

객체지향설계

Fall 2018



임성수 교수

Lecture 1: 객체지향 개요



수업 내용

1. 수업 목표 ←
2. 소프트웨어 개발 패러다임
3. 객체지향 모델링
4. 객체지향 개념

수업 목표



설계의 중요성

- 소프트웨어 개발에서 필수적인 요소는 코드 개발 능력뿐만 아니라 설계
- 설계는 개발의 전 단계 작업으로 구현 단계의 오류를 줄이고 개발 비용을 절감할 수 있는 중요한 단계
- 설계의 비중을 줄이고 시간에 쫓겨 구현하면 오류로 인한 더 많은 비용과 시간을 투자하는 악순환에 빠짐



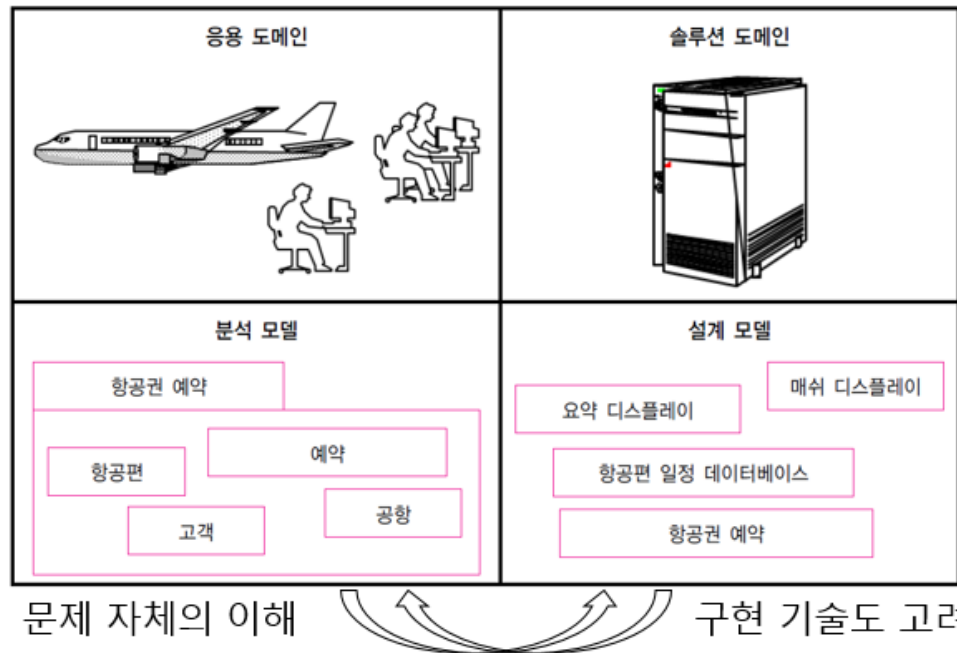
소프트웨어 개발 프로세스

수업 목표



객체지향 분석 및 설계의 특징: 모델링

- 분석과 설계의 과정이 순차적이 아닌 반복적인 사이클 형태
- 분석은 문제 영역, 설계는 솔루션 영역을 다룸

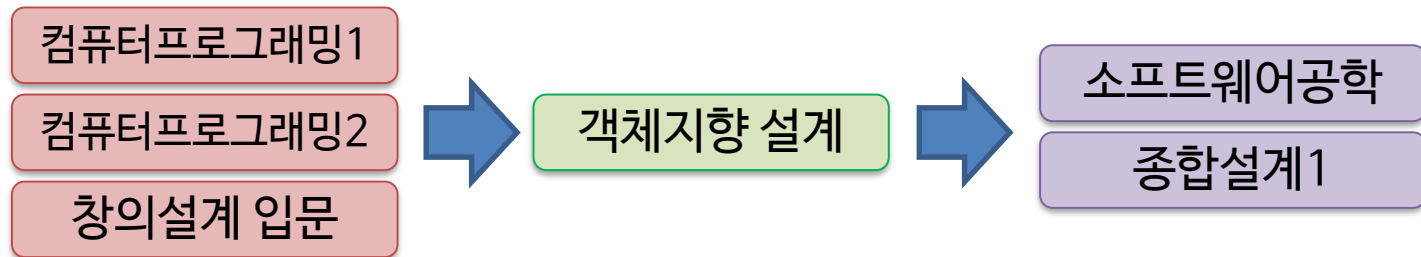


수업 목표



객체지향 설계의 첫 걸음

- 컴퓨터공학과에서 기초 프로그래밍을 익힌 후, 소프트웨어공학 및 설계의 고급 과목으로 넘어가기 전 단계로 개설
- 객체지향 개념을 더 자세히 익히고 설계에 접목할 수 있도록 학습



교과과정 이수체계도

공지사항



실습 진행

- 전반부: C++ 문법 학습 (6주차까지 예정)
- 중반부: UML 설계 학습 (프로젝트 진행을 위함)
- 후반부: 프로젝트 진행과 관련된 내용을 질문/답변

프로젝트

- 팀 선정: 다음주(~9/13)까지 1차로 희망 팀 받은 후, 나머지 인원들에 대해 제가 배정해드리겠습니다 (메일로 분반, 학번, 이름 제출).
- 제안서: 2쪽 내외로 최종 결과물에서 구현하려는 바를 항목별로 작성
- 교수 미팅: 제안서 제출 전 1회, 이후 필요에 따라 자율
- 중간 발표: 팀 당 5분 내외 및 피드백
- 최종 발표: 팀 당 10분 내외, 최종 결과물 제출



수업 내용

1. 수업 목표

2. 소프트웨어 개발 패러다임 ←

3. 객체지향 모델링

4. 객체지향 개념

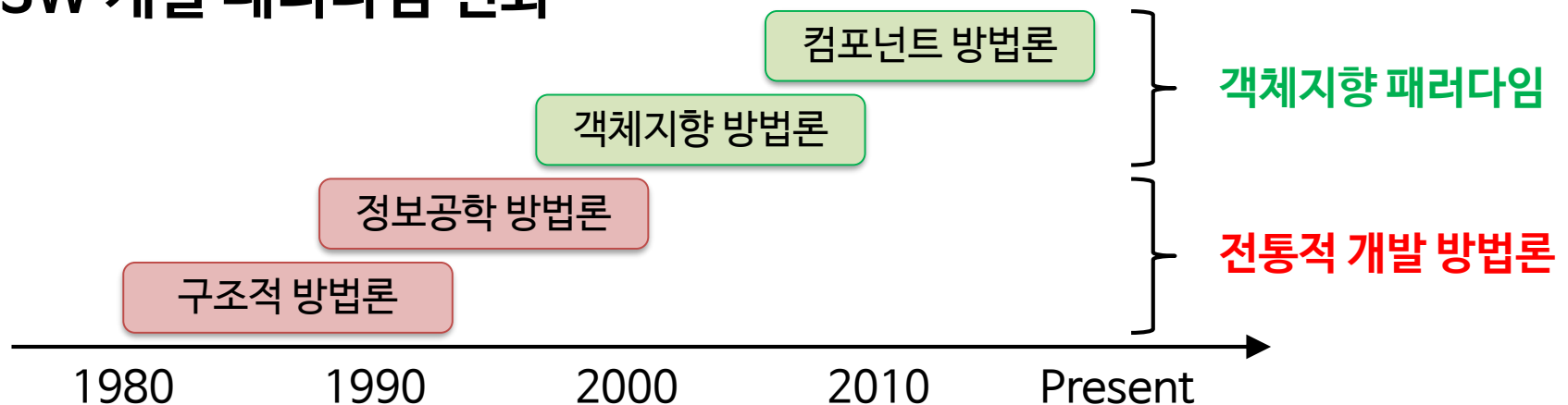
소프트웨어 개발 패러다임



SW 개발 방법론

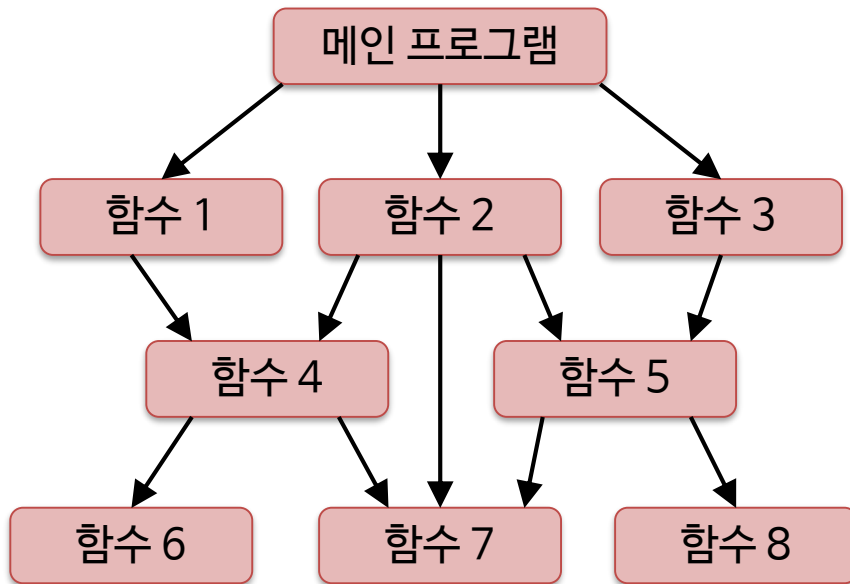
- SW 개발을 위한 단계별 작업활동, 산출물, 수행 기법 등을 정의한 체계
- 효과적인 프로젝트 관리 - 정형화된 절차와 용어로 의사소통 수단 제공
- 개발경험 축적 및 재활용을 통한 개발생산성 향상(작업 표준화/모듈화)

SW 개발 패러다임 변화

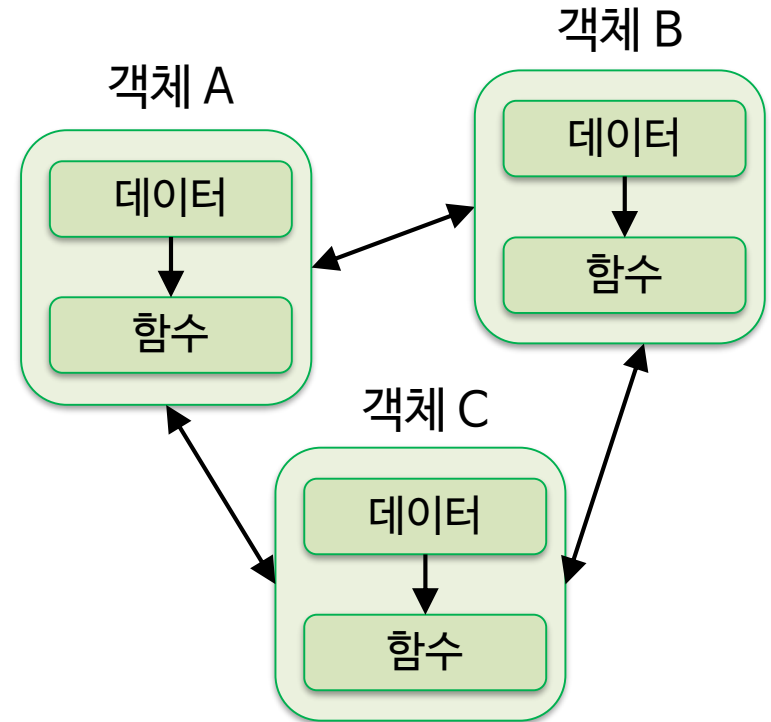




절차지향 vs 객체지향



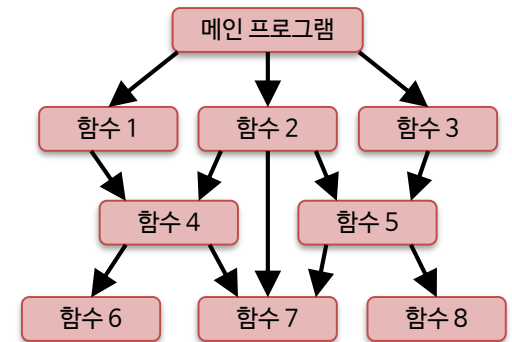
절차지향 (Procedural-oriented)



객체지향 (Object-oriented)

절차지향 패러다임

문제를 해결하는 절차를 중요하게 생각
주로 함수들의 집합으로 이루어짐



한계점

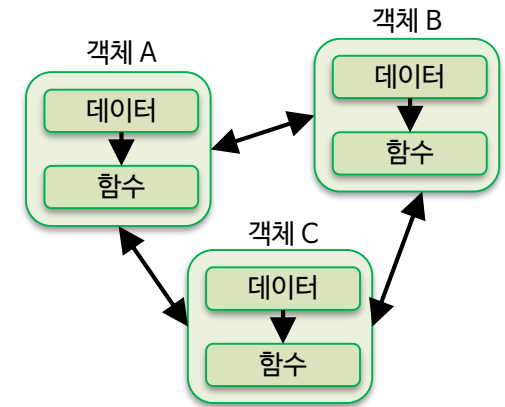
- 과도한 전역 변수의 사용
 - 모든 함수에 대해 변수가 개방되어 있어서 값이 쉽게 변할 수 있음
- 변경 및 확장의 어려움
 - 하나의 함수를 수정하면 다른 함수들도 영향을 받을 수 있음
- 프로그램 이해의 어려움
 - 함수의 개수가 많아지면 코드가 복잡하고 지저분해짐

객체지향 패러다임

절차지향에서 분산되어 있던 자료와 함수를 독립적 모듈인 객체 단위로 묶어서 구성


장점

- 코드의 재사용이 용이
 - 코드 관리가 편하고 유지보수가 용이
- 프로그래머가 사용하기 편리
 - 라이브러리 기능이 완전하여 사용하기 편리
- 응집력이 강함
 - 클래스 간 독립적으로 디자인하고 문제 해결을 위한 데이터를 모음
- **주의:** 지나치게 객체화하면 실제 개념과 달라서 이해가 어려울 수 있음



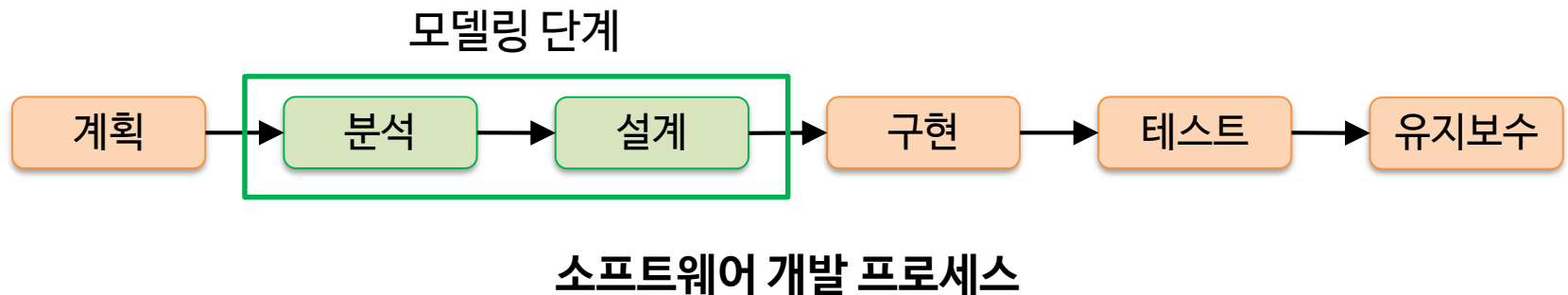


수업 내용

1. 수업 목표
2. 소프트웨어 개발 패러다임
3. 객체지향 모델링 ← 
4. 객체지향 개념

개념

- **모델링**: 현실 세계를 단순화시켜 표현하는 모델을 만드는 작업
- 소프트웨어 분야에 한정된 개념이 아닌 다양한 산업, 학문 분야에서 사용
 - 건축가의 건축 설계, 디자이너의 의상 디자인 작업 등
- 모델링 기법에 따라 모델들에 대한 형태는 다양
 - 다이어그램 형태, 명세서와 같은 스펙이나 프로토타입 등





중요성

- 개발할 시스템의 범위, 구조, 기능 이해
- SW의 규모, 복잡도가 기하급수적으로 증가함에 따라 체계적인 공학적 기법 없이 고품질의 신뢰성 있는 시스템 개발 어려움

목표

- 실세계 도메인을 잘 반영할 수 있는 모델을 선택
- 다양한 각도에서 표현할 수 있는 모델 구축
- 개발할 시스템에 적합한 모델을 선택



기대효과

- 시스템의 **가시화**
 - 개발될 시스템의 형태를 보여줘 요구사항에 부합한 시스템 개발
- 시스템의 **명세화**
 - SW구조, 기능 등에 대한 명세화를 통해 개발 시스템 범위 이해
- 시스템 **구축 기초 마련**
 - 실제 구현을 위한 기본 형태 제공
- 시스템 분석/설계의 **문서화**

소프트웨어 모델링



모델링 관점

■ 절차지향 관점

- 절차와 함수를 제어 관점에서 분할하여 시스템을 모형화
- 요구사항 변화에 적응력이 떨어짐
- 대규모 시스템에서 유지보수를 포함한 관리의 어려움

■ 객체지향 관점

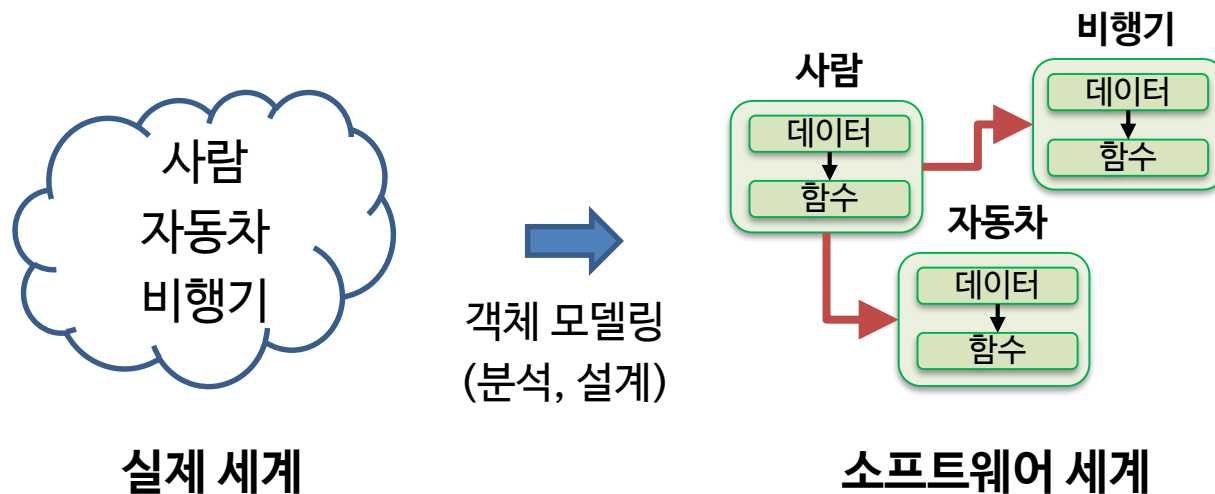
- 시스템의 기본 요소를 객체 또는 클래스로 파악한 모형화
- 객체: 사물을 말하며 고유성, 상태, 행동을 가짐
- 클래스: 공통적인 객체들의 집합



객체지향 모델링

객체지향 모델링: 실제 세계를 모델링하여 소프트웨어 개발

- 현실의 분리된 개체들이 SW 시스템 안에서 독립적 객체로 구현
- 인간의 사고 방식은 객체중심 모형과 유사함





객체지향 모델링과 구현

객체지향은 모델링에서 구현으로의 전환이 매끄러움

- 모델링 방법이 프로그래밍 단계로 쉽게 매핑 됨
- 절차적 방법: 프로세스와 모듈 간의 연결이 명확하지 않음

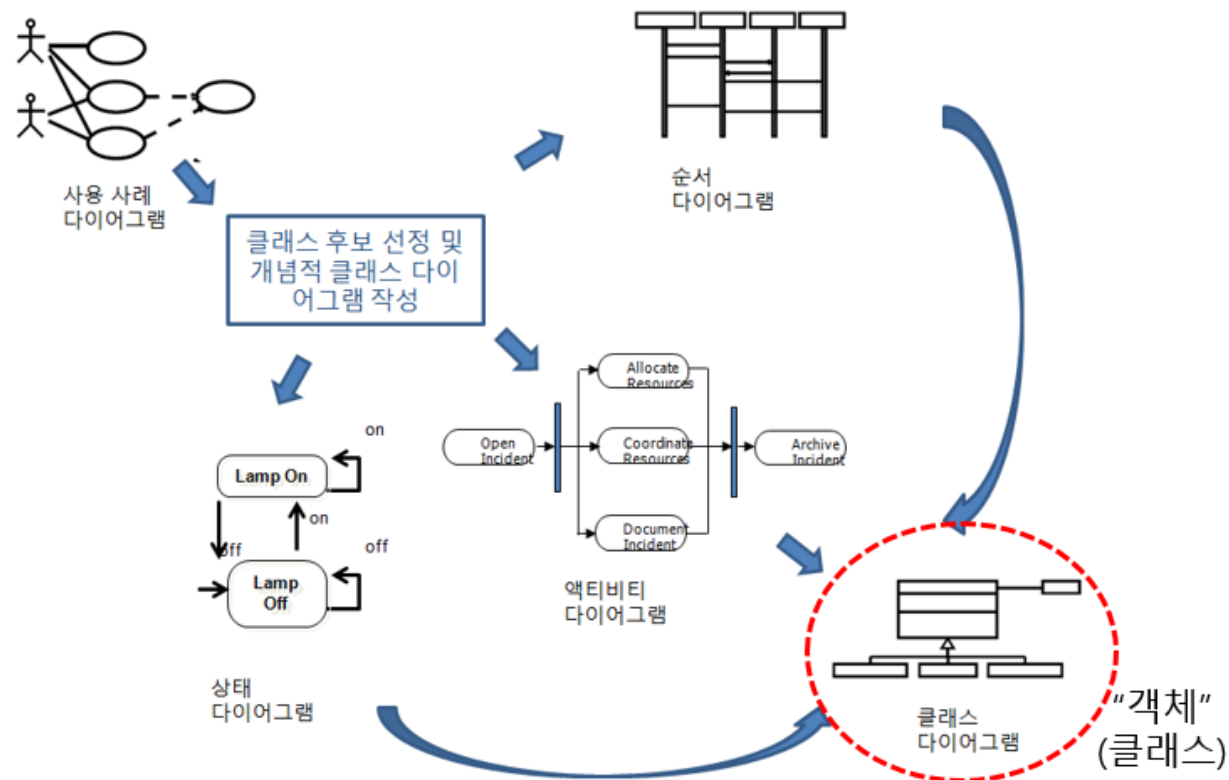
객체지향은 분석이나 설계 단계에 부분적 코딩 작업 가능

- 모델링 기초 개념이 같기 때문
- 개발 초기 단계부터 코딩 작업이 이루어질 수 있어 개발 기간 줄임
- 설계 기준: 소프트웨어가 시장에 출하되는 시점(Time-to-market)



객체지향 모델링 과정

뚜렷한 표준 프로세스보다는 경험적 요소에 의존적
시작은 사용사례, 최종 목표는 모형 작성



객체지향 모델링 과정



분석 단계

- 간단한 객체의 이름과 객체가 갖는 상태, 행위를 개념적으로 파악
- 사용자의 요구를 기능적 관점으로 파악하여 사용 사례 다이어그램 표현

설계 단계

- 객체의 상태를 구체적인 속성으로 정의
- 객체의 상태 변화를 행위(operation)로 정의
- 각각의 접근에 대한 제한을 구체화
- **요구 추출**: 사용 사례(Use case)로부터 기능 파악
- **요구 분석**: 추출된 요구 정리, 객체 사이 관계 정리, 모형 완성

객체지향 모델링 과정



요구 추출 작업

- 사용 사례 구체화: 시스템 명세 작성
- 사용 사례 관계 찾기: 시스템 명세 복잡성 줄이고 일관성 갖게 함
- 비기능적 요구 찾기: 시스템의 기능과 직접적 관련 없는 사항 찾음

요구 분석 작업

- 객체 찾기: 필요한 객체를 찾음
- 객체 사이 상호작용 모형화: 객체 사이의 관계를 정리
- 분석 모형 작성: 다이어그램 작성을 통해 모형을 완성
- 객체지향 설계: 분석 모형을 시스템 설계 모형으로 변환하는 과정

객체지향 모델링 언어



UML (Unified Modeling Language)

- 객체지향 시스템을 모델링하는 표준 그래픽 언어
- 시스템의 여러 측면을 그림으로 모델링
- 클래스 기술(정의)뿐 아니라 클래스 간 상호작용을 표현
- 하드웨어의 회로도 같은 의미

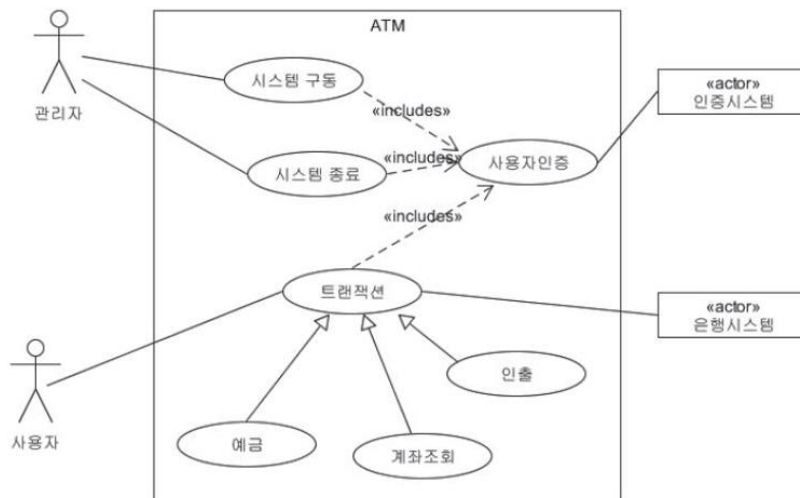
UML 다이어그램

- 기능적 모형 (Functional model): 사용자 측면에서 본 시스템 기능 표현
- 객체 모형 (Object model): 객체, 속성, 행위, 관계 등 시스템 구조 표현
- 동적 모형 (Dynamic model): 시스템 내부 동작 표현
 - 예) 메시지 교환, 상태 변화, 시스템의 제어 흐름 등

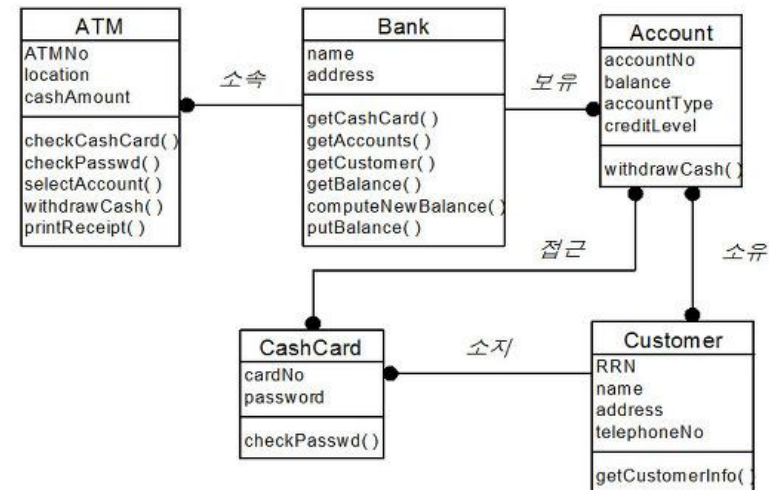
객체지향 모델링 언어



UML 다이어그램의 예



사용 사례 다이어그램



클래스 다이어그램



수업 내용

1. 수업 목표
2. 소프트웨어 개발 패러다임
3. 객체지향 모델링
4. 객체지향 개념 ←



객체 (Object)

정의

- 소프트웨어 모듈 (객체) = 자료구조 + 함수
- 객체들은 메시지 (message) 전달을 통해서 서로 간 상호 작용

객체의 구성

- 상태: 특징값, 속성
- 행위: 연산 (operation)을 수행할 수 있는 동작
- 정체성: 구별 가능성

클래스

- 비슷한 객체들의 구조와 행동을 클래스로 선언



클래스(Class)

정의

- 클래스는 객체들이 갖는 속성과 연산을 정의하는 틀(template)
- 클래스는 객체를 만드는 설계도
- 예) 직원 클래스
 - 속성: 이름, 직위, 월급, 전화번호 등
 - 함수: 진급, 월급 인상, 전화 변경 등

클래스 vs 인스턴스

- 클래스: 객체의 타입
- 인스턴스(instance): 클래스로부터 만들어지는 각각의 객체
 - 예) 도면 = 클래스, 제품 = 객체

추상화 (Abstraction)



정의

- 객체의 관계 (relation)를 분석하여 공통된 속성과 기능을 묶음
- 객체지향 관점에서 클래스를 정의하는 것을 추상화라 정의
 - 클래스: 추상 자료형
 - 객체: 추상 자료형의 인스턴스
 - 메소드: 추상 자료형에 정의된 연산
 - 메시지: 메소드의 호출

목적

- 추상화는 기계에서 일이 수행되는 구체적이고 상세한 것을 모르고도 컴퓨터의 수행작업을 쉽게 이해하도록 해줌

캡슐화 (Encapsulation)



정의

- 관련된 데이터와 연산들을 모아서 캡슐 안에 담음
- 예) 학사 관리 시스템에서 학생
 - 데이터: 학번, 이름, 주소
 - 함수: 평점 계산, 주소 변경, 수강 신청

정보 은닉 (Information hiding)

- 외부의 직접적 접근 불가
- 접근 제어를 위한 키워드 (public, private, protected)

```
class class_name {  
    private:  
        datatype member_variables;  
        datatype member_functions;  
    public:  
        datatype member_variables;  
        datatype member_functions;  
};  
  
main() {  
    class_name objectname1, objectname;  
}
```

상속(Inheritance)

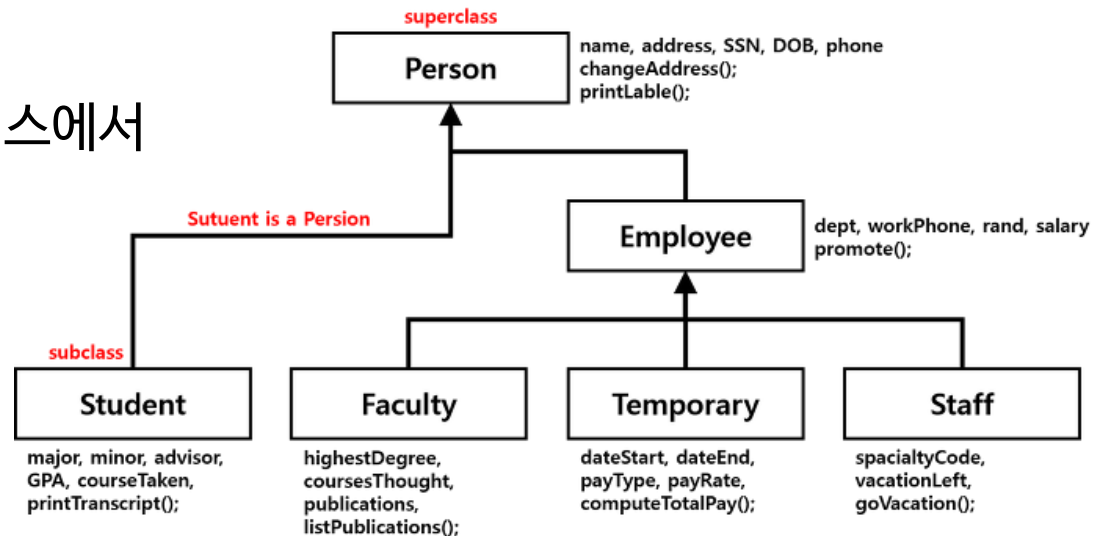


정의

- 상위 클래스의 속성과 연산을 물려 받음
- 슈퍼클래스(superclass): 상속되는 클래스, 부모 클래스
- 서브클래스(subclass): 상속받는 클래스, 자식 클래스

다중(multiple) 상속

- 두개 이상의 슈퍼클래스에서 상속 받음



다형성 (Polymorphism)



정의

- 객체가 취하는 동작이 상황에 따라서 달라짐을 의미
- 여러 형태를 표현 가능

오버로딩 (Overloading)

- 하나의 클래스에서 같은 이름의 여러 함수 존재
- 같은 이름의 함수라도 매개변수로 구분함

오버라이딩 (Overriding)

- 상속 받은 함수를 서브클래스 유형별로 재정의
- 서브 클래스 유형에 따라 구분



향후 일정



주차	내용	
1	객체지향 개요 (lecture note)	
2	클래스 (lecture note)	프로젝트 팀 1차 신청
3	객체 (Chap. 1)	프로젝트 팀 구성 완료
4	객체 생성과 활용 (Chap. 2)	
5	추상화 및 정보 은닉 (Chap. 4, Chap. 5)	
6	상속과 구성 (Chap. 14)	프로젝트 제안서 제출
7	다형성과 가상함수 (Chap. 15)	
8	중간고사	

질문 및 답변

