정보처리기사 필기

소프트웨어 설계 8 애플리케이션설계 ③

양문자 선생님

출처 : ncs 학습모듈(NCS능력단위 애플리케이션설계)
참고 : 시스템분석과설계/허원실/한빛아카데미
참조 : 객체지향소프트웨어공학/최은만/한빛아카데미

차례

(2) 객체지향 설계

- 객체지향이란?

: 현실 세계를 객체의 집합으로 보는 개념

- 객체(Object)?

: 객체 데이터(속성)와 행위(메소드)로 이루어진 것.

소프트웨어 설계

<객체지향 방법론의 역사>

1. 객체지향의 출현 – 객체지향 언어(1967년 노르웨이 "시뮬라")의 탄생에서 기인
2. '4대 객체지향 방법론'
   1) 부치 - Booch법
   2) 코드와 요던 – Coad/Yourdon법
   3) 슐레이어와 멜러 – Shlaer/Mellor법
   4) 럼보우 – OMT(Object-Modeling-Technique)법

소프트웨어 설계

<객체지향 방법론>

기법	원리
Rumbaugh(럼바우)의 OMT	• 클래스의 외부 명세를 정의함 • 객체 모델링, 동적 모델링, 기능 모델링으로 분류함
Booch 방법론	• 시스템 형성 구조를 모형화하는 **DFD**사용 • 클래스 다이어그램, 객체 다이어그램, 모듈 다이어그램, 프로세스 다이어그램
Coad/Yaurdon 방법론	• 객체지향 특징을 가장 충족시키는 방법 • **E-R 다이어그램**을 사용하여 객체의 행위를 모델링
shaler/Mellor 방법론	• 하나의 시스템을 몇 개의 영역으로 분할하여 서브 시스템을 구성함 • 정보 모델링, 상태 모델링, 처리 모델링

참고 : 시스템분석설계/이창희/정익사

문제풀이

1. 객체지향 분석 방법론 중 E-R 다이어그램을 사용하여 객체의 행위를 모델링하며, 객체식별, 구조 식별, 주체 정의, 속성 및 관계 정의, 서비스 정의 등의 과정으로 구성되는 것은?

① Coad와 Yourdon 방법　② Booch 방법
③ Jacobson 방법　④ Wirfs-Brocks 방법

(2020년 1, 2회 정보처리기사 필기 기출문제 소프트웨어 설계)

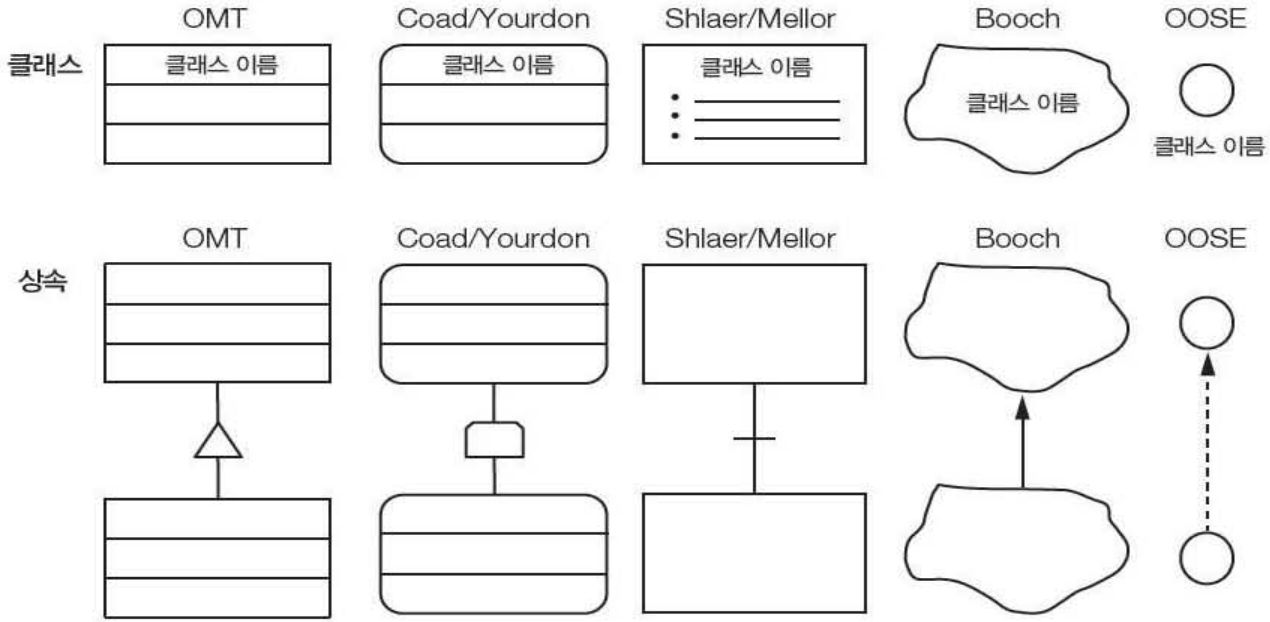
소프트웨어 설계

<객체지향 방법론>

- 소프트웨어 개발 과정은 기존 시스템을 분석하고 이를 표준화된 새로운 시스템 환경으로 전환하는데 초점을 맞추게 되었다.
- 소프트웨어를 개발할 때 작은 단위의 모듈들을 구성하여 만들고 추후 이를 재사용하여 소프트웨어의 효율성을 높이자는 생각에서 객체지향 방법론이 발전하게 되었다.
- 여러 객체지향 방법론들이 난립하게 되고 표기법만이라도 통일하자는 제안에 의해 UML을 사용하게 되었다.
- 객체, 클래스, 캡슐화, 데이터 은닉, 상속, 조합 다형성의 개념

<표기법의 차이>

- 객체지향 방법론 표기법의 차이가 문제로 대두
  - 레셔널사는 표기법만이라도 통일하자는 취지로 UML을 제안



소프트웨어 설계 방법론

<SOLID (객체 지향 설계)>

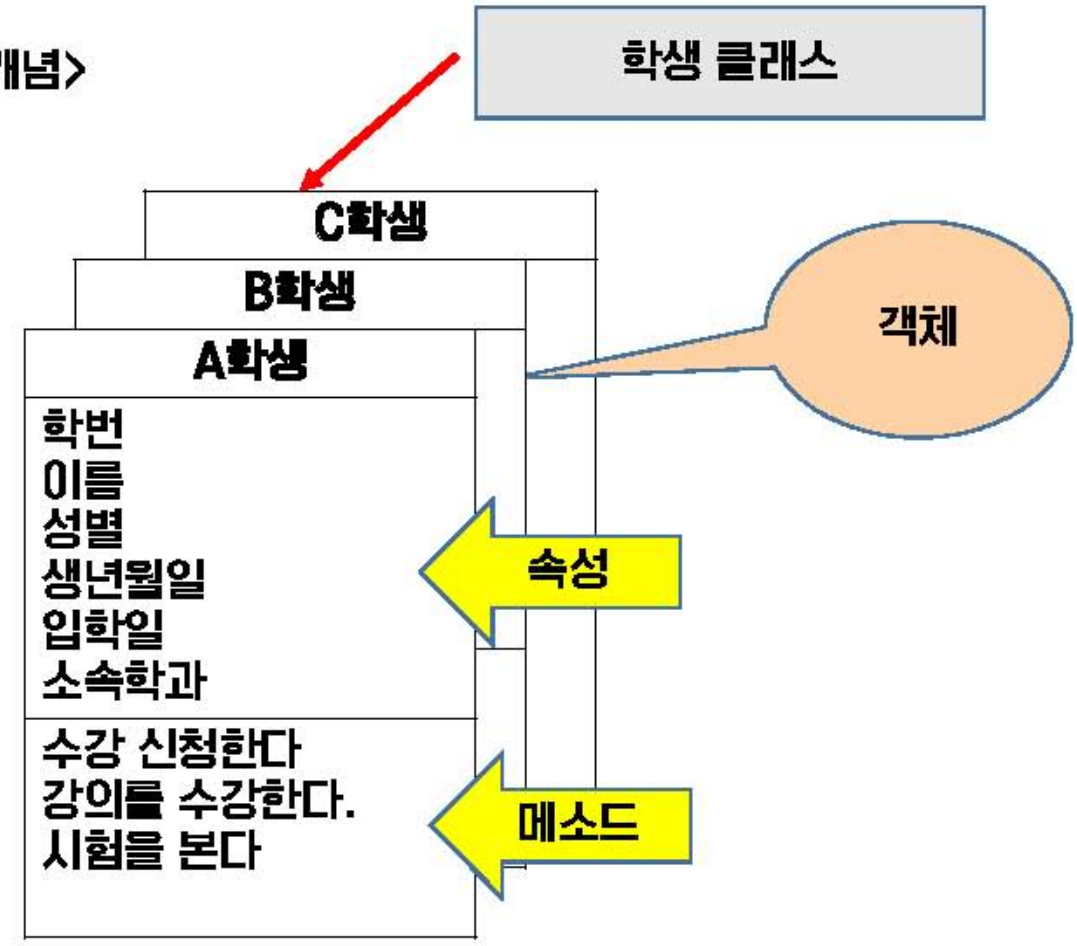
컴퓨터 프로그래밍에서 SOLID란 로버트 마틴이 2000년대 초반에 명명한 객체 지향 프로그래밍 및 설계의 다섯 가지 기본 원칙을 마이클 페더스가 두문자어 기억술로 소개한 것이다.

S	SRP	단일 책임 원칙 (Single responsibility principle)한 클래스는 하나의 책임만 가져야 한다.
O	OCP	개방-폐쇄 원칙 (Open/closed principle)"소프트웨어 요소는 확장에는 열려 있으나 변경에는 닫혀 있어야 한다."
L	LSP	리스코프 치환 원칙 (Liskov substitution principle)"프로그램의 객체는 프로그램의 정확성을 깨뜨리지 않으면서 하위 타입의 인스턴스로 바꿀 수 있어야 한다." 계약에 의한 설계를 참고하라.
I	ISP	인터페이스 분리 원칙 (Interface segregation principle)"특정 클라이언트를 위한 인터페이스 여러 개가 범용 인터페이스 하나보다 낫다."
D	DIP	의존관계 역전 원칙 (Dependency inversion principle)프로그래머는 "추상화에 의존해야지, 구체화에 의존하면 안된다." 의존성 주입은 이 원칙을 따르는 방법 중 하나다.

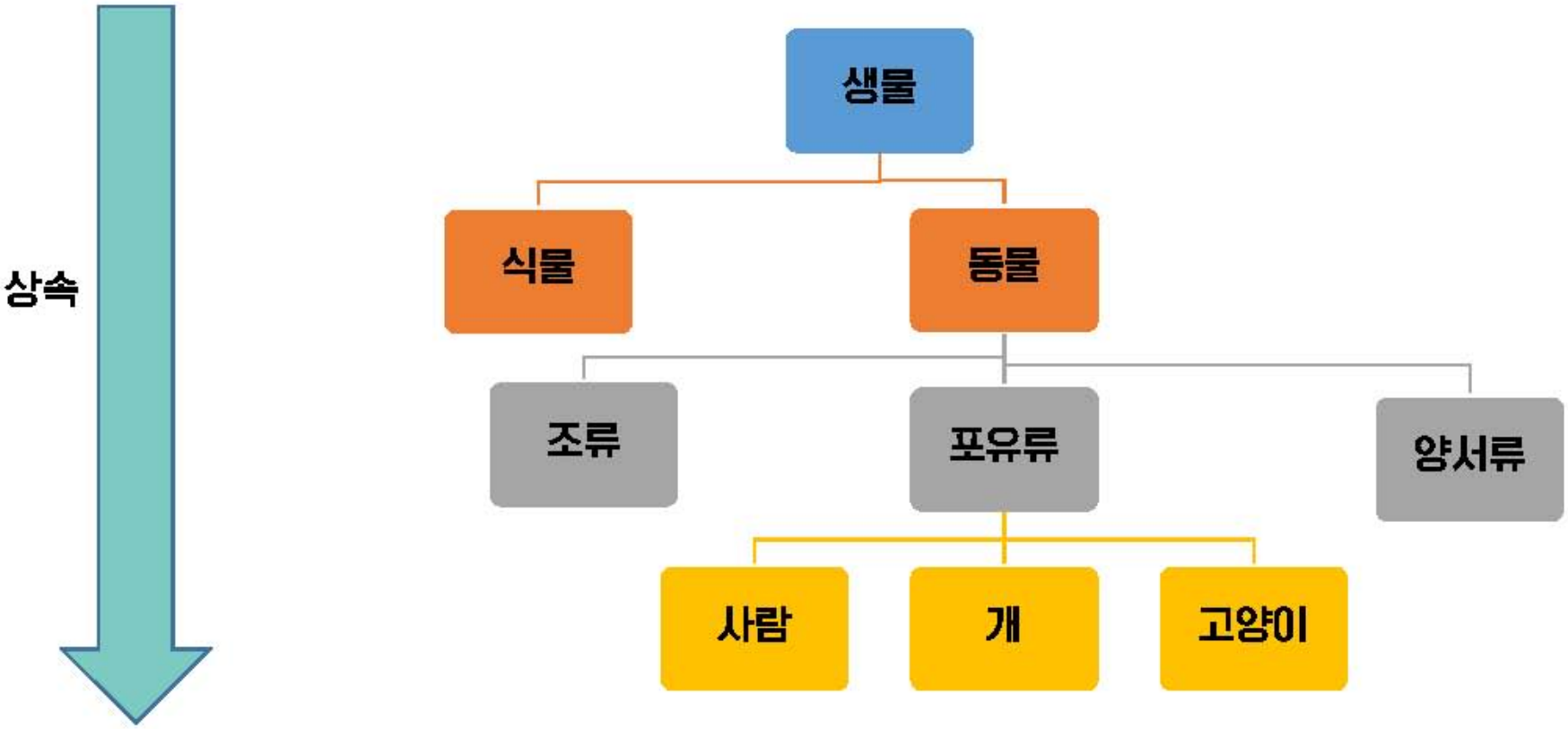
- 위키백과 -

소프트웨어 설계

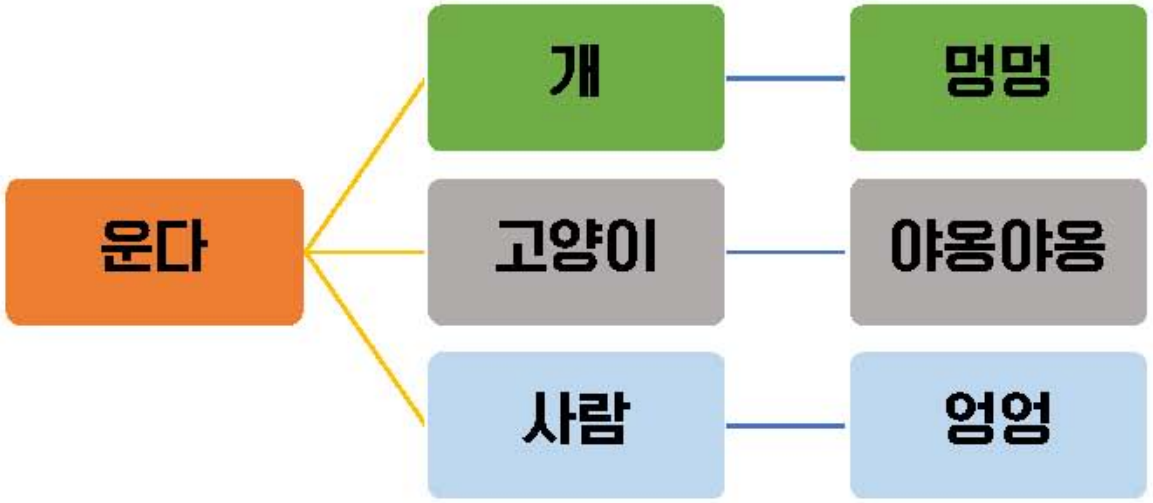
<객체지향의 핵심 개념>



소프트웨어 설계



다형성



소프트웨어 설계

<객체지향의 핵심 개념>

1. 객체 – 정보를 효율적으로 관리하기 위한 개념적인 단위
2. 클래스 – 공통속성과 행위를 가진 객체를 묶어 추상화한 개념
3. 속성 – 객체가 가지고 있는 데이터 값
4. 메소드 – 클래스로부터 생성된 객체를 사용하는 방법, 연산
5. 상속 – 상위클래스의 자료구조와 연산을 그대로 이어받아 사용할 수 있게 해주는 것
6. 다형성 – 같은 연산인데 다르게 수행하는 것
7. 메시지 – 객체들을 연결하는 수단, 객체에게 지시하는 방법
8. 캡슐화 – 자료구조와 연산을 묶어 놓은것. 불법적인 사용으로부터 자료보호
9. 정보 은닉 – 다른 객체에게 자신의 자료를 숨기고, 연산을 통해서만 그 자료에 대한 접근 허용

문제풀이

2. 객체지향에서 정보 은닉과 가장 밀접한 관계가 있는 것은?
① Encapsulation
② Class
③ Method
④ Instance

(2020년 3회 정보처리기사 필기 기출문제 소프트웨어 설계)

3. 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**

(2020년 1, 2회 정보처리기사 필기 기출문제 소프트웨어 설계)

소프트웨어 설계

<디자인 패턴>

- **디자인 패턴**(Design pattern)은 건축학 및 컴퓨터 과학에서 사용되는 용어로, 설계 문제에 대한 해답을 문서화하기위해 고안된 형식 방법이다. 이 방식은 건축가 크리스토퍼 알렉산더가 건축학 영역에서 고안한 것을 그 시초로 하며, 이후 컴퓨터 과학 등 여러 다른 분야에도 도입되었다.

- GoF(Gang of Four)라 불리는 네명의 컴퓨터 과학 연구자들이 쓴 서적 'Design Patterns: Elements of Reusable Object-Oriented Software'(재이용 가능한 객체지향 소프트웨어의 요소 - 디자인 패턴)이다. GoF는 컴퓨터 소프트웨어 공학 분야의 연구자인 에릭 감마, 리차드 헬름, 랄프 존슨, 존 블리시디스의 네명을 지칭한다.

-위키백과-

소프트웨어 설계

<디자인 패턴>

1. 반복적으로 나타나는 문제점들에 대한 해결 방안을 제시한 것.
2. 다양한 응용 소프트웨어 시스템들을 개발할 때 존재하는 공통되는 설계문제, 공통되는 해결책을 묶어 패턴이라고 함
3. 팀원 사이의 의사소통을 원활하게 해주는 역할
4. UML과 같은 일종의 설계 기법이며, 설계방법이다.
5. SW설계시 SW 재사용성, 호환성, 유지보수성을 보장하여 '올바른 설계 ' 를 빠르게 만들수 있도록 도와줌

<디자인 패턴 구성요소>

패턴 이름(Pattern Name)	패턴의 이름은 해당 패턴의 솔루션을 담고 있다.
문제(problem)	패턴이 적용되어 해결될 필요가 있는 디자인
해결(solution)	문제를 해결하기 위한 요소들 사이의 관계, 책임, 협력관계 다양한 상황에 적용할수 있는 템플릿과 같음
결과(Consequence)	패턴을 적용해서 얻은 결과와 장단점을 서술한다.

<디자인 패턴 분류>

생성(Creational)패턴

- 객체 생성에 관련된 패턴

구조(Structural)패턴

- 클래스나 객체를 조합해 더 큰 구조를 만드는 패턴

행위(Behavioral)패턴

- 객체나 클래스 사이의 알고리즘이나 책임 분배에 관련된 패턴

소프트웨어 설계

<GoF 디자인 패턴 분류>

생성패턴	구조패턴	행위패턴
• 간단한 블록의 코드로 여러 가지 다양한 객체를 생성 • 실행 시간에 객체의 다양한 버전을 생성 • 생성한 객체에 제한을 가함	• 더 큰 구조를 형성하기 위하여 클래스와 객체를 어떻게 합성 하는가? • 상속 기법 이용 – 인터페이스, 구현을 합성 • 구조화 패턴 – 새로운 기능을 구현하기 위하여 객체를 구성하는 방식 – 런타임에 객체 구조를 변경 – 유연성, 확장성	• 반복적으로 일어나는 객체들의 상호작용을 패턴화 해 놓은 것 • 객체들 간의 알고리즘이나 역할 분담에 관련된 것

소프트웨어 설계

<GoF 디자인 패턴 분류>

생성패턴	구조패턴	행위패턴
• 추상팩토리(Abstract Factory) • 빌더(Builder) • 팩토리메서드(Factory Method) • 프로토타입(Prototype) • 싱글톤(Singleton)	• 어댑터(Adpter) • 브리지(Bridge) • 컴퍼지트(Composite) • 데커레이터(Decorator) • 퍼사드(Facade) • 플라이웨이트(Flyweight) • 프록시(Proxy)	• 책임연쇄(Chain of Responsibility) • 커맨드(Command) • 인터프리터(Interpreter) • 이터레이터(Iterator) • 미디에이터(Mediator) • 메멘토(Memento) • 옵서버(Observer) • 스테이트(State) • 스트래티지(Strategy) • 템플릿메서드(Template Method) • 비지터(Visitor)

소프트웨어 설계

<디자인 패턴 분류>

생성(Creational)패턴

- 객체 생성에 관련된 패턴

추상 팩토리(Abstract Factory)

- 공장을 만들어내는 상위 공장을 먼저 정의하고, 여기서 구체적인 공장을 만든 후, 이 공장에서 객체 생성
- 구체적인 클래스에 의존하지 않음
- 서로 연관된 객체들의 조합을 만드는 인터페이스를 제공

팩토리 메서드

- 인스턴스를 만들어내는 공장를 통해 객체 생성
- 객체 생성 처리를 서브 클래스로 분리해 처리하도록 캡슐화

프로토타입(Prototype)

- 기존의 인스턴스를 그대로 복제(clone) 하여 새로운 객체를 생성하는 방법

소프트웨어 설계

<디자인 패턴 분류>

생성(Creational)패턴

- 객체 생성에 관련된 패턴

빌더 (Builder)

- 객체를 생성할 때 필요한 파라미터가 많은 경우, 각 파라미터가 무엇을 의미하는지 빌더라는 형태를 통해 명확히하여 생성할 수 있다.

싱글톤(Signleton)

- 전역변수를 사용하지 않음
- 객체를 하나만 생성
- 생성된 객체를 어디에서든지 참조할 수 있도록 함

소프트웨어 설계

<디자인 패턴 분류>

구조(Structural)패턴

- 클래스나 객체를 조합해 더 큰 구조를 만드는 패턴

컴퍼지드(Composite)

- 여러 객체들로 구성된 복합 객체와 단일 객체를 클라이언트에서 구별없이 다루게 해줌
- 어떤 클래스(Composite)가 자기 자신 혹은 다른 클래스(Leaf)를 가질 수 있는 구조

데커레이터(Decorator)

- 객체의 결합을 통해 기능을 유연하게 확장하게 해줌
- 각각의 기능을 담당하는 클래스(Decorator)들과 이 기능을 적용할
- 클래스(Component)를 분리한 뒤,
- 필요에 따라 동적으로 각 기능을 적용할 수 있는 구조

소프트웨어 설계

<디자인 패턴 분류>

구조(Structural)패턴

- 클래스나 객체를 조합해 더 큰 구조를 만드는 패턴

어댑터 (Adapter)

- 서로 다른 두 클래스(Client 와 Adaptee) 가 있고, 이 둘은 그대로 둔 채 이 둘의 인터페이스를 연결하고자 어댑터 클래스를 만들어 사용하는 구조

프록시 (Proxy)

- 구체적인 업무를 담당하고 있는 클래스에 접근하기 전에, 간단한 사전 작업 처리하는 클래스(Proxy)를 두는 구조

소프트웨어 설계

<디자인 패턴 분류>

행위(Behavioral)패턴

- 객체나 클래스 사이의 알고리즘이나 책임 분배에 관련된 패턴

옵서버(Observer)	스테이트(State)	스트래터지(Strategy)	방문자 (Visitor)
• 한 객체의 상태 변화에 따라 다른 객체의 상태도 연동 • 일대다 객체 의존관계 • 하나의 관찰대상 - 여러 개의 관찰자 구조	• 객체의 상태에 따라 객체의 행위 내용 변경	• 동일계열의 알고리즘들을 인터페이스-캡슐화하고, 알고리즘들을 컴포지션(위임 형태로) 가지는 패턴	• 방문자와 방문 공간을 분리하여, 방문 공간이 방문자를 맞이할 때, 이후에 대한 행동을 방문자에게 위임하는 패턴

소프트웨어 설계

<디자인 패턴 분류>

행위(Behavioral)패턴

- 객체나 클래스 사이의 알고리즘이나 책임 분배에 관련된 패턴

책임 연쇄 (Chain of responsibility)

- 요청을 처리하는 동일 인터페이스 객체들을 체인 형태로 연결해놓는 패턴

템플릿 메서드

- 어떤 작업을 처리하는 일부분을 서브 클래스로 캡슐화
- 전체 일을 수행하는 구조는 그대로 특정단계에서 수행하는 내역을 바꾸는 패턴

커맨드(Command)

- 특정 객체에 대한 커맨드를 객체화하여
  이 커맨드 객체를 필요에 따라 처리하는 패턴이다.
- 실행될 기능을 캡슐화
- 주어진 여러 기능을 실행할 수 있는 재사용성이 높은 클래스를 설계

문제풀이

1. 디자인 패턴 사용의 장·단점에 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.

2. GoF(Gangs of Four) 디자인 패턴 분류에 해당하지 않는 것은?
① 생성 패턴
② 구조 패턴
③ 행위 패턴
④ 추상 패턴

(2020년 4회 정보처리기사 필기 기출문제 소프트웨어 설계)

문제풀이

3. 디자인 패턴 중에서 행위적 패턴에 속하지 않는 것은?
① 커맨드(Command) 패턴
② 옵저버(Observer) 패턴
③ 프로토타입(Prototype) 패턴
④ 상태(State) 패턴

(2020년 3회 정보처리기사 필기 기출문제 소프트웨어 설계)

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

① Builder
② Visitor
③ Prototype
④ Bridge

(2020년 1, 2회 정보처리기사 필기 기출문제 소프트웨어 설계)

문제풀이

5. 디자인 패턴에 대한 설명으로 옳지 않은 것은?
① 일반적으로 디자인 패턴을 이용하면 좋은 설계나 아키텍처를 재사용하기 쉬워진다.
② 패턴은 사용 목적에 따라서 생성 패턴, 구조 패턴, 행위 패턴으로 분류할 수 있다.
③ 생성 패턴은 빌더(builder), 추상 팩토리(abstract factory) 등을 포함한다.
④ 행위 패턴은 가교(bridge), 적응자(adapter), 복합체(composite) 등을 포함한다.

(2015_9급_지방직_컴퓨터일반(A)_2015년 06월 27일 컴퓨터일반))

6. 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

(2007_7급_국가직_소프트웨어공학(공)_2007년 08월 09일 소프트웨어공학)