



# *Practical Object Modeling*

v 1.7



# 목 차

---

1. SW 기술 흐름
2. 시스템과 모델링
3. 객체지향 설계 원칙
4. 협업
5. 이해
6. 모델링 절차
7. 실습

## 1. SW 기술 흐름

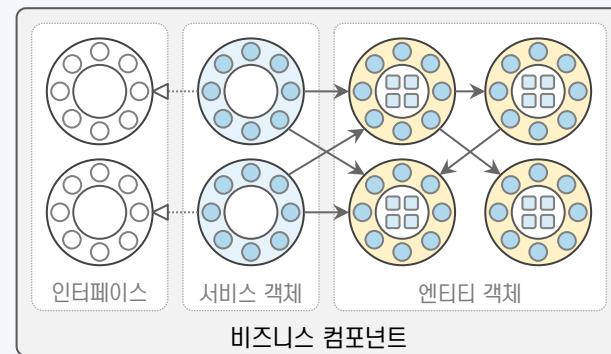
---

- ❖ 기술의 흐름 속에서 객체 지향이 차지하는 위상을 알아 봅니다.
- ❖ 우리의 현실 속에 객체 지향은 어떻게 다루어지는지 알아 봅니다.
- ❖ 언어의 변환이라는 관점에서 시스템 개발을 이해합니다.

- ✓ 왜 객체 지향인가?
- ✓ 우리들의 객체 지향
- ✓ 언어 매팅
- ✓ 요약

# 왜 객체 지향인가? (1/4)

- ✓ SW 설계 접근방법 → 절차 지향 설계와 객체 지향 개념을 바탕으로 하는 설계(OOAD, CBD, SOA, MSA)
- ✓ 두 세계는 패러다임이 전혀 다릅니다. 프로그래밍 언어 역시 두 가지로 나눌 수 있습니다.
- ✓ 객체 지향 설계 이후, SW 설계는 비약적으로 성장하였으며, 현재에 이르러서는 모든 산업의 기반이 되었습니다.
- ✓ OOAD는 많은 문제를 안고 있었지만, 그것을 극복하고 컴포넌트 기반 설계로 발전하였습니다.



질서정연한 객체들의 집합으로 구성한 비즈니스 컴포넌트

1995년 ~

OOAD (객체 지향 분석 설계)

절차 지향 분석 설계

2002년 ~

CBD (컴포넌트 기반 개발)

2008년 ~

SOA (서비스 지향 아키텍처)

2014년 ~

MSA (마이크로 서비스 아키텍처)

객체지향

온도<100 일 때만  
온도 1도 증가

1도 증가 요청  
물의 온도

절차지향

데이터

함수들

물의 온도

98

# 왜 객체 지향인가? (2/4)

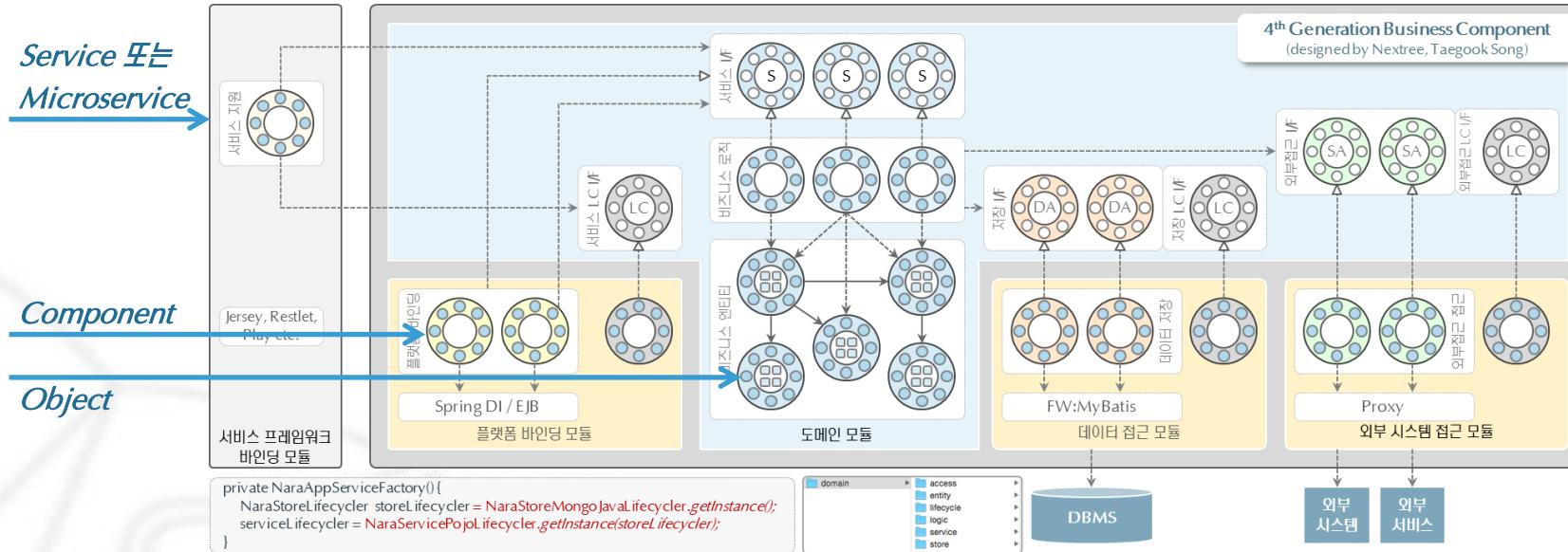
- ✓ 복잡한 컴퓨팅 환경은 기업 내부 분산을 넘어서 기업 간 분산 환경으로 발전하였습니다.
- ✓ SOA는 컴포넌트를 뛰어 넘어, 비즈니스 단위로 써의 서비스로 발전하였으며, 다양한 시련을 겪어 왔습니다.
- ✓ 컴포넌트 다음 세대의 서비스는 플랫폼과 환경, 표준 등의 문제를 극복하면서 마이크로 서비스로 발전하였습니다.
- ✓ 이전 기술을 버리고 새로운 기술로 발전한 것이 아니라, 이전 기술의 축적을 바탕으로 발전하여 갔습니다.



# 왜 객체 지향인가? (3/4)

- ✓ 객체 지향 기술과 컴포넌트 기술은 퇴색되지 않고, 서비스 저 안쪽에 탄탄하게 자리잡고 있습니다.
- ✓ 객체 모델링 역량은 컴포넌트와 서비스 내부를 잘 채우는데 도움을 줍니다.
- ✓ 컴포넌트 모델링 역량이 있어야 기술적으로 또는 업무적으로 의미 있는 객체 그룹을 구성할 수 있습니다.
- ✓ 무늬만 컴포넌트로 만드는 이유는 객체 모델링 역량과 컴포넌트 모델링 역량이 부족하기 때문입니다.

[ 4세대 비즈니스 컴포넌트와 서비스 ]



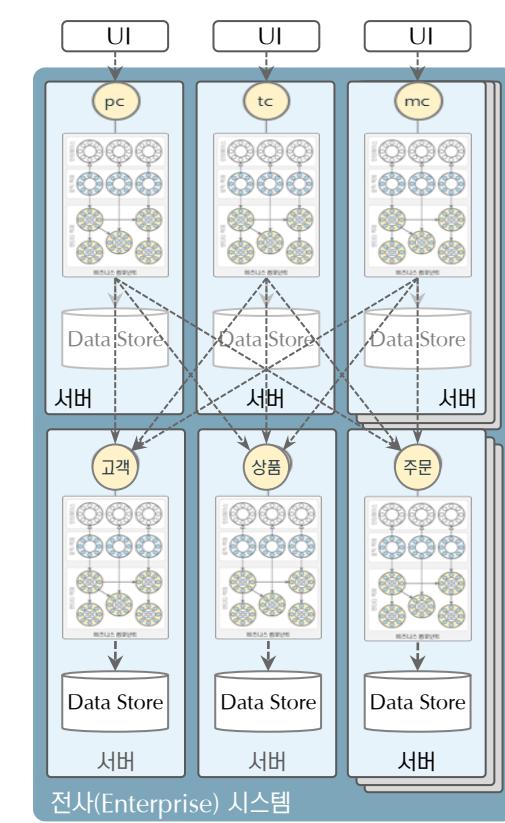
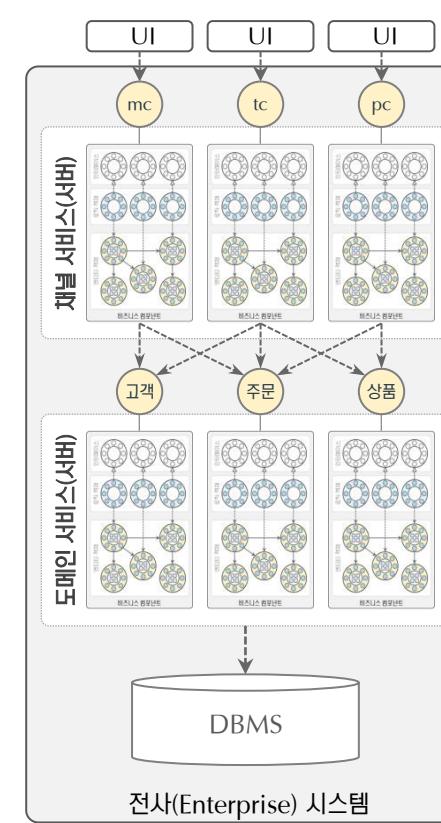
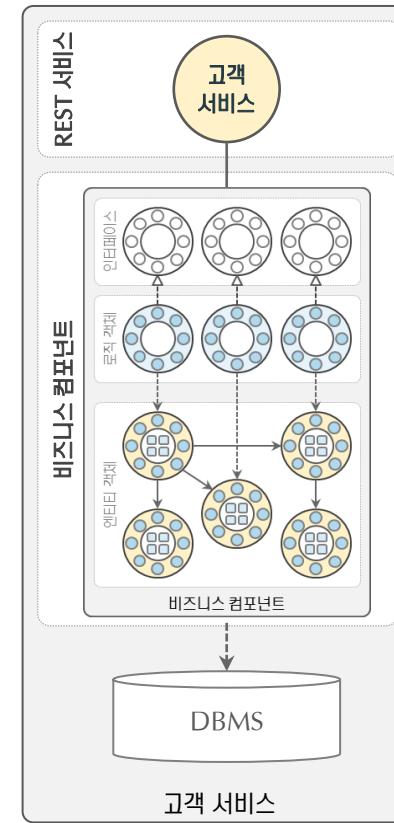
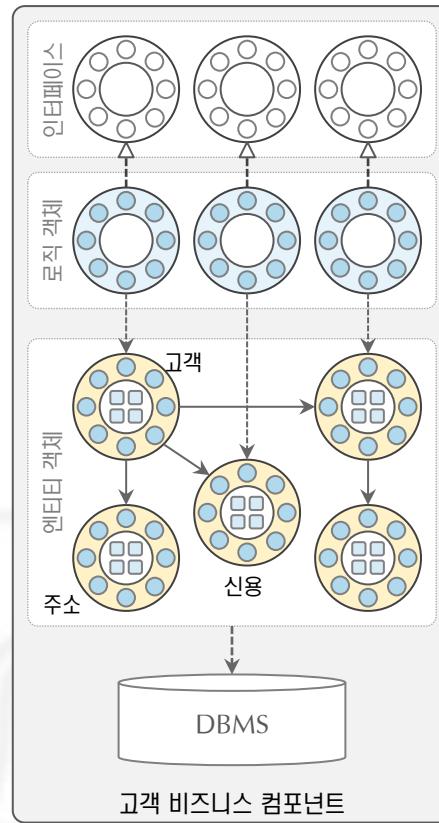
# 왜 객체 지향인가? (4/4)

- ✓ 객체 지향 프로그래밍 언어 출현 이후, 기술의 흐름은 모두 객체를 기반으로 하고 있습니다.
- ✓ 컴포넌트는 객체들의 질서정연한 구조를 의미하고, 서비스는 컴포넌트를 사용하는 방법을 의미합니다.
- ✓ CBD → SOA → Microservices 로의 흐름 내내 객체는 내부를 채우고 있습니다.

```

public class TdDiffNodeIterator {
    ...
    private int nodeSize;
    private int currentIndex;
    private List<TdDiffNode> attrModelList;
    ...
    public int size() {
        ...
        return attrModelList.size();
    }
    ...
    public boolean hasNext() {
        ...
        if (currentIndex < nodeSize) {
            return true;
        }
        return false;
    }
    ...
    public TdDiffNode next() {
        ...
        if (hasNext()) {
            return attrModelList.get(currentIndex++);
        } else {
            return null;
        }
    }
}

```



# 우리들의 객체 지향 (1/5)

- ✓ 개발자라면 누구나 “객체 지향” 개념을 이해하고 있고, “객체 지향” 언어를 사용하고 있습니다.
- ✓ 여러분들이 지금까지 배워온 객체 지향에 대해서 토론하는 시간입니다.
- ✓ 객체 지향 개념을 여러분들의 프로그래밍에 어떻게 적용하고 있습니까?
- ✓ 상속(inheritance) 개념을 알고 있기 때문에 여러분의 프로그램 재사용이 늘어났습니까?

→ 객체 지향의 핵심 개념은 무엇입니까?

→ 객체 지향 프로그래밍 언어라서 좋은 점은 무엇입니까?

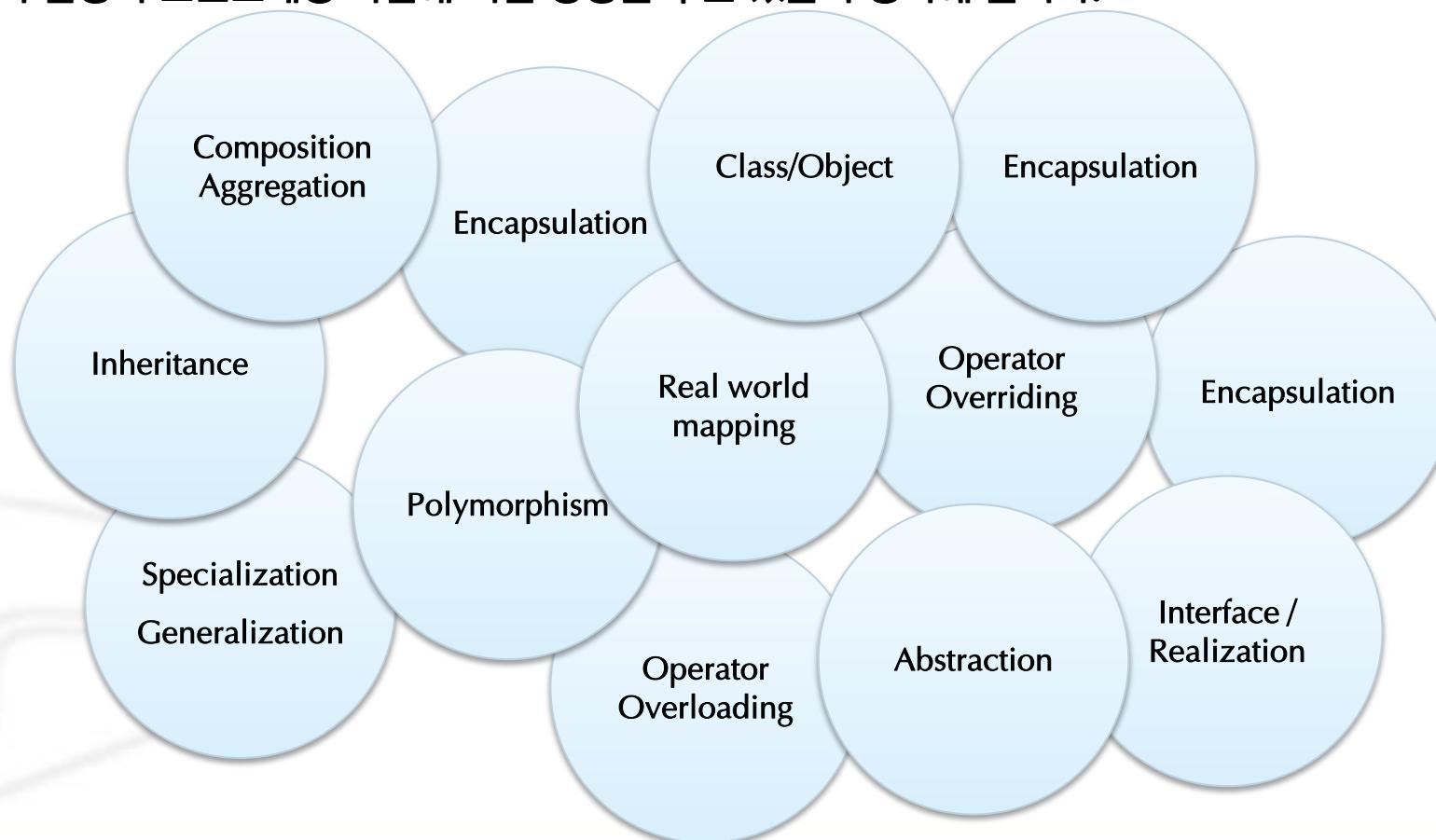
→ 상속은 객체 지향을 대표하는 개념인데 많이 사용합니까?

→ 여러분은 객체 지향 프로그래밍을 하십니까?

→ 여러분의 클래스, 메소드의 라인 수는 얼마입니까?

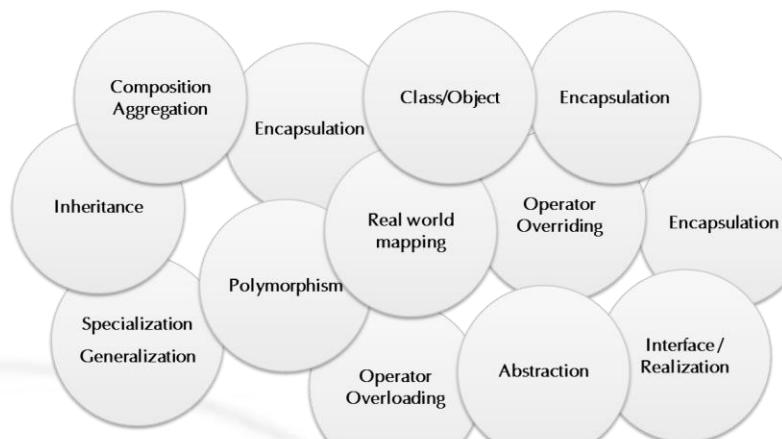
# 우리들의 객체 지향 (2/5)

- ✓ 처음 객체 지향 프로그램을 배울 때를 돌이켜 보면, 생소한 개념 때문에 힘들어 했던 기억이 있을 것입니다.
- ✓ 그런 개념을 이해하고 그 개념들을 프로그래밍 할 때 충분히 활용하였는지 생각해 봅시다.
- ✓ 참 많은 개념들이 있었으며, 그 끝에 남은 것은 클래스, 메소드 등의 개념이 아닌 Syntax는 아니었을까요...
- ✓ 이러한 개념들이 일상의 프로그래밍 작업에 어떤 영향을 주고 있는지 생각해 봅니다.



# 우리들의 객체 지향 (3/5)

- ✓ 처음 배웠던 어렵고 복잡한 객체 지향 개념들은 어디로 갔는지 생각해 봅니다.
- ✓ 여러분의 프로그램 속으로 들어갔다면 여러분의 프로그램은 멋진 프로그램일 수 밖에 없습니다.
- ✓ 어디론가 사라졌다면 여러분은 아직도 “절차 지향” 프로그래밍을 하고 있을 수 있습니다.
- ✓ 객체 지향 개념들이 우리 프로그램에 반영되지 않았다면, 왜 그렇게 되었는지 알아야 합니다.



여러분의 객체 지향 개념

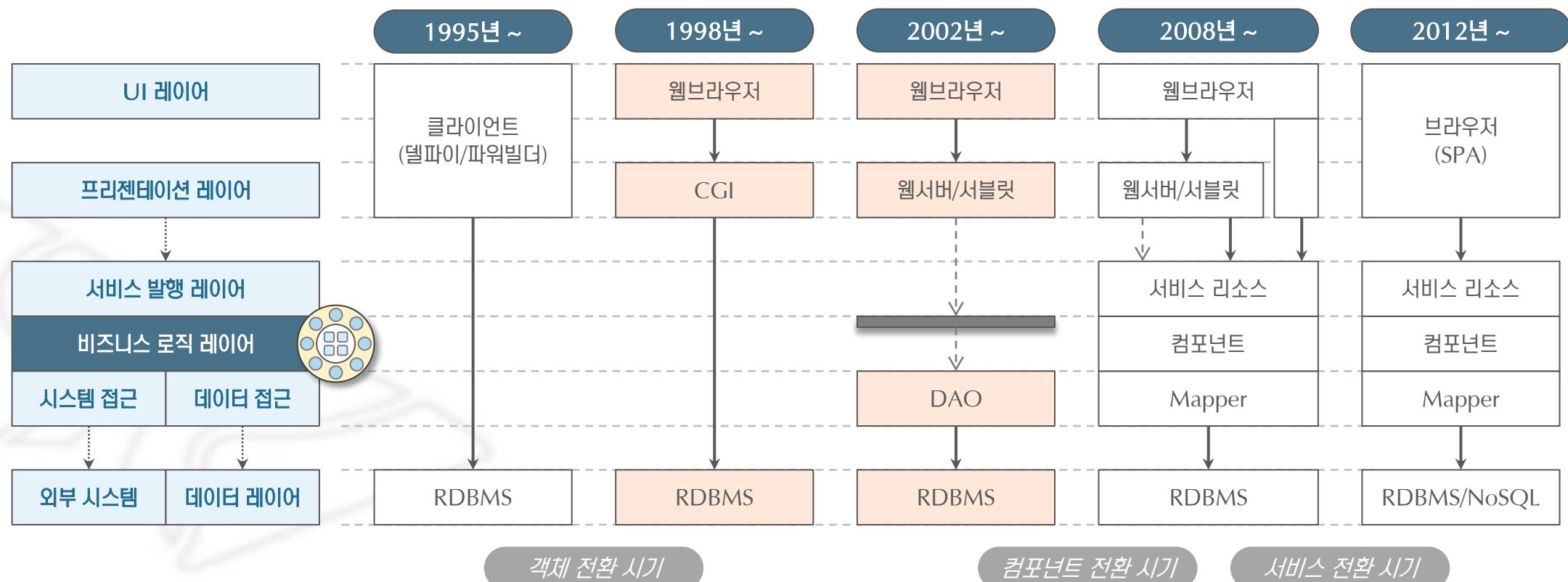
```
import com.nextree.omw.crossroad.Logger;  
public class TrafficLight extends TrafficDevice {  
    private GreenLight greenLight;  
    private RedLight redLight;  
    public TrafficLight(String name, RoadSide side) {  
        super(name);  
        super.setRoadSide(side);  
        this.greenLight = new GreenLight();  
        this.redLight = new RedLight();  
        this.greenLight.off();  
        this.redLight.on();  
    }  
    protected void toggleLight() {  
        // 지정시간 토글  
        if (isGreenOn()) {  
            setRedOn();  
        } else {  
            setGreenOn();  
        }  
    }  
    protected LightColor getLightColor() {  
        if (isGreenOn()) {  
            return LightColor.Green;  
        } else if (isRedOn()) {  
            return LightColor.Red;  
        }  
        throw new RuntimeException("신호등의 색상이 없습니다...");  
    }  
}
```

여러분의 객체 지향 프로그램



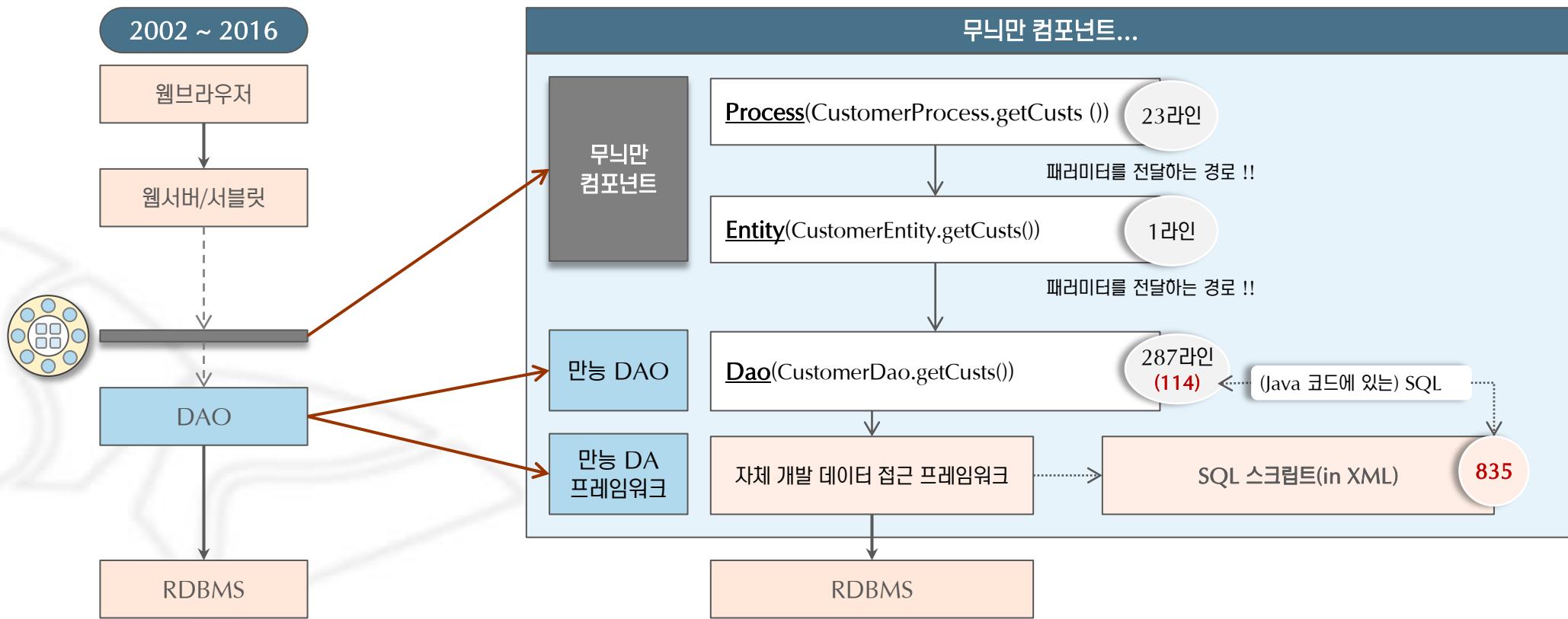
# 우리들의 객체 지향 (4/5) – 기술 유산

- ✓ 기업 컴퓨팅 환경은 2000년 이전 C/S 환경으로부터 시작하여 현재의 MSA까지 발전해 왔습니다.
- ✓ 국내 기업들 대부분 객체로 전환을 놓치고, 그 결과 “무늬만” 컴포넌트 시대를 지나왔습니다.
- ✓ 컴포넌트로 전환하지 못한 기업이 서비스로 전환하는 것은 불가능에 가까운 일이었습니다.
- ✓ 여전히 많은 기업이 1998년 또는 2002년 기술 구조에 머무르고 있으며, 그 원인은 객체 기술 부재입니다.



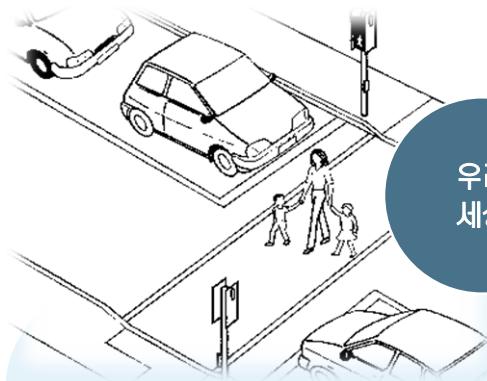
# 우리들의 객체 지향 (5/5) – 기술 유산

- ✓ 설계 기술이 없던 시절 잠시 유행하고 말았어야 할 “무늬만 컴포넌트”가 일상화 되고 말았습니다.
- ✓ 컴포넌트 자체가 없는 구조도 많지만, 있다고 하더라도 컴포넌트를 열어보면 그 속에는 아무것도 없습니다.
- ✓ 도메인 객체, 로직 객체 등으로 가득 차 있어야 할 컴포넌트는 파라미터 전달 경로로 사용할 뿐입니다.
- ✓ 객체가 주로 활동할 공간을 비워 둔 설계 유산 때문에 객체 지향 설계와 프로그래밍이 설 자리를 잃었습니다.

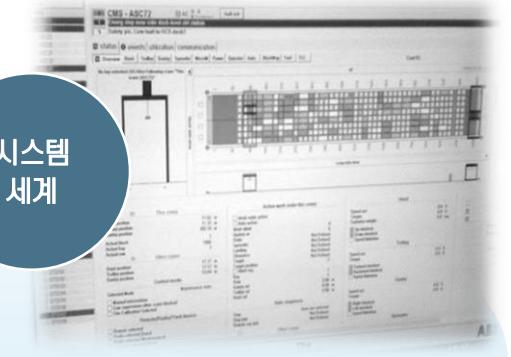


# 언어 매핑(1/3)

- ✓ 우리가 살고 있는 세상에 자주 발생하는 불편함을 해소하기 위해 [SW intensive] 시스템을 개발합니다.
- ✓ 우리의 불편을 해결할 시스템은 우리가 사용하는 표현으로부터 출발하여 자그마한 시스템 세계를 구성합니다.
- ✓ 자연어 → 모델링 언어 → 프로그래밍 언어를 거치면서 시스템 세계가 요구하는 특성을 점차 반영합니다.

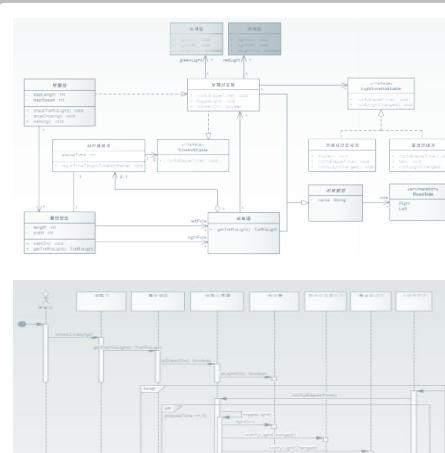


언어에서 언어로의 변환(Transformation)



시스템 세계

1. 행인이 횡단보도로 다가가며 보행자신호등을 확인한다. 적색 등이 켜져있다. 녹색등을 기다리며 멈춰선다.
2. 시간이 흘러 녹색등이 켜지고, 또 시간이 흘러 적색등이 켜진다. 녹색등과 적색등이 교대로 켜진다.
3. 녹색등이 켜지면 음성 안내기는 “녹색등입니다. 건너가십시오 오.”라는 안내를 하고 “뚜르르, 뚜르르....”하고 알림음을 낸다. 5초가 남으면 “위험합니다. 건너지 마십시오.”라는 안내를 하고, 0초가 남으면 알림음을 멈춘다.
4. 녹색등이 켜지면 남은시간표시기에 숫자가 20부터 시작하여 1초 단위로 줄여든다. 적색등이 켜지면 남은 시간 표시기는 깨진다.
5. 행인은 횡단보도를 건너간다. 시간이 부족하면 달린다.



요구사항 – 자연언어

모델링 – UML

프로그램 – 프로그래밍언어

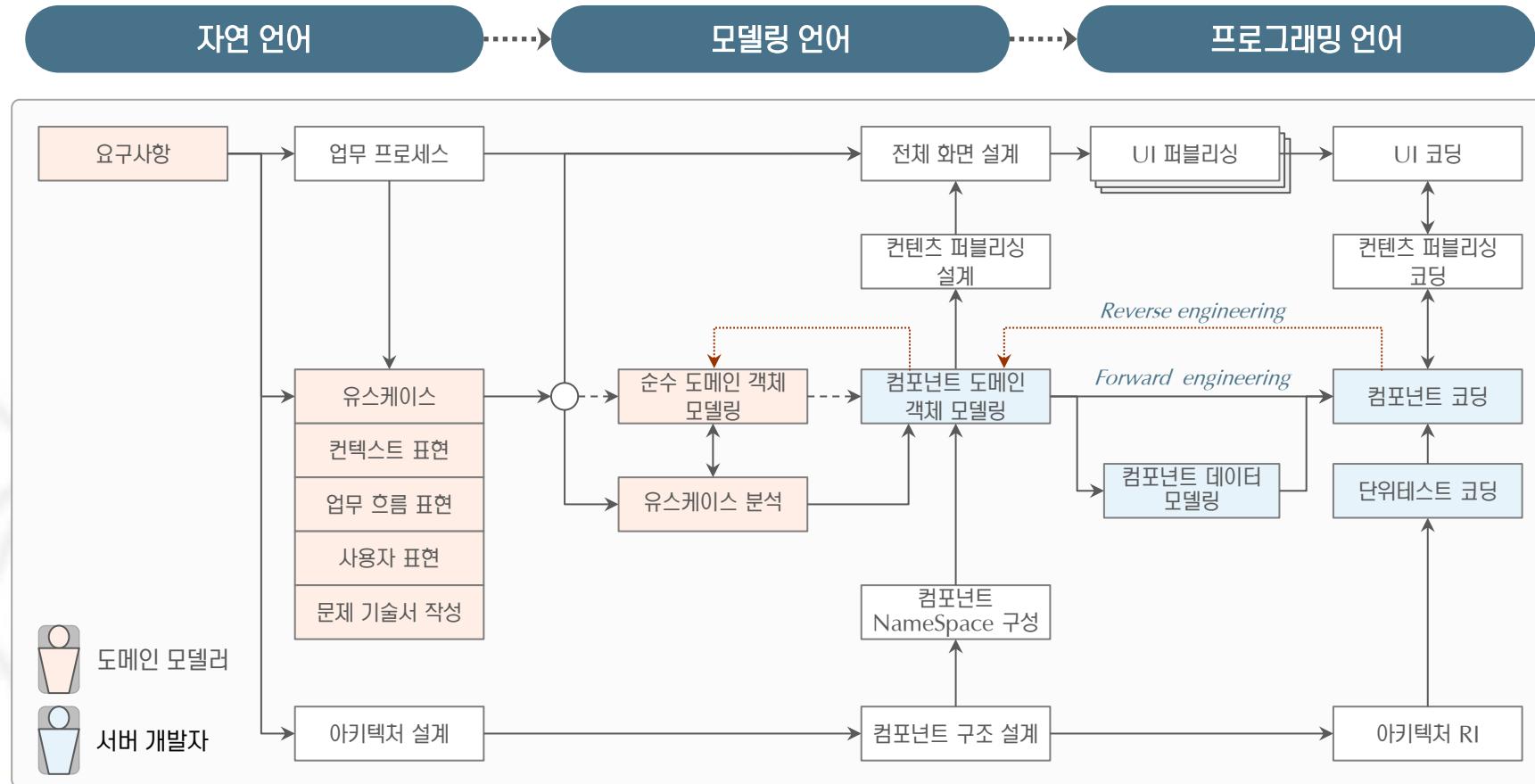
실행코드 – 기계어

```
import com.nextree.cmw.crossroad.Logger;
public class TrafficLight extends TrafficDevice {
    private GreenLight greenLight;
    private RedLight redLight;
    public TrafficLight(String name, RoadSide side) {
        super(name);
        super.setRoadSide(side);
        this.greenLight = new GreenLight();
        this.redLight = new RedLight();
        this.greenLight.off();
        this.redLight.on();
    }
    protected void toggleLight() {
        // 지정시간 동안
        if (isGreenOn()) {
            setRedOn();
        } else {
            setGreenOn();
        }
    }
    protected LightColor getLightColor() {
        if (isGreenOn()) {
            return LightColor.Green;
        } else if (isRedOn()) {
            return LightColor.Red;
        }
        throw new RuntimeException("신호등의 색상이 없습니다....");
    }
}
```

```
F9F9F9F9 F9F9F9F9 F9F9F9F9 D9E9E9E9 E9F9F9F9
229189F9 F9F9F9F9 F9F9F9F9 F9F9F9F9 F9F9F9F9
79F9F9F9 F9F9F9F9 F9F9F9F9 F9F9F9F9 F9F9F9F9
UUF8FUU FCFCFCFC FCFCFCFC FCFCFCFC FCFCFCFC
FCFCFCFC FCFCFCFC FCFCFCFC FCFCFCFC FCFCFCFC
FCFCFCFC FCFCFCFC FCFCFCFC FCFCFCFC FCFCFCFC
J1F035EA 7E7E7E7E 7E7E7E7E 7E7E7E7E 7E7E7E7E
7E7E7E7E 7E7E7E7E 7E7E7E7E 7E7E7E7E 7E7E7E7E
7E7E7E7E 7E7E7E7E 7E7E7E7E 7E7E7E7E 7E7E7E7E
3F3F3F3F 3F3F3F3F 3F3F3F3F 3F3F3F3F 3F3F3F3F
3F3F3F3F 3F3F3F3F 3F3F3F3F 3F3F3F3F 3F3F3F3F
3F3F3F3F 3F3F3F3F 3F3F3F3F 3F3F3F3F 3F3F3F3F
```

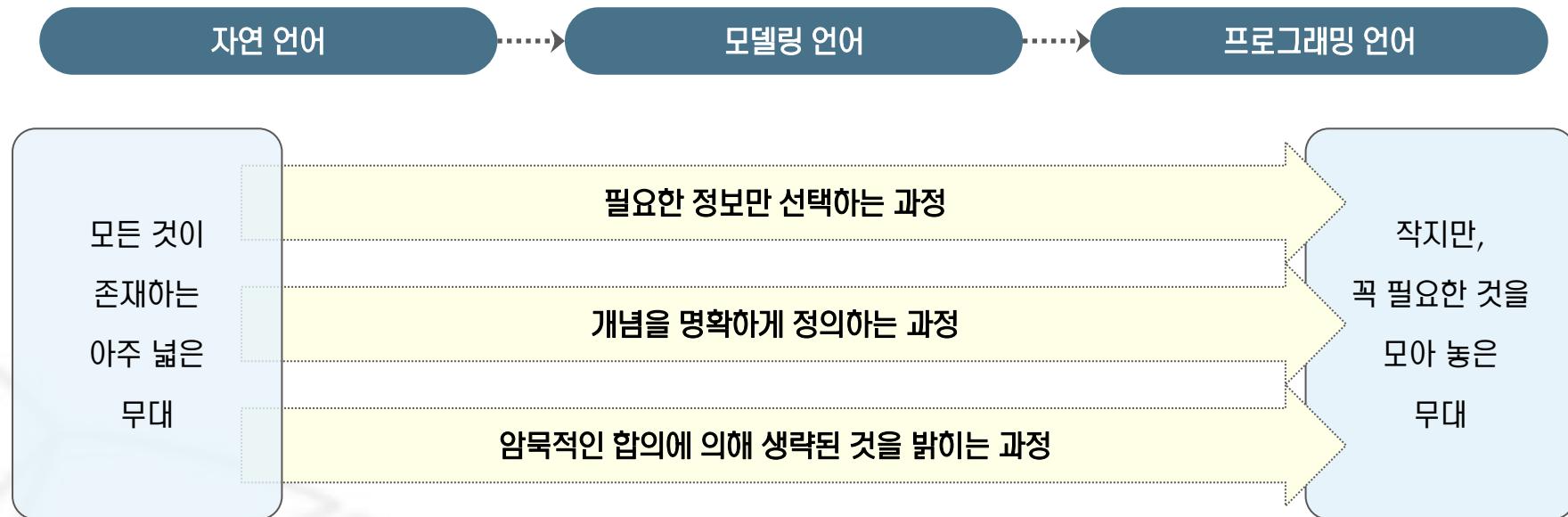
# 언어 매핑(2/3)

- ✓ SW 개발 절차를 살펴보면, 서로 다른 언어로의 변환(transformation)이 이어짐을 알 수 있습니다.
- ✓ 요구사항은 실세계의 자연 언어로 표현하고, 애플리케이션은 시스템 세계의 프로그래밍 언어로 표현합니다.
- ✓ 어떻게 보면, 시스템 세계는 특정 관점을 유지하며 구축한 작은 모형이라고 할 수 있습니다.
- ✓ 실세계는 모델링 세계로 매핑을 거치고, 모델링 세계는 다시 시스템 세계로 매핑이 됩니다.



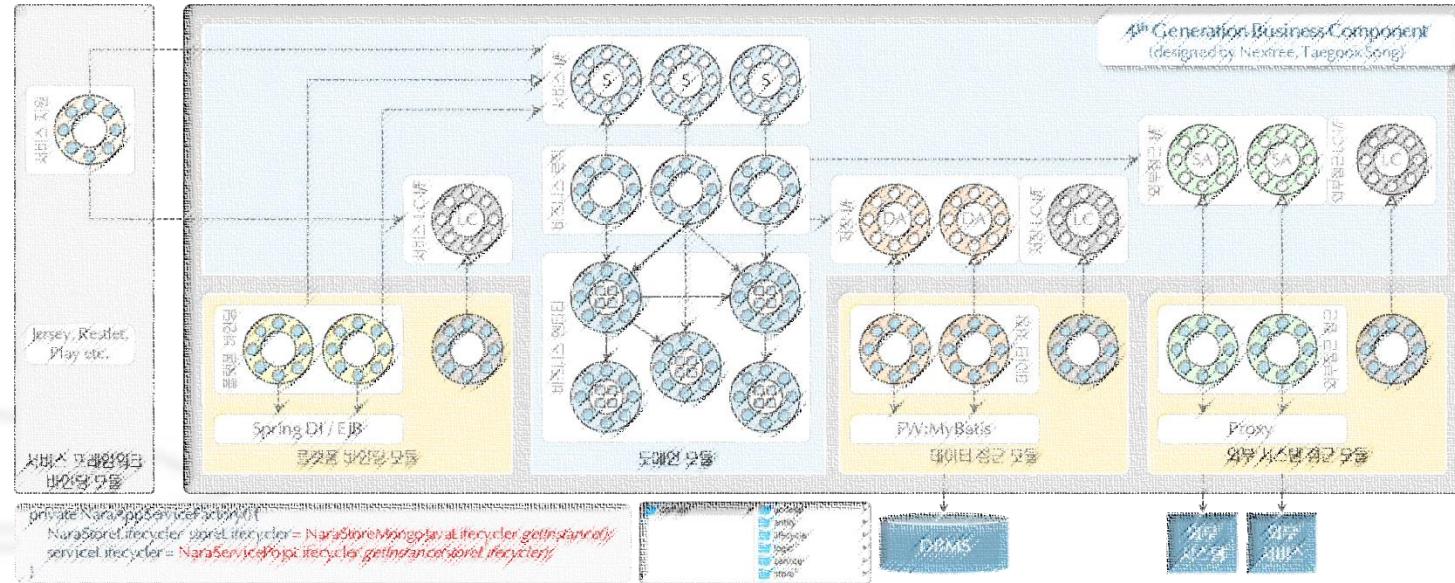
# 언어 매팅(3/3)

- ✓ 시스템 개발 과정에서 뺄 것은 빼고, 정의할 것은 정의하고, 밝힐 것은 밝히는 방식으로 개발을 진행합니다.
- ✓ 프로그래밍의 목표는, 작지만 꼭 필요한 것만 모아 놓은 것 같은 작은 무대를 시스템 안에 만드는 것입니다.
- ✓ 자연언어는 풍부한 표현을, 모델링 언어는 다양하지만 간결한 표현을, 프로그래밍 언어는 엄격한 표현을 가능하게 하므로, 우리가 필요한 것을, 구조화하여, 오류 없이 제공할 수 있습니다.



# 요약

- ✓ 현대의 소프트웨어 개발은 객체 → 컴포넌트 → 서비스 → 마이크로서비스로 흘러가고 있습니다.
- ✓ 각 단계 별로 충분히 성숙한 상태에서 다음 기술로 진화 발전할 수 있습니다.
- ✓ 하지만, 우리는 새로운 기술에 관심을 가질 뿐 지난 기술을 과거로 치부해 버리는 경향이 있습니다.
- ✓ 객체 지향 기술은 현대 소프트웨어 기술의 바탕이고 틀이 되는 접근 방법, 사고 방식, 그리고 기술입니다.



## 2. 시스템과 모델링

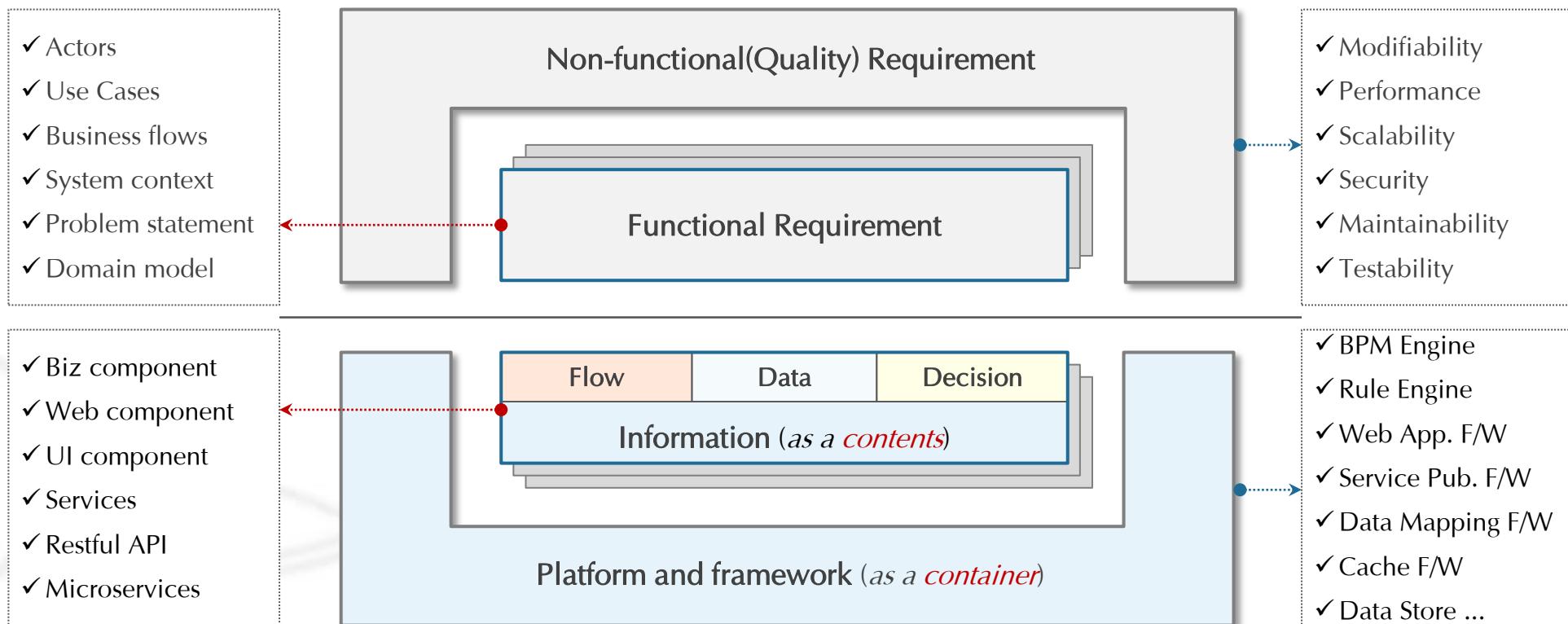
---

- ❖ 모델링을 해야 하는 이유를 생각해 봅니다.
- ❖ 모델링이 Thinking Process 인 이유를 이해합니다.
- ❖ 모델링을 해야 한다면, 올바른 절차는 무엇인지 알아 봅니다.

- ✓ 시스템 이해
- ✓ 모델링 이해
- ✓ 도메인 모델링
- ✓ 모델링 핵심 요소
- ✓ 모델링과 의사 결정
- ✓ 역할 – 도메인 모델러

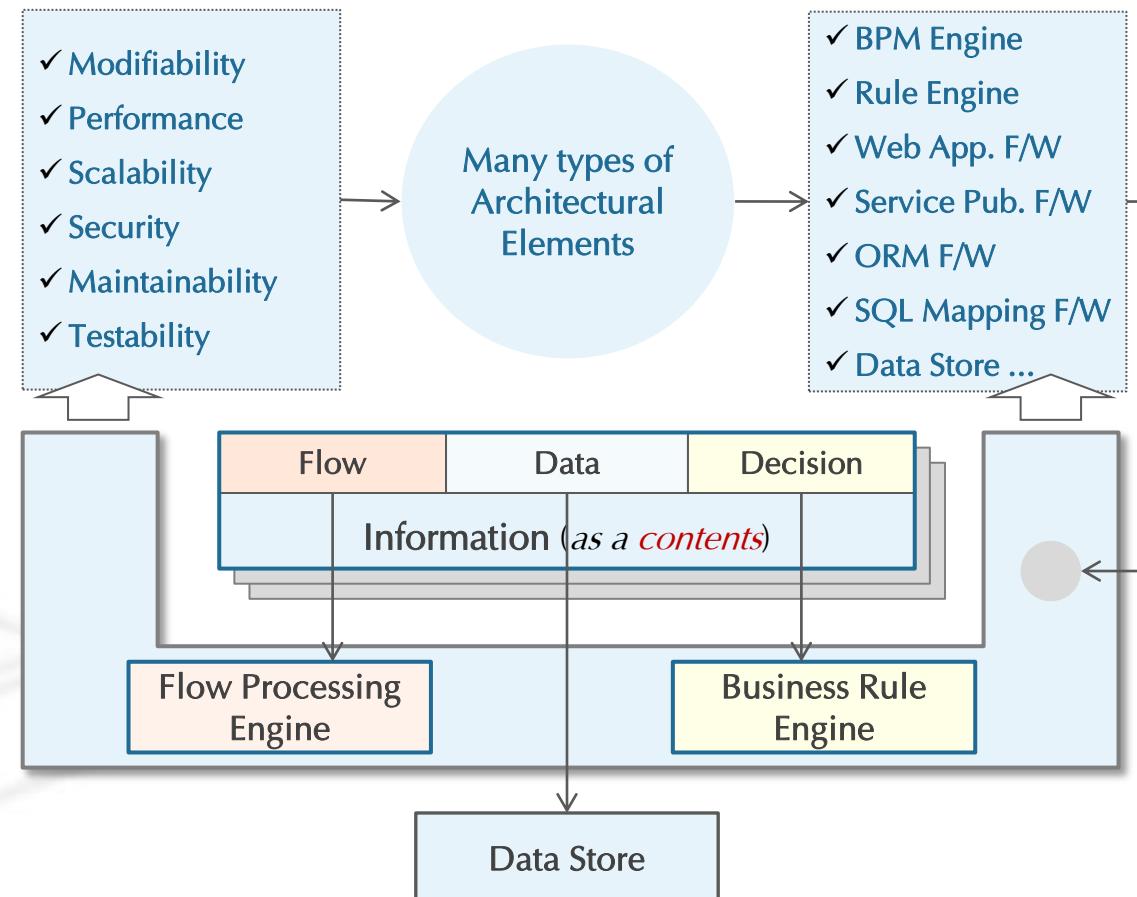
# 시스템 이해(1/4)

- ✓ 시스템의 요구사항은 기능 요구와 비기능 요구로 나누며, 두 가지를 반영하여 시스템을 개발합니다.
- ✓ 시스템은 비기능 요구를 반영한 결과를 컨테이너라고 표현할 수 있습니다. 그 속에는 많은 요소들이 있습니다.
- ✓ 컨텐츠는 기능 요구를 반영하며, 그 속에는 데이터, 흐름, 판단 등의 형태로 존재하는 정보들이 있습니다.



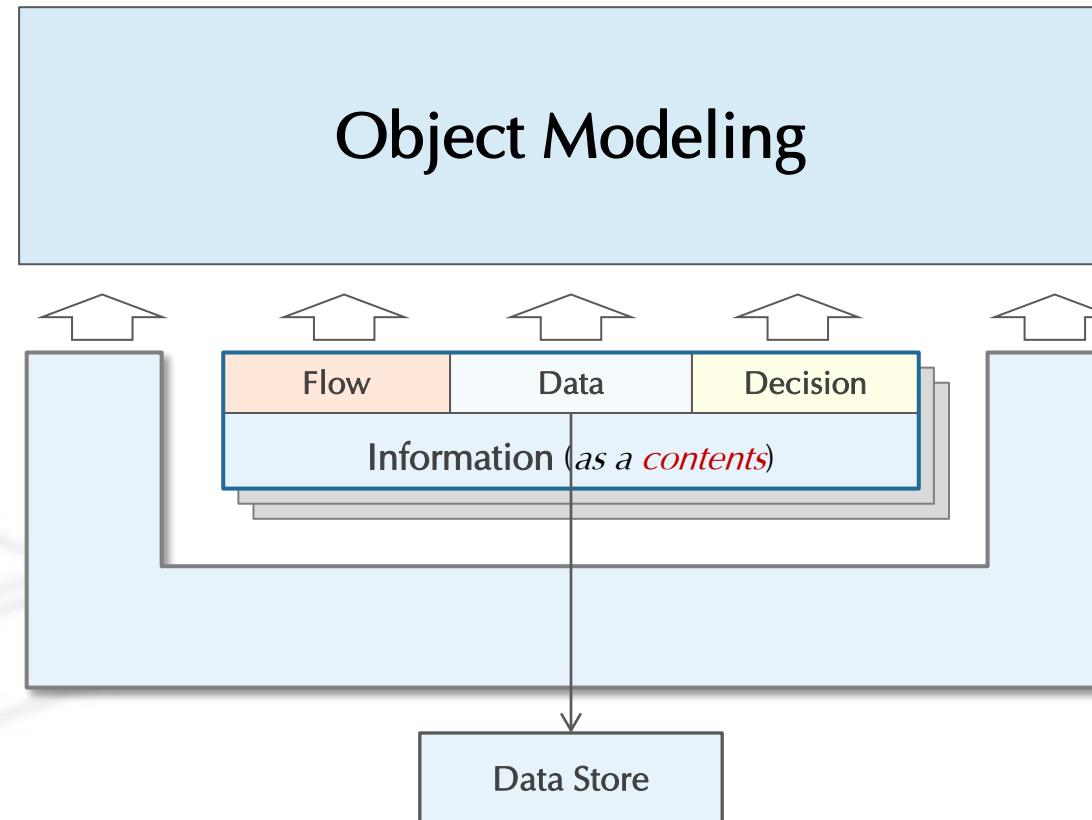
## 시스템 이해(2/4)

- ✓ 컨테이너를 구성하는 많은 아키텍처 요소들이 있습니다. → 프레임워크, 플랫폼, 라이브러리 등
- ✓ 컨테이너 구성 요소를 개발할 때도 모델링이 필요합니다. 이것은 아키텍처 모델링 영역입니다.
- ✓ 컨텐츠를 구성하는 컴포넌트, 서비스 등을 개발할 때 수행하는 모델링을 도메인 모델링이라고 합니다.



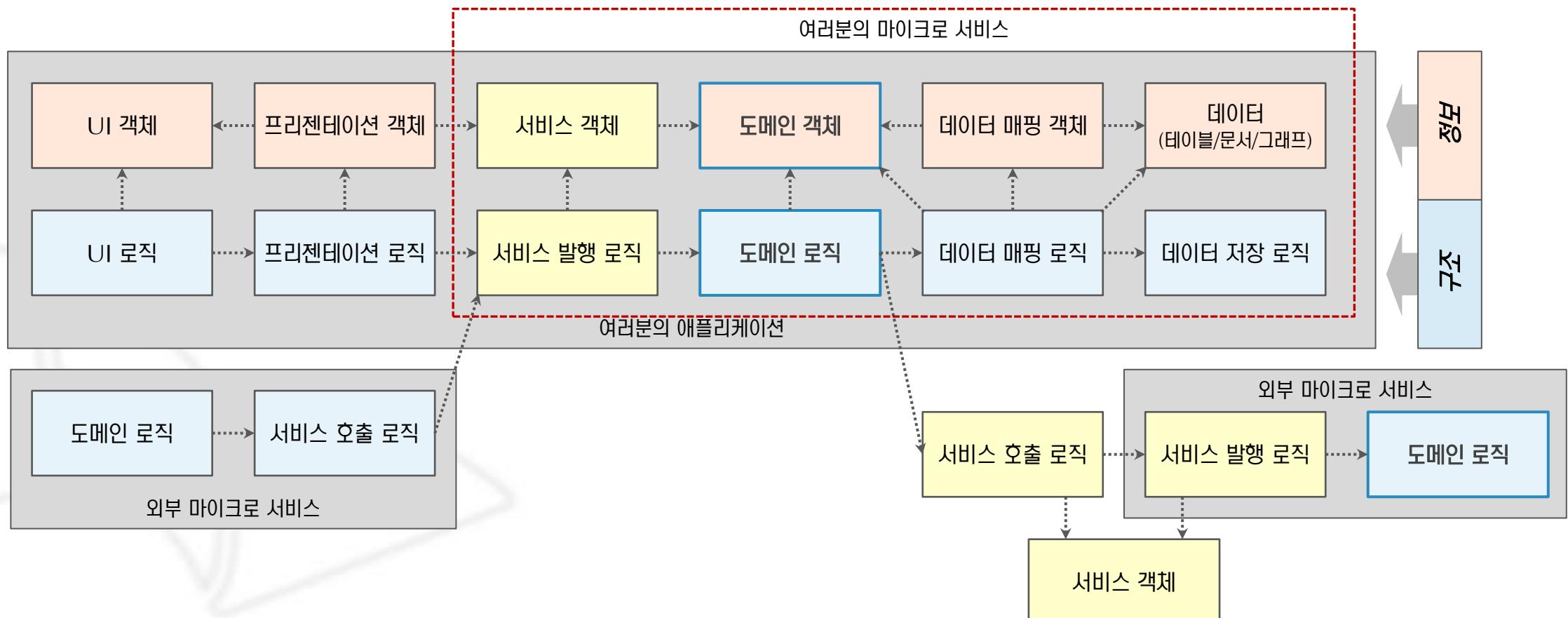
## 시스템 이해(3/4)

- ✓ 컨테이너를 구성하는 요소를 설계하는 컨텐츠를 구성하는 요소를 설계하는 객체 모델링이 기반이 됩니다.
- ✓ 비즈니스 컴포넌트를 설계해야 한다면, 당연히 객체 모델링 과정을 거쳐야 합니다.
- ✓ 데이터 접근 프레임워크나 연계 브로커를 설계해야 한다면, 이 활동 또한 객체 모델링 과정을 거쳐야 합니다.



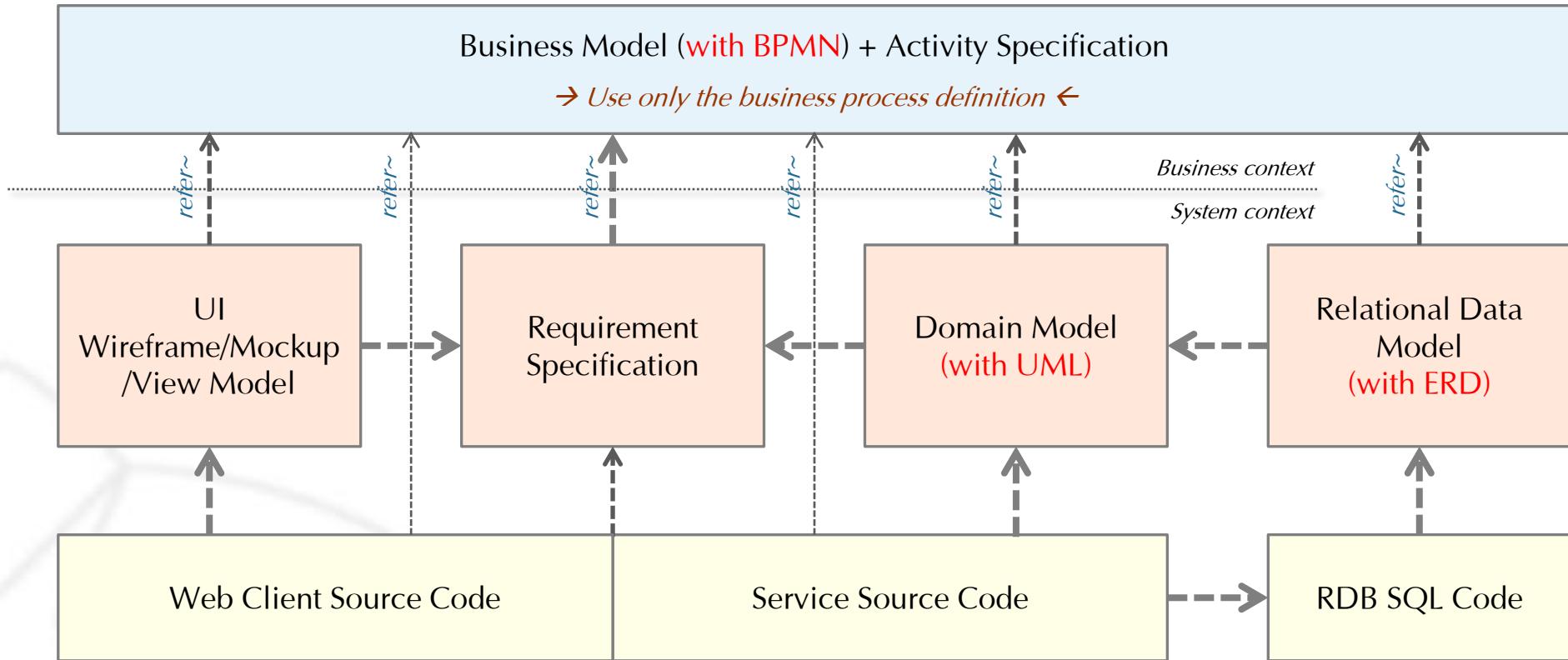
# 시스템 이해(4/4)

- ✓ 현대의 엔터프라이즈 아키텍처에서 독립형(stand-alone) 애플리케이션을 찾아보기 어렵습니다.
- ✓ 어떤 경우든 기존의 자원(서비스 시스템, 컴포넌트, 마이크로 서비스)을 사용하고, 다른 애플리케이션을 지원합니다.
- ✓ 복잡한 서비스 호출 네트워크가 현대 기업형 시스템의 특징이라고 할 수 있습니다.



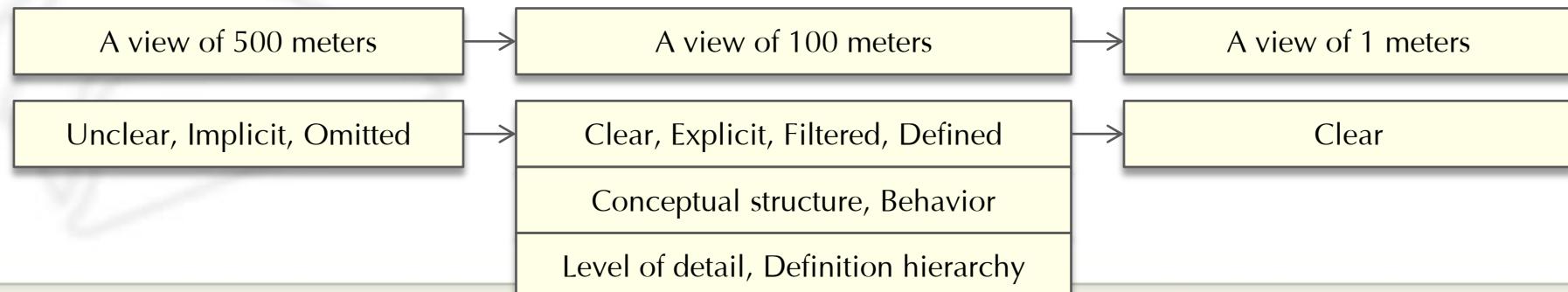
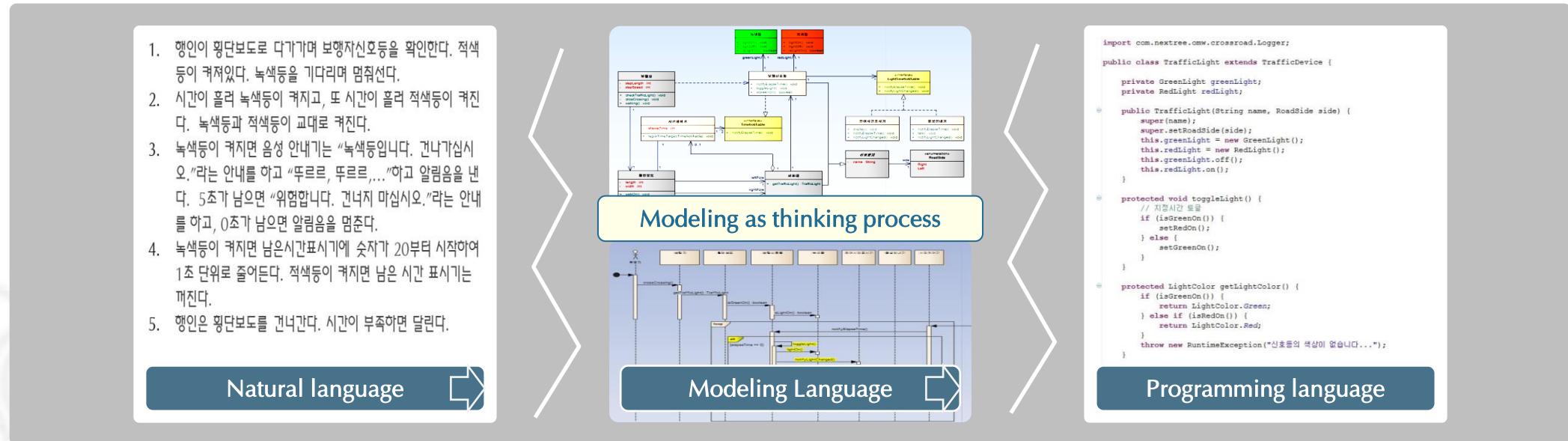
# 모델링 이해(1/4): 개요

- ✓ 표준으로 지정된 두 가지 모델링 언어가 있습니다. → BPMN, UML
- ✓ BPMN은 비즈니스 프로세스를 모델링을 목적으로 사용하는 모델링 언어입니다.
- ✓ UML은 시스템 모델링용 언어입니다. 컴포넌트를 설계하거나 프레임워크를 설계할 때 사용합니다.



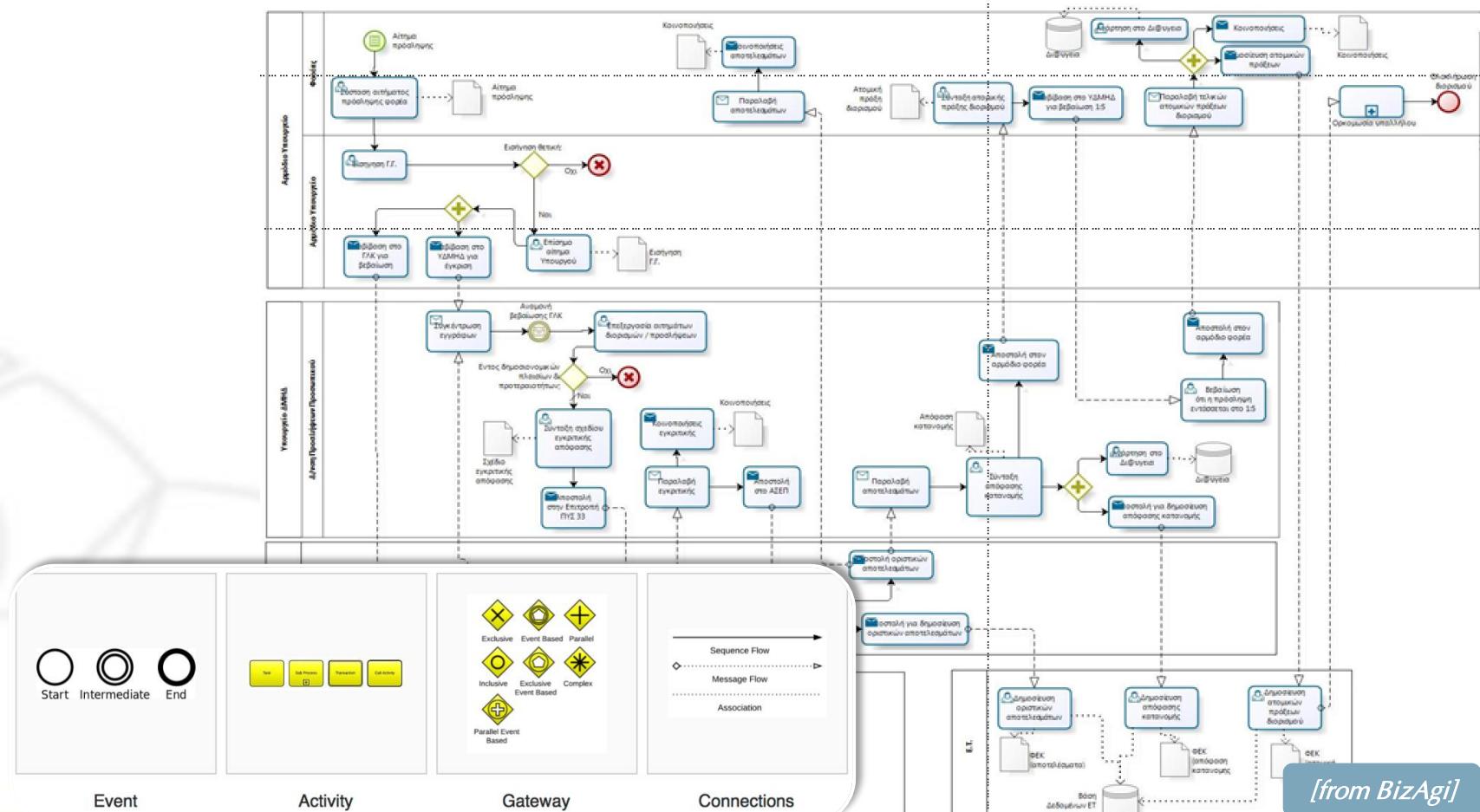
# 모델링 이해(2/4): Language transition

- ✓ 자연어, UML, Java는 서로 다른 목적을 가진 언어이지만 순서대로 참조를 합니다.
- ✓ 자연어로부터 모델링 언어를 자동 생성하거나, 모델링 언어로부터 프로그래밍 언어를 생성하려는 시도가 있었습니다. 일부 가능성도 확인하였지만, “고유의 목적”을 가진 언어 간의 경계를 없애려는 시도는 무의미합니다.



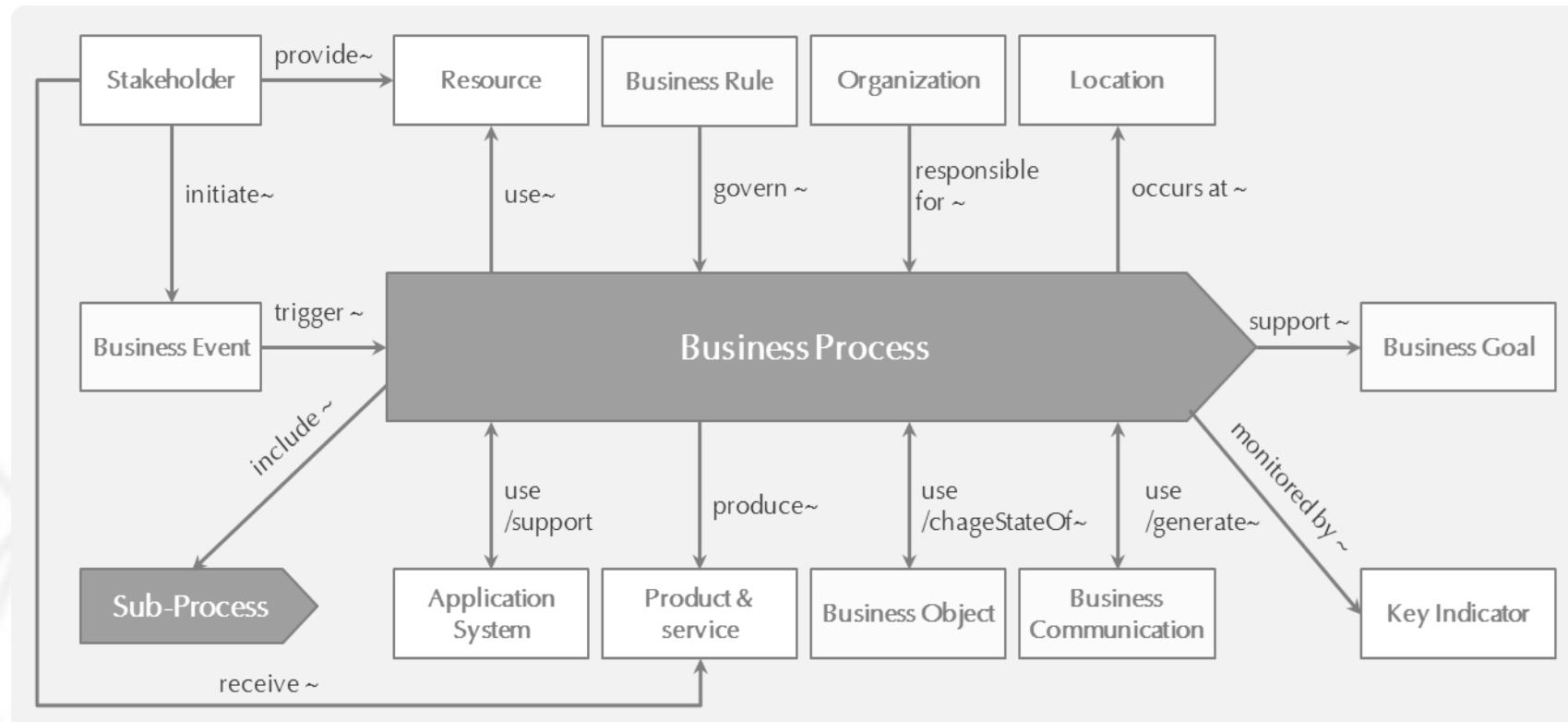
# 모델링 이해(3/4): Business Process Model 1

- ✓ BPMN의 일차 목적은 모든 비즈니스 이해관계자가 자연스럽게 이해할 수 있는 표준 표기법을 제공하는 것입니다. [from wikipedia.org]
- ✓ BPMN은 네 가지 유형의 개념이 바탕으로 합니다. Event, Activity, Gateway, and Connection.



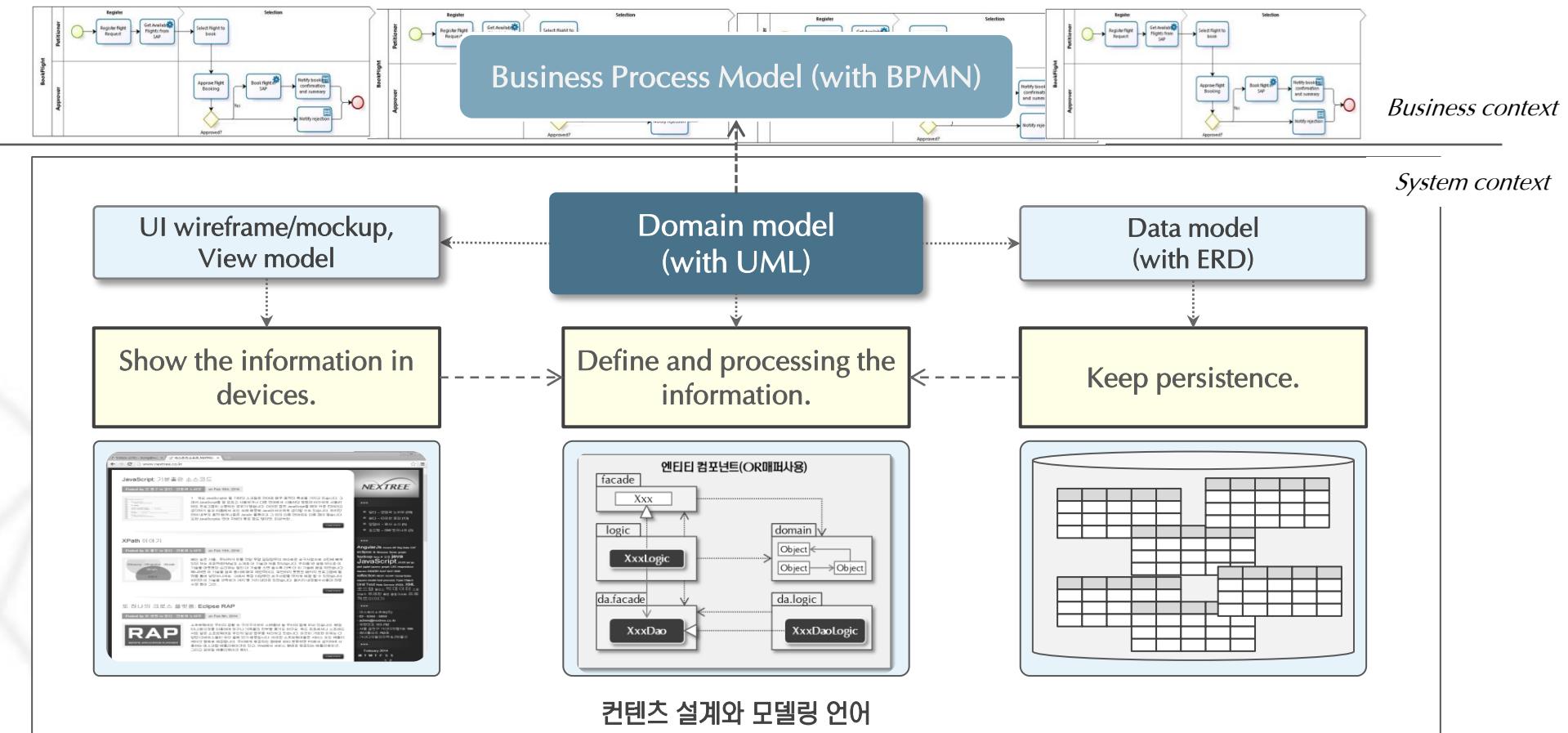
# 모델링 이해(4/4): Business Process Model 2

- ✓ 비즈니스 모델은 프로세스 뿐만 아니라 다양한 모델 요소를 포함하고 있습니다.
- ✓ BPMN은 그 중에서 프로세스 만을 모델링 대상으로 삼고 있습니다.
- ✓ 비즈니스 6대 모델에는 조직, 목표, 위치, 프로세스, 이벤트, 커뮤니케이션 등이 포함됩니다. ← 매우 다양함



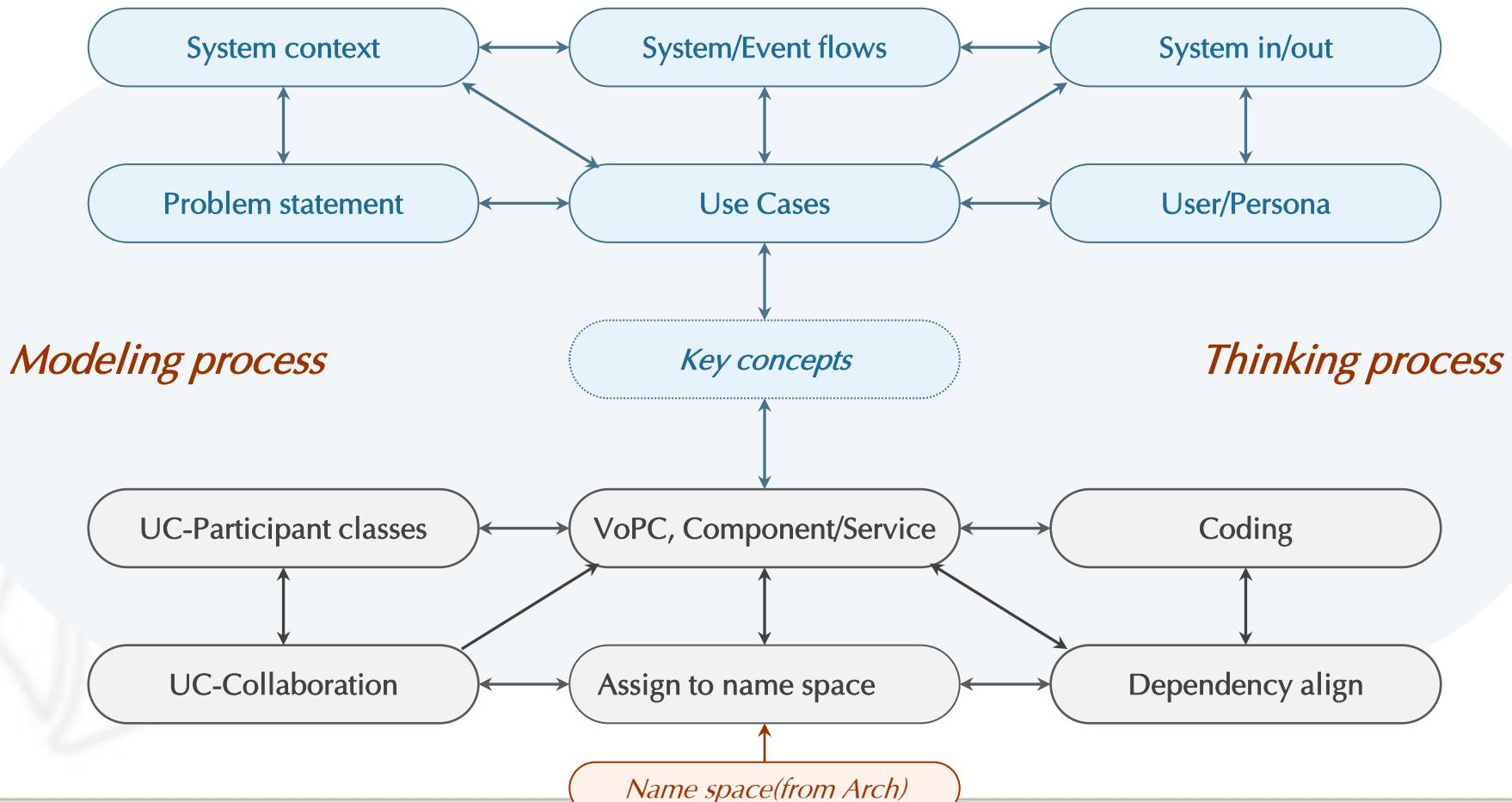
# 도메인 모델링 – 개요

- ✓ 도메인 모델은 모든 모델의 한 가운데 있으며, 업무를 표현하며, UML을 사용합니다.
- ✓ 데이터 모델은 도메인 모델을 참조하며, 도메인 모델 범위 안에서 설계를 하며, 저장 장치의 특성을 반영합니다.
- ✓ 뷰 모델은 클라이언트 디바이스의 영향을 받으며, 도메인 모델을 기반으로 합니다.



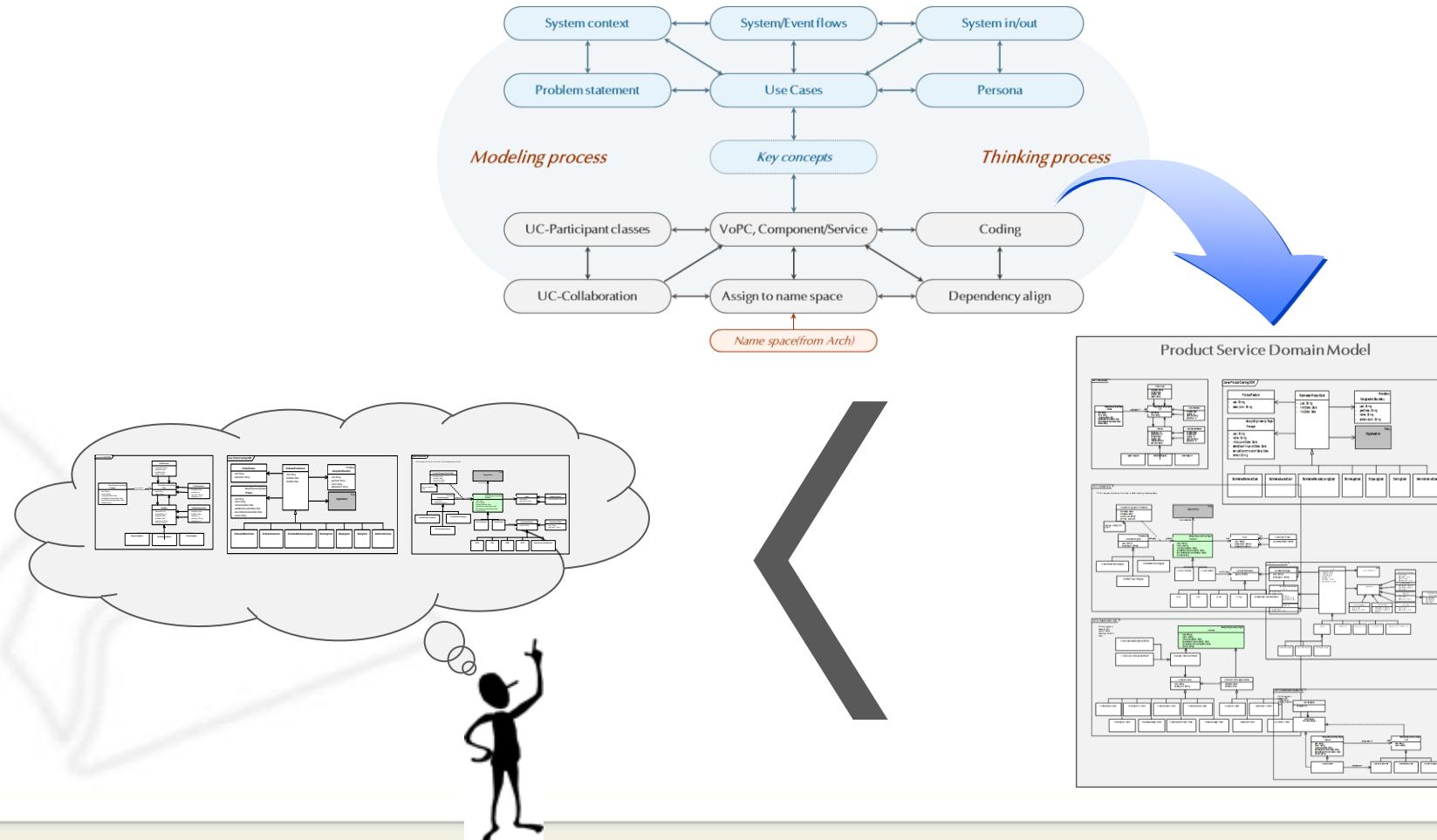
# 도메인 모델링 – 생각하는 과정

- ✓ 도메인 모델링은 도메인을 이해하고 모델러의 뇌 속에 정리하여 넣은 과정입니다.
- ✓ 모델링 절차는 도메인에 있는 대상을 다양한 관점과 다양한 상세도 수준에서 바라볼 수 있도록 도와 줍니다.
- ✓ 모델링은 단순히 클래스 다이어그램과 시퀀스 다이어그램을 그리는 활동이 아님을 알아야 합니다.



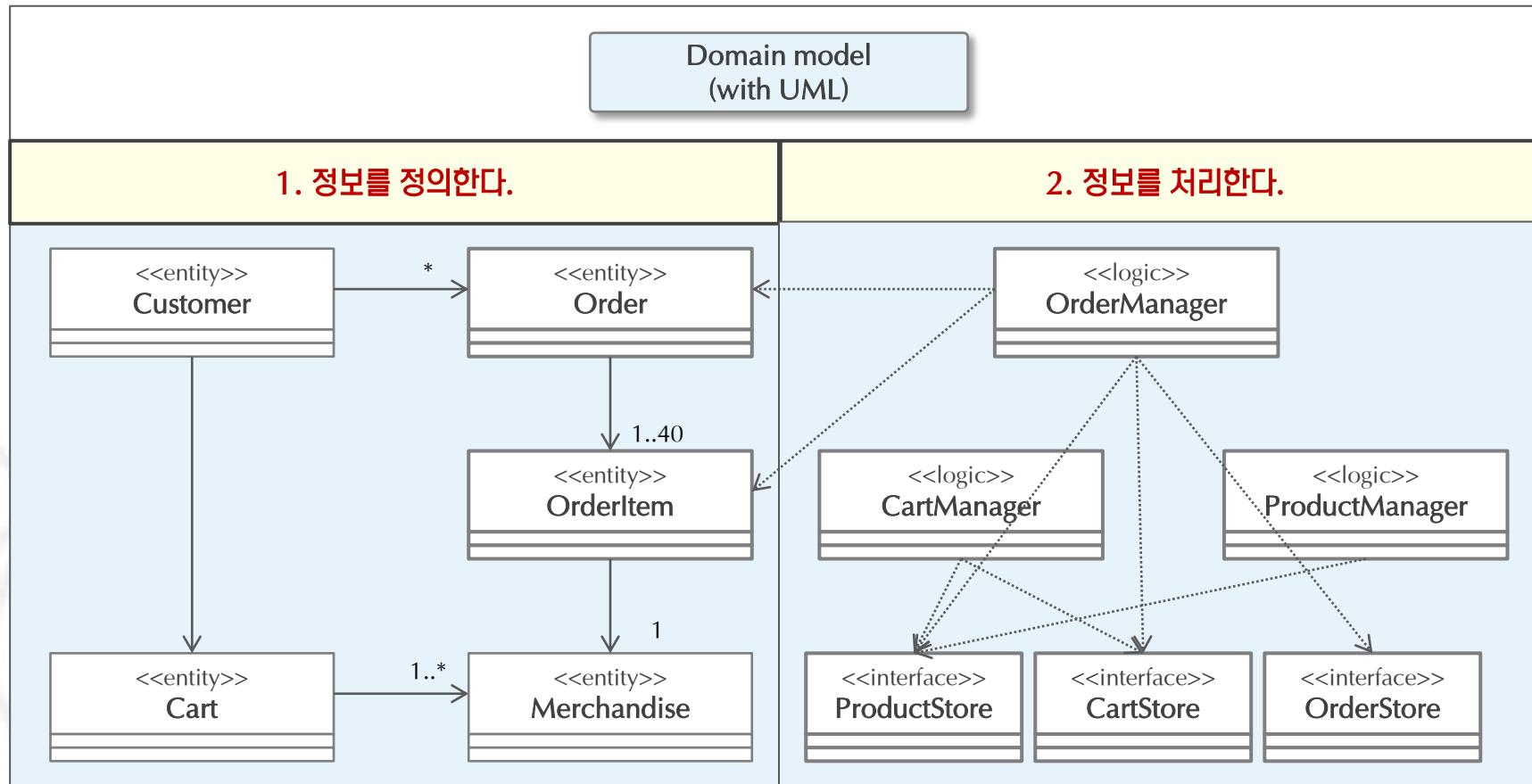
# 도메인 모델링 – 생각하는 과정

- ✓ 다양한 input 정보를 참조하여 생각하는 과정을 거치면 모델을 볼 수 있습니다.
- ✓ 하지만 이 모델 보다 더 중요한 것은 모델러의 머리 속에 질서정연하게 자리 잡은 도메인 개념입니다.
- ✓ 이제 모델러는 코딩을 할 수 있으며, 고객과의 대화를 통해 이해의 폭과 깊이를 더해 갈 수 있습니다.



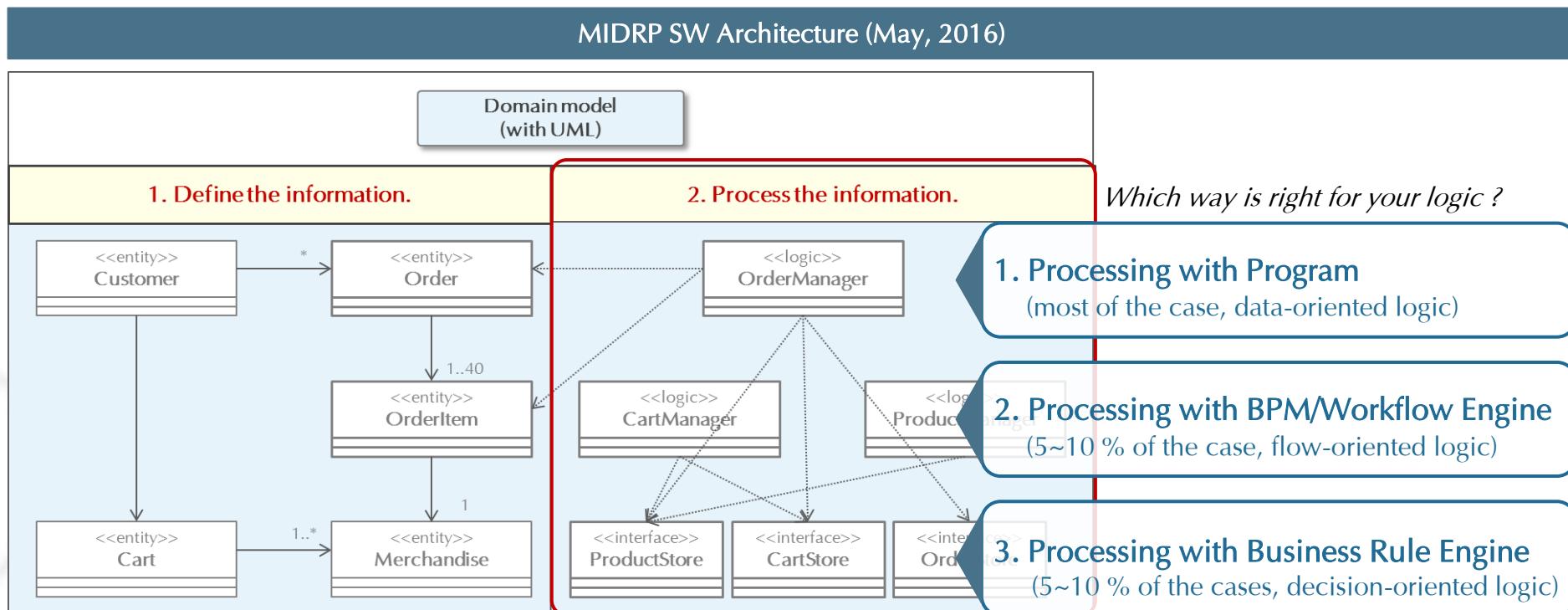
# 모델링 핵심 요소: 정의와 처리

- ✓ 비즈니스 컴포넌트를 구성하는 핵심 객체는 엔티티 객체와 로직 객체입니다.
- ✓ 엔티티 객체는 개념과 개념 간의 관계를 담고 있고, 로직 객체를 처리 절차를 담고 있습니다.
- ✓ 도메인 모델러는 도메인을 정확하게 이해하고, 그 내용을 모델로 표현하여야 합니다.



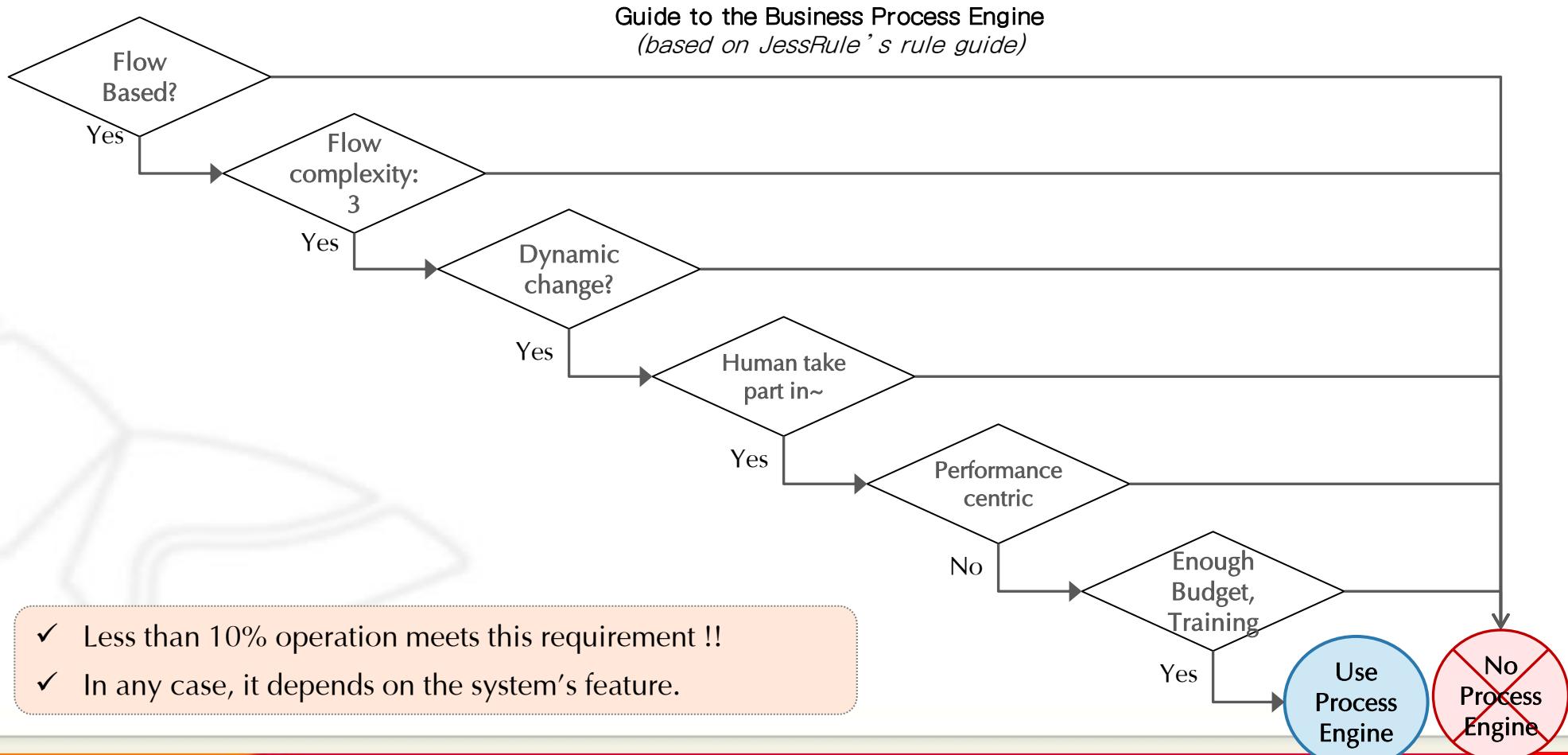
# 모델링 핵심 요소: 정의와 처리

- ✓ 로직 객체는 세 가지 처리 로직으로 분류할 수 있습니다. → 데이터 처리, 흐름 처리, 판단 처리
- ✓ 어떤 경우에는 흐름 처리나 판단 처리를 전문으로 하는 모듈이 필요할 수도 있습니다.
- ✓ 하지만, 어떤 경우든 프로그램으로 처리할 수 있습니다.



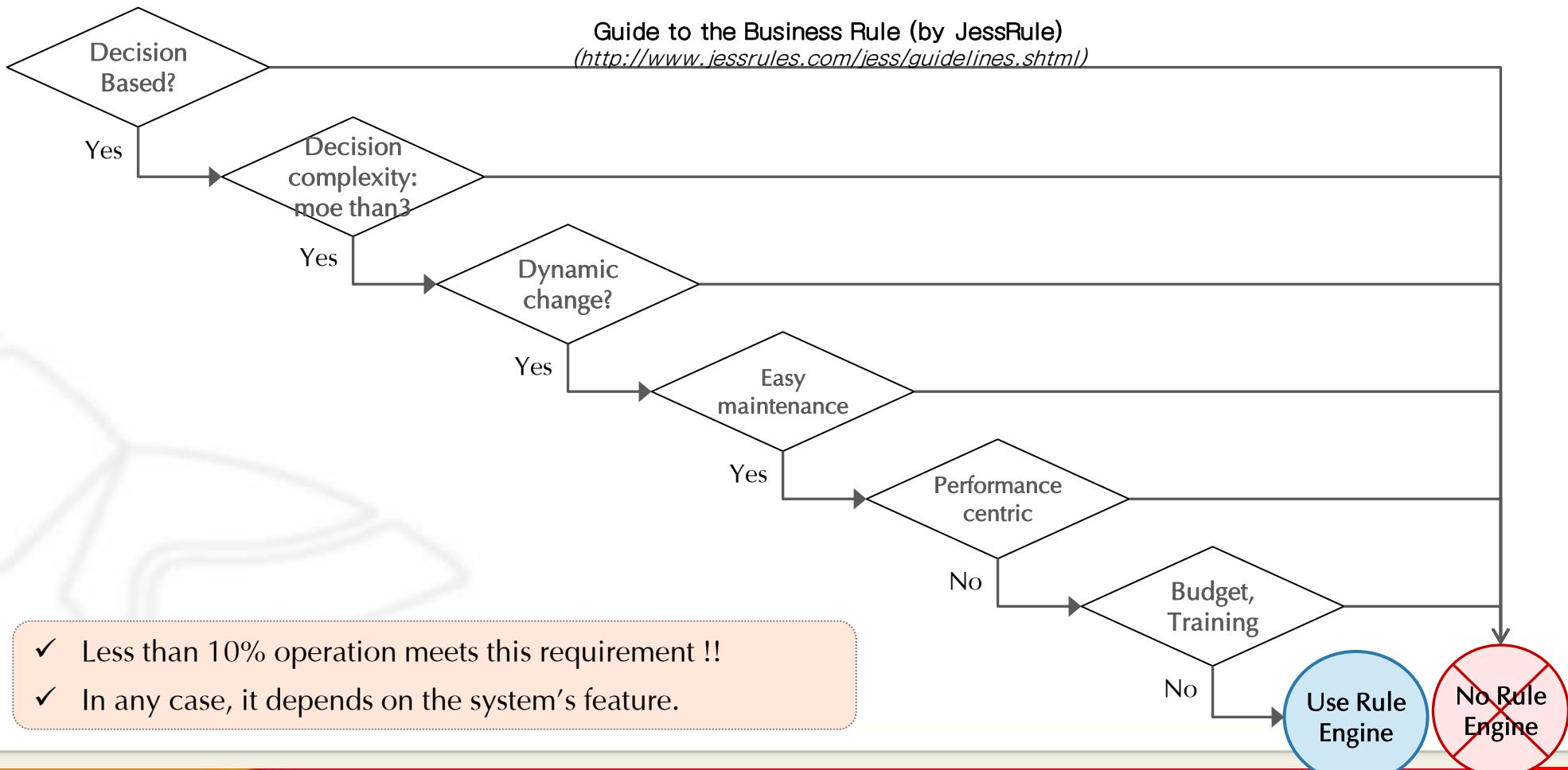
# 모델링 – 의사 결정: 프로세스 엔진

- ✓ 때로는 흐름이 복잡하거나 자주 변경하는 경우, 흐름 처리를 전문으로 하는 엔진 모듈을 사용할 수 있습니다.
- ✓ 프로그램으로 처리할 것인가 프로세스 엔진을 사용할 것인가에 대한 판단이 필요합니다.
- ✓ JessRule에서 제시한 룰 선택 가이드를 참조하여 프로세스 엔진 선택을 위한 의사결정 트리를 만들어 보았습니다.



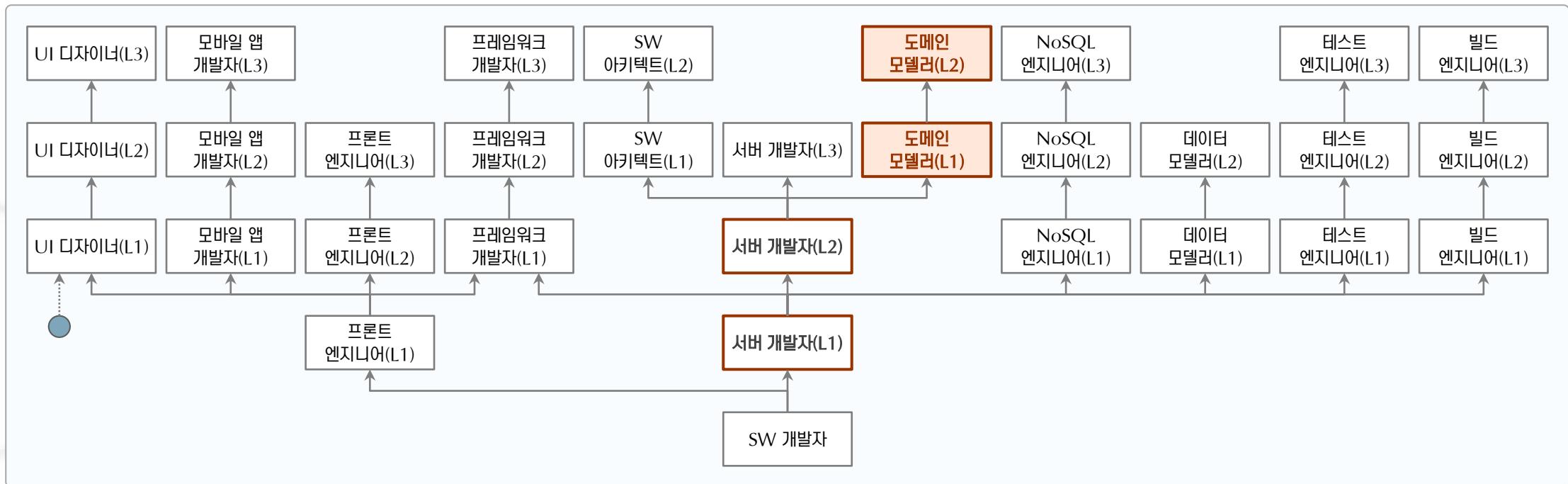
# 모델링 – 의사 결정: 룰엔진

- ✓ JessRule 에서 제시한 룰엔진 사용여부 판단을 위한 의사결정 트리는 아키텍트들이 자주 참조하는 자료입니다.
- ✓ 여섯 가지 질문에 대답을 하다보면 룰엔진을 사용할 지 프로그램으로 개발할지 여부를 결정을 할 수 있습니다.
- ✓ 시스템의 특성에 따라 결과는 다르겠지만, 보통의 시스템에서 룰 엔진이 필요한 경우는 10% 이내입니다.



# 역할 – 도메인 모델러

- ✓ SW 개발자 역할로부터 시작하여 SW 개발 영역 안에서 열 가지 서로 다른 역할을 가질 수 있습니다.
- ✓ 역할 간의 높낮이는 없으며, 역할 안에서 두 개 또는 세 개의 Level이 있습니다.
- ✓ 도메인 모델러는 아키텍트와 더불어 시스템 개발의 양축을 이끌어 가며, 컨텐츠 부분을 담당합니다.



# 역할 – 도메인 모델러

## 도메인 모델러

### 1. 책임(Responsibility)

기능 요구(functional requirement)를 명세하고, 이를 기반으로 도메인 모델링을 하고, 모델링 검증을 위해 PoC 프로토타입을 개발합니다. 레거시 개발 프로세스에서 정의한 요구사항 명세자, 시스템 분석가, 기능 설계자 세 가지 역할을 동시에 수행하는 역할입니다. 비즈니스 컴포넌트 기반으로 시스템을 개발할 경우, 아키텍처 팀에서 설계한 컴포넌트 구조에 도메인 모델을 결합하는 방식으로 비즈니스 컴포넌트 모델링을 수행합니다. 컴포넌트 모델링이 끝나면, 소스코드 생성을 통해 컴포넌트 소스 코드의 틀을 만들고, 컴포넌트 코드 별로 개발자를 배정한 후, 모델과 코드를 넘겨 주거나 도메인 로직은 직접 개발하기도 합니다.

### 2. 기술(Skills)

- ✓ 요구 사항 명세(유스케이스, 사용자 스토리, 기능 목록)
- ✓ 객체/컴포넌트 모델링
- ✓ 디자인 패턴
- ✓ 소통(communication)과 이해력
- ✓ 다양한 도메인에서 모델링 경험

### 3. 활동(Activities)

- ✓ 요구사항 명세(유스케이스 또는 사용자 스토리 또는 기능 목록 사용)
- ✓ 도메인 객체 모델링, 컴포넌트 모델링
- ✓ 도메인 객체 모델 검증을 위한 PoC 프로토타입 개발
- ✓ 비즈니스 컴포넌트 코딩
- ✓ 도메인 모델 검증을 위한 기능 테스트

### 4. 산출물(Work products)

- ✓ 모델(요구 사항 모델, 도메인 객체 모델)
- ✓ 소스코드(PoC 프로토타입 코드, 컴포넌트 코드, 단위 테스트 코드)
- ✓ 기능 테스트 케이스와 스크립트(테스트 엔지니어와 협업)

### 5. 추천 경로(Role path)

- ✓ 주요 경로: SW 개발자 → 서버 개발자(L1) → 서버 개발자(L2) → 도메인 모델러(L1,L2)
- ✓ 보조 경로: 프론트 엔지니어(L1), 데이터 모델러(L1), 모바일 앱 개발자(L1), 개발 리더(L1)
- ✓ 선택 경로: 빌드 엔지니어(L1)

### 6. 수준 정의(Level definition)

- ✓ 수준을 정의하는 기준은 두 개이며, 각 수준별 4년을 기준으로 합니다.
- ✓ L1 수준: 가장 빠른 경우는 서버 개발자 L1이 끝나는 6년수, 늦은 경우는 서버 개발자 L2가 끝나는 9년 수입니다. 이후 4년 정도의 도메인 모델러 역할 경험을 갖추고 L2로 올라갑니다.
- ✓ L2 수준: 10년수 이후 부터 시작하며, 탄탄한 모델링 지식과 다양한 모델링 경험을 갖추어야 합니다.

### 7. 역할 별칭(Name and alias)

- ✓ 도메인 모델러 역할은 Eric Evans가 저술한 Domain Driven Design 책이 소프트웨어 개발의 새로운 패러다임을 제시하면서 등장했습니다. 소프트웨어에서 기술 구조도 중요하지만 결국은 사용자의 필요에 꼭 맞는 기능을 제시하는 것이 더 중요합니다. 도메인에 대한 깊은 이해를 바탕으로 작성한 도메인 객체 모델(레거시 방법론에서 이야기 하는 분석/설계 모델)의 중요성을 강조합니다.
- ✓ 도메인 분석가, 도메인 설계자 등으로 불리는데, 분석과 설계의 경계가 애매하여 그냥 모델러라고 부릅니다.

### 8. 교육(Training)

- ✓ UML과 객체 모델링
- ✓ 기술 중립적인 컴포넌트 설계
- ✓ Domain Driven Design
- ✓ 요구사항 명세(유스케이스, 사용자 스토리, 기능 목록)

### 9. 참조(Reference)

- ✓ 도서: Domain Driven Design, Eric Evans, 2003
- ✓ 도서: Object-Oriented Analysis and Design with Applications, 2007, Grady Booch
- ✓ 도서: Model-driven Design using Business Patterns, 2006, Pavel Hruby

### 3. 객체 지향 설계 원칙

- ❖ Single Responsibility Principle 을 이해합니다.
- ❖ SRP가 프로그램에서 어떻게 사용하는지 실습합니다.
- ❖ 아주 간단한 예제를 통해서 SRP를 이해합니다.

#### 단일 책임 원칙(SRP): 구구단

- ✓ 구구단 1
- ✓ 구구단 2 – 책임 분리
- ✓ 구구단 3 – 추가 요구
- ✓ 구구단 4 – 당황스런 요구
- ✓ 구구단 5 – 여러 열로 출력하기(객체지향)
- ✓ 구구단 6 – 정방형으로 출력하기(객체지향)
- ✓ 구구단 7 – 지속적인 확장 실습
- ✓ 요약

# 구구단 1: 얼마나 많은 책임이 있는가?

- ✓ 구구단을 출력하는 프로그램을 작성해 봅니다. 아주 간단하여 하나의 클래스로도 충분합니다.
- ✓ 그런데, showTable() 메소드는 여러 가지 일을 하고 있습니다. 즉, 여러 가지 책임을 갖고 있습니다.
- ✓ 이 프로그램은 앞으로 발생 가능한 다양한 변경 요구에 “코드 변경”으로 대응할 수 밖에 없습니다.

Let's print out the multiplication tables.

```
-----  
2 times 1 = 2  
2 times 2 = 4  
2 times 3 = 6  
2 times 4 = 8  
2 times 5 = 10  
2 times 6 = 12  
2 times 7 = 14 |  
2 times 8 = 16  
2 times 9 = 18  
-----
```

```
3 times 1 = 3  
3 times 2 = 6  
3 times 3 = 9  
3 times 4 = 12  
3 times 5 = 15  
3 times 6 = 18  
3 times 7 = 21  
3 times 8 = 24  
3 times 9 = 27  
-----
```

```
public void showTable() {  
    //  
    System.out.println("Let's print out the multiplication tables.");  
    System.out.println("-----");  
  
    for (int leftNumber = 2; leftNumber<=9; leftNumber++) {  
        for (int rightNumber = 1; rightNumber <=9; rightNumber++) {  
            System.out.println(  
                String.format(" %2d times %d = %2d ",  
                            leftNumber,  
                            rightNumber,  
                            (leftNumber*rightNumber)));  
        }  
        System.out.println("-----");  
    }  
}
```

## Times Table

- + [main\(String\[\]\): void](#)
- + [TimesTable\(\)](#)
- + [showTable\(\): void](#)

## 구구단 2: 책임분리

- ✓ 전체 포맷, 항목 구성, 곱셈, 세 가지 책임으로 분리하여 봅니다.
- ✓ 책임 분리의 첫 번째는 메소드 분리입니다. showTable() → buildTimesItem(), multiply()
- ✓ 변경을 해야 할 때, 세 곳 중에 한 곳만 변경이 되어야 합니다. 동시에 여러 곳을 변경해야 한다면, 책임 분리를 제대로 하지 못한 것입니다.

코딩 실습 !!

```
public void showTable() {
    //
    System.out.println("Let's print out the multiplication tables.");
    System.out.println("-----");

    String itemFormat = " %d times %d = %2d ";

    for (int leftNumber = 2; leftNumber<=9; leftNumber++) {
        for (int rightNumber = 1; rightNumber <=9; rightNumber++) {
            System.out.println(buildTimesItem(itemFormat, leftNumber, rightNumber));
        }
        System.out.println("-----");
    }
}

public String buildTimesItem(String itemFormat, int leftNumber, int rightNumber) {
    //
    return String.format(itemFormat,
        leftNumber,
        rightNumber,
        multiply(leftNumber, rightNumber));
}

public static int multiply(int left, int right) {
    //
    return left * right;
}
```

# 구구단 3:추가 요구 – 여러 열로 출력하기

- ✓ 일상에서 흔히 만나는 좁절 상황 !
- ✓ 개발자 입장: 진작 이야기를 해 주었으면 다시 작업하는 일이 없었을 텐데...
- ✓ 고객사 입장: 우린 시장을 읽고 그 변화와 흐름을 비즈니스에 반영할 뿐... (이 정도도 예상 못했나요???)

코딩 실습 !!

```
-----  
2 times 1 = 2  
2 times 2 = 4  
2 times 3 = 6  
2 times 4 = 8  
2 times 5 = 10  
2 times 6 = 12  
2 times 7 = 14 |  
2 times 8 = 16  
2 times 9 = 18  
-----  
3 times 1 = 3  
3 times 2 = 6  
3 times 3 = 9  
3 times 4 = 12  
3 times 5 = 15  
3 times 6 = 18  
3 times 7 = 21  
3 times 8 = 24  
3 times 9 = 27  
-----
```



2 * 1 = 2	3 * 1 = 3	4 * 1 = 4	5 * 1 = 5
2 * 2 = 4	3 * 2 = 6	4 * 2 = 8	5 * 2 = 10
2 * 3 = 6	3 * 3 = 9	4 * 3 = 12	5 * 3 = 15
2 * 4 = 8	3 * 4 = 12	4 * 4 = 16	5 * 4 = 20
2 * 5 = 10	3 * 5 = 15	4 * 5 = 20	5 * 5 = 25
2 * 6 = 12	3 * 6 = 18	4 * 6 = 24	5 * 6 = 30
2 * 7 = 14	3 * 7 = 21	4 * 7 = 28	5 * 7 = 35
2 * 8 = 16	3 * 8 = 24	4 * 8 = 32	5 * 8 = 40
2 * 9 = 18	3 * 9 = 27	4 * 9 = 36	5 * 9 = 45
-----	-----	-----	-----
6 * 1 = 6	7 * 1 = 7	8 * 1 = 8	9 * 1 = 9
6 * 2 = 12	7 * 2 = 14	8 * 2 = 16	9 * 2 = 18
6 * 3 = 18	7 * 3 = 21	8 * 3 = 24	9 * 3 = 27
6 * 4 = 24	7 * 4 = 28	8 * 4 = 32	9 * 4 = 36
6 * 5 = 30	7 * 5 = 35	8 * 5 = 40	9 * 5 = 45
6 * 6 = 36	7 * 6 = 42	8 * 6 = 48	9 * 6 = 54
6 * 7 = 42	7 * 7 = 49	8 * 7 = 56	9 * 7 = 63
6 * 8 = 48	7 * 8 = 56	8 * 8 = 64	9 * 8 = 72
6 * 9 = 54	7 * 9 = 63	8 * 9 = 72	9 * 9 = 81



# 구구단 3:추가 요구 – 절차 지향적 해결

✓ 잠시 좌절 후, 이런 것 쯤이야 쉽게 해결할 수 있겠지요. 그런데 제약 조건이 좀 고약합니다.

```
public void showTable(int columnCount) {
    //
    if (columnCount > MaxTimes) {
        throw new RuntimeException(
            String.format("The column count should be less than %d, but it's %d. --> ", MaxTimes, columnCount));
    }

    int leftNumber = 2;
    while(true) {
        for (int rightNumber = 1; rightNumber <=MaxTimes; rightNumber++) {
            for(int offset=0; offset<columnCount; offset++) {
                if (leftNumber+offset > MaxTimes) {
                    break;
                }
                System.out.print(buildTimesItem(leftNumber+offset, rightNumber));
                System.out.print(" ");
            }
            System.out.println(" ");
        }
        System.out.println(" ");

        leftNumber += columnCount;
        if (leftNumber > MaxTimes) {
            break;
        }
    }
}

public String buildTimesItem(int leftNumber, int rightNumber) {
    //
    String itemFormat = this.getItemFormatStr(leftNumber);

    return String.format(itemFormat,
        leftNumber,
        rightNumber,
        multiply(leftNumber, rightNumber));
}
```



# 구구단 3:추가 요구 – 절차 지향적 해결과 문제점

- ✓ 문제점1 : Complexity 증가 → 7, 116라인으로 증가
- ✓ 더 큰 문제점 2: 보여 주기에 집중하면서, 컨텍스트 속의 구구단의 개념이 서서히 사라지고 있습니다.
- ✓ 즉, 프로그램은 구구단의 의미를 떠나서 숫자를 여러 컬럼으로 보여 주는 작업에 집중하고 있습니다.
- ✓ 컨텍스트가 프로그램 코드의 흐름과 판단 속으로 매몰되어 가고 있습니다.

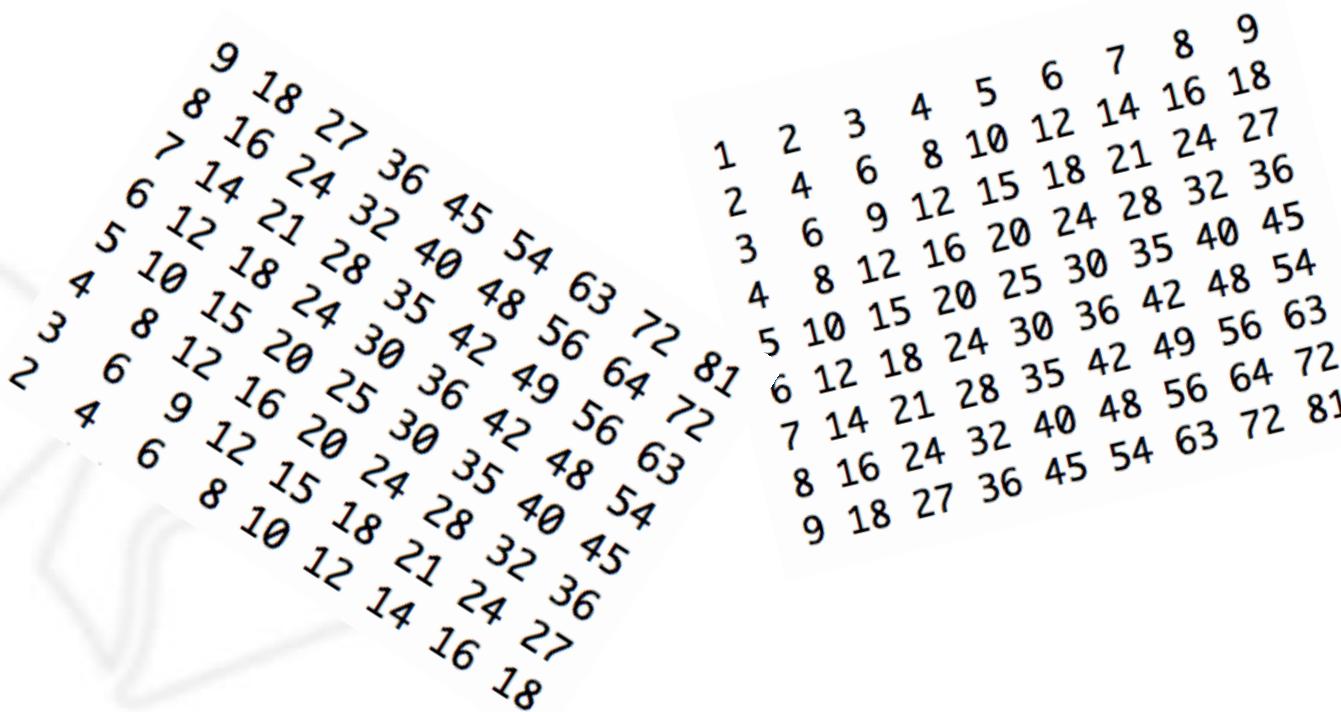
```
public void showTable(int columnCount) {  
    //  
    if (columnCount > MaxTimes) {  
        throw new RuntimeException(  
            String.format("The column count should be less than %d, but it's %d. --> ", MaxTimes, columnCount));  
    }  
  
    int leftNumber = 2;  
    while(true) {  
        for (int rightNumber = 1; rightNumber <=MaxTimes; rightNumber++) {  
            for(int offset=0; offset<columnCount; offset++) {  
                if (leftNumber+offset > MaxTimes) {  
                    break;  
                }  
                System.out.print(buildTimesItem(leftNumber+offset, rightNumber));  
                System.out.print(" ");  
            }  
            System.out.println(" ");  
        }  
        System.out.println(" ");  
  
        leftNumber += columnCount;  
        if (leftNumber > MaxTimes) {  
            break;  
        }  
    }  
}  
  
public String buildTimesItem(int leftNumber, int rightNumber) {  
    //  
    String itemFormat = this.getItemFormatStr(leftNumber);  
  
    return String.format(itemFormat,  
        leftNumber,  
        rightNumber,  
        multiply(leftNumber, rightNumber));  
}
```



## 구구단 4:당황스런 요구 – 정방형으로 출력하기

- ✓ 여러 열로 출력하는 구구단을 출력하고 나니 다음과 같은 추가 요구 사항이 기다리고 있습니다.
- ✓ 구구단을 숫자 간의 관계로 파악하기 좋은 형태로 표현하는 것입니다.
- ✓ 믿기 어렵겠지만, 이것도 구구단의 한 형태이므로, 개발자에게 추가되는 요구 사항은 타당합니다.
- ✓ 구구단 개념을 바탕으로 보면 유사한 요구이지만, 표현 관점에서 보면 서로 다른 요구사항입니다.

코딩 실습 !!



이것도 구구단 ???

# 구구단 4: 당황스런 요구 – 절차 지향적 해결과 문제점

- ✓ 잠시 고민 후에 별도의 메소드로 구현합니다.
- ✓ 문제점1 : Complexity 증가 → 12, 190라인으로 증가
- ✓ 더 큰 문제점 2: showLineTable() 메소드와 showBoxTable() 메소드는 서로 연관성이 없는 다른 메소드로써 모두 보여 주기에 집중하고 있습니다.
- ✓ 이 코드에서 구구단 개념은 더 이상 존재하지 않고 화면에 나타난 결과 값으로만 존재하고 있습니다.

```
public void showBoxTable(SortOrder sortOrder) {  
    //  
    int leftNumber = 2;  
  
    if(SortOrder.Descending.equals(sortOrder)) {  
        leftNumber = MaxTimes;  
    }  
  
    System.out.println();  
  
    while(true) {  
        for (int rightNumber = 1; rightNumber <=MaxTimes; rightNumber++) {  
            for(int offset=0; offset<columnCount; offset++) {  
                int newLeftNumber = leftNumber + offset;  
                if(SortOrder.Descending.equals(sortOrder)) {  
                    newLeftNumber = leftNumber - offset;  
                }  
                if (newLeftNumber < 2 || newLeftNumber > MaxTimes) {  
                    break;  
                }  
                System.out.print(buildTimesItem(newLeftNumber, rightNumber));  
                System.out.print(" ");  
            }  
            System.out.println();  
        }  
        System.out.println();  
    }  
}  
  
public void showLineTable(SortOrder sortOrder) {  
    //  
    int leftNumber = 2;  
    int addNumber = 1;  
  
    if(SortOrder.Descending.equals(sortOrder)) {  
        leftNumber = MaxTimes;  
        addNumber = -1;  
    }  
  
    System.out.println();  
  
    while(true) {  
        for(int rightNumber = 1; rightNumber <= MaxTimes; rightNumber++) {  
            System.out.print(getColumn(leftNumber, rightNumber));  
        }  
    }  
}
```

## 구구단 5: 단일 책임의 원칙(SRP)

- ✓ 절차 지향 문제 해결 방법에서 여러분이 개발한 메소드를 자세히 들여다 보세요.
- ✓ 계산, 수식 표현, 정렬 등 여러 가지 일을 동시에 하고 있음을 알 수 있습니다.
- ✓ 이러한 문제 해결 방식은 또 다른 변경 요구가 계산, 수식, 정렬 등 어느 곳에서 오던지 많은 변경이 필요합니다.
- ✓ 한 가지 일을 잘 하는 여러 전문가 객체를 이용하여 문제를 풀어 가야 합니다. → SRP

한 클래스는 하나 그리고 단 하나의 변경 이유를 가져야 한다. 이것은 클래스는 오직 하나의 일만을 가져야 함을 의미한다.

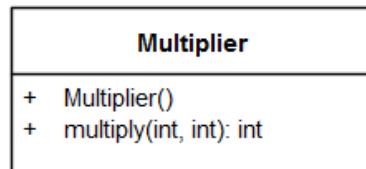
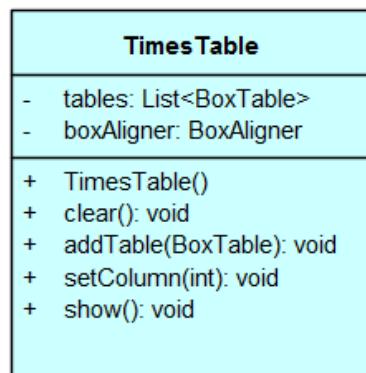
A class should have one and only one reason to change, meaning that a class should have only one job.

# 구구단 5: 여러 열로 출력하기(객체 지향)

- ✓ 개념을 찾고 책임을 찾아 봅니다. → 각 단 구성, 각 단의 스타일 적용, 계산, 전체 단 구성 조정 등
- ✓ 이제 TimesTable은 각 역할 담당 객체를 거느리고 구구단 출력을 합니다.
- ✓ 여기까지 왔으면, 사용자의 요구 사항이 어디로 뛰어 갈 수 있는지 대략 예측을 할 수 있어야 합니다.

```

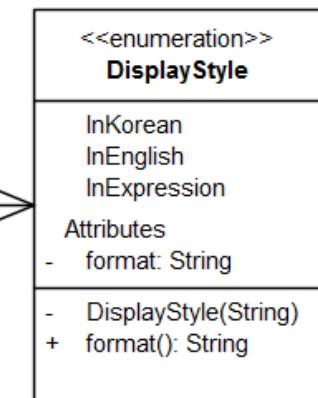
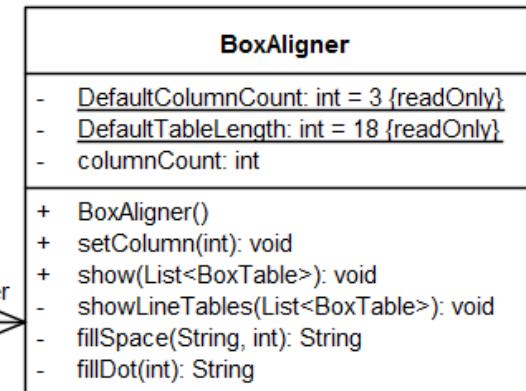
-----
2 times 1 = 2      3 * 1 = 3      4 * 1 = 4
2 times 2 = 4      3 * 2 = 6      4 * 2 = 8
2 times 3 = 6      3 * 3 = 9      4 * 3 = 12
2 times 4 = 8      3 * 4 = 12     4 * 4 = 16
2 times 5 = 10     3 * 5 = 15     4 * 5 = 20
2 times 6 = 12     3 * 6 = 18     4 * 6 = 24
2 times 7 = 14 |   3 * 7 = 21     4 * 7 = 28
2 times 8 = 16     3 * 8 = 24     4 * 8 = 32
2 times 9 = 18     3 * 9 = 27     4 * 9 = 36
-----
3 times 1 = 3      6 * 1 = 6      7 * 1 = 7      8 * 1 = 8
3 times 2 = 6      6 * 2 = 12     7 * 2 = 14     8 * 2 = 16
3 times 3 = 9      6 * 3 = 18     7 * 3 = 21     8 * 3 = 24
3 times 4 = 12     6 * 4 = 24     7 * 4 = 28     8 * 4 = 32
3 times 5 = 15     6 * 5 = 30     7 * 5 = 35     8 * 5 = 40
3 times 6 = 18     6 * 6 = 36     7 * 6 = 42     8 * 6 = 48
3 times 7 = 21     6 * 7 = 42     7 * 7 = 49     8 * 7 = 56
3 times 8 = 24     6 * 8 = 48     7 * 8 = 56     8 * 8 = 64
3 times 9 = 27     6 * 9 = 54     7 * 9 = 63     8 * 9 = 72
-----
```



-boxAligner

-tables  
0..\*

-multiplier



# 구구단 5: 여러 열로 출력하기(객체 지향)

- ✓ 이제 BoxTable, BoxAligner 등 객체의 책임을 코드로 구현합니다.
- ✓ 코딩을 하는 순간에는 해당 객체의 역할과 책임에만 집중할 수 있으며, 그 과정에서 전체 그림에서 발견하지 못한 작지만 중요한 기능들을 발견할 수도 있습니다.

```
public BoxTable(int leftNumber, DisplayStyle style) {  
    //  
    this(leftNumber);  
    setFormat(style);  
}  
  
public void setFormat(DisplayStyle style) {  
    //  
    this.style = style;  
}  
  
public void show() {  
    //  
    for (int right=0; right<=9; right++) {  
        System.out.println(getFormattedLine(right));  
    }  
}  
  
public String getFormattedLine(int rightNumber) {  
    //  
    return String.format(style.format(), leftNumber, rightNumber, multiply(rightNumber));  
}  
  
private int multiply(int rightNumber) {  
    //  
    return multiplier.multiply(leftNumber, rightNumber);  
}
```

```
public void show(List<BoxTable> tables) {  
    //  
    List<BoxTable> lineTables = new ArrayList<BoxTable>();  
  
    for (BoxTable table : tables) {  
        lineTables.add(table);  
        if (lineTables.size() == columnCount) {  
            showLineTables(lineTables);  
            lineTables.clear();  
        }  
    }  
  
    if (lineTables.size() > 0) {  
        showLineTables(lineTables);  
    }  
}  
  
private void showLineTables(List<BoxTable> tables) {  
    //  
    int tableSize = tables.size();  
    int lineLength = 0;  
  
    for (int right = 1; right<=9; right++) {  
        StringBuilder builder = new StringBuilder();  
        for (int i=0; i            String line = tables.get(i).getFormattedLine(right);  
            builder.append(fillSpace(line, DefaultTableLength));  
        }  
        lineLength = builder.length();  
        System.out.println(builder.toString());  
    }  
  
    System.out.println(fillDot(lineLength));  
}
```

## 구구단 6:정방형으로 출력하기(객체지향)

- ✓ 구구단의 본질이 무엇인가를 생각해 보면, 정방형으로 표현하는 것도 충분히 의미가 있어 보입니다.
- ✓ 이러한 표현 방식은 구구단에서 leftNumber와 rightNumber의 관계를 잘 보여줍니다.
- ✓ 예상치 못한 표현 방식이지만, 구구단의 본질이라는 관점에서 충분히 가능한 표현 방법입니다.
- ✓ 앞에서 고민하여 개념의 관계 구조를 만들었는데, 이제 확장을 위한 준비를 할 차례입니다.

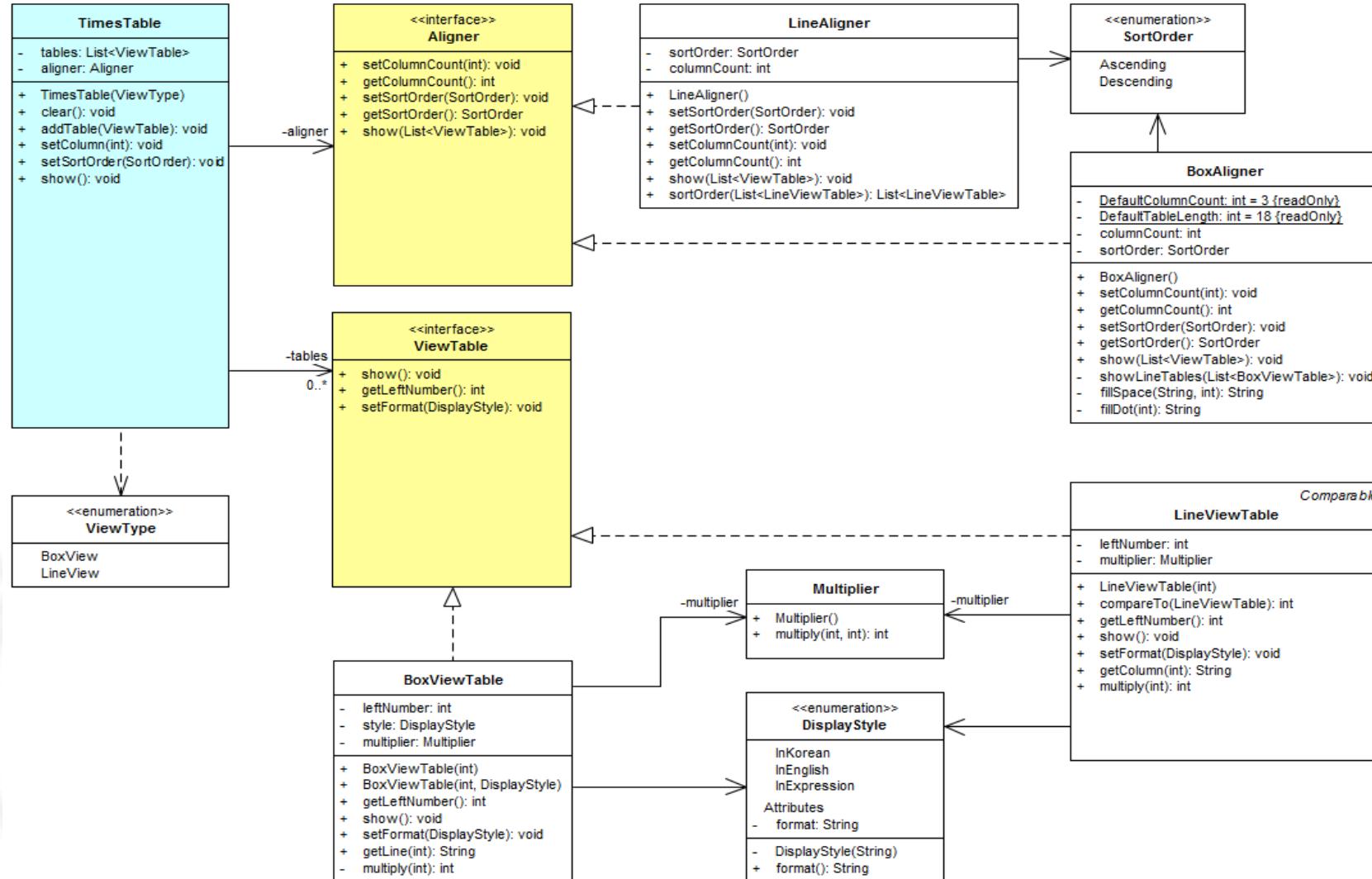
코딩 실습 !!



이것은 당연히  
구구단 !!!

# 구구단 6: 정방형으로 출력하기(객체지향)

✓ 이제 TimesTable은 다양한 Aligner와 ViewTable을 수용할 준비가 되어 있습니다.



## 구구단 6:정방형으로 출력하기(객체지향)

- ✓ 이제 TimesTable 클래스를 이용하여 다양한 표현을 간단하게 할 수 있습니다. 개발자와 TimesTable 간의 협업을 상상 할 수 있습니다.

```
public static void main(String[] args) {  
    //  
    showLineTableAscendingDemo();  
    showTimesTableObjectDemo();  
    showLineTableDescendingDemo();  
    showTimesTableProcedureDemo();  
}  
  
private static void showLineTableAscendingDemo() {  
    //  
    TimesTable timesTable = new TimesTable(ViewType.LineView);  
    timesTable.createDefaultViewTables();  
    timesTable.setSortOrder(SortOrder.Ascending);  
    timesTable.show();  
}  
  
private static void showTimesTableObjectDemo() {  
    //  
    TimesTable timesTable = new TimesTable(ViewType.BoxView);  
  
    // User defined tables  
    timesTable.addTable(new BoxViewTable(3, DisplayStyle.InKorean));  
    timesTable.addTable(new BoxViewTable(5, DisplayStyle.InEnglish));  
    timesTable.addTable(new BoxViewTable(7, DisplayStyle.InExpression));  
    timesTable.addTable(new BoxViewTable(9));  
  
    timesTable.setColumn(3);  
    timesTable.show();  
    timesTable.clear();  
  
    // Default times tables  
    timesTable.createDefaultViewTables();  
    timesTable.setColumn(5);  
    timesTable.show();  
}
```

# 구구단 7: 지속적인 확장 실습

- ✓ 이런 구구단은 어떨까요?
- ✓ 구구단을 처음 외우는 학생이나, 외국 인들에게는 유용하겠죠?
- ✓ 앞의 구구단 애플리케이션을 바탕으로 확장하면 됩니다.
- ✓ 새로운 객체가 등장할 수도 있고, 기존의 객체를 확장할 수 있습니다.
- ✓ 이제는 기존 코드 변경은 거의 없이, 확장으로 문제를 풀 수 있습니다.
- ✓ 여러분의 선택은?

$2 \times 1 = 2$	이 일은 이	$3 \times 1 = 3$	삼 일은 삼
$2 \times 2 = 4$	이 이는 사	$3 \times 2 = 6$	삼 이는 육
$2 \times 3 = 6$	이 삼은 육	$3 \times 3 = 9$	삼 삼은 구
$2 \times 4 = 8$	이 사는 팔	$3 \times 4 = 12$	삼 사 십이
$2 \times 5 = 10$	이 오는 십	$3 \times 5 = 15$	삼 오 십오
$2 \times 6 = 12$	이 육 십이	$3 \times 6 = 18$	삼 육 십팔
$2 \times 7 = 14$	이 칠은 십사	$3 \times 7 = 21$	삼 칠 이십일
$2 \times 8 = 16$	이 팔 십육	$3 \times 8 = 24$	삼 팔 이십사
$2 \times 9 = 18$	이 구는 십팔	$3 \times 9 = 27$	삼 구 이십칠

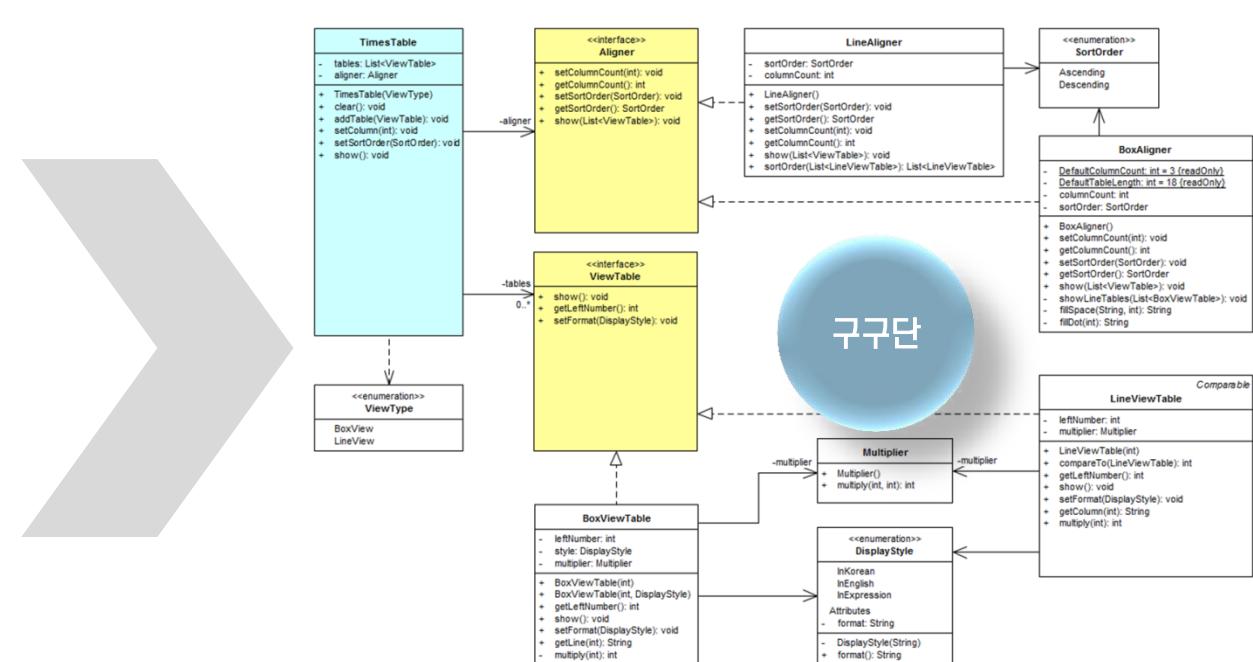
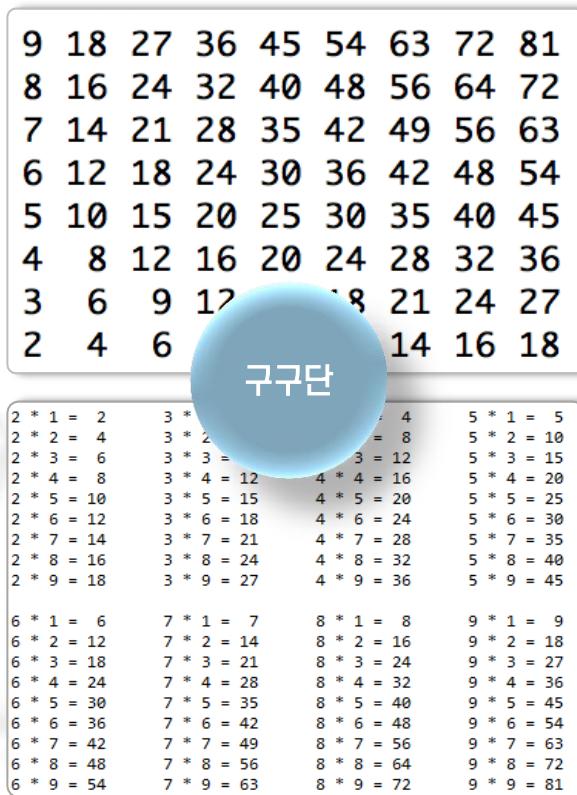
$4 \times 1 = 4$	사 일은 사	$5 \times 1 = 5$	오 일은 오
$4 \times 2 = 8$	사 이는 팔	$5 \times 2 = 10$	오 이는 십
$4 \times 3 = 12$	사 삼 십이	$5 \times 3 = 15$	오 삼 십오
$4 \times 4 = 16$	사 사 십육	$5 \times 4 = 20$	오 사 이십
$4 \times 5 = 20$	사 오 이십	$5 \times 5 = 25$	오 오 이십오
$4 \times 6 = 24$	사 육 이십사	$5 \times 6 = 30$	오 육 삼십
$4 \times 7 = 28$	사 칠 이십팔	$5 \times 7 = 35$	오 칠 삼십오
$4 \times 8 = 32$	사 팔 삼십이	$5 \times 8 = 40$	오 팔 사십
$4 \times 9 = 36$	사 구 삼십육	$5 \times 9 = 45$	오 구 사십오

$6 \times 1 = 6$	육 일은 육	$7 \times 1 = 7$	칠 일은 칠
$6 \times 2 = 12$	육 이 십이	$7 \times 2 = 14$	칠 이 십사
$6 \times 3 = 18$	육 삼 십팔	$7 \times 3 = 21$	칠 삼 이십일
$6 \times 4 = 24$	육 사 이십사	$7 \times 4 = 28$	칠 사 이십팔
$6 \times 5 = 30$	육 오 삼십	$7 \times 5 = 35$	칠 오 삼십오
$6 \times 6 = 36$	육 육 삼십육	$7 \times 6 = 42$	칠 육 사십이
$6 \times 7 = 42$	육 칠 사십이	$7 \times 7 = 49$	칠 칠 사십구
$6 \times 8 = 48$	육 팔 사십팔	$7 \times 8 = 56$	칠 팔 오십육
$6 \times 9 = 54$	육 구 오십사	$7 \times 9 = 63$	칠 구 육십삼

$8 \times 1 = 8$	팔 일은 팔	$9 \times 1 = 9$	구 일은 구
$8 \times 2 = 16$	팔 이 십육	$9 \times 2 = 18$	구 이 십팔
$8 \times 3 = 24$	팔 삼 이십사	$9 \times 3 = 27$	구 삼 이십칠
$8 \times 4 = 32$	팔 사 삼십이	$9 \times 4 = 36$	구 사 삼십육
$8 \times 5 = 40$	팔 오 사십	$9 \times 5 = 45$	구 오 사십오
$8 \times 6 = 48$	팔 육 사십팔	$9 \times 6 = 54$	구 육 오십사
$8 \times 7 = 56$	팔 칠 오십육	$9 \times 7 = 63$	구 칠 육십사
$8 \times 8 = 64$	팔 팔 육십사	$9 \times 8 = 72$	구 팔 칠십이
$8 \times 9 = 72$	팔 구 칠십이	$9 \times 9 = 81$	구 구 팔십일

# 요약

- ✓ 구구단의 의미를 유지하면서 변경 요구에 대응하는 것이 중요합니다. 언어 변환이지 개념 변환이 아닙니다.
- ✓ 추가 요구사항을 “변경”이 아닌 “확장”으로 대응할 수 있다면, 그것은 훌륭한 객체지향 프로그램입니다.
- ✓ 구구단에서 모델은 의미가 매우 약하여 BoxViewTable이나 LineViewTable과 결합한 상태입니다. 이것을 분리하여 개발한다면 어떤 모습일까요?



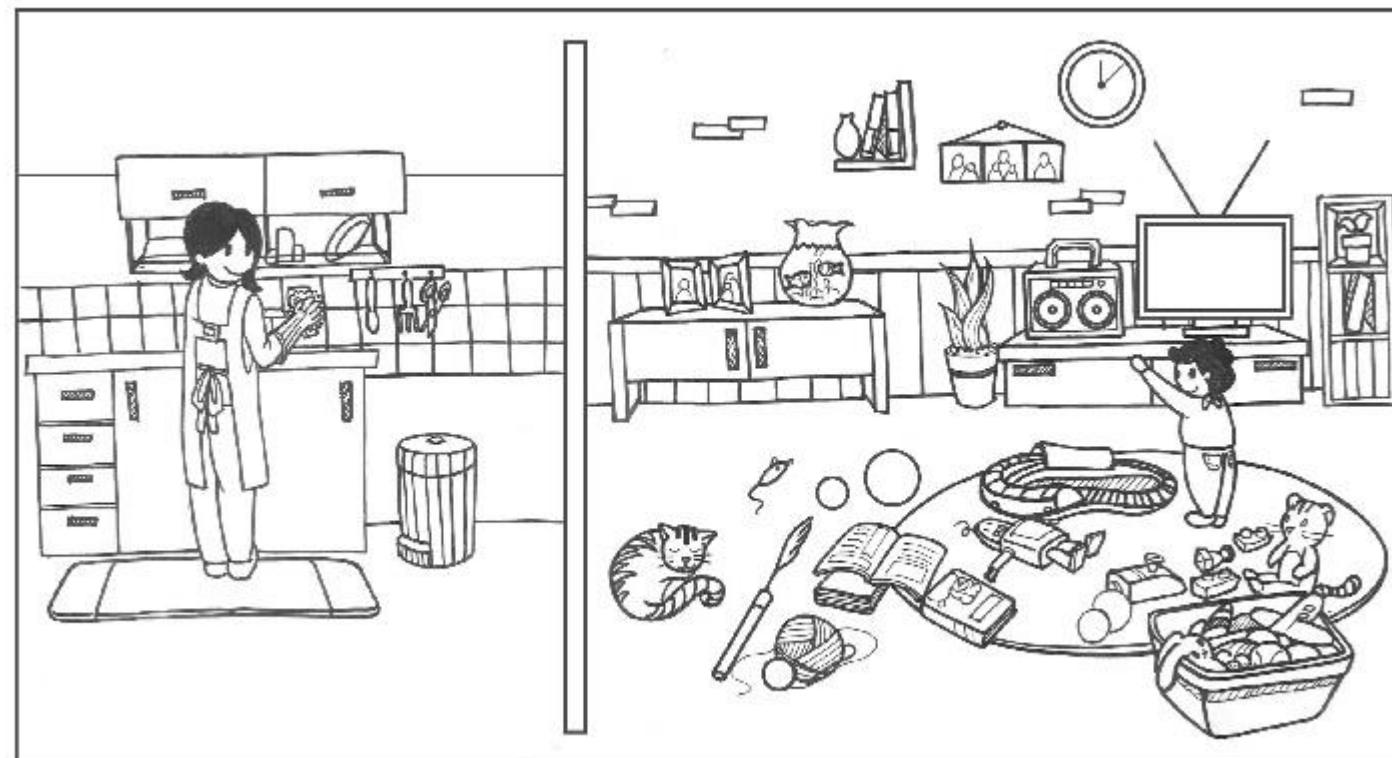
## 4. 협업(collaboration)

- ❖ 실세계에 존재하는 객체를 찾아 내고 협업을 하게 합니다.
- ❖ 시나리오에 참여하는 각 객체의 역할과 책임을 정의합니다.
- ❖ 보이지 않은 객체를 찾아내고, 드러나지 않던 역할과 책임을 부여합니다.
- ❖ 그 결과를 프로그램으로 표현합니다.

- ✓ 현장 스케치
- ✓ 시나리오
- ✓ 모델
- ✓ 프로그래밍
- ✓ 요약

# 라디오 – 현장 스케치

- ✓ 따뜻한 봄날 오후, 엄마는 부엌에서 점심 설거지를 하고 있습니다.
- ✓ 민수는 거실에서 혼자 기차 놀이를 하고 있습니다.
- ✓ 엄마는 갑자리 라디오가 듣고 싶어졌는데, 고무장갑을 끼고 있어서 라디오 켜기가 불편합니다.
- ✓ 민수에게 켜 달라고 해야겠지요?



# 라디오 - 시나리오

- ✓ 실세계에서 일어나는 일은, 일상 언어(natural language)를 이용하여 다양한 형식으로 표현할 수 있습니다.
- ✓ 글의 형식은 대화문, 서술문, 실용문, 설명문, 메모, 편지, 안내문, 광고문, 등으로 다양할 수 있습니다.
- ✓ 사건이나 상황을 정확하고 빨리 전달하려 할 경우 시나리오 형식을 사용합니다. ← 유스케이스, 사용자 스토리
- ✓ 이번 모듈에서 객체지향 프로그램을 위해 실습할 내용은 다음 시나리오입니다.

[엄마는 부엌에서 설거지를 하고 있고, 민수는 거실에서 놀고 있다. TV 옆에 티볼리 라디오가 놓여 있다.  
엄마는 라디오를 듣고 싶지만 젖은 고무장갑을 끼고 있어서 켤 수가 없다. 민수의 도움을 받기로 한다.]

엄마: 민수야 라디오 켤 수 있니?

민수: (음~ 내가 다섯 살이니까 켤 수 있겠지) 네, 켤 수 있어요. 엄마.

엄마: 그래? 그럼 라디오 좀 켜줄래?

민수: 네. (라디오를 켠다.)

라디오: 뉴스를 말씀드리겠습니다..

엄마: 고마워, 그런데 소리가 작은 것 같네. 소리 좀 높여줄래?

민수: 네. 엄마. (소리를 한 단 높인다.) 높였어요.

라디오: 뉴스를 말씀드리겠습니다...

엄마: 흠... 그런데, 아직도 소리가 좀 작아요. 조금 더 높여 주세요.

민수: 네. (다시 소리를 높인다.) 높였어요.

라디오: 뉴스를 말씀드리겠습니다..

엄마: 민수야 고마워. 참 잘 들리는구나.

# 라디오 – 시나리오

- ✓ “라디오” 시나리오는 자연 언어로 표현한 대화체 문장입니다. 이것을 프로그래밍으로 표현할 수 있을까요?
- ✓ 프로그램으로 표현하고, 실세계에서 엄마와 아이가 대화하듯이 실행을 할 수 있을까요?
- ✓ 물론, 프로그램 속의 아이와 엄마는 자신들이 하고 있는 것을 아래와 같이 열심히 출력해 주어야 겠지요.
- ✓ 자연 언어 → UML 모델링 언어 → Java 프로그래밍 언어라는 흐름을 지나서 아래와 같은 결과를 보여 주세요.

코딩 실습 →

Menu

.....  
0. Program exit

1. 일하면서 라디오 켜기
  2. 라디오 소리 조정
- .....

Select number: 1

<KimPD:Director> Yeongmi씨, 라디오 들으면서 일하실래요?

<Yeongmi:Mom> 그럴까요, 라디오를 듣겠습니다.

<Yeongmi:Mom> 민수야, 라디오 켤 수 있니?

<Minsoo:Child> 라디오를 켤 수 있느냐구요?

<Minsoo:Child> 예, 켤 수 있어요.

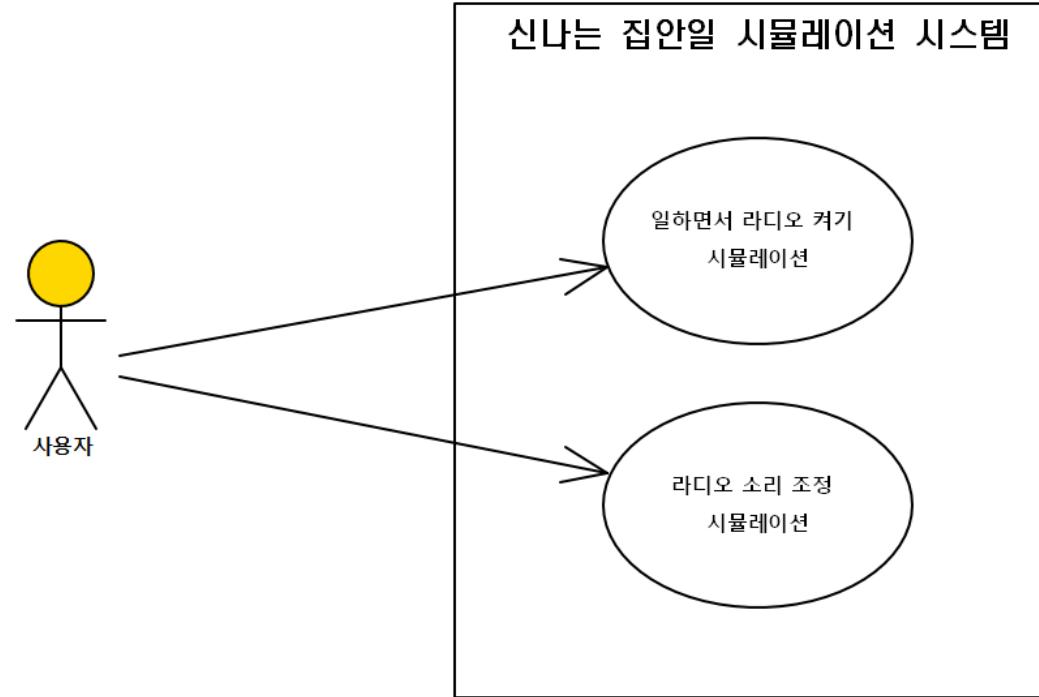
<Yeongmi:Mom> 그래, 그럼 라디오 좀 켜 줄래?

<Minsoo:Child> 예, 라디오 켤께요.

<Tivoli:Radio> [볼륨:1] 아, 아, 오늘의 뉴스를 말씀드리겠습니다...

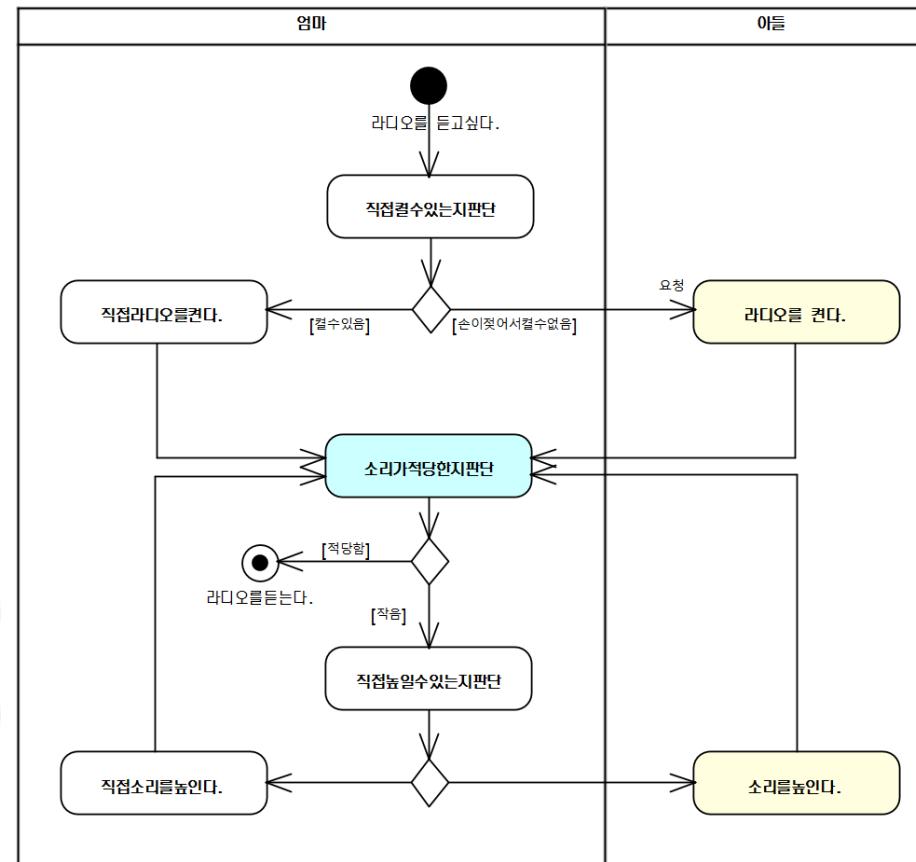
# 라디오 – 모델

- ✓ 실제 세계에서 벌어지는 일을 시스템 세계 안에서 표현하려할 때 만나는 첫번째 관문은 UML을 이용합니다.
- ✓ 시스템 요구사항을 명세하고, 그 명세를 충족하는 시스템을 모델링하는 것입니다.
- ✓ 이 시스템은 거실에서 일어난 일을 시스템 세계로 옮기는 작업으로 “신나는 집안일 시뮬레이터”라고 부릅니다.



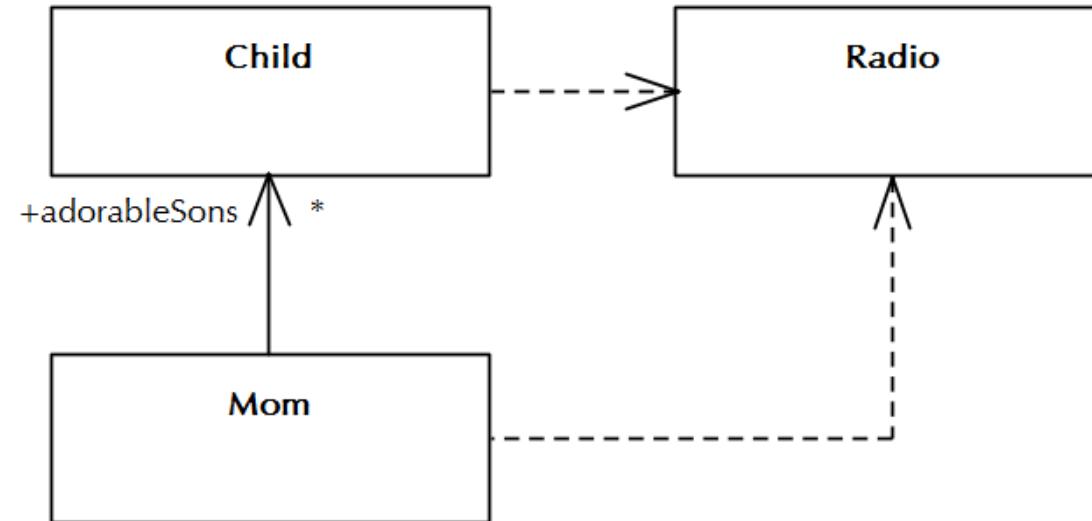
# 라디오 – 모델

- ✓ 요구사항을 이해하고 명세하는데 사용하는 도구에는 flow chart 등 여러가지가 있습니다.
- ✓ 아래 그림은 시스템 영역 안에서 발생하는 업무 흐름을 이해하여 표현하는 액티비티 다이어그램입니다.
- ✓ 엔지니어는 이 다이어그램을 작성하면서 전체 그림과 이야기의 흐름을 파악합니다.
- ✓ 요구사항 명세 단계에서는 시간 낭비를 하지 않고 큰 흐름을 파악하는데 초점을 두어야 합니다.



# 라디오 – 모델: 참여클래스1

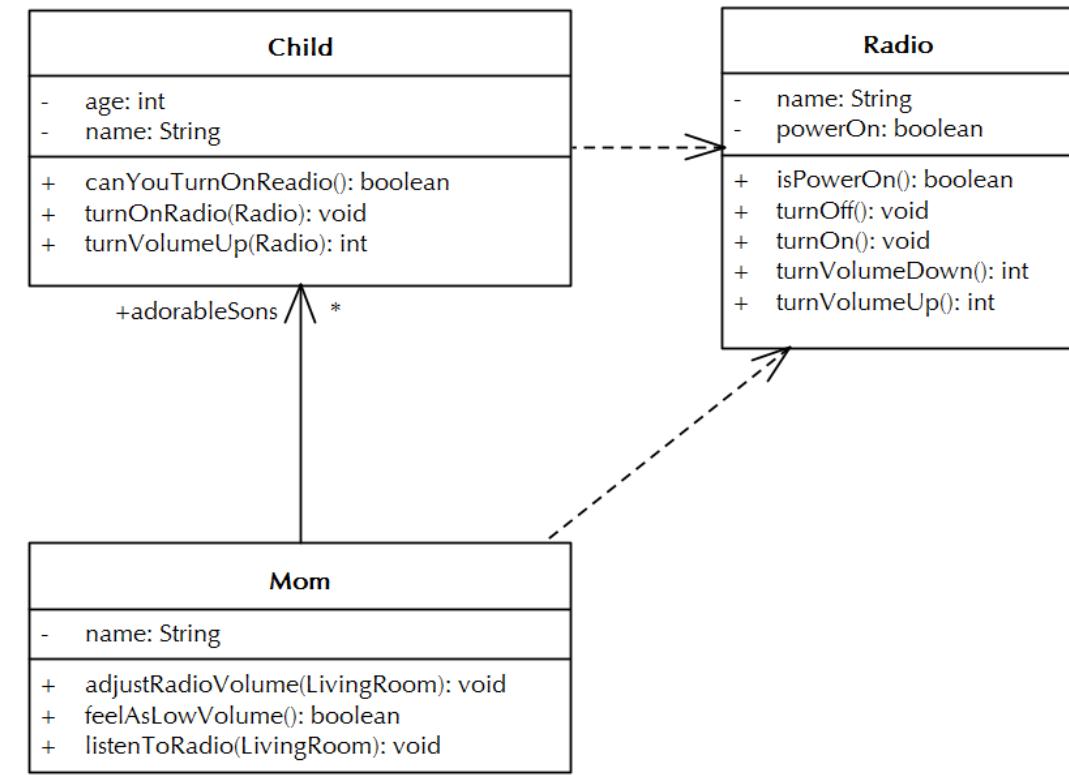
- ✓ 요구사항 명세를 마무리 하고 각 클래스가 어떤 역할과 책임을 가지고 시나리오에 참여하는지 고민을 합니다.
- ✓ 전체 시나리오를 진행되려면 각 클래스가 어떤 것인지를 알고, 어떤 활동을 해야 하는가를 생각해 봅니다.
- ✓ 처음 요구사항 명세에서 발견할 수 있는 클래스들은 엄마, 아이, 그리고 라디오 세 개입니다.



# 라디오 – 모델: 참여클래스2

- ✓ 전체 시나리오 흐름을 생각하면서 각 클래스의 역할과 책임을 고민합니다.
- ✓ 각 클래스의 책임을 자세히 들여다 보면, 각 책임은 “하나 이상의 메소드” 형태로 나타냅니다.
- ✓ 시나리오를 진행하기 위해 클래스가 내부 또는 외부의 요청을 받아서 수행 가능한 작업이 메소드입니다.

엄마의 역할과 책임
<ul style="list-style-type: none"><li>✓ 아들인 민수가 거실에 있다는 사실은 안다.</li><li>✓ 라디오가 거실에 있다는 사실을 안다.</li><li>✓ 민수가 라디오를 켤 수 있는지는 물어봐야 한다는 것을 안다.</li><li>✓ 민수에게 라디오 켜기를 요청할 수 있다.</li><li>✓ 민수에게 라디오 소리를 높여 달라고 요청할 수 있다.</li><li>✓ 스스로 라디오를 켤 수 있다.</li><li>✓ 스스로 라디오 소리를 높일 수 있다.</li><li>✓ 소리가 적당한지 판단할 수 있다.</li></ul>
아이의 역할과 책임
<ul style="list-style-type: none"><li>✓ 스스로 라디오를 켤 수 있는지 판단한다.</li><li>✓ 라디오를 주면 라디오를 켜거나 끌 수 있다.</li><li>✓ 라디오를 주면 라디오의 소리를 높이거나 낮출 수 있다.</li></ul>
라디오의 역할과 책임
<ul style="list-style-type: none"><li>✓ On/Off 여부 질문에 대답을 한다.</li><li>✓ 끄거나 켤 수 있도록 인터페이스를 제공한다.</li><li>✓ 소리를 높이거나 낮출 수 있는 서비스를 제공한다.</li><li>✓ 소리의 범위를 갖고 있어서 범위 안에서 조정한다.</li></ul>

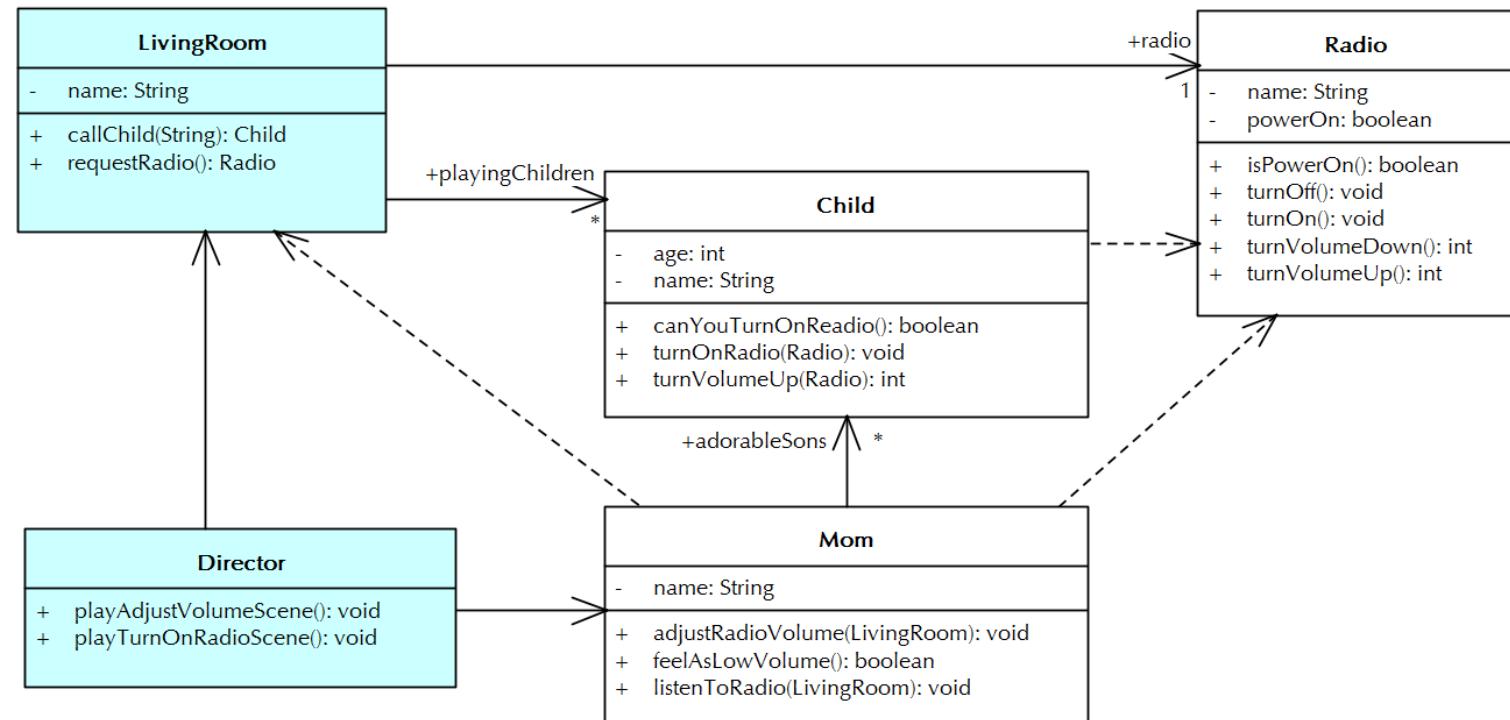


# 라디오 – 모델: 참여클래스3

- ✓ 엄마, 아이, 라디오가 함께 협업하는 공간인 거실(LivingRoom)은 우리가 찾아야 할 클래스입니다.
- ✓ LivingRoom은 라디오, 민수 등을 모두 껴안아 주는 존재이며, 그 자체가 이 시나리오의 무대 역할을 합니다.
- ✓ 무대를 준비하고 무대 위의 클래스 간의 관계를 설정해 주는 연출가(Director)클래스가 필요합니다.
- ✓ 연출가에 거실 클래스가 추가된, 총 다섯 개의 클래스가 협업하는 시나리오를 멋지게 진행해 봅시다.

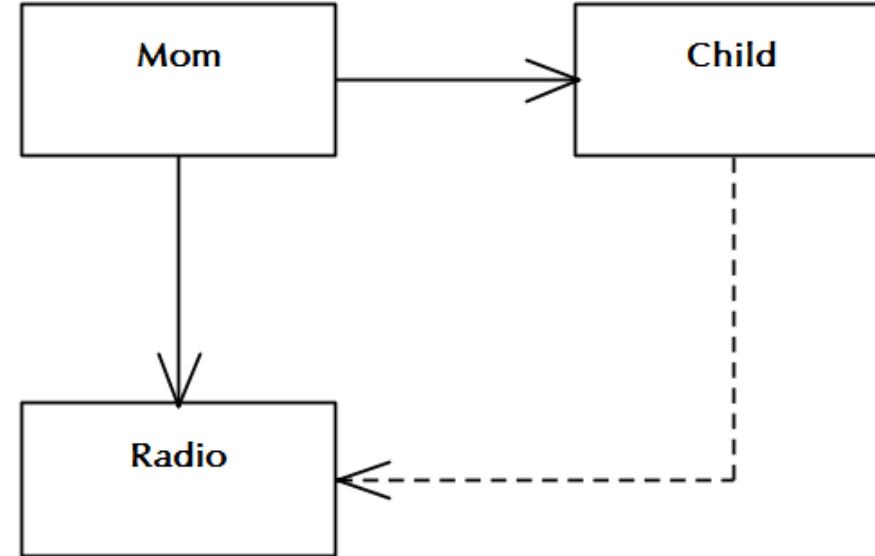
거실의 역할과 책임
✓ 라디오를 갖고 있다.
✓ 아이들이 놀 공간을 제공한다.
✓ 누군가 아이를 찾으면 불러준다.
✓ 향후 어떤 가전제품이든 받아 준다.

거실의 역할과 책임
✓ 무대를 준비한다.
✓ 라디오 켜기 장면을 연출하고 진행한다.
✓ 소리 조정 장면을 연출하고 진행한다.



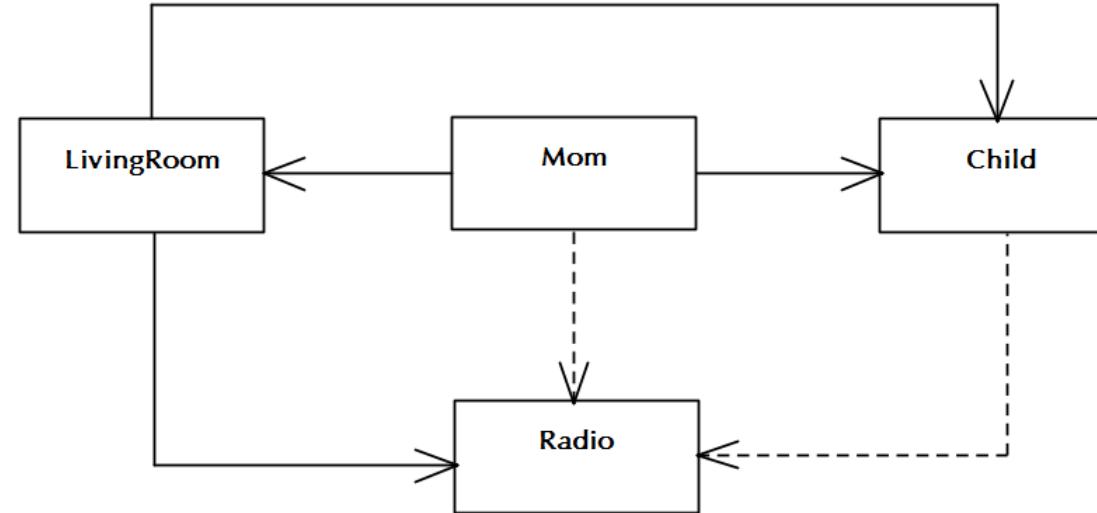
# 라디오 – 모델: 관계1

- ✓ 어떤 관계는 연관(association, 실선)관계이고, 어떤 관계는 의존(dependency, 점선)관계로 표현합니다.
- ✓ 상속(inheritance) 관계는 다른 관계 (연관 관계, 의존 관계 등)와 확실하게 구분할 수 있습니다.
- ✓ 하지만, 연관관계와 의존관계는 어느 정도 상황이나 주변 객체의 관계에 따라 변경되는 특성이 있습니다.
- ✓ 이 시나리오에서는 Mom이 Radio에 대한 소유권을 갖고 있는 모습이 자연스럽지 않습니다.



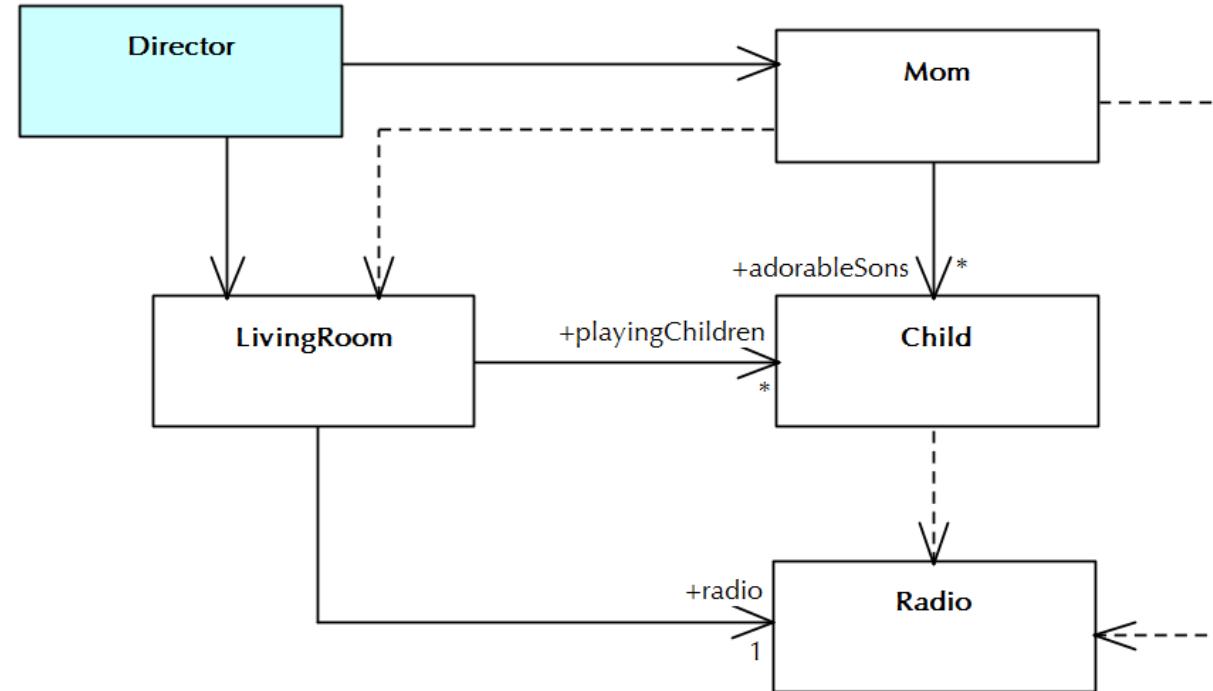
## 라디오 – 모델: 관계2

- ✓ LivingRoom이 Radio를 가지고 있으면 Mom은 Radio를 갖고 있어야 하는 책임으로부터 해방됩니다.
- ✓ LivingRoom과 Child는 일정 시간 지속적인 관계를 가지기 때문에 연관(association) 관계로 표현합니다.
- ✓ LivingRoom은 대본 진행을 위한 무대 역할을 담당하고 Mom은 LivingRoom과 협업하며 시나리오를 진행합니다. Mom과 LivingRoom 둘 중에 누가 시나리오를 이끌어 갈 것인가?



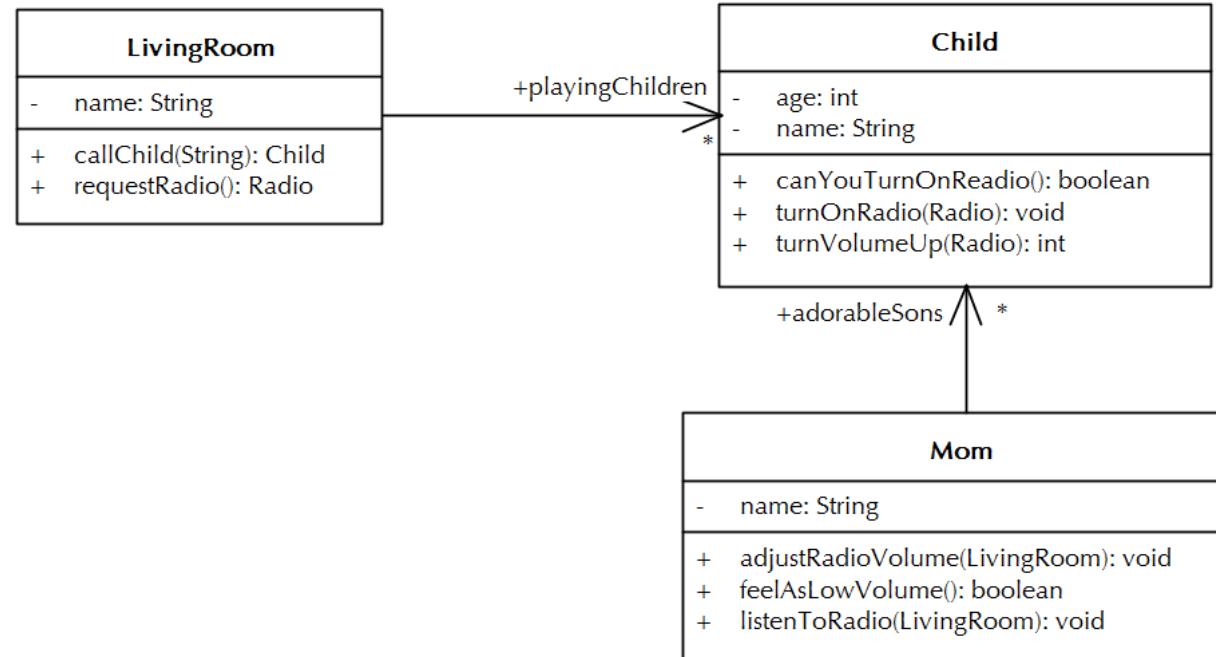
# 라디오 – 모델: 관계3

- ✓ 어떤 클래스가 협업하는 대상 클래스가 있고, 아무도 그 클래스를 잡고 있지 않다면 그 클래스를 직접 잡고 있어야 합니다. 즉, 연관(association) 관계를 갖고 있어야 합니다.
- ✓ 모델이 개선되면서 새로운 클래스가 등장하고 그 대상 클래스를 대신 잡아 주고 필요할 때 내게 넘겨줄 수 있다면, 그래서 필요할 때만 사용할 수 있다면 관계는 의존(dependency)관계로 느슨하게 변합니다.



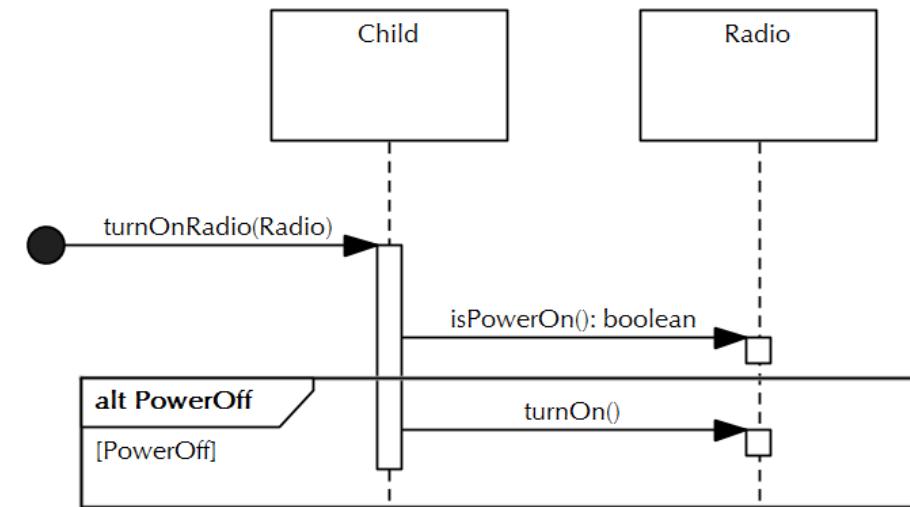
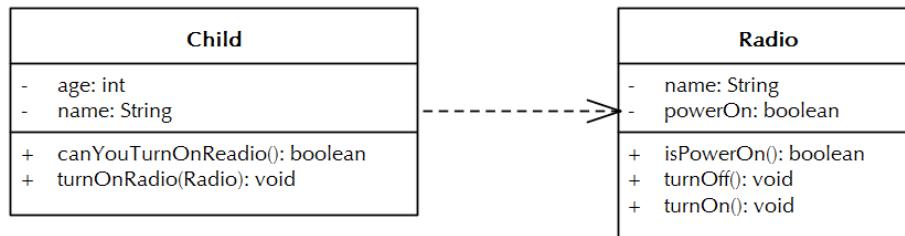
# 라디오 – 모델: 역할

- ✓ 세 개의 클래스 역할은 연관 관계를 가지는 두 클래스 간 항해(navigation)방향에서 발생합니다.
- ✓ 연관관계를 맺고 있는 상대방이 누구이며 어떤 관계인가에 따라 역할 이름은 달라집니다.
- ✓ 실세계에서도 A라는 사람은 아이들에게는 아빠이지만, 아내에게는 남편이며, 직장 후배들에게는 선배입니다.



# 라디오 – 모델: 협업 1 (라디오 켜기)

- ✓ 여러 객체가 참여하는 협업을 정의하기 전에 두 객체 사이에 간단한 상호작용을 모델로 표현해 봅니다.
- ✓ Child가 Radio를 켜는 과정을 표현합니다. Radio를 켤 때만 알고 있으면 되므로 의존 관계로 충분합니다.
- ✓ Child는 Radio가 제공하는 isPowerOn()이나 turnOn() 서비스 사용방법을 알아볼 수 있습니다.

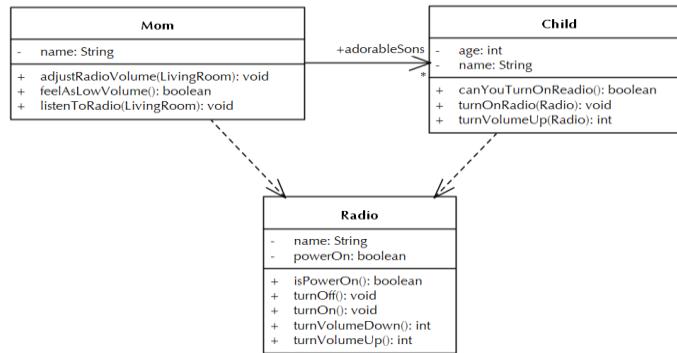


1. Radio상태를 Radio가 가지고 있을 때,
  - 1.1 켜져 있는지 라디오에게 물어 본다. 켜져 있지 않으면 켠다.
  - 1.2 라디오가 켜져 있으면 그냥 리턴하고 그렇지 않으면 켠다.
2. Radio상태를 Child가 가지고 있을 때,
  - 2.1 상태를 알고 있으므로, 켜져 있으면 그대로 둔다.
  - 2.2 상태를 알고 있으므로, 켜져 있지 않으면 켠다.

```
public void turnOn(Radio radio) {
    if (radio.isPowerOn()) {
        return;
    }
    radio.turnOn();
}
```

# 라디오 – 모델: 협업 2 [요청하기]

- ✓ Mom은 아들에게 라디오를 켤 수 있는지 물어 볼 수 있고, Radio를 켜달라고 부탁할 수 있습니다.
- ✓ Radio가 켜졌는지 확인하고 켜져있다면 바로 리턴합니다. 실제 코드는 상태를 확인하고 대응하는 코드들이 있습니다. 이런 정도의 내용에 집중하다 보면 주요 시나리오를 놓칠 수 있어 모델에서 표현할 필요는 없습니다.

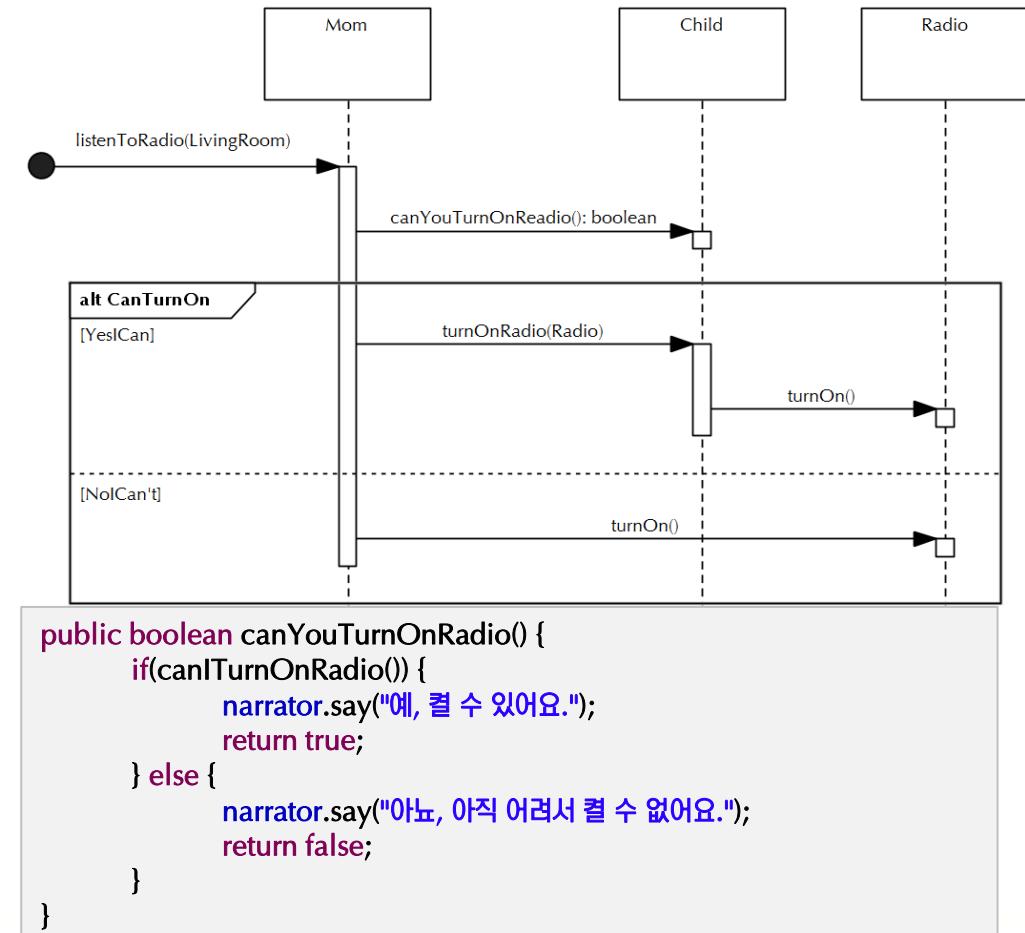


```

public void listenToRadio(LivingRoom livingRoom) {
    //
    Child smartSon = livingRoom.findFirstChild();
    Radio radio = livingRoom.findRadio();

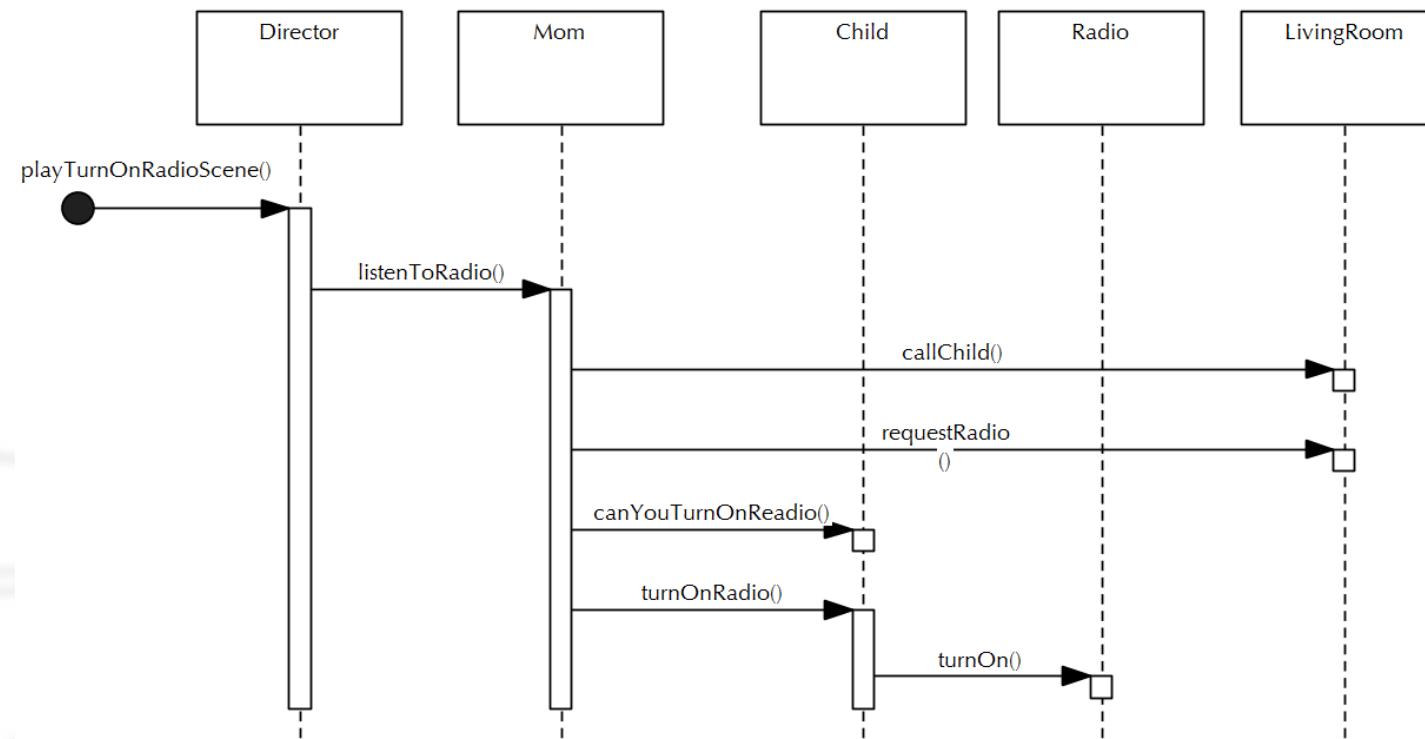
    if (radio.isPowerOn()) {
        return;
    }

    if(smartSon.canYouTurnOnRadio()) {
        smartSon.turnOnRadio(radio);
    } else {
        radio.turnOn();
    }
}
  
```



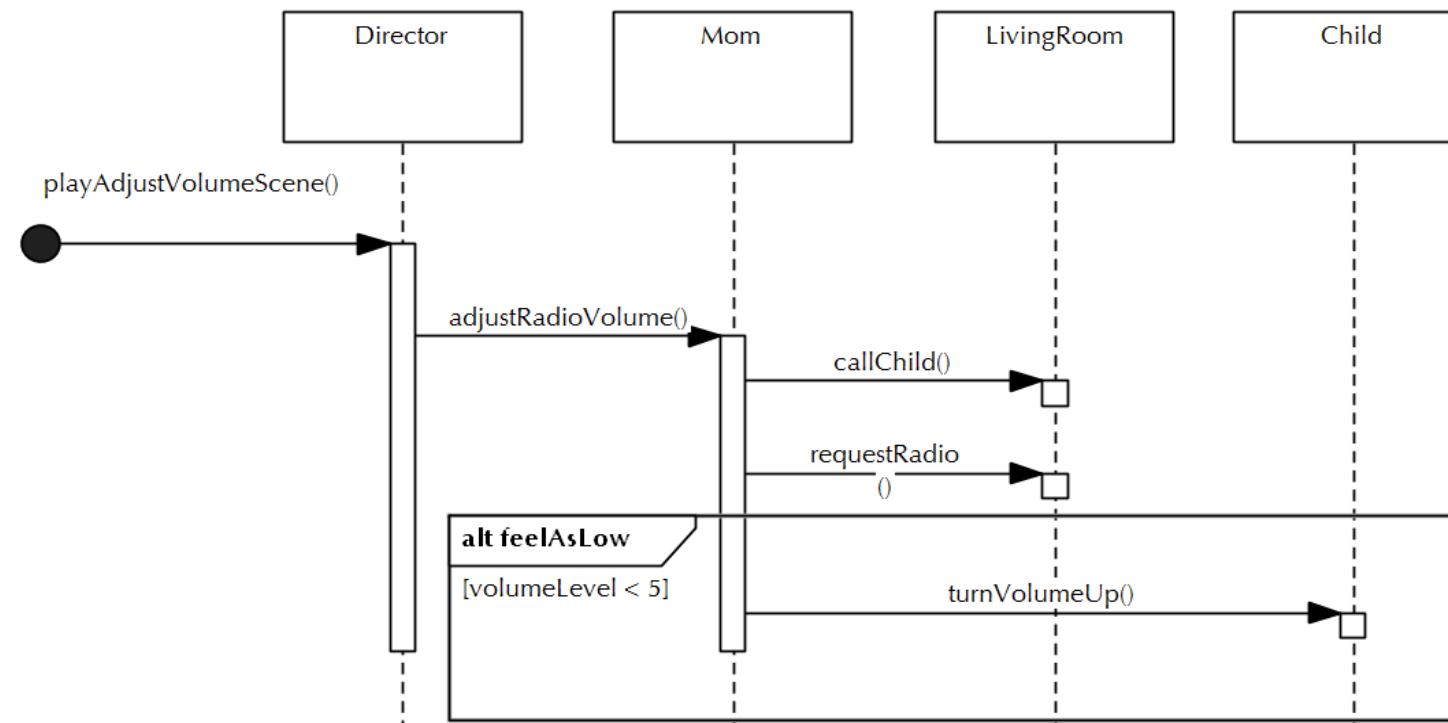
# 라디오 – 모델: 협업 3 (라디오 듣기)

- ✓ 라디오는 두 개의 유스케이스로 정의하고 각 유스케이스에서는 하나 이상의 협업 시나리오를 정의합니다.
- ✓ 앞에서 정의한 흐름에 따르면 복잡한 대안 경로를 가지지 않는 단순한 시나리오입니다.
- ✓ 라디오를 켜는 과정에서 이루어지는 협업 시나리오 (시퀀스 다이어그램)를 그려봅니다.



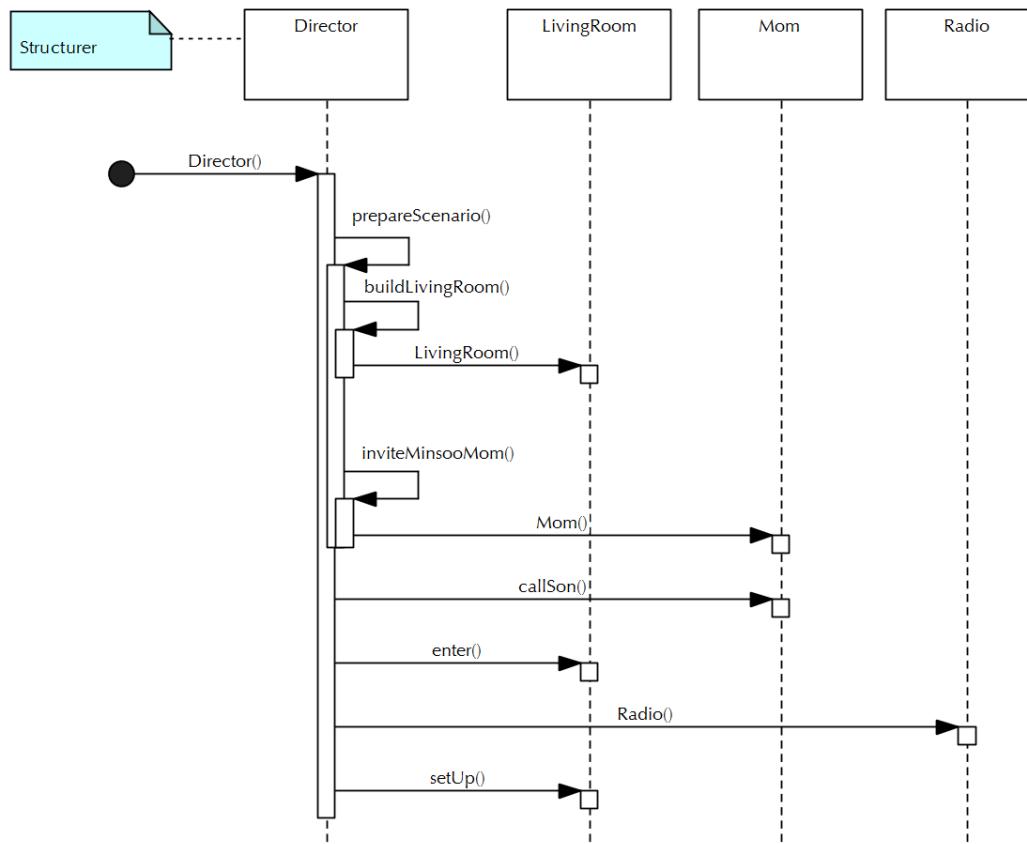
# 라디오 – 모델: 협업4 [소리조정]

- ✓ Radio를 켜고 나면 소리가 잘 안 들려서 Director는 Mom에게 소리 조정을 요청합니다.
- ✓ Mom은 아이가 거실에서 놀고 있는 것을 알고는 아이를 부른 다음, 거실에 있는 Radio를 달라고 합니다.
- ✓ Child에게 소리를 높여달라고 합니다. 소리 수준이 5가 될 때까지 반복해서 요청합니다.



# 라디오 – 모델: 협업5 (무대준비)

- ✓ 시나리오 진행에 필요한 무대는 누가 만드는 것일까요?
- ✓ 거실과 라디오, 엄마와 민수를 누가 무대 위로 옮겨놓았을까요?
- ✓ Director 클래스는 거실을 만들고, 민수 엄마를 초대하고, 아이를 불러서, 거실로 들어 보내고, 라디오를 만들어서 거실에 설치합니다.



```
public class Director {
    ...
    private void prepareScenario() {
        this.livingRoom = buildLivingRoom();
        this.mom = inviteMinsooMom("Yeongmi");
        livingRoom.enter(mom.callSon("Minsoo"));
        livingRoom.setUp(shopRadio("Tivoli"));
    }

    private LivingRoom buildLivingRoom() {
        LivingRoom livingRoom = new LivingRoom();
        return livingRoom;
    }

    private Mom inviteMinsooMom(String name) {
        return new Mom(name);
    }

    private Radio shopRadio(String brandName) {
        return new Radio("brandName");
    }
}
```

# 라디오 - 프로그래밍

- ✓ 처음 작성했던 프로그램을 버리셔도 됩니다. 물론 그곳으로 부터 시작하는 것도 좋습니다.
- ✓ 지금까지 거실이라는 실세계 공간에서 일어나는 일들을, 참여하는 객체들의 협업 관점에서 살펴보았습니다.
- ✓ 협업에 참여하는 모든 객체들의 행위를 잘 표현할 수 있도록 “객체지향적인” 프로그래밍을 작성해 봅니다.
- ✓ 실세계의 언어로 이해하고, 모델링 언어로 정리하고, 프로그래밍 언어로 확인합니다. 확인이 꼭 필요합니다.

코딩 실습 →

```
public void showMenuAndAction() {
    //
    while (true) {
        System.out.println();
        System.out.println(".....");
        System.out.println("  일하면서 라디오 듣기 메뉴");
        System.out.println(".....");
        System.out.println("  0. Program exit");
        System.out.println("  1. 라디오 켜기");
        System.out.println("  2. 라디오 소리 조정");
        System.out.println(".....");

        int inputNumber = acceptMenuItem("Select number");

        switch (inputNumber) {
        //
        case 1:
            turnOnTheRadio();
            break;
        case 2:
            adjustTheVolume();
            break;
        case 0:
            exitProgram();
            return;

        default:
            System.out.println("Choose again!");
        }
    }
}

public boolean canYouTurnOnRadio() {
    //
    narrator.say("라디오를 켤 수 있나요?");
    if(canITurnOnRadio()) {
        narrator.say("예, 켤 수 있어요.");
        return true;
    } else {
        narrator.say("아뇨, 아직 어려서 켤 수 없어요.");
        return false;
    }
}

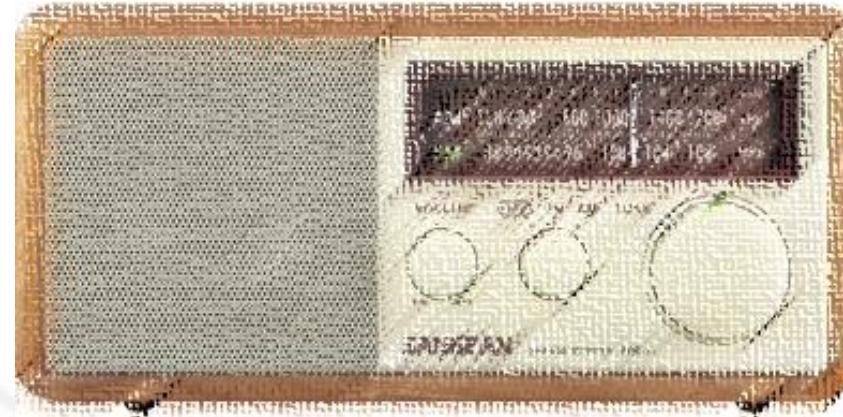
public boolean canYouAdjustVolume() {
    //
    narrator.say("소리를 조정할 수 있나요?");
    if(canIAdjustVolume()) {
        narrator.say("예, 할 수 있어요.");
        return true;
    } else {
        narrator.say("아뇨, 아직 어려서 할 수 없어요.");
        return false;
    }
}

public void turnOnRadio(Radio radio) {
    //
    if (!canITurnOnRadio()) {
        narrator.say("저는 라디오를 켤 수 없어요.");
        return;
    }

    if (radio.isPowerOn()) {
        narrator.say("이미 켜져 있는데요...");
        return;
    }
}
```

# 라디오 – 요약

- ✓ 실세계를 하나하나 따져 보면 상상할 수 없을 정도로 복잡한 메커니즘을 갖추고 있으며, 매우 효율적이고 효과적으로 모든 일이 돌아가고 있음을 알 수 있습니다.
- ✓ 객체 지향 사고의 핵심은 실세계의 메커니즘과 효율성을 시스템 세계로 투영하는 일입니다.
- ✓ 그러기 위해서는 실세계의 언어 → 모델링 언어 → 프로그래밍 언어로의 이전에 익숙해져야 합니다.
- ✓ 실세계 매핑의 첫 번째 시나리오인 라디오를 이해하고 프로그래밍 했습니다. 여러분의 생각은 어떠신가요?



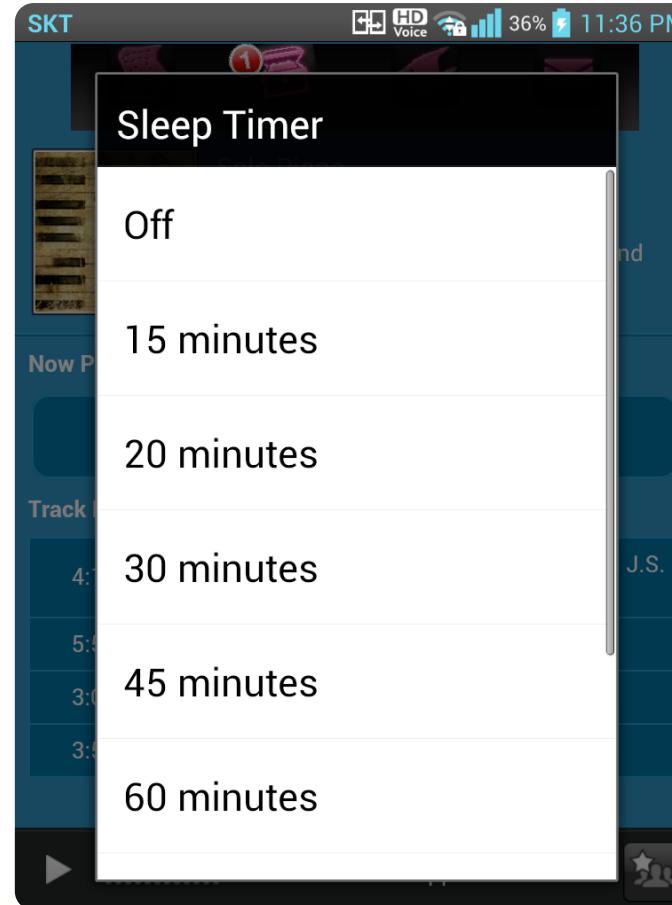
## 5. 이해

- ❖ 모델링을 해야 하는 이유를 생각해 봅니다.
- ❖ 모델링이 Thinking Process 인 이유를 이해합니다.
- ❖ 모델링을 해야 한다면, 올바른 절차는 무엇인지 알아 봅니다.

- ✓ 도메인 이해: 사례 I
- ✓ 도메인 이해: 사례 II
- ✓ 본질 파악을 위한 생각의 여정

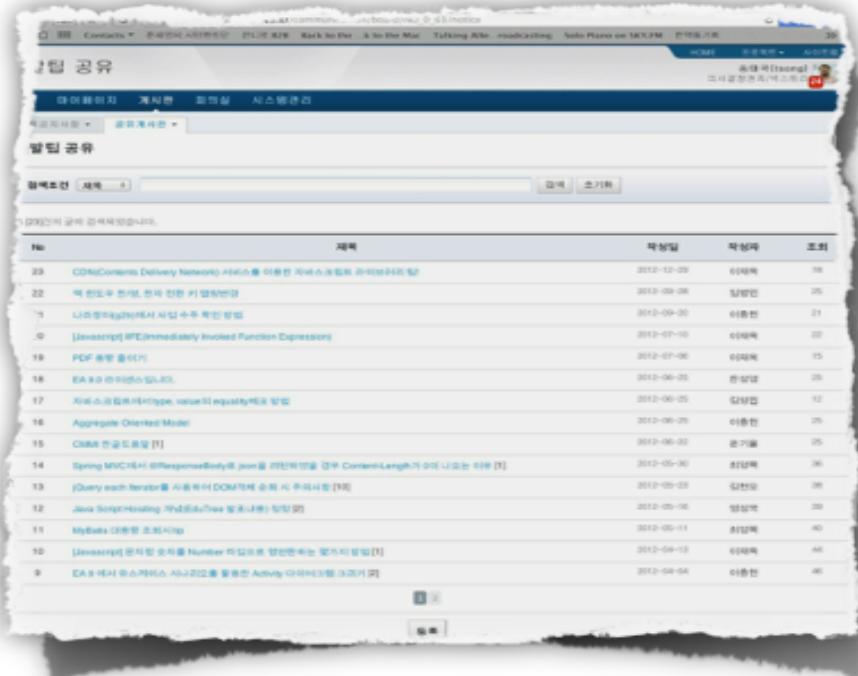
# 도메인 이해: 사례 I – Sleep timer

- ✓ 20분 후에 잠들고 싶은 애청자...



# 도메인 이해: 사례 II – Dummy board

- ✓ 게시판에 열받은 총무과장...
- ✓ Dummy board 사용 시나리오
  - [admin] 게시글을 등록한다.
  - [board] 무작정 기다린다...
  - [admin] 걱정이 된다... 궁금하다... 전화한다...



# 도메인 이해: 사례 II – NamooBoard (1/5)

## ✓ (가상의) Namoo Board 사용 시나리오

- [admin] 게시글 등록 [기간:2일, 보조통지수단:1-Email, 2-SMS, 완료보고]
- [board] 게시 직후 대상자 전원에게 Email을 보낸다.
- ... 일부 사용자가 이메일을 확인하고 게시글을 읽는다.
- 둘째 날 [board] 게시글을 읽지 않은 사용자에게 SMS를 보낸다.
- ... 나머지 사용자가 SMS를 확인하고 게시글을 읽는다.
- ... 둘째 날 오후 13:30분 마지막 사용자가 게시글을 읽는다.
- [board] 관리자에게 13:30분부로 모든 사용자가 게시글을 읽었음을 통지한다.
- [admin] 바로 다음 행동에 들어간다.

# 도메인 이해: 사례 II – NamooBoard (2/5)

## ✓ Namoo Board 사용 시나리오 (1/4) – 게시 및 옵션설정



NEXTREE 전체 공지사항

No 제 목

208 종무식 및 시무식 공지

207 2013년 다이어리 속지 신청

206 2012년 10월 프로젝트근무현황

205 2012년 창립기념 트레킹 팀원 알림

Posting...

OFF 옵션

199 2012년 기숙사 입주희망자 모집

198 플라워팀 제품명 공모합니다

197 인사 명령 (정기 진급 및 특별 진급)

▲ 댓글 쪽글이 없습니다.

▶ 마감일 2013년 01월 02일(수) 08시 ~ 10시

마감일 2013년 01월 02일(수) 08시 ~ 10시

내. 장소: 노보텔 앱페디 독산(신라호)

다. 협조 요청: 각 프로젝트에서는 고객들과 사건 협조를 통해 걸려간 시간에 종무식이 진행될 수 있도록 처리 바랍니다.

2. 시무식

내. 일시: 2013년 01월 02일(수) 08시 ~ 10시

다. 장소: 노보텔 앱페디 독산(신라호)

다. 협조 요청: 시무식 시작 전에 도착 바랍니다(차후 꺼 공지) 끝.

2012년 다이어리 속지 신청

1 2 3 4 5 6 7 8 9 10



ON 옵션

1월 3일 21:00 부터

1월 5일 18:00 까지

통지수단

레벨 1 E-mail

레벨 2 SMS

Report ME

매니간

# 도메인 이해: 사례 II – NamooBoard (3/5)

## ✓ Namoo Board 사용 시나리오 (2/4) – Email 통지



# 도메인 이해: 사례 II – NamooBoard (4/5)

- ✓ Namoo Board 사용 시나리오 (3/4) – SMS 통지



# 도메인 이해: 사례 II – NamooBoard (5/5)

- ✓ Namoo Board 사용 시나리오 (4/4) – 결과 보고

Eureka !!



둘째 날

**NEXTREE 전체 공지사항**

총무식 및 시무식 공지 | 정수미 2012-12-20

【총무식 및 시무식 공지】  
아래와 같이 2012년 종무식과 2013년 시무식 일정을 공지하오니 미리 일정 조정에 협조 바랍니다.

- 아 래 -

1. 종무식  
가. 일시 : 2012년 12월 28일(금) 17시~19시20분  
나. 장소 : 엠파이어 종로 씨푸드 레스토랑  
다. 협조 요청: 각 프로젝트에서는 고객들과 사전 협조를 통해 정해진 시간에 품무식이 진행될 수 있도록 처리 바랍니다.

2. 시무식  
가. 일시: 2013년 01월 02일(수) 08시 ~ 10시  
나. 장소: 노보텔 앱배서더 득산(신라홀)  
다. 협조 요청: 시무식 시작 전에 도착 바랍니다(차후 재 공지) 끝.

**OFF 옵션**

목록

▲ 윗글 윗글이 없습니다.  
▼ 아래글 2013년 다이어리 속지 신청



ON 옵션



1월 3일 21:00

부터



1월 5일 18:00

까지



통지수단

레벨 1

E-mail



Report ME

레벨 2

SMS



매시간

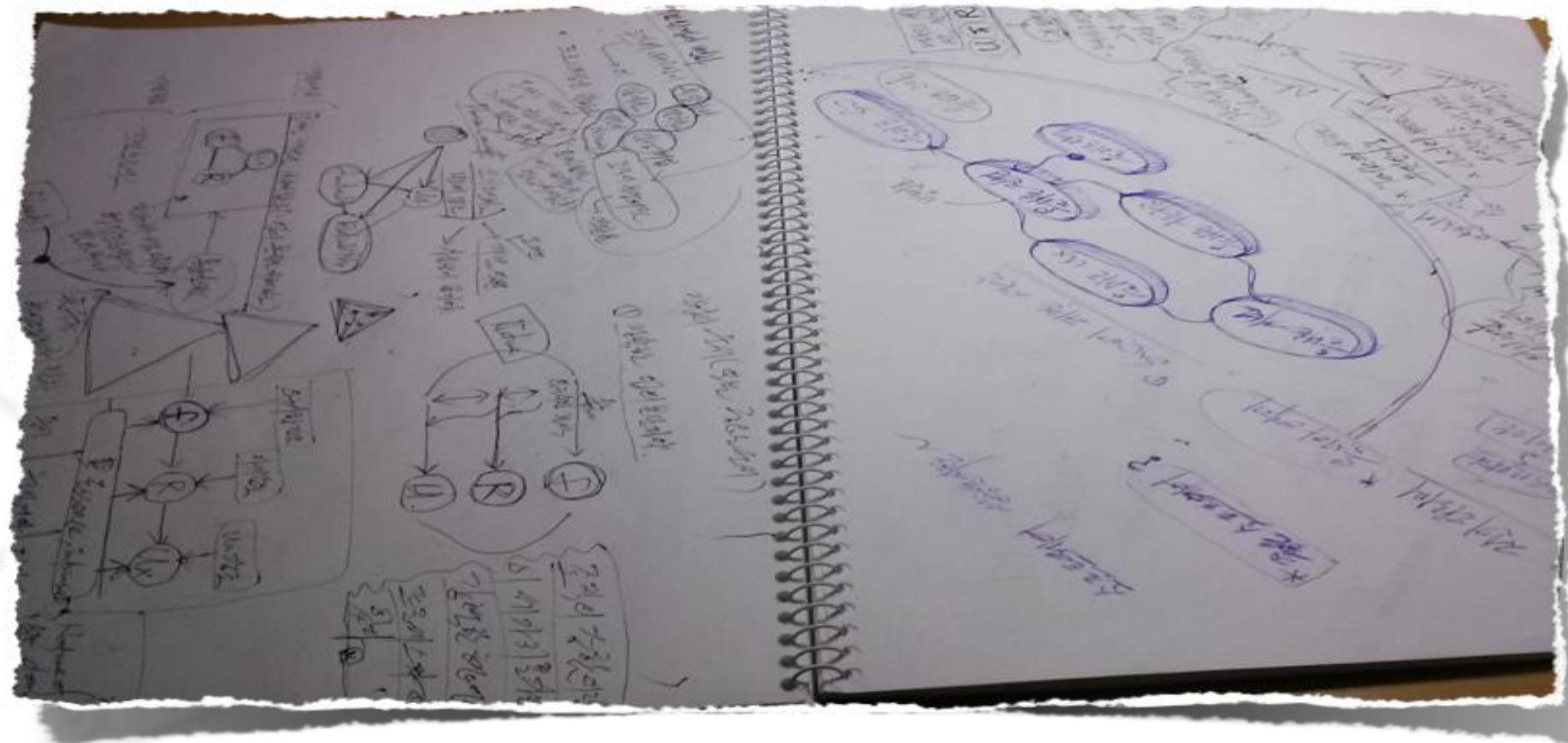
# 도메인 이해: 사례 III – 상품 가격

- ✓ 단순히 “상품은 가격을 갖고 있어야 한다.”로 모델링을 하면 다양한 Feature와 Option을 가지는 경우 곤란해 집니다. 판매 상품은 기본 모델에 하나 또는 여러 Feature 별로 Option을 선택한 결과입니다.
- ✓ 그리고 판매상품의 가격은 상품 모델의 가격 + (Feature.option) + ... 를 한 것이며, 재고 수량의 기준이 됩니다. 이 개념을 도메인 모델링에서 충분히 담아 주어야 실제로 옵션별 재고나 가격 정책을 가져갈 수 있습니다.



# 본질을 파악을 위한 생각의 여정

- ✓ “대상의 본질이 무엇인가”에 대한 이해없이 좋은 시스템을 개발할 수 없습니다.
- ✓ 대상의 본질에 대한 깊이 있는 이해를 얻기 위해서는 깊이있는 대화, 생각, 정보 등이 필요합니다.
- ✓ 스케치 북은 가장 좋은 모델링 도구(??) 입니다.



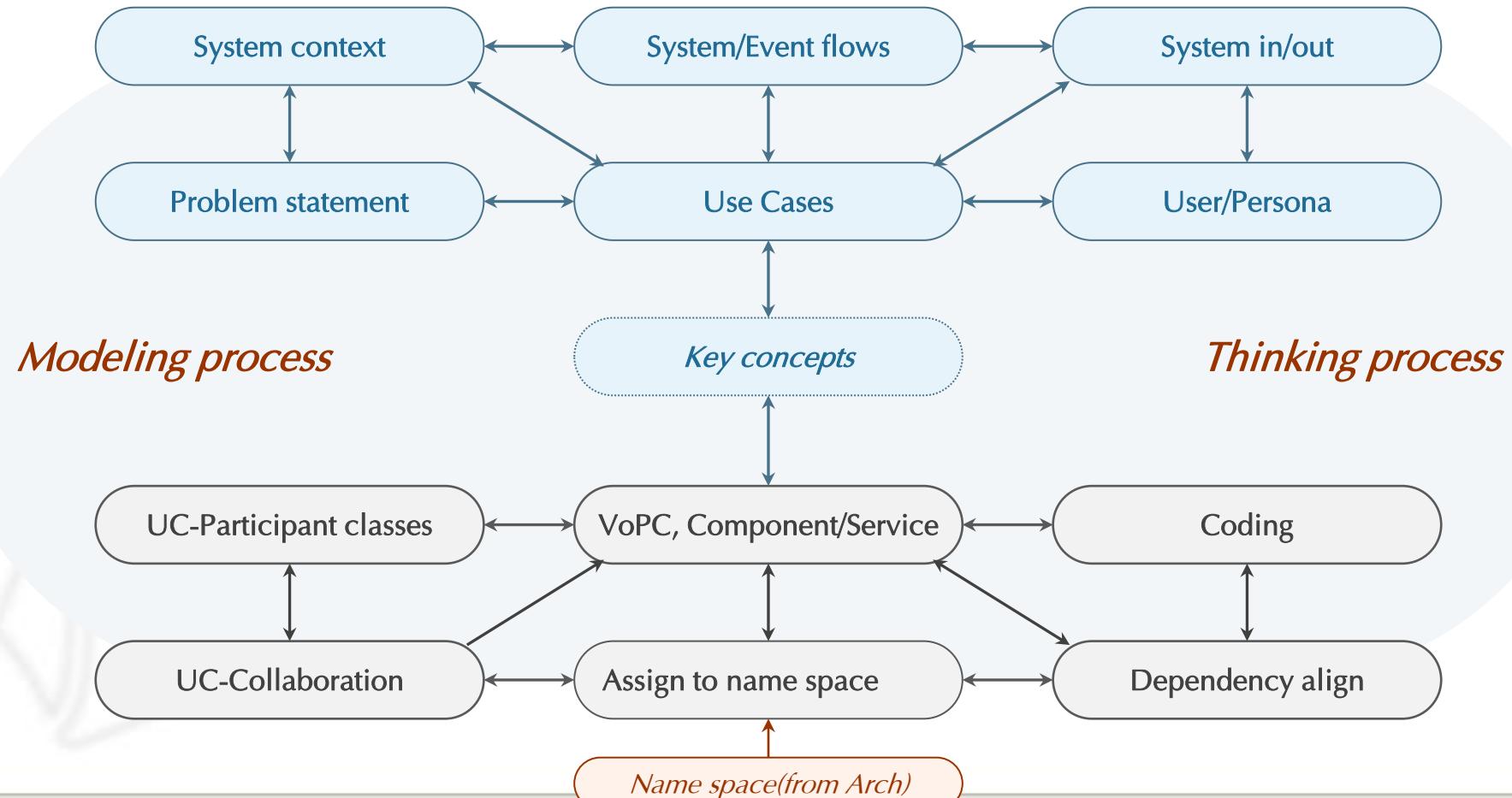
## 6. 모델링 절차

- ❖ 모델링을 해야 하는 이유를 생각해 봅니다.
- ❖ 모델링이 Thinking Process 인 이유를 이해합니다.
- ❖ 모델링을 해야 한다면, 올바른 절차는 무엇인지 알아 봅니다.

- ✓ 모델링 흐름
- ✓ 모델링 절차
- ✓ 시연
- ✓ 실습

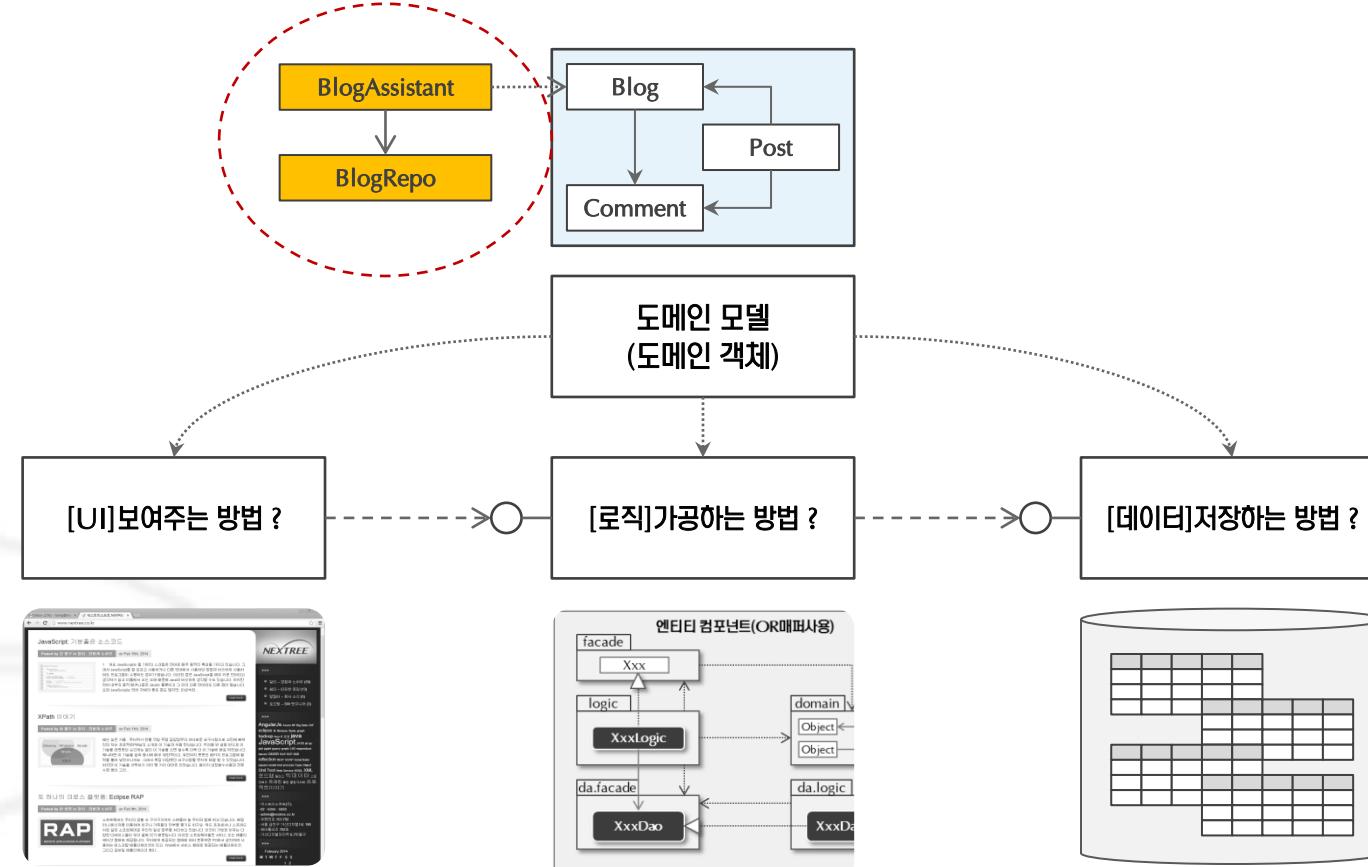
# 모델링 흐름(1/4)

- ✓ 요구사항으로부터 시작하여 소스코드로 이르는 길은 매우 다양합니다.
- ✓ 어떤 경로를 선택하든, 그것은 생각하는 절차이며, 그 절차는 관점과 높낮이를 포함하고 있습니다.
- ✓ 목표 시스템을 다양한 관점과 다양한 상세도로 바라보면서 시스템의 형상을 만들어 갑니다.



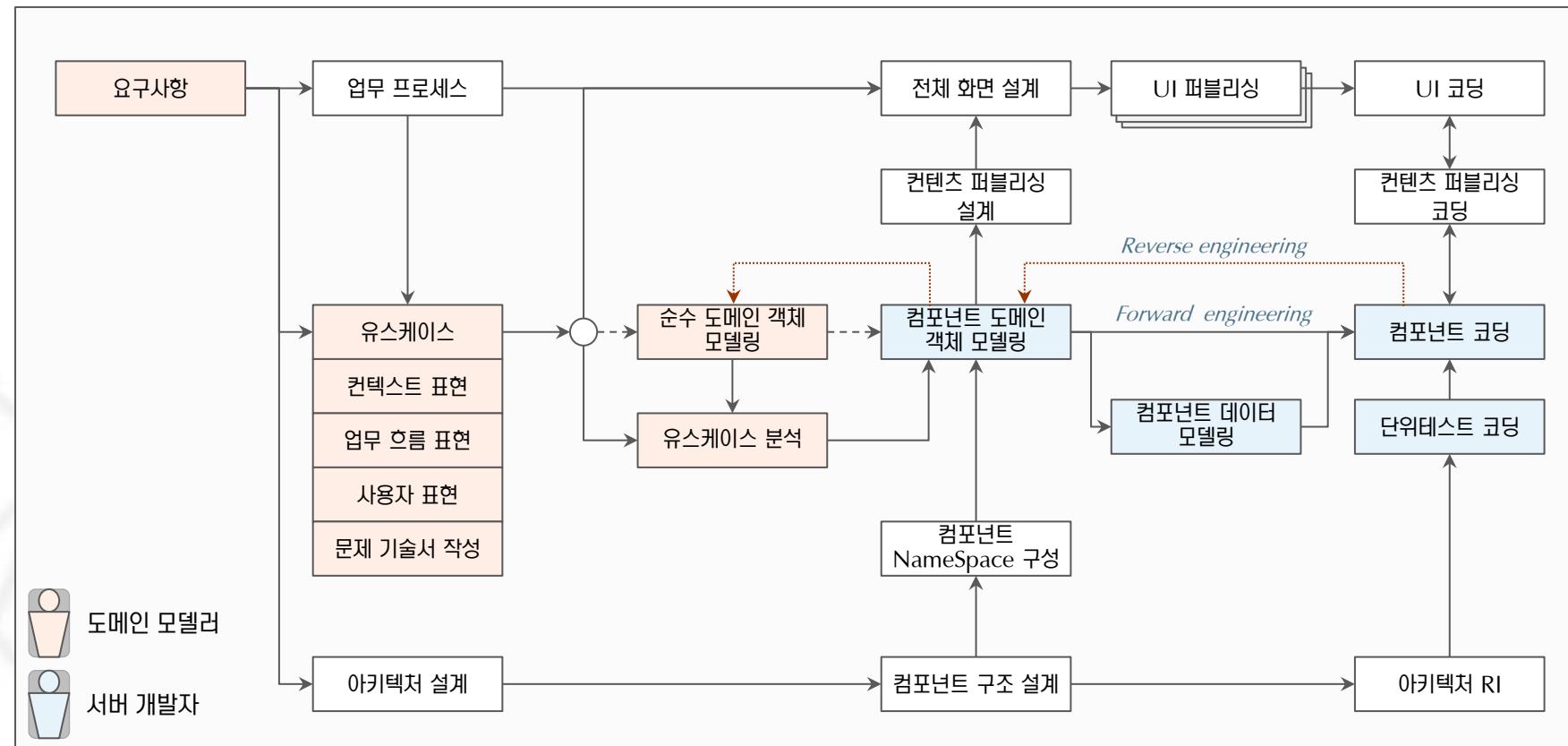
## 모델링 흐름(2/4)

- ✓ 도메인 모델의 초점이 어디에 있는지 정확하게 이해하여야 합니다.
- ✓ UI 모델과 도메인 모델, 그리고 데이터 모델은 서로 다른 프로세스를 따라 진행합니다.
- ✓ 모든 모델의 중심에는 도메인 모델이 있음을 잊지 말아야 합니다.



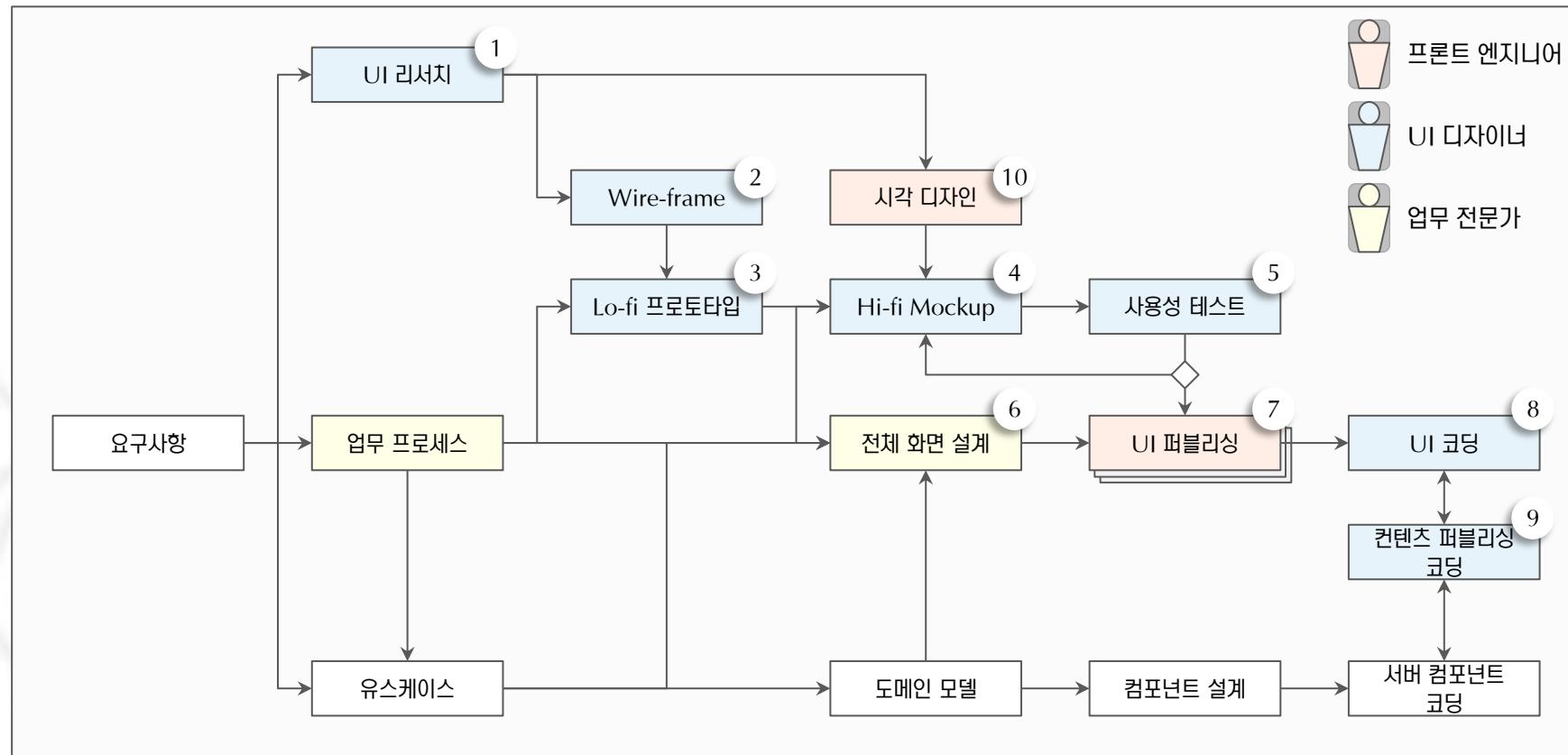
# 모델링 흐름(3/4)

- ✓ 조직에서 정의한 모델링 흐름은 개발 프로세스와 거의 일치합니다.
- ✓ 비록 작업 순서(생각하는 순서)는 모델러의 마음에 달려 있지만, 정규 프로세스로 표현하면 다음과 같습니다.
- ✓ 개발 프로세스(==개발 방법론)는 산출물을 만드는 순서가 아니라 바로 생각하는 순서입니다.



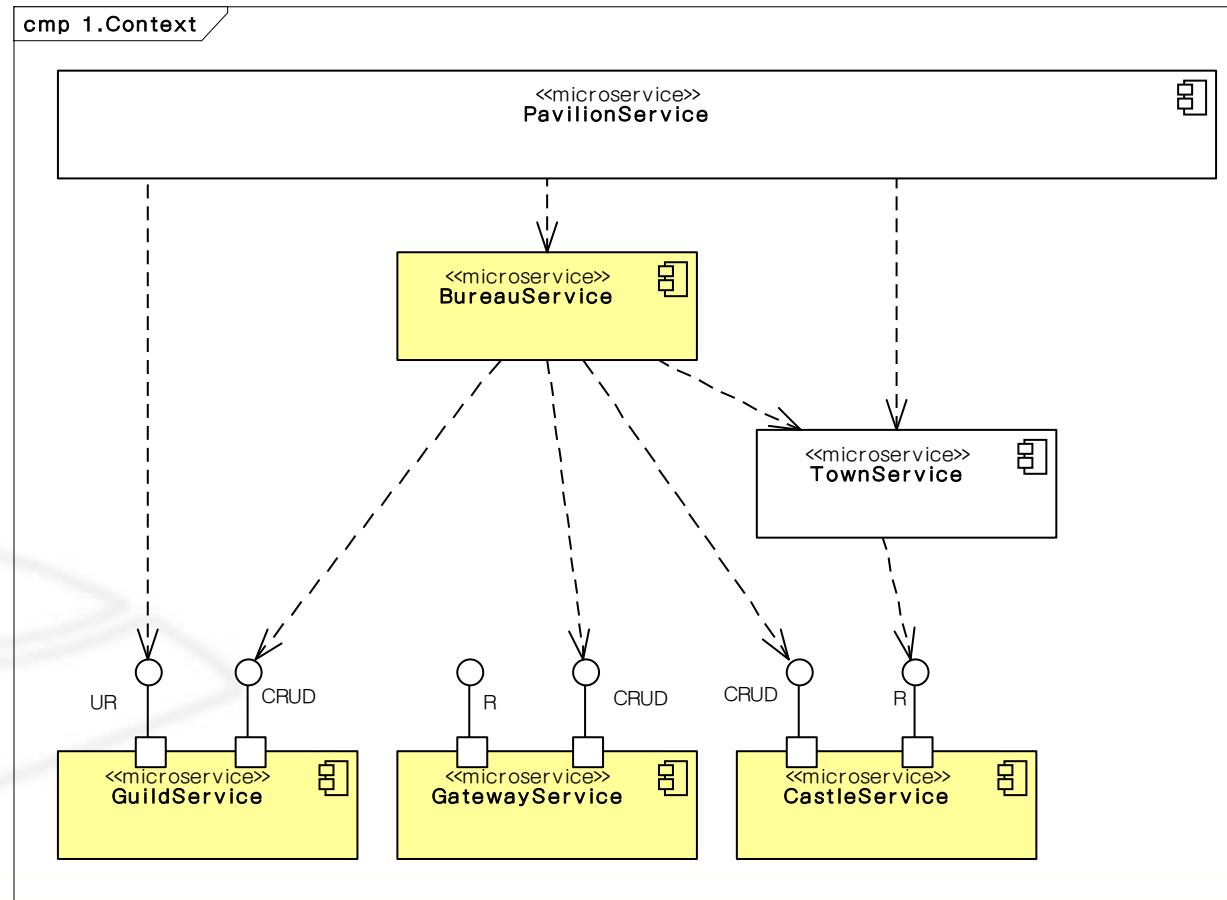
## 모델링 흐름(4/4)

- ✓ GUI 프론트 또는 웹 UI 프론트는 어떤 프로세스를 따라서 개발하는가에 따라 역할 담당자가 다양할 수 있습니다. UI 디자이너(10), UX 전문가(5), 프론트 엔지니어(8,9), 웹 퍼블리셔(7), UI 기획자(1,2,3,4) 등 다섯 가지로 역할로 나누는 경우도 있지만, “Cross-functional” 개념을 적용하여 두 개로 나눕니다.



# 모델링 – 시스템 컨텍스트

- ✓ 목표 시스템이나 서비스가 다른 시스템/서비스들과 어떤 관계를 갖고 있는지를 표현합니다.
- ✓ 아래 다이어그램에서 Bureau 서비스가 어떤 서비스를 사용하고 있는지 알 수 있습니다.
- ✓ 큰 틀에서 시스템/서브 시스템/서비스들의 R&R을 정의하거나 조정할 때 매우 중요한 View입니다.



# 모델링 – 문제 기술서

- ✓ 문제 기술서(problem statement)는 목표 시스템/서비스가 없을 때의 불편함과 바램을 서술합니다.
- ✓ 목표 시스템/서비스를 만들면 그런 불편함이 없어지거나 바램이 이루어지게 됩니다.
- ✓ 문제 기술서를 작성하다 보면, 새로운 기능을 찾아내기도 하지만, 중요한 것은 “필요성에 대한 이해”입니다.

## Stage 서비스의 문제 기술서 (예)

드라마 (as a microservice)는 작아야 합니다. 한 번에 하나의 기능을 잘 처리하도록 개발하여야 하며, 개별 드라마 단위로 업그레이드할 수 있어야 합니다. 예를 들어, 문제은행, 시험출제, 시험, 채점은 하나의 드라마에 서로 다른 피쳐와 옵션으로 구성할 수도 있습니다. 하지만, 기능을 추가하다보면 금방 규모가 커 질 수 있습니다. 따라서 제일 먼저 시험을 독립 드라마로 분류할 수 있습니다. 왜냐면 나머지 기능과 이 기능은 사용자가 다르기 때문입니다. 다양한 형식으로 시험을 볼 수 있도록 기능을 강화한 후 독립할 수 있습니다. 이 경우, 출제한 문제를 시험으로 이동하거나, 시험 본 결과를 채점으로 이동하여야 합니다. 이 때 무대(Stage)를 사용합니다. 다른 드라마와 느슨한 관계를 보장해 줍니다. 이런 목적 때문에 연관성있는 드라마 그룹이 무대를 지정하고 공유합니다. 따라서, 하나의 CinemaRoom 안에는 여러 개의 무대가 있을 수 있습니다. 현실은 무대 하나를 공유하고 있습니다. 우리는 이것을 무대 그룹으로 표현하면 무대를 하나로 다룰 수 있습니다. 무대 속에 들어가면 작은 무대 여러 개가 있다고 생각하면 됩니다. 그 중에 어떤 무대는 public 특성을 갖고 있고, 어떤 무대는 private 속성을 갖고 있습니다. public은 모든 드라마를 위한 무대의 한 부분이고, private 무대는 해당 드라마 그룹을 위해서 설치한 무대입니다. Talk, 게시판 등은 public 무대 부분을 많이 사용하고, 시험, 채점 등은 당연히 private 무대를 사용하겠죠. 맥은 어떤 앱이든 "공유하기"메뉴가 있습니다. 제가 편집하고 있는 이 화면을 이메일로 보낼 수도 있고, 텔레그램으로 보낼 수도 있습니다. 이 내용을 무대 개념으로 설명하면, 이 앱(드라마)이 자신을 상영하고 있는 무대(무대 위에서 드라마가 상영됨)에게 물어보죠. 이 무대에 참여하고 있는 드라마가 무엇이냐고(전후로 상영될 수 있는 드라마가 무엇이냐고). 그럼 무대가 이야기 해주죠, 이메일, 텔레그램, 메모 등이 있다고. 그 중에서 메모 앱을 선택하면, 무대에는 제가 보낸 정보 앞에 목적지를 "메모"라고 붙인 후 무대에 저장해 둡니다. 나중에 메모 앱이 실행되면(상영되면), 제일 먼저 무대에게 물어보죠. 혹시 내게 온 메시지 없어요... 라고. 무대는 공유하는 방식입니다. 그래서 개발할 때도 무대는 라이브러리 형태로 드라마 개발자에게 제공해 주어야 합니다. 개발할 때, 무대로 저장하거나 읽어오는 것이 가능해야 아니까요. 무대는 드라마 간의 공유 저장소입니다. 그런데 한 번 저장을 요청한 쪽은 다시는 읽을 수 없으며, 전달되었는지 여부도 관심이 없는 one-way 방식의 메시지 저장소입니다. 소규모의 드라마 그룹 전용의 메시지 큐라고 보시면 되겠네요.

“별도의 문서로 작성하는 것이 아니라, 다이어그램을 만들고 그 속에 “노트” 표기를 이용하여 서술합니다.”

# 모델링 – User/Persona

- ✓ 사용자의 이름과 역할 이름, 사진, 동기와 요구, 간략한 시나리오 등을 기술하면서 사용자를 이해합니다.
- ✓ 사용자의 요구(needs)와 행동 패턴(behavior)을 서술함으로써 사용자 관점에서 시스템을 생각해 봅니다.
- ✓ 사용자 조사와 사용자 세분화 작업 과정에서 수집한 데이터를 이용하여 작성합니다.

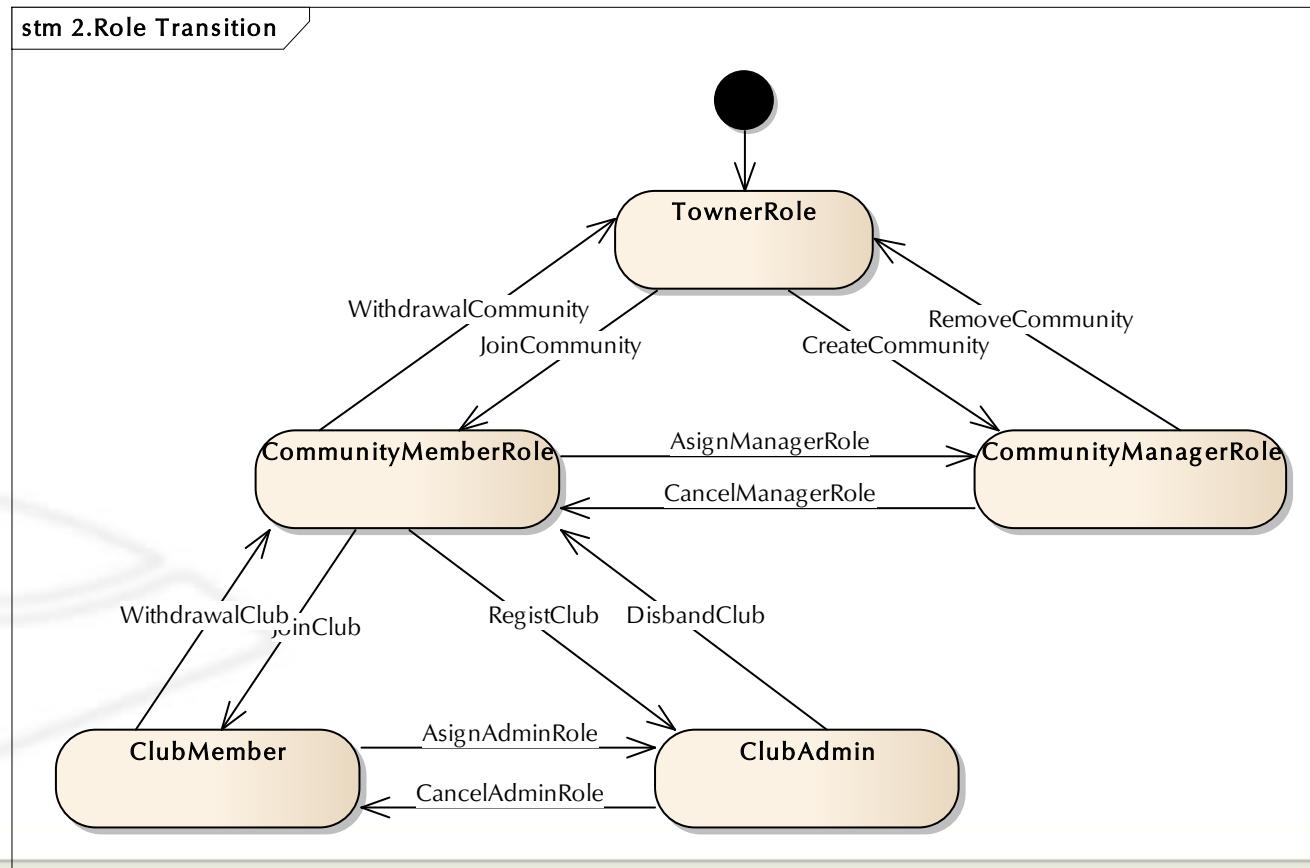
## 2.2. 박정훈 : 판매자

나이	38세	성별	남	거주지	서울 금천구
	학력	대학	PC 친숙도	상	웹 친숙도
박정훈은 소프트웨어 교육 강사로 근무하는 결혼 5년 차 기혼자이다. 주간에는 교육강사로 하루 종일 강의를 하지만 저녁에는 그의 아내와 함께 경매사이트를 이용해 아르바이트를 하고 있다. 그의 아내는 주변에 IT 종고 물품과 관련된 일에 종사하는 지인들이 많아 다양한 아이템을 싼 가격에 손쉽게 구할 수 있다. 지난 주에도 CD player를 싼 가격에 구해 경매사이트를 통해 적잖은 수익을 올렸다. 현재는 IT 종고 물품만을 취급하고 있지만 향후 다른 물품들도 취급할 계획을 가지고 있다..					
사용자 요구					
<ul style="list-style-type: none"><li>- 경매등록의 간편한 절차..</li><li>- 경매물품 별 관리방법 제공..</li><li>- 세금 관련 서류 온라인 출력..</li></ul>					

동기	시나리오	기능	행동
경매물품과 경매조건을 등록하려 함..	경매물품 및 경매조건 등록하기..	<ul style="list-style-type: none"><li>- 경매물품 상세정보 등록..</li><li>- 경매조건 등록..</li><li>- 경매등록..</li></ul>	박정훈은 경매물품 정보와 경매조건 등을 정리하고 “경매등록” 링크를 클릭함..
개별 경매의 진행상태를 확인하려 함..	개별 경매 진행상태 열람하기..	<ul style="list-style-type: none"><li>- 구매자들의 경매입찰 내역..</li></ul>	
경매와 관련된 구매자들의 질문사항에 답변하려 함..	구매자의 질문사항 답변하기..	<ul style="list-style-type: none"><li>- 질문사항 답변..</li></ul>	

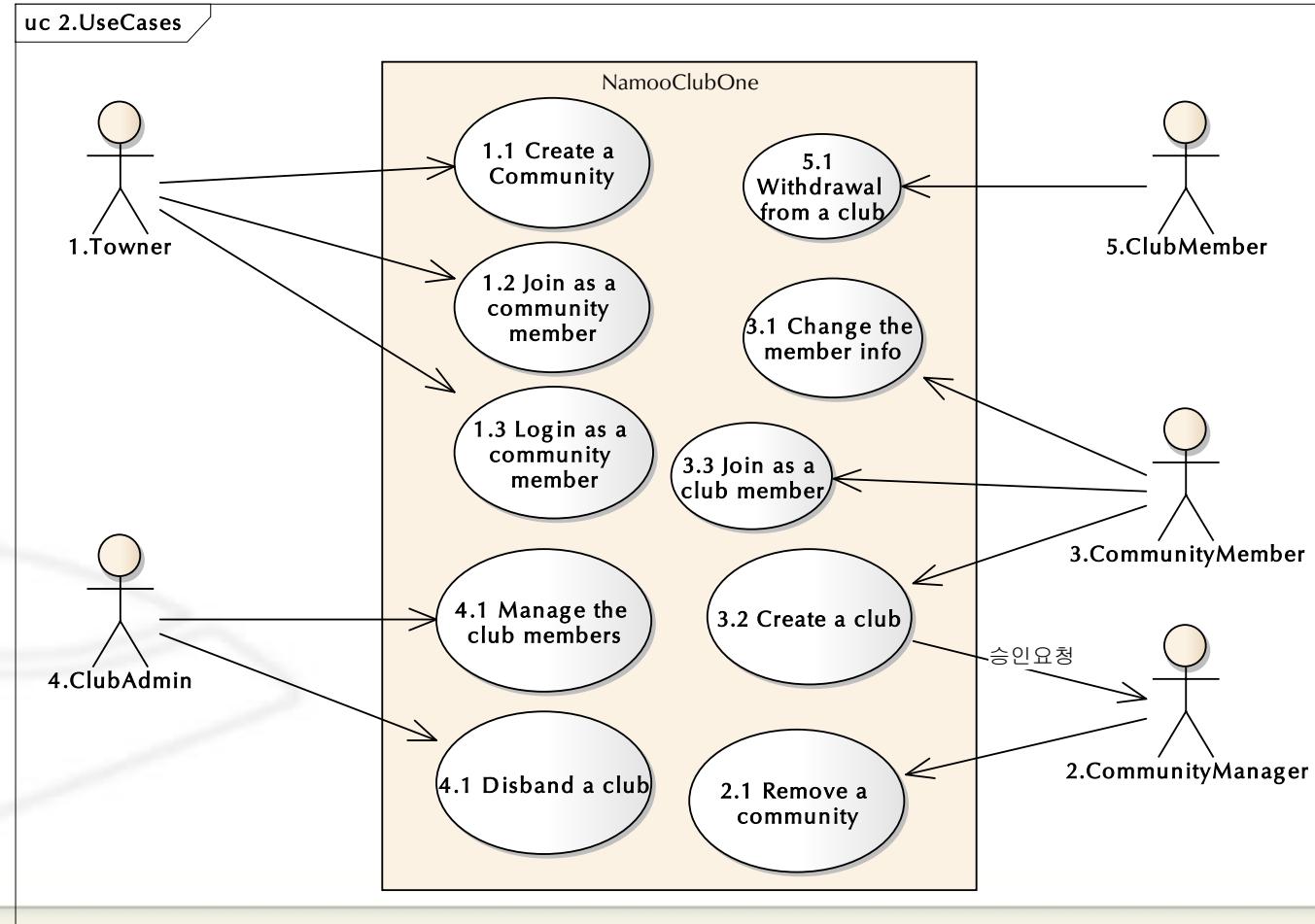
# 모델링 – User/Persona

- ✓ Actor라 불리는 역할 담당자들의 역할 상태가 어떻게 변경되는지 이해할 필요가 있습니다.
- ✓ “사용자” 액터가 회원가입을 하면 “회원” 역할 담당자가 됩니다. 동네사람(Tower)이 커뮤니티를 만들면 관리자(CommunityManager)가 됩니다. 커뮤니티 회원이 클럽에 가입하면 클럽회원(ClubMember)이 됩니다.



# 모델링: Use Cases

- ✓ Use Case를 식별하고 명세하면서 시스템이 사용자(actor)에게 무엇을 제공해 주어야 하는지 이해합니다.
- ✓ Use Case는 명세 보다는 식별이 더 중요하며, 자세한 명세서 없이도 다음 작업을 진행할 수 있습니다.
- ✓ 처음 2~3개만 식별하고도 다음 단계 작업을 진행할 수 있습니다. 오래 머물러 있을 필요가 없습니다.



# 모델링: Use Cases

- ✓ 필요할 경우, 시간이 충분하거나 자세한 요구사항 이해가 필요한 경우에는 Use Case 명세서를 작성합니다.
- ✓ Use Case 명세서 작성은 코번의 “Writing effective Use Cases”를 참조합니다.
- ✓ Use Case 명세서 작업은, 식별해 두었던 Use Case 들에 대한 분리, 통폐합 작업을 유발합니다.

목표: 경매 등록

수준: 사용자 수준

선조건: 회원은 로그인 상태임

## 이벤트 흐름

1. 판매자는 물품명과 모델명을 입력한다.

2. 시스템은 회원의 신용상태를 체크하고, 미결재 금액 여부를 체크한 후, 물품명과 모델명을 이용하여 추천 카테고리를 검색하여 제시한다.

3. 판매자는 희망 카테고리를 선택한 후에 경매 정보를 등록한다.

설명을 입력한다.

사진을 업로드한다.

경매방식 판매일 경우, 시작가를 입력하고, “즉시 구매 방식”일 경우, 즉시구매가격을 입력한다. 두 가지 판매 방식을 모두 선택할 수 있다.

지불 방식을 선택한다.

배송 정보를 입력한다(체적, 중량).

4. 시스템은 경매 정보를 저장하고, 경매 시작여부 메시지를 보여준다.

5. 판매자는 경매 정보를 확인하고 이상이 없으면 “경매시작”을 선택한다.

6. 시스템은 경매정보를 저장하고 경매를 시작한다.

## 이벤트 흐름 – 예외

2a. 신용상태가 제한 등급일 경우, 경매등록을 할 수 없는 사유를 메시지로 보여주고 등록을 종료한다.

2b. 미결재 금액이 있을 경우, 결재 안내 메시지를 보여주고 등록을 종료한다.

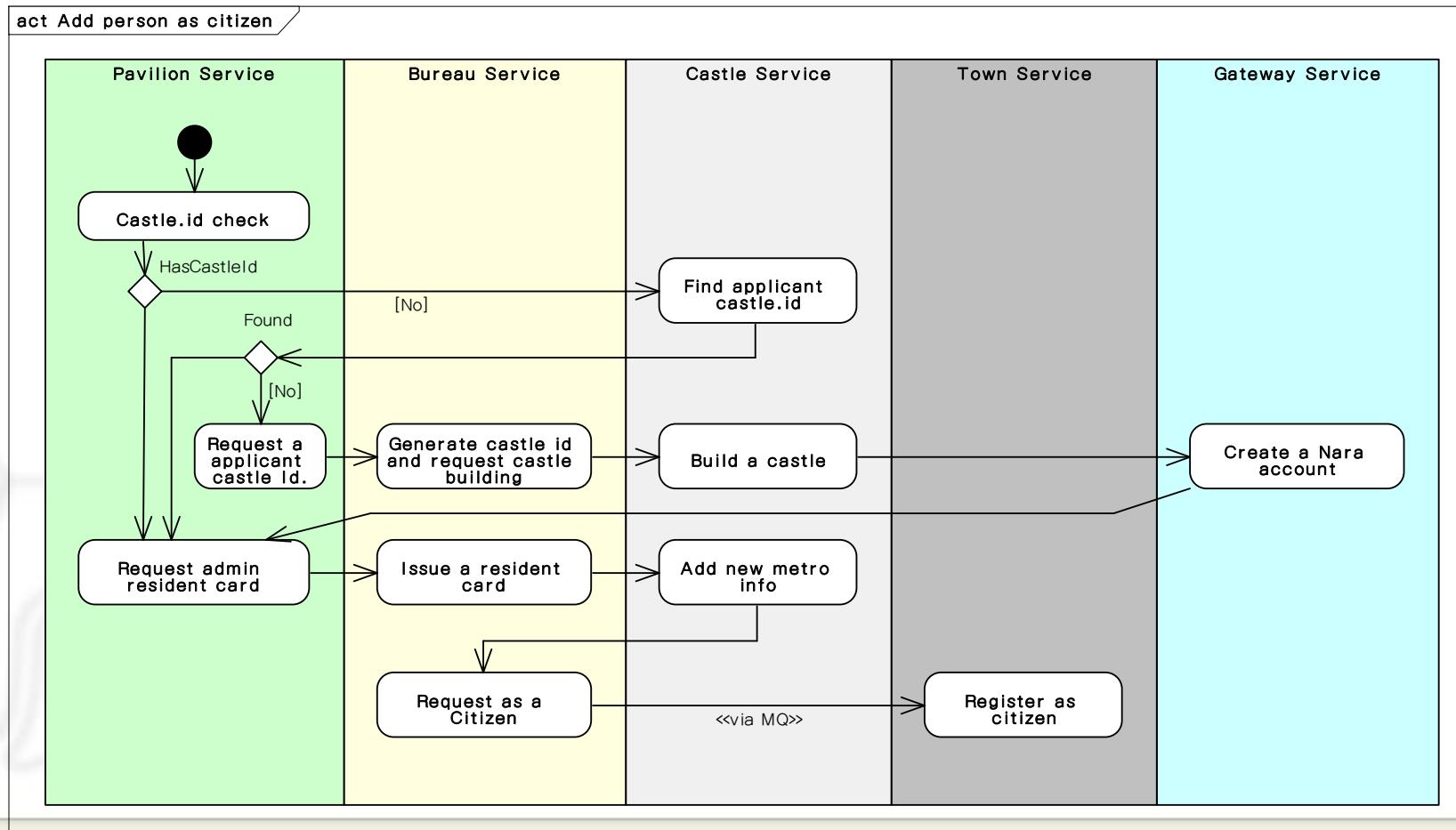
2c. 추천 카테고리가 없을 경우, 판매자가 직접 선택할 수 있도록 카테고리 찾기 화면을 보여준다. 이 때 판매자는 적절한 카테고리를 찾아서 판매물품의 카테고리로 지정한다.

5a. 경매 정보가 적절하지 않을 경우, “경매 정보 변경”을 선택한다.

5b. 경매를 취소하고 싶은 경우, “경매 등록 취소”를 선택한다.

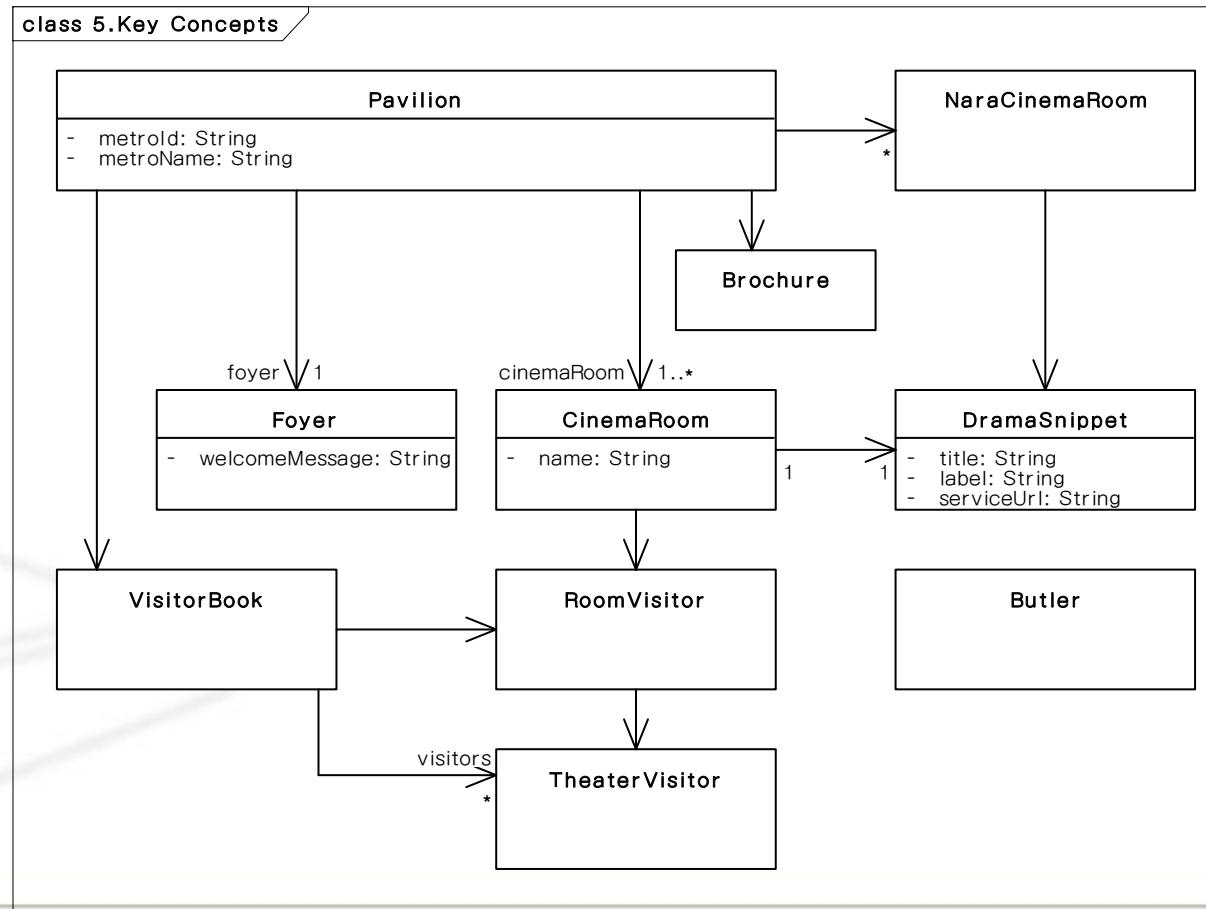
# 모델링: Event flow/Biz flow

- ✓ 주요 시나리오에 대해 흐름을 표현해 보면 대상 업무의 내용과 특성을 보다 명확하게 파악할 수 있습니다.
- ✓ 뿐만 아니라, 시나리오에 참여하는 다양한 이해관계자와 시스템/서비스 들의 R&R을 이해할 수 있습니다.
- ✓ 흐름은 전체 시스템을 가로지르는 것일 수도 있고, 서브시스템이나 서비스 내부의 흐름일 수도 있습니다.



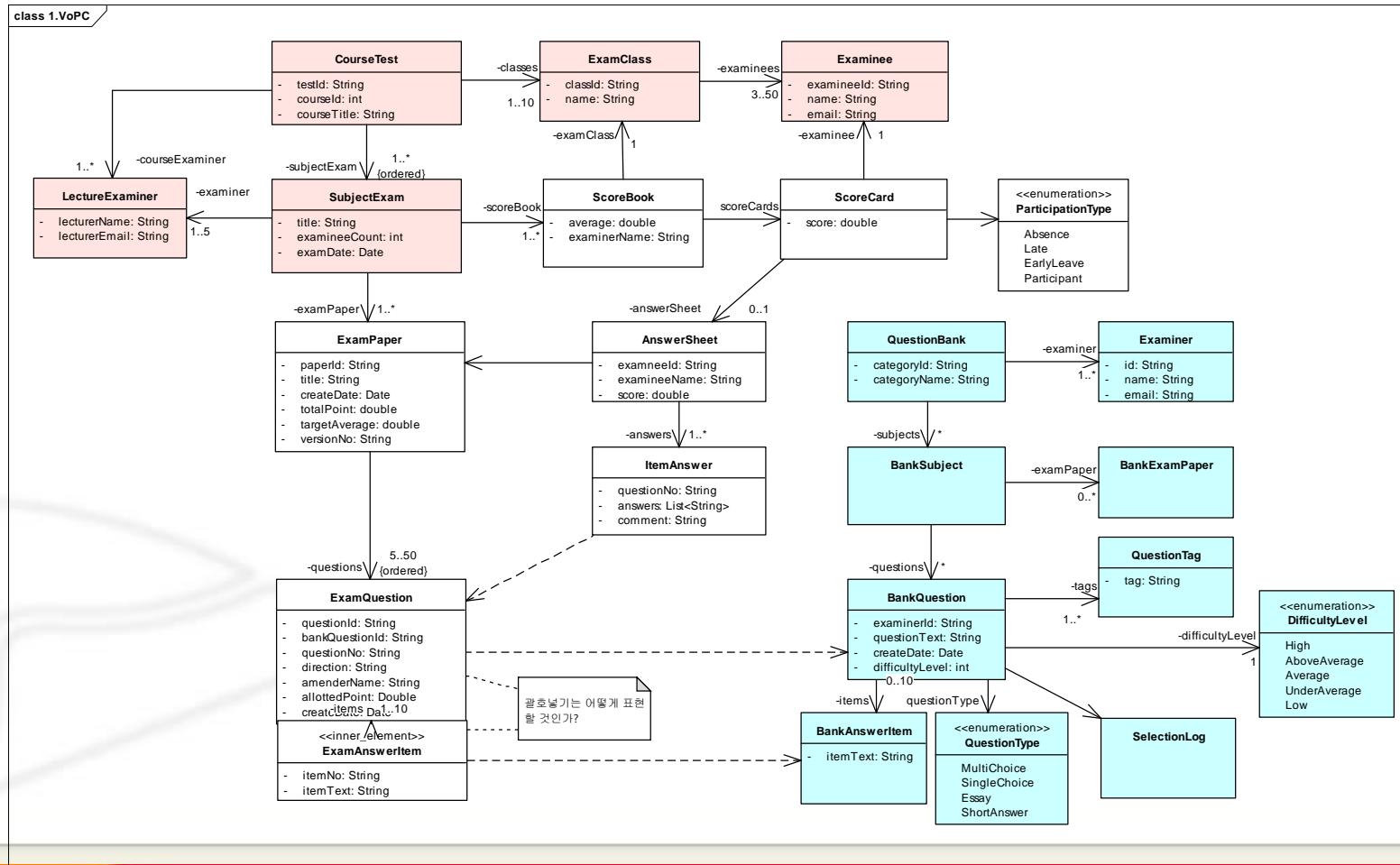
# 모델링: 핵심 개념

- ✓ 본격적으로 Use Case 분석에 들어가기 전에 시스템 전반 또는 특정 서비스에 존재하는 개념을 식별합니다.
- ✓ 해당 컨텍스트 또는 도메인에서는 어떤 개념들이 존재하는지에 대해 미리 정의하고 공유합니다.
- ✓ 핵심 개념의 주제를 파악하는 첫 번째 단계입니다. 신중하게 접근하여야 하며, 때로는 메타포를 사용합니다.



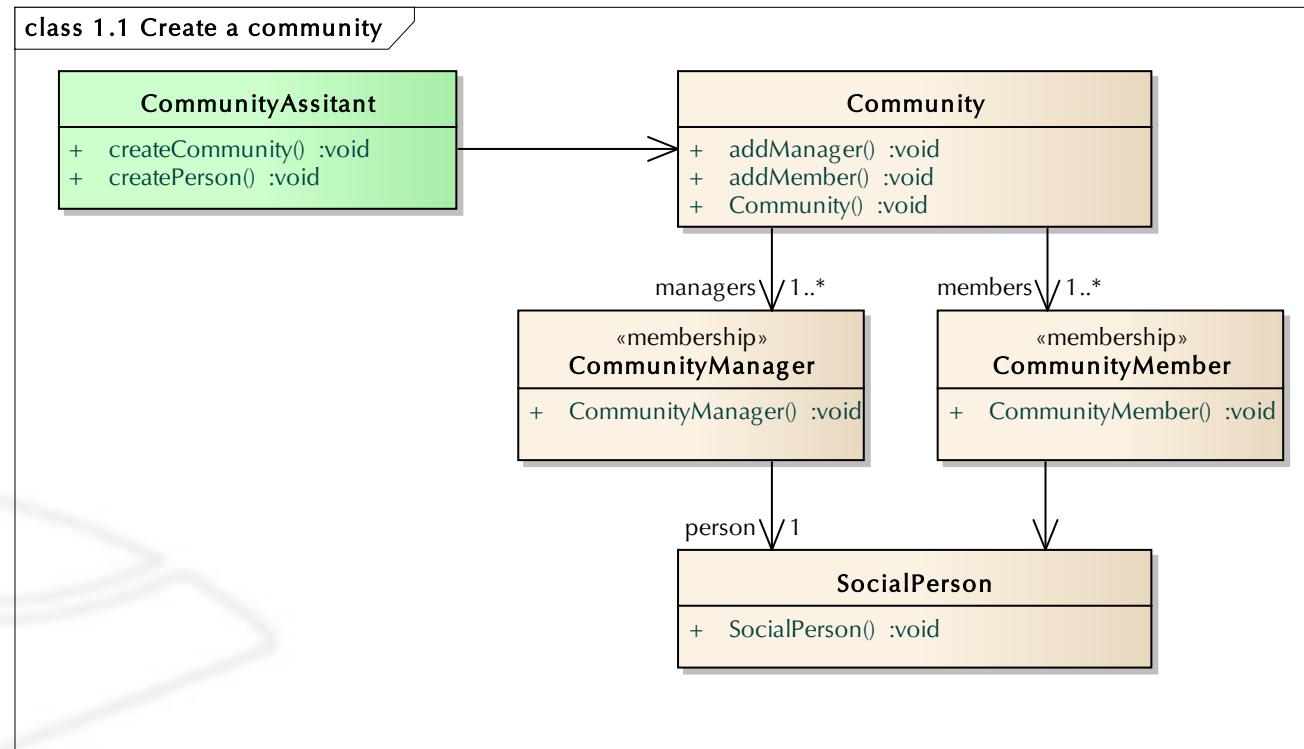
# 모델링: VoPC

- ✓ 시스템/서브시스템/마이크로서비스 등에 참여하는 모든 클래스를 한 다이어그램에 표현합니다.
- ✓ 전체를 볼 수 있어서 좋습니다. 물론 각 클래스는 서로 컴포넌트/서비스 소속입니다.
- ✓ 이 경우, 주로 엔티티 클래스들만 모아 둡니다. 필요할 경우, 로직 클래스에 대한 VoPC를 그릴 수 있습니다.



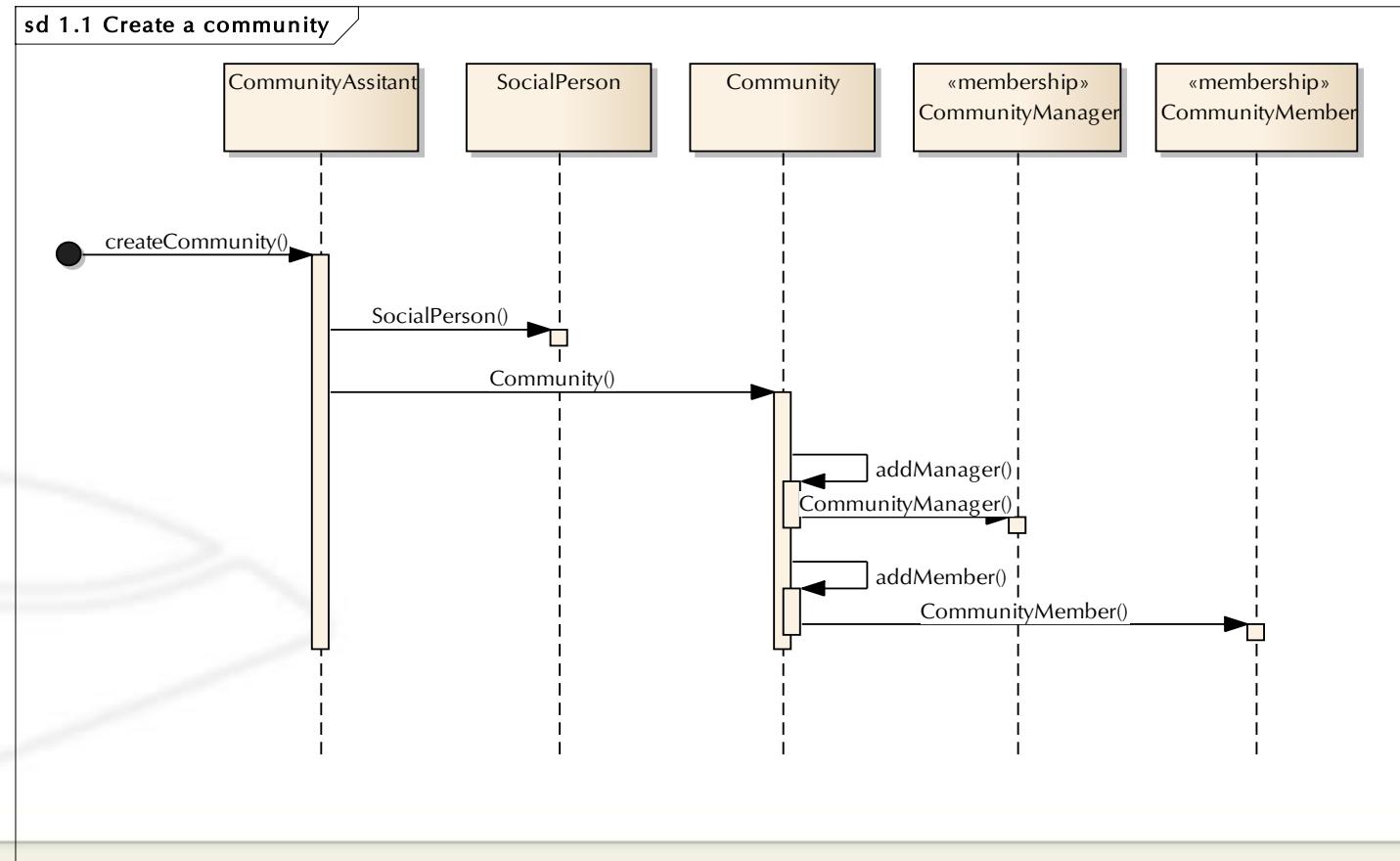
# 모델링: Use Case 분석(1/2)

- ✓ 하나의 유스케이스는 시스템의 동적인 측면을 표현합니다.
- ✓ 유스케이스의 흐름에 참여하는 객체들을 식별하고 관계를 정의합니다. → 클래스 다이어그램
- ✓ 도메인 객체 모델을 편하게 하기 위해 도우미(helper) 객체를 사용합니다.
- ✓ CommunityAssitant, CommunityRepository 등과 같은 객체를 활용하면 편리합니다.



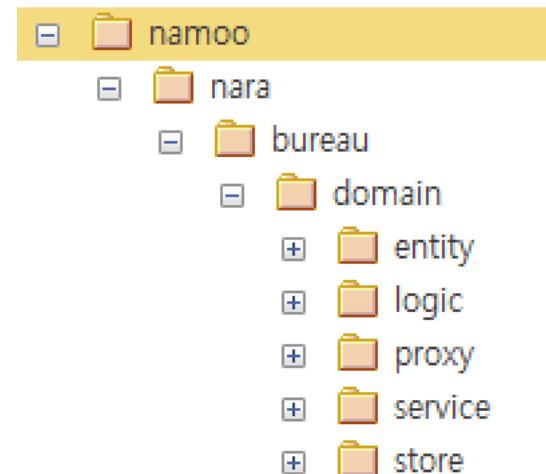
# 모델링: Use Case 분석(2/2)

- ✓ 유스케이스의 흐름인 동적인 측면을 표현하는 다이어그램입니다. → 시퀀스 다이어그램
- ✓ 객체들이 어떻게 서로 상호작용하는지를 표현합니다.
- ✓ 객체의 역할과 책임(Role and Responsibility) 설정에 따라 서로 다르게 협업합니다.
- ✓ 아래의 예는 Community 객체에 역할을 많이 부여한 상태입니다.



# 모델링: Namespace 배정

- ✓ 아키텍처 팀의 컴포넌트 설계 담당자는 패키지 구조를 설계하여 도메인 모델러에게 넘겨 줍니다.
- ✓ 도메인 모델러는 모델링의 마지막 단계로 해당 패키지로 설계한 클래스를 이동합니다.
- ✓ 이동이 끝나면 이제 소스코드 생성을 할 준비가 되어 있습니다. 물론 대부분의 경우 생성없이 직접 작업합니다.



## 7. 실습

---

- ❖ 모델링을 해야 하는 이유를 생각해 봅니다.
- ❖ 모델링이 Thinking Process 인 이유를 이해합니다.
- ❖ 모델링을 해야 한다면, 올바른 절차는 무엇인지 알아 봅니다.

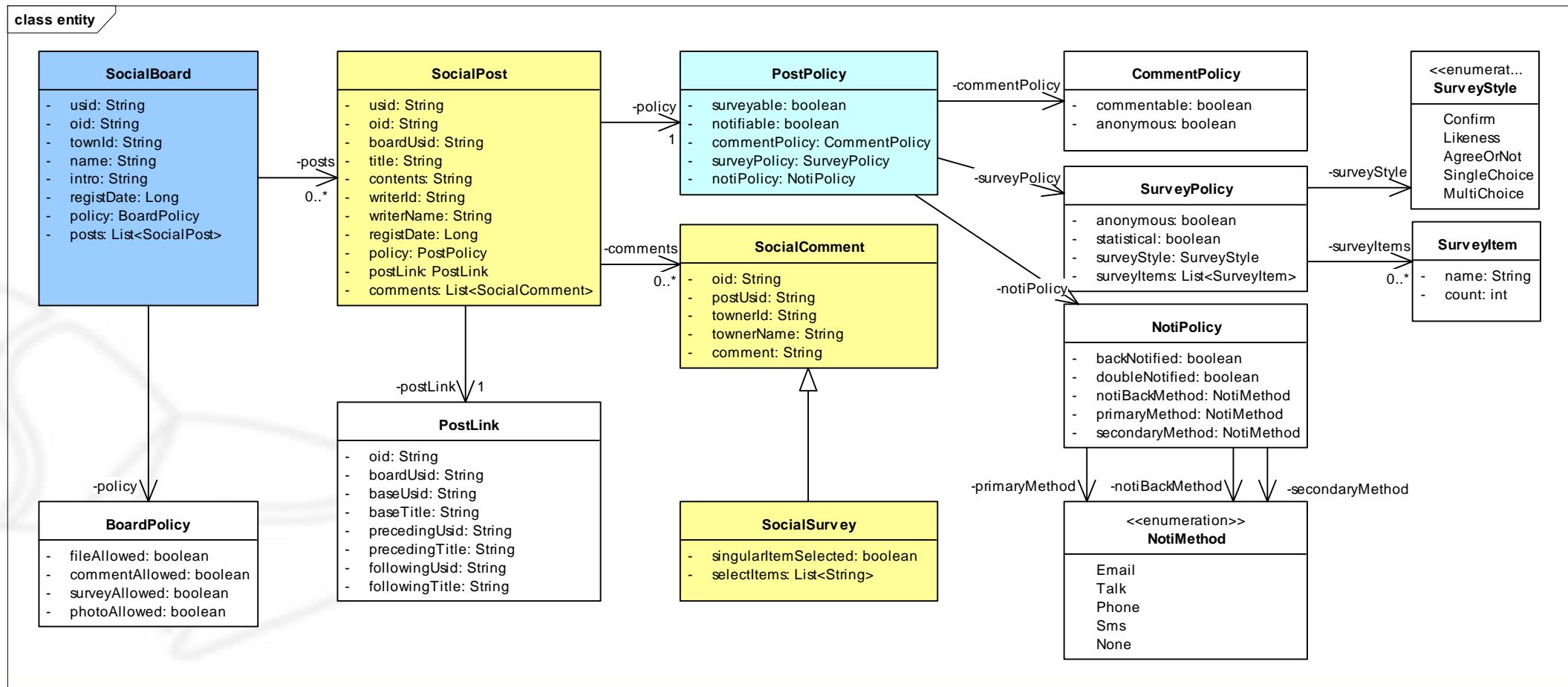
- ✓ 나무보드
- ✓ 기술과 역할
- ✓ 리팩토링과 모델

## 실습 #1: 나무보드 (1/2)

- ✓ 페이스 북의 홈페이지 같은 소셜 보드를 만들려고 합니다.
  - 게시글을 쓸 수 있습니다.
  - 게시글에 댓글을 달 수 있습니다.
  - 게시글에서 간단한 설문조사를 할 수 있습니다. 확인, 찬성/반대, 택일
  - 중요 게시글은 팀원들에게 통지를 할 수 있습니다.
  - 팀원 모두가 게시글을 읽었을 때, 관리자에게 결과를 전달할 수 있습니다.
- ✓ 도메인 모델링 → 코딩으로 가는 일련의 경로를 확인해 봅니다.
- ✓ 시연한 내용을 중, 일부 시나리오를 실습을 통해 확인합니다.

# 실습 #1 : 나무보드 (2/2)

- ✓ 시연한 내용을 중, 일부 시나리오를 실습을 통해 확인합니다.
- ✓ 결과 보다는 절차에 집중합니다. 어떤 경로를 거쳐서 코드가 나오는지 이해하도록 합니다.
- ✓ 산출물로써의 모델 보다는 생각하는 과정의 일부로써 모델로 이해해야 합니다.



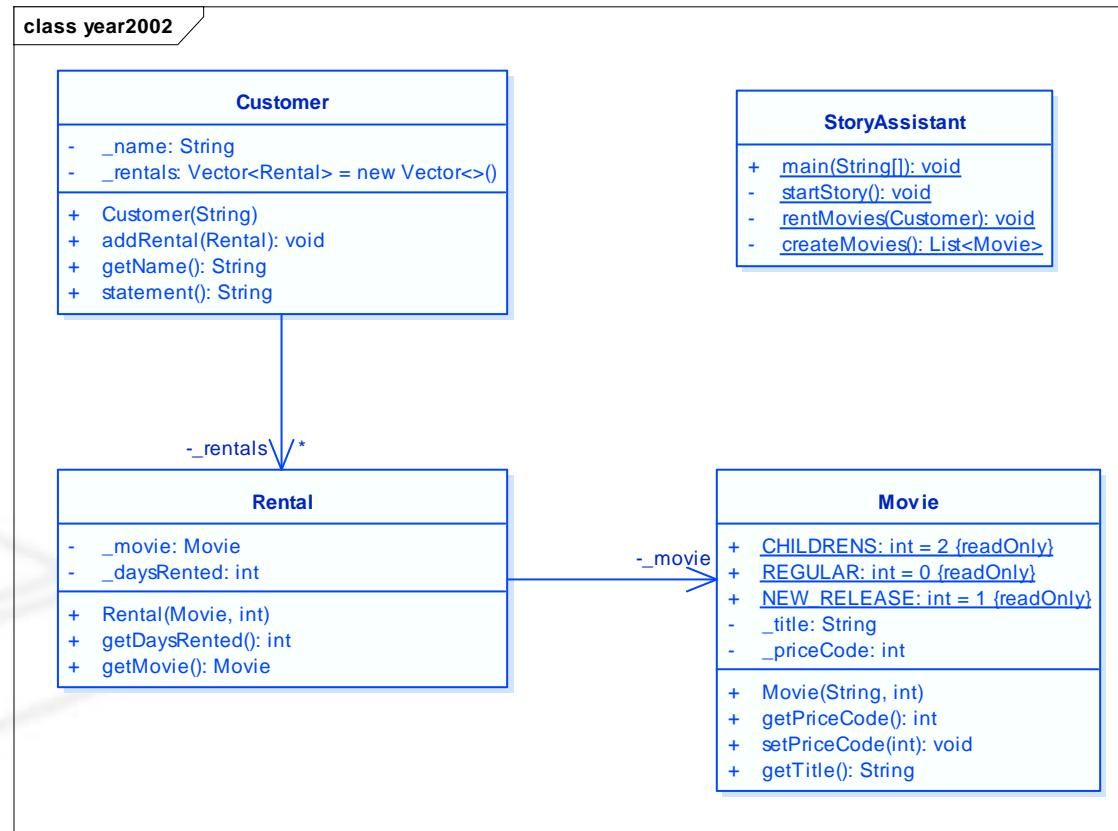
# 실습 #2 : 기술과 역할

- ✓ 역할(Role)과 기술(Skill)을 정리하고, 역량을 갖추기 위해 필요한 기술을 맵 형태로 정의하려고 합니다.
- ✓ 기술과 기술은 각각 역할 명세서, 기술 명세서 형식을 갖추어 세분화 하려 합니다.
- ✓ 기술, 역할, 명세서, 맵 간의 관계를 이해하고 이것을 객체 모델로 표현합니다.

B 카테고리	C 서브카테고리	D 단위기술	E 기준	F 중요도	G SW 개발자(L12)			H I J K L M 프론트 엔드			
					L1	Mandatory	Optional	L1/L2/L3	Mandatory	Optional	L1/L2/L3
언어(L)	프로그램	Java	G39	High	G1+MO(#01)	33		G+MO(#01)	100		
		C/C++	G39	High	G1+MO(#01)		33	G+MO(#01)		100	
		VC++	G39	High				G+MO(#01)		100	
		C#	G39	High	G1+MO(#01)		33	G+MO(#01)		100	
		ABAP	G39	High	G1+MO(#01)		33	G+MO(#01)		100	
		Pro*C	G24	Low				G+O		100	
		Python	G24	Middle							G+MO(#01)
		Delphi	G24	Low							G+MO(#01)
		Power Builder	G24	Low							G+MO(#01)
		Visual Basic	G24	Low							G+MO(#01)
	웹언어	Swift	G39	High							
		Objective C	G39	High							
	운영언어	HTML5	G24	Middle	G1+M	50					G1+M
		HTML/CSS	G24	Middle	G1+M	50					G1+M
		Javascript	G39	High	G1+M	33					G+MO(#01)
		JSP/Servlet	G24	Middle	G1+O		50				G+MO(#02)
		Php	G24	Middle							G+MO(#02)
		Asp	G24	Middle							G+MO(#02)
		HTML SAP UI5	G24	Middle							G+MO(#02)
		Webdyn ABAP	G24	Middle							G+MO(#02)
		Linux Shell	G24	Low				G1+M	50		G1+M
		Ruby	G24	Low							
		Perl	G24	Low							
		Groovy	G24	Low							
		Windows shell	G24	Low				G1+M	50		G1+M
		VBScript	G24	Low							
		Power Shell	G24	Low							
		Unix Shell	G24	Low							
		UML	G24	Middle	G1+M	50		G+M	100		

# 실습 #3 : 리팩토링과 모델

- ✓ 아래 비디오 삽 예제는 유명한 Refactoring 책의 처음에 나오는 예제입니다.
- ✓ 코드 리팩토링과 모델 리팩토링을 병행하면서 얻을 수 있는 최종 결과 모델은 어떤 모습이 될까요.
- ✓ 코드로부터 출발하되, 모델 뷰를 유지하면서 코드를 개선해 봅니다.



✓ 토론

## 감사합니다...

- ❖ 송태국 ([tsong@nextree.co.kr](mailto:tsong@nextree.co.kr))
- ❖ 넥스트리컨설팅(주) 대표이사
- ❖ 넥스트리소프트(주) 부사장
- ❖ [www.nextree.co.kr](http://www.nextree.co.kr)