Introduction to UML

Software Modeling & Analysis

Report #1

과 목 정 보		학 생 정 보	
학 교 명	건국대학교	전 공	컴퓨터 공학부
학 기	2015학년도 1학기	과 목 명	소프트웨어 모델링 및 분석

팀원

201011314 김민재 201011349 이규진

201011356 이종찬

Contents

1. UML에 대하여

1-1. UML 설명

1-1-1. UML 정의

UML이란, unified modeling language의 약어로 객체지향분석과 설계를 위한 모델링 언어이다. UML을 사용해서 개발자들은 시스템에 대한 사전 모델을 만들 수 있게 되었다. 다양한 프로그래밍 작업을 하게 되면, 사람과 사람사이의 전체 시스템에 대한 의사소통의 불일치가 있을 수 있는데 이것을 시각적이고 직관적으로 이해하기 쉽게 보여주는 것이 바로 UML이다. 따라서 개발자간의 의사소통이 쉬워진다.

UML은 프로그래밍 언어가 아니지만 존재하는 몇 가지의 추가 도구를 사용하여 여러 가지 다른 프로그래밍 언어로 변환이 가능하다. 그러므로 UML은 도식화된 언어이며 소프트웨어를 만드는데 긍정적 효과를 제공한다.

1-1-2. UML 역사

UML의 역사를 표현한 그림이다. UML은 1980년대 후반에서 1990년대 초반에 개발된 객체 지향 개발방법에 뿌리를 두고 있으며, 1990년대 중.후반 이후로 빠른 속도로 발전하고 있다.

1-1-3. UML 장단점

UML의 장점은 셀 수 없이 많지만, 그 장점 중에 큰 범위로 3개를 표현한다. 첫째, 팀 구성원의 의사소통 효율의 향상을 이끌어 낼 수 있다. 실제로 프로젝트 또는 어떤 일을 진행하다 보면, 의사소통으로 소비되는 시간이 매우 많다는 것을 알 수 있다. 특히 조직 또는 팀 구성원이 많을수록 의사소통 시간이 급격하게 증가할 것이다. UML은 설계와 디자인을 겸비한 도구이다. 모델링의 장점은 이해하기 쉽고, 서로 약속된 표현법을 활용하므로 오해의 소지가 줄어든다. 이처럼 UML을 활용하면, 의사소통 시간을 줄일 수 있고 데이터 모델을 활용하면 더욱 좋다.

둘째, UML은 국제 표준이다. 최근에는 객체 지향 개발방법뿐만 아니라 다양한 개발방법에서 활용

하고 있고, 설명서를 UML에 기반을 두어 작성하고 있다. 소프트웨어 아키텍처 패턴이나 디자인 패턴을 소개하는 문서를 살펴 볼 때 UML을 이해하지 못하면 해당 패턴을 이해하기가 매우 어려울 수 있다. 그러나 UML을 정확히 알고 있다면, UML로 작성된 방대한 정보에 쉽게 접근할 수 있고, 이 정보를 기반으로 다양한 아이디어를 만들어 낼 수 있을 것이다.

마지막으로 문서화가 쉽다. 프로젝트를 진행하다 보면, 프로젝트 일정과 비용 등으로 문서화를 진행하는데 어려움을 느낄 때가 있을 것이다. 하지만 UML을 사용하면 모델링 정보를 쉽게 접근할 수 있고 프로젝트를 문서화 할 때 훌륭한 매개체가 될 수 있다.

이처럼 여러 가지 UML의 장점도 있지만 UML을 사용할 때 생각해야 할 고려사항도 몇 가지가 있다. 첫째, UML을 사용하는 이유가 명확해야 한다. 프로젝트와 관련된 내용을 고려한 후, UML의 도입 방향을 제시해야한다. 굳이 사용하지도 않을 UML을 작성하면, 쓸데없이 시간과 비용을 소비하게 된다.

둘째, UML 다이어그램 작성이 쉽지 않다. UML 다이어그램은 모델링 과정이므로 다양한 경험과 숙련도가 필요하다. UML을 이해하고 있는 것과 보는 것 그리고 작성하는 것은 서로 부분이라고 볼 수 있다. 또한 소비자들이 요구하는 요구사항을 분석하여 UML에서 제시하는 다이어그램을 모두 효과적으로 작성할 수 있는 사람이 드물다.

마지막으로 UML 활용 방안의 문제이다. 매우 잘 작성된 UML 다이어그램이라도 프로젝트 구성원이 UML 다이어그램을 이해하지 못하면 빛 좋은 개살구 일뿐이다. 기본적으로 도구를 활용하는 데에는 추가적으로 비용이 소비되는데 어떤 도구든지 비용대비 효율성을 고려하여 합당하다고 인정될 때 사용하는 것이 합리적이다.

1-1-4. UML의 목적

UML의 최우선 목적은 아래와 같다.

- 사용자에게 즉시 사용가능하고 표현력이 강한 시각적 모델링 언어를 제공함으로써 사용자는 의미있는 모델들을 개발하고 서로 교환할 수 있다.
- 핵심적인 개념을 확장할 수 있는 확장성과 특수화 방법을 제공한다.
- 특정 개발 프로세스와 언어에 종속되지 않는다.
- 모델링 언어를 이해하기 위한 공식적인 기초를 제공한다.
- 객체 지향 툴 시장의 성장을 장려한다.

UML은 소프트웨어를 모델링하기 위한 언어이다. 그렇지만 반드시 소프트웨어에 국한되는 것만은 아니다. UML은 시스템의 모든 분야를 모델링 할 수 있어야 한다는 목적으로 탄생하였다. UML은 소프트웨어뿐만 아니라 비즈니스 모델링의 영역도 포함을 한다. 그리고 객체 모델링에 대한 개념뿐만 아니라 데이터 모델링까지도 가능하게 한다. 지금은 UML 스펙이 발전하면서 비즈니스 모델링 부분이 Business Process Modeling Notation 이라는 별도의 영역으로 분할이 되었고 UML은 Model Driven Architecture 사상으로 더 심화된 모습으로 전개되고 있다. 비즈니스 모델링 부분은 CBD(Component Based Development), MDA(Model Driven Architecture), EA(Enterprise Architecture), SOA(Service Oriented Architecture) 등의 트랜드가 자리 잡으면서 비즈니스 모델링의 중요성이 부각되어 UML을 비즈니스 모델링에 적용하는 것이 활발해 지면서 그 영역이 점점 넓어지고 있다.


Jacobson's
Use Case Model
Rumbaugh's
OMT
Booch's
Object Design
UML(Unified Modeling Language)
UML

1-2. UML의 특징

UML의 특징은 크게 4가지로 구분 할 수 있다.

- 가시화 언어
- 명세화 언어
- 구축 언어
- 문서화 언어

덧붙여 설명하자면 소프트웨어의 개념 모델을 시각적인 그래픽 형태로 작성, 표기법에 있어서는 Symbol에 명확한 정의가 존재하므로 개발자 사이에서 원활한 의사소통이 가능하게 해주는 것이 **가시화 언어**이고, **명세화 언어**란 정확하고 명백하며, 완전한 모델을 만드는 것을 의미한다. UML은 소프트웨어 개발과정인 분석, 설계, 구현 단계의 각 과정에서 필요ㄴ 모델을 명세화 할 수 있는 언어이고, **구축 언어**는 UML로 명세화된 설계모델은 JAVA, C++, VB 등 다양한 언어의 소스코드로 변환하여 구축 할 수 있다. 반대로 구축되어있는 소스코드를 UML로 변환하여 분석하는 Reverse도 가능하다. **문서화 언어**는 시스템 아키텍처와 이에 대한 모든 상세 내역에 대한 문서화를 다루며, 요구사항을 표현하고 시스템을 테스트 하는 언어도 제공한다.

1-3. UML의 구성요소

UML의 구성요소는 위의 그림과 같으며 UML의 구성요소를 Thing(사물), Relation(관계), Diagram (다이어그램) 3가지로 크게 나눌 수 있다.

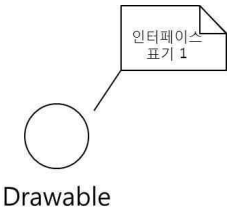
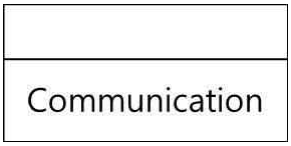
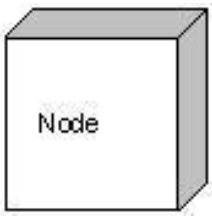
1-3-1. Things(사물)

Thing(사물)에는 4가지가 있다.

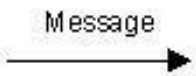
- 구조사물 (Structural)
- 행동사물 (Behavioral)
- 그룹사물 (Grouping)
- 주해사물 (Annotational)

구조사물(Structural)은 모델의 정적인 부분들을 정의, 물리적, 개념적 요소를 표현한다. 하위 요소로 클래스, 인터페이스, 통신, 유스케이스, 활성클래스, 컴포넌트, 노드 7개의 구조사물이 있다.

구조사물	형태	정의
클래스(Class)		이름, 속성, 오퍼레이션을 가지고 있으며 직사각형으로 표현한다.

인터페이스(Interface)		클래스의 외부적으로 가시화되는 요소의 행동을 표현, 특정 클래스나 컴포넌트의 전체 또는 일부분만의 행동
통신(Communication)		객체들 간에 주고받는 메시지
유스케이스(Use Case)		시스템이 수행하는 순차적 활동을 기술해준다. 특정 어플리케이션 프로그램의 상위 메뉴 구분과 비슷하다.
활성 클래스(Active Class)		하나 이상의 프로세스나 쓰레드를 갖는 객체를 파생하는 클래스 기술
컴포넌트(Component)		시스템의 물리적이고 대체 가능한 부분으로, DLL이나 EXE와 같은 물리적 단위에 해당함
노드(Node)		실행 시에 존재하는 실제 전산자원을 의미하며, 대부분의 경우 자체적인 메모리와 처리 능력을 가진 시스템을 말한다.

행동사물(Behavioral)은 UML 모델의 동적인 부분으로 구성되어 있다. 하위 항목으로는 교류(Interaction), 상태머신(State Machine)이 있다.

행동사물	형태	정의
교류(Interaction)		객체들 간의 주고받는 메시지
상태머신(State Machine)		상태의 순서를 지정하는 행동

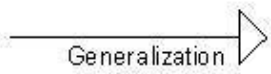
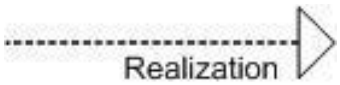
그룹사물(Grouping)은 UML 모델의 요소들을 그룹화 시켜준다. 하위요소로 패키지(Package)가 있는데 UML 모델의 요소들을 그룹화 메커니즘으로 정의한 것을 말한다.

주해사물(Annotation)은 모델을 설명하는 부분이다. 하위요소로는 노트(Note)가 있으며 UML 모델의 주석에 해당하는 것으로, 점선으로 노트와 설명할 대상을 연결한다.

1-3-2. Relationship(관계)

Relation(관계)에는 4가지가 존재한다.

- 의존(Dependency)
- 연관(Association)
- 일반화(Generalization)
- 실체화(realization)

관계	형태	정의
의존(Dependency)		두 사물간의 의미적 관계, 한쪽 사물의 변화가 다른 사물에 영향을 주는 관계
연관(Association)		구조적 관계로서 객체간의 연결을 나타냄, 집합 연관관계는 전체와 부분 간의 구조적 관계를 표현하는 특별한 연관 관계
일반화(Generalization)		주로 클래스간 상속관계를 표현하기 위해 사용
실체화(realization)		객체들 사이의 의미적 관계, 인터페이스와 인터페이스에 오퍼레이션이나 서비스를 제공하는 클래스나 컴포넌트 사이의 관계를 지정하기 위해서 사용

1-3-3. Diagram(다이어그램)

UML의 다이어그램은 두가지 유형이 있다.
구성요소를 표현하기 위한 구조적 다이어그램(Structural Diagram)과 행위를 표현하기 위한 행위 다이어그램(Behavioral Diagram)이 있다.

UML	다이어그램	정의
구조적 다이어그램 (Structural Diagram)	클래스 다이어그램 (Class Diagram)	모델의 조립 부품의 집합. 클래스와 관계에 의해서 구조와 관계를 표현한다.
	객체 다이어그램 (Object Diagram)	시스템을 구성하는 객체, 객체간의 관계를 표현

	컴포넌트 다이어그램 (Component Digram)	소프트웨어-유닛 간의 의존관계를 나타내는 것으로, 소프트웨어 모듈 구성이나 버전 관리도 표현할 수 있다.
	배치 다이어그램 (Deployment Diagram)	객체나 패키지, 파일 등을 실제 플랫폼이나 네트워크 노드 상의 어디에 배치할 것인지, 그리고 어느 프로세스 상에서 실행할 것이라는 물리적인 관점에서 시스템 구성을 표현한다.
행위 다이어그램 (Behavioral Diagram)	유스 케이스 다이어그램 (Use Case Diagram)	시스템의 문맥과 외부 기능의 설정
	순서 다이어그램 (Sequence Diagram)	상호작용하는 객체의 시간 순서 즉, 객체의 집단 메시지 송신에 대한 시계열표현이다.
	상호작용 다이어그램 (Interaction overview Diagram)	객체 집단에서의 상호작용에 대한 직접적 표현이나 객체 집단의 접속망의 형태와 메시지, 스레드의 순서 표현
	상태 다이어그램 (State Diagram)	1개의 객체 생성에서 소멸까지 상태. 즉 어떤 클래스에 속하는 객체의 싸이클 표현을 제공한다.
	액티비티 다이어그램 (Activity Diagram)	1개의 Interaction 전체에서의 순서 제어 플로우. 상태 다이어그램의 상대적 표현으로서 워크플로우에 초점을 맞춘 다이어그램

이 중에서 중요하게 생각되는 2가지 다이어그램을 조금 더 자세하게 서술한다.

- 클래스 다이어그램(Class Diagram)
- 순서 다이어그램(Sequence Diagram)

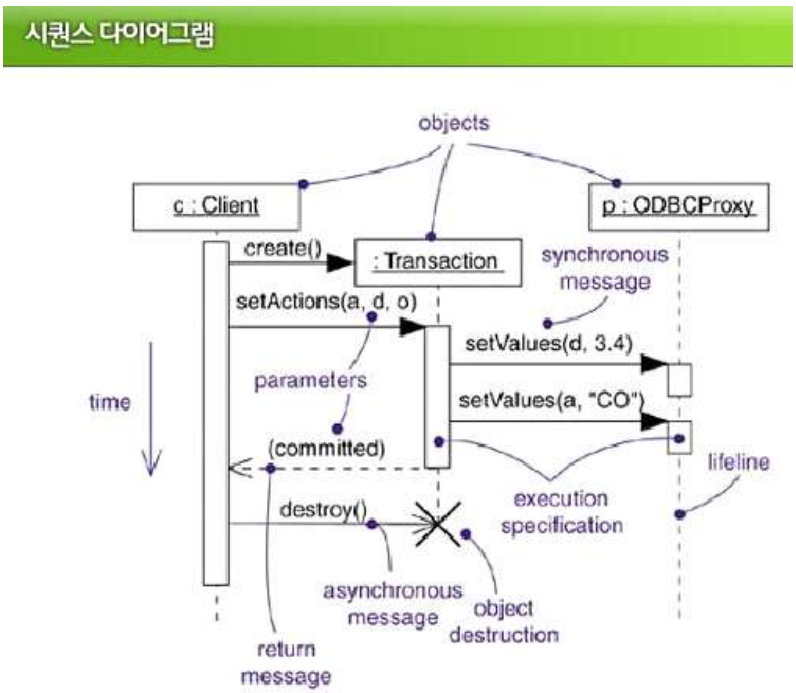
클래스 다이어그램(Class Diagram)

클래스 다이어그램은 시스템의 정적인 상태인 논리적인 구조를 표현한다. Class, Interface, Collaboration 간의 관계를 나타내며, 객체지향 개발에서 가장 공통적으로 많이 사용한다. 클래스 다이어그램을 구성하는 것은 클래스와 관계이다.

클래스 다이어그램은 다음과 같은 특징을 가진다.
가. 시스템의 요구사항에 표현된 작업 즉, 시스템이 처리해야 하는 작업에 대한 책임을 분할한다.
나. 모델은 점점 증가되며 관련된 클래스들 끼리 패키지화 시킨다
다. 클래스가 너무 작게 쪼개거나 기능을 너무 많이 포함하면 안되며 적절한 방법으로 구현한다.
클래스 다이어그램은 시스템의 정적 설계도인 Class Diagram과 시스템의 프로세스도인 Active Class Diagram으로 구분할 수 있다.

클래스를 구성하는 것은 클래스명, 속성, 메소드이다. 모든 클래스는 다른 클래스들과 구별되는 유일한 이름을 갖는다. 클래스명은 단순명과 경로명 두 가지 중 하나를 선택할 수 있다. 단순명은 클래스 이름만 표기하는 방법이며, 경로명은 패키지명을 포함하여 표기하는 방식이다.

순서 다이어그램(Sequence Diagram)



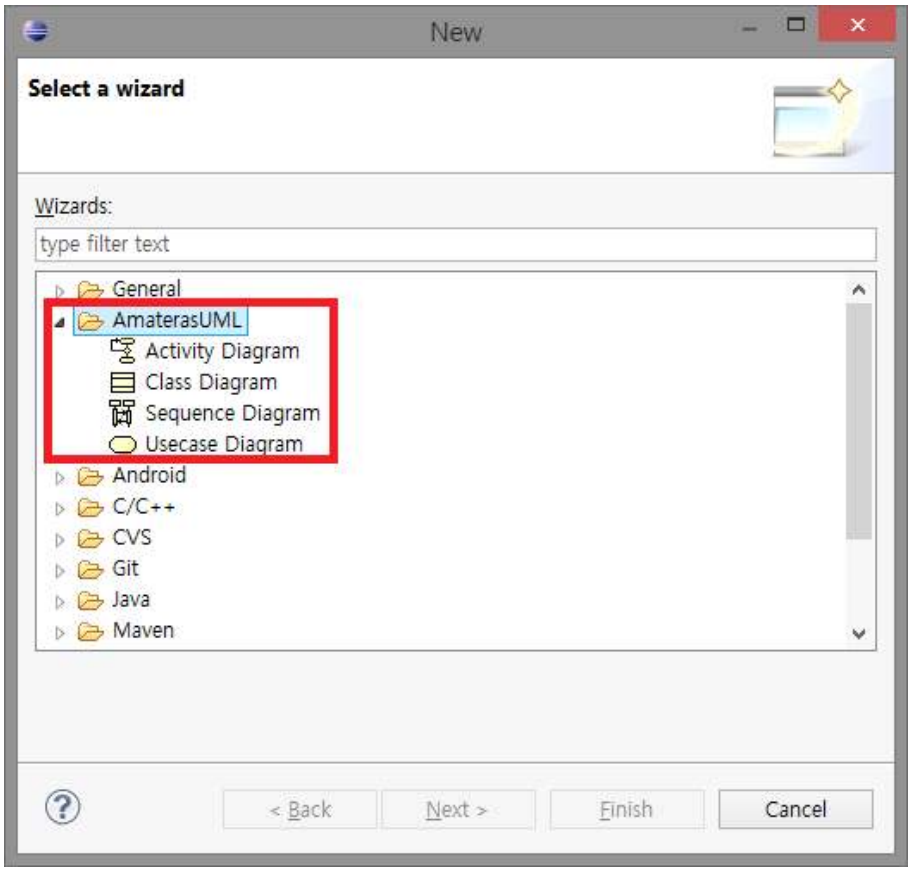
순서 다이어그램은 인스턴스들이 어떻게 상호작용을 하는지를 묘사한다. 하나의 협동-인스턴스집합에 포함된 인스턴스들 상호간에 주고받는 자극들의 집합인 상호작용-인스턴스집합을 직접적으로 표현한다. 순서 역할 다이어그램은 역할 중심의 관점을 반영한 반면, 순서 다이어그램은 인스턴스 중심의 관점을 반영한 것이다.

2. UML 도구 소개

2-1. Amateras UML

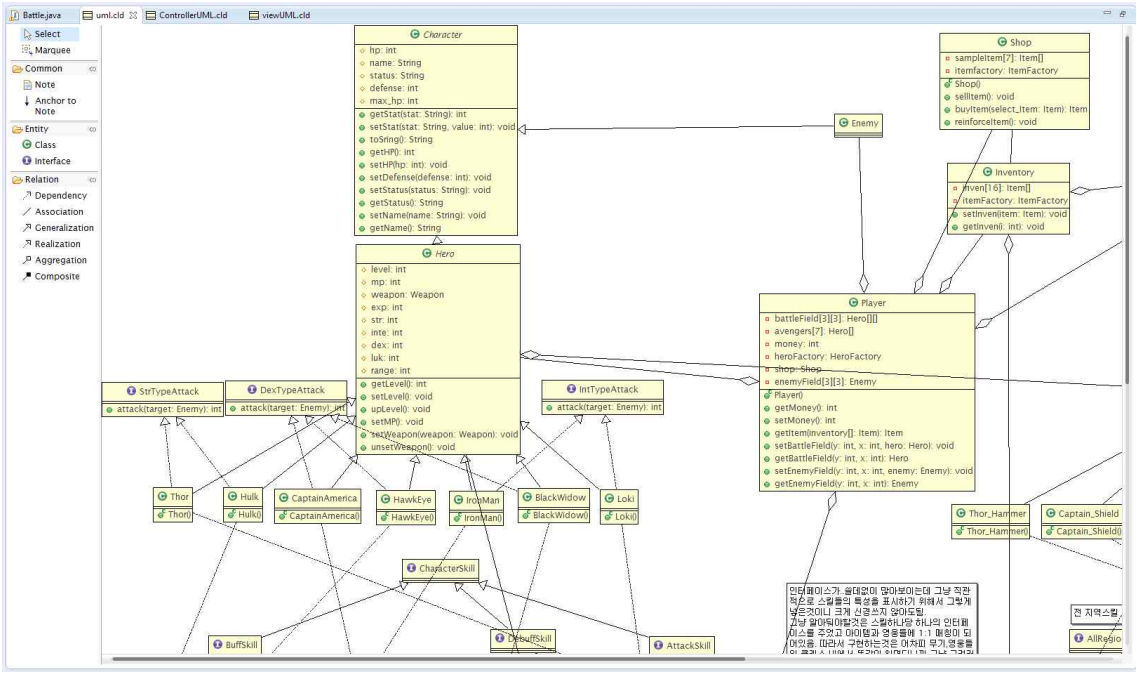
Amateras UML은 일본에서 만든 오픈소스 UML 도구로서 JAVA Eclipse에서 직접 Plug-in으로 연결하여 사용 할 수 있는 프로그램이다.

Amateras Download Link : http://sourceforge.jp/projects/amateras/releases/



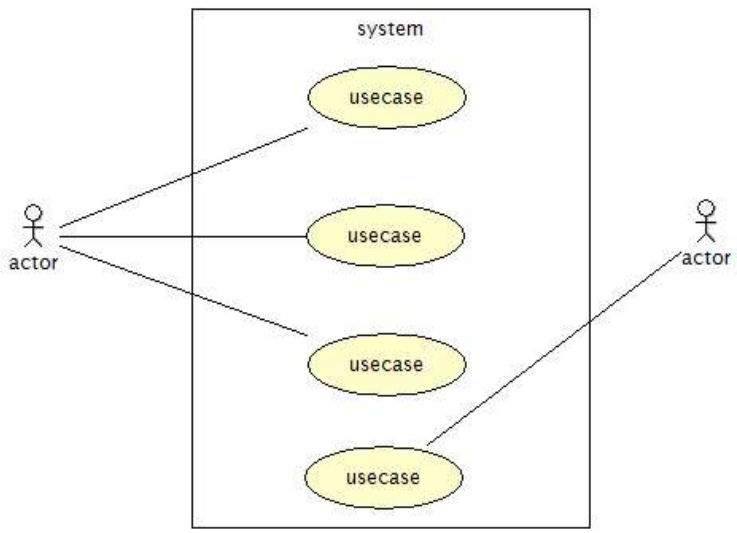
〈 Amateras UML Plug-in 실행 모습 〉

Amateras UML의 장점은 Eclipse에 직접 연동되어 이클립스 내에서 UML을 그린 후 해당 UML을 토대로 기본 코드를 자동으로 생성할 수 있는 기능이다. 이는 실제로 개발 시 편리함을 제공할 뿐만 아니라 디자인 한 대로 코드를 작성하도록 길라잡이 역할을 해준다.



Amateras UML을 이용하여 Elipse내에서 Class Diagram을 그린 모습, Amateras는 Class Diagram, Activity Diagram, Sequence Diagram, Use-Case Diagram 기능을 제공한다.

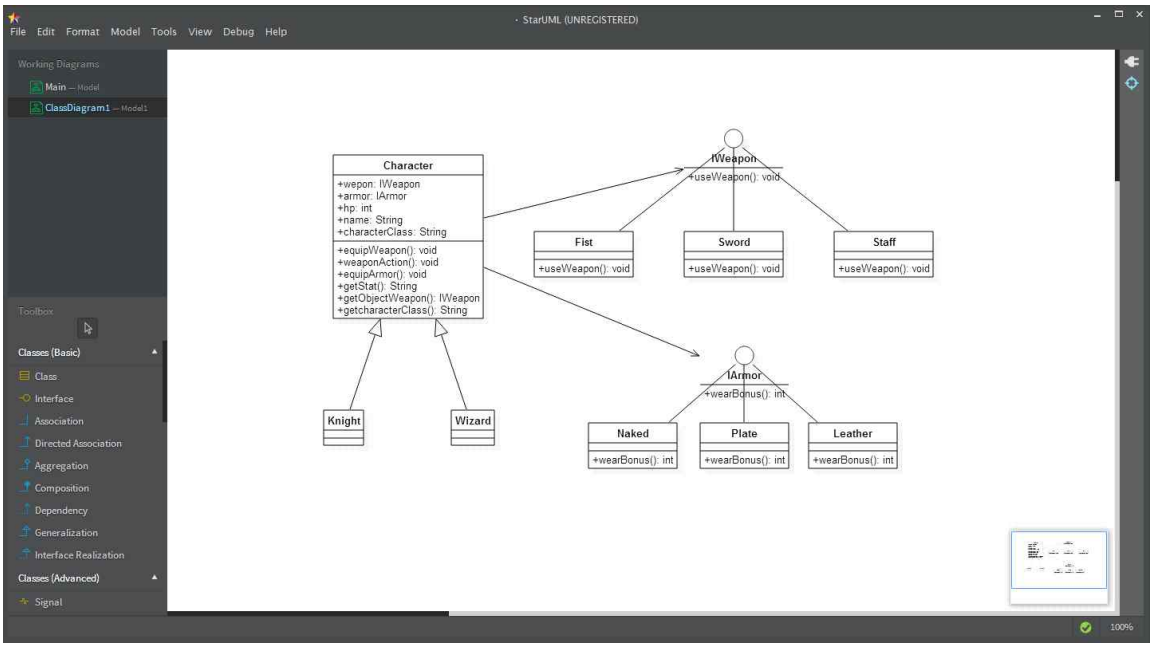
하지만 직접 사용해본 결과 Amateras는 안정성에 문제가 존재하였다. 저장 시에 문제가 발생하면 어떤 경고도 없이 통째로 Diagram이 날아가 버리거나 Diagram 수정 시에 하나를 고치면 다른 변수명이나 자료형이 마음대로 바뀌는 버그도 존재하였다. 물론 프리웨어이므로 완벽함을 바랄 수 없는 것이 사실이므로 어느 정도는 감안 할 수 있는 사항이다.



〈 Amateras UML의 Use-case Diagram 〉

2-2. Star UML (ver 2.0)

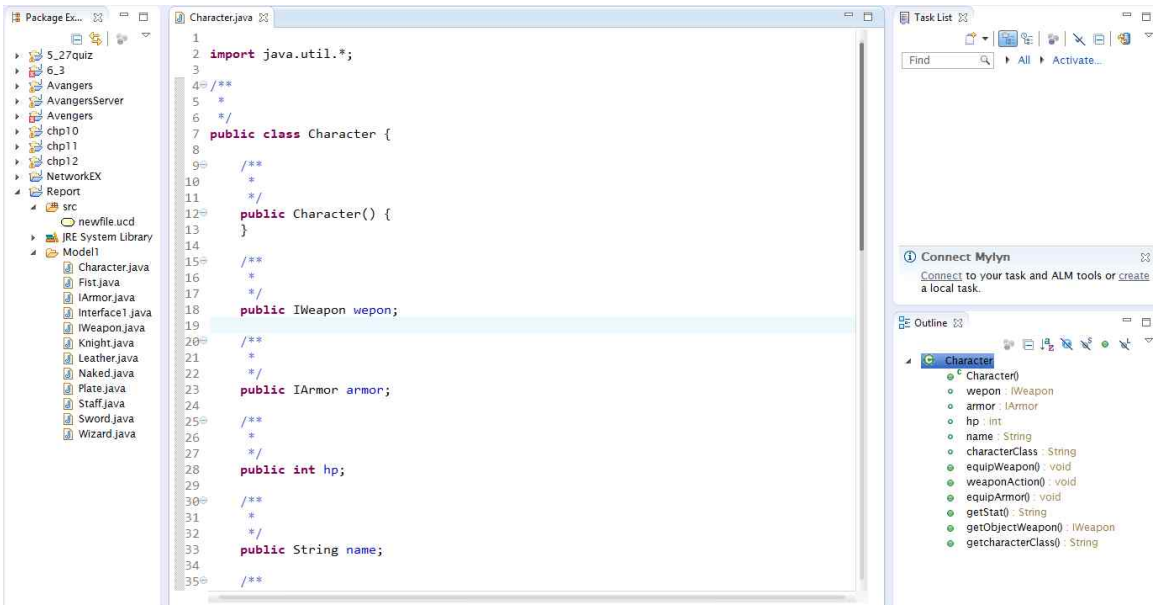
Star UML은 한국 개발자들이 만든 UML 도구이며 부분 프리웨어이다. (실행 시 구입을 권장하는 문구가 뜨지만 구입하지 않아도 무기한 사용가능하다) Star UML은 그 전에도 좋은 평가를 받아왔지만 2014년 Star UML 2를 개발하면서 더 깔끔한 인터페이스와 강력한 기능들이 추가 되었다.



〈 Star UML 2로 Class Diagram을 그린 모습 〉

Star UML 2는 Amateras UML과 달리 이클립스의 플러그인으로 직접 탑재되어지지는 않지만 Extension Manager라는 기능을 통해서 PHP, XML, JAVA, C#, C++과 연동하여 사용할 수 있다. 예를 들어 JAVA의 경우, UML을 코드로 Generate하는 기능을 이용하면 작성한 Class들의 기본 코드가 생성되어 쉽게 사용할 수 있다. 같은 프리웨어 툴인 아마테라스와 비교했을 때 Code Generate 기능에 깔끔하고 편리한 UI까지 얹은 Star UML이 한 수 위라고 표현할 수 있다.

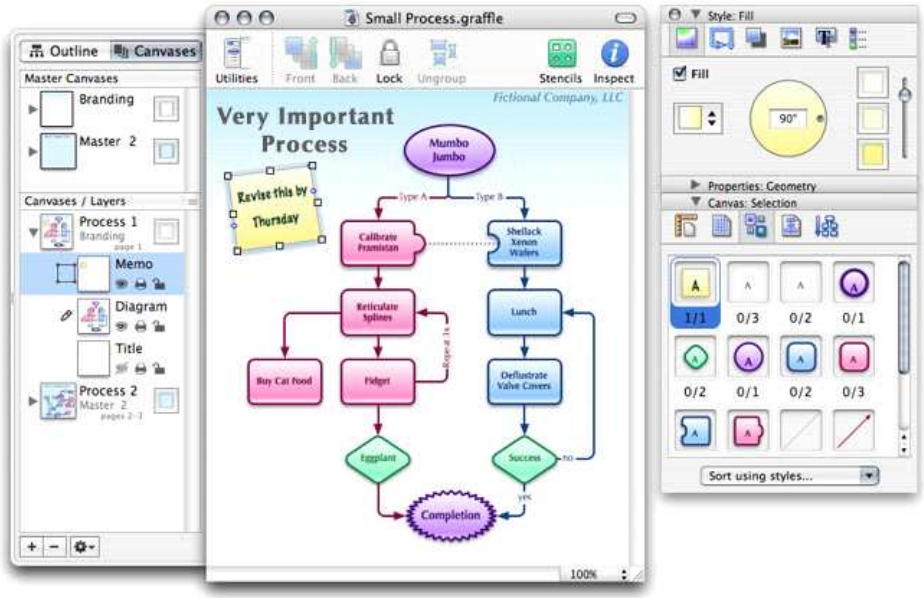
하지만 한 가지 의문점은 Interface와 Interface Realization 표기 방법이 기존 UML 툴들과 다르다는 것이다. 이렇게 하도록 한 이유는 잘 모르겠지만, Star UML로 작성한 Class Diagram을 처음 접하는 제 3자가 보았을 때 혼돈을 줄 수 있을 것으로 생각된다.



〈 Star UML에서 Generate한 코드를 이클립스에서 실행한 모습 〉

2-3. Omnigraffle

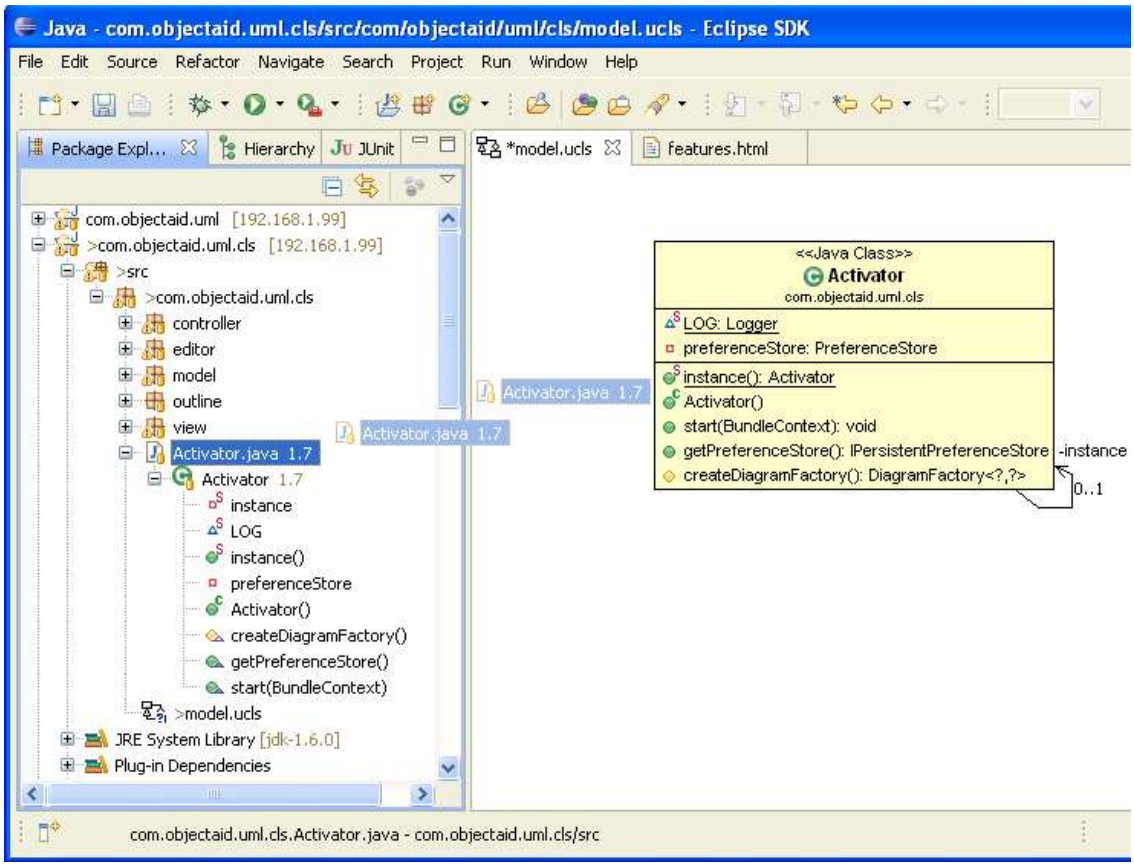
Omnigraffle은 매킨토시와 iOS를 위한 프로그램으로 유료 프로그램이다. 따라서 직접 사용해보진 못했지만 전문 UML Tool이라기 보다는 회의나 여러 가지 도형을 이용한 시각화를 필요로 할 때 사용하는 Tool이다. 따라서 전문적인 UML Tool은 아니므로 실제 Class Diagram을 작성할 때에는 불편함이 있을 수 있다. (실제로 옴니그라플을 사용하여 UML을 그릴 때 불편하다는 리뷰를 어렵지 않게 찾을 수 있었다) 하지만 사용하기 편리하고 직관적인 인터페이스는 옴니그라플의 큰 장점이다.



〈 Omigraffle 실행 화면 〉

2-4. Object Aid (http://www.objectaid.com/home)

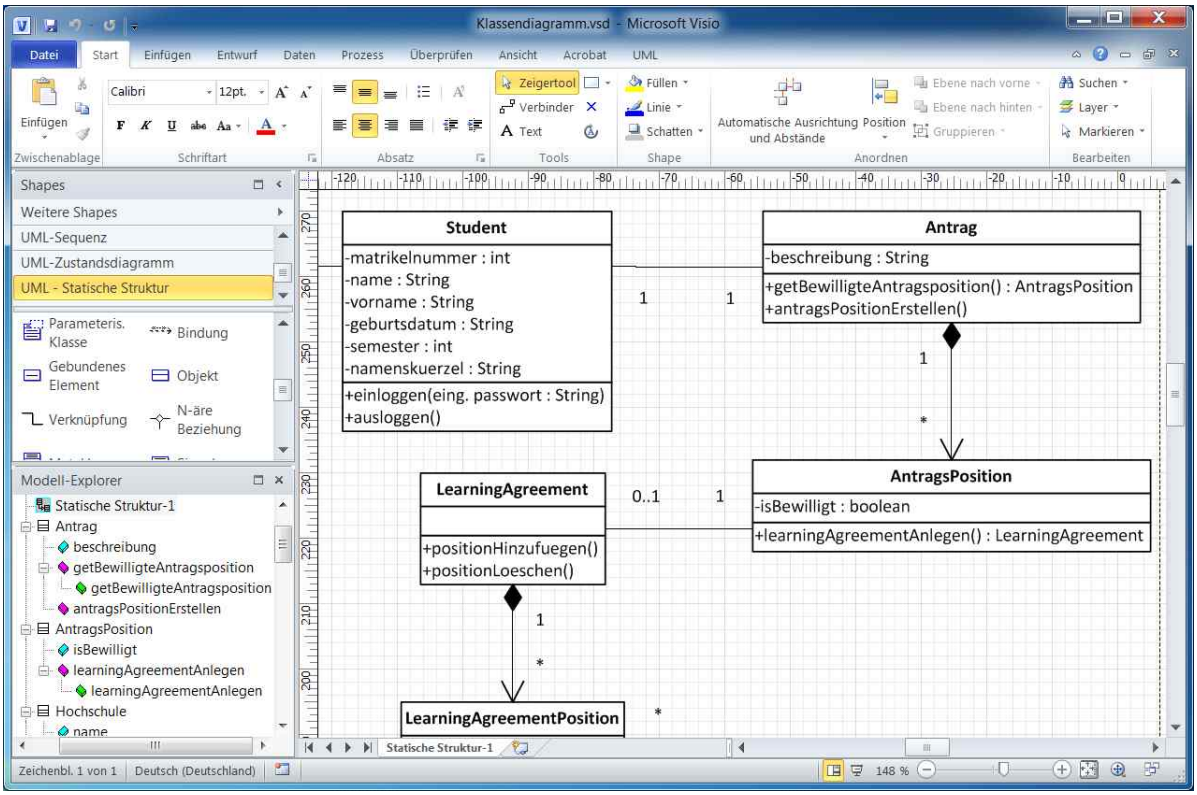
Object Aid는 Amateras UML과 마찬가지로 이클립스의 플러그인 중 하나이다. 하지만 기능은 아주 다르다. Amateras UML의 경우 직접 Diagram들을 편집한 후 코드화 하는 용도로 쓰이지만, Object Aid는 반대로 진행중이거나 완성된 프로젝트의 코드를 가지고 거꾸로 UML Diagram으로 만들어주는 역할을 한다. 다른 도구들과 역할이 다르므로 장단점을 비교할 수 는 없지만 다른 오픈소스 프로젝트들의 설계를 참고하거나 UML에 관하여 공부할 때 많은 도움을 줄 수 있는 도구이다.



Class 파일를 드래그 앤 드롭하면 Class Diagram으로 변환하여 보여준다.

2-5. MS Visio (http://office.microsoft.com/ko-kr/visio/)

Microsoft에서 제작한 다이어그램 전용 유료 프로그램이다. 다양한 다이어그램 템플릿과 편리한 기능들을 제공하고 있으며 벡터 그래픽을 사용하여 그림을 확대해도 깨지거나 손상되지 않는다. UML 작성 외에도 여러 가지 정보를 빠르게 시각화 할 수 있다는 점에서 비즈니스 도구로써 매우 활용도가 높다.



〈 MS Visio를 통해 그린 Class Diagram 〉

3. UML과 모델링 방법론

모델링이란, 모델을 만드는 일(추상화)로써 품질이 좋은 소프트웨어를 개발 및 배치할 수 있게 하는 모든 활동의 중심이다. 모델 구축을 통해 개발 대상 시스템에 대한 이해를 증진 시킬 수 있다.

UML에서 모델간의 구분을 하는 것은 굉장히 중요하다. 서로 다른 UML 모델 타입은 서로 다른 다이어그램을 사용하기 때문이다. UML 모델링은 다음과 같은 주요 요소들이 있다.

3-1. 구조 모델링 (Structural Modeling)

구조 모델링은 시스템의 전반적인 구성 뼈대를 잡아주는 역할을 한다. 구조 모델링은 다음과 같은 다이어그램들로 이루어진다.

3-1-1. Class Diagram

클래스 다이어그램(Class Diagram)은 시스템의 정적인 상태인 논리적인 구조(클래스)를 표현한다. Class, Interface, Collaboration 간의 관계를 나타내며, 객체지향 개발에서 널리 사용되고 있다. 클래스 다이어그램은 클래스와 관계로 이루어진다.

* 특징 및 주의점

1) 시스템 요구사항에 명시된 작업, 즉 시스템이 처리해야 하는 작업에 대한 클래스 간의 역할을 명시해준다.

2) 시스템의 규모에 따라 점점 커지게 되며, 클래스들을 묶어 패키지화 할 수 있다.

3) 클래스를 너무 작게 표현하거나 기능을 너무 많이 포함하면 효율적이지 않으므로 적절하게 나누어야 한다.

* 클래스 다이어그램의 구성요소

1) 클래스 (Class)

클래스를 구성하는 것은 클래스명, 속성, 메소드이다. 모든 클래스는 다른 클래스들과 구별되는 유일한 이름을 갖는다. 클래스명은 단순명과 경로명 두 가지 중 하나를 선택할 수 있다. 단순명은 클래스의 이름만 표기하는 방식이며, 경로명은 패키지 명까지 포함하여 표기하는 방식이다.

속성은 의미있는 명사형으로 표현한다.

```
Visibility Name: Type= Default Value
+ variableName: byte
```

Visibility는 접근 제한자를 나타내며 표기법은 다음과 같다.

표기법	접근 제한자
+	public
-	private
#	protection

메소드는 의미 있는 동사형으로 표현하며 표기법은 다음과 같다.

Visibility Name(Parameter-List): Return-Type e xpression 예)+ methodName (int param): int

클래스 표기법에는 스테레오 타입(Stereo-Type)을 붙일 수 있는데, 스테레오 타입이란 UML의 한정된 모델 요소를 가지고 새로운 어휘를 표현하기 위한 방법이다. 메소드 명 위에 아래의 예처럼 스테레오 타입을 붙이면 해당 메소드는 생성자라는 것을 표기하는 것이다.

<< constructor >>

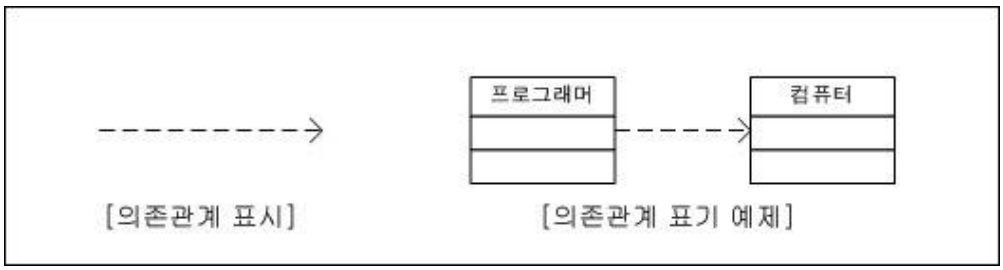
이런 방법으로 클래스명 위에 인터페이스나 추상 클래스임을 나타낼 수 있다.

2) 관계 (Relationship)

관계는 모델 요소 간의 논리적 또는 물리적인 연결을 의미하며, 여러 객체의 유기적인 관계를 통해 시스템이 실행된다.

1. 의존관계(Dependency)

'Using' 관계를 나타내며, 하나의 모델 요소가 다른 모델 요소를 사용하는 관계를 말한다. 사용되는 모델 요소가 변경되면 사용하는 요소가 영향을 받지만, 역은 성립되지 않는다. UML 표기법은 점선으로 된 화살표로 표현합니다. 화살표의 방향은 사용하는 쪽에서 사용되는 쪽으로 향한다.

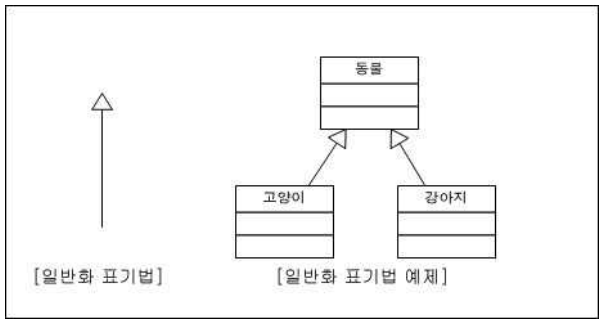


예제는 프로그래머 클래스가 사용하는 쪽이고 컴퓨터 클래스가 사용되는 쪽이다. 사용되는 클래스가 사용하는 클래스의 메소드 파라미터로 사용되는 경우, 사용되는 클래스가 사용하는 클래스의 메소드 로컬 변수로 사용되는 경우, 사용되는 클래스가 사용하는 클래스의 전역 변수로 사용되는 경우이다.

의존 관계는 has a 관계를 가지는 클래스들 간에 변수나, 메소드의 파라미터의 사용을 가지는 클래스의 관계를 표시한다.

2, 일반화(Generalization)

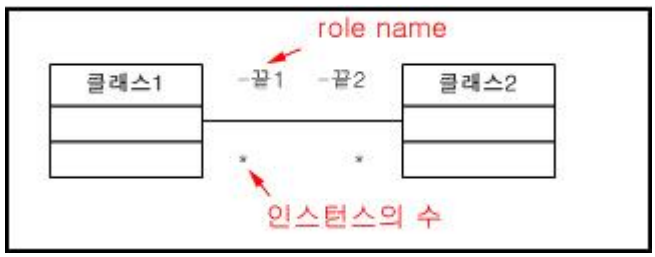
여러 클래스가 가진 공통적인 특징을 추출하여 공통적인 클래스를 일반화시키는 것을 의미하며, 반드시 클래스간의 'is a' 관계이어야 합니다. 객체지향의 상속 관계를 의미한다.



추상클래스(Abstract) 는 이탤릭체나 스테레오 타입으로 표시한다.

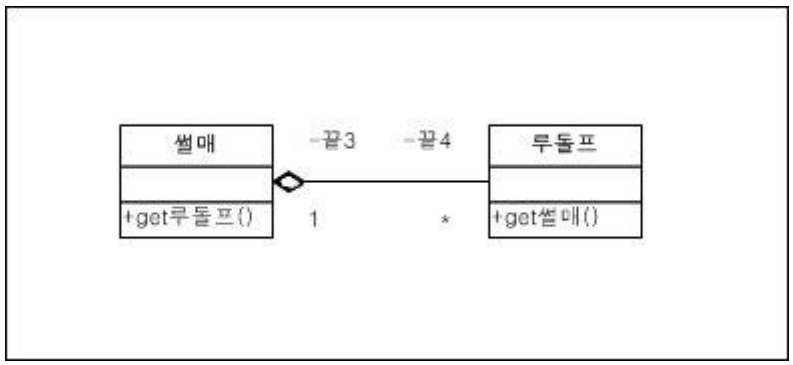
3. 연관관계(Association)

클래스로부터 생성된 인스턴스들 간의 관계를 표현한다. 의존관계와 일반화 관계는 단순히 클래스들 간의 관계를 나타내며, Classifire로부터 생성된 인스턴스 사이의 관계를 나타낸다. 상대방의 인스턴스를 가리킬 수 있는 속성을 가지며, 참조할 수 있는 속성은 UML 상에서 표현하지 않는다. 표현하고자 할 경우 Role name을 사용한다. 연관관계가 가리킬 수 있는 방향의 종류는 양방향과 단방향이 있다.



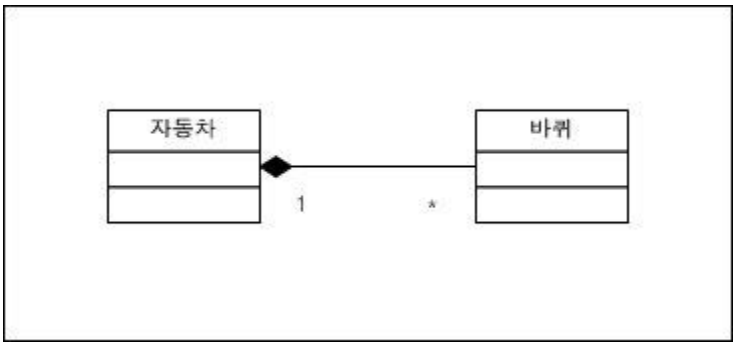
4. 집합연관관계(Aggregation)

전체와 부분을 나타내는 모델요소(whole-part)로 전체를 나타내는 클래스와 이를 이루고 있는 부분 클래스의 관계를 나타낸다. 'has a' 관계를 나타내며 집합연관관계는 전체와 부분은 서로 독립적인 관계를 나타낸다.



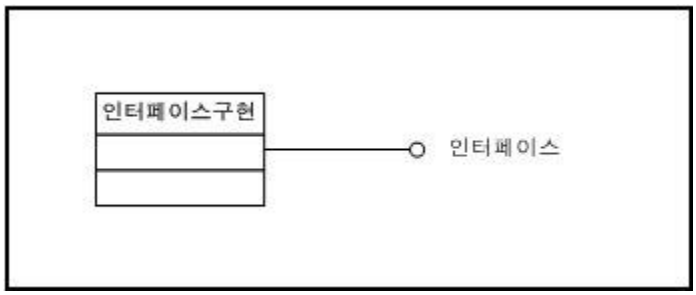
5. 복합연관관계(Composition)

전체와 부분을 나타내며(Whole-part), 젠체를 나타내는 클래스와 이를 이루고 있는 부분 클래스 관계를 나타낸다. 연관관계를 맺고 있는 클래스는 생명주기기 같다. 'has a'관계이다.



6. 실체화, 권력화(Realization)

인터페이스는 컴포넌트 간의 결합력을 느슨하게 한다.(Loose Coupling) 인터페이스는 프로그램의 수정 없이 쉽게 소프트웨어를 확장할 수 있다.



※ 클래스 다이어그램 UML 작성시 주의점

1. 클래스 다이어그램은 'is a'관계를 나타냅니다.
2. 일반화 관계는 균형있게 유지해야 합니다.
3. 선들이 교차하지 않도록 주의해야 합니다.
4. 이해하기 쉬운 정도로 간략하게 표시합니다.
5. 관련 있는 클래스들은 가까운 곳에 배치합니다.

3-1-2. Objects Diagram

객체 다이어그램은 객체를 대상으로 한 그램이다. 모델의 '어느 일부분'을 파악하여 해당 시점의 객체 구조를 나타낸다. 객체지향 시스템에 존재하는 클래스, 클래스 안의 필드, 메소드, 서로 협력하거나 상속하는 클래스 사이의 연결 관계를 표현하는 다이어그램이다.

3-1-2. Deployment Diagram

시스템을 구성하는 소프트웨어와 하드웨어와의 관계를 표현한다. 물리적인 시스템의 구조를 표현하며 네트워크를 사용하는 분산 컴퓨팅 환경을 모델링 할 때 사용하면 유용한다.

Deployment 다이어그램은 물리적인 표현은 노드(Node)를 이용하며, 노드의 종류는 프로세스(Processor)와 장치(Device)로 구성된다. 프로세스는 컴포넌트 실행 기능을 말하며, 장치는 시스템

과 외부 장치와의 연결에 사용한다.

3-1-3. Package Diagram

시스템을 이해하기 위한 목적으로 추상적인 개념들을 모은 하나의 그룹을 패키지라고 한다. 패키지는 요소들을 그룹으로 조직하기 위한 범용 메커니즘으로 모델의 요소들을 조직하고 이해할 수 있도록 해준다. 패키지에 담기는 것은 비단 클래스에만 국한되는 것은 아니며, Use Case, 활동다이어그램등과 같은 것들도 담을 수 있고, 다른 패키지들도 담을 수 있다.

패키지 내부의 모든 클래스들은 개념적, 기능적, 변화적, 관리적 측면에서 유사한면을 가지며, 하나의 패키지는 적절한 수의 클래스를 포함한다. 또한 하나의 패키지 내부의 클래스들은 밀접한 관련성을 가진다. (높은 응집도)

다른 패키지의 클래스들 과는 약한 의존관계가 있다. (낮은 결합도- : 서로 독립적이다) 그리고 패키지는 순환의존관계는 올바르지 않는 구조를 가지지 않아야 하며, 패키지 다이어그램은 패키지와 관계라는 두가지 요소로 표현한다.

패키지는 탭이 달린 폴더 모양으로 표현한다. 단순 표기법은 패키지안에 이름만을 표기하며, 확장 표기법은 내부에 클래스까지 표현한다.

패키지는 클래스를 직접 포함하거나, 이름만 명시하거나, 경로를 표시하거나, 패키지를 포함 할 수 있다.

3-1-4. Composite Structure Diagram

복합 구조 다이어그램은 컴포넌트의 내부 구조를 표현하는데 사용한다. 파트, 포트, 연결 자동의 개념을 이용하여 컴포넌트가 실현, 이용하는 인터페이스와 내부 구성요소간의 관계를 표현한다.

3-1-5. Component Diagram

시스템의 논리적인 요소를 물리적인 요소로 표현한다. 한 개 이상의 클래스를 구현하여 하나의 컴포넌트를 만들 수 있다. 컴파일 될 소스 코드 파일의 관계 등을 표현한다. 실행파일, 동적 링크 라이브러리(DLL), 문서들을 표현한다.

작성된 컴포넌트를 언제든지 재사용할 목적, 문서 작업을 원할하게 하기 위해, 개발자에게 작업 구조를 구체적으로 이해시키기 위해 사용한다.

구조 모델링은 시스템을 위한 뼈대를 나타내며, 그 뼈대에서 각 구성요소들이 어느 곳에 위치하는지를 보여준다. 그래서 Class 다이어그램, Component 다이어그램, Deployment 다이어그램은 구조 모델링의 일부분에 속한다. 이 다이어그램들은 구성요소들과 메커니즘이 어떻게 관련되는지를 보여준다.

하지만 구조 모델링은 시스템이 어떻게 작동하는지에 대해서는 보여줄 수 없다. 보통은 Class Diagram이 구조 모델링에서 많이 쓰인다.

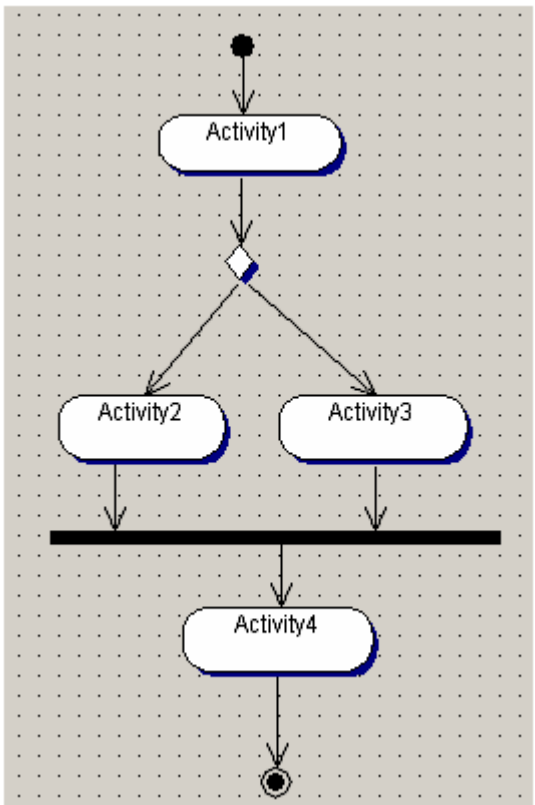
2) 행동 모델링 (Behavioral Modeling)

행동 모델링은 시스템에서 어떤 상호작용이 일어나는지를 보여준다. 행동 모델링은 구조 모델링에서 보여줄 수 없는, 각 구성요소들이 서로 간에 어떤 상호작용이 일어나는지를 나타낸다. 즉, 시스템의 전체적 흐름을 보여주는 것이다. 행동 모델링은 다음과 같은 다이어그램들로 이루어진다.

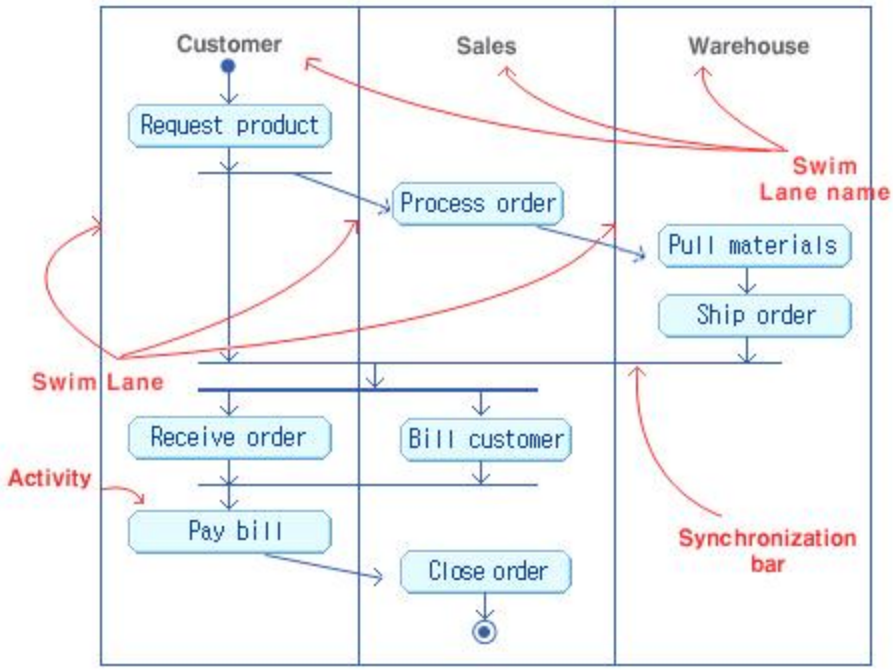
2-1) Activity Diagram

많은 프로젝트에서 요구사항 분석의 일부로서 사용자들의 작업흐름을 모델화하고 분석하는 작업이 필요하다. 우리가 원하는 것은 동시에 일어나도 되는 것은 어떤 것이고, 엄격한 순서에 따라 일어나야 하는 것은 어떤 것인지 찾아내는 것이다. UML 활동 다이어그램 같은 표기법을 사용해서 사용자들이 기술해 준 작업흐름을 기록하는 것이 한 방법이다. 활동(Activity) 다이어그램은 논리적인 처리과정이 있는 모든 대상을 상대로 그려질 수 있다. 특히 비즈니스를 돕는 소프트웨어일 경우 비즈니스 도메인에 대한 이해가 무엇보다도 중요하다.시퀀스에서는 객체간의 메시지를 이용한 상호연동(교류)을 시간의 흐름에 따라 상세히 기술하는 반면 활동 다이어그램에서는 객체간의 교류는 관심이 없다. 처리과정(활동)이 중요할 뿐이다.

이 다이어그램에서는 처리과정에 포함되어질 소시지 모양의 '활동'과 마름모 모양의 '조건' 둥근모양의 '시작점', '종료점' 그리고 긴 얇은 직사각형 모양의 '동시경로'로 구성되어 있다. 아래로 향하는 화살표는 진행과정(방향)을 나타낸다.



* 구성요소



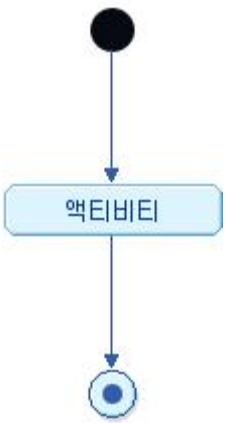
① Things

* Activity : 행위나 작업 (내부적으로 구조를 가지는 단위)

ex) 상품조회, 구매결정, 결재내용입력, 결재자지정....

* Initial State : ●

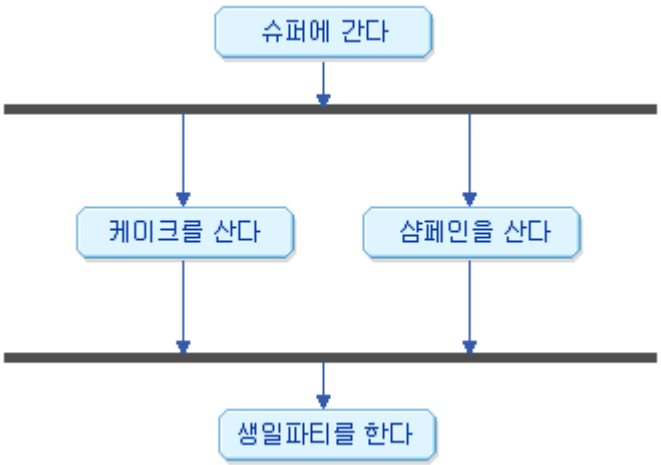
* Final State : ⦿



* Decision(Branch) : ◇

* Synchronization bar : 병렬처리절차가 시작되거나 모이는 지점

ex)



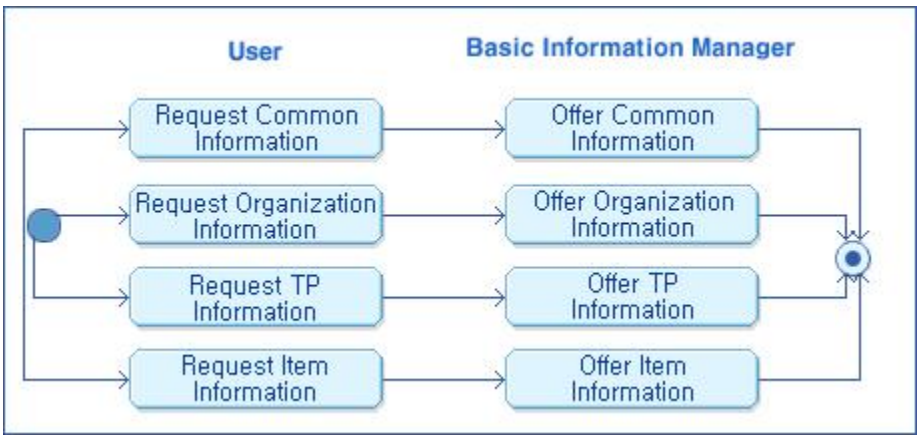
② Relationship

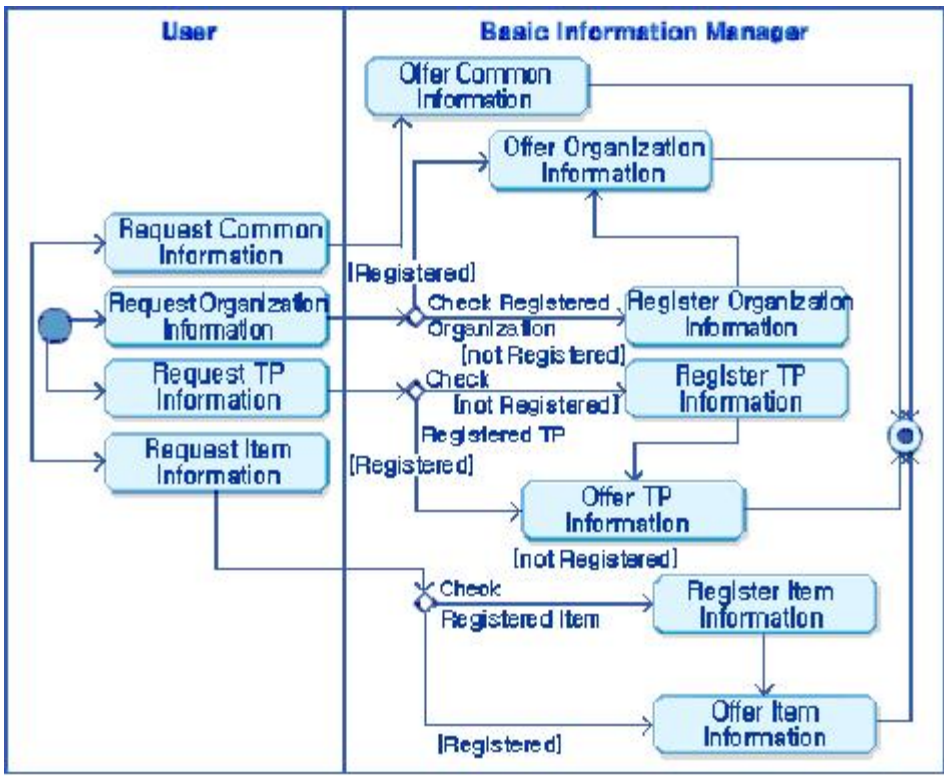
* Transition(전이) : 하나의 액티비티가 행위를 완료하고 다른 액티비티로 처리순서가 옮겨지는 제어흐름 표현

③ Swim lane : 하나의 처리를 구분지음.

* 사례

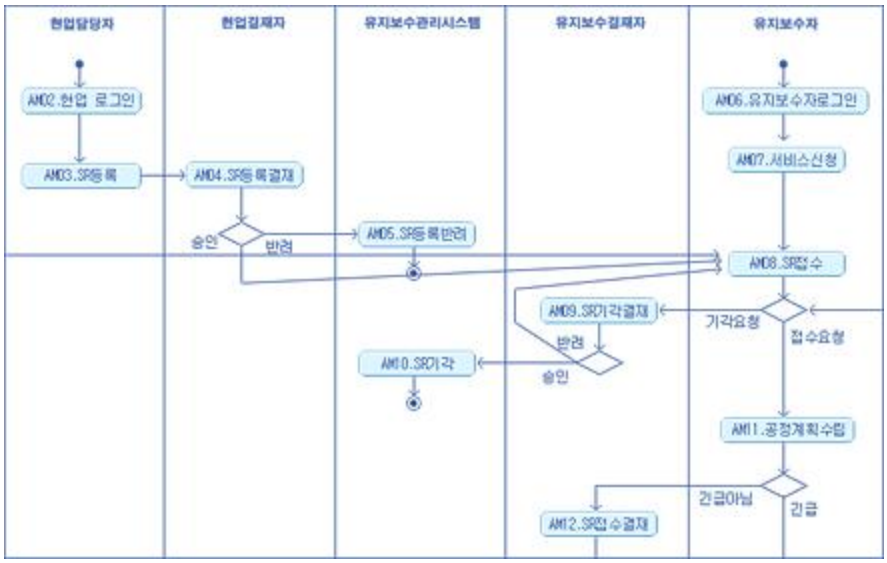
① SCM 시스템의 일반 정보에 대한 Role 액티비티 다이어그램





→ 모든 사용자에게 일반정보를 제공했던 것을 등록여부와 거래품목 등록여부 확인 후 등록된 사용자에게만 일반정보 제공.

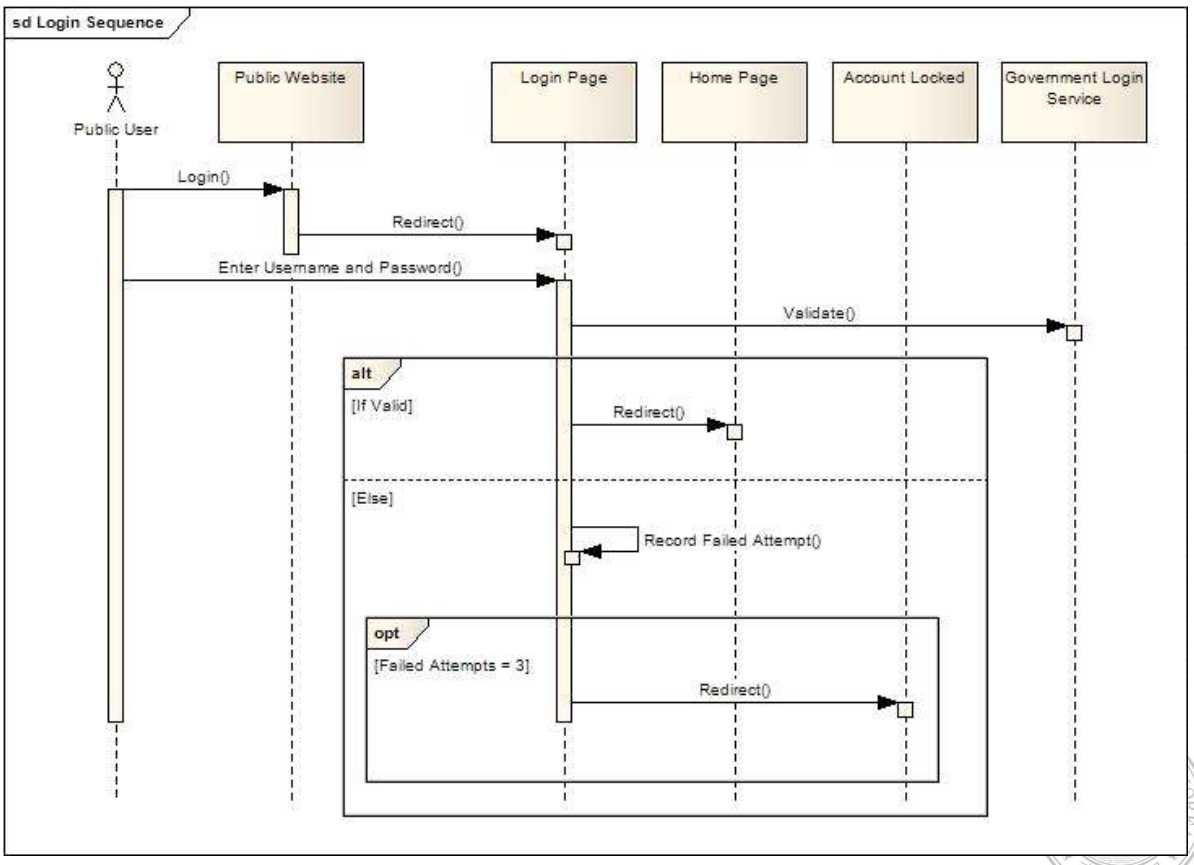
② 프리즘에서 유지보수 절차 프로세스를 정의한 액티비티 다이어그램



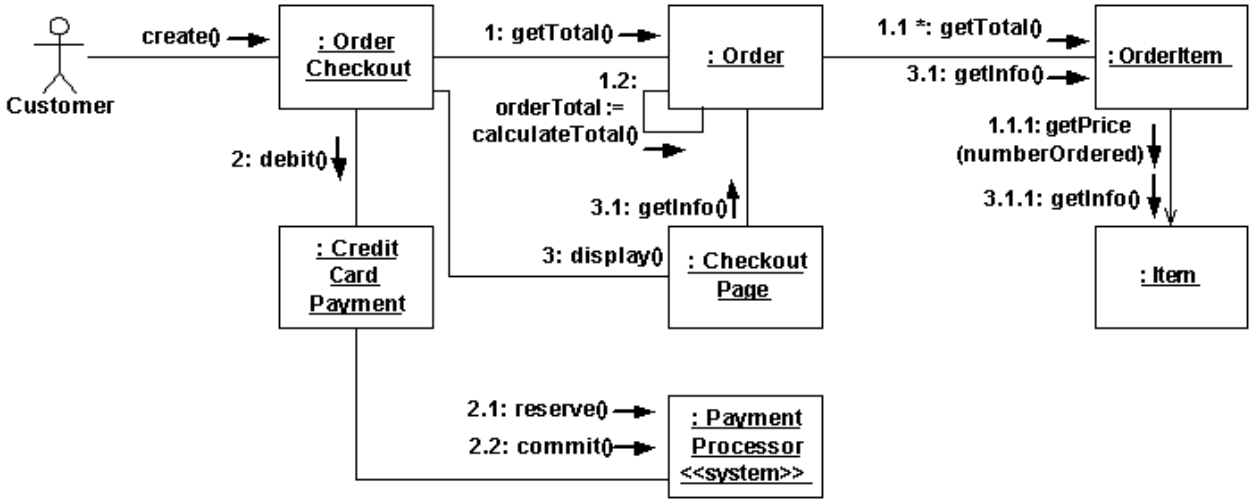
2-2) Interaction Diagram

상호작용 다이어그램은 오브젝트간에 주고받는 메시지의 교환을 모델화하는 것이다. 이러한 상호작용 다이어그램은 2개의 다이어그램을 포함하는데, 시퀀스 다이어그램과 협력 다이어그램이다. 2개의 다이어그램의 외형은 다르나 기본적인 기술방법은 같다. 실제로 몇몇 UML툴에서는 상호 다이어그램이 컨버팅 가능하도록 기능을 제공한다.

< 시퀀스 다이어그램 >



〈 협력 다이어그램 〉



* 구성요소

오브젝트

"오브젝트:클래스명" 이라고 표기하며 오브젝트 다이어그램에서 사용되는 표기와 동일하다,

메시지

어떤 오브젝트가 갖고 있는 메소드의 실행명령을 의미합니다. 메세지 포맷은 아래와 같다.

시퀀스번호[가드조건]*[반복조건] : 리턴값 리스트 := 메시지명(파라미터 리스트)
메시지 예 : [a==b] method("hello", 5)

메시지는 화살표로 표현하며, 화살표 끝부분이 꽉차있으면 동기통신, 화살표 끝부분이 비어있는 단순선으로 그려져 있으면 비동기통신을 의미한다.

시퀀스번호

메세지의 순서를 나타내는 번호

가드조건

메시지를 송신하기 위한 조건. 즉 가드조건이 성립 할 때 메세지가 송신된다.

링크

메시지가 교환되는 오브젝트들을 연결하는 것으로 협력 다이어그램만 그린다.

리턴

메시지가 종료한 것을 나타내며 시퀀스 다이어그램만 그린다. 필수 표기는 아니며 리턴값을 명시하고 싶을때 사용한다.

라이프라인

오브젝트의 생존기간을 나타낸다. 즉 라이프라인이 설정되어 있으면 오브젝트는 메모리에 존재하는 것이다.

활성구간

"제어 포커스"라고 부르며 오브젝트가 활동하고 있는 것을 나타낸다. 활성구간은 겹칠 수 있다.

상호작용 다이어그램의 사용처

오브젝트를 추출 할 때 구조를 나타내는 모델이 없을 경우 상호작용 다이어그램을 만든다는 것은 오브젝트를 자유롭게 만들어 나간다는 의미한다. 이러한 상호작용 다이어그램을 구현하고자 하는 모든 시나리오에 적용하면 필요한 오브젝트를 모두 열거할 수 있다. 이런경우 상호작용 다이어그램 중에서 협력다이어그램을 사용하는 것이 좋다.

로직 확인 할 때

상호작용 다이어그램을 사용하면 1개의 시나리오의 구현과 로직을 한눈에 보기에 좋다.

클래스 다이어그램을 확인 할 때

이미 구조적 모델, 특히 클래스 다이어그램이 있을 경우 상호작용 다이어그램을 통해 해당 클래스 다이어그램이 시스템의 요구사항을 제대로 만족시키는지 확인 할 수 있다.

책무 밸런스를 확인 할 때

클래스 다이어그램에서도 알 수 있지만 보다 알기 쉬운 것은 상호작용 다이어그램을 사용하여 확인하는 것이다. 혹시 책무가 특정 클래스에 집중되는지를 확인하여 균형 있게 배치되도록 수정할 수 있도록 도와준다.

* 주의사항

시나리오를 분명히 한다.

시나리오 없이 머릿속의 생각만으로 그리면 무엇을 만족하는 다이어그램이 완성되는지 명확하지 않다. 그래서 반드시 시나리오를 준비해야 한다.

1다이어그램 1시나리오

상호작용 다이어그램은 1개의 다이어그램이 1개의 시나리오에 대응하는 것이 기본이다.

긴 시나리오는 다이어그램을 분할한다.

시나리오가 길 경우 시퀀스 다이어그램을 분할해서 그리는 편이 이해가기 쉽다.

메시지명은 받는 측의 관점에서 붙인다.

메시지명은 받는 측의 관점에서 붙이는 것을 권장한다. 사실 그다지 이상할 것은 없으나 클래스 다이어그램을 만들어 보면 어느 쪽이 나은지는 알 수 있다.

모르는 오브젝트에 메시지를 보내지 않는다.

오브젝트간의 상호작용을 구현할 때 메시지를 보내려면 메시지를 보내기 전에 보내는 측 오브젝트는 받는 측 오브젝트를 어떠한 방법으로든 알고 있어야 한다.

생성과 소멸을 의식한다.

오브젝트의 생성/소멸을 의식하지 않고 그리면 실제 코드 구현시 메시지를 보냈는데 오브젝트가 미생성이든가 혹은 이미 소멸했거나 또는 소멸해야하는 오브젝트가 남아 메모리 부족이 일어나는 등 문제가 발생할 수 있다.

상호작용에 참가하지 않는 오브젝트는 그리지 않는다.

협력 다이어그램을 그리기 전에 오브젝트 다이어그램을 만들었다면 자기도 모르는 사이에 상호작용에 참여하지 않는 오브젝트를 그리게 되는 경우가 있다.

오브젝트명을 붙인다.

오브젝트에 제대로 이름을 붙이는 습관을 갖는 것이 중요하다. 오브젝트명으로 역할명을 붙이거나 리턴값명과 오브젝트명을 붙이면 식별하기가 쉽다.

파라미터도 쓴다.

메시지명만 쓰고 파라미터 쓰는 것을 잊어버리기 쉽다. 파라미터를 제대로 써 놓으면 데이터가 어떻게 전달되는지 알 수 있고, 메시지의 의미도 보다 알기 쉽기 때문에 가능하면 메시지에 파라미터도 추가하는 것이 좋다.

레이아웃을 고안한다.

협력 다이어그램을 그릴 때 주의사항이다. 클래스 다이어그램과 비교할 경우가 많기 때문에 클래스 다이어그램의 레이아웃과 비슷하게 배치하면 비교하기가 쉽다.

상세함에 주의한다.

상호작용 다이어그램은 아주 상세하게 그릴 수 있는 다이어그램이다. 따라서 작성자가 어디까지 요구하고 있는지를 이해하고 있어야 한다.

3) 설계 모델링 (Architectural Modeling)

설계 모델링은 시스템 구조에 대한 전반적인 형태를 표현한다. 설계 모델링은 구조 모델링과 행동 모델링을 모두 포함하고 있다. 즉, 전체 시스템의 윤곽을 정의한다.