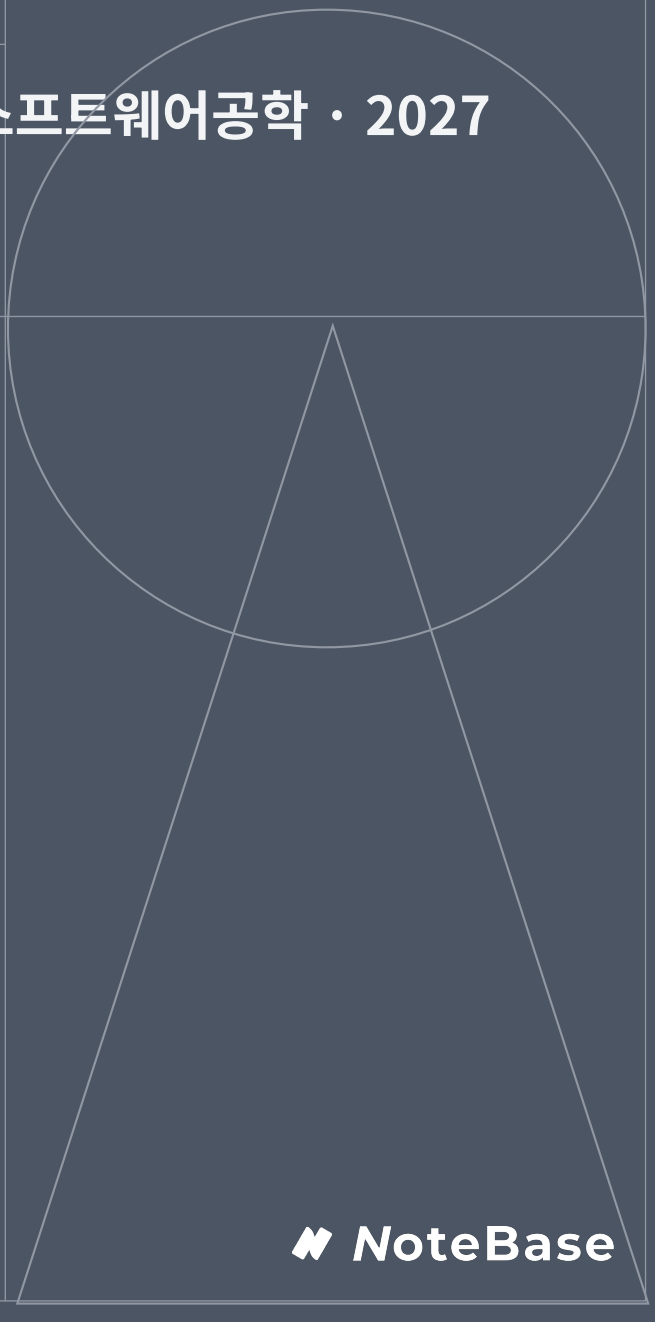
기출OX 해설집

공무원/경찰/공채(경위) · 소프트웨어공학 · 2027



NoteBase

기출OX 해설집

공무원/경찰/공채(경위) · 소프트웨어공학 · 2027

공무원/경찰/공채(경위) · 소프트웨어공학 · 2027

문 1. 프로젝트 착수 시 고객이 요구사항을 명확하게 정의하지 못하고 있다. 개발 프로세스 모델 중 현재 프로젝트에 적용하기에 가장 적절하지 않은 것은?

① 애자일(Agile) 모델

② 프로토타이핑(Prototyping) 모델

③ V 모델

④ 나선형(Spiral) 모델

정답 3번

간단한 해설

요구사항이 불명확하면 반복적인 고객 피드백으로 요구를 구체화해야 한다. 전통적인 V 모델은 **초기에 정한 요구사항을 기준으로 개발과 대응 테스트를 계획**하므로 이 상황에 상대적으로 가장 부적절하다.

NoteBase

핵심 풀이

핵심 단서는 프로젝트 착수 시점에 요구사항이 명확하지 않다는 점이다. 이 경우에는 고객에게 결과물을 보여 주고 피드백을 받아 **요구사항을 반복적으로 발견·수정할 수 있는 방식**이 유리하다.

V 모델은 요구사항 분석·설계 등 개발 단계와 인수·시스템·통합·단위 테스트를 대응시켜 검증을 계획하는 모델이다. 전통적인 순차형 V 모델은 초기에 요구사항을 비교적 명확히 정하고 그 기준으로 설계와 테스트를 구체화하므로, 요구사항이 불명확한 현재 상황에는 제시된 대안 중 가장 적절하지 않다. 변경이 절대 불가능하다는 뜻이 아니라 **요구 변경 시 앞 단계와 대응 테스트의 재작업 부담**이 커지기 쉽다는 뜻이다.

애자일은 짧은 반복과 고객 협업을 통해 변화하는 요구를 반영한다.

프로토타이핑은 시험용 구현이나 모형을 고객에게 보여 주어 막연한 요구를 구체화한다.

나선형은 반복마다 위험을 분석하고 필요하면 프로토타입 등을 활용하므로 요구사항의 불확실성을 점진적으로 다룰 수 있다.

따라서 요구사항을 탐색·정교화해야 하는 상황과 상대적으로 가장 맞지 않는 것은 전통적인 V 모델이며 ③과 일치한다. 애자일이나 나선형도 모든 프로젝트에 무조건 최적이라는 뜻은 아니며, 이 문제에서는 요구사항의 불명확성을 기준으로 비교한다.

NoteBase

문 2. 프로젝트 관리 산출물 중 WBS(Work Breakdown Structure)에 대한 설명으로 가장 적절하지 않은 것은?

① 전체 프로젝트 목표를 달성하고 결과물을 산출하기 위하여 수행해야 할 작업을 계층적으로 분할한다.

② 전체 목표와 중간의 세부 목표들이 트리(Tree)나 표(Table) 형태로 표현된다.

③ 전체 프로젝트 수행에 필요한 자원 예측과 프로젝트 일정 수립을 위한 스케줄링의 기초 자료로 활용된다.

④ WBS 작성 시 세부 작업 사이의 의존 관계와 임계경로를 파악하고 필요한 소요 자원을 할당한다.

정답 4번

간단한 해설

WBS는 수행할 작업의 범위를 계층적으로 분해한다. 작업 간 의존 관계와 임계경로를 파악하는 것은 **일정 네트워크 분석**과 구별해야 한다.

선지별 해설

① O: WBS는 프로젝트가 제공해야 할 산출물과 이를 완성하는 데 필요한 전체 작업 범위를 관리 가능한 하위 단위로 분해한 계층구조이다. 큰 작업을 더 작은 작업 패키지로 나누어 누락 없이 범위를 정의하는 데 사용한다.

② O: 전체 프로젝트 아래에 주요 산출물이나 작업 묶음을 두고 이를 세부 단위로 분해하므로, WBS는 트리나 들여쓰기된 표·목록으로 표현할 수 있다. 여기서 계층은 **상위 작업과 하위 작업의 포함 관계**를 뜻하며 시간적 선후 관계 자체를 뜻하지 않는다.

③ O: 분해된 작업 패키지를 기준으로 필요한 인력·장비·비용과 작업량 등을 추정하고, 이후 활동을 정의하여 일정을 수립할 수 있다. 따라서 WBS는 자원 예측과 스케줄링의 기초 자료이다.

④ X: 작업 간 선후 의존 관계를 연결하고 임계경로를 계산하는 것은 **네트워크 기반 일정계획의 역할**이다. 자원 할당 역시 이를 바탕으로 구체화하는 계획 활동이다. WBS 작성의 핵심은 작업 범위를 계층적으로 분해하는 것이며, WBS 자체가 의존 관계와 임계경로를 표시하는 일정 네트워크는 아니다.

따라서 WBS 작성과 일정 네트워크 분석을 혼동한 ④가 가장 적절하지 않다.

NoteBase

문 3. 고객관리시스템 업무인 회원관리 기능을 개발하기 위해 비용을 산정하고자 한다. 제시된 단위 업무 설명과 식별된 데이터 또는 트랜잭션의 수는 아래 표와 같다. 간이법을 통해 기능점수를 산출한다고 가정할 때, 총 기능점수는? (단, 평균 복잡도 가중치는 EI: 4.0, EO: 5.2, EQ: 3.9, ILF: 7.5, EIF: 5.4 이며, 제시된 단위 업무와 관련된 추가 세부 기능이나 다른 가정사항 및 제약조건은 없는 것으로 전제함)

단위 업무 설명	식별된 데이터 / 트랜잭션 수
고객관리시스템에는 회원관리 정보를 직접 저장하고 관리한다.	2
회원 고객의 결제 계좌 데이터는 이미 구축되어 운영중인 결제시스템으로부터 제공받는다.	1
신규고객은 '회원 가입하기'를 통해 회원 등록을 한다.	1
회원은 '회원 정보 조회'를 통해 자신의 정보를 조회한다.	1
회원은 '회원 정보 수정'을 통해 자신의 정보를 수정한다.	1
회원은 '회원 탈퇴'를 통해 비회원이 될 수 있다.	1
고객관리팀의 현업담당자는 '신규 회원 가입 통계 보고서 생성'을 통해 실시간 신규 회원 현황을 집계한다.	2

① 39.2

② 44.6

③ 45.4

④ 46.7

정답 4번

간단한 해설

등록·수정·탈퇴는 EI, 단순 조회는 EQ, 집계 보고서는 EO로 분류한다. 데이터 기능까지 포함해 **각 기능 수와 가중치의 곱을 합하면 46.7점**이다.

NoteBase

계산 과정

직접 저장·관리하는 회원관리 정보는 **시스템 내부에서 유지하는 데이터**이므로 내부논리파일 ILF이다. 표에 식별된 수 2를 적용하면 2×7.5=15.0점이다.

다른 결제시스템에서 유지하며 고객관리시스템이 제공받아 참조하는 결제 계좌 데이터는 외부연계파일 EIF이다. 1×5.4=5.4점이다.

회원 가입하기는 외부에서 받은 데이터로 내부 회원정보를 등록하므로 외부입력 EI이다. 1×4.0=4.0점이다.

회원 정보 조회는 추가 계산이나 파생 데이터 생성 없이 기존 정보를 검색하여 보여 주는 외부조회 EQ이다. 1×3.9=3.9점이다.

회원 정보 수정은 외부 입력으로 내부 회원정보를 변경하므로 외부입력 EI이다. 1×4.0=4.0점이다.

회원 탈퇴도 내부 회원정보를 삭제하거나 회원 상태를 변경하는 처리이므로 외부입력 EI이다. 1×4.0=4.0점이다.

신규 회원 가입 통계 보고서 생성은 단순 검색과 달리 **현황을 집계하여 파생 정보를 출력**하므로 외부출력 EO이다. 표에서 식별된 트랜잭션 수가 2이므로 2×5.2=10.4점이다.

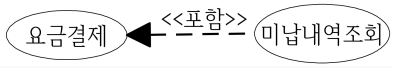
유형별로 합하면 ILF 2개, EIF 1개, EI 3개, EQ 1개, EO 2개이다. 간이법에서는 각 수에 제시된 평균 복잡도 가중치를 곱해 더한다.
총 기능점수=2×7.5+1×5.4+3×4.0+1×3.9+2×5.2=46.7점.

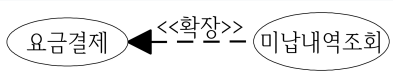
따라서 ④ 46.7에 해당한다.

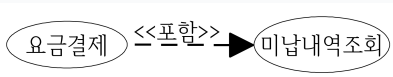
NoteBase

문 4. 다음 요구사항을 유스케이스 다이어그램으로 표현했을 때, 두 유스케이스 간의 관계로 가장 적절한 것은?

과금시스템은 요금 결제 시 미납 내역이 있는지를 확인하고 미납 내역이 있는 경우 결제금액에 포함하여 결제를 수행한다.

① 

② 

③ 

④ 

정답 2번

간단한 해설

미납분을 결제금액에 반영하는 처리는 **미납 내역이 있는 경우에만 추가**된다. 이 조건부 부가 흐름을 확장 관계로 나타내면 화살표는 **미납 관련 기능에서 기본 기능인 요금결제로** 향한다.

도식 해설

조건부 미납 처리: 요구사항에서 미납이 없으면 현재 요금만 결제하고, 미납이 있으면 미납분을 더하여 결제한다. 기본 결제 흐름에 이러한 부가 동작을 결합하는 관계가 확장(extend)이다. ②는 '미납내역조회'를 조회 결과에 따른 **미납분 반영까지 포괄하는 부가 흐름으로 해석할 때** 이해할 수 있는 도식이다.

확장 화살표의 방향: 확장 관계의 점선 화살표는 **확장하는 유스케이스에서 확장되는 기본 유스케이스로** 향한다. ②는 '미납내역조회 → 요금결제' 방향으로 '확장'을 표시한다. ④는 같은 확장 관계라도 방향이 반대여서, 요금결제가 미납내역조회를 확장하는 뜻이 된다.

포함 관계와의 구별: 포함(include)은 한 유스케이스의 동작을 다른 유스케이스의 동작 안에 넣어 사용하는 관계이며, 화살표는 포함하는 쪽에서 포함되는 쪽으로 향한다. 다만 이 요구사항에서 **미납 여부 확인은 항상 수행하고, 금액 합산은 미납이 있을 때 수행**한다. 따라서 '미납이 있을 때만 조회한다'고 설명해서는 안 된다. '미납내역조회'를 항상 수행하는 조회 동작만으로 좁게 분리한다면 ③의 '요금결제 → 미납내역조회' 포함 관계로 표현할 수 있다. ②의 확장 관계를 이해하려면 이 조회 동작과 조건부 미납 처리 흐름의 범위를 구별해야 한다. ①의 포함 화살표는 미납내역조회가 요금결제 동작을 포함한다는 반대 관계를 나타낸다.

NoteBase

문 5. 다음에 제시된 GetScore() 메소드에서는 value 변수를 매개변수로 사용한다. 소프트웨어 설계에서 발생하는 결합 유형으로 가장 적절한 것은? (단, 코드 실행에 대한 오류는 없는 것으로 함)

```
int GetScore(int value, void* buf, int count) {
                ...
    if ( value == 0 )
      /* read Buffer */
    else
      /* write Buffer */
                ...
}
```

변경 후

① 외부(external) 결합

② 제어(control) 결합

③ 데이터(data) 결합

④ 스탬프(stamp) 결합

정답 2번

간단한 해설

value는 읽기와 쓰기 중 수행할 동작을 고르는 값이다. 호출자가 전달한 정보로 **메소드 내부의 실행 흐름을 제어**하므로 제어 결합에 해당한다.

NoteBase

구성 요소별 해설

value의 역할: 호출자가 전달한 value가 0이면 버퍼 읽기, 0이 아니면 버퍼 쓰기 동작을 선택한다. 따라서 value는 단순히 처리될 자료가 아니라 **피호출 메소드의 내부 실행 흐름을 선택하는 제어 정보**이다.

결합 유형의 대응: 한 모듈이 다른 모듈에 플래그나 명령 값 등을 전달하여 그 모듈이 수행할 동작을 결정하면 제어 결합이다. 이 코드의 value와 읽기·쓰기 분기 관계가 바로 그 기준에 해당한다.

경쟁 유형과의 구분: 데이터 결합은 처리에 필요한 개별 데이터를 주고받는 관계이고, 스탬프 결합은 레코드·구조체 같은 복합 자료구조를 전달하는 관계이다. 외부 결합은 외부에서 정해진 자료 형식·통신 규약·장치 인터페이스 등에 함께 의존하는 관계이다. value가 정수형 매개변수라는 사실만으로 데이터 결합이라 판단하지 않고, **전달된 값이 수행 동작을 지시하는지**를 보아야 한다. buf와 count가 함께 전달된다는 사실도 질문이 지정한 value의 제어 역할을 바꾸지 않는다.

따라서 value에 의해 발생하는 결합은 ② 제어(control) 결합이다.

NoteBase

문 6. 다음 중 객체지향 설계에서 응집력(Cohesion)의 강도가 상대적으로 가장 높은 것은?

① 모듈 내부 요소들이 외부 출력 처리 루틴들로 묶여서 구성되어 있다.

② 모듈 내부 요소들이 모듈 내에서 연산이 수행되는 실행 순서에 따라 정의되어 묶여 있다.

③ 모듈 내부 요소들이 하나의 기능에 꼭 필요한 요소들로만 구성되어 있다.

④ 모듈 내부 요소들이 이전 문장의 실행 결과가 다음 문장의 입력으로 사용되는 관계로 묶여 있다.

정답 3번

간단한 해설

하나의 기능을 수행하는 데 꼭 필요한 요소만 모은 것은 기능적 응집력이다. **모든 요소가 단일 목적에 직접 기여**하므로 제시된 설명 중 가장 높은 응집력을 나타낸다.

선지별 해설

응집력은 한 모듈 내부의 요소들이 얼마나 밀접하게 하나의 목적에 연결되는지를 나타낸다. 제시된 분류에서는 단순한 작업 종류나 실행 순서의 공통성보다, 모든 요소가 한 가지 기능을 완성하는 데 꼭 필요한 관계를 더 높은 응집력으로 본다.

① 외부 출력 처리 루틴의 묶음: 여러 루틴을 출력이라는 유사한 작업 종류로 모은 것이므로 논리적 응집력에 해당하는 설명이다. 작업의 범주는 비슷하지만 **각 루틴이 하나의 동일한 기능을 위해 모두 필요한 것은 아니다**. 같은 데이터를 사용한다는 조건은 주어지지 않았으므로 통신적 응집력으로 단정하지 않는다.

② 실행 순서에 따른 묶음: 요소들이 일정한 수행 절차를 따른다는 이유로 모여 있으므로 절차적 응집력이다. **실행 순서의 연관성**은 있으나 한 요소의 결과가 다음 요소의 입력이라는 데이터 의존성까지 요구하는 것은 아니다.

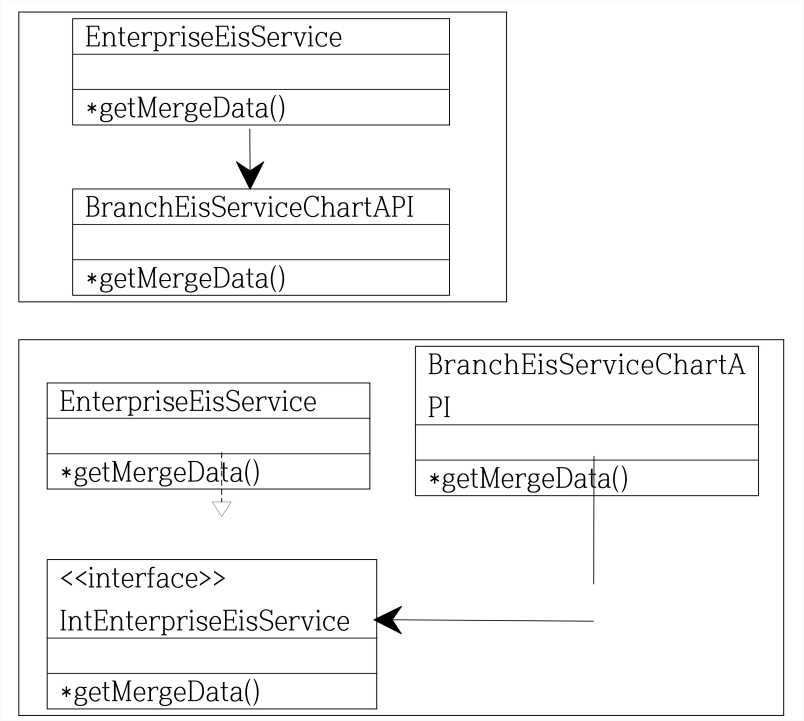
③ 하나의 기능에 꼭 필요한 요소의 묶음: 모든 요소가 단일한 기능을 수행하는 데 기여하므로 기능적 응집력이다. 단순히 종류나 순서가 비슷한 수준을 넘어 **전체가 한 기능의 완성을 위해 결합**하므로 제시된 네 설명 중 응집력이 가장 높다.

④ 앞 결과가 다음 입력이 되는 묶음: 연산 사이에 출력·입력의 흐름이 이어지므로 순차적 응집력이다. 절차적 응집력보다 데이터 흐름의 연결이 강하지만, 여러 처리 단계가 이어진다는 것만으로 모든 요소가 하나의 기능에 꼭 필요한 기능적 응집력과 같아지는 것은 아니다.

제시된 분류의 상대적 순서는 논리적 < 절차적 < 순차적 < 기능적 응집력이며, 가장 높은 것은 ③이다. 응집력은 모듈 내부의 관련성이고, 모듈 간 의존성을 나타내는 결합도와는 구별한다.

NoteBase

문 7. 본사의 EnterpriseEisService 클래스가 지사의 모듈 변경에 쉽게 영향을 받지 않도록 BranchEisServiceChartAPI 연계용 인터페이스 클래스를 정의하였다. 설계 변경에 반영된 SOLID 원칙으로 가장 적절한 것은?



① 개방 폐쇄의 원리

② 인터페이스 분리의 원리

③ 의존 관계 역전의 원리

④ 리스코프 교체의 원리

정답 3번

간단한 해설

특정 지사 구현에 직접 의존하지 않고 인터페이스라는 **추상화에 의존**하도록 변경했다. 구현 변경의 파급을 줄이는 의존 관계 역전 원리에 해당한다.

NoteBase

대응 관계 해설

변경 전의 의존 관계: 위쪽 그림에서는 본사의 EnterpriseEisService에서 지사의 BranchEisServiceChartAPI로 직접 향하는 관계가 제시된다. 본사의 상위 기능이 특정 지사 구현에 직접 의존하면 그 구현 변경의 영향을 받기 쉽다.

추상화의 역할: 아래쪽 그림에는 getMergeData()를 선언한 IntEnterpriseEisService 인터페이스가 추가되고 두 클래스의 관계가 이 인터페이스를 중심으로 연결된다. 핵심은 특정 구현을 직접 참조하는 대신 **안정된 추상 계약을 사이에 두는 것**이다.

원칙과의 대응: 의존 관계 역전 원리는 상위 수준 모듈과 하위 수준 모듈이 모두 추상화에 의존하도록 하고, 추상화가 세부 구현에 종속되지 않도록 한다. 이 구조에서는 인터페이스 계약을 유지하는 범위에서 지사 구현을 변경해도 본사 기능에 미치는 영향을 줄일 수 있다.

다른 원칙과의 구별: 개방 폐쇄 원리는 확장에는 열려 있고 기존 코드 수정에는 닫혀 있어야 한다는 원리이다. 인터페이스 분리 원리는 사용하지 않는 기능에 의존하지 않도록 인터페이스를 고객별로 나누는 원리이며, 인터페이스를 하나 도입했다는 사실만으로 해당하지 않는다. 리스코프 교체 원리는 하위 타입이 상위 타입의 계약을 지켜 대체 가능해야 한다는 원리이다. 이 사례의 직접적인 단서는 **구체 모듈에 대한 의존을 추상화로 전환**한 것이다.

따라서 가장 적절한 원칙은 ③ 의존 관계 역전의 원리이다.

NoteBase

문 8. 다음에서 설명하고 있는 소프트웨어 아키텍처 스타일로 가장 적절한 것은?

◯ 컴포넌트들을 사용자 인터페이스로부터 비즈니스 로직을 분리하여 뷰를 모델과 독립적으로 변경하고 제어할 수 있음
◯ 기능이나 특성에 따라 모듈을 보다 명확히 분리하여 유지보수성, 프로그램 확장성과 유연성을 높임
◯ 대표적 사례로는 GUI 모델이 있음

① MVC
② 파이프 필터
③ P2P(Peer-to-Peer)
④ 레파지토리(데이터 중심)

정답 1번

간단한 해설

사용자 인터페이스와 비즈니스 로직을 분리하고 뷰를 독립적으로 변경한다는 설명은 **MVC의 역할 분리**에 해당한다. 화면 표현은 뷰가, 데이터와 업무 처리는 모델이 담당한다.

공통 해설

MVC(Model–View–Controller)는 프로그램을 **모델·뷰·컨트롤러의 세 역할**로 분리하는 아키텍처 스타일이다. 모델은 데이터와 비즈니스 로직을, 뷰는 사용자에게 보여 주는 화면을 담당한다. 컨트롤러는 사용자 입력을 받아 모델의 처리와 뷰의 표시를 조정한다. 지문의 '사용자 인터페이스로부터 비즈니스 로직을 분리'하고 '뷰를 모델과 독립적으로 변경'한다는 특성은 이러한 역할 분리를 설명한다.

이처럼 역할을 나누면 화면 표현을 바꿀 때 업무 처리 부분에 미치는 영향을 줄일 수 있어 **유지보수성과 확장성·유연성**을 높일 수 있다. 예를 들어 업무 계산 규칙을 유지하면서 결과를 표시하는 화면 구성을 바꾸기 쉬워진다. 사용자 입력, 화면 표시, 내부 처리를 구분할 필요가 있는 GUI는 MVC를 적용하는 대표적인 사례이다.

NoteBase

문 9. 다음의 상황을 고려하여 GoF 패턴을 선정하고자 할 때, 가장 적절한 것은?

◯ 문제 및 배경 : 서브시스템에 다수의 인터페이스가 존재하기 때문에 모든 인터페이스와 상호작용하기 위한 구현을 개별적으로 수행해야 하는 상황임
◯ 솔루션 : 서브 시스템에 있는 다수의 인터페이스 집합에 대해 하나의 통일된 인터페이스를 제공할 수 있는 포괄된 개념의 인터페이스를 정의하고자 함
◯ 결과 및 기대효과 : 클라이언트가 다루어야 할 객체 수가 감소되면서도 서브시스템 구성 요소를 보호할 수 있고, 서브시스템과 클라이언트 코드 간 결합도가 감소될 것을 기대할 수 있음

① Facade
② Bridge
③ Adapter
④ Proxy

정답 1번

간단한 해설

서브시스템의 여러 인터페이스를 **하나의 통일된 인터페이스**로 감싸서 사용을 단순화하는 것은 Facade 패턴이다. 클라이언트가 내부 구성 요소를 일일이 다루지 않아도 되므로 서브시스템과의 결합도가 낮아진다.

공통 해설

Facade 패턴은 복잡한 서브시스템의 기능을 쉽게 이용하도록 **상위 수준의 통일된 인터페이스**를 제공한다. 클라이언트는 이 인터페이스에 작업을 요청하고, Facade 객체가 필요한 내부 객체들과의 상호작용을 맡는다. 따라서 클라이언트가 여러 내부 인터페이스를 직접 호출하도록 구현할 부담이 줄어든다.

제시된 상황은 다수의 인터페이스를 개별적으로 다루던 방식을 하나의 인터페이스를 통해 이용하는 방식으로 바꾸려는 것이므로 Facade의 목적에 해당한다. 클라이언트가 직접 알아야 하는 객체와 내부 호출 관계가 줄어들어 서브시스템의 구성 세부 사항을 감출 수 있고 결합도도 낮아진다. 이때 구성 요소를 보호한다는 것은 주로 **내부의 복잡성과 직접 의존을 줄인다**는 의미이며, 내부 객체에 대한 모든 직접 접근을 반드시 금지한다는 뜻은 아니다.

NoteBase

문 10. 이것은 UI 설계단계에서 페이지에 대한 개략적인 레이아웃이나 UI 요소 등에 대한 뼈대를 설계하는 방법으로, 각 페이지의 영역 구분 콘텐츠, 텍스트 배치 등을 화면 단위로 설계하는 일종의 웹페이지나 애플리케이션 레이아웃 스케치이며, 디자인 세부 사항보다는 구조와 기능에 중점을 둔다. 이것이 의미하는 UI 설계 과정의 주요 개념으로 가장 적절한 것은?

① 프로토타입

② 와이어프레임

③ 스토리보드

④ 여정 지도

정답 2번

간단한 해설

와이어프레임은 UI 요소와 콘텐츠의 위치를 잡는 **화면의 구조적 뼈대**이다. 세부 시각 디자인보다 영역 구분과 배치에 중점을 둔다는 단서로 구별한다.

대응 관계 해설

설명과 개념의 대응: 페이지의 영역을 나누고 콘텐츠·텍스트·버튼 등 UI 요소의 위치를 개략적으로 배치하는 화면 골격은 와이어프레임이다. 문항의 **화면 단위 레이아웃 스케치와 구조 중심 설계**라는 단서가 이에 해당한다.

프로토타입과의 구분: 프로토타입은 완성 제품을 만들기 전에 기능이나 상호작용을 시험하는 시제품이다. 낮은 충실도의 프로토타입도 있을 수 있으나, 이 문항은 동작 검증보다 화면의 뼈대와 배치에 초점을 둔다.

스토리보드와의 구분: 스토리보드는 화면 구성과 함께 화면 전환, 사용 시나리오, 처리 절차나 기능 설명 등을 흐름에 따라 정리하는 설계 자료이다. 화면의 개략적인 배치 자체를 가리키는 와이어프레임과 구별한다.

여정 지도와의 구분: 여정 지도는 사용자가 목표를 달성하는 과정의 단계·접점·행동·감정·불편 등을 정리한다. 개별 페이지의 요소 배치보다는 **사용자의 전체 경험 흐름**을 이해하는 데 목적이 있다.

따라서 제시된 설명은 ② 와이어프레임에 대응한다.

NoteBase

문 11. 통계적 방법을 사용하여 가능한 변수 조합을 선택함으로써, 테스트 케이스 수를 줄이면서도 대표적인 조합을 고르게 포함하고, 높은 커버리지를 유지할 수 있는 블랙박스 테스트 기법으로 가장 적절한 것은?

① 직교(orthogonal) 배열

② 페어와이즈(pairwise)

③ 원인-결과 그래프

④ n-way

정답 1번

간단한 해설

직교 배열은 통계적 실험계획에 따라 변수의 값 조합이 **균형 있게 나타나도록** 테스트 케이스를 선정한다. 모든 입력 조합을 검사하는 부담을 줄이면서, 정해진 범위의 조합을 고르게 포함하는 방식이다.

개념 구분

직교(orthogonal) 배열: 각 열은 입력 변수, 각 행은 하나의 테스트 케이스를 나타낸다. 정해진 개수의 변수를 어느 것으로 골라도 가능한 값 조합들이 **같은 횟수로 나타나도록** 배열한다. 이 통계적 균형을 이용해 전체 조합보다 적은 테스트로 정해진 조합 커버리지를 확보한다. 문항의 '통계적 방법'과 '대표적인 조합을 고르게 포함'한다는 설명이 이 특성에 해당한다. 다만 선택한 범위보다 더 많은 변수가 함께 작용하는 모든 경우까지 검사했다는 뜻은 아니다.

페어와이즈(pairwise): 임의의 두 변수에 대해 가능한 값의 쌍을 **적어도 한 번씩 포함**하도록 테스트를 구성한다. 테스트 수를 줄인다는 점은 직교 배열과 공통되지만, 모든 쌍이 같은 횟수로 나타나야 하는 것은 아니다. 두 변수 조합을 균형 있게 배치한 직교 배열도 페어와이즈 커버리지를 만족할 수 있으므로 두 기법을 완전히 별개로 보아서는 안 된다.

원인-결과 그래프: 입력 조건을 원인, 출력이나 동작을 결과로 두고 둘 사이의 **논리 관계**를 그래프로 표현하여 테스트를 도출한다. 예를 들어 두 조건이 모두 충족되어야 특정 동작이 일어나는 관계를 나타낼 수 있다. 통계적으로 값 조합의 출현 빈도를 고르게 만드는 것이 중심인 직교 배열과 구별된다.

n-way: 임의로 선택한 n개 변수에 대해 가능한 값 조합을 포함하도록 하는 조합 테스트의 커버리지 기준이다. 페어와이즈는 두 변수 조합을 다루는 2-way에 해당한다. n-way라는 명칭은 **함께 다루는 변수의 수**를 나타내며, 직교 배열처럼 각 조합의 출현 횟수가 같아야 한다는 조건까지 뜻하지는 않는다.

NoteBase

문 12. 통합테스트와 관련된 주요 용어에 대한 설명으로 가장 적절하지 않은 것은?

① 연쇄식 통합에서 연쇄(Thread)란 특정 기능을 수행하는 모듈의 최소 단위이다.

② 테스트 드라이버(Driver)는 시험 대상 모듈을 호출하는 역할을 하는 간이 소프트웨어이다.

③ 테스트 스텁(Stub)은 시험 대상 모듈에 의해 호출되는 모듈의 기능을 대신하는 간이 소프트웨어이다.

④ 테스트 오라클(Oracle)은 시스템을 테스트하기 위하여 작성된 별도의 프로그램으로 테스트가 끝난 후 삭제된다.

정답 4번

간단한 해설

테스트 오라클은 **예상 결과와 정오 판단의 기준**을 제공한다. 테스트 후 삭제되는 별도 프로그램으로 한정되는 개념이 아니다.

선지별 해설

① O: 연쇄식 통합은 하나의 기능이나 사용 시나리오가 실행되는 데 필요한 모듈들을 묶어 통합·시험한다. 여기서 연쇄는 **특정 기능을 수행하는 데 필요한 최소 통합 단위**로 이해한다. 운영체제의 실행 스레드 자체를 뜻하는 것이 아니라 기능을 따라 연결되는 모듈의 묶음이다.

② O: 테스트 드라이버는 시험 대상 모듈을 호출하고 입력을 제공하며 결과를 받는 상위 호출자 역할을 대신한다. 실제 상위 모듈이 아직 준비되지 않았을 때 대상 모듈을 실행할 수 있게 한다.

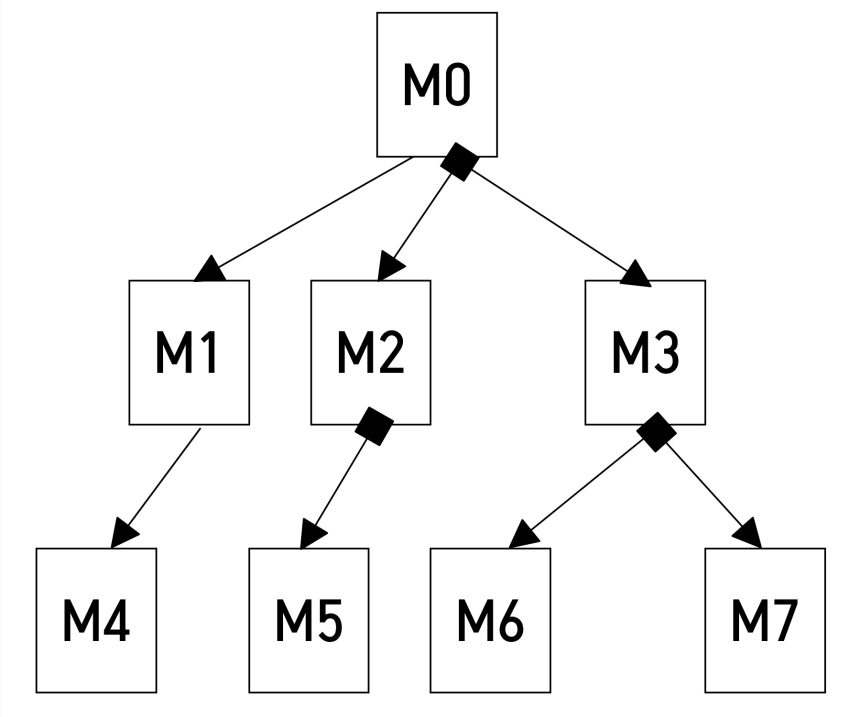
③ O: 테스트 스텁은 시험 대상 모듈이 호출하는 하위 모듈을 대신한다. 호출에 대해 미리 정한 응답을 반환하는 등 필요한 최소 동작을 제공하므로, 실제 하위 모듈이 없어도 상위 모듈을 시험할 수 있다.

④ X: 테스트 오라클은 테스트의 **기대 결과를 결정하고 실제 결과의 적합성을 판단하는 기준이나 정보원**이다. 명세, 검증된 계산 결과, 참조 시스템 등이 그 역할을 할 수 있으며 반드시 별도 프로그램일 필요는 없다. 테스트가 끝나면 삭제해야 한다는 성질도 없으므로 임시 시험용 프로그램이라는 설명과 구별해야 한다.

따라서 테스트 오라클의 의미를 잘못 설명한 ④가 가장 적절하지 않다.

NoteBase

문 13. 다음에 제시된 모듈 구조의 설계 복잡도 및 통합 복잡도에 대한 설명으로 가장 적절한 것은? (단, M은 모듈, 다이아몬드(◆)는 이진 조건 분기, 화살표(→)는 호출을 의미함)



① M0 모듈의 설계 복잡도는 13이다.

② M2 모듈의 설계 복잡도는 2이다.

③ M3 모듈의 설계 복잡도는 3이다.

④ M0 모듈의 통합 복잡도는 4이다.

정답 4번

간단한 해설

전체 호출 구조는 모듈 8개와 이진 분기 3개로 이루어져 설계 복잡도가 $8 + 3 = 11$이다. 통합 복잡도는 **모듈별 기본 경로의 중복을 제외**하여 $11 - 8 + 1 = 4$로 계산한다.

NoteBase

계산 과정

계산 기준: 이 문항에서 설계 복잡도는 해당 모듈을 루트로 하는 호출 구조의 S_0를 뜻한다. 각 모듈의 호출 관련 이진 분기 수에 1을 더한 모듈 설계 복잡도 iv를 합산하여 $S_0 = \sum iv$를 구한다. **개별 모듈과 하위 구조 전체의 복잡도를 구별**해야 한다.

전체 구조의 설계 복잡도: 모듈은 M0부터 M7까지 8개이고, 호출에 관여하는 이진 분기는 M0·M2·M3에 각각 하나씩 있어 총 3개이다. 따라서 $iv(M0) = iv(M2) = iv(M3) = 2$이고 나머지 다섯 모듈은 각각 1이다. $S_0(M0) = 3 \times 2 + 5 \times 1 = 11$이므로 13이 아니다.

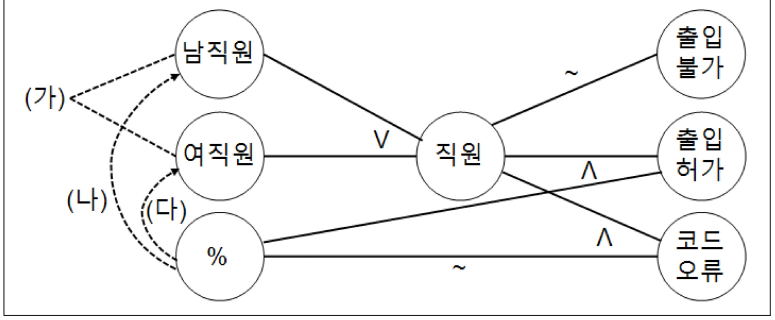
하위 구조의 설계 복잡도: M2를 루트로 하면 M2·M5의 두 모듈과 분기 하나가 있으므로 $S_0(M2) = 2 + 1 = 3$이다. 여기서 2는 M2 자체의 iv이지 하위 구조 전체의 값이 아니다. M3를 루트로 하면 M3·M6·M7의 세 모듈과 분기 하나가 있으므로 $S_0(M3) = 3 + 1 = 4$이다.

통합 복잡도: 모듈 수를 n이라 할 때 $S_1 = S_0 - n + 1$이다. 모듈마다 포함한 기본 경로의 중복을 제외하고 전체 구조의 기본 경로 하나를 남기므로, $S_1(M0) = 11 - 8 + 1 = 4$이다. **M0 구조의 통합 복잡도가 4**라는 ④와 일치한다. 복잡도는 화폐나 시간 단위가 없는 경로 수 기반 지표이다.

NoteBase

문 14. 다음은 소프트웨어 테스트를 위해 작성된 테스트 케이스와 원인-결과 그래프이다. (가)~(다)의 제한 조건 기호로 가장 적절한 것은?

테스트 케이스
○ 첫 번째 열이 A 또는 B이고 두 번째 열이 %이면 '출입 허가'를 출력한다. ○ 첫 번째 열이 A 또는 B가 아니면 '출입 불가'를 출력한다. ○ 첫 번째 열이 A 또는 B이고 두 번째 열이 %가 아니면 '코드 오류'를 출력한다. ○ A는 남직원, B는 여직원을 의미한다.
원인-결과 그래프



① (가): E, (나): R, (다): R

② (가): E, (나): M, (다): R

③ (가): I, (나): R, (다): M

④ (가): I, (나): M, (다): M

정답 1번

간단한 해설

첫 번째 열은 A와 B를 동시에 가질 수 없지만 둘 다 아닐 수 있으므로 **배타 제한 E**이다. 공식 답은 ①이며, 원인 사이의 방향 있는 점선 제한을 **R**로 식별한 기호 배열이다. 다만 그림의 두 화살표에 R의 의미를 그대로 적용하면 %가 참일 때 남직원과 여직원이 모두 참이어야 하므로 E와 충돌한다. 따라서 공식 채점 답과 테스트 케이스를 일관되게 표현하지 못하는 원문 도식의 한계를 구분해야 한다.

출제 포인트

남직원·여직원 사이의 점선 연결과 % 조건에서 이어지는 점선 화살표를 보고 원인-결과 그래프의 배타 제한 E와 요구 제한 R을 구분하는 문제이다.

NoteBase

핵심 개념

원인-결과 그래프는 입력 조건인 원인과 출력인 결과의 논리 관계를 표현한다. 제한 조건은 함께 성립할 수 있는 조건의 조합을 한정한다.

E(Exclusive, 배타): 연결된 원인들이 동시에 참일 수 없으며, 모두 거짓인 경우는 허용한다. 이 문제에서 첫 번째 열이 A이면 남직원 조건만 참이고, B이면 여직원 조건만 참이다. A도 B도 아닌 값이면 두 조건이 모두 거짓이 되어 출입 불가를 출력한다.

I(Inclusive, 포함): 연결된 원인 중 적어도 하나는 참이어야 한다. 모두 거짓인 경우를 허용하지 않는다는 점에서 E와 다르다.

R(Requires, 요구): 점선 화살표의 출발 원인이 참이면 도착 원인도 참이어야 한다. 그림에서 (나)는 %에서 남직원으로, (다)는 %에서 여직원으로 향한다. 두 제한을 R로 해석하면 %가 참일 때 남직원과 여직원이 각각 참이어야 한다.

M(Mask, 마스크): 한 결과가 참일 때 다른 결과를 거짓으로 만드는 제한이다. 이 문제의 (나)와 (다)는 결과가 아니라 입력 원인 사이에 있으므로 M을 쓰지 않는다.

도식의 한계: 두 R이 요구하는 직원 조건의 동시 참은 (가)의 E가 금지하므로, 이 제한들을 함께 적용하면 %가 참인 입력은 모두 배제된다. 첫 번째 열이 A이고 두 번째 열이 %인 경우, 테스트 케이스는 출입 허가를 요구하지만 (다)의 R은 거짓인 여직원 조건까지 참으로 요구한다. B·%인 경우도 (나)의 R이 거짓인 남직원 조건을 참으로 요구한다. 따라서 ①은 기호 종류를 구별한 공식 채점 답이며, 이 도식을 테스트 케이스에 부합하는 모델로 인정한다는 뜻은 아니다.

선지별 해설

① O(공식 채점 기준): (가)의 E는 남직원과 여직원 조건이 동시에 참일 수 없고 모두 거짓일 수 있다는 입력 조건에 맞는다. (나)와 (다)는 원인 사이의 방향 있는 점선 제한이므로 기호 형식상 R로 식별한다. 그러나 실제 방향은 각각 %→남직원, %→여직원이어서 두 R을 함께 적용하면 %가 참일 때 두 직원 조건이 모두 참이어야 한다. 이는 E와 충돌하고 A·% 및 B·%의 출입 허가 사례를 배제하는 원문 도식의 문제이다.

② X: 공식 답의 기호 배열과 비교하면 (가)의 E와 (다)의 R은 일치하지만, (나)의 M은 결과 사이의 마스크 제한이다. 원인 사이의 방향 있는 제한을 나타내는 기호로는 (나)의 M을 R로 고친다.

③ X: (가)의 I는 남직원 또는 여직원 조건이 반드시 하나 이상 참이어야 한다는 뜻이다. 이는 첫 번째 열이 A도 B도 아닌 경우를 허용하는 테스트 케이스와 맞지 않으므로 I를 E로 고친다. 공식 답의 기호 배열에서 (나)는 R이며, 원인 사이에 있는 (다)의 M도 R로 고친다.

④ X: (가)의 I는 두 직원 조건이 모두 거짓인 경우를 배제하므로 E로 고친다. 공식 답의 기호 배열에서는 원인 사이의 방향 있는 제한인 (나)와 (다)의 두 M을 모두 R로 고친다. 이러한 기호 교정만으로 두 R과 E의 논리적 충돌이 해결되는 것은 아니다.

NoteBase

오답 포인트

테스트 케이스에 A 또는 B가 나온다는 이유로 (가)를 I로 고르기 쉽다. 그러나 A도 B도 아닌 경우에 출입 불가를 출력하므로 두 직원 조건이 모두 거짓일 수 있다. 또한 점선 화살표의 모양만 보고 M을 고르면 원인 사이의 제한과 결과 사이의 제한을 혼동하게 된다.

다시 나오면 이렇게 판단

서로 동시에 참일 수 없는지와 모두 거짓일 수 있는지를 나누어 본다. 동시 참은 금지하고 모두 거짓은 허용하면 E, 하나 이상 반드시 참이어야 하면 I이다. 방향이 있는 제한은 연결된 노드가 입력 원인인지 출력 결과인지 확인하고, 다른 조건의 성립을 요구하는 R과 다른 결과를 억제하는 M을 구분한다.

암기 포인트

E는 동시 참 금지, I는 모두 거짓 금지, R은 성립 요구, M은 결과 억제이다.

NoteBase

문 15. 다음은 스크럼(scrum) 방식의 진행 절차에 대한 설명이다. 진행 절차의 순서로 가장 적절하게 나열된 것은?

(가) 스프린트 구현 목록을 작성하고 스프린트 개발 시간을 추정한다.
(나) 일일 스크럼 회의를 진행하고 스프린트 현황판 변경 및 소멸 차트를 표시한다.
(다) 실행 가능한 스프린트 결과물을 완성한다.
(라) 요구사항 목록에 우선순위를 매겨 제품 기능 목록을 작성한다.
(마) 스프린트 검토 회의를 수행하고 차기 스프린트 수행 계획을 수립한다.

① (라) → (나) → (가) → (다) → (마)
② (라) → (가) → (나) → (마) → (다)
③ (라) → (가) → (나) → (다) → (마)
④ (라) → (나) → (가) → (마) → (다)

정답 3번

간단한 해설

전체 요구사항의 우선순위와 이번 스프린트 계획을 먼저 정한 뒤 개발과 일일 점검을 진행한다. **완성된 결과물을 검토한 후** 다음 스프린트 준비로 이어진다.

NoteBase

올바른 순서

(라) → (가) → (나) → (다) → (마)

(라) 제품 기능 목록 작성: 요구사항을 정리하고 우선순위를 매겨 제품 백로그를 마련한다. 다음 스프린트에서 무엇을 개발할지 선택하려면 먼저 전체 요구사항과 우선순위가 있어야 한다.

(가) 스프린트 계획: 제품 백로그에서 이번 스프린트에 수행할 작업을 선택하고 스프린트 구현 목록을 작성하며 필요한 작업량과 시간을 추정한다. 개발 중의 진행 현황을 관리하기에 앞서 **이번 반복에서 수행할 범위와 계획**을 정한다.

(나) 스프린트 수행과 점검: 개발을 진행하면서 일일 스크럼으로 목표를 향한 진행 상황과 필요한 조정을 점검한다. 현황판과 소멸 차트는 남은 작업과 진행 상태를 보여 주는 도구로 활용한다.

(다) 실행 가능한 결과물 완성: 계획한 작업을 수행하여 사용 가능한 제품 증분을 완성한다. 작업 현황을 점검하는 것만으로 스프린트의 산출물이 완성되는 것은 아니므로 실제 결과물 확보가 필요하다.

(마) 스프린트 검토와 다음 작업 준비: 완성된 결과물을 이해관계자와 검토하고 피드백을 이후 작업에 반영한다. 제시된 절차에서는 그 결과를 바탕으로 차기 스프린트 수행을 준비한다. **검토할 결과물이 먼저 있어야** 검토와 후속 계획이 의미를 갖는다.

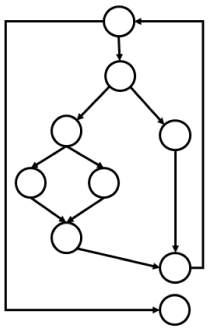
NoteBase

문 16. 다음 프로그램 (가)의 순환 복잡도와 제어흐름 그래프 (나)의 순환 복잡도로 가장 적절한 것은? (단, 코드 실행에 대한 오류는 없는 것으로 함)

(가)

```
bool func(int a, int b, int c)
{
  bool flag = true;
  int i;
  int sum;
  if (a < b)
    flag = false;
  if (a < c)
    flag= false;
  if (a > 0)
    i = 1;
  else {
    i = 0;
    a = -a;
  }
  sum = 0;
  while (i < a) {
    sum += 1;
    i++;
  }
  print(sum);
  return flag;
}
```

(나)



① (가): 4, (나): 4

② (가): 5, (나): 5

③ (가): 4, (나): 5

④ (가): 5, (나): 4

정답 4번

간단한 해설

(가)는 if 세 개와 while 한 개의 **조건 분기 네 개**가 있어 순환 복잡도가 5이다. (나)는 노드 9개와 간선 11개이므로 $11 - 9 + 2 = 4$이다.

NoteBase

계산 과정

공식과 계산 범위: 순환 복잡도는 제어흐름의 선형 독립 경로 수를 나타낸다. 이진 분기만 있는 단일 연결 구조에서는 $V = D + 1$로 계산하며, D는 조건 분기 노드 수이다. 그래프의 노드 수가 N, 간선 수가 E이면 같은 값을 $V = E - N + 2$로 구할 수 있다.

(가)의 조건 수: if (a < b), if (a < c), if (a > 0)의 세 조건과 while (i < a)의 반복 조건 하나가 있다. 따라서 $D = 3 + 1 = 4$이고 V_(가)=4+1=5이다. **else는 앞선 if의 반대 경로이지 별개의 조건이 아니다**. 대입문·출력문·증가 연산도 새로운 분기로 세지 않는다.

(나)의 노드와 간선: 원으로 표시된 노드는 맨 위 시작 노드부터 맨 아래 종료 노드까지 총 9개이다. 직선과 꺾인 선을 모두 포함하여 노드 사이의 방향 연결을 세면 간선은 11개이다. 따라서 V_(나)=11-9+2=4이다. **꺾인 선의 모서리는 노드가 아니며, 되돌아가는 연결도 간선 하나**로 센다.

교차 검산과 결과: (나)에서 두 갈래로 나뉘는 노드는 맨 위 노드, 그 아래 노드, 왼쪽 중간 노드의 총 3개이므로 $D + 1 = 3 + 1 = 4$로도 확인된다. 최종적으로 (가)는 5, (나)는 4이므로 ④와 일치한다. 두 그림은 각각 별도로 계산하며 복잡도에는 시간·길이 같은 물리 단위가 없다.

NoteBase

문 17. 다음은 소프트웨어 아키텍처 평가 방법에 대한 설명이다. 각 설명이 의미하는 평가 방법으로 가장 적절한 것은?

(가) 후보 아키텍처가 품질 속성을 만족하는지 판단하고 상호 간에 어떻게 충돌하고 작용하는지를 분석하여 아키텍처 설계에 대한 타협점을 찾는 평가 방법이다.
(나) 시스템 아키텍처 품질에서 얻을 수 있는 이득에 대해 투자 대비 효과를 경제적인 측면에서 평가하여 수익이 최대가 되는 소프트웨어 아키텍처를 선정하는 평가 방법이다.

① (가): ATAM, (나): CBAM
② (가): ATAM, (나): SAAM
③ (가): SAAM, (나): CBAM
④ (가): SAAM, (나): ATAM

정답 1번

간단한 해설

ATAM은 **품질 속성 간의 타협관계**를, CBAM은 **비용과 편익에 따른 경제성**을 분석한다. 두 설명은 각각 기술적 상충 분석과 투자 효과 평가로 구별된다.

대응 관계 해설

(가) ATAM: Architecture Tradeoff Analysis Method는 성능·보안·변경 용이성 등 여러 품질 속성을 대상으로 설계 결정의 영향과 **품질 속성 사이의 상충관계**를 분석한다. 한 속성을 개선하는 선택이 다른 속성을 악화시킬 수 있으므로, 후보 아키텍처의 위험과 타협점을 파악하는 설명에 대응한다.

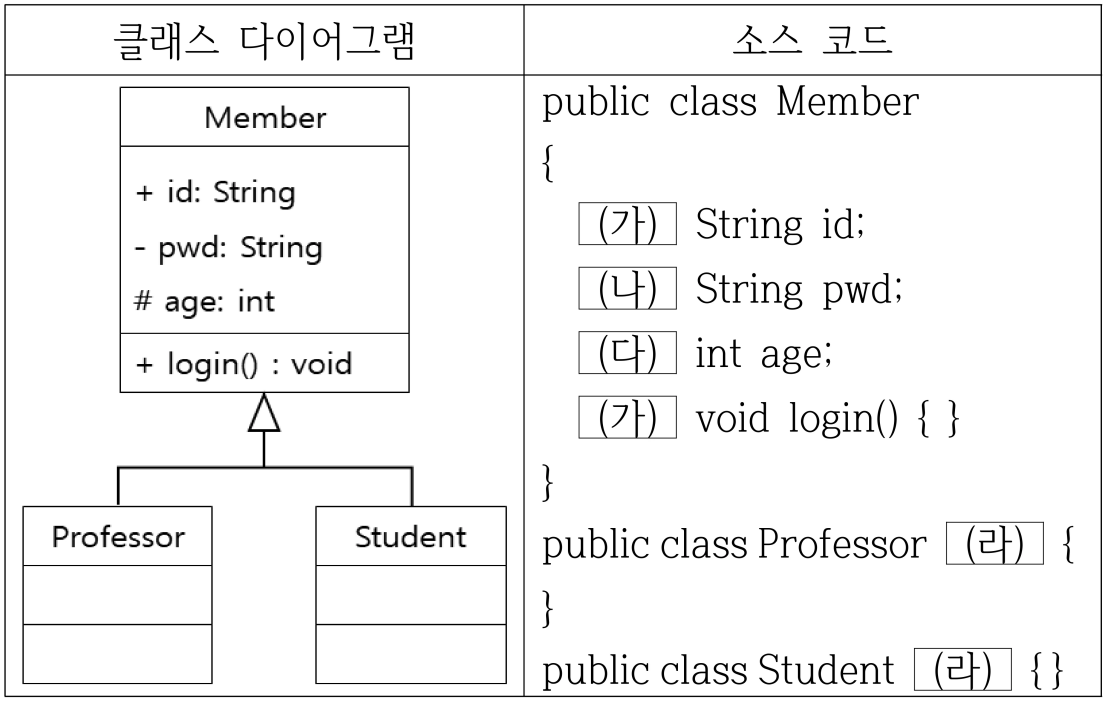
(나) CBAM: Cost Benefit Analysis Method는 아키텍처 대안에 필요한 비용과 그 대안이 제공하는 편익을 함께 평가한다. 기술적으로 가능한 대안 중에서도 **투자 대비 경제적 효과**를 고려하여 자원을 배분하고 선택한다는 점이 핵심이다.

SAAM과의 구별: Software Architecture Analysis Method는 시나리오를 활용해 아키텍처를 분석하는 방법으로, 특히 변경 시나리오의 영향과 변경 용이성을 살핀다. 이 문항의 명시적인 단서인 여러 품질 속성 간의 타협 분석은 ATAM, 비용과 편익의 경제적 비교는 CBAM으로 연결한다.

따라서 (가) ATAM, (나) CBAM의 조합인 ①에 해당한다.

NoteBase

문 18. 다음은 클래스 다이어그램을 자바 코드로 표현한 것이다. (가)~(라)에 가장 적절한 것은? (단, 코드 실행에 대한 오류는 없는 것으로 함)



① (가): public, (나): protected, (다): private, (라): extends Member

② (가): public, (나): private, (다): protected, (라): extends Member

③ (가): public, (나): protected, (다): private, (라): implements Member

④ (가): public, (나): private, (다): protected, (라): implements Member

정답 2번

간단한 해설

UML의 접근 기호 +, -, #는 각각 public, private, protected에 대응한다. 실선의 빈 삼각형이 Member를 가리키므로 **Member를 상속하는 extends**를 사용한다.

NoteBase

빈칸별 해설

(가) public: id 속성과 login() 연산 앞의 기호는 모두 +이다. UML에서 **+는 공개 접근을 뜻하는 public**이므로 두 곳에 공통으로 public을 넣는다. 클래스 이름 앞의 public과는 별개로 각 멤버의 접근 수준을 대응시킨 것이다.

(나) private: pwd 속성 앞에는 -가 있다. UML의 **-는 private**에 대응한다. 따라서 pwd는 외부 코드가 직접 접근하도록 공개하는 필드가 아니라 비공개 필드로 선언한다. protected와 자리를 바꾸면 다이어그램의 접근 수준과 달라진다.

(다) protected: age 속성 앞의 #는 protected를 나타낸다. 자바에서는 같은 패키지에서 접근할 수 있으며, 다른 패키지의 하위 클래스에서도 상속과 관련된 접근 규칙을 만족하는 범위에서 접근할 수 있다. **#와 -의 의미를 뒤바꾸지 않는다**.

(라) extends Member: Professor와 Student에서 Member 쪽을 향하는 실선과 속이 빈 삼각형은 일반화, 즉 클래스 상속을 나타낸다. **빈 삼각형이 가리키는 Member가 상위 클래스**이므로 두 하위 클래스의 선언에 extends Member를 넣는다. implements는 클래스가 인터페이스를 구현할 때 사용하는 표현으로, 그림의 클래스 상속과 구별한다.

따라서 (가) public, (나) private, (다) protected, (라) extends Member의 조합이므로 ②와 일치한다.

NoteBase

문 19. SQuaRE(Software Quality and Requirement Evaluation)라 불리는 ISO/IEC 25000의 각 부문에 대한 설명으로 가장 적절한 것은?

① 2501n 부문 - SQuaRE에 대한 개요 및 전체에 대한 계획과 관리

② 2502n 부문 - 소프트웨어 품질 모델과 품질 사용에 대한 안내

③ 2503n 부문 - 내/외부 메트릭스, 사용 중 품질 메트릭스 및 평가 모듈의 문서화

④ 2504n 부문 - 개발자, 구매자, 평가자 관점의 품질 평가 프로세스

정답 4번

간단한 해설

ISO/IEC 2504n은 **품질평가 부문**이다. 개발자·구매자·평가자가 품질을 평가하는 절차와 지침을 다룬다.

선지별 해설

① X: SQuaRE의 전체 개요와 계획·관리는 2500n 품질관리 부문이다. **2501n은 품질모델 부문**으로, 제품 품질과 사용 품질 등의 모델 및 활용 지침을 다룬다.

② X: 품질모델과 그 활용 안내는 2501n에 속한다. **2502n은 품질측정 부문**으로, 품질 측정의 기준 모형과 측정량·메트릭 및 적용 지침을 다룬다.

③ X: 2503n은 **품질요구사항 부문**으로 필요한 품질을 요구사항으로 명세하는 데 초점을 둔다. 내부·외부 품질 및 사용 품질의 메트릭은 품질측정과 관련되며, 평가 모듈의 문서화 지원은 품질평가 부문에서 다루므로 선지의 내용을 2503n으로 묶을 수 없다.

④ O: 2504n은 품질평가 부문으로, 개발자·구매자·독립 평가자 등이 제품의 품질을 평가할 때 적용하는 요구사항·권고·지침과 평가 프로세스를 제공한다.

따라서 부문과 설명을 적절하게 연결한 것은 ④이다.

NoteBase

문 20. 다음 설명에 해당하는 디자인 패턴으로 가장 적절한 것은?

◯ 애플리케이션이 대량의 객체를 사용해야 하거나 객체 수가 너무 많아져 저장 비용이 높아질 경우에 유용하며, 애플리케이션이 객체 식별자에 비중을 두지 않는 경우에 사용한다.
◯ 인스턴스를 필요한 만큼 생성하는 대신 동일한 인스턴스를 공유하여 메모리 사용량을 줄이는 목적을 가진 것으로, 구조적 패턴에 속한다.

① 프록시 패턴(Proxy pattern)
② 프로토타입 패턴(Prototype pattern)
③ 플라이웨이트 패턴(Flyweight pattern)
④ 컴포지트 패턴(Composite pattern)

정답 3번

간단한 해설

같은 데이터를 가진 객체를 매번 새로 만들지 않고 **공유하여 메모리를 절약**하는 것이 플라이웨이트 패턴이다. 각 사용처에서 객체의 고유한 식별보다 공유할 수 있는 상태가 중요할 때 적용한다.

공통 해설

플라이웨이트 패턴(Flyweight pattern)은 다수의 객체가 공통으로 사용하는 상태를 하나의 객체에 모아 공유함으로써 메모리 사용량을 줄이는 구조적 디자인 패턴이다. 공유 객체에는 여러 사용처에서 동일하게 유지되는 상태를 두고, 위치처럼 사용처마다 달라지는 상태는 외부에서 별도로 관리하여 사용할 때 전달한다. 핵심은 **객체를 매번 생성하는 대신 공통 상태를 공유**하는 데 있다.

예를 들어 같은 모양의 나무를 화면에 많이 표시한다면, 나무마다 동일한 모양 데이터를 중복 저장하지 않고 하나의 모양 데이터를 공유할 수 있다. 각 나무의 위치는 별도로 관리하므로, 위치가 달라도 공통 모양을 저장하는 데 필요한 메모리는 줄어든다.

지문의 대량 객체로 인한 저장 비용과 동일 인스턴스 공유는 이러한 메모리 절약 방식에 해당한다. 또한 공유하면 여러 사용처가 같은 객체를 참조하므로, **각 객체가 서로 다른 식별자를 가져야 하는지가 중요하지 않은 경우**에 적합하다. 이 조건들을 함께 만족하는 개념이 플라이웨이트 패턴이다.

NoteBase

문 21. 소프트웨어 형상 관리(configuration management) 절차에서 형상 항목 선정이나 베이스라인 기준 선정을 수행하는 단계로 가장 적절한 것은?

① 형상 감사(audit)

② 형상 통제(control)

③ 형상 식별(identification)

④ 형상 상태 보고(status reporting)

정답 3번

간단한 해설

형상 식별은 **무엇을 관리할지와 기준 상태를 정하는 활동**이다. 형상 항목 선정과 베이스라인 기준 선정이라는 단서로 구별한다.

대응 관계 해설

형상 식별과의 대응: 소스 코드·문서·설정 등 무엇을 형상 항목으로 관리할지 정하고, 각 항목을 구분할 식별 체계와 베이스라인을 정하는 활동이다. **관리 대상과 기준 상태를 선정**한다는 문항의 단서가 형상 식별에 해당한다.

형상 통제와의 구분: 이미 식별된 형상 항목과 베이스라인에 대한 변경 요청을 검토·승인하고 승인된 변경이 반영되도록 관리하는 활동이다. 대상을 처음 정하는 활동과 변경을 통제하는 활동은 구별된다.

형상 감사와의 구분: 형상 항목이 정해진 요구사항·명세와 일치하고 승인된 변경이 적절히 반영되었는지 확인한다. 선정 자체보다는 결과와 절차의 일치 여부를 검증하는 데 초점을 둔다.

형상 상태 보고와의 구분: 형상 항목의 버전, 변경 요청과 승인·반영 상태 등 관리 정보를 기록하고 보고한다. **현재 상태와 변경 이력을 추적 가능하게 제공**하는 역할이다.

따라서 형상 항목과 베이스라인 기준을 선정하는 활동은 ③ 형상 식별이다.

NoteBase

문 22. 다음 설명과 관련한 SOLID 설계원칙으로 가장 적절한 것은?

◯ 새로운 기능이 필요할 때 기존 코드를 수정하지 않고 새로운 코드를 추가로 작성하여 확장할 수 있어야 한다.
◯ 변하지 않는 것과 확장될 수 있는 것을 구분하여 변하지 않는 부분을 추상 부모 클래스로 만들고 확장 가능한 것을 자식 클래스로 설계한다.
◯ 수월한 유지보수와 재사용성을 위한 설계원칙이다.

① 단일 책임의 원리
② 개방 폐쇄의 원리
③ 인터페이스 분리의 원리
④ 리스코프 교체의 원리

정답 2번

간단한 해설

기존 코드를 계속 수정하기보다 새로운 구현을 추가해 기능을 확장한다는 설명은 **개방 폐쇄 원리**에 해당한다.
공통 추상화를 유지하면서 변하는 부분을 확장하는 것이 핵심이다.

NoteBase

대응 관계 해설

기능 추가 방식과의 대응: 개방 폐쇄 원리는 소프트웨어 구성 요소가 **확장에는 열려 있고 수정에는 닫혀 있어야 한다**는 원리이다. 예상되는 기능 변화를 기존 코드의 반복 수정 대신 새로운 구현의 추가로 수용하라는 첫 번째 설명에 해당한다.

추상화와 상속의 역할: 변하지 않는 공통 계약을 추상 부모 클래스에 두고 달라질 동작을 자식 클래스가 구현하면, 기존 코드는 공통 계약을 유지하면서 새로운 자식 클래스를 통해 확장할 수 있다. 이는 개방 폐쇄 원리를 구현하는 방법이며, 상속만이 유일한 방법이라는 뜻은 아니다.

유지보수·재사용성과의 관계: 안정된 기존 코드의 변경을 줄이면 이미 검증한 부분에 오류를 유발할 위험과 변경의 파급을 줄일 수 있다. 공통 추상화를 재사용하고 변하는 구현을 추가하는 방식으로 유지보수와 확장을 돕는다.

다른 원칙과의 구분: 단일 책임 원리는 하나의 변경 이유에 집중하도록 책임을 나누는 원리이고, 인터페이스 분리 원리는 사용하지 않는 메소드에 의존하지 않도록 인터페이스를 나누는 원리이다. 리스코프 교체 원리는 하위 타입이 상위 타입의 계약을 유지하며 대체 가능해야 한다는 원리이다. 지문의 중심은 대체 가능성 자체가 아니라 **기존 코드 수정 없이 기능을 확장하는 구조**이다.

따라서 세 설명에 가장 적절한 원칙은 ② 개방 폐쇄의 원리이다.

NoteBase

문 23. 다음은 UP(Unified Process) 모델의 개발 단계에 대한 설명이다. 이 설명에 해당하는 개발 단계로 가장 적절한 것은?

◯ 핵심 아키텍처를 확정하고 반복적으로 구현하여 시스템의 프레임워크를 확립한다.
◯ 생성된 첫 번째 프로토타입을 다음 단계의 베이스라인으로 활용하고 프로젝트 수행을 방해하는 중대한 위험 요소들을 찾아 위험을 축소하거나 제거한다.

① 도입 단계(inception phase)
② 전이 단계(transition phase)
③ 정련 단계(elaboration phase)
④ 구축 단계(construction phase)

정답 3번

간단한 해설

핵심 아키텍처를 실제로 구현해 검증하고 **중대한 위험을 줄여 다음 개발의 기준을 마련**하는 것은 정련 단계의 특징이다. 이때 확보한 아키텍처와 프로토타입이 이후 본격적인 기능 개발의 토대가 된다.

공통 해설

정련 단계(elaboration phase)는 **핵심 아키텍처를 확립하고 주요 위험을 해소**하여 이후 개발을 안정적으로 진행할 기반을 마련하는 단계이다. 아키텍처는 시스템의 주요 구성 요소와 그 관계를 정한 기본 구조이다. 이 구조가 실제로 작동하는지 반복적인 구현과 시험으로 확인하므로, 정련 단계에서도 프로토타입을 구현한다.

제시문의 '시스템의 프레임워크를 확립한다'는 이러한 기본 구조를 확정한다는 뜻이다. 또 프로토타입을 다음 단계의 베이스라인으로 활용한다는 것은 검증된 구현 결과를 이후 개발의 기준으로 삼는다는 의미이다. 여기서 구현의 목적은 **전체 기능의 완성보다 아키텍처의 타당성 확인과 위험 축소**에 있으므로 정련 단계에 해당한다.

NoteBase

문 24. MTTF는 400시간, MTTR은 100시간인 시스템에서, MTTR은 유지한 채 가용성을 90%로 개선하고자 한다. 이때 필요한 MTTF는?

① 600시간

② 800시간

③ 900시간

④ 1,000시간

정답 3번

간단한 해설

MTTR을 100시간으로 고정하면 목표 가용성은 $\frac{x}{x+100} = 0.9$이다. 이를 풀면 **필요한 MTTF는 900시간**이며, 증가량 500시간과 구별해야 한다.

계산 과정

변수와 공식: MTTF는 고장 전까지의 평균 정상 작동 시간, MTTR은 고장 후 복구에 걸리는 평균 시간이다. 정상 작동과 복구가 반복되는 시스템에서 가용성은 $A = \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}}$이다. 분모에는 **정상 작동 시간과 복구 시간을 모두 포함**한다.

현재 상태와 목표: 현재 가용성은 $\frac{400}{400+100} = 0.8 = 80\%$이다. 목표는 90%이고 MTTR은 100시간으로 유지한다. 필요한 새 MTTF를 x시간이라 두면 $0.9 = \frac{x}{x+100}$이다.

방정식 풀이: 양변에 $x + 100$을 곱하면 $0.9x + 90 = x$이므로 $0.1x = 90$, 따라서 $x = 900$시간이다. 같은 결과를 $x = \frac{A}{1-A}\text{MTTR} = \frac{0.9}{0.1} \times 100$으로도 구할 수 있다.

검산과 답 연결: $\frac{900}{900+100} = 0.9$이므로 목표 가용성을 만족하며 ③과 일치한다. 필요한 MTTF 자체는 900시간이고, 기존 400시간에서의 **증가량은 500시간**이다. 목표 90%와 90시간을 혼동하거나 증가량을 답으로 고르지 않는다.

NoteBase

문 25. 다음 요구사항 중 기능적 요구사항만을 모두 고른 것으로 가장 적절한 것은?

(가) 관리자는 규정 위반 회원의 계정을 정지시킬 수 있다.
(나) 시스템은 연중무휴 24시간 운영되어야 한다.
(다) 주문 완료 시 고객에게 전자 영수증을 이메일로 발송한다.
(라) 데이터베이스 접근은 표준 SQL만 사용해야 한다.

① (가), (나)
② (가), (다)
③ (나), (라)
④ (다), (라)

정답 2번

간단한 해설

계정 정지와 영수증 발송은 **시스템이 수행할 구체적인 기능**이다. 운영 시간과 사용 기술의 제한은 기능 자체가 아닌 비기능적 조건이다.

보기별 해설

(가) 기능적 요구사항: 관리자가 규정 위반 회원의 계정을 정지하는 구체적인 기능을 명시한다. 시스템이 어떤 사용자 행위를 지원하고 어떤 상태 변경을 수행해야 하는지에 관한 요구이다.

(나) 비기능적 요구사항: 연중무휴 24시간 운영은 개별 업무 기능이 아니라 **서비스의 가용성과 운영 시간**에 관한 품질·운영 조건이다.

(다) 기능적 요구사항: 주문이 완료되면 전자 영수증을 이메일로 발송한다는 입력 사건과 시스템의 구체적인 처리·출력을 명시한다. 주문 후 자동 발송 서비스를 제공하라는 기능 요구이다.

(라) 비기능적 요구사항: 데이터베이스에 접근하는 기술을 표준 SQL로 제한하는 **구현 기술상의 제약**이다. 사용자가 수행할 업무 기능이 아니라 그 기능을 어떤 방식으로 구현할지의 조건을 정한다.

따라서 기능적 요구사항은 (가), (다)이며 ②에 해당한다.

NoteBase

문 26. 다음 C 함수의 응집도 유형과 이를 개선하기 위한 리팩토링 기법으로 가장 적절한 것은? (단, 코드 실행에 대한 오류는 없는 것으로 함)

```
int Calc(int type, int n) {
  switch (type) {
    case 1: return n * 200;    // 미성년 공제
    case 2: return n * 150;    // 경로 공제
                               ...
    case 9: return n * 10;     // 일반 공제
  }
  return 0;
}
```

① 기능적 응집 - 메소드 추출

② 논리적 응집 - 조건문을 다형성으로 대체

③ 순차적 응집 - 클래스 추출

④ 우연적 응집 - 매직 넘버의 상수화

정답 2번

간단한 해설

유사한 공제 기능을 모으고 type 값으로 하나를 선택하므로 논리적 응집이다. **유형별 계산 동작을 분리하여 다형적으로 호출**하면 긴 조건 분기에 대한 의존을 줄일 수 있다.

NoteBase

핵심 풀이

코드의 핵심은 switch(type)이다. 미성년 공제, 경로 공제, 일반 공제처럼 유사한 범주의 여러 계산을 한 함수에 모은 뒤 **외부에서 전달된 type으로 실행할 기능을 선택**한다. 이런 선택형 묶음은 논리적 응집에 해당한다. 단순히 모두 공제라는 이름을 가진다고 해서 모든 요소가 하나의 기능을 수행하는 기능적 응집으로 보지는 않는다.

개선 방향은 조건문을 다형성으로 대체하는 것이다. 공제 유형마다 계산 동작을 분리하고 공통 연산을 통해 호출하면, 호출부가 긴 switch를 직접 나누는 대신 선택된 유형의 동작이 실행된다. 객체지향 설계로 옮긴다면 각 유형이 공통 메소드를 구현하도록 할 수 있다. **유형별 분기 책임을 유형별 구현으로 분산**하는 방식이다.

순차적 응집은 앞 처리의 출력이 뒤 처리의 입력으로 이어지는 관계인데, 이 코드의 각 case는 계산 결과를 곧바로 반환하므로 그러한 처리 사슬이 아니다. 우연적 응집처럼 서로 무관한 기능을 아무렇게나 모은 것도 아니며, 공제 계산이라는 논리적 공통성이 있다.

메소드 추출은 코드 덩어리를 별도 연산으로 분리하는 기법이고, 클래스 추출은 여러 책임을 가진 클래스에서 별도의 클래스를 나누는 기법이다. 매직 넘버를 상수로 바꾸면 200·150·10의 의미를 명확히 할 수 있지만, type에 의존하는 선택 구조 자체는 그대로 남는다. 따라서 제시된 짝 중 논리적 응집과 조건문을 다형성으로 대체를 연결한 ②가 가장 적절하다.

제시 코드는 C이므로 클래스 상속이나 가상 메소드를 그대로 문법에 추가할 수는 없다. 실제 적용에서는 함수 포인터 등으로 유형별 동작을 분리하거나 객체지향 환경으로 구조를 옮기는 구현 수단이 필요하다. 리팩토링 과정에서는 각 공제 계산과 해당 유형이 없을 때 0을 반환하는 기존 동작을 보존해야 한다.

NoteBase

문 27. 컴포넌트 기반 개발(CBD)에 대한 설명으로 가장 적절하지 않은 것은?

① 컴포넌트는 독립적으로 배포 가능한 소프트웨어 단위이다.

② 컴포넌트는 명세된 인터페이스를 통해 기능을 제공한다.

③ 컴포넌트를 재사용하려면 그 내부 소스 코드를 수정해야 한다.

④ 검증된 컴포넌트의 조립을 통해 개발 기간을 단축할 수 있다.

정답 3번

간단한 해설

컴포넌트는 내부 구현을 감추고 명세된 인터페이스로 조립·재사용한다. **내부 소스 코드 수정은 필수 조건이 아니다**.

선지별 해설

① O: 컴포넌트는 기능과 구현을 캡슐화한 독립적인 배포·조립 단위이다. 독립 배포가 가능하다는 것은 다른 컴포넌트나 실행 환경에 대한 의존성이 전혀 없다는 뜻이 아니라, 명시된 조건 아래에서 별도 단위로 배포하고 연결할 수 있다는 뜻이다.

② O: 컴포넌트는 **명세된 인터페이스**를 통해 기능을 제공하고 필요한 서비스를 요구한다. 이용자는 내부 구현의 세부 사항보다 인터페이스의 기능과 사용 조건을 바탕으로 컴포넌트를 결합한다.

③ X: 컴포넌트 기반 재사용은 내부 구현을 캡슐화하고 외부 인터페이스를 이용해 조립하는 것이 핵심이다. 따라서 **내부 소스 코드 수정은 재사용의 필수 조건이 아니다**. 필요에 따라 설정이나 어댑터를 이용할 수 있으며, 소스 코드를 제공받지 않아도 재사용 가능한 컴포넌트가 있다.

④ O: 이미 구현·검증된 컴포넌트를 활용하면 공통 기능을 새로 만드는 작업을 줄여 개발 기간을 단축할 수 있다. 다만 각 컴포넌트가 검증되었다고 해서 조립된 시스템의 인터페이스·통합 테스트까지 생략할 수 있는 것은 아니다.

따라서 재사용에 내부 소스 코드 수정이 반드시 필요하다고 한 ③이 가장 적절하지 않다.

NoteBase

문 28. UML 다이어그램에 대한 다음 설명 중 가장 적절하지 않은 것은?

① 순차 다이어그램은 메시지를 이용해 객체 간 상호작용을 나타내며, 링크(link)를 사용해 객체 간의 관계를 표현한다.

② 활동 다이어그램은 업무의 흐름에서 동작과 같은 행위가 수행되는 동안의 활동을 다이어그램으로 나타낸다.

③ 상태 다이어그램은 이벤트 발생이나 시간 흐름에 따라 바뀌는 객체의 상태 변화를 나타낸다.

④ 컴포넌트 다이어그램은 구현 관점에서 정적 모델링을 할 때 사용하는 것으로 어떤 실행 모듈이 존재하고 이들이 서로 어떤 연관성이 있는지의 종속 관계를 나타낸다.

정답 1번

간단한 해설

순차 다이어그램은 **생명선과 시간 순서에 따른 메시지**로 상호작용을 표현한다. 객체 간 링크를 중심으로 관계를 나타내는 특징은 통신 다이어그램과 구별해야 한다.

선지별 해설

① X: 순차 다이어그램은 객체·참여자의 생명선을 배치하고 그 사이의 메시지를 시간 흐름에 따라 나타낸다. 메시지로 상호작용을 표현한다는 부분은 맞지만, **객체 사이의 링크를 중심으로 관계를 배치**하는 설명은 통신 다이어그램의 특징이다. 순차 다이어그램의 핵심은 생명선과 메시지의 시간적 순서이다.

② O: 활동 다이어그램은 동작과 제어 흐름·객체 흐름을 통해 업무나 처리 절차를 표현한다. 조건 분기, 병렬 실행과 합류 등을 사용하여 여러 활동이 어떻게 이어지는지 보여 준다.

③ O: 상태 다이어그램은 객체가 취할 수 있는 상태와 상태 사이의 전이를 나타낸다. 이벤트나 시간 조건이 충족되었을 때 어떤 상태로 이동하는지, 전이 조건과 수행 행위가 무엇인지 표현할 수 있다.

④ O: 컴포넌트 다이어그램은 시스템을 구성하는 컴포넌트와 제공·요구 인터페이스 및 의존 관계를 보여 주는 구조 다이어그램이다. 구현을 구성하는 모듈들의 **정적인 구성과 종속 관계**를 파악하는 데 적합하며, 실행 시 메시지의 시간 순서를 그리는 도구는 아니다.

따라서 순차 다이어그램에 통신 다이어그램의 링크 중심 특징을 섞어 설명한 ①이 가장 적절하지 않다.

NoteBase

문 29. 다음 소프트웨어 개발 프로세스 모델과 설명을 바르게 연결한 것은?

(가) 프로토타입 모델
(나) 폭포수 모델
(다) 스크럼

A. 소프트웨어 개발을 순차적인 단계로 나누고, 각 단계를 마친 후 다음 단계로 진행한다.
B. 짧은 개발 주기를 반복하며 점진적으로 제품을 개발한다.
C. 완전한 소프트웨어를 만들기 전에, 사용자의 요구를 반영한 모형을 먼저 제작한다.

① (가): A, (나): B, (다): C
② (가): C, (나): A, (다): B
③ (가): C, (나): B, (다): A
④ (가): B, (나): A, (다): C

정답 2번

간단한 해설

모형 제작을 통한 요구 확인은 프로토타입, 단계별 순차 진행은 폭포수, 짧은 주기의 반복·점진 개발은 스크럼에 대응한다. **모형·순차 단계·반복 증분**이라는 핵심 표현을 구분한다.

대응 관계 해설

(가) 프로토타입 모델 → C: 완전한 제품을 개발하기 전에 요구를 반영한 모형을 제작한다. 사용자가 모형을 확인하고 피드백을 주면 개발자는 요구를 구체화하거나 수정할 수 있다. **모형으로 요구를 확인하는 것**이 구분 기준이며, 반드시 처음 만든 모형을 최종 제품으로 사용하는 것은 아니다.

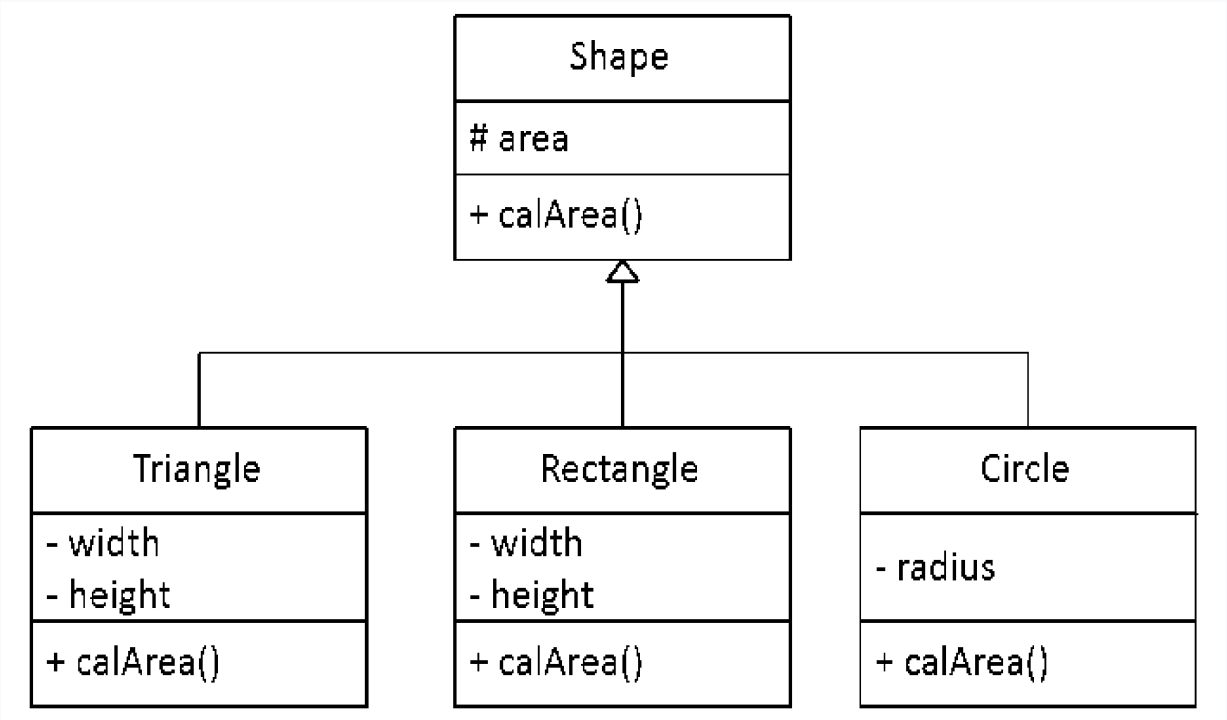
(나) 폭포수 모델 → A: 요구사항 분석, 설계, 구현, 테스트 등의 개발 단계를 정해진 흐름에 따라 진행한다. **한 단계의 산출물을 바탕으로 다음 단계로 넘어가는 순차성**이 핵심이다. 실제 프로젝트에서 수정이 발생할 수 없다는 뜻으로 과장하지 않는다.

(다) 스크럼 → B: 짧은 개발 주기인 스프린트를 반복하며 사용할 수 있는 제품의 증분을 만든다. 결과를 점검하고 피드백을 다음 작업에 반영하므로 **반복적·점진적인 제품 개발**에 해당한다. 단지 모형으로 요구를 확인하는 C와 달리 제품의 가치를 계속 완성해 가는 데 초점이 있다.

따라서 (가) C, (나) A, (다) B로 연결되므로 ②와 일치한다.

NoteBase

문 30. 다음 클래스 다이어그램에 적용된 설계 개념으로 가장 적절하지 않은 것은?



① 상속(inheritance)

② 오버라이딩(overriding)

③ 오버로딩(overloading)

④ 다형성(polymorphism)

정답 3번

간단한 해설

하위 도형들이 같은 calArea()를 재정의하므로 상속·오버라이딩·다형성에 해당한다. **매개변수 목록이 다른 동명 연산**은 없으므로 오버로딩의 근거는 제시되지 않았다.

NoteBase

개념 구분

도식의 상속 관계: Triangle, Rectangle, Circle에서 Shape를 향하는 실선과 속이 빈 삼각형은 일반화를 나타낸다. 세 도형 클래스는 공통 상위 클래스 Shape의 특성을 이어받는 하위 클래스이므로 상속 개념이 적용된다.

같은 연산의 재정의: Shape와 세 하위 클래스에 모두 매개변수가 없는 calArea()가 표시되어 있다. 하위 클래스에서 자기 도형의 넓이에 맞게 같은 연산을 재정의하는 구조이므로 **상속 관계에서 동일한 연산을 재정의하는 오버라이딩**에 해당한다.

다형성과의 연결: 공통 Shape 관점에서 calArea()를 호출하더라도 실제 도형이 삼각형·사각형·원인지에 따라 각 하위 클래스의 계산 동작을 사용하도록 설계할 수 있다. **동일한 연산 호출에 서로 다른 구현이 대응**하는 다형성의 구조이다.

오버로딩과의 구별: 오버로딩은 같은 이름의 연산을 매개변수의 개수나 타입 등이 다르게 정의하는 것이다. 그림의 calArea()는 모두 매개변수가 없는 동일한 형태이며, 매개변수 목록이 다른 동명 연산은 제시되지 않았다. 이름이 여러 클래스에 반복된다는 사실만으로 오버로딩이 되는 것은 아니다. 따라서 적용된 개념으로 가장 적절하지 않은 것은 ③ 오버로딩이다.

NoteBase

문 31. 소프트웨어 개발 프로젝트를 수행하는 데 다음과 같은 조건이 적용될 때, 프로젝트 완료에 필요한 최소 소요 기간은?

◯ 추정 소스코드 라인 수 : 200,000 LOC
◯ 개발자 1인당 월 평균 생산성 : 1,200 LOC/MM
◯ 개발자 수 : 5명
◯ 프로젝트 복잡도 계수 : 1.2

① 28개월
② 34개월
③ 40개월
④ 46개월

정답 3번

간단한 해설

소스 코드 규모를 생산성으로 나누고 복잡도를 반영하면 **총 200인월**이 필요하다. 이를 개발자 5명으로 나누면 40개월이다.

계산 과정

기본 개발 노력: 소스 코드 규모가 200,000 LOC이고 생산성이 1,200 LOC/MM이므로, 기본 노력은 $200{,}000 \div 1{,}200 = 166\frac{2}{3}$ MM이다. MM은 개발자 한 명이 한 달 일하는 양인 인월을 뜻하며, 곧바로 프로젝트의 경과 개월 수를 뜻하지 않는다.

복잡도 보정: 프로젝트 복잡도 계수 1.2를 필요한 노력에 곱하면 $166\frac{2}{3} \times 1.2 = 200$ MM이다. 즉 제시된 조건에서 **복잡도를 반영한 총 작업량은 200인월**이다.

개발자 수에 따른 기간: 개발자 5명이 동일한 생산성으로 병렬 작업한다고 보는 문제의 단순 산정 모형에서, 소요 기간은 200 MM ÷ 5명 = 40개월이다. 일정을 구할 때에는 총 인월을 투입 인원으로 나누어야 한다.

따라서 필요한 최소 소요 기간은 ③ 40개월이다.

NoteBase

문 32. 요구분석명세서의 품질 검증을 위한 방법으로 가장 적절하지 않은 것은?

① 요구사항 구현을 위한 구조 및 알고리즘 명세를 구체적으로 작성하였는지 확인한다.

② 기능 요구사항뿐 아니라 성능, 제약사항 등의 비기능 요구사항도 빠짐없이 작성하였는지 확인한다.

③ 서술된 명세서의 내용이 모호하지 않고 모든 참여자가 명확히 이해할 수 있도록 작성하였는지 확인한다.

④ 요구사항 간 모순되거나 충돌되는 점이 없도록 작성하였는지 확인한다.

정답 1번

간단한 해설

요구사항은 무엇을 해야 하는지를, 설계는 그것을 어떻게 구현할지를 구체화한다. **상세 구조·알고리즘의 작성 여부**를 요구분석명세서의 일반적인 품질 기준으로 삼는 것은 적절하지 않다.

선지별 해설

① X: 요구분석명세서는 시스템이 제공해야 할 기능과 만족해야 할 조건, 즉 무엇을 해야 하는지를 명시한다. 구현 구조와 상세 알고리즘을 빠짐없이 구체화했는지는 주로 설계 산출물의 검토 대상이다. 따라서 요구분석명세서의 일반적인 품질 기준을 **상세 구현 설계의 작성 여부로 대체**한 점이 적절하지 않다. 필수적인 설계 제약이 요구사항에 포함될 수는 있지만, 모든 구조와 알고리즘을 상세히 정해야 한다는 뜻은 아니다.

② O: 기능뿐 아니라 성능, 제약 등 필요한 비기능 요구사항도 명세되었는지 살피는 것은 **완전성**의 점검이다. 필요한 기능만 적고 응답 시간이나 적용 제약을 누락했다면 이해관계자의 요구를 충분히 담았다고 보기 어렵다.

③ O: 동일한 문장을 참여자마다 다르게 해석하지 않도록 용어와 조건을 명확히 하는 것은 **비모호성**의 점검이다. 예를 들어 구체적 기준 없이 빠르게 처리한다고만 쓰면 해석 차이가 생길 수 있으므로, 의미를 일관되게 이해할 수 있도록 명세해야 한다.

④ O: 요구사항 사이에 동시에 만족할 수 없는 조건이나 상충하는 규칙이 있는지 확인하는 것은 **일관성**의 점검이다. 개별 문장이 분명하더라도 요구사항끼리 충돌할 수 있으므로 비모호성과 구별하여 검토해야 한다.

따라서 요구분석명세서의 품질 검증 방법으로 가장 적절하지 않은 것은 ①이다.

NoteBase

문 33. CPM으로 일정계획을 수립한 결과가 다음과 같을 때, 가장 적절하지 않은 것은? (단, 작업은 FS(Finish to Start) 관계임)

작업	A	B	C	D	E	F	G	H
선행 작업	-	-	A, B	C	C	D	E	F, G
작업별 빠른 시작 시간	0	0	4	7	7	8	11	13
작업별 늦은 시작 시간	2	0	4	7	8	8	12	13

① 여유 시간이 가장 많은 작업은 A이다.

② D 작업은 임계 경로에 포함된다.

③ E 작업은 임계 경로에 포함되지 않는다.

④ G 작업에 필요한 최소 시간은 2이다.

정답 4번

간단한 해설

G는 늦어도 12에 시작하여 후속 작업 H의 시작 시각 13 전에 끝나야 하므로 **소요 시간은 1**이다. 빠른 시작 시각끼리의 차이 2에는 여유 시간이 포함된다.

NoteBase

계산 과정

여유 시간 계산: 빠른 시작 시간을 ES, 늦은 시작 시간을 LS라고 하면 총여유 시간은 LS−ES이다. 표에서 A는 2−0=2, B는 0, C는 0, D는 0, E는 8−7=1, F는 0, G는 12−11=1, H는 0이다. 따라서 **총여유 시간이 가장 큰 작업은 A**이다.

임계경로 판정: 여유 시간이 0인 작업을 선행 관계에 따라 연결하면 B→C→D→F→H이다. D는 이 임계경로에 포함되지만, E는 여유 시간이 1이므로 포함되지 않는다. 따라서 앞의 세 선지 내용은 계산 결과와 일치한다.

G의 소요 시간 계산: G의 직접 후속 작업은 H이며 FS 관계이므로, 역방향 계산에서 G의 늦은 완료 시간은 H의 늦은 시작 시간 13이다. G의 늦은 시작 시간은 12이므로 **G의 작업 소요 시간은 13−12=1**이다.

시간 차이의 해석: H의 빠른 시작 13에서 G의 빠른 시작 11을 빼면 2이지만, 이것을 G의 작업 시간으로 보면 안 된다. G는 시각 11에 시작해 12에 끝날 수 있고 H는 다른 선행 작업 F도 끝날 때까지 기다려야 한다. 이 차이에는 G 경로의 여유 1이 포함된다.

따라서 G에 필요한 시간을 2라고 한 ④가 가장 적절하지 않다.

NoteBase

문 34. 다음 자바 언어로 작성된 소스 코드에서 나타나는 응집도로 가장 적합한 것은? (단, 코드 실행에 대한 오류는 없는 것으로 함)

```
public class Calculator {
    public int add(int num1, int num2) {
        return num1 + num2;
    }

    public int sub(int num1, int num2) {
        return num1 - num2;
    }

    public int mul(int num1, int num2) {
        return num1 * num2;
    }

    public int div(int num1, int num2) {
        if(num2 != 0)
            return num1 / num2;
    }
}
```

① 기능적 응집

② 절차적 응집

③ 논리적 응집

④ 우연적 응집

정답 3번

간단한 해설

덧셈·뺄셈·곱셈·나눗셈은 각각 독립적인 기능이지만, **사칙연산이라는 같은 범주**에 속하므로 Calculator 클래스에 함께 묶여 있다. 이러한 결합 관계가 논리적 응집이다.

NoteBase

개념 구분

응집도는 모듈 내부 요소들이 얼마나 밀접하게 관련되어 있는지를 나타낸다. Calculator 클래스의 add, sub, mul, div는 각각 덧셈·뺄셈·곱셈·나눗셈을 수행한다. 서로 다른 연산을 **사칙연산이라는 논리적 관련성**에 따라 한 클래스에 모았으므로 논리적 응집에 해당한다.

기능적 응집은 모든 구성 요소가 하나의 명확한 기능을 수행하는 데 기여하는 경우이다. 이 코드에서는 덧셈 결과를 구하는 데 sub, mul, div가 필요하지 않다. 개별 메서드가 하나의 연산을 수행한다는 사실과 **클래스 전체에 여러 독립 연산을 묶은 기준**을 구별해야 한다.

절차적 응집은 정해진 실행 순서에 따라 수행해야 하는 작업들을 묶은 경우이다. 메서드가 코드에 나열되어 있다고 해서 그 순서대로 호출되는 것은 아니며, 제시된 코드에는 **덧셈부터 나눗셈까지 차례로 수행해야 하는 절차**가 없다.

우연적 응집은 서로 관련 없는 기능들을 특별한 기준 없이 묶은 경우이다. 여기서는 네 메서드가 모두 사칙연산에 속한다는 공통점이 있으므로, 무관한 기능을 우연히 모은 경우와 구별된다.

NoteBase

문 35. 결합도에 대한 다음 설명 중 가장 적절하지 않은 것은?

① 결합도는 모듈 간의 관계가 얼마나 강하게 연결되어 있는지를 평가하는 기준이다.

② 데이터 결합은 배열, 레코드, 구조체 등의 데이터를 매개변수로 사용하는 모듈 사이의 결합을 의미한다.

③ 제어 결합은 제어 플래그(flag)를 매개변수로 사용하여, 한 모듈이 다른 모듈의 동작을 직접 결정하는 방식을 의미한다.

④ 공통 결합은 여러 모듈이 공통 변수를 공유하는 방식을 의미한다.

정답 2번

간단한 해설

필요한 개별 값의 전달은 데이터 결합이고, 복합 자료구조 전체의 전달은 이 문항의 분류상 스탬프 결합이다.
전달하는 데이터의 구조와 의존 범위를 구별해야 한다.

선지별 해설

① O: 결합도는 **모듈 사이의 의존 정도**를 나타낸다. 다른 모듈의 내부 구조나 공유 상태를 많이 알아야 동작한다면 변경의 영향이 퍼지기 쉽다. 한 모듈 내부 요소들의 관련성을 뜻하는 응집도와 구분한다.

② X: 이 문항의 전통적인 결합도 분류에서 배열·레코드·구조체 등 **복합 자료구조를 통째로 매개변수로 전달하는 것은 스탬프 결합**이다. 데이터 결합은 필요한 개별 데이터 항목을 매개변수로 교환하는 경우이다.
단순히 모두 데이터라는 이유로 두 분류를 같게 보아서는 안 되며, 복합 구조에 의존하는지 구별해야 한다.

③ O: 제어 플래그를 전달하여 상대 모듈이 수행할 기능이나 처리 경로를 지정하면 제어 결합이다. 핵심은 매개변수의 자료형이 아니라 **전달된 정보가 상대의 동작을 선택·제어한다는 역할**이다. 단순 계산값을 전달하는 데이터 결합과 구분한다.

④ O: 여러 모듈이 전역 변수와 같은 공통 데이터 영역을 함께 이용하면 공통 결합이다. **공유 상태의 변경이 여러 모듈에 영향을 줄 수 있다**는 특징이 있다. 각 호출에서 필요한 값만 명시적으로 전달하는 방식과 달리, 공통 영역에 대한 의존이 생긴다.

따라서 스탬프 결합의 설명을 데이터 결합으로 부른 ②가 가장 적절하지 않다.

NoteBase

문 36. 리스코프 교체의 원리에 대한 설명으로 가장 적절하지 않은 것은?

① 상위 클래스의 객체가 들어갈 자리에 하위 클래스의 객체를 넣어도 문제없이 작동해야 한다.

② 상위 클래스는 추상 클래스와 추상 메소드로 정의해야 한다.

③ 하위 클래스는 상위 클래스의 책임을 무시하거나 재정의하지 않고 확장만 수행해야 한다.

④ 클라이언트는 자신이 사용하지 않는 메소드와 의존 관계를 맺으면 안 된다.

정답 4번

간단한 해설

'사용하지 않는 메소드에 의존하지 않아야 한다'는 **인터페이스 분리 원칙**의 설명이다. 리스코프 교체 원칙은 **상위 클래스의 객체를 하위 클래스의 객체로 바꾸어도 기존 동작의 규약을 지켜야 한다**는 원칙이다.

출제 포인트

리스코프 교체 원칙의 대체 가능성과 인터페이스 분리 원칙의 불필요한 의존 관계 제거를 구별한다.

핵심 개념

리스코프 교체 원칙(LSP): 상위 타입을 사용하는 코드가 하위 타입의 객체를 받아도 상위 타입이 약속한 동작을 유지해야 한다.
인터페이스 분리 원칙(ISP): 클라이언트가 쓰지 않는 기능까지 의존하도록 강요하지 않도록 인터페이스를 나눈다. 클라이언트는 객체의 기능을 호출하는 코드이다.
예를 들어 상위 타입이 약속한 읽기 기능을 하위 타입이 지원하지 않는다면, 하위 객체를 대신 넣었을 때 기존 코드가 실패하므로 LSP 위반이다. 읽기만 필요한 코드에 쓰기 기능까지 의존하도록 강요하는 문제는 ISP의 판단 대상이다.

NoteBase

선지별 해설

① O
상위 클래스의 객체 대신 하위 클래스의 객체를 넣어도 기존 코드가 기대하는 동작이 유지되어야 한다. 이는 LSP의 대체 가능성을 직접 설명한다.

② O
이 문항에서는 추상 클래스와 추상 메소드로 공통 규약을 정하고 하위 클래스가 이를 구현하는 설계 방식으로 해석한다. 다만 추상 클래스 사용 자체가 LSP의 필수 조건은 아니다. 일반 클래스의 상속에서도 LSP가 적용되며, 추상 클래스를 사용해도 하위 타입이 동작의 규약을 어기면 LSP를 위반한다.

③ O
하위 클래스는 상위 클래스가 약속한 책임을 유지하면서 기능을 확장해야 한다. 여기서 '재정의하지 않고'는 메소드 오버라이딩을 금지한다는 뜻이 아니라, 상위 클래스의 책임과 동작의 규약을 바꾸어 대체 가능성을 깨뜨리지 않아야 한다는 뜻이다. 규약을 지키는 오버라이딩은 가능하다.

④ X
'사용하지 않는 메소드와 의존 관계를 맺으면 안 된다'는 ISP에 해당한다. 이를 LSP의 설명으로 제시한 것이 부적절하다. LSP의 설명으로 고치면 '클라이언트가 상위 타입의 객체를 하위 타입의 객체로 교체해도 상위 타입이 약속한 동작을 유지해야 한다'가 된다.

오답 포인트

두 원칙 모두 클라이언트에 영향을 주므로 혼동하기 쉽다. ④의 '사용하지 않는 메소드'는 ISP의 단서이고, ①의 '상위 객체 자리에 하위 객체'는 LSP의 단서이다. ②의 추상 클래스나 ③의 재정의 표현만으로 LSP를 추상 클래스 사용 의무나 오버라이딩 금지로 외우면 안 된다.

다시 나오면 이렇게 판단

객체를 하위 타입으로 교체했을 때 기존 동작의 규약을 지키는지를 묻는다면 LSP로 판단한다. 사용하지 않는 기능에 대한 의존을 줄이는지를 묻는다면 ISP로 판단한다. 상속 문법보다 클라이언트가 기대하는 동작이 유지되는지가 LSP의 기준이다.

암기 포인트

LSP는 하위 객체로 교체해도 규약 유지, ISP는 쓰지 않는 기능에 의존하지 않기이다.

NoteBase

문 37. 프로그램 언어를 사용하여 변수, 함수, 메소드 등의 이름을 지을 때 사용하는 명명 규칙과 사용 예를 바르게 연결한 것은?

(가) 파스칼 케이스(pascal case)
(나) 카멜 케이스(camel case)
(다) 스네이크 케이스(snake case)
A. StudentName
B. studentName
C. student_name

① (가): A, (나): B, (다): C
② (가): A, (나): C, (다): B
③ (가): B, (나): A, (다): C
④ (가): C, (나): B, (다): A

정답 1번

간단한 해설

StudentName은 파스칼, studentName은 카멜, student_name은 스네이크 케이스이다. **첫 글자의 대소문자와 단어 사이의 밑줄**을 기준으로 구분한다.

대응 관계 해설

(가) 파스칼 케이스 → A. StudentName: 단어를 이어 쓰되 **첫 단어를 포함하여 각 단어의 첫 글자를 대문자**로 쓴다. Student와 Name이 각각 대문자 S와 N으로 시작하므로 StudentName이 해당한다. 단어 사이에 밑줄은 넣지 않는다.

(나) 카멜 케이스 → B. studentName: 이 문항은 파스칼 케이스와 구분하여 첫 단어를 소문자로 시작하는 lower camel case를 뜻한다. **첫 단어는 소문자로, 이후 단어의 첫 글자는 대문자**로 쓰므로 studentName이다. StudentName과는 맨 앞 글자의 대소문자가 다르다.

(다) 스네이크 케이스 → C. student_name: **단어 사이를 밑줄로 구분**한다. student와 name 사이에 밑줄이 있는 student_name이 해당한다. 이 예는 소문자를 사용하는 형태이며, 단어 경계를 대문자로 표시하는 앞의 두 방식과 구별된다.

따라서 (가) A, (나) B, (다) C로 연결되므로 ①과 일치한다. 이러한 표기 방식은 이름 작성 관례이며, 어떤 대상에 어느 방식을 권장하는지는 언어와 프로젝트의 규칙에 따라 달라질 수 있다.

NoteBase

문 38. 프로젝트 제안서 평가 점수에 따른 등급이 다음 표와 같고, 경계값 분석 기법으로 테스트할 때 입력값으로 포함되지 않는 것은?

평가 점수	등급
70점 이상 ~ 100점 이하	우수
60점 이상 ~ 70점 미만	보통
0점 이상 ~ 60점 미만	미흡

① -1

② 50

③ 69

④ 100

정답 2번

간단한 해설

등급과 유효 범위의 경계는 0, 60, 70, 100이다. **50은 구간 내부 값**이므로 경계값 분석으로 선정하는 값과 구별된다.

계산 과정

구간과 경계 확인: 표의 유효 점수 범위는 0점부터 100점까지이다. 미흡은 0 이상 60 미만, 보통은 60 이상 70 미만, 우수는 70 이상 100 이하이므로 등급이나 유효성이 바뀌는 경계는 0, 60, 70, 100이다.

경계 주변 값 선정: 제시된 정수 점수 단위에서 경계값과 그 직전·직후를 확인한다. 범위 밖 입력까지 포함하면 0 주변은 −1·0·1, 60 주변은 59·60·61, 70 주변은 69·70·71, 100 주변은 99·100·101을 후보로 삼을 수 있다. **이상·미만·이하의 비교 연산 오류**를 발견하는 것이 목적이다.

선택지 대조: −1은 유효 범위의 하한 0 바로 밖의 값이고, 69는 우수 등급의 하한 70 바로 아래의 값이다. 100은 유효 범위의 상한이므로 모두 경계 관련 입력이다. 반면 **50은 미흡 구간의 내부 값**으로 어느 경계에도 인접하지 않는다.

따라서 경계값으로 선택되지 않는 입력은 ② 50이다. 50을 다른 목적의 테스트에 사용할 수 없다는 뜻은 아니다.

NoteBase

문 39. 코드 검증기준(coverage)에 대한 다음 설명 중 가장 적절하지 않은 것은?

① 문장 검증기준(statement coverage)은 원시 코드 내의 모든 문장이 최소한 한 번은 실행되도록 테스트하는 방식이다.

② 조건 검증기준(condition coverage)은 조건문이 참(T)일 때와 거짓(F)일 때의 분기 경로를 각각 한 번 이상 실행하도록 테스트하는 방식이다.

③ 다중 조건 검증기준(multiple condition coverage)은 두 개 이상의 개별 조건식이 and 또는 or로 연결되었을 때, 단일 조건 검증에서 발생할 수 있는 마스크(mask) 문제를 해결하기 위해 활용하는 기준이다.

④ 경로 검증기준(path coverage)은 원시 코드의 독립적인 경로가 최소한 한 번은 실행되는 테스트 케이스를 찾아 테스트를 수행하는 방식이다.

정답 2번

간단한 해설

조건 커버리지는 개별 조건식의 참·거짓을, 결정·분기 커버리지는 조건문 전체의 결과에 따른 분기를 검사한다. **개별 조건과 전체 결정을 혼동**한 설명을 구분해야 한다.

NoteBase

선지별 해설

① O: 문장 커버리지는 **실행 가능한 문장을 적어도 한 번씩 실행**하는지를 기준으로 한다. 문장이 모두 실행되었다고 해서 각 조건의 참·거짓 결과나 모든 경로까지 확인했다는 뜻은 아니다. 주석이나 선언의 표기 자체를 실행 대상으로 세는 개념도 아니다.

② X: 조건 커버리지는 복합 조건을 구성하는 **각 개별 조건식이 참과 거짓으로 평가되는 경우**를 검사한다. 반면 조건문 전체가 참일 때와 거짓일 때의 분기 경로를 실행하는 것은 결정 또는 분기 커버리지에 해당한다. 개별 조건의 결과와 전체 결정의 결과를 바꾸어 설명했으므로 잘못이다.

③ O: 다중 조건 커버리지는 개별 조건들의 참·거짓 조합을 검사한다. 각 조건이 참·거짓을 한 번씩 가졌다는 사실만으로는 조건들 사이의 상호작용이 충분히 검증되지 않을 수 있다. 따라서 **다른 조건에 효과가 가려지는 경우까지 조합으로 살피는 방식**으로 마스크 문제를 보완한다. 실제로 불가능한 조합이나 단락 평가로 평가되지 않는 조건은 별도로 고려해야 하며, 이것이 모든 오류의 부재를 보장하지는 않는다.

④ O: 여기서 독립 경로를 선정하여 실행한다는 설명은 **기본 경로 테스트**의 의미로 이해한다. 제어흐름 그래프에서 선형 독립 경로 집합을 정하고 각 경로를 실행하도록 테스트를 설계하는 방식이다. 엄밀히 말해 가능한 경로 전체의 실행 여부를 나타내는 완전 경로 커버리지와는 다르므로, 독립 경로만 실행하면 모든 경로 조합도 실행한 것이라고 해석해서는 안 된다.

이 문항의 독립 경로 기반 검증이라는 용어 범위에서, 개별 조건의 검증을 전체 분기의 검증으로 잘못 설명한 ②가 가장 적절하지 않다.

NoteBase

문 40. 다음은 CMMI V3.0의 성숙도 단계별 프로세스 영역을 나타낸 것이다. 성숙도 단계와 프로세스 영역의 활동 중 하나를 가장 적절하게 연결한 것은?

(가) 초기 단계
(나) 관리 단계
(다) 정의 단계
(라) 정량적 관리 단계
(마) 최적화 단계

A. 요구사항 관리
B. 프로세스 없음
C. 조직 차원의 프로세스 성과 관리
D. 근본 원인 분석 및 해결
E. 조직 차원의 혁신 활동 전개
F. 프로세스/제품 품질 보증
G. 통합된 프로젝트 관리
H. 협력 업체 관리
I. 조직 차원의 교육 훈련

① (가): A, (나): H, (다): I, (라): C, (마): D
② (가): A, (나): G, (다): H, (라): D, (마): E
③ (가): B, (나): A, (다): F, (라): C, (마): E
④ (가): B, (나): F, (다): G, (라): C, (마): D

정답 4번

간단한 해설

정량적 관리 단계는 **프로세스 성과를 수치와 통계로 관리**하고, 최적화 단계는 **근본 원인을 분석·해결하여 지속적으로 개선**하는 데 중점을 둔다. 다만 이 문항의 보기는 구판의 프로세스 영역 명칭과 단계별 배치를 사용하고 있다.

NoteBase

대응 관계 해설

발문에는 CMMI V3.0이라고 되어 있지만, 요구사항 관리·통합된 프로젝트 관리·조직 차원의 프로세스 성과 관리 등은 구판의 프로세스 영역 명칭이다. 아래 대응은 문항에 제시된 구판 체계를 기준으로 하며, V3.0의 실무 영역과 수준을 그대로 분류한 것으로 이해해서는 안 된다.

(가) 초기 단계 → B. 프로세스 없음: 초기 단계에서는 업무가 임시방편으로 수행되고 결과를 예측하기 어려우며, 개인의 역량에 크게 의존한다. 구판의 초기 단계에는 별도로 배정된 프로세스 영역이 없다. 따라서 B는 **모든 작업 절차가 전혀 없다는 뜻이 아니라**, 체계적으로 관리·제도화된 프로세스가 부족하다는 취지로 이해해야 한다. A의 요구사항 관리는 관리 단계에 배정된다.

(나) 관리 단계 → F. 프로세스/제품 품질 보증: 프로젝트를 계획하고 수행 상황과 산출물을 관리하는 단계이다. 품질 보증은 프로세스와 산출물이 적용 기준·절차를 따르는지 객관적으로 확인하고, 미준수 사항을 해결하도록 한다. 이처럼 **프로젝트 수준에서 기본적인 관리 체계를 갖추는 활동**이 관리 단계에 해당한다. A의 요구사항 관리와 H의 협력 업체 관리도 구판에서는 이 단계의 활동이다.

(다) 정의 단계 → G. 통합된 프로젝트 관리: 조직의 표준 프로세스를 정립하고, 개별 프로젝트가 이를 특성에 맞게 조정하여 사용한다. 통합된 프로젝트 관리는 **조직 표준에 기초한 프로젝트 프로세스**로 관련 활동과 이해관계자의 참여를 조정하는 활동이므로 정의 단계에 해당한다. I의 조직 차원의 교육 훈련도 이 단계에 배정되며, F의 품질 보증은 관리 단계에 배정된다.

(라) 정량적 관리 단계 → C. 조직 차원의 프로세스 성과 관리: 프로세스 성과를 측정·분석하여 성과 기준선과 예측 모형을 마련하고 정량적 관리의 근거로 사용한다. 단순히 표준 절차를 갖추는 것을 넘어 **성과와 변동을 정량적으로 이해하고 관리**하는 단계이다. 구판 체계에서 D의 근본 원인 분석 및 해결은 최적화 단계에 배정된다.

(마) 최적화 단계 → D. 근본 원인 분석 및 해결: 결함이나 성과 문제의 원인을 찾아 제거하고 재발을 방지하여 프로세스를 지속적으로 개선한다. 정량적 관리의 성과 자료를 바탕으로 **문제의 원인을 해결하고 개선을 조직에 반영**하는 데 초점을 둔다. E의 조직 차원의 혁신 활동 전개 역시 구판에서 최적화 단계에 배정되는 활동이다. 다만 V3.0에서는 원인 분석 및 해결의 실무가 여러 수준에 걸쳐 구성되므로, 이를 최적화 단계에만 존재하는 활동으로 일반화해서는 안 된다.

제시된 보기에서 각 단계의 활동을 하나씩 연결하면 (가) B, (나) F, (다) G, (라) C, (마) D이며, ④의 연결과 일치한다.

NoteBase