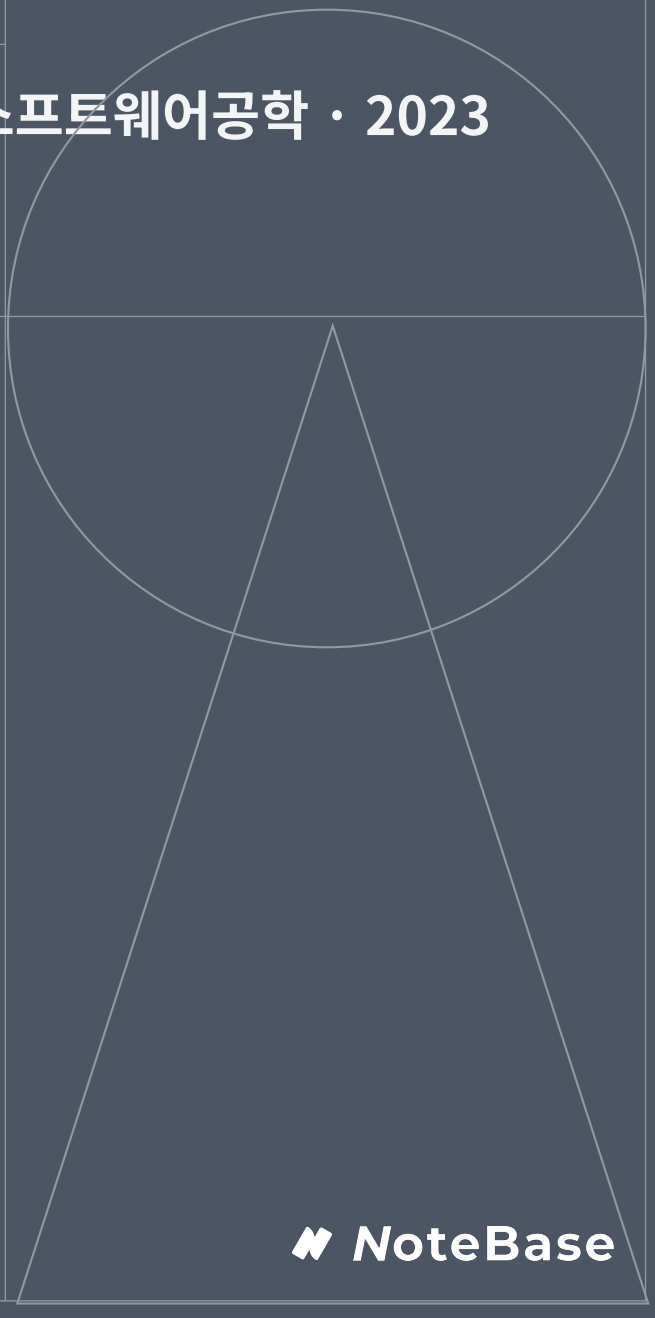
기출OX 해설집

공무원/경찰/공채(경위) · 소프트웨어공학 · 2023



NoteBase

기출OX 해설집

공무원/경찰/공채(경위) · 소프트웨어공학 · 2023

공무원/경찰/공채(경위) · 소프트웨어공학 · 2023

문 1. 작업 분류체계(WBS, Work Breakdown Structure)에 대한 설명으로 적절하지 않은 것은?

① 작업 프로세스(process) 또는 산출물 위주로 작성한다.

② 프로젝트 목표와 수행할 작업을 정의하기에 효과적이다.

③ 사용자와 개발자의 의사소통 도구로 사용할 수 있다.

④ 시스템 구축 프로젝트에 대한 일정관리 부문 결과물이다.

정답 4번

간단한 해설

WBS는 프로젝트에서 수행할 전체 작업을 계층적으로 나누는 **범위관리의 결과물**이다. 일정계획을 세우는 기초 자료로 사용되지만, 작업의 순서·기간·시점을 정한 일정관리 결과물과 같지는 않다.

NoteBase

선지별 해설

작업 분류체계는 프로젝트 목표를 달성하기 위해 필요한 전체 작업을 상위 항목에서 하위 작업 단위로 분해한 구조이다. '무슨 일을 어디까지 할 것인가'를 구체화하며, 이를 토대로 일정·비용·자원 계획을 수립한다.

① O: 작업을 개발 프로세스의 단계별로 분해하는 방식과 산출물 중심으로 분해하는 방식이 사용된다. 예를 들어 분석·설계·구현 단계의 작업을 세분화하거나, 완성할 시스템의 구성요소와 산출물을 기준으로 나눌 수 있다. 어느 방식이든 전체 작업 범위가 누락되지 않도록 해야 하며, 산출물 중심의 구조는 달성할 결과를 명확히 드러내는 데 유리하다.

② O: 큰 프로젝트 목표를 구체적인 하위 작업으로 나누면 목표 달성에 필요한 일이 무엇인지 확인할 수 있다. 수행 범위의 누락·중복을 점검하고 담당 작업을 명확히 할 수 있으므로, 목표와 실제 작업을 연결하여 정의하는 데 효과적이다.

③ O: 사용자와 개발자가 같은 작업 구조를 보면서 필요한 기능·산출물과 수행 범위를 확인할 수 있다. WBS는 무엇을 제공하고 어디까지 수행할지에 대한 이해 차이를 줄이며, 변경이 어느 작업에 영향을 주는지도 논의할 수 있게 하는 의사소통 도구이다.

④ X: WBS는 전체 작업 범위를 구조화하는 **범위관리의 산출물**이다. 일정관리에서는 이 작업 구조를 기초로 활동을 구체화하고 선후관계·소요기간·자원 등을 고려하여 일정을 개발한다. 따라서 WBS가 일정 수립과 관리에 활용된다는 사실과 WBS 자체가 일정관리 부문의 결과물이라는 주장은 구별해야 한다.

NoteBase

문 2. 다음에서 설명하는 소프트웨어 개발 방법으로 적절한 것은?

기존의 문서 중심의 전통적 개발 방법으로 고객의 변경 요청을 처리하고 통합하는 데 많은 비용과 시간이 필요한 (㉮)과 달리 (㉯)은 다양한 변화에 신속히 대처하기 위해, 짧고 반복적인 주기마다 고객에게 프로젝트에 대한 피드백을 받아 방향을 평가받고 요구사항을 반영하여 배포하는 소스코드 구현 중심의 점진적 개발 방법이다.

① ㉮: 나선형(spiral) 모델, ㉯: 애자일(agile) 모델

② ㉮: 폭포수(waterfall) 모델, ㉯: 애자일(agile) 모델

③ ㉮: 폭포수(waterfall) 모델, ㉯: 프로토타입(prototype) 모델

④ ㉮: 나선형(spiral) 모델, ㉯: 프로토타입(prototype) 모델

정답 2번

간단한 해설

단계별 문서와 계획을 중심으로 진행하여 후반의 변경 부담이 큰 ㉮는 **폭포수 모델**이다. 짧은 주기의 구현과 고객 피드백으로 변화를 반영하는 ㉯는 **애자일 모델**이다.

빈칸별 해설

㉮: 폭포수 모델이다. 요구분석·설계·구현·시험 등 단계를 순차적으로 진행하고 각 단계의 문서와 산출물을 다음 단계의 기준으로 삼는 전통적 접근이다. 개발이 진행된 뒤 요구사항이 바뀌면 앞 단계의 문서와 설계부터 다시 손봐야 하므로 **후반 변경의 비용과 시간이 커지기 쉽다**. 나선형 모델은 반복 주기마다 위험을 분석하고 대응하는 것이 중심이므로 지문의 문서 중심·순차적 개발과 구별된다.

㉯: 애자일 모델이다. 짧은 반복 주기마다 작동 가능한 기능을 구현하여 고객의 평가를 받고, 그 피드백으로 요구사항과 다음 개발 방향을 조정한다. **잦은 피드백과 변화 대응**을 통해 기능을 점진적으로 확장하는 것이 핵심이다. 프로토타입 모델도 시제품으로 요구사항을 확인하지만, 지문은 시제품 자체보다 짧은 주기의 구현·피드백·배포를 지속하는 개발 방식을 강조한다. 애자일이 문서나 계획을 전혀 사용하지 않는다는 뜻은 아니다.

따라서 ㉮는 폭포수 모델, ㉯는 애자일 모델이므로 ②에 해당한다.

NoteBase

문 3. 다음은 작업 소요 기간에 대한 전문가 의견이다. 이를 기반으로 PERT(Program Evaluation and Review Technique) 분석 기법을 통해 베타 분포를 가정하여 예측한 기간은?

가. 본 개발 작업은 기술적 위험이 높은 것으로 예상한다.
나. 일반적으로 9일, 낙관적으로 7일, 비관적으로 14일의 소요기간을 예측한다.

① 9.5일
② 10.5일
③ 11.5일
④ 12.5일

정답 1번

간단한 해설

베타분포를 가정한 PERT는 **가장 가능성이 높은 기간에 가중치 4**를 적용한다. 낙관적 7일, 일반적 9일, 비관적 14일을 사용하면 기대기간은 $\frac{7+4\times9+14}{6}=9.5$일이다.

계산 과정

PERT의 세 점 추정은 작업기간의 불확실성을 고려하여 낙관적 기간, 가장 가능성이 높은 기간, 비관적 기간을 함께 사용한다. 기술적 위험이 높다는 조건은 이러한 불확실성을 고려할 배경이지만, 별도의 가산율이 주어지지 않았으므로 임의로 기간을 더하지 않는다.

낙관적 기간을 a, 가장 가능성이 높은 기간을 m, 비관적 기간을 b라고 하면, 베타분포를 가정하는 전통적인 PERT 기대기간은 $t_e=\frac{a+4m+b}{6}$으로 추정한다. **가장 가능성이 높은 기간에 가중치 4**를 주고, 가중치 합인 6으로 나누는 방식이다.

주어진 일반적인 예상기간 9일을 m으로 대응시키면 $a=7$, $m=9$, $b=14$이며 모두 일 단위이다. 따라서 $t_e=\frac{7+4\times9+14}{6}=\frac{57}{6}=9.5$일이다.

이 값은 세 기간의 단순 산술평균이나 비관적 기간 자체가 아니라 **가중평균으로 추정한 기대기간**이다. 실제 작업이 반드시 그 기간에 끝난다는 보장은 아니다.

계산된 기대기간은 9.5일이므로 ①과 일치한다.

NoteBase

문 4. 요구공학(requirement engineering)에 대한 설명으로 적절하지 않은 것은?

① 소프트웨어 요구공학은 요구사항 개발과 요구사항 관리로 분류된다.

② 요구사항 개발은 추출, 분석, 명세, 검증·확인, 명세서 산출물 작성 절차로 진행한다.

③ 요구사항 추출 단계는 요구 정보 출처 확인, 요구 정보 취합, 요구와 제약사항 정리 등의 활동을 수행한다.

④ 요구사항 검증·확인 단계는 도메인 분석, 내·외부 인터페이스를 포함하는 행위에 대한 평가, 요구사항의 우선순위 평가, 제약사항에 대한 대안 검토 등의 활동들을 수행한다.

정답 4번

간단한 해설

도메인·인터페이스·우선순위·제약에 따른 대안 검토는 주로 **요구사항 분석** 활동이다. 검증·확인은 명세된 요구가 실제 필요에 맞고 누락·모순·모호함 등이 없는지를 확인한다.

선지별 해설

① O: 요구공학은 **요구사항 개발**과 **요구사항 관리**로 나누어 설명할 수 있다. 개발은 이해관계자의 요구를 찾아 분석·명세·검토하는 활동이고, 관리는 합의된 요구의 변경, 버전, 추적 관계 등을 지속적으로 통제하는 활동이다.

② O: 요구사항을 추출하여 모은 뒤 분석하고, 명세로 표현하여 검증·확인을 거치며, 그 결과를 명세서 산출물로 정리한다. 명세서는 검토 전에 초안이 작성되고 검토 결과를 반영하여 보완되므로, 이 절차를 **한 번만 진행하는 엄격한 일방향 과정으로 이해해서는 안 된다**.

③ O: 추출은 요구를 얻을 정보원과 이해관계자를 찾아 정보를 수집하는 활동이다. 사용자 면담, 기존 문서 조사, 관찰 등을 통해 필요한 기능과 운용상의 제약을 파악하고 정리한다. 이후 분석할 요구 정보를 확보하는 것이 핵심이다.

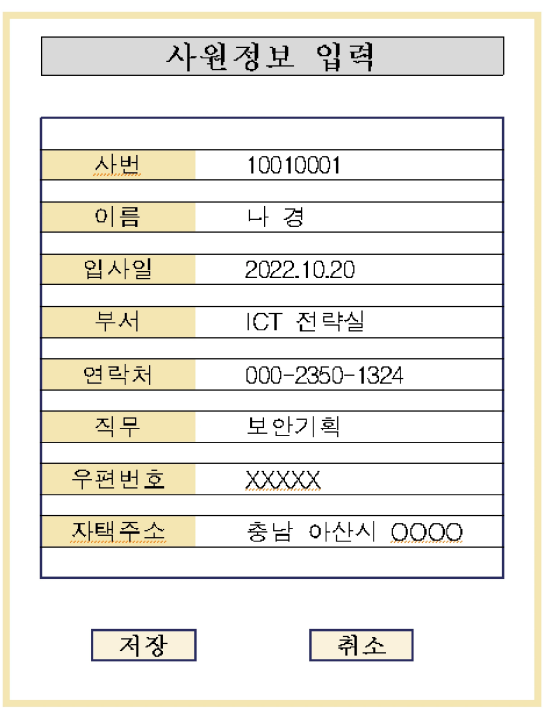
④ X: 도메인을 이해하고 내·외부 인터페이스와 행위를 평가하며, 우선순위와 제약에 따른 대안을 검토하는 것은 주로 **요구사항 분석**에 해당한다. 검증·확인은 명세된 요구가 실제 이해관계자의 필요를 반영하는지, 누락·모순·모호함이 없는지, 실현 가능하고 검증 가능한지를 점검하는 활동이다. 요구 내용과 대안을 구체화하는 분석과, 정리된 요구의 적합성·품질을 확인하는 검토를 구별해야 한다.

따라서 분석 활동을 검증·확인 단계의 활동으로 제시한 ④가 적절하지 않다.

NoteBase

문 5. 다음 인사관리시스템의 사원정보 입력 기능을 대상으로 FP(Function Point) 기법을 이용하여 업무 규모 측정 시, 이에 대한 설명으로 적절하지 않은 것은? (단, 트랜잭션(transaction)명은 출력 화면 정보를 기준으로 한다.)

가. '저장' 버튼 클릭 시, 확인 메시지가 출력되고, 해당 정보는 사원정보 테이블에 저장된다.
나. 부서 항목은 연계된 조직관리 시스템에 저장된 하나의 테이블을 읽어 조회된 값을 선택한다.
다. 우편번호, 자택 주소의 기본 정보항목은 연계된 표준화 시스템에 저장된 하나의 테이블을 읽어 조회된 값을 선택한다.



① 트랜잭션 데이터는 '사원정보'로 정의하고, 기능 항목을 ILF(Internal Local File)로 식별한다.
② 트랜잭션 명은 '사원정보 입력'으로 정의하고, 기능 항목을 EI(External Input)로 식별한다.
③ '사원정보 입력' 트랜잭션의 FTR(File Type Reference) 개수는 3이다.
④ '사원정보 입력' 트랜잭션의 DET(Data Element Type) 개수는 9이다.

정답 4번

간단한 해설

사원정보 입력 화면의 데이터 항목은 8개이다. 여기에 **저장 실행 동작 1개와 확인 메시지 1개**를 포함하므로 DET는 9개가 아니라 **10개**이다.

NoteBase

선지별 해설

① O: 사원정보는 이 인사관리시스템의 저장 기능으로 생성·유지되는 사용자 식별 가능한 논리적 데이터 집합이므로 **내부논리파일(ILF)**로 분류한다. 데이터 기능은 '사원정보', 이를 처리하는 트랜잭션 기능은 '사원정보 입력'으로 구별한다. 다만 선지의 영문 풀어쓰기 'Internal Local File'은 인쇄상의 오기이며, ILF의 표준 명칭은 **Internal Logical File**이다. 여기서 적절한 것은 사원정보를 ILF로 식별한 기능 분류이다.

② O: 화면의 '사원정보 입력'은 외부에서 들어온 사원 데이터를 받아 내부논리파일인 사원정보를 유지하는 단위 프로세스이므로 **외부입력(EI)**에 해당한다. 저장 후 확인 메시지를 출력하더라도 주된 목적이 내부 데이터 유지이므로, 그 메시지만을 이유로 외부출력(EO)으로 분류하지 않는다. 트랜잭션 명은 문제의 조건에 따라 화면 제목인 '사원정보 입력'으로 정한다.

③ O: FTR은 해당 트랜잭션이 유지하거나 참조하는 논리파일의 수이다. 저장 대상인 사원정보 파일 1개, 부서 선택을 위해 읽는 조직관리 시스템의 파일 1개, 우편번호와 자택 주소를 조회하는 표준화 시스템의 파일 1개를 합해 **3개**이다. 외부 시스템에서 유지되는 두 파일은 이 시스템 입장에서 외부연계파일(EIF)이다. 우편번호와 주소는 동일한 하나의 파일에서 읽으므로 서로 다른 FTR로 중복 계산하지 않는다.

④ X: DET는 사용자가 식별하는 중복되지 않는 데이터 요소를 센다. 화면에는 사번, 이름, 입사일, 부서, 연락처, 직무, 우편번호, 자택 주소의 **8개 입력 항목**이 있다. 여기에 해당 입력 프로세스를 실행하는 '저장' 동작 1개와 조건 가에서 명시한 확인 메시지 1개를 더하므로 **DET는 8 + 1 + 1 = 10개**이다. 부서·우편번호·주소를 연계 조회 후 선택하더라도 저장되는 입력 데이터 요소에 포함한다. 입사일이나 연락처를 화면에 표시된 구분 기호별로 쪼개어 세지 않으며, '취소' 버튼은 사원정보를 저장하는 이 입력 프로세스의 DET를 하나 더 늘리지 않는다. 따라서 9개라는 설명은 적절하지 않다.

NoteBase

문 6. COCOMO II(Constructive COst MOdel II) 비용 산정 모델의 서브 모델(sub model)이 아닌 것은?

① 반 결합형 모델(semi-detached model)

② 응용 결합 모델(application-composition model)

③ 초기 설계 모델(early design model)

④ 포스트 아키텍처 모델(post architecture model)

정답 1번

간단한 해설

반 결합형은 COCOMO 81에서 유기형·내장형과 함께 구분하던 **프로젝트의 개발 모드**이다. COCOMO II의 하위 모델은 응용 결합, 초기 설계, 포스트 아키텍처 모델로 구분되므로 서로 다른 분류를 혼동해서는 안 된다.

선지별 해설

COCOMO II는 개발 방식과 확보된 설계 정보의 상세 수준에 맞추어 비용과 노력을 추정한다. 아래 O/X는 제시된 명칭이 COCOMO II의 하위 모델에 해당하는지를 나타낸다.

① X: 반 결합형은 기존 COCOMO 81의 개발 모드이며, 유기형과 내장형 사이의 중간적인 프로젝트 특성을 다룬다. 개발 경험과 제약 조건 등이 서로 다른 프로젝트에 다른 계수를 적용하기 위한 구분이지, COCOMO II에서 설계 정보가 구체화되는 단계에 따라 선택하는 하위 모델의 명칭은 아니다.

② O: 응용 결합 모델은 재사용 가능한 구성요소를 결합하여 응용 프로그램이나 시제품을 빠르게 만드는 경우에 적합하다. 화면·보고서·프로그램 구성요소 등을 반영한 **객체점수**로 규모를 평가하고, 재사용 정도와 생산성 등을 고려하여 필요한 노력을 추정한다.

③ O: 초기 설계 모델은 요구사항과 대략적인 규모는 알지만 상세한 아키텍처와 개발 조건이 아직 확정되지 않은 단계에서 사용한다. 비교적 적은 수의 통합된 비용 요인을 이용하여 **설계 대안의 비용을 개략적으로 비교**하는 데 적합하다.

④ O: 포스트 아키텍처 모델은 아키텍처가 정해지고 제품·플랫폼·인력·프로젝트에 대한 상세 정보를 활용할 수 있을 때 적용한다. 초기 설계 모델보다 세분화된 비용 요인과 규모 요인을 반영하여 개발 노력을 더 상세히 추정한다. '포스트 아키텍처'는 아키텍처 수립 이후라는 뜻이며, 소프트웨어 개발이 모두 끝난 뒤에만 사용한다는 뜻은 아니다.

NoteBase

문 7. 소프트웨어 요구사항 명세서의 내용으로 적절하지 않은 것은?

① 학생은 학부과정과 대학원 과정을 동시에 수강할 수 없다.

② 교수는 사번과 패스워드를 이용하여 시스템에 로그인한다.

③ 시스템의 페이지는 최대한 빨리 화면에 출력되어야 한다.

④ 시스템의 모든 트랜잭션은 128비트 암호화되어야 한다.

정답 3번

간단한 해설

요구사항은 충족 여부를 확인할 수 있어야 한다. **'최대한 빨리'는 측정 가능한 시간 기준이 없어** 검증하기 어려우므로, 시험 조건과 응답시간의 상한을 구체적으로 정해야 한다.

선지별 해설

① O: 이 시스템에서 학부과정과 대학원 과정의 동시 수강을 허용하지 않는다는 **업무 규칙**을 명시한 것이다. 두 과정에 대한 중복 수강 신청을 제한하는 기능과 시험 조건을 도출할 수 있다. 모든 대학에 통용되는 사실이라는 뜻이 아니라, 개발 대상 시스템이 따라야 할 규칙을 정한 문장이다.

② O: 교수라는 사용자와 사번·패스워드라는 인증 수단을 지정한 로그인 기능 요구사항이다. 시스템이 어떤 입력으로 사용자를 인증해야 하는지 확인할 수 있으므로 요구사항 명세의 내용으로 적절하다. 실제 명세에서는 인증 실패 시 처리 등 관련 조건도 추가로 정의할 수 있다.

③ X: **'최대한 빨리'에는 객관적인 합격 기준이 없다**. 개발자와 고객이 서로 다르게 해석할 수 있고, 화면 출력 시간이 요구를 충족하는지 시험으로 판정하기 어렵다. 예를 들어 운영환경과 동시 접속 부하를 정한 뒤 '페이지 요청을 받은 시점부터 화면 출력 완료까지 2초 이내'처럼 측정 구간과 허용 시간을 명시해야 한다. 이 수치는 구체화 방식의 예시이며 원문에 주어진 값은 아니다.

④ O: 모든 트랜잭션에 암호화를 적용하고 128비트라는 수치 조건을 둔 **보안 요구사항**이다. 명세서에는 기능뿐 아니라 보안 같은 비기능 요구사항도 포함할 수 있다. 실무에서는 사용할 암호 방식과 보호 대상·구간 등을 함께 구체화해야 하지만, 이 문장은 '최대한'처럼 기준 없는 희망사항과는 구별된다. 128비트라는 표현만으로 실제 보안성이 모두 보장된다는 뜻은 아니다.

NoteBase

문 8. 비기능적 소프트웨어 요구사항으로 적절하지 않은 것은?

① 자료 요구사항(data requirement)

② 품질 요구사항(quality requirement)

③ 성능 요구사항(performance requirement)

④ 보안 요구사항(security requirement)

정답 1번

간단한 해설

기능적 요구사항은 시스템이 처리할 데이터와 제공할 동작 등 **무엇을 해야 하는지**를 정한다. 비기능적 요구사항은 품질·성능·보안과 같이 그 기능과 시스템이 충족해야 할 수준이나 제약을 정한다.

선지별 해설

기능적 요구사항은 제공할 동작·서비스를, 비기능적 요구사항은 품질 특성이나 수행상의 제약을 규정한다. 아래 O/X는 각 항목이 이 문항에서 비기능적 요구사항의 범주에 해당하는지를 나타낸다.

① X: 자료 요구사항은 이 문항의 분류에서 시스템이 **어떤 데이터를 입력·처리·저장·출력할지**를 정하는 내용으로, 기능적 요구사항과 연결된다. 예를 들어 주문에 포함할 항목이나 저장·조회할 자료를 정하는 요구가 이에 해당한다. 다만 데이터에 관한 문장이라고 해서 모두 기능적인 것은 아니며, 저장용량의 한계나 처리속도·보호 수준 같은 제약은 비기능적 요구사항이 될 수 있다.

② O: 품질 요구사항은 신뢰성, 사용성, 유지보수성 등 소프트웨어가 갖추어야 할 **품질 수준**을 정한다. 특정 업무 기능의 존재뿐 아니라 그 기능과 시스템을 얼마나 안정적이고 편리하게 사용할 수 있어야 하는지를 규정하므로 비기능적 요구사항의 대표적인 범주이다.

③ O: 성능 요구사항은 응답시간, 처리량, 자원 사용량 등 **수행 능력과 효율의 기준**을 정한다. 예를 들어 검색 결과를 일정 시간 안에 반환해야 한다는 요구는 검색 기능 자체를 지정하는 것과 달리 그 기능이 충족할 성능 수준을 제한한다.

④ O: 보안 요구사항은 기밀성·무결성·가용성 등 **시스템과 데이터의 보호 수준**을 규정하는 범주로서 비기능적 요구사항에 포함된다. 이러한 보안 목표를 구현하기 위해 인증·권한 검사 같은 구체적인 기능 요구사항이 도출될 수 있으므로, 보안이라는 단어가 포함된 모든 개별 요구를 무조건 비기능적으로 분류하는 것은 아니다.

이 문항의 분류에서 비기능적 요구사항으로 적절하지 않은 것은 자료 요구사항인 ①이다.

NoteBase

문 9. 모듈화(modularization)에 대한 설명으로 적절하지 않은 것은?

① 모듈은 독립적인 기능이 있는 논리적 단위로 다른 프로젝트에 재사용할 수 있다.

② 분석에서 구현까지 소프트웨어 개발의 전체 수명주기에 걸쳐 적용될 수 있다.

③ 소프트웨어의 문제 발생 시 원인 모듈을 수정한 후 적용한다.

④ 모듈화는 문제를 팀 단위의 개발 요소가 될 만한 수준으로 분할하는 과정이다.

정답 4번

간단한 해설

④의 '팀 단위'는 업무를 배분하는 규모를 나타낼 뿐, 모듈화의 핵심 기준이 아니다. 모듈화는 소프트웨어를 **명확한 기능과 책임을 가진 독립적인 단위**로 나누어 개발·수정하기 쉽게 하는 과정이다.

선지별 해설

① O: 모듈은 특정 기능을 담당하는 논리적 단위이다. 외부에서 이용하는 방법인 인터페이스가 명확하고 다른 부분에 대한 의존이 작으면, 같은 기능이 필요한 다른 프로젝트에서도 **재사용**할 수 있다. 모든 모듈이 아무 수정 없이 재사용된다는 뜻은 아니다.

② O: 모듈화는 코드를 작성할 때만 사용하는 기법이 아니다. 분석 단계에서 기능과 책임을 나누고, 설계 단계에서 모듈의 경계와 인터페이스를 정하며, 구현 단계에서 이를 코드로 작성하는 등 **개발 수명주기 전반**에 적용할 수 있다.

③ O: 모듈별 기능과 경계가 명확하면 오류가 발생한 기능을 담당하는 모듈을 찾아 수정하기 쉽다. 다른 모듈과의 의존이 작을수록 **수정의 영향 범위**도 줄어든다. 수정한 모듈의 동작과 다른 모듈과의 연계를 확인한 뒤 적용한다.

④ X: '팀 단위의 개발 요소가 될 만한 수준'이라는 규모를 모듈화의 기준으로 제시한 점이 부적절하다. 모듈화는 소프트웨어를 개발자가 이해하고 관리할 수 있는 단위로 나누되, 각 단위가 **명확한 기능과 책임을 갖고 다른 단위에 대한 의존을 줄이도록** 구성하는 과정이다. 모듈의 개발을 팀에 맡길 수는 있지만, 팀별로 일을 나누었다는 사실만으로 모듈화가 이루어지는 것은 아니다.

NoteBase

문 10. 객체지향 설계에서 속성(attribute)과 오퍼레이션(operation)을 함께 묶어 표현한 개념은?

① 클래스(class)

② 구조체(structure)

③ 루틴(routine)

④ 서브루틴(sub-routine)

정답 1번

간단한 해설

클래스는 같은 종류의 객체가 가지는 **속성과 오퍼레이션을 함께 정의**하는 설계 단위이다. 속성은 상태를, 오퍼레이션은 동작이나 서비스를 나타낸다.

개념 구분

클래스는 같은 종류의 객체가 갖는 **속성과 오퍼레이션을 함께 정의**하는 설계 단위이다. 속성은 객체의 상태를 나타내는 정보이고, 오퍼레이션은 객체가 제공하는 동작이나 서비스이다. 클래스는 이 둘의 공통 구조를 정하고, 개별 객체는 그 클래스의 인스턴스로 구별된다.

예를 들어 '도서' 클래스에 제목·대출 상태라는 속성과 대출·반납이라는 오퍼레이션을 정의할 수 있다. 특정 도서 객체는 자신의 제목과 대출 상태를 가지며, 정의된 동작에 따라 상태가 바뀐다. 데이터와 관련 동작을 하나의 단위로 묶는 것은 캡슐화의 핵심이며, 그 내용을 표현하는 대표적인 객체지향 설계 개념이 클래스이다.

구조체는 전통적으로 관련된 데이터 항목을 하나로 묶는 자료형을 설명할 때 쓰는 말이다. 다만 C++처럼 구조체에도 멤버 함수를 둘 수 있는 언어가 있으므로, 모든 언어에서 구조체는 동작을 가질 수 없다고 단정해서는 안 된다. 이 문항은 특정 언어의 선언 문법보다 객체지향 설계에서 상태와 동작을 함께 정의하는 일반 개념을 묻는다.

루틴과 서브루틴은 특정 처리를 수행하는 명령들의 단위이며, 서브루틴은 다른 부분에서 호출하여 사용하는 하위 처리 단위를 뜻한다. 객체 종류의 속성과 동작을 함께 정의하는 설계 단위와는 초점이 다르다.

따라서 발문이 가리키는 개념은 클래스이며, 선택지는 ①이다.

NoteBase

문 11. 소프트웨어 설계에서 사용하는 추상화(abstraction)의 종류가 아닌 것은?

① 제어(control) 추상화

② 검증(verification) 추상화

③ 과정(procedural) 추상화

④ 자료(data) 추상화

정답 2번

간단한 해설

소프트웨어 설계의 대표적인 추상화는 제어·과정·자료 추상화이다. **검증은 명세 충족 여부를 확인하는 활동으로, 이 추상화 분류에 해당하지 않는다**.

선지별 해설

① O: **제어 추상화**는 프로그램의 실행 흐름이나 동기화 방식을 세부 구현과 구분하여 표현한다. 예를 들어 세마포어로 작업 사이의 동기화 관계를 나타낼 때, 그 내부에서 대기 작업을 관리하는 구체적인 절차까지 모두 드러내지는 않는다.

② X: **검증은 소프트웨어가 명세나 정해진 조건을 만족하는지 확인하는 활동**이다. 이 문항에서 묻는 소프트웨어 설계의 대표적 추상화 분류는 제어·과정·자료 추상화이므로, 검증 추상화는 여기에 해당하지 않는다. 이는 검증 과정에서 추상화 기법을 활용할 수 없다는 뜻은 아니다.

③ O: **과정 추상화**는 특정 기능을 수행하는 일련의 처리 과정을 하나의 절차나 함수로 표현하고 내부 알고리즘의 세부 단계를 생략하는 것이다. 예를 들어 정렬 기능을 호출하는 쪽은 입력과 결과를 이용하면서 실제 정렬 절차를 일일이 알 필요가 없다.

④ O: **자료 추상화**는 자료의 핵심 특성과 제공되는 연산에 집중하고 구체적인 저장 표현을 감추는 것이다. 예를 들어 스택을 삽입·삭제 등의 연산으로 다루면, 내부를 배열로 구현했는지 연결 구조로 구현했는지와 분리하여 사용할 수 있다.

NoteBase

문 12. 객체지향 설계 원리 중 캡슐화(encapsulation)에 대한 설명으로 적절하지 않은 것은?

① 객체 제공자와 사용자(외부 객체)를 명확히 분리할 수 있다.

② 객체 내부 정보의 손상이나 오용(misuse)을 막을 수 있다.

③ 객체 간 인터페이스를 복잡하게 만든다.

④ 객체 간 독립성이 구조적으로 보장된다.

정답 3번

간단한 해설

캡슐화는 내부 구현의 복잡성을 감추고 **외부에 필요한 인터페이스만 공개**하여 객체를 사용하기 쉽게 만든다. 객체 간 인터페이스를 복잡하게 만드는 원리라는 설명은 반대이다.

선지별 해설

① O: 캡슐화는 객체의 데이터와 이를 다루는 연산을 묶고, 외부에는 필요한 사용 방법만 공개한다. 객체 제공자는 내부 구현을 담당하고 사용자는 공개된 인터페이스를 이용하므로 **구현과 사용을 분리**할 수 있다.

② O: 내부 상태에 대한 직접 접근을 제한하고 정해진 연산을 통해서만 상태를 바꾸게 하면 잘못된 변경을 통제할 수 있다. 예를 들어 값을 저장하기 전에 허용 범위를 검사하도록 만들 수 있다. 이는 내부 정보의 손상과 오용을 줄이는 효과이며, 모든 논리 오류나 보안 사고를 자동으로 막는다는 뜻은 아니다.

③ X: 캡슐화는 외부에서 알아야 할 내부 세부사항을 감추고 **필요한 연산만 인터페이스로 제공**하여 사용 관계를 단순하고 명확하게 만드는 데 목적이 있다. 따라서 객체 간 인터페이스를 복잡하게 만드는 원리라는 설명을, 내부 복잡성을 숨겨 인터페이스 사용을 단순화한다는 설명으로 고쳐야 한다.

④ O: 내부 상태와 구현을 감추는 경계가 있으므로 다른 객체가 그 내부에 직접 의존하지 않게 하는 구조를 마련한다. 여기서 독립성은 **공개 인터페이스의 약속을 유지하면서 내부 구현을 변경**할 수 있다는 의미이다. 객체 사이의 호출이나 인터페이스에 대한 의존까지 전혀 없어진다는 뜻은 아니다.

따라서 적절하지 않은 설명은 ③이다.

NoteBase

문 13. 다음과 관련한 SOLID 설계 원칙으로 적절한 것은?

리팩토링(refactoring)을 통해 기존의 설계를 수정하여 추상 클래스 계층과 구현 클래스 계층을 분리하고 인터페이스 클래스를 통해 구현 클래스를 실체화함으로써, 인터페이스를 클라이언트별로 다양화하였다.

① SRP(Single Responsibility Principle)
② DIP(Dependency Inversion Principle)
③ ISP(Interface Segregation Principle)
④ LSP(Liskov Substitution Principle)

정답 3번

간단한 해설

인터페이스를 **클라이언트별로 필요한 기능에 맞게 분리**한 것이 핵심이다. 클라이언트가 사용하지 않는 기능에 의존하지 않도록 하는 설계 원칙이 ISP(인터페이스 분리 원칙)이다.

공통 해설

ISP(Interface Segregation Principle, 인터페이스 분리 원칙)는 하나의 범용 인터페이스에 모든 기능을 모으기보다, 클라이언트가 필요로 하는 기능별로 인터페이스를 나누는 원칙이다. 여기서 클라이언트는 해당 기능을 호출하여 사용하는 코드를 뜻한다. 각 클라이언트가 **사용하지 않는 기능에 의존하지 않게** 하면, 다른 클라이언트만 사용하는 기능의 변경이 자신에게 미치는 영향을 줄일 수 있다.

예를 들어 인쇄 기능만 필요한 코드와 스캔 기능만 필요한 코드가 있다면, 인쇄·스캔을 모두 포함한 인터페이스 대신 각각 필요한 기능의 인터페이스에 의존하도록 나눌 수 있다. 하나의 복합기 구현 클래스가 두 인터페이스를 모두 구현하는 것도 가능하다.

지문에서도 추상 클래스와 구현 클래스의 계층을 분리한 데서 나아가, '인터페이스를 클라이언트별로 다양화'하였다. 단순히 인터페이스를 사용했다는 사실보다 **클라이언트의 요구에 따라 인터페이스를 분리했다는 점**이 ISP를 식별하는 기준이다.

NoteBase

문 14. MVC(Model-View-Controller) 아키텍처에 대한 설명으로 적절하지 않은 것은?

① 컨트롤러는 뷰가 요청한 결과를 모델로부터 가져와 가공 및 처리한다.

② 모델은 DBMS(DataBase Management System)를 이용해 데이터베이스의 자료를 수정한다.

③ 사용자는 브라우저를 통해 컨트롤러에게 데이터 처리를 요청한다.

④ 뷰는 사용자에게 보여줄 결과물을 생성하여 컨트롤러에게 반환한다.

정답 1번

간단한 해설

이 문항의 웹 MVC 흐름에서 컨트롤러는 **사용자 요청을 받아 처리를 조정**하며, 모델은 데이터와 업무 처리를, 뷰는 결과의 표현을 맡는다. 컨트롤러의 주된 역할을 뷰가 요청한 모델 결과를 가공·처리하는 것으로 설명하면 요청 주체와 역할 분담을 혼동하게 된다.

선지별 해설

MVC는 모델·뷰·컨트롤러의 책임을 분리하는 구조이다. 제시된 보기들은 브라우저의 요청을 컨트롤러가 받아 모델 작업과 뷰의 결과 생성을 연결하는 웹 MVC의 논리적 처리 흐름을 설명한다. 구체적인 호출·반환 경로는 프레임워크에 따라 달라질 수 있다.

① X: 이 문항의 흐름에서 요청의 출발점은 사용자이며, 컨트롤러는 사용자 입력을 해석하여 필요한 모델 작업을 요청하고 결과를 표시할 뷰를 선택한다. 모델은 데이터와 업무 규칙에 따른 처리를, 뷰는 표현을 담당하므로, 컨트롤러를 뷰의 요청에 따라 모델 결과를 가공·처리하는 주체로 설명한 것은 역할을 부정확하게 나타낸다. 다만 컨트롤러가 모델의 결과를 조회하거나 뷰에 전달할 데이터를 정리하는 것까지 금지된다는 뜻은 아니다.

② O: 데이터베이스를 사용하는 웹 응용에서 모델 측은 데이터를 조회·수정하고 업무 규칙을 적용하는 책임을 맡는다. 저장된 자료를 수정하려면 DBMS의 기능을 이용하므로 보기의 설명과 연결된다. 이는 모든 모델 객체가 직접 DBMS에 접속해야 한다는 뜻은 아니며, 데이터 접근 계층 등을 통해 수행할 수도 있다.

③ O: 사용자가 브라우저에서 화면 조회나 자료 변경을 요청하면 웹 응용의 컨트롤러가 요청을 받아 처리 흐름을 정한다. 컨트롤러는 요청에 맞는 모델 작업을 실행하도록 연결하고, 필요한 데이터와 뷰를 선택하여 응답을 준비한다.

④ O: 뷰는 모델의 처리 결과와 전달받은 데이터를 이용하여 사용자가 볼 화면 등 **표현 결과**를 생성한다. 보기의 '컨트롤러에게 반환'은 뷰의 처리가 끝난 뒤 컨트롤러 측의 응답 흐름을 통해 사용자에게 결과가 전달된다는 논리적 설명이다. 실제 구현에서는 프런트 컨트롤러나 뷰 엔진이 렌더링과 응답을 처리할 수 있으므로, 항상 동일한 메서드 반환 경로를 거친다는 의미로 확대하지 않는다.

NoteBase

문 15. 다음의 고객 요구사항을 반영하기 적절한 디자인 패턴은?

표를 통해 정보가 추가/수정/삭제되며, 변경사항은 관련한 분석 그래프에 동시에 반영되어 사용자에게 화면으로 제공한다.

① Iterator pattern
② Singleton pattern
③ Observer pattern
④ Factory Method pattern

정답 3번

간단한 해설

표의 데이터가 바뀌면 그 데이터에 의존하는 그래프들도 갱신되어야 한다. 주체의 변경을 등록된 관찰자들에게 전달하는 **Observer pattern의 변경 통지 관계**가 이 요구에 맞는다.

대응 관계 해설

요구사항의 **표 데이터 변경 → 관련 그래프 갱신**은 Observer pattern에 대응한다. 변경 대상 데이터 모델을 주체(Subject), 이를 표시하는 그래프들을 관찰자(Observer)로 두고, 관찰자들이 주체에 등록하도록 한다. 표에서 정보가 추가·수정·삭제되면 주체가 등록된 관찰자들에게 변경을 통지하고, 각 그래프는 새 상태를 반영하여 화면을 갱신한다. 이는 한 객체의 상태 변화가 이에 의존하는 객체들에 전달되는 일대다 관계이다. '동시에 반영'은 관련 화면이 변경과 연동되어 갱신된다는 요구이며, 패턴 자체가 모든 갱신의 물리적 동시 실행이나 원자성을 보장한다는 뜻은 아니다.

Iterator pattern은 집합 객체의 내부 표현을 노출하지 않고 원소를 차례로 순회하게 한다. 표의 데이터를 순서대로 읽는 데 사용할 수 있지만, 데이터가 바뀔 때 그래프에 변경을 알리는 관계를 핵심으로 삼는 패턴은 아니다.

Singleton pattern은 클래스의 인스턴스를 하나로 제한하고 그 인스턴스에 접근할 수 있게 한다. 인스턴스 수를 관리하는 목적이므로 여러 그래프에 변경사항을 전달하는 요구를 직접 설명하지 않는다.

Factory Method pattern은 객체 생성용 인터페이스를 마련하고 구체적으로 어떤 클래스의 객체를 생성할지를 하위 클래스가 정하도록 한다. 그래프 객체를 만드는 방식에는 활용할 수 있으나, 생성된 그래프들이 데이터 변경 통지를 받는 관계와는 구별된다.

따라서 주어진 고객 요구사항과 연결되는 디자인 패턴은 ③ Observer pattern이다.

NoteBase

문 16. GoF(Gang of Four) 디자인 패턴 분류 중 행위 패턴 (behavioral pattern)에 속하지 않는 것은?

① Strategy pattern

② State pattern

③ Singleton pattern

④ Iterator pattern

정답 3번

간단한 해설

Singleton은 클래스의 인스턴스를 하나로 제한하고 접근 지점을 제공하는 **생성 패턴**이다. 객체의 알고리즘·상태별 동작·순회 방식을 다루는 행위 패턴에 속하지 않는다.

선지별 해설

아래 O/X는 각 패턴이 'GoF의 행위 패턴에 속한다'는 판단을 나타낸다.

① O: Strategy(전략)는 **행위 패턴**이다. 같은 목적을 달성하는 여러 알고리즘을 각각 캡슐화하고 교체하여 사용할 수 있게 한다. 예를 들어 요금 계산 방식을 바꾸더라도 이를 호출하는 코드는 공통 인터페이스를 이용할 수 있다.

② O: State(상태)는 **행위 패턴**이다. 객체의 내부 상태에 따라 같은 요청에 대한 동작이 달라지도록 상태별 행위를 분리한다. Strategy가 사용할 알고리즘의 교체에 초점을 둔다면, State는 객체의 상태 변화와 그에 따른 동작 변화에 초점을 둔다.

③ X: Singleton(싱글턴)은 **생성 패턴**이다. 특정 클래스의 인스턴스가 하나만 존재하도록 생성·접근을 통제하고, 그 인스턴스에 접근하는 지점을 제공한다. 알고리즘이나 객체 간 행위의 분담보다 인스턴스의 생성과 수를 다루므로 행위 패턴으로 분류하지 않는다.

④ O: Iterator(반복자)는 **행위 패턴**이다. 집합 객체의 내부 표현을 외부에 노출하지 않고 그 요소들에 차례로 접근하는 방법을 제공한다. 저장 구조 자체를 조립하는 것이 아니라 요소에 접근하고 순회하는 행위를 분리한다.

따라서 행위 패턴에 속하지 않는 것은 ③ Singleton pattern이다.

NoteBase

문 17. UML 다이어그램(diagram)에 대한 설명으로 적절하지 않은 것은?

① 사용사례(usecase) 다이어그램은 업무 프로세스를 나타내는 사용사례, 시스템을 사용하는 액터(actor), 이들 간 상호작용은 간선으로 표현한다.

② 클래스 다이어그램은 클래스 내에서 이벤트를 처리하는 논리적인 과정을 표현하며, 프로세스의 처리 과정에 대한 도식화에 사용된다.

③ 상태(state) 다이어그램은 상태, 전이, 이벤트로 구성되며, 객체의 상태에 대한 이벤트 발생이나 시간의 흐름에 따른 변화를 나타낸다.

④ 컴포넌트(component) 다이어그램은 구현 관점에서 정적 모델링을 할 때, 구성된 실행 모듈과 이들 간 연관성 및 관계를 표현한다.

정답 2번

간단한 해설

클래스 다이어그램은 클래스의 속성·연산과 관계를 보여 주는 **정적 구조**의 표현이다. 분기나 작업 순서 같은 프로세스의 처리 흐름은 주로 **활동 다이어그램**으로 나타낸다.

NoteBase

선지별 해설

UML 다이어그램은 모델의 구조를 나타내는지, 기능·상태 변화 등의 행동을 나타내는지에 따라 용도가 달라진다.

① O: 사용사례 다이어그램은 **액터와 시스템이 제공하는 기능**의 관계를 나타낸다. 액터는 시스템과 상호작용하는 외부의 역할이며, 사람뿐 아니라 다른 시스템도 될 수 있다. 사용사례는 액터의 목표를 달성하는 기능 단위이고, 액터와 사용사례의 참여 관계는 연결선으로 표현한다. 여기서 업무를 나타낸다는 것은 사용자 관점의 기능과 범위를 표현한다는 뜻이며, 업무 내부의 세부 처리 순서를 직접 그린다는 뜻은 아니다.

② X: 클래스 다이어그램은 클래스의 속성·연산과 연관·일반화 등의 관계를 표현하는 **정적 구조 다이어그램**이다. 연산을 표시할 수는 있지만 그 연산이 실행되는 논리적 처리 흐름 자체를 나타내는 것은 아니다. 활동의 진행, 분기, 병렬 처리 등 프로세스의 처리 과정을 도식화하는 것은 주로 **활동 다이어그램**의 역할이다.

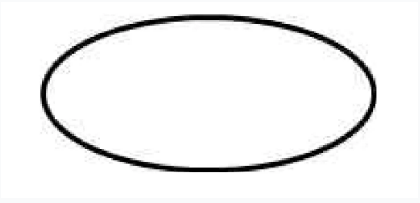
③ O: 상태 다이어그램은 객체 등이 가질 수 있는 **상태와 상태 전이**를 나타낸다. 이벤트가 발생하거나 시간 조건이 충족될 때 어떤 상태로 이동하는지를 표현하며, 필요한 경우 전이를 허용하는 조건과 수행할 동작도 포함한다. 상태를 중심으로 사건에 반응하는 변화를 모델링한다는 점에서 활동의 처리 흐름을 중심으로 하는 도식과 구별된다.

④ O: 컴포넌트 다이어그램은 시스템을 구성하는 **구현 단위와 인터페이스·의존 관계**를 정적으로 나타낸다. 예를 들어 어떤 모듈이 다른 컴포넌트의 서비스를 요구하거나 제공하는지를 표현할 수 있다. 구현 관점의 구성과 연결을 보여 주는 것이며, 실제 실행 중 메시지가 오가는 순서나 작업의 시간 흐름을 보여 주는 것은 아니다.

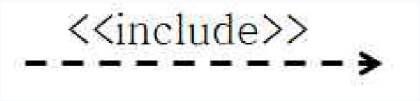
클래스 다이어그램을 프로세스의 논리적인 처리 흐름을 표현하는 도식으로 설명한 ②가 적절하지 않다.

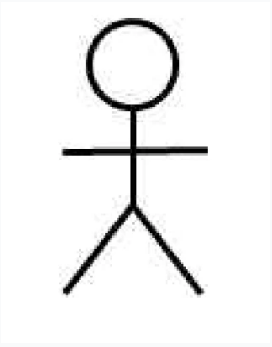
NoteBase

문 18. 사용사례 다이어그램에서 구현(implementation) 대상이 아닌 것은?

① 

② 

③ 

④ 

정답 4번

간단한 해설

사용사례와 포함·확장 관계가 나타내는 기능은 개발 대상 시스템에 구현한다. 사람 모양의 Actor는 **시스템 경계 밖에서 상호작용하는 역할**이므로 그 행위자 자체는 구현 대상에 포함되지 않는다.

NoteBase

선지별 해설

① O: 타원은 **사용사례(Use Case)**를 나타낸다. 사용사례는 시스템이 외부 행위자에게 제공해야 하는 기능과 행위를 기술하므로, 그 내용을 충족하는 시스템 동작이 구현 대상이다.

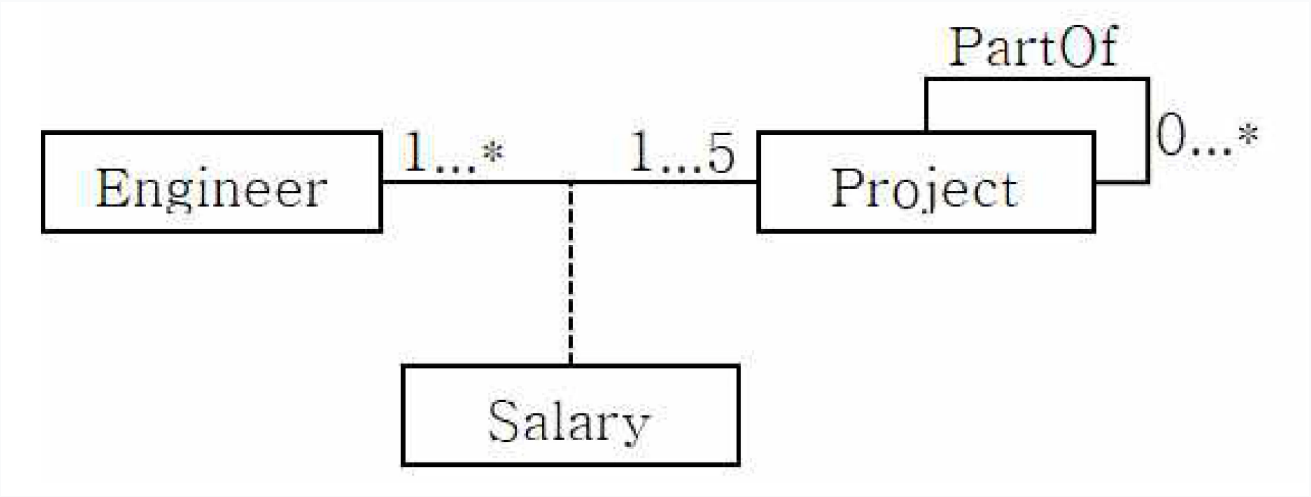
② O: 'extend'가 표시된 점선 화살표는 확장 관계이다. 기본 사용사례에 추가될 행위를 나타내며, 확장 조건이 있다면 그 조건이 충족될 때 확장 지점에서 추가 행위가 수행된다. 따라서 **확장 행위와 그 실행 조건**을 시스템에 구현해야 한다. 화살표 자체를 코드로 만든다는 뜻은 아니며, 화살표는 확장 사용사례에서 확장되는 기본 사용사례를 향한다.

③ O: 'include'가 표시된 점선 화살표는 포함 관계이다. 포함하는 사용사례의 동작 안에 다른 사용사례의 동작을 삽입해 수행한다는 뜻이므로, **포함되는 기능과 이를 사용하는 흐름**이 구현에 반영되어야 한다. 여러 사용사례의 공통 기능을 분리해 재사용하는 데 활용하며, 화살표는 포함하는 사용사례에서 포함되는 사용사례를 향한다.

④ X: 사람 모양 기호는 **시스템 외부의 행위자(Actor)**를 나타낸다. 행위자는 시스템과 상호작용하는 역할이며 사용자뿐 아니라 외부 장치나 다른 시스템일 수도 있다. 개발 대상 시스템과 그 행위자의 접점 및 대응 기능은 구현하지만, 행위자 자체는 해당 시스템의 경계 밖에 있으므로 그 시스템의 구현 대상에 포함되지 않는다.

NoteBase

문 19. 다음 클래스 다이어그램에 대한 설명으로 적절하지 않은 것은?



① 한 명의 Engineer는 최대한 5개의 Project에 참여할 수 있다.

② 하나의 Project는 최소한 한 명 이상의 Engineer를 포함한다.

③ 하나의 Project는 최소한 하나 이상의 Project들을 포함한다.

④ Engineer는 Project에 참여해야만 Salary를 받는다.

정답 3번

간단한 해설

PartOf 관계의 다중성 **0..*는 0개도 허용**한다. 따라서 Project가 반드시 하나 이상의 하위 Project를 포함해야 한다는 설명은 성립하지 않는다.

NoteBase

선지별 해설

연관선 끝의 다중성은 반대편 객체 하나에 연결될 수 있는 해당 끝의 객체 수를 나타낸다.

① O: Engineer 한 명에 연결될 수 있는 Project 수는 **Project 쪽의 1..5**를 읽는다. 따라서 한 명의 Engineer는 최소 1개, 최대 5개의 Project에 참여할 수 있으므로 최대가 5개라는 설명과 일치한다.

② O: Project 하나에 연결될 수 있는 Engineer 수는 **Engineer 쪽의 1..***를 읽는다. 하한 1은 최소 한 명이 필요함을 뜻하고, *는 상한을 특정 수로 제한하지 않음을 뜻한다.

③ X: Project의 PartOf 자기연관에 표시된 **0..*의 하한은 0**이다. 따라서 하위 Project가 하나도 없는 경우도 허용되며, 반드시 하나 이상을 포함해야 하는 것은 아니다. '최소한 하나 이상'을 '0개 이상'으로 고쳐야 한다.

④ O: Salary가 Engineer–Project 연관선에 점선으로 연결되어 있으므로 **연관 클래스**이다. Salary 정보는 Engineer 자체에 독립적으로 붙는 것이 아니라 특정 Engineer가 특정 Project에 참여하는 관계에 결부된다. 따라서 이 모델에서 Salary는 Project 참여 관계를 전제로 한다. 실제 급여의 지급 시점·금액까지 그림이 규정하는 것은 아니다.

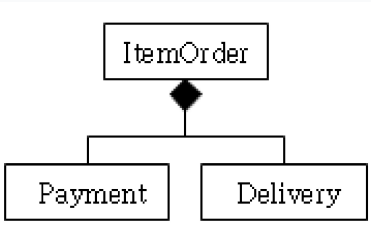
따라서 적절하지 않은 설명은 ③이다.

NoteBase

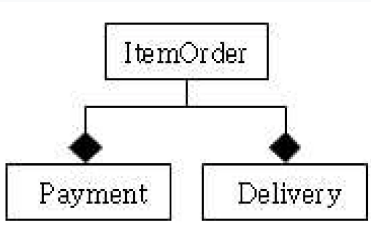
문 20. 다음 ItemOrder 소스코드에 대한 클래스 다이어그램으로 적절한 것은? (단, 주어진 소스 코드 이외의 부분은 가정하지 않는다.)

```
Public class ItemOrder {
    String ItemId;
    int amount;
    ...
    Payment PaymentType;
    Delivery DeliveryType;
    ...
    public ItemOrder(String ItemId, int amount, ...) {
        ...
        Payment payment = new Payment( );
        Delivery delivery = new Delivery( );
        ...
    }
    Public void viewOrder( ) { ... }
    ...
}
```

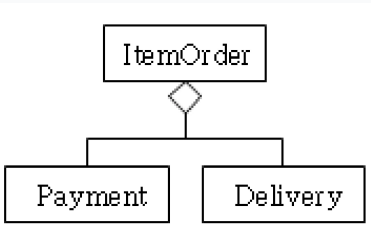
①



②

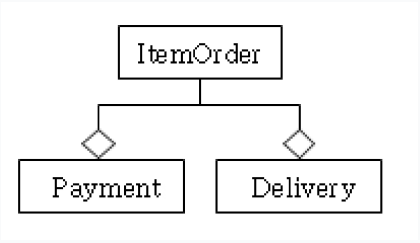


③



NoteBase

④



정답 1번

간단한 해설

ItemOrder 생성자에서 Payment와 Delivery 객체를 직접 생성하는 구조를 이 문항에서는 **합성 관계**로 나타낸다. 채워진 마름모는 **전체인 ItemOrder 쪽**에 둔다.

출제 포인트

코드의 객체 생성 구조를 UML의 전체·부분 관계로 해석하고 마름모의 색과 위치가 맞는지 판단하는 문제이다.

선지별 해설

① O: ItemOrder 쪽에 채워진 마름모가 있다. ItemOrder를 전체, Payment와 Delivery를 부분으로 나타낸 합성 관계이다. 생성자 안의 'new Payment()'와 'new Delivery()'를 이 문항에서 합성의 단서로 해석한 도식과 일치한다.

② X: 채워진 마름모가 Payment와 Delivery 쪽에 있어 두 클래스를 전체로 나타낸다. 전체·부분의 방향이 뒤집혔다. 마름모를 ItemOrder 쪽으로 옮겨야 한다.

③ X: 마름모는 ItemOrder 쪽에 있지만 속이 비어 있어 집약 관계를 나타낸다. 이 문항이 내부 객체 생성 구조에 적용한 관계는 합성이므로 빈 마름모를 채워진 마름모로 바꿔야 한다.

④ X: 빈 마름모가 Payment와 Delivery 쪽에 있다. 집약으로 표시한 관계의 종류와 전체·부분의 방향이 모두 맞지 않는다. ItemOrder 쪽에 채워진 마름모를 두는 합성 관계로 고친다.

오답 포인트

채워진 마름모는 합성, 빈 마름모는 집약을 뜻하며 두 경우 모두 마름모가 붙은 쪽이 전체이다. ②는 색만, ③은 위치만 맞기 때문에 한 요소만 보면 오답을 고르기 쉽다. 합성은 전체가 부분을 소유하는 강한 전체·부분 관계이며 전체의 삭제가 포함된 부분의 삭제로 이어지는 의미를 갖는다. 다만 제시된 코드에는 삭제 처리나 외부 공유 여부가 보이지 않는다. 이 문항의 내부 생성 단서를 해석하되 코드에 없는 동작까지 단정하지 않는다.

NoteBase

다시 나오면 이렇게 판단

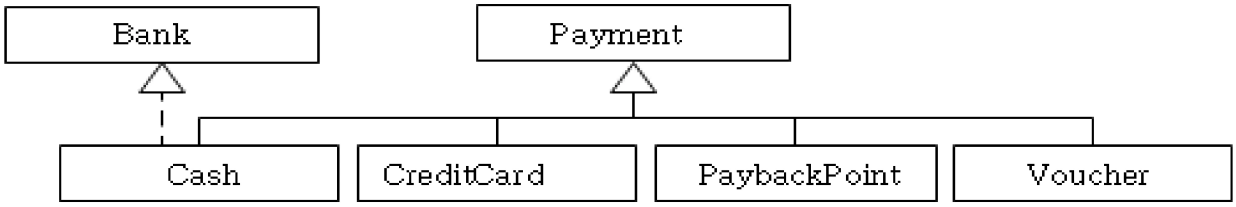
코드에서 다른 객체를 필드로 참조하는지, 내부에서 생성하는지, 외부에서 전달받는지를 확인한다. 소유와 생명주기 조건까지 주어지면 이를 함께 검토해 합성과 집약을 구분한다. 도식에서는 관계의 종류를 나타내는 마름모의 색과 전체를 나타내는 마름모의 위치를 각각 확인한다. 내부의 new 호출 하나만을 모든 코드에서 합성을 확정하는 규칙으로 외우지 않는다.

암기 포인트

합성은 채워진 마름모, 집약은 빈 마름모. 마름모는 전체 쪽에 둔다.

NoteBase

문 21. 다음 클래스 다이어그램을 JAVA 소스코드로 구현 시, 적절하지 않은 것은?



① Public class Payment { ... }

② Public class Bank { ... }

③ Public class Cash extends Payment implements Bank { ... }

④ Public class Voucher extends Payment { ... }

정답 2번

간단한 해설

Cash에서 Bank로 향하는 점선과 속이 빈 삼각형은 인터페이스 구현 관계를 뜻한다. 따라서 **Bank는 class가 아니라 interface로 선언해야 한다**.

NoteBase

선지별 해설

그림의 실선과 속이 빈 삼각형 화살표는 일반화, 점선과 속이 빈 삼각형 화살표는 실체화 관계이다. Payment는 네 하위 클래스의 상위 클래스이며, Bank는 Cash가 구현하는 인터페이스로 해석한다. 아래 O/X는 이 관계를 올바르게 반영했는지를 기준으로 한다. 원문의 공통 표기 Public은 실제 Java 코드에서는 소문자 public으로 작성해야 한다.

① O: Cash, CreditCard, PaybackPoint, Voucher에서 올라오는 일반화 화살표가 Payment를 가리키므로, Payment를 상위 클래스로 선언하는 구조가 맞다. 선언은 public class Payment { ... }와 같이 작성할 수 있다.

② X: Cash에서 Bank로 향하는 **점선과 속이 빈 삼각형은 인터페이스 구현 관계**를 나타낸다. 따라서 Bank를 class로 선언하면 도식의 관계를 반영하지 못한다. public interface Bank { ... }와 같이 인터페이스로 선언해야 한다.

③ O: Cash는 Payment를 상속하는 동시에 Bank를 구현한다. Java에서는 **클래스 상속에 extends, 인터페이스 구현에 implements**를 사용하므로 public class Cash extends Payment implements Bank { ... }라는 선언 구조가 알맞다. 실제 클래스 본문에는 필요한 구현을 제공해야 한다.

④ O: Voucher와 Payment 사이에는 일반화 관계가 있으므로 Voucher는 Payment를 상속한다. 따라서 public class Voucher extends Payment { ... }라는 선언 구조가 맞다. 그림에는 Voucher가 Bank를 직접 구현한다는 관계가 표시되어 있지 않다.

NoteBase

문 22. 다음 JAVA 소스코드를 UML 클래스 다이어그램으로 표현 시, Sedan과 Factory 클래스 간 나타나는 관계로 적절한 것은? (단, 주어진 소스 코드 이외의 부분은 가정하지 않는다.)

```
class Sedan extends Vehicle { ... }
class Factory {
public Vehicle order(int passengers) {
if(passengers < 5)
return new Sedan(passengers);
}
...
}
```

① 합성(composition)

② 구현(realization)

③ 일반화(generalization)

④ 의존(dependency)

정답 4번

간단한 해설

Factory의 order 메서드는 new Sedan(passengers)로 **Sedan 객체를 직접 생성해 사용**한다. 이처럼 한 클래스의 구현이 다른 클래스를 필요로 하는 관계는 의존 관계이다.

NoteBase

핵심 풀이

의존 관계는 한 요소가 자신의 구현이나 동작을 위해 **다른 요소를 사용**하는 관계이다. 주어진 코드에서 Factory의 order 메서드는 승객 수가 5보다 작을 때 new Sedan(passengers)를 실행하여 Sedan 객체를 생성하고 반환한다. 따라서 Factory의 구현은 Sedan 클래스와 그 생성자의 정의를 사용한다.

의존의 방향은 **사용하는 Factory에서 사용되는 Sedan으로** 향한다. UML에서는 일반적으로 점선과 열린 화살촉으로 나타낸다. 반환형이 Vehicle이라고 선언되어 있어도 메서드 본문에서 구체 클래스인 Sedan을 직접 생성한다는 의존이 없어지지는 않는다.

Sedan extends Vehicle은 **Sedan과 Vehicle 사이의 일반화**를 뜻한다. 이 상속 관계의 상대는 Factory가 아니므로, 발문이 묻는 Sedan과 Factory의 관계를 일반화로 판단할 수 없다.

합성은 전체와 부분 사이의 강한 소유·생명주기 관계를 나타낸다. 객체를 생성한다는 사실만으로 합성이 성립하는 것은 아니며, 제시된 코드에는 Factory가 Sedan을 자신의 구성 부분으로 보유·관리한다는 근거가 없다. 구현 관계는 어떤 명세나 인터페이스의 계약을 실현하는 관계인데, 두 클래스 사이에 그러한 선언도 제시되지 않았다.

문제에서 주어진 코드 이외의 부분을 가정하지 말라고 했으므로, 생략된 부분에 소유 필드나 추가적인 상속·구현 관계가 있다고 보충하지 않는다. 승객 수 조건은 객체 생성이 일어나는 분기를 정할 뿐, 확인되는 사용 의존의 종류를 바꾸지 않는다.

따라서 Sedan과 Factory 사이에서 확인되는 관계는 의존 관계인 ④이다.

NoteBase

문 23. 액티비티 다이어그램(activity diagram)에 대한 설명으로 적절하지 않은 것은?

① 포크(fork)는 단일 입력 트랜지션(transition)과 다수 출력 트랜지션(transition)을 가진다.

② 조인(join)은 다중 입력 트랜지션과 단일 출력 트랜지션을 가진다.

③ 스윔레인(swim lane)은 액티비티에 참여하는 클래스를 구분한다.

④ 프레임(frame)은 액티비티를 묶어 조건에 의한 택일을 표현한다.

정답 4번

간단한 해설

조건에 따라 진행 경로 하나를 선택하는 기능은 **결정 노드(decision node)와 가드 조건**으로 표현한다. 프레임은 다이어그램이나 활동의 범위를 둘러 나타내는 틀이므로, 프레임 자체가 조건에 의한 택일을 뜻하지는 않는다.

선지별 해설

① O: 포크(fork)는 **하나의 입력 흐름을 여러 병행 흐름으로 분리**한다. 따라서 입력 트랜지션은 하나이고 출력 트랜지션은 여러 개이다. 조건에 맞는 경로 하나만 고르는 결정 노드와 달리, 분기된 흐름들이 병행하여 진행하도록 한다.

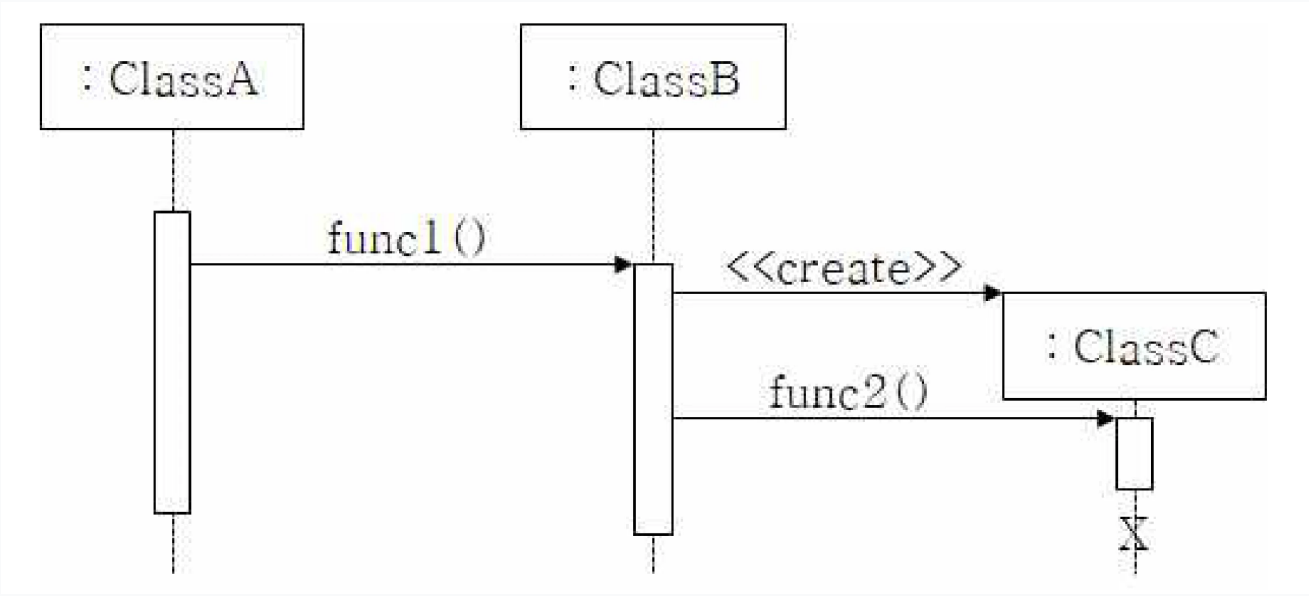
② O: 조인(join)은 여러 입력 흐름을 받아 **병행 흐름을 동기화한 뒤 하나의 흐름으로 연결**한다. 기본적인 AND 동기화에서는 각 입력 흐름이 준비되어야 다음 흐름으로 진행한다. 여러 대안 경로를 동기화 없이 합치는 병합(merge) 노드와 구별된다.

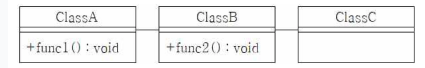
③ O: 스윔레인(swim lane)은 활동을 수행하는 책임 주체별로 영역을 나누어, 각 동작을 어느 클래스가 담당하는지 구분할 수 있게 한다. 클래스 외에 역할·조직 등으로 책임을 나누는 데도 사용하므로, 핵심은 **활동의 수행 책임을 구분**하는 데 있다. 경로가 레인 경계를 넘을 수 있으며, 레인의 위치 자체가 실행 순서를 결정하지는 않는다.

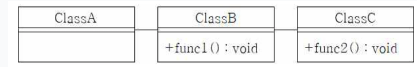
④ X: 프레임(frame)은 다이어그램이나 활동의 범위를 둘러싸서 표시하는 틀이다. 조건에 따라 여러 경로 중 하나를 선택하려면 **마름모 모양의 결정 노드와 출력 경로의 가드 조건**을 사용한다. 예를 들어 승인 여부에 따라 서로 다른 후속 동작으로 나누는 것은 결정 노드의 역할이며, 단순히 활동들을 프레임으로 묶는 것만으로 택일 의미가 생기지는 않는다.

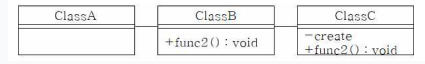
NoteBase

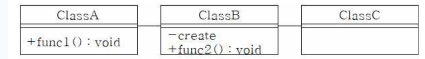
문 24. 다음 시퀀스 다이어그램 구성 및 동작이 가능한 클래스 다이어그램으로 적절한 것은?



① 

② 

③ 

④ 

정답 2번

간단한 해설

시퀀스 다이어그램에서 연산은 **호출 메시지를 받는 객체의 클래스**에 있어야 한다. func1()의 수신자는 ClassB이고, 생성된 ClassC가 func2()를 받으므로 각 연산은 ClassB와 ClassC에 배치된다.

NoteBase

도식 해설

호출 연산의 소속은 메시지를 보내는 객체가 아니라 **메시지를 받는 객체의 클래스**에서 확인한다. 시퀀스 다이어그램의 ':ClassA', ':ClassB', ':ClassC'는 각 클래스의 객체를 나타낸다.

func1() 화살표는 ClassA 객체에서 ClassB 객체로 향한다. 따라서 이 호출을 받는 ClassB에 func1()이 있어야 한다. ClassA가 호출을 시작했다는 이유로 func1()을 ClassA에 배치해서는 안 된다.

ClassB 객체는 <<create>> 메시지로 ClassC 객체를 만든 뒤 그 객체에 func2()를 호출한다. 따라서 **func2()는 ClassC의 연산**이다. ClassC가 생성 지점부터 나타나고 생명선 끝에 X가 있는 것은 생성 후 소멸하는 흐름을 뜻한다. 생성 메시지가 있다고 해서 송신자인 ClassB에 'create'라는 일반 연산을 반드시 적어야 하는 것은 아니다.

①은 func1()을 ClassA에, func2()를 ClassB에 두어 두 호출의 수신 클래스와 맞지 않는다.

②는 ClassB에 +func1(): void, ClassC에 +func2(): void를 두어 두 호출에 맞고, +는 외부에서 접근할 수 있는 공개 연산을 나타낸다.

③은 ClassC에 func2()가 있지만 ClassB에는 func1() 대신 func2()가 표시되어 첫 호출에 맞지 않는다.

④는 func1()이 ClassA에 있고 func2()가 ClassB에 있어 두 연산의 소속이 모두 맞지 않는다. 'create'라는 표기의 유무만으로 선택하지 않고 실제 호출의 수신 대상을 대조해야 한다.

따라서 ClassB가 func1(), ClassC가 func2()를 제공하는 ②가 주어진 시퀀스의 구성과 동작에 대응한다.

NoteBase

문 25. 객체지향 소프트웨어 설계 시 모듈 간 결합도가 강한 것부터 약한 순서로 적절하게 나열한 것은?

가. 모듈 간 상호 교류 시, 단순 타입의 매개변수를 교환하는 경우 발생되는 의존
나. 다른 모듈의 자료 값을 수정하거나, 내부로 분기하는 경우 발생하는 의존
다. 외부 변수로 선언된 데이터를 서로 다른 모듈에서 참조함으로써 발생되는 의존
라. 여러 모듈이 전역 변수를 참조하는 경우 발생 가능성이 증가하는 의존
마. 레코드, 배열, 구조체 형태의 복합 자료를 모듈 간 선별적으로 공유 시 발생되는 의존
바. 다른 모듈 내의 기능을 제어하거나 두 모듈이 분리된 경우 발생되는 의존

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

정답 2번

간단한 해설

결합도는 모듈 사이의 의존 정도이며, **상대 내부를 직접 침범**할수록 강하고 필요한 값만 명시적으로 교환할수록 약하다. 전역 공유 상태, 외부 선언, 제어 정보, 복합 자료구조에 대한 의존을 구별하면 각 항목의 위치가 정해진다.

NoteBase

올바른 순서

나 → 라 → 다 → 바 → 마 → 가

나. 내용 결합도: 다른 모듈의 내부 자료를 직접 수정하거나 내부 위치로 분기하여 **모듈의 내부 구현을 침범**하는 경우이다. 정상적인 인터페이스를 우회하므로 내부 수정이 다른 모듈에 바로 영향을 미친다. 제시된 종류 중 의존이 가장 강하다.

라. 공통 결합도: 여러 모듈이 **공통의 전역 데이터 영역**을 참조하는 경우이다. 한 모듈이 공유 상태를 바꾸면 다른 여러 모듈의 동작에 영향을 줄 수 있고, 변경 주체와 영향 범위를 추적하기 어렵다. 상대 내부로 직접 들어가지는 않지만 공유 범위가 넓어 강한 결합으로 분류된다.

다. 외부 결합도: 이 문항의 전통적 분류에서는 **외부에 선언된 데이터 항목을 함께 참조**하는 의존을 뜻한다. 공통 데이터 영역 전체에 광범위하게 의존하는 경우와 구별하여, 외부로 공개된 개별 데이터 선언에 대한 의존으로 본다. 공통 결합보다 범위가 제한되지만 여러 모듈이 같은 외부 선언의 변경에 영향을 받을 수 있다.

바. 제어 결합도: 한 모듈이 제어 정보 등을 전달하여 **다른 모듈이 수행할 기능이나 실행 경로**를 결정하는 경우이다. 단순한 처리 대상 데이터뿐 아니라 상대의 처리 방식까지 알아야 하므로 자료나 스탬프 결합보다 강하다. ‘두 모듈이 분리된 경우’라는 표현만으로는 제어 결합이 성립하지 않으며, 이 항목에서는 다른 모듈의 기능을 제어한다는 단서가 분류의 근거이다.

마. 스탬프 결합도: 레코드·배열·구조체 같은 **복합 자료구조**를 모듈 사이에 전달하거나 공유하는 경우이다. 일부 항목만 필요하더라도 자료구조의 형식에 의존할 수 있어 구조 변경이 영향을 준다. 상대의 실행을 지시하는 제어 결합보다는 약하지만 필요한 단순 값만 교환하는 자료 결합보다는 강하다.

가. 자료 결합도: 모듈 간에 **필요한 단순 데이터 값**을 매개변수로 교환하는 경우이다. 상대 모듈의 내부 구조나 제어 논리를 알 필요가 적고 입출력 관계가 명시적이므로, 제시된 종류 중 결합도가 가장 약하다.

NoteBase

문 26. 객체지향 소프트웨어 설계 시 모듈 내 응집도가 가장 높은 것과 가장 낮은 것을 적절하게 짝지어진 것은?

① 절차적(procedural) 응집도 - 논리적(logical) 응집도

② 기능적(functional) 응집도 - 우연적(coincidental) 응집도

③ 대화적(communicational) 응집도 - 절차적(procedural) 응집도

④ 논리적(logical) 응집도 - 우연적(coincidental) 응집도

정답 2번

간단한 해설

모듈의 모든 요소가 하나의 명확한 기능을 위해 모인 **기능적 응집도**가 가장 높다. 서로 관련 없는 처리들이 임의로 묶인 **우연적 응집도**가 가장 낮다.

선지별 해설

응집도는 한 모듈 내부의 구성 요소들이 서로 얼마나 긴밀하게 관련되는지를 나타낸다. 이 문항에서는 전통적인 응집도 분류를 기준으로 각 짝의 앞부분이 최고, 뒷부분이 최저에 해당하는지 비교한다.

① 절차적 응집도는 처리들이 정해진 실행 순서를 따른다는 이유로 묶인 경우이고, 논리적 응집도는 입력·출력처럼 논리적으로 비슷한 종류의 처리를 묶은 경우이다. 실행 순서만 공유하는 절차적 응집도가 가장 높은 것은 아니며, 논리적 응집도보다 관련성이 더 낮은 우연적 응집도가 있으므로 최저값도 아니다.

② 기능적 응집도는 **모든 구성 요소가 하나의 명확한 기능 수행에 기여**하는 경우이다. 이 문항의 전통적 응집도 분류에서 가장 높다. 우연적 응집도는 **서로 뚜렷한 관련이 없는 처리들이 임의로 함께** 모인 경우이므로 가장 낮다. 따라서 두 극단을 모두 올바르게 제시한 짝이다.

③ 대화적 응집도는 여러 처리가 같은 입력 자료를 사용하거나 같은 출력 자료를 만드는 등 자료를 공유하기 때문에 묶인 경우이다. 단순히 실행 순서를 공유하는 절차적 응집도보다 높게 평가되지만, 하나의 기능에 모든 요소가 집중되는 기능적 응집도보다는 낮다. 절차적 응집도도 우연적 응집도보다 높으므로 최고·최저의 짝이 아니다.

④ 논리적 응집도는 관련 처리를 같은 범주로 묶었다는 공통점은 있지만, 하나의 기능을 완결하기 위해 모든 요소가 긴밀하게 협력하는 수준은 아니다. 따라서 가장 높은 응집도가 아니다. 우연적 응집도를 최저로 제시한 뒷부분은 맞지만, 앞부분이 최고 조건을 충족하지 못한다.

모듈 내부 요소의 기능적 관련성을 기준으로 비교하면 최고는 기능적 응집도, 최저는 우연적 응집도이므로 두 조건을 함께 충족하는 것은 ②이다.

NoteBase

문 27. 다음 C 언어 소스코드에서 swap과 sort 함수 간 결합도와 응집도는?

```
void swap(int* x, int* y) {
    int temp = *x;
    *x = *y;
    *y = temp;
}
void sort(int* nums, int size) {
    ...
            if(*(nums + i)  < *(nums + j))
                swap(nums + i , nums + j));
    ...
}
```

① 결합도: 내용(content) 결합, 응집도: 기능적(functional) 응집

② 결합도: 내용(content) 결합, 응집도: 절차적(procedural) 응집

③ 결합도: 외부(external) 결합, 응집도: 기능적(functional) 응집

④ 결합도: 외부(external) 결합, 응집도: 절차적(procedural) 응집

정답 1번

간단한 해설

swap은 전달받은 주소를 통해 sort가 다루는 배열 원소를 직접 바꾼다. 이 문항에서는 이 데이터 변경을 **내용 결합**으로 분류한다. 각 함수의 내부 처리는 각각 값 교환과 정렬이라는 단일 목적에 집중하므로 **기능적 응집**에 해당한다.

출제 포인트

함수 사이의 데이터 접근 방식으로 결합도를, 각 함수 내부의 처리 목적을 기준으로 응집도를 구분하는 문제이다.

NoteBase

핵심 개념

결합도는 서로 다른 모듈이 얼마나 의존하는지를 나타내고 응집도는 한 모듈 안의 요소들이 얼마나 긴밀하게 하나의 목적에 모이는지를 나타낸다. 응집도는 함수 사이가 아니라 각 함수 내부를 평가한다.
내용 결합: 한 모듈이 다른 모듈의 데이터나 내부 처리에 직접 접근하거나 이를 변경하는 관계이다. 이 문항은 swap이 sort가 다루는 원소를 직접 변경한다는 점에 초점을 둔다.
외부 결합: 모듈들이 외부에서 정한 통신 규약, 파일 형식, 장치 인터페이스 등에 함께 의존하는 관계이다. 다른 함수를 호출한다는 뜻의 외부가 아니다.
기능적 응집: 모듈 내부의 모든 처리가 하나의 명확한 기능을 수행하기 위해 모인 경우이다. swap의 세 대입문은 모두 두 값의 교환에 필요한 처리이다.
절차적 응집: 서로 다른 처리들이 특정 실행 순서를 따라야 한다는 이유로 모인 경우이다. 명령문에 실행 순서가 있다는 사실만으로 절차적 응집이 되지는 않는다.

선지별 해설

① O
sort는 swap(nums + i, nums + j)로 두 원소의 주소를 넘긴다. swap의 *x = *y와 *y = temp는 그 주소에 있는 원소를 바꾸므로 sort가 다루는 배열에도 변경이 반영된다. 이 문항에서는 이를 내용 결합으로 본다. swap은 값 교환, sort는 정렬이라는 각자의 기능에 집중하므로 기능적 응집으로 분류한다.

② X
내용 결합이라는 분류는 이 문항의 판단에 맞지만 절차적 응집이라는 부분이 틀렸다. swap의 임시 저장과 두 대입은 순서만으로 묶인 별개의 작업이 아니라 값 교환을 완성하는 처리이다. 응집도는 기능적 응집으로 고친다.

③ X
기능적 응집은 맞지만 외부 결합이 틀렸다. 제시된 관계는 외부 규약이나 장치 형식에 대한 공동 의존이 아니라 배열 원소의 직접 변경이다. 이 문항의 결합도 분류는 내용 결합으로 고친다.

④ X
외부 결합과 절차적 응집이 모두 틀렸다. 외부 규약에 대한 의존은 제시되지 않았고 함수 내부의 처리도 각각 값 교환과 정렬이라는 단일 기능을 수행한다. 이 문항에서는 내용 결합과 기능적 응집으로 고친다.

NoteBase

오답 포인트

다른 함수를 호출한다는 이유로 외부 결합을 고르거나, 대입문이 순서대로 실행된다는 이유로 절차적 응집을 고르기 쉽다. 외부 결합의 외부는 외부 환경이나 규약을 뜻한다. 실행 순서가 필요한 처리라도 모두 하나의 기능을 완성한다면 기능적 응집이다. 포인터를 쓴다는 사실만으로 모든 코드를 내용 결합으로 분류해서도 안 된다. 일반적인 자료 결합은 필요한 개별 데이터를 정상적인 매개변수 인터페이스로 주고받는 관계이다. 내용 결합은 정상적인 인터페이스의 범위를 넘어 다른 모듈의 내부 코드나 비공개 데이터에 직접 의존하는 관계이다. 이 코드에서 swap은 매개변수로 받은 두 정수의 주소만 사용하며 sort의 인덱스 i, j나 정렬 절차에는 의존하지 않는다. 따라서 매개변수 인터페이스를 기준으로는 이 관계를 자료 결합으로 해석한다. 여기서는 배열 원소를 직접 바꾸는 동작을 내용 결합으로 해석하는 문항의 판단을 따른다.

다시 나오면 이렇게 판단

결합도는 두 모듈이 어떤 데이터나 규약을 통해 의존하는지 확인한다. 응집도는 한 모듈의 처리가 하나의 기능을 완성하는지, 서로 다른 작업을 순서 때문에 묶었는지 확인한다. 코드에서는 인수의 전달 방식과 실제 변경 대상을 함께 살핀다.

암기 포인트

결합도는 모듈 사이의 의존, 응집도는 모듈 내부의 목적이다. 외부 규약 의존은 외부 결합, 한 기능의 완성은 기능적 응집이다.

NoteBase

문 28. 사용자 인터페이스 설계 원리 중 아래에 기술된 지침을 모두 만족하는 UI 설계개념으로 적절한 것은?

가. 초보와 숙련자 모두가 쉽게 배우고, 사용할 수 있게 한다.
나. 소프트웨어 사용자를 즉각적으로 만족시켜야 한다.
다. 사용자 입력을 최소화하고 불필요한 입력은 제외한다.
라. 사용자 액션에 적절하고 의미있는 피드백을 제공한다.

① 마법사(wizard)
② 사이트 이동 경로(breadcrumbs)
③ 메타포(metaphor)
④ GUI 프레임워크(GUI Framework)

정답 1번

간단한 해설

마법사는 복잡한 작업을 단계별로 안내하여 사용자가 필요한 정보만 입력하고 결과를 확인하도록 돕는다. 쉬운 안내, 입력 부담의 감소, 적절한 피드백을 함께 설계하는 것이 핵심이다.

NoteBase

대응 관계 해설

마법사는 하나의 작업을 여러 단계로 나누고, 각 단계에서 필요한 설명과 입력 항목을 제시하여 완료까지 안내하는 UI이다. 제시된 지침은 이러한 안내 과정을 사용하기 쉽게 설계하는 조건으로 연결된다.

가. 학습과 사용의 용이성: 초보자는 단계별 설명을 따라 작업을 익히고, 숙련자는 익숙한 선택과 기본값을 활용해 빠르게 진행할 수 있도록 한다. 숙련자에게도 불필요한 단계를 강제하면 효율이 떨어지므로 **사용자 수준에 맞는 안내**가 필요하다.

나. 즉각적인 만족: 사용자가 시작부터 필요한 작업을 수행하고 진행 결과를 확인하도록 하여 목표 달성에 대한 성취감을 제공한다. 이는 모든 처리를 즉시 끝낸다는 뜻이 아니라, 불필요한 도입 화면과 대기를 줄이고 유용한 반응을 제공한다는 뜻이다.

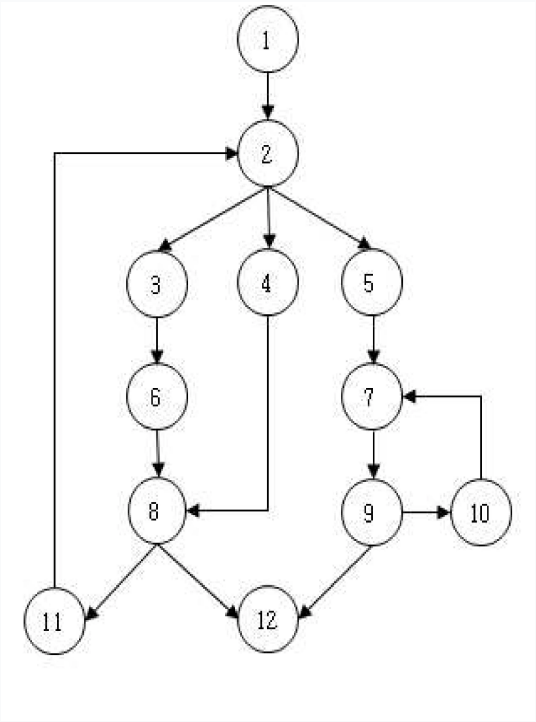
다. 입력의 최소화: 기본값과 이전 입력의 재사용 등을 활용하고, 현재 선택과 관계없는 질문은 생략한다. 사용자는 **작업에 필요한 정보만 입력**하며 다음 단계로 진행하게 된다.

라. 의미 있는 피드백: 선택의 반영 여부, 입력 오류, 현재 진행 상태와 완료 결과를 알려 준다. 사용자는 자신의 액션이 어떤 결과를 냈는지 이해하고 필요한 수정을 할 수 있다.

다른 개념과의 구분: 사이트 이동 경로는 사이트 구조 속 현재 위치와 상위 경로를 보여 주는 탐색 수단이다. 메타포는 익숙한 사물·행동에 빗대어 기능을 이해시키는 표현 방식이며, GUI 프레임워크는 그래픽 인터페이스를 구현하기 위한 공통 개발 기반이다. 단계별 작업 안내에 위 지침을 함께 적용하는 개념은 마법사이므로 ①에 해당한다.

NoteBase

문 29. 다음 제시된 그래프(흐름도)에서 McCabe의 순환복잡도는?



① 5

② 6

③ 7

④ 8

정답 1번

간단한 해설

하나로 연결된 흐름 그래프이므로 **간선 수 − 노드 수 + 2**를 적용한다. 간선 15개와 노드 12개에서 순환복잡도는 15 − 12 + 2 = 5이다.

NoteBase

계산 과정

하나로 연결된 단일 진입·단일 종료 흐름 그래프에서는 McCabe 순환복잡도를 $V(G) = E - N + 2$로 계산한다. E는 간선 수, N은 노드 수이며, 이 값은 기본적인 독립 경로의 수를 나타낸다.

그림의 노드는 1번부터 12번까지이므로 N = 12이다. 나가는 화살표를 기준으로 세면 2번에서 3개, 8번과 9번에서 각각 2개가 나온다. 나머지 1·3·4·5·6·7·10·11번에서는 각각 1개씩 나오고, 종료 노드 12번에서는 나오지 않는다. 따라서 E = 3 + 2 + 2 + 8 = 15이다. 10→7과 11→2의 되돌아가는 간선도 포함한다.

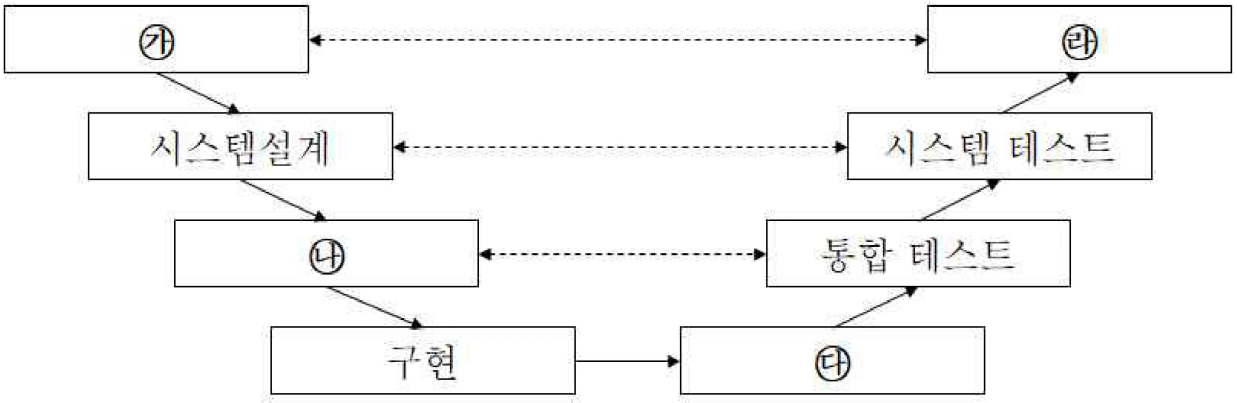
공식에 대입하면 다음과 같다.
$V(G) = 15 - 12 + 2 = 5$

분기 수로도 확인할 수 있다. 2번은 **3갈래 분기이므로 복잡도에 2를 더하고**, 8번과 9번의 2갈래 분기는 각각 1을 더한다. 기본 경로 1을 합하면 2 + 1 + 1 + 1 = 5이다. '분기 노드 수 + 1'은 모든 분기가 2갈래인 경우에 바로 적용할 수 있으므로, 2번을 단순히 1로 세면 안 된다.

따라서 순환복잡도는 5이며, 선택지 ①에 해당한다.

NoteBase

문 30. V&V 모델의 ㉮ ~ ㉲에 들어갈 내용으로 적절한 것은?



① ㉮: 상세설계, ㉯: 요구분석, ㉰: 성능 테스트, ㉲: 복구 테스트

② ㉮: 상세설계, ㉯: UI설계, ㉰: 보안 테스트, ㉲: 성능 테스트

③ ㉮: 요구분석, ㉯: UI설계, ㉰: 성능 테스트, ㉲: 인수 테스트

④ ㉮: 요구분석, ㉯: 상세설계, ㉰: 단위 테스트, ㉲: 인수 테스트

정답 4번

간단한 해설

구현 직후에는 **단위 테스트**를 하고, 이를 결합한 통합·시스템 테스트를 거쳐 사용자 요구의 충족 여부를 **인수 테스트**로 확인한다. 왼쪽의 시스템설계 앞은 요구분석, 구현 바로 앞은 상세설계에 해당한다.

NoteBase

도식 해설

이 그림은 왼쪽에서 요구와 설계를 구체화하여 구현으로 내려가고, 오른쪽에서 개별 단위부터 전체 시스템과 사용자 수용 수준으로 시험 범위를 넓히는 V 모형이다. 수평 점선은 개발 단계의 산출물과 시험 단계의 대응을 표시한다. 단계 이름과 세분 방식은 도식에 따라 달라질 수 있으므로, 여기서는 실제 표시된 위치와 연결을 기준으로 해석한다.

㉮ 요구분석: 그림의 왼쪽 맨 위에 있으며 시스템설계보다 앞선다. 사용자가 필요로 하는 기능과 제약을 정리하여 **무엇을 만들어야 하는지**를 명확히 하는 단계이다. 이 요구는 반대편 마지막 단계에서 완성된 시스템을 받아들일지를 판단하는 기준이 된다.

㉯ 상세설계: 왼쪽에서 시스템설계 다음, 구현 직전에 놓여 있다. 전체 구조를 정한 뒤 모듈의 처리 논리, 자료와 인터페이스 등을 **코드로 구현할 수 있는 수준**으로 구체화하는 단계이다. 제시된 간략 도식에서는 이 설계 내용과 통합 테스트가 점선으로 연결되어 있으며, 설계한 구성 요소들의 연결이 구현 결과에서 제대로 작동하는지도 확인한다.

㉰ 단위 테스트: 구현에서 바로 이어지고 통합 테스트보다 앞선 오른쪽 아래 단계이다. 함수·클래스·모듈 등 **개별 구현 단위의 동작**을 먼저 확인한다. 그다음 통합 테스트에서 단위들을 결합했을 때의 인터페이스와 상호작용을 검사하므로 이 위치에 단위 테스트가 들어간다.

㉱ 인수 테스트: 시스템 테스트 다음의 오른쪽 맨 위에 있고, 요구분석과 점선으로 연결된다. 완성된 시스템이 사용자 요구와 합의된 인수 기준을 충족하여 **사용자가 받아들일 수 있는지**를 확인하는 단계이다. 개별 부품의 결함보다 사용자 관점의 수용 가능성에 초점이 있다.

따라서 ㉮ 요구분석, ㉯ 상세설계, ㉰ 단위 테스트, ㉱ 인수 테스트로 연결되며 ④와 일치한다.

NoteBase

문 31. 블랙박스(black box) 테스트 기법으로 적절하지 않은 것은?

① 조건 커버리지(condition coverage) 기법
② 동등 분할(equivalence partitioning) 기법
③ 경곗값 분석(boundary value analysis) 기법
④ 원인-결과 그래프(cause-effect graph) 기법

정답 1번

간단한 해설

조건 커버리지는 코드 내부의 개별 조건이 참과 거짓을 갖는지를 검사하므로 **화이트박스 테스트**에 해당한다. 블랙박스 테스트는 내부 코드 구조보다 명세의 입력·출력과 외부 동작을 기준으로 한다.

선지별 해설

아래 O/X는 각 항목이 '블랙박스 테스트 기법에 해당한다'는 판단이다.

① X: 조건 커버리지는 코드의 결정식에 포함된 **각 개별 조건이 참과 거짓을 적어도 한 번씩** 갖도록 검사하는 화이트박스 테스트 기준이다. 내부 논리 구조를 기준으로 테스트를 설계하므로 블랙박스 기법으로 분류하지 않는다. 개별 조건을 다루는 조건 커버리지와 결정식 전체의 참·거짓을 다루는 결정 커버리지도 구별해야 한다.

② O: 동등 분할은 명세상 동일한 방식으로 처리될 것으로 기대되는 입력들을 동등 클래스로 나누고, 각 클래스의 대표값으로 검사하는 블랙박스 기법이다. 유효한 입력뿐 아니라 유효하지 않은 입력도 구분하여, 모든 값을 일일이 시험하는 부담을 줄인다.

③ O: 경곗값 분석은 입력 범위나 동등 클래스의 **경계와 그 바로 안팎**을 중심으로 테스트하는 블랙박스 기법이다. 범위의 최솟값·최댓값과 인접값 등은 비교 연산이나 범위 처리 오류가 드러나기 쉬운 지점이므로 집중적으로 확인한다.

④ O: 원인-결과 그래프는 명세의 입력 조건을 원인, 출력이나 시스템 반응을 결과로 놓고 논리 관계와 제약을 그래프로 표현하여 테스트를 도출하는 블랙박스 기법이다. 여러 입력 조건의 조합이 결과에 미치는 영향을 다루며, 프로그램 내부의 제어 흐름 그래프를 분석하는 기법과 다르다.

따라서 블랙박스 기법으로 적절하지 않은 것은 ①이다.

NoteBase

문 32. 다음 빈칸에 들어갈 단어를 적절하게 짝지어진 것은?

소프트웨어 개발 절차에서, (㉮)는/은 개발자 관점에서 사용자의 요구사항 명세에 따라 개발되었는지 과정의 준수 여부와 결과물의 적정성을 검토하고, (㉯) 는/은 개발된 제품이 사용자의 요구사항에 부합하는지를 검토하는 활동이다.

① ㉮: 검증(verification), ㉯: 검사(validation)

② ㉮: 인증(authentication), ㉯: 검증(verification)

③ ㉮: 검사(validation), ㉯: 인증(authentication)

④ ㉮: 검사(validation), ㉯: 검증(verification)

정답 1번

간단한 해설

명세대로 올바르게 개발했는지를 보는 것은 **verification**이고, 만들어진 제품이 사용자의 실제 목적에 맞는지를 보는 것은 **validation**이다. 두 빈칸은 각각 명세 준수와 사용자 요구 충족이라는 서로 다른 판단 기준을 가리킨다.

빈칸별 해설

㉮ 검증(verification): 개발 과정의 산출물이 정해진 요구사항 명세·설계·기준을 올바르게 충족하는지 확인한다. 지문의 '개발자 관점', '요구사항 명세에 따라 개발', '과정의 준수 여부와 결과물의 적정성'이 단서이다. 핵심 질문은 **명세에 맞게 제품을 만들고 있는가**이다. 문서·코드 검토뿐 아니라 명세와 실제 동작을 대조하는 시험도 검증에 활용할 수 있다.

㉯ 검사(validation): 개발된 제품이 의도된 사용 환경에서 사용자의 실제 요구와 목적을 충족하는지 확인한다. 지문의 '개발된 제품이 사용자의 요구사항에 부합'이 이 관점에 대응한다. 핵심 질문은 **사용자가 필요로 하는 제품을 만들었는가**이다. 명세 자체가 사용자 의도를 잘못 담았다면 그 명세를 충실히 구현했더라도 이 판단을 통과하지 못할 수 있다.

인증(authentication)은 사용자나 시스템 등이 주장하는 신원을 확인하는 개념이다. 개발 산출물의 명세 적합성이나 제품의 사용 목적 충족 여부를 구분하는 두 빈칸에는 해당하지 않는다. 이 문제의 '검사'는 괄호에 제시된 validation의 의미로 읽는다.

따라서 ㉮는 검증(verification), ㉯는 검사(validation)이며 ①의 짝과 일치한다.

NoteBase

문 33. ISO/IEC 25010 국제표준모델에서 제시하고 있는 제품 품질의 신뢰성 평가 항목으로 적절하지 않은 것은?

① 가용성(availability)

② 회복성(recoverability)

③ 결함허용성(fault-tolerance)

④ 시험가능성(testability)

정답 4번

간단한 해설

시험가능성은 유지보수성에 속한다. 가용성·회복성·결함허용성은 시스템을 필요한 때 이용하고, 결함이나 장애에 대응하여 기능을 유지·회복하는 신뢰성의 하위 특성이다.

선지별 해설

제품 품질의 신뢰성은 시스템이 정해진 조건과 기간 동안 요구된 기능을 수행하는 정도이다. ISO/IEC 25010:2011의 신뢰성 하위 특성에는 성숙성, 가용성, 결함허용성, 회복성이 포함된다. 아래 O/X는 각 항목이 신뢰성에 속하는지를 뜻한다.

① O: 가용성은 사용이 필요할 때 시스템이 작동하고 접근 가능한 정도이다. 즉, **필요한 시점에 이용할 수 있는가**를 평가하므로 신뢰성에 포함된다.

② O: 회복성은 중단이나 실패가 발생했을 때 영향을 받은 데이터를 복구하고 시스템을 원하는 상태로 되돌릴 수 있는 정도이다. **장애 후 복구**에 관한 특성이므로 신뢰성에 포함된다.

③ O: 결함허용성은 하드웨어나 소프트웨어에 결함이 있어도 시스템이 의도한 대로 동작하는 정도이다. 결함이 발생하지 않게 하는 것과 달리 **결함이 있어도 기능을 유지**하는 데 초점이 있다.

④ X: 시험가능성은 시험 기준을 정하고 그 기준의 충족 여부를 확인하는 시험을 효과적·효율적으로 수행할 수 있는 정도이다. 이는 **유지보수성의 하위 특성**이다. 시험을 통해 신뢰성을 확인할 수 있다는 사실과 시험가능성 자체의 품질 분류는 구별해야 한다.

NoteBase

문 34. ISO/IEC 9126 소프트웨어 품질 평가 모델에서 정의한 주 품질 특성에 속하지 않는 것은?

① 기능성(functionality)

② 신뢰성(reliability)

③ 사용성(usability)

④ 보안성(security)

정답 4번

간단한 해설

ISO/IEC 9126에서 보안성은 독립된 주 품질 특성이 아니라 **기능성의 하위 특성**이다. 보안의 중요성 여부가 아니라 품질 모델 안의 분류 수준을 구분하는 문제이다.

선지별 해설

ISO/IEC 9126의 주 품질 특성은 기능성·신뢰성·사용성·효율성·유지보수성·이식성의 여섯 가지이다. 아래 O/X는 각 항목이 이 주 품질 특성에 속하는지를 나타낸다.

① O: 기능성은 명시적·묵시적 요구를 충족하는 기능을 제공하는 능력이다. 사용자가 필요로 하는 기능을 적절하고 정확하게 수행하는가를 다루며, 주 품질 특성에 속한다.

② O: 신뢰성은 정해진 조건에서 일정 기간 요구되는 성능을 유지하는 능력이다. 오류 발생 시 기능을 유지하거나 장애 후 회복하는 능력 등과 관련된 주 품질 특성이다.

③ O: 사용성은 사용자가 소프트웨어를 이해하고 배우며 조작하기 쉽게 하는 능력이다. 필요한 기능이 존재하는가를 묻는 기능성과 달리, **그 기능을 얼마나 이해하고 사용하기 쉬운가**에 초점을 둔다.

④ X: 보안성은 정보나 데이터에 대한 허가되지 않은 접근을 막고, 권한 있는 사용자의 접근을 보장하는 능력이다. ISO/IEC 9126에서는 **기능성의 하위 특성**으로 분류되므로 독립된 주 품질 특성에는 속하지 않는다. 보안성을 품질 평가에서 제외한다는 뜻은 아니다.

따라서 주 품질 특성에 속하지 않는 항목은 ④이다.

NoteBase

문 35. 부정적 위험에 대한 대응방안 중 회피 전략으로 적절한 것은?

① 대외 정보시스템 간 연계 작업 혼선으로 인한 위험을 감지하여 관련된 모든 작업의 기간과 일정을 조정하였다.

② 태풍 등 계절적 영향으로 장비 조달 및 설치의 위험이 있어 해당 범위에 한정된 보험에 가입하였다.

③ 기술적 위험이 높은 사업의 성격을 반영하여 위험 발생 확률과 영향도 기준을 수용 가능 한계로 하향 조정하였다.

④ 위험을 제거하는 것이 어렵다고 판단하여 프로젝트 계획 변경 없이 상황을 지속적으로 모니터링한다.

정답 1번

간단한 해설

회피는 **계획을 바꾸어 위협을 없애거나 그 영향에서 프로젝트를 보호**하는 전략이다. 연계 작업의 혼선을 피하도록 작업 기간과 일정을 조정한 사례가 이에 대응한다.

선지별 해설

이 문항은 각 사례가 '회피 전략'에 해당하는지를 묻는다. O/X는 해당 전략의 적합성을 나타내며, 다른 대응 전략 자체가 잘못된 관리 방식이라는 뜻은 아니다.

① O: 이 사례의 일정 조정은 연계 작업의 혼선이 생기는 기존 계획을 바꾸어 해당 위험을 피하려는 조치로 해석한다. **위협을 없애거나 그 영향에서 프로젝트를 보호하도록 계획을 변경**하는 것이 회피이다. 다만 모든 일정 변경이 자동으로 회피가 되는 것은 아니며, 위험을 남겨 둔 채 발생 가능성이나 피해만 줄이는 일정 조정은 완화에 해당할 수 있다.

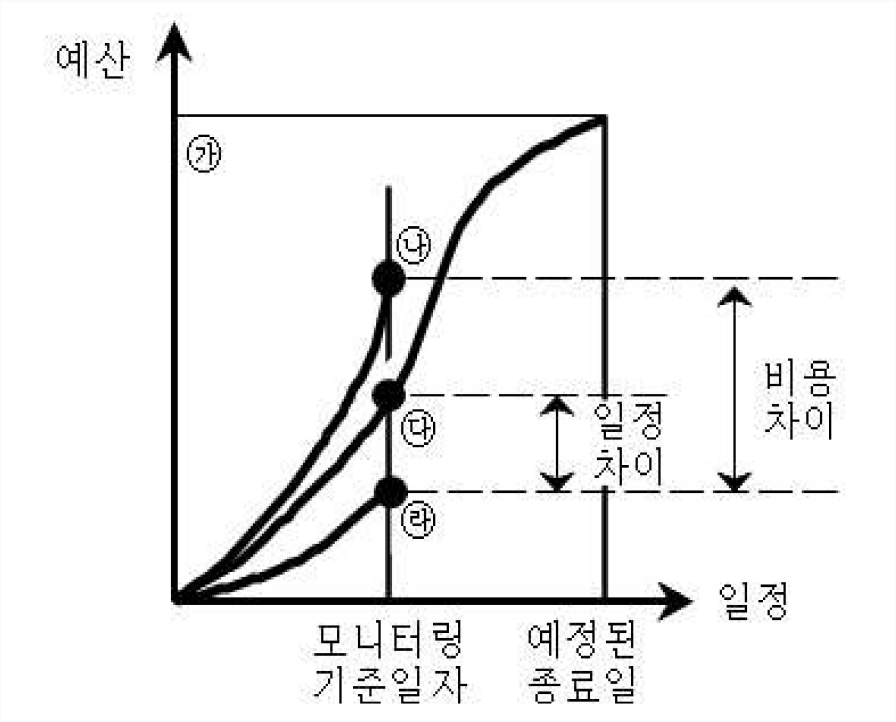
② X: 보험 가입은 보장 범위 안의 손실 부담을 보험자에게 옮기는 **전가** 전략이다. 태풍이나 조달·설치 차질 자체가 없어지는 것은 아니므로 회피와 구별된다.

③ X: 위험의 발생 확률이나 영향도를 수용 가능한 수준까지 실제로 낮추는 전략은 **완화**이다. 원문처럼 '기준'을 조정했다는 표현만으로 실제 위험이 감소했다고 단정할 수는 없지만, 어느 경우에도 위협 자체를 제거하는 회피를 설명한 것은 아니다. 평가 기준을 바꾸는 것과 위험을 줄이는 조치는 구별해야 한다.

④ X: 위험을 인식하면서 계획 변경 없이 상황을 관찰하는 것은 **수용** 전략이다. 위험을 무시한다는 뜻이 아니라, 제거 조치를 취하지 않은 채 감시하고 필요한 대응 여부를 판단하는 것이다. 특히 제시된 내용은 소극적 수용에 해당한다.

NoteBase

문 36. 다음 EVA(Earning Value Analysis) 개념도에서 ㉮ ~ ㉲에 들어갈 지표명이 올바르게 나열된 것은?



① ㉮: BAC, ㉯: AC, ㉰: EV, ㉱: PV

② ㉮: EAC, ㉯: EV, ㉰: AC, ㉱: PV

③ ㉮: BAC, ㉯: AC, ㉰: PV, ㉱: EV

④ ㉮: EAC, ㉯: PV, ㉰: AC, ㉱: EV

정답 3번

간단한 해설

예정 종료일의 계획 총예산은 **BAC**이다. 기준일의 비용 차이는 **AC와 EV**, 일정 차이는 **PV와 EV** 사이에 표시되므로 ㉯는 AC, ㉰는 PV, ㉱는 EV이다.

NoteBase

도식 해설

세로축은 금액으로 표현한 예산·비용, 가로축은 일정이다. 모니터링 기준일자에서 실제 지출, 계획된 작업의 예산가치, 완료한 작업의 예산가치를 비교한다. 단순히 곡선이 높다는 이유가 아니라 **계획 곡선의 종점과 차이 표시의 연결**로 지표를 판독해야 한다.

㉮는 BAC(Budget at Completion)이다. 계획 곡선이 예정된 종료일에 도달하는 총예산을 나타낸다. EAC(Estimate at Completion)는 현재 실적과 전망을 반영하여 새로 예상한 완료 총비용이므로, 원래 계획의 총예산인 BAC와 구별된다.

㉯는 AC(Actual Cost)이다. 기준일까지 실제로 투입한 누적 비용이다. 그림에서 ㉯와 ㉱ 사이가 비용 차이로 표시되어 있으므로, 완료한 작업의 예산가치인 EV와 비교되는 실제 비용에 해당한다.

㉰는 PV(Planned Value)이다. 기준일까지 **완료하기로 계획한 작업**에 배정된 예산가치이며, 예정 종료일의 BAC로 이어지는 계획 곡선 위에 있다.

㉱는 EV(Earned Value)이다. 기준일까지 **실제로 완료한 작업**을 예산 기준으로 평가한 가치이다. 지출액 자체인 AC와 다르며, 그림에서는 PV보다 낮으므로 계획 대비 작업 진도가 뒤처져 있다.

비용편차는 $CV = EV - AC$, 일정편차는 $SV = EV - PV$이다. 그림에서는 두 값 모두 음수이다. 일정 차이도 이 그래프에서는 세로축의 예산가치 차이로 표현하므로 지연 일수 자체를 뜻하지 않는다.

따라서 ㉮ BAC, ㉯ AC, ㉰ PV, ㉱ EV이며, 이 대응을 제시한 선택지는 ③이다.

NoteBase

문 37. 변경 대상 형상 항목(configuration item)에 대한 수정 작업이 완료되면 검토 및 승인 절차를 거친 후 형상 관리 서버(server)에 다시 등록하는 행위는?

① 체크아웃(check-out)

② 체크인(check-in)

③ 리포지토리(repository)

④ 배포(release)

정답 2번

간단한 해설

수정한 형상 항목을 검토·승인 후 형상 관리 저장소에 다시 반영하는 행위는 **체크인(check-in)**이다. 수정 작업을 위해 항목을 작업 영역으로 가져오는 체크아웃과는 반대 방향의 처리이다.

개념 구분

체크인(check-in): 작업 영역에서 수정한 형상 항목을 **형상 관리 저장소에 새 버전으로 반영**하는 행위이다. 발문의 '수정 작업 완료', '검토 및 승인', '서버에 다시 등록'은 수정 결과를 관리 대상 버전으로 되돌려 넣는 과정에 해당한다. 검토·승인은 문제에서 정한 관리 절차이며, 체크인이라는 용어의 모든 도구별 동작에 동일한 승인 절차가 내장되어 있다는 뜻은 아니다.

체크아웃(check-out): 수정하려는 관리 항목을 저장소에서 작업 영역으로 가져오거나 편집 가능한 상태로 만드는 행위이다. 이 문제의 체크아웃·체크인 방식에서 체크아웃은 수정 작업을 시작하기 위한 쪽이고, 체크인은 수정 결과를 다시 등록하는 쪽이다.

리포지토리(repository): 형상 항목과 그 버전·변경 이력을 보관하는 저장소이다. 변경 결과가 등록되는 대상 장소를 뜻하며, 등록 행위 자체의 이름은 아니다.

배포(release): 일정한 기능과 품질을 갖춘 버전을 묶어 사용자나 운영 환경에 제공하는 과정이다. 개별 형상 항목의 수정분을 저장소에 등록하는 단계와는 구별된다.

따라서 발문이 설명하는 행위는 ② 체크인(check-in)이다.

NoteBase

문 38. 진화하는 시스템의 유지보수 프로세스는 시스템이 소멸될 때까지 일정한 평균 작업량을 보인다는 Lehman의 소프트웨어 진화 법칙은?

① 지속적인 변경의 원칙

② 지속적 성장의 법칙

③ 안정성 유지의 법칙

④ 엔트로피, 복잡도 증가의 원칙

정답 3번

간단한 해설

안정성 유지의 법칙은 진화하는 시스템에 투입되는 조직의 평균적인 유효 작업률이 장기적으로 안정된다는 법칙이다. 기능의 증가나 복잡도의 증가가 아니라 **평균 작업량의 안정성**이 판단 단서이다.

개념 구분

'일정한 평균 작업량'은 시스템의 기능이나 코드가 변하지 않는다는 뜻이 아니라, 개발·유지보수 조직이 지속적으로 수행하는 **평균적인 유효 작업률이 안정되는 경향**을 말한다. 이를 안정성 유지의 법칙 또는 조직적 안정성 보존의 법칙이라 한다. 개별 시점의 작업량까지 항상 같다는 주장은 아니다.

지속적인 변경의 원칙은 현실 환경과 요구가 변하므로 사용 중인 시스템도 계속 적응·변경해야 한다는 뜻이다. 변경하지 않으면 시스템에 대한 만족도가 낮아진다.

지속적 성장의 법칙은 사용자 만족을 유지하기 위해 시스템의 **기능적 내용이 계속 확장**되어야 한다는 뜻이다. 작업률이 안정된다는 설명과는 판단 대상이 다르다.

엔트로피, 복잡도 증가의 원칙은 시스템이 진화할수록 복잡도가 증가하며, 이를 억제하거나 줄이는 별도의 노력이 필요하다는 뜻이다. 여기서 엔트로피는 소프트웨어 구조가 복잡해지고 무질서해지는 경향을 가리키는 표현이다.

따라서 발문의 평균 작업량에 관한 설명은 안정성 유지의 법칙에 해당하므로 ③과 연결된다.

NoteBase

문 39. 소프트웨어 개발 프로세스 수행 중 변경 요청에 대한 평가, 조정, 승인 여부를 결정하는 형상 관리 단계는?

① 형상 식별(identification)

② 형상 통제(control)

③ 상태 보고(status accounting)

④ 형상 감사(audits)

정답 2번

간단한 해설

변경 요청의 영향을 평가하고 **승인 또는 거부를 결정하여 변경을 관리**하는 활동은 형상 통제이다. 변경 상태를 기록하는 상태 보고나 산출물의 일치 여부를 확인하는 형상 감사와 구별된다.

핵심 풀이

형상 통제: 기준으로 확정한 형상에 변경 요청이 들어오면, 필요성과 비용·일정·품질 등의 영향을 검토하고 관련 작업을 조정하여 **변경을 승인할지 결정**한다. 승인된 변경을 정해진 절차에 따라 반영하도록 관리하는 활동도 포함한다. 문항의 '평가, 조정, 승인 여부'가 이 기능을 가리킨다.

형상 식별: 소스코드·문서 등 관리할 형상 항목을 정하고 각 항목과 버전·기준선을 식별한다. 무엇을 관리하는지 구분하는 활동이며, 개별 변경 요청을 승인할지 판단하는 활동과는 다르다.

상태 보고: 형상 항목의 현재 버전, 변경 요청의 처리 상태, 승인된 변경의 반영 여부 등을 기록하고 보고한다. 승인 결과를 기록할 수는 있지만, **승인 여부의 결정과 그 상태의 기록**은 서로 다른 기능이다.

형상 감사: 실제 산출물이 요구사항과 승인된 형상 문서에 부합하는지, 승인된 변경이 제대로 반영되었는지 등을 확인한다. 변경을 허가하는 통제와, 결과의 일치·완전성을 확인하는 감사를 구별한다.

따라서 변경 요청의 평가·조정·승인 여부 결정에 해당하는 단계는 ② 형상 통제이다.

NoteBase

문 40. CMMI(Capability Maturity Model Integration)의 성숙도 모델에서 최적화 단계에 해당하는 것은?

① 근본 원인 분석 및 해결

② 정량적 프로젝트 관리

③ 의사 결정 분석 및 해결

④ 프로세스/제품 품질 보증

정답 1번

간단한 해설

최적화 단계는 측정·관리된 프로세스를 지속적으로 개선하는 **성숙도 5단계**이다. 결함이나 문제의 **근본 원인을 분석하고 제거**해 재발을 방지하는 활동이 여기에 해당한다.

선지별 해설

① O: 근본 원인 분석 및 해결(CAR)은 이 문제의 CMMI 단계별 프로세스 영역 분류에서 **5단계 최적화(Optimizing)**에 해당한다. 발생한 결함이나 문제를 개별적으로 처리하는 데 그치지 않고 원인을 분석하고 개선 조치를 적용하여 재발을 줄이고 프로세스 성과를 높이는 활동이다.

② X: 정량적 프로젝트 관리(QPM)는 **4단계 정량적 관리(Quantitatively Managed)**에 해당한다. 품질과 프로세스 성과 목표를 세우고 측정 자료 및 통계적·정량적 기법으로 프로젝트의 수행을 관리한다. 정량적으로 안정된 수행을 관리하는 4단계와, 이를 바탕으로 지속적인 개선을 추진하는 5단계를 구별해야 한다.

③ X: 의사 결정 분석 및 해결(DAR)은 **3단계 정의(Defined)**에 해당한다. 미리 정한 평가 기준에 따라 여러 대안을 공식적인 평가 절차로 비교하고 적절한 대안을 선택하는 활동이다. 문제의 근본 원인을 제거해 프로세스를 개선하는 CAR와 달리, 의사결정 대안을 체계적으로 평가하는 데 초점이 있다.

④ X: 프로세스/제품 품질 보증(PPQA)은 **2단계 관리(Managed)**에 해당한다. 수행한 프로세스와 작업 산출물이 적용되는 절차·표준·요구사항을 준수하는지 객관적으로 평가하고, 부적합 사항을 식별하여 해결되도록 관리한다. 따라서 최적화 단계에 새로 대응되는 활동으로 고를 수 없다.

NoteBase