

# 7. 설계 및 구현

---

# 주요내용

---

- ❖ 프로젝트에서 설계란 무엇인가?
- ❖ 프로젝트에서 설계는 왜 중요한가?
- ❖ 프로젝트에서 설계 원리는 무엇인가?
- ❖ 효과적인 모듈 설계는 어떻게 해야 하는가?
- ❖ 객체지향 설계란 무엇인가?
- ❖ UML 기반의 설계란 무엇인가?
- ❖ 구현 작업이란 무엇인가?

# 목차

---

## ❖ 강의 내용

- 설계의 정의
- 상위 설계와 하위 설계
- 설계 원리
- 효과적인 모듈 설계
- 객체지향의 개념
- UML 기반의 설계
- 구현

## ❖ 팀 프로젝트 (11주차)

- 설계 문서 작성 및 제출

# 설계란?

---

## ❖ 정의

- 설계는 개발될 제품에 대한 의미 있는 공학적 표현
- 설계는 고객의 요구사항으로 추적 가능해야 하며, 동시에 좋은 설계라는 범주에 들도록 품질에 대해서도 검증되어야 한다  
[IEEE-Std-610]

## ❖ 소프트웨어의 설계(design)

- 본격적인 프로그램의 구현에 들어가기 전에 소프트웨어를 구성하는 뼈대를 정의해 구현의 기반을 만드는 것
- 종류
  - 상위 설계(High-Level Design)
  - 하위 설계(Low-Level Design)

# 상위 설계와 하위 설계

---

## ❖ 상위 설계(High-Level Design)

### - 의미

- 아키텍처 설계(Architecture Design), 예비 설계(Preliminary Design)라고 함
- 시스템 수준에서의 소프트웨어 구성 컴포넌트들 간의 관계로 구성된 시스템의 전체적인 구조
- 시스템 구조도(Structure Chart), 외부 파일 및 DB 설계도(레코드 레이아웃, ERD), 화면 및 출력물 레이아웃 등이 포함됨

## ❖ 하위 설계(Low-Level Design)

### - 의미

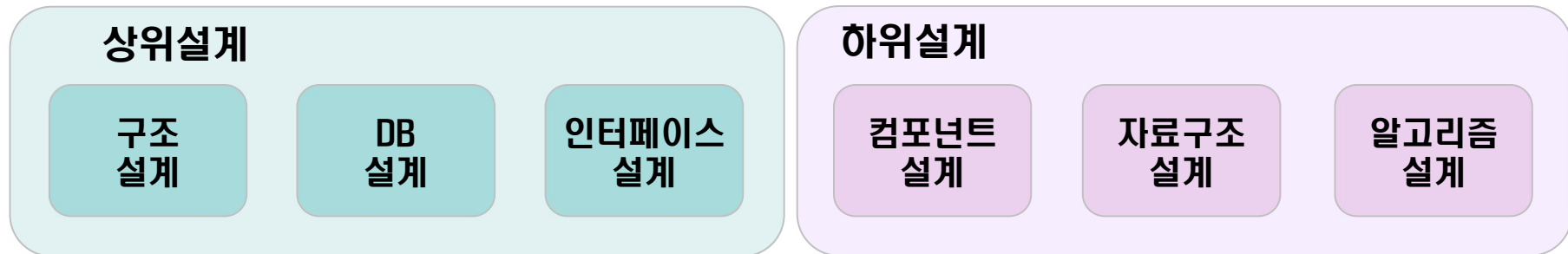
- 모듈 설계(Module Design), 상세 설계 (Detail Design)이라고 함
- 시스템의 각 구성 요소들의 내부 구조, 동적 행위 등을 결정
- 각 구성 요소의 제어와 데이터들간의 연결에 대한 구체적인 정의를 하는 것

### - 하위 설계 방법

- 절차기반(Procedure-Oriented), 자료위주(Data-Oriented), 객체지향(Object-Oriented) 설계 방법

# 상위 설계와 하위 설계의 구조도

---



# 설계 프로세스

---

# 설계 프로세스

---

## ❖ 좋은 설계란

- 요구사항 명세서의 모든 내용을 구현해야 한다
- 이해가 쉬워서 구현 또는 테스트로 추적이 가능해야 한다
- 유지 보수 시 변경이 용이해야 한다

## ❖ 설계 방식

- 프로세스 지향 설계(Process Oriented Design)
- 객체지향 설계(Object Oriented Design)



# 설계 방식

---

## ❖ 프로세스 지향 설계(Process Oriented Design)

- 업무의 처리절차를 중심으로 설계의 구성 요소들을 구분
- 어떠한 절차를 거쳐서 작업을 수행하는가, 어떠한 입출력 자료를 생성하는가에 초점
- 시스템은 “기능과 데이터” 들이 노드를 이루고 이들의 관계가 링크를 형성하는 그래프

## ❖ 객체지향 설계(Object Oriented Design)

- 시스템의 실제 객체 요소를 중심으로 설계
- 자료구조와 그에 대한 연산을 묶어서 구성되는 객체들을 정의하고 이들이 상호 작용의 기본이 되도록 설계
- 객체들이 노드를 이루고 이들간의 관계가 링크를 형성하는 그래프



# 설계 원리

---

# 설계 원리

---

- ❖ 추상화(Abstraction)
- ❖ 단계적 분해(Stepwise refinement)
- ❖ 모듈화(Modularization)

# 추상화(Abstraction)

---

## ❖ 의미

- 자세한 구현에 전에, 상위 레벨에서의 제품의 구현을 먼저 생각해보는 것

## ❖ 단계

- 상위 레벨에서 설계를 생각해본 후 점차 구체적인 단계로 옮겨가는 것

## ❖ 종류

- 과정 추상화(Procedure Abstraction)
- 데이터 추상화(Data Abstraction)
- 제어 추상화(Control Abstraction)

# 추상화의 종류 [1/2]

---

## ❖ 과정 추상화

- 수행 과정의 자세한 단계를 고려하지 않고, 상위 수준에서 수행 흐름만 먼저 설계

## ❖ 데이터 추상화

- 데이터 구조를 대표할 수 있는 표현으로 대체하는 것
- 예) 날짜 구조를 단순히 “날짜” 로 추상화 하는 것

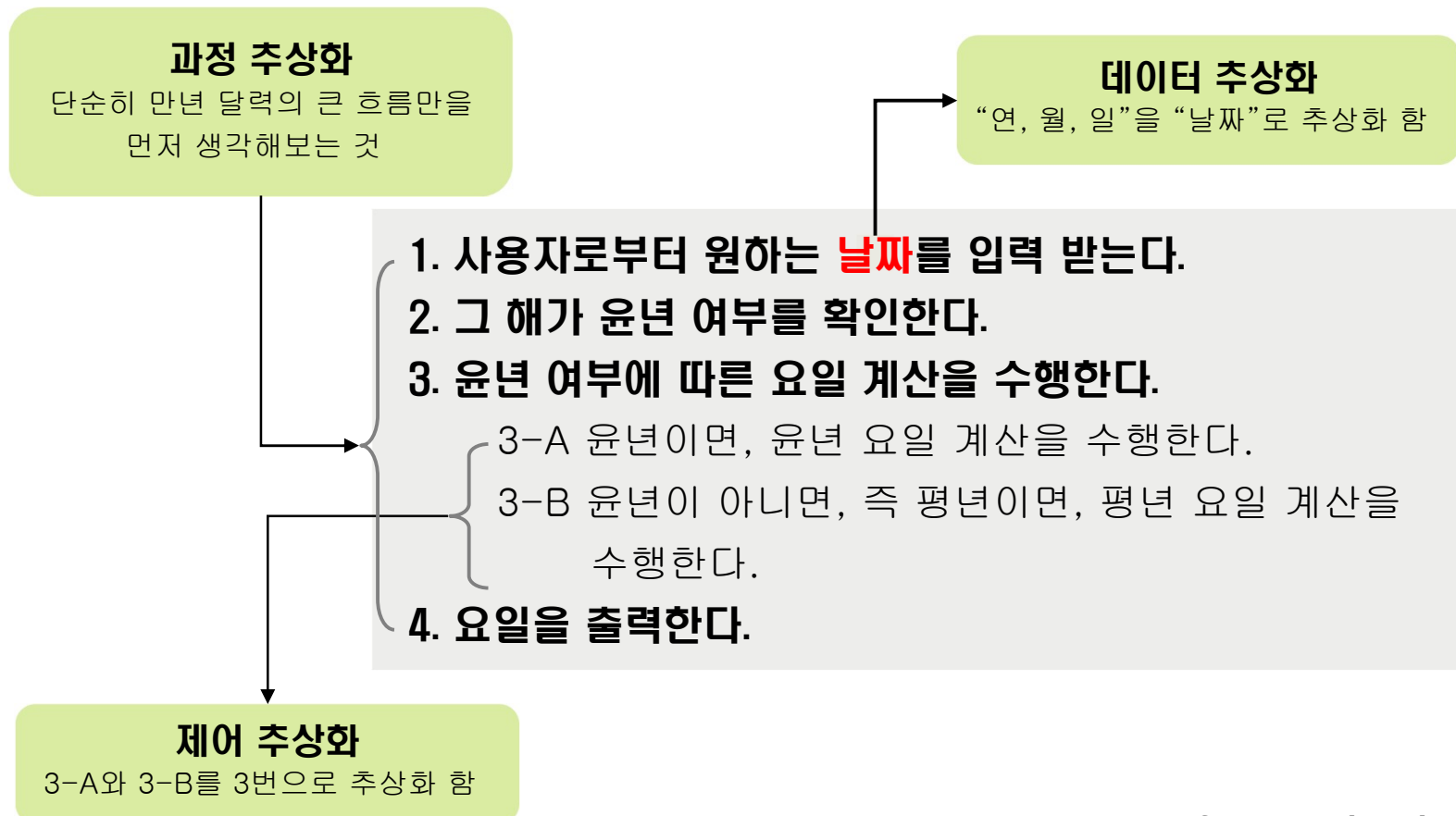
## ❖ 제어 추상화

- 3-A와 3-B를 “3. 윤년 여부에 따라 요일 계산을 수행한다.”로 추상화 하는 것

## 추상화의 종류 [2/2]

### ❖ 예제

- 원하는 날짜를 입력으로 받아 요일을 알려주는 만년 달력 프로그램



# 단계적 분해(Stepwise Refinement)

---

## ❖ 의미

- Niklaus Wirth에 의해 제안됨
- 문제를 상위 개념부터 더 구체적인 단계로 분할하는 하향식 기법의 원리
- 모듈에 대한 구체 설계를 할 때 사용

## ❖ 과정

- 문제를 하위 수준의 독립된 단위로 나눈다.
- 구분된 문제의 자세한 내용은 가능한 한 뒤로 미룬다.
- 점증적으로 구체화 작업을 계속한다.



# 모듈화

---

## ❖ 모듈의 의미

- 수행 가능 명령어, 자료구조 또는 다른 모듈을 포함하고 있는 독립 단위

## ❖ 특성

- 이름을 가지며
- 독립적으로 컴파일 되고
- 다른 모듈을 사용할 수 있고
- 다른 프로그램에서 사용될 수 있다

## ❖ 모듈의 예

- 완전한 독립 프로그램, 라이브러리 함수, 그래픽 함수 등

## ❖ 모듈의 크기

- 되도록 쉽게 이해될 수 있도록 가능한 한 작아야 함
- 너무 작은 모듈로 나뉘지지 않도록 함

# 효과적인 모듈 설계

---

# 정보 은닉(Information Hiding)

---

## ❖ 의미

- 각 모듈 내부 내용에 대해서는 비밀로 묶어두고, 인터페이스를 통해서만 메시지를 전달 할 수 있도록 하는 개념
- 설계상의 결정 사항들이 각 모듈 안에 감추어져 다른 모듈이 접근하거나 변경하지 못하도록 함

## ❖ 장점

- 모듈의 구현을 독립적으로 맡길 수 있음
- 설계 과정에서 하나의 모듈이 변경되더라도 설계에 영향을 주지 않음

# 정보은닉의 예

---

```
void main( )
{
    stack st1;
    char x, y;

    create_stack(st1);
    push(st1, 'a');
    push(st1, 'b');

    x = pop(st1);
    y = pop(st1);

    destroy_stack(st1);
    printf("%c, %c/n", x, y);
}
```

예제 A

```
void main( )
{
    stack* st1;
    char x, y;

    st1 = new stack;
    st1->top = 0;
    push(st1->stack_value[st1->top], 'a');
    push(st1->stack_value[st1->top+1], 'b');

    x = pop(st1->stack_value[st1->top]);
    y = pop(st1->stack_value[st1->top-1]);

    delete st1;
    printf("%c, %c/n", x, y);
}
```

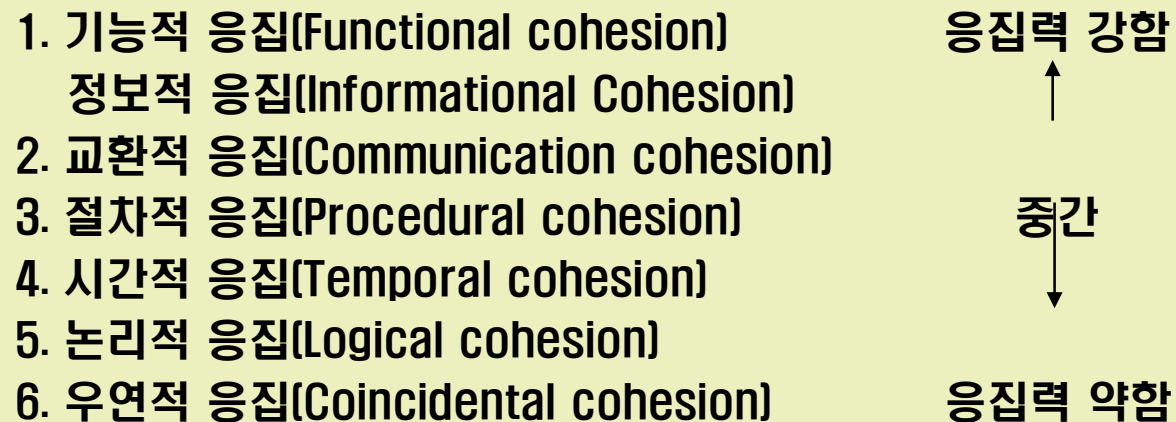
예제 B

# 모듈의 응집력 [1/2]

## ❖ 모듈의 응집력이란?

- 모듈을 이루는 각 요소들의 서로 관련되어 있는 정도
- 강력한 응집력을 갖는 모듈을 만드는 것이 모듈 설계의 목표

## ❖ Myers의 응집력 정도 구분



# 모듈의 응집력 [2/2]

---

## ❖ 응집력의 종류

- **기능적 응집(Functional cohesion)**
  - 모듈이 잘 정의된 하나의 기능만을 수행할 때 기능적 응집도가 높아짐
- **교환적 응집(Communication cohesion)**
  - 한 모듈 내에 2개 이상의 기능이 존재하고 단계별 순서에 의해서만 수행되는 경우
  - 각 기능은 동일한 입력 자료를 사용하면서도 서로 다른 출력을 생성함
- **절차적 응집(Procedural cohesion)**
  - 모듈 안의 작업들이 큰 테두리 안에서 같은 작업에 속하고, 입출력을 공유하지 않지만 순서에 따라 수행될 필요가 있는 경우
- **시간적 응집(Temporal cohesion)**
  - 프로그램의 초기화 모듈 같이 한 번만 수행되는 요소들이 포함된 형태
- **논리적 응집(Logical cohesion)**
  - 비슷한 성격을 갖거나 특정 형태로 분류되는 처리 요소
- **우연적 응집(Coincidental cohesion)**
  - 아무 관련 없는 처리 요소들로 모듈이 형성되는 경우

# 모듈의 결합도 [1/2]

---

## ❖ 의미

- 모듈간에 연결되어 상호 의존하는 정도
- 낮은 결합도를 갖는 모듈(Loosely coupled)을 만드는 것이 모듈 설계의 목표

## ❖ 모듈간의 의존도

1. 자료 결합(Data coupling)
2. 구조 결합(Stamp coupling)
3. 제어 결합(Control coupling)
4. 공통 결합(Common coupling)
5. 내용 결합(Content coupling)

결합도 약함



결합도 강함

# 모듈의 결합도 (2/2)

---

## ❖ 결합도의 종류

- **자료 결합(data coupling)**
  - 모듈 간의 인터페이스가 자료 요소로만 구성된 경우
  - 가장 이상적인 형태의 결합
- **구조 결합(struct coupling)**
  - 모듈 간의 인터페이스로 배열이나 레코드 등의 자료 구조가 전달되는 경우
- **제어 결합(control coupling)**
  - 한 모듈이 다른 모듈에게 제어 요소(function code, switch, tag 등)를 전달하는 경우
- **공통 결합(common coupling)**
  - 여러 모듈이 공동 자료 영역을 사용하는 경우
- **내용 결합(content coupling)**
  - 한 모듈이 다른 모듈의 일부분을 직접 참조 또는 수정하는 경우



# 객체지향(Object-Oriented) 개념

---

# 객체지향

---

## ❖ 객체지향의 등장 배경

- 기존의 구조적 기법으로 유지보수가 어렵다는 단점을 극복하기 위해 등장

## ❖ 객체란?

- 특성(Attribute)와 행위(Behavior)를 가지고 있는 인지할 수 있는 개체(Entity)
  - 특성
    - 해당 객체에 저장되어 있는 데이터
  - 행위
    - 객체가 할 수 있는 일, 객체의 상태가 변하게 하는 원인을 제공
- 다른 객체와 구별할 수 있는 정체성(identity)을 가짐
  - 정체성
    - 해당 객체를 다른 개체와 구별 할 수 있는 식별 값

## ❖ 객체의 예: 차

- 특성: 검정색 차체, 6기통 엔진, 자동 변속기, 4개의 바퀴 등
- 행위: 출발하다, 정지하다, 가속하다, 감속하다 등
- 정체성: 차량 번호

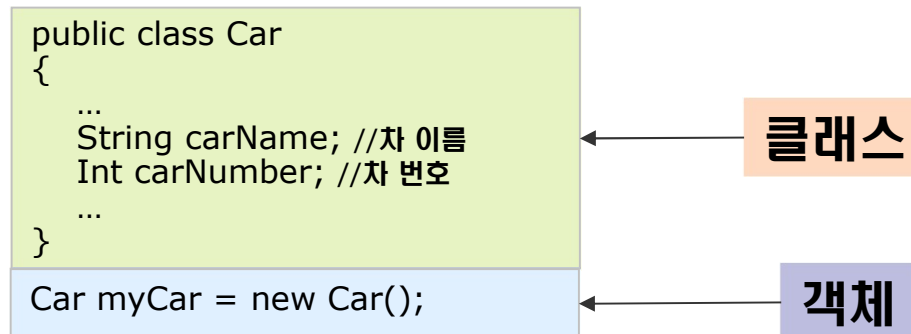
# 클래스와 객체

## ❖ 클래스(Class)

- 여러 객체들을 위한 대표적 구조
- 객체들이 내부적으로 어떻게 구성되어 있는지 설명
- 객체의 특성은 클래스의 변수로, 행위는 메소드로 표현됨

## ❖ 객체(Object)

- 클래스의 실례 또는 실체(Instance)라고도 부름
  - 객체가 클래스에서 정의하는 변수, 메소드를 그대로 가지면서 메모리에 할당되기 때문
- 각 객체 내부의 변수 이름은 같지만 서로 독립적임



# 객체지향 방법의 특징

---

## ❖ 절차를 강조하는 구조적 방법

- 데이터를 소홀히 하게 됨

## ❖ 객체지향 방법

- 시스템을 구성하는 요소들은 객체로,
- 시스템 개발의 복잡한 문제들을 캡슐화(Encapsulation), 상속(Inheritance), 다형성(Polymorphism) 개념으로 해결하려 함

# 캡슐화(Encapsulation) [1/2]

---

## ❖ 의미

- 소프트웨어 모듈인 객체의 내부에 가진 상세한 정보와 처리 방식을 외부로부터 감추는 것
- 객체의 추상화를 통해 독립성을 보장해 주는 개념

# 캡슐화(Encapsulation) (2/2)

## ❖ 캡슐화의 예

```
class Car
{
    ...
    private String carName; //차 이름
    private Int carNumber; //차 번호

    public String getCarName() // 차 이름 반환
    {
        return carName;
    }

    public Int getCarNumber() // 차 번호 반환
    {
        return carNumber;
    }
    ...
}
```

private 이용  
외부에 감춤

public 이용  
외부에 공개

# 상속(Inheritance) [1/2]

## ❖ 의미

- 다른 클래스의 속성을 물려받아 내 것처럼 쓰는 것



# 상속(Inheritance) [1/2]

## ❖ 상속의 예

```
class Car
{
    ...
    String carName; //차 이름
    Int carNumber; //차 번호
    ...
}
```

상위클래스

```
class Bus extends Car{
    ...
    int seatCount; //좌석 수
    ...
}
```

하위클래스

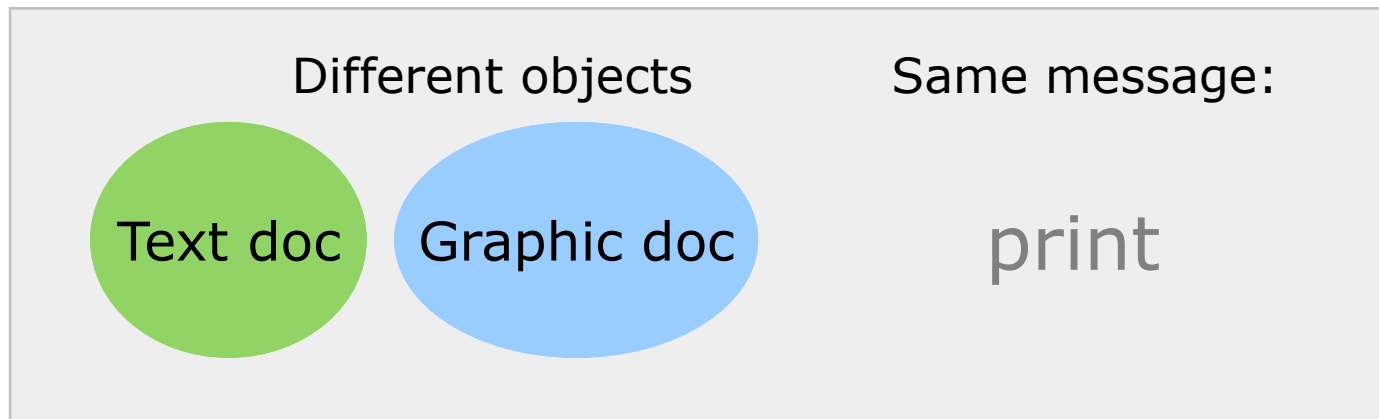


# 다형성(Polymorphism)

---

## ❖ 의미

- 하나의 인터페이스를 통해 서로 다른 구현을 제공하는 것



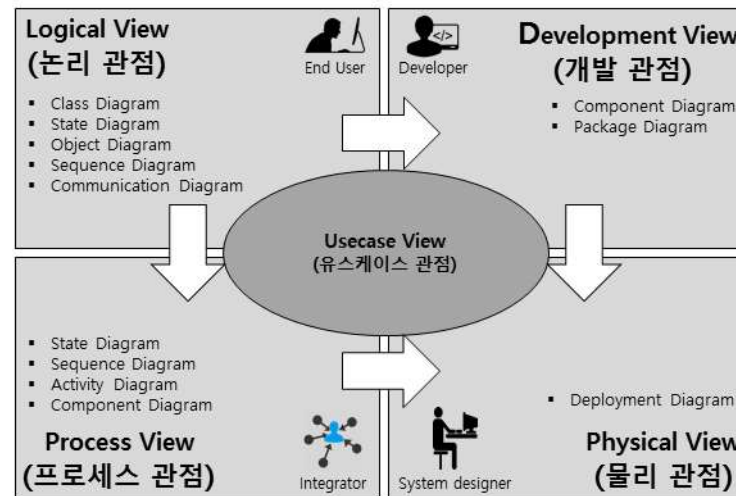
# UML 기반의 설계

---

# 4+1 View(1/4)

## ❖ 4+1 View

- 시스템의 규모가 커지고 복잡해짐에 따라 개발에 참여하는 사람[이해관계자]이 늘어남
- 시스템 개발 이해관계자들 마다 역할 및 관점이 다름
  - 각각의 관점을 중심으로 시스템을 정의함



# 4+1 View(2/4)

---

## ❖ 유스케이스 관점(Usecase View)

- 시스템 개발에 참여하는 모든 이해관계자가 시스템에 대해 이해할 수 있도록 요구사항을 정의 하는 것을 목적
- 시스템의 기능과 전체 규모에 대해 정의하며 모든 개발의 기반으로 사용
- UML-diagram : Usecase Diagram

## ❖ 논리 관점(Logical View)

- 유스케이스 다이어그램과 요구사항 기술서에 기술된 기능들을 프로그래밍 시각으로 변경하는 관점
- 객체를 추출하고, 객체들의 세부적인 속성과 동작에 대해 정의
- UML-diagram: Class, State, Object, Sequence, Communication diagram

# 4+1 View(3/4)

---

## ❖ 프로세스 관점(Process View)

- Integrator가 이해관계자들이 수행한 작업을 통합하기 위해 사용
- 시스템 개발 전반의 작업 수행 주체와 작업의 흐름 정의에 중점
- UML-diagram : Dynamic(state, sequence, collaboration activity), Implementation(component and development)

## ❖ 개발 관점(Development View)

- 실제 코드를 작성하는 개발자들의 관점으로 시스템이 어떻게 개발되는지에 대해 정의
- 계층 구조, 제약 사항, 코드 재사용 등과 같은 시스템 구현을 위한 요건을 정의
- UML-diagram : Component(logical 단위), Package(physical 단위) diagrams

# 4+1 View(4/4)

---

## ❖ 물리 관점(Physical View)

- 소프트웨어와 하드웨어의 관계와 결합구조를 정의하기 위한 관점
- 시스템 디자이너가 시스템의 설치 방법과 실행 환경을 정의하는데 사용
- UML-diagram : Deployment diagram

# 클래스 다이어그램

---

# 클래스 다이어그램(1/4)

---

## ❖ 클래스 다이어그램(Class Diagram)

- 클래스(Class)들과 그들의 관계를 나타내는 다이어그램
- 클래스를 식별하고 그 관계를 정의한다.

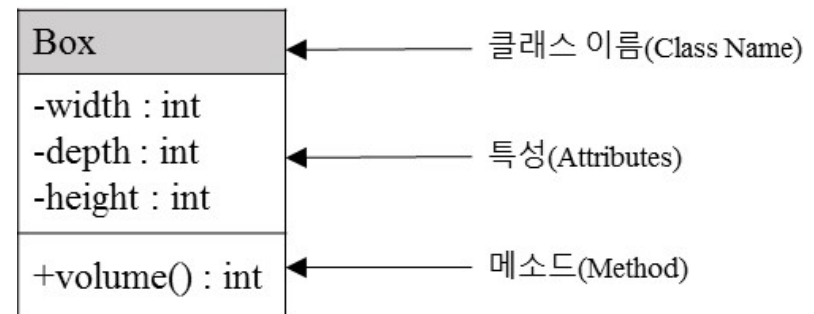


# 클래스 다이어그램(2/4)

## ❖ 구성 요소와 표기법

### - 클래스(Class)

- 클래스는 클래스 이름과 특성, 메소드 등으로 구성된다.
- 클래스 이름
  - 만들고자 하는 클래스의 고유한 이름
- 속성
  - 클래스가 갖는 속성으로, 클래스의 데이터
- 메소드
  - 클래스의 행위적 특징을 나타내므로 행위(Behavior)

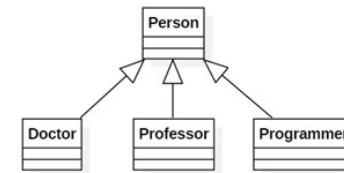


# 클래스 다이어그램(3/4)

## ❖ 구성 요소와 표기법

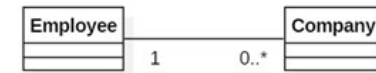
### - 관계(Relationship)

- 클래스는 독립된 단일 기능을 제공하여, 클래스 간의 상호작용을 관계로 나타낸다.
- 종류
  - 일반화(Generation)
    - 어떤 클래스의 속성을 물려받아 새로운 클래스를 도출하는 것 즉, 상속(Inheritance)



### ➢ 연관(Association)

- 클래스 간에 서로 개념적으로 연결된 경우를 의미하며, 실선으로 표현



# 클래스 다이어그램(4/4)

## ❖ 구성 요소와 표기법

### - 관계(Relationship)

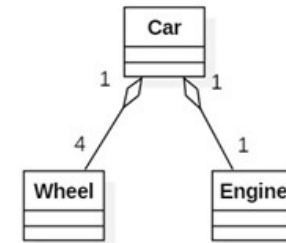
#### ➤ 직접 연관(Directional association)

- 연관 관계이지만 하나의 클래스만 다른 클래스의 행위를 요청하고 사용하는 관계



#### ➤ 집합(Aggregation)

- 여러 개의 독립적인 클래스들이 하나의 클래스를 구성하는 경우
- 다이아몬드 형태가 붙어있는 쪽이 전체를 나타내는 클래스이고, 붙어있지 않은 쪽이 전체를 구성하는 클래스들을 나타낸다.



### • 다중성(Multiplicity)

- 클래스들 간의 관계에서 대응하고 있는 각 클래스들에게 허용되는 객체의 개수

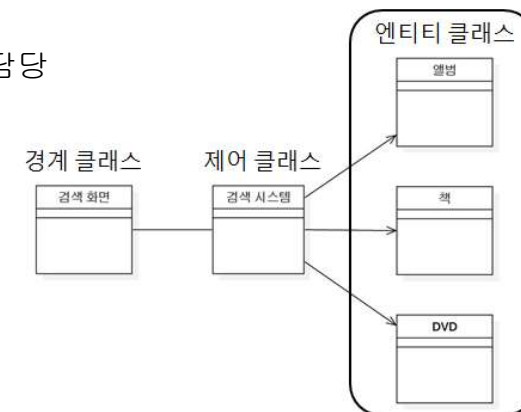
표현방법	내용
1	한 객체와 연관된다.
0..1	0개 또는 1개의 객체와 연관된다.
0..*	0개 또는 그 이상의 객체가 연관된다.
n	n개의 객체와 연관된다.
n..m	n개에서 m개까지의 객체가 연관된다.

# 클래스 다이어그램 작성[1/2]

❖ 클래스 다이어그램을 작성하기 위해서는 먼저 어떤 종류의 클래스들이 있고, 그 클래스들이 어떤 역할을 하는지를 알아야 한다.

## - 클래스 타입(Class Type)

- 클래스들은 그 역할에 따라 엔티티 클래스, 경계 클래스, 제어 클래스로 분류
- 엔티티 클래스(Entity Class)
  - 문제 영역을 구성하는 컴포넌트들
  - 사용 사례에서 반복되어 나오는 용어로 명사 추출법을 이용해 추출
- 경계 클래스(Boundary Class)
  - 시스템 외부의 액터와 상호 작용하는 클래스로 사용자 인터페이스의 역할을 제공
- 제어 클래스(Control Class)
  - 경계 클래스와 엔티티 클래스 사이에서 중간 역할을 담당



# 클래스 다이어그램 작성[2/2]

---

## - 클래스 추출

- 시스템에서 클래스를 추출할 때, 클래스의 타입 별로 이를 추출하는 과정에 차이가 있다.
- 경계 클래스
  - 화면 요구사항을 참고해 클래스를 추출
- 제어 클래스
  - 유스케이스를 참고해 클래스를 추출
    - 예를 들면, 인터넷 서점 시스템에서 유스케이스들로 추출된 '검색하다', '장바구니에 담는다' 등의 기능들을 참고하여 제어 클래스를 만든다.
- 엔티티 클래스
  - '명사 추출법'을 이용해 추출
    - 명사 추출법이란, 요구사항이 적혀있는 한 문단의 시스템 개요에서 명사들을 식별한 후 클래스로 적당하지 않은 명사들을 삭제해 엔티티 클래스를 추출하는 방법


- 1) 문단으로 시스템 개요를 서술
- 2) 명사들을 식별
- 3) 문제의 경계 밖에 있는 명사나 추상 명사 제외
- 4) 클래스 후보 도출
- 5) 클래스 선정

# 예제[1/7]

- ❖ 아래에 적힌 요구사항은 인터넷 서점 시스템에 대한 요구사항의 일부를 나타낸다. 아래 화면은 요구사항을 기반으로 구축된 화면 중 일부이다. 주어진 시스템 개요와 화면 요구사항을 참고해 초기 클래스 다이어그램을 작성해보자.

[시스템 개요]

· 본 시스템은 책, 음반, DVD를 구입할 수 있는 시스템이다. 시스템을 이용해서 상품 목록을 조회할 수 있고, 상품명을 통한 검색을 할 수 있다. 이 때, 출력되는 화면에는 종류(책, 음반, DVD), 상품명, 가격이 포함된다. 출력된 목록에서 상품을 선택해 구입할 수 있다.



종류	상품명	가격
----	-----	----

## 예제[2/7]

### ❖ 클래스 추출

- 인터넷 서점 시스템에 대한 시스템 개요와 화면 요구사항을 참고해 엔티티 클래스 및 경계 클래스, 제어 클래스를 추출해보자.

### - 엔티티 클래스

- 위에서 설명한 명사 추출법을 사용해 엔티티 클래스를 추출 한다.

#### 1) 문단으로 시스템 개요를 서술

- 제시된 시스템 개요를 사용

#### 2) 명사들을 식별

시스템, 책, 음반, DVD, 시스템, 상품 목록, 상품명, 화면, 종류, 가격, 상품

#### 3) 문제의 경계 밖에 있는 명사나 추상 명사 제외

시스템, 화면, 종류

#### 4) 엔티티 클래스 후보 도출

책, 음반, DVD, 상품 목록, 상품명, 가격, 상품

#### 5) 엔티티 클래스 선정

책, 음반, **DVD**, 상품, 상품 목록

상품
-productName: String -productPrice: int -numberOfStock: int +decreaseNumberOfStock(int num)

상품 목록
-dvdList: DVD[] -bookList: 책[] -albumList: 음반[] +delDVD(dvd: DVD) +delBook(book: 책) +delAlbum(album: 음반) +addDVD(dvd: DVD) +addBook(book: 책) +addAlbum(album: 음반)

DVD
-productName: String -productPrice: int -numberOfStock: int -runTime: int +decreaseNumberOfStock(int num)

책
-productName: String -productPrice: int -numberOfStock: int -numberOfPage: int +decreaseNumberOfStock(num : int)

음반
-productName: String -productPrice: int -numberOfStock: int -trackList: String[] +decreaseNumberOfStock(int num)

# 예제(3/7)

## ❖ 클래스 추출

### - 경계 클래스

- 제시된 화면 요구사항에서 경계 클래스를 추출해 보면 ‘메인 화면’ 클래스를 추출할 수 있다.

메인 화면
-조회 버튼 -구입 버튼 -검색 버튼 -검색 텍스트 필드 -목록 테이블
+이벤트핸들러()

### - 제어 클래스

- 제어 클래스는 유스케이스 다이어그램의 유스케이스들을 참조해야 한다.

조회 시스템
-productList: 상품 목록
+getProductContents()

검색 시스템
-productList: 상품 목록
+getSearchContents()

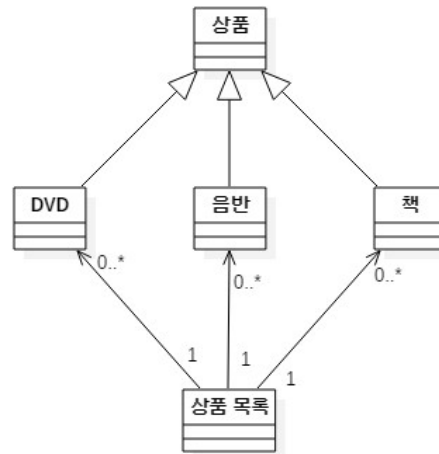
구입 시스템
-productList: 상품 목록
+구입 가능 여부 확인() +구입()



## 예제[4/7]

### ❖ 클래스 관계

- 클래스의 초기 모델을 작성하고 난 후에는 그 클래스들 간의 관계를 정의
- 엔티티 클래스 간의 관계
  - DVD, 음반, 책은 상품 클래스가 가진 가격이나 제품 이름 등의 특성을 상속
  - 상품 목록 클래스는 시스템에 등록된 DVD, 음반, 책들의 목록을 가짐

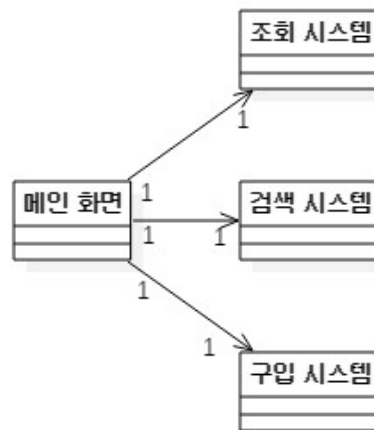


# 예제[5/7]

## ❖ 클래스 관계

### - 경계 클래스와 제어 클래스의 관계

- 경계 클래스인 메인 화면에서 상품 조회 버튼을 클릭하는 이벤트가 발생할 때 조회 시스템을 호출
- 검색 버튼을 클릭하는 이벤트가 발생할 때 검색 시스템을 호출
- 상품을 선택하고 구입 버튼을 클릭하는 이벤트가 발생할 때에는 구입 시스템을 호출

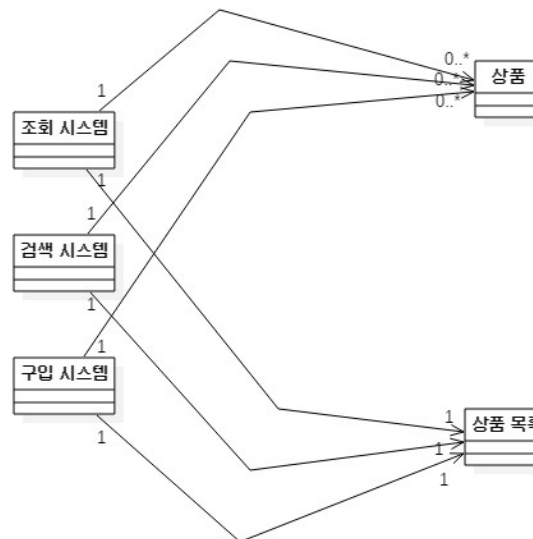


# 예제[6/7]

## ❖ 클래스 관계

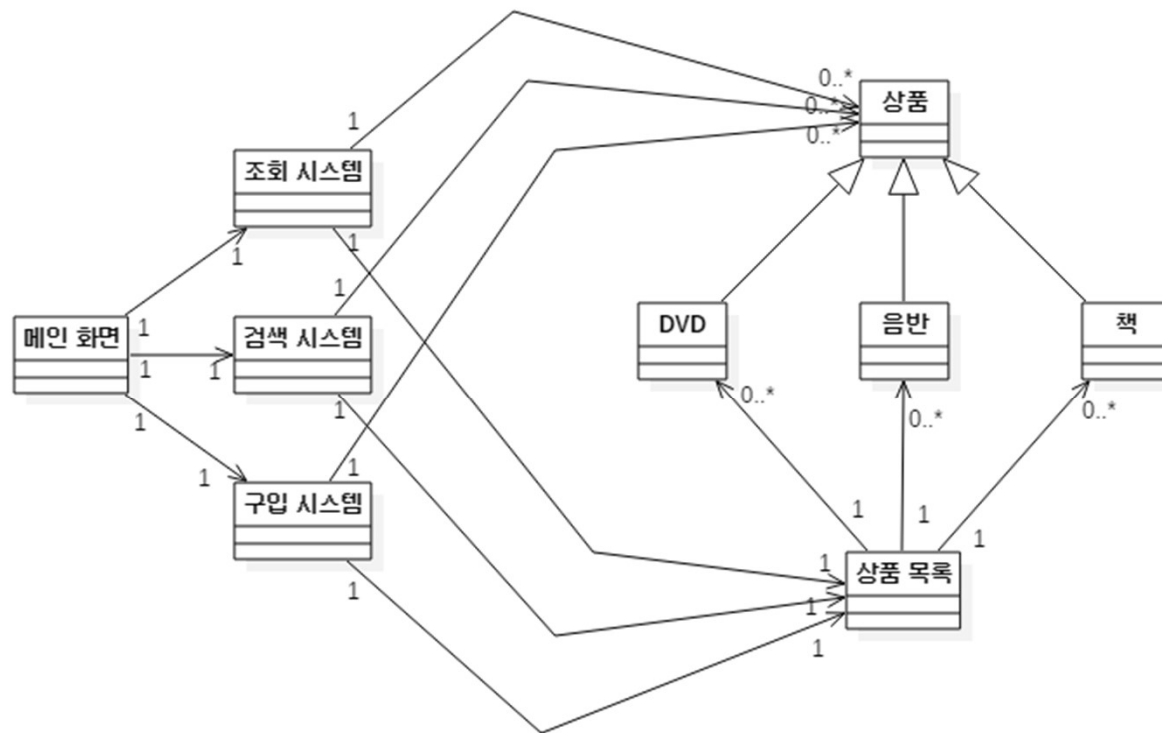
### - 제어 클래스와 엔티티 클래스의 관계

- 제어 클래스들인 조회 시스템, 검색 시스템, 구입 시스템은 모두 엔티티 클래스인 DVD, 음반, 책의 정보를 호출
- DVD, 음반, 책 클래스는 모두 상위 클래스인 상품 클래스의 관계를 상속
- 조회 시스템과 검색 시스템은 시스템에 등록된 상품 목록의 정보를 호출



## 예제(7/7)

- ❖ 앞에서 나타낸 클래스들 간의 관계를 기반으로 전체 클래스 간의 관계를 나타내면 아래와 같다.



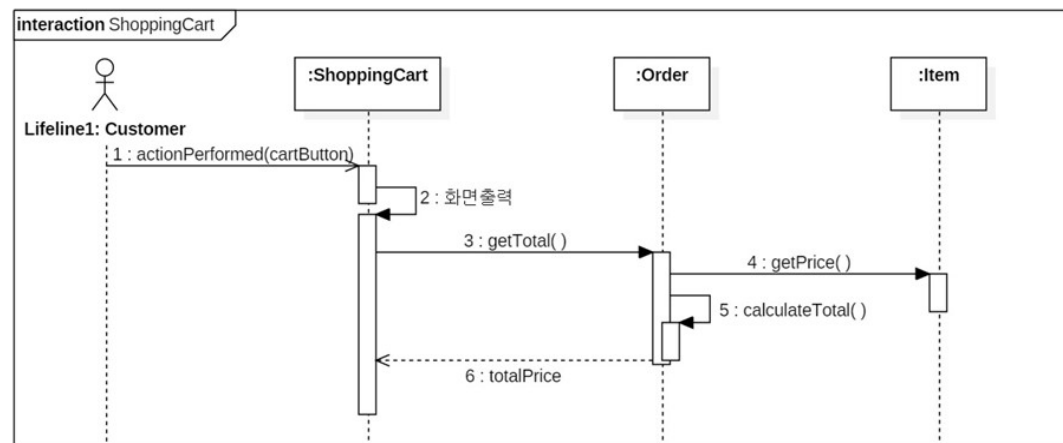
# 시퀀스 다이어그램

---

# 시퀀스 다이어그램(1/)

## ❖ 시퀀스 다이어그램(Sequence Diagram)

- 객체들 사이에서 시간에 따라 발생하는 상호작용(Interaction)을 보여주는 다이어그램
- 일반적으로 유스케이스 기술서를 바탕으로 작성된 화면 요구사항(UI Requirement)과 클래스 다이어그램(Class Diagram)을 기반으로 작성



# 시퀀스 다이어그램(2/)

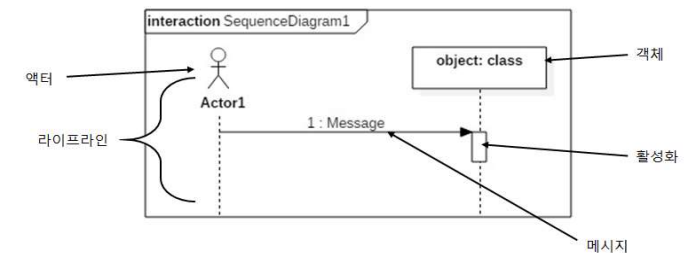
## ❖ 구성 요소와 표기법

### - 객체(Object), 액터(Actor)

- 객체(Object)란 메시지를 송, 수신 하는 주체
- 액터(Actor) 또한 객체이며 객체를 정의할 때에는 “객체명 : 클래스명” 형태로 정의
  - 객체 명 없이 클래스 명만으로 객체를 표기 하는 것도 가능

### - 라이프라인(Lifeline)

- 전체적인 흐름(Scenario)에서 참여하는 액터 또는 객체의 생성, 소멸, 활성화될 때를 나타내는 선
- 객체 아래쪽으로부터 뻗어나가도록 쇄선을 점선으로 표현



# 시퀀스 다이어그램(3/)

## ❖ 구성 요소와 표기법

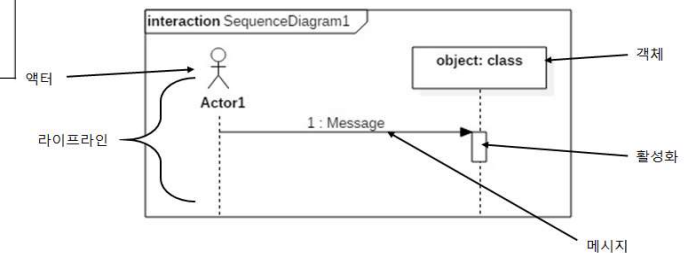
### - 활성화(Activation)

- 객체가 동작하고 있음을 표현하는 것
- 라이프라인을 따라서 좁다란 직사각형 형태로 표현

### - 메시지(Message)

- 각 개체를 호출하는 선
- 객체가 다른 객체에게 접근하기 위한 장치
- 이 선으로 객체들의 상호작용을 표현하며, 객체는 스스로에게 메시지 전송이 가능
- 메시지의 종류와 표기법은 다음과 같다.

종류	설명	기호
메시지(Message)	리턴이 있는 호출 also called call message, synchronous message	→
비동기 메시지 (Asynchronous message)	상대 라이프라인에 대한 일방적인 메시지	→
반환 메시지 (Reply message)	메시지에 대한 응답 메시지	←
셀프 메시지 (Self message)	스스로를 호출하는 메시지 예) 클래스 내부 함수 호출	↪





# 예제[1/3]

- ❖ 앞의 클래스 다이어그램의 예제였던 인터넷 서점에 대한 유스케이스 기술서 중 하나인 '상품을 검색한다' 와 화면 요구사항이 다음과 같이 주어진다.
- ❖ 이를 이용해 추출된 클래스 다이어그램을 이용해 시퀀스 다이어그램을 도출해보자.

설명	사용자가 원하는 키워드를 입력 후 검색한다.	
관련 액터	사용자	
사전 조건	시스템은 사용자에게 검색 화면을 출력한다.	
사후 조건	시스템은 사용자에게 해당 키워드가 포함된 이름을 가진 상품 목록을 출력한다.	
기본 흐름	B01	사용자는 키워드를 입력 후 검색 버튼을 클릭한다.
대안 흐름		
예외 흐름	E01	해당하는 상품이 없는 경우 -1. 빈 목록을 출력한다.
시나리오	SN001	B01
	SN002	B01 > E01-1

## 예제[2/3]

❖ 이를 기반으로 추출된 클래스들은 다음과 같다

- 경계 클래스

SearchUI
-JButton SearchButton -JTextField serachTextField -JTable courseTable -SearchSystem serachSystem
+void showSearchList() +void actionPerformed(ActionEvent e)

- 제어 클래스

SearchSystem
-Book[] bookList -DVD[] dvdList -Album[] albumList
+String[][] getSerchContents(String key)

- 엔티티 클래스

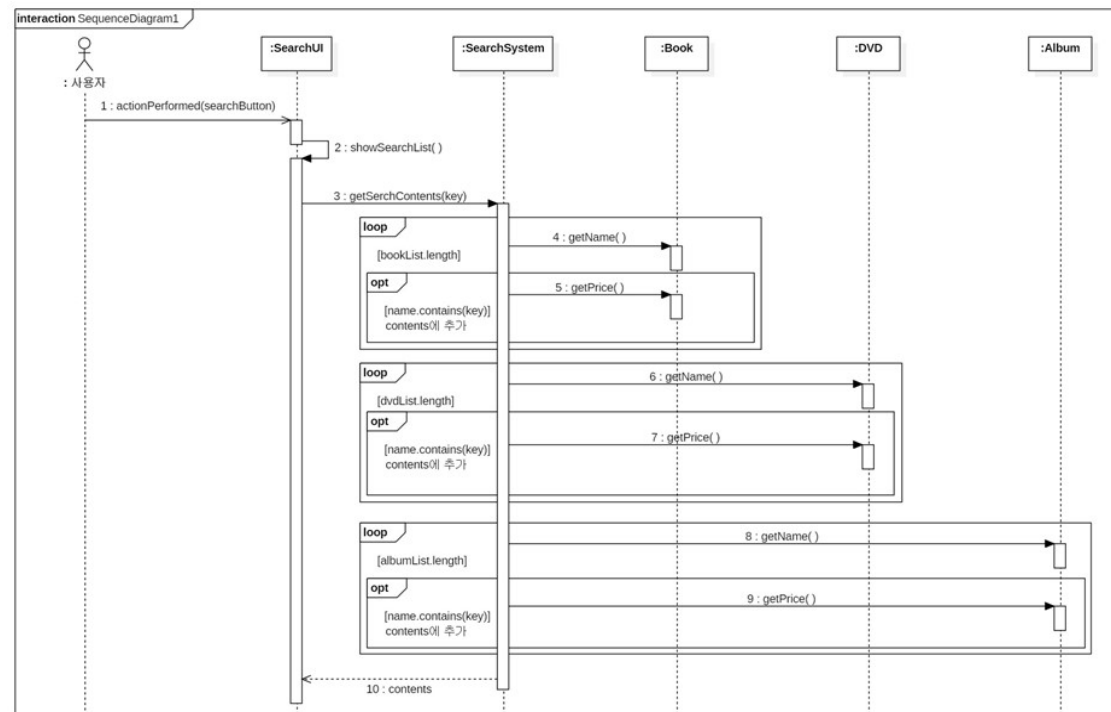
Book
-int pageNumber -String Name -int Price
+int getPageNumber() +String getName() +int getPrice()

DVD
-int runtime -String Name -int Price
+int getRuntime() +String getName() +int getPrice()

Album
-String[] musicList -String Name -int Price
+String[] musicList() +String getName() +int getPrice()

## 예제(3/3)

- ❖ 추출된 클래스 다이어그램으로부터 작성된 시퀀스 다이어그램은 다음과 같다.



# 구현

---

# 구현 [1/5]

---

## ❖ 의미

- 코드 작성 또는 프로그래밍이라고 함
- 설계의 최하위 상세화 과정
- 코드 작성, 디버깅, 통합, 개발자 테스트(단위 테스트, 통합 테스트) 작업을 포함

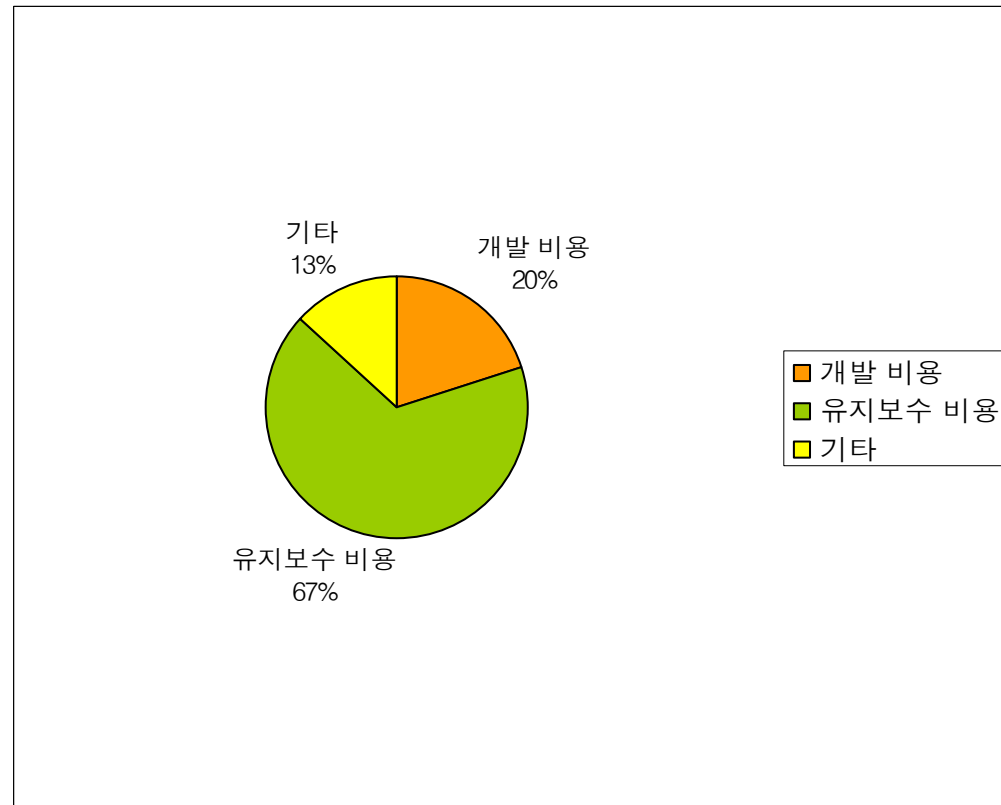
## ❖ 개발자의 코딩 스타일

- 일의 효율에 영향을 끼칠 수 있음
- 각 개발사는 코딩 스타일 지침서를 구비하여 팀원들이 지침대로 코드를 작성하도록 조율하기도 함

## 구현 [2/5]

---

### ❖ 소프트웨어 개발의 비용 분포



## 구현 (3/5)

---

### ❖ 코딩 스타일

- 한 줄에 한 문장만 써라

A

```
int score1;int score 2; double avrg;
```

B

```
int score1;    /* 첫 번째 점수의 입력 값 */  
int score 2;   /* 두 번째 점수의 입력 값 */  
double avrg; /* 두 입력 값의 평균 값 */
```

## 구현 [4/5]

### ❖ 코딩 스타일

- 선언문과 실행문을 구분하라

A

```
int score1;  
int score 2;  
double avrg;  
printf("첫 번째 점수를 입력하세요. \n");  
Scanf("%d", &score1);  
...
```

B

```
int score1;  
int score 2;  
double avrg;  
  
printf("첫 번째 점수를 입력하세요. \n");  
Scanf("%d", &score1);  
...
```



# 구현 (5/5)

---

## ❖ 코딩 스타일

- 괄호가 문장의 이해를 돕는다면 적극 활용하라.

A

```
total = score1 + ++a + score2- ++b - c;
```

B

```
total = score1 + ++a + score2- ++b - c;
```

C

```
total = score1 + (++a) + score2- (++b) - c;
```

# 코딩 표준(1/2)

---

ISO/IEC 9899:1999 (E)

©ISO/IEC

## 5.1.2.2.1 Program startup

The function called at program startup is named **main**. The implementation declares no prototype for this function. It shall be defined with a return type of **int** and with no parameters:

```
int main(void) { /* ... */ }
```

or with two parameters (referred to here as **argc** and **argv**, though any names may be used, as they are local to the function in which they are declared):

```
int main(int argc, char *argv[]) { /* ... */ }
```

or equivalent,<sup>9)</sup> or in some other implementation-defined manner.

[ISO/IEC 9899]

# 코딩 표준 [2/2]

---

## 5.1 Implementation Comment Formats

Programs can have four styles of implementation comments: block, single-line, trailing and end-of-line.

### 5.1.1 Block Comments

Block comments are used to provide descriptions of files, methods, data structures and algorithms. Block comments may be used at the beginning of each file and before each method. They can also be used in other places, such as within methods. Block comments inside a function or method should be indented to the same level as the code they describe.

A block comment should be preceded by a blank line to set it apart from the rest of the code.

```
/*
 * Here is a block comment.
 */
```

Block comments can start with `/*-`, which is recognized by `indent(1)` as the beginning of a block comment that should not be reformatted. Example:

```
/*-
 * Here is a block comment with some very special
 * formatting that I want indent(1) to ignore.
 *
 *   one
 *     two
 *       three
 */
```

[JAVA]

# 연습문제

---

1. 설계 품질을 평가하기 위해서는 반드시 좋은 설계에 대한 기준을 세워야 한다. 좋은 설계 기준은 무엇인가?
2. 결합도(coupling)가 강한 순서대로 나열하라.
3. 한 모듈이 다른 모듈의 내부 기능 및 그 내부 자료를 참조하는 경우를 무슨 결합이라고 하는가?
4. 데이터 설계에 있어서 응집도(Cohesion)는 무슨 의미인가?
5. 효과적인 모듈화 설계 방안은 무엇인가?
6. 응집도가 강한 것부터 약한 순서로 나타내어라.
7. 모듈의 구성요소가 하나의 활동으로부터 나온 출력 자료를 그 다음 활동의 입력 자료로 사용하는 같은 모듈 내에서 응집의 정도를 나타내는 것은 무엇인가?
8. 소프트웨어 개발 방법론에서 구현에 대해 설명하라. .
9. 코딩 스타일(Coding Style)과 코딩 표준(Coding Standard)의 차이점은 무엇인가?

# 팀 프로젝트

---

11주차

# 이번 주 할일

---

- ❖ 각 팀은 설계 단계에 들어간다
  
- ❖ 설계 문서 평가 기준 (5점 만점)
  - 요구사항 명세서에 맞게 설계 되었는가
  - 모듈 설계가 3개 이상으로 되어 있는가
  - UML작성은 명확한가
  - 구현에 들어갈 만큼 상세화 되어 있는가
  
- ❖ 결과
  - 3.5점 이상이면 통과 함

# 다음 주 제출 문서

---

❖ 설계 문서를 제출한다