1과목-소프트웨어 설계

(Part 3. 애플리케이션 설계)

애플리케이션 설계 세부 섹션

애플리케이션 설계 파트는 총 8개의 작은 장으로 구성된다.

01 소프트웨어 아키텍처

02 아키텍처 패턴

03 객체지향(Object-Oriented)

04 객체지향 분석 및 설계

05 모듈

06 공통 모듈

07 코드

08 디자인 패턴

1. 애플리케이션 설계-SEC_01(소프트웨어 아키텍처)

- 이 장을 공부하면서 반드시 알아두어야 할 키워드

; 파이프-필터 패턴, 캡슐화, 럼바우의 분석 기법, 객체지향 설계 원칙, 결합도, 응집도, 효과적인 모듈 설계 방안, 생성 패턴, 구조 패턴, 행위 패턴

1) 소프트웨어 아키텍처의 설계

; 소프트웨어 아키텍처는 소프트웨어의 골격이 되는 기본 구조이자, 소프트웨어를 구성하는 요소들 간의 관계를 표현하는 시스템의 구조 또는 구조체이다.

- 소프트웨어 개발 시 적용되는 원칙과 지침이며, 이해 관계자들의 의사소통 도구로 활용된다.
- 소프트웨어 아키텍처의 설계는 기본적으로 좋은 품질을 유지하면서 사용자의 비기능적 요구사항으로 나타난 제약을 반영하고, 기능적 요구사항을 구현하는 방법을 찾는 해결 과정이다.
- 애플리케이션의 분할 방법과 분할된 모듈에 할당될 기능, 모듈 간의 인터페이스 등을 결정한다.
- 소프트웨어 아키텍처 설계의 기본 원리로는 모듈화, 추상화, 단계적 분해, 정보은닉이 있다.
- 시스템이 갖춰야 할 필수적인 기능에 대한 요구항목들을 기능적 요구사항이라고 하며, 그 외의 품질이나 제약사항에 관한 것을 비기능적 요구사항이라고 한다.

1. 애플리케이션 설계-SEC_01(소프트웨어 아키텍처)

1) 소프트웨어 아키텍처의 설계

; 소프트웨어 개발의 설계 단계는 크게 상위 설계와 하위 설계로 구분할 수 있다.

구분	상위 설계	하위 설계
별칭	아키텍처 설계, 예비 설계	모듈 설계, 상세 설계
설계 대상	시스템의 전체적인 구조	시스템의 내부 구조 및 행위
세부 목록	구조, DB, 인터페이스	컴포넌트, 자료구조, 알고리즘

2) 모듈화(Modularity)

; 모듈화란 소프트웨어의 성능을 향상시키거나 시스템의 수정 및 재사용, 유지 관리 등이 용이하도록 시스템의 기능들을 모듈 단위로 나누는 것을 의미한다.

- 자주 사용되는 계산식이나 사용자 인증과 같은 기능들을 공통 모듈로 구성하여 프로젝트의 재 사용성을 향상시킬 수 있다.
- 모듈의 크기를 너무 작게 나누면 개수가 많아져 모듈 간의 통합 비용이 많이 들고, 너무 크게 나누면 개수가 적어 통합 비용은 적게 들지만 모듈 하나의 개발 비용이 많이 든다.
- 모듈화를 통해 기능의 분리가 가능하여 인터페이스가 단순해진다.
- 모듈화를 통해 프로그램의 효율적인 관리가 가능하고 오류의 파급 효과를 최소화 할 수 있다.

모듈 : 모듈화를 통해 분리된 시스템의 각 기능들로 서브루틴, 서브 시스템 소프트웨어 내의 프로그램, 작업 단위 등과 같은 의미로 사용된다.

1. 애플리케이션 설계-SEC_01(소프트웨어 아키텍처)

3) 추상화(Abstraction)

; 추상화는 문제의 전체적이고 포괄적인 개념을 설계한 후 차례로 세분화하여 구체화시켜 나가는 것이다.

- 인간이 복잡한 문제를 다룰 때 가장 기본적으로 사용하는 방법으로, 완전한 시스템을 구축하기 전에 그 시스템과 유사한 모델을 만들어서 여러 가지 요인들을 테스트할 수 있다.
- 추상화는 최소의 비용으로 실제 상황에 대처할 수 있고, 시스템의 구조 및 구성을 대략적으로 파악할 수 있게 해준다.
- 추상화의 유형

과정 추상화	자세한 수행 과정을 정의하지 않고, 전반적인 흐름만 파악할 수 있게 설계하는 방법
데이터 추상화	데이터의 세부적인 속성이나 용도를 정의하지 않고, 데이터 구조를 대표할 수 있는 표현으로 대체하는 방법
제어 추상화	이벤트 발생의 정확한 절차나 방법을 정의하지 않고, 대표할 수 있는 표현으로 대체하는 방법

1. 애플리케이션 설계-SEC_01(소프트웨어 아키텍처)

4) 단계적 분해(Stepwise Refinement)

; 단계적 분해는 Niklaus Wirth에 의해 제안된 하향식 설계 전략으로, 문제를 상위의 중요 개념으로부터 하위의 개념으로 구체화시키는 분할 기법이다.

- 추상화의 반복에 의해 세분화된다.
- 소프트웨어의 기능에서부터 시작하여 점차적으로 구체화하고, 알고리즘, 자료 구조 등 상세한 내역은 가능한 한 뒤로 미루어 진행한다.

5) 정보 은닉(Information Hiding)

; 정보 은닉은 한 모듈 내부에 포함된 절차와 자료들의 정보가 감추어져 다른 모듈이 접근하거나 변경하지 못하도록 하는 기법이다.

- 어떤 모듈이 소프트웨어 기능을 수행하는데 반드시 필요한 기능이 있어 정보 은닉된 모듈과 커뮤니케이션 할 필요가 있을 때는 필요한 정보만 인터페이스를 통해 주고 받는다.
- 정보 은닉을 통해 모듈을 독립적으로 수행할 수 있고, 하나의 모듈이 변경되더라도 다른 모듈에 영향을 주지 않으므로 수정, 시험, 유지보수가 용이하다.

정보 은닉의 예 : 캡슐로 된 감기약을 예로 들면, 정보 은닉은 감기약 캡슐에 어떤 재료가 들어 있는지 몰라도 감기 걸렸 을 때 먹는 약이라는 것만 알고 복용하는 것과 같은 의미라고 보면 된다.

1. 애플리케이션 설계-SEC_01(소프트웨어 아키텍처)

6) 소프트웨어 아키텍처의 품질 속성

; 소프트웨어 아키텍처의 품질 속성은 소프트웨어 아키텍처가 이해 관계자들이 요구하는 수준의 품질을 유지 및 보장할 수 있게 설계되었는지를 확인하기 위해 품질 평가 요소들을 시스템 측면, 비즈니스 측면, 아키텍처 측면으로 구분하여 구체화시켜 놓은 것이다.

● 시스템 측면

품질 속성	내용
성능	사용자의 요청과 같은 이벤트가 발생했을 때, 이를 적절하고 빠르게 처리하는 것이다.
보안	허용되지 않은 접근을 막고, 허용된 접근에는 적절한 서비스를 제공하는 것이다.
가용성	장애 없이 정상적으로 서비스를 제공하는 것이다.
기능성	사용자가 요구한 기능을 만족스럽게 구현하는 것이다.
사용성	사용자가 소프트웨어를 사용하는데 헤매지 않도록 명확하고 편리하게 구현하는 것이다.
변경 용이성	소프트웨어가 처음 설계 목표와 다른 하드웨어나 플랫폼에서도 동작할 수 있도록 구현하는 것이다.
확장성	시스템의 용량 처리능력 등을 확장시켰을 때 이를 효과적으로 활용할 수 있도록 구현하는 것이다.
기타 속성	테스트 용이성, 배치성, 안정성 등이 있다.

1. 애플리케이션 설계-SEC_01(소프트웨어 아키텍처)

6) 소프트웨어 아키텍처의 품질 속성

● 비즈니스 측면

품질 속성	내용
시장 적시성	정해진 시간에 맞춰 프로그램을 출시하는 것이다.
비용과 혜택	• 개발 비용을 더 투자하여 유연성이 높은 아키텍처를 만들 것인지를 결정하는 것이다. • 유연성이 떨어지는 경우 유지보수에 많은 비용이 소모될 수 있다는 것을 고려해야 한다.
예상 시스템 수명	• 시스템을 얼마나 오랫동안 사용할 것인지를 고려하는 것이다. • 수명이 길어야 한다면 시스템 품질의 '변경 용이성', '확장성'을 중요하게 고려해야 한다.
기타 속성	목표 시장, 공개 일정, 기존 시스템과의 통합 등이 있다.

● 아키텍처 측면

품질 속성	내용
개념적 무결성	전체 시스템과 시스템을 이루는 구성 요소들 간의 일관성을 유지하는 것이다.
정확성, 완결성	요구사항과 요구사항을 구현하기 위해 발생하는 제약사항들을 모두 충족시키는 것이다.
구축 가능성	모듈 단위로 구분된 시스템을 적절하게 분배하여 유연하게 일정을 변경할 수 있도록 하는 것이다.
기타 속성	변경성, 시험성, 적응성, 일치성, 대체성 등이 있다.

1. 애플리케이션 설계-SEC_01(소프트웨어 아키텍처)

7) 소프트웨어 아키텍처의 설계 과정

; 아키텍처의 설계 과정은 설계 목표 설정, 시스템 타입 결정, 아키텍처 패턴 적용, 서브 시스템 구체화, 검토 순으로 진행된다.

① 설계 목표 설정 : 시스템의 개발 방향을 명확히 하기 위해 설계에 영향을 주는 비즈니스 목표, 우선순위 등의 요구사항을 분석하여 전체 시스템의 설계 목표를 설정한다.

② 시스템 타입 결정 : 시스템과 서브 시스템의 타입을 결정하고, 설계 목표와 함께 고려하여 아키텍처 패턴을 선택한다.

③ 아키텍처 패턴 적용 : 아키텍처 패턴을 참조하여 시스템의 표준 아키텍처를 설계한다.

④ 서브 시스템 구체화 : 서브 시스템의 기능 및 서브 시스템 간의 상호작용을 위한 동작과 인터페이스를 정의한다.

⑤ 검토 : 아키텍처가 설계 목표에 부합하는지, 요구사항이 잘 반영되었는지, 설계의 기본 원리를 만족하는지 등을 검토한다.

1. 애플리케이션 설계-SEC_01(소프트웨어 아키텍처)

8) 시스템 타입 / 협약에 의한 설계

① 시스템 타입

; 시스템 타입은 일반적으로 다음 네 가지 타입으로 나눌 수 있다.

- 대화형 시스템 : 사용자의 요구가 발생하면 시스템이 이를 처리하고 반응하는 시스템

  예) 온라인 쇼핑몰과 같은 대부분의 웹 애플리케이션

- 이벤트 중심 시스템 : 외부의 상태 변화에 따라 동작하는 시스템

  예) 전화, 비상벨 등의 내장 소프트웨어

- 변환형 시스템 : 데이터가 입력되면 정해진 작업들을 수행하여 결과를 출력하는 시스템

  예) 컴파일러, 네트워크 프로토콜 등

- 객체 영속형 시스템 : 데이터베이스를 사용하여 파일을 효과적으로 저장, 검색, 갱신할 수 있는 시스템

  예) 서버 관리 소프트웨어

1. 애플리케이션 설계-SEC_01(소프트웨어 아키텍처)

8) 시스템 타입 / 협약에 의한 설계

② 협약(Contract)에 의한 설계

; 컴포넌트를 설계할 때 클래스에 대한 여러 가정을 공유할 수 있도록 명세한 것으로, 소프트웨어 컴포넌트에 대한 정확한 인터페이스를 명세한다.

● 협약에 의한 설계 시 명세에 포함될 조건에는 선행 조건, 결과 조건, 불변 조건이 있다.

선행 조건(Precondition)	오퍼레이션이 호출되기 전에 참이 되어야 할 조건
결과 조건(Postcondition)	오퍼레이션이 수행된 후 만족되어야 할 조건
불변 조건(Invariant)	오퍼레이션이 실행되는 동안 항상 만족되어야 할 조건

1. 애플리케이션 설계- SEC_01(소프트웨어 아키텍처) 기출 및 예상 문제

기출 및 예상 문제(소프트웨어 아키텍처)

1. 소프트웨어의 상위 설계에 속하지 않는 것은?

① 아키텍처 설계 ② 모듈 설계

③ 인터페이스 정의 ④ 사용자 인터페이스 설계

설명

상위 설계 : 아키텍처 설계, 예비 설계, 시스템의 전체적인 구조, 구조, DB, 인터페이스(정의, 사용자 인터페이스 설계)

하위 설계 : 모듈 설계, 상세 설계, 시스템의 내부 구조 및 행위, 컴포넌트, 자료구조, 알고리즘

2. 다음 ( ) 안에 들어갈 내용으로 옳은 것은?

컴포넌트 설계 시 "( )에 의한 설계"를 따를 경우, 해당 명세에서는

(1) 컴포넌트의 오퍼레이션 사용 전에 참이 되어야 할 선행 조건(선행 조건)

(2) 사용 후 만족되어야 할 결과 조건(결과 조건)

(3) 오퍼레이션이 실행되는 동안 항상 만족되어야 할 불변 조건 등이 포함되어야 한다.(불변 조건)

① 협약(Contract) ② 프로토콜(Protocol)

3. 소프트웨어 설계에서 사용되는 대표적인 추상화(Abstraction) 기법이 아닌 것은?

① 자료 추상화 ② 제어 추상화

③ 과정 추상화 ④ 강도 추상화

설명 : 추상화란 문제의 전체적이고 포괄적인 개념을 설계한 후 차례로 세분화하여 구체화시켜 나가는 것이다.

4. 객체지향 설계에서 정보 은닉(Information Hiding)과 관련한 설명으로 틀린 것은?

① 필요하지 않은 정보는 접근할 수 없도록 하여 한 모듈 또는 하부 시스템이 다른 모듈의 구현에 영향을 받지 않게 설계되는 것을 의미 한다.

② 모듈들 사이의 독립성을 유지시키는 데 도움이 된다.

③ 설계에서 은닉되어야 할 기본 정보로는 IP주소와 같은 물리적 코드, 상세 데이터 구조 등이 있다.

④ 모듈 내부의 자료 구조와 접근 동작들에만 수정을 국한하기 때문에 요구사항 등 변화에 따른 수정이 불가능하다.

설명 : 정보 은닉이란 한 모듈 내부에 포함된 절차와 자료들의 정보가 감추어져 다른 모듈이 접근하거나 변경하지 못하도록 하는 기법이다. 정보 은닉을 통해 모듈을 독립적으로 수행할 수 있고, 하나의 모듈이

1. 애플리케이션 설계- SEC_01(소프트웨어 아키텍처) 기출 및 예상 문제

기출 및 예상 문제(소프트웨어 아키텍처)

5. 소프트웨어 아키텍처 설계에서 시스템 품질 속성이 아닌 것은?

① 가용성(Availability)

② 독립성(Isolation)

③ 변경 용이성(Modifiability)

④ 사용성(Usability)

설명

시스템 측면 : 성능, 보안, 가용성, 기능성, 사용성, 변경 용이성, 확장성, 테스트 용이성, 배치성, 안정성이 있다.

6. 아키텍처 설계 과정이 올바른 순서로 나열된 것은?

㉮ 설계 목표 설정

㉯ 시스템 타입 결정

㉰ 스타일 적용 및 커스터마이즈

㉱ 서브 시스템의 기능, 인터페이스 동작 작성

㉲ 아키텍처 설계 검토

① 가 -> 나 -> 다 -> 라 -> 마

7. 모듈화(Modularity)와 관련한 설명으로 틀린 것은?

① 소프트웨어의 모듈은 프로그래밍 언어에서 Subroutine, Function 등으로 표현될 수 있다.

② 모듈의 수가 증가하면 상대적으로 각 크기가 커지며, 모듈 사이의 상호교류가 감소하여 과부하(Overload) 현상이 나타난다.

③ 모듈화는 시스템을 지능적으로 관리할 수 있도록 해주며, 복잡도 문제를 해결하는 데 도움을 준다.

④ 모듈화는 시스템의 유지 보수와 수정을 용이하게 한다.

설명 : 모듈화란 소프트웨어의 성능을 향상시키거나, 시스템의 수정 및 재사용, 유지 관리 등이 용이하도록 시스템의 기능들을 모듈 단위로 나누는 것이다.

8. 소프트웨어 아키텍처 설계에 대한 설명으로 옳지 않은 것은?

① 이해 관계자들의 의사소통 도구로 활용된다.

② 설계된 모듈을 프로그래밍 언어를 통해 구현한다.(구현 단계)

③ 애플리케이션을 모듈로 분할하고, 모듈 간 인터페이스를 결정하는 과정이다.

④ 기본 원리에는 모듈화, 추상화, 단계적 분해, 정보은닉이 있다.

1. 애플리케이션 설계- SEC_01(소프트웨어 아키텍처) 기출 및 예상 문제

기출 및 예상 문제(소프트웨어 아키텍처)

9. 소프트웨어 모듈화의 장점이 아닌 것은?

① 오류의 파급 효과를 최소화한다.

② 기능의 분리가 가능하여 인터페이스가 복잡하다.

③ 모듈의 재사용 가능으로 개발과 유지보수가 용이하다.

④ 프로그램의 효율적인 관리가 가능하다.

설명 : 기능의 분리가 가능하여 인터페이스가 단순해진다.

1. 애플리케이션 설계-SEC_02(아키텍처 패턴)

1) 아키텍처 패턴(Patterns)의 개요

; 아키텍처 패턴은 아키텍처를 설계할 때 참조할 수 있는 전형적인 해결 방식 또는 예제를 의미한다.

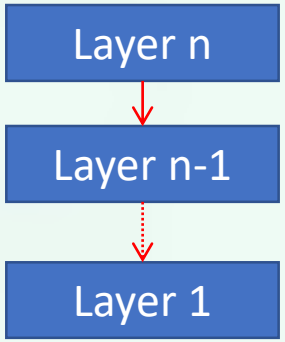
- 아키텍처 패턴은 소프트웨어 시스템의 구조를 구성하기 위한 기본적인 윤곽을 제시한다.
- 아키텍처 패턴에는 서브시스템들과 그 역할이 정의되어 있으며, 서브시스템 사이의 관계와 여러 규칙 지침 등이 포함되어 있다.
- 아키텍처 패턴을 아키텍처 스타일 또는 표준 아키텍처라고도 한다.
- 아키텍처 패턴의 장점
  - 시행착오를 줄여 개발 시간을 단축시키고, 고품질의 소프트웨어를 생산할 수 있다.
  - 검증된 구조로 개발하기 때문에 안정적인 개발이 가능하다.
  - 이해관계자들이 공통된 아키텍처를 공유할 수 있어 의사소통이 간편해진다.
  - 시스템의 구조를 이해하는 것이 쉬워 개발에 참여하지 않은 사람도 손쉽게 유지보수를 수행할 수 있다.
  - 시스템의 특성을 개발 전에 예측하는 것이 가능해진다.
- 아키텍처 패턴의 종류에는 레이어 패턴, 클라이언트-서버 패턴, 파이프-필터 패턴, 모델-뷰-컨트롤러 패턴 등이 있다.

1. 애플리케이션 설계-SEC_02(아키텍처 패턴)

2) 레이어 패턴(Layers Pattern)

; 레이어 패턴은 시스템을 계층(Layer)으로 구분하여 구성하는 고전적인 방법 중의 하나다.

- 레이어 패턴은 각각의 서브시스템들이 계층 구조를 이루며, 하위 계층은 상위 계층에 대한 서비스 제공자가 되고, 상위 계층은 하위 계층의 클라이언트가 된다.
- 레이어 패턴은 서로 마주보는 두 개의 계층 사이에서만 상호작용이 이루어지며, 변경 사항을 적용할 때도 서로 마주보는 두 개의 계층에만 영향을 미치므로 변경 작업이 용이하다.
- 레이어 패턴은 특정 계층만을 교체해 시스템을 개선하는 것이 가능하다.
- 대표적으로 OSI 참조 모델이 있다.



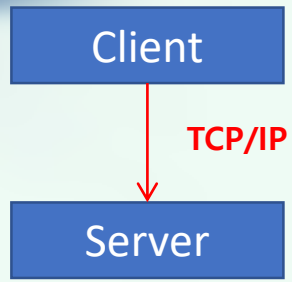
OSI 참조 모델은 국제 표준화 기구(ISO)에서 네트워크 프로토콜을 계층별로 구분한 모델로 물리 계층, 데이터 링크 계층, 네트워크 계층, 전송 계층, 세션 계층, 표현 계층, 응용 계층으로 구성되어 있다.

1. 애플리케이션 설계-SEC_02(아키텍처 패턴)

3) 클라이언트-서버 패턴(Client-Server Pattern)

; 클라이언트-서버 패턴은 하나의 서버 컴포넌트와 다수의 클라이언트 컴포넌트로 구성되는 패턴이다.

- 클라이언트-서버 패턴에서 사용자는 클라이언트와만 의사소통을 한다. 즉 사용자가 클라이언트를 통해 서버에 요청하고 클라이언트가 응답을 받아 사용자에게 제공하는 방식으로 서비스를 제공한다.
- 서버는 클라이언트의 요청에 대비해 항상 대기 상태를 유지해야 한다.
- 클라이언트나 서버는 요청과 응답을 받기 위해 동기화되는 경우를 제외하고는 서로 독립적이다.



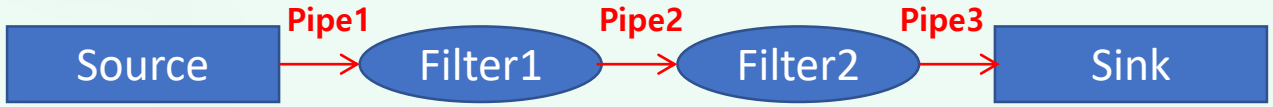
컴포넌트(Component) : 컴포넌트는 독립적인 업무 또는 기능을 수행하는 실행코드 기반으로 작성된 모듈을 의미한다
TCP/IP : 컴퓨터가 서로 통신하는 경우, 특정 규칙이나 프로토콜을 사용하여 순서대로 데이터를 전송 및 수신할 수 있다. 전 세계를 통해 가장 일상적으로 사용되는 프로토콜 세트 중 하나가 TCP/IP(Transmission Control Protocol/Internet Protocol)이다. (그러나 유럽에서는 대부분 X.25 프로토콜을 사용한다.) TCP/IP의 사용에 있어서 일부 일반적인 기능은 메일, 컴퓨터 간 파일 전송, 원격 로그인 등이다.
동기화(Synchronized) : 사전적 의미로는 "작업들 사이의 수행 시기를 맞추는 것"으로 되어 있다. 그리고 프로그래밍 언어에서 공유 데이터가 사용되어 동기화가 필요한 부분을 임계영역(critical section)이라고 부르며, 임계영역에 synchronized 키워드를 사용하여 여러 스레드가 동시에 접근하는 것을 금지함으로써 동기화를 할 수 있다.
일반적인 동기화의 의미는 인터넷에 저장되어 있는 자신의 데이터 자료와 자신의 스마트 폰 또는 컴퓨터의 데이터를 서로 주고 받아서 정보의 최신성을 동일하도록 만드는 것이다.

1. 애플리케이션 설계-SEC_02(아키텍처 패턴)

4) 파이프-필터 패턴(Pipe-Filter Pattern)

; 파이프-필터 패턴은 데이터 스트림 절차의 각 단계를 필터(Filter) 컴포넌트로 캡슐화하여 파이프(Pipe)를 통해 데이터를 전송하는 패턴이다.

- 필터 컴포넌트는 재사용성이 좋고, 추가가 쉬워 확장이 용이하다.
- 필터 컴포넌트들을 재배치하여 다양한 파이프라인을 구축하는 것이 가능하다.
- 파이프-필터 패턴은 데이터 변환, 버퍼링, 동기화 등에 주로 사용된다.
- 필터 간 데이터 이동 시 데이터 변환으로 인한 오버헤드가 발생한다.
- 대표적으로 UNIX의 쉘(Shell)이 있다.



데이터 스트림(Data Stream) : 데이터 스트림은 데이터가 송·수신되거나 처리되는 일련의 연속적인 흐름이며 데이터가 흐르는 공간이다.
파이프라인(Pipeline) : 파이프라인은 필터와 파이프를 통해 처리되는 일련의 처리 과정이다.
버퍼(Buffer) : 버퍼링을 하기 위한 메모리(공간)
버퍼링(Buffering) : 컴퓨터 시스템에서의 처리를 어떤 장치로 부터 다른 장치로 데이터를 일방통행으로 전송할 때 양쪽의 속도 차를 수정하기 위해서 중간에 데이터를 일시적으로 기억 장치에 축적하는 방법이다.

1. 애플리케이션 설계-SEC_02(아키텍처 패턴)

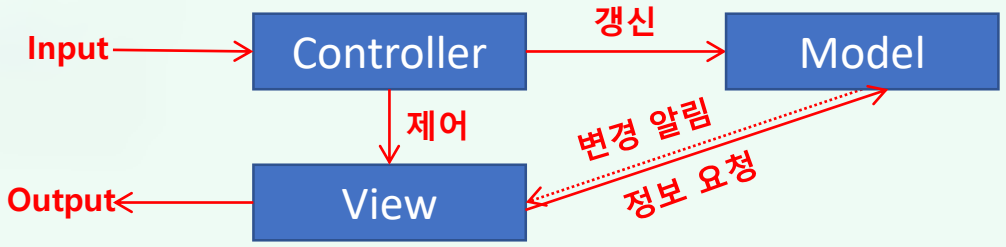
5) 모델-뷰-컨트롤러 패턴(Model-View-Controller Pattern)

; 모델-뷰-컨트롤러 패턴은 서브시스템을 3개의 부분으로 구조화하는 패턴이며, 각 부분의 역할은 다음과 같다.

- 모델(Model) : 서브 시스템의 핵심 기능과 데이터를 보관한다.
- 뷰(View) : 사용자에게 정보를 표시한다.
- 컨트롤러(Controller) : 사용자로부터 입력된 변경 요청을 처리하기 위해 모델에게 명령을 보낸다.

● 모델-뷰-컨트롤러 패턴의 각 부분은 별도의 컴포넌트로 분리되어 있으므로 서로 영향을 받지 않고 개발 작업을 수행할 수 있다.

● 모델-뷰-컨트롤러 패턴에서는 여러 개의 뷰를 만들 수 있으므로 한 개의 모델에 대해 여러 개의 뷰를 필요로 하는 대화형 애플리케이션에 적합하다.



대화형 애플리케이션 : 대화형 애플리케이션은 온라인 쇼핑몰 사이트나 스마트폰 앱과 같이 사용자의 요구가 발생하면 시스템이 이를 처리하고 반응하는 소프트웨어를 의미한다.

1. 애플리케이션 설계-SEC_02(아키텍처 패턴)

6) 기타 패턴

마스터-슬레이브 패턴 (Master-Slave Pattern)	•마스터 컴포넌트는 동일한 구조의 슬레이브 컴포넌트로 작업을 분할한 후, 슬레이브 컴포넌트에서 처리된 결과물을 다시 돌려받는 방식으로 작업을 수행하는 패턴이다. •마스터 컴포넌트는 모든 작업의 주체이고, 슬레이브 컴포넌트는 마스터 컴포넌트의 지시에 따라 작업을 수행하여 결과를 반환한다. •장애 허용 시스템과 병렬 컴퓨팅 시스템에서 주로 활용된다.
브로커 패턴 (Broker Pattern)	•사용자가 원하는 서비스와 특성을 브로커 컴포넌트에 요청하면 브로커 컴포넌트가 요청에 맞는 컴포넌트와 사용자를 연결해준다. •원격 서비스 호출에 응답하는 컴포넌트들이 여러 개 있을 때 적합한 패턴이다. •분산환경 시스템에서 주로 활용된다.
피어-투-피어 패턴 (Peer-To-Peer Pattern)	•피어(Peer)를 하나의 컴포넌트로 간주하며, 각 피어는 서비스를 호출하는 클라이언트가 될 수도 서비스를 제공하는 서버가 될 수도 있는 패턴이다. •피어-투-피어 패턴에서 클라이언트와 서버는 전형적인 멀티스레딩 방식을 사용한다.
이벤트-버스 패턴 (Event-Bus Pattern)	•소스가 특정 채널에 이벤트 메시지를 발행(Publish)하면, 해당 채널을 구독(Subscribe)한 리스너들이 메시지를 받아 이벤트를 처리하는 방식이다. •4가지 주요 컴포넌트 - 이벤트를 생성하는 소스(Source) - 이벤트를 수행하는 리스너(Listener) - 이벤트의 통로인 채널(Channel) - 채널들을 관리하는 버스(Bus)
블랙보드 패턴 (Blackboard Pattern)	•모든 컴포넌트들이 공유 데이터 저장소와 블랙보드 컴포넌트에 접근이 가능한 형태로, 컴포넌트들은 검색을 통해 블랙보드에서 원하는 데이터를 찾을 수 있다. •해결책이 명확하지 않은 문제를 처리하는데 유용한 패턴이다. •음성 인식, 차량 식별, 신호 해석 등에 주로 활용된다.
인터프리터 패턴 (Interpreter Pattern)	•프로그램 코드의 각 라인을 수행하는 방법을 지정하고, 기호마다 클래스를 갖도록 구성된다. •특정 언어로 작성된 프로그램 코드를 해석하는 컴포넌트를 설계할 때 사용 되어진다.

1. 애플리케이션 설계- SEC_02(아키텍처 패턴) 기출 및 예상 문제

기출 및 예상 문제(아키텍처 패턴)

1. 파이프 필터 형태의 소프트웨어 아키텍처에 대한 설명 으로 옳은 것은?

① 노드와 간선으로 구성된다.

② 서브시스템이 입력 데이터를 받아 처리하고 결과를 다음 서브시스템으로 넘겨주는 과정을 반복한다.

③ 계층 모델이라고도 한다.

④ 3개의 서브시스템(모델, 뷰, 제어)으로 구성되어 있다.

설명 : 파이프 필터 패턴은 데이터 스트림 절차의 각 단계를 필터 컴포넌트로 캡슐화하여 파이프를 통해서 데이터를 전송하는 패턴이다.

2. 소프트웨어 아키텍처와 관련한 설명으로 틀린 것은?

① 파이프 필터 아키텍처에서 데이터는 파이프를 통해 양방향으로 흐르며, 필터 이동 시 오버헤드가 발생하지 않는다.(단방향, 오버헤드가 데이터 변환 시 발생 가능성)

② 외부에서 인식할 수 있는 특성이 담긴 소프트웨어의 골격이 되는 기본 구조로 볼 수 있다.

③ 데이터 중심 아키텍처는 공유 데이터 저장소를 통해

3. 서브시스템이 입력 데이터를 받아 처리하고 결과를 다른 시스템에 보내는 작업이 반복되는 아키텍처 스타일은?

① 클라이언트 서버 구조 ② 계층 구조

③ MVC 구조 ④ 파이프 필터 구조

설명 : 시스템이 파이프처럼 연결되어 있어서 앞 시스템의 처리 결과물을 파이프를 통해 전달받아 처리한 후, 그 결과물을 다시 파이프를 통해서 다음 시스템으로 넘겨주는 패턴을 반복하는 아키텍쳐 스타일은 파이프-필터 패턴이다.

4. 분산시스템을 위한 마스터-슬레이브(Master-Slave) 아키텍처에 대한 설명으로 틀린 것은?

① 일반적으로 실시간 시스템에서 사용된다.

② 마스터 프로세스는 일반적으로 연산, 통신, 조정을 책임진다.

③ 슬레이브 프로세스는 데이터 수집 기능을 수행할 수 없다.

④ 마스터 프로세스는 슬레이브 프로세스들을 제어할 수 있다.

설명 : 마스터와 슬레이브는 구조가 동일하므로 기능도 동일하게 수행할 수 있고, 다만 연산, 통신, 조정 기능은 슬레이브 제어를 위해서 일반적으로 마스터가 수행한다. 실시간 시스템, 병렬 컴퓨팅, 장애 허용 시스템에서 주로 활용된다.

1. 애플리케이션 설계- SEC_02(아키텍처 패턴) 기출 및 예상 문제

기출 및 예상 문제(아키텍처 패턴)

5. 네트워크 프로토콜의 OSI 참조 모델과 가장 관련이 깊은 아키텍처 모델은?

① Peer-To-Peer Model ② MVC Model

③ Layers Model ④ Client-Server Model

설명 : 레이어 패턴은 시스템을 계층(Layer)으로 구분하여 구성하는 고전적인 방법 중 하나이다.
하위계층은 상위계층에 서비스 제공자가 되고, 상위계층은 하위계층의 클라이언트가 된다. 서로 마주보는 두 개의 계층 사이에서만 상호작용이 이루어지며 변경사항을 적용할 때도 마주보는 2개의 계층에만 영향을 미친다.
대표적으로 OSI 참조 모델이 있다.

6. 소프트웨어 아키텍처 모델 중 MVC와 관련한 설명으로 틀린 것은?

① MVC 모델은 사용자 인터페이스를 담당하는 계층의 응집도를 높일 수 있고, 여러 개의 다른 UI를 만들어 그 사이 에 결합도를 낮출 수 있다.

② 모델(Model)은 뷰(View)와 제어(Controller)사이에서

7. 아키텍처 패턴(Architecture Pattern)에 대한 설명 중 가장 옳지 않은 것은?

① 소프트웨어 초기 설계에서 발생하는 문제들을 해결하기 위한 전형적인 해결 방식들을 의미한다.

② 검증된 구조로 개발하기 때문에 오류가 적어 개발시간을 단축할 수 있다.

③ 서브시스템들에 대한 역할을 정의하고 있지만, 그들 간의 인터페이스에 대한 지침은 없다.

④ 시스템에 대한 이해가 쉬워지고, 특성을 예측할 수 있게 된다.

8. 다음 중 클라이언트-서버(Client-Server) 모델에 대한 설명으로 가장 거리가 먼 것은?

① 사용자는 클라이언트를 통해서 요청을 전달하며, 서버는 이에 응답하는 방식이다.

② 서버는 클라이언트의 요청에 대비하여 항상 대기 상태를 유지한다.

③ 서버와 클라이언트는 서로 독립적이다.

④ 다수의 서버와 하나의 클라이언트로 구성되는 패턴으로 분산 환경 시스템에 적합하다.

설명 : 하나의 서버와 다수의 클라이언트로 구성되는 패턴이고

1. 애플리케이션 설계- SEC_02(아키텍처 패턴) 기출 및 예상 문제

기출 및 예상 문제(아키텍처 패턴)

9. 여러 컴포넌트들 중 각 컴포넌트들이 서비스를 제공하는 서버가 될 수도 있고, 서비스를 요청하는 클라이언트도 될 수 있는 패턴으로 전형적인 멀티스레딩을 사용하는 방식의 패턴을 무엇이라 하는가?

① 클라이언트-서버 ② 블랙보드

③ 이벤트-버스 ④ 피어-투-피어

설명 : 피어 투 피어 패턴은 여러 컴포넌트들 중 각 컴포넌트들이 서비스를 제공하는 서버가 될 수도 있고, 서비스 를 요청하는 클라이언트도 될 수 있는 패턴으로 전형적인 멀티스레딩을 사용하는 방식이다.

10. 다음 중, 이벤트-버스 패턴(Event-Bus Pattern)의 주요 컴포넌트가 아닌 것은?

① 소스 ② 리스너

③ 채널 ④ 버퍼

설명

이벤트-버스 패턴의 컴포넌트의 종류

- 소스 : 이벤트를 생성함

1. 애플리케이션 설계-SEC_03(객체 지향(Object-Oriented))

1) 객체지향의 개요

; 객체지향은 현실 세계의 개체(Entity)를 기계의 부품처럼 하나의 객체(Object)로 만들어, 기계적인 부품들을 조립하여 제품을 만들 듯이 소프트웨어를 개발할 때에도 객체들을 조립해서 작성할 수 있는 기법을 의미한다.

- 객체지향 기법은 구조적 기법의 문제점으로 인한 소프트웨어 위기의 해결책으로 채택되어 사용되고 있다.
- 객체지향은 소프트웨어의 재사용 및 확장이 용이하여 고품질의 소프트웨어를 빠르게 개발할 수 있고 유지보수가 쉽다.
- 객체지향은 복잡한 구조를 단계적·계층적으로 표현하고, 멀티미디어 데이터 및 병렬 처리를 지원한다.
- 객체지향은 현실 세계를 모형화하므로 사용자와 개발자가 쉽게 이해할 수 있다.
- 객체지향의 주요 구성 요소와 개념에는 객체(Object), 클래스(Class), 캡슐화(Encapsulation), 상속(Inheritance), 다형성(Polymorphism), 연관성(Relationship)이 있다.

현실 세계의 개체 : 사람, 자동차, 컴퓨터, 고양이 등과 같이 우리 주위에서 사용되는 물질적이거나 개념적인 것으로 명사로 사용된다.
구조적 기법 : 프로시저에 근간을 두고 하나의 커다란 작업을 여러 개의 작은 작업으로 분할하고 분할된 각각의 소작업을 수행하는 모듈 을 작성한 다음 이들을 한 곳 에 모아 큰 작업을 수행하는 하나의 완벽한 프로그램으로 작성하는 기법이다.
구조적 기법의 문제점은 유지보수는 고려하지 않고 개발 공정에만 너무 집중되며 개발이 시작된 이후 추가적인 요구사항에 대응하기 어렵다. 아울러, 재사용이 어려워 이전에 개발한 소프트웨어와 유사한 소프트웨어를 다시 개발할 때도 시간과 인력이 동일하게 소모된다.

1. 애플리케이션 설계-SEC_03(객체 지향(Object-Oriented))

2) 객체(Object)

; 객체는 데이터와 데이터를 처리하는 함수를 묶어 놓은(캡슐화한) 하나의 소프트웨어 모듈이다.

데이터	• 객체가 가지고 있는 정보로 속성이나 상태, 분류 등을 나타낸다. • 속성(Attribute), 상태, 변수, 상수, 자료 구조라고도 한다.
함수(메소드)	• 객체가 수행하는 기능으로 객체가 갖는 데이터(속성,상태)를 처리하는 알고리즘이다. • 객체의 상태를 참조하거나 변경하는 수단이 되는 것으로 메소드(Method, 행위), 서비스(Service), 동작(Operation), 연산이라고도 한다.

● 객체의 특성

- 객체는 독립적으로 식별 가능한 이름을 가지고 있다.

  예) 자동차는 번호판으로 다른 자동차 객체와 구별된다.

- 객체가 가질 수 있는 조건을 상태(State)라고 하는데, 일반적으로 상태는 시간에 따라 변한다.

  예) 자동차는 '정지', '이동' 등의 상태가 존재하며, 이러한 '정지'와 '이동'의 상태는 고정된 것이 아니라 시간에 따라 변한다.

- 객체와 객체는 상호 연관성에 의한 관계가 형성된다.

  예) 화재 발생 시 소방차, 구급차, 경찰차는 긴밀하게 협조하여 화재를 진압하고 환자를 이송하며, 교통을 정리하는 관계가 형성된다.

1. 애플리케이션 설계-SEC_03(객체 지향(Object-Oriented))

2) 객체(Object)

- 객체의 특성
  - 객체가 반응할 수 있는 메시지(Message)의 집합을 행위라고 하며, 객체는 행위의 특징을 나타낼 수 있다.

    예) 자동차 객체는 '가속 페달을 밟는 행위를 하면 가속'하는 특징을 나타내고, '브레이크를 밟는 행위를 하면 '감속'하는 특징을 나타낸다.
  - 객체는 일정한 기억 장소를 가지고 있다.

    예) 자동차는 주차장에 있거나 도로 위에 있거나, 일정한 물리적 공간을 점유한다.
- 객체의 메소드는 다른 객체로부터 메시지를 받았을 때 정해진 기능을 수행한다.

메시지 : 객체들 간에 상호작용을 하는 데 사용되는 수단으로 객체에게 어떤 행위를 하도록 지시하는 명령 또는 요구사항이다.

1. 애플리케이션 설계-SEC_03(객체 지향(Object-Oriented))

3) 클래스(Class)

; 클래스는 공통된 속성과 연산(행위)을 갖는 객체의 집합으로, 객체의 일반적인 타입(Type)을 의미한다.

- 클래스는 각각의 객체들이 갖는 속성과 연산을 정의하고 있는 틀이다.
- 클래스는 객체지향 프로그램에서 데이터를 추상화하는 단위이다.
- 클래스에 속한 각각의 객체를 인스턴스(Instance)라 하며, 클래스로부터 새로운 객체를 생성하는 것을 인스턴스화(Instantiation)라고 한다.
- 동일 클래스에 속한 각각의 객체(인스턴스)들은 공통된 속성과 행위를 가지고 있으면서, 그 속성에 대한 정보가 서로 달라서 동일 기능을 하는 여러 가지 객체를 나타내게 된다.
- 최상위 클래스는 상위 클래스를 갖지 않는 클래스를 의미한다.
- 슈퍼 클래스(Super Class)는 특정 클래스의 상위(부모)클래스이고, 서브 클래스(Sub Class)는 특정 클래스의 하위(자식) 클래스를 의미한다.

1. 애플리케이션 설계-SEC_03(객체 지향(Object-Oriented))

4) 캡슐화(Encapsulation)

; 캡슐화는 데이터(속성)와 데이터를 처리하는 함수를 하나로 묶는 것을 의미한다.

- 캡슐화된 객체는 인터페이스를 제외한 세부 내용이 은폐(정보 은닉)되어 외부에서의 접근이 제한적이기 때문에 외부 모듈의 변경으로 인한 파급 효과가 적다.
- 캡슐화된 객체들은 재사용이 용이하다.
- 객체들 간의 메시지를 주고 받을 때 상대 객체의 세부 내용은 알 필요가 없으므로 인터페이스가 단순해지고, 객체 간의 결합도가 낮아진다.

5) 상속(Inheritance)

; 상속은 이미 정의된 상위 클래스(부모 클래스)의 모든 속성과 연산을 하위 클래스(자식 클래스)가 물려받는 것이다.

- 상속을 이용하면 하위 클래스는 상위 클래스의 모든 속성과 연산을 자신의 클래스 내에서 다시 정의하지 않고서도 즉시 자신의 속성으로 사용할 수 있다.
- 하위 클래스는 상위 클래스로부터 상속받은 속성과 연산 외에 새로운 속성과 연산을 첨가하여 사용할 수 있다.

1. 애플리케이션 설계-SEC_03(객체 지향(Object-Oriented))

5) 상속(Inheritance)

- 상위 클래스의 속성과 연산을 하위 클래스가 사용할 수 있기 때문에 객체와 클래스의 재사용, 즉 소프트웨어의 재사용(Reuse)을 높이는 중요한 개념이다.
- 다중 상속(Multiple Inheritance) : 한 개의 클래스가 두 개 이상의 상위 클래스로부터 속성과 연산을 상속받는 것이다. 다중 상속은 클래스 계층을 복잡하게 만들어 상속 순서 추적이 어렵고, 상위 클래스의 변경이 하위 클래스에 의도하지 않은 영향을 미칠 수도 있어 다중 상속을 허용하지 않는 프로그래밍 언어들도 있다. 다중 상속이 가능한 프로그래밍 언어에서도 다중 상속을 이용할 때는 이를 고려하여 신중히 사용해야 한다.

1. 애플리케이션 설계-SEC_03(객체 지향(Object-Oriented))

6) 다형성(Polymorphism)

; 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 능력을 의미한다.

- 객체(클래스)들은 동일한 메소드 명을 사용하며 같은 의미의 응답을 한다.
- 응용 프로그램 상에서 하나의 함수나 연산자가 두 개 이상의 서로 다른 클래스의 인스턴스들을 같은 클래스에 속한 인스턴스처럼 수행할 수 있도록 하는 것이다.

예1) '+'연산자의 경우 숫자 클래스에서는 덧셈, 문자 클래스에서는 문자열의 연결 기능으로 사용된다.

예2) 오버로딩(Overloading) 기능의 경우 메소드(Method)의 이름은 같지만 인수를 받는 자료형과 개수를 달리하여 여러 기능을 정의할 수 있다.

예3) 오버라이딩(Overriding, 메소드 재정의) 기능의 경우 상위 클래스에서 정의한 메소드(Method)와 이름은 같지만 메소드 안의 실행 코드를 달리하여 자식 클래스에서 재정의해서 사용할 수 있다.

1. 애플리케이션 설계-SEC_03(객체 지향(Object-Oriented))

7) 연관성(Relationship)

; 연관성은 두 개 이상의 객체(클래스)들이 상호 참조하는 관계를 말하며 종류는 다음과 같다.

종류	의미	특징
is member of	연관화(Association)	2개 이상의 객체가 상호 관련되어 있음을 의미함
is instance of	분류화(Classification)	일한 형의 특성을 갖는 객체들을 모아 구성하는 것
is part of	집단화(Aggregation)	관련 있는 객체들을 묶어 하나의 상위 객체를 구성하는 것
is a	일반화(Generalization)	공통적인 성질들로 추상화 한 상위 객체를 구성하는 것
	특수화/ 상세화(Specialization	상위 객체를 구체화 하여 하위 객체를 구성하는 것

1. 애플리케이션 설계-SEC_03(객체 지향(Object-Oriented)) 기출 문제

기출 및 예상 문제(객체 지향(Object-Oriented))

1. 객체에 대한 설명으로 틀린 것은?

① 객체는 상태, 동작, 고유 식별자를 가진 모든 것이라 할 수 있다.

② 객체는 공통 속성을 공유하는 클래스들의 집합이다.

③ 객체는 필요한 자료 구조와 이에 수행되는 함수들을 가진 하나의 독립된 존재이다.

④ 객체의 상태는 속성값에 의해 정의된다.

설명 : 객체(Object)는 데이터(속성, 필드, 멤버 변수)와 데이터를 처리하는 함수(메소드, 기능)를 묶어 놓은(캡슐화) 하나의 소프트웨어 모듈이다.

객체가 클래스의 집합이 아니라, 클래스가 공통된 속성과 연산(기능, 메소드, 행위)을 갖는 객체의 집합이다.

2. 객체지향개념 중 하나 이상의 유사한 객체들을 묶어 공통된 특성을 표현한 데이터 추상화를 의미하는 것은?

① Method ② Class

③ Field ④ Message

설명 : Class는 공통된 속성과 연산(기능)을 갖는 객체의

3. 객체지향의 주요 개념에 대한 설명으로 틀린 것은?

① 캡슐화는 상위 클래스에서 속성이나 연산을 전달받아 새로운 형태의 클래스로 확장하여 사용하는 것을 의미한다.

② 객체는 실세계에 존재하거나 생각할 수 있는 것을 말한다.

③ 클래스는 하나 이상의 유사한 객체들을 묶어 공통된 특성을 표현한 것이다.

④ 다형성은 상속받은 여러 개의 하위 객체들이 다른 형태의 특성을 갖는 객체로 이용될 수 있는 성질이다.

설명

상속 : 상위 클래스에서 속성이나 연산을 전달받아 새로운 형태의 클래스로 확장하여 사용하는 것을 의미한다.

4. 객체지향에서 정보은닉과 가장 밀접한 관계가 있는 것은?

① Encapsulation ② Class

③ Method ④ Instance

설명 : Instance는 클래스의 속한 각각의 객체를 Instance라고 하며, 클래스로부터 새로운 객체를 생성하는 것을 인스턴스화라고 한다.

Instance는 클래스가 메모리(힙)에 올라가는 것을 말한다.

Encapsulation(캡슐화)는 데이터(속성)와 데이터를 처리하는

1. 애플리케이션 설계-SEC_03(객체 지향(Object-Oriented)) 기출 문제

기출 및 예상 문제(객체 지향(Object-Oriented))

5. 객체지향 기법에서 클래스들 사이의 '부분 전체(Part-Whole)' 관계 또는 부분(is-a-part-of)'의 관계로 설명되는 연관성을 나타내는 용어는?

① 일반화 ② 추상화

③ 캡슐화 ④ 집단화

설명

is a : 일반화, 특수화, 상세화

is member of : 연관화

is part of : 집단화

is instance of : 분류화

6. 객체에게 어떤 행위를 하도록 지시하는 명령은?

① Class ② Package

③ Object ④ Message

설명 : Message는 객체들 간에 상호작용을 하는데 사용되는 수단으로 객체에게 어떤 행위를 하도록 지시하는 명령 또는 요구사항이다.

7. 객체지향 기법에서 같은 클래스에 속한 각각의 객체를 의미하는 것은?

① Instance ② Message

③ Method ④ Module

설명 : 클래스에 속한 각각의 객체를 인스턴스(Instance)라 하며, 클래스로부터 새로운 객체를 생성하는 것을 인스턴스화(Instantiation)라고 한다.

8. 객체지향 개념에서 연관된 데이터와 함수를 함께 묶어 외부와 경계 를 만들고 필요한 인터페이스만을 밖으로 드러내는 과정은?

① 메시지(Message) ② 캡슐화(Encapsulation)

③ 다형성(Polymorphism) ④ 상속(Inheritance)

1. 애플리케이션 설계-SEC_03(객체 지향(Object-Oriented)) 기출 문제

기출 및 예상 문제(객체 지향(Object-Oriented))

9. 객체지향 기법에서 상위 클래스의 메소드와 속성을 하위 클래스가 물려받는 것을 의미하는 것은?

① Abstraction(추상화) ② Polymorphism(다형성)

③ Encapsulation(캡슐화) ④ Inheritance(상속)

10. 객체지향 개념에서 다형성(Polymorphism)과 관련한 설명으로 틀린 것은?

① 다형성은 현재 코드를 변경하지 않고 새로운 클래스를 쉽게 추가할 수 있게 한다.

② 다형성이란 여러 가지 형태를 가지고 있다는 의미로, 여러 형태를 받아들일 수 있는 특징을 말한다.

③ 메소드 오버라이딩(Overriding)은 상위 클래스에서 정의한 일반 메소드의 구현을 하위 클래스에서 무시하고 재정의할 수 있다.

④ 메소드 오버로딩(Overloading)의 경우 매개 변수 타입은 동일하지만 메소드 명을 다르게 함으로써 구현, 구분할 수 있다.

설명 : 메소드 오버로딩(Overloading)의 경우는 메서드 명 은

1. 애플리케이션 설계-SEC_04(객체지향 분석 및 설계)

1) 객체지향 분석의 개념

; 객체지향 분석(OOA: Object Oriented Analysis)은 사용자의 요구사항을 분석하여 요구된 문제와 관련된 모든 클래스(객체), 이와 연관된 속성과 연산, 그들 간의 관계 등을 정의하여 모델링 하는 작업이다.

- 소프트웨어를 개발하기 위한 비즈니스(업무)를 객체와 속성, 클래스와 멤버, 전체와 부분 등으로 나누어서 분석한다.
- 분석가에게 중요한 모델링 구성 요소인 클래스, 객체, 속성, 연산들을 표현해서 문제를 모형화할 수 있게 해준다.
- 객체는 클래스로부터 인스턴스화되고, 이 클래스를 식별하는 것이 객체지향 분석의 주요한 목적이다.

1. 애플리케이션 설계-SEC_04(객체지향 분석 및 설계)

2) 객체지향 분석의 방법론

; 객체지향 분석을 위한 여러 방법론이 제시되었으며 각 방법론은 다음과 같다.

- Rumbaugh(럼바우) 방법 : 가장 일반적으로 사용되는 방법으로 분석 활동을 객체 모델, 동적 모델, 기능 모델로 나누어 수행하는 방법이다.
- Booch(부치) 방법 : 미시적(Micro) 개발 프로세스와 거시적(Macro) 개발 프로세스를 모두 사용하는 분석 방법으로, 클래스와 객체들을 분석 및 식별하고 클래스의 속성과 연산을 정의한다.
- Jacobson 방법 : Use Case를 강조하여 사용하는 분석 방법이다.
- Coad와 Yourdon 방법 : E-R 다이어그램을 사용하여 객체의 행위를 모델링하며, 객체 식별, 구조 식별, 주제 정의, 속성과 인스턴스 연결 정의, 연산과 메시지 연결 정의 등의 과정으로 구성하는 기법이다.
- Wirfs Brock 방법 : 분석과 설계 간의 구분이 없고, 고객 명세서를 평가해서 설계 작업까지 연속적으로 수행하는 기법이다.

Use Case(사용 사례) : 사용자, 외부 시스템, 다른 요소들이 시스템과 상호 작용하는 방법을 기술한 설명

1. 애플리케이션 설계-SEC_04(객체지향 분석 및 설계)

3) 럼바우(Rumbaugh)의 분석 기법

; 럼바우의 분석 기법은 모든 소프트웨어 구성 요소를 그래픽 표기법을 이용하여 모델링 하는 기법으로, 객체 모델링 기법(OMT, Object-Modeling Technique)이라고도 한다.

● 분석 활동은 '객체 모델링 - 동적 모델링 - 기능 모델링' 순으로 통해 이루어진다.

객체 모델링 (Object Modeling)	정보 모델링이라고도 하며, 시스템에서 요구되는 객체를 찾아내어 속성과 연산 식별 및 객체들 간의 관계를 규정하여 객체 다이어그램으로 표시하는 것이다.
동적 모델링 (Dynamic Modeling)	상태 다이어그램(상태)을 이용하여 시간의 흐름에 따른 객체들 간의 제어흐름, 상호작용, 동작 순서 등의 동적인 행위를 표현하는 모델링이다.
기능 모델링 (Functional Modeling)	자료 흐름도(DFD)를 이용하여 다수의 프로세스들 간의 자료 흐름을 중심으로 처리 과정을 표현한 모델링이다.

객체 다이어그램 : 소프트웨어를 구성하는 객체와 객체 간의 관계를 표현하는 그래픽 표기법
상태 다이어그램 : 객체의 상태가 시간에 따라 어떻게 변하는지를 표현하는 그래픽 표기법

1. 애플리케이션 설계-SEC_04(객체지향 분석 및 설계)

4) 객체지향 설계 원칙

; 객체지향 설계 원칙은 시스템 변경이나 확장에 유연한 시스템을 설계하기 위해 지켜야 할 다섯 가지 원칙으로, 다섯 가지 원칙의 앞 글자를 따 SOLID 원칙이라고도 불린다.

단일 책임 원칙(SRP, Single Responsibility Principle)	•객체는 단 하나의 책임만 가져야 한다는 원칙이다. •응집도는 높고, 결합도는 낮게 설계하는 것을 의미한다.
개방폐쇄 원칙(OCP, Open-Closed Principle)	•기존의 코드를 변경하지 않고 기능을 추가할 수 있도록 설계해야 한다는 원칙이다. •공통 인터페이스를 하나의 인터페이스로 묶어 캡슐화하는 방법이 대표적이다.
리스코프 치환 원칙(LSP, Liskov Substitution Principle)	•자식 클래스는 최소한 자신의 부모 클래스에서 가능한 행위는 수행할 수 있어야 한다는 설계 원칙이다. •자식 클래스는 부모 클래스의 책임을 무시하거나 재정의하지 않고 확장만 수행하도록 해야 한다.
인터페이스 분리 원칙(ISP, Interface Segregation Principle)	•자신이 사용하지 않는 인터페이스와 의존 관계를 맺거나 영향을 받지 않아야 한다는 원칙이다. •단일 책임 원칙이 객체가 갖는 하나의 책임이라면, 인터페이스 분리 원칙은 인터페이스가 갖는 하나의 책임이다.
의존 역전 원칙(DIP, Dependency Inversion Principle)	•각 객체들 간의 의존 관계가 성립될 때, 추상성이 낮은 클래스보다 추상성이 높은 클래스와 의존 관계를 맺어야 한다는 원칙이다. •일반적으로 인터페이스를 활용하면 이 원칙은 준수된다.

1. 애플리케이션 설계-SEC_04(객체지향 분석 및 설계) 기출 문제

기출 문제(객체지향 분석 및 설계)

1. 객체지향 분석 방법론 중 Coad-Yourdon 방법에 해당하는 것은?

① E-R 다이어그램을 사용하여 객체의 행위를 데이터 모델링 하는데 초점을 둔 방법이다.

② 객체, 동적, 기능 모델로 나누어 수행하는 방법이다.

③ 미시적 개발 프로세스와 거시적 개발 프로세스를 모두 사용하는 방법이다.

④ Use Case를 강조하여 사용하는 방법이다.

설명

Coad-Yourdon 방법 : E-R 다이어그램을 사용하여 객체의 행위를 데이터 모델링 하는데 초점을 둔 방법이다. 객체 식별, 구조 식별, 주제 정의, 속성과 인스턴스 연결 정의, 연산과 메시지 연결 정의 등의 과정으로 구성하는 기법

럼바우 방법 : 객체 모델, 동적 모델, 기능 모델로 나누어 수행하는 방법이다.

Booch(부치) 방법 : 미시적 개발 프로세스와 거시적 개발 프로세스를 모두 사용하는 방법이다.

3. 럼바우(Rumbaugh)의 객체지향분석 절차를 가장 바르게 나열한 것은?

① 객체 모형 → 동적 모형 → 기능 모형

② 객체 모형 → 기능 모형 → 동적 모형

③ 기능 모형 → 동적 모형 → 객체 모형

④ 기능 모형 → 객체 모형 → 동적 모형

설명 : 럼바우 분석 기법에서는 객체 모형 → 동적 모형 → 기능 모형 순으로 분석한다.

4. 다음 내용이 설명하는 객체지향 설계 원칙은?

- 클라이언트는 자신이 사용하지 않는 메소드와 의존관계를 맺으면 안 된다.
- 클라이언트가 사용하지 않는 인터페이스 때문에 영향을 받아서는 안 된다.

① 인터페이스분리 원칙 ② 단일 책임 원칙

③ 개방 폐쇄의 원칙 ④ 리스코프 교체의 원칙

설명

인터페이스 분리 원칙 : 자신이 사용하지 않는 인터페이스와 의존 관계를 맺거나 영향을 받지 않아야 한다는 원칙이다.

1. 애플리케이션 설계-SEC_04(객체지향 분석 및 설계) 기출 문제

기출 문제(객체지향 분석 및 설계)

5. 클래스 설계 원칙에 대한 바른 설명은?

① 단일 책임 원칙 : 하나의 클래스만 변경 가능해야 한다.

② 개방, 폐쇄의 원칙 : 클래스는 확장에 대해 열려 있어야 하며 변경에 대해 닫혀 있어야 한다.

③ 리스코프 교체의 원칙 : 여러 개의 책임을 가진 클래스는 하나의 책임을 가진 클래스로 대체 되어야 한다.

④ 의존관계 역전의 원칙 : 클라이언트는 자신이 사용하는 메소드와 의존관계를 갖지 않도록 해야 한다.

6. 소프트웨어를 개발하기 위한 비즈니스(업무)를 객체와 속성, 클래스와 멤버, 전체와 부분 등으로 나누어서 분석해 내는 기법은?

① 객체지향 분석 ② 구조적 분석

③ 기능적 분석 ④ 실시간 분석

설명 : 객체 지향 분석(OOA)은 사용자의 요구사항을 분석하여 요구된 문제와 관련된 모든 클래스, 이와 연관된 속성과 연산, 그들 간의 관계 등을 정의하여 모델링 하는 작업

7. 럼바우(Rumbaugh)의 객체지향 분석 기법 중 자료 흐름도(DFD)를 주로 이용하는 것은?

① 기능 모델링 ② 동적 모델링

③ 객체 모델링 ④ 정적 모델링

설명

객체 모델링 : 정보 모델링이라고도 하며, 시스템에서 요구되는 객체를 찾아내어 속성과 연산 식별 및 객체들 간의 관계를 규정하는 것

동적 모델링 : 상태 다이어그램을 이용하여, 시간의 흐름에 따른 객체들 간의 제어 흐름, 상호작용, 동작 순서 등의 동적인 행위를 표현하는 모델링이다.

기능 모델링 : 자료 흐름도(DFD)를 이용하여 다수의 프로세스들 간의 자료 흐름을 중심으로 처리과정을 표현한 모델링이다.

8. 럼바우(Rumbaugh) 분석 기법에서 정보 모델링이라고도 하며, 시스템에서 요구되는 객체를 찾아내어 속성과 연산 식별 및 객체들 간의 관계를 규정하여 다이어그램을 표시하는 모델링은?

① Object(객체) ② Dynamic(동적인)

③ Function(함수, 기능, 연산) ④ Static(정적인)

1. 애플리케이션 설계-SEC_05(모듈)

1) 모듈의 개요

; 모듈은 모듈화를 통해 분리된 시스템의 각 기능들로, 서브루틴, 서브시스템, 소프트웨어 내의 프로그램, 작업 단위 등과 같은 의미로 사용된다.

- 모듈은 단독으로 컴파일이 가능하며, 재사용 할 수 있다.
- 모듈의 기능적 독립성은 소프트웨어를 구성하는 각 모듈의 기능이 서로 독립됨을 의미하는 것으로, 모듈이 하나의 기능만을 수행하고 다른 모듈과의 과도한 상호작용을 배제함으로써 이루어진다.
- 독립성이 높은 모듈일수록 모듈을 수정하더라도 다른 모듈들에게는 거의 영향을 미치지 않으며, 오류가 발생해도 쉽게 발견하고 해결할 수 있다.
- 모듈의 독립성은 결합도(Coupling)와 응집도(Cohesion)에 의해 측정되며, 독립성을 높이려면 모듈의 결합도는 약하게, 응집도는 강하게, 모듈의 크기는 작게 만들어야 한다.

모듈화(Modularity) : 모듈화는 소프트웨어의 성능을 향상시키거나 시스템의 수정 및 재사용 유지 관리 등이 용이하도록 시스템의 기능들을 모듈 단위로 분해하는 것을 의미한다.
•루틴(Routine) : 기능을 가진 명령들의 모임
•메인 루틴(Main Routine) : 프로그램 실행의 큰 줄기가 되는 것
•서브루틴(Subroutine) : 메인 루틴에 의해 필요할 때 마다 호출되는 루틴
서브시스템(Subsystem) : 서브시스템은 시스템을 구성하는 요소의 하나로, '단위시스템'이라고도 불리며, 서브시스템 자체로도 하나의 시스템에 필요한 요소들을 갖추고 있다. 예를 들어 메인 시스템이 '통합 경영정보 시스템' 이라면 여기에 속하는 서브시스템으로 '영업 관리 시스템', '생산관리 시스템', '인사관리 시스템' 등이 있을 수 있다.

1. 애플리케이션 설계-SEC_05(모듈)

2) 결합도(Coupling)

; 결합도는 모듈 간에 상호 의존하는 정도 또는 두 모듈 사이의 연관 관계를 의미한다.

- 다양한 결합으로 모듈을 구성할 수 있으나 결합도가 약할수록 품질이 높고, 강할수록 품질이 낮다.
- 결합도가 강하면 시스템 구현 및 유지보수 작업이 어렵다.
- 결합도의 종류에는 자료 결합도, 스탬프 결합도, 제어 결합도, 외부 결합도, 공통 결합도, 내용 결합도가 있으며 결합도의 정도는 다음과 같다.

자료 결합도	스탬프 결합도	제어 결합도	외부 결합도	공통 결합도	내용 결합도

결합도 약함 ⟷ 결합도 강함

1. 애플리케이션 설계-SEC_05(모듈)

2) 결합도(Coupling)

자료 결합도 (Data Coupling)	•모듈 간의 인터페이스가 자료 요소로만 구성될 때의 결합도이다. •어떤 모듈이 다른 모듈을 호출하면서 매개 변수나 인수로 데이터를 넘겨주고 호출 받은 모듈은 받은 데이터에 대한 처리 결과를 다시 돌려주는 방식이다. •모듈 간의 내용을 전혀 알 필요가 없는 상태로서 한 모듈의 내용을 변경하더라도 다른 모듈에는 전혀 영향을 미치지 않는 가장 바람직한 결합도이다.
스탬프(검인) 결합도 (Stamp Coupling)	•모듈 간의 인터페이스로 배열이나 레코드 등의 자료 구조가 전달될 때의 결합도이다. •두 모듈이 동일한 자료 구조를 조회하는 경우의 결합도이며, 자료 구조의 어떠한 변화, 즉 포맷이나 구조의 변화는 그것을 조회하는 모든 모듈 및 변화되는 필드를 실제로 조회하지 않는 모듈에까지도 영향을 미치게 된다.
제어 결합도 (Control Coupling)	•어떤 모듈이 다른 모듈 내부의 논리적인 흐름을 제어하기 위해 제어 신호를 이용하여 통신하거나 제어 요소(Function Code, Switch, Tag, Flag)를 전달하는 결합도이다. •한 모듈이 다른 모듈의 상세한 처리 절차를 알고 있어 이를 통제하는 경우나 처리 기능이 두 모듈에 분리되어 설계된 경우에 발생한다. •하위모듈에서 상위 모듈로 제어 신호가 이동하여 하위 모듈이 상위 모듈에게 처리 명령을 내리는 권리 전도현상이 발생하게 된다.
외부 결합도 (External Coupling)	•어떤 모듈에서 선언한 데이터(변수)를 외부의 다른 모듈에서 참조할 때의 결합도이다. •참조되는 데이터의 범위를 각 모듈에서 제한할 수 있다.
공통(공유) 결합도 (Common Coupling)	•공유되는 공통 데이터 영역을 여러 모듈이 사용할 때의 결합도이다. •공통 데이터 영역의 내용을 조금만 변경하더라도 이를 사용하는 모든 모듈에 영향을 미치므로 모듈의 독립성을 약하게 만든다.
내용 결합도 (Content Coupling)	•한 모듈이 다른 모듈의 내부 기능 및 그 내부 자료를 직접 참조하거나 수정할 때의 결합도이다. •한 모듈에서 다른 모듈의 내부로 제어가 이동하는 경우에도 내용 결합도에 해당된다.

1. 애플리케이션 설계-SEC_05(모듈)

3) 응집도(Cohesion)

; 응집도는 정보 은닉 개념을 확장한 것으로, 명령어나 호출문 등 모듈의 내부 요소들의 서로 관련되어 있는 정도, 즉 모듈이 독립적인 기능으로 정의되어 있는 정도를 의미한다.

- 다양한 기준으로 모듈을 구성할 수 있으나 응집도가 강할수록 품질이 높고, 약할수록 품질이 낮다.
- 응집도의 종류에는 기능적 응집도, 순차적 응집도, 교환(통신)적 응집도, 절차적 응집도, 시간적 응집도, 논리적 응집도, 우연적 응집도가 있으며 응집도의 정도는 다음과 같다.

기능적 응집도	순차적 응집도	교환적 응집도	절차적 응집도	시간적 응집도	논리적 응집도	우연적 응집도

응집도 강함 ⟷ 응집도 약함

1. 애플리케이션 설계-SEC_05(모듈)

3) 응집도(Cohesion)

기능적 응집도 (Functional Cohesion)	모듈 내부의 모든 기능 요소들이 단일 문제와 연관되어 수행될 경우의 응집도
순차적 응집도 (Sequential Cohesion)	모듈 내 하나의 활동으로부터 나온 출력 데이터를 그 다음 활동의 입력 데이터로 사용할 경우의 응집도
교환(통신)응집도 (Communication Cohesion)	동일한 입력과 출력을 사용하여 서로 다른 기능을 수행하는 구성 요소들이 모였을 경우의 응집도
절차적 응집도 (Procedural Cohesion)	모듈이 다수의 관련 기능을 가질 때 모듈 안의 구성 요소들이 그 기능을 순차적으로 수행할 경우의 응집도
시간적 응집도 (Temporal Cohesion)	특정 시간에 처리되는 몇 개의 기능을 모아 하나의 모듈로 작성할 경우의 응집도
논리적 응집도 (Logical Cohesion)	유사한 성격을 갖거나 특정 형태로 분류되는 처리 요소들로 하나의 모듈이 형성되는 경우의 응집도
우연적 응집도 (Coincidental Cohesion)	모듈 내부의 각 구성 요소들이 서로 관련 없는 요소로만 구성된 경우의 응집도

1. 애플리케이션 설계-SEC_05(모듈)

4) 팬인(Fan-In) / 팬아웃(Fan-Out)

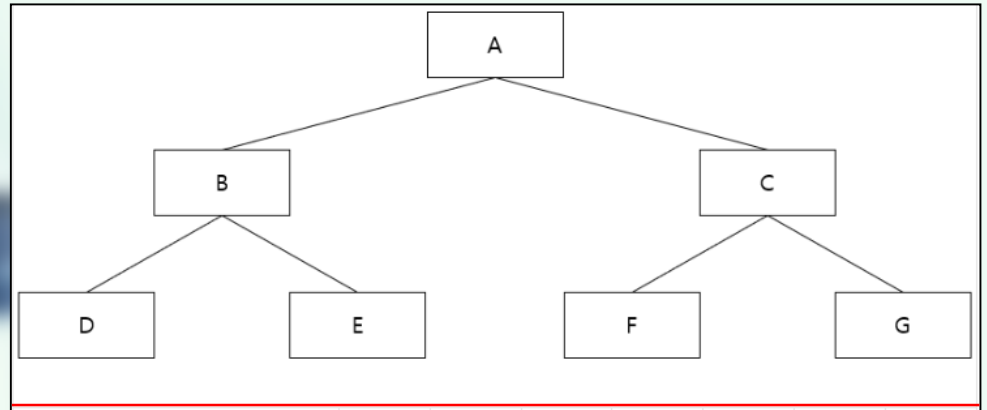
- 팬인은 어떤 모듈을 제어(호출)하는 모듈의 수를 나타낸다.
- 팬아웃은 어떤 모듈에 의해 제어(호출)되는 모듈의 수를 나타낸다.
- 팬인과 팬아웃을 분석하여 시스템의 복잡도를 알 수 있다.
- 팬인이 높다는 것은 재사용 측면에서 설계가 잘 되어 있다고 볼 수 있으나, 단일 장애점이 발생할 수 있으므로 중점적인 관리 및 테스트가 필요하다.
- 팬아웃이 높은 경우 불필요하게 다른 모듈을 호출하고 있는지 검토하고, 단순화 시킬 수 있는지 여부에 대한 검토가 필요하다.
- 시스템의 복잡도를 최적화하려면 팬인은 높게, 팬아웃은 낮게 설계해야 한다.

팬인, 팬아웃은 단순하게 생각하면 이해가 쉽다.
모듈에 들어오면(In) 팬인, 모듈에서 나가면(Out) 팬아웃이다.
단일 장애점(SPOF, Single Point Of Failure) : 단일 장애점은 시스템의 구성 요소 중 동작하지 않으면 전체 시스템이 중단되어 버리는 요소를 의미하며, 단일 실패점 이라고도 한다.

1. 애플리케이션 설계-SEC_05(모듈)

4) 팬인(Fan-In) / 팬아웃(Fan-Out)

; 다음의 시스템 구조도에서 각 모듈의 팬인(Fan-In)과 팬아웃(Fan-Out)을 구하시오.



	A	B	C	D	E	F	G
팬인	0	1	1	1	1	1	1
팬아웃	2	2	2	0	0	0	0

1. 애플리케이션 설계-SEC_05(모듈)

5) N-S 차트(Nassi-Schneiderman Chart)

; N-S 차트는 논리의 기술에 중점을 둔 도형을 이용한 표현 방법으로 박스 다이어그램, Chapin Chart라고도 한다.

- 연속 선택 및 다중 선택, 반복 등의 제어 논리 구조를 표현한다.
- GOTO나 화살표를 사용하지 않는다.
- 조건이 복합되어 있는 곳의 처리를 시각적으로 명확히 식별하는 데 적합하다.
- 선택과 반복 구조를 시각적으로 표현한다.
- 이해하기 쉽고, 코드 변환이 용이하다.
- 읽기는 쉽지만 작성하기가 어려우며, 임의로 제어를 전이하는 것이 불가능하다.
- 총체적인 구조 표현과 인터페이스를 나타내기가 어렵다.
- 단일 입구와 단일 출구로 표현한다.

1. 애플리케이션 설계-SEC_05(모듈) 기출 문제

기출 문제(모듈)

1. 결합도(Coupling)에 대한 설명으로 틀린 것은?

① 데이터 결합도(Data Coupling)는 두 모듈이 매개변수로 자료를 전달할 때, 자료 구조 형태로 전달되어 이용될 때 데이터가 결합되어 있다고 한다.

② 내용 결합도(Content Coupling)는 하나의 모듈이 직접적으로 다른 모듈의 내용을 참조할 때 두 모듈은 내용적으로 결합되어 있다고 한다.

③ 공통 결합도(Common Coupling)는 두 모듈이 동일한 전역 데이터를 접근한다면 공통 결합 되어 있다고 한다.

④ 결합도(Coupling)는 두 모듈 간의 상호작용, 또는 의존도 정도를 나타내는 것이다.

설명

데이터(자료) 결합도 : 모듈 간에 인터페이스가 자료 요소로만 구성될 때의 결합도를 의미한다. 어떠한 모듈이 다른 모듈을 호출하면서 매개변수나 인수로 데이터를 넘겨주고 호출 받은 모듈은 받은 데이터에 대한 처리 결과를 다시 돌려주는 방식이다. 모듈 간의 내용을 전혀 알 필요가 없는

3. 어떤 모듈이 다른 모듈의 내부 논리 조직을 제어하기 위한 목적으로 제어 신호를 이용하여 통신하는 경우이며, 하위 모듈에서 상위 모듈로 제어 신호가 이동하여 상위 모듈에게 처리 명령을 부여하는 권리 전도 현상이 발생하게 되는 결합도는?

① Data Coupling ② Stamp Coupling

③ Control Coupling ④ Common Coupling

설명 : 제어 결합도(Control Coupling)는 어떤 모듈이 다른 모듈의 내부 논리 조직을 제어하기 위한 목적으로 제어 신호(Function Code, Switch, Tag, Flag)를 이용하여 통신하는 경우이며, 하위 모듈에서 상위 모듈로 제어 신호가 이동하여 상위 모듈에게 처리 명령을 부여하는 권리 전도 현상이 발생하게 되는 결합도이다.

4. 소프트웨어 개발에서 모듈(Module)이 되기 위한 주요 특징에 해당하지 않는 것은?

① 다른 것들과 구별될 수 있는 독립적인 기능을 가진 단위(Unit)이다.

② 독립적인 컴파일이 가능하다.

③ 유일한 이름을 가져야 한다.

④ 다른 모듈에서의 접근이 불가능 해야 한다.

설명 : 모듈들은 상호 작용을 통해서 더 큰 시스템을 구성해야 하므로

1. 애플리케이션 설계-SEC_05(모듈) 기출 문제

기출 문제(모듈)

5. 응집도의 종류 중 서로 간에 어떠한 의미 있는 연관관계 도 지니지 않은 기능 요소로 구성되는 경우이며, 서로 다른 상위 모듈에 의해 호출되어 처리상의 연관성이 없는 서로 다른 기능을 수행하는 경우의 응집도는?

① Functional Cohesion ② Sequential Cohesion

③ Logical Cohesion ④ Coincidental Cohesion

설명 : 우연적 응집도는 모듈 내부의 각 구성요소들이 서로 관련 없는 요소로만 구성된 경우의 응집도이다.

6. 모듈화(Modularity)와 관련한 설명으로 틀린 것은?

① 시스템을 모듈로 분할하면 각각의 모듈을 별개로 만들고 수정할 수 있기 때문에 좋은 구조가 된다.

② 응집도는 모듈과 모듈 사이의 상호의존 또는 연관 정도를 의미한다.

③ 모듈 간의 결합도가 약해야 독립적인 모듈이 될 수 있다.

④ 모듈 내 구성 요소들 간의 응집도가 강해야 좋은 모듈 설계이다.

설명

7. 응집도가 가장 낮은 것은?

① 기능적 응집도② 시간적 응집도

③ 절차적 응집도 ④ 우연적 응집도

설명

응집도의 강한 순서

기능적 응집도->순차적 응집도->교환(통신) 응집도->절차적 응집도->시간적 응집도->논리적 응집도->우연적 응집도 순이다.

8. N-S(Nassi-Schneiderman) Chart에 대한 설명으로 거리가 먼 것은?

① 논리의 기술에 중점을 둔 도형식 표현 방법이다.

② 연속, 선택 및 다중 선택, 반복 등의 제어 논리 구조로 표현한다.

③ 주로 화살표를 사용하여 논리적인 제어 구조로 흐름을 표현한다.

④ 조건이 복합되어 있는 곳의 처리를 시각적으로 명확히 식별하는데 적합하다.

설명

N-S차트는 논리의 기술에 중점을 둔 도형식 표현 방법이다.

GOTO나 화살표를 사용하지 않는다.

읽기는 쉽지만 작성하기가 어려우며, 임의로 제어를 전이하는 것이

1. 애플리케이션 설계-SEC_05(모듈) 기출 문제

기출 문제(모듈)

9. 결합도가 낮은 것부터 높은 순으로 옳게 나열한 것은?

(ㄱ) 내용 결합도 (ㄴ) 자료 결합도

(ㄷ) 공통 결합도 (ㄹ) 스탬프 결합도

(ㅁ) 외부 결합도 (ㅂ) 제어 결합도

① (ㄱ) → (ㄴ) → (ㄹ) → (ㅂ) → (ㅁ) → (ㄷ)

② (ㄴ) → (ㄹ) → (ㅁ) → (ㅂ) → (ㄷ) → (ㄱ)

③ (ㄴ) → (ㄹ) → (ㅂ) → (ㅁ) → (ㄷ) → (ㄱ)

④ (ㄱ) → (ㄴ) → (ㄹ) → (ㅁ) → (ㅂ) → (ㄷ)

설명

자료 결합도->스탬프(검인) 결합도->제어 결합도->
외부 결합도->공통(공유) 결합도->내용 결합도 순으로
결합도가 강해진다.

10. 다음 중 가장 강한 응집도(Cohesion)는?

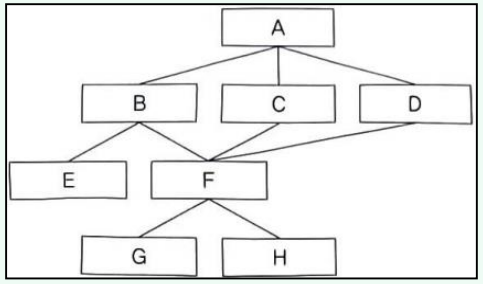
① Sequential Cohesion ② Procedural Cohesion

③ Logical Cohesion ④ Coincidental Cohesion

설명

응집도의 강한 순서

11. 다음은 어떤 프로그램 구조를 나타낸다. 모듈 F에서의 fan-in, fan-out의 수는 얼마인가?



① fan-in: 2, fan-out: 3

② fan-in: 3, fan-out: 2

③ fan-in: 1, fan-out: 2

④ fan-in: 2, fan-out: 1

설명

팬인(fan-in) : 모듈에 들어오는 것

팬아웃(fan-out) : 모듈에서 나가는 것

12. 프로그램 설계도의 하나인 NS-Chart에 대한 설명으로 가장 거리가 먼 것은?

① 논리의 기술에 중점을 두고 도형을 이용한 표현 방법이다.

② 이해하기 쉽고 코드 변환이 용이하다.

③ 화살표나 GOTO를 사용하여 이해하기 쉽다.

1. 애플리케이션 설계-SEC_06(공통 모듈)

1) 공통 모듈의 개요

; 공통 모듈은 여러 프로그램에서 공통적으로 사용할 수 있는 모듈을 의미한다.

- 자주 사용되는 계산식이나 매번 필요한 사용자 인증과 같은 기능들이 공통 모듈로 구성될 수 있다.
- 모듈의 재 사용성 확보와 중복 개발 회피를 위해 설계 과정에서 공통 부분을 식별하고 명세를 작성할 필요가 있다.
- 공통 모듈을 구현할 때는 다른 개발자들이 해당 기능을 명확히 이해할 수 있도록 다음의 명세 기법을 준수해야 한다.

정확성(Correctness)	시스템 구현 시 해당 기능이 필요하다는 것을 알 수 있도록 정확히 작성한다.
명확성(Clarity)	해당 기능을 이해할 때 중의적으로 해석되지 않도록 명확하게 작성한다.
완전성(Completeness)	시스템 구현을 위해 필요한 모든 것을 기술한다.
일관성(Consistency)	공통 기능들 간 상호 충돌이 발생하지 않도록 작성한다.
추적성(Traceability)	기능에 대한 요구사항의 출처 관련 시스템 등의 관계를 파악할 수 있도록 작성한다.

1. 애플리케이션 설계-SEC_06(공통 모듈)

2) 재사용(Reuse)

; 재사용은 비용과 개발 시간을 절약하기 위해 이미 개발된 기능들을 파악하고 재구성하여 새로운 시스템 또는 기능 개발에 사용하기 적합하도록 최적화 시키는 작업이다.

- 재사용을 위해서는 누구나 이해할 수 있고 사용이 가능하도록 사용법을 공개해야 한다.
- 재사용되는 대상은 외부 모듈과의 결합도는 낮고, 응집도는 높아야 한다.
- 재사용 규모에 따른 분류

함수와 객체	클래스나 메소드 단위의 소스 코드를 재사용한다.
컴포넌트	•독립적인 업무 또는 기능을 수행하는 실행 코드 기반으로 작성된 모듈이다. •컴포넌트 자체에 대한 수정 없이 인터페이스를 통해 통신하는 방식으로 재사용 한다.
애플리케이션	공통된 기능들을 제공하는 애플리케이션을 공유하는 방식으로 재사용한다.

함수 = 메소드 : 객체의 데이터를 처리하는 알고리즘
객체 : 데이터와 함수를 캡슐화한 소프트웨어 모듈
클래스: 객체를 정의하는 틀(설계도)
애플리케이션 : 어떠한 목적을 갖고 개발된 소프트웨어

1. 애플리케이션 설계-SEC_06(공통 모듈)

3) 효과적인 모듈 설계 방안

- 결합도는 줄이고 응집도는 높여서 모듈의 독립성과 재 사용성을 높인다.
- 모듈의 제어 영역 안에서 그 모듈의 영향 영역을 유지시킨다.
- 복잡도와 중복성을 줄이고 일관성을 유지시킨다.
- 모듈의 기능은 예측이 가능해야 하며 지나치게 제한적이어서는 안 된다.
- 유지보수가 용이해야 한다.
- 모듈 크기는 시스템의 전반적인 기능과 구조를 이해하기 쉬운 크기로 분해한다.
- 하나의 입구와 하나의 출구를 갖도록 해야 한다.
- 인덱스 번호나 기능 코드들이 전반적인 처리 논리 구조에 예기치 못한 영향을 끼치지 않도록 모듈 인터페이스를 설계해야 한다.
- 효과적인 제어를 위해 모듈 간의 계층적 관계를 정의하는 자료가 제시되어야 한다.

결합도 : 모듈 간에 상호 의존하는 정도 또는 두 모듈 사이의 연관 관계
응집도 : 모듈의 내부 요소들의 서로 관련되어 있는 정도
모듈의 제어 영역 : 프로그램의 계층 구조 내에서 어떤 특정 모듈이 제어하는 하위 모듈
모듈의 영향 영역 : 특정 모듈이 다른 모듈들에게 미치게 되는 영향의 범위

1. 애플리케이션 설계-SEC_06(공통 모듈)기출 및 출제 예상 문제

기출 문제 및 출제 예상문제(공통 모듈)

1. 공통 모듈에 대한 명세 기법 중 해당 기능에 대해 일관 되게 이해되고 한 가지로 해석될 수 있도록 작성하는 원칙은?

① 상호 작용성 ② 명확성

③ 독립성 ④ 내용성

설명

1.정확성 : 시스템 구현 시 해당 기능이 필요하다는 것을 알 수 있도록 정확히 작성한다.

2.명확성 : 해당 기능을 이해할 때 중의적으로 해석되지 않도록 명확하게 작성한다.

3.완전성 : 시스템 구현을 위해 필요한 모든 것을 기술한다.

4.일관성 : 공통 기능들 간 상호 충돌이 발생하지 않도록 한다.

5.추적성 : 기능에 대한 요구사항의 출처 관련 시스템 등의 관계를 파악할 수 있도록 작성한다.

2. 공통모듈의 재사용 범위에 따른 분류가 아닌 것은?

① 컴포넌트 재사용 ② 더미코드 재사용

③ 함수와 객체 재사용 ④ 애플리케이션 재사용

3. 명백한 역할을 가지고 독립적으로 존재할 수 있는 시스템의 부분으로 넓은 의미에서는 재사용 되는 모든 단위라고 볼 수 있으며, 인터페이스를 통해서만 접근할 수 있는 것은?

① Model ② Sheet

③ Component ④ Cell

4. 바람직한 소프트웨어 설계 지침이 아닌 것은?

① 적당한 모듈의 크기를 유지한다.

② 모듈 간의 접속 관계를 분석하여 복잡도와 중복을 줄인다.

③ 모듈 간의 결합도는 강할수록 바람직하다.

④ 모듈 간의 효과적인 제어를 위해 설계에서 계층적 자료 조직이 제시되어야 한다.

설명

결합도는 줄이고 응집도는 높여서 모듈의 독립성과 재사용성을 높인다.

복잡도와 중복성을 줄이고 일관성을 유지시킨다.

모듈의 기능은 예측이 가능해야 하며 지나치게 제한적이어서는 안된다.

모듈 크기는 시스템의 전반적인 기능과 구조를 이해하기 쉬운 크기로 분해한다.

1. 애플리케이션 설계-SEC_06(공통 모듈)기출 및 출제 예상 문제

기출 문제 및 출제 예상문제(공통 모듈)

5. 소프트웨어의 일부분을 다른 시스템에서 사용할 수 있는 정도를 의미하는 것은?

① 신뢰성(Reliability)

② 유지보수성(Maintainability)

③ 가시성(Visibility)

④ 재사용성(Reusability)

설명 : 재사용성은 비용과 개발 시간을 절약하기 위해 이미 개발된 기능들을 파악하고 재구성하여 새로운 시스템 또는 기능 개발에 사용하기 적합하도록 최적화 시키는 작업이다.

6. 좋은 소프트웨어 설계를 위한 소프트웨어의 모듈간의 결합도(Coupling)와 모듈 내 요소 간 응집도(Cohesion)에 대한 설명으로 옳은 것은?

① 응집도는 낮게 결합도는 높게 설계한다.

② 응집도는 높게 결합도는 낮게 설계한다.

③ 양쪽 모두 낮게 설계한다.

④ 양쪽 모두 높게 설계한다.

7. 효과적인 모듈 설계를 위한 유의사항으로 거리가 먼 것은?

① 모듈간의 결합도를 약하게 하면 모듈 독립성이 향상된다.

② 복잡도와 중복성을 줄이고 일관성을 유지시킨다.

③ 모듈의 기능은 예측이 가능해야 하며 지나치게 제한적이어야 한다.

④ 유지보수가 용이해야 한다.

설명

모듈의 기능은 예측이 가능해야 하며 지나치게 제한적이지 않아야 한다.

8. 소프트웨어 재사용에 대한 내용 중 옳지 않은 것은?

① 비용을 절감하고 개발 시간을 단축하여 생산성이 증가한다.

② 재사용되는 대상은 외부 모듈과의 결합도가 낮아야 한다

③ 규모에 따라 변수, 함수, 객체, 컴포넌트 단위로 재사용 된다.

④ 재사용을 위해서는 사용법이 공개되어야 한다.

설명

재사용 규모에 따른 분류 : 함수와 객체, 컴포넌트, 애플리케이션으로 재사용 된다.

1. 애플리케이션 설계-SEC_06(공통 모듈)기출 및 출제 예상 문제

기출 문제 및 출제 예상 문제(공통 모듈)

9. 공통 모듈을 설계할 때 공통 부분을 명세하기 위한 기법 에 해당하지 않는 것은?

① 독립성　　② 명확성

③ 일관성　　④ 정확성

설명

공통 부분 명세 기법 : 정확성, 명확성, 완전성, 일관성, 추적성이 있다.

1. 애플리케이션 설계-SEC_07(코드)

1) 코드(Code)의 개요

; 코드는 컴퓨터를 이용하여 자료를 처리하는 과정에서 분류, 조합 및 집계를 용이하게 하고, 특정 자료의 추출을 쉽게 하기 위해서 사용하는 기호이다.

- 코드는 정보를 신속, 정확, 명료하게 전달할 수 있게 한다.
- 코드는 일정한 규칙에 따라 작성되며, 정보 처리의 효율과 처리된 정보의 가치에 많은 영향을 미친다.
- 일반적인 코드의 예로 주민등록번호, 학번, 전화번호 등이 있다.
- 코드의 주요 기능에는 식별 기능, 분류 기능, 배열 기능, 표준화 기능, 간소화 기능이 있다.

식별 기능	데이터 간의 성격에 따라 구분이 가능하다.
분류 기능	특정 기준이나 동일한 유형에 해당하는 데이터를 그룹화 할 수 있다.
배열 기능	의미를 부여하여 나열할 수 있다.
표준화 기능	다양한 데이터를 기준에 맞추어 표현할 수 있다.
간소화 기능	복잡한 데이터를 간소화할 수 있다.

1. 애플리케이션 설계-SEC_07(코드)

2) 코드의 종류

; 코드의 종류에는 다음과 같은 것들이 있다.

순차 코드 (Sequence Code)	자료의 발생 순서, 크기 순서 등 일정 기준에 따라서 최초의 자료부터 차례로 일련번호를 부여하는 방법으로 순서 코드 또는 일련번호 코드라고도 한다. 예) 1, 2, 3, 4...
블록 코드 (Block Code)	코드화 대상 항목 중에서 공통성이 있는 것끼리 블록으로 구분하고, 각 블록 내에서 일련번호를 부여하는 방법으로구분 코드라고도 한다. 예) 1001~1100 : 총무부, 1101~1200: 영업부
10진 코드 (Decimal Code)	코드화 대상 항목을 0~9까지 10진 분할하고, 다시 그 각각에 대하여 10진 분할하는 방법을 필요한 만큼 반복하는 방법으로, 도서 분류식 코드라고도 한다. 예) 1000 : 공학, 1100 : 소프트웨어 공학, 1110 : 소프트웨어 설계
그룹분류 코드 (Group Classification Code)	코드화 대상 항목을 일정 기준에 따라 대분류, 중분류, 소분류 등으로 구분하고, 각 그룹 안에서 일련번호를 부여하는 방법이다. 예) 1-01-001 : 본사-총무부-인사계, 2-01-001 : 지사-총무부-인사계
연상 코드 (Mnemonic Code)	코드화 대상 항목의 명칭이나 약호와 관계 있는 숫자나 문자, 기호를 이용하여 코드를 부여하는 방법이다. 예) TV-40 : 40인치 TV, L-15-220 : 15W 220V 램프
표의 숫자 코드 (Significant Digit Code)	코드화 대상 항목의 성질, 즉 길이, 넓이, 부피, 지름, 높이 등의 물리적 수치를 그대로 코드에 적용시키는 방법으로, 유효 숫자 코드라고도 한다. 예) 120-720-1500 : 두께x폭x길이가 120×720x1500인 강판
합성 코드 (Combined Code)	필요한 기능을 하나의 코드로 수행하기 어려운 경우 2개 이상의 코드를 조합하여 만드는 방법이다. 예) 연상 코드 + 순차 코드 KE-711 : 대한항공 711기, AC-253 : 에어캐나다 253기

약호 : 간단하고 알기 쉽게 만든 부호를 의미한다.

1. 애플리케이션 설계-SEC_07(코드)

3) 코드의 부여 체계

; 코드 부여 체계는 이름만으로 개체의 용도와 적용 범위를 알 수 있도록 코드를 부여하는 방식을 말한다.

- 코드 부여 체계는 각 개체에 유일한 코드를 부여하여 개체들의 식별 및 추출을 용이하게 한다.
- 코드를 부여하기 전에 각 단위 시스템의 고유한 코드와 개체를 나타내는 코드 등이 정의되어야 한다.
- 코드 부여 체계를 담당하는 자는 코드의 자릿수와 구분자, 구조 등을 상세하게 명시해야 한다.

예1) 모듈 식별을 위한 코드 부여 체계

자릿수	구분자를 포함한 11자리
기본 구조	AAA-MOD-000
상세 구조	AAA •영문 및 숫자 3자리 •단위 시스템의 코드 3 자리 • 전체 시스템의 경우 'PJC' 고정 MOD •영문 3자리 •모듈은 MOD, 공통 모듈은 COM을 사용 000 •숫자 3자리 •순차적 일련번호 001~999

예2) 코드 부여 체계에 따른 코드 작성

- PJC-COM-003 : 전체 시스템 단위의 3번째 공통 모듈
- PY3-MOD-010 : PY3이라는 단위 시스템의 10번째 모듈

개체 : 소프트웨어 개발에서 코드를 부여할 대상이 되는 개체에는 모듈, 컴포넌트 인터페이스 등이 있다.

1. 애플리케이션 설계-SEC_07(코드)기출 및 출제 예상 문제

기출 문제 및 출제 예상 문제(코드)

1. 코드의 기본 기능으로 거리가 먼 것은?

① 복잡성 ② 표준화

③ 분류 ④ 식별

설명 : 코드라는 것은 컴퓨터를 이용하여 자료를 처리하는 과정에서 분류, 조합 및 집계를 용이하게 하고, 특정 자료의 추출을 쉽게 하기 위해서 사용하는 기호이다.

코드의 기능으로는 식별 기능, 분류 기능, 배열 기능, 표준화 기능, 간소화 기능이 있다.

2. 코드 설계에서 일정한 일련번호를 부여하는 방식의 코드는?

① 연상 코드 ② 블록 코드

③ 순차 코드 ④ 표의 숫자 코드

설명 : 순차 코드(Sequence Code)는 자료의 발생 순서, 크기 순서 등 일정 기준에 따라서 최초의 자료부터 차례로 일련번호를 부여하는 방법이고 순서 코드나 또는 일련번호 코드라고도 한다.

예)1, 2, 3, 4...

3. 코드화 대상 항목의 중량, 면적, 용량 등의 물리적 수치를 이용하여 만든 코드는?

① 순차 코드 ② 10진 코드

③ 표의 숫자 코드 ④ 블록 코드

설명 : 표의 숫자 코드는 코드화 대상 항목의 성질, 즉 길이, 넓이, 부피, 지름, 높이, 중량, 면적, 용량 등의 물리적 수치를 그대로 코드에 적용시키는 방법이고, 유효 숫자 코드라고 한다.

예) 120-500-1500 : 두께, 폭, 높이가 120*500*1500인 책상

4. 사원 번호의 발급 과정에서 둘 이상의 서로 다른 사람에게 동일한 번호가 부여된 경우에 코드의 어떤 기능을 만족시키지 못한 것인가?

① 표준화 기능 ② 식별 기능

③ 배열 기능 ④ 연상 기능

설명

표준화 기능 : 다양한 데이터를 기준에 맞추어 표현할 수 있다.

식별 기능 : 데이터 간에 성격에 따라 구분이 가능하다.

배열 기능 : 의미를 부여하여 나열할 수 있다.

분류 기능 : 특정 기준이나 동일한 유형에 해당하는 데이터를 그룹화 할 수

1. 애플리케이션 설계-SEC_07(코드)기출 및 출제 예상 문제

기출 문제 및 출제 예상 문제(코드)

5. 회사에서 각 부서의 명칭을 코드화하기 위하여 대분류, 중분류, 소분류 등으로 나누어 나타내고자 한다. 이 때 가장 적합한 코드의 종류는?

① 구분 코드(Block Code)

② 그룹 분류 코드(Group Classification Code)

③ 연상 기호 코드(Mnemonic Code)

④ 순차 코드(Sequence Code)

설명

그룹 분류 코드 : 코드화 대상 항목을 일정 기준에 따라 대분류, 중분류, 소분류 등으로 구분하고, 각 그룹 안에서 일련번호를 부여하는 방법

연상 코드 : 코드화 대상 항목의 명칭이나 약호와 관계 있는 숫자나 문자, 기호를 이용하여 코드를 부여하는 방법

6. 코드화 대상 자료 전체를 계산하여 이를 필요로 하는 분류 단위로 블록을 구분하고, 각 블록 내에서 순서대로 번호를 부여하는 방식으로, 적은 자릿수로 많은 항목 표시 가 가능하고 예비 코드를 사용할 수 있어 추가가 용이하다. 구분

7. 코드 부여 체계에 대한 설명으로 옳지 않은 것은?

① 모듈이나 컴포넌트에 식별할 수 있는 코드를 부여하는 것을 말한다.

② 프로그래머가 모듈을 개발할 때 마다 임의로 코드를 부여한다.

③ 하나 이상의 코드를 조합하여 사용한다.

④ 코드 부여 체계의 담당자는 코드 규칙을 상세히 정의해야 한다.

설명 : 코드 부여 체계는 이름만으로 개체(모듈, 컴포넌트, 인터페이스)의 용도와 적용 범위를 알 수 있도록 코드를 부여하는 방식을 의미한다. 유일한 코드를 부여하여 개체들의 식별 및 추출을 용이하게 한다. 코드 부여 체계 담당자는 코드의 자릿수와 구분자, 구조 등을 상세하게 명시해야 한다.

8. 코드의 주요 기능에서 다양한 데이터를 기준에 맞추어 표현할 수 있는 기능은?

① 식별 기능 ② 분류 기능

③ 표준화 기능 ④ 배열 기능

1. 애플리케이션 설계-SEC_07(코드)기출 및 출제 예상 문제

기출 문제 및 출제 예상 문제(코드)

9. 필요한 기능을 하나의 코드로 수행하기 어려운 경우 2개 이상의 코드를 조합하여 만드는 방법은 무엇인가?

① 구분 코드(Block Code)

② 합성 코드(Combined Code)

③ 연상 기호 코드(Mnemonic Code)

④ 순차 코드(Sequence Code)

10. 소프트웨어 개발에서 코드를 부여할 대상이 되는 개체 가 아닌 것은?

① 모듈 ② 컴포넌트

③ 클래스 ④ 인터페이스

설명 : 코드를 부여할 대상이 되는 개체에는 모듈, 컴포넌트, 인터페이스가 있다. 클래스는 해당하지 않는다.

1. 애플리케이션 설계-SEC_08(디자인 패턴)

1) 디자인 패턴(Design Pattern)의 개요

; 디자인 패턴은 각 모듈의 세분화된 역할이나 모듈들 간의 인터페이스와 같은 코드를 작성하는 수준의 세부적인 구현 방안을 설계할 때 참조할 수 있는 전형적인 해결 방식 또는 예제를 의미한다.

- 디자인 패턴은 문제 및 배경, 실제 적용된 사례, 재사용이 가능한 샘플 코드 등으로 구성되어 있다.
- '바퀴를 다시 발명하지 마라(Don't reinvent the wheel)'라는 말과 같이, 개발 과정 중에 문제가 발생하면 새로 해결책을 구상하는 것보다 문제에 해당하는 디자인 패턴을 참고하여 적용하는 것이 더 효율적이다.
- 디자인 패턴은 한 패턴에 변형을 가하거나 특정 요구사항을 반영하면 유사한 형태의 다른 패턴으로 변화되는 특징이 있다.
- 디자인 패턴은 1995년 GoF(Gang of Four)라고 불리는 에릭 감마(Erich Gamma), 리차드 헬름(Richard Helm), 랄프 존슨(Ralph Johnson), 존 블리시디스(John Vlissides)가 처음으로 구체화 및 체계화 하였다.
- GoF의 디자인 패턴은 수많은 디자인 패턴들 중 가장 일반적인 사례에 적용될 수 있는 패턴들을 분류하여 정리함으로써, 지금까지도 소프트웨어 공학이나 현업에서 가장 많이 사용되는 디자인 패턴이다.
- GoF의 디자인 패턴은 유형에 따라 생성 패턴 5개, 구조 패턴 7개, 행위 패턴 11개 총 23개의 패턴으로 구성된다.

1. 애플리케이션 설계-SEC_08(디자인 패턴)

1) 디자인 패턴(Design Pattern)의 개요

; 아키텍처 패턴과 디자인 패턴은 모두 소프트웨어 설계를 위한 참조 모델이지만 다음과 같은 차이가 있다.

- 아키텍처 패턴은 디자인 패턴보다 상위 수준의 설계에 사용된다.
- 아키텍처 패턴이 전체 시스템의 구조를 설계하기 위한 참조 모델이라면, 디자인 패턴은 서브시스템에 속하는 컴포넌트들과 그 관계를 설계하기 위한 참조 모델이다.
- 몇몇 디자인 패턴은 특정 아키텍처 패턴을 구현하는데 유용하게 사용된다.

2) 디자인 패턴 사용의 장단점

- 범용적인 코딩 스타일로 인해 구조 파악이 용이하다.
- 객체 지향 설계 및 구현의 생산성을 높이는 데 적합하다.
- 검증된 구조의 재사용을 통해 개발 시간과 비용이 절약된다.
- 초기 투자 비용이 부담될 수 있다.하지만 이후에는 검증구조를 재사용함으로써 요구사항 변경에 유연하게 대처할 수 있고, 안정적인 유지보수가 가능하고 개발의 전체적인 측면에서는 비용이 절약된다.
- 개발자 간의 원활한 의사소통이 가능하다.
- 설계 변경 요청에 대한 유연한 대처가 가능하다.
- 객체지향을 기반으로 한 설계와 구현을 다루므로 다른 기반의 애플리케이션 개발에는 적합하지 않다.

1. 애플리케이션 설계-SEC_08(디자인 패턴)

3) 생성 패턴(Creational Pattern)

; 생성 패턴은 객체의 생성과 관련된 패턴으로 총 5개의 패턴이 있다.

● 생성 패턴은 객체의 생성과 참조 과정을 캡슐화하여 객체가 생성되거나 변경되어도 프로그램의 구조에 영향을 크게 받지 않도록 하여 프로그램에 유연성을 더해준다.

추상 팩토리 (Abstract Factory)	•구체적인 클래스에 의존하지 않고, 인터페이스를 통해 서로 연관, 의존하는 객체들의 그룹으로 생성하여 추상적으로 표현한다. •연관된 서브 클래스를 묶어 한 번에 교체하는 것이 가능하다.
빌더 (Builder)	•작게 분리된 인스턴스를 건축 하듯이 조합하여 객체를 생성한다. •객체의 생성 과정과 표현 방법을 분리하고 있어, 동일한 객체 생성에서도 서로 다른 결과를 만들어 낼 수 있다.
팩토리 메소드 (Factory Method)	•객체 생성을 서브 클래스에서 처리하도록 분리하여 캡슐화한 패턴이다. •상위 클래스에서 인터페이스만 정의하고 실제 생성은 서브 클래스가 담당한다. •가상 생성자(Virtual Constructor) 패턴이라고도 한다.
프로토타입 (Prototype)	•원본 객체를 복제하는 방법으로 객체를 생성하는 패턴이다. •일반적인 방법으로 객체를 생성하며, 비용이 큰 경우 주로 이용한다.
싱글톤 (Singleton)	•하나의 객체를 생성하면 생성된 객체를 어디서든 참조할 수 있지만, 여러 프로세스가 동시에 참조할 수는 없다. •클래스 내에서 인스턴스가 하나뿐임을 보장하며, 불필요한 메모리 낭비를 최소화할 수 있다. 예) DB 커넥션

1. 애플리케이션 설계-SEC_08(디자인 패턴)

4) 구조 패턴(Structural Pattern)

; 구조 패턴은 클래스나 객체들을 조합하여 더 큰 구조로 만들 수 있게 해주는 패턴으로 총 7개의 패턴이 있다.

● 구조 패턴은 구조가 복잡한 시스템을 개발하기 쉽게 도와준다.

어댑터 (Adapter)	•호환성이 없는 클래스들의 인터페이스를 타 클래스가 이용할 수 있도록 변환해주는 패턴이다. •기존의 클래스를 이용하고 싶지만 인터페이스가 일치하지 않을 때 이용한다.
브리지 (Bridge)	•구현부에서 추상층을 분리하여, 서로가 독립적으로 확장할 수 있도록 구성한 패턴이다. •기능과 구현을 두 개의 별도 클래스로 구현한다.
컴포지트 (Composite)	•여러 객체를 가진 복합 객체와 단일 객체를 구분 없이 다루고자 할 때 사용하는 패턴이다. •객체들을 트리 구조로 구성하여 디렉터리 안에 디렉터리가 있듯이 복합 객체 안에 복합 객체가 포함되는 구조를 구현할 수 있다.
데코레이터 (Decorator)	•객체 간의 결합을 통해 능동적으로 기능들을 확장할 수 있는 패턴이다. •임의의 객체에 부가적인 기능을 추가하기 위해 다른 객체들을 덧붙이는 방식으로 구현한다.
퍼싸드 (Facade)	•복잡한 서브 클래스들을 피해 더 상위에 인터페이스를 구성함으로써 서브 클래스들의 기능을 간편하게 사용할 수 있도록 하는 패턴이다. •서브 클래스들 사이의 통합 인터페이스를 제공하는 Wrapper 객체가 필요하다.
플라이웨이트 (Flyweight)	•인스턴스가 필요할 때마다 매번 생성하는 것이 아니고 가능한 한 공유해서 사용함으로써 메모리를 절약하는 패턴이다. •다수의 유사 객체를 생성하거나 조작할 때 유용하게 사용할 수 있다.
프록시 (Proxy)	•접근이 어려운 객체와 여기에 연결하려는 객체 사이에서 인터페이스 역할을 수행하는 패턴이다. •네트워크 연결, 메모리의 대용량 객체로의 접근 등에 주로 이용한다.

1. 애플리케이션 설계-SEC_08(디자인 패턴)

5) 행위 패턴(Behavioral Pattern)

; 행위 패턴은 클래스나 객체들이 서로 상호작용 하는 방법이나 책임 분배 방법을 정의하는 패턴으로 총 11개의 패턴이 있다.

● 행위 패턴은 하나의 객체로 수행할 수 없는 작업을 여러 객체로 분배하면서 결합도를 최소화 할 수 있도록 도와준다.

책임 연쇄 (Chain of Responsibility)	•요청을 처리할 수 있는 객체가 둘 이상 존재하여 한 객체가 처리하지 못하면 다음 객체로 넘어가는 형태의 패턴이다. •요청을 처리할 수 있는 각 객체들이 고리(Chain)로 묶여 있어 요청이 해결될 때까지 고리를 따라 책임이 넘어간다.
커맨드 (Command)	•요청을 객체의 형태로 캡슐화하여 재이용하거나 취소할 수 있도록 요청에 필요한 정보를 저장하거나 로그에 남기는 패턴이다. •요청에 사용되는 각종 명령어들을 추상 클래스와 구체 클래스로 분리하여 단순화한다.
인터프리터 (Interpreter)	•언어에 문법 표현을 정의하는 패턴이다. •SQL이나 통신 프로토콜과 같은 것을 개발할 때 사용한다.
반복자 (Iterator)	•자료 구조와 같이 접근이 잦은 객체에 대해 동일한 인터페이스를 사용하도록 하는 패턴이다. •내부표현 방법의 노출 없이 순차적인 접근이 가능하다.
중재자 (Mediator)	•수많은 객체들 간의 복잡한 상호작용(Interface)을 캡슐화하여 객체로 정의하는 패턴이다. •객체 사이의 의존성을 줄여 결합도를 감소시킬 수 있다. •중재자는 객체 간의 통제와 지시의 역할을 수행한다.
메멘토 (Memento)	•특정 시점에서의 객체 내부 상태를 객체화함으로써 이후 요청에 따라 객체를 해당 시점의 상태로 돌릴 수 있는 기능을 제공하는 패턴이다. •Ctrl + Z와 같은 되돌리기 기능을 개발할 때 주로 이용한다.

1. 애플리케이션 설계-SEC_08(디자인 패턴)

5) 행위 패턴(Behavioral Pattern)

옵서버 (Observer)	•한 객체의 상태가 변화하면 객체에 상속되어 있는 다른 객체들에게 변화된 상태를 전달하는 패턴이다. •주로 분산된 시스템 간에 이벤트를 생성, 발행(Publish)하고, 이를 수신(Subscribe)해야 할 때 이용한다.
상태 (State)	•객체의 상태에 따라 동일한 동작을 다르게 처리해야 할 때 사용하는 패턴이다. •객체 상태를 캡슐화하고 이를 참조하는 방식으로 처리한다.
전략 (Strategy)	•동일한 계열의 알고리즘들을 개별적으로 캡슐화하여 상호 교환할 수 있게 정의하는 패턴이다. •클라이언트는 독립적으로 원하는 알고리즘을 선택하여 사용할 수 있으며, 클라이언트에 영향 없이 알고리즘의 변경이 가능하다.
템플릿 메소드 (Template Method)	•상위 클래스에서 골격을 정의하고, 하위 클래스에서 세부 처리를 구체화 하는 구조의 패턴이다. • 유사한 서브 클래스를 묶어 공통된 내용을 상위 클래스에서 정의함으로써 코드의 양을 줄이고 유지보수를 용이하게 해준다.
방문자 (Visitor)	•각 클래스들의 데이터 구조에서 처리 기능을 분리하여 별도의 클래스로 구성하는 패턴이다. •분리된 처리 기능은 각 클래스를 방문(Visit)하여 수행한다.

1. 애플리케이션 설계-SEC_08(디자인 패턴)기출 및 출제 예상 문제

기출 문제 및 출제 예상 문제(디자인 패턴)

1. 소프트웨어 설계에서 자주 발생하는 문제에 대한 일반적이고 반복적인 해결 방법을 무엇이라고 하는가?

① 모듈 분해 ② 디자인 패턴

③ 연관 관계 ④ 클래스 도출

설명 : 디자인 패턴은 각 모듈의 세분화된 역할이나 모듈들 간의 인터페이스와 같은 코드를 작성하는 수준의 세부적인 구현 방안을 설계할 때, 참조할 수 있는 전형적인 해결방식 또는 예제를 의미한다.

2. GoF(Gangs of Four) 디자인 패턴에서 생성(Creational) 패턴에 해당하는 것은?

① 컴포지트(구조 패턴) ② 어댑터(구조 패턴)

③ 추상 팩토리 ④ 옵서버(행위 패턴)

설명 : 생성 패턴은 객체의 생성과 참조 과정을 캡슐화하여 객체가 생성되거나 변경되어도 프로그램의 구조에 영향을 크게 받지 않도록 하여 프로그램에 유연성을 더해준다.
생성 패턴의 종류 : 추상 팩토리, 빌더, 팩토리 메소드, 프로토타입, 싱글톤 패턴이 있다.

3. 디자인 패턴 사용의 장단점에 대한 설명으로 거리가 먼 것은?

① 소프트웨어 구조 파악이 용이하다.

② 객체지향 설계 및 구현의 생산성을 높이는데 적합하다.

③ 재사용을 위한 개발 시간이 단축된다.

④ 절차형 언어와 함께 이용될 때 효율이 극대화된다.

설명 : 디자인 패턴의 단점으로는 초기 투자의 비용이 부담이 될 수 있다. 객체지향을 기반으로 한 설계와 구현을 다루기 때문에 다른 기반의 애플리케이션 개발에는 적합하지 않다.

4. 다음 내용이 설명하는 디자인 패턴은?

◆ 객체를 생성하기 위한 인터페이스를 정의하여 어떤 클래스가 인스턴스화 될 것인지는 서브 클래스가 결정하도록 하는 것

◆ Virtual-Constructor 패턴이라고도 함

① Visitor 패턴(행위 패턴) ② Observer 패턴(행위 패턴)

③ Factory Method 패턴 ④ Bridge 패턴(구조 패턴)

설명 : Factory Method패턴은 객체 생성을 서브 클래스에서 처리하도록 분리하여 캡슐화한 패턴이며, 상위 클래스에서 인터페이스만 정의하고 실제 생성은 서브 클래스가 담당한다. 가상 생성자 패턴이라고도 한다.

1. 애플리케이션 설계-SEC_08(디자인 패턴)기출 및 출제 예상 문제

기출 문제 및 출제 예상 문제(디자인 패턴)

5. GoF(Gang of Four)디자인 패턴을 생성, 구조, 행동 패턴 의 세 그룹으로 분류할 때, 구조 패턴이 아닌 것은?

① Adapter 패턴 (구조 패턴) ② Bridge 패턴(구조 패턴)

③ Builder 패턴(생성 패턴) ④ Proxy 패턴(구조 패턴)

6. 디자인 패턴 중에서 행위적 패턴에 속하지 않는 것은?

① 커맨드(Command) 패턴(행위 패턴)

② 옵서버(Observer) 패턴(행위 패턴)

③ 프로토타입(Prototype) 패턴(생성 패턴)

④ 상태(State) 패턴(행위 패턴)

설명 : 행위 패턴은 하나의 객체로 수행할 수 없는 작업을 여러 객체로 분배하면서 결합도를 최소화 할 수 있도록 도움을 주는 패턴이다.

행위 패턴의 종류에는 책임 연쇄, 커맨드, 인터프리터, 반복자, 중재자, 메멘토, 옵서버, 상태, 전략, 템플릿 메소드, 방문자 패턴이 있다.

7. 디자인 패턴을 이용한 소프트웨어 재사용으로 얻어지는 장점이 아닌 것은?

① 소프트웨어 코드의 품질을 향상시킬 수 있다.

② 개발 프로세스를 무시할 수 있다.

③ 개발자들 사이의 의사소통을 원활하게 할 수 있다.

④ 소프트웨어의 품질과 생산성을 향상시킬 수 있다.

8. GoF(Gangs of Four) 디자인 패턴에 대한 설명으로 틀린 것은?

① Factory Method Pattern은 상위 클래스에서 객체를 생성하는 인터페이스를 정의하고, 하위클래스에서 인스턴스를 생성하도록 하는 방식이다.

② Prototype Pattern은 Prototype을 먼저 생성하고 인스턴스를 복제 하여 사용하는 구조이다.

③ Bridge Pattern은 기존에 구현되어 있는 클래스에 기능 발생 시 기존 클래스를 재사용할 수 있도록 중간에서 맞춰주는 역할을 한다.

④ Mediator Pattern은 객체간의 통제와 지시의 역할을 하는 중재자를 두어 객체지향의 목표를 달성하게 해준다.

설명 : 브리지 패턴은 구현부에서 추상층을 분리하여 서로가 독립적으 로

1. 애플리케이션 설계-SEC_08(디자인 패턴)기출 및 출제 예상 문제

기출 문제 및 출제 예상 문제(디자인 패턴)

9. GoF(Gangs of Four) 디자인 패턴 중 생성 패턴으로 옳은 것은?

① Singleton Pattern ② Adapter Pattern(구조 패턴)

③ Decorator Pattern(구조 패턴) ④ State Pattern(행위 패턴)

설명 : 생성 패턴의 종류에는 추상 팩토리, 빌더, 팩토리 메서드, 프로토타입, 싱글톤 패턴이 있다.

10. GoF(Gangs of Four) 디자인 패턴의 생성 패턴에 속하지 않는 것은?

① 추상 팩토리(Abstract Factory)

② 빌더(Builder)

③ 어댑터(Adapter) – 구조 패턴

④ 싱글톤(Singleton)

11. GoF(Gang of Four) 디자인 패턴과 관련한 설명으로 틀린 것은?

① 디자인 패턴을 목적(Purpose)으로 분류할 때 생성, 구조, 행위로 분류할 수 있다.

② Strategy 패턴은 대표적인 구조 패턴으로 인스턴스를 복제하여 사용하는 구조를 말한다.

③ 행위 패턴은 클래스나 객체들이 상호작용하는 방법과 책임을 분산하는 방법을 정의한다.

④ Singleton 패턴은 특정 클래스의 인스턴스가 오직 하나임을 보장 하고, 이 인스턴스에 대한 접근 방법을 제공한다.

설명 : Strategy 패턴은 동일한 계열의 알고리즘들을 개별적으로 캡슐화하여 상호 교환할 수 있게 정의하는 패턴이며, 클라이언트는 독립적으로 원하는 알고리즘을 선택하여 사용할 수가 있고 클라이언트에 영향 없이 알고리즘 변경이 가능하다.

프로토타입 패턴은 생성 패턴으로 인스턴스를 복제하여 사용하는 구조를 말한다.

12. GoF(Gang of Four)의 디자인 패턴에서 행위 패턴에 속하는 것은?

① Builder (생성 패턴) ② Visitor(행위 패턴)

감사합니다.