5장. 개념 II

(2023)

시작하기 전에

이번 장은 개념에 대한 그래픽 표현에 대해서 설명합니다.
연관에 대해 좀 더 깊은 수준으로 설명합니다.

이번 장을 통해 데이터타입, 클래스, 연관을 클래스 다이어그램에 표현할 수 있어야 합니다.

1절. 그래픽 표현

1.1 데이터타입과 클래스와 연관은 UML 클래스 다이어그램에 작성됩니다.

1.2 데이터타입은 데이터타입이 사용되는 곳에 데이터타입 이름을 직접 작성하면 됩니다. ':' 뒤에 데이터타입 이름을 작성합니다.
속성 이름이 name이고 데이터타입이 String이면 'name: String'와 같이 작성합니다.

1.3 열거형과 데이터타입은 사용자가 정의해야 합니다.
기본형은 언어에서 정의해서 제공하는 타입이기 때문에 직접 정의할 필요가 없습니다.

1.4 클래스는 사각형으로 작성됩니다.
클래스는 이름 부분과 속성 부분과 오퍼레이션 부분을 갖습니다. 각각의 부분은 구분선으로 나누어집니다.

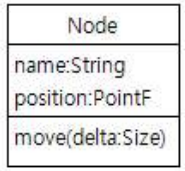


그림 5.1

그림 5.1에서
클래스의 이름은 'Node'입니다. 이름이 작성된 부분이 이름 부분입니다.

name과 position은 속성입니다. 각각의 타입은 String과 PoinF입니다. 속성들이 작성되는 부분을 속성부분이라고 합니다.
move는 오퍼레이션입니다. 매개변수로 Size타입의 delta가 있습니다. 오퍼레이션들이 작성되는 부분을 오퍼레이션 부분이라고 합니다.

각각의 영역은 목적에 따라 감출 수 있습니다. 속성이나 오퍼레이션은 목적에 따라 세부사항을 감출 수도 있습니다.

속성에 대한 그래픽 표현은 9장 속성에서, 오퍼레이션에 대한 그래픽 표현은 11장 행위 I에서 좀 더 자세히 다룹니다.

1.5 열거형은 enumeration 키워드를 갖는 사각형으로 작성됩니다.
키워드는 '<<'와 '>>' 사이에 작성합니다.

그림 5.2

열거형은 열거형값으로 사용할 수 있는 값들을 정의합니다. 열거형에 정의된 개별 값들을 열거형리터럴이라고 합니다.
열거형리터럴들은 클래스의 속성 부분과 같이 이름 부분 아래에 작성됩니다.

그림 5.2에서
이것은 <<enumeration>> 키워드를 가지므로 열거형입니다.
이름은 CustomerTypeKind이고, 이 타입을 갖는 객체의 특성은 Gold, Silver, Bronze 중 하나를 사용할 수 있습니다. 다른 것은 사용할 수 없습니다.

그림 5.3은 Customer라는 클래스의 customerType 속성의 타입으로 CustomerTypeKind 열거형이 사용되는 예입니다.

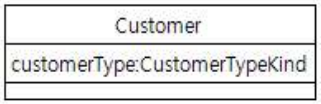


그림 5.3

그림 5.4는 Customer 클래스로부터 생성된 c1이라는 개체의 customerType인 CustomerTypeKind의 Bronze라는 것을 보여줍니다. 열거형 리터럴 값이 사용될 때는 어떤 열거형의 리터럴인지를 작성해 주어야 합니다.

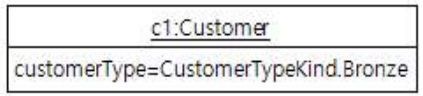


그림 5.4

1.6 UML은 스테레오타입과 키워드에 같은 표현을 사용합니다.

키워드는 다른 개념인데 표현이 비슷해서 구분할 필요가 있을 때 사용하는 것이고, 스테레오타입은 같은 개념인데 의미를 조금 확장할 필요가 있을 때 사용하는 것입니다.
UML에서는 스테레오타입에 키워드와 같은 표현을 사용합니다.

1.7 데이터타입은 dataType 키워드를 갖는 사각형으로 작성됩니다.

데이터타입은 데이터값에 대한 개념으로 인식의 효율성을 위해 의미적으로 가까운 여러 값들의 타입을 묶어 놓은 것입니다.
그래픽 객체의 위치를 표현할 때는 x좌표의 값과 y좌표의 값을 사용합니다. x좌표의 값과 y좌표의 값은 각각을 별개로 사용해도 문제가 되지 않습니다.
x좌표와 y좌표는 좌표의 한 점을 나타낸다는 점에서 의미적인 거리가

가깝습니다.
x좌표와 y좌표를 Point로 묶어서 관리할 수 있다면, 각각의 x좌표와 y좌표를 인식하기 전에 먼저 Point를 인식하면 되기 때문에 인식의 효율성이 높아집니다.

그림 5.5에서
이것은 <<dataType>> 키워드를 가지므로 데이터타입입니다. int 타입의 x와 int 타입의 y를 하나의 PointF로 묶은 것입니다.

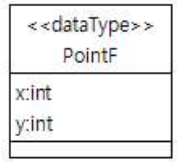


그림 5.5

그림 5.1의 position은 PointF 데이터타입을 갖는 속성임을 보여줍니다. 그림 5.1은 그림 5.6과 같이 표현해도 같은 것을 나타냅니다.

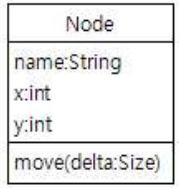


그림 5.6

그림 5.7은 Node 클래스로 생성된 객체의 이름이 n1이고, position이 x좌표는 10, y좌표는 10임을 보여줍니다.

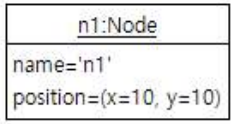


그림 5.7

그림 5.8은 그림 5.6과 같이 x좌표와 y좌표가 분리된 클래스로부터 생성되는 객체를 보여줍니다.

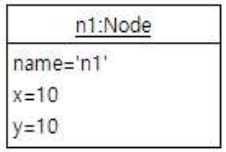


그림 5.8

1.8 연관은 링크와 같이 두 클래스 사이에 역할쌍을 가진 실선으로 작성합니다.

3절. 연관(Association)

3.1 연관은 링크에 대한 개념으로 클래스와 클래스 사이의 연결입니다.

3.2 연관은 역할쌍에 의해 결정됩니다.

링크의 끝에는 한 쌍을 이루는 역할 태그가 붙어 있습니다.
링크들은 같은 역할쌍으로 모을 수 있습니다.

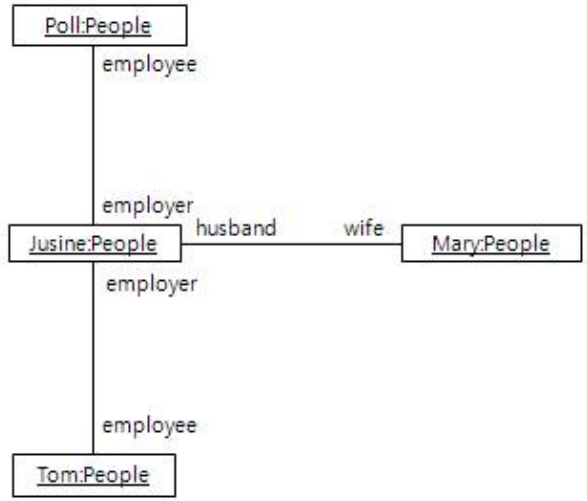


그림 5.9

그림 5.9와 같은 객체 다이어그램에서 같은 역할쌍별로 링크를 모아보면, 그림 5.10과 같이 employer-employee, husband-wife의 역할쌍별로 링크들을 모을 수 있습니다.

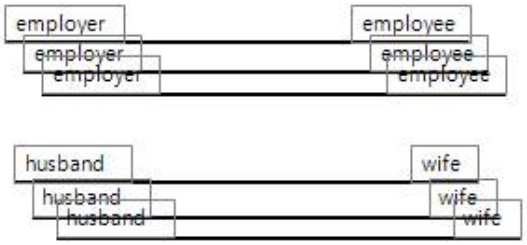


그림 5.10

역할쌍 이름이 붙여진 링크집합이 만들어졌습니다. 즉, 역할쌍으로 개념이 만들어졌습니다.

개념이 있으면 그 개념을 통해 실체를 생성할 수 있다고 했습니다. 역할쌍 개념이 있으니 역할쌍 실체를 만들 수 있습니다.

역할쌍 개념은 그림 5.11과 같이 표현됩니다.

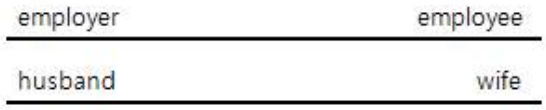


그림 5.11

역할쌍 employer-employee 개념으로 실체를 생성하면 양쪽 끝에 employer, employee라는 태그가 붙은 링크가 생성됩니다. 이것을 세 번 반복하면 세 개의 링크가 생성됩니다.

husband-wife에 대해서도 마찬가지입니다.

링크의 개념은 연관입니다. 그런데 지금까지 살펴본 것처럼 역할쌍 개념으로 링크를 생성했습니다. 단순화하면 역할쌍이 연관라고 할 수 있습니다.

연관에는 역할쌍 외에도 방향성과 다중성 같은 좀 더 많은 것들이 포함되기 때문에 '역할쌍이 연관이다'라고 하기는 어렵습니다. 하지만 역할쌍에 의해 연관이 결정된다고는 할 수 있습니다.

3.3 역할은 연관지점에 작성됩니다.
역할 이름은 클래스 이름과 같은 경우 생략할 수 있습니다.
연관의 역할은 객체의 상대적인 역할을 나타냅니다.

클래스와 연결되는 연관의 끝점을 연관지점(association end)이라고 합니다. 역할은 연관지점에 작성되는데 보통 연관 선 위에 작성됩니다.

그림 5.12에서 Room 쪽의 역할이 생략되어 있으니 클래스 이름과 같음을 알 수 있습니다.
Room에 대한 Member의 상대적인 역할은 occupant입니다. Member에 대한 Room의 상대적인 역할은 room입니다.

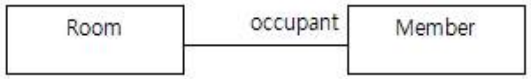


그림 5.12

그림 5.13은 그림 5.12의 클래스와 연관으로 생성된 객체들과 링크를 나타냅니다.

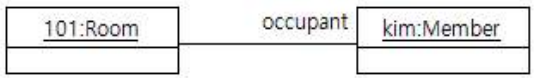


그림 5.13

"Room 101의 occupant가 누구야?"라고 했을 때 연결된 링크를 통해 "Member kim이야"라고 답할 수 있습니다.
Room 101의 room 역할의 상대적 역할인 occupant로 연결된

Member인 kim을 찾은 것입니다.

3.4 하나의 역할쌍에 연결되는 객체들의 수를 제약할 수 있습니다.

'한 employer는 여러 employee에 연결될 수 있고, 한 husband는 한 wife에만 연결될 수 있다.'와 같이 하나의 역할쌍에 연결될 수 있는 링크 수를 제약할 수 있습니다.

3.5 한 역할쌍에 연결될 수 있는 링크의 수에 대한 제약을 다중성(multiplicity)이라 합니다.

다중성 값은 범위로 작성되는데 그 범위는 최소와 최대로 다음과 같이 작성합니다.

<최솟값>'..'<최댓값>

최솟값으로는 0 또는 1을 작성할 수 있습니다. 최솟값이 0이라는 것은 링크가 없는 것도 허용한다는 의미입니다. 최솟값이 1이라는 것은 반드시 객체가 생성될 때 해당 역할쌍에 해당하는 링크가 필수적으로 하나는 있어야 함을 의미합니다.

최댓값으로는 1또는 다를 작성할 수 있습니다. 최댓값이 1이라는 것은 링크를 하나까지만 허용하겠다는 것입니다. 다라는 것은 여러 개의 링크가 있을 수 있다는 것을 의미합니다. 2이상이면 여러 개입니다. 최댓값이 다일 때는 '*'를 사용해서 표현합니다.

이 기준으로 다중성의 최솟값과 최댓값을 조합하면 '0..1, 1..1, 0..*, 1..*'와 같이 4가지 경우가 가능합니다.

좀 더 단순화하면 1..1은 1로, 0..*는 *로 표현할 수 있기 때문에 다중성으로 가능한 경우는 '0..1, 1, *, 1..*'이 됩니다.

1은 기본 값입니다. 연관의 다중성으로 아무것도 작성되지 않으면 1이

라는 것입니다.

UML에서는 이외에도 다양하게 다중성을 작성할 수 있도록 하고 있지만, 우리는 이러한 것들을 불필요한 것으로 간주하고 위에서 제시한 4가지 방법으로만 표현하도록 합니다.

3.6 다중성의 최댓값은 링크된 객체들을 기준으로 파악할 수 있습니다.

최댓값이란 1이상을 의미하기 때문에 링크된 객체들만을 살피면 됩니다. 다중성이란 하나의 객체가 같은 역할쌍으로 몇 개의 객체들과 연결될 수 있는지에 대한 제약이므로 하나의 객체를 기준으로 해야 합니다.

그림 5.14에서 Justine을 기준으로 해서 employer-employee 역할쌍과 husband-wife 역할쌍을 살펴보면, Justine은 employer로서 두 employee와 연결되고, husband로서 한 wife와 연결됩니다.

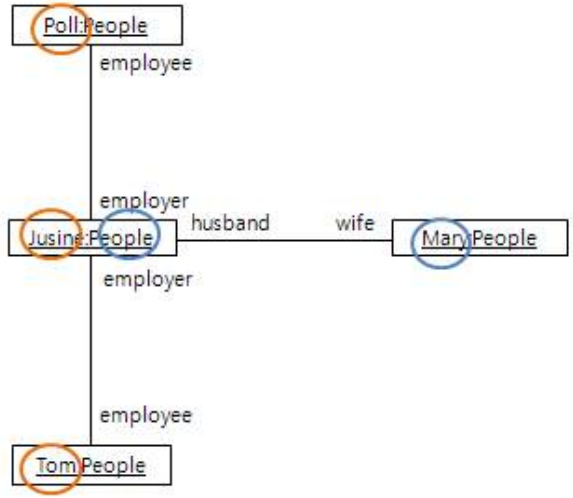


그림 5.14

employer에 대한 employee로의 최댓값은 2이므로 다로 표현할 수 있습니다. husband에 대한 wife로의 최댓값은 1입니다.

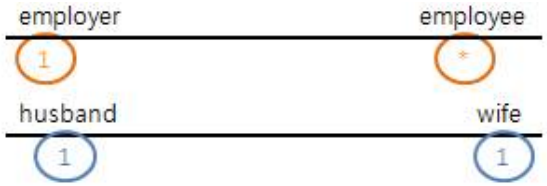


그림 5.15

3.7 다중성의 최솟값은 링크되지 않은 객체들을 기준으로 파악할 수 있습니다.

최솟값은 선택적인지 필수적인지가 중요합니다. 선택적이라는 것은 링크가 없다는 것입니다. 1인 것은 객체 다이어그램에 링크로 나타났으니 링크로 나타나지 않는 것 0인 것만 파악하면 됩니다.

3.8 다중성의 최솟값은 연관을 작성한 후에 파악하는 것이 쉽습니다.

employer에 해당하는 연관지점을 선택하고, 여기에 연결되는 employee가 없을 수도 있는지를 파악합니다. Mary의 employee가 있습니까? 없습니다. employee의 최솟값은 0이 됩니다. 나머지 부분도 동일한 방법으로 최솟값을 파악합니다.

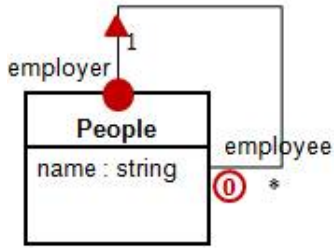


그림 5.16

3.9 다중성의 최댓값이 2이상이라는 것은 컬렉션임을 의미합니다.

컬렉션에는 순서를 고려할 것인지의 여부와 중복을 허용할 것인지의 여부에 따라 Set, OrderedSet, Bag, Sequence로 구분됩니다.

중복 허용	순서 고려	컬렉션
아니요	아니요	Set
아니요	예	OrderedSet
예	아니요	Bag
예	예	Sequence

중복되지 않는 다는 것은 {1, 2}와 같이 컬렉션 안에 같은 요소가 있어서는 안 된다는 것입니다. 중복을 허용한다면 {1, 2, 2, 2}와 같이 컬렉션 안에 같은 요소가 여러 번 나타나도 된다는 것입니다.
순서가 있다는 것은 {1, 2, 3}에서 '1은 첫 번째 위치에 2는 두 번째 위치에 있고, 3은 세 번째 위치에 있다'라고 말할 수 있다는 것입니다.

Set은 기본 값입니다. 다중성의 최댓값이 *인데 추가적으로 아무것도 작성되어 있지 않다면 Set을 나타냅니다.
OrderedSet은 {ordered}로, Bag은 {bag}으로, Sequence는 {seq} 또는 {sequence}로 보통 다중성 다음에 작성합니다.

3.10 연관은 한 개의 연관이 몇 개의 클래스와 연결되는지에 따라 재귀(recursive), 이중(binary), 다중(n-ary)으로 구분됩니다.

연결되는 클래스가 하나인 경우 재귀, 둘 인 경우 이중, 셋 이상인 경우 다중이라 합니다. 대부분의 연관은 이중이다.

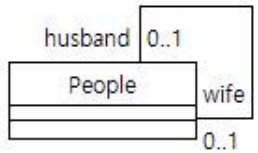
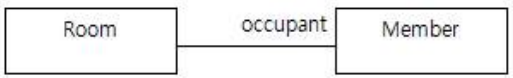
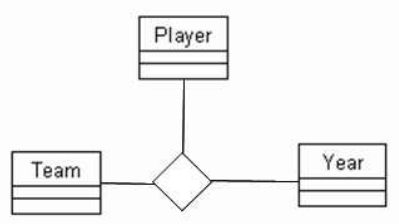


그림 5.17 재귀



5.18 이중



5.19 다중

우리는 다중 연관을 잘못된 것으로 간주하고 작성을 금합니다. 이에 대해서는 12장 컬레보레이션에 자세히 다룹니다.

3.11 연관에 이름과 읽는 방향을 나타낼 수 있습니다.

그림 5.20과 같이 연관 선 중앙에 연관 이름을 작성할 수 있습니다. 연관 이름 아래에는 읽는 방향을 작성할 수 있습니다.
5.20은 'Person은 Company를 위해 일한다.'를 나타낸 것입니다.

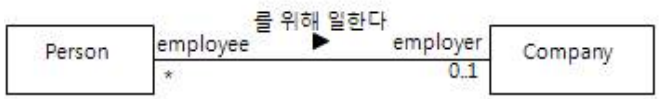


그림 5.20

우리는 연관이름 작성 또한 잘못된 형태로 간주하고 작성을 금합니다. 이에 대해서는 12장 컬레보레이션에 자세히 다룹니다.

3.12 접근성을 제약할 수 있습니다.

링크는 방향성이 없습니다. 양쪽 객체 모두에서 상대 객체를 찾을 수 있습니다. 그런데 한 쪽에서만 접근해도 되는 경우가 있다면 양쪽 모두에서 서로에게 접근할 수 있게 하는 것은 비효율적입니다. 이런 경우를 위해 연관은 방향성을 제약합니다.

방향성에는 방향성에 대한 결정여부까지 표현할 수 있습니다.
그림 5.21 (a)는 아직 방향성을 결정하지 않은 상태입니다. (b)는 A에서 B로는 접근하지만, B에서 A로는 접근할 수 없음을 나타냅니다. (c)는 양쪽 모두에 접근할 수 있음을 나타냅니다.

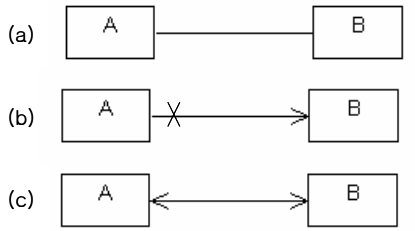


그림 5.21

우리는 방향성 결정까지 표현하는 것은 불필요하다고 여기고, 양방향을 표현할 때 (a) 같이, 일방향을 표현할 때는 x를 사용하지 않고 한쪽 방향만 작성하도록 합니다.

3.13 연관에 의해 생기는 정보는 연관클래스에 작성합니다.

링크된 객체들을 특정 조건으로 한정하고 싶을 때는 한정자(Qualifier)를 사용합니다.

학생이 어떤 강좌를 수강한 결과로 학점이 나왔다면 이 학점은 학생에 둘 수도 없고 강좌에 둘 수도 없습니다. 학생에 학점을 두면 어떤 강좌에 대한 것인지를 알 수 없고, 강좌에 학점을 두면 어떤 학생의 학점인지를 알 수 없습니다.

이를 위해 UML은 연관클래스를 제공합니다. 그림 5.22가 연관클래스입니다. 연관클래스는 클래스와 같이 작성하고 연관에서 실선으로 연결합니다.

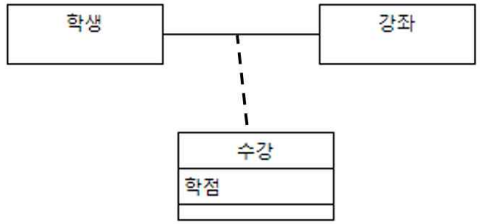


그림 5.22

은행은 다수의 사람들과 연결될 수 있습니다. 이러한 연결은 은행에 연결된 모든 사람들을 찾도록 해 줍니다. 그러나 특정 계좌번호를 가진 사람을 찾고자 할 때는 유용하지 않습니다. 계좌번호로 연결을 한정해야 합니다. 계좌번호를 한정자로 해서 은행과 사람들 사이의 연결을 한정하는 것입니다.

한정자는 그림 5.23과 같이 한정되는 연관지점에 사각형 모양으로 작성합니다.

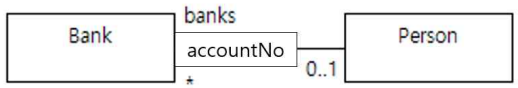


그림 5.23

우리는 연관클래스와 한정자 또한 잘못된 형태로 간주하고 작성을 금합니다. 이에 대해서는 12장 컬레보레이션에 자세히 다룹니다.

4절. 클래스 다이어그램 작성

그림 5.24와 같이 작성된
객체 다이어그램을 가지고 클래스 다이어그램을 작성합니다.

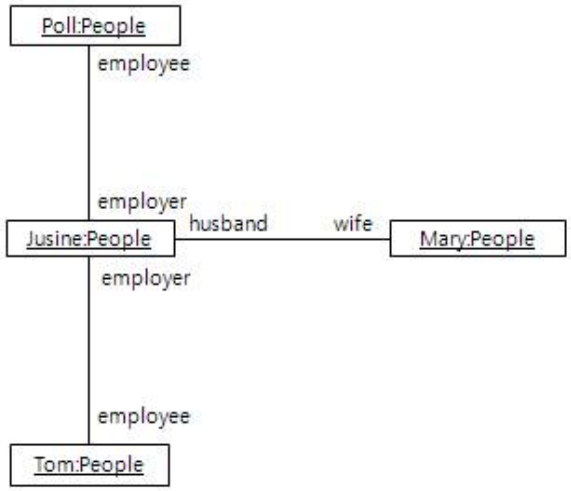


그림 5.24

4.1 클래스를 작성합니다.

클래스는 객체의 개념으로, 객체 이름 뒤에 작성됩니다.
클래스는 People 하나만 있습니다.

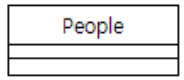


그림 5.25

4.2 속성을 작성합니다.

속성은 특성의 개념입니다. 그림 5.24에는 특성이 보이지 않습니다. 객체의 이름 부분만 반영하면 됩니다.

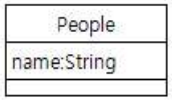


그림 5.26

4.3 연관을 작성합니다.

연관은 링크의 개념으로, 역할쌍에 의해 결정됩니다.
역할쌍은 'employer-employee'와 'husband-wife'가 있습니다.

하나의 클래스만 있으므로 재귀 연관입니다.

도메인 전문가의 설명을 들으니 양쪽 모두에서 서로를 알아야 한다고 합니다. 양방향입니다.

다중성은 employer-employee에 연결되지 않은 Mary가 있고, husband-wife에 연결되지 않은 Tom이 있기 때문에 최솟값은 0입니다. employee는 여럿 있을 수 있음으로 다가 되고, husband와 wife는 서로에 대해 많아야 하나만 있을 수 있음으로 1입니다.

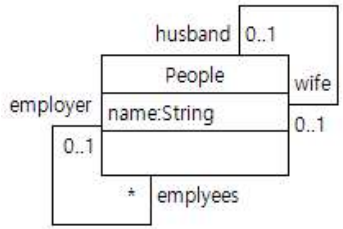


그림 5.27

마치기 전에

이번 장에서는 데이터타입과 클래스와 연관에 대한 그래픽 표현에 대해서 설명했습니다.

연관을 설명하는 여러 부분에서 '잘못된 형태로 간주하고 작성을 금한다.'가 등장합니다.
우리는 처음부터 잘못된 형태로 생각하는 것들을 제외해서, 왜 이것이 잘못된 것인가에 대한 설명의 필요성을 없애는 것에 대해서 고심했습니다.
고심 끝에 '내용을 모르면서 안 쓰는 것과, 내용을 알고 그것이 왜 잘못되었는지를 알고 안 쓰는 것은 다르다'라고 결론을 내렸습니다.
왜 이러한 것들이 사용되는지, 왜 이러한 것들이 잘못된 것인지에 대해서는 판단했습니다.

4절의 내용을 여러 번 반복해서 해 보십시오.
현장에서 객체 다이어그램을 작성하고, 그것을 기반으로 클래스 다이어그램을 작성하지는 않습니다.
하지만 실체와 개념 사이의 관계에 대해서 좀 더 명확히 이해하는 데는 도움이 됩니다. 특히 링크를 통해 연관 작성을 반복하다보면 연관에 대해 좀 더 분명한 감이 잡힙니다.

우리는 감을 잡는 것을 중요하게 여깁니다.
설명 내용에 따라 용어정의를 외우고 개념에 대해 이해했더라도 '아하'라는 감이 없으면 내 것으로 만들었다고 보기 어렵기 때문입니다.
내 것이 되지 않은 것을 가지고는, 현장에서 확신을 가지고 모델링할 수 없기 때문입니다.