

Introduction to OOAD using UML tools

<10 조>

200910045 이호진

200911388 박미관

200911415 이지호

<객체지향 모델링 절차>

- ① 문제기술서
- ② 클래스 도출(클래스)
- ③ 속성/연산 도출(속성, 연산)
- ④ 관계 도출(가시성과 정보은닉, 클래스간의 관계, 상속)

[예제 시스템] Whiteboard 시스템

1) Whiteboard 시스템

- 사용자가 캔버스에 도형을 작성하고, 이에 대한 편집을 지원하는 시스템

2) 기능

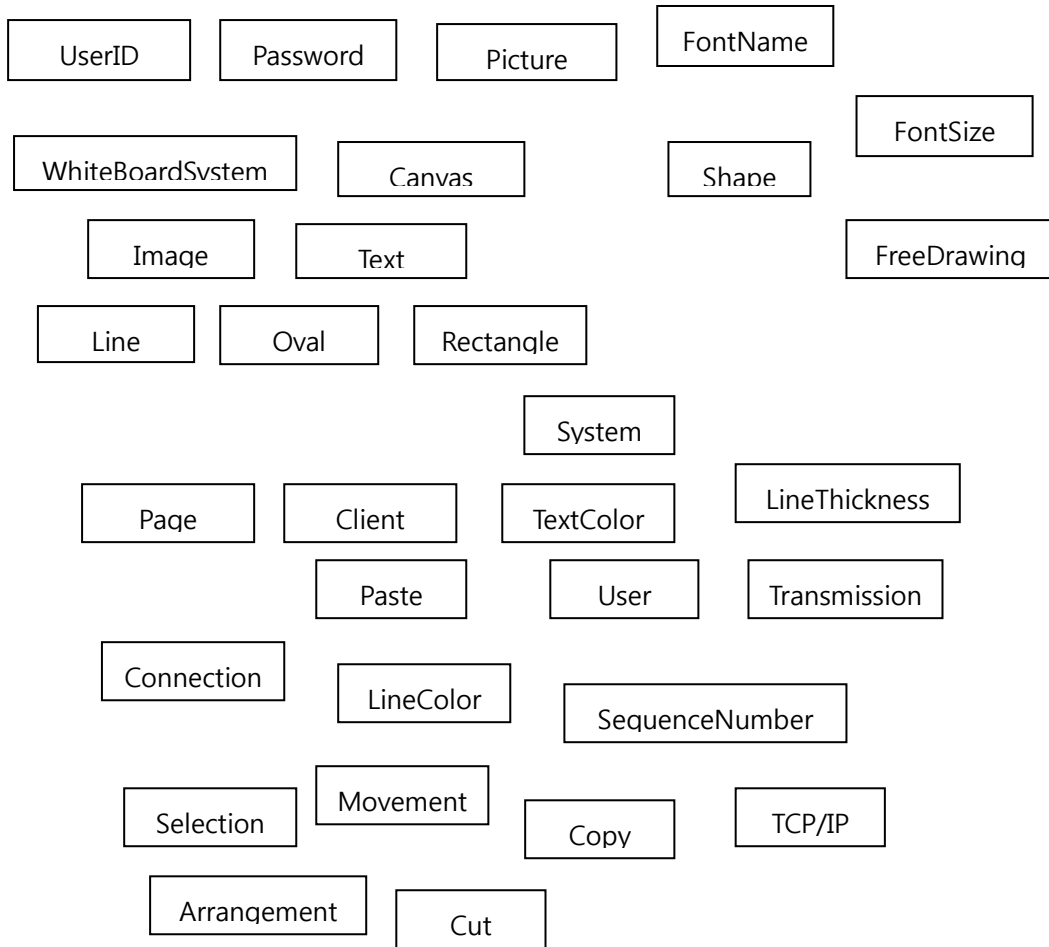
- ① 도형 편집 기능 - 사용자는 캔버스에 각종 도형을 생성/편집/삭제 가능
- ② 캔버스 관리 기능
- ③ 녹화 재생 기능 - 자신이 수행한 모든 편집 동작을 그대로 기록
- ④ 방송 기능 - 도형에 대한 편집 동작을 시스템에 연결된 클라이언트에게 실시간으로 전송

1. 클래스 도출

1) 후보 클래스 도출

(1) 명사 분석 방법

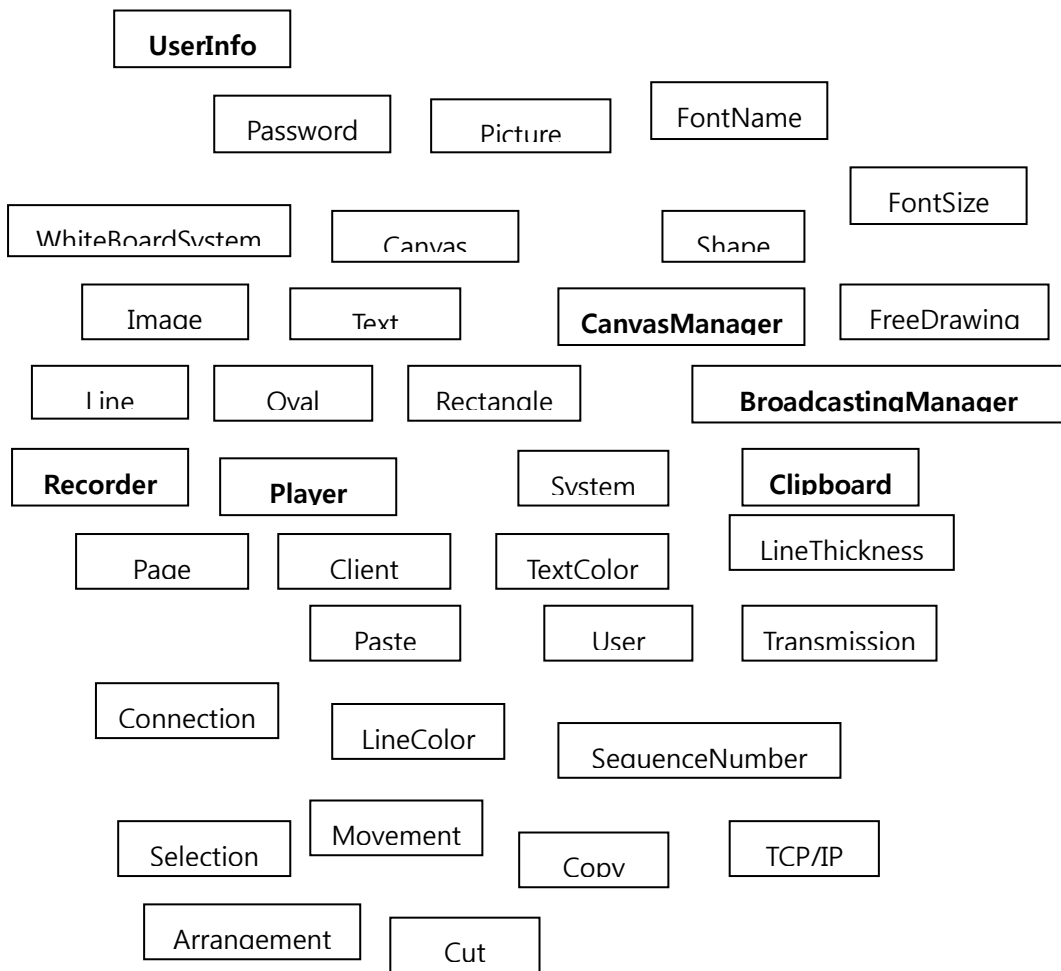
- 클래스를 도출하는 가장 간단하면서도 효과적인 방법으로서, 문제 기술서를 바탕으로 명사를 찾는 방법. 문제 기술서에는 시스템에 대한 개괄적인 소개를 포함하여 시스템이 제공할 기능에 대한 설명이 담겨 있다. 시스템에 대한 소개 및 기능에 대한 설명은 시스템에 의해서 구현될 도메인 상에서 존재하는 중요한 개념 즉 클래스를 포함할 것이다. 따라서 문제 기술서가 충분히 명확하고 구체적으로 작성되었다면, 문제 기술서를 정독하면서 문제 기술서에 나타난 명사들을 찾는 것이 클래스를 도출하는 효과적인 방법이 될 수 있다.



[화이트보드 시스템의 후보 클래스 도출 - 명사 분석 방법]

(2) 문제 영역에 대한 배경 지식 활용

- 개발자는 문제 기술서에는 나오지 않는 것이라 하더라도 시스템이 동작하는 문제 영역에 대한 지식과 상식을 바탕으로 추가적으로 클래스를 선정할 수도 있다. 문제 기술서는 많은 경우 해당 분야에 대한 지식을 가지고 있는 고객에 의해서 작성되기 때문에 배경 지식 보다는 시스템이 제공할 기능에 대한 소개가 주를 이룬다. 그러므로 문제 기술서에는 누락된 정보가 클래스로서 도출되어야 할 중요한 개념이 경우가 많다. 따라서 개발자는 자신의 상식을 포함하여 그 분야에 대한 배경 지식을 바탕으로 클래스를 도출해야 한다. 그런데, 전문적인 개발자인 경우에는 해당 분야에 대한 지식이 부족할 것이다. 이럴 경우에는 해당 분야에 대한 정보를 얻을 수 있는 모든 방법(그 분야에 대한 입문 도서, 고객 사에서 작성한 업무 매뉴얼, 인터뷰 등)을 동원하여 개발자는 그 분야의 중요한 대상과 개념을 클래스로서 도출할 수 있도록 해야한다.



[화이트보드 시스템의 후보 클래스 도출 - 문제 영역에 대한 배경 지식 활용]

(3) 전형적인 유형의 클래스 도출

- 클래스를 도출할 때는 몇 가지 전형적인 유형의 클래스에 초점을 두고 찾아보는 것도 하나의 방법이다.

실 세계에 존재하는 수많은 대상과 개념들은 어떤 기준에 따라서 분류하고 체계화한다. 이렇게 체계화된 개념 각각이 클래스가 될 수 있다.

따라서 클래스를 도출할 때도 개발자는 정의된 분류 기준에 따라서 클래스를 적극적인 방식으로 도출할 수 있다.

① 물리적인 객체

- 실 세계에 실제로 존재하며 시각 등에 의해서 인지될 수 있는 대상을 뜻한다. 이런 유형의 객체는 직관적으로 인식되므로 비교적 쉽게 클래스로서 도출될 수 있다. 실 생활에서 사람들이 사용하는 장치, 기계 등을 감시 또는 제어하는 유형의 시스템에서 쉽게 발견된다.

② 업무와 작업의 참여자 및 결과물

- 정보 시스템의 경우에는 사용자가 실제 업무의 수행 결과로 산출되는 많은 유형의 결과물 즉 문서 등이 클래스로서 파악되는 경우가 일반적이다. 다시 말해서 업무를 수행하기 위하여 사용되는 중요한 문서들이 바로 클래스에 해당된다. 뿐만 아니라 해당 업무를 직접 수행하거나 수행하는 과정에서 참여하는 여러 유형의 담당자도 클래스로서 도출될 수 있다.

③ 시스템 외부와의 인터페이스

- 시스템이 시스템 외부에 존재하는 장치 또는 다른 시스템과의 협력 없이 완전히 독립적인 방식으로 수행되는 경우가 있다.

④ 개념적인 객체

- 물리적으로 인지할 수 있거나 업무와 작업의 참여자 또는 입력과 결과물 또는 외부 시스템과 연결하는 클래스와 같이 비교적 구체적인 클래스 외에 실제로 인지할 수 없는 추상적인 개념도 클래스로서 간주될 수 있다.

ex) 화이트보드 시스템의 후보 클래스

2) 부적절한 클래스 제거

(1) 시스템 외부의 존재

- 후보 클래스가 시스템 외부의 대상과 존재를 뜻하는 경우에 그 후보 클래스는 시스템 외부의 것이므로 제외되어야 한다. 즉 시스템 외부에 존재하는 사용자 또는 다른 시스템은 그 자체가 클래스는 될 수가 없다. 대신에 사용자에게 대한 정보 또는 외부 시스템과의 통신을 전담하는 것을 클래스로 볼 수 있다.

(2) 중복된 클래스

- 동일한 대상과 개념을 뜻하는 후보 클래스 중에서 하나만 선정되고 나머지 후보 클래스는 제외되어야 한다. 또한 하나의 객체가 여러 역할을 수행하는 경우에 만약 각 역할별로 명확히 다른 정보가 필요하거나 행위를 제공한다면 각 역할별로 별도의 클래스를 정의해야 한다. 반면에 이름이 다르거나 실 세계에서 다른 역할을 수행하는 것처럼 판단되지만 시스템 관점에서 볼 때는 동일한 성격을 가지는 경우에는 하나의 클래스로 모델링한다.

(3) 시스템의 기능과 무관한 클래스

- 문제 기술서, 상식, 전문 지식을 바탕으로 도출된 후보 클래스라고 하더라도 개발할 시스템과 무관한 것들은 제외되어야 한다.
구체적으로 개발할 시스템의 기능과 관계가 있는지의 여부는 해당 클래스에 대해서 추적과 관리할 정보가 있거나 해당 클래스가 시스템의 기능에 기여하는 어떤 행위를 제공하는지에 달려 있다.

(4) 속성

- 독립적인 객체를 나타내기 보다는 다른 대상과 개념의 의미를 상세화하는 역할을 하는 경우에는 클래스가 아니라 해당 클래스의 속성으로 표현하는 것이 타당하다.

(5) 연산

- 다른 클래스가 제공하는 행위 즉 클래스의 연산에 해당되는 경우에는 클래스가 아니라 해당 클래스의 연산으로 표현한다.

(6) 구현과 관련된 클래스

- 시스템에 제공할 기능 측면 보다 이를 구현하기 위하여 필요한 것들을 클래스로 파악해서는 안된다. 해당 분야의 사용자 또는 전문가가 이해하지 못하는 것은 구현과 관련된 클래스로 판단할 수 있다.
즉, 개발자가 시스템을 구현하기 위하여 필요한 구현 기술과 관련된 개념이다.

3) 클래스에 대한 검토

- ① 오직 하나의 대상과 개념만을 나타내고 있는가
- ② 구체적이며 명확한 이름을 가지고 있는가
- ③ 높은 응집도와 낮은 결합도를 가지는가

2. 속성

1) 속성의 정의

- 클래스가 나타내는 객체들이 저장하는 데이터. 즉, 속성은 객체에 대한 정보를 기술하기 위하여 사용

2) 속성 표현

- 클래스 다이어그램에서 속성은 해당 클래스의 이름 하단 부분에 기술된다. 한 클래스는 여러 개의 속성을 가질 수 있으며, 각 속성에 대해서는 가시성, 이름, 다중성 타입, 초기값을 구체적으로 기술할 수 있다.

클래스 이름
속성 1
속성 2
속성 3
.....

구체적인 정보를 기술하는 표현법

[가시성]이름[: 타입][다중성][=초기값]

① 가시성

- 정보은닉을 지원하는 역할을 제공

가시성 유형	표현 기호	설명
공용	+	클래스에 접근할 수 있는 모든 부분에서 공용 속성에 대한 접근은 가능하다.
전용	-	클래스의 연산에서만 전용 속성은 이용될 수 있다.
보호	#	클래스의 연산을 포함하여 하위 클래스(subclass) 연산에서도 보호 속성은 접근 가능하다.
패키지	~	클래스와 동일한 패키지 내부에서는 패키지 유형 가시성의 속성은 접근이 가능하다.

② 이름

- 속성이 나타내는 정보를 잘 드러낼 수 있는 가장 구체적이고 명확한 명사 단어를 사용해야 함.
- 명확한 이해를 위하여 표준화된 약어 이외에는 줄여서 기술하지 않도록 한다.

③ 다중성

- 속성이 복수 개임을 표현하는 방법

④ 타입

- 속성에 저장될 수 있는 값의 유형과 적용할 수 있는 연산을 결정

⑤ 초기값

- 클래스로부터 생성된 객체에 대해서 해당 속성의 초기값을 지정하는 것

3) 속성 찾기

(1) 객체의 정보

- 각 클래스에 대해서 해당 객체에 대한 정보 또는 상태를 찾아보는 것이 속성을 도출하는 첫 작업이다.
즉, "해당 객체에 대해서 알아야 할 정보가 무엇인가" 라는 질문에 대한 답을 생각하는 방식으로 클래스의 속성을 찾을 수 있다.

대상 클래스	설명
Text 클래스	텍스트 도형은 문자열 에 대한 입력을 허용하며, 글꼴 이름, 글꼴 크기, 색상 등을 변경할 수 있다.
FreeDrawing 클래스	자유선은 사용자가 마우스를 자유롭게 드래킹하여 결정된 각 점을 연결하는 직선을 연결할 점의 집합 을 뜻한다. 선의 굵기, 색상 을 변경할 수 있다.
Line 클래스	직선은 사용자가 선택한 두 점을 연결하는 직선으로서 선의 굵기, 색상 을 변경할 수 있다.
Canvas 클래스	각 캔버스는 일련번호 를 가지며, 사용자는 일련번호를 지정함으로써 특정 캔버스로 바로 이동할 수 있다.

(2) 객체의 유용한 정보

- 클래스의 속성은 관점에 따라서 많은 속성이 도출될 수 있다. 그러나 클래스의 용도는 이로부터 객체를 생성하여 구축할 시스템에서 활용하는 것이므로 클래스에 대해서 파악될 수 있는 모든 속성이 아니라 유용한 속성 즉 시스템의 개발 시 필요한 속성만을 도출해야 한다.

(3) 속성과 객체

- 속성은 값 자체를 의미하고 독립적인 존재로서의 의미는 없다.
하나의 대상을 속성으로 간주할지 또는 객체로 간주할지는 시스템에 대한 요구사항에 따라서 달라질 수 있다. 한 대상에 대해서 이를 속성으로 간주할지 클래스로서 간주할지를 판단하는 간단한 방법은 그 대상에 대해서 두 개 이상의 정보를 부

여할 수 있다면 클래스로서 간주하고 하나의 정보만을 필요로 한다면 속성으로 볼 수 있다.

(4) 객체 고유의 정보

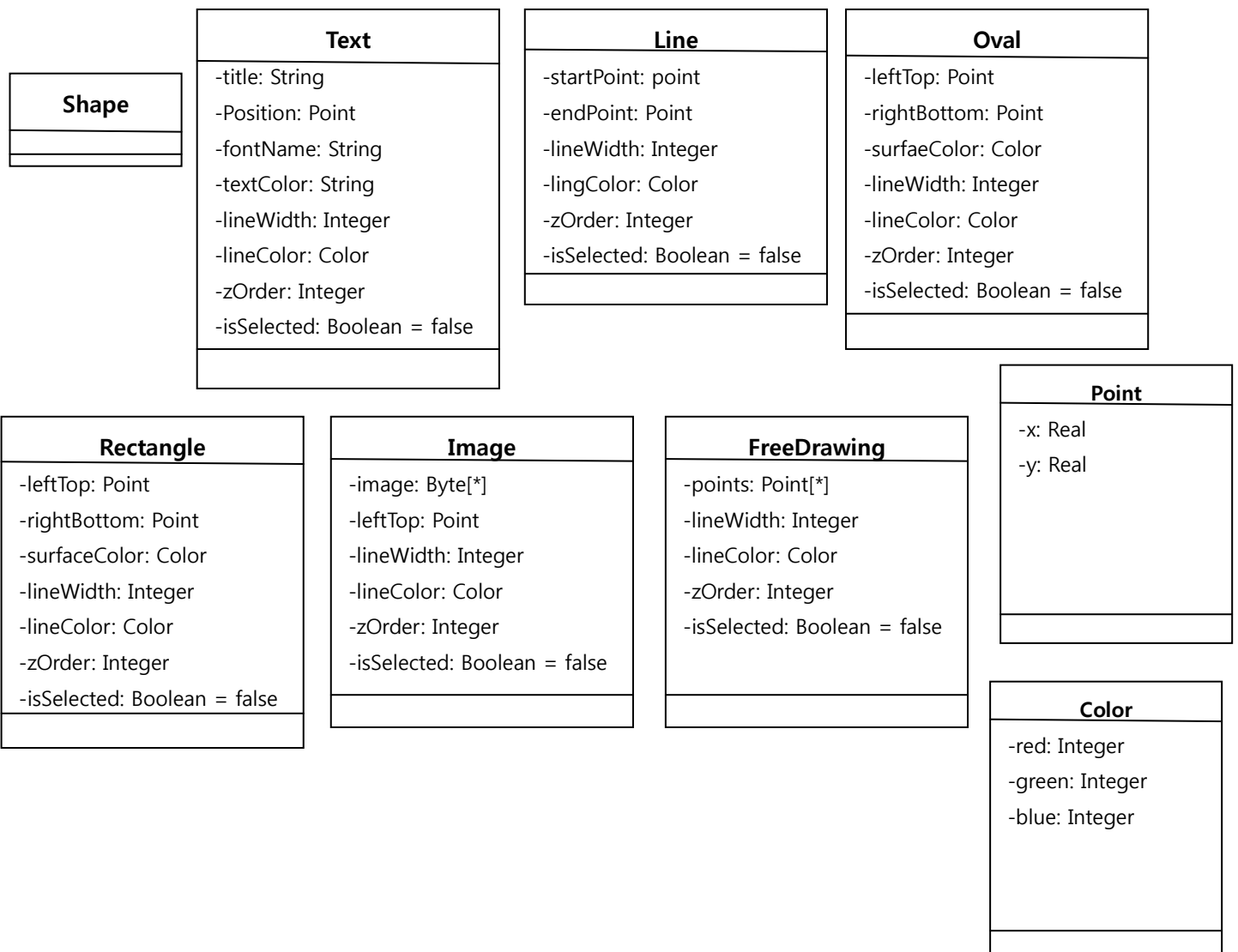
- 한 클래스의 속성은 다른 클래스의 존재 여부와 무관하게 독립적인 의미를 가져야 한다.

(5) 속성과 아이디

- 객체는 기본적으로 고유하다. 각 개체는 특별한 방법을 사용하지 않는다 하더라도 구분이 된다. 따라서 각 객체를 구분하기 위한 아이디를 속성으로서 기술할 필요가 없다.

(6) 속성 간의 관련성

- 클래스의 각 속성은 동일한 하나의 대상과 객체를 기술하는 정보 항목이다. 따라서 속성들 사이에는 밀접한 관련이 있어야 한다.
만약 한 클래스를 구성하는 여러 속성들 사이에 관련성이 낮으면 부적절하게 속성 또는 클래스를 정의한 것에 해당된다.



클래스 이름	속성 설명
Text	Title: 캔버스에 표시되는 텍스트
	Position: 텍스트가 표시되는 좌표
	fontName: 글꼴의 이름
	fontSize: 글꼴의 크기
	textColor: 텍스트 색상
	zOrder: 배치 순서, 0 이상의 값으로서 큰 값의 도형이 보당 위에 배치된다.
	isSelected: 선택 여부, 사용자가 도형을 클릭함으로써 현재 도형이 선택된 상태인지를 나타낸다.
Line	startPoint: 직선의 한 쪽 끝점의 좌표
	endpoint: 직선의 다른 한 쪽 끝점의 좌표
	lineWidth: 선의 굵기
	lineColor: 선의 색상
	zOrder: 배치 순서
	isSelected: 선택 여부
Oval	leftTop: 타원의 외접 사각형의 좌측 상단 좌표
	rightBottom: 타원의 외접 사각형의 우측 하단 좌표
	lineWidth: 선의 굵기
	lineColor: 선의 색상
	surfaceColor: 면의 색상
	zOrder: 배치 순서
	isSelected: 선택 여부
Rectangle	leftTop: 사각형의 좌측 상단 좌표
	rightBottom: 사각형의 우측 하단 좌표
	lineWidth: 선의 굵기
	lineColor: 선의 색상
	surfaceColor: 면의 색상
	zOrder: 배치 순서
	isSelected: 선택 여부
Image	Image: 이미지 데이터
	zOrder: 배치 순서
	isSelected: 선택 여부
	leftTop: 이미지의 좌측 상단 좌표
	lineWidth: 테두리 선의 굵기
	lineColor: 테두리 선의 색상
FreeDrawing	Points: 자유선을 구성하는 일련의 점의 좌표

	lineWidth: 선의 굵기
	lineColor: 선의 색상
	zOrder: 배치 순서
	isSelected: 선택 여부

CanvasManager
-canvasCount: Integer
-canvases: Canvas[*]
-clipboard: Clipboard

Canvas
-seqNumber: Integer
-shapeCount: Integer = 0
-prevEditingCmd: CanvasEditingCommand

Clipboard
-shapes: Shape[*]

CanvasEditingCommand
-interval: Integer
-command: String
-ard: String

BroadcastingManager
-isBroadcastingOn: Boolean = false
-userCount: Integer
-loggedUsers: UserInfo[*]

UserInfo
-id: String
-password: String

Recorder
-isRecordingOn: Boolean = false

Player
-delayRate: Real

[화이트보드 시스템 클래스 속성]

4) 속성에 대한 검토

- ① 각 속성은 오직 하나의 정보만을 나타내고 있는가
- ② 구체적이며 명확한 이름을 가지고 있는가
- ③ 한 클래스의 속성들 사이에 높은 응집도를 가지는가

3. 연산

1) 연산의 정의

- 클래스의 연산은 클래스에 의해서 추상화된 객체가 제공하는 행위를 나타낸다.
즉, 객체의 행위가 추상화된 것이 바로 클래스의 연산이다.
그러므로 실 세계에서 객체가 다른 객체에게 제공하는 각 행위는 해당 클래스의 연산으로 표현된다.

2) 연산의 표현

- 클래스 다이어그램에서 연산은 해당 클래스의 가장 하단 즉 속성 아래 부분에 기술된다. 한 클래스는 여러 개의 연산을 가질 수 있으며, 각 연산에 대해서는 가시성, 이름, 인자, 반환 타입을 구체적으로 기술할 수 있다.

클래스 이름
속성 1
속성 2
속성 3
...
연산 1
연산 2
연산 3
...

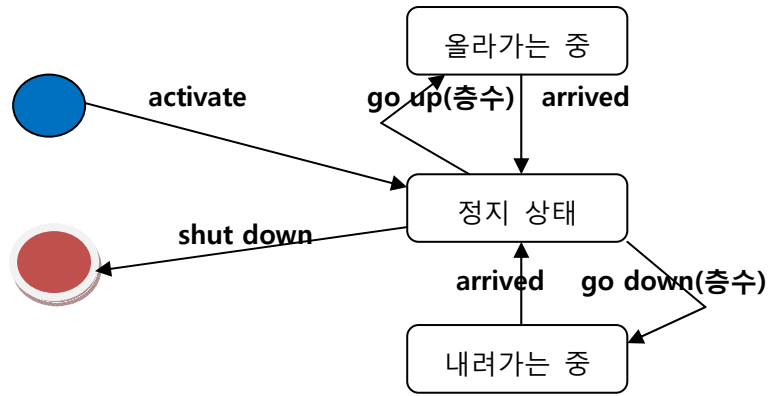
구체적인 정보를 기술하는 표현법

[가시성]이름(인자1:타입,인자2:타입,...):반환타입

3) 연산 찾기

(1) 상태에 바탕을 둔 방법

- 상태는 객체가 존재할 수 있는 가능한 조건 중의 하나라고 정의된다.
한 객체는 한 순간에 오직 하나의 상태에만 머물 수 있고, 시간이 경과된 후에는 다른 상태에 있을 수도 있다.
UML에서는 상태차트 다이어그램을 이용하여 객체가 가질 수 있는 상태 및 상태 간의 전이를 표현할 수 있다.



(2) 속성에 바탕을 둔 방법

- 클래스의 속성은 객체에 대한 상태 정보를 간접적으로 표현하는 것이며, 정보은닉의 개념 때문에 속성에 대한 직접적인 접근은 허용되지 않는다. 따라서 클래스의 속성을 조회하고 조작하는 기능이 연산으로서 정의될 수 있다. 요약하면, 클래스의 속성에 바탕을 둔 연산 도출에서는 각 속성값을 단순히 조회하고 갱신하는 접근자를 정의하거나, 속성값을 바탕으로 복잡한 계산을 하고 조작하는 연산을 정의할 수 있다.

(3) 책임에 바탕을 둔 방법

- 클래스의 연산은 다른 객체에 의해서 요구되는 행위를 나타낸다. 따라서 연산 도출하는 가장 좋은 방법은 해당 객체에 대해서 다른 객체가 어떤 기능을 요구하는지를 조사하는 것이다. 즉, 객체가 외부의 다른 객체에게 제공할 서비스와 기능이 무엇인지를 조사하고 이를 클래스의 연산으로서 정의하면 된다.

(4) 연산의 이름 규칙

- 클래스의 이름, 속성의 이름과 마찬가지로 연산에 적절한 이름을 주는 것은 매우 중요하다. 연산의 수행 방법 또는 과정이 아니라 결과를 명확하게 나타낼 수 있는 구체적 이름을 사용해야 한다.

p.121 [화이트보드 시스템 클래스의 연산]

4) 연산에 대한 검토

- ① 각 연산은 오직 하나의 기능만을 나타내고 있는가
- ② 구체적이며 명확한 이름을 가지고 있는가
- ③ 한 클래스의 연산들끼리 높은 응집도를 가지는가

4. 가시성과 정보은닉

1) 가시성

- UML에서는 클래스의 구성 요소 즉 속성과 연산에 대해서 가시성을 설정할 수 있다. 즉, 클래스의 속성 및 연산에 대한 접근의 가능 여부를 지정할 수 있다.

가시성 유형	표현 기호	설명
공용	+	클래스에 접근할 수 있는 모든 부분에서 공용 속성과 공용 연산에 대한 접근은 가능하다.
전용	-	동일 클래스의 연산에서만 전용 속성과 전용 연산은 이용될 수 있다.
보호	#	동일 클래스의 연산을 포함하여 하위 클래스(subclass)의 연산에서도 보호 속성과 보호 연산은 접근 가능하다.

2) 정보은닉

- 모듈의 근본적인 기능이 아니라 모듈을 설계하는 과정에서 내려진 인위적인 판단이 모듈의 사용자에게 노출되서는 안 되는 것을 뜻한다.
구체적으로 말하면, 모듈의 사용자는 모듈의 이름, 인자, 반환값에 의해서 모듈을 이해하고 사용하게 된다.
즉, 모듈의 이름, 인자, 반환 타입은 오직 모듈의 근본적인 기능에만 관여가 되어야 한다.

3) 속성과 정보은닉

- 클래스의 속성은 클래스가 제공할 연산을 구현하기 위하여 필요한 데이터 및 데이터의 구조로서 이는 설계 상의 판단 결과다.
따라서 속성은 설계 상의 판단에 영향을 미쳤던 요소들이 변경되는 경우에는 속성 자체도 당연히 변경될 수 있다. 그래서 변경 가능성이 높은 속성은 정보은닉 원칙을 적용하여 전용으로 선언함으로써 변경이 발생하여도 클래스의 외부로 파급되는 것을 방지하였다.

접근자 : 속성값을 접근할 때는 속성값을 접근하기 위한 목적으로 정의된 연산을 이용하면 된다. 일반적으로 속성값을 조회하고 갱신하기 위한 용도로 정의된 연산을 접근자라고 부른다.

4) 전용 연산

- 동일한 클래스의 연산들에 의해서만 이용되는 연산은 전용으로 선언한다.
- 이와 같은 연산을 공용으로 선언하여도 프로그램의 수행 상의 문제점은 없다.
- 그러나 클래스의 외부에서 이용될 필요가 없는 연산을 공용으로 선언하여 노출시킬 경우 클래스 이용자에게는 의미가 없는 연산을 이용해야하는 부담을 줄 수 있다.

5. 클래스 간의 관계

1) 연관 관계

(1) 연관 관계의 의미

- 클래스 사이의 가장 일반적인 관계로서 한 클래스가 다른 클래스를 인지함을 의미한다. 연관 관계는 실제 생활에서 한 사람이 다른 사람을 알고 있는 관계에 비유될 수 있다

클래스 다이어그램에서 클래스 사이의 실선으로 표시

연관 관계는 상대 클래스의 객체에게 메시지를 전달할 때 사용되는 통로의 역할을 한다. 즉, 연관 관계를 맺고 있는 클래스의 객체에게만 메시지를 전송하여 그 객체에게 무언가를 요구할 수 있다.

(2) 연관 관계의 방향성

- 연관 관계는 기본적으로 양방향성이다. 화살표가 없는 실선으로 표현된 연관 관계는 양방향을 가지며 이는 두 클래스가 서로 상대 클래스를 인지하는 것이다. 따라서 각 클래스는 상대 클래스에게 모두 메시지를 전송할 수 있다.

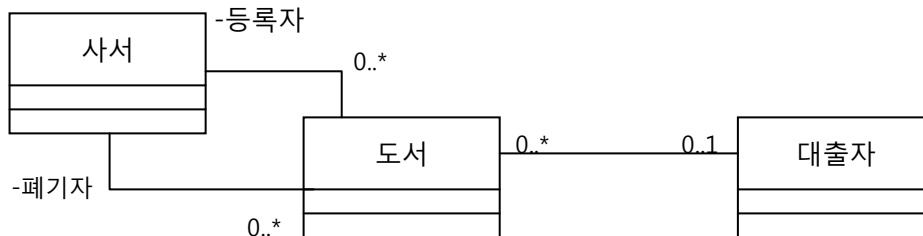
클래스 다이어그램에서 연관 관계의 방향성은 실선의 한쪽 끝에 화살표로 표시

연관의 방향성은 모델링하려고 하는 실제 대상을 그대로 표현하면 된다.

(3) 연관의 이름과 역할

- 연관 관계는 한 클래스가 다른 클래스를 인지하는 것을 나타내므로 매우 광범위한 의미를 가진다. 그러므로 두 클래스 사이의 연관 관계에 대한 의미는 두 클래스 사이의 연관 관계에 대한 의미는 두 클래스로부터 유추해야 한다. 그러나 두 클래스만으로는 연관 관계의 의미를 정확하게 판단하기 어려운 경우가 적지 않다. 따

라서 연관 관계의 의미를 모호하지 않도록 구체적으로 표현할 필요가 있다. UML에서는 연관 이름 또는 역할을 이용하여 연관 관계의 의미를 보다 구체화시킬 수 있다.



다중성 표현	의미
1	한 객체가 연관된다. 표시하지 않아도 되는 기본값이다.
0..1	0개 또는 1개의 객체가 연관된다.
0..*	0개 또는 많은 수의 객체가 연관된다.
*	0..*와 동일
1..*	1개 이상의 객체가 연관된다.
3..5	3개에서 5개까지의 객체가 연관된다.
3..5, 7, 9	3개에서 5개까지 또는 7개 또는 9개의 객체가 연관된다.

(4) 복수 연관

- 동일한 두 클래스 사이에 두 개 이상의 연관 관계가 맺어지는 것을 복수 연관이라고 한다. 복수 연관은 두 클래스가 명확하게 다른 의미의 관계를 맺는 경우에 사용될 수 있다.

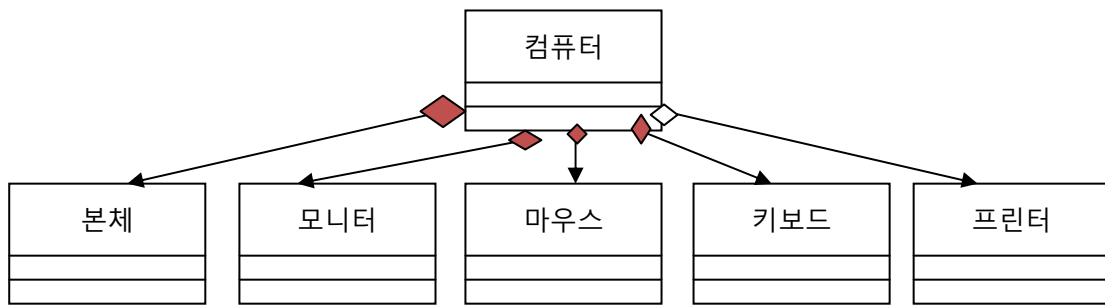
(5) 다중성

- 연관 관계가 있는 두 클래스가 있을 때 실제로 연관을 가지는 객체의 수를 나타낸다. 다중성은 연관 관계의 양쪽 끝에 지정된다.

2) 집합 관계

(1) 집합 관계의 의미

- 집합 관계는 두 객체 간 소속의 의미를 표현할 때 사용한다. 즉, 한 클래스가 전체의 역할을, 반대로 다른 클래스는 부분의 역할을 할 때 사용한다. 집합 관계는 보다 구체적인 의미를 가지는 연관 관계로서 한 객체가 다른 객체를 포함하는 것을 나타낸다.



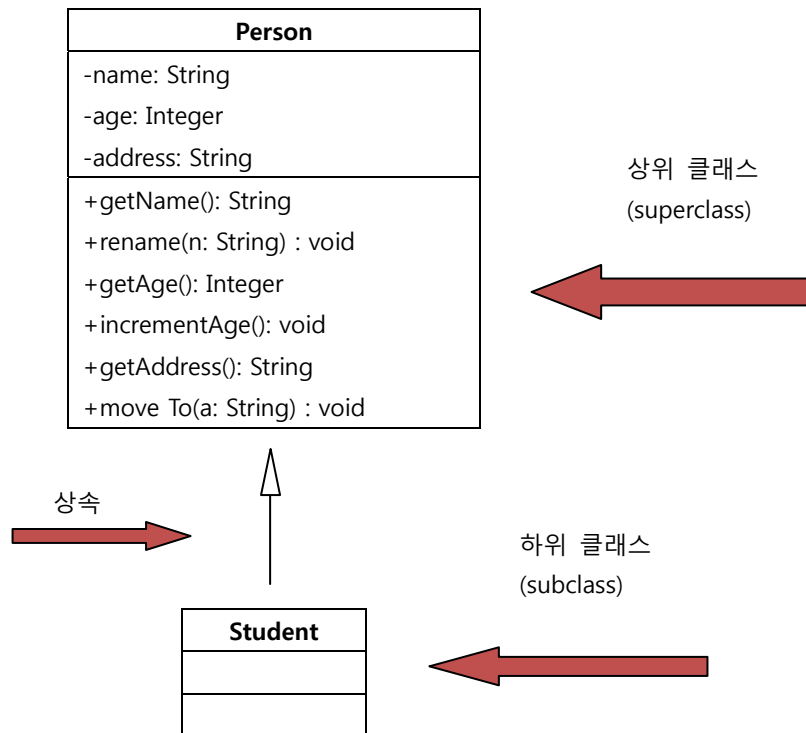
(2) 집합 관계와 포함 관계

표현법		
용어	포함 (composition 또는 composite aggregation)	집합 (aggregation 또는 shared aggregation)
의미	부분 객체는 전체 객체에 전속됨	부분 객체는 여러 전체 객체에 의해서 공유될 수 있음

6. 상속

1) 상속이란

- 상속은 두 클래스 간의 관계로서 클래스(subclass)가 다른 클래스(superclass)의 모든 속성과 연산을 물려받는 것을 말한다.



하위 클래스는 상위 클래스의 모든 속성과 연산을 물려받는 것뿐만 아니라 하위 클래스에서 새롭게 속성과 연산을 추가할 수 있다.

따라서 항상 하위 클래스의 속성과 연산은 상위 클래스의 것보다 많아지게 된다.

2) 서브클래싱

- 하위 클래스가 상위 클래스의 모든 것을 물려받는 것뿐만 아니라 하위 클래스 고유의 특징을 구현하기 위하여 새로운 속성과 연산을 추가하거나 물려받은 연산을 재정의하는 것을 서브클래싱 이라고 한다.

① 속성의 추가

- 하위 클래스는 상위 클래스로부터 속성을 물려받는다. 따라서 이 시점에서 하위 클래스의 속성은 상위 클래스의 속성과 동일하다.
그러므로 하위 클래스가 상위 클래스와의 다른 점 즉 고유의 정보가 하위 클래스에는 없는 상황이다. 따라서 하위 클래스는 상위 클래스로부터 물려받은 속성을 포함하여 상위 클래스와 다른 고유의 상태와 정보를 속성으로서 추가해야 한다.

② 연산의 추가

- 하위 클래스는 상위 클래스로부터 속성을 포함하여 연산을 물려받는다. 즉, 위 클래스가 제공할 수 있는 기능을 하위 클래스도 제공한다.
그리고 하위 클래스는 하위 클래스 고유의 기능을 추가할 수 있다.

③ 상속받은 속성의 초기화

- 상위 클래스로부터 상속받은 속성도 하위 클래스의 객체의 구성하는 한 부분이 된다. 다만, 상위 클래스의 속성과 유사하기 때문에 생략된 것 뿐이며 객체의 관점에서 보면 물려받은 속성과 새로 추가한 속성을 구분하지 않고 모두 함께 자신의 상태를 구성하는 요소가 된다.
그리고 객체는 생성될 때 항상 초기화가 되어야 한다. 즉, 객체의 속성값을 적절히 설정하여 객체의 상태를 초기화 시켜야 한다.
따라서 객체의 초기 상태를 설정하기 위하여 속성값을 적절히 설정하는 일은 생성자에서 수행되어야 한다. 그리고 하위 클래스의 생성자는 자신의 클래스에서 추가한 속성 뿐만 아니라 상위 클래스로부터 물려받은 속성도 함께 초기화해야 한다.

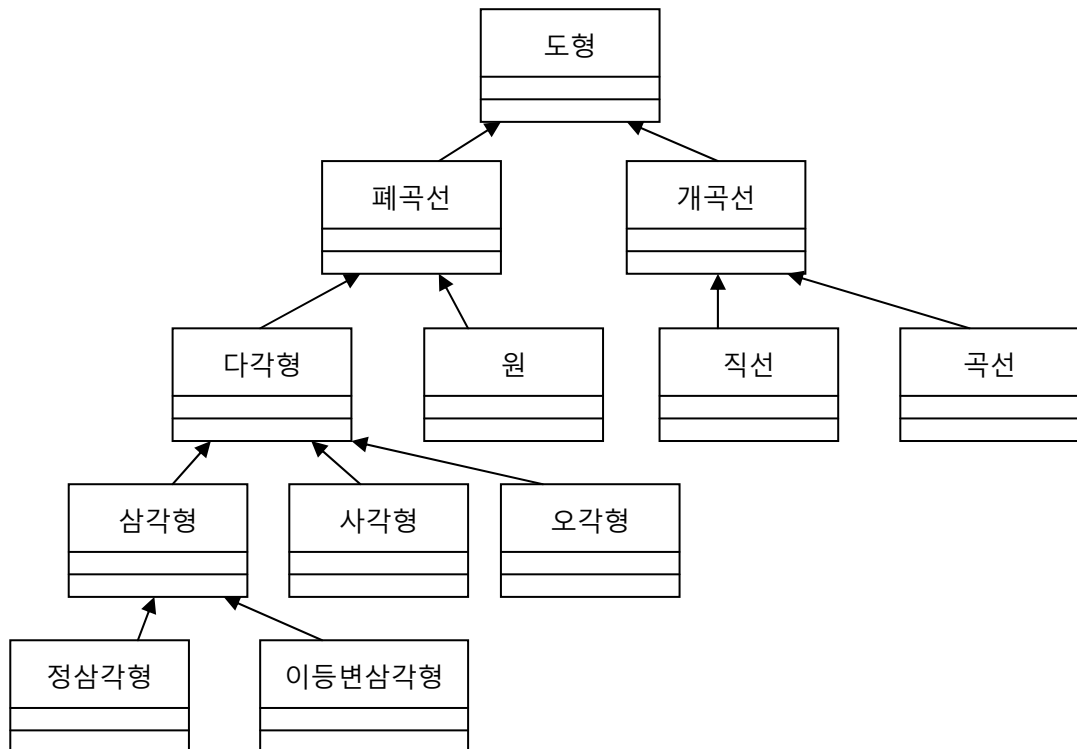
④ 연산의 재정의

- ▶ 상위 클래스로부터 물려받은 연산을 동일한 형식으로 하위 클래스에서 정의함으로써 하위 클래스의 상황에 맞추어 물려받은 연산을 다시 구현하는 것을 말한다. 재정의는 하위 클래스에서 새로운 속성과 연산을 추가하는 방법과 더불어 상속을 이용하여 하위 클래스를 정의할 때 빈번하게 사용되는 방법으로 객체지향 클래스 라이브러리를 구성할 때 매우 중요한 역할을 한다.

3) 일반화와 특별화

(1) 상속 트리

- 여러 클래스 간의 상속 관계를 보여주는 트리 모양의 구조를 말한다.



(2) 일반화

- 상위 클래스는 모든 하위 클래스의 공통적인 속성과 연산만을 가지고 있어야 한다. 그러므로 상위 클래스는 하위 클래스에 비하여 보다 일반적인 대상 또는 개념을 나타낸다고 말할 수 있다. 즉, 상속트리의 상단 방향으로 갈수록 각 클래스가 나타내는 개념은 보다 일반화한다고 볼 수 있다.

(3) 특별화

- 하위 클래스는 상위 클래스의 속성과 연산을 상속받는 것과 함께 새로운 속성과 연산을 하위 클래스에 추가하거나 상속받은 연산을 재정의하는 방법으로, 상위 클래스가 나타내는 개념이 하위 클래스에서 구체화되면서 특별화된다.

상속 트리의 하위 방향으로 갈수록 각 클래스가 나타내는 개념은 보다 특별화한다고 볼 수 있다.

※ Star UML을 사용하여 모델링 하기

UseCase 다이어그램 모델링하기


Actor

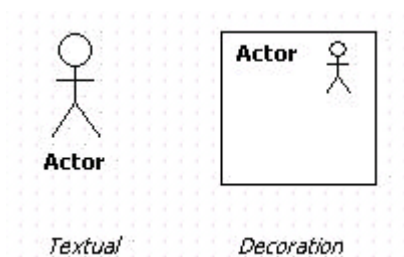
의미:

액터(Actor)는 일반적으로 시스템 외부에 존재하면서 시스템과 상호작용하는 개체입니다. 액터는 사람이거나 기계 혹은 소프트웨어 등이 될 수 있습니다.

Actor 생성 방법:

Actor 를 생성하려면, Toolbox>UseCase 의 Actor 버튼을 클릭하고 Main 윈도우창에서 Actor 가 위치할 곳을 클릭합니다. Actor 는 Stick Man 형태로 표현되지만, 사각형 모양에 오른쪽 상단에 아이콘이 포함된 Decoration View 형태로 사용되기도 합니다. Actor 를 Decoration View 형태로 보여지도록 하기 위해서는

[Format] -> [Stereotype Display] -> [Decoration] 메뉴 아이템을 선택하거나 툴바의 버튼에서 [] 콤보 버튼의 [Decoration] 항목을 선택합니다.




UseCase

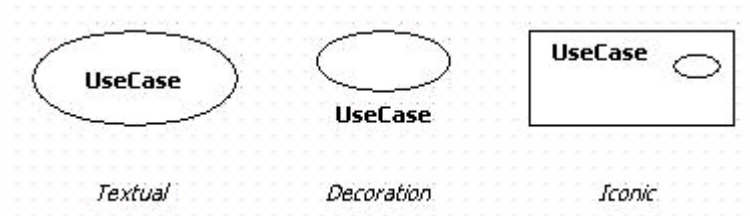
의미:

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

UseCase 를 생성하는 방법:

UseCase 를 생성하려면, [Toolbox] -> [UseCase] -> [UseCase] 버튼을 클릭하고 Main 윈도우창에서 UseCase 가 위치할 곳을 클릭합니다.

UseCase 는 Textual, Decoration, Iconic 의 3 가지 형태로 표현 가능합니다. [Format] -> [Stereotype Display]의 하부 메뉴 아이템을 선택하거나 [] 버튼의 아이템을 선택하면, UseCase 의 스타일을 변경할 수 있습니다.



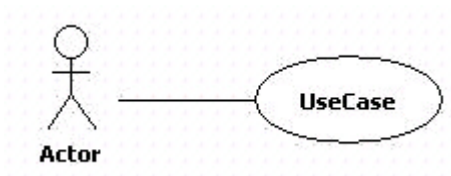
Association / Directed Association

의미:

연관(Association)은 클래스류(Class, Interface, Enumeration, Signal, Exception, Component, Node, UseCase, Actor) 사이의 의미적 관계를 정의합니다.

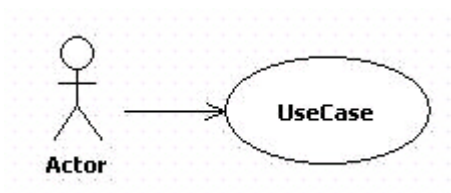
Association 생성하는 방법:

Association 를 생성하려면, [Toolbox] -> [UseCase] -> [Association] 버튼을 클릭하고 Main 윈도우창에서 연결하려는 첫번째 요소에서 두번째 요소로 마우스를 누르고 드래그하면 됩니다.

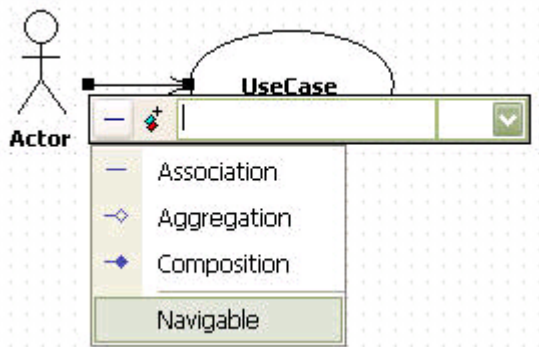


DirectedAssociation 생성하는 방법:

Association 생성방법과 동일하며, 두 요소간 마우스 드래그를 화살표 방향으로 합니다.



또는 Association 을 생성하고 Actor 쪽 association 의 끝을 클릭하고 Quick Dialog 의 Navigable 의 체크를 취소하면 DirectedAssociation 으로 변합니다.



Generalization

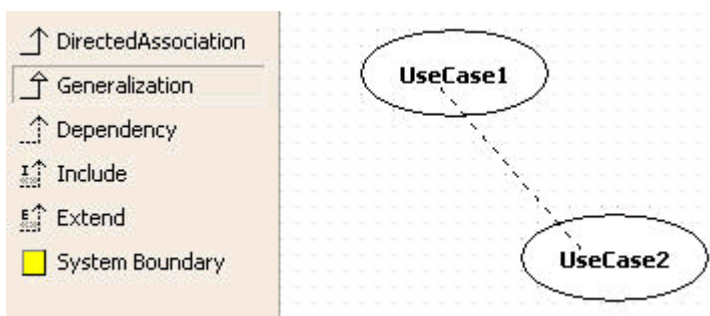
의미:

일반화(Generalization)">일반화(Generalization)는 더 일반적인 요소와 더 구체적인 요소를 연결하는 관계입니다.

Generalization 생성하는 방법:

Procedure for creating generalization

Generalization 를 생성하려면, [Toolbox] -> [UseCase] ->[Generalization] 버튼을 클릭하고 Main 윈도우창에서 연결하려는 자식 요소에서 부모 요소로 마우스를 누르고 드래그하면 됩니다.



Dependency

의미:

의존관계(Dependency)">의존관계(Dependency)는 어떤 요소의 구현이나 기능을 위해 다른 요소의 존재가 요구 되어지는 의존적인 관계를 의미합니다.

Dependency 생성 방법:

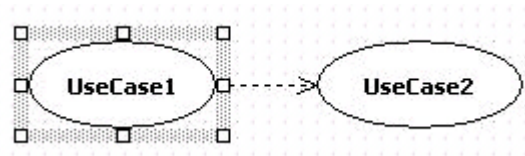
Dependency 를 생성하려면, [Toolbox] -> [UseCase] ->[Dependency] 버튼을 클릭하고 Main 윈도우창에서 요소에서 의존 하는 요소로 마우스를 누르고 드래그하면 됩니다.

UseCase 로부터 의존하는 다른 UseCase 생성하는 방법:

쿼디어로로그의 단축 생성구문을 다음과 같이 입력하면 됩니다.



그러면 다음과 같이 두 요소간의 Dependency가 생성됩니다.



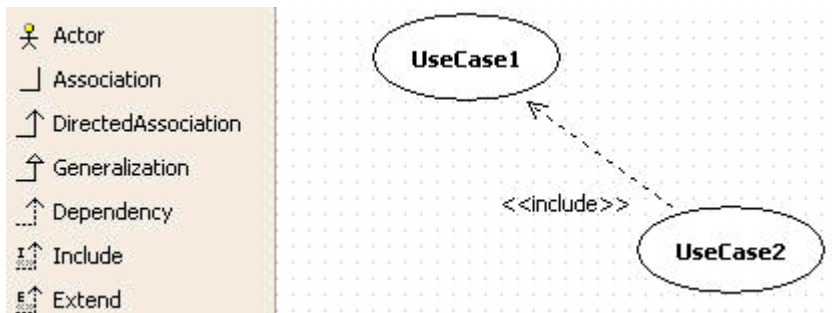
Include

의미:

포함관계(Include)는 어떤 유스케이스가 특정 유스케이스의 행위를 포함한다는 것을 정의합니다.

Include 생성 방법:

Include를 생성하려면, [Toolbox] -> [UseCase] -> [Include] 버튼을 클릭하고 Main 윈도우창에서 요소에서 포함할 요소로 마우스를 누르고 드래그하면 됩니다.



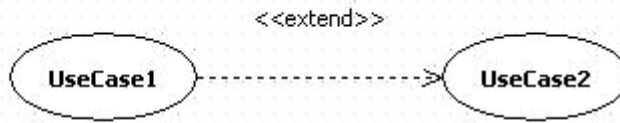
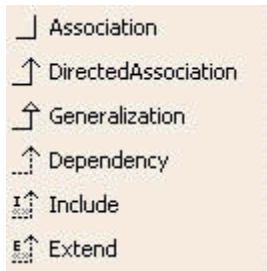
Extend

의미:

확장관계(Extend)">확장관계(Extend)는 어떤 유스케이스가 특정 유스케이스에 정의된 행위로 추가 확장될 수 있다는 것을 나타냅니다.

Extend 생성 방법:

Extend를 생성하려면, [Toolbox] -> [UseCase] -> [Extend] 버튼을 클릭하고 Main 윈도우창에서 요소에서 확장할 요소로 마우스를 누르고 드래그하면 됩니다.



System Boundary

System Boundary 를 생성하는 방법:

System Boundary 를 생성하려면, [Toolbox] -> [UseCase] -> [System Boundary] 의 System Boundary 버튼을 클릭하고 Main 윈도우창에서 System Boundary 가 삽입될 위치에 마우스를 클릭하고 생성될 크기 만큼을 드래그합니다.

Activity 다이어그램 모델링

ActionState

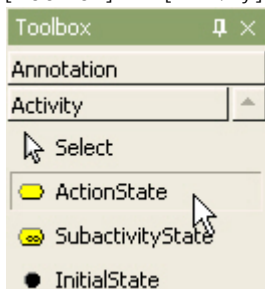
의미:

활동상태(ActionState)">활동상태(ActionState)는 하나의 진입-액션(Entry Action)을 가지고 있는 상태(State)입니다. 활동은 개념적으로 하나의 활동, 작업 등을 표현하며 활동 다이어그램에서 표현되어지는 요소입니다.

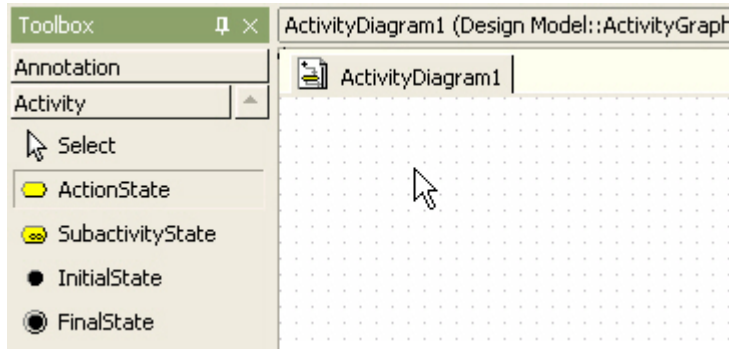
생성 방법:

ActionState 를 생성하려면,

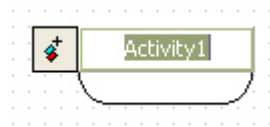
1. [Toolbox] -> [Activity] -> [ActionState] 버튼을 클릭하고



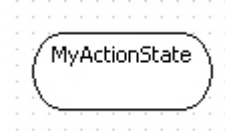
2. Main 윈도우창에서 ActionState 가 위치할 곳을 클릭합니다.



3. action state 가 생성되어지고 쿼다이얼로그가 나타납니다.



4. action state 의 이름을 쿼다이얼로그에서 입력하고 [Enter] 키를 누르면 다음과 같이 action state 가 보여집니다.



SubactivityState

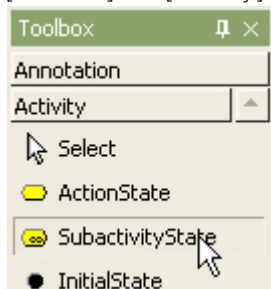
의미:

하위-활동상태(SubactivityState)">하위-활동상태(SubactivityState)는 하나의 액티비티그래프(ActivityGraph)를 호출합니다. 하위-활동상태로 수행이 진입하면 해당 활동그래프(ActivityGraph)가 수행을 시작하고 수행이 종료되면 하위-활동상태의 수행이 종료된 것으로 간주됩니다.

생성 방법:

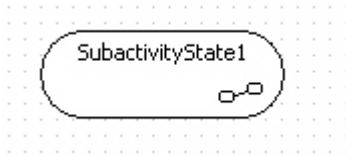
SubactivityState 를 생성하려면,

1. [Toolbox] -> [Activity] -> [SubactivityState] 버튼을 클릭하고



2. Main 윈도우창에서 SubactivityState 가 위치할 곳을 클릭합니다. subactivity state 가 다이어그램상에 생성되어지고 쿼다이얼로그가 나타나면 subactivity state 의 이름을 입력하고

[Enter] 키를 누릅니다.

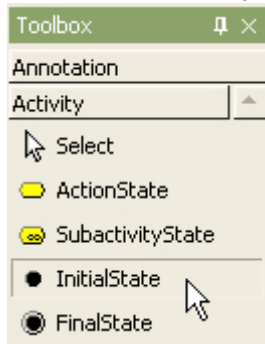


InitialState

생성 방법:

InitialState 를 생성하려면,

1. [Toolbox] -> [Activity] -> [InitialState] 버튼을 클릭하고



2. Main 윈도우창에서 InitialState 가 위치할 곳을 클릭합니다.



FinalState

생성 방법:

FinalState 를 생성하려면,

1. [Toolbox] -> [Activity] -> [FinalState] 버튼을 클릭하고



2. Main 윈도우창에서 FinalState 가 위치할 곳을 클릭합니다.

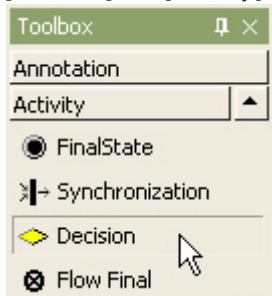


Decision

생성 방법:

Decision 를 생성하려면,

1. [Toolbox] -> [Activity] -> [Decision] 버튼을 클릭하고



2. Main 윈도우창에서 Decision 가 위치할 곳을 클릭합니다. 그러면 다음과 같이 decision 이 다이어그램에 생성됩니다.

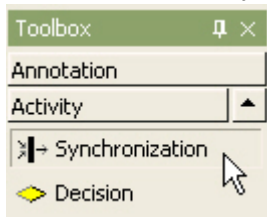


Synchronization

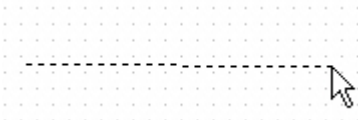
생성 방법:

Synchronization 를 생성하려면,

1. [Toolbox] -> [Activity] -> [Synchronization] 버튼을 클릭하고



2. Main 윈도우창에서 Synchronization 가 위치할 곳을 클릭하고 원하는 크기만큼 드래그합니다.



3. 그러면 다음과 같이 synchronization 이 다이어그램에 생성됩니다.

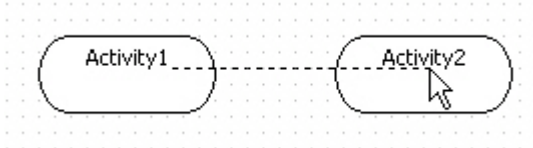


Transition

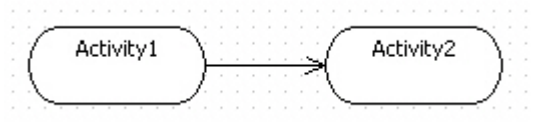
Transition 생성 방법:

Transition 를 생성하려면,

1. [Toolbox] -> [Activity] -> [Transition] 버튼을 클릭하고
2. Main 윈도우창에서 State 에서 전이하는 방향의 다른 State 로 마우스를 누르고 드래그하면 됩니다.



3. 그러면 transition 이 생성됩니다.



Class 다이어그램 모델링하기

Subsystem

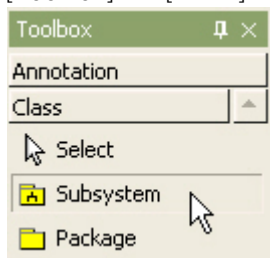
의미:

서브시스템(Subsystem)">서브시스템(Subsystem)은 물리적인 시스템의 부분 혹은 전체를 명세화하기 위해 요소들을 그룹화하는 요소입니다.

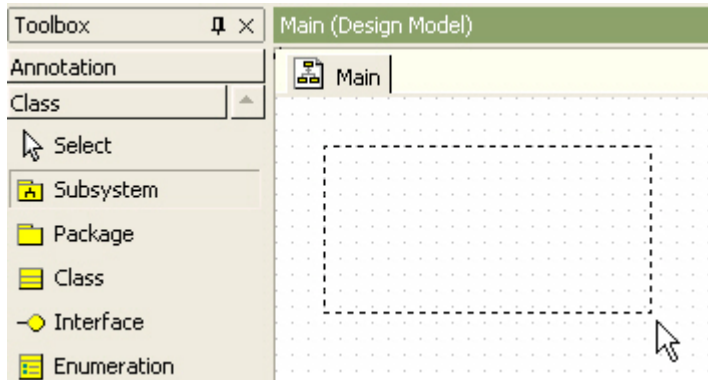
Subsystem 생성하는 방법:

Subsystem 을 생성하려면,

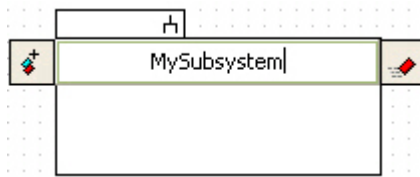
1. [Toolbox] -> [Class] -> [Subsystem] 버튼을 클릭하고



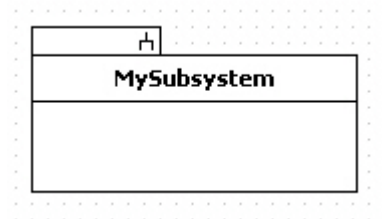
2. Main 윈도우창에서 Subsystem 가 위치할 곳을 클릭하고 원하는 크기만큼 드래그합니다.



3. 그러면 subsystem 이 class diagram 상에 생성되고 쿼다이얼로그가 나타납니다.



4. 쿼다이얼로그에서 subsystem 의 이름을 입력하고 [Enter] 키를 누릅니다.



Class

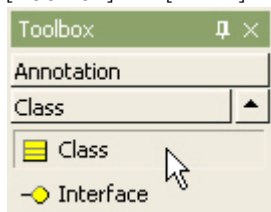
의미:

클래스(Class)>클래스(Class)는 객체의 구조와 행위를 묘사하는 속성(Attribute)과 연산(Operation)의 집합을 선언하는 요소입니다. 그리고 클래스는 템플릿 파라미터(Template Parameter)를 가질 수 있습니다.

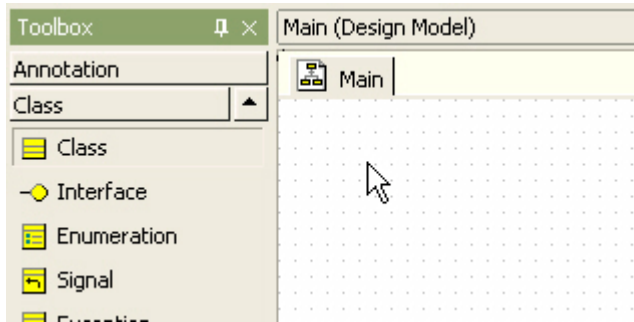
Class 생성하는 방법:

Class 를 생성하려면,

1. [Toolbox] -> [Class] -> [Class] 버튼을 클릭하고



2. Main 윈도우창에서 Class 가 위치할 곳을 클릭합니다.



3. 새로운 class 가 다이어그램상에 생성되고, 쿼다이얼로그가 나타납니다.
4. 쿼다이얼로그에서 class 의 이름을 입력하고 [Enter] 키를 누릅니다.



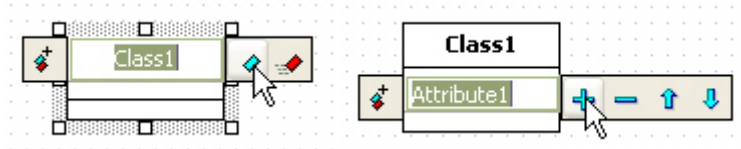
Attribute 추가하는 방법:

Class 에 attribute 를 추가하는 방법은 다음과 같이 3 가지 방법이 있습니다.

- Quick Dialog 를 이용
- Class 또는 Model Explorer 의 팝업 메뉴 이용
- Collection Editor 이용

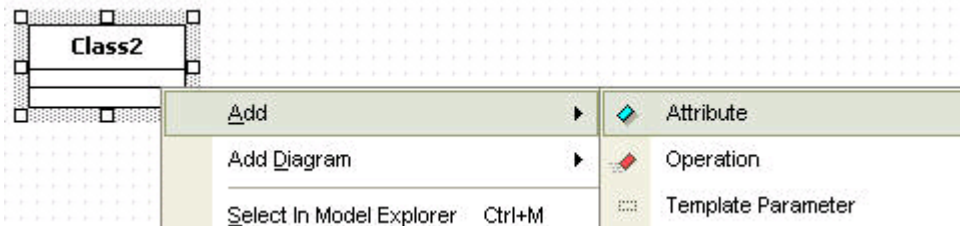
Quick Dialog 를 이용하는 경우

1. class 를 더블클릭합니다.
2. [Add Attribute] 버튼을 누릅니다.



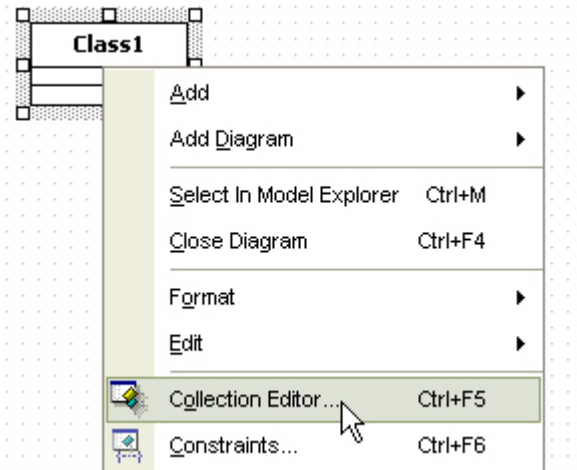
Class 또는 Model Explorer 의 팝업 메뉴 이용하는 경우

1. main window 또는 model explorer 에서 class 를 선택합니다.
2. 그리고 마우스 오른쪽 클릭을 통해서 [Add] -> [Attribute] 팝업 메뉴를 선택하여 attribute 를 추가합니다.

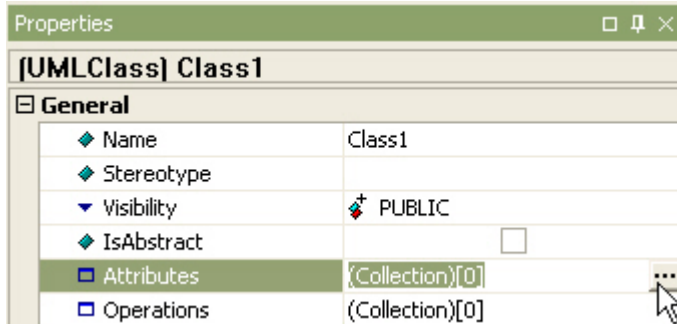



Collection Editor 이용하는 경우에는

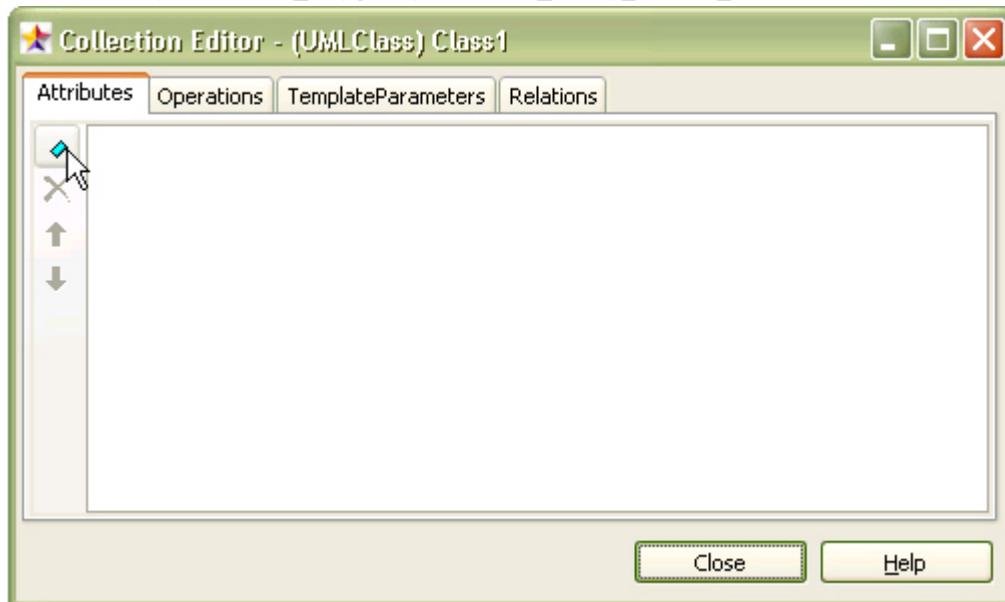
1. Class 의 [Collection Editor...] 팝업 메뉴를 선택합니다.



2. 또는 Properties 윈도우의 Attributes 의 ... 편집 버튼을 통해서 Collection Editor 를 열어서

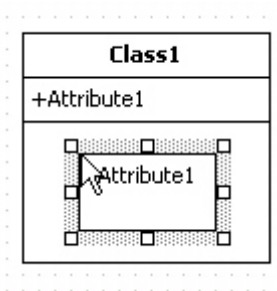
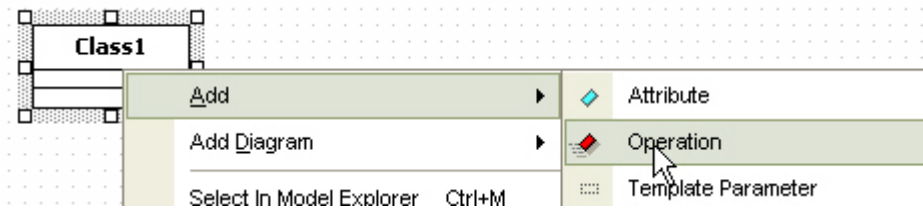


3. Attributes 탭에서  버튼을 이용하여 Attribute 를 입력할 수 있습니다.



Operation 추가하는 방법:

Class 또는 Model Explorer 의 팝업 메뉴 이용하는 경우에는 Main 윈도우 또는 Model Explorer 에서 Class 를 선택하고 오른쪽 마우스 버튼을 눌러서 [Add] -> [Operation] 팝업 메뉴를 선택하여 Operation 를 입력할 수 있습니다.

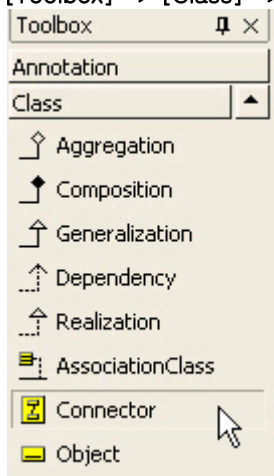


1.

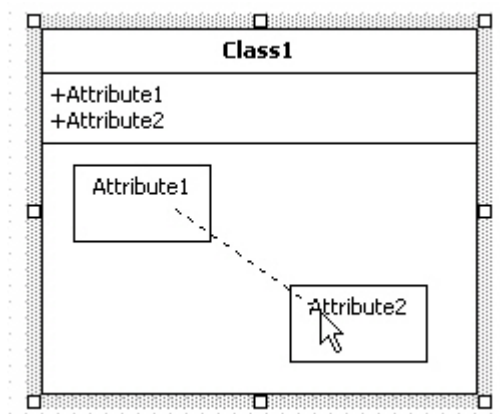
Connector 생성 방법:

Connector 를 생성하려면,

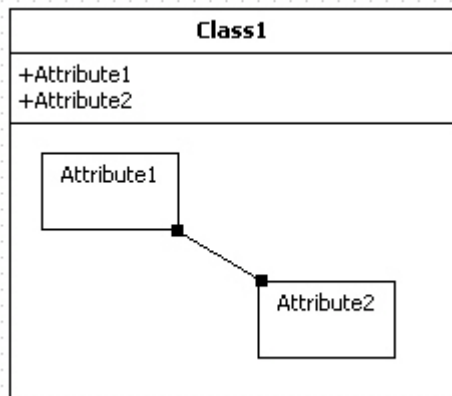
1. [Toolbox] -> [Class] -> [Connector] 버튼을 클릭하고



2. Main 윈도우에서 Part 에서 연결할 Part 로 마우스를 누르고 드래그하면 됩니다.



3. 그러면 두개의 part 사이에 다음과 같이 connector 가 생성됩니다.



Interface

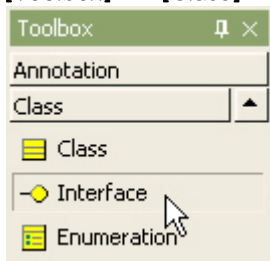
의미:

인터페이스(Interface)">인터페이스(Interface)는 클래스에 의해 제공되는 서비스를 구성하는 연산들을 포함하는 요소입니다. 또한 연산들을 효과적인 그룹으로 나누고 그것들을 특징지을 수 있는 방법을 제공합니다. 인터페이스에서부터 객체가 생성될 수 없습니다.

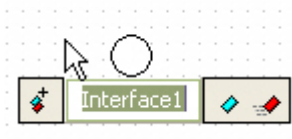
Interface 을 생성하는 방법:

Interface 를 생성하려면,

1. [Toolbox] -> [Class] -> [Interface] 버튼을 클릭하고



2. Main 윈도우창에서 Interface 가 위치할 곳을 클릭하면 쿼다이얼로그가 나타납니다.



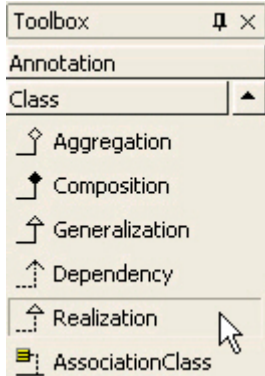
3. 쿼다이얼로그에서 interface 의 이름을 입력하고 [Enter] 키를 누르면 다음과 같이 interface 가 생성됩니다.



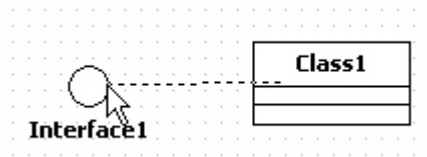
Interface Providing 관계 생성 방법:

Interface Providing 관계를 설정하려면,

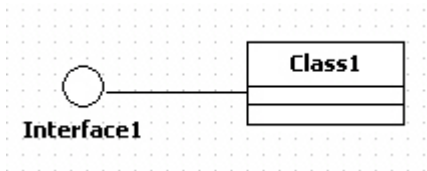
1. [Toolbox] -> [Class] -> [Realization] 버튼을 클릭하고



2. Main 윈도우창에서 요소(Class, Port, Part, Package, Subsystem)를 선택하고 Interface 로 마우스를 누르고 드래그하면 됩니다.



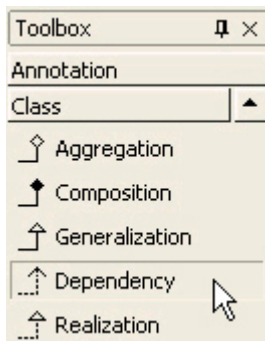
3. 그러면 providing interface 관계가 다음과 같이 생성됩니다.



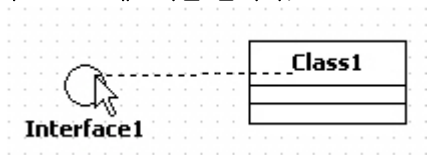
Interface Requiring 관계 생성 방법:

Interface Requiring 관계를 설정하려면,

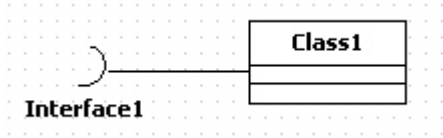
1. [Toolbox] -> [Class] -> [Dependency] 버튼을 클릭하고



2. Main 윈도우창에서 요소(Class, Port, Part, Package, Subsystem)를 선택하고 Interface 로 마우스를 누르고 드래그하면 됩니다.



3. 그러면 requiring interface 관계가 다음과 같이 생성됩니다.



Signal

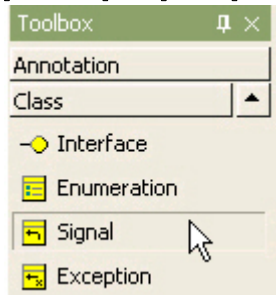
의미:

시그널(Signal)은 객체간의 비동기적(asynchronous) 통신 신호에 대한 명세(specification) 입니다.

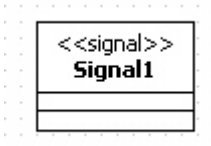
Signal 생성 방법:

Signal 을 생성하려면,

1. [Toolbox] -> [Class] -> [Signal] 버튼을 클릭하고



2. Main 윈도우창에서 Signal 이 위치할 곳을 클릭합니다.



Exception

의미:

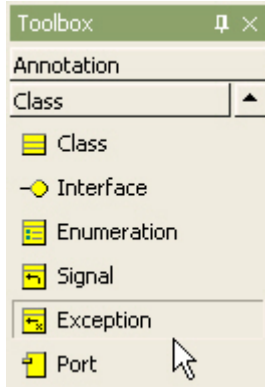
예외(Exception)">예외(Exception)는 실행 오류시에 연산(Operation)에 의해 발생하는 시그널(Signal)입니다.</

Exception 생성 방법:

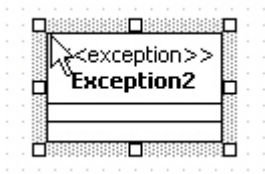
Procedure for creating exception

Exception 을 생성하려면,

1. [Toolbox] -> [Class] -> [Exception] 버튼을 클릭하고



2. Main 윈도우창에서 Exception 이 위치할 곳을 클릭합니다.



Association

의미:

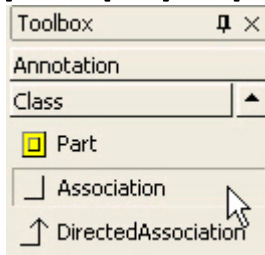
연관(Association)">연관(Association)은 클래스류(Class, Interface, Enumeration, Signal, Exception, Component, Node, UseCase, Actor) 사이의 의미적 관계를 정의합니다.현재의 연관-끝(AssociationEnd) 방향의 포함관계를 의미합니다. (- NONE: 집합이 아님을 나타냄, - AGGREGATE: 집합을 나타냄, - COMPOSITE: 합성을 나타냄)

Association 생성 방법:

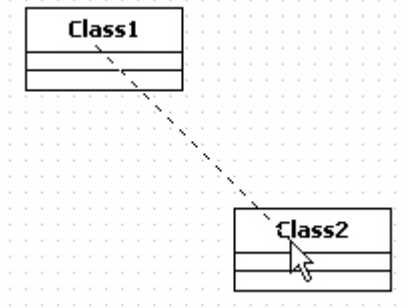
Procedure for creating association

Association 를 생성하려면,

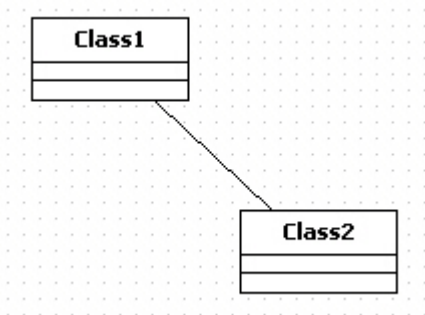
1. [Toolbox] -> [Class] -> [Association] 버튼을 클릭하고



2. Main 윈도우창에서 요소에서 연관할 요소로 마우스를 누르고 드래그하면 됩니다.



3. 두개의 class 사이에 새로운 association 이 다음과 같이 생성됩니다.

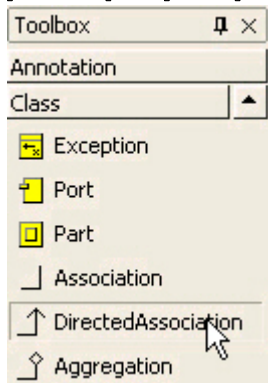


DirectedAssociation

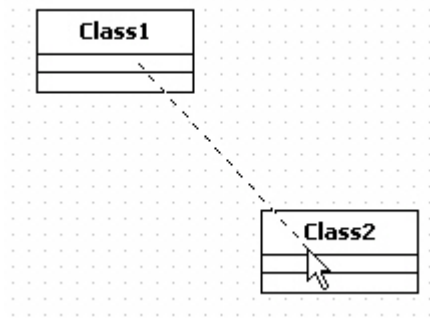
DirectedAssociation 생성 방법:

Association 생성방법과 동일합니다.

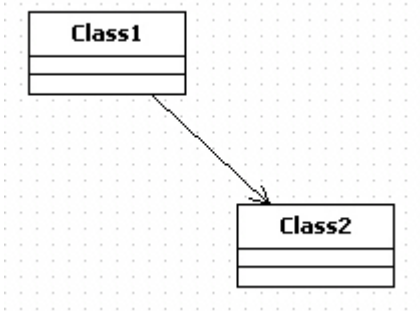
1. [Toolbox] -> [Class] -> [DirectedAssociation] 버튼을 클릭합니다.



2. 두개의 요소사이에 화살표 방향으로 드래그합니다.

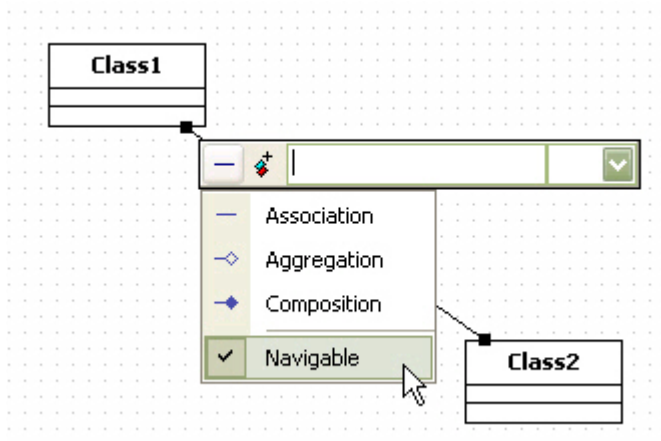


3. 결과는 다음과 같습니다.



Association 으로부터 DirectedAssociation 으로 변경 방법: Procedure for changing association to directed association

Association 을 생성하고 화살표 반대편쪽 association 의 끝을 클릭하고 Quick Dialog 의 Navigable 의 체크를 취소하면 DirectedAssociation 으로 변합니다.

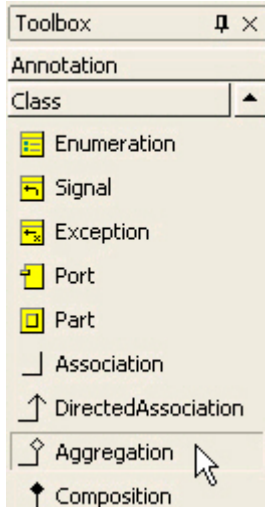


Aggregation

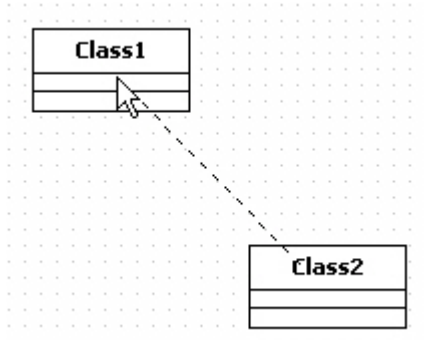
Aggregation 생성 방법:

Aggregation 를 생성하려면,

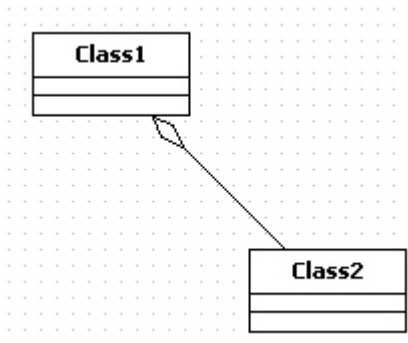
1. [Toolbox] -> [Class] -> [Aggregation] 버튼을 클릭하고



2. Main 윈도우창에서 포함되어지는 요소에서 포함하는 요소로 마우스를 누르고 드래그하면 됩니다.



3. 결과는 다음과 같습니다.



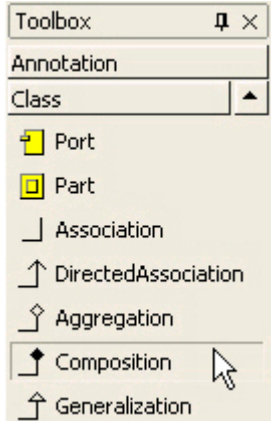
Composition

Composition 생성 방법:

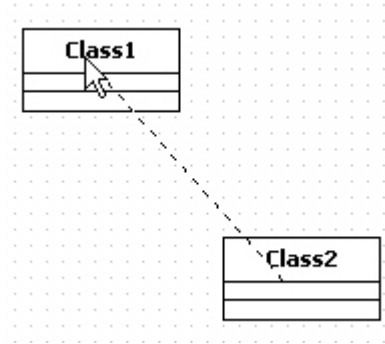
Procedure for creating composition

Composition 를 생성하려면,

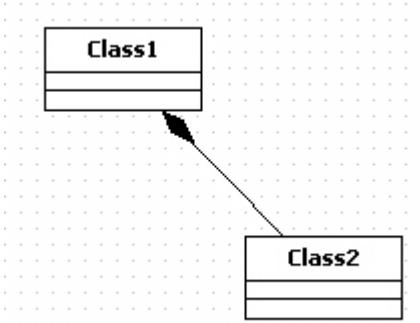
1. [Toolbox] -> [Class] -> [Composition] 버튼을 클릭하고



2. Main 윈도우창에서 포함되어지는 요소에서 포함하는 요소로 마우스를 누르고 드래그하면 됩니다.



3. 그러면 두개의 class 사이에 새로운 composition 관계가 다음과 같이 생성됩니다.



Generalization

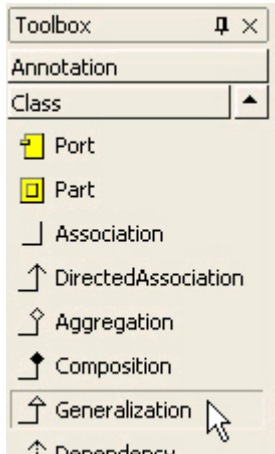
의미:

일반화(Generalization)">일반화(Generalization)는 더 일반적인 요소와 더 구체적인 요소를 연결하는 분류학적 관계입니다.

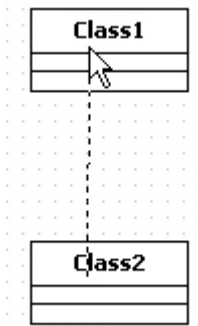
Generalization 생성 방법:

Generalization 를 생성하려면,

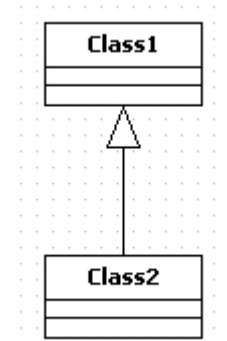
1. [Toolbox] -> [Class] -> [Generalization] 버튼을 클릭하고



2. Main 윈도우창에서 자식 요소에서 부모 요소로 마우스를 누르고 드래그하면 됩니다.



3. 그러면 새로운 generalization 이 다음과 같이 생성되어집니다.



Dependency

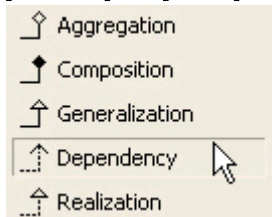
의미:

의존관계(Dependency)">의존관계(Dependency)는 어떤 요소의 구현이나 기능을 위해 다른 요소의 존재가 요구 되어지는 의존적인 관계를 의미합니다.

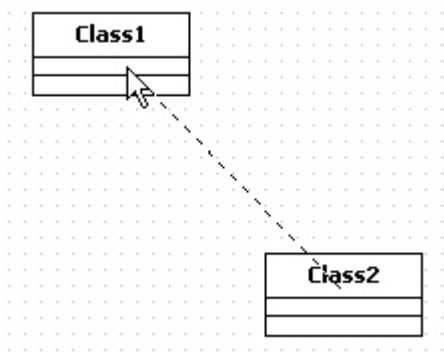
Dependency 생성 방법:

Dependency 를 생성하려면,

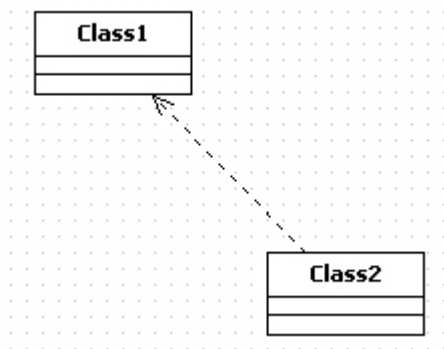
1. [Toolbox] -> [Class] -> [Dependency] 버튼을 클릭하고



2. Main 윈도우창에서 요소가 의존하는 다른 요소 방향으로 마우스를 누르고 드래그하면 됩니다.



3. 그러면 두개의 요소사이에 dependency가 생성됩니다.



Realization

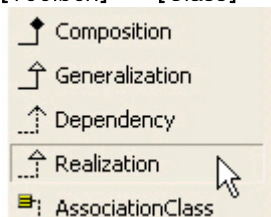
의미:

실체화(Realization)">실체화(Realization)는 명세(specification) 요소와 그것을 구현하는(implementation) 요소의 실체화 관계를 정의합니다. 주로 인터페이스(Interface)와 그것을 구현하는 요소(클래스, 컴포넌트 등)를 연결하는데 사용합니다.

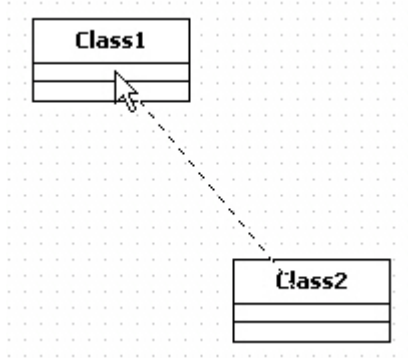
Realization 생성 방법:

Realization 를 생성하려면,

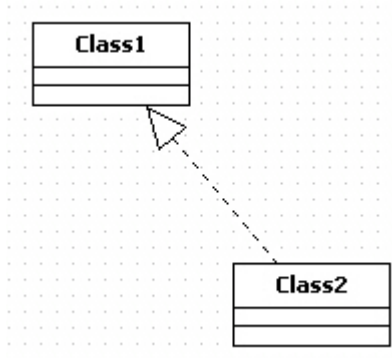
1. [Toolbox] -> [Class] -> [Realization] 버튼을 클릭하고



2. Main 윈도우창에서 요소가 Realize 의 부모 요소 방향으로 마우스를 누르고 드래그하면 됩니다.



3. 결과는 다음과 같습니다.



AssociationClass

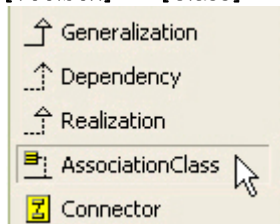
의미:

연관클래스(AssociationClass)">연관클래스(AssociationClass)는 클래스(Class)와 연관(Association)을 연결하여 연관자체가 클래스의 의미도 가질 수 있도록 하는 연결 고리 역할을 합니다.

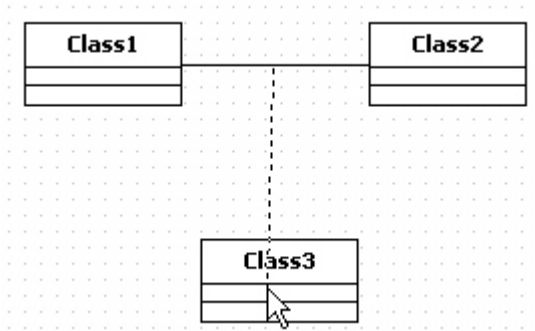
AssociationClass 생성 방법:

AssociationClass 를 생성하려면,

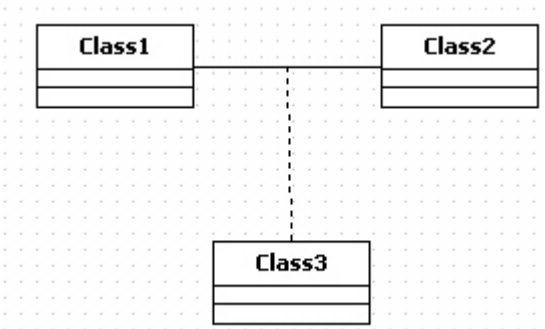
1. [Toolbox] -> [Class] -> [AssociationClass] 버튼을 클릭하고



2. Main 윈도우에서 Association 에서 AssociationClass 가 되는 Class 방향으로 마우스를 누르고 드래그하면 됩니다.



3. 결과는 다음과 같습니다.



Object

의미:

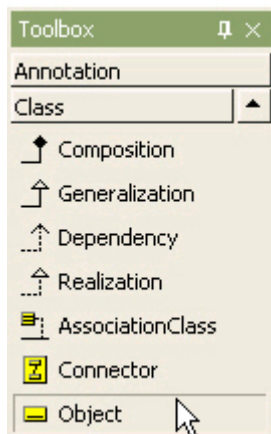
객체(Object)는 특정 클래스의 인스턴스(instance)입니다.

Object 생성 방법:

Procedure for creating object

Object 를 생성하려면,

1. [Toolbox] -> [Class] -> [Object] 버튼을 클릭하고



2. Main 윈도우창에서 Object 가 위치할 곳을 클릭합니다.




Object 에 AttributeLink 추가 방법:

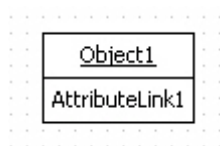
Object 에 AttributeLink 를 추가하는 방법은 다음과 같이 2 가지 방법이 있습니다.

- Object 또는 Model Explorer 의 팝업 메뉴 이용
- Collection Editor 이용

Object 또는 Model Explorer 의 팝업 메뉴 이용하는 경우에는 Main 윈도우 또는 Model Explorer 에서 Object 를 선택하고 오른쪽 마우스 버튼을 눌러서 [Add] -> [Attribute Link] 팝업 메뉴를 선택하여 Attribute Link 를 추가할 수 있습니다.



Collection Editor 이용하는 경우에는 Object 의 [Collection Editor...] 팝업 메뉴 또는 Properties 윈도우의 Slots 의 편집 버튼을 통해서 Collection Editor 를 열어서 Slots 탭에서  버튼을 이용하여 Attribute Link 를 입력할 수 있습니다.



Link

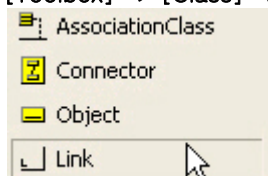
의미:

링크(Link)는 객체사이의 연결(connection)입니다.

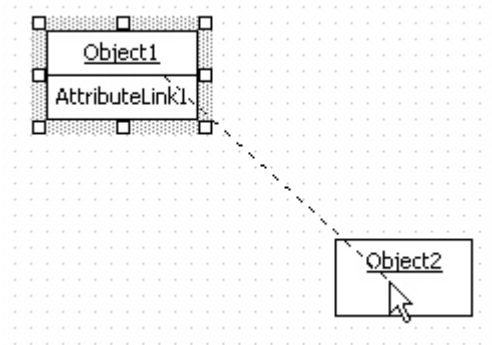
Link 생성 방법:

Link 를 생성하려면,

1. [Toolbox] -> [Class] -> [Link] 버튼을 클릭하고



2. Main 윈도우창에서 Link 할 Object 에서 다른 Object 방향으로 마우스를 누르고 드래그하면 됩니다.



3. 결과는 다음과 같습니다.

