소프트웨어 공학 개론

팀 레포트

- Introduction to OOAD using UML tools -

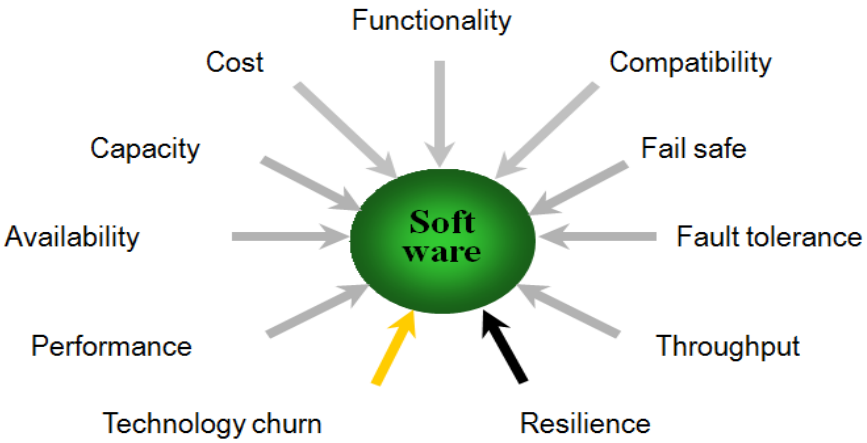
과 목 명:	소프트웨어공학개론
학 과:	컴퓨터 공학부
팀 원:	200312461 김계성, 200310405 류규현, 200412302 김무진, 200714175 이정현,
제 출 일:	20010년 10월 27일(수)
담당교수:	유준범 교수님

< CONTENTS>

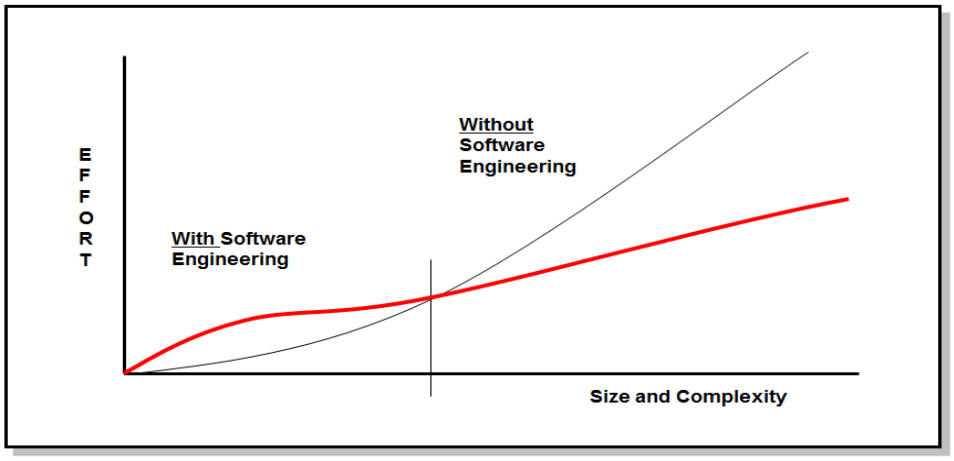
Part1. OOAD (Object-Oriented Analysis and Design)

1. 배경

- Software의 Size와 Complexity가 증가됨에 따라 Software에 대한 요구가 증가되었다.



- 소프트웨어 프로젝트 일정의 지연으로 인해 시스템 완성이 지연되거나 프로젝트의 예산을 초과, 또는 요구사항을 충족하지 못하는 시스템을 발생하는 등의 System development problems들을 의미하는 Software Crisis가 이슈가 되었고, 이에 따라 Software Engineering에 대한 중요성이 대두되었다.



- Software Crisis를 해결하기 위한 Solution으로 Rational Unified Process, Catalysis, CBD96, UML, Design Patterns, Refactoring, XP(eXtreme Programming) 등의 기법들이 등장하였다.
- 그 중, OOAD (Object-Oriented Analysis and Design)은 Software Crisis를 극복하기 위한 개발 방법 중 가장 최근에 나타난 것으로 현재까지 나타난 소프트웨어 개발의 문제점을 해결해줄 많은 장점을 보유하고 있다.

2. 정의

- OOAD(Object-Oriented Analysis and Design)란 소프트웨어를 개발하는 하나의 방법론으로 모든 소프트웨어 시스템의 주요 기본 요소를 사물을 가리키는 객체와 그 객체들을 하나의 집합으로 묶은 클래스로 구성하는 객체 지향적인 분석과 설계 방법을 말한다.
- 객체지향방법론은 데이터와 행위를 하나로 묶어 객체로 정의하고 이 객체들을 추상화 한다.
- 기존의 데이터와 행위가 분리되었던 개발 방법의 복잡성과 통합의 어려움을 극복하였다.

3. 장점

- 단일한 패러다임 : 사용자, 분석자, 설계자, 구현자 간에 단일한 언어를 사용한다.
- 유연한 아키텍쳐와 코드 재사용 (Reuse)
  : 시스템은 요구사항 변경을 수용할 수 있도록 유연성과 적응력을 갖도록 설계할 수 있다.
- 모델들은 현실세계들을 보다 더 가깝게 반영.
- 안정성(Stability)
  : 요구사항에 대한 작은 변경이 개발된 시스템 내부에 커다란 변화를 주지 않는다.

4. Key Concept

(1) Use Case Driven

- 우리는 미래의 사용자의 요구사항을 적확히 알아야 성공적인 시스템을 구축할 수 있다.
- 따라서 사용하는 Case에 대한 기능 요구사항을 Use Case Model로 작성하고, 이 Use Case Model을 기반으로 개발자는 프로그램의 Use Case를 구현함에 따라 기능적 요구사항 및 분석, 디자인, 구현, 테스트로의 추적 및 일관성을 유지한다.

(2) Architecture Centric

- Architecture는 시스템 전체에 대한 설계 모습이다.
- Architecture는 모든 Use Case를 실현(Realization)할 수 있는 대략적인 Outline을 설계해야 하며 Use Case들은 설계된 Architecture의 subsystem, class, components로 대응된다.
- Technology Churn은 계속되고 시스템은 더욱 분산화되고 병렬화 될 것이다. 결국 점점 복잡해지는 시스템과 컴포넌트들을 조립하고 이를 통제하기 위해서는 시스템의 밑그림이 되는 Architecture가 매우 중요하다.

(3) Risk Driven

- Risk는 프로젝트를 위험에 빠뜨리고 또한 성공하기 위해서 제거되어야 하는 요소이다.
- 시스템을 개발할 때 초기에 위험순위가 높은 요소를 먼저 제거하여 나감으로써 초기에 시스템에 대한 핵심이 되는 안정적인 Architectural Baseline을 구축하여 프로젝트의 안정성을 높이고 프로젝트 중반 이후에는 이미 안정적으로 구축된 기본 Architecture 위에 나머지 부분을 쌓아 올라감으로써 튼튼한 Architecture를 갖은 유지 보수하기 좋은 시스템을 구축한다.
- 특히 중요한 위험요소는 수행 능력, 신뢰성, 이용성, 시스템 인터페이스, 이식성 이다. 이런 위험 요소는 프로젝트 초기 단계에는 확연히 드러나지 않기 때문에 프로젝트가 진행 되면서 이를 찾아 적절한 시점에 해결해야 한다.

Part2. Software Develop Process

1. Iteration & Increment

(1) 정의

- Interactive approach는 반복을 통해 Risk 요소들을 제거하는 기법이다.
- 하나의 반복을 완성하기 위하여 계속적으로 프로그램을 진화시킨다.
- 따라서, 예측 가능하며, Disruption을 적게 하면서 요구사항의 변경을 수용한다.
- Documentation가 아닌 실행 가능한 prototype의 Evolving에 바탕을 두고 있다.
- 전체 프로세스에 걸쳐 사용자와 고객을 참여시킨다.
- Risk Driven에 의해 진행된다.

(2) 전제

- 개발 전반부에 (Up Front) 필요한 모든 정보를 가질 수 없으며 개발 기간 동안에 변경되는 것들이 있을 수 있다. 그렇기 때문에 다음과 같은 사항이 발생할 수 있음을 고려해야 한다.

✓ 몇몇 위험들은 계획된 대로 제거되지 않을 것이다.
✓ 진행 중에 새로운 위험들을 발견할 것이다.
✓ 어떤 것들은 Rework이 필요할 것이다.
✓ 한 반복을 위하여 개발된 코드 중 몇몇 라인들은 버려질 것이다.
✓ 진행 중 요구사항이 변경될 것이다.

(3) Interactive Approach의 3가지 중요한 특성(Features)

- 지속적인 통합 (Continuous Integration)
  : Delivery Data에 이르러서 한꺼번에 One Lump 수행되지 않는다.
- 빈번한 실행 가능한 릴리즈 (Frequent, Executable Release)
- Progress를 보여주어 위험요소들을 제거함.
  : 진행상황은 Documentation나 공학적인 추정(Engineering Estimates)이 아닌 Product로 측정.

(4) Interaction Plan

- 반복을 시작하기 전, 다음 사항에 바탕을 둔 반복의 일반적인 목적(Objectives)을 수립해야 한다.

이전 반복의 결과 / 프로젝트에 대한 최신의 위험 평가(Risk Assessment) / 평가기준(Evaluation Criteria)의 결정 / Development Plan에 포함된 상세한 반복 계획의 준비 / 진행사항(Process)을 감시하기 위한 Intermediate Milestone / Walkthrough / 검토 회의(Review)

1) 요구사항 수집 (Requirements Capture)
- 이 반복에서 구현될 유스케이스의 선정 및 정의
- 추가로 발견되는 도메인 클래스들과 연관을 반영하여 객체 모델을 수정
- 반복을 위한 테스트 계획의 개발

2) 분석 및 설계(Analysis & Design)
- 이 반복에서 개발되거나 수정될 클래스의 결정.
- 추가로 발견되는 도메인 클래스들과 연관을 반영하여 객체 모델을 수정.
- 필요하다면 아키텍쳐 문서를 수정.
- 테스트 절차(Test Procedure)의 개발 시작.

3) 구현(Implementation)
- 설계 모델로부터 코드를 자동으로 생성
- 연산을 위한 코드를 수작업으로 생성
- 테스트 절차들을 완성
- 단위 테스트(Unit Test)와 통합 테스트(Integration Test)를 개최

4) 테스트

- (이전 Release 에서 만들어진) 시스템의 나머지 부분들과 개발된 코드들을 통합하고 테스트
- 테스트 결과를 수집하고 검토
- 평가기준(Evaluation Criteria)에 따라 테스트 결과들을 평가
- 반복 평가(Iteration Assessment)를 개최
- Release Description의 준비
- 코드와 설계 모델을 동기화(Synchronize)
- 통제된 라이브러리(Controlled Libraries)내에 반복의 제품(Product)들을 배치

(5) Iteration의 평가

- 반복 계획 동안에 수립된 평가 기준들에 의하여 반복 결과들을 평가한다.

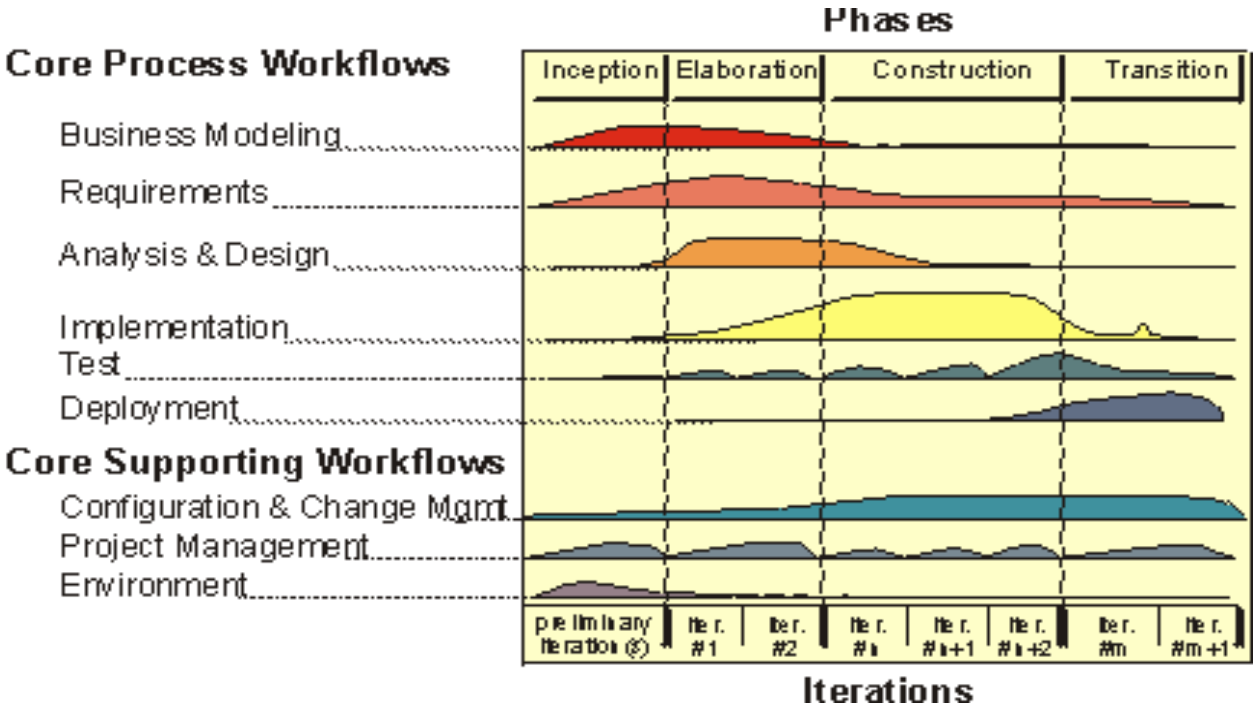
기능(Functionality) / 성능(Performance) / 용량(Capacity) / 품질 평가(Quality Measures)

- 해당 반복 동안에 발생되는 외부의 변경사항을 고려한다.

  ex) 요구사항, 사용자 요구사항, 경쟁사의 계획(Competitor's Plans) 등의 변경.

- 필요하다면 재 작업이 필요한지를 결정하고 이를 남아있는 반복들에 배정한다.

2. 일반적인 소프트웨어 개발 절차와 단계별 위험 프로파일



- **일반적인 소프트웨어 개발 절차**

<table>
<tr><td>요구사항 정의</td><td colspan="2">분석</td><td>설계</td><td colspan="2">구현</td></tr>
<tr><td rowspan="3">시스템에 대한
요구사항 파악</td><td colspan="4">요구사항을 충족시키는 시스템의 구축</td></tr>
<tr><td colspan="3">시스템에 대한 고안</td><td rowspan="2">각종 소스코드 개발
및 배치</td></tr>
<tr><td>플랫폼 무시
기능적 요구사항만 고려</td><td colspan="2">플랫폼 고려
비기능적 요구사항도 고려</td></tr>
</table>

(1) Inception Parse(개념화 단계)

- 어떤 개념에 대한 증명을 만들어서 프로젝트의 위험을 그룹화(Bracket)

1) Inception 단계의 목적

- 새로운 시스템의 Business Case를 설립하거나 기존 시스템의 주요 수정사항을 설정
- Business case를 고려하여 시스템의 범위(scope)를 규정하고 한계(limit)를 정의한다.
- 전체 시스템을 guide할 수 있는 candidate architecture를 잡는다.
- 위험요소(critical risk)를 식별하고 대비한다.(초기대응)
- 경제적인 관점에서 Business case을 평가한다.
- Project team에게 개발환경에 필요한 교육을 실시한다.

2) Inception 초기 단계의 준비사항

- Inception 단계에 초기에 프로젝트의 concept을 이해하고, 요구사항에 대한 작업량의 개략적인 측정과 자원의 확보 및 스케줄을 계획하여야 한다.
- Inception Phase를 계획 할 때는 관련된 모든 사람으로부터 정보를 수집하여야 하며 요구사항

을 명확이 뽑아내기 위한 방법에 관한 계획과 이것을 바탕으로 초기 아키텍쳐를 잡아서 평가해 보기 위한 계획이 잡혀져야 한다.

- 다만, 초기계획은 요구분석을 해가면서 바뀌는 부분이므로 완벽하게 계획할 수 있는 것은 아니다

3) Inception 단계의 산출물

- Required products
  : 프로젝트를 위한 핵심 요구사항
  : 초기의 위험 평가(위험요소 우선순위 결정)
- Optional products
  : 개념적인 프로토타입
  : 초기의 문제영역 모델(10%~20%정도)

(2) Elaboration parse(상세화 단계)

- 최종사용자와 도메인 전문가와 함께 시나리오를 조사하여 시스템의 영역(System's Scope)과 원하는 행위(Desired Behavior)에 대해 일반적인 이해를 한다.
- 시스템의 아키텍쳐를 수립한다.
- 시스템 전체의 이슈를 처리하기 위한 공통 메커니즘을 설계한다.

1) Elaboration 단계의 목적

- Elaboration단계의 주요 목적은 개발할 Domain에 대한 요구사항으로부터 문제 영역을 분석하고 중요 아키텍쳐 기반을 수립하며 그에 따른 중요 위험 요소를 제거합니다. 그리고 전체 Domain에 대한 대략적인 프로젝트 계획을 수립하는 것을 목적으로 한다.

문제영역 분석 / 아키텍쳐 기반 설립 / 프로젝트 계획 수립 / 프로젝트의 가장 위험한 요소 결정 / 가장 큰 위험요소 제거 / 프로젝트의 개괄적인 계획 수립 / 적합한 기반 아키텍쳐 설정

2) Elaboration 초기단계의 준비사항

- Elaboration단계를 시작하게 전에 준비 되는 사항으로 Elaboration 단계의 plan과 Team구성을 하고 개발 환경을 설정한다.
- inception과 Elaboration은 구현 환경이 변화 될 수 있다.
- Elaboration에 대한 평가 기준을 미리 설정을 한다.

3) Elaboration 단계의 산출물

Use Case Model / Supplementary Requirement / Software Architecture document / Risk List와 비즈니스케이스 / 전체 프로젝트에 대한 개발 계획 /

- Use Case Model

  : 최소 80%의 완성도

  : 모든 Use Case와 액터가 식별되고 대부분의 Use Case에 대한 기술이 이루어진다.
- Suppliementary Requirement

  : 성능과 같은 외적인 요구사항 및 특정 Use Case에 관련되지 않은 요구사항을 나타낸다
- 소프트웨어 아키텍쳐의 기술와 실행 가능한 아키텍쳐를 반영한 프로토타입
- 반복(Iteration)과 각각의 반복마다의 평가 기준을 망라한 개략적인 프로젝트 계획을 포함한 전체 프로젝트에 대한 개발 계획
- User-interface prototype과 실행 가능한 아키텍쳐를 반영한 프로토타입 등이 있을 수 있다.

4) Elaboration 단계의 평가

- Elaboration의 완료 시점에서 만들어진 산출물을 평가 하여 평기 기준에 미달될 경우 작업이 중지 되거나 또 한번의 반복내지는 재 고려 될 수 있다.
- 이 시점에서는 시스템의 목표와 범위, 아키텍쳐의 선정, 주요 위험요소의 해결방법에 대해 자세히 검사 한다.

✓ 아키텍쳐가 안정적인가?
✓ 프로토타입의 시연을 통해 주요한 위험요소가 파악되고 신뢰성 있게 해결되었는지를 알 수 있었는가?
✓ 시스템 완성 단계를 위한 계획이 자세하고 정확하게 수립되었는가?
✓ 평가를 위한 기준치가 믿을만하게 산출되었는가
✓ 실제 사용된 자원이 계획과 비교하여 허용 가능한 범위에 있는가?

(3) Construction parse(구축 단계)

- 아키텍쳐를 정제(Refine)한다.
- 위험에 의하여 반복을 수행한다.
- 지속적인 통합(Continuous integration)을 수행한다

1) Construction 단계의 목적

- 점증적으로 사용자 조직에 전이할 수 있는 완성된 소프트웨어 제품을 개발한다.
- 남아있는 요구사항을 명확히 하고, Architecture baseline에 살을 붙여 시스템을 완성한다.
- Inception과 Elaboration이 연구적인 성격이 강한데 반해 Construction은 개발의 성격 강하다.
- 자원관리, 비용,일정, 품질의 최적화가 강조한다.
- Project manager, Architect, senior developer가 use case들에 순위를 분류하여 순서대로 개발을 진행한다.
- 개발 진행 단계에서의 정확한 Risk List 관리한다.
- Architect는 Construction이 Architecture에 맞게 진행되는 지를 살피고 필요하다면 Architecture를 수정한다.

2) Construction 초기단계의 준비사항

- Construction phase에 대한 plan을 실제 환경의 변화에 맞게 수정.
- 인력 구성
- Use Case engineer, component engineer, test engineer, system integrator, integration tester, system tester, and architect
- 평가 기준 수립
- Product release는 사용자 환경에 배포해도 될 정도로 충분히 안정적인가?
- Stakeholder들은 Transition phase에 대비되어 있는가?
- 추가 부담 발생은 허용범위 내에 있는가?

  ex) User material – transition phase의 사용자들을 지원할 수 있을 정도로 충분한가?

3) Construction 단계의 평가

- 계획된 것 중 완료된 것에 대한 확인
- 다음 Iteration에서 완료 시킬 일에 대한 계획 수립
- Build가 다음 Iteration으로 나아가도 좋을 정도인지에 대한 결정.
- Risk List의 보완
- 마지막 Iteration후 Product가 beta버전으로써 문제가 없는지에 대한 확인.
- 프로젝트 계획 보완
- Transition phase에 대한 계획 수립

4) Construction 단계의 산출물

- A stream of executable release
- Behavioral prototypes
- 품질보증 결과
- 시스템과 사용자 문서
- Deployment plan
- 다음 반복 단계를 위한 평가 기준

(4) Transition parse(전이 단계)

- 사용자 승인을 촉진(Facilitate user acceptance)
- 사용자 만족을 평가(Measure User Satisfaction)

1) Transition 단계의 목적

- Product를 작동환경에 적용시키는 것이 주 포커스이다..
- Operating site에서 오는 feedback에 대한 꾸준한 모니터링이 필요하다.
- 큰 수정보다는 자잘한 보완이 주 목적이다.
- 제품 외적인 작업도 포함(DB Conversion, 환경 세팅, 사용자 교육 등)한다.
- 다수의 사용자를 위한 application의 경우 installation program과 Help Desk를 통한 지원으로 대신한다.

2) Transition 초기단계의 준비사항

- 계획 수립
- 베타 테스트 documentation준비, 베타 사용자 선택 등
- 베타 테스트의 feedback 해결을 위한 인원의 준비, 인력구성
- Construction phase와 비슷한 인력구성을 가짐
- 평가 기준 수립

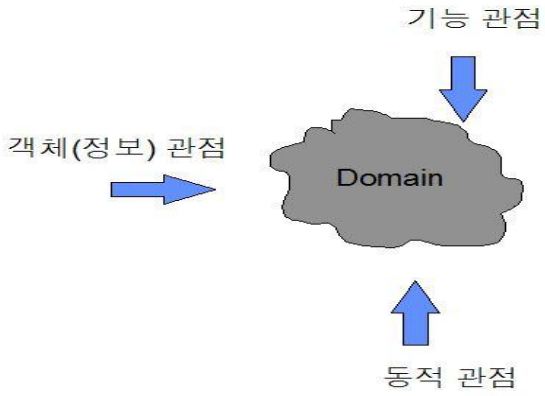
✓ 베타 테스터들은 모든 기능에 대한 테스트를 수행했는가?
✓ 제품은 고객이 제시한 테스트 조건을 만족하는가?
✓ 사용자를 위한 산출물은 만족할 만한 수준인가?
✓ Course 산출물은 사용 가능한가?
✓ 고객과 사용자는 제품에 만족하는가?

(5) Post deployment cycles(배치 이후 과정)

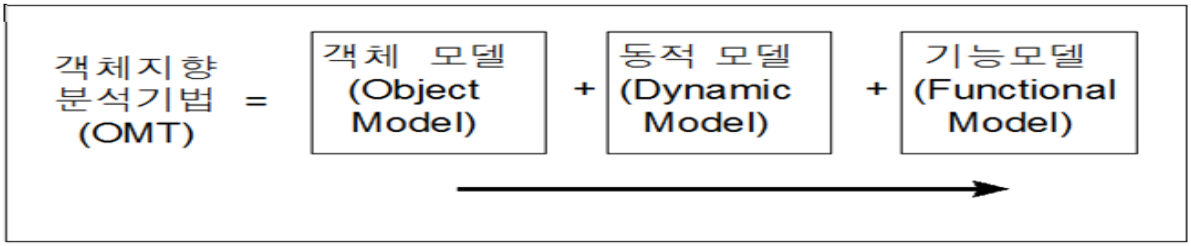
- 진화적인 접근방법을 진행 (Continue evolutionary approach)
- Preserve architectural integrity

Part3. OOA (Object-Oriented Analysis)

(1) 객체지향 분석의 3 관점



- 시스템의 서로 다른 관점을 보여주고 있고, 이를 통합하여 소프트웨어를 설계해야 한다.
- 이들을 통합, 유연성과 적응력을 가진 우수한 품질의 소프트웨어를 만드는 최적의 방법이다.
- 객체지향 분석기법은 시스템의 요구 사항을 분석하기 위해, 세 가지 모델링 기법을 단계별로 적용하여 그 결과를 통합한다.



- ✓ 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 라이프 사이클을 보여준다.
- ✓ 기능 모델링 : 각 객체의 형태 변화에서 새로운 상태로 들어갔을 때 수행되는 동작들을 기술하는데 사용한다.

- 객체지향 개발 방법은 소프트웨어 공학에서 추구하는 많은 장점들을 제공한다.
- 객체지향 개발 방법을 제대로 활용하기 위해서는, 기존의 기술과는 다른 높은 차원의 기술력을 분석가나 디자이너에게 요구한다.
- 소프트웨어 개발 과정의 이해, 정보 모델, 동적 모델, 기능 모델에 대한 지식이 요구 모델들 사이의 연관성을 바탕으로 모델링의 결과를 통합할 수 있어야 한다.

(2) 객체지향 분석 기법 개요

- 객체지향 개발 방법은 우리가 태어나면서부터 배워온 객체 개념에 기초한다.
- 우리 인간(객체)은 자라면서 상대방(객체)을 인식, 그의 속성을 찾아내고 그와의 관계 규명한다.
- 인간이 살아가는 과정은 자신(객체)과 다른 사람(객체)의 관계를 넓혀 가는 객체지향 과정
- 특히 객체지향 분석기법은 기존의 분석기법에 비해 실 세계의 현상을 보다 정확히 모델링 할 수 있어 어려운 응용 분야들에 적용이 가능
- 분석과 설계의 표현에 큰 차이점이 없어 시스템의 개발을 용이하게 해준다.
- 또한 분석, 설계, 프로그래밍의 결과가 큰 변화 없이 재사용될 수 있어 확장성이 용이하고 시스템 개발 시 시제품이나 나선형 패러다임의 적용이 가능하다.

(3) 객체지향 분석의 목적

- 요구 사항들을 더욱 명확히 이해한다.
- 요구사항을 구조화 한다.
- 개발자 관점에서 요구 사항을 좀 더 깊이 분석하여 기술.
- 기능적(functional) 요구 사항을 Software Concept으로 변환(translating)하는 작업을 한다.
- Conceptual Object Model
- Design에 대한 Input 으로서의 역할을 한다.
- 보다 세밀히(precise) 요구 사항을 기술한다.
- 사용자(customer)에서 개발자(developer)관점으로 전이된다.
- Refining and structuring the requirements
- Analysis 에 의한 구조는 (비기능적 요구사항을 고려한) 설계, 구현 시 변경(compromise)될 수 있다.

(4) Requirement Capture

(1) 목적

- 개발자가 시스템의 요구사항의 해단 이해를 높인다.
- 개발할 시스템의 영역을 정의한다.(제한한다.)
- 시스템이 무엇을 하는지 고객과 사용자에게 동의를 구한다.
- 반복을 계획하는데 기초가 된다.
- 시스템에 대한 UI prototype을 정의한다.

(2) Use Case Model

1) Use Case

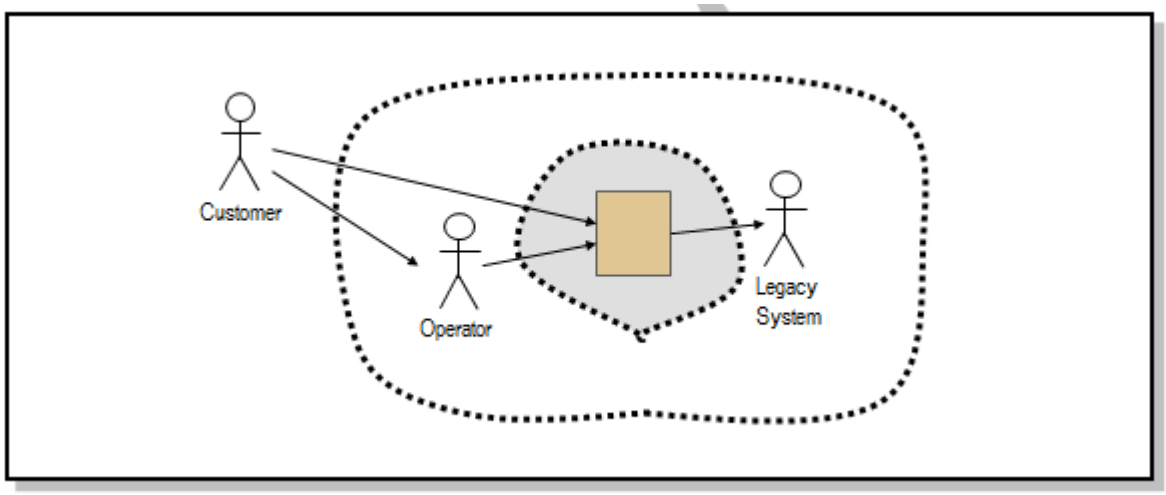
- A sequence of actions a system performs.
- 그 자체로 완전하고 하나의 의미를 갖는 Flow.
- Functional Requirement
- Stereotype: <<include>> , <<extend>>

2) Actor

- 액터는 개발하고 있는 시스템과 상호 작용하는 누군가(Someone) 혹은 무엇(Something)을 의미한다.
- 시스템과 상호 작용하는 Type, Role.
- 시스템과 정보를 능동, 수동적으로 교환한다.
- 시스템의 일부분(a part)은 아니다.
- 사람일 수도 있고 System일 수도 있다.

3) System Boundary

- 개발할 시스템에 대한 영역을 결정한다.
- 누가, 무엇이 시스템과 상호 작용하느냐에 따라 영역이 결정된다.



4) Use Case Modeling

- Use Case Model은 개발자와 사용자 사이에 요구사항을 명확히 이해하고, 상호 의사 소통을 하기 위한 수단이다.
- 개발할 시스템의 Internal View 이다.

5) Use Case Specification

- Use-Case Scenario

  : 사용자 관점에서 상세한 유스케이스의 시나리오를 기술한다.

  (무엇이 입력되고 무엇이 보여지는지)
- 100% 사용자 관점.
- 시스템이 무엇을 하는 지 기술
- Flow of Events
  - ✓ Main Flow
  - ✓ Alternative Flow
  - ✓ Activity Diagram
- Special Requirement
- Pre-conditions
- Post-conditions
- Extension Point

6) Use Case Survey

- Use Case 모델 안에 어떤 Use Case와 Actor 가 있는지 기술한다.
- Use Case Package에 대한 구성을 기술한다.
- *유스케이스 모델에 대한 시나리오를 기술한다.*
- The Use Case Model Hierarchy
- Scenario of the Use Case Model
- Diagram of the Use Case Model

7) Supplementary Specification

- 목적: 시스템의 비기능적인 요구 사항인 개발 환경 및 소프트웨어의 질(Quality), 관리 계획 등을 결정, 명시하여 구현 환경을 이해한다.
  - ✓ Functionality
  - ✓ Usability
  - ✓ Reliability
  - ✓ Performance
  - ✓ Supportability
  - ✓ Design Constraint

8) Glossary

- 프로젝트에서 사용되는 주요 어구(term)을 정의한다.
- Domain expert와 developer 사이의 통신 수단으로서 유용하게 사용된다.
- Inception, Elaboration 단계에 95% 이상 구축 되어야 한다.
  (프로젝트 초기에 common terminology에 대해 정확히 정의 되어야 한다.)

(5) Analysis Model

(1) 목적

- 요구 사항들을 더욱 명확히 이해한다.
- 요구사항을 구조화 한다.
- 개발자 관점에서 요구 사항을 좀 더 깊이 분석하여 기술.
- 기능적(functional) 요구 사항을 Software Concept으로 변환(translating)하는 작업을 한다.
- Conceptual Object Model

- Design에 대한 Input 으로서의 역할을 한다.
- 보다 세밀히(precise) 요구 사항을 기술한다.
- 사용자(customer)에서 개발자(developer)관점으로 전이된다.
- Refining and structuring the requirements
- Analysis 에 의한 구조는 (비기능적 요구사항을 고려한) 설계, 구현 시 변경(compromise)될 수 있다.

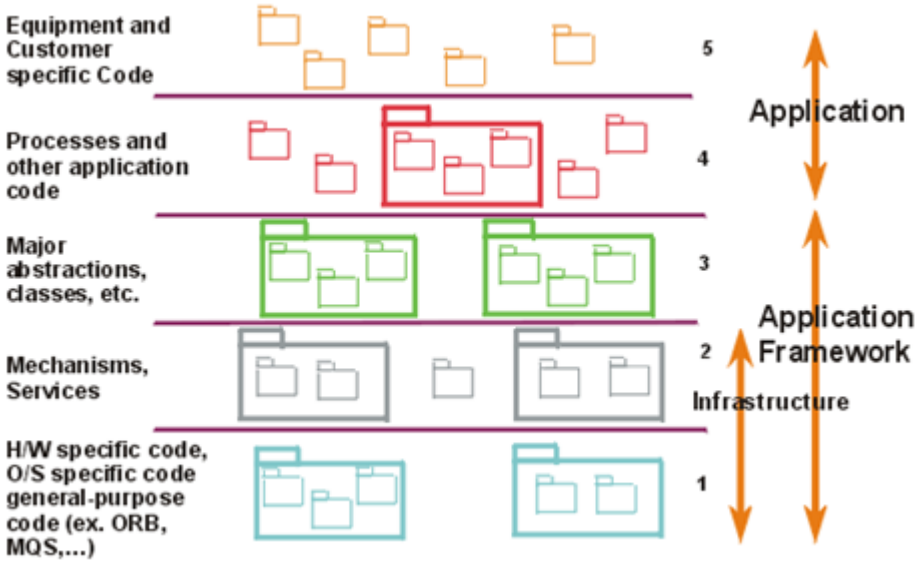
(2) Use Case Model VS. Analysis Model

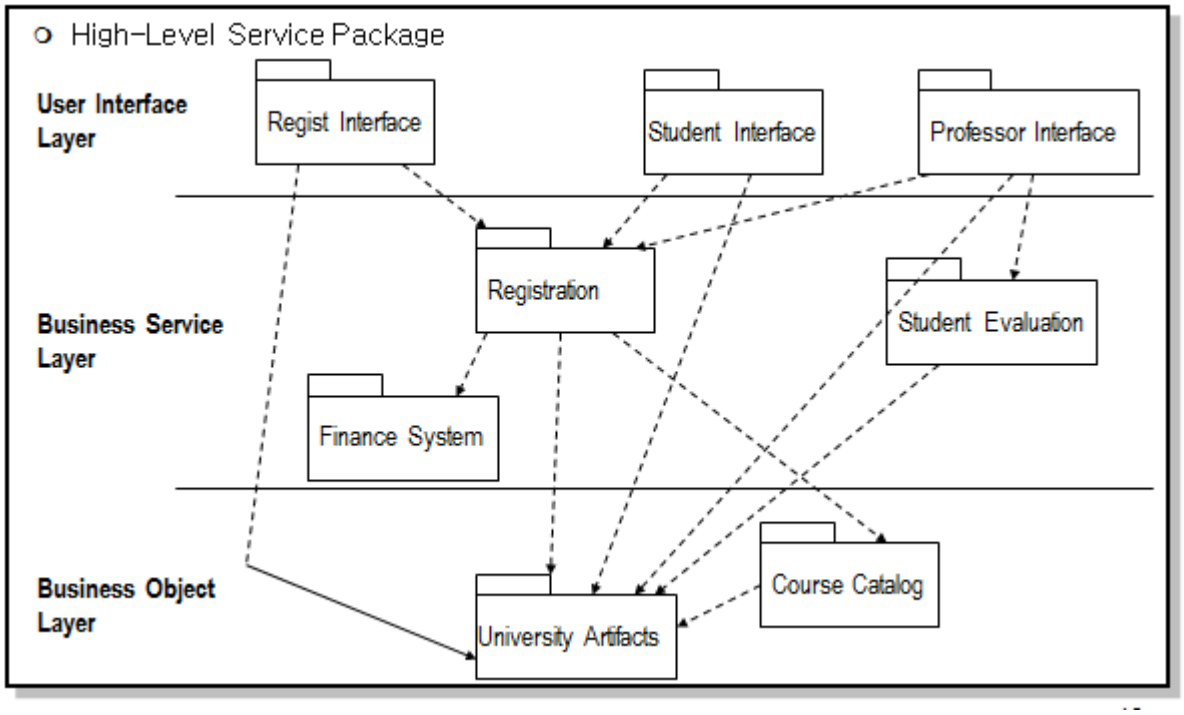
Use Case Model	Analysis Model
사용자(customer)의 언어를 사용하여 기술한다. (사용자 관점)	개발자의 언어를 사용하여 기술한다. (개발자 관점)
시스템의 External View	시스템의 Internal View
Use-Case에 의해 구성된다	시스템의 기능적 요소들이 어떻게 실현(realization)되는지 대한 개괄(outline)한다.
시스템의 기능적 요소를 수집한다.	Many Layers
분석 되어질 Use-Case를 정의한다.	Use-Case Model로부터 각각의 Use-Case를 분석하여 Use-Case Realization을 정의한다.

(3) Architectural Analysis

- Outline the Analysis Model
- Analysis Package를 설계(identify)함으로써 시스템을 구조화 한다.
- Design Model에 대한 초기 구조(initial architecture)를 정의한다.
- 초기의 package들과 이들의 dependency 그리고 이 package들을 layer에 매핑 시킨다.

1) Develop Architecture Overview
2) Survey Available Assets
3) Define the High-Level Organization of Subsystems
   - Layered Pattern





4) Identify Analysis Mechanisms

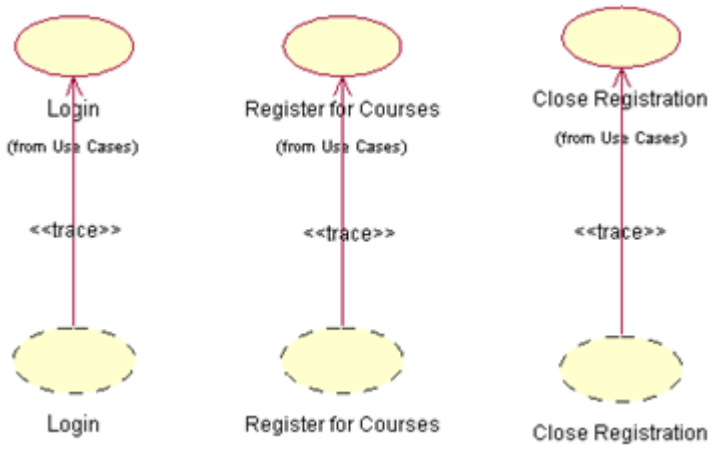
- Software Architecture의 중요한 측면 중 하나는 설계자가 객체에 생명을 주기 위한 Architecture Mechanism을 정의하고 선정하는 것이다.

5) Identify Key Abstraction

- 시스템이 다룰 주요 key entity들을 명시한다.
- Use-Case Analysis 단계에서 지금까지 분석된 시스템에 대한 기본적인 지식을 기반으로 초기 entity class를 정의 한다.
- 분석, 디자인 단계에서 refine 될 것이므로 너무 많은 시간을 낭비 하지 않는다.

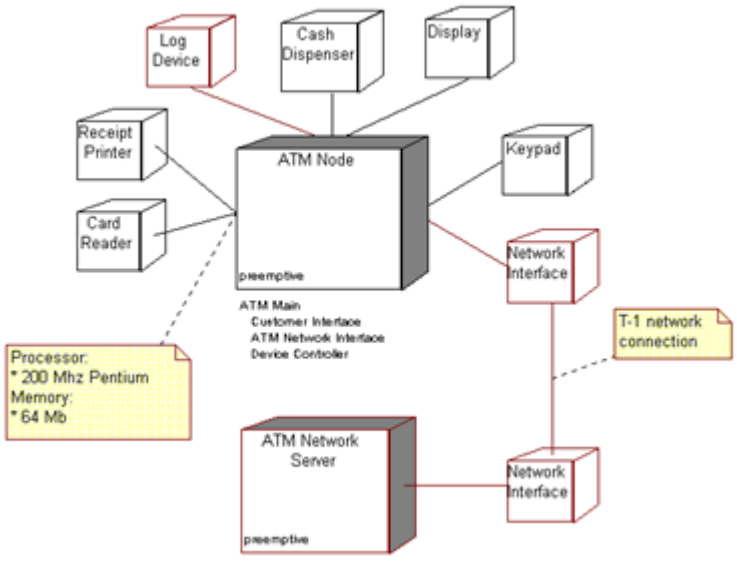
6) Create Use-Case Realization

- Use Case는 분석, 설계 모델에서 해당 시나리오에 참여하는 클래스들로서 표현되어진다.
- Use Case 의 Internal View



7) Develop High Level Deployment Model

- Deployment Diagram



8) Review the Results

(4) Use Case Analysis

1) Supplement the Use-Case Descriptions

- 사용자(customer)관점으로 작성된 flow of event를 개발자 관점, 시스템 내부 관점으로 상세화 시켜 재 기술한다.

2) For each use case realization

✓ Find Analysis Classes from Use-Case Behavior

- Analysis Class를 도출한다.
- Analysis Class는 책임(Responsibility), 행동(Behavior)을 갖고 있는 시스템의 요소에 대한 개념적(Conceptual) 모델이다.
- 시스템의 객체 모델에 대한 first-draft, rough-cut.
- Analysis Class는 그 역할에 따라 3가지 형태로 나뉜다. (boundary, control, entity)
- 각각의 형태로 구분된 Analysis Class는 stereotype, << >>,을 붙여 명시한다.

✓ Distribute Behavior to Analysis Classes

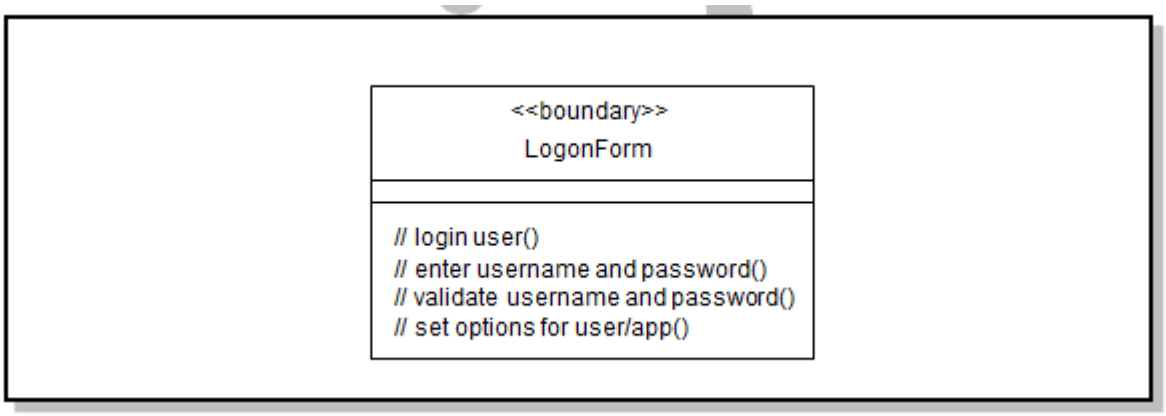
- Analysis 객체들 사이의 상호 작용을 명시한다.
- Flow of event를 통해 발견된 행동(behavior)을 표현함으로써 각 Analysis Class에 대한 책임(responsibility)를 명시한다.
- Collaboration Diagram.
- 한 Use-Case에 대한 Flow of event는 하나 이상의 Collaboration Diagram으로서 표현된다.
- Flow of Event – Analysis

Collaboration Diagram	Sequence Diagram
✓ To model flows of control by organization ✓ Structural collaboration of objects ✓ Use-Case Analysis	✓ To model flows of control by time ordering ✓ Explicit sequence of messages ✓ Real-time specification and complex senarios ✓ Use-Case Design

3) For each resulting analysis class

4) Describe Responsibilities

- 클래스에 정의된 행위에 대한 응축된 집합니다.
- 설계시 하나 이상의 operation으로 진화(evolve) 된다.
- Collaboration Diagram을 작성하면서 분석된 message으로부터 나온다.
- '//'말머리를 붙인다

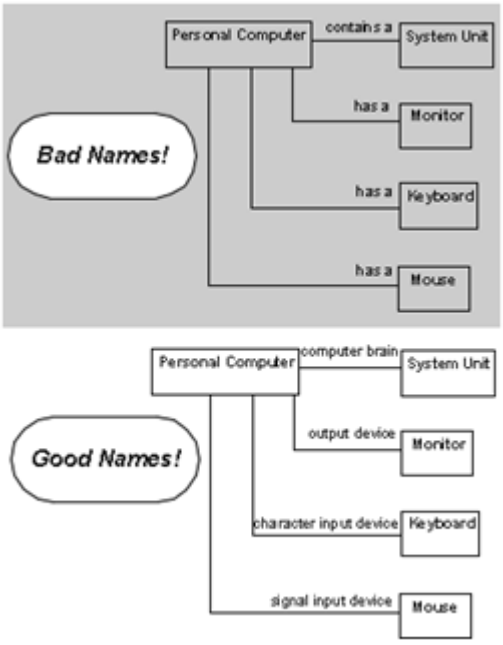


5) Describe Attributes and Associations

✓ Describe Attributes

- 정보를 저장한다.
- 어떠한 책임(responsibility)을 갖지 않는 atomic thing.
- 이 정보는 operation에 의해 get, set 되어야 한다.
- Attribute : boundary, control, entity

✓ Association

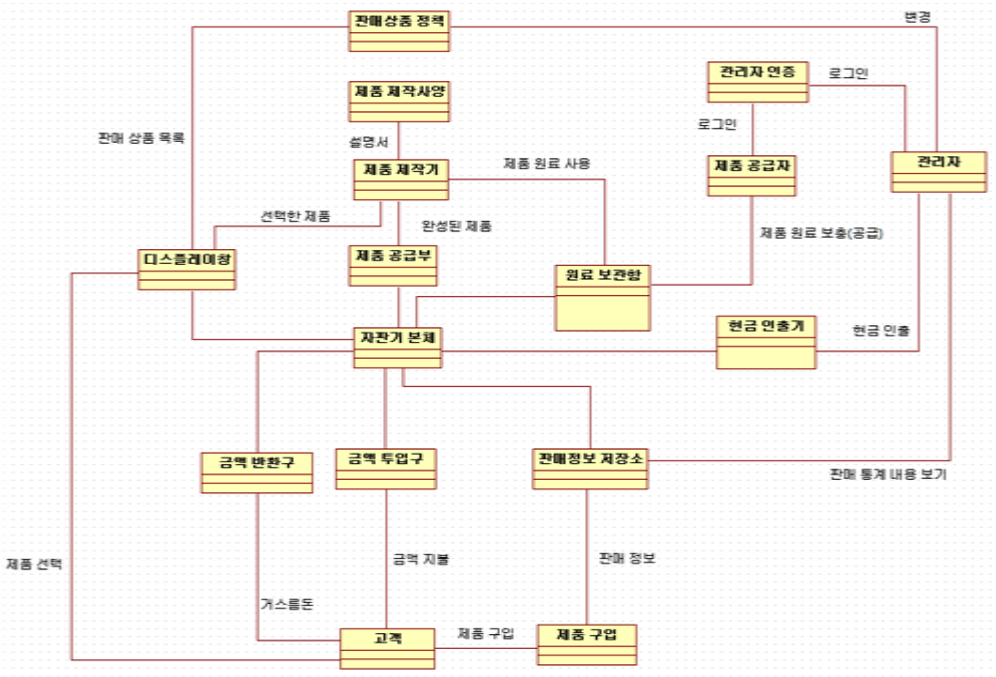


6) Define Attributes

7) Establish Associations between Analysis Classes

Category	Examples
A is a physical part of B	디스플레이창 – 자판기본체, 제품 공급부 – 자판기본체, 원료보관함 – 자판기본체, 현금인출기 – 자판기본체, 판매정보저장소 – 자판기본체, 금액투입구 – 자판기본체, 금액 반환구 – 자판기본체
A is a logical part of B	제품 제작사양 – 제품 제작기
A is physically contained in/on B	원료 보관함 – 제품 제작기
A is logically contained in B	판매 상품정책 – 디스플레이창
A is line item of a transaction or report B	제품 제작기 – 제품 공급부
A is known/logged/recorded/reported/captured in B	제품구입 – 판매정보 저장소
A uses or manages B	고객 – 금액 투입구, 고객 – 금액반환구, 관리자 – 현금인출기, 제품공급자 – 원료보관함, 관리자 – 판매상품정책, 관리자 – 판매정보저장소, 고객 – 디스플레이창
A communicates with B	판매상품정책 – 디스플레이창, 디스플레이창 – 제품 제작기
A is related to a transcation B	고객 – 제품구입

8) Describe Event Dependencies between Analysis Classes



9) Qualify Analysis Mechanisms

- Class에 적용될 Analysis Mechanism을 정의한다.
- Analysis Mechanism을 어떻게 적용할 것인지에 대한 추가적인 상세한 정보를 기술한다.

10) Evaluate the Results of Use-Case Analysis

Part4. OOD (Object-Oriented Designed)

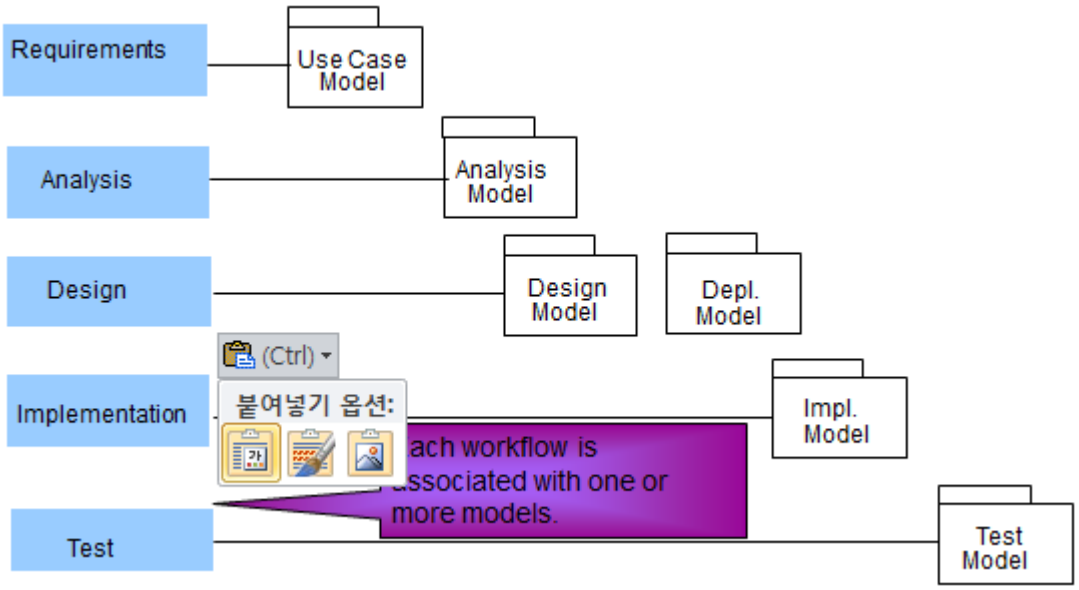
1. Modeling

(1) Model

- 모델이란 하나의 시스템을 개발하는데 있어 현실을 추상화 시켜 놓은 청사진을 말한다. 예를 들자면 건물을 짓기 위한 총체적인 설계도와 사용자의 요구사항을 반영한 추상적인 개념을 말한다.
- 건축 공학의 전유물이 아니고 최근에는 소프트웨어 공학(Software Engineering)에서 많이 사용된다. 즉 기업에서 필요한 거대한 정보시스템 모델을 만들어 나가는 것이 모델링이다.

(2) Modeling

- 모델링이란 이러한 정보시스템 설계 모델을 실제 현실 세계에 맞게 구현하는 작업을 말한다.
- 모델이 필요한 이유는 미리 모델을 만들어 보면서 앞으로 개발하려는 시스템을 좀 더 구체적이고도 명확하게 인식할 수 있기 때문이다.



1) Modeling의 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.

2) Modeling의 원리

- 생성할 모델을 선택하는데 따라 문제를 공략하는 방법과 해결책을 실현하는 방법에 큰 영향을 준다.
- 모든 모델은 다양한 수준의 정밀도를 표현될 수 있다.
- 가장 훌륭한 모델은 현실을 반영한다.
- 하나의 모델로는 충분치 않다. 특수한 모델은 여러 개의 모델을 선정, 그 중에 하나를 선택해 모델링을 해야 한다.

3) Modeling 과 프로그래밍의 비교

	모델링	프로그래밍
목적	구축할 시스템의 모습 정의	시스템의 실제 구현
세부 수행 활동	요구사항 정의, 분석, 설계	소스 코드 편집, 컴파일, 디버깅
결과물	모델	소스코드를 포함한 구현된 시스템
표기법	모델링 언어(UML, ERD, DFD)	프로그래밍 언어 (Java, C#)
지원도구	CASE 도구 (Rose, Together)	개발도구 (Jbuilder, Visual Studio.NET)

(3) 객체 모델링 (Object Modeling)

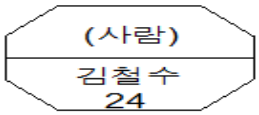
- 객체 모델링은 시스템에 요구되는 객체들을 보여줌으로써 주로 시스템의 정적인 구조를 포착하는 데 사용한다. 세 가지 모델링 중 객체지향 분석에서 가장 중요하며 선행되어야 할 모델링이며, 객체 모델링의 복잡도를 관리하기 위해 정보 모델링에서 언급된 추상화, 분류화, 일반화의 개념들이 사용된다. 또한 집단화 개념이 추가되어 다양한 객체들을 모아 더 높은 단계 객체를 추출하는 작업이 이루어진다.
- 객체 모델링은 시스템의 요구 사항을 기술한 문제 기술로부터 시스템에 요구되는 객체들을 추

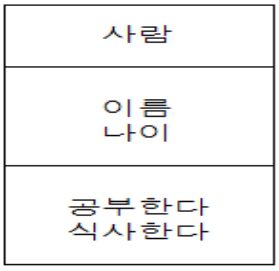
출한다.

- 객체들이 발견된 다음 객체들 사이 연관성을 찾아내고 객체들을 정의하기 위한 속성 추가한다.

1) 객체와 Class

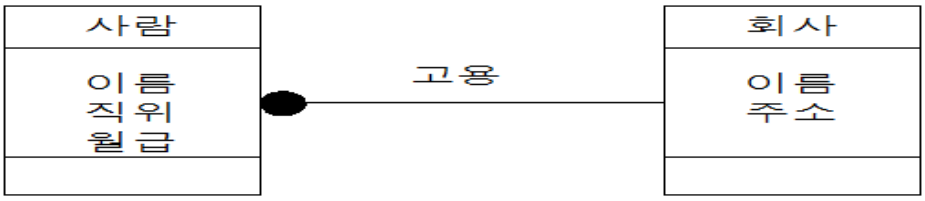
- 객체는 객체 모델링의 기본 단위이며 유사한 객체들의 모임이 클래스이다.
- 이는 앞에서 다룬 정보 모델링의 엔티티와 엔티티 타입에 해당한다.
- 객체는 구석이 다듬어진 사각형으로 표시하고 클래스는 사각형으로 표시이다.





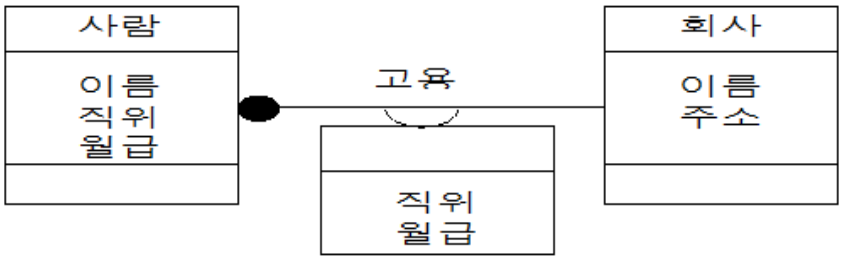
2) 클래스와 관계

- 클래스들 사이의 연관성은 관 에 의해 표시되며 정보 모델링에서는 마름모로 표시
- OMT에서는 그 관계를 단순히 관계 이름만으로 표시할 수 있다.



3) Class의 관계

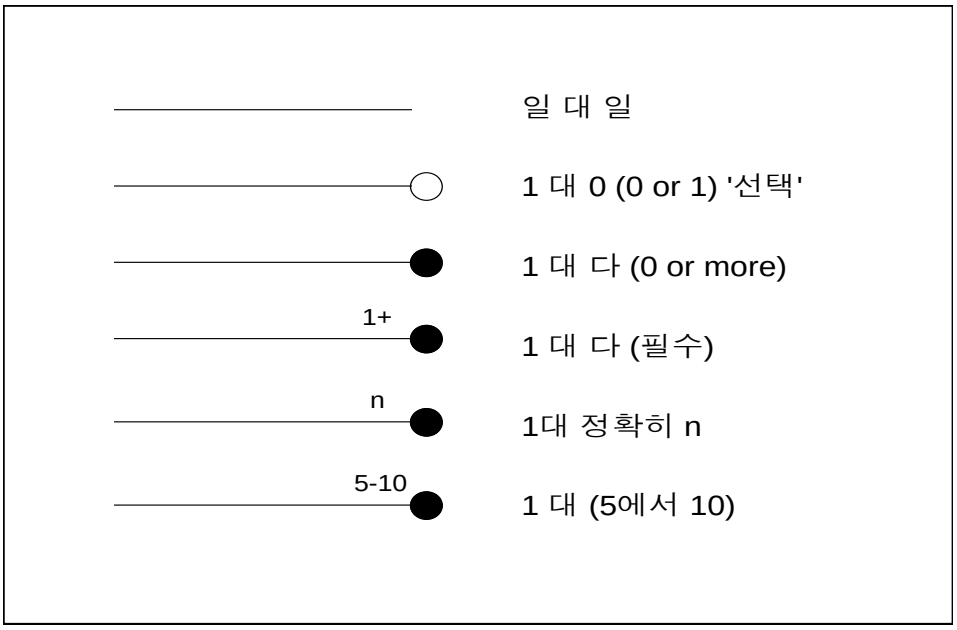
- 만약 관계가 새로운 속성이나 동작을 가질 경우 다음 그림과 같이 나타낼 수 있다.



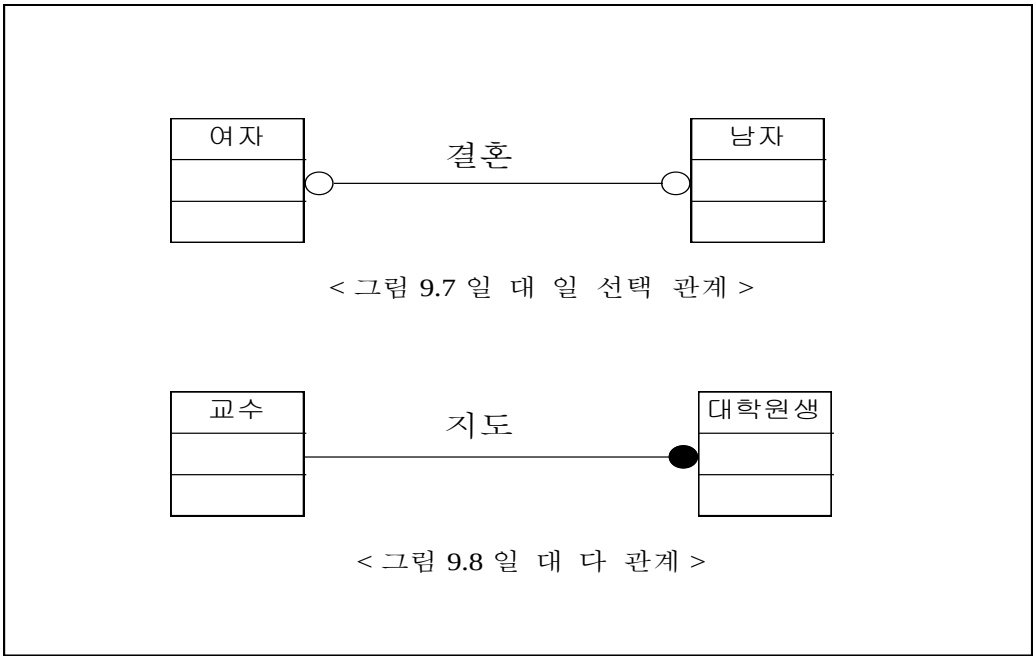
- 경우에 따라서는 관계를 한 클래스로 모델링 하는 것이 효과적인 경우가 있다.
- 이런 경우 관계는 하나의 클래스로 정의되어 속성과 동작을 가지게 되며 다른 새로운 클래스와 연관을 맺을 수 있다.

4) Mapping 계약 조건

- 기호는 클래스들 사이에 맺어질 수 있는 관계를 통해 객체들이 지켜야 하는 제약조건을 지정하는 것이다.
- 각 클래스의 객체들 사이에 맺어질 수 있는 맵핑 제약조건과 참여 제약조건은 정보 모델링에서 다루어진 개념과 동일하며 OMT에서는 다음과 같은 기호로 표시한다.



5) Mapping 관계



< 그림 9.7 일 대 일 선택 관계 >

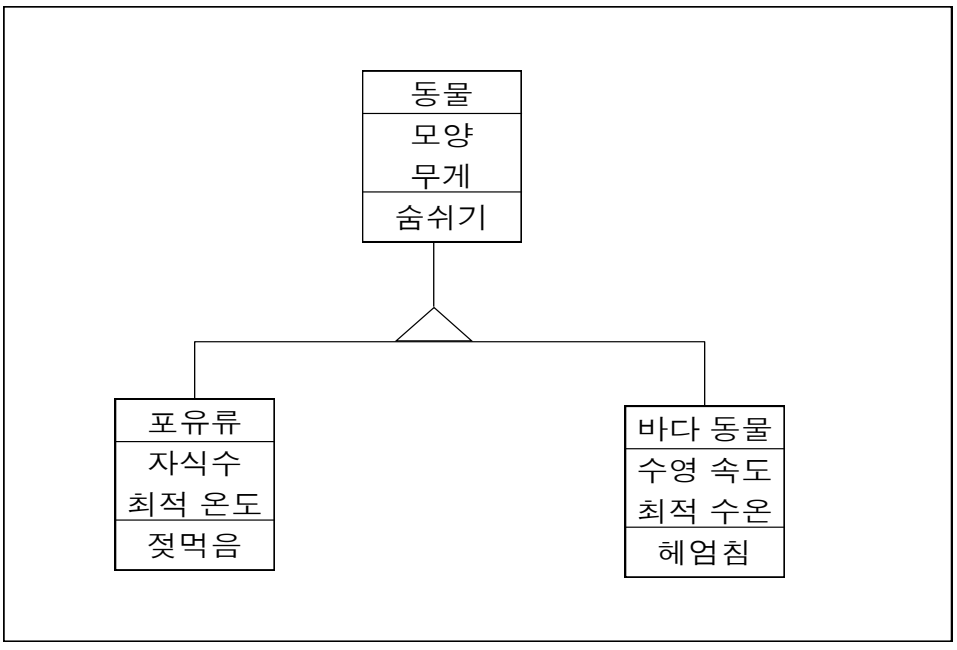
< 그림 9.8 일 대 다 관계 >

6) 일반화

- 일반화는 유사한 클래스들 사이의 공유되는 속성과 동작을 묶어주 며, 한편 그들 사이의 다른 점을 보존할 수 있게 하여주는 효과적인 추상화 기법이다.
- 일반화는 클래스들 사이의 특수한 관계로 볼 수 있으며 앞의 EER 모델에서도 이 개념이 소개된 바 있다.
- 일반화를 통해 클래스들 사이의 계층 구조가 만들어지며 각 하위 클래스는 상위 클래스의 속성, 동작, 그리고 다른 클래스와의 연관성을 상속받는다.
- 일반화를 통해 공통의 정보는 오직 한 번 정의될 수 있어 분석의 결과를 재사용할 수 있게 하여 주며 데이터의 무결성을 향상시켜 준다.
- 특수화는 앞의 정보 모델링에서도 언급되었듯이 일반화의 역작용이다.
- 특수화를 통해 하위 클래스로 정의되는 경우는 하위 클래스가 고유의 속성이나 동작을 가지고 있거나 하위 클래스가 다른 클래스들과 고유의 관계를 가지고 있을 때이다.

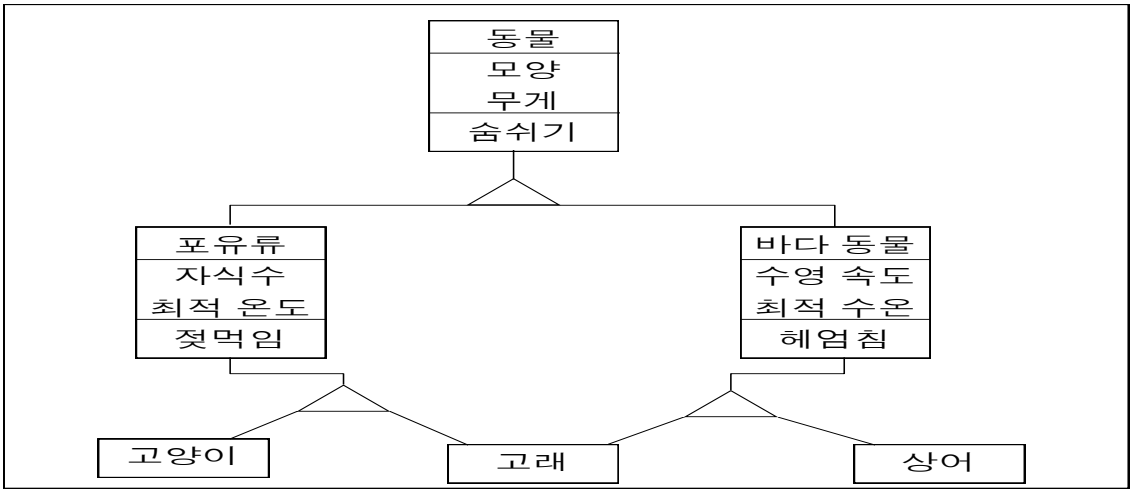
7) Class의 일반화

- 일반화는 'is_a' 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.



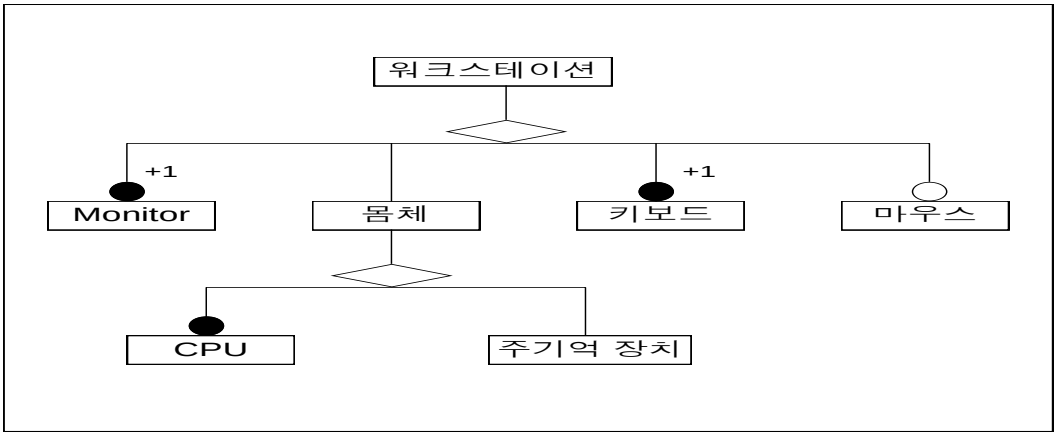
8) 다중 상속

- 만약 어떤 클래스의 상위 클래스가 두 개 있는 경우 문제가 발생할 수 있다.
- 하위 클래스는 2 이상의 상위 클래스에서 동시에 속성과 동작을 상속받는다.
- 이러한 현상을 다중 상속이라 한다.
- 다중 상속으로 인하여 분석 과정에서 모순이 발견될 수 있다.
- 이러한 모순이 속성이나 동작에서 감지되면 모호성을 없애기 위해 요구사항 명세서나 자료 사전에 이를 기록해야 한다.



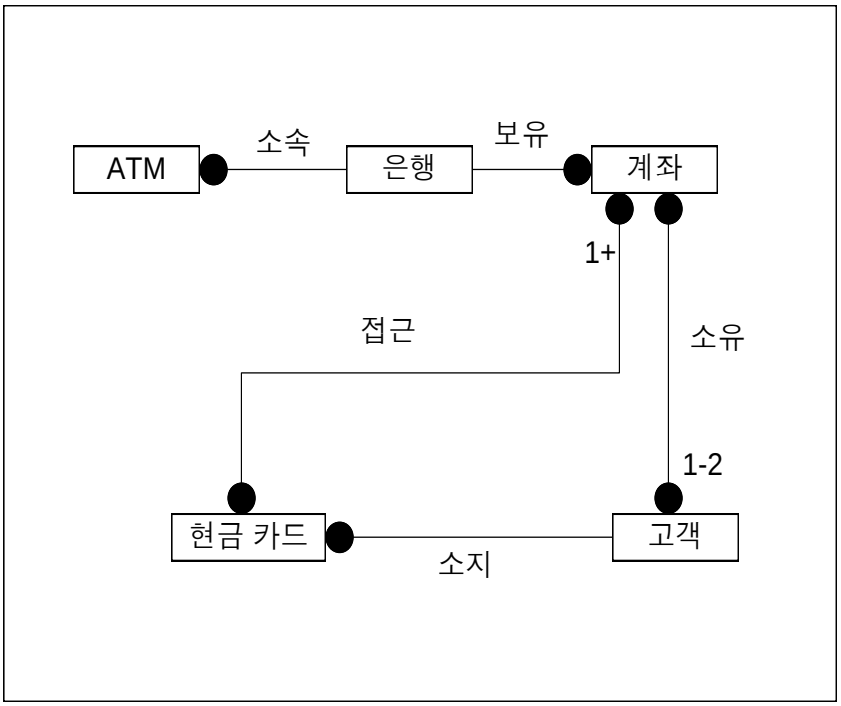
9) 집단화

- 집단화는 클래스들 사이의 "부분-전체" 관계, "부분" 관계로 설명되는 연관성을 나타낸다.
- 집단화는 여러 부속 객체들이 조립되어 하나의 객체가 구성되는 것을 의미
- 집단화는 앞의 EER 모델에서는 구체적으로 나타내지 않던 개념이며 객체지향 기법에서는 활발히 사용되고 있다.
- 집단화 기호는 조그만 마름모 기호로 표시되며 맵핑 제약조건과 참여 제약조건에 대한 기호는 일반화의 경우와 동일하다.



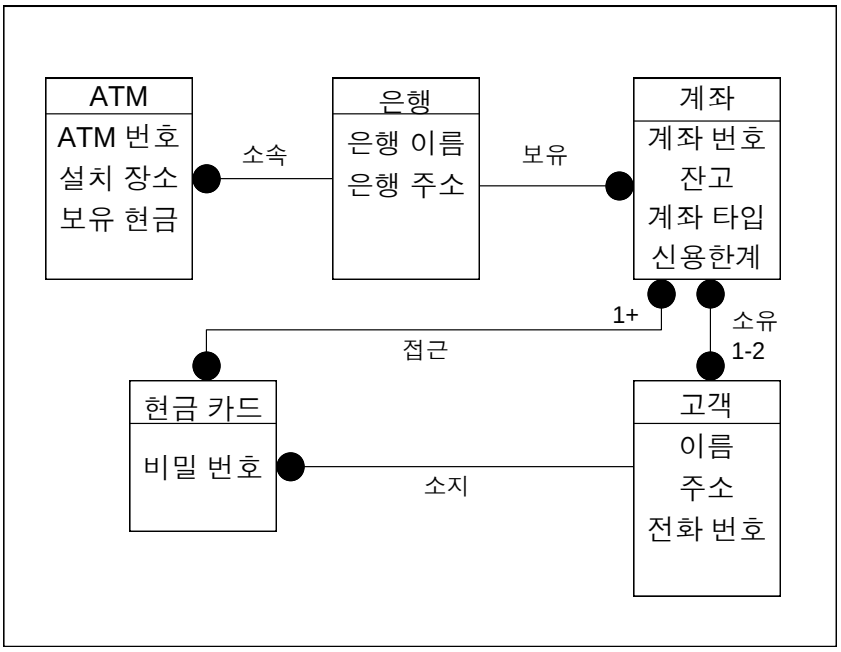
10) 객체들 사이의 연관성

- 구체적인 요구 사항 규명



11) 클래스의 속성

- 클래스와 클래스들 사이 연관성이 밝혀진 다음 클래스의 특징을 밝혀주는 속성과 동작들이 밝혀져야 한다.
- 객체 모델링에서는 속성에 우선 초점을 맞추게 된다.



(4) 동적 모델링 (Dynamic Modeling)

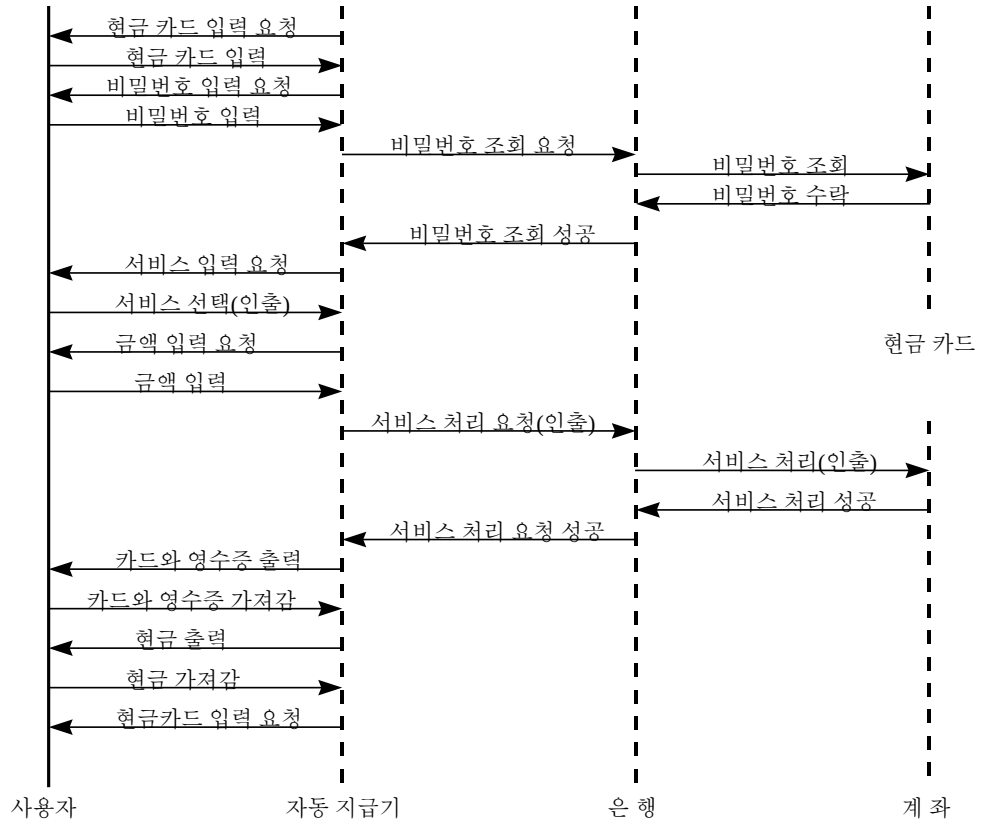
- 앞에서 다룬 객체 모델링을 통해 시스템에 요구되는 객체들의 구조와 객체들 사이의 관계 정립한다.
- 그 다음 시간의 흐름에 따른 객체들과 객체들 사이의 변화를 조사 하게 되며 이를 동적 모델링이라 한다.
- 동적 모델링은 객체들 사이의 제어 흐름, 상호 작용, 동작의 순서 등을 다룬다.
- 동적 모델의 중요한 개념은 상태, 사건, 동작 등 객체지향 분석의 특징은 시스템에 요구되는 객체를 우선 구한 후 객체들의 동적인 면을 찾아내기 위해 동적 모델링을 적용한다.
- 동적 모델은 다음에 나올 기능 모델과 관계가 있으며 시스템의 동작은 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 표시한다.
- 각 사건들은 객체 모델링에서 규명된 객체들 사이의 정보 흐름을 나타내게 되고 각 사건의 정보를 보내는 객체와 정보를 받는 객체가 밝혀진다.
- 사건의 순서와 사건을 주고받는 객체들이 사건 추적도에 나타나게 되며 수직선은 객체를 표시하고 수평 화살표는 사건의 흐름을 나타낸다.
- 사건은 위에서 아래로의 순서에 의해 수행한다.

1) 시나리오

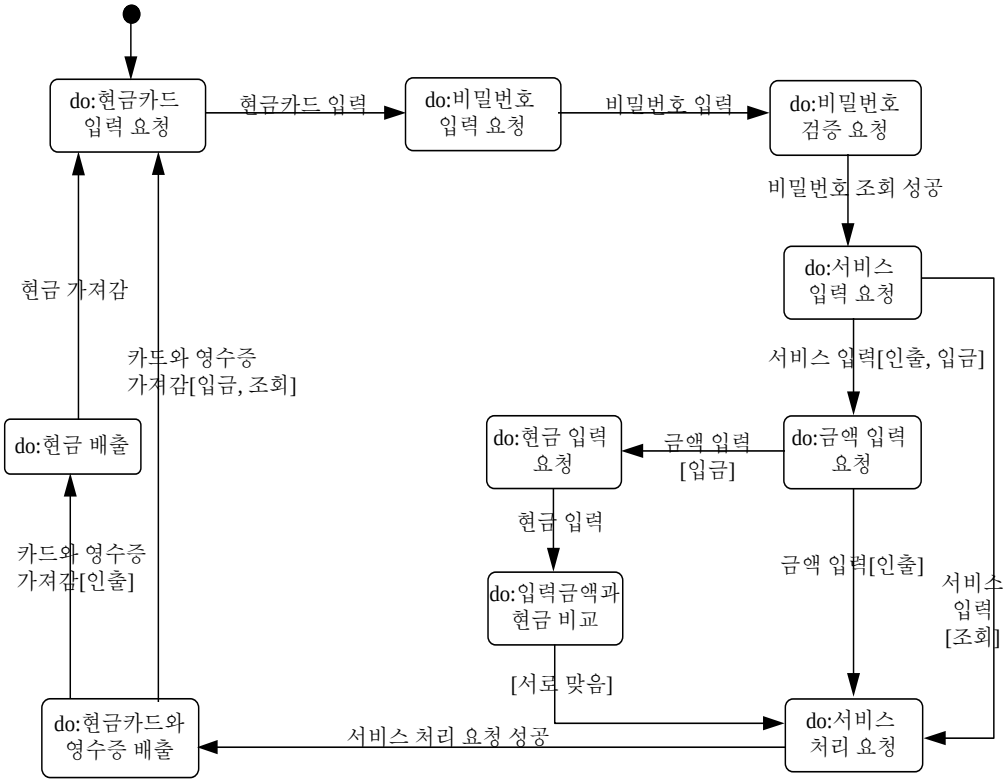
- 자동 지급기가 현금 카드를 입력할 것을 요구한다.
- 사용자가 현금 카드를 자동 지급기의 카드 입구에 넣는다.
- 자동 지급기는 현금 카드로부터 계좌 번호와 카드 번호를 읽고 사용자에게
- 비밀 번호를 요구한다.
- 사용자가 비밀번호를 입력한다.
- 자동 지급기는 현금 카드 소속 은행에게 비밀 번호 대조를 요청한다.
- 은행은 현금 카드에게 비밀 번호 대조를 요청한다.
- 현금 카드는 은행에게 비밀 번호가 일치함을 알린다.
- 은행은 자동 지급기에게 비밀 번호가 일치함을 알린다.
- 자동 지급기는 사용자에게 가능한 서비스를 보여준다.
- 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 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.

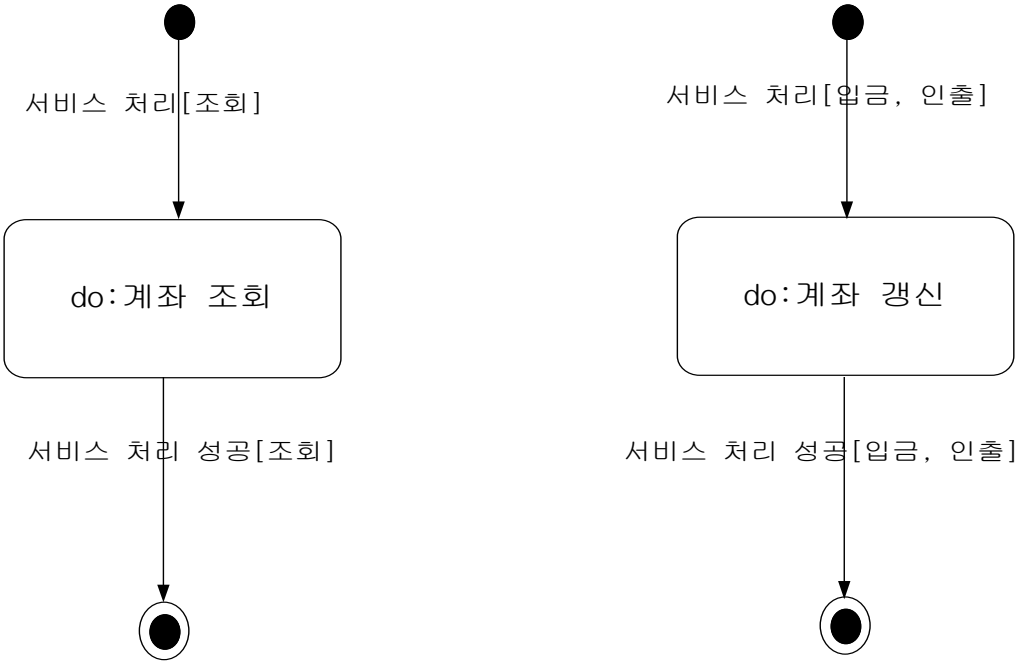
2) 사전 추적도



3) 상태변화도

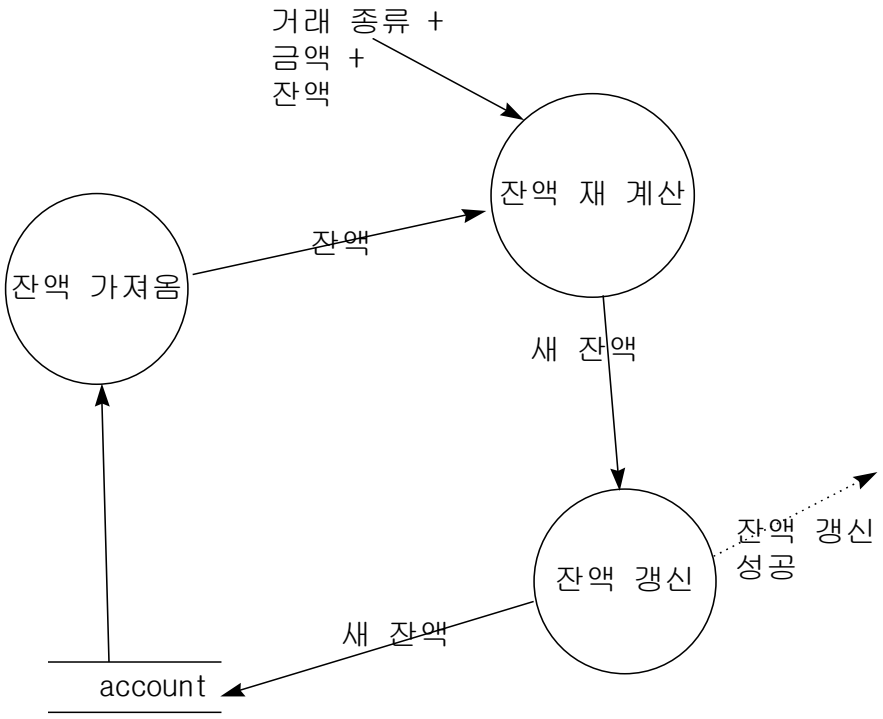

do:현금카드
입력 요청
현금카드 입력
do:비밀번호
입력 요청
비밀번호 입력
do:비밀번호
검증 요청
비밀번호 조회 성공
do:서비스
입력 요청
서비스 입력[인출, 입금]
do:금액 입력
요청
금액 입력
[입금]
do:현금 입력
요청
현금 입력
do:입력금액과
현금 비교
[서로 맞음]
금액 입력[인출]
서비스
입력
[조회]
do:서비스
처리 요청
서비스 처리 요청 성공
do:현금카드와
영수증 배출
카드와 영수증
가져감[인출]
do:현금 배출
현금 가져감
카드와 영수증
가져감[입금, 조회]

4) 계좌의 STD


서비스 처리[조회]
do:계좌 조회
서비스 처리 성공[조회]
서비스 처리[입금, 인출]
do:계좌 갱신
서비스 처리 성공[입금, 인출]

4) 기능 모델링 (Functional Modeling)

- 기능 모델링은 객체지향 분석기법에서 객체 모델링, 동적 모델링에 이어 시스템을 기술하는 세 번째 단계이다.
- 기능 모델은 입력값으로 부터 계산을 거쳐 어떤 결과가 나타나는지를 보여주며 이것이 어떻게 (how to) 유도되었는지의 구현 방법은 고려하지 않는다.
- 기능 모델은 앞에서 다룬 기능 모델링과 마찬가지로 자료흐름도(DFD)를 사용할 수 있다.
- DFD의 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 변경 동작의 기능 : 다음 그림은 계좌 갱신 상태의 DFD이다.



5) 객체지향 설계

- 만약 프로세스가 입력 흐름을 사용하고 입력 흐름을 고쳐 새로운 결과를 만들어내는 경우, 입출력 흐름은 같은 객체이며 입력 흐름이 대상 객체가 된다.
- 프로세스가 여러 입력 자료 흐름으로부터 하나의 출력 값을 생성한다면 이 프로세스와 연관된 동작은 출력 클래스에 적용되는 동작으로 해석할 수 있다.

- 프로세스가 자료저장소나 외부 객체의 데이터를 읽거나 결과를 저장하는 경우, 자료저장소나 외부 객체가 이 오퍼레이션의 대상 객체가 된다.
- 이러한 가이드라인을 이용하여 객체 모델에 동적 모델과 기능 모델을 통합한다.
- 이 결과는 객체에 속한 모든 가능한 정보, 객체들 사이의 관계, 객체의 오퍼레이션들을 나타낼 수 있게 되어 시스템에 대한 완벽한 기술이 이루어진다.
- 객체들 사이의 관계는 상대 객체를 나타내는 포인터 변수를 객체의 속성으로 나타냄으로써 이루어질 수 있다.

6) 모델의 통합
- 분석 단계를 통하여 앞에서 논의된 세 가지의 서로 다른 모델이 순서대로 개발
- 세 모델을 하나로 통합하는 작업이 이루어져야 하며, 이 단계는 일반적으로 객체지향 설계단계에 해당된다.
- 세 모델의 통합은 객체의 정적인 구조와 오퍼레이션을 함께 포함하여 객체를 정의하는 것을 의미한다.
- 이는 동적 모델의 사건, 동작 및 활동을 객체의 오퍼레이션에 맵핑 하고, 기능 모델의 프로세스를 객체 모델의 오퍼레이션에 통합시키는 것이다.
- 객체 수준의 상태 변화도는 한 객체가 생명주기 동안 가질 수 있는 상태들을 기술하여 준다. 상태의 변환은 객체의 오퍼레이션으로 맵핑한다.
- 유사한 방법으로 객체에 주어진 사건은 다른 객체의 동작으로 나타낼 수 있다.
- 한 사건은 이전 사건에 의하여 초기화된 활동의 결과이며, 동작과 이에 반응하는 동작으로 맵핑 될 수 있다.

2. UML을 이용한 Modeling

(1) UML이란?

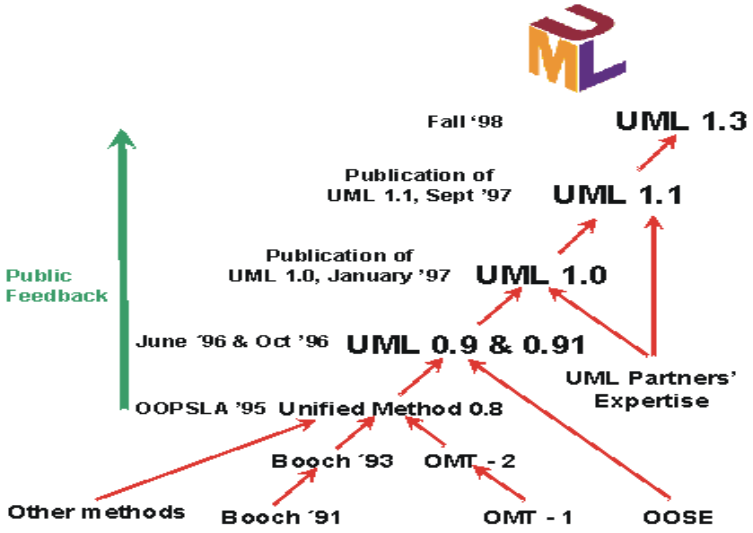
- UML은 3Amigos(Grady Booch, Ivar Jacobson, James Raumbauch)를 중심으로 만든 모델링 방법이다.
  - ✓ Booch : Booch/ 특히 설계, 구축단계에서 표현력이 좋다.
  - ✓ Jacobson : OOSE/요구사항 수집, 분석, 상위수준의 설계에 대한 강점이 있으며 Use Case에 대한 지원이 좋다.
  - ✓ Raumbaugh : OMT-2/ 분석과 데이터 중심의 시스템을 설계하는데 유용하다.
- 객체지향 설계를 하는데 있어서 효율적인 방법을 제공한다. 객체를 추출하고 설계하는 방법을 제

공 함으로서 보다 쉽고 빠르게 작업을 마칠 수 있게 한다. 최근에는 대규모 프로젝트를 추진하는 데 있어서 가장 중요한 분야로 등장하였고, 그 중요성은 갈수록 증가하고 있다. 실제 UML은 최근에 각종 개발 툴 개발업체들이 지속적으로 업그레이드 하고 있고 새로운 기능들을 추가하고 있다. 그 대표적 회사로는 Rational Rose 사를 들 수 있다.

(2) UML의 장점

- 분석에서부터 설계, 구현까지 Seamless하다.
- 풍부하고 일관된 표기법을 가지고 있다.
  - ✓ 의사소통에 용이하다.
  - ✓ 어느 곳이든 맵핑이 가능하다. (언어, DB 등)
  - ✓ 프로젝트가 소규모이든 대규모이든 어느 곳에나 적용이 가능하다.
  - ✓ 재사용이 가능하다

(3) UML의 역사



- 1994년 10월 james Rumbaugh(GE 사)의 OMT(Object Modeling Technique) 개발 방법론, Ivar Jacobson(Objectory사)의 OOSE(Object-Oriented Software Engineering) 방법론, Grady Booch(National Spftware 사) 방법론 등을 하나로 통합하려는 노력에 의해 만들어 졌다.
- 1995년 10월 0.8 버전 발표 : Booch와 OMT 통합
- 1996년 7월 : UML 0.9 버전 발표 (OOSE 와 통합)
- 1997년 7월 : UML 1.0버전 OMG(Object Management Group)에 제출

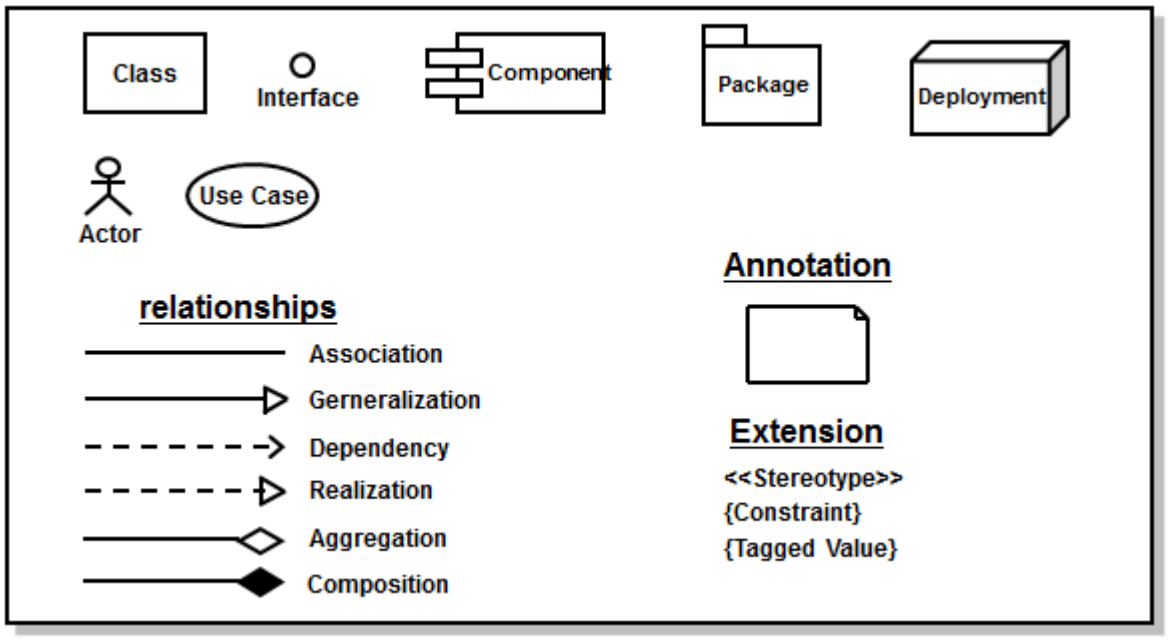
- 1997년 11월 : UML은 OMG 에 의해 표준 모델링 언어로 지정됨.

(4) UML의 특징

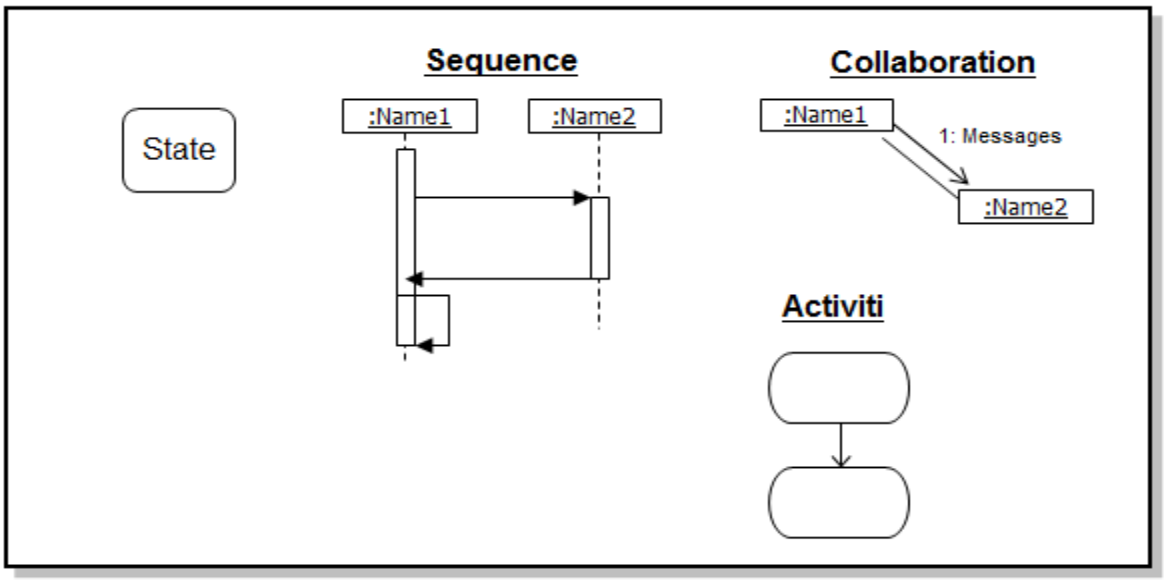
- 모델링은 시스템의 필수적인 부분을 수집한다.
- 모델링은 비즈니스 프로세스(요구사항)를 수집한다.
- 의사소통의 도구이다.
- 모델링은 복잡성을 관리한다.
- 모델링은 재사용성을 높인다.
- 모델링은 소프트웨어 아키텍쳐를 정의한다

(5) UML의 구성요소

1) Structure Element



2) Behavioral Elements



3. UML Diagram

(1) 개발 단계에 따른 UML 산출물

개별활동	산출물		
	산출물명	세부산출물명	UML 다이어그램
요구사항 정의	요구사항 모델	Use Case 모델	Use Case
		Use Case 명세서	N/A
		이벤트 흐름 모델	시퀀스
		화면 흐름 모델	액티비티
분석	분석모델	분석 객체 모델	클래스
		분석 Use Case 실현 모델	시퀀스
설계	설계모델	설계 객체 모델	클래스
		데이터 모델	
		설계 Use Case 실현 모델	시퀀스
		배치 모델	배치
		컴포넌트 모델	컴포넌트

(2) Use Case Diagram

- 시스템에 대한 요구사항을 정의하기 위하여 시스템과 상호 작용을 하는 외부 환경와 시스템이 제공해야 하는 기능을 표현할 때 사용한다.

1) Use Case란?

- 시스템의 사용에 대한 시나리오의 집합이다.

2) 목적

- 사용자의 관점에서 시스템을 모델링 하기 위한 방법으로 사용자가 시스템에 대하여 바라는 바를 표현할 수 있다.
- 사용자의 시점을 빨리 이해함으로써 쓸모있고(useful), 쓸수있는(usable) 시스템을 만들 수 있도록 한다.

3) 사용

- 대개 의뢰인과 개발팀이 참조하는 설계 문서의 한 부분으로 사용한다.
- 새로 만들어진 시스템을 테스트하는데 사용한다.
  : 사용자의 시스템 사용 시나리오를 표현한 것임으로 테스트도 그 시나리오에 따라서 하면 됨.
- 시스템 분석가와 사용자와 힘을 합쳐 시스템의 사용 방법을 결정하는데 도움
- 시스템이 가진 User Interface 설계 & 프로그래밍 방식에 대해 도움.
- [요구사항 수집]의 "시스템 요구사항 파악"의 과정과 [분석] 과정에서의 출력물
  → 사용자와의 대화를 통하여 얻어낸 정보를 표현한다.

Ex) 음료수 자동 판매기 시스템을 설계

시나리오 : "음료수 사기" << Use Case Name
1. 동전을 넣는다
2. 자판기가 종이컵에다가 음료를 따른다.

4) Use Case의 분석과정 및 방법 : 사용자를 만나 의견을 듣는 것으로 시작한다.

- 시스템 사용자를 시스템 분석과 설계의 초기 단계에 참여 시킴.
- [요구 사항 수집] 의뢰인과 대화 → 초기 클래스 다이어그램 → 사용하고 있는 용어를 통해서 개념 정리 → 사용자와의 대화가 유연해짐.

- 사용자와 대화 ➔ 설계하고자 하는 시스템을 가지고 어떤 일을 하는지 질문 ➔ 얻어낸 대답은 미래에 Use Case를 만드는 토대로 사용.
- Use Case에 간단한 설명을 붙임 ➔ 설명이 많을 수록 사용자와 할 수 있는 대화의 밀도는 진해진다.

5) Use Case의 재사용을 위한 방법

- 포함 (Inclusion) : 다른 Use Case를 구성하는 하나의 진행 과정으로 Use Case를 사용한다.
- 확장 (extension) : 기존의 Use Case에 몇 개의 단계를 추가하여 새로운 Use Case를 만든다.

6) 액터 (Actor)

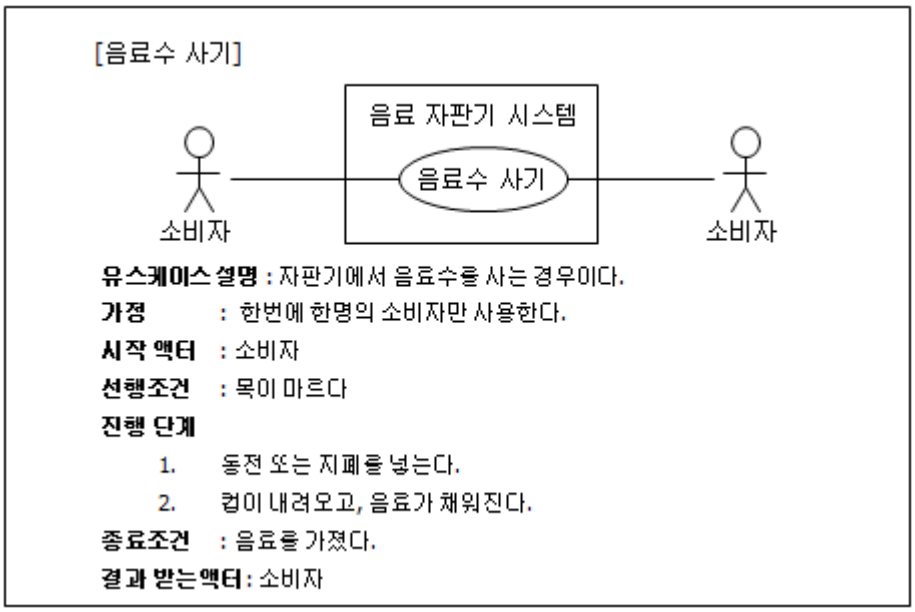
- 시스템과 교류하는 사람이나 시스템 또는 장치이다.
- Use Case를 시작시키다.
- Use Case를 구성하는 진행 단계가 끝나면 그 결과를 받는다.
- 막대기로 사람 모양을 표현한다.
- 액터의 이름은 막대인간 아래에 쓴다.

7) 유스케이스 (Use Case)

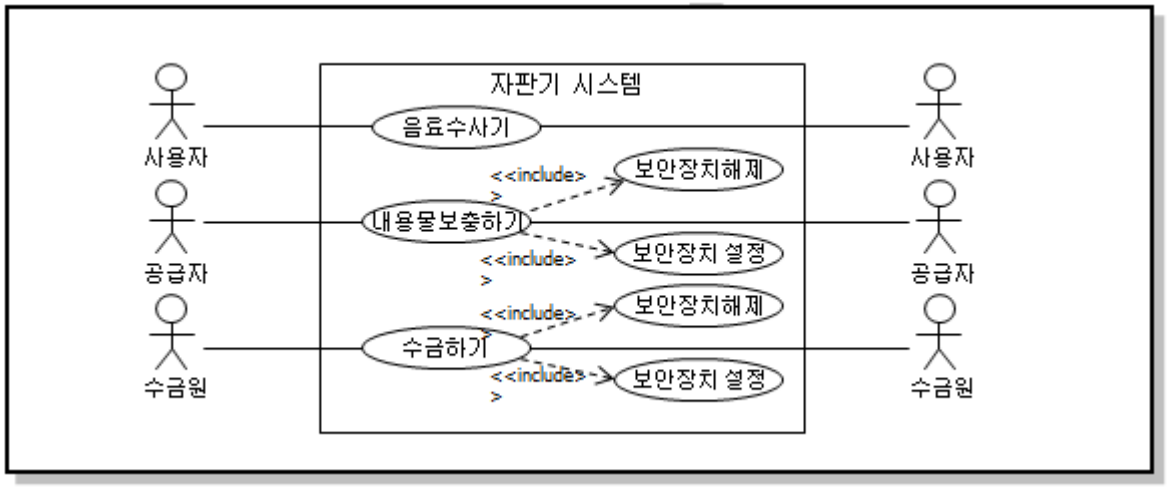
- 타원으로 표현한다.
- Use Case의 이름은 타원의 안쪽 또는 아래에 쓴다.
- Use Case 를 시작시킨 액터는 왼쪽에 위치한다.
- 결과를 받는 액터는 오른쪽에 표현한다.
- 액터와 Use Case 간의 연결은 실 선으로 그린다.

8) 시스템

- Use Case 를 둘러싸는 사각형으로 표현.
- 액터는 대개 시스템 외부에 있는 반면, Use Case는 시스템의 내부에 있다.
- 즉, Use Case 분석을 통하여 시스템과 외부세계와의 경계를 효과적으로 보여줄 수 있다.



9) 예제:: 자판기 내용물 공급자와 수금원이 포함된 자판기 시스템의 Use Case Diagram



- “내용물 보충하기” Use Case와 “수금하기” Use Case의 시나리오에서 공통으로 중복된 보안장치 해제나 보안장치 설정과 관련된 진행 단계들을 따로 떼어내서 새로운 Use Case로 만들고 포함 시켰다.

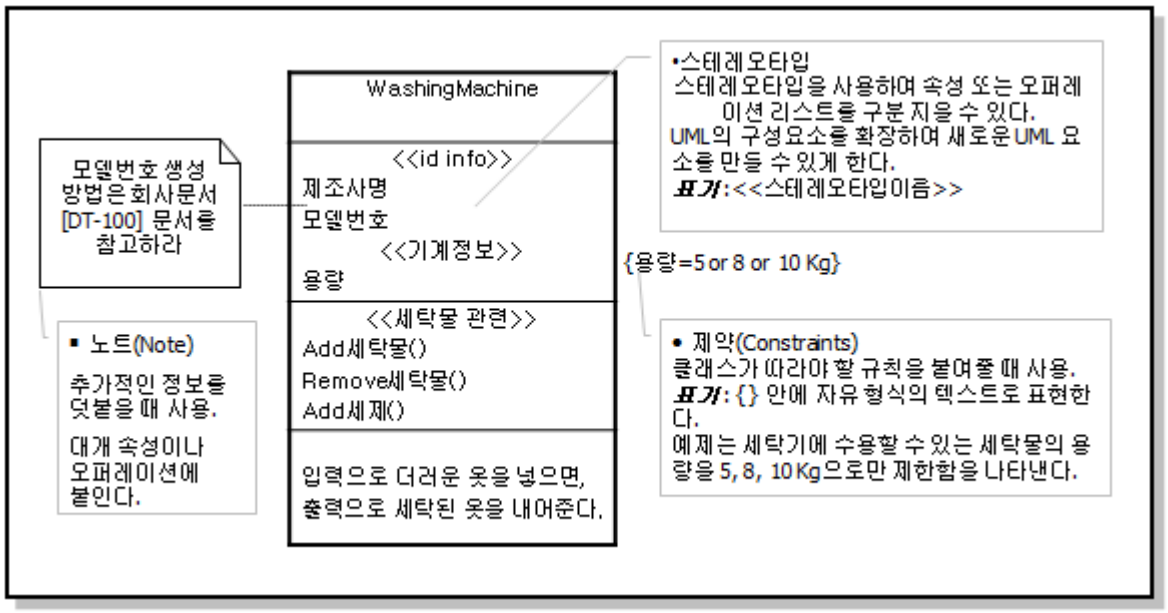
(3) Class Diagram

- 시스템을 구성하는 클래스 및 각 클래스의 속성과 연산 그리고 클래스 사이의 관계를 표현하는 데 사용된다.
- 객체지향 시스템 개발 시 가장 중요한 역할을 한다.

- 클래스는 지식 도메인에 기반한 어휘와 용어로부터 만들어진다. 시스템 분석가는 의뢰인과 상담하여 그들이 가지고 있는 지식 도메인을 파악하여 정리하고, 그 도메인에서 발생되는 문제를 해결할 컴퓨터 시스템을 설계해 나가면서, UML에 사용할 용어를 선정하고 이것을 클래스로 모델링 하는 것이다.
- 고객과의 대화 속에서 명사와 동사에 귀를 기울여야 한다. 명사는 클래스의 이름이 될 확률이 높다. 동사는 모델링한 클래스의 Operation이 될 가능성이 있다.
- 클래스의 핵심이 되는 리스트(클래스 이름, 속성, 오퍼레이션)를 모두 정리한 다음에는, 각각의 클래스가 의뢰인의 업무에서 어떤 역할을 할 지에 대하여 묻도록 하자.

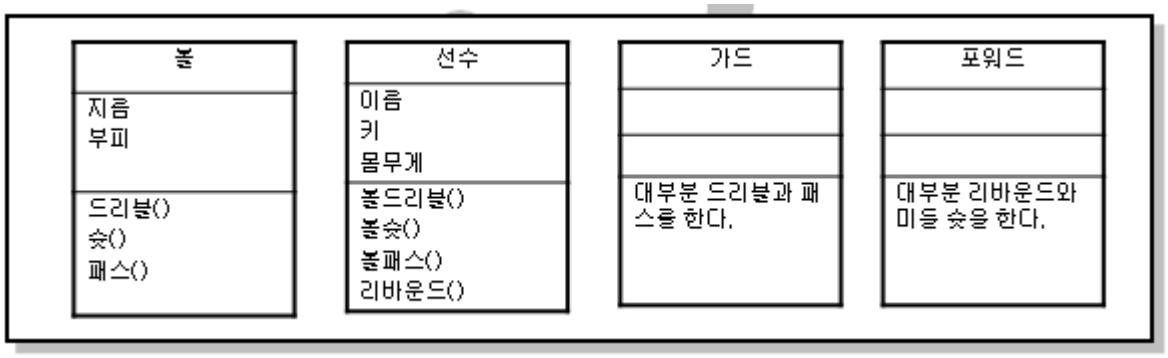
  → 이에 대한 대답은 클래스의 책임 설명으로 적을 수 있다.

1) 예제 :: 세탁기 클래스



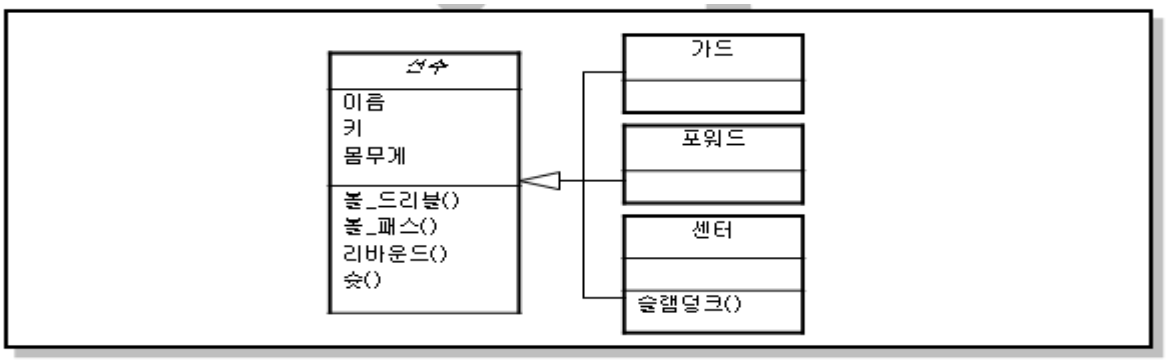
2) 예제 :: 농구 게임을 Class 모델링하기

- 명사 : 볼, 바스켓, 팀, 선수, 가드, 포워드, 센터, 슛, 슛 시간, 3점라인, 자유투, 파울, 자유투 라인, 코트, 게임시간
- 동사 : 슛하다, 드리블하다, 패스하다, 파울하다, 리바운드하다
- 이 외에도 개인의 상식을 동원해서 클래스 모델링에 사용할 수 있다.
- 다음은 이런 정보를 바탕으로 만든 Class Diagram이다. 이렇게 만든 Class Diagram은 나중에 다시 농구팀 감독과 이야기할 때 충분한 기본 자료로 사용하여 더 많은 정보를 얻어내는데 도움을 줄 것이다.



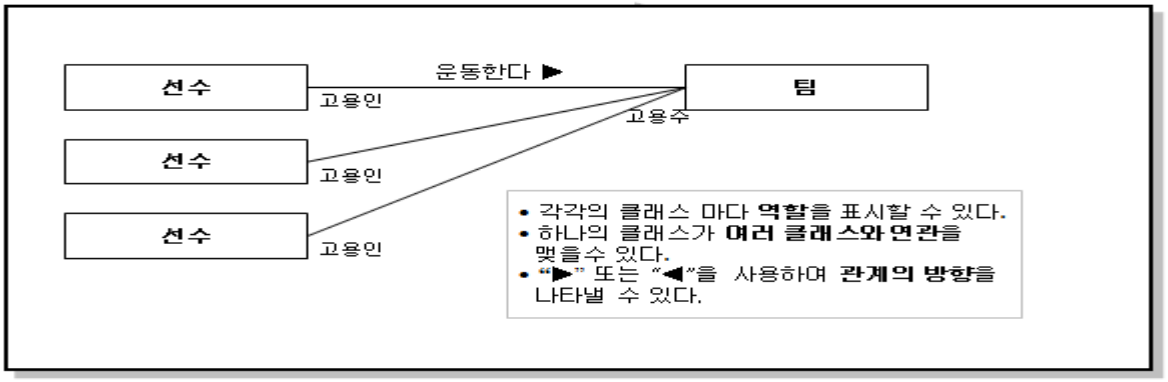
3) 추상클래스 (Abstract Class)

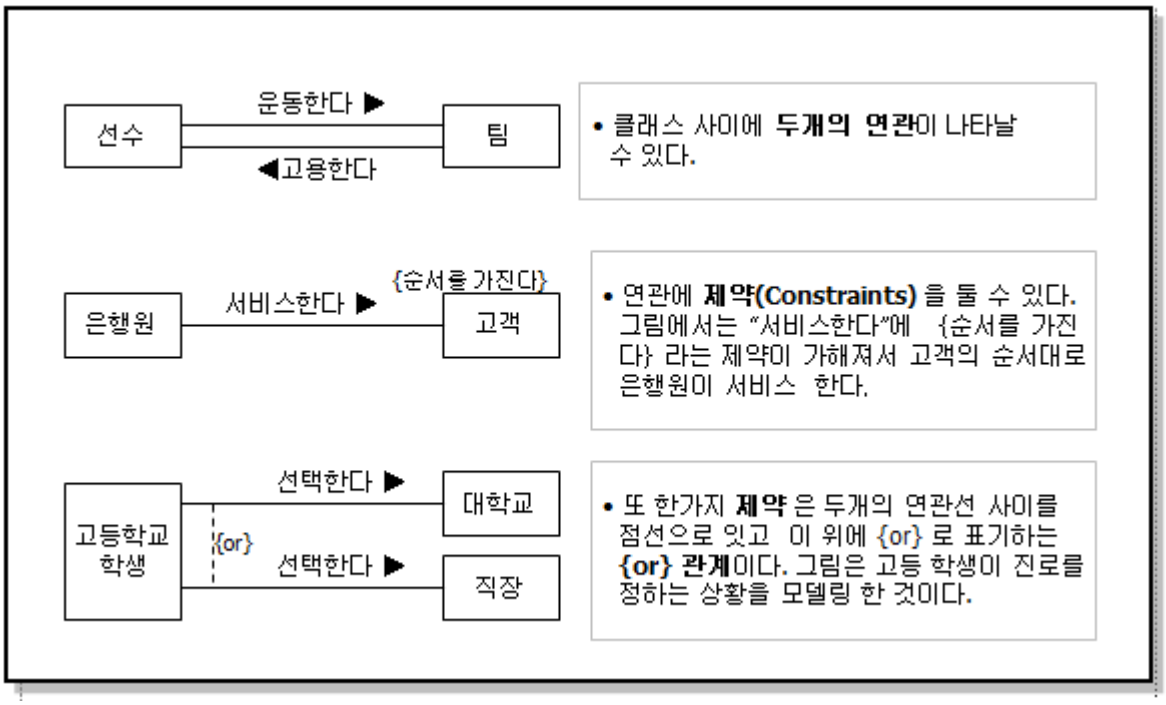
- 어떤 Sub Class의 Super Class가 있을 때, 만약 이 Super Class의 구체적인 인스턴스(Instance)를 만들 필요가 없을 때에는 “추상 클래스”로 만들자. 즉, 클래스의 객체를 생성하지 않는 클래스를 “추상 클래스”로 만든다.
- 표기 : 클래스명을 이탤릭으로 쓴다.



4) 연관 (Association)

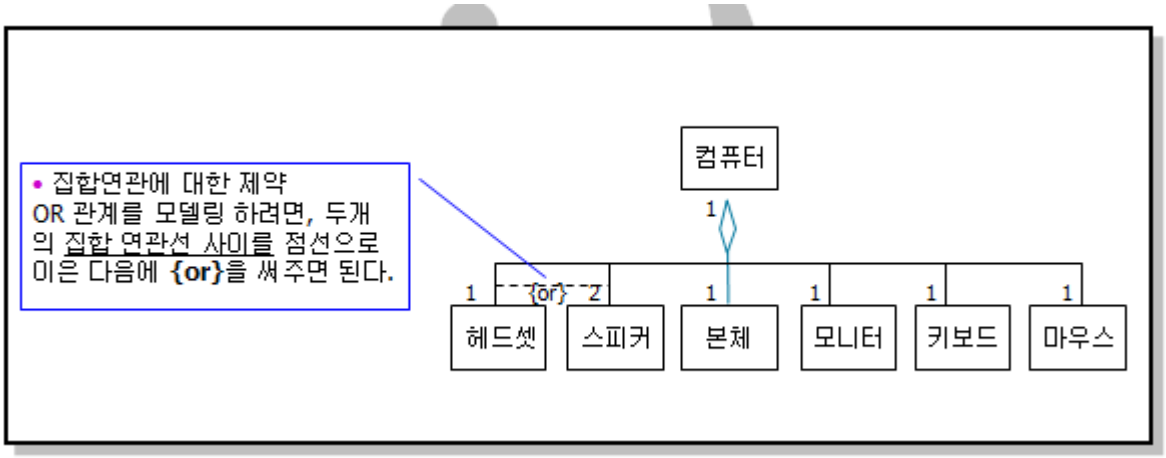
- 클래스가 개념적으로 서로 연결되어 있음을 말한다.
- Ex) 선수와 농구팀의 관계





5) 집합연관 (Aggregation)

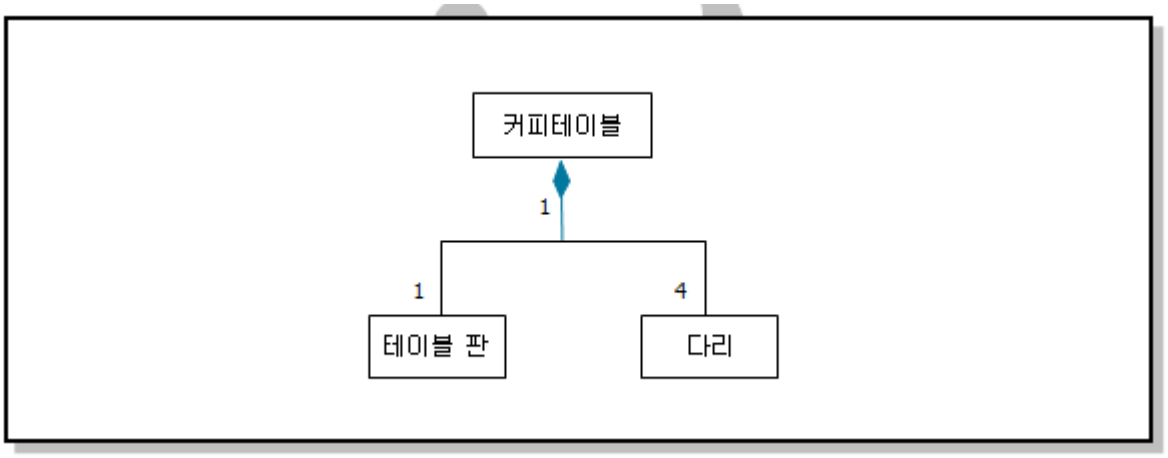
- 하나의 클래스가 여러 개의 컴포넌트 클래스로 구성되어 있는 경우가 있다. 즉, 컴포넌트 클래스와 전체 클래스가 "부분-전체" 연관 관계를 가질 때 집합연관이 된다.
- 표기: 컴포넌트 클래스와 전체 클래스를 선으로 잇고, 빈 마름모꼴을 전체 클래스 쪽에 붙여서 나타낸다.



6) 복잡연관 (Composition)

- 강한 집합 연관으로써 각 컴포넌트 클래스가 오직 하나의 전체 클래스에서만 의미를 가질 때, 복합연관으로 표현한다.

- 표기 : 각각의 컴포넌트는 전체 클래스 쪽으로 향하여 안이 채워진 마름모 꼴의 화살표를 연결한다.



7) 다중성

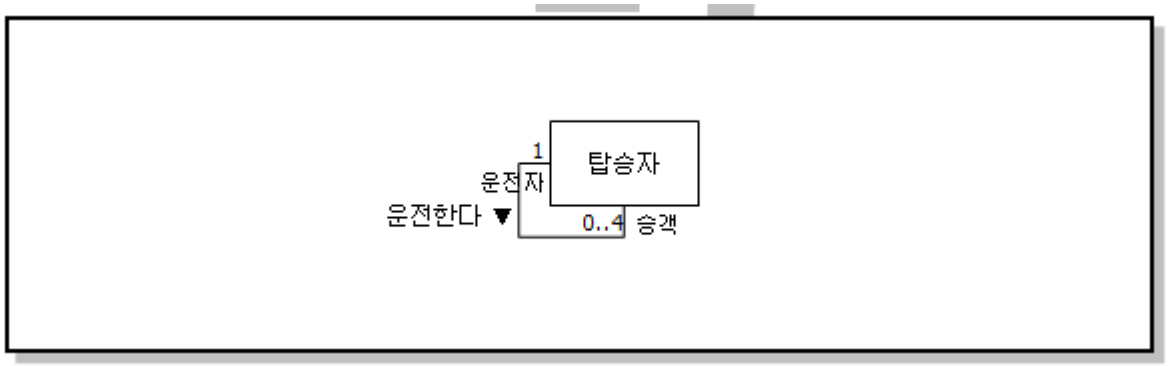
- 연관되어 있는 두 클래스 사이에서 한 클래스의 객체와 관계를 가질수 있는 다른 클래스의 객체 개수. 이것을 다이어그램에서 나타내면 해당 클래스 가까이(그리고 연관선 위)에 객체의 수를 써 준다.
- 예를 들면, 팀의 입장에서 보면 다섯명의 선수와 연관되어 있지만 선수의 입장에서 보면 한 팀과 연관되어 있다는 것이다.
- 표기 : “more” 와 “many”를 표현하는 기호로서 ‘*’ 를 사용한다.

1..* → 1또는 그이상 (one or more)
2..7 → 2이상 7까지 (2 through 7)
5,7 → 5 또는 7 (5 or 7)

8) 반사연관 (Reflexive)

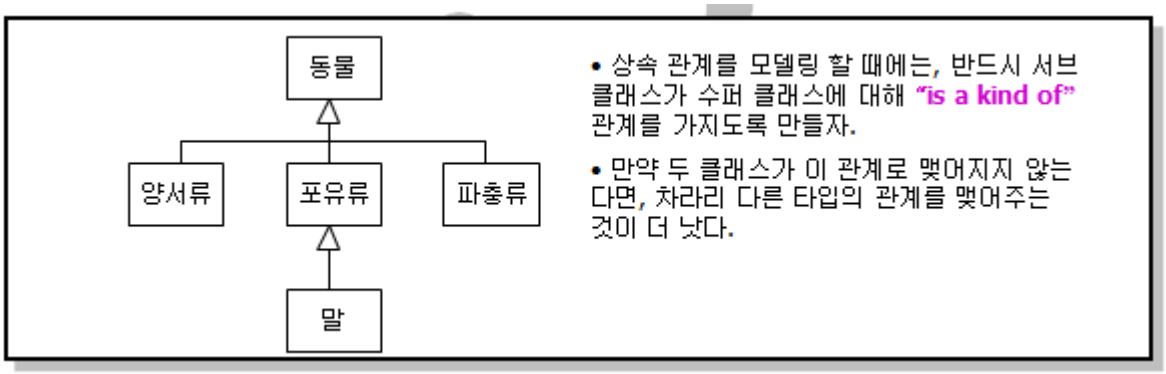
- 클래스는 자기 자신과 연관을 가질 수도 있다. 하나의 클래스가 여러가지의 역할을 가질 때 “재귀연관”을 갖도록 한다.
- 예를 들면, 탑승자(CarOccupant)는 운전자(Driver)도 될 수 있고 승객(Passenger)도 될 수 있다.

Ex) 운전자의 역할을 맡은 탑승자는 승객의 역할을 맡은 탑승자를 0명이상 4명까지 실어 나를 수 있다. UML은 “more” 와 “many”를 표현하는 기호로서 ‘*’ 를 사용한다



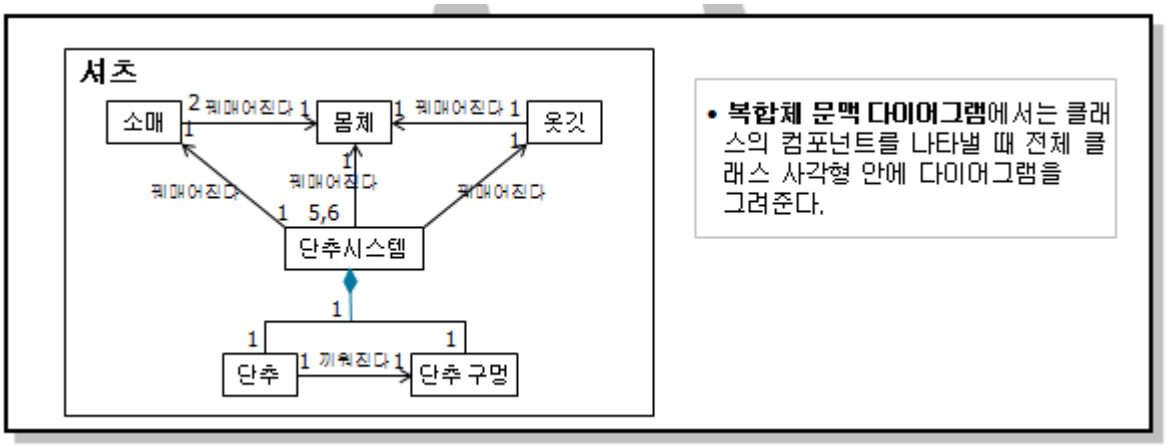
9) 상속, 일반화

- 한 클래스는 다른 클래스로부터 속성과 오퍼레이션을 물려 받을 수 있다. 이것을 객체지향 개념에서는 “상속” 이라 하고, UML에서는 “일반화” 라 한다.
- 상속 관계에서 상속을 받는 쪽을 Child Class 또는 Sub Class 라고 하고, 상속을 해주는 쪽을 Parent Class 또는 Super Class 라고 한다.
- Sub Class 에서 Super Class 쪽으로 속이 빈 화살표(→)를 연결한다. 이러한 타입의 연결 관계를 “…의 일종(is a kind of)” 이라고 부른다.



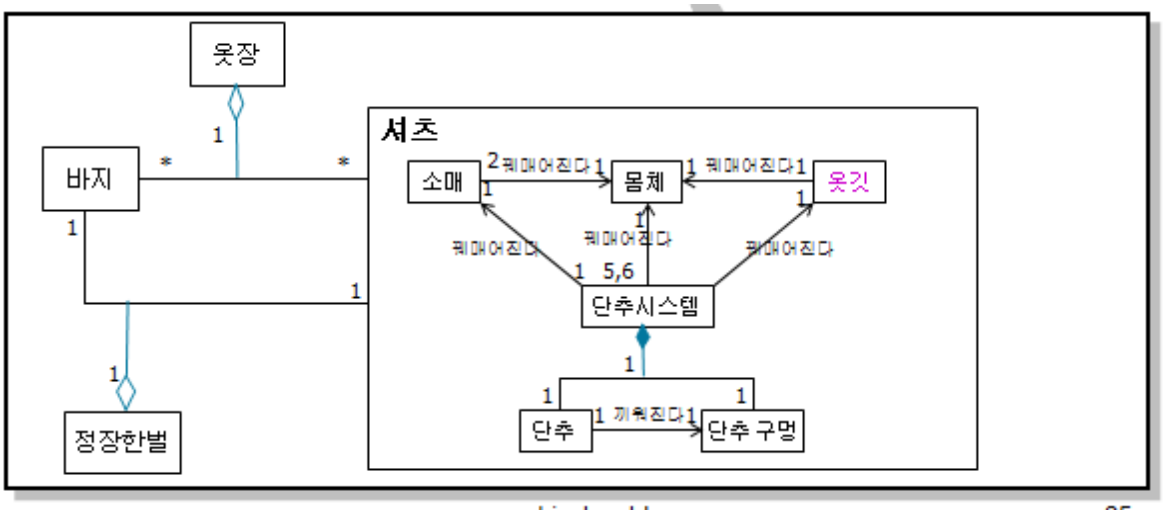
10) 문맥 (Context) – 복합체

- 문맥 다이어그램은 큰 지도의 일부를 확대해서 그린 것이다. 상세한 정보를 더 써주어야 할 필요도 있기 때문이다.
- 예를 들면, “셔츠” 클래스가 있다면 이 클래스를 구성하는 컴포넌트 클래스는 무엇이 있고, 이들 사이의 연관 관계는 어떻게 이루어져 있는지를 표현한다.



11) 문맥(Context) – 시스템

- 시스템 문맥 다이어그램은 한 클래스의 컴포넌트와 이 클래스가 시스템내의 다른 클래스와 어떤 관련을 가지고 있는지를 보여준다.
- 여기서 “옷장”, “바지”, “정장한벌” 클래스도 “셔츠” 클래스 처럼 상세한 정보를 가진 문맥 다이어그램으로 표현될 수 있다



12) 구조가 좋은 Class Diagram

- System의 정적 설계 View의 한 관점을 전달하는데 초점을 맞춘다.
- 해당 관점을 이해하는데 필수적인 요소들만 표현한다.
- 추상화 계층에 알맞은 정도의 상세 내용을 제공하고 이해하는데 필수적인 장식만을 사용한다.
- 중요한 의미를 이해할 수 있을 정도로 복잡하게 설계한다.

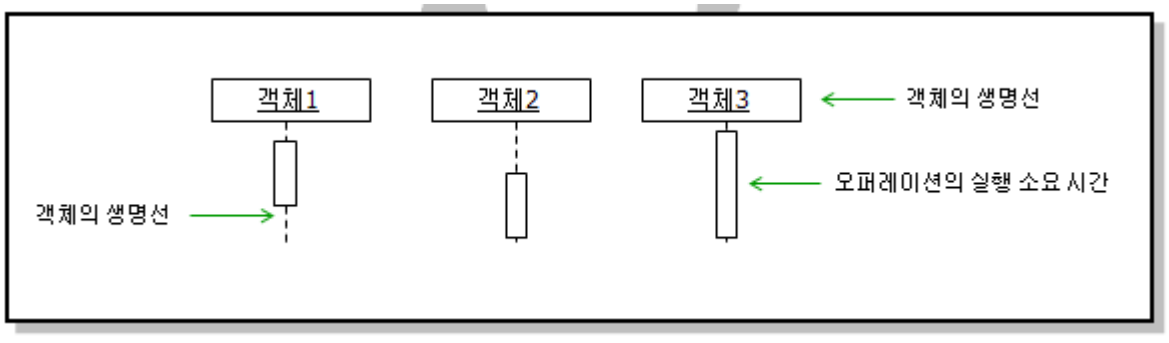
13) Class Diagram 작성 규칙

- 목적에 맞는 명칭 사용한다.
- 서로 교차하는 선(관계 표현)을 최소화 하도록 배치한다.
- 의미적으로 관련 있는 Class들을 가까운 위치에 배치한다.
- 시각적 암시를 활용 (Note 또는 색깔)한다.
- 너무 많은 관계를 나타내지 않도록 도해한다.

(4) Sequence Diagram

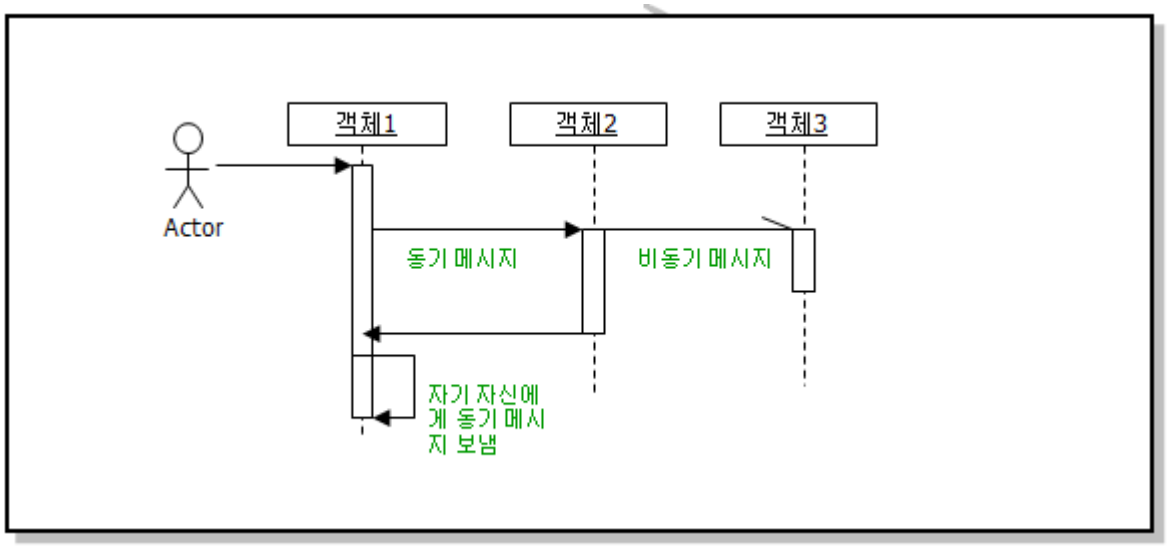
1) 객체

- 시퀀스 다이어그램의 가장 위 부분에 위치, 왼쪽에서 오른쪽으로 배열한다.
- 배열순서 - 다이어그램을 간략하게 하는 방향으로 기준을 삼는다
- 각 객체로부터 아래로 뻗어 가는 점선은 객체의 생명선(lifeline)이라 불린다.
- 생명선을 따라 좁다란 사각형이 나타나는데, 이 부분은 실행(activation)이라 한다.
- 즉, 객체가 수행되고 있음을 나타낸다. 사각형의 길이는 오퍼레이션의 실행 소요 시간을 나타낸다.



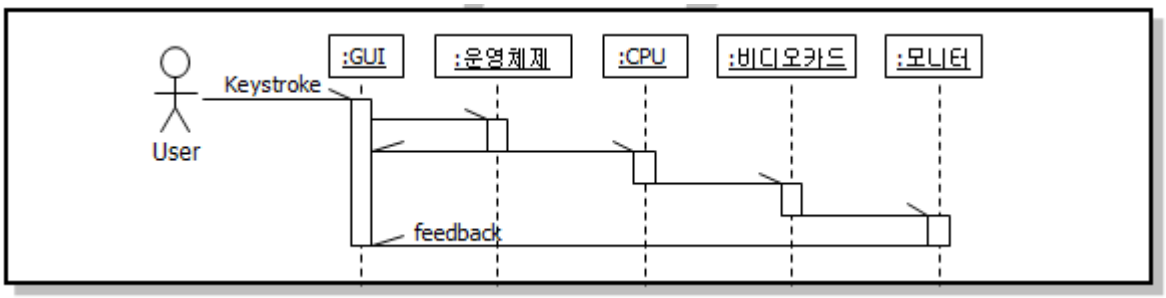
2) 시간

- 시간을 수직 방향으로 표현한다.
- 시간의 흐름은 위에서 아래로 흐른다.

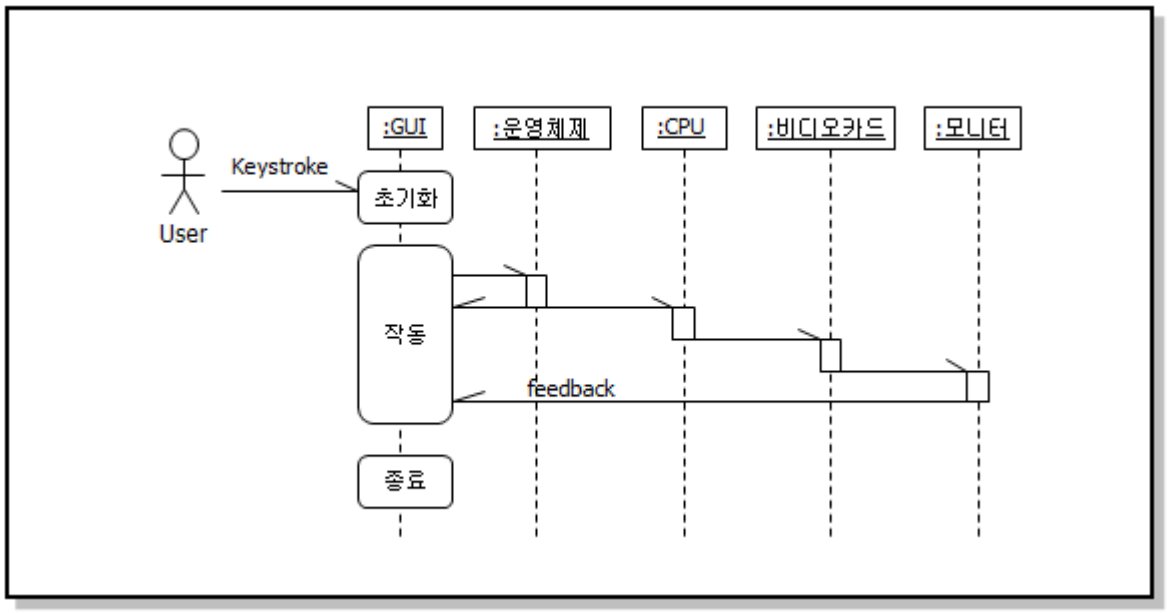


3) 예제 :: State Diagram의 GUI 시스템 예에서 GUI 가 다른 객체들과 어떻게 교류를 하고 시간에 따라 어떻게 작동하는지를 알아보자.

- 1: GUI는 키 입력을 운영체제에게 알린다.
- 2: 운영체제는 CPU에게 그 사실을 알린다.
- 3: 운영체제는 GUI를 갱신한다.
- 4: CPU는 비디오 카드에게 GUI 갱신에 필요한 명령을 내린다.
- 5: 비디오 카드는 모니터로 메시지를 전송한다.
- 6: 모니터는 화면에 Alpha Numeric 문자를 표시하고, 사용자에게 피드백을 제공한다.



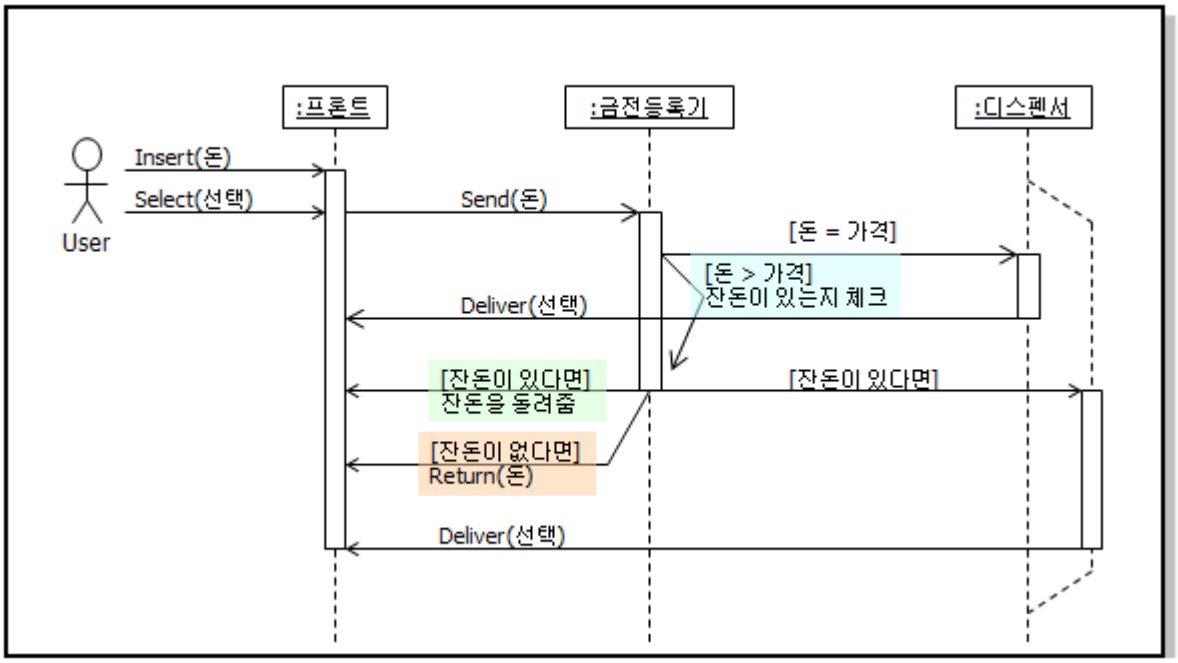
4) 예제 :: State Diagram에서 본 GUI 객체 State Diagram을 포함한 Sequence Diagram



5) 예제 :: "음료수 사기" Use Case 에서 우리는 또 다른 시나리오를 가정할 수 있다. 선택된 음료수가 다 떨어진 경우 또는 소비자가 넣은 돈이 음료수 값과 맞지 않을 때. 이런 모든 경우를 하나의 시퀀스 다이어그램에 표현하고자 할 때 일반 시퀀스 다이어그램을 그리면 된다. 다음은 '액수에 맞지 않는 경우' 의 시나리오 이다.

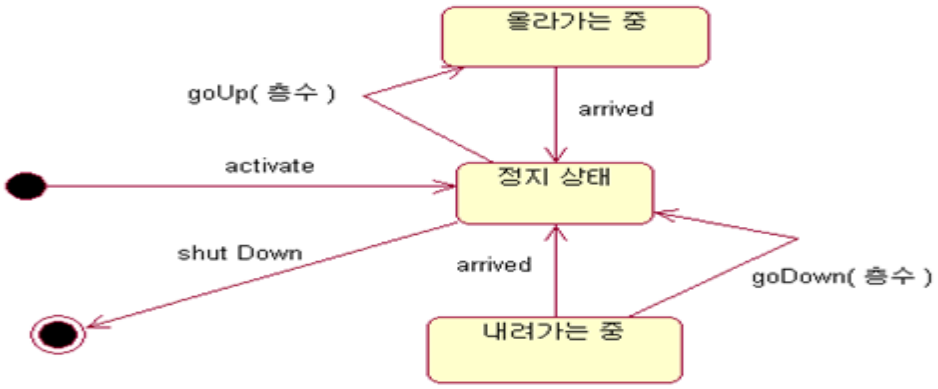
- ✓ 등록기는 소비자가 투입한 돈의 액수(input)가 음료수의 값(price)과 맞는지 체크한다.
- ✓ 만일 액수가 음료수 값보다 많으면, 등록기는 차액을 계산하고 그 만큼의 현금 잔고가 있는지 체크한다.
- ✓ 만일 차액 만큼의 현금이 잔고(cash reserve)에 남아 있다면, 등록기는 거스름돈(change)을 내어주고 나머지 동작은 그전과 똑같이 진행한다.
- ✓ 만일 차액 만큼의 현금이 잔고에 남아 있지 않으면, 등록기는 소비자가 투입한 돈을 그대로 돌려주고 "맞는 액수의 돈을 넣어 주세요"란 메시지를 표시한다.
- ✓ 만일 소비자가 투입한 돈의 액수가 음료수 값보다 적으면, 등록기는 아무것도 하지 않고 돈이 더 들어올 때까지 대기한다.

- 조건문을 표현하기 위해서는 대괄호 [ ] 안에 조건을 써주고, 이것을 메시지 화살표 위에 놓으면 된다.
- "액수가 맞지 않는 경우"의 시나리오



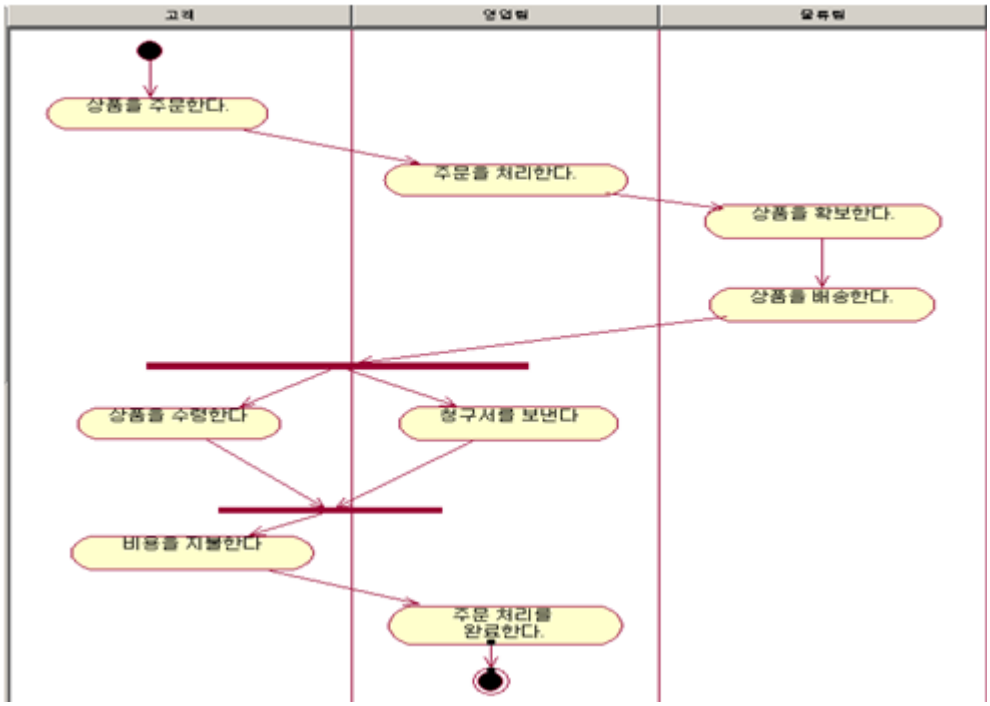
(5) State Chat Diagram

- 하나의 객체 또는 시스템 전체에 대해서 객체 내부 또는 시스템의 자세한 행동을 기술하는데 이용된다.
- 엘리베이터 객체의 행동 표현



(6) Activity Diagram

- 수행되는 활동의 순서와 활동 수행의 주체 등을 기술할 때 유용한다.
- 업무 흐름, 화면 흐름, 연산의 알고리즘을 표현할 때 이용된다.
- 상품 주문에 대한 처리 과정

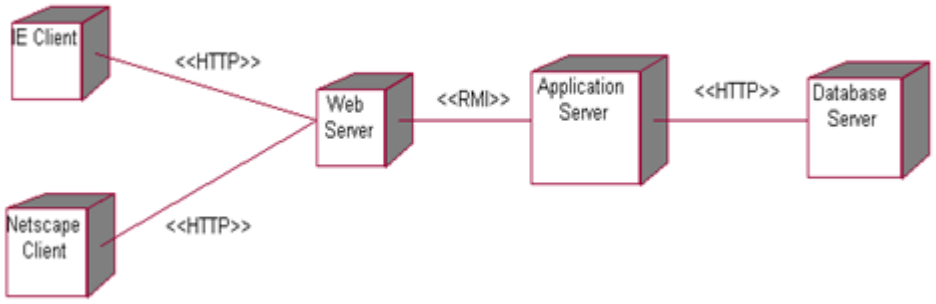


(7) Component Diagram

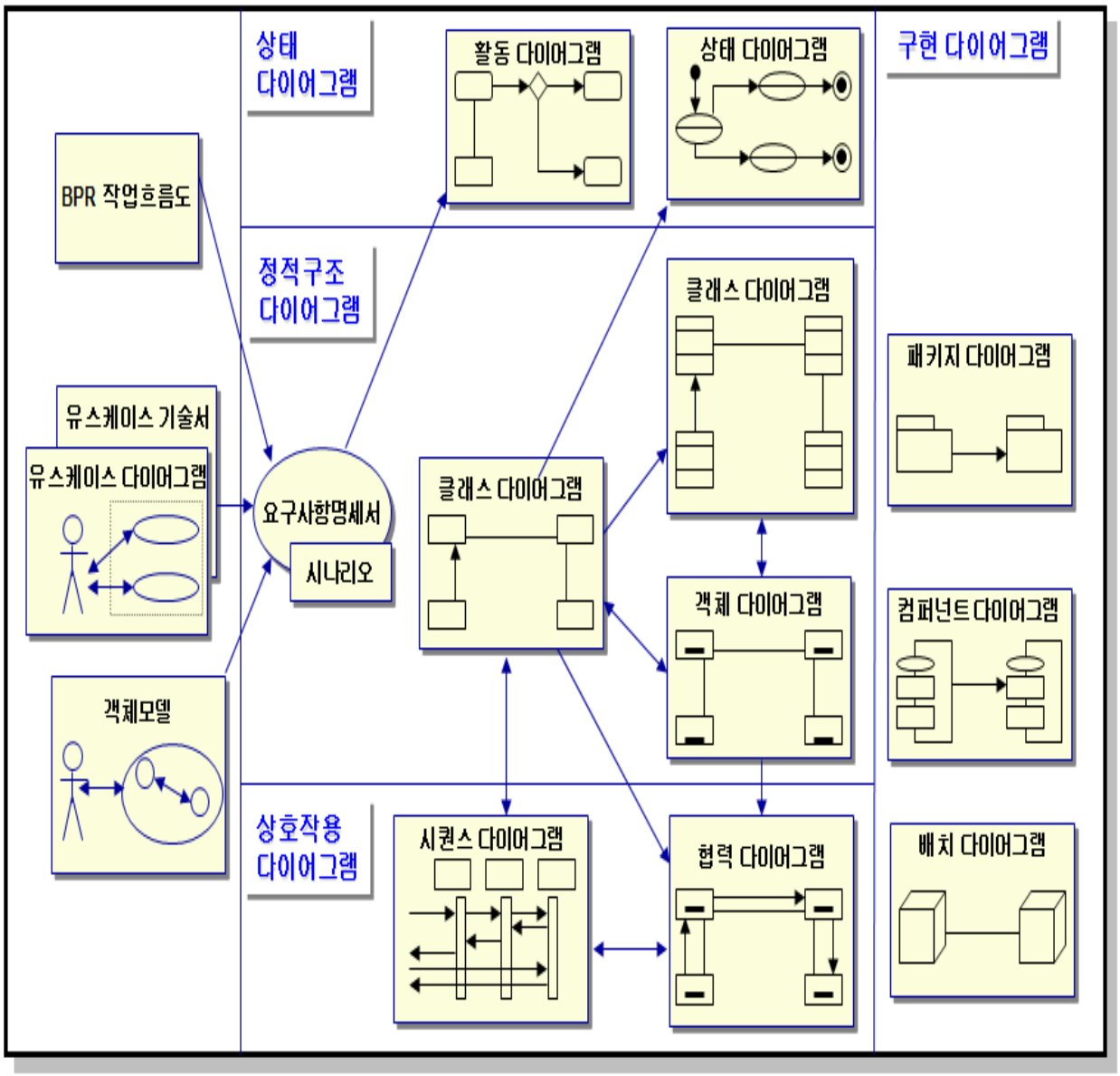
- 시스템을 구성하는 물리적인 컴포넌트와 그들 사이의 의존 관계를 표시한다.
- UML 에서의 컴포넌트란 실제로 구현되어서 존재하는 것(파일)

(8) 배치 다이어그램

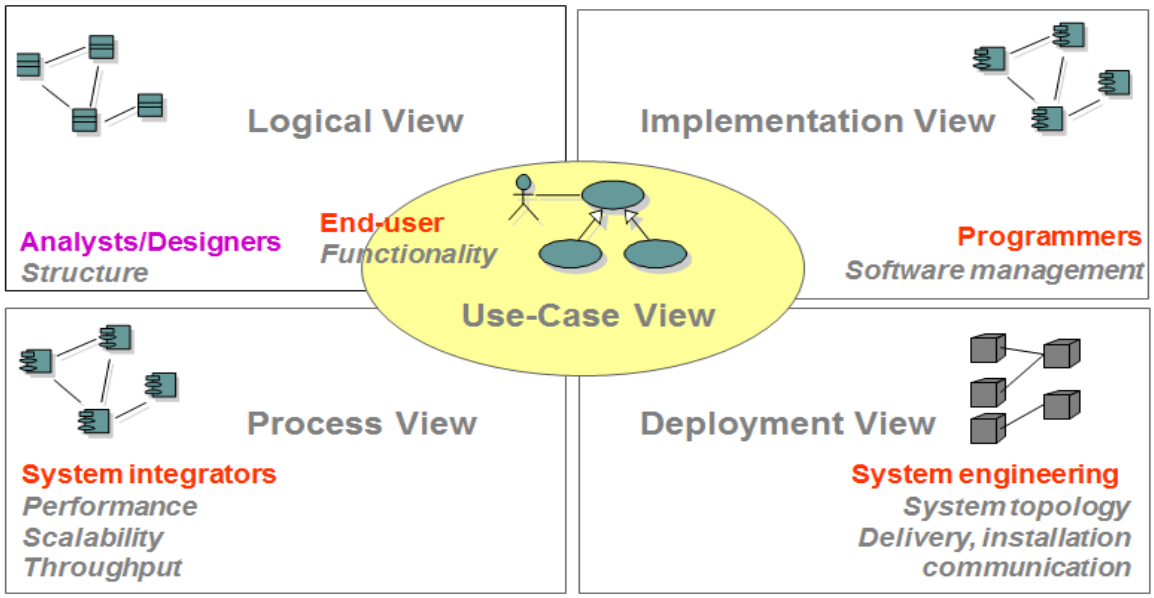
- 시스템의 물리적인 요소를 모델링 할 때 사용된다.
- 시스템을 구성하는 처리 장치 즉, 컴퓨터와 그들 사이의 통신 경로를 기술할 때 사용된다.



(9) UML Diagram들간의 관계



4. View of Software (4+1 View)



(1) Use Case View

- 시스템의 행동을 시나리오 형태로 모델링
- 시스템의 아키텍쳐를 형상화한다.
- 시스템을 볼 때 시스템 외부에 있으면서 시스템과 상호작용을 하는 대상과 시스템이 제공해야 할 기능이 무엇인지에 초점을 둔다.
- 개발자뿐만 아니라 고객에게도 중요하다.
- 시스템을 검증 시 중요한 역할을 한다.
- 시스템이 제공해야 하는 기능이 무엇(What)인가에 초점을 둔다.

(2) Design View

- 시스템의 기능적(functional)인 요구사항을 작성한다.
- Class Diagram으로 정적인 면을 보여주고 동적인 면은 Interaction, Activity, Statechat Diagram으로 표시한다.
- Use Case 관점에서 정의된 기능을 시스템이 제공하기 위해서는 어떤 클래스가 필요하고 이들 클래스들이 서로 어떻게 이용/호출되는 지에 초점을 맞춘다.
- 시스템의 내부 구조 및 내부 기능에 초점을 맞춘다.

(3) Process View

- 주로 시스템의 성능(performance), 스케일(Scalability), 처리능력(throughout)을 보여준다
- 시스템을 구성하는 구현된 물리적인 요소(컴포넌트) 즉, 파일과 파일 간의 의존 관계에 초점을 둔다.
- 컴포넌트 다이어그램이 이용된다.

(4) Implementation View

- 배포 판의 현상관리를 다루며 주로 시스템의 모듈 조직을 관리한다.

(5) Deployment View

- (Physical) 시스템을 구성하고 있는 분산, 인도, 설치를 다룬다.
- 시스템을 구성하는 물리적인 처리 장치와 각 처리 장치에 배치되는 컴포넌트에 초점을 둔다.
- 배치 다이어그램

Part 5. 요약

- 객체지향 개발 방법은 시스템의 확장이나 변경을 쉽게 할 수 있는 뛰어난 기법으로 인식
- 객체지향 분석기법은 기존의 분석기법이 가지고 있는 한계점을 극복 하고 시스템의 3가지 관점을 체계적으로 통합하여 기술하고, 확장성과 적응력이 뛰어난 시스템을 설계하는 것이 그 목적이다.
- 객체지향 개발 방법은 우선 시스템에서 사용되는 데이터에 초점을 맞추기 때문에 변화에 잘 견디는 우수한 품질의 소프트웨어를 만들 수 있다.
- 객체지향 개발 방법은 시스템의 확장성이 용이하고 엔지니어링의 결과를 재사용할 수 있어 원형 패러다임이나 나선형 패러다임을 적용한 시스템 개발에 많은 도움을 줄 수 있다.
- 튼튼하고 견고한 시스템은 분석 과정에서 집중적으로 이루어지는 철저한 검증 과정과 이에 따른 수정 없이는 불가능하다.
- 많은 프로그래머들이 객체 중심보다는 기능 중심에서 생각하여 시스템을 개발하여 왔으며 객체지향 사고로의 전환 및 문화의 변화를 필요로 한다.
- 객체지향 개발 방법은 소프트웨어 공학의 목적인 고품질의 소프트웨어를 얻을 수 있는 바른 기법이며 수정, 이해, 재사용을 용이하여 많은 개발 분야에 확산되고 있다.
- 객체지향 설계는 응용분야의 관점에서 컴퓨터 관점으로, 사용자 관점에서 개발자 관점으로 이동하여 객체지향 분석 결과에 살을 붙여가는 과정이다.