Introduction to UML

소프트웨어 모델링

유준범 교수님

201111397 황정아

201111341 김성민

201111379 이한빈

1. Subject : UML (Unified Modeling Language) and UML Tools
2. Outline
    1. Subject
    2. Outline
    3. What is UML?
        A. 정의
        B. 배경
        C. RUP
        D. 개발
        E. 특징
        F. 구성 (Diagrams)
    4. UML Tools
        A. Star UML
        B. Amateras UML
        C. Object Aid
        D. Rational Rose
        E. Omondo
    5. Conclusion
    6. Reference

3. What is UML?

A. 정의

UML(Unified Modeling Language)는 요구분석, 시스템 설계, 시스템 구현 등의 시스템 개발 과정에서, 개발자간의 의사소통을 원활한 하기 위하여 표준화한 모델링 언어이다.

B. 배경

객체지향 개발 방법을 주장한 Grady Booch의 Object-Oriented Design(OOD)와 James Rumbaugh의 Object Modeling Technique(OMT) 그리고 Ivar Jacobson의 Object-Oriented Software Engineering(OOSE) 방법론이 유명하였다. 이렇게 많은 방법론들이 존재하다 보니 모델을 표현하기 위한 동일한 기호, 언어를 사용하는 것에 대한 필요성을 느끼게 된다.

C. RUP (Rational Unified Process)?

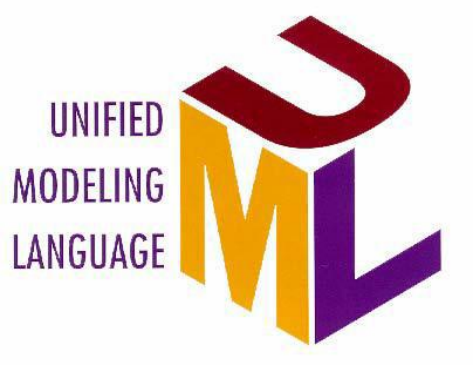
UML은 객체 지향으로 시스템을 개발하기 위한 모델링 표기법이다. 하지만 UML뿐만 아니라 객체 지향 방법론을 사용해야 한다. 많은 객체 지향 방법론이 존재하지만 이 중 가장 부각되고 있는 것이 RUP이다. 무엇보다 RUP는 Rational의 소프트웨어군을 이용한 개발 방법론으로서 이론 뿐만 아니라 구체적인 솔루션이 동반된다는 강점을 지니고 있다. 즉, Rational의 도구들과 RUP에 맞춰서 UML을 사용하여 개발해야 한다.

< RUP 개발 공정 >

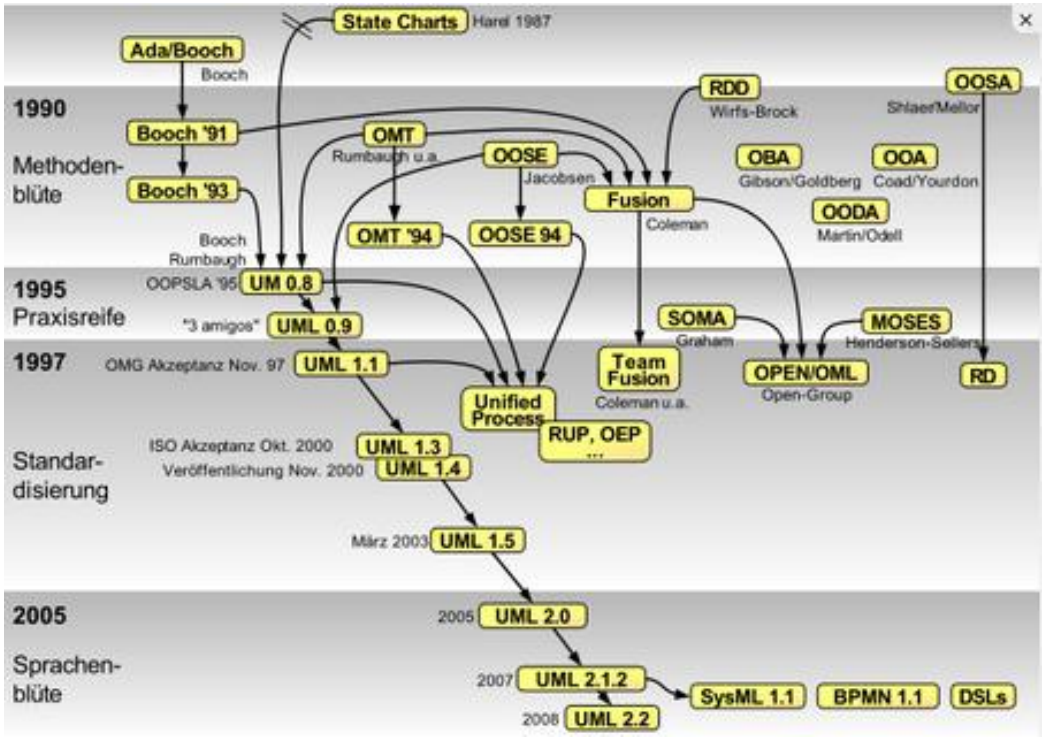
RUP의 개발 공정은 크게 두 축으로 나눠 볼 수 있다. 우선 그림의 가로축으로 시간의 흐름에 따른 네 가지 단계(Phases)로 구분할 수 있고, 세로축의 9가지 웍플로우(Workflow)로 나뉜다. 웍플로우는 컴포넌트처럼 작업의 성격에 따라 일을 분리한 것이다.

기존의 방법론이 도입기에는 주로 타당성 검증 등을 하고, 분석 및 설계, 구현, 검증 및 배포와 같은 식으로 일원적인 관점에서 개발을 했다면 RUP는 이차원적인 관점을 갖는다. 도입기라고 할 수 있는 도입(Inception) 단계에서는 주로 비즈니스 모델링(Business Modeling)을 수행하지만 이를 위해 상당량의 요구사항 분석을 수행해야 하고, 개발 프로젝트의 타당성이나 위험도 등의 검증을 위해 프로토타입을 만들어 본다든가 하는 구현도 일부분 수행하게 된다. 마찬가지로 향후 프로젝트를 정교하게 발전시켜가는 정련(Elaboration) 단계에서도 요구사항 수집과 분석 설계는 물론 도입 단계에서 만들어진 비즈니스 모델링(Business Modeling)을 검증하고 더욱 정교하게 수정하는 일도 계속한다. 프로젝트 관리자에 의해서 이러한 적절한 조합이 계획되는데 이를 이터레이션(Iteration)이라고 한다.

D. 개발



i. Booch와 Rumbaugh의 방법론을 통합시킨 Unified Method 0.8로 시작되었고, OOSE와 다른 기능들을 통합하면서 UML0.9가 탄생하였다. 그리고 1997년 9월 UML 1.1이 표준으로 채택되면서 꾸준히 수정, 보완되고 있다.



ii. UML은 발전 될수록 Model-driven Development(MDD)에 필요한 고급 자동화를 지원해서 모델의 모호함과 부정확성을 제거하고, 프로그램이 모델의 변형 및 조작을 가능하게 한다. 향상된 언어 구조를 갖고 있어서 사용자가 언어에 보다 쉽게 전근 할 수 있으며, 도구 간 내부 작동을 활성화 할 수 있는 모듈식 구조를 갖고 있다. 규모가 큰 시스템의 모델링 향상을 위해서 시스템이 더 복잡해지고 있으므로, 이를 지원하기 위해 유연한 새로운 계층 기능이 언어에 추가되어 소프트웨어 모델링을 지원한다. 확장 메커니즘을 이용해서 기본적인 언어가 보다 정확하고 단순해지도록 정리되어서 도메인 스펙의 특성화 지원 향상을 해준다.

그리고 다양한 모델링 개념들의 정리, 개념화, 정의를 통해서 보다 단순하고 일관성 있는 언어고 중복된 개념을 제거하고, 많은 정의들을 정리했으며 텍스트 정의와 예제를 추가 했다.

E. 특징

i. UML은 모델링 언어로써 표기법만을 제시하고 소프트웨어 개발에 사용하기 위한 여러 가지 다이어그램들을 정의하고 다이어그램들의 의미에 대해 정의하고 있다.

ii. UML은 다이어그램을 통해서 소프트웨어 개발과정을 시각화 형태로 제공하고, 개발자와 고객 또는 개발자들 간의 의사소통을 원활하게 할 수 있도록 해준다.

iii. 다양하고 일관성 있는 표현방법을 제공하여 확장성이 우월한 것 규모에 관계없이 소형부터 대형프로젝트까지 모두 적용 할 수 있다.

iv. 사용자에게 간단하고 표현이 풍부한 시각화 언어를 제공해주고 특정 한 개의 개발/방법론에 얽매이지 않는 개방적, 독립적인 표기체계이다.

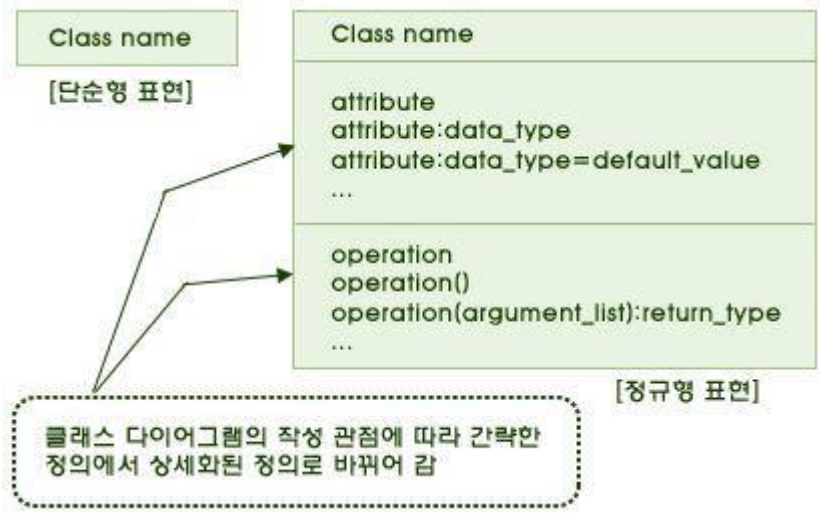
F. 구성

모델	계층	다이어그램	설명
정적	구조 다이어그램	클래스(Class)	클래스 구조를 나타내는 다이어그램
		오브젝트(Object)	인스턴스 구조를 나타내는 다이어그램
		컴포넌트(Component)	시스템을 구성할 컴포넌트들의 구조를 나타내는 다이어그램
		컴포지트(Composite)	시스템을 실행할 때의 구조를 나타내는 컴포지트 구조를 표현
		패키지(Package)	내부에 모델 요소를 포함할 수 있는 패키지 구성을 표현
		디플로이먼트(Deployment)	시스템의 물리적은 하드웨어 구성을 나타내는 다이어그램
동적	인터액션	시퀀스(Sequence)	오브젝트 사이의 메시지 교환을 나타내는 다이어그램
		타이밍(Timing)	한 상태에서 객체가 얼마나 오랜 시간을 지체하는지 명시

		커뮤니케이션 (Communication)	오브젝트 사이의 메시지 교환을 나타내는 다이어그램
	인터액션 행위	인터액션(Interaction)	오브젝트 간의 메시지 교환을 액티비티 다이어그램과 같은 형태로 나타내는 다이어그램
	인터액션 내 행위	액티비티(Activity)	액티비티의 실행순서나 실행조건, 실행자의 관계를 표현
		스테이트머신(State-Mac.)	상태전이와 상태 전이에 따른 액션을 나타내는 다이어그램
요구 분석		유즈케이스(Use-Case)	시스템이 제공할 서비스와 그 이용자의 관계를 표현

i. Class Diagram

클래스 다이어그램은 클래스를 식별하고 그 관계를 정의하는 유용한 방식을 제공하고 시스템을 이해하는데 용이하게 해준다. 또한, 오퍼레이션과 속성을 정의함으로써 SW 시스템을 설계하고 일관된 형식으로 SW 시스템을 분석, 설계하는



방식을 제공한다.

- Conceptual level에서는 단순한 관계를 도출하는데 중점을 두고 업무 과점의 class 들만 도출하고 구현에 관련된 시각은 최대한으로 배제한다.
- Specification level에서는 구현 관점을 살려 모델링을 수행하고 코딩에 대한 관점을 배제, 클래스의 속성과 오퍼레이션을 상세히 정의하고, 구체적인 플

랫폼, 개발언어의 특성을 반영하지 않는다.

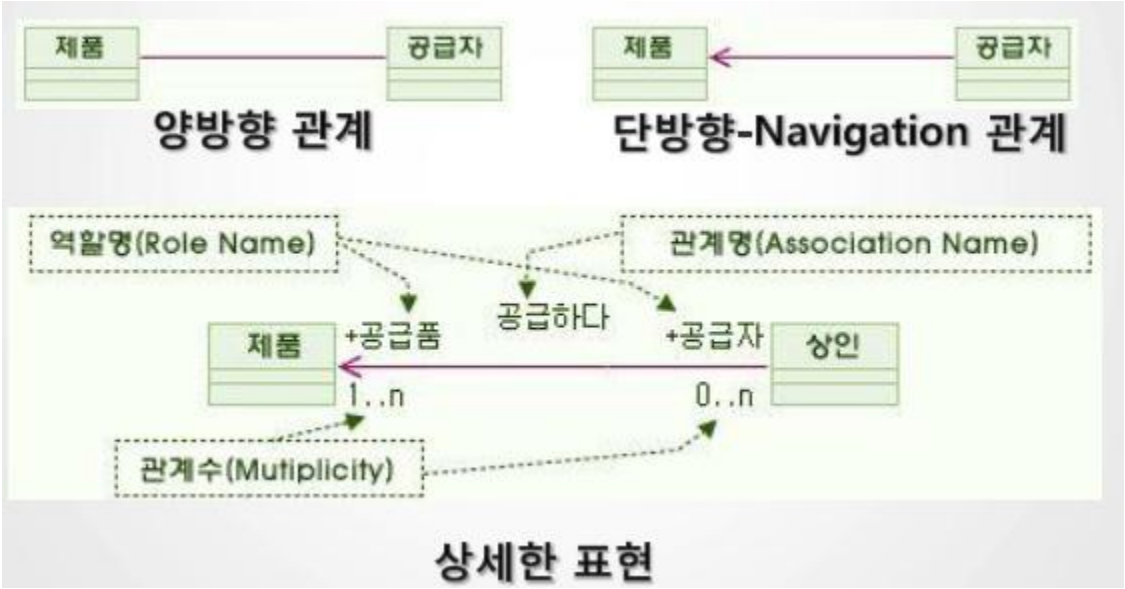
- Implementation level에서는 언어와 개발플랫폼이 가진 특성 및 제한 사항을 반영하여 정의된 class를 보고 정해진 개발언어로 개발자가 코딩을 하기에 충분한 정보를 모두 표현 한다.

< Class Diagram Relations >

- 연관관계(Association)

  연관관계는 클래스간 일반적인 협력 관계가 있을 경우 정의를 하고 객체들 사이에 존재하는 공통의 성질, 의미를 갖는 링크들의 집합을 표현한다.

  두 클래스가 Association관계가 있다면 한쪽 객체에서 다른 객체를 **참조** 할 수 있음을 의미한다.



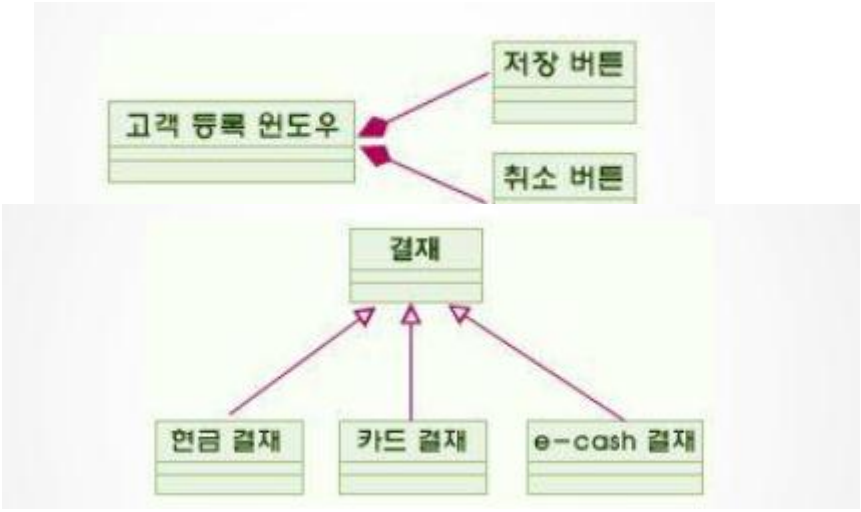
- 집합연관(Aggregation)

  집합연관은 클래스간에 "전체-부분(whole-part)"의 관계가 있을 경우 정의하고 Aggregation관계는 클래스 각각이 독립적인 생명주기를 갖는다. 그리고 Aggregation 관계는 Association 관계의 일종이다.

- 합성연관(Composition)

  합성연관은 Association관계의 일종이며 Aggregation 관계와 유사하게 두 클래스 간에 "부분-전체(part-of)"의 관계가 있을 경우 정의한다.

Composition 관계는 부분의 생명주기가 전체의 생명주기에 종속적인 관계라는 것에서 Aggregation과 차이를 보인다.



- 일반화관계(Generalization)
  일반화 관계는 두 클래스는 일반화-특수화 관계가 있을 때 정의(is-a관계)되고 **상속(inheritance)**의 특성을 갖는다.

- 의존관계(Dependency)
  의존관계는 한 쪽 클래스가 실행 도중 다른 쪽 클래스의 실행을 요청하는 경우에 정의되고 클래스간의 사용 관계를 표현한다. Association 관계에 비해 훨씬 종속적인 성향을 갖는다.
  Association은 존재하는 단순히 다른 객체를 참조하고 실행을 의뢰하지만, Dependency관계는 다른 객체를 생성하고, 소멸시키는 등 보다 종속적인 관계에 대해 정의한다.

< Class들 간 관계들 >

- Zero or more(0..n) : 클래스 B의 인스턴스 하나에 관계된 A 인스턴스가 없거

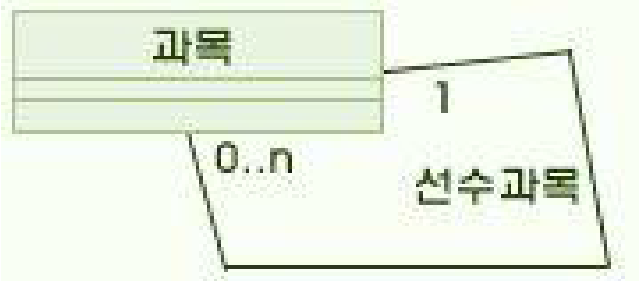
나 여러 개있는 경우

- One to Ten(1..10) : 클래스 B의 인스턴스 하나에 관계된 A 인스턴스가 1개보다 많고 10개 보다 적음

- Exactly two, three or five(2,3,5) : 클래스 B의 인스턴스 하나에 관계된 A 인스턴스가 2개 혹은 3개 혹은 5개

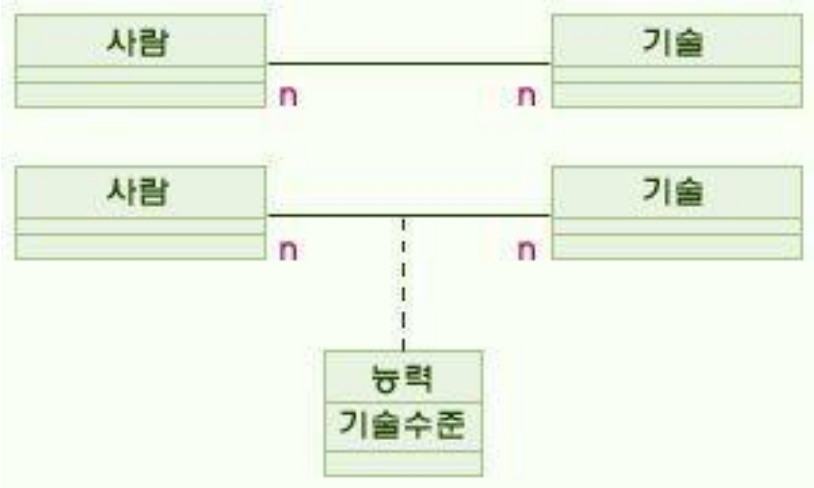
- 다중 연관관계(Multiple Association) : 두 클래스 간에 두 가지 이상의 Associataion이 존재하는 경우

- 재귀 연관관계(Reflexive Associataion) : 같은 클래스끼리 맺어지는 관계가 존재하는 경우



- Qualifier 연관관계 : 관계수가 복잡할 때 사용(many/one-to-many)

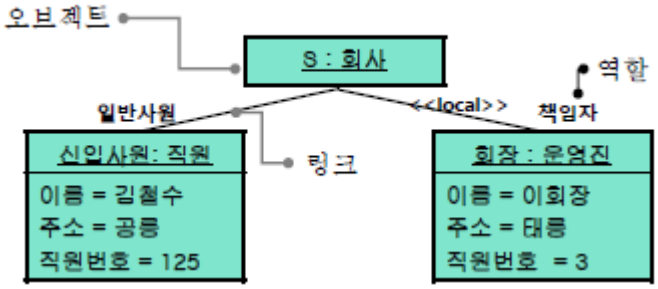
- 연관 클래스(Associataion Class) : Association 관계가 고유의 속성이나 오퍼레이션이 필요 할 경우에 정의



ii. Object Diagram

오브젝트 다이어그램은 오브젝트를 대상으로 한 그림이다. 모델의 '어느 일순간'을 파악해서 파악 시점의 오브젝트 구조를 나타내며 오브젝트와 오브젝트 간의 관계를 나타낸다. UML 1.x에서는 클래스 다이어그램의 일부분이어서 제대로 정의되지 않다가 UML 2.0부터 정식 다이어그램이 되었다.

스테레오 타입	설명	역할
<<association>>	관련으로써 유지보수하고 있는 오브젝트의 링크를 나타낼 경우	연관관계
<<parameter>>	조작의 파라미터로 전달된 오브젝트를 나타날 경우	의존관계
<<local>>	조작 내의 로컬 오브젝트로 만들어진 오브젝트의 링크를 나타낼 경우	의존관계
<<global>>	글로벌 오브젝트의 링크를 나타낼 경우. 의존 관계를 나타냄	의존관계
<<self>>	자기 자신으로의 링크를 나타낼 경우	명시안함



※ 오브젝트 이름 또는 클래스 이름은 생략 가능하다.

오브젝트 이름 생략 클래스 이름 생략

- 구성요소

  오브젝트는 인스턴스를 나타내며 클래스 표기와 비슷하지만 속성만 나타낼 수 있고 메소드는 나타낼 수 없다.

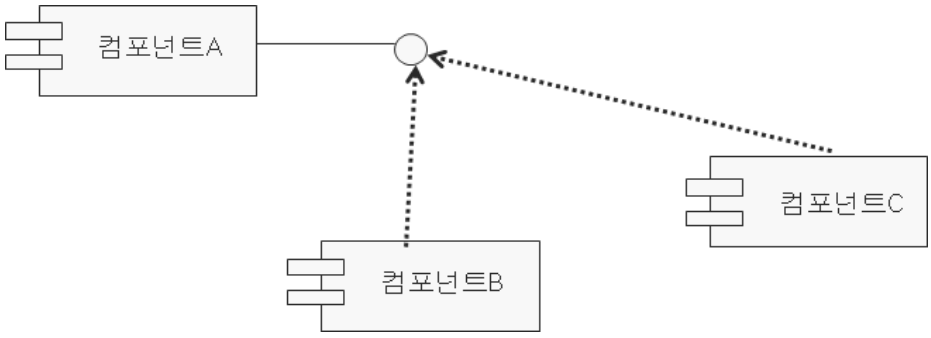
  링크는 오브젝트간 접속을 나타내며 클래스 다이어그램에서는 연관이나 의존 관계에 해당하지만, 링크만으로는 어떤 오브젝트에 속하는지 알 수 없다. 그래서 스테레오 타비을 붙여 링크의 종류를 나타낼 수도 있다.

  스테레오 타입은 링크의 종류를 나타낸다.

  역할명은 링크에 부여하는 것으로 오브젝트가 다이어그램 내에서 어떤 역할을 담당하는지를 나타낸다.

  오브젝트 다이어그램은 객체지향에서 필요한 '오브젝트'를 직접 나타낼 수 있는 다이어그램이기 때문에 제대로 작성 할 수 있는지에 따라, 객체지향에 대해 얼마나 잘 이해하고 있는지를 판단 할 수 있다.

iii. Component Diagram



컴포넌트는 교환 가능한 시스템의 구성 부품 중의 하나로, 내부에 구현을 포함

하고 외부에 인터페이스를 공개 하고 있는 것 이며 1개의 컴포넌트는 1개 이상의 클래스, 인터페이스, 컴포넌트로 구성되어 있다.
객체 지향 개발에서는 전체 시스템의 클래스들을 기능적인 연관성을 고려하여 결합력이 강한 클래스들을 그룹으로 묶어 새로운 단위의 하위 시스템을 구성 할 수 있고 이것을 컴포넌트라고 한다.

- 컴포넌트에서의 인터페이스 : 인터페이스는 컴포넌트가 외부에 공개하고 있는 것이고 컴포넌트가 제공할 기능에 대한 명세이며 컴포넌트의 오퍼레이션은 그 컴포넌트의 인터페이스를 통해서만 사용 할 수 있다.

- 컴포넌트의 장점 : 기존의 함수, 클래스 등에 비해 큰 규모이기 때문에 재사용을 하는 경우 효과가 더 커지게 되고 매우 강한 수준의 정보 은닉 개념을 지원한다. 그리고 기존 컴포넌트를 수정하는 것이 아니라 새로운 컴포넌트로 대체하는 것도 가능하다.

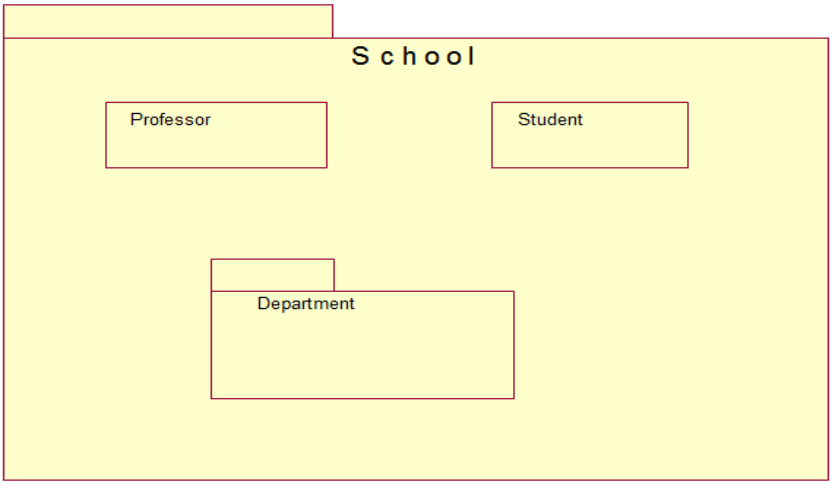
- 컴포넌트 다이어그램은 물리적인 것을 모델링하기 위한 다이어그램이며 기본설계 및 상세설계 단계에서 작성하며 컴포넌트 사이의 관계를 나타내기 위한 다이어그램이다.

iv. Composite Diagram

컴포지트 다이어그램은 구성 요소들 간의 계층적 연관 관계를 보여주는 다이어그램으로써 한 구성 요소가 어떠한 내부 구조를 가지고 있는지, 각 내부 구조의 구성 요소들이 어떠한 연관 관계를 가지면서 계층구조를 형성하는 지 등을 정의한다.
컴포지트 다이어그램에서, 포트는 클래스와 환경 간의 또는 클래스와 내부 파트 간의 상호작용 지점을 정의한다. 포트를 사용하여 클래스가 제공하고 환경에서 필요로 하는 서비스를 지정할 수 있다.

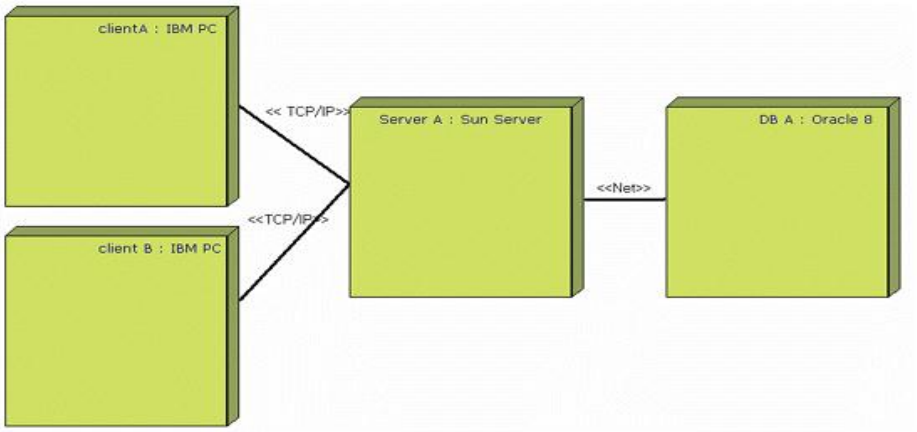
v. Package Diagram

패키지 다이어그램은 클래스와 같은 여러 모델 요소들을 그룹화 시킬 수 있는 수단이고 패키지 내에 다른 패키지를 포함 할 수 있다.
모든 구성요소는 단지 하나의 패키지에만 포함 될 수 있고 각 패키지는 하나의 네임스페이스를 구성하고 이 의미는 두 개의 모델요소가 각기 다른 패키지에 속한다고 할 때 이들이 동일한 이름을 갖는 것을 허용한다는 것을 뜻한다. 패키지를 제거하면 패키지 내 모델요소도 함께 제거된다.



vi. Deployment Diagram

디플로이먼트 다이어그램은 실제 시스템의 물리적인 모습을 보여주는 다이어그램으로써 시스템을 구성할 때 각 구성 요소를 물리적으로 어떻게 배치할 것인가를 정의한다.



동적 모델은 인터액션, 인터액션 행위, 인터액션 내 행위로 나뉘며

시퀀스

타이밍

커뮤니케이션

인터액션

액티비티

스테이트 머신

이렇게 6가지로 나뉘게 된다.

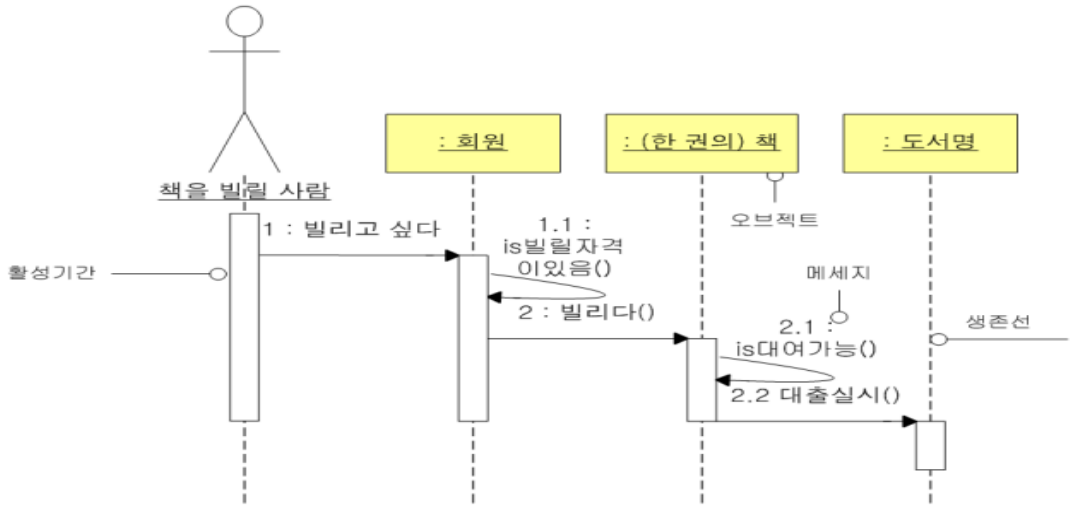
vii. Sequence Diagram

시퀀스 다이어그램은 해결해야 할 문제가 주어진 상황에서 그 문제를 해결하기 위해 필요한 객체를 정의하고 객체간의 동적인 상호관계를 시간 순서에 따라 정

의함으로써 주어진 문제를 해결한다. 시퀀스 다이어그램은 어떻게 하기로 정의된 것을 실제로 그렇게 되도록 실현 하는 것이다.

시퀀스 다이어그램의 수직방향은 시간흐름을 나타내고 수평방향은 상호작용을 하는 객체들을 위치하고 시간개념이 중시되고 시간이 위쪽에서 아래쪽으로 증가하는 것을 전제로 하기 때문에 아래쪽에 표현된 메시지는 위족의 메시지보다 나중에 발생했음을 알 수 있다.

다이어그램의 상호작용을 유발하는 객체는 왼쪽에 위치시키고 종속적인 객체는 오른쪽에 위치시킨다.



시퀀스 다이어그램은 클래스 다이어그램을 검증하고 객체의 오퍼레이션과 속성을 상세히 정의하고 유즈케이스를 실현하고 프로그래밍 사양을 정의한다.

viii. Timing Diagram

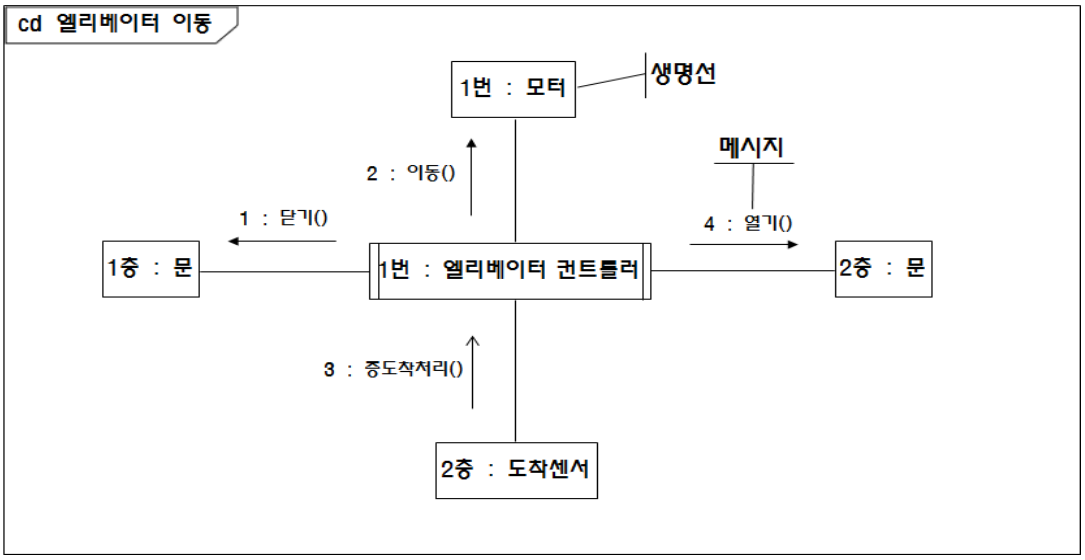
시퀀스 다이어그램과 스테이트 머신 다이어그램을 합쳐놓은 다이어그램으로써 시간에 따른 각 객체의 상태와 그 상태를 변경시키는 방아쇠 역할을 하는 메시지를 정의한다.

ix. Comunication Diagram

커뮤니케이션 다이어그램은 상호작용 다이어그램의 일종으로 여러 객체/컴포넌트들 사이의 상호작용을 표현하기 위해 사용하고 관계를 명시적으로 표현한다.

상호작용에 참여하는 객체/컴포넌트 간의 관계 유무를 표현/파악한다.

시스템 클래스를 파악한 후, 객체들 간에 발생할 상호 작용을 나타내기 위한 것과 복잡한 형태를 가진 오퍼레이션의 실현을 나타내기 의해 작성한다.



x. Interaction Diagram

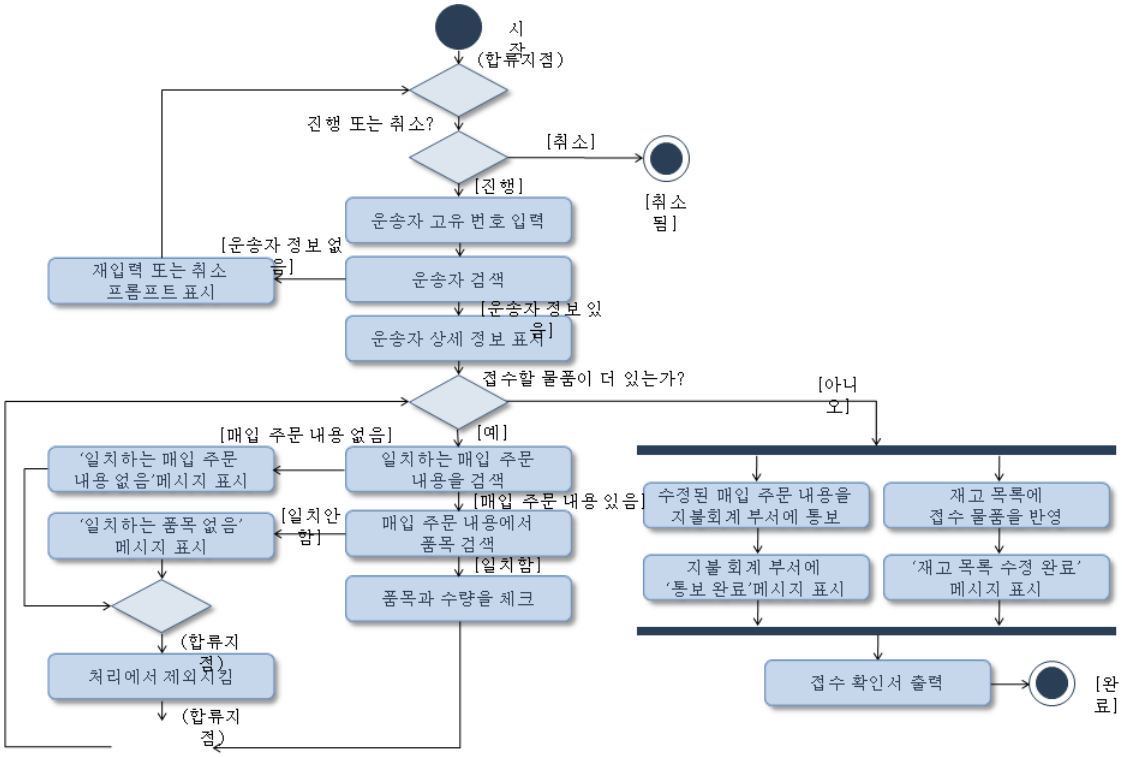
인터액션 다이어그램은 시스템의 동적인 측면을 모델링하는데 사용되어서 시스템이 어떻게 수행되는지 시각화 하는 도움을 준다.

객체의 잡합이 액터와 어떻게 인터액션이 이루어지는지 나타내는 것이 목적이다.

xi. Activity Diagram

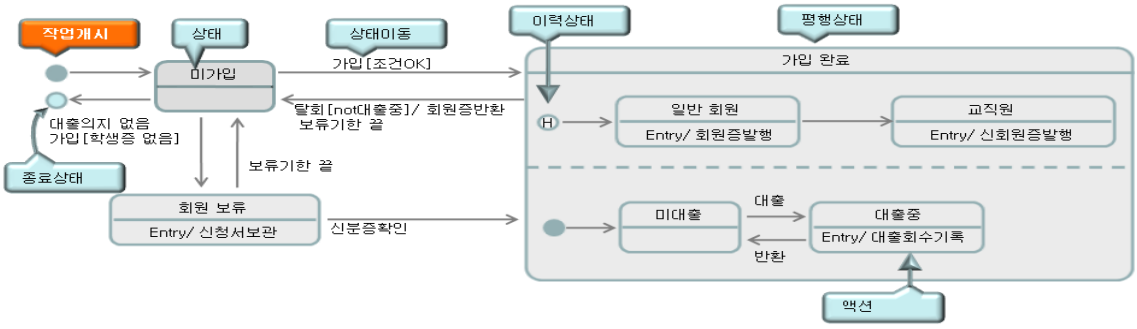
액티비티 다이어그램은 시스템영역에서 다양하게 존재하는 각종 처리 로직이나 조건에 따른 처리 흐름을 순서에 따라 저의 한 모델이다. 활동과 활동간의 제어의 흐름을 보여주는 흐름도이다.

하나의 활동에서 다음 활동으로 순서가 바뀌면서 처리되는 과정을 표현하기 때문에 순서/분기/처리절차의 표현을 필요로 하는 대상에 대해 제한 없이 적용이 가능하다.



xii. 스테이트 머신 다이어그램

스테이트 머신 다이어그램은 상태를 나타내기 위한 다이어그램으로 대상이 어떠한 상태인지 찾아내서 각각의 상태에 대해 할 수 있는 것과 할 수 없는 것을 명확히 하고, 상태가 어떠한 이벤트에 의해 변해가는지 나타내는 것이다.



상태는 개시상태, 종료상태, 상태, 상태이동, 평행상태, 이력상태, 포크, 조인등이 있다.

- 개시상태 : 상태 이동의 시작 지점이며 시작 시의 상태 이동은 시스템 부팅이나 오브젝트 생성을 나타내고 개시 상태는 스테이트 머신 다이어그램 안에서 반드시 1개는 있어야 한다.

- 종료상태 : 상태 이동의 종료 지점으로 종료로의 상태 이동은 시스템 정지나 오브젝트 소멸을 나타내고 종료 상태에서는 상태 이동은 없고 생략 가능하다.

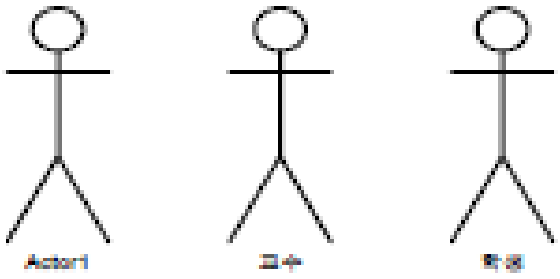
- 상태 : 대상의 라이프 사이를(발생부터 소멸까지)안의 어떤 상황을 나타내며 상태 안에는 액션이나 내부 이동을 정의 할 수 있다.
- 상태이동 : 상태에서 상태로의 이동을 나타내고 지정된 이벤트가 발생 했을 때 상태가 이동한다.
- 평행상태 : 상태 안에는 여러 개의 상태 이동을 정의 할 수 있고 상태 안에 상태 이동을 포함한 상태를 혼합 상태, 포함되어 있는 상태를 서브 상태라 한다. 혼합 상태에서 여러 개가 동시에 동작하는 상태 이동을 평행 상태라 고 한다. 평행상태에 있는 서브 상태들의 상태 이동은 독립적으로 일어난다.
- 이력상태 : 혼합 상태에서 다른 상태로 이동 한 후, 다시 혼합 상태로 되돌아 왔을 때 어떤 서브 상태에 있었는지 저장한다.
- 포크 : 상태 이동을 분리
- 조인 : 머지, 포크된 이동이 전부 종료 할 때까지 기다린다.

xiii. Use Case Diagram

유즈케이스 다이어그램은 개발자가 아닌 사용자 입장에서 시스템을 보았을 때, 시스템이 제공해야 할 기능을 나타내야 한고 모든 요구사항을 만족 하는지를 확인하기 위해 사용한다.

- 액터

  액터는 시스템의 외부에 존재 하면서 시스템과 상호작용을 하는 모든 것(사람, 기계, 다른 시스템)이며 사람 심볼로 표시하며 액터명은 단일 명사를 사용한다.

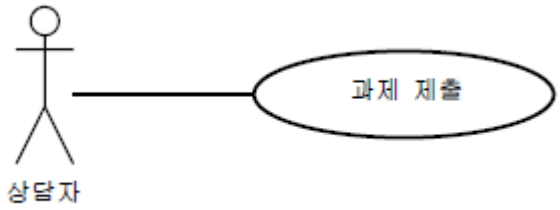


- Use Case

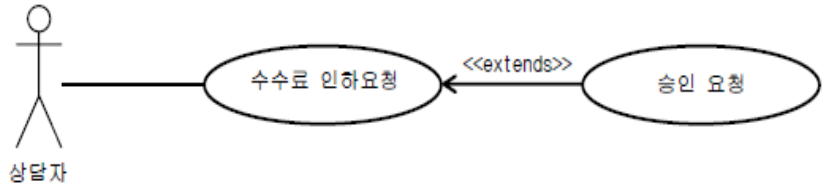
  유즈케이스는 액터의 요구에 의해 시스템이 어떻게 사용될 것인가를 표현하고 고객의 입장에서 본 기능적인 요구사항을 나타내고 그 자체로 완전하고 하나의 의미를 갖는 업무처리단위이다.

- 유즈케이스의 관계

  ✓ 연관관계(Association) : 상호작용하는 액터와 유즈케이스 간의 관계를 표현하고 액터는 하나 이상의 유즈케이스와 연관 될 수 있으며 유즈케이스 또한 하나 이상의 액터와 연관 될 수 있다.



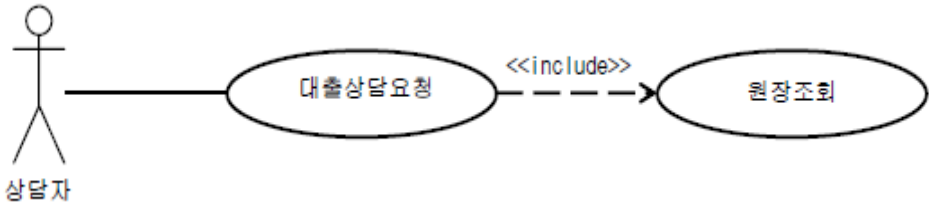
  ✓ 확장(Extend)



:

선택적인 행위관계이며, 어떤 특수한 조건에서만 수행된다. 수행순서는 화살표 반대 방향으로 발생한다.

  ✓ 포함(Include)



필수적인 행위 관계, 수행순서는 화살표 방향으로 발생하며 하나의 유즈케이스가 다른 유즈케이스를 반드시 수행하는 경우에 사용 된다.

4. UML Tools

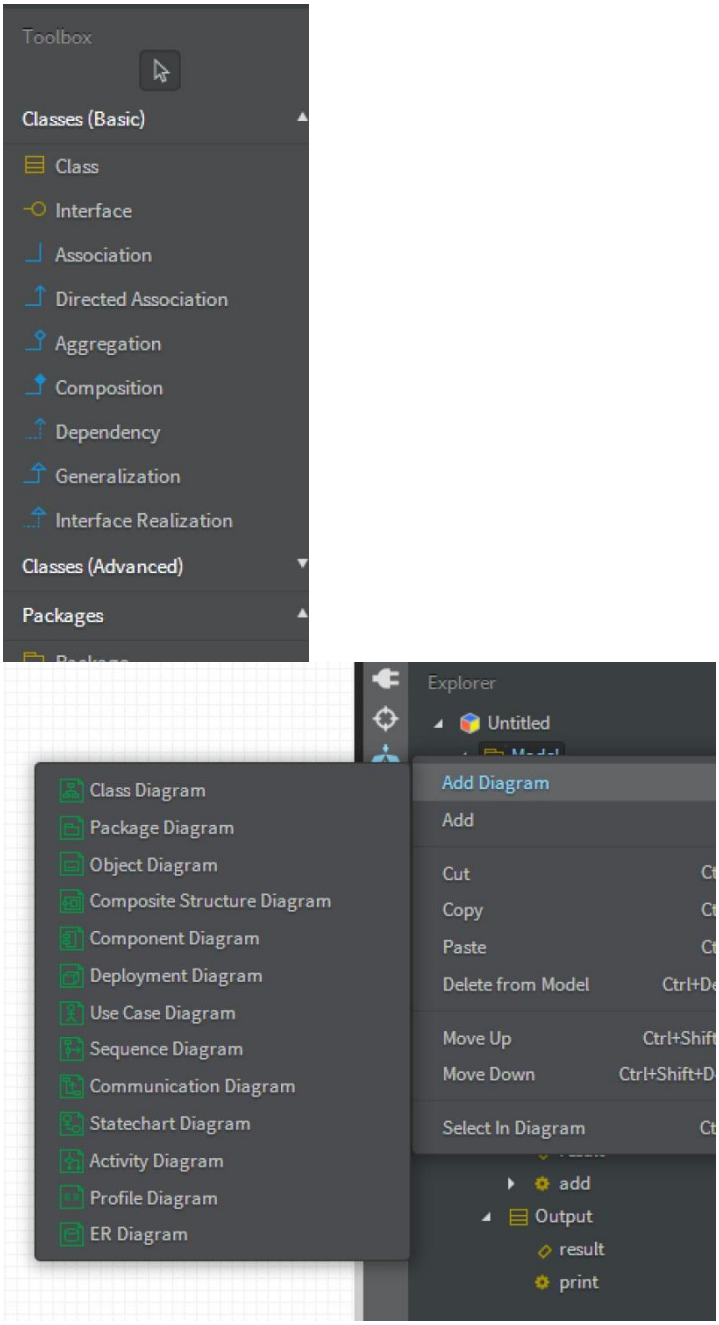
Name	Creator	Platform / OS	First public release	Latest stable release	Open source	Software license	Programming language used
AgileJ StructureViews	AgileJ	Cross-platform (Java)			No	Commercial	Java
Altova UModel	Altova	Microsoft Windows	2005-05		No	Commercial	Java, C#, Visual Basic
ArgoUML	Tigris.org	Cross-platform (Java)	1998-04	2011-12-15[1]	Yes	EPL	Java
astah*	Change Vision, Inc.	Multi-platform		2011-09-19	No	Commercial, Free trial, Free edition (Community version)	Java, C++, C#
ATL	Obeo, INRIA Free software community	Cross-platform (Java)		2010-06-23	Yes	EPL	Java
Borland Together	Borland	Cross-platform (Java)		2008	No	Commercial	
BOUML	Bruno Pagès	Cross-platform		2011-10	No	Commercial starting from v5.0[2], GPL before v5.0	C++/Qt
Dia	Alexander Larsson/GNOME Office	Cross-platform (GTK+)	2004?	2011-12-18	Yes	GPL	C
Eclipse UML2 Tools[3]	Eclipse Foundation	Cross-platform (Java)	Planning	Planned	Yes	EPL?	Java
Enterprise Architect	Sparx Systems	Windows (Supports Linux & Mac installation)	2000	2011-12-01	No	Commercial	C++
MagicDraw UML	No Magic	Cross-platform (Java)	1998	2010-11-29	No	Commercial	Java
Modelio	Modeliosoft	Windows, Linux	2009	2012-01-25	Yes	GPL V3, Apache 2.0	Java, C++
Objecteering	Objecteering Software	Windows, Linux	1992		No	Commercial	
objectiF	microTOOL	Microsoft Windows	1992	2010-09-21	No	Commercial	Java, C#, C++
Open ModelSphere	Grandite	Cross-platform (Java)	2002-02	2009-11-04	Yes	GPL	Java
Papyrus	Commissariat à l'Énergie Atomique, Atos Origin	Windows, Linux		2010-12-15	Yes	EPL	Java
Poseidon for UML	Gentleware	Cross-platform (Java)		2009	No	Commercial	Java
PowerDesigner	Sybase	Windows	1989	2010	No	Commercial	
RISE	RISE to Bloome Software	Windows (.NET)	2008	2010-09-03	No	Freeware	C#
Software Ideas Modeler	Dusan Rodina	Windows (.NET), Linux (Mono)	2009-08-27	2012-02-06	No	Commercial, Freeware for non-commercial use	C#
StarUML	Plastic Software	Windows	2005-11-01	2006-08-07	Yes	GPL, modified	Delphi
Umbrello UML Modeller	Umbrello Team	Unix-like; Windows	2006-09-09	2009-08-04	Yes	GPL	C++, KDE
Visual Paradigm for UML	Visual Paradigm Int'l Ltd.	Cross-platform (Java)	2002-06-20	2011-09-19	No	Commercial, Free Community Edition	Java

A. Star UML

국내 소프트웨어 업체인 Plastic Software에서 개발된 Plastic 에서 유래되었다.

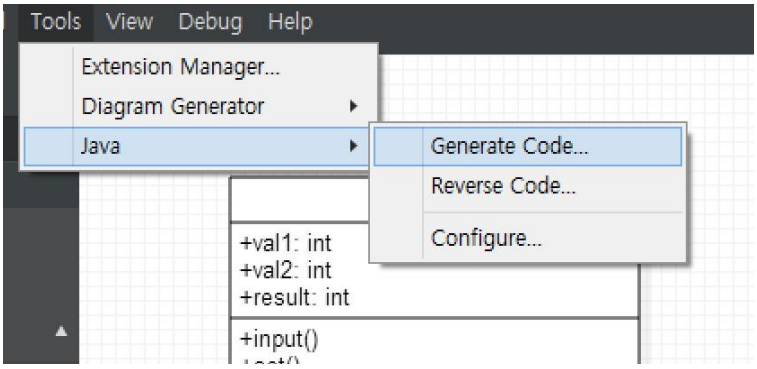
i. 장점

- StarUML은 빠르고, 유연하고 확장 가능하며 풍부한 기능을 제공한다.
- 한국 기업에서 만든 UML 도구이며, 라이선스가 무료라 쉽게 사용이 가능하다. 또한 수업에서의 요구 조건인 'Java Code Generation' 을 충족 가능하다.
- StarUML은 총 11가지의 다양한 종류의 다이어그램을 제공한다.
- 무료 UML 도구로서 보급률이 높기 때문에 관련 자료를 찾기 용이하다.
- 한글 매뉴얼을 제공하고 있어, 사용법을 쉽게 익힐 수 있다.
- UI가 단순하고 직관적이어서 쉽게 익힐 수 있다.

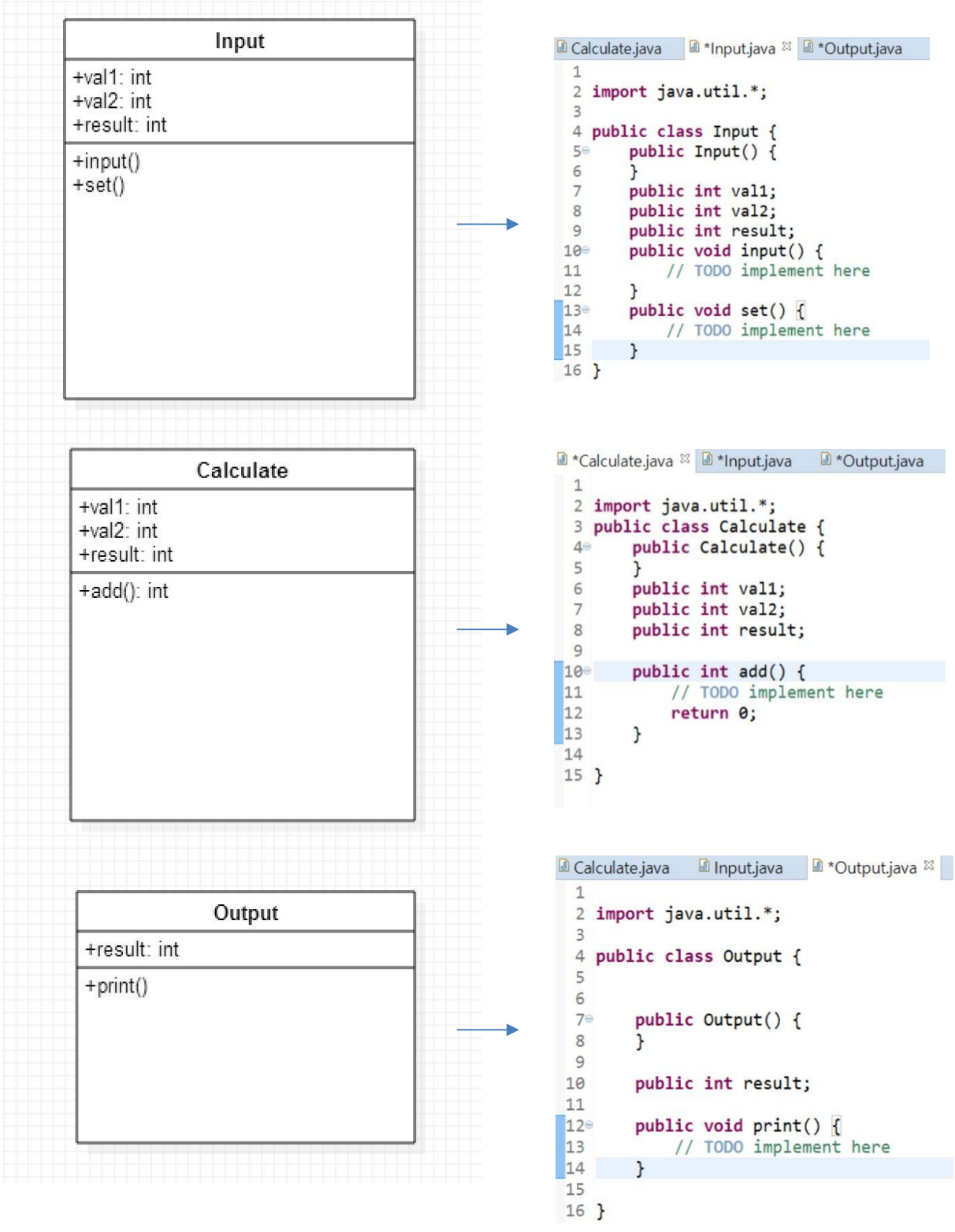


ii. 단점

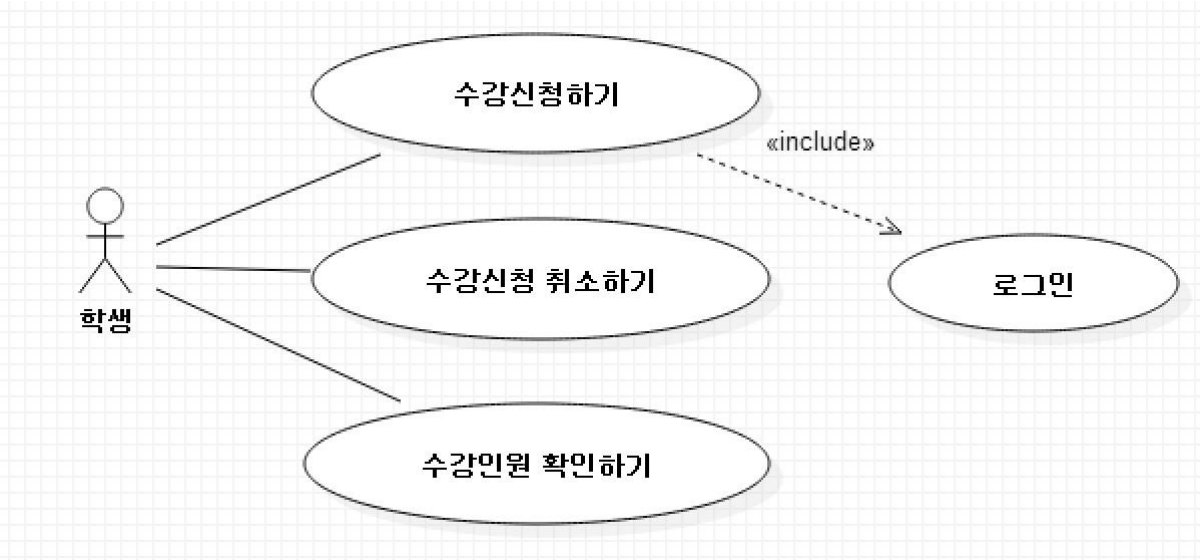
- Java 연동하기 위해 Plug in을 따로 설치해야 한다.



< Class Diagram 작성 후 Java 코드로 변환 >


Input
+val1: int
+val2: int
+result: int
+input()
+set()
Calculate
+val1: int
+val2: int
+result: int
+add(): int
Output
+result: int
+print()
Calculate.java
*Input.java
*Output.java
1 import java.util.*;
2
3 public class Input {
4 public Input() {
5 }
6 public int val1;
7 public int val2;
8 public int result;
9 public void input() {
10 // TODO implement here
11 }
12 public void set() {
13 // TODO implement here
14 }
15 }
*Calculate.java
*Input.java
*Output.java
1 import java.util.*;
2 public class Calculate {
3 public Calculate() {
4 }
5 public int val1;
6 public int val2;
7 public int result;
8
9 public int add() {
10 // TODO implement here
11 return 0;
12 }
13
14 }
Calculate.java
Input.java
*Output.java
1 import java.util.*;
2
3 public class Output {
4
5
6 public Output() {
7 }
8
9 public int result;
10
11 public void print() {
12 // TODO implement here
13 }
14
15 }

< Use Case Diagram 예시 >



B. Amateras UML

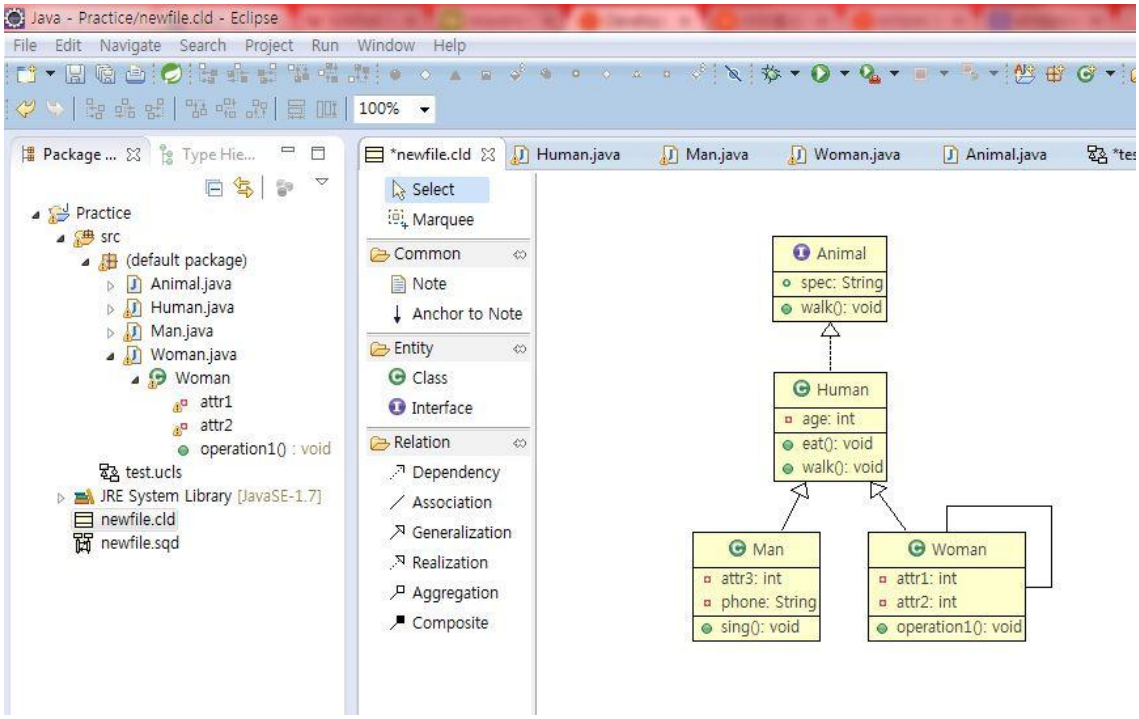
i. 장점

- Eclipse Plug-in으로 해당 URL에서 다운받아 쉽게 설치할 수 있다.
- activity diagram, class diagram, sequence diagram, use case diagram을 무료로 지원한다.
- diagram을 수정가능하고 수정한 내용이 코드에 영향을 준다.
- 개발 완료 후, 추가 작업 시에도 편리하게 사용할 수 있다.
- Class Diagram을 작성했을 때 초기 개발 단계에서 시간과 비용을 절약할 수 있다.

ii. 단점

- private, public와 같은 접근자나 자료형이 기본 세팅 되어 있어서 Export를 통해 코드화를 시킨 후에 수정해야 하는 번거로움이 있다.

< 간단한 Diagram을 작성해본 화면 >



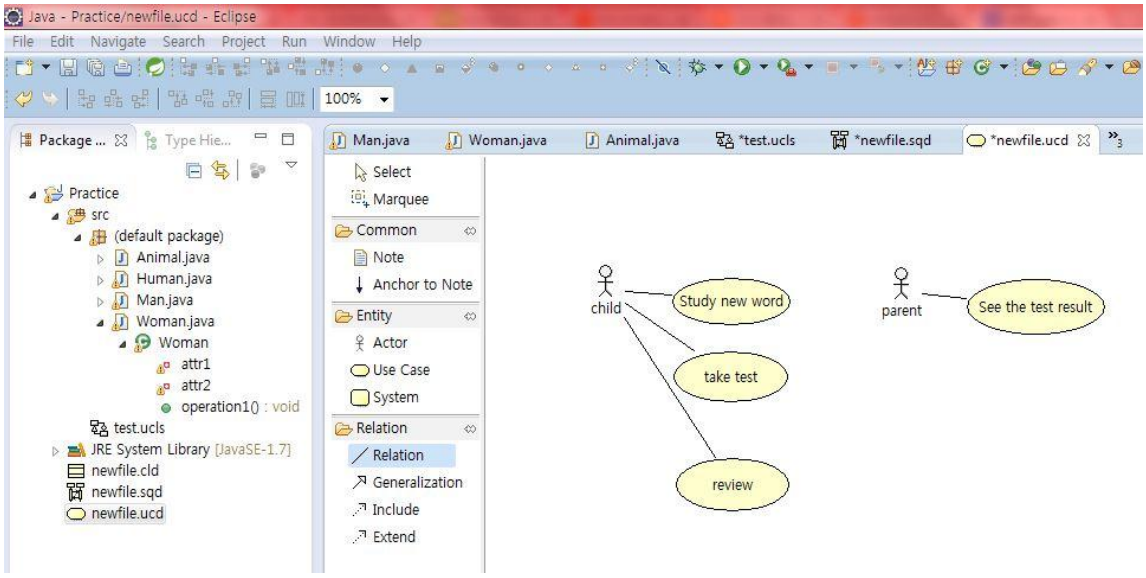
< Diagram를 Export 하여 코드화 >

```
public interface Animal {
    public String spec = null;

    public void walk();

}
```

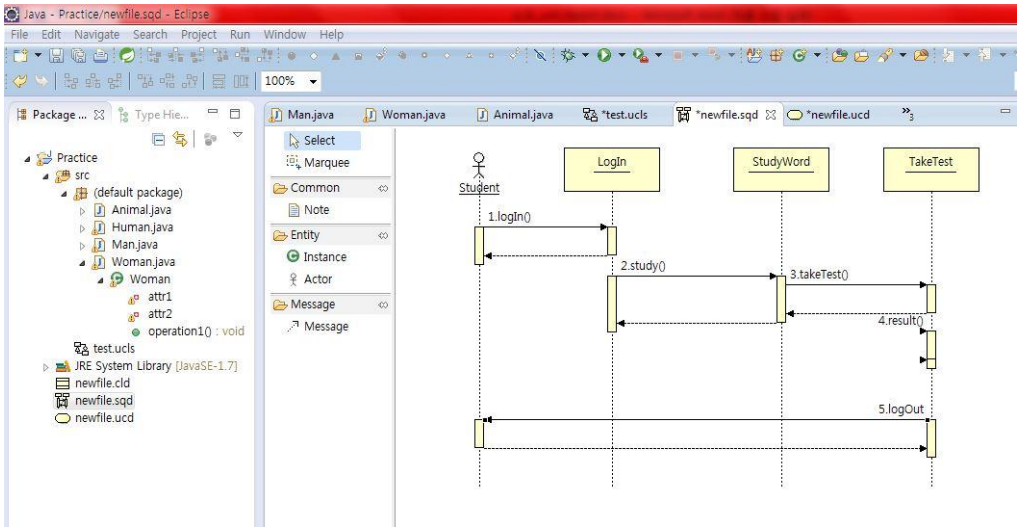
```
public class Human implements Animal {
    private int age;

    public void eat(){
    }

    public void walk(){
    }

}
```

```
public class Man extends Human {
    private int attr3;
    private String phone;

    public void sing(){
    }

}
```

< 간단한 UseCase Diagram 작성한 화면>



< 간단한 Sequence Diagram 작성한 화면 >



C. Object Aid UML

i. 장점

- Eclipse Plug-in으로 해당 URL에서 다운받아 쉽게 설치할 수 있다.
- 가볍고 빠르며 가독성이 높다.
- class diagram(무료) 과 sequence diagram(유료) 을 지원한다.

ii. 단점

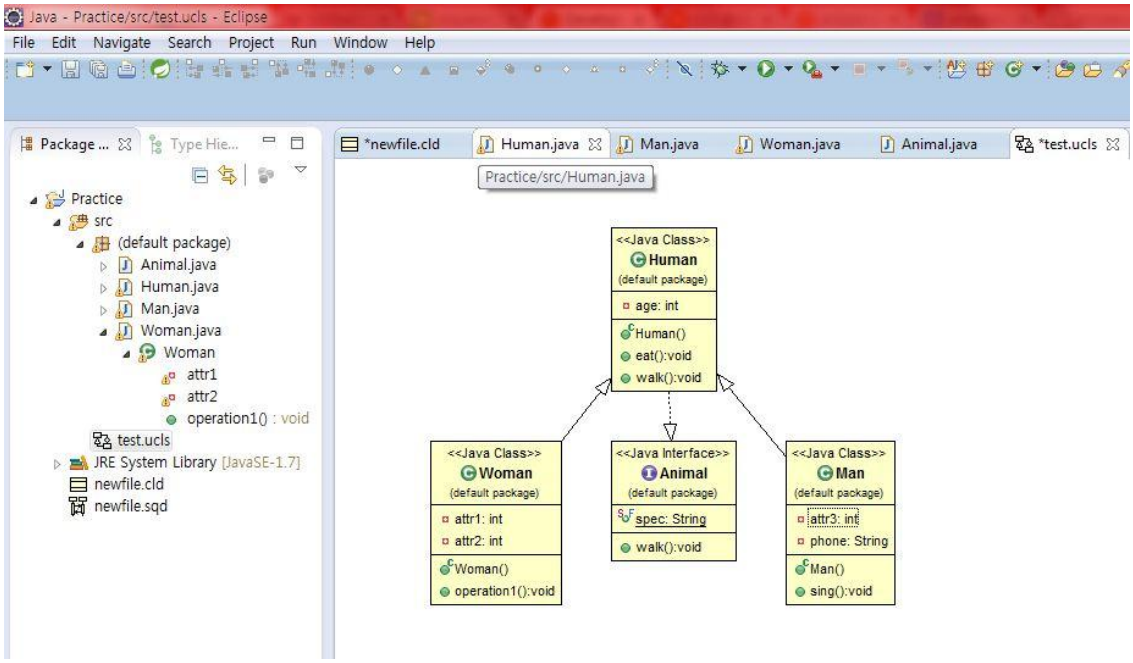
- 자바파일에 의존도가 높다.

- diagram을 수정할 수 없다.

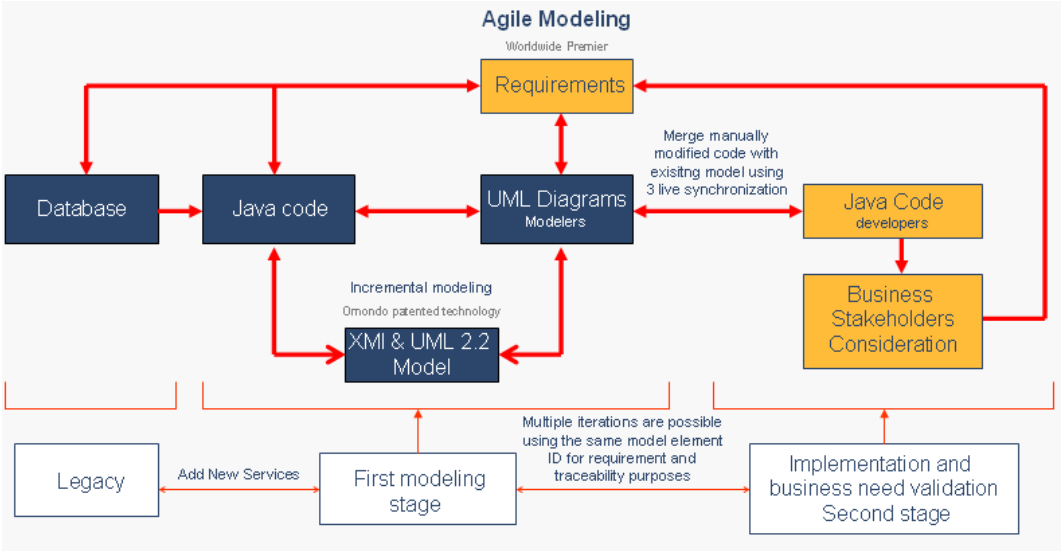
iii. 사용 방법

- Package Explorer에서 자바 파일을 선택하여 화면으로 드래그 & 드롭 한다.

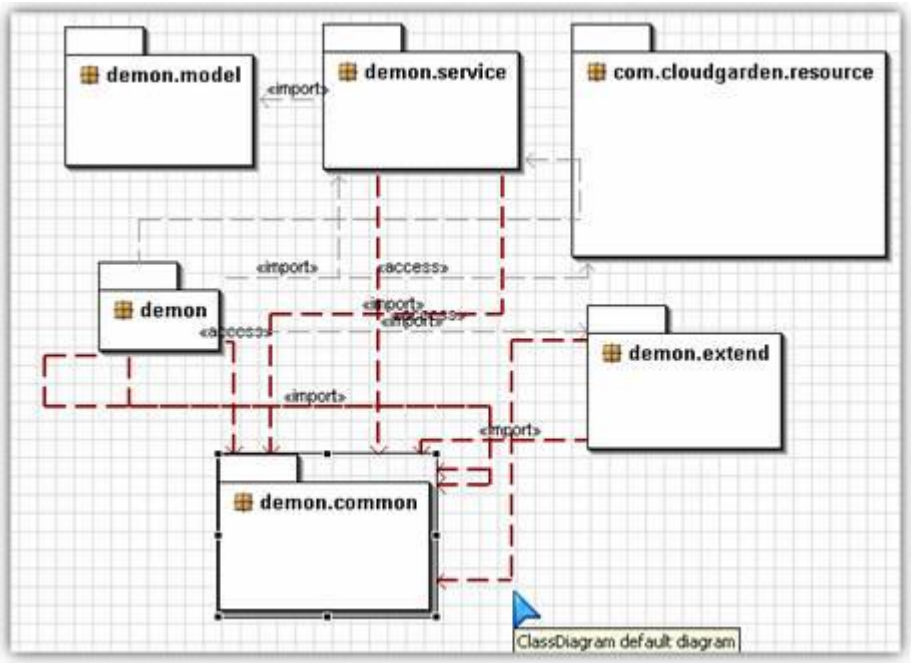
< 작성한 코드로 Diagram 생성한 화면 >



D. Omondo



Eclipse Plug-in 유료판과 무료판이 있다.



E. Rational Rose

Rational 사에서 출시한 UML Tool

가장 강력하고 많은 기능을 제공하여 많이 사용되고 있지만 비용이 상당하기 때문에 조사에서 배제하였다.

5. Conclusion

여러 가지 UML들의 특징들을 조사해보고 장단점을 비교해본 결과, 빠르고 가벼운 Star UML을 사용하기로 결정했습니다. 비교적 단기간에 소규모의 프로그램을 개발하는 것이고, UML을 처음 사용해보기 때문에 자세한 설명과 관련 자료가 많은 Star UML을 사용하는 것이 좋을 것이라고 판단했습니다.

6. Reference

A. Wikipedia Dictionary