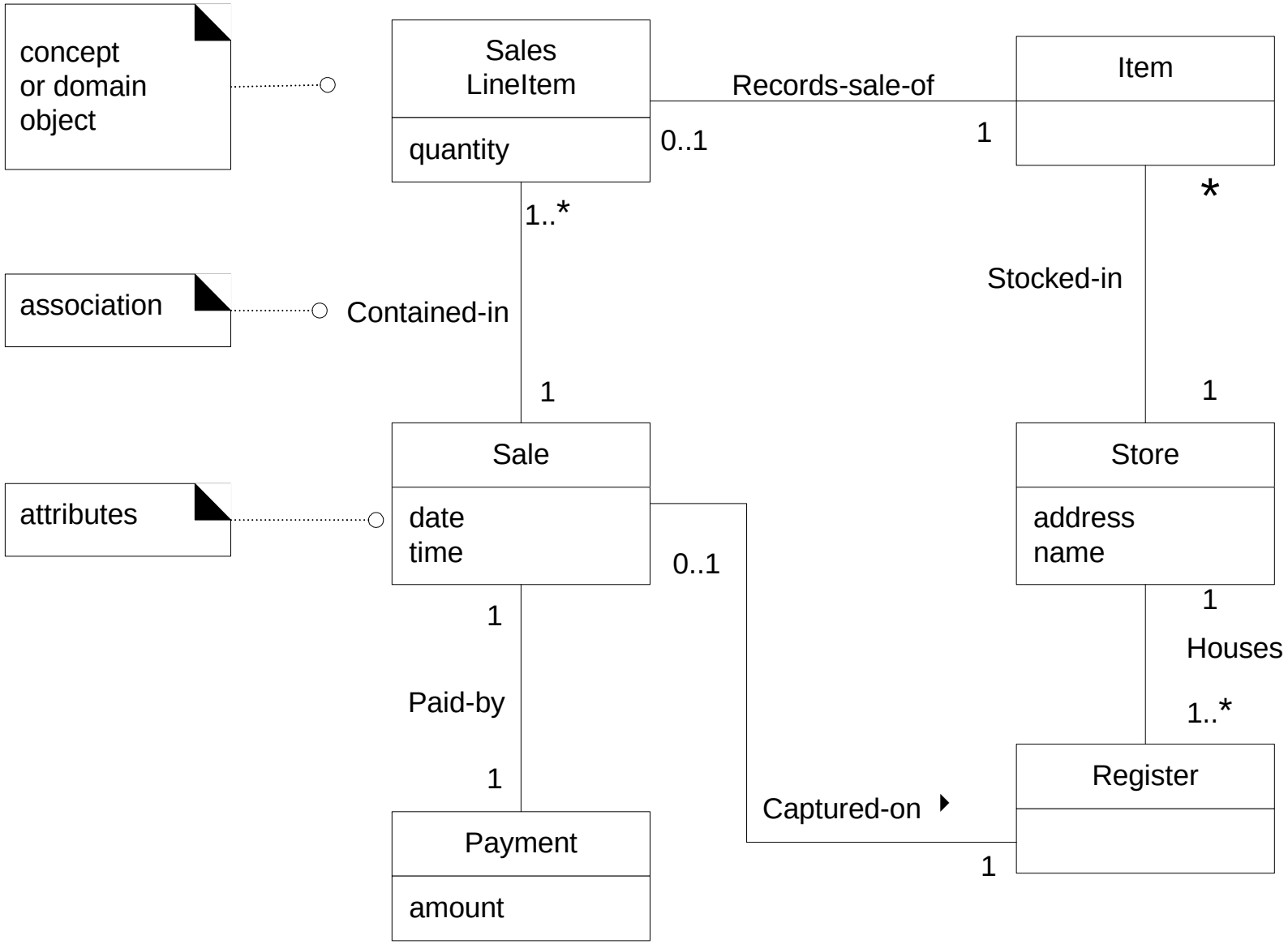
Domain Model

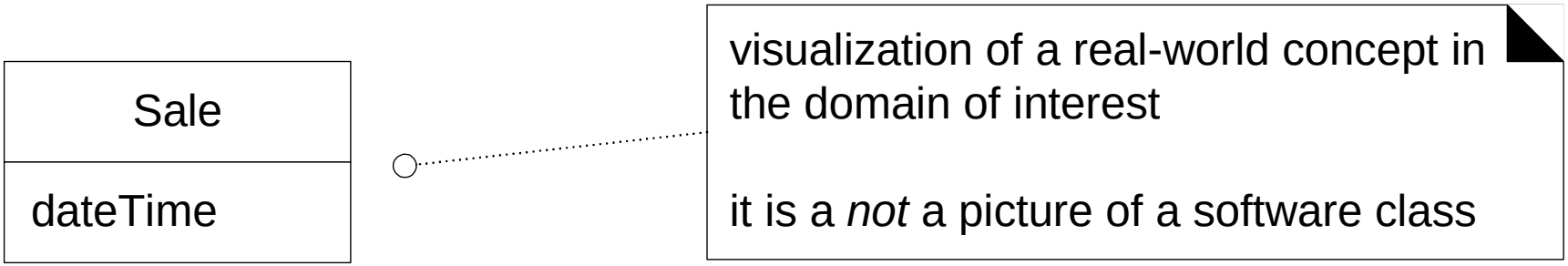
Domain Model in UML Class Diagram Notation

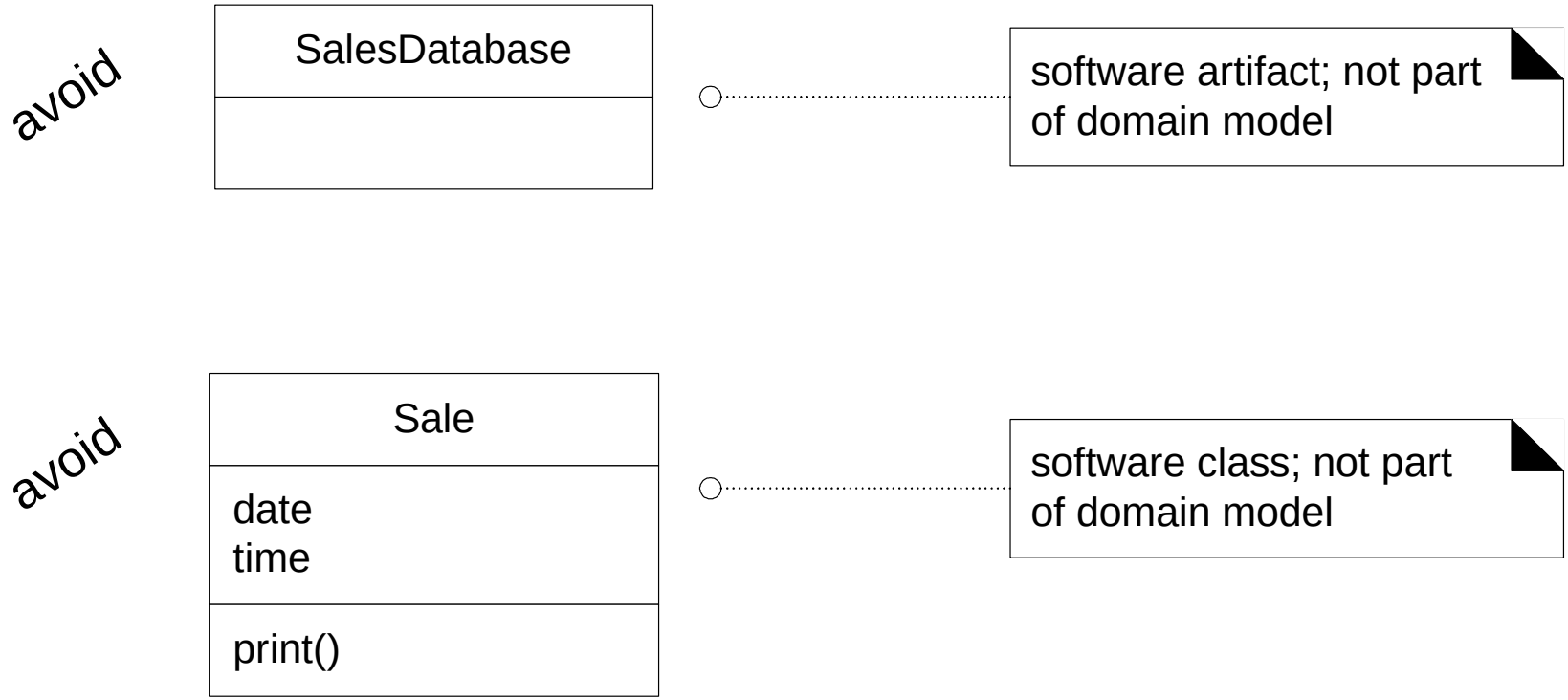


A “visual dictionary”

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


Sale
dateTime
visualization of a real-world concept in the domain of interest
it is a *not* a picture of a software class


avoid
SalesDatabase
software artifact; not part of domain model
avoid
Sale
date
time
print()
software class; not part of domain model

Software Class Names Inspired by Domain Model Objects

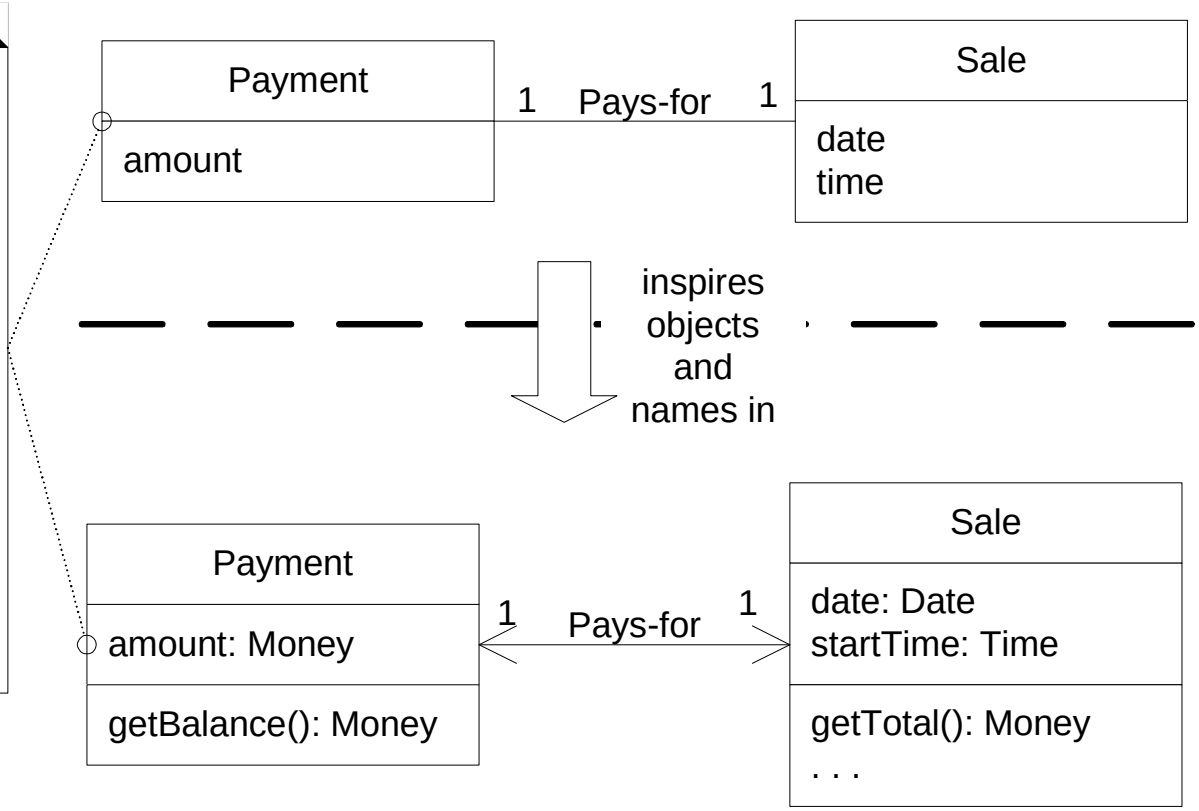
A Payment in the Domain Model is a concept, but a Payment in the Design Model is a software class. They are not the same thing, but the former *inspired* the naming and definition of the latter.

This reduces the representational gap.

This is one of the big ideas in object technology.

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.

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	*Register* *Airplane*
specifications, designs, or descriptions of things	*ProductSpecification* *FlightDescription*
places	*Store* *Airport*
transactions	*Sale, Payment* *Reservation*
transaction line items	*SalesLineItem*
roles of people	*Cashier* *Pilot*
containers of other things	*Store, Bin* *Airplane*
things in a container	*Item* *Passenger*

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

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

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.

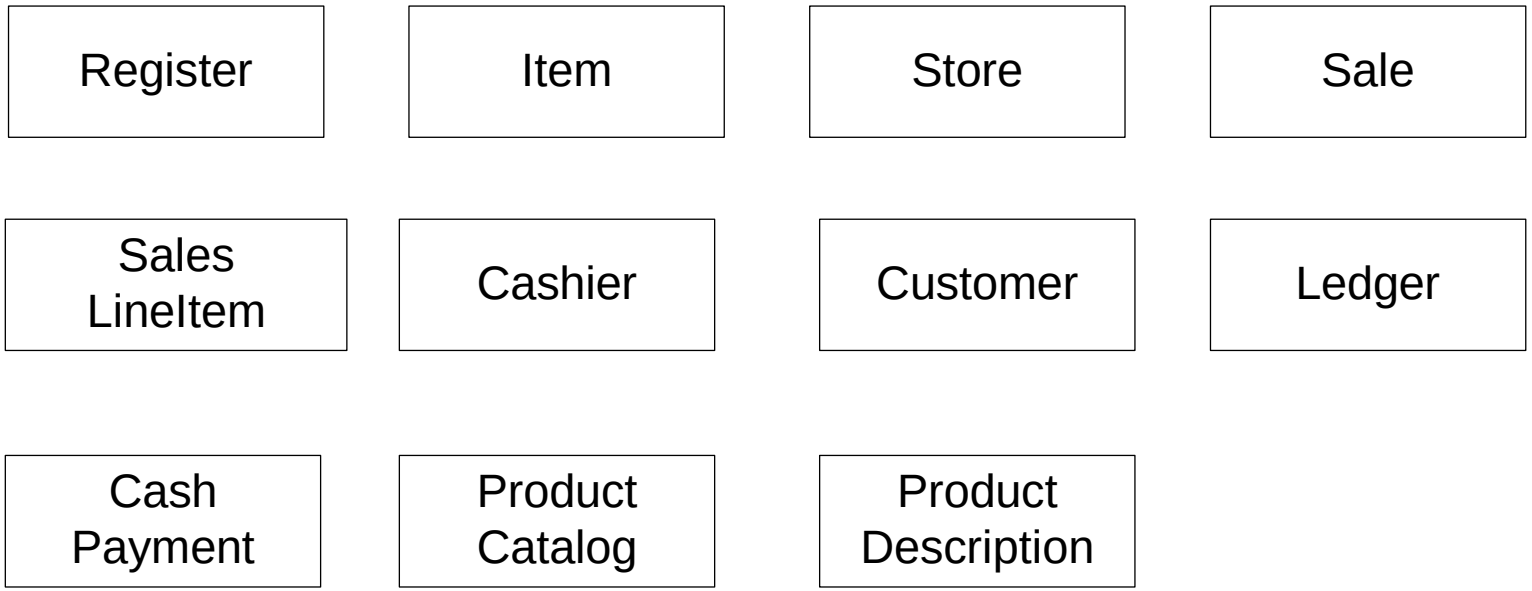
5. System presents total with **taxes** calculated.
6. Cashier tells Customer the total, and asks for **payment.**
7. Customer pays and System handles payment.
8. System logs the completed **sale** and sends sale and payment information to the external **Accounting** (for accounting and **commissions)** and **Inventory** systems (to update inventory).
9. System presents **receipt.**
10. Customer leaves with receipt and goods (if any).

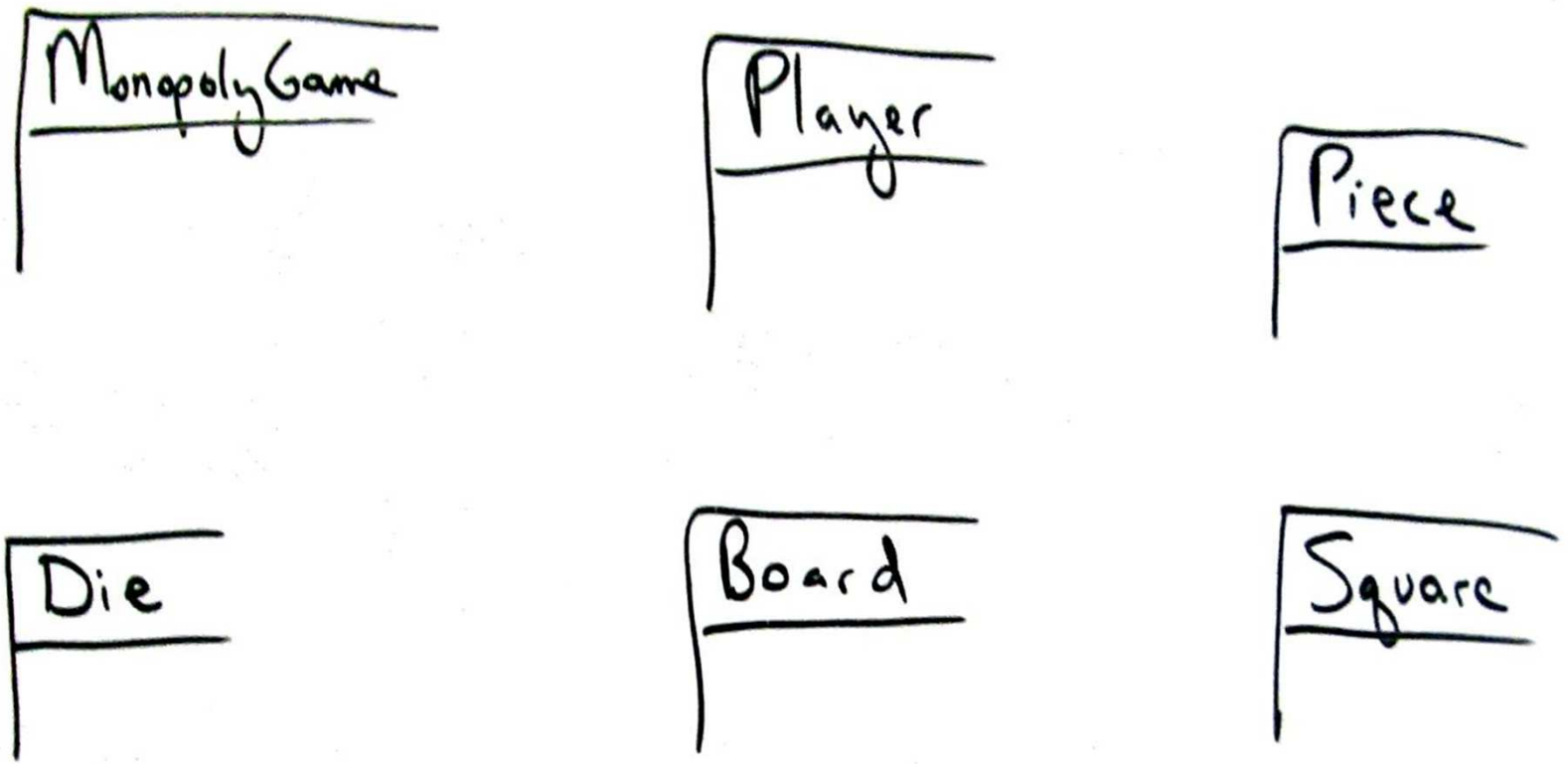
Extensions (or Alternative Flows):

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




MonopolyGame
Player
Piece
Die
Board
Square

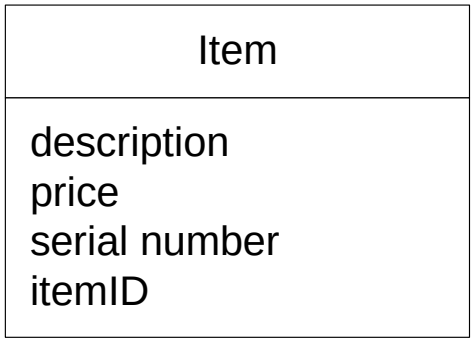
Attributes vs. Classes

- 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 thing of something as simply a number or text in real life, it should be an attribute.
  - Otherwise it should be a conceptual class.

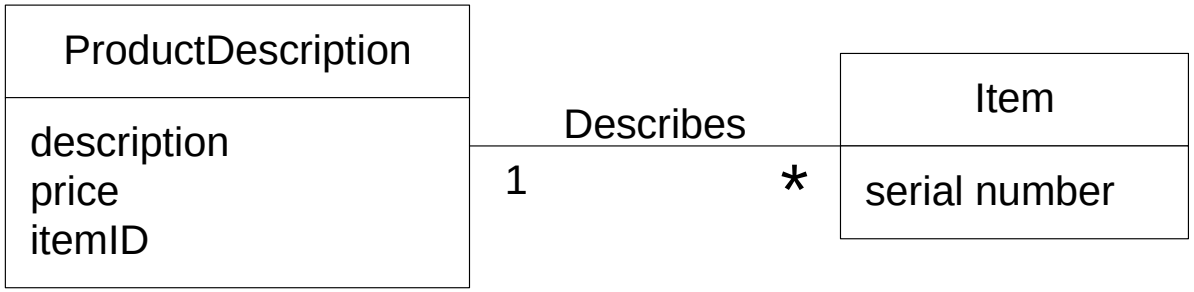
“Description” Classes

- 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

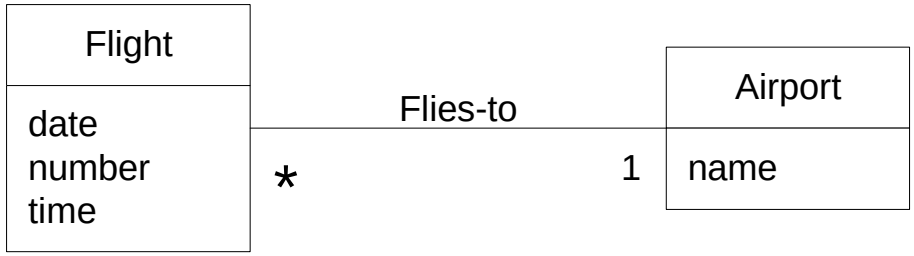


Worse

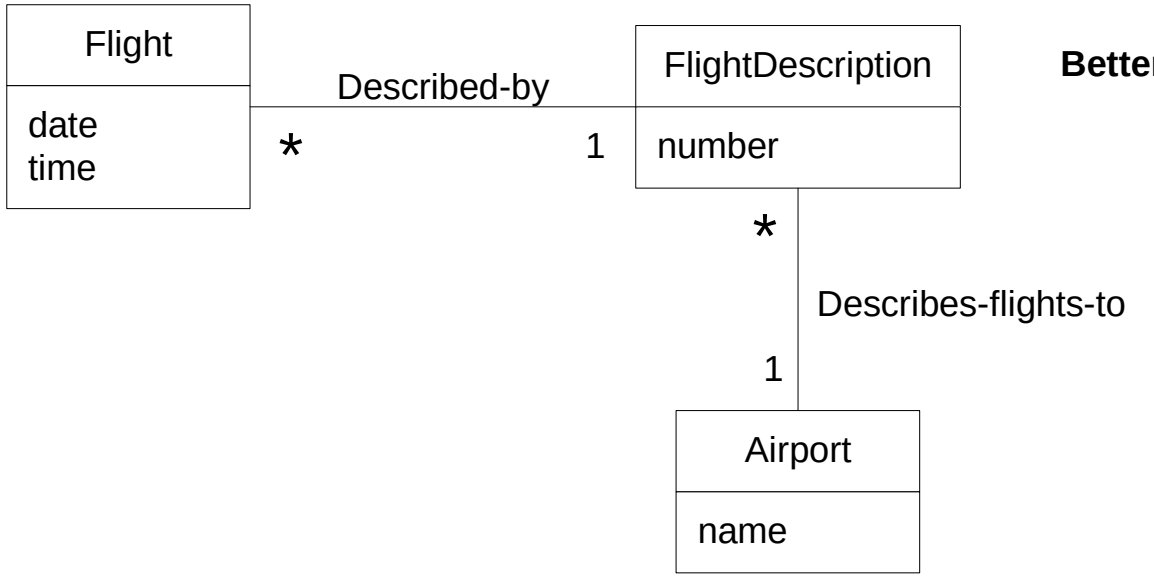


Better

Description class example



Worse

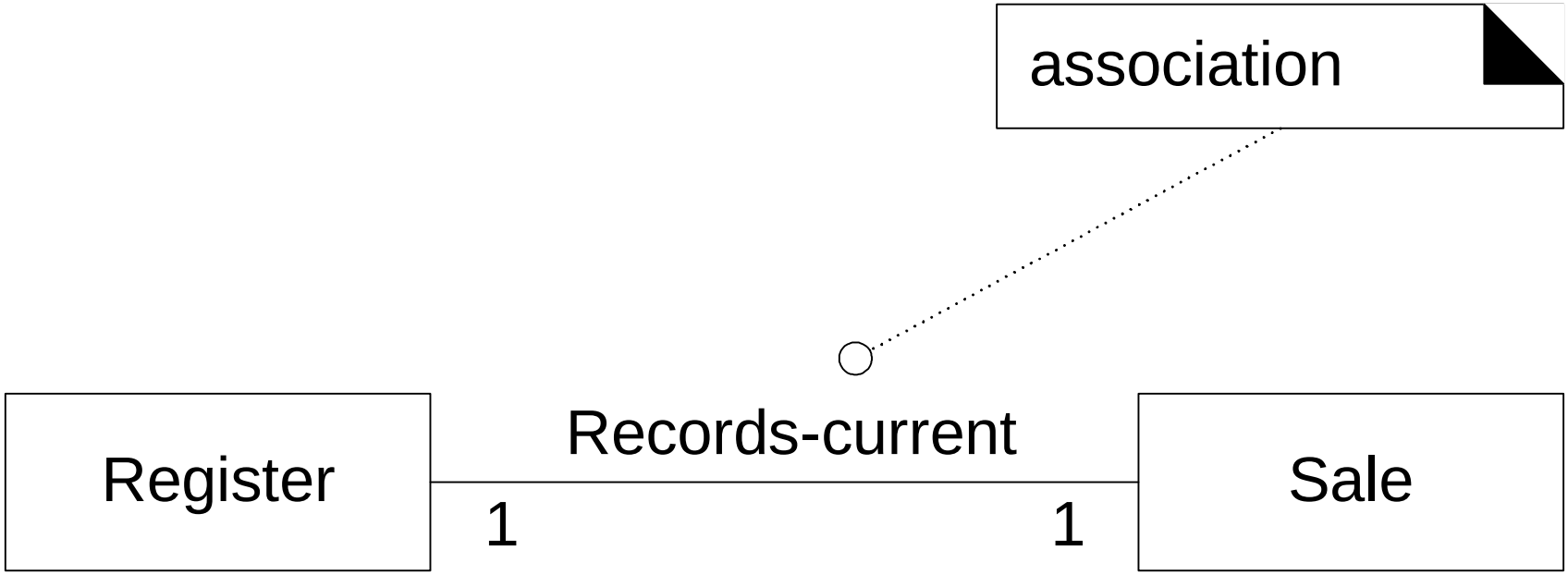


Better

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

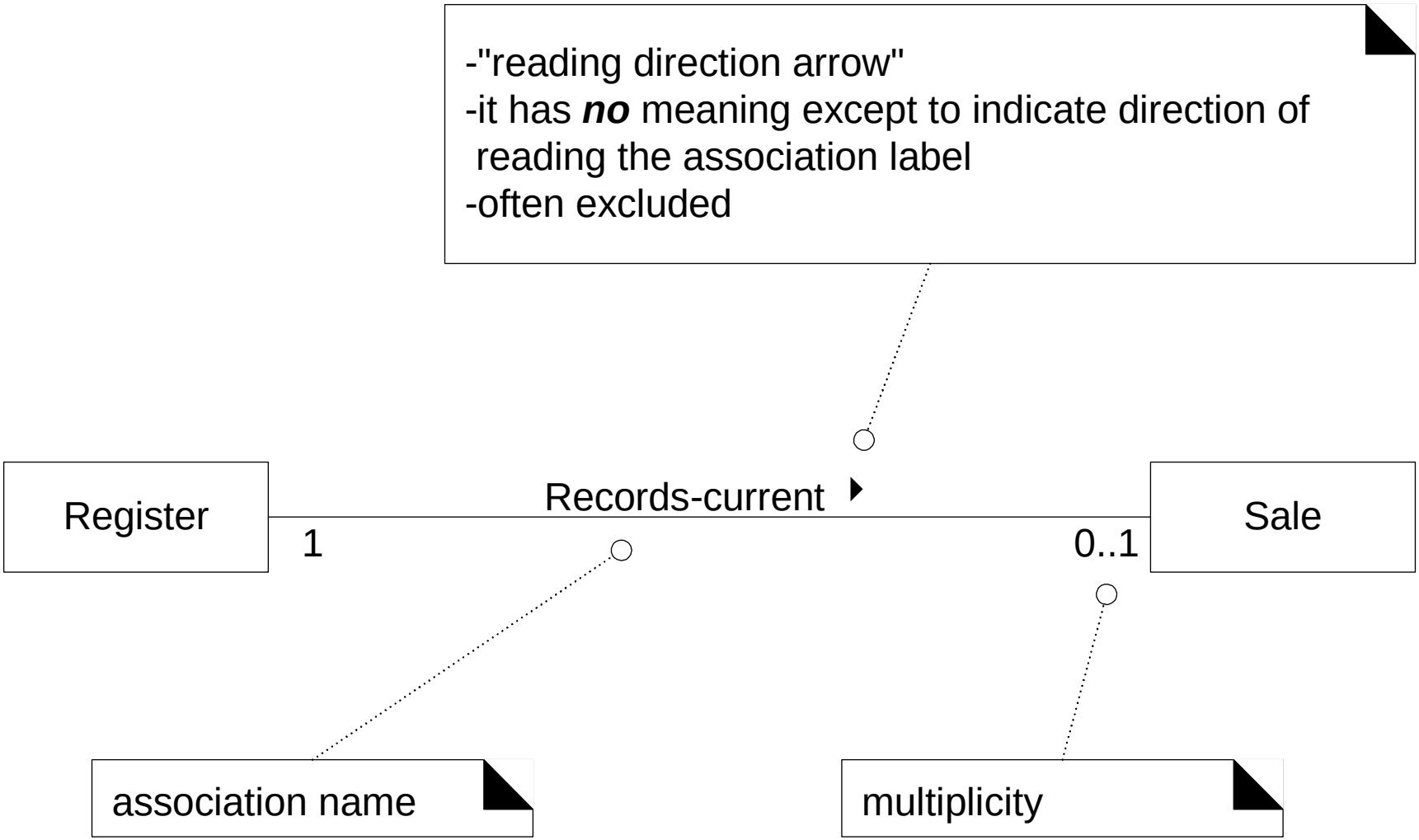
Common Associations List (ii)

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

Common Associations List (iii)

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

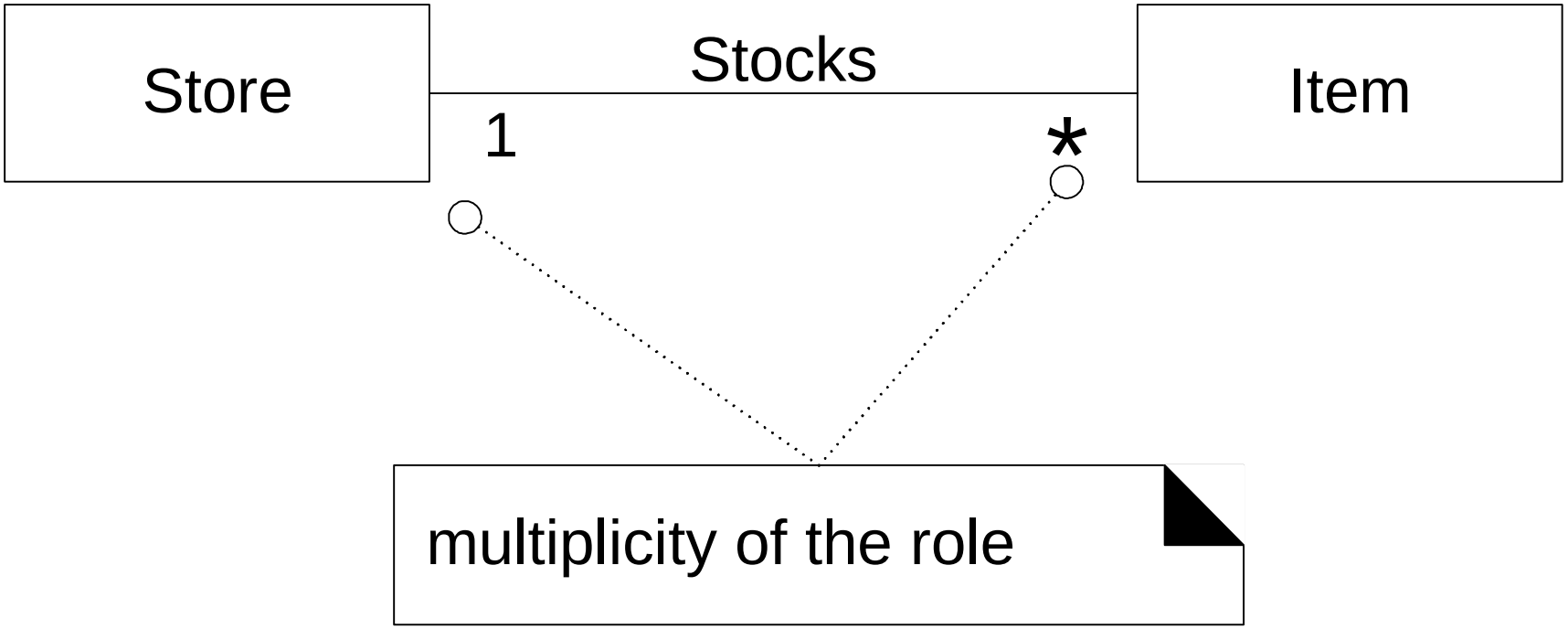
Association Directionalities



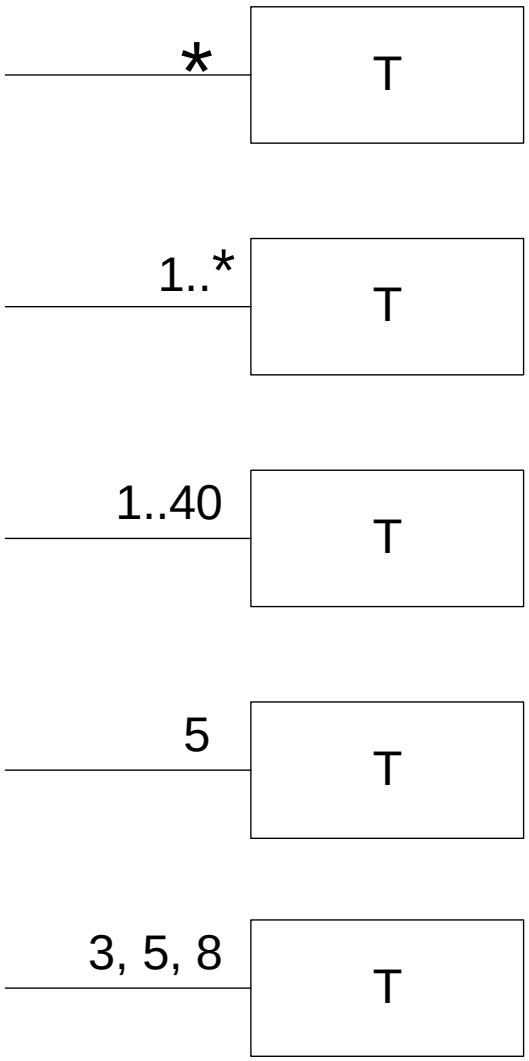
How to name an association?

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

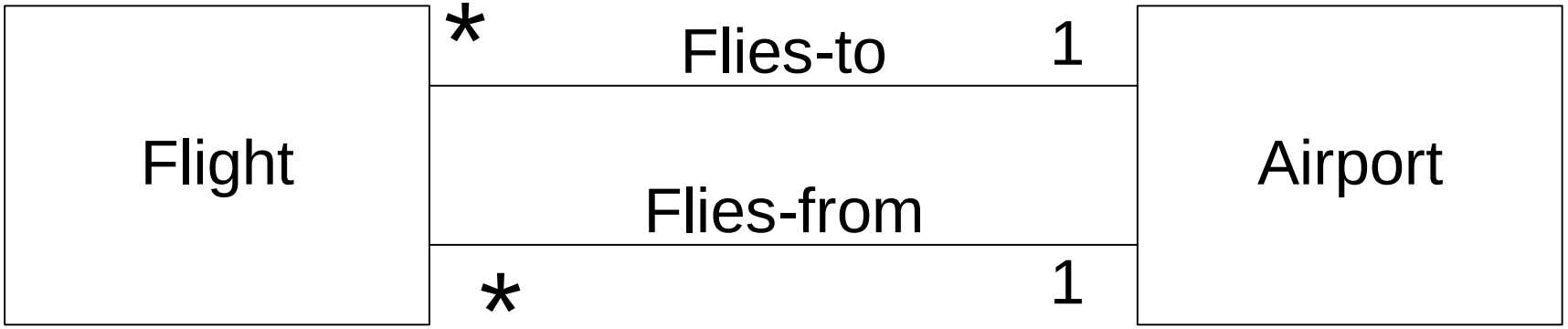
Association Multiplicities



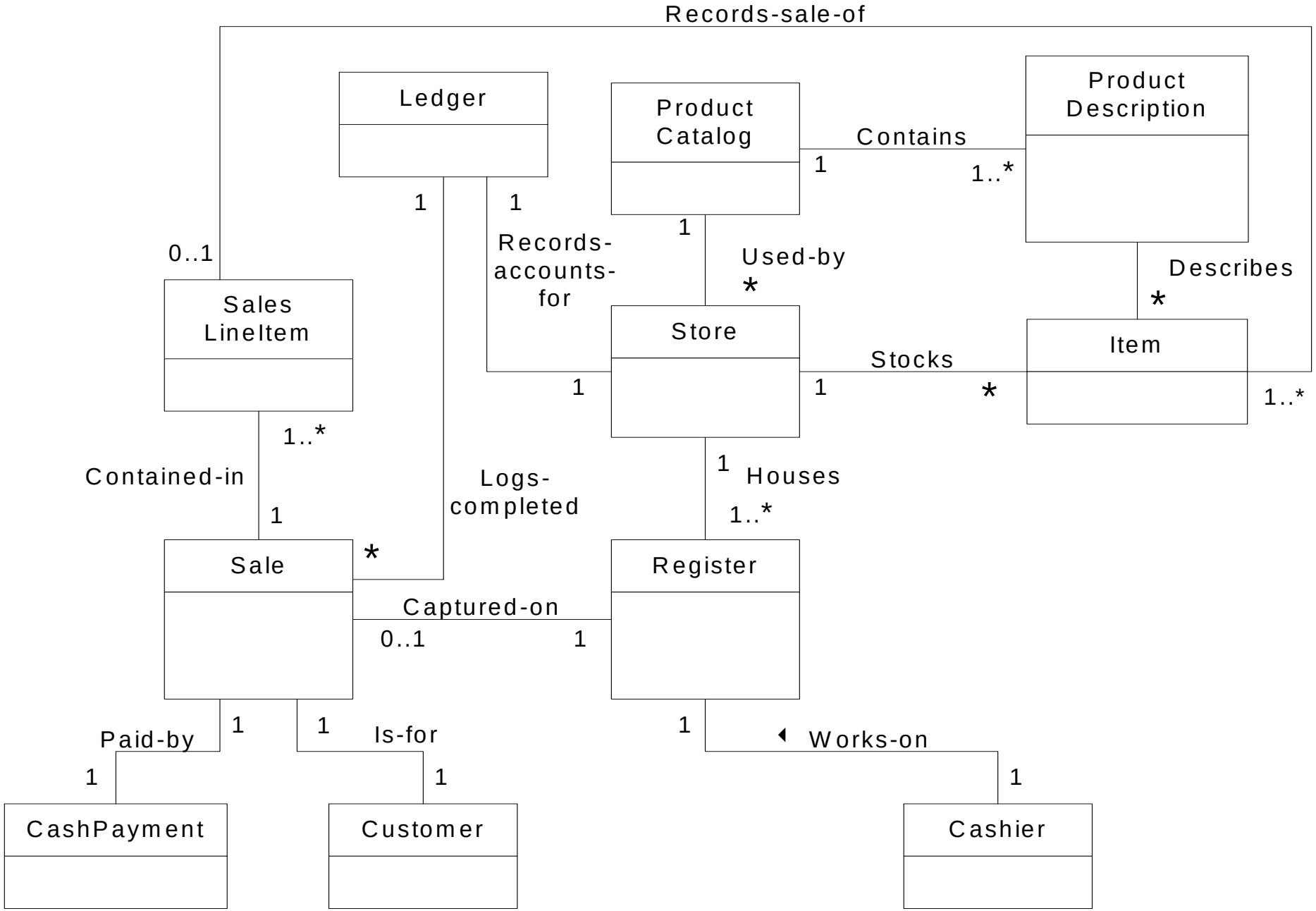
Multiplicities



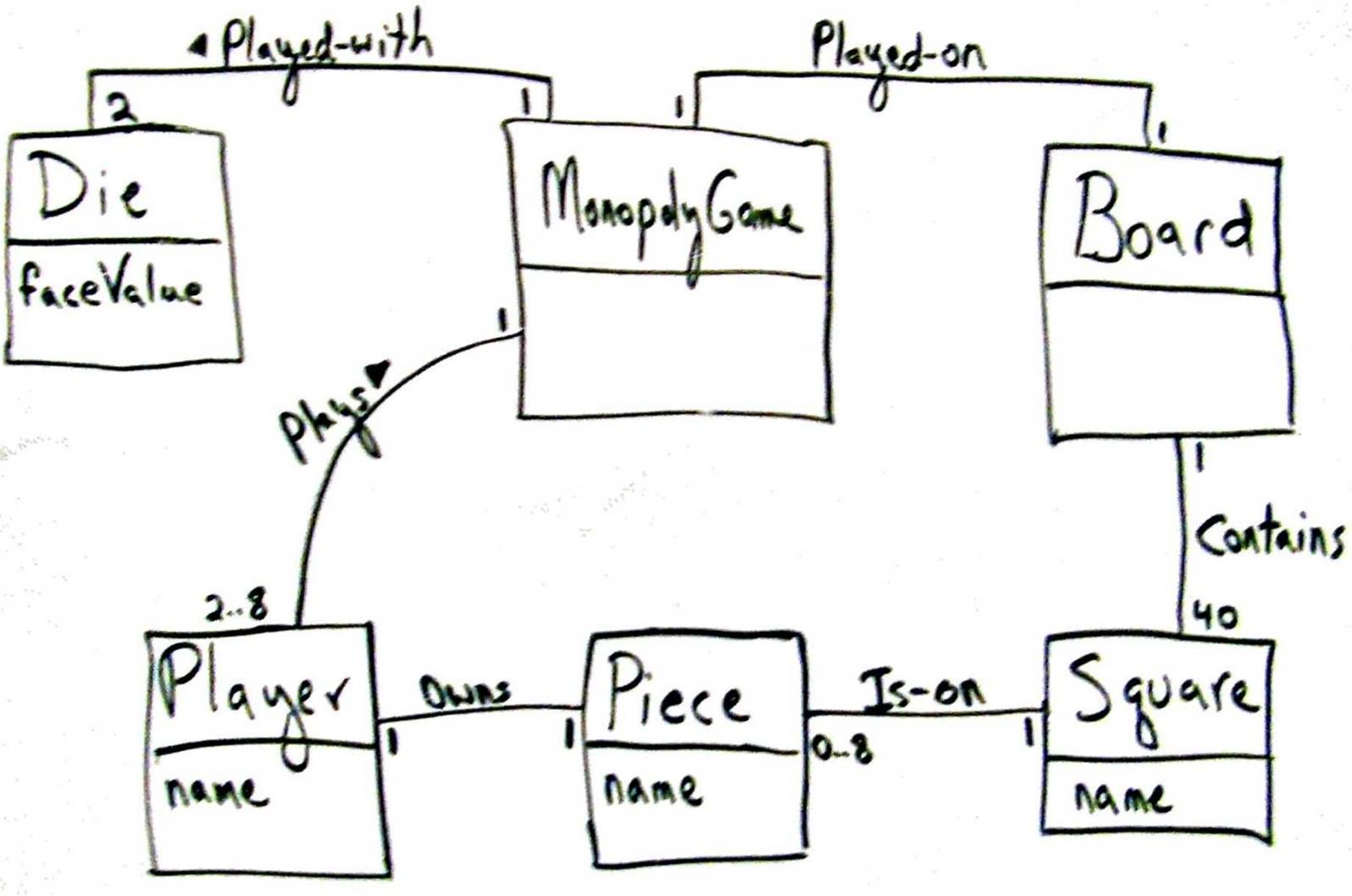
Multiple Associations



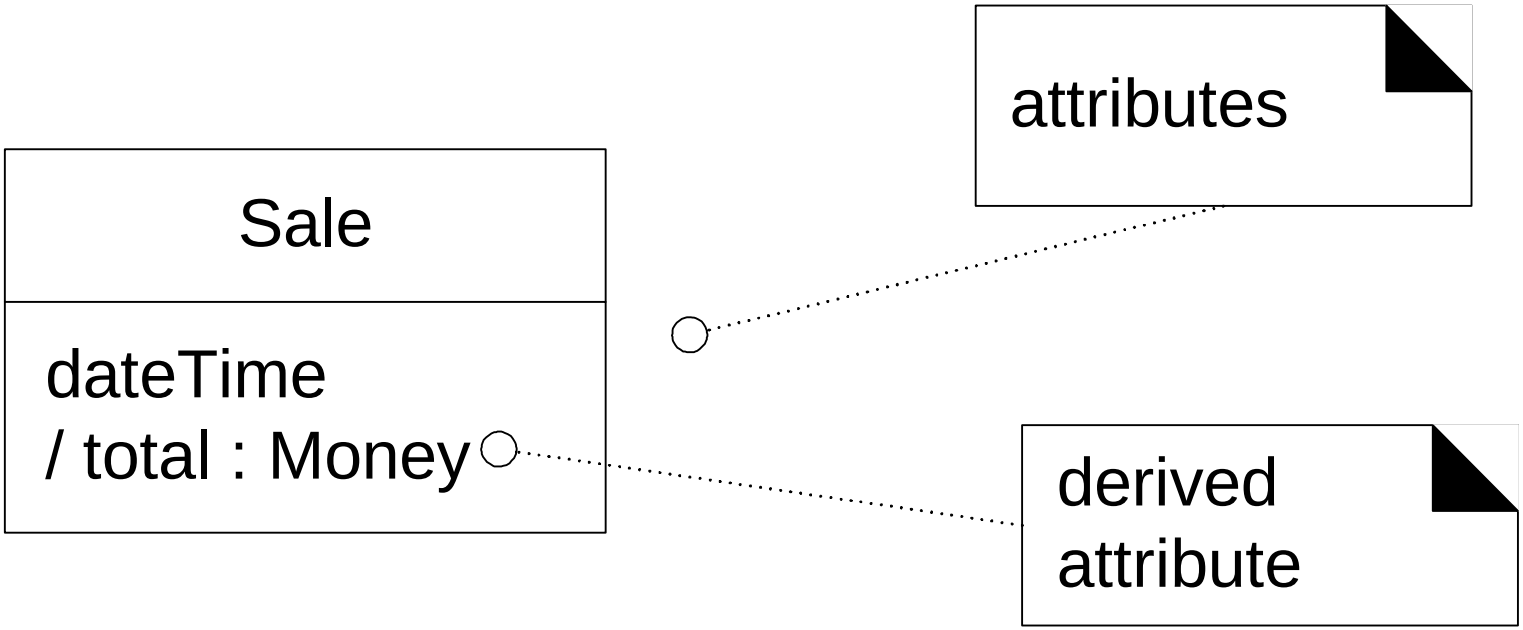
POS Domain Model Example:



Example: Domain Model for the Monopoly Game



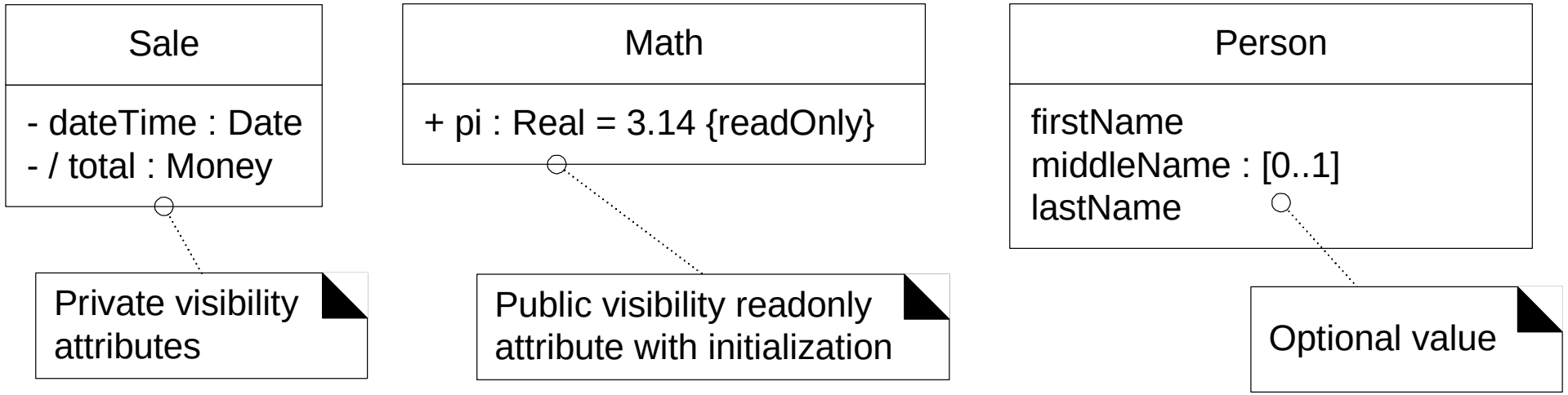
Attributes



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

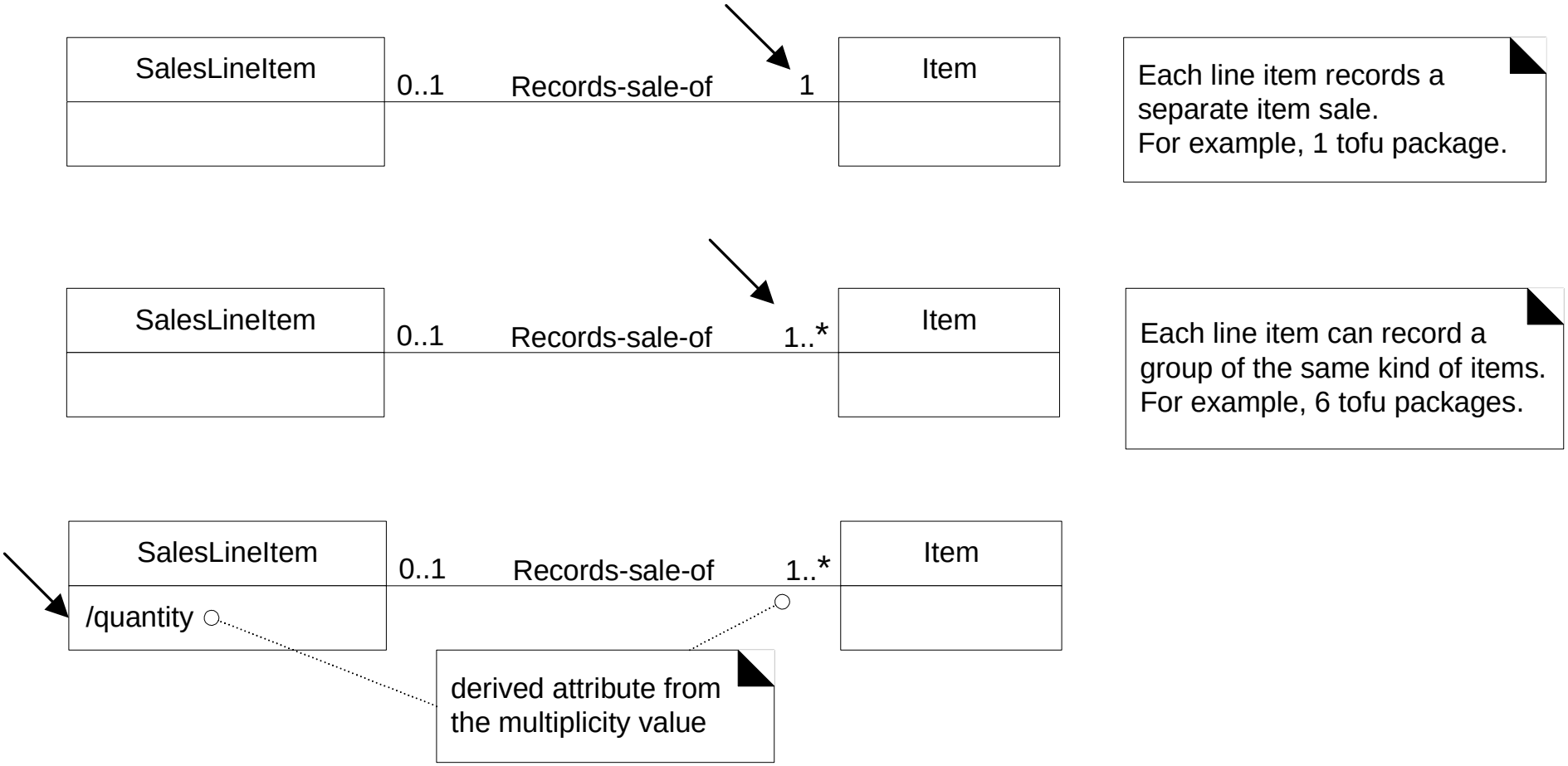
More detailed attribute notation

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



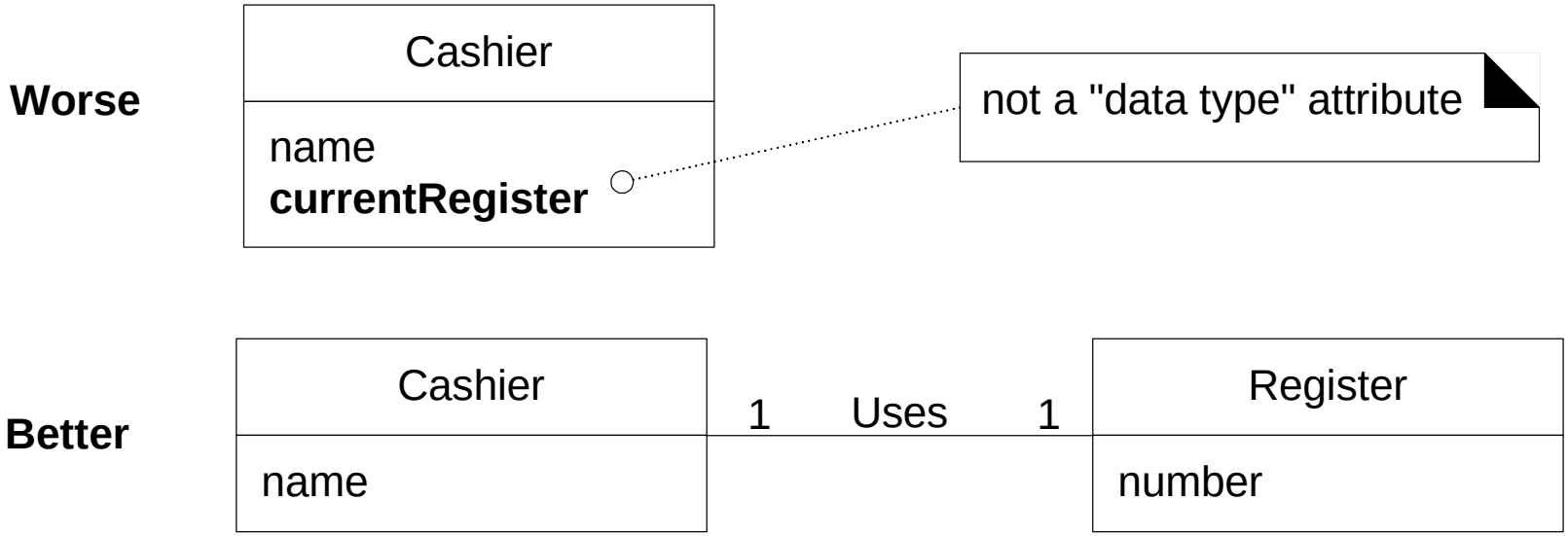
- Attribute requirements (such as an optional middle name) should also be placed in the Glossary.

Derivable attributes



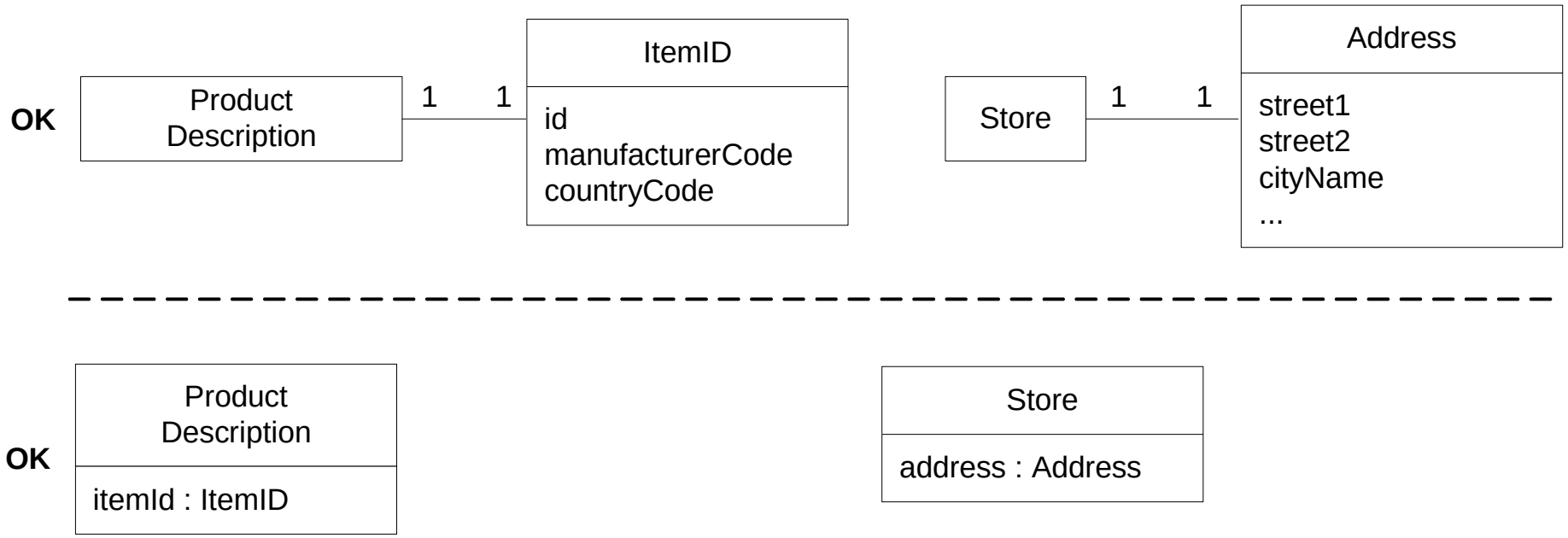
- The “/” sign: the value of this attribute can be calculated or derived from other attributes
- Still, it is noteworthy and may be recorded separately

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.

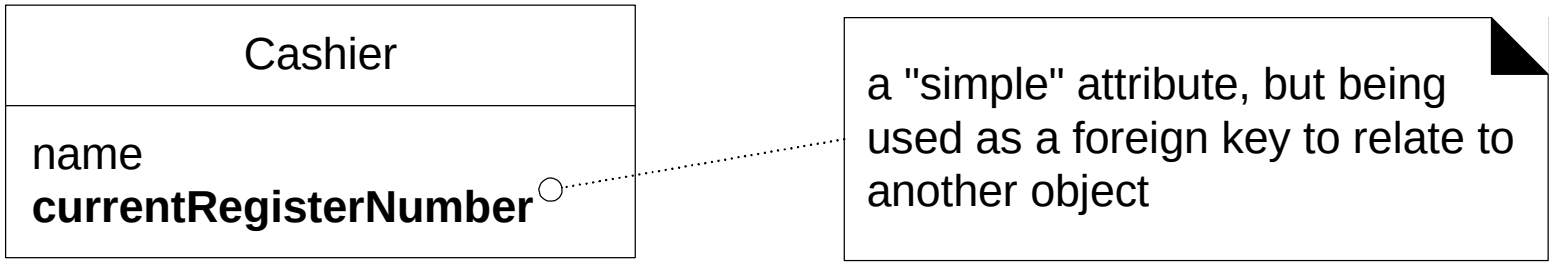
Do we need a new box for these classes?



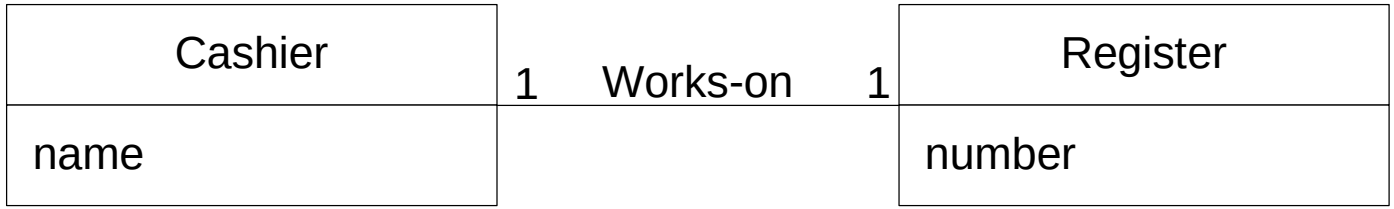
Attributes representing “foreign keys”

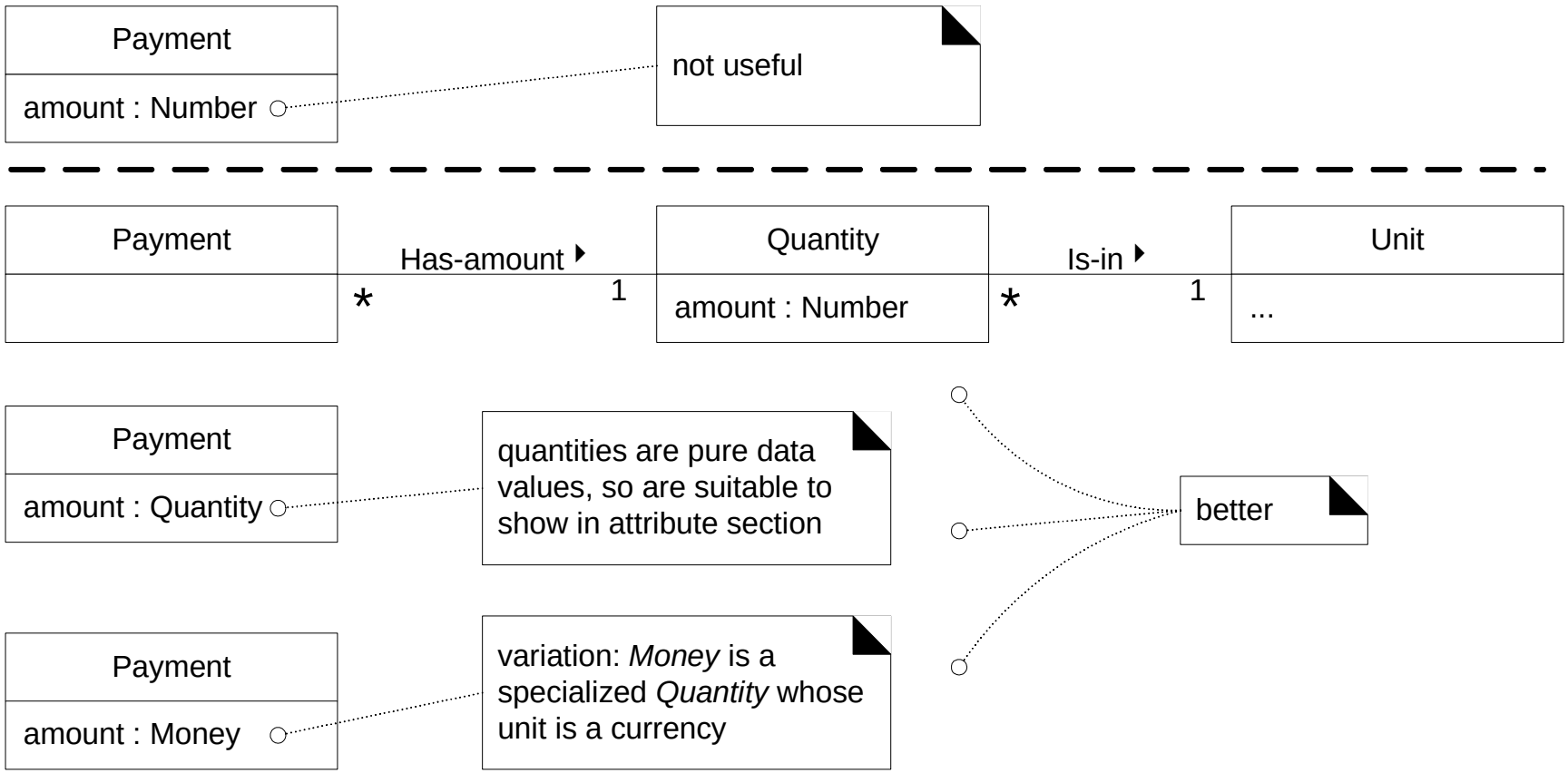
- Avoid them!

Worse

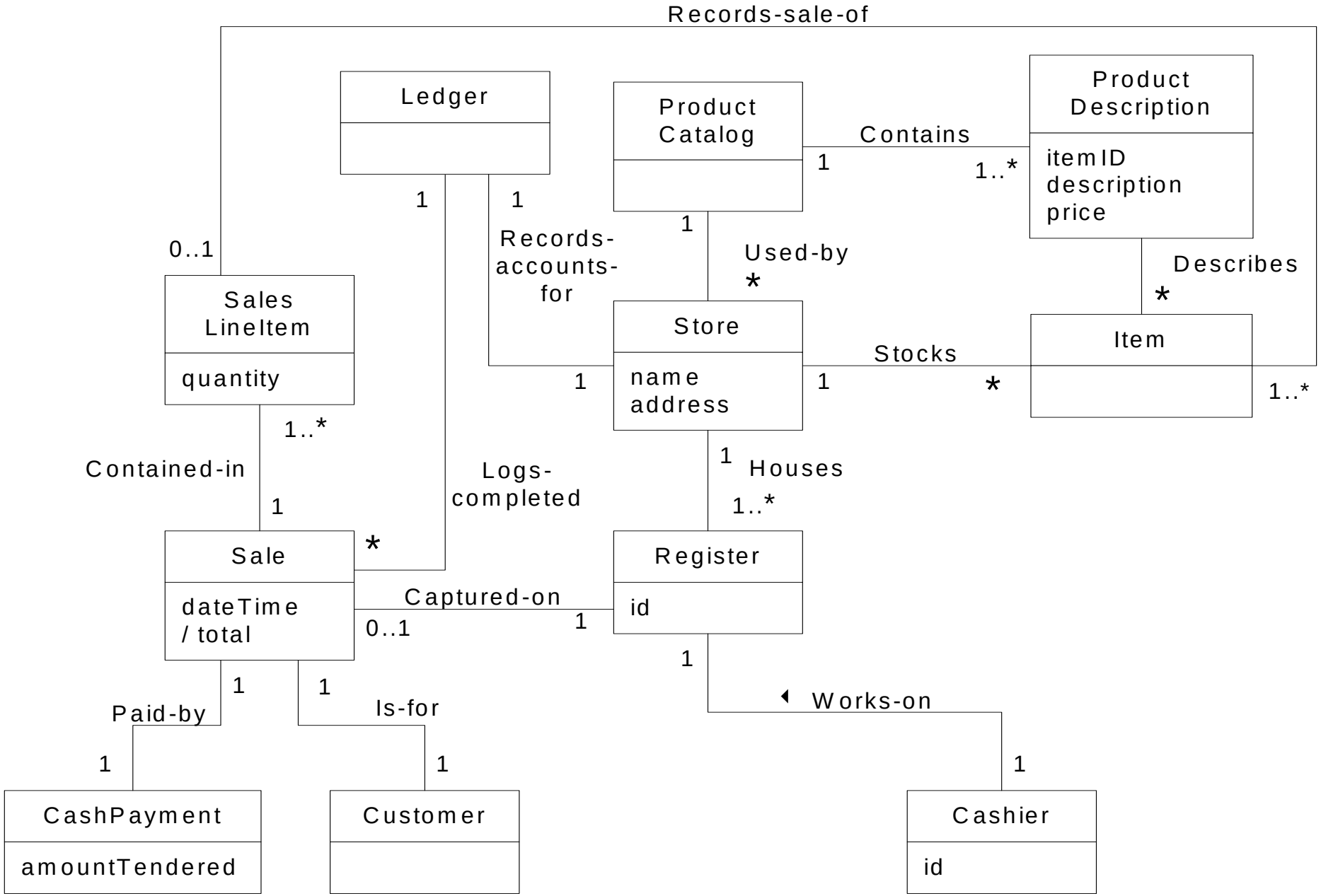


Better




Payment
amount : Number
not useful
Payment
Has-amount
Quantity
Is-in
Unit
*
1
amount : Number
*
1
...
Payment
amount : Quantity
quantities are pure data values, so are suitable to show in attribute section
better
Payment
amount : Money
variation: *Money* is a specialized *Quantity* whose unit is a currency

POS Domain Model Example:



POS Domain Model Example:

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

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

Monopoly Domain Model Example