



Clean Code & Model
(ver 1.2)

목 차

0. 들어가기 전에...
1. Clean Code – 객체 지향
2. Clean Code – 책임 표현
3. Clean Code – 협업
4. 레이어
5. 도메인과 객체 모델링
6. 컴포넌트
7. RESTful API
8. 데이터 처리 변화
9. 프레임워크



0. 들어가기 전에...

-
- 0.1 기술 흐름
 - 0.2 코드, 코드 모델, 코드 명세
 - 0.3 코드 복잡도
 - 0.4 기술 유산
 - 0.5 Clean Code I – Zoom in
 - 0.6 Clean Code II – Zoom out
 - 요약

0.1 기술 흐름(1/7) – OOAD,CBD

- ✓ 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 극복하는 과정에서 컴포넌트 기반 설계로 발전하였습니다.



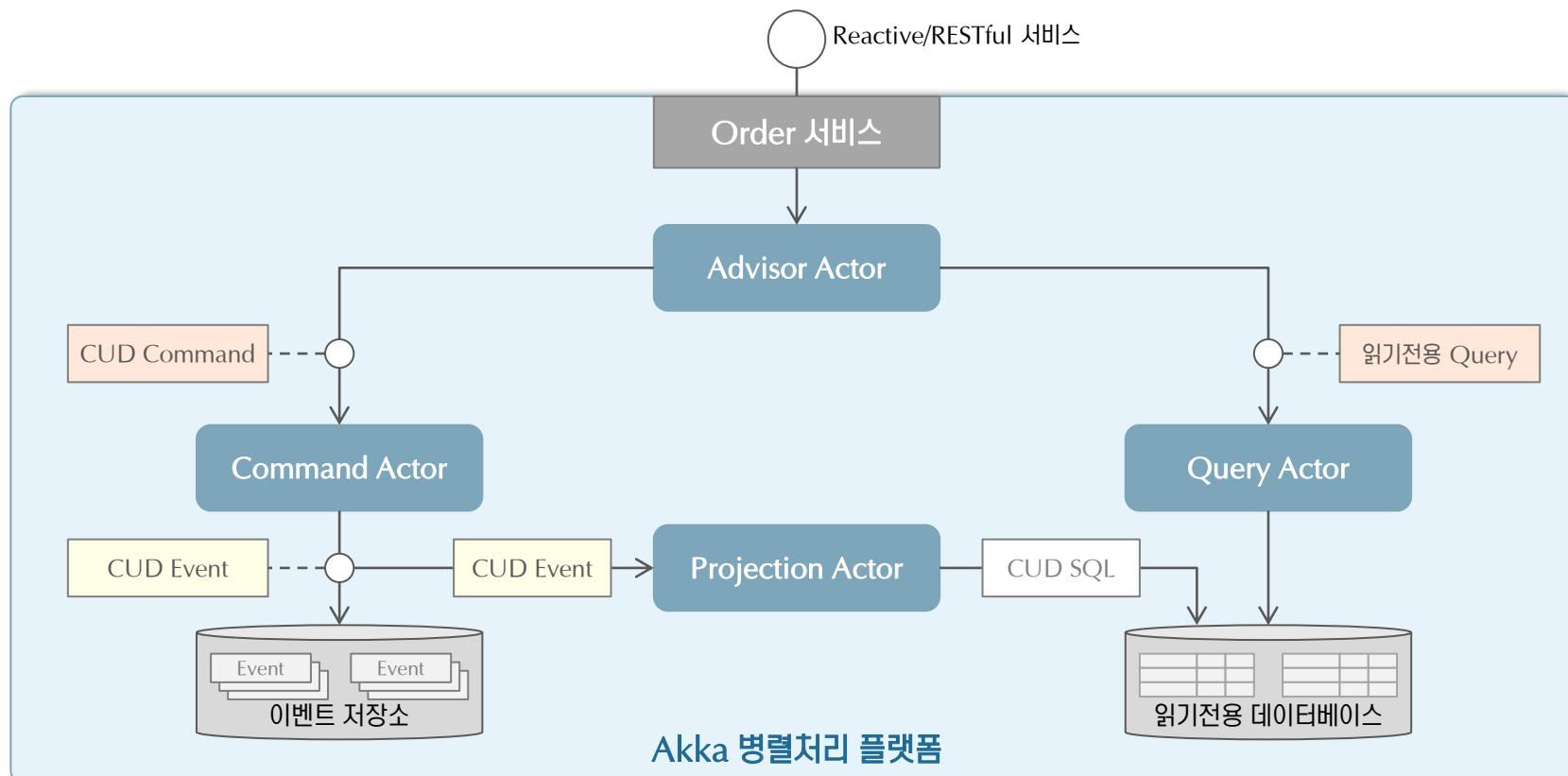
0.1 기술 흐름(2/7) – SOA, MSA

- ✓ 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.



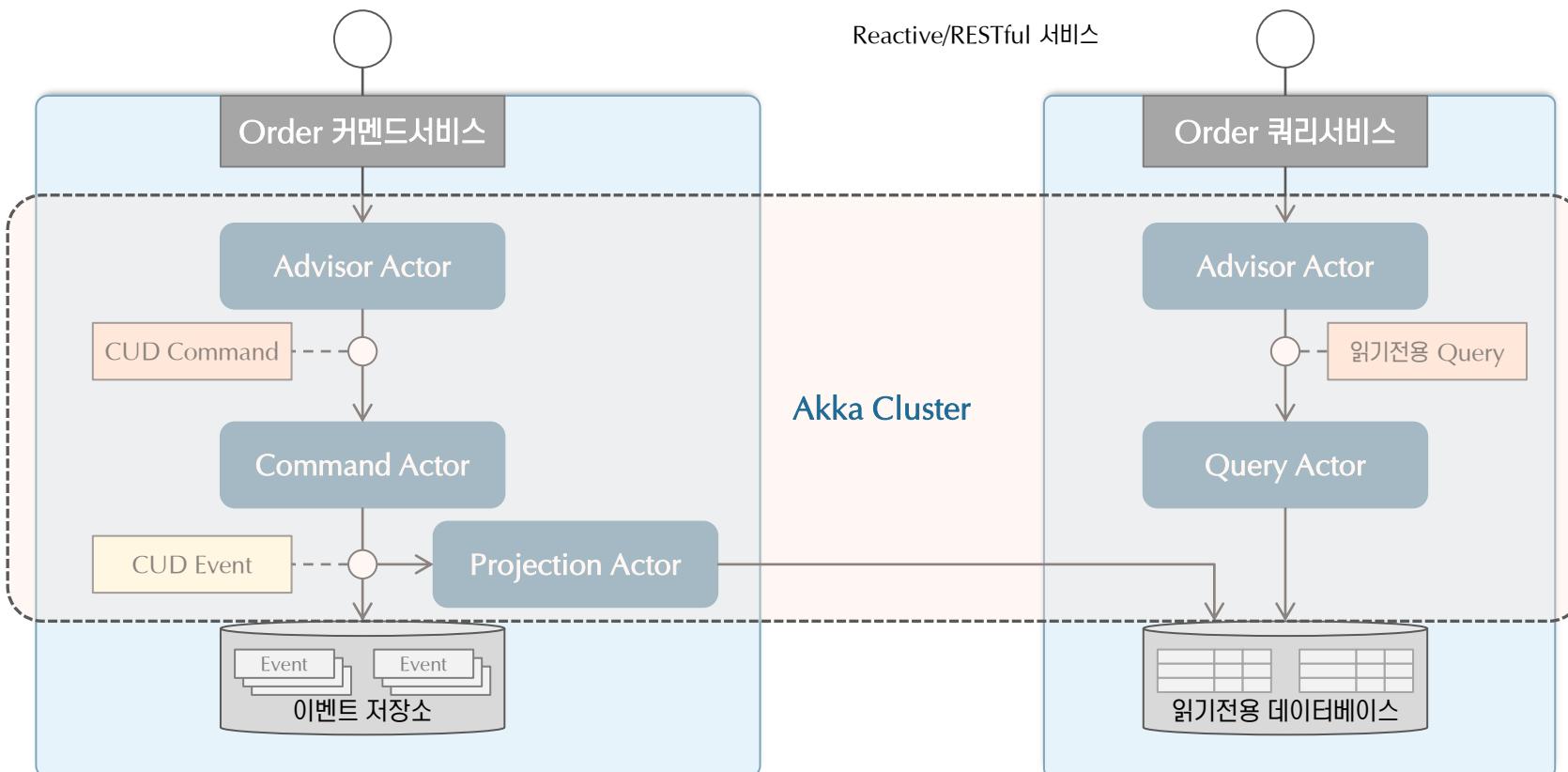
0.1 기술 흐름(3/7) – MSA (CQRS, Event Sourcing) 1

- ✓ IoT 디바이스, 웨어러블 디바이스로부터 끝임없이 쏟아져 들어오는 데이터를 처리할 수 있어야 합니다. ← Reactive, stream
- ✓ 멀티 코어 환경에서 효율적인 동시성 처리를 위한 병렬 프로그래밍 모델이 필요합니다. ← Actor 모델
- ✓ 잠금을 바탕으로 하는 ACID 트랜잭션은 더 이상 유효하지 않습니다. ← BASE 트랜잭션
- ✓ 업데이트가 아닌 이벤트를 축적하는 방식으로 극단적인 가용성 환경에 대응하여야 합니다. ← Event sourcing



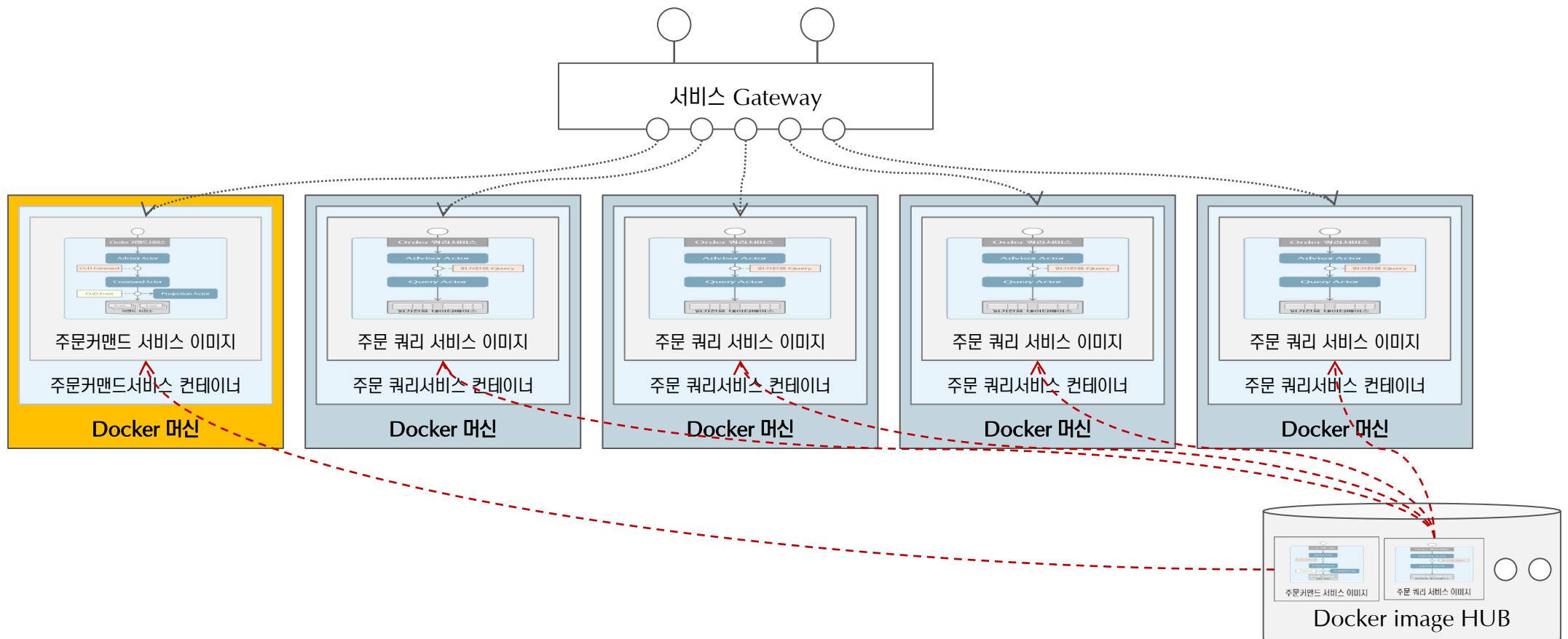
0.1 기술 흐름(4/7) – MSA (CQRS, Event Sourcing) 2

- ✓ 잘 설계한 서비스는 필요에 따라 분리, 결합이 가능합니다. 그리고 플랫폼은 이 기능을 지원합니다.
- ✓ 커맨드 서비스는 호출 횟수는 적지만 한 번 호출에 시간이 많이 걸립니다.
- ✓ 쿼리 서비스는 호출 횟수가 절대적으로 많으며, 많은 사용자 빠른 조회 성능이 필요합니다.
- ✓ Scalability 요구가 있는 곳에서는 Command 서비스와 Query 서비스를 분리하여 구현합니다.



0.1 기술 흐름[5/7] – MSA [CQRS, Event Sourcing] 3

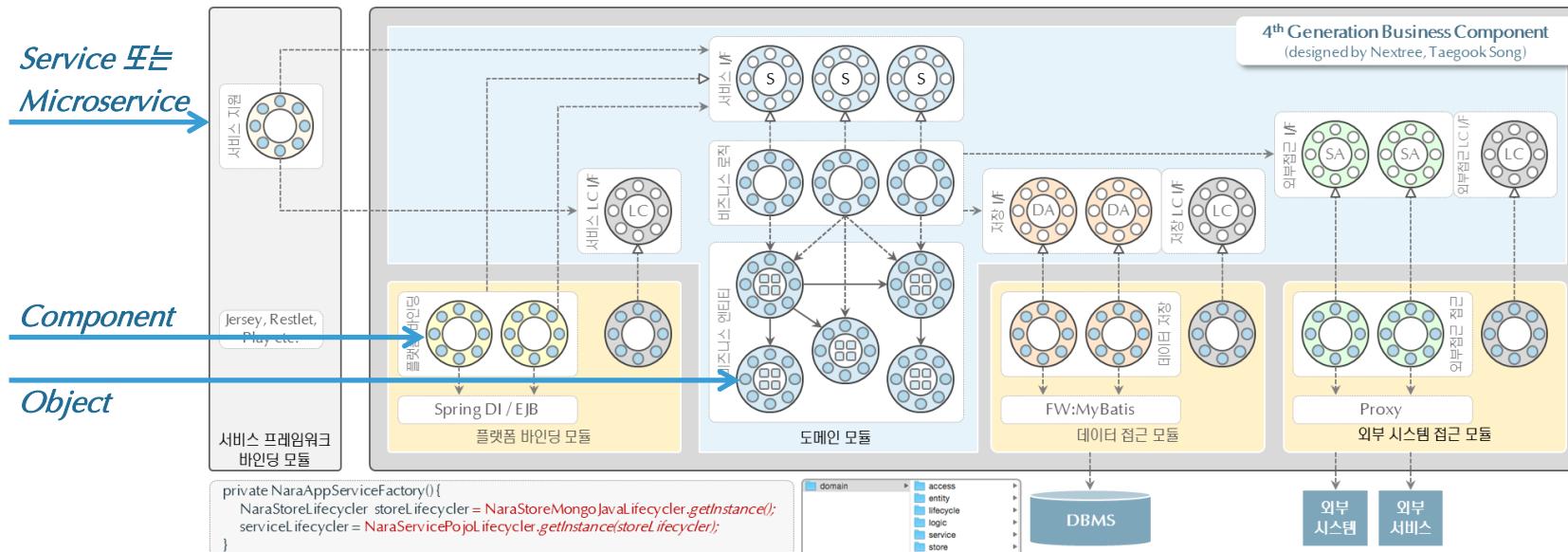
- ✓ 커맨드 서비스는 소수의 서비스 인스턴스(또는 dockerized image)가 필요하고, 쿼리 서비스는 여러 대가 필요합니다.
- ✓ 필요에 따라서 서비스 인스턴스 숫자를 자동으로 늘리고 줄일 수 있어야 합니다. ← scale-out
- ✓ 클라우드 환경에서 기존의 인스턴스에 배포하는 것이 아니라 도커 컨테이너에 미리 준비한 서비스 이미지를 배포합니다.
- ✓ 이제 PODA(Package Once Deploy Anywhere) 환경이 되었습니다. ← Write Once Run Anywhere와 비교



0.1 기술 흐름(6/7) – 서비스 close-up 1

- ✓ 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.

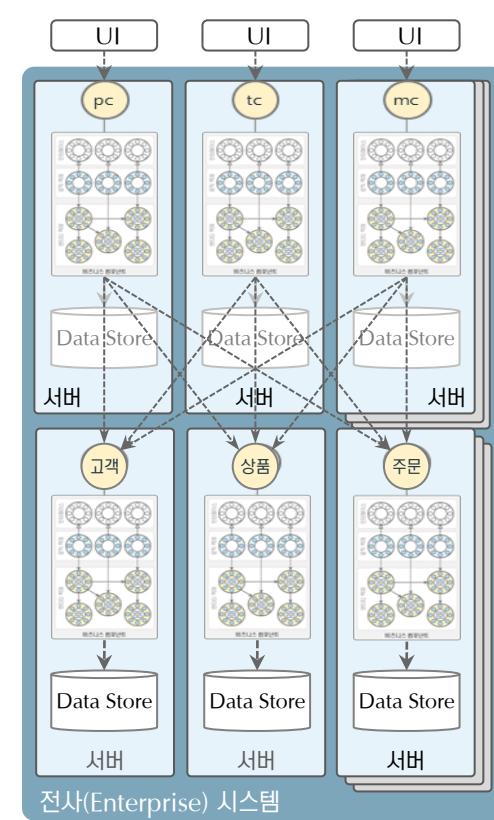
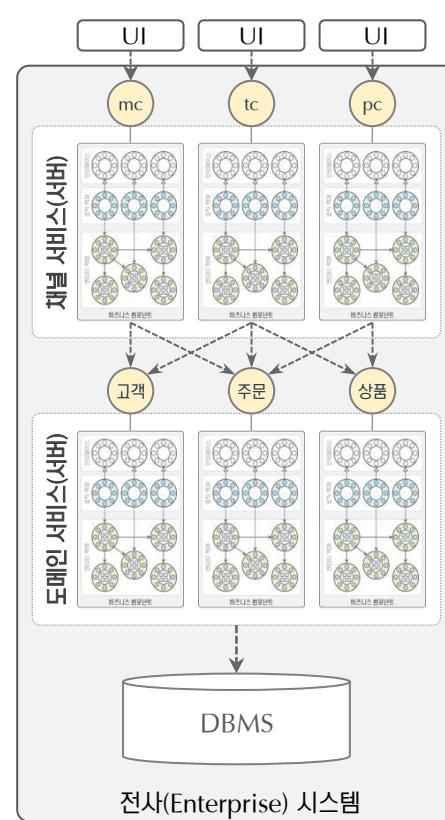
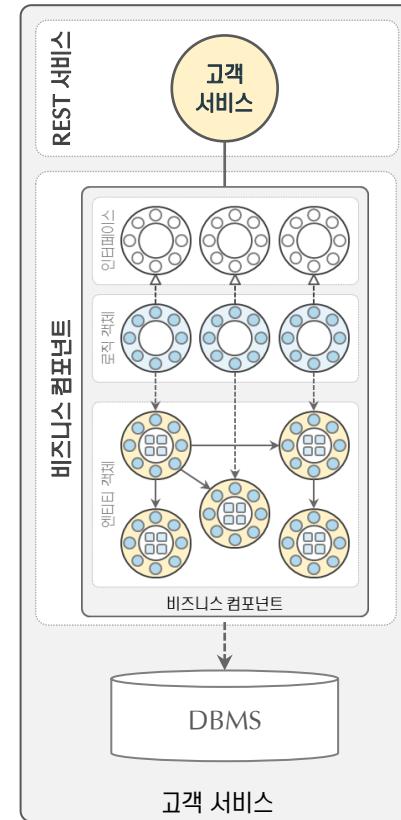
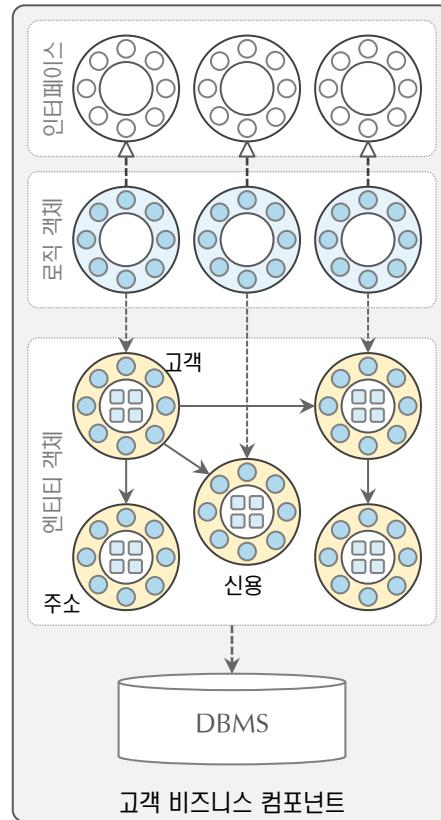
[4세대 비즈니스 컴포넌트와 서비스]



0.1 기술 흐름(7/7) – 서비스 close-up 2

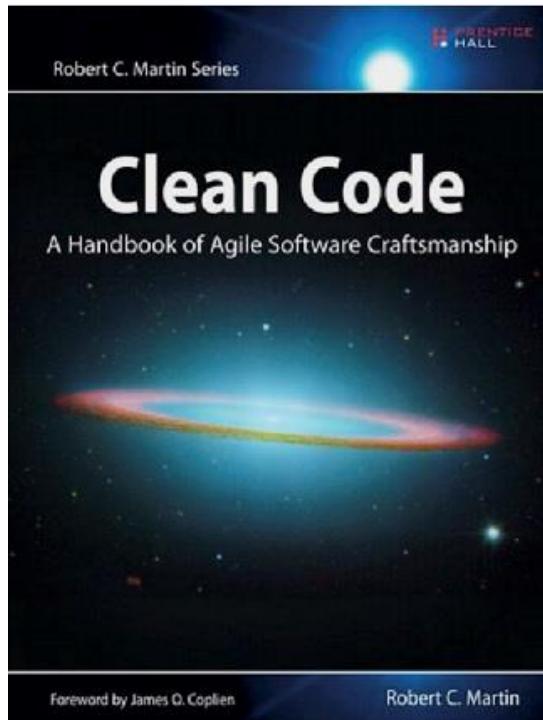
- ✓ 객체 지향 프로그래밍 언어 출현 이후, 기술의 흐름은 모두 객체를 기반으로 하고 있습니다.
- ✓ 컴포넌트는 객체들의 질서정연한 구조를 의미하고, 서비스는 컴포넌트를 사용하는 방법을 의미합니다.
- ✓ CBD → SOA → Microservices 로의 흐름 내내, 객체로 내부를 채우고 있습니다.

```
public class TdDiffNodeIterator {  
    private int nodeSize;  
    private int currentIndex;  
    private List<TdDiffNode> attrNodeList;  
    ...  
    public TdDiffNodeIterator(TdDiffNode attrNode) {  
        this.attrNodeList = initNodeList(attrNode);  
        this.currentIndex = 0;  
    }  
  
    public int size() {  
        // ...  
        return attrNodeList.size();  
    }  
  
    public boolean hasNext() {  
        // ...  
        if (currentIndex < nodeSize) {  
            return true;  
        }  
        return false;  
    }  
  
    public TdDiffNode next() {  
        // ...  
        if (!hasNext()) {  
            return attrNodeList.get(currentIndex++);  
        } else {  
            return null;  
        }  
    }  
  
    public class TdDiffNodeIterator {  
        ...  
        private int nodeSize;  
        private int currentIndex;  
        private List<TdDiffNode> attrNodeList;  
        ...  
        public TdDiffNodeIterator(TdDiffNode attrNode) {  
            this.attrNodeList = initNodeList(attrNode);  
            this.currentIndex = 0;  
        }  
  
        public int size() {  
            // ...  
            return attrNodeList.size();  
        }  
  
        public boolean hasNext() {  
            // ...  
            if (currentIndex < nodeSize) {  
                return true;  
            }  
            return false;  
        }  
  
        public TdDiffNode next() {  
            // ...  
            if (!hasNext()) {  
                return attrNodeList.get(currentIndex++);  
            } else {  
                return null;  
            }  
        }  
    }  
}
```



0.2 코드, 코드 모델, 코드 명세 1

- ✓ 이 강좌는 “Clean Code”의 저자 로버트 마틴의 생각과 방향에 전적으로 동의합니다.
- ✓ 모델이 중요하다고 생각하는 분들께 코드가 바로 모델이라고, 명세서가 중요하다가 생각하는 분들께 코드는 명세라고 이야기 합니다. 덧붙이자면 소프트웨어 개발 프로젝트에서 코드는 처음이자 마지막이 되는 가장 중요한 산출물입니다.
- ✓ 프로젝트 결과물에서 가장 덜 중요한 것 하나씩 없애라고 했을 때, 마지막에 남는 것은 당연히 코드입니다.



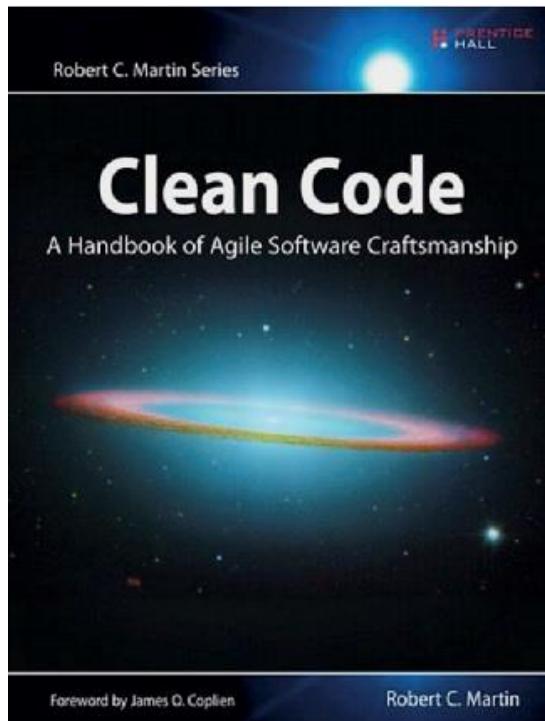
어떤 사람들은 “코드”에 대한 책이 시대에 뒤떨어진 것이라고 주장합니다. 코드는 더 이상 이슈가 될 수 없으며, 코드 대신에 모델과 요구 사항에 더 집중해야 한다고 합니다. 실제로 어떤 사람들은 코드의 종말이 가까이 왔으며, 코드란 작성하는 것이 아니라 자동으로 생성될 것이라고 주장합니다. 비즈니스 담당자들이 명세서로부터 프로그램을 생성하므로 프로그래머는 더 이상 필요하지 않을 것이라고 이야기 합니다.

이건 말도 안되는 소리입니다. 우리는 절대로 코드를 없앨 수 없습니다. 코드는 요구 사항을 자세하게 표현하기 때문입니다. 상세 수준에서 세부 내용을 생략하거나 추상화할 수 없습니다. 반드시 명세되어야 합니다. 요구 사항을 그 상세 수준에서 명세하고, 기계가 그것을 실행할 수 있도록 하는 활동이 프로그램 작성입니다. 그 명세가 바로 코드입니다.

Clean Code, 로버트 마틴

0.2 코드, 코드 모델, 코드 명세 2

- ✓ 코드를 깔끔하게(clean) 유지하는 것 만이 개발 생산성을 유지하는 길이라는 것을 알고 있습니다.
- ✓ 그런데 문제는 그 사실을 안다는 것 자체가 해결책이 아니라는 것입니다.
- ✓ 어떤 코드가 깔끔한 것인지 모르고 그런 코드를 작성할 줄 모른다면, 깔끔한 코드를 가질 수 없습니다.
- ✓ 깔끔한 코드를 작성할 수 있기까지 많은 지식과 경험의 치열한 융합 속에 얻은 깨우침 같은 것들이 필요합니다. → Art...



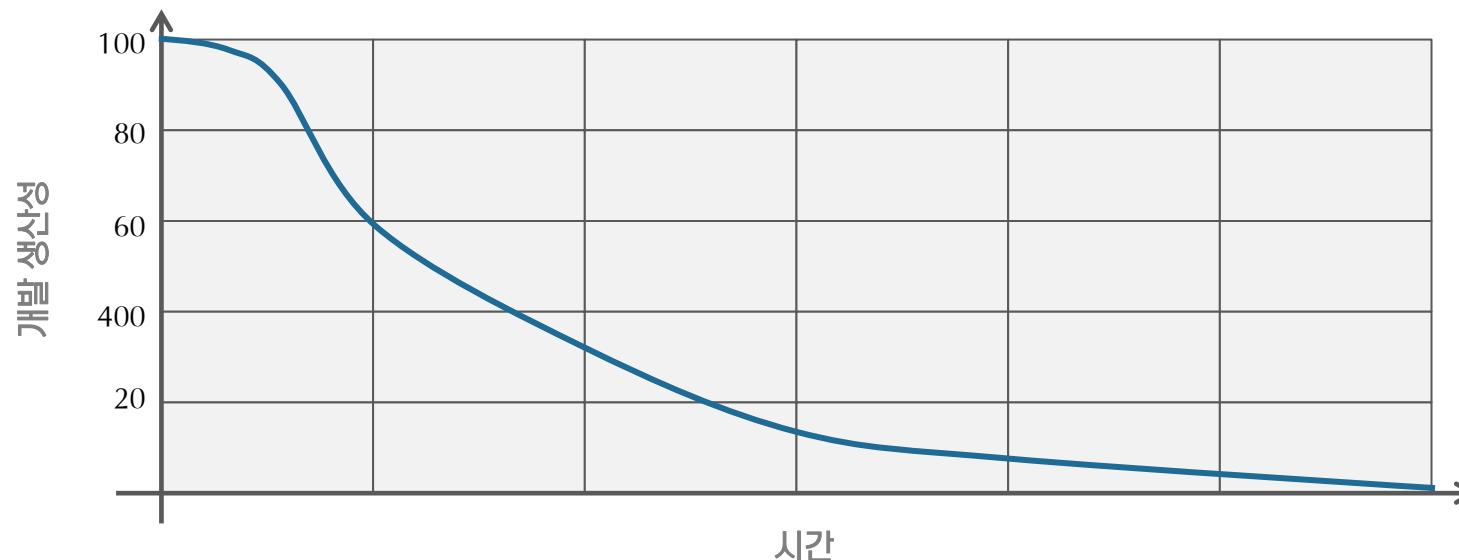
뒤엉킨 코드가 명백한 장애물이라고 믿는다 치자. 그 코드를 깔끔하게 유지 하는 것 이 빨리 가는 유일한 길이라고 인정했다 치자. 그러면 자신에게 “깔끔한 코드를 어 떻게 작성하죠?”라고 물어 봐야 한다. 코드가 깔끔하다는 것이 무엇을 의미하는 지 모른채로 깔끔한 코드를 작성하려고 노력하는 것이 바람직하지 않다.

나쁜 소식은 깔끔한 코드를 작성하는 것은 그림을 그리기와 같다라는 것이다. 사람들은 잘 그린 그림과 못 그린 그림을 알 수 있다. 나쁜 그림으로 부터 좋은 그림을 가져 내는 것과 그림을 잘 그리는 것은 다른 이야기이다. 마찬가지로 뒤엉킨 코드에서 깔끔 한 코드를 가져 낼 수 있다고 해서 깔끔한 코드를 작성할 수 있는 것은 아니다.

Clean Code, 로버트 마틴

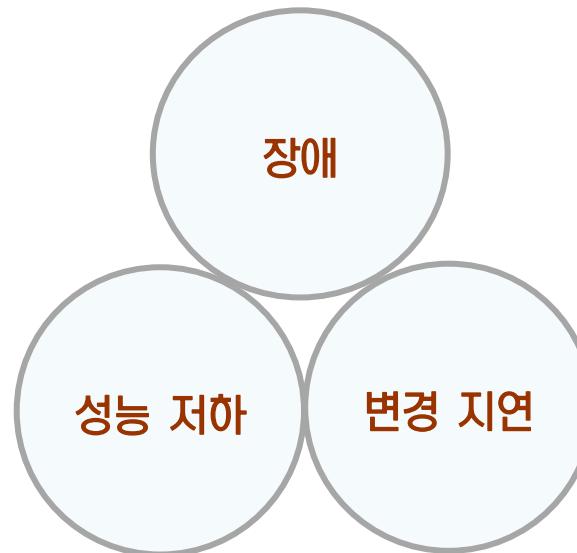
0.3 코드 복잡도(1/7)

- ✓ 시간이 흐를 수록 코드는 점점 더 복잡해지고, 코딩은 점점 더 힘들어집니다.
- ✓ 그 결과를 바로 개발 생산성과 연결되어, 생산성을 지수적으로 떨어뜨립니다.
- ✓ 어느 시점에서는 도저히 생산성이 나지 않는 또는 비용 대비 용납할 수 없는 시점이 오게 됩니다.
- ✓ 그 시점이 흔히 말하는 “신시스템(2000)”, “차세대 (2000중 후반)”, “포스트 차세대(현재)”를 시작해야 하는 때입니다.



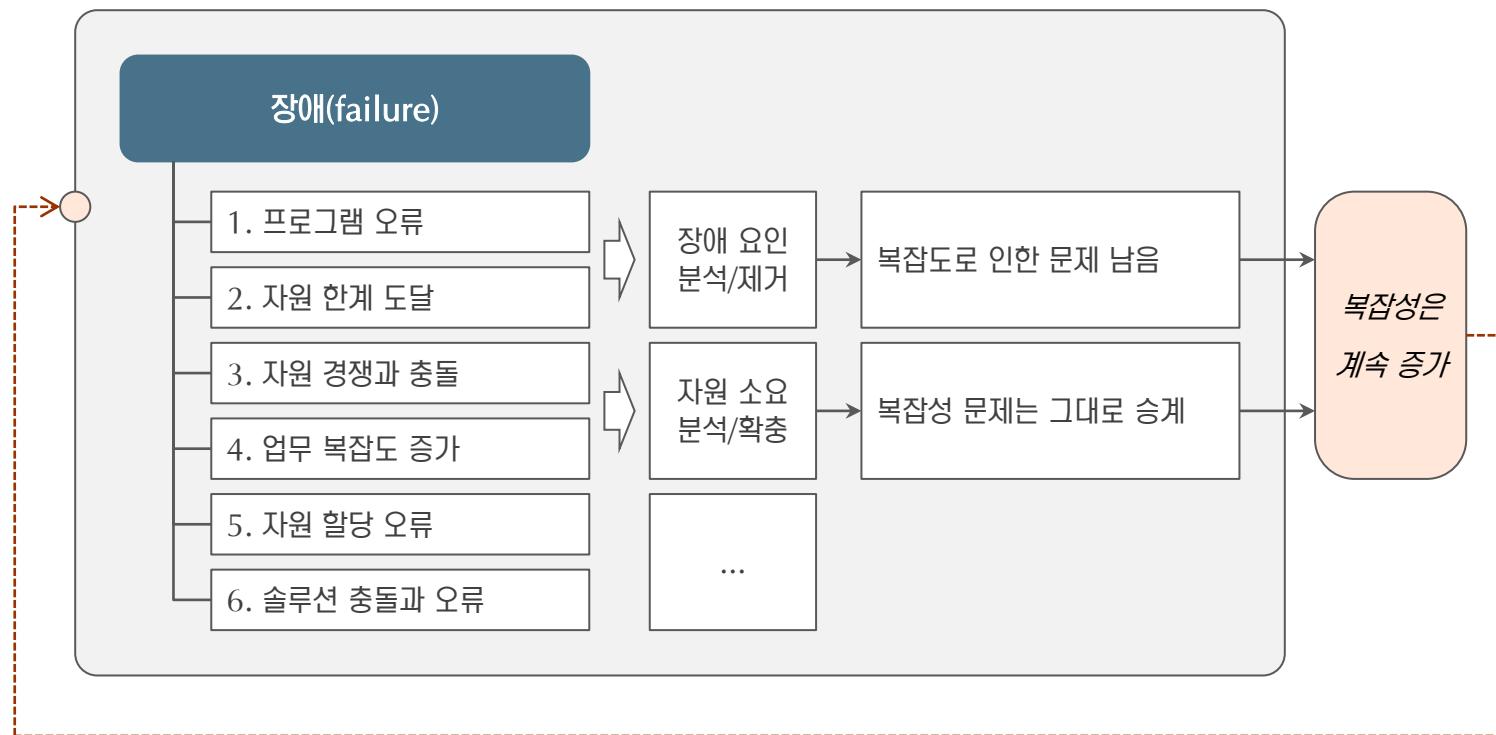
0.3 코드 복잡도(2/7) – 기업 IT 운영의 3대 과제

- ✓ IT 운영 조직의 최대 관심사는 운영 시스템의 장애, 성능, 변경 대응, 세 가지로 볼 수 있습니다.
- ✓ 최근 들어 보안, 규제 등도 관심사이지만, 이러한 이슈는 변경 대응으로 볼 수 있습니다.
- ✓ 세 가지 관심사는 해결할 수 있고, 해결하려는 의지는 있지만, 해결이 불가능한 “언제나 이슈”입니다.
- ✓ 결국 세 가지 이슈를 제대로 해결하지 못하면서 IT 예산은 급증하는 경향이 있습니다. ← 복잡해지기 때문



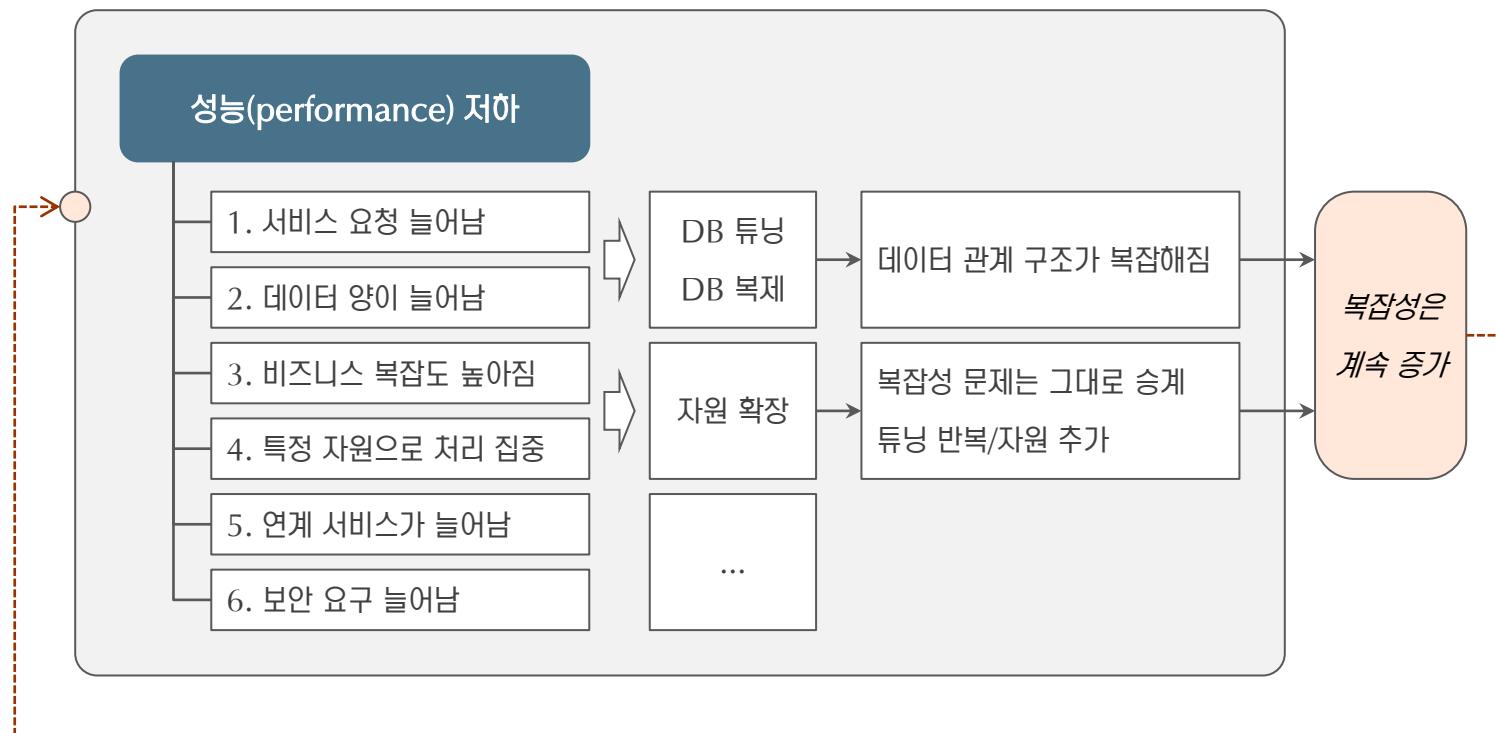
0.3 코드 복잡도(3/7) – 장애(failure)

- ✓ 컴퓨터 시스템을 구성하는 모든 요소 중에 장애로부터 완전히 자유로운 요소는 없습니다.
- ✓ 장애는 피할 수 없습니다. 따라서, 장애로 인한 피해를 줄이는 방법은 장애 대응 역량에 있습니다.
- ✓ 대응 역량은 대응 가능한 환경 구축, 대응 가능한 시스템 구조 설계, 대응 가능한 운영 방식 등을 포함합니다.
- ✓ 장애 대응의 결과의 중요한 부분은 코드 복잡성이 증가했다는 사실입니다.



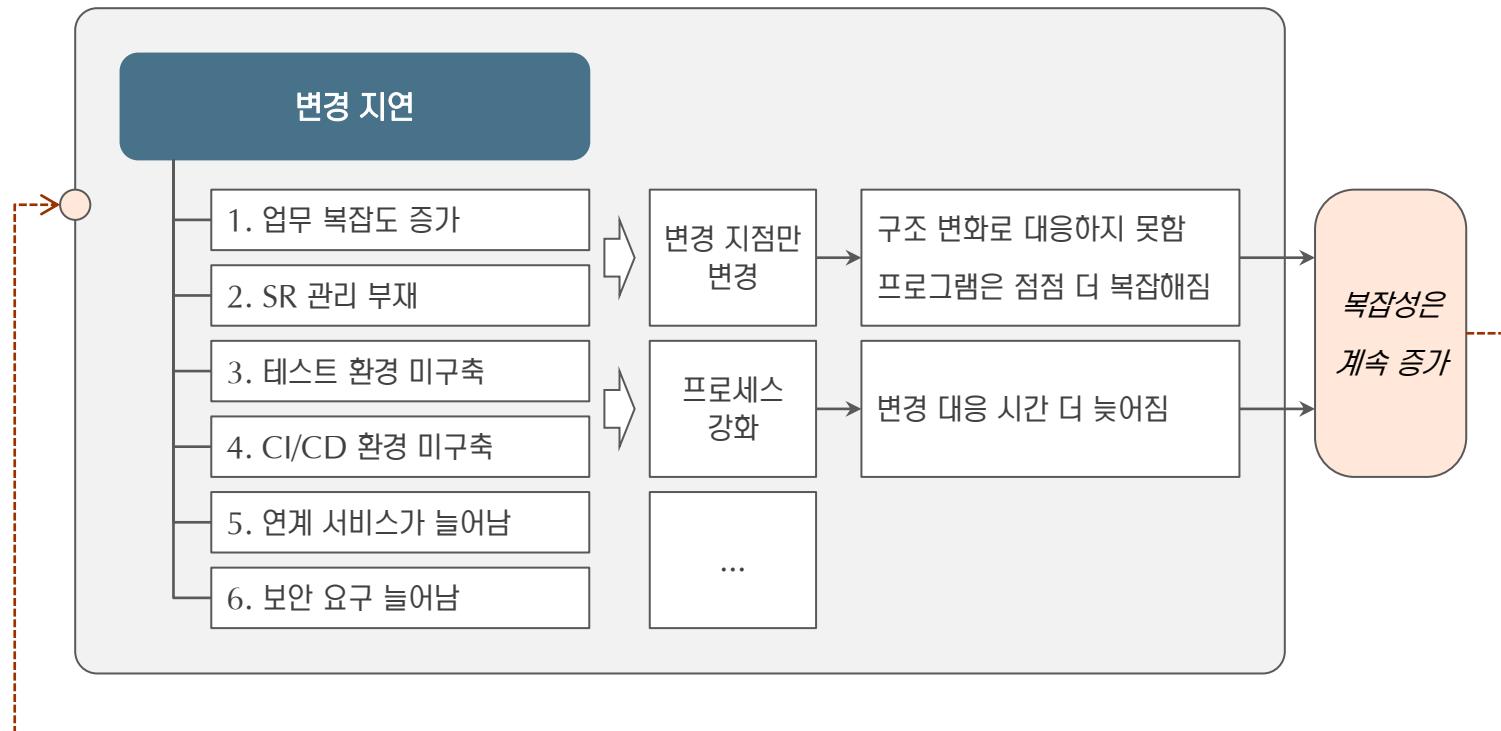
0.3 코드 복잡도(4/7) – 성능(performance)

- ✓ 어느 시스템이나 구축 후 오픈 초기에는 적절한 수준의 성능을 제공하도록 설계합니다.
- ✓ 발전하는 조직일수록 시스템 성능 문제에 빨리 부딪힙니다. 사용자와 데이터가 빨리 늘어나기 때문입니다.
- ✓ 대체로 자원을 최적화하는 방식으로 문제를 해결하지만, 어느 순간부터 scale-up 한계에 부딪히게 됩니다.
- ✓ 성능을 높이기 위해 다양한 튜닝 과정을 거치고, 그 과정에서 다시 복잡성은 증가합니다.



0.3 코드 복잡도(5/7) – 변경 지연

- ✓ 시스템이 복잡해 질수록 비즈니스 변경에 따른 프로그램 변경 대응이 늦어지는 경향이 있습니다.
- ✓ 변경 대응 지연 보다 더 큰 문제는 때로는 변경 자체가 불가능하며, 그에 따라 비즈니스 확장을 방해합니다.
- ✓ 업무 규모 크고 시스템 규모가 클 수록 변경 대응 시간은 지수적으로 지연되는 경향이 있습니다.
- ✓ 지연으로 인한 손실을 감당할 수 없는 지점에 이르는데, 이것은 복잡성이 0에 가까웠음을 의미합니다.



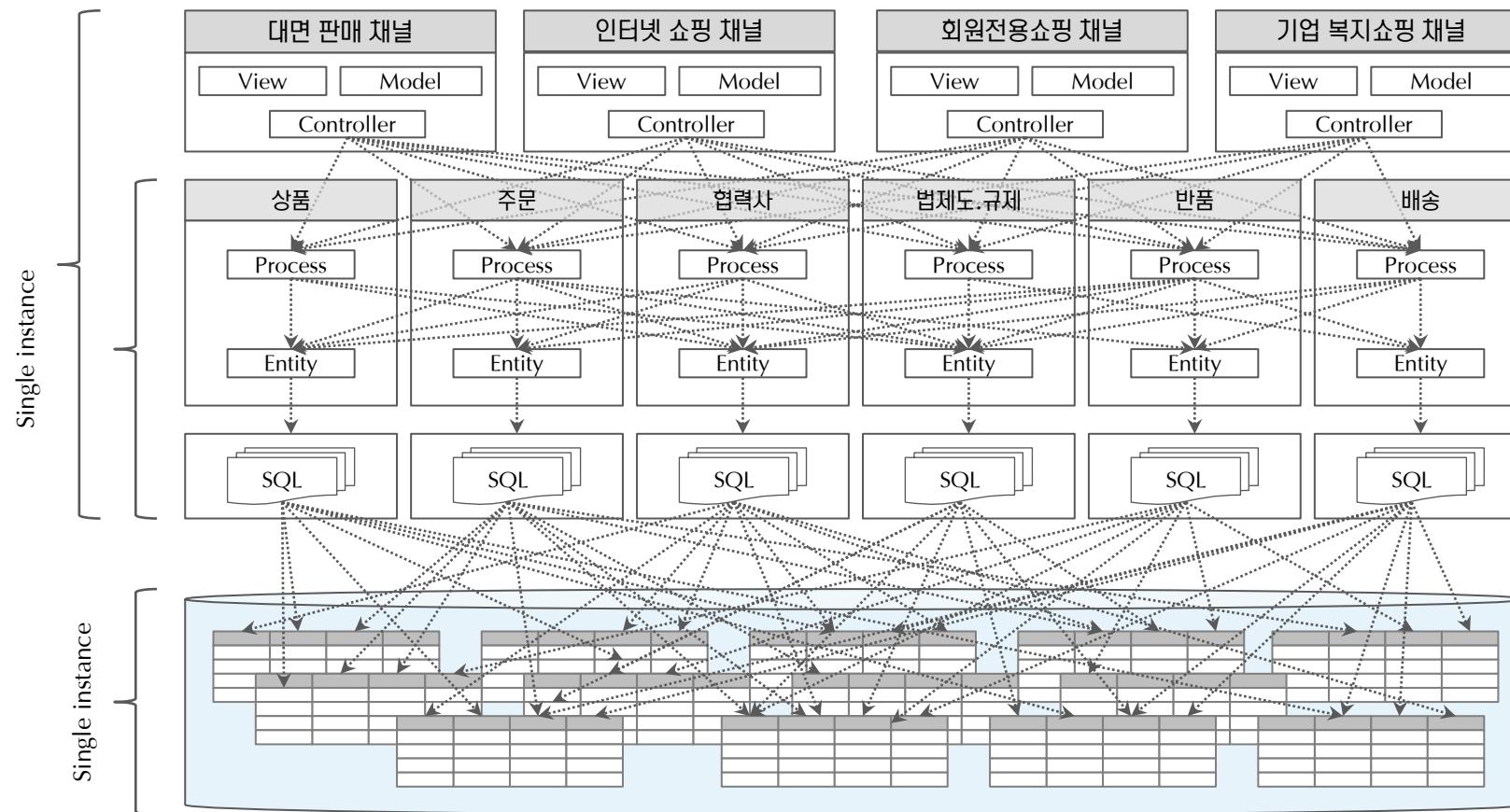
0.3 코드 복잡도(6/7) – 결론 1

- ✓ 세 가지 핵심 이슈를 해결하기 위한 활동이 활발할 수록 시스템의 복잡도는 빠른 속도로 증가합니다.
- ✓ 복잡한 시스템은 다시 장애, 성능 저하, 변경 지연의 또 다른 이유가 됩니다.
- ✓ 시스템 구조 관점에서 낮은 복잡도를 유지하도록 설계하는 것이 “해답”이 될 수 있는 이유입니다.
- ✓ 물론 이러한 설계가 모든 문제를 해결할 수는 없지만, 적어도 문제에 대한 신속한 대응 역량을 제공합니다.



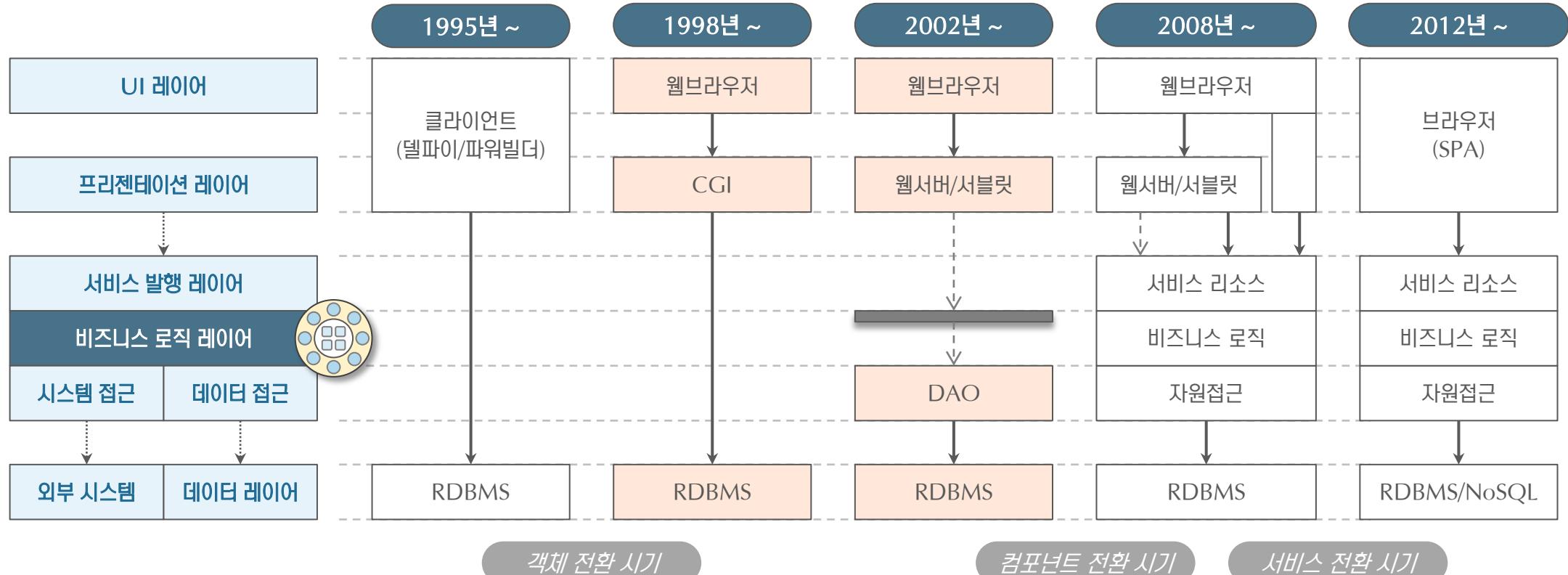
0.3 코드 복잡도(7/7) – 결론 2

- ✓ IT 기반의 서비스를 제공하는 조직이 운영하는 엔터프라이즈 시스템은 거대한 규모를 자랑합니다.
- ✓ 어떤 금융사의 경우, 시스템 배포에만 두 시간 이상 걸릴 정도로 규모가 매우 큽니다.
- ✓ 어떤 컴포넌트에 한 줄이라도 변경을 한다면, 전체를 다시 배포해야 합니다. 이 거대한 시스템의 구조는 그대로 둔 채로 지속적으로 추가하거나 변경하는 방식으로 시스템을 운영하고 있습니다. → 복잡도는 지수적으로 증가합니다.



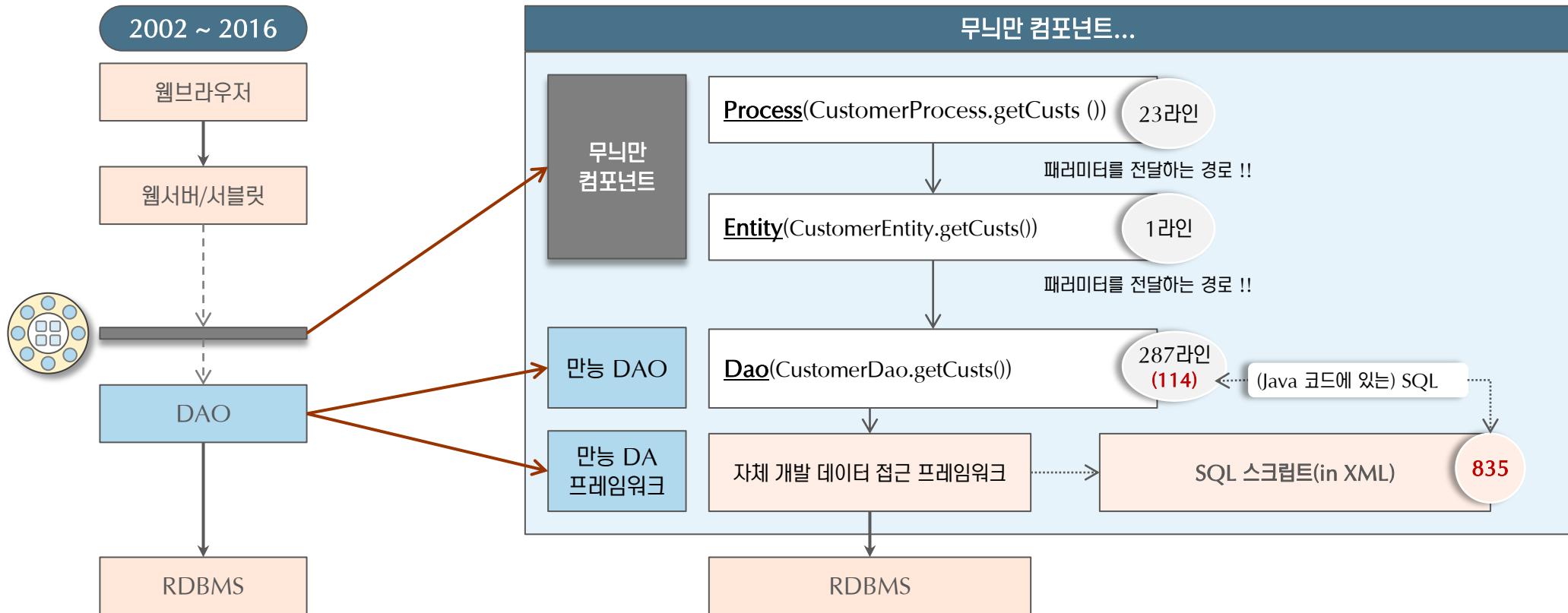
0.4 기술 유산 – 1

- ✓ 기업 컴퓨팅 환경은 2000년 이전 C/S 환경으로부터 시작하여 현재의 MSA까지 발전해 왔습니다.
- ✓ 국내 기업들 대부분 객체로 전환을 하지 못했고, 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, 객체 기술 부재가 가장 큰 원인입니다.



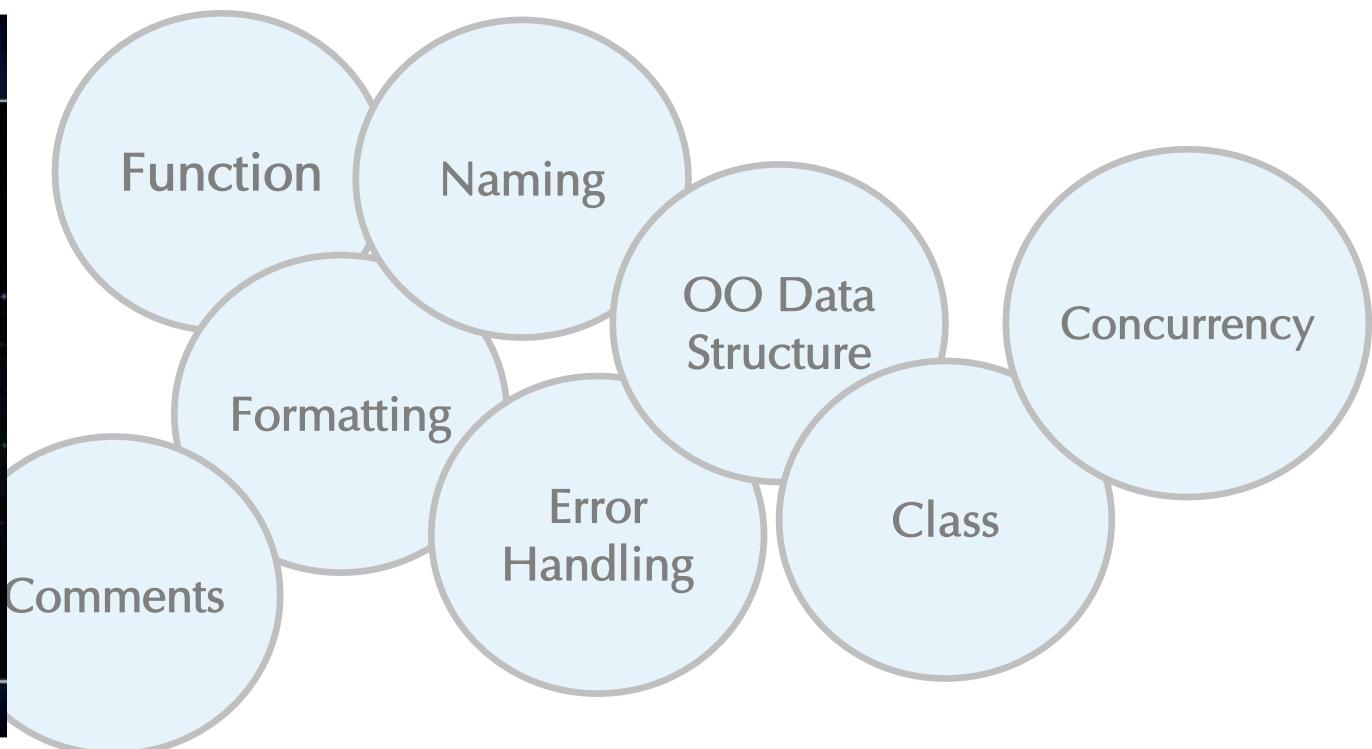
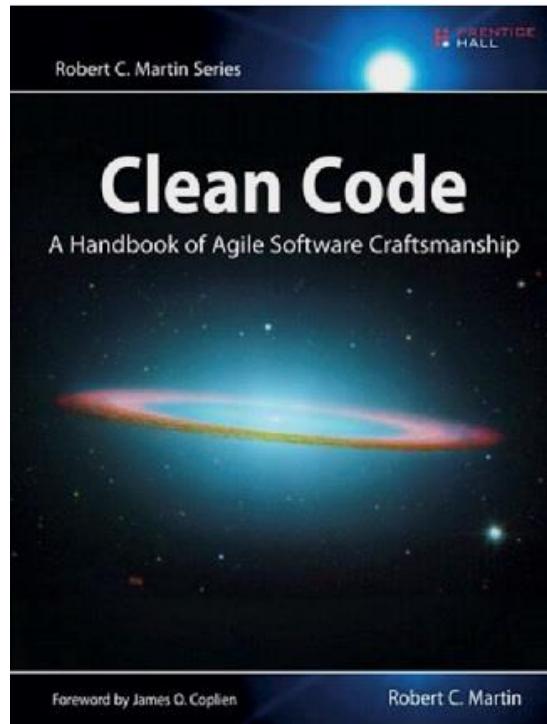
0.4 기술 유산 – 2

- ✓ 객체 모델링 역량이 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 매개변수 전달 경로로 사용할 뿐입니다.
- ✓ 객체가 주로 활동할 공간을 비워 둔 설계 유산 때문에 객체 지향 설계와 프로그래밍이 설 자리를 잃었습니다.



0.5 Clean Code I (1/2)

- ✓ “Clean Code”에서 다루는 내용들은 로버트 마틴이 깔끔한 코드를 위해서 필요하다는 것들을 다룹니다.
- ✓ 이름 붙이기, 함수 구성, 코드 포맷, 주석, 에러 처리 등 코딩의 모든 부분을 다루고 있습니다.
- ✓ 이 강의에서는 이 내용들은 거의 다루지 않습니다. 책읽기를 통해서 충분히 습득 가능한 내용들이기 때문입니다.
- ✓ 그리고 중요한 것은 지금까지 이야기 했던 기업 시스템의 복잡도는 다른 관점의 Clean Code가 필요합니다.



0.5 Clean Code I (2/2)

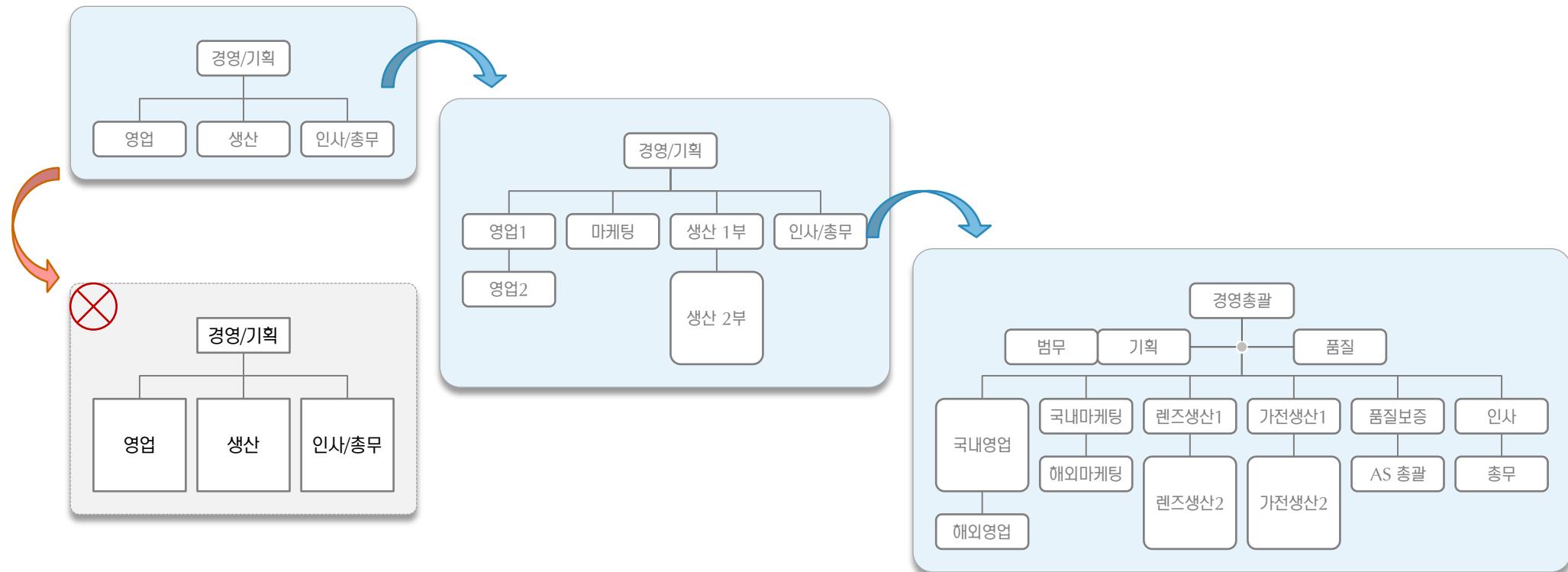
- ✓ 오른쪽 코드와 아래 쪽 코드는 같은 일을 합니다. 물론 오른 쪽 코드는 어떤 부분을 메소드로 처리했습니다.(increaseCapacity(), shiftRightFrom())
- ✓ 어떤 코드가 더 “Clean” 한가요?

```
public void add(int index, T t) {  
    //  
    isValidIndex(index);  
  
    resizing(1);  
  
    //JNI(Java Native Interface – arraycopy 사용.  
    T[] tempArray = (T[]) (new Object[internalMaxSize]);  
  
    if (index == 0) {  
        //맨 처음 입력  
        System.arraycopy(internalArray, 0, tempArray, 1, currentArraySize - index);  
  
    } else if (index == (size() - 1)) {  
        //맨 마지막 입력  
        System.arraycopy(internalArray, 0, tempArray, 0, index);  
        System.arraycopy(internalArray, index, tempArray, index + 1, 1);  
    } else {  
        // 나머지 경우  
        System.arraycopy(internalArray, 0, tempArray, 0, index);  
        System.arraycopy(internalArray, index, tempArray, index + 1, currentArraySize - index);  
    }  
    tempArray[index] = t;  
    modifyCurrentArraySize(1);  
    internalArray = tempArray;  
}
```

```
@Override  
public void add(int index, E element) {  
    //  
    if(length == currentCapacity) {  
        increaseCapacity();  
    }  
  
    shiftRightFrom(index);  
  
    elements[index] = element;  
    length++;  
}
```

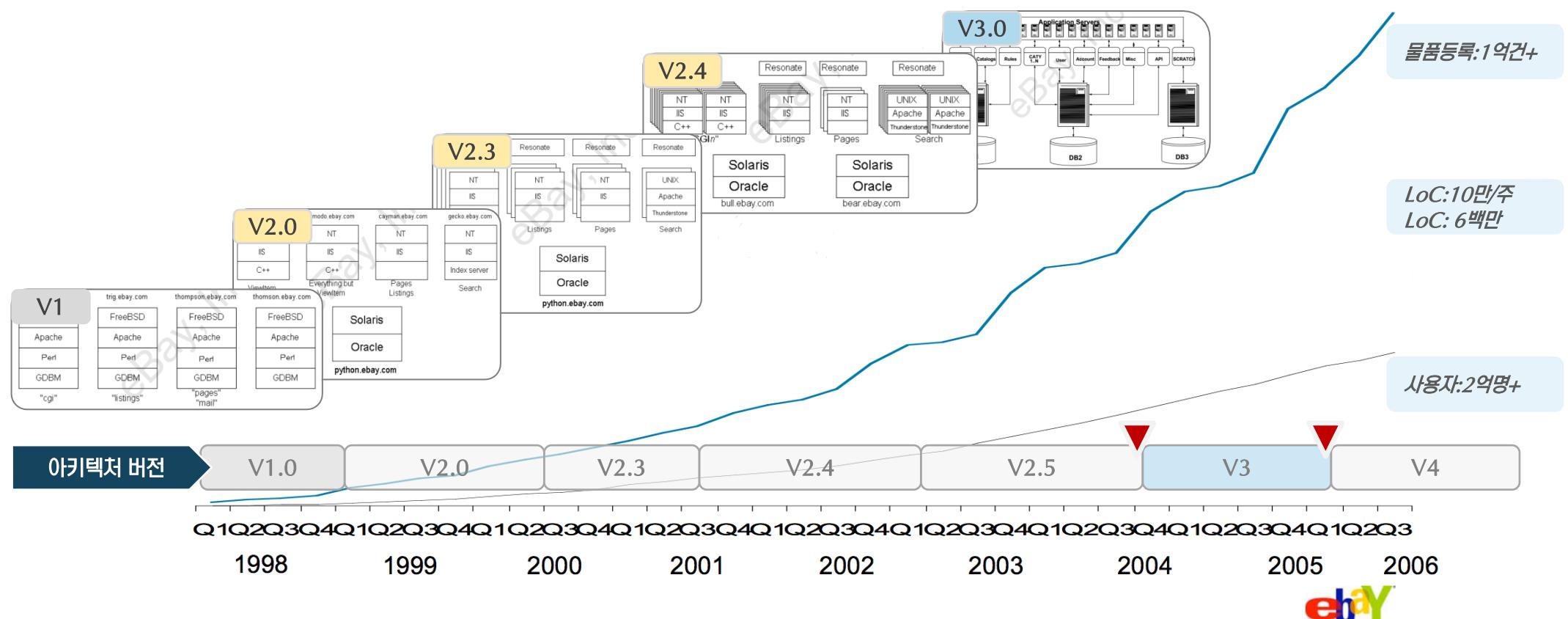
0.6 Clean Code II – 구조적인 대응 [1/2] – 조직

- ✓ 성장하는(growing) 기업은 급변하는 비즈니스 환경에 대응하기 위해 다양한 전략을 마련하고 실천합니다.
- ✓ 조직[구조]의 변화를 통한 경쟁력 확보는 무엇보다 중요한 전략 요소입니다.
- ✓ 기존 팀의 분할하거나 합치고, 새로운 팀을 추가하고, 팀의 규모를 늘리는 방식으로 조직 역량을 높여 갑니다.
- ✓ 성장하는 기업의 조직은 끝없이 성장함으로써, 기업의 경쟁력을 확보하여 성장할 수 있는 발판을 마련합니다.



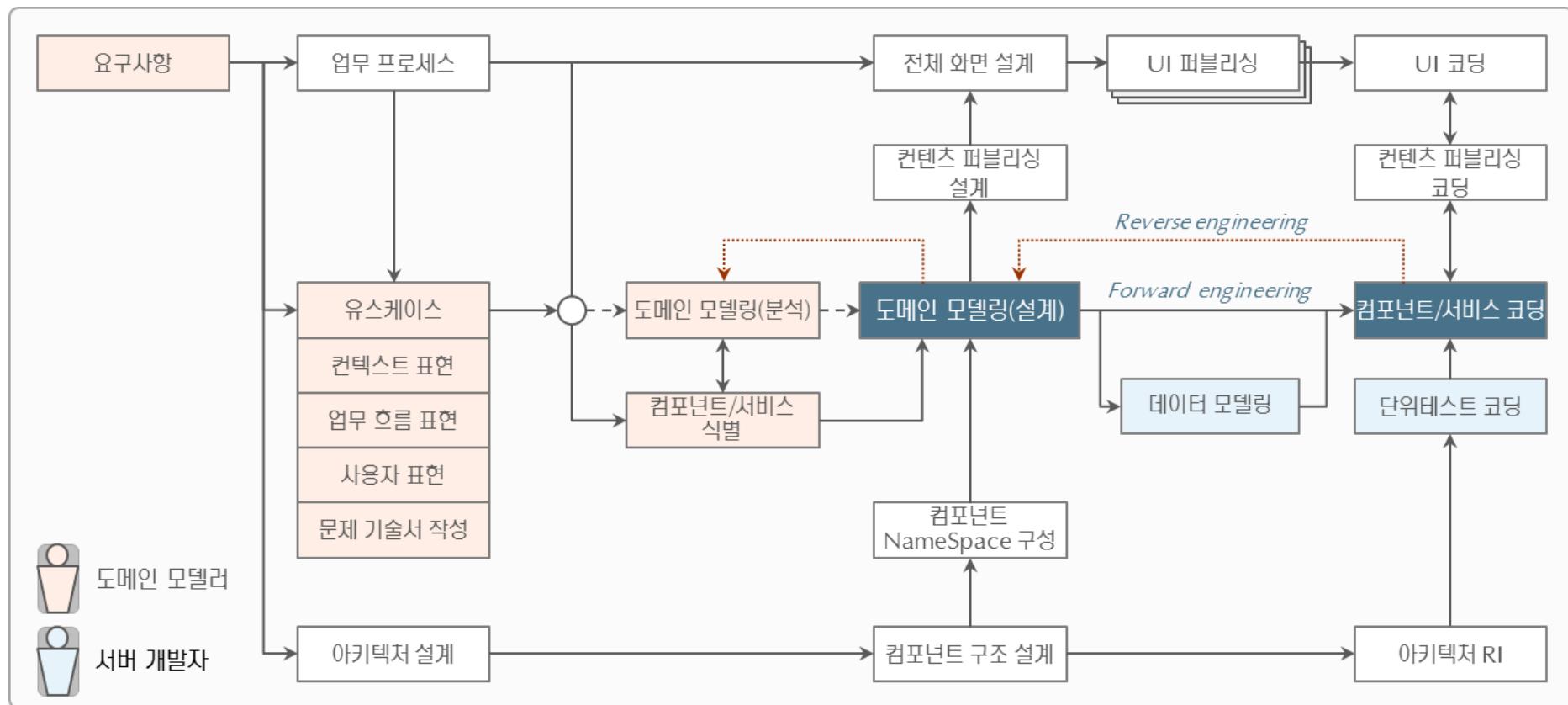
0.6 Clean Code II – 구조적인 대응 [2/2] – ebay

- ✓ ebay.com은 빠른 속도로 성장하여 온 e-commerce 서비스의 대표 기업입니다.
- ✓ 사용자와 물품 등록 건수의 증가에 따라, 아키텍처(구조)를 지속적으로 개선하는 방식으로 대응했습니다.
- ✓ 각 아키텍처(시스템 구조) 버전 별로 감당할 수 있는 사용자와 물품 등록 건수가 있습니다.
- ✓ 급격한 확장이 필요한 시스템에서 구조적인 대응은 반드시 필요하며, 이것을 아키텍처의 변화를 가져옵니다.



0.6 Clean Code II – Zoom-out 1

- ✓ 코드가 만들어지는 일련의 과정을 살펴보고, 코드를 위한 대상 이해가 시작되는 부분부터 Clean을 생각합니다.
- ✓ Clean Code는 개념을 정확히 표현하고 있어야 하며, 그 개념은 올바른 구조 속에 표현되어야 합니다.
- ✓ Clean은 단순히 코딩 기술의 문제가 아니라, 모든 개발 활동의 결과로 나타나는 것입니다.
- ✓ 개발 활동에서 직접 코드와 연결되는 곳(아키텍팅, 도메인 모델링, 등)에 초점을 두고 Clean Code를 이야기 합니다.



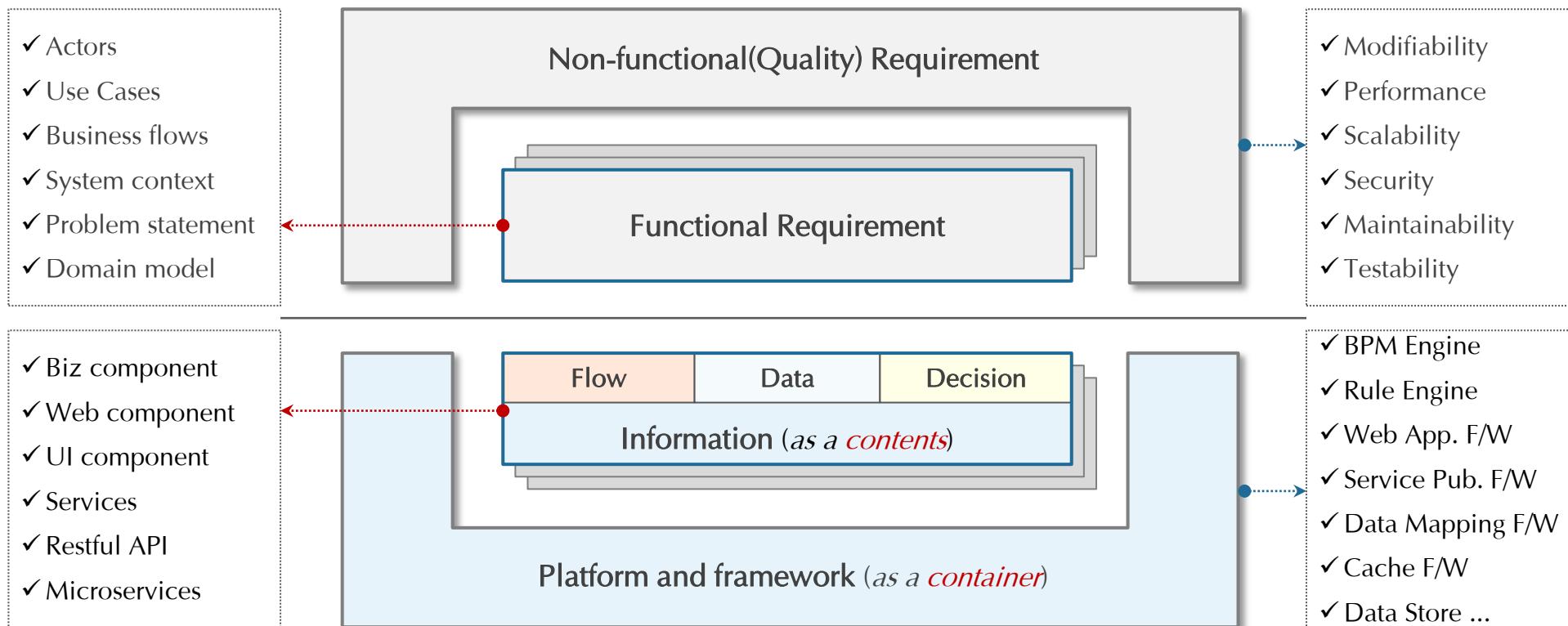
0.6 Clean Code II – Zoom-out 2

- ✓ 앞의 개발 흐름을 간략하게 표현하면, 도메인 모델 → 도메인 소스코드, 아키텍처 모델 → 아키텍처 소스코드 입니다.
- ✓ 코드 수준에서 기술과 구조를 표현하는 아키텍처 코드와 업무를 표현하는 도메인 코드를 제대로 분리하여야 합니다.
- ✓ 모든 유형의 코드에는 객체지향 개념과 논리적인 사고가 코드를 뒷받침하고 있습니다.
- ✓ 이 강의에서는 아래 여섯 가지 블럭을 중심으로 Clean Code를 다룰 계획입니다.



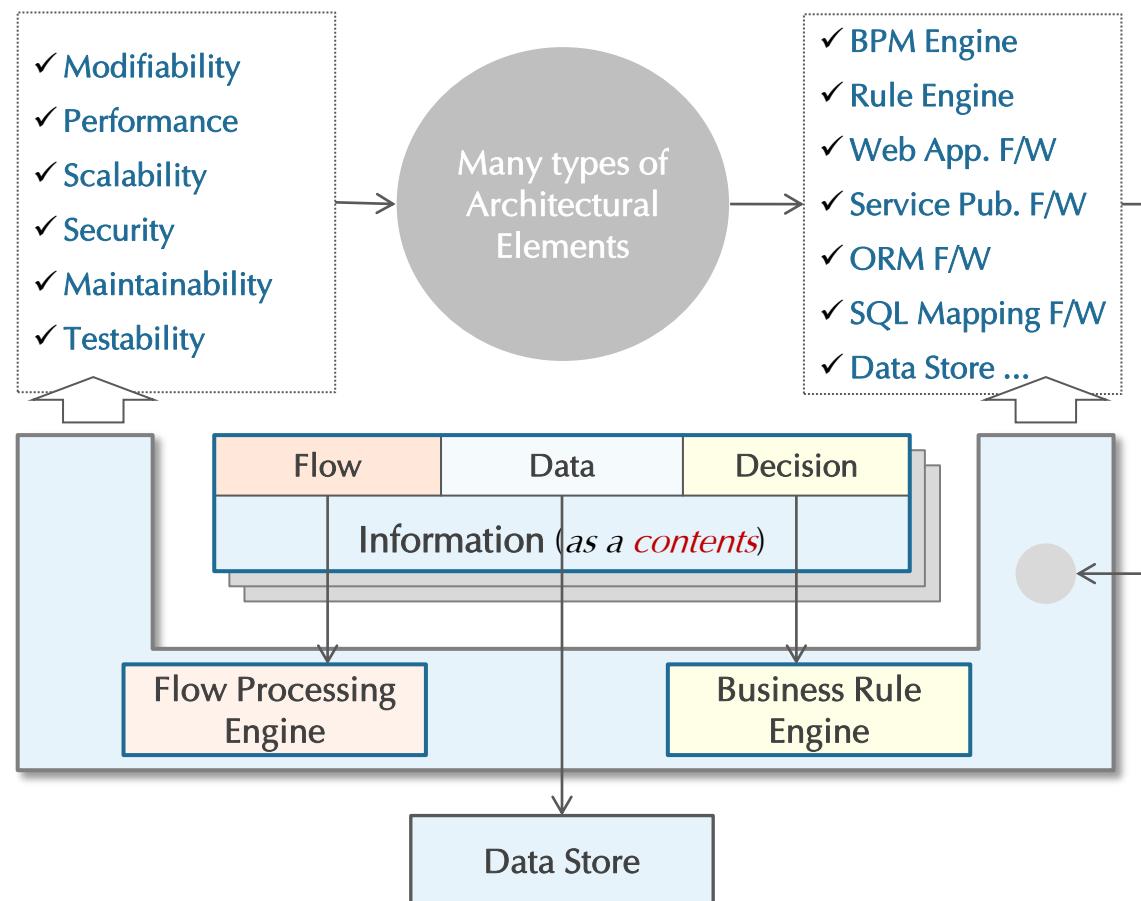
0.6 Clean Code II – Zoom-out 3

- ✓ 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.
- ✓ 우리가 말하는 코드가 어디에 어떤 형태로 존재하는 것일까요?



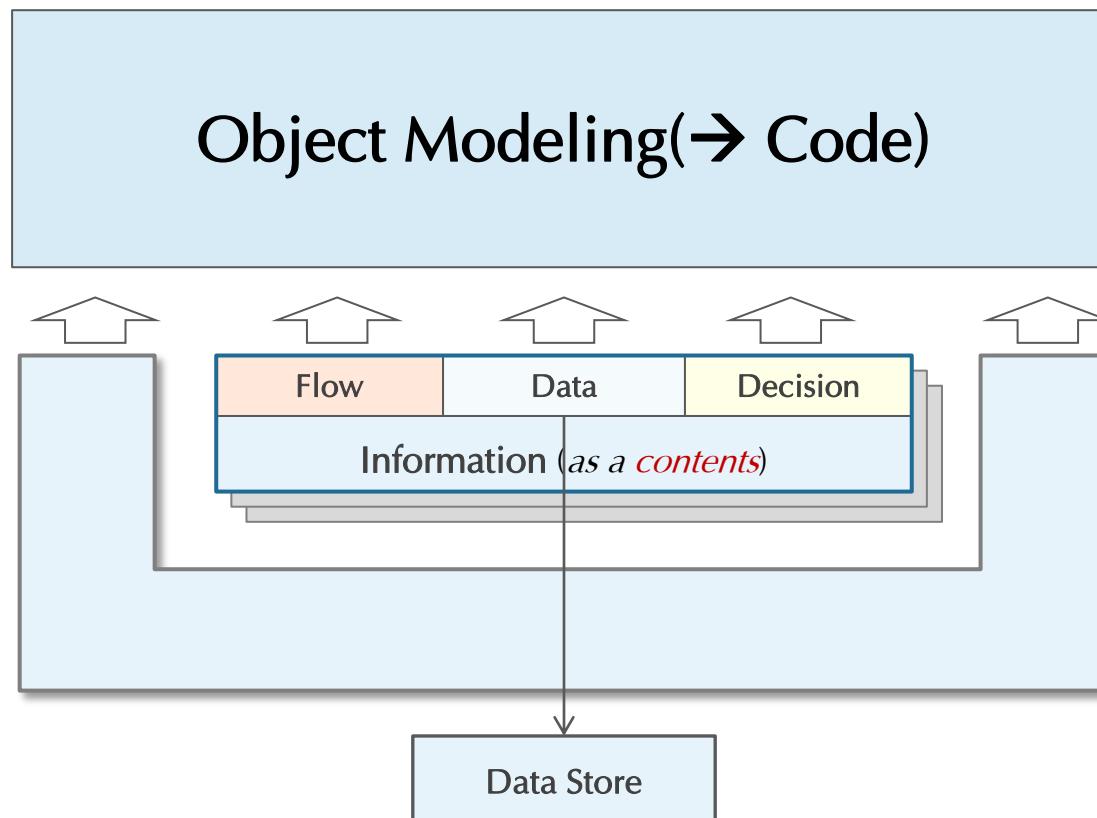
0.6 Clean Code II – Zoom-out 4

- ✓ 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.
- ✓ 이러한 그림의 요구 사항이나 특성들은 어떤 형식으로 표현될까요?



0.6 Clean Code II – Zoom-out 5

- ✓ 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.
- ✓ 객체 모델링의 결과는 “클래스의 협업”이고, 각 클래스는 결국 코드로 표현되어야 의미를 가지고 실행할 수 있습니다.



요약

- ✓ SW 기술이 낙후한 조직의 개발 트랜드는 기술 보다는 관리입니다.
- ✓ 그런 조직은 소프트웨어 개발에서 코딩이 차지하는 비중은 아주 적다고 이야기 합니다.
- ✓ Clean Code는 이해 → 모델링 → 아키텍팅을 제대로 거친 후에야 마지막 단계에서 얻을 수 있는 가장 중요한 산출물임을 많은 사람들이 잊고 있습니다. 코드는 확인 가능한 모델이며, 명세입니다.
- ✓ Clean Code 작성법을 코드 뿐만 아니라 아키텍처, 더 나아가 모델링 관점에서 살펴 봅니다.

		요구사항	분석/설계	구축	테스트	유지보수					
기술부문	기법	커뮤니케이션기법 프로토타입	구조적 설계 MDA, SOA, PL	개발 방법론	-Top-down, Bottom-up Big-bang, Sandwich Backbone	-Correction -Adaption -Perfective					
	주요활동	-요구사항 프로세스 -요구사항 추출 -요구사항 분석 -요구사항 명세 -요구사항 확인	구조적 아키텍처 설계 분석 평가 SW설계 표기준 SW설계 전략방법	개발관리 기본설계 코딩, 개발 테스팅, 재사용, 개발품질, 통합	-테스트 수준 -테스트 기법 -테스트 흐름 -테스트 관리	-유지보수 프로세스 -유지보수 기법 -유지보수 관리					
	도구	DOORS, RequisitePro 등	System Architect, ROSE, Together 등	Visual Studio, Eclipse, JBuilder 등	JUnit, Apache Ant, BoundsChecker 등	ClearCASE, GNU CVS 등					
형성관리											
주요활동	기법	형성관리 방법론									
	도구	형상관리 프로세스 관리, 형상징목 식별, 형상 통제, 형상 상태보고, 형상 검사, 혼리즘 관리/인도									
		CVS, SVN, dimension 등									
SW공학관리											
도구	기법	Planing, Team Building, PERT, Gantt Chart, Estimation, Risk Management 등									
	주요활동	작수/범위 정의, 프로젝트 계획, 프로젝트 법규, 검토/평가, 프로젝트 종료, SW공학측정									
		MS Project, OpenProj, Workspace, Planner, PMBOK 등									
프로세스											
관리부문	기법	SW-Lifecycle									
	주요활동	프로세스 이행/변경, 프로세스 정의, 프로세스 평가, 프로세스/제품 측정									
	도구	CMMI, SPICE, SP모델 등									
도구 및 기법											
기법	Heuristic 방법론(장성)법 등, Formal 방법론(역시지향, OBD, Agile, XP 등), Prototyping 방법론										
	SW공학률 적용, SW공학기법 적용										
	도구	Ant, CAP, Embrace, Mantis 등									
품질											
기법	QFD, TQC 등										
	SW품질관리 프로세스, 품질보증, 검증/확인, 검토/감사, 품질 요구사항										
	도구	SIIK, Q2, DO 소프트웨어, Infometica, First-logic, DQ miner 등									

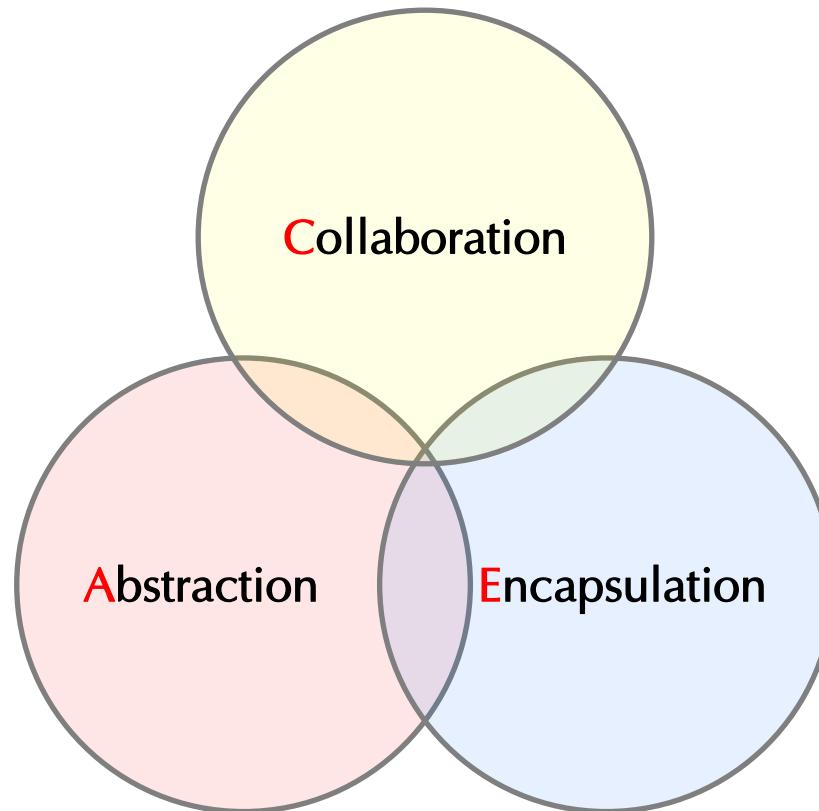


1. Clean Code – 객체 지향

-
- 1.1 객체 지향
 - 1.2 실습1-1: 구구단 1열 출력
 - 1.3 실습1-2: 구구단 4열 출력
 - 1.4 실습1-3: 구구단 정방형 출력
 - 1.5 토론
 - 1.6 객체 지향 해법
 - 1.7 실습1-4: 구구단 본질
 - 1.8 실습1-5: ASCII 테이블 출력(옵션)

1.1 객체 지향 – 개요

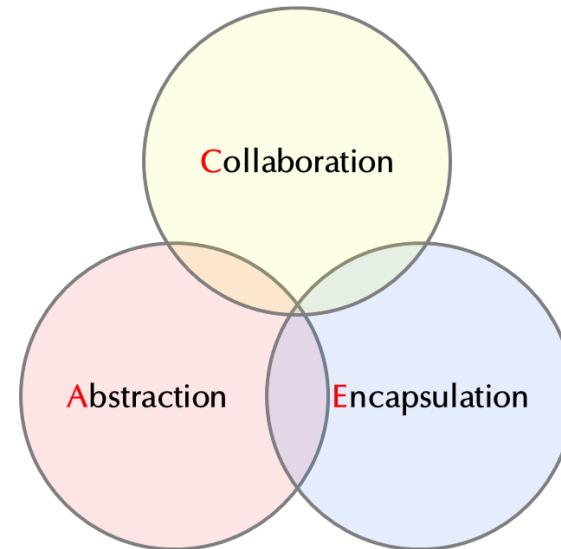
- ✓ 빛의 3원색은 빨강(Red), 녹색(Green), 청색(Blue) 입니다. 줄여서 RGB라고 합니다.
- ✓ 객체 지향 프로그래밍의 3대 요소는 무엇으로 축약할 수 있을까요? 여러 가지 의견들이 많이 있습니다.
- ✓ 추상화(abstraction), 상속(inheritance), 캡슐화(encapsulation), 다형성(polymorphism), ...
- ✓ 우리는 추상화와 캡슐화를 바탕으로 하는 협업(Collaboration) 세 가지를 제시합니다.



1.1 객체 지향 – 개요

- ✓ 어떤 접근 방법을 사용하든 SW 설계는 낮은 결합도와 높은 응집도를 유지하는 활동이어야 합니다.
- ✓ 원하는 수준의 결합도와 응집도를 유지하기 위해서 관심사를 분리(Separation of Concern)해야 합니다.
- ✓ 객체 지향의 세 가지 특성은 SW 설계 원칙(Loose coupling, high cohesion)을 아주 잘 지원합니다.
- ✓ 뿐만 아니라 강력한 표현력을 제공하여 주기 때문에 복잡한 대상을 단순하게 처리할 수 있습니다.

Loose coupling, high cohesion !!



1.1 객체 지향 – 추상화(Abstraction)

- ✓ 추상화는 공통적인 특징을 찾아 이름과 의미를 부여하는 사고 활동으로 표현력을 극대화할 수 있습니다.
- ✓ 추상화의 힘을 빌리면, 복잡하거나 숫자가 많은 대상을 필요한 만큼 간결하게 표현할 수 있습니다.
- ✓ Abstraction의 범주에는 inheritance, polymorphism, operator overriding 등이 있습니다.
- ✓ 추상화 역량은 인간(human being)을 동물과 구별하는 대표적인 역량입니다.

사육사가 {조류 우리}에 들러서 {새}들의 상태를 확인하다가, 홍학 우리에서 {다리를 다친 홍학}을 발견하고는 수의사에게 치료를 부탁했습니다.

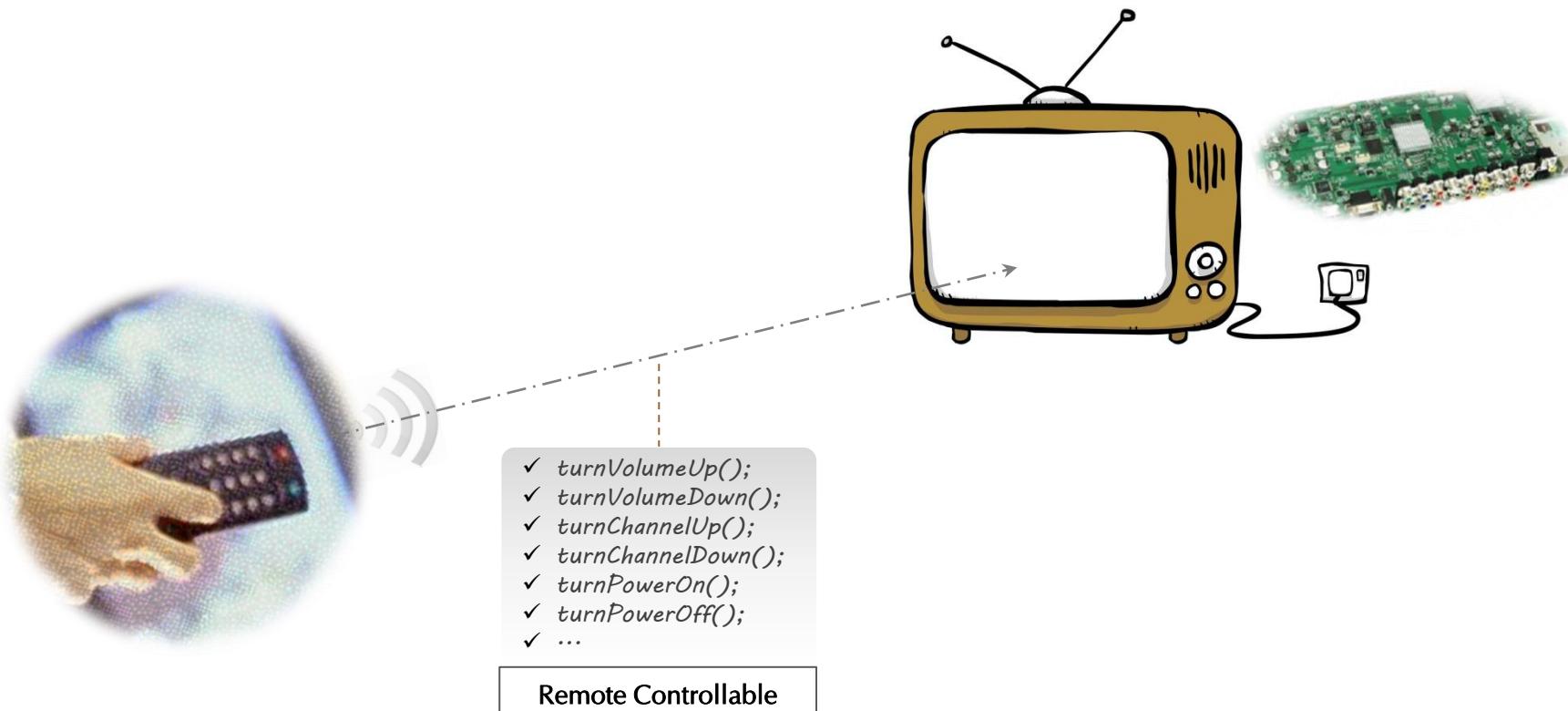


/출처/ <http://blog.sangwoodiary.com/entry/20100502-at-the-zoo>

1.1 객체 지향 – 캡슐화(Encapsulation)

- ✓ 어떤 기기가 동작하는 방식을 알아야 사용할 수 있다면 TV 조작조차 간단하지 않을 겁니다.
- ✓ 하지만, 우리는 리모컨이라는 작은 기기를 가지고, 소리가 커지는 원리를 몰라도, 쉽게 소리를 키울 수 있습니다.
- ✓ 복잡한 것은 안으로 숨기고, 밖으로는 간단한 인터페이스 만을 내 놓아서, 누구나 쉽게 사용해 주기 때문입니다.
- ✓ 프로그램에서도 복잡한 것은 안쪽에 숨기고(캡슐화를 하고), 밖으로는 간단한 인터페이스를 내 놓아야 합니다.

Abstraction은 효율적인 캡슐화를 하는 방법을 제공합니다. 인터페이스를 실체화(realization)하는 관계는 추상화 계층을 이용하는 방식입니다.



1.1 객체 지향 – 협업(Collaboration)

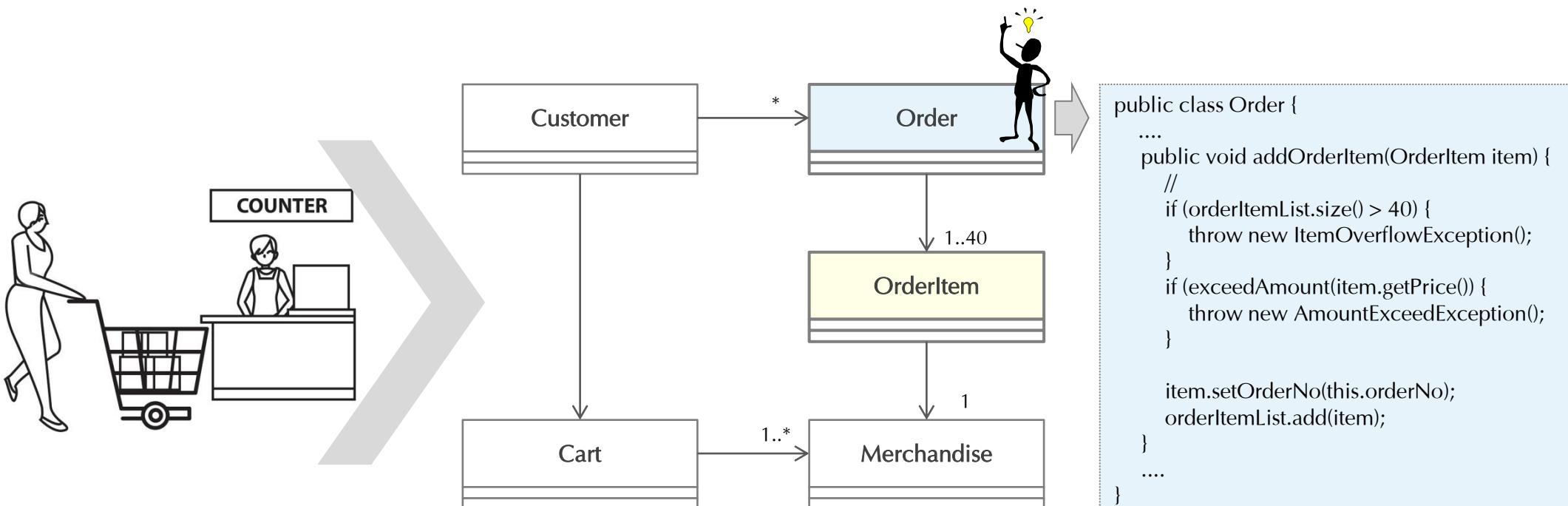
- ✓ 실세계에 존재하는 모든 사람과 동식물의 생활은 모두가 협업의 연속입니다.
- ✓ 오래 세월을 거치면서 협업 방식은 매우 고도화 되었고, 높은 수준의 효율을 보여주고 있습니다.
- ✓ 실세계의 이런 유산을 시스템 세계로 끌고 들어가서 시스템 세계를 건설하는데 사용하여야 합니다.
- ✓ 객체와 개체 간의 협업이라는 관점에서 사고를 하고, 필요하면 실세계에서 지혜를 얻어야 합니다. ← 실세계 매핑



<http://epiccollaboration.com/news-update/mass-collaboration-challenge-integration>

1.1 객체 지향 – 실세계 매핑

- ✓ SW 시스템 세계, 즉 애플리케이션 세계는 실세계의 미니어처입니다.
- ✓ 하지만, 어떤 부분은 실세계의 복잡한 개념을 그대로 옮겨 두어야 합니다. ← 정확한 개념 파악이 필요합니다.
- ✓ 예를 들면, 실제 물품은 여러 개인데 상황에 따라 하나로 또는 여러 개로 다루거나 다른 이름으로 부릅니다.
- ✓ 실세계에 존재하는 객체(entity, object)에 지능을 부여하여 협업하도록 합니다.



1.2 실습1-1: 구구단 1열로 출력하기

- ✓ 구구단을 출력하는 프로그램을 작성합니다. 2단에서부터 9단까지 한 컬럼으로 출력합니다.
- ✓ 아주 간단하여 클래스 하나로도 충분할 수 있습니다.
- ✓ 코딩을 끝낸 후 코딩 시점에 문제를 어떻게 이해 했는지 잠시 생각해 봅시다.

코딩 실습 !!

Times table.

2 × 1 = 2
2 × 2 = 4
2 × 3 = 6
2 × 4 = 8
2 × 5 = 10
2 × 6 = 12
2 × 7 = 14
2 × 8 = 16
2 × 9 = 18

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

1.3 실습1-2 : 구구단 4열로 출력하기

- ✓ 이번에는 4 열로 출력합니다. 서식이 조금 변경되었습니다.
- ✓ 콘솔 출력이 갖는 특성과 한계 때문에 약간의 고민이 필요하겠군요.
- ✓ 1 열에서 4열로 변경되었으니, 열은 고정이 아니라, 충분히 변경될 수 있음을 알 수 있습니다.
- ✓ 마찬가지로, 코딩이 끝나면 코딩 시점에 문제를 어떻게 이해 했는지 잠시 생각해 봅시다.

코딩 실습 !!

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

1.4 실습1-3 : 구구단 정방형으로 출력하기

- ✓ 정방형 구구단을 보신 적 있으신가요? 이런 형식의 구구단으로 학생을 가르치는 나라도 있다고 합니다.
- ✓ 이것 또한 구구단이니 실습 1-2의 코드를 확장하여 정방형 구구단을 출력을 해보겠습니다.
- ✓ 예상치 못한 변화일 수 있습니다. 그리고 이것은 앞의 구구단과 완전히 다르다고 생각할 수 있습니다.
- ✓ 마찬가지로, 코딩이 끝나면 코딩 시점에 문제를 어떻게 이해 했는지 잠시 생각해 봅시다.

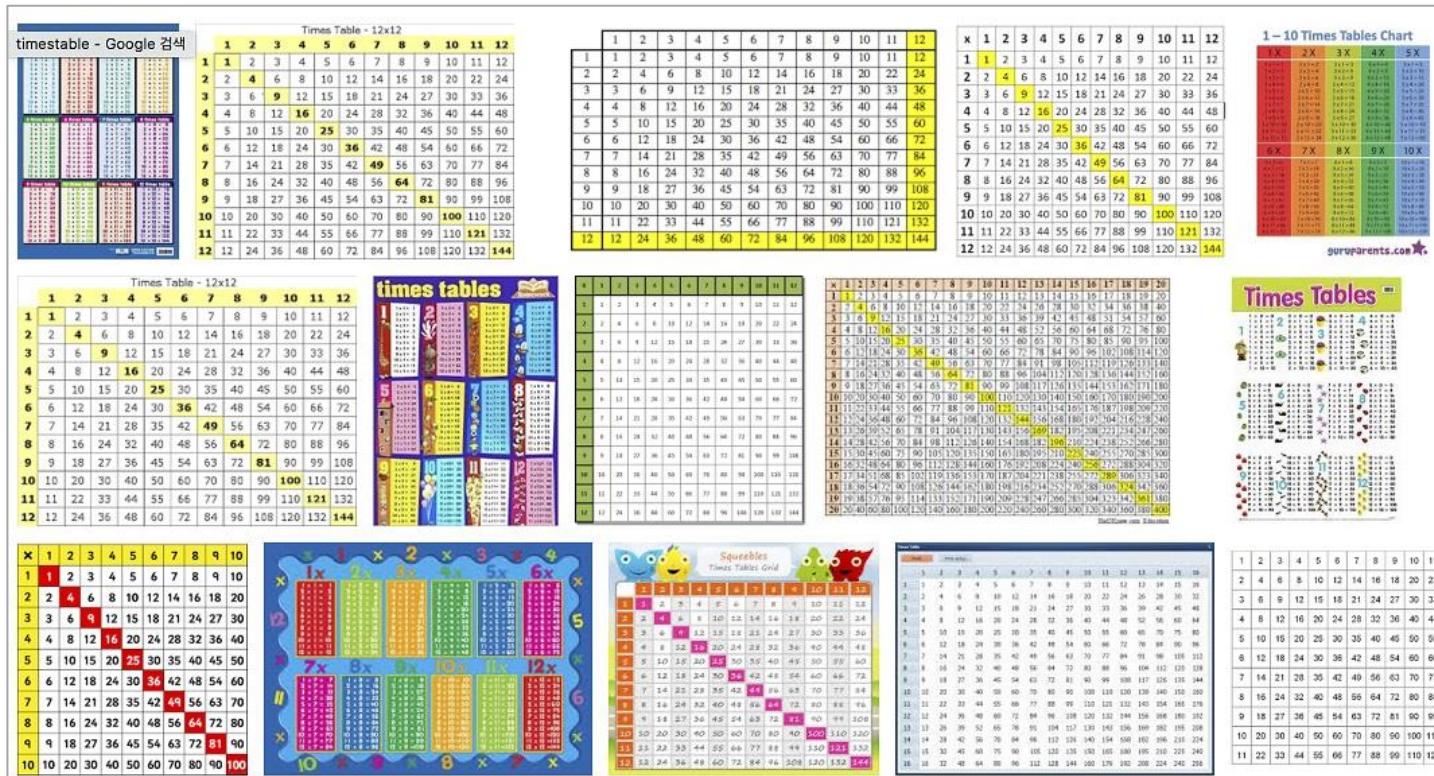
코딩 실습 !!



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

1.5 토론 : 본질, 변화, 다양성 1

- ✓ 구글에서 “구구단”이나 “Timetable”을 검색하면 매우 다양한 형식의 구구단을 볼 수 있습니다.
- ✓ 여러분의 코드는 이런 다양한 구구단에 대응할 수 있습니까?
- ✓ 대응할 수 있다면 이유는 무엇이고, 대응할 수 없다면 그 이유는 무엇입니까?
- ✓ 변화에 대응하는 방법으로 본질을 이해하고 본질이 아닌 것을 찾아서 대응하는 방법이 있습니다.



/이미지 출처/ 구글 검색

1.5 토론 : 본질, 변화, 다양성 2

- ✓ 여러분의 코드는 어떠합니까?
- ✓ 문제를 해결할 때, 여러분들이 고민한 내용은 “구구단”이라는 본질과 관계가 있습니까?
- ✓ 아니며, 당장 문제로 제시된 구구단 표현(representation)에 집중했습니까?
- ✓ 그리고, 여러분의 코드를 QA 관점에서는 어떻게 판단할까요?

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

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

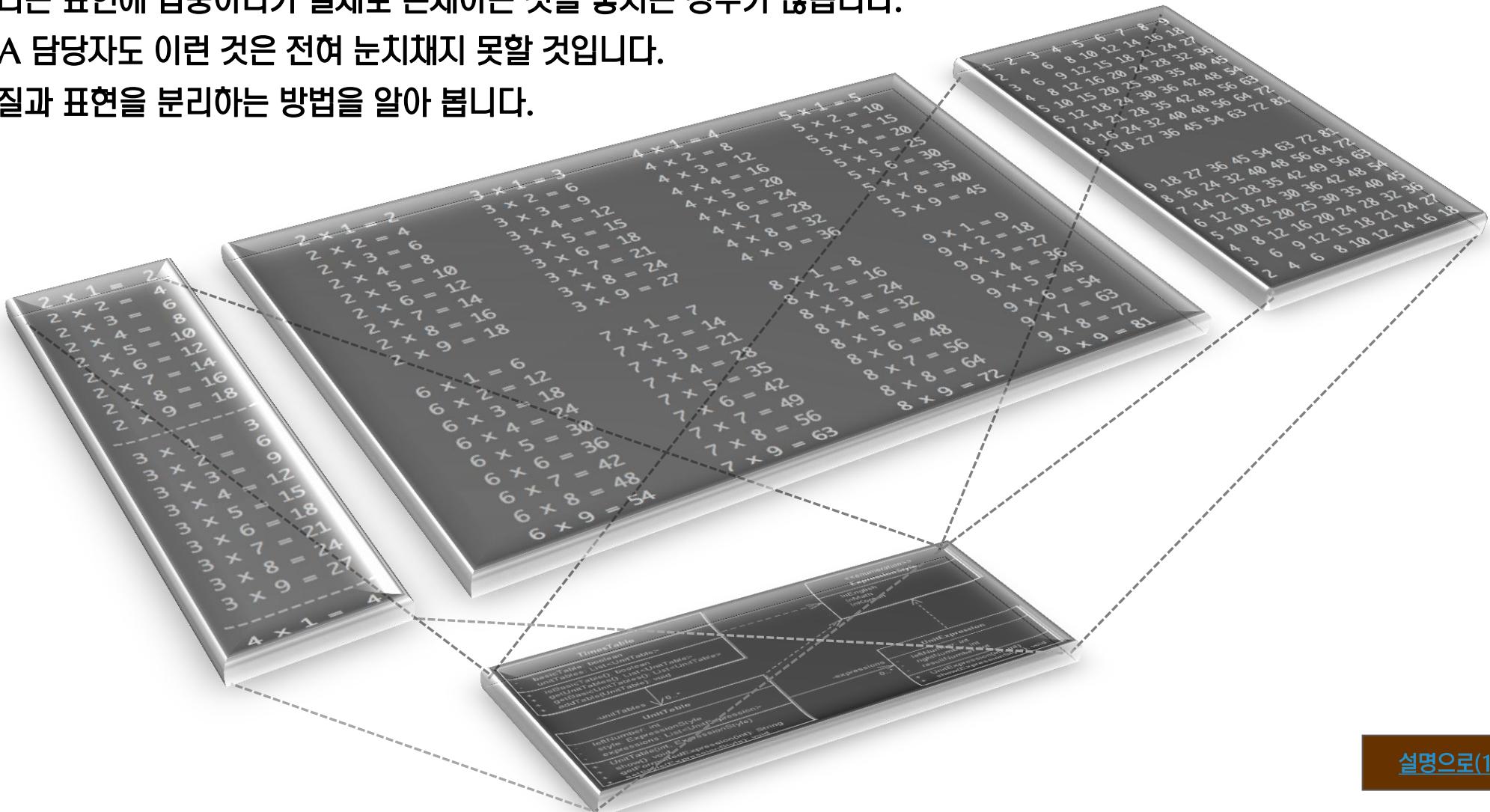
    for (int leftNumber = 2; leftNumber<=9; leftNumber++) {
        for (int rightNumber = 1; rightNumber <=9; rightNumber++) {
            System.out.println(buildTimesItem(itemFormat, leftNumber, rightNumber));
        }
        System.out.println("-----");
    }
}

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;
}
```

1.6 객체 지향 해법

- ✓ 구구단 실습은 객체지향의 기본 특징이 실세계에 존재하는 개념을 프로그램 세계로 끌어올리는 것을 보여 줍니다.
- ✓ 우리는 표현에 집중하다가 실제로 존재하는 것을 놓치는 경우가 많습니다.
- ✓ QA 담당자도 이런 것은 전혀 눈치채지 못할 것입니다.
- ✓ 본질과 표현을 분리하는 방법을 알아 봅니다.



설명으로(1h)

1.7 실습1-4: 구구단 본질 1

- ✓ 이제 변화하는 것과 변하지 않는 것, 본질과 비본질에 대해 충분히 알아 보았습니다.
- ✓ 앞의 설명을 참조하여, 이제 실습1-1~실습1-3에 대응할 수 있는 여러분의 코드를 작성해 봅니다.
- ✓ 모델을 어떻게 하셔도 좋습니다. 그것은 문제를 이해하고 풀어 가는 방법입니다.
- ✓ 중요한 것은 “구구단”이라는 도메인의 변하지 않는 본질이 있으며, 그것을 코드에 반영해야 한다는 사실입니다.

코딩 실습 !!

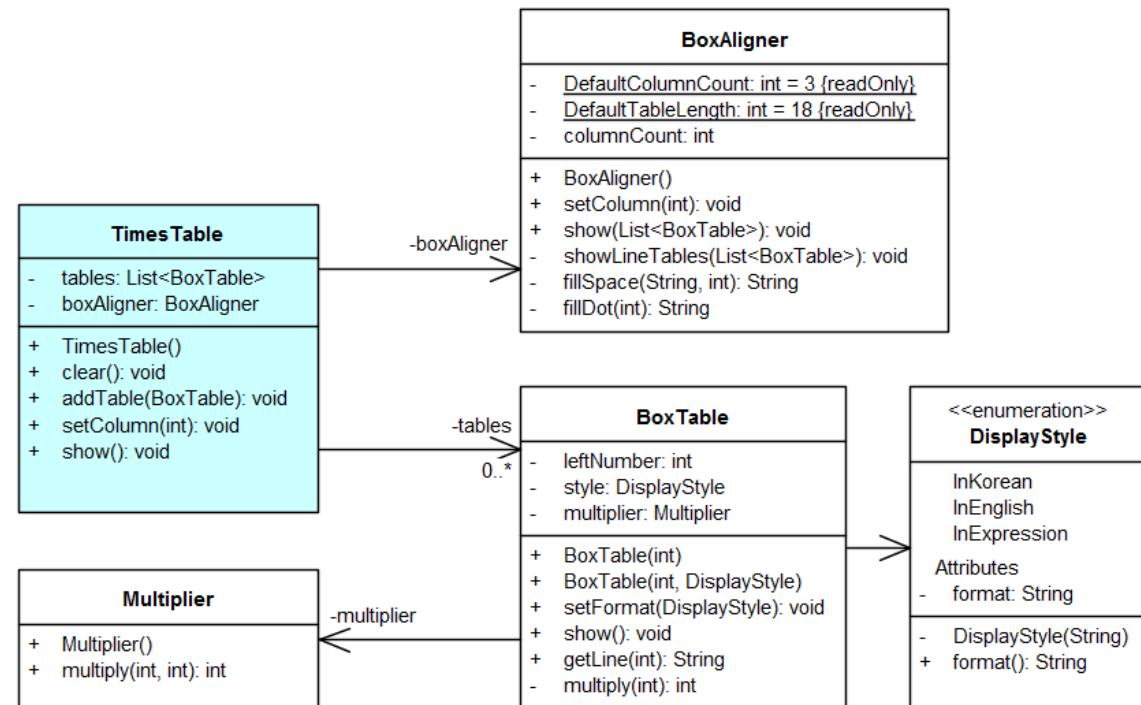
```

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

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

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

```



1.7 실습1-4: 구구단 본질 2

- ✓ TimesTable을 사용하는 데모 프로그램은 TimesTable과 뷰를 나눠서 다루며,
- ✓ 데모 프로그램이라 할 지라도 아래와 같이 읽기 쉽고 간결한 Clean Code로 작성할 수 있습니다.

코딩 실습 !!

```
public class TimesTableDemo {  
    //  
    private TimesTable timesTable;  
  
    public TimesTableDemo() {  
        //  
        this.timesTable = new TimesTable();  
        this.timesTable.initialize();  
    }  
  
    public void showTableLineDemo() {  
        //  
        ConsoleView consoleView = new ConsoleView(LineType.TableLine);  
        consoleView.setColumnSize(4);  
        consoleView.setStyle(ExpressionStyle.InMath);  
        consoleView.show(this.timesTable);  
        consoleView.showLineSeparator();  
    }  
  
    public void showSimpleLineDemo() {  
        //  
        ConsoleView consoleView = new ConsoleView(LineType.SimpleLine);  
        consoleView.setColumnSize(4);      // ignored  
        consoleView.show(this.timesTable);  
        consoleView.showLineSeparator();  
    }  
}
```

1.8 실습1-5 ASCII 테이블 출력(옵션)

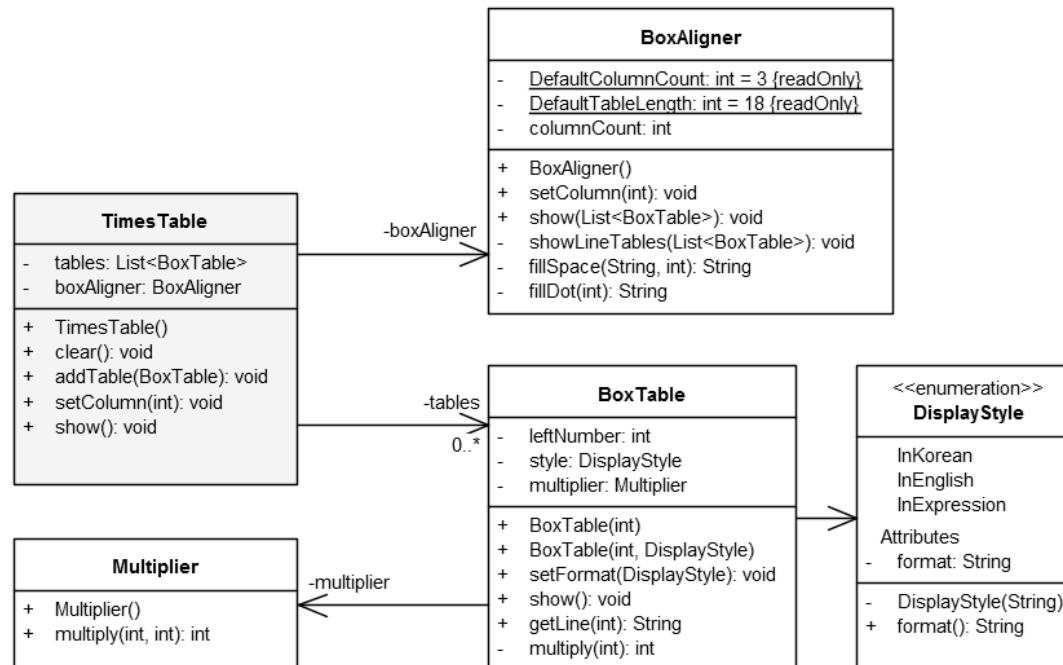
- ✓ ASCII 테이블을 출력해 봅니다.
- ✓ 표현하는 방식에 대한 니즈는 매우 다양합니다.
- ✓ 이 프로그램을 작성하는 여러분의 접근 방법은 무엇입니까?
- ✓ ASCII의 본질은 무엇일까요? 여러분은 그 본질을 이해하기 전 까지 코드를 작성하면 안됩니다. 잘못된 코드를 작성할 확률이 높기 때문입니다.

① hex	Char	② c	Binary	Oct	Hex	Char
000	NULL	000	00000000	000	0X00	NULL
001	SOH	001	00000001	001	0X01	SOH
002	STX	002	00000010	002	0X02	STX
003	ETX	003	00000011	003	0X03	ETX
004	EOT	004	00000100	004	0X04	EOT
005	ENQ	005	00000101	005	0X05	ENQ
006	ACK	006	00000110	006	0X06	ACK
007	BEL	007	00000111	007	0X07	BEL
008	BS	008	00001000	010	0X08	BS
009	HT	009	00001001	011	0X09	HT
010	LF	010	00001010	012	0X0A	LF
011	VT	011	00001011	013	0X0B	VT
012	FF	012	00001100	014	0X0C	FF
013	CR	013	00001101	015	0X0D	CR
014	SO	014	00001110	016	0X0E	SO
015	SI	015	00001111	017	0X0F	SI
...	...	016	00010000	020	0X10	DLE

③ hex	Dec	Code		Hex	Dec	Code		Hex	Dec	Code		Hex	Dec	Code		Hex	Dec	Code		Hex	Dec	Code								
0X00	000	NULL		0X10	016	DLE		0X20	032	SP		0X30	048	0		0X40	064	@		0X50	080	P		0X60	096	`		0X70	112	p
0X01	001	SOH		0X11	017	DC1		0X21	033	!		0X31	049	1		0X41	065	A		0X51	081	Q		0X61	097	a		0X71	113	q
0X02	002	STX		0X12	018	SC2		0X22	034	"		0X32	050	2		0X42	066	B		0X52	082	R		0X62	098	b		0X72	114	r
0X03	003	ETX		0X13	019	SC3		0X23	035	#		0X33	051	3		0X43	067	C		0X53	083	S		0X63	099	c		0X73	115	s
0X04	004	EOT		0X14	020	SC4		0X24	036	\$		0X34	052	4		0X44	068	D		0X54	084	T		0X64	100	d		0X74	116	t
0X05	005	ENQ		0X15	021	NAK		0X25	037	%		0X35	053	5		0X45	069	E		0X55	085	U		0X65	101	e		0X75	117	u
0X06	006	ACK		0X16	022	SYN		0X26	038	&		0X36	054	6		0X46	070	F		0X56	086	V		0X66	102	f		0X76	118	v
0X07	007	BEL		0X17	023	ETB		0X27	039	'		0X37	055	7		0X47	071	G		0X57	087	W		0X67	103	g		0X77	119	w
0X08	008	BS		0X18	024	CAN		0X28	040	(0X38	056	8		0X48	072	H		0X58	088	X		0X68	104	h		0X78	120	x
0X09	009	HT		0X19	025	EM		0X29	041)		0X39	057	9		0X49	073	I		0X59	089	Y		0X69	105	i		0X79	121	y
0X0A	010	LF		0X1A	026	SUB		0X2A	042	*		0X3A	058	:		0X4A	074	J		0X5A	090	Z		0X6A	106	j		0X7A	122	z
0X0B	011	VT		0X1B	027	ESC		0X2B	043	+		0X3B	059	;		0X4B	075	K		0X5B	091	[0X6B	107	k		0X7B	123	{
0X0C	012	FF		0X1C	028	FS		0X2C	044	,		0X3C	060	<		0X4C	076	L		0X5C	092	\		0X6C	108	l		0X7C	124	
0X0D	013	CR		0X1D	029	GS		0X2D	045	-		0X3D	061	=		0X4D	077	M		0X5D	093]		0X6D	109	m		0X7D	125	}
0X0E	014	SO		0X1E	030	RS		0X2E	046	.		0X3E	062	>		0X4E	078	N		0X5E	094	^		0X6E	110	n		0X7E	126	~
0X0F	015	SI		0X1F	031	US		0X2F	047	/		0X3F	063	?		0X4F	079	O		0X5F	095	_		0X6F	111	o				

요약

- ✓ 프로그램을 작성할 때, 짧은 일정, 충분하지 않은 인력 등을 이유로 “표현”에 집중하여 개발하는 경향이 있습니다.
- ✓ 표현 영역은 휘발성(volatile)이 매우 강한 영역으로 환경과 비즈니스 변화에 따라 변경이 자주 발생합니다.
- ✓ 따라서 표현 영역 보다는 본질을 올바로 이해하고 표현하는 방식으로 코딩을 하여야 Clean Code를 얻을 수 있습니다.
- ✓ 외부 변화에 유연하게 대응하지 못하는 코드는 Clean Code가 될 수 없습니다.





2. Clean Code – 책임 표현

-
- 2.1 책임이란?
 - 2.2 책임과 오퍼레이션
 - 2.3 실습 2-1 – ArrayList
 - 2.4 실습 2-2 - LinkedList
 - 2.5 토론

2.1 책임이란? (1/2)

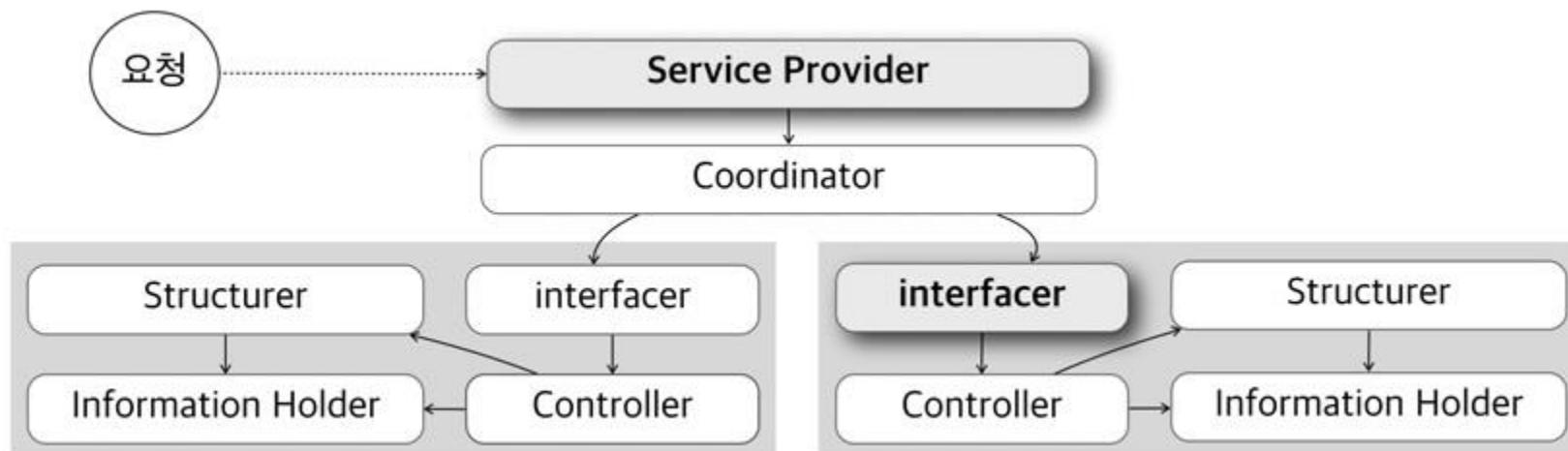
- ✓ 객체 지향 기술 초기에 객체를 찾고 명세하기 위해 사용하는 CRC 카드가 있었습니다.
- ✓ Class Name, Responsibility, Collaborator 세 가지 관점에서 객체를 정의하고 서술하여였습니다.
- ✓ Class를 특징짓는 주요 요소는 책임이며, 해당 책임을 수행할 수 있어야 비로소 협업에 참여할 수 있습니다.
- ✓ 책임은 클래스 오퍼레이션 형식으로 표현합니다. 책임을 잘 표현하는 것이 클래스 범위 Clean Code의 핵심입니다.

<u>Class</u> : CardReader	
<u>Responsibilities</u> : <ul style="list-style-type: none">✓ Request card insert✓ Read card information✓ Validation check✓ Eject card✓ Retain card	<u>Collaborators</u> : <ul style="list-style-type: none">✓ ATM✓ Card

[클래식 CRC Card 예제]

2.1 책임이란? (2/2)

- ✓ 프로그램은 객체들의 협업을 통해 기능을 수행합니다.
- ✓ 따라서, 개발자들이 설계하고 코딩하는 모든 소스코드는 객체에 대한 정보와 활동을 기록하는 것입니다.
- ✓ 그런데, 이런 객체들은 역할에 따라 여섯 가지로 분류할 수 있습니다.
- ✓ 객체가 어떤 유형의 책임을 갖고 있는지 이해하는데 도움이 되는 분류 방법입니다.



- ✓ **Information holder** — knows and provides information
- ✓ **Structurer** — maintains relationships between objects and information about those relationships
- ✓ **Service provider** — performs work and, in general, offers computing services
- ✓ **Coordinator** — reacts to events by delegating tasks to others
- ✓ **Controller** — makes decisions and closely directs others' actions
- ✓ **Interfacer** — transforms information and requests between distinct parts of our system

[출처] Object Design: Roles, Responsibilities, and Collaborations [Rebecca Wirfs-Brock](#)

2.2 책임과 오퍼레이션

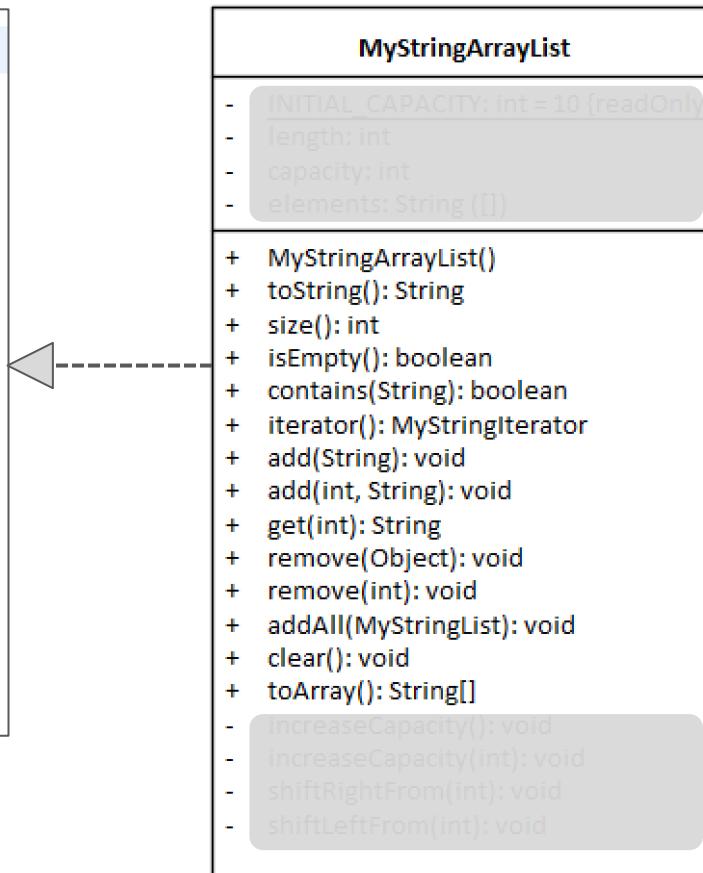
- ✓ 인터페이스 MyStringList에서 제시한 오퍼레이션들은 모두 “책임(responsibility)”입니다.
- ✓ 즉, MyStringList라 불리기 위해서 또는 MyStringList라는 자격을 얻기 위해서 제대로 수행해야 할 책임입니다.
- ✓ 제시한 인터페이스 오퍼레이션을 구현하는 구체적인 코딩은 백인백색입니다.
- ✓ 물론 Clean 구현 코드의 선결 조건은 Clean 한 인터페이스 정의입니다.

```
public interface MyStringList {  
    //  
    int size();  
    boolean isEmpty();  
    boolean contains(String element);  
    MyStringIterator iterator();  
    void add(String element);  
    void add(int index, String element);  
    String get(int index);  
    void remove(Object o);  
    void remove(int index);  
    void addAll(MyStringList sourceList);  
    void clear();  
    String[] toArray();  
}
```

2.3 실습 2-1 – ArrayList 1

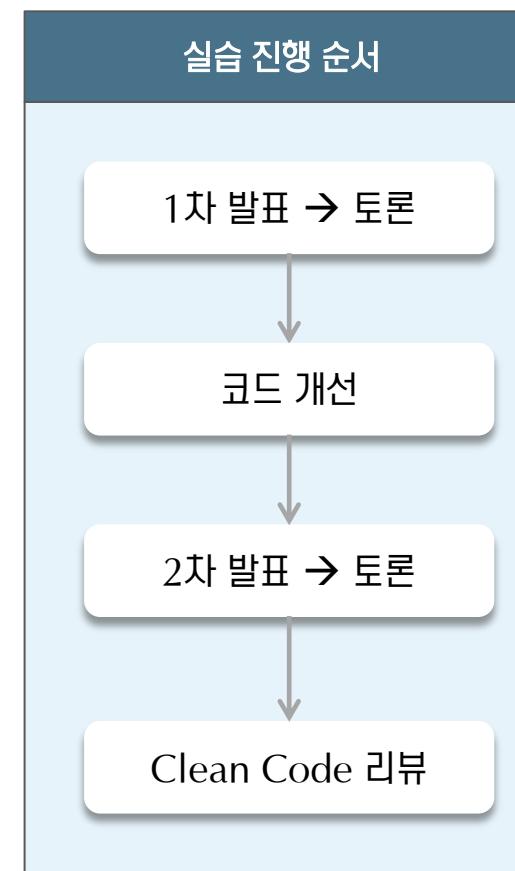
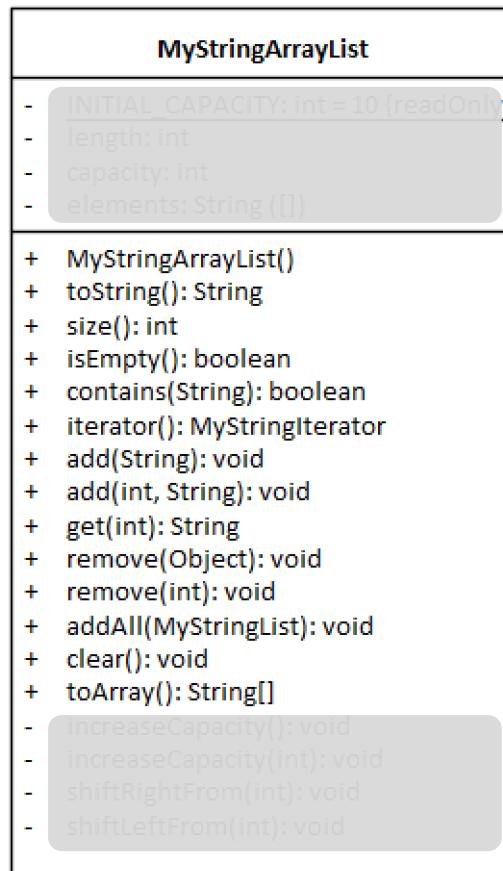
- ✓ 인터페이스 MyStringList에서 제시한 오퍼레이션들은 모두 “책임(responsibility)”입니다.
- ✓ 즉, MyStringList라 불리기 위해서 또는 MyStringList라는 자격을 얻기 위해서 제대로 수행해야 할 책임입니다.
- ✓ MyStringArrayList를 구현한다면 제시한 오퍼레이션을 작은 책임으로 잘 나누어서 구현을 하여야 합니다.
- ✓ MyStringArrayList를 구현하고, 팀 단위로 발표합니다.

```
public interface MyStringList {  
    //  
    int size();  
    boolean isEmpty();  
    boolean contains(String element);  
    MyStringIterator iterator();  
    void add(String element);  
    void add(int index, String element);  
    String get(int index);  
    void remove(Object o);  
    void remove(int index);  
    void addAll(MyStringList sourceList);  
    void clear();  
    String[] toArray();  
}
```



2.3 실습 2-1 – ArrayList 2

- ✓ ArrayList는 말 그대로 내부에 배열을 이용하여 List를 구현하는 실습입니다.
- ✓ 주어지 시간 동안 구현을 완료한 후, 지정한 수강생의 1차 발표가 있고, 토론을 진행하겠습니다.
- ✓ 토론에서 나온 개선 내용을 반영하여, 다시 코딩을 진행한 후, 2차 발표를 진행합니다.
- ✓ 2차 발표와 토론이 끝나면, Clean Code라고 할 수 있는 코드를 리뷰하는 시간을 갖겠습니다.



2.3 실습 2-1 – ArrayList 3

- ✓ Java 언어의 Generic 타입에 익숙한 개발자는 앞의 MyStringList 대신 범용의 MyList<E>로 구현할 수 있습니다.
- ✓ MyStringList 구현이 훨씬 더 간단하지만, 코드는 MyList<E>와 대체로 유사합니다.
- ✓ 따라서, MyStringList를 먼저 깔끔하게 구현한 후에 Generic 타입 버전으로 확장하는 것이 좋은 접근방법입니다.

```
public interface MyStringList {  
    //  
    int size();  
    boolean isEmpty();  
    boolean contains(String element);  
    MyStringIterator iterator();  
    void add(String element);  
    void add(int index, String element);  
    String get(int index);  
    void remove(Object o);  
    void remove(int index);  
    void addAll(MyStringList sourceList);  
    void clear();  
    String[] toArray();  
}
```

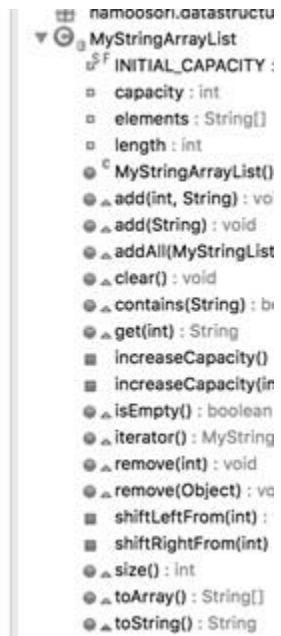
```
public interface MyList<E> {  
    //  
    int size();  
    boolean isEmpty();  
    boolean contains(Object o);  
    MyIterator<E> iterator();  
    void add(E element);  
    void add(int index, E element);  
    E get(int index);  
    void remove(Object o);  
    void remove(int index);  
    void addAll(MyList<? extends E> c);  
    void clear();  
    <T> T[] toArray(T[] a);  
}
```

2.3 실습 2-1 – ArrayList 4– Clean Code 리뷰

- ✓ 토론과 개선 내용을 바탕으로 ArrayList를 구현한 Clean Code를 살펴 봅니다.
- ✓ Clean Code는 간결하며, 읽고 이해하기 쉬운 코드입니다.
- ✓ Clean Code와 Performance Code는 관련은 있지만 서로 다른 관점의 주제입니다.
- ✓ Clean Code에서 Clean 하지 않은 부분은 있습니까? 있다면 어떤 코드가 왜 Clean 하지 않는지 토의 합니다.



```
60      for(int index = 0; index<length; index++) {  
61          //  
62          if (elements[index].equals(o)) {  
63              contained = true;  
64              break;  
65          }  
66      }  
67  
68      return contained;  
69  }  
70  
71 @Override  
72 public MyStringIterator iterator() {  
73     //  
74     return new MyStringIteratorLogic(toArray());  
75 }  
76  
77 @Override  
78 public void add(String e) {  
79     //  
80     if(length == capacity) {  
81         increaseCapacity();  
82     }  
83 }
```



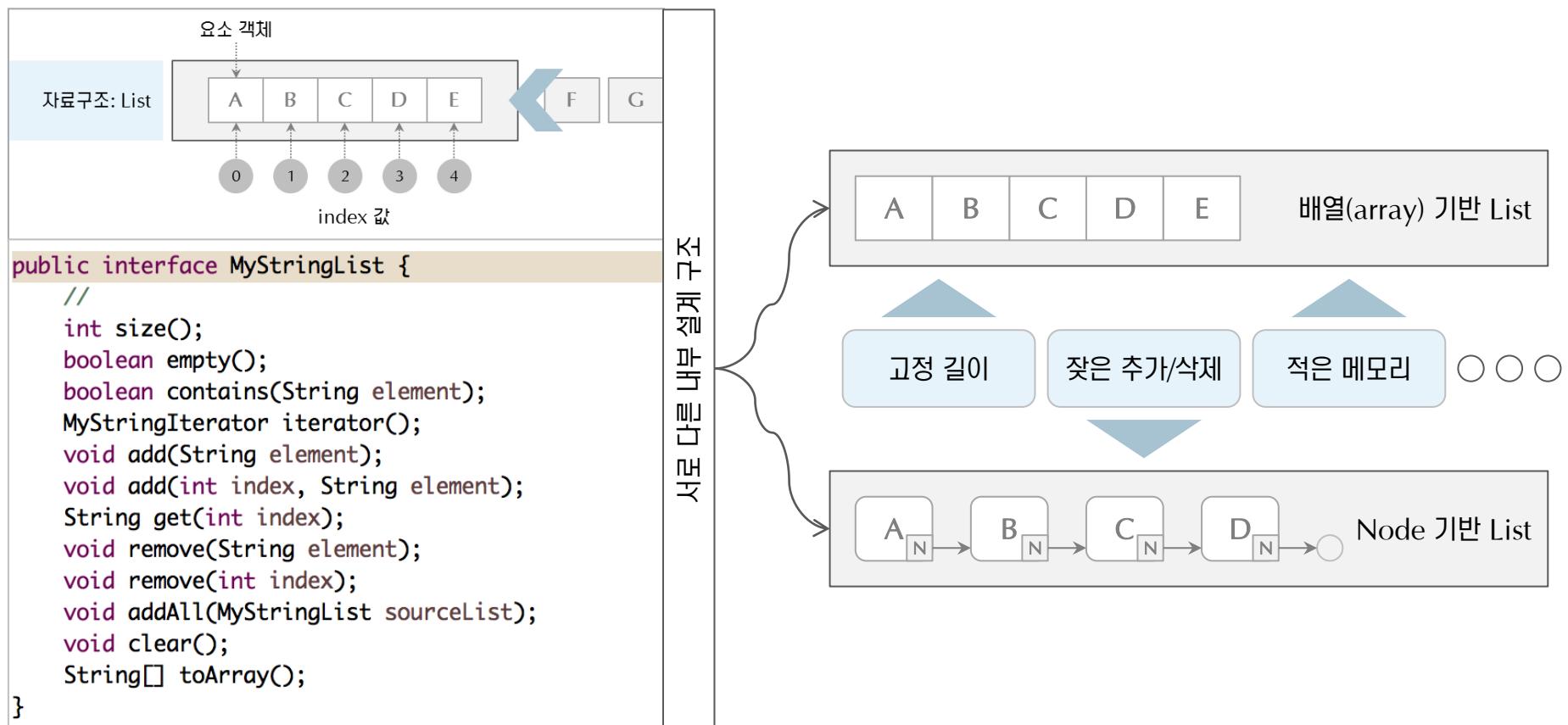
2.4 생각의 흐름: List – 인터페이스

- ✓ List는 Java JDK에서 정의하여 사용하므로, Generic 타입용 MyList, String 타입용 MyStringList로 정의합니다.
- ✓ List의 주요 오퍼레이션은 add(), get(), remove()이고, 나머지는 유필리티 오퍼레이션 성격을 갖고 있습니다.
- ✓ String 타입을 위한 MyStringList를 먼저 구현한 후에, Generic 타입을 위한 MyList로 확장하는 것이 좋습니다.
- ✓ Java List 인터페이스는 JavaDoc을 참조합니다. → <https://docs.oracle.com/javase/8/docs/api/java/util/List.html>

```
public interface MyList<E> {  
    //  
    int size();  
    boolean empty();  
    boolean contains(Object object);  
    MyIterator<E> iterator();  
    void add(E element);  
    void add(int index, E element);  
    E get(int index);  
    void remove(Object object);  
    void remove(int index);  
    void addAll(MyList<? extends E> collection);  
    void clear();  
    <T> T[] toArray(T[] some);  
}  
  
public interface MyStringList {  
    //  
    int size();  
    boolean empty();  
    boolean contains(String element);  
    MyStringIterator iterator();  
    void add(String element);  
    void add(int index, String element);  
    String get(int index);  
    void remove(String element);  
    void remove(int index);  
    void addAll(MyStringList sourceList);  
    void clear();  
    String[] toArray();  
}
```

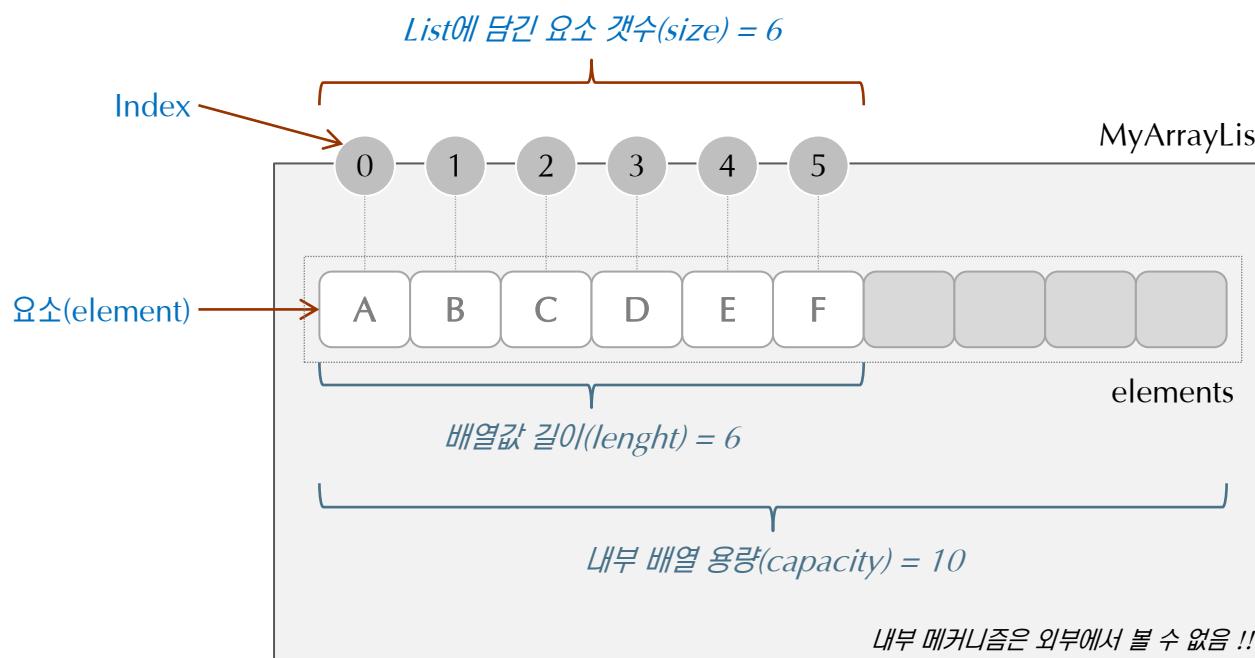
2.4 생각의 흐름: List – 설계 방안

- ✓ List 객체를 개발할 때, 안에서 정보를 다루는 방법에 따라 두 가지로 나눕니다.
- ✓ 배열(array)을 사용하는 List는 ArrayList라고 부릅니다.
- ✓ Node 객체에 요소를 저장하고 다른 Node 객체와 연결(link)하는 방식을 사용하는 List를 LinkedList라고 합니다.
- ✓ 두 가지 List 객체는 각기 장단점을 갖고 있어서 상황에 따라 적절한 것을 선택하여 사용합니다.



2.4 생각의 흐름: ArrayList – 개요

- ✓ ArrayList 객체는 내부에 배열을 갖고 있습니다. 그런데 배열은 고정 길이 저장 공간입니다.
- ✓ 내부의 배열 길이를 담을 수 있는 용량의 뜻으로 capacity라고 합니다. 외부에서는 그 값을 알 수 없습니다.
- ✓ ArrayList 객체는 정해진 용량(capacity)을 초과하여 객체를 넣을 때도 내부 용량을 늘려서 객체를 담아 주어야 합니다.
- ✓ size 값은 현재 List 객체에 담아 둔 객체의 갯수를 의미하며, 밖에서는 이 숫자만 알 수 있습니다.



2.4 생각의 흐름: ArrayList – 추가(add)

- ✓ ArrayList를 구현할 때, 초기 배열 용량(capacity)을 5로 설정한 경우의 예를 들어 봅니다.
- ✓ 프로그램은 1~6 숫자를 String 객체로 변경하여 저장하려 합니다.
- ✓ 앞에서부터(index 0에서부터) String 객체를 하나 하나 채워 나갑니다.
- ✓ 1~5까지 값을 추가할 때는 처음에 정한 용량이 5이므로 아무 문제없이 담을 수 있습니다.

```
MyStringArray newArray = new MyStringArray();
```



✓ Capacity : 5
✓ Size : 0

```
for(int i=1; i<6; i++) {  
    newArray.add(String.valueOf(i));  
}
```



✓ Capacity : 10
✓ Size : 6

처음에 1 추가 →



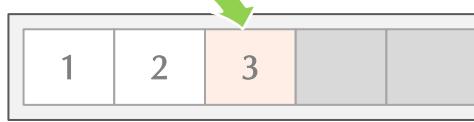
✓ Capacity : 5
✓ Size : 1

끝에 2 추가 →



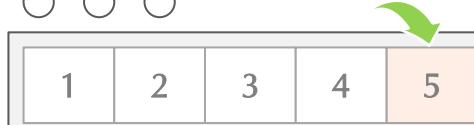
✓ Capacity : 5
✓ Size : 2

끝에 3 추가 →



✓ Capacity : 5
✓ Size : 3

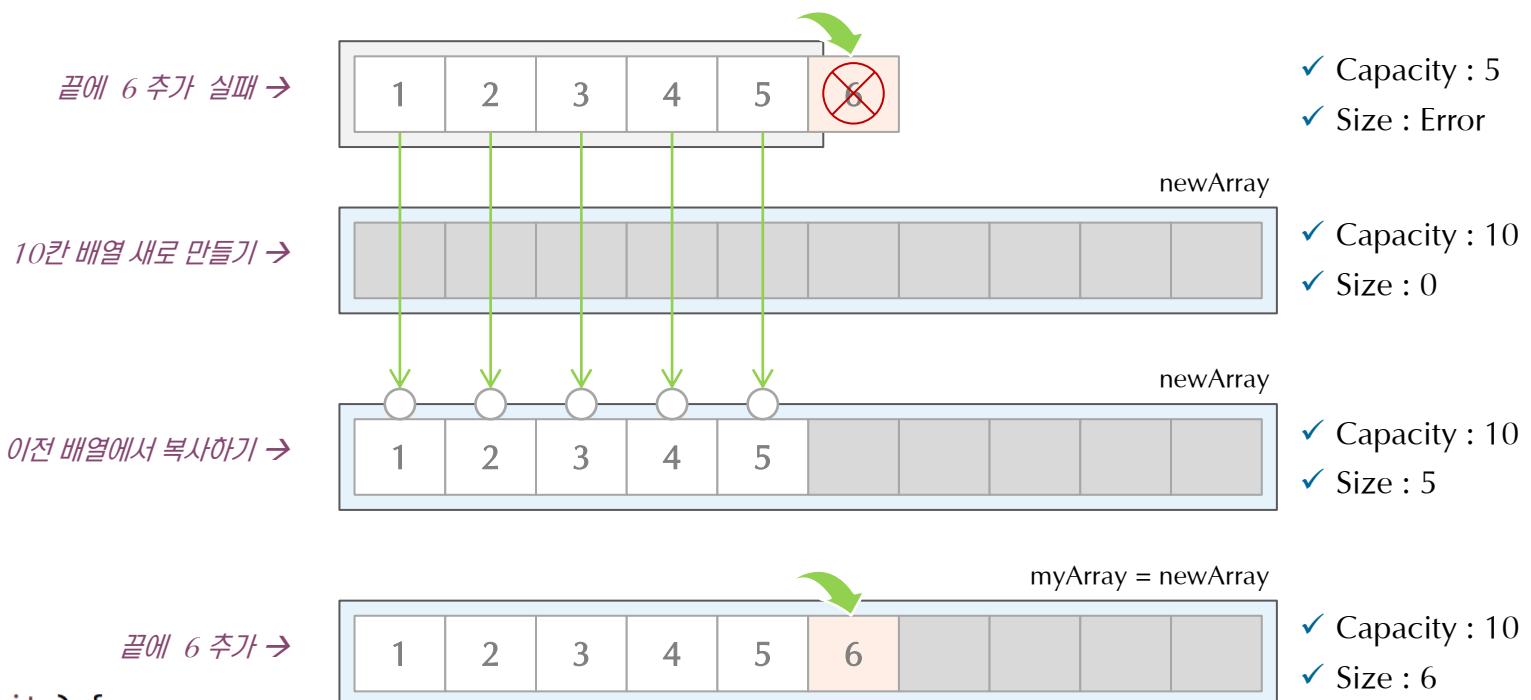
끝에 5 추가 →



✓ Capacity : 5
✓ Size : 5

2.4 생각의 흐름: ArrayList – 용량(capacity) 늘리기

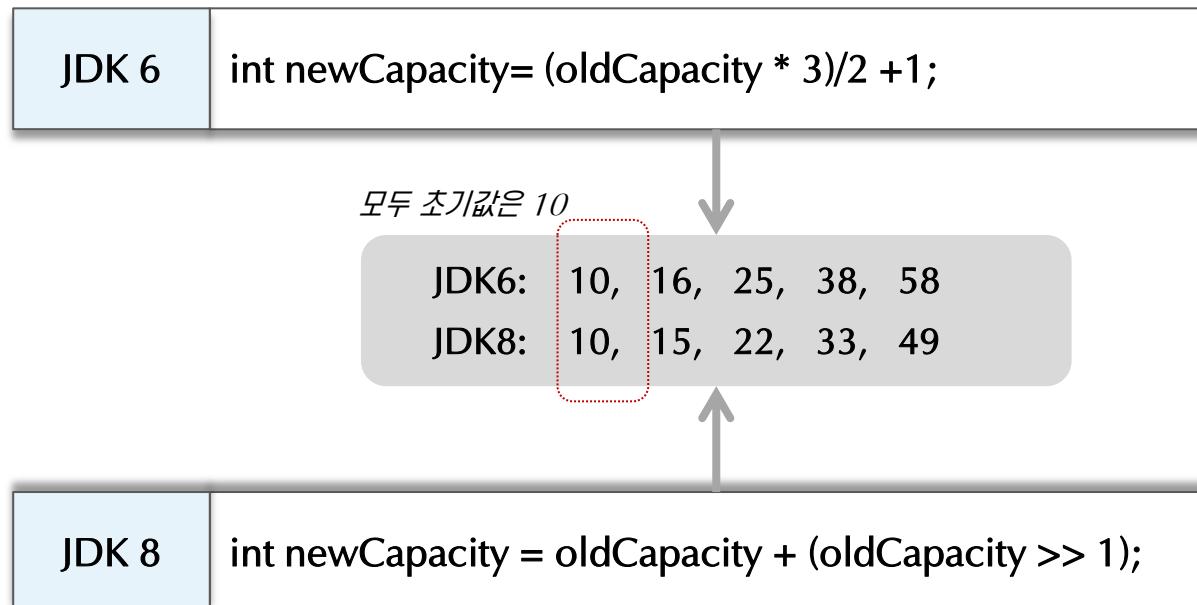
- ✓ 여섯 번째 요소인 “6”을 저장하려면 할 때, 더 이상 남은 공간이 없습니다. 이럴 경우, 못 담아 준다면 이것은 ArrayList 객체가 아니라 그냥 배열일 뿐이며, ArrayList 객체가 될 자격이 없습니다.
- ✓ 내부에 미리 준비한 배열의 용량(Capacity)은 초기값으로부터 지속적으로 증가하며, 높은 수준의 정책이 필요합니다.
- ✓ 예제에서는 capacity 의 두 배에 해당하는 배열을 새로 만들고, 이전 배열의 값을 이곳으로 복사하고, 이름을 변경합니다.



```
private void increaseCapacity(int capacity) {  
    //  
    String[] newElements = new String[capacity];  
    System.arraycopy(elements, 0, newElements, 0, length);  
    elements = newElements;  
}
```

2.4 생각의 흐름: ArrayList – 용량(capacity) 결정하기

- ✓ 새로운 용량(capacity)을 합리적으로 결정하는 것은 매우 어려운 일입니다. 상황에 종속적이기 때문입니다.
- ✓ 너무 많은 용량을 할당해도 사용하지 않으면 문제일 수 있고, 너무 적게 늘리면 반복해서 늘려야 하는 문제가 있습니다.
- ✓ JDK6와 JDK8은 새로운 용량을 정하는 규칙이 서로 다릅니다.
- ✓ 주의] 예제에서는 JDK 규칙을 따르지 않고, 초기값 10에 이후 계속 초기값을 더하는 단순한 규칙을 사용합니다.



[Java JDK에서 용량(capacity)을 늘리는 방식]

2.4 생각의 흐름: ArrayList – 끝이 아닌 곳에 삭제하기

- ✓ 리스트에서 오른쪽 끝에 있는 요소를 삭제하면 삭제로 끝난다.
- ✓ 하지만, 가운데 요소를 삭제하면 그 빈칸을 메워야 한다.
- ✓ 빈칸으로 왼쪽의 모든 요소들을 이동하여야 한다.
- ✓ 소스코드 참조 → shiftLeftTo(index)

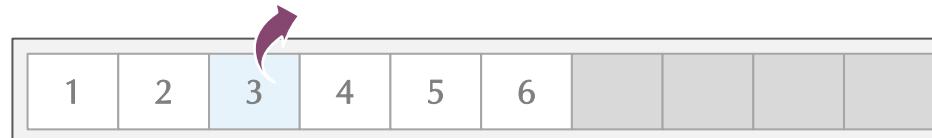
```
private void shiftLeftTo(int index) {  
    //  
    for(int i=index+1; i<length; i++) {  
        elements[i-1] = elements[i];  
    }  
}
```

newArray.remove(2); // index 2 지점의 요소 제거

요소에 null 할당(삭제)→



✓ Capacity : 10
✓ Size : 6



✓ Capacity : 10
✓ Size : 6



4를 왼쪽으로 이동 →



5를 왼쪽으로 이동 →



6을 왼쪽으로 이동 →



✓ Capacity : 10
✓ Size : 5

2.4 생각의 흐름: ArrayList – 끝이 아닌 곳에 추가하기

- ✓ 리스트에서 오른쪽 끝에 있는 요소를 삭제하면 삭제로 끝난다.
- ✓ 하지만, 가운데 요소를 삭제하면 그 빈칸을 메워야 한다.
- ✓ 빈칸으로 왼쪽의 모든 요소들을 이동하여야 한다.
- ✓ 소스코드 참조 → shiftRigthFrom(index)

```
private void shiftRightFrom(int index) {  
    //  
    for(int i=length; i>index; i--) {  
        elements[i] = elements[i-1];  
    }  
}
```

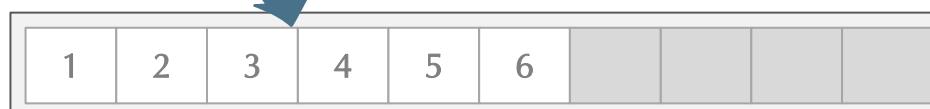
newArray.add(3,"A"); // index 3 지점에 요소 추가

6을 오른쪽으로 이동→



✓ Capacity : 10
✓ Size : 6

5를 오른쪽으로 이동→



✓ Capacity : 10
✓ Size : 6

4를 오른쪽으로 이동→



빈 공간에 A 할당→



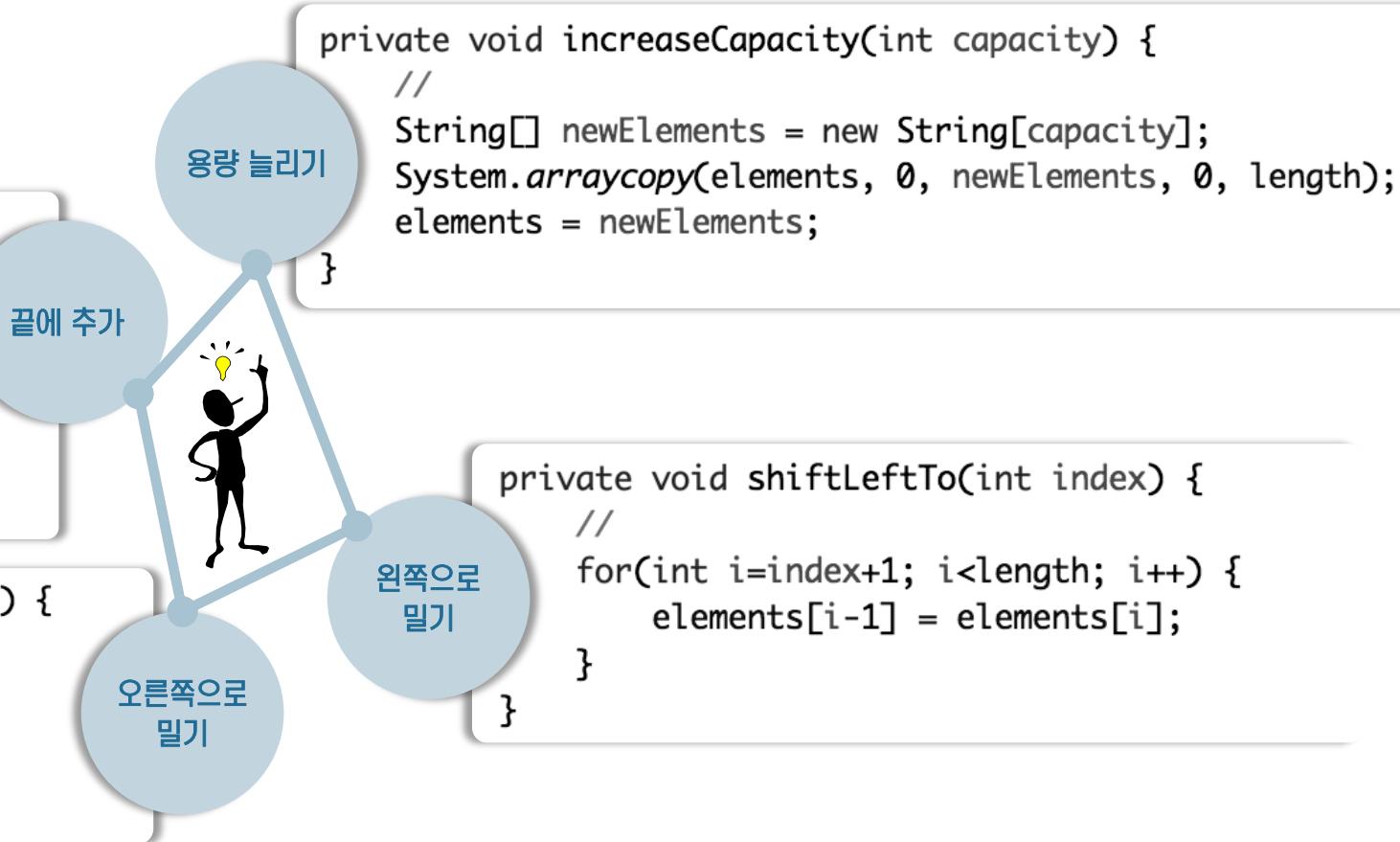
✓ Capacity : 10
✓ Size : 5

2.4 생각의 흐름: ArrayList – 설계 포인트

- ✓ ArrayList의 설계 포인트는 단순 추가, 용량 늘리기, 왼쪽으로 밀기, 오른쪽으로 밀기 네 가지입니다.
- ✓ ArrayList는 내부에서 원시 데이터 타입인 array를 사용하고 있고, array는 길이가 고정되어 있어, 용량을 늘려야 합니다.
- ✓ array 가운데 새로운 요소를 입력하려 할 때, 해당 지점을 비우기 위해 우측 요소들을 오른쪽으로 한 칸씩 밀어야 합니다.
- ✓ array 가운데 새로운 요소를 삭제하려면, 삭제되어 비어있는 칸을 메우기 위해 왼쪽으로 한 칸씩 당겨야 합니다.

```
public void add(String element) {  
    //  
    if(length == capacity) {  
        increaseCapacity();  
    }  
    this.elements[length] = element;  
    this.length++;  
}
```

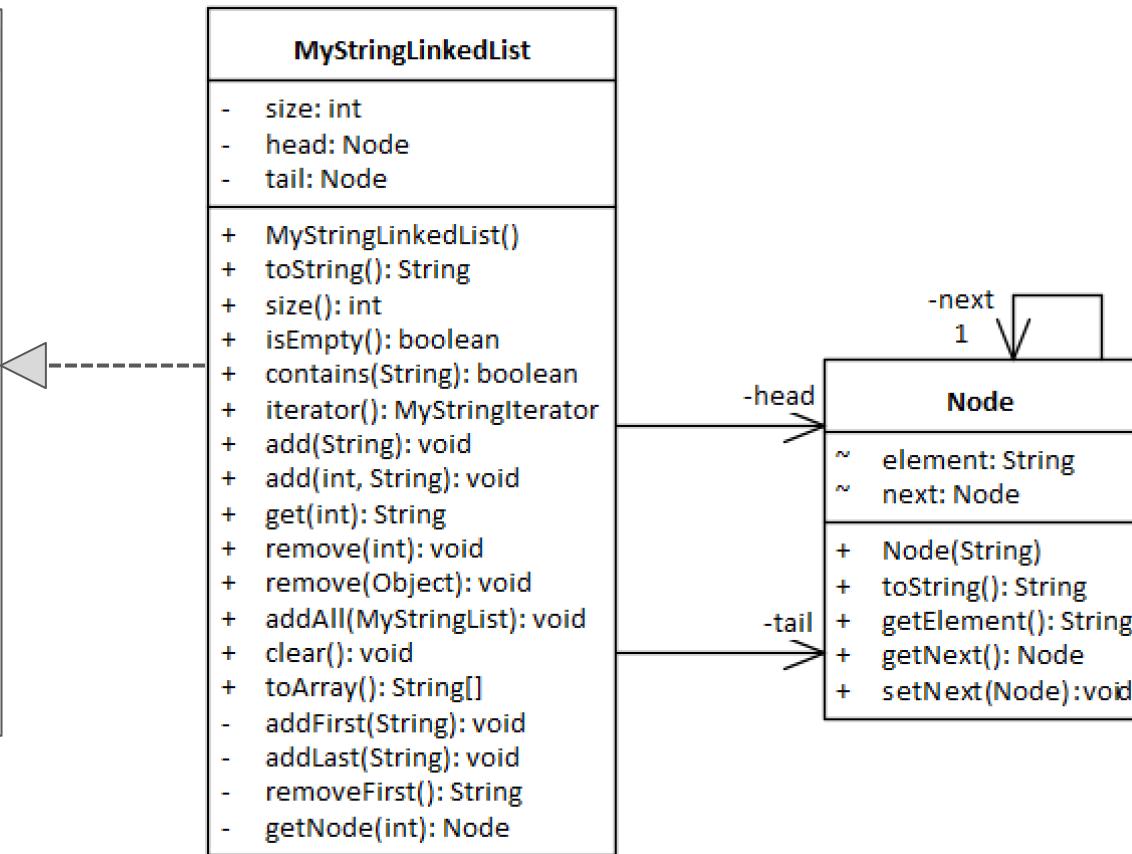
```
private void shiftRightFrom(int index) {  
    //  
    for(int i=length; i>index; i--) {  
        elements[i] = elements[i-1];  
    }  
}
```



2.5 실습 2-2 – LinkedList

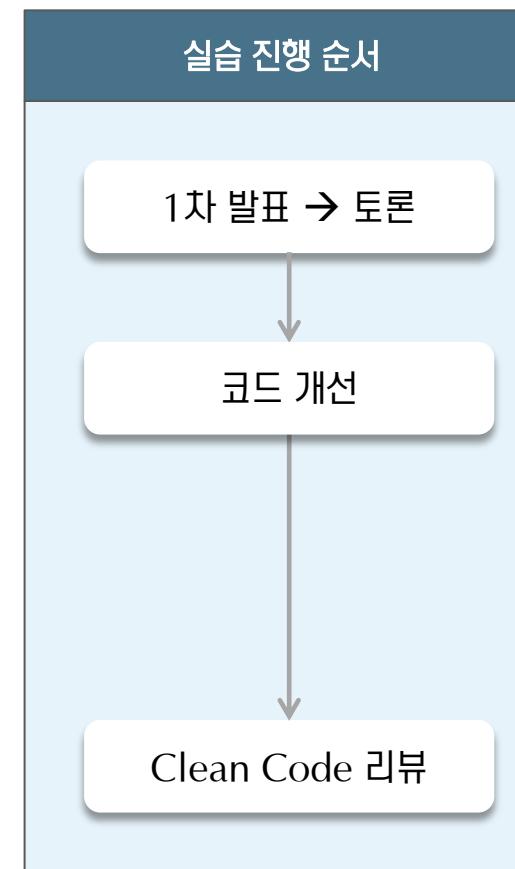
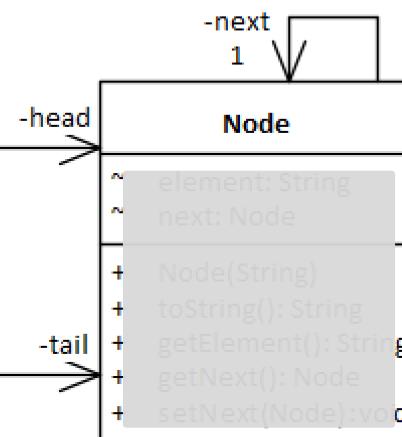
- ✓ 같은 인터페이스 MyStringList에 대해, 이번에는 Link Node를 이용하여 LinkedList 방식으로 구현합니다.
- ✓ Node 클래스가 필요합니다. 이전 실습의 토론 결과를 반영하여, 독자성을 가진 책임으로 잘 나누어서 구현합니다.
- ✓ LinkedList의 특성과 장점을 생각하면서 구현합니다.
- ✓ 독자성을 가진 작은 책임으로 나누지 못하면 Clean Code 목표를 달성하기 어렵습니다.

```
public interface MyStringList {  
    //  
    int size();  
    boolean isEmpty();  
    boolean contains(String element);  
    MyStringIterator iterator();  
    void add(String element);  
    void add(int index, String element);  
    String get(int index);  
    void remove(Object o);  
    void remove(int index);  
    void addAll(MyStringList sourceList);  
    void clear();  
    String[] toArray();  
}
```



2.5 실습 2-2 – LinkedList

- ✓ ArrayList 실습의 경험에 있으므로 발표와 토론은 한 차례로 진행합니다.
- ✓ 객체와 객체의 이름(label), 객체 아이디에 대한 명확한 생각을 가져야 합니다.



2.5 실습 2-2 – LinkedList – Clean Code 리뷰

- ✓ List 외에 Queue, Stack 등도 생각해 볼 수 있습니다. 무엇을 구현하든 지켜야 할 원칙은 비슷합니다.
- ✓ 이제 Clean Code에 대한 기본 개념은 충분히 다져진 상태입니다.
- ✓ LinkedList와 ArrayList의 차이점은 무엇이며, 언제 어떤 List를 사용해야 하는지 토의 합시다.
- ✓ Clean Code에 대해 동의하십니까? 동의하지 못한다면 어떤 부분을 왜 동의하지 못하는지 토의 합니다.

The screenshot shows the file structure of a Java project named 'datastructure'. The 'list' package contains several classes: MyArrayList.java, MyStringArrayList.java, MyList.java, MyStringList.java, and MyLinkedList.java. The MyStringLinkedList.java file is currently selected and its code is displayed in the center pane. The right pane shows the class hierarchy for MyStringLinkedList, including its methods like addLast, clear, toArray, etc., and its inner class Node.

```
String newElement = iter.next();
addLast(newElement);
}

@Override
public void clear() {
    //
    while(size > 0) {
        removeFirst();
    }
    head = null;
    tail = null;
}

@Override
public String[] toArray() {
    //
    String[] newElements = new String[size];
    Node node = head;
    for(int i=0; i<size; i++) {
        newElements[i] = node.getElement();
        node = node.getNext();
    }
}
```

2.6 토론

- ✓ List를 두 가지 서로 다른 방식으로 Clean Code로 구현했습니다.
- ✓ 어떤 방식으로 구현을 하든 Clean Code를 얻는데 필요한 원칙을 같습니다.
- ✓ 실습 가운데 범한 오류를 없애려면 다음 가이드를 생각해 봅니다.
- ✓ 클래스 내부 범위에서 이 가이드는 어떤 코드를 작성하든 유효합니다.

1. 자연 언어로 생각하고, 프로그래밍 언어로 구현합니다.
2. 적절한 일의 크기로 나눕니다. 재사용, 작업 독립성, ...
3. 오퍼레이션에서 탈출 지점은 가능한 한 곳으로.
4. 전제 조건은 미리 필터링 하고 들어갑니다.
5. 최대 Indentation을 3으로 설정하고 로직을 구성합니다.
6. Naming에서 실패하면 Clean Code는 없습니다.

요약

- ✓ List 컬렉션은 명확한 책임을 갖고 있으며, 그 책임은 인터페이스의 오퍼레이션으로 표현합니다.
- ✓ 외부에서 보이는 이 책임을 수행할 때, 내부에서는 또 다른 작은 책임으로 나누어서 수행합니다.
- ✓ 책임의 크기를 적절하게 잘 나누어 개발함으로써 읽기 편하고 이해하기 쉬운 Clean Code를 작성하여 보았습니다.
- ✓ 어떤 유형의 개발을 하든 이런 코딩 원칙을 일관성 있게 적용한다면 Clean Code를 얻을 수 있습니다.

```
public interface MyStringList {  
    //  
    int size();  
    boolean isEmpty();  
    boolean contains(String element);  
    MyStringIterator iterator();  
    void add(String element);  
    void add(int index, String element);  
    String get(int index);  
    void remove(Object o);  
    void remove(int index);  
    void addAll(MyStringList sourceList);  
    void clear();  
    String[] toArray();  
}
```



3. Clean Code – 협업

3.1 현장 스케치

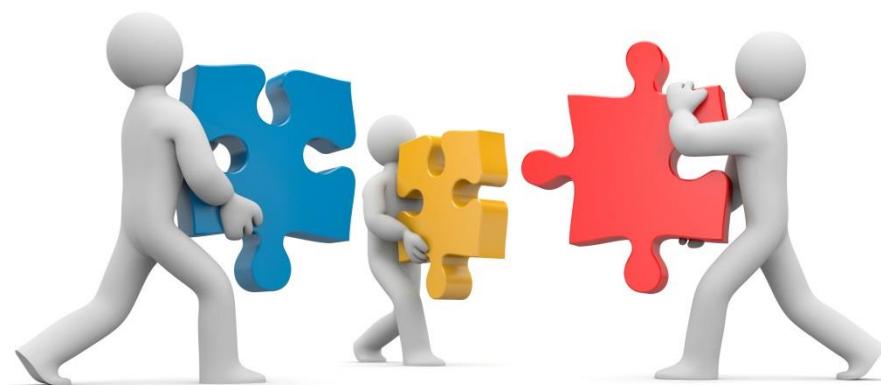
3.2 시나리오

3.3 모델

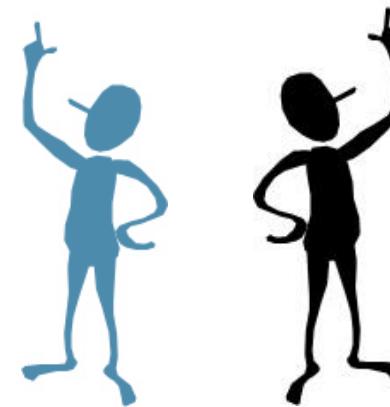
3.5 프로그래밍

들어가기 전에...

- ✓ 누군가에게 어떤 일을 시켜야 하는 상황입니다.
- ✓ 힘이 아주 쎈 한 사람에게 시켜야 할지, 협업을 잘 하는 팀에게 맡겨야 할지 고민을 해 보세요.
- ✓ 일은 이번 만이 아니고 다음에도 또 있고, 그 일의 내용은 자주 변경됩니다.
- ✓ 여러분의 선택은 무엇입니까?



<https://planopedia.com/wp-content/uploads/2014/10/Project-team.jpg>

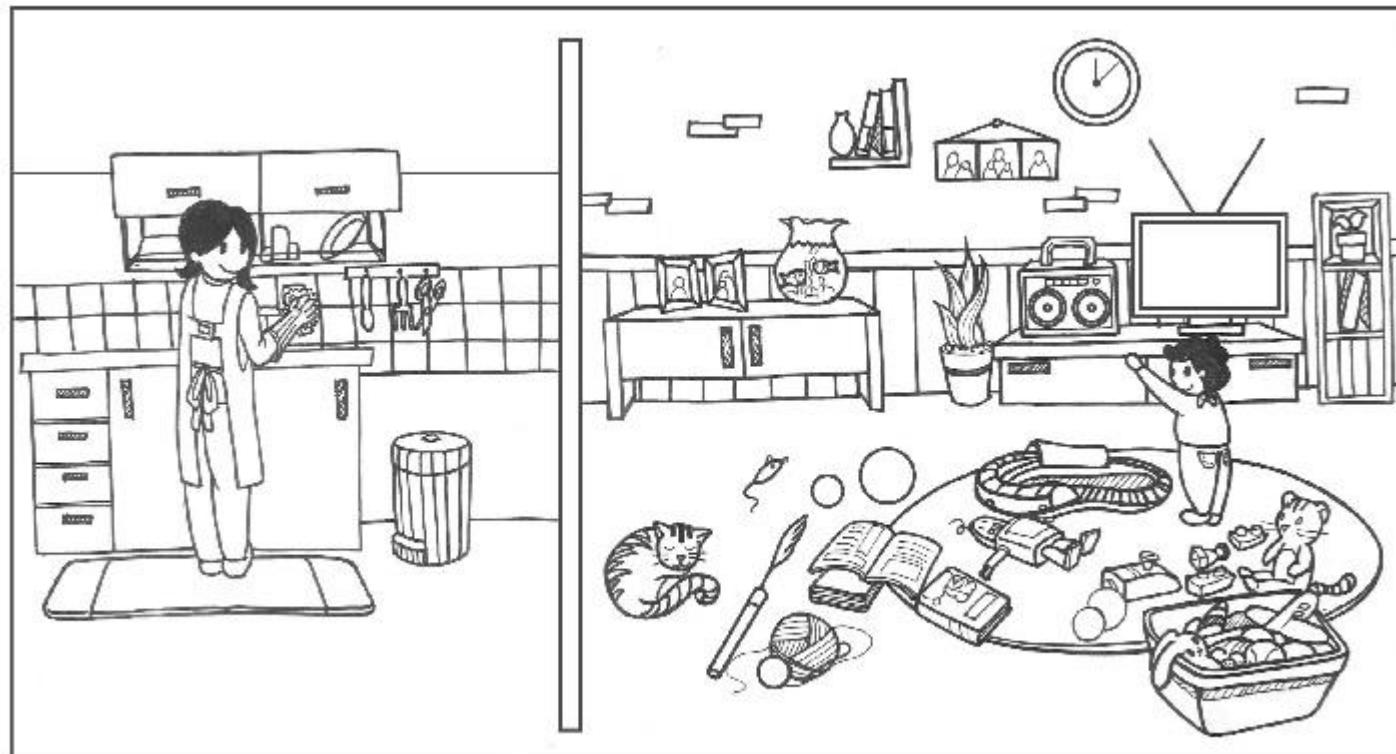


[출처] https://static.giantbomb.com/uploads/original/0/7383/2283313-andre_the_giant_giant.png



3.1 현장 스케치

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



3.2 시나리오 1

- ✓ 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 티브리 라디오가 놓여 있다.
엄마는 라디오를 듣고 싶지만 젖은 고무장갑을 끼고 있어서 켜 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: 흄... 그런데, 아직도 소리가 좀 작아요. 조금 더 높여 주세요.

민수: 네. (다시 소리를 높인다.) 높였어요.

라디오: 뉴스를 말씀드리겠습니다..

엄마: 민수야 고마워. 참 잘 들리는구나.

3.2 시나리오 2

- ✓ “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.

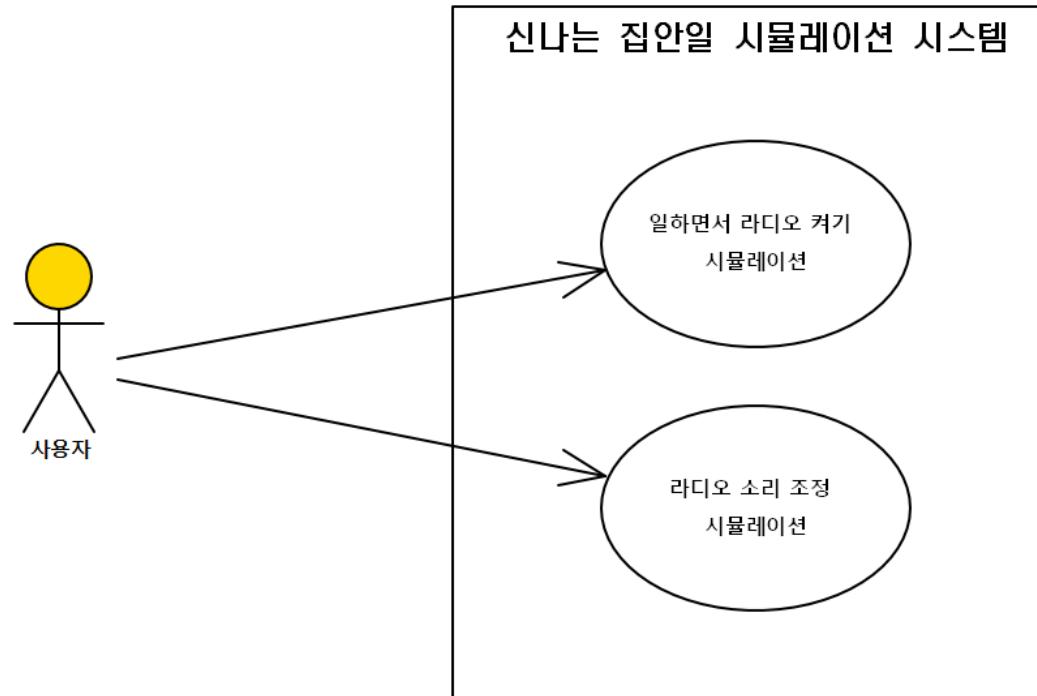
<Yeongmi:Mom> 그래, 그럼 라디오 좀 켜 줄래?

<Minsoo:Child> 예, 라디오 켤께요.

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

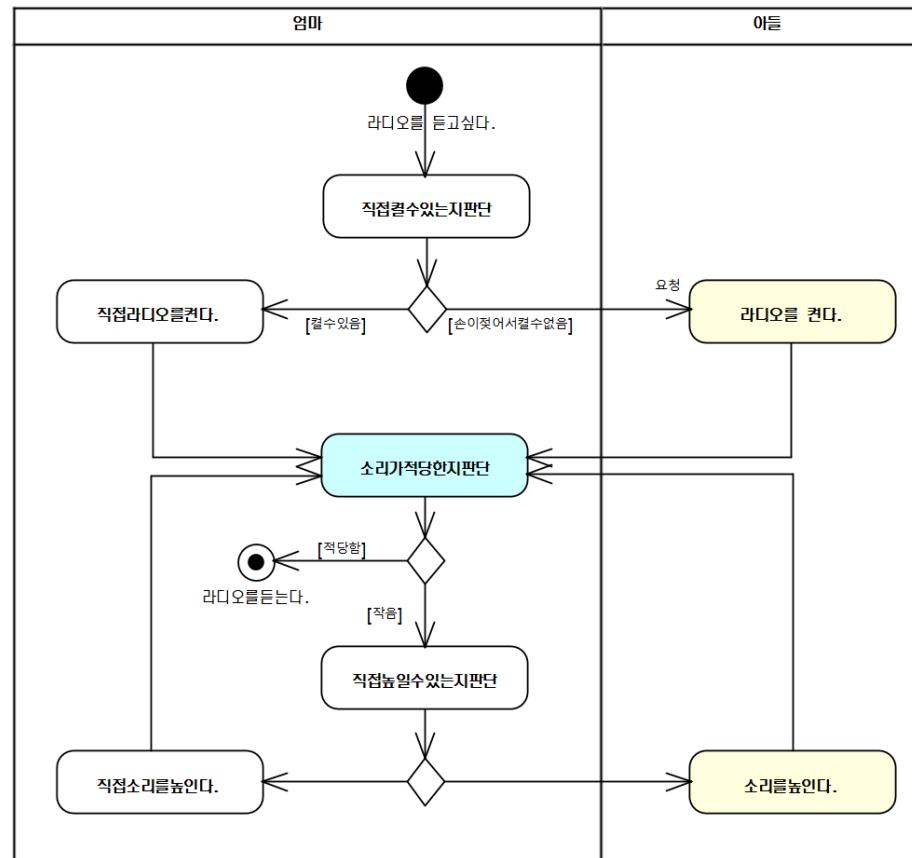
3.3 모델 [1/14]: 기능

- ✓ 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.



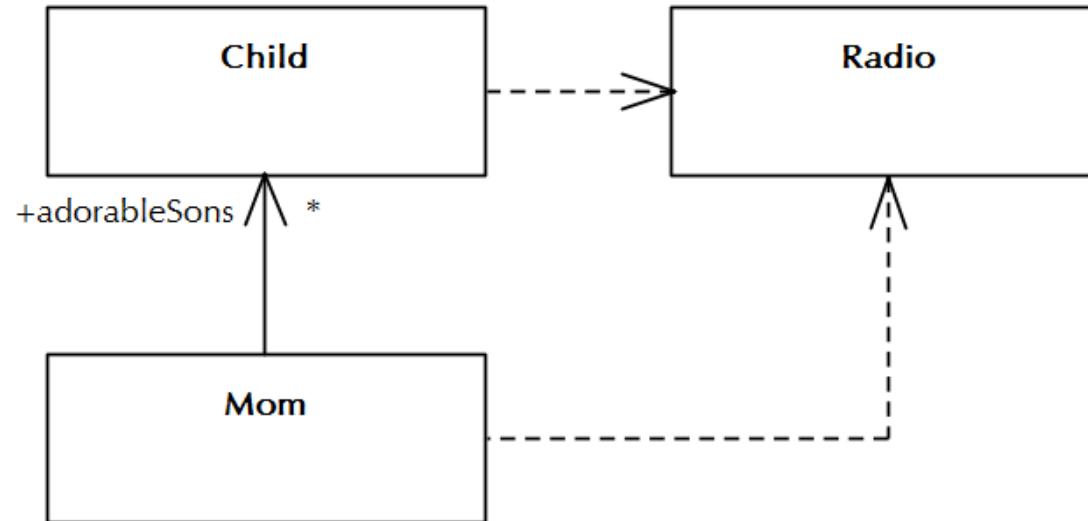
3.3 모델 (2/14) : 업무 흐름

- ✓ 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.



3.4 모델(3/14): 참여 클래스1

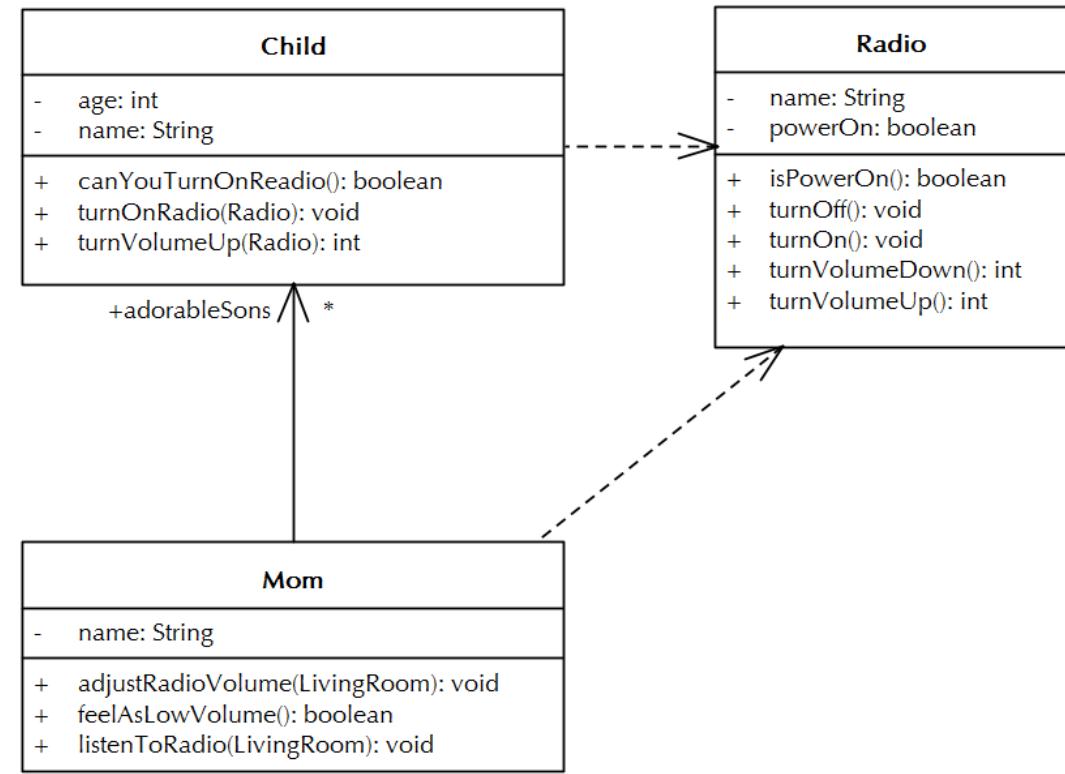
- ✓ 요구사항 명세를 마무리 하고 각 클래스가 어떤 역할과 책임을 가지고 시나리오에 참여하는지 고민을 합니다.
- ✓ 전체 시나리오를 진행되려면 각 클래스가 어떤 것인지를 알고, 어떤 활동을 해야 하는가를 생각해 봅니다.
- ✓ 처음 요구사항 명세에서 발견할 수 있는 클래스들은 엄마, 아이, 그리고 라디오 세 개입니다.



3.4 모델(4/14): 참여 클래스2

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

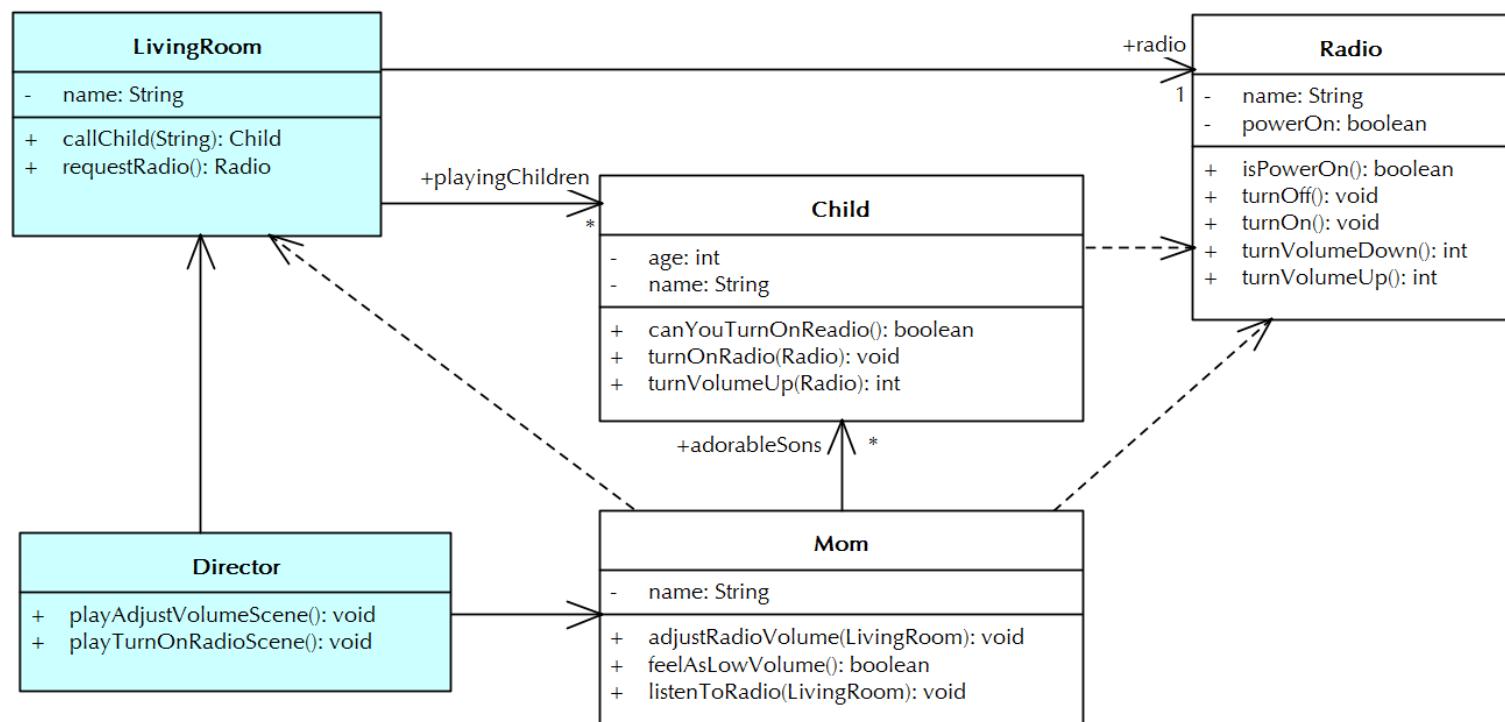


3.4 모델(5/14): 참여 클래스3

- ✓ 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.

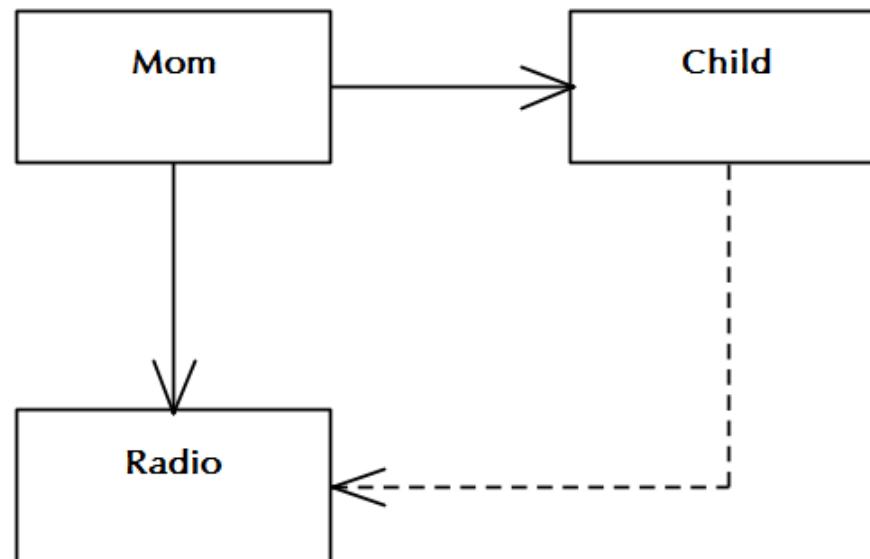
거실의 역할과 책임
✓ 라디오를 갖고 있다.
✓ 아이들이 놀 공간을 제공한다.
✓ 누군가 아이를 찾으면 불러준다.
✓ 향후 어떤 가전 제품이든 받아 준다.

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



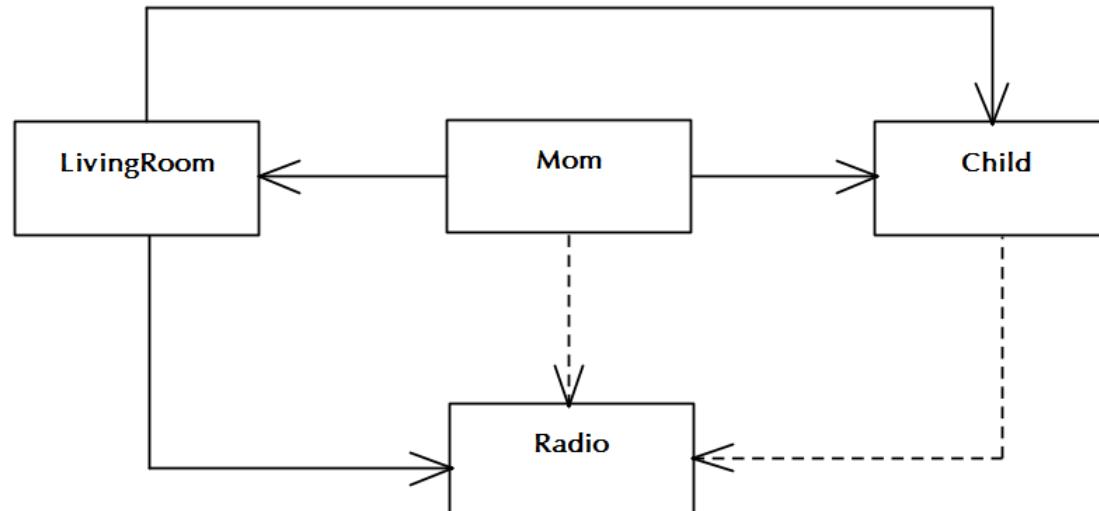
3.4 모델(6/14): 관계1

- ✓ 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.



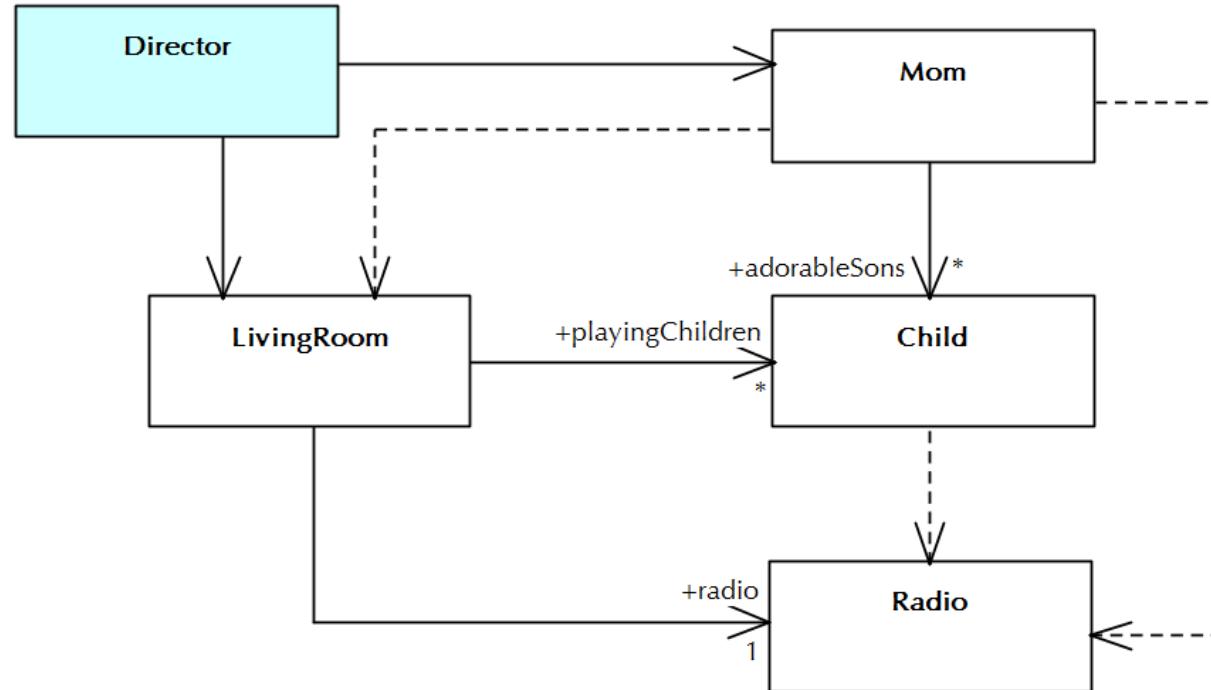
3.4 모델(7/14): 관계2

- ✓ 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?



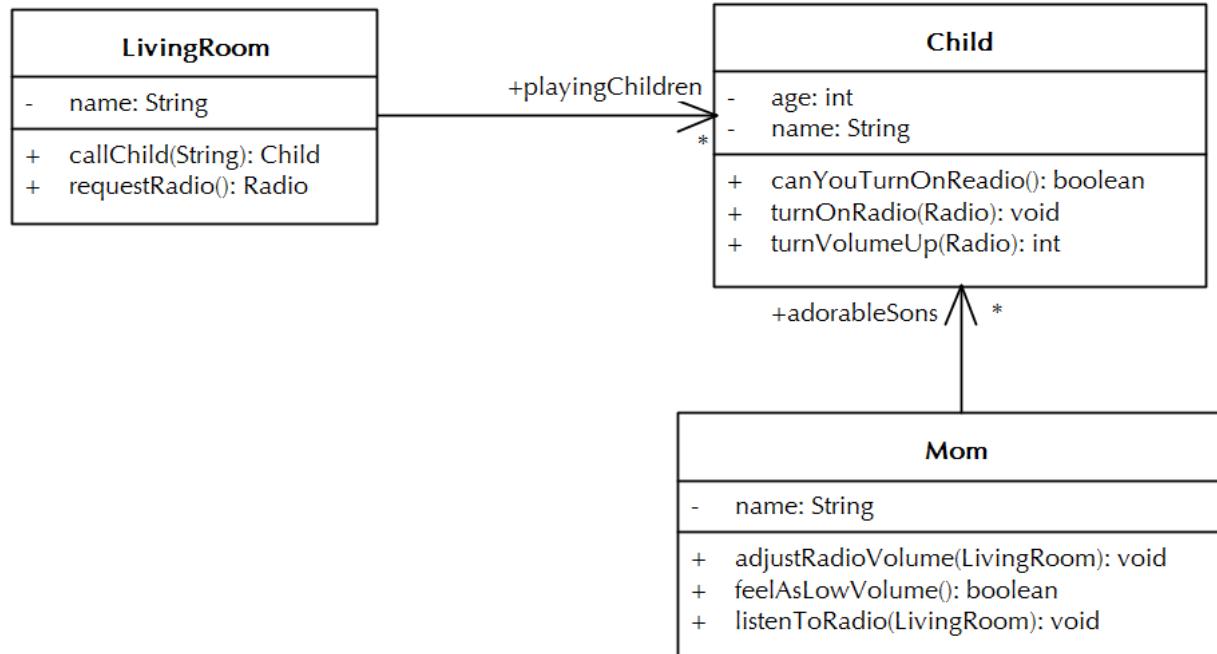
3.4 모델(8/14): 관계3

- ✓ 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.



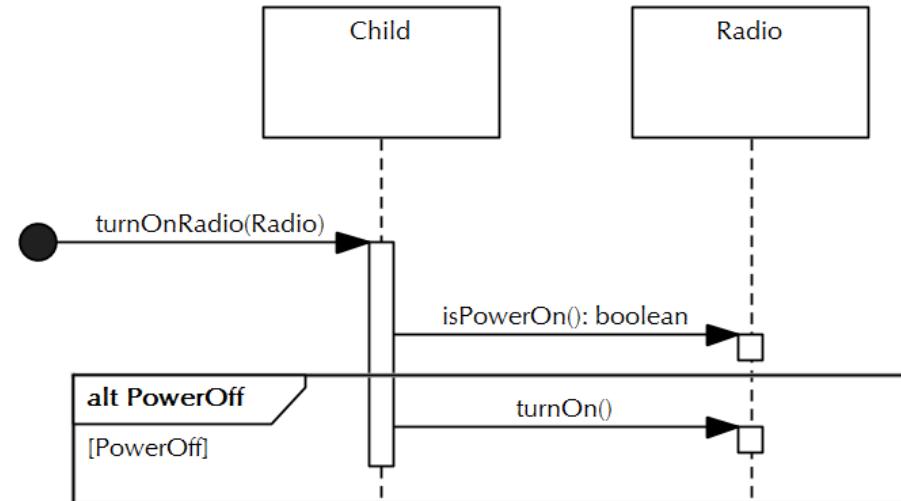
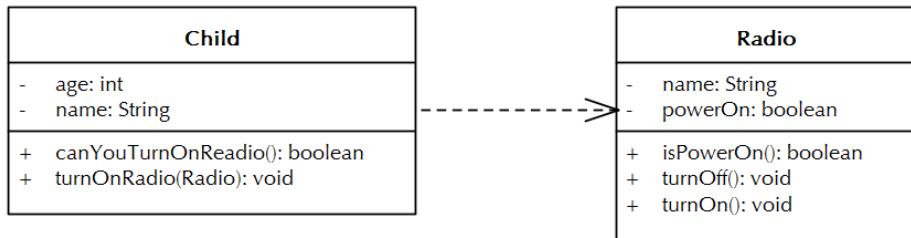
3.4 모델(9/14): 역할

- ✓ 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.
- ✓ 실세계(real world)에서도 A라는 사람은 아이들에게는 아빠이지만, 아내에게는 남편이며, 직장 후배들에게는 선배입니다.



3.4 모델(10/14): 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.

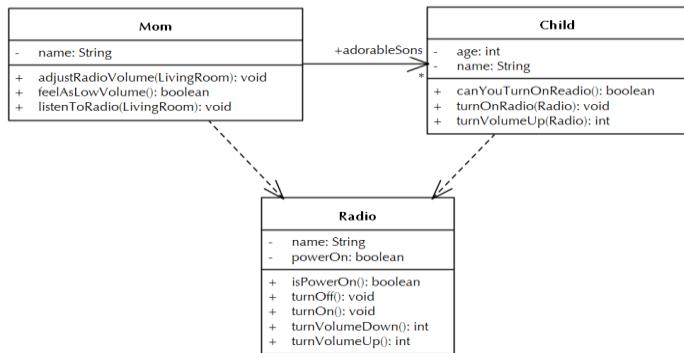


1. Radio상태를 Radio가 가지고 있을 때,
 - 1.1 켜져 있는지 라디오에게 물어 본다. 켜져 있지 않으면 켠다.
 - 1.2 라디오가 켜져 있으면 그냥 리턴하고 그렇지 않으면 켠다.
2. Radio상태를 Child가 가지고 있을 때,
 - 2.1 상태를 알고 있으므로, 켜져 있으면 그대로 둔다.
 - 2.2 상태를 알고 있으므로, 켜져 있지 않으면 켠다.

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

3.4 모델(11/14): 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.

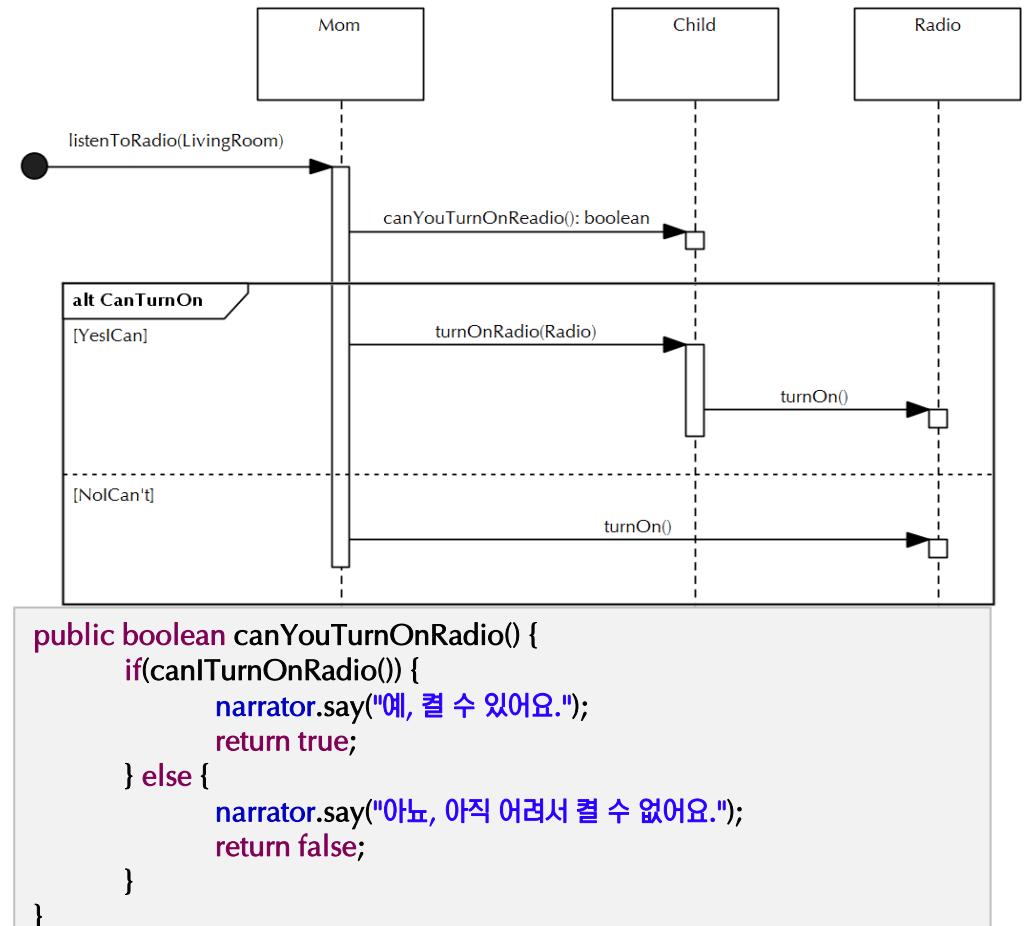


```

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

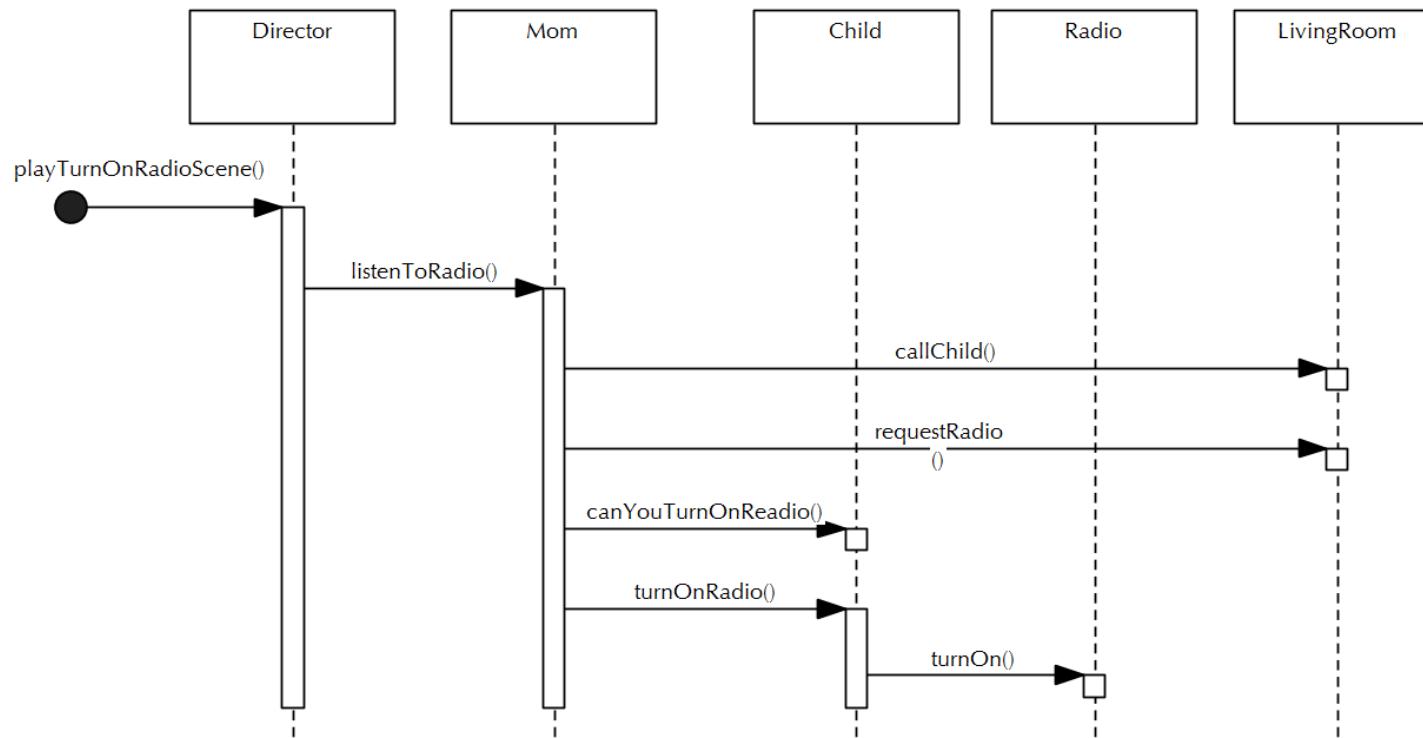
    if (radio.isPowerOn()) {
        return;
    }

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



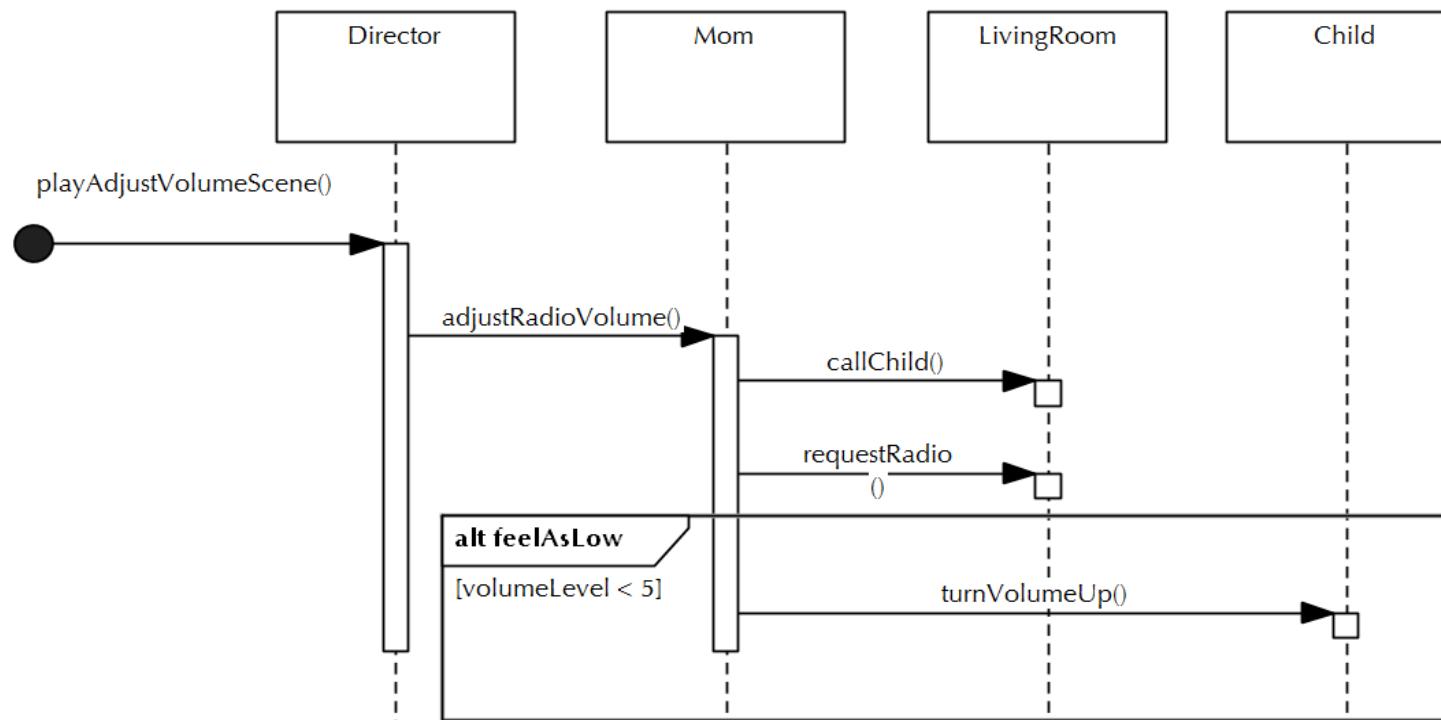
3.4 모델(12/14): 협업 3 [라디오 듣기]

- ✓ 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.



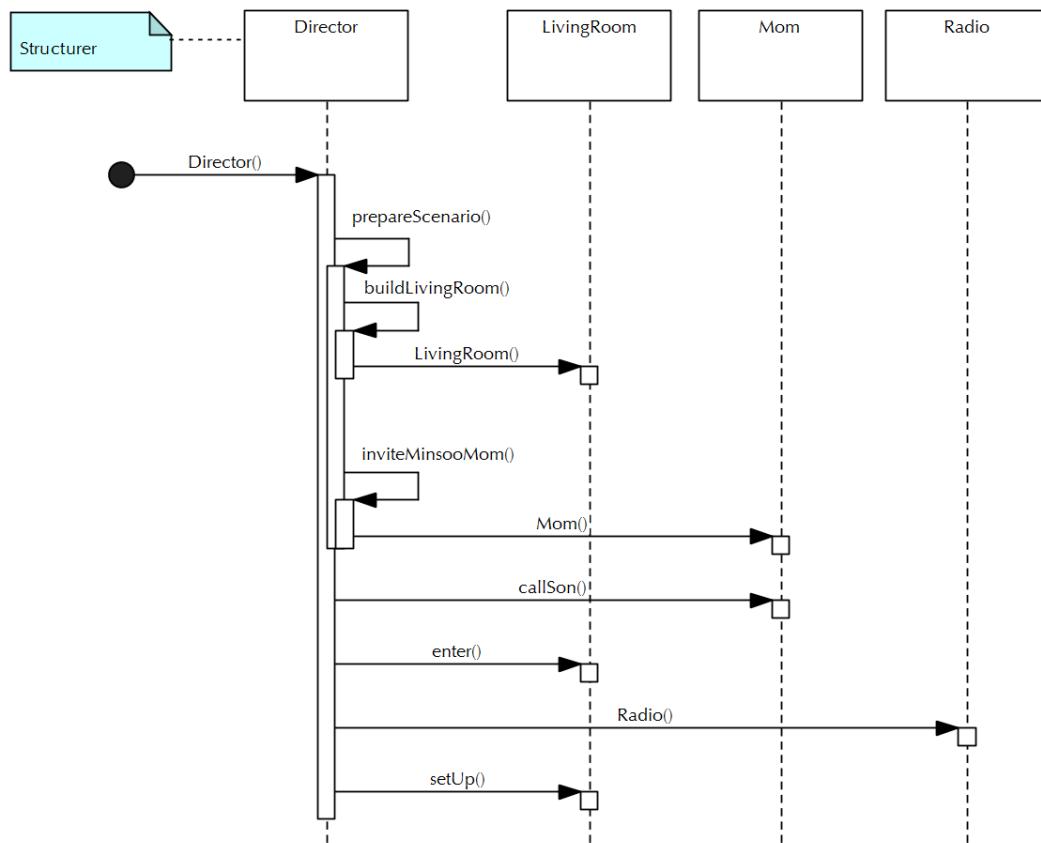
3.4 모델(13/14): 협업4 [소리조정]

- ✓ 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.



3.4 모델(14/14): 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");  
    }  
}
```

3.5 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;
    }
}

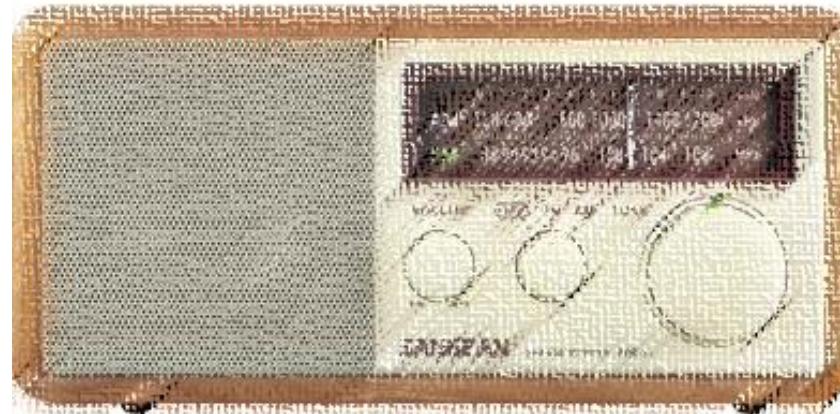
public boolean canYouAdjustVolume() {
    //
    narrator.say("소리를 조정할 수 있나요?");
    if(canIAdjustVolume()) {
        narrator.say("예, 할 수 있어요.");
        return true;
    } else {
        narrator.say("아뇨, 아직 어려서 할 수 없어요.");
        return false;
    }
}

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

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

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?



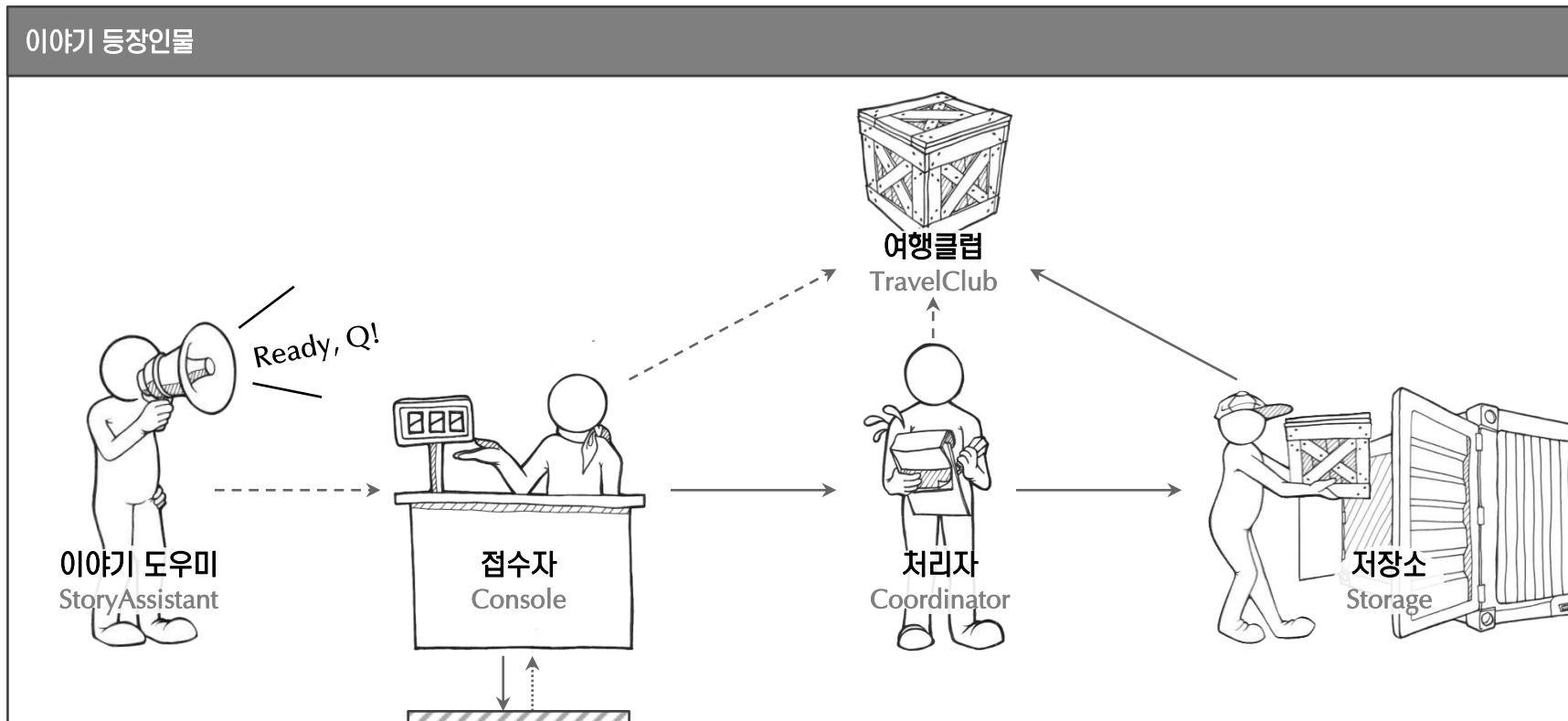


4. 레이어

- 4.1 실습 4-1
- 4.2 자원 접근
- 4.3 실습 4-2
- 4.4 정보시스템과 레이어
- 4.5 레이어드
- 4.6 레이어와 티어
- 4.7 서브시스템과 레이어

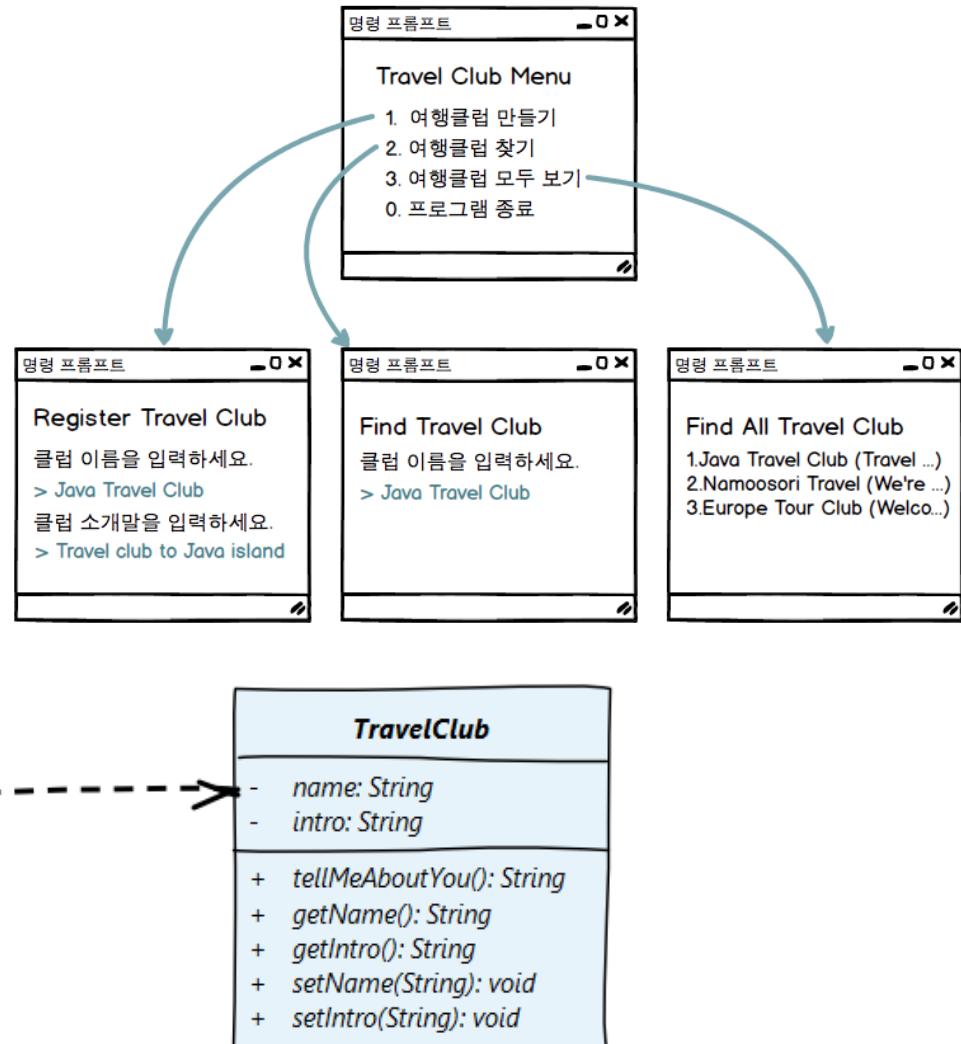
4.1 실습 4-1 – TinyClub

- ✓ 아주 작은 애플리케이션을 만들려고 합니다. 이름은 TinyClub 입니다.
- ✓ 클럽 정보를 등록하고, 조회할 수 있는 아주 간단한 애플리케이션입니다.
- ✓ 아무리 작은 애플리케이션이라 하더라고, 그 속에서 기본적으로 갖추어야 하는 것이 있습니다.
- ✓ 그 내용을 생각하며, 가장 빠른 속도로 TravelClub(name, intro)을 등록하고, 조회하는 프로그램을 작성합니다.



4.1 실습 4-1 – TinyClub

- ✓ UI는 콘솔 기반의 Command Line Interface로 구현합니다.
- ✓ 그 외 나머지는 모두 Java로 구현합니다.
- ✓ 데이터는 메모리에 Map 형식으로 저장합니다.



4.1 실습 4-1 – TinyClub

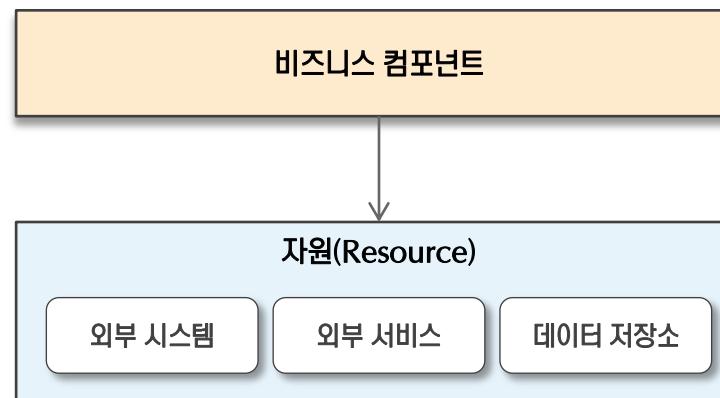
- ✓ TinyClub 애플리케이션은 TravelClub, ClubCoordinator, ClubConsole, ClubStorage로 구성되어 있습니다.
- ✓ 네 개 밖에 없지만, 각 클래스는 애플리케이션에서 각 고유의 역할을 담당하고 있습니다.
- ✓ 각 역할 영역은 미래의 확장이라는 개념을 갖고 있습니다.
- ✓ ClubConsole → GUI → WebUI, Map 저장소 → 파일 저장소 → RDB 저장소 → 문서 저장소

The screenshot shows the Eclipse IDE interface. On the left, the Project Explorer view displays a project named 'javastory.club.stage1' with a master branch. Inside the 'src' folder, there is a package named 'javastory.club.stage1' containing several source files: 'step1.java', 'step2.java', 'step3.java', 'step31.java', 'step4.java', 'ClubConsole.java', 'ClubCoordinator.java', 'ClubStorage.java', 'StoryTeller.java', 'TravelClub.java', 'step41.java', and 'step42.java'. The file 'ClubConsole.java' is currently selected and highlighted with a brown bar at the bottom of the list. On the right, the code editor window shows the content of 'ClubConsole.java':

```
14 public class ClubConsole {
15     //
16     private Scanner scanner;
17     private ClubCoordinator clubCoordinator;
18
19     public ClubConsole() {
20         //
21         this.scanner = new Scanner(System.in);
22         this.clubCoordinator = new ClubCoordinator();
23     }
24
25     public void showMenu() {
26         //
27         int inputNumber = 0;
28
29         while (true) {
30             displayMainMenu();
31             inputNumber = selectMainMenu();
32
33             switch (inputNumber) {
34                 //
35                 case 1:
```

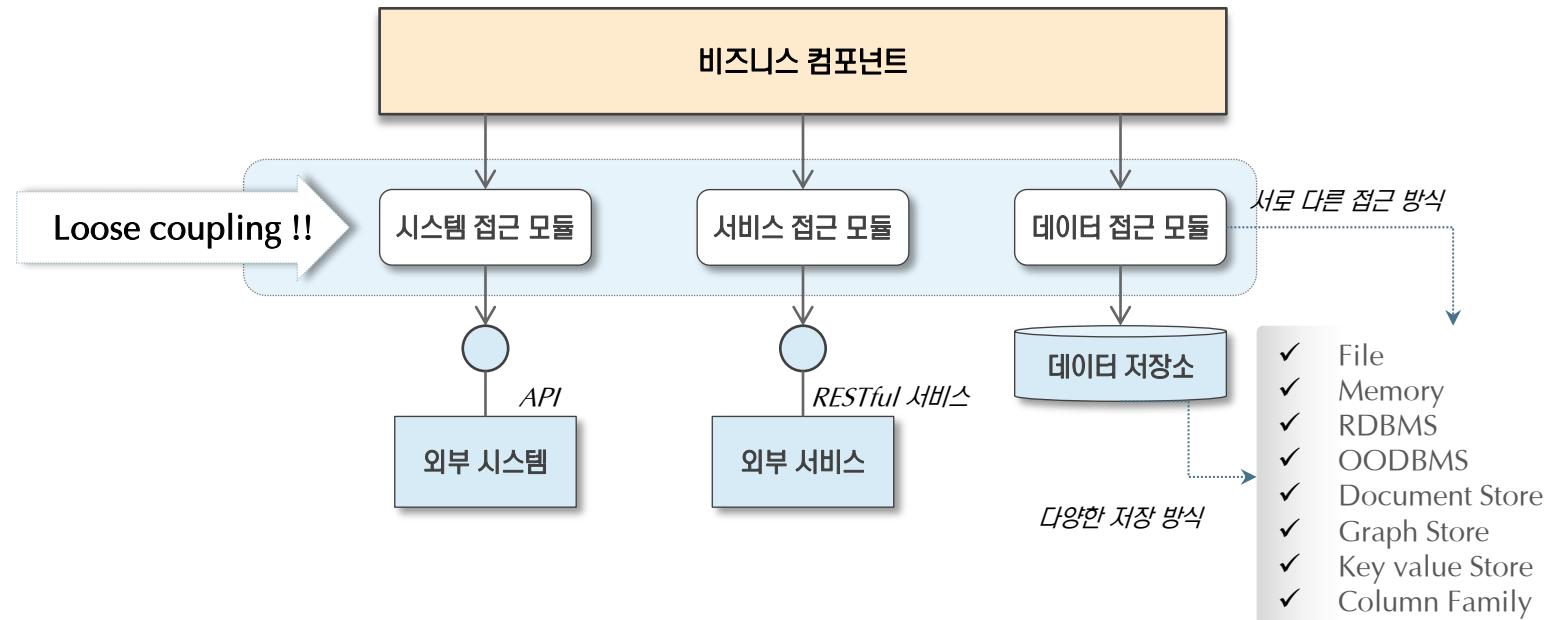
4.2 자원 접근(1/5) – 개요

- ✓ 애플리케이션 내부의 비즈니스 컴포넌트는 주어진 [업무/제어 등] 처리를 하기 위해 자원을 사용하거나 접근합니다.
- ✓ 컴포넌트가 사용하는 자원에는 데이터 저장소, 외부 시스템이나 서비스 등이 있습니다.
- ✓ 자원으로의 접근을 위해 자원의 특성을 이해하여야 하며, 자원이 요구하는 방식으로 접근해야 합니다.
- ✓ 아울러 자원은 변경 가능한(variable) 특성을 갖고 있으므로, 대응할 수 있는 구조로 설계해야 합니다.



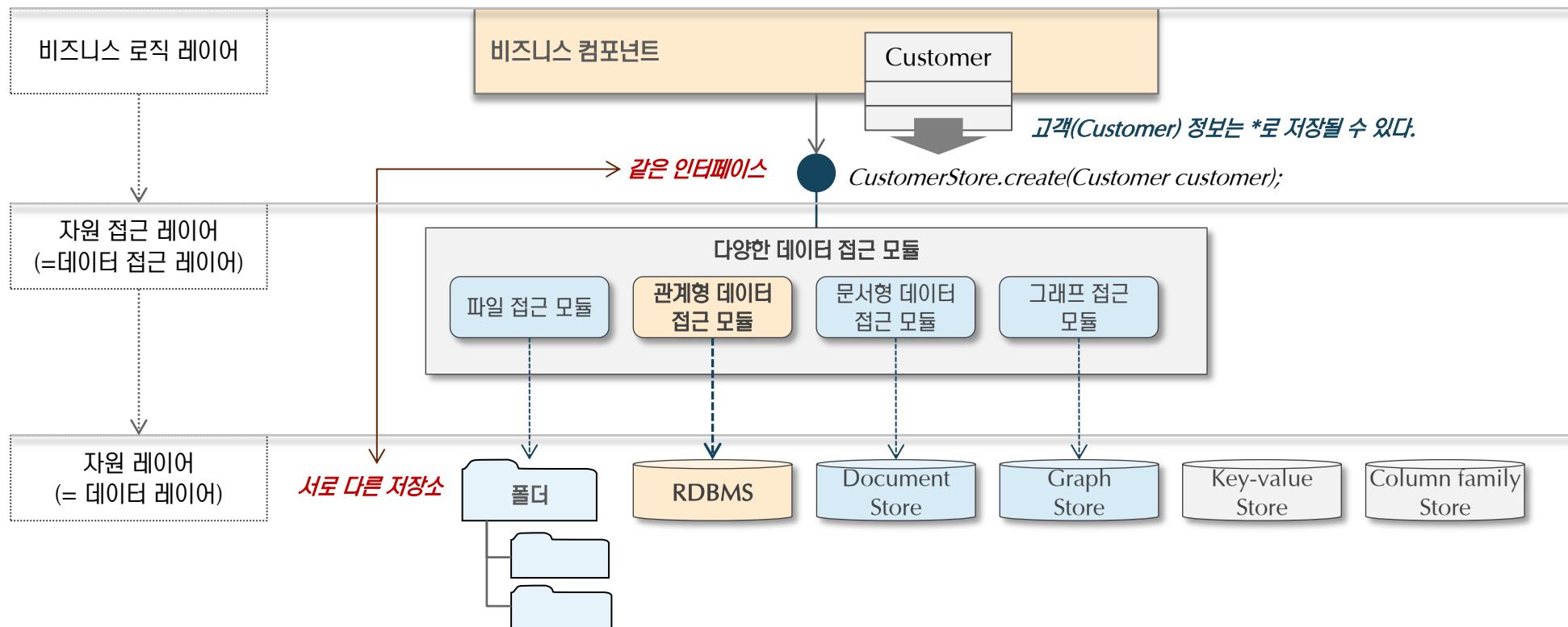
4.2 자원 접근(2/5) – 느슨한 결합

- ✓ 비즈니스 컴포넌트는 외부 자원을 사용하되, 외부 자원과의 관계를 느슨하게 유지해야 합니다.
- ✓ 비즈니스 컴포넌트 입장에서 [외부] 자원(resource)은 대체 가능한 모듈이어야 하므로 대체 가능하도록 설계해야 합니다.
- ✓ 시스템이나 서비스 자원은 외부로 인식하지만, 데이터는 내부 자원으로 인식하므로 "느슨한 결합" 방식으로 설계해야 한다는 설계자의 의무를 잊어버리는 경우가 많습니다.



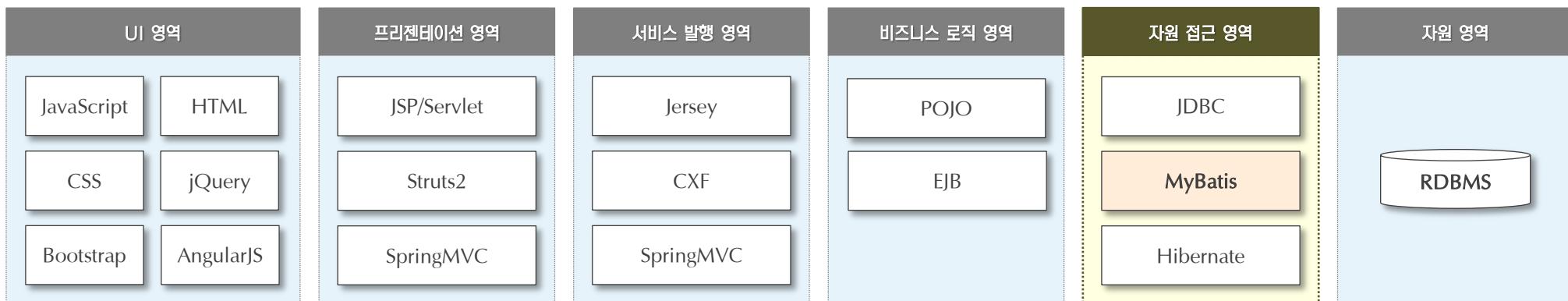
4.2 자원 접근(3/5) – 레이어

- ✓ 비즈니스 컴포넌트는 비즈니스 로직 레이어에, 자원 접근 모듈은 자원 접근 레이어에 놓여 있습니다.
- ✓ 이들 간의 관계를 생각할 때, 좌측에 표현한 레이어 관점에서 볼 수 있습니다.
- ✓ 어떤 객체의 정보(예, Customer)가 자원 접근 레이어를 통해서 저장될 때는 하나의 인터페이스를 타고 들어갑니다.
- ✓ 하지만, 실제로 그 데이터가 저장되는 경로는 자원 접근 레이어에서 어떤 모듈이 처리하는가에 따라 다릅니다.



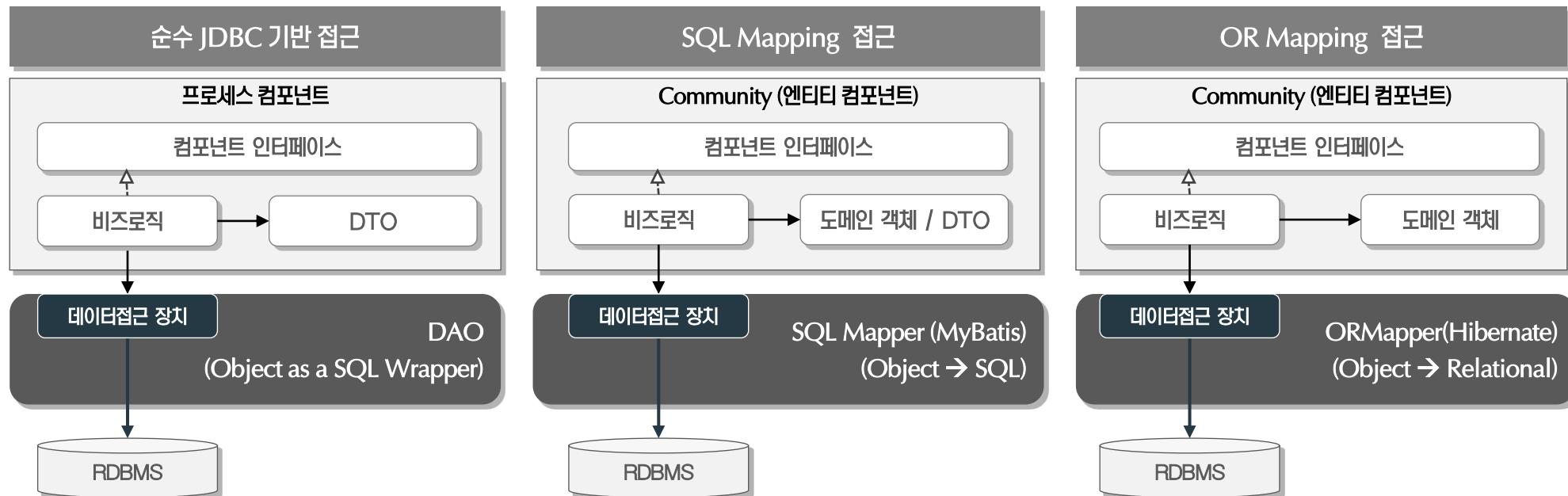
4.2 자원 접근(4/5) – 레이어 뷰

- ✓ MyBatis는 관계형 데이터베이스로 접근할 때 사용하는 오픈 소스 데이터 접근 프레임워크입니다.
- ✓ 데이터베이스 등 영속성을 위한 자원과 상호작용하는 영역을 자원 (Data Access) 접근 영역이라 하며, 이 영역에 적용하는 프레임워크가 데이터 접근 프레임워크 또는 Persistence(영속성) 프레임워크입니다.
- ✓ MyBatis는 관계형 데이터베이스 프로그래밍을 좀더 쉽고 효율적으로 하도록 도와주는 개발 프레임워크입니다.



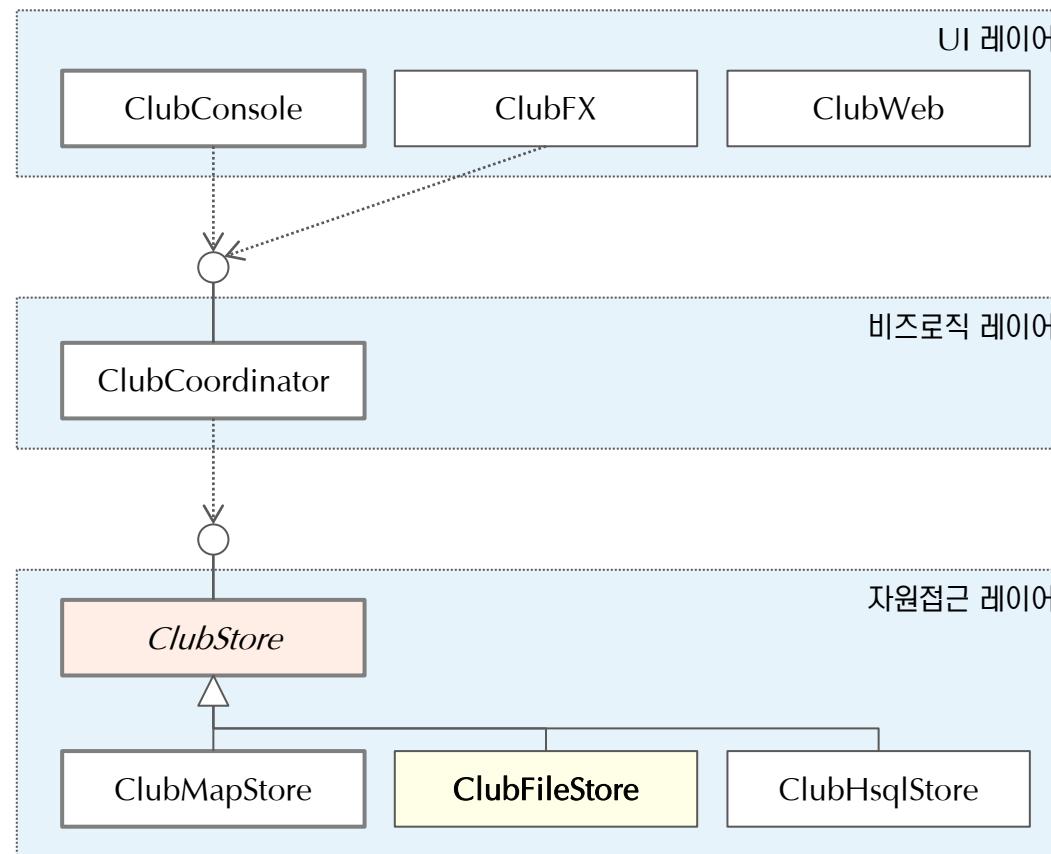
4.2 자원 접근(5/5) – 접근 방법

- ✓ 데이터 접근 프레임워크는 로직에서 DB연결 설정을 분리하여, 개발자가 비즈로직에 집중할 수 있도록 도와줍니다.
 - 순수 JDBC를 적용하면, DB자원 연결 및 사용에 관련된 코드가 메소드마다 중복됩니다.
 - 상황에 맞는 데이터 접근 프레임워크를 적용하면, 개발 편의성 뿐 아니라 성능 및 유지보수에도 큰 이점이 됩니다.
- ✓ 관계형 데이터 접근 프레임워크는 크게 SQL Mapping과 OR Mapping 접근 기반 프레임워크로 나눠집니다.
 - SQL Mapping 프레임워크는 자바 객체와 쿼리 결과를 매핑합니다.
 - OR Mapping 프레임워크는 자바 객체와 데이터베이스 릴레이션(테이블)을 매핑합니다.



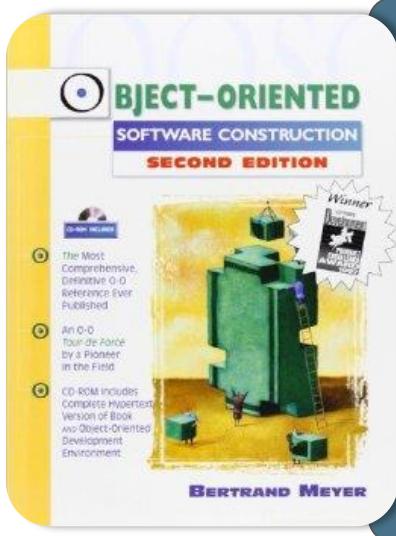
4.3 실습 4-2(1/5) – TinyClub 저장소 확장

- ✓ 실습 3-1 의 결과는 Club 정보를 메모리 맵에 저장하는 애플리케이션입니다. 지속성이 없습니다.
- ✓ 실습 3-2는 데이터 지속성을 제공하는 것이 목표입니다. 파일이나 DB를 사용할 수 있습니다.
- ✓ 실습 편의 상 DB는 사용하지 않습니다. 파일을 이용하여 지속성 제공 목표를 달성합니다.
- ✓ 이 과정에서 향후 확장 가능한 구조로 만들어야 합니다.



4.3 실습 4-2[2/5] – OCP 1

- ✓ 소프트웨어는 태생적으로 성장하는(growing) 존재입니다. 그래서 버전 개념을 내포하고 있습니다.
- ✓ 1988년 Bertrand Meyer는 자신의 책 “Object-Oriented Software Construction”에서 변경에 대응할 수 있는 설계 가이드를 제시했습니다.
- ✓ “Open for extension, but closed for modification” 개념이 SOLID의 두 번째 원칙 OCP입니다.



소프트웨어 요소(클래스, 모듈, 함수, 등)는 확장에 대해 열려 있어야 하고, 변경에 대해서는 닫혀 있어야 한다.

Software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification.

4.3 실습 4-2[3/5] – OCP 2

- ✓ 변경 할 수 있어야 하는데, 소스 코드나 라이브러리는 변경하지 말라는 의미는 매우 난해할 수 있습니다.
- ✓ Meyer는 이런 원칙을 가능하도록 하려면 “상속” 개념이 필요하다고 했습니다.
- ✓ OCP의 키워드는 바로 “추상화(abstraction)”이며, 인터페이스 기반 설계가 필요합니다.
- ✓ 추상 클래스를 통한 확장 보다는 인터페이스를 통한 확장이 바람직합니다.

[OPEN] 확장에 대해 열려 있고...

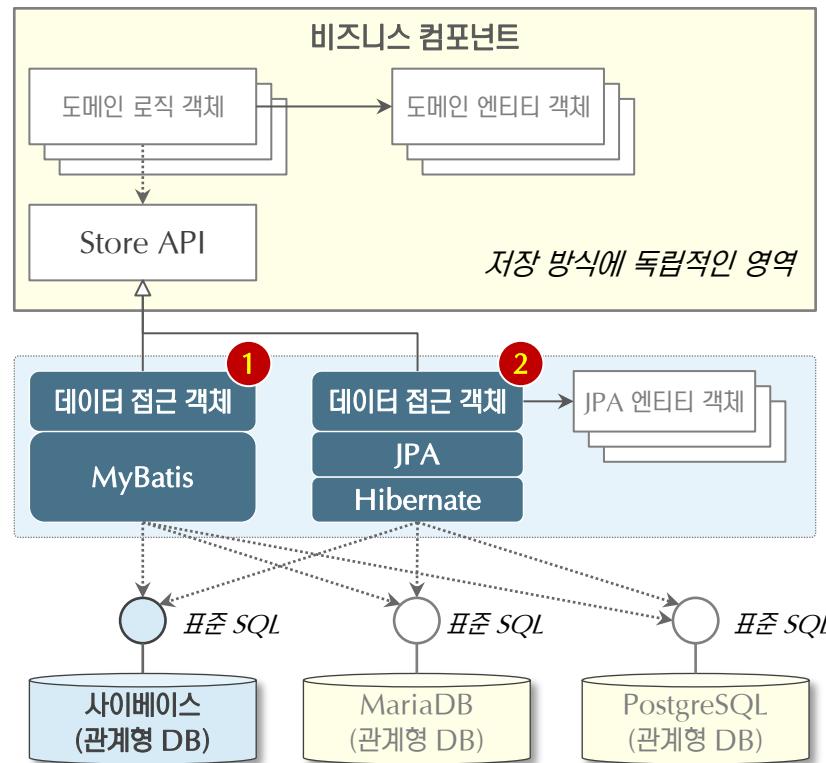
모듈의 기능은 확장 가능해야 한다. 새로운 변경 요구를 충족하기 위해서 모듈에 새로운 기능을 추가할 수 있어야 한다.

[CLOSE] 변경에 대해 닫혀 있어야...

모듈의 기능을 확장하는 과정에서 기존 기능을 잘 수행하고 있는 모듈의 소스 코드나 라이브러리를 변경해서는 안된다.

4.3 실습 4-2[4/5] – OCP 3

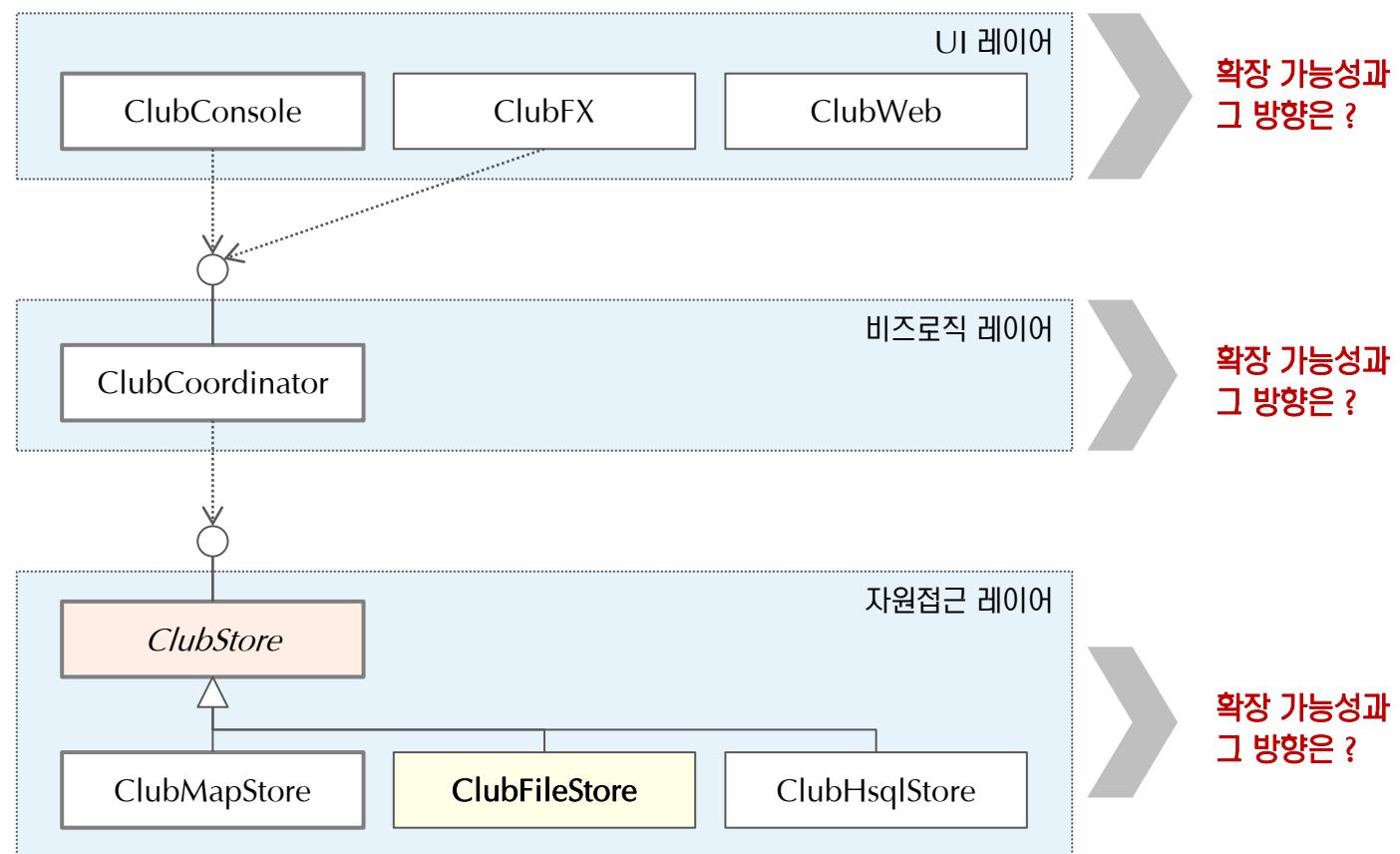
- ✓ 예를 들면, “나무소리” 사는 소셜보드 솔루션을 만들어서 판매하고 있었습니다.
- ✓ 솔루션은 고객의 요구에 맞추어서, 상당한 부분에 대해 customization 작업을 해야 합니다.
- ✓ 최근에 작업을 시작한 고객사에서는 데이터 접근 방식에 대한 변경을 요구했습니다.
- ✓ 나무소리 게시판 솔루션은 SQL 매팅 프레임워크인 MyBatis를 기본 프레임워크로 사용하였습니다.



[변경 내용] SQL 매팅 → OR 매팅 프레임워크로 데이터 접근 방식
변경, 하지만 다른 고객사에서는 여전히 SQL 매팅을 사용하고 있음.
기존 코드는 변경하지 말고, 새로운 기능을 추가하여야 함

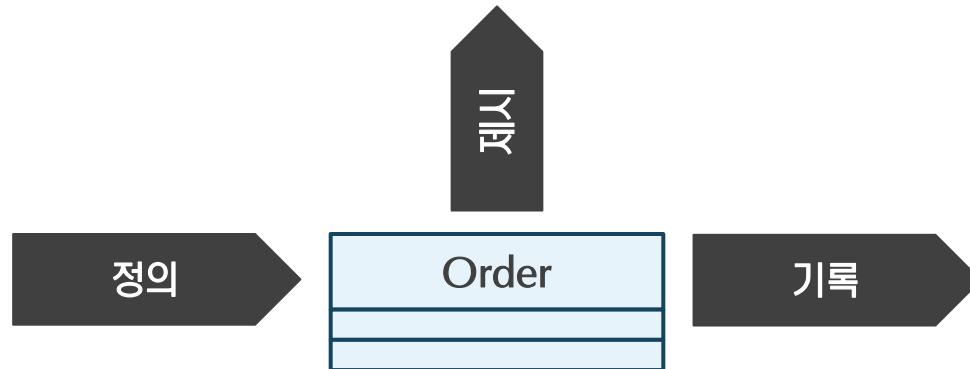
4.3 실습 4-2[5/5] – 토론

- ✓ 레이어의 근간을 이루는 개념은 무엇이며, 아래 세 개의 레이어로 충분한가?
- ✓ UI 확장은 어떤 것이 있으며, UI확장에 대응하기 위해서 어떤 구조가 필요합니까?
- ✓ 비즈니스 로직 레이어를 확장할 수 있는가? 있다면 어떻게 할 수 있고, 없다면 왜 확장이 불가능한가?
- ✓ 데이터가 애플리케이션의 자원(resource) 전부인가? 다른 자원은 없는가? 그런 자원으로의 확장은 어떻게 해야 하는가?



4.4 정보 시스템과 레이어(1/3)

- ✓ 아키텍트 관점에서 정보의 정의는 무엇일까요?
- ✓ 정보에 대한 충분한 개념, 영속성을 갖기 위한 기록, 유용성을 갖기 위한 제시, 세 가지를 생각할 수 있습니다.
- ✓ 이런 정보의 특성에 대한 이해는 정보 시스템을 설계하는데 도움이 됩니다.
- ✓ 정보 관리 시스템은 이 세 가지 특성을 기반으로 설계하고 그 결과 정보 시스템은 레이어드 특성을 갖습니다.

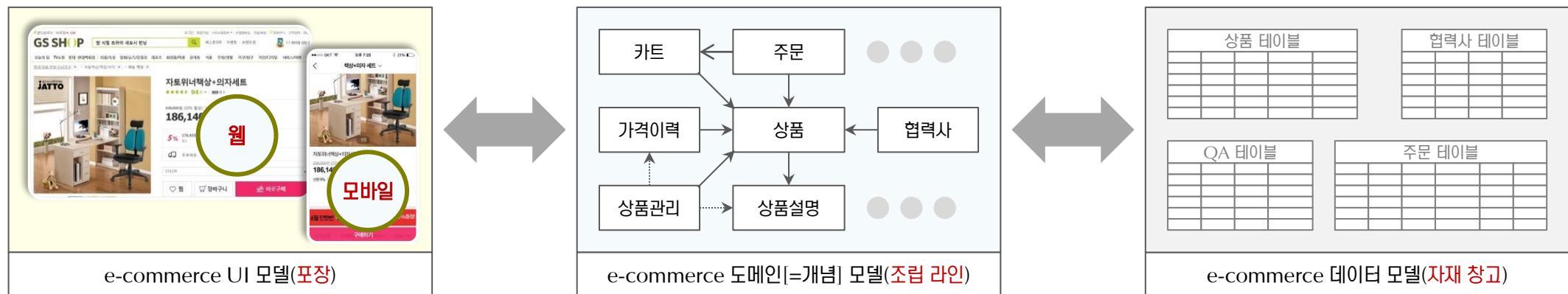


정보란 개념이 정의되어야 하고, 영속성을 가져야 하고, 필요할 때 제시할 수 있어야 합니다.

4.4 정보 시스템과 레이어(2/3)

- ✓ 정보 시스템을 구축할 때, 개념을 정의(도메인 모델링)하고 컴포넌트를 개발하고, 데이터와 UI 부분을 개발합니다.
- ✓ 하지만, 이러한 정보의 특성에 대해 서로 다른 생각을 가지고 시스템 구축에 접근하는 경우도 있습니다.
- ✓ 데이터를 중심으로 설계하는 경우도 있고(1995 ~ 2000, 정보 공학), UI를 중심으로 설계하는 경우도 있습니다.
- ✓ 업무가 복잡할수록 올바른 개념 정의가 중요하게 되었고, 그 결과 DDD(Domain Driven Design)가 주류가 되었습니다.

정보 시스템의 고민 1 : 상품, 주문 등의 개념을 세 가지 서로 다른 방식으로 표현을 해야 합니다.

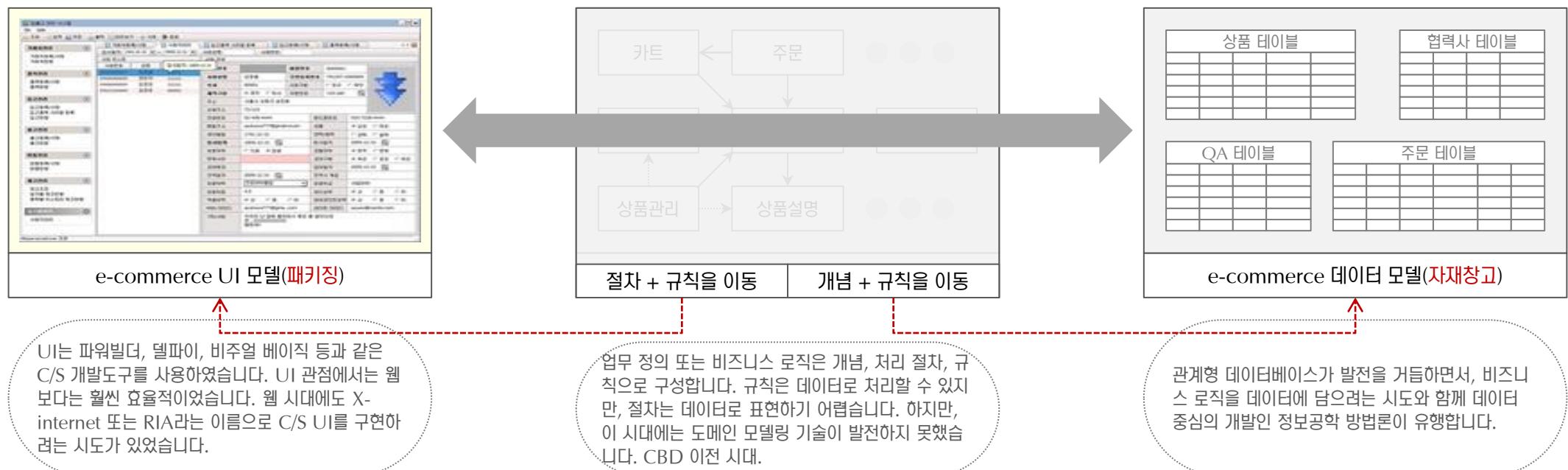


SW 설계 원칙: 관심사 분리(Separation of concern)

4.4 정보 시스템과 레이어(3/3)

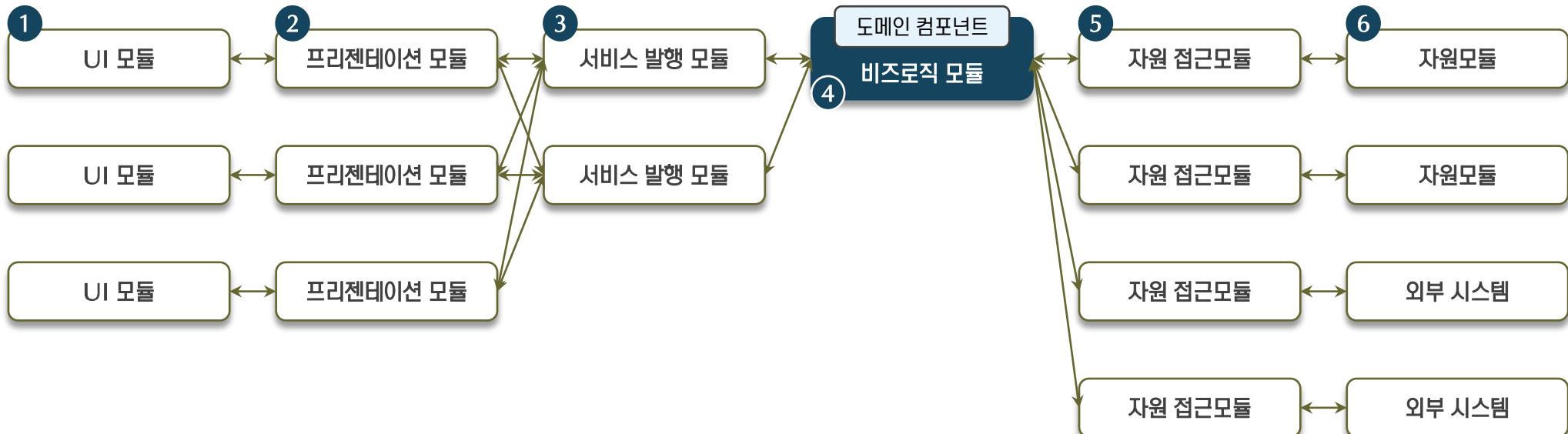
- ✓ 1990년대 중반부터 RDBMS가 널리 사용되면서 데이터의 중요성이 강조하는 개발 접근방법이 유행하였습니다.
- ✓ 그 시대에는 데이터를 중심으로 하는 개발 접근 방법인 정보 공학(Information Engineering) 방법이 주류가 되었습니다.
- ✓ 개발의 시작은 데이터 모델링으로 출발하고, 이후 UI 개발 후 연결하는 방식입니다.
- ✓ 아키텍처 관점에서는 2 티어 구조를 가지는 클라이언트-서버(사실상 DB 서버) 개발 방식입니다.

복잡한 정보를 담을 수 있지만, 잦은 변경을 처리하기에는 너무나 많은 노력이 필요하여, 비즈니스 변화를 제 때에 지원하지 못하면서 한계를 드러냅니다. ← 빅뱅 프로젝트의 원인



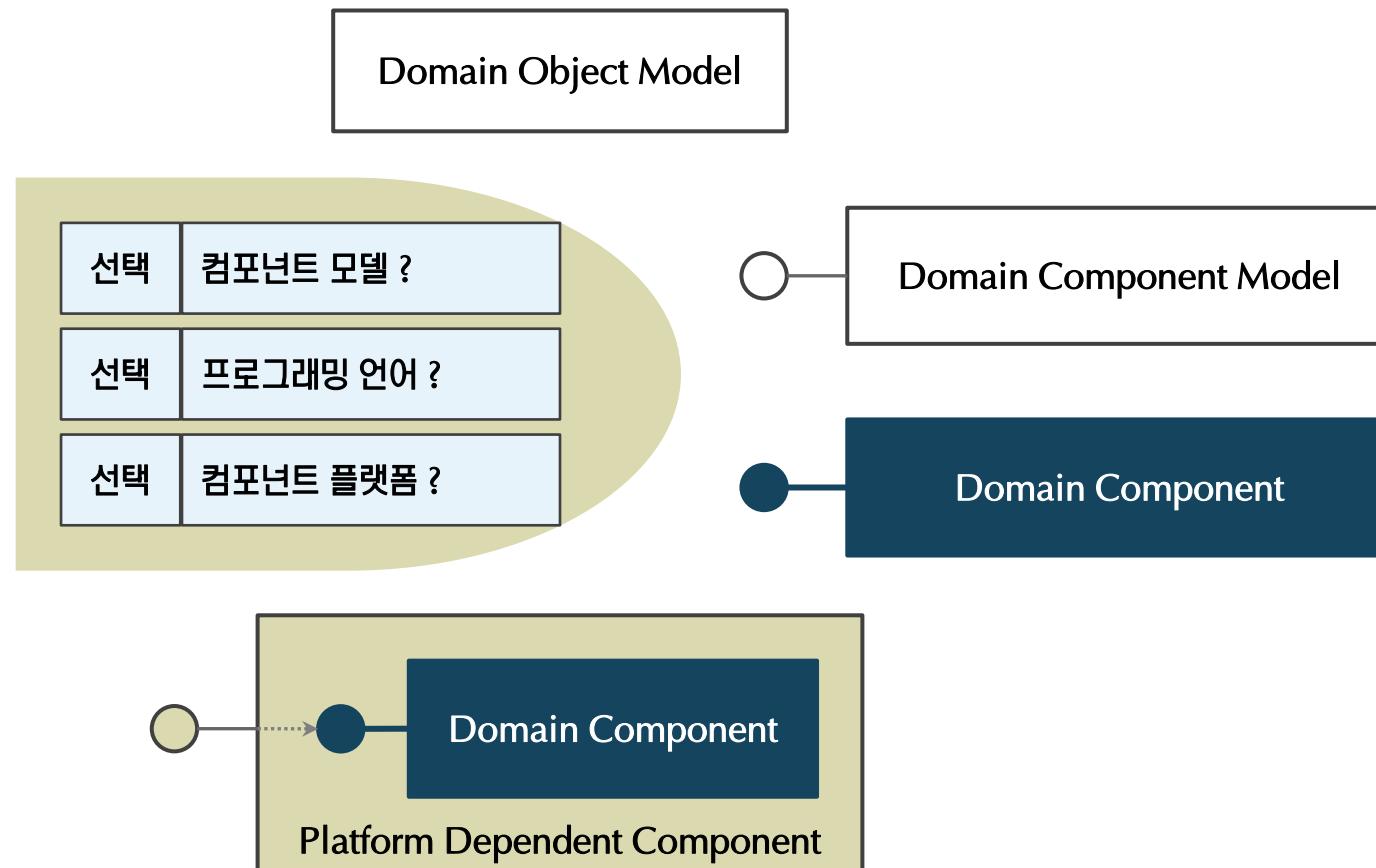
4.5 레이어드(1/14): 개요

- ✓ 잘 설계된 애플리케이션은 Loose coupling, High cohesion 원칙을 잘 지켰을 겁니다.
- ✓ 서로 다른 시스템 구성 요소인 도메인과 기술이 잘 분리되었을 겁니다. 분리와 결합이 자유롭습니다.
- ✓ 기술 요소의 변화는 레이어를 기준으로 자유로우며, 도메인은 기술의 변화에 영향을 받지 않습니다.
- ✓ 도메인 컴포넌트는 각 레이어 기술 요소로부터 중립적이며, 의존성을 갖지 않습니다.



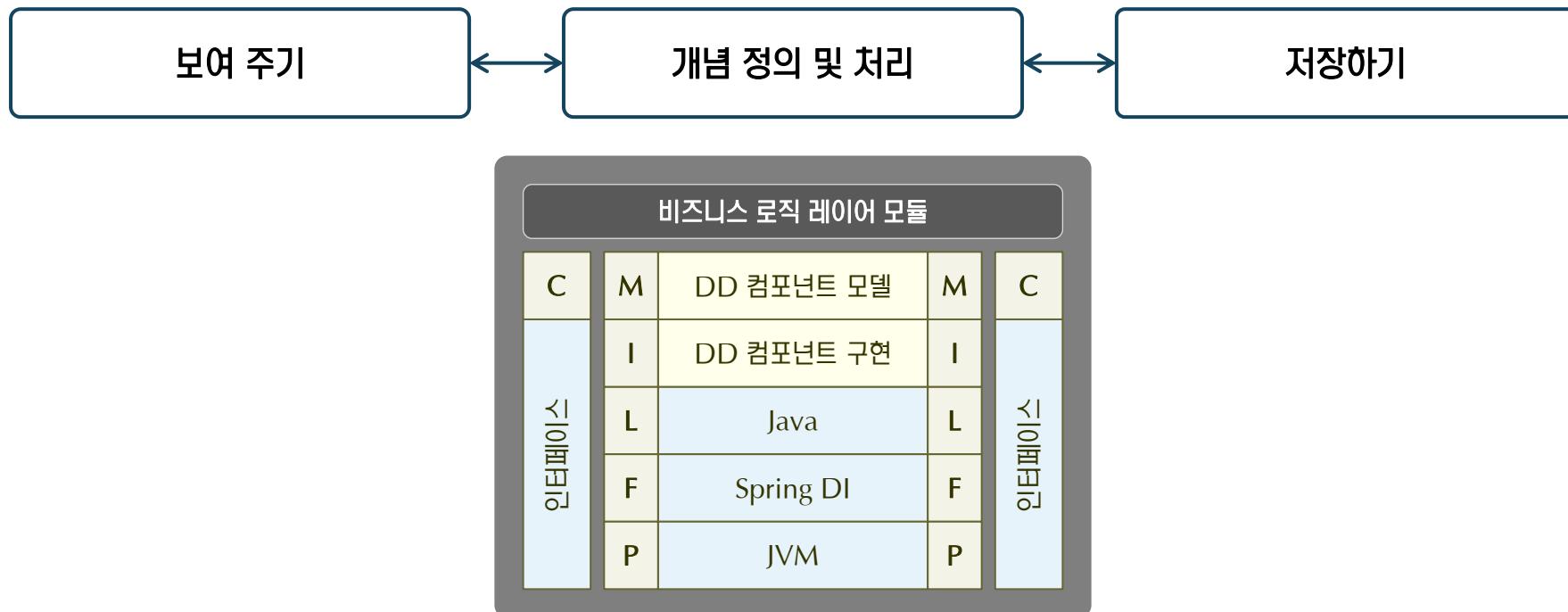
4.5 레이어드(2/14): 컴포넌트 구조 설계

- ✓ 도메인 객체 모델을 담을 컴포넌트 모델을 어떤 구조로 할 것인지 선택해야 합니다.
- ✓ 목표 시스템이 어느 수준의 기술 독립성을 가져야 하는지, 팀원의 기술 수준 등에 따라 컴포넌트 모델을 결정합니다.
- ✓ 컴포넌트를 구현할 언어와 컴포넌트 플랫폼을 선택해야 합니다.
- ✓ 도메인 모델링과 도메인 컴포넌트 구현은 분석가가 할 수도 있습니다.



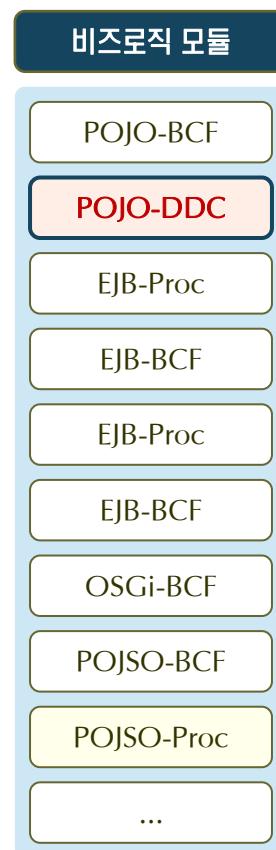
4.5 레이어드(3/14): 비즈니스 로직 레이어 모듈 1

- ✓ 도메인 컴포넌트를 감싸고 있는 플랫폼 종속적인 컴포넌트는 비즈니스 로직이 됩니다.
- ✓ 각 레이어 모듈에서 선택 가능한 요소는 대략 여섯 가지 정도입니다.
- ✓ 플랫폼, 프레임워크/라이브러리, 표현 언어, 사용자 정의 모델, 모델 구현체, 외부로의 연결 등은 모두 선택 사항입니다.



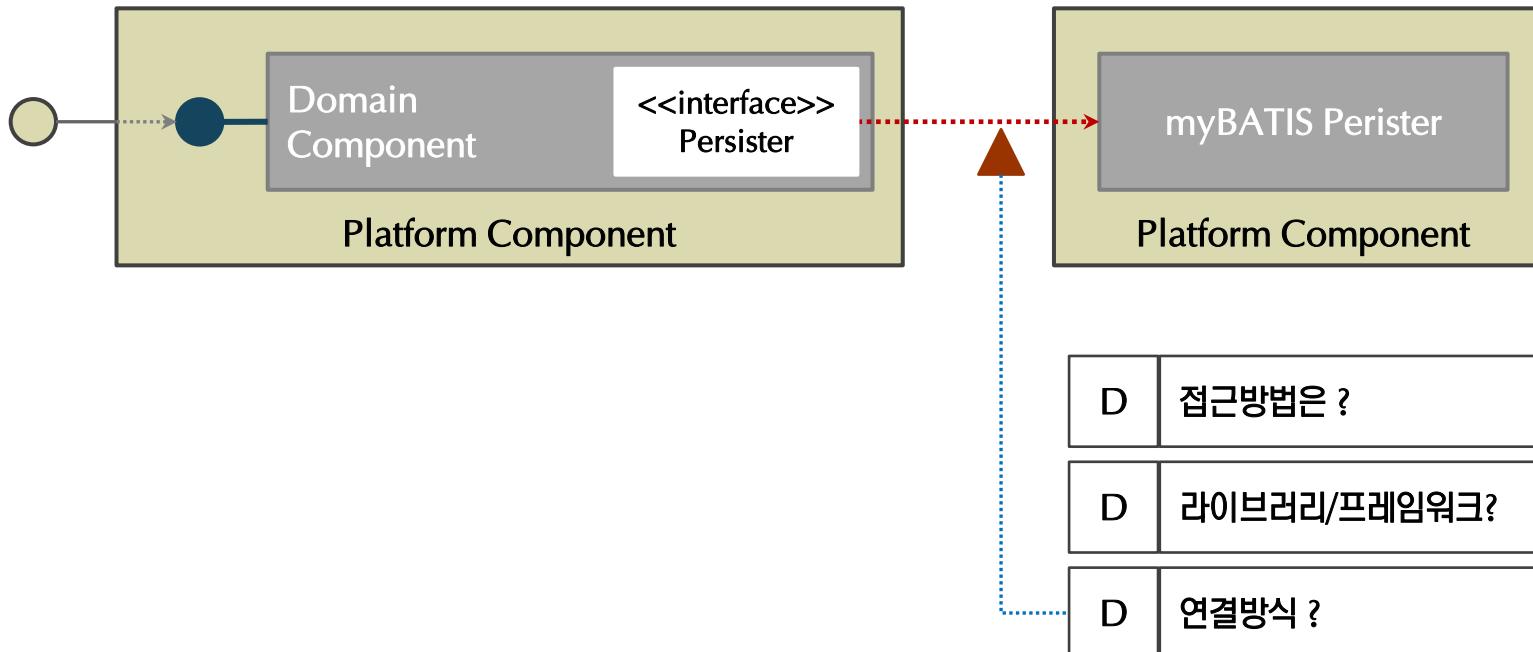
4.5 레이어드(4/14): 비즈니스 로직 레이어 모듈 2

- ✓ 여섯 가지 선택 요소는 결국 레이어 모듈의 특징을 결정하게 합니다.
- ✓ 비즈니스 로직 레이어를 구성하는 모듈은 컴포넌트 모델과 플랫폼 선택에 따라 다음과 같이 다양할 수 있습니다.
- ✓ 우리는 POJO 기반의 Domain [Driven] Component를 선택했습니다.



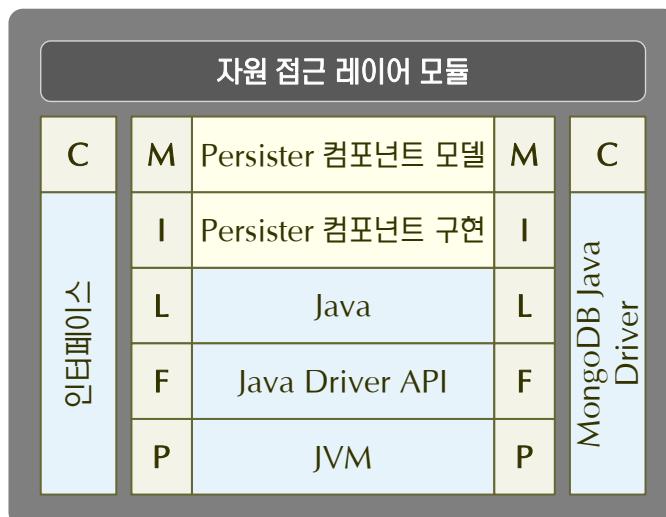
4.5 레이어드(5/14): 자원접근 레이어 모듈 1

- ✓ 다음은 컴포넌트로부터 자원에 접근하는 구조를 설계합니다.
- ✓ 자원 접근 영역이 완전히 투명해야 하는가 아니면 어느 정도의 의존성을 허용하는가에 대해 고민을 합니다.
- ✓ 어떤 경우든 자원 접근 모듈을 대체 가능하도록 설계하는 것이 필요합니다.



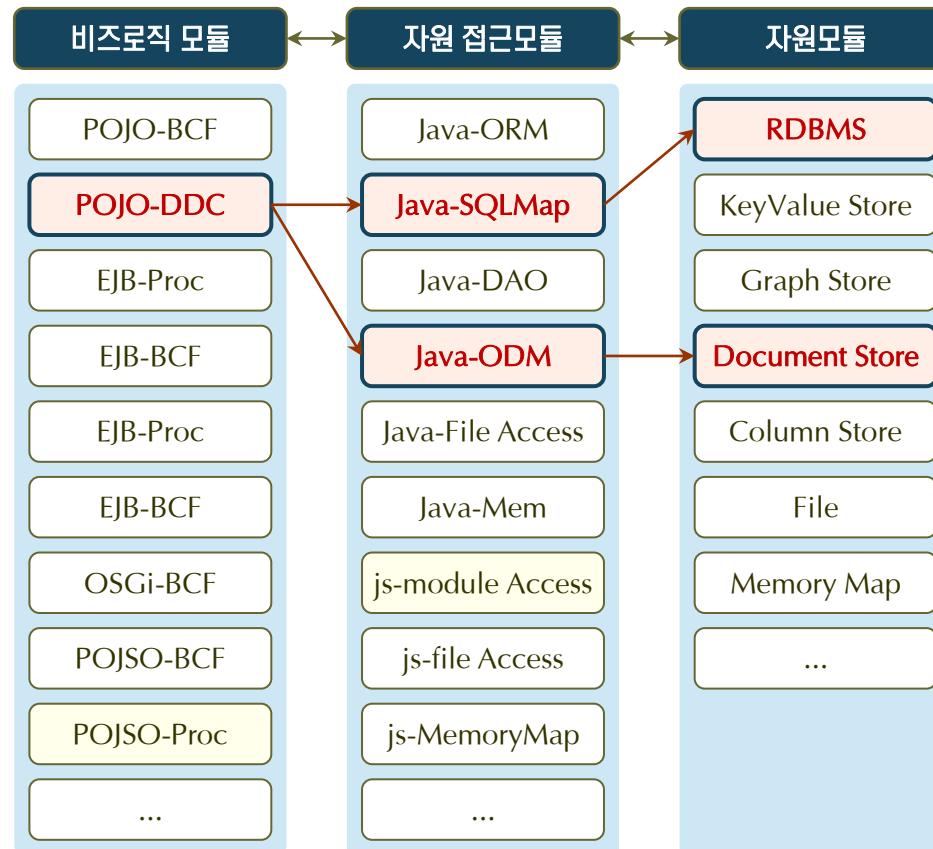
4.5 레이어드(6/14): 자원접근 레이어 모듈 2

- ✓ 고객의 요구와 시스템 환경 등을 고려하여 두 가지 접근 방법을 선택합니다.
- ✓ 읽기 쓰기는 오라클 DB로 읽기 전용은 MongoDB로 접근하도록 설계를 했습니다.
- ✓ 이 모듈은 동일한 Persister 인터페이스를 구현하므로, 필요할 경우 서로 대체할 수 있습니다.



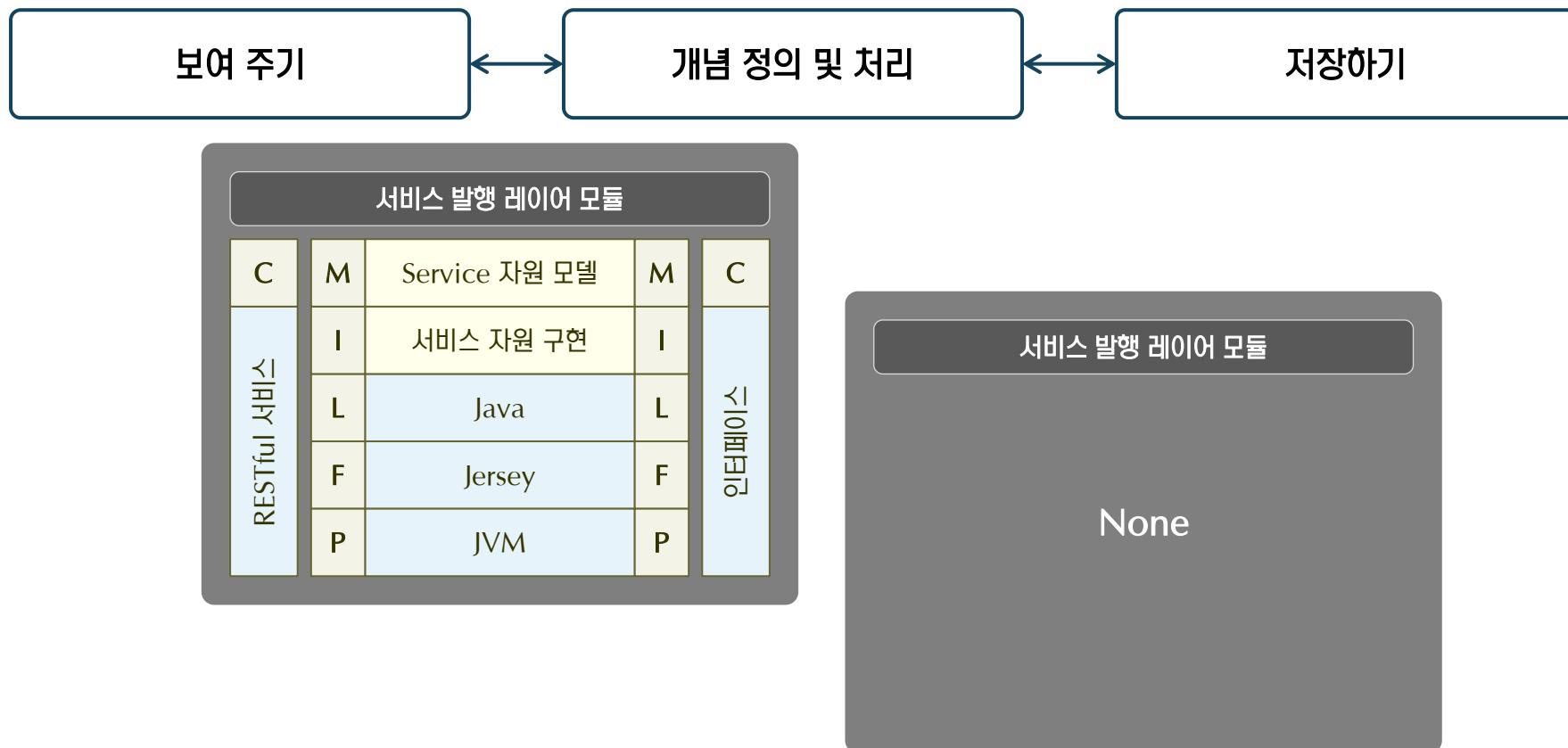
4.5 레이어드(7/14): 자원접근 레이어 모듈 3

- ✓ 자원 모듈과 자원 접근은 매우 밀접한 관계가 있으므로 함께 선택합니다.
- ✓ 선택 가능한 자원 접근 방법 중에 두 가지를 선택했습니다.
- ✓ 트랜잭션 등에 대한 고민은 틀을 완성할 때까지는 뒤로 미룹니다.



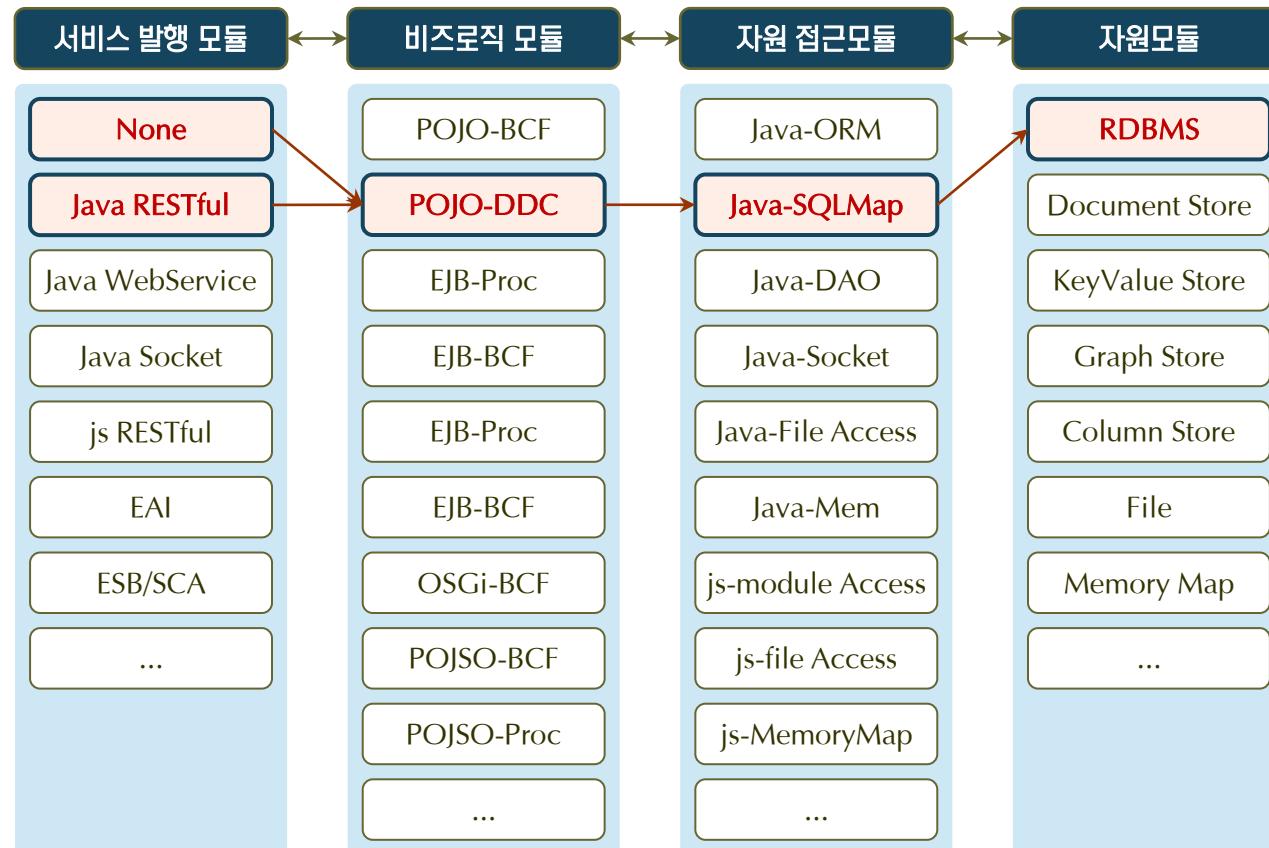
4.5 레이어드(8/14): 서비스 발행 레이어 모듈 1

- ✓ 정의한 정보나 처리된 결과 정보를 외부로 내보내는 레이어입니다.
- ✓ UI나 프리젠테이션 레이어로부터 직접 플랫폼 커포넌트에 접근한다면 레이어 모듈은 “None”입니다.
- ✓ 웹 페이지의 Javascript 모듈에서 REST 서비스를 이용하여 데이터를 가져가는 경로도 준비했습니다.



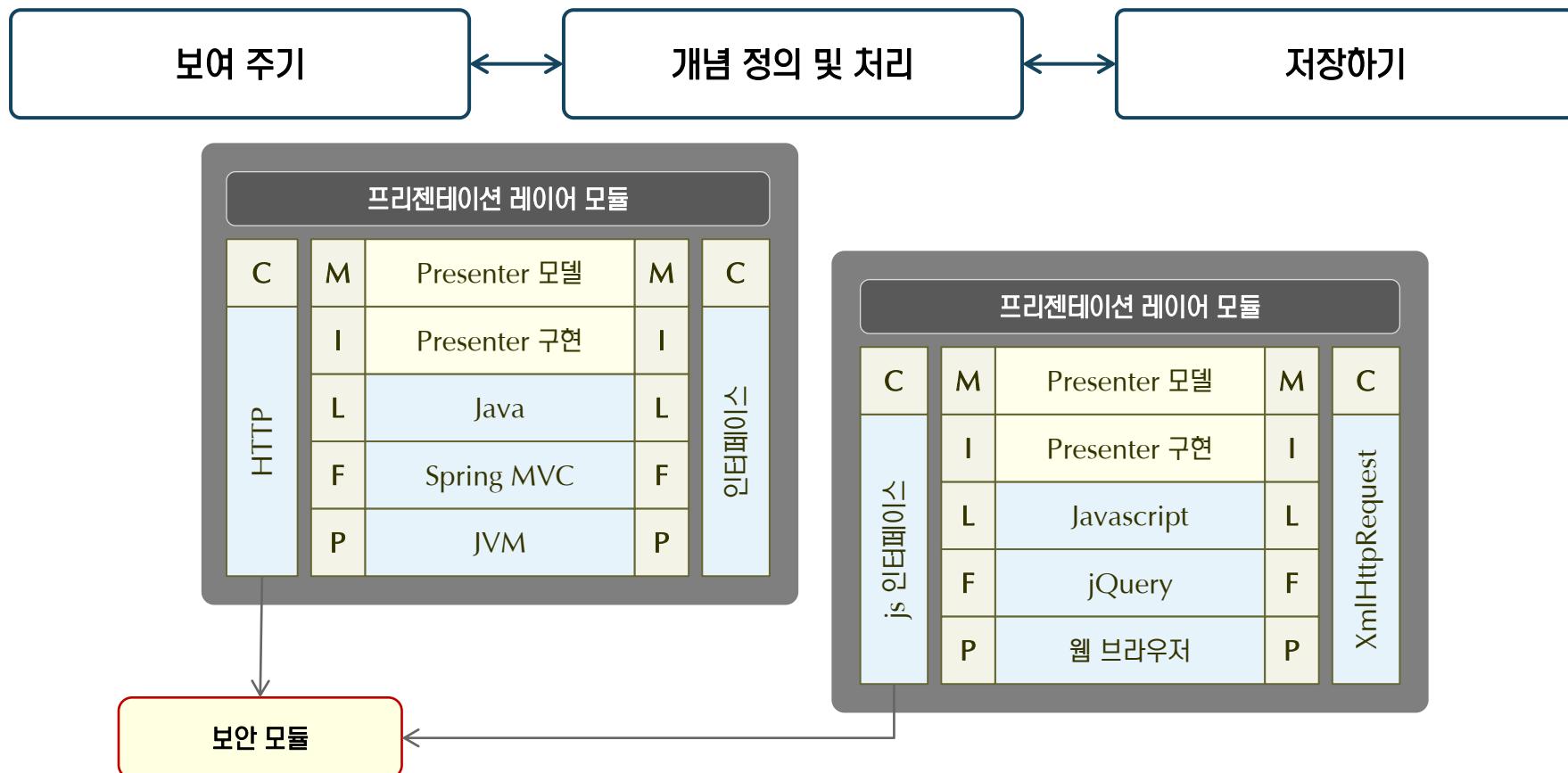
4.5 레이어드(9/14): 서비스 발행 레이어 모듈 2

- ✓ 프리젠테이션 모듈에서 컴포넌트로 직접 접근 경로와 RESTful 서비스로 접근하는 경로를 선택했습니다.
- ✓ 그 외 다양한 서비스 발행 방법이 있습니다.
- ✓ 요즘은 Node.js 기반의 Javascript RESTful 서비스 발행도 많이 선택되고 있습니다.



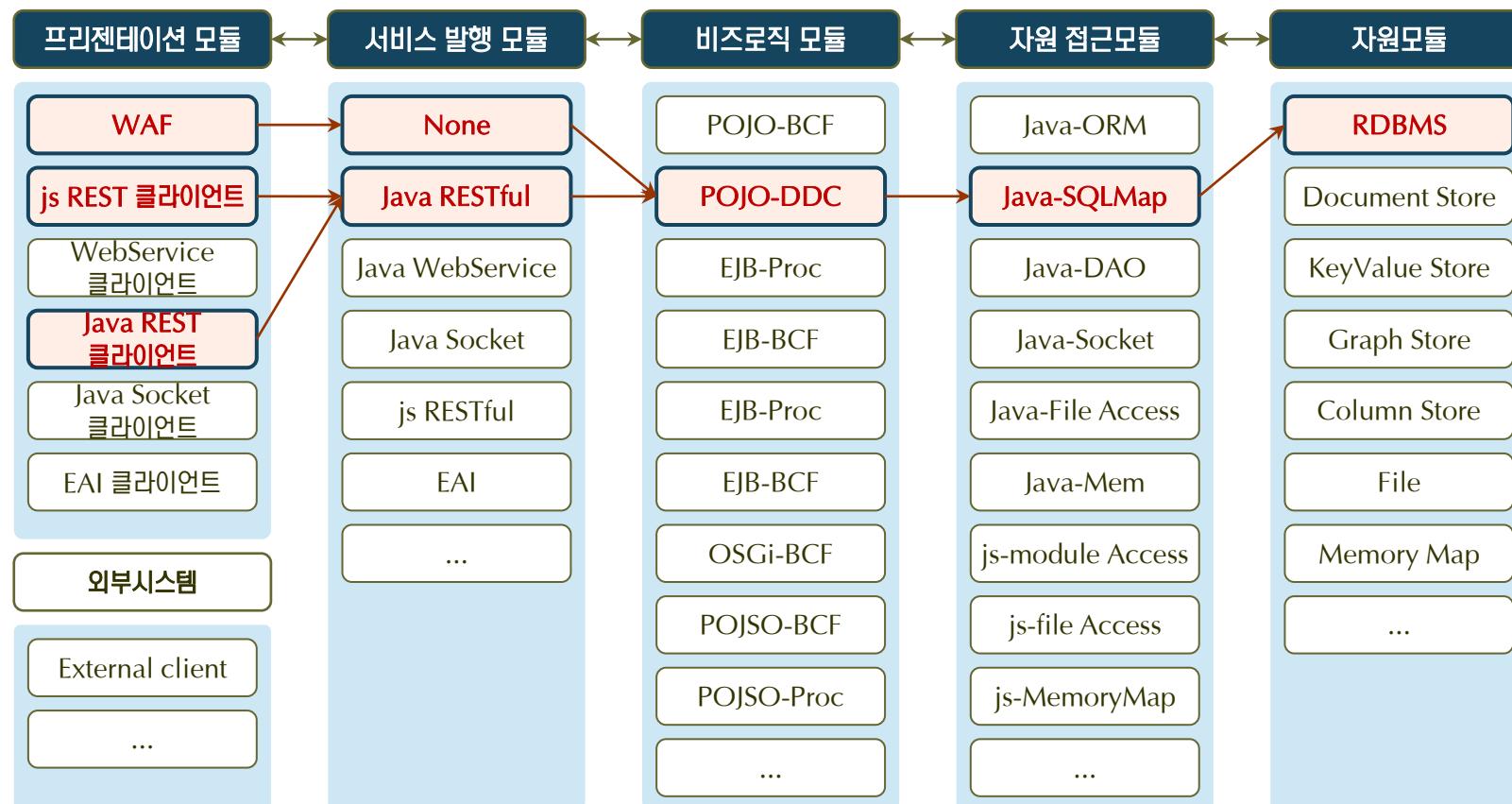
4.5 레이어드(10/14): 프리젠테이션 레이어 모듈 1

- ✓ 프리젠테이션 모듈은 두 가지를 선택하였습니다.
- ✓ Java와 Web Application Framework 위의 Presenter 모듈이 페이지 단위 UI 요청 처리를 합니다.
- ✓ 웹 브라우저에 내려가 있는 jQuery 모듈이 서버와 Ajax 통신을 하여 데이터를 가져 옵니다.



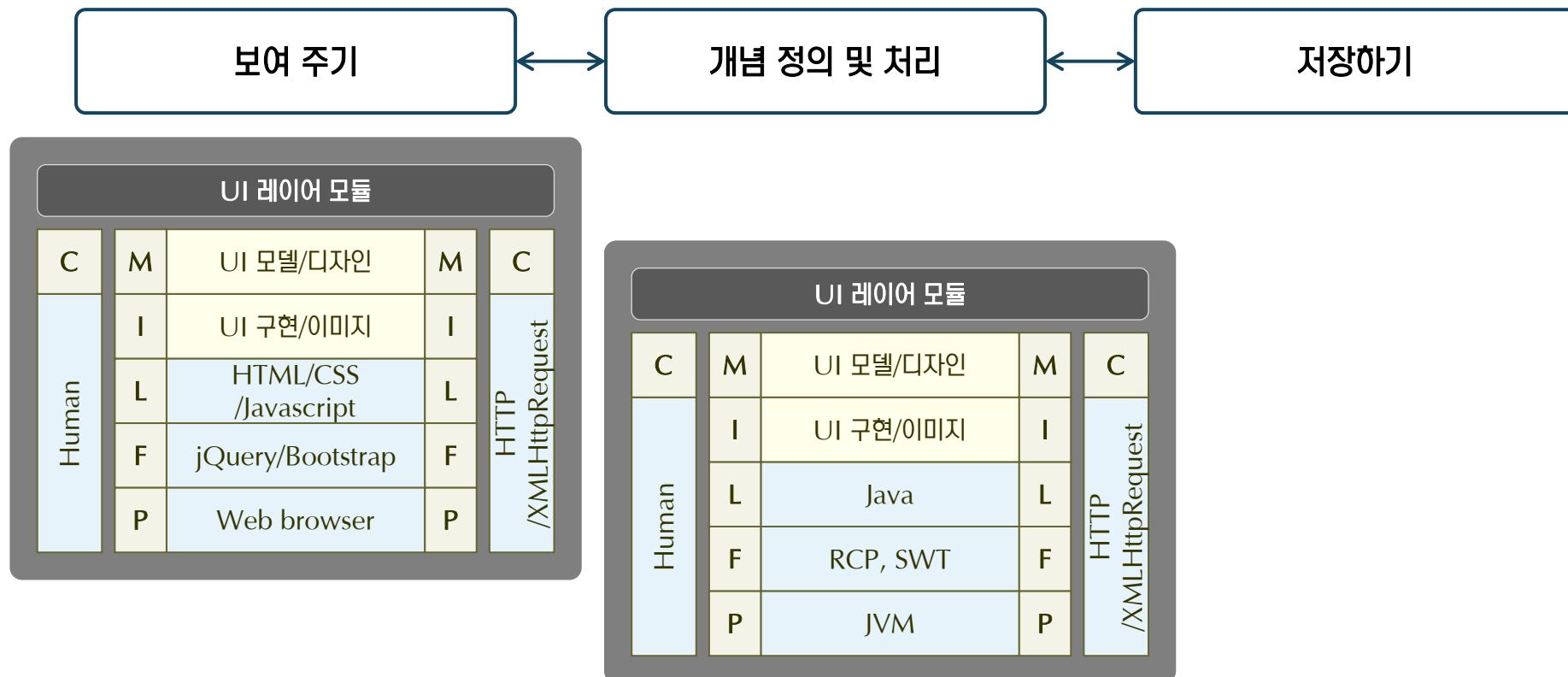
4.5 레이어드(11/14): 프리젠테이션 레이어 모듈 2

- ✓ 프리젠테이션 레이어는 UI 레이어와의 구분이 어려운 레이어입니다.
- ✓ 프리젠테이션 레이어에서는 두 개의 서로 다른 경로를 선택했습니다.
- ✓ WAF는 서버 사이드에서, Javascript RESTful 클라이언트는 웹 브라우저에서 동작합니다.



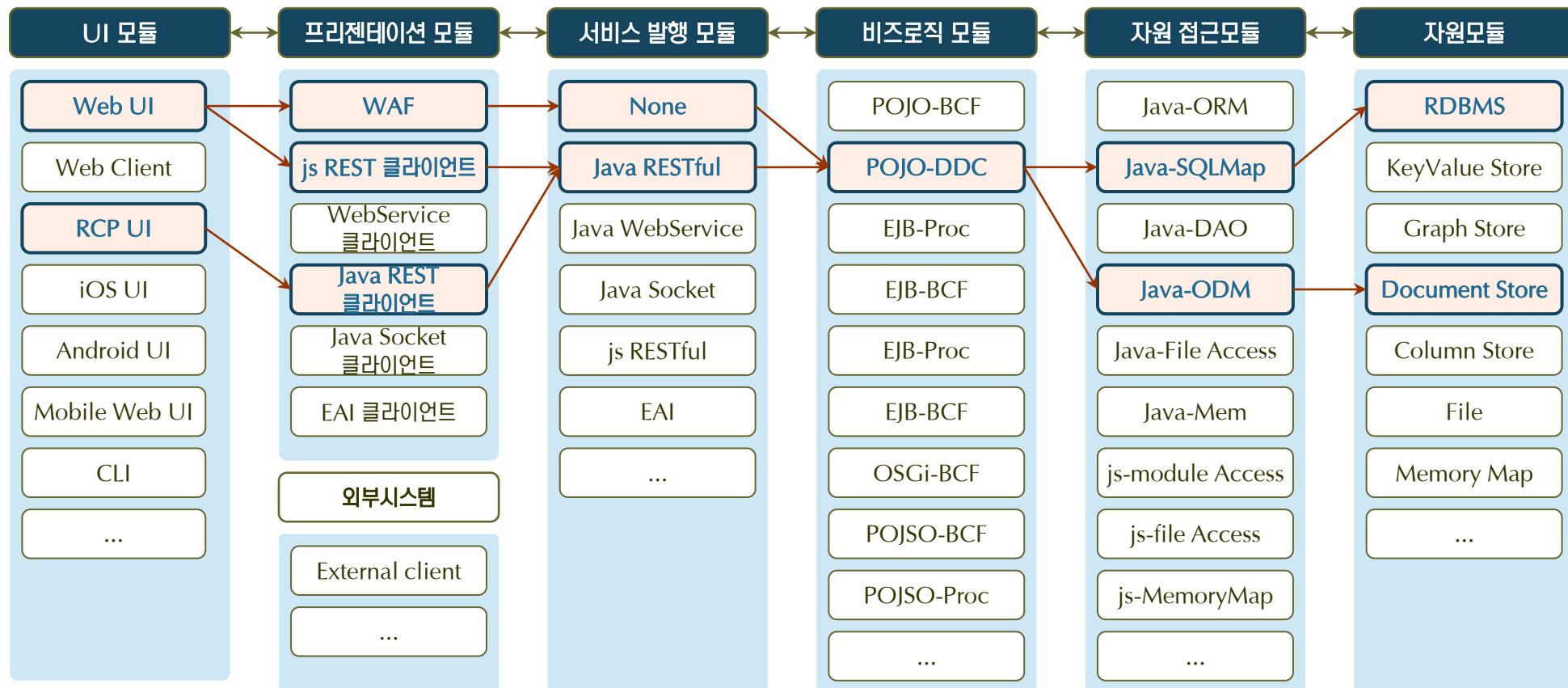
4.5 레이어드(12/14): UI 레이어 모듈 1

- ✓ UI 레이어를 위한 플랫폼으로 웹 브라우저와 JVM을 선택했습니다.
- ✓ 하나는 전형적인 웹 아키텍처에 AJAX 기능 보완한 구조를 가지고 있습니다.
- ✓ 또 다른 하나는 관리자를 위한 모니터링 기능을 제공하기 위해 eclipse RCP를 선택하였습니다.



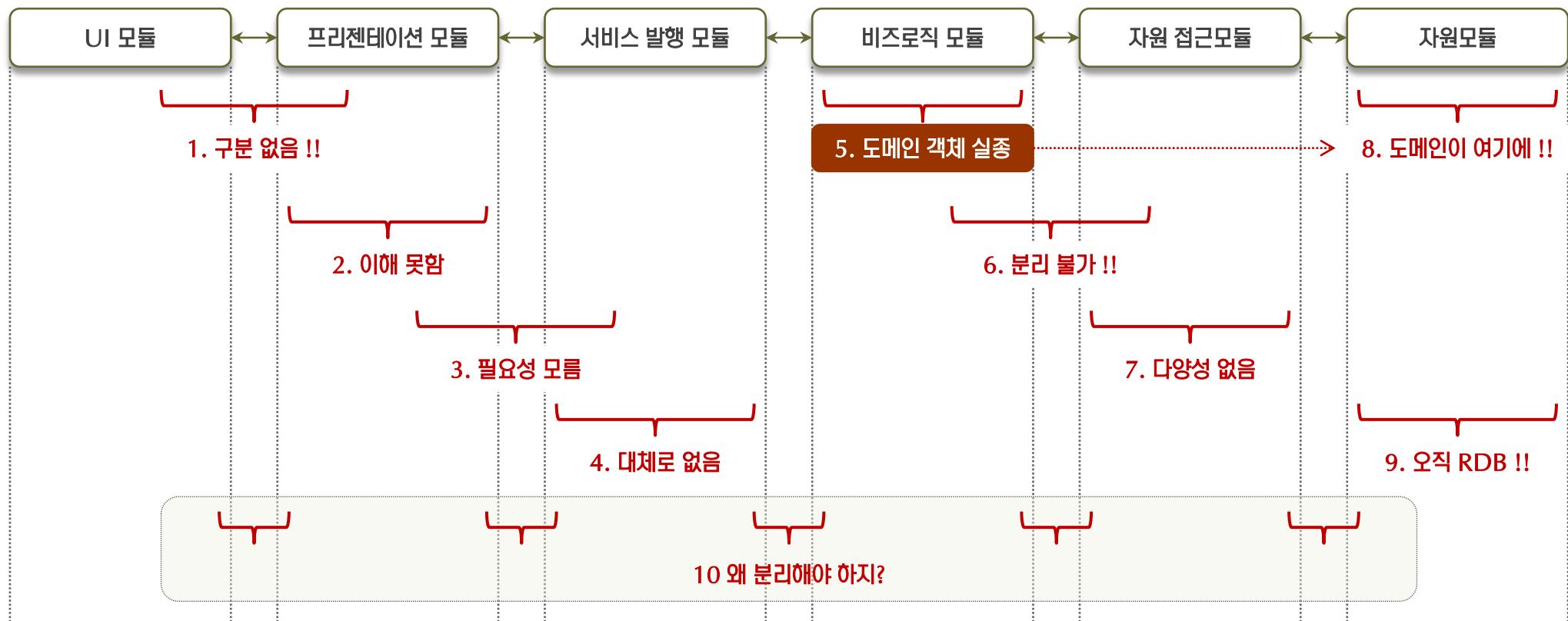
4.5 레이어드(13/14): UI 레이어 모듈 2

- ✓ 다양한 UI 중에 두 가지를 선택하였습니다. 일반 사용자용 웹 UI와 관리자용 RCP UI입니다.
- ✓ 이제 여섯 개의 레이어의 모듈을 모두 [설계 | 선택] 했습니다.
- ✓ 서비스나 프리젠테이션 레이어에서 도메인 컴포넌트의 인터페이스를 사용할 것인가에 대한 선택이 남아 있습니다.
- ✓ 각 레이어 진입점(connector)에 필요한 장치들에 대한 선택이 남아 있습니다.



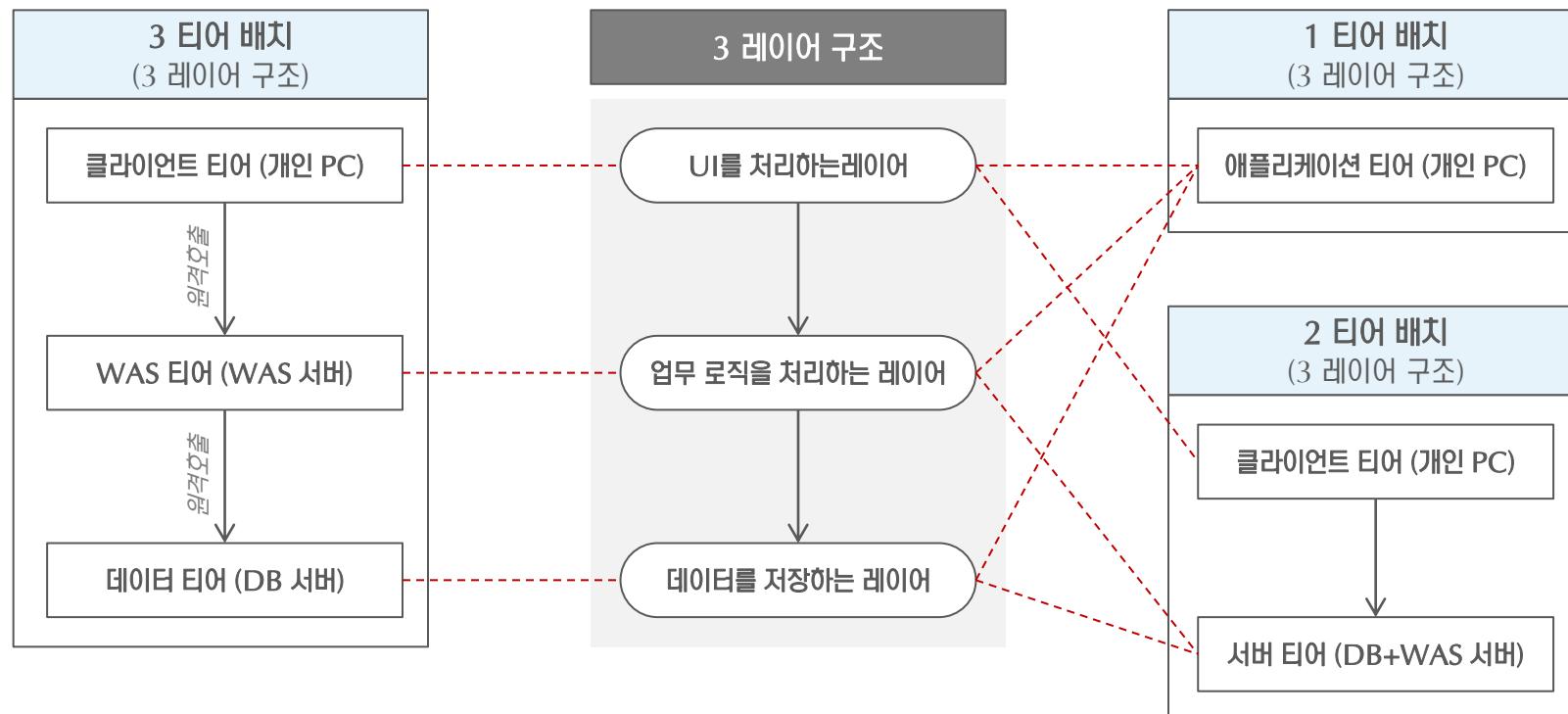
4.5 레이어드(14/14): 아키텍처 설계 회고

- ✓ 하지만, 우리는 레이어를 잘 분리해서 멋지게 설계한 시스템을 본적이 별로 없습니다.
- ✓ 체계적인 설계 역량이 부족하거나, 시간이 부족하거나, 관심이 부족하기 때문이겠죠.
- ✓ 아무렇게나 지어도 사람이 들어가서 살 수 있듯이, 아무렇게 개발해도 실행되기 때문입니다.
- ✓ 우리는 아키텍처 설계에 대한 고민을 많이 하지 않았으며, 그 결과로 유연성 없는 불량 시스템을 양산하고 있습니다.



4.6 레이어와 티어(1/4)

- ✓ 티어는 레이어와 비슷하면서도 다른 개념이므로 많은 사람들이 혼용해서 사용합니다.
- ✓ 레이어는 시스템의 논리 구조를 표현하고, 티어는 물리적인 서버를 표현합니다.
- ✓ 3레이어 구조로 설계한 시스템을 3티어, 2티어, 1티어로 배치(deployment)할 수 있습니다.
- ✓ 레이어 구성에 따라 레이어와 티어와의 관계를 설정할 수 있습니다. 아래 3 레이어 설계의 예를 살펴봅니다.



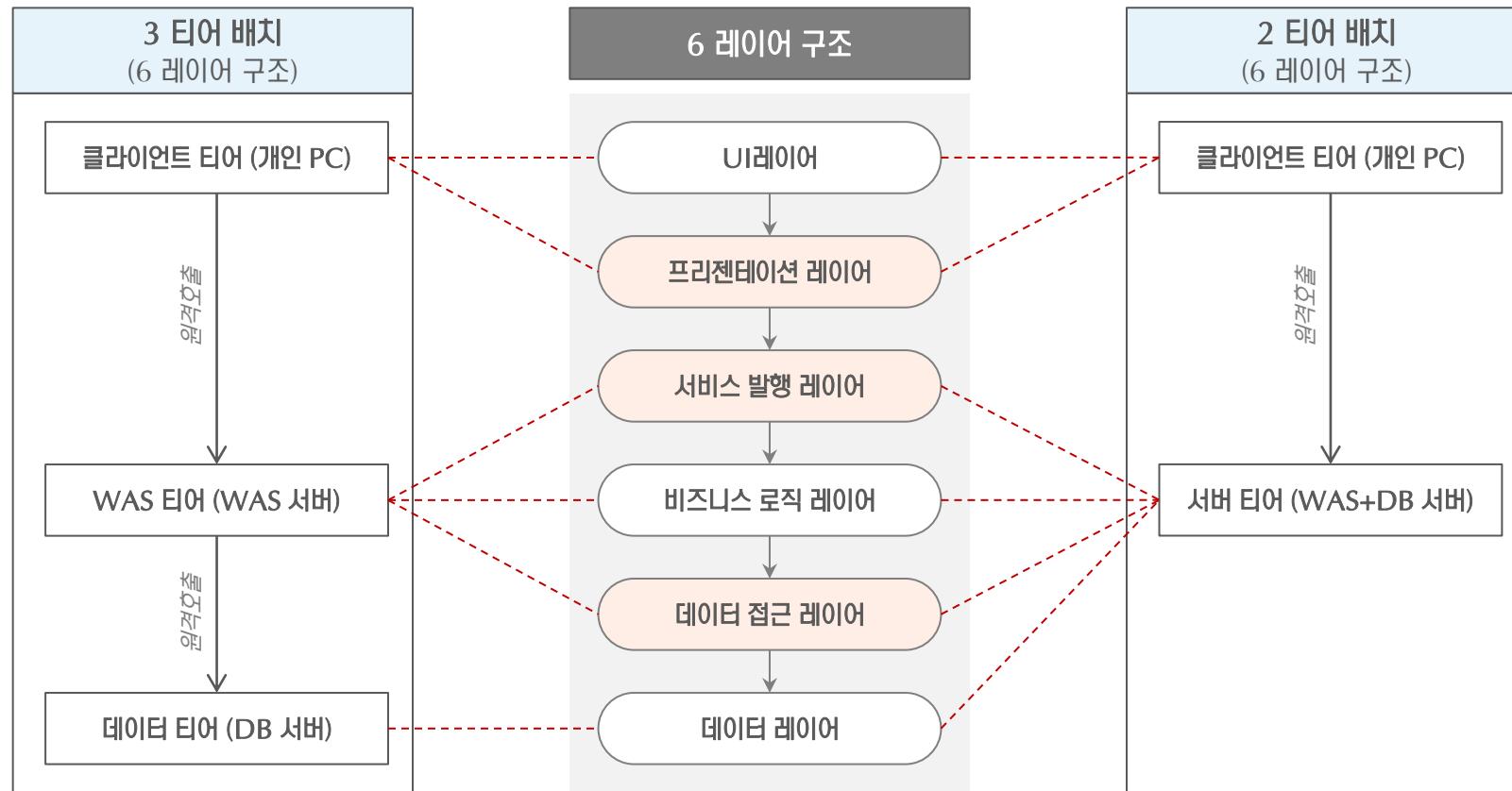
4.6 레이어와 티어(2/4)

- ✓ 레이어를 나눌 때, “서로 다른 작업은 서로 다른 레이어에서 한다.”는 원칙으로 세 개의 레이어를 나누었습니다.
- ✓ 레이어를 나누는 또 다른 기준은 “인접 레이어의 변화 충격을 흡수하는 레이어를 둔다.”입니다.
- ✓ 데이터 저장 방식이 변경될 때, 비즈니스 로직 대신 그 변화 충격을 흡수하는 “데이터 접근 레이어”가 필요합니다.
- ✓ 다양한 서비스 제공(웹 서비스 기반, REST 기반, TCP 기반, 등)에 따른 변화는 “서비스 발행 레이어”에서 소화합니다.



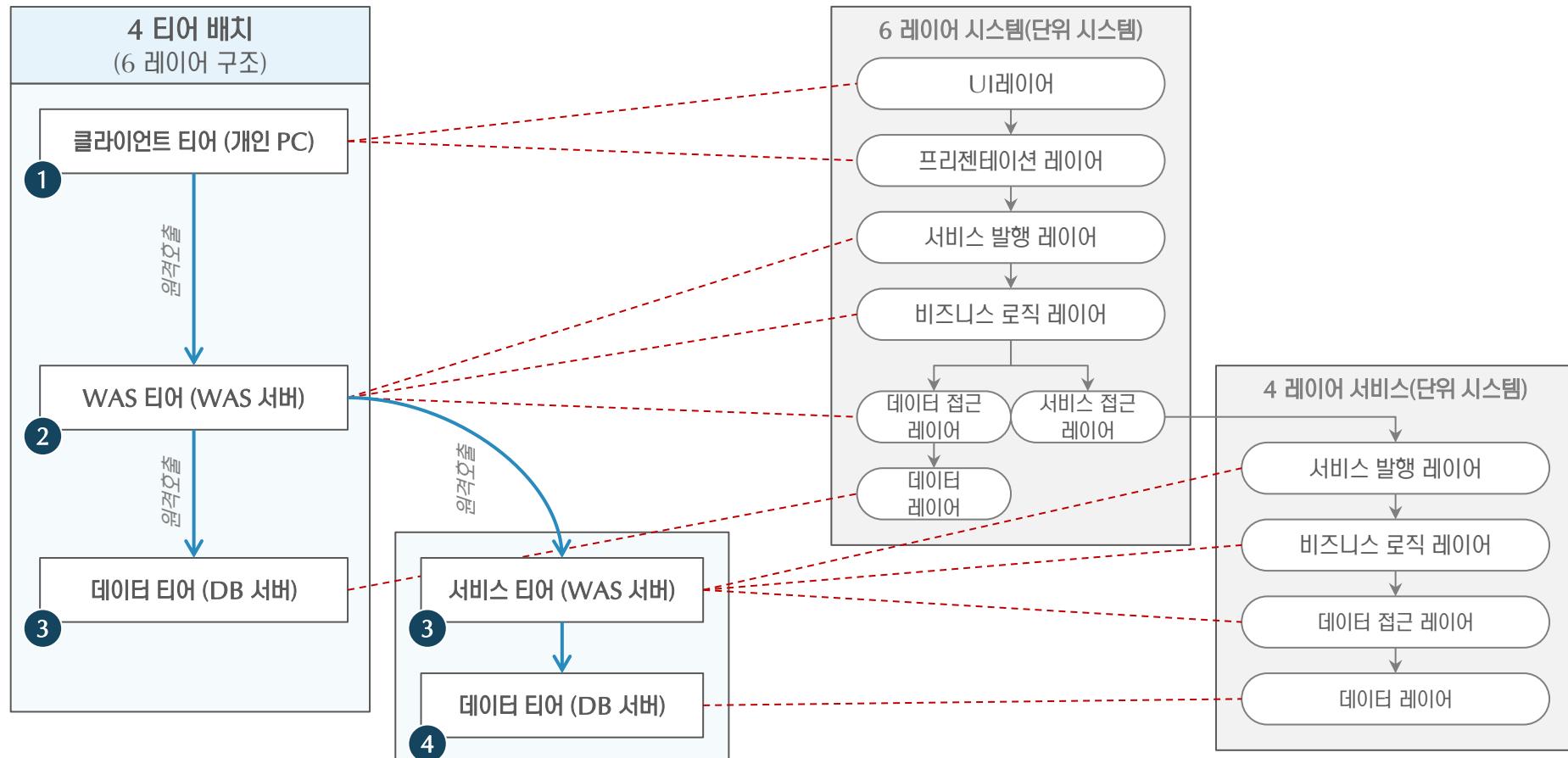
4.6 레이어와 티어(3/4)

- ✓ 레이어가 늘어나더라도 티어의 변화가 없을 수도 있습니다. 6 레이어로 변경되었으나 티어는 여전히 3티어입니다.
- ✓ “처리 대상(UI, 로직, 데이터)”이 주로 티어와 일치하고, “변화 대응”을 위한 레이어는 별도의 티어를 만들지 않습니다.
- ✓ 다른 관점에서 티어를 실행 프로세스와 거의 일치하기도 합니다.
- ✓ 이 예를 보면, 티어와 레이어는 관련이 있지만, 일치하지는 않음을 알 수 있습니다.



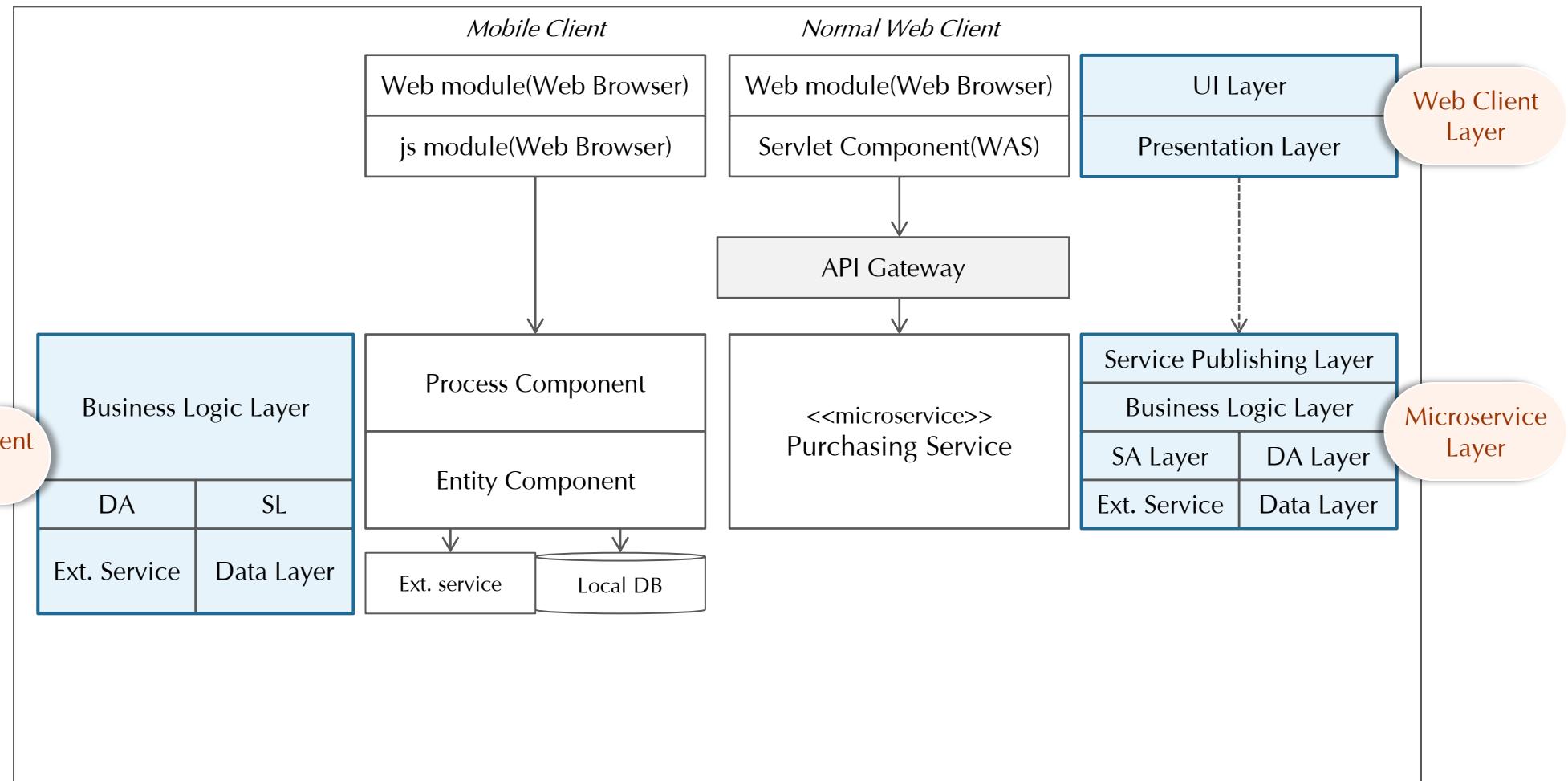
4.6 레이어와 티어(4/4)

- ✓ 레이어 개수는 단위 시스템(시스템, 서브 시스템, 서비스, 마이크로 서비스) 안에서 결정합니다.
- ✓ 티어 개수는 하나의 기능을 완결하는데 필요한, 모든 호출 경로를 포함하는 최대 티어 개수로 결정합니다.
- ✓ 아래 시스템은 4티어 아키텍처를 가지고 있으며, 각 단위 시스템 별로 6레이어, 4레이어로 설계하였습니다.



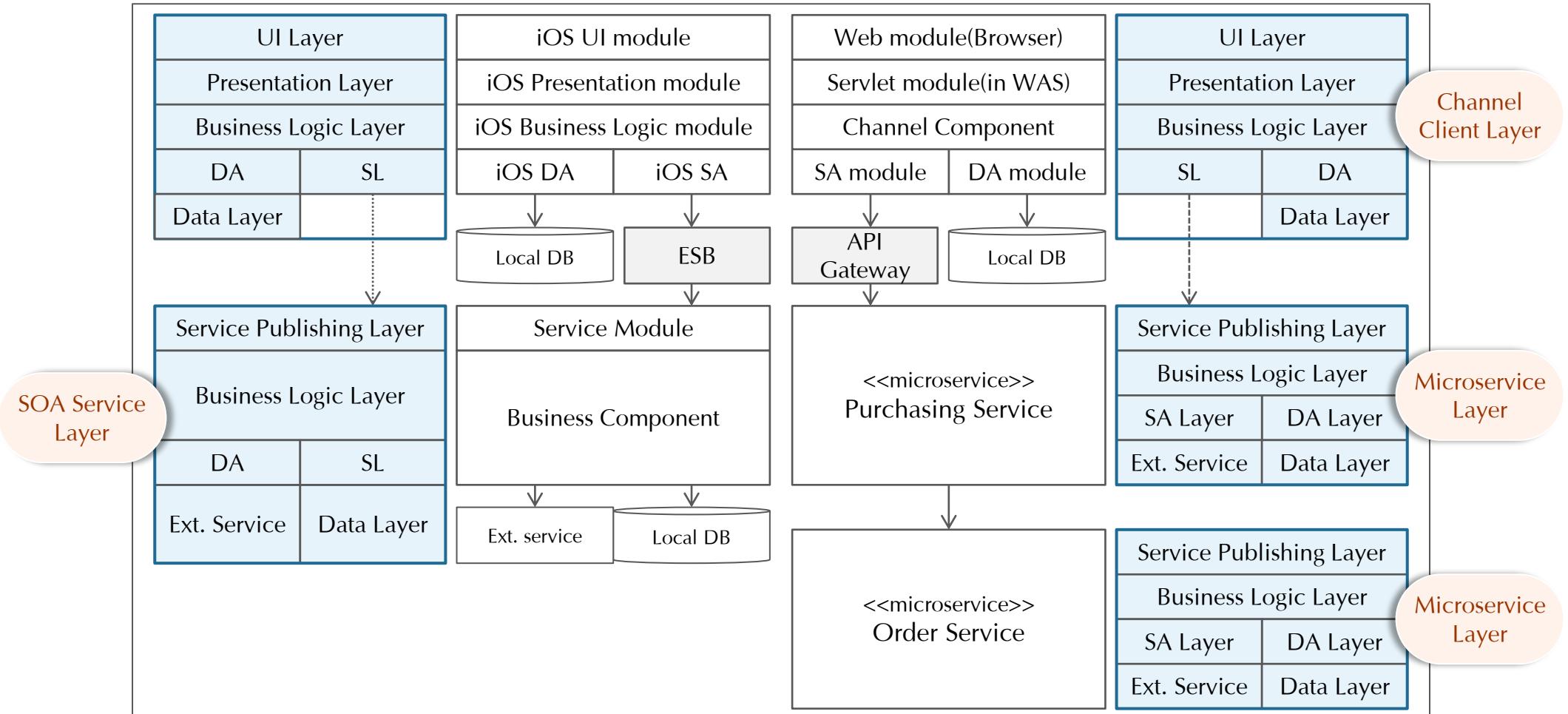
4.7 서브 시스템과 레이어(1/2)

- ✓ 컨테이너 관점에서 바라 보았을 때, 다양한 유형의 서브 시스템/서비스를 볼 수 있습니다.
- ✓ 각 서브 시스템/서비스 별로 서로 다른 레이어 구조를 가질 수 있습니다. 로직의 유무에 따라 레이어 구성이 달라집니다.



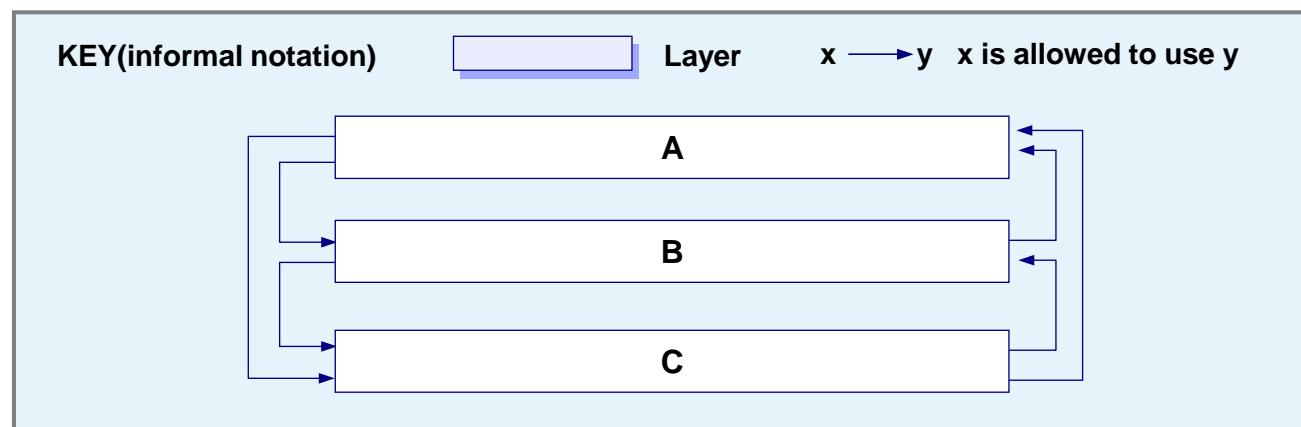
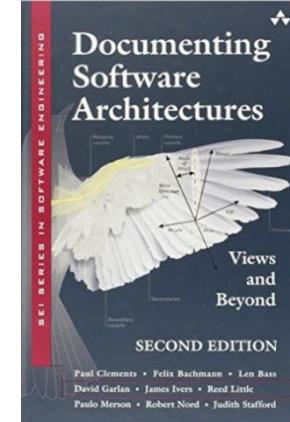
4.7 서브 시스템과 레이어(2/2)

- ✓ 로직 처리를 포함하는 클라이언트와 로직을 포함하지 않는 클라이언트의 레이어 구조는 다를 수 밖에 없습니다.
- ✓ 서버 사이드의 서브 시스템/서비스와 클라이언트 사이드 서브 시스템/서비스의 레이어 구조가 다를 수 밖에 없습니다.



4.8 레이어 스타일 (from classic)

- ✓ 소프트웨어를 레이어 단위로 분할
- ✓ 각 레이어는 인터페이스를 가진 가상 머신(*virtual machine*)
- ✓ 가상 머신은 엄격한 순서 관계에 따라 상호작용
 - 레이어 스타일 상에서의 관계 (A, B)
 - 레이어 A 는 레이어 B 의 상위 레이어
 - 레이어 A 는 레이어 B 에 의해 제공되는 서비스의 사용이 허가됨(*allowed to use*)
 - 레이어 아키텍처에서 하위 레이어는 상위 레이어 사용 불가
 - 상위 레이어의 경우 바로 하위 레이어 뿐만 아니라 임의의 하위 레이어도 사용 가능
 - 사용 관계는 상위 레이어에서 하위 레이어로 향함



[유사하지만 레이어 아키텍처가 아닌 예]

4.8 레이어 스타일 (from classic)

✓ 가상 머신

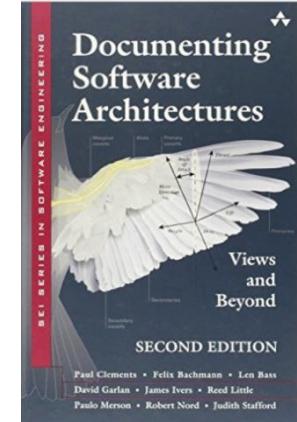
- 레이어의 목적은 상위 레이어에 가상 머신 제공
- 이식성(*portability*) 향상이 목적
 - 특정 플랫폼에 종속적인 기능을 노출해서는 안됨
 - 플랫폼에 독립적인 추상 인터페이스로 은폐

✓ 사용 허가(*allowed to use*) 관계

- 레이어가 낮을 수록 더 적은 기능(*facility*)을 가짐: 낮은 레이어는 특정 어플리케이션에 독립적
- 레이어가 높을 수록 플랫폼에 독립적: 높은 레이어는 어플리케이션의 세부사항에만 관련

✓ 레이어는 소스 코드를 통해 유도될 수 없음 ← 논리적인 단위임

- 사용 관계는 소스 코드에 표현 가능
- 사용 허가(*allowed to use*) 관계는 표현하지 못함



4.8 레이어 스타일 (from classic)

✓ 수정 용이성과 이식성 향상

- 정보 은폐(*information hiding*)

- 하위 레이어의 변경은 인터페이스 뒤에서 은폐됨
- 하위 레이어의 변경이 상위 레이어에 영향을 미치지 않음
- 이식성 지원을 위한 성공적인 기법
- 인터페이스는 API(*Application Programming Interface*) 이상의 의미

✓ 레이어 스타일은 실행 시간 오버헤드를 가지지 않음

✓ 시스템 구축을 위한 청사진의 역할

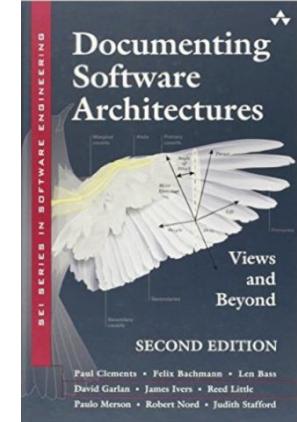
- 코딩 환경에서 어떤 서비스에 의존 가능한지를 알게 됨
- 종종 개발 팀의 작업 할당 정의

✓ 의사소통 수단

- 레이어 분할은 복잡도 관리와 구조에 대한 의사전달을 위한 중요한 도구

✓ 분석 역할

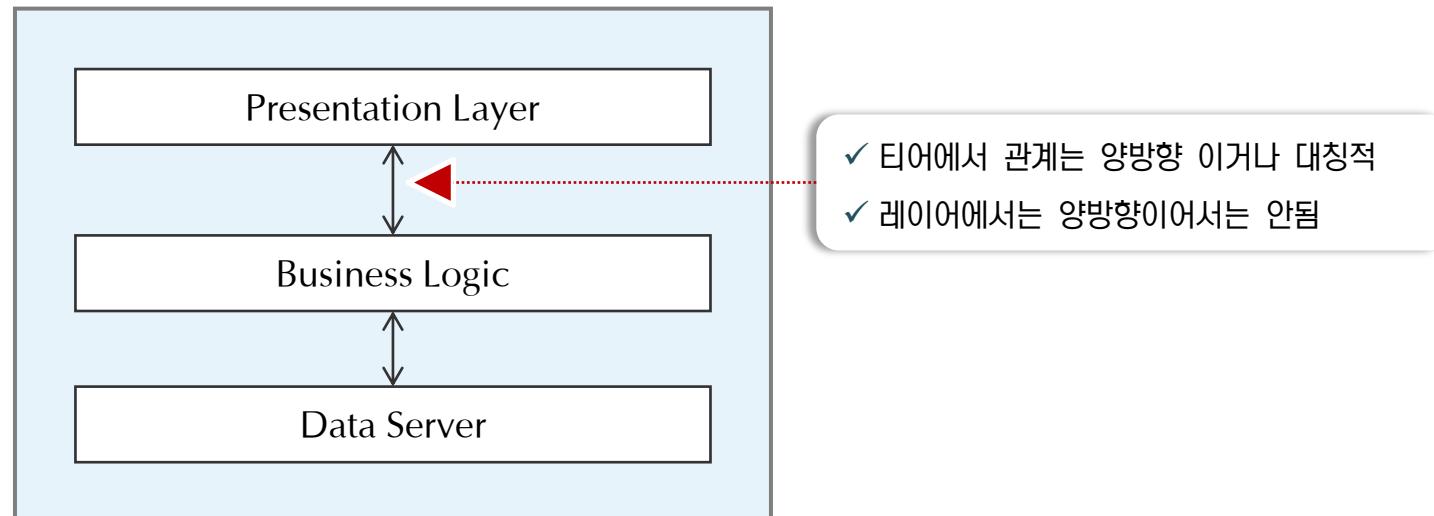
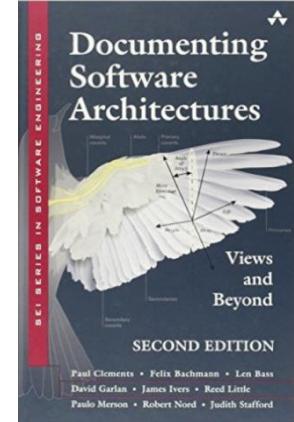
- 설계 변경이 가져오는 영향도 분석에 도움이 됨



4.8 레이어 스타일 (from classic)

✓ 티어(Tiers)

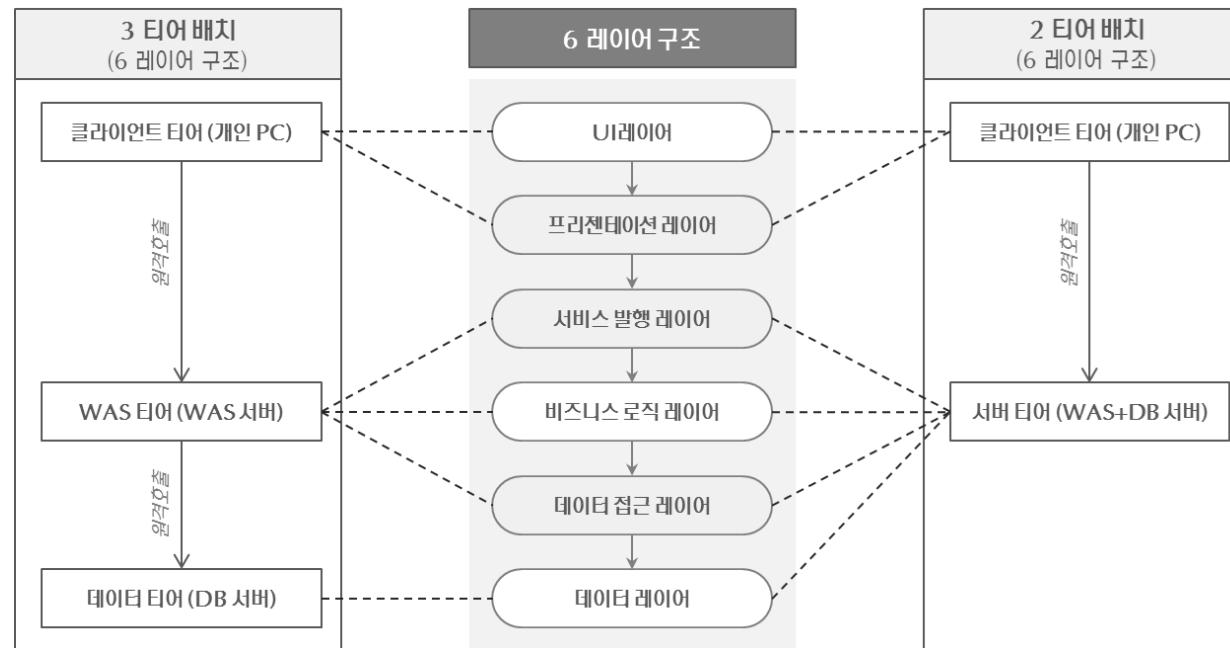
- 레이어는 티어가 아님: 레이어에는 표시되지 않지만 티어에는 표시되는 정보
 - 분산 환경 하에서 하드웨어에 대한 할당
 - 요소들 간의 데이터 흐름
 - 통신 채널의 사용과 존재 표현
 - 실행시간 효율을 기반으로 티어에 모듈 할당
 - 변경과 부분집합 개발의 용이성을 기반으로 레이어 모듈 할당
- 티어는 컴포넌트-커넥터와 할당 뷰 타입의 혼합 형태



- ✓ 티어에서 관계는 양방향 이거나 대칭적
- ✓ 레이어에서는 양방향이어서는 안됨

요약

- ✓ 소프트웨어 관련 문서에서 가장 많이 쓰이는 개념이나 그 개념을 표현하는 단어 중에 하나가 “레이어”입니다.
- ✓ 많이 사용하는 만큼 잘못 쓰이기도 하는 개념 중에 하나입니다.
- ✓ 정보를 다루는 시스템은 대부분이 레이어드 스타일을 바탕으로 하고 있어서, 올바른 이해가 특히 중요합니다.
- ✓ 레이어에 대한 올바른 이해를 통해 정보 시스템 아키텍처 수준을 한 단계 끌어 올릴 수 있습니다.



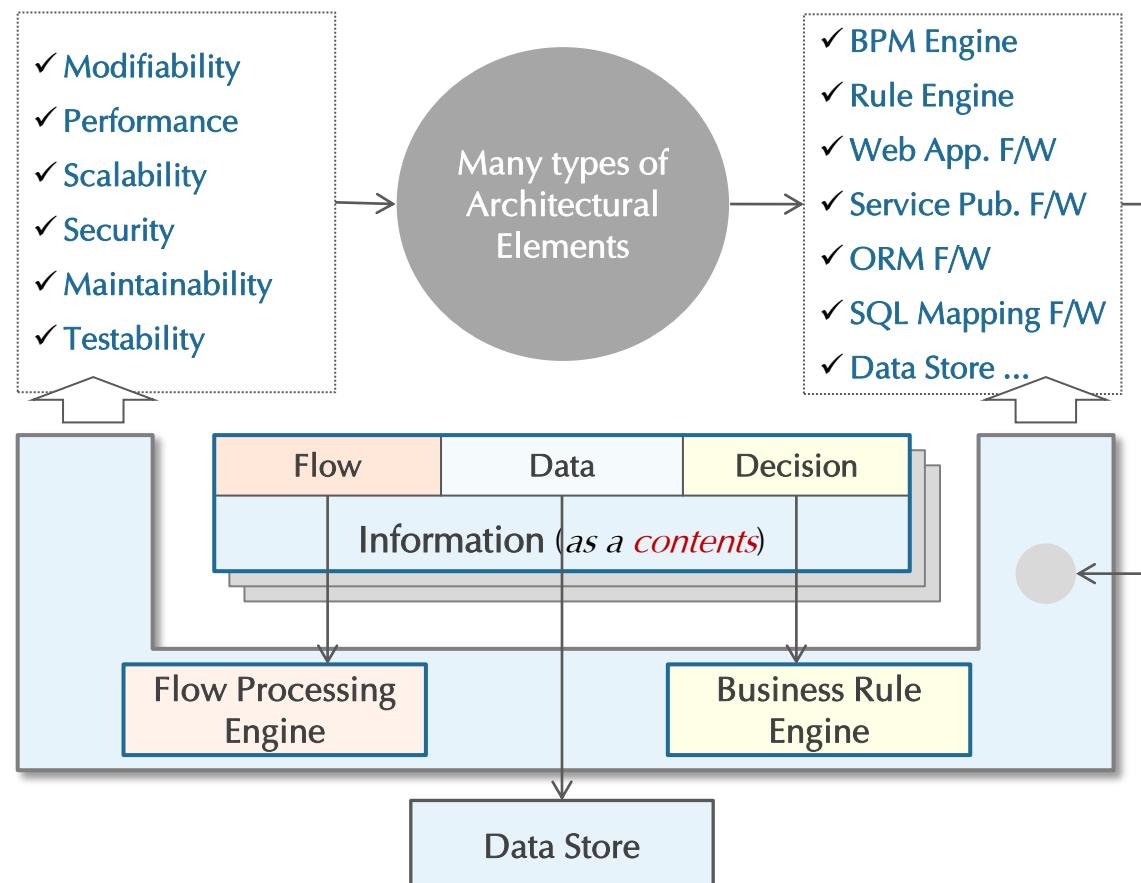


5. 도메인과 객체 모델링

-
- 5.1 도메인 위치
 - 5.2 도메인 이해
 - 5.3 도메인 객체/데이터
 - 5.4 객체 지향 프로그래밍
 - 5.5 객체 모델링
 - 5.6 도메인 모델러

5.1 도메인 위치(1/3) – 기능과 비기능

- ✓ 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.



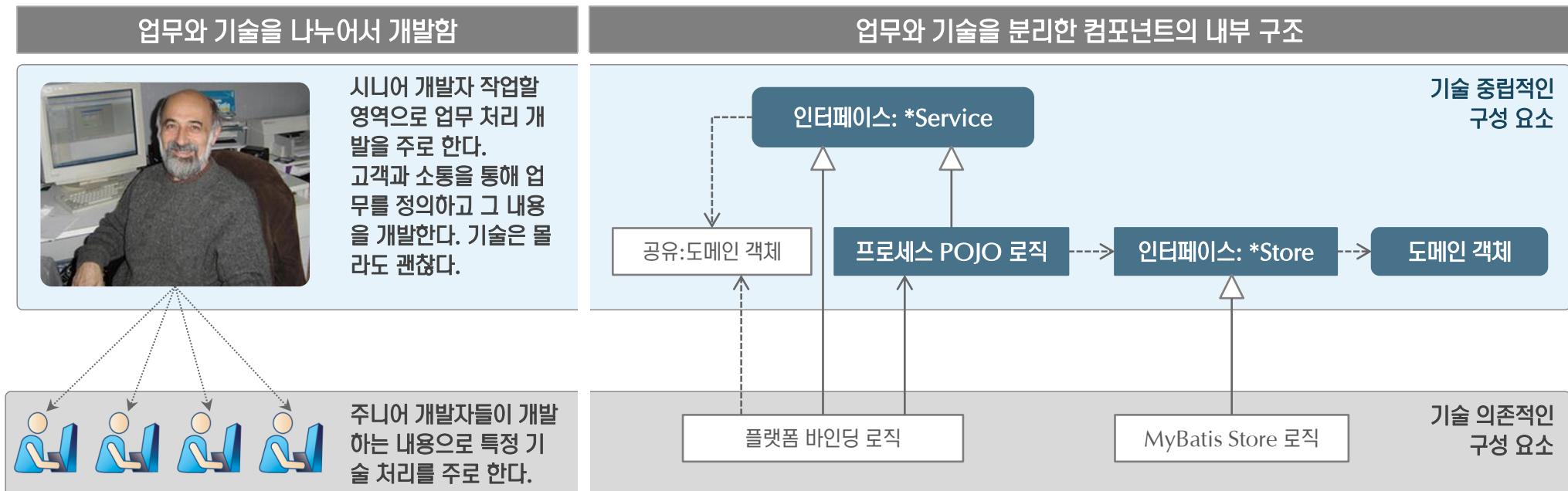
5.1 도메인 위치(2/3) – 도메인과 메커니즘

- ✓ 시스템을 구성하는 여러 요소들을 기능 관점과 비기능 관점으로 나누어 볼 수 있습니다.
- ✓ 정보 시스템에서 사람들이 다루고자 하는 것은 “정보와 처리”이며, 이것은 “도메인 객체”로 표현합니다.
- ✓ UI에서 보는 정보는 보기 쉽도록 가공한 정보이며, 데이터 저장소의 데이터는 저장하기 편리하도록 가공한 정보입니다.
- ✓ 시스템은 크게 도메인과 메커니즘(아키텍처의 예전 표현)으로 나눈다고 할 수 있습니다.



5.1 도메인 위치(3/3) – 업무와 기술 분리

- ✓ 컴포넌트 설계를 통해서 개발자의 역할 배정을 효율적으로 할 수 있습니다. ← 멋지고 경제적인 설계 아이디어
- ✓ 컴포넌트를 업무를 다루는 모듈과 기술에 종속적인 모듈로 나눈다면, 의외의 아주 큰 소득(예산/인력)이 있습니다.
- ✓ 시니어 개발자는 새로운 기술에 약해서 개발에 어려움을 겪고, 주니어 개발자는 업무에 약해서 개발에 어려움을 겪습니다.
- ✓ 결국 시니어 개발자는 개발에서 손을 떼고, 업무를 잘 모르는 주니어 개발자가 개발을 주도합니다. ← 품질 문제 발생

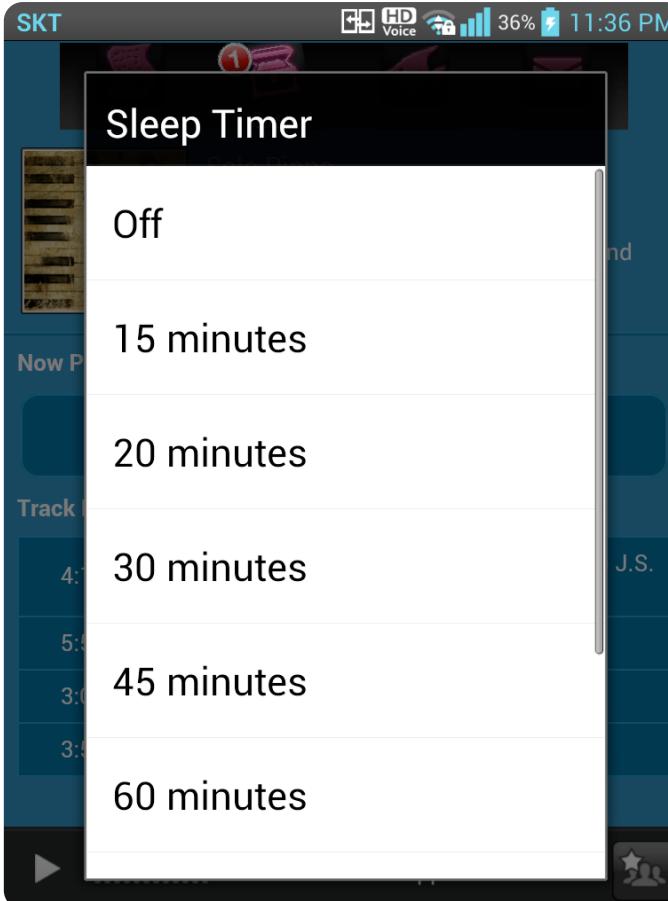


정확한 업무 이해를 바탕으로 하는 개발과 세밀한 기술을 바탕으로 하는 개발을 나누어서 진행할 수 있습니다.

업무와 기술 두 가지 모두에서 비약적인 품질 향상을 얻을 수 있고, 엔지니어를 활용도를 높일 수 있습니다.

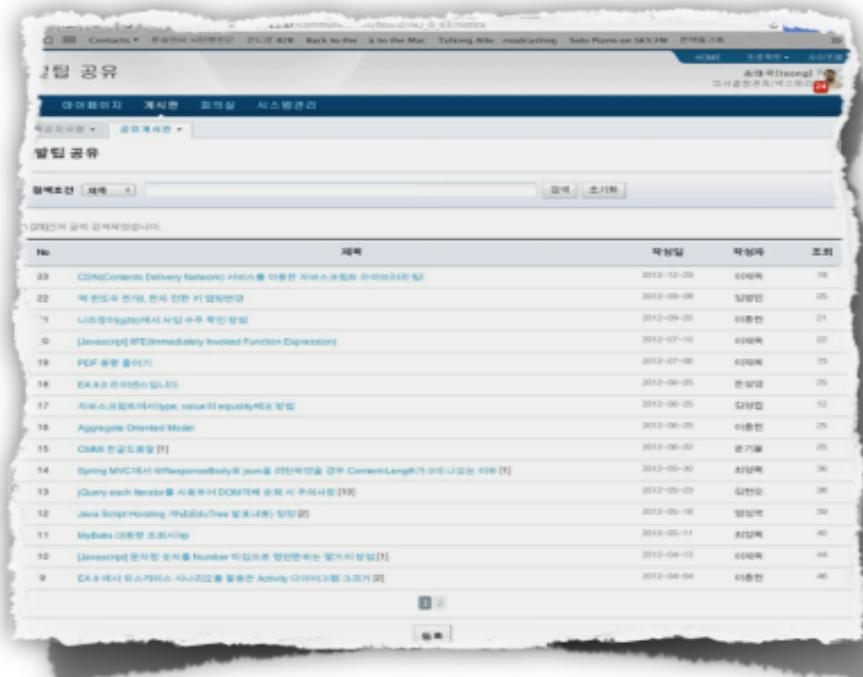
5.2 도메인 이해: 사례 I – Sleep timer

- ✓ 20분 후에 잠들고 싶은 애청자...



5.2 도메인 이해: 사례 II – Dummy board

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



5.2 도메인 이해: 사례 II – NamooBoard 1

✓ (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.

5.2 도메인 이해: 사례 II – NamooBoard 2

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

The image shows a user interface for posting content on the NamooBoard 2 platform. On the left, there is a blue user icon followed by a large black arrow pointing towards the main interface. The main interface consists of two overlapping windows.

The top window is titled "NEXTREE 전체 공지사항" (NEXTREE General Notice). It displays a list of notices with their numbers and titles:

- No 208 종무식 및 시무식 공지
- No 207 2013년 다이어리 속지 신청
- No 206 2012년 10월 프로젝트근무현황
- No 205 2012년 창립기념 트레킹 팀원 알림

In the center of the interface is a large yellow button with the text "Posting...". Below this button is a small rectangular box labeled "OFF 옵션" (Option OFF). A thick red arrow points from this box down to the bottom window.

The bottom window is titled "ON 옵션" (Option ON). It contains several configuration options:

- A date range selector showing "부터" (From) 1월 3일 21:00 and "까지" (Until) 1월 5일 18:00.
- A "통지수단" (Notification Method) section with "레벨 1" set to "E-mail" and "레벨 2" set to "SMS". There is also a checkbox for "Report ME".
- A "매니간" (Manian) checkbox.

A large red arrow points from the "OFF 옵션" box in the top window down to the "ON 옵션" window, indicating the transition from disabling to enabling options.

5.2 도메인 이해: 사례 II – NamooBoard 3

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



5.2 도메인 이해: 사례 II – NamooBoard 4

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



5.2 도메인 이해: 사례 II – NamooBoard 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 종무식이 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 레벨 2 SMS 매시간

Report ME



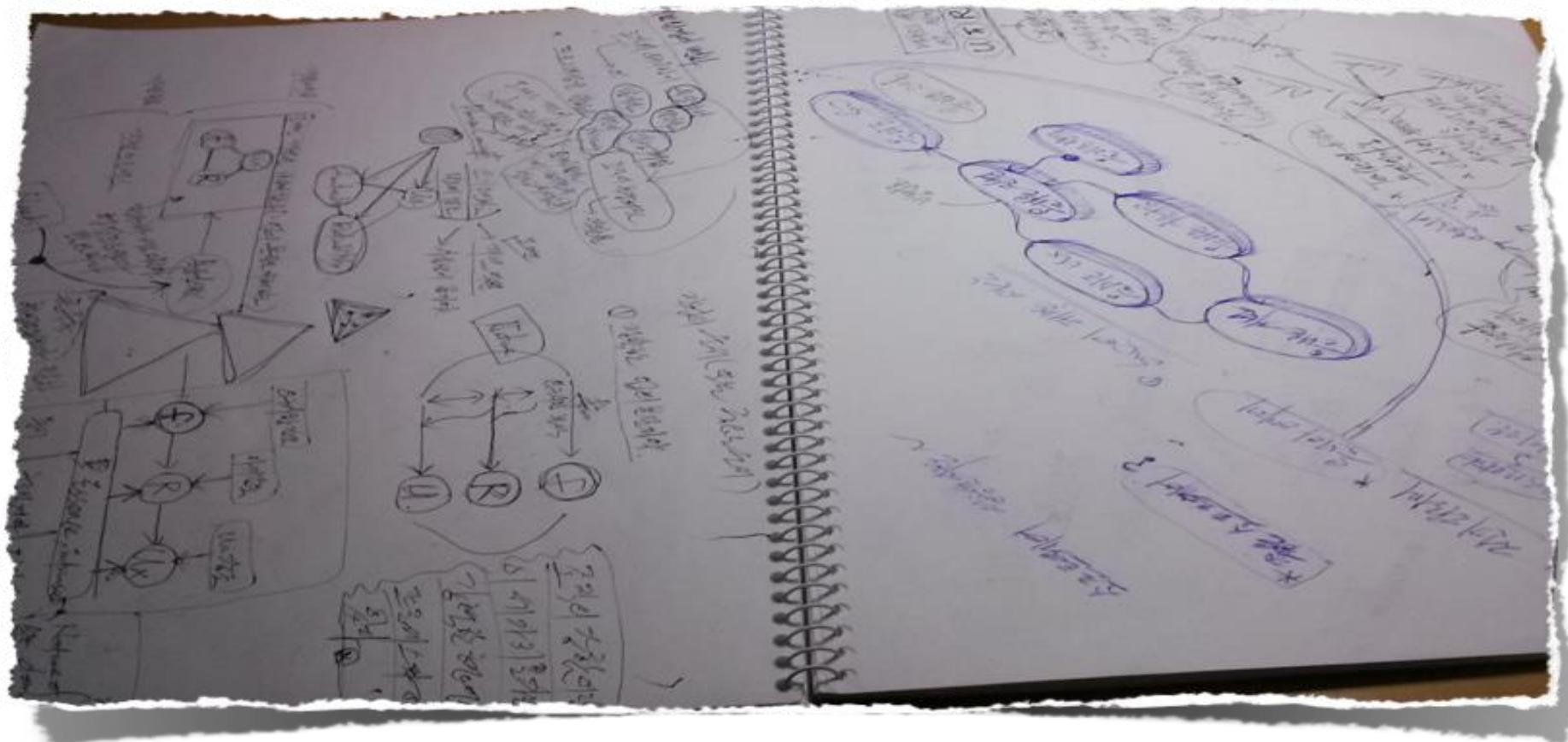
5.2 도메인 이해: 사례 III – 상품 가격

- ✓ 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.



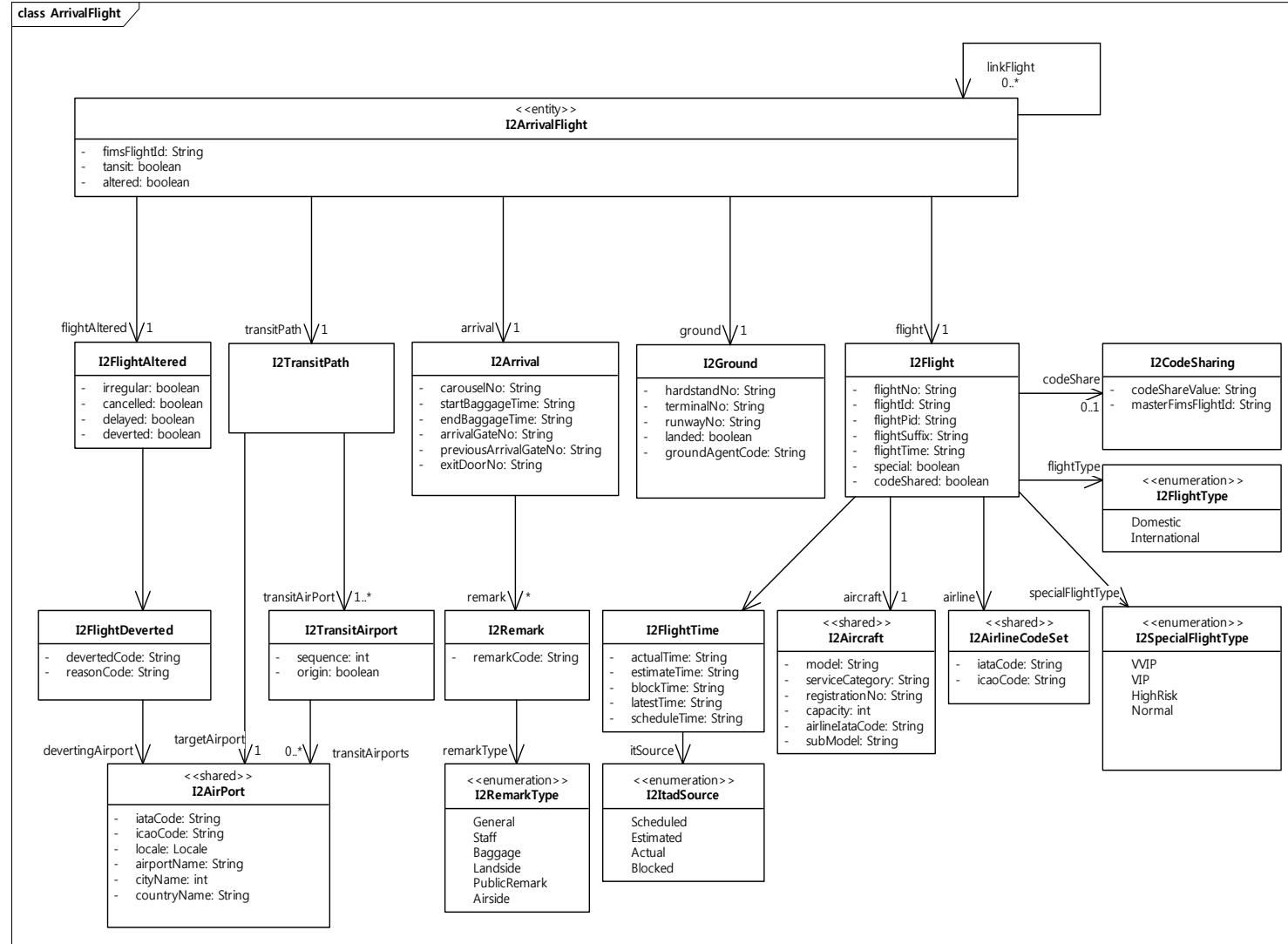
5.2 도메인 이해: 생각의 여정

- ✓ “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.

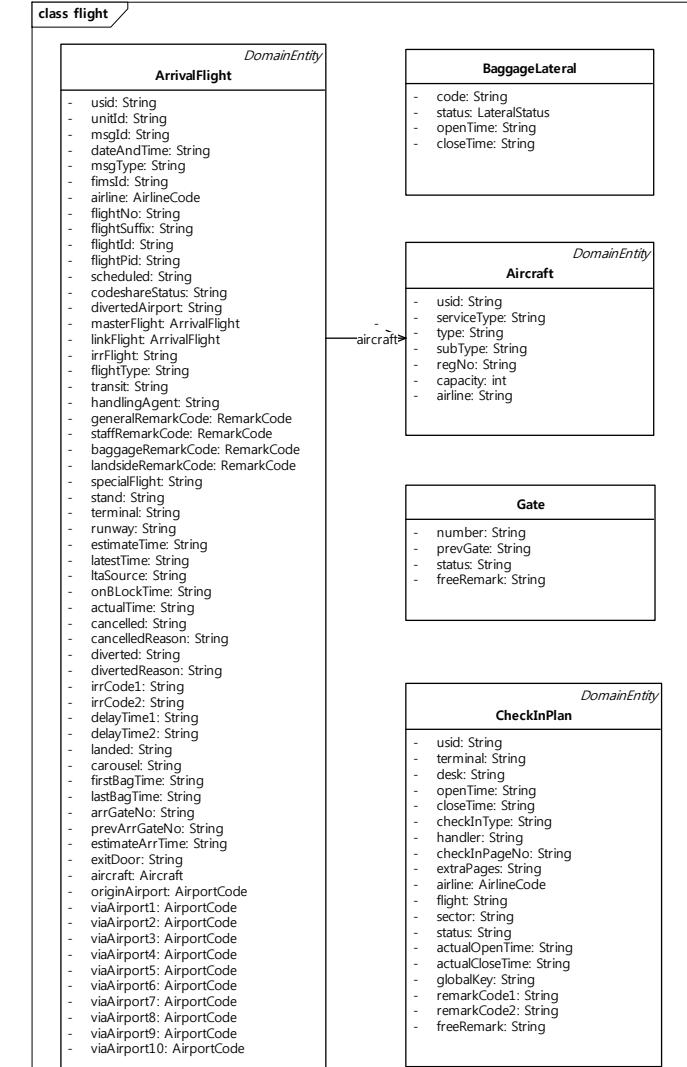


5.3 도메인 객체/데이터 1

Domain model for Arrival Flight

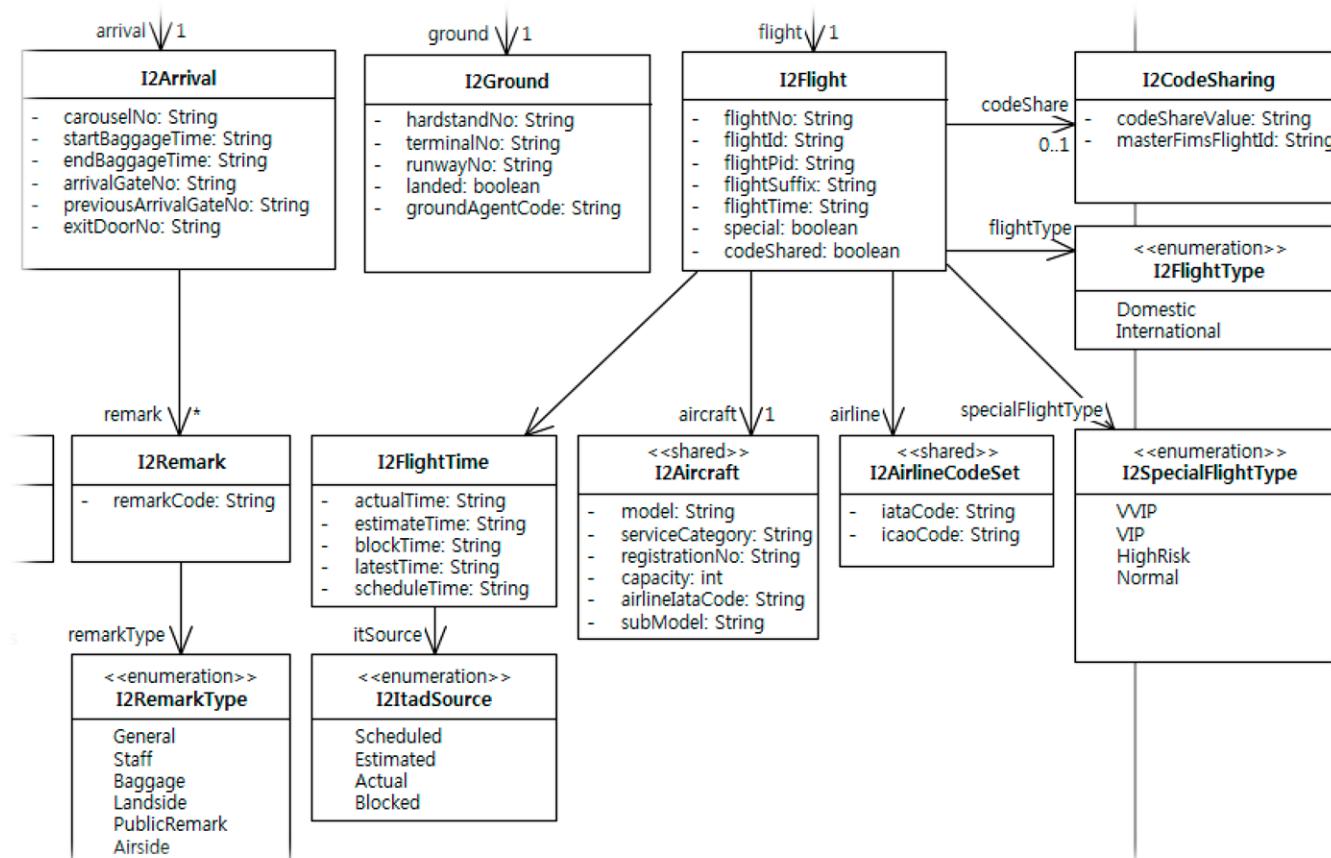


DTO for Arrival Flight



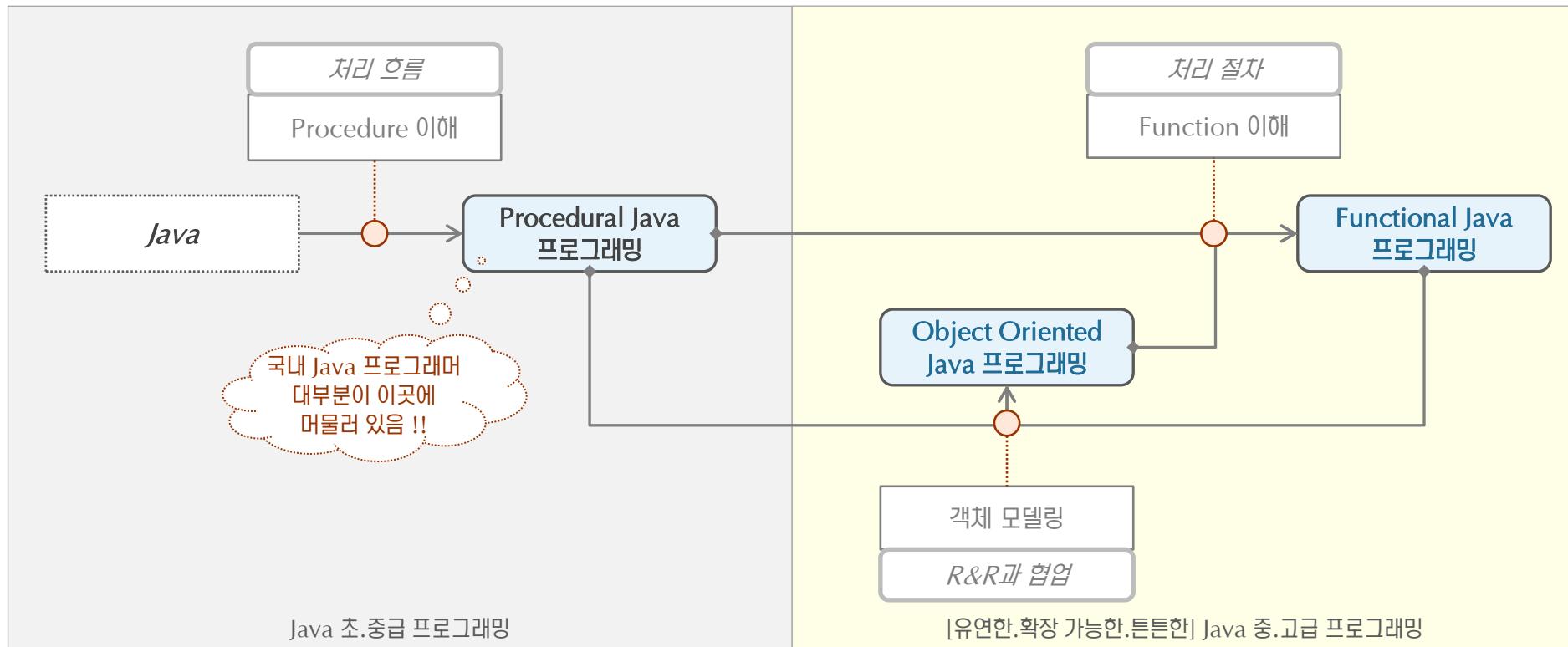
5.3 도메인 객체/데이터 2

- ✓ 데이터 기반 처리 방식은 ArrivalFlight가 갖고 있는 80여 개의 속성의 값을 직접 다루어야 합니다.
- ✓ 객체 기반 처리 방식은 ArrivalFlight를 구성하는 개념을 객체로 식별하고 객체를 기반으로 로직을 처리합니다.
- ✓ 프로그램을 살펴 보면 도메인의 개념을 프로그램 안으로 자연스럽게 옮겨왔음을 알 수 있습니다.
- ✓ 예를 들면, Remark, FlightType, Aircraft, Flight, Ground, ...



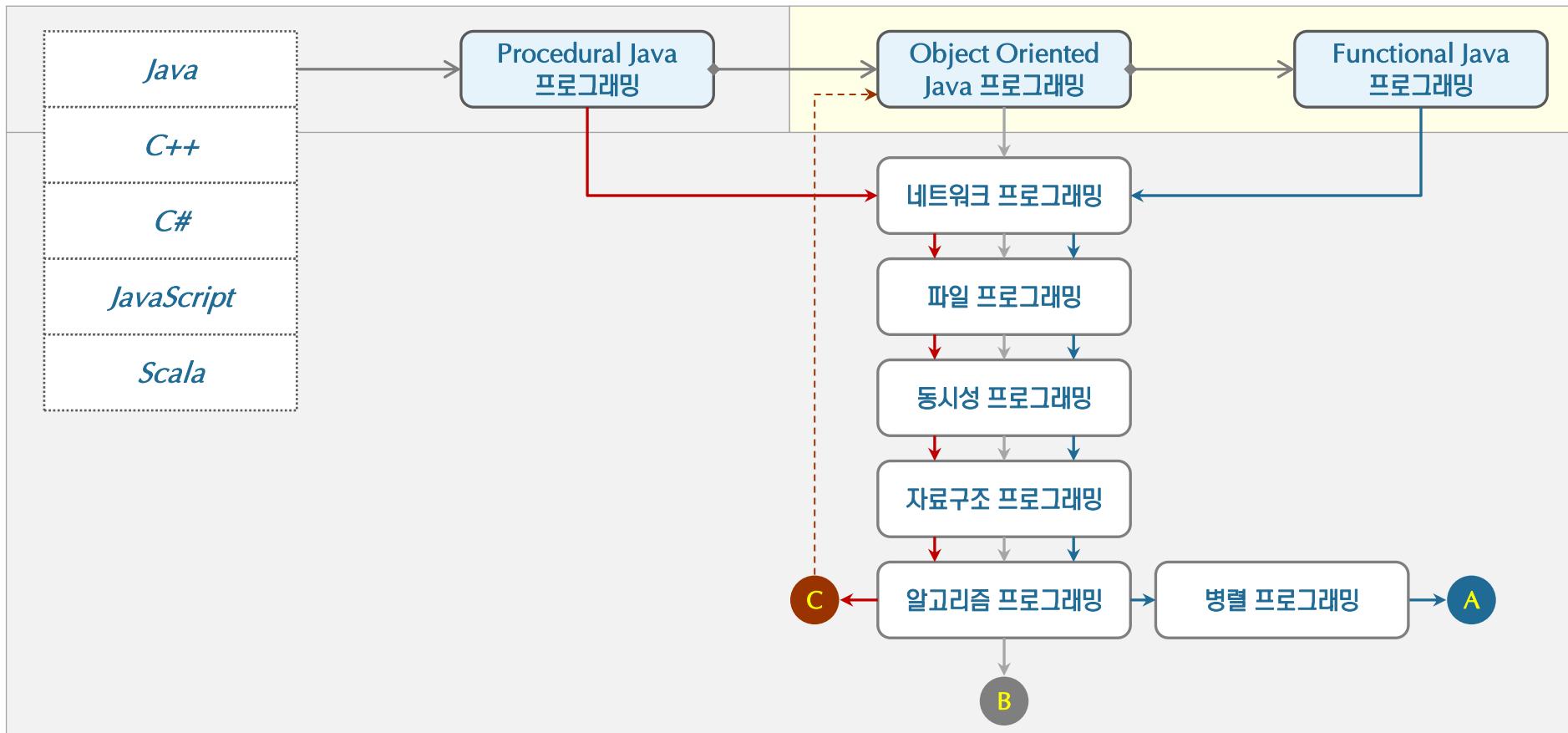
5.4 객체 지향 프로그래밍 1

- ✓ 현대의 프로그래밍 언어는 대부분 객체 지향이나 함수적인 모델링 특성을 반영하고 있습니다.
- ✓ 프로그래밍 언어를 처음 배우면, 노이만 로직에 근거한 절차 지향 특성에 기반하여 순차적 처리 개념을 익힙니다.
- ✓ 이후 객체 모델링이나 함수적인 접근 방법을 익히고 적용함으로써, 객체 지향 언어의 힘을 사용할 수 있게 됩니다.
- ✓ Functional 프로그래밍과 객체 지향 프로그래밍은 학습 순서가 없으며, 대체로 객체 지향을 먼저 익혀야 합니다.



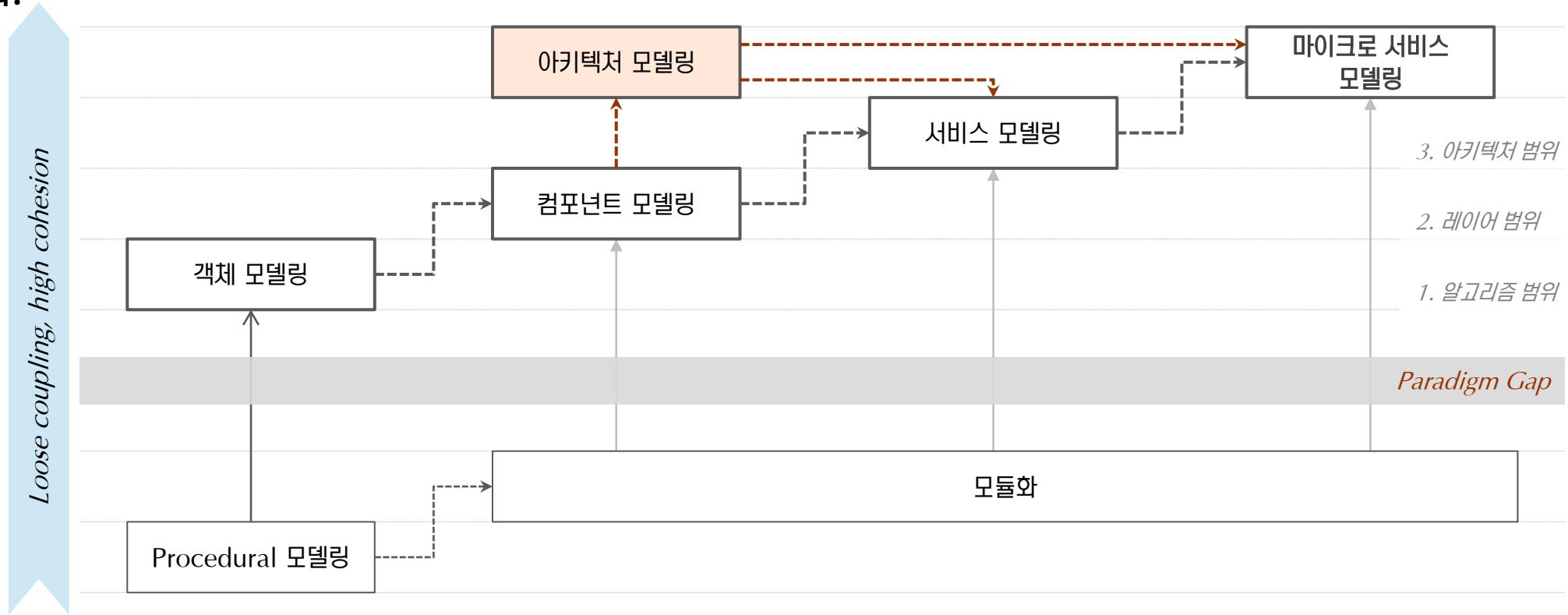
5.4 객체 지향 프로그래밍 2

- ✓ SW 엔지니어링 역량은 프로그래밍 역량이라고 할 수 있습니다. 프로그램은 모델링 결과물이기 때문입니다.
- ✓ 그리고 프로그래밍에는 여러 가지 특정 기술이 필요합니다. ← 네트워크, 파일, 동시성, 자료구조, 알고리즘 등
- ✓ 이러한 특정 기술은 한 번만 익히면 다른 언어에서도 쉽게 접근할 수 있는 특징이 있습니다.
- ✓ 프로그래밍 역량을 지속적으로 높이기 위해서는 이러한 기술 역량을 지속적으로 확보하여 가야 합니다.



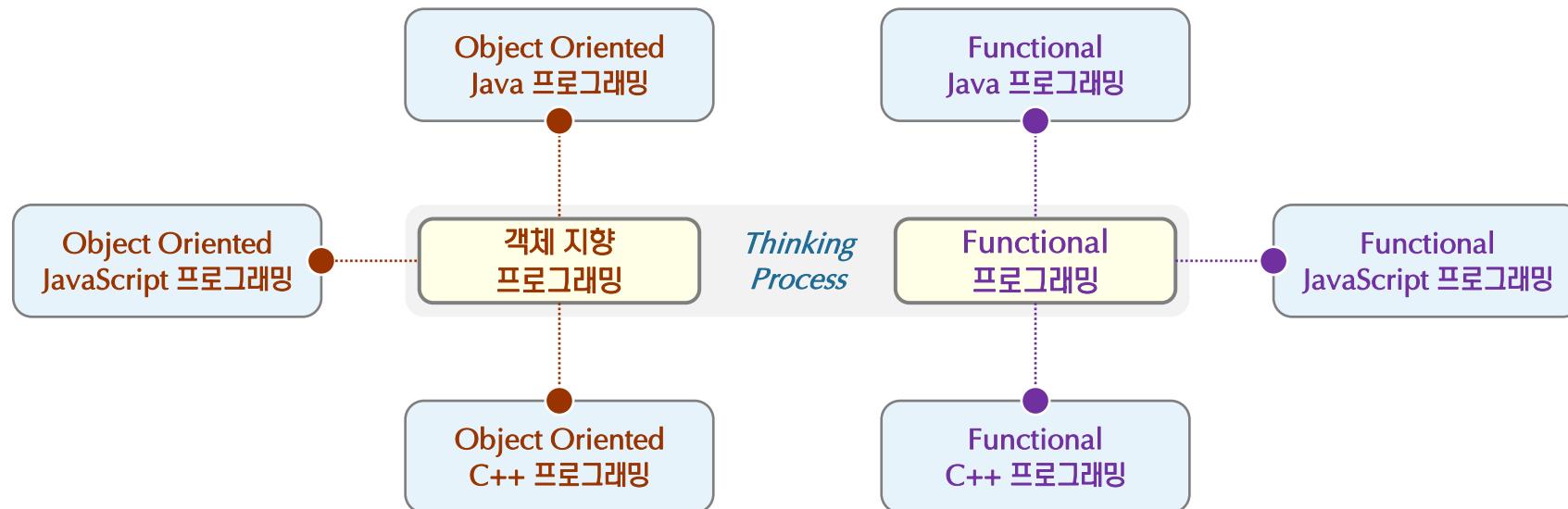
5.4 객체 지향 프로그래밍 3

- ✓ 모델링은 문제를 풀 때, 대상을 단순화 해주고, 구조적으로 생각하는 방식과 절차를 제시해 줍니다.
- ✓ 현대의 소프트웨어 모델링은 객체 모델링으로부터 시작합니다. 객체 지향 개념은 기존의 절차 지향 과는 완전히 다릅니다.
- ✓ 객체 → 컴포넌트 → 서비스 개념으로 모델링 기술이 발전하면서 SW는 눈부시게 발전을 거듭했습니다.
- ✓ 비즈니스 지원 애플리케이션 설계의 종착역은 “마이크로서비스 모델링”이며, 최근 2~3년 사이에 모델링 트랜드가 되었습니다.



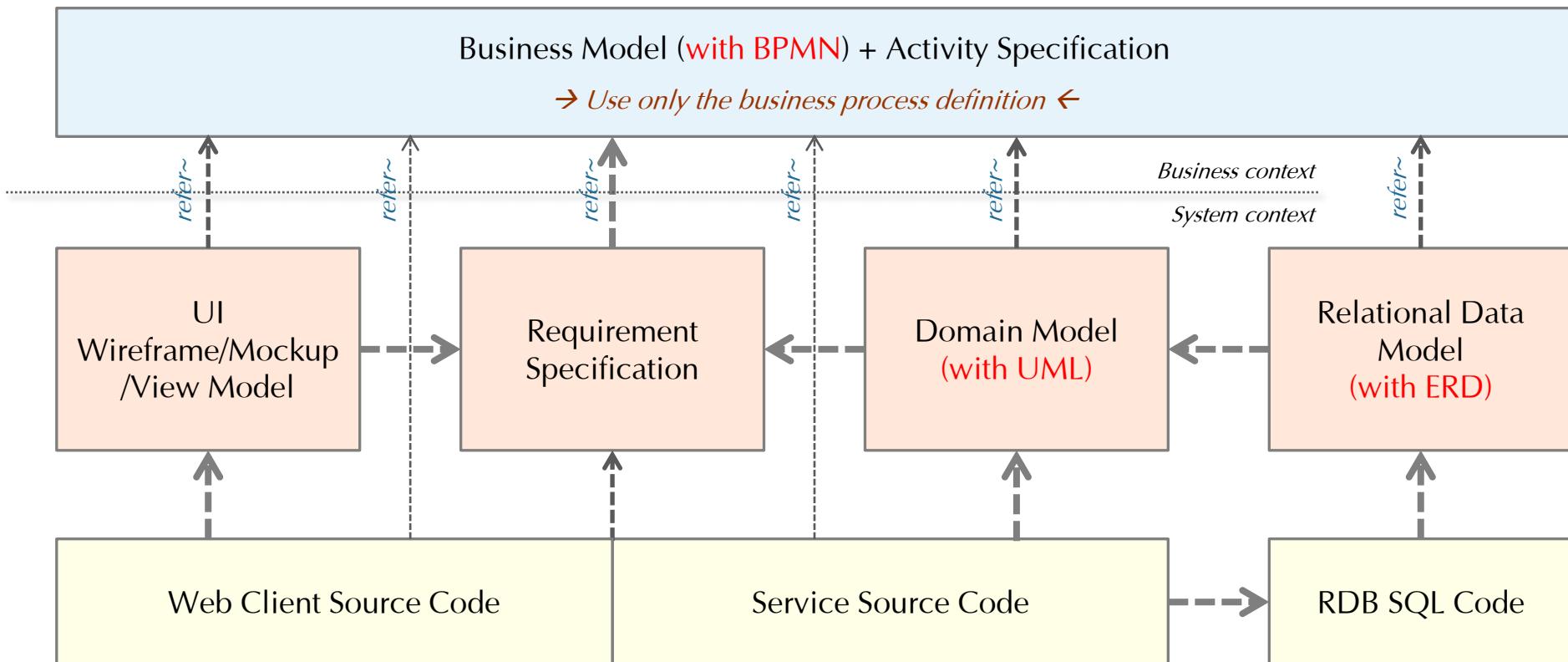
5.4 객체 지향 프로그래밍 4

- ✓ 객체 모델링 특성이나 함수적인 접근 방법 특성은 한 번 이해하고 나면 다른 언어에도 그대로 적용할 수 있습니다.
- ✓ Java 프로그래밍에 객체 지향 특성을 반영하는데 익숙하다면, JavaScript나 C++에서도 그대로 적용할 수 있습니다.
- ✓ Functional 프로그래밍도 마찬가지입니다. 한 가지 언어로 습득하고 나면 다른 언어에도 바로 적용할 수 있습니다.
- ✓ 프로그래밍 언어가 도구라면, 객체 지향과 Functional 접근 방법은 문제를 풀어가는 방식, 즉 생각하는 방법입니다.



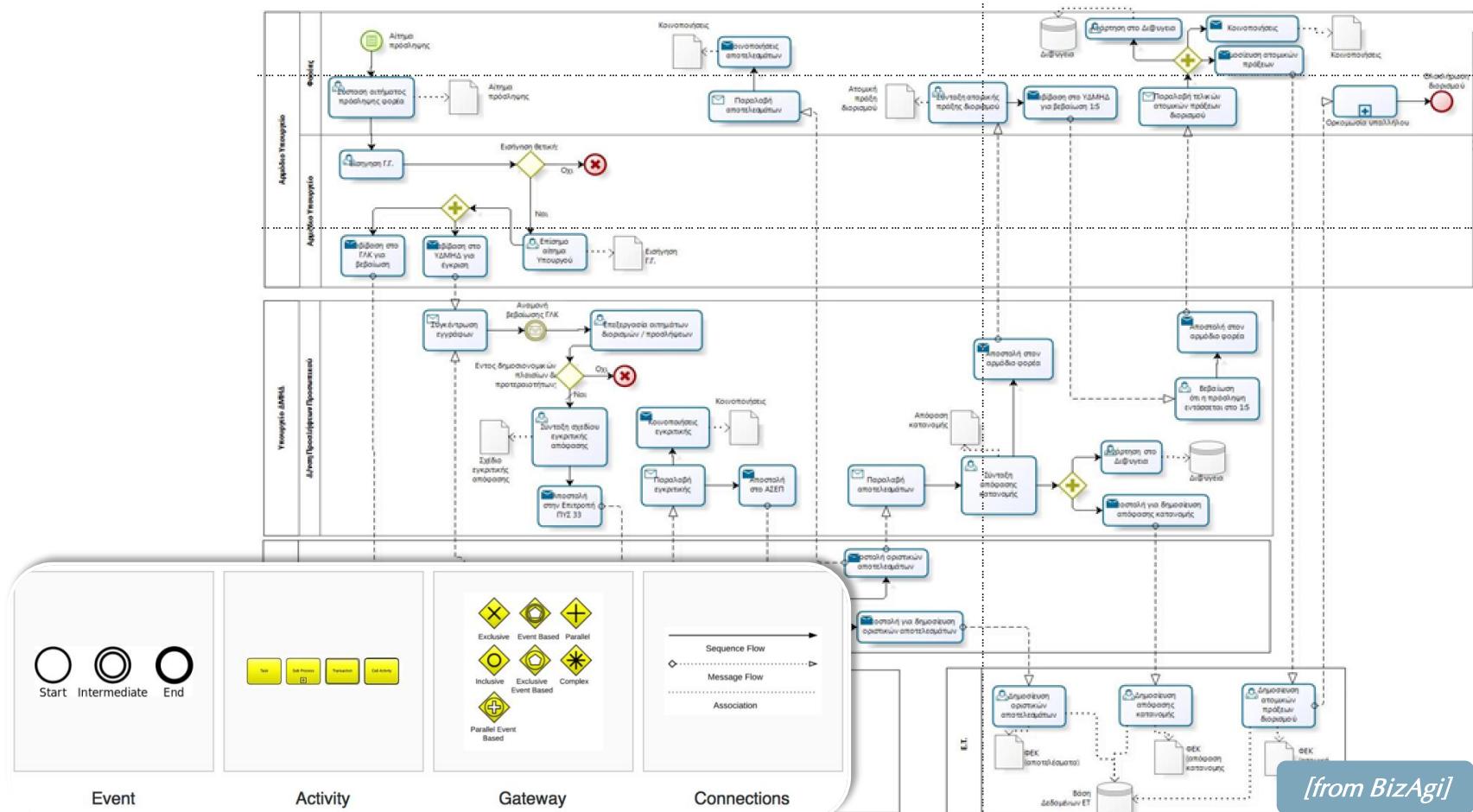
5.5 객체 모델링(1/11) – 개요(도메인 중심)

- ✓ 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.



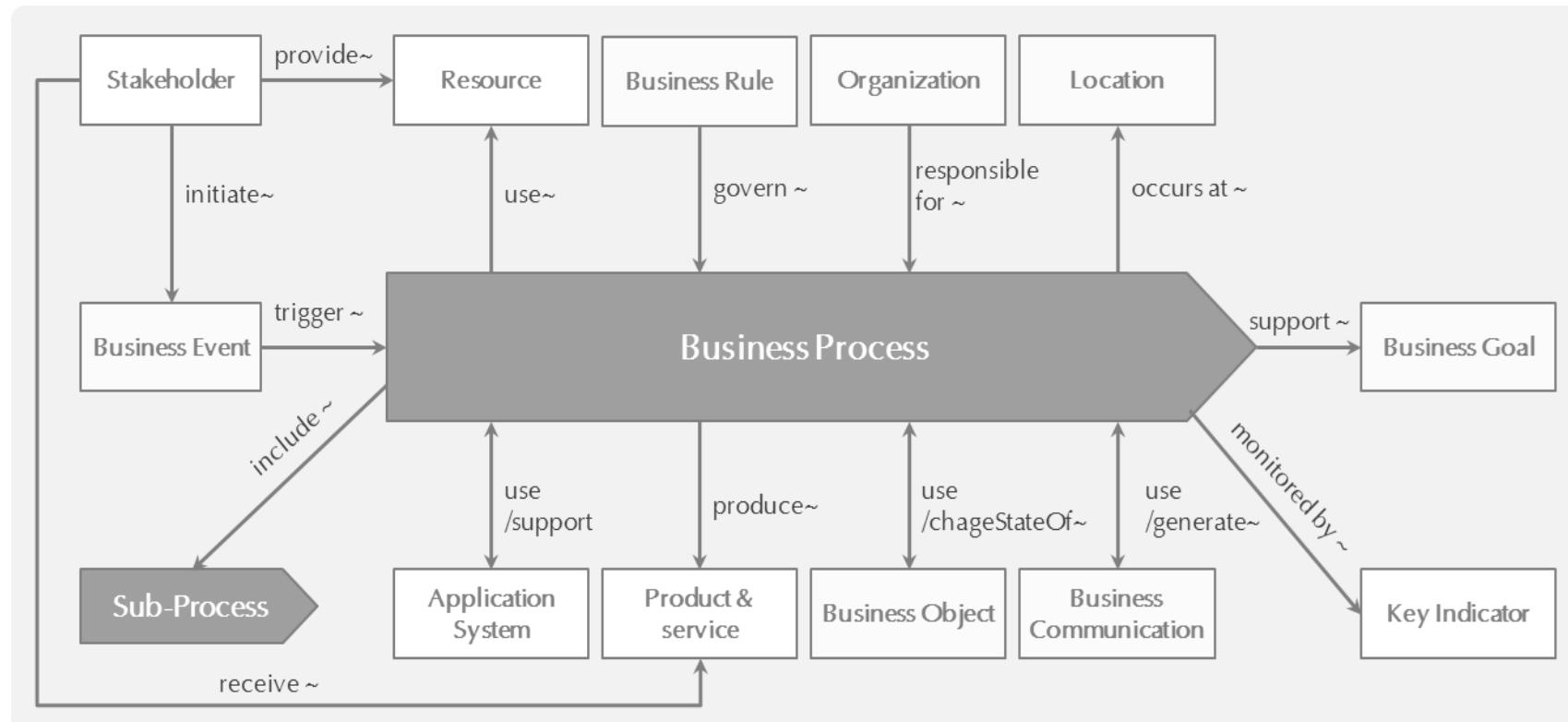
5.5 객체 모델링(2/11) – Business Process Model 1

- ✓ BPMN의 일자 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.



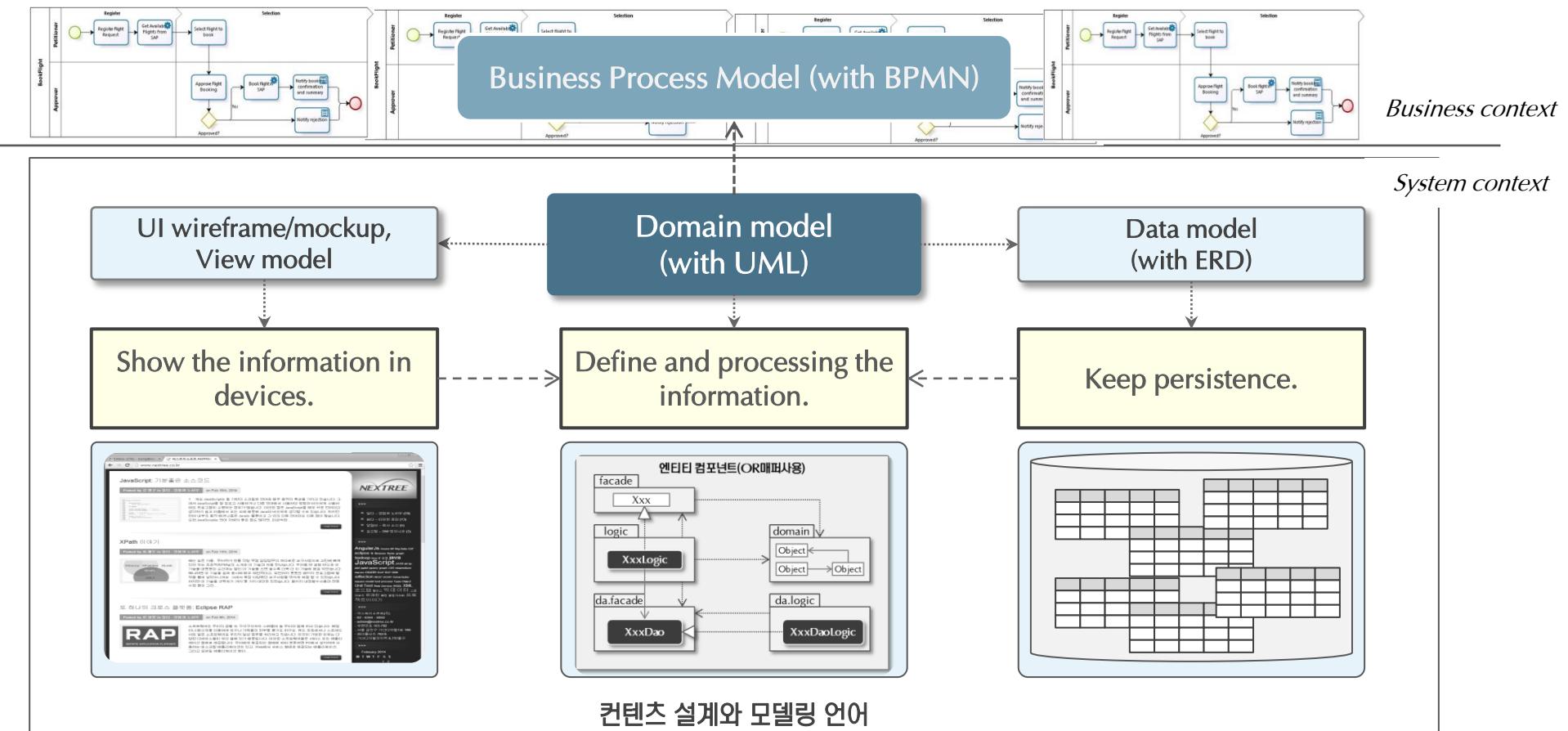
5.5 객체 모델링(3/11) – Business Process Model 2

- ✓ 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



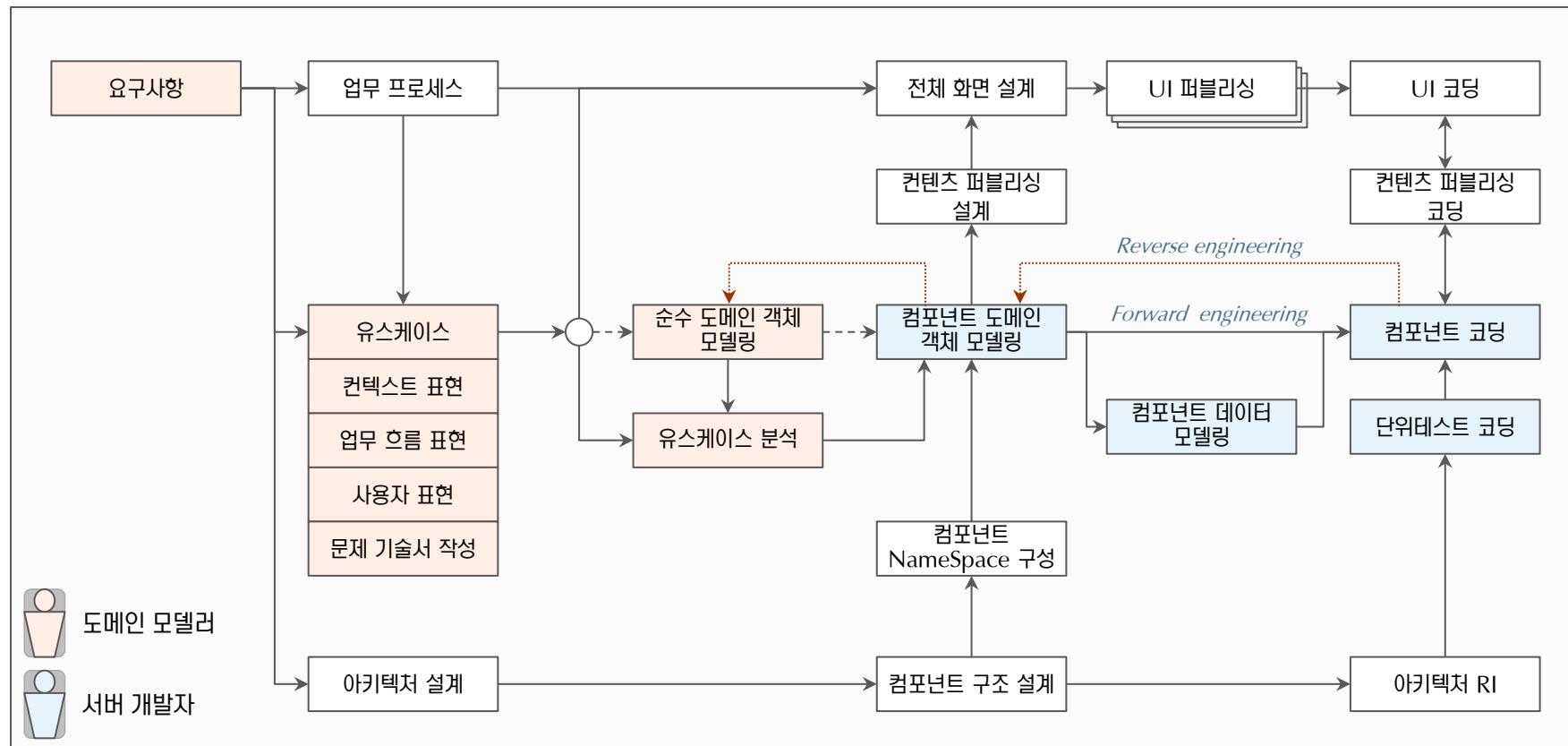
5.5 객체 모델링(4/11) – 도메인 모델

- ✓ 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.



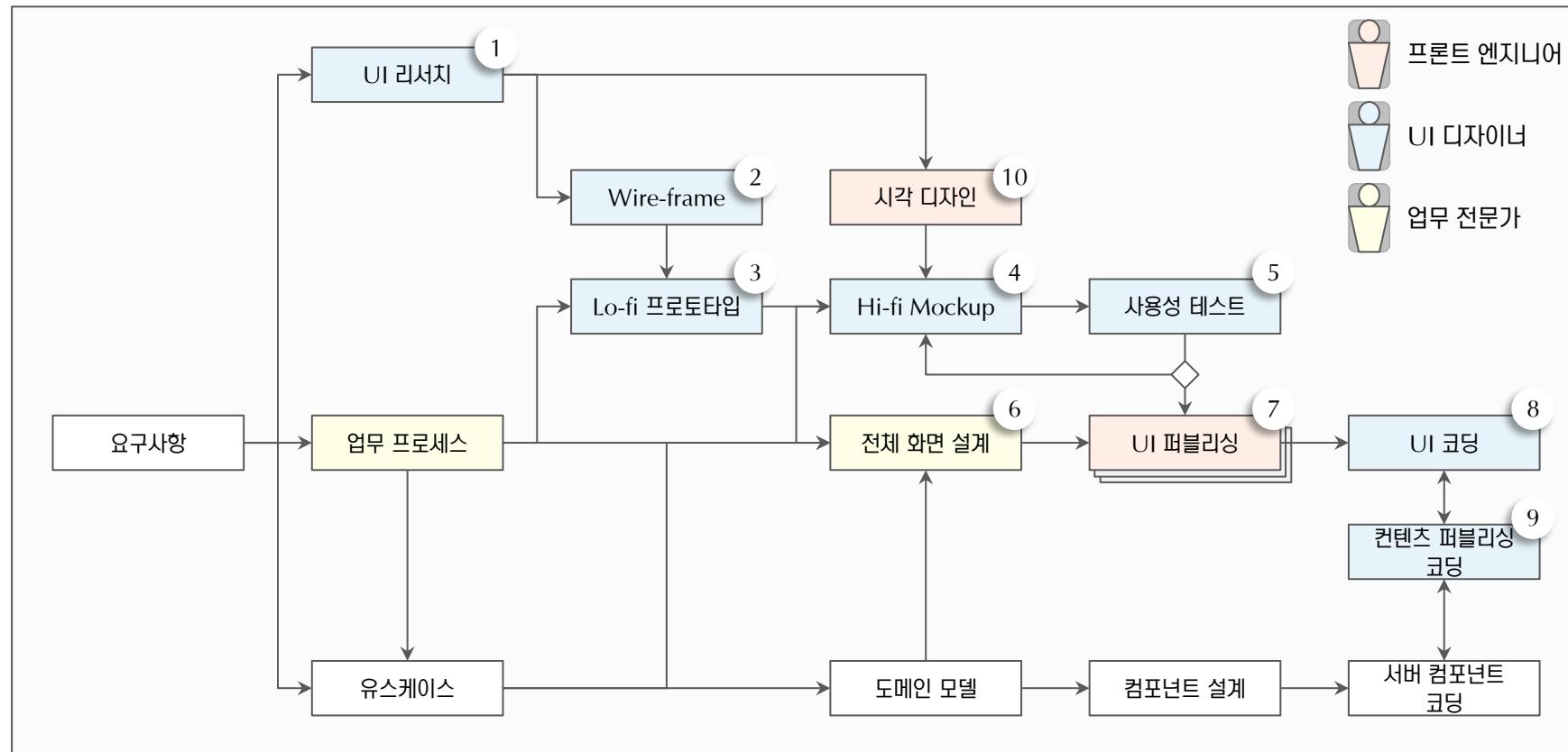
5.5 객체 모델링(5/11) – 흐름 1

- ✓ 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.



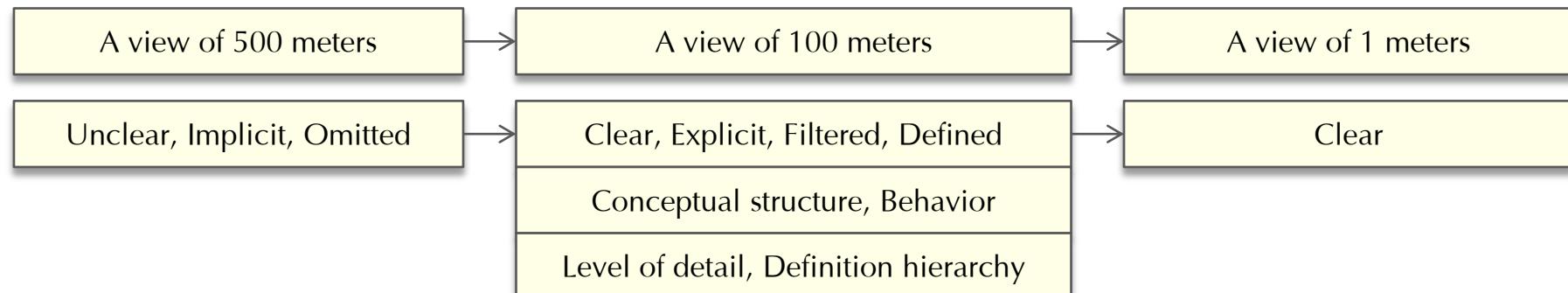
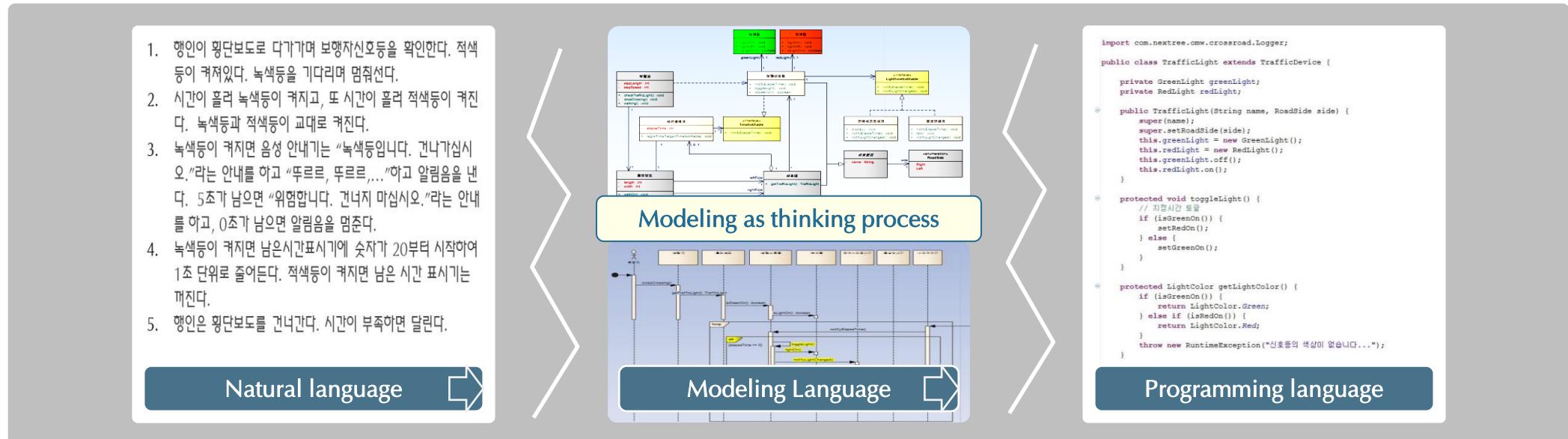
5.5 객체 모델링(6/11) – 흐름 2

- ✓ 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.



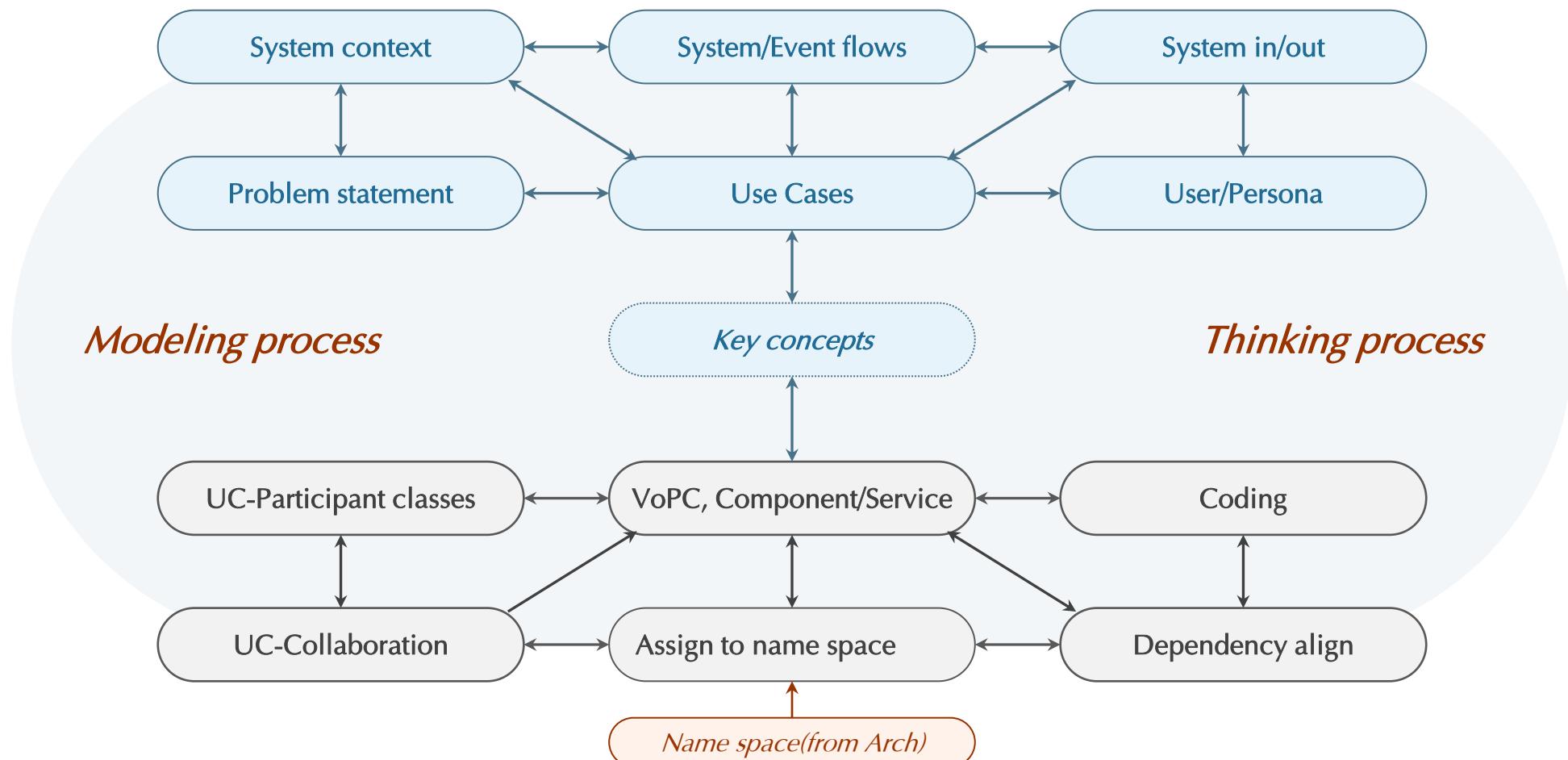
5.5 객체 모델링(7/11) – Language transition

- ✓ 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.



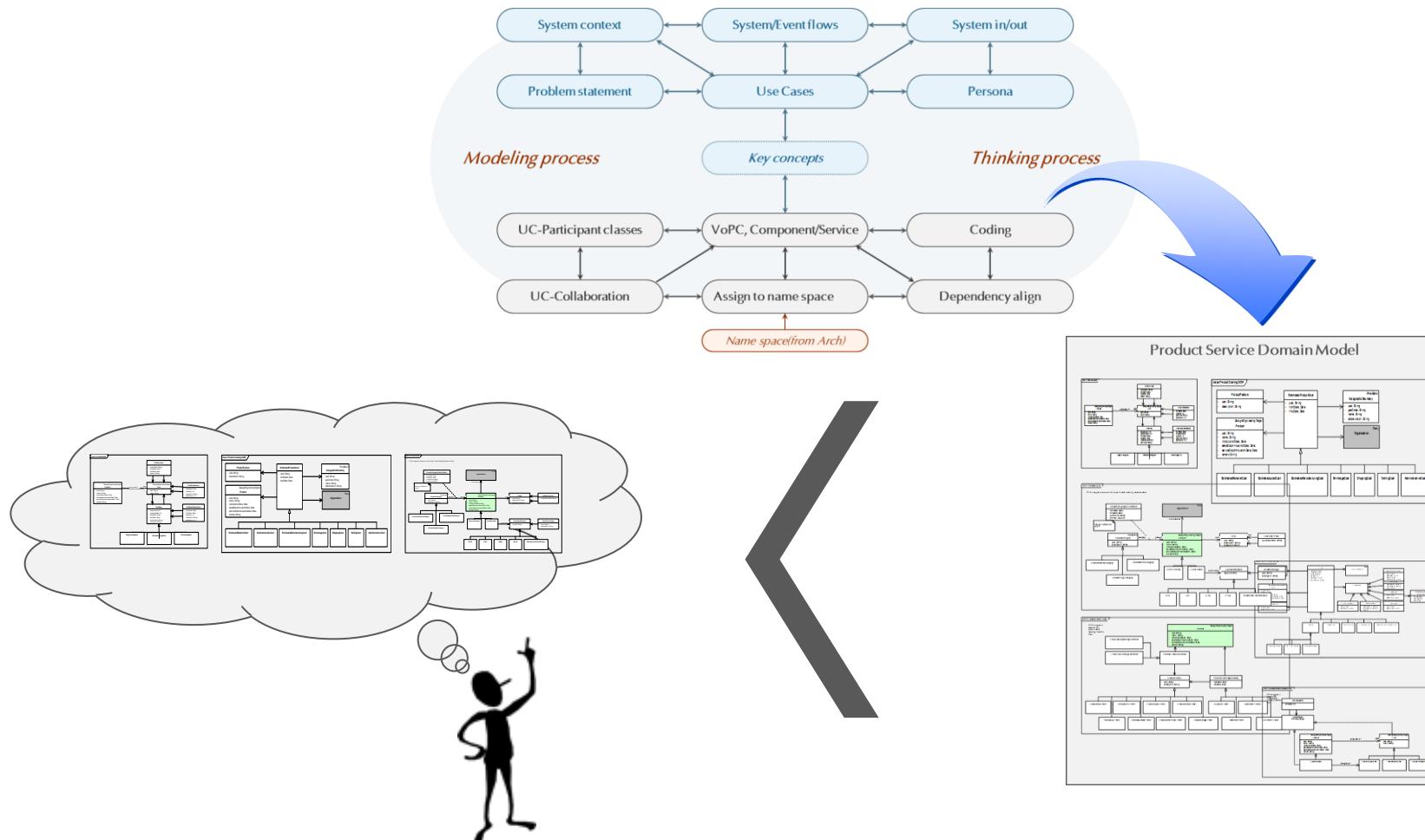
5.5 객체 모델링(8/11) – 도메인 을 생각하는 과정 1

- ✓ 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.



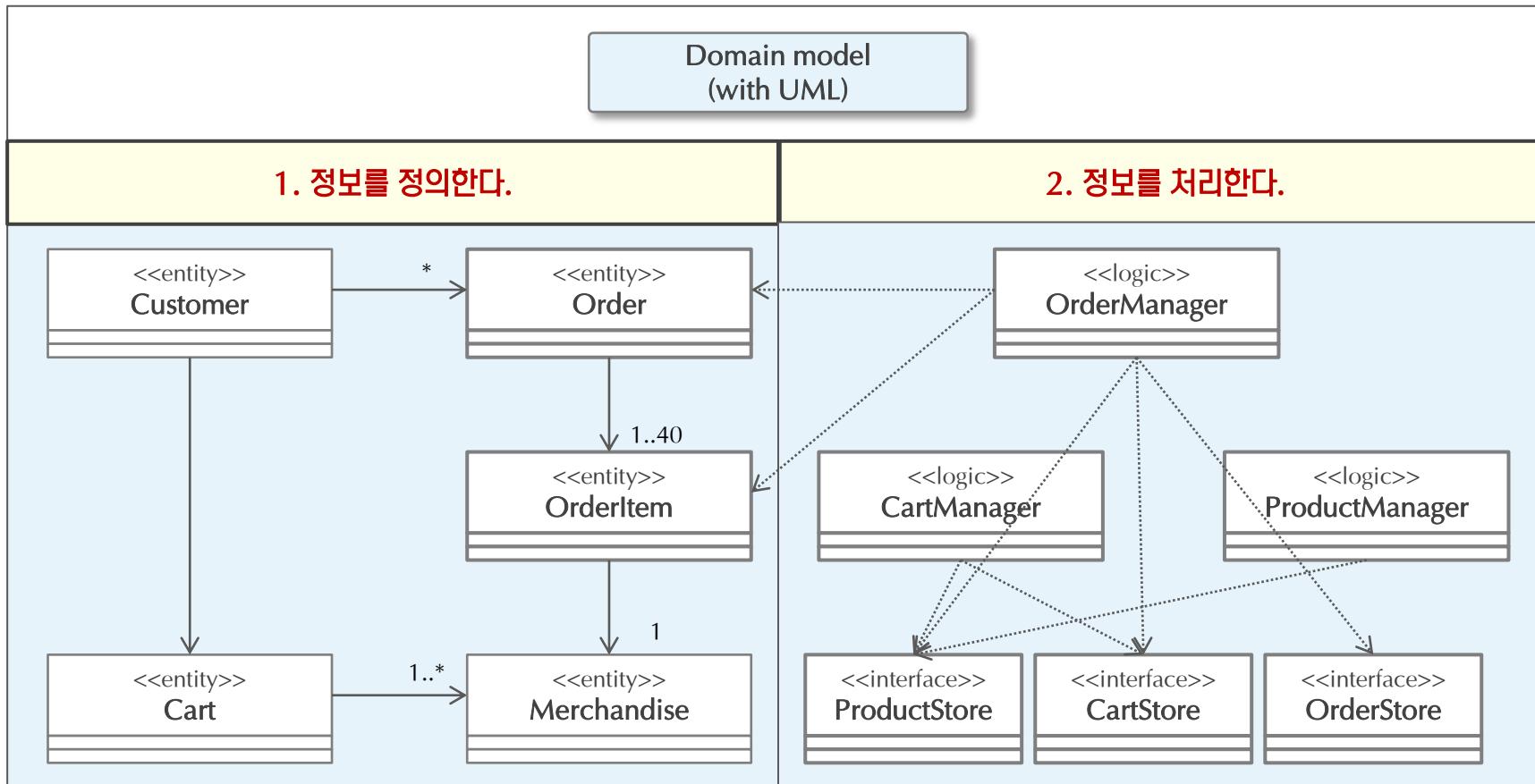
5.5 객체 모델링(9/11) – 도메인을 생각하는 과정 2

- ✓ 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.



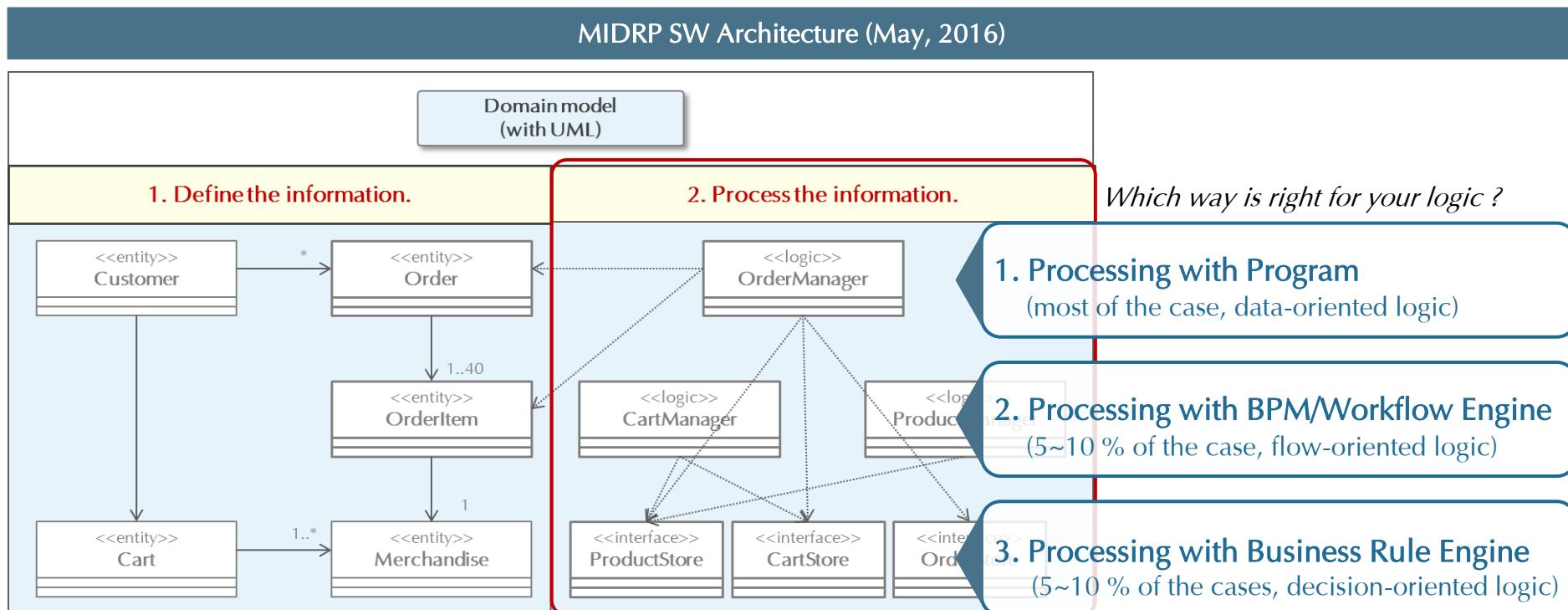
5.5 객체 모델링(10/11) – 정의와 처리 1

- ✓ 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.



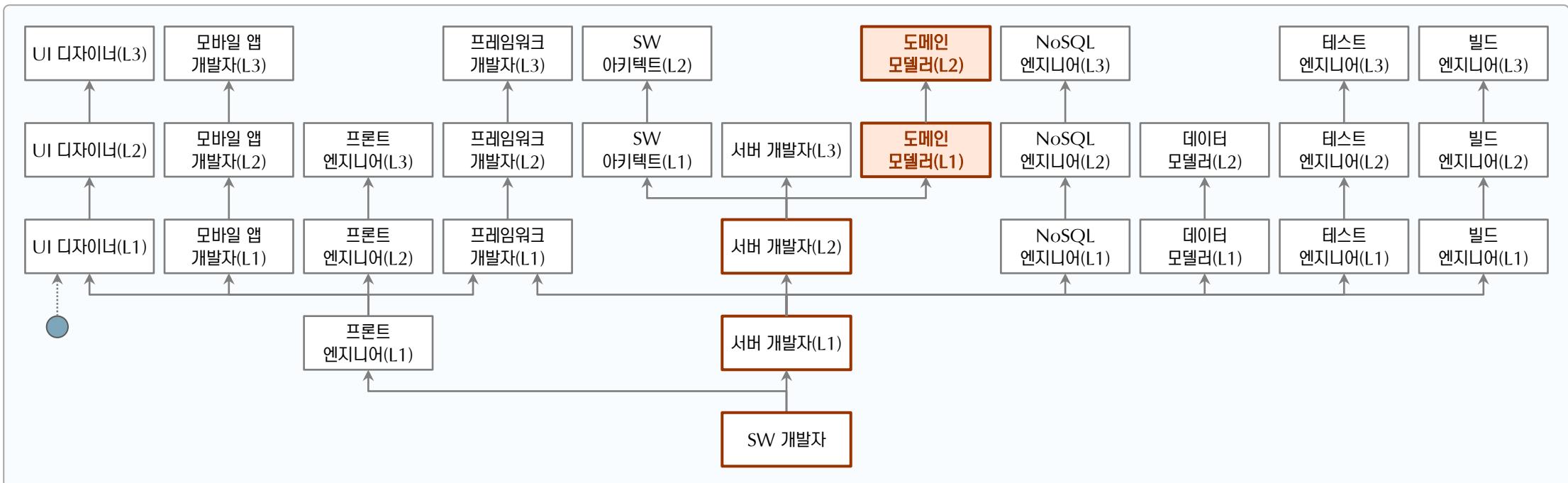
5.5 객체 모델링(11/11) – 정의와 처리 2

- ✓ 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.



5.6 도메인 모델러 1

- ✓ 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.



5.6 도메인 모델러 2

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)

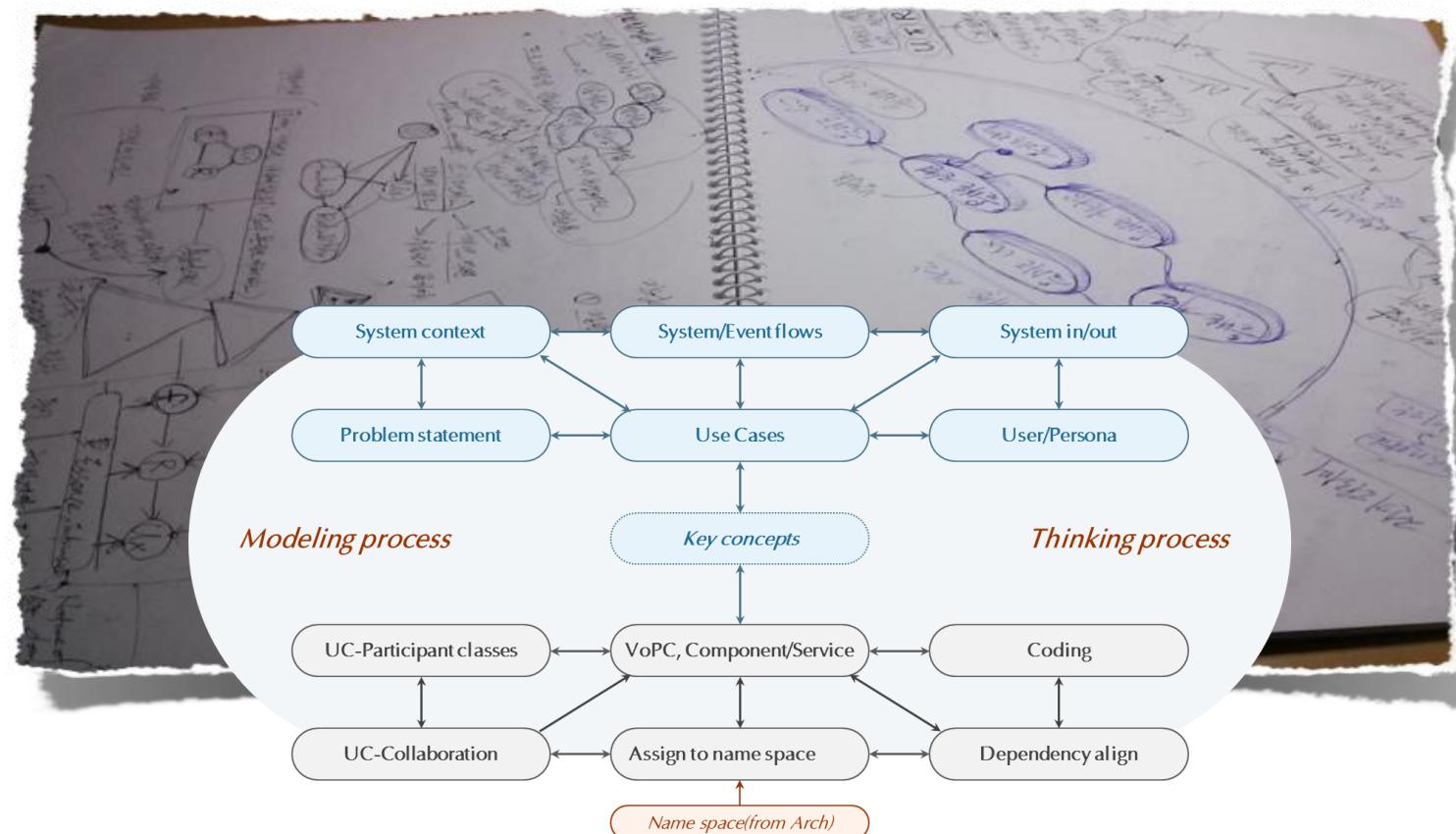
- ✓ UML과 객체 모델링
- ✓ 기술 중립적인 컴포넌트 설계
- ✓ Domain Driven Design
- ✓ 요구사항 명세(유스케이스, 사용자 스토리, 기능 목록)

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

요약

- ✓ 도메인을 제대로 이해하는 것이 모델링의 출발점입니다.
- ✓ 도메인에 대한 이해는 정보를 나열하고 정리하는 것이 아니라 문맥을 이해하는 사고 활동입니다.
- ✓ 모델링은 컨텍스트에 존재하는 모든 것들에 대한 올바른 이해와 그 이해를 바탕으로 하는 문제 해결입니다.
- ✓ 기술이 발달할 수록, 기술 모듈 보다는 도메인에 대한 이해가 점점 더 중요해지고 있습니다.



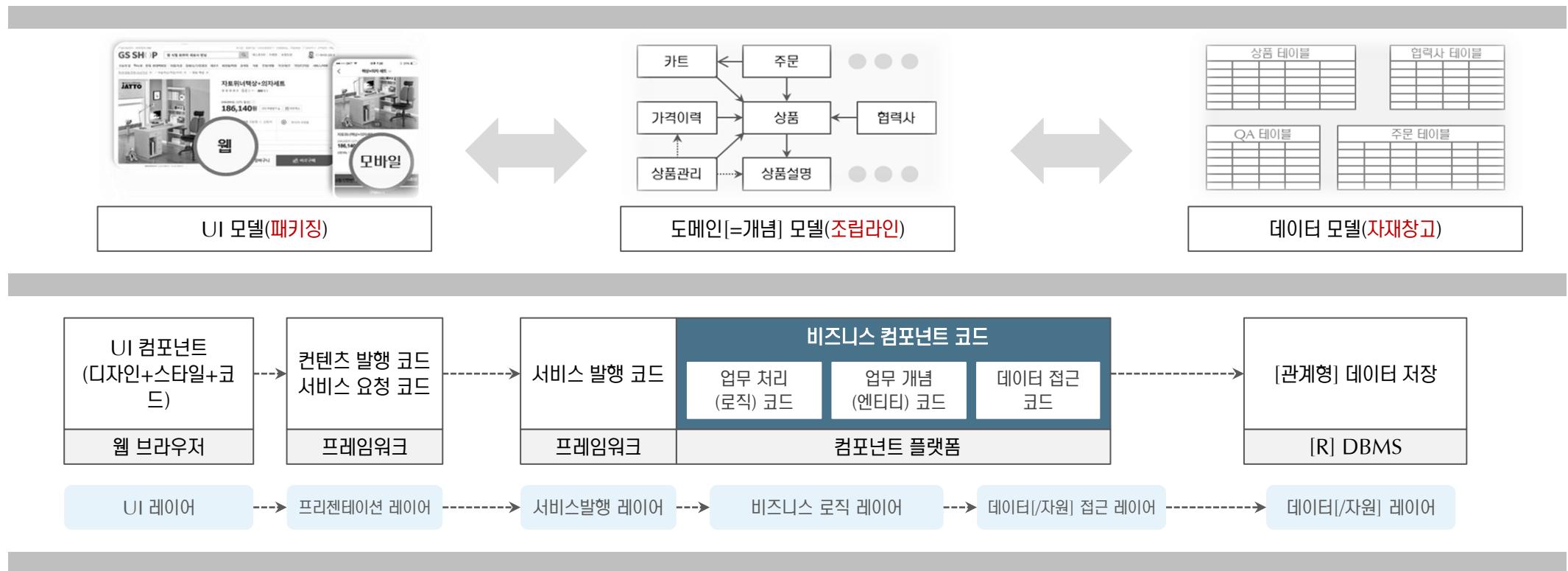


6. 컴포넌트

-
- 6.1 컴포넌트의 현재 – 도론
 - 6.2 컴포넌트 구성
 - 6.3 클래식 컴포넌트
 - 6.4 PURE 도메인 컴포넌트
 - 6.5 DDD-모델 요소
 - 6.6 DDD-도메인 객체 생명주기
 - 6.7 실습 6-1

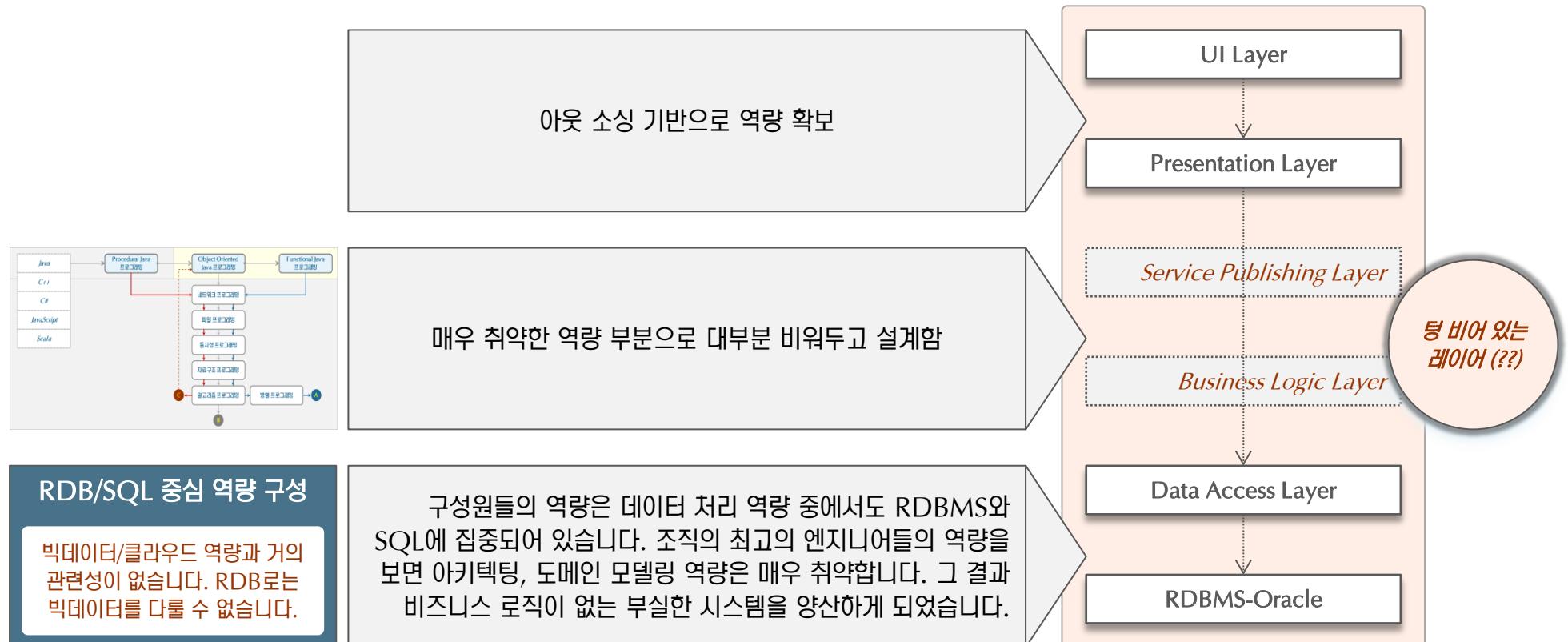
6.1 컴포넌트의 현재 – 토론 1

- ✓ [비즈니스] 컴포넌트는 정보를 다루는 시스템에서 가장 많은 부분을 차지합니다.
- ✓ 따라서 올바른 컴포넌트 설계와 구현은 올바른 시스템과 직결됩니다.
- ✓ 특히 컴포넌트는 도메인을 안고 있습니다. 시간이 흐를 수록 컴포넌트의 중요성은 더 해가고 있습니다.
- ✓ SOA, MSA 방식의 설계에서도 서비스 바로 안쪽에 컴포넌트가 자리하고 있습니다.



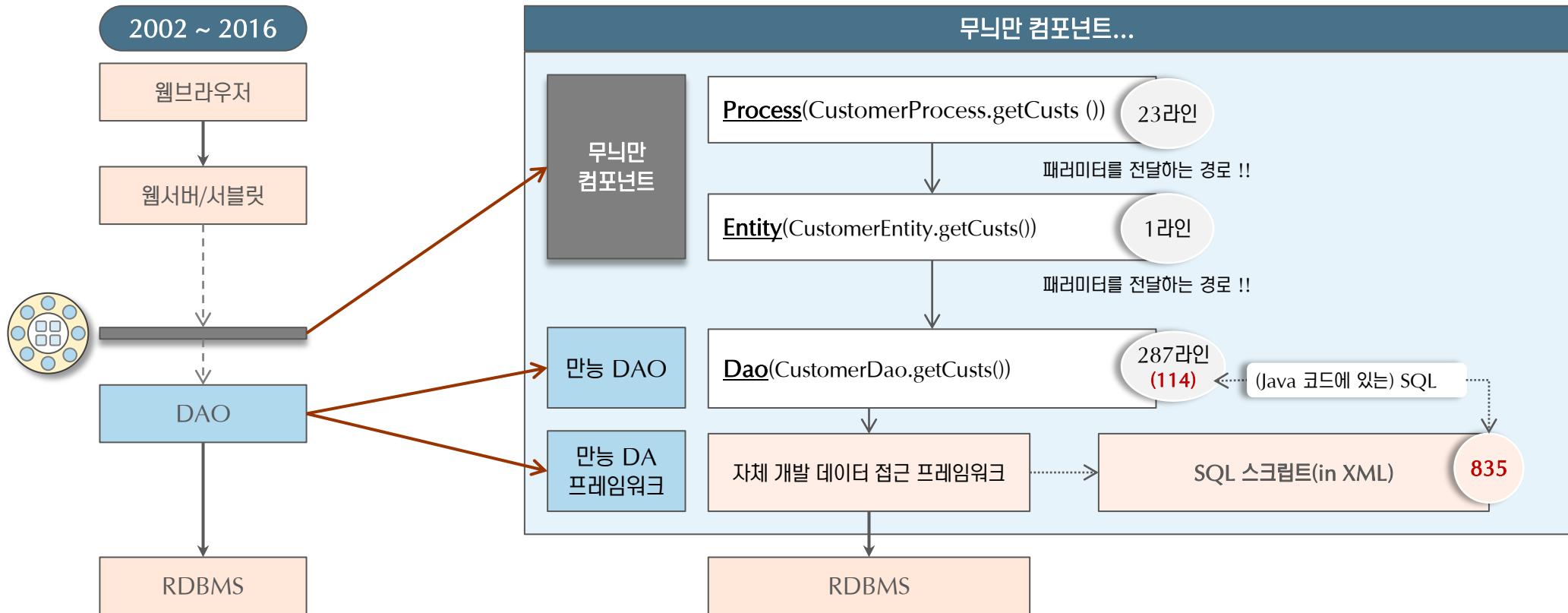
6.1 컴포넌트의 현재 – 토론 2

- ✓ Java로 작성한 비즈니스 로직을 볼 수 없거나 있더라도 아주 적은 분량의 코드가 있습니다.
- ✓ 주요 로직과 주요 정보의 관계는 데이터 모델로 표현하고 SQL로 처리를 하고 있습니다.
- ✓ 특히 SI 시장에서 개발자의 역량은 데이터에 최적화 되어 있으며, Java 프로그래밍 기회가 줄어 들고 있습니다.
- ✓ 정보 공학 기반의 데이터 모델링 강국 신화는 2000년 이후 지속되고 있습니다.



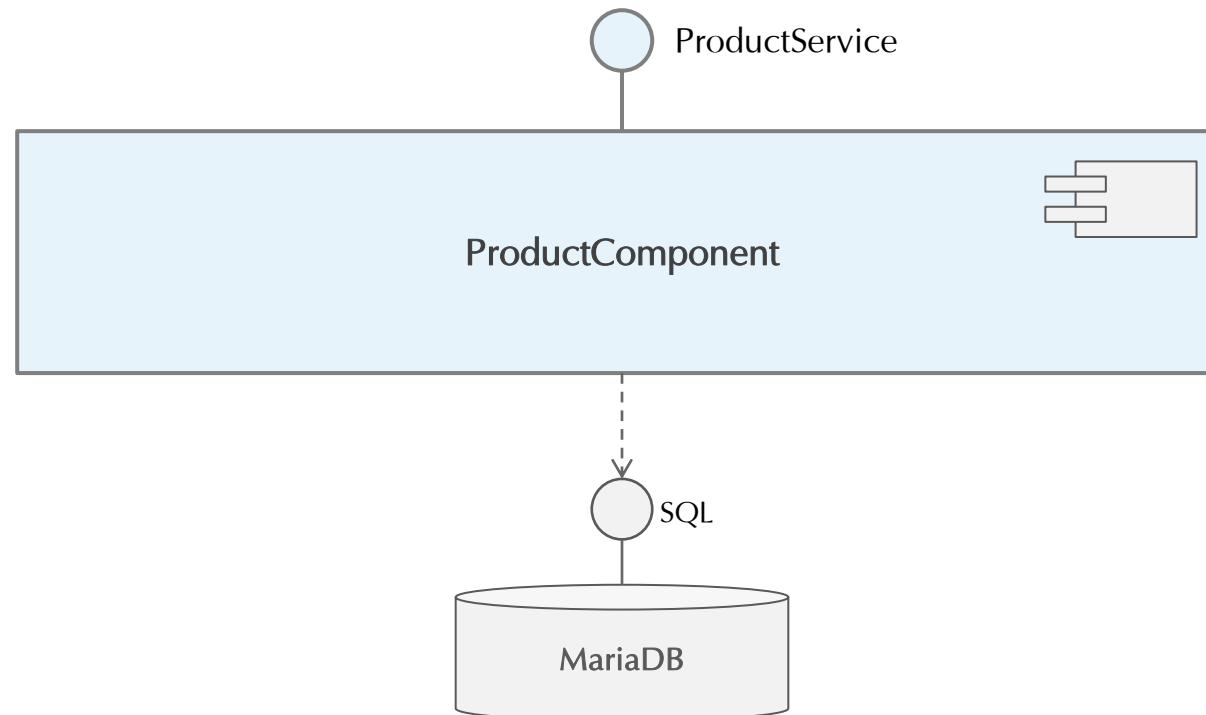
6.1 컴포넌트의 현재 – 토론 3

- ✓ 객체 모델링 역량이 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 매개변수 전달 경로로 사용할 뿐입니다.
- ✓ 객체가 주로 활동할 공간을 비워 둔 설계 유산 때문에 객체 지향 설계와 프로그래밍이 설 자리를 잃었습니다.



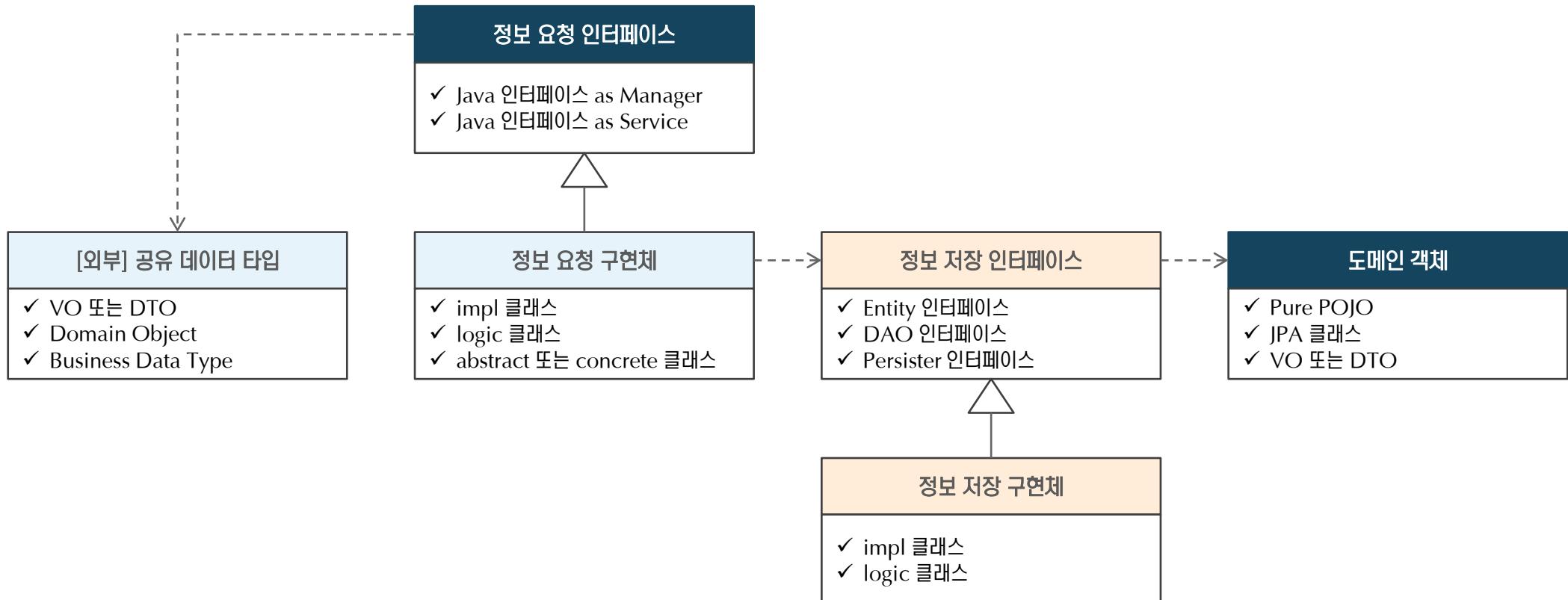
6.2 컴포넌트 구성(1/3) – 개요

- ✓ 비즈니스 컴포넌트 속에는 무엇이 있을까요?
- ✓ 정보에 해당하는 도메인 엔티티 객체, 로직에 해당하는 도메인 로직 객체, 데이터 저장소에 접근하는 객체, ...
- ✓ 여러 가지 유형의 인터페이스와 객체들이 규칙적으로 정렬되어 있을 겁니다.
- ✓ 컴포넌트 내부를 구성하는 요소(객체들의 그룹)는 어떤 것들이 있는지 생각해 봅시다.



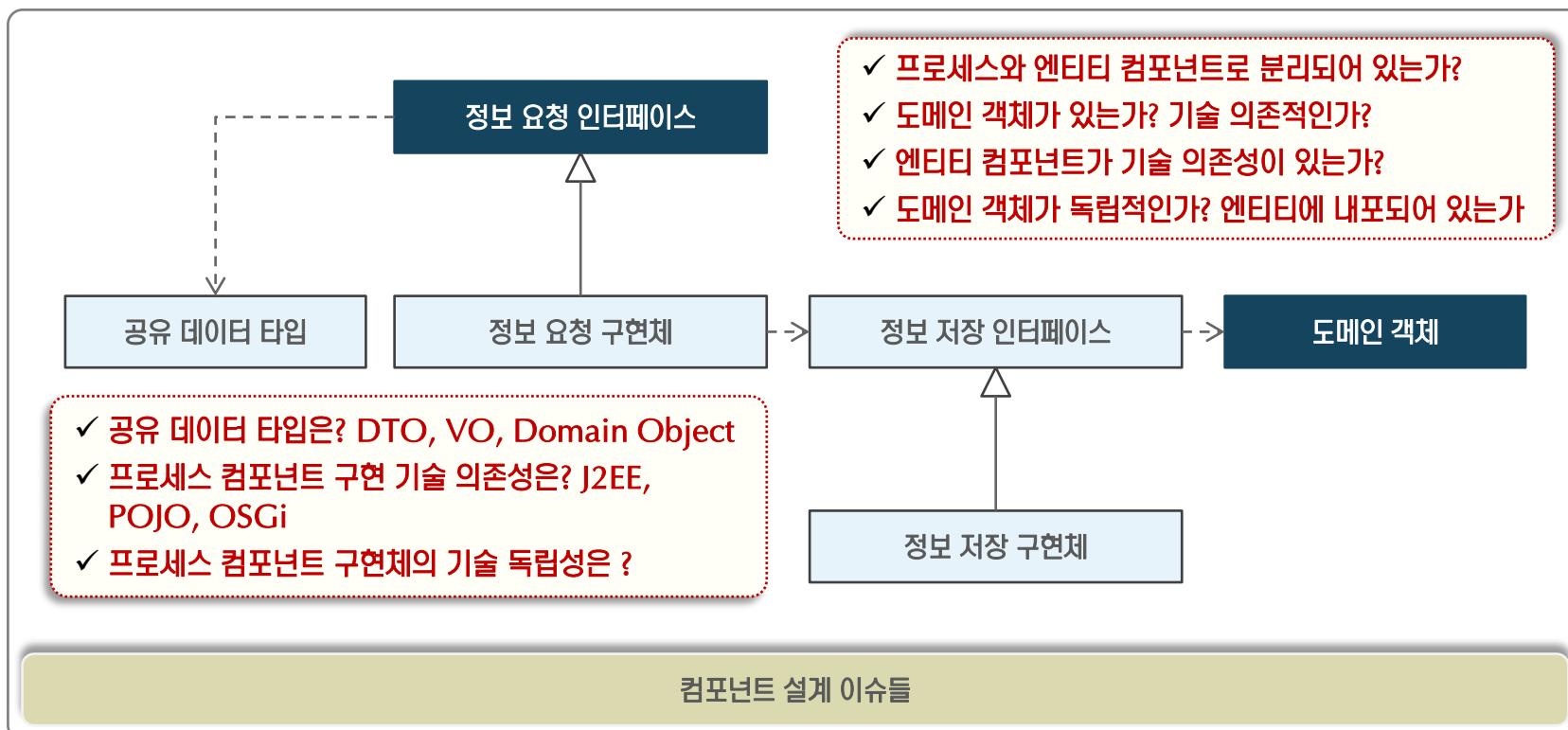
6.2 컴포넌트 구성(2/3) – 요소

- ✓ 비즈니스 컴포넌트 인터페이스는 서비스 요청 인터페이스와 정보 저장 인터페이스로 구성
- ✓ 각 요소를 구현하는 방식에 따라 컴포넌트의 특징을 결정할 수 있습니다.
- ✓ 컴포넌트 구조 설계 시, 사용하는 요소의 이름은 서로 달라서 혼란스럽긴 하지만, 결국은 아래 여섯 가지 요소 중 하나입니다.
- ✓ 가장 이상적인 컴포넌트에서부터 무늬만 컴포넌트 까지 아래의 여섯 가지 구성 요소의 조합으로 결정됩니다.



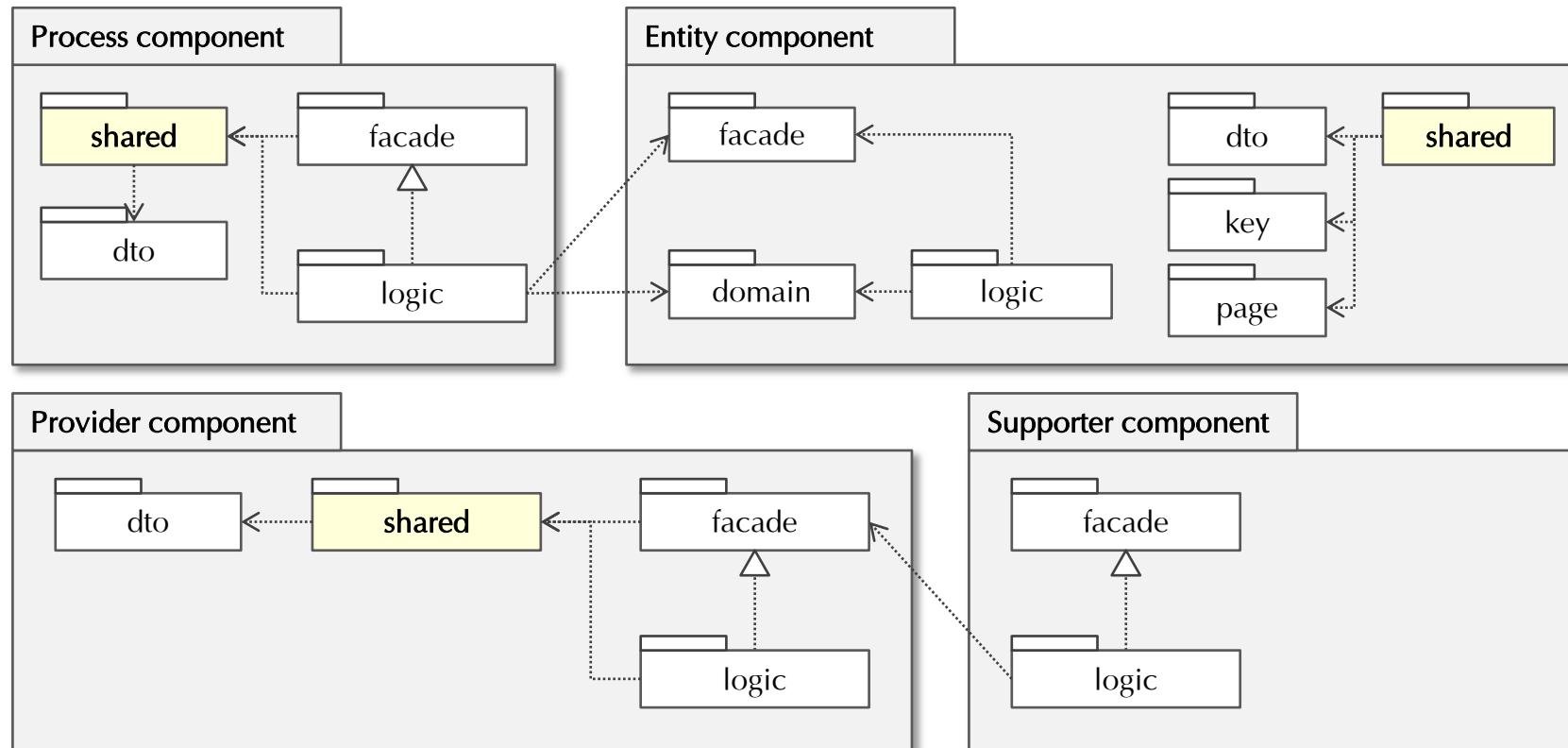
6.2 컴포넌트 구성(3/3) – 이슈

- ✓ 프로세스와 엔티티 컴포넌트로 분리되어 있는가? 분리되어 있다면, 엔티티 컴포넌트를 무엇이라 부르는가?
- ✓ 도메인 객체가 존재하는가? 존재한다면 기술 종속적인가, 아니면 PURE POJO인가?
- ✓ 도메인 객체가 엔티티 컴포넌트 내부에 있는가? 아니면 외부에 독립적으로 존재하는가?
- ✓ 프로세스 컴포넌트의 비즈니스 로직은 기술 독립적인가? 독립적이라면 추상 클래스인가, 실제 클래스인가?



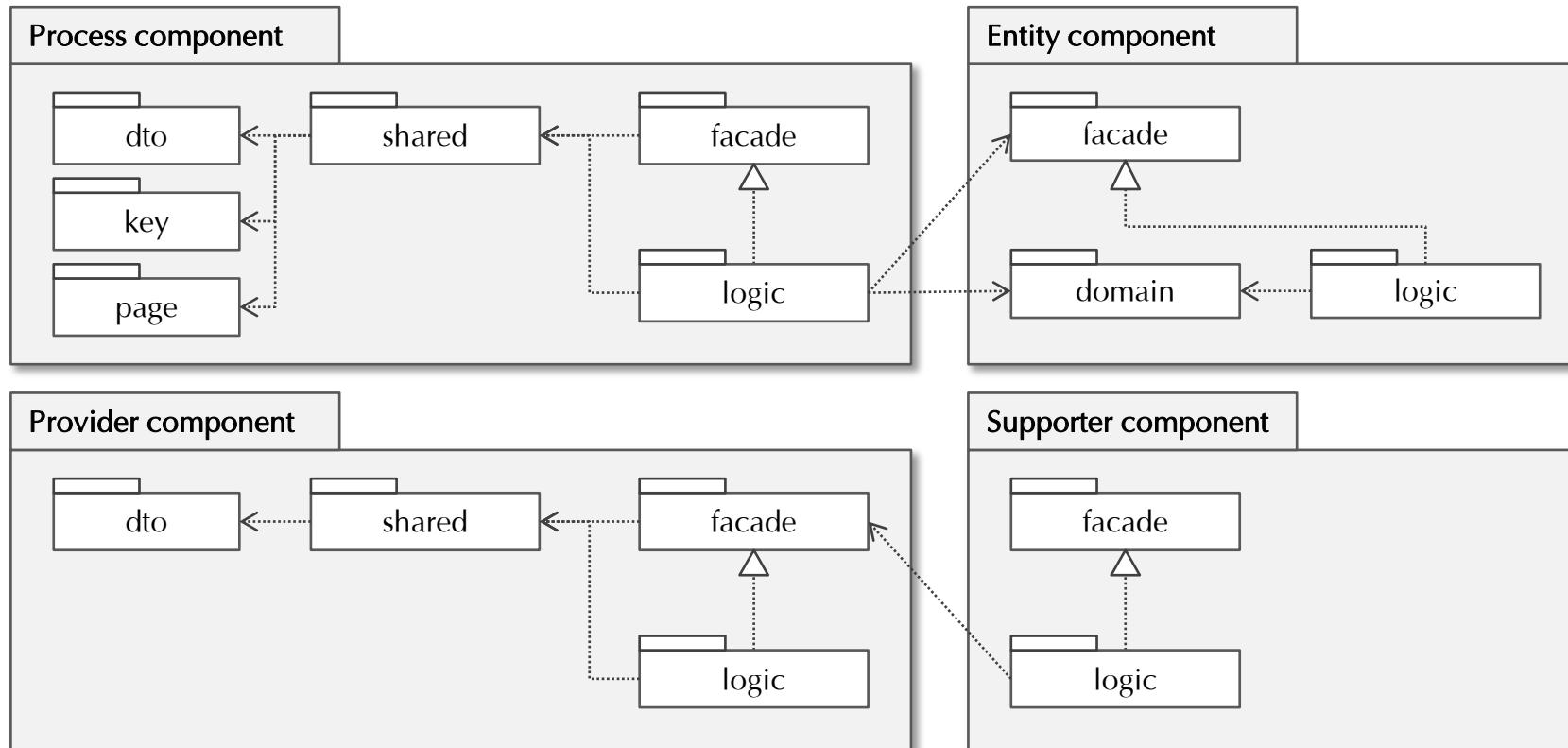
6.3 클래식 컴포넌트(1/10) – 2005년

- ✓ 컴포넌트 유형별로 서로 다른 내부구조를 가지는데 이것은 컴포넌트의 역할과 관계가 있습니다.
- ✓ 프로세스 타입 컴포넌트 내부에 있는 shared 영역을 엔티티 타입 컴포넌트에도 두어야 할지 고민이 필요합니다.
- ✓ Supporter 컴포넌트는 shared 영역이 없으며 가장 단순한 구조를 가져야 합니다.



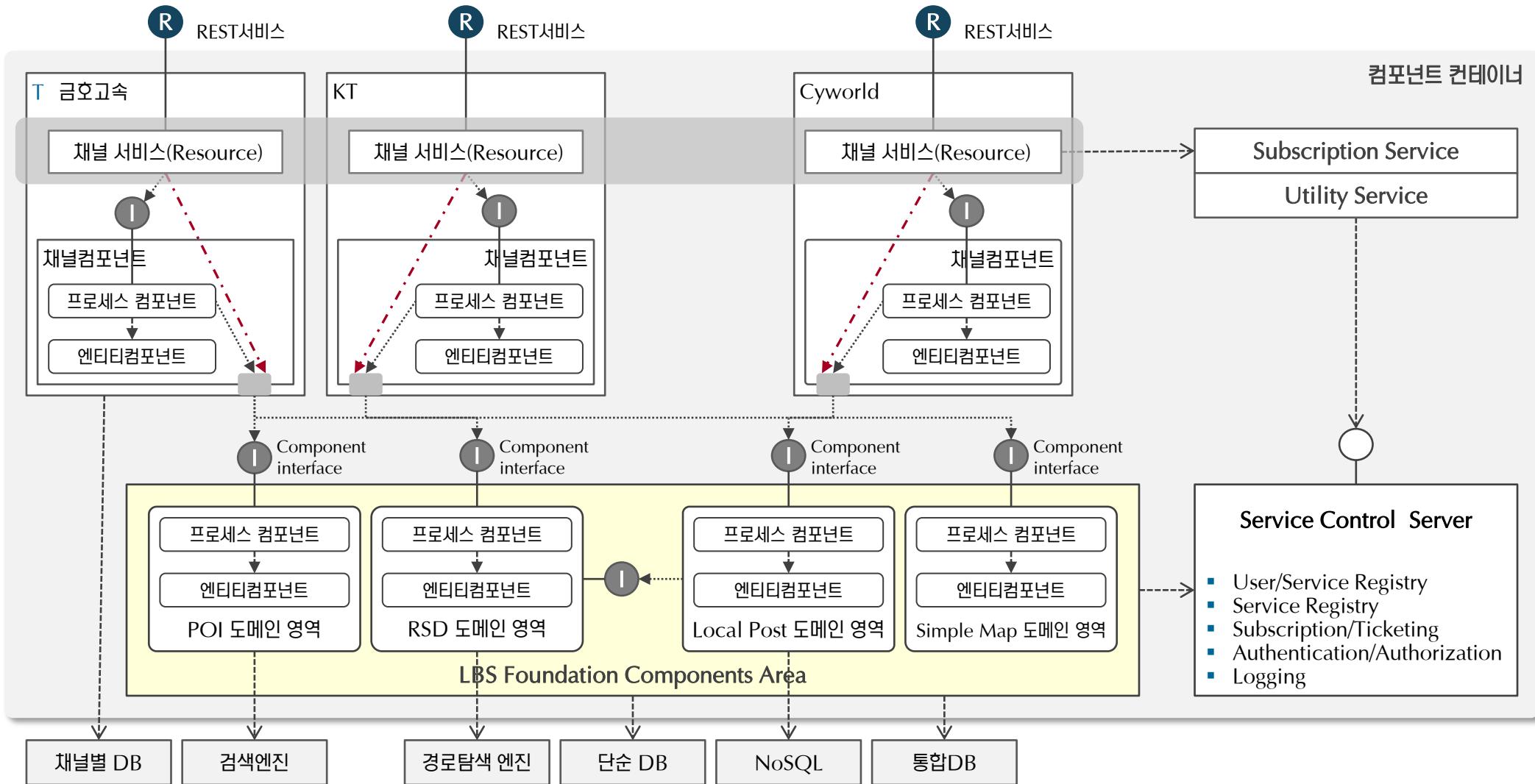
6.3 클래식 컴포넌트(2/10) – 2005년

- ✓ Entity 컴포넌트는 공유정보를 가지지 않고, 모든 공유정보는 Process 컴포넌트 내부로 두는 구조입니다.
- ✓ 하나의 프로세스 컴포넌트가 여러 Entity 컴포넌트를 사용하는 경우, Process 컴포넌트의 공유영역이 복잡합니다.
- ✓ 설계의 원칙을 준수하지만 현실적인 어려움이 발생하는 구조입니다.



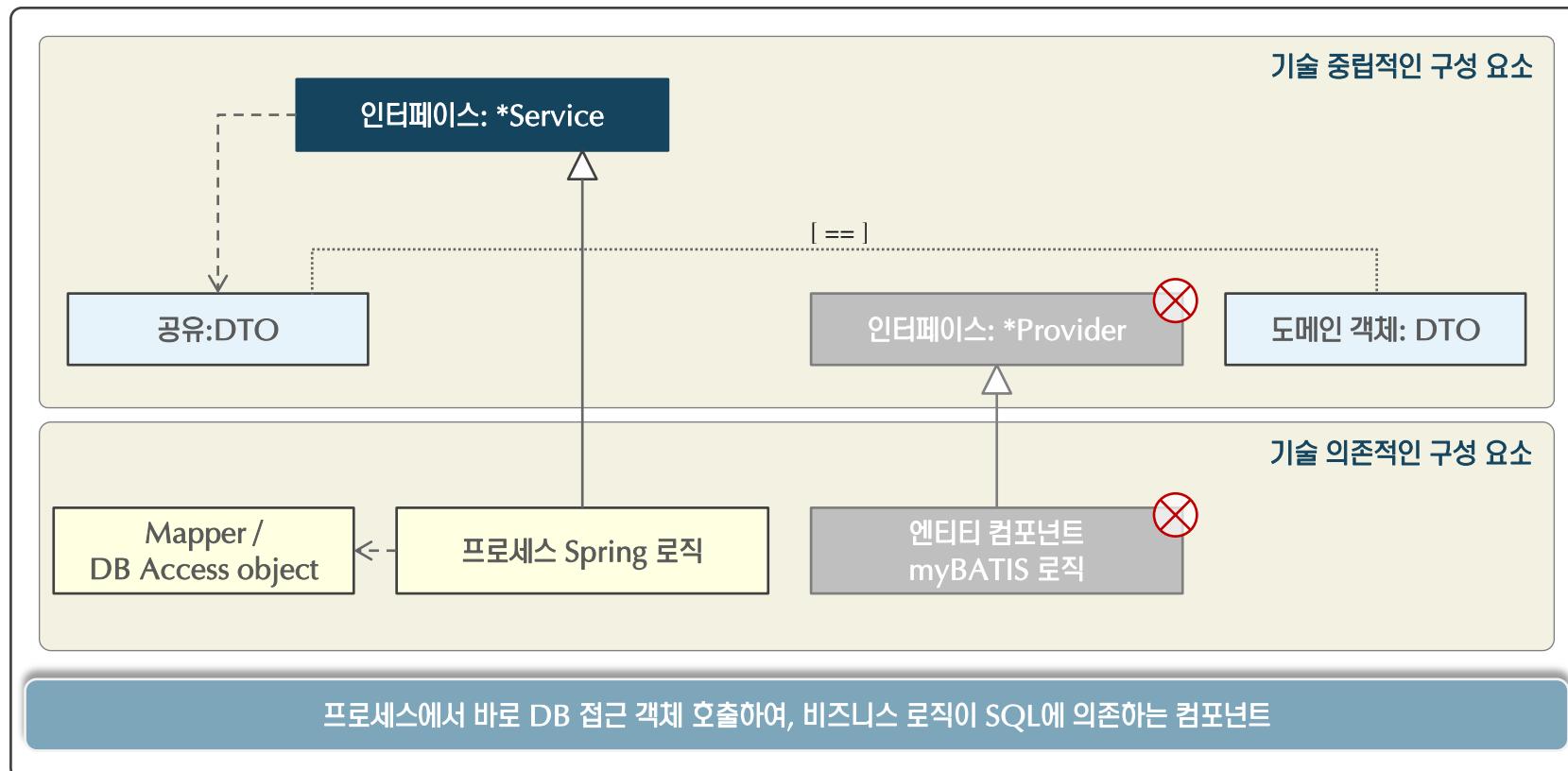
6.3 클래식 컴포넌트(3/10) – 2011년 – 채널 구조

✓ 채널 구조가 필요한 시스템의 설계 예제입니다.



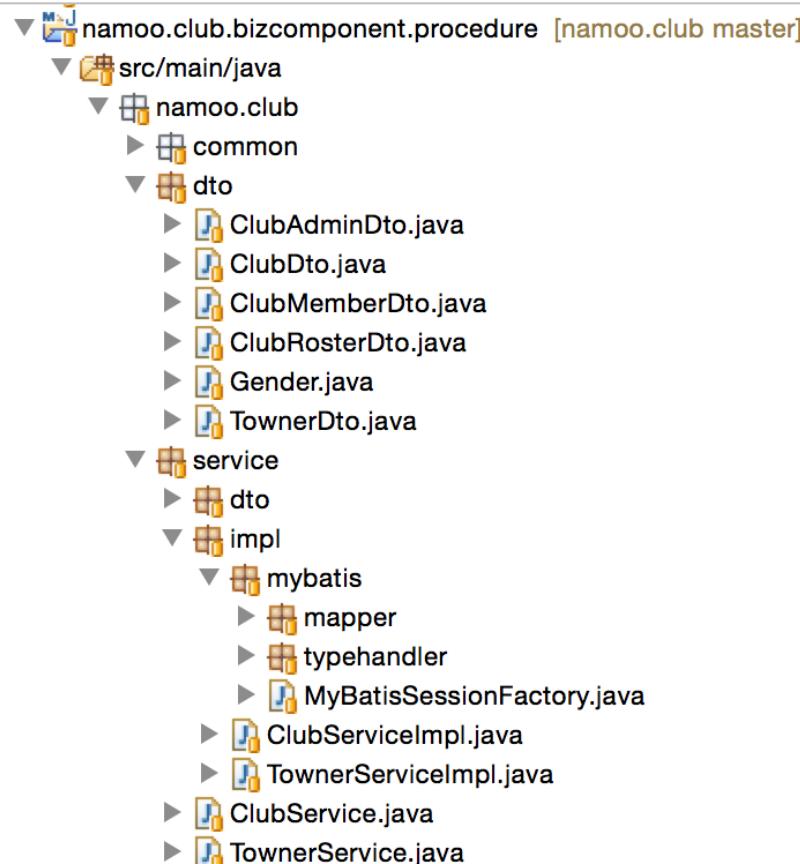
6.3 클래식 컴포넌트(4/10) – 무느만 컴포넌트(1/2)

- ✓ 컴포넌트 내부에 프로세스 로직과 엔티티 로직이 분리되어 있지 않습니다.
- ✓ 이런 구조는 기존의 절차 지향 구조를 컴포넌트 인터페이스 속으로 감춘 것에 지나지 않습니다.
- ✓ 도메인 객체가 없을 뿐만 아니라, DTO는 비즈니스 본질을 담기 보다는 UI에서 요구하는 정보를 담아 줄 뿐입니다.
- ✓ SI 프로젝트에서 이런 구조를 심심치 않게 봅니다. 어찌되었건 myBATIS는 사용을 했으니 문제 될 것이 없다고 합니다.



6.3 클래식 컴포넌트(5/10) – 무느만 컴포넌트(2/2)

- ✓ 비즈니스 로직에서 Persistence를 책임지는 객체를 직접 다루도록 설계하였습니다.
- ✓ SQL에 많이 의지 합니다.
- ✓ 저장 방식에 의존적입니다.



```
public class ClubServiceImpl implements ClubService {
    //
    @Override
    public String registerClub(ClubCDto clubCDto) throws NamooClubException {
        //
        SqlSession session = MyBatisSessionFactory.getSession();

        try {
            ClubMapper clubMapper = session.getMapper(ClubMapper.class);
            TownerMapper townnerMapper = session.getMapper(TownerMapper.class);
            ClubMemberMapper clubMemberMapper = session.getMapper(ClubMemberMapper.class);

            if (clubMapper.selectClubByName(clubCDto.getClubName()) != null) {
                throw new NamooClubException("Already existed club --> " + clubCDto.getClubName());
            }

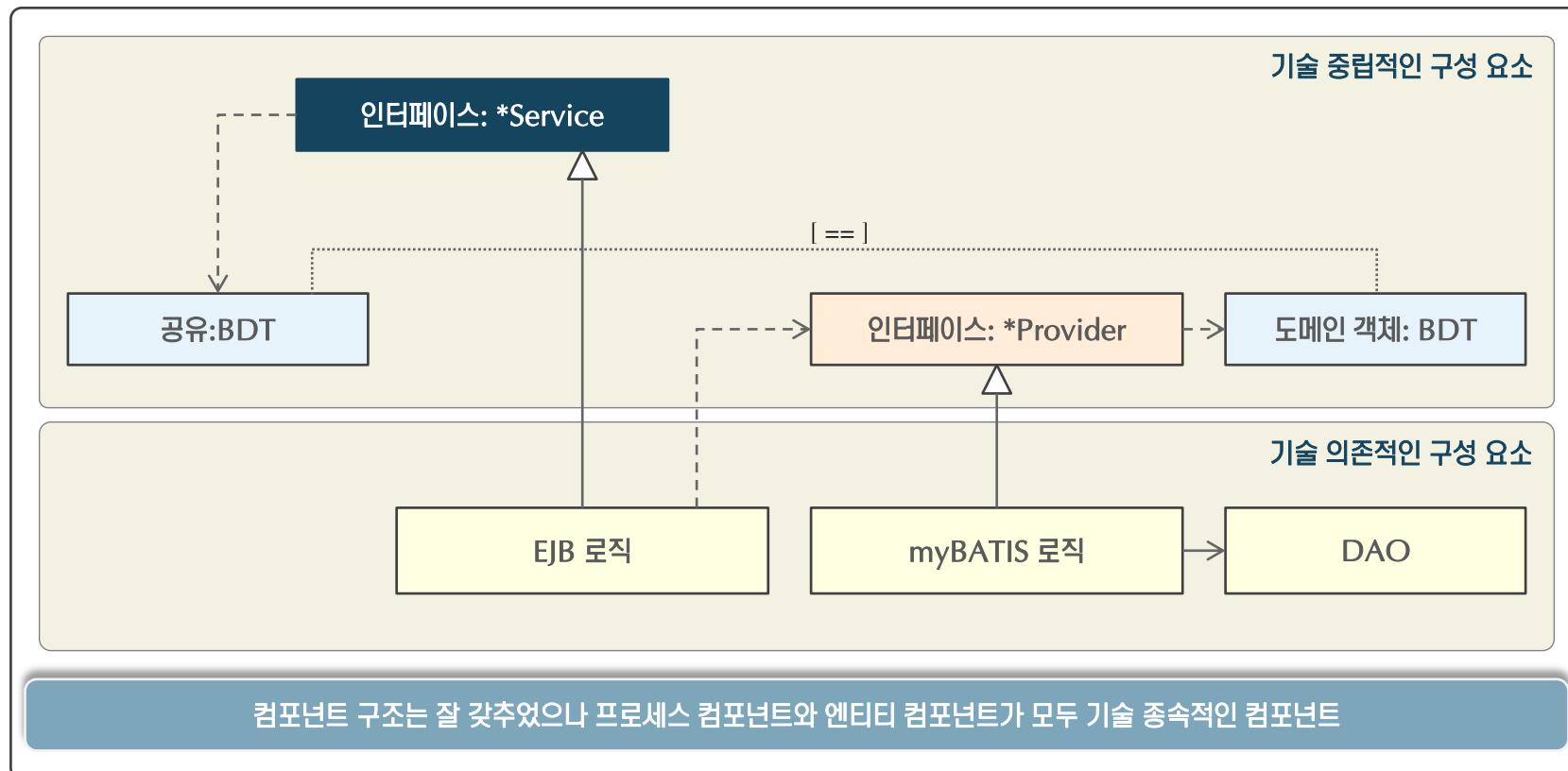
            TownerDto townner = null;
            try {
                townner = townnerMapper.selectTownerByEmail(clubCDto.getAdminEmail());
            }
            catch (EmptyResultException e) {
                throw new NamooClubException(e);
            }

            ClubDto club = clubCDto.createDomain(new ClubAdminDto(townner));
            clubMapper.insertClub(club);

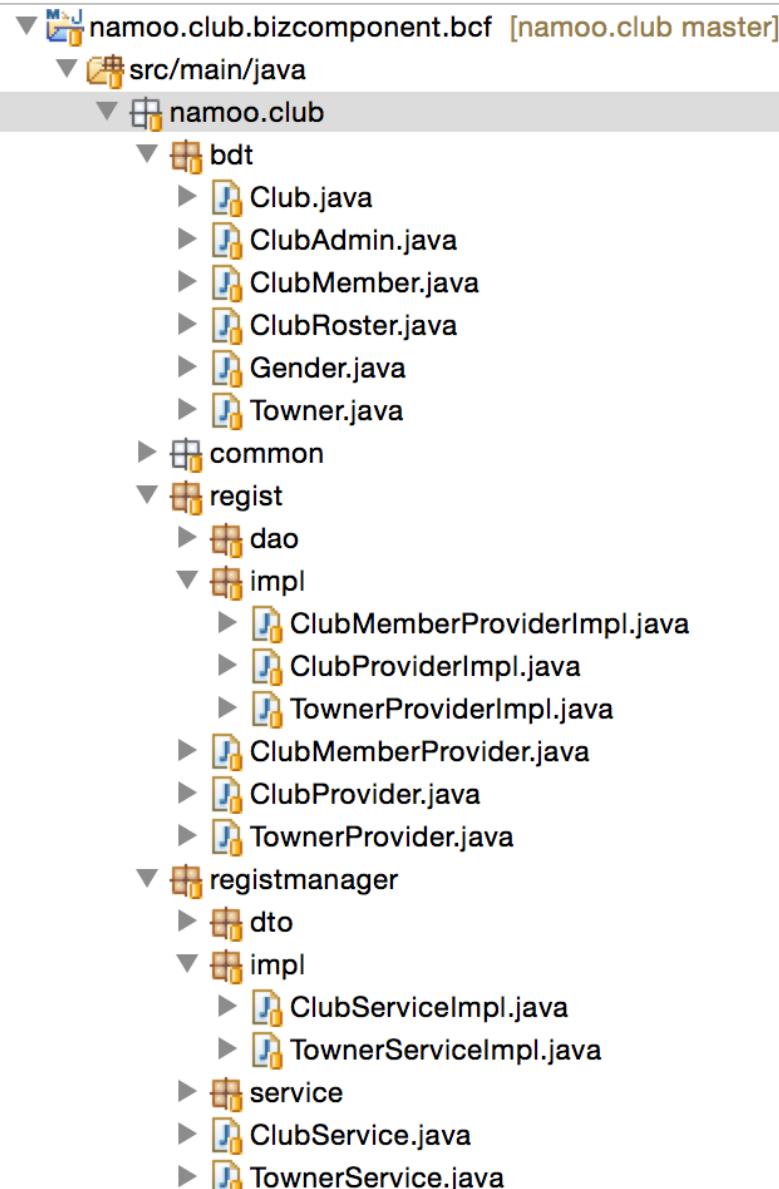
            ClubMemberDto clubMember = new ClubMemberDto(club, townner);
            clubMemberMapper.insertClubMember(club.getId(), clubMember);
            return club.getId();
        } finally {
            session.close();
        }
    }
}
```

6.3 클래식 컴포넌트(6/10) – BCF 컴포넌트(1/2)

- ✓ 도메인 개념을 표현하는 도메인 객체 관점 보다는 외부와의 공유 관점에 초점을 둔 Business Data Type을 정의합니다.
- ✓ 프로세스 컴포넌트는 플랫폼에 종속적입니다. 이런 구조를 가질 때는, 플랫폼을 변경할 가능성이 없을 때입니다.
- ✓ 엔티티 인터페이스는 중립 영역에 있으며, 구현은 지속성 기술에 종속적입니다.
- ✓ 물론 DAO를 통해서 DB에 접근함으로써, DB 변경에 따른 유연성은 가질 수 있는 구성입니다.



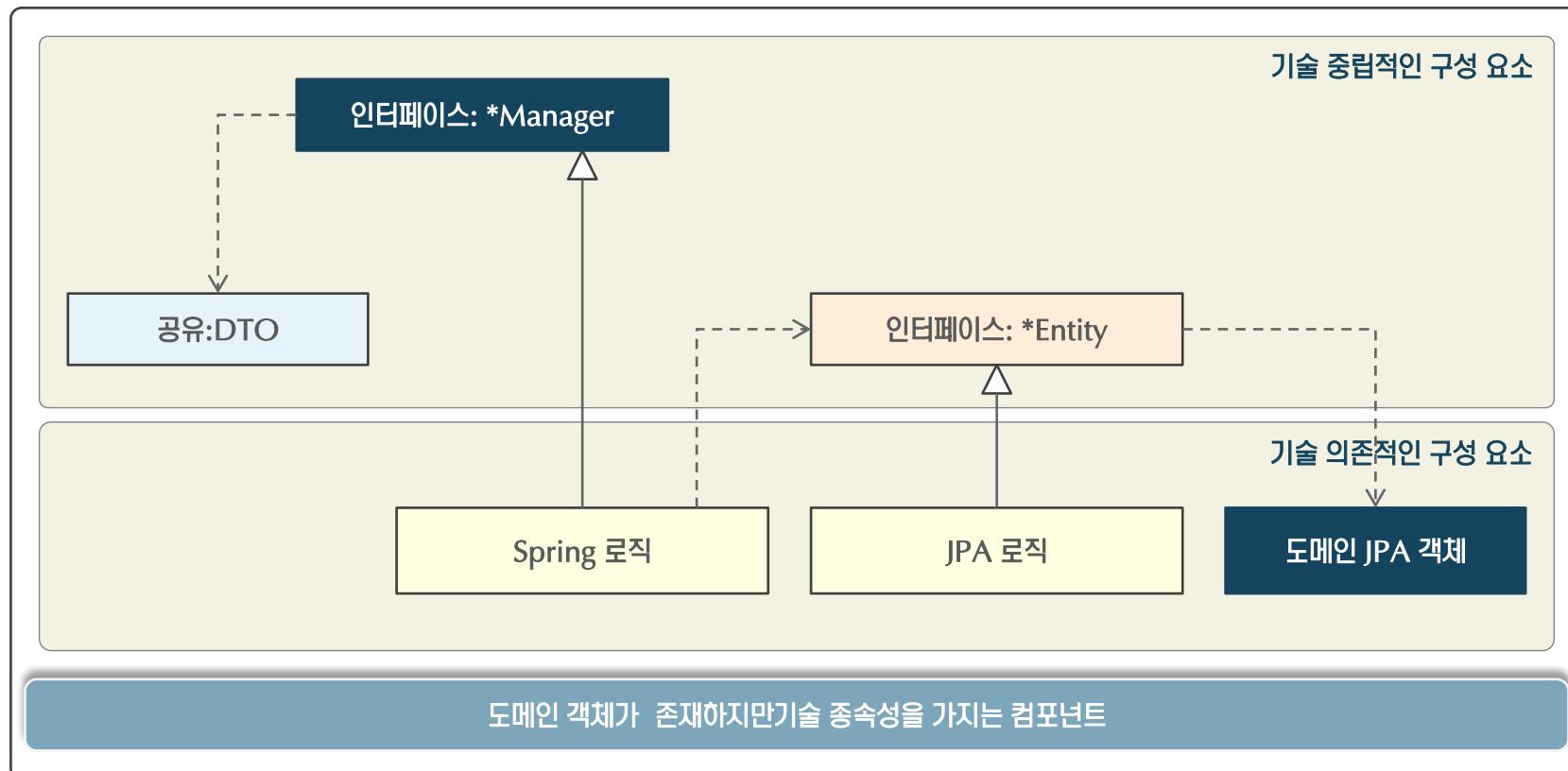
6.3 클래식 컴포넌트(7/10) – BCF 컴포넌트(2/2)



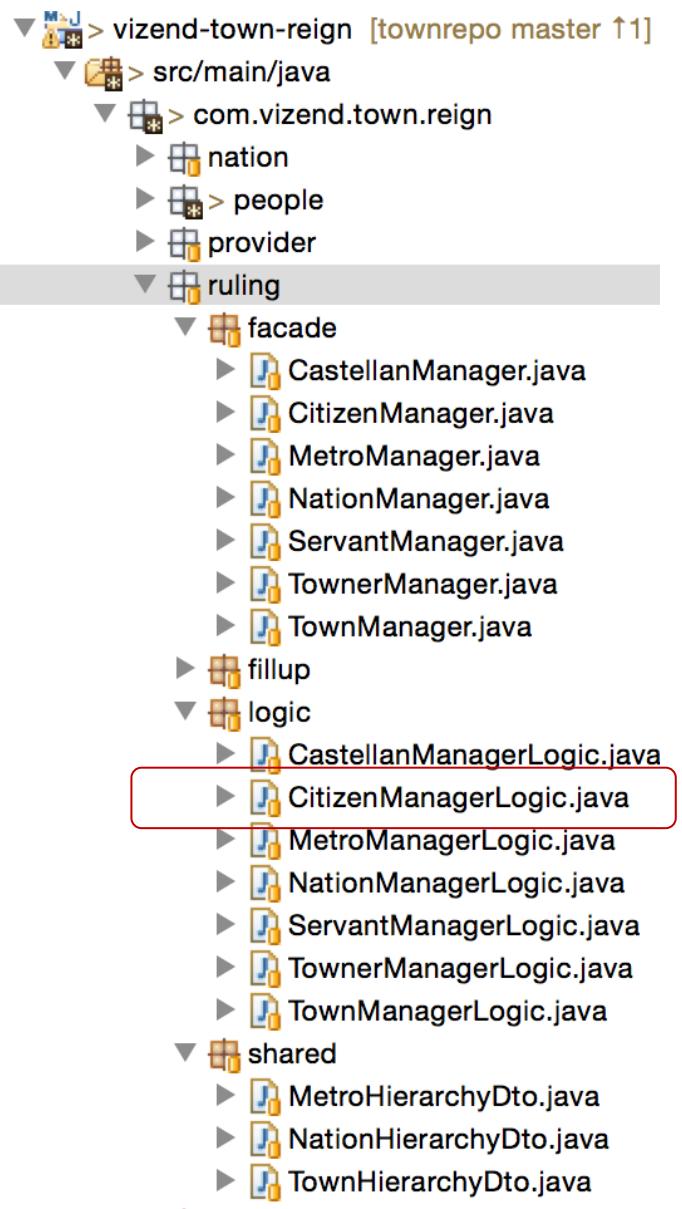
```
public class ClubServiceImpl implements ClubService {  
    //  
    private ClubProvider clubProvider;  
    private ClubMemberProvider clubMemberProvider;  
    private TownerProvider ownerProvider;  
  
    //---  
    public ClubServiceImpl() {  
        //  
        RegistManagerServiceFactory serviceFactory = RegistManagerServiceFactory.getInstance();  
  
        this.clubProvider = serviceFactory.getClubProvider();  
        this.clubMemberProvider = serviceFactory.getClubMemberProvider();  
        this.ownerProvider = serviceFactory.getTownerProvider();  
    }  
  
    //---  
    @Override  
    public String registerClub(ClubCDto clubCDto) throws NamooClubException {  
        //  
        if (clubProvider.isExistClubByName(clubCDto.getClubName())) {  
            throw new NamooClubException("Already existed club --> " + clubCDto.getClubName());  
        }  
  
        Towner owner = null;  
        try {  
            owner = ownerProvider.retrieveByEmail(clubCDto.getAdminEmail());  
        } catch (EmptyResultException e) {  
            throw new NamooClubException(e);  
        }  
  
        Club club = clubCDto.createDomain(new ClubAdmin(owner));  
        clubProvider.create(club);  
        ClubMember clubMember = new ClubMember(club, owner);  
        clubMemberProvider.addMember(club.getId(), clubMember);  
  
        return club.getId();  
    }  
}
```

6.3 클래식 컴포넌트(8/10) – 도메인 컴포넌트(1/3)

- ✓ 도메인 객체가 분명하게 존재하며, 개념을 잘 담고 있습니다. 하지만, 기술에 의존적입니다.
- ✓ 예를 들면, JPA를 사용하여 지속성을 처리할 때, 도메인 객체에 JPA annotation이 포함됩니다.
- ✓ 프로세스와 엔티티 컴포넌트 구분이 명확합니다. 프로세스 로직도 별도로 존재하지만, 기술 종속적입니다.
- ✓ 컴포넌트 플랫폼과 지속성 플랫폼 또는 솔루션을 변경하지 않을 때 유용한 컴포넌트 구조입니다.



6.3 클래식 컴포넌트(9/10) – 도메인 컴포넌트(2/3)



```
@Override  
@Transactional(propagation = Propagation.REQUIRED)  
public String registCitizen(String metroId, CitizenCDto citizenCDto) {  
    //  
    // validation  
    if(citizenEntity.existByLoginId(citizenCDto.getLoginId())){  
        throw new VizendTownException("loginId already exists! >> loginId : "+citizenCDto.getLoginId());  
    }  
  
    Castellan castellan = null;  
  
    if (!castellanEntity.existByAuthEmail(citizenCDto.getEmail())) {  
        Nation nation = nationEntity.retrieveByMetroId(metroId);  
        castellan = new Castellan(  
            nation.getId(),  
            townIdGenerator.generateCastellanId(nation.getId()),  
            citizenCDto.getLoginName(),  
            citizenCDto.getEmail());  
        castellanEntity.create(castellan);  
    } else {  
        castellan = castellanEntity.retrieveByAuthEmail(citizenCDto.getEmail());  
    }  
  
    String citizenId = townIdGenerator.generateCitizenId(metroId);  
    Citizen citizen = citizenCDto.createDomain(castellan, metroId, citizenId);  
    citizen.setPassword(EncryptUtils.encrypt(citizenCDto.getPassword()));  
  
    return citizenEntity.create(citizen);  
}
```

6.3 클래식 컴포넌트(10/10) – 도메인 컴포넌트(3/3)

The image shows a Java code editor with the following structure:

- File Structure:**
 - vizend-town-reign [townrepo master ↑1]
 - src/main/java
 - com.vizend.town.reign
 - nation
 - people
 - domain
 - Castellan.java
 - Citizen.java
 - CivicAdmin.java
 - PublicServant.java
 - StatusCode.java
 - Towner.java
 - domainbackup
 - facade
 - CastellanEntity.java
 - CitizenEntity.java
 - CivicAdminEntity.java
 - ServantEntity.java
 - TownerEntity.java
 - logic
 - CastellanEntityLogic.java
 - CitizenEntityLogic.java
 - CivicAdminEntityLogic.java
 - ServantEntityLogic.java
 - TownerEntityLogic.java
 - shared
 - dto
 - CastellanCDto.java
 - CastellanRDto.java
 - CitizenCDto.java
 - CitizenRDto.java
 - CivicAdminCDto.java
 - CivicAdminRDto.java
 - ServantCDto.java
 - ServantRDto.java
 - TownerRDto.java
 - helper

```
@Repository
public class CitizenEntityLogic implements CitizenEntity {
    //
    @PersistenceContext
    private EntityManager em;

    //-----
    // methods

    @Override
    public String create(Citizen citizen) {
        //
        em.persist(citizen);
        return citizen.getId();
    }

    @Override
    public void createBackup(CitizenBackup citizenBackup) {
        //
        em.persist(citizenBackup);
    }

    @Override
    public Citizen retrieve(String citizenId) {
        //

        String queryStr = "FROM Citizen WHERE id = :citizenId";
        Query query = em.createQuery(queryStr);
        query.setParameter("citizenId", citizenId);

        return SafeQuery.getUniqueResult(query);
    }
}
```

```
@Entity
public class Citizen {
    //
    @Id
    @Column(name = "CITIZEN_ID")
    private String id; // MetroId-ABC1

    @Column(nullable = false)
    private String metroId;

    @Column(nullable = false)
    private String loginId;

    @Column(nullable = true)
    private String loginName;

    @Column(nullable = false)
    private String password;

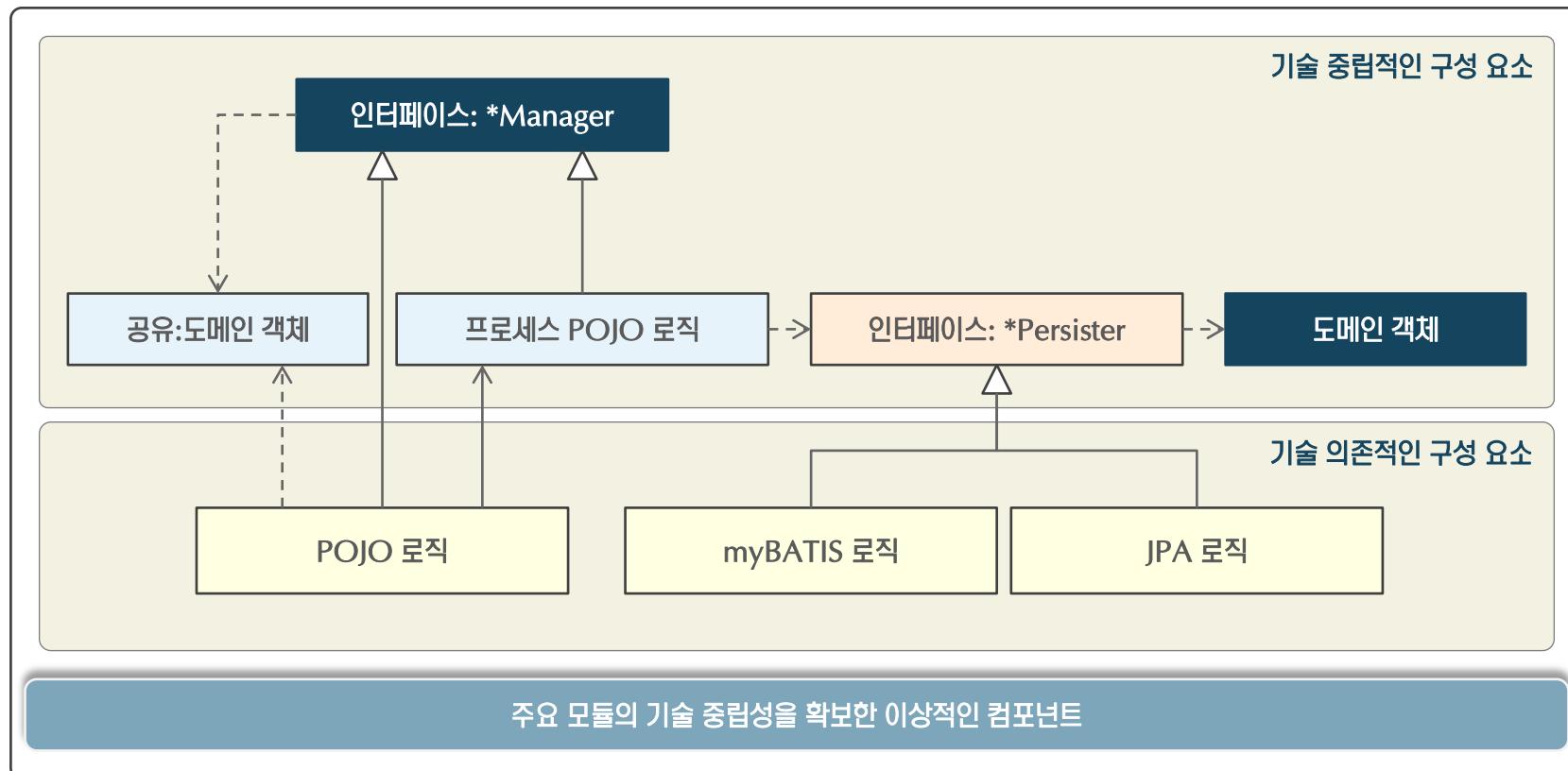
    @Column(nullable = true)
    private String email;

    @Column(nullable = true)
    private String phoneNumber;

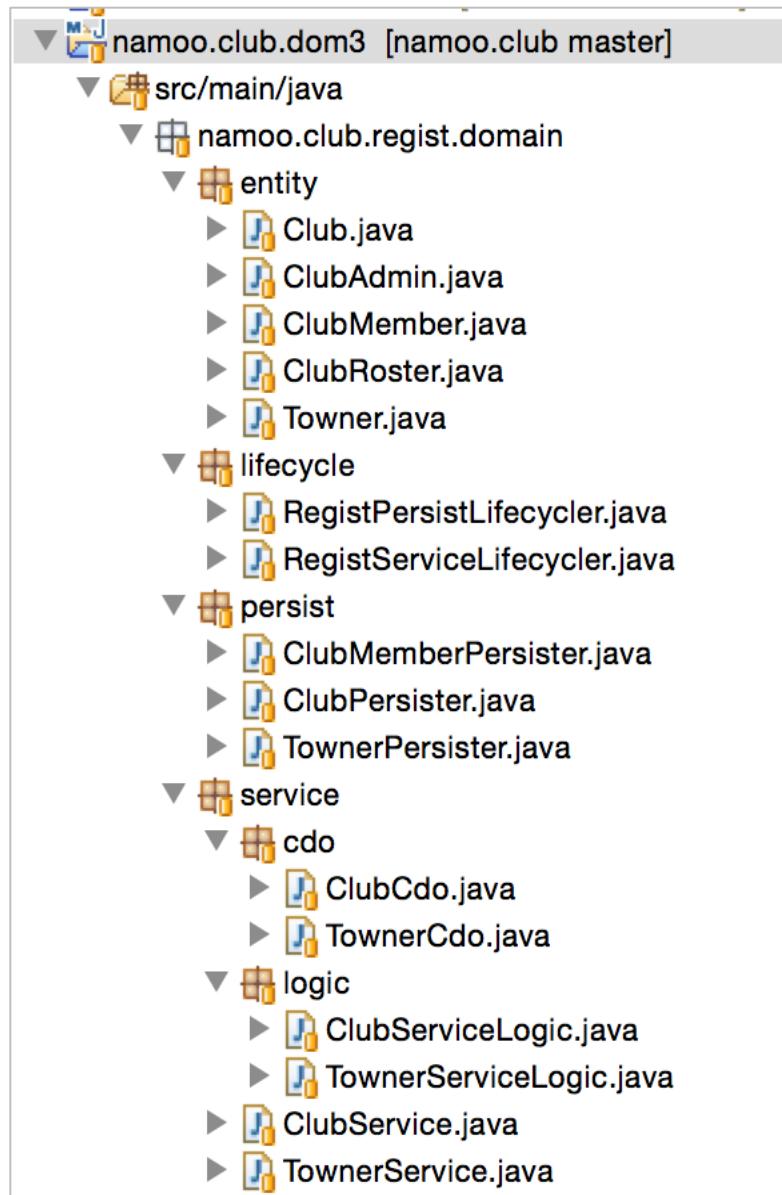
    @Column(nullable = true)
    private String photoId;
}
```

6.4 PURE 도메인 컴포넌트(1/9)

- ✓ 처리 로직이 “프로세스 POJO 로직”으로 구성되어, 어떤 플랫폼에서도 로직을 사용할 수 있습니다.
- ✓ 지속성 처리가 외부의 구현체로 설계하여, 어떤 지속성 구현체가 오든 기술 중립적인 구성 요소는 영향을 받지 않습니다.
- ✓ 가장 이상적인 형태의 컴포넌트 구성이며, 도메인 객체는 지속성이 아닌 외부로 공유되는 구조입니다.
- ✓ 도메인 객체 모델을 오픈하여 공유함으로써 DDD의 “Ubiquitous Language”로 사용합니다.



6.4 PURE 도메인 컴포넌트(2/9)



```
public class ClubServiceLogic implements ClubService {
    //
    private ClubPersister clubPersister;
    private ClubMemberPersister clubMemberPersister;
    private TownerPersister townnerPersister;

    /**
     * @param RegistPersistLifecycle lifecycler
     */
    public ClubServiceLogic(RegistPersistLifecycle lifecycler) {
        //
        this.clubPersister = lifecycler.getClubPersister();
        this.clubMemberPersister = lifecycler.getClubMemberPersister();
        this.townnerPersister = lifecycler.getTownerPersister();
    }

    /**
     * @Override
     */
    public String registerClub(ClubCdo clubCdo) throws NamooClubException {
        //
        if (clubPersister.isExistClubByName(clubCdo.getClubName())) {
            throw new NamooClubException("Already existed club --> " + clubCdo.getClubName());
        }

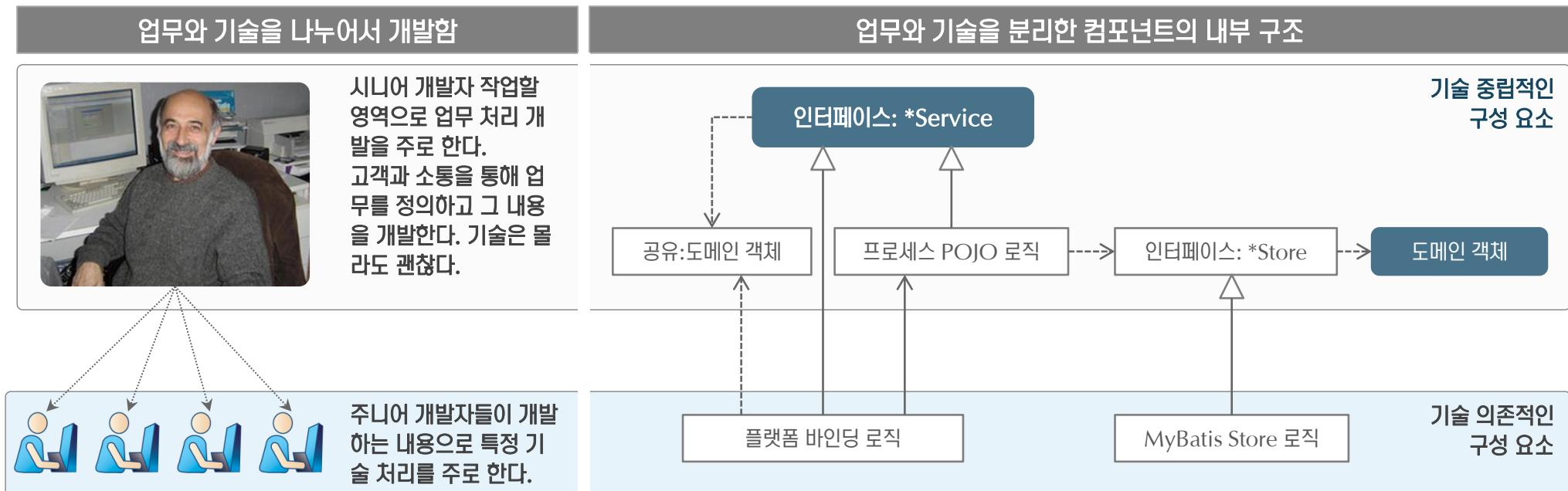
        Towner townner = null;
        try {
            townner = townnerPersister.retrieveByEmail(clubCdo.getAdminEmail());
        } catch (EmptyResultException e) {
            throw new NamooClubException(e);
        }

        Club club = clubCdo.createDomain(new ClubAdmin(townner));
        clubPersister.create(club);
        ClubMember clubMember = new ClubMember(club, townner);
        clubMemberPersister.addMember(club.getId(), clubMember);

        return club.getId();
    }
}
```

6.4 PURE 도메인 컴포넌트(3/9) – 업무와 기술 분리

- ✓ 컴포넌트 설계를 통해서 개발자의 역할 배정을 효율적으로 할 수 있습니다. ← 멋지고 경제적인 설계 아이디어
- ✓ 컴포넌트를 업무를 다루는 모듈과 기술에 종속적인 모듈로 나눈다면, 의외의 아주 큰 소득(예산/인력)이 있습니다.
- ✓ 시니어 개발자는 새로운 기술에 약해서 개발에 어려움을 겪고, 주니어 개발자는 업무에 약해서 개발에 어려움을 겪습니다.
- ✓ 결국 시니어 개발자는 개발에서 손을 떼고, 업무를 잘 모르는 주니어 개발자가 개발을 주도합니다. ← 품질 문제 발생

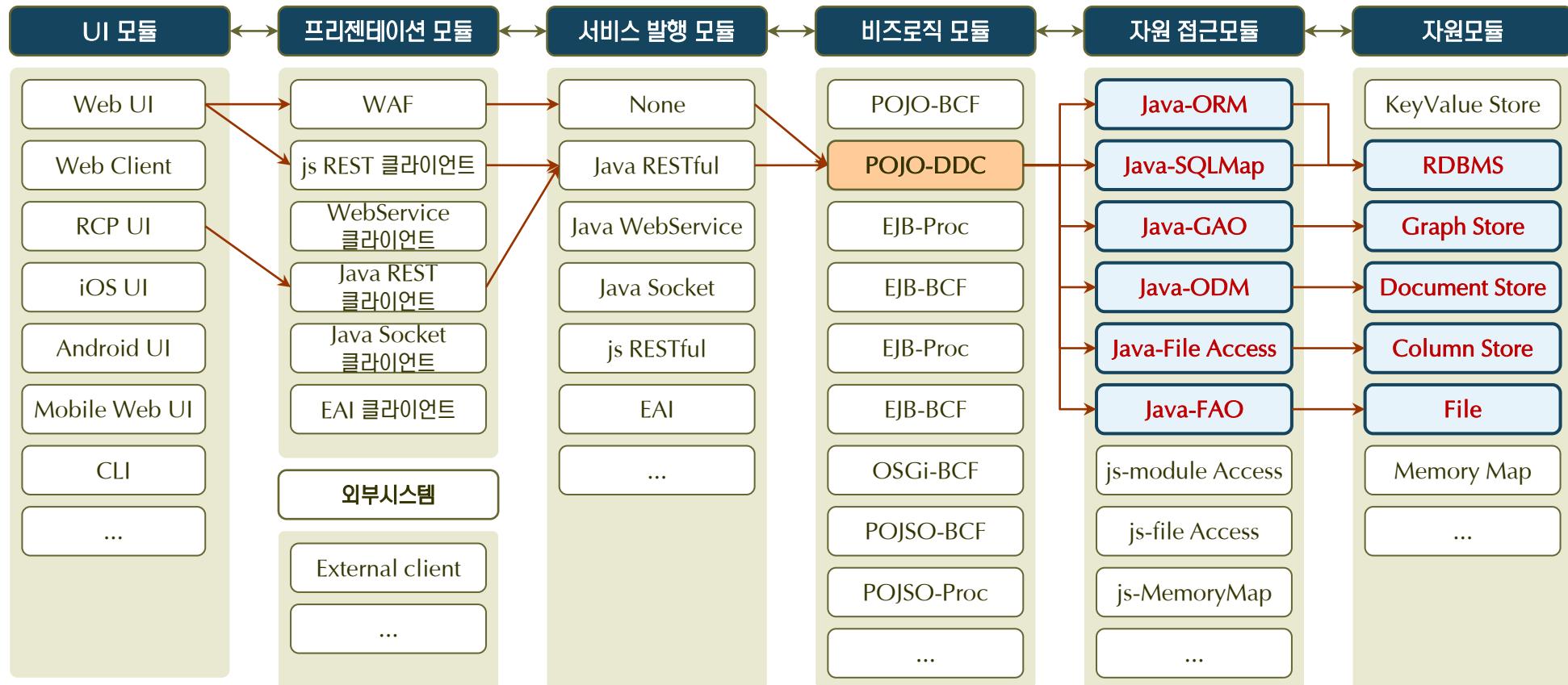


정확한 업무 이해를 바탕으로 하는 개발과 세밀한 기술을 바탕으로 하는 개발을 나누어서 진행할 수 있습니다.

업무와 기술 두 가지 모두에서 비약적인 품질 향상을 얻을 수 있고, 엔지니어를 활용도를 높일 수 있습니다.

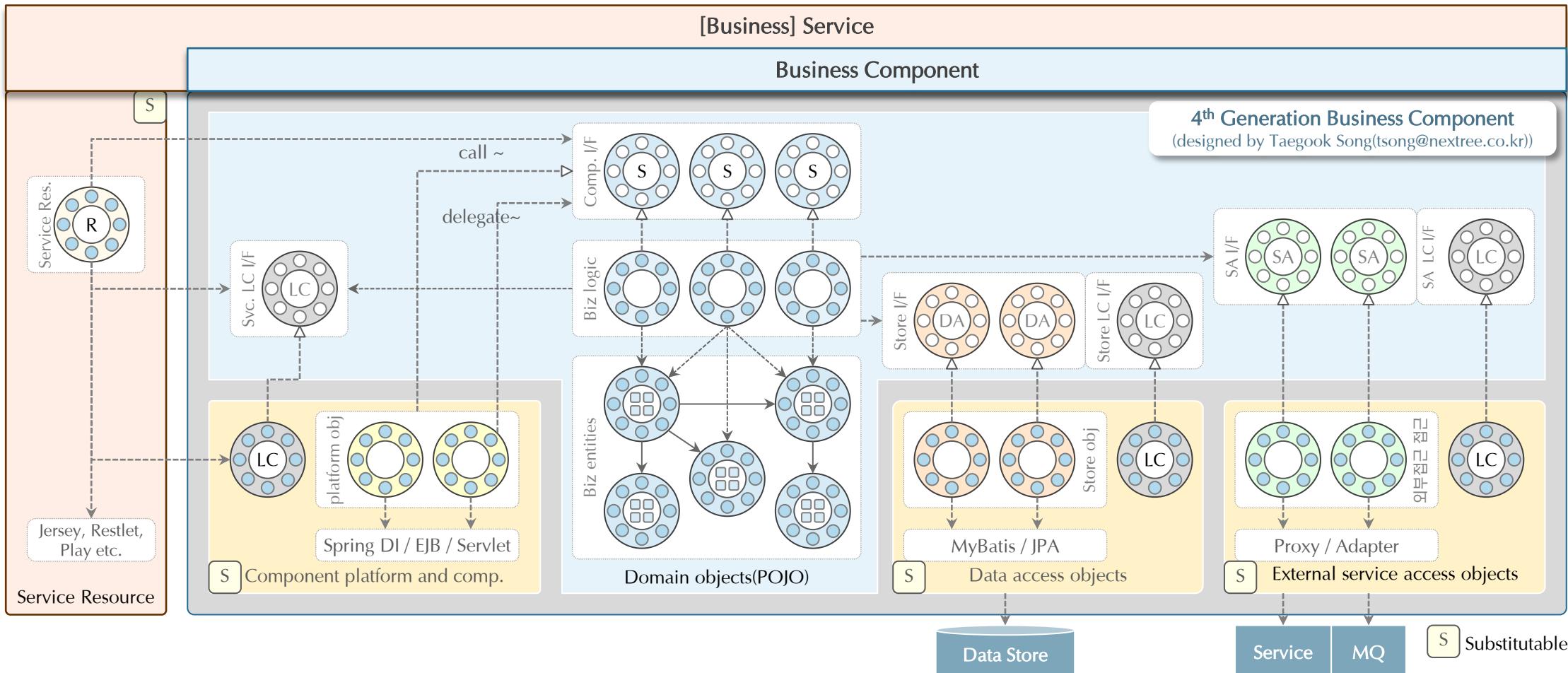
6.4 PURE 도메인 컴포넌트(4/9) – 저장소로부터 자유로움

- ✓ Big data 시대에는 데이터의 특성에 따른 최적의 저장 방식을 찾아야 합니다.
- ✓ 기존의 Relational Database Management System는 데이터 관리 + 데이터 저장의 역할을 동시에 수행했습니다.
- ✓ 데이터의 특성에 따른 최적화된 처리를 위해 관리(management)는 빼고 저장(store)에 집중하고 있습니다.



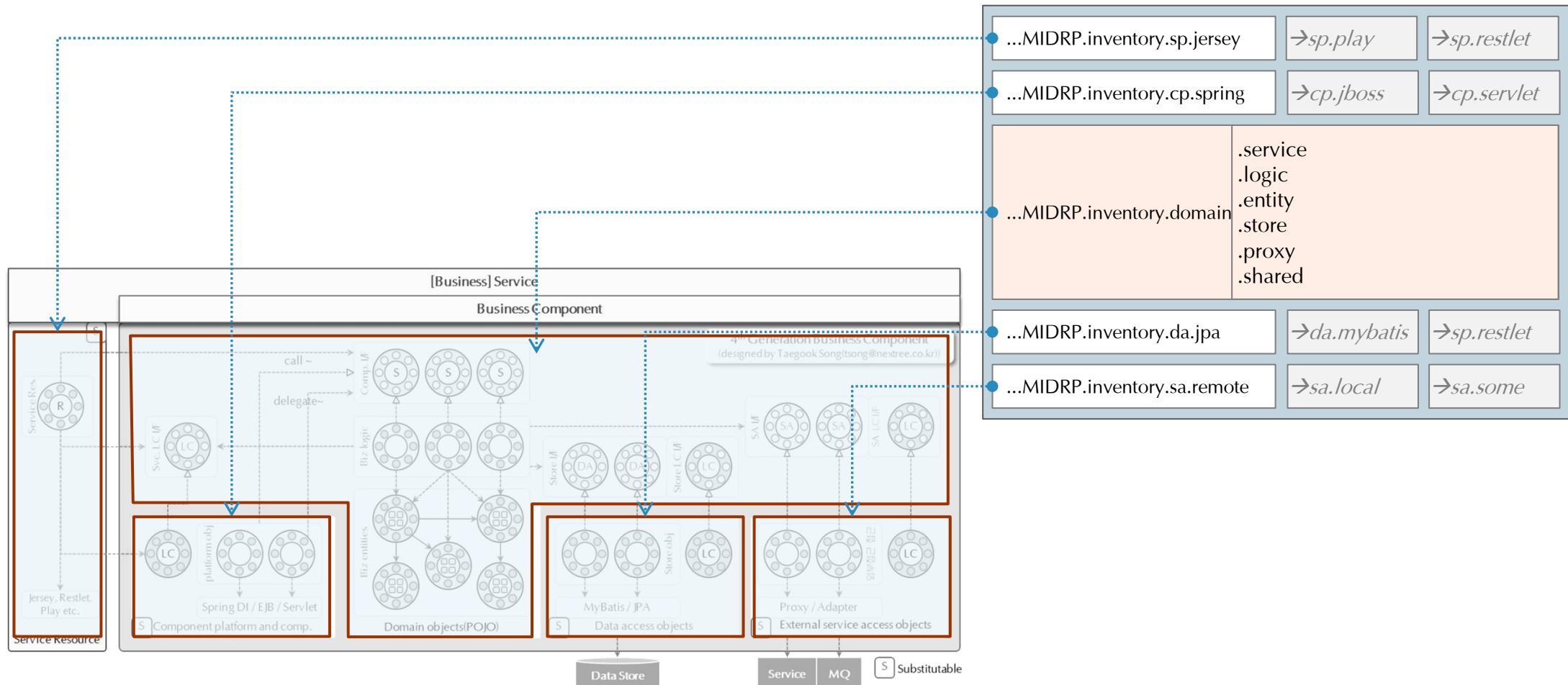
6.4 PURE 도메인 컴포넌트(5/9) – 시각적인 표현

- ✓ 컴포넌트 구조를 시각적으로 표현하여 보면 다음과 같습니다.
- ✓ 컴포넌트와 서비스의 경계를 명확하게 볼 수 있습니다. 서비스는 인터페이스 퍼블리싱 임을 알 수 있습니다.
- ✓ 교제 가능한 부분과 그렇지 않은 부분을 명확하게 알 수 있습니다.



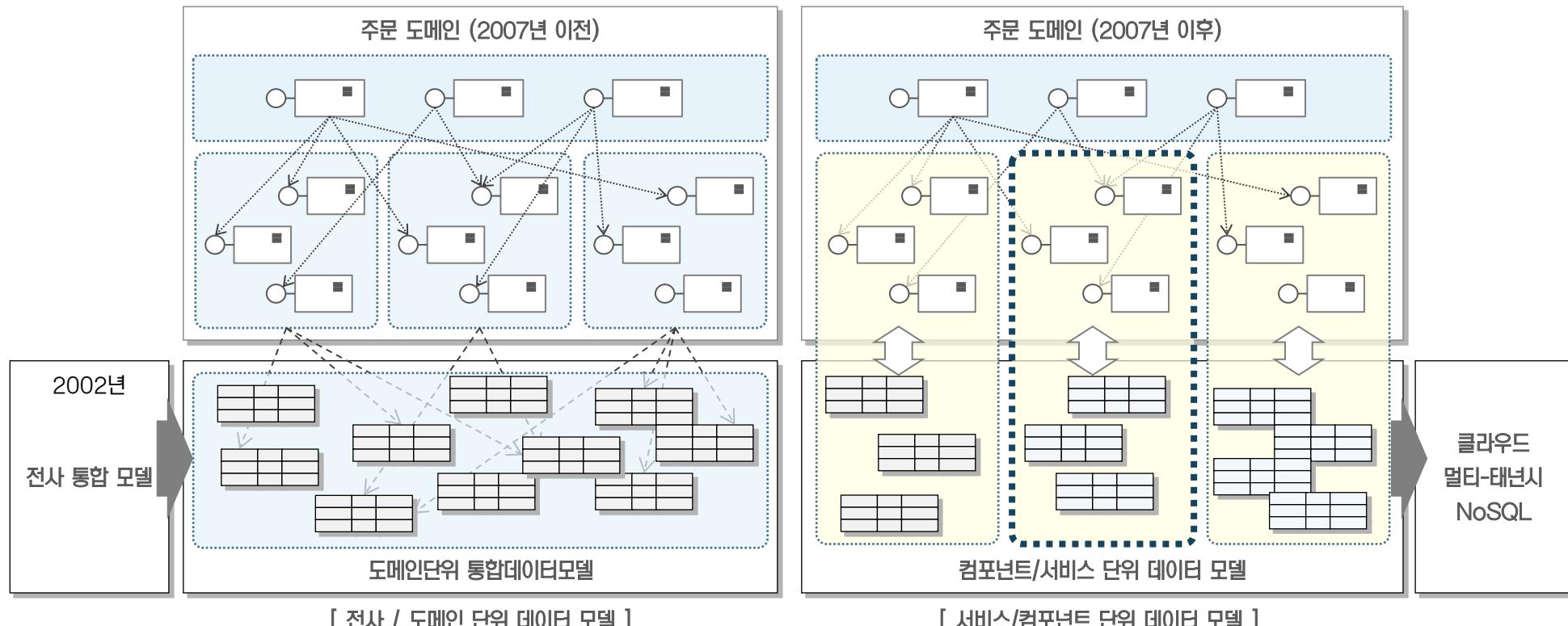
6.4 PURE 도메인 컴포넌트(6/9) – 패키지 맵핑

- ✓ 컴포넌트 내부 구조는 개발 시점의 프로젝트 구성과 패키지 구성과 직접 연결되어 있습니다.
- ✓ 프로젝트 구성은 개발 접근방법, 팀 구성, 구성원 기술 특성, 관리 용이성 등을 반영하여야 합니다.
- ✓ Pure 도메인 컴포넌트는 다양한 프로젝트 구조를 지원할 수 있습니다.



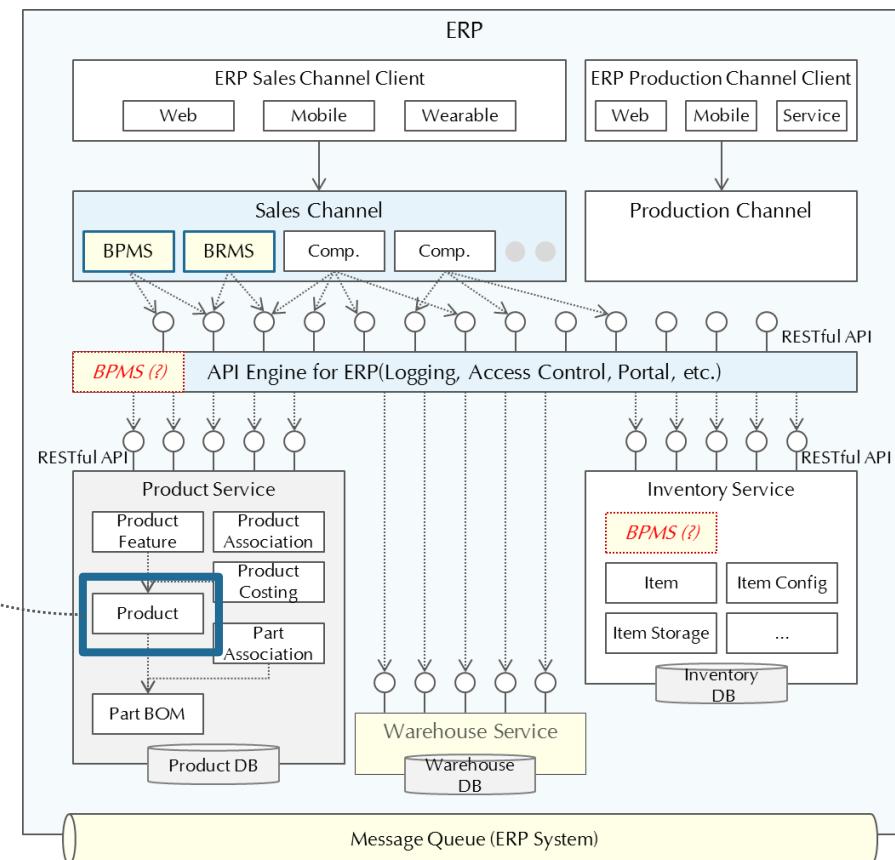
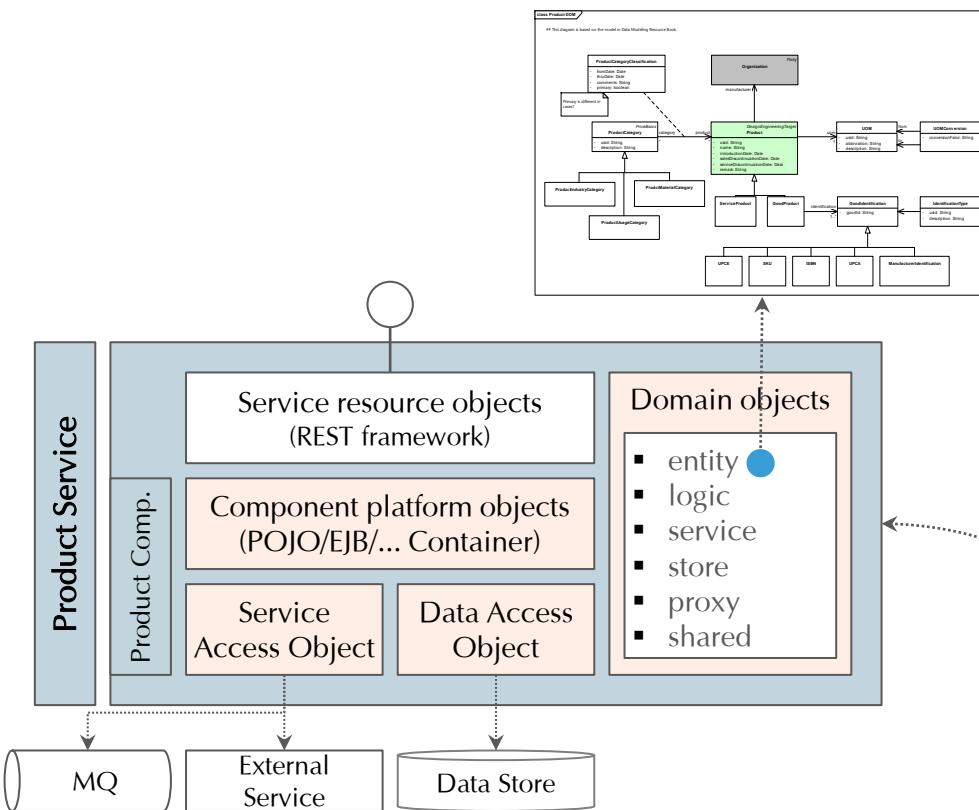
6.4 PURE 도메인 컴포넌트(7/9) – 도메인 모델/데이터 모델 2

- ✓ AS-IS 시스템의 설계는 전사 통합 데이터 모델 사상 위에 비즈니스 컴포넌트를 올린 형태입니다.
- ✓ 전사 통합데이터모델 → 도메인 단위 통합모델 → 서비스/컴포넌트 단위 모델 순으로 시스템 설계가 진화하고 있습니다.
- ✓ 올바른 컴포넌트를 구성하기 위해서 서비스/컴포넌트 단위 데이터 모델로 변경하여야 합니다.



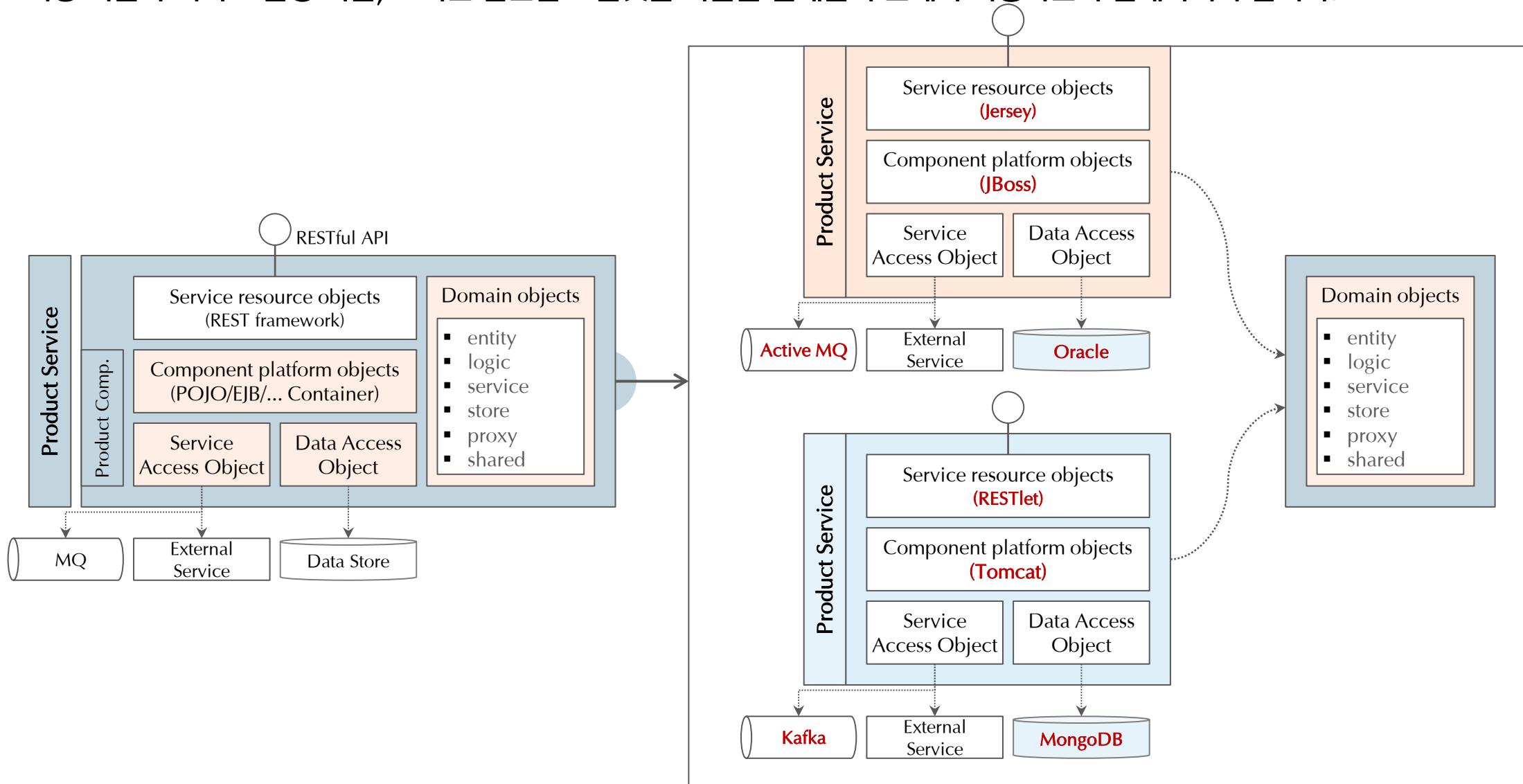
6.4 PURE 도메인 컴포넌트(8/9) – Close up 1

- ✓ 대규 시스템 속에서 컴포넌트와 서비스의 위치를 확인하여 봅니다.
- ✓ 그 속에서 엔티티 클래스들의 위치 또한 확인함으로써, 시스템, 컴포넌트, 서비스를 줌-인, 줌-아웃 하여 봅니다.
- ✓ 현재의 컴포넌트는 그대로 서비스(또는 마이크로서비스)로 확장하는 것을 고려하여 설계합니다.



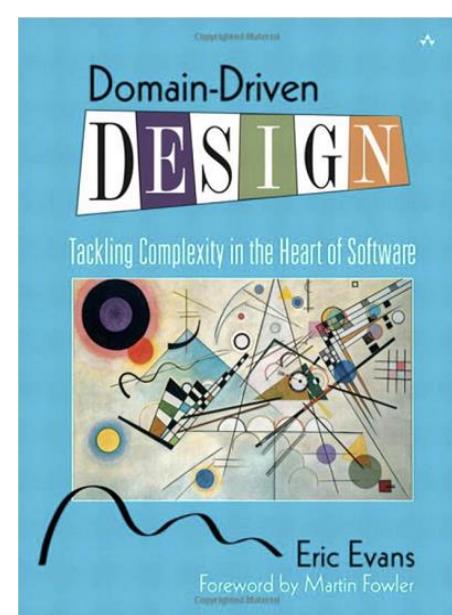
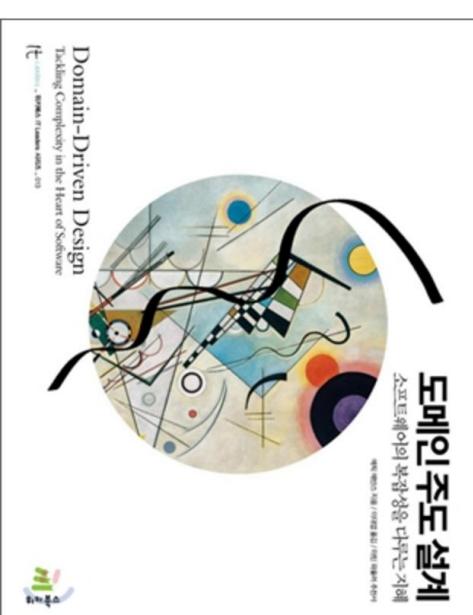
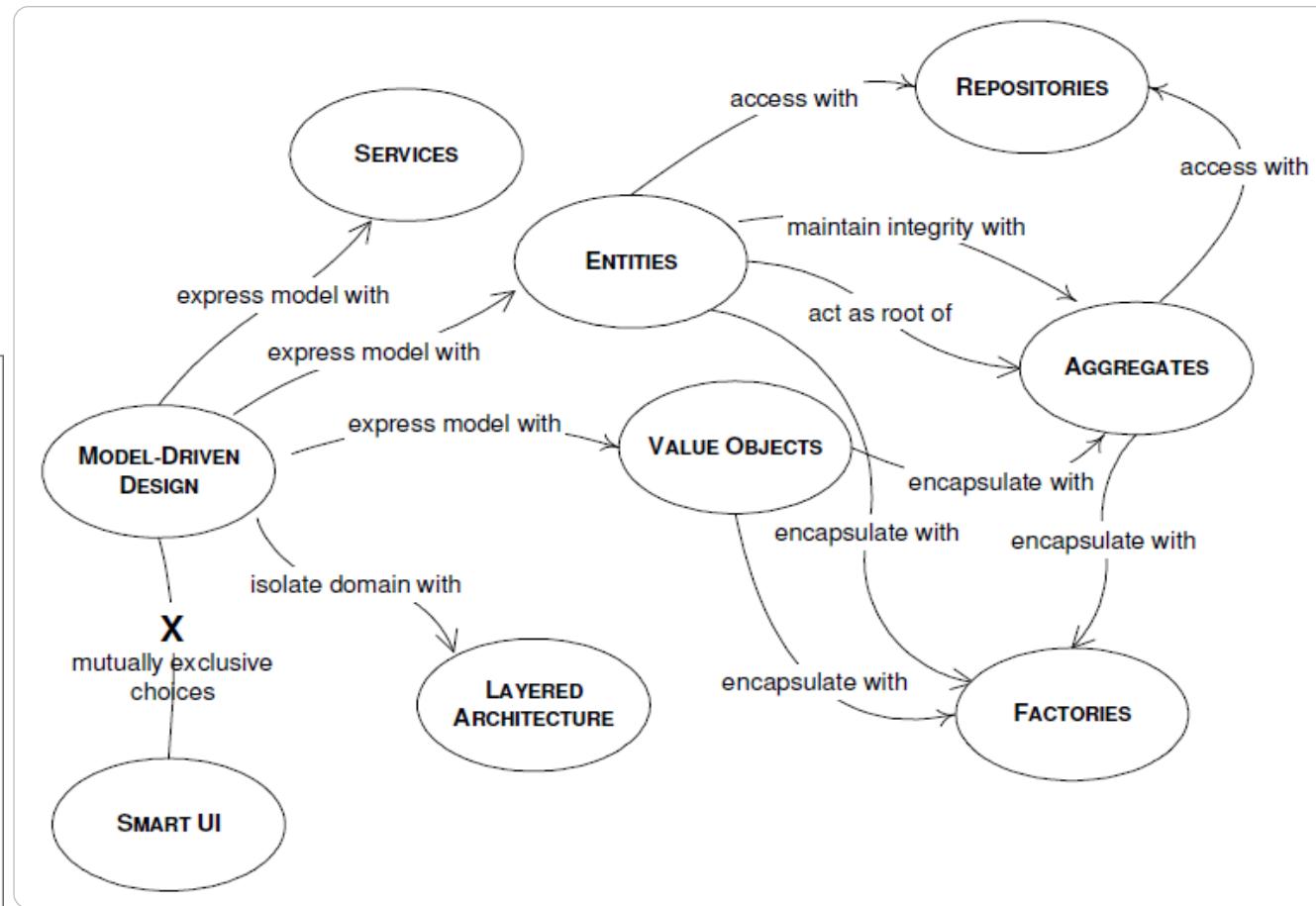
6.4 PURE 도메인 컴포넌트(9/9) – Close up 2

✓ 저장 기술과 서비스 발행 기술, 그리고 컴포넌트 플랫폼 기술은 언제든지 교체가 가능하도록 설계하여야 합니다.



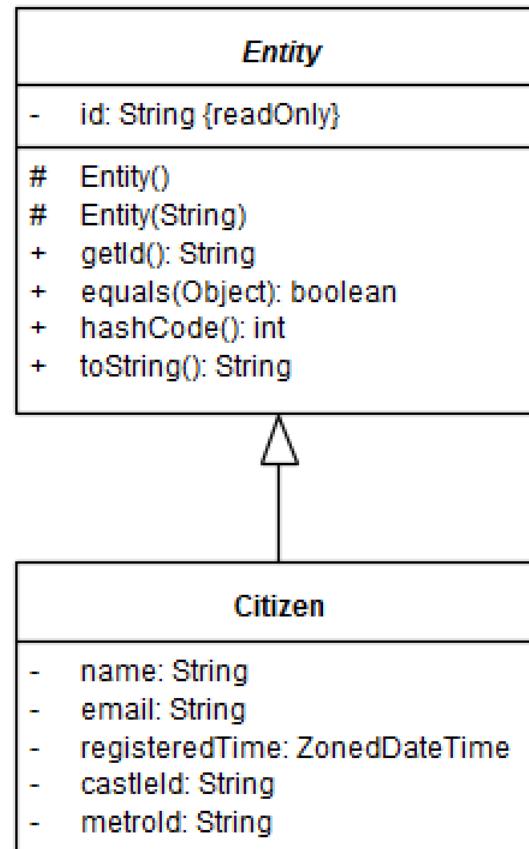
6.5 DDD – 모델 요소(1/7) – 기본 블럭

- ✓ DDD를 이끌어 가는 기본 개념(패턴, 또는 패턴 언어) 간의 관계는 navigation map에서 확인할 수 있습니다.
- ✓ 도메인을 표현하는 기본 모델 요소는 Entities, Value Objects, Services 세가지입니다.
- ✓ 세 요소를 효율적으로 표현하기 위해 Repositories, Aggregates, Factories 세 가지 개념을 추가했습니다.



6.5 DDD – 모델 요소(2/7) – Entity 1

- ✓ 객체는 다른 객체와 식별이 되고, 식별 자체가 연속성을 갖는가를 기준으로 정의합니다.
- ✓ 따라서, 엔티티는 생명주기 동안 형태와 내용이 변경될 수도 있지만, 연속성은 유지해야 합니다.
- ✓ Entity는 상태를 변경할 수 있으며, Value Object 와는 다른 라이프 사이클을 가집니다.



6.5 DDD – 모델 요소(3/7) – Entity 2

- ✓ Entity는 별도로 분리되어 있을 때, 자신의 책임을 가장 잘 수행합니다.
- ✓ Entity를 정의할 때, 속성이나 행위에 집중하기 보다는 본질적인 특성에 초점을 두고 정의합니다.
- ✓ 아래 abstract 클래스인 Entity는 Identity를 설정하고, 유지하고, 비교에 활용하는 기능을 제공합니다.

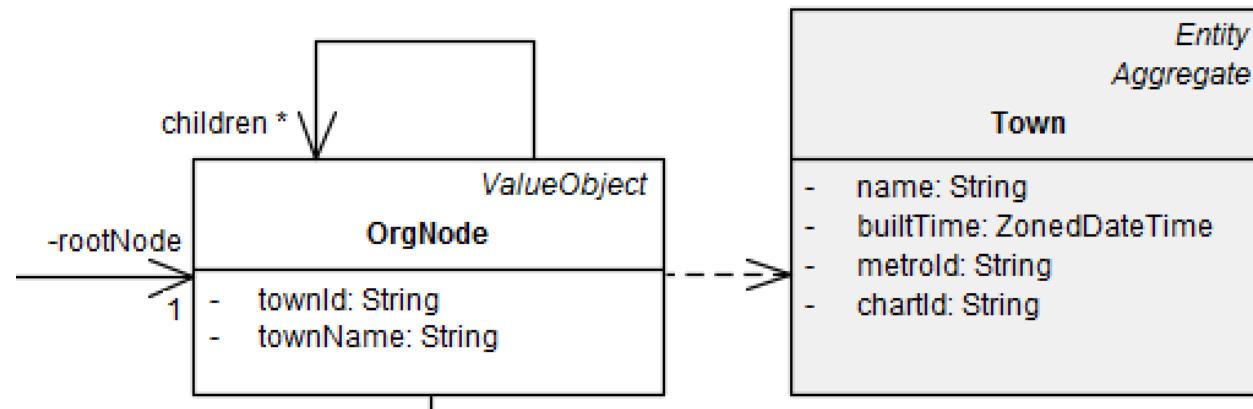
<pre>public abstract class Entity { // private final String id; protected Entity() { this.id = UUID.randomUUID().toString(); } protected Entity(String id) { this.id = id; } public String getId() { return this.id; } }</pre>	<pre>public boolean equals(Object target) { if(this == target) { return true; } else if(target != null && this.getClass() == target.getClass()) { Entity entity = (Entity)target; return Objects.equals(this.id, entity.id); } else { return false; } public int hashCode() { return Objects.hash(new Object[]{this.id}); } public String toString() { return "id:" + this.id; } }</pre>
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

6.5 DDD – 모델 요소(4/7) – Value Object 1

- ✓ 개념적으로 식별이 필요없고, 단순히 값만을 갖고 있는 객체를 Value Object라고 합니다.
- ✓ 따라서 Value Object는 값을 변경할 수 없는, 즉 상태를 변경할 수 없는 Immutable 객체입니다.
- ✓ 한 번 만들어진 객체의 값은 불변이므로 전송이나 멀티 스레드 환경에서 안전성을 보장합니다.

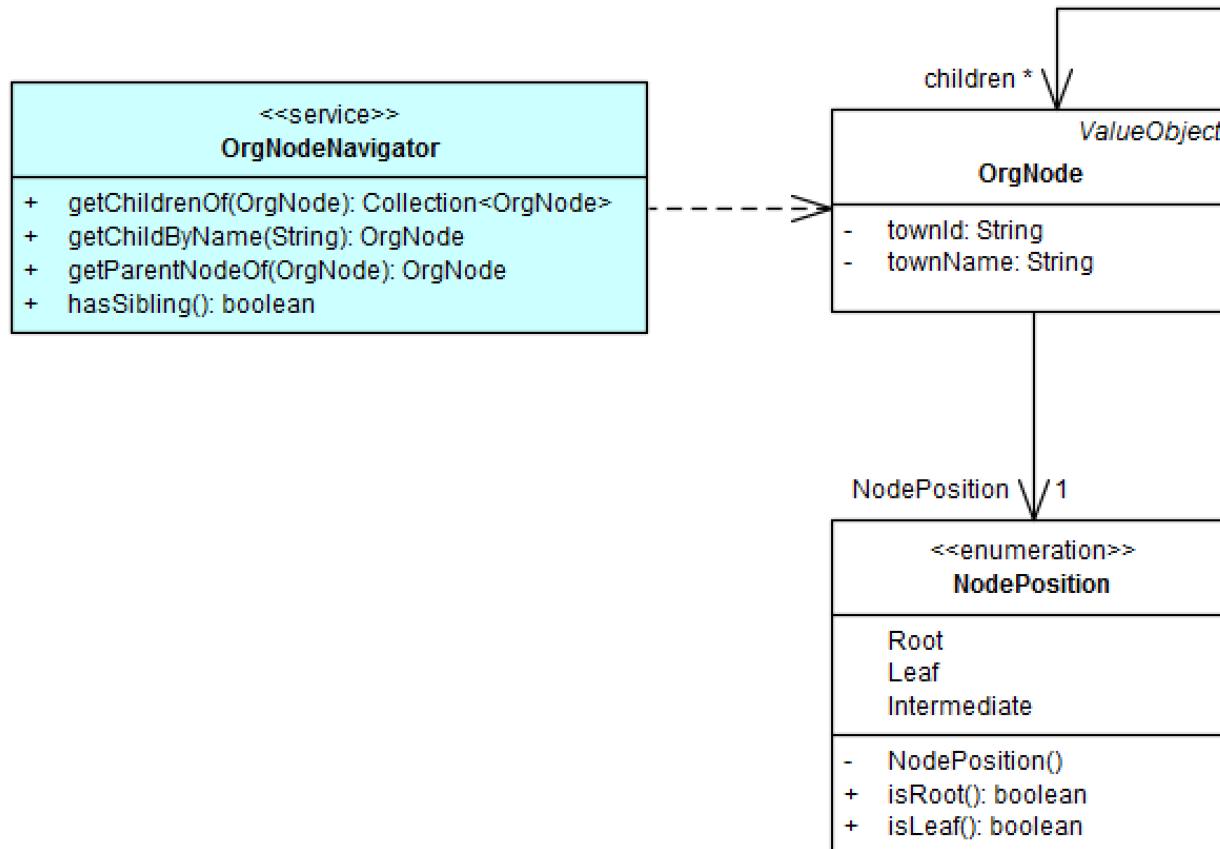
6.5 DDD – 모델 요소(5/7) – Value Object 2

- ✓ Value Object 라고 반드시 단순한 것 만이 아니라, 복합 객체 형식을 가진 것도 있을 수 있습니다.
- ✓ Value Object 도 Entity를 참조할 수 있고, 통신 구간에서 값을 전달하는 용도로 사용할 수 있습니다. → DTO
- ✓ Address 라고 모두 Value Object가 아닙니다. 우체국의 주소체계를 관리 SW에서 Address는 Entity입니다.
- ✓ Value Object 가 서로 참조하는 것은, Value Object가 식별성이 없으므로 표현상의 오류입니다.



6.5 DDD – 모델 요소(6/7) – Service 1

- ✓ 설계가 매우 명확하고 실용적이더라도 어떠한 객체에도 포함할 수 없는 연산을 Service로 정의합니다.
- ✓ 이러한 연산은 본질적으로 사물이 아니라 활동(activity)이나 행동(action)입니다.
- ✓ 복잡한 연산을 분리하지 못하면, 객체를 복잡하게 하거나 개념을 흩어 놓아 모델을 망칩니다.

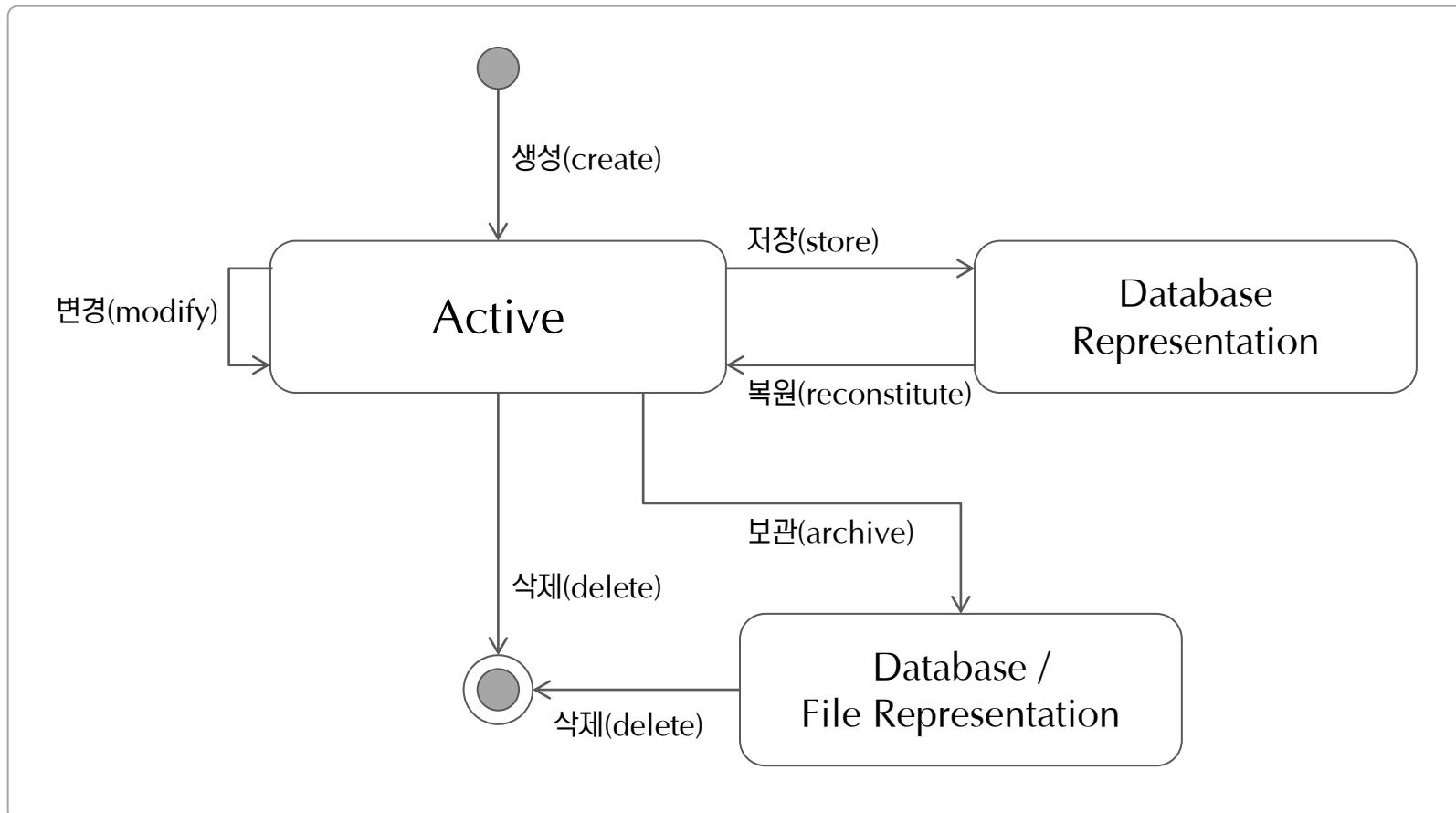


6.5 DDD – 모델 요소(7/7) – Service 2

- ✓ 연산이 Entity나 Value Object의 한 부분이 아니라 도메인 개념과 관련이 있을 때, Service로 정의합니다.
- ✓ 도메인 모델의 외적인 요소에 의해 인터페이스가 결정될 때, Service로 정의합니다.
- ✓ 도메인의 프로세스나 연산이 어떤 Entity나 Object Value의 책임이 아니라면 Service로 정의합니다.
- ✓ 모델의 언어라는 측면에서 인터페이스를 정의하고, Ubiquitous Language로 이름을 정합니다.
- ✓ 기술적인 요소를 포함하고 있는 것은 응용 서비스입니다. 도메인 Service와 철저하게 분리합니다. → File, Excel

6.6 DDD – 도메인 객체 생명 주기(1/7)

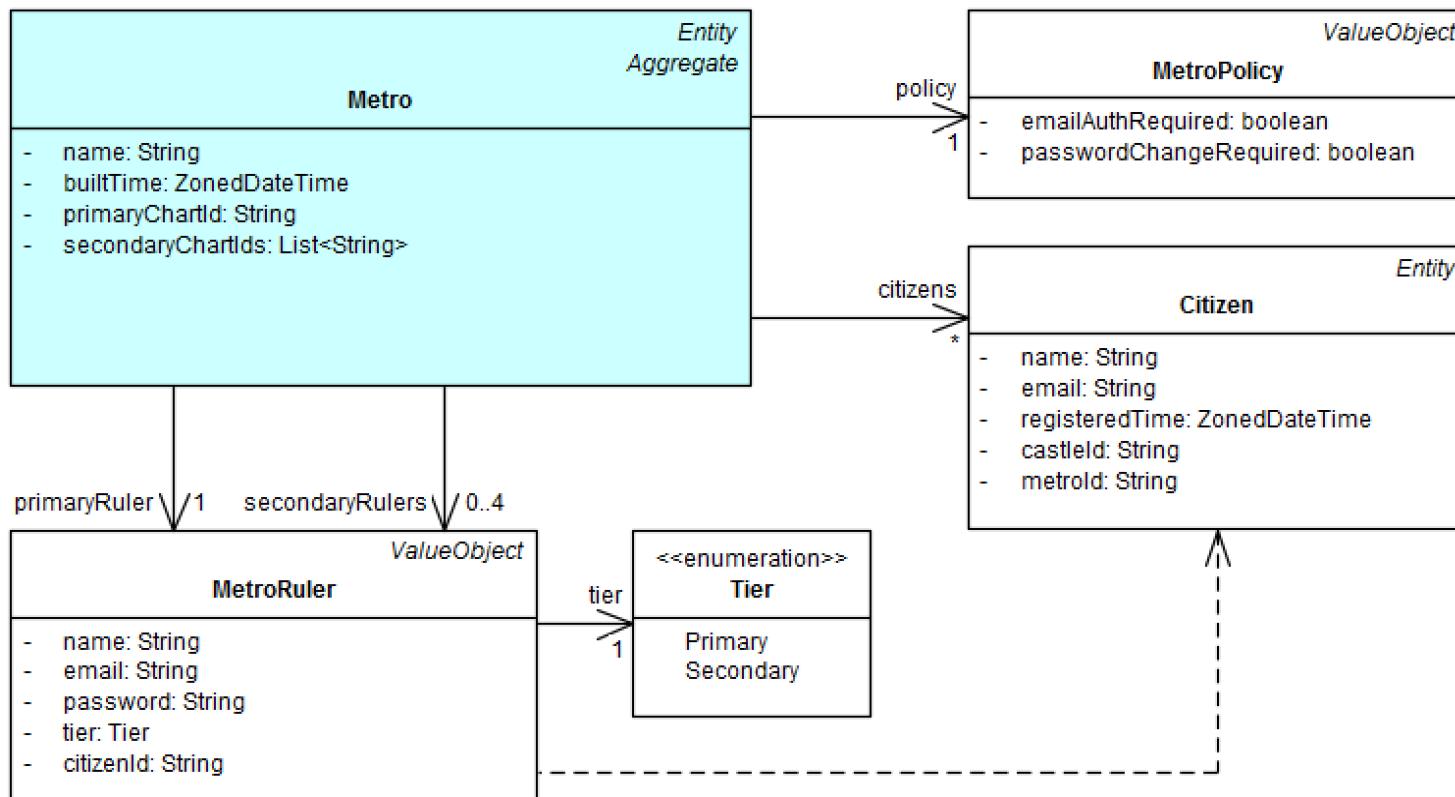
- ✓ 도메인 객체는 생성으로부터 소멸에 이르기까지 세 가지 상태를 가질 수 있습니다.
- ✓ 정보 처리를 위해 비즈니스 로직 영역(메모리 영역)으로 읽어 들였을 때를 활성(active) 상태라고 합니다.
- ✓ 지속성을 보장하고 필요할 때 사용하려면 DB에 저장하거나, 오랫동안 보관을 위해 파일/DB로 저장합니다.



[A lifecycle of a domain object, Eric Vans]

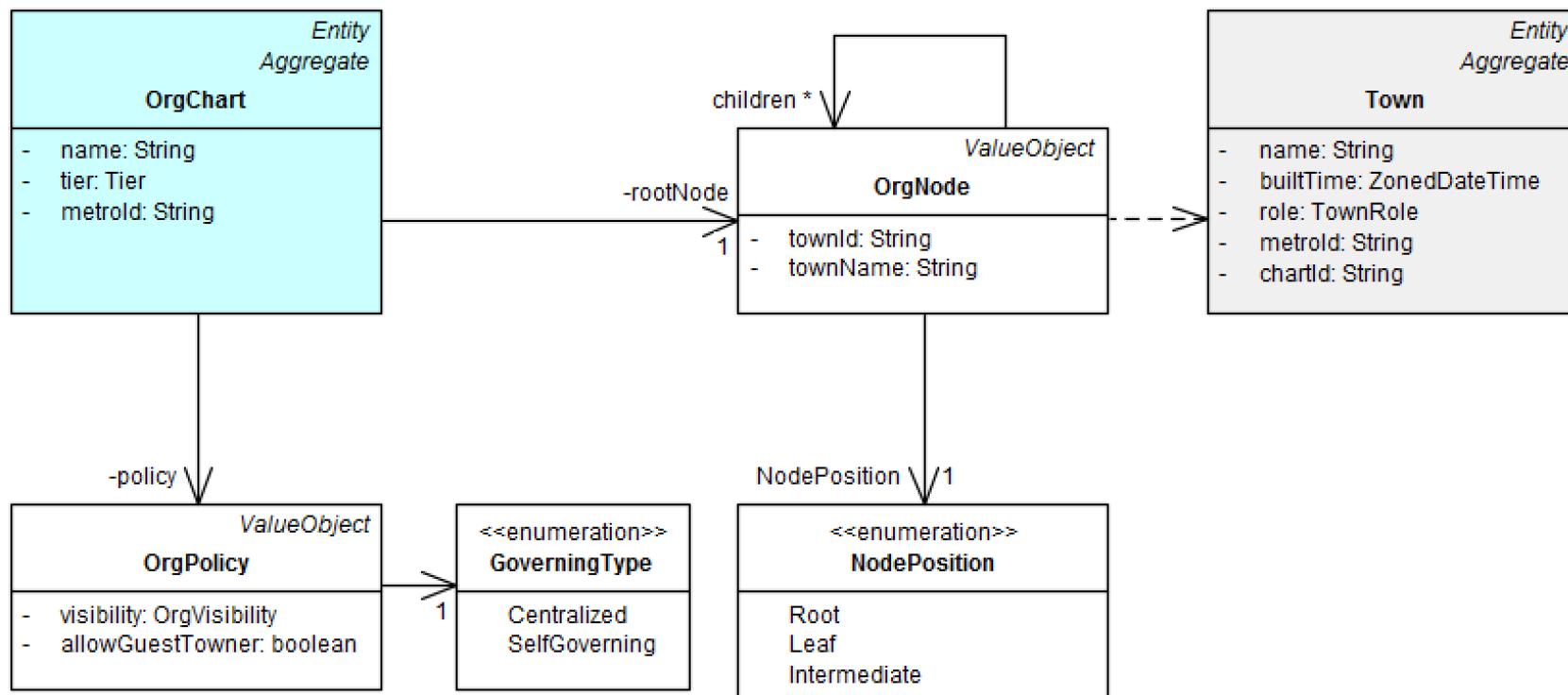
6.6 DDD – 도메인 객체 생명 주기(2/7) – Aggregate 1

- ✓ 업무 관점의 응집도를 유지하는 선에서 연관관계를 가능한 단순하게 유지합니다.
- ✓ Aggregate은 소유권과 경계를 명확하게 정의하여, 객체 관계에 질서를 부여합니다.
- ✓ 도메인 객체를 처리하는 트랜잭션 단위이며 따라서 무결성을 유지하는 단위이기도 합니다.



6.6 DDD – 도메인 객체 생명 주기(3/7) – Aggregate 2

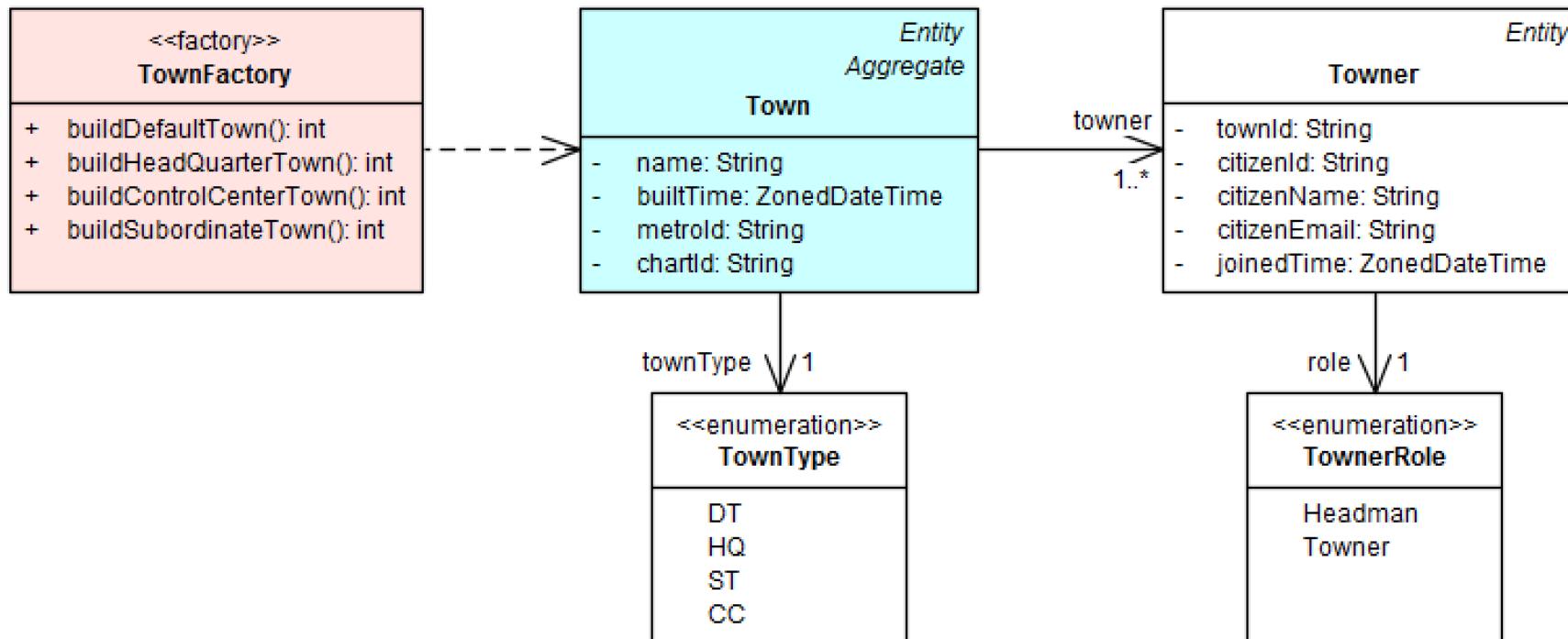
- ✓ Aggregate 루트는 Identity를 가지는 Entity이면서 동시에 내부 객체들에 대한 생명주기를 책임집니다.
- ✓ Aggregate 루트는 전역 식별 범위를 가지며, 내부의 Entity에 대한 참조를 외부로 전달할 수 있습니다.
- ✓ Aggregate 안의 객체는 서로 참조할 수 있지만, 외부 Aggregate는 루트만 참조할 수 있습니다.



[OrgChart as an Aggregate root]

6.6 DDD – 도메인 객체 생명 주기(4/7) – Factory 1

- ✓ Aggregate을 생성하는 일이 복잡하거나, 내부 구조를 많이 드러내어야 하는 경우, Factory를 사용합니다.
- ✓ 복잡한 생성 과정을 Aggregate에게 맡기면, Aggregate은 내부 구조를 관리하는 수준의 복잡한 일을 합니다.
- ✓ 외부의 클라이언트에게 생성을 넘기면, Encapsulation 원칙을 깨는 결과를 가져옵니다.

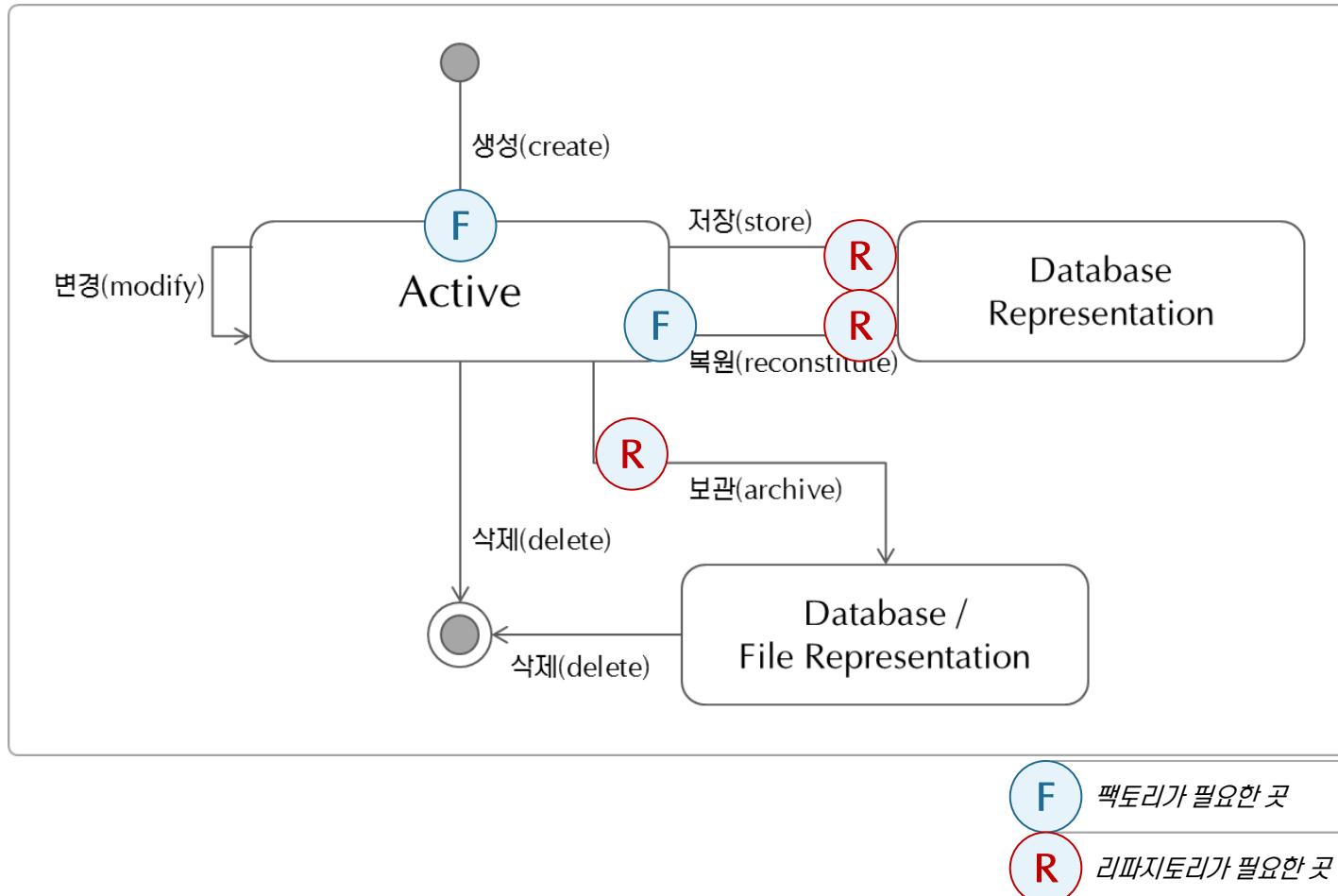


6.6 DDD – 도메인 객체 생명 주기(5/7) – Factory 2

- ✓ 생성이 상대적으로 간단한 경우, Aggregate 루트에 Factory 메소드를 둡니다.
- ✓ Aggregate 내부에 있는 어떤 Entity가 Aggregate 바깥에 있어야 하는 Entity를 만듭니다. → 정보가 있으므로
- ✓ Aggregate 외부에 별도의 Factory 객체를 두는 방식으로 설계합니다.
- ✓ Entity를 위한 Factory를 필요한 최소한의 매개변수를 넘겨 받아 Entity를 생성하고 나머지 값은 후에 받습니다.
- ✓ Entity를 위한 Factory는 식별을 위한 Identity 값을 외부로부터 넘겨 받거나 내부에서 만듭니다.
- ✓ Value object를 위한 Factory는 한번에 완전한 Value object를 만들어야 합니다. → 상태 변화가 없으므로
- ✓ Factory는 생성 뿐만 아니라 복원(reconstitute) 활동 역시 지원하여야 합니다.

6.6 DDD – 도메인 객체 생명 주기(6/7) – Repository 1

- ✓ Factory의 Repository는 생명 주기의 시작 점을 다루는 측면이 있으므로 차이점을 알기 어렵습니다.
- ✓ Factory는 생성과 복원(또는 재구성)에 사용하고, Repository는 저장과 찾아오기에 사용합니다.
- ✓ Repository는 도메인을 기술과 구분하는 경계 지점 역할을 합니다.



6.6 DDD – 도메인 객체 생명 주기(7/7) – Repository 2

- ✓ Repository를 설계할 때, 해당 타입의 객체로 구성한 컬렉션이 있다고 가정하고 인터페이스를 정의합니다.
- ✓ Repository에 객체를 추가하고 삭제하는 메소드를 제공하여, 물리적인 저장소로의 저장과 질의를 추상화합니다.
- ✓ Repository는 Aggregate 루트에만 제공하고, 모든 내부 객체 저장과 접근을 위임합니다.

❖ Repository의 장점

1. Repository는 저장된 객체를 획득하고, 객체의 생명주기를 관리하는 단순한 모델을 제시한다.
2. Repository는 여러 저장 방식으로부터 도메인을 분리하여 준다.
3. Repository 인터페이스를 활용하면 테스트를 용 Mock-up 객체를 쉽게 만들 수 있다.

6.7 실습 6-1: 컴포넌트 설계와 구현

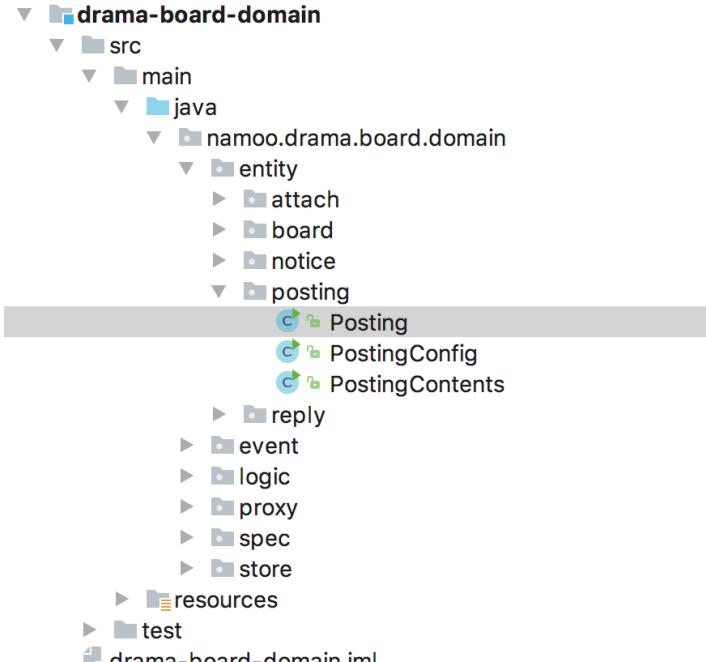
- ✓ 도메인 스트레스를 줄이기 위해 누구나 잘 알고 있는 도메인을 목표로 설정하였습니다.
- ✓ 교재(5.2 도메인 이해 사례 II)에서 예로 사용했던 나무보드가 목표 시스템입니다.
- ✓ 도메인 모델링(2:1) → 컴포넌트 구조 설계(1:1) → 컴포넌트 구현(4:2) 세 가지 작업을 수행합니다. ←(:) 속은 시간 배분
- ✓ 먼저 도메인 모델링 부터 시작합니다. 실습이 끝난 후 Clean 도메인 모델에 대한 리뷰를 합니다.
- ✓ 다음으로 컴포넌트 구조 설계를 합니다. 실습이 끝난 후 Clean 컴포넌트 구조에 대한 리뷰를 합니다.
- ✓ 마지막으로 컴포넌트를 구현합니다. 실습이 끝난 후 Clean 컴포넌트를 리뷰합니다.

The screenshot shows a web application interface. On the left, there is a vertical sidebar with three items: '자유게시판' (Free Board), '공지게시판' (Notice Board), and '사내 공지' (Internal Notice). The '사내 공지' item is highlighted with a dark grey background. The main content area has a title '사내 공지' and a subtitle '사내 공지'. Below the title is a table with two rows of data. The table has columns for 'No.', '제목' (Title), '작성일' (Written Date), '작성자' (Writer), and '조회' (View). The first row contains No. 0002, Title '경비사용 항목에 대한 알림입니다.', Written Date '2017. 3. 20.', Writer 'doitsong', and Views '2'. The second row contains No. 0001, Title '3월 첫번째 산행 공지입니다.', Written Date '2017. 3. 20.', Writer 'doitsong', and Views '2'. At the bottom right of the table is a blue button labeled '글쓰기' (Write Article). At the very bottom center are navigation icons: a double-left arrow, a central number '1' in a blue box, and a double-right arrow.

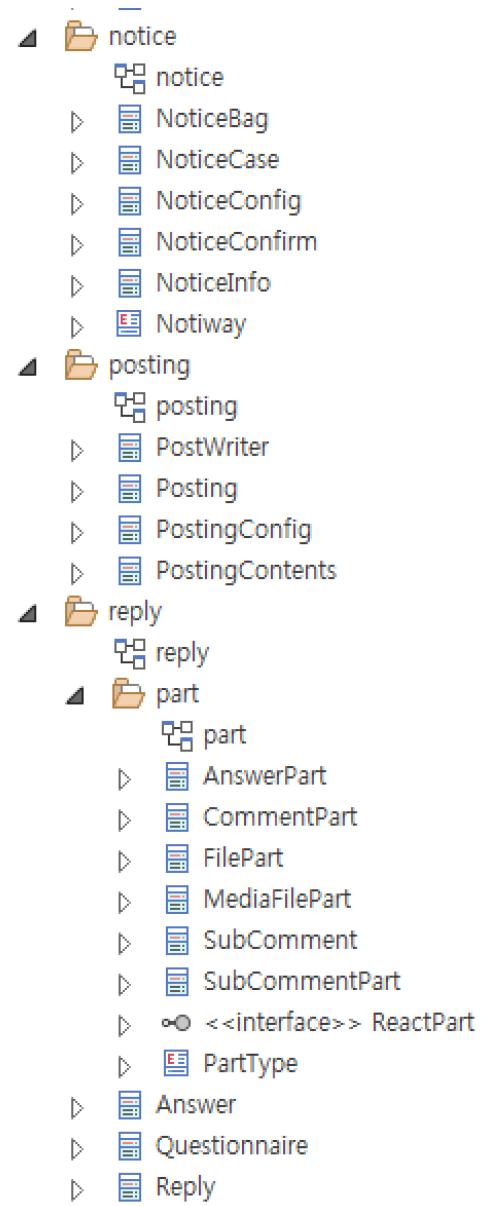
No	제목	작성일	작성자	조회
0002	경비사용 항목에 대한 알림입니다.	2017. 3. 20.	doitsong	2
0001	3월 첫번째 산행 공지입니다.	2017. 3. 20.	doitsong	2

6.7 실습 6-1: Clean 컴포넌트 리뷰

- ✓ 잘 작성한 Clean 컴포넌트 예제에 대해, 모델, 컴포넌트 구조, 소스코드를 리뷰합니다.
 - ✓ Clean 컴포넌트는 유지보수가 용이한 유연한 컴포넌트이며, 변화하는 환경이 쉽게 적응할 수 있습니다.
 - ✓ 모델의 목적과 컴포넌트 구조의 목적, 그리고 소스코드의 목적이 서로 다릅니다. ← 각각의 목표는 무엇입니까?
 - ✓ 저장 방식의 변화에 컴포넌트에 미치는 영향에 대해 토의합니다.
 - ✓ 컴포넌트와 서비스의 차이는 무엇인지에 대해 토의합니다.

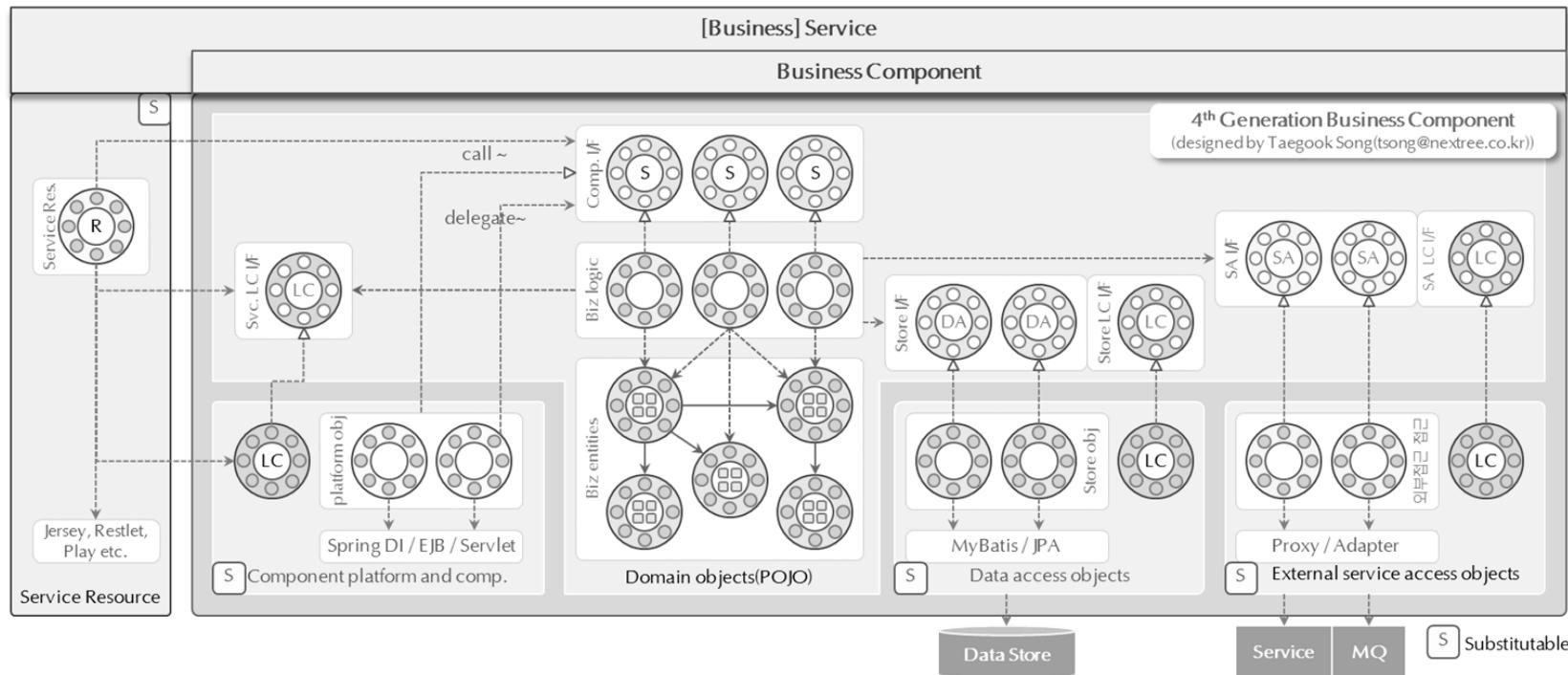


```
3 import ...
10
11 public class Posting extends Entity implements Aggregate {
12     //
13     private final String boardId;
14     private final Actor writer;
15     private final Instant postingTime;
16
17     private String title;
18     private int readCount;
19     private PostingConfig config;
20     private PostingContents contents;
21
22     public Posting(String boardId, String id, Actor writer, String tit
23         //
24         super(id);
25         this.boardId = boardId;
26         this.writer = writer;
27         this.postingTime = Instant.now();
28
29         this.title = title;
30         this.contents = contents;
31         this.config = new PostingConfig();
32         this.readCount = 0;
33     }
34
35     public Posting(String boardId, String id, Actor writer, String tit
```



요약

- ✓ 컴포넌트는 정보 시스템 코드에서 60%가 넘는 비중을 차지하고 있습니다.
- ✓ 객체지향 모델을 품은 컴포넌트는 문제의 본질을 정확히 이해하고 담고 있는 컴포넌트입니다.
- ✓ 도메인 객체 기반, DDD 스타일 컴포넌트는 변경 요구에 매우 빨리 대응할 수 있습니다.
- ✓ 도메인 중심의 컴포넌트는 기술과 업무를 분리함으로써 기술의 변화에 따른 영향을 최소화합니다.



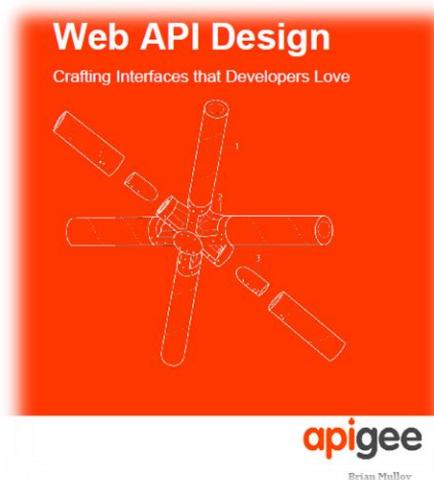


7. RESTful API

-
- 7.1 실용주의 REST API
 - 7.2 명사와 동사
 - 7.3 연관 관계 단순화
 - 7.4 예외 처리
 - 7.5 버전 팀
 - 7.6 페이지 처리와 부분 응답
 - 7.7 자원이 없는 API
 - 7.8 API 호출 예제
 - 7.9 실습 7-1

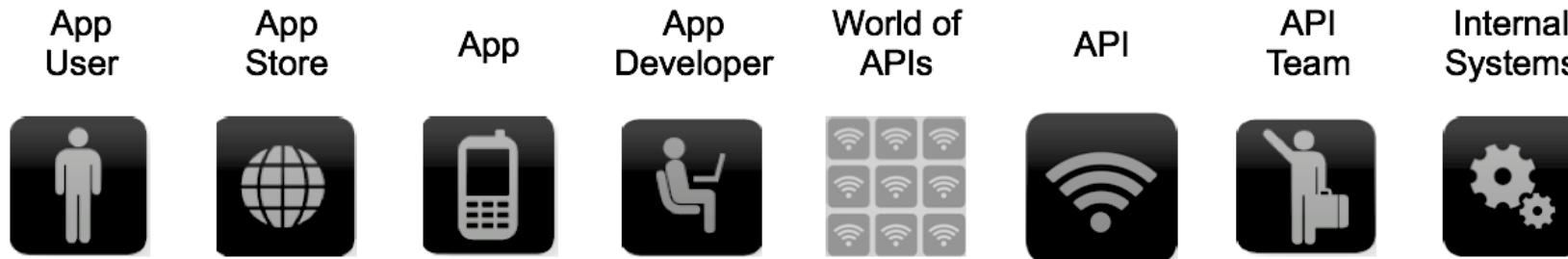
7.1 실용주의 REST API (1/3)

- ✓ 이 장은 apigee(www.apigee.com) 사의 eBook을 기반으로 진행합니다. ← 무료 다운로드 가능함
- ✓ REST API 설계는 아키텍처 스타일이지 표준이 아닙니다. 적용에 유연한 입장을 취할 수 있습니다.
- ✓ API 설계의 핵심은 실용주의 관점을 유지하는 REST입니다.
- ✓ API 설계는 얼마나 많은 개발자들이 빨리 여러분의 API를 이용하느냐로 증명됩니다.



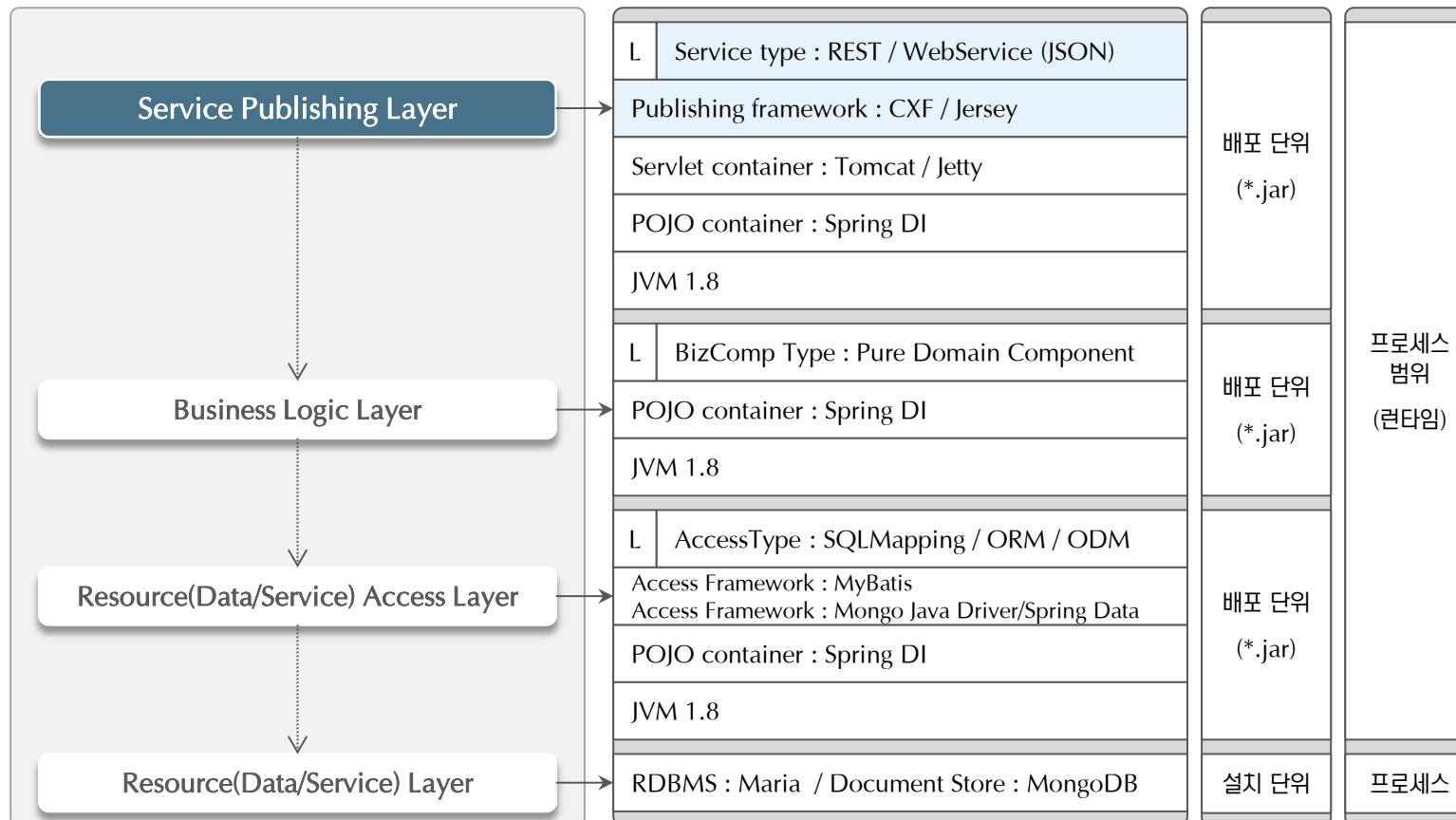
7.1 실용주의 REST API (2/3)

- ✓ API의 목표는 API를 사용하는 개발자의 성공을 돋는 것입니다.
- ✓ 따라서 API 설계의 첫번째 원칙은 개발자의 생산성과 성공을 극대화 하는 것입니다. ← 실용주의 REST
- ✓ 개발자의 관점에서 보면 많은 팁과 경험을 포함하는 훌륭한 가이드가 가장 중요합니다.



7.1 실용주의 REST API (3/3)

- ✓ RESTful API는 애플리케이션 레이어에서 보면 서비스 발행 레이어에 놓입니다.
- ✓ 아키텍처 설계 - 서비스 발행 플랫폼(또는 프레임워크)를 선택하고 RI를 개발한 후, API 가이드를 만듭니다.
- ✓ 도메인 모델(또는 서비스 명세) – API 가이드에 따라 외부로 서비스 API를 정의한 후 발행합니다.



7.2 명사와 동사 [1/3]

- ✓ 실용적인 RESTful 디자인의 첫번째 원칙은 간결하고 직관적인 기준 URL을 유지하는 것입니다.
- ✓ 기준 URL 설계는 API에서 가장 중요한 설계 행동유도성(affordance)입니다.
- ✓ 행동유도성은 가이드 문서가 필요없는 설계 재산입니다. ← 직관적인 설계



설계 행동유도성(affordance)과 문서 간의 충돌 !!

7.2 명사와 동사 (2/3)

- ✓ 자원(resource) 별로 두 개의 기준 url을 사용합니다.
- ✓ 기준 url에는 동사를 두지 않습니다.
- ✓ 컬렉션이나 요소들을 다룰 때는 HTTP 동사(??)를 사용합니다.

// 목록을 위한 URL

/dogs

// 목록 중 특정 개체를 위한 URL

/dogs/1234

Resource	POST(create)	GET(read))	PUT(update)	DELETE(delete)
/dogs	새로운 dog 생성	dogs 목록	dogs에 대한 대량 업데이트	모든 dogs 삭제
/dogs/1234	예러	특정 dog 보기	있으면 업데이트, 없으면 예러	삭제

7.2 명사와 동사 (3/3) – 복수 명사와 구체적인 이름

- ✓ 개발자가 API를 이용할 때, 예측하거나 추측할 수 있도록 일관성을 유지해야 합니다.
- ✓ 직관적인 API는 단수 명사 보다는 복수를 사용합니다.
- ✓ 좋은 API는 추상적인 이름이 아닌 구체적인 이름을 사용합니다.
- ✓ /items이나 /assets 보다는 /blogs나 /videos라는 식으로 구체적인 이름을 사용하는 것이 좋습니다.

Foursquare	GroupOn	Zappos
/checkins	/deals	/Product

7.3 연관관계 단순화

- ✓ URL을 5내지 6레벨까지 구성하는 것은 읽기 어려울 뿐만 아니라 바람직하지 않습니다.
- ✓ 자원 간의 연관 관계를 단순하게 유지하는 방법으로 API의 직관성을 유지해야 합니다.
- ✓ HTTP 물음표(?) 아래 여러 패러미터를 두고 복잡성을 감추어야 합니다.

```
GET /dogs?color=red&state=running&location=park
```

7.4 에러 처리 (1/3)

- ✓ 실용주의 REST API에서는 에러를 어떻게 처리하는가?
- ✓ Facebook은 상태 코드가 200과 #803 에러가 났다고 하지만 그 에러가 무엇인지 알려주지 않습니다.
- ✓ Twilio는 상태코드가 401이며, 에러와 에러에 해당된 내용을 확인할 수 있는 문서정보까지 제공합니다.
- ✓ SimpleGeo는 상태코드가 401이며, 에러코드 외에 다른 정보는 제공하지 않습니다.

Facebook

HTTP Status Code: 200

```
{"type": "OAuthException", "message": "#803 Some of the aliases you requested do not exist: foo.bar"}
```

Twilio

HTTP Status Code: 401

```
{"status": "401", "message": "Authenticate", "code": 20003, "more info": "http://www.twilio.com/docs/errors/20003"}
```

SimpleGeo

HTTP Status Code: 401

```
{"code": 401, "message": "Authentication Required"}
```

7.4 에러 처리 (2/3)

- ✓ HTTP 상태 코드를 사용합니다.
- ✓ 실제 상태 코드는 세 가지 정도만 제공하면 되고 필요하면 더 추가할 수 있습니다.
- ✓ 더 추가할 경우, 최대 8개를 넘지 않아야 합니다.

추천 기본 상태

200 - OK

400 - Bad Request

500 - Internal Server Error

추가상태

201 - Created

304 - Not Modified

404 - Not Found

401 - Unauthorized

403 - Forbidden

7.4 에러 처리 (3/3)

- ✓ 상태코드를 사용하더라도 사용자에게 가능한 자세한 메시지를 제공하여야 합니다.

코드를 위한 코드

200 - OK 401 - Unauthorized

사용자를 위한 메시지

```
{"developerMessage": "Verbose, plain language description of the problem for the app developer with hints about how to fix it.", "userMessage": "Pass this message on to the app user if needed.", "errorCode": 12345, "more info": "http://dev.teachdogrest.com/errors/12345"}
```

7.5 버전 팀 (1/2)

- ✓ 버전 없이 API를 릴리즈하면 안됩니다. 그리고 버전은 옵션이 아닌 필수입니다.
- ✓ 다른 서비스에서 사용하는 **versioning** 예제입니다.
 - Twilio URL에 timestamp를 사용합니다.
 - Salesforce.com은 URL의 중간 지점에 v20.0과 같은 버전 정보를 둡니다.
 - Facebook은 v. 와 같은 버전 표기법을 사용하지만 버전 정보는 선택 가능한 패러미터입니다.

Twilio

/2010-04-01/Accounts/

salesforce.com

/services/data/v20.0/sobjects/Account

Facebook

?v=1.0

7.5 버전 팀 (2/2)

✓ 실용주의적인 REST에서 버전 번호를 어떻게 처리해야 하는가?

- 단순한 서수를 사용합니다.
- 적어도 하나의 버전을 유지합니다.
- 버전을 폐기하기 전에 개발자들이 대응할 수 있는 시간을 충분히(한 사이클 정도) 주어야 합니다.

7.6 페이지 처리와 부분 응답 (1/2)

- ✓ “부분 응답”은 개발자들이 필요로 하는 것만 제공하는 것입니다.
- ✓ 아래 정보는 3가지 서비스에서 제공하고 있는 “부분 응답” 예제입니다.
- ✓ 다음은 각 서비스에서 쉼표로 구분한 옵션 필드 등을 이용하여 제공하는 “부분 응답” API 예제입니다.

LinkedIn

/people:(id,first-name,last-name,industry)

Facebook

/joe.smith/friends?fields=id,name,picture

Google

?fields=title,media:group(media:thumbnail)

7.6 페이지 처리와 부분 응답 [2/2]

- ✓ 개발자가 페이지 처리를 하기 쉽게 offset과 limit을 제공하도록 합니다.
- ✓ 다음은 offset과 limit 값의 예입니다.

Facebook

offset 50 and limit 25

Twitter

page 3 and rpp 25 (페이지 당 레코드 개수)

LinkedIn

start 50 and count 25

7.7 자원이 없는 API

- ✓ DB에 저장된 자원(resource)과 관련이 없는 응답을 처리할 때는 다음과 같이 하는 것을 추천합니다.
 - 명사가 아닌 동사를 사용합니다.
 - 비자원 처리 API는 다른 시나리오를 사용함으로 명확하게 서술합니다.
- ✓ 아래는 100유로를 중국 yen으로 바꿀 때의 예제입니다.

```
/convert?from=EUR&to=CNY&amount=100
```

7.8 API 호출 예제 (1/3)

- ✓ 이름이 AI인 갈색 강아지 객체를 생성합니다.
- ✓ AI라는 이름을 가진 강아지의 이름을 Rover로 이름을 변경합니다.

Request

```
POST /dogs  
name=AI&furColor=brown ← 바디로 전달
```

Response

```
200 OK  
{ "dog": { "id": "1234", "name": "AI", "furColor": "brown" } }
```

Request

```
PUT /dogs/1234  
name=Rover ← 바디로 전달
```

Response

```
200 OK  
{ "dog": { "id": "1234", "name": "Rover", "furColor": "brown" } }
```

7.8 API 호출 예제 (2/3)

- ✓ 특정 강아지 정보를 조회합니다.
- ✓ 모든 강아지 정보를 가져옵니다.

Request

```
GET /dogs/1234
```

Response

```
200 OK
```

```
{ "dog": { "id": "1234", "name": "Rover", "furColor": "brown" } }
```

Request

```
GET /dogs
```

Response

```
200 OK
```

```
{ "dogs": [
    { "dog": { "id": "1233", "name": "Fido", "furColor": "white" } },
    { "dog": { "id": "1234", "name": "Rover", "furColor": "brown" } }
],
"_metadata": [ { "totalCount": 327, "limit": 25, "offset": 100 } ]
}
```

7.8 API 호출 예제 (3/3)

- ✓ 1234라는 아이디를 가진(여기서는 Rover라는 이름을 가진) 강아지 정보를 삭제합니다.

Request

```
DELETE /dogs/1234
```

Response

```
200 OK
```

7.9 실습 7-1 – 서비스 발행

- ✓ 앞 장의 실습 7-1에서 구현한 컴포넌트의 인터페이스를 외부에 서비스로 발행(publishing) 해 봅니다.
- ✓ 간단한 publishing을 위해 Spring boot을 사용합니다.
- ✓ 컴포넌트 인터페이스와 서비스 자원 간의 관계를 생각하면서 구현을 합니다.
- ✓ 서비스 데이터 객체(Service data object) 와 도메인 객체 간의 관계 설정에 대해 고민합니다.

```
BoardService
package namoo.drama.board.domain.spec.board;
import ...
@OptionLabel(
    supportEditions = {"Community", "Professional", "Enterprise"}, 
    feature = "BOARD"
)
@RoleLabel(names = "user")
public interface BoardService {
    // ...
    @RoleLabel(names = "admin")
    String openBoard(BoardSdo boardSdo);
    BoardSdo findBoard(String boardId);
    List<BoardSdo> findBoards(String bookId);
    @ActionLabel(name = "BOARD MODIFY")
    @RoleLabel(names = "admin")
    void modifyBoard(String boardId, BoardSdo boardSdo);
    @ActionLabel(name = "BOARD REMOVE")
    @RoleLabel(names = "admin")
    void removeBoard(String boardId);
}
```

```
PostingService
package namoo.drama.board.domain.spec.posting;
import ...
@OptionLabel(
    supportEditions = {"Community", "Professional", "Enterprise"}, 
    feature = "POSTING"
)
@RoleLabel(names = "user")
public interface PostingService {
    // ...
    String registerPosting(String boardId, PostingSdo postingSdo);
    PostingSdo findPosting(String boardId, String postingId);
    List<PostingSdo> findPostings(String boardId, int offset);
    int countAllPostingsOf(String boardId);
    @RoleLabel(names = "admin")
    void modifyPosting(String boardId, String postingId, PostingSdo postingSdo);
    @RoleLabel(names = "admin")
    void removePosting(String boardId, String postingId);
}
```

요약

- ✓ RESTful API는 서비스 발행(publishing)을 위한 표준입니다.
- ✓ Clean API는 RESTful API 경험 법칙을 바탕으로 간결하게 정의한 API입니다.
- ✓ 누구나 이해하기 쉽게 예측 가능한 형태로 API를 구성하는 것은 간단합니다. 앞에서 나열한 규칙을 준수하면 됩니다.
- ✓ 과거에는 이러한 API 설계가 특정 기업이나 개인의 노하우에 속했지만, 현재는 공개되어 누구나 쉽게 사용할 수 있습니다.

Resource	POST(create)	GET(read))	PUT(update)	DELETE(delete)
/dogs	새로운 dog 생성	dogs 목록	dogs에 대한 대량 업데이트	모든 dogs 삭제
/dogs/1234	애러	특정 dog 보기	있으면 업데이트, 없으면 애러	삭제



8. 데이터 처리 변화

-
- 8.1 NoSQL 정의
 - 8.2 NoSQL 저장소
 - 8.3 NoSQL 비즈니스 드라이버
 - 8.4 NoSQL 사례와 연구
 - 8.5 설계 단순화
 - 8.6 DB 트랜잭션
 - 8.7 CAP 이론
 - 8.8 CQRS

8.1 NoSQL 정의(1/3): 개요

- ✓ 샌프란시스코 Bay Area, 대용량 오픈소스 데이터베이스 토론 그룹에서 처음 사용
- ✓ 전통적인 RDBMS와 차별화 하는 의미로 사용하기도 함
- ✓ 하지만, 일반적인 정의에서 SQL이나 RDBMS를 배제하지는 않음

NoSQL 은 데이터를 빠르고(rapid) 효율적인(efficient) 처리를 가능하게
하는 개념의 집합으로, 성능, 신뢰성, 민첩성에 초점을 두고 있다.

8.1 NoSQL 정의(2/3): NoSQL은 ~ 이다.

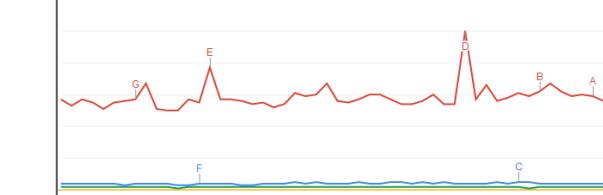
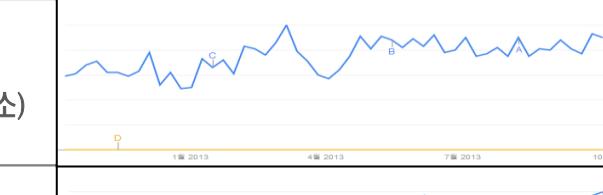
- ✓ 테이블에 저장하는 로우(row) 그 이상이다. → 키-밸류, 그래프, 컬럼-패밀리, 문서, 테이블 로우
- ✓ join으로부터 자유롭다. → join없이 간단한 인터페이스를 사용하여 데이터를 추출함
- ✓ 스키마가 없다. → ER 모델 없이 정보를 저장할 수 있음
- ✓ 여러 프로세서 상에서 동작한다. → 여러 프로세서에 데이터를 저장하고, 빠른 검색이 가능함
- ✓ 일반 판매용 컴퓨터를 사용함
- ✓ 선형적인 확장을 지원한다. → 프로세서를 추가하면, 비례하여 성능 향상을 얻을 수 있음

8.1 NoSQL 정의[3/3]: NoSQL은 ~가 아니다.

- ✓ SQL 언어에 관한 것이 아니다. → NoSQL은 SQL도 사용함
- ✓ 오픈소스여야 하는 것은 아니다. → 상업용 NoSQL 시스템도 있음
- ✓ 빅데이터만을 처리하는 것은 아니다. → Volume과 Velocity가 NoSQL의 주요 특성이지만, Variability와 Agility도 중요한 특성임
- ✓ 클라우드 컴퓨팅이어야 하는 것은 아니다. → 클라우드 뿐만 아니라 기업 데이터센터에서도 실행할 수 있음
- ✓ RAM과 SSD 를 혼명하게 사용하는 것 만은 아니다. → 일반 하드웨어에서도 실행할 수 있음
- ✓ 소수만이 사용할 수 있는 뛰어난 제품 그룹은 아니다. → NoSQL이 자신의 비즈니스 문제를 푸는데 도움이 된다는 확신만 있으면 누구든 사용할 수 있음

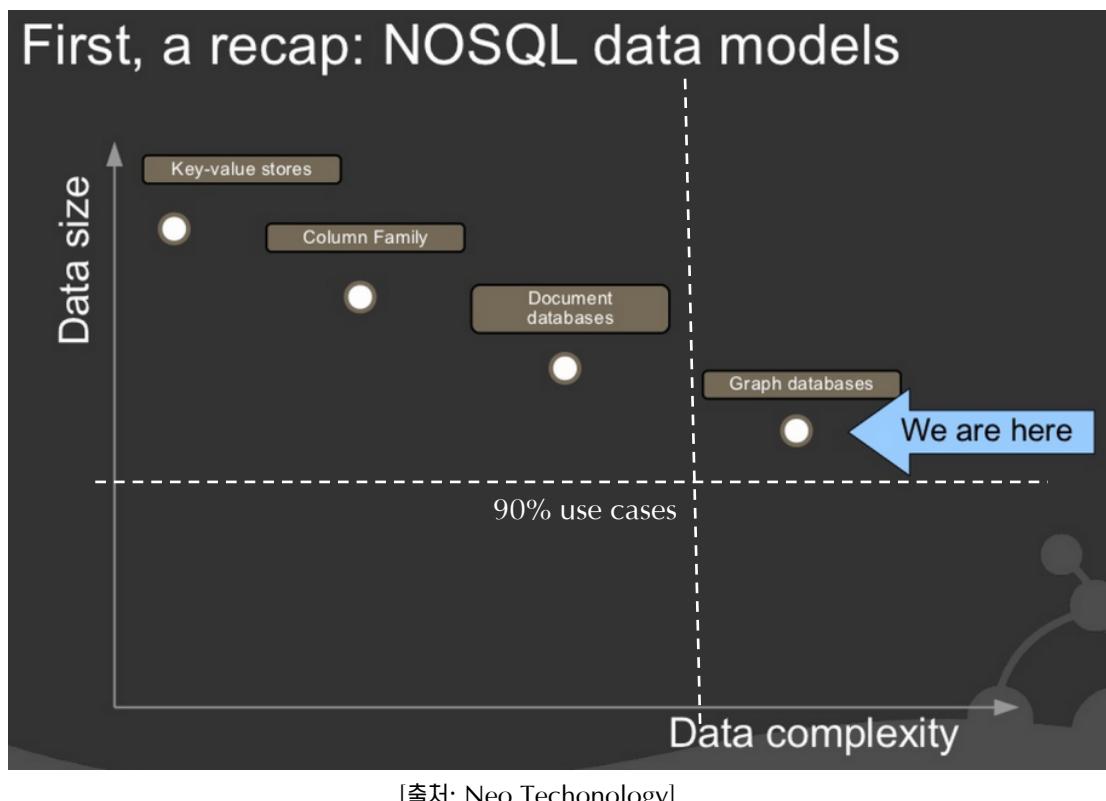
8.2 NoSQL 저장소[1/4]

- ✓ 키-밸류 저장소: 밸류에 접근하기 위해 키를 사용하는 단순한 데이터 저장소
- ✓ 컬럼-패밀리 저장소: 로우와 컬럼을 키로 사용하는 스파스 매트릭스(sparse matrix) 시스템
- ✓ 그래프 저장소: 개체 간의 관계(relationship) 중심적인 문제를 그래프로 처리하는 저장소
- ✓ 문서 저장소: 복잡한 계층 구조를 가진 문서 데이터를 저장하는 저장소

저장소 종류	용도	NoSQL DB	구글 트랜드
키-밸류 저장소	<ul style="list-style-type: none">▪ 이미지 저장소▪ 키 기반 파일시스템▪ 객체 캐시▪ 확장 가능한 설계	<ul style="list-style-type: none">▪ Berkeley DB▪ Memcache▪ Redis▪ Riak▪ DynamoDB	
컬럼-패밀리 저장소	<ul style="list-style-type: none">▪ 웹 크롤링 결과 저장▪ 일관된 규칙이 필요한 빅데이터	<ul style="list-style-type: none">▪ Apache HBase▪ Apache Cassandra▪ Hypertable▪ Apache Accumulo▪ Google BigTable	
그래프 저장소	<ul style="list-style-type: none">▪ 소셜 네트워크▪ 사기 방지▪ 복잡한 관계 중심 데이터	<ul style="list-style-type: none">▪ Neo4j▪ AllegroGraph▪ Bigdata(RDF 데이터 저장소)▪ InfiniteGraph	
문서 저장소	<ul style="list-style-type: none">▪ 매우 다양한 데이터▪ 문서 검색 및 출판▪ 통합 및 연계 허브▪ 웹 컨텐츠 관리	<ul style="list-style-type: none">▪ MongoDB▪ CouchDB▪ MarkLogic▪ eXist-db	

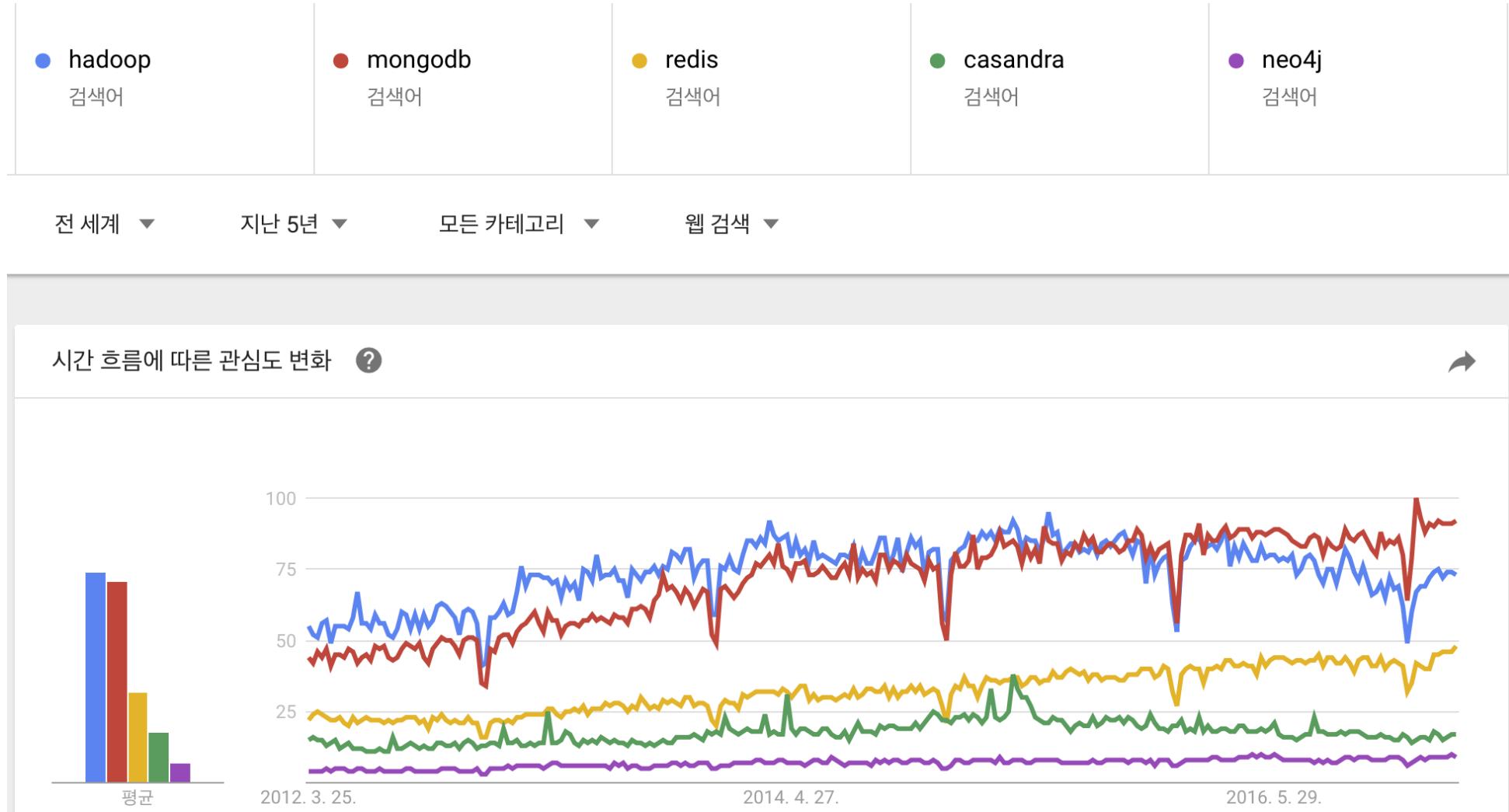
8.2 NOSQL 저장소[2/4]

- ✓ 각 NoSQL 패턴 별로 목표가 서로 다르므로 타겟이 서로 다릅니다.
- ✓ 단순할 수록 더 큰 규모의 데이터를 다룰 수 있습니다. → 키-밸류 저장소 패턴
- ✓ 복잡한 관계 처리는 그래프 저장소 패턴이 적절합니다.



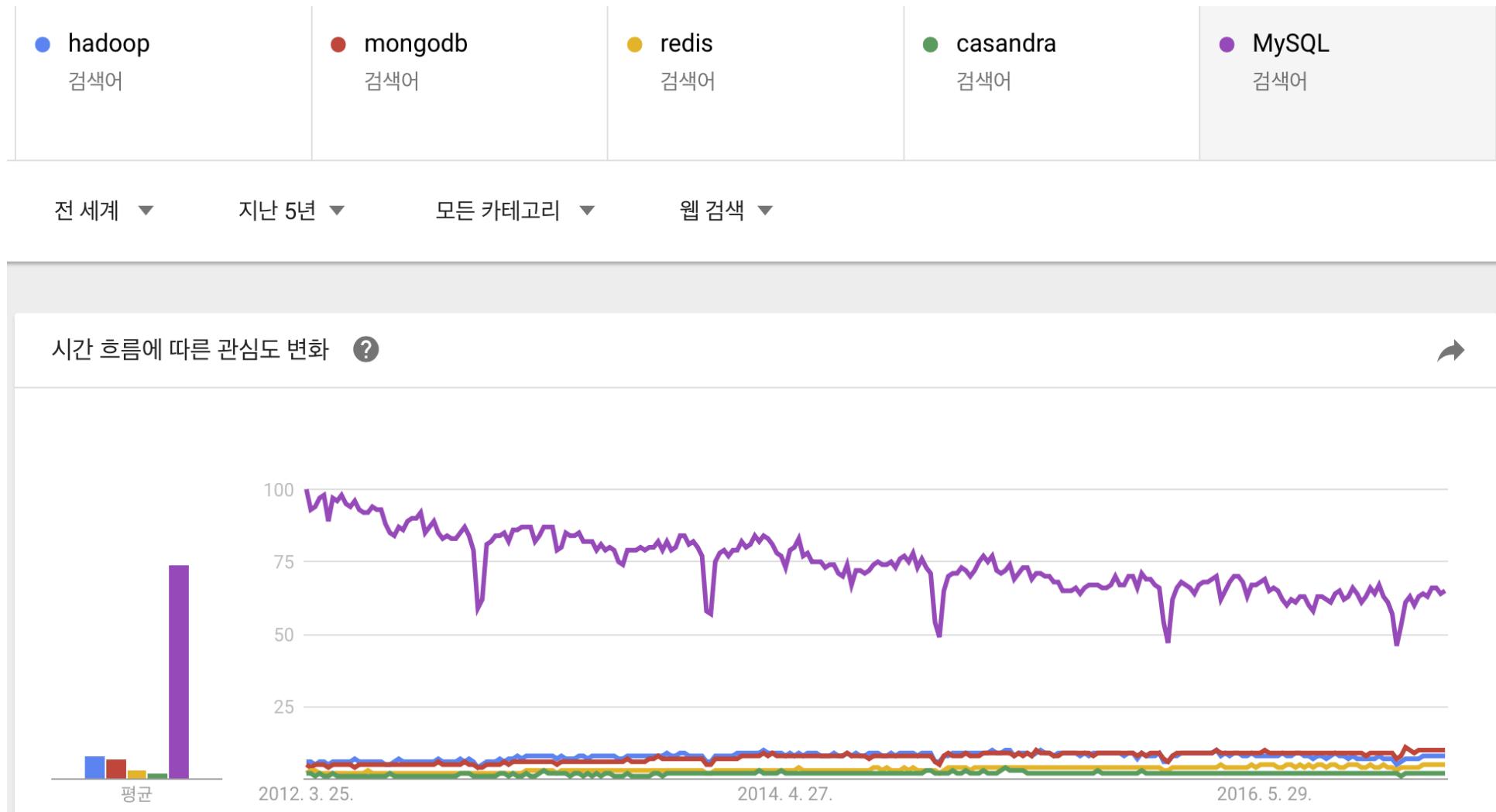
8.2 NOSQL 저장소(3/4) – 트랜드(3/3)

- ✓ NoSQL 저장소들은 지속적인 관심을 받고 있으며, 그 중에서 MongoDB에 대한 관심도가 증가하고 있습니다.



8.2 NOSQL 저장소[4/4] – 트랜드[3/3]

- ✓ 여전히 RDB가 대세를 이루고 있지만, 그 관심도는 지속적으로 떨어지고 있습니다.



8.3 NoSQL 비즈니스 드라이버

✓ 용량(Volume),

- 일반 프로세서로 구성한 클러스터를 사용한 빅데이터 검색 수요가 있음
- Power wall 현상으로 단일 칩의 속도를 높이기 보다는 여러 프로세서 협업 방식을 선호함
- 즉, scale up(빠른 프로세스를 통해) 보다는 scale out(수평확장)을 선호함
- 따라서, 조직은 직렬 처리에서 병렬 처리로 진화하고 있으며, 그에 따라 데이터는 서로 다른 프로세서로 나누어 처리함

✓ 속도(Velocity),

- 단일 프로세서 RDBMS로는 빠른 입력(insert), 읽기(read) 등에 한계가 있어, 리얼타임 요구를 충족하지 못함
- 특히 트래픽이 폭증하는 웹 시스템의 백-엔드 시스템에 RDBMS를 사용할 경우, 급격한 성능저하를 겪음

✓ 다양성(Variability),

- 고객의 새로운 데이터를 추가할 경우, RDBMS에 새로운 컬럼이 추가되고, 이는 대용량 데이터에서 많은 시간과 비용을 요구함
- 다양한 유형의 비정형 데이터, 복잡한 관계 데이터, 바이너리 데이터 등을 쉽고 빠르게 처리할 수 있는 요구가 늘어남

✓ 민첩성(Agility),

- 객체로 표현되는 도메인의 데이터를 테이블로 표현되는 RDBMS 데이터로 변환하는데 많은 시간과 노력이 필요함
- 데이터를 RDBMS에 관리하기 위해서는 정교한 모델링(스키마 구성), 객체 관계 매핑, 등의 복잡한 과정을 거쳐야 하며, 변경이 발생할 경우, 이 절차를 모두 반복해야 할 뿐만 아니라 기존의 데이터 변환도 해야 하므로 많은 시간이 필요함
- RDB기반의 개발의 경우, 개발 초기에 모델링이 정확하게 되어 있어하므로 초기 모델링에 많은 시간이 필요함

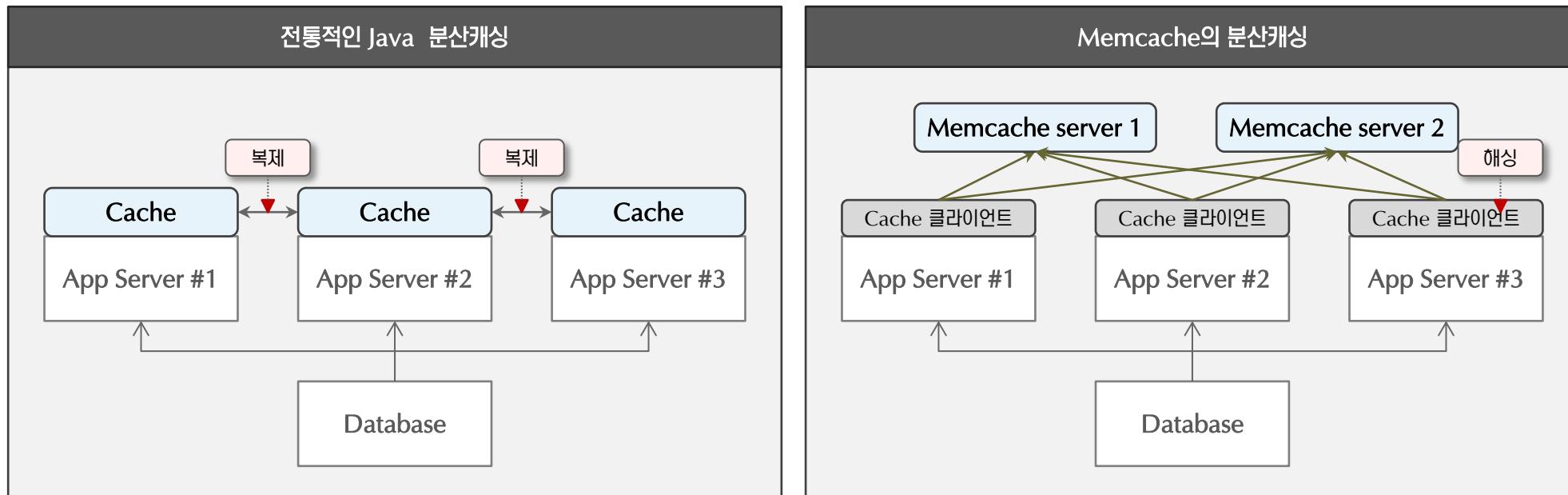
8.4 NoSQL 사례 연구(1/4)

- ✓ 시장환경은 끝없이 변하고 있으며, 기업은 고객의 요구를 충족하기 위해 경쟁적으로 노력하고 있습니다.
- ✓ 비용-효율적인 방식으로 기술을 활용하고 있으며, 전통적인 기술과는 거리가 먼 새로운 접근방법을 취합니다.
- ✓ 아래 사례들은 보다 빠르게, 보다 낮은 비용으로, 보다 효과적으로 문제를 해결하였습니다.

사례/표준	동인(Driver)	교훈(Finding)
LiveJournal의 Memcache	<ul style="list-style-type: none">▪ DB 성능 향상이 필요함	해싱과 캐싱을 사용함으로써, RAM의 데이터를 공유함으로써 DB로 가는 읽기 요청을 줄임으로써 성능을 비약적으로 향상시킴
Google의 맵리듀스	<ul style="list-style-type: none">▪ 저비용 하드웨어로 수십억 건의 웹 페이지에 대한 색인을 생성해야 함	다수의 일반 프로세서를 사용하여 병렬처리를 함으로써, 수십억 건의 웹페이지에 대한 색인을 신속하게 처리함
Google의 빅테이블	<ul style="list-style-type: none">▪ 분산 시스템에 테이블 형식의 데이터를 유연하게 저장해야 함	스파스 매트릭 접근방법을 사용하여, 사전 모델링 없이 수십억 건의 로우와 수백만 건의 컬럼을 가진 단일 테이블에 저장한 효과를 가져옴
Amazon의 다이나모	<ul style="list-style-type: none">▪ 웹에서 24x7으로 주문을 받아야 함	단순한 인터페이스를 가진 키-밸류 저장소는 대용량 데이터임에도 불구하고 쉽게 복제(replication)가 가능함

8.4 NoSQL 사례 연구(2/4): Live Journal의 Memcache

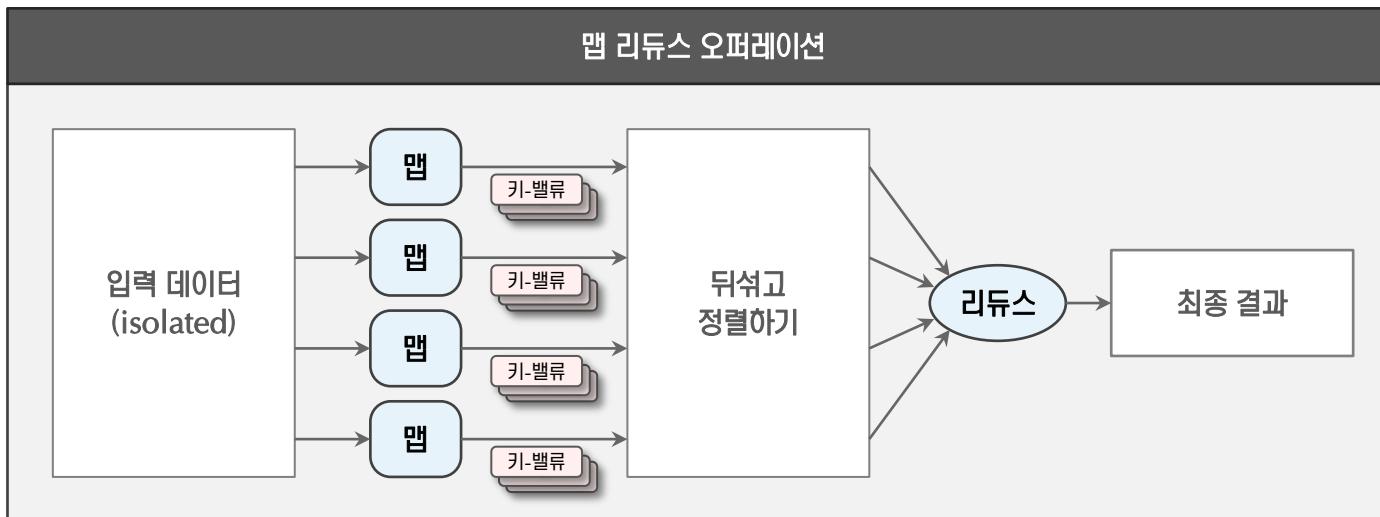
- ✓ LiveJournal은 급증하는 사용자와 컨텐츠를 감당하기 위해 서버를 지속적으로 증설하고 있었습니다.
- ✓ 성능을 높이기 위해 각 서버 별 캐시를 사용하였으나, 효율이 매우 낮았습니다.
- ✓ 문제를 스스로 해결한 후, 결과를 오픈하였으며(Memcache), 웹 프론트 통신을 표준화 하였습니다.(Memcached)



>> Key: SQL 문
>> 서버선택: Key의 hashCode() % 서버 갯수

8.4 NoSQL 사례 연구(3/4): 구글의 맵리듀스

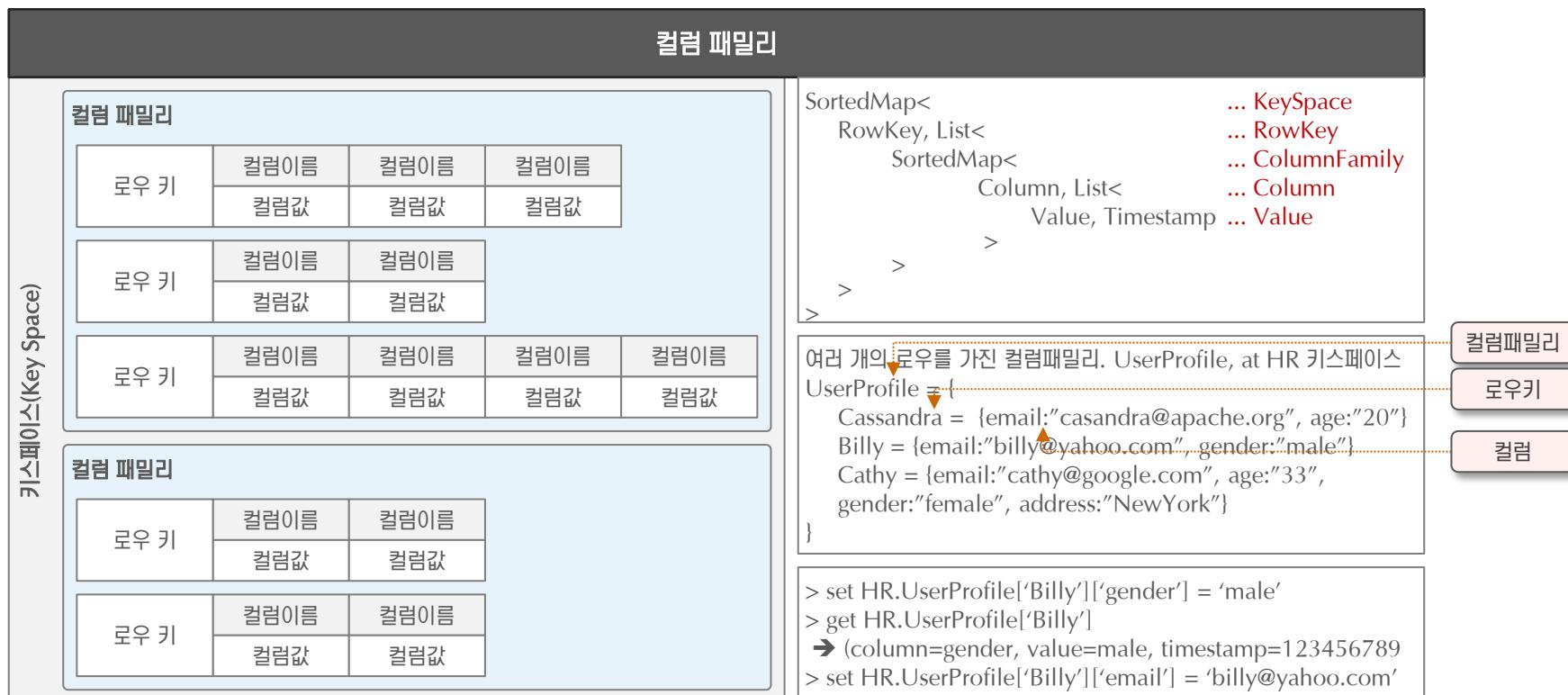
- ✓ 구글은 웹 데이터 컨텐츠를 저비용의 일반 CPU를 이용하여 변환하고 그 절차를 공유하였습니다.
- ✓ 맵과 리듀스 개념은 이전에 있었으며, 구글의 사례에서는 두 단계의 데이터 변환절차를 의미합니다.
- ✓ 맵 오퍼레이션은 데이터를 추출, 변환, 필터링하는 일을 수행하여 결과를 리듀스에게 넘겨줍니다.
- ✓ 리듀스 오퍼레이션은 최종결과를 얻기위해 맵의 결과를 정렬, 결합, 합산하는 작업을 수행합니다.



- 맵과 리듀스 기능은 대규모 데이터 세트를 작은 조각으로 나눔으로써, 격리된 상태에서 독립된 변환작업을 할 수 있음.
- 핵심은 각 기능을 격리시킴으로써 여러 서버로 확장(scale out)이 가능하는 것임

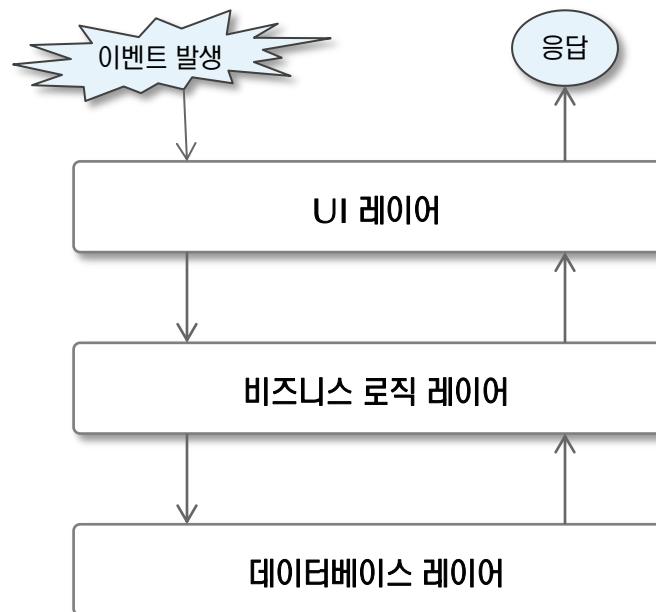
8.4 NoSQL 사례 연구(4/4): 구글의 BigTable

- ✓ “A Distributed Storage System for Structured Data.” 논문으로 발표하였습니다.
- ✓ 높은 응용성(applicability), 확장성(scalability), 높은 성능, 그리고 가용성(availability) 목표를 가지고 개발하였습니다.
- ✓ 구글 내에서만 60여개의 제품과 프로젝트에서 활용함으로써 그 가능성을 보여줌, 구글 Analytics, 구글 Earth, 등
- ✓ BigTable 프로젝트는 매우 성공적이었으며, 개발자들에게 대용량 데이터에 대한 단일 테이블 뷰를 제공하여 주었습니다.



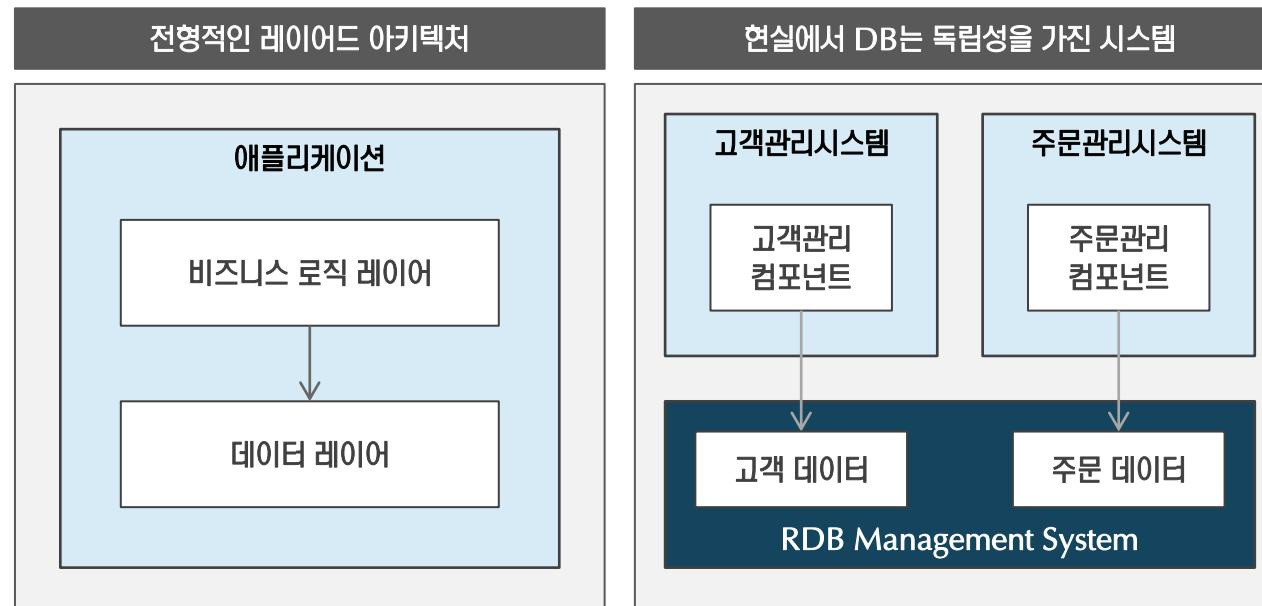
8.5 설계 단순화(1/5)

- ✓ 하나 이상의 애플리케이션 아키텍처를 객관적으로 평가할 때 레이어의 역할을 이해하는 것이 필요합니다.
- ✓ 두 시스템을 공평하고 객관적으로 비교하는 방법은 애플리케이션 전체를 조망하여 전체론적인 접근방법을 가지는 것입니다.
- ✓ 디어 안에서 외부의 영향 없이 변경이나 추가가 가능하여, 유연성과 재사용성을 높여 주었습니다.
- ✓ 관심사의 분리(Separation of Concern) 원칙



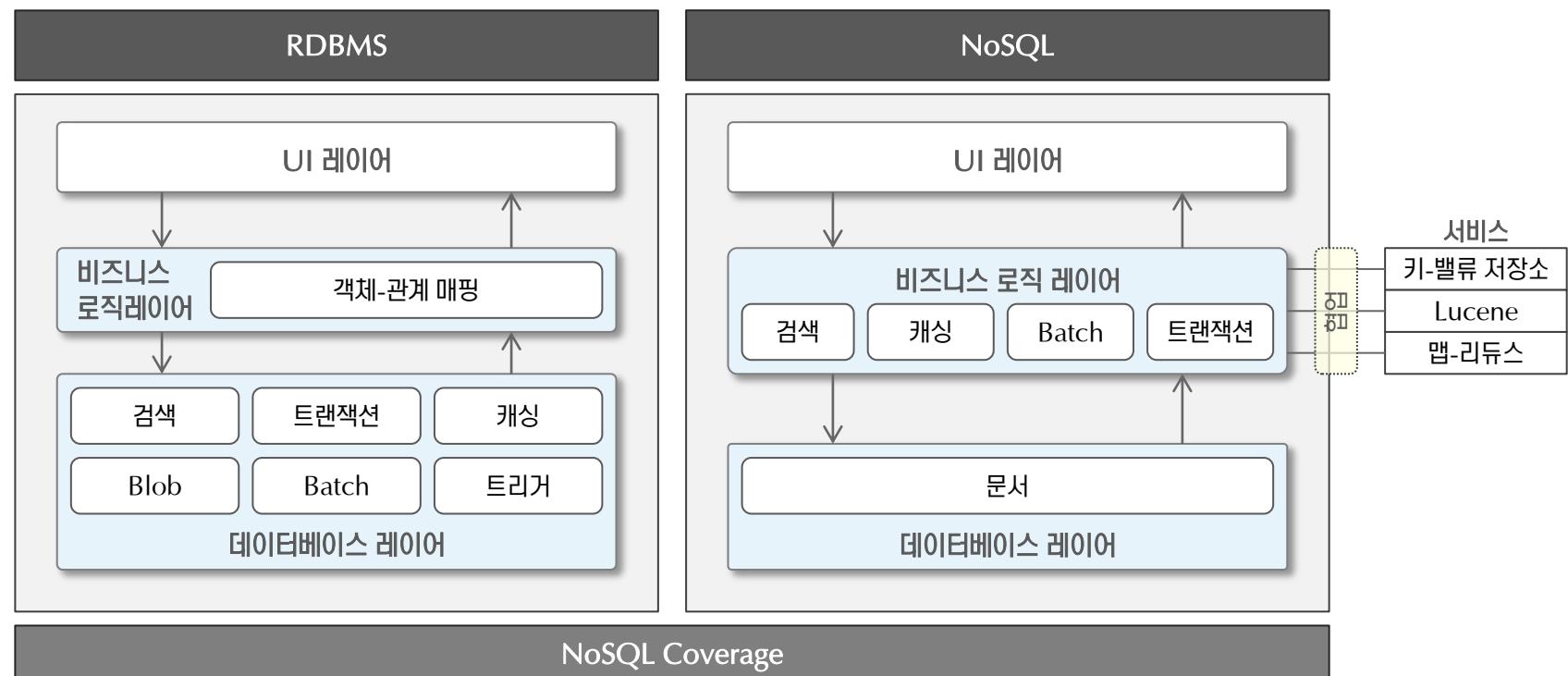
8.5 설계 단순화(2/5)

- ✓ 전형적인 레이어드 아키텍처에서 데이터 레이어는 애플리케이션 컨텍스트 안에 존재합니다.
- ✓ 하지만, 현실에서 데이터베이스는 애플리케이션과는 독립적인 관리 시스템으로 설계되고 운영됩니다.
- ✓ 그 결과 RDB는 독립적인 시스템에 요구되는 많은 기능을 수행하게 되었습니다.
- ✓ 현실속의 RDBMS는 데이터 자원을 관리하는 역할과 데이터를 사용하는 여러 애플리케이션을 통합하는 역할을 수행합니다.



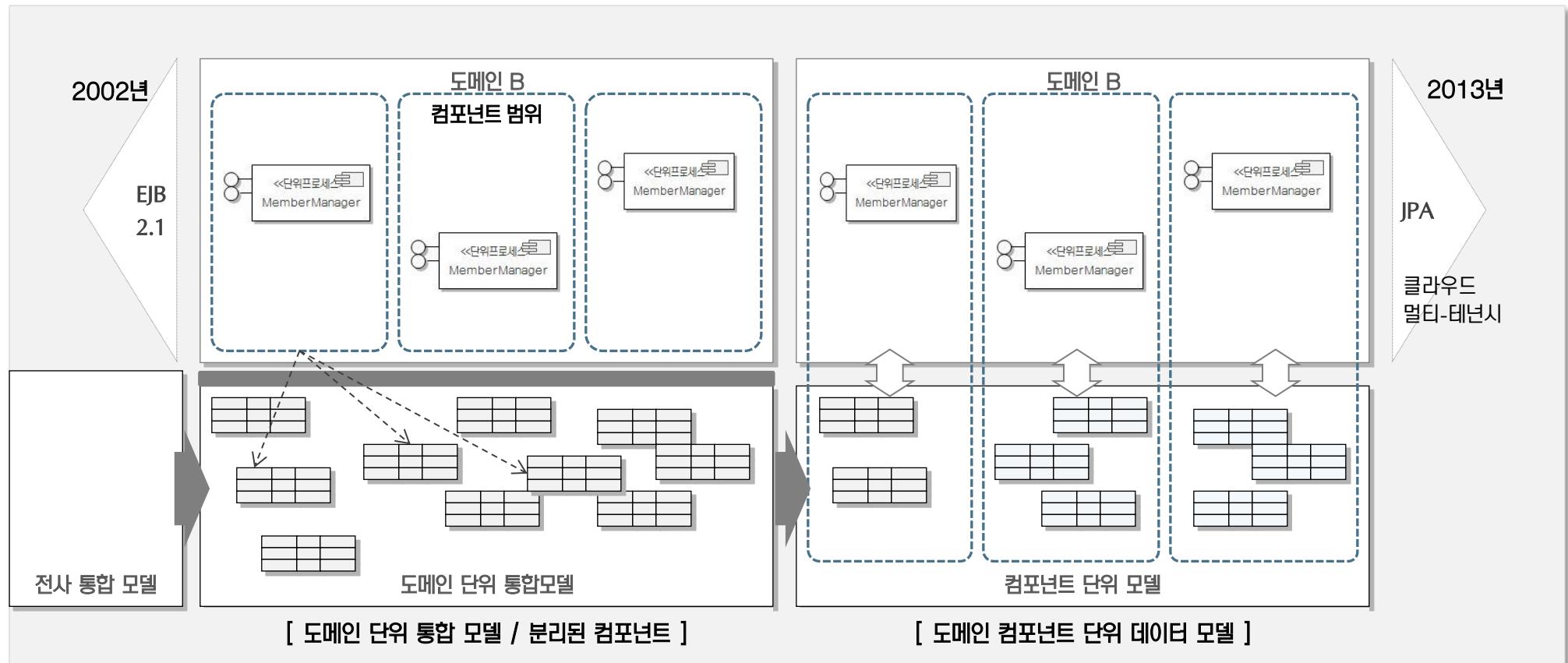
8.5 설계 단순화(3/5)

- ✓ RDBMS에서 애플리케이션 기능이 RDBMS에 집중되어 있지만,
- ✓ NoSQL에서는 비즈니스 로직레이어로 대부분의 기능이 있고, 데이터베이스 티어는 단순한 저장소 역할을 수행합니다.
- ✓ NoSQL에서 다양한 서비스를 이용하여 기존 RDBMS가 제공하는 기능을 대체합니다.



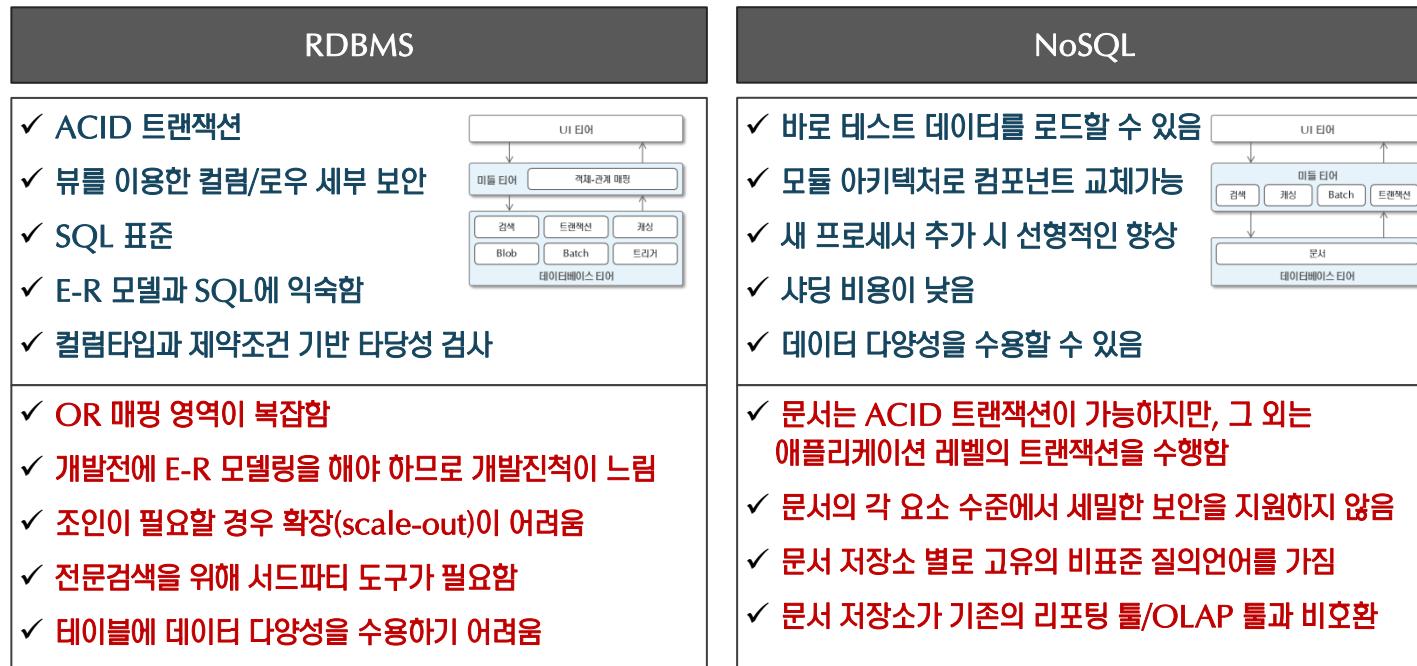
8.5 설계 단순화(4/5)

- ✓ 기술의 발전과 데이터의 복잡성 증가에 따라 데이터 모델링 범위는 전사 통합 → 도메인 통합 → 컴포넌트로 범위가 줄어듦
- ✓ 기업의 데이터 모델은 대체로 [도메인 통합모델]이므로 각 업무별로 별도의 모델을 가집니다.
- ✓ 기존 데이터 유지가 중요한 곳은 도메인 통합, 새로운 모델이 필요한 곳은 컴포넌트 단위 모델을 지향합니다.



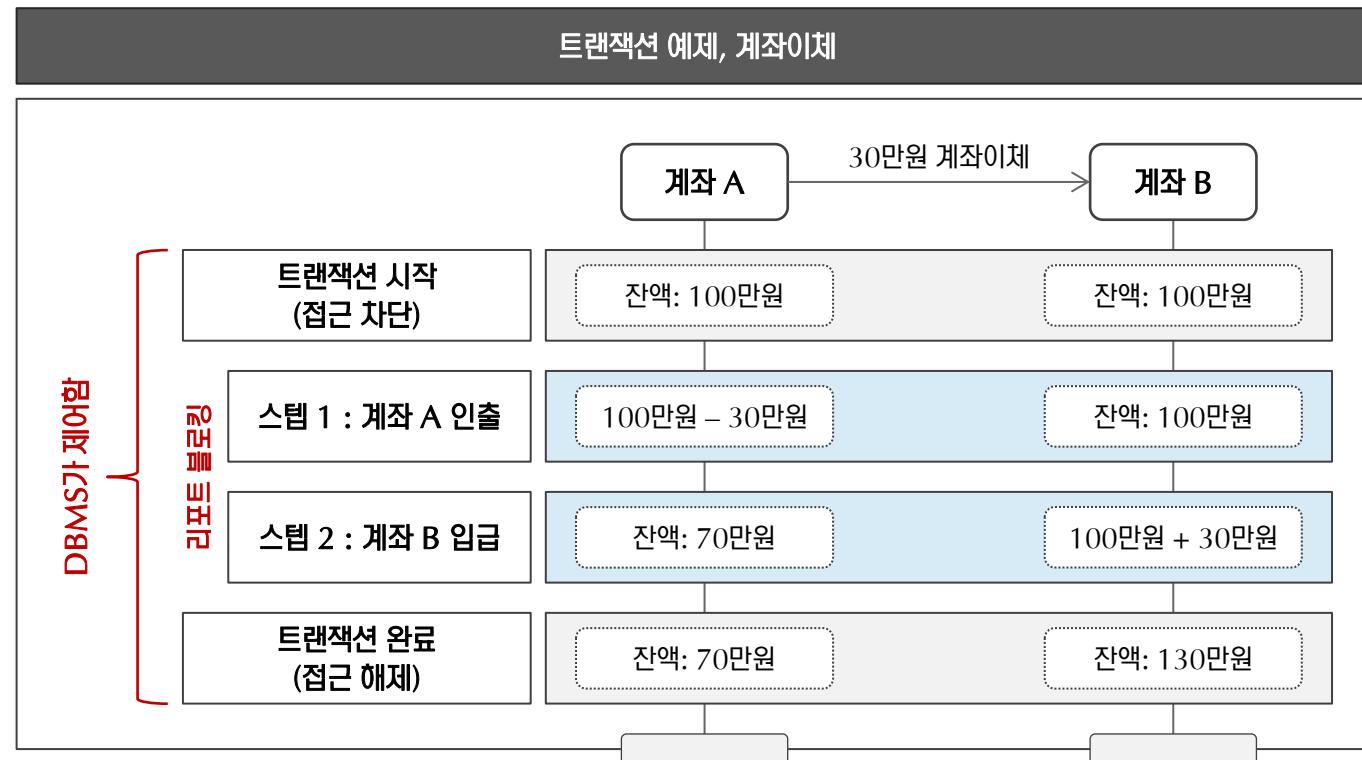
8.5 설계 단순화(5/5)

- ✓ 애플리케이션 기능을 어느 티어에 두는가는 아키텍처 설계의 주요 의사결정 사항입니다.
- ✓ NoSQL을 채택했다고 해서 RDBMS를 배제하는 것이 아니므로, 양쪽 모두 의사결정 시 고려하여야 합니다.
- ✓ NoSQL은 데이터베이스 관점이 아닌 애플리케이션 관점에서의 설계라는 특징이 있습니다.



8.6 DB 트랜잭션(1/4) – ACID와 BASE

- ✓ 분산 컴퓨팅 환경에서 성능과 일치성(consistency) 관점에서 트랜잭션은 매우 중요합니다.
- ✓ RDB는 ACID 트랜잭션을 사용하고, NoSQL은 BASE 트랜잭션을 사용하며, 둘 간의 차이는 애플리케이션 개발자가 얼마나 수고를 해야 하는가와 트랜잭션 제어의 위치에 있습니다.
- ✓ 트랜잭션은 “All or nothing” 특성을 가지고 있으며, 대표적인 예로 두 은행 계좌 간의 계좌이체를 들 수 있습니다.



8.6 DB 트랜잭션(2/4) – ACID 트랜잭션

- ✓ 자원 잠금 → 자원 복사 → 트랜잭션 수행 → 자원 해제절차를 거치는 트랜잭션은 매우 복잡하고 어려운 기능입니다.
- ✓ DB 제품들은 초기 버전에서 DB레벨 트랜잭션을 지원하지 않다가 성숙도가 높아지면서 지원하는 경향이 있습니다.
- ✓ 다수의 DB들이 단일 CPU에서 트랜잭션을 제한함, 네트워크에는 수많은 실패가능성이 존재하고 있기 때문입니다.

Atomicity	은행계좌 이체 예제에서 두 계좌 간의 자금이체는 “all-or-nothing” 트랜잭션이어야 한다. 트랜잭션 간에 디스크 오류, 네트워크 실패, 하드웨어 오류, SW 오류 등이 발생할 수 있다. 이러한 오류를 극복하고 완료를 하여 성공을 하든, 실패를 하여 원래로 돌려놓든 해야 한다.
Consistency	계좌이체의 예에서 두 계좌에서 이체가 발생할 경우, 계좌의 전체 잔고가 변경되어서는 안된다. DB는 atomic 오퍼레이션으로 진행되는 계좌이체 중에는 모든 리포트를 차단한다. 리포트를 허용하면 트랜잭션 처리 속도에 영향을 줄 수 있다.
Isolation	한 트랜잭션의 각 부분(또는 단계)은 트랜잭션의 다른 부분과 관계없이 독립적으로 실행되어야 한다. 계좌이체의 예에서 계좌입금은 계좌출금에 대해서 전혀 알지 못하는 상태, 즉 독립적으로 실행되어야 한다.
Durability	일단 트랜잭션이 완료되고 나며 그 결과는 영속적이어야 한다. 시스템 크래시가 발생하여 백업을 할 경우에도, 트랜잭션이 완료된 상태가 보장되어야 한다.

8.6 DB 트랜잭션(3/4) – BASE 트랜잭션

- ✓ NoSQL은 자원 차단(blocking)과 그로인한 대기 시간에 반대하는 접근방법을 가집니다.
- ✓ 몇 분간의 데이터 불일치를 막기위해 웹스토어의 주문을 받지 않거나 지연시키면 안됩니다.
- ✓ 전세계에서 쓸아지는 주문을 ACID 트랜잭션 기반으로 처리하는 것은 거의 불가능에 가깝습니다.
- ✓ RDB가 일치성(consistency)에 초점을 두고 있다면, NoSQL은 가용성(availability)에 초점을 두고 있습니다.

Basic availability	일시적인 불일치를 허용하고 트랜잭션은 관리가능하다. BASE 시스템에서 정보와 서비스 역량은 기본적으로 가용하다.
Soft-state	일시적인 불일치를 허용하고, 데이터는 리소스 소비를 줄이기 위해 사용중에도 변경될 것이다.
Eventual consistency	모든 서비스 로직이 실행되었을 때, 결국 시스템은 일치된 상태가 될 것이다.

8.6 DB 트랜잭션(4/4) – BASE 트랜잭션

- ✓ BASE 트랜잭션은 비록 짧은 기간은 동기가 깨지더라도 새로운 데이터를 입력받는 것이 목표입니다.
- ✓ 잠금과 해제 기능이 없으므로 간단하고 빠르며, 트랜잭션이 끝나지 않더라도 리포트를 허용합니다.
- ✓ NoSQL 시스템은 흩어져 있으므로 ACID를 보장할 필요가 없습니다.

ACID	BASE
<ul style="list-style-type: none">✓ 트랜잭션을 세부적으로 잘 처리함✓ 작업 중에는 모든 레포트를 차단함✓ 무엇이든 잘못 될 수 있다는 비관적인 입장✓ 상세 테스트와 실패모드 분석✓ 차단과 해제의 반복	<ul style="list-style-type: none">✓ 쓰기는 절대로 차단하지 않음✓ 일치성이 아니라, 처리에 초점을 둠✓ 서비스가 실패할 수 있겠지만 결국 바로 처리될 것이라는 낙관적인 입장✓ 일부 리포트는 일시적으로 일치하지 않을 수 있으나 신경쓰지 않음✓ 모든 것에 단순함을 유지하고 차단을 피함

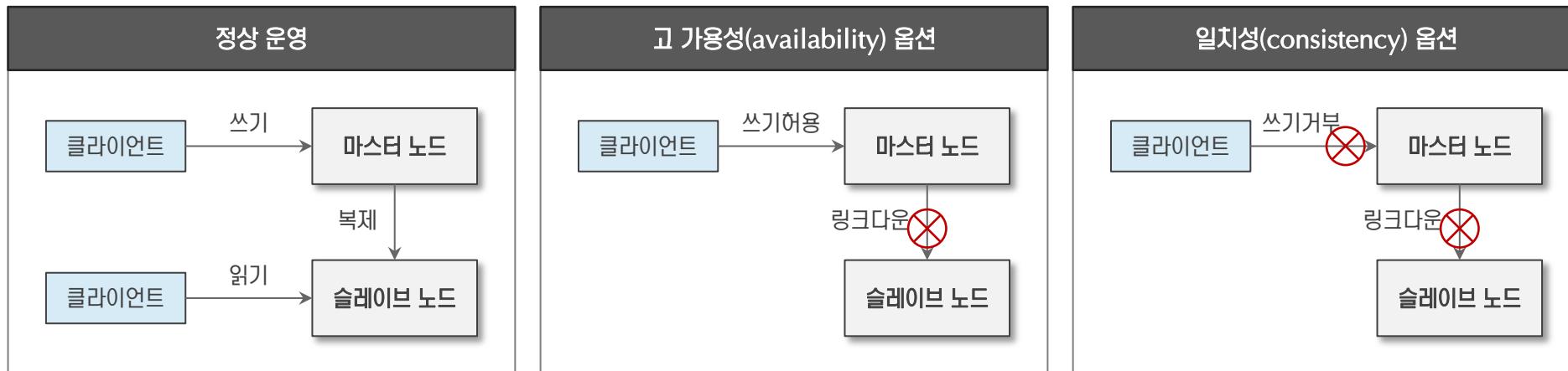
8.7 CAP이론(1/3)

- ✓ 시스템 실패의 경우, 신뢰할 수 없는 네트워크 상에서 분산시스템을 다룰 때, 일치성과 가용성 특성을 고려해야 합니다.
- ✓ CAP 이론(Eric Brewer, 2000)은 어떤 분산시스템이든 세 가지 요구특성 중에 두 가지는 가지는 가질 수 있습니다.
- ✓ CAP 이론은 클러스터 안의 파티션 간에 연결이 끊어졌을 때만 적용하는 것입니다.
- ✓ CAP 이론은 네트워크 실패 상황에서 가용성-일치성 스펙트럼에서 생각할 기준을 제공합니다.

Consistency	하나의, 최신의, 읽기 버전의 데이터가 모든 클라이언트에게 가용한 것. 이것은 ACID에서 말하는 일치성(consistency)과는 다르다. 여기서의 Consistency는 복제된 파티션에서 여러 클라이언트가 동일한 아이템을 읽었을 때 일치된 결과를 보여주는 것을 의미한다.
High Availability	분산 DB는 DB 클라이언트들이 항상 지연없이 항목을 갱신하도록 할 것임을 암시하는 것. 복제된 데이터 간의 내부 통신 실패는 갱신을 막지 못한다.
Partition tolerance	DB 파티션 간에 통신실패가 있더라도 클라이언트 요청에 응답을 반드시 할 수 있는 역량. 이것은 노드의 부분들 간에 링크가 제대로 동작하지 않더라도 사람들이 지적인 대화를 나눌 수 있는 것과 유사하다.

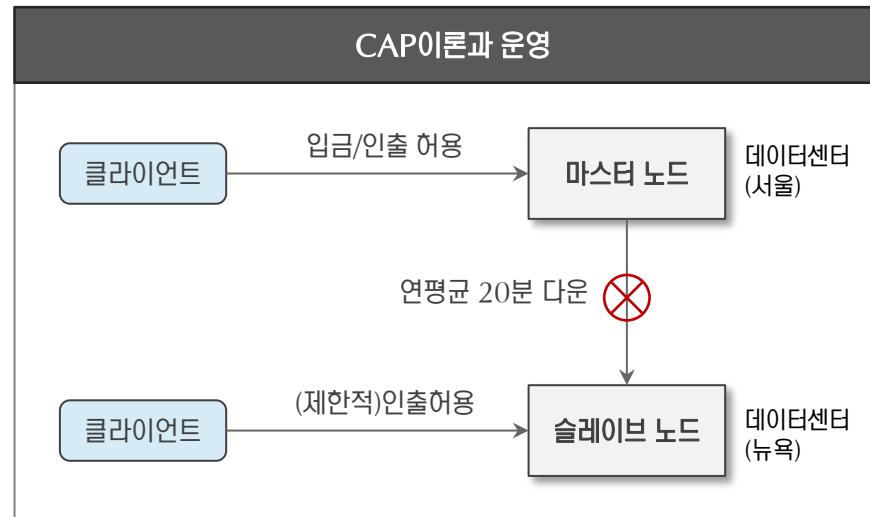
8.7 CAP이론(2/3)

- ✓ 네트워크 실패 시, 가용성 대 일치성 중에서 상대적으로 장점이 있는 것을 결정하는데 도움을 줍니다.
- ✓ 일반적인 상황에서 마스터 노드에 쓰기를 하고, 그 결과는 슬레이브 노드로 복제되며, 슬레이브 노드를 읽습니다.
- ✓ 고가용성 옵션은 마스터와 슬레이브 간에 불일치가 발생하지만 가용성을 높이기 위해 쓰기를 허용합니다.
- ✓ 일치성 옵션은 마스터와 슬레이브 간의 불일치를 허용하지 않으며, 따라서 마스터에 대한 쓰기를 허용하지 않습니다.



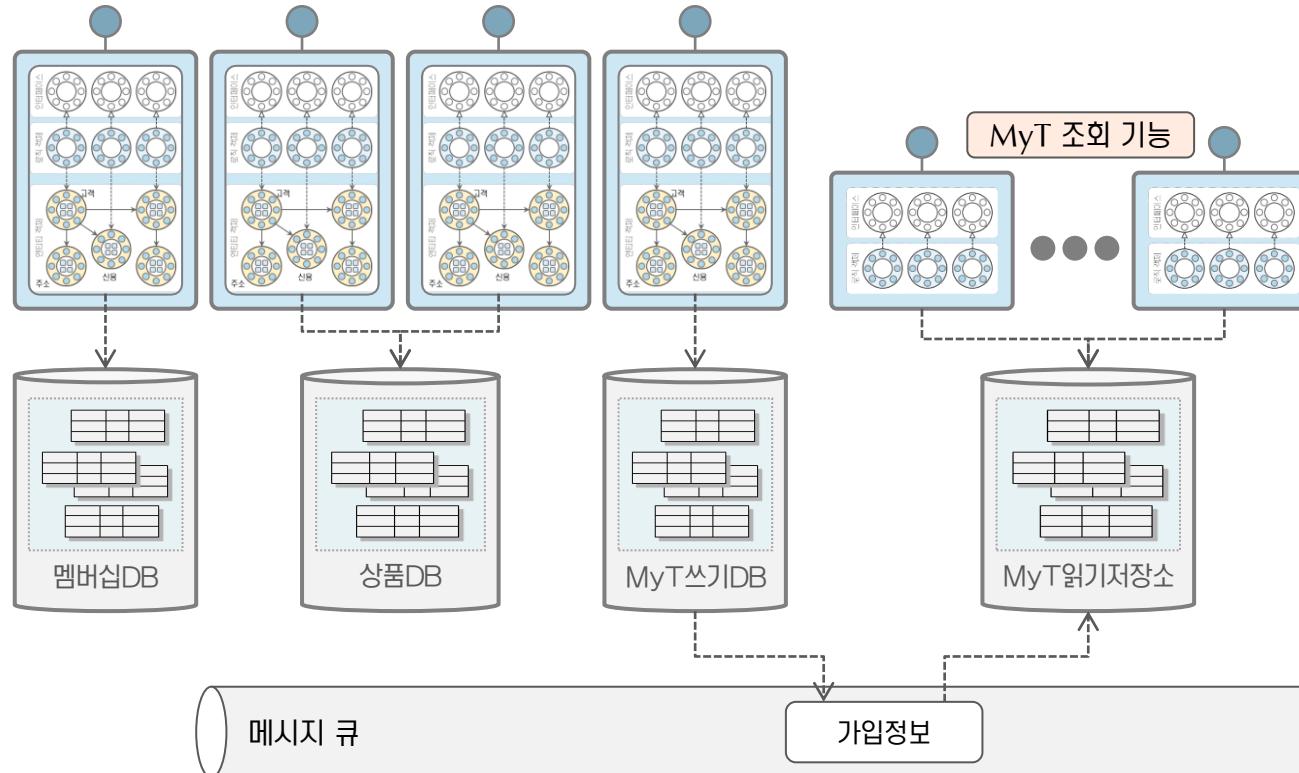
8.7 CAP이론(3/3)

- ✓ 서울의 데이터 센터에 마스터 노드가 있고, 뉴욕의 데이터 센터에 슬레이브 노드가 있습니다.
- ✓ 두 데이터 센터 간에는 연평균 20분 정도의 네트워크 실패가 발생합니다.
- ✓ 네트워크 실패 시, 고 가용성(availability) 옵션을 선택하여, 마스터 노드에 입금과 인출을 허용하고,
- ✓ 슬레이브 노드에서는 가용성을 최대한 높이되 리스크를 줄이기 위해 제한된 인출을 허용합니다, 예 10만원 한도



8.8 CQRS

- ✓ 마이크로 서비스는 개별 컨테이너 구조로 빠른 배포와 기능별 최적화가 가능합니다.
- ✓ CQRS 아키텍처 패턴은 읽기, 쓰기 기능을 별도의 서비스로 구성하여 상황에 적합한 스케일을 지원합니다.
- ✓ 각 서비스 별 부하에 따른 동적인 Scale-out 은 빠른 Performance를 보장하여 줍니다.
- ✓ 처리 내용이 다르면(Command:CUD, Query: Read only) 기술 구조도 다르게 설계합니다.



CQRS
(Command Query
Responsibility Segregation)

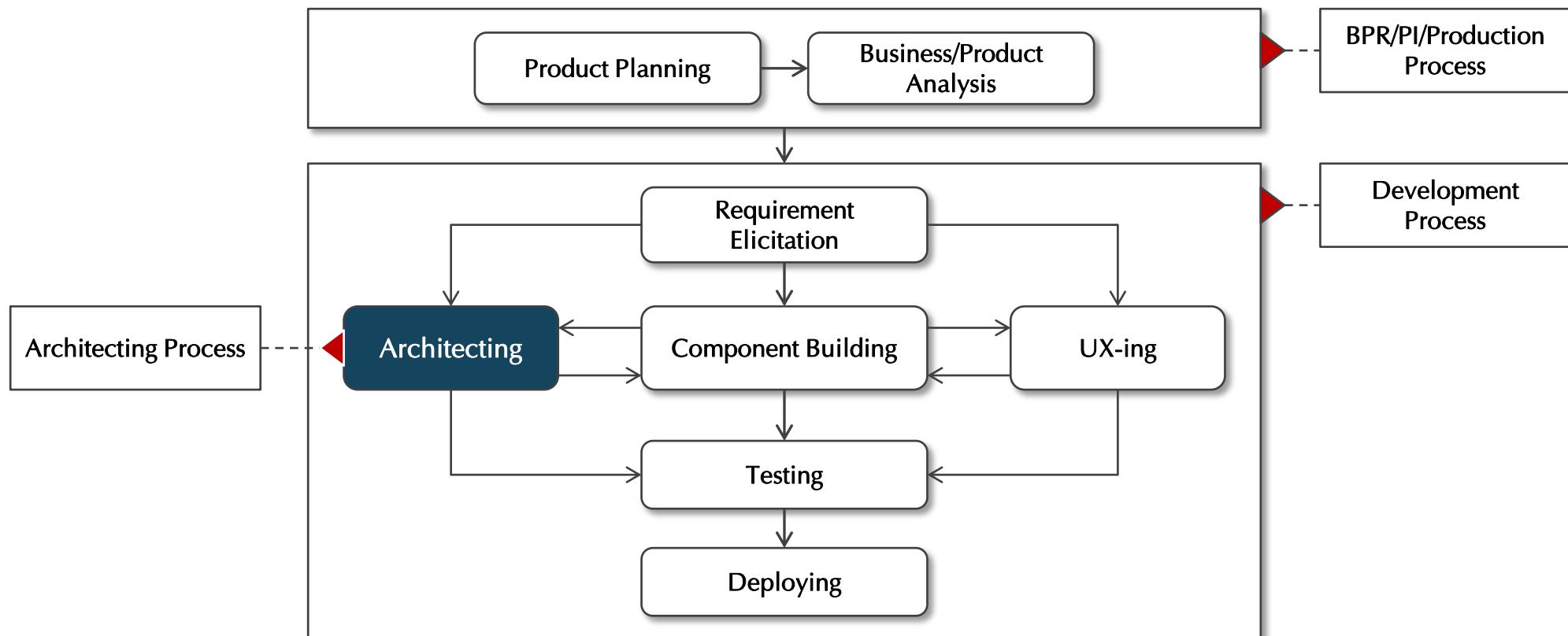


9. 프레임워크

-
- 9.1 개발 절차
 - 9.2 아키텍팅 프로세스
 - 9.3 아키텍처
 - 9.4 아키텍처 요소 획득
 - 9.5 빌드 사례 1
 - 9.6 실습 9-1

9.1 개발 절차

- ✓ SW를 기반으로 하는 제품이나 업무용 Application을 개발할 때, 다양한 프로세스가 필요합니다.
- ✓ 아키텍팅 활동의 목표 시스템의 틀과 원칙을 설계하는 중요한 활동입니다.
- ✓ 아키텍팅 프로세스는 개발 프로세스의 하위 프로세스이며, 아키텍처 설계, RI 등을 포함합니다.

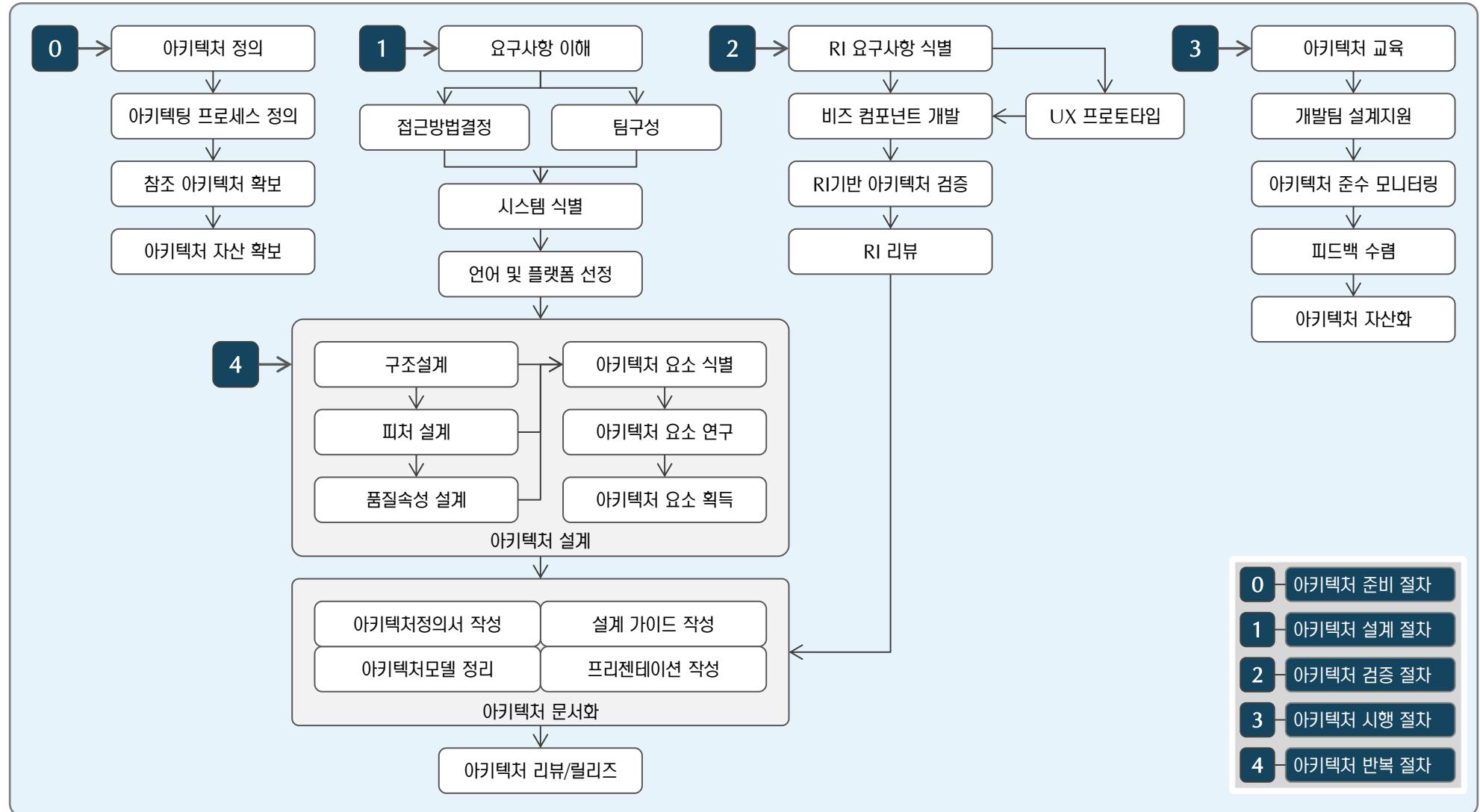


9.2 아키텍팅 프로세스(1/3)

✓ 아키텍팅 프로세스는 IEEE 1471 확장 모델을 기반으로 합니다.

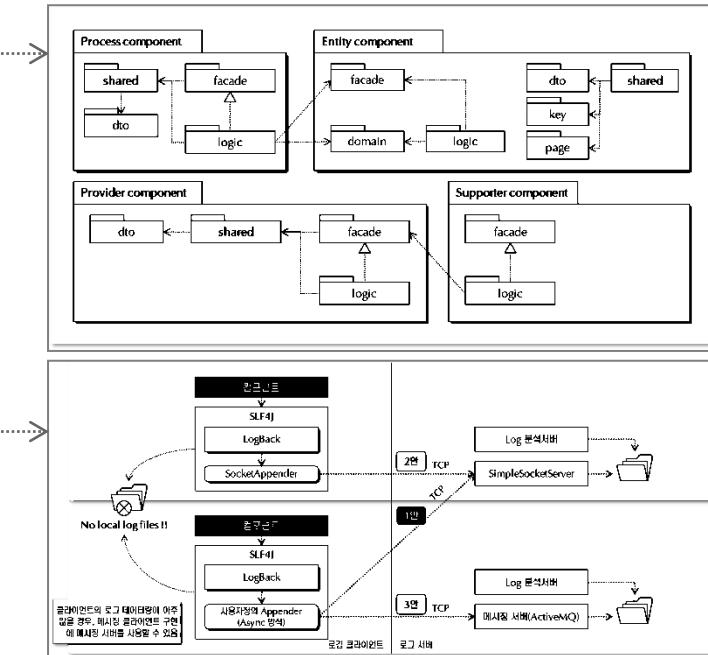
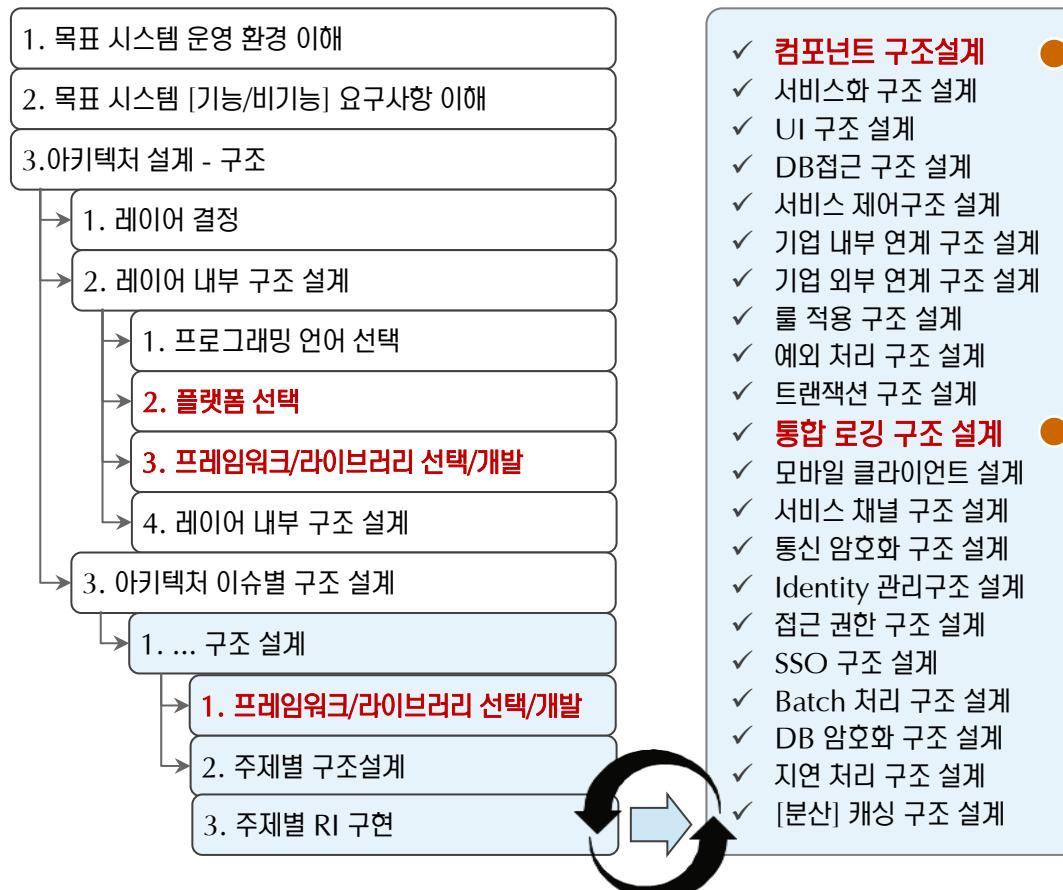
2011년

5. IEEE 1471 확장 모델 기반 아키텍팅 프로세스



9.2 아키텍팅 프로세스(2/3)

- ✓ 현대의 [비즈니스 지원] 시스템 설계 절차는 전체 구조 설계로부터 시작하여 각 주제별 구조 설계로 이어집니다.
- ✓ 시스템의 구조를 설계할 때, 주요 부분인 플랫폼, 프레임워크, 라이브러리를 대부분 오픈 소스 진영으로 부터 가져옵니다.
- ✓ 따라서 과거의 아키텍트는 설계(design) 역량이 중요했다면, 현대의 아키텍트는 선택(selection) 역량이 중요합니다.
- ✓ 올바른 선택을 위해서는 선택 대상에 대한 다양한 경험과 지식이 필요하며, 새로운 요소의 경우 빠른 이해가 중요합니다.



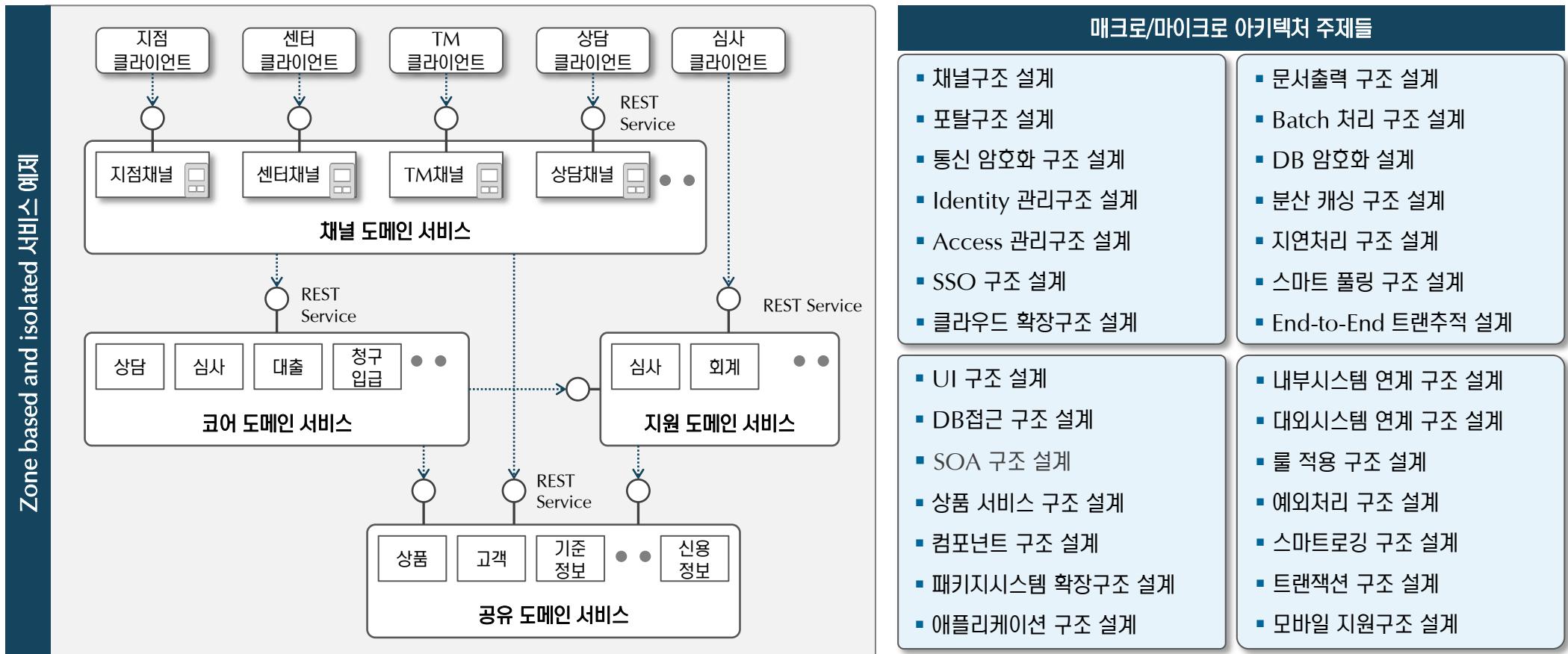
9.2 아키텍팅 프로세스(3/3) – 획득의 변화

- ✓ 핵심 아키텍처 요소(architectural element)의 유형은 플랫폼, 프레임워크, 라이브러리, 비즈니스 컴포넌트 입니다.
- ✓ SW 기술의 발전에 따라 각 요소들은 사용자 개발 → 3rd party → 오픈소스 방식으로 획득방법이 변경되어 왔습니다.
- ✓ 따라서 좋은 아키텍처 설계의 개념도 주요 요소 직접 설계(design)에서 선택(selection)으로 이동하고 있습니다.



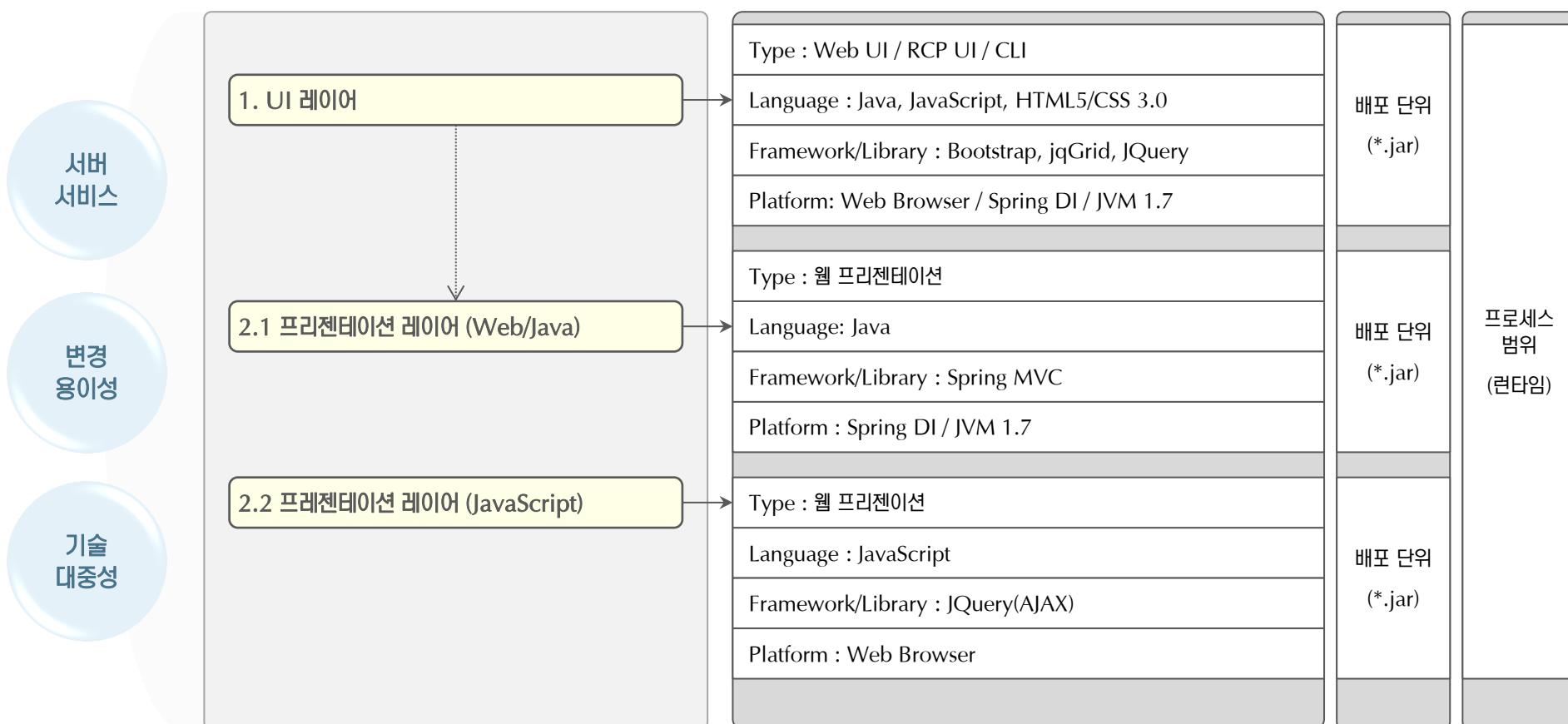
9.3 아키텍처(1/3)

- ✓ 기업의 전사 시스템에서 이상적으로 생각하고 설계한 매크로 아키텍처의 예상 모습입니다.
- ✓ 어떤 주제는 전체를 가로지르는 것이고, 어떤 주제는 특정 영역을 위한 설계입니다.
- ✓ MSA 접근방법에서는 마이크로서비스 자체와 매크로 서비스 주제로 나눌 수 있습니다.



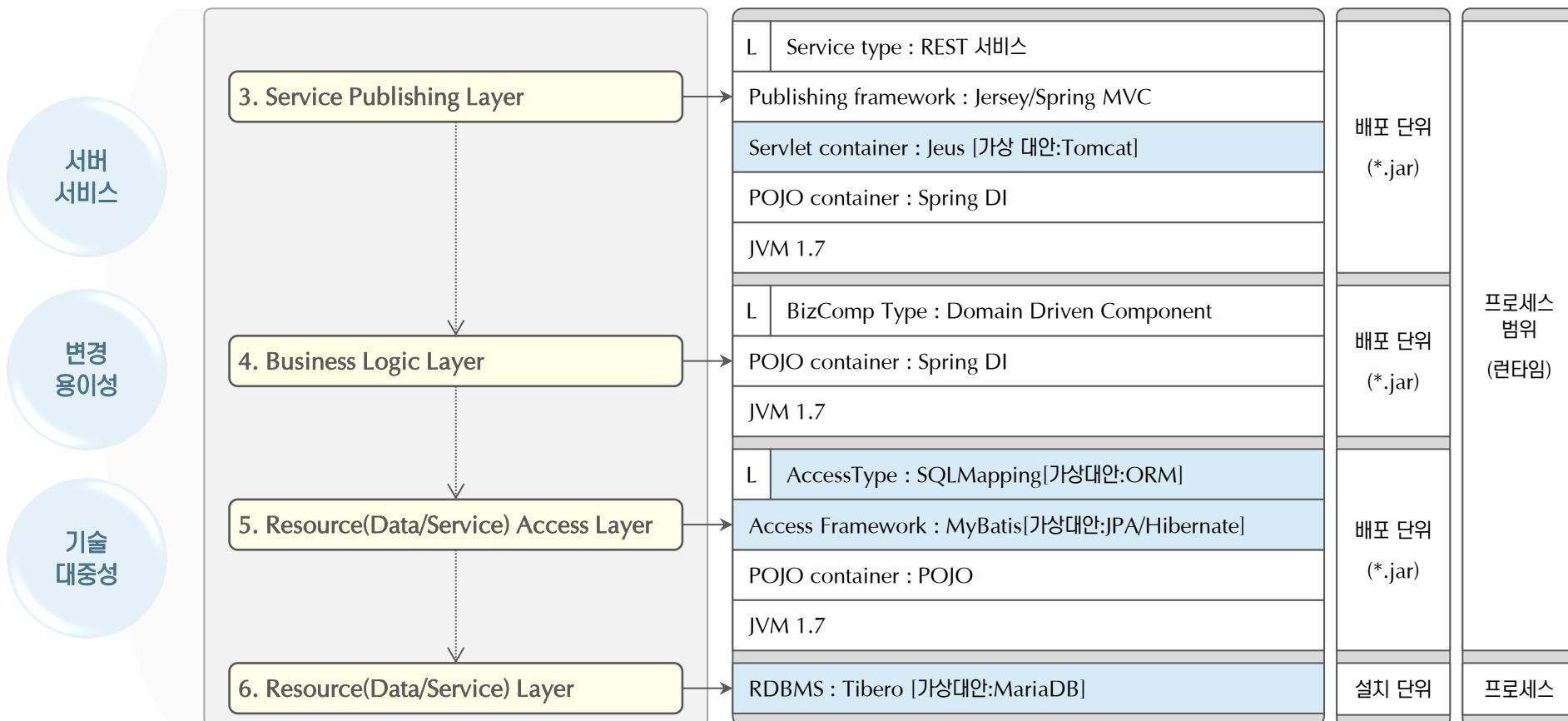
9.3 아키텍처(2/3) – 레이어 별 기술 1

- ✓ 각 레이어는 레이어 모듈을 구성하는 아키텍처 요소(시스템을 구성하는 기술 요소)와 관련이 있습니다.
- ✓ 기술 선택의 기준은 “서버 서비스”, “변경 용이성”, “기술 대중성”이며, 서버 서비스는 안정성과 관리 용이성을 의미합니다.
- ✓ 상황에 따라 세 개의 레이어(서비스 발행, 비즈니스 로직, 자원 접근)별로 배포 컴포넌트를 구성합니다.
- ✓ 배포 전략에 따라 다양한 배포 단위로 구성할 수 있다는 가정 하에 아키텍처를 설계하였습니다.



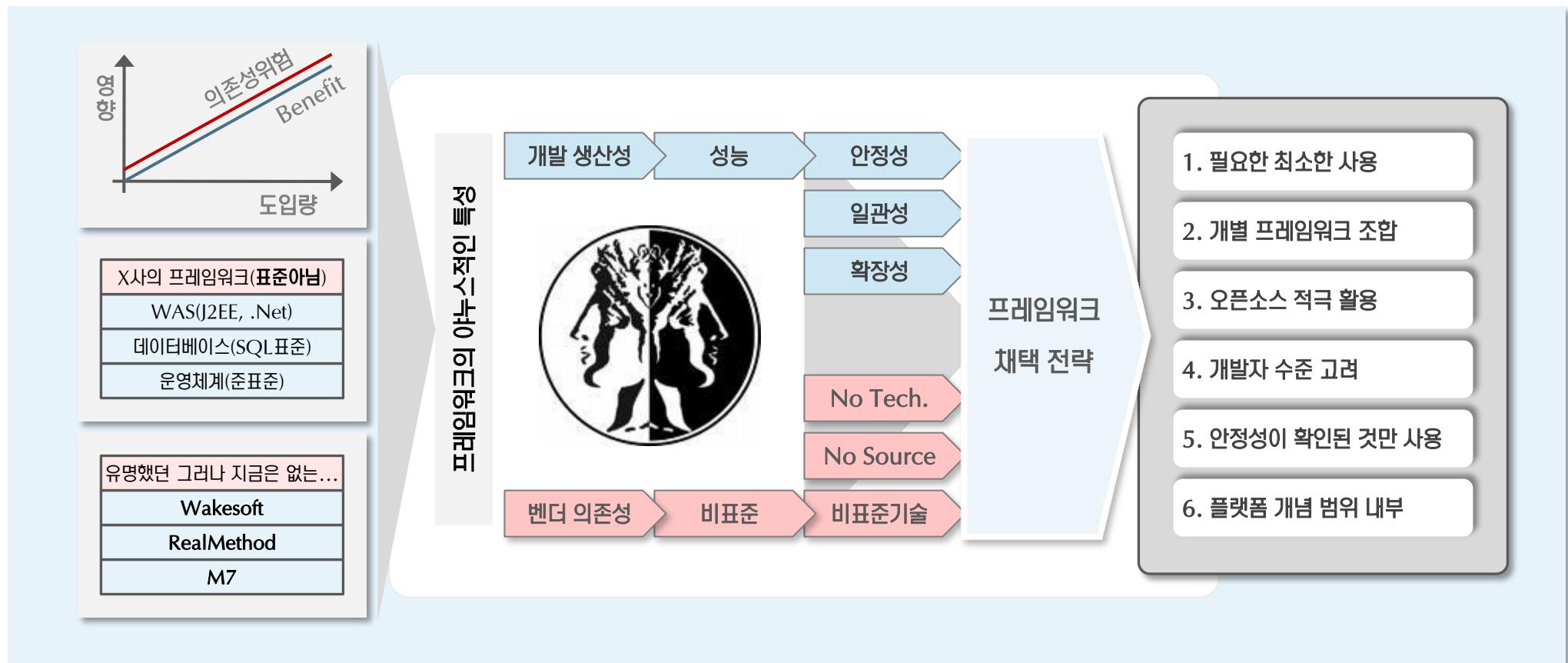
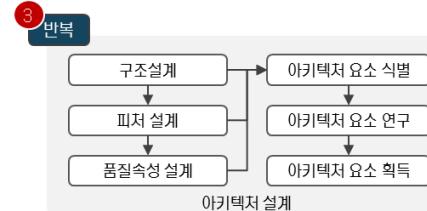
9.3 아키텍처(3/3) – 레이어 별 기술 2

- ✓ UI와 프리젠테이션 레이어는 기술 요소가 변경될 경우, 변경 내역이 많아 새로 개발해야 하는 경우도 있습니다.
- ✓ 레이어 설계를 통한 확장성과 이식성을 높이기 위해서 비즈니스 로직 레이어를 지키는 것이 가장 중요합니다.
- ✓ 비즈니스 로직 레이어에 FIDS의 핵심이 모두 들어가 있기 때문입니다.
- ✓ 따라서 비즈니스 로직 레이어를 제외한 나머지 레이어에서 다양한 대안(alternative) 기술을 가정하고 설계합니다.



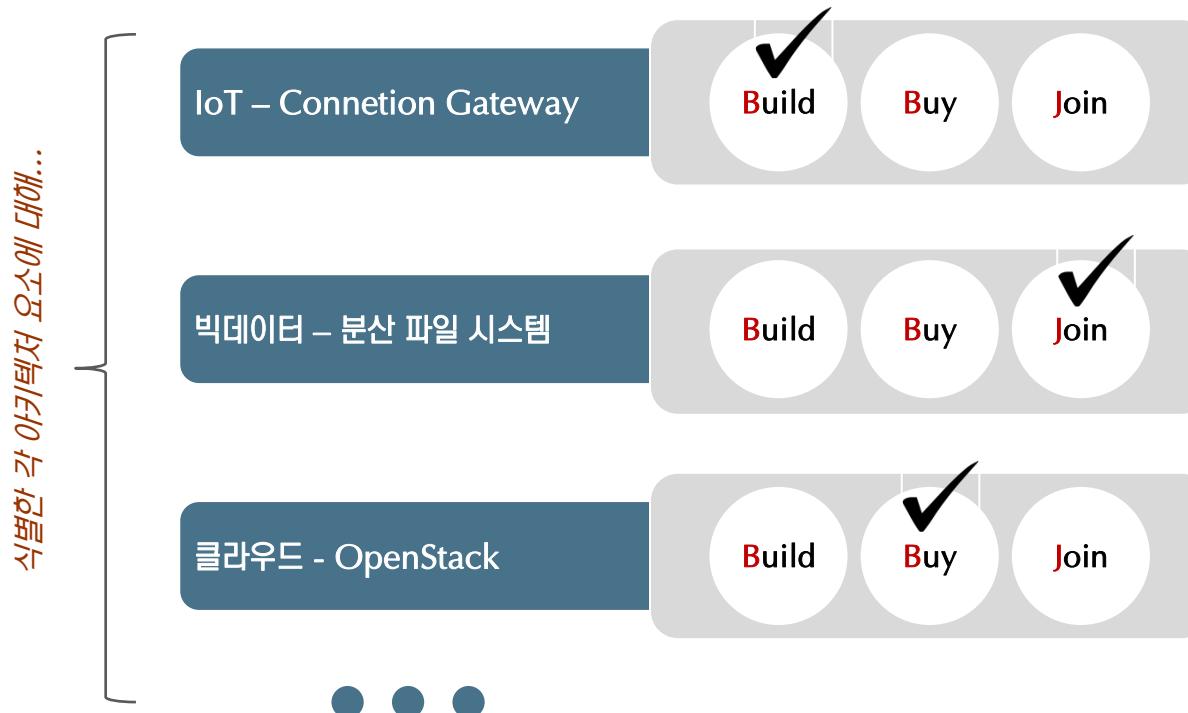
9.4 아키텍처 요소[프레임워크] 획득(1/2) – 전략

- ✓ 벤더의 특정 솔루션, 특히 표준 기반이 아닌 솔루션의 경우 시스템 개선에 걸림돌이 될 수 있습니다.
- ✓ AS-IS 시스템의 경우, 서버측의 모든 코드가 C사의 프레임워크에 강한 의존관계를 가지고 있습니다.
- ✓ 구조적인 결함이 있어도, 개선이 불가능한 상태입니다. 차세대는 이런 의존성을 제거해야 합니다.
- ✓ 프레임워크와 플랫폼은 중요한 아키텍처 요소입니다. 아래 원칙을 지키며 선정해야 합니다.



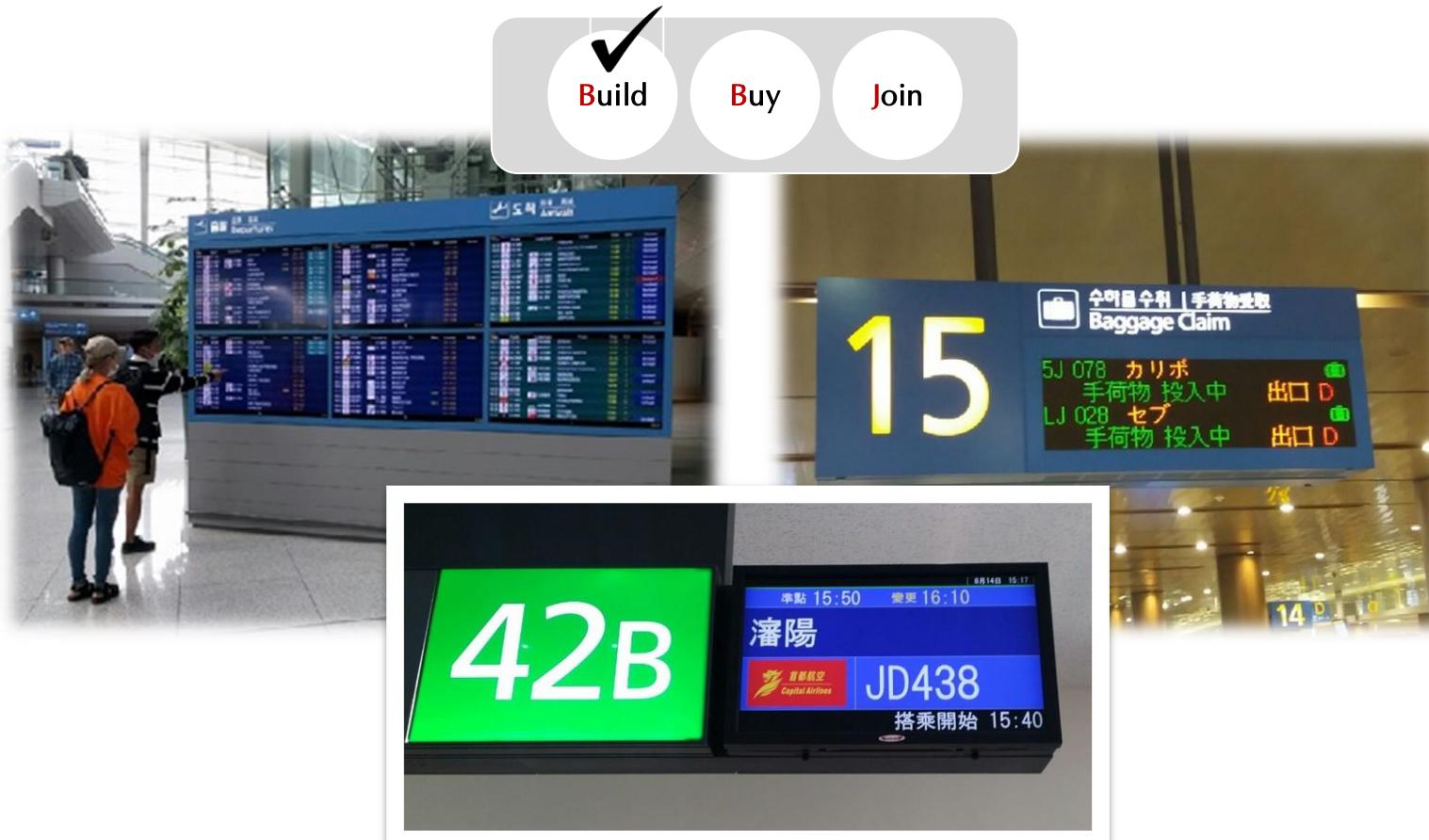
9.4 아키텍처 요소[프레임워크] 획득(2/2) – Build, buy, join

- ✓ 최근의 트랜드 기술은 하나의 요소가 아니라 기술 요소의 그룹으로 구성되어 있습니다. 규모가 크고 복잡합니다.
- ✓ 이렇게 다양하고 복잡하고 기술을 받아들일 때, 조직은 세 가지 접근 방법을 생각할 수 있습니다.
- ✓ Time-to-market이 중요한 기술은 대체로 “Buy”, 길게 보았을 때 핵심 기술은 “Build” 접근방법을 선택합니다.
- ✓ 핵심 기술이지만 “Build”가 불가능 기술 즉, 규모가 크고 복잡한 기술은 오픈 소스 “Join”접근방법을 선택합니다.



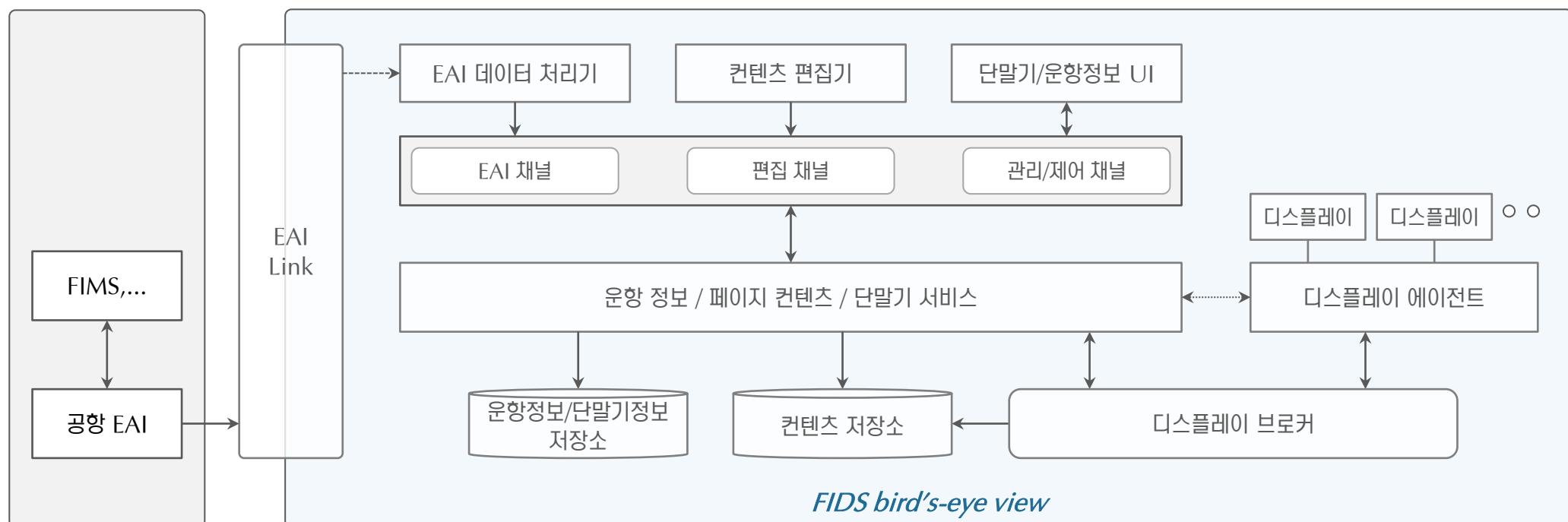
9.5 빌드 사례 1 – Display Broker

- ✓ 공항에서 승객들이 가장 많이 찾는 정보가 항공기 출발과 도착 정보입니다.
- ✓ 다양한 형식의 디스플레이로 출도착 정보를 동시에 보여줘야 합니다.
- ✓ 많은 곳은 5000 개가 넘는 디스플레이가 있으며, 디스플레이 종류 또한 구형에서 신형에 이르기 까지 매우 다양합니다.
- ✓ FIDS(Flight Information Display System) 시스템에서 시간 지체없이 “동시 표출”은 중요한 과제입니다.



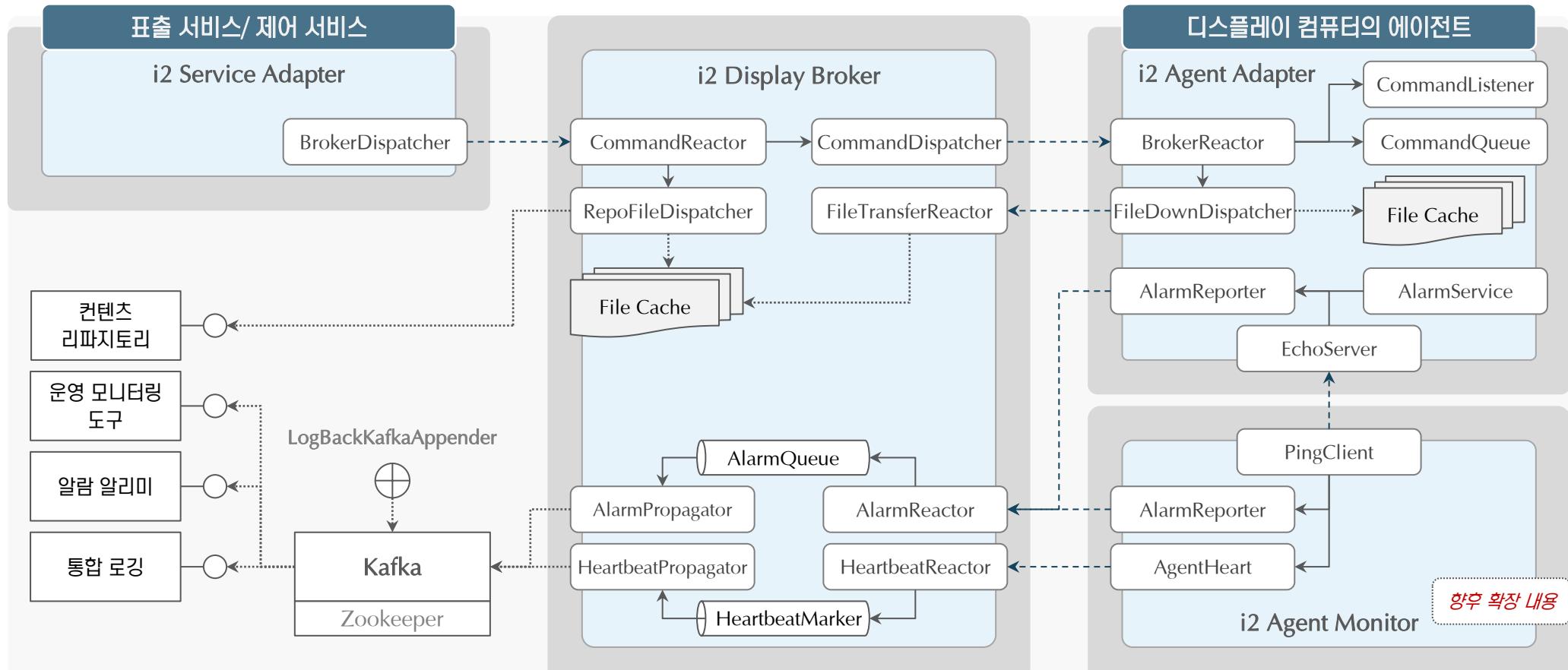
9.5 빌드 사례 1 – Display Broker

- ✓ FIDS를 구성하는 서브 시스템은 서비스, 메시지 큐, 제어 브로커, DCU Agent 등 모두 여덟 가지입니다.
- ✓ 시스템의 가장 중요한 특성인 디스플레이 제어와 표출을 위해 실패로부터 안전한 두 가지 장치를 추가했습니다.
- ✓ 디스플레이 메시지 브로커는 표출할 컨텐츠를 DCU로 전달하는 역할을 합니다. 속도보다는 보증이 중요합니다.
- ✓ 디스플레이 제어 브로커는 단말기를 모니터링, 제어, 진단하며, 동기 방식으로 작동합니다. 속도가 무엇보다 중요합니다.



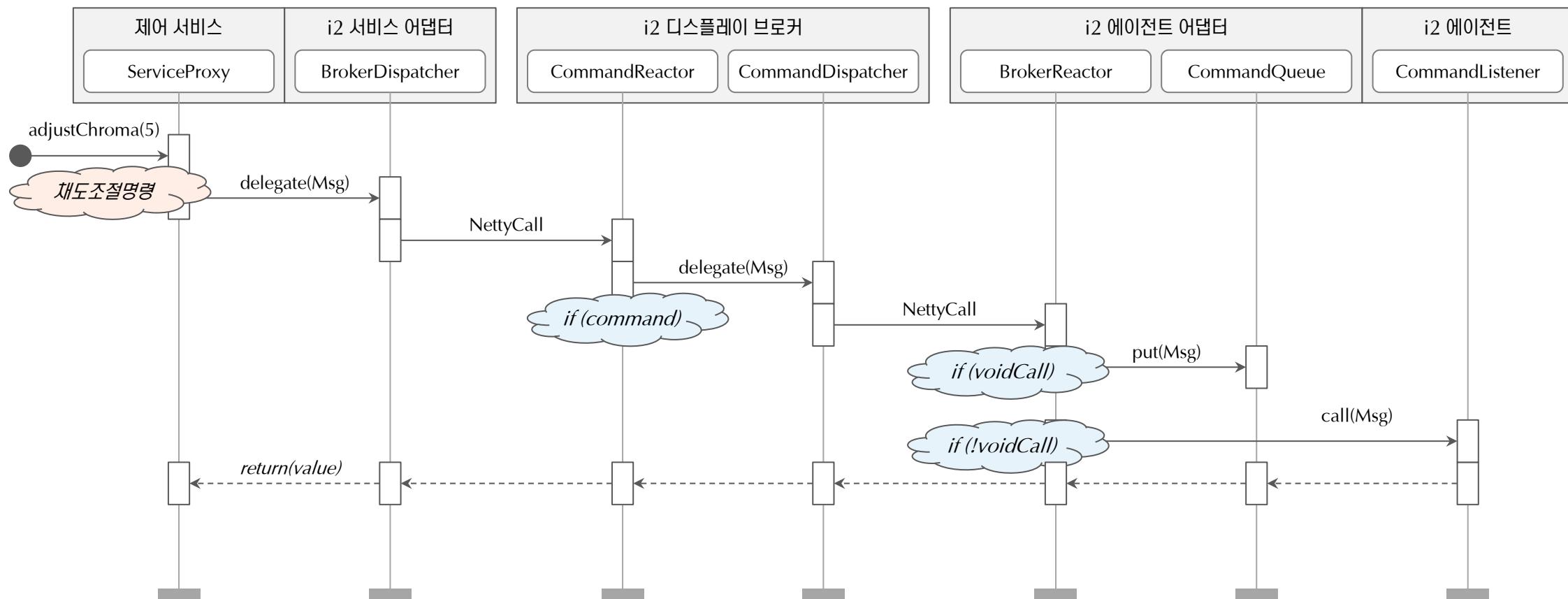
9.5 빌드 사례 1 – Display Broker

- ✓ i2 Display Broker는 명령(제어/진단), 파일 전송, 알람 전송, Heartbeat 전송 네 가지 태스크를 수행합니다.
- ✓ 빠른 파일 송수신을 위하여 브로커 내부에 파일 캐시를 두어서 관리합니다.
- ✓ 에이전트로부터 올라 온 상태 정보는 kafka를 경유하여 그 정보를 필요로 하는 곳으로 전달합니다.
- ✓ 에이전트 모니터는 Agent를 감시하며 브로커로 상태를 보고하고, 오류 발생으로 에이전트가 다운되면 다시 실행합니다.



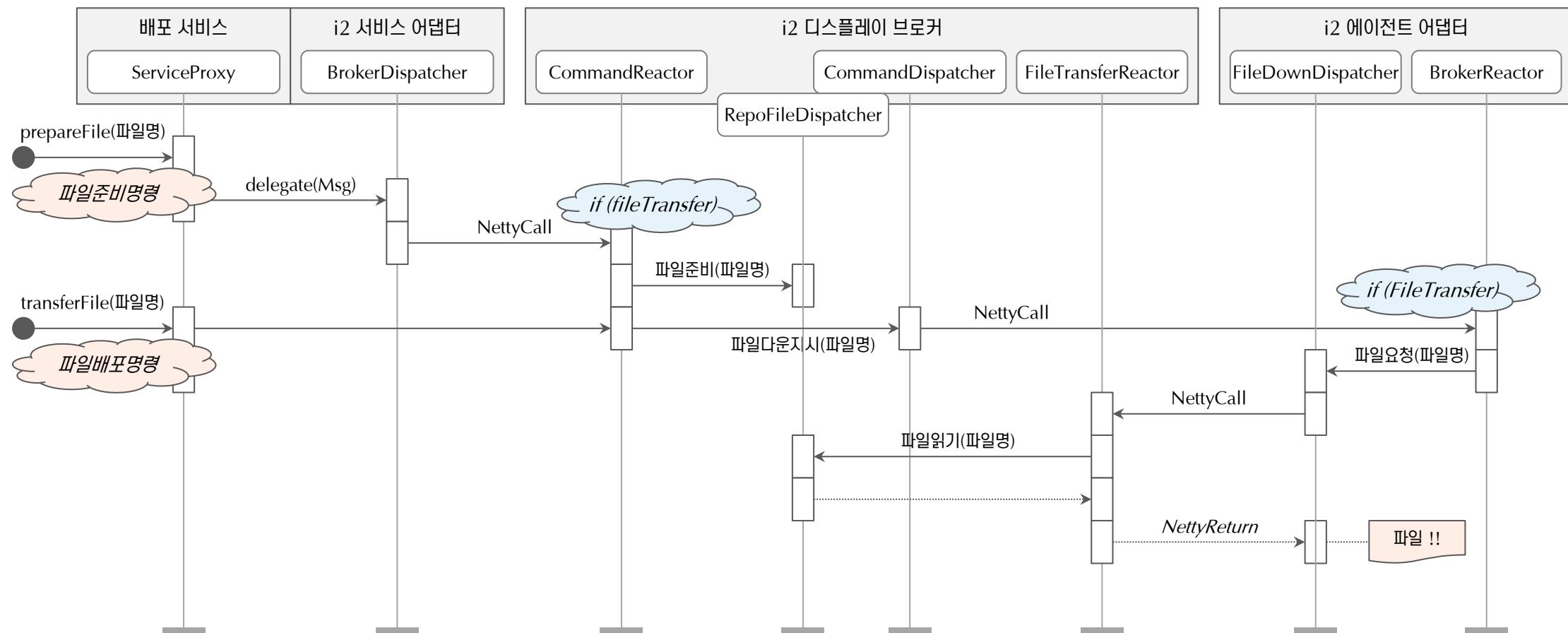
9.5 빌드 사례 1 – Display Broker

- ✓ 명령(예, 채도 조절, 밝기 조절, 전원 켜기/끄기)을 내리면, 서비스 어댑터는 메시지를 만들어 브로커로 전달합니다.
- ✓ 전달하는 메시지는 JSON 포맷이며, 제어 정보 객체는 JSON 속에 JSON 으로 변환하여 전달합니다.
- ✓ 브로커는 명령 여부를 판단하여 단순한 명령이면, CommandDispatcher를 이용하여 에이전트 어댑터로 전달합니다.
- ✓ 에이전트 어댑터는 Void 호출이면 큐에 메시지를 넣고 바로 리턴, 아니면 Listener를 실행하고 그 결과를 리턴합니다.



9.5 빌드 사례 1 – Display Broker

- ✓ 배포 서비스에서 파일을 전송하라는 명령을 내리면, 어댑터는 그 명령을 브로커에게 전달합니다.
- ✓ 브로커는 명령 유형이 “파일 전송”이면, RepoFileDispatcher에게 해당 파일을 파일 저장소에서 가져오라고 합니다.
- ✓ 파일이 준비되면 기본 명령 경로를 통해 에이전트 어댑터에게 지정된 시간에 파일을 가져가라고 명령합니다.
- ✓ 에이전트는 명령이 “파일 전송”이면 브로커에게 해당 파일을 요청하여 받아 갑니다.



9.6 실습 9-1 : POI Wrapper

- ✓ 매우 복잡한 엑셀을 다루어야 하는 상황입니다. 개발자들은 아파치 POI 라이브러리를 사용하고 있습니다.
- ✓ 엑셀의 특정 시트로부터 특정 라인을 키워드(예, Java)로 찾아서 읽어 오는 작업을 해야 합니다.
- ✓ 모든 개발자들이 POI 라이브러리의 복잡한 API를 익힐 시간이 없습니다.
- ✓ 아키텍트 팀원인 여러분은 POI를 쉽게 사용할 수 있도록 Wrapper를 만들어야 합니다.

C 서브카테고리	D 단위기술	E 기준	F 중요도	G SW 개발자(L12)			J 서버 개발자(L39)			M 프론트 엔지니어(L39)		
				L1	Mandatory	Optional	L1/L2/L3	Mandatory	Optional	L1/L2/L3	Mandatory	Optional
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)		100
	Delphi	G24	Low							G+MO(#01)		100
	Power Builder	G24	Low							G+MO(#01)		100
	Visual Basic	G24	Low							G+MO(#01)		100
웹언어	Swift	G39	High									
	Objective C	G39	High									
	HTML5	G24	Middle	G1+M	50					G1+M		50
	HTML/CSS	G24	Middle	G1+M	50					G1+M		50
	Javascript	G39	High	G1+M	33					G+MO(#01)		100
	JSP/Servlet	G24	Middle	G1+O		50				G+MO(#02)		100
	Php	G24	Middle							G+MO(#02)		100
	Asp	G24	Middle							G+MO(#02)		100
	HTML SAP UI5	G24	Middle							G+MO(#02)		100
	Webdyn ABAP	G24	Middle							G+MO(#02)		100
	Linux Shell	G24	Low				G1+M	50		G1+M		50
	Ruby	G24	Low									

9.6 실습 9-1 : POI Wrapper

- ✓ 인터페이스 수준에서 도메인 모델을 먼저 작성합니다.
- ✓ 어떤 POI를 사용하기 위해서 어떤 개념들이 필요할까요? → ExpowFile, ArraySheet, ExpowCell, ExpowRow, ...
- ✓ 엑셀을 단순하게 보는 방안에 대해 고민을 합니다.
- ✓ 실습의 목표는 엑셀 래퍼 라이브러리입니다.

```
public interface ExpowFile {  
    //  
    public String toJson();  
    public String toPrettyJson();  
  
public interface ExpowRow {  
    //  
    public String toJson();  
    public String toPrettyJson();  
    public boolean hasCellValue(String valueStr);  
    public boolean hasCellValueAt(int columnIndex, String valueStr);  
    public ExpowCell requestCell(int columnIndex);  
    public ExpowCell requestCell(String valueStr);  
    public ExpowCell requestRightCellOf(String valueStr);  
    public ListIterator<ExpowCell> requestCellsIteratorFrom(String valueStr);  
    public ListIterator<ExpowCell> requestCellsIterator();  
    public int countCells();  
    public int getRowIndex();  
}  
  
public interface ArraySheet {  
    //  
    public String toJson();  
    public String toPrettyJson();  
    public String toPrettyTable();  
    public String toPrettyRow(int rowIndex);  
    public String toPrettyColumn(int columnIndex);  
    public int getSheetIndex();  
    public String getSheetName();  
    public int countRow();  
    public String requestCellValue(int rowIndex, int columnIndex);  
    public String[] requestColumnArray(int columnIndex);  
    public String[] requestRowArray(int rowIndex);  
    public List<XYCoord> requestValueCoordinates(String cellValue);  
}
```

요약

- ✓ 아키텍처 설계에 필요한 재료로써 플랫폼, 프레임워크, 라이브러리가 필요합니다.
- ✓ 수많은 오픈 소스들이 있어서 편하게 선택할 수 있지만, 그럼에도 현장의 니즈를 모두 충족할 수 없습니다.
- ✓ 따라서, 설계 중에 필요한 아키텍처 요소를 획득하는 전략(Build, Buy, Join) 이 필요합니다.
- ✓ Buy 전략은 장점이 많지만 의존성 문제가 발생하므로 다른 안이 없을 경우의 마지막 선택이어야 합니다.



✓ 질문과 대답

✓ 토론

감사합니다...

- ❖ 송태국 대표 (tsong@nextree.co.kr)
- ❖ 넥스트리컨설팅(주)
- ❖ www.nextree.co.kr