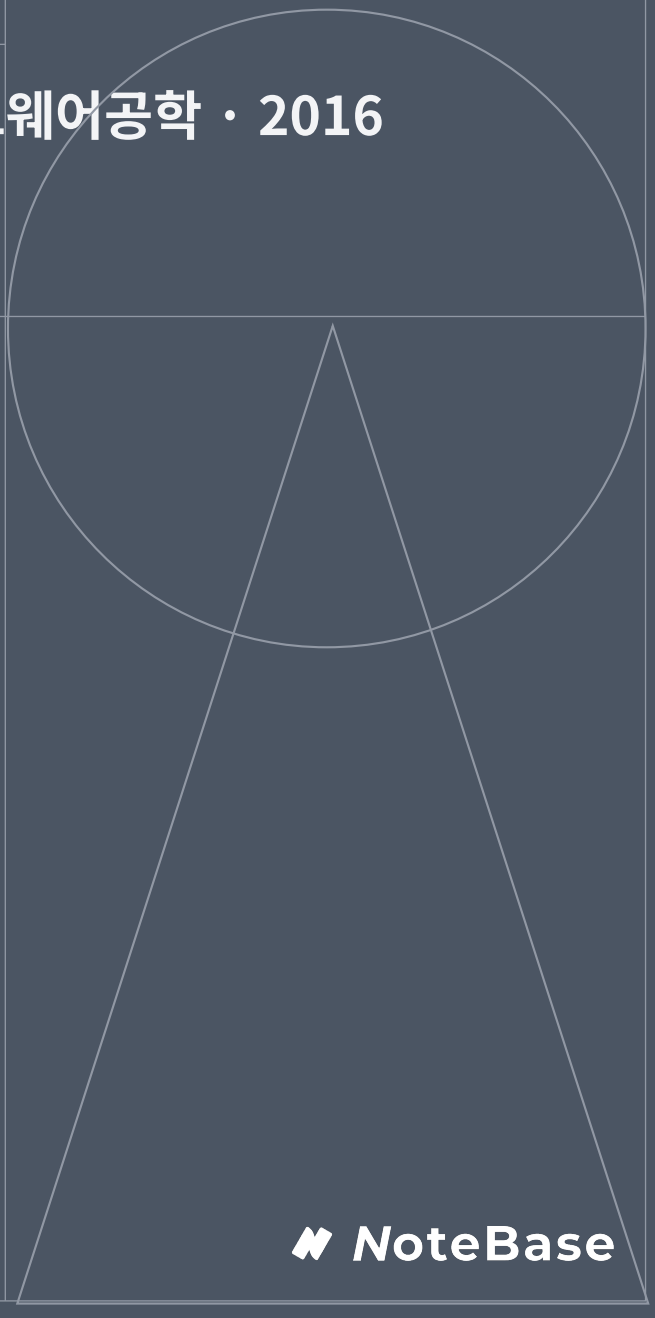
# 기출OX 해설집

공무원/국가직/7급 · 소프트웨어공학 · 2016



NoteBase

# 기출OX 해설집

## 공무원/국가직/7급 · 소프트웨어공학 · 2016

**공무원/국가직/7급 · 소프트웨어공학 · 2016**

**문 1. 객체지향 시스템 테스트의 설명으로 옳지 않은 것은?**

① 클래스에 대한 테스트 시 필요로 하는 객체 상태를 고려하지 않는다.

② 클래스 클러스터 차원에서 상속, 캡슐화, 동적 바인딩, 다형성 등의 이유로 관계가 맺어진 클래스들의 집합은 한 덩어리로 간주하여 연결 관계를 확인해야 한다.

③ 서브클래스를 테스트할 때 부모클래스의 테스트 케이스를 재사용할 수 있다.

④ 클래스 차원에서 메서드 상호 간, 메서드와 데이터 멤버 간의 연결 관계의 정확성을 확인해야 한다.

**정답 1번**

**출제 포인트**

이 문제는 객체지향 시스템 테스트에서 클래스 테스트와 클래스 클러스터 테스트의 관점을 구분하는 문제이다.

**핵심 개념**

클래스 테스트는 객체의 메서드와 데이터 멤버, 그리고 객체 상태 변화까지 함께 검증하는 테스트이다. / 결정 차이축: 개별 클래스 내부의 상태와 행위의 일관성을 보는가 / 키워드: 객체 상태, 메서드-데이터 결합
클래스 클러스터 테스트는 상속, 다형성, 메시지 전달로 연결된 여러 클래스를 하나의 집합으로 보고 상호작용을 검증하는 테스트이다. / 결정 차이축: 클래스 간 연결과 협력을 보는가 / 키워드: 클래스 집합, 연결 관계

**선지별 해설**

① X: 클래스 테스트에서는 객체 상태를 반드시 고려해야 한다. 객체지향 클래스는 같은 메서드라도 객체의 현재 상태에 따라 결과가 달라질 수 있기 때문이다. 틀린 표현은 "객체 상태를 고려하지 않는다"이다. 올바르게는 "객체 상태를 고려하여 상태별 메서드 동작을 테스트한다"이다.
② O: 상속, 캡슐화, 동적 바인딩, 다형성으로 연결된 클래스들은 독립적으로만 보면 결함을 놓치기 쉽다. 따라서 관련 클래스 집합을 클래스 클러스터로 보고 연결 관계와 상호작용을 확인해야 한다는 설명은 타당하다.
③ O: 서브클래스는 부모클래스의 속성과 연산을 상속하므로, 부모클래스에서 작성한 테스트 케이스 중 상속된 기능 검증 항목은 재사용할 수 있다. 이후 서브클래스에서 추가되거나 재정의된 부분을 별도로 보완하면 된다.
④ O: 클래스 차원의 테스트는 메서드 상호 간 호출 관계와 메서드와 데이터 멤버 간 결합의 정확성을 확인하는 과정이다. 따라서 클래스 내부 연결 관계의 정확성을 확인해야 한다는 설명은 맞다.

NoteBase

### 오답 포인트

객체지향 테스트를 일반적인 함수 테스트처럼 생각하면 ①처럼 상태를 배제해도 된다고 오해하기 쉽다. 이 문항은 객체가 상태와 행위를 함께 가진다는 점을 놓치면 틀리도록 만든 문제이다.

### 다시 나오면 이렇게 판단

객체지향 테스트 문항에서는 먼저 검증 단위가 개별 클래스인지, 클래스 집합인지 구분한다. 개별 클래스라면 객체 상태, 메서드, 데이터 멤버의 결합을 본다. 여러 클래스가 함께 나오면 상속, 다형성, 연결 관계를 보는 통합 관점인지 판단한다.

### 암기 포인트

클래스 테스트는 상태까지, 클래스 클러스터 테스트는 관계까지 본다.

NoteBase

**문 2. 소프트웨어 재사용에 대한 설명으로 옳지 않은 것은?**

① 사용자 인터페이스와 같은 표준은 표준화된 재사용 가능한 컴포넌트의 집합으로 구현될 수 있다.

② 응용시스템, 컴포넌트, 객체, 함수는 재사용이 가능하다.

③ 소프트웨어 재사용은 기존의 소프트웨어를 활용하므로 수정하거나 이해하기 위한 추가의 노력이 들지 않는다.

④ 동일한 작업을 수차례 반복하는 대신 전문가들의 지식이 요약된 재사용 가능한 소프트웨어를 개발, 활용할 수 있다.

**정답 3번**

**출제 포인트**

이 문제는 소프트웨어 재사용의 대상과 장점, 그리고 재사용에도 이해·적응·수정 비용이 든다는 한계를 구분할 수 있는지를 묻는 문제이다.

**핵심 개념**

소프트웨어 재사용은 기존의 응용시스템, 컴포넌트, 객체, 함수 같은 소프트웨어 자산을 새 개발에 다시 활용하는 방식이다. 키워드는 자산 활용이다.
재사용 비용은 기존 소프트웨어라도 탐색, 이해, 적응, 통합, 수정에 시간이 들 수 있다는 점이다. 키워드는 이해·수정 비용이다.

NoteBase

### 선지별 해설

① O
사용자 인터페이스와 같은 표준은 재사용 가능한 컴포넌트 집합으로 구현할 수 있다. 공통 UI 위젯이나 화면 구성 요소를 표준화해 여러 시스템에서 반복 사용한다는 설명이므로 옳다.

② O
소프트웨어 재사용은 큰 단위의 응용시스템부터 컴포넌트, 객체, 함수 같은 작은 단위까지 가능하다. 재사용 단위의 범위를 묻는 표현으로 타당하다.

③ X
틀린 부분은 "수정하거나 이해하기 위한 추가의 노력이 들지 않는다"이다. 기존 소프트웨어를 재사용하더라도 내용을 파악하고 현재 환경에 맞게 적응·수정·통합하는 비용이 발생할 수 있다. 올바르게는 "기존 소프트웨어를 활용하더라도 이해와 수정에 추가 노력이 들 수 있다"이다.

④ O
재사용 가능한 소프트웨어는 반복 작업을 줄이고, 전문가의 지식이나 검증된 해결 방법을 자산으로 축적해 다시 활용하게 한다. 재사용의 대표적 장점을 설명하므로 옳다.

### 오답 포인트

③은 "기존 것을 쓰니 추가 비용이 없다"라는 식으로 재사용의 장점만 과장한 표현이 함정이다. 재사용은 개발 생산성을 높이지만, 기존 자산을 이해하고 현재 요구에 맞게 맞추는 비용까지 사라지는 것은 아니다.

### 다시 나오면 이렇게 판단

재사용 문제는 먼저 재사용 대상이 어떤 단위까지 포함되는지 본다. 그다음 장점 문장인지, 아니면 비용과 한계를 부정하는 과장 문장인지 구분하면 된다. "추가 노력 없음", "수정 불필요" 같은 절대 표현이 나오면 오답 가능성이 크다.

### 암기 포인트

재사용은 범위가 넓고 이점이 크지만, 이해·적응·수정 비용까지 0이 되는 것은 아니다.

NoteBase

**문 3. 형상관리에 대한 설명으로 옳은 것은?**

① 컴포넌트 기반의 소프트웨어 개발 프로젝트에서는 형상관리가 필요하지 않다.

② 베이스라인은 단위 테스트를 통과한 원시 코드의 집합을 의미한다.

③ ISO 9000에서는 형상관리에 대한 언급이 없으나 CMMI에서는 형상관리 프로세스를 요구한다.

④ 형상관리는 소프트웨어 품질 보증을 위한 주요한 요소이며, 주 임무는 변경의 통제이다.

**정답 4번**

**출제 포인트**

이 문제는 형상관리의 목적, 베이스라인의 의미, 품질보증과의 관계를 정확히 구별하는지를 묻는 문제이다.

**핵심 개념**

형상관리: 소프트웨어 산출물의 버전과 변경을 식별·통제하여 일관성과 무결성을 유지하는 활동이다. / 결정 차이축: 변경을 막는 것이 아니라 승인된 절차로 통제하는가 / 키워드: 버전관리, 변경통제, 기준선
베이스라인: 공식 검토와 승인을 거쳐 이후 변경 시 형상통제를 받는 기준 산출물이다. / 결정 차이축: 단순 작업본인지 공식 기준본인지 / 키워드: 승인, 기준선, 변경관리
품질보증: 개발 전 과정이 표준과 절차를 따르는지 보증하는 활동이며 형상관리는 이를 뒷받침하는 핵심 수단이다. / 결정 차이축: 품질활동 전반인지 그중 변경·버전 통제인지 / 키워드: 절차준수, 감사, 변경통제

NoteBase

**기출OX 해설집 · 공무원/국가직/7급 · 소프트웨어공학 · 2016**

**선지별 해설**

① X
컴포넌트 기반 개발도 재사용 컴포넌트의 버전, 의존관계, 교체 이력을 관리해야 하므로 형상관리가 필요하다.
틀린 부분은 "필요하지 않다"이다.
올바르게는 "컴포넌트 기반 개발에서도 형상관리가 필요하다"이다.

② X
베이스라인은 단위 테스트를 통과한 원시 코드 집합만을 뜻하지 않는다. 공식 승인되어 이후 변경 통제의 기준이 되는 산출물을 말한다.
틀린 부분은 "단위 테스트를 통과한 원시 코드의 집합"이다.
올바르게는 "공식 검토와 승인을 거쳐 변경 통제의 기준이 되는 산출물"이다.

③ X
CMMI가 형상관리를 요구하는 것은 맞지만, ISO 계열의 품질관리 체계도 형상관리 개념과 지침을 다룬다. 따라서 "ISO 9000에서는 형상관리에 대한 언급이 없다"가 틀렸다.
틀린 부분은 "ISO 9000에서는 형상관리에 대한 언급이 없다"이다.
올바르게는 "ISO 계열에서도 형상관리의 필요성과 지침을 다룬다"이다.

④ O
형상관리는 산출물의 동일성, 추적성, 무결성을 유지해 품질보증을 뒷받침하는 핵심 활동이며, 주요 업무에 변경 통제가 포함된다.

**오답 포인트**

②는 베이스라인을 "테스트가 끝난 코드 묶음" 정도로 축소해서 외우면 틀리기 쉽다. 베이스라인은 승인된 기준본이라는 점이 핵심이다.
③은 "CMMI는 형상관리, ISO는 품질관리"처럼 이분법으로 외우면 "ISO는 언급 없음"이라는 극단 표현에 속기 쉽다.

**다시 나오면 이렇게 판단**

형상관리 문제는 먼저 기준선이 무엇인지, 변경을 금지하는지 통제하는지, 버전·승인·추적성을 다루는지를 본다. 선지에 "필요하지 않다", "언급이 없다"처럼 절대 표현이 나오면 우선 의심하는 것이 안전하다.

**암기 포인트**

형상관리=식별-기준선-변경통제-상태기록-감사, 베이스라인=승인된 기준본이다.

NoteBase

**문 4. 다음에서 설명하는 CMMI의 성숙도 단계에 해당하는 것은?**

표준 프로세스를 면밀히 검토하여 보완하고 최신 기술들을 반영하여 지속적으로 프로세스를 개선하고 이 개선된 프로세스를 전 조직이 사용하도록 한다.

① 관리(managed) 단계
② 정의(defined) 단계
③ 정량적 관리(quantitatively managed) 단계
④ 최적화(optimizing) 단계

**정답 4번**

**출제 포인트**

이 문제는 CMMI 성숙도 단계 중 조직 표준 프로세스를 지속적으로 개선하는 단계가 무엇인지 구분하는 문제이다.

**핵심 개념**

관리 단계는 프로젝트 차원에서 계획·수행·추적·통제를 가능하게 한 단계이다.
정의 단계는 조직 차원의 표준 프로세스를 정립하여 각 프로젝트가 이를 기반으로 수행하는 단계이다.
정량적 관리 단계는 프로세스를 수치로 측정하고 통계적으로 관리하는 단계이다.
최적화 단계는 정량적 이해를 바탕으로 표준 프로세스를 지속적으로 개선하고 새로운 기술과 방법을 조직 전체에 반영하는 단계이다.

**선지별 해설**

① X: 관리 단계는 프로젝트 수행을 계획하고 추적하는 수준이 핵심이다. 틀린 부분은 '표준 프로세스를 면밀히 검토하여 보완하고 최신 기술들을 반영하여 지속적으로 개선'이라는 표현이다. 관리 단계가 아니라 프로젝트 단위의 관리 체계를 갖춘 수준으로 고쳐야 한다.
② X: 정의 단계는 조직의 표준 프로세스를 문서화하고 전 조직이 공유하게 하는 단계이다. 틀린 부분은 '지속적으로 프로세스를 개선하고 최신 기술을 반영'이라는 표현이다. 정의 단계는 표준화가 중심이며, 지속적 개선과 혁신은 최적화 단계로 고쳐야 한다.
③ X: 정량적 관리 단계는 프로세스 성과를 수치와 통계기법으로 관리하는 단계이다. 틀린 부분은 '최신 기술들을 반영하여 지속적으로 프로세스를 개선'이라는 표현이다. 정량적 관리 단계로 고치려면 측정 지표와 통계적 관리로 프로세스를 통제한다고 해야 한다.
④ O: 최적화 단계는 표준 프로세스를 계속 보완하고 기술 혁신과 피드백을 반영하여 조직 전체의 프로세스를 개선하는 단계이므로 제시된 설명과 일치한다.

NoteBase

### 오답 포인트

② 정의 단계와 ④ 최적화 단계를 혼동하기 쉽다. '전 조직이 사용'이라는 문구만 보면 정의 단계처럼 보이지만, 이 문항의 결정적 단서는 '최신 기술 반영'과 '지속적 개선'이며 이는 최적화 단계의 특징이다.

### 다시 나오면 이렇게 판단

CMMI 단계 문제는 먼저 핵심 동사를 본다. 표준화·공유이면 정의 단계, 수치·통계 관리이면 정량적 관리 단계, 지속적 개선·기술 혁신이면 최적화 단계로 판단된다.

### 암기 포인트

정의는 표준화, 정량적 관리는 수치관리, 최적화는 지속적 개선과 혁신이다.

NoteBase

**문 5. 인수 테스트(acceptance test)에 대한 설명으로 옳지 않은 것은?**

① 인수 테스트의 목적은 사용자에게 소프트웨어가 개발되어 사용될 준비가 되었다는 확신을 주기 위한 것이다.

② 알파 테스트는 선택된 사용자가 사용자 환경에서 수행하는 인수 테스트이다.

③ 사용자 스토리를 작성하면서 함께 작성한 테스트 시나리오에 따라 고객이 직접 테스트한다.

④ 개발자팀이 소프트웨어를 사용자에게 배포하여 사용자가 자신의 컴퓨터 환경 또는 실제 상황에서 수행하는 테스트이다.

**정답 2번**

**출제 포인트**

이 문제는 인수 테스트의 목적과 알파 테스트, 베타 테스트의 수행 환경 차이를 구별하는 문제이다.

**핵심 개념**

인수 테스트는 사용자가 소프트웨어가 요구사항을 충족하며 실제 사용 준비가 되었는지 확인하는 테스트이다.
알파 테스트는 개발자 측의 통제된 환경에서 사용자가 참여해 수행하는 인수 테스트이다.
베타 테스트는 소프트웨어를 사용자에게 배포한 뒤 실제 사용자 환경에서 수행하는 인수 테스트이다.

**선지별 해설**

① O: 인수 테스트는 최종 사용자가 제품을 받아들일 수 있는지 확인하는 단계이므로, 소프트웨어가 사용될 준비가 되었다는 확신을 주는 것이 목적이라는 설명은 옳다.
② X: 알파 테스트를 사용자 환경에서 수행한다고 한 부분이 틀리다. 알파 테스트는 개발자 측의 통제된 환경에서 수행하고, 실제 사용자 환경에서 수행하는 것은 베타 테스트이다.
③ O: 사용자 스토리와 함께 정리한 테스트 시나리오를 기준으로 고객이 직접 확인하는 방식은 인수 테스트의 전형적인 모습이므로 옳다.
④ O: 개발팀이 소프트웨어를 배포한 뒤 사용자가 자신의 컴퓨터 환경이나 실제 운영 상황에서 수행하는 테스트는 베타 테스트에 해당하므로 옳다.

**오답 포인트**

알파 테스트와 베타 테스트를 가르는 기준은 참여자보다 수행 환경이다. 사용자 참여라는 공통점만 보고 ②를 맞는 설명으로 오인하기 쉽다.

**다시 나오면 이렇게 판단**

인수 테스트 문제는 먼저 테스트가 어디서 수행되는지를 본다. 개발자 측의 통제된 환경이면 알파 테스트이고, 실제 사용자 환경이면 베타 테스트이다.

NoteBase

**암기 포인트**

알파 테스트는 안쪽의 통제 환경, 베타 테스트는 바깥의 실제 사용자 환경이다.

NoteBase

**문 6. McCabe의 순환복잡도(CC : Cyclometic Complexity)에 대한 설명으로 옳지 않은 것은?**

① 프로그램의 논리적인 복잡도를 정량적으로 측정하기 위한 척도(metric)이다.

② 프로그램의 분기 노드의 수보다 항상 큰 값을 갖는다.

③ 프로그램의 모든 경로를 최소 한 번 이상 실행하도록 테스트하기 위한 입력의 개수에 해당한다.

④ 플로우 그래프(flow graph)에서 간선(edge)의 수를 $E$, 노드의 수를 $N$이라고 하면 $CC = E - N + 2$이다.

**정답 3번**

**출제 포인트**

이 문제는 McCabe의 순환복잡도가 무엇을 세는 지표인지와 분기 노드 수, 독립 경로 수의 관계를 구별하는 문제이다.

**핵심 개념**

순환복잡도는 프로그램 제어 흐름의 논리적 복잡도를 정량화한 지표이다.
핵심은 '모든 경로'가 아니라 '선형 독립 경로 수'를 기준으로 본다는 점이다.
단일 입구·출구의 플로우 그래프에서는 $CC = E - N + 2$이며, 구조적으로는 분기(결정) 노드 수에 1을 더한 값으로 이해한다.

**선지별 해설**

① O
순환복잡도는 프로그램의 논리적 복잡도를 수치로 나타내는 대표적인 metric이므로 옳은 설명이다.

② O
분기 노드를 결정 노드로 보면 순환복잡도는 보통 '분기 노드 수 + 1'로 계산되므로 분기 노드 수보다 항상 크다.

③ X
틀린 표현은 '모든 경로'이다.
순환복잡도는 프로그램의 모든 가능한 경로 수를 뜻하지 않고, 최소한의 테스트를 위해 고려할 선형 독립 경로 수에 해당한다.
올바르게는 '프로그램의 선형 독립 경로를 최소 한 번 이상 실행하도록 테스트하기 위한 입력 수의 기준이 된다'라고 고쳐야 한다.

④ O
플로우 그래프에서 간선 수를 E, 노드 수를 N이라 하면 단일 연결 성분 기준으로 $CC = E - N + 2$가 성립하므로 옳다.

NoteBase

### 오답 포인트

수험생이 가장 많이 헷갈리는 지점은 ③의 '모든 경로' 표현이다. 순환복잡도는 가능한 경로를 전부 세는 값이 아니라 독립적으로 추가되는 경로의 수를 세는 값이므로, 루프가 있는 프로그램의 모든 실행 경로와 같은 개념으로 보면 틀리게 된다.

### 다시 나오면 이렇게 판단

순환복잡도 문제에서는 먼저 '무엇을 세는가'를 본다. 선지에 '모든 경로'가 나오면 의심하고, '독립 경로', '분기 수 + 1', '$E - N + 2$'와 연결되면 맞는 설명일 가능성이 높다.

### 암기 포인트

순환복잡도 = 모든 경로 수가 아니라 독립 경로 수이다.

NoteBase

**문 7. COCOMOII의 서브모델이 아닌 것은?**

① 초기 설계 모델(early design model)

② 임베디드 모델(embedded model)

③ 응용 결합 모델(application composition model)

④ 재사용 모델(reuse model)

**정답 2번**

**출제 포인트**

이 문제는 COCOMO II의 서브모델과 원래 COCOMO의 개발 모드를 구분하는지를 묻는 문제이다.

**핵심 개념**

COCOMO II의 대표 서브모델은 응용 결합 모델, 초기 설계 모델, 재사용 모델, 후기 아키텍처(post-architecture) 모델이다.
embedded는 COCOMO II의 서브모델이 아니라 원래 COCOMO에서 프로젝트 특성을 나누는 개발 모드의 하나이다.

**선지별 해설**

① O: 초기 설계 모델은 COCOMO II의 서브모델이다. 시스템 구조가 완전히 확정되기 전의 상위 설계 정보를 바탕으로 노력량을 추정한다.
② X: 임베디드 모델은 COCOMO II의 서브모델이 아니다. 틀린 부분은 'COCOMO II의 서브모델'이라는 전제이다. 올바르게는 원래 COCOMO의 개발 모드 중 하나이며, COCOMO II의 서브모델로는 후기 아키텍처 모델 등이 들어간다.
③ O: 응용 결합 모델은 COCOMO II의 서브모델이다. 객체점수나 화면 구성처럼 매우 초기 산출물을 이용해 개발 비용을 추정한다.
④ O: 재사용 모델은 COCOMO II의 서브모델이다. 기존 소프트웨어를 수정하거나 통합하여 재사용하는 경우의 작업량을 반영한다.

**오답 포인트**

embedded라는 용어가 널리 알려져 있어 COCOMO II에도 그대로 포함된다고 착각하기 쉽다. 구형 COCOMO의 개발 모드와 COCOMO II의 서브모델 목록을 섞으면 ②를 놓치기 쉽다.

**다시 나오면 이렇게 판단**

COCOMO II 문제는 먼저 서브모델 4개를 떠올리면 판단이 빨라진다. 응용 결합, 초기 설계, 재사용, 후기 아키텍처에 없으면 구형 COCOMO의 모드이거나 다른 분류일 가능성이 크다.

NoteBase

**암기 포인트**

COCOMO II는 응용 결합-초기 설계-재사용-후기 아키텍처이고, embedded는 구형 COCOMO의 개발 모드이다.

NoteBase

**문 8. 객체지향 설계 기법 중 “클라이언트는 자신이 사용하지 않는 메서드에 의존 관계를 맺으면 안 된다.”라는 설계 원칙으로 옳은 것은?**

① 단일 책임의 원칙

② 개방 폐쇄의 원칙

③ 의존 관계 역전의 원칙

④ 인터페이스 분리의 원칙

**정답 4번**

**출제 포인트**

이 문제는 SOLID 원칙 중 인터페이스 분리 원칙과 다른 객체지향 설계 원칙을 구별하는 문제이다.

**핵심 개념**

단일 책임의 원칙은 클래스가 하나의 책임만 가져 변경 이유가 하나여야 한다는 원칙이다.
개방 폐쇄의 원칙은 기존 코드를 크게 수정하지 않고 확장에는 열려 있고 변경에는 닫혀 있어야 한다는 원칙이다.
의존 관계 역전의 원칙은 고수준 모듈과 저수준 모듈이 모두 구체 클래스가 아니라 추상화에 의존해야 한다는 원칙이다.
인터페이스 분리의 원칙은 클라이언트가 자신이 사용하지 않는 메서드에 의존하지 않도록 작은 인터페이스로 분리해야 한다는 원칙이다.

NoteBase

### 선지별 해설

① X
단일 책임의 원칙은 책임의 수와 변경 이유를 줄이는 원칙이지, 사용하지 않는 메서드 의존을 막는 원칙은 아니다.
틀린 표현은 "사용하지 않는 메서드에 의존하지 않는다"를 단일 책임의 원칙으로 본 부분이다. 올바르게는 인터페이스 분리의 원칙이다.
② X
개방 폐쇄의 원칙은 확장 가능성과 변경 최소화를 말한다. 클라이언트가 불필요한 메서드에 묶이지 않게 하는 내용과는 다르다.
틀린 표현은 "사용하지 않는 메서드에 의존하지 않는다"를 개방 폐쇄의 원칙으로 본 부분이다. 올바르게는 확장에는 열려 있고 변경에는 닫혀 있어야 한다는 원칙이 개방 폐쇄의 원칙이다.
③ X
의존 관계 역전의 원칙은 구체 구현이 아니라 추상화에 의존하게 만드는 원칙이다. 질문의 문장은 인터페이스의 크기와 클라이언트 의존성을 다루므로 의존 관계 역전의 원칙이 아니다.
틀린 표현은 "사용하지 않는 메서드에 의존하지 않는다"를 의존 관계 역전의 원칙으로 본 부분이다. 올바르게는 고수준 모듈과 저수준 모듈 모두 추상화에 의존해야 한다는 내용이 의존 관계 역전의 원칙이다.
④ O
인터페이스 분리의 원칙은 클라이언트별로 필요한 기능만 제공하도록 인터페이스를 잘게 나누어, 쓰지 않는 메서드에 대한 의존을 제거하는 원칙이다. 제시문과 정확히 일치한다.

### 오답 포인트

SOLID 원칙은 모두 설계 품질을 높인다는 공통점이 있어 이름만 보고 혼동하기 쉽다. 특히 ①의 단일 책임과 ④의 인터페이스 분리는 모두 "분리"의 느낌이 있어 헷갈리지만, 하나는 책임의 분리이고 다른 하나는 인터페이스 의존의 분리라는 점이 다르다.

### 다시 나오면 이렇게 판단

선지에 "사용하지 않는 메서드", "불필요한 의존", "클라이언트별 인터페이스" 같은 표현이 나오면 인터페이스 분리 원칙을 먼저 떠올리면 된다. 반대로 "하나의 책임", "확장과 변경", "추상화 의존"이 나오면 각각 단일 책임, 개방 폐쇄, 의존 관계 역전 원칙으로 구분된다.

### 암기 포인트

ISP는 안 쓰는 메서드 의존 금지, DIP는 추상화 의존, SRP는 책임 하나, OCP는 확장 개방·변경 폐쇄이다.

NoteBase

**문 9. 소프트웨어 품질에 대한 설명으로 옳지 않은 것은?**

① ISO/IEC 12119 - 패키지 소프트웨어의 일반적인 제품 품질 요구사항 및 테스트를 위한 국제 표준 규격이다.
② 매트릭(metric) - 품질 기준별로 측정방법과 스케일 등을 정의하여 정확히 품질 기준을 측정할 수 있게 한다.
③ 인스펙션(inspection) - 체크리스트를 가지고 본인이 개발한 코드와 산출물 등을 검토하는 것이다.
④ 품질 기준(criteria) - 개발자 측면의 내부 관점을 나타내는 것이다.

**정답 3번**

**출제 포인트**

이 문제는 소프트웨어 품질 관련 용어인 ISO/IEC 12119, 품질 기준, 메트릭, 인스펙션의 의미를 구분하는 문제이다.

**핵심 개념**

ISO/IEC 12119는 패키지 소프트웨어의 품질 요구사항과 시험을 다루는 국제 표준이다.
품질 기준(criteria)은 품질을 판단하기 위한 내부적 세부 기준으로, 개발자 관점에서 품질 특성을 구체화한 것이다.
메트릭(metric)은 각 품질 기준을 실제로 측정할 수 있게 만드는 측정 방법이나 척도이다.
인스펙션(inspection)은 실행 없이 문서, 설계, 코드, 산출물을 체크리스트에 따라 동료들이 정적으로 검토하는 공식적 리뷰 기법이다.

**선지별 해설**

① O
ISO/IEC 12119는 패키지 소프트웨어의 일반적 제품 품질 요구사항과 테스트를 위한 국제 표준이므로 설명이 맞다.

② O
메트릭은 품질 기준별로 측정 방법과 스케일을 정해 품질을 정량적으로 평가할 수 있게 하므로 설명이 맞다.

③ X
틀린 부분은 "본인이 개발한"이다.
인스펙션은 작성자 혼자 점검하는 자기검토가 아니라, 체크리스트를 바탕으로 동료 검토자들이 코드와 산출물을 정적으로 검토하여 결함을 찾는 공식 리뷰이다.

④ O
품질 기준(criteria)은 품질 요인을 개발자 관점의 내부적 속성으로 세분화한 판단 기준이므로 설명이 맞다.

NoteBase

**오답 포인트**

인스펙션은 체크리스트를 사용한다는 점만 보고 맞다고 판단하기 쉽다. 그러나 이 문항의 함정은 "본인이 개발한"이라는 표현에 있으며, 자기검토와 인스펙션을 구분해야 한다.

**다시 나오면 이렇게 판단**

소프트웨어 품질 용어 문제에서는 먼저 용어의 역할을 나눠 본다. 표준은 무엇을 규정하는지, 기준은 무엇을 볼 것인지, 메트릭은 어떻게 잴 것인지, 인스펙션은 누가 어떤 방식으로 검토하는지로 구분하면 판단이 쉬워진다.

**암기 포인트**

기준은 볼 항목, 메트릭은 재는 척도, 인스펙션은 동료의 정적 검토이다.

NoteBase

**문 10. 요구사항 품질 속성에 대한 설명으로 옳지 않은 것은?**

① 기능성(functionality) - 요구분석명세서가 '어떻게' 보다 '무엇을'에 관점을 두고 기술되어야 함을 의미한다.

② 명확성(unambiguity) - 요구사항이 모든 이해관계자들에 의해 한 가지 의미로 해석되어야 한다.

③ 완전성(completeness) - 요구사항을 바로 구현할 수 있도록 자료구조나 알고리즘이 명시되어야 한다.

④ 일관성(consistency) - 요구사항들이 서로 모순되지 않아야 한다.

**정답 3번**

**출제 포인트**

이 문제는 요구사항 명세서의 품질 속성인 기능성, 명확성, 완전성, 일관성의 의미를 정확히 구분하는지를 묻는 문제이다.

**핵심 개념**

기능성은 요구사항이 시스템이 무엇을 해야 하는지를 제대로 담고 있는 성질이다.
명확성은 모든 이해관계자가 요구사항을 하나의 의미로 동일하게 해석할 수 있는 성질이다.
완전성은 필요한 기능, 제약, 예외 상황이 빠짐없이 기술되어 있는 성질이지 구현 알고리즘이나 자료구조를 적는 성질이 아니다.
일관성은 요구사항들 사이에 서로 충돌하거나 모순되는 내용이 없는 성질이다.

**선지별 해설**

① O - 기능성은 요구분석명세서가 구현 방법보다 시스템이 제공해야 할 기능과 목적, 즉 '무엇을' 중심으로 기술되어야 함을 뜻하므로 옳다.
② O - 명확성은 같은 요구사항을 사용자, 분석가, 개발자 등이 읽어도 한 가지 의미로 해석되어야 한다는 뜻이므로 옳다.
③ X - 틀린 부분은 '바로 구현할 수 있도록 자료구조나 알고리즘이 명시되어야 한다'는 표현이다. 완전성은 요구사항이 빠짐없이 서술되어 있어야 한다는 뜻이지, 구현 수준의 자료구조나 알고리즘까지 적어야 한다는 뜻이 아니다. 올바르게는 '완전성은 필요한 기능, 제약조건, 입력과 출력, 예외 상황 등이 누락 없이 기술되어야 한다'이다.
④ O - 일관성은 개별 요구사항들이 서로 충돌하지 않아야 한다는 뜻이므로 옳다.

**오답 포인트**

③은 완전성과 상세설계 수준의 구현 명세를 혼동하게 만드는 함정이다. 요구사항 명세는 구현 방법인 '어떻게'보다 요구 기능인 '무엇을'을 다루므로, 알고리즘과 자료구조를 적어야 한다는 문장이 나오면 설계 단계와 섞였는지를 의심해야 한다.

NoteBase

### 다시 나오면 이렇게 판단

품질 속성 문제에서는 먼저 그 문장이 요구사항의 본질인 '무엇을' 설명하는지, 아니면 설계와 구현의 '어떻게'를 끌어들이는지 구분하면 된다. 해석이 하나로 모이면 명확성, 빠짐이 없으면 완전성, 충돌이 없으면 일관성으로 판단할 수 있다.

### 암기 포인트

요구사항 명세는 '무엇을', 설계 명세는 '어떻게'이다. 완전성은 누락 없음이지 알고리즘 명시가 아니다.

NoteBase

**문 11. 통합 프로세스 모델(UP: Unified Process)의 특징을 설명한 것으로 옳지 않은 것은?**

① 반복적(iterative)이고, 점진적(incremental)인 개발 방법이다.

② 유스케이스를 기반으로 한다.

③ 아키텍처 중심의 개발을 지향한다.

④ 위험 관리는 포함되어 있지 않다.

**정답 4번**

**출제 포인트**

통합 프로세스 모델(UP)의 핵심 특징인 반복적·점진적 개발, 유스케이스 기반, 아키텍처 중심, 위험 중심 개발을 정확히 구별하는 문제이다.

**핵심 개념**

UP는 소프트웨어를 한 번에 완성하는 방식이 아니라 반복적으로 개발하고 각 반복마다 기능을 점진적으로 확장하는 프로세스이다.
유스케이스 기반은 요구사항을 유스케이스 중심으로 정리해 분석·설계·테스트의 기준으로 삼는 특징이다.
아키텍처 중심은 초기에 시스템의 핵심 구조를 안정화해 이후 반복의 기반을 마련하는 특징이다.
위험 중심은 반복마다 중요한 기술적·관리적 위험을 먼저 식별하고 줄이는 특징이다.

**선지별 해설**

① O: UP는 반복적(iterative)이고 점진적(incremental)인 개발 방법이 맞다. 한 번의 폭포식 진행이 아니라 여러 반복을 거치며 제품을 점차 완성한다.
② O: UP는 유스케이스 기반이 맞다. 요구사항을 유스케이스로 표현하고 이를 개발 전 과정의 기준으로 활용한다.
③ O: UP는 아키텍처 중심 개발을 지향한다. 초기 반복에서 핵심 아키텍처를 확립해 이후 기능 확장을 안정적으로 진행한다.
④ X: 틀린 부분은 "위험 관리는 포함되어 있지 않다"이다. 실제로는 위험 관리가 UP의 핵심 특징에 포함되며, 중요한 위험을 초기에 다루는 위험 중심 접근을 취한다.

**오답 포인트**

④는 위험 관리를 별도 관리 활동 정도로만 떠올리면 속기 쉽다. 그러나 UP는 반복·점진, 유스케이스 기반, 아키텍처 중심과 함께 위험 중심 개발을 핵심 축으로 본다.

NoteBase

### 다시 나오면 이렇게 판단

UP 특징 문제는 반복적·점진적 개발인지, 유스케이스 기반인지, 아키텍처 중심인지, 위험 중심인지를 네 축으로 나누어 판단하면 된다. 이 네 축 중 하나를 부정하거나 다른 방법론의 특징으로 바꿔 놓은 선지가 오답이 된다.

### 암기 포인트

UP는 반복·점진, 유스케이스, 아키텍처, 위험 중심의 네 축으로 정리된다.

NoteBase

**문 12. 소프트웨어 테스트에 대한 설명으로 옳지 않은 것은?**

① 동료 검토(peer review)는 개발한 원시 코드나 여러 가지 산출물에 대한 검토를 의뢰하여 오류를 찾는 방법이다.

② 기본 경로 테스트(basic path test)는 원시 코드의 독립적인 경로가 최소한 한 번은 실행되는 테스트 케이스를 찾아 테스트를 수행한다.

③ 회귀 테스트(regression test)는 한 모듈의 수정이 다른 부분에 미치는 영향을 최소화하기 위해 필요하다.

④ 블랙 박스 테스트(black box test)는 입력 값에 대한 예상 출력 값을 정해 놓고 그대로 결과가 나오는지 원시 코드를 보며 확인한다.

**정답 4번**

**출제 포인트**

이 문제는 동료 검토, 기본 경로 테스트, 회귀 테스트, 블랙 박스 테스트의 정의와 적용 기준을 구별하는지를 묻는 문제이다.

**핵심 개념**

동료 검토: 프로그램을 실행하지 않고 원시 코드나 문서 같은 산출물을 사람이 검토하여 결함을 찾는 정적 테스트 기법이다. 키워드는 리뷰, 산출물 검토이다.
기본 경로 테스트: 프로그램 내부 제어 흐름의 독립 경로가 최소 한 번씩 실행되도록 테스트 케이스를 설계하는 화이트 박스 테스트 기법이다. 키워드는 독립 경로, 내부 로직이다.
회귀 테스트: 수정이나 기능 추가 뒤에 기존 기능이 예상과 다르게 깨지지 않았는지 다시 확인하는 테스트이다. 키워드는 변경 영향, 부작용 점검이다.
블랙 박스 테스트: 내부 코드 구조를 보지 않고 입력과 출력 명세만으로 기능을 검증하는 테스트이다. 키워드는 명세 기반, 입력-출력이다.

**선지별 해설**

① O. 동료 검토는 개발자가 만든 원시 코드, 설계서, 문서 등의 산출물을 동료가 검토하여 결함을 찾는 방식이므로 설명이 맞다. 실행 결과보다 산출물 자체의 오류를 찾는 정적 검토가 핵심이다.
② O. 기본 경로 테스트는 제어 흐름 그래프를 바탕으로 독립 경로들을 최소 한 번씩 실행하도록 테스트 케이스를 구성하는 기법이므로 설명이 맞다. 핵심 기준은 독립 경로의 커버리지이다.
③ O. 회귀 테스트는 한 부분을 수정한 뒤 다른 기능에 부작용이 생기지 않았는지 확인하는 테스트이므로 설명이 맞다. 수정의 영향을 점검하여 기존 동작이 깨지는 문제를 발견하는 데 목적이 있다.
④ X. 틀린 표현은 '원시 코드를 보며 확인한다'이다. 블랙 박스 테스트는 원시 코드를 보지 않고 입력 값과 예상 출력 값을 기준으로 외부 동작을 검증한다. 원시 코드를 보며 내부 경로를 확인하는 방식은 화이트 박스 테스트에 가깝다.

NoteBase

### 오답 포인트

④는 앞부분의 '입력 값에 대한 예상 출력 값'만 보면 블랙 박스 테스트처럼 보이지만, 뒤의 '원시 코드를 보며'가 결정적으로 틀린 표현이다. 블랙 박스와 화이트 박스는 입력·출력만 보는지, 내부 코드를 보는지로 구별된다.

### 다시 나오면 이렇게 판단

테스트 기법 문제는 먼저 정적 검토인지 실행 테스트인지 나누고, 실행 테스트라면 내부 구조를 보는지 여부를 확인하면 된다. '독립 경로'가 나오면 기본 경로 테스트, '수정 후 기존 기능 점검'이 나오면 회귀 테스트, '코드를 보지 않고 입출력만 확인'이면 블랙 박스 테스트이다.

### 암기 포인트

동료 검토는 산출물 리뷰, 기본 경로 테스트는 독립 경로, 회귀 테스트는 수정 후 재검증, 블랙 박스 테스트는 코드 비공개 상태의 입출력 검증이다.

NoteBase

**문 13. 요구분석명세서를 작성할 때 고려해야 할 사항으로 옳지 않은 것은?**

① 시스템의 기능을 구현하는 방법에 대하여 중점적으로 기술한다.

② 제안된 시스템에 영향을 주는 제약조건을 기술한다.

③ 시스템의 인수를 위한 테스트 기준을 제공한다.

④ 소프트웨어의 품질 기준에 대한 우선순위를 정한다.

**정답 1번**

**출제 포인트**

이 문제는 요구분석명세서가 무엇을 기술해야 하는지와 무엇을 기술하면 안 되는지를 구분하는 문제이다.

**핵심 개념**

요구분석명세서는 사용자가 원하는 기능과 제약조건을 명확히 적는 문서이다. 핵심은 시스템이 무엇을 해야 하는지를 기술하는 것이며, 어떻게 구현할지는 설계 단계의 내용이다.

**선지별 해설**

① X
요구분석명세서는 구현 방법보다 요구 기능과 조건을 기술하는 문서이다.
틀린 부분은 "구현하는 방법"이다.
올바르게는 "시스템이 제공해야 할 기능과 요구사항을 기술한다"이다.

② O
제안된 시스템에 영향을 주는 제약조건은 요구분석명세서에 포함된다. 성능, 운영환경, 법적 제약, 인터페이스 조건 같은 제한사항을 명시해야 한다.

③ O
요구사항은 이후 검증과 인수 테스트의 기준이 된다. 사용자가 요구한 기능이 충족되었는지 판단하려면 요구분석명세서가 테스트 기준 역할을 해야 한다.

④ O
요구사항 간 우선순위와 품질기준은 요구분석 과정에서 정해 둘 수 있다. 특히 성능, 신뢰성, 사용성 같은 품질 요구의 중요도를 정리하는 것은 타당하다.

NoteBase

### 오답 포인트

요구분석과 설계를 혼동하면 ①이 맞아 보이기 쉽다. 이 문항의 함정은 "무엇을 할 것인가"와 "어떻게 만들 것인가"를 뒤섞는 데 있다.

### 다시 나오면 이렇게 판단

선지에 구현방법, 내부구조, 알고리즘처럼 해법 중심 표현이 나오면 설계 내용인지 먼저 본다. 반대로 기능, 제약조건, 검증기준, 우선순위가 나오면 요구분석명세서의 내용으로 판단할 수 있다.

### 암기 포인트

요구분석은 무엇을, 설계는 어떻게를 다룬다.

NoteBase

**문 14. 소프트웨어 유지보수의 형태에 대한 설명으로 옳지 않은 것은?**

① 수정 유지보수(corrective maintenance)는 개발된 소프트웨어를 사용자가 인도받은 후 사용하면서 발견되는 오류를 잡는 것이다.

② 예방 유지보수(preventive maintenance)는 미리 예상되거나 예측되는 오류를 찾아 수정하는 것이다.

③ 적응 유지보수(adaptive maintenance)는 개발 과정에서 바로 잡지 못한 오류를 유지보수 단계에서 해결하는 것이다.

④ 완전 유지보수(perfective maintenance)는 결함으로 인해 요청된 변경뿐만 아니라 시스템의 일부 측면을 향상시키기 위한 변경을 포함하고 있다.

**정답 3번**

**출제 포인트**

이 문제는 소프트웨어 유지보수의 네 가지 유형을 목적에 따라 구분할 수 있는지를 묻는 문제이다.

**핵심 개념**

수정 유지보수는 운영 중 발견된 결함을 바로잡는 유지보수이다. 키워드는 사후 오류 수정이다.
예방 유지보수는 앞으로 발생할 가능성이 있는 결함을 미리 줄이기 위한 유지보수이다. 키워드는 잠재 결함 예방이다.
적응 유지보수는 하드웨어, 운영체제, 업무 환경, 제도 변화 등에 맞추어 소프트웨어를 바꾸는 유지보수이다. 키워드는 환경 변화 대응이다.
완전 유지보수는 성능, 사용성, 기능 등 시스템의 품질과 가치를 높이기 위한 유지보수이다. 키워드는 기능·품질 향상이다.

**선지별 해설**

① O: 수정 유지보수는 사용자가 인도받아 실제로 사용한 뒤 발견한 오류를 수정하는 활동이므로 옳다.
② O: 예방 유지보수는 미래에 발생할 수 있는 오류를 미리 찾아 구조를 정리하거나 취약 부분을 고쳐 두는 활동이므로 옳다.
③ X: 틀린 부분은 "적응 유지보수는 개발 과정에서 바로 잡지 못한 오류를 유지보수 단계에서 해결하는 것이다"라는 설명이다. 이는 적응 유지보수가 아니라 수정 유지보수에 대한 설명이다. 올바르게는 적응 유지보수는 운영 환경이나 외부 조건의 변화에 맞게 소프트웨어를 변경하는 것이다.
④ O: 완전 유지보수는 결함 수정 요청만 처리하는 것이 아니라 기능 개선, 성능 향상, 사용 편의성 개선처럼 시스템의 일부 측면을 더 낫게 만드는 변경도 포함하므로 옳다.

NoteBase

### 오답 포인트

적응 유지보수와 수정 유지보수를 모두 "문제가 생기면 고친다"로 뭉뚱그리면 ③에 속기 쉽다. ③의 핵심 오답 표현은 "개발 과정에서 바로 잡지 못한 오류"이며, 오류 수정은 수정 유지보수이고 적응 유지보수는 환경 변화 대응이라는 점이 구분 기준이다.

### 다시 나오면 이렇게 판단

유지보수 유형 문제는 먼저 변경 이유를 본다. 이미 드러난 결함이면 수정, 앞으로 생길 위험을 줄이면 예방, 외부 환경 변화에 맞추면 적응, 성능이나 기능을 더 좋게 만들면 완전으로 판단한다.

### 암기 포인트

수정은 이미 난 오류, 예방은 날 오류, 적응은 바뀐 환경, 완전은 더 나은 품질이다.

NoteBase

**문 15. 허가받지 않은 사용자가 데이터 접근을 통해 변경을 시도했을 때 보호할 수 있는지 정도를 나타내는 품질 특성에 해당하는 것은?**

① 무결성(integrity)
② 융통성(flexibility)
③ 안전성(safety)
④ 안정성(stability)

**정답 1번**

**출제 포인트**

이 문제는 소프트웨어 품질 특성 중 무결성, 융통성, 안전성, 안정성의 의미를 구별하는 문제이다.

**핵심 개념**

무결성: 허가되지 않은 접근이나 변경으로부터 데이터와 프로그램을 보호하는 특성이다. 키워드는 비인가 변경 방지이다.
융통성: 환경이나 요구 변화에 맞게 소프트웨어를 수정하거나 적응시키기 쉬운 특성이다. 키워드는 변화 대응이다.
안전성: 소프트웨어가 사람이나 재산, 환경에 피해를 주지 않도록 하는 특성이다. 키워드는 위해 방지이다.
안정성: 수정이나 장애 상황에서도 예상하지 못한 부작용 없이 정상 상태를 유지하는 특성이다. 키워드는 흔들리지 않음이다.

**선지별 해설**

① O - 무결성은 비인가 사용자가 데이터에 접근해 내용을 바꾸려 할 때 이를 막거나 보호하는 정도를 뜻하므로 발문과 정확히 일치한다.
② X - 틀린 부분은 "융통성"이다. 융통성은 변경 시도 자체를 막는 보안 성질이 아니라 변화하는 환경이나 요구에 맞게 수정하기 쉬운 성질이다. 올바르게는 "요구사항이나 운영환경 변화에 잘 적응하는 정도"이다.
③ X - 틀린 부분은 "안전성"이다. 안전성은 시스템 사용으로 인한 인명, 재산, 환경상의 피해를 막는 성질이지 비인가 데이터 변경 방지를 뜻하지 않는다. 올바르게는 "시스템이 위험이나 손해를 유발하지 않는 정도"이다.
④ X - 틀린 부분은 "안정성"이다. 안정성은 수정이나 오류 상황에서도 시스템이 쉽게 흔들리거나 오동작하지 않는 성질이다. 올바르게는 "변경 후에도 예기치 않은 부작용 없이 일관되게 동작하는 정도"이다.

**오답 포인트**

비인가 접근과 변경이라는 표현이 나오면 보안 관련 품질 특성을 먼저 떠올려야 한다. 융통성, 안전성, 안정성은 모두 좋은 성질이지만, 보안과 직접 연결되는 것은 무결성이라는 점에서 선지가 갈린다.

NoteBase

### 다시 나오면 이렇게 판단

발문에 허가되지 않은 사용자, 접근 통제, 데이터 변경 방지 같은 표현이 있으면 무결성을 우선 판단 기준으로 삼는다. 변화 대응이면 융통성, 피해 방지이면 안전성, 변경 후 흔들림 방지이면 안정성으로 구분된다.

### 암기 포인트

무결성은 비인가 변경 방지, 융통성은 변화 대응, 안전성은 위해 방지, 안정성은 변경 후에도 흔들리지 않음이다.

NoteBase

**문 16. 애자일(agile) 선언문에 대한 설명으로 옳지 않은 것은?**

① 프로세스와 도구보다 개인과 그들의 협업에 더 가치를 둔다.

② 포괄적인 문서화보다 제대로 작동하는 소프트웨어에 더 가치를 둔다.

③ 고객과의 협력보다는 계약 협상에 더 가치를 둔다.

④ 계획에 따르기보다는 변화에 대응하는 것에 더 가치를 둔다.

**정답 3번**

**출제 포인트**

이 문제는 애자일 선언문의 4가지 핵심 가치를 정확히 구분하는지 묻는 문제이다.

**핵심 개념**

애자일 선언문은 전통적 개발에서 중시하던 형식과 통제를 전부 버린다는 뜻이 아니라, 비교되는 두 가치가 충돌할 때 오른쪽보다 왼쪽을 더 우선한다는 뜻이다.
개인과 상호작용은 프로세스와 도구보다 우선한다.
작동하는 소프트웨어는 포괄적 문서화보다 우선한다.
고객과의 협력은 계약 협상보다 우선한다.
변화에 대한 대응은 계획 따르기보다 우선한다.

**선지별 해설**

① O: 애자일 선언문의 첫 번째 가치와 일치한다. 애자일은 절차와 도구 자체를 부정하는 것이 아니라, 사람 사이의 협업과 상호작용을 더 중시한다.
② O: 애자일 선언문의 두 번째 가치와 일치한다. 문서가 불필요하다는 뜻이 아니라, 문서보다 실제로 작동하는 소프트웨어가 더 중요한 산출물이라는 뜻이다.
③ X: 틀린 부분은 "고객과의 협력보다는 계약 협상에 더 가치를 둔다"는 표현이다. 실제로는 고객과의 협력을 계약 협상보다 더 가치 있게 본다가 맞다.
④ O: 애자일 선언문의 네 번째 가치와 일치한다. 초기 계획은 필요하지만, 변화가 생기면 계획 고수보다 변화 대응을 우선한다.

**오답 포인트**

애자일 선언문은 모두 "A보다 B"가 아니라 "B보다 A" 구조로 암기해야 한다. ③은 고객과 계약 협상의 우선순위를 뒤집어 놓은 선지라서, 문장을 대충 읽으면 선언문의 원문 방향을 반대로 기억한 수험생이 걸리기 쉽다.

NoteBase

### 다시 나오면 이렇게 판단

애자일 선언문 문제는 각 문장을 "무엇을 무엇보다 더 중시하는가"의 짝으로 확인하면 된다.
사람-소프트웨어-고객협력-변화대응이 앞에 오면 옳고, 프로세스-문서-계약협상-계획고수가 앞에 오면 방향이 뒤집힌 것으로 판단할 수 있다.

### 암기 포인트

애자일 4가치는 사람, 작동, 협력, 대응을 프로세스, 문서, 계약, 계획보다 앞에 둔다.

NoteBase

**문 17. 다음에서 설명하는 디자인 패턴에 해당하는 것은?**

이미 만들어져 있는 클래스를 사용하고 싶지만 인터페이스가 원하는 방식과 일치하지 않을 때, 또는 관련성이 없거나 예측하지 못한 클래스들과 협동하는 재사용 가능한 클래스를 생성하기를 원할 때 사용한다.

① Bridge
② Adapter
③ Composite
④ Facade

**정답 2번**

**출제 포인트**

이 문제는 구조 패턴 중 기존 클래스의 인터페이스를 클라이언트가 원하는 형태로 바꾸는 Adapter 패턴의 정의를 정확히 구별하는지를 묻는 문제이다.

**핵심 개념**

Adapter: 기존 클래스의 인터페이스를 클라이언트가 기대하는 다른 인터페이스로 변환하는 패턴이다. 결정 차이축: 인터페이스 변환 여부이다. 키워드: 호환, 변환, 재사용이다.
Bridge: 추상화와 구현을 분리하여 각각 독립적으로 확장할 수 있게 하는 패턴이다. 결정 차이축: 계층 분리와 독립적 변화이다. 키워드: 추상화, 구현 분리이다.
Composite: 객체들을 부분-전체의 트리 구조로 구성하여 단일 객체와 복합 객체를 동일하게 다루는 패턴이다. 결정 차이축: 계층적 구성과 일관 처리이다. 키워드: 트리, 부분-전체이다.
Facade: 복잡한 서브시스템에 대한 단순한 통합 인터페이스를 제공하는 패턴이다. 결정 차이축: 사용 단순화이다. 키워드: 단순 창구, 서브시스템 감춤이다.

NoteBase

### 선지별 해설

① X: Bridge는 추상화와 구현을 분리하는 패턴이므로 기존 클래스의 인터페이스를 원하는 형태로 바꾸는 상황과 다르다. 틀린 표현은 인터페이스 불일치 문제를 Bridge로 해결한다고 본 점이다. 올바르게는 인터페이스 불일치와 기존 클래스 재사용 문제는 Adapter가 담당한다.
② O: Adapter는 이미 존재하는 클래스의 인터페이스가 필요한 형식과 맞지 않을 때 중간 변환 계층을 두어 호환되게 만든다. 제시문의 "이미 만들어져 있는 클래스", "인터페이스가 원하는 방식과 일치하지 않음", "재사용 가능한 클래스"가 모두 Adapter의 전형적 설명과 일치한다.
③ X: Composite는 개별 객체와 복합 객체를 동일하게 다루기 위한 트리 구조 패턴이므로 서로 다른 인터페이스를 맞추는 목적이 아니다. 틀린 표현은 재사용 클래스의 협동을 위해 인터페이스를 변환하는 상황을 Composite로 본 점이다. 올바르게는 Composite는 부분-전체 계층을 표현할 때 사용한다.
④ X: Facade는 복잡한 서브시스템 앞에 단순한 통합 창구를 제공하는 패턴이므로 기존 클래스의 인터페이스 자체를 클라이언트 요구 형식으로 바꾸지는 않는다. 틀린 표현은 인터페이스 변환 문제를 Facade로 본 점이다. 올바르게는 Facade는 사용을 단순화하는 패턴이고, 인터페이스 호환은 Adapter가 담당한다.

### 오답 포인트

Bridge와 Facade도 중간 계층을 둔다는 점 때문에 Adapter와 혼동하기 쉽다. 그러나 이 문항의 핵심 표현은 "인터페이스가 원하는 방식과 일치하지 않을 때"이다. 이 문구가 나오면 단순화나 분리가 아니라 인터페이스 변환을 먼저 떠올려야 한다.

### 다시 나오면 이렇게 판단

문장에서 기존 클래스를 그대로 쓰고 싶지만 형식이 맞지 않는다고 하면 Adapter로 판단한다. 추상화와 구현을 나누면 Bridge이고, 부분-전체 트리를 만들면 Composite이며, 복잡한 기능을 단순 창구로 감싸면 Facade이다.

### 암기 포인트

Adapter는 인터페이스 변환, Bridge는 추상화-구현 분리, Composite는 트리 구성, Facade는 단순 창구이다.

NoteBase

**문 18. 비기능 요구사항에 대한 설명으로 옳지 않은 것은?**

① 예산의 제약, 조직의 정책, 다른 소프트웨어와 하드웨어 시스템과의 상호 운영성, 안정성 규칙과 프라이버시 보호법과 같은 사용자의 필요에 의해 발생한다.

② 요구사항과 목표를 혼합한 문서를 사용하여 비기능적 요구사항을 표현한다.

③ 시스템에서 제공되는 서비스나 기능에 대한 제약이다.

④ 시스템이 제공해야 하는 서비스와 시스템이 특정 입력에 대해 어떻게 반응하는지, 시스템이 특정 상황에서 어떻게 동작해야 하는지에 관한 사항이다.

**정답 4번**

**출제 포인트**

이 문제는 비기능 요구사항과 기능 요구사항의 차이를 구별하는 문제이다.

**핵심 개념**

비기능 요구사항은 시스템이 무슨 기능을 하느냐보다 그 기능이 어떤 제약과 품질 조건 아래 제공되어야 하는지를 정한 요구사항이다. 키워드는 성능, 신뢰성, 보안, 상호운용성, 조직 정책, 법적 제약이다.
기능 요구사항은 시스템이 제공해야 하는 서비스, 입력에 대한 반응, 상황별 동작을 정한 요구사항이다. 키워드는 서비스, 반응, 동작이다.
비기능 요구사항은 성능 수치처럼 명확한 요구로도 쓰이지만, 초기에는 목표와 요구가 섞인 형태로 표현되기도 한다.

**선지별 해설**

① O 예산 제약, 조직 정책, 다른 시스템과의 상호운용성, 안전성 규칙, 프라이버시 보호와 같은 조건은 기능 자체가 아니라 시스템이 따라야 하는 제약과 품질 조건이므로 비기능 요구사항의 설명에 해당한다.
② O 비기능 요구사항은 항상 딱 잘라 수치 요구만으로 적히는 것이 아니라 목표와 요구사항이 섞인 문서 형태로 표현되기도 한다. 특히 품질 속성은 초기에는 목표 수준으로 서술되는 경우가 많다.
③ O 비기능 요구사항은 시스템이 제공하는 서비스나 기능 자체를 추가로 정의하는 것이 아니라, 그 서비스와 기능에 어떤 제약이 걸리는지를 정한다는 점에서 맞는 설명이다.
④ X 틀린 부분은 "시스템이 제공해야 하는 서비스", "특정 입력에 대해 어떻게 반응하는지", "특정 상황에서 어떻게 동작해야 하는지"이다. 이는 기능 요구사항의 정의이다. 비기능 요구사항으로 고치면 "시스템의 성능, 신뢰성, 보안, 상호운용성 등 서비스에 대한 제약과 품질 조건에 관한 사항"이 된다.

NoteBase

### 오답 포인트

④는 요구사항 정의 문장에서 매우 자주 나오는 기능 요구사항의 전형적 표현을 그대로 사용한다. '서비스', '입력에 대한 반응', '상황별 동작'이라는 표현이 나오면 기능 요구사항으로 판단해야 하는데, 이를 비기능 요구사항 설명으로 착각하기 쉽다.

### 다시 나오면 이렇게 판단

선지에 시스템이 무엇을 해야 하는지가 나오면 기능 요구사항으로, 시스템이 어떤 품질과 제약을 지켜야 하는지가 나오면 비기능 요구사항으로 판단한다. 서비스 내용인지, 서비스 조건인지의 축으로 나누면 빠르게 정리된다.

### 암기 포인트

기능 요구사항은 무엇을 하는가, 비기능 요구사항은 어떤 조건으로 해야 하는가이다.

NoteBase

**문 19. 객체지향 개념에 대한 설명으로 옳지 않은 것은?**

① 정보은닉(information hiding)은 모듈 안에 있는 자료구조와 메서드에 사용된 알고리즘을 외부에서 직접 변경할 수 없고, 공개 인터페이스로 정의된 메서드를 통해서만 접근할 수 있는 것을 의미한다.

② 전체 객체에 전속되어 독립된 객체로 존재할 수 없는 부분 객체도 있는데 이와 같은 관계를 특별히 집합 관계(aggregation relationship)라고 한다.

③ 리스코프 교체 원칙(LSP: Liskov Substitution Principle)은 부모 클래스와 자식 클래스 사이의 행위가 일관성이 있어야 한다는 의미이다.

④ 다형성(polymorphism)은 같은 오퍼레이션이 다른 클래스에서 다르게 동작하는 것을 말하며, 하나의 함수 이름이나 연산자가 여러 목적으로 사용될 수 있는 것을 의미한다.

**정답 2번**

**출제 포인트**

이 문제는 객체지향의 정보은닉, 집합 관계와 합성 관계, 리스코프 교체 원칙, 다형성의 차이를 정확히 구별하는지를 묻는 문제이다.

**핵심 개념**

정보은닉은 객체 내부의 자료구조와 구현 알고리즘을 외부에 직접 드러내지 않고 공개된 인터페이스를 통해서만 접근하게 하는 개념이다.
집합 관계는 전체와 부분의 관계이지만 부분 객체가 전체와 별개로 독립 존재할 수 있는 느슨한 포함 관계이다.
합성 관계는 부분 객체가 전체 객체에 강하게 종속되어 독립적으로 존재하기 어려운 강한 포함 관계이다.
리스코프 교체 원칙은 자식 클래스가 부모 클래스를 대체해도 프로그램의 기대 동작이 깨지지 않아야 한다는 원칙이다.
다형성은 같은 메시지나 연산이 객체의 실제 타입에 따라 다르게 동작하는 성질이다.

NoteBase

### 선지별 해설

① O
정보은닉은 모듈 내부 구현을 감추고 외부에서는 공개된 메서드로만 접근하게 하는 개념이므로 옳다. 자료구조와 알고리즘을 직접 건드리지 못하게 하여 변경 영향 범위를 줄인다.

② X
틀린 부분은 "전체 객체에 전속되어 독립된 객체로 존재할 수 없는 부분 객체"를 "집합 관계(aggregation relationship)"라고 한 표현이다.
올바르게는 부분 객체가 독립적으로 존재할 수 없을 정도로 전체에 종속되면 합성 관계(composition)이다. 집합 관계는 부분 객체가 전체와 분리되어도 존재할 수 있는 경우를 가리킨다.

③ O
리스코프 교체 원칙은 부모 타입이 사용되는 자리에 자식 타입을 넣어도 기대한 행위가 유지되어야 한다는 뜻이므로 옳다. 제시문처럼 부모 클래스와 자식 클래스 사이의 행위 일관성을 요구하는 설명과 맞는다.

④ O
다형성은 같은 오퍼레이션이 객체 타입에 따라 다르게 구현되어 동작하는 성질이므로 옳다. 하나의 함수 이름이나 연산자가 여러 형태로 해석되는 것도 다형성의 대표적인 모습이다.

### 오답 포인트

전체-부분 관계라는 공통점만 보고 aggregation과 composition을 같은 것으로 보면 ②에 속기 쉽다. 이 문항의 핵심 함정은 "독립된 객체로 존재할 수 없는"이라는 표현이며, 이 말이 나오면 약한 포함인 집합 관계가 아니라 강한 포함인 합성 관계를 떠올려야 한다.

### 다시 나오면 이렇게 판단

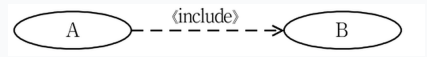
전체-부분 관계 문제는 먼저 부분 객체의 독립 생존 가능 여부를 본다. 분리되어도 존재할 수 있으면 aggregation이고, 전체에 종속되어 함께 생명주기를 가지면 composition이다. LSP와 다형성은 각각 대체 가능성의 원칙과 같은 메시지의 다른 동작으로 구별하면 된다.

### 암기 포인트

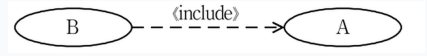
독립 생존 가능이면 aggregation, 불가능이면 composition이다. 대체 가능성은 LSP이고, 같은 메시지의 다른 동작은 polymorphism이다.

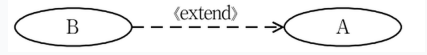
NoteBase

**문 20. 유스케이스 다이어그램에서 A 유스케이스를 수행하는 도중에 특정 조건을 만족하면 B 유스케이스를 수행한다. A 유스케이스와 B 유스케이스 간의 관계로 옳은 것은?**

① 


② 


③ 


④ 


**정답 4번**

### 출제 포인트

이 문제는 유스케이스 다이어그램에서 `include`와 `extend`의 의미 차이와 화살표 방향을 도식으로 구분하는 문제이다. `특정 조건을 만족하면`처럼 조건부로 추가 기능이 수행되면 기본 유스케이스 A를 확장하는 B이므로 `B <<extend>> A`로 본다.

### 선지별 해설

① X. `A <<include>> B`는 A를 수행할 때 B가 항상 포함되어 수행되는 관계이다. 문제의 핵심은 조건을 만족할 때만 B가 수행된다는 점이므로 `include`가 아니다. 틀린 표현은 `include`이며, 올바르게는 `extend`이다.
② X. `A <<extend>> B`는 A가 B를 확장하는 도식이므로 방향이 반대이다. 조건부로 추가되는 기능은 B가 되고, 기본 흐름은 A가 된다. 틀린 표현은 화살표의 출발점이 A인 부분이며, 올바르게는 `B <<extend>> A`이다.
③ X. `B <<include>> A`는 B를 수행할 때 A가 반드시 포함된다는 뜻이다. 문제는 A를 수행하는 도중 조건이 성립할 때 B가 추가되는 상황이므로 포함 관계도 아니고 주체도 반대이다. 틀린 표현은 `include`와 유스케이스의 역할 배치이며, 올바르게는 B가 A를 `extend`한다.
④ O. `B <<extend>> A`는 A가 기본 유스케이스이고 B가 조건이 만족될 때만 추가로 수행되는 확장 유스케이스라는 뜻이다. 문제의 서술인 `A 수행 도중 특정 조건을 만족하면 B 수행`과 정확히 일치한다.

### 오답 포인트

수험생은 `A를 수행하는 도중 B를 수행한다`는 문장만 보고 A에서 B로 화살표가 나가야 한다고 오해하기 쉽다. 그러나 유스케이스 다이어그램에서는 수행 순서 느낌보다 관계의 의미가 우선이며, `항상 포함`이면 `include`, `조건부 확장`이면 `extend`이다. 특히 `extend`는 확장 유스케이스에서 기본 유스케이스로 화살표가 향한다는 점이 함정이다.

NoteBase

### 다시 나오면 이렇게 판단

문장에 `항상`, `반드시`, `공통으로 수행`이 보이면 `include`를 먼저 떠올리고, `특정 조건에서만`, `선택적으로`, `예외적으로 추가`가 보이면 `extend`를 먼저 떠올린다. 그다음 화살표는 `include`는 기본 유스케이스에서 포함 유스케이스로, `extend`는 확장 유스케이스에서 기본 유스케이스로 향하는지 확인하면 된다.

### 암기 포인트

`include`는 항상 포함, `extend`는 조건부 확장이다. 조건이 붙으면 확장 유스케이스가 기본 유스케이스를 향해 `extend` 화살표를 보낸다.

NoteBase