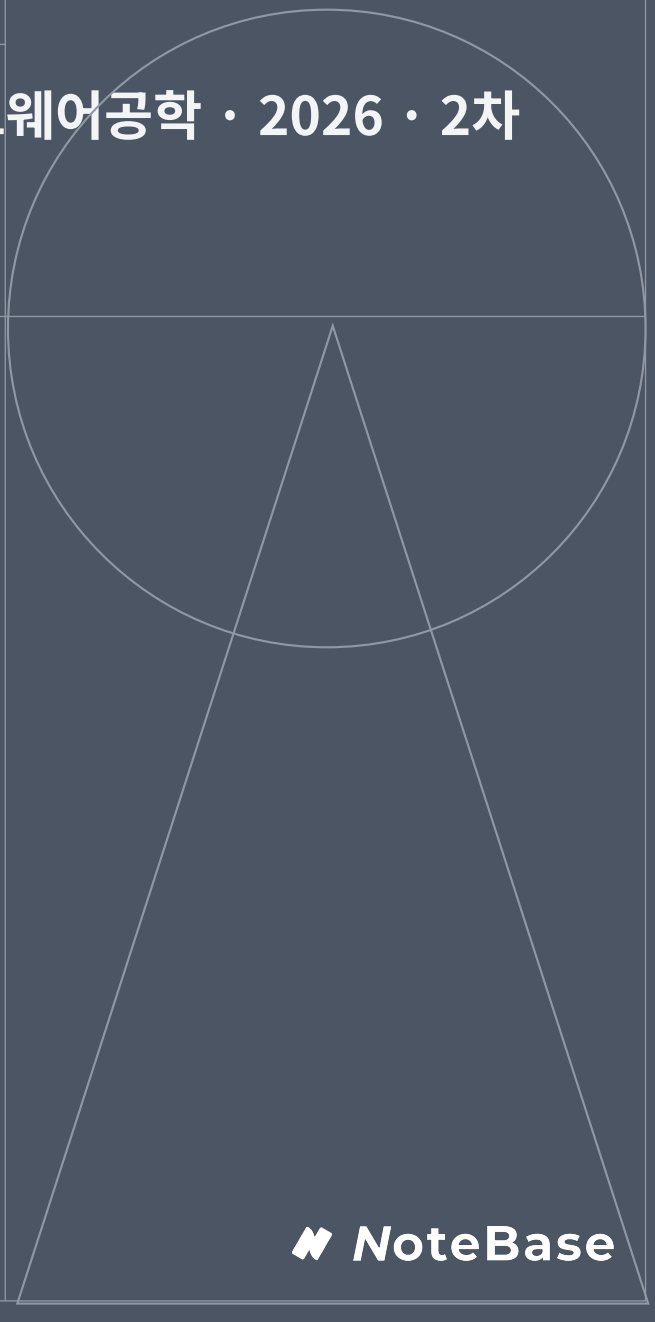
기출OX 해설집

공무원/국가직/7급 · 소프트웨어공학 · 2026 · 2차



NoteBase

기출OX 해설집

공무원/국가직/7급 · 소프트웨어공학 · 2026 · 2차

공무원/국가직/7급 · 소프트웨어공학 · 2026 · 2차

문 1. 소프트웨어 품질 평가모델 ISO/IEC 9126의 속성과 그 속성에 대한 설명으로 옳지 않은 것은?

① 이식성(portability): 별도의 작동이나 수단 없이 다양한 환경에서 적응될 수 있는 능력

② 사용성(usability): 쉽게 이해되고 배울 수 있고 사용될 수 있는 능력

③ 기능성(functionality): 사용되는 자원의 양에 따라 적절한 성능을 제공할 수 있는 능력

④ 신뢰성(reliability): 소프트웨어가 주어진 환경에서 정해진 수준의 성능을 유지할 수 있는 능력

정답 3번

간단한 해설

③의 '사용되는 자원의 양에 따라 적절한 성능을 제공하는 능력'은 **효율성**에 대한 설명이다. **기능성**은 정해진 사용 조건에서 명시적·묵시적 요구를 충족하는 기능을 제공하는 능력이다.

선지별 해설

① O: 이식성은 소프트웨어를 **한 환경에서 다른 환경으로 옮겨 사용할 수 있는 능력**이다. 선지는 이식성의 하위 특성인 적응성을 중심으로 설명한다. 여기서 '별도의 작동이나 수단 없이'란 환경 적응을 위해 소프트웨어에 제공된 수단 외에 추가적인 조치나 수단을 요구하지 않는다는 뜻이다.

② O: 사용성은 정해진 사용 조건에서 사용자가 소프트웨어를 **이해하고 배우며 사용할 수 있는 능력**이다. 사용 방법을 파악할 수 있는지, 조작법을 익히기 쉬운지, 원하는 작업을 수행하기 쉬운지 등이 판단 기준이다.

③ X: 사용한 자원의 양에 비추어 적절한 성능을 제공하는 능력은 **효율성**이다. 기능성은 정해진 사용 조건에서 명시적·묵시적 요구를 충족하는 기능을 제공하는 능력이므로, 선지의 속성명을 '기능성'에서 '효율성'으로 고쳐야 한다. 예를 들어 필요한 검색 기능을 제공하는지는 기능성의 문제이고, 그 검색을 수행할 때의 응답 시간과 메모리 사용량은 효율성의 문제이다.

④ O: 신뢰성은 **정해진 조건에서 요구되는 성능 수준을 유지하는 능력**이다. 자원을 얼마나 사용하여 성능을 내는지를 보는 효율성과 달리, 소프트웨어가 요구된 수준으로 계속 동작할 수 있는지를 평가한다.

NoteBase

문 2. 다음은 K 공공기관의 새로운 대국민 헬스케어 플랫폼 구축 프로젝트에 대한 요구사항 명세서의 일부이다. 이 프로젝트를 수행하기 위해 가장 적합한 소프트웨어 개발 프로세스 모델은?

◯ 도입하려는 AI 진단 알고리즘이 시장에 처음 적용되는 신기술이므로 초기 단계에서 시스템의 정확한 요구사항을 완벽히 정의하기 어렵다.
◯ 만약 프로젝트가 실패하거나 오류가 발생할 경우, 대규모 예산 낭비뿐만 아니라 국민 건강과 직결되는 심각한 위험(risk)이 도사리고 있다.
◯ 따라서 단번에 시스템을 완성하기보다는 주기마다 [목표 결정 → 위험 분석 → 개발 및 평가 → 다음 단계 계획]의 4가지 활동을 반복하며 점진적으로 완성도를 높여 가야 한다.

① 폭포수(waterfall) 모델
② 나선형(spiral) 모델
③ 컴포넌트 기반 개발(CBD) 모델
④ V(verification) 모델

정답 2번

간단한 해설

요구사항이 불확실하고 실패의 피해가 큰 프로젝트에서는 **위험 분석을 중심으로 개발을 반복하는 나선형 모델**이 적합하다. 매 주기마다 위험을 검토하고 개발 결과를 평가하여 다음 개발 범위를 구체화한다.

핵심 풀이

나선형 모델은 **위험을 분석하고 줄이면서 시스템을 점진적으로 개발**하는 모델이다. 제시된 프로젝트는 신기술을 도입하므로 초기 요구사항을 완전히 확정하기 어렵고, 오류가 예산 낭비와 국민 건강의 위험으로 이어질 수 있다. 따라서 요구사항의 불확실성을 반복적으로 해소하면서 큰 위험을 먼저 검토하는 나선형 모델이 적합하다.

주기별 활동은 '목표 결정 → 위험 분석 → 개발 및 평가 → 다음 단계 계획'으로 연결된다. 이번 주기의 목표와 대안을 정한 뒤, 각 대안의 위험과 해결 방안을 검토한다. 이를 바탕으로 해당 범위를 개발하고 결과를 평가하며, 평가 결과를 다음 주기의 계획에 반영한다. 단순히 개발을 여러 번 수행한다는 점보다 **매 주기에 위험 분석을 포함한다는 점**이 결정적인 단서이다.

폭포수 모델은 요구사항 분석·설계·구현·시험 등을 순차적으로 진행하는 방식이고, V 모델은 개발 단계와 이에 대응하는 시험 단계를 연결하여 검증을 강조한다. 컴포넌트 기반 개발(CBD) 모델은 재사용 가능한 컴포넌트를 조립·통합하는 데 중점을 둔다. 이들과 구별되는 나선형 모델의 핵심은 **위험 분석에 따라 반복 개발의 방향을 정한다는 것**이다.

NoteBase

문 3. 요구분석에 대한 설명으로 옳지 않은 것은?

① 요구분석 단계에서는 시스템의 기능이 무엇(what)인지에 초점을 두고 정리한다.

② 요구사항은 기능 요구사항과 비기능 요구사항으로 구분할 수 있으며, 기능 요구사항은 품질을 포함한다.

③ 신뢰성(reliability)은 소프트웨어를 믿고 사용할 수 있는 것을 의미하며, 정확성(correctness)은 사용자가 요구하는 기능을 충족하는 정도 또는 요구분석 명세서와 일치하는 정도를 의미한다.

④ 가용성은 소프트웨어의 총 운용시간 중 고장 없이 정상적으로 가동된 시간의 비율(%)로 나타낸 것으로 "[MTTF ÷ (MTTF + MTTR)] × 100"으로 계산한다.

정답 2번

간단한 해설

②의 '기능 요구사항은 품질을 포함한다'는 품질 요구사항의 분류를 잘못 설명한 것이다. 기능 요구사항은 시스템이 수행할 기능을 나타내며, 성능·신뢰성 등 품질에 관한 요구는 **비기능 요구사항**에 해당한다.

선지별 해설

① O: 요구분석은 사용자의 필요를 파악하여 시스템이 **무엇을 해야 하는지**를 명확히 하는 단계이다. 제공할 기능과 제약조건 등을 요구사항으로 정리하며, 구체적인 구현 방법을 정하는 설계와 구별된다.

② X: 기능 요구사항은 입력 처리, 정보 조회, 결과 출력처럼 시스템이 제공해야 할 동작이나 서비스를 규정한다. 성능·신뢰성·사용성 등 품질에 관한 요구는 **비기능 요구사항**으로 분류한다. 예를 들어 '상품을 검색할 수 있어야 한다'는 기능 요구사항이고, '검색 결과를 2초 이내에 표시해야 한다'는 성능에 관한 비기능 요구사항이다. 따라서 '기능 요구사항은 품질을 포함한다'는 '비기능 요구사항은 품질에 관한 요구를 포함한다'로 고친다.

③ O: 신뢰성은 정해진 조건에서 일정 기간 **고장 없이 기능을 수행하는 능력**이며, 그 확률로 나타낼 수 있다. 정확성은 소프트웨어가 사용자의 요구와 요구분석 명세서에 맞게 기능을 수행하는 정도이다. 즉, 신뢰성은 운용 중 기능을 안정적으로 수행하는지에, 정확성은 요구된 기능과 결과가 명세에 부합하는지에 초점을 둔다.

④ O: 가용성은 전체 운용시간 중 **정상적으로 사용할 수 있는 시간의 비율**이다. 정상 가동과 고장 후 복구가 반복되는 운용에서 MTTF는 고장까지의 평균 정상 가동시간이고, MTTR은 평균 복구시간이다. 평균 정상 가동시간과 평균 복구시간을 더한 값이 한 주기의 평균 총시간이므로, 가용성(%)은 $\frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}} \times 100$으로 계산한다. 같은 MTTF에서 MTTR이 짧아지면 사용할 수 없는 시간이 줄어 가용성이 높아진다.

NoteBase

문 4. 다음에서 설명하고 있는 프로젝트 관리 방법론의 주요 특징으로 옳지 않은 것은?

이 방법론은 프로젝트를 작은 단위의 작업과 단계로 나누어 신속한 피드백과 반복 작업을 통해 진행하는 관리 방식으로, 변화하는 요구사항에 유연하게 대응하며, 팀원 간의 협력과 소통을 중시한다. 기간이 고정된 스프린트 단위로 업무를 진행하고, 지속적인 검토를 통해 개선된 가치를 제공하며, 유연성과 팀워크를 우선으로 하여 빠른 변화에 적응할 수 있도록 설계되었다.

① 모든 설계 결정을 초기에 고정하고 변경을 최소화한다.

② 변화 요구에 대해 수용적이며 고객과의 지속적 협업을 강조한다.

③ 반복적 개발을 통해 기능을 점진적으로 추가하며 배포한다.

④ 문서화 중심이 아닌 실행 가능한 소프트웨어 중심의 개발 접근 방식이다.

정답 1번

간단한 해설

①의 '모든 설계 결정을 초기에 고정'한다는 설명은 애자일의 변화 대응 원칙과 맞지 않는다. 애자일은 반복 과정에서 얻은 피드백을 바탕으로 **요구사항과 설계를 유연하게 조정**한다.

선지별 해설

제시문의 짧은 반복, 신속한 피드백, 변화하는 요구사항에 대한 대응은 애자일의 특징이다. 기간을 고정한 스프린트는 애자일을 실천하는 스크럼에서 사용하는 반복 작업 단위이다.

① X: 애자일은 개발 초기에 세운 설계를 확정된 것으로 보지 않고, 구현 결과와 고객 피드백에 따라 **필요한 설계 변경을 수용**한다. 따라서 '모든 설계 결정을 초기에 고정하고 변경을 최소화한다'는 표현은 '변화하는 요구사항에 맞추어 설계를 점진적으로 발전시킨다'로 고쳐야 한다.

② O: 고객과 지속적으로 협업하면 실제로 필요한 기능과 우선순위를 확인할 수 있다. 애자일은 이 과정에서 드러난 **요구사항의 변화를 수용**하여 고객에게 제공할 가치를 높인다.

③ O: 짧은 개발 주기를 반복하면서 작동하는 기능을 조금씩 완성하고 전달한다. 각 반복에서 얻은 피드백을 다음 개발에 반영하므로, 전체 기능을 한꺼번에 완성하기보다 **기능을 점진적으로 추가**해 나간다.

④ O: 애자일은 방대한 문서를 만드는 것보다 **작동하는 소프트웨어의 제공**에 더 큰 가치를 둔다. 실제 동작을 통해 고객의 요구가 충족되는지 확인할 수 있기 때문이다. 이는 문서화를 없앤다는 뜻이 아니라 필요한 문서를 작성하되 소프트웨어가 제공하는 가치를 우선한다는 뜻이다.

NoteBase

문 5. 객체지향 분석 및 설계 과정에서 시스템의 모델링을 위해 다양한 UML 다이어그램이 활용된다. 다음 도서 관리 시스템 개발 시나리오에 적용할 가장 적절한 UML 다이어그램을 바르게 연결한 것은?

(가) 사용자가 도서 대출을 요청할 때, 회원 객체, 대출 관리 객체, 도서 객체 간에 발생하는 메시지 호출과 응답을 시간의 흐름에 따라 도식화하여 객체 간의 상호작용을 분석한다.
(나) 단일 도서 객체가 시스템에 등록되어 대출 가능, 대출 중, 예약됨, 파손 및 폐기에 이르기까지 발생하는 외부 이벤트에 의해 객체 내부가 어떻게 전이되는지를 모델링한다.
(다) 사용자가 도서를 반납할 때 연체료 계산 처리와 반납 완료 안내 이메일 발송 처리가 동시에 병렬로 실행되고, 두 작업이 모두 완료된 후 다음 단계로 합쳐지는 전체적인 비즈니스 업무 흐름을 시각적으로 도식화한다.
A. 순차 다이어그램(sequence diagram)
B. 유스케이스 다이어그램(use case diagram)
C. 활동 다이어그램(activity diagram)
D. 상태 다이어그램(state diagram)

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

정답 2번

간단한 해설

한 객체가 외부 이벤트에 따라 어떤 상태로 바뀌는지는 **상태 다이어그램**으로 나타낸다. 여러 업무의 병렬 실행과 완료 후 동기화는 **활동 다이어그램**으로 표현한다.

NoteBase

대응 관계 해설

(가) A. 순차 다이어그램: 여러 객체 사이에서 **메시지가 오가는 시간적 순서**를 표현한다. 회원 객체, 대출 관리 객체, 도서 객체가 대출 요청을 처리하며 주고받는 호출과 응답이 분석 대상이므로 순차 다이어그램에 해당한다. B의 유스케이스 다이어그램은 외부 사용자와 시스템이 제공하는 기능 사이의 관계를 나타내므로, 객체 간 호출과 응답의 순서를 상세히 표현하는 목적과 구별된다.

(나) D. 상태 다이어그램: **한 객체의 상태와 이벤트에 따른 상태 전이**를 표현한다. 여기서는 도서 객체 하나를 중심으로 대출 가능, 대출 중, 예약됨 등의 상태가 외부 이벤트에 따라 어떻게 바뀌는지를 모델링하므로 상태 다이어그램에 해당한다. 여러 작업의 진행 흐름보다 도서 객체 자체의 상태 변화가 핵심이다.

(다) C. 활동 다이어그램: 업무를 이루는 활동과 그 사이의 제어 흐름을 표현하며, 병렬 처리와 동기화도 나타낼 수 있다. 연체료 계산과 이메일 발송을 병렬로 시작하는 것은 분기(fork), **두 작업이 모두 끝날 때까지 기다린 뒤 다음 단계로 진행**하는 것은 동기화 합류(join)에 해당한다. 따라서 전체 반납 업무의 흐름을 표현하는 활동 다이어그램이 적합하다.

따라서 (가)는 A, (나)는 D, (다)는 C로 연결되며, 이 조합은 ②와 일치한다.

NoteBase

문 6. 다음 요구사항 명세서에 대한 설명 중 옳지 않은 것은?

◯ 요구사항 ID: REQ-0001
◯ 기능: 사용자 로그인
◯ 설명: 사용자는 로그인 단계에서 아이디와 비밀번호를 입력하여 로그인한다.
◯ 제약조건: 비밀번호는 반드시 8자 이상이어야 하며, 대문자, 소문자와 숫자를 포함해야 한다.

① 요구사항 ID: REQ-0001는 각 요구사항을 구분하는 유일한 식별자이다.
② 기능은 요구사항의 핵심 기능을 서술한다.
③ 설명은 사용자 로그인 기능의 구현 방법을 상세히 기술한다.
④ 제약조건은 요구사항의 조건이나 제한사항을 명확히한다.

정답 3번

간단한 해설

'설명'에는 사용자가 아이디와 비밀번호를 입력해 로그인한다는 **요구 기능의 동작**이 적혀 있다. 이는 시스템이 무엇을 해야 하는지를 나타내며, 기능을 어떻게 구현할지에 관한 상세 설계는 아니다.

선지별 해설

① O: 요구사항 ID는 **각 요구사항을 고유하게 식별**하기 위한 값이다. 제시된 REQ-0001은 이 로그인 요구사항을 다른 요구사항과 구분하는 식별자이다.

② O: '기능' 항목은 요구사항이 다루는 **핵심 기능을 간략히 나타낸다**. 제시문에서는 '사용자 로그인'으로 수행할 기능을 명시하고, 구체적인 동작은 '설명' 항목에 적었다.

③ X: '설명' 항목은 사용자가 로그인 단계에서 아이디와 비밀번호를 입력한다는 **요구 기능의 동작**을 서술한다. 로그인 기능을 어떤 알고리즘이나 내부 처리 구조로 구현하는지는 제시하지 않았다. 따라서 '구현 방법을 상세히 기술한다'는 표현은 '사용자 로그인 기능이 수행해야 할 동작을 기술한다'로 고쳐야 한다.

④ O: '제약조건' 항목은 기능을 제공할 때 지켜야 할 **조건이나 제한사항**을 명시한다. 이 요구사항에서는 비밀번호가 반드시 8자 이상이어야 하고, 대문자·소문자·숫자를 모두 포함해야 한다는 조건을 정했다.

NoteBase

문 7. 다음 Java로 작성된 코드의 기본경로 테스트를 위한 테스트 케이스 수는?

```
void test(int x) {
   int y;
   String str;
   if(x >= 90){
      y = 100;
      str = "Pass";
   }
   else{
      str = "Not Yet";
      if(x >= 80){
         y = 80;
         str = str + ": Try Again";
      }
      else
         y = 50;
   }
}
```

① 1

② 2

③ 3

④ 4

정답 3번

간단한 해설

분기 조건은 x ≥ 90과 x ≥ 80의 **두 개**이다. 기본경로 테스트의 테스트 케이스 수는 순환복잡도로 구하므로 2 + 1 = 3개이다.

NoteBase

계산 과정

기본경로 테스트는 제어 흐름에서 서로 독립적인 경로를 실행하도록 테스트 케이스를 구성한다. 이 코드처럼 하나로 연결된 제어 흐름에서 각 판정이 참·거짓으로 갈라지면, 순환복잡도는 $V(G) = D + 1$로 구한다. 여기서 D는 판정 노드 수이고, 순환복잡도는 독립 경로 수를 나타낸다.

판정 노드는 x ≥ 90과 x ≥ 80의 두 개이다. **else는 앞선 조건이 거짓일 때의 흐름**이므로 별도의 판정 노드로 세지 않는다. 따라서 D = 2이고, 순환복잡도는 2 + 1 = 3이다.

경로 1: x ≥ 90이면 첫 번째 if의 참 분기로 들어가 y에 100을 대입하고 str에 'Pass'를 대입한 뒤 종료한다. 예를 들어 x = 90으로 실행할 수 있다.

경로 2: x < 90이면서 x ≥ 80이면 첫 번째 if의 거짓 분기로 들어간 뒤 두 번째 if의 참 분기를 실행한다. y에는 80이 대입된다. 예를 들어 x = 80으로 실행할 수 있다.

경로 3: x < 80이면 두 조건이 모두 거짓이므로 안쪽 else에서 y에 50을 대입하고 종료한다. 예를 들어 x = 79로 실행할 수 있다.

이 세 독립 경로를 각각 실행하는 테스트 케이스가 필요하므로 테스트 케이스 수는 3개이며, ③의 값과 일치한다.

NoteBase

문 8. 아래와 같은 C 프로그램이 있을 때, 아래 프로그램의 함수 write_i와 read_i 간 결합(coupling) 수준으로 옳은 것은?

```
int global_i;
void write_i (int x) {
   global_i = x;
}
int read_i () {
   return global_i;
}
int main() {
   int ret;
   write_i(4);
   ret = read_i();
   return 0;
}
```

① 데이터 결합(data coupling)

② 스탬프 결합(stamp coupling)

③ 내용 결합(content coupling)

④ 공통 결합(common coupling)

정답 4번

간단한 해설

write_i가 값을 저장하고 read_i가 읽는 대상은 두 함수가 공유하는 전역 변수 global_i이다. 이처럼 **전역 변수를 통해 데이터를 공유**하는 관계를 **공통 결합**이라고 한다.

NoteBase

핵심 풀이

공통 결합은 여러 모듈이 같은 전역 데이터를 공유하여 서로 의존하는 관계이다. 프로그램 맨 위의 global_i는 함수 밖에 선언된 전역 변수이다. write_i는 global_i = x로 그 값을 바꾸고, read_i는 return global_i로 같은 변수의 값을 읽는다. main에서 write_i(4)를 호출하면 global_i에 4가 저장되고, 이어 read_i()를 호출하면 그 값이 반환되어 ret에 4가 저장된다. 두 함수의 연결 통로가 **공유 전역 변수 global_i**이므로 공통 결합에 해당한다.

데이터 결합과의 구분: 데이터 결합은 모듈 사이에서 필요한 개별 데이터를 **매개변수로 전달**하는 관계이다. write_i에 매개변수 x가 있다는 사실만으로 write_i와 read_i 사이를 데이터 결합이라고 판단하면 안 된다. x는 main이 write_i에 값을 전달할 때 사용하며, read_i는 write_i로부터 인수를 받지 않고 전역 변수에 저장된 값을 읽는다.

스탬프 결합·내용 결합과의 구분: 스탬프 결합은 구조체 같은 복합 자료구조를 매개변수로 전달하는 관계이다. 내용 결합은 한 모듈이 다른 모듈의 내부 자료나 코드에 직접 접근하는 관계이다. 이 프로그램은 복합 자료구조를 전달하지 않으며, 두 함수 중 하나가 다른 함수의 내부에 접근하지도 않는다. **함수 밖의 공통 데이터를 공유**한다는 점이 판단 기준이다.

NoteBase

문 9. 다음에 제시된 소프트웨어 유지보수 종류 (가) ~ (라)와 설명 A ~ D를 바르게 연결한 것은?

(가) 수정 유지보수(corrective maintenance)
(나) 적응 유지보수(adaptive maintenance)
(다) 완전 유지보수(perfective maintenance)
(라) 예방 유지보수(preventive maintenance)
A. 시스템이 정상 작동하더라도 더 나은 품질이나 성능을 위해 이루어지는 유지보수
B. 기존 시스템을 새로운 자료나 운영체제, 하드웨어 환경으로 이식하기 위한 유지보수
C. 발견된 오류나 결함의 원인을 찾아 문제를 해결하는 유지보수
D. 장차 발생할 수 있는 오류나 성능 저하에 대비하여 사전에 방지하기 위한 유지보수

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

정답 4번

간단한 해설

정상 작동하는 시스템의 품질·성능을 더 높이는 것은 **완전 유지보수**이고, 앞으로 생길 오류나 성능 저하를 미리 막는 것은 **예방 유지보수**이다. 두 유형은 각각 현재의 개선과 미래 문제의 방지에 목적이 있다.

NoteBase

대응 관계 해설

(가) 수정 유지보수 → C: **이미 발견된 오류나 결함을 해결**하는 유지보수이다. 잘못된 계산 결과를 내는 프로그램에서 오류 원인을 찾아 코드를 고치는 경우가 이에 해당한다.

(나) 적응 유지보수 → B: **달라진 운영 환경에서도 사용할 수 있도록** 소프트웨어를 변경하는 유지보수이다. 새로운 자료 형식, 운영체제 또는 하드웨어 환경으로 기존 시스템을 이식하는 작업이 해당한다.

(다) 완전 유지보수 → A: 시스템이 정상 작동하더라도 **품질이나 성능을 더 높이기 위해** 개선하는 유지보수이다. 예를 들어 정상적으로 결과를 내는 기능의 응답 시간을 줄이는 작업이 해당한다. 발견된 결함을 고치는 수정 유지보수와 목적이 다르다.

(라) 예방 유지보수 → D: **미래에 발생할 수 있는 문제를 사전에 방지**하는 유지보수이다. 아직 오류가 드러나지 않았더라도 잠재적인 결함 요인을 제거하여 향후 오류나 성능 저하의 가능성을 줄인다.

따라서 (가)는 C, (나)는 B, (다)는 A, (라)는 D에 대응하며, 이 연결은 ④와 일치한다.

NoteBase

문 10. 다음은 비기능 요구사항을 측정 가능한 형태로 명세한 것이다. 각 요구사항이 나타내는 시스템의 비기능적 속성을 순서대로 바르게 나열한 것은?

(가) 사용자 요청에 대한 응답시간은 2초 이내여야 한다.
(나) 신규 사용자는 2시간 이내의 교육으로 주요 기능을 사용할 수 있어야 한다.
(다) 시스템의 평균 고장 간격은 1,000시간 이상이어야 한다.
(라) 장애 발생 후 시스템은 30초 이내에 재시작되어야 한다.
(마) 시스템은 최소 3개의 대상 운영환경에서 실행될 수 있어야 한다.

① (가): 속도, (나): 사용성, (다): 신뢰성, (라): 견고성, (마): 이식성
② (가): 신뢰성, (나): 사용성, (다): 속도, (라): 견고성, (마): 이식성
③ (가): 속도, (나): 신뢰성, (다): 사용성, (라): 이식성, (마): 견고성
④ (가): 속도, (나): 사용성, (다): 견고성, (라): 신뢰성, (마): 이식성

정답 1번

간단한 해설

평균 고장 간격은 **얼마나 드물게 고장 나는지**를 나타내는 신뢰성의 지표이고, 장애 후 재시작 시간은 **장애에 대처하고 회복하는 능력**인 견고성의 지표이다. 같은 시간 단위로 명세되어도 측정하는 대상에 따라 비기능적 속성이 달라진다.

NoteBase

대응 관계 해설

(가) 속도: 사용자 요청에 응답하기까지 걸리는 시간은 **처리의 빠르기**를 측정한다. 응답시간을 2초 이내로 제한하므로 속도에 관한 요구사항이다.

(나) 사용성: 신규 사용자가 주요 기능을 익히는 데 필요한 교육시간은 **배우고 사용하기 쉬운 정도**를 측정한다. 2시간 이내의 교육으로 주요 기능을 사용할 수 있어야 한다는 조건은 사용성에 해당한다. 시스템 자체의 처리시간을 측정하는 속도와 구별된다.

(다) 신뢰성: 신뢰성은 주어진 조건에서 고장 없이 기능을 수행하는 능력이다. **평균 고장 간격**은 고장이 얼마나 자주 발생하는지를 나타내며, 이 간격을 1,000시간 이상으로 요구하는 것은 신뢰성에 관한 명세이다.

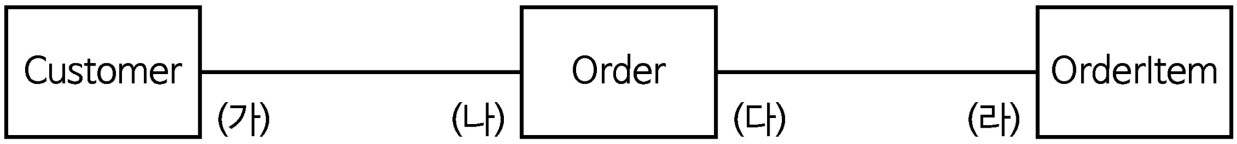
(라) 견고성: 견고성은 오류나 장애 상황에 대처하는 능력이며, 이 문항에서는 **장애 후 재시작에 걸리는 시간**으로 측정한다. 장애가 발생한 뒤 30초 이내에 재시작해야 한다는 조건이므로 견고성에 해당한다. 고장 발생 사이의 평균 간격을 측정하는 신뢰성과는 측정 대상이 다르다.

(마) 이식성: 이식성은 소프트웨어를 다른 운영환경으로 옮겨 실행할 수 있는 정도이다. **실행 가능한 대상 운영환경의 수**를 최소 3개로 요구하므로 이식성에 관한 명세이다.

따라서 (가) 속도, (나) 사용성, (다) 신뢰성, (라) 견고성, (마) 이식성의 대응이며, ①의 조합과 일치한다.

NoteBase

문 11. 다음은 온라인 쇼핑몰 시스템의 도메인 모델을 나타낸 클래스 다이어그램의 일부에 대한 설명이다. 주어진 비즈니스 규칙을 바탕으로 도출한 다중성(multiplicity) 표기로 옳은 것은?



<비즈니스 규칙>

○ 고객(Customer)과 주문(Order) 클래스는 (가):(나) 관계이다. 하나의 고객은 여러 번 주문할 수 있으며, 아직 단 한 번도 주문하지 않은 신규 가입 고객도 시스템에 존재할 수 있다. 특정 주문은 오직 하나의 고객에게만 종속된다.

○ 주문(Order)과 주문품목(OrderItem) 클래스는 (다):(라) 관계이다. 하나의 주문에는 반드시 하나 이상의 주문품목이 포함되어야 하며, 특정 주문품목은 오직 하나의 주문 번호에만 종속된다.

① (가): 1, (나): 1..*, (다): 1, (라): 1

② (가): 1, (나): 0..*, (다): 1, (라): 1..*

③ (가): 1..*, (나): 1, (다): 0..*, (라): 1..*

④ (가): 0..*, (나): 1, (다): 1..*, (라): 1

정답 2번

간단한 해설

주문 이력이 없는 고객도 존재하므로 고객 한 명에 연결되는 주문은 **0..***이다. 반면 주문 하나에는 주문품목이 반드시 하나 이상 있어야 하므로 주문품목 쪽은 **1..***이다.

NoteBase

빈칸별 해설

클래스 다이어그램에서 선의 한쪽 끝에 적힌 다중성은 반대쪽 객체 하나에 연결될 수 있는 해당 쪽 객체의 수를 뜻한다. 1은 정확히 하나, 0..*는 0개 이상, 1..*는 1개 이상을 나타낸다.

(가) 1: Customer 쪽에 있으므로 주문 하나에 연결되는 고객 수를 나타낸다. 특정 주문은 **오직 하나의 고객**에게만 종속되므로 1이다.

(나) 0..*: Order 쪽에 있으므로 고객 한 명에 연결되는 주문 수를 나타낸다. 여러 번 주문할 수 있어 상한은 제한되지 않으며, **주문하지 않은 신규 고객도 존재**하므로 하한은 0이다. 1..*로 표시하면 주문이 없는 고객을 허용하지 못한다.

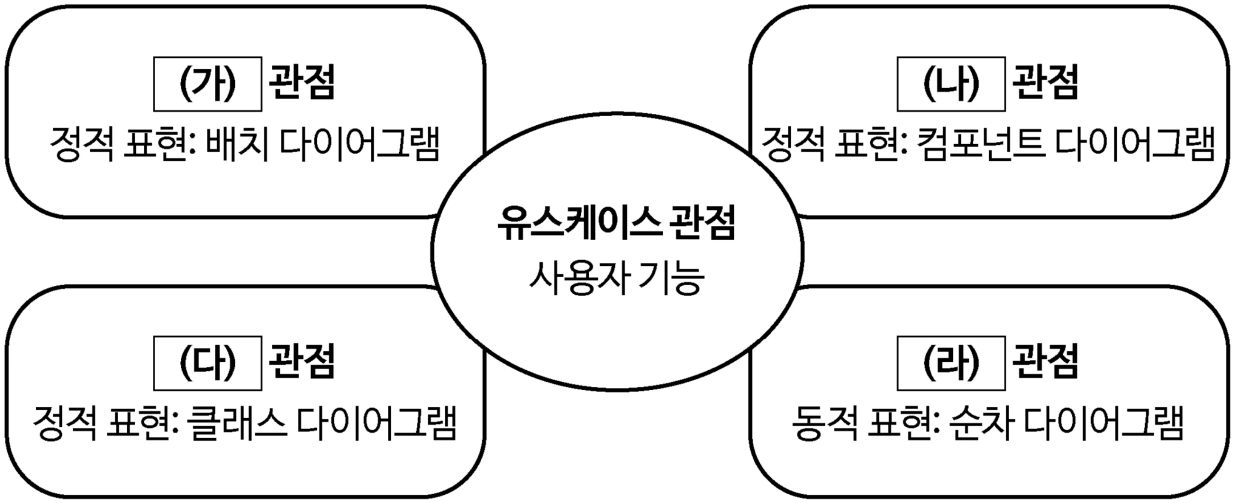
(다) 1: Order 쪽에 있으므로 주문품목 하나에 연결되는 주문 수를 나타낸다. 특정 주문품목은 **오직 하나의 주문번호**에만 종속되므로 1이다.

(라) 1..*: OrderItem 쪽에 있으므로 주문 하나에 연결되는 주문품목 수를 나타낸다. 하나의 주문에는 **반드시 하나 이상의 주문품목**이 포함되어야 하므로 하한은 1이고, 여러 품목을 포함할 수 있으므로 상한은 제한되지 않는다.

따라서 (가) 1, (나) 0..*, (다) 1, (라) 1..*의 조합이며, ②와 일치한다.

NoteBase

문 12. 다음은 소프트웨어 아키텍처의 UML 4 + 1 관점에 대한 그림과 설명이다. 빈칸 (가) ~ (라)에 들어갈 관점에 대해 바르게 연결한 것은?



(가) 시스템을 구성하는 처리 장치 간 물리적 배치에 초점을 둔다.
(나) 소프트웨어 서브 시스템의 모듈 간 연관 관계와 이들이 어떻게 연결되는지에 초점을 둔다.
(다) 시스템의 기능을 제공하기 위해 필요한 클래스, 컴포넌트 및 이들의 관계에 초점을 둔다.
(라) 독자적인 제어 스레드를 가질 수 있는 클래스에 집중하며, 시스템의 동시성과 동기화에 초점을 둔다.

① (가): 배치, (나): 구현, (다): 논리적, (라): 프로세스
② (가): 구현, (나): 프로세스, (다): 논리적, (라): 배치
③ (가): 배치, (나): 논리적, (다): 프로세스, (라): 구현
④ (가): 배치, (나): 프로세스, (다): 논리적, (라): 구현

정답 1번

간단한 해설

소프트웨어 모듈이 어떻게 구성되고 연결되는지는 **구현 관점**에서, 실행 중 스레드들이 어떻게 함께 동작하고 조정되는지는 **프로세스 관점**에서 다룬다. 그림의 컴포넌트 다이어그램과 동작을 나타내는 순차 다이어그램도 이 차이를 보여 준다.

NoteBase

빈칸별 해설

(가) 배치 관점: 시스템을 구성하는 **처리 장치의 물리적 배치**와 연결을 다룬다. 소프트웨어를 어느 장치에서 실행할지와 관련되므로, 그림 왼쪽 위의 배치 다이어그램과 대응한다.

(나) 구현 관점: 소프트웨어를 이루는 **모듈의 구성과 의존 관계**를 다룬다. 서브시스템의 모듈들이 어떻게 나뉘고 연결되는지를 나타내므로, 그림 오른쪽 위의 컴포넌트 다이어그램과 대응한다. 실행 중의 동시성보다 소프트웨어의 구성 구조에 초점을 둔다.

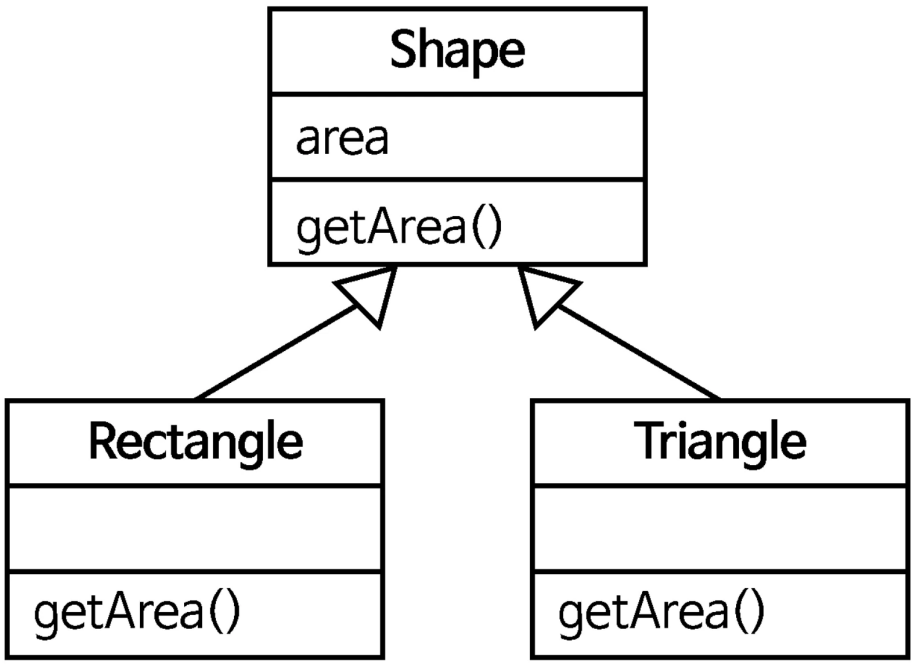
(다) 논리적 관점: 시스템이 요구된 기능을 제공하려면 어떤 클래스와 관계가 필요한지를 다룬다. 핵심은 **기능을 실현하는 논리적 구조**이므로, 그림 왼쪽 아래의 클래스 다이어그램과 대응한다. 구현 관점이 모듈 구성에 초점을 둔다면, 논리적 관점은 기능을 담당하는 클래스와 그 관계에 초점을 둔다.

(라) 프로세스 관점: 독자적인 제어 스레드를 갖는 실행 주체들과 그 상호작용을 다룬다. 여러 실행 흐름이 함께 진행되는 동시성과, 필요한 순서를 맞추는 동기화가 핵심이다. 그림 오른쪽 아래의 순차 다이어그램은 메시지 교환 순서를 동적으로 나타내며, 지문의 **제어 스레드·동시성·동기화**가 프로세스 관점을 가리키는 결정적 단서이다. 이 문항의 '프로세스'는 개발 절차를 뜻하지 않는다.

따라서 (가) 배치, (나) 구현, (다) 논리적, (라) 프로세스로 연결되며, 이는 ①의 조합과 일치한다.

NoteBase

문 13. 다음 클래스 다이어그램은 Shape 클래스와 그 하위 클래스 Rectangle, Triangle의 관계를 나타낸 것이다. 아래 코드 실행 결과를 올바르게 예측한 것은?




```
Shape p1 = new Rectangle();
Shape p2 = new Triangle();

p1.getArea();
p2.getArea();
```

① getArea()는 하위 클래스에만 선언되어 있기 때문에 컴파일 오류가 발생한다.

② p1.getArea()와 p2.getArea() 모두 Shape 클래스의 getArea()가 실행된다.

③ p1.getArea()는 Rectangle의 메서드가, p2.getArea()는 Triangle의 메서드가 실행된다.

④ Rectangle이나 Triangle은 Shape의 메서드를 상속받기 때문에 각각의 getArea()는 실행되지 않는다.

정답 3번

간단한 해설

상위 클래스 타입의 참조 변수로 호출해도, 재정의된 메서드는 **실제 객체의 타입**에 따라 실행된다. p1은 Rectangle 객체를, p2는 Triangle 객체를 참조하므로 각 하위 클래스의 getArea()가 실행된다.

NoteBase

공통 해설

이 문제는 메서드 재정의에 따른 **동적 바인딩**을 적용한다. 상위 클래스의 메서드를 하위 클래스가 재정의하면, 상위 클래스 타입의 참조 변수로 호출하더라도 실행 시 실제 객체에 맞는 메서드가 선택된다. 참조 변수의 선언 타입은 호출 가능한 메서드를 확인하는 기준이고, 실제 객체의 타입은 실행할 재정의 메서드를 결정하는 기준이다.

다이어그램의 빈 삼각형 화살표는 Rectangle과 Triangle이 Shape를 상속함을 나타낸다. Shape에도 getArea()가 선언되어 있으므로 Shape 타입인 p1과 p2로 이 메서드를 호출할 수 있다. 따라서 '하위 클래스에만 선언되어 있다'는 이유로 컴파일 오류가 발생하지 않는다. 두 하위 클래스에도 getArea()가 표시되어 있어 각각 재정의한 메서드를 사용한다.

p1은 new Rectangle()로 생성한 객체를 참조하므로 p1.getArea()는 Rectangle의 메서드를 실행한다. p2는 new Triangle()로 생성한 객체를 참조하므로 p2.getArea()는 Triangle의 메서드를 실행한다. 두 변수의 선언 타입이 Shape라는 이유로 Shape의 구현이 실행되는 것은 아니며, 상속받은 메서드를 재정의하면 해당 객체의 하위 클래스 구현이 실행된다.

NoteBase

문 14. 소프트웨어 개발에서 모델링에 대한 설명으로 옳지 않은 것은?

① 자연어를 사용하여 모델링하는 경우 일반 사용자가 이해하기 쉬운 장점이 있으나, 해석이 달라질 수 있는 문제가 있다.

② 자연어로 작성하기에 복잡한 요구사항들을 모델링할 때, 형식 언어(formal language)를 사용하면 보다 간결하게 표현할 수 있다.

③ 데이터 흐름도(DFD, Data Flow Diagram) 표기법은 소프트웨어를 모델링하기 위한 구조적 방법이다.

④ E-R 다이어그램 표기법은 소프트웨어를 모델링하기 위한 객체지향 방법이다.

정답 4번

간단한 해설

④의 E-R 다이어그램은 **개체·속성·관계로 데이터 구조를 표현**하는 모델링 표기법이다. 이를 객체의 상태와 동작을 함께 다루는 **객체지향 방법으로 분류한 것**이 잘못이다.

선지별 해설

① O: 자연어는 일상적으로 쓰는 언어이므로 전문적인 표기법을 배우지 않은 일반 사용자도 요구사항을 이해하기 쉽다. 다만 단어나 문장의 의미가 문맥과 독자에 따라 달라져 **동일한 요구사항을 서로 다르게 해석**할 수 있다.

② O: 형식 언어는 문법과 의미가 엄밀하게 정해진 언어이다. 수학적 기호와 논리식을 이용하면 자연어로 길고 복잡하게 서술해야 하는 조건을 **간결하고 명확하게 표현**할 수 있다. 다만 이를 이해하려면 해당 기호와 규칙에 대한 지식이 필요하다.

③ O: 데이터 흐름도(DFD)는 데이터가 어디에서 들어와 어떤 처리를 거쳐 어디로 전달되거나 저장되는지를 나타낸다. 시스템을 처리 기능으로 나누고 이들 사이의 **데이터 흐름을 중심으로 분석**하므로 구조적 방법에 쓰이는 모델링 표기법이다.

④ X: E-R 다이어그램은 개체, 속성, 개체 간 관계를 이용하여 **데이터의 구조를 모델링**한다. 예를 들어 '학생'과 '과목'이라는 개체 사이에 '수강' 관계를 표현한다. 따라서 '객체지향 방법'이라는 분류가 잘못이다. 객체지향 모델링에서는 클래스 다이어그램 등을 이용해 객체의 속성과 동작, 객체 사이의 관계를 표현한다.

NoteBase

문 15. 화이트박스 테스트 검증 기준에 대한 설명 중 옳지 않은 것은?

① 기본경로 테스트(basic path test)는 프로그램의 실행 흐름을 기준으로 원시 코드의 독립적인 경로가 최소 한 번은 실행되도록 테스트 케이스를 설계한다.

② 다중조건 커버리지(multiple condition coverage)는 프로그램 내의 결정문을 결정하는 조건의 참/거짓의 모든 조합이 실행될 수 있도록 테스트 케이스를 설계한다.

③ 조건 커버리지(condition coverage)는 프로그램 내의 결정문을 결정하는 조건의 참/거짓이 최소한 한 번은 실행될 수 있도록 테스트 케이스를 설계한다.

④ 조건/결정 커버리지(condition/decision coverage)는 분기를 결정하는 각 개별 조건식이 다른 개별 조건식에 영향을 받지 않고 전체 조건식의 결과에 독립적으로 영향을 주도록 테스트 케이스를 설계한다.

정답 4번

간단한 해설

④의 '각 개별 조건이 전체 결과에 독립적으로 영향을 준다'는 **변경 조건/결정 커버리지(MC/DC)**의 요구 사항이다. 조건/결정 커버리지는 **각 개별 조건과 전체 결정 결과가 각각 참·거짓을 최소 한 번씩** 갖도록 한다.

선지별 해설

① O: 기본경로 테스트는 프로그램의 제어 흐름을 분석하여 **서로 독립적인 기본 경로들**을 구하고, 각 경로를 최소 한 번 실행하도록 테스트 케이스를 설계한다. 반복 횟수까지 달리한 모든 가능한 실행 경로를 전부 검사한다는 뜻은 아니다.

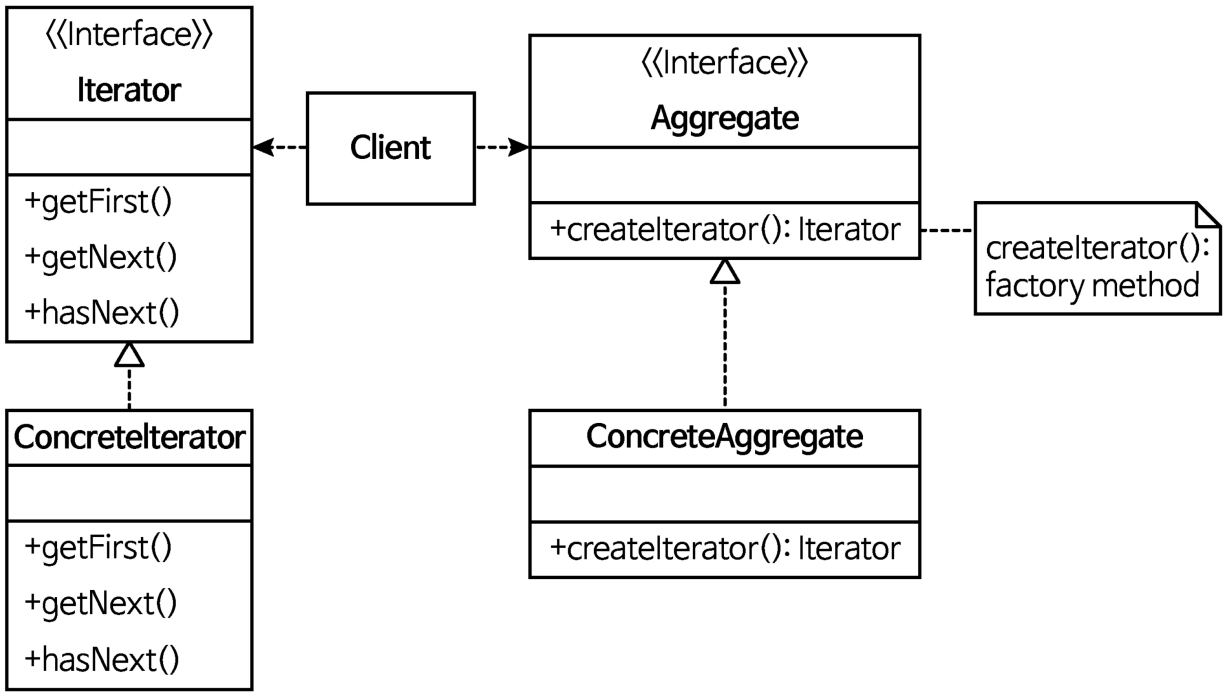
② O: 다중조건 커버리지는 하나의 결정을 구성하는 **개별 조건들의 참·거짓 조합 전체**를 검사한다. 예를 들어 두 조건 A와 B가 각각 참·거짓을 자유롭게 가질 수 있다면, (참, 참), (참, 거짓), (거짓, 참), (거짓, 거짓)의 네 조합을 모두 다룬다.

③ O: 조건 커버리지는 **각 개별 조건이 참과 거짓을 최소 한 번씩** 갖도록 한다. 개별 조건의 결과를 검사하는 기준이므로, 조건들의 모든 조합이나 전체 결정 결과의 참·거짓까지 모두 검사하도록 요구하지는 않는다.

④ X: 조건/결정 커버리지는 조건 커버리지와 결정 커버리지를 함께 만족하는 기준이다. 각 개별 조건이 참·거짓을 최소 한 번씩 갖고, 전체 결정 결과도 참·거짓을 최소 한 번씩 가져야 한다. 선지의 '전체 조건식의 결과에 독립적으로 영향을 주도록'은 **변경 조건/결정 커버리지(MC/DC)**의 요구 사항이다. 예를 들어 A AND B에서 A의 독립적 영향을 확인하려면, B를 참으로 고정하고 A만 참에서 거짓으로 바꾸어 전체 결과가 참에서 거짓으로 달라지는지 확인한다. 단순히 개별 조건과 전체 결과가 참·거짓을 모두 가졌다는 것만으로 이러한 독립적 영향까지 입증되지는 않는다.

NoteBase

문 16. 다음은 GoF Iterator 설계 패턴의 클래스 다이어그램이다. 해당 다이어그램에 대한 설명 중에서 옳지 않은 것은?



① 복합 객체 요소의 내부 구조를 외부에 노출하지 않고 순차적으로 접근하기 위한 설계 패턴이다.

② Java 언어의 Vector와 같이 불규칙한 크기의 객체 집합인 자료구조에는 적용할 수 없다.

③ Aggregate 인터페이스는 Client가 Iterator를 생성할 수 있도록 한다.

④ ConcreteAggregate는 적합한 ConcreteIterator의 인스턴스를 반환하기 위한 인스턴스 생성 인터페이스를 구현한다.

정답 2번

간단한 해설

Iterator 패턴은 집합의 크기가 고정되어 있어야만 사용할 수 있는 패턴이 아니다. 크기가 늘거나 줄 수 있는 **Java의 Vector에도 적용할 수 있으며**, Vector는 실제로 iterator()를 통해 순회 객체를 제공한다.

NoteBase

선지별 해설

① O: Iterator 패턴은 **집합의 내부 표현을 노출하지 않고 요소를 순차적으로 접근**하게 한다. 그림에서 Client는 Iterator 인터페이스의 getFirst(), getNext(), hasNext()를 이용하므로, 집합이 내부적으로 요소를 어떻게 저장하는지 알지 못해도 순회할 수 있다.

② X: **집합의 크기가 고정되어 있지 않아도 적용할 수 있다**. Iterator가 현재 순회 위치를 관리하고 다음 요소에 접근하는 기능을 제공하므로, Client가 집합의 크기나 저장 구조에 맞춰 직접 순회 방식을 구현할 필요가 없다. Java의 Vector는 요소 추가·삭제에 따라 크기가 늘거나 줄 수 있는 자료구조이며, iterator()로 요소를 순서대로 순회하는 Iterator를 얻을 수 있다. 따라서 '적용할 수 없다'는 '적용할 수 있다'로 고쳐야 한다.

③ O: Aggregate는 **순회 객체를 얻기 위한 createIterator()를 선언**한다. 그림의 반환형은 Iterator이다. Client는 이 연산을 호출하여 순회 객체를 얻으므로, 구체적인 ConcreteIterator 클래스를 직접 지정해 생성할 필요가 없다.

④ O: ConcreteAggregate는 Aggregate의 createIterator()를 구현하여 **자신의 저장 구조에 적합한 ConcreteIterator 인스턴스를 반환**한다. 그림에서 ConcreteAggregate가 Aggregate 인터페이스를 구현하고 같은 생성 연산을 제공하며, createIterator()는 팩터리 메서드로 표시되어 있다. 실제 반환 객체는 ConcreteIterator이지만 Client는 이를 Iterator 인터페이스로 사용한다.

NoteBase

문 17. 다음 Java 코드에 적용된 디자인 패턴과 Student 클래스와 Member Generator 클래스 간 관계를 옳게 짝 지은 것은?

```
interface Member{ void printType(); }

class Student implements Member {
   public void printType()
      { System.out.println("Student"); }
}

class Staff implements Member {
   public void printType()
      { System.out.println("Staff"); }
}

class Professor implements Member {
   public void printType()
      { System.out.println("Professor"); }
}

class MemberGenerator {
   public Member newMember(String memberType){
      if(memberType.equals("Student"))
         return new Student();
      else if(memberType.equals("Staff"))
         return new Staff();
      else if(memberType.equals("Professor"))
         return new Professor();
      return null;
   }
}
```

① singleton 패턴 - 연관 관계

② singleton 패턴 - 의존 관계

③ factory method 패턴 - 연관 관계

④ factory method 패턴 - 의존 관계

정답 4번

간단한 해설

newMember()가 전달받은 종류에 따라 **객체를 생성해 반환**하므로 팩터리 방식이다. MemberGenerator는 Student 객체를 필드로 보관하지 않고 메서드 안에서 생성하므로, 두 클래스 사이에는 **의존 관계**가 성립한다.

NoteBase

핵심 풀이

디자인 패턴: MemberGenerator의 newMember()는 memberType에 따라 Student·Staff·Professor 중 하나를 생성하고, 공통 인터페이스인 Member 타입으로 반환한다. 호출자는 생성할 구체 클래스의 선택을 이 메서드에 맡긴다. 제시된 선택지에서는 이러한 **객체 생성의 캡슐화**를 factory method 패턴에 대응시킨다. 다만 코드 자체는 조건문으로 생성 대상을 고르는 단순 팩터리 형태이며, 생성 메서드를 하위 클래스에서 재정의하는 엄밀한 GoF Factory Method 구조와는 구별된다. singleton 패턴은 인스턴스를 하나로 제한하여 공유하는 방식이다. 이 코드에서는 newMember()에 'Student'를 전달할 때마다 new Student()가 실행되어 새 객체가 만들어지므로 singleton에 해당하지 않는다.

클래스 간 관계: MemberGenerator는 newMember() 안에서 new Student()를 실행하므로 Student 클래스의 정의를 사용한다. 이처럼 다른 클래스의 객체를 생성하는 데 그 클래스를 필요로 하는 것은 **생성에 따른 의존 관계**이다. 의존 방향은 MemberGenerator에서 Student로 향한다. 연관 관계는 객체들 사이의 구조적 연결이며, 코드에서는 보통 다른 객체의 참조를 필드로 유지하는 형태로 나타난다. 제시된 MemberGenerator에는 Student 객체의 참조를 유지하는 필드가 없으므로, 메서드 안에서 객체를 생성한다는 사실을 연관 관계로 해석하지 않는다.

NoteBase

문 18. 중환자실 환자 모니터링 시스템은 생체신호 이상에 대한 신속한 경보, 환자 정보 보호, 경보 기능의 안전성, 24시간 서비스 가용성, 의료기기 변경에 대한 유지보수성이 중요하다. 이러한 요구사항을 고려한 아키텍처 설계 방안으로 옳은 것은?

① 신속한 처리가 필요한 경보 연산을 다수의 컴포넌트에 분산하여 상호 통신하도록 하고, 안전 관련 기능은 소수의 컴포넌트에 모아 배치한다.

② 신속한 경보에 필요한 핵심 연산은 소수의 컴포넌트에 집중하고, 변경이 잦은 의료기기 연동 기능은 작고 독립적인 컴포넌트로 구성한다.

③ 안전 관련 기능은 가능한 여러 컴포넌트에 분산하고, 중요 환자 정보는 보호 수준이 높은 내부 계층에 배치한다.

④ 높은 가용성을 위해 컴포넌트의 중복을 최소화하고, 유지보수성을 높이기 위해 여러 컴포넌트가 하나의 공유 데이터 구조를 사용하도록 한다.

정답 2번

간단한 해설

경보의 핵심 연산을 **소수의 컴포넌트에 집중**하면 컴포넌트 사이의 통신 부담을 줄여 신속한 처리에 유리하다. 변경이 잦은 의료기기 연동 기능은 **작고 독립적인 컴포넌트**로 분리하면 변경의 영향을 해당 부분으로 제한하기 쉽다.

NoteBase

선지별 해설

① X: 경보 연산을 다수의 컴포넌트에 분산하면 통신과 조정에 드는 시간이 늘어 신속한 경보에 불리할 수 있다. 성능이 중요한 핵심 연산은 **소수의 컴포넌트에 집중하여 통신을 최소화**하는 방향이 적합하다. 안전 관련 기능을 소수의 컴포넌트에 모으는 부분은 안전성 분석과 검증 범위를 제한할 수 있으므로 적절하다.

② O: 신속한 경보에 필요한 연산을 소수의 컴포넌트에 모으면 통신 지연과 호출 부담을 줄일 수 있다. 반면 의료기기 연동처럼 변경이 잦은 기능은 작고 독립적으로 구성하여 **다른 부분에 미치는 변경 영향을 줄인다**. 예를 들어 특정 의료기기의 통신 규격이 바뀌어도 해당 연동 컴포넌트의 수정으로 대응하도록 설계할 수 있다. 두 방안은 각각 성능과 유지보수성 요구에 부합한다.

③ X: 안전 관련 기능을 가능한 여러 컴포넌트에 분산하면 기능 사이의 상호작용까지 분석해야 하므로 안전성 검증이 복잡해진다. 따라서 안전에 중요한 기능은 **하나 또는 소수의 컴포넌트에 모아** 분석과 검증을 용이하게 하는 것이 적합하다. 중요 환자 정보를 보호 수준이 높은 내부 계층에 배치하는 부분은 외부의 직접 접근을 제한하는 계층형 보안 설계에 부합한다.

④ X: 높은 가용성은 장애가 발생해도 서비스를 계속 제공하는 능력이다. 이를 위해서는 컴포넌트의 중복을 최소화하기보다 **중복 구성과 장애 대처 기능**을 두어 일부 고장을 견디도록 해야 한다. 또한 여러 컴포넌트가 하나의 데이터 구조에 직접 의존하면 그 구조의 변경이 여러 부분으로 퍼질 수 있다. 유지보수성을 높이려면 컴포넌트를 작고 교체 가능하게 구성하고, 내부 데이터 구조에 대한 의존을 줄여야 한다.

NoteBase

문 19. 최근 소프트웨어 유지보수 및 운영 환경에서 필수적으로 적용되는 지속적 통합(CI), 지속적 배포(CD) 파이프라인과 데브옵스(DevOps) 환경에 대한 설명으로 옳지 않은 것은?

① 지속적 통합은 다수의 개발자가 작성한 소스 코드를 통합하여 소프트웨어 제품 형상을 개발하고 검증하는 프로세스를 말한다.

② 지속적 배포 단계에서 릴리스는 서버에 자원을 할당하고 인프라를 구축하여 최종 사용자에게 서비스를 오픈하는 단계를 의미하며, 설치는 배포 가능한 소프트웨어 패키지를 작성하는 단계를 의미한다.

③ 데브옵스 환경에서는 코드의 품질을 유지하고 최상의 품질을 배포할 수 있도록 자동화 테스트 도구를 활용한다.

④ 데브옵스에서는 소프트웨어가 정확히 동작하고 있는지 지속적으로 모니터링해야 하며, 모니터링 과정에서 발견된 문제에 대해 신속하게 대응한다.

정답 2번

간단한 해설

②는 릴리스와 설치의 설명을 서로 바꾸었다. 릴리스는 **배포 가능한 소프트웨어 패키지를 준비**하는 과정이고, 설치는 그 패키지를 **운영 환경에 배치·설정하여 실행할 수 있게 하는** 과정이다.

선지별 해설

① O: 지속적 통합(CI)은 여러 개발자의 코드 변경을 공통 저장소에 자주 통합하고, **자동 빌드와 테스트로 통합 결과를 검증**하는 방식이다. 각자의 코드가 따로 동작하더라도 합친 뒤에는 오류가 생길 수 있으므로, 통합된 소프트웨어를 반복해서 빌드하고 검사하여 문제를 조기에 발견한다.

② X: 릴리스와 설치에 해당하는 작업을 반대로 설명했다. 릴리스는 실행 파일과 필요한 구성 요소 등을 묶어 **배포 가능한 패키지를 준비**하는 과정이다. 설치는 준비된 패키지를 대상 서버에 배치하고 자원을 할당하거나 실행 환경을 설정하여 **서비스를 운영할 수 있게 하는** 과정이다. 패키지를 만드는 일과 그 패키지를 실제 운영 환경에서 실행하도록 구성하는 일을 구분해야 한다.

③ O: 데브옵스에서는 반복적인 코드 변경을 빠르게 검증하기 위해 자동화 테스트 도구를 활용한다. 단위 테스트와 통합 테스트 등을 반복 실행하면 **기존 기능이 변경 후에도 유지되는지** 확인하고, 결함이 있는 변경 사항을 배포 전에 발견하여 품질을 관리할 수 있다.

④ O: 배포 전 테스트를 통과했더라도 실제 운영 환경의 부하나 사용 조건에서 문제가 발생할 수 있다. 따라서 응답 시간, 오류 발생, 자원 사용량 등을 **지속적으로 모니터링**하고, 이상이 발견되면 원인을 분석하여 수정하거나 이전 버전으로 되돌리는 등 신속하게 대응한다.

NoteBase

문 20. 다음 Java 코드의 조건 커버리지(condition coverage)를 가장 크게 만족하는 테스트 케이스는? (단, 단축 연산(short-circuit evaluation)은 수행하지 않는다)

```
void method(int x, int y, int z) {
   if (x > 20 && y == 5) {
      x = 10;
      z = 5;
   }
   if (x == 10 || z % 2 == 0) {
      z += 10;
   }
}
```

① (x=10, y=10, z=4), (x=5, y=5, z=3)

② (x=25, y=5, z=3), (x=8, y=5, z=3)

③ (x=10, y=10, z=4)

④ (x=25, y=5, z=3), (x=10, y=5, z=3), (x=5, y=5, z=1)

정답 1번

간단한 해설

조건 커버리지는 **각 기본 조건이 참과 거짓을 모두 갖는지**를 본다. ①의 두 입력은 x > 20의 참만 빠뜨리고 나머지 결과를 모두 충족하므로, 조건 커버리지가 87.5%로 가장 높다.

NoteBase

선지별 해설

조건 커버리지는 전체 if 조건식의 결과가 아니라, 그 안의 각 기본 조건에서 참과 거짓이 각각 실행되었는지를 측정한다. 이 코드의 기본 조건을 A: x > 20, B: y == 5, C: x == 10, D: z % 2 == 0으로 두면 확인할 결과는 총 4 × 2 = 8개이다. D는 z가 짝수인지 확인하는 조건이다. 문제에서 단축 연산을 하지 않으므로 &&와 ||의 양쪽 조건을 모두 평가한다. 또한 C와 D에는 **첫 if문을 실행한 뒤의 x와 z**를 사용한다. 아래 입력은 (x, y, z) 순서이며, T는 참, F는 거짓이다.

① (10, 10, 4)의 (A, B, C, D)는 (F, F, T, T)이고, (5, 5, 3)은 (F, T, F, F)이다. 두 입력 모두 첫 if문의 본문을 실행하지 않으므로 x와 z는 입력값 그대로이다. A는 거짓만, **B·C·D는 참과 거짓을 모두** 실행한다. 따라서 충족한 결과는 1 + 2 + 2 + 2 = 7개이며, 조건 커버리지는 $\frac{7}{8} \times 100 = 87.5\%$이다.

② (25, 5, 3)은 A와 B가 모두 참이므로 첫 if문의 본문에서 **x = 10, z = 5로 바뀐다**. 따라서 C는 참, D는 거짓으로, (A, B, C, D)는 (T, T, T, F)이다. (8, 5, 3)은 첫 if문의 본문을 실행하지 않아 (F, T, F, F)이다. A와 C는 참·거짓을 모두 실행하지만 B는 참만, D는 거짓만 실행한다. 충족한 결과는 2 + 1 + 2 + 1 = 6개이며, 조건 커버리지는 $\frac{6}{8} \times 100 = 75\%$이다.

③ (10, 10, 4) 하나만 실행하므로 (A, B, C, D)는 (F, F, T, T)이다. 첫 if문의 본문은 실행되지 않는다. 각 기본 조건에서 **한쪽 결과만** 확인하므로 충족한 결과는 4개이며, 조건 커버리지는 $\frac{4}{8} \times 100 = 50\%$이다.

④ (25, 5, 3)은 첫 if문의 본문에서 x = 10, z = 5로 바뀌므로 (A, B, C, D)가 (T, T, T, F)이다. (10, 5, 3)은 (F, T, T, F), (5, 5, 1)은 (F, T, F, F)이다. A와 C는 참·거짓을 모두 실행하지만, **B는 계속 참이고 D는 계속 거짓**이다. 입력이 세 개여도 새로운 조건 결과를 더 충족하지 못하므로, 충족한 결과는 6개이고 조건 커버리지는 $\frac{6}{8} \times 100 = 75\%$이다.

조건 커버리지는 ① 87.5%, ② 75%, ③ 50%, ④ 75%이므로, 가장 큰 값을 갖는 테스트 케이스 묶음은 ①이다.

NoteBase

문 21. 다음 제시된 Java 코드에 포함된 코드 스멜과 이를 해결하기 위한 리팩토링 기법에 대한 설명으로 옳지 않은 것은?

```
public class Employee {
   public static final int EMP1 = 0;
   public static final int EMP2 = 1;
   public static final int EMP3 = 2;
   private int empType;

   public Employee(int empType) {
      this.empType = empType;
   }

   public int paySalary() {
      switch (empType) {
         case EMP1:
            return 5000;
         case EMP2:
            return 6000;
         case EMP3:
            return 9000;
         default:
            throw new IllegalArgumentException(
               "Invalid employee type");
         }
   }
}
```

① 새로운 조건이 생길 때마다 switch 문을 수정해야 하므로 버그 발생 가능성이 증가한다.

② 코드 스멜 해결을 위해 empType을 서브클래스로 분리하고, paySalary() 메서드를 다형성으로 구현하는 리팩토링 기법을 사용한다.

③ 주어진 코드는 객체지향 설계원리 SOLID의 OCP(개방-폐쇄 원칙)를 위반한다.

④ 주어진 코드에는 높은 결합도로 인해 변경에 취약한 유사 데이터들의 그룹 중복(data clumps) 코드 스멜이 포함된다.

정답 4번

간단한 해설

④의 data clumps는 **함께 쓰이는 여러 데이터 항목이 여러 곳에서 묶음으로 반복되는 현상**을 뜻한다. 제시된 코드의 문제는 데이터 묶음의 중복이 아니라, 직원 유형 코드에 따라 급여 처리를 나누는 **switch 기반 조건 분기**이다.

NoteBase

선지별 해설

① O: paySalary()는 empType을 검사하여 EMP1·EMP2·EMP3별로 서로 다른 급여를 반환한다. 새로운 직원 유형과 그 급여 규칙을 추가하려면 **기존 switch 문에 case를 추가**해야 한다. 이처럼 기능을 늘릴 때 기존 메서드를 계속 수정하면 분기 누락이나 잘못된 급여 반환 등의 오류가 생길 가능성이 커진다.

② O: empType으로 구분하던 직원 유형을 Employee의 하위클래스로 만들고, 각 하위클래스에서 **paySalary()를 재정의**할 수 있다. 예를 들어 EMP1에 해당하는 하위클래스는 5000을, EMP2에 해당하는 하위클래스는 6000을 반환하도록 한다. 호출자는 같은 paySalary()를 호출하고, 실제 객체의 유형에 맞는 구현이 실행된다. 이는 '타입 코드를 서브클래스로 전환'하고 '조건문을 다형성으로 전환'하는 리팩토링이다.

③ O: OCP는 소프트웨어 구성 요소가 **확장에는 열려 있고 수정에는 닫혀 있어야 한다**는 원칙이다. 제시된 구조에서는 직원 유형을 추가하는 확장을 위해 기존 Employee의 paySalary()를 수정해야 하므로 이 원칙을 위반한다. 유형별 동작을 하위클래스에 분리하면 새로운 유형의 급여 구현을 추가하면서 기존 급여 분기 코드를 수정하는 일을 줄일 수 있다.

④ X: data clumps는 서로 관련된 여러 데이터 항목이 여러 클래스의 필드나 메서드의 매개변수 목록 등에 함께 반복되는 코드 스멜이다. 제시된 EMP1·EMP2·EMP3는 empType이 가질 수 있는 유형별 상수이며, 함께 전달되는 데이터 묶음의 중복을 보여 주지 않는다. 이 코드의 문제는 **타입 코드에 따라 동작을 나누는 switch 조건 분기**로 설명해야 한다. data clumps라면 반복되는 데이터 묶음을 하나의 객체로 묶는 방식으로 개선하지만, 여기서는 유형별 동작을 다형성으로 분리하는 방식이 적합하다.

NoteBase

문 22. 『SW사업 대가산정 가이드(2025년)』 및 『소프트웨어사업 계약 및 관리감독에 관한 지침(2023년)』에 따라서 지능형 냉난방 조절 시스템 개발에 대한 평균 노력을 추정하고자 한다. 해당 시스템의 기능별 개수 및 평균 복잡도와 사업 규모별 1인 생산성이 <표 1> 및 <표 2>와 같이 주어졌을 때 평균 노력(MM)을 계산한 결과로 옳은 것은? (단, 필요한 보정 계수는 1.5로 계산한다)

<표 1> 기능별 개수 및 평균 복잡도

기능 유형	개수	평균 복잡도
외부입력	15	4.0
외부출력	5	5.2
외부조회	12	3.9
내부 논리 파일	4	7.5
외부 인터페이스 파일	2	5.4

<표 2> 사업 규모별 1인 생산성

사업규모(FP)	1인 생산성(FP/MM)
1,000 미만	19
1,000 이상 ~ 2,000 미만	22
2,000 이상 ~ 3,000 미만	24
3,000 이상	22

① 약 0.25 MM

② 약 9.14 MM

③ 약 13.71 MM

④ 약 17.27 MM

정답 3번

간단한 해설

기능별 개수에 평균 복잡도를 곱해 합산하면 173.6 FP이다. 이를 **1인 생산성 19 FP/MM으로 나누고 보정 계수 1.5를 곱하면** 평균 노력은 약 13.71 MM이다.

NoteBase

계산 과정

기능점수(FP)는 개발할 소프트웨어의 기능 규모를 나타낸다. <표 1>에서 각 기능의 개수에 평균 복잡도를 곱하고, 그 결과를 합산하여 전체 기능점수를 구한다.

외부입력: 15 × 4.0 = 60 FP이다.

외부출력: 5 × 5.2 = 26 FP이다.

외부조회: 12 × 3.9 = 46.8 FP이다.

내부 논리 파일: 4 × 7.5 = 30 FP이다.

외부 인터페이스 파일: 2 × 5.4 = 10.8 FP이다.

전체 기능점수: 60 + 26 + 46.8 + 30 + 10.8 = 173.6 FP이다.

생산성 적용: 173.6 FP는 <표 2>의 **1,000 FP 미만** 구간에 해당하므로 1인 생산성은 19 FP/MM이다. 이는 1 MM의 노력으로 19 FP를 개발한다는 뜻이다. MM은 한 사람이 한 달 동안 투입하는 작업량을 기준으로 한 노력의 단위이다.

평균 노력 계산: 기능점수를 1인 생산성으로 나누면 필요한 노력이 MM 단위로 산출된다. 여기에 문제에서 지정한 **보정 계수 1.5를 곱한다**. 보정 전 노력은 $\frac{173.6}{19} \approx 9.14$ MM이고, 보정 후 평균 노력은 $\frac{173.6}{19} \times 1.5 \approx 13.71$ MM이다. 중간값을 반올림하지 않고 계산한 뒤 최종 결과를 소수 둘째 자리까지 나타낸 값이다.

따라서 평균 노력은 약 13.71 MM으로, ③의 값과 일치한다.

NoteBase

문 23. CMMI에 대한 설명 중 옳지 않은 것은?

① 조직의 프로세스를 개선하기 위한 프레임워크로 소프트웨어 개발, 시스템 엔지니어링, 서비스 관리 등 다양한 도메인에서 활용될 수 있다.

② CMMI 수준 평가는 성숙도 수준(maturity level)과 역량 수준(capability level)의 두 가지 수준을 평가하는 방법으로 나뉜다.

③ 정의(defined) 수준에는 조직을 위한 표준 프로세스는 없고, 프로젝트별 프로세스만 있다.

④ 협력 업체 관리는 성숙도 수준에서 관리(managed) 수준에 속하는 프로세스 영역이다.

정답 3번

간단한 해설

정의(Defined) 수준은 **조직 차원의 표준 프로세스를 갖추는** 성숙도 3수준이다. 각 프로젝트는 이 표준을 프로젝트 특성에 맞게 조정하여 사용하므로, '조직을 위한 표준 프로세스는 없다'는 설명이 잘못되었다.

선지별 해설

① O: CMMI는 조직의 업무 수행 과정을 평가하고 개선하기 위한 프레임워크이다. 소프트웨어 개발뿐 아니라 시스템 엔지니어링, 서비스 관리 등에도 적용할 수 있으며, 공통적으로 **프로세스의 역량과 조직의 성숙도를 높이는 것**을 지향한다.

② O: 단계적 표현에서는 여러 프로세스 영역을 묶어 **조직의 전반적인 성숙도**를 평가하고, 연속적 표현에서는 **개별 프로세스 영역의 역량**을 평가한다. 전자는 성숙도 수준(maturity level), 후자는 역량 수준(capability level)을 사용하므로 두 평가 관점은 대상과 범위가 다르다.

③ X: 정의(Defined)는 성숙도 3수준으로, **조직 차원의 표준 프로세스가 수립되어 있는** 단계이다. 프로젝트는 조직 표준을 정해진 조정 지침에 따라 자신의 특성에 맞게 조정하여 사용한다. 따라서 '조직을 위한 표준 프로세스는 없고, 프로젝트별 프로세스만 있다'는 설명은 조직 표준을 기반으로 한다는 특징을 반대로 서술한 것이다. 프로젝트마다 프로세스와 절차가 상당히 다를 수 있는 관리(Managed) 2수준과 구별해야 한다.

④ O: 협력 업체 관리에 해당하는 공급자 계약 관리(Supplier Agreement Management, SAM)는 **관리(Managed) 2수준**의 프로세스 영역이다. 공급자로부터 제품이나 서비스를 확보하는 과정을 관리하며, 공급자와의 계약 수립 및 계약 이행 관리 등이 포함된다.

NoteBase

문 24. 소프트웨어 품질 평가모델에 대한 설명으로 옳지 않은 것은?

① ISO/IEC 9126은 내부품질/외부품질 및 사용품질로 구성되어 있으며 내부품질/외부품질 특성으로는 기능성, 사용성, 효율성, 신뢰성, 이식성, 유지보수성이 있다.

② ISO/IEC 25000 시리즈는 ISO/IEC 9126과 ISO/IEC 14598 시리즈를 통합한 프레임워크로 품질 모델 영역, 품질 측정 영역 등 4개 영역으로 구성된다.

③ ISO/IEC 14598은 소프트웨어의 개발 과정 또는 완성된 제품에 대해 공급자와 구매자가 객관적으로 품질을 평가할 수 있도록 절차와 방법을 정의한 국제표준으로 평가지원 부분과 평가 프로세스로 구성한다.

④ ISO/IEC 12119는 테스트의 절차와 결과의 일관성을 확보하기 위한 지침을 제공하며 패키지 소프트웨어의 품질 요구사항을 정의한다.

정답 2번

간단한 해설

②의 '4개 영역'은 **5개 영역**으로 고쳐야 한다. ISO/IEC 25000 시리즈의 기본 구성은 품질 관리, 품질 모델, 품질 측정, 품질 요구사항, 품질 평가이다.

선지별 해설

① O: ISO/IEC 9126은 품질 모델을 내부품질·외부품질과 사용품질로 구분한다. 내부품질과 외부품질에는 기능성, 사용성, 효율성, 신뢰성, 이식성, 유지보수성이라는 **동일한 6개 품질 특성**을 적용한다. 내부품질은 소프트웨어 자체의 속성을, 외부품질은 실행 시 나타나는 동작을 평가하며, 사용품질은 특정 사용 환경에서 사용자가 얻는 효과를 평가한다.

② X: ISO/IEC 25000 시리즈인 SQuaRE는 ISO/IEC 9126의 품질 모델·측정과 ISO/IEC 14598의 품질 평가를 통합한 체계이다. 다만 기본 구성은 4개가 아니라 **5개 영역**이다. 품질 관리(2500n), 품질 모델(2501n), 품질 측정(2502n), 품질 요구사항(2503n), 품질 평가(2504n)로 나뉜다. 품질 관리 영역은 다른 영역에서 공통으로 쓰는 모델·용어와 품질 요구사항 및 평가 활동의 관리·지원 기준을 제공한다.

③ O: ISO/IEC 14598은 개발 중인 소프트웨어와 완성된 제품의 품질을 객관적으로 평가하기 위한 절차와 방법을 다룬다. 평가지원 부분은 평가의 계획·관리와 평가 모듈 문서화 등을 지원하고, 평가 프로세스는 개발자·구매자·평가자가 수행할 평가 활동을 규정한다. ISO/IEC 9126이 품질을 판단할 특성을 제시한다면, ISO/IEC 14598은 **그 품질을 어떻게 평가할지**에 초점을 둔다.

④ O: ISO/IEC 12119는 **패키지 소프트웨어의 품질 요구사항과 시험 지침**을 다룬다. 제공·납품되는 소프트웨어 패키지가 요구사항을 충족하는지 시험하는 기준을 제시하므로 시험 절차와 결과의 일관성을 확보하는 데 쓰인다. 소프트웨어의 생산 과정이나 공급자의 품질시스템 자체를 평가하는 표준과는 적용 대상이 다르다.

NoteBase

문 25. 어느 대규모 소프트웨어 개발 프로젝트의 작업 항목, 소요 기간, 선행 작업 관계가 다음 표와 같다. 이 프로젝트의 일정 계획 및 임계 경로 (critical path) 분석에 대한 설명으로 옳지 않은 것은? (단, 모든 작업은 종료 후 시작(FS, Finish to Start) 관계로 연결되어 있다)

작업	소요 기간	선행 작업
A	4일	없음
B	3일	A
C	6일	A
D	4일	A
E	6일	B
F	4일	C, D
G	5일	E, F
H	3일	D
I	4일	G, H

① 프로젝트의 임계 경로는 A→C→F→G→I이며, 프로젝트의 최소 완료 기간은 23일이다.

② 만약 작업 B가 2일 지연된다면 전체 프로젝트의 최소 완료 기간은 24일로 늘어나고 임계 경로가 변경된다.

③ 작업 D가 3일 지연되더라도 전체 프로젝트의 최소 완료 기간은 지연되지 않는다.

④ 작업 H는 이 프로젝트의 모든 작업 중 가장 많은 8일의 여유 시간을 확보하고 있다.

정답 3번

간단한 해설

작업 D의 여유 시간은 **2일**이므로 3일 지연되면 여유를 1일 초과한다. 이에 따라 F·G·I의 완료도 1일씩 늦어져 프로젝트의 최소 완료 기간은 **24일**이 된다.

NoteBase

선지별 해설

프로젝트 시작을 0일 시점으로 두고 계산한다. FS 관계에서는 선행 작업이 여러 개이면 모두 끝난 뒤에 후속 작업을 시작할 수 있다. 임계 경로는 시작부터 완료까지 소요 기간이 가장 긴 경로이며, 이 경로의 작업은 여유 시간이 0일이다. 여기서 여유 시간은 다른 조건이 같을 때 프로젝트 완료를 늦추지 않고 해당 작업을 지연할 수 있는 기간인 총여유 시간을 뜻한다.

① O: A는 4일 시점에 끝나므로 B·C·D의 가장 이른 완료 시점은 각각 7일·10일·8일이다. E는 B가 끝난 뒤 6일을 수행하여 13일에 끝난다. F는 **C와 D가 모두 끝나는 10일**에 시작하여 14일에 끝나고, G는 E와 F 중 늦게 끝나는 F를 기다려 14일에 시작하여 19일에 끝난다. H는 D가 끝난 8일에 시작하여 11일에 끝나므로 I는 G가 끝나는 19일에 시작하여 23일에 끝난다. 따라서 임계 경로는 A→C→F→G→I이고, 소요 기간은 4 + 6 + 4 + 5 + 4 = 23일이다.

② O: B가 2일 지연되면 완료 시점이 7일에서 9일로 늦어지고, E의 완료 시점도 13일에서 15일로 늦어진다. F는 원래대로 14일에 끝나지만 G는 **E가 끝나는 15일까지 기다려야 한다**. 따라서 G는 20일, I는 24일에 끝나며, 임계 경로는 A→B→E→G→I로 바뀐다. 이 경로의 원래 소요 기간은 22일이므로 B의 2일 지연 중 1일은 여유로 흡수되고 나머지 1일이 전체 완료를 늦춘다.

③ X: 원래 F는 C가 끝나는 10일에 시작하며, D는 8일에 끝나므로 D의 여유 시간은 10 − 8 = 2일이다. D가 3일 지연되면 완료 시점이 11일이 되어 F의 시작도 11일로 늦어진다. 이후 F는 15일, G는 20일, I는 24일에 끝난다. 따라서 '최소 완료 기간은 지연되지 않는다'는 설명은 **'최소 완료 기간이 1일 늘어나 24일이 된다'**로 고쳐야 한다.

④ O: H는 D가 끝나는 8일에 시작하여 11일에 끝난다. 후속 작업 I는 G를 기다려 19일에 시작하므로 H는 19일까지 끝나면 되며, 가장 늦게 시작할 수 있는 시점은 19 − 3 = 16일이다. 따라서 H의 총여유 시간은 **16 − 8 = 8일**이다. 다른 작업의 총여유 시간은 B·E가 각각 1일, D가 2일, A·C·F·G·I가 각각 0일이므로 H가 가장 크다.

NoteBase