* UML(Unified Modeling Language)이란?

소프트웨어 공학에서 사용되는 표준화된 범용 모델링 언어.

소프트웨어 개념을 다이어그램으로 그리기 위해 사용하는 시각적인 표기법

---

* UML(Unified Modeling Language)의 필요성

1) 의사소통에 유용.

2) 대규모 프로젝트 구조의 로드맵 구축에 유용.

3) 개발할 시스템 구축에 대한 기초 마련.

4) 백엔드 문서용으로 적합.(해당 프로젝트를 다른 팀이 맡을 경우 유용)

---

* 유스케이스 다이어그램(Usecase Diagram) 이란?

Actor와 시스템이 수행하는 활동간의 관계를 표시하며,

시스템의 기능적인 요구사항을 설명하기 위한 도구

* 유스케이스 다이어그램의 구성요소

System : 만들고자 하는 어플리케이션

Actor : 시스템의 외부에 있으면서 시스템과 상호 작용을 하는 사람 또는 다른 시스템

Usecase : 시스템이 액터에게 제공해야하는 기능의 집합

---

* 시스템 요구사항에서 유스케이스 다이어그램 액터 / 유스케이스 도출.

[온라인 쇼핑몰 시스템 요구사항]

- 온라인 쇼핑몰의 회원을 **회원**과 **비회원**으로 구분한다.
- 회원은 ID, 이름, 주민번호, 마일리지 회원등급 정보를 갖는다.
- 비회원은 이름, 주민번호 정보를 갖는다.
- 회원은 **Login, 등록, 수정, 삭제, 조회** 처리가 가능하며, 비회원은 등록, 수정, 삭제, 조회 처리가 불가능하다.(비회원은 Login처리만 가능하다.)

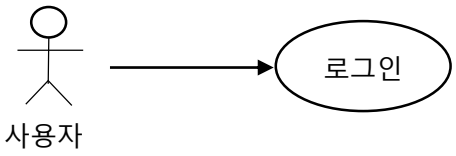
액터 : 회원, 비회원

유스케이스(기능) : Login, 등록, 수정, 삭제, 조회

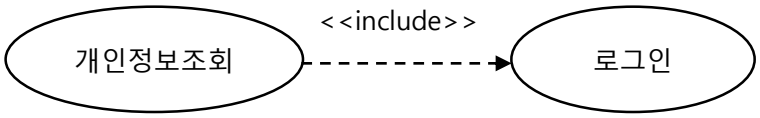
---

* 유스케이스 다이어그램의 관계 종류.

- **연관 관계** : 유스케이스와 액터간 상호 작용을 의미하는 관계

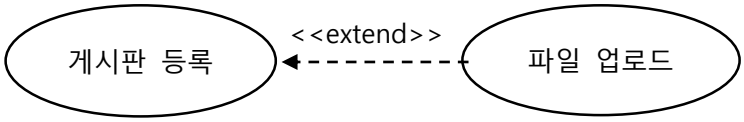


- **포함 관계** : 한 유스케이스가 수행되기 위해 다른 유스케이스의 기능이 포함되는 관계
  (**반드시 선행으로 수행** 되어야하는 관계)
  (**먼저 수행**되어야 하는 유스케이스 쪽으로 **화살표 머리가 향함**)

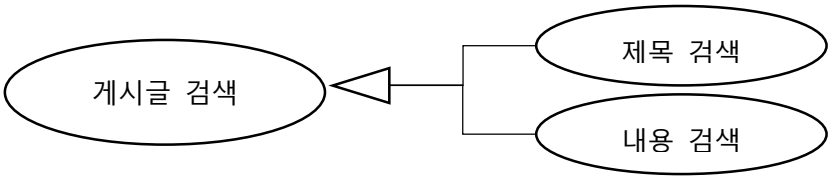


- **확장 관계** : 기존 유스케이스에서 특정 조건이나 액터의 선택에 따라
  다른 유스케이스로 기능을 확장할 수 있는 관계
  (**선택적으로** 다른 유스케이스 기능을 수행할 수 있는 관계)
  (확장되는 유스케이스에서 **선행 되는 유스케이스 쪽으로** 화살표 머리가 향함.)

-

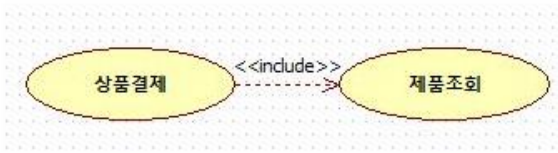


- **일반화 관계** : 유사한 유스케이스 또는 액터들을 모아 그들을 추상화한
  유스케이스/액터를 연결시켜 그룹핑하는 것



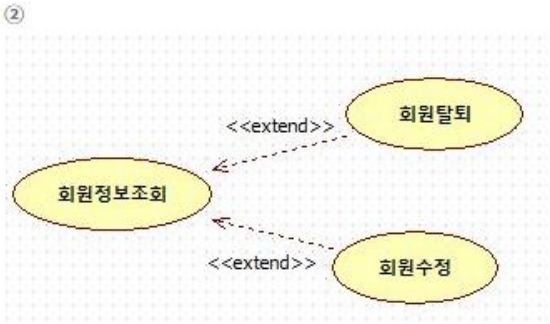
* 다음 유스케이스 다이어그램 해석 예시.

①



1) 상품 결제를 하기 위해서는
제품 조회를 반드시 먼저 수행해야 한다.

②



2) 회원 정보 조회를 하고 나면 회원 탈퇴 또는
회원 수정을 선택적으로 수행 할 수 있다.

---

* 유스케이스 정의서란?

유스케이스 다이어그램을 보완하기위한 산출물로 유스케이스다이어그램이 시스템의 기능을 표현한다면 각각의 유스케이스에 대해서 해당 유스케이스가 어떻게 수행되는지를 표현한다.

---

* 유스케이스 정의서에 포함되어 있는 항목

1. 유스케이스명 2. 관련 액터 3. 개요 4. 사전 조건 5. 사후 조건	6. 정상흐름 7. 대안흐름 8. 예외흐름 9. 비기능적 요구사항

* 기능적 요구사항 / 비기능적 요구사항

- **기능적 요구사항**
  목표로 하는 어플리케이션의 구현을 위해 소프트웨어가 가져야 하는 기능적인 속성

- **비기능적 요구사항 :**
  어플리케이션의 품질 기준 등을 만족시키기 위해
  소프트웨어가 가져야하는 성능, 사용의 용이

---

* 클래스 다이어그램의 메소드 표현 방법

- 접근제어자 메소드명(매개변수) : 리턴타입 순으로 작성한다.
- 접근제어자는 public(+), default(~), protected(#), private(-) 기호로 각각 표현
- 매개변수는 (변수명:데이터타입) 형태로 표현한다.

메소드 소스 코드	클래스 다이어 그램 표현 방법
public String printMsg(String message){}	+printMsg(message:String):String

---

* 클래스 다이어그램의 예약어 표현 방법

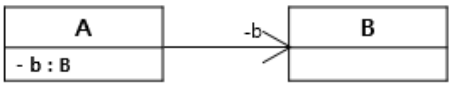
- **static** : 밑줄
- **Constant**(상수) : 대문자
- **Abstract**(추상) : 기울어진 문자

* 클래스 다이어그램의 관계 및 Java 코드 작성 방법

연관 관계(Associaction)

한 클래스가 **멤버 변수로** 다른 클래스 참조 변수를 가지는 관계.
한 클래스가 다른 클래스에서 제공하는 기능을 사용하는 상황을 의미

-



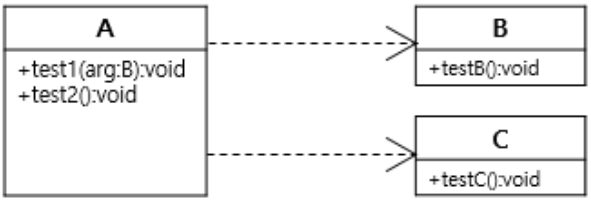

```
public class A{
        private B b;
}
```

```
public class B{

}
```

(A 클래스가 B 클래스를 멤버 변수로 참조하고 있다.)

의존 관계((Dependency)

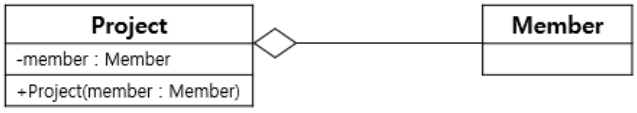
클래스간의 **지역 변수로** 참조하는 관계를 의미
ex) A -----> B 인경우 A클래스의 지역변수에서 B클래스를 참조함




```
public class A{
        public void test1(B arg){
                arg.testB();
        }
        public void test2(){
                C c= new C ();
                c.testC();
        }
}
```

집합 관계(Aggregation)

멤버 변수로 클래스를 참조하는 관계. (연관 관계와 비슷)
참조하는 클래스와 참조되는 클래스의 생명주기가 독립적임.
-> 생성자 매개변수로 전달 받은 값을 멤버 변수에 대입함.



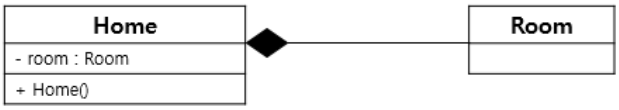

```
public class Project{
        private Member member;

        public Project(Member member){
                this.member  = member;
        }
}
```

프로젝트는 멤버로 구성된다
멤버는 프로젝트의 부분이다
멤버는 다른 프로젝트에도 공유된다

➔ Member 객체가 생성되는 시기는 Project 객체가 생성되는 것과 관련이 없다.
(Project 생성자에서 이미 만들어진 Member 객체를 매개변수로 전달받아
멤버 변수에 세팅하는 것.)

합성 관계(Composition)

멤버 변수로 클래스를 참조하는 관계
참조하는 클래스와 참조되는 클래스의 생명주기가 일치함.



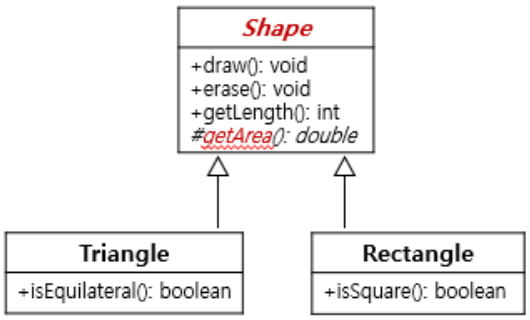

```
public class Home{
        private Room room;

        public Home(){
                this.room  = new Room();
        }
}
```

집은 방으로 구성된다.
방은 집의 부분이다.
집이 사라지면 방도 사라진다.

➔ Home 객체 생성시 Home 생성자에 의해서 Room 객체도 같이 만들어짐.

- **일반화 관계(Generalization)**

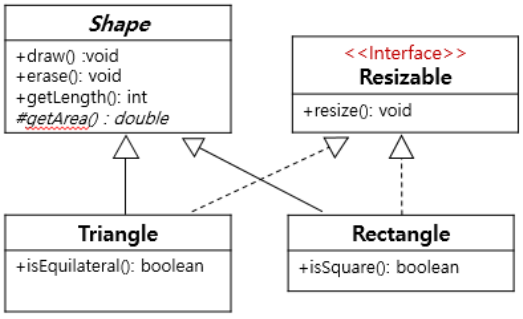
클래스 상속 관계를 의미




```
public abstract class Shape {
        public void draw() {...}
        public void erase() {...}
        public int getLength() {...}
        protected abstract double getArea();
}
public class Triangle extends Shape {
        public boolean isEquilateral() {...}
        protected double getArea() {...}
}
public class Rectangle extends Shape {
        public boolean isSquare() {...}
        protected double getArea() {...}
}
```

실체화 관계(Realization)

Generalization 과 마찬가지로 상속관계를 의미하지만 인터페이스 implement 를 의미




```
public interface Resizable {
        void resize();
}
public class Triangle extends Shape
                        implements Resizable  {
        public boolean isEquilateral() {...}
        protected double getArea() {...}
        public void resize() {...}
}
public class Rectangle extends Shape
                        implements Resizable {
        public boolean isSquare() {...}
        protected double getArea() {...}
        public void resize() {...}
}
```