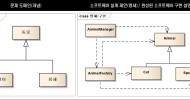
http://www.nextree.co.kr/p6753/

[UML: 클래스 다이어그램과 소스코드 매핑](http://www.nextree.co.kr/p6753/)

**Posted by**[**강 형구**](http://www.nextree.co.kr/author/hkkangnextree-co-kr/)**in**[**배움터 - 열공**](http://www.nextree.co.kr/category/study/)**,**[**일터 - 경험과 노하우**](http://www.nextree.co.kr/category/knowhow/)on Mar 8th, 2014

[](http://www.nextree.co.kr/p6753/)

불과 몇 년 되지 않은 학생 시절… 처음으로 UML을 접했고, UML의 기초적인 그리는 법과 사용법을 배웠습니다. 개인적으로 쉽지 않은 수업이었는데 그 중 가장 많이 사용되는 클래스 다이어그램에서 클래스간의 relationship(관계)가 제일 어려웠습니다.Generalization(일반화)과 Realization(실체화)은 명확하기 때문에 이해하는데 어려움이 없었고 Dependency(의존) 부터 조금 어려워 지더니 Association(연관)과 Aggregation(집합), Composition(합성) 3종 세트에 가서는 머리가 복잡해졌습니다. 특히 Aggregation과 Composition이 논리적으로 약하고 강한 집합이라는 차이는 알겠지만 ‘그래서 어떤 명확한 차이점이 발생하는 거지? 저걸 코드로 만들 때는 어떻게 만들어야 하는 거지?’라는 생각과 함께 잘 와 닿지 않았습니다.

그래서 2013년 9월에 진행 됐던 ’2013 넥스트리 신입사원 인큐베이션 세미나’에서 발표 주제를 UML 클래스 다이어그램으로 정하고 그 동안 배운 것들과 추가로 공부를 더한 것들로 발표를 했고, 많은 도움이 됐던 시간이었습니다. 세미나 발표 후 몇가지 내용에 대한 아쉬움이 있었는데, 이번 기회에 그 아쉬움과 부족했던 부분을 보완하여 글로 정리하게 되었습니다. UML 클래스 다이어그램의 기본적인 요소들과 클래스간의 관계, 그리고 제가 제일 어려웠던 이 관계들을 어떻게 코드로 나타내어야 할지에 대해서 썼으며 언어는 Java 기준으로 하였습니다. [’2013 신입사원 인큐베이션 세미나’](http://www.nextree.co.kr/p4738)에 대한 글도 Nextree 홈블로그에서 보실 수 있습니다.

**1. UML**

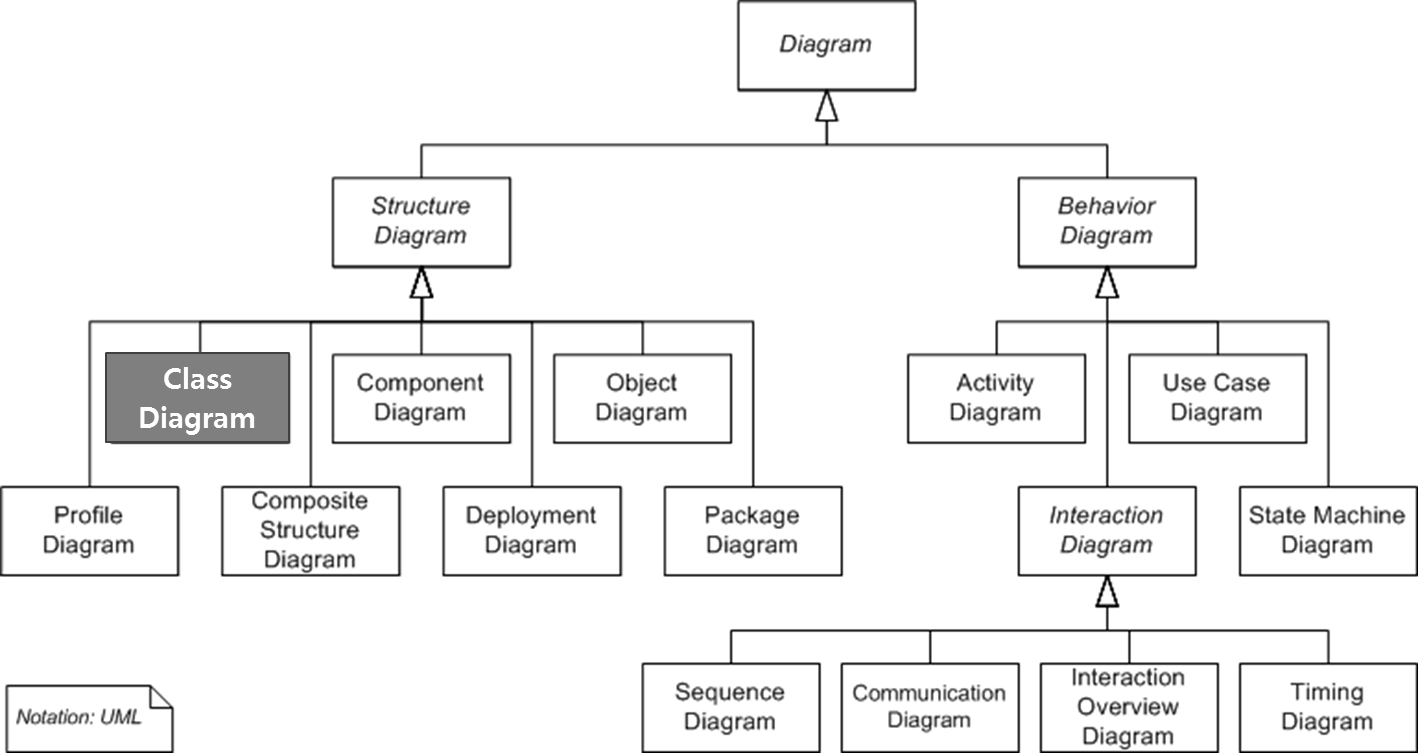
UML이란 Unified Modeling Language의 약자로 1997년 OMG(Object Management Group)에서 표준으로 채택한 통합모델링언어 입니다. 즉, 모델을 만드는 표준언어인 것입니다. 모델이란 것은 어떤 것을 실제로 만들 때 이렇게 만들면 잘 작동할지 미리 검증해 보는 것이며 실제 물건을 만드는 비용보다 비용이 훨씬 적을 경우에 모델을 만들어 설계를 검사합니다.

소프트웨어에서의 모델은 건축, 항공 등의 모델과는 좀 다른 면이 있습니다. 건물을 짓고, 항공기를 만드는 것과 설계를 그리고 만드는 것은 비용의 엄청난 차이가 있습니다. 하지만 UML 다이어그램을 그리며 모델을 만드는 일은 개발보다 비용이 적긴 하지만 훨씬 적게 드는 것이 아니며 때로는 오히려 개발보다 비용이 더 많이 들 수도 있습니다. 그래서 UML은 시험해 볼 구체적인 것이 있고, 그것을 코드로 시험해 보는 것보다 UML로 시험해 보는 쪽이 비용이 덜 들 때 주로 사용합니다. 이러한 목적으로 UML을 사용하는 유형에는 다음 3가지 정도가 있습니다.

* 다른 사람들과의 의사소통 또는 설계 논의
* 전체 시스템의 구조 및 클래스의 의존성 파악
* 유지보수를 위한 설계의 back-end 문서

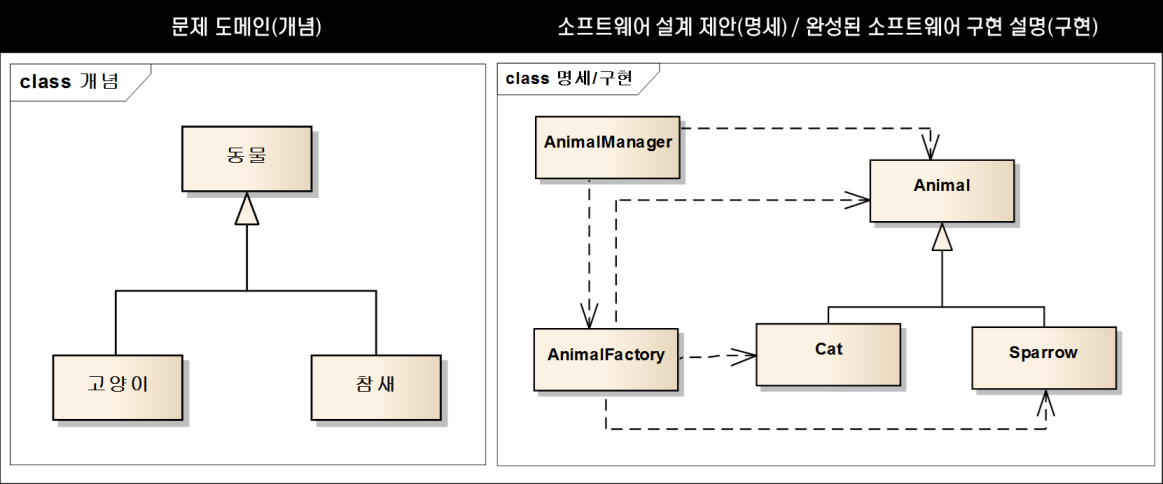
UML을 그리는데 가장 좋은 도구는 종이와 펜이라는 말이 있듯이 습관적으로 만드는 게 아니라 필요에 의해 만드는 것이 가장 좋은 것 같습니다.

**2. Class Diagram (클래스 다이어그램)**

[](http://www.nextree.co.kr/wp-content/uploads/2014/02/%EA%B7%B8%EB%A6%BC1-UML-%EC%A2%85%EB%A5%98.png)

[그림 1] UML 다이어그램의 종류  
출처: www.ask.com/wiki/Unified\_Modeling\_Language#UML\_2.x

UML은 구조 다이어그램 7개, 행위 다이어그램 7개로 총 14종류의 다이어그램이 있습니다. 구조 다이어그램은 시스템의 개념, 관계 등의 측면에서 요소들을 나타내고 각 요소들의 정적인 면을 보기 위한 것이고 행위 다이어그램은 각 요소들 혹은 요소들간의 변화나 흐름, 주고받는 데이터 등의 동작을 보기 위한 것으로, 클래스 다이어그램은 구조 다이어그램에 해당합니다.클래스 다이어그램은 클래스 내부의 정적인 내용이나 클래스 사이의 관계를 표기하는 다이어그램으로 시스템의 일부 또는 전체의 구조를 나타낼 수 있습니다. 클래스 다이어그램은 의존 관계를 명확히 보게 해주며, 순환 의존이 발생하는 지점을 찾아내서 어떻게 이 순환 고리를 깨는 것이 가장 좋은지 결정할 수 있게 해줍니다.



[그림 2] 목적 별 클래스 다이어그램

UML은 목적에 따라 다르게 사용되는데, 크게 3가지로 개념, 명세, 구현의 차원들 입니다.먼저 개념 차원의 UML은 문제 도메인의 구조를 나타내며 사람이 풀고자 하는 문제 도메인 안에 있는 개념과 추상적 개념을 기술하기 위한 것입니다. 이것은 소스코드와 관계가 그렇게 깊지 않으며 오히려 사람의 자연언어와 더 관련이 있습니다. 또한 의미론적(언어의 뜻을 규정하는) 규칙에 그다지 얽매이지 않으며, 따라서 의미하는 바도 모호하거나 해석에 따라 달라질 수 있는 부분이 존재합니다.

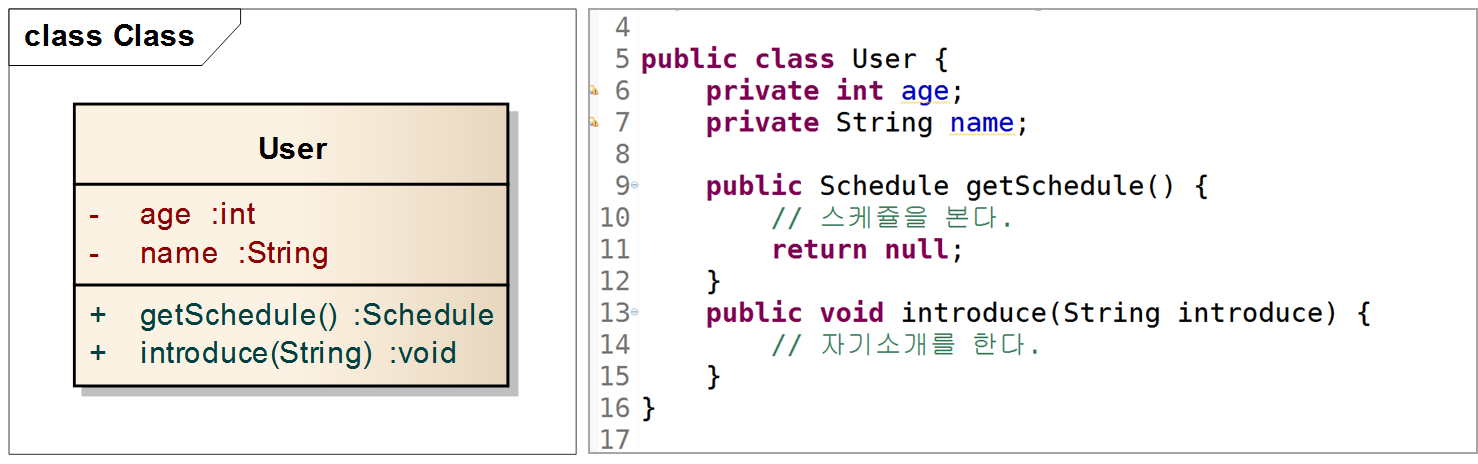
반면, 명세와 구현 차원의 UML은 소프트웨어의 설계 혹은 완성된 소프트웨어의 구현 설명 목적 등으로 사용하며 설계를 해서 소스코드로 바꾸거나 구현 된 소스코드를 설명하려고 사용하기 때문에 소스코드와 관계가 깊습니다. 이 두 차원의 클래스 다이어그램은 제약이 많아서 반드시 일정한 규칙과 의미론을 지켜야 합니다. 또한 모호성이 거의 없도록 하고 형식도 최대한 맞춰야 합니다.

앞으로 나올 내용들은 구조 다이어그램 중 하나인 클래스 다이어그램이며 그 중에서도 개념 차원은 배제하고 소스코드와 관계가 깊은 명세, 구현 차원에 해당하는 클래스 다이어그램만으로 한정 지어 소스코드와 어떻게 연관 되는지 이해해 보도록 하겠습니다. 따라서 개념 차원의 클래스 다이어그램과는 조금 다른 부분이 있을 수 있습니다.

**3. 클래스 다이어그램의 요소(Element)**

**Class (클래스)**

클래스는 보통 3개의 compartment(구획)으로 나누어 클래스의 이름, 속성, 기능을 표기합니다. 속성과 기능은 옵션으로 생략이 가능하지만 이름은 필수로 명시해야 합니다.

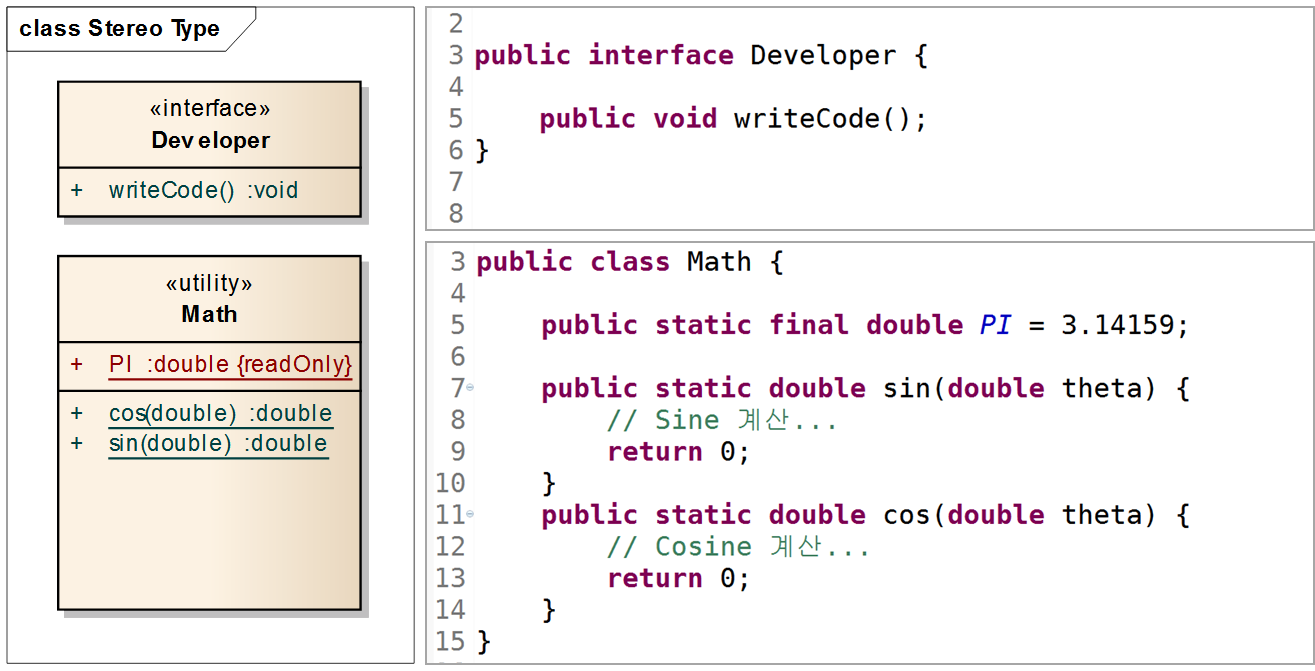


[그림 3] 클래스

클래스의 세부사항은 필드와 메서드의 Access modifier(접근제한자), 필드명(메서드명), 데이터타입, parameter(매개변수), 리턴 타입 등을 나타낼 수 있습니다. 클래스의 세부사항들을 상세하게 적는 것이 유용할 때도 있지만, UML 다이어그램은 필드나 메서드를 모두 선언하는 곳이 아니기 때문에 다이어그램을 그리는 목적에 필요한 것만 사용하는 것이 좋습니다.추가로 보통 3개의 구획(compartment)을 사용 하지만 다른 미리 정의되거나 사용자 정의 된 모델 속성(비즈니스 룰, 책임, 처리 이벤트, 발생된 예외 등)을 나타내기 위한 추가 구획도 사용할 수 있다고 합니다.

**Stereo Type (스테레오 타입)**

스테레오 타입이란 UML에서 제공하는 기본 요소 외에 추가적인 확장요소를 나타내는 것으로 쌍 꺾쇠와 비슷하게 생긴 길러멧(guillemet, « ») 사이에 적습니다. 이 길러멧이란 기호는 쌍 꺾쇠와는 좀 다른 것으로 폰트 크기보다 작습니다. 종이나 화이트보드에 그릴 때는 상관없지만 공식적인 문서라면 이 기호를 구분해서 사용하는 것이 좋을 것 같습니다.

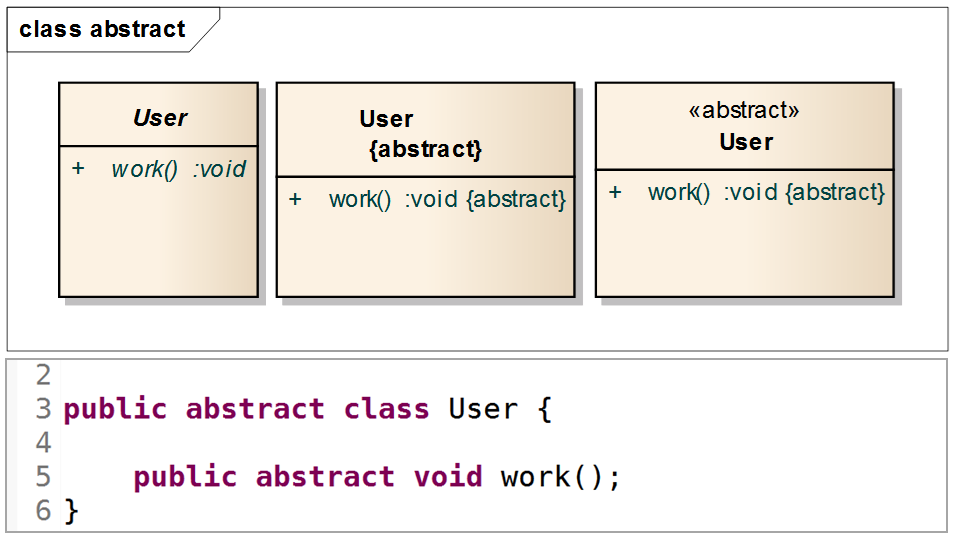


[그림 4] 스테레오 타입

위의 다이어그램은 인터페이스와 유틸리티 클래스를 표현하고 있으며 필드, 메소드 밑의 밑줄은 static(정적)필드 또는 메서드를, {readOnly}는 final 키워드를 사용하는 상수를 의미합니다. 스테레오 타입으로 많이 사용되는 것은 «interface», «utility», «abstract», «enumeration» 등이 있습니다.

**Abstract Class/Method (추상 클래스 / 메서드)**

추상클래스란 1개 이상의 메서드가 구현체가 없고 명세만 존재하는 클래스를 말합니다.



[그림 5] 추상클래스

추상 클래스의 이름과 메서드는 italic체나, {abstract} 프로퍼티를 사용하여 표기합니다. UML 툴에서는 italic체로 표기하는 것이 많지만 종이나 칠판에 그릴 때는 힘들게 italic체로 기울여서 적는 것 보다는 {abstract} 프로퍼티로 표기하는 것이 쉽고 명확할 것 같습니다.  
또한 공식적인 것은 아니지만 스테레오타입을 사용하여 추상 클래스를 표기하기도 합니다.

**4. 클래스간의 관계**

클래스 다이어그램의 주 목적은 클래스간의 관계를 한눈에 쉽게 보고 의존 관계를 파악하는 것에 있습니다. 그렇기 때문에 클래스 다이어그램에서 가장 중요한 것이 클래스간의 관계입니다.

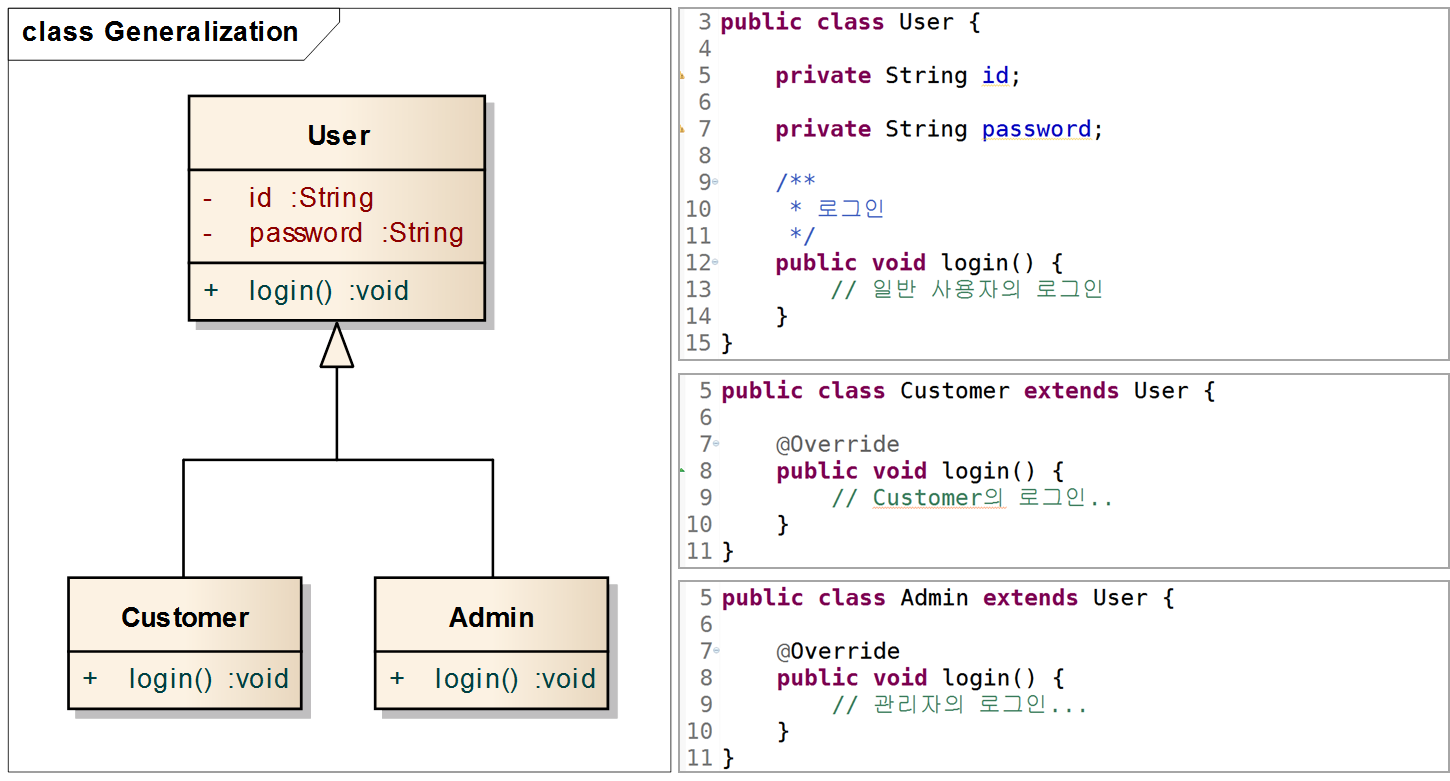


[그림 6] 클래스 관계 종류

위의 그림은 클래스간의 관계들의 종류와 표기법을 나타낸 것입니다. 이 외에 다른 표기법도 있습니다만 많이 사용된다고 생각하는 것들만 나타냈습니다.

**Generalization (일반화)**

Generalization은 슈퍼(부모)클래스와 서브(자식)클래스간의 Inheritance(상속) 관계를 나타냅니다. 여기서 Generalization이란 서브 클래스가 주체가 되어 서브 클래스를 슈퍼 클래스로 Generalize 하는 것을 말하고 반대의 개념은 슈퍼 클래스를 서브 클래스로 Specialize(구체화) 하는 것입니다. 상속은 슈퍼 클래스의 필드 및 메서드를 사용하며 구체화 하여 필드 및 메서드를 추가 하거나 필요에 따라 메서드를 overriding(오버라이딩) 하여 재정의 합니다. 또는 슈퍼 클래스가 추상 클래스인 경우에는 인터페이스의 메서드 구현과 같이 추상 메서드를 반드시 오버라이딩 하여 구현하여야 합니다.

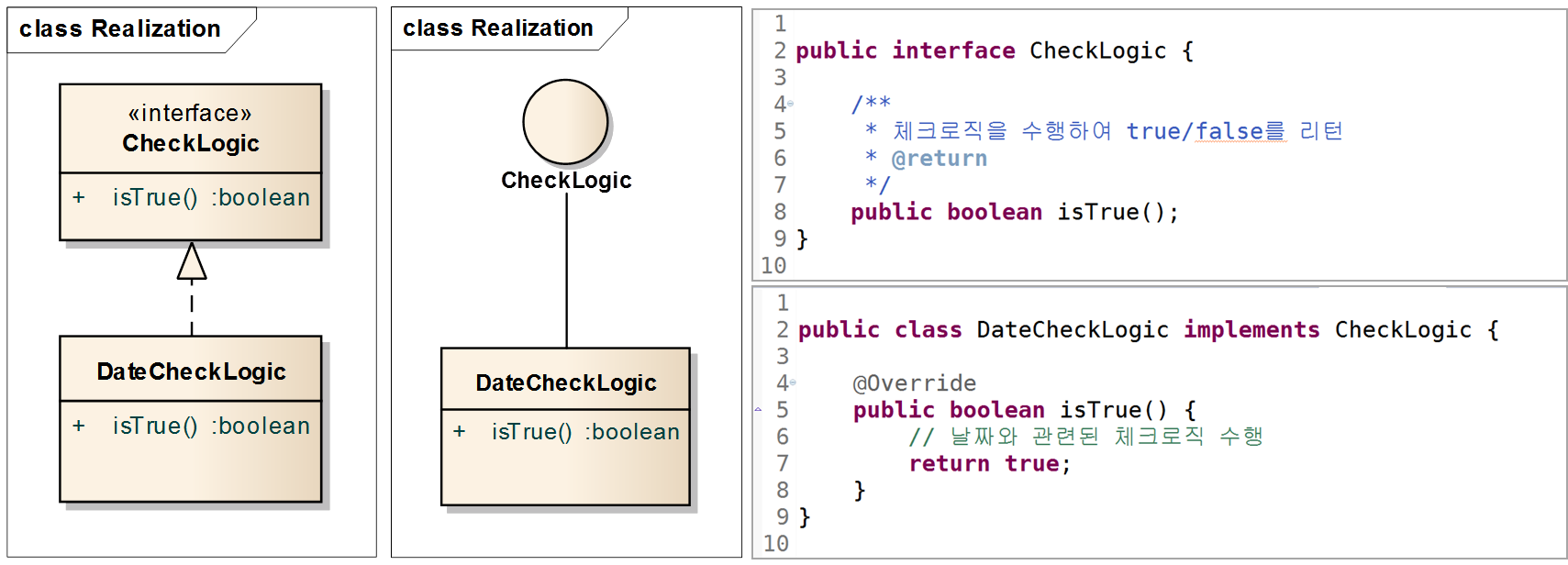


[그림 7] Generalization

위와 같이 표기법은 클래스 사이에 실선을 연결하고 슈퍼 클래스 쪽에 비어 있는 삼각형으로 나타내고 자바에서는 extends 키워드를 사용하여 상속을 구현합니다.

**Realization (실체화)**

Realization은 interface의 spec(명세, 정의)만 있는 메서드를 오버라이딩 하여 실제 기능으로 구현 하는 것을 말합니다.

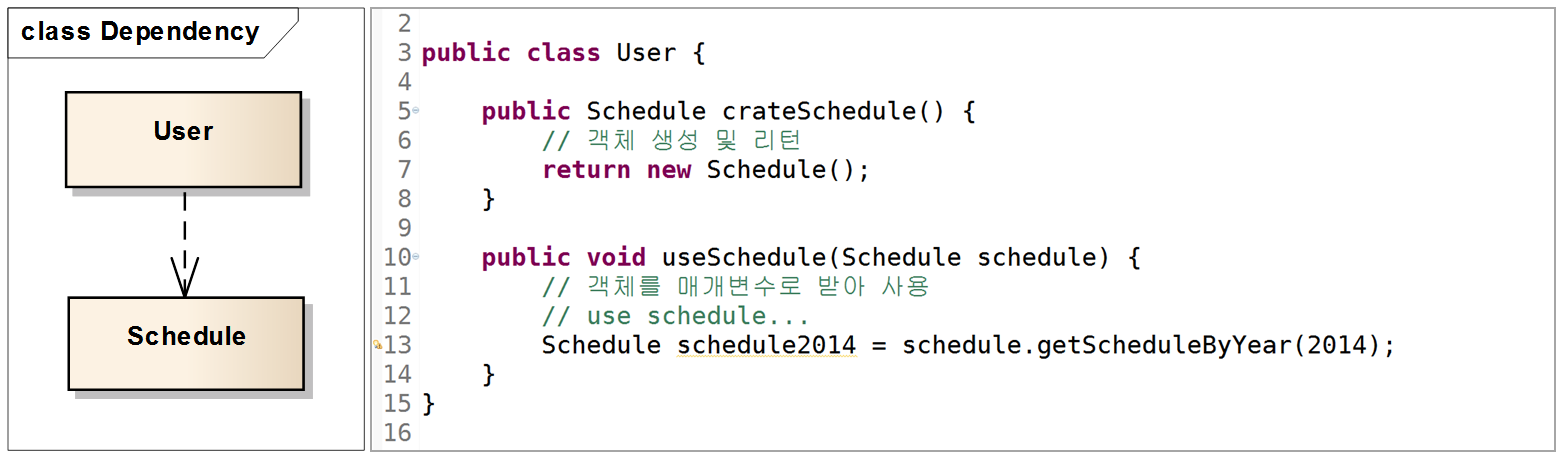


[그림 8] Realization

Realization을 나타내는 표기법은 2가지가 있습니다. 첫 번째는 인터페이스를 클래스처럼 표기하고 스테레오 타입 «interface»를 추가합니다. 그리고 인터페이스와 클래스 사이의 Realize 관계는 점선과 인터페이스 쪽의 비어있는 삼각형으로 연결합니다. 두 번째는 인터페이스를 원으로 표기하고 인터페이스의 이름을 명시합니다. 그리고 인터페이스와 클래스 사이의 관계는 실선으로 연결합니다.  
자바에서는 위와 같이 implements 키워드를 사용하여 인터페이스를 구현합니다.

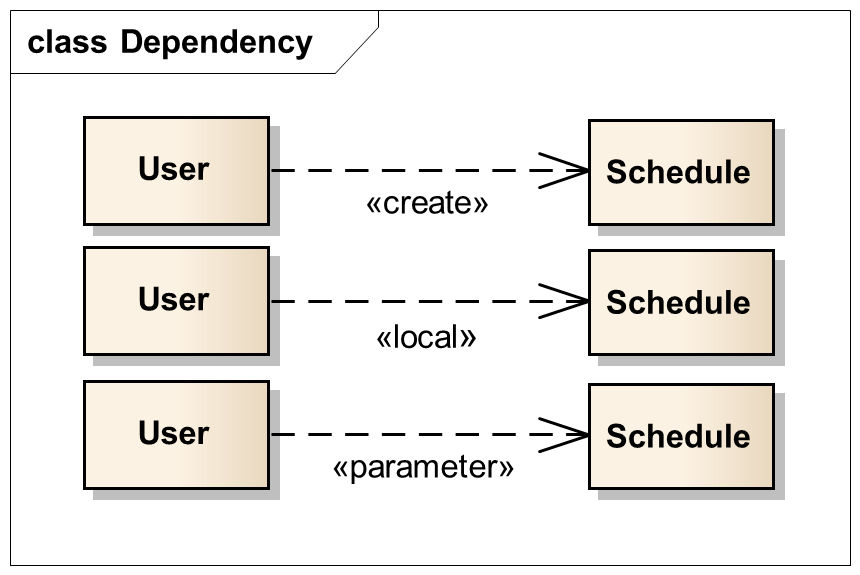
**Dependency (의존)**

Dependency는 클래스 다이어그램에서 일반적으로 제일 많이 사용되는 관계로서, 어떤 클래스가 다른 클래스를 참조하는 것을 말합니다.



[그림 9] Dependency

위의 그림은 자바에서 참조하는 형태에 대해 코드를 보여주고 있습니다. 참조의 형태는 메서드 내에서 대상 클래스의 객체 생성, 객체 사용, 메서드 호출, 객체 리턴, 매개변수로 해당 객체를 받는 것 등을 말하며 해당 객체의 참조를 계속 유지하지는 않습니다.

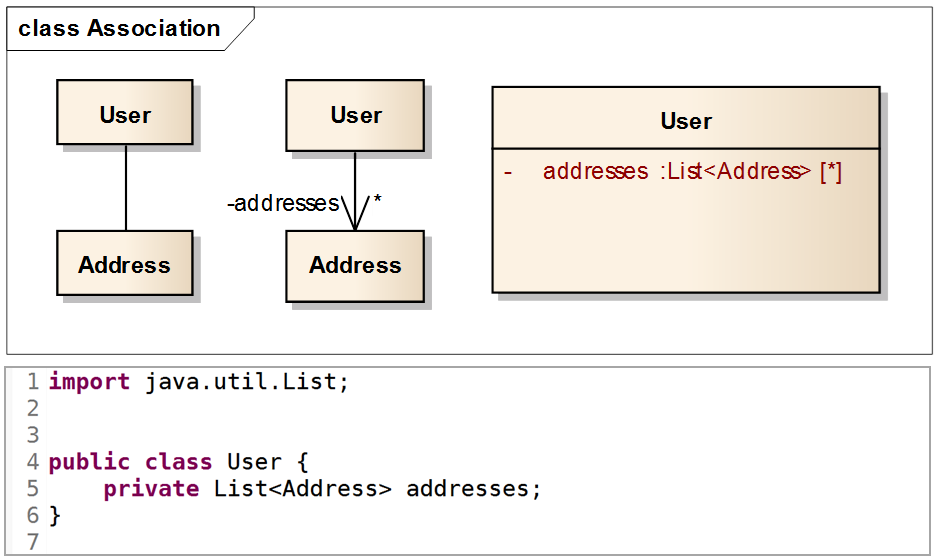


[그림 10] Dependency Stereo Type

추가로 위와 같이 스테레오 타입으로 어떠한 목적의 Dependency인지 의미를 명확히 명시 할 수도 있는데 Dependency의 목적 또는 형태가 중요할 경우 사용할 수도 있을 것 같습니다.

**Association (연관), Directed Association(방향성 있는 연관)**

클래스 다이어그램에서의 Association은 보통 다른 객체의 참조를 가지는 필드를 의미합니다. 아래 클래스 다이어그램은 두 가지 형태의 Association을 나타내고 있습니다.



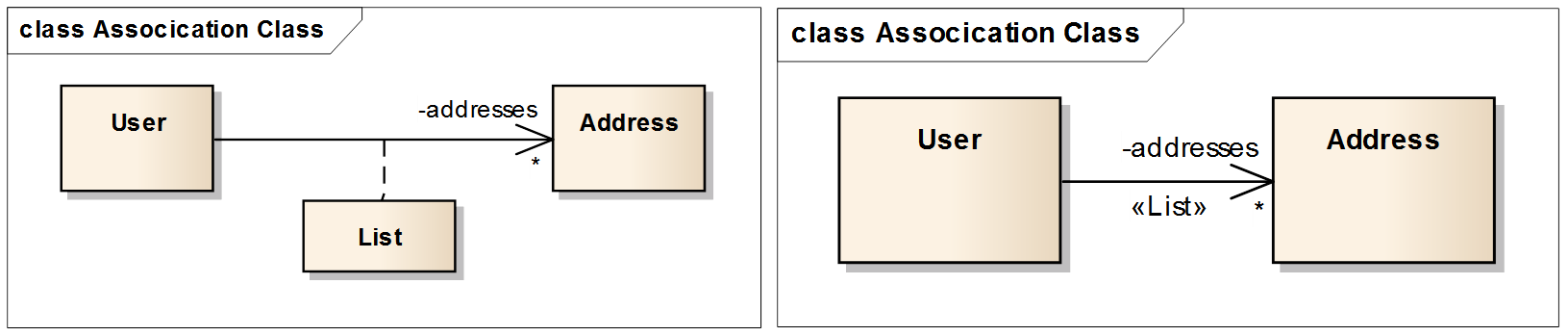
[그림 11] Assocication

첫 번째 다이어그램은 일반적인 Association으로 단지 실선 하나로 클래스를 연결하여 표기하고 두 번째 다이어그램은 Directed Association으로 클래스를 실선으로 연결 후 실선 끝에 화살표를 추가합니다.Association과 Directed Association의 차이는 화살표가 의미하는 navigability(방향성)인데 이것에 따라 참조 하는 쪽과 참조 당하는 쪽을 구분합니다. 두 번째 다이어그램은 User에서 Address 쪽으로 화살표가 있으므로 User가 Address를 참조하는 것을 의미합니다. Navigability가 없는 Association은 명시되지 않은 것으로 User가 Address를 참조할 수도, Address가 User를 참조할 수도, 또는 둘 다일 수도 있는 것을 의미합니다. 화살표 옆에 있는 –addresses는 roleName(역할명)을 나타내고 Address가 User 클래스에서 참조될 때 어떤 역할을 가지고 있는지를 의미합니다.

\*는 Multiplicity(개수)을 나타내는데 대상 클래스의 가질 수 있는 인스턴스 개수 범위를 의미합니다. 0…1 과 같이 점으로 구분하여 앞에 값은 최소값, 뒤에 값은 최대값을 의미하는데 \*은 0…\*과 같은 의미로 객체가 없을 수도 있고 또는 수가 정해지지 않은 여러 개일 수도 있다는 것을 의미합니다. 이전에는 Multiplicity가 아닌 Cardinality로 불렸는데 “특정 집합 또는 다른 그룹에 있는 요소의 수”라는 Cardinality의 사전적 의미가 실제 인스턴스의 수가 아닌 가질 수 있는 범위를 지정하는 클래스 다이어그램에서의 의미에 적합하지 않다는 이유로 바뀌었습니다.

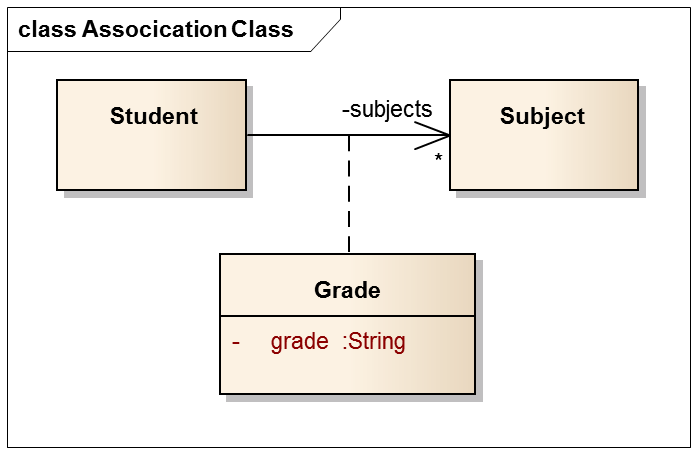
세 번째의 다이어그램은 두 번째의 다이어그램과 비슷한 의미를 가지고 있지만 다른 형태인 속성 표기법으로 나타낸 것입니다. 여기서 보는 바와 같이 roleName은 보통 클래스의 필드명이 됩니다. 속성 표기법이 두 번째 클래스 다이어그램과 조금 다른 점은 여러 개의 객체에 대한 Container(컨테이너)가 List라는 것까지 알려주고 있습니다.

그럼 두 번째와 세 번째의 다이어그램이 완전히 동일한 의미를 가지게 하려면 어떻게 해야 할까요?



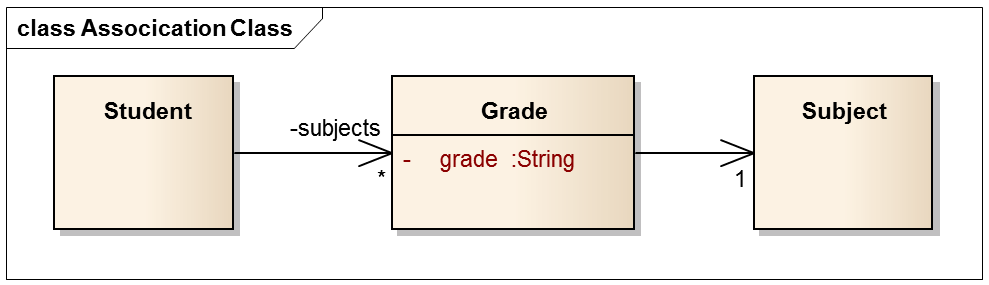
[그림 12] Association Class

위와 같이 Association Class(연관클래스)를 사용하여 어떤 종류의 컨테이너 클래스가 사용되는지 까지 나타낼 수 있습니다. 하지만 사실 특이한 클래스도 아니고 자바에서 기본으로 제공되는 List를 저렇게 표기하는건 조금 귀찮은 일일 수 있을 것 같습니다. 그래서 스테레오 타입으로 표기할 수도 있습니다. 이로써 [그림11] 세 번째 다이어그램의 속성 표기법으로 표현된 것과 모든 의미가 같아졌습니다.보통은 클래스의 속성이 기본 제공 클래스가 아니거나 중요 또는 강조하고 싶을 때 Association 관계로 나타냅니다.Association Class는 조금 다른 의미로도 사용될 수 있습니다. 예를 들어 학생과 수강과목 클래스가 Association 관계를 가지고 있는데 단순 Association 관계가 아니라 각 관계마다 해당 과목의 학점이라는 속성이 필요하다면 어떻게 나타낼 수 있을까요? 이럴 때도 Association Class를 사용하여 나타낼 수 있습니다.



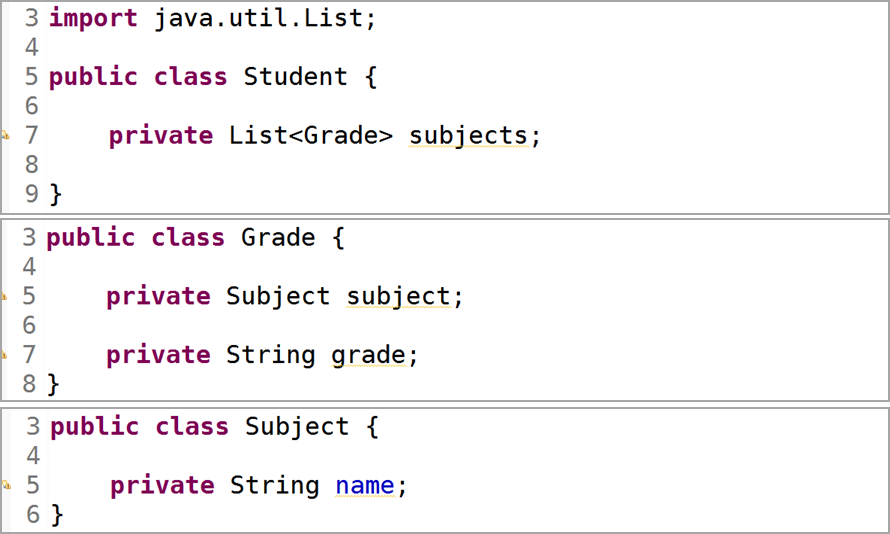
[그림 13] Association Class

물론 grade라는 값을 Subject 클래스 자체의 속성으로 할 수도 있지만 Subject의 속성이라기 보다는 Student와 Subject 사이의 관계에 대한 속성이라는 관점에서 위의 다이어그램처럼 Association Class로 나타낼 수 있습니다.



[그림 14] Association Class Vs Association

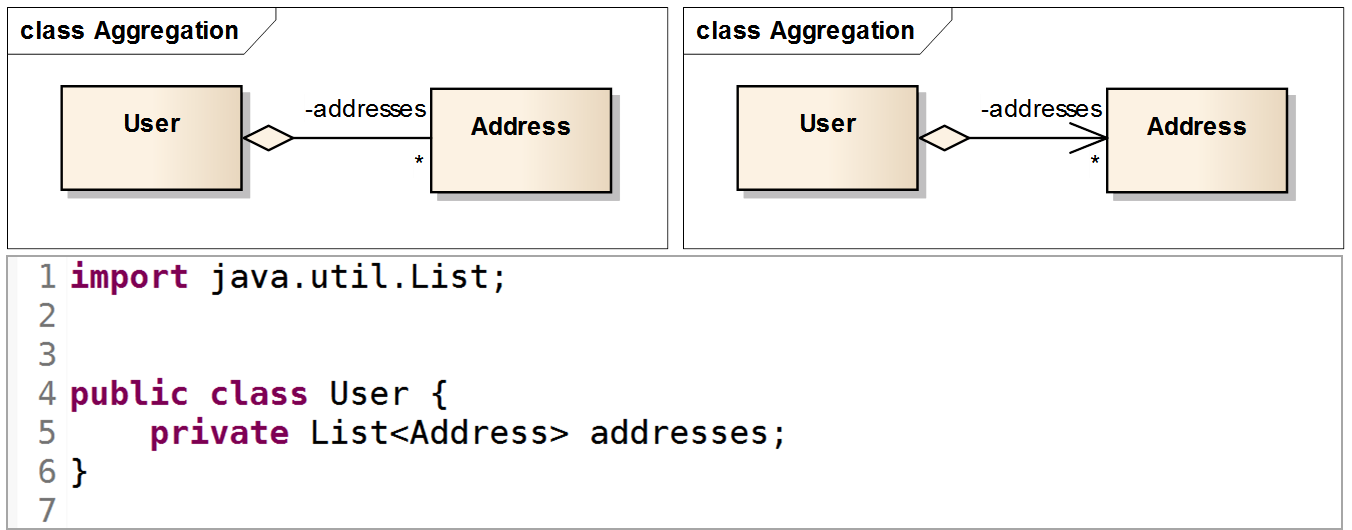
또한 [그림13]의 Association Class를 풀어서 위처럼 Association 관계만으로도 나타낼 수 있습니다. [그림13]은 Student와 Subject의 관계를 나타내고 싶은데 그 관계에 대한 속성값으로 Grade가 있는 것이고 [그림14]는 단순 3개 클래스를 Association 관계로 나타낸 것처럼 의미는 조금 다를 수 있으나 구현되는 코드는 같을 것입니다.



[그림 15] Association Class Code

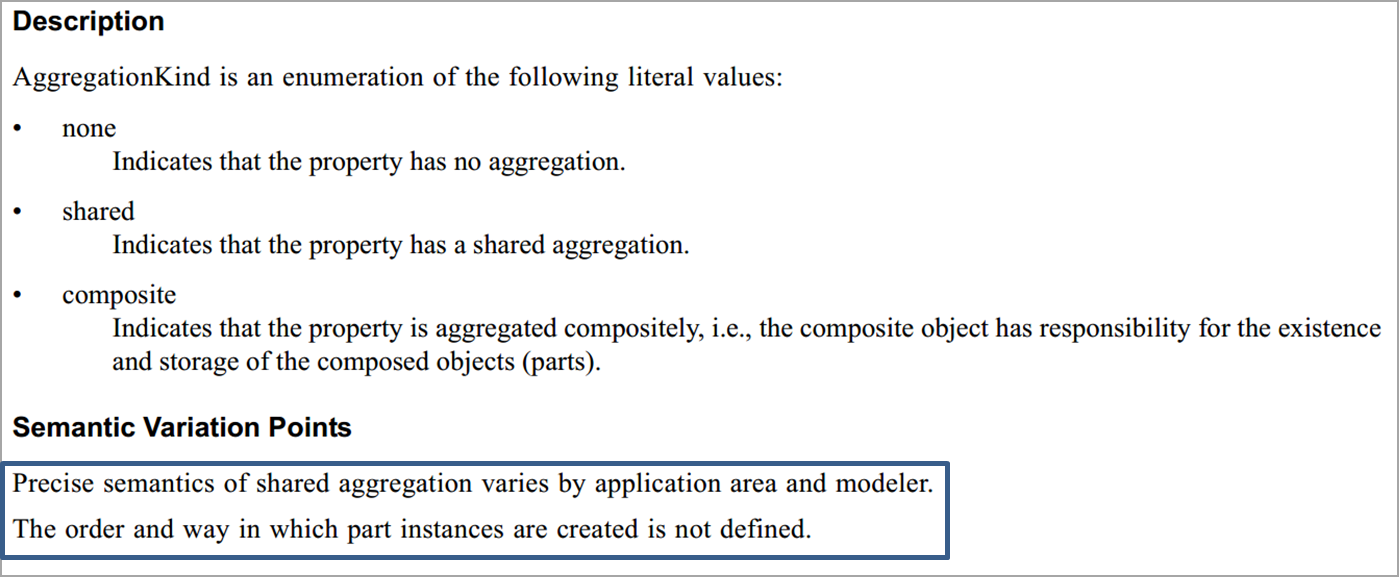
**Aggregation (Shared Aggregation, 집합)**

Aggregation은 Shared Aggregation이라고도 하며 Composition(Composite Aggregation)과 함께 Association 관계를 조금 더 특수하게 나타낸 것으로 whole(전체)와 part(부분)의 관계를 나타냅니다. Association은 집합이라는 의미를 내포하고 있지 않지만 Aggregation은 집합이라는 의미를 가지고 있습니다.



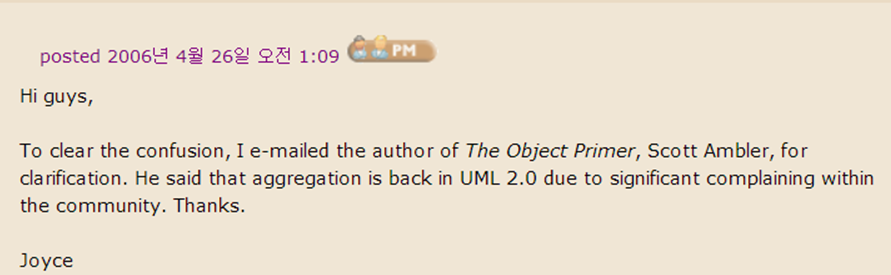
[그림 16] Aggregation

표기법은 위와 같이 whole과 part를 실선으로 연결 후 whole쪽에 비어있는 다이아몬드를 표기합니다. Part쪽에는 화살표를 명시하여도 되고 명시하지 않아도 됩니다. Aggregation의 다이아몬드가 이미 navigability의 방향을 표현하고 있기 때문입니다.그런데 코드를 보시면 위에서 보았던 Association의 코드와 똑같습니다. Association과 Aggregation은 집합이라는 개념적인 차이는 있지만 코드에서는 이 차이를 구분하기 힘듭니다.



[그림 17] UML Aggreagtion  
출처: OMG Unified Modeling Language Superstructure 2.4.1

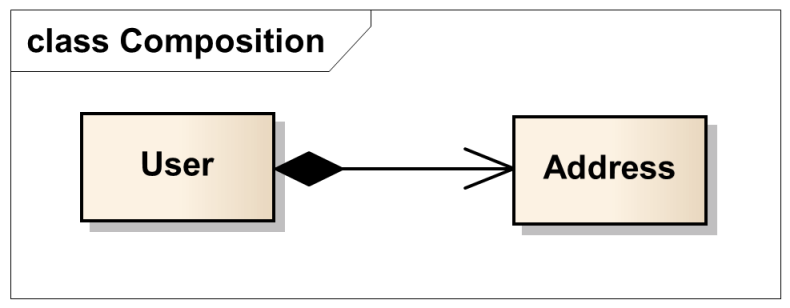
위와 같이 UML은 집합이라는 개념 외에 명확한 Aggregation의 정의를 제공하지 않습니다. 그래서 여러 프로그래머나 분석가, 설계사가 Aggregation 관계에 대해 자기 나름의 정의를 내렸기 때문에 혼란이 생겼고 Aggregation은 사용하지 않는 것이 좋다고 합니다. 저도 이에 동의 하는 바이지만….. 아직도 논란이 되고 있는 부분이고 정확한 답을 내기는 어려울 것 같습니다.그리고 도서 “The Object Primer: Agile Model-Driven Development with UML 2.0(저자 Scott W. Ambler)”과 “UML 실전에서는 이것만 쓴다(저자 Robert C. Martin)”에 UML 2.0에서는 위의 문제 때문에 Aggregation이 제외 되었다는 말이 있습니다. 그래서 근거를 찾기 위해 이곳 저곳 찾아본 결과 처음엔 제외 되었다가 커뮤니티의 강력한 항의로 다시 들어왔다고 합니다. 실제로도 UML 1.x 버전에 비해 2.x 버전에는 Aggregation에 대한 내용이 많이 사라지긴 했지만 몇 군데 남아 있는 걸 확인했습니다.



[그림 18] Aggregation Dropped  
출처: www.coderanch.com/t/154620/java-Architect-SCEA/certification/Association-Aggregation

**Composition (Composite Aggregation, 합성)**

Composition(또는 Composite Aggregation)도 Aggregation과 비슷하게 whole(전체)와 part(부분)의 집합 관계를 나타내지만 개념적으로 Aggregation보다 더 강한 집합을 의미합니다.



[그림 19] Composition

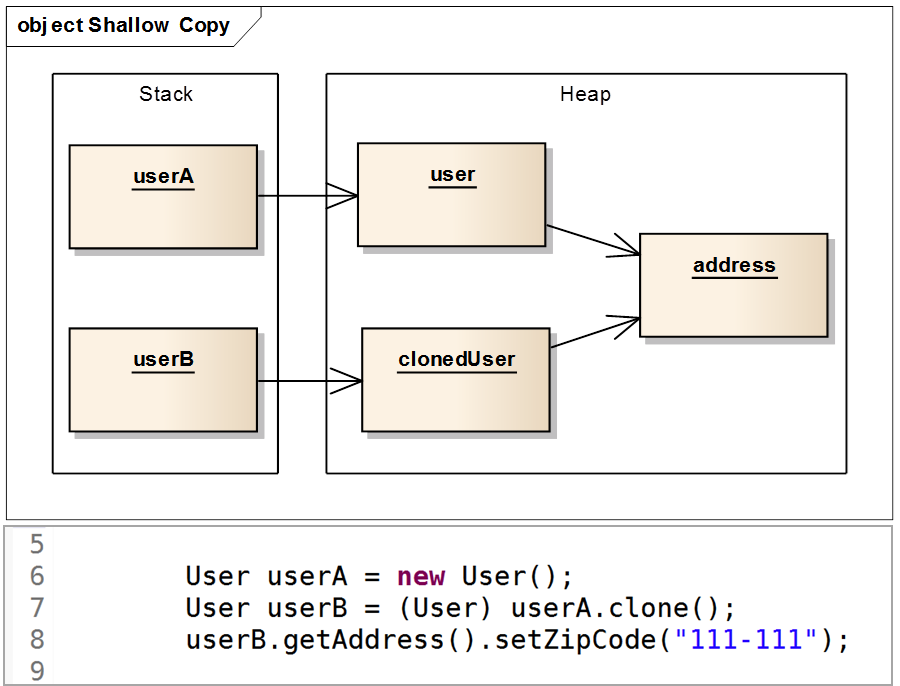
Composition의 표기법 또한 위와 같이 Aggregation과 비슷하지만 다이아몬드의 내부가 채워져 있다는 점만 다릅니다. 그럼 Composition의 개념과 코드에서는 Aggregation과 어떤 차이가 있는지 보겠습니다.Composition은 Aggregation보다 강한 집합이라고 했습니다. 여기서 강한 집합이란 part가 whole에 대해 독립적이서 whole이 part를 가져다 쓰는 것과 비슷합니다. 반면 Aggregation은 part가 whole에 종속적이어서 part가 whole의 것입니다. 이러한 의미 때문에 Aggregation과는 다르게 명확하게 나타나는 점이 있습니다.

* 첫 번째, part를 가지는 whole 인스턴스가 part 인스턴스의 전체 수명을 책임진다.
* 두 번째, part에 해당하는 인스턴스는 공유 될 수 없다.

첫 번째의 whole 인스턴스가 part 인스턴스의 전체 수명을 책임진다는 의미는 다음과 같습니다.

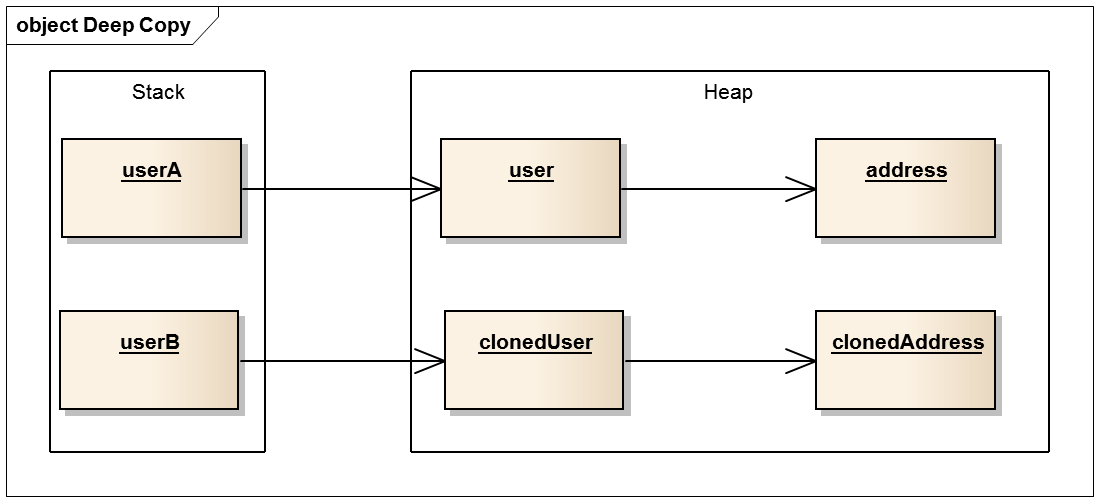
* whole 인스턴스가 part 인스턴스를 생성
* whole 인스턴스가 소멸되면 part 인스턴스도 함께 소멸
* whole 인스턴스가 복사되면 part 인스턴스도 함께 복사

두 번째의 part에 해당하는 인스턴스는 공유 될 수 없다는 의미를 먼저 보겠습니다.



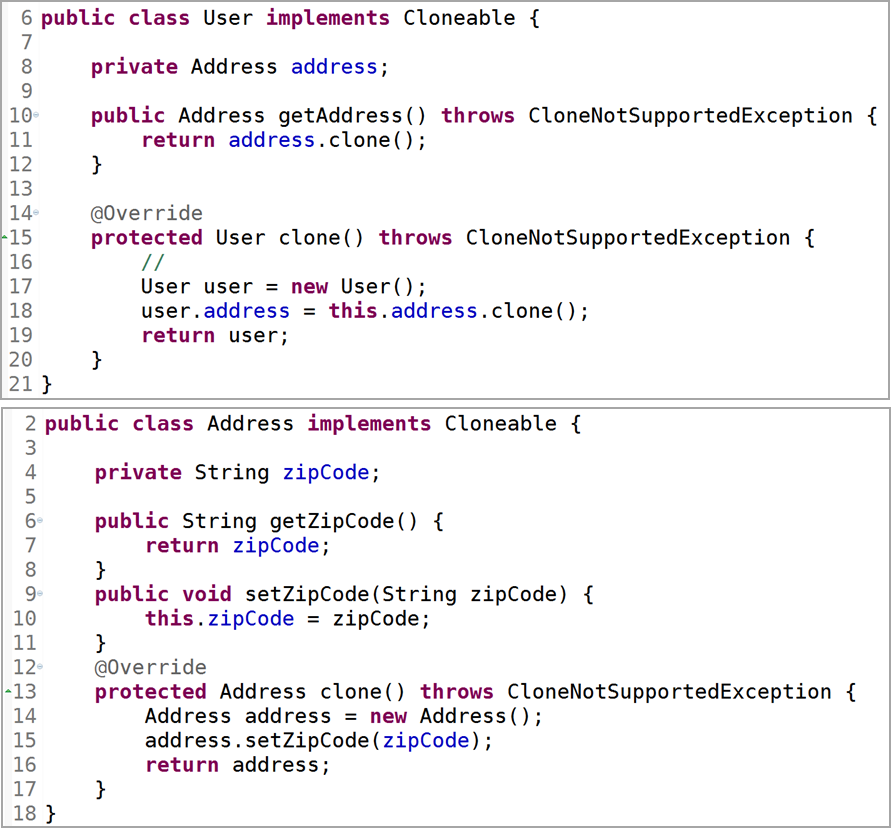
[그림 20] Shallow Copy

위의 예제는 클래스 다이어그램이 아닌 객체 다이어그램으로 [그림19]를 객체로 표현한 것입니다. 참조변수 userA가 참조하고 있는 user 객체를 clone하여 clonedUser 객체를 만들고 참조변수 userB에서 참조하고 있습니다. user객체는 제대로 복사가 됐지만 user객체 안에서 참조하고 있는 address는 clonedUser 객체도 똑같이 참조하고 있습니다. 이것을 shallow copy(얕은 복사)라고 하며 이 경우에 part에 해당하는 address 객체가 공유 된 것입니다. 또한 첫 번째 조건의 의미 중 whole 인스턴스가 복사되면 part인스턴스도 복사되어야 한다는 조건도 충족시키지 못했습니다.



[그림 21] Deep Copy Object Diagram

위의 다이어그램에서는 user 객체가 복사되어 clonedUser 객체가 생성될 때 user 객체가 참조하여 가지고 있는 address 객체 또한 같이 복사 되어 clonedUser 객체는 새로운 clonedAddress 객체를 참조하여 가지고 있습니다.



[그림 22] Deep Copy Code

Aggregation 관계와 Composition 관계를 UML 툴에서 그린 후 Code Generation을 하면 똑같은 코드가 생성됩니다. 하지만 Composition에서는 개발자가 구현해야 할 부분이 몇 가지 있습니다. 위에서 본 part에 해당하는 인스턴스가 공유되지 않게 하기 위한 Deep Copy 구현과 part를 가지는 whole 인스턴스가 part 인스턴스의 수명 전체를 책임져야 한다는 것에 따라 whole 클래스의 생성자 또는 기타 메서드 내에서 part 인스턴스를 생성해야 하고 외부에서 part 객체를 생성하지 못하도록 whole 클래스에는 part 인스턴스에 대한 setter가 있으면 안됩니다. 다른 언어에서는 생명주기와 관련된 다른 추가적인 것도 있겠지만 자바에서의 객체 소멸은 Garbage Collector가 수행하므로 part 인스턴스의 소멸은 신경쓰지 않아도 됩니다.

**5. 마치며**

이로써 클래스 다이어그램의 주요 요소들과 관계들이 소스코드와 어떻게 연관되는지를 살펴 봤습니다. 우선 많은 용어들에 대해 영문발음 그대로 한글로 표기하는 것들은 처음에만 영문 표기 후 한글로 표기하였고 영문 용어가 해석에 따라 달라질 수 있는 것들(예: Association->연관)은 최대한 영문 그대로 표기하였습니다.  
또한 UML에서 attribute(속성), operation(기능)이라고 부르는 것들이 자바에서는 filed(필드), method(메서드)라 불리우기 때문에 의미는 비슷하지만 사용 관점에 따라 분리하여 사용한 것들도 있으며 초급 개발자로서 UML에 관련된 내용을 쓰려니 혹시 내가 잘못 알고 있는 것은 아닐까 조심스러워서 UML 스펙 문서를 많이 찾아가며 최대한 표준에 맞게 쓰려고 하였습니다.

저는 처음에 클래스 다이어그램을 공부하면서 개념적으로 들었을 때는 이해가 갈듯 말듯 하다가 코드를 보면서 아 이런 거구나 하며 그나마 조금 더 이해가 갔던 기억이 납니다. 설계를 하시는 분들이 아닌 저와 같이 이제 막 개발자의 길로 들어서신 초급 개발자 분들이 설계 다이어그램을 보며 어떤 식으로 이해를 해야 하고 어떻게 코드를 만들어야 할지 조금이라도 도움이 되었으면 합니다.

그리고… 개인적인 내용입니다만 아직 많이 모자라지만 1년동안 많은 도움을 주셨던, 특히나 UML과 설계와 관련된 많은 지식을 전해주신 제 멘토 김현오 선임께 감사하단 말씀을 제대로 못 드린 것 같아 여기서 감사의 말씀을 드리고 싶습니다..^^  
감사합니다.

**예제 자료**