Introduction to OOAD using UML tools

소프트웨어공학개론 A 반 5 조

200611495 유상원

200611496 유승현

200611512 임재식

200611524 허준호

목 차

1. UML -- 4

1-1. UML 개요 -- 4

1-2. 모델링과 모델이란? --6

1-3. UML 의 구성요소 --7

1-3-1. Things --7

1-3-2. Relationships --10

1-3-3. Diagram --- 14

2. OOAD --- 24

2-1. 객체지향의 기본 개념 ----------------------------------- 25

2-2. 객체지향 분석의 관점 ----------------------------------- 26

2-3. 객체 모델링 --27

2-3-1. 객체와 클래스 --27

2-3-2. 클래스의 관계 --28

2-3-3. 매핑 제약조건 --29

2-3-4. 클래스의 일반화 --- 30

2-3-5. 다중 상속 -- 31

2-3-6. 집단화 ---32

2-4. 동적 모델링 --- 33

2-5. 기능 모델링 --- 36

2-6. 객체지향 설계 --37

3. UML tool 을 사용한 OOAD 의 구현 ---------------- 38

3-1. 식당 관리 프로그램 ------------------------------------- 38

3-1-1. Use Case Diagram --39

3-1-2. Class Diagram -- 40

3-1-3. Sequence Diagram --------------------------------------- 41

3-1-4. Collaboration Diagram -----------------------------------42

3-1-5. Statechart Diagram ---------------------------------------43

3-1-6. Activity Diagram ---44

3-1-7. Component Diagram -------------------------------------- 45

3-1-8. Deployment Diagram --------------------------------------46

3-1-9. Object Diagram --- 47

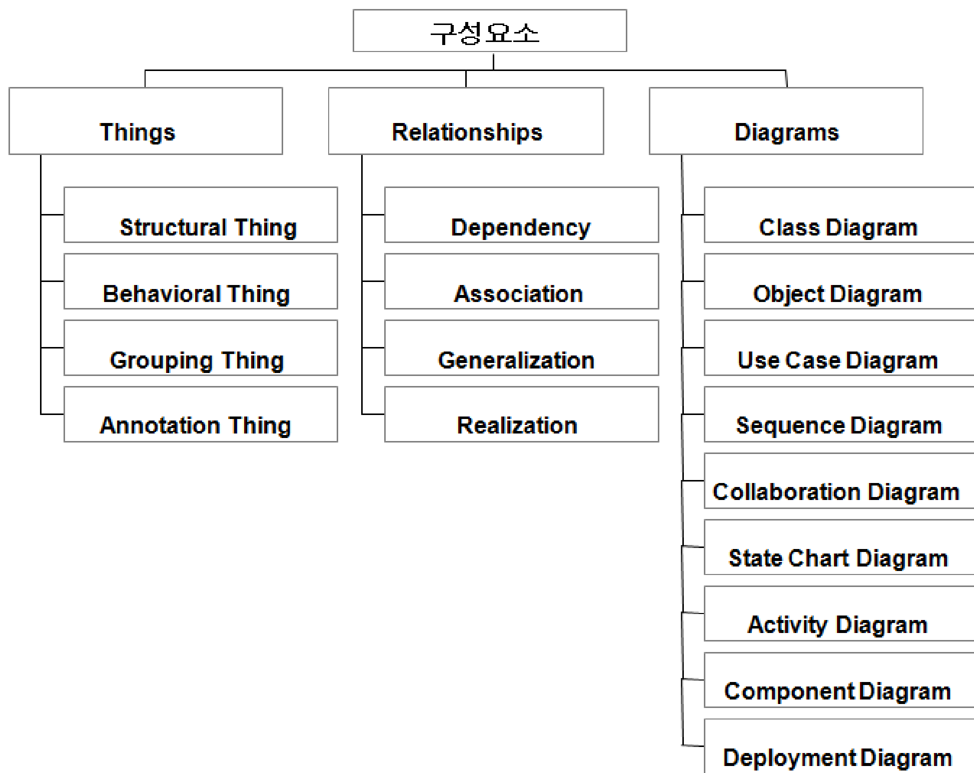
1. UML

1-1. UML 개요

UML 이란, Unified Modeling Language 의 약자로서, 요구분석, 시스템설계, 시스템 구현 등의 시스템 개발 과정에서, 개발자간의 의사소통을 원활하게 이루어지게 하기 위하여 표준화한 모델링 언어이다. 그리고 OMG 표준기구로부터 인정받은 표준화된 그래픽언어이다. 또한, 개발 시스템과 관련된 사람에게 비전을 공유하고, 의견을 얻을 수 있도록 한다.
UML 은 기호와 diagram 을 이용하고, UML 의 작성 목적은 객체 지향 시스템을 가시화, 명세화, 문서화 하는 것이다.

UML 은 구체적으로는 객체지향 소프트웨어의 구조를 원시코드보다 더 추상적인 형태로 구조적이며, 형식적으로 표기하는 언어를 제공하고, UML 에 의한 모델링을 배움으로써, 소프트웨어의 구조에 대하여 적은 정보량으로 많은 지식을 다른 사람에게 전달할 수 있게 되는 효과를 가지고 있다.

UML 의 구성요소는 다음과 같다.



1-2. 모델링과 모델이란?

모델링:

모델을 만드는 일 (추상화)로써, 품질이 좋은 소프트웨어를 개발 및 배치할 수 있게 하는 모든 활동의 중심이며, 모델 구축을 통해 개발 대상 시스템에 대한 이해의 증진을 말한다.

모델:

현실을 단순화/가시화 시키는 것이며, 시스템의 청사진을 제공한다. 그리고 개발 고려 시스템의 총체적인 계획 및 상세 계획을 표현하며, 중요 영향 요소의 파악, 불필요 요소의 생략 및 시스템 구축 제약 조건을 표현하는 것이다.

모델링:

생성할 모델의 신중한 선택을 해야 하며(선택 모델에 따라 문제를 공략하는 방법과 해결책을 실현하는 방법에 많은 영향이 있음), 모든 모델을 다양한 수준의 정밀도로 표현, 그리고 현실을 반영한 모델 작성을 해야 하며 마지막으로 상호 독립적인 모델들 몇 가지를 선택하여 모델링에 착수해야 한다.

객체지향 모델링:

소프트웨어의 모델링 관점으로는 알고리즘 관점이 있다. 이 관점은 소프트웨어의 주요 구성 요소인 Procedure 와 Function 을 제어 관점에서 분할하여 시스템을 모형화 한다. 하지만 요구사항 변화에 적응력이 없고, 대규모 시스템에서는 유지보수를 포함한 관리의 어려움이 있다.
객체 지향 모델링 관점으로는 소프트웨어 시스템의 기본 요소를 객체 또는 클래스로 파악하여 문제 영역과 해결 영역을 모형화 한다. 여기서 객체는 사물(Thing)을 말하며 고유성과 상태, 행동을 갖는다. 그리고 클래스는 공통적인 객체들의 집합을 뜻한다.

결국 UML(Unified Modeling Language)의 목적은 객체 지향 시스템을 가시화, 명세화, 문서화 하는 것이다.

1-3. UML 의 구성요소

1-3-1. Things

Things 는 추상적 개념으로 UML 을 이용한 모델링의 기본요소이다.
이 Things 의 종류는 위에서 보았듯이 4 가지로 구성되어 있다.

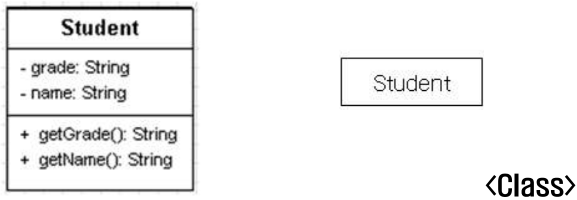
Structural Thing :
UML 모델의 명사형으로서, 모델의 정적인 부분이며 개념적이거나 물리적인 요소를 표현한다.
Behavioral Thing :
UML 모델의 동사형으로서, 모델의 동적인 부분이며 시간과 공간에 따른 행위 요소를 표현한다.
Grouping Thing :
모델을 그룹화 하여 요소를 표현한다.
Annotation Thing :
UML 모델요소를 설명하고, 명확히 하는 표현방법이다.

1-3-1-1. Structural Thing

Structural Thing 은 Class 와 Interface, Use case, Component 그리고 Node 가 있다.

Class:

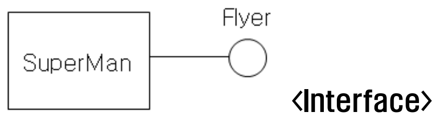
속성과 동작으로 구성된 객체를 표현한다. (아래 그림은 표현방법의 예시이다)



<Class>

Interface:

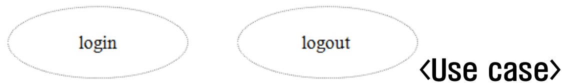
Class 또는 Component 의 동작을 명세화한 operation 의 집합이다.



<Interface>

Use case:

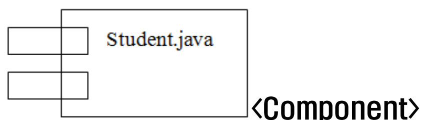
시스템이 수행해야 하는 기능을 기술한다.



<Use case>

Component:

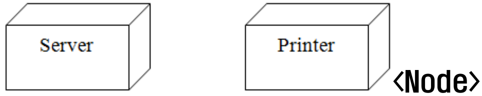
객체지향에서의 모듈 화 된 자원을 말한다. (문서, 소스코드 , 파일, 라이브러리 등)



<Component>

Node:

실행 시에 존재하는 물리적인 요소이며, 주로 컴퓨터 자원을 표현한다.



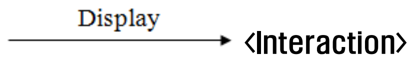
<Node>

1-3-1-2. Behavioral Thing

Behavioral Thing 은 Interaction, State Machine, Package, 그리고 Note 가 있다.

Interaction:

행위를 의미하며 특정문맥에 속한 Object 간의 Messages 들로 구성



<Interaction>

State Machine:

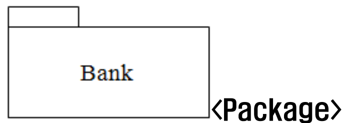
객체의 시간에 따른 상태를 표현한다.

<State Machine>

1-3-1-3. Grouping Thing

Package:

요소들을 그룹으로 묶는 방법, 탭이 달린 폴더로 표현한다. (Framework, Subsystem 표현)



<Package>

1-3-1-4. Annotation Thing

Note:

Comment 로서, 모델 요소를 명확하게 표현 설명하기 위한 방법이다.

주로 제약조건이나 내용을 설명하기 위해 사용한다.

<Note>

1-3-2. Relationships

Relationships 는 구성요소간의 의미 있는 연관성을 표현한다.

(일반적으로 Class 간의 관계 표현 시 사용한다.)

Relationships 의 종류는 크게 4 가지이지만 Association Relationship 의 경우 Aggregation 과 Composition Relationship 으로 다시 분류할 수가 있다.

Dependency Relationship, Generalization Relationship,

Association Relationship (Aggregation / Composition Relationship),

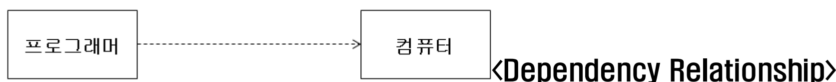
Realization Relationship 이렇게 크게 4 가지 (많게는 5 가지)가 Relationships 의 종류이다.

1-3-2-1. Dependency Relationship

Dependency Relationship 은 한 클래스가 다른 클래스를 사용하는 사용 관계를 의미한다.

즉, 하나의 클래스의 변화가 다른 클래스에 영향을 주는 관계이다.

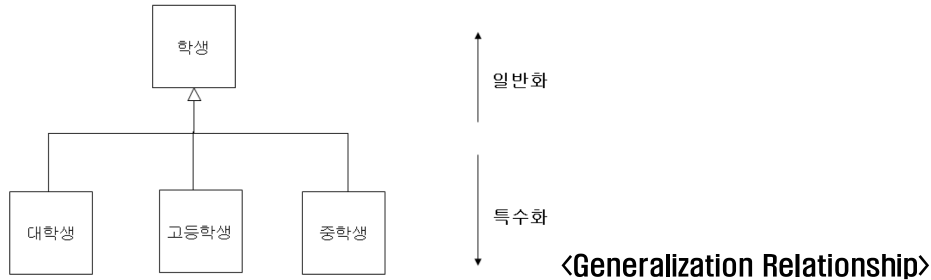
UML 표기법은 점선으로 된 화살표로 표기한다.



<Dependency Relationship>

1-3-2-2. Generalization Relationship

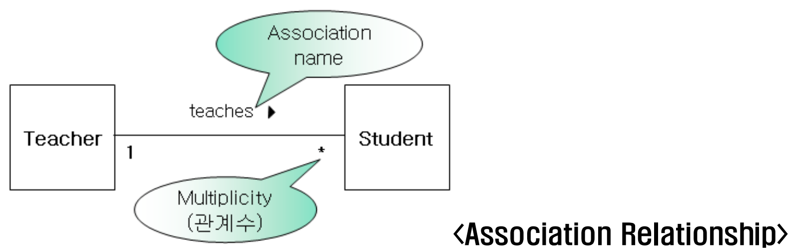
Generalization Relationship 은 일반화와 특수화의 관계이며, 객체지향의 상속관계를 의미한다.



<Generalization Relationship>

1-3-2-3. Association Relationship

Association Relationship 은 클래스로부터 생성된 객체간의 일반적 협력관계를 의미한다.

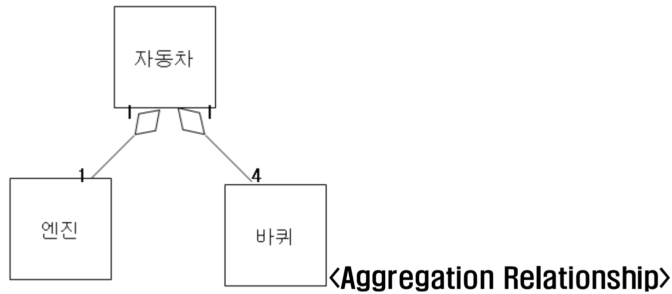


<Association Relationship>

하지만 위에서 설명했듯이 Association Relationship 은 자세히 분류할 수 있는데 그 종류에는 Aggregation Relationship 과 Composition Relationship 이 있다.

Aggregation Relationship:

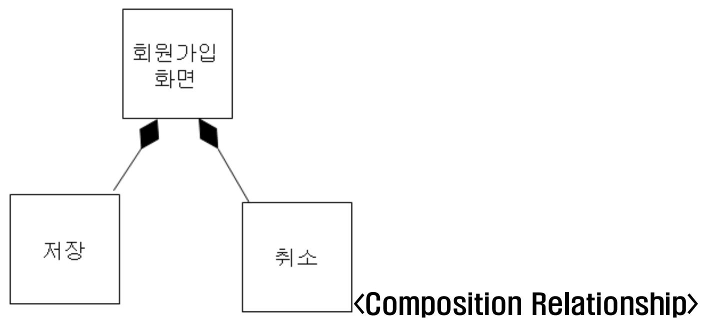
두 클래스간의 전체-부분 관계 (Whole-part)를 나타내며, 각 클래스가 독립적인 생명주기를 갖는다. 하나의 클래스가 여러 개의 컴포넌트 클래스로 구성되어 있는 형태이다.



<Aggregation Relationship>

Composition Relationship:

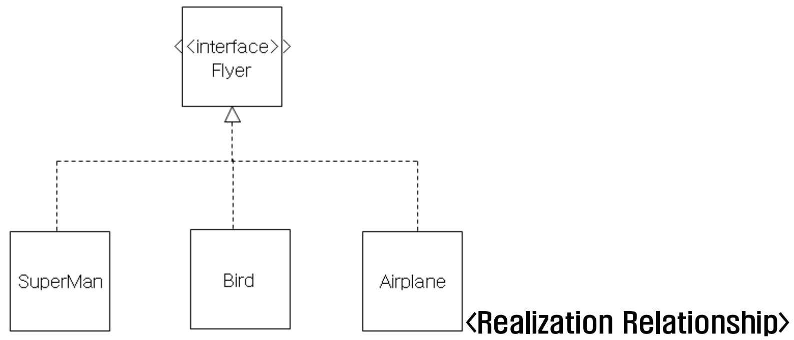
두 클래스간의 부분-전체 관계(Part-Whole)를 나타내며, 부분의 생명주기가 전체의 영향을 받는다. 하나의 클래스가 여러 개의 컴포넌트 클래스로 구성된다.



<Composition Relationship>

1-3-2-4. Realization Relationship

Realization Relationship 은 Interface 와 실제 구현된 Class 간의 관계를 나타낸다.



<Realization Relationship>

1-3-3. Diagram

UML 에서 사용하는 Diagram 은 종류가 무려 9 가지나 된다.

일단, 그 종류와 특징을 표로 만들어 보았다.

구조성	클래스 다이어그램 (Class diagram)	클래스와 클래스 간의 관계를 나타냄
	객체 다이어그램 (Object diagram)	객체와 객체 간의 관계를 나타냄
행동성	유스 케이스 다이어그램 (Use case diagram)	시스템의 쓰임새를 나타냄
	순서 다이어그램 (Sequence diagram)	객체의 상호관계를 시간 축에 따라 나타냄
	콜라보레이션 다이어그램 (Collaboration diagram)	객체간의 상호작용을 나타냄
	스테이트차트 다이어그램 (State chart diagram)	객체의 상태를 나타냄
	액티비티 다이어그램 (Activity diagram)	객체의 행동 변화를 나타냄
구현성	컴포넌트 다이어그램 (Component diagram)	컴포넌트와 컴포넌트 간의 관계를 나타냄
	배치 다이어그램 (Deployment diagram)	객체와 물리적인 배치를 나타냄

* 여기서 Sequence, Collaboration Diagram 은 묶어서 Interaction Diagram 에 속한다.

1-3-3-1. 누가 어떠한 Diagram 을 주로 사용하는가?

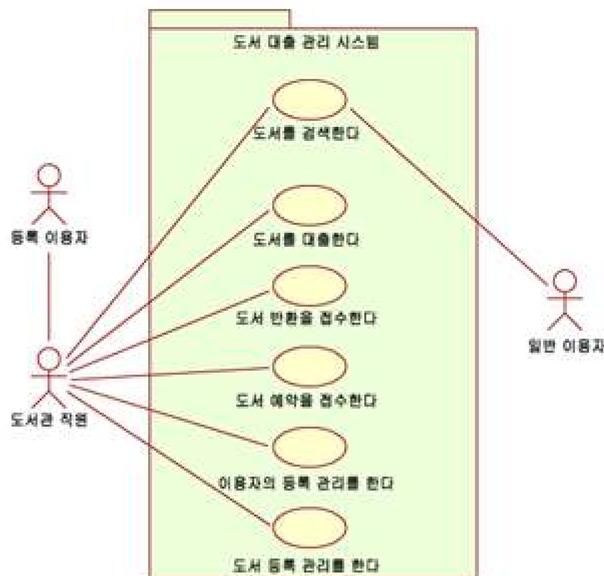
* 소프트웨어를 설계하고 구축하는 모든 사람이 위에서 설명한 9 가지 다이어그램을 모두 이해해야 하는 것은 아니다.
그렇다고 Programmer 가 Use case diagram 을 완전히 몰라도 된다는 것은 아니다. 따라서, 아래의 표의 0 표는 주로 사용하기 때문에 충분한 이해가 필요한 다이어그램에 해당되며, 0 표가 없다고 몰라도 된다는 뜻은 아니다.

Diagram	Architect (설계자)	Programmer (개발자)	Analyst (요구분석을 담당하는 분석가)
Class diagram	0	0	0
Object diagram	0		
Use case diagram	0		0
Sequence diagram	0	0	
Collaboration diagram	0		
State chart diagram	0		
Activity diagram	0		0
Component diagram	0		
Deployment diagram	0		

* UML 에서 특히 객체지향 개발을 하는 경우에 중요한 다이어그램의 종류는 그리 많지 않다. Class Diagram 과 Sequence Diagram 만으로도 해결할 수 있으며, 요구 관리까지 하고 싶다면 Use case diagram 까지 사용하면 매우 충분하다. 이 3 가지의 Diagram 을 제대로 이해 한 후에, 다른 Diagram 을 필요에 따라서 잘 다룰 수 있다면 충분한 것이다.
이 보고서에서는 Use case diagram, Class diagram, Sequence diagram 을 기본으로 사용하여 설명을 할 것이다.
다음에 나오는 내용은 위 3 가지 diagram 을 좀 더 자세히 설명한 것이다.

1-3-3-2. Use case diagram

Use case diagram 을 사용하여 그 시스템이 이용자에게 제공해야 할 기능, 서비스의 범위를 확정한다. 그 때 어떤 입장이나 역할의 이용자(관련 외부 시스템 포함)가 있는지를 *[액터]*로 분류한다. 그 이용자의 형태=액터 에게 시스템이 제공해야 할 서비스 내용을 *[유스 케이스]*로서 식별하고, Use case diagram 에 유스 케이스를 등록한다. 그와 동시에 그 유스 케이스의 내용을 구체적인 이용 시나리오(이벤트 플로우 기술)로서 문서화한다. Use case diagram 은 어디까지나 유스 케이스 기술 문서의 카탈로그에 지나지 않는다.
즉, Use case diagram 자체가 시스템 요구에 대해서 그만큼 많은 정보를 나타내는 것은 아니다. 본래의 시스템 요구 내용은 유스 케이스 기술(문서) 쪽에 있기 때문이다. 따라서, 프로젝트에 의해 관리하기 쉬운 것부터 굳이 Use case diagram 을 사용하지 않고 Excel 의 표로 유스 케이스 리스트로서 관리하는 경우도 자주 볼 수 있다.
보통 *[액터]*는 사람 모양 으로 표기를 하고, *[유스 케이스]*는 타원형으로 표기한다.
아래의 그림은 Use case diagram 의 예를 보여 준다.



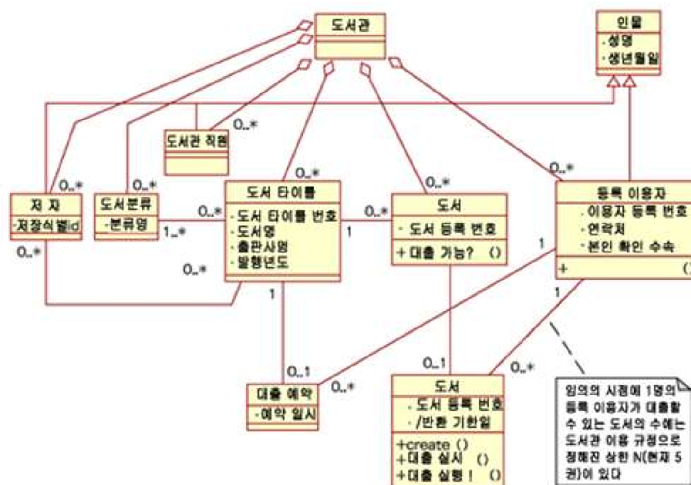
1-3-3-3. Class diagram

Class diagram 은 그 시스템이 대응하려고 하는 업무 영역의 겨냥도를 나타내는데 사용하거나, 그 업무를 지원하기 위한 시스템의 구조를 표현하는데 사용한다. 이것은 대상이 되는 영역이나 시스템을 그 구성 요소로서의 클래스군과 클래스 간의 의미적, 논리적, 물리적인 관계로서 모델화하는데 사용한다. 그리고, 구성요소를 나타내는 각 클래스는 속성(정적 지식=데이터)과 조작(동적 지식=수속)을 자신의 자원으로서 관리한다.

속성과 조작은 (클래스의) 박스의 제 2 란과 제 3 란에 표시된다. Class diagram 안의 각 박스가 클래스를 표현하고 클래스 간을 잇는 선이 관련을 나타낸다. 만약, 클래스 간에 유사성이 있다면 범화관계(색이 없는 삼각형 화살표선으로 표시)를 이용하여 클래스의 분류 구조를 포현할 수도 있다.

따라서, Class diagram 이라고 해도 3 레벨이 있다. 그 첫 번째는 *[개념 클래스 다이어그램]*, 두 번째는 *[사양 클래스 다이어그램]*, 그리고 마지막 세 번째는 *[실현 클래스 다이어그램]*이 있는데, 이 다이어그램들은 디자인이 진행됨에 따라 추상적인 표현이 줄어들고 모델이 구체화, 현실화되어 간다. 시스템의 설계라는 관점에서 보면 Class diagram 은 그 시스템이 기능할 수 있도록 하는데 필요한 구조를, 각각 전문 책임 범위를 가진 클래스의 제휴로서 실현된다. 그것을 위한 겨냥도가 Class diagram 이다. 각 클래스는 업무를 나타내는 중요한 개념과 Application 으로서 필요한 서비스, 정보를 나타내는 객체에 대한 사양을 나타내게 된다.

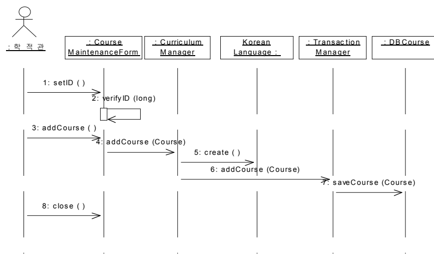
아래 그림은 Class diagram 의 예를 보여 준다.



1-3-3-4. Sequence diagram

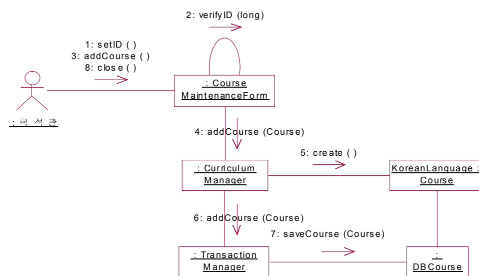
Sequence diagram 은 상호작용 다이어그램의 일종이다. 일반적으로 실현하고 싶은 유스케이스의 구체적인 시나리오는 서비스에 따라서 그리게 되는데, 그 서비스를 실현함에 있어서 사용하지 않으면 안되는 객체를 모두 등록시키고 객체 사이에서 어떤 메시지의 교환이 순서대로 일어나면 적절하게 *[협조 작업]*이 진행되며 결과적으로 기대했던 서비스가 실시되어 필요한 시스템 상태에 도달할 것인지를 시간에 따라서 메시지 송신열로써 표현한 것이 Sequence diagram 이다. 이 Sequence diagram 의 표현방법은 세로에 등장 객체의 존재를 배치하고 위에서 순서대로 필요한 메시지 송신열을 배치해나간다. 가장 위가 그 서비스의 실행 전 시스템 상태를 표현하며 맨 밑이 그 서비스 실시 후의 시스템 상태를 나타낸다고 생각할 수 있다. 굵은 기둥으로 표현된 부분이 객체가 메시지를 받아서 활성화되어 있는 상태를 나타낸다. 또 점선의 화살표는 하고 있는 상태를 나타내고 있다. 그리고 점선의 화살표는 *리턴(메시지의 반환)*을 나타낸다. 이 Sequence diagram 의 주의점은 Class diagram 중의 '속성이나 조작의 식별은 각 클래스에 어떤 역할로 동작을 책임지도록 할 것이다.' 라는 설계 판단과 연동된다. 즉, Sequence diagram 에서의 메시지 송수신 정의와 Class diagram 상세 정의란 왔다 갔다 하는 병행작업이 될 것이다. Try & Error 를 거듭하면서 적절한 객체에 적절한 동작 책임(그 동작에 대응하는 조작 정의 = 메소드와 그 실행에 필요한 자원으로서의 속성 설정)을 각 클래스의 역할 분담으로서 균형 있게 분산 배치해 나간다.

아래 그림은 Sequence diagram 의 예를 보여준다.



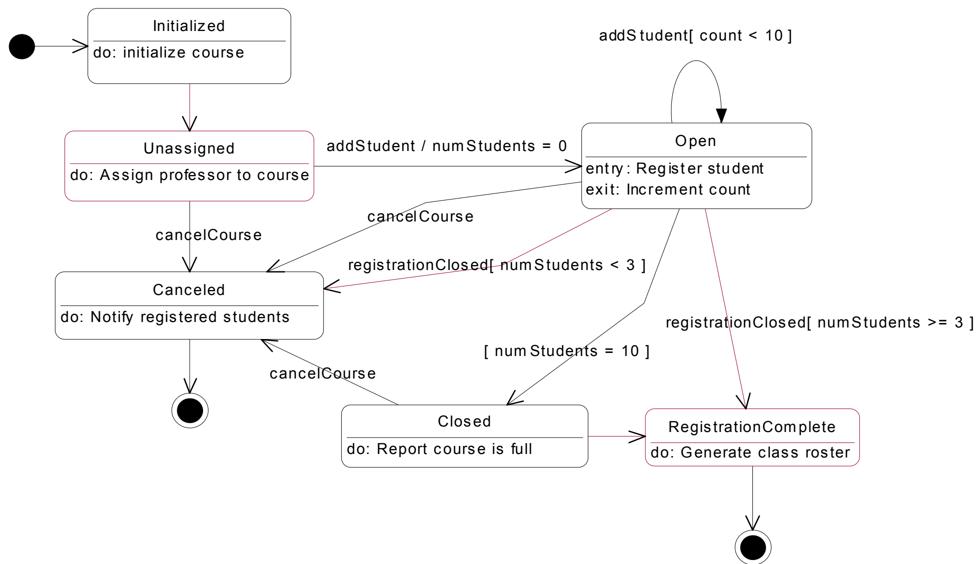
1-3-3-5. Collaboration Diagram

Collaboration Diagram 은 Sequence Diagram 과 같이 상호작용을 나타내는 또 다른 표현기법이다. Sequence Diagram 에서와 같이 객체들은 상자모양 아이콘으로 표시하고, 객체들이 주고받는 메시지 역시 객체간의 화살표로 표시한다. 사용사례를 구현하기 위한 메시지의 순서는 메시지에 번호를 매겨 표시한다. 메시지에 번호를 매기는 것은 Sequence Diagram 보다 순서를 보기에 불편한 점이 있다. 반면에 객체들을 공간적으로 배치시킴으로써 아키텍처 등 다른 중요한 정보를 강조할 수 있다. 객체에 이름을 주는 규칙은 objectName : ClassName 으로 한다.



1-3-3-6. State Diagram

사용사례와 시나리오는 시스템의 행동양식 즉 객체들의 상호작용을 기술하는 기법이다. 때로는 한 객체의 행동양식을 기술할 필요가 있다. State Diagram 은 한 객체가 자신의 생명주기 안에서 취할 수 있는 상태들과 그 상태 간 전이를 일으키는 이벤트들, 그리고 상태 간 변화에서 발생하는 작용들을 표현한다.

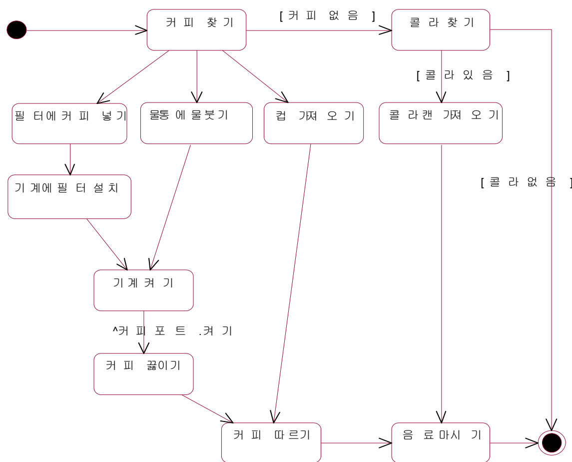


위 그림은 수강관리시스템에서 Course 라는 클래스가 취할 수 있는 여러 상태와 그들 간의 전이를 표현한 것이다.

State Diagram 은 시스템의 모든 클래스에 대해 그릴 필요는 없으며 의미 있는 행동양식을 보여주는 주요 클래스들에 대해서 그린다. 가능한 상태나 이벤트 역시 필요에 따라 간단하거나 복잡한 수준으로 표현한다.

1-3-3-7. Activity Diagram

Activity Diagram 은 작업흐름과 연계되어 병행 처리가 많은 행동양식을 기술하기에 특히 유용한 여러 기법들을 조합한 것이다.

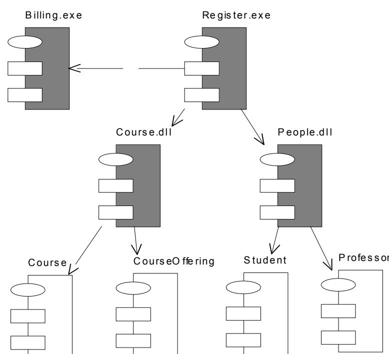


위 그림은 음료를 마시는 활동에 대한 작업흐름을 표현한 Activity Diagram 이다. 그림에서 핵심 요소는 활동(activity)이다. 활동이란 개념적 관점에서 보면 사람이나 컴퓨터가 행하는 어떤 작업을 의미할 수도 있으며, 구현 관점에서 보면 클래스의 메소드를 의미할 수도 있다.

그림에서 [커피 찾기] 다음에 이어지는 [필터에 커피 넣기], [물통에 물 붓기], [컵 가져오기]의 세 활동은 병행 처리를 의미한다. 순서도(flowchart)가 보통 순차적인 프로세스에 제한되어 있는 반면에 Activity Diagram 은 병행 프로세스를 기술할 수 있다.

1-3-3-8. Component Diagram

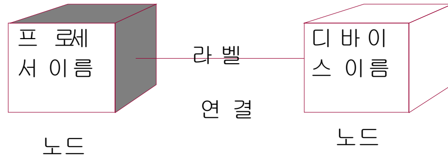
Component Diagram 은 시스템을 구성하는 실제 Software component 간의 구성체계를 기술하므로 아키텍처를 표현하기에 좋다. component 란 용어는 시스템의 물리적 구조를 구성하는 소스 코드 단위로부터 Subsystem 같은 실행 프로그램에 이르기까지 다양하게 지칭한다. 예를 들어 개별 클래스의 헤더와 구현파일로부터 그들을 조합하여 만들어진 EXE, DLL 등까지 component 로 표시할 수 있다. 이러한 다양한 수준의 component 를 사용하여 시스템의 아키텍처를 표현한다.
Component Diagram 에는 각 component 를 그리고 component 의 dependency 관계를 화살표로 나타낸다.



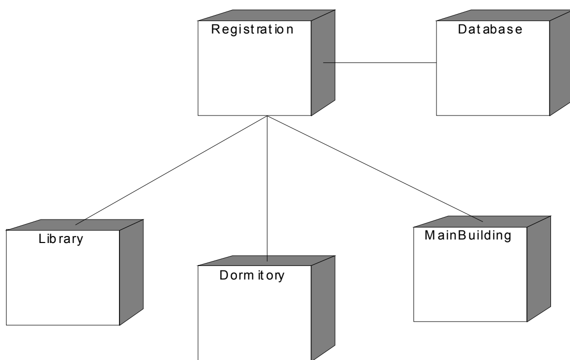
위 그림은 수강관리시스템의 Component 들을 dependency 를 중심으로 간단하게 표현한 것이다. 시스템은 두 개의 EXE 과 두 개의 DLL 로 이루어져 있으며 각 DLL 이 사용하는 클래스들을 표현하고 있다.

1-3-3-9. Deployment Diagram

Deployment Diagram 은 인도될 시스템의 Software 와 Hardware component 간의 물리적 관계를 표현한다.



위 그림은 Deployment Diagram 의 모델링 요소를 표현한 것이다. Deployment Diagram 은 Node 와 Node 간 연결로 구성된다. Node 는 프로세서나 디바이스처럼 독립된 하드웨어 요소를 의미하며 보통 클라이언트 PC 나 서버 워크스테이션, 때로는 간단한 주변장치나 Mainframe 을 표시하기도 한다.



위 그림은 수강관리시스템의 Deployment Diagram 으로서 서버와 단말기 요소들을 표현하고 있다.

1-3-3-10. Object Diagram

Object Diagram 은 Class Diagram 의 변형으로서, Class Diagram 과 비슷하다. 그렇기 때문에 따로 그림 설명은 하지 않는다.

Object Diagram 은 객체와 그들 간의 관계를 표현하며, Class Diagram 에 있는 요소의 Instance 에 대한 정적 Snap Shot 을 나타낸다.

2. OOAD

OOAD 는 Object-Oriented Analysis and Design 의 약자로 객체지향분석 및 설계를 말하는 것이다. 이 OOAD 는 최근에 발표된 개념이고, 개발의 문제점을 해결해 줄 많은 장점을 보유하고 있다. 그리고 요구사항 변경을 수용하며 (유연성과 적응력 필요) 기존의 데이터와 행위가 분리되었던 개발 방법의 복잡성과 통합의 어려움을 극복하려는데 있다. (데이터와 행위는 객체를 정의내리고 객체를 추상화시키는 작업을 말한다.)

이 Chapter 를 들어가기 전에 간단한 용어 정의를 하자면,

Object = Data 변수 + Procedure
Attribute = 객체내의 Data 변수
Method = Procedure
Message = Method 를 실행시키라는 신호
Class = 객체들을 하나의 집합으로 묶는 추상화의 명칭
Instance = Class 에 속한 객체
Inheritance = 기존의 Class 를 사용하여 하위 Class 를 정의하는 것
계층적 연관관계에 따라 : 일반화/집단화/결합화
상속의 형태에 따라 : 단일 상속/다중 상속

이 될 것이다.

2-1. 객체지향의 기본 개념

객체지향의 기본 사상은 복잡한 메커니즘의 현실 세계를 인간이 이해하는 방식으로 시스템에 적용시켜 보자는 것이다. 객체지향 기술에는 객체(object)와 객체들의 범주를 나타내는 클래스(class), 그리고 객체간의 상호작용을 위한 메시지(message)가 있다.
참고로 복합 객체란 여러 개의 객체를 포함한 객체를 뜻한다.

<table>
<tr><td colspan="5">객체</td><td>객체의 특징</td></tr>
<tr><td>유형객체</td><td colspan="4">무형객체</td><td rowspan="3">* 상태를 갖음
* 한 클래스의 Instance
* 행동에 의해 특성 표현
* 이름을 갖음
* 다른 객체와의 연관성</td></tr>
<tr><td rowspan="2">실제행위를 능동적으로 수행하는 물리적인 속성을 가진 것</td><td>임무객체</td><td>사건객체</td><td>상호작용객체</td><td>명세서객체</td></tr>
<tr><td>역할/행위 수반</td><td>행위호출</td><td>객체간 통합/분기 결과 생성</td><td>행위의 참고 및 표준</td></tr>
</table>

객체지향 개발 방법은 객체와 그의 속성과 동작, 유사한 객체의 집합으로 나누어진 클래스, 객체 사이의 관계 등을 기본 개념으로 한다. 그리고 이 방법은 소프트웨어 개발 전 과정에 걸쳐 동일한 방법론과 표현 기법이 적용 될 수 있는 것이 장점이며, 특히 객체지향 분석기법은 기존의 분석기법에 비해 현실세계의 현상을 보다 정확히 모델링 할 수 있어 어려운 응용 분야들에 적용이 가능하다. 거기에 분석과 설계의 표현에 큰 차이점이 없어 시스템의 개발을 용이하게 해준다. 그리고 분석, 설계, 프로그래밍의 결과가 큰 변화 없이 재사용될 수 있어 확장성이 좋고, 시스템 개발 시 시제품이나 나선형 패러다임의 적용이 가능하다. 또한, 객체 중심의 상향식 접근 방법이 도입되고, 기능 중심이 아닌 정보 중심, 데이터 중심으로 시스템의 개발이 이루어진다.
객체지향 개발 방법은 기존의 정보 모델링에 기초하고 있으며 여기서 나타난 객체의 정적인 정보에 객체의 동작을 추가 시켜 객체를 완벽하게 기술하고 구현하는 방법론이다.

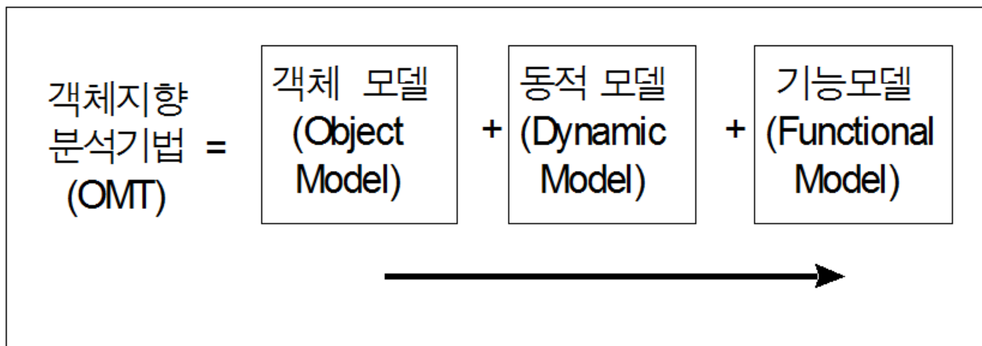
2-2. 객체지향 분석의 관점

객체지향 분석은 3 개의 관점이 있는데, 객체, 동적, 기능적 관점을 말한다.

객체 모델링 : 정보 모델링이라고도 부르며 시스템에서 요구되는 객체를 찾아내어
객체들의 특성과 객체들 사이의 관계를 규명한다.
동적 모델링 : 객체 모델링에서 규명된 객체들의 행위와 객체들의 상태를 포함하는
Life cycle 을 보여준다.
기능 모델링 : 각 객체의 형태 변화에서 새로운 상태로 들어갔을 때
수행되는 동작들을 기술하는데 사용한다.

객체지향 개발 방법은 소프트웨어 공학에서 추구하는 많은 장점들을 제공한다. 이 방법을 제대로 활용하기 위해서는 기존의 기술과는 다른 높은 차원의 기술력을 분석가나 디자이너에게 요구해야 하는데, 소프트웨어 개발 과정에 대한 이해와 정보 모델, 동적 모델, 기능 모델에 대한 지식이 있어야 하며, 모델들 사이의 연관성을 바탕으로 모델링의 결과를 통합할 수 있어야 한다.

아래의 그림은 기본적인 개념을 나타낸 것이다.

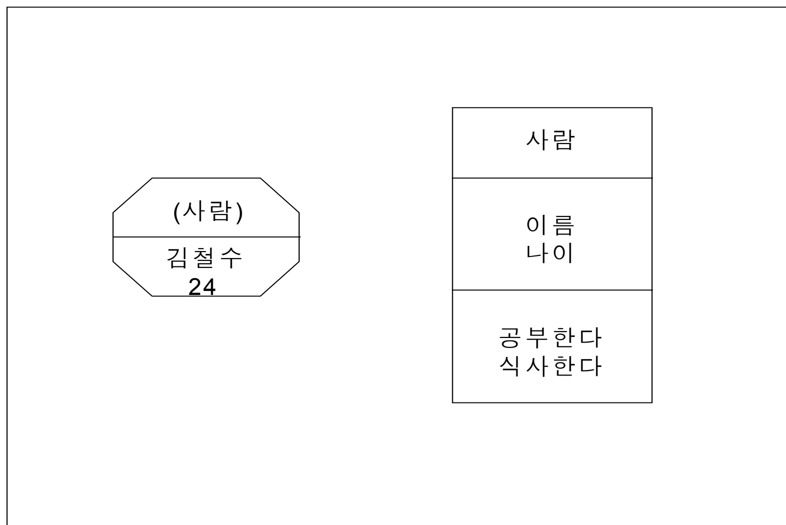


2-3. 객체 모델링

2-3-1.객체와 클래스

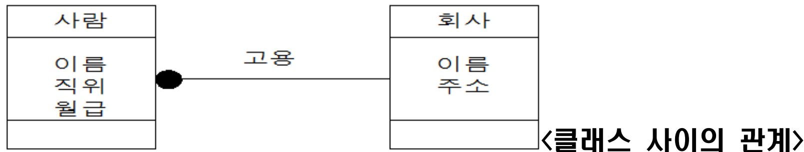
객체는 객체 모델링의 기본 단위이며, 클래스는 유사한 객체들의 모임을 말한다.
이것은 정보 모델링의 엔티티와 엔티티 타입에 해당하며, 객체는 구석이 다듬어진 사각형으로 표시하고 클래스는 사각형으로 표시한다.

아래 그림은 그 예를 보여 준다.

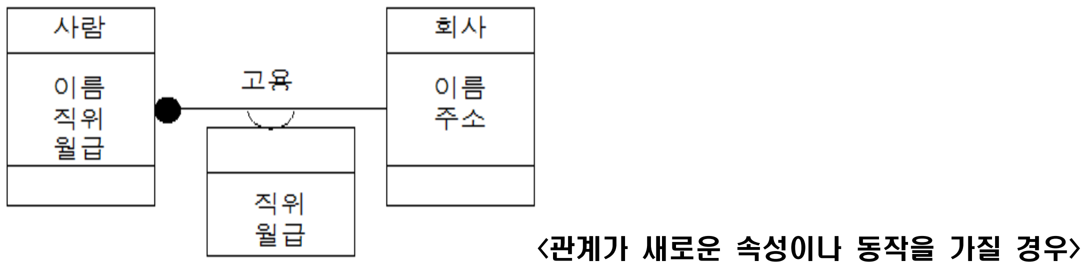


2-3-2. 클래스의 관계

클래스들 사이의 연관성은 관에 의해 표시되며 정보 모델링에서는 마름모로 표시한다. 그리고 OMT 에서는 그 관계를 단순히 관계 이름만으로 표시할 수 있다. 만약 관계가 새로운 속성이나 동작을 가질 경우 아래 2 번째 그림과 같이 나타낼 수 있다.

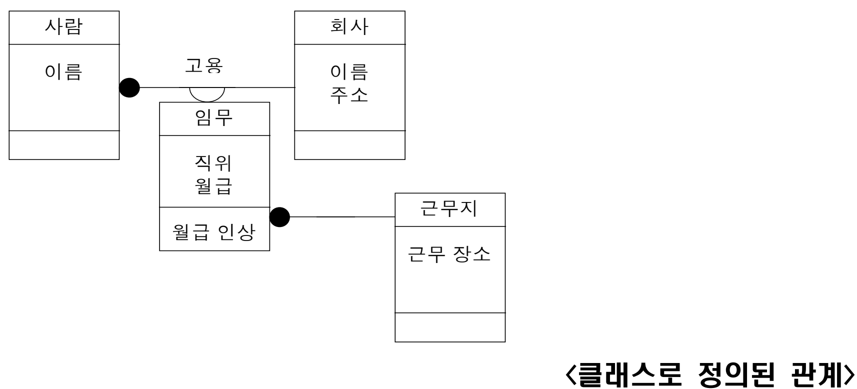


<클래스 사이의 관계>



<관계가 새로운 속성이나 동작을 가질 경우>

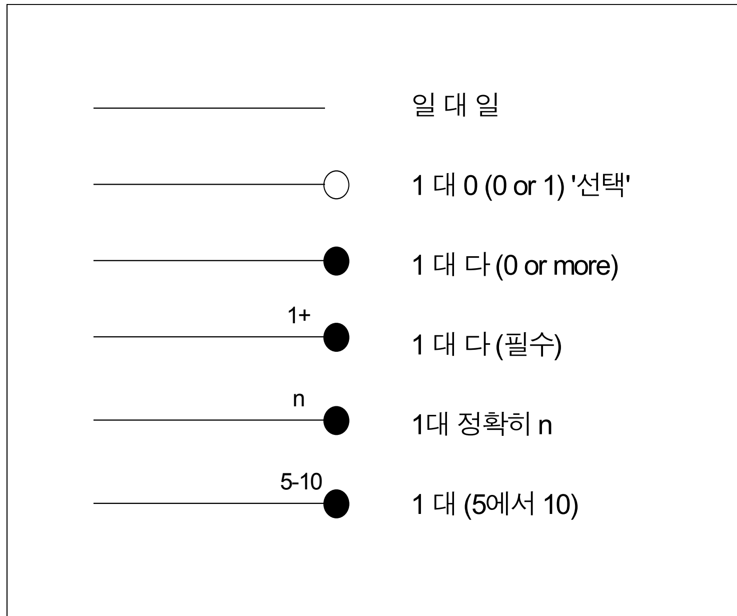
그런데 경우에 따라서는 관계를 한 클래스로 모델링하는 것이 효과적인 경우가 있다. 이 경우 관계는 하나의 클래스로 정의되어 속성과 동작을 가지게 되며 다른 새로운 클래스와 연관을 맺을수 있게 된다.



<클래스로 정의된 관계>

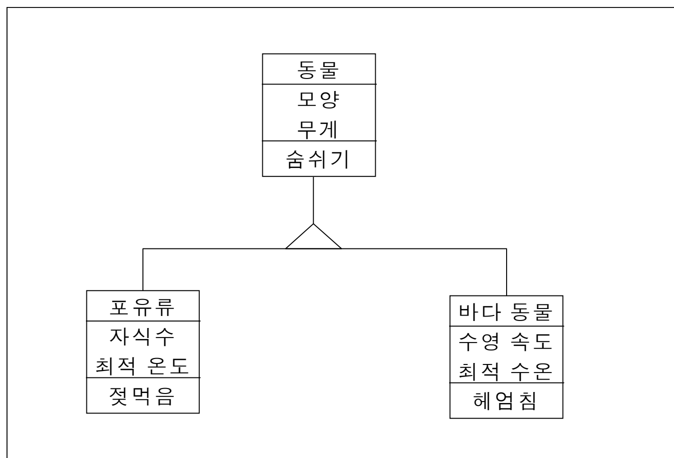
2-3-3. 매핑 제약조건

다음 기호들은 클래스들 사이에 맺어질 수 있는 관계를 통해 객체들이 지켜야 하는 제약조건을 지정하는 것이다. 각 클래스의 객체들 사이에 맺어질 수 있는 매핑 제약조건과 참여 제약조건은 정보 모델링에서 다루어진 개념과 동일하며 OMT 에서는 다음과 같은 기호로 표시한다.



2-3-4. 클래스의 일반화

일반화는 유사한 클래스들 사이의 공유되는 속성과 동작을 묶어주며, 그들 사이의 다른 점을 보존할 수 있게 하여주는 효과적인 추상화 기법이다. 일반화는 클래스들 사이의 특수한 관계로 볼 수 있다. 이러한 일반화를 통해 클래스들 사이의 계층 구조가 만들어지며 각 하위 클래스는 상위 클래스의 속성, 동작, 그리고 다른 클래스와의 연관성을 상속받게 된다.
일반화를 통해 공통의 정보는 오직 한 번만 정의될 수 있어 분석의 결과를 재사용할 수 있게 하여 주며 데이터의 무결성을 향상시켜 준다.
일반화와 반대되는 개념으로 특수화라는 것이 있는데, 이것은 말 그대로 일반화의 반대이다.
특수화를 통해 하위 클래스로 정의되는 경우는 하위 클래스가 고유의 속성이나 동작을 가지고 있거나 하위 클래스가 다른 클래스들과 고유의 관계를 가지고 있을 때이다.
일반화는 is a 관계 또는 kind of 관계이며 각 하위 클래스의 인스턴스는 상위 클래스의 인스턴스가 된다. 일반화를 통해 다계층 구조를 만들어 나갈 수 있으며, 이 경우 한 하위 클래스는 이 구조상에 있는 모든 상위 클래스의 속성과 동작을 상속받는다.

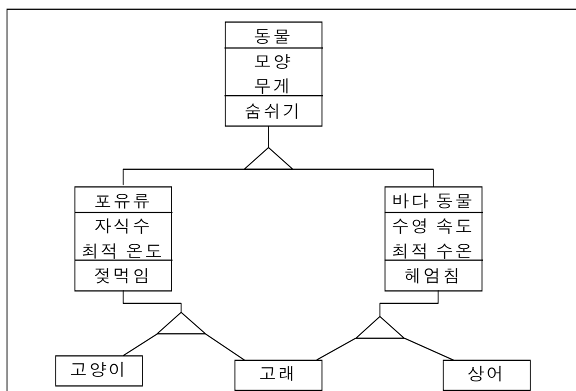


<클래스의 일반화>

2-3-5. 다중 상속

2-3-4 에서 보았던 클래스의 일반화에서 만약, 어떤 클래스의 상위 클래스가 2 개가 있는 경우 문제가 발생할 수 있다. 이 때 하위 클래스는 2 개 이상의 상위 클래스에서 동시에 속성과 동작을 상속받게 되는데, 이를 다중 상속이라고 한다. 이렇게 다중 상속이 되면, 분석 과정에서 모순이 발견될 수가 있는데, 이러한 모순이 속성이나 동작에서 감지되면 반드시 요구사항 명세서나 자료 사전에 이를 기록해야 한다.

아래는 그 다중 상속이 일어난 예이다.

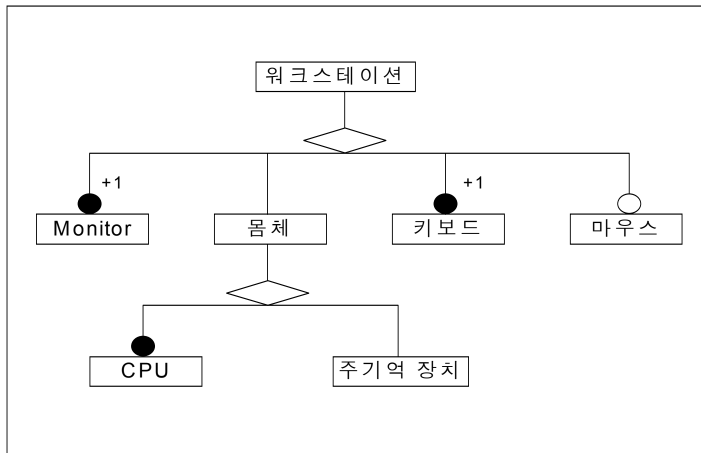


<어떤 클래스의 상위 클래스가 2 개 있는 경우 (다중상속)>

2-3-6. 집단화(Aggregation)

집단화는 클래스들 사이에 part-whole 관계 또는 part 관계로 설명되는 연관성을 나타낸다. (1-3-3-1 에서 설명했던 Aggregation relationship 과 같은 방식으로 이해하면 된다.)

집단화는 여러 부속 객체들이 조립되어 하나의 객체가 구성되는 것을 의미하며, 객체지향 기법에서는 활발히 사용되고 있다. 이 집단화 기호는 마름모 기호로 표시되며, 매핑 제약조건과 참여 제약조건에 대한 기호는 일반화의 경우와 동일하다.



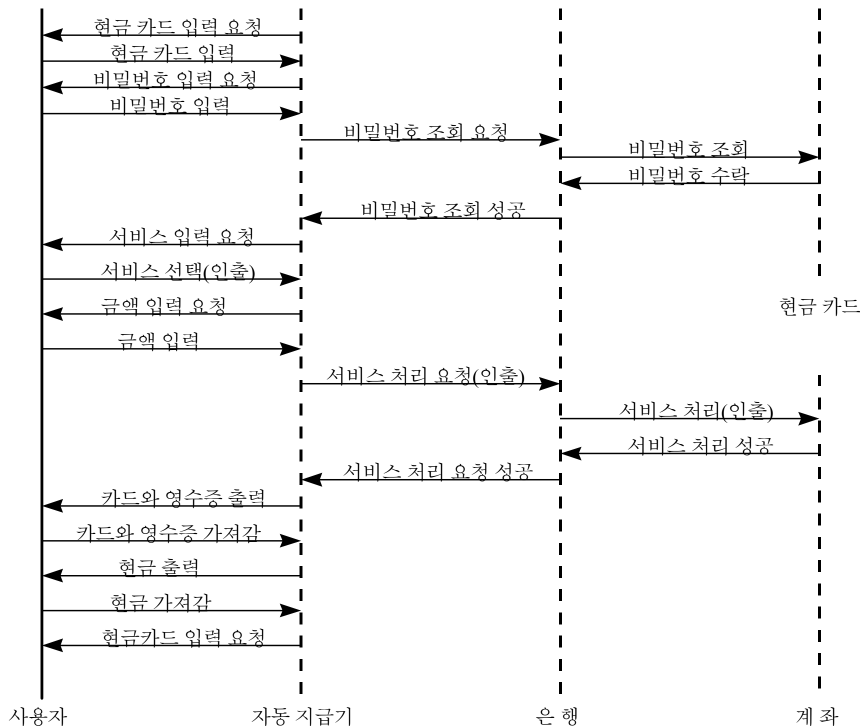
2-4. 동적 모델링

2-3 에서 설명한 객체 모델링을 통해 시스템에 요구되는 객체들의 구조와 객체들 사이의 관계를 정립하고, 시간의 흐름에 따른 객체들과 객체들 사이의 변화를 조사하는 것을 동적 모델링 이라고 한다.
동적 모델링은 객체들 사이의 제어 흐름, 상호 작용, 동작의 순서 등을 다루며, 중요한 개념은 상태, 사건, 동작 등등 이다.
객체지향 분석의 특징은 시스템에 요구되는 객체를 우선 구한 후 객체들의 동적인 면을 찾아내기 위해 동적 모델링을 적용한다는 점이다. 동적 모델은 다음에 나올 기능 모델과 관계가 있으며 시스템의 동작과 기능을 수행하기 위해 필요하다.
OMT 기법은 동적 모델링에 상태변화도(STD) 사용한다.
우선 제안서로부터 시나리오를 만들게 되며 시나리오는 객체 또는 시스템의 한 실행 과정을 사건들의 흐름으로 표시하고 각 사건들은 객체 모델링에서 규명된 객체들 사이의 정보 흐름을 나타내게 되고 각 사건의 정보를 보내는 객체와 정보를 받는 객체가 밝혀진다. 그리고 사건의 순서와 사건을 주고받는 객체들이 사건 추적도에 나타나게 되며 수직선은 객체를 표시하고 수평 화살표는 사건의 흐름을 나타내게 된다.(사건은 위에서 아래로의 순서에 의해 수행된다)
만약, 다음과 같은 시나리오가 있다고 가정하면, 사건추적도와 상태변화도는 다음과 같게 된다.

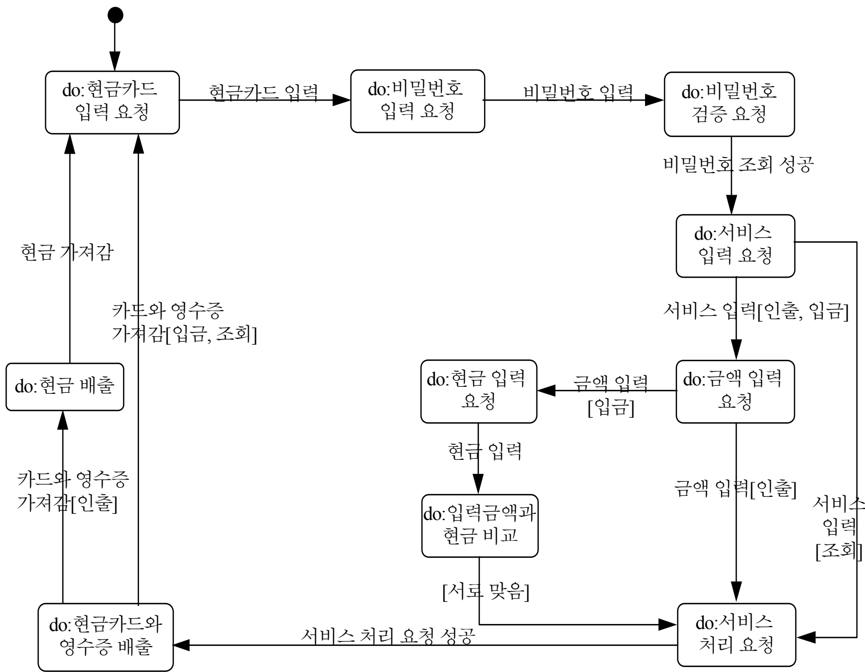
<시나리오>

- 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.

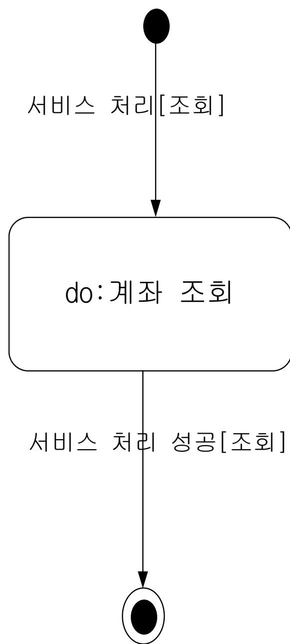
- 사용자가 인출할 금액을 입력한다.
- 자동 지급기는 해당 은행에게 인출할 금액 인출을 요구한다.
- 은행은 해당 계좌에게 인출할 금액 인출을 요구한다.
- 계좌는 잔액에서 인출할 금액을 인출하고 인출이 성공적으로 끝났음 을 은행에 알린다.
- 은행은 자동 지급기에게 현금 인출이 성공적으로 끝났음을 알린다.
- 자동 지급기는 사용자에게 카드와 영수증을 내어준다.
- 사용자가 카드와 영수증을 가져간다.
- 자동 지급기가 인출 금액을 내준다.
- 사용자가 인출 금액을 가져간다.
- 자동 지급기가 현금 카드를 입력할 것을 요구한다.

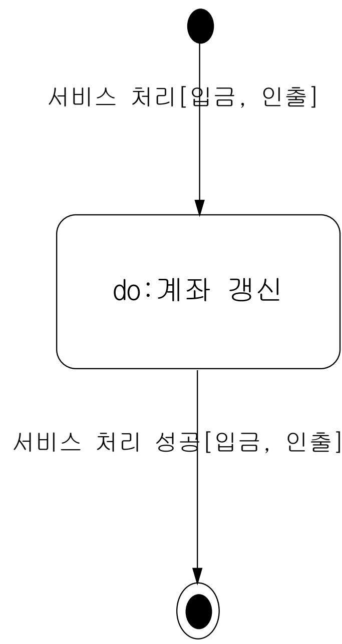


<사건추적도>



<자동지급기의 상태변화도(STD)>

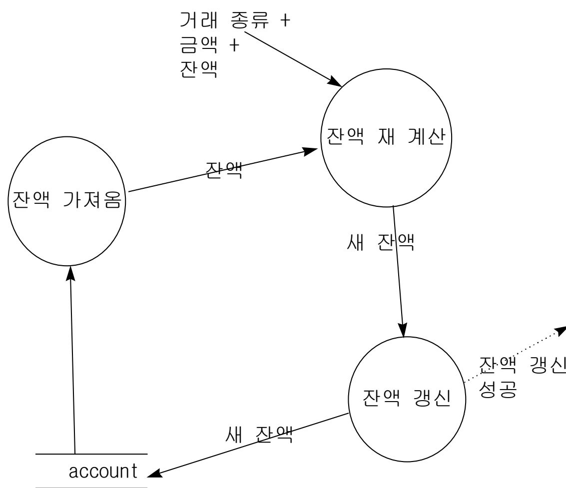




<계좌의 상태변화도(STD)>

2-5. 기능 모델링

기능 모델링은 객체지향 분석기법에서 객체 모델링, 동적 모델링에 이어 시스템을 기술하는 세 번째 단계이다. 기능 모델은 입력값으로부터 계산을 거쳐 어떤 결과가 나타나는지를 보여주며 이것이 어떻게 유도되었는지의 구현 방법은 고려하지 않는다. 기능 모델은 자료흐름도 (DFD)를 사용할 수 있고, 이 DFD 의 경우 입력 흐름은 프로세스를 통해 변환되어 출력 흐름으로 바뀌고, 다른 프로세스의 입력이나 외부로의 출력으로 작용한다. 기능 모델링은 동적 모델링에서 나타난 동작이 어떠한 기능으로 이루어져 있는지 보여주며 자료흐름도의 정보 흐름은 객체에 해당된다.



<계좌 갱신 상태의 자료흐름도(DFD)>

2-6. 객체지향 설계

분석 단계를 통하여 앞에서 소개한 세 가지의 서로 다른 모델이 순서대로 개발된다.
세 모델을 하나로 통합하는 작업이 이루어져야 하며, 이 단계는 일반적으로 객체지향 설계단계에 해당이 된다. 세 모델의 통합은 객체의 정적인 구조와 오퍼레이션을 함께 포함 하여 객체를 정의하는 것을 의미하며, 이는 동적 모델의 사건, 동작 및 활동을 객체의 오퍼레이션에 매핑 하고, 기능 모델의 프로세스를 객체 모델의 오퍼레이션에 통합시키는 것이다.
객체 수준의 상태변화도(STD)는 한 객체가 생명주기 동안 가질 수 있는 상태들을 기술하여 준다. (상태의 변환은 객체의 오퍼레이션으로 매핑한다.)
유사한 방법으로 객체에 주어진 사건은 다른 객체의 동작으로 나타낼 수 있다.
한 사건은 이전 사건에 의하여 초기화된 활동의 결과이며, 동작과 이에 반응하는 동작으로 매핑 될 수 있다.
자료흐름도(DFD)는 다른 두 모델과 연관성을 가지고 있으며 상태 변환에 의하여 이루어진 동작은 기능 모델의 자료흐름도(DFD)와 연관을 가질 수 있다.
서로 연결되어 있는 프로세스의 집합들은 동작의 구체적인 기능 표시하고, DFD 의 프로세스들은 STD 의 동작에 의하여 수행되는 하위 기능으로 나타낸다.
대상객체는 오퍼레이션이 정의되고 속하여 있는 객체를 의미하며, 프로세스가 여러 입력 자료 흐름으로부터 하나의 출력 값을 생성한다면 프로세스와 연관된 동작은 출력 클래스에 적용되는 동작으로 해석할 수 있다.
또한, 프로세스가 자료저장소나 외부 객체의 데이터를 읽거나 결과를 저장 하는 경우,
자료저장소나 외부 객체가 이 오퍼레이션의 대상 객체가 된다.
그리고 객체 모델에 동적 모델과 기능 모델 을 통합한 결과는 객체에 속한 모든 가능한 정보, 객체들 사이의 관계, 객체의 오퍼레이션들을 나타낼 수 있게 되어 시스템에 대한 완벽한 기술이 이루어지게 된다.
(객체들 사이의 관계는 상대 객체를 나타내는 포인터 변수를 객체의 속성으로 나타냄으로서 이루어질 수 있다.)

3. UML tool 을 사용한 OOAD 의 구현

3-1. 식당관리 프로그램

사용 Tool : StarUML 5.0
객체 : 손님, 종업원, 주방장, 프론트

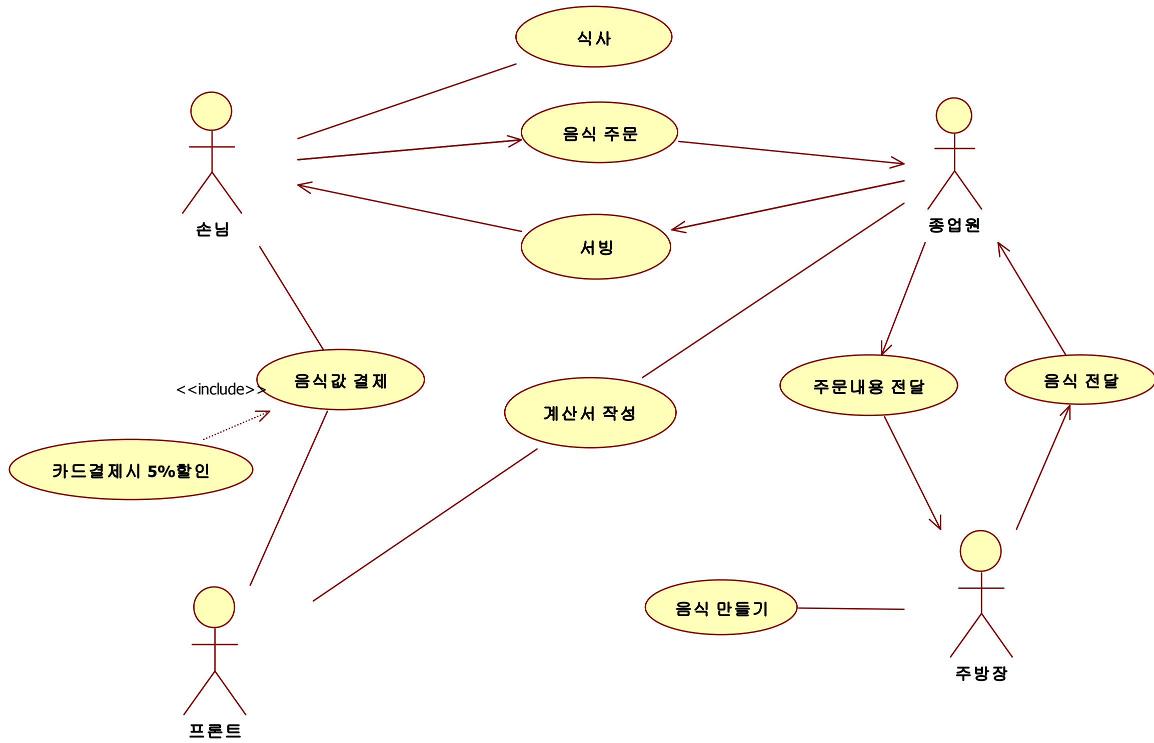
손님 : 주문, 식사, 음식값 지불

종업원 : 주문 받기, 주방에 주문내용 알려주기, 각종 서빙, 계산서 작성

주방장 : 주문내용대로 음식 만들기, 종업원에게 전달

프론트 : 계산서 받아서 음식값 받기, 현금결제 or 카드결제(카드 결제시 5% 할인)

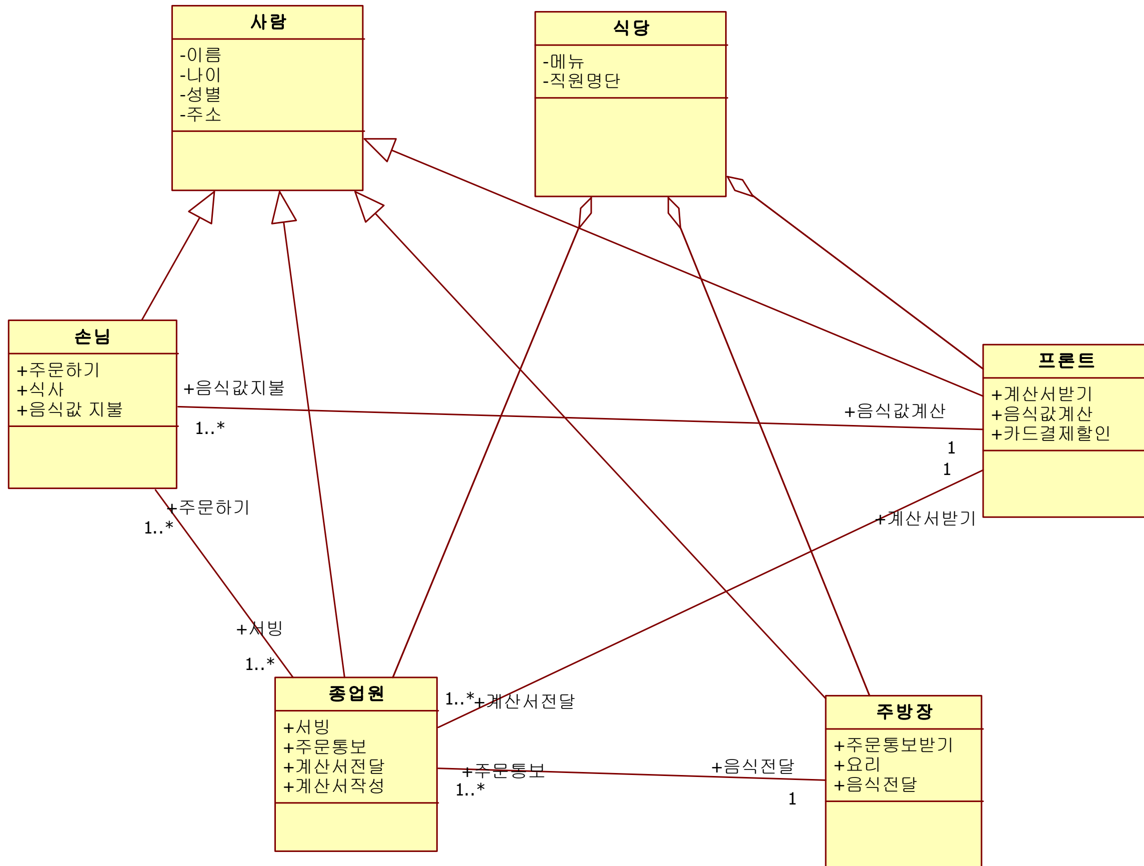
3-1-1. Use Case Diagram



식사관리 프로그램을 Use Case Diagram 으로 나타내었다. Actor 는 손님, 종업원, 프론트, 주방장의 네가지이며, Use Case 로 서로 연관되어 있다. 카드결제시 5% 할인되는 Use Case 는 음식값 결제에 포함된다.

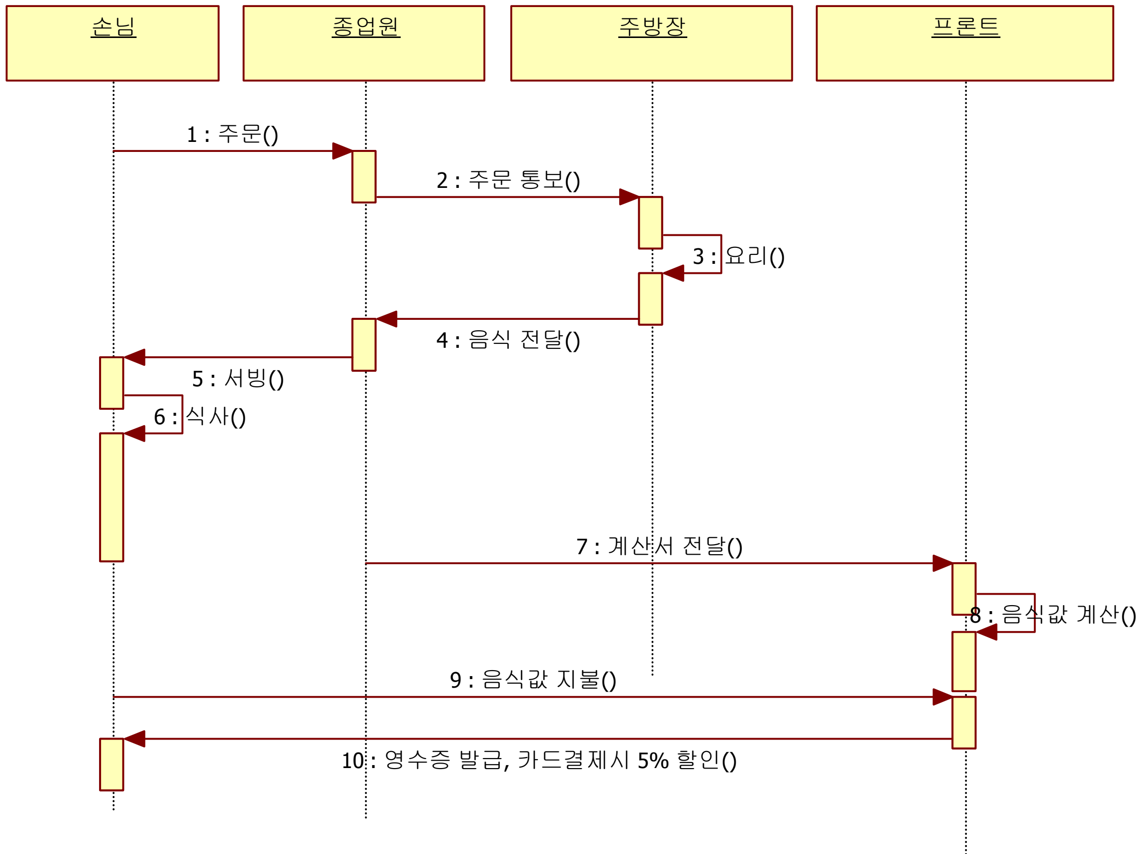
3-1-2. Class Diagram

아래 그림은 식당관리 프로그램을 Class Diagram 으로 나타낸 것이다. 손님, 종업원, 주방장, 프론트는 사람으로 일반화되고, 종업원, 주방장, 프론트는 식당이라는 집합으로 표현할 수 있다. 손님, 종업원은 여러 명일 수 있고, 주방장, 프론트는 한 명이라는 가정 하에 관계를 나타내었다.

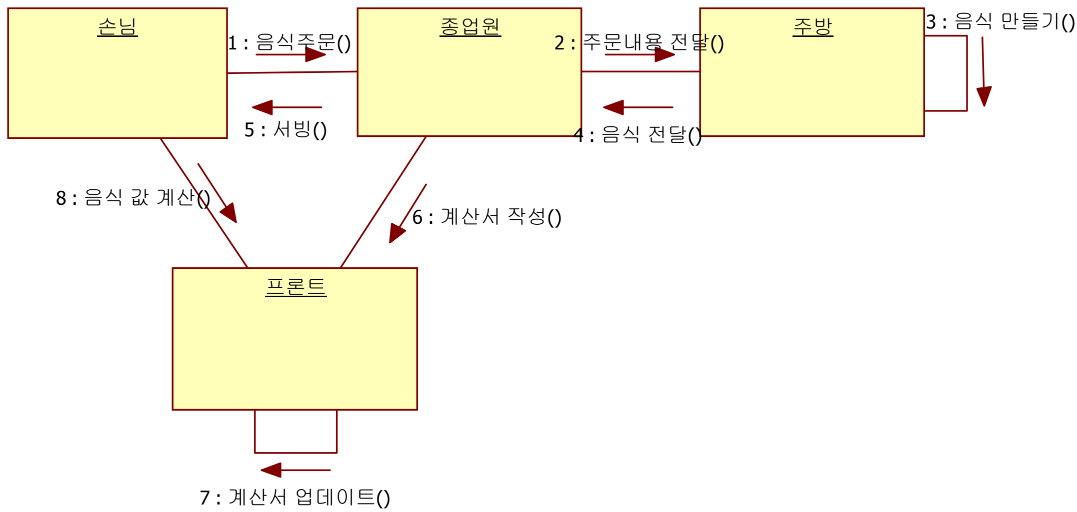


3-1-3. Sequence Diagram

아래 그림은 식당 관리 프로그램의 수행 과정을 시간의 경과에 따라 나타낸 것이다.



3-1-4. Collaboration Diagram

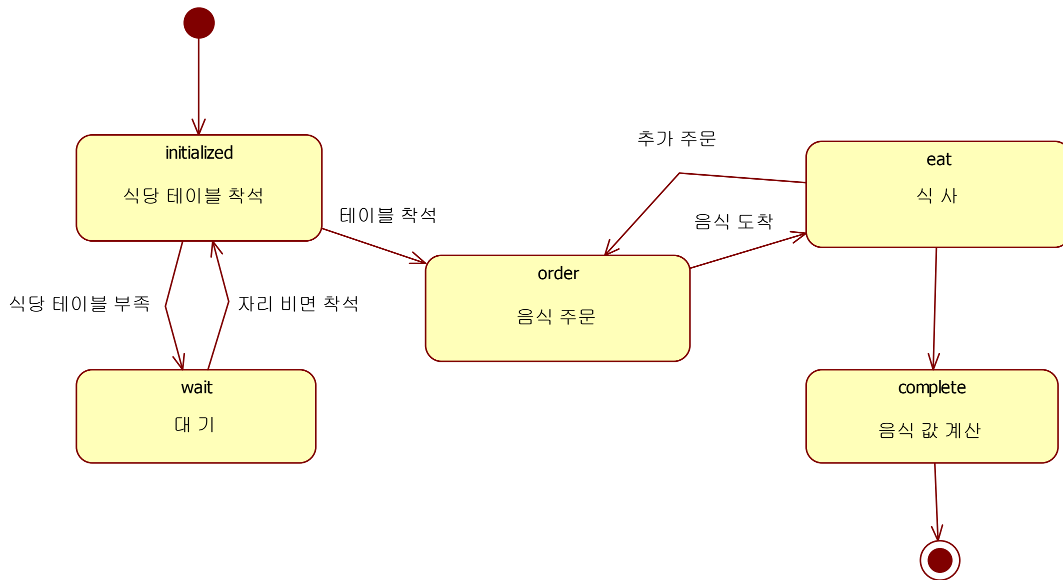


위의 그림은 식당관리 프로그램을 Collaboration Diagram 으로 표현한 것이다.

1. 음식주문() : 손님은 종업원에게 원하는 음식을 주문한다.
2. 주문내용전달() : 종업원은 손님이 원하는 음식이 원한 주문내용을 전달한다.
3. 음식 만들기() : 주방에서 손님이 원하는 음식을 만들어 낸다.
4. 음식 전달() : 주방에서 다 만든 음식을 종업원에게 전달한다.
5. 서빙() : 종업원은 다 만든 음식을 손님에게 대접한다.
6. 계산서 작성() : 종업원은 손님이 주문한 음식에 대한 계산서를 작성한다.
7. 계산서 업데이트() : 해당 손님에 대한 계산서를 업데이트 한다.
8. 음식 값 계산() : 손님이 음식 값을 계산한다.

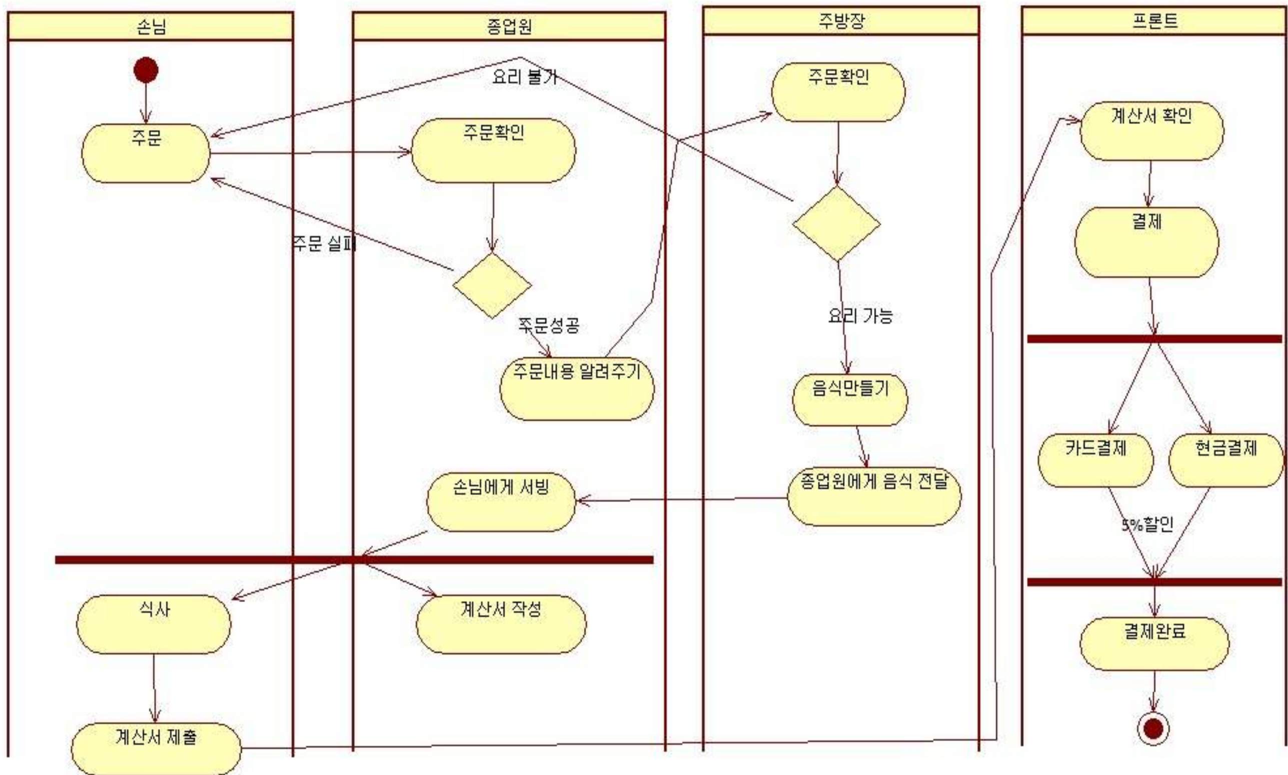
Sequence Diagram 과 Collaboration Diagram 이 모두 객체들 사이의 상호작용을 나타내므로 Interaction Diagram 이라 통칭하기도 한다. 이런 Interaction diagram 이 필요한 경우는 하나의 use case 내부에 포함된 객체들 사이의 행위를 보일 때 필요하다. 하지만 객체 상호간의 관계에 역점을 두었기 때문에 객체 하나의 행위를 정확하게 표현하기에는 부적절한 것이 단점이다.

3-1-5. Statechart Diagram



식당관리 프로그램에서 가장 의미 있는 행동양식을 보여주는 손님 객체가 행동 가능한 상태는 크게 나누어서 보면 처음 식당에 들어와 다른 손님이 차지하지 않은 테이블에 착석한 상태, 음식을 주문하는 상태, 식사 하는 상태, 음식 값을 계산하는 상태가 될 것이다.
위의 그림은 손님 객체가 취할 수 있는 여러 상태와 그들 간의 전이를 State Diagram 으로 표현한 것이다.

3-1-6. Activity Diagram

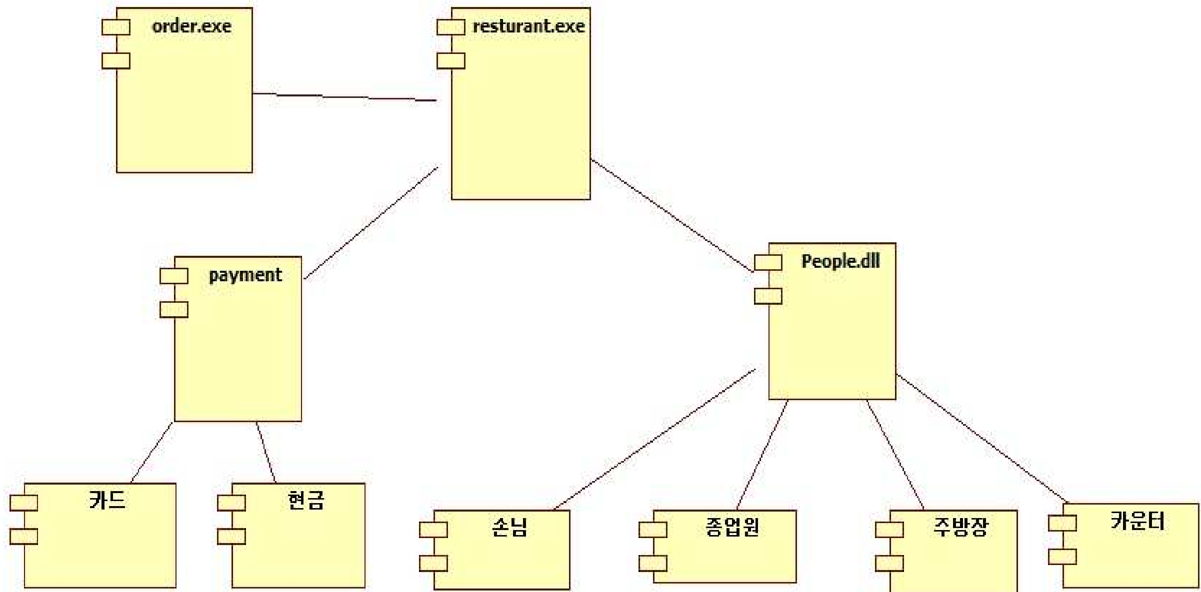


시작은 손님의 주문으로 이루어진다. 손님이 종업원에게 주문을 하게 되면 종업원은 주문을 확인한다. 주문이 잘못되면 다시 주문을 받고, 정상적으로 주문이 진행 되면 주문받은 음식을 주방장에게 알려준다.

주방장은 주문 받은 음식이 요리가 가능한지 불가능한지를 판단 한뒤, 요리가 불가능하다면 종업원으로부터 다시 손님에게 주문을 받게 한다. 요리가 가능하다고 판단이 되면 음식을 만들고, 완성된 음식을 종업원에게 전달하게 된다. 음식을 전달받은 종업원은 손님에게 서빙을 한다. 손님은 식사를 하고, 종업원은 계산서를 작성한다.

손님이 식사를 마치게 되면 프론트에 계산서를 제출한다. 프론트에서는 계산서를 확인 한 뒤 결제 과정을 거친다. 결제방법에는 카드결제와 현금결제가 있고, 카드로 결제를 하게 될 시에는 5%할인 혜택이 있다. 결제가 완료 되면 모든 과정이 종료된다.

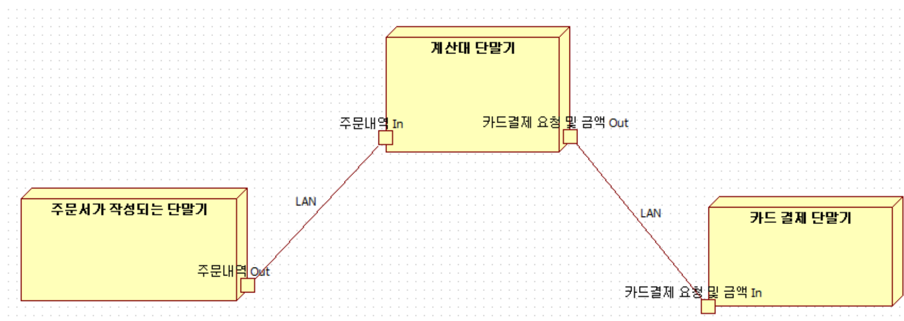
3-1-7. Component Diagram



Component 들을 의존성을 위주로 간단하게 표현 해 본 Component Diagram 이다. 시스템은 두 개의 EXE 와 하나의 DLL 로 이루어져 있다.
식당관리 프로그램은 주문시스템에 포함 되어 있다. (order.exe)
people.dll 은 사용하는 클래스들을 표현하고 있다. people.dll 에서는 손님클래스, 종 원클래스, 주방장 클래스, 카운터 클래스를 사용하고 있는 것을 확인할 수 있다.

3-1-8. Deployment Diagram

Deployment diagram 은 위에서 설명했듯이, 시스템의 software 와 hardware component 간의 물리적 관계를 표현하는 diagram 이다. 위에서 설명한 여러 가지 diagram 들을 통해서, 기본적인 것들을 모두 표현을 하였는데, 이제 그것을 배치해보겠다.



3-1-9. Object Diagram

Object diagram 은 위에서 설명했듯이, 객체와 그들 간의 관계를 표현하며, Class diagram 에 있는 요소의 Instance 에 대한 정적 Snap Shot 을 나타내는 diagram 이다.
좀 더 자세히 말하자면, 이 Object diagram 은 Class diagram 에서는 표현할 수 없는 Instance 끼리의 복잡한 관계를 보충할 때 효과적이다. 아래 그림은 모든 것을 완료한 Object diagram 의 형태이다.