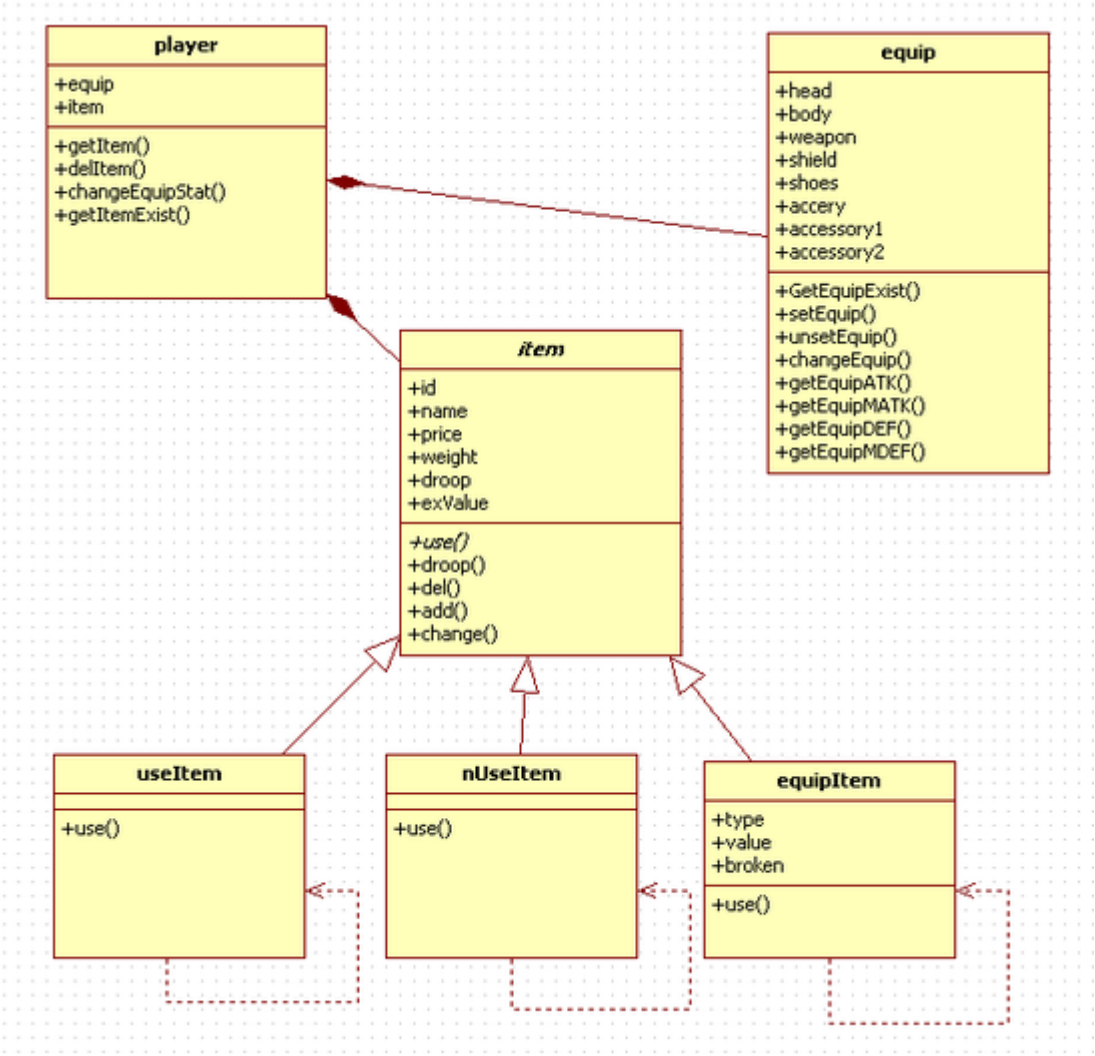
UML

UML(Unified Modeling Lauguage)이란 객체지향 프로그래밍이 발전하면서 모델링 언어의 표준화에 따라 만들어진 언어이다. 이거 모르면 객체 지향 언어 해봤다고 말할 자격 없다. UML은 다이어그램으로 표현되며 다이어그램도 여러가지가 있으나 이 중 많이 쓰이는 것은 Class Diagram, Usecase Diagram, Sequence Diagram 정도이다. 아래는 Class Diagram의 예이다.



관련 프로그램

UML을 그릴 수 있는 프로그램 중 무료로 얻을 수 있는 대표적인 프로그램은 StarUML이다. 현재는 상용 버전인 StarUML NS도 출시되었지만 무료 버전도 다운로드 가능하다. 아래 주소에서 다운 가능하다.

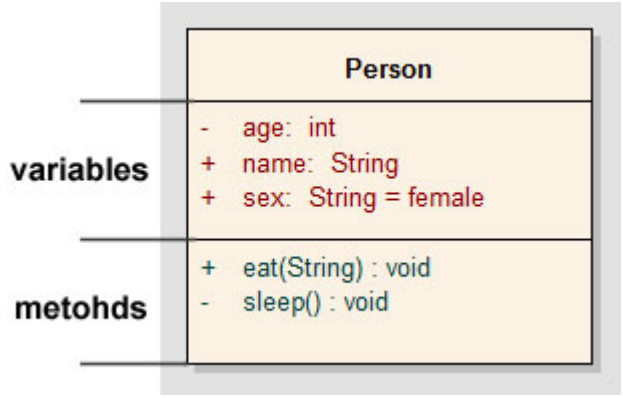
- 공식홈페이지
- http://sourceforge.net/projects/staruml/files/staruml/5.0/

Class Diagram

UML 중 가장 중요하고 많이 쓰이는 다이어그램이다. 이것만 알아도 프로그램 설계하는데 큰 도움을 준

다.

기본 표기 형식



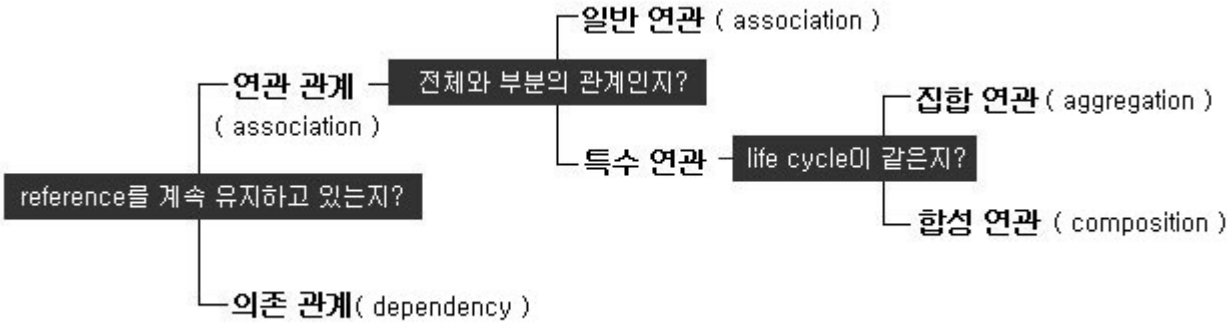
이름 뒤에 붙어있는 type은 생략 가능하다. (그림은 ActionScript 3.0 형식의 변수, 함수 선언) 그림에서 variable과 method 앞에 붙어 있는 기호는 다음을 의미한다.

기호	의미
+	public
-	private
#	protected

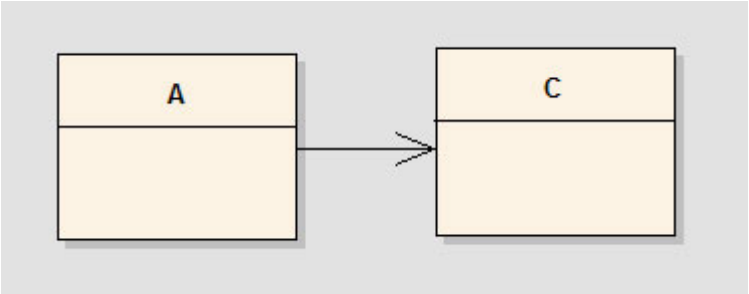
클래스의 실제 코드에 들어가는 모든 변수와 함수를 넣을 필요는 없고 중요한 것만 넣으면 된다. 클래스 다이어그램의 목적이 코드 구조를 쉽게 보기 위한 것이기 때문이다. 프로그램의 전체 클래스를 다 표현하는 경우에는 변수, 함수까지 포함하면 너무 많아지므로 suppress 기능을 통해 클래스 이름만 표현하기도 한다.

관계 표현

클래스는 다른 클래스에 포함되기도 하고 상속관계를 이루기도 하는 등 여러 가지 종류의 관계를 가질 수 있다. 이를 유형별로 분류하면 다음 그림과 같다.



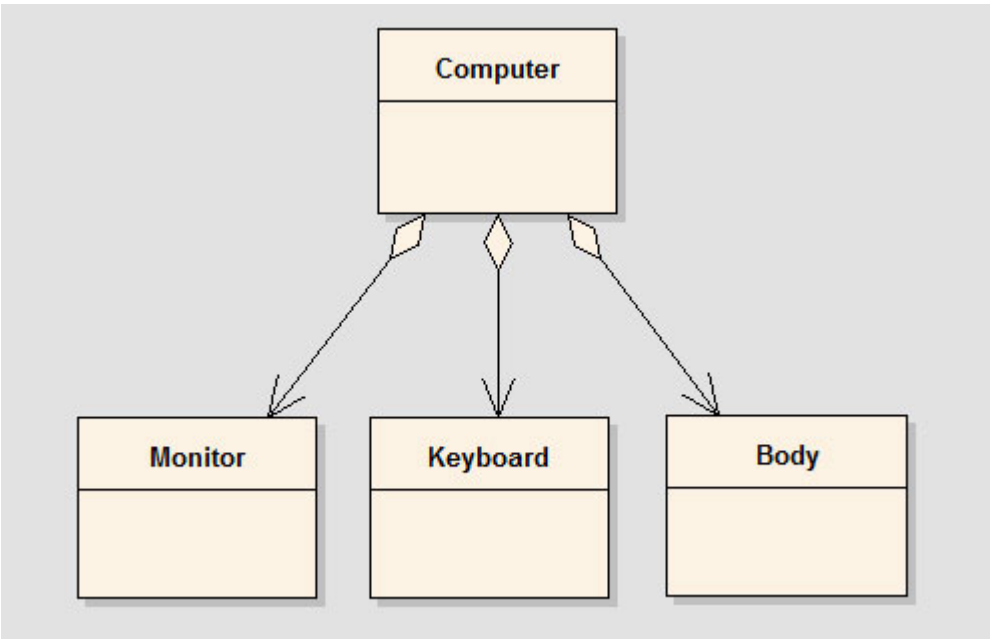
연관 관계




```
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();
        }
    }
}
```

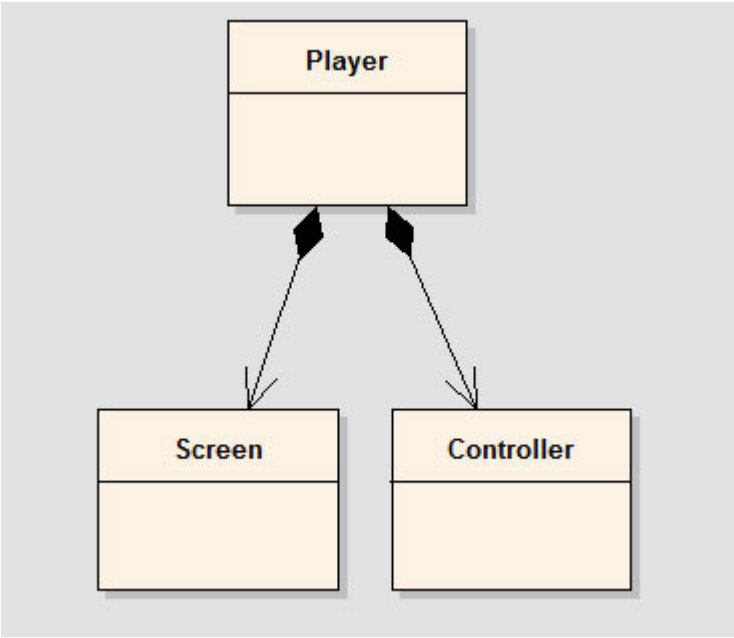
일반적으로는 위 그림처럼 표현하나 life cycle(클래스 또는 그 인스턴스의 생성, 소멸 주기)에 따라 aggregation, composition으로 분류할 수도 있다.

aggregation



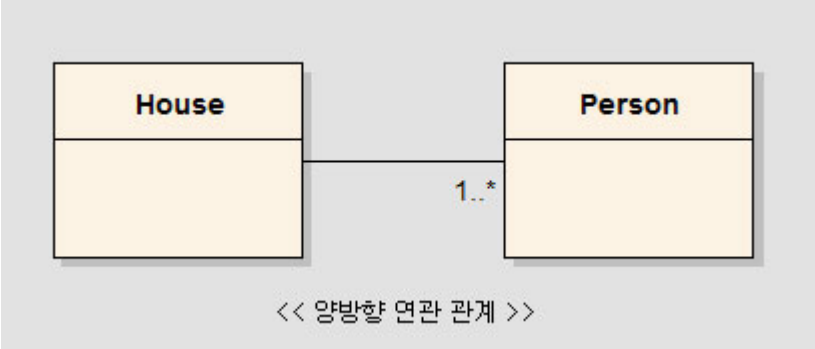
```
package classes
{
    public class Computer
    {
        private var monitor:Monitor;
        private var body:Body;
        private var keyboard:Keyboard;

        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;
        }
    }
}
```

composition



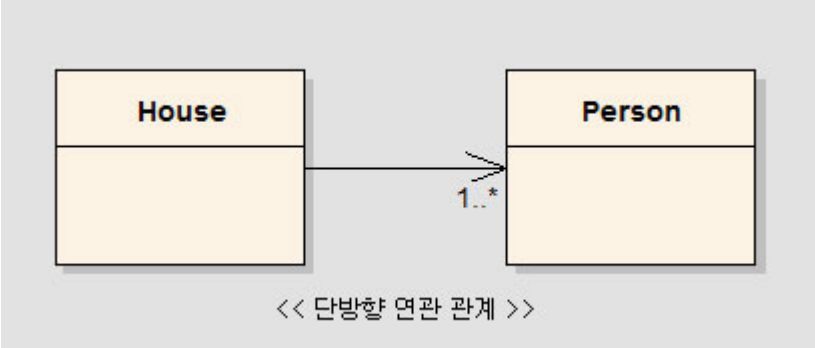
```
package classes
{
    public class Player
    {
        private var screen:Screen;
        private var controller:Controller;

        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();
        }
    }
}
```

양방향 연관 관계



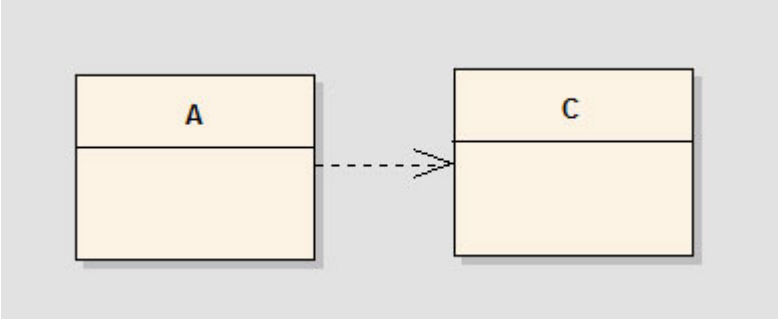
연결된 클래스들이 서로의 존재를 알고 있다는 의미이다. 위의 UML 을 해석하자면 House 와 Person 클래스는 서로의 존재를 알고 있으며, 반드시 한 사람 이상이 House에 속해야 한다는 것을 뜻한다.

단방향 연관 관계



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.

의존 관계




```
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();
        }
    }
}
```

참고

StarUML 5.0 사용자 가이드
UML - 기본편 (기본 표기 형식 및 관계 표현법)
UML: 클래스 다이어그램 기초