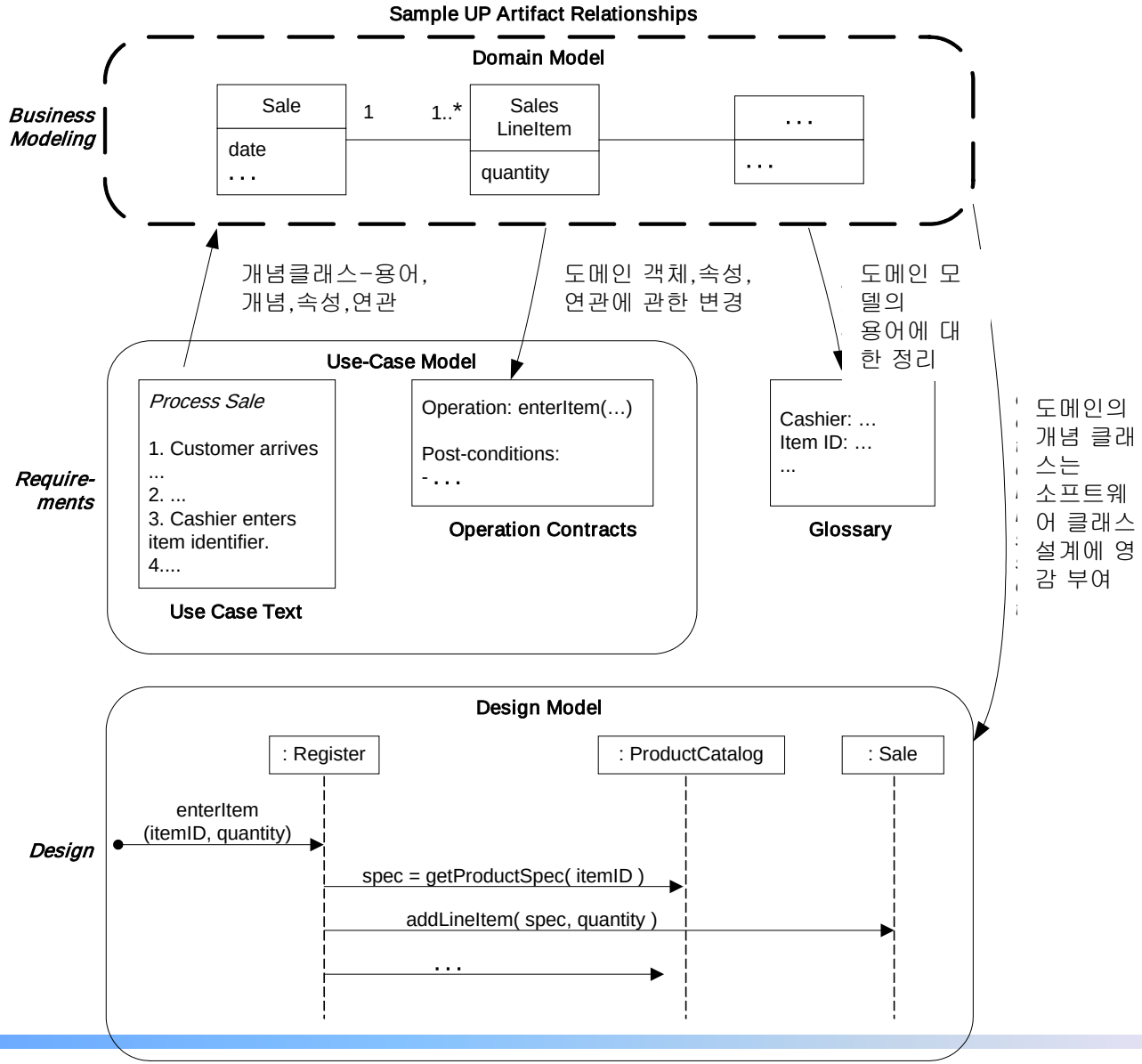
Domain Model : 문제영역을 시각화하기

목표

- 개념 클래스 식별
- 초기 도메인 모델 작성
- 속성 식별
- 명세 개념 클래스 추가
- 개념 관점과 구현 관점 비교/대조

도메인 모델의 작성 및 쓰임새



소개

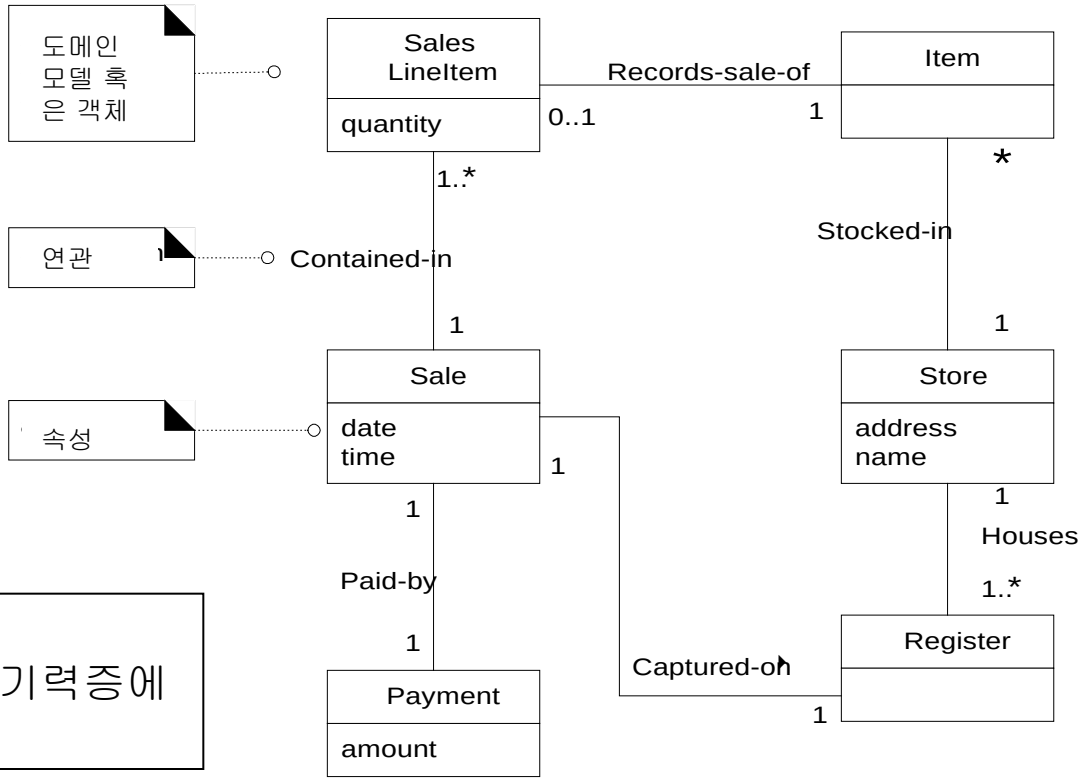
- 도메인 모델
  - 소프트웨어 객체 설계에 영감을 불어넣는 원천, 향후 나올 산출물들의 필수 입력
  - 문제 도메인의 의미 있는 개념 클래스

핵심 아이디어
도메인 모델은 소프트웨어 컴포넌트가 아니라 실세계 개념 클래스들의 표현이다. 소프트웨어 클래스 또는 책임을 갖고 있는 소프트웨어 객체를 나타내는 다이어그램 집합이 아니다.

1. 도메인 모델(1/4)

- 도메인 모델
  - 관련 도메인의 개념 클래스 또는 실세계 객체들의 시각적 표현
  - UML 노테이션

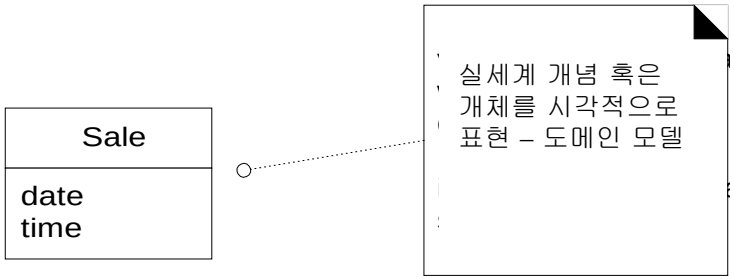
    클래스 다이어그램 :
    - 도메인 객체 또는 개념 클래스,
    - 개념 클래스들 간의 연관
    - 개념 클래스의 속성
- 핵심 아이디어
  - 추상화의 시각적인 사전

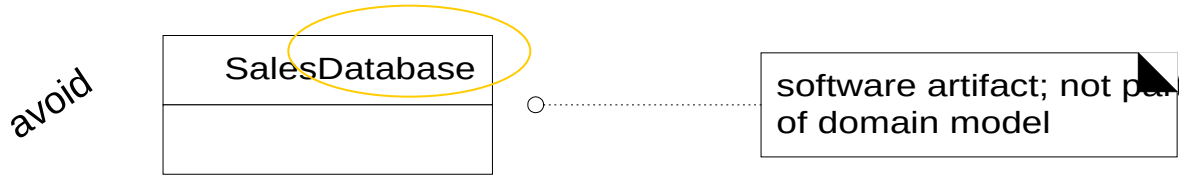


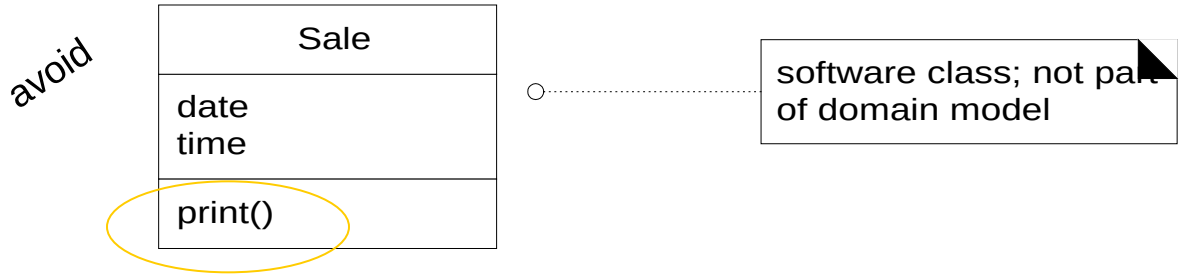
가이드라인
적절한 수준의 분석만 하라. 분석 무기력증에 빠지지 않도록.

2. 도메인 모델(2/4)

- 도메인 모델은 소프트웨어 컴포넌트 모델이 아니다.
- 도메인 모델은 개념을 시각화 – 시각적 사전

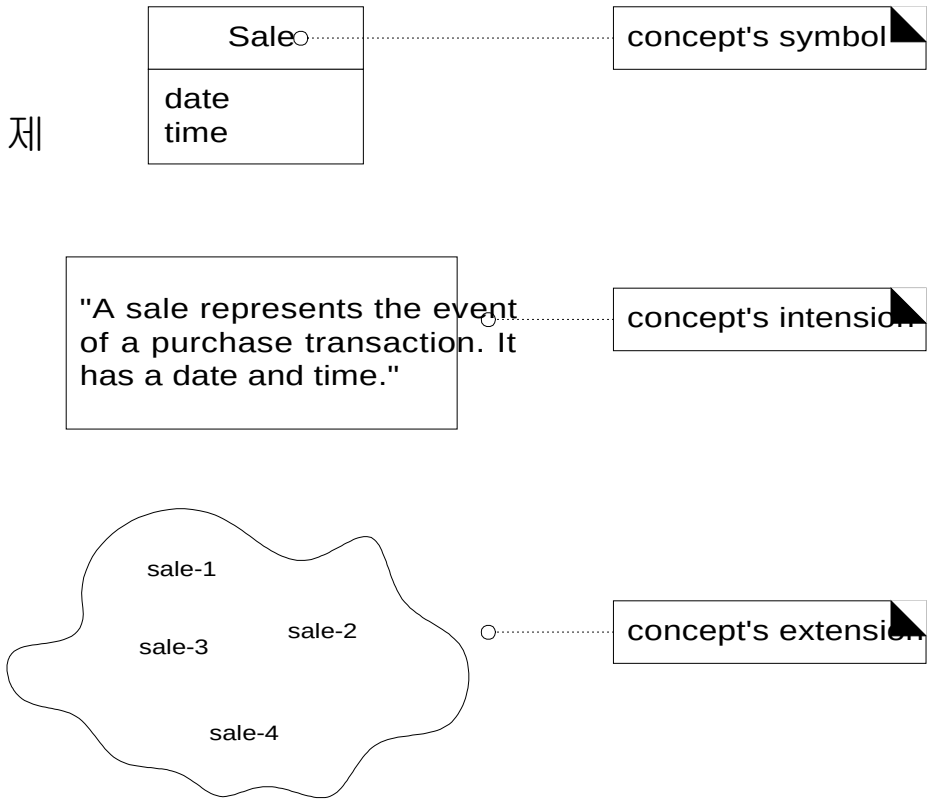






도메인 모델(3/4)

- 개념 클래스 : 하나의 생각, 사물 혹은 객체로 세 가지 측면에서 고려
  - 상징 : 개념 클래스의 단어 또는 이미지
  - 의도 : 개념 클래스 정의
  - 확장 : 개념 클래스가 적용되는 예제 집합



3. 도메인 모델(4/4)

- **도메인 모델과 분해**
  - 도메인의 명확한 이해
  - 객체의 분할
  - 사람의 생각과 소프트웨어의 표현과의 차이 수렴

- **판매 도메인에서의 개념 클래스**

Store	Register	Sale

객체지향과 구조적 분석의 차이는 함수의 분할이 아니라 개념 클래스의 분할에 있다.

4. 개념 클래스 식별(1/3)

많은 상세 개념 클래스들이 있는 도메인 모델이 적은 것보다는 낫다

- 개념 클래스 식별 기법
  1. 기존의 모델을 재사용하거나 수정한다.
  2. 분류 리스트 사용
  3. 명사 어구 식별

- 도메인 모델 생성 절차
  1. 개념적 클래스를 식별한다. (식별기법 이용)
  2. 이들을 UML 클래스 다이어그램으로 그린다.
  3. 연관관계를 추가한다.
  4. 속성을 추가한다.

5. 개념 클래스 식별(2/3)

- 개념 클래스 분류 리스트 (표 9.1)

개념 클래스 카테고리	예
물리적, 유형 객체	Register, Airplane
사물의 명세, 설계, 기술	ProductSpec, FlightDescription
장소	Store, Airport
트랜잭션	Sale, Payment
트랜잭션 라인 아이템	SalesLineItem
사람의 역할	Cashier, Pilot
다른 사물의 컨테이너	Store, Bin, Airplane
시스템 외부의 컴퓨터	AirTrafficControl
추상 명사 개념	Hunger, Acrophobia
조직	SalesDepartment
이벤트	Sale, Meeting,

개념 클래스 카테고리	예
프로세스	SellingAProduct
룰과 정책	RefundPolicy
카탈로그	ProductCatalog
재정, 작업, 계약, 법적 사건 기록	Receipt, Ledger
재정 도구 및 서비스	LineOfCredit, Stock
매뉴얼, 문서, 참조 문서, 책	DailyPriceChage List, RepairManual

개념 클래스 식별(3/3)

- 명사 어구로 개념 클래스 식별

조심스럽게 적용; 명사-클래스 매핑 x, 자연어의 단어는 애매모호

기본 성공 시나리오(기본 흐름)

1. **고객**은 사려는 **상품**이나 **서비스**를 갖고 POS 계산대에 도착한다.
2. 출납원은 새로운 판매를 시작한다.
3. 출납원은 품목 식별자를 입력한다.
4. 시스템은 품목을 기록하고 품목 설명, 가격, 합계를 보여준다. 가격은 가격 룰에 의해 계산된다.
5. 시스템은 세금을 포함한 총액을 보여준다.
6. …

6. “판매”도메인의 개념 클래스 후보(1/2)

- 개념 클래스 카테고리 리스트와 명사 어구 분석 – “판매 처리” 요구사항

  - Register
  - Item
  - Store
  - Payment
  - ProductCatalog
  - ProductSpecification
  - SalesLineItem
  - Cashier
  - Customer
  - Manager

Register	Item	Store	Sale
Sales LineItem	Cashier	Customer	Ledger
Cash Payment	Product Catalog	Product Description	

“판매”도메인의 개념 클래스 후보(2/2)

- Report Objects – 모델에 영수증을 포함하나?
  - 영수증 : 판매 및 지불의 기록, 상대적으로 중요한 개념. 반드시 개념 모델에 나와야 하나?
    - 판매의 기록, 다른 정보로부터 정보를 추출, 중복 정보

    → 제외
    - 비즈니스 룰에서 특별한 역할 : 물품 반품 시 반드시 필요

    → 포함

→ “반품 처리” 시 사용

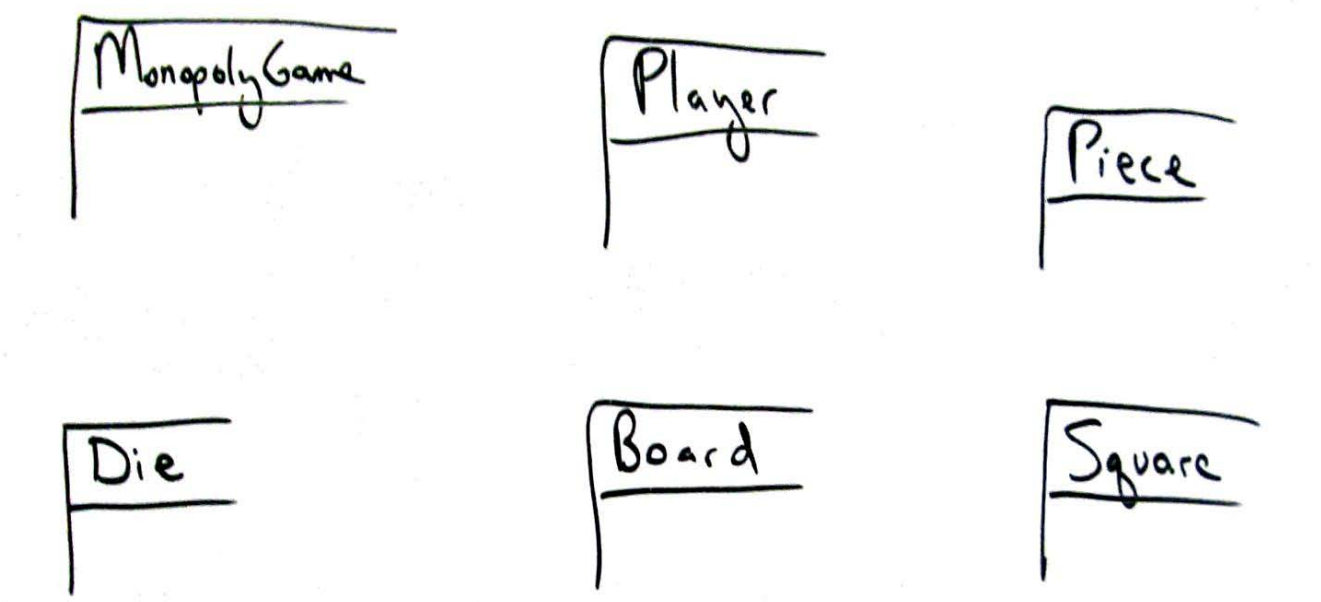
도메인 모델링 가이드라인(1/3)

- 도메인 모델을 어떻게 만들 것인가?

1. 개념 클래스 카테고리 리스트와 현재 요구사항에 관련한 명사 식별 테크닉을 이용하여 후보 개념 클래스를 나열한다.
2. 도메인 모델을 후보 개념 클래스를 그린다.
3. 메모리를 유지해야 할 필요가 있는 관계를 나타내기 위해 필요한 연관관계를 추가한다.
4. 자료 요구사항을 만족시키기 위해 필요한 속성을 추가한다.

7. 개념 클래스 다이어그램 스케치하기

- 우측 하단 부분을 열린 채로 그리기
  - 속성 등을 확장 가능하게
- 화이트보드/디지털카메라를 이용한 스냅샷
  - 설계의 힌트로만 사용하기 때문에 나중에 필요 없을 수 있음



10. 도메인 모델링 가이드라인(2/3)

- 명명 및 사물 모델링(Mapmaker)

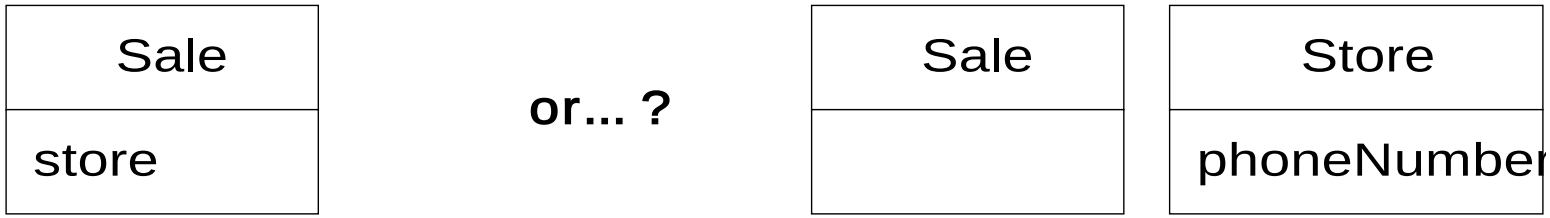
지도제작자가 어떻게 일하는가를 염두에 두고 도메인 모델을 만들어라
- 문제 영역에 있는 이름을 사용하라.
- 관련 없는 특징은 제외하라
- 거기 없는 건 추가하지 마라

  - 지도제작자는 그 영역에 있는 지명을 사용한다,
  - 지도제작자는 지도 목적에 관련한 것이 아니면 지도에서 지운다.
  - 지도제작자는 실제 있지 않은 것을 추가하지 않는다.

12. 도메인 모델링 가이드라인(3/3)

- **개념 클래스 식별에서의 공통된 실수**
  - 개념을 속성으로 나타내는 것!

실수를 피하기 위한 기본 원칙:
실세계에서 X를 숫자나 텍스트로 생각하지 않는다면 X는 속성이 아니라 개념 클래스이다.



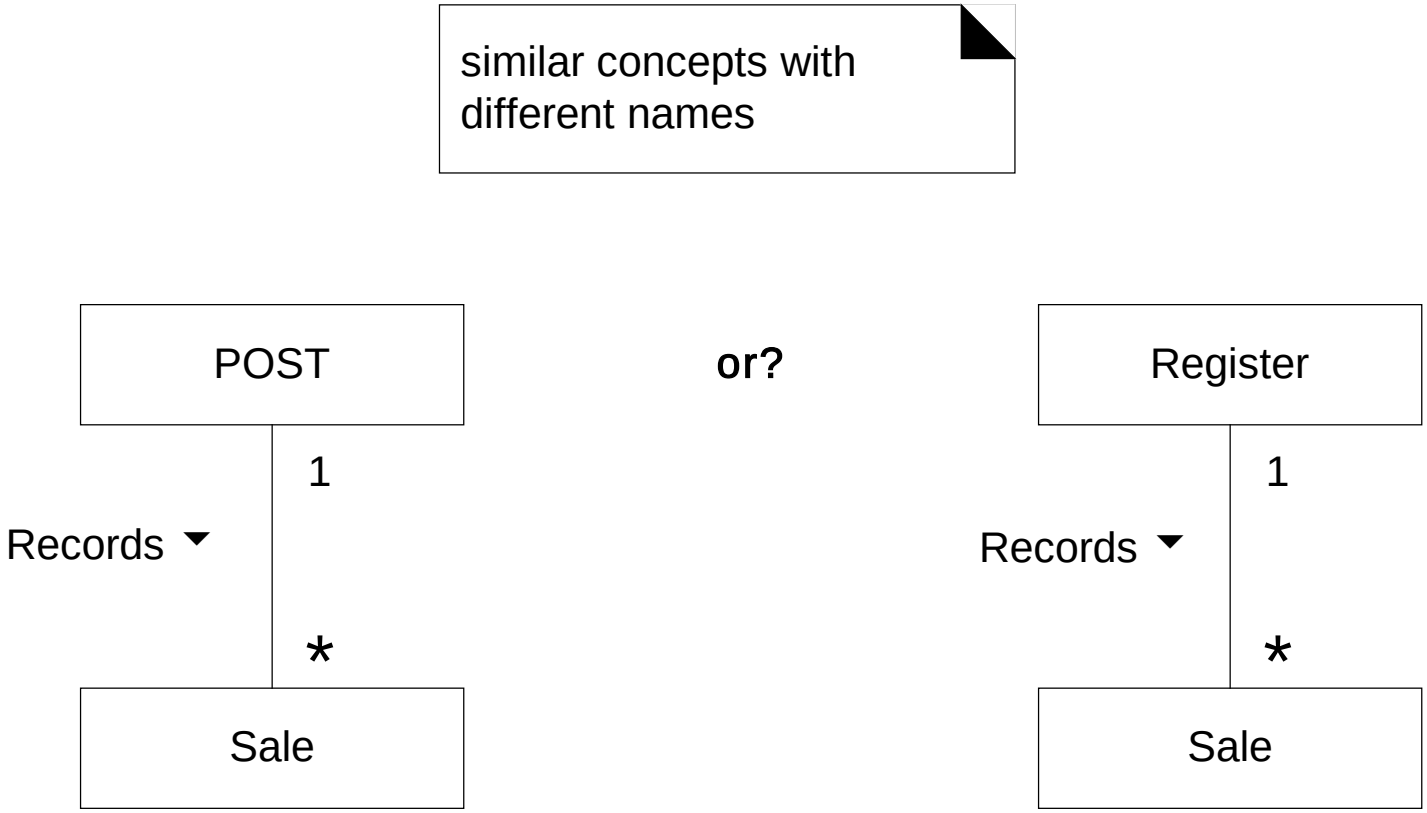
유사한 개념 클래스들 해결 – Register vs. “POST”(1/2)

- POST : Point-Of-Sale Terminal
  - 등록기 “Register”의 역할
  - 판매와 지불 내역을 기록
- 도메인 모델에서 Register vs. POST

> 기본 원칙 : 도메인 모델은 절대적으로 옳거나 그른 것은 없다.
> 단지 더 유용하냐 덜 유용하냐의 차이이다.;
> 즉 상호 의사소통의 도구이다.

  - POST : 익숙하고 의사소통 관점에서 유용
  - Register : 보다 추상적, 구현 독립적이라는 면에서 유용

유사한 개념 클래스들 해결 – Register vs. “POST”(2/2)



“Unreal” 세계 모델링

- 비즈니스 도메인과 소프트웨어 시스템의 유사성이 거의 없을 때
  - 통신 분야
    Message, Connection, Port, Dialog, Route, Protocol
  - 도메인 모델링 가능, 높은 추상화 요구

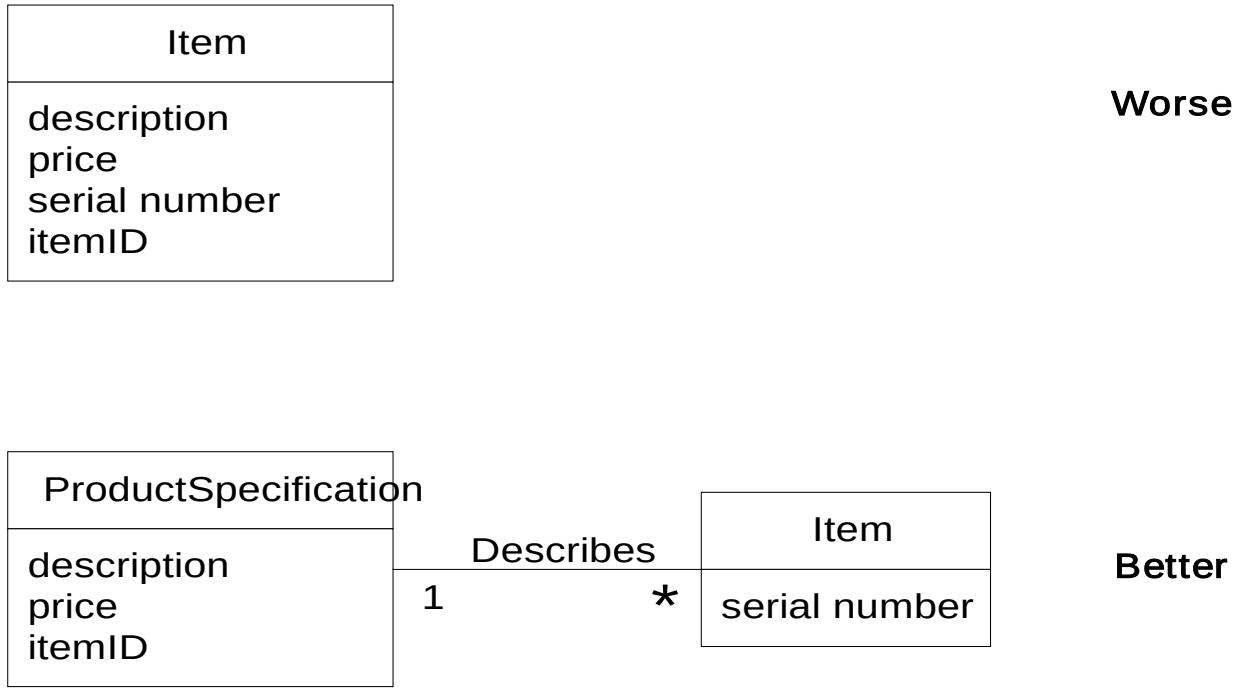
13. 명세 또는 설명(기술) 클래스 (1/4)

- 상황
  - *Item* 인스턴스는 가게에 있는 물리적인 상품을 나타낸다. 시리얼 번호를 갖고 있다.
  - *Item*은 설명, 가격, itemID를 갖고 있는데 다른 데는 저장되어 있지 않다.
  - 그 가게에서 일하는 모든 사람은 기억상실증을 갖고 있다.
  - 실제 물리적 상품이 팔릴 때마다, 해당하는 *Item*의 소프트웨어 인스턴스는 "소프트웨어 나라"에서 삭제된다.

- 어느날 특정 상품(오브젝 버거)이 매진된 날, 누군가 들어와 "오브젝 버거는 얼마예요?"라고 묻는다면?

명세 또는 기술 개념 클래스(2/4)

- 명세 또는 기술 개념 클래스 필요

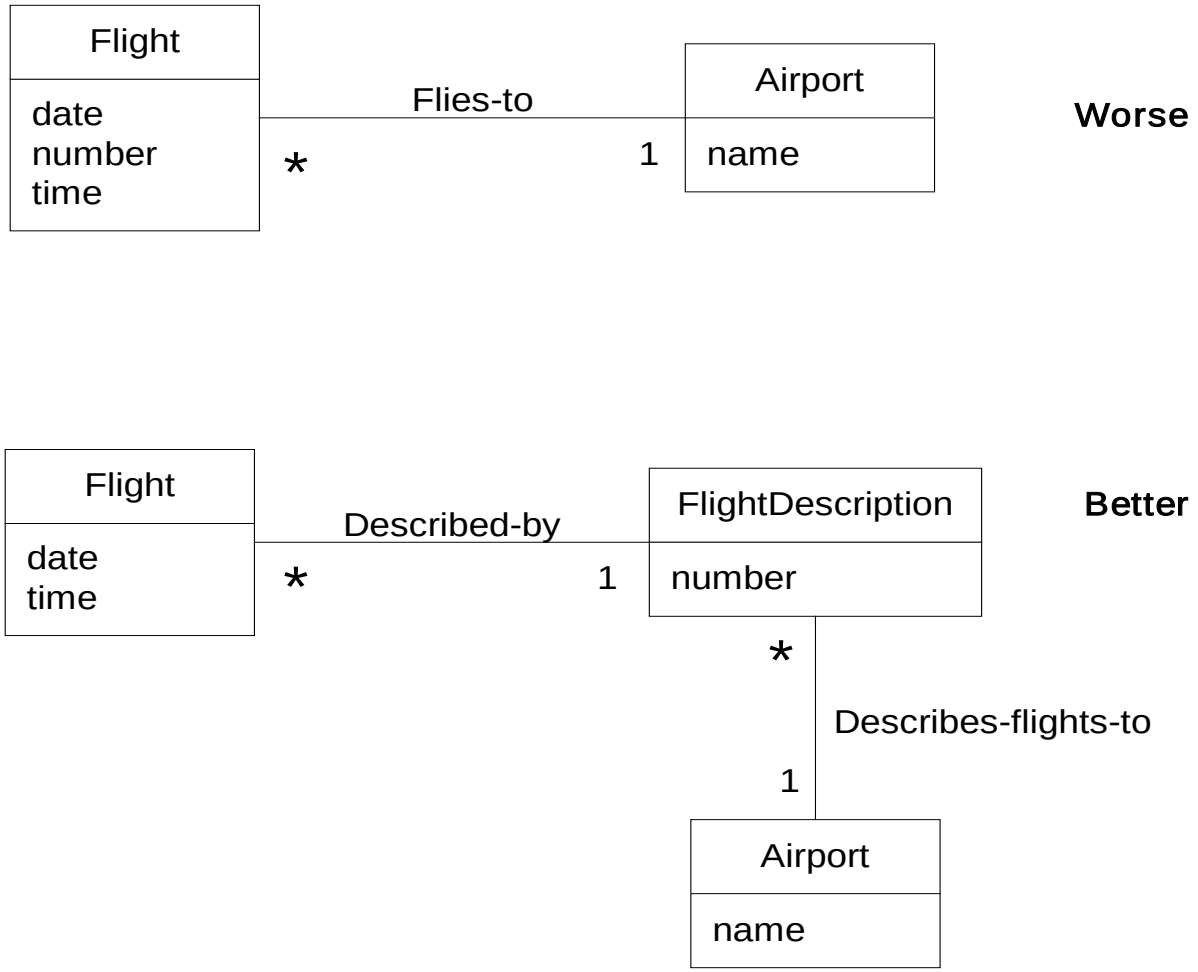


명세 또는 기술 개념 클래스(3/4)

- 언제 명세 개념 클래스가 필요한가?
  - 아이템 또는 서비스의 예가 현재 존재하는가에 무관하게 설명이 있어야 할 때
  - 인스턴스를 삭제하는 것이 유지해야 할 정보의 손실을 가져올 때(삭제된 인스턴스와 정보 간 잘못된 관련으로 인해)
  - 반복적 또는 중복 정보를 감소시킬 때

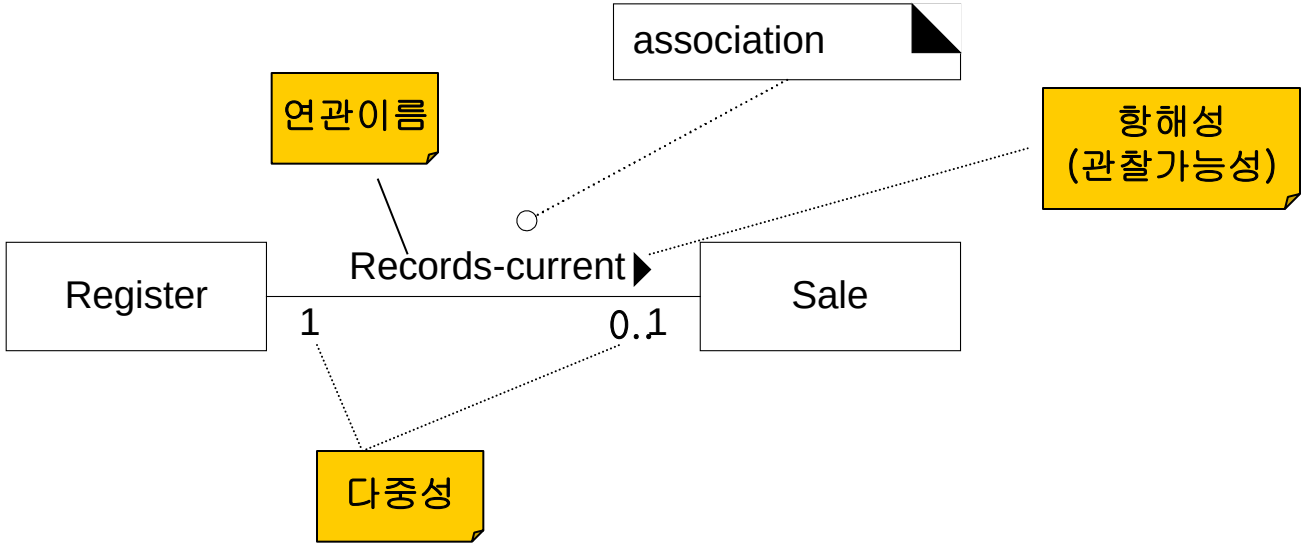
명세 또는 기술 개념 클래스(4/4)

- 서비스 예



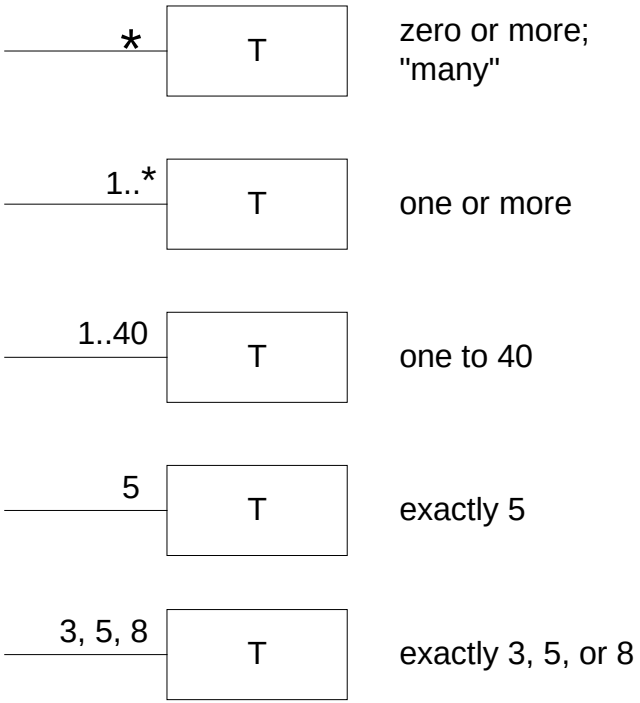
14 연관관계

- **클래스간 연관관계의 사용**
  - 일정 기간 동안 기억해야 할 필요가 있는 지식
  - 라인으로 연결하고 관계의 이름과 다중성을 기록
  - 예) *금전등록기는 현재 판매내역을 기록한다.*
    - 현재 판매내역의 개수는 하나 또는 없음. (다중성)
    - 등록기 객체는 판매객체의 정보의 위치를 알지만, 판매정보는 등록기 객체를 알지 못함



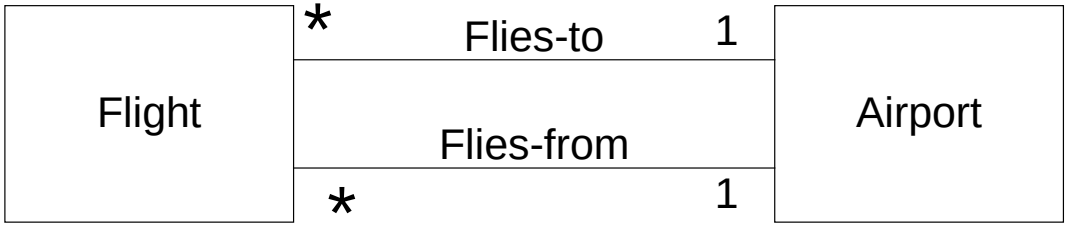
다중성

- 어떤 클래스의 한 객체에 대해 연관 있는 다른 클래스의 객체가 몇 개 대응되는가 혹은 기억되어야 하는가?



다중 연관관계

- 두 클래스에 대해 연관관계가 2개 이상일 때
  - 항공노선('Flight')은 공항과 2개의 연관을 가짐
  - 한 항공노선에 대해 출발공항, 목적공항으로서의 연관



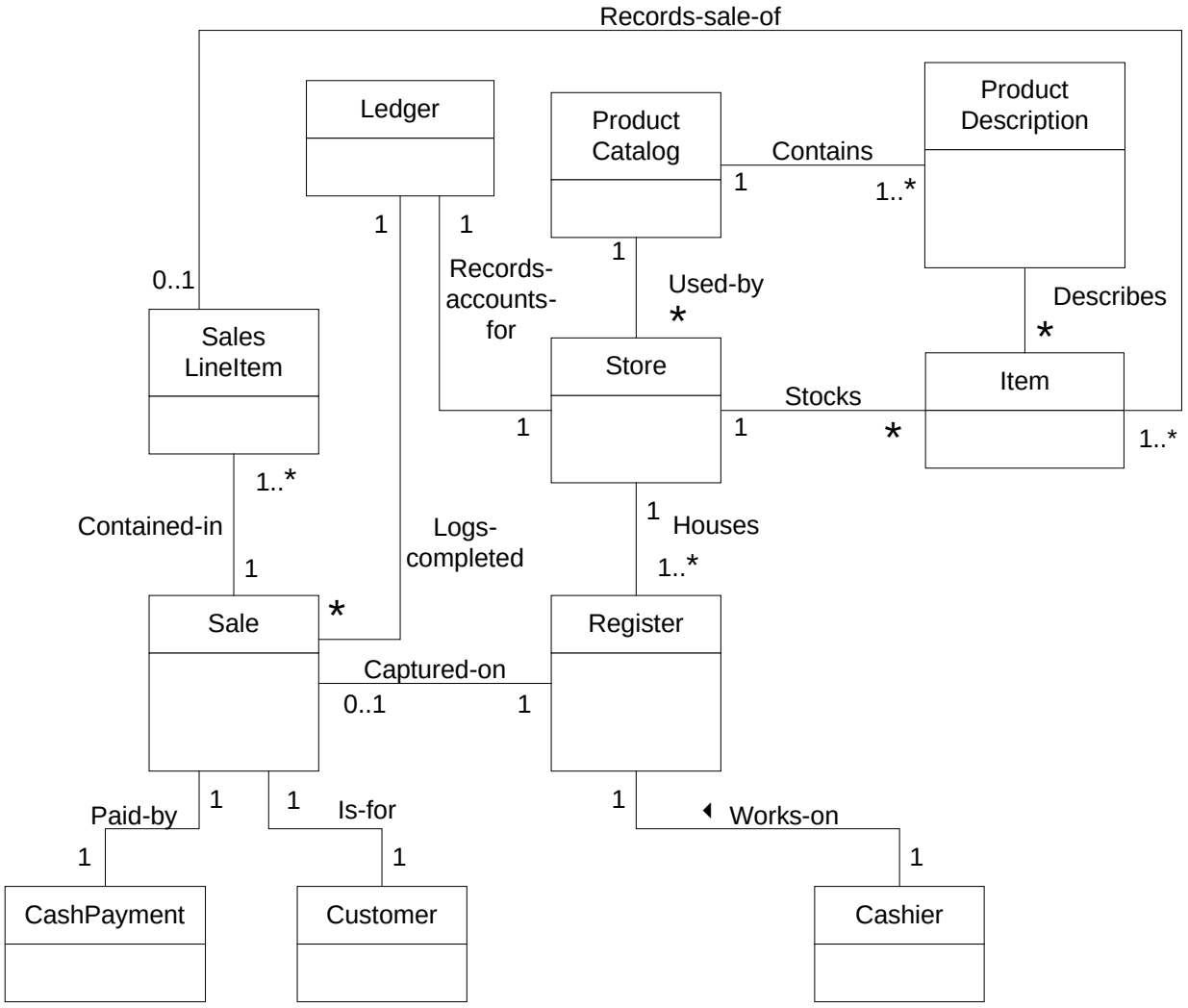
연관관계 찾기 규칙

범주	예제
A는 B의 물리적인 한 부분	*Drawer-Register*
A는 B의 논리적인 한 부분	*SalesLineItem-Sale*
A는 B에 물리적으로 포함	*Register-Store*
A는 B에 논리적으로 포함	*ItemDescription-Catalog*
A는 B를 설명	*ItemDescription-Item*
A는 트랜잭션 또는 보고서B의 line item	*SalesLineItem-Sale*
A는 B에 의해서 파악되고/기록되고/획득	*Sale-Register*
A는 B의 멤버	*Cashier-Store*

연관관계 찾기 규칙 (계속)

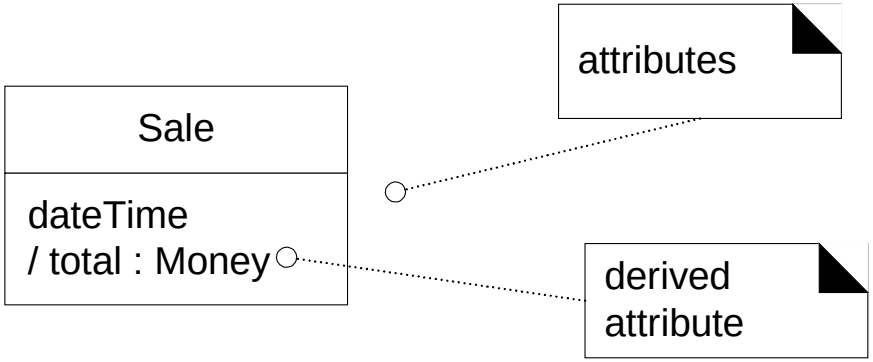
범주	예제
A는 B의 조직적인 하위 단위	*Department-Store*
A는 B를 사용 또는 관리	*Cashier-Register*
A는 B와 의사소통	*Customer-Cashier*
A는 트랜잭션B와 연관	*Customer-Payment*
A는 다른 트랜잭션B와 연관된 트랜잭션	*Payment-Sale*
A는 B의 옆(?)	*SalesLineItem-SalesLineItem*
A는 B에 의해서 소유	*Register-Store*
A는 B와 관련한 이벤트	*Sale-Customer*

연관의 예: POS


Records-sale-of
Ledger
Product Catalog
Contains
Product Description
1
1..*
1
1
1
Records-accounts-for
Used-by
*
Describes
*
0..1
Sales LineItem
Store
Stocks
Item
1
1
*
1..*
1..*
Contained-in
1
Logs-completed
1
Houses
1..*
Sale
*
Register
Captured-on
0..1
1
1
1
1
Paid-by
Is-for
Works-on
1
1
1
1
CashPayment
Customer
Cashier

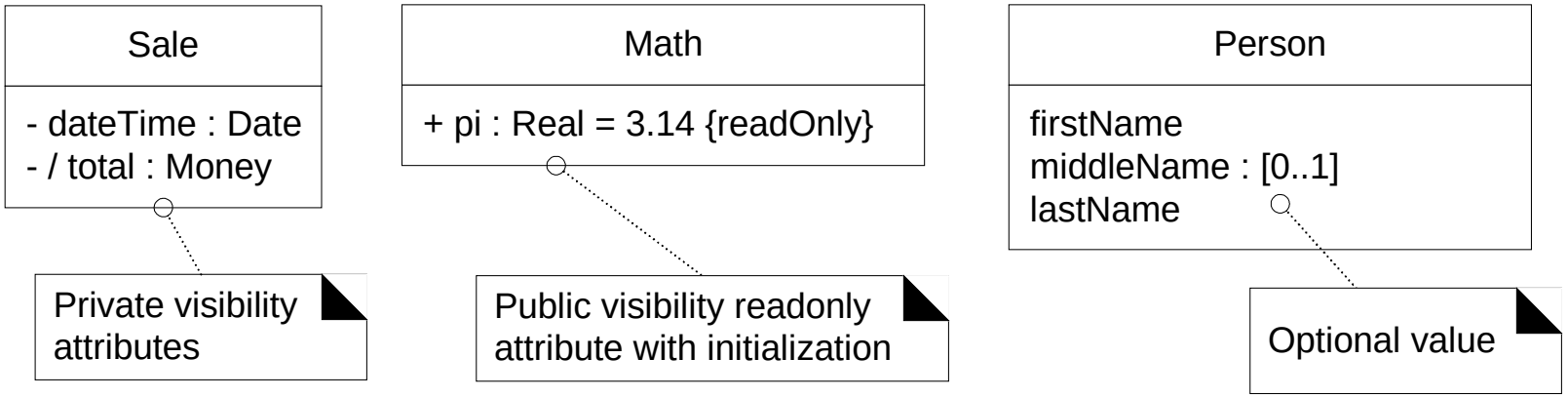
16. 속성 (Attributes)

- 클래스가 가지는 논리적인 데이터
  - Sale 클래스는 dateTime 속성이 필요하다.
    - 합계는 판매된 품목의 집합(SalesLineItem)으로부터 유도된 속성
  - Store 는 이름과 주소가 필요하다.
  - 판매원은 ID가 필요하다.



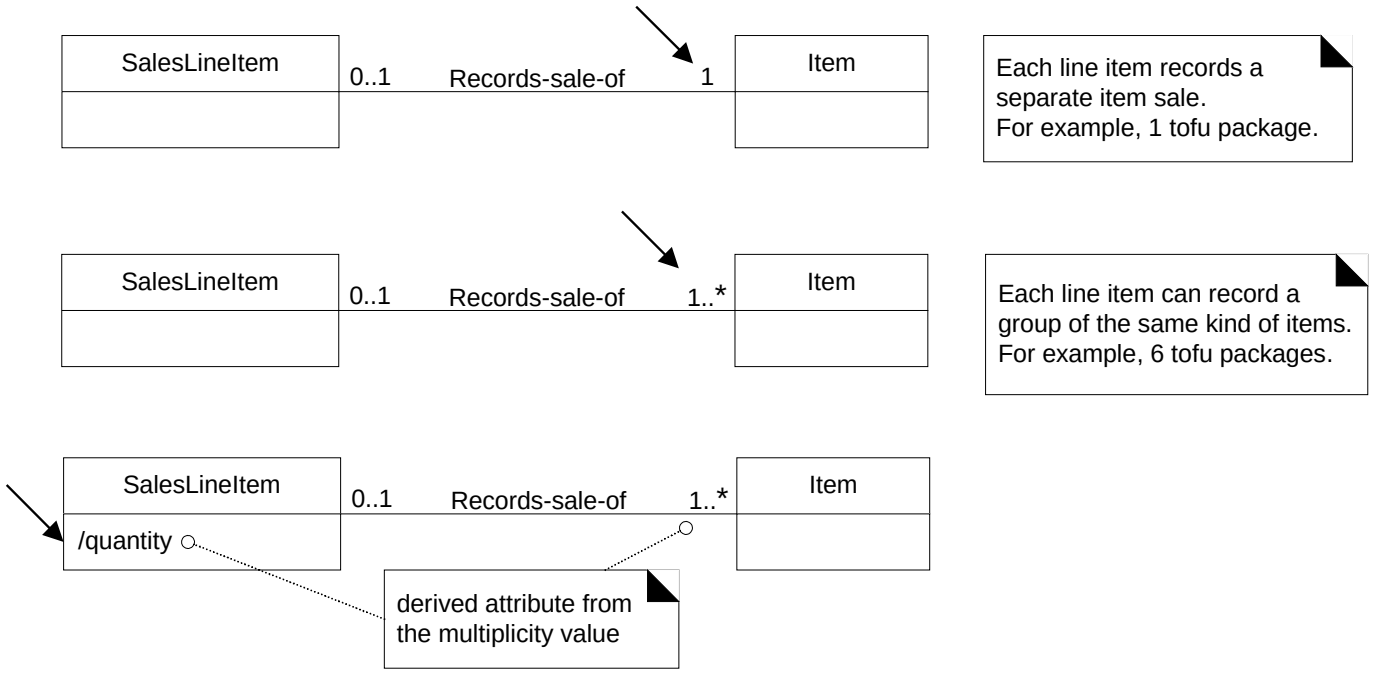
속성의 표기법

- 가시성 속성이름 : 데이터타입 다중성 = 초기값 {제한사항}
  - 가시성: + (public), - (private), # (protected)
  - 제한 사항: 속성이 가지는 제한 사항
- 처음에는 속성이름만으로 시작!
  - 구체화 될 수록 데이터타입, 가시성, 다중성, 초기값, 제한사항을 추가적으로 기록



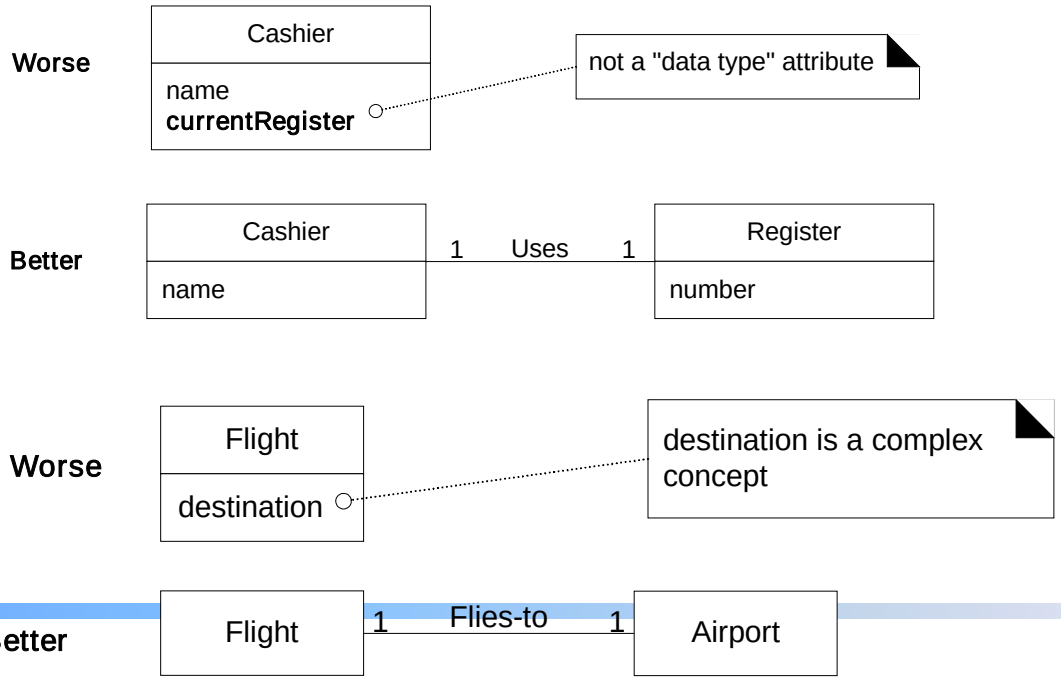
유도된 속성

- 다른 클래스의 속성으로부터 유도될 수 있는 속성
  - 예) Sale 의 합계는 SalesLineItem의 quanty,price에서 유도 가능



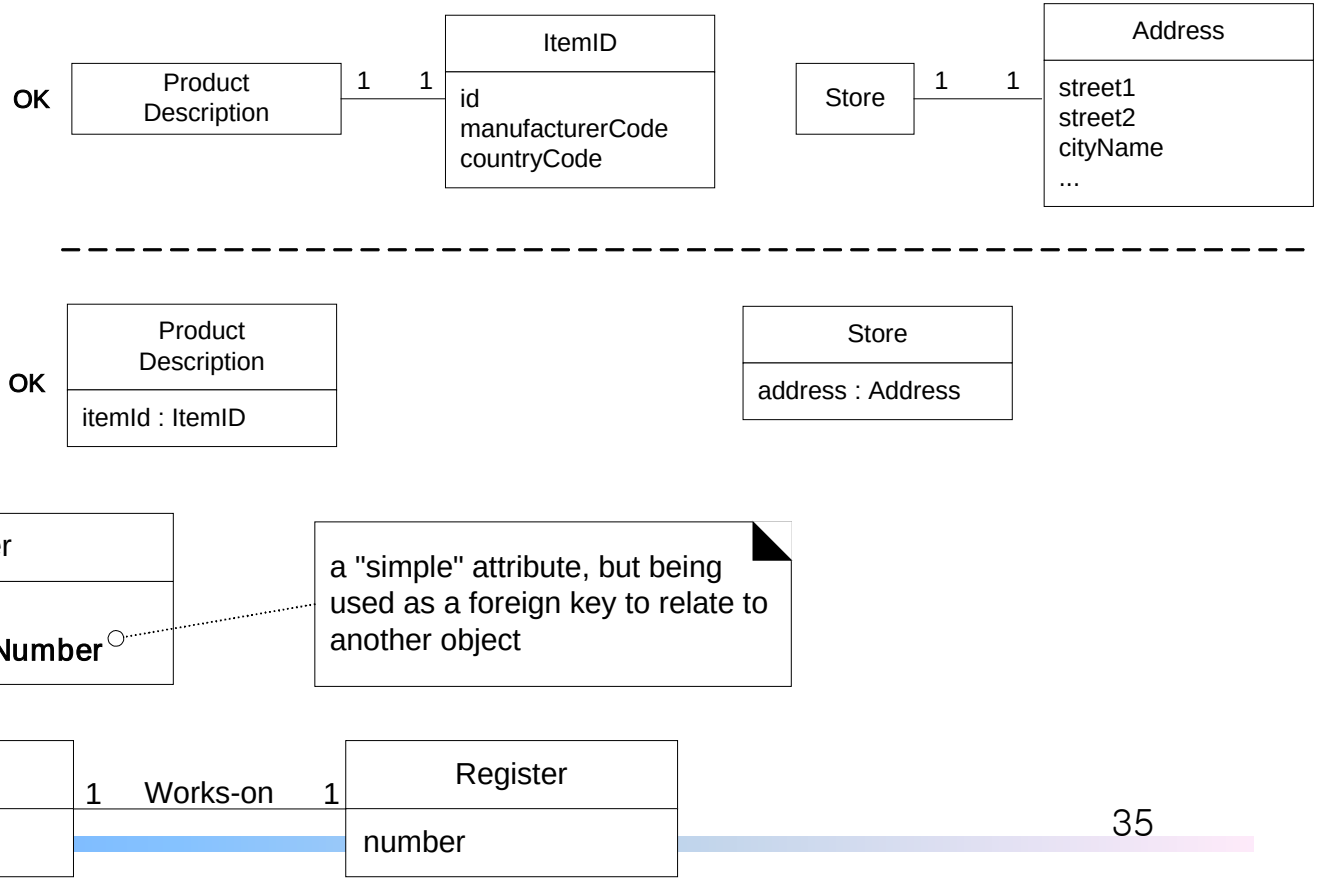
데이터 타입

- **속성 데이터 타입**
  - 데이터 타입: 빌트인, 사용자 정의
    - Boolean, Date, DateTime, Number, Character, String, Time 등의 빌트인
    - Address, Color, Geometrics, PhoneNumber, SSN, Universal Product Code, ZIP, Enumerated Type 등
    - 개념모델에서는 구현과 관계 없이 어떤 타입이라도 가능
- **복잡한 개념을 나타내는 것은 데이터 타입이 아니라 클래스로!**
  - 개념적 클래스는 다른 클래스의 속성이 아니라 연관관계로 표현!



데이터 타입

- 데이터 타입은 클래스 혹은 묵시적으로 표현
  - 추후 설계 때 구체적으로 반영
- 어떠한 속성도 외래 속성을 표현하지 않음.

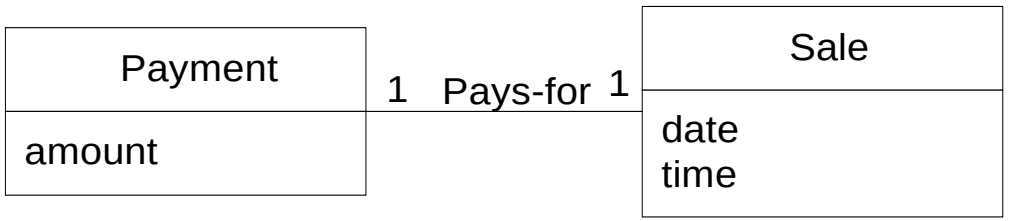


UML 기호, 모델링, 방법: 다양한 관점(1/3)

UML은 단순히 클래스 다이어그램, 시퀀스 다이어그램과 같은 다이어그램만 단순히 명시한다. 그 위에 어떤 방법이나 모델링 관점은 없다. 프로세스(UP)가 UML에 방법론적으로 정의된 모델 문맥을 적용한다.

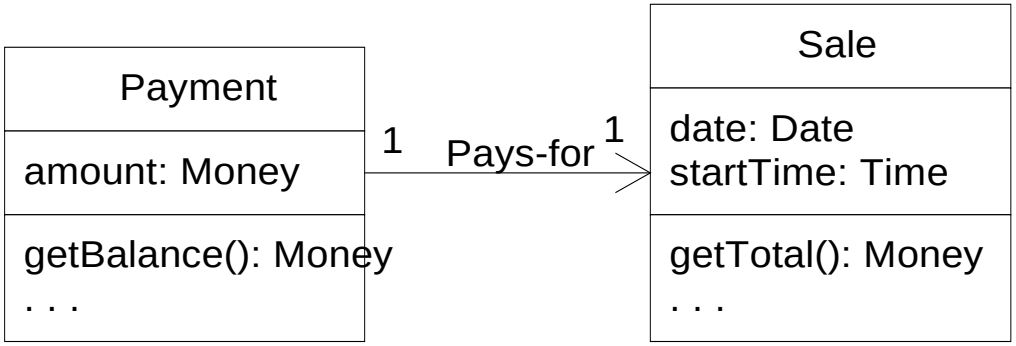
- **관점은 다르나 같은 표기법을 사용**
  1. **본질적 또는 개념적 관점**
     관련 도메인 또는 실세계의 사물
  2. **명세서 관점**
     소프트웨어 추상화 또는 명세서와 인터페이스를 가진 컴포넌트
  3. **구현 관점**
     특정 기술과 언어(예. Java)에 관한 소프트웨어 구현

UML 노테이션, 모델링, 메소드: 다양한 관점(2/3)



UP Domain Model

Raw UML class diagra notation used in an essential model visualizing real-world concepts.



UP Design Model

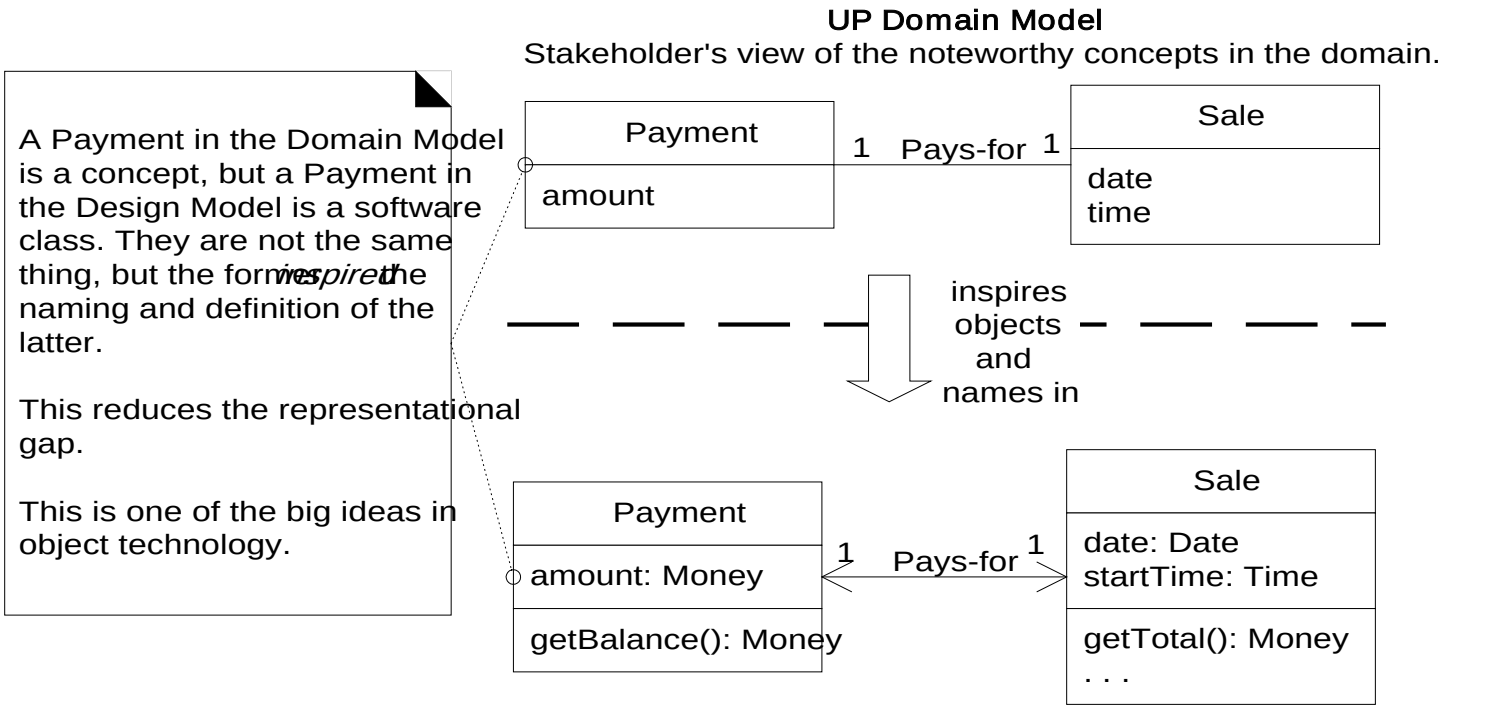
Raw UML class diagra notation used in a specification model visualizing software components.

UML 기호, 모델링, 메소드: 다양한 관점(3/3)

- 용어 : UML vs. 방법
  - 개념 클래스 : 실세계 개념/사물, 본질적 또는 개념적 관점. UP 도메인 모델은 개념 클래스를 포함
  - 소프트웨어 클래스 : 소프트웨어 컴포넌트의 명세 혹은 구현 관점을 나타내는 클래스, 프로세스나 방법에 무관
    - 설계 클래스 : UP 설계 모델의 구성요소. 소프트웨어 클래스와 유사하지만, 설계 모델에 있는 클래스.
    - 구현 클래스 : 자바 같은 객체 지향 언어로 구현된 클래스
  - 클래스 : UML에서와 같이 실세계 사물(개념 클래스) 또는 소프트웨어 사물(소프트웨어 클래스)를 나타내는 일반적 용어

Lowering the Representational Gap(1/2)

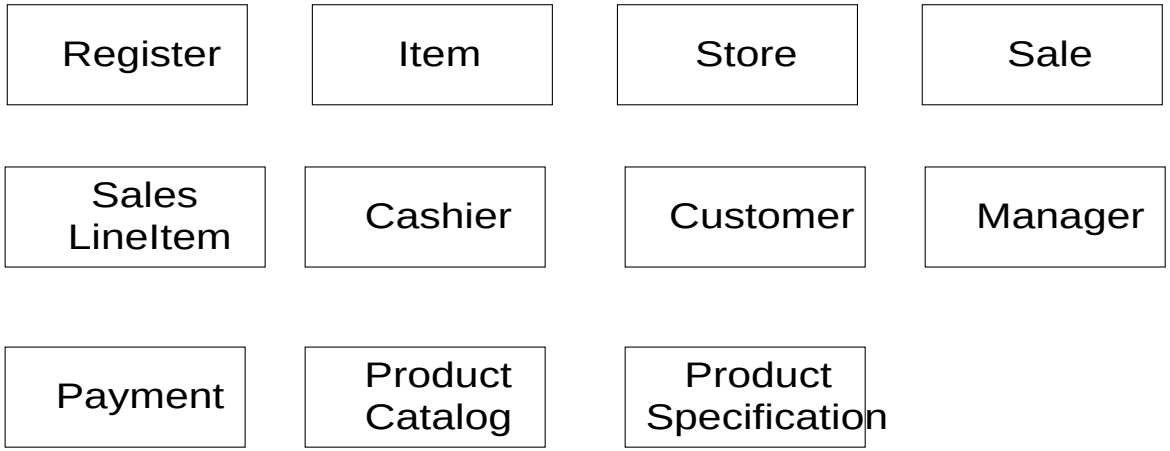
도메인 모델은 도메인 단어와 개념에 대해 시각적인 사전을 제공하는데 소프트웨어 설계에서 명명 시 “직관”을 사용한다.



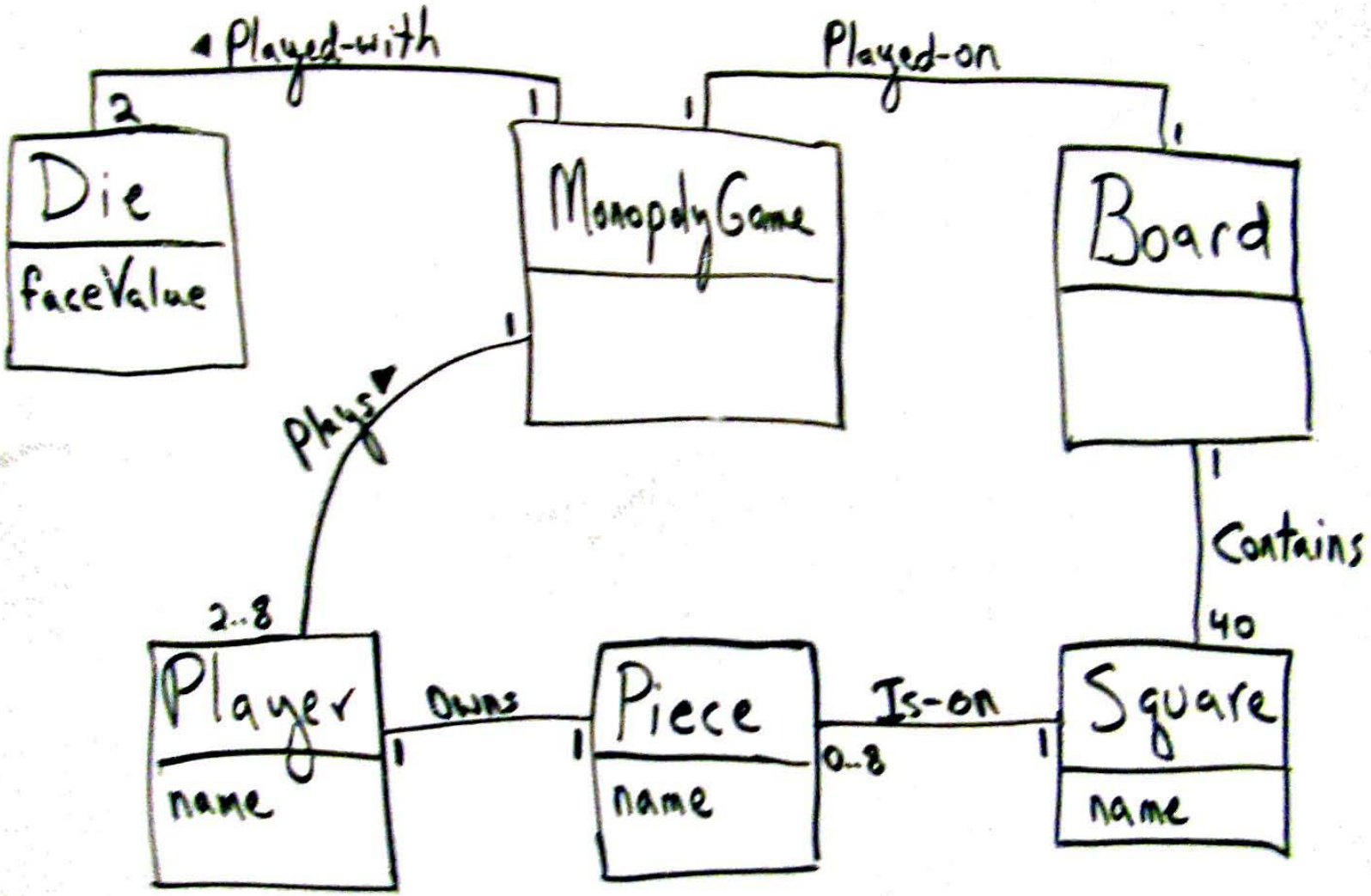
Lowering the Representational Gap(2/2)

- 표현상의 차이(의미상 차이) 줄이기: 심리상의 모델과 소프트웨어에서의 표현 간
  - 도메인 단어와 소프트웨어 단어를 1:1로 매핑

Example : The NextGen POS Domain Model



예) 부분적인 도메인 모델


Played-with
Played-on
2
1
1
1
Die
faceValue
MonopolyGame
Board
Plays
1
1
Contains
2..8
40
Player
name
Owns
1
1
Piece
name
Is-on
0..8
1
Square
name

Domain Models Within the UP(1/2)

Discipline	Artifact Iteration →	Incep. I1	Elab. E1..En	Const. C1..Cn	Trans. T1..T2
Business Modeling	*도메인 모델*		**s**		
Requirements	유스케이스 모델	**s**	**r**		
	비전	**s**	**r**		
	보충명세서	**s**	**r**		
	용어집	**s**	**r**		
Design	설계 모델		**s**	**r**	
	소프트웨어 아키텍춰 모델		**s**		
	데이터 모델		**s**	**r**	
Implementation	**Implementation Model**		**s**	**r**	**r**
Project Management	**SW Development Plan**	**s**	**r**	**r**	**r**
Testing	**Test Model**		**s**	**r**	
Environment	**Development Case**	**s**	**r**		

Domain Models Within the UP(2/2)

- UP 비즈니스 객체 모델(BOM) vs. 도메인 모델

UP BOM
비즈니스 작업자와 비즈니스 엔터티들이 어떻게 서로 연관되고
비즈니스를 수행하기 위해 어떻게 협업하는지에 대한 추상화

  - UP에서 도메인 모델은 BOM의 부분집합 또는 특수화한 BOM
    - “도메인에서 중요한 “사물”과 프로덕트에 초점을 맞춘 불완전한 BOM”

Further Readings

- Object-Oriented Methods A Foundation: Odell
  - 개념적 도메인 모델링의 소개
- Analysis Patterns : Fowler
  - 도메인 모델의 패턴
- Java Modeling in Color with UML : Coad
  - 공통 패턴 식별
- Object-Oriented Analysis from Textual Specifications : Moreno [Moreno97]
  - 자연어 분석 예

UP Artifacts