①객체지향 개요

1. 소프트웨어 패러다임의 변화

패러다임 : 바라보는 시각(관점, 뷰) → S/W 개발 패러다임 : S/W 개발에 대한 시각

S/W 개발 → ①절차 중심 개발, ②객체지향 개발

①절차 중심 개발(소공;구조적 개발 방식) : 기능 중심 또는 절차 중심
→ 전역화로 인한 재사용, 유지보수 한계 문제점
→ 데이터와 함수가 서로 분산되어 있음

②객체지향 개발 방식
→ 캡슐화, 정보은닉을 통해 국지화 → 모듈성↑ → 개발 비용, 품질, 재사용, 유지보수 향상
→ 속성과 메서드를 특정 목적*에 따라 독립적인 모듈 단위(객체)로 그룹핑 → 객체 단위로 동작

※ 전역화 : 변수를 프로그램 전체에서 사용할 수 있도록 설정하는 것(여러 함수들이 데이터를 공유)
※ 국지화 : 변수를 특정 함수나 절차 내부에서만 사용할 수 있도록 범위를 제한하는 것

2. 객체지향 모델링

모델링

모델을 만드는 작업, 현실 세계를 단순화시켜 표현하는 방법 → 특정 목적에 따라 공통된 요소만 남긴 것
→ 분석/설계 단계에서 이루어지는 방법으로, 결과물은 모델 → 다이어그램 형태, 명세서, 프로토타입 등
→ 목적*에 맞게 단순화(추상화) = 왜곡(공학적 관점, 정보를 깎아내는 것), 예: 목차(전체를 커버하면서 단순화)
→ 모델링 : 기준이 존재, 개발방법론에도 영향을 줌 → ①클래스 관점(속성부터 정의) ②행동 관점(메서드부터)

모델링 목적 → 3가지

개발할 시스템의 범위/구조/기능을 쉽게 이해하고, 고품질의 신뢰성 있는 시스템 구축 위함
→ S/W의 규모, 복잡도가 기하급수적으로 증가 → 개발 과정에서 도메인에 대한 깊이 있는 이해가 중요해짐
→ 일종의 시뮬레이션*(예측을 위함) → 다시 설계하거나 수정하려면 비용이 많이 들기 때문

①현실의 도메인을 잘 반영할 수 있는 모델 선택
→ 모델이 실제 세계의 사물이나 개념을 정확하게 표현해야 한다는 의미(객체, 속성, 행위, 관계 등)

②다양한 각도에서 표현할 수 있는 모델 생성 → 하나의 모델로는 모든 구조/행위/상태/작업 흐름 표현 불가능

③개발할 시스템에 적합한 모델 선택

기대효과★ → ①가시화, ②명세화, ③개발할 시스템 구축에 대한 기초 마련

①가시화 : 어떤 기능/내부 구조로 구성돼 있는지 다양한 형태의 모델을 통해 개발할 시스템의 형태를 가시화
→ 고객-개발자 간 의사소통 원활 → 요구사항에 부합한 시스템 개발 가능해짐

②명세화 : 구조, 기능, 워크플로우, 상태 등 다양한 명세화 가능 → 개발할 시스템의 범위를 정확히 이해 가능

③개발할 시스템 구축에 대한 기초를 마련 → 모델은 구현을 위한 기반이 됨(입력물)

구조적 모델링 기법

산출물들 간 연계성 반영이 어려움, 구현에 필요한 입력물 기능 하지 못함

객체지향 모델링 기법

다양한 기법이 존재하여 사용하기 어려웠음 → 통합 필요성 대두 → UML 등장(부치, 야콥슨, 럼바우)
→ UML은 기법이 아닌, 언어임 → 기법은 프로젝트마다 다양하기 때문에 기법으로 만들면 유연성이 떨어짐
→ UML은 언어이기도 하지만, 하나의 노테이션(표기체) → 모양도 언어임(객체지향을 표현하기 위함)
→ UML은 설계를 표현하기 위한 수단(①기호 ; 가시성이나 화살표 + ②구문 ; 박스나 관계)이기도 하지만
→ 본질적으로는 분석을 위한 것에 더 가까움 → 문제를 명확하게 설명하기 위해 만든 언어
→ UML이 주된 목적이 아니라, 클래스를 어떻게 시각적으로 표현할까에서 나온 것(관계, 구조, 상호작용 등)

3. 객체지향 언어

클래스 → 속성/프로퍼티, 행위, 접근 제어자 → 메시지를 통해

객체 → 멤버필드, 메서드, 접근 제어자 → new 연산자 + 생성자로 객체 생성 → 객체 단위로 동작(상호작용)

상속(extends) → ①일반 클래스 간의 상속 관계

②추상 클래스와 일반 클래스 간의 상속 관계

③인터페이스 간의 상속 관계

→ 상속을 통해 메서드 오버라이딩 메커니즘 실현 가능

→ 부모 클래스의 속성과 메서드를 자식 클래스가 물려받는 개념 → 재사용성↑, 계층적인 구조

→ 부모 클래스는 자식 클래스의 존재(구현의 세부사항)을 알지 못 한다. : 결합도↓ → 유연성, 재사용성↑

추상클래스(abstract)

인터페이스(interface) → public 메서드들의 집합 → 클래스를 통해서만 구현 가능(implements)

컴포넌트 → 기능적으로 구분된 독립적인 객체들의 집합으로, 특정 기능이나 역할을 수행하는 모듈화된 단위

→ 독립성, 재사용성, 캡슐화

→ 객체지향은 의인화*해서 생각하면 이해하기 쉬움

→ 독립성, 재사용성, 캡슐화, 유지보수성(변경, 확장에 용이), 모듈성, 정보은닉, 추상화 → 이거 돌고 도는 듯

②클래스

1. 클래스의 개념과 특성

관점에 따른 클래스 개념 → ①분석 설계 입장, ②프로그래머 입장

①분석 설계 입장 : 추상적 개념으로 표현

②프로그래머 입장 : 구체적인 구현 관점

→ 클래스가 객체 지향 설계의 기본 단위로 객체를 정의, 객체가 수행할 수 있는 행동과 상태를 포함한다 뜻

크게 2가지 관점 → ①classism 관점(부치, 럼버) ②behaviorism 관점(야콥슨, 애자일) → 캡슐화와 연관

→ 구조와 관계 관점 → 행동과 상호작용 관점

목적에 따라 클래스를 설계할 때, 속성을 먼저 생각하고 할건지, 행동을 먼저 생각할건지 판단하면 됨

예) ①학생이면 학번, 이름 등이 있어야지부터, ②수강신청시스템이면 수강신청등록/취소 등이 있어야지부터

클래스는 명세 장치이다.

여러 유사 객체들의 공통적 속성이나 행위를 기술하는 명세 장치 → Object Type is Category of objects

※ 명세 : 규격을 명확히 하는 설계도, 역할/관계/행동 등을 정의하거나 설명한 것

→ 예 : 고객 객체들이 회원가입 할 때, 항목의 상태(값)은 다르지만 회원가입의 항목은 같은 것처럼

클래스는 개념, 부류임 → 객체들을 한 범주로 그룹핑(공통적* 요소만 남겨서)

예 : 시스템 모델링 수강생(클래스) ↔ 각각 수강생들(인스턴스) : 주로 객체로 표현

인스턴스와 객체는 명확하게 다른 말임 → 메타 클래스의 경우 인스턴스가 클래스인 경우도 있음

→ 객체는 메모리에 실체화된 사례

클래스 특성 → ①고유한 이름, ②속성, ③행위

①고유한 이름

클래스는 다른 클래스들과 구별되기 위한 고유한 이름을 갖는다. → 도메인이 다르면 같은 이름 사용 가능

만약 특정 도메인에서 클래스 추출 시 이름이 달라도, 의미가 동일한 클래스가 존재하면 안됨

→ 해당 도메인에서 많이 사용하는 용어를 채택하는 것이 중요 ※ 도메인 : 해결하려는 문제의 영역

②속성을 지닌다. (정보 저장소 역할)

명세 역할을 하기 때문에 속성을 내포 → 객체는 상태를 갖는다와 연관성을 가짐(∵ 속성 X → 상태도 X)

③행위를 가진다.

잘 정의된 행위를 가져야 한다. → 예 : 회원가입 하기, 회원정보 수정하기, 회원 탈퇴하기 등

클래스* → 이름(기본) + 속성, 프로퍼티(속성) + 오퍼레이션(행위) → 클래스는 설계 관점

클래스와 객체 → 3가지 → 예 : 회원가입을 통해 1명의 고객이 생성

①클래스는 틀이고 객체는 실사례이다. → 실사례의 예가 객체인 것 (객체와 인스턴스가 완전히 동일한 것은 X)

→ 예: 탬플릿 자 → 우유 팩(개념)이 뭐임? → 서울 우유 팩을 보여주는 것(실체)

②클래스는 상태가 없고, 객체는 있다. (클래스는 선언적이다.)

→ 클래스는 속성에 대한 선언만 내포, 객체는 선언된 각 속성들에 대해 상태(값)을 지닌다.

→ 예외 : 정적변수 (여러 객체들이 공통적으로 공유해야 할 속성의 경우)

③객체와 클래스는 동일한 속성 수와 메서드 수를 갖는다.

2. UML에서의 클래스 모델링

UML에서 클래스는 시스템의 구성 요소를 나타내는 표기 → 박스 자체가 구문임(정해진 규칙)

→ 클래스 표기법을 이용하여 클래스 다이어그램*을 작성, 클래스 다이어그램은 시스템의 정적인 구조를 설계

→ 3개의 구역으로 나뉜 박스를 가지고 있음, 상세 설계 중심이냐, 관계 중심이냐에 따라 클래스명만 사용 가능

→ 다이어그램은 다양함 → 유스케이스(기초) + 클래스 다이어그램 + 상태 다이어그램 + 컴포넌트/배치/패키지

①속성 : 가시성, 속성명, 데이터 타입, (초기값) → 가시성은 정보은닉 → 디폴트는 package, 기본값은 priavte

→ -price : int (= 0) → 태만적인 것임(해줘야 하는건데 안함)

②행위 : 가시성, 행위명(매개변수), 데이터 타입(없으면 표기X) → 유일할 필요 없음(오버로딩), 기본값은 public

→ +add(id:String, pw:String):int (①Intra ②같은 메서드명 ③매개변수 개수 ④순서 ⑤타입 순으로 비교)

→ 오버로딩 : 인트라 클래스, 같은 이름의 메서드를 여러 개 정의할 수 있는 기능을 의미 → 가독성, 재사용성↑
→ 클래스 다이어그램을 자바 코드로 구현할 때 → 일반적으로는 메서드의 오퍼레이션(시그니처)만 작성
이는 클래스 다이어그램의 목적이 설계와 구조를 나타내는 것이기 때문

클래스들 간의 관계 표기법 → ①연관 관계, ②포함 관계, ③상속 관계
①연관 ②포함(Aggregation, 부분-전체) ③상속(Inheritance) 관계 표기 이용
→ 실선(방향성), 연관명, 다중성* → 관계 따라 구분하여 구현
→ 연관 관계를 맺는 클래스 간에 얼마나 많은 인스턴스가 관계를 형성할 수 있는지
①연관 관계 : 클래스들 간 요구하는 행위를 사용하는 관계
→ 호출하는 클래스의 메서드에서 연관관계를 갖는 클래스의 메서드를 호출하는 경우(p.32) ↔ 속성이면 포함
①연관 : 클래스 간의 일반적인 관계, ②포함 : 전체와 부분의 관계, ③상속 : 클래스 간의 계층 구조를 정의

3. 자바 클래스 구현
①속성 : 가시성, class 예약어, 클래스명, 중괄호(속성, 행위 구현) → 클래스 다이어그램에선 오퍼레이션만 사용
②행위 : 가시성, 리턴 타입, 메서드명, 매개변수, 중괄호 → 반환 타입 없으면 void 명시 ↔ UML과 차이점
→ 객체지향 open/hide 정신에서 비롯
안 쓰면 무조건 안 써줘야, 해줘야 하면 무조건 해줘야

*의존 관계 코딩하는 법 → 한 객체가 다른 객체의 기능을 사용할 때 성립(동적 관계)
→ ①멤버로 갖는다(나의 일부로), ②안에서 객체를 생성(내부), ③파라미터로 외부에서 받아
≪ ≫ → 의미에 따라 모양을 다 달리 해주면 외우기 힘드니 스테레오 타입 사용, 예 : ≪interface≫로 표현

※ 요구사항↔클래스 추출

기술서	→ **추출**
명사/형용사	→ 객체
동사	→ 행위

1. 요구사항으로부터 명사나 명사구 추출
2. 클래스로 적당하지 않은 명사 삭제 → ①중복 클래스, ②관계없는 클래스, ③불확실한 클래스
3. 클래스의 속성이나 메서드는 클래스가 될 수 없음
4. 요구사항에 정의되어 있지 않지만 필요한 클래스 추가
5. 최종 추출 클래스

※ 추출한 것 중에 필요한 것들 선택해서 쓰면 됨

```
// 계좌 추가 메서드 (연관관계)
    public void addAccount(Account account) {
        // 이 메서드는 실제 계좌를 저장하지 않음
        // 대신, 해당 계좌를 사용할 수 있는 로직을 정의
    }
```

③객체

1. 객체의 개념과 특성

(다양한 각도에서의) 객체의 개념* → 3가지 → 만질 수 있냐 없냐의 차이

①객체지향 개념 및 설계, 분석 단계 : 객체는 물리적 사물(구상) 또는 논리적 개념(추상)이다.
②구현 관점 : 객체는 모듈이다. → 캡슐화의 단위(서로 관련된 멤버필드+메서드)
③추상화 단위 : 캡슐화와 정보은닉의 도구이다. → 모듈화, 추상화↑ → 재사용성, 유지보수↑
→ 객체지향 시스템은 객체 1개만 있는 것은 인정하지 않음(절차지향이나 다를 바 없음)
최소 객체가 2개 이상 되어야 하고, 관계가 무조건 있어야 함
클래스와 객체는 닭이 먼저, 달걀이 먼저 문제가 아님 → 객체들이 있기 때문에 클래스를 만든거지(추상적)
객체는 원래 대상/타겟이라는 뜻 → 강의) 교수자, 수강자 하나하나가 객체 → 식별이 필요(식별은 구현 관점)
문제 해결 과정으로 클래스로부터 객체를 만드는거지, 사실 객체가 먼저 설명되는 것이 맞음
①객체는 행위부터 생각함(행동 주의) ↔ ②클래스는 속성부터 생각함(클래스 주의) → 관점이 다른 것임
→① 객체가 필요한 이유는 행위를 하는 것을 필요로 하니까 만들잖아 → 행위부터 생각
→② 반대로 실체가 없으니 개념을 알아가는 것이 중요하겠지 → 자연스럽게 속성부터 생각(클래스부터 생각)
→ 객체는 행위 관점에 가까움 → 특정 행동을 수행할 수 있는 독립적인 단위로 간주(속성을 가진 데덩보단)

객체의 특성(≒ 클래스의 특성) → ①유일한 식별자, ②상태, ③잘 정의된 메서드

①유일한 식별자 : 서로 다른 객체와 구별하기 위한 구별자 → 객체 참조값을 통해 구별
②상태* : 속성은 같아도 상태는 다를 수 있다. → (객체의 생명주기 동안) 상태는 또한 변경이 가능
③잘 정의된 메서드 : 객체 내의 정의된 멤버필드를 처리하는 메서드 → 서로 관련성이 없는 것들 캡슐화하면 X
→ 행위의 관점에서 객체를 파악/만든다가 맞음(O) → 객체는 행위가 중심이다(X)
→ 예) 학생을 만들어라 → 행위를 생각해보자 → 공부한다, 시험친다.. ↔ 클래스 중심

객체의 생명주기 → ①생성, ②사용, ③소멸

클래스지향이 아닌 객체지향인 이유 → 실행의 주체 단위는 객체이기 때문(메모리 할당, 메시지 주고 받는 것도)
→ 클래스는 객체의 기반이 되는 틀이기 때문에, 클래스 위주로 먼저 분석/설계/구현하는 것
→ 객체지향은 객체를 프로그래밍하는 것이 아니라, 클래스를 주로 프로그래밍하는 것임 ≪create≫
→ 클래스는 객체의 속성과 행동을 정의하는 설계도 → 클래스를 기반으로 객체를 생성
①생성 : 메모리의 특정 주소에 공간을 확보하여 존재 → new 연산자+생성자 → 객체 선언, 인스턴스화, 초기화
→ 실제 메모리 확보(힙 영역)
→ 객체 생성에는 ①디폴트 생성자(노파라미터) 방식과 ②파라미터로 초기화하는 방식 → 둘다 많이 사용
②사용 : 상태 변경, 메시지 전송을 통해 정보 교환(객체 간 상호 작용) → 객체명을 통해 사용(멤버필드, 메서드)
③소멸 : 더 이상 사용할 필요가 없을 때 메모리로부터 제거 → 가비지 컬렉터 ≪destory≫

2. UML에서의 객체 모델링(설계 관점)

객체표기법 → ①객체명, ②속성

속성에 대한 상태를 정의, 2개의 구역으로 된 박스(표기법) → 객체명 + 속성값(행위는 정의 X)
①객체명 : 객체명 : 클래스명 OR : 클래스명 → 객체명이 정해지지 않은 경우엔 생략 가능 → :(클론)*
②속성값 : 속성명=속성값 OR (속성명:데이터타입)=속성값 → 속성은 =가 중심

객체들 간의 관계 표기법 → 클래스와 유사

클래스 연관관계에 대한 실제 사례 → 클래스와 유사, But 상태를 갖는 각 객체들 간에 독립적으로 방향성 설계
다중성 → 한 클래스가 다른 클래스를 얼마나 사용하고 있는지를 나타내는 것 p.59

연관 관계(Association)에 대한 구현 = 메시지 전송

→ 한 객체가 다른 객체와 상호작용하는 것을 의미(일반적인 관계) → 다른 객체의 특정 행위를 호출
①의존 관계나 인터페이스 관계인지 확인 → 동적 관계, 일시적

→ 두 개체 간의 의존 관계 또는 인터페이스 관계가 성립하는지를 먼저 판단 → 객체 간의 연결 유무 차이임
②없으면 나머지는 전부 연관 관계 → 정적 관계, 지속적
→ 만약 D나 I 관계가 아니라면, 기본적으로 연관 관계로 간주 → 객체 간의 일반적인 관계로 처리된다는 뜻

클래스도 ①클래스를 정의하고 객체를 생성, ②객체들의 관계를 파악해보니 클래스가 나오는 경우 → 두 가지임
→ 다이어그램도 ①클래스 다이어그램으로 객체 다이어그램을 그릴 수도 있고, ②객체 다이어그램을 보고 클래스 다이어그램을 그릴 수도 있는 것 → 예) 다수성 밝히기가 힘들면 객체를 먼저 만들어보고

꼭 클래스 다이어그램이 1:1 관계라고 객체도 1개만 존재하는 것은 아님
→ 여러 객체가 존재하는데 배타적이면 n개가 나타날 수 있음
→ 예) 오븐 객체도 되고 티비 객체도 되는데, 같이 담길 수는 없는 경우
혹은 여러 클라이언트 중, 서버-클라이언트 관계에서 서버에 1개의 클라이언트만 연결되어야 하는 경우

관계는 객체들 간의 관계를 표기한 것, 관계는 동사여야 한다. → 즉 화살표는 동사(메서드)를 의미한다.
명사를 쓸 때는 Role(역할)을 표기할 때 쓰는 것 → 책에서도 주문이 아니라 담겨지기가 맞음, 하다못해 동명사로

p.62 그림3-16*
코드를 보면 멤버로 있어 구성 관계로 볼 수 있고 OR 쇼핑카트가 없으면 주문이 불가하니 의존 관계로도
→ 이처럼 여러 관계가 나올 수 있음. 소스코드만 봐서는 알 수가 없음
객체지향에서 소스코드만으로 모든걸 표현할 수 없음 → 항상 설계 도구가 같이 제공되어야 함
소스코드만 봐서는 프로그래밍 의도를 파악하기 힘듦(설계도를 같이) → 동작 원리가 설계도에 나타나 있음

예를 들어 **<학생 평균 계산 시스템>**을 구축 한다고 했을 때,
객체는 입력, *계산*, 출력으로 역할을 나눌 수 있음
유스케이스 다이어그램에선 입력에선 1. DB로부터 입력을 받고/전달, 계산에선 평균값 계산/결과 알려줌
(입력에서 전달을 받기 때문에, 따로 또 요청을 하면 의미가 중복; 하나만 있어야 함), 출력은 외부에 알려줌
→ 메서드를 어느 정도 수준으로 열어줄 것인가(계산은 모두가? 아니면 관련자만?)는 의도에 따라 다른 것임

Inception 단계(발상, 착안) → 목표와 요구사항 파악하는 단계 → 데이터의 흐름이나 의존성에 따라 달라짐
**→ 얘를 이제 클래스 다이어그램으로 나타내면 □→□→□가 될수도 있고, □←□←□가 될수도 있음.
데이터가 없으면 계산이 안된다고 생각할 수도, 계산이 없는데 입력 역할이 필요한가?라고 생각할수도 있음
그래서 **설계도와 설계 이유가 함께 제공되어야 하는 것임
→ 여러 관점 중에 가장 합리적인 설계를 하나 결정하는 것
→ 이런 생각들을 표현하기 위해 UML이 도구로 사용되는 것. 즉, 생각을 효율적으로 주고 받기 위해**
그래서 UML을 배우기 전에 객체지향을 먼저 배우는 것이 순서가 맞는 것임

문제 정의 → 많은 히스토리(그림 그리고, 클래스→ 속성,메서드 줘서) → 설계

3. 자바 객체 구현(구현 관점)
객체 구현
UML에서의 설계된 객체를 통해 구현 p.62

1:1 관계는 "객체가 1개만 존재해야 한다"는 것이 아닌 → 각 객체가 서로 단 하나의 연결만 유지해야 한다는 뜻
이 관계에서 객체의 수는 여러 개일 수 있지만, 각 객체 간의 연결은 하나로 제한됩니다.

④캡슐화 & ⑤정보은닉 #객체지향 5대 특성 → 재사용성, 정보은닉, 캡슐화, 상속, 다형성

④-1. 캡슐화의 개념

캡슐화(번들링) : 관련된 속성과 행위를 묶는 메커니즘(구조) → 클래스에 대한 적정성을 판단하는 기준 중 하나
①클래스를 통해 캡슐을 정의
②한 캡슐 내에는 서로 관련된 속성과 행위가 묶여있어야 한다.
③캡슐화는 추상화를 표현한 방법 중 하나이다.

***캡슐화의 특성** → 관점이라 굉장히 중요
①추상화의 단위* : 복잡한 내용은 숨기고, 중요한 부분만 표현 → 캡슐화된 객체(클래스)가 최소 단위*
≒ 모델링의 방법 중 하나(→ 목적에 맞는 펙터만 남김) → 예) 유스케이스 액터 → 왜곡, 손실
②재사용의 단위 : 클래스 혹은 객체 단위로 재사용 → 데이터/함수 단위로 재사용하는 것에 비해 재사용성↑
예) 복합객체
③정보은닉을 실현하는 장치 : 공개 인터페이스(public 메서드)를 제외한 속성, 메서드를 캡슐 속으로 숨김
* 정보은닉 → 물리적 형태(비유)가 캡슐화와 동전의 양면 같은 개념이다. (서로를 전제)
→ 두 개념이 서로 밀접하게 연결, 각각이 존재하기 위해서는 상대방이 필요
캡슐화가 없으면 정보은닉을 구현할 수 없음 → 속성과 메서드가 한데 묶이지 않기 때문에, 내부 구현 세부사항을 숨기는 것이 불가능
정보은닉이 없다면 캡슐화는 의미가 없어짐 → 캡슐화 되어도 내부 데이터가 외부에 노출되면, 캡슐화 이점 상실

캡슐화와 정보은닉
캡슐화 : 관련된 요소를 묶어줌으로써 캡슐 내부와 외부를 구별짓는 메커니즘(구조)이다. → 구조적 측면 성격
→ 객체 간 서로 상대방 객체의 공개 인터페이스만 볼 수 있음, 속성은 보지 못함 → 보안적 측면 성격
정보은닉 : 캡슐 내의 요소들에 대한 세부 구현 사항들을 외부에 숨기는 장치 → 구현의 세부사항을 숨기는 것
→ 외부로 노출될 필요가 없는 속성이나 행위를 내부로 숨기는 것 → 외부에서 직접 접근하지 못함
→ 캡슐화가 되었다고 반드시 정보은닉이 되는 것은 아님 → 가시성을 조절하여 정보은닉을 실현
→ 캡슐화와 정보은닉 → (전제) 쪼갤 수 있는 최소 단위는 클래스 또는 객체임
근데 왜 공개가 아닌 은닉에 초점을 두었을까? 은닉에 더 관심이 많기 때문
→ 캡슐화를 왜 하느냐? 정보를 은닉하기 위해 ↔ 캡슐을 전제로 했을 때 얼마만큼 공개할 것이냐?
※ 은닉이 있으면 공개도 있는 것임

④-2. UML에서의 캡슐화(설계 관점)

캡슐화 표기법
캡슐화 표기는 클래스 설계 시 묵시적으로 반영되어 설계

④-3. 자바에서의 캡슐화(구현 관점)

캡슐화 표기법 → 예 : Product 단위로 재사용 가능, 복합 객체 멤버필드로도 사용 가능
캡슐화 구현은 클래스 구현을 기반으로 추상화, 재사용, 정보은닉(-)

⑤-1. 정보은닉의 개념

정보은닉의 정의
정보은닉 : 캡슐 속 속성과 메서드들 가운데, 외부로 노출시키지 않고 내부에 감싸는 장치 → 예 : 빨간 알약
①정보은닉은 블랙박스이다. : 노출시킬 필요가 없는 속성 또는 메서드들만 은폐 → *속성을 은닉, 행위를 공개
②공개 인터페이스(public 메서드)를 제공해야 한다. : 반드시 하나 이상 → 객체와 객체를 인터페이스
→ 다른 객체들에게 자신이 어떠한 서비스를 제공할 수 있는지를 선언하는 것
→ public 메서드는 자신의 오퍼레이션만 제공, 세부적인 구현 사항은 제공하지 않음 → 예:마우스-마우스 버튼
→ S/W 공학에서의 관심 분리*를 반영 → 클래스는 배타적이어야 함 → 다른 객체들과 동시에 혹은 포함 X
→ 목적이 다르기 때문에 분리하는 것. → ∵ 비슷한 관심을 다 섞어버리면 용도나 목적이 분명하지 않게 됨

→ 객체 각각의 응집력은 높이고, 객체간의 결합력은 낮추는 것(하나의 목적을 지향)
→ 오퍼레이션은 내부용(private)와 외부용(public) : 공개 인터페이스(접촉된다는 의미)이 존재
※ 관심 분리 : S/W 공학의 중요한 원칙 중 하나로, 프로그램이나 시스템을 서로 다른 관심사로 나누어 모듈화
이를 통해 각 모듈이 특정 기능, 책임에만 집중 가능 → 전체 시스템의 복잡성↓, 유지보수성↑
정보은닉은 캡슐화와 함께 사용된다. → +(public), -(private), #(protected), ~(package) → 존재 X(표기 X)가 아님

정보은닉의 이점 → 이점/단점은 특징으로부터 나오는 당연한 것, 꼭 모든 장점을 만족해야 정보은닉은 아님
①추상화를 향상시켜준다. : 캡슐화와 정보은닉을 통해 객체 간 추상화 정도 비교 가능, 정보은닉 많이 될수록 ↑
②내부 속성이나 메서드의 변경이 용이하다. : 객체 내 속성, 메서드 변경 시 타 객체에 영향을 주지 않음
→ 통제 가능하기 때문
③모듈의 독립성↑ : 다른 모듈과의 의존도로 독립성 정도를 판단 = 변경이 타 모듈에 끼치는 영향의 정도
④확장성↑ : 정보은닉은 객체 단위로 이루어짐 → 삭제 시 영향 X, 추가도 용이
→ 객체의 내부 구현/특정 속성을 삭제할 때, 해당 객체를 사용하는 다른 객체나 모듈에 영향을 미치지 않음
→ 새로운 속성이나 메서드를 추가할 때, 기존의 코드나 객체에 미치는 영향이 적음(∵공개 인터페이스로만 소통)
→ 캡슐화로 모듈 독립성이 존재하게 되고, 정보은닉을 통해 추상화 정도를 한 차원 향상시키는 것

⑤-2. UML에서의 정보은닉
정보은닉 표기법 → 추상화 정도 ↑
'-' 접근자 이용 → 외부로 공개되지 않을 속성, 오퍼레이션에 대해 블랙박스화(불필요한 부분 외부에서 접근 X)

⑤-3. 자바에서의 정보은닉
private, public, protected, 패키지 전용(디폴트) 접근 제어자를 통해 정보은닉을 실현
→ 내부에서만 사용할 메서드를 private하면 사용자는 혼란을 줄일 수 있음

Intra 클래스 → 오버로딩 ↔ Inter 클래스 → 상속, 오버라이딩

시험 출제

① 해당 클래스를 UML로 표현하라. ↔ UML로 표현한 것을 보고 무슨 뜻인지 설명하라.

→ **p.31 → UML을 자바로 변환하시오, 자바를 UML로 변환하시오. → 100% Compotable하다. (런타임에선 X)

② 중요한 개념 → static은 무엇인가? 혹은 관심 분리가 의미하는 바를 쓰시오. (주요 키워드를 중심으로 풀어 쓰기)

→ *도메인을 잘 반영할 수 있는 모델링이라는 말의 의미가 무엇인가?(O) → 모델링 목표를 쓰시오(X) →개조식 X

③ 장점, 단점, 특징, 비교* → 공부의 마무리는 비교

④ 100점 방지 혹은 0점 방지용 → 집중력 떨어지거나, 모를거 같은거 + 틀릴 수가 없는 문제

⑥ 시험에선 *선무당이 사람 잡는다를 uml로 표현하시오. 같은 것들이 나올 것임. (클래스 릴레이션 선언)

⑦ *의존 관계 코딩하는 법

⑧ 시험에서 묻는 항목은 모두 써야 함, A와 B에 대해 물었는데 A만 아무리 잘 써도 5점임

→ 쓴거 보면 이해한건지, 암기한건지 다 보임, 단순 암기는 점수 잘 안 줄거임 → 표현이 서툴더라도 이해가 중요

⑨ 요약, 기초, 연습, 실습문제 꼭 풀어볼 것 → 정의하시오, 요구사항↔클래스 추출(이게 최종 목표임, 중간 이후엔 계속 이것만 함)

→ 챕터 끝나면 연습문제 풀어보기 → 책 안에서만 + 없는건 내 생각 쓰기 → 다른 단원이랑 엮어서도 낼 수 있음*

⑩ 시스템 구현 응용 실습 간단한 문제 2문제 정도 출제할 생각임. → 쉬운거 1개, 어려운거 1개

→ 어떤 객체? 내용은 어디에 들어가야 할 것인지? 등

⑪ 혹은 어떤 말에 대해서 문장을 더 정교하게 표현해봐라 같은?

*항상 카테고리(도메인)을 생각해야 함 → 학교에선 학생, 집에선 아들이듯 → *근거가 중요

→ 서술할 때 대전제임 → 시험/과제에서 서술할 때도 ~관점에서 설계하였다. → 도메인부터 정의하고 시작해라