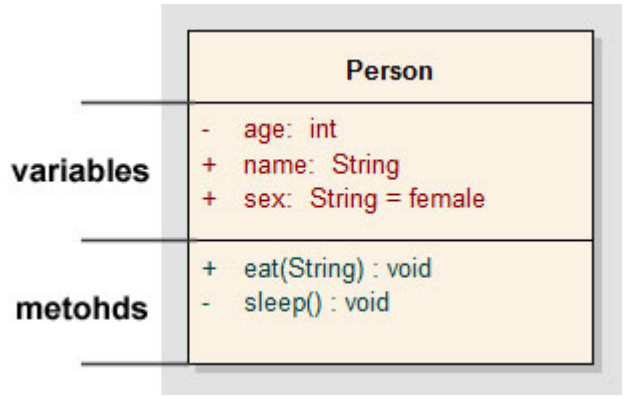
UML - 기본편 (기본 표기 형식 및 관계 표현법)

관리자 2008.11.18 14:50:08

Class 및 Class instance 의 기본 표기 형식

Class 표기형식

UML Diagram 중에서 가장 기본적인 표현 단위인 클래스의 표기형식을 알아보자.



+ : public
– : private
: protected

* variables, methods 는 생략이 가능하나 class 이름은 반드시 명시해주어야 한다.

위의 class 를 소스코드로 표현하면 아래와 같다.

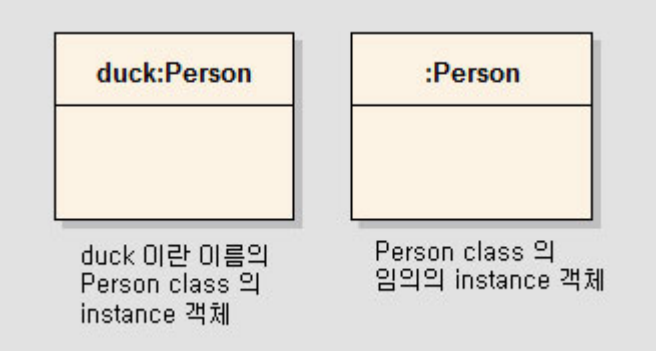
```
public class Person
{
    private var age: int;
    public var name: String;
    public var sex: String = "female";


    public function eat(food:String): void
    {
    }

    private function sleep(): void
    {
    }

}
```

Class instance 객체의 표기형식

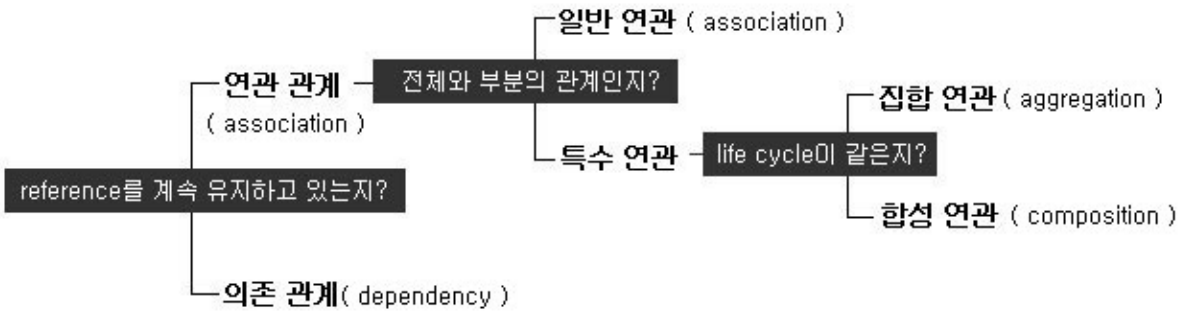


Relationships(관계 표현)

서로 의미있는 클래스들의 관계에는 크게 4가지 종류가 있다.

일반적인 의미의 연결 관계인 연관(association) 관계, 전체와 부분을 나타내는 집합(aggregation) 관계, 다른 클래스의 재산을 물려받는 상속(inheritance) 관계, 그리고 한 클래스가 다른 클래스에 영향을 미치는 의존(dependency) 관계가 있다.

이 중에서도 association 과 aggregation, composition 이 세가지 관계가 가장 헷갈릴 수 있는데 간략하게 정리를 해보자면 아래의 그림과 같다. 회색 부분이 각각의 관계를 구분짓는 기준이 된다고 볼 수 있다.



association 과 dependency 를 구분짓는 가장 큰 기준은 ' 참조하는 클래스 인스턴스의 레퍼런스를 계속 유지하고 있느냐, 아니냐 ' 이다. 아래의 소스 코드들을 보면서 이해해보자.

[소스 코드 1] dependency

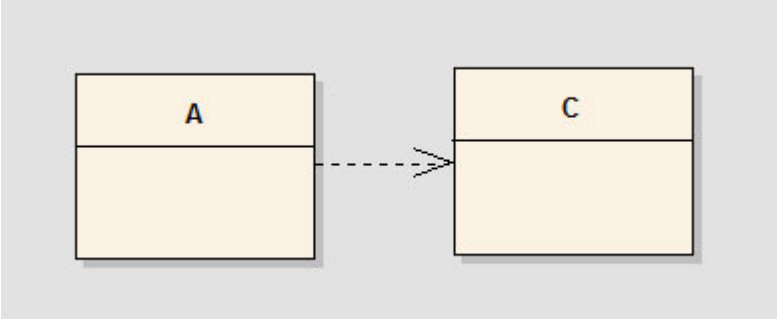
```
package classes
{
    public class A
    {
        private var c:C;

        public function A()
        {

        }

        private function methodA():void
        {
            // 의존 관계 : 이 메소드 내부에서만 c의 레퍼런스를 유지함
            // 함수 실행이 끝나면 c 의 참조도 사라짐
            var objC:C = new C();
            c.methodC();
        }
    }
}
```

[UML] dependency (A ----> B)



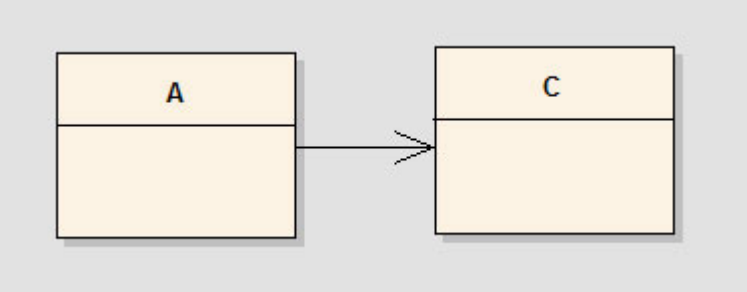
[소스 코드 2] association

```
package classes
{
    public class A
    {
        private var c:C;

        public function A()
        {
            // 연관 : 클래스 C 에 대한 reference를 계속 유지하고 있음
            this.c = new C();
        }

        private function methodA():void
        {
            this.c.methodC();
        }
    }
}
```

[UML] association (A - > B)



◆ dependency(의존 관계)

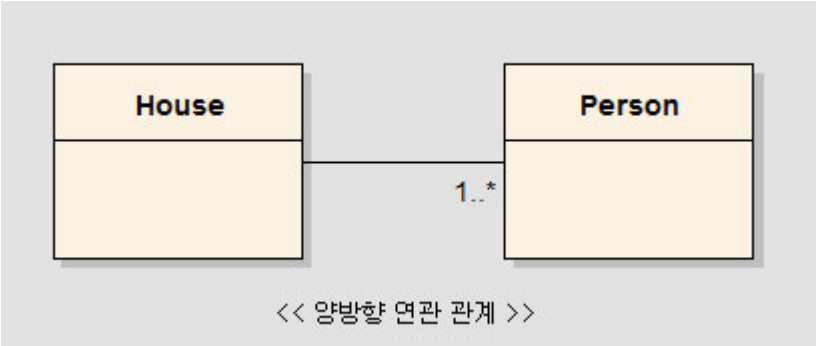
클래스가 연관, 상속, 집합 관계로 엮여 있는 것은 아니지만, 한 곳이 변경되면 그것을 사용하는 다른 곳도 같이 변경해줘야 하는 관계를 표현할 때 주로 사용한다. 단, 주의해야 할 점은 association 과 달리 dependency 의 경우에는 클래스 인스턴스의 레퍼런스를 유지하고 있지 않다는 점이다. 레퍼런스를 계속적으로 유지하게 되면 이는 association 으로 표현해야 한다.

주로 다음과 같은 세 가지 경우에 의존 관계로 표현한다.

1. 한 클래스의 메소드가 다른 클래스의 객체를 인자로 받아 그 메소드를 사용한다.(가장 일반적)
2. 한 클래새의 메소드가 또 다른 클래스의 객체를 반환한다.
3. 다른 클래스의 메소드가 또 다른 클래스의 객체를 반환한다. 이때 이 메소드를 호출하여 반환되는 객체의 메소드를 사용한다.

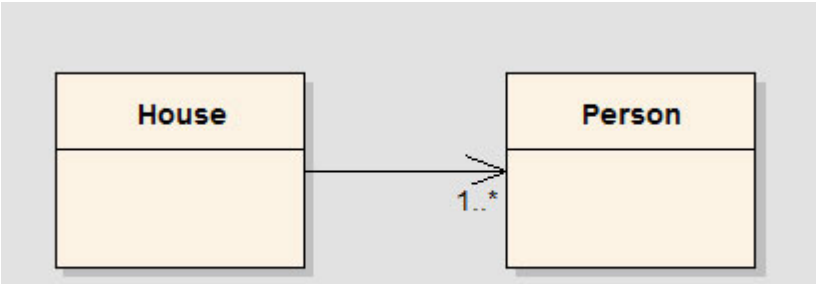
◆ association(연관 관계)

한 객체가 다른 객체와 연결되어 있음을 나타낼 때 그들을 연관관계로 지칭한다. 이러한 연관관계에서 중요하게 볼 점은 ' 연관 관계의 방향(navigability) 과 멀티플리시티(multiplicity) 이다.



양방향 연관 관계 : 연결된 클래스들이 서로의 존재를 알고 있다는 의미이다.

위의 UML 을 해석하자면 House 와 Person 클래스는 서로의 존재를 알고 있으며, 반드시 한 사람 이상이 House에 속해야 한다는 것을 뜻한다.



<< 단방향 연관 관계 >>

단방향 연관 관계 : House 클래스는 Person 클래스의 존재를 알고 있지만, Person 은 House 클래스의 존재를 모르고 있다고 이해하면 된다. 이와 같은 경우는 House 클래스만 Person 클래스에 대한 참조값을 가지고 있고, Person 은 House 에 대한 어떠한 참조값도 가지고 있지 않는다.

관계 표현을 나타낸 그림에서 보면 일반 연관과 특수 연관이라고 나뉘어 지는데, 특수 연관이라는 것은 임의로 만든 단어이다. 일반 연관이란 앞에서 살펴본 association 을 나타내며, association 중에서도 ' 부분과 전체 ' 로 나눌 수 있는 관계를 aggregation 과 composition 으로 묶어서 분류한 것이다.

aggregation 과 composition 은 모두 association 의 한 특별한 형태로 각각을 구분하는 기준은 ' life cycle 이 같느냐, 같지 않느냐 ' 이다. life cycle 이란 클래스 인스턴스의 생명 주기를 말하는 것으로 생성에서 소멸까지의 과정을 말한다. 즉, life cycle 이 같다는 것은 관계된 클래스 혹은 그 인스턴스의 생성과 소멸이 동시에 이루어 진다는 것을 뜻한다.

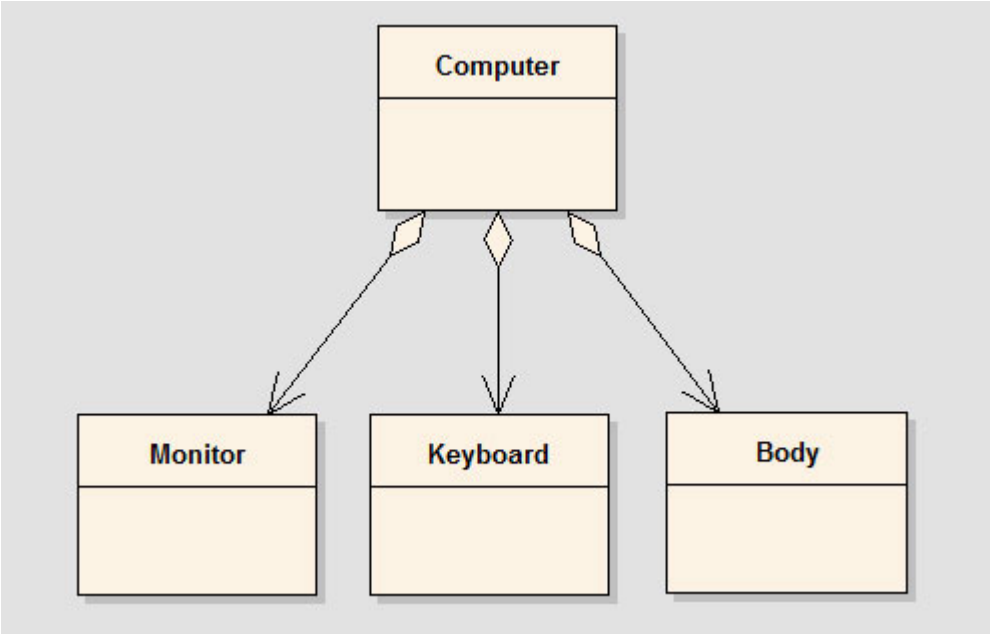
쉽게 예를 들어 표현하자면 모자와 안경을 쓴 사람을 놓고 보자. 현재 이 사람을 구성하고 있는 요소에는 눈, 팔, 다리와 같이 사람이 죽으면 같이 없어지는 요소들이 있고, 안경 모자와 같이 바꿔 사용할 수 있는 요소들이 있다. 즉, 눈, 팔, 다리는 사람과 life cycle 이 같은 composition 관계 이고, 안경이나 모자는 aggregation 관계인 것이다. 이러면 이해가 좀 쉽게 갈라나? ^-^

[소스 코드 1] aggregation

```
package classes
{
    public class Computer
    {
        private var monitor:Monitor;
        private var body:Body;
        private var keyboard:Keyboard;

        public function Computer(_monitor:Monitor, _body:Body, _keyboard:Keyboard)
        {
            // 집합 연관 : 부분과 전체의 관계
            // 부분이 되는 객체를 외부에서 생성하여 넘겨받음
            // 따라서 computer 클래스가 없어져도 부분이 되는 객체들은 사라지지 않음
            this.monitor = _monitor;
            this.body = _body;
            this.keyboard = _keyboard;
        }
    }
}
```

[UML] aggregation

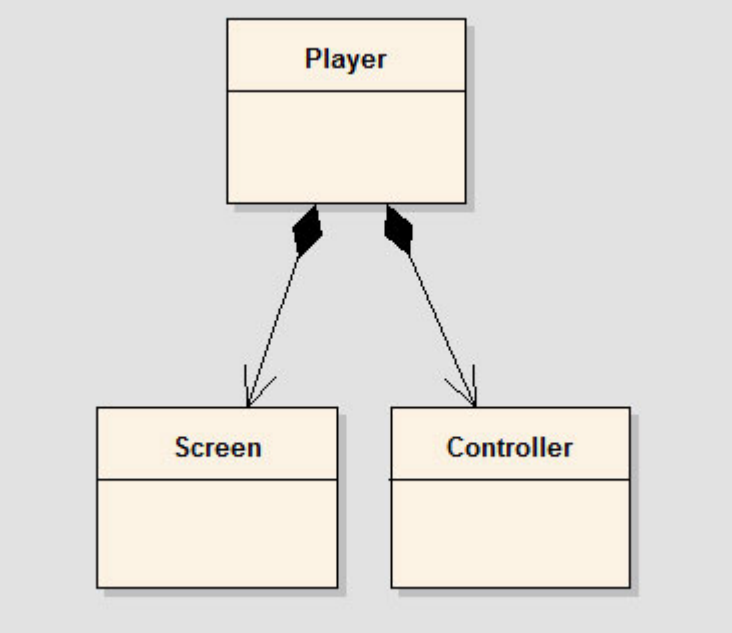


[소스 코드 2] composition

```
package classes
{
    public class Player
    {
        private var screen:Screen;
        private var controller:Controller;

        public function Player()
        {
            // 합성 : 집합 관계 중에서도 강한 집합체의 의미를 가짐
            // 부분을 이루는 객체가 없이는 전체가 아무 의미를 갖지 못함
            // Player 클래스가 사라져 버리면 내부에서 생성된 screen, controller도 같이 사라짐
            this.screen = new Screen();
            this.controller = new Controller();
        }
    }
}
```

[UML] composition



◆ 상속 관계 (inheritance)

[소스 코드]

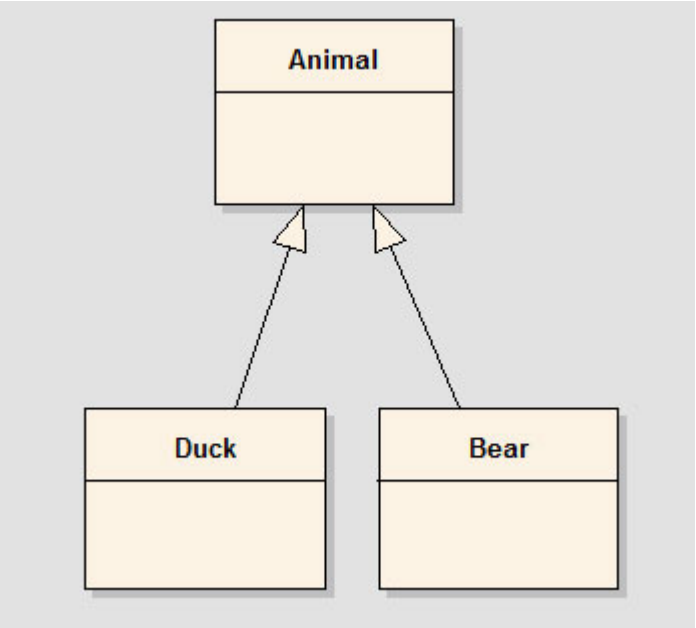
```
public class Duck extends Animal
{

}

public class Bear extends Animal
{

}
```

[UML] inheritance



◆ interface

[소스 코드]

```
public class LaserPrint implements IPrintable
{

}

public class XrayMachine implements IPrintable
{

}
```

[UML]