[소프트웨어 설계>애플리케이션 설계>공통 모듈 설계]

[기출 예상 문제]

1. 공통 모듈 명세 작성 원칙에 대한 설명으로 가장 적절한 것은?

① 추적성: 시스템의 구현 시 요구사항과 필요한 모든 것을 기술한다.
② 일관성: 해당 기능에 대한 일관된 이해와 하나로 해석될 수 있도록 작성한다.
③ 완전성: 해당 기능에 대한 요구사항의 출처와 관련 시스템 등 유기적 관계에 대한 식별이 가능하도록 작성한다.
④ 정확성: 실제 시스템 구현 시 필요한 기능인지 여부를 알 수 있도록 정확하게 작성한다.

[기출 예상 문제]

2. 모듈화에 대한 설명으로 가장 거리가 먼 것은?

① 모듈의 크기가 너무 작은 경우 모듈의 개수가 많아지므로 모듈 간의 통합 비용이 과다하게 발생할 수 있다.
② 응집도가 높을수록 필요한 요소들로 구성되어 지고 낮을수록 요소들 간의 관련성이 적은 요소들로 구성되어 진다.
③ 모듈의 크기가 작을수록 하나의 모듈을 개발하는데 소요되는 비용이 줄어든다.
④ 결합도는 모듈 내부의 구성 요소간 관계의 밀접 정도로 평가된다.

[이전 기출 문제]

3. 모듈 내부의 응집도는 가장 약한 정도(가장 바람직 하지 못한 경우)에서 가장 강한 정도(가장 바람직한 경우)에 이르기까지 나열할 수 있다. 다음 중 모듈 내부의 응집도가 약한 정도에서 강한 정도 순으로 가장 적절하게 나열한 것은?

① coincidental cohesion – logical cohesion – temporal cohesion – communicational cohesion – procedural cohesion – sequential cohesion – functional cohesion
② coincidental cohesion – logical cohesion – temporal cohesion – procedural cohesion – communicational cohesion – sequential cohesion – functional cohesion
③ coincidental cohesion – logical cohesion – sequential cohesion – communicational cohesion – procedural cohesion – temporal cohesion – functional cohesion
④ coincidental cohesion – logical cohesion – temporal cohesion – sequential cohesion – communicational cohesion – procedural cohesion – functional cohesion

[소프트웨어 설계>애플리케이션 설계>공통 모듈 설계]

[이전 기출 문제]

4. 모듈화에 대한 설명으로 가장 거리가 먼 것은?

① 항상 연관되어 발생하는 조건식에 대한 입력 값의 검증은 별도의 공통 모듈로 구성한다.
② 모듈화는 프로그램이 효율적으로 관리될 수 있도록 시스템을 분해하고 추상화하는 기법이다.
③ 공통 모듈로 구성할 경우 프로젝트의 재사용성을 향상시킬 수 있다.
④ 모듈의 크기가 작아 모듈 개수가 많아지면 모듈 하나를 개발하는 데 비용이 커진다.

[이전 기출 문제]

5. <보기>는 모듈화를 중심으로 한 소프트웨어 설계방법에 대한 설명이다. 빈칸의 내용을 올바르게 나열한 것은?

- 결합도(coupling)와 응집도(cohesion)는 모듈의 (ㄱ)을 판단하는 기준이다.
- 결합도란 모듈 (ㄴ)의 관련성을 의미하며, 응집도란 모듈 (ㄷ)의 관련성을 의미한다.
- 좋은 설계를 위해서는 결합도는 (ㄹ), 응집도는 (ㅁ) 방향으로 설계해야 한다.

	ㄱ	ㄴ	ㄷ	ㄹ	ㅁ
①	독립성	사이	내부	작게	큰
②	독립성	내부	사이	크게	작은
③	추상성	사이	내부	작게	큰
④	추상성	내부	사이	크게	작은

[이전 기출 문제]

6. 소프트웨어 모듈 평가 기준으로 판단할 때, 다음 4명 중 가장 좋게 설계한 사람과 가장 좋지 않게 설계한 사람을 순서대로 바르게 나열한 것은?

- 철수: 절차적 응집도 + 공통 결합도
- 영희: 우연적 응집도 + 내용 결합도
- 동수: 기능적 응집도 + 자료 결합도
- 민희: 논리적 응집도 + 스탬프 결합도

① 철수, 영희 ② 철수, 민희
③ 동수, 영희 ④ 동수, 민희

[이전 기출 문제]

7. 응집도의 종류 중 서로간에 어떠한 의미 있는 연관관계도 지니지 않은 기능요소로 구성되는 경우이며, 서로 다른 기능을 수행하는 경우의 응집도는?

① Coincidental Cohesion
② Functional Cohesion
③ Sequential Cohesion
④ Logical Cohesion

[소프트웨어 설계>애플리케이션 설계>공통 모듈 설계]

[이전 기출 문제]
8. 모듈 결합도(module coupling)에 대한 설명으로 옳지 않은 것은?

① 모듈 결합도란 두 모듈간의 상호 의존도를 측정하는것 으로서 좋은 설계가 이루어지도록 하기 위해서는 가능한 한 모듈을 독립적으로 생성한다.
② 데이터 결합(data coupling)은 모듈 간에 매개변수를 통해서만 의사소통을 하도록 하여 다른 모듈에게 불필요한 데이터는 전송하지 않도록 한다.
③ 스템프 결합(stamp coupling)은 두 모듈이 동일한 자료 구조를 조회하는 경우의 결합성이다.
④ 모듈 결합도에서 가장 바람직한 결합도는 내용 결합도(content coupling)이다.

[이전 기출 문제]
9. 두 모듈이 동일한 자료구조를 조회하는 경우의 결합성이며 자료구조의 어떠한 변화, 즉 포맷이나 구조의 변화는 그것을 조회하는 모든 모듈 및 변화되는 필드를 실제로 조회하지 않는 모듈에까지도 영향을 미치게 되는 결합성은?

① data coupling ② stamp coupling
③ control coupling ④ content coupling

[이전 기출 문제]
10. 모듈의 응집도에 대한 설명으로 <보기>에서 옳은 것만을 모두 고른 것은?

ㄱ. 모듈 내 한 구성 요소의 출력이 다른 구성 요소의 입력이 되는 경우는 순차적 응집도(sequential cohesion)에 해당한다.
ㄴ. 모듈 내 구성 요소들이 서로 다른 기능을 같은 시간대에 함께 실행하는 경우는 우연적 응집도(coincidental cohesion)에 해당한다.
ㄷ. 모듈이 여러 가지 기능을 수행하며 모듈 내 구성 요소들이 같은 입력 자료를 이용하거나 동일 출력 데이터를 만들어내는 경우는 통신적 응집도(communicational cohesion)에 해당한다.

① ㄱ ② ㄴ
③ ㄱ, ㄷ ④ ㄴ, ㄷ

[기출 예상 문제]
11. 소프트웨어 설계 프로세스 중 요구 명세의 분류와 관련 없는 것은?

① 기능적 요구 ② 형태적 요구
③ 보안상 요구 ④ 품질적 요구

[소프트웨어 설계>애플리케이션 설계>공통 모듈 설계]

[기출 예상 문제]
12. 설계 모델의 행위 모델은 시스템 행위와 컴포넌트 행위로 분류할 수 있다. 다음 중 컴포넌트 행위와 관련 없는 것은?

① 계산 ② 컴포넌트 제어
③ 실행제어 ④ 상호작용

[기출 예상 문제]
13. 다음 중 설계 모델을 구조 모델과 행위 모델로 분류한 설명으로 가장 거리가 먼 것은?

① 구조 모델은 시스템의 구성 요소들의 기능적 특성에 대한 모델링이다.
② 구조 모델의 구성요소에는 프로시저, 데이터구조, 모듈, 파일구조가 있다.
③ 행위 모델은 상태전이, 데이터 흐름경로, 사건발생 순서를 모델링한다.
④ 행위 모델은 시스템의 구성 요소들이 언제 어떠한 순서로 수행되는가와 같은 동적 특성들의 모델링이다.

[기출 예상 문제]
14. 인터페이스, 데이터 구조, 알고리즘 구조와 관련된 설계 모델 구성 요소로 가장 적절한 것은?

① 시스템 구조 ② 시스템 행위
③ 컴포넌트 구조 ④ 컴포넌트 행위

[기출 예상 문제]
15. 빈칸의 내용을 올바르게 나열한 것은?

(ㄱ)은(는) 소프트웨어를 구성하는 컴포넌트들의 유형, 인터페이스, 내부 설계 구조 및 이들의 상호 연결 구조를 모델링하는 것이다. (ㄴ)은(는) 소프트웨어의 구성 요소들의 기능들과 이들이 언제, 어떠한 순서로 기능을 수행하고 상호작용하는지를 모델링하는 것이다.

	(ㄱ)	(ㄴ)
①	행위 모델링	구조 모델링
②	내부 모델링	외부 모델링
③	외부 모델링	내부 모델링
④	구조 모델링	행위 모델링

[기출 예상 문제]
16. 다음 중 설계 모듈의 동적(Dynamic)요소와 정적(Static)요소에 대한 설명으로 가장 거리가 먼 것은?

① 구조 모델의 정적 요소에는 구성 요소들의 상호 작용 채널이 있다.
② 상호작용 프로토콜, 상호작용 실행 경로는 행위 모델에 속한다.
③ 동적 생성 및 소멸, 동적 결합/연결은 행위 모델의 동적 요소이다.
④ 행위 모델의 정적 요소에는 입력/출력 데이터가 있다.

[소프트웨어 설계>애플리케이션 설계>공통 모듈 설계]

[기출 예상 문제]
17. 다음 중 설계 모듈 요소에 대한 내용으로 가장 거리가 먼 것은?

① 구조 모델 정적 요소: 구성 요소들의 배열, 결합 관계
② 구조 모델 동적 요소: 상태 전이
③ 행위 모델 정적 요소: 입출력 맵핑
④ 행위 모델 동적 요소: 알고리즘

[기출 예상 문제]
18. 다음 중 구조(Structure) 모델링과 행위(Behavior) 모델링에 대한 설명으로 가장 거리가 먼 것은?

① 구조 모델링은 시스템의 구성 요소들과 이들 사이의 구조적인 관계와 특성들의 모델링이다.
② 행위 모델링은 각 구성 요소들의 기능적 특성에 대한 모델링이다.
③ 구조 모델은 입/출력 데이터, 데이터 흐름, 데이터 변환, 데이터 저장 등을 모델링한다.
④ 행위 모델은 시스템 행위와 컴포넌트 행위로 분류 될 수 있다.

[기출 예상 문제]
19. 다음은 무엇에 대한 설명인가?

> 소프트웨어 시스템의 구조를 비롯한 시스템 개발에 중요한 영향을 미치는 결정들로, 소프트웨어 시스템 개발에서 특정 시스템에 대하여 요구되는 기능과 품질을 확보하고 또한 소프트웨어 시스템의 구축 및 지속적인 개선이 용이하도록 하는 역할을 한다.

① 소프트웨어 인터페이스(Software Interface)
② 소프트웨어 설계(Software Design)
③ 소프트웨어 요구사항(Software Requirement)
④ 소프트웨어 아키텍처(Software Architecture)

[기출 예상 문제]
20. 좋은 소프트웨어를 만들기 위한 내용으로 가장 거리가 먼 것은?

① 전체적인 구조가 유기적으로 잘 구성되어야 한다.
② 요구 분석, 설계 단계에서부터 품질 특성을 고려하여 개발해야 한다.
③ 소프트웨어의 단순성의 문제를 해결해야 한다.
④ 개발할 소프트웨어의 전체 구조를 생각하여 소프트웨어 아키텍처를 설계한다.

[소프트웨어 설계>애플리케이션 설계>공통 모듈 설계]

[기출 예상 문제]

21. 다음 중 아키텍처 설계 절차와 적용 원리에 대한 설명으로 가장 거리가 먼 것은?

① 품질속성 설계전술: 단일 품질속성 응답을 제어하는 데 영향력 있는 설계의 결정이다.
② 품질속성, 검증가능성, 품질속성시나리오: 아키텍처 드라이버을 먼저 검증할 수 있는 형태로 바꾸어 놓은 것이다.
③ 아키텍처 설계 절차: 아키텍처를 결정하는 요구사항들로부터 먼저 어떤 아키텍처적인 설계의 선택들이 있는가를 판단한다.
④ 설계의 일반원리: 반복적으로 발생하는 문제에 대해 미리 만들어진 솔루션이다.

[기출 예상 문제]

22. 아키텍처 설계 절차와 적용 원리 순서로 알맞은 것은?

ㄱ.아키텍처 드라이버	ㄴ.설계의 일반 원리
ㄷ.아키텍처 패턴	ㄹ.아키텍처 설계 절차
ㅁ.아키텍처의 분석	ㅂ.품질속성 설계 전술
ㅅ.컴포넌트와 커넥터	ㅇ.아키텍처 스타일

① ㄷ-ㅅ-ㅂ-ㅁ-ㄴ-ㅇ-ㄱ-ㄹ
② ㄱ-ㅅ-ㅇ-ㄴ-ㄹ-ㄷ-ㅂ-ㅁ
③ ㄱ-ㄴ-ㄷ-ㅇ-ㅂ-ㄹ-ㅅ-ㅁ
④ ㄷ-ㄹ-ㅇ-ㅅ-ㄱ-ㄴ-ㅂ-ㅁ

[기출 예상 문제]

23. 다음은 아키텍처 구축 절차의 어느 단계에 대한 설명이다. 어떤 단계에 대한 설명인가?

- 관점 정의: 이해 관계자를 파악하고, 이해 관계자 별 관점(view)을 정의한다.
- 아키텍처 스타일 선택: Pipe-Filter, MVC, Layer 등 스타일을 혼용하여 적용할 수 있다.
- 후보 아키텍처 도출: 배경도(Context Diagram) 및 각 관점 별 다이어그램을 작성한다. 소프트웨어 아키텍처 명세서(SAD: Software Architecture Desciption)를 기술한다.

① 요구사항 분석 ② 아키텍처 분석
③ 아키텍처 설계 ④ 검증 및 승인

[기출 예상 문제]

24. 다음은 무엇에 대한 설명인가?

필터에 해당되는 서브시스템이 하나의 데이터를 입력으로 받아 처리한 후 그 결과를 다음 서브시스템으로 넘겨주는 과정을 반복한다. 일반적으로 데이터를 변환하는 시스템에서 주로 사용한다.

① 데이터 흐름 구조 ② MVC 구조
③ 레이어 구조 ④ 클라이언트-서버 구조

[소프트웨어 설계>애플리케이션 설계>공통 모듈 설계]

[기출 예상 문제]

25. 빈칸의 내용을 올바르게 나열한 것은?

> 구현하려는 전체 어플리케이션을 (ㄱ), (ㄴ), (ㄷ) 로 구분하여 유저 인터페이스와 비즈니스 로직을 서로 분리하여 개발하는 방법이다. (ㄱ)은(는) 사용자 요청을 처리해 사용자에게 출력할 데이터를 만드는 요소이고, (ㄴ)은(는) 모델이 처리한 결과를 화면에 보여주는 요소이다. 그리고 (ㄷ)은(는) 사용자의 요청을 받아 그요청을 처리할 모델을 호출하고, 모델이 처리 후 결과를 뷰에게 전달하는 요소이다.

	(ㄱ)	(ㄴ)	(ㄷ)
①	View	Model	Controller
②	Model	View	Controller
③	Controller	Model	View
④	View	Controller	Model

[기출 예상 문제]

26. 은행 업무 시스템에서 가장 유용한 소프트웨어 아키텍처 스타일은?

① MVC 구조 ② 파이프 필터 구조
③ 저장소 구조 ④ 계층 구조

[기출 예상 문제]

27. 아키텍처 스타일에서 사용되는 MVC(Model-View-Controller) 구조는 서브시스템을 모델, 뷰, 컨트롤러로 구조화 할 때 Model 부분의 기능으로 가장 적절한 것은?

① 사용자에게 정보를 표시한다.
② 사용자로부터 받은 입력을 처리한다.
③ 뷰를 제어한다.
④ 서브시스템의 핵심 기능과 데이터를 보관한다.

[기출 예상 문제]

28. 다음은 무엇에 대한 설명인가?

> 시스템의 여러 가지 측면을 고려하기 위한 다양한 관점(View)을 바탕으로 정의된다. 고객 요구사항을 중심으로 4가지 관점으로 소프트웨어 아키텍처를 설계하는 기법이다.

① Master-Slaver Architecture
② Software Architecture 4+1 View
③ Model-View-Controller Architecture
④ Peer-To-Peer Architecture

[소프트웨어 설계>애플리케이션 설계>공통 모듈 설계]

[기출 예상 문제]

29. 다음 중 소프트웨어 아키텍처 4+1 View에 대한 설명으로 가장 거리가 먼 것은?

① 사용사례 관점은 시스템 외부 사용자 관점에서 시스템이 사용자에게 제공하는 기능에 관심이 있다.
② 구현 관점은 독립적으로 실행되는 컴포넌트와 이들 간 관계를 정의한다.
③ 배치 관점은 시스템을 구성하는 처리 장치 간의 논리적인 배치에 초점을 둔다.
④ 프로세스 관점은 모든 클래스가 아닌, 독자적인 제어 스레드(Thread)를 가질 수 있는 클래스에 초점을 맞춘다.

[소프트웨어 설계>애플리케이션 설계>객체지향 설계]

[기출 예상 문제]

1. 객체 지향 프로그래밍에 대한 설명으로 옳지 않은 것은?

① 하나의 클래스를 사용하여 여러 객체를 생성하는데, 각각의 객체를 클래스의 인스턴스(instance)라고 한다.
② 객체는 속성(attributes)과 행동(behaviors)으로 구성된다.
③ 메시지(Message)는 클래스로부터 생성된 객체를 사용하는 방법이다.
④ 객체지향 프로그램의 기본적인 사용자 정의 데이터형은 클래스(Class)이다.

[이전 기출 문제]

2. 객체지향 프로그래밍(Object-oriented programming :OOP) 개발 기법에 대한 설명으로 옳지 않은 것은?

① 절차 중심 프로그래밍 기법이다.
② 객체지향 프로그래밍 언어에는 Smalltalk, C++ 등이 있다.
③ 객체모델의 주요 요소는 추상화, 캡슐화, 모듈화, 계층 등이다.
④ 설계 시 자료와 자료에 가해지는 프로세스를 묶어 정의하고 관계를 규명한다.

[이전 기출 문제]

3. 객체 지향 언어에서 클래스 A와 클래스 B는 상속관계에 있다. A는 부모 클래스, B는 자식 클래스라고 할 때 클래스 A에서 정의된 메서드(method)와 원형이 동일한 메서드를 클래스 B에서 기능을 추가하거나 변경하여 다시 정의하는 것을 무엇이라고 하는가?

① 추상 클래스(abstract class)
② 인터페이스(interface)
③ 오버로딩(overloading)
④ 오버라이딩(overriding)

[이전 기출 문제]

4. 다음에서 설명하는 객체지향 프로그래밍의 특징은?

- 객체를 구성하는 속성과 메서드가 하나로 묶여 있다.
- 객체의 외부와 내부를 분리하여 외부 모습은 추상적인 내용으로 보여준다.
- 객체 내의 정보를 외부로부터 숨길 수도 있고, 외부에 보이게 할 수도 있다.
- 객체 내부의 세부 동작을 모르더라도 객체의 메서드를 통해 객체의 기능을 활용할 수 있다.

① 구조성　② 다형성
③ 상속성　④ 캡슐화

[소프트웨어 설계>애플리케이션 설계>객체지향 설계]

[이전 기출 문제]
5. 객체지향 기법에서 상속(INHERITANCE)의 결과로서 얻을 수 있는 가장 주요한 이점은?

① 모듈 라이브러리의 재이용
② 객체지향 DB를 사용할 있는 능력
③ 클래스와 오브젝트들을 재사용할 수 있는 능력
④ 프로젝트를 보다 효과적으로 관리할 수 있는 능력

[이전 기출 문제]
6. 객체지향 시스템의 다형성(polymorphism)에 대하여 바르게 기술한 것은?

① 한 객체가 가지고 있는 데이터의 여러 속성들
② 한 객체가 가지고 있는 여러 가지 연산 기능들
③ 한 클래스에서 여러 개의 객체를 생성하는 것
④ 한 메시지가 객체에 따라 다른 방법으로 응답할 수 있는 것

[이전 기출 문제]
7. 객체는 다른 객체로부터 자신의 자료를 숨기고 자신의 연산만을 통하여 접근을 허용하는 것은 무엇이라 하는가?

① abstraction ② information hiding
③ modularity ④ typing

[기출 예상 문제]
8. 오버라이딩(Overriding)과 오버로딩(Overloading)에 대한 설명으로 가장 적절한 것은?

① 오버라이딩(Overriding)의 매개변수 개수와 타입은 반드시 동일해야 한다.
② 오버로딩(Overloading)의 접근 범위는 같거나 넓어야 한다.
③ 하나의 클래스 내에서 같은 이름으로 여러 개의 메소드를 정의하는 것은 오버라이딩(Overriding)이다.
④ 오버로딩(Overloading)은 상속관계에서 상위 클래스의 메소드를 하위클래스에서 재정의한다.

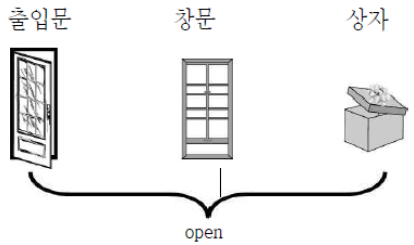
[기출 예상 문제]
9. 다음 중 객체지향의 기법에 대한 설명으로 가장 거리가 먼 것은?

① 다형성(Polymorphism)은 높은 응집도 실현이 가능하다.
② 현실세계를 자연스럽게 표현한 것으로, 공통 성질을 추출하여 슈퍼클래스로 구성하는 기법은 추상화(Abstraction)이다.
③ 정보은닉(Information Hiding)은 캡슐화된 항목을 다른 객체(Obejct)로 부터 숨기는 기법이다.
④ 상속성(Inheritance)은 수직/수평적 구조로 과도한 상속은 결합도 상승의 요인이 될 수 있다.

[소프트웨어 설계>애플리케이션 설계>객체지향 설계]

[이전 기출 문제]

10. <보기>에서 설명하는 객체지향 개념은?



- 그림에서 'open'이라는 오퍼레이션(operation)은 객체마다 다르게 기능한다.
- Java 언어에서 오버로딩(overloading), 오버라이딩(overriding)으로 구현되는 개념이다.

① 캡슐화(encapsulation) ② 인스턴스(instance)
③ 다형성(polymorphism) ④ 상속(inheritance)

[이전 기출 문제]

11. 이미 정의되어 있는 상위 클래스의 메소드를 비롯한 모든 속성을 하위 클래스가 물려받는 것으로, 이를 이용하면 하위 클래스는 상위 클래스의 메소드 및 모든 속성을 자신의 클래스 내에 다시 정의하지 않고서도 자신의 속성으로 가질 수 있는 것은?

① method ② information hidden
③ inheritance ④ polymorphism

[이전 기출 문제]

12. 클래스 설계 원칙에 대한 설명으로 옳은 것은?

① 인터페이스 분리(Interface Segregation) 원칙:추상 클래스나 인터페이스에 의존하지 않고 자주 변경되는 클래스에 의존해야 한다.
② 개방 – 폐쇄(Open-Closed) 원칙:클래스는 확장(extension)에 대해서는 열려있어야 하며 변경(change)에 대해서 닫혀있어야 한다.
③ 리스코프 교체(Liskov Substitution) 원칙:여러 개의 책임을 가진 클래스는 하나의 책임을 가진 클래스로 대체되어야 한다.
④ 의존 관계 역전(Dependency Inversion) 원칙:클라이언트는 자신이 사용하는 메소드와 의존 관계를 갖지 않도록 해야 한다.

[이전 기출 문제]

13. 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

[소프트웨어 설계>애플리케이션 설계>객체지향 설계]

[이전 기출 문제]
14. ㉠에 들어갈 용어로 옳은 것은?

(㉠) (은)는 유사한 문제를 해결하기 위해 설계들을 분류하고 각 문제 유형별로 가장 적합한 설계를 일반화하여 체계적으로 정리해 놓은 것으로 소프트웨어 개발에서 효율성과 재사용성을 높일 수 있다.

① 디자인 패턴
② 요구사항 정의서
③ 소프트웨어 개발 생명주기
④ 소프트웨어 프로세스 모델

[이전 기출 문제]
15. 디자인 패턴에 대한 설명으로 가장 거리가 먼 것은?

① 문제를 해결하려는 요소들을 일반화하여 잘 정리한 것이다.
② 특정 문제에 대한 가능한 모든 해결책들 및 소스 코드를 모두 제공한 것이다.
③ 여러 가지 상황에 적용될 수 있는 템플릿과 같은 것이다.
④ 문제에 대한 설계를 추상적으로 표현한 것이다.

[이전 기출 문제]
16. <보기 1>의 디자인 패턴 분류와 <보기 2>의 디자인 패턴을 바르게 연결한 것은?

<보기 1>
ㄱ. 생성 패턴 ㄴ. 구조 패턴 ㄷ. 행위 패턴

<보기 2>
A. Bridge 패턴 B. Singleton 패턴 C. Interpreter 패턴

	ㄱ	ㄴ	ㄷ
①	A	B	C
②	B	A	C
③	B	C	A
④	C	A	B

[이전 기출 문제]
17. GoF 디자인 패턴에 대한 설명으로 가장 거리가 먼 것은?

① 데커레이터 패턴은 생성 패턴에 속한다.
② 목적에 따라 생성 패턴, 구조 패턴, 행위 패턴으로 구분한다.
③ 생성 패턴은 객체 생성과 관련된 패턴이다.
④ 행위 패턴에는 커맨드 패턴, 이터레이터 패턴, 옵저버 패턴 등이 있다.

[소프트웨어 설계>애플리케이션 설계>객체지향 설계]

[이전 기출 문제]
18. 행동 패턴에 대한 설명으로 가장 거리가 먼 것은?

① 행동 패턴 종류에는 팩토리 메소드 패턴, 프로토 타입 패턴이 있다.
② 행동 패턴은 클래스나 객체들이 상호작용하는 방법과 책임을 분산하는 방법이다.
③ 행동 패턴 종류 중 이터레이터 패턴은 내부 표현부를 노출하지 않는다.
④ 옵저버 패턴은 행동 패턴에 속한다.

[이전 기출 문제]
19. 디자인 패턴 중 생성 관련 패턴(Creational Pattern)과 가장 거리가 먼 것은?

① Builder ② Singleton
③ Abstract Factory ④ Observer

[이전 기출 문제]
20. 다음 중 성격이 다른 설계 패턴은?

① 브리지(bridge) 패턴
② 팩토리 메소드(factory method) 패턴
③ 프로토타입(prototype) 패턴
④ 싱글톤(singleton) 패턴

[이전 기출 문제]
21. CPU나 메모리와 같은 컴퓨터 부품의 가격은 다양한 이유로 가격 결정 정책이 자주 변경될 수 있다. 이를 위해 가격 결정 정책을 클래스로 캡슐화하여 쉽게 변경 및 추가할 수 있는 설계패턴은?

① Visitor ② Observer
③ Template Method ④ Strategy

[이전 기출 문제]
22. 상속을 사용하지 않고도 객체의 기능을 동적으로 확장할 수 있도록 해주는 설계 패턴은?

① 데코레이터(decorator) 패턴
② 어댑터(adapter) 패턴
③ 컴포지트(composite) 패턴
④ 퍼사드(facade) 패턴

[이전 기출 문제]
23. 다음에서 설명하는 디자인 패턴에 해당하는 것은?

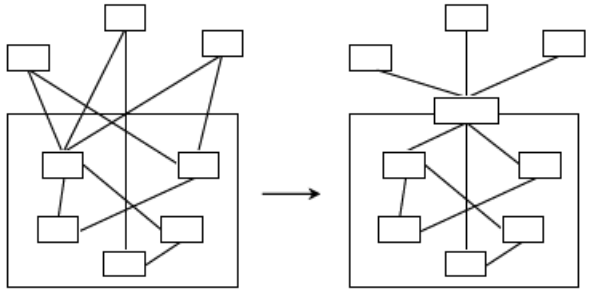
> 이미 만들어져 있는 클래스를 사용하고 싶지만 인터페이스가 원하는 방식과 일치하지 않을 때, 또는 관련성이 없거나 예측하지 못한 클래스들과 협동하는 재사용 가능한 클래스를 생성하기를 원할 때 사용한다.

① Bridge ② Adapter
③ Composite ④ Facade

[소프트웨어 설계>애플리케이션 설계>객체지향 설계]

[이전 기출 문제]

24. 그림과 같이 서브시스템 사이의 의사소통 및 종속성을 최소화하기 위하여 단순화된 하나의 인터페이스를 제공하는 디자인 패턴은?



① Adapter 패턴　② Bridge 패턴
③ Decorator 패턴　④ Facade 패턴

[이전 기출 문제]

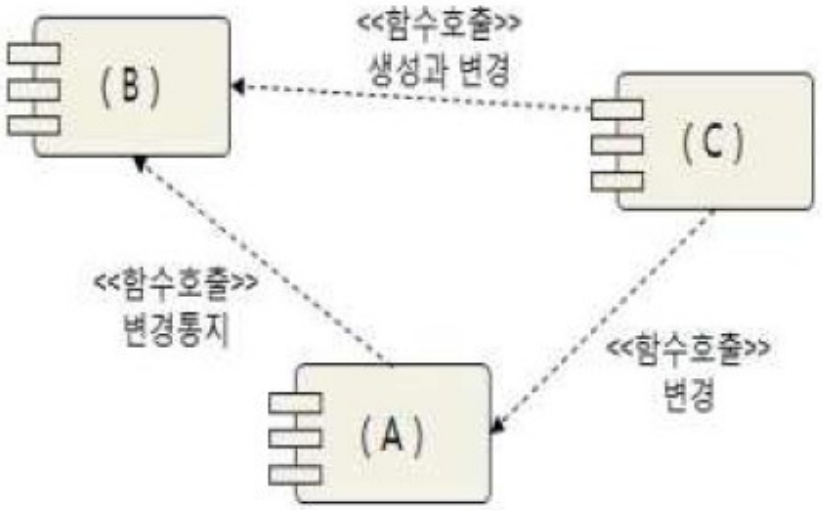
25. 다음 설명에 해당되는 디자인 패턴은?

> 1대 다(多)의 객체 의존관계를 정의한 것으로 한 객체가 상태를 변화시켰을 때, 의존관계에 있는 다른 객체들에게 자동적으로 통지하고 변경시킨다.

① Observer　② Facade
③ Mediator　④ Bridge

[이전 기출 문제]

26. 다음 그림은 MVC(Model-View-Controller) 아키텍처 스타일을 나타낸 것이며, 세 가지 요소가 별도의 컴포넌트 또는 스레드로 구성된다. (A), (B), (C)에 해당되는 요소를 바르게 나열한 것은?



① 모델 – 제어 – 뷰　② 뷰 – 모델 – 제어
③ 뷰 – 제어 – 모델　④ 모델 – 뷰 – 제어

[정답] [소프트웨어 설계>애플리케이션 설계]

1. 공통 모듈 설계

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
4	4	2	4	1	3	1	4	2	3	3	2	1	3	4	3	2	3	4	3
21	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**											
4	2	3	1	2	3	4	2	3											

2. 객체지향 설계

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
3	1	4	4	3	4	2	1	4	3	3	2	4	1	2	2	1	1	4	1
21	**22**	**23**	**24**	**25**	**26**														
4	1	2	4	1	4														

[정답 및 해설] [소프트웨어 설계>애플리케이션 설계>공통 모듈 설계]

공통 모듈 설계 1. ④

공통 모듈 명세 작성 원칙
- 정확성: 실제 시스템 구현 시 필요한 기능인지 여부를 알 수 있도록 정확하게 작성한다.
- 명확성: 해당 기능에 대한 일관된 이해와 하나로 해석될 수 있도록 작성한다.
- 완전성: 시스템의 구현 시 요구사항과 필요한 모든 것을 기술한다.
- 일관성: 공통 기능 사이에 충돌이 발생하지 않도록 작성한다.
- 추적성: 해당 기능에 대한 요구사항의 출처와 관련 시스템 등 유기적 관계에 대한 식별이 가능하도록 작성한다.

공통 모듈 설계 2. ④

④ 응집도에 대한 설명이다. 결합도는 모듈과 모듈 사이의 관련성이 어느 정도인가를 나타낸다.

공통 모듈 설계 3. ②

응집도가 약한 정도에서 강한 정도 순으로 나열하면 우연적(Coincidental) 응집도<논리적(Logical) 응집도<시간적(Temporal) 응집도<절차적(Procedural) 응집도<교환(통신)적(Communication) 응집도<순차적(Sequential) 응집도<기능적(Functional) 응집도이다.

공통 모듈 설계 4. ④

④ 모듈의 크기가 클수록 하나의 모듈을 개발하는데 소용되는 비용이 커진다.

공통 모듈 설계 5. ①

모듈의 독립성은 결합도와 응집도에 의해 측정되며, 독립성을 높이려면 모듈의 결합도는 약하게, 응집도는 강하게, 모듈의 크기는 작게 만들어야한다. 결합도는 모듈 간에 상호 의존하는 정도 또는 두 모듈 사이의 연관 관계를 의미하고, 응집도는 모듈의 내부 요소들의 서로 관련되어 있는 정도를 의미한다.

공통 모듈 설계 6. ③

- 결합도가 강한 정도에서 약한 정도 순으로 나열하면 내용 결합도>공통 결합도>외부 결합도>제어 결합도>스탬프 결합도>자료 결합도이다.
- 응집도가 강한 정도에서 약한 정도 순으로 나열하면 기능적 응집도>순차적 응집도>교환(통신)적 응집도>절차적 응집도>시간적 응집도>논리적 응집도이다.

공통 모듈 설계 7. ①

우연적 응집도(Coincidental Cohesion)은 모듈 내부의 각 구성요소들이 서로 관련 없는 요소로만 구성된 경우의 응집도이다.

[정답 및 해설] [소프트웨어 설계>애플리케이션 설계>공통 모듈 설계]

공통 모듈 설계 8. ④

④ 가장 높은 품질을 결합도 순으로 나열하면 자료 결합>스탬프 결합>제어 결합>외부 결합>공통 결합>내용 결합이다.

공통 모듈 설계 9. ②

스탬프 결합도(Stamp Coupling)은 두 모듈 사이에서 정보를 교환할 때 필요한 데이터만 주고받을 수 없고 필요 없는 데이터까지 전체(데이터 구조, 자료 구조)를 주고 받아야 하는 경우이다.

공통 모듈 설계 10. ③

ㄴ. 모듈 내 구성 요소들이 서로 다른 기능을 같은 시간대에 함께 실행하는 경우는 시간적 응집도(temporal cohesion)에 해당한다.

공통 모듈 설계 11. ③

소프트웨어 설계 프로세스의 요구 명세에는 기능적 요구, 품질적 요구, 형태적 요구가 있다.

공통 모듈 설계 12. ②

② 컴포넌트 제어는 시스템 행위의 구성 요소이다.

공통 모듈 설계 13. ①

① 행위 모델에 대한 설명이다.

공통 모듈 설계 14. ③

③ 컴포넌트 구조의 구성 요소에는 인터페이스, 데이터 구조, 알고리즘 구조가 있다.

공통 모듈 설계 15. ④

구조(Structure) 모델링은 소프트웨어를 구성하는 컴포넌트들의 유형, 인터페이스, 내부 설계 구조 및 이들의 상호 연결 구조를 모델링한다. 행위(Behavior) 모델링은 소프트웨어의 구성요소들의 기능들과 이들이 언제, 어떠한 순서로 기능을 수행하고 상호작용하는지를 모델링한다.

공통 모듈 설계 16. ③

③ 동적 생성 및 소멸, 동적 결합/연결은 구조 모델의 동적(Dynamic)요소 이다.

공통 모듈 설계 17. ②

② 상태전이는 행위 모델의 동적 요소이다.

공통 모듈 설계 18. ③

③ 행위 모델에 대한 설명이다.

[정답 및 해설] [소프트웨어 설계>애플리케이션 설계>공통 모듈 설계]

공통 모듈 설계 19. ④

소프트웨어 아키텍처는 소프트웨어의 골격이 되는 구조이자, 소프트웨어를 구성하는 요소들 간의 관계를 표현하는 시스템의 구조 또는 구조체이다. 소프트웨어 개발 시 적용되는 원칙과 지침이며, 이해 관계자들의 의사소통 도구로 활용된다.

공통 모듈 설계 20. ③

③ 소프트웨어의 복잡성의 문제를 해결해야 한다.

공통 모듈 설계 21. ④

④ 설계의 일반 원리: 소프트웨어 아키텍처에 적용되는 보편적인 원칙이다. / 아키텍처 패턴: 반복적으로 발생하는 문제에 대해 미리 만들어진 솔루션이다.

공통 모듈 설계 22. ②

아키텍처 설계 절차와 적용 원리
1. 아키텍처 드라이버
2. 품질 속성, 검증 가능성, 품질속성 시나리오
3. 문제 분석
4. 컴포넌트와 커넥터
5. 아키텍처 스타일
6. 소프트웨어 아키텍처를 보는 관점 체계
7. 설계의 일반원리
8. 아키텍처 설계 절차
9. 아키텍처 패턴
10. 품질속성 설계전술
11. 아키텍처의 분석
12. 아키텍처의 평가

공통 모듈 설계 23. ③

③ 아키텍처 구축 절차 중 아키텍처 설계 단계의 설명이다.

공통 모듈 설계 24. ①

데이터 흐름 구조(파이프 필터 구조)에 대한 설명이다. 데이터 흐름 구조의 예는 이미지 프로세싱(영상처리)가 있다.

[정답 및 해설] [소프트웨어 설계>애플리케이션 설계>공통 모듈 설계]

공통 모듈 설계 25. ②

- 모델(Model): 서브시스템의 핵심 기능과 데이터를 보관한다.
- 뷰(View): 사용자에게 정보를 표시한다.
- 컨트롤러(Controller): 사용자로부터 받은 입력을 처리한다.

공통 모듈 설계 26. ③

저장소 구조는 저장소에 공동으로 활용하는 데이터를 보관하고, 모든 서브시스템이 여기에 저장된 공유 데이터에 접근하여 정보를 저장, 검색, 변경하는 역할을 한다. 대량의 데이터를 공유하는 은행 업무 시스템에는 저장소 구조가 매우 유용하다.

공통 모듈 설계 27. ④

① 뷰(View)의 역할이다.
②, ③ 컨트롤러(Controller)의 역할이다.

공통 모듈 설계 28. ②

소프트웨어 아키텍처 4+1 View에 대한 설명이다.

공통 모듈 설계 29. ③

③ 배치 관점(Deployment View)는 시스템을 구성하는 처리 장치 간의 물리적인 배치에 초점을 둔다. 또 하나 시스템의 분산 구조와 실행할 때 컴포넌트들의 배치 상태를 나타낸다.

[정답 및 해설] [소프트웨어 설계>애플리케이션 설계>객체지향 설계]

객체지향 설계 1. ③

③ 메시지(Message)는 객체(Object)간의 통신을 말한다. 클래스로부터 생선된 객체를 사용하는 방법은 메소드(Method)이다.

객체지향 설계 2. ①

① 객체지향 프로그래밍은 객체 중심 프로그래밍 기법이다.

객체지향 설계 3. ④

오버라이딩(Overriding)은 상속관계에서 상위 클래스의 메소드를 하위클래스에서 재정의한다. 오버로딩(Overloading)은 하나의 클래스 내에서 같은 이름으로 여러 개의 메소드를 정의한다.

객체지향 설계 4. ④

캡슐화는 데이터(속성)와 데이터를 처리하는 함수(메서드)를 하나로 묶는 것을 의미한다. 캡슐화된 객체는 세부 내용이 은폐(정보 은닉)되어 외부에서의 접근이 제한적이다. 객체들 간의 메시지를 주고 받을 때 상대 객체의 세부 내용은 알 필요가 없다.

객체지향 설계 5. ③

상속(Inheritance)은 이미 정의된 상위 클래스의 모든 속성과 연산을 하위 클래스가 물려 받는 것이다. 상위 클래스의 속성과 연산을 하위 클래스가 사용할 수 있기 때문에 소프트웨어 재사용을 높이는 중요한 개념이다.

객체지향 설계 6. ④

다형성은 메시지에 의해 객체(클래스)가 연산을 수행하게 될 때 하나의 메시지에 대해 각각의 객체(클래스)가 가지고 있는 고유한 방법(특성)으로 응답할 수 있는 능력이다.

객체지향 설계 7. ②

② 정보은닉(Information Hiding)은 캡슐화된 객체를 다른 객체로부터 숨기고, 메시지만으로 객체와의 상호작용을 하게 하는 성질을 의미한다.

객체지향 설계 8. ①

② 오버로딩(Oveloading)의 접근은 제한이 없다.
③ 하나의 클래스 내에서 같은 이름으로 여러 개의 메소드를 정의하는 것은 오버로딩(Overloading)이다.
④ 오버라이딩(Overrding)은 상속관계에서 상위 클래스의 메소드를 하위클래스에서 재정의한다.

[정답 및 해설] [소프트웨어 설계>애플리케이션 설계>객체지향 설계]

객체지향 설계 9. ④

④ 상속성(Ingeritance)은 수직적 구조로 과도한 상속은 결합도 상승의 요인이 될 수 있다.

객체지향 설계 10. ③

다형성(Polymorphism)은 동일 인터페이스에 대해 서로 다른 처리 방식으로 구현 가능한 특성이다.

객체지향 설계 11. ③

상속성(Inheritance)은 부모(Super) 클래스의 메소드와 속성을 자식(Sub) 클래스가 물려 받는 특성이므로 부모 클래스의 메소드와 연산을 자식 클래스가 사용할 수 있다.

객체지향 설계 12. ②

① 인터페이스 분리(Interface Segregation) 원칙: 하나의 일반적인 인터페이스보다 구체적인 여러 개의 인터페이스가 낫다.
③ 리스코프 교체(Liskov Substitution) 원칙: 하위 클래스 및 타입들은 상위 타입들이 사용되는 곳에 대체될 수 있어야 하는 설계 원칙이다ㅏ.
④ 의존 관계 역전(Dependency Inversion) 원칙: 추상화된 것에 의존하게 만들고 구체클래스에 의존하도록 만들지 않도록 한다.

객체지향 설계 13. ④

인터페이스 분리 원칙은 하나의 일반적인 인터페이스보다 구체적인 여러 개의 인터페이스가 낫다는 원칙으로 자신이 이용하지 않는 기능에는 영향을 받지 않아야 한다는 의미이다.

객체지향 설계 14. ①

디자인 패턴(Design Pattern)은 반복적으로 나타나는 문제들을 해결해 온 전문가들의 경험을 모아서 정리한 일관된 솔루션으로 설계의 재사용을 통해 생산성 향상을 위한 기법이다.

객체지향 설계 15. ②

② 디자인 패턴은 소스 코드를 제공하지 않는다.

객체지향 설계 16. ②

- 생성패턴: Singleton, Factory Method, Abstract Factory, Prototype, Builder 패턴
- 구조패턴: Decorator, Adaptor, Compsite, Facade, Proxy, Bridge, Flyweight 패턴
- 행위패턴: Strategy, Observer, State, Command, Iterator, Template Method, Interpreter 패턴

[정답 및 해설] [소프트웨어 설계>애플리케이션 설계>객체지향 설계]

객체지향 설계 17. ①

① 데커레이터 패턴(Decorator Pattern)은 구조 패턴에 속한다.

객체지향 설계 18. ①

① 팩토리 메소드 패턴(Factory Method Pattern), 프로토 타입(Prototype Pattern)은 생성 패턴에 속한다.

객체지향 설계 19. ④

- 생성패턴: Singleton, Factory Method, Abstract Factory, Prototype, Builder 패턴
- 구조패턴: Decorator, Adaptor, Compsite, Facade, Proxy, Bridge, Flyweight 패턴
- 행위패턴: Strategy, Observer, State, Command, Iterator, Template Method, Interpreter 패턴

객체지향 설계 20. ①

① 구조 패턴
②, ③, ④ 생성패턴

객체지향 설계 21. ④

전략(Strategy) 패턴은 다형성을 이용하여 특정 객체에 종속되지 않도록 한다. 다른 부분을 분리하여 캡슐화하는 패턴이다.

객체지향 설계 22. ①

데코레이터(Decorator) 패턴은 기존 객체의 메서드에 새로운 행동을 추가하거나 오버라이딩 할 수 있다.

객체지향 설계 23. ②

어댑터(Adapter) 패턴은 인터페이스가 호환되지 않는 클래스들을 함께 이용할 수 있도록한다. (구조가 다른 클래스 연결)

객체지향 설계 24. ④

퍼사드(Facade) 패턴은 메소드를 단순화하여 라이브러리를 쉽게 사용하고 이해할 수 있게한다.

객체지향 설계 25. ①

옵저버(Observer) 패턴은 어떤 클래스에 변화가 일어났을 때, 이를 감지하여 다른 클래스에 통보해준다.(모니터링)

객체지향 설계 26. ④

(A)→(C) 컨트롤러(Controller)가 정보를 갱신하여 모델(Model)에게 전달한다.
(A)→(B) 모델(Model)이 뷰(View)에게 변경을 알린다.
(C)→(B) 컨트롤러(Controller)가 뷰(View)를 제어(생성/변경)한다.