Object Oriented Analysis & Design (OOAD)

Elaboration-Iteration 1

Chapter 8

Iteration 1, Elaboration:

- uses many OOA/D skills
- should be architecture-centric (big picture) and risk-driven
- early iterations don't do the whole Use Case.
- more Requirements Analysis to come; helped by feedback

NextGen POS 1st Iteration Requirements

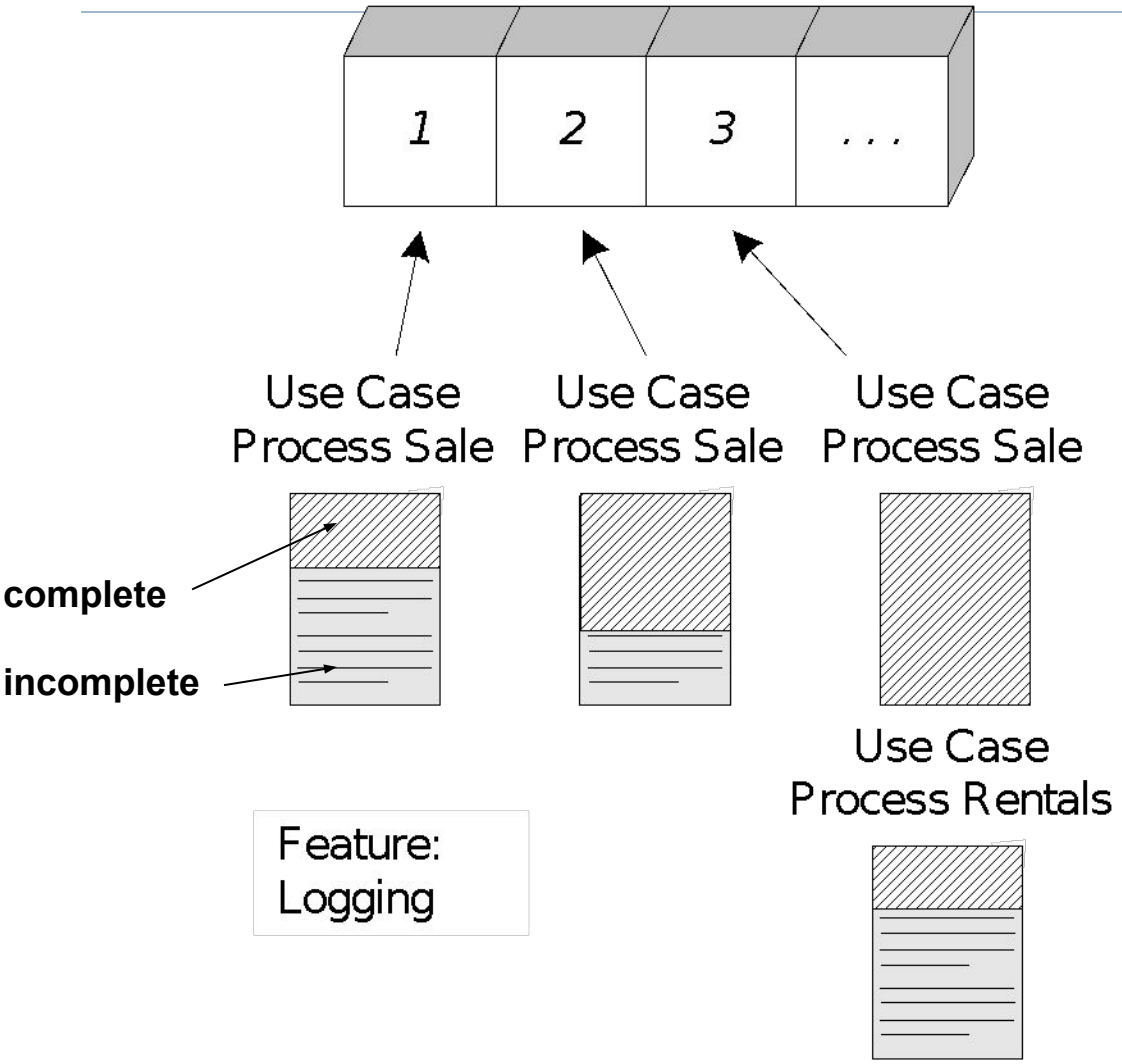
- Implement a basic, key scenario from the *Process Sale* use case: enter item, receive cash
- Implement *start up* use case
- Nothing fancy or complex, "happy" scenario only; not the whole Use Case
- No collaboration with external services like tax
- No complex pricing rules.

Subsequent iterations grow on this foundation

What we do in this Iteration:

- Iteration 1 requirements are subsets of complete requirements; e.g. cash sales only
- We build production-quality, tested code for a subset of requirements
- We start development without a complete requirements analysis

Fig. 8.1



A use case or feature is often too complex to complete in one short iteration.

Therefore, different parts or scenarios must be allocated to different iterations.

What we did in Inception…

- At *Inception* we used only a week and determined basic feasibility, risk and scope. Activities were
  - short requirements workshop
  - most actors, goals and use cases named
  - most use cases in brief format; 10% fully dressed
  - risks identified and listed
  - UI prototypes looked at
  - what to buy, what to build
  - plan for 1st iteration
  - tools list

What we will do in Elaboration…

- At *Elaboration* we move on to a series of iterations. Activities are:
  - core, risky software is coded and tested
  - majority of requirements are discovered
  - major risks mitigated
  - short time-boxed iterations
  - program early
  - adaptively design, code and test the core
  - test early
  - adapt based on feedback
  - write most of the use cases

NOTE: “Risk” includes business value

Planning the Next Iteration:

- Project planning is a big topic in standard software engineering courses. It is less important in this course.
- For us, we will try to organize ourselves around the three topics – risk, coverage and criticality
  - **risk**: complexity, unknown effort or usability
  - **coverage**: are all major parts at least touched on? Should be "wide and shallow".
  - **criticality**: things of high business value e.g.
- these issues revisited at the end of each iteration; perhaps they need adaptive thinking too

Domain Modeling

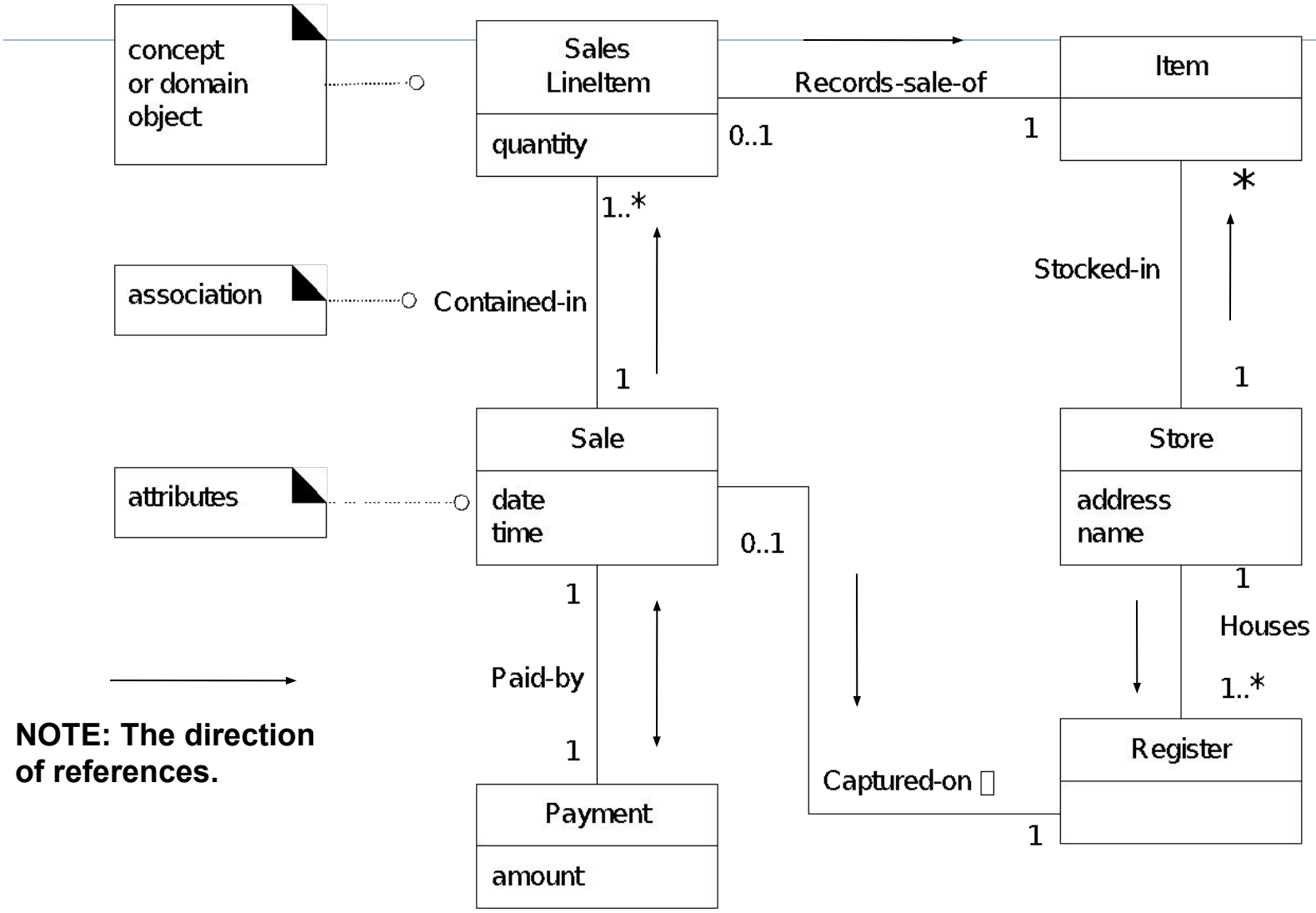
Domain Modeling

- After you have your **requirements** you start modeling the domain.
- You are still modeling the business, not the program classes.
- Various influences:
  - Source of info is the use-cases
  - operational contracts (like pre- and post-conditions)
  - conceptual classes give us ideas for what to name our program classes

Example:

- The following (next slide) are the conceptual classes for Payment and Sale.
- Part of the *Conceptual Perspective*
- Don't be too thorough; avoid *waterfall.*

Fig. 9.2


concept
or domain
object
association
attributes
Sales
LineItem
quantity
Records-sale-of
0..1
1
Item
*
1..*
Contained-in
Stocked-in
1
1
Sale
date
time
Store
address
name
0..1
1
1
Houses
1..*
Paid-by
1
1
Payment
amount
Captured-on
Register
1
NOTE: The direction
of references.

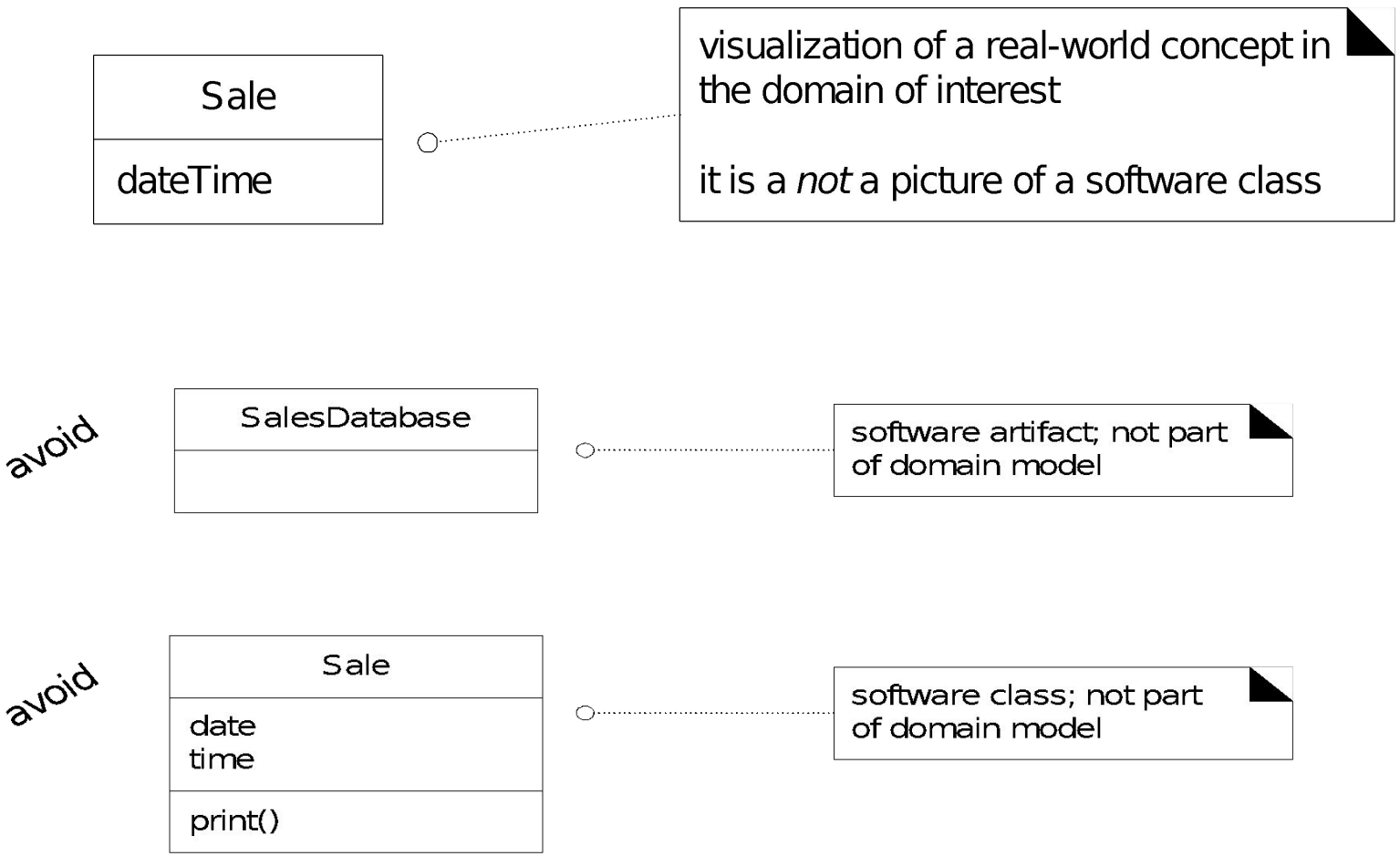
Domain Model (DM)

- Decompose the domain into noteworthy **concepts** and/or **objects**.
- The DM is a **visual** representation of real-situation objects and conceptual classes.
- Sometimes called Conceptual Model.
- In UP, this is one of the Business Modeling artifacts.
- Specifically, a class diagram with no operations or behavior.
- DM Shows:
  - **Domain objects** e.g. Sale
  - **Associations** e.g. PaidBy
  - **Attributes** e.g. date, time etc

What not to include:

- Software artifacts like salesDatabase, databases, windows
- Responsibilities (i.e, methods)

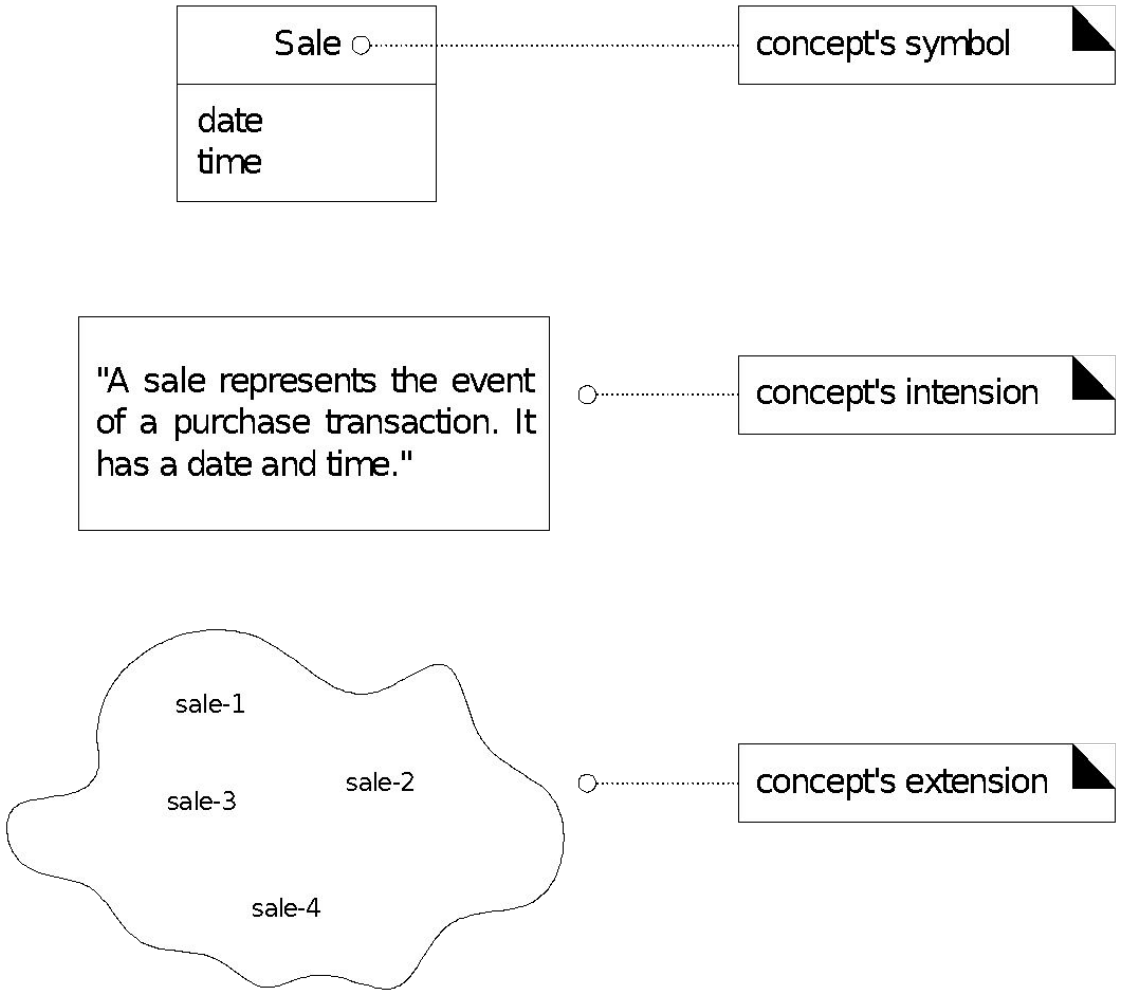
Fig. 9.3


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

Where does it fit?

- The Domain Model feeds into just one part of a program's design.
- Typical program uses the MVC (Model-View-Controller) paradigm; the DM feeds the Model part only.
- The author uses the term "domain layer" to mean the software components that underlie the UI layer.
- We consider a conceptual class from 3 points of view
  - *symbol*: what words we use to name it e.g. Sale
  - *intension*: its complete definition
  - *extension*: a set of examples of the class i.e. objects of Sale

Fig. 9.5


Sale
date
time
concept's symbol
"A sale represents the event of a purchase transaction. It has a date and time."
concept's intension
sale-1
sale-2
sale-3
sale-4
concept's extension

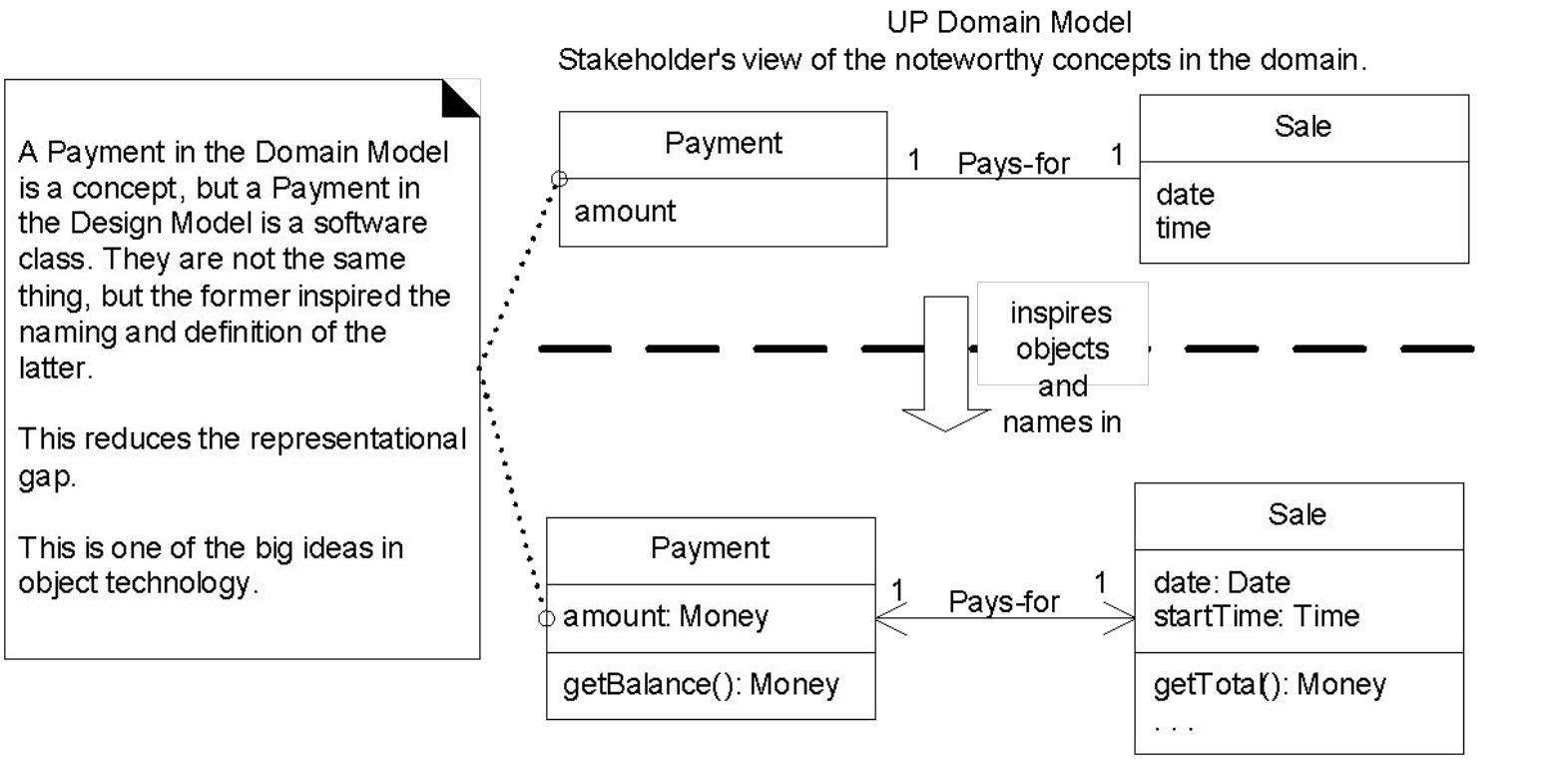
Are Domain and Data Models the same?

- A Data Model is a model of persistent data; i.e. data in a database or file system.
- Unlike a Data Model, just because we may never build the class, doesn't mean we don't put it in a Domain Model.
- Some Domain Model classes are purely behavioural

Why Build a Domain Model?

- The developer has to understand the business so that is a good reason to build the Domain Model
- We want to use software artifact names that "remind" us of what real-world things they model. So it is a good idea to list those domain concepts (visually).

Fig. 9.6



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 Identify Classes (Guidelines)?

- Limit yourself to the interests of the current iteration; don't try to draw the entire Domain Model up front (waterfall).

1. **To decide what classes to use, look around at what other developers have done here or in this area of endeavour (We will see this when we discuss software design patterns).**
2. **Make Lists**
3. **Linguistic Analysis**

List of Categories and possible classes

Conceptual Class Category	Examples
Business Transactions	Sale, Payment, Reservation
Transaction Line Items	SalesLineItem
Products or Services	Item, Flight, Seat, Meal
Where Transaction Recorded	Register, Ledger, Manifest
People's Roles	Cashier, Customer, Passenger
Place of Transaction	Store, Airport, Plane, Seat
Noteworthy Events	Sale, Payment, Flight
Physical Objects	Item, Register, Board, Airplane
Descriptions of Things	ProductDescription, FlightDescription

List of Categories and possible classes

Conceptual Class Category	Examples
Catalogs	ProductCatalog, FlightCatalog
Containers for Things	Store, Bin, Airplane
Things in Containers	Item, Square (on a Board), Passenger
Other Systems	CreditAuthSystem, AirTrafficControl
Records of Finance, Work	Receipt, Ledger, MaintenanceLog
Financial Instruments	Cash, Check, LineOfCredit,TicketCredit
Schedules, Manuals, Docs	DailyPriceChangeList, RepairSchedule

Linguistic Analysis:

- **Identify the nouns and verbs in text descriptions like a text-based Use-Case; consider these as candidate conceptual classes.**

Fully Dressed Use Case (1):

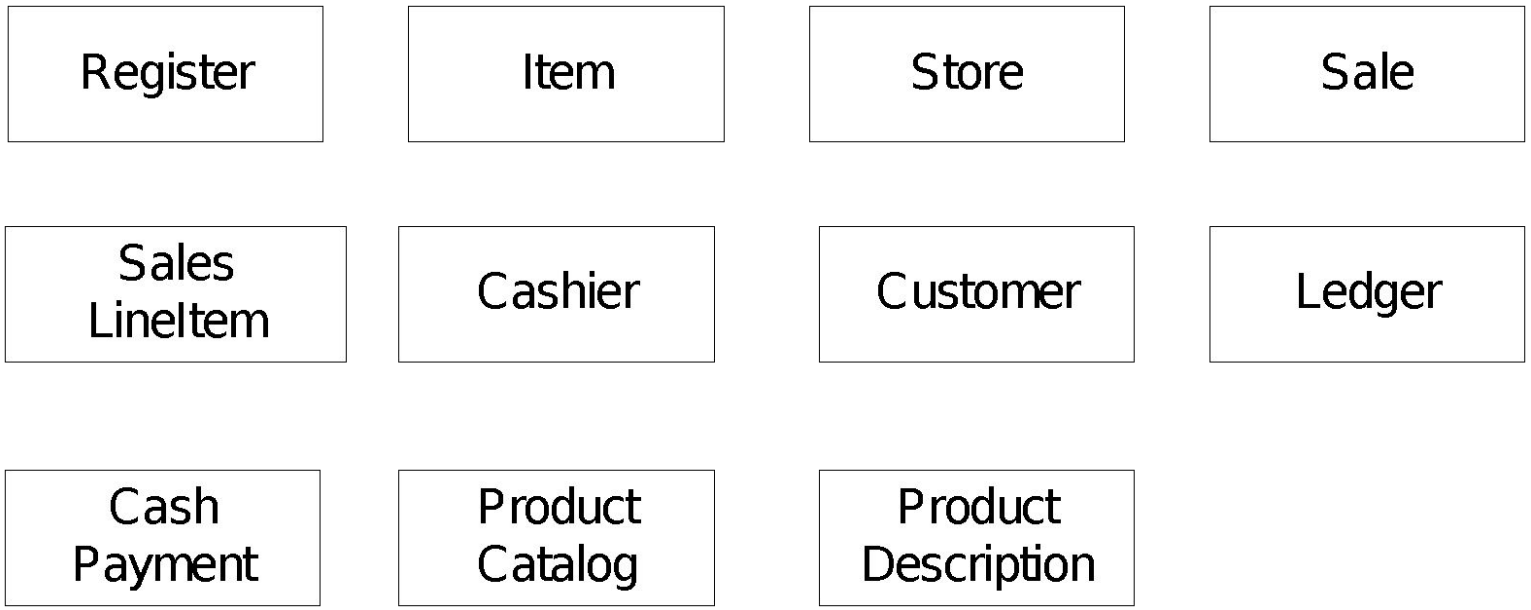
- *Scope*: NextGenPOSApplication
- *Level*: User Goal
- *PrimaryActor*: **Cashier**
- *StakeholdersAndInterests*:
  - **cashier** wants: …
  - **customer** wants: …
  - **company** wants: …
  - **manager** wants: …
  - **government** wants: …
  - **CC company** wants: …

Fully Dressed Use Case (2):

- *Preconditions*: **Cashier identified** and **authenticated**
- *Postcondtions*: **Sale** completed, **tax calculated**, **inventory updated**, **commission recorded**, **cc approval** recorded
- *MainSuccessScenario*:
  1. Customer arrives with **goods**
  2. Cashier starts new sale
  3. Cashier enters **item identifier**
  4. Systems **records sale** and presents description and running total

     (repeat 3-4 many times)
  1. System presents total
  2. …

Identified Conceptual Classes



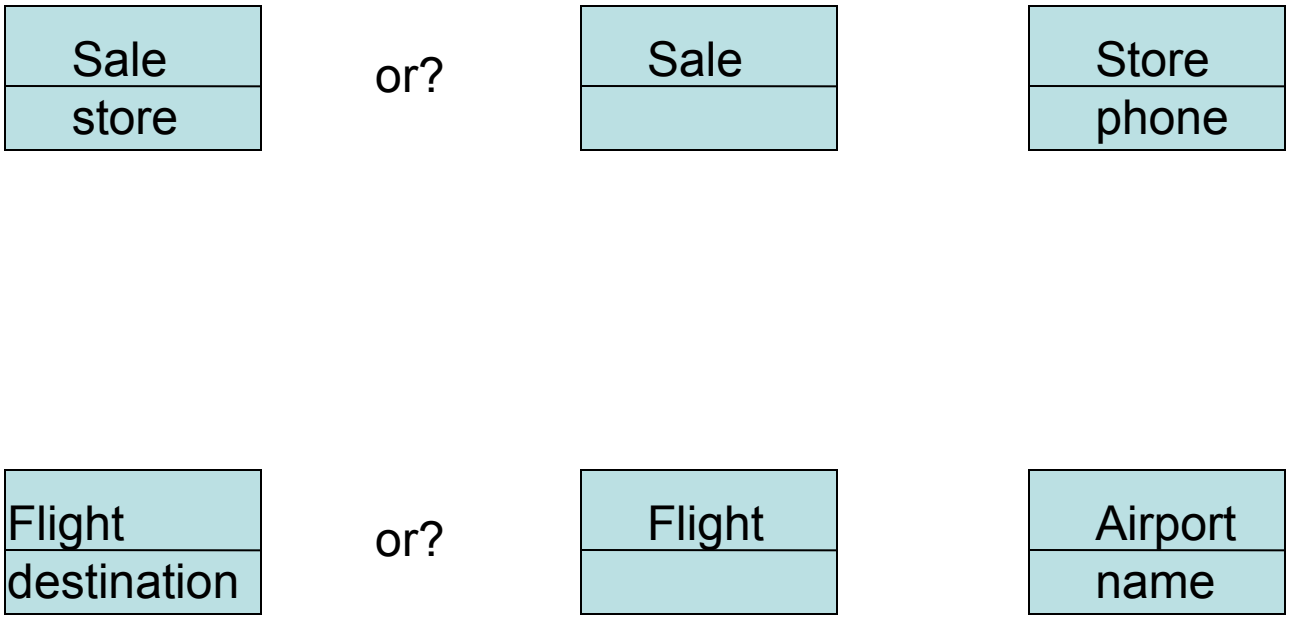
What about Using a Tool?

- There is little motivation to update a model unless it is easy to do so.
- That is why the author likes the whiteboard.
- The practical reason for updating is to keep the model as a documentation tool (for version 2).

Attributes vs Classes:

- A common mistake is to cross identify these things.
- **Rule of Thumb 1**: If you don't think of something as a number or string in the real world, it probably is a class.
- **Rule of Thumb 2**: Classes are capable of independent existence; attributes are properties of things.

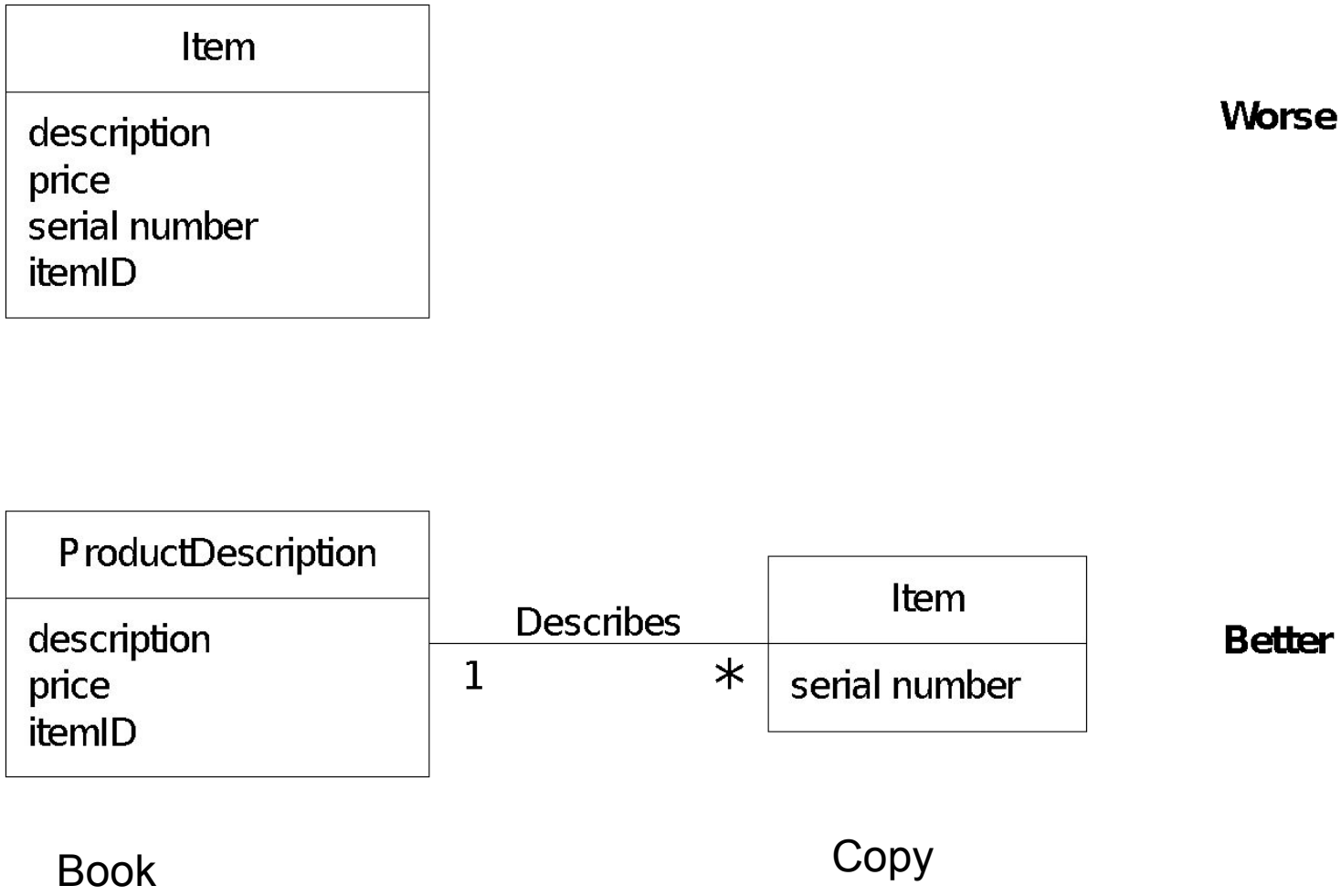
Examples:


Sale
store
or?
Sale
Store
phone
Flight
destination
or?
Flight
Airport
name

When to Model Descriptions?

- A **description class** describes something else.
- The difference between an object and a description of it is like the difference between a copy of a book and a book itself. If you run out of copies you may still need to know about the book.
- Intension vs Extension
- Physical Object vs Type of Thing
- So we need an *Item* class and a *ProductDescription*.
- How do you tell them apart? Identify their keys or identifying attributes.

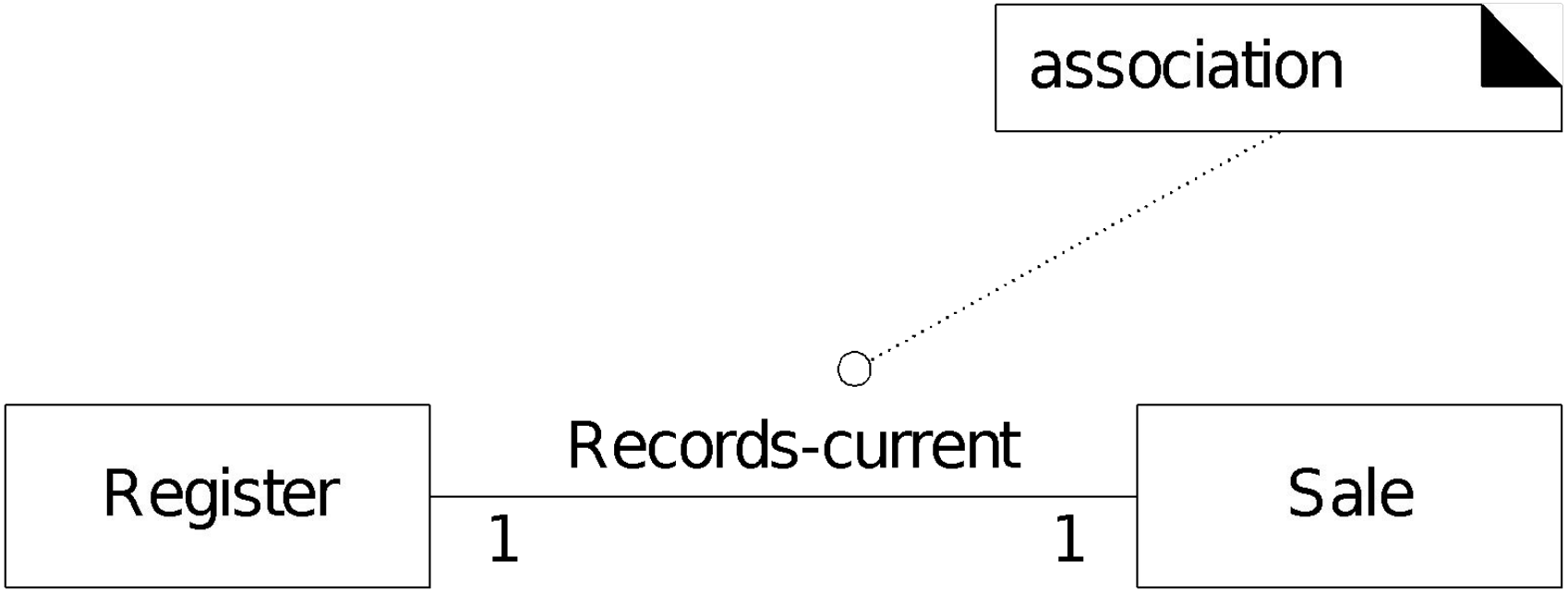
Fig. 9.9


Item
description
price
serial number
itemID
Worse
ProductDescription
description
price
itemID
Describes
1
*
Item
serial number
Better
Book
Copy

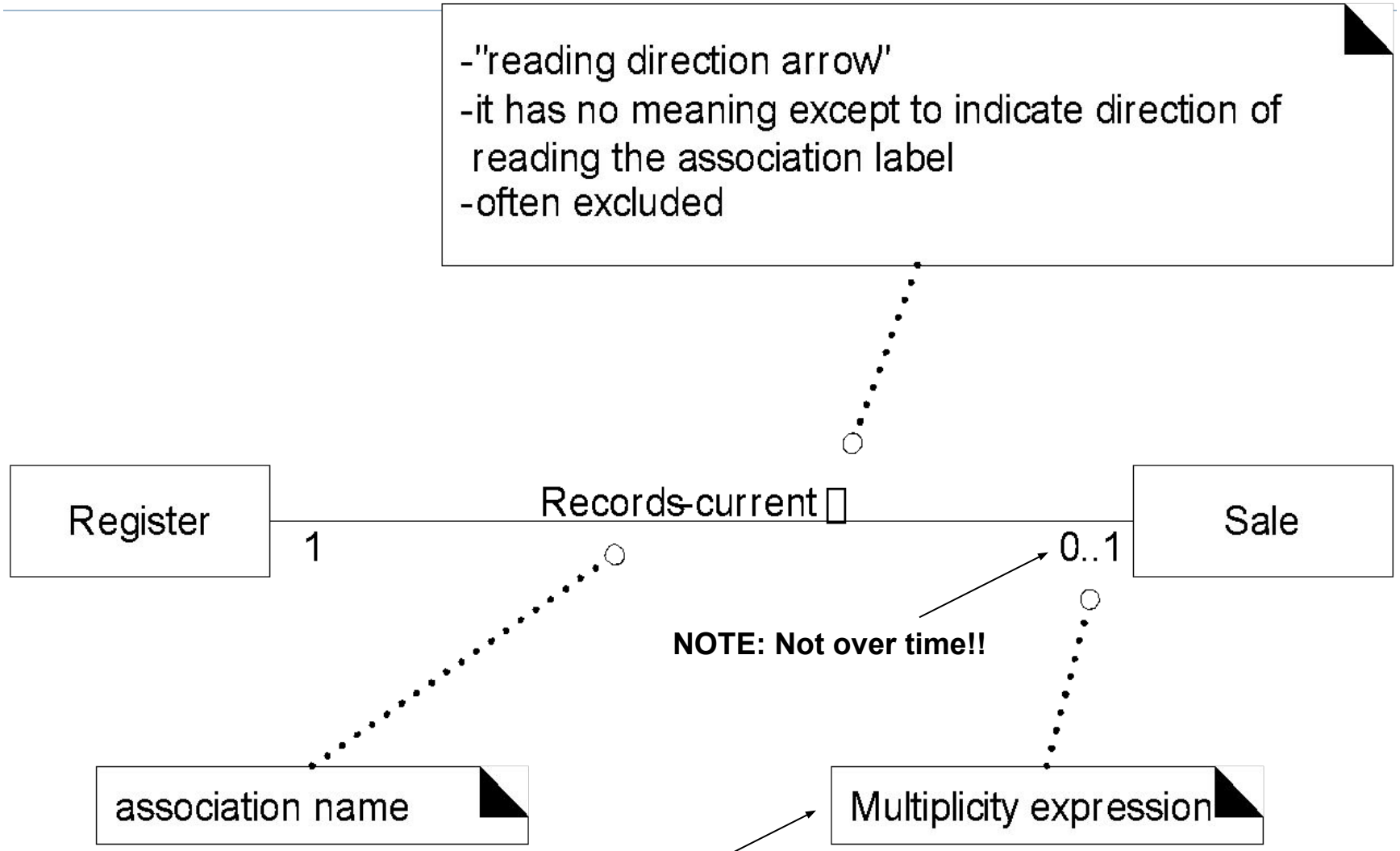
Associations:

- A relationship between classes (or instances of classes).
- When do we show them?
  - when we need (real-world) to remember the other to fulfill client requirements…
- Do we need to record what *SaleLineItem* instances are associated with a *Sale* instance….?
- Too many associations make for an unwieldy (difficult to handle) diagram
- What associations are "implemented"?
  - Not all
  - some new ones appear between software objects

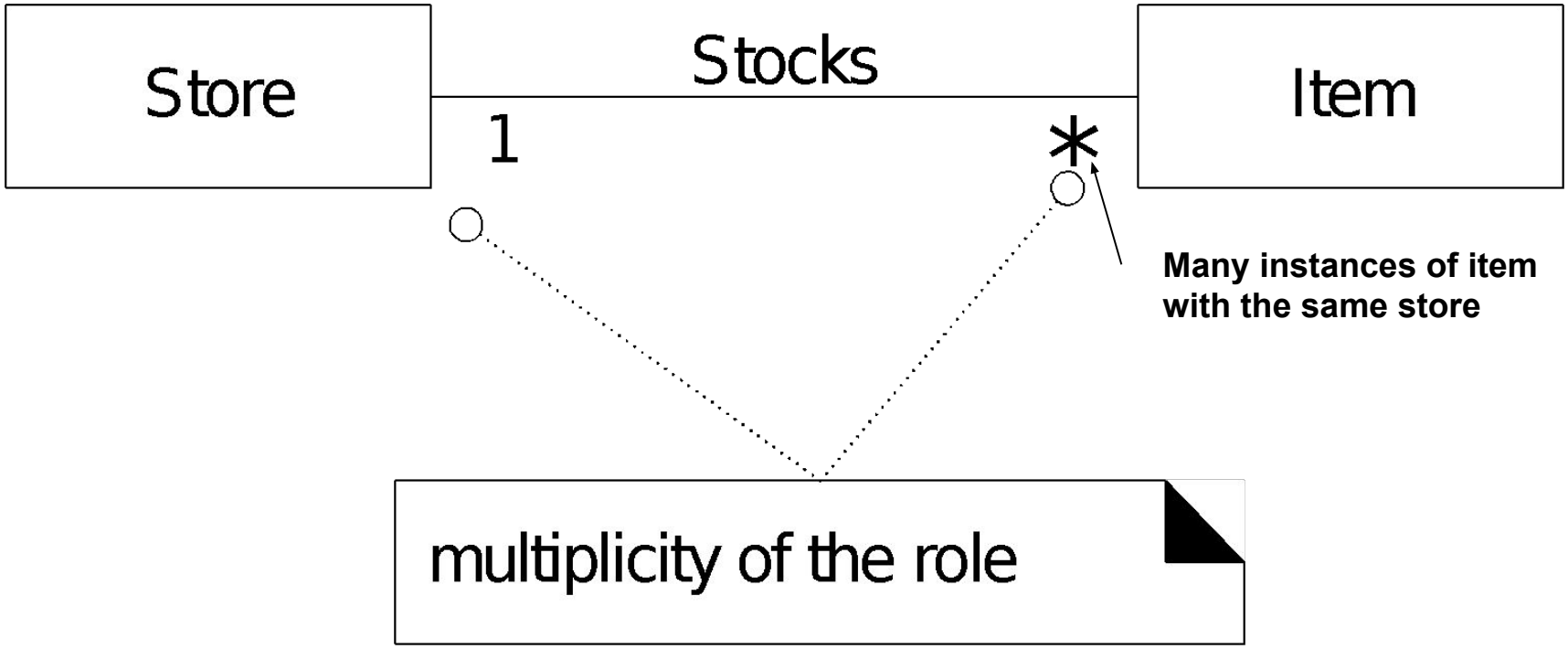
Association Example



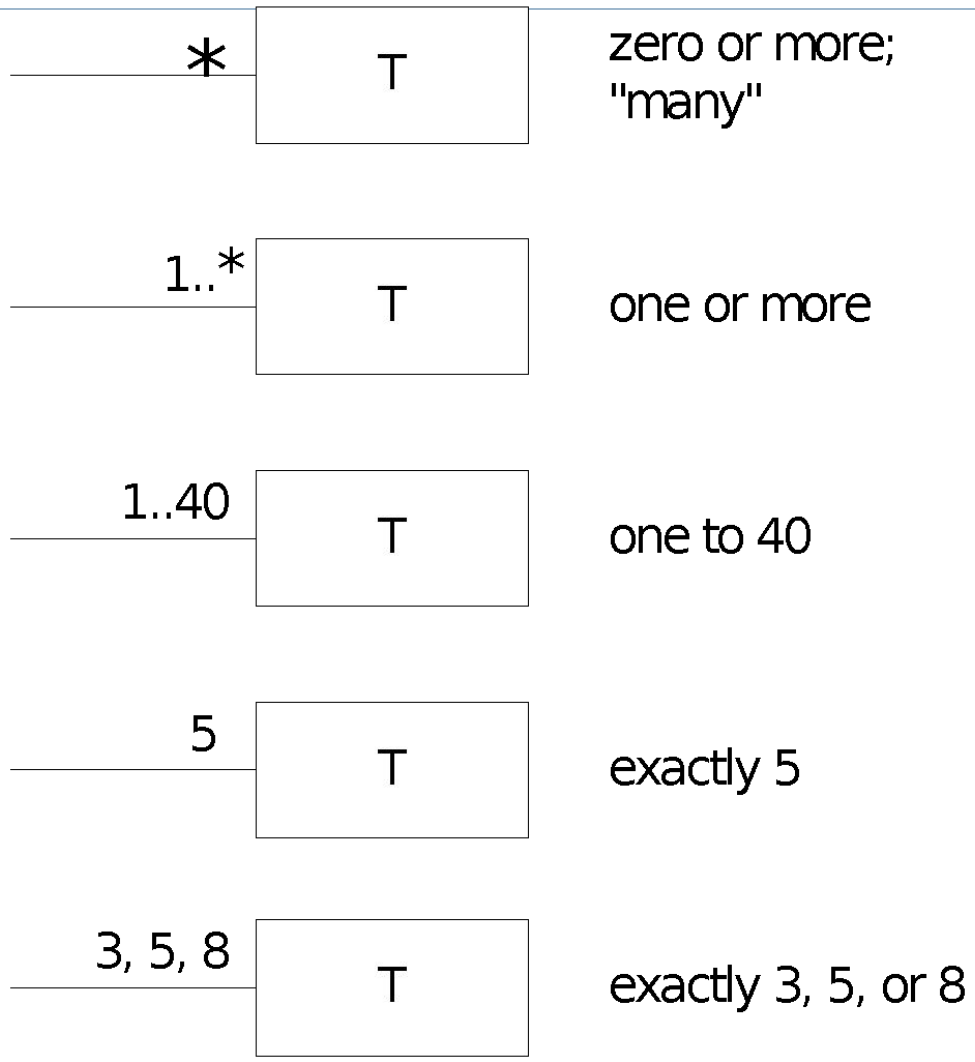
The UML Notation for Associations



UML Roles

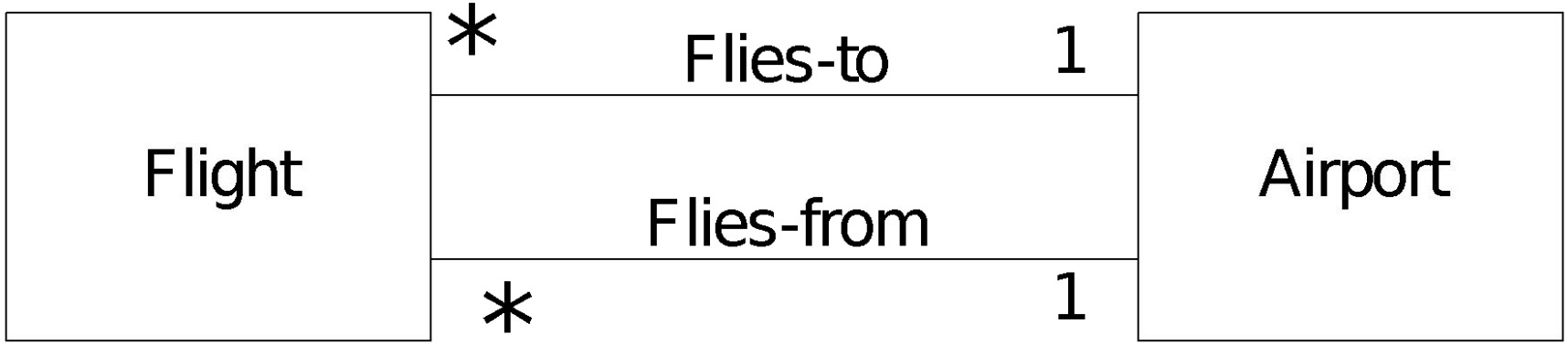

Store
Stocks
Item
1
*
Many instances of item with the same store
multiplicity of the role

Multiplicity Values



Multiplicity values reflecting domain constraints

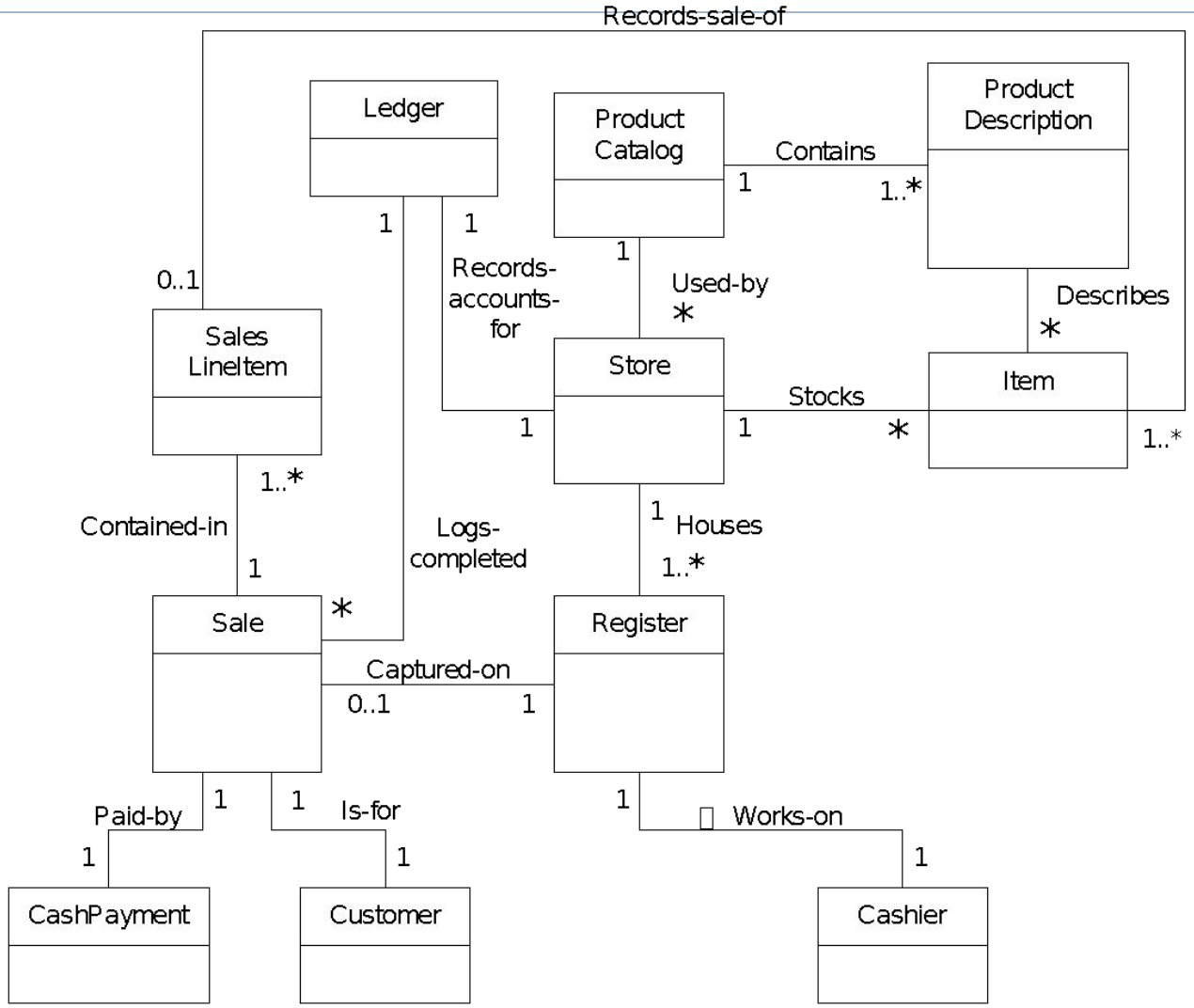
Multiple Associations



Where to Start Adding Associations:

A is a transaction related to another transaction B	**Payment - Sale**
A is a line item of a transaction B	**SalesLineItem - Sale**
A is a product or service for a transaction (or line item) B	**Item - SalesLineItem**
A is a role related to a transaction B	**Customer - Payment**
A is a physical or logical part of B	**Drawer - Register**
A is physically or logically contained in B	**Register - Store**
A is a description of B	**ProductDescription - Item**
A is known/logged/captured in B	**Sale - Register**
A is a member of B	**Cashier - Department**
A is an organizational subunit of B	**Department - Store**
A uses or manages or owns B	**Cashier - Register**
A is next to B	**SalesLineItem - SalesLineItem**

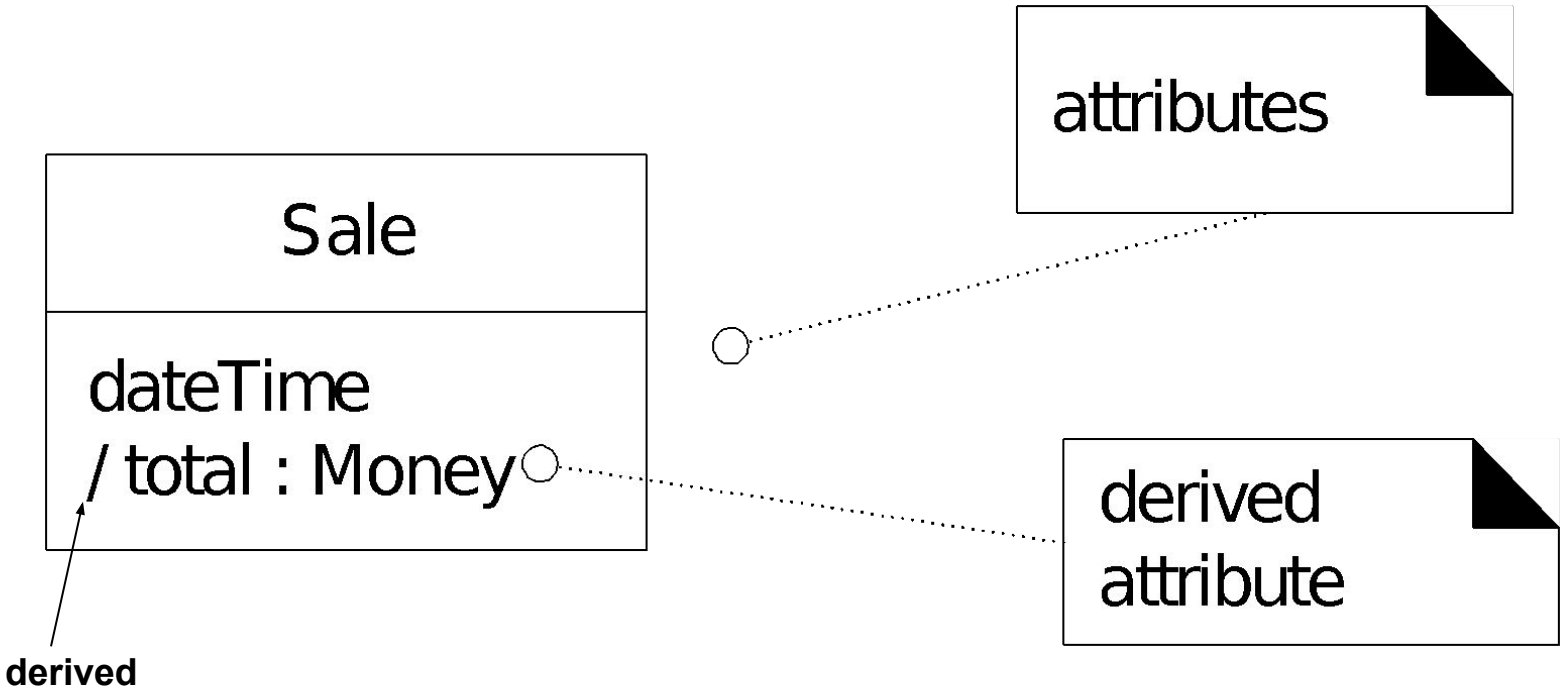
POS Partial Domain Model with Associations



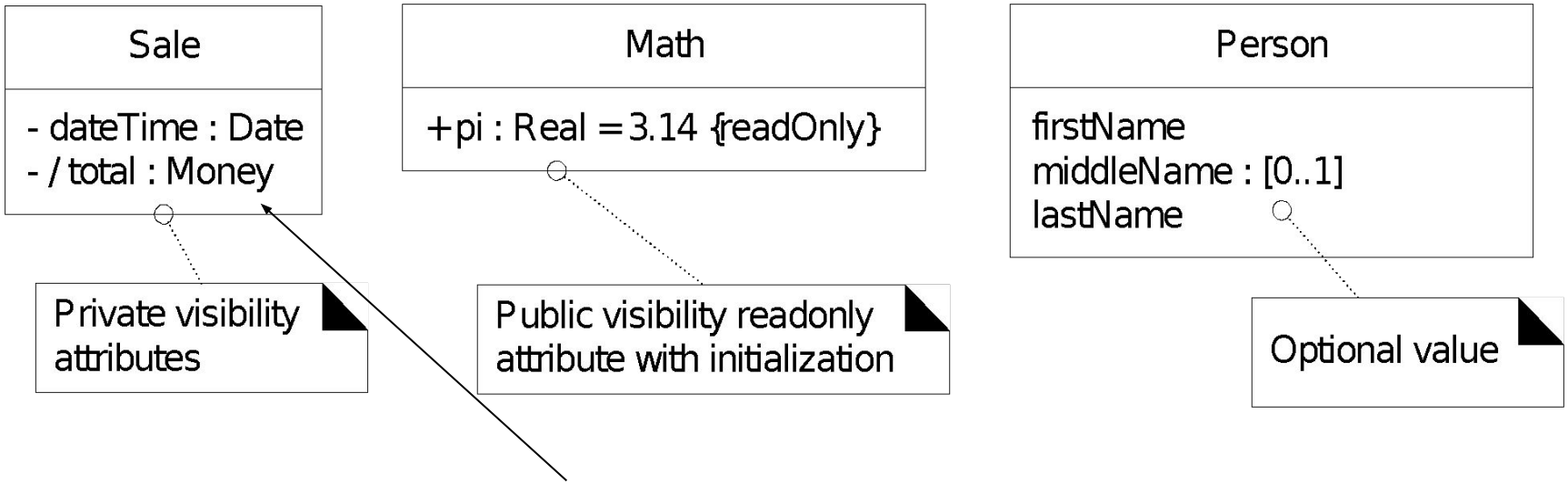
Attributes:

- An attribute is a logical data value of an object.
- Find them in the text analysis of the requirements
- Show in DM when requirements suggest a need to remember information e.g. a receipt (which prints info related to Sale) in the Process Sale use case need to include Date, Time, StoreName, Address among many other things.

Class & Attributes

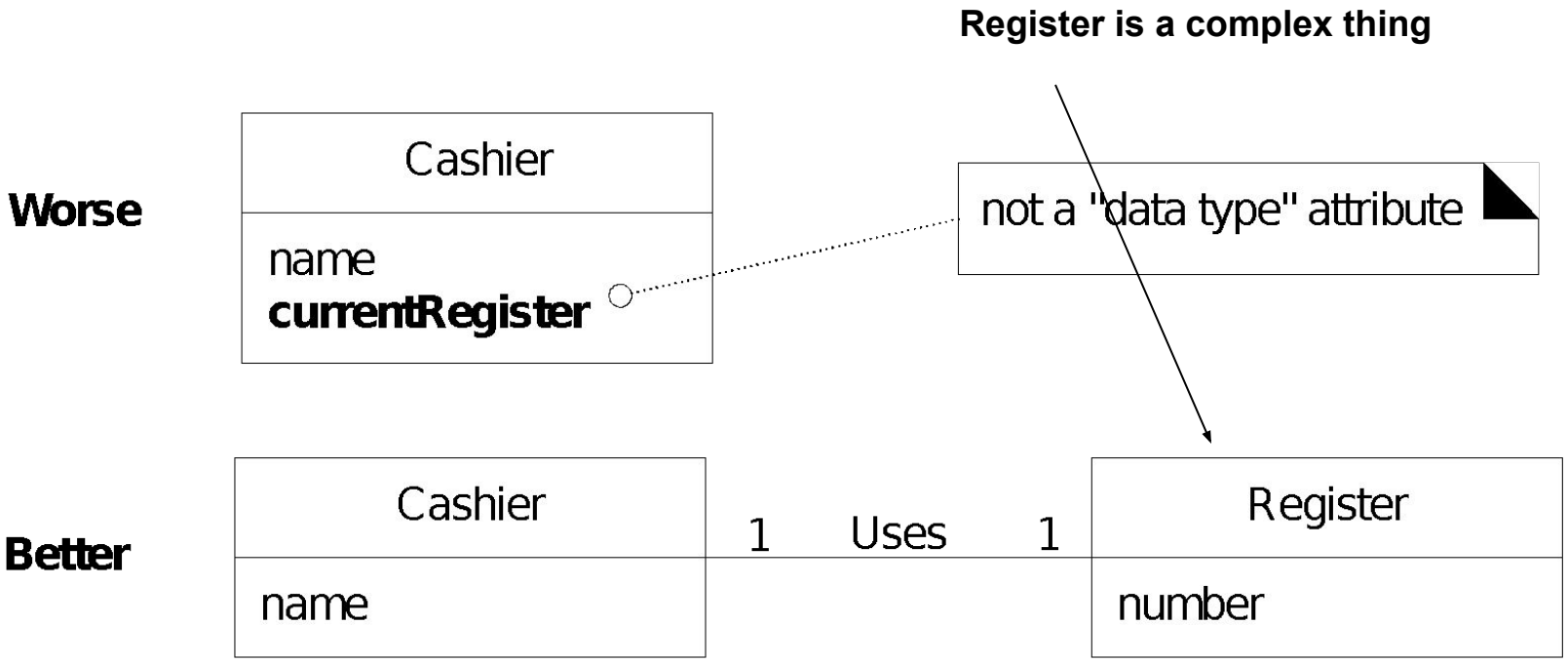


Attribute Notation in UML



NOTE: You can "invent" obvious data types on your own

Don’t Show Complex things as attributes



Contd…

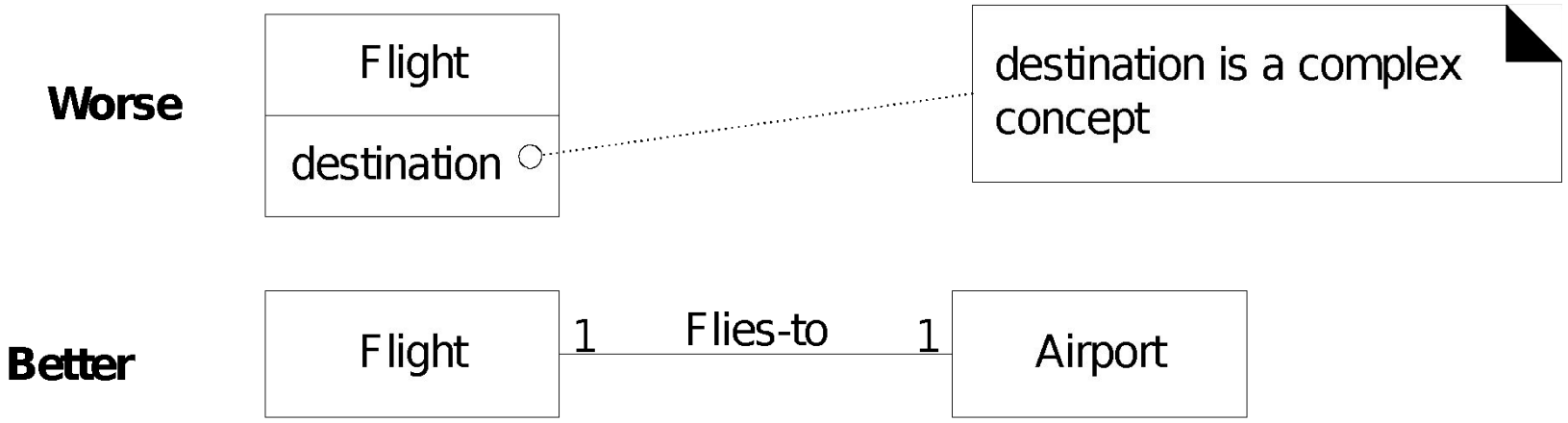

Worse
Flight
destination
destination is a complex concept
Better
Flight
1
Flies-to
1
Airport

Fig. 9.24

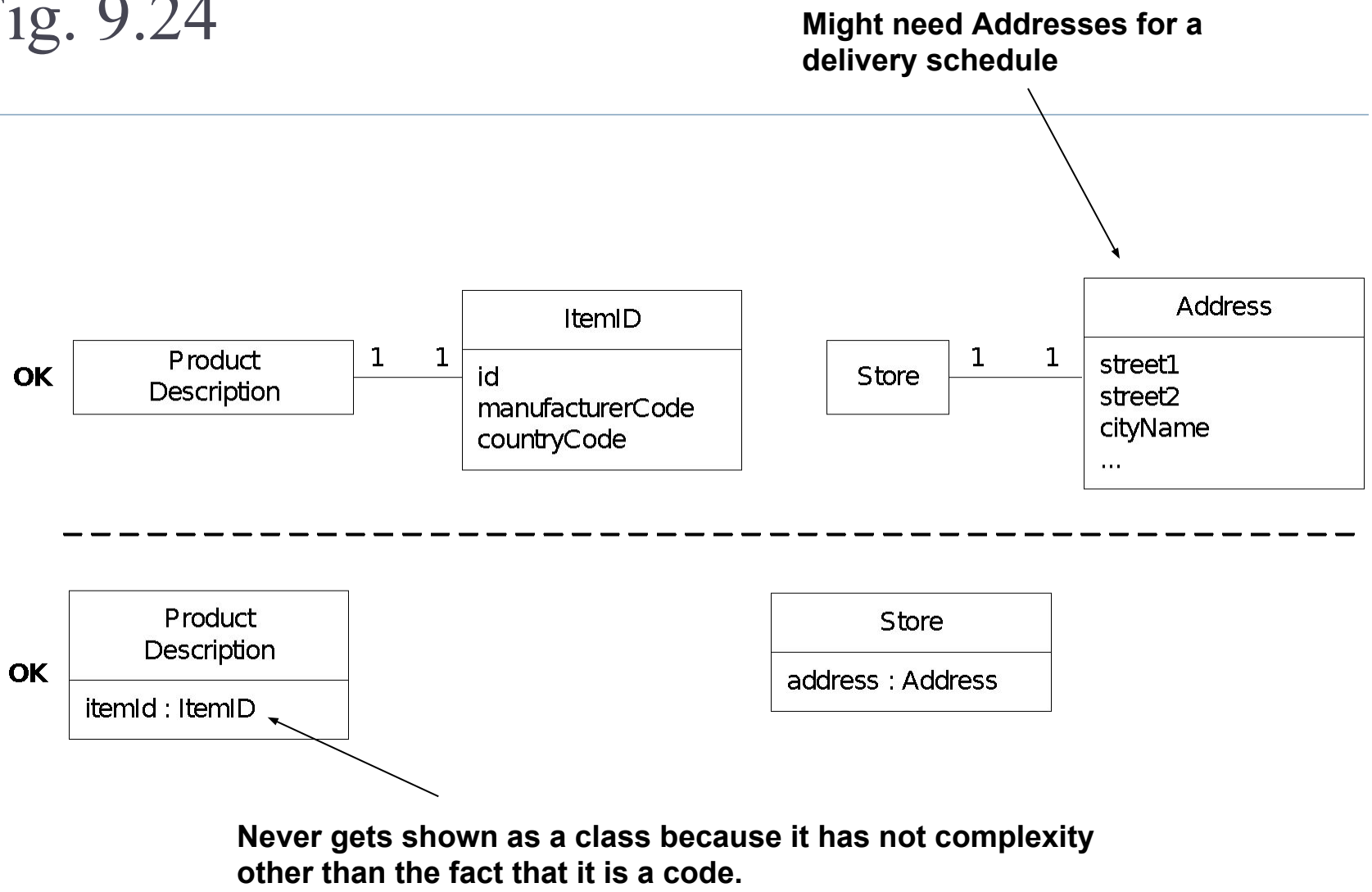
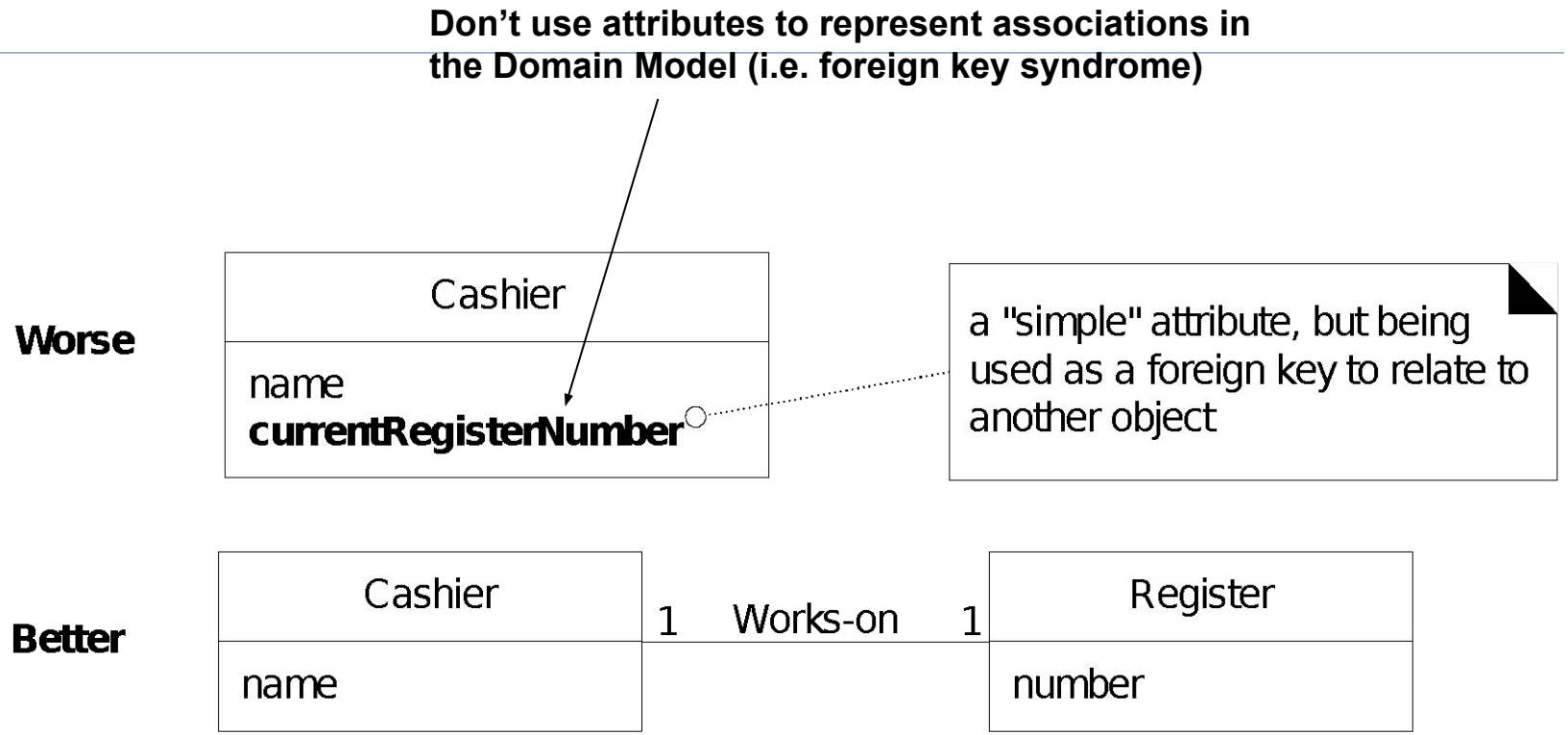

Might need Addresses for a delivery schedule
OK
Product Description
1
1
ItemID
id
manufacturerCode
countryCode
Store
1
1
Address
street1
street2
cityName
...
OK
Product Description
itemId : ItemID
Store
address : Address
Never gets shown as a class because it has not complexity other than the fact that it is a code.

Fig. 9.25


Don’t use attributes to represent associations in the Domain Model (i.e. foreign key syndrome)
Worse
Cashier
name
currentRegisterNumber
a "simple" attribute, but being used as a foreign key to relate to another object
Better
Cashier
name
1
Works-on
1
Register
number

Partial Domain Model with Attributes