Introduction to UML

Project Team
T3

Date
2013-03-21

Team Information
양승민 200911400
정세진 200911418
한종철 200911429

1. Basic concept of UML

- UML이란

  UML은 소프트웨어 시스템이나 업무 모델링 그리고 기타 비 소프트웨어 시스템 등을 나타내는 가공물을 구체화하고, 시각화하고, 구축하고, 문서화하기 위해 만들어진 언어이다.

- UML의 개발 필요성

  UML은 Rational Software와 그 협력회사에 의해 개발되었다. 업무 처리과정에서 그 업무의 범위와 규모가 커짐에 따른 시스템의 복잡성을 처리할 필요성을 느끼게 되었는데, 특히 물리적인 시스템의 분산, 동시성, 반복성, 보안, 결점 보완, 시스템들의 부하에 대한 균등화와 같은 반복해서 발생하는 구조적 문제에 대한 프로세스가 필요하게 되었다. 또한 웹의 발전에 따라 시스템을 만들기는 쉬워졌으나 이러한 구조적 문제는 더욱 악화되었기에 이러한 모든 필요성에 의해 UML이 만들어졌다.

- UML의 목적

  . 사용자에게 즉시 사용가능하고 직관적인 시각적 모델링 언어를 제공함으로써 사용자는 의미 있는 모델들을 개발하고 서로 교환할 수 있어야 한다.

  . 핵심적인 개념을 확장할 수 있는 확장성과 특수화 방법을 제공한다.

  . 특정 개발 프로세스와 언어에 종속되지 않아야 한다.

  . 모델링 언어를 이해하기 위한 공식적인 기초를 제공한다.

  . 협동(Collaboration), 프레임 워크, 패턴, 컴포넌트 같은 고수준의 개발 개념을 제공한다.

2. Kinds of UML Tool

모델링을 하기 위해 사용하는 여러 가지 uml 도구들이 있다.

-IBM RSA(Rational Software Modeler/Architect)

-Together Architect/Designer/Developer (Borland)

-MagicDraw (No Magic)

-Visual Paradigm for UML (Visual Paradigm)

-EA(Enterprise Architect)

-Power Designer

-Visio

등의 상용 UML도구와

-StarUML

-ArgoUML

-JUDE/Community

-TOPCASED-UML2 (TOPCASED Modeling Framework Open Source Project)

-MDT-UML2Tools

등의 비상용 UML도구들이 있다. 이 중에서 starUML을 사용할 것이므로 starUML의 사용법을 설명하도록 하겠다.

3. Use of UML(starUML)

▷ 새 다이어그램 생성하기

StarUML™은 모두 11가지의 UML 다이어그램을 지원한다. 사용자는 필요에 따라 원하는 다이어그램을 생성하여 작성할 수 있다.

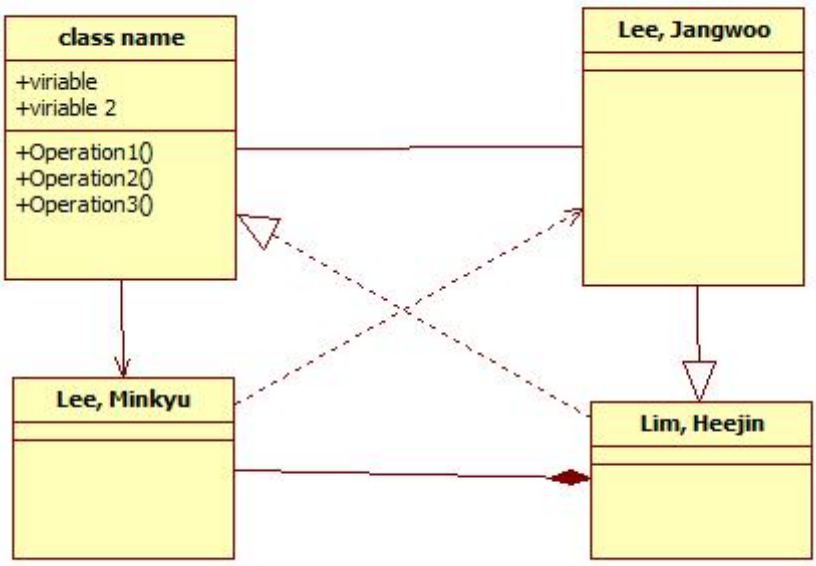
다이어그램 종류	설명
Class Diagram	클래스 다이어그램(Class Diagram)은 클래스관련 요소들의 여러 가지 정적인 관계를 시각적으로 표현한 것이다. 클래스 다이어그램은 클래스(Class) 뿐만 아니라 인터페이스(Interface), 열거형(Enumeration), 패키지(Package) 및 여러 가지 관계들뿐만 아니라 인스턴스(Instance)와 그것들의 연결(Link) 등도 포함할 수 있다.
Use Case Diagram	유스케이스 다이어그램(Use Case Diagram)은 특정 시스템 혹은 개체내의 유스케이스(Use Case)들과 그 외부의 액터(Actor)들 간의 관계를 표현한 것이다. 유스케이스는 해당 시스템의 기능을 표현하며 그것들이 어떤 외부 액터들과 상호작용하는지를 나타낸다.
Sequence Diagram	시퀀스 다이어그램(Sequence Diagram)은 인스턴스들이 어떻게 상호작용을 하는지를 묘사한다. 하나의 협동-인스턴스집합(CollaborationInstanceSet)에 포함된 인스턴스(Instance)들 상호간에 주고받는 자극(Stimulus)들의 집합인 상호작용-인스턴스집합(InteractionInstanceSet)을 직접적으로 표현한다. 시퀀스 역할 다이어그램(Sequence Role Diagram)은 역할(ClassifierRole) 중심의 관점을 반영한 반면, 시퀀스 다이어그램(Sequence Diagram)은 인스턴스(Instance) 중심의 관점을 반영한 것이다.
Sequence Diagram (Role)	시퀀스 역할 다이어그램(Sequence Role Diagram)은 역할 개념들이 어떻게 상호작용을 하는지를 묘사한다. 하나의 협동(Collaboration)에 포함된 역할(ClassifierRole)들 상호간에 주고받는 메시지(Message)들의 집합인 상호작용(Interaction)을 직접적으로 표현한다. 시퀀스 다이어그램(Sequence Diagram)은 인스턴스(Instance) 중심의 관점을 반영한 반면, 시퀀스 역할 다이어그램(Sequence Role Diagram)은 역할(ClassifierRole) 중심의 관점을 반영한 것이다.
Collaboration Diagram	협동 다이어그램(Collaboration Diagram)은 인스턴스들이 어떻게 협동하는지를 묘사한다. 하나의 협동-인스턴스집합(CollaborationInstanceSet)에 포함된 인스턴스(Instance)들의 협동 모델을 직접적으로 표현한다. 협동 역할 다이어그램(Collaboration Role Diagram)은 역할(ClassifierRole) 중심의 관점을 반영한 반면, 협동 다이어그램(Collaboration Diagram)은 인스턴스(Instance) 중심의 관점을 반영한 것이다.
Collaboration Diagram (Role)	협동 역할 다이어그램(Collaboration Role Diagram)은 역할 개념들이 어떻게 협동하는지를 묘사한다. 하나의 협동(Collaboration)에 포함된 역할(ClassifierRole)들의 협동 모델을 직접적으로 표현한다. 협동 다이어그램(Collaboration Diagram)은 인스턴스(Instance) 중심의 관점을 반영한 반면, 협동 역할 다이어그램(Collaboration Role Diagram)은 역할(ClassifierRole) 중심의 관점을 반영한 것이다.
Statechart Diagram	상태 다이어그램(Statechart Diagram)은 특정 개체의 동적인 행위를 상태(State)와 그것들 간의 전이(Transition)를 통해 묘사한다.

	일반적으로 클래스의 인스턴스에 대한 행위를 묘사하는데 사용되지만 그 밖의 요소들에 대해서도 얼마든지 사용될 수 있다.
Activity Diagram	액티비티 다이어그램(Activity Diagram)은 상태 다이어그램의 특별한 형태로써, 활동들의 수행 흐름을 묘사하는데 적합하다. 일반적으로 작업흐름(Workflow)을 표현하기 위해 많이 사용되며, 클래스, 패키지 혹은 연산 등의 개체에 대해 주로 사용된다.
Component Diagram	컴포넌트 다이어그램(Component Diagram)은 소프트웨어 컴포넌트 사이의 의존관계를 묘사한다. 소프트웨어 컴포넌트를 구성하는 요소들과 그것들을 구현하는 요소들도 모두 표현될 수 있다.
Deployment Diagram	디플로이먼트 다이어그램(Deployment Diagram)은 물리적인 컴퓨터 및 장비 등의 하드웨어 요소들과 그것에 들이 배치되는 소프트웨어 컴포넌트, 프로세스 및 객체들의 형상을 묘사한다.
Composite Structure Diagram	복합구조 다이어그램(Composite Structure Diagram)은 분류자(Classifier)의 내부 구조를 표현하는 다이어그램이다. 여기에는 Classifier가 시스템의 다른 부분들과의 상호작용하는 지점 등을 포함한다.

이렇게 11가지의 UML diagram 이 존재한다.

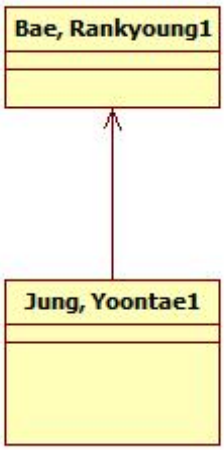
- diagram1. Class Diagram

Class Diagram은 문제영역을 구성하는 모델 요소간의 관계를 나타내는 다이어그램이다.

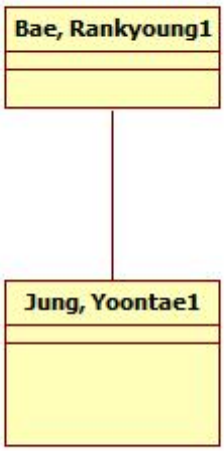


위 그림과 같이 class를 사각형의 형태로 그리며, class 안에는 class의 멤버변수들과 멤버 메소드들의 이름이 들어가게 된다. 멤버 메소드들의 리턴 타입들을 설정해 줄 수 있다.

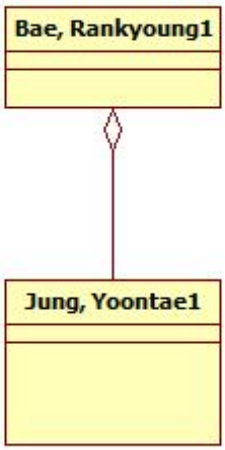
또한 각 class의 관계들을 다양한 화살표로 나타낼 수 있다.



위 그림은 direct association관계로 setter의 관계처럼 일방 적으로 이용하는 경우에 연결하는 화살표 이다.

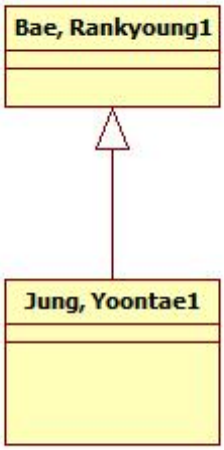


위 그림은 일반적인 association관계로 각 클래스가 서로 각자의 멤버 메소드나 멤버 변수에 대해서 필요로 할 때 일반적으로 연결하는 선이다.

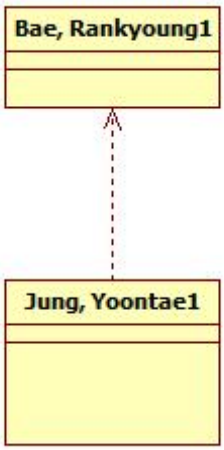


위 그림은 aggregation의 관계를 나타내는 그림이다. 하위 class가 상위 class의 한 부분(집합)이 될 경우 사용하는 화살표이다.

위 그림은 composition의 관계를 나타내는 그림이다. aggregation과 비슷한 개념이지만 aggregation에 비해서 상위class와 하위 class간이 연결이 더 밀접한 경우에 사용하는 화살표이다.



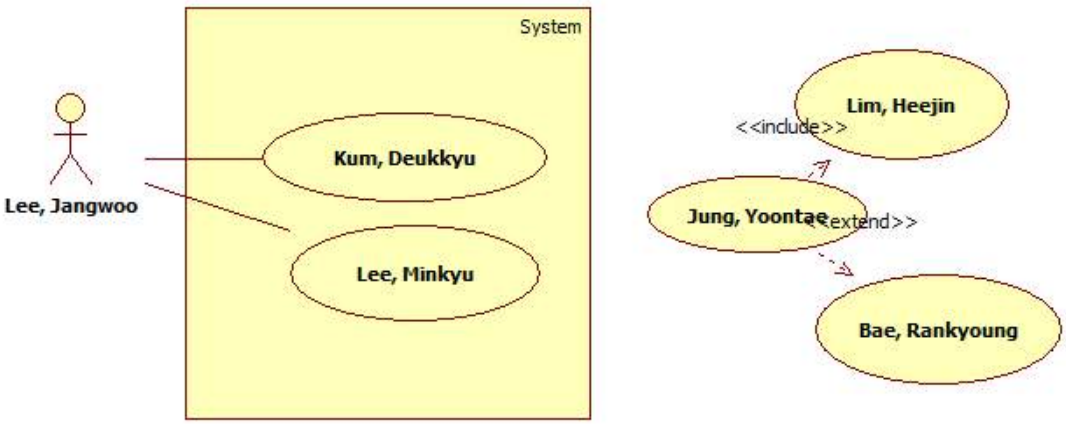
위 그림은 generalization의 관계를 나타낸다. generalization은 일반화로 보통 is-a 의 관계에 사용하는 화살표이다. (ex) 초식동물은 동물이다.(초식동물 is a 동물)) Jung, Yoontae is a Bae, Rankyoung



위 그림은 dependency를 나타낸다. 이는 의존의 관계로 화살표를 보내는 쪽이 받는 쪽의 멤버 변수나 멤버 메소드들을 이용할 경우에 사용한다.

- diagram2. Use Case Diagram

유스케이스(UseCase)는 시스템의 행위(behavior)를 정의하기 위해 사용하는 요소이다. 일반적으로 유스케이스는 액터와 상호작용한다.



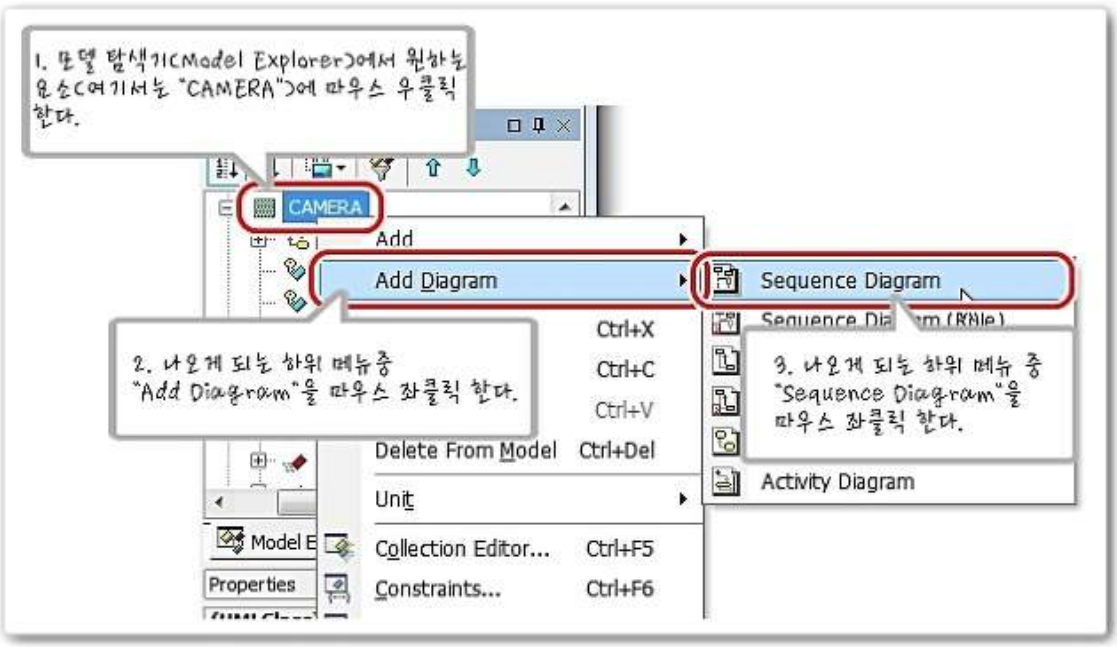
그림과 같이 왼쪽의 사람 모양 actor와 system boundary를 그려서 association관계를 설정해 줄 수 있다.

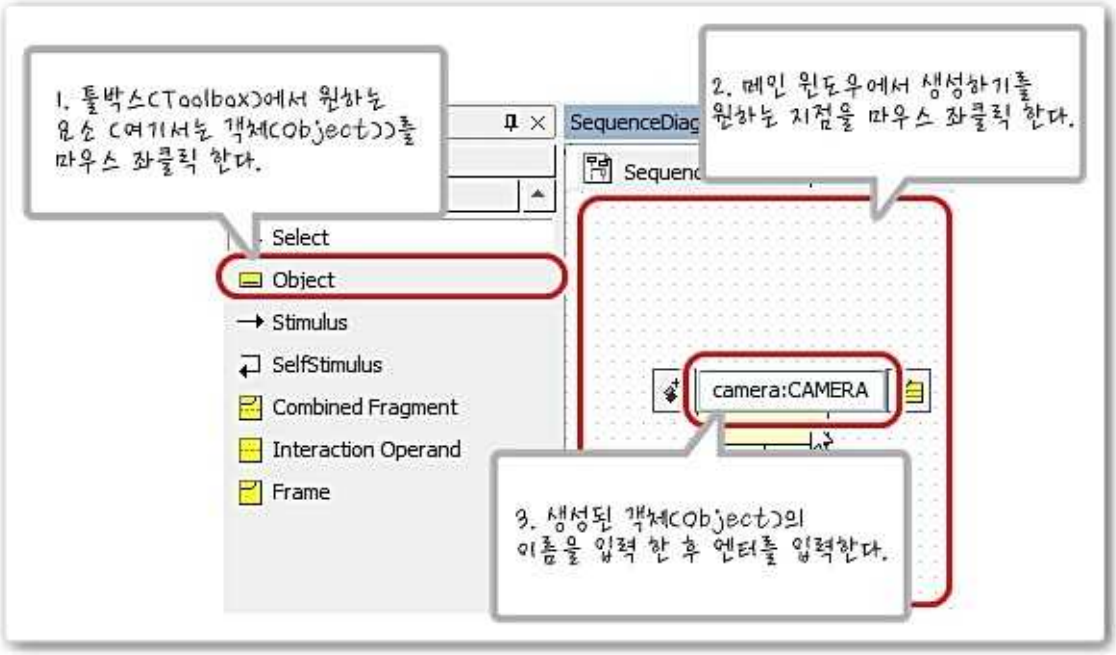
또한 오른쪽 그림처럼 각 use case 간의 extend와 include를 설정하여 그려 줄 수 있다.

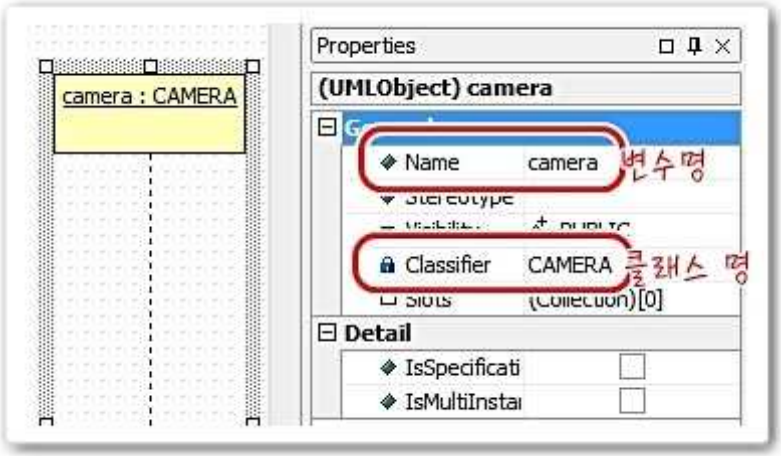
- diagram3. Sequence Diagram

시간에 따라 위에서 아래로 수행 되고, 옆으로 오브젝트 간에 메시지가 왕래한다.

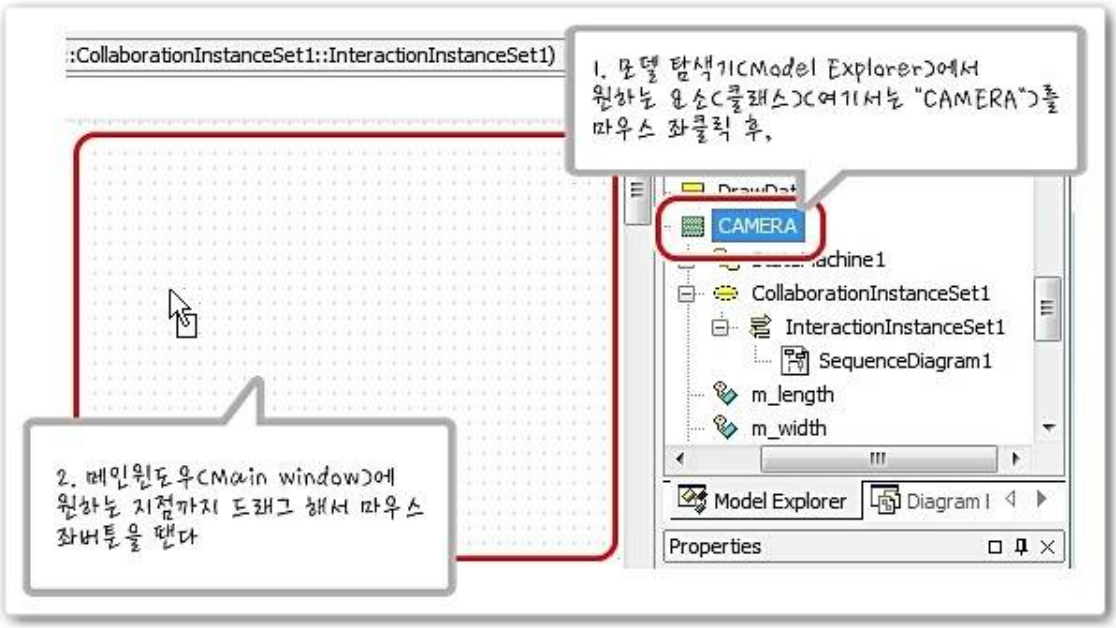
Collaboration Diagram과는 다르게 시간의 흐름을 표현한다.

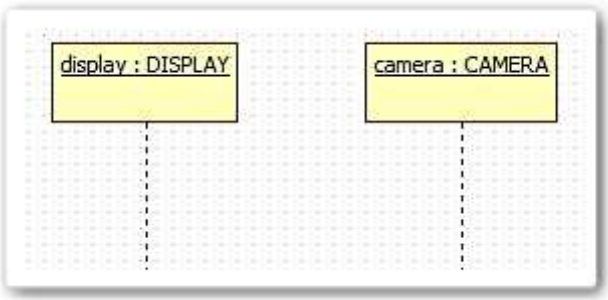




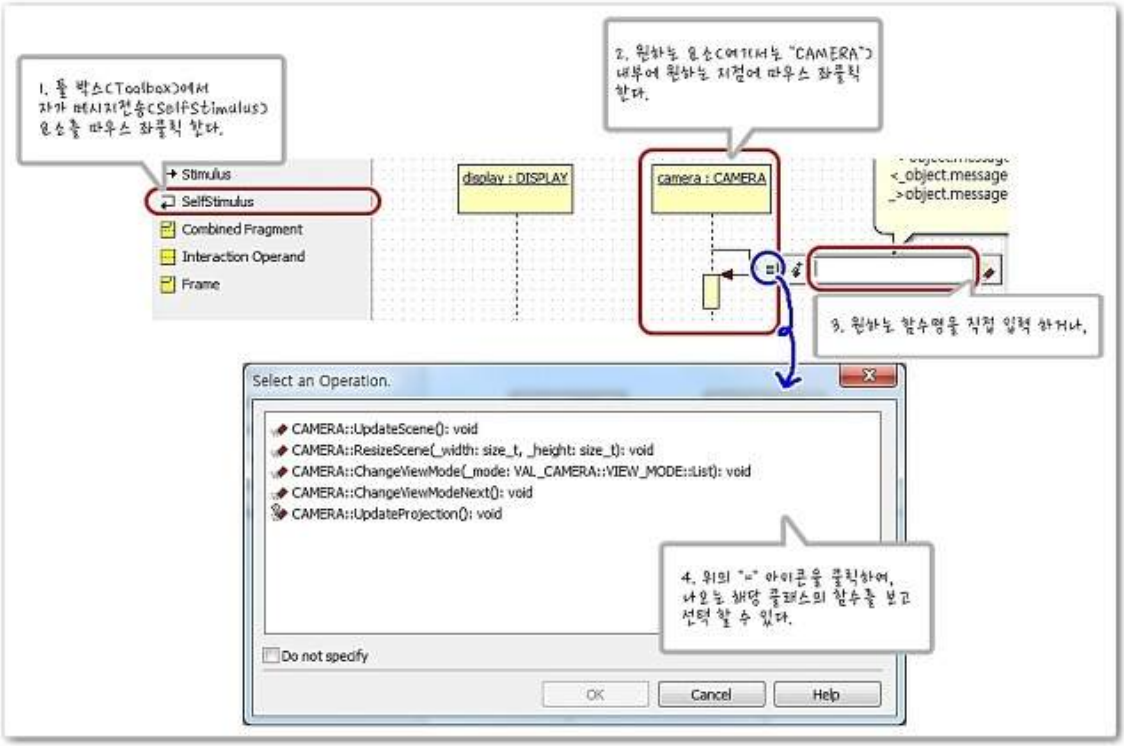


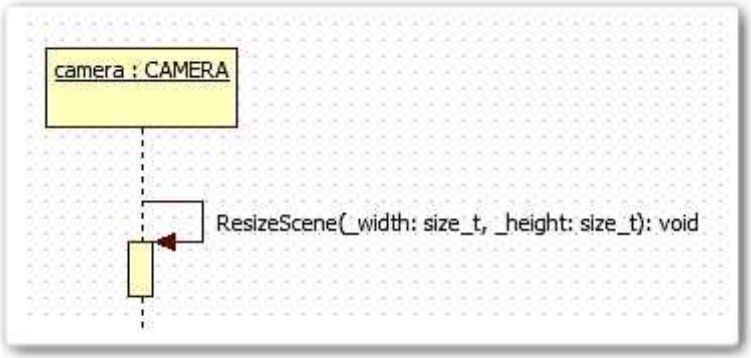
방금 생성된 객체를 선택 후 속성창(Properties)를 보게 되면, Name과 Classifier가 있는데, 이들은 위에 표기 한대로 변수명과 클래스명을 의미한다.


:CollaborationInstanceSet1::InteractionInstanceSet1)
1. 모델 탐색기(Model Explorer)에서 원하는 요소(클래스)(여기서는 "CAMERA")를 마우스 좌클릭 후,
CAMERA
CollaborationInstanceSet1
InteractionInstanceSet1
SequenceDiagram1
m_length
m_width
Model Explorer
Diagram I
Properties
2. 메인윈도우(Main window)에 원하는 지점까지 드래그 해서 마우스 좌버튼을 땐다

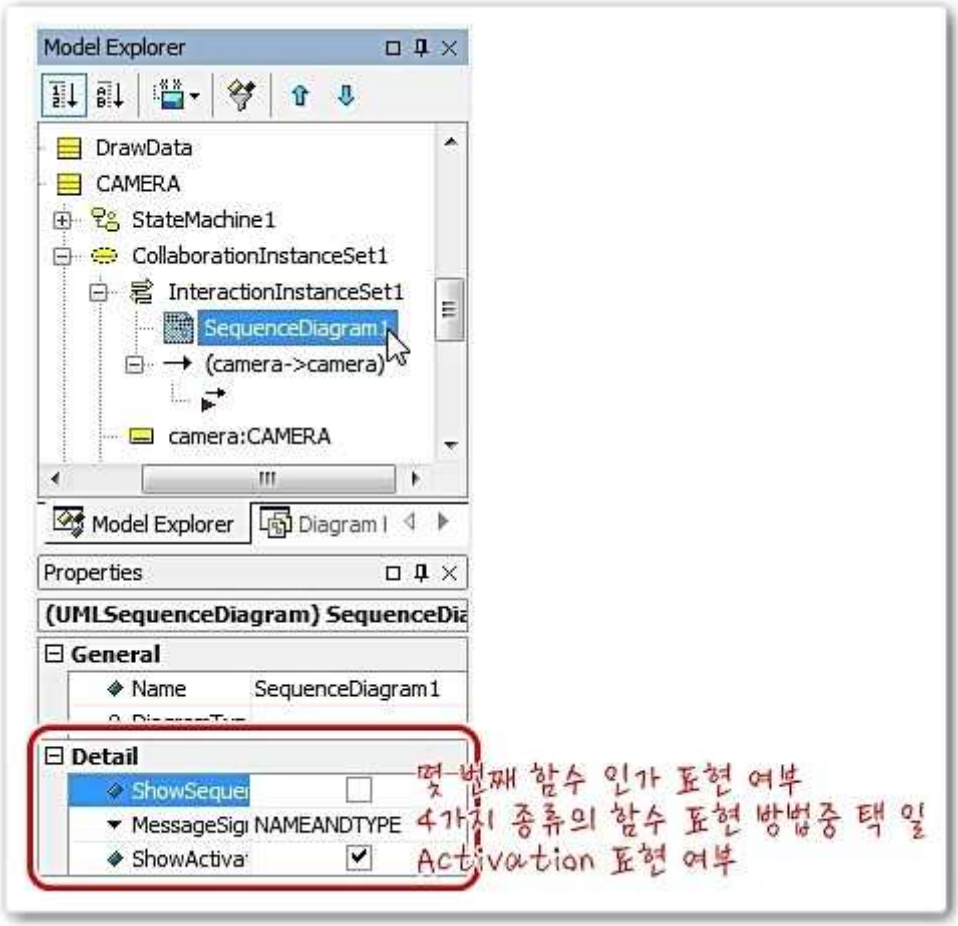

display : DISPLAY
camera : CAMERA

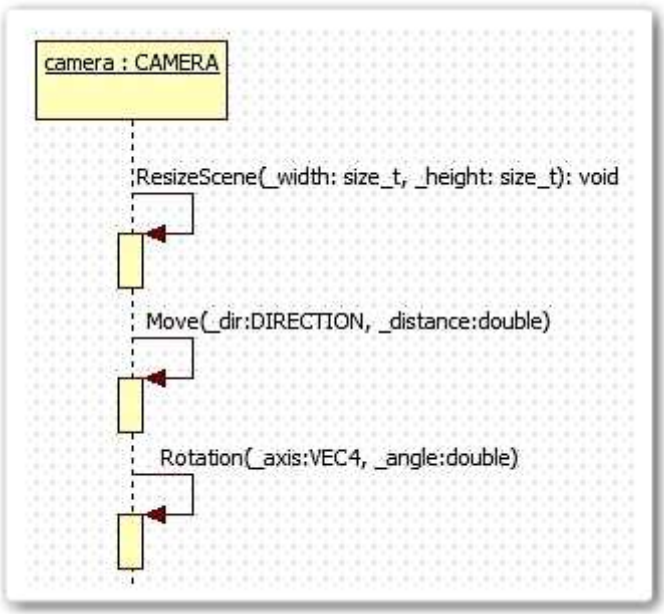
자가 메시지 전송(SelfStimulus)은 아래와 같이 삽입 할 수 있다.





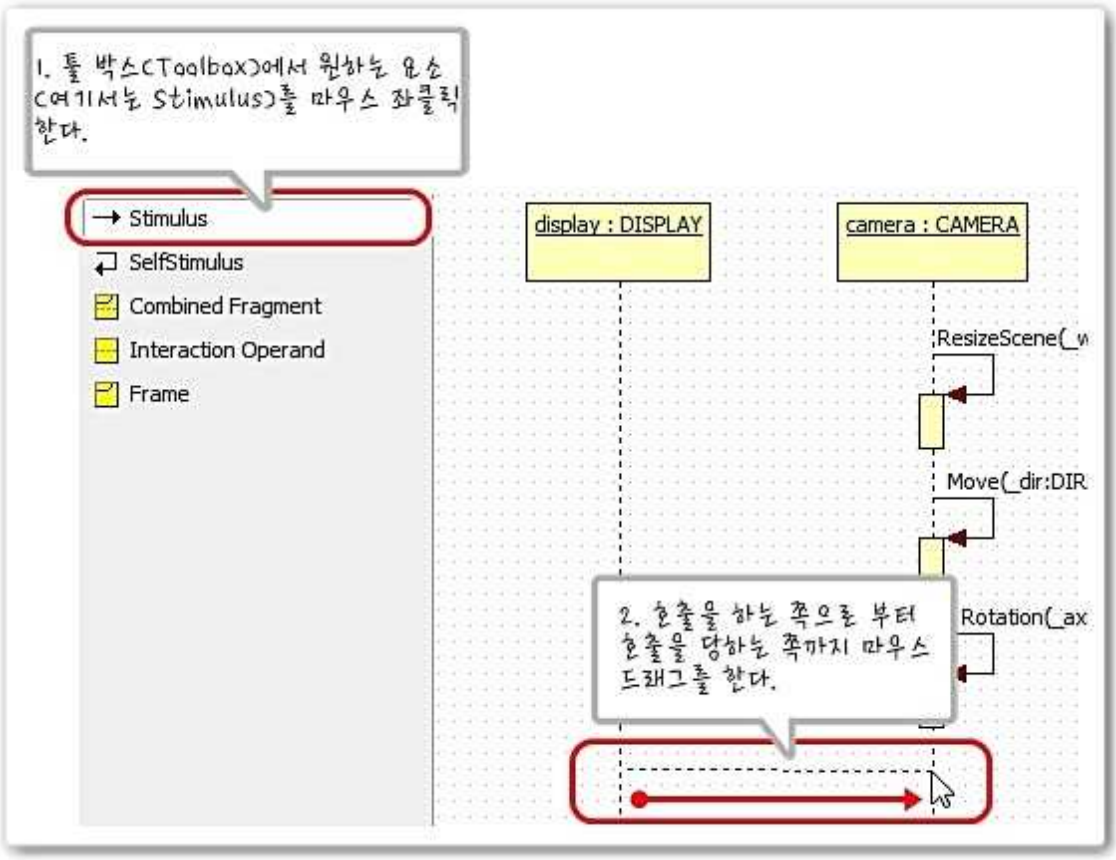
모델 탐색기(Model Explorer)에서 해당 시퀀스 다이어그램의 요소를 선택 한 후, 속성창(Properties)을 보게 되면 아래와 같은 속성을 볼 수 있다.



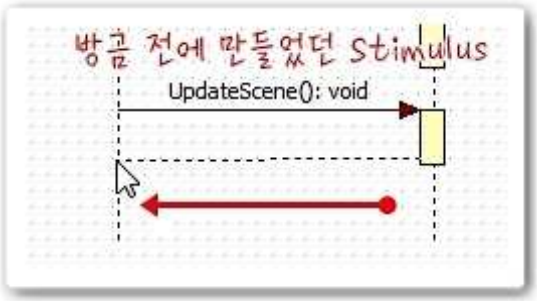


"CAMERA" 객체는 "DISPLAY" 객체로 부터 "UpdateScene" 메시지를 받게 된다. 해당 함수(Operation)는 "CAMERA" 객체가 가지고 있는 것이고, 이 부분은 스스로 호출 하는 것이 아니다.

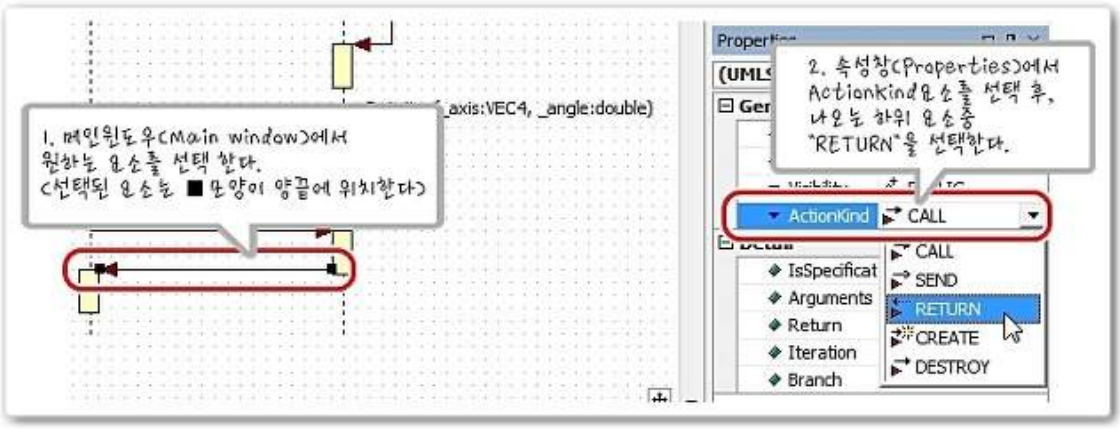
거의 비슷한 방법으로 Stimulus(메시지 전송)는 드래그 하는 방식으로 만들어 진다.



메시지 전송(Stimulus)를 역으로 만들어 주면 아래와 같은 모양이다.

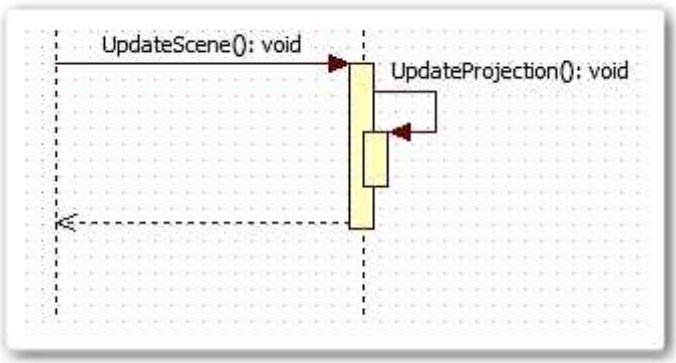


아래 그림은 return을 나타낸다. display에 가서 별다른 작업을 하지 않을 테니까 Activation이 필요 없으므로 아래와 같이 한다.



.

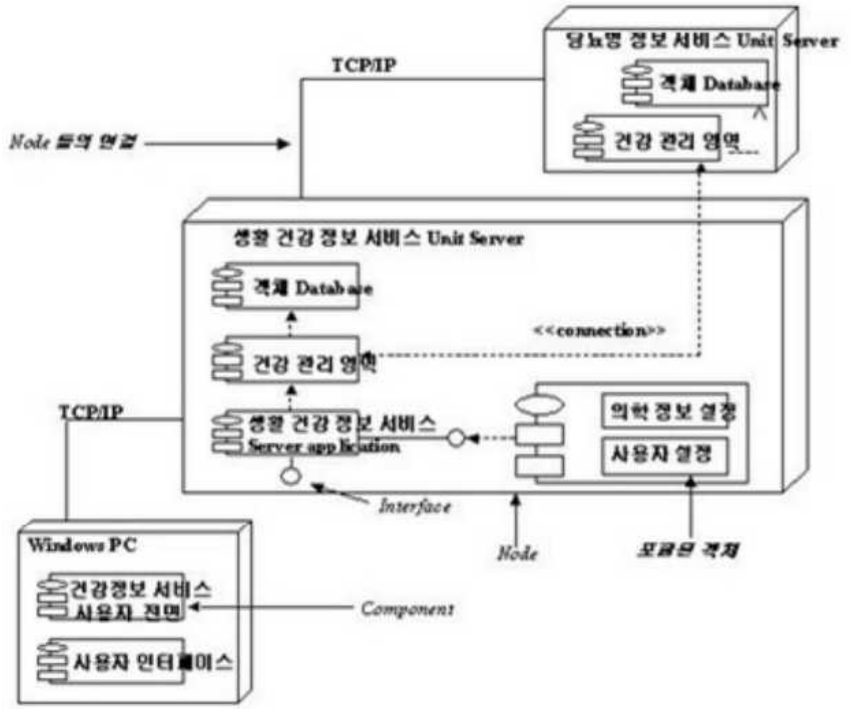
UpdateScene이 호출 되면, 내부적으로 "CAMERA"객체는 UpdateProjection을 호출 하게 되고 UpdateProjection은 UpdateScene의 함수(Operation)내에서 이루어지는 것이기 때문에 표현하면 아래와 같이 된다.



- diagram4. Deployment Diagram

각 시스템의 하드웨어, 소프트웨어 컴포넌트들의 관계를 나타낸다.

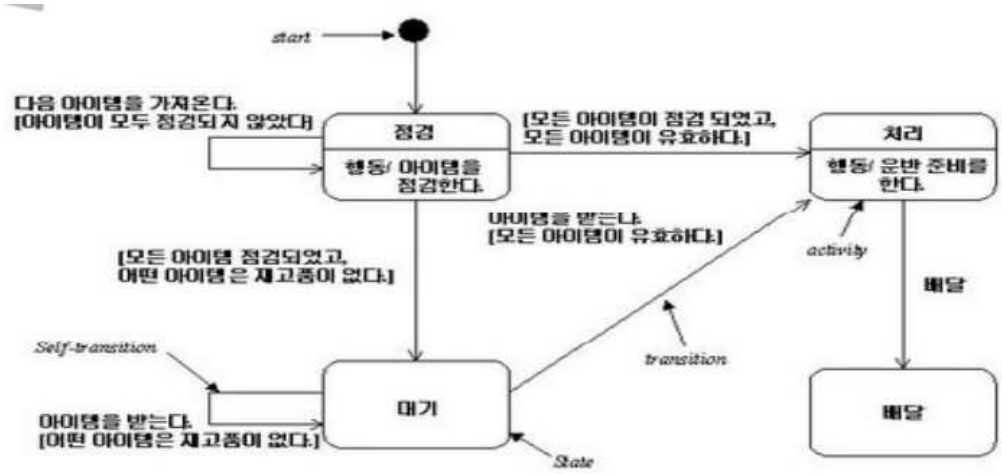
Node라는 notation으로 computational unit(대부분 하드웨어적인 부분)을 나타낸다.



- diagram5. Collaboration Diagram

Sequence Diagram과의 차이점을 살펴보면, Sequence Diagram이 시간에 따른 행위의 흐름에 역점을 두고 표현하지만 Collaboration Diagram의 경우 객체들 사이의 정적인 구조에 더 중점을 둔다.

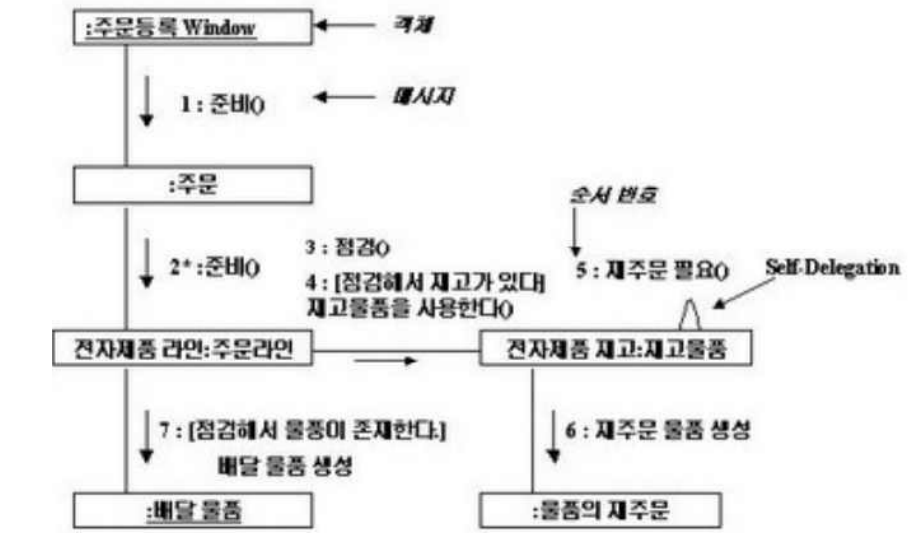
객체 상호간의 관계에 역점을 두었기 때문에 객체 하나의 행위를 정확하게 표현하기에는 부적절하다.

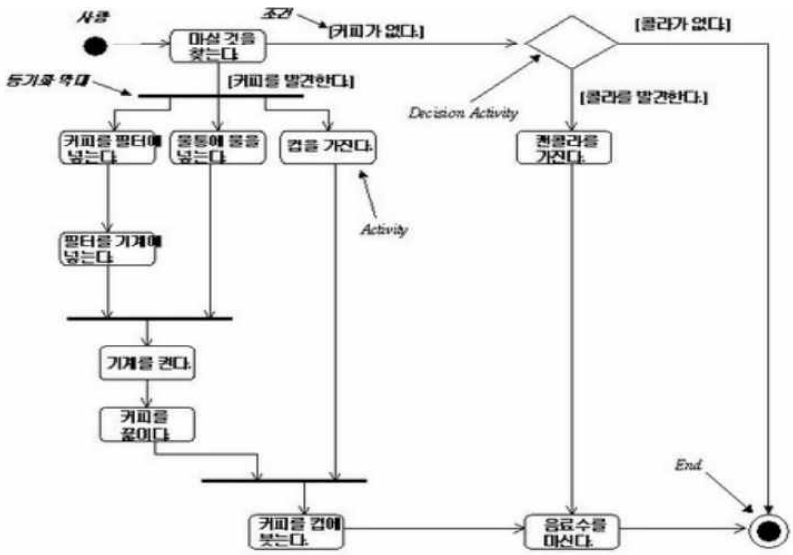


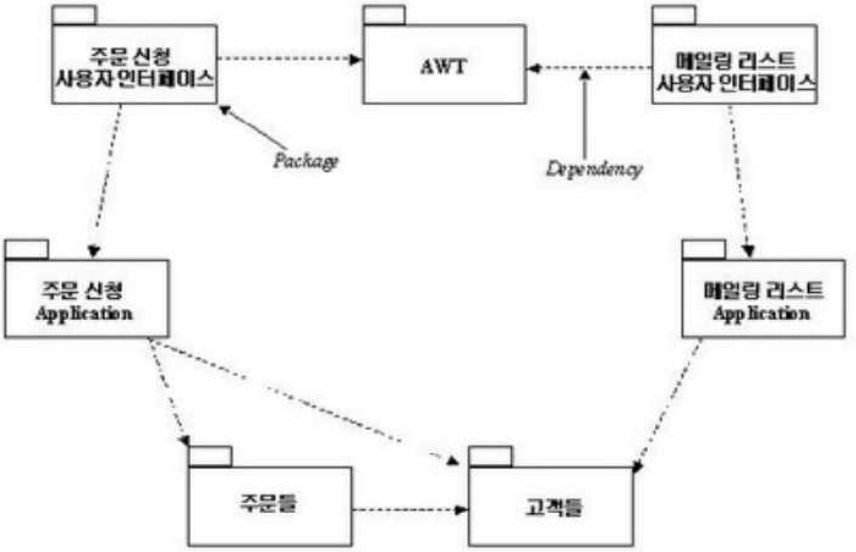
- diagram6. Activity Diagram
순서도나 병렬적인 처리를 요하는 행위를 표현할 때 사용하면 유용하다.
순서에 따른 activity를 나타내는 것으로 모델링하고 있는 diagram에서의 activity의 의미를 파악하는 것이 중요하다.

- diagram7. Package Diagram
class라는 개념의 등장으로 class들이 상위개념으로의 그룹화 방법으로 제시된다.
package란 하나의 class가 아닌 system에서의 하나의 modeling element이다.

- diagram8. State Chart Diagram
Object가 가질 수 있는 모든 상태와 어떠한 event를 받았을 때 어떠한 상태로 변화하는지를 나타낸다.







4. Code generation and Conclusion

UML tool을 사용해 Class Diagram을 생성할 경우 Code generation하면 생성한 Class Diagram 관계대로 코드가 생성되어 코딩 할 수 있는 환경을 만들어준다.
UML tool의 사용은 GUI로 되어 있어서 마우스 클릭과 드래그로 편하게 그릴 수 있다.