Introduction to OOAD using UML tools

200911385 박기남
200911425 조서경
200911426 조성완
200911427 조아라

0. Index

Chapter	Page
1. What about OOAD	
1) Definition & History	3
2) Terms	4
3) Modeling	8
4) OOA & OOD	11
5) Summary of OOAD	17
2. What about UML	
1) Definition & History	20
2) Characteristic and Composition of UML	21
3) Summary of UML	45

1. What about OOAD

1) Definition & History

① 정의

소프트웨어를 개발하는 하나의 방법론으로 모든 소프트웨어 시스템의 주요 기본요소를 사물을 가리키는 객체와 그 객체들을 하나의 집합으로 묶은 클래스로 구성하는 객체지향적인 분석과 설계 방법을 말한다.

객체지향적이란 것은 현실세계에 실제 존재하는 사물, 즉 객체들을 지향한다는 것이다. 그리함으로써 보다 현실세계의 개념을 편리하고 빠르게, 즉 실용적으로 이용할 수 있다.

② 역사

▶1980년대
-객체지향 기술을 기반으로 한 방법론, 절차, 도구들이 본격적으로 등장
-구조적 방법이 소프트웨어 개발에 널리 사용되고 있던 시기

▶구조적 프로그래밍에 기반을 둔 구조적 방법
-개발 대상을 기능에 초점
-기능간의 자료 흐름을 다이어그램으로 표현
-일괄처리 방식인 자료 변환을 중심으로 응용 소프트웨어 개발에 활용

▶자료 구조에 기반을 둔 정보공학 방법
-자료구조를 결정한 다음 연산 함수를 정의하는 실체 접근
-데이터베이스를 사용하여 자료를 관리할 트랜잭션 처리

▶제어용 소프트웨어 개발 방법론
-임의의 시점에서 발생하는 이벤트에 대응, 행동을 정의할 상태 접근

▶기존 방법론들
-소프트웨어 개발의 생산성 향상과 품질 개선을 위해 제안

▶기존 방법들이 해결하지 못한 4 가지 문제점
-소프트웨어 개발 환경의 급격한 변화
-소프트웨어 개발 자체의 어려움

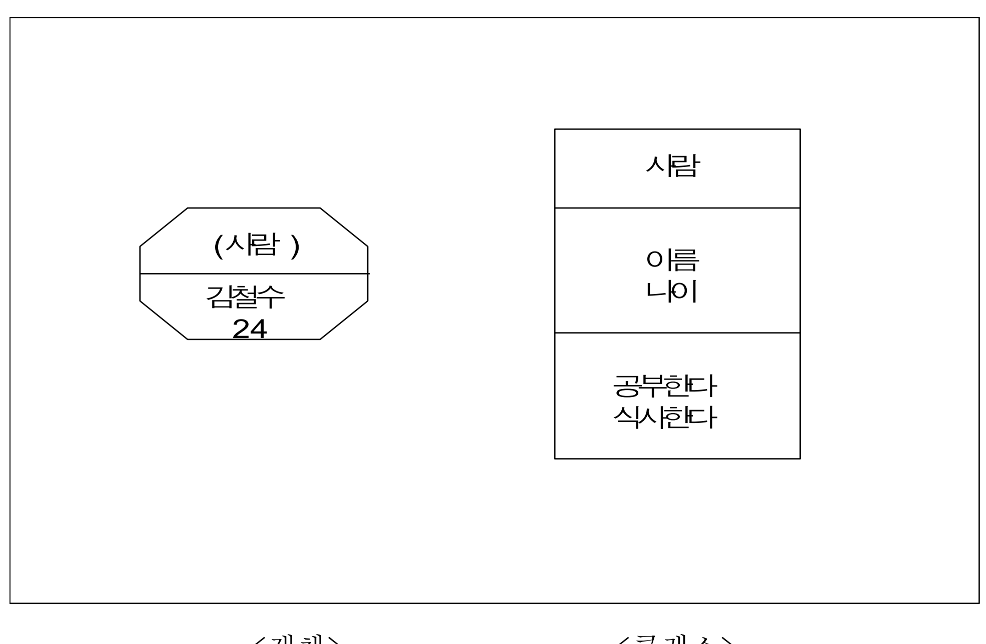
1. 실세계 모델링의 대상이 기능 중심
2. 자료와 절차가 분리□
3. 유사 소프트웨어 부품의 중복 개발□
4. 묵시적 이용 절차가 전제

2) Terms

① 객체(Object) & 클래스(Class)

▶객체 : 객체 모델링의 기본 단위이며 구석이 다듬어진 사각형으로 표시

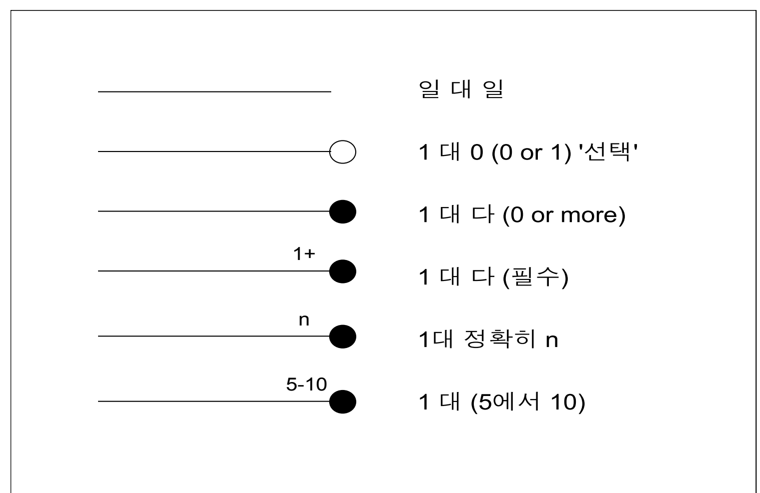
▶클래스 : 유사한 객체들의 모임이며 사각형으로 표시



<객체> <클래스>

② 매핑제약조건

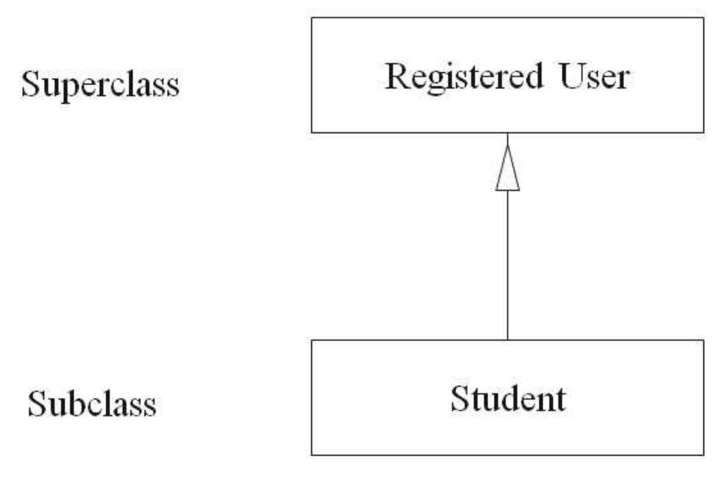
클래스들 사이에 맺어질 수 있는 관계를 통해 객체들이 지켜야 하는 제약조건



<매핑수와 참여 제약조건>

③ 상속성(Inheritance)
 객체기술의 가장 핵심이 되는 개념으로 프로그램을 쉽게 확장할 수 있도록 해주는 강력한 수단이 된다.

 상속의 효과는 클래스를 체계화할 수 있으며, 기존의 클래스로부터 확장이 용이하다는 것이다. 함수와 변수를 슈퍼클래스(상위클래스)에서만 정의하고 서브클래스(하위클래스)에서는 따로 정의하지 않아도 된다. 대신 상위의 클래스의 내용에다 추가적인 특성을 덧붙이기만 하면 되므로 매우 효율적이다. 또한 공통의 특성을 서브클래스마다 반복적으로 기술하지 않고 한번만 기술하기 때문에 중복을 줄여 준다.



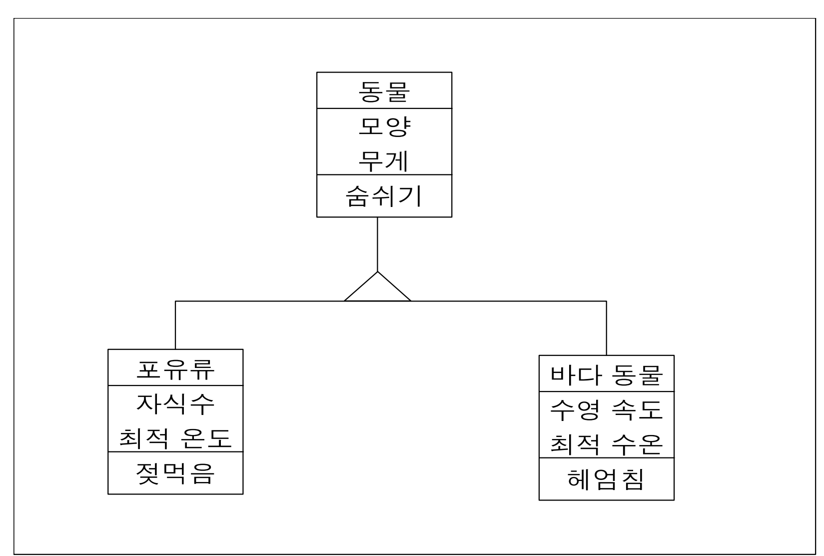
<일반적인 상속>

<다중 상속>

④ 일반화(Generalization)

 유사한 클래스들 사이의 공유되는 속성과 동작을 묶어주며, 다른 한편 그들 사이의 다른 점을 보존할 수 있게 하여주는 효과적인 추상화 기법. 일반화는 'is_a' 또는 'kind_of' 관계이며 각 하위 클래스의 [1]인스턴스는 상위 클래스의 인스턴스가 된다.

 일반화를 통해 다계층 구조를 만들어 나갈 수 있으며 이 경우 한 하위클래스는 모든 상위 클래스의 속성과 동작을 상속받는다.



<클래스의 일반화>

⑤ 추상화(Abstraction)

 현실세계의 사실을 그대로 객체로 표현하기 보다는 문제의 중요한 측면을 주목하여 상세내역을 없애나가는 과정

 복잡한 프로그램을 간단하게 해주고 분석의 초점을 명확히 함

-프로세스 추상화(process abstraction)

 프로그램에서 자주 나오는 상세한 부분을 함수로 묶어 호출하게 함으로써 이해하기 쉽고 간단한 모양으로 만든다.

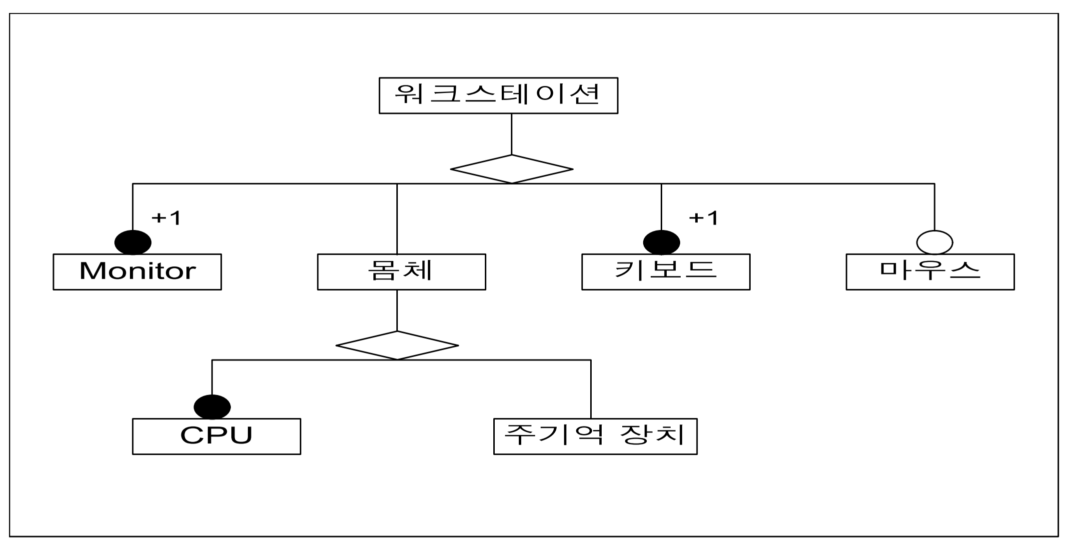
-데이터 추상화(data abstraction)

 integer, real, date 같은 데이터 타입을 개발자는 정수 연산의 값의 범위나 진법, 보수 형태의 치환 등을 일일이 프로그램해 넣을 필요 없이 단지 선언만 한다.

1) 클래스에서 생성되는 각 객체를 지칭

⑥ 집단화(Aggregation)
 집단화는 클래스들 사이의 "부분-전체" 관계 또는 "부분" 관계로 설명되는 연관성을 나타낸다. (has-a 관계)
 다시 말해 집단화는 여러 부속 객체들이 조립되어 하나의 객체가 구성되는 것을 의미하며 객체지향 기법에서는 활발히 사용되고 있다.

 집단화 기호는 조그만 마름모 기호로 표시되며 매핑 제약조건과 참여 제약조건에 대한 기호는 일반화의 경우와 동일하다.



<다계층 집단화>

⑦ 재사용(Reusing)
 객체기술의 장점으로 객체, 클래스는 모두 다른 프로그램 개발시에도 재사용 될 수 있다.
 각 Component는 종속되어있지 않고 개별적으로 행동할 수 있기 때문에 새로운 조합 또한 가능하다.

3) Modeling

① 모델(Model)

모델이란 하나의 시스템을 개발하는데 있어 현실을 추상화 시켜놓은 청사진을 말한다. 예를 들면 건물을 짓기 위한 총체적인 설계도와 사용자의 요구사항을 반영한 추상적인 개념을 말한다.

② 모델링(Modeling)

정보시스템 설계모델을 현실세계에 맞게 구현하는 작업을 말한다.
모델링이 필요한 이유는 미리 모델을 만들어 보면서 앞으로 개발하려는 시스템을 좀 더 구체적이고도 명확하게 인식 할 수 있기 때문이다.

생성할 모델을 선택하는데 따라 문제를 공략하는 방법과 해결책을 구현하는 방법에 큰 영향을 준다.
모든 모델은 다양한 수준의 정밀도를 표현할 수 있으며, 훌륭한 모델은 현실세계를 잘 반영한다.

하나의 모델로는 충분치 않다. 특수한 모델은 여러 개의 모델을 선정, 그 중 하나를 선택해 모델링을 해야 한다.

③ 모델링의 목적

-모델은 시스템을 현재 또는 원하는 모습으로 가시화하도록 도움을 준다.
-모델은 시스템의 구조와 행동을 명세화 할 수 있게 해준다.
-모델은 시스템을 구축하는 틀을 제공해 준다.
-모델은 우리가 결정한 것을 문서화 한다.

④ 모델링의 종류

▶객체 모델링

-객체 모델링은 시스템에 요구되는 객체들을 보여줌으로써 주로 시스템의 정적인 구조를 포착하는 데 사용
-모델링 중 객체지향 분석에서 가장 중요하며 선행되어야 할 모델링
-객체 모델링의 복잡도를 관리하기 위하여 정보 모델링에서 언급된 추상화, 집단화, 일반화의 개념들이 사용
-객체들이 발견된 다음 객체들 사이의 연관성을 찾아내고 객체들을 정의하기 위한 속성 추가

▶동적 모델링

-객체 모델링을 통해 시스템에 요구되는 객체들의 구조 와 객체들 사이의 관계 정립

-객체들 사이의 제어 흐름, 상호 작용, 동작의 순서 등을 다룬다.

-OMT 기법은 동적 모델링에 상태변화도(STD) 사용

-제안서로부터 시나리오를 만들게 되며 시나리오는 객체 또는 시스템의 한 실행 과정을 사건들의 흐름으로 표시

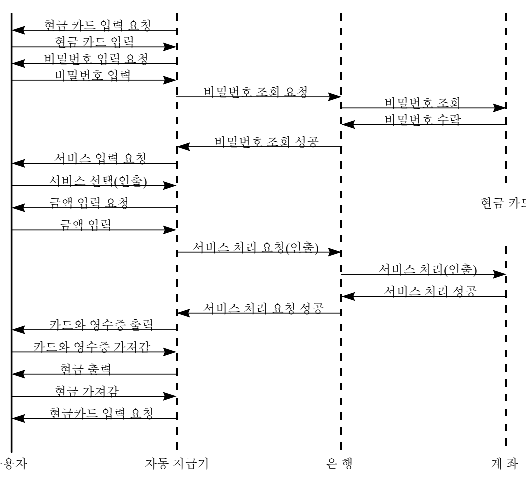
-사건의 순서와 사건을 주고받는 객체들이 사건추적도에 나타나게 되며 수직선은 객체를 표시하고 수평 화살표는 사건의 흐름을 나타낸다.

-사건은 위에서 아래로의 순서에 의해 수행

Ex) ATM System Development

- 자동 지급기가 현금 카드를 입력할 것을 요구한다.
- 사용자가 현금 카드를 자동 지급기의 카드 입구에 넣는다.
- 자동 지급기는 현금 카드로부터 계좌 번호와 카드 번호를 읽고 사용자에게 비밀 번호를 요구한다.
- 사용자가 비밀번호를 입력한다.
- 자동 지급기는 현금 카드 소속 은행에게 비밀 번호 대조를 요청한다.
- 은행은 현금 카드에게 비밀 번호 대조를 요청한다.
- 현금 카드는 은행에게 비밀 번호가 일치함을 알린다.
- 은행은 자동 지급기에게 비밀 번호가 일치함을 알린다.
- 자동 지급기는 사용자에게 가능한 서비스를 보여준다.
- 사용자가 현금 인출을 선택한다.
- 자동 지급기는 인출할 금액을 물어본다.
- 사용자가 인출할 금액을 입력한다.
- 자동 지급기는 해당 은행에게 인출할 금액 인출을 요구한다.
- 은행은 해당 계좌에게 인출할 금액 인출을 요구한다.
- 계좌는 잔액에서 인출할 금액을 인출하고 인출이 성공적으로 끝났음을 은행에 알린다.
- 은행은 자동 지급기에게 현금 인출이 성공적으로 끝났음을 알린다.
- 자동 지급기는 사용자에게 카드와 영수증을 내어준다.
- 사용자가 카드와 영수증을 가져간다.
- 자동 지급기가 인출 금액을 내준다.
- 사용자가 인출 금액을 가져간다.
- 자동 지급기가 현금 카드를 입력할 것을 요구한다.

<시나리오>



<사건추적도>

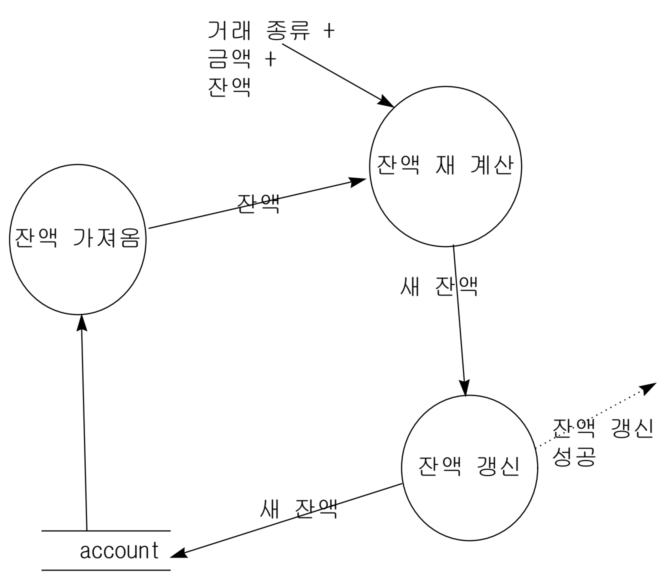
▶기능 모델링

-객체 모델링, 동적 모델링에 이어 시스템을 기술하는 세 번째 단계

-입력 값으로부터 계산을 거쳐 어떤 결과가 나타나는지를 보여주며 어떻게(how to) 유도되었는지의 구현 방법은 고려하지 않는다.

-자료흐름도(DFD)의 경우 입력 흐름은 프로세스를 통해 변환되어 출력 흐름으로 바뀌고, 다른 프로세스의 입력이나 외부로의 출력으로 작용한다.

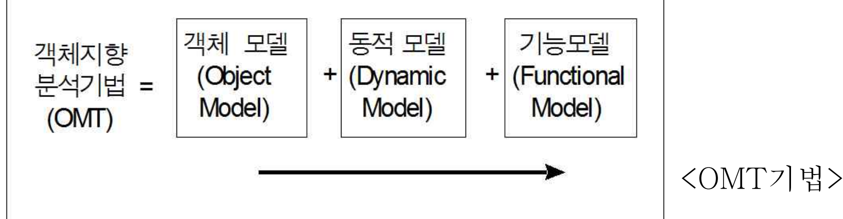
-자료흐름도(DFD)의 정보 흐름은 객체에 해당된다.



<자료흐름도>

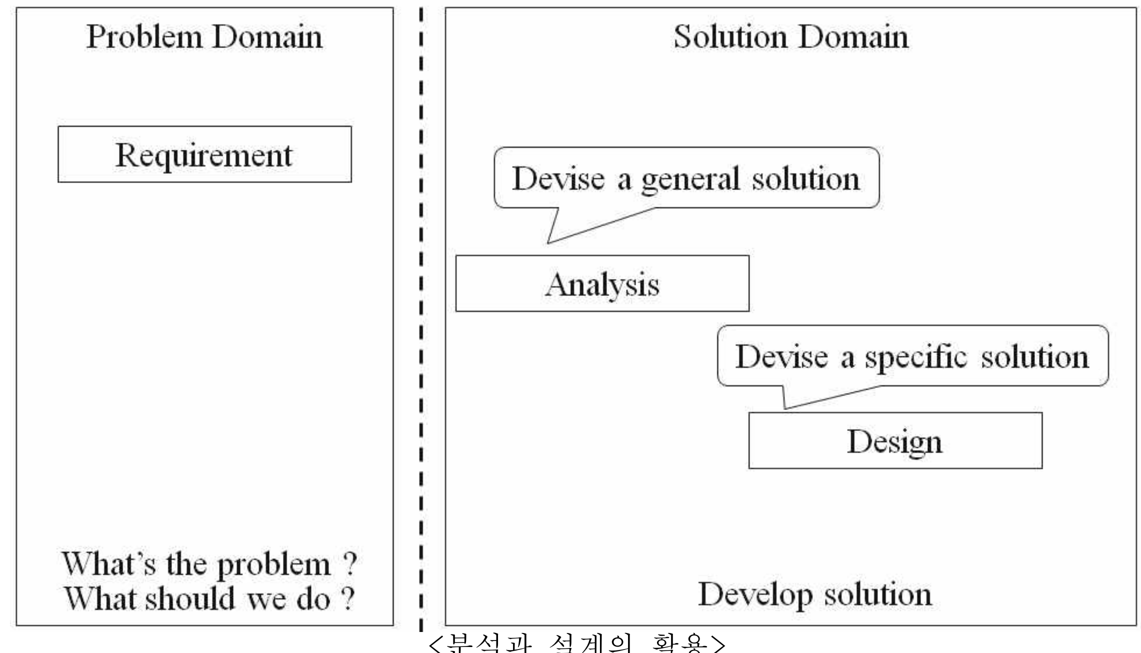
⑤ 모델의 통합

세 가지 모델링기법을 순차적으로 진행, 통합하면 객체에 속한 모든 가능한 정보, 객체들 사이의 관계, 객체의 오퍼레이션들을 나타낼 수 있게 되어 시스템에 대한 완벽한 기술이 이루어진다.



<OMT기법>

4) OOA(Object-Oriented Analysis) & OOD(Object-Oriented Design)



<분석과 설계의 활용>

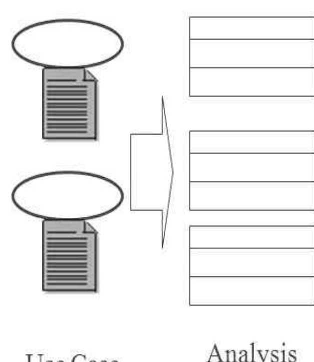
① 객체지향분석(Object-Oriented Analysis)

객체지향관점에서 고객의 요구사항을 상세하게 분석하여 일반적인 방향을 고안한다.

② 분석단계

Use Case를 활용하여 고객의 요구 사항을 빠짐없이 분석하는 업무를 장기간에 걸쳐 실행하여, 보다 구체적인 고객 요구 사항을 추출해낸다.

프로젝트 성공을 결정하는 가장 중요한 작업으로 전체 프로젝트의 성패를 결정짓는 중요한 작업이다.



<Use Case 를 통한 Analysis>

▶Use Case Behavior로부터 클래스를 도출

-Use Case

* 시스템이 제공하는 서비스 혹은 기능으로 시스템이 외부에 제공하는 사용자 관점의 기능단위이며 외부의 요청에 반응하여 원하는 처리를 수행하거나 원하는 정보를 제공한다.

* 그 자체로 의미있는 자기완결형(Self Contained)의 서비스 단위이며 개발자는 이를 모델링해야한다.

-기능적 요구사항과 비기능적 요구사항

* 기능적 요구사항(Functional Requirement)
  : 시스템의 기능에 대한 내용. 시스템이 무엇을 수행하는지에 대한 내용을 담고 있는 요구사항

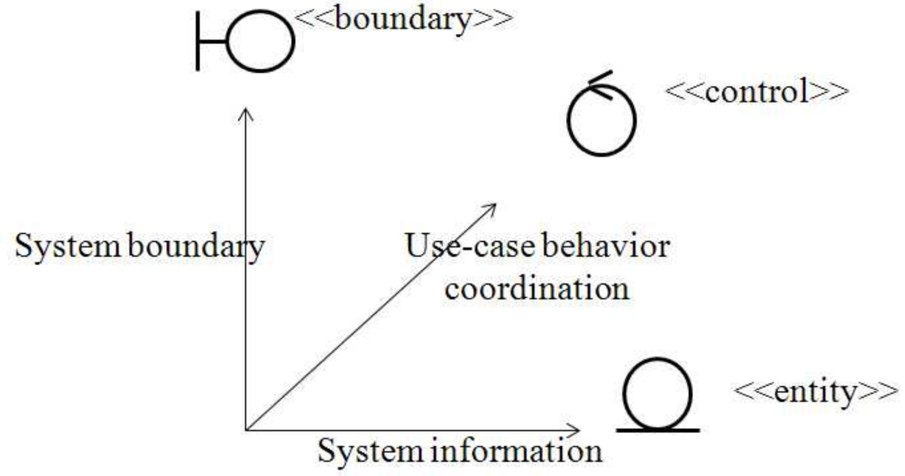
* 비기능적 요구사항(Non Functional Requirement)
  : 시스템의 기능성 이외에 시스템을 기동시키거나 운영하는 데 필요한 환경적인 요소들에 대한 요구사항

Functionality(기능성) : 기능적 요구사항
시스템 처리 절차
처리방법
수행결과
Usability(사용편리성) : 비기능적 요구사항
UI 편의성
문서에 의한 지원
미적 기준
Reliability(신뢰성) : 비기능적 요구사항
정확성
예측성
MTBF
Performace(수행성능) : 비기능적 요구사항
속도
응답시간
작업처리량
Supportability(지원성) : 비기능적 요구사항
적용성
확장성
호환성

-Use Case Behavior

Use Case에 기술된 Process의 행동양식으로

Entity Class → Boundary Class → Control Class 의 순서로 분류



<boundary, entity, control class의 관계>

-Boundary Class

* 사용자와 인터페이스를 담당하는 클래스로 주로 UI(User Interface)에 짝을 이루어 정의된다.

역할
사용자로부터 UI 이벤트를 직접 받아서 시스템 내부에 처리 의뢰
시스템으로부터의 응답을 UI를 조작하여 사용자에게 보여줌
사용자의 적절치 못한 입력에 대해서는 오류 메시지 등을 통해 직접 처리

* 세부적으로 조사하기 보다는 클래스간 관계에 초점을 맞추어 조사하는 것이 바람직하다.

-Entity Class

＊ 원하는 처리에 필요한 정보(데이터)를 관리하는 클래스

역할
처리에 필요한 데이터를 읽어옴
데이터를 저장
데이터를 삭제
데이터를 수정

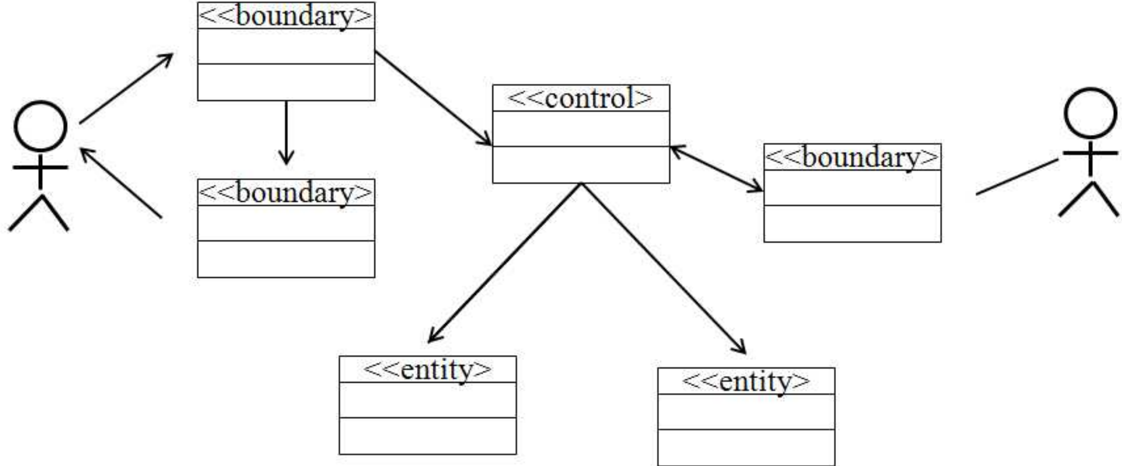
＊ 세부적으로 조사하기 보다는 클래스간 관계에 초점을 맞추어 조사하는 것이 바람직하다.

-Control Class

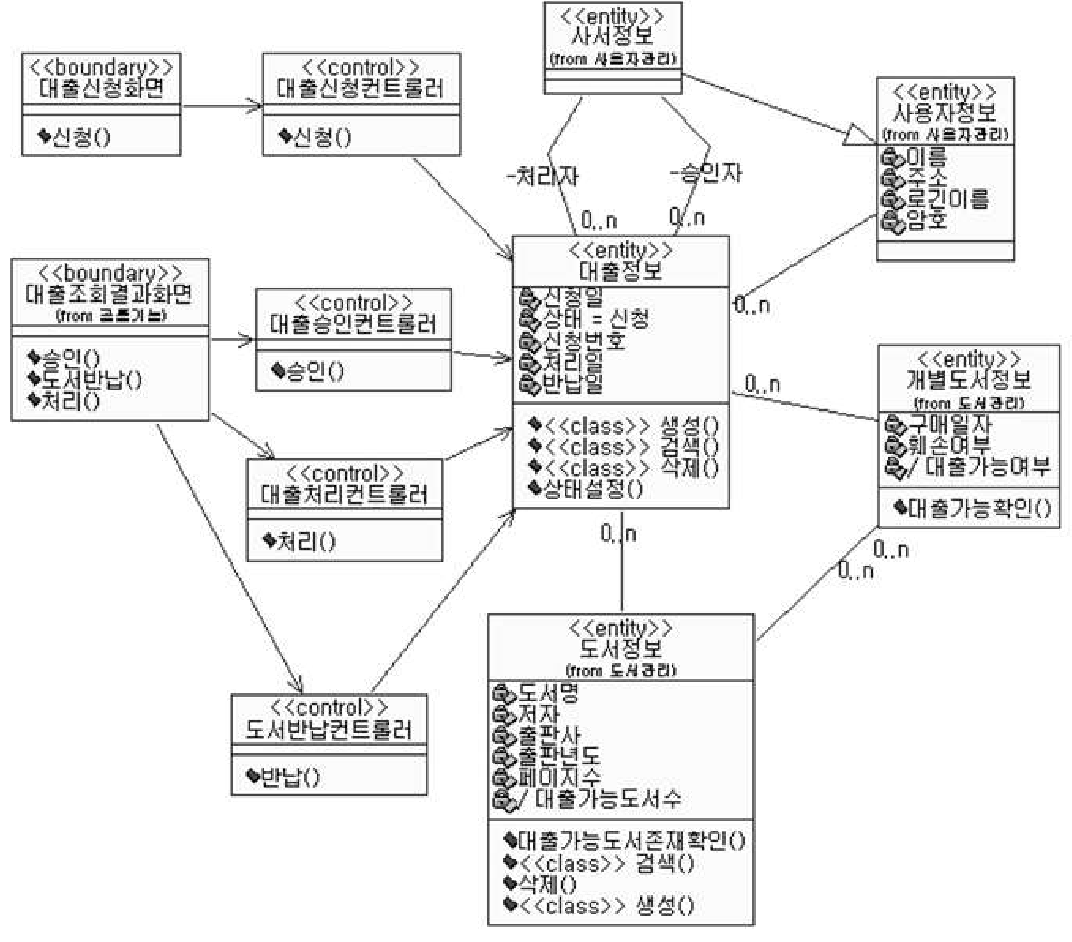
＊ Boundary Class와 Entity Class사이의 처리를 중재하고 제어하는 클래스

역할
여러 클래스간의 협력작업을 제어하고 클래스간의 실행순서를 통제
결과를 조합하여 Boundary Class에 전달
데이터를 처리하는 도중에 발생하는 Transaction을 관리

＊ Boundary Class와 Entity Class의 상호작용을 분석하므로 실질적인 분석단계라고 볼 수 있다.



<Use Case Analysis의 모식도>



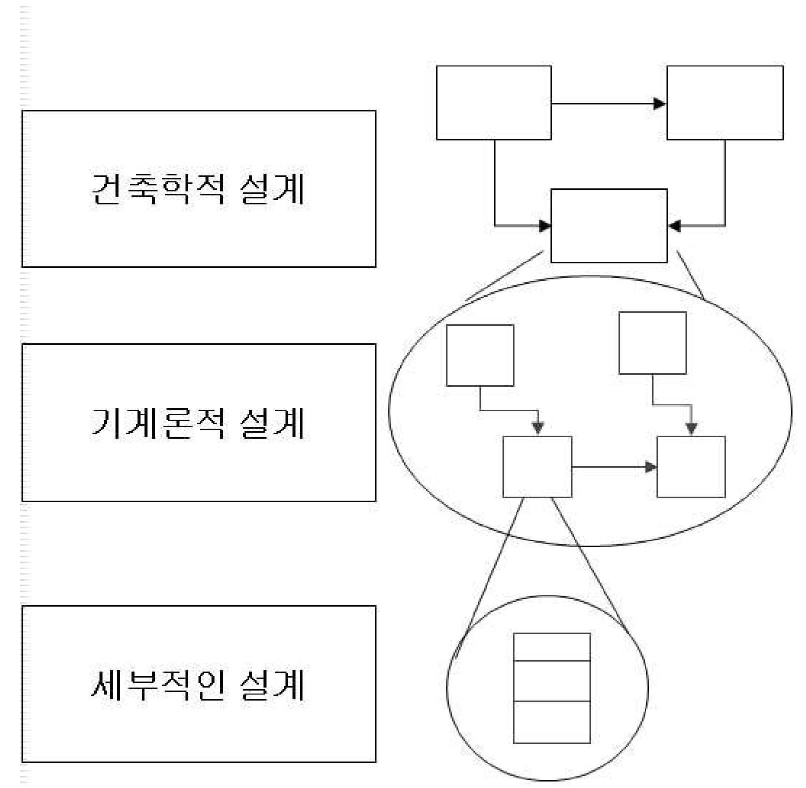
<도서관 관리 프로그램의 Use Case Analysis>

③ 설계단계

-분석 단계를 통하여 앞에서 논의된 세 가지의 서로 다른 모델을 하나로 통합하는 작업이 이루어 짐

-이는 동적 모델의 사건, 동작 및 활동을 객체의 오퍼레이션에 매핑, 기능 모델의 프로세스를 객체 모델의 오퍼레이션에 통합시키는 것

-설계단계에서의 결과물로 UML에서는 class diagram, sequence diagram, collaboration diagram, state chart diagram, activity diagram, component diagram, deployment diagram 이 나오게 된다. 이들은 다음 장에서 자세하게 기술 할 것이다.



<설계의 기본적인 Process>

▶건축학적 설계(Architectural Design)

-객체나 클래스를 Packing하여 Visible한 [2]Attributes를 제시

▶기계론적 설계(Mechanistic Design)

-건축학적 설계를 통해 Packing된 구조를 Leveling하여 계층적으로 제시

-이 단계에서 예외처리를 수행할 수 있다.

---

2) 속성 (=Property)

▶세부적인 설계(Detailed Design)
-변수, 함수 등을 보다 상세하게 제시(단, Private하게)
-고객이 보다 이해하기 쉬운 구조로 제시

5) Summary of OOAD

① 장점

장점	설명
실세계를 정확히 반영	현실의 실체를 닮은 "객체"를 중심으로 사고하고 문제를 해결하는 체계이다. 객체란 실체를 인간의 머릿속에 매핑시킨 것이고 이를 바탕으로 현실에 가장 가까운 방식으로 문제를 분석한다. 이러한 방식 덕분에 객체지향 SW는 다음과 같은 장점이 있다. - 이해하기 쉽고, 유지보수성이 향상 - Architecture 기반으로 SW구축이 용이 - 문제영역에 대한 현실적 SW분석과 설계
하나의 패러다임	객체지향은 한 분야의 기술에만 적용되는 방식이 아니고, SW의 전 분야에적용되는 하나의 패러다임이다. 객체지향은 비즈니스 분석부터 SW의 분석, 설계, 개발, 구현에 이르는 SW프로세스 측면 뿐 아니라 프로그래밍 언어, Middleware, OS에 이르는 개발도구, 체계에까지 하나의 주제로 통일된 체계를 형성하고 있다. 따라서 분야에 따라 각기 다른 방식과 체계로 접근하는 불편함은 객체지향을 채택하는 순간 해소될 수 있다.
재 사용성	객체지향은 상속과 캡슐화를 통해 기존 프로그램의 재사용 가능성을 극대화시킨 체계이다. 코드를 복사해서 해당 부분에 삽입하는 방식의 재사용이아닌, 정보 은폐와 상속이라는 객체지향만의 특별한 장치로 재사용을 지원하고 있다. 객체지향의 재사용은 기존 클래스를 상속하는 방식으로 이루어지기 때문에 대단히 쉽고, 간단하게 지원되고 있다. 실제 클래스 라이브러리와 기존 코드의 재사용은 매우 흔한 일이다. 물론 분석과 설계과정에서도 기존 모델을 재사용하기 매우 쉽고, 기존 코드의 재사용을 통해 생산성 향상과 품질의 안정을 꾀할 수 있다.
높은 안정성	객체지향에서는 다른 방식에 비해 SW 개발 중에 요구사항이 변경될 경우 기존 프로그램을 많이 변경하지 않아도 되기 때문에 안정성이 높다. 해당되는 몇몇의 객체만을 선택해서 수정하면 다른 객체들은 아무렇지 않게 동작할 수 있는데 이는 객체의 단위가 본질적으로 다른 객체에 대한 결합도가 낮은 독립성이 강한 단위이기 때문이다.

② 단점

단점	설명
전문가의 부족	객체지향은 성숙된 다른 방식에 비해 아직 활성화 된 지 오래되지 않았기 때문에 전문적인 객체지향 사고와 기법을 갖춘 전문가가 부족하다. 많은 회사와 SW 개발관련자들이 아직 객체지향 기술과 먼 거리를 유지하고 있는 실정이고, 특히 객체지향 분석설계 및 개발 능력을 갖추고 있으면서 Architecture에 대한 전문적 지식을 가진 전문가는 드물다. 객체지향의 초심자가 처음 객체지향 기술을 도입하여 SW 시스템 개발을 진행하려고 할 때 도움을 받을 수 있는 전문가가 아직은 부족하다.
구현기술만 사용할때의 한계	객체지향 기술을 구현기술로만 적용하면 원래 의도한 대로의 체계적인 프로그램을 얻기 힘들다. SW에 대해 제대로 분석하고 제대로 설계하는 과정을 거쳐야만 객체지향 개념에 충실한 좋은 프로그램을 얻게 되는데 한계가 있을 수 있다.
다소간의 시스템성능 저하	동일한 환경에서 동일한 기능을 구현한 시스템이라면 객체지향을 적용한 시스템의 응답성능이 다소 떨어지는 것이 일반적이다. 이는 기존 프로그램은 직접문제를 해결하는데 비해, 객체지향 프로그램은 객체간의 협업에 의해 문제를 해결하기 때문이다. 객체와 객체간의 협업은 필연적으로 메시지 전달에 따른 지연이 발생하기 마련이긴 하지만 하드웨어 성능이 발전을 거듭하고 있어 이러한 작은 시간지연 문제는 무시해도 되는 쪽으로 가는 추세이다.

③ 요약

▶객체지향 개발 방법은 시스템의 확장이나 변경을 쉽게 할 수 있는 뛰어난 기법이다.

▶객체지향 분석기법은 기존의 분석기법이 가지고 있는 한계점을 극복하고 시스템의 3가지 관점을 체계적으로 통합하여 기술하고, 확장성과 적응력이 뛰어난 시스템을 설계하는 것이 그 목적이다.

▶객체지향 분석기법은 상향식 접근 방법(상속에 의해)이라 여겨질 수 있으며 이는 구조적 기법이 순수 하향식인 것과 대조

▶객체지향 분석의 결과는 설계로 넘어가 큰 변화 없이 사용되며, 앞의 세 모델의 결과가 통합되어 완벽한 객체와 클래스의 구현가능하다.

▶그러나 튼튼하고 견고한 시스템은 분석 과정에서 집중적으로 이루어지는 철저한 검증 과정과 이에 따른 수정 없이는 구현 불가능하다.

▶깊은 클래스 계층 구조가 있는 경우, 시스템 작동 중 객체에 대한 동적 할당과 제거시 성능에 대한 문제가 제기될 수도 있다.

▶객체지향 설계는 응용분야의 관점에서 컴퓨터 관점으로, 사용자 관점에서 개발자 관점으로 이동하여 객체지향 분석 결과를 종합

▶통신분야, 의료진단, 일기예보, 명령제어 분야 등에서 객체지향 개발방법이 적용되어 성공적인 시스템들이 만들어졌다.

▶데이터보다는 성능을 중요시하며, 많은 계산을 요하는 시스템에서는 적합지 않을 수 있다.

2. What about UML

1) Definition & History

① 정의

UML(Unified Modeling Language)은 기존의 Booch 방법론, Rumbaugh 등의 OMT(Object Modeling Technique), 그리고 Jacobson의 OOSE 방법론 등을 연합하여 만든 모델링을 위한 표준으로 객체지향 시스템 모델을 작성하기 위한 객체지향적 분석과 설계 개념의 표기법을 제공한다.
그러므로 UML 자체가 방법론이라 할 수는 없으나 방법론을 수행하기 위한 최적의 도구라 할 수 있다.

객체지향 개발 방법론의 특징 중 하나가 전통적 개발 방법론보다 분석, 설계에 할애하는 시간과 노력이 많다는 것은 기술된 바 있다. 그러므로 분석 설계 단계를 표준화 된 다이어그램과 모델링 툴을 이용해 수행하고 그 산출물이 객체지향 언어를 통해 체계적으로 번역되어 소스 코드가 생성될 수 있게 하는 것이 매우 중요하며 이것이 프로젝트 성공의 관건이라 할 수 있다.

UML은 Class Diagram, Object Diagram, Use Case Diagram 등 여러 Diagram을 기반으로 객체지향 어플리케이션을 개발하기 위한 분석 및 설계를 뒷받침하는 훌륭한 방식을 제공하고 있다.

시중에는 UML 적용을 효과적으로 하기 위한 객체지향 케이스 툴로 Paradigm Plus, ROSE, SELECT 등 이 있다.

② 역사

시간	갱신
1967년	- 최초의 객체지향 언어 SIMULA 탄생
1980년대와 90년대 초반	- 다양한 객체지향기반 표기체계 및 방법론(약 50여 가지)이 난립
1995년	- Booch의 Booch 방법론과 Rumbaugh의 OMT방법론이 통합 - 객체지향 학술대회인 OOPSLA('95)에 발표
1996년	- Jacobson의 OOSE 방법론이 추가로 통합 - 대표적인 3대 방법론에서 쓰이던 표기체계가 통합됨 - UML(Unified Modeling Language)로 명명되어 v0.9로 정의
1997년	- 1월 : UML ver. 1.0 발표 - 9월 : UML ver. 1.1 발표 객체지향 기술표준 기구인 OMG에 표준화안 상정 - 11월 : OMG 표준 인증
1999년	- UML ver. 1.3 발표
2001년	- UML ver. 1.4 발표
2002년	- UML ver. 2.0 예정

2) Characteristic and Composition of UML

① UML의 특징

가시화 언어	명세화 언어
- 개념 모델 - 오류 없이 전달 - 의사 소통의 용이 - graphic 언어	- 정확한 모델 제시 - 완전한 모델 작성 - 분석, 설계의 결정 - 표현
구축 언어	**문서화 언어**
- 다양한 프로그램, 언어와 연결 - 왕복 공학 가능(순공학/역공학) - 실행 시스템 예측 기능	- 시스템에 대한 통제, 평가, 의사소통의 문서 (요구사항, 아키텍처, 설계, 소스코드, 프로젝트 계획, 테스트, 프로토타입, 릴리즈)

▶가시화 언어(Language for Visualizing)

UML은 여러 개의 그래픽 기호로 구성되어 있으며 각 기호들은 정확한 의미를 가지고 있다. 그러므로 UML로 모델링한 것은 통일된 의미를 갖기 때문에 UML로 작성된 문서를 보는 사람들은 시스템에 대해 동일한 의미를 공유할 수 있다.

▶명세화 언어(Language for Specifying)

명세화란 정확하고, 명백하며, 완전한 모델을 의미하는데, UML에서는 분석, 설계, 구현에서의 모든 중요한 결정에 대한 명세서를 다루는 것이 가능하다.

▶구축 언어(Language for Constructing)

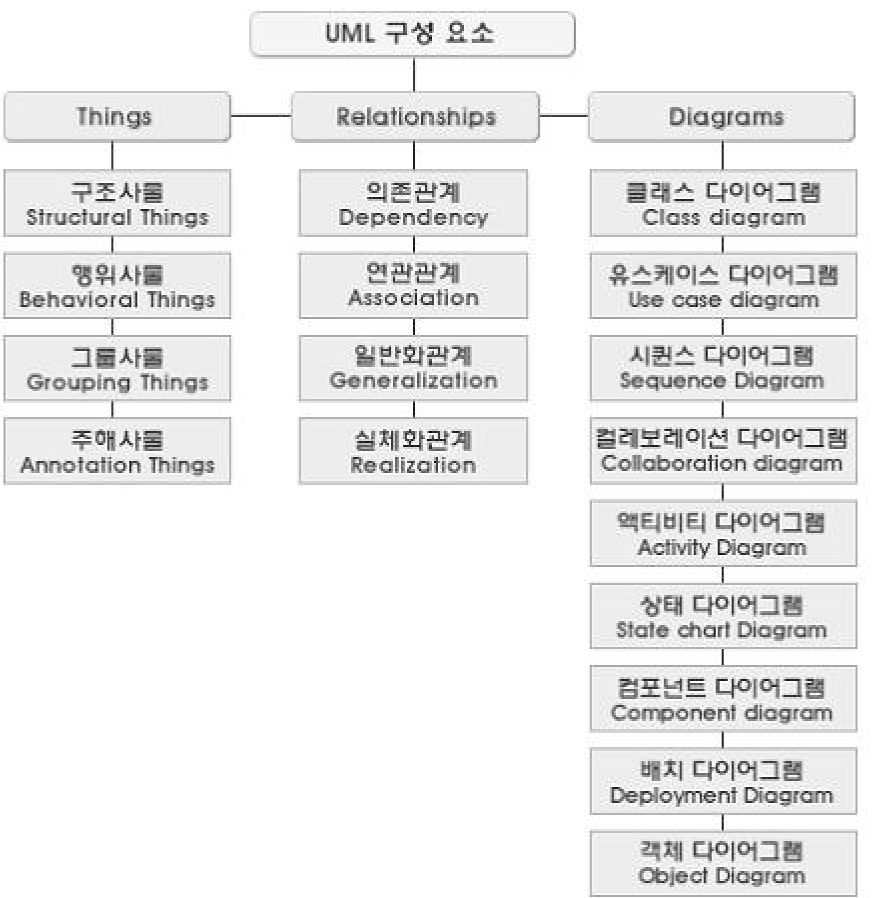
UML 언어에서는 프로그래밍 코드를 생성하는 것이 가능하고, 또한 구현된 코드로부터 UML 모델을 다시 생성할 수 있는 역공학 (reverse engineering)도 가능하다.

▶문서화 언어(Language for Documenting)

UML은 시스템 구조와 그것의 모든 상세 내역에 대한 문서화를 다루며, 요구사항을 표현하고 시스템을 시험하는 언어와 프로젝트 계획과 배포관리 액티비티[3]를 모델링하는 기능을 제공한다.

3) 일련의 활동

② 구성요소



<UML을 구성하는 요소들>

▶사물(Things)

추상적인 개념으로, 모델에서는 주제를 나타내는 요소이다. 언어로서의 UML을 생각해 볼 때 단어에 해당하며, 단어 중에서도 명사 혹은 동사의 의미를 가지는 요소라고 볼 수 있다.

UML을 그림으로 생각해 볼 때 Things는 도형의 형식을 가지는데 Things의 유형과 종류가 정해져 있어서 마음대로 도형을 추가할 수 없으며 이 Things는 사전에 부여된 명확한 의미를 가진다.

▶관계(Relationships)

Things의 의미를 확장하고 더욱 명확히 하는 요소이다.
즉 Relationships는 Things와 Things를 연결하여 그들 간의 관계를 표현하는 요소이다.
언어로서의 UML을 생각해 볼 때 Relationships는 단어에 해당하며, 단어 중에서도 형용사나 부사에 해당한다.

UML을 그림으로 생각해 볼 때 Relationships는 선의 형식을 가진다.
마찬가지로 UML에서는 정해진 선만을 사용해서 Relationships를 표현해야 하고 이 선들은 사전에 부여된 명확한 의미를 가진다.

▶도해(Diagrams)

Things과 Relationships을 모아 그림으로 표현한 것이다. Diagrams에는 Things와 Relationships가 어우러진 한 장의 그림으로 보는데 UML에서는 그 그림의 형식을 9가지로 정의하고, 각 그림에 대해 용도와 목적을 정의하고 있다.

보통 UML이라는 용어는 9개의 Diagrams와 동일한 의미로 쓰일 때가 많다.
언어로서의 UML을 생각해 볼 때 Diagrams는 주제를 담은 문장들로 이루어진 글월에 해당하고 이 글월은 문장(Things와 Relationships 몇 개가 합쳐진 형태)과 단어(Things, Relationships)들로 구성된다.

Diagram 한 장은 바로 하나의 모델을 의미한다.
따라서 UML은 9가지 종류의 모델 체계를 제시하고 있다고 할 수 있다.

③ 용어

▶사물(Things)의 종류

용어	설명
클래스 (Class)	의미를 공유하는 객체의 집합, 객체의 타입
인터페이스 (Interface)	class나 component의 기능 중 외부로 가시화 하는 부분을 정의
컬레보레이션 (Collaboration)	구현 관점에서 어떤 목적을 달성하기 위한 일련의 행위를 정의
Use Case (Use Case)	시스템이 제공하는 서비스 혹은 기능
액티브 클래스 (Active Class)	클래스에서 파생된 객체가 하나 또는 그 이상의 프로세스나 스레드[4]를 갖는 클래스를 기술
컴포넌트 (Component)	시스템에서 물리적으로 관리되는 한 묶음의 소프트웨어를 표현
노드 (Node)	연산 능력이 있는 물리적 요소를 표현. 컴퓨터나 네트워크 기기 등을 의미

▶관계(Relationships)의 종류

용어	설명
의존 (Dependency)	한쪽 사물의 변화가 다른 사물에 영향을 주는 관계 주로 일시적으로 의존함을 의미
일반화 (Generalization)	객체의 특성 중 상속을 표현하는 관계
연관 (Association)	사물들 간의 일반적인 참조 관계 구조적 관계이며, 지속적으로 참조함을 의미
실체화 (Realization)	정의하는 사물과 이를 구현하는 사물 간에 표현하는 관계

4) 둘 이상의 프로세스가 동시에 진행

▶도해(Diagrams)의 종류

용어	설명
UseCase Diagram	- 시스템이 제공하는 서비스와 외부 환경과의 관계를 표현 - 시간개념과 순서개념이 없으므로 정적인 관점에서의 모델 - 사용자 관점에서 시스템의 기능을 정의하고 외부 환경정의 목적
Class Diagram	- 클래스와 클래스 간의 관계를 나타내며 UML의 모델 가운데 가장 보편적 - 직접적으로 프로그래밍과 관련 - 시스템의 정적인 관점
Sequence Diagram	- 외부의 특정한 처리요청을 해결하기 위해 필요한 객체들과 그 객체들이 참여한 시간적, 순서적 처리흐름을 표현 - 시스템의 동적인 관점을 나타내며, 시스템의 동적 모델 중 하나
Collaboration Diagram	- Sequence Diagram과 목적과 용도가 같음 - Sequence Diagram이 시간순서를 중시한 모델인 반면 객체와 메시지를 구조적으로 표현하는 데 유리한 표현체계 - 두 다이어그램은 표현형태만 다를 뿐이어서 서로 의미의 손실 없이 변환이 가능 - 시스템의 동적인 관점을 나타내며, 시스템의 동적 모델 중 하나
State Chart Diagram	- 하나의 객체가 생성되어 소멸될 때까지 가질 수 있는 가능한 모든 상태(state)를 분석하고, 표현 - 시스템에서 복잡한 역할을 수행하는 핵심 객체에 대해 자세한 변화를 추적하여 완전성을 기하기 위해 작성 - 시스템의 동적인 관점을 나타내며, 시스템의 동적 모델 중 하나
Activity Diagram	- 처리흐름을 모델링하는 범용적인 다이어그램 - 대상은 클래스의 처리 흐름일 수도있고, 비즈니스측면의 워크플로우 일 수도 있으며, 기타 다른 다양한 분야가 대상이 될 수 있음 - 논리 흐름과 처리 순서, 프로세스 플로우 등에 대해 판단, 처리, 액티비티를 사용하여 분석
Component Diagram	- 클래스로 구성된 물리적인 배치 단위인 컴포넌트와 컴포넌트간의 구성과 의존관계를 나타냄 - 컴포넌트는 컴퓨터 장치에 독립적으로 배치할 수 있는 단위 - 시스템의 정적인 구현관점을 표현
Deployment Diagram	- 시스템이 실행되는 환경인 노드와 그 노드에 배치된 컴포넌트의 구성 - 컴포넌트 다이어그램과 관련이 있는데, 이는 일반적으로 하나의 노드는 Component Diagram에 정의된 컴포넌트를 수용하기 때문
Object Diagram	- 특정 시점에서의 객체들의 상태와 그들 간의 관계를 표현 - Class Diagram에 있는 요소들의 인스턴스에 대한 정적인 스냅 샷 표현

-Use Case Diagrams

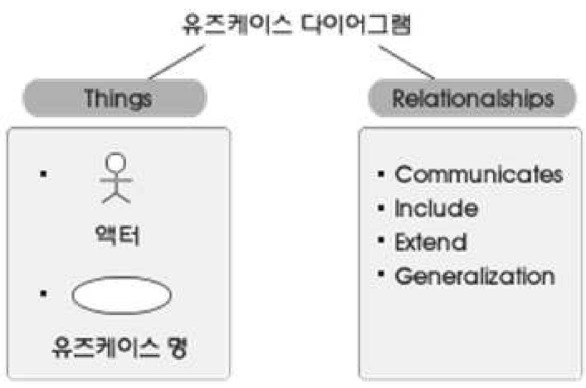
* Use Case Diagram은 다른 Diagram보다 먼저 작성되어야 한다.

Use Case Diagram을 최우선으로 작성해야 하는 이유
- Use Case 다이어그램은 대개의 경우 다른 다이어그램들보다 먼저 작성됨
- Use Case 다이어그램은 다른 다이어그램을 작성할 때의 기준이 됨
- 다른 다이어그램들은 Use Case 다이어그램에서 정의한 내용을 보다 상세하고 구체적으로 추가 정의하는 형태로 작성될 때가 많음

* Use Case Diagram은 다음의 목적으로 작성된다.

Use Case Diagram 작성 이유
- SW 시스템의 업무범위를 정의
- SW 시스템을 사용하게 되는 사용자를 정의
- SW 시스템의 업무기능을 정의
- 사용자 요구사항을 정의
- 사용자와 개발팀간의 의사소통의 도구로서의 기능
- 이후 수행할 분석,설계 작업의 기준
- 테스트의 기준

* Use Case Diagram의 구성요소



용어	설명
액터 이름	- 왼쪽과 같은 사람모양의 그림으로 표기(스틱맨)
	- 스틱맨 아래에 액터의 이름을 명사로 표시 단 조직, 개인의 이름이 아닌 역할(Role)을 작성
	- SW 사용자, SW System, 하드웨어
유즈케이스 명	- 왼쪽과 같은 타원 모양의 그림으로 표기
	- Use Case 아래에는 Use Case 명을 짧은 문장 혹은 행위를 나타내는 명사형으로 표시
	- 경우에 따라 Use Case 명을 타원 안에 표기

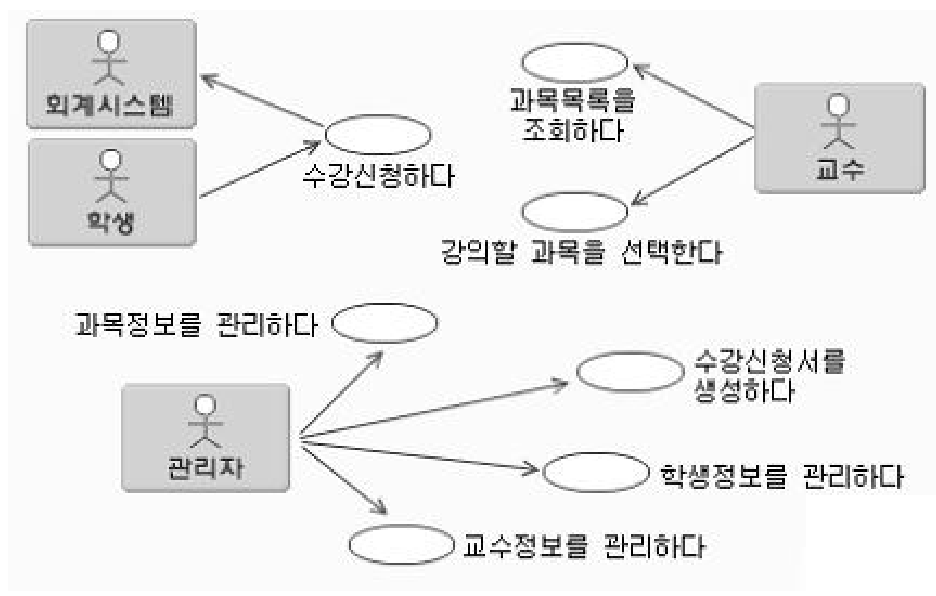
* Use Case Diagram의 표기법

관계	모델
extend	- Use Case와 Use Case 사이에 정의되는 관계
	- 한 Use Case가 다른 Use Case의 서비스 수행을 요청하는 관계
	- 수행을 의뢰하는 Use Case는 조건에 따라 수행을 의뢰할 수도 있고 의뢰하지 않을 수도 있음
include	- Use Case와 Use Case 사이에 정의되는 관계
	- 한 Use Case가 다른 Use Case의 서비스 수행을 요청하는 관계
	- 이 경우 수행요청을 의뢰받은 서비스는 반드시 수행
generalization	- 액터와 액터, Use Case와 Use Case 사이에 정의되는 관계
	- 두 개체가 일반화 관계에 있음을 의미
	- 상속(inheritance)의 특성을 가짐
communicates	- 액터와 Use Case 사이에 정의되는 관계
	- 두 개체가 일반 상호작용 관계에 있다는 것을 의미

* Use Case Diagram의 작성 단계

활동	설명
액터 식별	- 시스템의 모든 사용자의 역할을 식별
	- 시스템과 상호작용하는 타 시스템 식별
	- 시스템과 정보를 주고받는 하드웨어나 지능형 장치 식별
Use Case 식별	- 액터가 요구하는 서비스를 식별합니다.
	- 액터가 요구하는 정보를 식별합니다.
	-액터가 시스템과 상호작용하는 행위를 식별합니다.
관계 정의	-액터와 액터간의 관계를 분석하고 정의(generalization 관계)
	-액터와 Use Case간의 관계를 분석하고 정의(communicates)
	- Use Case와 Use Case 간의 관계를 분석하고 정의 (include, extend, generalization)
Use Case 구조화 (factoring)	- 두 개 이상의 Use Case에 존재하는 공통 서비스 추출
	- 추출된 서비스를 Use Case로 정의
	- 공통 서비스를 사용하는 Use Case와 신규 Use Case의 관계를 정의(include)
	- 조건에 따른 서비스 수행부분을 분석하여 추출
	- 추출된 서비스를 독립 Use Case로 정의
	- 추출된 서비스를 사용하는 Use Case와 관계를 정의(extend)

* Use Case Diagram의 예



-Class Diagrams

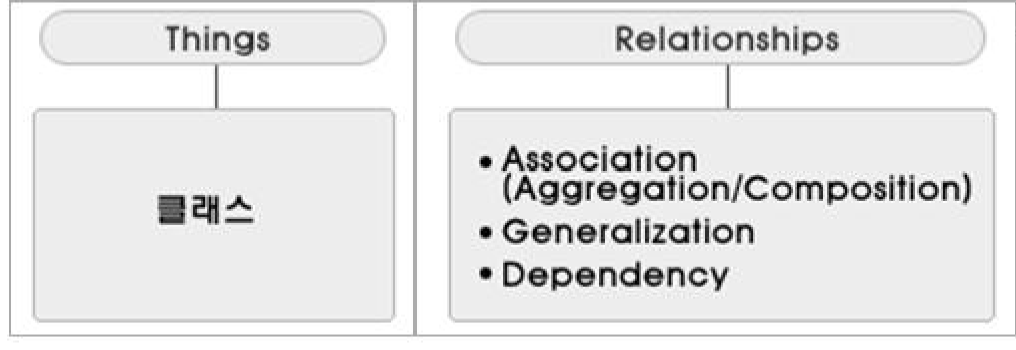
* Class Diagram은 다음의 목적으로 작성된다.

Class Diagram 작성 이유
- 객체지향 SW시스템의 기본 단위인 클래스를 식별하고 그 관계를 정의하는 유용한 방식을 제공
- 클래스 간의 정적인 협력관계를 정의함으로써 시스템을 이해하는데 용이
- 클래스의 오퍼레이션과 속성을 상세히 정의함으로써 SW시스템을 설계
- 논리적인 관점으로부터 물리적인 관점까지 시종 일관된 형식으로 SW 시스템을 분석, 설계하는 방식을 제공합니다.

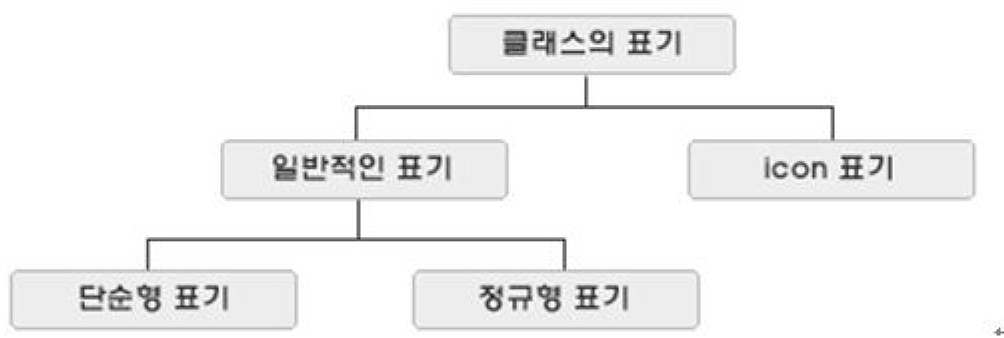
* Class Diagram은 System Development의 분석, 설계단계에서 여러 번 작성된다.

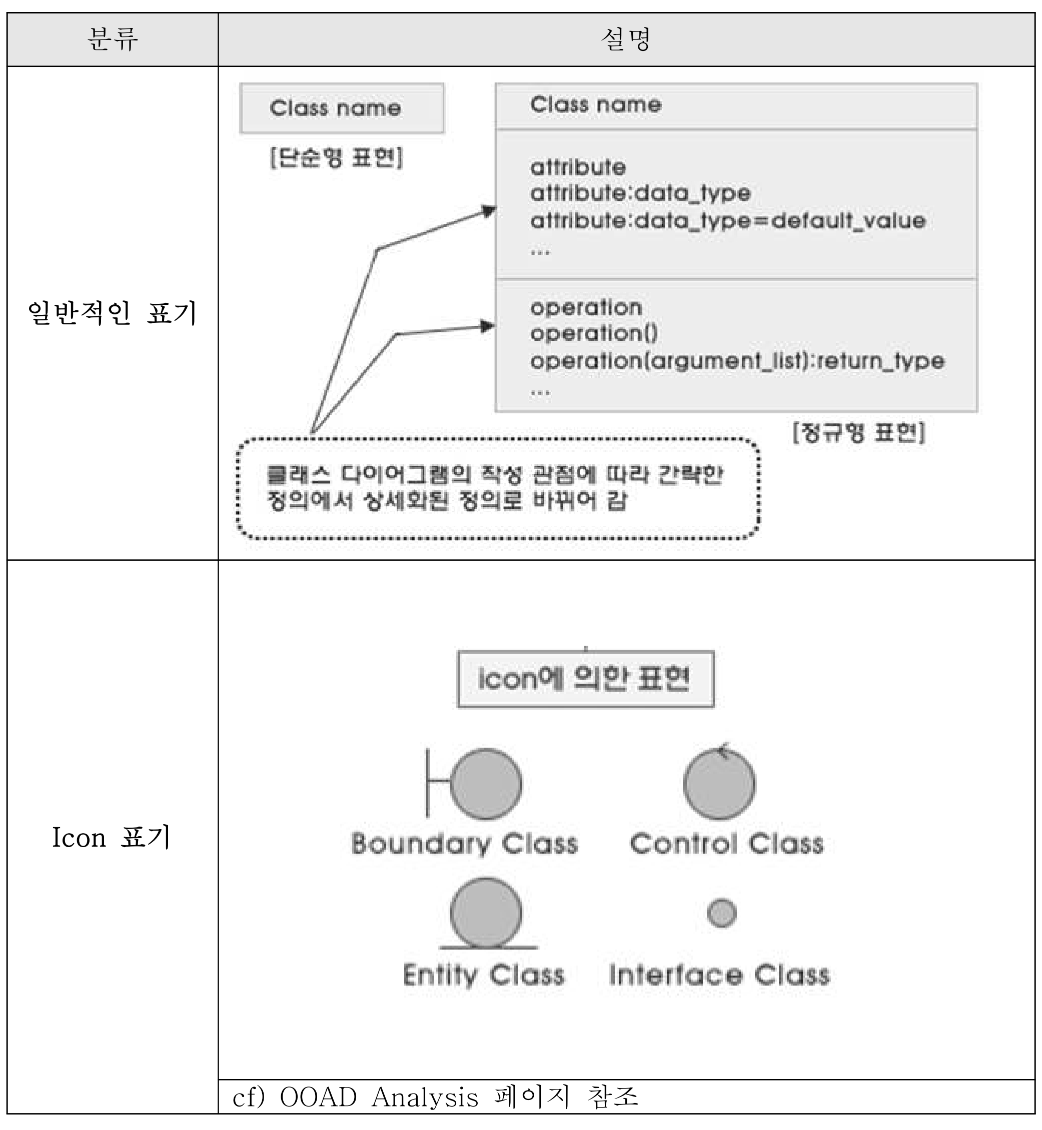
<일반적인 Class Diagram의 작성 시기>

* Class Diagram의 구성요소



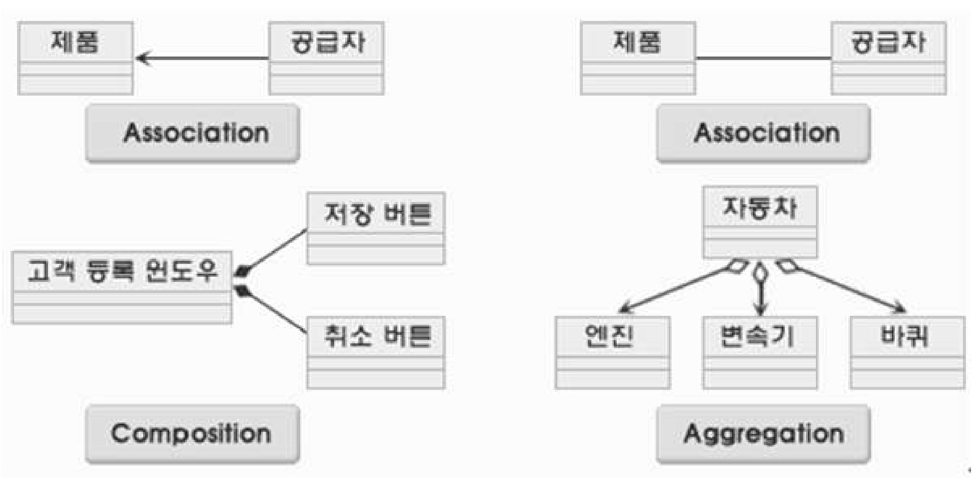
* Class의 표기 방법

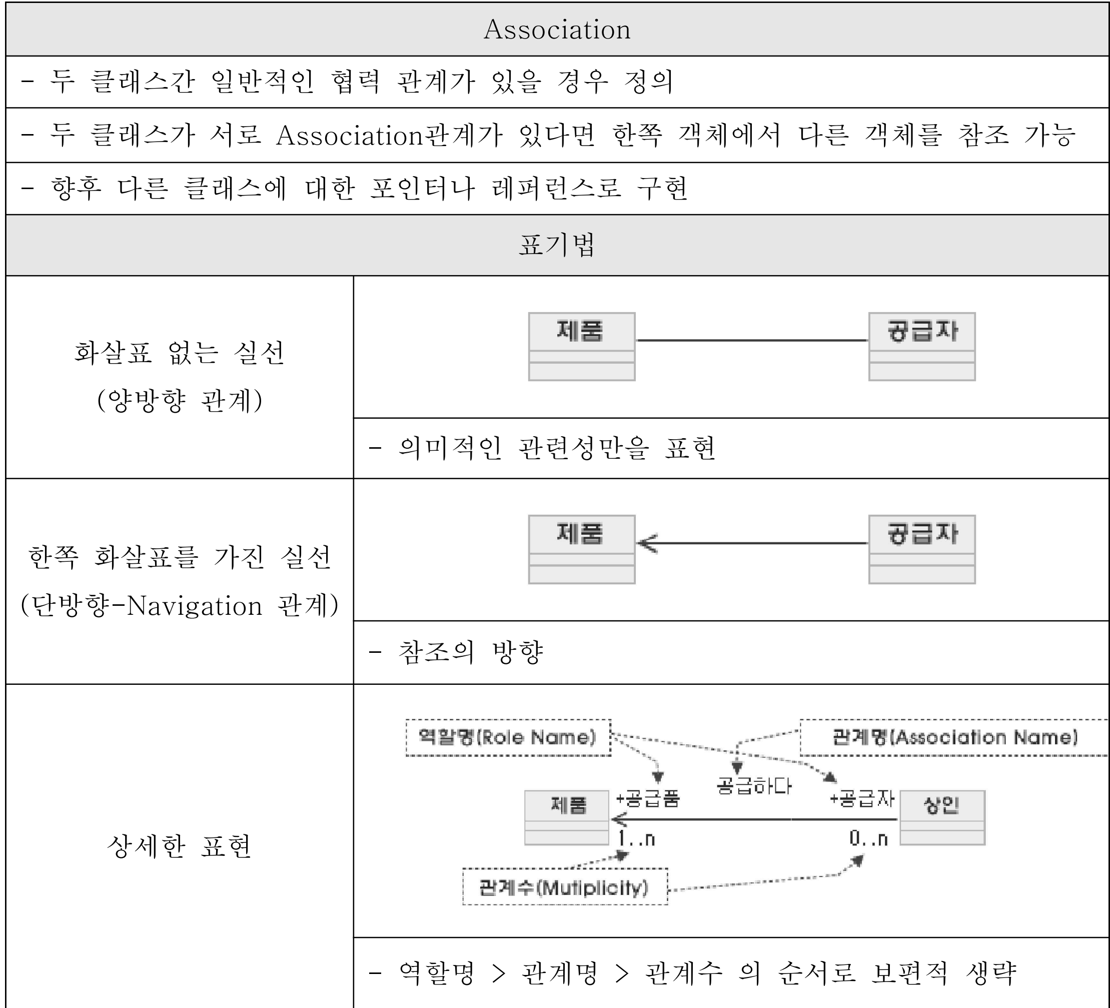




분류	설명
일반적인 표기	Class name [단순형 표현] Class name attribute attribute:data_type attribute:data_type=default_value ... operation operation() operation(argument_list):return_type ... [정규형 표현] 클래스 다이어그램의 작성 관점에 따라 간략한 정의에서 상세화된 정의로 바뀌어 감
Icon 표기	icon에 의한 표현 Boundary Class Control Class Entity Class Interface Class
	cf) OOAD Analysis 페이지 참조

* Class Diagram의 표기법 -1



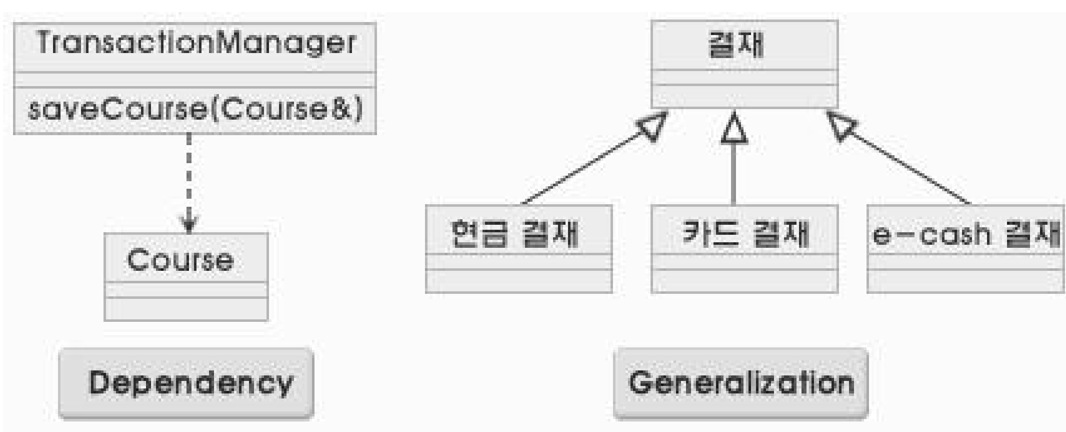


Association	
- 두 클래스간 일반적인 협력 관계가 있을 경우 정의	
- 두 클래스가 서로 Association관계가 있다면 한쪽 객체에서 다른 객체를 참조 가능	
- 향후 다른 클래스에 대한 포인터나 레퍼런스로 구현	
표기법	
화살표 없는 실선 (양방향 관계)	제품 ――― 공급자
	- 의미적인 관련성만을 표현
한쪽 화살표를 가진 실선 (단방향-Navigation 관계)	제품 ←―― 공급자
	- 참조의 방향
상세한 표현	역할명(Role Name), 관계명(Association Name), 제품, +공급품, 공급하다, +공급자, 상인, 1..n, 0..n, 관계수(Mutiplicity)
	- 역할명 > 관계명 > 관계수 의 순서로 보편적 생략

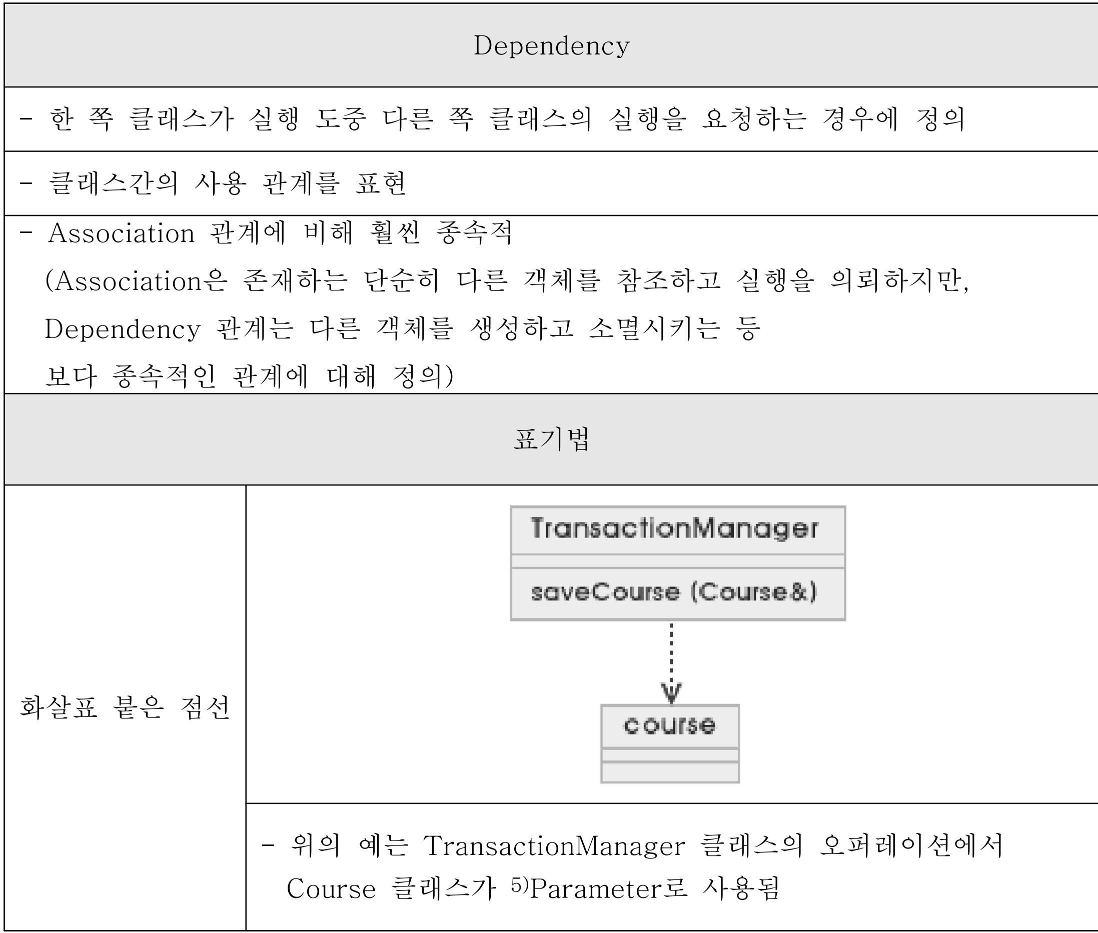
Aggregation	
- Association관계의 일종	
- 두 클래스 간에 "전체-부분(whole-part)"의 관계가 있을 경우 정의	
- 클래스 각각이 독립적인 생명주기를 가지는데 즉 전체에 해당되는 클래스가 소멸되더라도 부분에 해당되는 클래스는 소멸하지 않고 계속 존재할 수 있음	
표기법	
속이 빈 마름모가 붙은 실선	자동차 / 엔진 / 변속기 / 바퀴 (1, 1, 1, 1, 1, 4)
	- 마름모가 붙은 쪽이 "전체" 클래스, 반대쪽은 "부분" 클래스

Composition	
- Association 관계의 일종	
- Aggregation 관계와 유사하게 두 클래스 간에 "부분-전체" (part-of)의 관계가 있을 경우 정의	
- 부분의 생명주기가 전체의 생명주기에 종속적인 관계인데 즉 전체가 생성될 때 부분도 생성되고, 전체가 소멸될 때 부분도 함께 소멸	
표기법	
속이 찬 마름모가 붙은 실선	고객 등록 윈도우 / 저장 버튼 / 취소 버튼
	- Aggregation관계와 "전체-부분"의 의미는 동일

* Class Diagram의 표기법 -2



<table>
<tr><th colspan="2">Generalization</th></tr>
<tr><td colspan="2">- 두 클래스는 일반화-특수화 관계가 있을 때 정의하는데 즉 보다 보편적인 것과 보다 구체적인 것 사이의 관계(is-a 관계)</td></tr>
<tr><td colspan="2">- 상속(inheritance)의 특성</td></tr>
<tr><th colspan="2">표기법</th></tr>
<tr><td rowspan="2">삼각형 화살표가 붙은 실선</td><td>결재
현금 결재 카드 결재 e-cash 결재</td></tr>
<tr><td>- 결재 클래스의 속성과 오퍼레이션은 공통적이고 일반적인 것으로 정의되고, 하위의 3개의 클래스는 자신만의 특수한 속성과 오퍼레이션을 정의함</td></tr>
</table>



Dependency	
- 한 쪽 클래스가 실행 도중 다른 쪽 클래스의 실행을 요청하는 경우에 정의	
- 클래스간의 사용 관계를 표현	
- Association 관계에 비해 훨씬 종속적 (Association은 존재하는 단순히 다른 객체를 참조하고 실행을 의뢰하지만, Dependency 관계는 다른 객체를 생성하고 소멸시키는 등 보다 종속적인 관계에 대해 정의)	
표기법	
화살표 붙은 점선	TransactionManager saveCourse (Course&) course
	- 위의 예는 TransactionManager 클래스의 오퍼레이션에서 Course 클래스가 [5]Parameter로 사용됨

*Class Diagram의 표기법 -3

유형	설명
Many (* 또는 n)	- 클래스 B의 인스턴스 하나에 A 인스턴스 여러 개와 관계 A * B
Exactly five(5)	- 클래스 B의 인스턴스 하나에 A 인스턴스 다섯 개와 관계 A 5 B
Zero or more (0..n)	- 클래스 B의 인스턴스 하나에 관계된 A 인스턴스가 없거나 여러 개 A 0..n B

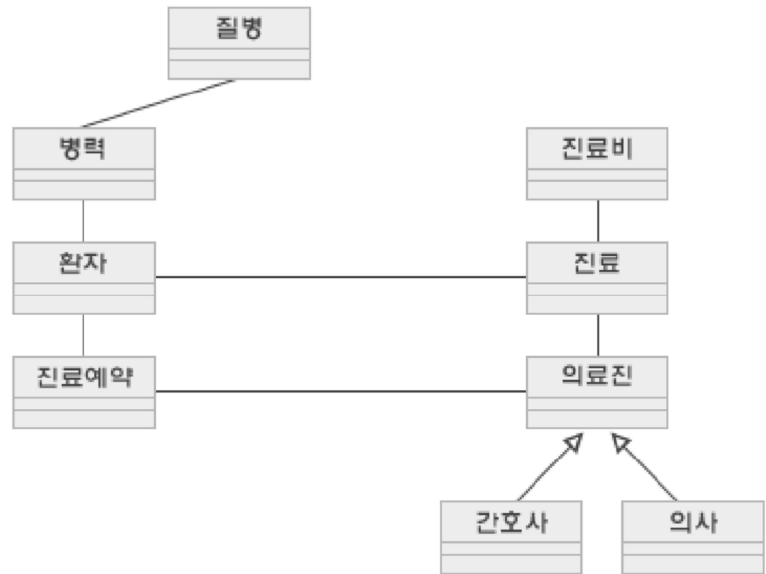
5) Method의 매개변수(Argument)

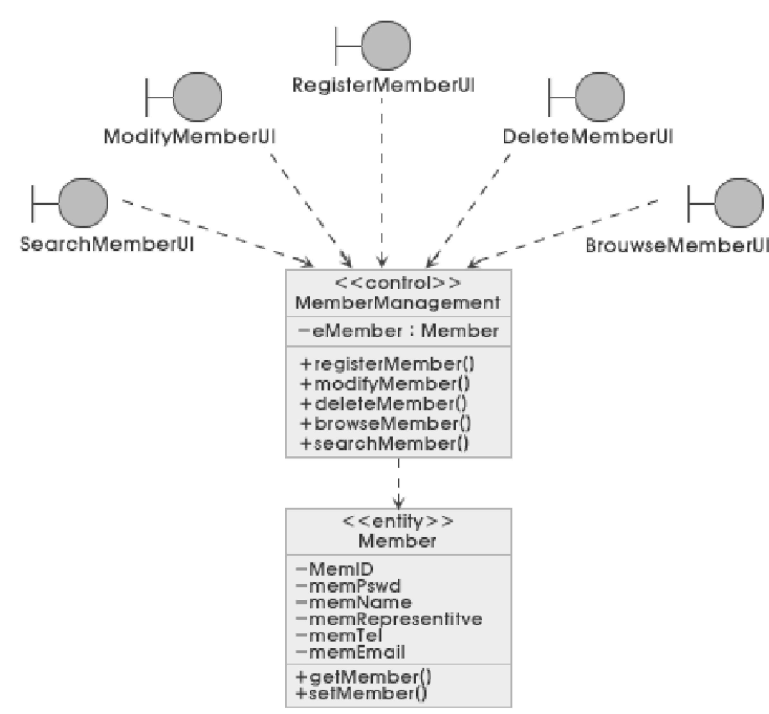
One to Ten (1..10)	- 클래스 B의 인스턴스 하나에 관계된 A 인스턴스가 1개보다 많고 10개 보다 적음 A —1..10— B
Exactly two, three or five (2,3,5)	- 클래스 B의 인스턴스 하나에 관계된 A 인스턴스가 2개 혹은 3개 혹은 5개임 A —2,3,5— B

* Class Diagram의 작성 단계

활동	설명
객체 식별 클래스를 정의	- 사용자 문서나 Use Case 정의서, 문제 기술서등을 참고하여 객체를 식별
	- 식별된 객체를 바탕으로 클래스를 정의
	- 이 단계에서는 클래스 명 정도만 표현
속성 & 오퍼레이션 정의	- 클래스의 속성과 오퍼레이션을 정의
	- 한 번에 상세한 정의를 마치지 못하므로 여러 번 정제과정을 거쳐야 함
클래스간 관계 정의	- 클래스와 클래스간 관계를 정의
	- 관계의 종류를 결정하고, 관계 명을 부여
	- 관계 수를 정의
정제과정을 반복	- Class Diagram은 분석과 설계 과정에서 지속적으로 정제되어야 함
	- 다른 모델을 작성하는 과정에 새로운 클래스가 추가되기도 하고, 관계가 변경되기도 함
	- 이러한 변화와 상세화 과정을 Class Diagram에 반영

* Class Diagram의 예



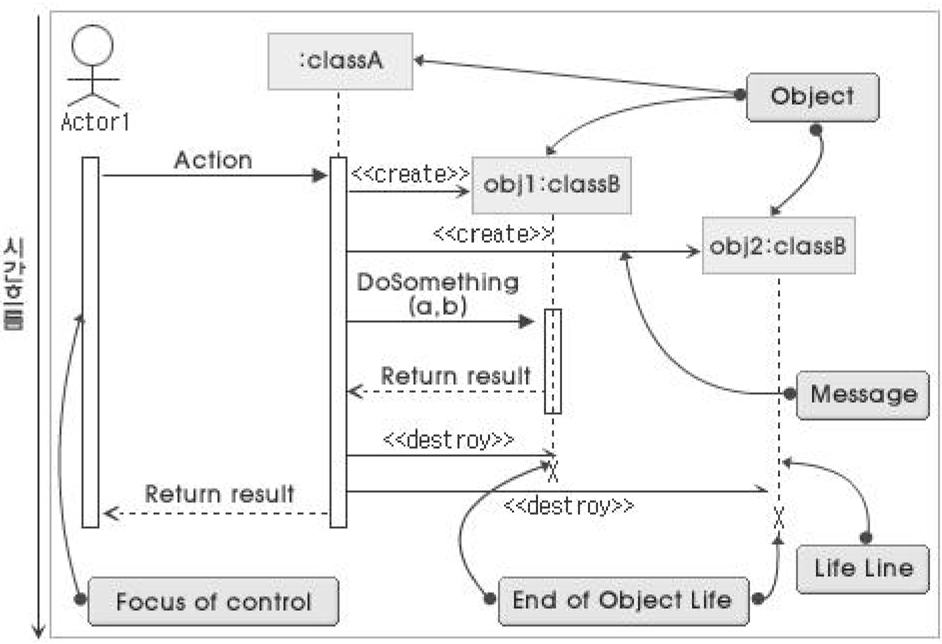


-Sequence Diagram

* Sequence Diagram은 다음의 목적으로 작성된다.

Sequence Diagram 작성 이유
- 객체간의 동적 상호작용을 시간적 개념을 중시하여 모델링
- 객체의 오퍼레이션과 속성을 상세히 정의합니다.
- Use Case를 실현(Realization)
- 프로그래밍 사양을 정의

* Sequence Diagram의 구성요소

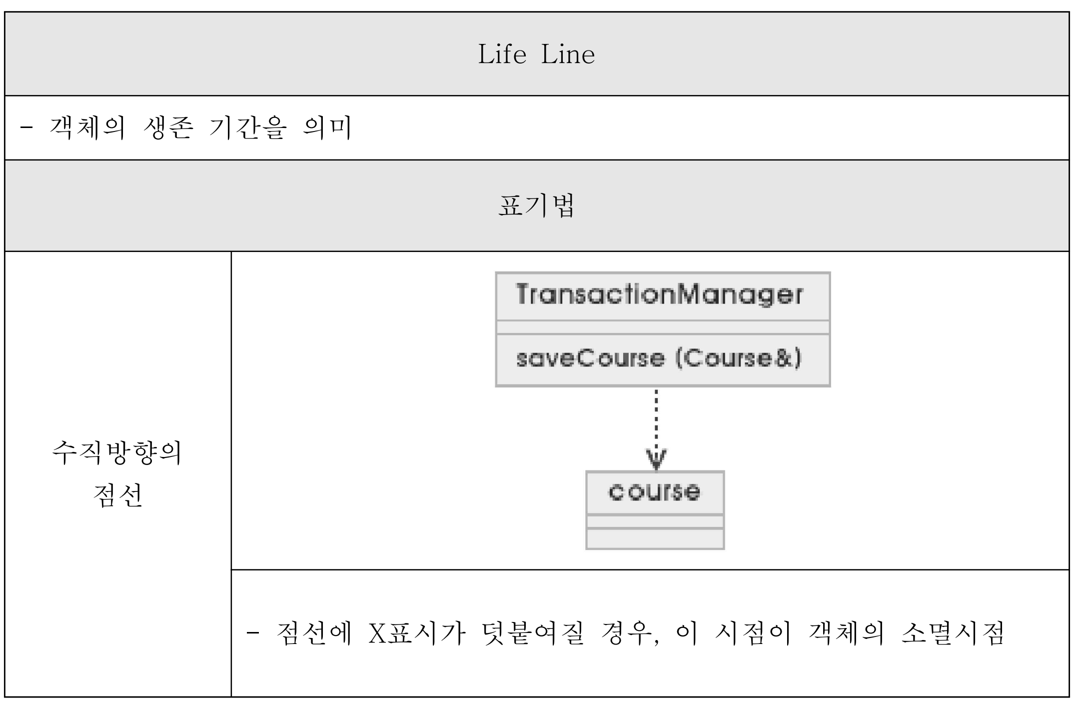


분류	성분
Things	- 액터(Actor), 객체(Object)
Relationships	- 메시지(Massage)
기타	- Life Line , Focus of control

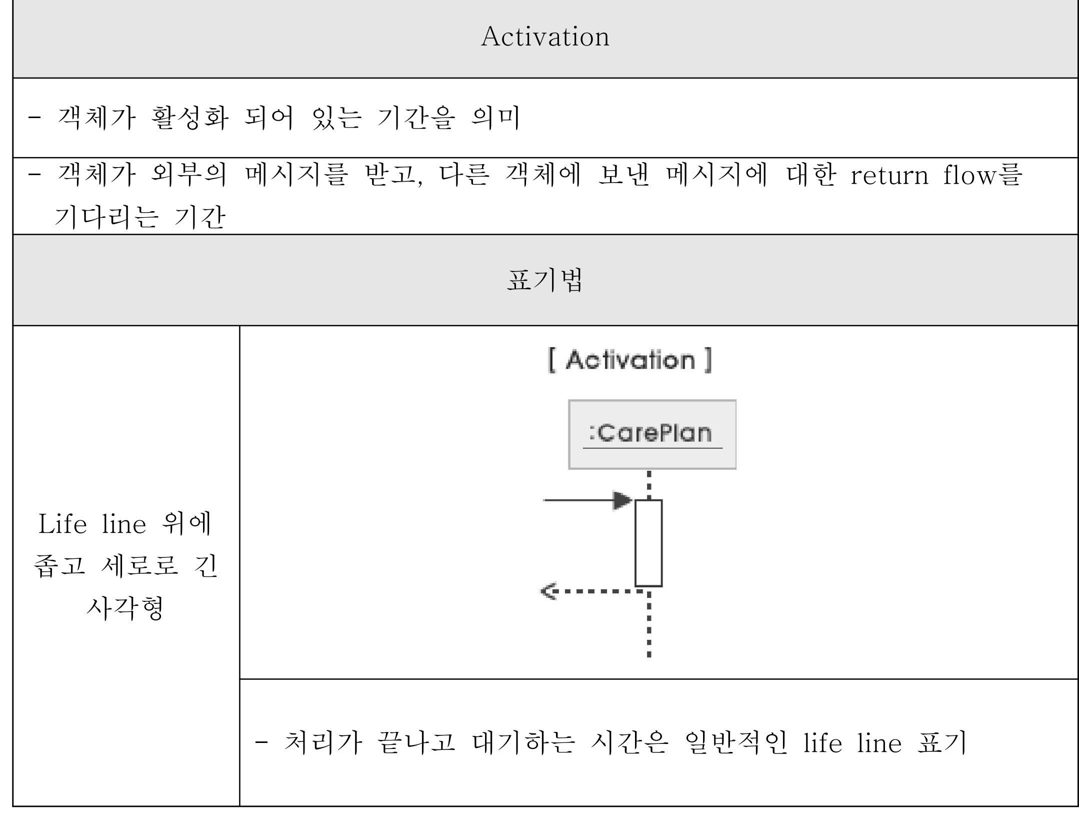
* Sequence Diagram의 표기법 -1

메세지	표기법
Flat flow of control	:CarePlan - 객체에 메시지를 연결할 때 사용
Nested flow of control	:CarePlan - 메시지의 결과가 돌려지게 될때까지 다음 처리를 진행하지 않는 Synchronous 메시지
Asynchronous flow of control	:CarePlan - 객체가 보낸 메시지의 결과를 기다리지 않고 다음 처리를 진행
Return flow	:CarePlan - 메시지를 처리한 결과(return)

* Sequence Diagram의 표기법 -2



Life Line	
- 객체의 생존 기간을 의미	
표기법	
수직방향의 점선	
	- 점선에 X표시가 덧붙여질 경우, 이 시점이 객체의 소멸시점

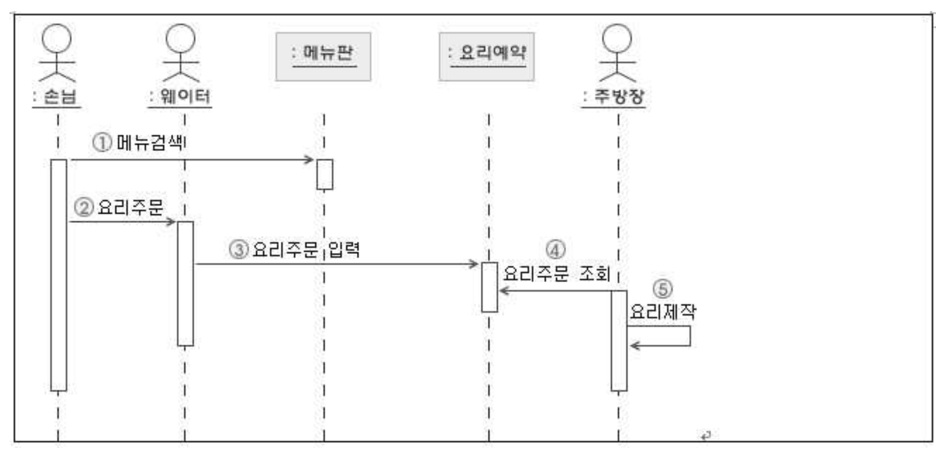


Activation	
- 객체가 활성화 되어 있는 기간을 의미	
- 객체가 외부의 메시지를 받고, 다른 객체에 보낸 메시지에 대한 return flow를 기다리는 기간	
표기법	
Life line 위에 좁고 세로로 긴 사각형	
	- 처리가 끝나고 대기하는 시간은 일반적인 life line 표기

* Sequence Diagram의 작성 단계

활동	설명
작성 대상을 선정	- Use Case 다이어그램을 이용하여 시퀀스 다이어그램의 작성대상을 선정한 후, 하나의 Use Case를 선택하고, Use Case 정의서를 분석
Use Case의 액터을 파악하여 Sequence Diagram에 위치	- 액터가 둘 이상일 경우라도 모두 좌측부터 액터를 위치시켜야 합니다. 순서는 중요하지 않지만 메시지 선이 적게 교차하도록 배치
Use Case를 실현하기 위해 참여할 클래스(객체)들을 정해 Sequence Diagram에 위치	- 정의된 클래스 중에 Use Case의 처리에 참여하는 것들을 식별하고 Sequence Diagram에 위치시켜야 하지만 순서는 중요하지 않음
시간 순서를 감안하여 객체간 메시지를 정의	- Use Case를 실현하기 위해 필요한 객체들 간의 메시지를 정의하되, 시간 순서에 유의 (시간흐름은 위에서 아래로)
필요한 객체를 추가로 정의	- 요구된 처리를 위해 필요하지만 아직 정의되지 않은 객체를 새로 정의하여 시퀀스 다이어그램에 추가해 나가며 또 이렇게 추가된 객체 사이의 메시지도 정의하여 추가

* Sequence Diagram의 예



-Collaboration Diagrams

* 객체들 사이의 행위를 나타내는 것은 sequence diagram과 동일하다. 둘의 차이점은 sequence diagram이 시간에 따른 행위의 흐름에 역점을 두고 표현하지만 collaboration diagram의 경우 객체들 사이의 정적인 구조에 더 역점을 두고 있다.

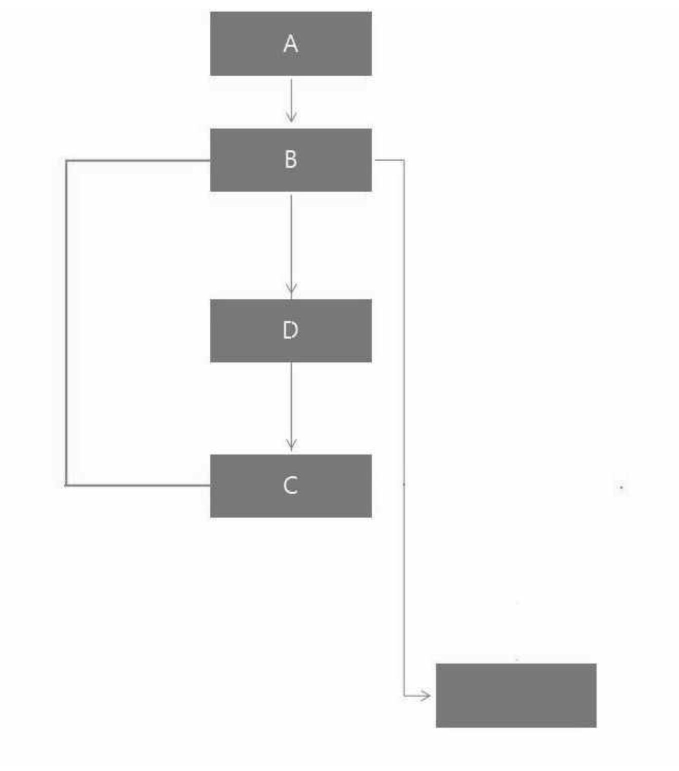
-Active Diagrams

* 동적 측면의 Diagram으로 시스템에서 발생하는 활동을 강조한다. (순서도의 일종)

- Data의 활동, 흐름, 또는 과정 사이사이의 의사 결정을 표현
- 하나의 사례에서 일어나는 활동들을 분할하여 표현하기 위해 사용
- 시스템의 기능을 모형화
- 객체간의 제어 흐름을 표현
- 순서나 병렬적인 처리를 필요로 하는 행동들을 표현

* Active Diagram의 예

-> For(A; B; C;) { D; }



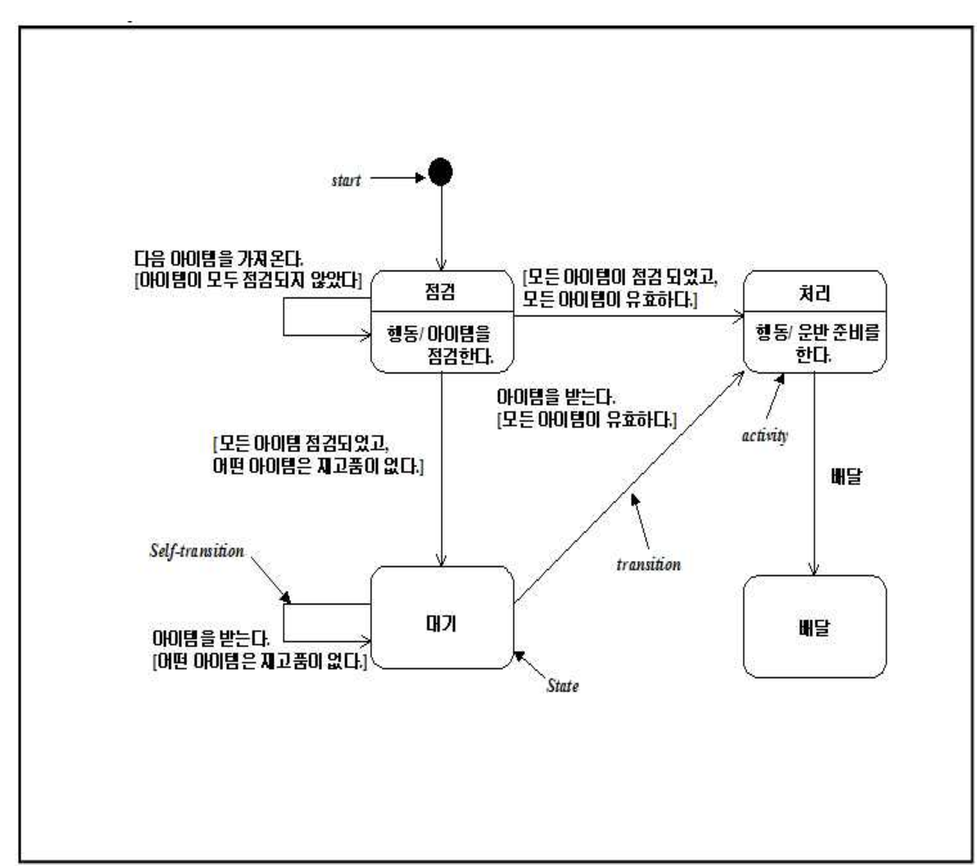
< Activity Diagram 으로 표현한 For문(반복문) >

-State Chart Diagrams

* 시스템의 동적 측면의 Modeling이다.

- 어떤 객체가 갖는 상태에 초점을 맞춤
- 순차적으로 발생하는 '행동'에 중점을 둠
- State Diagram에서 객체의 초기상태를 나타내는 Starting-State는 오직 한 개만 존재
- 단, Ending-State는 여러 개로 존재 가능

* State Chard Diagram의 예



-Component Diagrams

* Component에 초점을 두고 행하는 Diagram이다.

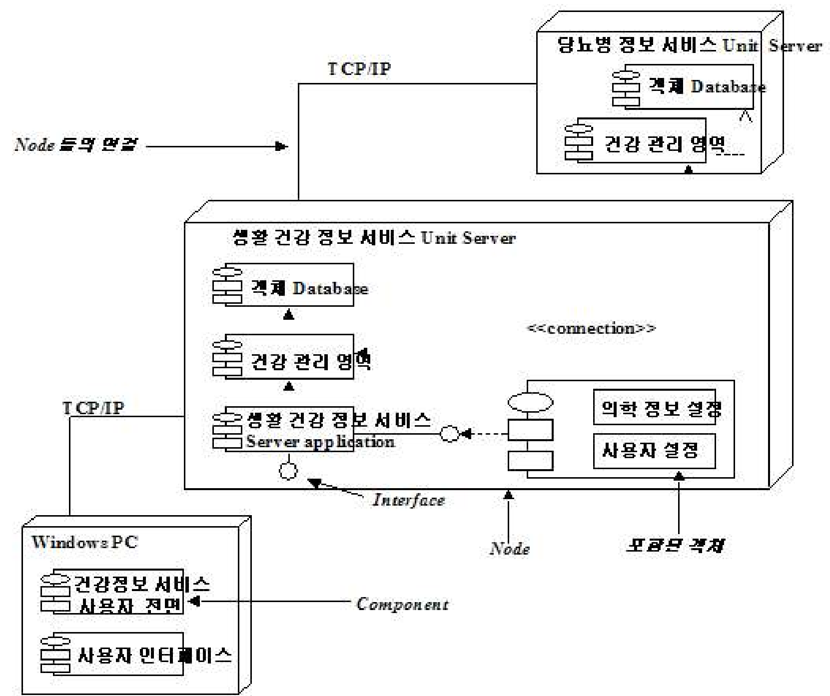
- 실행 Node에서 실행 가능한 Component를 명세
- Interface 자체적으로 실행이 가능한 Component의 집합을 표현
- 물리적인 구성단위로 이루어진 Component 간의 구성, 관계를 표현
- 시스템의 Logical-Model을 개발자의 관점에서 바라본 Physical-Model로 재배치할 때 사용
- Class를 어떤 File에 넣어서 모듈을 만들어 낼 것인지를 정의

-Deployment Diagrams

* Node들의 집합과 그들 사이의 연관관계를 보여주는 Diagram이다.

* 아키텍처의 정적-관점(Static-View)을 표현한다.

* Deployment Diagram의 예



-Object Diagram의 예

* 특정 객체들이 Class Model 안에서 사용되어지는 방법을 나타내기 위해 Class Diagram에 추가적으로 객체들 사이의 관계를 표현하여 나타낸 Diagram이다.
* 객체들의 상호 작용을 정의한다.

3) Summary of UML

① 장점

장점	설명
모델에 관한 공통 언어	- 동일한 용어로 이야기하기 때문에 그만큼 커뮤니케이션 비용 (정보 교환에 드는 비용)을 줄일 수 있음 - 비용을 줄인다는 것은 프로젝트 전체의 품질 향상과 직결
전세계의 표준	- 객체지향이 널리 퍼짐에 따라 사실상 UML이 전세계 표준으로 사용 - 그러므로 전세계에 존재하는 정보나 노하우 습득 용이

② 단점

단점	설명
부적절한 도입	- 개발 작업에 아무런 영향을 주지 않는 모델, 만든 후에도 참조되지 않는 모델링 가능성 존재 - 그러므로 작업 시간만 소비
모델링이 어려움	- 모델러(모델을 만드는 사람)의 모델화 능력에서 비롯되는 문제 - 모델은 작성 목적을 달성할 수 있을 만큼의 품질이 요구되므로 숙달된 모델러에게만 유용할 수 있음

③ 요약

▶ 기존에 사용되던 안정되고 검증된 표기체계들을 이어받고 통합한 (Unified) 언어(Language)이다.

▶ UML사용 시 다음 사항을 필히 체크해야 한다.

□목적(독자)을 확인하고 있는가

□명명 시 추상도, 정확성, 표현의 통일은 확인했는가

□규모는 일정한가

□기능 분할로 되어 있지는 않는가

□<<include>>, <<extend>>, 일반화 관계의 사용법이 잘못되어 있지는 않은가

▶ UML은 언어이므로 단어와 문법이 있다.

▶ 적절한 Diagram을 선택하여 작성하여야 한다.

▶ 모델링은 최대한 실제와 가깝게 해야한다.