면접 질문 리스트

[면접 팁]

과정교재(코스웍), cep을 쭈욱 훓어보고 나야 면접 대비가 좋다.

디자인패턴은 확실히 알아야 한다. 특히 컴포넌트의 용어를 중요하게 생각하는 것 같다.

ConcreateClass라던지, 뭐 그런것들 있잖아요.

패턴에 나오는 네모 박스들에 붙는 이름들 그게 패턴마다 조금 씩 달라서 그런걸 정확히 이야기 하면 좋아하실것 같다.

그리고 각 패턴이 결국 어떤 문제를 해결하는 정형화된 방법이니 이 문제는 어떤 패턴으로 해결한다. 이런게 정리가 되어야 있어야 함.

[김수동 교수님의 prefinal 리뷰시간 면접준비 정리]

1)2분 정도 내가 하는 일과 AA느낀점 말하는 시간

예상답 :

2)이 패턴은 언제 사용하는게 좋을까?

예상답 : ex) composite패턴 : 복수들의 객체들을 트리형태로 recursive하게 표현하는 패턴으로, 사용자가 단일 객체와 복합 객체 모두 동일하게 다루도록 한다. 예로 폴더구조

데코레이터 : 기능 추가할때 반복해서 쓴다. 상속에 대비했을때 이런장점 있다. 무슨장점???

3)ocp:부모의 공통적인 것에는 닫혀 있고, 자식한테는 열려있다.

4) 동적바인딩

예상답 :

5) what is the information hiding mechanism in OOP?

예상답 : 모듈간에 디펜던시를 낮춘다(커플링이 낮아짐, 모듈성이 올라감, 유지보수가 좋아진다.)

예를 들면 C의 전역변수를 사용하면 디펜던시가 많아진다.

6) Consistency

예상답 : state machine diagram & class diagram : class 하나당 state diagram하나

객체 상태변화(event)

7) 언제 activity다이어그램을 사용할까?

예상답 :

8)

예상답 :

[선배들이 받은 질문]

1)디자인 원칙중(solid)에서 무엇이 제일 중요하다고 생각하나요? 왜 그렇게 생각하나요?

예상답: SRP 가 가장 중요하다고 생각합니다. 객체지향에서 가장 기본이자 중요한 부분은 각 클래스나 모듈에 책임을 할당하는것이라고 생각합니다.

무엇이 가장 중요할까요? 리즈닝을 잘 해야 할텐데...

|  |  |  |
| --- | --- | --- |
| **두문자** | **약어** | **개념** |
| **S** | [SRP](https://ko.wikipedia.org/wiki/%EB%8B%A8%EC%9D%BC_%EC%B1%85%EC%9E%84_%EC%9B%90%EC%B9%99) | **[단일 책임 원칙 (Single responsibility principle)](https://ko.wikipedia.org/wiki/%EB%8B%A8%EC%9D%BC_%EC%B1%85%EC%9E%84_%EC%9B%90%EC%B9%99)**  한 [클래스](https://ko.wikipedia.org/wiki/%ED%81%B4%EB%9E%98%EC%8A%A4_(%EC%BB%B4%ED%93%A8%ED%84%B0_%EA%B3%BC%ED%95%99))는 하나의 책임만 가져야 한다. |
| **O** | [OCP](https://ko.wikipedia.org/wiki/%EA%B0%9C%EB%B0%A9-%ED%8F%90%EC%87%84_%EC%9B%90%EC%B9%99) | **[개방-폐쇄 원칙 (Open/closed principle)](https://ko.wikipedia.org/wiki/%EA%B0%9C%EB%B0%A9-%ED%8F%90%EC%87%84_%EC%9B%90%EC%B9%99)**  “소프트웨어 요소는 확장에는 열려 있으나 변경에는 닫혀 있어야 한다.” |
| **L** | [LSP](https://ko.wikipedia.org/wiki/%EB%A6%AC%EC%8A%A4%EC%BD%94%ED%94%84_%EC%B9%98%ED%99%98_%EC%9B%90%EC%B9%99) | **[리스코프 치환 원칙 (Liskov substitution principle)](https://ko.wikipedia.org/wiki/%EB%A6%AC%EC%8A%A4%EC%BD%94%ED%94%84_%EC%B9%98%ED%99%98_%EC%9B%90%EC%B9%99)**  “프로그램의 [객체](https://ko.wikipedia.org/wiki/%EA%B0%9D%EC%B2%B4)는 프로그램의 정확성을 깨뜨리지 않으면서 하위 타입의 인스턴스로 바꿀 수 있어야 한다.” [계약에 의한 설계](https://ko.wikipedia.org/w/index.php?title=%EA%B3%84%EC%95%BD%EC%97%90_%EC%9D%98%ED%95%9C_%EC%84%A4%EA%B3%84&action=edit&redlink=1)를 참고하라. |
| **I** | [ISP](https://ko.wikipedia.org/wiki/%EC%9D%B8%ED%84%B0%ED%8E%98%EC%9D%B4%EC%8A%A4_%EB%B6%84%EB%A6%AC_%EC%9B%90%EC%B9%99) | **[인터페이스 분리 원칙 (Interface segregation principle)](https://ko.wikipedia.org/wiki/%EC%9D%B8%ED%84%B0%ED%8E%98%EC%9D%B4%EC%8A%A4_%EB%B6%84%EB%A6%AC_%EC%9B%90%EC%B9%99)**  “특정 클라이언트를 위한 인터페이스 여러 개가 범용 인터페이스 하나보다 낫다.”[[4]](https://ko.wikipedia.org/wiki/SOLID_(%EA%B0%9D%EC%B2%B4_%EC%A7%80%ED%96%A5_%EC%84%A4%EA%B3%84)#cite_note-martin-design-principles-4) |
| **D** | [DIP](https://ko.wikipedia.org/wiki/%EC%9D%98%EC%A1%B4%EA%B4%80%EA%B3%84_%EC%97%AD%EC%A0%84_%EC%9B%90%EC%B9%99) | **[의존관계 역전 원칙 (Dependency inversion principle)](https://ko.wikipedia.org/wiki/%EC%9D%98%EC%A1%B4%EA%B4%80%EA%B3%84_%EC%97%AD%EC%A0%84_%EC%9B%90%EC%B9%99)**  프로그래머는 “추상화에 의존해야지, 구체화에 의존하면 안된다.”[[4]](https://ko.wikipedia.org/wiki/SOLID_(%EA%B0%9D%EC%B2%B4_%EC%A7%80%ED%96%A5_%EC%84%A4%EA%B3%84)#cite_note-martin-design-principles-4) [의존성 주입](https://ko.wikipedia.org/wiki/%EC%9D%98%EC%A1%B4%EC%84%B1_%EC%A3%BC%EC%9E%85)은 이 원칙을 따르는 방법 중 하나다. |

2)생성자 패턴에 대해서 예를 들고 설명을 해봐라

예상답:

생성(Creational) 패턴

객체 생성에 관련된 패턴

객체의 생성과 조합을 캡슐화해 특정 객체가 생성되거나 변경되어도 프로그램 구조에 영향을 크게 받지 않도록 유연성을 제공한다.

추상 팩토리(Abstract Factory)

구제적인 클래스에 의존하지 않고 서로 연관되거나 의존적인 객체들의 조합을 만드는 인터페이스를 제공하는 패턴

팩토리 메서드(Factory Method)

객체 생성 처리를 서브 클래스로 분리해 처리하도록 캡슐화하는 패턴

싱글턴(Singleton)

전역 변수를 사용하지 않고 객체를 하나만 생성하도록 하며, 생성된 객체를 어디에서든지 참조할 수 있도록 하는 패턴

https://gmlwjd9405.github.io/2018/07/06/design-pattern.html

3)시퀀스 다이어그램 fragments에 대해서 (loop, opt, alt 등) 설명

예상답: alt은 워크 플로우의 다른 시나리오를 설명하는 데 사용됩니다. 옵션 중 하나만 실행됩니다.

opt은 워크 플로의 선택적 단계를 설명하는 데 사용됩니다.

예를 들어 온라인 쇼핑 구매 순서 다이어그램의 경우 사용자가 원하는 경우 사용자가 선물 포장을 추가하는 방법을 설명하기 위해 opt을 사용할 수 있습니다. alt은 신용 카드 또는 은행 송금을 사용하는 두 가지 지불 방법을 설명하는 데 사용될 수 있습니다

loop는 반복문

<http://blog.naver.com/phrack/80040621662>

<http://egloos.zum.com/gggura/v/3452986>

인터렉션 연산자  
  
의미: 시퀀스 다이어그램의 반족, 조건, 병행 처리 등을 표현  
  
표현: 메시지를 포함하는 사각 테두리(Frame)의 좌측 상단에 기술  
  
종류  
  
- break: loop를 벗어날 경우  
  
- par: parallel 병렬로 수행하는 경우  
  
- seq: sequence 메시지 전달 순서가 엄격하지 않은 경우  
  
- strict: 메시지 전달 순서가 엄격한 경우  
  
- neg: negative 절대 발생하면 안되는 상황  
  
- critical: atomic하게 처리  
  
- ignore: 특정 메시지를 무시할 경우  
  
- consider: 특정 메시지를 중요하게 고려할 경우

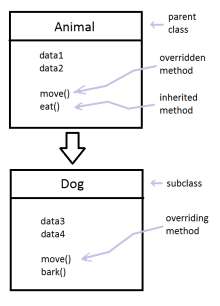
4) 오버로딩, 오버라이딩 설명 해보세요.

예상답:

오버로딩 :

같은 함수 이름을 가지고 있으나 매개변수, 리턴타입 등의 특징은 다른 여러개의 [서브프로그램](https://ko.wikipedia.org/w/index.php?title=%EC%84%9C%EB%B8%8C%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%A8&action=edit&redlink=1) 생성을 가능하게 한다.

오버라이딩 :



메소드 오버라이딩 그림.

**메소드 오버라이딩**,(method overriding)은 [객체 지향 프로그래밍](https://ko.wikipedia.org/wiki/%EA%B0%9D%EC%B2%B4_%EC%A7%80%ED%96%A5_%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%B0%8D)에서 서브클래스 또는 자식 클래스가 자신의 슈퍼클래스들 또는 부모 클래스들 중 하나에 의해 이미 제공된 [메소드](https://ko.wikipedia.org/wiki/%EB%A9%94%EC%86%8C%EB%93%9C_(%EC%BB%B4%ED%93%A8%ED%84%B0_%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%B0%8D))를 특정한 형태로 구현하는 것을 제공하는 언어의 특징이다. 서브클래스에서의 구현은 부모 클래스에서 같은 이름, 같은 파라미터 또는 시그니처 그리고 같은 반환형을 갖는 메소드를 제공함으로써 슈퍼클래스에서의 구현을 오버라이드한다.[[1]](https://ko.wikipedia.org/wiki/%EB%A9%94%EC%86%8C%EB%93%9C_%EC%98%A4%EB%B2%84%EB%9D%BC%EC%9D%B4%EB%94%A9#cite_note-flanagan-1)

- 함수를 바인딩하는 2가지 방법

(1) 정적 바인딩 (일반 함수)

컴파일 시간에 호출될 함수로 점프할 주소가 결정되어 바인딩 되는 것.

(2) 동적 바인딩 (가상 함수)

실행 파일을 만들 때 바인딩 되지 않고 보류 상태 둔다.

점프할 메모리 번지를 저장하기 위한 메모리 공간(4 byte)을 가지고 있다가 런타임에 결정.

=> 단점 : 타입 체킹으로 인한 수행 속도 저하 / 메모리 공간 낭비

어떤 포인터에 의해 접근되었는 지에 상관없이 참조된 인스턴스의 실제 클래스형에 따라 재정의된 함수 호출이 가능!

5) 현재 하고 있는 프로젝트에서 생각하는 중요한 nfr이 무엇인지 말해보세요.

예상답:

6) 사용 해봤던 디자인 패턴은 무엇인가?

예상답:

7) complexity관련해서 설명 해보세요.

예상 답: 시간복잡도와 공간 복잡도가 있다.

**시간 복잡도**는 문제를 해결하는데 걸리는 시간과 입력의 함수 관계를 가리킨다.

공간 복잡도(Space Complexity)란, 프로그램을 실행시킨 후 완료하는 데 필요로 하는 자원 공간의 양을 말합니다.

8) UML다이어그램 머 써봤냐?

CLASS 다이어그램 많이 써봤다고 하니까

CLASS 다이어그램의 relationship이 뭘 의미 하냐고 물어봐서 대답을 잘 못했는데, class instance들 사이의 관계라고 알려주시더라고요. 그거 말고도 잔뜩 있었는데 기억이 안나요….

\*클래스 다이어그램을 깊게 물어봤다고 합니다.

무엇을 물어볼지에 대해서 같이 대비 해보면 좋을것 같아요.

예상답:

9)과정중에 제일 인상깊었던것은 무엇이냐?

예상답: 사람마다 다를것 같음

성훈님은 ai 관련해서 많이 물어볼것 같은데…AI 경험이 있나요?

저는 없죠….

10)blackboard 물어봄 ….

예상답:

"해법이 어려운 문제에 대해서 답을 아는 사람이 해법을 적는 것 처럼   
Black Board가 이벤트를 발생해 Knowledge Source들이 이에 대한 답을 제시하는 방식입니다. 그런데 이 style은 mission-critical system에 쓰일수 없습니다. knowledge source의 정확성이 보장이 되지 않기 때문입니다. 그래서 게시판 같은 open source community에 쓰이는 style입니다."

11) 프레임웍팀에서는 패턴 머 써봤냐?

예상 답) 써본것 없었고 aa 과정에서 배운걸 기초로 해서 현업에서 적용 예정(어떤 어떤걸 적용 예정) 이건 예상 질문과 같이 나와서 자신을 낮추고 교수님한테 배운게 짱이라는 인식을 심게 했음

근데 이건 분위기 보면서 전략을 변경 해야 할것 같다고 말함.(정말 잘하는 사람을 원한다 vs 이번과정을 통해서 현업에 적용가능성)

12) 브로커 스타일의 장단점?

예상답:

등록된 서버와 클라이언트 사이에서 이종간의 통신을 중개한다.

Client - Broker - Server

장점 : 브로커를 중심으로 서버와 클라이언트는 바꿀수 있고 확장 가능하며 재사용할 수 있다. 또한 런타임에 컴포넌트도 변경할 있다.

단점: Broker에 fault가 발생하면 문제가 커질 수 있고 테스트도 어렵다.(많은 양의 proxy들 때문에)

프록시들의 오버헤드때문에 비효율성

13) 액티비티 다이어그램과 시퀀스 다이어그램의 차이점?

예상답: 시퀀스 다이어그램은 하나의 유스케이스를 시간의 순서대로 각 기능 모듈간에 어떠한 메시지를 주고 받는지를 표현한것이고, 액티비티 다이어그램은 전체시스템의 action들이 순차,병행 방식으로 어떻게 수행되는지를 표현한다.

* Activity diagram is good at depicting **the control flowing from one activity to another**, especially good at the logic of conditional structures, loops, concurrency.
* Sequence diagram is good at depicting **the sequence of messages flowing from one object to another**, how their messages/events are exchanged in what time-order.

14) 템플릿 메소드 패턴 설명?

예상답: 알고리즘의 구조는 고정되어있고 알고리즘의 일부를 서브클래스가 재정의한다.

15) c++과 자바의 차이 설명?

예상답:

* **C++**는 이름에서 나타나듯이 [C](https://ko.wikipedia.org/wiki/C_(%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%B0%8D_%EC%96%B8%EC%96%B4))를 더 확장하여 만들었다. [절차적 프로그래밍](https://ko.wikipedia.org/wiki/%EC%A0%88%EC%B0%A8%EC%A0%81_%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%B0%8D) 언어에 효율적인 실행을 목표로 설계되었다. 정적 자료형 검사 [객체 지향 프로그래밍](https://ko.wikipedia.org/wiki/%EA%B0%9D%EC%B2%B4_%EC%A7%80%ED%96%A5_%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%B0%8D), [예외 처리](https://ko.wikipedia.org/wiki/%EC%98%88%EC%99%B8_%EC%B2%98%EB%A6%AC), [RAII](https://ko.wikipedia.org/w/index.php?title=RAII&action=edit&redlink=1), [제네릭 프로그래밍](https://ko.wikipedia.org/wiki/%EC%A0%9C%EB%84%A4%EB%A6%AD_%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%B0%8D)을 지원한다. 범용 컨터이너와 알고리즘을 포함한 [C++ 표준 라이브러리](https://ko.wikipedia.org/wiki/C%2B%2B_%ED%91%9C%EC%A4%80_%EB%9D%BC%EC%9D%B4%EB%B8%8C%EB%9F%AC%EB%A6%AC)도 추가되었다.
* **자바**는 처음에는 가전제품에 탑재되어 네트워크 컴퓨팅을 지원하기 위해서 만들었다. 가상 머신 위에서 실행되기 때문에 안전성을 가지며 또한 이식성이 높다. 하위 플랫폼을 완벽히 추상화시켜 주는 광대한 분량의 라이브러리를 가지고 있다. 자바는 C와 비슷한 문법을 사용할뿐 직접적인 호환성은 없다. 사용하기 편하고 많은 사람이 이해하기 쉬운 언어를 목표로 설계되었다.

두 언어는 개발의 목적이 다르기 때문에 결과적으로 서로 다른 원리, 방침, 설계에서 트레이드오프에 차이가 생겼다.

|  |  |
| --- | --- |
| **C++** | **자바** |
| C 소스 코드와 하위 호환성 | 다른 언어와 소스 코드 호환성은 없음 |
| 직접적인 시스템 라이브러리 호출 가능 | [자바 네이티브 인터페이스](https://ko.wikipedia.org/wiki/%EC%9E%90%EB%B0%94_%EB%84%A4%EC%9D%B4%ED%8B%B0%EB%B8%8C_%EC%9D%B8%ED%84%B0%ED%8E%98%EC%9D%B4%EC%8A%A4)를 이용 |
| 저수준 시스템 접근 가능 | 안전하게 보호되는 가상 머신 위에서 실행됨 |
| 선택적 자동 [경계 검사](https://ko.wikipedia.org/wiki/%EA%B2%BD%EA%B3%84_%EA%B2%80%EC%82%AC) | 항상 자동 경계 검사함 |
| 부호없는(unsigned) 연산 지원 | 부호없는 연산 지원 안 함 |
| 값에 의한 매개변수 전달 또는 참조에 의한 매개변수 전달 | 항상 값에 의한 매개변수 전달. *매개변수로 객체에 대한 참조값을 사용할 수는 있다. 참조 대상의 내용을 변경할 수는 있지만, 참조값 자체는 변경할 수 없다; 메서드 호출 후에도 참조하는 객체는 다른 객체로 바뀌지 않을 것이다.* |
| 명시적 메모리 관리, 가비지 콜렉션은 추가적으로 라이브러리를 이용해야 함 | 항상 자동 가비지 콜렉션 |
| 명시적인 자료형 재정의 허용 | 자료형 안전성에 엄격함 |
| [C++ 표준 라이브러리](https://ko.wikipedia.org/wiki/C%2B%2B_%ED%91%9C%EC%A4%80_%EB%9D%BC%EC%9D%B4%EB%B8%8C%EB%9F%AC%EB%A6%AC)는 적절한 범위까지 지원함 | 광대한 분량의 라이브러리 |
| [연산자 오버로딩](https://ko.wikipedia.org/wiki/%EC%97%B0%EC%82%B0%EC%9E%90_%EC%98%A4%EB%B2%84%EB%A1%9C%EB%94%A9) | 연산자는 재정의 할 수 없음 |

C++는 강력하지만 복잡하고 어려운 언어로, 성능 위주의 응용 프로그램이나 라이브러리에 적당하다. 자바는 대개 배우기 쉽지만, 플랫폼 자체가 가지는 전체 기능 이용이나 완벽한 성능 활용을 기대하기는 어렵다.

16) association과 aggregation의 차이 설명?

예상답:

객체 간의 관계를 나타내는 방법중 하나로 association은 주기적으로 통신하는 객체간의 관계 aggregation은 whole-part 관계입니다.

17) cep에서 옵저버 패턴을 사용했는 데, ocp로 보면 어떤게 오픈되어 있고 어떤게 클로즈 되어있는지 물어봤음

예상답:

중간에, 관찰 대상 객체를 변경하지 않고도 새로운 관찰하는 객체를 추가할 수 있도록 해서 ...   
관찰 대상 객체(subject)가 close  
새로운 관찰 객체(new observers)가 open

18) 시퀀스 다이어그램에서 critical이 무엇인가?

예상답:

critical: 하나의 쓰레드만 접근하여야 하는 critical region (section)을 표현하는 operator입니다. (즉 atomic하게 처리)

19) 스테이트 다이어그램에서 orthogonal 스테이트가 무엇인지에 대해서 물어봤음

예상답: ? compatible and independent 하게 동시에 존재하는 2개 이상의 스테이트

20) 디자인패턴중 유사한 패턴들이 어떤건지 물어봤다.

예상답:

21) cep과제에서 nfr 2번,3번을 선택했는데 왜 선택했는지 기술적으로 설명하세요.

예상답:

22) 현업으로 복귀시 배운 내용을 어떻게 활용 할것 인가?

예상답: 요구사항 정제, 품질 향상, Stakeholder 참여 유도->안심

23) 파트리더와 의견 충돌시 어떻게 해결해 나갈것인가?

예상답: 파트 리더의 스타일에 따라 다르다.

24) aa수료자로서 어떤 책임을 지닐 것인가?

예상답: SW 공학에 꾸준한 관심을 갖고, 배운것을 잊지 않고, 현업에 적용

25) SOA에 대해서 설명하세요.

예상답: 기능을 서비스 단위로 독립적으로 구성하고, 서비스간 통합을 통해 더 큰 비즈니스 프로세스를 구현한다.

장점 : 서비스의 느슨한 결합으로 플랫폼 및 기술 제한에 관계없이 기업이 사용 가능한 모든 서비스를 사용할 수 있는 유연성을 제공한다.

확장성과 재사용성에 좋다.

단점: 서비스 통합에 필요한 middleware의 overhead가 점점 커짐. 서비스가 많아질수록 다른 서비스와 상호작용이 늘어서 오버헤드가 발생해 퍼포먼스가 감소할 수있다. 또한 서비스가 많아지면 관리가 어렵다.

26) 마이크로서비스에 대해서 설명하세요.

예상답: Fine-grained SOA

마이크로 서비스 아키텍처란 기능별로 분리된 모듈을 의존성없이 교체할 수 있도록 만든 아키텍처 스타일입니다. 핵심은 각각의 기능모듈을 느슨하게 연결해서 독립적으로 개발 및 유지보수가 가능하도록 만드는 구조 즉 기능간 분리입니다.

장점 : 개발 테스트 및 유지보수에 드는 시간과 비용이 절감할 수 있다. 개별 모듈을 병렬로 개발할 수 있습니다.

단점: 네트웍을 통한 통신으로 인한 오버헤드가 발생합니다. 마이크로 서비스 하나의 오류가 전체시스템으로 전파됩니다.

27) 디자인 패턴의 종류(행위, 구조, 생성) ? 이것에 대해서 설명?

예상답:

생성(Creational) 패턴

객체 생성에 관련된 패턴

객체의 생성과 조합을 캡슐화해 특정 객체가 생성되거나 변경되어도 프로그램 구조에 영향을 크게 받지 않도록 유연성을 제공한다.

구조(Structural) 패턴

클래스나 객체의 합성에 관한 패턴

클래스나 객체를 조합해 더 큰 구조를 만드는 패턴

예를 들어 서로 다른 인터페이스를 지닌 2개의 객체를 묶어 단일 인터페이스를 제공하거나 객체들을 서로 묶어 새로운 기능을 제공하는 패턴이다.

행위(Behavioral)

클래스나 객체들이 상호작용하는 방법과 책임을 분산하는 방법 정의

객체나 클래스 사이의 알고리즘이나 책임 분배에 관련된 패턴

한 객체가 혼자 수행할 수 없는 작업을 여러 개의 객체로 어떻게 분배하는지, 또 그렇게 하면서도 객체 사이의 결합도를 최소화하는 것에 중점을 둔다.

https://gmlwjd9405.github.io/2018/07/06/design-pattern.html

28)객체를 빨리생성하는 패턴이 뭐죠?

프로토 타입 패턴

prototype pattern은 object 생성이 높은 비용으로 수 많은 요청을 하는 경우, 또는 비슷한 object를 지속적으로 생성해야 할 때 유용하게 사용할 수 있다. prototype pattern은 본래의 object로 부터 새로운 object를 만들어내며(서로 다른 인스턴스), 각 객체에 따라 데이터 수정이 가능한 메커니즘을 제공한다.

복잡한 객체 또는 생성에 알고리즘이 필요하거나 시간이 많이 걸리는 객체를 생성할 때 미리 생성된 객체를 복사해서 생성함으로써 빠르게 생성 가능함.

ISO9126 품질속성에 대해서 설명해 보세요

: 6 Main Factors and Sub-Factors for each  
 1. 기능성: 적절성, 정확성, 상호운용성, 보안성, 준수성  
 - 필요한 기능을 제공하는가?  
 2. 신뢰성: 성숙성, 결함수용성, 복구용이성, 준수성  
 - 믿을 만 한가?  
 3. 사용성: 이해용이성, 성습성, 운영성, 기호성, 준수성  
 - 쓰기 쉬운가?  
 4. 효율성: 시간효율성, 자원활용성, 준수성  
 - 사용되는 자원의 양에 따라 요구된 성능을 만족하는가?  
 5. 유지보수성: 분석성, 변경성, 안정성, 시험성, 준수성  
 - 고치기는 쉬운가?  
 6. 이식성: 환경적응성, 설치용이성, 치환성, 상호공존성, 준수성  
 - 다른 환경으로 쉽게 바꿀 수 있는가?

유이기사효신. 유이의 기사가 박효신이다? ㅋㅋ

================================================================================  
1. opening 멘트   
2. 열몇개 UML 중에서 concurrency를 볼 수 있는 다이어그램은 어떤것들이 있나?

-> behavioral diagram은 모두 확인 가능. sequence diagram, activity diagram, state machine diagram, timing diagram (<https://www.omg.org/ocup-2/documents/concurrency_in_uml_version_2.6.pdf>)

3. mediator 패턴과 facade 패턴의 차이

둘다 중재 역할을 수행하나, mediator는 객체간의 workflow를 관장하는 것이 목적이고, facade는 subsystem interface를 통합한 하나의 객체로 제공하여 편리성을 제공하는 목적입니다.

4. N개의 서버를 운영할 때 사용할 수 있는 architecture style?  
 -> dispatcher 대답했더니.....  
5. broker 도 사용할 수 있는데 dispatcher와 broker의 차이?

-> session 생성이후에도 계속 관여하면 broker, session 이후 직접 통신하면 dispatcher  
6. performance 관점에서 dispatcher와 broker 중 어떤 것이 유리한가?

-> overhead가 적으므로 dispatcher가 유리. availability 관점에서는 server instance의 상태를 지속 관리하므로 broker가 더 유리.  
7. 현업에서 무슨일 하고 '이 상황에서 이런 디자인패턴을 적용할 수 있었겠다' 하는 상황이 있었는지?  
 -> observer를 얘기했더니..한참후에 관련질문 들어옴..  
8. security 업무를 할 때 가장 중요한 view 가 무엇이라고 생각하는지?  
 -> 대답못하고 있었는데.. 아래처럼 view 설명해주심...  
9. view 의 종류가 funtional, information, ... 등이 있는데   
security 부서와 협업할 때 보통 '이 view 주세요' 하는 것이 있다. 무엇인지?  
 -> 도대체 뭐지 ㅠ  
10. 재설계에서 product line 강의에 몇시간정도 할애했는지?   
 -> 보통 2~3시간 강의했었다고 하셨음...  
11. product line 설명  
12. 우리 회사에 product line 컨셉이 적용될 수 있다고 보는지?  
13. 이런 컨셉이 적용되면 좋다는 것을 많은 사람들이 알고 있을텐데 왜 못하고 있다고 보는지?  
 -> DS에서 일부 적용하고 있는 곳도 있다고 코멘트해주셨음....  
14. 위와 연속적인 질문....기억안남..횡설수설 ㅠ  
15. observer 패턴 얘기했었는데 단점이 무엇인지?  
 -> subject의 state가 바뀔때마다 noti가 올것임. 특정 상태일때만 알면 될때는 overhead가 될 수 있음. 그래서 ChangeManager를 중간에 둘수도 있겠음.  
16. push 방식과 observer 의 차이? 각각 어떤 경우 사용할지?

-> observer를 push 방식, pull 방식 둘다 구현가능.   
17. closing 멘트  
================================================================================  
1. 소감  
2. Tizen Platform 하고 계신데, product line engineering 을 적용할 수 없겠냐  
 -> domain engineering 에 시간이 많이 걸려서 scope 정하는게 중요하고  
 적용할 수 있는 부분은 굉장히 많다고 생각한다.  
3. Layered Architecture 의 layer 는 어떻게 정해야 하나  
 -> NFR과 SRS 통해서 architecuture driver 를 도출하고 이를 바탕으로  
 skeleton architecture 를 선택하면서 layer 를 도출해야 할 것 같다  
4. HAL layer 의 아래와 윗 레이어에는 어떤 레이어가 들어가냐  
 -> 횡설수설 하다가, 아래 레이어는 각 device 에 대한  
 driver 그리고, interface 가 정의되고, 윗 layer 는 control layer 가 될 것 같다  
5. HAL layer 는 facade, mediator, adapter 중에 어떤 패턴으로 디자인하면 좋겠냐  
 -> mediator 는 각 component 간의 communication 중재라, control layer 에 쓰면 좋겠고  
 adapter 는 interface 의 compatibility를 맞추는 것이라, 맞지 않는 device 가 추가되면  
 차후에 추가하므로, 제 생각엔 일단 facade pattern 으로 디자인하는게 좋겠다  
6. Design pattern 을 성격에 따라 pairing 한다면 어떻게 하겠냐  
 -> decorator, composite//strategy, template method 등등 얘기함

7. flyweight pattern 에 대해 설명해 보라

-> 크기가 작은 객체가 여러 개 있을때, 공유를 통해서 효율적으로 지원하는 패턴임.

객체는 공통 상태를 가지고 있고, 부가적인 상태는 context parameter로 받아서 처리함.

8. proxy pattern 과 adapter pattern 에 대해 설명해 보라

-> real object의 접근을 decoupling 시키는 것이 목적이고, interface 불일치는 해결하는 것이 목적이다.  
9. platform 을 설계할 때 요구사항 측면에서 어떻게 single system 이랑 다를까요  
 -> 일단 stakesholder 가 developer 이고 여러 application으로 특성화 되어야 하므로  
 공통성과 가변성에 대한 고려가 다를 것 같다  
11. timing diagram과 state diagram 의 차이점은 그리고, state diagram 으로 timing diagram 을 대체할 수 있겠냐  
 -> timing diagram 은 객체들간의 상태 변화를 시간 제약으로 표현한 것이므로,  
 state diagram 으로는 대체가 어렵다  
12. 맺음말

[Interview 및 인증 통과 핵심 포인트]

- CEP과제는 열심히 하면 점수가 어느 정도 나오는 것 같습니다

- 문제에서 요구한 것은 모두 해결하셔야 하고 (몇 가지를 빼먹고 하신 분들도 있던데 좀 안타까웠습니다), 그 이후에 practical한 접근 (예를 들어, 서버 architecture라면 AWS를 접목)을 하시는 것이 나을 것 같습니다

- 인터뷰는 준비를 하고 임하시는 것이 좋겠습니다

? CEP과제에서 아주 좋은 점수를 받았는데, 인터뷰를 통해서 본인의 실력이 아니거나 architect로서의 기본기가 갖추어지지 않았다고 판단되는 경우가 수 차례 있었습니다

- 인터뷰에서의 핵심은 물어 본 것에 대한 답을 하는 것입니다

? 1기 interview에서 아주 많은 분들이 질문과 상관 없는 배경 설명을 하셨는데, 이는 architect의 자질 중 하나인 communication 능력에 대한 감점으로 간주가 됩니다

- 아마도 1기 interview에 나왔던 문제들은 이미 공유가 되었을 것으로 예상하는 데, 미리 구해서 보시고 아래의 문제 유형을 잘 생각해서 준비를 해주시는 것을 추천 드립니다

[Interview 문제 유형]

- 인터뷰 처음 또는 마지막에 “개인적인 소감”, “내가 architect로서 왜 자질이 있는가”, “내가 양성과정 중 잘 한 것은 무엇인가”, “양성 과정 중 감명 깊었던 내용은 무엇인가”, “Architect가 된 후에 부서에서 어떤 역할을 할 것인가” 등의 질문이 나옵니다.

? 점수를 주기 위한 질문인데, 여기서 동문서답을 해서 점수를 대폭 깎아 먹는 분들이 있었습니다. 예를 들어, architect의 자질이 있는지에 대해서 질문을 했는데, “이번 양성 과정은 좀 어려웠고 CEP은 어쩌고 저쩌고”와 같이 대답하는 식입니다. 이런 거 절대 안됩니다.

? 문제에 대한 답만 하시는 것이 포인트입니다

? 예상 질문 몇 개를 뽑아서 대비하고 가시는 것이 좋겠습니다

- CEP 문제의 핵심 포인트에 대한 질문

? Non-functional requirement를 어떻게 해결 했는지를 주로 물어 보게 됩니다

? 본인이 CEP과제를 충실히 했으면 어렵지 않게 대답할 수 있는 문제인데, 가끔 헤매시는 분들이 있었습니다

? Architecture style을 결정할 때, 고민했던 다른 대안들을 물어 보기도 합니다

- Design pattern에 대한 질문은 거의 받는 다고 보시면 됩니다

? 쉬운 질문으로는 design pattern중에 본인이 좋아하는 것을 하나 설명해 보세요

? Design pattern을 설명해보세요: 생성패턴, 행동패턴, 구조패턴이 있고, 생성패턴에는 어쩌고 저쩌고, 행동패턴에는 어쩌고 저쩌고, 구조패턴에는 어쩌고 저쩌고 정도로 하시면 될 것 같습니다

? 문제 상황을 주고 어떤 패턴을 사용하는 것이 좋을지 물어 보는 질문들: Portability를 높이기 위해서 사용할 수 있는 design pattern은?

-> 브릿지 패턴. 기능과 구현을 분리할 수 있으니까, 플랫폼 종속적인 코드는 구현쪽에만 개발하는 것으로 대응 가능.

? Decoration pattern과 class 계승(상속)의 비교

-> 두개다 기능을 확장하는 방법 이지만 상속은 동적인 확장이 어렵다. 데코레이터는 런타임에 기능이 추가 가능하다(책임을 선택적으로 덧붙일 수 있다.)

? Facade와 mediator를 잘 구분하실 수 있어야 함

-> 목적이 다름. 편리성과 workflow 관장.

? 수업시간에 visitor pattern을 열심히 강의 해서 인지, 피면접자분들이 visitor pattern으로 문제를 해결하려는 경향이 있었는데 좋지 않은 예입니다

- 문제 상황이나 기존의 서비스 들을 예로 들면서 architecture style를 결정해보라는 질문

? 정답이 있는 것은 아니지만, 본인이 선택한 style이 왜 선택되었는지 논리적으로 설명해야 함

? Facebook과 같은 SNS에는 어떤 style이 적합할까요?

- Tier와 Layer의 구분

? Computing instance가 나누어지면 tier

? 하나의 computing instance 내에서 module로 구분되면 layer

- Quality attribute scenario의 6개 구성 요소 설명

- Stakeholder간의 conflict이 발생했을 때 어떻게 해결할 것인가

? 정답이 있는 것은 아니니, 요구공학에서 배운 내용을 토대로 잘 설명을 하면 될 듯

- Quality attribute간의 trade-off 상황이 발생했을 때, 사용하는 tactic에 대한 설명

? Security를 높일 수 있는 tactic은?

? 방금 대답한 tactic은 다른 quality attribute (예를 들어, performance)에 어떤 영향을 주는가?

- Cohension과 coupling에 대한 설명과 이 둘의 관계

- ATAM에 대한 설명

- Usability를 높이기 위한 architecture적인 접근

- Architecture style에 대한 질문

? MVC, shared repository, publish-subscribe, 등등 중에서 하나를 설명해 보세요

? 해당 style을 구현하는 데 유용하게 사용될 수 있는 design pattern은?

- OOAD적인 접근과 Quality-driven design이 conflict이 발생할 때, 어떻게 해결할 것인가?

- SoC (separation of concerns)에 대한 개념을 확실히!

제가 받은 Interview 질문 내용 공유드립니다.   
  
1 . 데코레이터 설명해봐라

-> 객체의 결합을 통해 기능을 동적으로 확장할 수 있게 하는 패턴입니다. 기능 확장이 필요할 때 상속대신 쓸 수 있는 유연한 대안이 될수 있다.

Component를 상속받은 Decorator 클래스가 Component를 다시 composite함으로써 기능을 계속해서 덧붙여 나갈수 있다.

객체의 responsibility나 behavior가 동적으로 변할때 사용한다.

2 . 데코레이터 말고 그냥 상속쓰면 안되냐

-> 동적으로 기능 붙이기가 안된다.

3 . 스트레티지와 템플릿차이

-> strategy는 알고리즘을 캡슐화 해서 통째로 교체 가능하고 템플릿은 알고리즘 구조는 동일하고 sub step만 sub class에서 재정의한다.

4 . 자바의 try catch는 무슨 패턴인가  
- Chiain of Responsibility 패턴 임.  
- nested catch 구조와 forwarding을 바탕으로  
5 . 서비스를 다운로더블 하게 하려면 어떤 아키텍처 적용하냐  
6 . 유즈케이스 클래스 시퀀스 액티비티 스테이트머신 을 두고 이것들의 traceability는 어떻게 보나  
7 . 아키텍쳐 드라이버를 도출하는 단계를 설명하라

stakeholder과 concern사이의 association을 표현하고 그 중에서 우선순위가 높은 concern을 선택 🡪 선택된 concern을 더 구체화시키고 refine해서 architecture driver로 선정  
8 . Repository style과 black board style의 공통점과 차이점을 설명하고 저장되는 내용의 차이도 설명하라  
9 . 커널단에서는 C언어기반 구조적 프로그래밍인데 어떻게 아키텍처를 적용할거냐  
10 . Dependency와 association에 대해 비교해봐라 (유나님, 1번 문제)

- 객체간의 관계를 나타낸다. Dependency 일시적으로 호출. Association에서 호출하는 객체에서 가지고 있어서 지속적으로 호출.  
  
  
1. strategy / Template method 비교 설명  
2. strategy를 쓰는데 parameter가 다르다.. super class와 subclass의 function이 같아야 하는데 => 이럴 때 어떻게 할래?  
3. 질문은 기억이 안나는데 상황 주고 어떤 패턴 쓸래? => observer 패턴 이 답이였고 pushing 과 observer 패턴이 어떻게 다르냐?  
4. UI에서 어쩌고 저쩌고 상황을 주고 어떤 패턴 쓸래? => command 패턴을 설명하는 상황이였으나 계속 state 패턴이냐라고 대답했던 기억이 나네요.   
(삼성 사람이 물어 본 질문인데 질문 자체가 좀 이해가 안 갔음. 그래서 옆에서 김수동교수님이 질문에 대한 부연 설명을 해주구 막 그랬는데 command 패턴 이름이 안 떠 오름)  
5. state diagram 을 잘 그려두면 state pattern으로 옮길 수 있는데 어떻게 matching되냐. (그전에 state패턴 적용 상황 주고 답하란 얘기도 했던거 같아요)  
6. activity diagram과 sequence diagram과의 차이에 대해서?  
=>답을 못 하니 김수동 교수님이 설명 해줌  
7. android와 일반 java 프로그램과의 차이점이 머냐?ㅋㅋ  
8. IPC 의 전통 방식에 대해서 알고 있는 것 설명 해주라.  
9. performance에 대한 tactic에 대해 알고 있는 것 얘기 해주라  
  
  
1. 자기가 한 과제중에 자랑할 수 있는 것 하나는?  
2. 배운 패턴중에 기억에 남는것은?  
3. 패턴의 종류가 뭐뭐 있고 각 종류별로 대표적인 패턴은?  
4. 아키텍으로서 팀원들과 의견이 다르면?  
5. 배운거 어떻게 써 먹을래?  
6. Product line에 대해서 설명하고 application과 domain 관점에서 어찌 다르냐?  
7. 확장성과 수정 가능성에 대해서 설명해봐라  
8. 종합과제에 어떤 스타일 적용했냐  
9. 아키텍처 스타일과 디자인 패턴의 차이점은?

-> 아키텍처는 비즈니스 품질속성을 반영한 제약사항의 집합이고, 디자인 패턴은 개발관점에서 설계에 대한 문제 해결방식이다.  
10. 종합 과제에서 수정가능성을 고려했는가?

11. 종합 과제에서 본인의 설계가 타당하다는 것을 어떻게 확신할 수 있는가?  
12. 본인의 설계와 팀원의 설계가 다를땐 어떻게 할래?  
   
   
1. Architecture Design / Architecture Style / Architecture Pattern / Design Pattern 의 차이점을 설명하시오.  
2. 본인의 설계가 타당함을 검증해보시오.  
3. 품질 속성 6가지 항목에 대해 설명하시오.  
4. 패턴과 Framework의 공통점/차이점  
5. 안드로이드에서 샌드박스간 통신 방법  
6. Shared Repository Architecture Style의 장/단점 및 사례  
7. Product line에 대해 설명하고 application과 domain 관점에서 차이점  
8. 종합 과제에 적용한 스타일?  
9. 아키텍처 드라이버 도출 단계  
10. SRP와 SOC 차이점  
11. 품질 속성과 그에 따른 택틱들은?  
12. differ binding 이란?  
13. proxy 패턴 종류와 설명  
14. 오버로딩 / 오버라이딩 / 동적 바인딩 설명  
15. aggregation과 dependecy 차이점 설명  
  
  
  
[면접 시작과 끝]  
- CEP 또는 교육과정을 통해 얻은 자신의 강점  
- 시작할 때 과정에 대해 느낀점 이야기 해보라  
- 가장 도움이 많이 되었던 과정은?  
- 끝날때 하고 싶은 말 해보라 (마지막. 2분간 Closing 멘트해주세요)  
  
[CEP 관련]  
- CEP 과제에서 선택한 아키텍쳐 스타일에 대해서 이야기 하고 왜 선택 하였는지?  
- NFR 만족시키기 위해 사용한 택틱들에 대해 왜 선택하였는지? (+ 어떻게 적용하였는가?)  
- CEP 에서 인터페이스를 정의할때 facade 패턴을 사용하였는데 장점은 무엇이고 단점은 무엇인가 ?  
  
[코스웍]  
- 가장 인상깊은 Design Pattern은 무엇인가?

: 스트레이터지 패턴이였습니다.   
- 그 밖에 알고 있는 Design Pattern은 무엇인가 ?  
- Adapter 패턴은 언제 쓰이나?  
- balanced mvc 란?

- MVC 패턴에서 공통된 부분과 변경가능한 부분은 무엇인가? (+단점)

공통 : 모델

ui변경에 따른 뷰

컨트롤러 :

단점 : 개발의 복잡성 증가

- proxy pattern 이란?  
- State 패턴의 장단점은 무엇인가?  
- 행위패턴에 무엇이 있나? 그중 비지터 패턴은 어떨때 쓰는가?  
- 수업에서 들었던 패턴들의 특징 공통점과 차이점  
  
- product line 이란?  
- platform과 product line 과의 공통점과 차이점  
  
- 플랫폼에서 중요한 부분은 무엇이며, 어떻게 표현이 되었는지?  
- Platform 일때 architecture 에 고려해야 할점은?  
- Platform 일때 요구사항 공학에 적용되어야될 점은?  
  
- Android platform 에 적용된 디자인패턴 아키텍처  
- C 도메인에서 state 관리가 중요한데 어떤패턴 적용해야 좋은가?  
  
- Publish-Subscriber Style과 Publish-Broker-Subscriber Style의 장단점은 무엇인가?  
- observer패턴은 Publish-Subscriber Style인가 ? Publish-Broker-Subscriber Style인가?  
- 여러 스타일을 적용할 때, 발생할 수 있는 Conflits은 어떻게 해결할 수 있나?  
  
- Open close principle  
- OO 에서 encapsulation, information hiding 등등 설명해보기  
- Dynamic binding 에 설명해 주세요.  
- abstract 와 interface 와의 공통점 차이점  
- Association에서 Aggrecation과 Composition의 차이점이 무엇인가?  
  
- 평가방법들과 atam과 saam 의 차이점  
- Atam에 대해 설명해보시오  
   
[ETC]  
- Architecture description 이해를 못하거나 할때 어떻게 해야하나? (-> 이건 아마도 상대방이 이해를 못할때 설득하는 방법을 말하는 것 같습니다.)  
- PL, TL, Architect 의 차이점은?  
- 아키텍쳐를 상사에게 설득방법  
  
  
1. Architecture Design, Architecture Style, Architecture Pattern, Design Pattern ?  
2. 본인의 설계에서 확장 가능성, 변경성이 적용된 부분?  
3. 본인의 설계가 타당한지 어떻게 확신하는가?  
4. 품질 속성 시나리오에 사용되는 6가지 항목  
5. CEP에서 아쉬웠던 점  
6. Functional View, Information View, Concurrency View, Development View 등 설명  
7. Abstract와 Interface의 차이  
8. 행위 패턴의 종류와 설명  
9. Override 와 동적 바인딩  
10. OCP ?  
11. State 패턴 설명  
12. Share Repository Architecture 설명 및 장/단점, 사례, 단점은 어떻게 극복?

대용량 데이터 처리, 스키마의 변경에 취약  
13. Façade 패턴  
14. 과제 중 자랑할 만한 것  
15. 배운 거 어떻게 써먹을 건지  
16. Product line 과 application과 domain 관점에서 차이  
17. 확장성과 수정 가능성  
18. CEP에 적용한 스타일  
19. Architecture Driver 도출 단계  
20. Repository 스타일과 Blackboard 스타일 비교, 차이, 정보의 차이  
21. 커널단이 C언어 기반일 때 어떻게 Architecture를 적용할 것인가  
22. Dependecy와 Association 차이  
23. 코스웍에서 가장 잘 아는 부분, 현업에서 잘하는 부분  
24. 인터페이스에서 가변성이 예측될 때 어떻게 만들어야 하는지  
25. Provided, required 인터페이스 측면에서 설명  
26. Security 택틱을 나열해보고 설명  
27. Deffer binding time은 어떤 품질 속성이며 관련된 디자인 패턴은?  
28. Proxy 패턴 종류?  
29. Traceability 유지를 위해 무엇을 하면 되는지  
30. strategy / Template method 패턴 비교 설명  
31. strategy를 쓰는데 parameter가 다르다.. super class와 subclass의 function이 같아야 하는데 파라메터가 다른 경우  
32. observer 패턴과 publish-subscriber 스타일의 차이  
33. Command 패턴 설명  
34. State machine diagram과 state 패턴과의 matching 관계  
35. Activity diagram과 sequence diagram 차이  
36. Android와 Java 프로그래밍 차이  
37. IPC 전통 방식 설명  
38. 성능에 대한 tactic ?  
39. Decorator 패턴 설명  
40. 자바의 try catch는 어떤 패턴인가  
41. 서비스를 downloadable 하게 하려면 어떻게(?)

: 서비스 로케이션을 분명히 알고 있을 때: 쉐어드 리파지토리

: 내가 뭘 받아야 할지 모르겠다면: 블랙보드

: 플러그인 다운로드: 마이크로 커널

: 이미 다운로드 받은 서비스의 업데이트 Pub-sub 패턴

**카톡 깜짝 퀴즈**

|  |  |
| --- | --- |
| 문제 | 예상답 |
| visitor 패턴은 캡슐화 위반 | 캡슐화 위반입니다. |
|  | . |
| COR과 decorator는 responsibility를 덧붙이는 구조로 유사한것 같은데 차이점은 무엇인가? | cor은 연결리스트와 같이 수평한 구조로 책임을 덧붙이며, decorator는 계층적으로 책임을 덧붙입니다. |
| state패턴과 strategy패턴은 클래스다이어그램이 동일한 데 어떤 차이점이 있나요? | state패턴은 state에 종속적인 동작을 할 때 사용하고, strategy는 알고리즘이 다양하고 상호교환이 필요할 때 사용합니다. |
| Class Diagram과 State Machine Diagram의 일관성이 있으려면 어떤 부분이 같아야 할까요? | state머신의 event가 class diagram의 public method로 표현되어야 한다.  하나의 클래스가 하나의 state 머신 다이어그램으로 표시되어야 한다.  action은 private method 표현된다.  state diagram은 언제 사용하나요?  현재 스테이트에 따라서 기능이 바뀔 때 사용된다. |
| 시퀀스 다이어그램과 클래스다이어그램의 consistency는 어떻게 확인할 수 있나요? | 시퀀스 다이어그램의 메세지 내용들이 class diagram의 메소드와 파라미터로 정의되어야 한다. |
| Override? | 부모 클래스의 메소드와 동일한 메소드를 재정의 함으로써 기능을 수정한다. |
| information hiding | 정보 은닉에는 객체의 필드 및 메소드를 은닉하는 캡슐화가 있고, 인터페이스 혹은 Abstract class로 구현 자체를 은닉할수 있다. |
| 정보 은닉의 장점 | ocp?  코드의 확장성과 유지보수성이 좋아진다.  정보를 노출하면 접근이 다양해지고 커플링이 높아지고 ripple effect가 문제가 될수 있다. 따라서 정보를 숨길수록 교체나 변경이 쉬워져서 유지보수에 용이하다.  또한 인터페이스를 은닉하면 동일한 인터페이스를 구현한 클래스들을 동적으로 서로 교체 가능하다.  setter부분에서 값의 validation체크도 가능하다. |
| 추상화 | * 공통의 속성이나 기능을 묶어 이름을 붙이는 것 * 객체 지향적 관점에서 클래스를 정의하는 것을 바로 추상화라고 정의 내릴 수 있겠다. |
| lsp | 자식 클래스는 부모 클래스에서 가능한 behavior(행위)를 모두 수행할수 있어야 한다. |
| dip | 구체적인 클래스가 아닌 추상화된 클래스에 의존하라.  상위 클래스는 하위클래스에 디펜던시를 갖지 마라.  자신보다 변하기 쉬운 것에 의존 하지 마라. |
| 템플릿 메소드 패턴 | 알고리즘의 구조는 고정되어있고 알고리즘의 일부를 서브클래스가 재정의한다. |

GRASP

* Information expert: 데이터를 가지고 있는 애에게 일을 시켜라
* Creator
* Controller
* Low Coupling
* High Cohesion
* Don’t talk to Stranger