소프트웨어 공학개론

Introduction to Software Engineering

2022-1학기 (Spring)
선문대학교 AI소프트웨어학과

4월 21일 (목)
4월 26일 (화)

강의 목차

- 6. 객체지향분석
  - 6.1 객체지향분석이란?
  - 6.2 객체지향의 기본개념
  - 6.3 객체지향개발방법론
  - 6.4 객체지향분석의 순서
- 연습 문제

6.1 객체지향분석이란?

선문대학교

목차

6.1 객체지향분석이란?

개요

- 객체지향분석 (OOA : Object-Oriented Analysis)
  - 시스템이 무엇을 하는지에 관한 기능을 중심으로 하지 않고, 데이터와 그에 대한 처리를 일체화시킨 객체에 주목하여 객체들의 구조와 동작을 명확하게 함으로써 고객과 이용자의 요구사항을 사양화하는 기법
  - 현실세계의 모델을 소프트웨어로 직접 표현하는 것을 목적으로 객체를 구성 기본단위로하여 소프트웨어를 구축하는 체제

  - 특징
    - 인간의 인지방법에 가능한 한 유사하게 적용시킨 기법
    - 실세계의 시스템을 직감적으로 표현하기 수월함
    - 소프트웨어개발에 있어 일관되게 객체를 중심으로 생각할 수 있음
    - 분석, 설계, 구현단계의 각 성과물들이 유연하고 일관성있게 이행가능함

6.1 객체지향분석이란?

선문대학교

6.1.1 객체지향의 개요

- 1960년대 등장했던 시뮬레이션용 언어 Simula로부터 기본개념 도입
- 1970년대 등장했던 Smalltalk에 의해 널리 전파
  - 객체지향을 도입한 프로그래밍 언어 : C++, Java, C#, Ruby 등
- 1980년후반~1990년대 다수의 객체지향개발방법론 제안
- 현재에는 UML이라는 모델표기법이 통일됨

- 인지과학 관점에서의 객체지향에서 인간이 사물과 개념을 인식하거나 전달하는 경우 3가지 개념 사용
  - (1) 내포 : 기능에 의한 인지. 무엇을 하는가
    - 내포 기반 기법: 구조화분석에서 시스템의 기능을 단계적으로 상세화
  - (2) 외연 : 분류에 의한 인지. 무엇과 유사한가
  - (3) 속성 : 구조에 의한 인지. 무엇으로 구성되어 있는가
    - 속성 기반 기법: 시스템의 구조를 ERD에 의해 명확하게 함

6.1 객체지향분석이란?

선문대학교

6.1.1 객체지향의 개요

- 1960년대 등장했던 시뮬레이션용 언어 Simula로부터 기본개념 도입
- 1970년대 등장했던 Smalltalk에 의해 널리 전파
  - 객체지향을 도입한 프로그래밍 언어 : C++, Java, C#, Ruby 등
- 1980년후반~1990년대에는 다수의 객체지향개발방법론 제안
- 현재에는 UML이라는 모델표기법이 통일됨

- 인지과학 관점에서의 객체지향에서 인간이 사물과 개념을 인식하거나 전달하는 경우 3가지 개념 사용
  - (1) 내포 : 기능에 의한 인지. 무엇을 하는가
    - 구조화분석에서 시스템의 기능을 단계적으로 상세화
  - (2) 외연 : 분류에 의한 인지. 무엇과 유사한가
  - (3) 속성 : 구조에 의한 인지. 무엇으로 구성되어 있는가
    - 시스템의 구조를 ERD에 의해 명확하게 함

6.1 객체지향분석이란?

선문대학교

6.1.1 객체지향의 개요

- 그림 예: 온라인판매업무에 대한 현실세계와 객체지향 분석에 의한 모델
  - 현실세계에 등장하는 사물과 개념이 객체지향모델의 객체에 각각 대응
  - 내포(기능에 의한 인지): "주문자"가 무엇을 하는지를 찾아냄
    - 주문자는 프린터와 디지털카메라를 구입할 수 있음
  - 외연(분류에 의한 인지): "프린터"와 "디지털카메라"의 공통점을 찾아내는 것
    - 공통점: 모두 판매상품임
  - 속성(구조에 의한 인지): "프린터"의 구조에 주목하여 프린터의 구성부품과 특성(크기와 무게)을 고려하는 것

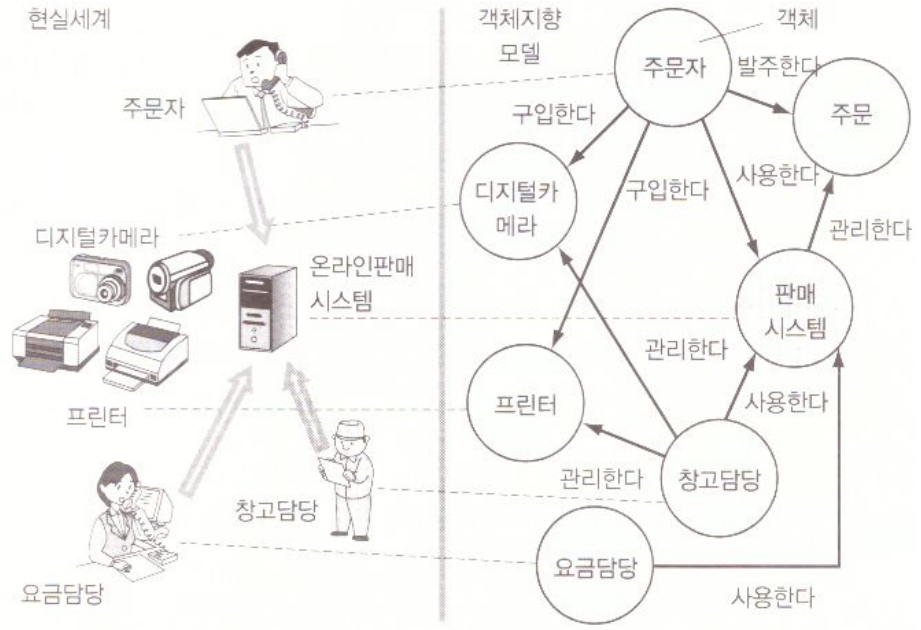

그림 6.1 | 현실세계와 객체지향모델의 예

6.1 객체지향분석이란?

6.1.1 객체지향의 개요

- 객체
  - 인간이 인지할 수 있는 구체적 또는 추상적인 “사물”/“개념”

  - 객체지향에서는 내포, 외연, 속성의 모든 측면에서 포착된 이미지를 “객체(조작의 대상이 되는 데이터)”로 표현하고, 객체를 중심으로 시스템 구축
    - 기능, 분류, 구조의 모든 관점에서 시스템을 분석하여 소프트웨어로 표현
  - 인간의 인지방법에 가능한 한 근접시킨 기법
    - 객체지향에 의해 구축된 모델은 인간에게 있어 이해가 수월함

6.1.2 객체

- 소프트웨어의 주요구성요소
- 사람이나 책과 같은 물리적인 것뿐만아니라, 역할이나 개념도 포함
  - 역할 : 상대적인 입장
    - 주문자와 관리자는 동일인물일 수 있음
    - 한 인물이 어떤 온라인판매에서는 주문자 역할, 또 다른 온라인판매에서는 관리자 역할을 수행
    - 객체지향에서는 역할들을 각각 서로다른 객체들로서 인식할 수 있음

6.1.2 객체

- 객체의 성질
  - (1) 상태
    - 객체의 현재 모습
    - 상태는 형태나 색 등과 같이 시간이나 조건에 의해 변화함
      - 예: “로그인 중” 등과 같이 현재의 상황도 상태임
  - (2) 동작
    - 객체가 실행할 수 있는 동작
    - 외부에 제공하는 서비스뿐만아니라 내부의 상태를 변화시킬 수 있는 동작도 포함
  - (3) 식별성
    - 어떤 객체가 다른 객체와 명확하게 구별될 수 있는 성질
    - 객체지향분석에서는 객체의 속성과 동작을 정의하는 것이 주된 작업임
      - 속성(Property) : 객체의 상태
      - 메소드(Method) : 객체의 동작
    - 예 : 2개의 객체가 상태와 동작이 똑같다 하더라도 2개의 객체는 서로 다른 객체임
      - 식별자를 부여함으로써 각각의 객체를 구별

6.2 객체지향의 기본개념

개요

- 객체지향분석을 이해하기 위해서 필요한 기본개념들을 설명

목차

6.2 객체지향의 기본개념

6.2.1 캡슐화(Encapsulation)

- 데이터와 이를 다루기 위한 처리를 일체화시킨 추상데이터형의 개념을 토대로 하는 모듈화
- 모듈분할의 평가기준에 있어서 정보적 응집도에 해당
- 그림 예 : 객체와 캡슐화
  - “상품명”, “제조사” 등 현실세계에서 프린터가 소유하고 있는 데이터 →프린터 객체의 속성으로서 보존됨
  - “가격을 알아낸다”, “가격을 변경한다” 등 → 프린터 객체의 속성에 대한 처리를 수행하는 동작 정의

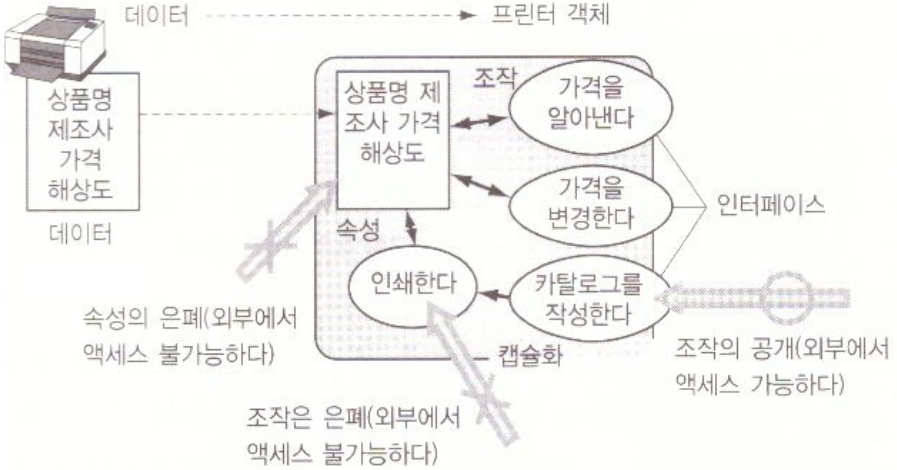

그림 6.2 상용 컴퓨터의 진화와 소프트웨어위기

6.2.1 캡슐화(Encapsulation)

- 정보은폐
  - 캡슐화의 최대 장점
  - 객체의 속성은 캡슐화에 의해 외부로부터 은폐시킬 수 있음
    - 외부로부터 액세스할 수 없게 지정가능
    - 만약 외부로부터의 액세스가 필요한 경우 전용 조작(액세스 메소드)을 준비하도록 함
      - 액세스 메소드를 통해 속성에 액세스함으로써 객체는 액세스가 타당한 것인지 검사 가능
      - 예: 어떤 속성에 액세스하기 전에 액세스권한과 패스워드를 검사하는 처리를 추가
        - 속성이 부당하게 읽혀지거나 변경되는 것을 방지할 수 있음
  - 캡슐화를 사용함으로써 외부에 공개하는 메소드와 내부에 은폐시키는 메소드로 분리 가능
    - 인터페이스 : 외부에 공개하는 메소드의 집합
      - 객체가 어떤 동작을 할 수 있는지를 규정
    - 외부에서는 객체의 인터페이스만 살펴볼 수 있기 때문에 해당 객체를 이용하는 개발자는 객체 내부의 속성과 메소드에 관한 구현내용을 알고 있을 필요가 없음
    - 인터페이스와 처리내용을 변경하지 않는한 데이터구조와 알고리즘 등의 내부구현을 해당객체의 개발자가 자유롭게 변경가능

6.2.1 캡슐화(Encapsulation)

- 정보은폐
  - 그림 예: “인쇄”를 비롯한 다른 3개의 메소드가 외부에 공개됨
    - 다른 객체들은 이러한 메소드 호출 가능
      - 프린터의 가격을 알고 싶은 경우, “가격을 알아낸다”라는 메소드를 호출하면 됨

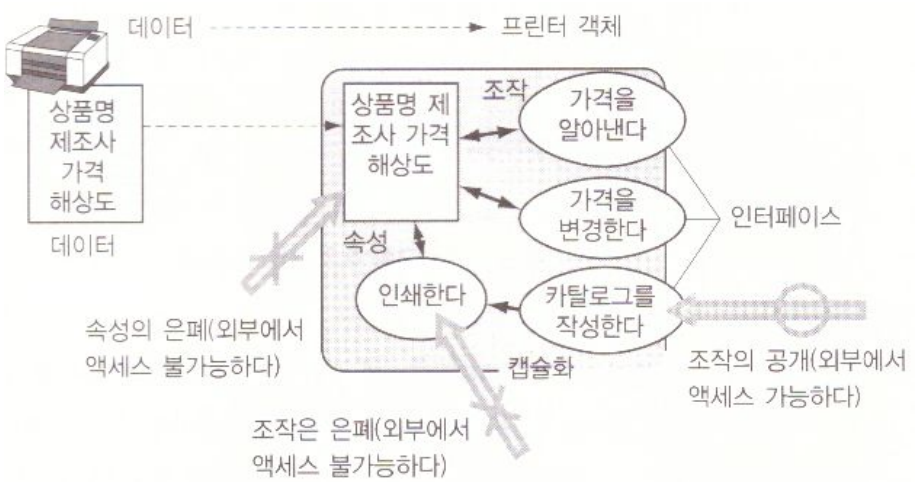

그림 6.2 상용 컴퓨터의 진화와 소프트웨어위기

6.2.2 메시지전송(Message Sending)

- 각 개체들에게 처리를 의뢰하는 체계
- 객체에 대한 메소드 호출
  - 호출된 객체는 속성의 값을 변경(상태 변화) or 속성값과 계산결과를 반환 or 다시 다른 객체를 호출 등과 같은 동작 수행
  - 그림 예 : 메시지전송
    - "상품일람표작성" 메소드 호출: 창고담당자가 상품관리객체에게 메시지 전송
      → "표지작성" 메소드 호출: 메시지를 수신한 상품관리객체는 표지 작성
      → "카탈로그작성" 메시지 호출: 각각의 프린터 객체들에게 메시지 전송
      → "인쇄" 메소드 호출: 메시지를 수신한 프린터객체들은 각자 가지고있는 프린터정보 인쇄

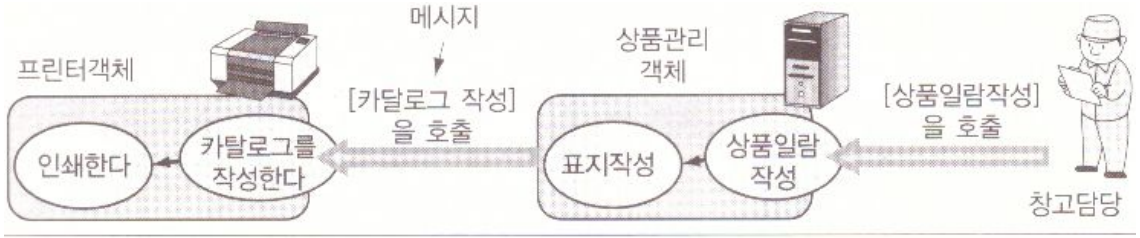

그림 6.3 | 메시지전송

6.2.2 메시지전송(Message Sending)

- 각 개체들은 자신의 책임으로 처리가능한 동작만을 수행
  - 자신의 책임이 아닌 동작에 관해서는 다른 객체들에게 의뢰(위임)
  - 의뢰에 의한 메시지 전파로 소프트웨어내부의 객체들을 서로 협조하여 최종목표 달성

6.2.3 클래스(Class)와 인스턴스(Instance)

- 클래스(Class)
  - 공통적인 속성과 동작을 갖는 객체들을 추상화함으로써 객체들의 본질을 기술한 모형(템플릿, template)
  - 그림 예 : 2대의 프린터에 대응하는 객체를 프린터클래스로 추상화 가능
    - 상품명, 제조사, 가격, 해상도가 서로 다른 프린터 2대는 전송받은 데이터를 인쇄한다는 점이 같음(공통)
    - 프린터객체가 소유하고 있는 공통적인 속성들과 동작들은 프린터클래스의 속성과 동작이 됨
    - 클래스는 3개의 구역을 갖는 사각형 으로 표현
      - 최상단 구역: 클래스이름
      - 중간 구역: 속성
      - 최하단 구역: 동작(메소드)

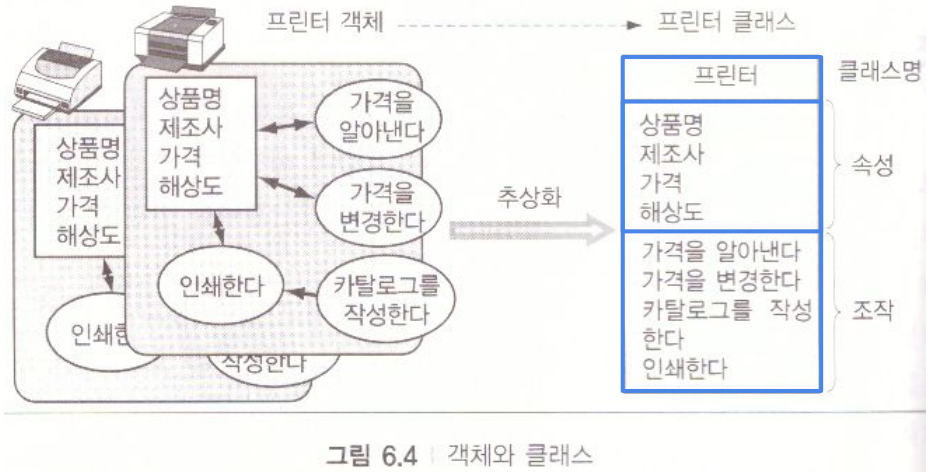

그림 6.4 | 객체와 클래스

6.2.3 클래스(Class)와 인스턴스(Instance)

- 인스턴스(Instance)
  - 클래스로부터 생성된 객체
    - 인스턴스화(Instantiation): 클래스로부터 객체를 생성하는 것
    - 클래스는 객체의 설계도 역할을 하므로 클래스를 사용하여 새로운 객체를 생성 가능
    - 예: 한번 프린터클래스를 작성해두면 프린터클래스로부터 새로운 프린터객체를 언제든지 생성 가능
    - 객체라는 단어는 현실세계에 등장하는 “사물”을 지칭하므로 클래스를 객체(클래스객체)로 취급하는 경우도 있음

6.2.3 클래스(Class)와 인스턴스(Instance)

- 클래스와 객체의 관계
  - 인스턴스가 갖는 성질은 클래스에 의해 정의됨
  - 같은 클래스로부터 생성된 인스턴스들의 속성과 동작은 서로 같음
  - 서로 다른 인스턴스들은 독립적으로 취급할 수 있으며 서로 다른 속성값(내부상태)을 가질 수 있음

- 객체지향에서는 클래스를 중심으로 소프트웨어를 구축
  - 개발자는 분석에 의해 클래스와 클래스들 사이의 관계를 추출하고 각 클래스들을 설계, 구현
  - 소프트웨어 실행시 클래스들로부터 인스턴스들이 생성
  - 인스턴스들이 메시지들을 송수신함으로써 계산을 수행

6.2.4 관련(Association)

- 관련(Association) : 링크를 추상화한 클래스들 사이의 이용관계
  - 링크(Link) : 객체들 사이의 이용관계

- 관련관계
  - 객체가 메시지전송에 의해 다른 객체들과 협조할 때 어떤 객체가 다른 객체들을 이용하는 관계
  - 관련관계에는 방향성이 있으므로 관련관계의 시작점과 도착점(어떤 것이 주체가 되는지)에 주의 필요
  - 관련관계는 2개의 클래스를 직선으로 연결하는 것으로 표현됨

6.2.4 관련(Association)

- 그림 예
  - 주문자 객체는 프린터객체에 대해 “구입한다”라는 링크를 가짐
  - 임의의 주문자가 임의의 프린터를 구입할 가능성을 추상화하면 주문자클래스와 프린터클래스 사이에는 “구입한다”라는 관련관계가 성립

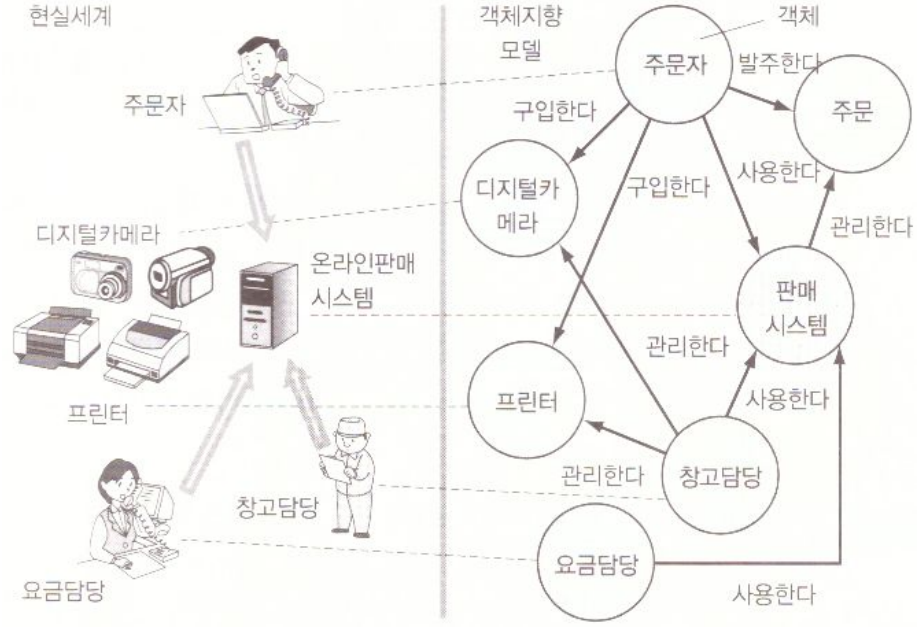

그림 6.1 | 현실세계와 객체지향모델의 예

6.2.5 계승(상속, Inheritance)

- 공통적인 성질을 갖는 여러 개의 클래스들에서 공통성질을 공유시킴으로써 쓸모없는 중복을 삭제하는 체계
- 그림 예: 프린터클래스와 디지털카메라클래스의 계승(상속)관계 구축
  - (둘다 판매대상 상품인) 프린터클래스와 디지털카메라클래스에 공통적인 속성과 동작들을 묶어 새롭게 작성한 판매상품클래스로 이동
    - 이동 후 프린터클래스에는 "해상도" 속성이 남게 됨
    - 이동 후 디지털카메라클래스에는 "화소수" 속성이 남게 됨
    - 메소드 "인쇄한다"는 메소드이름이 같지만 인쇄처리내용이 서로 다르므로 각 클래스에 남겨둠

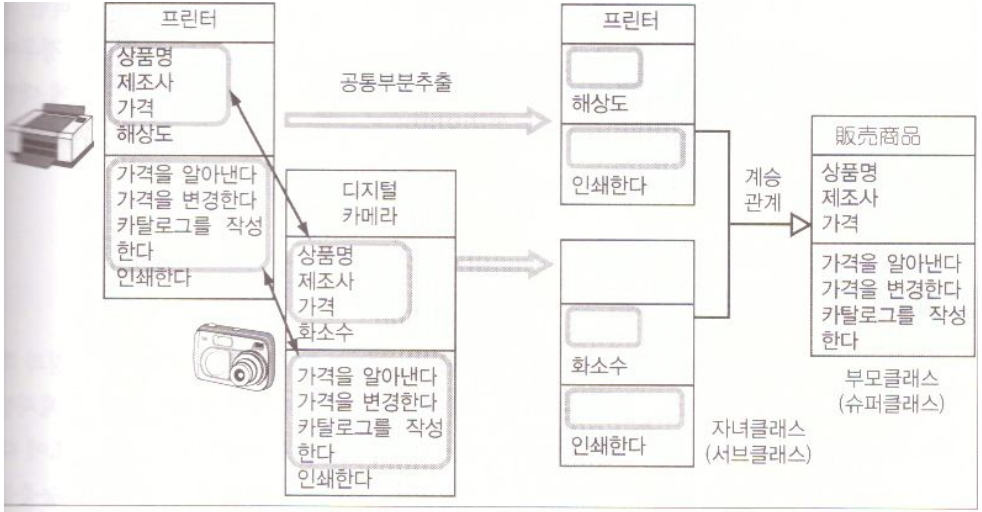

그림 6.5 | 계승관계

6.2.5 계승(상속, Inheritance)

- 계승(상속)관계는 삼각형 화살표(⟶)로 표현
- 그림 예
  - 판매상품 클래스: 프린터클래스와 디지털카메라클래스의 부모클래스(또는 슈퍼클래스, Superclass)
  - 프린터클래스와 디지털카메라클래스: 판매상품클래스의 자식클래스(또는 서브클래스, Subclass)

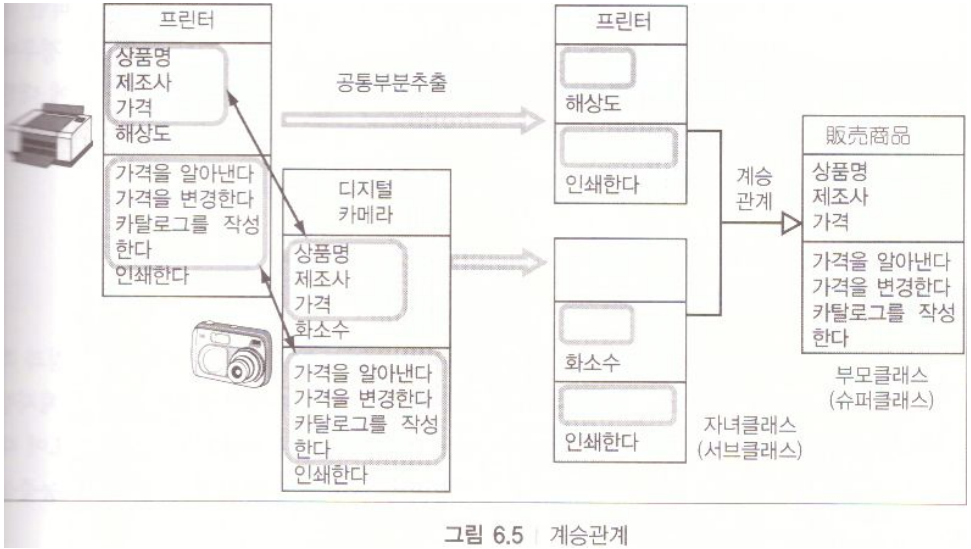

그림 6.5 | 계승관계

- 자식클래스 :부모클래스에서 정의된 속성과 동작을 물려받아서 자신의 속성과 동작으로 취급가능
  - 자식클래스로부터 생성된 인스턴스는 자식클래스뿐만아니라 부모클래스의 속성과 동작을 소유
- 계승(상속)은 추이적인 성질을 가짐
  - 자식클래스의 자식클래스도 또한 자식클래스(자손클래스)
  - 부모클래스의 부모클래스도 부모클래스(선조클래스)

6.2.5 계승(상속, Inheritance)

- 계승(상속)을 이용할 때의 주의사항
  - 계승(상속)은 클래스들 사이에 정적으로 밀접한 관계 형성
    - 모듈의 독립성을 높인다는 관점에서 불필요하게 계승(상속)을 이용하지 않는 것이 좋음
    - 클래스의구현만을 재이용하거나 성질의 일부분만을 계승(상속)할 목적으로는 계승(상속)을 이용하지 않는 것이 좋음
  - 계승(상속)에서 자식클래스는 부모클래스(의 일종)이며, 부모클래스의 모든 성질을 물려받음
    - 예: 프린터클래스는 판매상점클래스의 모든 성질(속성과 메소드)을 물려받으므로 판매상품클래스와 같은 동작을 함
      - 판매상품클래스의 인스턴스 대신 프린트클래스의 인스턴스를 사용할 수 있음은 전제
      - 이와 같은 관계를 구축할 수 있거나 구축하고 싶은 경우에만 계승(상속) 이용

6.2.6 집약(집계, Aggregation)

- 전체와 구성부품들 사이의 관계 → 예: 프린터는 여러 개의 부품으로 구성되어 있음
- 여러 개의 서로 다른 부품객체들을 묶어서 취급하는 체계

- 계승(상속)관계는 정적(프로그램 작성시)으로 결정되지만,
  집약(집계)관계는 동적(소프트웨어 실행시)으로 전환 가능

- 집약(집계)관계는 마름모를 정점으로 하는 화살표로 표현
  - 마름모가 붙은 쪽이 전체 → 그림에서는 프린터클래스가 전체

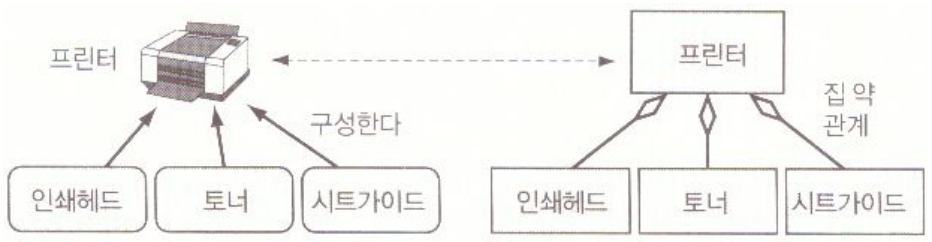

그림 6.6 집약관계

6.2.6 집약(집계, Aggregation)

- 합성(구성, Composition) 관계
  - 집약(집계)관계에서 전체객체와 부품객체에 강한 소유관계가 있는 경우를 말함
    - 강한 소유관계: 부품객체가 다른 부품전체객체에 동시에 공유됨없이 생존기간이 일치하는 것
  - 그림 예
    - 집약(집계)관계: 인쇄헤드와 토너가 교환가능하다면 프린터와 생존기간이 일치하지 않음
    - 합성(구성)관계: 인쇄헤드와 토너가 교환불가능한 일체형 프린터인 경우 프린터를 폐기하는 것과 동시에 인쇄헤드도 폐기됨

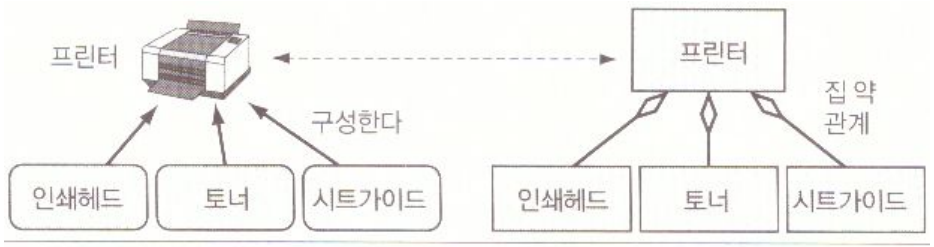

그림 6.6 집약관계

6.3 객체지향개발방법론

선문대학교

개요

- 객체지향개발방법론
  - 객체지향을 토대로 소프트웨어를 개발하는 공정과 객체지향소프트웨어의 모델화 기법(표기법)

목차

6.3.1 객체지향개발공정

- 반복형 소프트웨어개발 공정 사용
  - 이유 : 객체지향에서는 각 단계에서의 작업이 유연하게 수행될 수 있음
    - 객체지향에서는 개발의 모든 각 단계에서 객체(또는 클래스)에 초점
    - 각 공정에서 작성되는 성과물은 상세 정도가 서로 다를 뿐, 기법을 포함하여 모두 같은 것임

- 반복형 소프트웨어개발 공정은 아래 2가지 공정을 혼합시킨 것
  - 반복적 소프트웨어개발 : 분석, 설계, 구현, 평가의 각 단계를 연속적으로 반복하여 수행함으로써 개발 진행
  - 점증적 소프트웨어개발 : 소프트웨어를 여러 개의 부분으로 나누어서 조금씩 단계적으로 개발 진행

6.3.2 UML(Unified Modeling Language)

- 통일된 모델 표기법
- 개발공정과 독립되어 있음
- 객체지향개발에서는 UML의 도면들 중 필요한 도면만을 실제의 개발에서 도입하여 사용하면 됨

표 6.1 UML의 도면들

측면	도면	설명
구조	Class Diagram	클래스의 구조(속성과 동작)와 클래스들 사이의 정적인 관계
	Object Diagram	어떤 시점에서 객체의 상태와 객체들간의 관계
	Package Diagram	패키지의 구성과 패키지들 사이의 의존관계
	Composite Structure Diagram	실행시의 클래스 내부구조
	Component Diagram	컴포넌트의 구조와 의존관계
	Deployment Diagram	시스템에서의 물리적 배치
동작	Use Case Diagram	시스템에서 제공하는 기능과 이용자 사이의 관계
	Activity Diagram	작업의 순서와 병행성
	State Machine Diagram	객체의 상태와 이벤트에 의한 상태천이
	Sequence Diagram	객체들 사이의 상호작용(시간흐름 중심)
	Communication Diagram	객체들 사이의 상호작용(객체들사이의 관계 중심)
	Timing Diagram	객체의 상호작용 타이밍
	Interaction Overview Diagram	Sequence Diagram과 Activity Diagram의 개요

6.3.3 객체지향의 잇점

- 높은 변경용이성 및 확장성

- 그림 예: 기능중심기법과 객체중심기법의 차이점
  - 그림(a) 기능중심기법
    - 상품관리시스템의 기능으로서 "카달로그작성"을 추출하여 함수로 구현
    - 기능에 따라 모듈분할을 수행→일반적으로 한 개의 함수가 다양한 데이터구조를 다룸
    - 그림(a)에서 프린터의 데이터구조에 변경이 발생하였다면 "카탈로그작성"함수도 변경해야 함
      - 문제점: "카탈로그작성"함수는 프린터의데이터만을 취급하는 것이 아님 → 프린터의 데이터구조만 변경하면 된다는 것을 알지만, "카탈로그작성"함수 전체를 조사하여 디지털카메라의 데이터 처리는 변경되지 않도록 변경작업 수행 필요

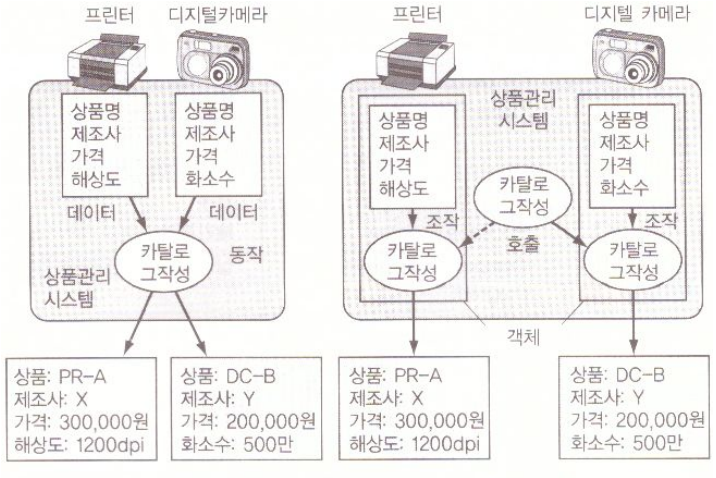

그림 6.7 | 기능중심기법과 객체중심기법에 의한 모델화의 비교

6.3.3 객체지향의 잇점

- 그림 예: 기능중심기법과 객체중심기법의 차이점
  - 그림(b) 객체중심기법
    - 프린터의 데이터 및 디지털카메라의 데이터를 처리하는 메소드는 각각의 클래스에 독립적으로 정의되어 있음
    - 상품관리시스템의 "카탈로그작성"메소드는 각 클래스의 "카탈로그작성"메소드를 호출하여 그림(a)와 같은 결과를 출력가능
    - 그림(b)에서 프린터의 데이터구조에 변경이 발생하였다면 프린터클래스의 "카탈로그작성" 메소드만 변경하면 됨
      - 디지털카메라클래스의 "카탈로그작성" 메소드는 프린터클래스의 "카탈로그작성" 메소드와 독립되어 있어 디지털카메라클래스를 조사, 변경 불필요

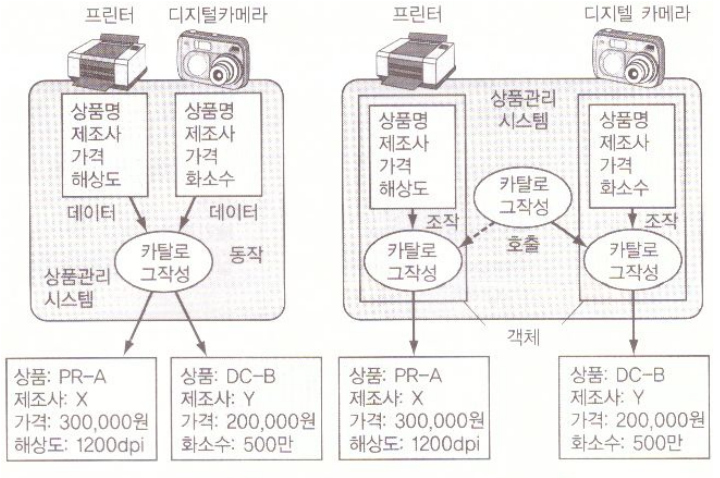

그림 6.7 | 기능중심기법과 객체중심기법에 의한 모델화의 비교

6.3.3 객체지향의 잇점

- 그림 예: 기능중심기법과 객체중심기법의 차이점
  - 상품관리시스템에 새로운 상품(예: TV)을 추가하는 경우
    - 그림(a): "카탈로그작성"함수를 확장해야 함
      - 새로 추가된 처리가 원래의 처리에 모순되지 않도록 작업 수행
    - 그림(b): 새롭게 TV클래스를 작성하고 내부에 TV에 관한 데이터만을 다루는 "카탈로그작성" 메소드를 기술하는 작업만 수행
      - 새로운 클래스의 추가에 의해 다른 클래스에 영향을 끼칠 가능성이 낮음

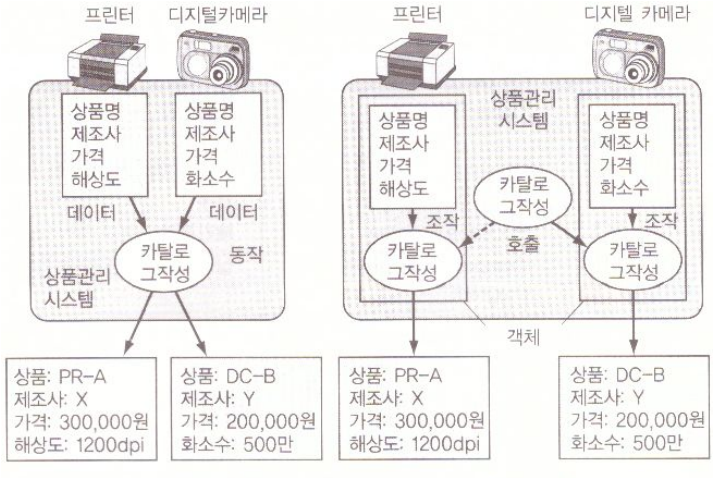

그림 6.7 | 기능중심기법과 객체중심기법에 의한 모델화의 비교

6.4 객체지향분석의 순서

선문대학교

개요

- 객체지향분석의 2가지 기법 중 책임구동형 기법을 기반으로 하는 객체지향분석에 대해 설명
  - 데이터구동형 기법: 시스템 내의 데이터를 우선적으로 인식
  - 책임구동형 기법: 시스템의 동작(책임)을 우선적으로 인식
    - UML을 사용한 최근의 객체지향분석에서 사용

목차

6.4 객체지향분석의 순서

6.4.1 유즈케이스(Use Case)의 추출

- Use Case 이란?
  - 이용자가 어떻게 시스템을 사용하는지를 나타내는 전형적인 사용사례

- Use Case Diagram 이란?
  - Use Case를 도면(Diagram)으로 표현한 것
  - 시스템의 각 기능마다 기술
  - Use Case Diagram 에서는 Use Case와 Actor의 관계를 명확히 함
    - Actor: 시스템과 상호작용하는 외부의 실체(인간, 장치 등)가 수행하는 역할

6.4 객체지향분석의 순서

6.4.1 유즈케이스(Use Case)의 추출

- Use Case Diagram 을 기술할 때
  - 우선, 시스템의 경계를 명확하게 함
  - 사각형: 시스템의 경계를 나타내며,
    시스템경계의 바깥쪽이 시스템의 외부가 됨
    - 그림 예: 주문자, 요금담당자, 창고담당자는 시스템의 외부에 존재함
  - 타원 : Use Case
  - 사람모양의 아이콘(Stickman이라고 부름): Actor(Initiator라고도 부름)

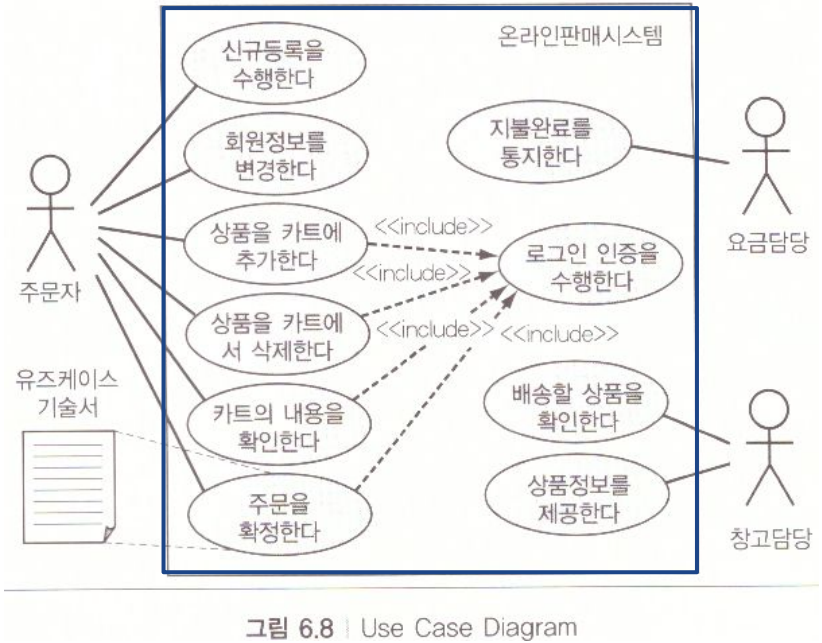

그림 6.8 | Use Case Diagram

6.4 객체지향분석의 순서

6.4.1 유즈케이스(Use Case)의 추출

- Use Case Diagram 을 기술할 때
  - 직선으로 연결: Use Case와 Actor 사이에 관계가 있는 경우
  - 그림 예
    - 주문자는 직선으로 연결된 6개의 Use Case를 실행가능
    - 요금담당자에 대해서는 직선으로 연결된 1개의 Use Case를 실행가능
    - 창고담당자는 2개의 Use Case를 실행가능
    - 그러나, 직선으로 연결되어 있지 않은 Use Case를 실행하는 것은 불가능
      - 온라인판매시스템에 대해서 주문자가 "지불완료를 통지한다"라는 Use Case를 실행하는 것은 허용되지 않음

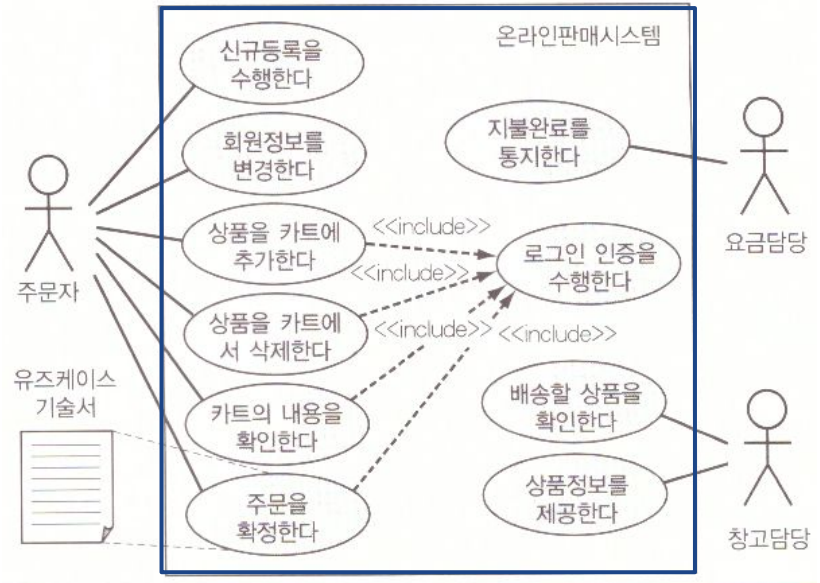

그림 6.8 | Use Case Diagram

6.4 객체지향분석의 순서

6.4.1 유즈케이스(Use Case)의 추출

- Use Case Diagram 에는 Use Case 사이 또는 Actor 사이의 관계를 기술가능
  - 그림 예: <<include>>
    - 어떤 Use Case가 다른 Use Case를 포함하여 이용하는 것
    - “주문을 확정한다”라는 Use Case: 내부에서 “로그인인증을 수행한다”라는 Use Case 실행

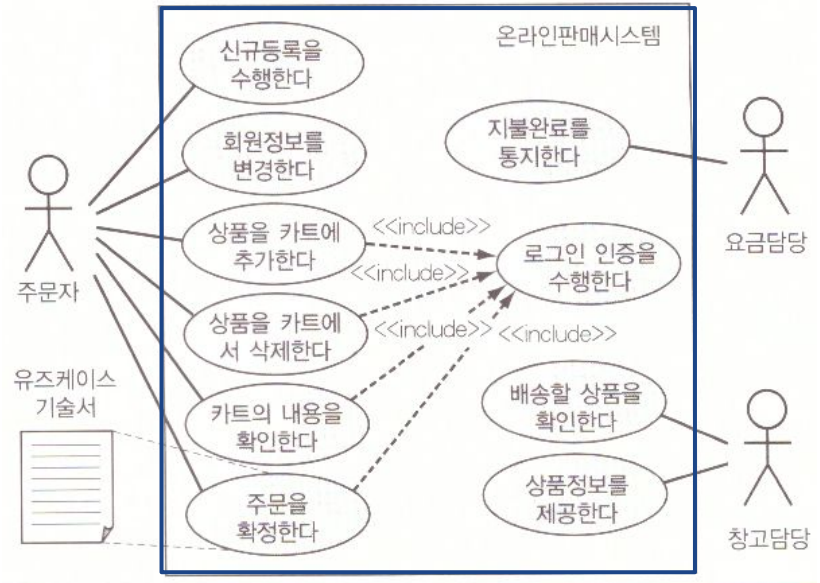

그림 6.8 | Use Case Diagram

6.4 객체지향분석의 순서

6.4.1 유즈케이스(Use Case)의 추출

- 유즈케이스 기술서(Use Case Description)
  - Use Case에 대하여 정해진 사용기법을 정의한 것
  - 일반적인 시나리오형식으로 기술

  - 그림 예: "주문을 확정한다"라는 Use Case에 대한 기술서(Use Case Description)
    - 이용자의 조작이 정상적으로 실행된 경우뿐만아니라 이용자의 조작이 실패한 경우에 대한 예외처리에 대해서도 기술됨
    - Use Case를 실행하기 전에 성립해야 되는 조건을 사전조건으로 기술 가능
    - Use Case의 실행 후에 성립되어야 하는 조건을 사후조건으로 기술 가능

명칭	주문을 확정한다.
시작Actor	주문자
목적	카트에 존재하는 상품을 구입한다.
사전조건	주문자는 로그인 인증에 성공해야 한다.

정상처리 시나리오

1. 주문자가 주문확정버튼을 누른다.
2. 시스템은 카트 속의 상품들에 대한 합계금액을 계산한다.
3. 시스템은 카트의 내용과 합계금액을 표시한다.
4. 주문자가 신용카드번호를 입력하고 구입버튼을 누른다.
5. 시스템은 요금담당자에게 주문자의 지불상황을 확인한다.
6. 지불이 성공하면, 지불완료를 통지한다.
7. 시스템은 창고담당자에게 상품의 발송을 의뢰한다.
8. 시스템은 카트를 비운다.
9. 시스템은 상품의 재고를 갱신한다.

예외처리

5에서 지불이 실패한다.

a. 시스템은 지불실패 메시지를 표시한다.

그림 6.9 | 유즈케이스 기술서

6.4 객체지향분석의 순서

6.4.2 Class Diagram의 작성

- Use Case와 요구사양으로부터 시스템에 등장하는객체를 찾아내서 클래스 추출
  - 클래스는 시나리오와 요구사양에 나타나는 명사에 대응하는 경우가 많음
- 이후 추출된 클래스들 사이에 존재하는 관련관계를 찾아내서 대략적인 Class Diagram을 작성
  - Class Diagram : 클래스의 구조와 클래스들 사이에 성립하는 관계(관련, 계승(상속), 집약(집계))를 정리하기 위해 사용
    - “개념모델”이라고 부름
  - 관련관계는 시나리오와 요구사양에 나타나는 동사에 대응하는 경우가 많음

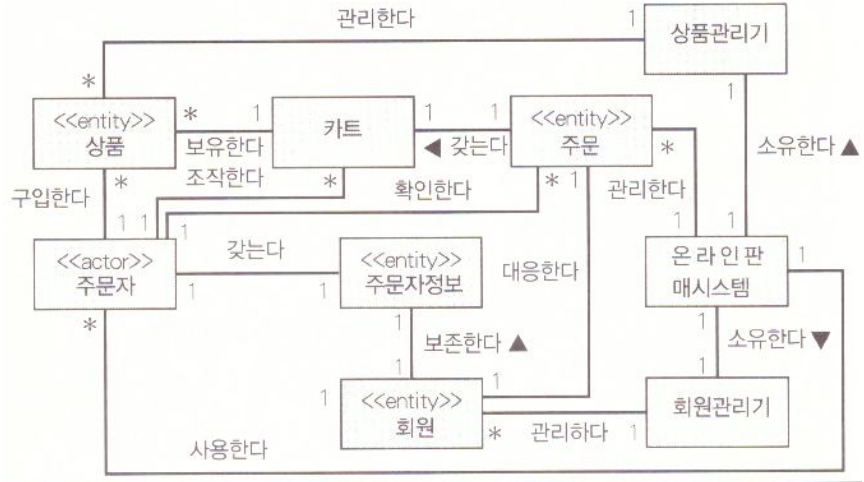

그림 6.10 온라인판매시스템의 개념모델

6.4.2 Class Diagram의 작성

- 그림 예: 개념모델의 일부분(주문자에 관련있는 부분만 표현)
  - <<actor>>: Use Case Diagram의 Actor
  - <<entity>>가 표시된 클래스: 데이터를 보존하는 역할
  - 클래스들 사이의 직선: 관련관계
    - 검은색 삼각형: 관련관계의 방향
  - 관련관계의 양쪽 끝에는 다중도를 표시
    - 다중도: 1개의 관련관계(링크)에 대해 각 클래스의 인스턴스가 몇 개씩 결합되어 있는지를 나타냄
    - 다중도 "1": 1개의 인스턴스가 결합
    - 다중도 "*": 0개 이상의 인스턴스가 결합

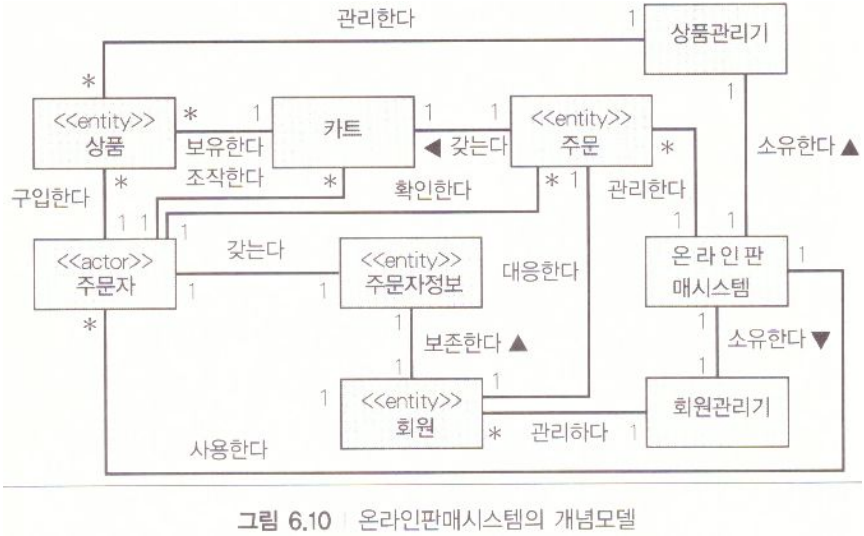

그림 6.10 온라인판매시스템의 개념모델

6.4 객체지향분석의 순서

선문대학교

6.4.2 Class Diagram의 작성

- 그림 예: 주문자클래스와 주문자정보클래스의 관련관계
  - 각각의 다중도는 “1”: 1개의 링크에 대해서 주문자클래스의 인스턴스와 주문자정보클래스의 인스턴스는 1개씩 대응
    → 각 주문자클래스의 인스턴스는 반드시 1개의 주문자정보클래스의 인스턴스를 가지며, 각 주문자정보클래스의 인스턴스는 반드시 1개의 주문자클래스의 인스턴스에 보존됨
  - 상품관리클래스의 인스턴스는 상품클래스의 인스턴스를 0개이상 관리

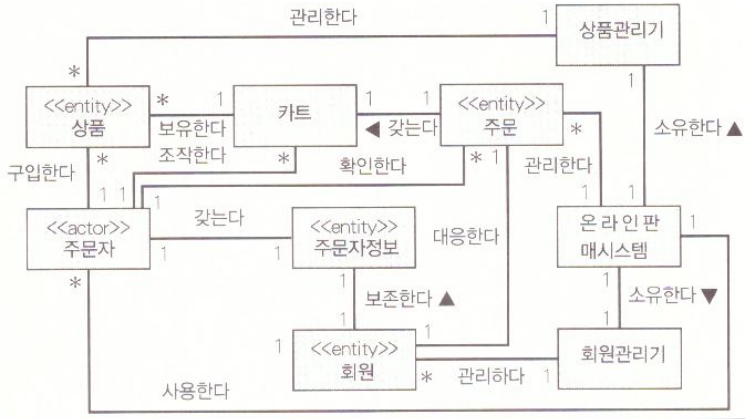

그림 6.10 온라인판매시스템의 개념모델

- 실제 소프트웨어 구축시, 무한개의 상품을 취급하는 것은 불가능하므로 반드시 상한이 존재함
- 상한이 명확하게 요구사양에 기술되어 있지 않은 경우, 설계 및 구현공정에서 결정하면 됨
- 분석공정에서는 인스턴스의 개수를 정확하게 정의할 필요는 없으며 1개 또는 다수개를 나타내는 것으로 충분

6.4 객체지향분석의 순서

6.4.2 Class Diagram의 작성

- 개념모델의 작성 후에 클래스의 메소드와 속성을 추출하여 Class Diagram을 완성해감

- 메소드는 속성보다 우선적으로 추출하는 것이 좋음
  - 메소드는 클래스의 외부에 공개, 속성은 클래스 내부에 은폐됨
  - 객체의 동작을 생각할 때, 외부에 공개되는 특성이 내부에 은폐되는 속성보다 중요하기 때문
- 그림 예: 클래스의 메소드와 속성, 클래스들 사이의 계승(상속) 관계와 집약(집합) 관계 등을 추가한 Class Diagram(주문에 관한 부분만 표현)

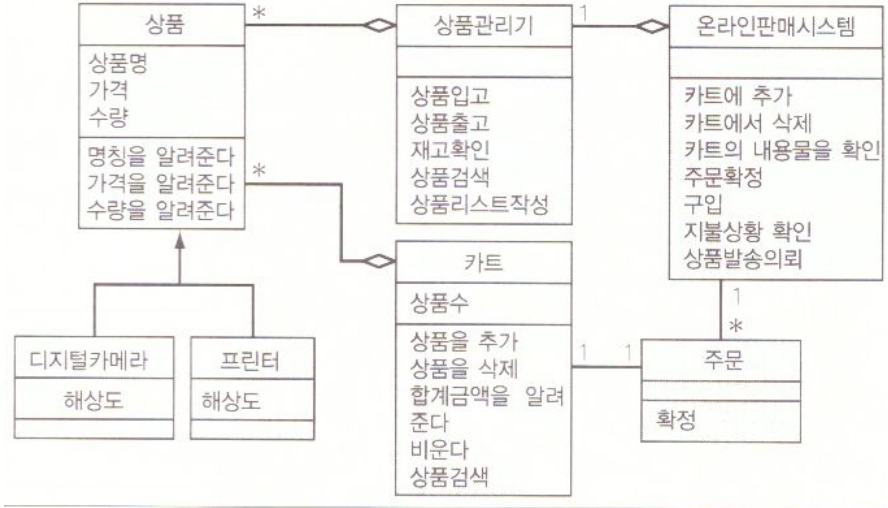

그림 6.11 Class Diagram

6.4.3 동작(Activity)의 기술

- Activity Diagram
  - 여러 개의 동작(Activity)들 사이의 제어흐름을 표현
    - 동작(Activity): 어떤 기능을 실현하기 위한 작업의 기본단위
  - Activity Diagram을 기술함으로써 작업의 순서가 명확하게 됨
  - 그림 예: "주문을 확정한다"라는 Use Case에 대응하는 Activity Diagram
    - Activity : 모서리가 둥근 사각형
    - 분기: 마름모
      - 각각의 분기에 조건(Guard조건) 기술
    - ●: 초기상태
    - ◉: 종료상태
    - 굵은 직선: 병행동작의 시작과 종료에 대한 동기화

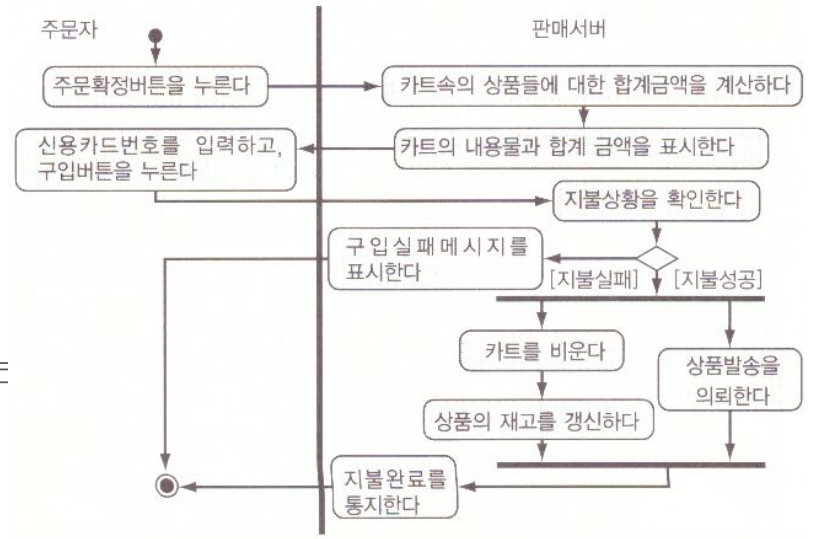

그림 6.12 | Activity Diagram

6.4.3 동작(Activity)의 기술

- Sequence Diagram
  - 각 객체들의 메시지 송수신을 시간흐름에 따라 표현
  - 메시지의 흐름을 직감적으로 파악하려는 경우 적합
  - Use Case Diagram에 있는 Use Case마다 각각 작성

  - 그림 예: "주문을 확정한다"라는 Use Case에 대응하는 Sequence Diagram
    - 가로축: 객체들 나열
    - 세로축: 시간흐름
    - 메시지: 위에서 아래로 순차적으로 송수신
    - 생명선: 각 개체들로부터 아래쪽으로 그어진 점선
    - 활성화구간(Activation Bar): 각 생명선 위에 배치된 직사각형으로 각 개체들이 어떤 처리를 수행하고 있음을 나타냄

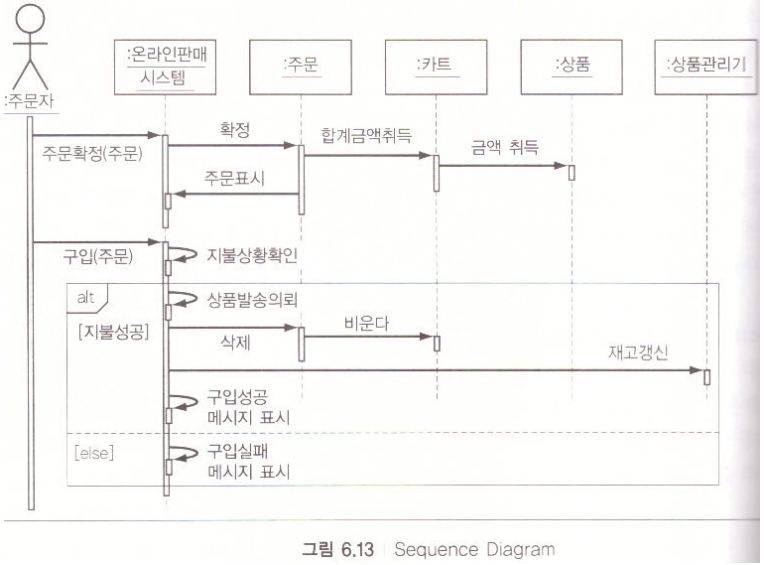

그림 6.13 Sequence Diagram

6.4.3 동작(Activity)의 기술

- Sequence Diagram
  - 비활성화 상태의 객체가 메시지를 수신하면 객체는 활성화되고 수신한 메시지에 대응하는 처리를 실행
  - 활성화된 객체가 다른 객체에게 메시지를 보내는 경우도 있음(메시지 전파)
  - 자기호출: 활성화구간이 중복되어 있는 부분으로 자기자신에게 메시지를 보냄

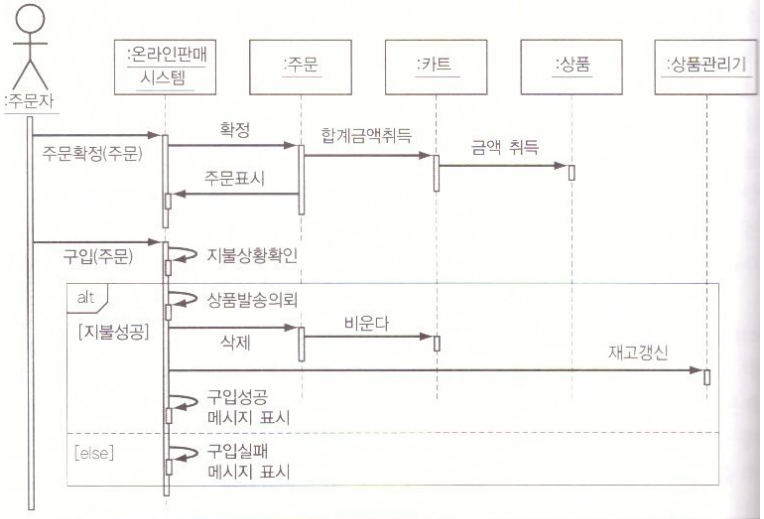

그림 6.13 Sequence Diagram

6.4 객체지향분석의 순서

6.4.3 동작(Activity)의 기술

- Sequence Diagram
  - 시간의 흐름을 분기
    - 선택(alt)을 지정함으로써 2개의 시간흐름을 동시에 표현
    - 지불이 성공한 경우(조건 [지불성공]), 해당 조건을 포함하는 직사각형으로 둘러싸인 부분이 실행됨
    - 그 외의 경우 [else]를 포함하는 사각형으로 둘러싸인 부분이 실행됨
  - 메시지는 그대로 각 객체들의 동작에 대응
    - Sequence Diagram을 기술함으로써 클래스의 메소드도 동시 추출 가능

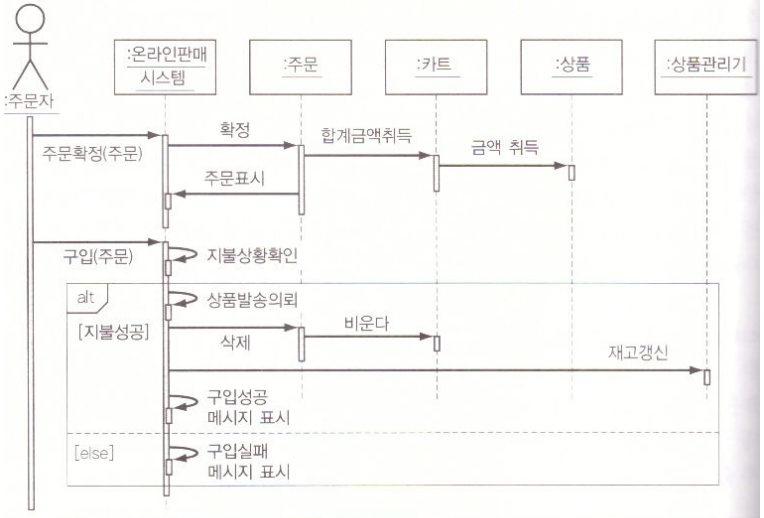

그림 6.13 Sequence Diagram

6.4 객체지향분석의 순서

6.4.3 동작(Activity)의 기술

- State Machine Diagram
  - 각 객체들의 동적인 움직임 표현

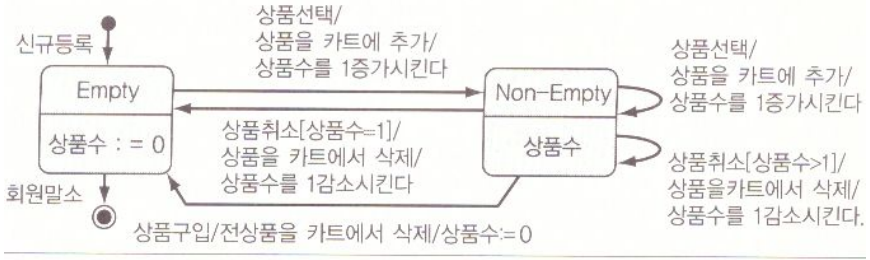

그림 6.14 | State Machine Diagram

  - 그림 예: "카드"객체의 State Machine Diagram
    - 2개의 모서리가 둥근 사각형: "카트"객체의 상태
      - Activity Diagram에서의 ● 초기상태, ◉ 종료상태와 유사한 표기법
    - 화살표 : 상태천이
      - "Empty"상태에 있는 "카드"객체가 "상품선택"이벤트(화살표에 붙여진 레이블의 첫 번째 항목)을 수신하면 "Non-Empty"상태로 천이
      - 이때 객체의 동작으로서 "상품을 카트에 추가"와 "상품수를 1 증가"(화살표에 붙여진 레이블의 두 번째 항목)가 실행
    - 이벤트를 나타내는 레이블에 있는 중괄호([]): 상태천이에 대한 가드조건
      - "Non-Empty"상태에 있는 "카트"객체가 "상품취소"이벤트를 수신하고 가드조건 (상품수=1)을 만족하는 경우, "Empty"상태로 천이

6.4.4 모델의 세련화

- 동작에 대한 분석이 종료되면 다시 한번 Class Diagram을 작성하는 단계로 돌아가서 각 도면을 세련화시킴
- 소개하지 않았던 도면들에 관해서도 분석, 설계, 구현의 각 단계에서 필요할 때 적절히 기술
  - Object Diagram을 기술함으로써 클래스들 사이의 관계에 누락된 것이 없는지 확인가능
  - 타이밍이 중요한 시스템에서는 Timing Diagram을 기술할 필요가 있음
  - 구현 공정에 가까워질수록 Package Diagram, Component Diagram, Deployment Diagram 등 기술

연습문제 (객관식)

각 질문에 대한 알맞은 답의 번호(숫자)를 괄호안에 표시하시오.

- 1. 객체지향에서 객체의 성질이 아닌 것은 무엇인가?
  (1) 상태 (2) 동작 (3) 식별성 (4) 분류

- 2. 클래스와 객체의 관계로 옳지 않은 것은 무엇인가?
  (1) 인스턴스가 갖는 성질은 클래스에 의해 정의된다.
  (2) 같은 클래스로부터 생성된 인스턴스들의 속성과 동작은 서로 같다.
  (3) 서로 다른 인스턴스들은 독립적으로 취급할 수 없다.
  (4) 서로 다른 인스턴스들은 서로 다른 속성값(내부상태)을 가질 수 있다.

- 3. 캡슐화의 최대 장점인 정보은폐에 대한 설명으로 옳지 않은 것은 무엇인가?
  (1) 객체의 속성은 캡슐화에 의해 외부로부터 은폐시킬 수 있다.
  (2) 외부에 공개하는 메소드와 내부에 은폐시키는 메소드로 분리 가능하다.
  (3) 객체의 속성이 부당하게 읽혀지거나 변경되는 것을 방지할 수 있다.
  (4) 객체를 이용하는 개발자는 객체 내부의 속성과 메소드에 관한 구현내용을 알고 있어야 한다.

객관식 문제 (답안지)

- 1. (4) → 객체의 성질: 상태, 동작, 식별성
- 2. (3) → 서로 다른 인스턴스들은 독립적으로 취급할 수 있다.
- 3. (4) → 외부에서는 객체의 인터페이스만 살펴볼 수 있기 때문에 해당 객체를 이용하는 개발자는 객체 내부의 속성과 메소드에 관한 구현내용을 알고 있을 필요가 없다.

연습문제 (객관식)

각 질문에 대한 알맞은 답의 번호(숫자)를 괄호안에 표시하시오.

❏ 4. 유즈케이스 기술서(Use Case Description)에 대한 설명으로 옳지 않은 것은 무엇인가?
(1) Use Case에 대하여 정해진 사용기법을 정의한 것이다.
(2) 이용자의 조작이 정상적으로 실행된 경우만 기술하면 된다.
(3) Use Case를 실행하기 전에 성립해야 되는 조건을 사전조건으로 기술 가능하다.
(4) Use Case의 실행 후에 성립되어야 하는 조건을 사후조건으로 기술 가능하다.

❏ 5. Class Diagram에 대한 설명으로 옳은 것은 무엇인가?
(1) Use Case에 대하여 정해진 사용기법을 정의한 것이다.
(2) Use Case와 요구사양으로부터 시스템에 등장하는객체를 찾아내서 클래스를 추출 후 관련관계를 찾아 작성한다.
(3) 여러 개의 동작(Activity)들 사이의 제어흐름을 표현한다.
(4) 각 객체들의 메시지 송수신을 시간흐름에 따라 표현한다.
(5) 각 객체들의 동적인 움직임을 표현한다.

객관식 문제 (답안지)

❏ 4. (2) → 이용자의 조작이 정상적으로 실행된 경우뿐만아니라 이용자의 조작이 실패한 경우에 대한 예외처리에 대해서도 기술된다.

❏ 5. (2) → Class Diagram

---

(1) → 유즈케이스 기술서(Use Case Description)
(3) → Activity Diagram
(4) → Sequence Diagram
(5) → State Machine Diagram

연습문제 (주관식)

선문대학교

주관식 문제 (괄호안에 알맞은 단어를 입력하시오.)

- 1. (　　)이란 데이터와 이에 대한 처리를 일체화한 객체에 주목하여 객체의 구조와 동작을 명확하게 함으로써 고객과 이용자의 요구사항을 사양화하는 기법이다.
- 2. (　　)란 인간이 인지할 수 있는 구체적 또는 추상적인 사물/개념이며, 상태와 동작, 식별성과 같은 3가지 특성을 갖추고 있다.
- 3. 객체지향소프트웨어에서 객체를 상태는 (　　)으로, 동작은 (　　)로 구현되는 것이 일반적이다.
- 4. 데이터와 이에 대한 처리를 일체화하여 외부로부터 액세스를 제한하는 체계를 (　　)라고 한다.
- 5. 공통적인 속성과 동작을 갖는 객체를 추상화하여 작성된 템플릿을 (　　)라고 하고, 이것으로부터 생성된 객체를 (　　)라고 한다.

주관식 문제 (답안지)

- 1. 객체지향분석 (OOA : Object-Oriented Analysis)
- 2. 객체
- 3. 속성 (Property), 메소드 (Method)
- 4. 캡슐화 (Encapsulation)
- 5. 클래스 (Class), 인스턴스 (Instance)

연습문제 (주관식)

선문대학교

주관식 문제 (괄호안에 알맞은 단어를 입력하시오.)

- 6. 이용자가 어떻게 시스템을 사용하는지를 나타내는 전형적인 사례를 도면으로 표현한 것이 ( )이다.
- 7. ( )은 클래스의 구조와 클래스들 사이에 성립하는 관계(관련, 계승(상속), 집약(집계))를 정리하기 위해 사용된다.
- 8. Class Diagram을 작성할 때, 관련관계의 양쪽 끝에 표시하여 1개의 관련관계(링크)에 대해 각 클래스의 인스턴스가 몇 개씩 결합되어 있는지를 나타내는 것을 ( )라고 한다.
- 9. Sequence Diagram에서 활성화구간이 중복되어 있는 부분으로 자기자신에게 메시지를 보내는 것을 ( )이라고 한다.
- 10. 객체지향의 이점(장점)을 간단히 쓰시오.

주관식 문제 (답안지)

- 6. Use Case Diagram

  (또는,
  유즈케이스 다이어그램,
  유즈케이스 도면,
  유스케이스 다이어그램,
  유스케이스 도면)
- 7. Class Diagram
- 8. 다중도
- 9. 자기호출
- 10. 높은 변경용이성 및 확장성

Any questions?