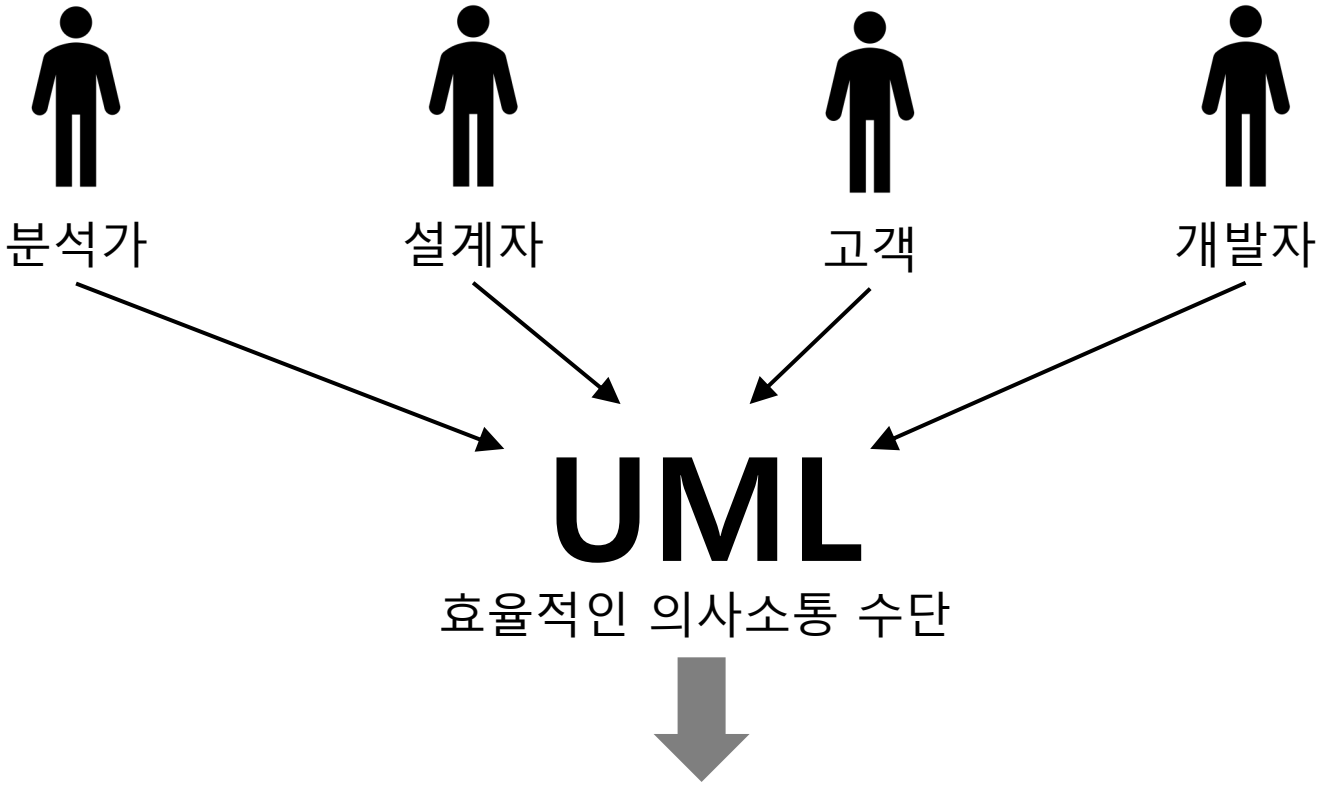
UML 사용법

엄진영

참고 문헌

- 객체지향 설계와 분석을 위한 UML 기초와 응용 [한빛아카데미]
- UML 2 and the Unified Process, 2nd [Addison-Wesley]
- Applying UML and Patterns [Pearson]

UML Unified Modeling Language 이란?



시스템 설계를 위한 시각적인 설계 표기

명세화　　시각화　　문서화

UML Unified Modeling Language 이란?

UML

✕ 개발 방법론

✕ 개발 프로세스

✕ 프로그래밍 언어

→ **표준화된 모델링 언어**

객체 지향 시스템을 가시화하고 명세화하고 문서화하는 도구

UML Unified Modeling Language 이란?

모델링

- 시스템을 구축할 때 개발자가 고민하고 결정하는 모든 활동.
  예) 요구사항 정의, 분석, 설계

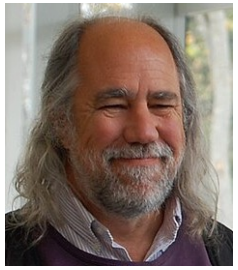
모델링 언어: UML, ERD, DFD 등

모델

- 모델링 활동의 결과.
  예) 요구사항 모델, 분석 모델, 설계 모델

UML Unified Modeling Language 이란?

모델링 방법



Grady Booch

Booch method

설계 중심의 방법론
→ 시스템을 몇 개의 뷰로 분석

James Rumbaugh

OMT

Object-Modeling Technique

객체모델, 동적모델, 기능모델
→ 객체 속성과 객체 간의 관계를 규명

Ivar Jacobson

OOSE

Object-Oriented Software Engineering

유스케이스를 강조한 방법론
→ 외부 행위자와 상호작용하는 시스템 요구사항 정의

UML

1997년 OMG Object Management Group
: 객체 지향 시스템을 모델링하는 표준 언어로 채택

1997 – UML 1.0
2000 – UML 1.3
2001 – UML 1.4
2005 – UML 2.0
2009 – UML 2.2
2010 – UML 2.3
2011 – UML 2.4.1
2015 – UML 2.5
2017 – UML 2.5.1

UML 구조

UML

- **Building blocks**
  → UML 모델을 구성하는 가장 기본적인 요소. 모델의 '어휘'를 제공,
  예) Things, Relationships, Diagrams

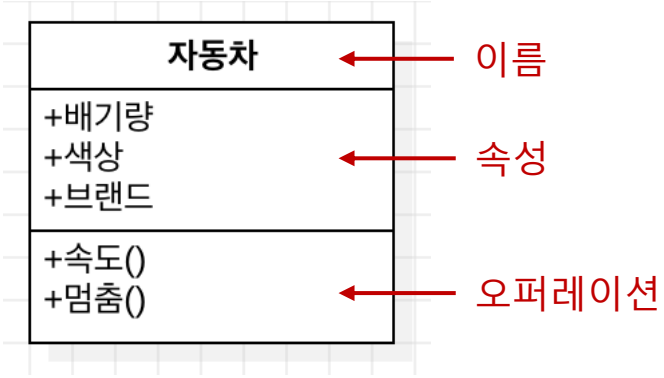
- **Common mechanism**
  → UML의 통일성과 일관성을 유지하는 도구.
  예) Specifications, Adornments, Common divisions, Extensibility mechanisms

- **Architecture**
  → 시스템의 조직적 구조 및 설계 원칙에 대한 UML의 시각, 전략적인 큰 그림.
  예) 4+1 뷰
  (Logical/Process/Implementation/Deployment View + Use case View)

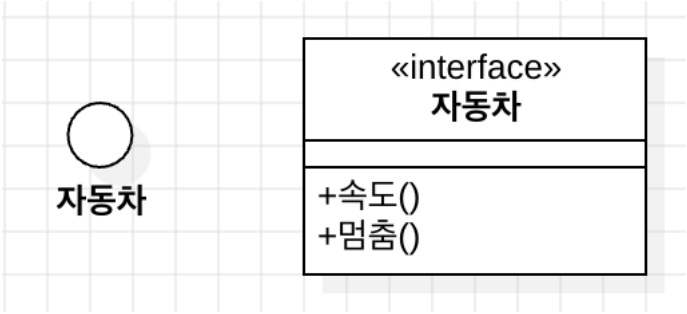
UML 구조 > Building blocks > Things > Structural

: 개념적, 물리적 요소를 표현하는 명사

클래스



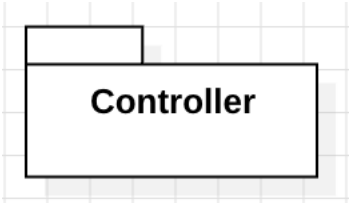
인터페이스



컴포넌트

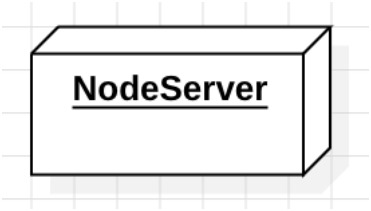
: 독립적으로 개발/배포
: 물리적이고 대체 가능
예) JAR, DLL, HTML

패키지



예) 개념적 그룹

노드



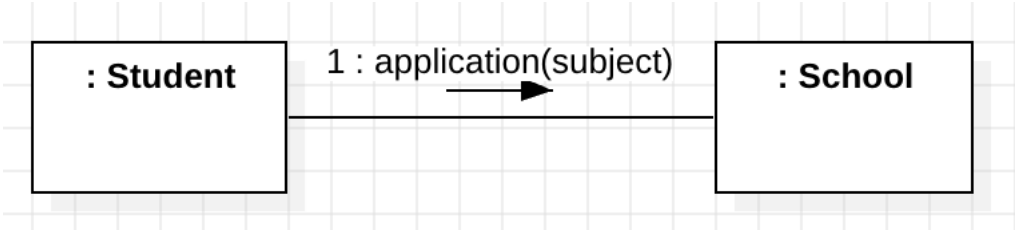
: 실행할 때 존재하는 물리적 요소
예) 물리적 요소, 전산 자원

UML 구조 > Building blocks > Things > Behavioral

: 모델의 동적인 부분을 동사로 표현하며, 시간과 공간에 따른 동작을 나타냄

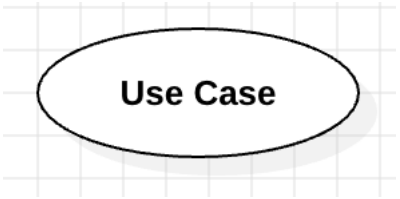
교류

목적을 달성하기 위해 특정한 문맥에 속한
객체들 간에 주고받는 메시지로 구성.



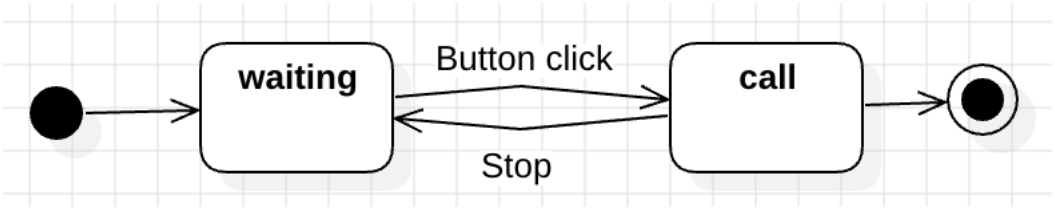
유스케이스

시스템이 수행하는 활동들을 순차적으로
기술하며, 액터에게 의미 있는 결과 값 제공.



상태 머신

외부 이벤트에 대한 객체의 상태와
상태의 변화 순서를 기술.

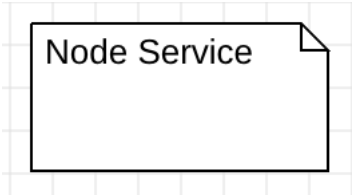


UML 구조 > Building blocks > Things > Annotational

: 모델링에 참여하지는 않지만 모델링을 이해하는 데 필요한 정보나 설명을 표시함

노트

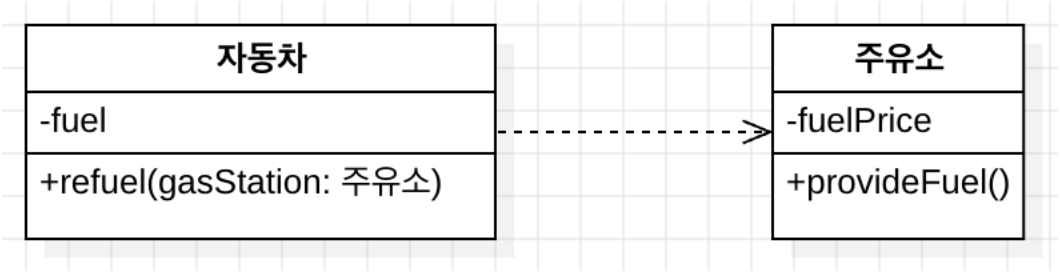
: 주석 또는 제약을 기술



UML 구조 > Building blocks > Relationships

Dependency

두 사물의 의미적 관계. 한 사물의 명세가 바뀌면 다른 사물에 영향을 준다.
한 클래스가 다른 클래스의 **기능을 일시적으로 사용하거나 참조하는 관계**.



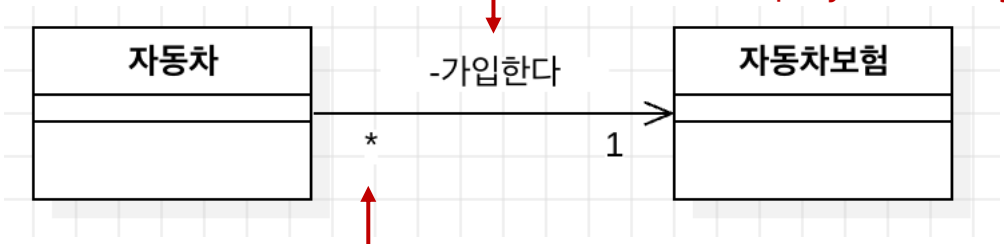
: 자동차는 주유소를 영구적으로 보유하지 않는다.
: 자동차의 refuel() 메서드가 실행될 때 **일시적으로 주유소를 사용**한다.

UML 구조 > Building blocks > Relationships

Association

객체 사이의 연결관계로, **지속적으로 유지되는 관계**다.
예) "자동차는 자동차보험에 가입한다."

연관 이름: 관계의 의미를 설명하는 **동사구**로 표현
예) employs, works, joins 등



: 자동차는 자동차보험에 가입되어 있으므로,
자동차보험 객체를 **영구적으로 참조하기 위해 필드에 보관**한다.

다중성(multiplicity)
: 얼마나 많은 클래스 A 인스턴스들이
클래스 B 인스턴스 하나와 연관될 수 있는가를 정의
: 어떤 특정 시점에 얼마나 많은 인스턴스가
다른 인스턴스와 연관될 수 있는가를 보여준다.
예) 한 자동차는 **항상 한 개의 자동차보험**에 가입한다.
예) 한 자동차보험은 **많은 자동차**의 보험을 보유한다.

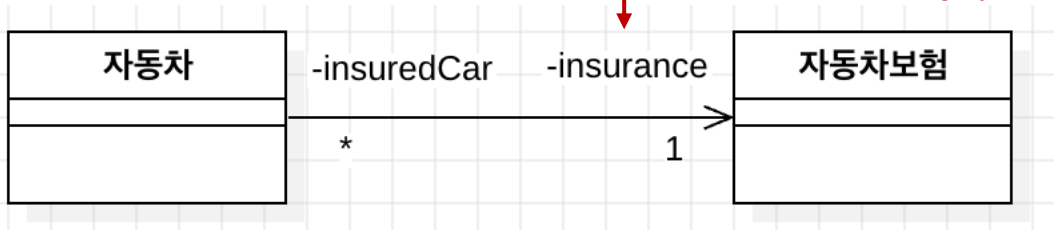
예)	
0..1	0 또는 1
1	1
0..*	0 이상
*	0 이상
1..*	1 이상
1..6	1에서 6까지

UML 구조 > Building blocks > Relationships

Association

객체 사이의 연결관계로, **지속적으로 유지되는 관계**다.
예) "자동차는 자동차보험에 가입한다."

역할 이름
: 연관에 참여한 클래스의 역할.
: 연관 관계의 양 끝에 붙인다. 한쪽 또는 양쪽 모두 붙일 수 있다.
: **명사**나 **명사구**로 표현한다.

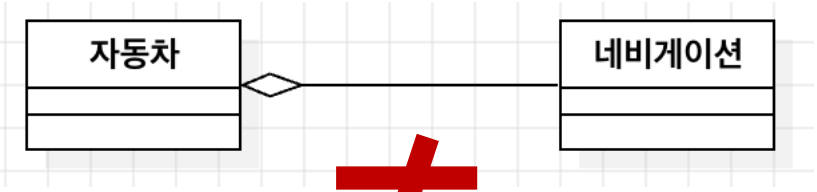


연관 이름이나 **역할 이름** 둘 다 붙일 수 있지만,
둘 중 하나만 붙이는 것이 더 낫다.

UML 구조 > Building blocks > Relationships

Aggregation

객체 간의 **전체-부분 관계**로, "**has-a 관계**"의 일종이다.
한 객체가 다른 객체를 포함하지만, **포함된 객체가 독립적으로 존재**하는 관계.
예) "자동차는 네비게이션을 가진다. 네비게이션은 자동차가 없어도 존재할 수 있다."



≠

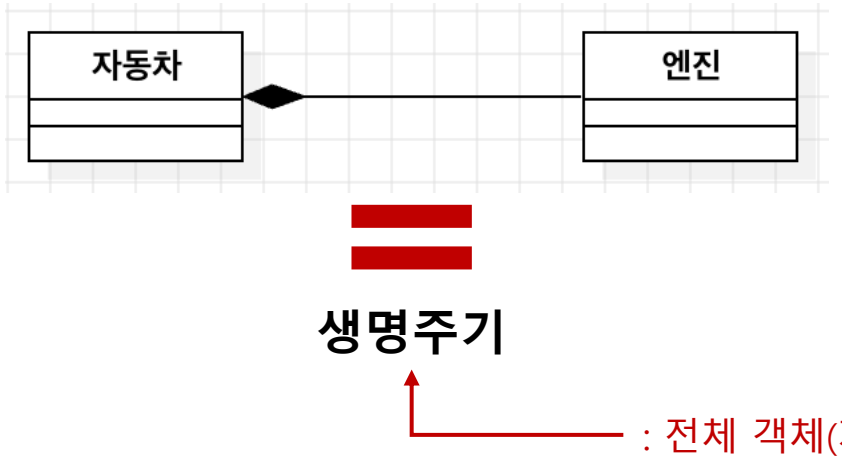
생명주기

: 전체 객체(자동차)가 사라져도 부분 객체(네비게이션)는 남는다.

UML 구조 > Building blocks > Relationships

Composition

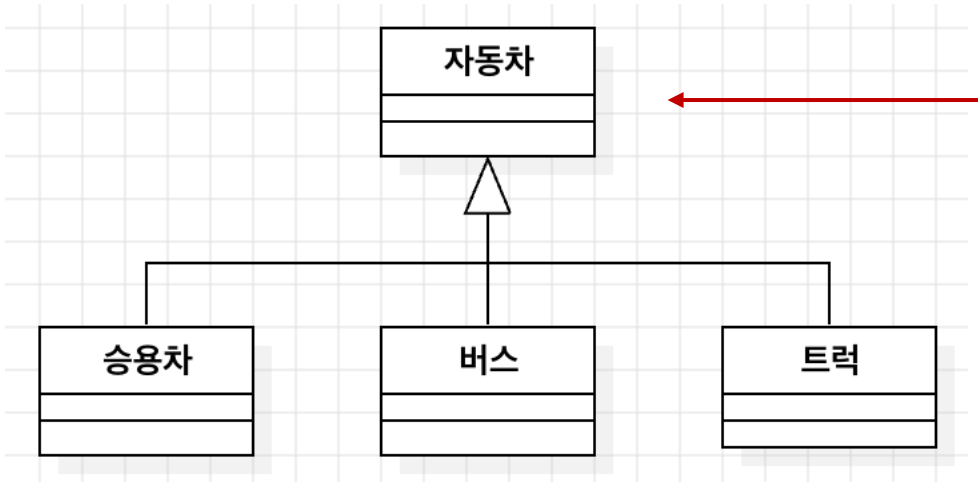
객체 간의 **전체-부분 관계**로, "**has-a 관계**"의 일종이다.
영구적이며 강한 연관 관계로 구성되며, **포함된 객체가 전체 객체에 종속**되는 관계.
예) "자동차는 엔진을 가진다. 자동차가 없으면 엔진도 존재하지 않는다."



UML 구조 > Building blocks > Relationships

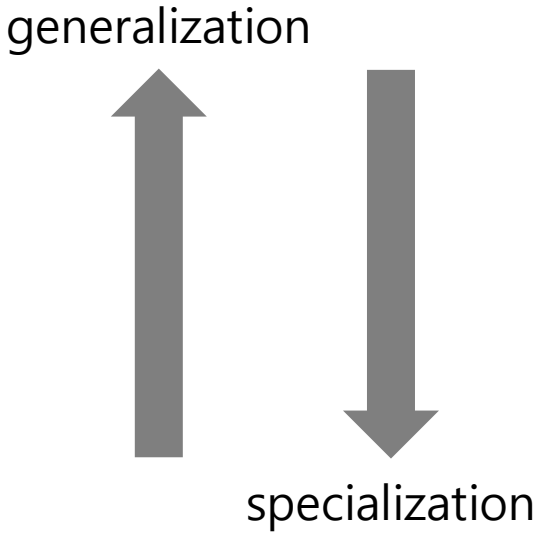
Generalization

일반화된 사물과 좀 더 특수화된 사물 사이의 관계로, **kind-of 관계**다.
예) "승용차는 자동차의 한 종류다 → A Sedan is a **kind of** Car"



수퍼 클래스(super/parent)
: 좀더 일반적인 개념

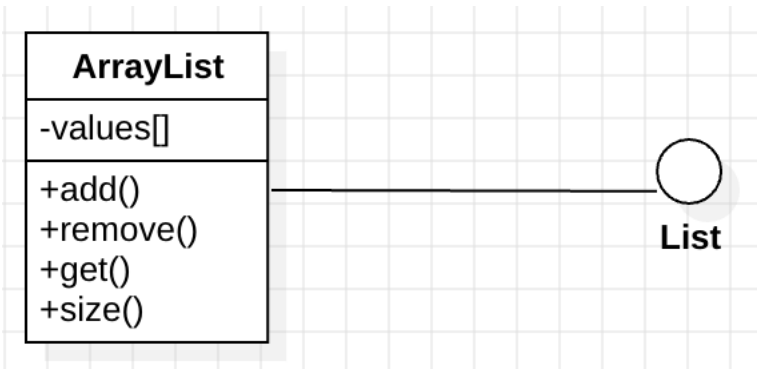
서브 클래스(sub/child)
: 좀더 상세화 된 개념



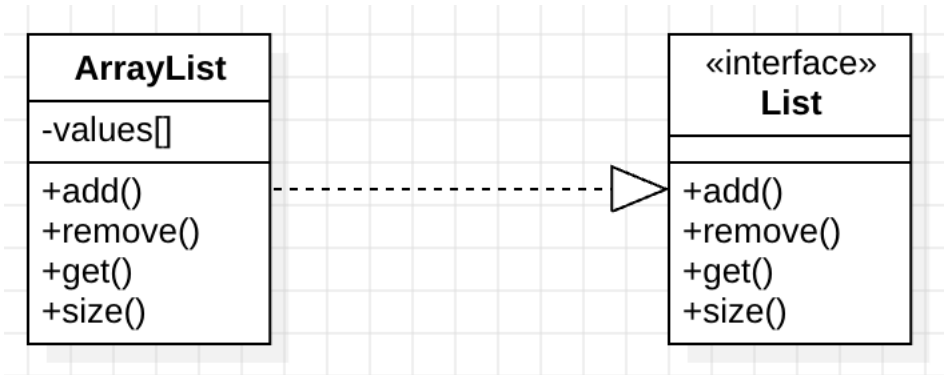
UML 구조 > Building blocks > Relationships

Realization

한 요소가 다른 요소에 정의된 행동(contract)을 구현(fulfill)하는 관계다.
즉 계약을 명세하는 클래스와 그 계약을 실현하는 클래스 사이의 관계다.
예) "ArrayList는 List 계약을 구현한다."



또는



UML 구조 > Building blocks > Diagrams

: 요소들과 사물 간의 관계를 그래프로 나타낸 것.

다이어그램

- **정적 다이어그램**
  - **클래스 다이어그램(Class)**
  - 객체 다이어그램(Object)
  - **컴포넌트 다이어그램(Component)**
  - **배치 다이어그램(Deployment)**
  - **패키지 다이어그램(Package)**
  - 복합 구조 다이어그램(Composite Structure)
- **동적 다이어그램**
  - **유스케이스 다이어그램(Use Case)**
  - **활동 다이어그램(Activity)**
  - **상태 머신 다이어그램(State Machine)**
  - **순차 다이어그램(Sequence)**
  - **통신 다이어그램(Communication)**

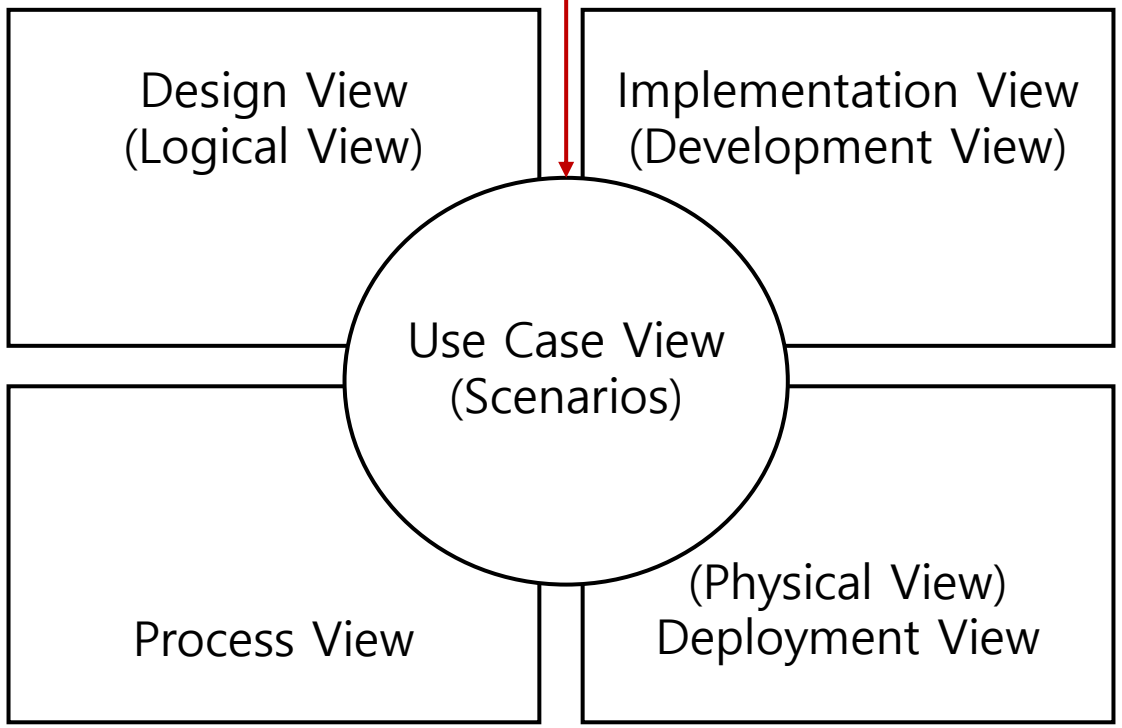
UML 구조 > Architecture > UML 뷰

: 아키텍처는 **시스템의 조직적 구조와 설계 원칙**을 의미한다
: UML은 "**4 + 1 뷰**" 아키텍처를 통해 **다양한 이해관계자의 관점에서 시스템 구조를 시각화**
: 사용자, 개발자, 운영자 모두의 관점에서 시스템을 이해할 수 있음

고객, 사용자, 분석가
: 시스템의 기능적 요구사항과 시나리오 정의

설계자, 개발자
: 시스템의 기능적 요구사항을 중심으로, 주요 클래스의 관계를 보여줌

개발자
: 소스코드구조 및 컴포넌트 구성, 패키지 의존성 중심



시스템 아키텍트
: 동시성, 프로세스 간 상호작용, 스레드 동작 등을 표현

시스템 엔지니어
: 시스템 배포환경 – 하드웨어 노드, 서버, 네트워크 등을 표현

UML 구조 > Architecture > UML 뷰 > 유스케이스 뷰

: **기능적 요구사항 표현** - 사용자 관점에서 시스템의 기능적 요구사항과 시나리오를 정의

주요 이해 관계자

- 고객, 사용자, 분석가

목적

- 사용자가 시스템으로부터 원하는 기능이 '무엇'인지를 정의한다.
- 시스템이 무엇을 해야 하는가? (기능 요구 사항)
- 시스템이 어떤 기능을 제공해야 하는가?

다이어그램

- 정적 측면: **Use Case Diagram**
- 동적 측면: Sequence/Communication Diagram

UML 구조 > Architecture > UML 뷰 > 설계 뷰(논리 뷰)

: **시스템의 구조 표현** - 시스템의 기능적 요구사항을 중심으로, 주요 클래스와 관계를 보여줌

주요 이해 관계자

- 설계자, 개발자

목적

- 유스케이스 뷰에서 정의된 기능을 시스템이 제공하기 위해 어떤 클래스와 어떤 컴포넌트가 필요하고, 이들이 서로 어떻게 이용하고 호출하는지 파악해 기술한다.
- 기능을 어떻게 구조화 할 것인가?
- 어떤 클래스/객체가 어떤 기능을 담당하는가?

다이어그램

- 정적 측면: **Class Diagram**, Object Diagram, Package Diagram
- 동적 측면: **State Machine Diagram**, **Sequence/Communication Diagram**

UML 구조 > Architecture > UML 뷰 > 구현 뷰(개발 뷰)

: **코드/모듈 구성** - 소스 코드 구조 및 컴포넌트 구성, 패키지 의존성 중심

주요 이해 관계자

- 개발자

목적

- 시스템의 구현 상태를 나타내기 위해 컴포넌트와 같은 구현 모듈과 그들 사이의 관계 또는 각종 파일과 파일 간의 의존 관계 등을 보여준다.
- 소스 코드와 모듈 구조는 어떻게 구성되는가?
- 코드는 어떻게 조직되어 있는가?

다이어그램

- 정적 측면: **Component Diagram**, Package Diagram

UML 구조 > Architecture > UML 뷰 > 프로세스 뷰

: **동적 동작 표현** - 동시성, 프로세스 간 상호작용, 스레드 동작 등을 표현

주요 이해 관계자

- 시스템 아키텍트

목적

- 프로세스와 스레드를 나타내며, 이들의 책임과 이들이 어떻게 협력하는지를 기술한다. 서브 시스템, 클래스 등의 논리적 요소가 어떻게 프로세스/스레드에 할당되는지를 보여준다.
- 실행 시점에서 병렬성과 동기화는 어떻게 되는가?
- 시스템은 동시에 어떻게 동작하는가?

다이어그램

- 정적 측면: **Class Diagram**
- 동적 측면: **Sequence/Communication Diagram**

UML 구조 > Architecture > UML 뷰 > 배치 뷰(물리 뷰)

: **시스템의 배포 환경** – 하드웨어 노드, 서버, 네트워크 등을 표현

주요 이해 관계자

- 시스템 엔지니어

목적

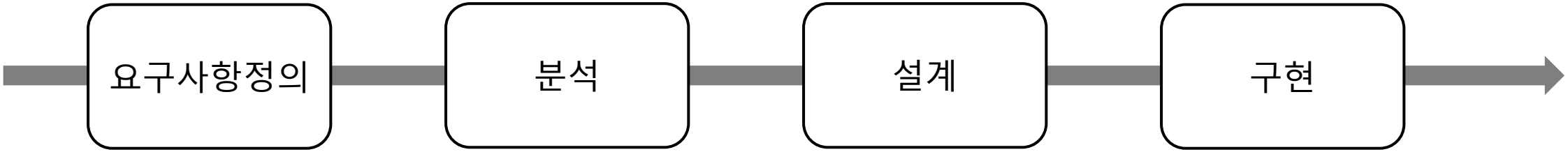
- 시스템을 구성하는 처리 장치, 즉 컴퓨터와 컴퓨터 간의 통신 방법에 중점을 둔다.
- 시스템은 어떤 하드웨어/네트워크 환경에 배포되는가?
- 시스템은 어디서, 어떻게 실행되는가?

다이어그램

- 정적 측면: **Deployment Diagram**

UML 구조 > Architecture > UML 뷰 + 개발 활동

: 개발 절차에 따라 중점을 두는 뷰가 다르다.



- **유스케이스 뷰**
- **설계 뷰**

- **설계 뷰**
- 프로세스 뷰
- 구현 뷰
- 배치 뷰

- 구현 뷰
- 배치 뷰

간단한 시스템 개발
: 유스케이스 뷰와 설계 뷰면 충분하다

UML 구조 > Common mechanisms > Specifications

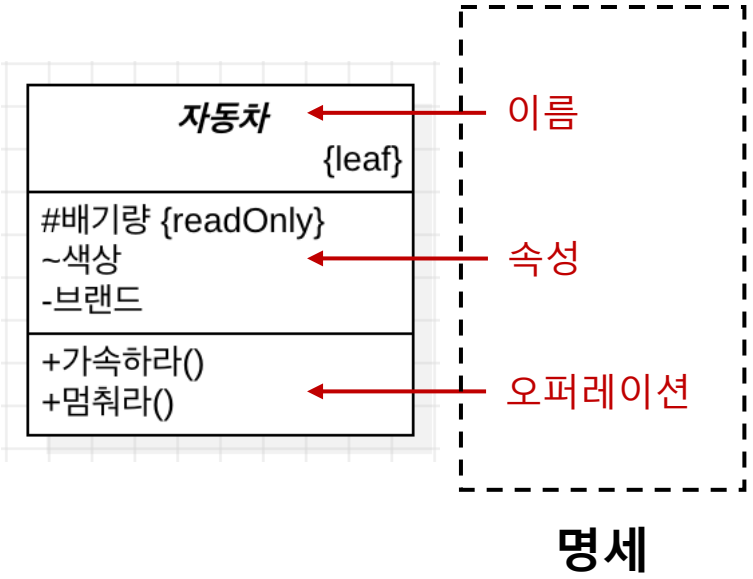
: UML의 **그림은 의미를 보여주는 표현**에 불과하고, **진짜 모델은 그 밑에 있는 명세**다.

그래픽 차원(Graphical dimension)
다이어그램, 아이콘으로 시각화

표현

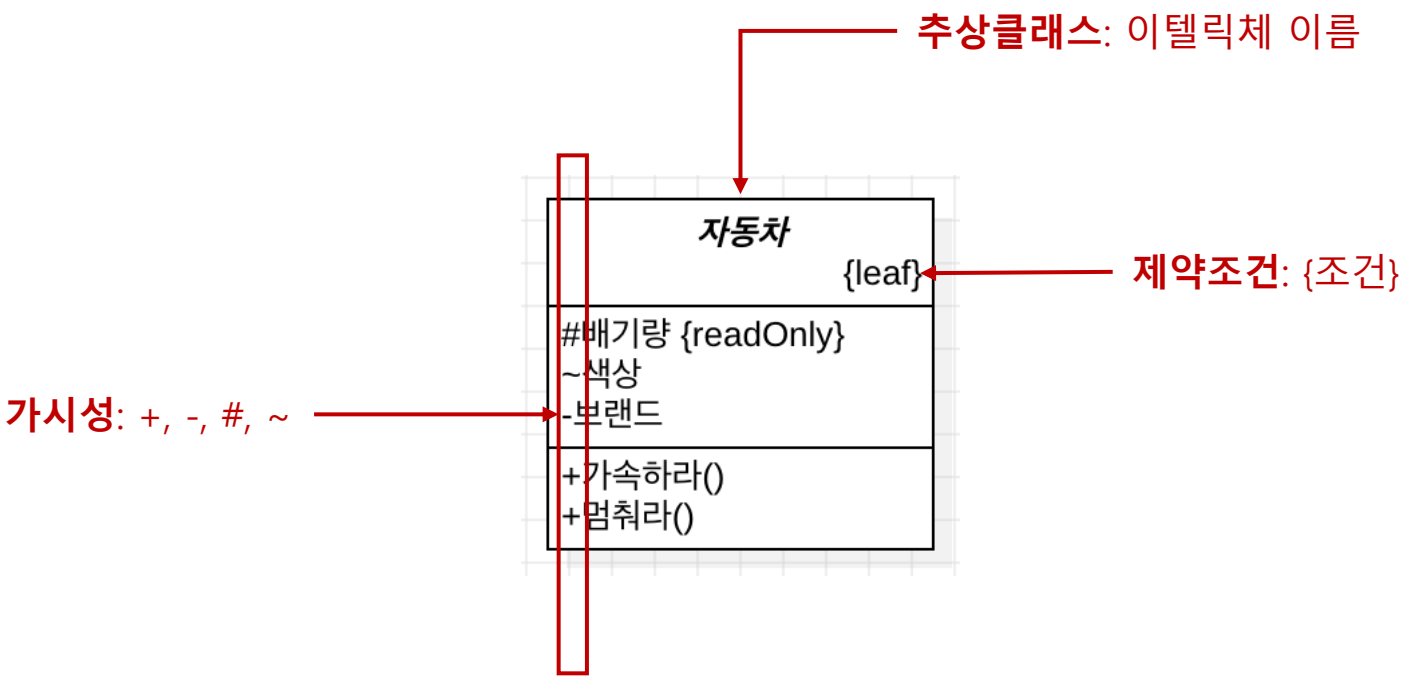
텍스트 차원(Textual dimension)
모델 요소의 의미를 기술하는 명세로 표현

UML 모델



UML 구조 > Common mechanisms > Adornments

: UML에서는 요소의 **추가 정보나 세부 속성을 기호나 부호로 표시**한다. 이를 '**장식**'이라 한다.

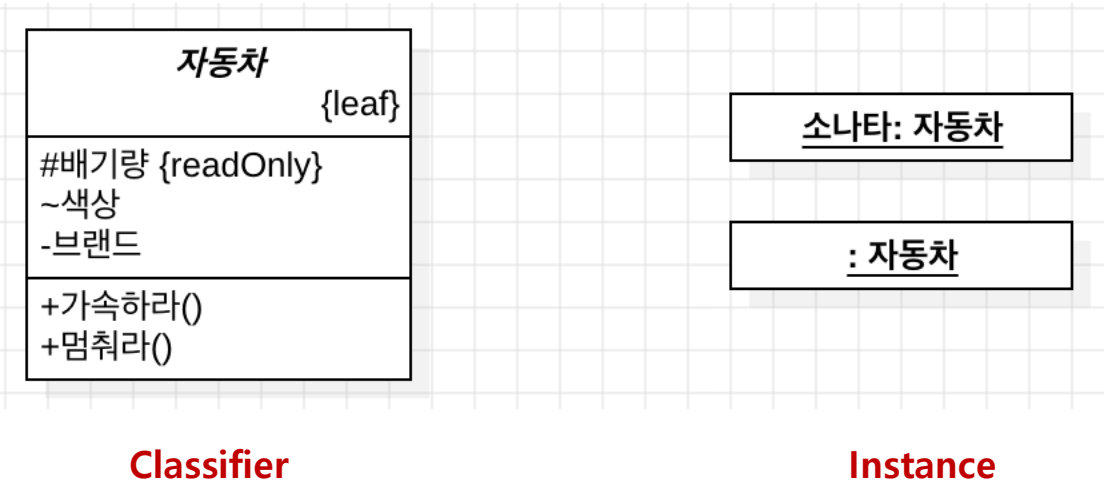


UML 구조 > Common mechanisms > Common Divisions

: UML은 각 모델 요소를 **두 가지 관점**으로 나누어 볼 수 있는 **공통 분할**의 개념이 있다.

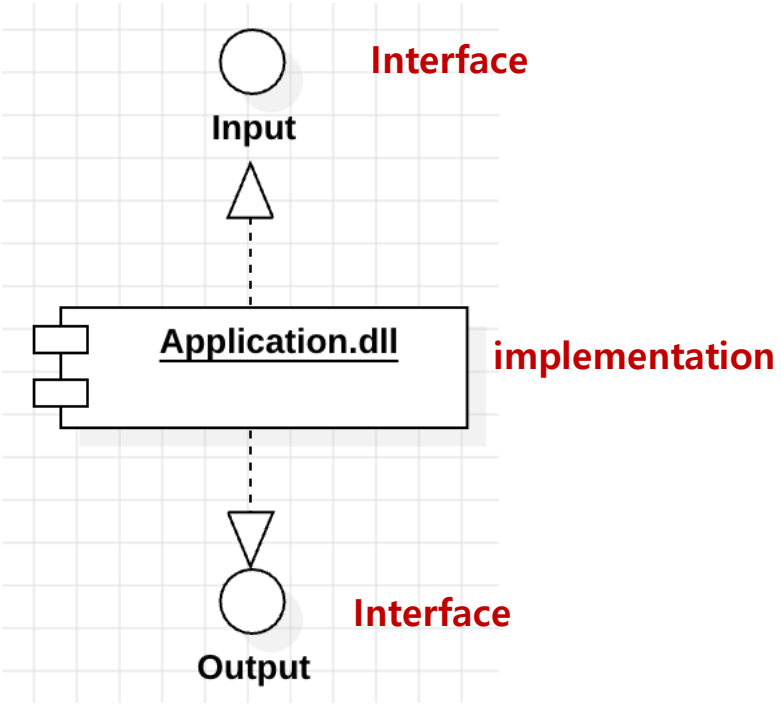
Classifier/Instance

- 클래스(classifier)는 사물의 **추상 개념**이고,
  객체(instance)는 추상 개념을 **구체적으로 명시**한 것.
- 예) 유스케이스/유스케이스 인스턴스,
  컴포넌트/컴포넌트 인스턴스, 노드/노드 인스턴스



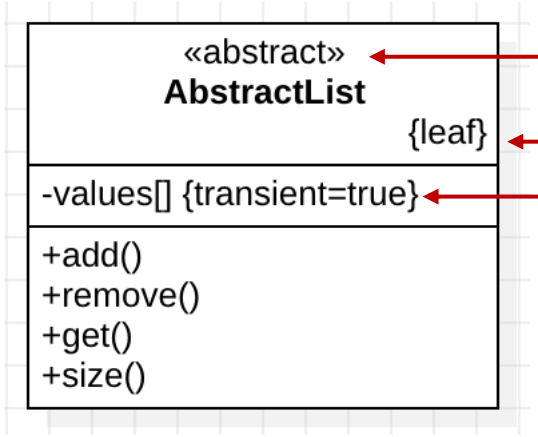
Interface/Implementation

- 무엇을 하는지(**계약**)와 어떻게 하는지(**구현**)을 분리하는 것.
- 예) 오퍼레이션/이를 구현하는 메서드
  유스케이스/이를 실현하는 통신



UML 구조 > Common mechanisms > Extensibility mechanisms

: UML은 각 모델 요소를 **두 가지 관점**으로 나누어 볼 수 있는 **공통 분할**의 개념이 있다.



Stereotype

- 모델 요소의 **새로운 의미나 역할**을 정의
- 예) «interface», «controller», «entity»

Constraints

- 중괄호 {} 안의 테스트 문자열로, 모델 요소의 **새 규칙이나 조건**을 추가
- OCL(Object Constraints Language) 등을 이용해 논리 제약조건 표현
- 예) {size > 0}

Tagged Values

- 모델 요소에 **새로운 속성**을 추가
- 문법은 {tag1=value, tag2=value}
- 예) {transient=true}

Use Case

Use Case > 개요

: Use Case 모델을 시각화하는 기본 메커니즘이다.

Use Case 모델링

- 시스템의 요구사항을 도출하고 문서화하는 **활동** 또는 **프로세스**
- **다이어그램을 작성**하고 **관계를 설정**하고 **사물을 정의**하는 과정

활동: 시스템 경계 찾기, 액터 찾기, 유스케이스 찾기

Use Case 모델

- 유스케이스 모델링 활동의 **산출물** 또는 **아티팩트**
- 시스템 요구사항을 캡쳐하는 **소프트웨어 요구사항 명세(SRS)**의 일부
- **객체와 클래스의 주요 출처**가 되며, **클래스 모델링의 입력으로 사용**됨

핵심 요소: 시스템 경계, 액터, 유스케이스, 관계

Use Case Diagram

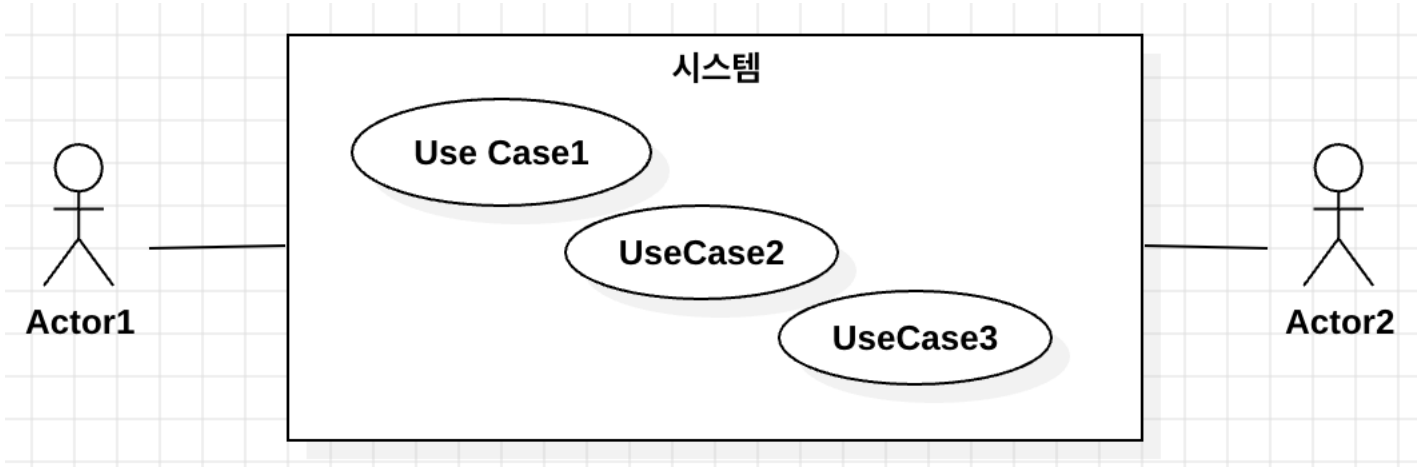
- 유스케이스 모델의 **뷰** 또는 **창**이며, **모델을 시각화**하는 기본 메커니즘
- 시각적 요소 통해 **시스템의 범위를 정의하는데 사용**됨

시각적 요소: 시스템 경계, 액터, 유스케이스, 상호작용

Use Case > 목적

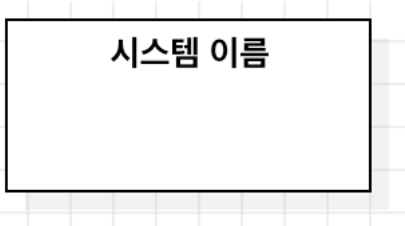
: 시스템의 요구사항을 담은 **명세의 내용을 이해하기 쉽도록 시각적으로 표현**하는 청사진의 역할

- **누가 시스템을 사용할 것인가?**
- **시스템은 사용자를 위해 무엇을 해야 하는가?**
- **사용자와 상호작용하기 위해 시스템이 제공해야 할 인터페이스는 무엇인가?**



Use Case > 구성 요소

: 유스케이스 다이어그램의 시각적 요소는 시스템 경계, 액터, 유스케이스, 상호작용이 있다.

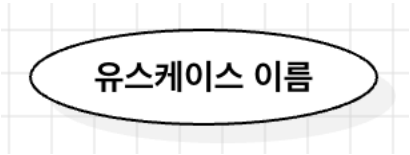


System boundary

모델링 하는 시스템의 경계를 나타내기 위해 유스케이스 주변에 그리는 상자다.
UML 2에서는 **주제(subject)**라고 부른다. 주제는 **시스템을 사용하는 주체(액터)**와
시스템이 그들에게 제공하는 이점(유스케이스)에 의해 정의된다.
시스템의 일부인 것(경계 내부)과 시스템 외부의 것(경계 외부)을 분리하여, **시스템의 범위를 정의**한다.

Actor

시스템과 **직접 상호 작용**할 때 맡는 **역할**을 명시한다.
항상 시스템 외부에 배치된다. 시스템으로부터 **어떤 가치를 제공받는 외부 사용자**를 나타낸다.
사용자 역할, 다른 시스템, 하드웨어 등 **시스템의 경계에 닿는 모든 외부 엔티티**를 대변한다.

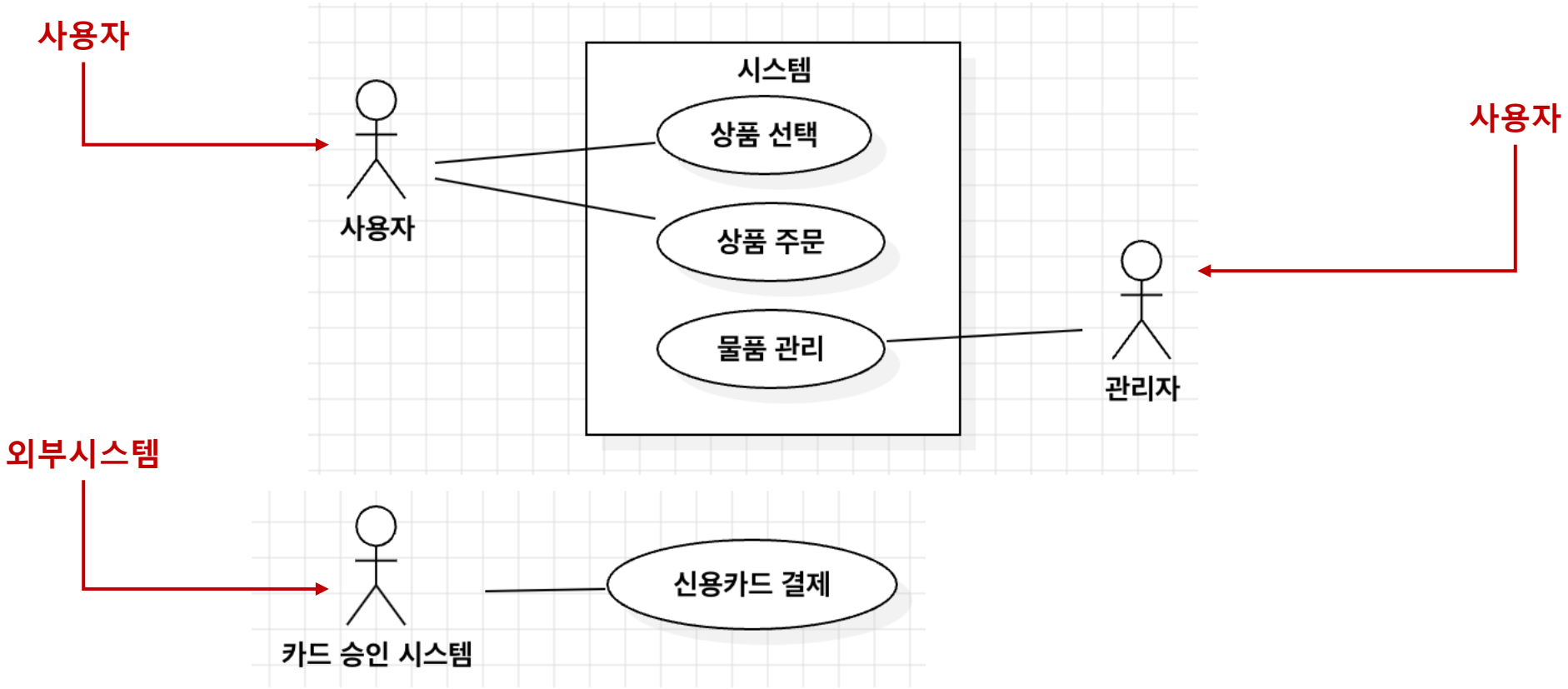


Use case

시스템이 하나 이상의 **액터에게 이익을 제공하기 위해 수행하는 행동을 설명**하는 요소다.
시스템 내부에 배치되며, 시스템의 행동을 구성한다.
항상 액터에 의해 시작되어야 하며, 시스템이 액터를 대신해 수행하는 기능을 나타낸다.

Use Case > 관계 > 액터와 유스케이스의 관계

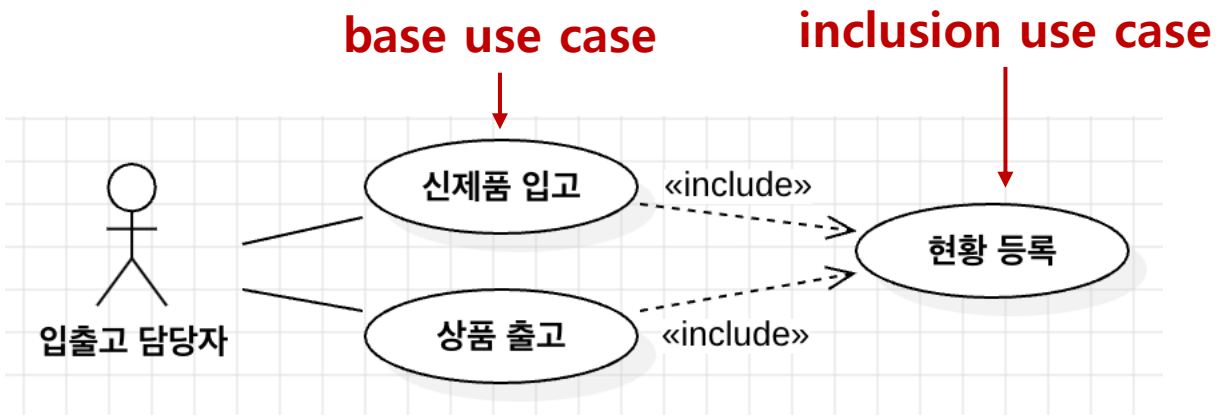
: 액터와 유스케이스 사이에는 **연관(Association) 관계**가 존재한다.
: 액터는 사용자 또는 외부 시스템이 될 수 있다.



Use Case > 관계 > 유스케이스 포함 관계(필수)

: 하나의 유스케이스가 다른 유스케이스의 행위를 **포함하는 관계**

«include»

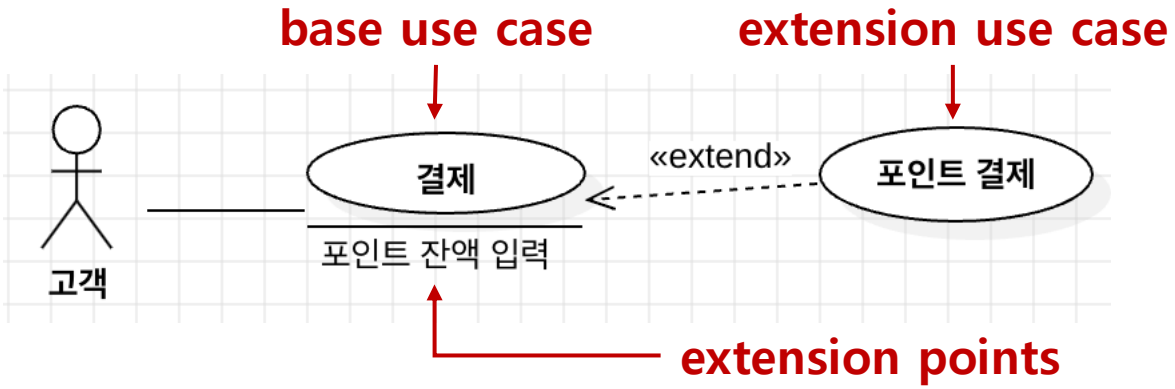


- 하나의 유스케이스를 수행할 때, **다른 유스케이스가 반드시 수행**되는 관계
- 여러 유스케이스 흐름에 반복되는 단계를 **별도의 유스케이스로 분리**하여 필요할 때 마다 **재사용**
- «include» 의 과도한 사용은 유스케이스 모델을 이해하기 어렵게 만든다.
- 유스케이스 모델을 기능 분해 방식으로 변질되는 것을 경계해야 한다.

Use Case > 관계 > 유스케이스 확장 관계(선택)

: 기존 유스케이스에 **새로운 행위를 삽입하여 확장**하는 관계

«extend»

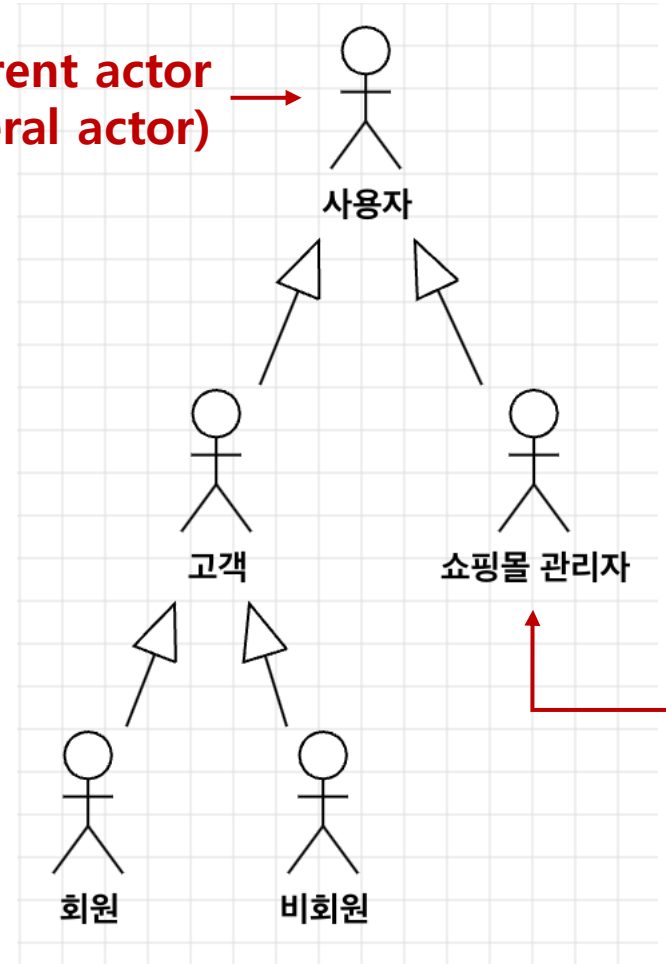


- 기존 유스케이스의 행위를 확장할 수 있도록 한다.
- **기반 유스케이스** - 확장이 삽입되는 유스케이스이다. 확장 유스케이스가 없어도 온전히 완벽하다
- **확장 유스케이스** - 기반 유스케이스에 행위를 추가하는 역할. 완전한 유스케이스가 아니다.
- **확장 포인트** - 기반 유스케이스의 이벤트 흐름이 수행되다가 확장 포인트를 만나면 확장 유스케이스의 이벤트 흐름으로 분기된다.
- 포함 관계와 달리 **조건이 만족되는 경우에만** 확장 유스케이스의 **이벤트 흐름으로 분기**된다.

Use Case > 관계 > 액터의 일반화 관계

: 두 개 이상의 액터에서 **공통적인 행동을 추출하여 부모 액터를 정의**하는 것

Generalization

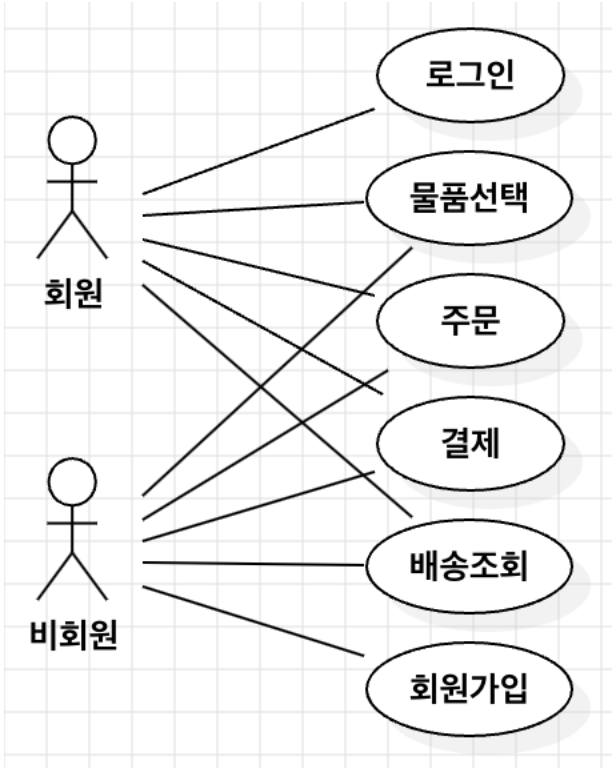


- **공통 행동 추출** – 여러 액터가 동일한 유스케이스와 상호작용하는 경우, 이 공통적인 역할을 상위 액터에게 할당함으로써, **유스케이스 다이어그램을 단순화**하고 명확성을 높인다.
- **상속** – 자식 액터는 부모 액터의 모든 역할과 유스케이스와의 관계를 상속한다.
- **대체 가능성 원칙** – 자식 액터는 부모 액터의 역할에 대신 들어갈 수 있다.
- **추상 액터와 구체적 액터** – 공통 행동을 추출하기 위해 도입된 부모 액터는 "추상 액터"라 부른다. 반면, 실제 사람이나 시스템의 역할을 수행하는 액터는 "구체적 액터"라 부른다.

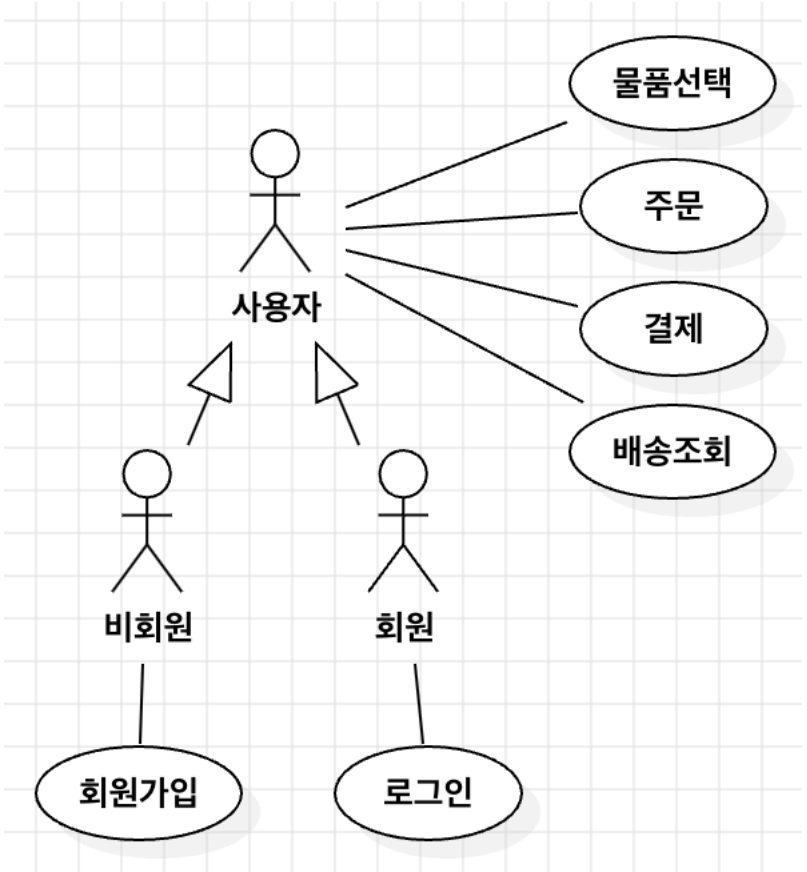
Use Case > 관계 > 액터의 일반화 관계 전과 후

: 공통적인 역할을 상위 액터에게 할당하면, **유스케이스 다이어그램이 더 간결하고 명확**해진다.

일반화 전



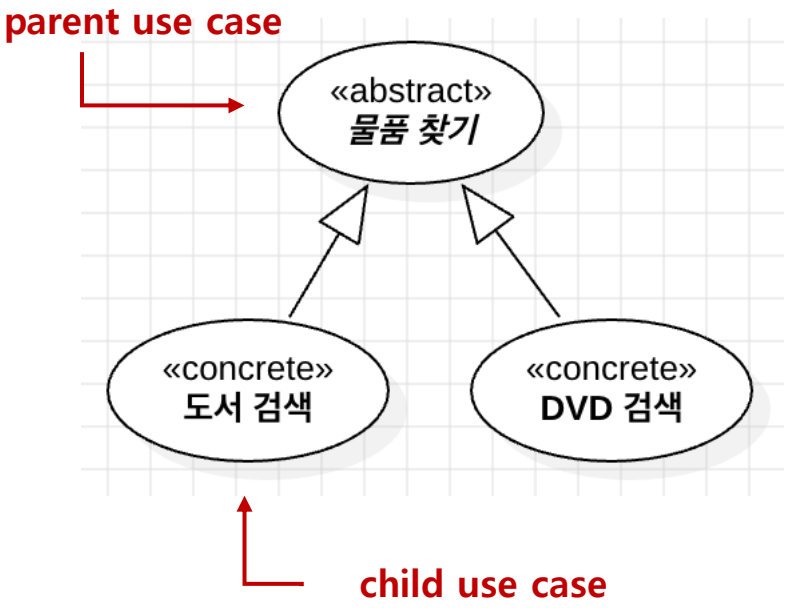
일반화 후



Use Case > 관계 > 유스케이스의 일반화 관계

: 둘 이상의 유스케이스에서 **공통적인 행위나 특징을 추출**하여 **부모 유스케이스를 정의**

Generalization



- **공통 행동 추출** – 여러 유스케이스에 공통적인 행위나 특징이 있을 경우, 이를 부모 유스케이스로 묶어 별도로 분리한다.
- **특수화 표현** – 자식 유스케이스는 부모 유스케이스의 더욱 구체화된 형태
- **모델 단순화** – 유스케이스 모델을 단순화하고 명확하게 만들 때 사용해야 한다.
- **자식 유스케이스** – 부모 유스케이스의 모든 기능을 자동으로 상속 받는다.
- **추상 유스케이스** – 부모 유스케이스는 종종 추상적인 상태로 모델링한다. 흐름이 누락되거나 불완전하므로 시스템에 의해 실행될 수 없다.

Top-down 방식으로 기능을 쪼개는 절차적 프로그래밍 기법으로
객체지향 시스템과의 개념적으로 불일치 발생!

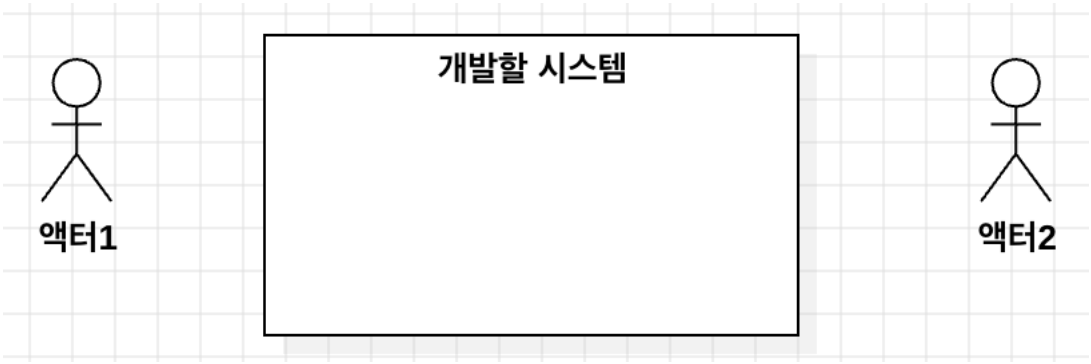
주의!

유스케이스 모델링의 고급 기능(일반화, include, extend)을 과도하게 사용하면 오히려 모델이 이해하기 어려워지거나, **기능 분해 방식으로 변질**될 수 있다. 즉, 요구사항을 포착하는 데 초점을 맞추기 보다, 요구사항을 구조화하는 데 초점을 맞추는 상황에 빠질 수 있다.

Use Case > 모델링 > 액터 식별

액터란?

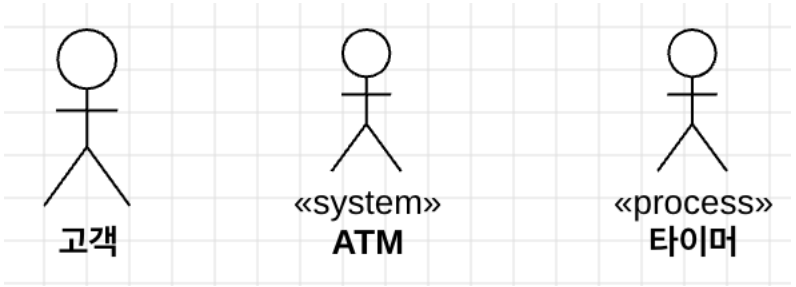
- **시스템과 상호작용하는 외부 엔티티**로서, 시스템에 **요청을 보내거나 서비스를 받는** 존재
- 시스템 밖에서 시스템을 사용하는 주체로, **시스템의 경계를 정의하는 역할**



Use Case > 모델링 > 액터 식별

주요 액터(primary actor)

- 시스템을 직접 사용하거나 기능을 요청하는 주체
- 시스템을 사용하는 사람 또는 외부 시스템
- 유스케이스를 시작시킨다.
- 왜 식별하는가?
  유스케이스를 수행시키는 사용자의 목적을 알기 위함.

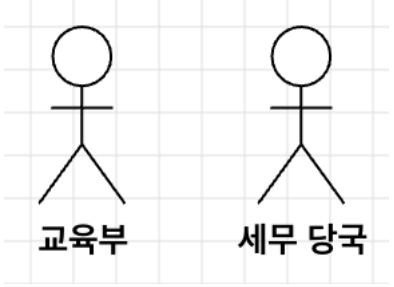


지원 액터(supporting actor)

- 시스템에 서비스를 제공하는 외부 주체
- 유스케이스 수행 과정에서 시스템을 지원하는 역할
- 왜 식별하는가?
  외부 인터페이스와 프로토콜을 명확히 하기 위함.

숨겨진 액터(offstage actor)

- 유스케이스에 직접 참여하지 않지만,
  시스템의 결과나 상태에 이해관계를 가지는 외부 주체
- 즉 시스템에 영향을 받거나 이해관계가 있는 사람/조직
- 왜 식별하는가?
  모든 필수적인 이해 관계를 식별하고 만족시키기 위함.



Use Case > 모델링 > 액터 식별

액터 식별에 사용되는 질문

- 시스템을 사용하는 주체는 누구인가? 또는 무엇인가? 그들의 역할 무엇인가?
- 시스템 설치는 누가 하는가?
- 시스템을 시작하고 종료하는 주체는?
- 시스템은 누가 유지보수 하는가?
- 시스템과 상호 작용하는 다른 시스템은 무엇인가?
- 시스템에 정보를 제공하거나 시스템으로부터 정보를 얻는 주체는 누구 또는 무엇인가?
- 정해진 시간에 발생하는 일이 있는가?

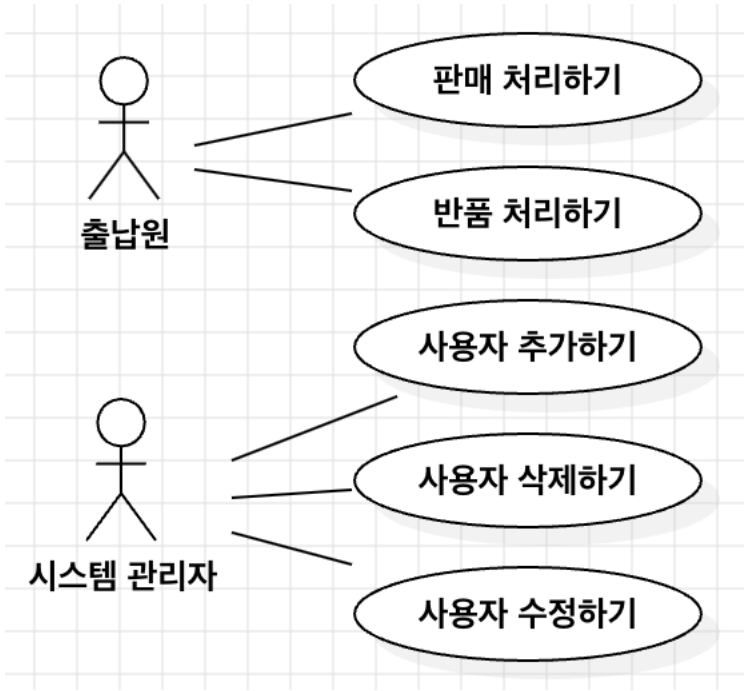
간과할 수 있는 액터

- 누가 사용자와 보안을 관리하는가?
- 누가 시스템을 관리하는가?
- 시스템 실패 시 재시작 하는 모니터링 프로세스는 있는가?
- 누가 시스템 활동과 성능을 평가하는가?
- 누가 로그를 평가하는가? 원격으로 검색되는가?
- 소프트웨어 업데이트는 어떻게 수행되는가? 내려받기(pull) 방식인가? 밀어넣기(push) 방식인가?
- 시스템 오류가 발생하거나 동작에 실패할 경우 누구에서 통보되는가?

Use Case > 모델링 > 유스케이스 식별 및 정의

유스케이스 식별 지침

- 액터는 시스템을 사용하여 무엇을 하고자 하는가?
- 액터의 목적는 무엇인가?
- 해당 목적을 통해 어떤 가치를 달성하는가?
    - 상위 목적을 알기 위한 질문이다.
    - 기존의 방식을 개선한 새 방안을 식별할 수 있다.



유스케이스 정의

- 하나의 목적을 하나의 유스케이스로 정의하라.
- 유스케이스 이름은 액터의 목적과 비슷하게 부여한다.
- 유스케이스 이름은 동사로 시작하라. (영어인 경우)
    - "**액터가 시스템을 통해 달성하려는 목표**(Goal)"를 짧고 명확하게 표현한 문장으로 작성하라.

Use Case > 모델링 > 유스케이스 이름 지침

액션 중심으로 표현

시스템의 기능이나 서비스는 행동(동사)으로 표현

예) "회원가입하기(SignUp)", "로그인하기(Login)"

일관된 어조와 형태 유지

모든 유스케이스 이름의 문법/형태를 동일하게 유지

"~하기" (한글), "Verb + Object" (영문)

액터의 관점에서 작성

시스템이 하는 일이 아니라, 사용자가 달성하려는 목적 중심

"상품주문(OrderProduct)" (O)

"상품처리하기(ProcessOrder)" (X)

결과 중심으로 작성

과정(process) 보다는 **결과(goal)**를 표현

"주문완료하기(CompleteOrder)" (O),

"상품목록보기(ViewProducts)" (O)

간결하고 구체적으로

한 줄로 요약되게, 모호한 단어 피하기

"비밀번호재설정(ResetPassword)" (O)

"계정관리(AccountManagement)" (X)

시스템 내부 동작은 안됨!

시스템 내부 로직이 아니라, 외부 사용자의 행동 단위다.

"DB저장하기" (X),

"API호출하기" (X)

Use Case > 모델링 > 유스케이스 유효성 테스트 I

보스(boss) 테스트

보스: "**당신은 오늘 하루 동안 어떤 일을 했습니까?**"

답변: "로그인 했습니다."

보스: ? (업무가 아니거나 중요하지 않은 일이다.)

통과 기준:

- 즉, **비즈니스 가치가 명확하게 설명**되어야 한다.
- "왜 이 기능이 필요한가?"라는 **질문에 경영자나 고객도 이해**할 수 있는 답이어야 한다.
- 시스템 내부 절차(DB 저장, 로그 기록 등)는 가치가 없음

예시:

- "DB에 사용자 정보 저장하기" → 내부 구현 행위 (X)
- "고객이 상품 주문을 완료한다" → 고객 가치 명확 (O)

크기 테스트

유스케이스가 너무 크거나, 너무 작아서 관리하기 어려운지 확인하는 테스트

너무 작으면 → 단순 기능 수준 (버튼 클릭, 필드 입력 등)

너무 크면 → 여러 비즈니스 목표가 섞여 있음

통과 기준:

- 주 시나리오가 5 ~ 9단계 내외
- 하나의 비즈니스 목표를 표현
- **한 액터의 단일 목적 단위로 기술** 가능

예시:

- "로그인 버튼 클릭하기" → 너무 작음 (X)
- "고객 관리하기" → 회원가입, 수정 등 여러 목표 혼합 (X)
- "회원 가입하기" → 하나의 목표, 명확한 종료 조건 (O)

Use Case > 모델링 > 유스케이스 유효성 테스트 II

EBP 테스트

기본 비즈니스 프로세스(EBP; elementary business process)란?

- 한 번에 한 장소에서 한 사람에 의해 수행되는 업무
- 측정 가능한 업무 가치를 내고
- 일관된 상태로 데이터를 남기는 비즈니스 업무

관측할 수 있고 **측정**할 수 있는 **비즈니스 가치**인가?

즉, 사용자가 **가시적 결과**를 얻는 **완전한 단위의 업무 절차**인가?

하나의 유스케이스는 **하나의 완결된 업무 목표**를 달성해야 한다.

통과기준:

- 한 액터가 **단일 세션에서 완료**할 수 있어야 함.
- 유스케이스가 끝나면 **외부적으로 확인 가능한 상태 변화**가 있어야 함.
- 결과가 "부분 작업"이 아니라 "**완전한 결과**" 일 것.

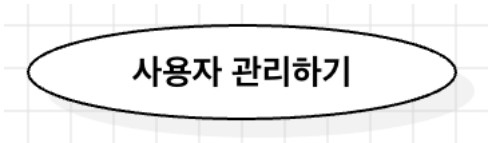
예시:

- "주문하기" (O)
- "상품 선택하기" → 중간 단계, 결과 없음 (X)
- "비밀번호 재설정하기" (O)

Use Case > 모델링 > 유스케이스 실무 리팩토링

CRUD 유스케이스

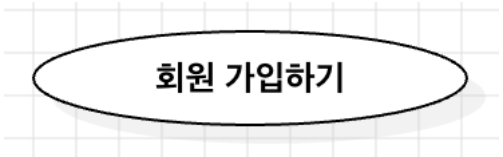
관리의 용이성을 위해 CRUD(생성, 조회, 갱신, 삭제)로 분리된 목적들을 하나의 유스케이스로 합쳐서 “ManageXxx” 로 정의할 수 있다.



- 사용자 추가하기
- 사용자 조회하기
- 사용자 상세보기
- 사용자 수정하기
- 사용자 삭제하기

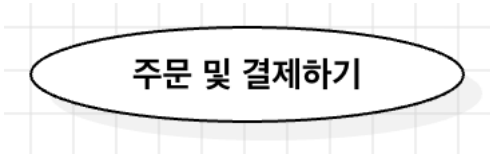
유스케이스 통합

효율성을 위해 서로 유사하거나 밀접하게 연관된 유스케이스들을 하나의 유스케이스로 통합할 수 있다.



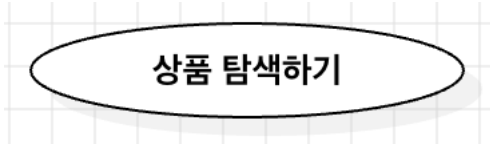
시나리오가 거의 동일할 때

- 이메일 회원 가입하기
- SNS 회원 가입하기



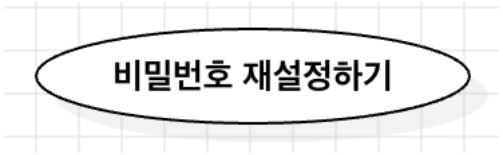
다른 유스케이스를 항상 포함할 때

- 주문하기
- 결제하기



액터가 같고 목표가 유사할 때

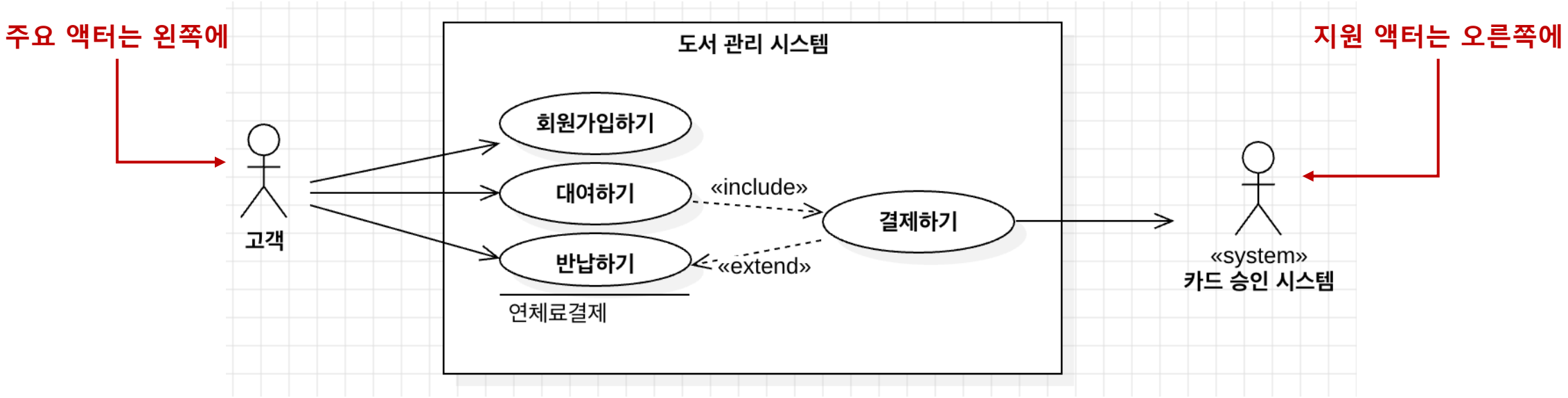
- 상품 조회하기
- 상품 상세보기



단일 결과를 만들 때

- 비밀번호 초기화하기
- 비밀번호 찾기

Use Case > 모델링 > 유스케이스 다이어그램



다이어그램에 많은 노력을 들이지 말라!

유스케이스 모델링 작업에서 중요한 것은 다이어그램을 그리거나 유스케이스 관계를 식별하는 것이 아니라, **텍스트를 작성하는 것**이다. 즉 **유스케이스 명세서**를 작성하는 것이다. 만약 텍스트를 작성하는 것보다 유스케이스 다이어그램을 작성하고 유스케이스 관계를 얘기하는 데 많은 시간을 들인다면, 잘못하고 있는 것이다.

Use Case > 모델링 > 유스케이스 명세서

약식(brief)

보통 주요 성공 시나리오를 한 문단으로 간결하게 요약

자유형(casual)

비형식적인 문단형태. 다양한 시나리오를 여러 문단에 걸쳐 표현한다.

완전 명세(fully dressed)

가장 상세한 형태. 모든 단계와 변형을 상세히 기록한다. 전제조건이나 성공보장과 같은 보조 항목들을 둔다.

Use Case > 모델링 > 유스케이스 명세서 > 약식

유스케이스: 유스케이스 이름을 적는다

주요 액터: 유스케이스를 시작시키는 주요 액터 이름을 적는다.

설명:

주요 시나리오를 2 ~ 4 문장으로 요약하여 간단히 적는다. 유스케이스에서 시스템 동작은 블랙 박스 형태로 표현한다. 즉, 시스템의 세부적인 처리나 구체적인 사용자 인터페이스 등을 수반하지 않고, 단지 책임(responsibility)에 충실한 형태를 말한다. 즉, '어떻게(how)' 보다는 '무엇을(what)' 에 초점을 맞춘다는 의미다. 따라서, 쓰임새 작성시 DB나 팝업 메뉴 등과 같은 용어 사용을 자제해야 한다.

Use Case > 모델링 > 유스케이스 명세서 > 약식 예제

유스케이스: 회원 가입하기

주요 액터: 비회원

설명:

비회원은 회원가입 페이지에서 이름, 이메일, 비밀번호 등의 필수 정보를 입력하고 회원 약관에 동의한다. 시스템은 입력 값을 검증하고 중복 이메일을 확인한 후, 새로운 회원 계정을 생성한다. 가입이 완료되면 사용자에게 환영 메시지를 표시한다.

유스케이스: 주문하기

주요 액터: 고객

설명:

고객은 장바구니에 담긴 상품을 확인하고, 배송 주소 및 결제 수단을 선택한다. 시스템은 재고를 확인하고 결제 요청을 처리한 후, 주문번호를 생성하여 고객에게 주문 완료 확인 메시지를 보여준다. 주문 결과는 고객 계정과 관리자 시스템에 기록된다.

Use Case > 모델링 > 유스케이스 명세서 > 완전 명세

유스케이스: 유스케이스 이름을 적는다

주요 액터: 유스케이스를 시작시키는 주요 액터 이름을 적는다.

관련자 및 관심사항: 이해 관계자와 그들의 관심사항을 적는다.

사전 조건: 유스케이스 실행 전에 반드시 참이어야 하는 시스템 상태를 적는다.

사후 조건: 유스케이스를 실행한 후에 반드시 보장되는 결과를 적는다.

주요 성공 시나리오: 성공하기 위한 전형적이고 조건이 없는 경로의 시나리오를 적는다.

확장: 성공이나 실패에 대한 대안 시나리오를 적는다.

Use Case > 모델링 > 유스케이스 명세서 > 완전 명세 예제

유스케이스: 회원 가입하기

주요 액터: 비회원

관련자 및 관심사항:

- 비회원 – 가입 절차가 간단하고 오류 없이 완료되길 원함
- 운영자 – 정확하고 검증된 회원 데이터 확보
- 시스템 보안 관리자 – 중복 계정 방지 및 비밀번호 암호화
- 마케팅 부서 – 신규 회원 데이터 확보를 통한 홍보 활용

사전 조건:

- 사용자가 회원 페이지에 접근 가능한 상태여야 한다.
- 사용자에게 유효한 이메일 주소가 있어야 한다.

사후 조건:

- 성공: 신규 회원 계정이 생성되고, 시스템에 저장됨.
- 실패: 계정 생성 실패 시 오류 메시지가 표시되고, 데이터는 저장되지 않음.

Use Case > 모델링 > 유스케이스 명세서 > 완전 명세 예제(계속)

주요 성공 시나리오:

1. 비회원이 '회원가입' 버튼을 클릭할 때 유스케이스를 시작한다.
2. 시스템은 회원가입 페이지를 표시한다.
3. 액터가 이름, 이메일, 비밀번호를 입력한다.
4. 액터는 이용약관에 동의한다.
5. 액터는 '가입하기' 버튼을 클릭한다.
6. 시스템은 입력 값 검증 및 이메일 중복여부를 확인한다.
7. 시스템은 새 회원 계정을 생성한다.
8. 시스템은 회원가입 성공 메시지를 표시하고 로그인 페이지로 이동한다.

확장:

7-a. 이메일이 존재하면, 시스템은 "이미 등록된 메일입니다" 안내 메시지를 표시하고 3단계로 돌아간다.

7-b. 비밀번호 형식이 잘못되었으면, 시스템은 비밀번호 규칙 안내 메시지를 표시하고 3단계로 돌아간다.

7-c. 약관에 동의하지 않으면, 시스템은 "약관에 동의해야 가입할 수 있습니다." 경고를 표시한 후 4번으로 돌아간다.

8-a. 시스템 오류가 발생한다면, "시스템 오류입니다." 메시지를 표시한다.