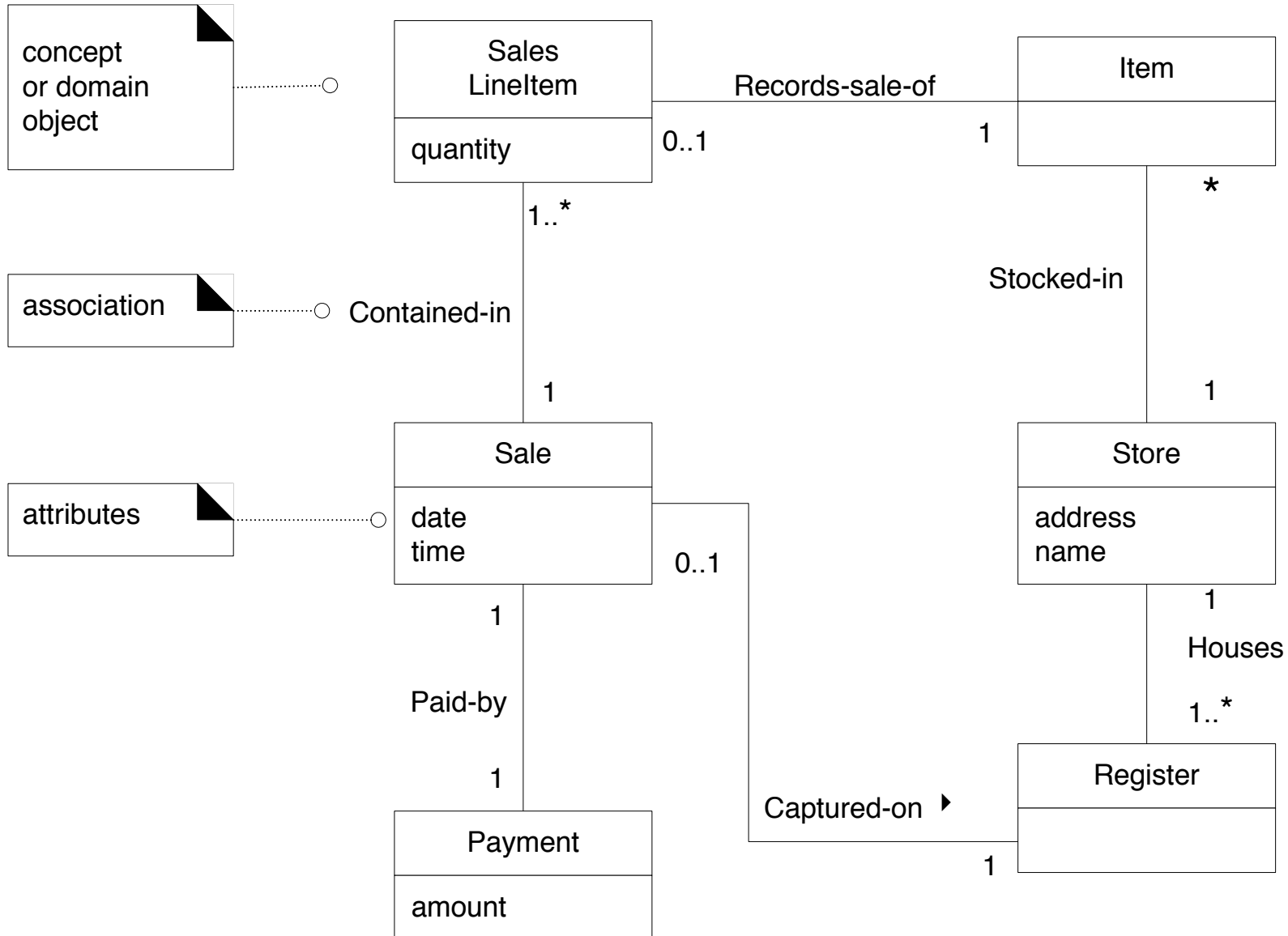


# **Chapter 9**

## **Domain Models**

# Domain Model in UML Class Diagram Notation



A “visual dictionary”

## Domain Models

- Key object-oriented analysis step: Decompose domain into noteworthy concepts or objects
- UML class diagrams used to draw domain models
  - Conceptual perspective. Shows:
    - Domain objects (conceptual classes)
    - Associations between domain objects
    - Attributes of conceptual classes
- Domain model is NOT a model of software objects or our design
- The following should NOT be in a domain model
  - Software artifacts: Window, database, ...
  - Responsibilities or methods

**Fig. 9.3**



visualization of a real-world concept in  
the domain of interest

it is a *not* a picture of a software class

**Fig. 9.4**

avoid



software artifact; not part  
of domain model

avoid

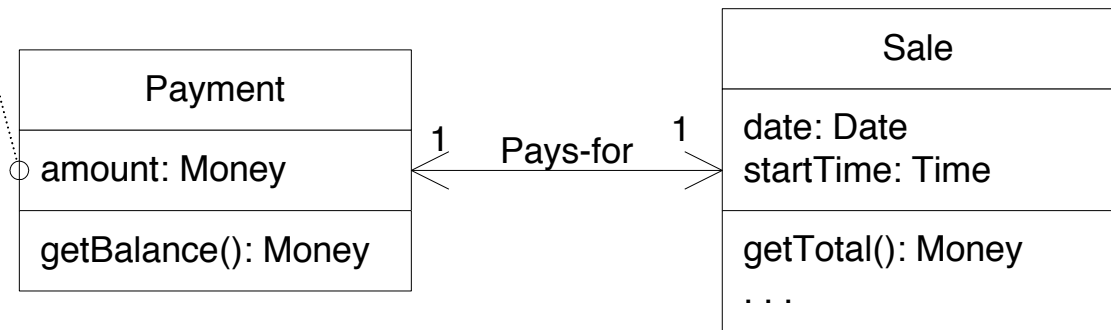
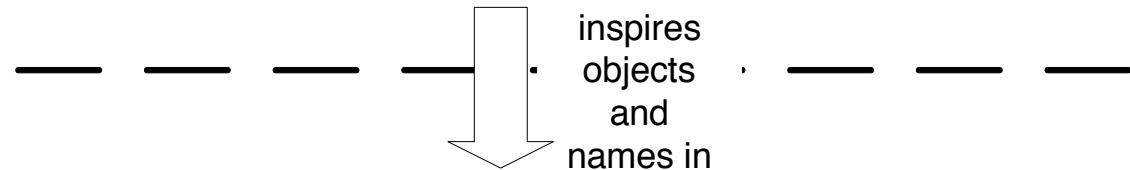
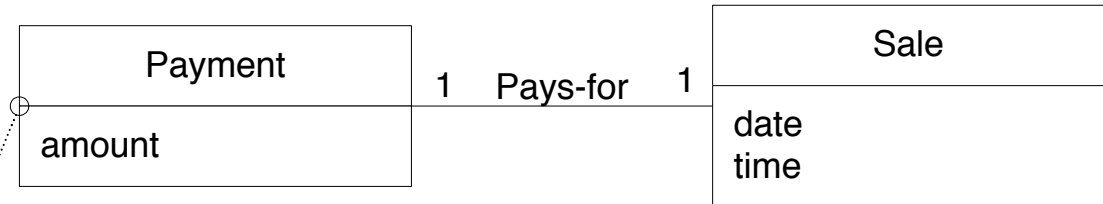


software class; not part  
of domain model

# Software Class Names Inspired by Domain Model Objects

## UP Domain Model

Stakeholder's view of the noteworthy concepts in the domain.



## UP Design Model

The object-oriented developer has taken inspiration from the real world domain in creating software classes.

Therefore, the representational gap between how stakeholders conceive the domain, and its representation in software, has been lowered.

A **Payment** in the Domain Model is a concept, but a **Payment** in the Design Model is a software class. They are not the same thing, but the former *inspired* the naming and definition of the latter.

This reduces the representational gap.

This is one of the big ideas in object technology.

## How to create a domain model?

- Find conceptual classes
  - How?
    - Re-use or modify existing models
    - Use a category list
    - Identify noun phrases
    - More on these later
- Draw them as classes in a UML class diagram
- Add associations and attributes
  - More on this later

Conceptual Class Category	Examples
physical or tangible objects	<i>Register</i> <i>Airplane</i>
specifications, designs, or descriptions of things	<i>ProductSpecification</i> <i>FlightDescription</i>
places	<i>Store</i> <i>Airport</i>
transactions	<i>Sale, Payment</i> <i>Reservation</i>
transaction line items	<i>SalesLineItem</i>
roles of people	<i>Cashier</i> <i>Pilot</i>
containers of other things	<i>Store, Bin</i> <i>Airplane</i>
things in a container	<i>Item</i> <i>Passenger</i>



other computer or electro-mechanical systems external to the system	<i>CreditPaymentAuthorizationSystem</i> <i>AirTrafficControl</i>
abstract noun concepts	<i>Hunger</i> <i>Acrophobia</i>
organizations	<i>SalesDepartment</i> <i>ObjectAirline</i>
events	<i>Sale, Payment, Meeting</i> <i>Flight, Crash, Landing</i>
processes (often <i>not</i> represented as a concept, but may be)	<i>SellingAProduct</i> <i>BookingASeat</i>
rules and policies	<i>RefundPolicy</i> <i>CancellationPolicy</i>
catalogs	<i>ProductCatalog</i> <i>PartsCatalog</i>

Conceptual Class Category	Examples
records of finance, work, contracts, legal matters	<i>Receipt, Ledger, EmploymentContract MaintenanceLog</i>
financial instruments and services	<i>LineOfCredit Stock</i>
manuals, documents, reference papers, books	<i>DailyPriceChangeList RepairManual</i>

# Identifying nouns as candidates for domain objects

## Main Success Scenario (or Basic Flow):

1. **Customer** arrives at a **POS checkout** with **goods** and/or **services** to purchase.
2. **Cashier** starts a new **sale**.
3. **Cashier** enters **item identifier**.
4. System records **sale line item** and presents **item description**, **price**, and running **total**. Price calculated from a set of price rules.  
Cashier repeats steps 2-3 until indicates done.
5. System presents total with **taxes** calculated.
6. Cashier tells Customer the total, and asks for **payment**.
7. Customer pays and System handles payment.
8. System logs the completed **sale** and sends sale and payment information to the external **Accounting** (for accounting and **commissions**) and **Inventory** systems (to update inventory).
9. System presents **receipt**.
10. Customer leaves with receipt and goods (if any).

## Extensions (or Alternative Flows):

### 7a. Paying by cash:

1. Cashier enters the cash **amount tendered**.
2. System presents the **balance due**, and releases the **cash drawer**.
3. Cashier deposits cash tendered and returns balance in cash to Customer.
4. System records the cash payment.

## Candidate Conceptual Classes: Process Sale, Iteration 1

Register

Item

Store

Sale

Sales  
LineItem

Cashier

Customer

Ledger

Cash  
Payment

Product  
Catalog

Product  
Description

Fig. 9.8

Monopoly Game

Player

Piece

Die

Board

Square

## Attributes vs. Classes

- Should “something” be
  - an attribute of a class, or
  - a separate conceptual class
- Examples:
  - Store: An attribute of Sale or separate class
    - Store: Not a number or text
    - Should be separate conceptual class
  - Flight destination: An attribute or a separate class
    - Destination is an airport, not a number
    - Should be separate conceptual class
- Guideline:
  - If we think of something as simply a number or text in real life, it should be an attribute.
  - Otherwise it should be a conceptual class.

I HIRED ALL OF YOU  
BECAUSE THE PROJECT  
WILL TAKE 300 MAN  
DAYS TO COMPLETE.



www.dilbert.com  
scottadams@aol.com

THERE ARE 300 OF YOU,  
SO I WANT YOU TO  
FINISH BY FIVE O'CLOCK  
AND CLEAN OUT YOUR  
DESKS. YOU'RE ALL  
FIRED.



9-3-07 © 2007 Scott Adams, Inc./Dist. by UFS, Inc.

IF IT TAKES MORE  
THAN ONE MEETING  
TO MANAGE A PROJECT,  
I LOSE INTEREST.



## **“Description” Classes**

- A description class contains information that describes something else
  - Example: ProductDescription
  - Records price, picture and text description of an item
- Why use them? Why not include all that information in the Product class?
  - We need this information stored and represented even though there are no objects of that particular product type.
  - Don't want to duplicate product information for each duplicate product object
    - Serial number belongs with product object
    - Picture of product belongs with product description
- Need objects that are “specifications” or “descriptions” of other objects

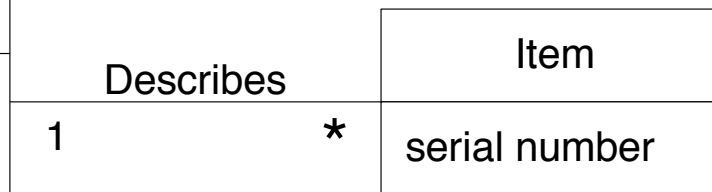


# Description class example

Item
description price serial number itemID

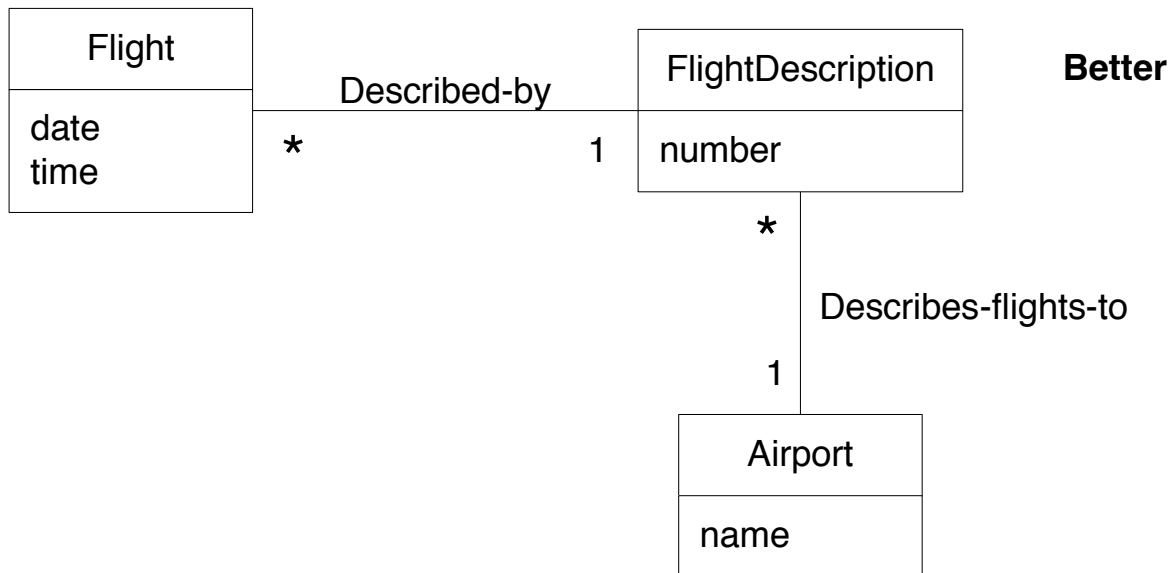
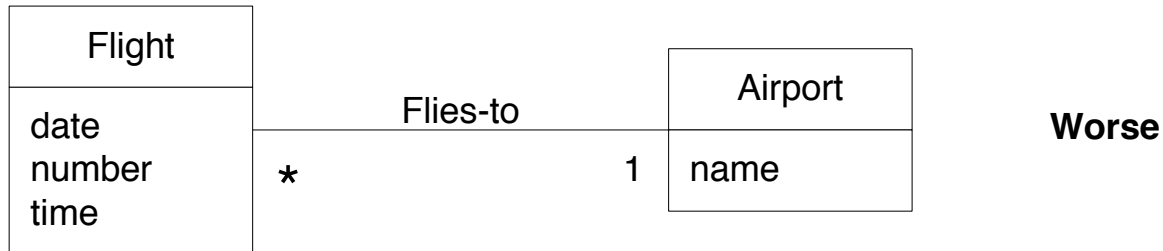
**Worse**

ProductDescription
description price itemID



**Better**

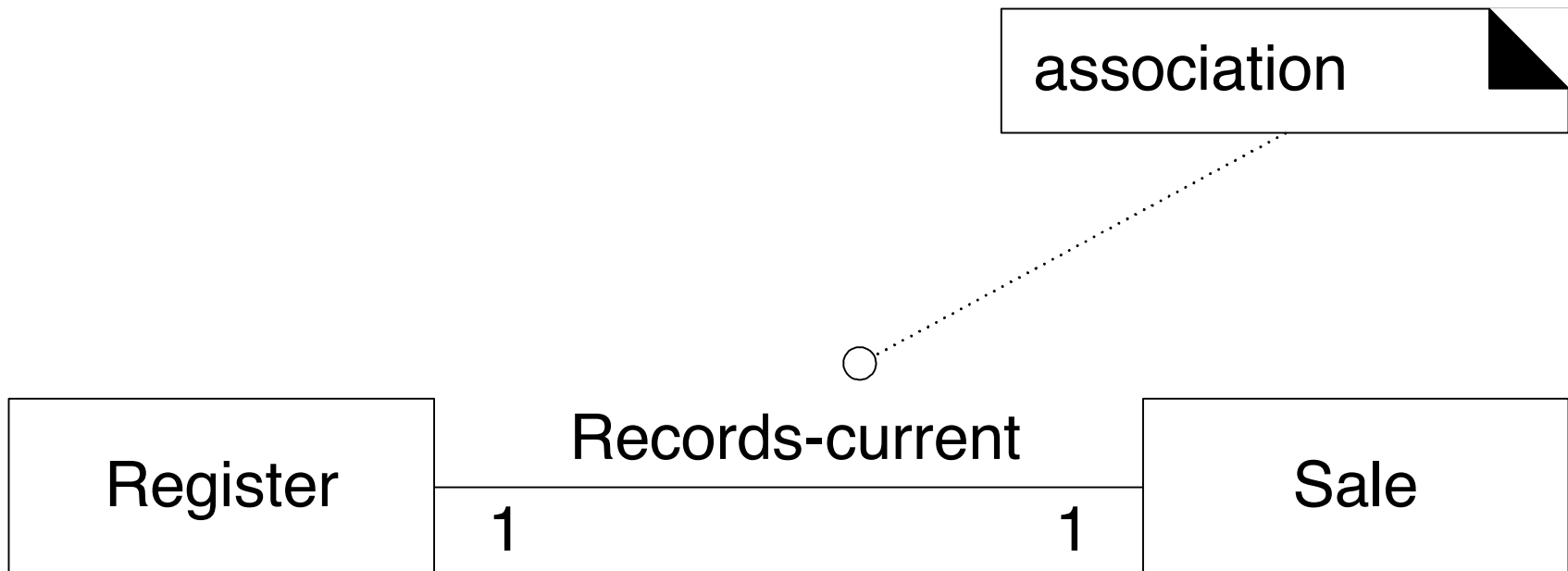
# Description class example



Even when that flight is not scheduled that day, the flight description exists.

## Associations

- Association: A relationship between (instances of) classes that indicates some meaningful and interesting connection.
- Used to show relationships that need to be remembered and preserved for some duration



## Which relationships need to be remembered?

- Example: Does a Sale-SalesLineItem relationship need to be remembered (preserved)?\
  - Yes, otherwise can't process returns or exchanges.
- Example: Cashier looks up ProductDescription
  - Don't need to remember/store.
- Example:
  - What square is a Monopoly player on?
    - Need to remember
  - Dice total tells us which Square to move to
    - Do we need to store this fact with the Dice or the Square?
    - No!

## Common Associations List (i)

Category	Examples
A is a physical part of B	<i>Drawer — Register (or more specifically, a POST)</i> <i>Wing — Airplane</i>
A is a logical part of B	<i>SalesLineItem — Sale</i> <i>FlightLeg—FlightRoute</i>
A is physically contained in/on B	<i>Register — Store, Item — Shelf</i> <i>Passenger — Airplane</i>
A is logically contained in B	<i>ItemDescription — Catalog</i> <i>Flight— FlightSchedule</i>
A is a description for B	<i>ItemDescription — Item</i> <i>FlightDescription — Flight</i>
A is a line item of a transaction or report B	<i>SalesLineItem — Sale</i> <i>Maintenance Job — Maintenance-Log</i>
A is known/logged/recorded/reported/captured in B	<i>Sale — Register</i> <i>Reservation — FlightManifest</i>

## Common Associations List (ii)

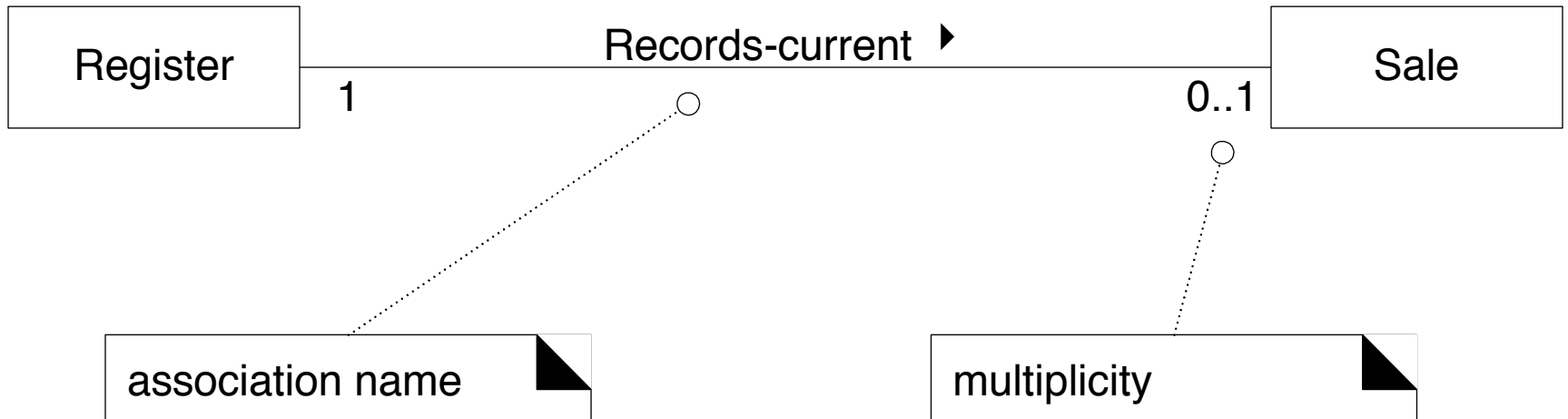
A is a member of B	<i>Cashier — Store</i> <i>Pilot — Airline</i>
A is an organizational subunit of B	<i>Department — Store</i> <i>Maintenance — Airline</i>
A uses or manages B	<i>Cashier — Register</i> <i>Pilot — Airplane</i>
A communicates with B	<i>Customer — Cashier</i> <i>Reservation Agent — Passenger</i>
A is related to a transaction B	<i>Customer — Payment</i> <i>Passenger — Ticket</i>
A is a transaction related to another transaction B	<i>Payment — Sale</i> <i>Reservation — Cancellation</i>
A is next to B	<i>SalesLineItem — SalesLineItem</i> <i>City — City</i>

## Common Associations List (iii)

Category	Examples
A is owned by B	<i>Register — Store</i> <i>Plane — Airline</i>
A is an event related to B	<i>Sale — Customer, Sale — Store</i> <i>Departure — Flight</i>

# Association Directionalities

- "reading direction arrow"
- it has **no** meaning except to indicate direction of reading the association label
- often excluded

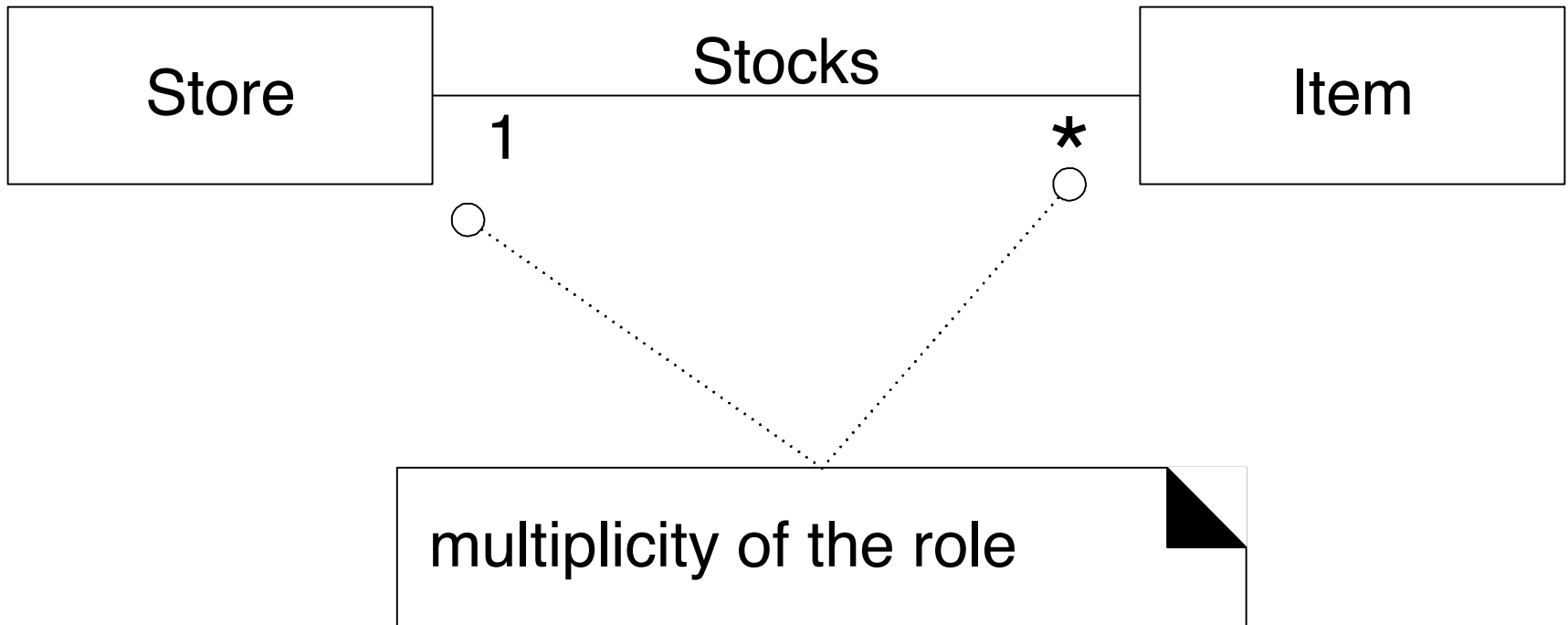




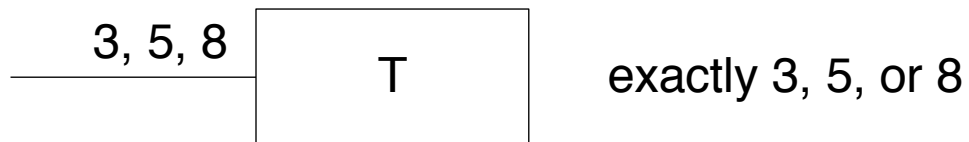
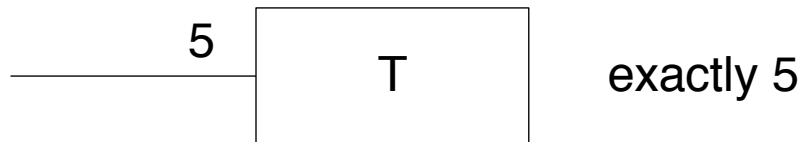
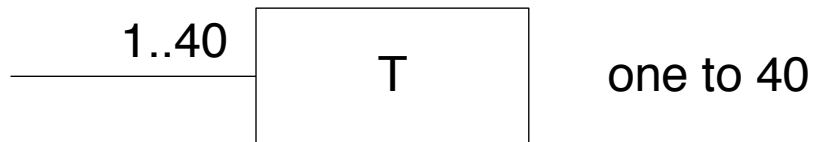
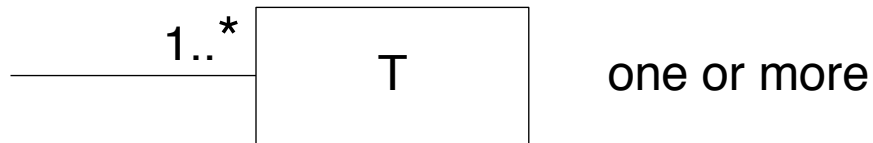
## How to name an association?

- Pattern:
  - Class name – verb phrase – class name
  - Readable and meaningful
- Try to avoid “has” or “uses”. These give no extra information

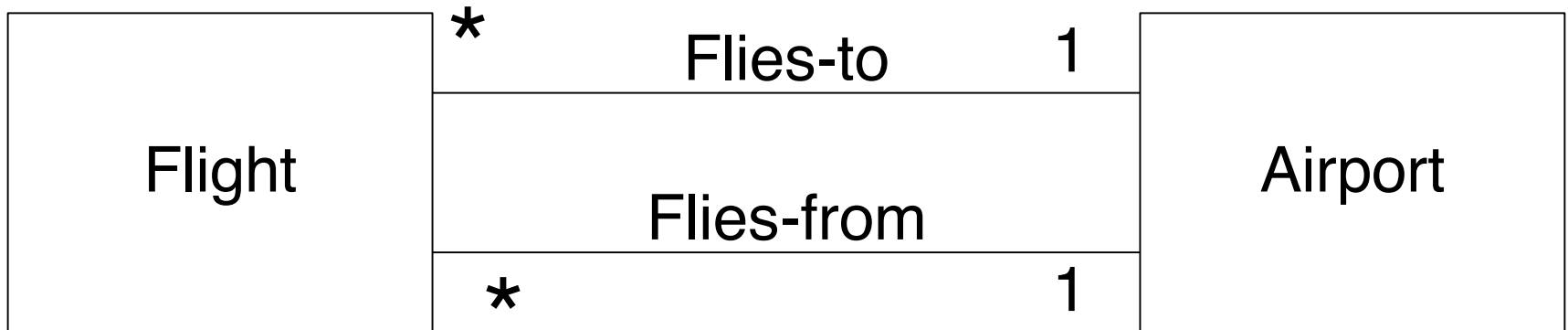
# Association Multiplicities



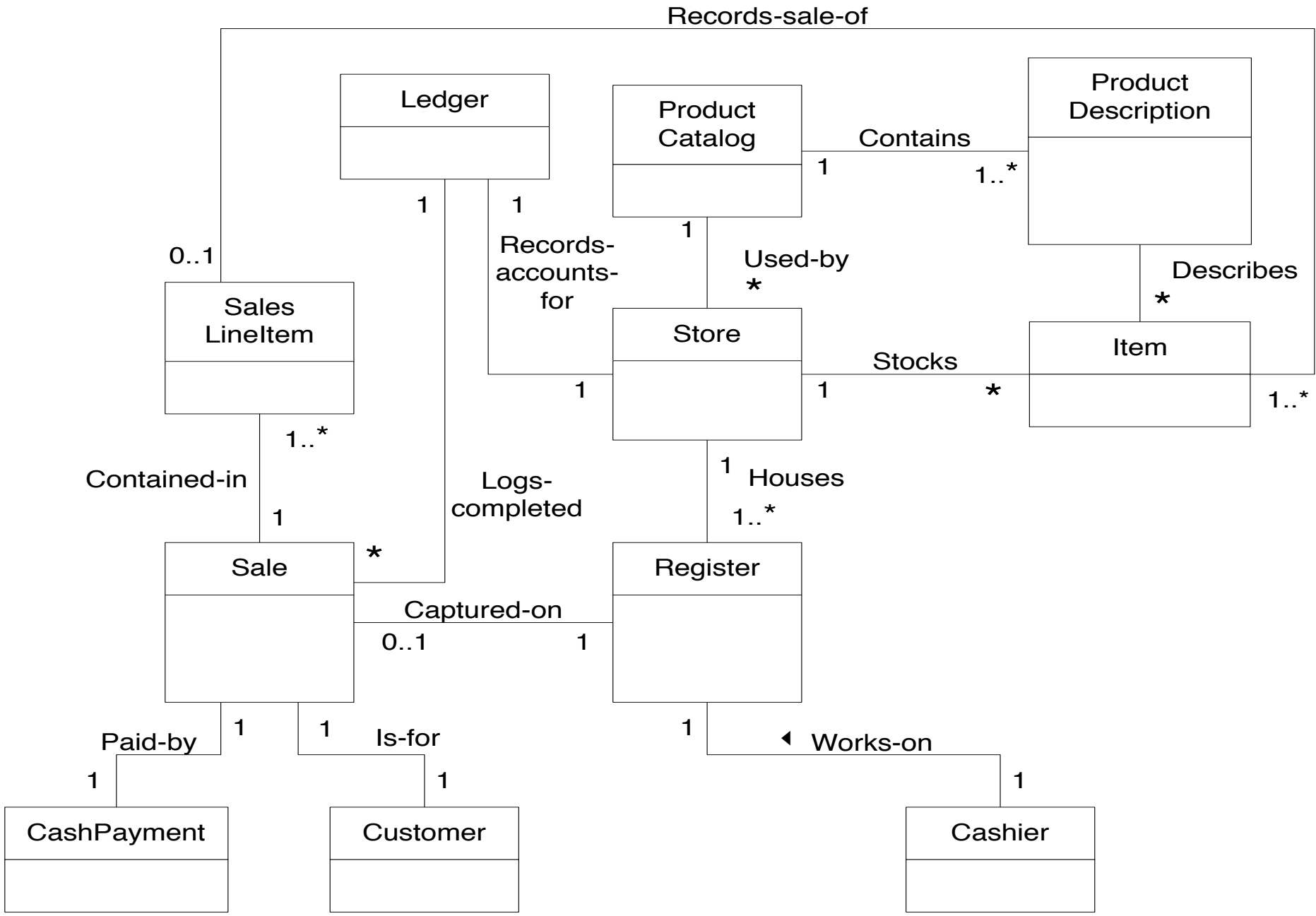
# Multiplicities



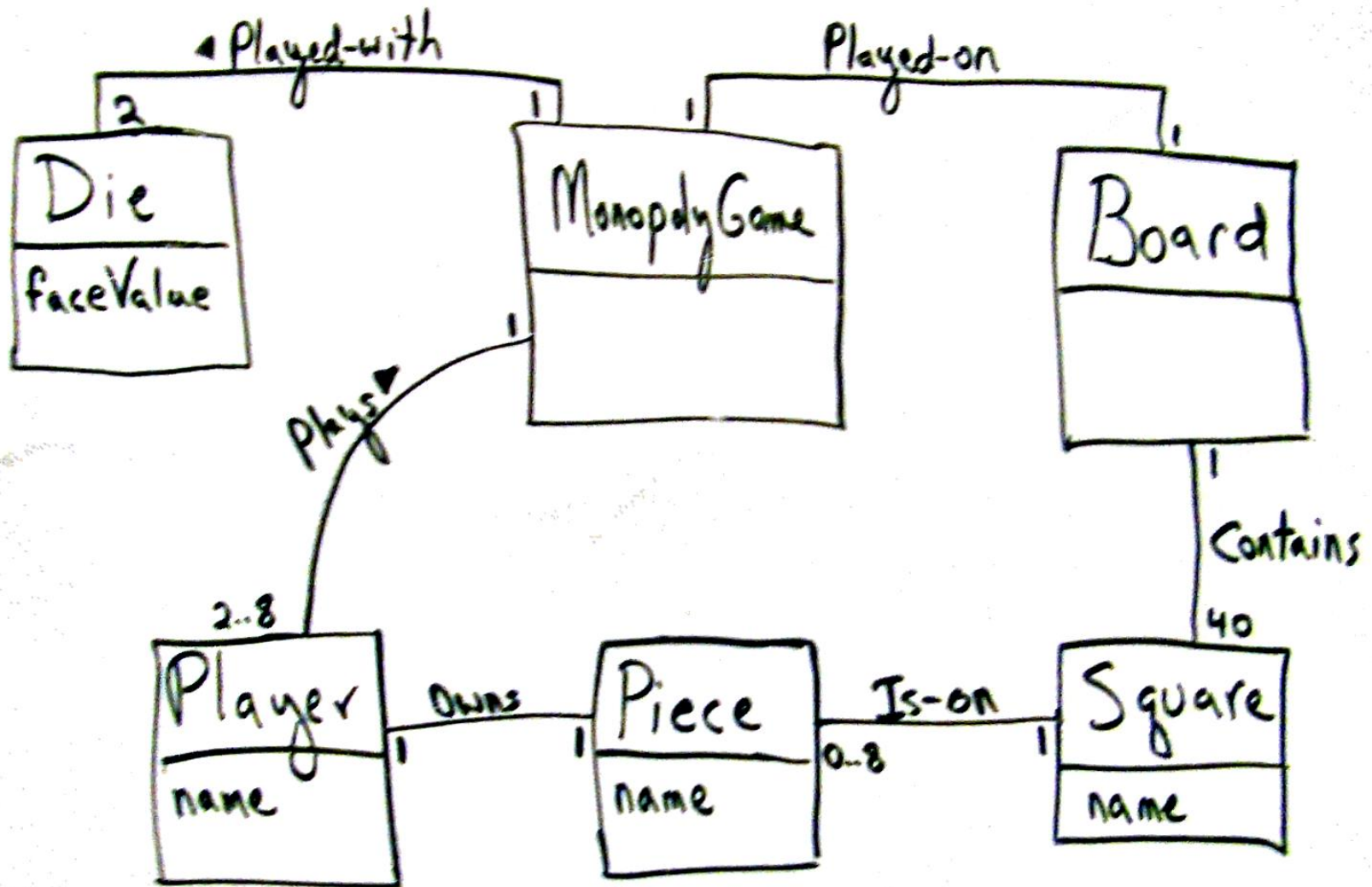
## Multiple Associations



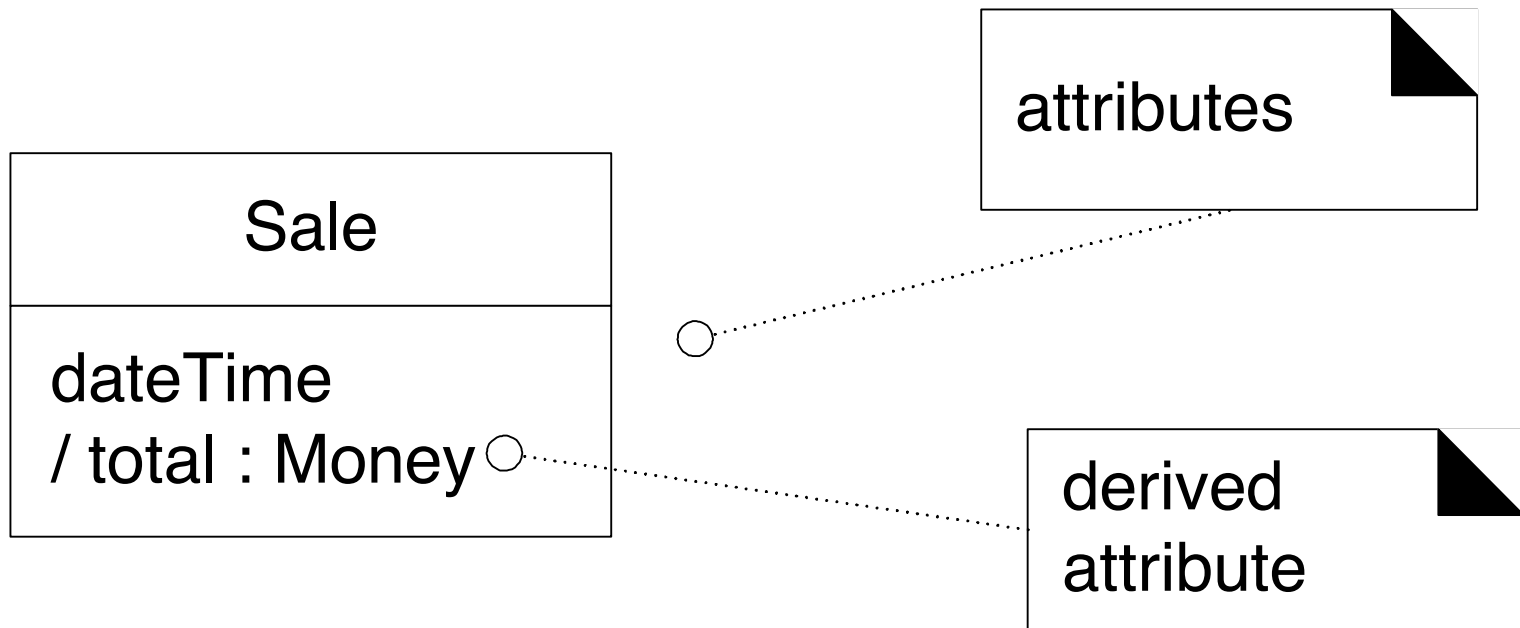
# POS Domain Model Example:



## Example: Domain Model for the Monopoly Game



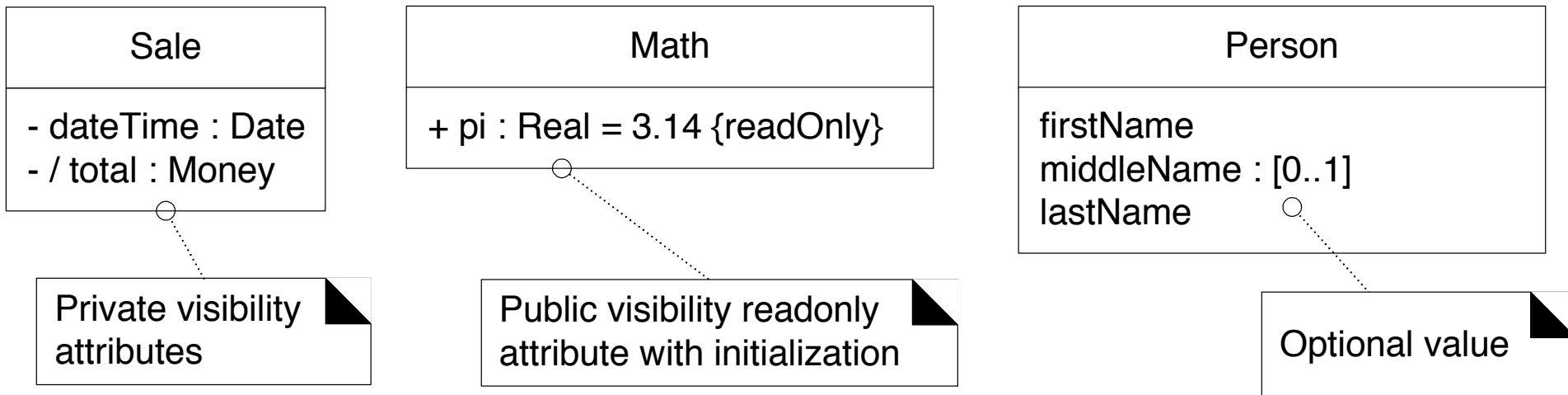
## Attributes



- An attribute: A logical data value stored in an object

## More detailed attribute notation

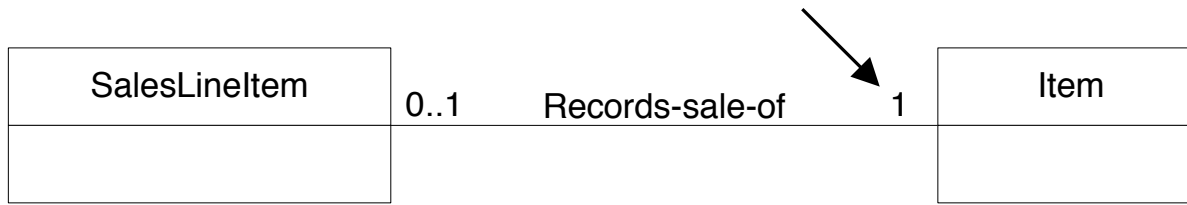
visibility name: type multiplicity = default value {property-string}



- Attribute requirements (such as an optional middle name) should also be placed in the Glossary.



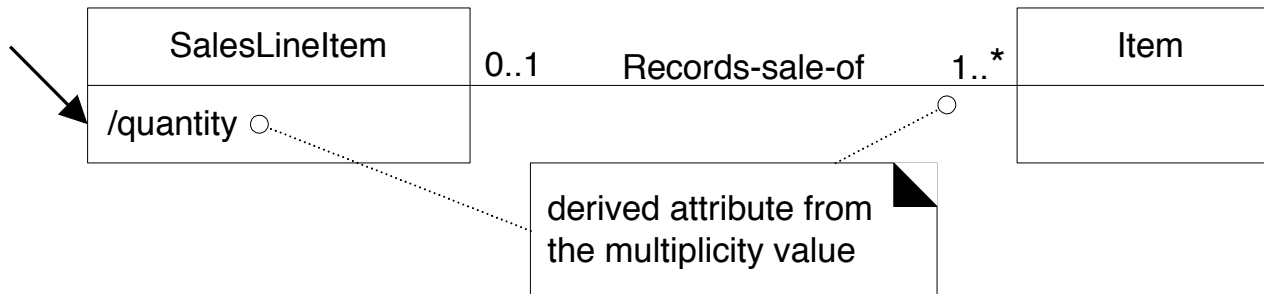
# Derivable attributes



Each line item records a separate item sale.  
For example, 1 tofu package.

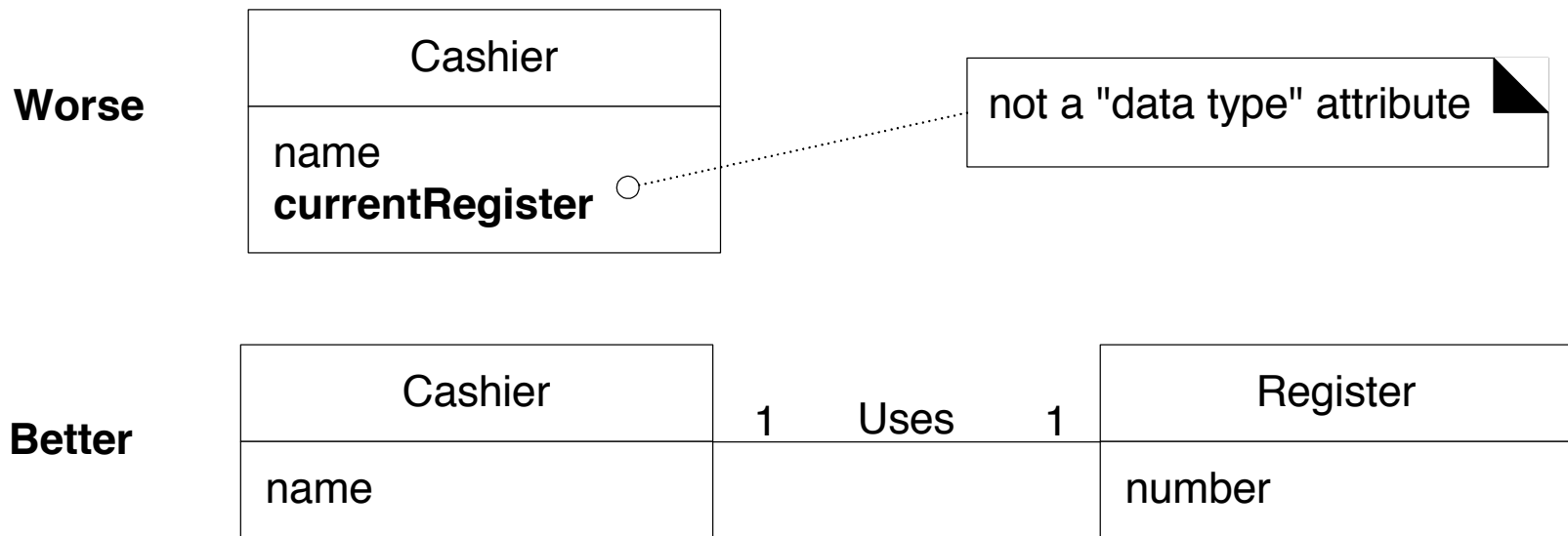


Each line item can record a group of the same kind of items.  
For example, 6 tofu packages.



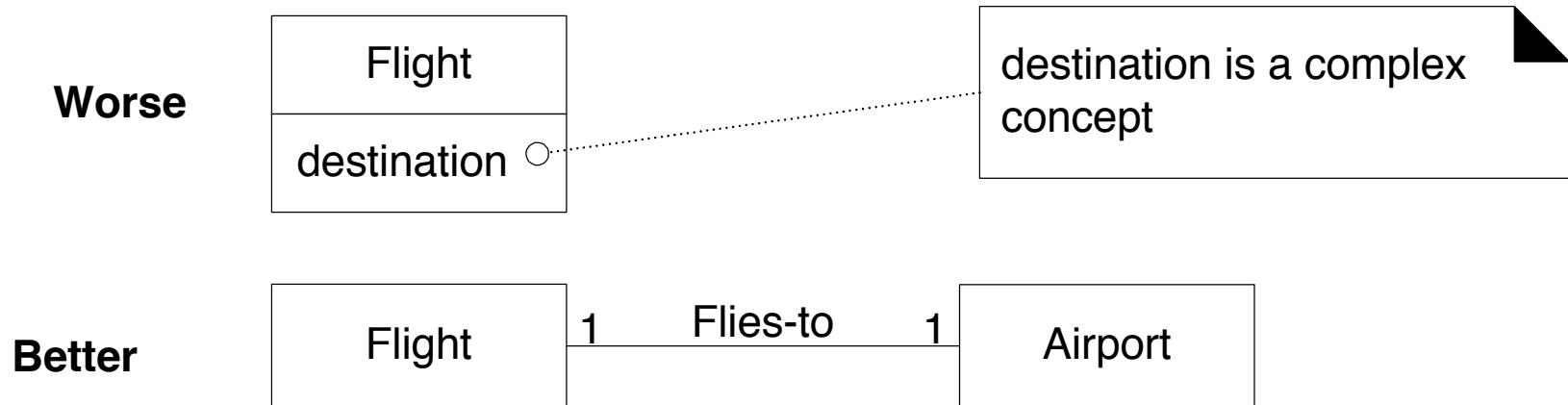
- The “/” sign: the value of this attribute can be calculated or derived from other attributes
- Still, it is noteworthy and may be recorded separately

## Most attribute types should be primitive data types



- Guideline: The attributes in a domain model should be (simple, primitive) data types
  - Boolean, Date, Number (Integer, Real), String (Text), Time, Phone Number, ID Number, Postal Code, ...
- Non-data type relationships (i.e., conceptual class relationships) should be expressed using associations, not attributes.

# How to tell “data types” from “conceptual classes”?



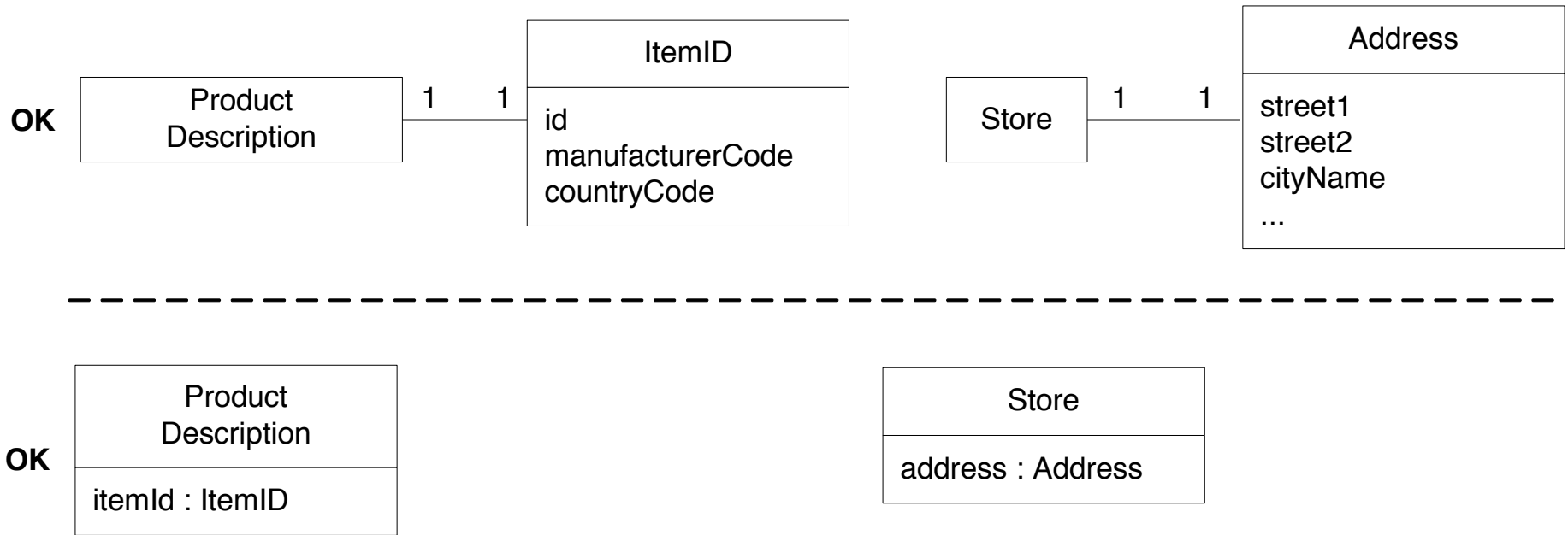
- Guideline: Is the equality test based on identity or value
  - Examples: Equality test based on value
    - Separate instances of the Integer 5
    - Separate instances of the date “February 21, 2005”
  - Examples: Equality test based on identity
    - Two employees with the same name
    - Two copies of the same book

## When to define a new class?

Represent what may initially be considered a primitive data type (such as a number or string) as a non-primitive class if:

- It is composed of separate sections.
  - o phone number, name of person
- There are operations usually associated with it, such as parsing or validation.
  - o social security number
- It has other attributes.
  - o promotional price could have a start (effective) date and end date
- It is a quantity with a unit.
  - o payment amount has a unit of currency
- It is an abstraction of one or more types with some of these qualities.
  - o item identifier in the sales domain is a generalization of types such as Universal Product Code (UPC) or European Article Number (EAN)

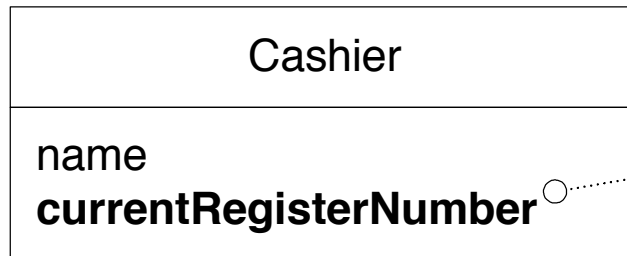
# Do we need a new box for these classes?



## Attributes representing “foreign keys”

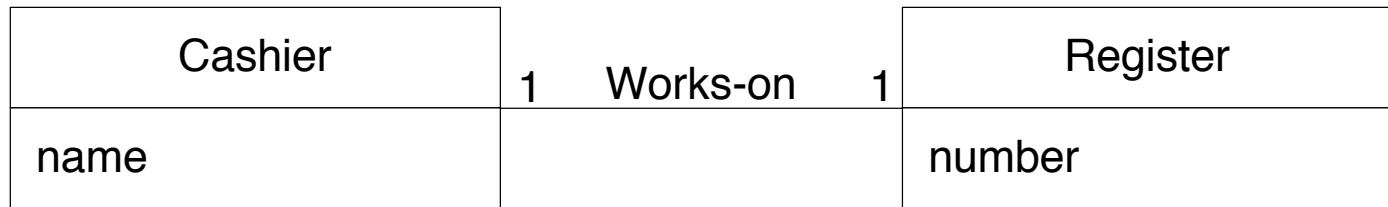
- Avoid them!

**Worse**

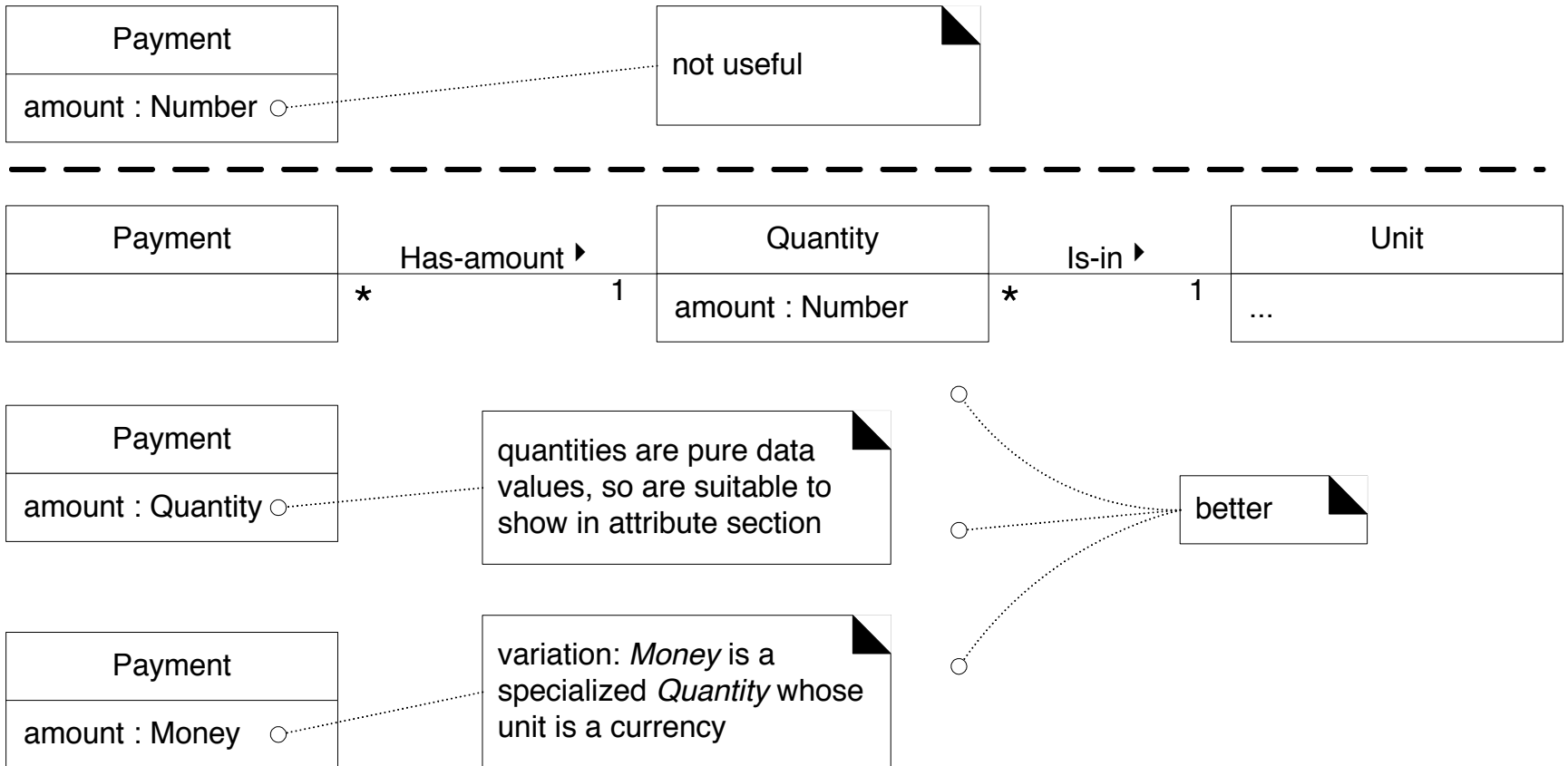


a "simple" attribute, but being used as a foreign key to relate to another object

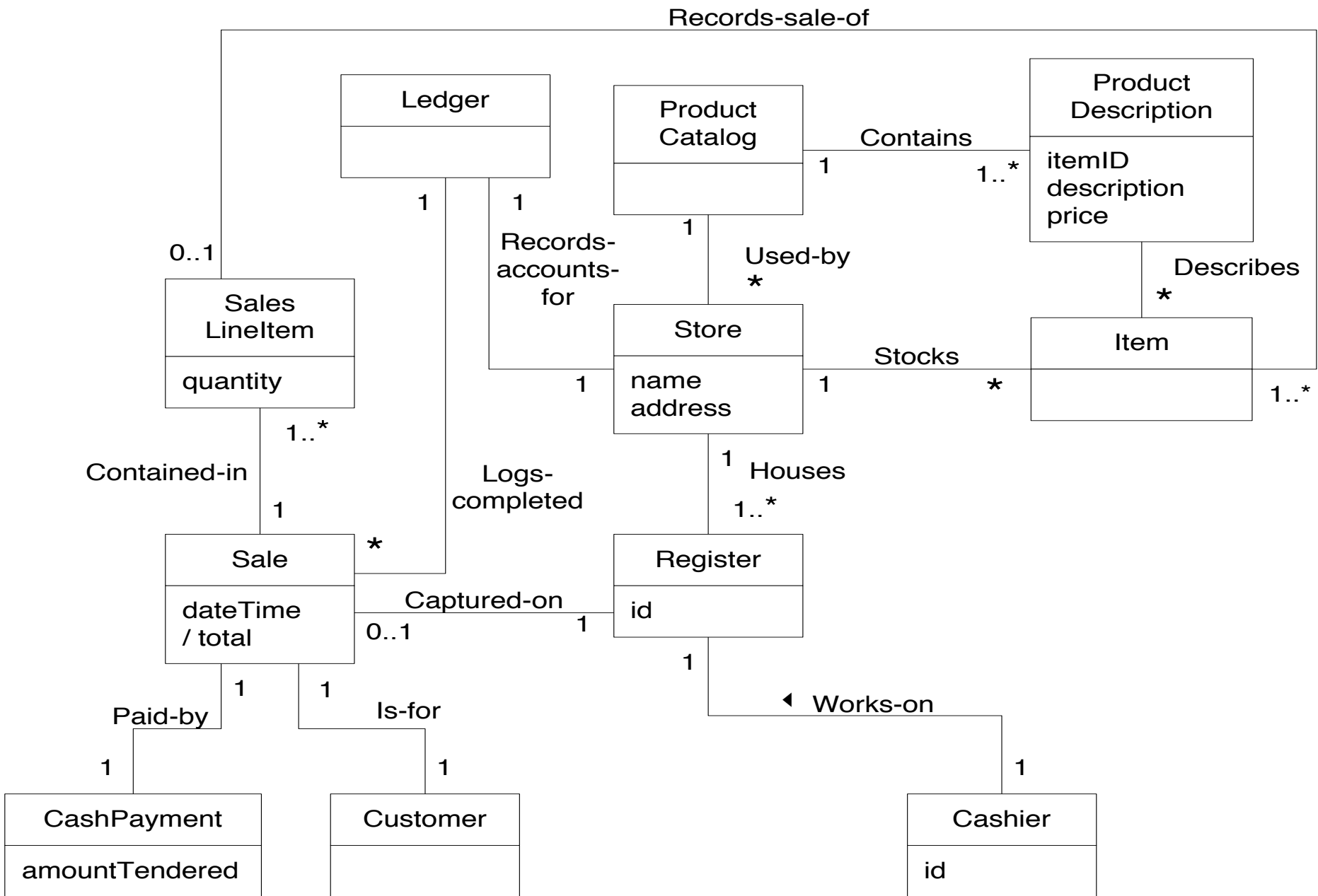
**Better**



# Fig. 9.26



# POS Domain Model Example:





# POS Domain Model Example:

The attributes chosen reflect the requirements for this iteration—the Process *Sale* scenarios of this iteration.

<i>Payment</i>	<i>amount</i> —To determine if sufficient payment was provided, and to calculate change, an amount (also known as "amount tendered") must be captured.
<i>Product-Specification</i>	<i>description</i> —To show the description on a display or receipt.  <i>id</i> —To look up a <i>ProductSpecification</i> , given an entered itemID, it is necessary to relate them to a <i>id</i> .  <i>price</i> —To calculate the sales total, and show the line item price.
<i>Sale</i>	<i>date, time</i> —A receipt is a paper report of a sale. It normally shows date and time of sale.
<i>SalesLineItem</i>	<i>quantity</i> —To record the quantity entered, when there is more than one item in a line item sale (for example, <i>five</i> packages of tofu).
<i>Store</i>	<i>address, name</i> —The receipt requires the name and address of the store.

# Monopoly Domain Model Example

