 수 있어 복잡성을 낮추는 데 도움이 된다.

③ O: 객체지향적인 데이터 추상화에서는 상태와 그 상태에 작용하는 연산을 객체에 묶는다. 각 객체가 공개한 연산을 통해 서비스를 제공하고 서로 협력한다고 보면, 시스템을 서비스를 제공하는 객체들의 집합으로 이해할 수 있다. 이는 데이터와 조작을 함께 정의한다는 관점과 연결된다.

④ X: 함수의 내부 처리 절차를 감추고 호출 방법과 수행 결과에 집중하는 것은 **절차 추상화 또는 기능 추상화**에 대한 설명이다. 데이터 추상화의 초점은 **데이터의 표현을 감추고 데이터와 허용 연산을 정의**하는 데 있다. 두 추상화 모두 세부사항을 감춘다는 공통점은 있지만, 주된 추상화 대상이 함수의 처리 과정인지 데이터와 연산의 묶음인지가 다르다.

NoteBase

문 35. 화이트 박스 테스트 기법으로 적절하지 않은 것은?

① 분기 커버리지(Branch coverage) 기법

② 기본 경로 테스트(Basis Path Test) 기법

③ MC/DC(Modified Condition/Decision Coverage) 기법

④ 도메인 테스트(Domain Test) 기법

정답 4번

간단한 해설

정답은 ④ **도메인 테스트**이다. 도메인 테스트는 입력 영역과 경계값을 중심으로 시험하는 블랙 박스 기법으로 분류한다. 분기 커버리지·기본 경로 테스트·MC/DC는 프로그램 내부의 제어 구조를 분석하는 화이트 박스 기법이다.

선지별 해설

① (O) 분기 커버리지는 프로그램의 각 분기가 실행되었는지를 확인한다. 예를 들어 조건문에서 참 분기와 거짓 분기를 각각 실행하도록 테스트를 구성한다. 코드의 제어 구조를 기준으로 하므로 화이트 박스 기법이다.

② (O) 기본 경로 테스트는 제어 흐름 그래프를 분석하여 선형 독립적인 기본 경로들을 선정하고 실행한다. 순환 복잡도를 독립 경로 수를 판단하는 데 활용하며, 프로그램 내부의 제어 흐름에 근거하는 화이트 박스 기법이다. 가능한 모든 실행 경로를 빠짐없이 시험한다는 뜻은 아니다.

③ (O) MC/DC는 결정식을 이루는 각 조건이 전체 결정 결과에 독립적으로 영향을 줄 수 있음을 확인하는 기법이다. 각 조건과 결정 결과의 참·거짓을 확인하고, 특정 조건의 영향이 드러나는 테스트 쌍을 구성한다. 내부 조건식 구조를 기준으로 하므로 화이트 박스 기법이다.

④ (X) 도메인 테스트는 입력값의 영역을 구분하고 영역 내부와 경계에 해당하는 값을 선정하여 동작을 확인하는 데 초점을 둔다. 이 문항의 일반적인 분류에서는 **입력 영역을 바탕으로 하는 블랙 박스 기법**에 해당하며, 코드의 분기·경로·조건 구조를 직접 기준으로 삼는 나머지 기법과 구별된다.

NoteBase

문 36. 프로그램에 결함이 있어도 오작동이 늘 일어나는 것은 아니다. 결함이 오작동으로 이어지기 위한 조건이 아닌 것은?

① 도달성 조건(Reachability condition)

② 분할 조건(Partitioning condition)

③ 감염 조건(Infection condition)

④ 오차 전파 조건(Error propagation condition)

정답 2번

간단한 해설

정답은 ② **분할 조건**이다. 결함이 오작동으로 나타나려면 결함 위치에 **도달**하고, 내부 상태가 **감염**되며, 그 오차가 관찰 가능한 결과로 **전파**되어야 한다.

선지별 해설

① (O) 도달성 조건은 테스트 실행이 결함이 있는 코드 위치까지 도달해야 한다는 조건이다. 결함이 포함된 분기를 전혀 실행하지 않으면 그 결함이 해당 실행에서 오작동을 일으킬 기회도 생기지 않는다.

② (X) 분할 조건은 결함이 오작동으로 이어지는 데 필요한 도달·감염·전파의 조건에 포함되지 않는다. 입력 영역을 나누어 대표 테스트를 선택하는 분할 기법과, 실행 중 결함이 외부 오작동으로 드러나는 조건을 혼동해서는 안 된다.

③ (O) 감염 조건은 결함이 실행된 결과 프로그램의 내부 상태가 올바른 프로그램의 상태와 달라져야 한다는 조건이다. 결함 위치에 도달했더라도 특정 입력에서 계산 결과가 우연히 같다면 내부 상태가 감염되지 않아 오작동으로 이어지지 않을 수 있다.

④ (O) 오차 전파 조건은 잘못된 내부 상태가 최종 출력이나 관찰 가능한 동작까지 영향을 미쳐야 한다는 조건이다. 중간에 잘못된 값이 올바른 값으로 덮어써지거나 결과에 사용되지 않으면, 감염이 발생했어도 외부 오작동은 나타나지 않을 수 있다.

NoteBase

문 37. 객체지향 방법에 따른 개발 순서가 적절한 것은?

가. 클래스 단위로 테스트한다.
나. 릴리즈하여 운영중인 시스템으로 통합한다.
다. 코딩 표준을 작성한다.
라. 클래스 구현이 끝나면 인스펙션(Inspection)을 수행한다.
마. 요구사항과 상세 설계를 반영하여 메서드를 코딩한다.

① 가 - 나 - 다 - 라 - 마
② 마 - 라 - 다 - 가 - 나
③ 마 - 라 - 다 - 나 - 가
④ 다 - 마 - 라 - 가 - 나

정답 4번

간단한 해설

코딩 전에 공통 기준을 정하고, 구현된 클래스는 **인스펙션과 실행 테스트를 거친 뒤** 릴리즈하여 운영 시스템에 통합한다. 코드가 있어야 검토할 수 있고, 검증된 결과를 운영에 반영한다는 의존관계가 핵심이다.

NoteBase

올바른 순서

다 → 마 → 라 → 가 → 나

다. 코딩 표준 작성: 이름 규칙, 코드 형식, 예외 처리 방식 등 구현 시 따를 공통 기준을 먼저 정한다. **코드를 작성하기 전에 기준을 마련**해야 개발자들이 일관된 방식으로 구현하고 이후 검토에서도 같은 기준을 사용할 수 있다.

마. 메서드 코딩: 정해진 코딩 표준을 지키면서 요구사항과 상세 설계에 따라 클래스의 메서드를 구현한다. 인스펙션이나 실행 테스트를 하려면 먼저 검토·실행할 코드가 있어야 한다.

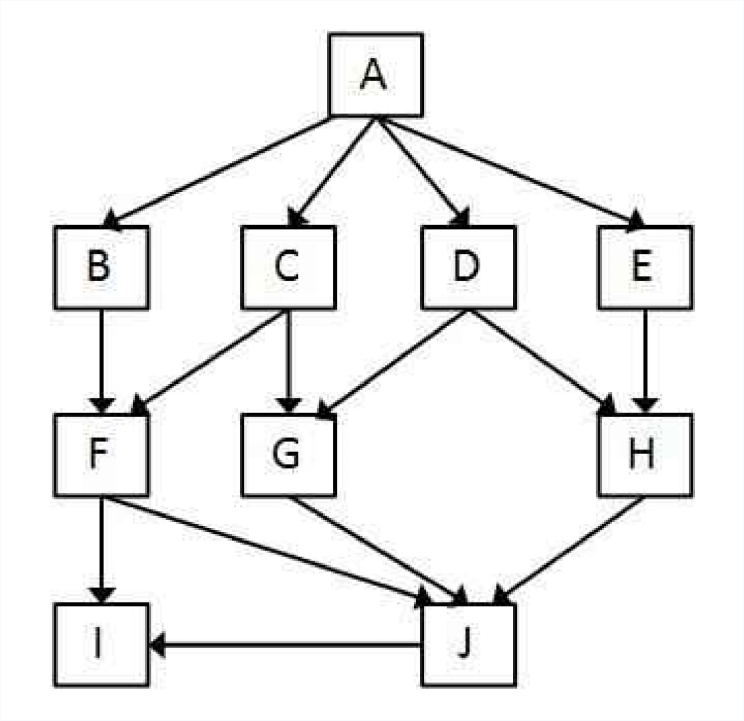
라. 인스펙션 수행: 클래스 구현이 끝나면 코드를 실행하기보다 사람이 체계적으로 검토하는 **정적 검토**를 수행한다. 설계 반영 여부, 코딩 표준 위반, 논리적 결함 등을 찾아 수정하여 다음 테스트 단계에 넘긴다.

가. 클래스 단위 테스트: 검토를 거친 클래스를 실행하여 메서드의 동작, 클래스 내부 상태와 경계 조건 등을 확인한다. 인스펙션이 코드를 읽고 결함을 찾는 방식이라면, 테스트는 **실행 결과를 기대 결과와 비교**한다.

나. 릴리즈 및 운영 시스템 통합: 앞 단계에서 검토·테스트한 구현물을 릴리즈하고 운영 중인 시스템에 통합한다. 제시된 단계에서는 구현 검증을 거친 뒤 운영에 반영하므로 이 활동이 마지막이다.

NoteBase

문 38. 아래 그림은 소프트웨어 구성 요소인 모듈의 계층적 구성을 나타내는 프로그램 구조도이다. 팬인(Fan-in)과 팬아웃(Fan-out)의 차이가 1인 모듈의 전체 개수는? (단, 화살표는 모듈 간 호출을 나타낸다.)



① 2

② 3

③ 4

④ 5

정답 3번

간단한 해설

정답은 ③ **4개**이다. 팬인과 팬아웃의 차이가 1인 모듈은 **C, D, G, H**이며, 각 모듈에 직접 들어오고 나가는 호출만 센다.

NoteBase

계산 과정

팬인은 해당 모듈을 **직접 호출하는 모듈의 수**, 팬아웃은 해당 모듈이 **직접 호출하는 모듈의 수**이다. 그림에서 들어오는 화살표와 나가는 화살표를 각각 센다. 여기서 두 값의 차이는 |팬인-팬아웃|로 계산한다.

A는 들어오는 호출이 없고 B·C·D·E를 호출하므로 (팬인, 팬아웃)이 (0, 4)이다.

B는 A의 호출을 받고 F를 호출하여 (1, 1)이다.

C는 A의 호출을 받고 F·G를 호출하여 (1, 2),

D는 A의 호출을 받고 G·H를 호출하여 (1, 2)이다.

E는 A의 호출을 받고 H를 호출하여 (1, 1)이다.

F는 B·C의 호출을 받고 I·J를 호출하여 (2, 2)이다.

G는 C·D의 호출을 받고 J를 호출하여 (2, 1),

H는 D·E의 호출을 받고 J를 호출하여 (2, 1)이다.

I는 F·J의 호출을 받고 다른 모듈을 호출하지 않으므로 (2, 0)이다.

J는 F·G·H의 호출을 받고 I를 호출하므로 (3, 1)이다. J → I 화살표의 방향을 반대로 세지 않도록 주의한다.

차이가 1인 모듈은 **C, D, G, H**이다. C·D는 $|1-2|=1$, G·H는 $|2-1|=1$이므로 전체 **4개**, 정답은 ③이다. 간접적으로 호출되는 후손 모듈까지 합산하지 않는다.

NoteBase

문 39. 다음에서 설명하는 디자인 패턴으로 적절한 것은?

가. 하위 클래스로 객체 생성에 대한 책임을 위임하고 하위 클래스의 정보를 은닉하고자 할 때 사용한다.
나. 객체를 만들기 위한 인터페이스를 정의하되, 하위 클래스가 인스턴스를 생성하는 클래스를 결정하도록 한다.
다. 생성해야 할 객체를 명세하는 하위 클래스가 필요한 경우에 사용한다.

① Adapter 패턴
② Facade 패턴
③ Factory method 패턴
④ Decorator 패턴

정답 3번

간단한 해설

어떤 구체 객체를 생성할지 하위 클래스가 결정하도록 하는 것이 Factory method 패턴의 핵심이다. 상위 수준에는 공통 생성 메서드를 두고 구체 생성 책임은 하위 클래스로 넘긴다.

NoteBase

대응 관계 해설

Factory method 패턴은 객체를 생성하는 공통 메서드를 정의하고, 실제로 어떤 구체 제품 객체를 만들지는 하위 클래스가 결정하도록 하는 생성 패턴이다. 세 설명은 모두 생성 책임과 구체 클래스 선택을 분리하는 이 구조에 대응한다.

가. 생성 책임의 위임과 정보 은닉: 상위 클래스는 구체 제품 클래스를 직접 지정해 생성하는 대신 생성 메서드를 호출한다. 하위 클래스가 그 메서드를 구현하거나 재정의하여 알맞은 제품을 만든다. 따라서 객체를 사용하는 공통 로직은 구체 생성 클래스의 세부사항을 알 필요가 줄어든다.

나. 생성 인터페이스와 구체 클래스의 결정: 객체 생성 메서드의 공통 형식은 상위 수준에서 정하지만, **반환할 구체 제품의 종류는 하위 클래스가 결정**한다. 여기서 인터페이스를 정의한다는 말은 생성 메서드의 계약을 뜻하며, 특정 언어의 interface 키워드 사용만을 의미하지 않는다.

다. 생성 객체를 명세하는 하위 클래스: 서로 다른 제품을 만들어야 한다면 각 하위 클래스가 생성 메서드에서 어떤 제품을 제공할지 지정할 수 있다. 공통 처리 흐름은 유지하면서 생성 결과를 하위 클래스에 따라 달리할 필요가 있을 때 적합하다.

따라서 세 설명에 대응하는 것은 ③ Factory method 패턴이다. Adapter는 호환되지 않는 인터페이스를 변환하고, Facade는 복잡한 하위 시스템에 단순한 창구를 제공하며, Decorator는 객체를 감싸 책임을 동적으로 추가한다. 이들은 구체 객체의 생성 결정을 하위 클래스에 맡기는 것이 핵심인 패턴과 구별된다.

NoteBase

문 40. 다음에서 설명하는 소프트웨어 테스트 방법으로 적절한 것은?

가. 변경된 소프트웨어 컴포넌트에 중점을 둔 테스트로, 변경으로 인하여 영향을 받게 될 소프트웨어의 기능에 초점을 맞춰 변경이 추가 오류를 가져오지 않는다는 것을 보장하는 활동이다.
나. 특수하고 중요한 기능을 가지는 최소 모듈 집합을 먼저 구현하고 보조적인 기능의 모듈은 나중에 구현하여 테스트한 후 계속 추가한다.

① 가: 회귀(Regression) Test, 나: 스모크(Smoke) Test
② 가: 회귀(Regression) Test, 나: 연쇄식(Threads) 통합 Test
③ 가: 백투백(Back-to-back) Test, 나: 스모크(Smoke) Test
④ 가: 백투백(Back-to-back) Test, 나: 연쇄식(Threads) 통합 Test

정답 2번

간단한 해설

변경 때문에 기존 기능에 새로운 오류가 생겼는지 확인하는 것은 **회귀 테스트**이다. 중요 기능을 수행하는 최소 모듈 집합부터 시험하고 보조 모듈을 추가하는 것은 **연쇄식 통합 테스트**이다.

대응 관계 해설

가와 나는 각각 변경 이후의 영향 확인과 기능 단위의 점진적 통합을 설명한다. 무엇을 확인하는 테스트인지와 어떤 단위로 모듈을 통합하는지를 구별한다.

가 → 회귀 테스트: 소프트웨어를 수정한 뒤 **기존에 정상 동작하던 기능에 새로운 결함이 생겼는지** 확인한다. 변경된 부분뿐 아니라 그 영향이 미칠 수 있는 기능을 다시 시험한다. 선지의 보장은 이러한 검증 목적을 뜻하며, 테스트가 모든 오류의 부재를 절대적으로 증명한다는 의미는 아니다. 백투백 테스트는 같은 입력을 서로 다른 구현이나 버전에 적용한 결과를 비교하는 방식이므로 제시된 설명과 구별된다.

나 → 연쇄식 통합 테스트: 특수하고 중요한 하나의 기능을 수행하는 데 필요한 **최소 모듈 집합을 먼저 연결하여 시험**하고, 보조 기능의 모듈을 순차적으로 추가한다. 모듈 계층의 위치만이 아니라 기능을 수행하는 처리 흐름을 통합 단위로 삼는다. 스모크 테스트는 새 빌드의 핵심 기능을 간단히 점검하여 본격적인 시험을 진행할 수 있는 상태인지 확인하는 것으로, 제시된 모듈 추가 전략 자체를 뜻하지 않는다.

따라서 가는 회귀 테스트, 나는 연쇄식 통합 테스트이며, 이 조합을 제시한 ②가 적절하다.

NoteBase