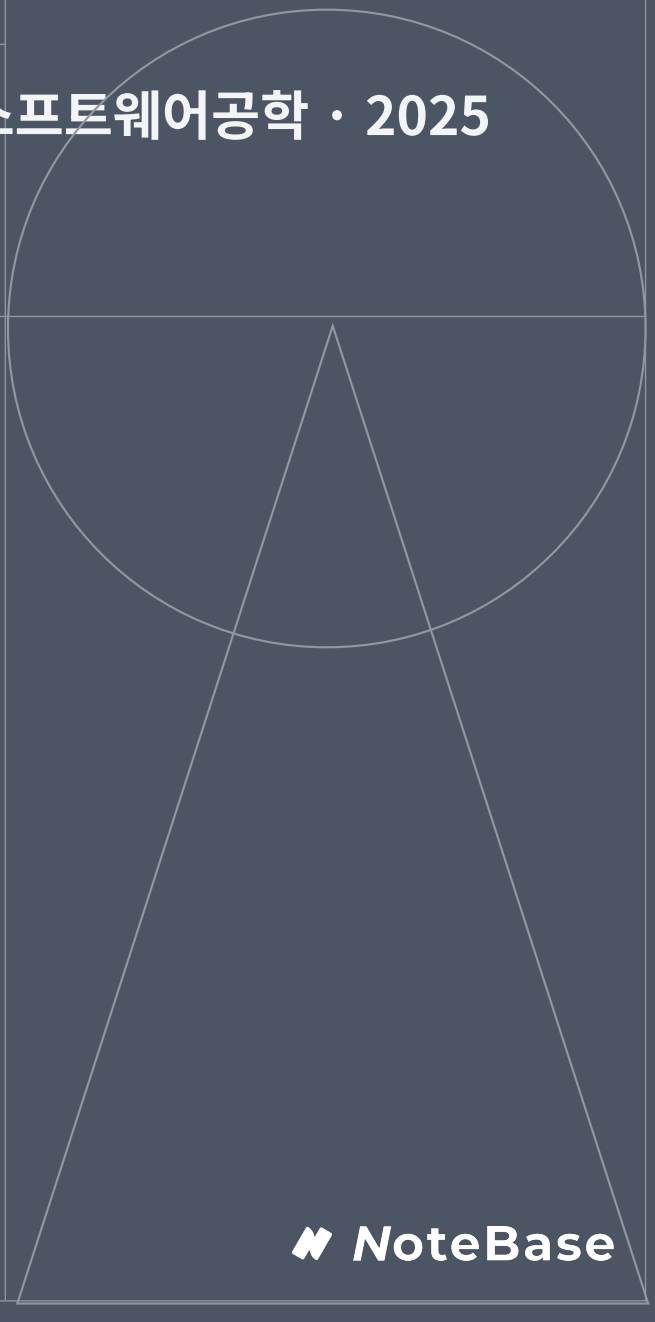
기출OX 해설집

공무원/경찰/공채(경위) · 소프트웨어공학 · 2025



NoteBase

기출OX 해설집

공무원/경찰/공채(경위) · 소프트웨어공학 · 2025

공무원/경찰/공채(경위) · 소프트웨어공학 · 2025

문 1. 다음 보기에서 소프트웨어공학이 등장하게 된 배경으로 올바른 것을 모두 고른 것은?

가. 소프트웨어 개발 인력 공급 과잉
나. 소프트웨어 개발 기간의 지연 및 개발 비용의 증가
다. 하드웨어 컴퓨터의 발전 속도 저하
라. 소프트웨어 유지보수의 어려움

① 가, 나
② 가, 다
③ 나, 다
④ 나, 라

정답 4번

간단한 해설

소프트웨어공학은 개발 규모와 복잡성이 커지면서 나타난 **일정 지연·비용 증가·유지보수의 어려움** 등 소프트웨어 위기에 대응하기 위해 등장하였다.

NoteBase

보기별 해설

가. X: 소프트웨어 수요와 규모가 커지는 데 비해 숙련된 개발 인력과 개발 생산성이 이를 따라가지 못한 것이 문제였다. 따라서 배경은 인력의 공급 과잉이 아니라 **수요에 비해 부족한 개발 역량**이다.

나. O: 체계적인 개발 방법과 관리가 부족하면 요구사항 변경과 복잡성 증가에 대응하기 어려워 일정이 지연되고 비용이 초과된다. 소프트웨어공학은 **개발 일정·비용·품질을 체계적으로 관리**하려는 필요에서 등장하였다.

다. X: 하드웨어의 성능과 활용 범위는 빠르게 발전했지만, 이를 활용할 소프트웨어의 개발 능력이 충분히 따라가지 못했다. 따라서 하드웨어 발전 속도의 저하가 아니라 **하드웨어 발전과 소프트웨어 개발 능력 사이의 격차**가 배경이다.

라. O: 구조가 복잡하고 문서화가 부족한 소프트웨어는 오류 수정이나 기능 변경 때 영향을 파악하기 어렵다. 개발 후의 **수정·확장·유지보수를 용이하게 하는 체계적 방법**이 필요해진 것도 소프트웨어공학의 등장 배경이다.

올바른 배경은 나, 라이므로 ④에 해당한다.

NoteBase

문 2. 다음과 같은 특징을 갖고 있는 방법론으로 가장 올바른 것은?

가. 기업 업무 중심, 자료 중심, 도형 중심의 접근
나. 프로젝트계획, 개발운영 단계의 명확한 구조 기반 제시
다. CASE 도구 등 정보시스템 개발의 자동화 지향
라. 고객 지향적, 최신 정보 기술의 능동적 수용

① 컴포넌트 기반 방법론
② 정보공학 방법론
③ 객체지향 방법론
④ 애자일 방법론

정답 2번

간단한 해설

기업의 업무와 자료를 도형으로 모델링하고, CASE 도구로 체계적인 정보시스템 개발을 지원하는 것은 정보공학 방법론의 특징이다. 고객 지향성만이 아니라 자료·도형 중심 접근과 개발 자동화를 함께 제시했다는 점이 결정적이다.

공통 해설

정보공학 방법론은 기업 전체의 업무와 자료를 통합적으로 모델링하고, 그 모형을 바탕으로 정보시스템의 계획·분석·설계·구축을 체계화하는 방법론이다. 개별 프로그램에만 초점을 두기보다 **기업 업무와 자료의 관계를 개발 전반에 연결**한다.

가의 '자료 중심'은 업무에 필요한 자료와 자료 사이의 관계를 명확히 한다는 뜻이고, '도형 중심'은 이를 도표나 다이어그램으로 표현한다는 뜻이다. 나의 단계별 구조는 이러한 모형과 산출물을 계획에서 개발·운영으로 일관되게 연결하는 틀이다. 다의 CASE는 소프트웨어 개발을 컴퓨터 도구로 지원하는 것으로, 모형 작성·관리 등 개발 작업의 자동화를 뒷받침한다. 라의 고객 지향성과 최신 정보 기술의 수용도 기업의 업무 요구를 충족하는 정보시스템을 구축한다는 목적과 연결된다.

고객 지향성은 애자일 방법론에서도 나타날 수 있지만, 그것만으로 제시된 특징 전체를 설명할 수는 없다. 이 문항에서는 **기업 차원의 자료·도형 중심 모델링과 체계적인 개발 지원**을 함께 제시하고 있으므로 정보공학 방법론에 해당한다.

NoteBase

문 3. 다음 보기에서 소프트웨어개발 프로세스와 방법론의 차이점으로 올바르지 않은 것을 모두 고른 것은?

가. 프로세스는 소프트웨어 각 작업을 어떻게 수행하느냐를 정의한 것이다.
나. 방법론은 소프트웨어개발 조직이 자유로이 선택할 수 있으며 프로세스의 구현이라고 생각할 수 있다.
다. 방법론은 개발 패러다임에 의해 좌우되지 않는다.
라. 소프트웨어 프로세스는 방법론에 따라 결정된다.

① 가, 나
② 가, 다
③ 가, 다, 라
④ 나, 다, 라

정답 3번

간단한 해설

이 문항은 프로세스를 **무엇을 어떤 흐름으로 수행할지 정하는 틀**, 방법론을 **그 작업을 어떻게 수행할지 구체화한 방법**으로 구분한다. 방법론은 개발 패러다임의 영향을 받으며 프로세스를 구현한다.

보기별 해설

가. X: 소프트웨어 프로세스는 요구분석·설계·구현·시험 등 어떤 활동을 수행하며 활동들이 어떤 흐름과 관계를 갖는지 정한다. 각 작업을 어떤 기법으로 구체적으로 수행하는지를 중심으로 설명하는 것은 방법론에 해당하므로, 선지는 두 개념의 초점을 바꾸었다.

나. O: 방법론은 프로세스를 실제로 수행하기 위한 절차·기법·표현 방법 등을 구체화한다는 점에서 프로세스의 구현으로 볼 수 있다. 개발 조직은 프로젝트의 특성과 기술·인력 여건에 맞는 방법론을 선택할 수 있다. 여기서 선택 가능하다는 것은 외부 계약이나 규제 등 현실적 제약이 전혀 없다는 뜻은 아니다.

다. X: 방법론은 **개발 패러다임의 영향을 받는다**. 예를 들어 구조적 방법론은 기능과 데이터의 분해를, 객체지향 방법론은 객체와 그 상호작용을 중심으로 분석·설계하므로 모델과 기법이 달라진다.

라. X: 이 문항의 개념 구분에서는 프로세스가 개발 활동의 상위 틀이고 방법론이 그 수행 방식을 구체화한다. 따라서 **방법론이 프로세스를 구현한다**는 관계를 '프로세스가 방법론에 따라 결정된다'고 뒤집은 설명은 적절하지 않다. 실제 조직에서 구체적인 프로세스를 조정할 수 있다는 사실과 이 개념적 구분은 별개이다.

따라서 올바르지 않은 보기는 **가, 다, 라**이다.

NoteBase

문 4. 다음 중 소프트웨어개발 계획 단계에서 도출해야 할 중요한 2가지 요소로 가장 올바른 것은?

① 비용과 일정

② 시간과 일정

③ 자원과 도구

④ 비용과 자원

정답 1번

간단한 해설

개발 계획은 필요한 작업과 자원을 추정하여 **비용과 일정**으로 구체화한다. 예산이 얼마나 필요한지와 언제 완료할지를 함께 정해야 실행 가능성을 판단할 수 있다.

핵심 풀이

소프트웨어개발 계획은 정해진 범위와 품질을 달성하기 위해 필요한 작업과 자원을 추정하고, 실행 가능한 예산과 시간계획으로 구체화하는 과정이다. 제시된 선택지에서는 **얼마의 비용이 필요한지와 언제 완료할지**를 함께 나타내는 조합을 찾아야 한다.

비용: 개발 규모와 작업량을 바탕으로 인건비, 장비·도구 사용료 등 필요한 지출을 추정한다. 이는 예산을 확보하고 사업의 실행 가능성을 판단하는 기준이 된다.

일정: 작업을 나누고 선후관계와 자원 가용성을 고려하여 시작·종료 시점, 기간, 주요 점검 시점을 정한다. 단순한 총 소요시간보다 어떤 작업을 언제 수행할지를 구체화한 계획이다.

다른 조합과의 구별: '시간과 일정'은 시간 측면에 치우쳐 비용을 포함하지 않는다. '자원과 도구'는 개발에 투입할 수단이며 비용·일정을 산정할 때 고려할 항목이다. '비용과 자원'에는 언제 완료할지에 관한 일정이 빠져 있다. 자원이나 도구도 계획의 중요 대상이지만, 문항이 묻는 대표적인 두 관리 기준을 함께 제시한 것은 아니다.

따라서 비용과 일정을 묶은 ①이 가장 적절하다.

NoteBase

문 5. 소프트웨어 개발 방법 중 UP(Unified Process) 개발 단계에 관련된 것으로 가장 올바르지 않은 것은?

① 도입 단계(Inception phase)는 고객과의 커뮤니케이션을 통해 소프트웨어에 대한 비즈니스적 요구사항을 확인하고 개략적인 시스템 구조를 제안, 기초를 수립한다.

② 정련 단계(Elaboration phase)는 도입 단계에서 도출된 예비 유스케이스를 구체화하며 구조 표현을 모델에 따라 확장한다.

③ 구축 단계(Construction phase)는 유스케이스에 따르는 소프트웨어 컴포넌트를 개발하고 각 컴포넌트의 단위 테스트를 설계 및 수행한다.

④ 전이 단계(Transition phase)는 사용자에게 전달하기 전 내부 테스트의 마지막 단계로써 통합 소프트웨어의 기능 검증 및 다음 단계에서의 사용자 피드백을 실시하기 위한 준비를 수행한다.

정답 4번

간단한 해설

전이 단계는 내부 테스트만 마무리하는 단계가 아니라 **제품을 사용자 환경에 전달·배포**하는 단계이다. 사용자 교육, 베타 테스트와 피드백 반영도 이 단계에 포함된다.

선지별 해설

① O: 도입 단계는 이해관계자와의 의사소통을 통해 프로젝트의 목적·범위와 주요 요구사항을 파악하고 사업적 타당성을 검토한다. 주요 유스케이스와 개략적인 시스템 구조를 제시하여 **개발을 추진할 기초**를 세운다. 이 단계에서 모든 세부 요구나 구조가 확정되는 것은 아니다.

② O: 정련 단계는 주요 요구사항과 유스케이스를 구체화하고, 분석·설계 모델을 발전시켜 **핵심 아키텍처와 주요 위험**을 검증한다. 이후 본격적인 구현을 진행할 수 있도록 구조의 기반을 안정화한다.

③ O: 구축 단계에서는 유스케이스를 실현하는 컴포넌트와 기능을 개발하고 통합한다. 각 컴포넌트에 대한 단위 테스트를 설계·수행하는 것도 포함되며, 완성도를 높여 사용자에게 인도할 수 있는 제품으로 발전시킨다. 테스트가 이 단계에서만 이루어진다는 뜻은 아니다.

④ X: 전이 단계의 목적은 **제품을 사용자 환경에 인도하고 사용할 수 있게 하는 것**이다. 배포·설치, 사용자 교육, 베타 테스트, 사용자 피드백 수집과 문제 수정 등이 포함된다. 따라서 사용자 전달 이전의 내부 테스트로만 규정하거나 사용자 피드백을 다음 단계의 일로 미루는 설명은 부적절하다.

NoteBase

문 6. 애자일 소프트웨어 개발 방법에 관련된 설명으로 가장 올바르지 않은 것은?

① XP(eXtreme Programming)에서 고객은 개발팀의 일부이고 다른 팀원들과 시나리오에 대해 토론한다.

② 스크럼(Scrum) 프로세스는 계략적인 계획과 구조설계, 스프린트(Sprint) 사이클, 프로젝트 마감 단계를 가진다.

③ 적응적 소프트웨어 개발(ASD, Adaptive Software Development)은 추측(Speculation), 분할(Decomposition), 학습(Learning) 세 가지 단계의 생명주기를 정의하였다.

④ 소프트웨어의 요구사항은 프로젝트 수행 중에 변화할 수 있으며, 고객의 요구사항에 적극 대응하여 반영 개발한다.

정답 3번

간단한 해설

ASD의 세 단계는 추측·**협력**·학습이다. '분할(Decomposition)'은 이 생명주기의 단계가 아니다.

선지별 해설

① O: XP는 고객이 개발팀과 긴밀히 협력하면서 사용자 스토리 등으로 필요한 기능을 설명하고 우선순위와 수용 기준을 논의하도록 한다. 고객의 지속적인 참여로 **요구사항과 실제 구현 사이의 차이**를 줄인다.

② O: 초기 스크럼 프로세스의 큰 흐름은 계획·구조설계, 반복적인 스프린트, 마감으로 설명된다. 먼저 목표와 구조를 잡고 **스프린트에서 기능을 점진적으로 개발**한 뒤 제품을 정리·인도하는 흐름이다. 이는 초기 프로세스의 구분이며, 현재 스크럼의 공식 이벤트 명칭을 나열한 것은 아니다.

③ X: ASD의 생명주기는 추측(Speculation), **협력(Collaboration)**, 학습(Learning)이다. 추측은 불확실성을 인정하고 방향을 설정하는 단계, 협력은 구성원이 함께 결과물을 개발하는 단계, 학습은 결과와 피드백을 통해 다음 활동을 개선하는 단계이다. '분할(Decomposition)'을 '협력'으로 바꾸어야 한다.

④ O: 애자일은 처음 정한 요구사항을 무조건 고정하기보다 고객과 소통하고 결과물을 확인하면서 변화에 대응한다. **요구사항의 변화를 피드백으로 반영**하여 고객에게 가치 있는 소프트웨어를 점진적으로 제공한다.

따라서 ASD의 협력 단계를 분할로 제시한 ③이 가장 올바르지 않다.

NoteBase

문 7. 다음 중 기능 점수(Function Point) 산출을 위한 항목으로 올바른 서술을 모두 고른 것은?

가. 외부 입력(External Input) : 내부 파일의 내용에 영향을 주는 사용자 데이터 또는 제어입력을 트랜잭션 단위로 카운트한다.
나. 외부 조회(External Inquiry) : 계산 통계 그래프 최종 보고서 등 소프트웨어 외부 출력되는 데이터 제공을 카운트한다.
다. 내부 논리 파일(Internal Logical File) : 소프트웨어에 의해 생성, 사용, 관리되는 파일을 포함, 시스템에 존재하는 사용자 데이터 및 제어 정보 그룹을 카운트한다.
라. 외부 인터페이스 파일(External Interface File) : 소프트웨어 시스템 사이에 전달되거나 공유된 파일을 카운트한다.

① 가, 다

② 가, 다, 라

③ 나, 다, 라

④ 가, 나, 다, 라

정답 2번

간단한 해설

계산·통계 처리 등으로 가공한 결과를 제공하는 기능은 **외부 조회(EQ)가 아니라 외부 출력(EO)**이다. 외부 조회는 계산이나 데이터 갱신 없이 저장된 정보를 조회하여 제공한다.

NoteBase

보기별 해설

기능 점수는 사용자에게 제공되는 기능의 논리적 규모를 측정한다. 입력·출력·조회는 트랜잭션 기능이고, 내부 논리 파일·외부 인터페이스 파일은 데이터 기능이다.

가. O: 외부 입력(EI)은 응용프로그램 경계 밖에서 들어온 데이터나 제어정보를 처리하여 **내부 논리 파일을 유지하거나 시스템 동작을 변경**하는 단위 기능이다. 개별 데이터 항목 수가 아니라 사용자가 인식하는 완결된 트랜잭션을 식별하여 센다.

나. X: 계산·통계 처리로 가공한 결과를 제공하는 것은 **외부 출력(EO)**에 해당한다. 외부 조회(EQ)는 저장된 정보를 조회하여 제공하는 기능으로, 계산·파생 데이터 생성·내부 논리 파일 갱신을 수반하지 않는다. 보고서나 그래프라는 표시 형식만이 아니라 추가 처리의 유무로 구분한다.

다. O: 내부 논리 파일(ILF)은 응용프로그램 내부에서 유지·관리하는, 사용자가 식별할 수 있는 관련 데이터 또는 제어정보의 논리적 집합이다. 따라서 **물리적 파일 개수가 아니라 논리적 데이터 그룹**을 센다.

라. O: 외부 인터페이스 파일(EIF)은 해당 응용프로그램이 참조하지만 **다른 응용프로그램에서 유지·관리**하는 논리적 데이터 그룹이다. 보기의 시스템 간 공유 파일은 이러한 외부 데이터 기능을 가리킨다. 단순히 파일을 전달했다는 사실만으로 모든 전송 파일을 EIF로 세는 것은 아니다.

따라서 올바른 서술은 가, 다, 라이며, 이에 해당하는 선택지는 ②이다.

NoteBase

문 8. 다음 보기 중 COCOMO 모델의 프로젝트 유형을 모두 고른 것은?

가. 조직형(Organic)
나. 반분리형(Semi-detached)
다. 혼합형(Mixed)
라. 내장형(Embedded)

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

정답 3번

간단한 해설

원래 COCOMO의 프로젝트 유형은 **조직형·반분리형·내장형**이다. 반분리형에 여러 경험 수준이 섞인다는 특성이 있더라도 '혼합형'이라는 별도의 네 번째 유형이 생기는 것은 아니다.

보기별 해설

COCOMO는 소프트웨어 개발 노력과 비용을 추정하는 모형이다. 이 문항의 프로젝트 유형은 원래 COCOMO의 개발 모드 구분을 뜻한다.

가. O: 조직형(Organic)은 비교적 작은 팀이 익숙한 업무 분야에서, 상대적으로 유연한 요구와 제약 아래 개발하는 유형이다. 개발자의 경험과 작업환경의 친숙도가 높고 복잡성이 비교적 낮은 프로젝트를 대표한다.

나. O: 반분리형(Semi-detached)은 조직형과 내장형의 중간적 특성을 갖는 유형이다. 개발팀의 경험 수준이 다양하고 프로젝트 규모·복잡성이나 제약도 중간적인 경우를 나타낸다.

다. X: **혼합형(Mixed)은 COCOMO의 독립적인 프로젝트 유형이 아니다**. 반분리형을 혼합된 경험과 중간적 특성을 갖는 프로젝트라고 설명할 수는 있지만, 그것을 별도 유형으로 추가해서는 안 된다.

라. O: 내장형(Embedded)은 하드웨어·소프트웨어·운영환경 등의 엄격한 제약이 서로 긴밀하게 연결된 유형이다. 시간·성능·자원 등 까다로운 조건을 만족해야 하는 개발이 대표적이다.

따라서 해당하는 보기는 **가, 나, 라**이다.

NoteBase

문 9. 다음은 어떤 소프트웨어 프로젝트를 구성하는 선행 작업과 소요기간을 계획한 표이다. 하지만 실제 작업에서 T4의 소요기간이 3일 추가되어 5일이 되었다. 이로 인하여 이 프로젝트가 지연된 총 일수는?

작업	T1	T2	T3	T4	T5	T6	T7	T8	T9	종료
계획소요기간(일)	1	4	3	2	3	4	3	4		
선행작업	-	-	-	T1 T2	T2 T3	T3 T4	T4 T5	T1 T5	T6 T7 T8	T9

① 지연없음

② 1일

③ 2일

④ 3일

정답 3번

간단한 해설

T4가 늦어지기 전에는 T9의 착수 시점이 11일이지만, 변경 후에는 T6가 13일에 끝나 T9도 그때 착수한다.
따라서 프로젝트 지연은 추가 작업기간 3일이 아니라 **2일**이다.

NoteBase

계산 과정

프로젝트 시작을 0일 시점으로 두고, 선행작업이 모두 끝나면 다음 작업에 착수한다고 본다. 선행작업이 여러 개이면 **그중 가장 늦은 완료 시점**이 다음 작업의 착수 시점이다. 표에서 T9는 T6·T7·T8이 모두 끝나야 시작하며, 프로젝트는 T9 뒤에 종료된다.

계획상 T1·T2·T3의 완료 시점은 각각 1일·4일·3일이다. T4는 T1과 T2 완료 뒤 시작하므로 4 + 2 = 6일에, T5는 T2와 T3 완료 뒤 시작하므로 4 + 3 = 7일에 끝난다.

계획상 T6는 T3과 T4 중 늦은 6일에 시작하여 6 + 4 = 10일에 끝난다. T7은 T4와 T5 중 늦은 7일에 시작하여 7 + 3 = 10일에 끝난다. T8은 T1과 T5 중 늦은 7일에 시작하여 7 + 4 = 11일에 끝난다. 따라서 T9는 11일 시점에 착수한다.

T4의 기간이 5일이 되면 그 완료 시점은 4 + 5 = 9일로 바뀐다. T5의 완료 시점은 여전히 7일이다. T6는 9 + 4 = 13일에, T7은 9 + 3 = 12일에 끝난다. T8은 T4에 의존하지 않으므로 그대로 11일에 끝난다. 따라서 T9의 착수 시점은 13일이 된다.

표에는 T9 자체의 소요기간이 적혀 있지 않지만, 이 문제에서는 T4만 변경되므로 **T9의 소요기간은 변경 전후 공통**이다. 따라서 프로젝트 종료 시점의 차이는 T9 착수 시점의 차이인 13 − 11 = 2일이다. T4의 추가 기간 3일을 그대로 프로젝트 지연으로 더하면 안 된다.

프로젝트가 지연된 총 일수는 2일이므로 ③과 일치한다.

NoteBase

문 10. Java 프로그래밍 언어에서 특정 클래스 X의 속성에는 접근 수식자를 명기하지 않거나 private, public, protected로 선언할 수 있다. 다음 중 이에 대한 서술로 가장 올바르지 않은 것은?

① 접근 범위는 private, protected, 수식자를 명기하지 않음, public 순으로 확대된다.

② protected로 선언하였을 경우 X로부터 상속받은 다른 패키지의 클래스에서 접근 가능하다.

③ private로 선언하였을 경우 동일 패키지의 동일 클래스에서 접근 가능하다.

④ public으로 선언하였을 경우 다른 패키지의 클래스에서 접근 가능하다.

정답 1번

간단한 해설

접근 수식자를 생략하면 같은 패키지에서만 접근할 수 있지만, protected는 다른 패키지의 하위 클래스에도 조건부 접근을 허용한다. 따라서 **수식자 생략보다 protected의 접근 범위가 넓다**.

선지별 해설

① X: 일반적인 멤버 접근 범위는 private → 접근 수식자 생략(package-private) → protected → public 순으로 넓어진다. 수식자를 생략하면 같은 패키지로 제한되지만, protected는 여기에 **다른 패키지의 하위 클래스에서의 접근**을 추가로 허용한다. 두 접근 수준의 순서가 바뀌었다.

② O: protected 멤버는 같은 패키지뿐 아니라 다른 패키지에 있는 하위 클래스에서도 상속 관계를 통해 접근할 수 있다. 다만 다른 패키지의 하위 클래스가 임의의 X 객체를 통해 인스턴스 멤버에 접근할 수 있다는 뜻은 아니며, **하위 클래스 측의 접근 조건**을 따라야 한다.

③ O: private 멤버는 선언한 클래스 내부에서 접근할 수 있으므로 X 자신의 메서드 등에서는 이용할 수 있다. **같은 패키지라는 사실만으로 다른 독립 클래스에 접근이 허용되지는 않는다**.

④ O: public 멤버는 해당 클래스를 접근할 수 있는 곳에서 패키지가 달라도 접근 가능하다. 이때 **선언 클래스 자체의 접근 가능성**도 전제되며, public 멤버라는 이유만으로 접근 불가능한 클래스까지 공개되는 것은 아니다.

따라서 protected와 수식자 생략의 순서를 뒤바꾼 ①이 가장 올바르지 않다.

NoteBase

문 11. 다음 보기에서 소프트웨어 비용 산출에 있어 하향식 산정 방법으로 가장 올바른 것을 모두 고른 것은?

가. 원시코드 라인(LOC) 수 산정기법
나. 델파이 산정기법
다. COCOMO 산정기법
라. 기능점수 산정기법
마. 전문가 판단 산정기법

① 가, 나
② 나, 마
③ 나, 라, 마
④ 다, 라, 마

정답 2번

간단한 해설

하향식 비용 산정은 **전문가의 경험과 판단으로 전체 비용을 추정**하는 방식이다. 델파이 기법은 여러 전문가의 추정과 반복적인 의견 수렴을 활용한다.

NoteBase

보기별 해설

가. 규모 기반 산정: LOC 기법은 개발할 프로그램의 원시코드 줄 수를 추정하고 생산성 자료 등을 이용해 노력과 비용으로 환산한다. 세부 기능의 규모를 추정해 합산하는 방식으로 사용되며, 전문가의 전체 판단에 의한 하향식 기법으로 분류하지 않는다.

나. 하향식 산정: 델파이 기법은 여러 전문가가 개별적으로 추정한 값을 모으고, 의견을 환류하며 반복 추정하여 합의에 접근한다. 특정 전문가의 주관이나 집단 토론의 편향을 줄이려는 방법이다.

다. 모형 기반 산정: COCOMO는 소프트웨어 규모와 비용 영향 요인 등을 수학적 모형에 적용하여 개발 노력을 추정하는 경험적·알고리즘적 기법이다. 이 문항에서 묻는 전문가 판단 중심의 하향식 기법과 구별된다.

라. 기능 규모 기반 산정: 기능점수 기법은 사용자에게 제공하는 입력·출력·조회·자료 기능 등을 계량하여 소프트웨어의 기능적 크기를 산정한다. 그 크기에 생산성·단가 자료 등을 적용해 비용을 추정하므로, 전문가의 전체 비용 판단을 뜻하는 하향식 기법과 구별된다.

마. 하향식 산정: 전문가 판단 기법은 유사 프로젝트 경험과 전문 지식을 바탕으로 전체 개발 노력·기간·비용을 추정한다. 세부 작업의 비용을 먼저 모두 계산하여 더하는 방식과 대비된다.

따라서 하향식 산정 방법에 해당하는 것은 나와 마이며, 이 조합은 ②와 일치한다.

NoteBase

문 12. 다음 CMMi 소프트웨어 프로세스 개선 방식 중에서 단계적(Staged) 모델의 특징으로 가장 올바른 핵심 내용들을 모두 고른 것은?

가. 소프트웨어 CMM모델에 적합하며 1부터 5까지의 성숙도 수준을 할당한다.
나. 프로세스의 개선은 모델의 낮은 수준에서 높은 수준으로 이동하여 달성한다.
다. 조직간 비교를 가능하게 하는 단일 등급 체계를 제공한다.
라. 자신의 필요와 요구사항에 따라 조직을 개선하기 위한 프로세스를 선택할 수 있다.
마. 소프트웨어 SPICE 모델에 적합하며 24개의 프로세스 영역으로 분류한다.

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

정답 1번

간단한 해설

단계적 모델은 정해진 개선 경로에 따라 **조직의 성숙도를 1~5단계로 평가**하고 하나의 등급으로 나타낸다.
조직이 필요한 프로세스 영역을 골라 개선하는 설명은 **연속적 모델**과 연결된다.

NoteBase

보기별 해설

이 문항은 CMMI의 전통적인 단계적 표현과 연속적 표현을 비교한다. 단계적 표현은 조직의 전반적 성숙도에, 연속적 표현은 선택한 프로세스 영역의 능력에 초점을 둔다.

가. O: 단계적 모델은 소프트웨어 CMM의 단계별 성숙도 접근을 계승한다. 성숙도 수준은 초기, 관리, 정의, 정량적 관리, 최적화의 1~5단계로 구분한다.

나. O: 단계적 모델은 낮은 성숙도에서 확보한 관리 기반 위에 상위 단계의 개선을 쌓아 가도록 한다. 따라서 정해진 단계별 요구를 충족하면서 더 높은 성숙도로 이동하는 방식으로 프로세스 개선을 추진한다.

다. O: 여러 프로세스 영역의 달성 상태를 조직의 **하나의 성숙도 등급**으로 나타내므로 조직 간 비교에 활용할 수 있다. 다만 등급이 모든 제품의 품질이나 사업성과를 단독으로 보장한다는 뜻은 아니다.

라. X: 조직의 필요와 우선순위에 따라 개선할 프로세스 영역을 선택하는 것은 **연속적 모델의 특징**이다. 단계적 모델은 성숙도 단계별로 정해진 프로세스 영역의 집합을 중심으로 개선한다.

마. X: SPICE와 연결되는 프로세스별 능력 평가 관점은 연속적 모델에 가깝다. 따라서 이를 단계적 모델의 핵심 특징으로 고를 수 없다. 또한 프로세스 영역의 개수는 모형의 버전과 적용 범위에 따라 달라질 수 있으므로, ‘24개’라는 수치를 CMMI 전체에 공통되는 고정 정의로 보아서는 안 된다.

따라서 단계적 모델의 핵심 특징은 **가, 나, 다**이다.

NoteBase

문 13. 고장 간 평균 시간(MTBF, Mean Time Between Failures)이 250시간이고 평균 수리 시간(MTTR, Mean Time To Repair)이 40시간인 소프트웨어를 개선하여 평균 수리 시간은 유지하였으나, 평균 가동 시간(MTTF, Mean Time To Failure)은 380시간으로 개선하였다. 개선 전과 개선 후의 각각의 가용성(Availability)은? (단, 소수점 이하에서 반올림한다)

① 84%, 90%

② 86%, 90%

③ 84%, 89%

④ 86%, 89%

정답 1번

간단한 해설

개선 전의 250시간은 수리 시간을 포함한 MTBF이므로 정상 가동 시간은 210시간이다. 개선 후 380시간은 **정상 가동 시간인 MTTF**이므로 수리 시간 40시간을 더한 420시간을 기준으로 계산하며, 가용성은 각각 **84%, 90%**이다.

계산 과정

가용성은 전체 운영 주기 중 정상 가동할 수 있는 시간의 비율이다. 이 문제는 MTBF를 **정상 가동 시간과 수리 시간을 합친 고장 간 주기**로 구분하므로 $MTBF = MTTF + MTTR$를 적용한다. 따라서 가용성은 $A = \frac{MTTF}{MTTF + MTTR} \times 100\%$이다.

개선 전에는 MTBF가 250시간, MTTR이 40시간이므로 정상 가동 시간은 250 − 40 = 210시간이다. 전체 주기는 이미 250시간에 포함되어 있으므로 수리 시간을 분모에 다시 더하지 않는다. 개선 전 가용성은 A_전=(210)/(250)×100\%=84\%이다.

개선 후에는 **MTBF가 아니라 MTTF가 380시간**으로 주어졌다. 수리 시간 40시간은 그대로이므로 전체 주기는 380 + 40 = 420시간이다. 개선 후 가용성은 A_후=(380)/(420)×100\%≈90.476\%이며, 소수점 이하에서 반올림하면 90%이다.

따라서 개선 전 84%, 개선 후 90%로 ①과 일치한다.

NoteBase

문 14. 소프트웨어 요구추출 방법과 그 특징에 대한 설명으로 가장 올바르지 않은 것은?

① 관찰 - 사용자의 업무를 관찰하여 메모

② 인터뷰 - 당사자를 만나 준비된 질문, 대답 및 경청

③ 브레인스토밍 - 이해당사자들이 다양한 아이디어 토론

④ 유스케이스 분석 - 시스템의 내부 기능 파악

정답 4번

간단한 해설

유스케이스 분석은 시스템 내부 구현이 아니라 **액터와 시스템의 상호작용**을 통해 사용자에게 제공할 기능을 파악하는 방법이다.

선지별 해설

① O: 관찰은 사용자가 실제 업무를 수행하는 과정과 작업 환경을 살펴보고 기록하는 방법이다. 사용자가 당연하게 여겨 말하지 않는 **암묵적인 업무 절차와 요구**도 발견할 수 있다.

② O: 인터뷰는 이해당사자에게 질문하고 답변을 경청하면서 목표·업무·불편 사항을 확인하는 방법이다. 준비한 질문에 더해 답변에 따른 추가 질문으로 **모호한 요구를 구체화**할 수 있다.

③ O: 브레인스토밍은 여러 이해당사자가 다양한 아이디어를 제시하고 공유하여 요구 후보를 폭넓게 찾는 방법이다. 아이디어를 내는 동안 성급한 비판을 유보하여 **다양한 대안의 발산**을 촉진한다.

④ X: 유스케이스 분석은 액터가 목표를 달성하기 위해 시스템과 어떻게 상호작용하는지를 기술하여 **외부 관점의 기능 요구사항**을 파악한다. 내부 알고리즘이나 구현 구조를 분석하는 방법으로 설명해서는 안 된다.

따라서 유스케이스 분석을 내부 기능 파악으로 설명한 ④가 가장 올바르지 않다.

NoteBase

문 15. ISO/IEC 25010 소프트웨어 품질 특성 설명으로 가장 올바르지 않은 것은?

① ISO/IEC 25010으로 개정하는데 ISO/IEC 14598 표준안이 포함되었다.

② 기존 ISO/IEC 9126에서 추가된 주특성은 보안성, 호환성이다.

③ 주특성에 따른 부특성도 27개에서 31개로 증가하였다.

④ 기존 ISO/IEC 9126의 주특성이 6개에서 8개로 증가하였다.

정답 1번

간단한 해설

ISO/IEC 25010은 **제품 품질모델**을 다루며 ISO/IEC 9126-1을 개정·대체했다. ISO/IEC 14598의 평가 체계까지 통합·재편한 범위는 25010 하나가 아니라 **SQuaRE 전체 계열**이다.

선지별 해설

이 문항은 ISO/IEC 9126-1:2001에서 ISO/IEC 25010:2011로 개정된 제품 품질모델을 비교한다. 뒤에 개정된 판의 특성 수와 구분해야 한다.

① X: ISO/IEC 25010:2011은 **품질모델인 ISO/IEC 9126-1을 개정·대체**한 표준이다. ISO/IEC 14598의 제품 품질평가 체계까지 포괄하여 재구성한 것은 상위의 SQuaRE(ISO/IEC 25000 계열)이며, 품질평가는 ISO/IEC 2504n 계열에서 다룬다. SQuaRE 전체의 통합을 25010 하나의 개정 내용으로 설명한 것이 오류이다.

② O: 보안성과 호환성이 독립된 주특성이 되었다. 보안성은 종전 기능성의 하위 항목에서 분리되었고, 호환성에는 공존성과 상호운용성이 포함된다. 전혀 없던 개념이 생겼다는 뜻이 아니라 **주특성 수준에서 신설·재편**되었다는 의미이다.

③ O: 9126-1의 주특성별 부특성 수는 기능성 5개, 신뢰성 4개, 사용성 5개, 효율성 3개, 유지보수성 5개, 이식성 5개로 합계 27개이다. 25010:2011에서는 기능적합성 3개, 성능효율성 3개, 호환성 2개, 사용성 6개, 신뢰성 4개, 보안성 5개, 유지보수성 5개, 이식성 3개로 합계 31개이다. 기존 항목의 삭제·이동과 새 항목의 추가가 함께 이루어진 결과이다.

④ O: 9126-1의 기능성·신뢰성·사용성·효율성·유지보수성·이식성이라는 6개 주특성이, 25010:2011에서는 기능적합성·성능효율성·호환성·사용성·신뢰성·보안성·유지보수성·이식성의 **8개 주특성**으로 재편되었다.

따라서 품질모델 표준과 SQuaRE 전체의 통합 범위를 혼동한 ①이 가장 올바르지 않다.

NoteBase

문 16. 다음 보기에서 컴포넌트 기반 개발방법론에 대한 설명으로 올바른 것을 모두 고른 것은?

가. 기존 방법론에 비해 소프트웨어 개발 기간을 단축시킨다.
나. 컴포넌트 통합을 위해 표준 인터페이스가 제공되므로 미들웨어를 지원하지 않는다.
다. 컴포넌트는 개발자에게 다른 컴포넌트 및 모듈들과 상호 작용하기 위해 인터페이스를 제공한다.
라. COTS(Component Off-The-Shelf)는 외부에서 구입하는 상용컴포넌트이다.

① 가, 라
② 가, 나, 다
③ 가, 다, 라
④ 나, 다, 라

정답 3번

간단한 해설

표준 인터페이스는 컴포넌트가 서로 연결되는 규약이며, 이를 제공한다고 해서 **미들웨어를 지원하지 않는 것은 아니다**. 미들웨어는 컴포넌트 사이의 통신과 연동을 지원할 수 있다.

보기별 해설

가. O: 컴포넌트 기반 개발은 이미 만들어진 기능 단위를 재사용하고 조립하여 소프트웨어를 구축하는 방식이다. 모든 기능을 처음부터 구현하는 작업이 줄어들므로 **개발 기간을 단축**할 수 있다.

나. X: 표준 인터페이스는 다른 컴포넌트의 기능을 어떤 규약으로 호출할지를 정한다. 미들웨어는 컴포넌트 사이의 통신과 연동 등을 지원하므로, 표준 인터페이스 제공과 미들웨어 사용은 양립할 수 있다. '미들웨어를 지원하지 않는다'는 설명은 **미들웨어를 이용하여 컴포넌트 통합을 지원할 수 있다**로 고쳐야 한다.

다. O: 컴포넌트는 외부에서 사용할 수 있는 기능과 호출 규약을 인터페이스로 제공한다. 다른 컴포넌트나 모듈은 **내부 구현을 직접 다루지 않고 인터페이스를 통해** 기능을 이용하고 상호 작용한다.

라. O: 외부에서 구입하는 상용 컴포넌트라는 설명을 기준으로 한 판정이다. COTS는 이미 만들어져 상업적으로 구입할 수 있는 제품을 뜻하며, 컴포넌트 기반 개발에서는 이러한 제품을 구입하여 재사용할 수 있다. 다만 원문의 'Component Off-The-Shelf'는 정확한 풀네임이 아니며, COTS의 정확한 풀네임은 **Commercial Off-The-Shelf**이다.

라의 상용 컴포넌트라는 설명을 기준으로 올바른 보기는 가·다·라이며, 이는 ③의 조합에 해당한다.

NoteBase

문 17. 다음 중 기능적 요구사항과 비기능적 요구사항을 가장 올바르게 분류한 것은?

가. 사용자 인터페이스의 버튼은 하단 중앙에 위치한다.
나. 검색 버튼을 누르면 결과는 3초 이내 표시된다.
다. 보안성에 대한 요구 사양을 만족시켜야 한다.
라. 사용자 입력값에 대한 범위와 단위를 설정한다.
마. 사용자의 권한에 따른 분류를 설정하여 구분한다.
바. 가용성(Availability)은 최소 90% 이상을 보장한다.

① 기능적 요구사항: 가, 나, 라, 비기능적 요구사항: 다, 마, 바
② 기능적 요구사항: 가, 라, 마, 비기능적 요구사항: 나, 다, 바
③ 기능적 요구사항: 나, 다, 마, 비기능적 요구사항: 가, 라, 바
④ 기능적 요구사항: 다, 라, 마, 비기능적 요구사항: 가, 나, 바

정답 2번

간단한 해설

구체적인 화면·입력·권한 구분의 사양은 이 문항에서 기능적 요구사항으로, **응답시간·보안성·가용성의 품질 조건**은 비기능적 요구사항으로 분류한다. 같은 보안 관련 내용이라도 구체적인 권한 구분 기능과 포괄적인 보안성 요구는 다르다.

NoteBase

구성 요소별 해설

가 → 기능적 요구사항: 버튼의 위치를 하단 중앙으로 정하는 구체적인 사용자 인터페이스 사양이다. 이 문항은 이러한 화면 인터페이스 사양을 기능적 요구사항에 포함하는 분류를 사용한다. 다만 요구공학의 분류체계에 따라 화면 배치 같은 사항을 별도의 인터페이스 요구사항이나 설계 제약으로 다루기도 한다.

나 → 비기능적 요구사항: 검색 결과를 제공한다는 기능 자체보다 **3초 이내에 표시해야 한다는 응답시간**을 규정한다. 이는 성능 요구사항이다.

다 → 비기능적 요구사항: 보안성에 관한 요구 사양을 만족하라는 포괄적인 품질 조건이다. 특정 인증·권한검사 동작을 명시한 기능 요구가 아니라 보안이라는 품질 속성에 초점을 둔다.

라 → 기능적 요구사항: 사용자 입력값에 적용할 범위와 단위를 정하는 구체적인 입력 처리 사양이다. 시스템이 어떤 값을 받아들이고 어떤 단위로 해석·처리할지와 관련된다.

마 → 기능적 요구사항: 사용자의 권한에 따라 분류하고 구분하는 구체적인 기능이다. 보안을 목적으로 하더라도 시스템이 수행할 동작을 명시했으므로, 다의 포괄적 보안성 조건과 구별한다.

바 → 비기능적 요구사항: 가용성을 최소 90% 이상 보장하라는 것은 사용 가능한 상태를 유지하는 정도에 대한 품질 요구이다. 어떤 업무 기능을 수행하는지보다 서비스의 가용 수준을 수치로 규정한다.

따라서 이 문항의 분류는 **기능적 요구사항: 가·라·마, 비기능적 요구사항: 나·다·바**이다.

NoteBase

문 18. 다음 중 모델링에 관련된 올바른 설명을 모두 고른 것은?

가. 절차 중심 방법은 데이터의 구조와 이를 이용한 함수들의 집합으로 구성된다.
나. 객체 지향 방법은 객체에 데이터와 이에 대한 함수를 모아 놓은 것이다.
다. 클래스는 각각의 개체들이 갖는 속성과 연산을 정의하는 템플릿이다.
라. 캡슐화는 정보은닉(Information hiding)으로 오퍼레이션과 속성을 외부에 노출되지 않게 할 수 있다.
마. 다중 상속(Multiple inheritance)은 여러 클래스의 메소드를 상속할 수 있는 기능으로, 객체지향 언어의 핵심 요소이다.
바. 다형성(Polymorphism)은 동일한 명칭의 메소드로 여러 클래스에 정의하여 객체에 따라 다른 처리를 가능하게 한다.

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

정답 4번

간단한 해설

다중 상속은 여러 상위 클래스로부터 특성을 물려받는 기능이지만, **모든 객체지향 언어에 필수적인 것은 아니다**. 일반적인 상속 개념을 다중 상속과 동일시하지 않아야 한다.

NoteBase

보기별 해설

가. O: 절차 중심 방법은 처리 절차와 함수를 중심으로 시스템을 구성하고, 함수들이 데이터를 입력받거나 공유하여 처리한다. 데이터 구조와 이를 이용하는 함수들이 존재하지만, **데이터와 함수를 객체로 묶는 방식**을 중심으로 삼는 것은 아니다.

나. O: 객체 지향 방법은 관련 데이터와 그 데이터를 다루는 연산을 하나의 객체에 모은다. 객체는 상태를 나타내는 데이터와 행동을 나타내는 메소드를 가지며, 객체 사이의 상호작용으로 시스템을 구성한다.

다. O: 클래스는 객체들이 공통으로 갖는 속성과 연산을 정의하는 **템플릿**이다. 그 정의에 따라 생성된 구체적인 객체를 인스턴스라고 하며, 같은 클래스의 객체라도 각자의 상태 값은 다를 수 있다.

라. O: 캡슐화는 관련 속성과 연산을 묶고 접근 가능한 경계를 정한다. 이를 통해 내부 속성이나 구현용 연산을 외부에 직접 노출하지 않는 **정보은닉**을 실현할 수 있다. 모든 연산을 숨긴다는 뜻은 아니며, 필요한 공개 인터페이스를 통해 객체를 사용하게 한다.

마. X: 다중 상속은 하나의 클래스가 둘 이상의 상위 클래스로부터 특성을 상속받는 기능이다. 메소드뿐 아니라 언어와 정의에 따라 속성도 상속 대상이 될 수 있다. 그러나 **다중 상속이 모든 객체지향 언어의 필수 요소인 것은 아니다**. 단일 클래스 상속만 허용하거나 다른 재사용 수단을 제공하는 언어도 객체지향을 지원하므로, 일반적인 상속 개념과 다중 상속을 동일시하면 안 된다.

바. O: 다형성은 같은 인터페이스나 메소드 호출이 실제 객체의 종류에 따라 서로 다른 구현으로 수행될 수 있게 하는 성질이다. 예를 들어 같은 그리기 요청에도 원 객체와 사각형 객체가 각자의 방식으로 처리한다. 핵심은 단순히 이름이 같다는 사실이 아니라 **공통 요청에 대한 객체별 동작의 차이**이다.

적절한 설명은 가, 나, 다, 라, 바이므로 ④와 일치한다.

NoteBase

문 19. 다음 중 설계원칙에 대한 올바른 설명을 모두 고른 것은?

가. 단일 책임(Single responsibility) 원칙은 한 클래스는 하나의 책임을 갖도록 설계함을 의미한다.
나. 리스코프 치환(Liskov substitution) 원칙은 상위 클래스가 사용된 곳을 상속받은 하위 클래스로 대체하여도 문제가 발생하지 않도록 하는 것이다.
다. 개방 폐쇄(Open-closed) 원칙은 클래스의 공통 기본기능을 변경하지 않도록 하면서 새로운 기능은 확장 사용하는 것이다.
라. 인터페이스 분리(Interface segregation) 원칙은 서로 다른 성격의 인터페이스를 분리, 클라이언트 쪽에서 여러 인터페이스를 갖도록 하는 것이다.
마. 의존 관계 역전(Dependency inversion) 원칙은 추상 클래스(Abstraction class)가 저수준의 구체적 클래스에 의존하도록 설계하는 원칙을 의미한다.

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

정답 1번

간단한 해설

의존 관계 역전 원칙에서는 **구체적 구현이 추상화에 의존**해야 한다. 추상 클래스가 저수준의 구체적 클래스에 의존하도록 한다는 설명은 의존 방향이 반대이다.

NoteBase

보기별 해설

가. O: 단일 책임 원칙은 하나의 클래스가 서로 무관한 책임을 함께 맡지 않도록 하는 것이다. 책임은 **클래스를 변경해야 하는 이유**와 연결되며, 한 클래스에 메서드를 하나만 두라는 뜻은 아니다.

나. O: 리스코프 치환 원칙은 상위 타입을 기대하는 코드에 하위 타입의 객체를 넣어도 프로그램의 올바른 동작이 유지되어야 한다는 원칙이다. 하위 클래스는 **상위 타입이 약속한 동작과 계약**을 깨뜨리지 않아야 한다.

다. O: 개방 폐쇄 원칙은 기존 코드를 계속 고치지 않고도 새로운 동작을 추가할 수 있도록 설계하는 것이다. 즉 **확장에는 열려 있고 수정에는 닫혀 있어야** 하며, 추상화와 다형성 등을 이용해 기존 코드의 변경 영향을 줄인다.

라. O: 인터페이스 분리 원칙은 성격과 사용 목적이 다른 기능을 하나의 큰 인터페이스에 몰아넣지 않고 역할별로 나누는 것이다. 클라이언트가 필요한 인터페이스만 이용하게 하여 **사용하지 않는 기능에 의존하지 않도록** 한다. 모든 클라이언트에 여러 인터페이스를 무조건 강제하는 뜻은 아니다.

마. X: 의존 관계 역전 원칙은 고수준 정책과 저수준 구현이 모두 추상화에 의존하도록 하는 것이다. **추상화가 구체적 구현에 의존하는 것이 아니라 구체적 구현이 추상화에 의존**해야 한다. 제시문은 의존 방향을 거꾸로 설명하였다.

올바른 설명은 가, 나, 다, 라이므로 ①에 해당한다.

NoteBase

문 20. 다음 소프트웨어 모듈에 관련된 내용에서 결합도가 약한 것으로부터 강해지는 순서로 가장 올바르게 나열한 것은?

가. 전역 변수 데이터를 모듈간 공유한다.
나. 모듈의 호출에서 레코드나 배열, 구조체의 전달이 이루어진다.
다. 모듈의 호출에서 매개변수를 통한 값의 전달이 이루어진다.
라. 호출하는 모듈의 매개변수 값으로 호출되는 모듈의 내부 제어 흐름이 결정된다.
마. 호출하는 모듈이 호출되는 모듈의 내부 데이터 값을 수정한다.

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

정답 4번

간단한 해설

결합도는 모듈 사이의 의존 정도이며, **필요한 값만 전달하는 경우보다 상대의 제어·공유 자료·내부 구현에 관여할수록 강해진다**. 전역 자료를 공유하는 공통 결합보다 상대 모듈의 내부 데이터를 직접 수정하는 내용 결합이 더 강하다.

NoteBase

올바른 순서

다 → 나 → 라 → 가 → 마

다. 자료 결합도: 모듈이 필요한 값을 매개변수로 전달한다. 다른 모듈의 내부 구조나 제어 방법을 알지 않고 필요한 자료만 주고받으므로, 제시된 유형 중 의존성이 가장 낮다.

나. 스탬프 결합도: 레코드·배열·구조체 같은 복합 자료구조를 통째로 전달한다. 개별 값뿐 아니라 **전달되는 자료구조의 형식에도 의존**하므로 자료 결합도보다 강하다.

라. 제어 결합도: 전달한 플래그나 제어 값이 상대 모듈이 실행할 처리 경로를 정한다. 단순히 자료 형식을 공유하는 것을 넘어 **상대 모듈의 처리 방식에 관여**하므로 스탬프 결합도보다 강하다.

가. 공통 결합도: 여러 모듈이 전역 변수와 같은 공통 자료 영역을 공유한다. 한 모듈의 변경이 공유 자료를 사용하는 다른 모듈에도 영향을 미칠 수 있어, 매개변수를 통한 제어 결합보다 의존성이 강하다.

마. 내용 결합도: 한 모듈이 다른 모듈의 내부 자료를 직접 참조하거나 수정한다. 상대의 내부 구현에 직접 의존하여 모듈의 경계를 침범하므로, 제시된 유형 중 결합도가 가장 강하다.

NoteBase

문 21. 소프트웨어 개발을 위한 SDLC(Software Development Life Cycle)의 요구사항 분석 단계에서 작성하는 산출물 중 가장 올바르지 않은 것은?

① 사용자 스토리(User story)

② 프로세스 분해 다이어그램(Process decomposition diagram)

③ 유스케이스 다이어그램(Use case diagram)

④ 데이터 플로우 다이어그램(Data flow diagram)

정답 2번

간단한 해설

이 문항은 프로세스 분해 다이어그램을 **처리 기능의 계층 구조를 구체화하는 설계 산출물**로 구분한다. 다만 기능 분해는 요구사항 분석에서도 활용되므로 **설계 단계에서만 쓰이는 도구는 아니다**.

출제 포인트

사용자 요구를 표현하는 요구사항 분석 산출물과 처리 기능의 구조를 구체화하는 산출물을 구분하는 문제이다.

핵심 개념

요구사항 분석은 사용자에게 필요한 기능과 시스템이 충족해야 할 조건을 파악하고 명세하는 단계이다. 설계는 그 요구를 실현할 시스템의 구조와 구성 방식을 구체화하는 단계이다.
사용자 스토리는 사용자의 역할·목표·가치를 짧은 문장으로 표현한다. 유스케이스 다이어그램은 액터와 시스템이 제공하는 기능의 관계를 표현한다. 데이터 플로우 다이어그램은 외부 개체·처리 과정·자료 저장소 사이의 데이터 흐름을 표현한다. 프로세스 분해 다이어그램은 상위 처리 기능을 하위 기능으로 나누어 계층 관계를 표현한다.
예를 들어 '회원이 주문 내역을 조회한다'는 사용자 요구이며, '주문 처리'를 '주문 접수·결제·배송'으로 나누는 것은 기능 분해이다. 이러한 분해 자체는 분석과 설계 모두에 활용될 수 있다.

NoteBase

선지별 해설

O/X는 이 문항의 단계 구분에서 요구사항 분석 산출물에 해당하는지를 나타낸다.
① O: 사용자 스토리는 사용자가 원하는 기능과 그 목적을 사용자 관점에서 기록한다. 요구사항을 표현하는 산출물이므로 이 문항에서 제외할 대상이 아니다.
② X: 프로세스 분해 다이어그램은 처리 기능을 하위 기능으로 나누어 계층 구조를 나타낸다. 이 문항의 단계 구분에서는 이를 설계 쪽 산출물로 보아 요구사항 분석의 대표 산출물에서 제외한다. 다만 프로세스 분해가 요구사항 분석에서 사용될 수 없다는 뜻은 아니다.
③ O: 유스케이스 다이어그램은 시스템 밖의 액터가 어떤 기능을 이용하는지 나타낸다. 시스템의 기능과 범위를 파악하는 요구사항 분석에 적합하므로 제외할 대상이 아니다.
④ O: 데이터 플로우 다이어그램은 입력 자료가 어떤 처리를 거쳐 어디로 전달되거나 저장되는지 나타낸다. 특히 논리적 데이터 흐름은 구현 세부사항 없이 필요한 처리를 분석하므로 요구사항 분석 산출물에 해당한다.

오답 포인트

②의 '분해'를 ④의 데이터 흐름도 상세화와 혼동하기 쉽다. 프로세스 분해 다이어그램은 기능 사이의 상하위 관계에, 데이터 플로우 다이어그램은 데이터의 이동과 변환에 초점을 둔다. 이 문항의 구분을 '프로세스 분해는 분석에서 절대 사용하지 않는다'는 일반 규칙으로 확대하면 안 된다.

다시 나오면 이렇게 판단

먼저 사용자의 요구나 시스템의 기능·데이터 흐름을 표현하는지, 그 기능의 구성 구조를 구체화하는지 살핀다. 단계별 산출물을 묻는 문제에서는 문항이 전제한 구분을 따르되, 분석과 설계에서 함께 쓰일 수 있는 도구의 활용 범위도 구별한다.

암기 포인트

사용자 스토리=사용자의 목표, 유스케이스=액터와 기능, 데이터 흐름도=데이터 이동, 프로세스 분해도=기능의 계층 관계.

NoteBase

문 22. 다음 C프로그램의 함수 TaxDeduct()에서 나타나는 응집에 대한 설명으로 가장 올바른 것은?

```
int TaxDeduct(int ftype, int n)
{
    int i;
    i = 0;
    switch (ftype) {
        case 1: // 미성년 부양가족 인원당 200,000 공제
                i = n * 200000;
                break;
        case 2: // 출산 특별공제 신생아당 1,000,000 공제
                i = n * 1000000;
                break;
        case 3: // 미혼청년공제 500,000 n 해당없음
                i = 500000;
                break;
    }
    i = i + 50000; // 공통기본공제 50,000 추가
    return i;
}
```

① 순차적 응집

② 논리적 응집

③ 시간적 응집

④ 절차적 응집

정답 2번

간단한 해설

TaxDeduct()는 여러 공제 계산을 한 함수에 모아 두고, **외부에서 전달받은 ftype으로 실행할 기능을 선택**한다. 이처럼 논리적으로 비슷한 작업을 선택적으로 묶은 구조는 논리적 응집이다.

NoteBase

핵심 풀이

TaxDeduct()는 ftype과 n을 전달받아 i를 0으로 초기화한 뒤, switch(ftype)에서 공제 종류를 선택한다. ftype이 1이면 미성년 부양가족 수에 따라 200000*n*, 2이면 신생아 수에 따라 1000000*n*, 3이면 n과 관계없이 500000을 i에 대입한다. 각 case의 break 때문에 이 세 계산은 차례로 모두 실행되는 것이 아니라 하나만 선택된다.

선택된 계산이 끝난 뒤에는 공통 기본공제 50000을 더하고 i를 반환한다. 코드의 공제명과 금액은 주어진 예제의 내용일 뿐, 실제 세법상 공제기준을 설명하는 것은 아니다.

판별의 핵심은 **유사한 여러 작업을 하나의 모듈에 넣고 제어 매개변수로 선택**한다는 점이다. 따라서 함수 전체의 대표적인 응집 유형은 **논리적 응집**이다.

순차적 응집은 한 처리의 출력이 다음 처리의 입력이 되는 데이터 흐름을 중심으로 묶인 경우이다. 이 함수에도 공통 금액을 더하는 후속 연산은 있지만, 서로 다른 case를 선택하는 구조를 설명하는 핵심은 아니다. 시간적 응집은 초기화 등 같은 시점에 수행한다는 이유로 작업을 묶는 것이고, 절차적 응집은 정해진 실행 순서를 중심으로 묶는 것이다. 따라서 문장이 순서대로 실행되거나 i가 초기화된다는 사실만으로 이들을 선택해서는 안 된다.

NoteBase

문 23. 소프트웨어 테스트에 대한 설명으로 가장 올바르지 않은 것은?

① 테스트 오라클은 테스트 결과의 판단 기준이다.

② 휴리스틱 오라클은 모든 임의의 입력값에 대한 발생 오류의 결과를 제공하는 방법이다.

③ 상태 기반 테스트는 상태와 전이(State & transition)를 중점적으로 테스트하는 방법이다.

④ 테스트 하니스는 일부 기능만 테스트하기 위하여 소프트웨어를 변경하는 방법이다.

정답 2번

간단한 해설

휴리스틱 오라클은 경험적 규칙이나 추정으로 결과를 판단하므로 **모든 입력값에 대한 정확한 결과를 보장하지 않는다**.

선지별 해설

① O: 테스트 오라클은 실제 실행 결과가 기대한 결과와 일치하는지 판단하는 기준이나 판단 수단이다. 요구 명세, 계산된 기대값 등이 **정상·오류 판정의 근거**가 될 수 있다.

② X: 휴리스틱 오라클은 경험적 규칙이나 추정에 의존하여 결과의 타당성을 판단한다. **모든 입력에 대해 정확한 기대 결과를 보장하지 않는다**. 모든 입력의 결과를 정확히 판단할 수 있는 완전한 오라클과 구별된다.

③ O: 상태 기반 테스트는 시스템이 현재 어떤 상태에 있고 특정 입력이나 사건에 의해 어떤 상태로 바뀌어야 하는지를 확인한다. **상태와 상태 전이**를 기준으로 정상 전이, 허용되지 않는 전이 등을 검사한다.

④ O: 일부 기능을 따로 시험하도록 호출부나 의존 기능을 시험용 코드로 구성·대체한다는 취지이다. 엄밀히 테스트 하니스는 **드라이버·스텁 등 테스트 실행을 지원하는 환경과 도구의 집합**이며, 대상 기능을 호출하고 주변 기능을 대신하여 독립적인 테스트를 가능하게 한다. 제품의 기능 자체를 반드시 변경해야 한다는 뜻은 아니다.

따라서 휴리스틱 오라클이 모든 입력의 결과를 제공한다고 한 ②가 가장 올바르지 않다.

NoteBase

문 24. 구조적 분석과 설계에서 많이 활용되는 자료 흐름도(Data flow diagram)의 구성 요소로 가장 올바르지 않은 것은?

① 데이터 저장소(Data store)

② 외부 엔터티(External entity)

③ 프로세스(Process)

④ 의사결정(Decision)

정답 4번

간단한 해설

자료 흐름도의 기본 요소는 프로세스·자료 흐름·자료 저장소·외부 엔터티이다. **의사결정은 조건 분기를 표현하는 순서도의 요소**이며, 자료 흐름도의 기본 구성 요소가 아니다.

선지별 해설

자료 흐름도는 시스템에서 **어떤 자료가 어디로 이동하고 어떻게 변환되는지**를 나타낸다. 기본 구성 요소는 프로세스, 자료 흐름, 자료 저장소, 외부 엔터티이다.

① O: 자료 저장소는 처리를 위해 보관하거나 처리 결과를 저장하는 자료의 집합을 나타낸다. 예를 들어 주문 기록이나 고객 정보처럼 나중에 읽거나 갱신하는 자료가 이에 해당한다.

② O: 외부 엔터티는 분석 대상 시스템의 경계 밖에서 자료를 보내거나 받는 주체이다. 고객·다른 시스템 등이 될 수 있으며, 자료의 외부 출처 또는 도착점을 나타낸다.

③ O: 프로세스는 입력 자료를 처리하여 출력 자료로 변환하는 기능이다. 예를 들어 주문 자료를 받아 청구 자료를 만드는 처리처럼, 단순한 보관이 아니라 **자료의 변환**을 표현한다.

④ X: 의사결정은 조건에 따라 실행 경로가 갈라지는 분기를 나타내는 순서도의 요소이다. DFD의 독립적인 기본 구성 요소가 아니며, DFD는 제어 순서나 조건 분기 자체보다 자료의 이동과 처리를 표현한다. 선택지에는 없지만 **자료 흐름**은 DFD의 기본 요소로서 구성 요소 사이에 이동하는 자료를 화살표로 나타낸다.

NoteBase

문 25. 다음 보기에서 소프트웨어개발 분석, 설계 단계 중 같은 단계에서 사용되는 다이어그램들을 모두 고른 것은?

가. 자료 흐름도 - 시퀀스 다이어그램
나. 자료 흐름도 - 유스케이스 다이어그램
다. 구조적 도표 - 클래스 다이어그램
라. 구조적 도표 - 유스케이스 다이어그램

① 가, 나
② 나, 다
③ 가, 나, 다
④ 나, 다, 라

정답 2번

간단한 해설

자료 흐름도와 유스케이스 다이어그램은 **요구 기능과 자료 흐름을 분석**하는 데, 구조적 도표와 클래스 다이어그램은 **이를 구현할 내부 구조를 설계**하는 데 대응한다. 이는 대표적인 모델링 목적에 따른 구분이며, 특정 다이어그램을 한 단계에서만 쓸 수 있다는 뜻은 아니다.

NoteBase

보기별 해설

분석은 시스템이 무엇을 해야 하는지 파악하고, 설계는 그 기능을 어떤 내부 구성과 협력 방식으로 실현할지 구체화한다. 이 문항은 다이어그램의 대표적인 모델링 목적을 기준으로 두 단계를 구분한다.

가. X: 자료 흐름도는 자료의 입력·처리·저장·출력과 그 사이의 흐름을 표현하여 요구사항을 분석한다. 시퀀스 다이어그램은 객체 간 메시지 교환을 시간 순서로 표현하며, **내부 객체들이 기능을 실현하는 협력 방식**을 구체화할 때 설계에 사용한다. 이 목적에 따라 구분하면 분석과 설계의 짝이다. 다만 시퀀스 다이어그램으로 사용자와 시스템의 상호작용이나 요구 시나리오를 분석할 수도 있으므로, 분석에서 사용할 수 없는 도구로 이해해서는 안 된다.

나. O: 자료 흐름도는 필요한 자료 처리와 정보의 흐름을, 유스케이스 다이어그램은 외부 액터와 시스템이 제공해야 할 기능의 관계를 표현한다. 두 도구 모두 구현할 모듈을 정하기에 앞서 **시스템이 수행해야 할 기능을 파악**하는 분석에 대응한다.

다. O: 구조적 도표는 프로그램을 구성하는 모듈의 계층과 호출 관계를 표현한다. 클래스 다이어그램은 클래스의 속성·연산과 클래스 사이의 관계를 표현하며, 이를 구체화하여 구현할 클래스 구조를 설계한다. 표현 단위는 모듈과 클래스로 다르지만, **기능을 실현할 내부 구성 요소와 관계**를 정한다는 점에서 모두 설계에 대응한다. 클래스 다이어그램도 분석 단계에서 문제 영역의 개념과 관계를 파악하는 데 사용할 수 있다.

라. X: 구조적 도표는 **내부 모듈의 구성과 호출 관계**를 설계하고, 유스케이스 다이어그램은 **사용자 관점의 요구 기능**을 분석한다. 대표적인 목적에 따라 각각 설계와 분석에 대응하므로 같은 단계의 짝으로 묶지 않는다.

이 기준에서 같은 단계에 대응하는 짝은 나와 다이며, ②의 조합과 일치한다.

NoteBase

문 26. 다음 보기에서 소프트웨어 아키텍쳐 MVC의 특징을 모두 고른 것은?

가. 사용자 인터페이스로부터 비즈니스 로직을 분리하여 애플리케이션의 시각적 요소나 비즈니스 로직을 서로 영향 없이 쉽게 고칠 수 있다.
나. 컴포넌트의 결합이 강해 다른 부분에 영향을 주지 않고 수정할 수 없다.
다. 하나의 모델을 위하여 다수의 다른 뷰를 쉽게 제공할 수 있다.
라. 응용프로그램은 모델-뷰-컨트롤러 세 가지 구성 요소로 이루어진다.

① 가, 나
② 가, 다
③ 가, 나, 다
④ 가, 다, 라

정답 4번

간단한 해설

MVC는 **데이터·업무 로직, 화면 표시, 입력 처리의 책임을 분리**하여 변경의 영향을 줄인다. 구성 요소를 강하게 결합시키는 구조가 아니며, 하나의 모델을 여러 뷰로 표현할 수 있다.

보기별 해설

가. O: MVC는 데이터·업무 처리와 화면 표시의 책임을 분리한다. 따라서 인터페이스가 유지되는 범위에서 화면 표현을 바꾸거나 업무 로직을 수정할 때 **다른 부분의 변경을 최소화**할 수 있다. '서로 영향 없이'는 이러한 분리의 장점을 뜻하며, 모든 변경에서 영향이 절대로 없다는 뜻은 아니다.

나. X: MVC는 역할 분리로 구성 요소 간 결합을 낮추어 유지보수를 쉽게 하는 구조이다. 구성 요소가 서로 협력하기 때문에 의존성이 완전히 없어지는 것은 아니지만, 결합이 강하여 독립적인 수정이 불가능하다는 설명은 설계 취지와 반대이다.

다. O: 모델은 데이터와 업무 규칙을 담당하고 뷰는 이를 표시한다. **같은 모델을 서로 다른 방식으로 표현하는 여러 뷰**를 둘 수 있으므로, 동일한 자료를 목록이나 그래프 등으로 나타내기 쉽다.

라. O: 모델(Model)은 데이터와 업무 로직을, 뷰(View)는 사용자에게 보여 주는 표현을, 컨트롤러(Controller)는 사용자 입력의 해석과 그에 따른 처리를 조정한다. 이 세 책임으로 응용프로그램을 나누는 것이 MVC의 기본 구성이다.

따라서 MVC의 특징에 해당하는 보기는 가, 다, 라이며, 이 조합은 ④와 일치한다.

NoteBase

문 27. 다음 보기에서 객체간의 상호 작용을 설명하고 있는 동적 UML다이어그램을 모두 고른 것은?

가. 시퀀스 다이어그램
나. 배치 다이어그램
다. 타이밍 다이어그램
라. 유스케이스 다이어그램

① 가, 나
② 가, 다
③ 가, 나, 다
④ 가, 다, 라

정답 4번

간단한 해설

보기 중 시퀀스·타이밍·유스케이스는 **행위 측면의 다이어그램**이고, 배치는 구조 다이어그램이다. 다만 **유스케이스를 엄밀한 상호작용 다이어그램의 하위 유형으로 분류하지는 않는다**는 점을 구별해야 한다.

보기별 해설

문항의 '동적 UML 다이어그램'은 구조 다이어그램과 대비되는 행위 다이어그램의 넓은 의미로 구분한다. UML에서 상호작용 다이어그램은 행위 다이어그램의 일부이므로 두 범주의 범위가 완전히 같은 것은 아니다.

가. O: 시퀀스 다이어그램은 참여하는 객체 등의 생명선 사이에서 오가는 메시지를 시간적 순서에 따라 나타낸다. 행위 다이어그램이면서 상호작용 다이어그램에 해당한다.

나. X: 배치 다이어그램은 소프트웨어 산출물이 어떤 하드웨어 노드나 실행환경에 배치되는지 표현한다. 이는 실행환경의 **구조를 나타내는 구조 다이어그램**으로, 동적 행위 다이어그램이 아니다.

다. O: 타이밍 다이어그램은 시간 흐름에 따른 상태·값의 변화와 시간 제약을 표현한다. 객체 등의 시간적 행위를 분석하는 상호작용 다이어그램이므로 행위 다이어그램에도 포함된다.

라. O: 유스케이스 다이어그램은 액터가 시스템을 통해 달성하려는 기능과 관계를 나타내는 행위 다이어그램이다. 따라서 넓은 의미의 동적·행위 다이어그램에는 포함된다. 다만 **시퀀스·타이밍처럼 메시지 교환을 상세히 표현하는 상호작용 다이어그램은 아니다**.

따라서 동적·행위 다이어그램에 해당하는 보기는 **가, 다, 라**이다.

NoteBase

문 28. 시퀀스 다이어그램(Sequence diagram)에 대한 설명으로 가장 올바르지 않은 것은?

① 유스케이스에 관련된 추가 객체(Participating object)를 찾을 수 있다.

② 다이어그램은 객체, 메시지, 활성막대, 프레임 등으로 구성된다.

③ 시스템의 동작을 정형화하고 객체들의 메시지 교환을 울타리 형태로 시각화한다.

④ 제어 흐름, 동기화, 병렬 수행 등의 기능을 분석할 수 있다.

정답 4번

간단한 해설

④의 제어 흐름·동기화·병렬 수행 분석은 **활동 다이어그램의 대표적 용도**에 해당한다. 시퀀스 다이어그램의 중심은 **객체 간 메시지 교환의 시간적 순서**이다. 다만 UML 2의 시퀀스 다이어그램도 par 결합 조각으로 병렬 상호작용을 표현할 수 있으므로, 병렬 수행을 전혀 표현할 수 없다는 뜻은 아니다.

선지별 해설

① O: 유스케이스를 실제로 수행하려면 어떤 객체들이 어떤 메시지를 주고받아야 하는지 구체화해야 한다. 시퀀스 다이어그램을 작성하는 과정에서 메시지를 보내거나 받을 **참여 객체를 추가로 발견**할 수 있다.

② O: 객체는 생명선으로, 객체 사이의 통신은 메시지 화살표로 나타낸다. 활성막대는 생명선 위에서 **동작을 실행하는 구간**을 표시하고, 프레임은 상호작용이나 결합 조각의 범위를 감싼다. 이들은 시퀀스 다이어그램의 주요 구성 요소이다.

③ O: 시퀀스 다이어그램은 객체들을 가로 방향으로 배치하고 각 객체의 생명선을 세로로 그린 뒤, 그 사이에 메시지 화살표를 표시한다. 세로 생명선과 이를 잇는 메시지들이 울타리와 비슷한 형태를 이루며, 메시지는 기본적으로 **위에서 아래로 시간적 순서**에 따라 읽는다. 이를 통해 시스템의 동작을 객체 간 상호작용으로 구체화한다.

④ X: 작업의 제어 흐름을 나타내고, fork로 병렬 흐름을 나누거나 join으로 흐름을 동기화하는 것은 **활동 다이어그램의 대표적 용도**이다. 시퀀스 다이어그램은 특정 시나리오에서 **어떤 객체가 누구에게 어떤 순서로 메시지를 보내는지**를 중심으로 표현한다. 따라서 다이어그램의 주된 목적을 구별하는 이 문항에서는 활동 다이어그램의 용도를 제시한 설명이 가장 적절하지 않다. 다만 UML 2의 시퀀스 다이어그램에도 par 결합 조각과 동기·비동기 메시지가 있으므로, 동기화나 병렬 수행을 표현할 수 없다고 일반화해서는 안 된다.

NoteBase

문 29. 다음 중 유지보수에 대한 서술로 가장 올바르지 않은 것은?

① 수정 유지보수(Corrective maintenance) : 소프트웨어 설치 후에 발견된 결함을 수정하여 바로잡기 위해서 하는 유지보수

② 적응 유지보수(Adaptive maintenance) : 소프트웨어 설치 후에 이루어지는 환경 변화에 대응하기 위하여 실시하는 유지보수

③ 완전 유지보수(Perfective maintenance) : 소프트웨어 설치 후에 다양한 대상 시스템에 동작성을 시험, 완전성을 확인하기 위하여 실시하는 유지보수

④ 예방 유지보수(Preventive maintenance) : 시스템의 결함을 예방하며 잠재적 문제점을 보완하기 위하여 실시하는 유지보수

정답 3번

간단한 해설

완전 유지보수는 다양한 시스템에서 완전성을 시험하는 것이 아니라 **새로운 요구를 반영하고 기능·성능·유지보수성 등을 개선**하는 활동이다.

선지별 해설

① O: 수정 유지보수는 운영 중 발견된 오류나 결함을 바로잡는 활동이다. 잘못된 계산이나 비정상 종료의 원인을 고치는 것처럼 **이미 드러난 결함의 수정**이 목적이다.

② O: 적응 유지보수는 운영체제·하드웨어·외부 인터페이스 등의 환경이 바뀌어도 소프트웨어를 계속 사용할 수 있게 조정하는 활동이다. 기준은 기능의 단순 개선이 아니라 **변경된 외부 환경에 대한 대응**이다.

③ X: 완전 유지보수는 사용자의 새로운 요구를 반영하거나 성능·사용성·유지보수성 등을 개선하는 활동이다. **기능과 품질을 향상**하는 것이 핵심이며, 여러 시스템에서 동작을 시험하여 완전성을 확인한다는 뜻이 아니다.

④ O: 예방 유지보수는 앞으로 장애를 일으킬 수 있는 잠재적 결함과 취약한 구조를 미리 보완하는 활동이다. **문제가 실제 장애로 나타나기 전에** 수정하여 위험을 줄인다는 점에서 발견된 오류를 고치는 수정 유지보수와 구별된다.

따라서 완전 유지보수의 목적을 잘못 설명한 ③이 가장 올바르지 않다.

NoteBase

문 30. 소프트웨어 인수테스트에 대한 설명으로 가장 올바르지 않은 것은?

① 인수테스트는 크게 알파테스트와 베타테스트로 구분한다.

② 알파테스트는 인수자가 설치될 인수시스템 환경에서 테스트한다.

③ 베타테스트는 인수자가 다양한 외부 컴퓨팅 환경에서 테스트한다.

④ 소프트웨어 사용자가 시험하는 매우 현실적인 테스트이다.

정답 2번

간단한 해설

알파테스트는 사용자가 **개발자 측 시험 환경**에서 수행하는 인수테스트이다. 개발자 측 환경 밖의 실제 사용자 환경에서 시험하는 것은 베타테스트이다.

선지별 해설

인수테스트는 소프트웨어가 사용자의 업무 요구와 인수 기준을 만족하는지 확인하여 실제 사용·인수 가능 여부를 판단하는 시험이다.

① O: 사용자 참여 방식과 시험 환경을 기준으로 알파테스트와 베타테스트를 대표적인 인수테스트 방식으로 구분한다. 알파는 개발자 측 환경, 베타는 개발자 측 환경 밖에서 수행한다. 다만 인수테스트의 분류가 이 둘로만 한정되는 것은 아니며, 사용자·운영·계약상 인수테스트 등도 있다.

② X: 알파테스트는 잠재 사용자나 고객, 개발조직과 독립된 시험 담당자 등이 **개발자 측의 시험 환경**에서 수행한다. 인수자의 실제 설치·운영 환경에서 하는 시험으로 설명한 것은 알파의 시험 장소를 잘못 연결한 것이다. 사용자 측 환경에서 실제 사용해 보는 방식은 베타테스트에 해당한다.

③ O: 베타테스트는 사용자나 고객이 **개발자 측 시험 환경 밖의 실제 사용 환경**에서 수행한다. 여러 사용자의 장비·운영체제·설정 등 다양한 조건에서 사용해 봄으로써 개발 환경에서 드러나지 않은 문제와 사용자 의견을 얻을 수 있다.

④ O: 실제 사용자 관점에서 업무 시나리오와 요구사항의 충족 여부를 확인하므로 현실적인 사용 적합성을 평가한다. 단순히 내부 코드가 정상 작동하는지만 보는 것이 아니라 **사용자가 필요로 하는 일을 수행할 수 있는지**를 판단하는 데 초점이 있다.

따라서 알파테스트의 수행 환경을 잘못 설명한 ②가 가장 올바르지 않다.

NoteBase

문 31. 다음 보기에서 설명하는 테스트의 종류로 가장 올바른 것은?

시스템을 변경한 후 변경된 부분이 기존 시스템에 부작용을 일으키지 않는지를 조사하기 위해 이미 사용한 테스트 케이스를 재사용하여 실행하는 테스트

① 스트레스(Stress) 테스트
② 신뢰성(Reliability) 테스트
③ 회귀(Regression) 테스트
④ 인수(Acceptance) 테스트

정답 3번

간단한 해설

회귀 테스트는 시스템 변경으로 **기존에 정상 동작하던 기능에 부작용이 생기지 않았는지** 확인한다. 이미 사용한 테스트 케이스를 다시 실행하는 것은 이러한 변경 영향을 점검하는 방법이다.

대응 관계 해설

변경 이후의 부작용 점검 → 회귀 테스트: 프로그램을 수정하거나 기능을 추가하면 변경하지 않은 기존 기능에도 오류가 생길 수 있다. 회귀 테스트는 **기존에 정상 동작하던 기능이 변경 후에도 정상인지** 확인한다. 보기의 '시스템 변경 후', '기존 시스템의 부작용 조사', '기존 테스트 케이스 재사용'이 이 목적에 대응한다.

스트레스 테스트와의 구별: 처리 용량의 한계를 넘는 부하나 자원 부족 등 극한 조건에서 시스템의 동작과 한계를 조사한다. 변경으로 기존 기능이 손상되었는지 확인하는 것이 주목적은 아니다.

신뢰성 테스트와의 구별: 정해진 조건과 시간 동안 요구 기능을 고장 없이 수행하는 능력을 평가한다. 보기처럼 변경의 부작용을 찾기 위해 기존 테스트를 다시 실행하는 절차와 구별된다.

인수 테스트와의 구별: 사용자나 고객의 요구와 인수 기준을 만족하여 시스템을 받아들일 수 있는지 확인한다. 기존 테스트의 반복 실행 여부보다 인수 가능성의 판단이 핵심이다.

따라서 제시된 설명에 대응하는 명칭은 회귀(Regression) 테스트이며, ③과 일치한다.

NoteBase

문 32. 다음 소프트웨어 테스트 중 분류가 나머지와 다른 것은?

① 직교 배열(Orthogonal array) 테스트

② 자료 흐름(Data flow) 테스트

③ 페어와이즈 조합(Pairwise combination) 테스트

④ 결정 테이블(Decision table) 테스트

정답 2번

간단한 해설

자료 흐름 테스트는 프로그램 내부의 **변수 정의와 사용 경로**를 검사하는 화이트박스 기법이다. 나머지는 입력 조합이나 명세의 조건·결과를 바탕으로 설계하는 블랙박스 기법으로 묶인다.

선지별 해설

테스트 대상을 내부 코드 구조에서 도출하는지, 외부 입력·조건·기대 동작에서 도출하는지를 기준으로 비교한다.

① 직교 배열 테스트: 입력요인의 수준을 직교 배열에 배치하여 일부 조합을 체계적으로 시험한다. 모든 조합을 실행하기 어려울 때 조합 수를 줄이면서 요인 간 상호작용을 확인하려는 블랙박스 계열의 기법이다. 필요한 상호작용의 차수는 선택한 배열의 특성에 따라 달라진다.

② 자료 흐름 테스트: 변수에 값을 정의하는 지점과 그 값을 사용하는 지점 사이의 경로를 분석한다. 선택한 정의·사용 커버리지 기준을 만족하는 경로를 실행하도록 테스트를 설계하므로, **프로그램 내부 구조에 근거한 화이트박스 테스트**에 해당한다.

③ 페어와이즈 조합 테스트: 각 입력요인 두 개 사이의 값 조합이 적어도 한 번씩 포함되도록 테스트 집합을 구성한다. 명세상 입력값들의 조합을 이용하는 블랙박스 계열 기법이며, 모든 세 요인 이상의 상호작용이나 모든 결함의 검출을 보장하는 것은 아니다.

④ 결정 테이블 테스트: 조건들의 조합과 그에 따라 수행해야 할 동작을 표로 정리한 뒤 규칙별 테스트를 만든다. 요구사항이나 업무 규칙을 기반으로 하므로 블랙박스 테스트에 해당한다.

따라서 나머지와 분류가 다른 것은 **자료 흐름 테스트**이다.

NoteBase

문 33. 다음 보기에서 소프트웨어 통합 테스트 방법으로 올바른 것을 모두 고른 것은?

가. 상향식(Bottom up) 통합 테스트
나. 연쇄식(Threads) 통합 테스트
다. 스모크(Smoke) 테스트
라. 스트레스(Stress) 테스트
마. 빅뱅(Big bang) 테스트

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

정답 4번

간단한 해설

상향식·연쇄식·빅뱅은 모듈 통합 방식이며, 스모크 테스트는 통합 빌드의 기본 동작을 확인한다. 반면 **스트레스 테스트는 극한 부하나 자원 제약에서의 동작**을 검증하는 방법이다.

NoteBase

보기별 해설

통합 테스트는 개별적으로 개발·시험한 모듈을 결합하여 **모듈 사이의 인터페이스와 상호작용**을 검증한다. 통합 순서나 기능 단위의 결합 방식뿐 아니라 통합 빌드가 기본적으로 동작하는지 확인하는 방법도 사용한다.

가. O: 상향식 통합 테스트는 하위 모듈부터 결합·검증하고 상위 모듈로 올라가는 방식이다. 아직 결합되지 않은 상위 호출부를 대신하여 하위 모듈을 구동하는 테스트 드라이버를 사용할 수 있다.

나. O: 연쇄식 통합 테스트는 하나의 기능이나 이벤트를 처리하는 **실행 흐름에 필요한 모듈들을 묶어** 상호작용을 시험한다. 여기서 스레드는 기능 수행의 연쇄를 뜻하며, 단순히 운영체제 스레드의 성능만 시험한다는 의미는 아니다.

다. O: 스모크 테스트는 새로 통합한 빌드에 대해 핵심 기능이 동작하는지를 넓고 얕게 확인한다. 빌드가 더 상세한 테스트를 진행할 수 있는 상태인지 빠르게 판단하므로 **반복적인 통합 과정의 기본 동작 검증**에 활용된다. 모든 기능을 깊이 검증하는 완전한 테스트는 아니다.

라. X: 스트레스 테스트는 과도한 부하나 제한된 자원 등 극한 조건에서 시스템의 한계·안정성·실패 후 동작 등을 확인한다. 이는 테스트의 조건과 목적에 따른 분류로, 이 문제에서 묻는 **모듈 결합과 상호작용을 검증하는 통합 방법**으로 분류하지 않는다. 통합된 시스템을 대상으로 스트레스 테스트를 실시할 수 없다는 뜻은 아니다.

마. O: 빅뱅 테스트는 준비된 모듈들을 한꺼번에 결합하여 전체를 시험하는 비점진적 통합 방식이다. 통합 절차는 단순하지만 여러 모듈이 동시에 결합되므로 실패가 발생하면 원인 모듈이나 인터페이스를 좁혀 찾기 어렵다.

해당하는 것은 가, 나, 다, 마이므로 ④와 일치한다.

NoteBase

문 34. 소프트웨어의 검증(Verification)과 확인(Validation)에 대한 설명으로 가장 올바른 것은?

① 확인은 사용자 관점에서 설계명세서들을 점검하는 작업이다.

② 검증과 확인은 모두 계획단계에서 명세서를 점검하는 작업이다.

③ 확인은 사용자 관점에서 발주자와 합의한 요구사항들을 점검하는 작업이다.

④ 검증은 개발자 관점에서 계획서를 점검하는 작업이다.

정답 3번

간단한 해설

확인(Validation)은 소프트웨어가 **사용자의 요구와 의도한 용도를 충족하는지** 평가한다. 검증(Verification)은 산출물이 정해진 명세를 따르는지 점검하는 관점이다.

선지별 해설

① X: 확인의 핵심 기준은 설계명세서 자체가 아니라 **사용자의 실제 요구와 의도한 용도**이다. 설계가 정해진 요구사항에 맞게 작성되었는지, 구현이 설계명세를 따르는지 등을 점검하는 것은 검증의 대표적인 관점이다.

② X: 검증과 확인은 계획단계의 명세서 점검에 한정되지 않는다. 요구사항·설계·구현·시험 등 **개발 생명주기 전반**에서 산출물의 적합성과 사용자 요구 충족 여부를 점검할 수 있다.

③ O: 확인은 개발한 소프트웨어가 사용자가 필요로 하는 기능과 사용 목적을 충족하는지 평가한다. 발주자와 합의한 요구사항을 사용자 관점에서 점검하여 **필요한 제품을 만들었는지** 살피는 설명이다.

④ X: 검증은 단순히 개발 계획서를 점검하는 활동으로 정의되지 않는다. 각 단계의 산출물이 이전 단계에서 정한 요구사항이나 명세를 따르는지, 즉 **제품을 명세대로 올바르게 만들고 있는지** 점검하는 활동이다.

따라서 사용자 관점에서 합의한 요구사항을 점검한다고 한 ③이 가장 올바르다.

NoteBase

문 35. 다음은 소프트웨어 개발에서 사용자 요구사항 개발 프로세스들이다. 순서대로 가장 올바르게 나열한 것은?

가. 요구사항 분석
나. 요구사항 검증
다. 요구사항 도출
라. 요구사항 명세

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

정답 1번

간단한 해설

사용자의 요구를 먼저 수집한 뒤 분석·조정하고, 이를 **명세로 구체화한 다음 적절성을 검증**한다. 검증은 구현된 프로그램의 실행 시험만을 뜻하는 것이 아니라 요구사항 자체를 점검하는 활동이다.

올바른 순서

다 → 가 → 라 → 나

다. 요구사항 도출: 인터뷰·관찰·설문 등으로 이해관계자가 원하는 기능, 업무 목적과 제약조건을 수집한다. 이후 분석하려면 먼저 **사용자의 요구를 파악**해야 한다.

가. 요구사항 분석: 도출된 요구를 분류하고 우선순위·실현 가능성을 검토하며, 모순·중복·누락을 찾아 이해관계자와 조정한다. 단순한 요구 목록을 검토하여 무엇을 충족해야 하는지 구체화한 결과가 명세의 바탕이 된다.

라. 요구사항 명세: 분석한 요구사항을 문서나 모델로 명확하게 표현한다. 개발자와 사용자가 같은 기준을 공유하고 시험으로 충족 여부를 판단할 수 있도록 **명확하고 검증 가능한 형태**로 기록한다.

나. 요구사항 검증: 명세된 요구사항이 실제 사용자 의도를 반영하는지, 완전성·일관성·실현 가능성·검증 가능성을 갖추었는지 점검한다. 검토할 명세가 먼저 있어야 하며, 발견된 문제는 앞 단계로 돌려보내 수정한다. 따라서 실제 개발에서는 반복과 피드백이 일어날 수 있다.

NoteBase

문 36. 역공학 관련 설명으로 가장 올바르지 않은 것은?

① 코드만 남아있고 관련 문서가 존재하지 않을 때 적용하는 기술이다.

② 역공학 도구가 필요하다.

③ 추출한 문서를 기반으로 시스템을 개선, 변경하는 작업이다.

④ 소스코드를 분석하여 이해하기 쉬운 다이어그램으로 변환한다.

정답 3번

간단한 해설

역공학은 기존 구현을 분석하여 **설계와 구조를 복원하는 이해 중심의 작업**이다. 복원한 정보를 이용해 시스템을 개선·변경하는 재공학과 구별된다.

선지별 해설

① O: 역공학은 기존 구현을 분석하여 설계·구조·기능과 같은 상위 수준의 정보를 복원하는 작업이다. 관련 설계 문서가 없고 코드만 남은 시스템을 이해하거나 문서를 다시 만들 때 활용할 수 있다. 문서가 전혀 없는 경우에만 적용되는 것은 아니다.

② O: 실제 소프트웨어의 구조를 효율적으로 복원하려면 코드 분석, 호출 관계 추출, 자료 흐름 분석, 다이어그램 생성 등을 지원하는 역공학 도구를 활용한다. 여기서 도구의 필요성은 복잡한 시스템을 체계적으로 분석하기 위한 지원을 뜻하며, 사람이 코드를 읽고 수작업으로 분석하는 것 자체가 불가능하다는 뜻은 아니다.

③ X: 역공학 자체의 목적은 **기존 시스템을 분석하여 이해와 표현을 복원**하는 것이지, 그 시스템을 개선·변경하는 것이 아니다. 추출한 설계 정보 등을 바탕으로 시스템을 개선하여 다시 구축하는 작업은 재공학에 해당하며, 복원된 설계에서 구현으로 나아가는 과정은 순공학의 방향이다.

④ O: 소스코드에서 모듈·클래스·호출 관계 등의 구조를 추출해 다이어그램이나 설계 표현으로 나타낼 수 있다. 이는 코드의 세부 사항을 더 이해하기 쉬운 상위 추상 수준으로 옮기는 역공학의 대표적 작업이다.

NoteBase

문 37. 코드 스멜의 설명으로 가장 올바르지 않은 것은?

① 명확히 주석문 처리가 되어 있는 프로그램

② 중복된 로직을 가진 프로그램

③ 복잡한 조건문이 포함된 프로그램

④ 읽기 어려운 프로그램

정답 1번

간단한 해설

코드 스멜은 중복·과도한 복잡성·낮은 가독성처럼 개선이 필요할 수 있음을 알리는 징후이다. **명확하고 유용한 주석 자체**는 코드 스멜이 아니다.

선지별 해설

코드 스멜은 곧바로 실행 오류를 뜻하지는 않지만 유지보수상의 문제를 의심하게 하는 특징이다.

① X: 코드의 의도나 필요한 배경을 명확히 전달하는 주석은 이해를 돕는다. **주석이 명확하다는 사실 자체는 코드 스멜이 아니다**. 다만 난해한 코드를 설명하는 주석에만 의존하거나 오래된 주석이 실제 코드와 어긋나는 경우에는 개선이 필요할 수 있다.

② O: 같은 로직이 여러 곳에 중복되면 변경할 때 여러 위치를 함께 고쳐야 하고 일부 수정이 누락될 수 있다. **중복 로직으로 인한 변경의 불일치**는 공통 기능 추출 등의 리팩터링을 검토하게 하는 신호이다.

③ O: 복잡하게 중첩된 조건문은 실행 경로와 예외 조건을 파악하기 어렵게 한다. **판단 흐름의 과도한 복잡성**은 조건 분리나 메서드 추출 등으로 구조를 단순화할 필요가 있다는 신호이다.

④ O: 의미가 불명확한 이름이나 뒤엉킨 구조로 읽기 어려운 코드는 이해·수정·검토 비용을 높인다. **낮은 가독성**은 설계나 코드 구조를 개선할 필요가 있음을 보여 주는 코드 스멜이다.

따라서 명확한 주석 처리를 코드 스멜로 제시한 ①이 가장 올바르지 않다.

NoteBase

문 38. 소프트웨어 아키텍처 스타일에 대한 설명으로 가장 올바르게 짝지어진 것은?

① 데이터 저장소(Repository): 사용자 인터페이스와 데이터 처리 로직을 독립적으로 분리하여 변경 영향을 줄인다.

② 파이프 필터(Pipe and filter): 제어 대상이 되는 컴포넌트는 컨트롤러에 이벤트를 보내고 처리 명령을 받는다.

③ 클라이언트-서버: 클라이언트는 서버에 네트워크를 통한 서비스를 요청하고, 서버는 결과를 전송한다.

④ 계층형: 디스패처에 의해 트랜잭션을 하나씩 읽어 내용에 따라서 필요 처리로 연쇄된다.

정답 3번

간단한 해설

클라이언트-서버 스타일은 **서비스를 요청하는 클라이언트**와 **요청을 처리하여 응답하는 서버**로 역할을 나눈다. 네트워크를 통한 요청과 결과 전송이라는 설명이 이 구조에 대응한다.

선지별 해설

① X: 저장소 스타일은 여러 컴포넌트가 **공유 데이터 저장소를 중심으로 자료를 읽고 쓰는 구조**이다. 사용자 인터페이스와 데이터·처리 책임을 분리하여 변경 영향을 줄인다는 설명은 MVC와 같은 관심사 분리 방식에 가깝다. 저장소의 핵심은 화면과 로직의 분리가 아니라 공유 자료를 통한 컴포넌트의 협력이다.

② X: 파이프 필터 스타일은 자료를 변환하는 필터들을 파이프로 연결하여 **한 단계의 출력을 다음 단계의 입력**으로 전달하는 구조이다. 컴포넌트가 중앙 컨트롤러에 이벤트를 보내고 명령을 받는다는 설명은 중앙 집중형 제어 방식의 성격이며, 자료가 필터 사이를 흐르는 파이프 필터의 정의가 아니다.

③ O: 클라이언트-서버 스타일에서는 서비스를 요청하는 클라이언트와 서비스를 제공하는 서버의 역할을 나눈다. 클라이언트가 네트워크를 통해 요청을 보내면 서버는 요청을 처리하고 결과를 응답하므로, 제시된 설명과 일치한다.

④ X: 계층형 스타일은 시스템을 여러 계층으로 나누고, 일반적으로 상위 계층이 하위 계층의 서비스를 이용하도록 구성한다. 핵심은 **계층별 책임과 서비스 의존 관계**이다. 디스패처가 트랜잭션의 내용을 보고 적절한 처리 경로로 보내는 것은 트랜잭션 중심 처리 방식에 관한 설명이며, 계층형의 정의가 아니다.

NoteBase

문 39. 다음 중 디자인 패턴(Design pattern) 유형이 올바르게 분류되지 않은 것은?

① Command - Observer - Visitor - Iterator - Template method

② Bridge - Proxy - Facade - Adapter - Decorator

③ Builder - Factory method - Prototype - Singleton - Abstract factory

④ Flyweight - Composite - Memento - State - Mediator

정답 4번

간단한 해설

Flyweight와 Composite는 **구조 패턴**이고, Memento·State·Mediator는 **행위 패턴**이다. 서로 다른 유형을 같은 묶음에 넣은 부분을 찾아야 한다.

선지별 해설

GoF 디자인 패턴은 목적에 따라 생성·구조·행위 패턴으로 분류한다. 생성은 객체를 만드는 방법, 구조는 객체와 클래스의 구성, 행위는 책임 분배와 상호작용에 초점을 둔다.

① O: 모두 행위 패턴이다. Command는 요청을 객체로 캡슐화하고, Observer는 상태 변화의 통지를 구성하며, Visitor는 객체 구조에 적용할 연산을 분리한다. Iterator는 내부 표현을 노출하지 않고 요소를 순회하고, Template method는 알고리즘의 골격을 정한 뒤 일부 단계를 하위 클래스에 맡긴다.

② O: 모두 구조 패턴이다. Bridge는 추상화와 구현을 분리하고, Proxy는 다른 객체에 대한 접근을 대리·통제한다. Facade는 서브시스템의 간결한 통합 인터페이스를 제공하고, Adapter는 기존 인터페이스를 필요한 형태로 변환하며, Decorator는 객체를 감싸 책임을 추가한다.

③ O: 모두 생성 패턴이다. Builder는 복잡한 객체의 구성 과정을 분리하고, Factory method는 생성할 구체 클래스를 하위 클래스에서 정하도록 한다. Prototype은 원형 객체를 복제하고, Singleton은 해당 클래스의 인스턴스를 하나로 제한하며, Abstract factory는 관련 객체들의 집합을 구체 클래스에 의존하지 않고 생성하도록 한다.

④ X: **Flyweight·Composite는 구조 패턴, Memento·State·Mediator는 행위 패턴**이므로 한 유형으로 묶을 수 없다. Flyweight는 공유 가능한 상태를 재사용해 자원을 절약하고, Composite는 부분과 전체를 트리 구조로 다룬다. Memento는 내부 상태를 보관·복원하고, State는 상태별 행동을 객체로 분리하며, Mediator는 객체 사이의 상호작용을 중재한다.

NoteBase

문 40. 다음 내용에 부합하는 것으로 가장 올바른 것은?

- 전체 시스템을 여러 작은 단위 서비스로 분할한다.
- 단위 서비스는 독립적으로 실행 가능하다.
- 각 서비스의 유지보수와 배포는 개별적으로 처리 가능하다.
- 시스템 전체를 단일 서비스로 구성하는 모놀리식(Monolithic)과 상반된 방법이다.

① MSA(Microservice Architecture)

② ROA(Resource Oriented Architecture)

③ MDA(Model Driven Architecture)

④ Data-flow architecture

정답 1번

간단한 해설

시스템을 작은 서비스들로 나누고 각 서비스를 **독립적으로 실행·배포**하는 것은 MSA의 특징이다. 서비스별로 변경하고 유지보수할 수 있다는 점이 모놀리식과 구별된다.

공통 해설

MSA(Microservice Architecture)는 하나의 시스템을 작은 서비스들의 집합으로 구성하는 아키텍처이다. 각 서비스는 독립적으로 실행되며, 다른 서비스와 전체 시스템을 한꺼번에 다시 배포하지 않고도 **서비스별로 유지보수하고 배포**할 수 있다. 따라서 지문의 서비스 분할, 독립 실행, 개별 유지보수·배포라는 단서는 모두 MSA를 가리킨다.

예를 들어 주문 서비스와 결제 서비스로 나눈 시스템에서는 결제 기능을 수정한 뒤 결제 서비스만 새 버전으로 배포할 수 있다. 반면 모놀리식은 여러 기능을 하나의 애플리케이션으로 묶어 배포하므로, 내부 기능을 나누어 작성했더라도 배포 단위는 하나이다. 지문의 핵심은 단순히 기능을 작게 나누는 데 그치지 않고 **실행·배포 단위까지 분리**한다는 데 있다.

NoteBase