Applying UML and Patterns

**An Introduction to
Object-oriented Analysis
and Design
and Iterative Development**

Part III Elaboration Iteration I – Basic[1]

Chapters

8. ***Iteration 1 – basics***
9. ***Domain models***
10. System sequence diagrams
11. Operation contracts
12. Requirements to design – iteratively
13. Logical architecture and UML package diagrams
14. On to object design
15. UML interaction diagrams
16. UML class diagrams
17. GRASP: design objects with responsibilities
18. Object design examples with GRASP
19. Design for visibility
20. Mapping design to code
21. Test-driven development and refactoring

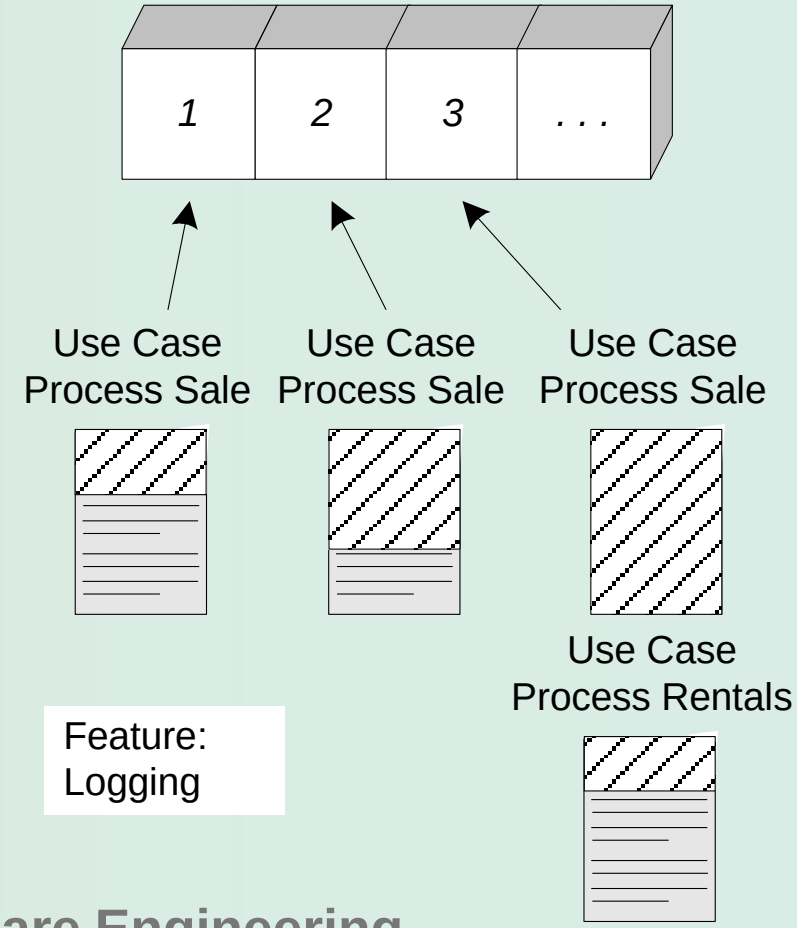
Chap 8 Iteration 1 Basics

Iteration 1

- Iteration 1 of the elaboration phase
  - Requirements and Emphasis: Core OOA/D Skills
  - Architecture-centric and risk-driven.
- In Iterative Development, Don't Implement All the Requirements at Once
- **Incremental Development for the Same Use Case Across Iterations**

Iteration 1

- Use case implementation may be spread across iterations



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.

POS Iteration 1

- Requirements for iteration 1 of the POS application
  - Implement a basic, key scenario of the Process Sale use case: entering items and receiving a cash payment.
  - Implement a Start Up use case as necessary to support the initialization needs of the iteration.
  - Nothing fancy or complex is handled, just a simple happy path scenario, and the design and implementation to support it.
  - There is no collaboration with external services, such as a tax calculator or product database.
  - No complex pricing rules are applied.
  - The design and implementation of the supporting UI, database, and so forth, would also be done

Elaboration 1

- **Elaboration:** Build the core architecture, resolve the high-risk elements, define most requirements, and estimate the overall schedule and resources.
- Elaboration is the initial series of iterations during project
  - the core, risky software architecture is programmed and tested
  - the majority of requirements are discovered and stabilized
  - the major risks are mitigated or retired
- Elaboration often consists of two or more iterations;
  - each iteration is recommended to be 2~6 weeks
- Elaboration is not a design phase or a phase when the models are fully developed in preparation for implementation.

Elaboration 2

- Executable architecture/Architectural baseline/ Architectural prototype
  - to describe the partial system.
  - a production subset of the final system.
- Some key ideas and best practices will manifest in elaboration:
  - do short time boxed risk-driven iterations
  - start programming early
  - adaptively design, implement, and test the core and risky parts of the architecture
  - test early, often, realistically
  - adapt based on feedback from tests, users, developers
  - write most of the use cases and other requirements in detail, through a series of workshops, once per elaboration iteration

Elaboration 3

Artifact	Comment
Domain Model	This is a visualization of the domain concepts; it is similar to a static information model of the domain entities.
Design Model	This is the set of diagrams that describes the logical design. This includes software class diagrams, object interaction diagrams, package diagrams, and so forth.
Software Architecture Document	A learning aid that summarizes the key architectural issues and their resolution in the design. It is a summary of the outstanding design ideas and their motivation in the system.
Data Model	This includes the database schemas, and the mapping strategies between object and non-object representations.
Use-Case Storyboards, UI Prototypes	A description of the user interface, paths of navigation, usability models, and so forth.

Planning the Next Iteration

- Organize requirements and iterations by risk, coverage, and criticality.
  - Risk includes both technical complexity and other factors, such as uncertainty of effort or usability.
  - Coverage implies that all major parts of the system are at least touched on in early iterations perhaps a "wide and shallow" implementation across many components.
  - Criticality refers to functions the client considers of high business value.

POS Risk List

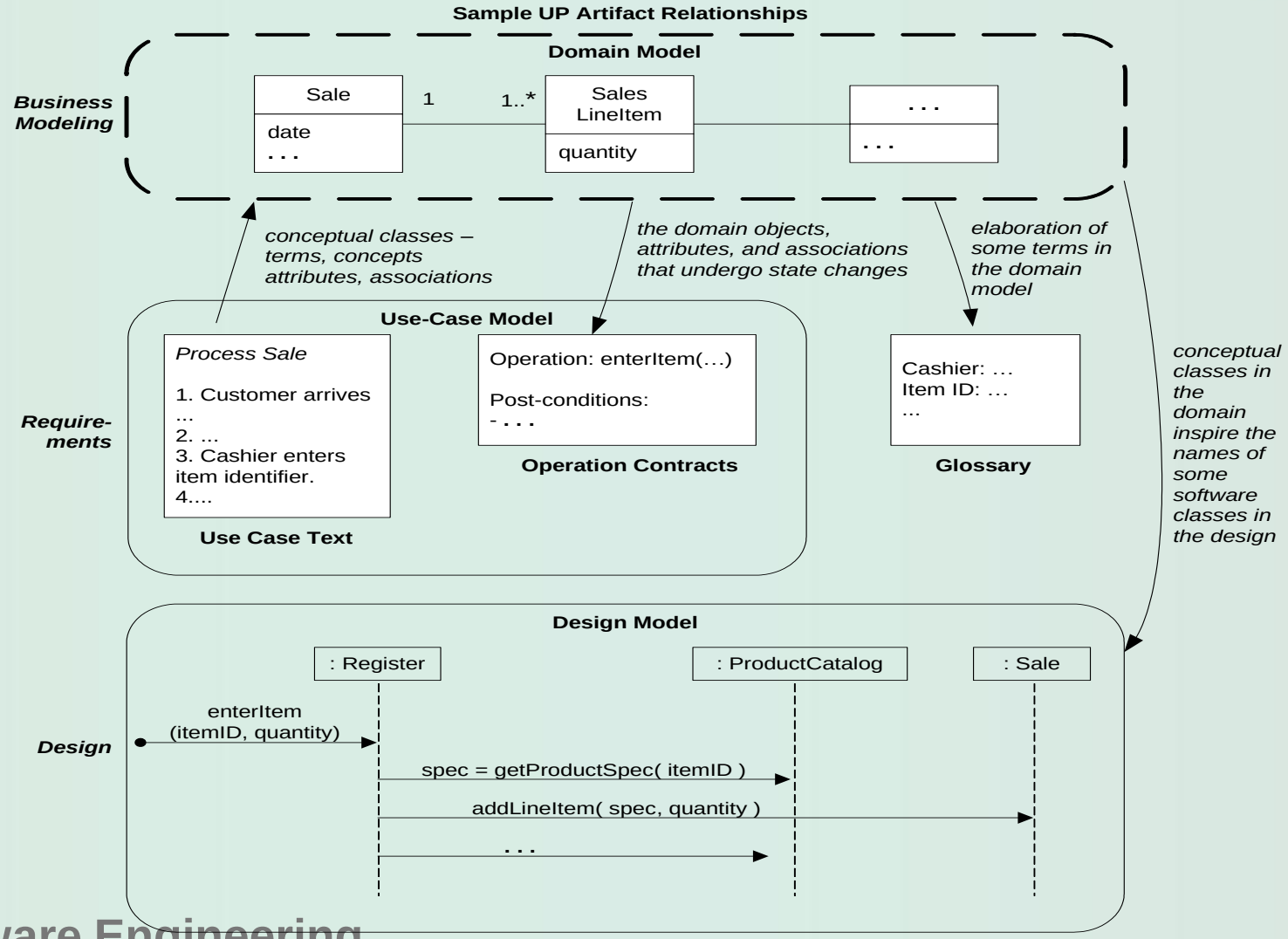
Rank	Requirement (Use Case or Feature)	Comment
High	Process Sale Logging …	Scores high on all rankings. Pervasive. Hard to add late. …
Medium	Maintain Users …	Affects security subdomain. …
Low	…	…

Chap 9
Domain Models

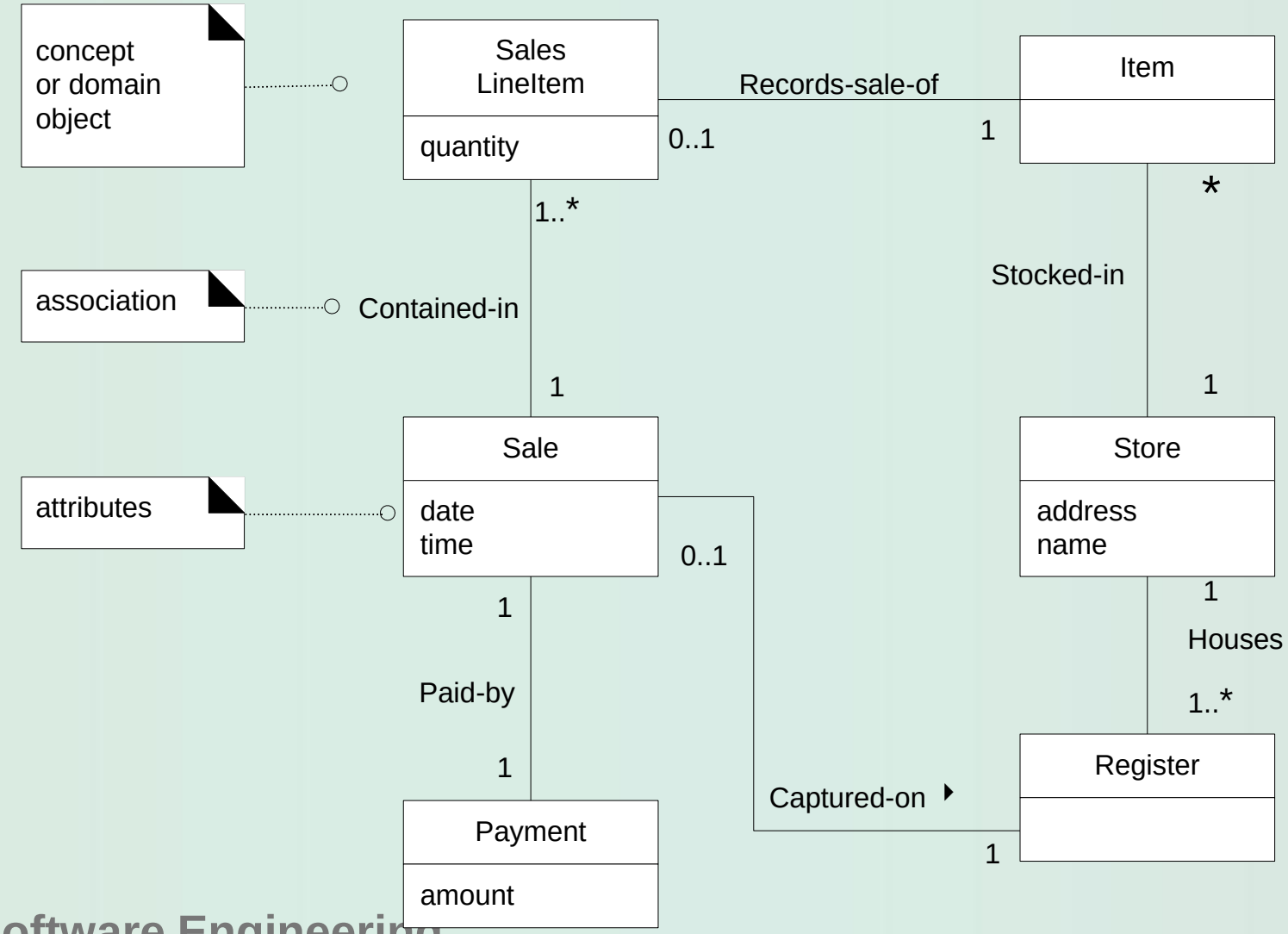
Introduction

- A domain model
  - the most important and classic model in OO analysis.
  - be a visual representation of conceptual classes or ***real situation objects*** in a domain.
  - Also called ***conceptual models***, ***domain object models***, and analysis object models.
  - "focusing on explaining 'things' and products important to a business domain“, such as POS related things.
- Guideline
  - Avoid a waterfall-mindset big-modeling effort to make a thorough or "correct" domain model
  - it won't ever be either, and such over-modeling efforts lead to analysis paralysis, with little or no return on the investment.

Sample UP artifact influence



POS Domain Model

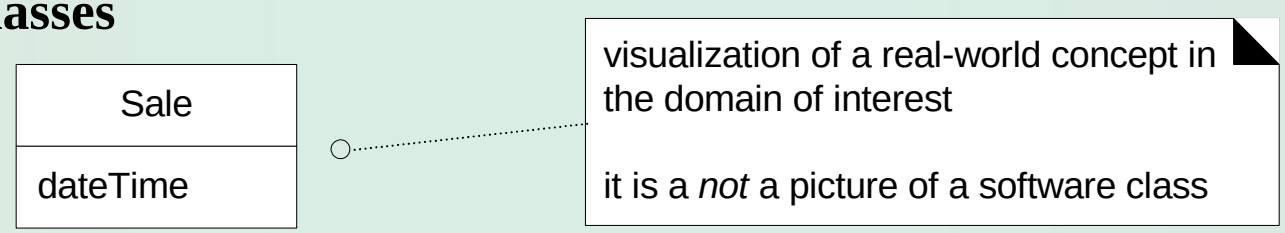


Domain Model 1

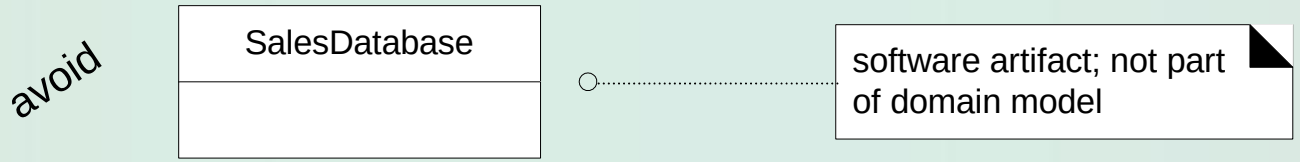
- It provides a conceptual perspective.
  - domain objects or conceptual classes
  - associations between conceptual classes
  - attributes of conceptual classes
- Following elements are not suitable in a domain model
  - Software artifacts, such as a window or a database, unless the domain being modeled is of software concepts, such as a model of graphical user interfaces.
  - Responsibilities or methods.

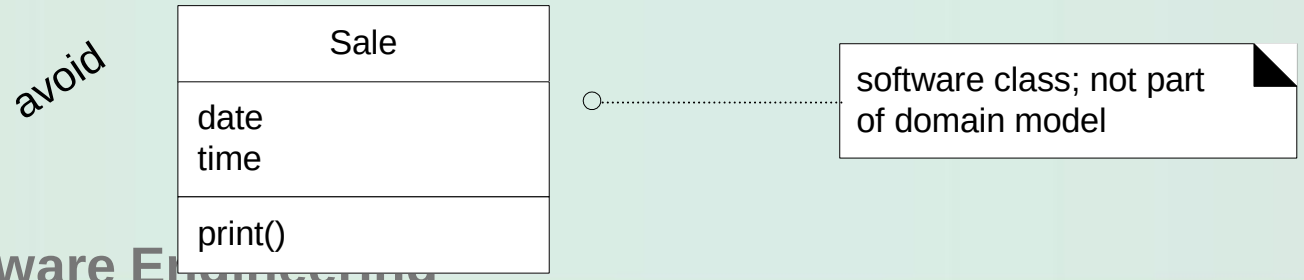
Domain Model 2

- **A domain model shows real-situation conceptual classes, not software classes**



- **A domain model does not show software artifacts or classes**

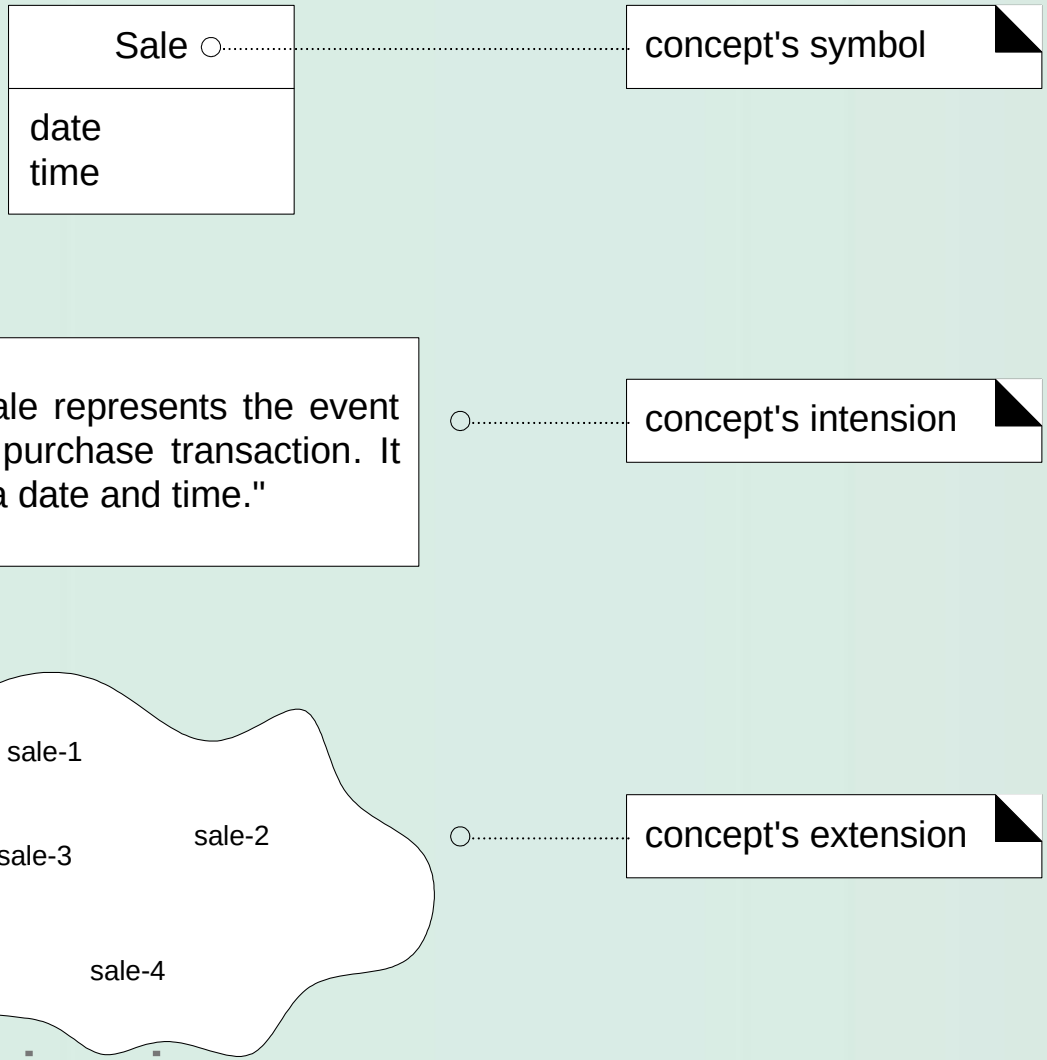




Domain Model 3

- A conceptual class is an idea, thing, or object.
  - Symbol words or images representing a conceptual class.
  - Intension the definition of a conceptual class.
  - Extension the set of examples to which the conceptual class applies

Domain Model 4



Domain Model 5

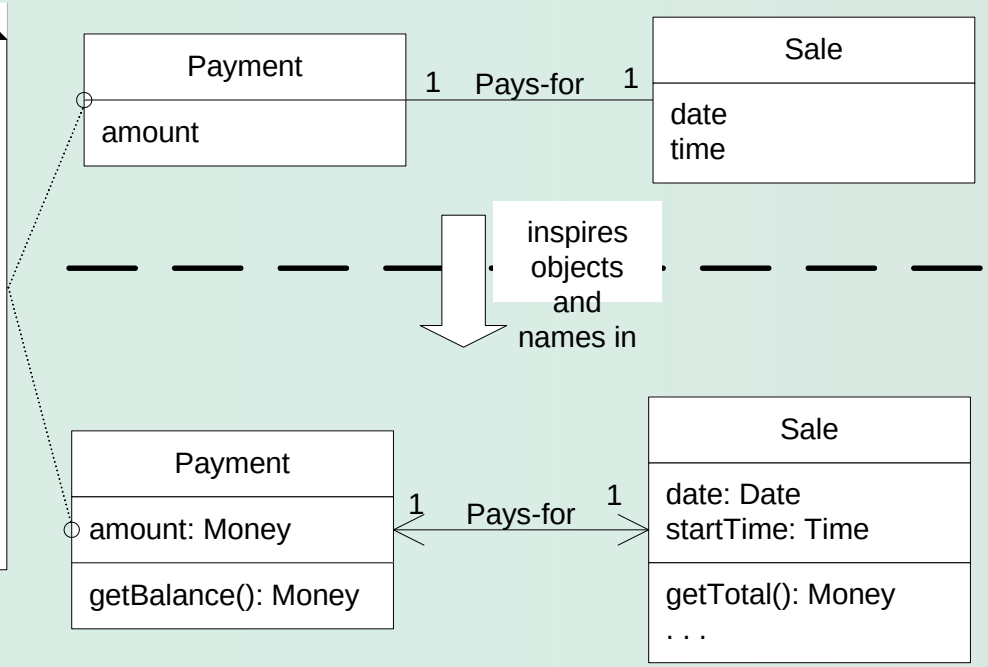
A Payment in the Domain Model is a concept, but a Payment in the Design Model is a software class. They are not the same thing, but the former *inspired* the naming and definition of the latter.

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.

- **Lower representational gap with OO modeling.**

Guideline: Create a Domain Model

- Bounded by the current iteration requirements under design
  - Find the conceptual classes (see a following guideline).
  - Draw them as classes in a UML class diagram.
  - Add the association ***ncessary*** to record relationships for which there is a need to preserve some memory.
  - Add the attributes ***necessary*** to fulfill the information requirements.

Guideline: Find Conceptual Classes 1

- Reuse or modify existing models.
- Use a category list.
- Identify noun phrases from the case text

Conceptual Class Category	Examples
business transactions **Guideline: These are critical (they involve money), so start with transactions.**	**Sale, Payment** **Reservation**
transaction line items **Guideline: Transactions often come with related line items, so consider these next.**	**SalesLineItem**
product or service related to a transaction or transaction line item **Guideline: Transactions are for something (a product or service). Consider these next.**	**Item** **Flight, Seat, Meal**
where is the transaction recorded? **Guideline: Important.**	**Register, Ledger** **FlightManifest**

Guideline: Find Conceptual Classes 2

Conceptual Class Category	Examples
place of transaction; place of service	Store Airport, Plane, Seat
noteworthy events, often with a time or place we need to remember	Sale, Payment, Flight
physical objects Guideline: This is especially relevant when creating device-control software, or simulations.	Item, Register Board, Piece, Die Airplane
descriptions of things Guideline: See p. 147 for discussion.	Product Description Flight Description
catalogs Guideline: Descriptions are often in a catalog.	Product Catalog Flight Catalog
containers of things (physical or information)	Store, Bin Board Airplane

Guideline: Find Conceptual Classes 3

Conceptual Class Category	Examples
things in a container	Item Square (in a Board) Passenger
other collaborating systems	Credit Authorization System Air Traffic Control
records of finance, work, contracts, legal matters	Receipt, Ledger MaintenanceLog
financial instruments	Cash, Check, LineOfCredit TicketCredit
schedules, manuals, documents that are regularly referred to in order to perform work	DailyPriceChangeList RepairSchedule

Guideline: Find Conceptual Classes 4

Conceptual Class Category	Examples
Physical or tangible object	POST, Airplane
Roles of people	Cashier, Pilot
Abstract noun concepts	Hunger, Acrophobia
Organizations	Sales Department
Events	Sale, Meeting, Flight
Process	SellingAProduct, Booking
Rules and policies	RefundPolicy

Find Conceptual Classes: POS

- A concept is an idea or notion that we apply to the things.
  - Intension: the definition of concept, e.g. the Customer may be a person or organization that purchases goods or services
  - Extension: the set of all objects to which the concept applies, e.g. the Customer may be “ John”, Tom”

<<tangible object>> POSTerminal	<<thing in container>> Item	<<place>> Store	<<event or transactions>> Sale
<<transaction line items>> SalesLineItem	<<role of people>> Cashier	<<roles of people>> Customer	<<transaction>> Payment
<<catalog>> ProductCatalog	<<descritpions of things>> ProductSpec		

Guideline: Find Conceptual Classes 5

- Identify noun phrases.
  - Identify the nouns and noun phrases in textual descriptions of a domain, and consider them as candidate conceptual classes or attributes
  - Some of these noun phrases may refer to conceptual classes that are ignored in this iteration (e.g., "Accounting" and "commissions"), and some may be simply attributes of conceptual classes.
  - A weakness of this approach is the imprecision of natural language; different noun phrases may represent the same conceptual class or attribute, among other ambiguities.

Find Conceptual Classes: POS 1

- 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).

Find Conceptual Classes: POS 2

- 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.

Find Conceptual Classes: POS 3

- For iteration-1, the basic cash-only scenario of Process Sale.
  - Sale, Cashier, Cash, Payment,
  - Customer, Sales Line Item,
  - Store, Item, Product Description,
  - Register, Product Catalog, Ledger.

Guideline:

Agile Modeling Maintain the Model in a Tool

- Perfection is not the goal of Agile, and agile models are usually discarded shortly after creation.
  - From this viewpoint, there is no motivation to maintain or update the model.
- If someone wants the model maintained and updated with new discoveries
  - do the drawing with a UML tool.

Guideline:

Report Objects Include 'Receipt' in the Model

- Receipt is a noteworthy term in the POS domain. But perhaps it's only a report of a sale and payment, and duplicate information. Two factors to consider
  - **Exclude it:** Showing a report of other information in a domain model is not useful since all its information is ***derived or duplicated*** from other sources.
  - **Include it:** it has a special role in terms of the business rules: It confers the right to the bearer of the receipt to return bought items.
- Since **item returns** are not being considered in this iteration, Receipt will be excluded.
  - During the iteration that tackles the Handle Returns use case, we would be justified to include it.

Guideline:
Think Like a Mapmaker; Use Domain Terms

- Make a domain model in the spirit of how a cartographer or mapmaker works:
  - Use the existing names in the territory. For example, if developing a model for a library, name the customer a "Borrower" the terms used by the library staff.
  - Exclude irrelevant or out-of-scope features.
  - Do not add things that are not there.

Guideline:
How to Model the Unreal World

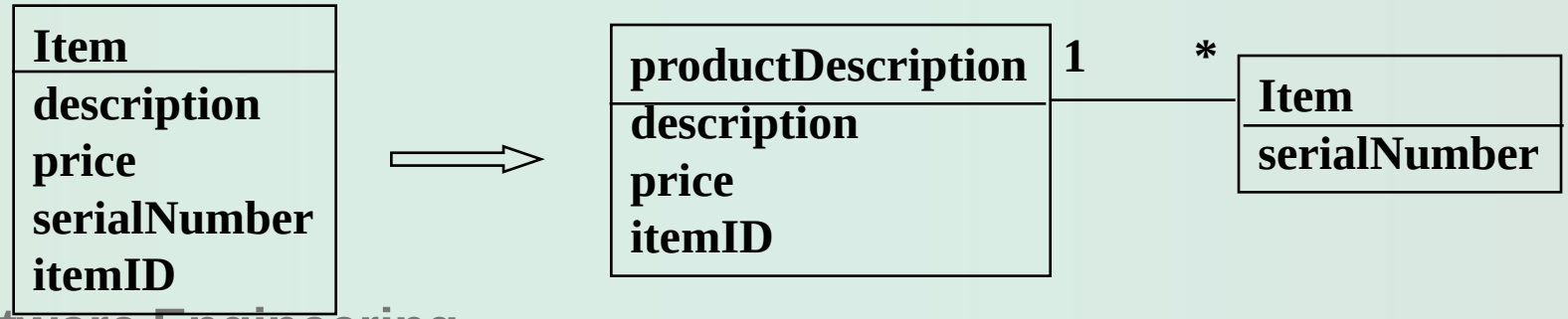
- It requires a high degree of abstraction, and listening carefully to the ***core vocabulary and concepts*** that domain experts use.

Guideline:
A Common Mistake with Attributes vs. Classes

- If we do not think of some conceptual class X as a number or text in the real world, X is probably a conceptual class, not an attribute.
  - In the real world, a **store** is not considered a number or text, the term suggests a legal entity, an organization, and something that occupies space. Therefore, Store should be a conceptual class.

Guideline:
When to Model with 'Description' Classes

- A description class contains information that describes something else. For example, a ProductDescription that records the price, picture, and text description of an Item.
  - Problems: if implemented in software similar to the domain model, it has duplicate data (space-inefficient, and error-prone). Because the description, price, and itemID are duplicated for every Item instance of the same product
  - A particular Item may have a serial number; it represents a physical instance. A ProductDescription wouldn't have a serial number



Association 1

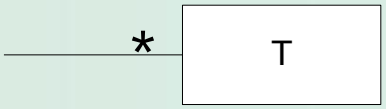
- Association
  - a relationship between classes (instances of those classes) that indicates ***some meaningful and interesting*** connection.
- Guideline: When to Show an Association?
  - Associations imply knowledge of a relationship that needs to be preserved for some duration.
- Guideline: Avoid Adding Many Associations
  - Many lines on the diagram will obscure it with "visual noise."
- Perspectives: Will the Associations Be Implemented In Software?
  - During domain modeling, an association is not data flows, database foreign key relationships, instance variables, or object connections in software solution.
  - it is meaningful in a purely conceptual perspective in the real domain.

Association 2

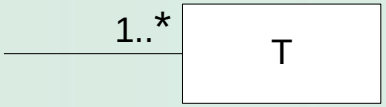
- Guideline
  - Name an association based on a ClassName-VerbPhrase-ClassName format where the verb phrase creates a sequence that is readable and meaningful.
  - Association names should start with a capital letter, since an association represents a classifier of links between instances;
  - e.g. Sale Paid-by CashPayment: bad example (doesn't enhance meaning): Sale Uses CashPayment
  - e.g. Player Is-on Square: bad example (doesn't enhance meaning): Player Has Square
- Applying UML: Roles
  - Each end of an association is called a role. Roles may optionally have multiplicity expression, name, navigability.

Association 3

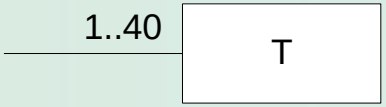
- Applying UML: Multiplicity
  - Multiplicity defines how many instances of a class A can be associated with one instance of a class B



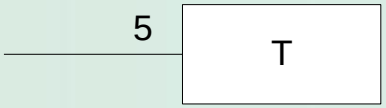
zero or more; "many"



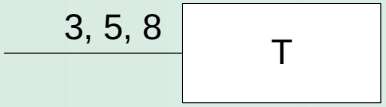
one or more



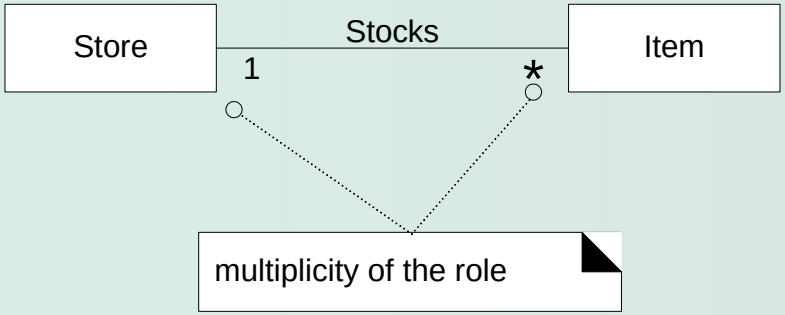
one to 40



exactly 5

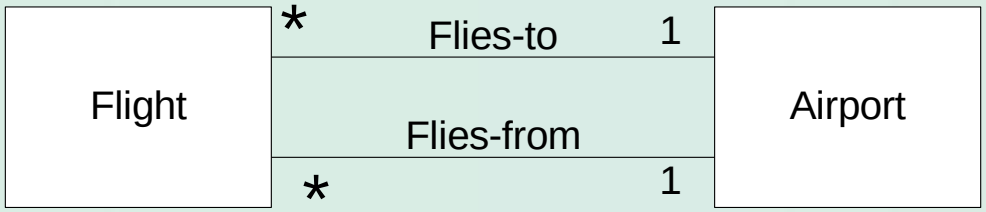


exactly 3, 5, or 8



Association 4

- Applying UML: Multiple Associations Between Two Classes



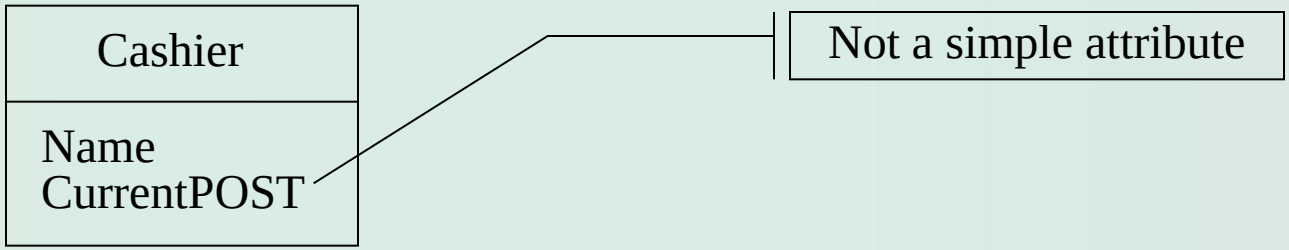
- Guideline: Find Associations with a Common Associations List

Category	Examples
A is a transaction related to another transaction B	**CashPaymentSale CancellationReservation**
A is a line item of a transaction B	**SalesLineItemSale**
A is a product or service for a transaction (or line item) B	**ItemSalesLineItem (or Sale), FlightReservation**
A is a role related to a transaction B	**CustomerPayment PassengerTicket**

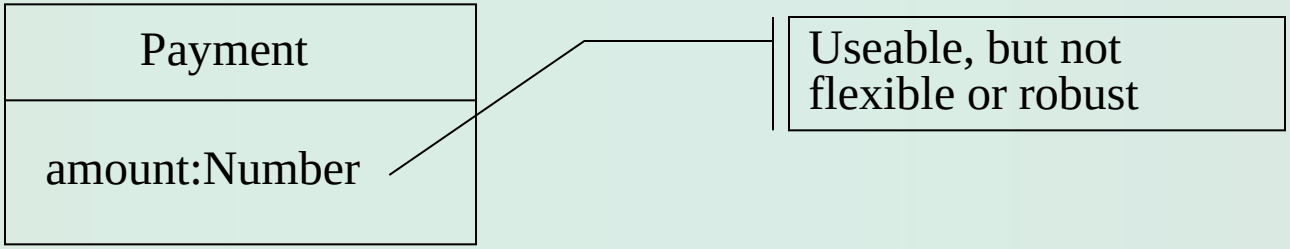
Association 5

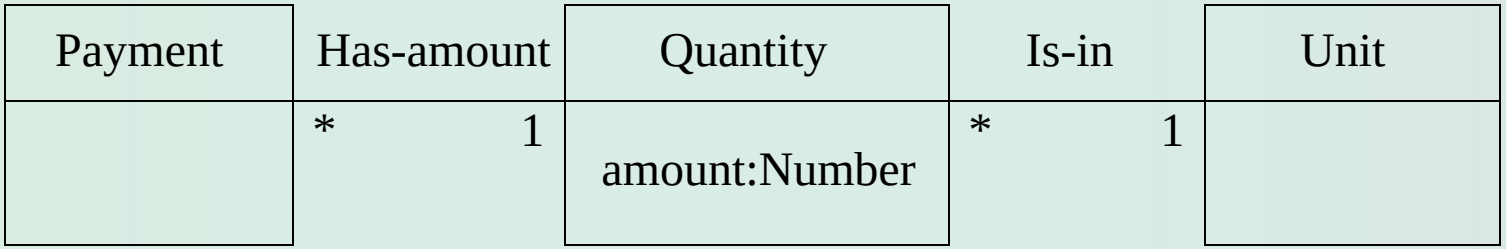
Category	Examples
A is a physical or logical part of B	DrawerRegister, SquareBoard, SeatAirplane
A is physically or logically contained in/on B	RegisterStore, ItemShelf, SquareBoard, PassengerAirplane
A is a description for B	ProductDescriptionItem, FlightDescriptionFlight
A is known/logged/recorded/ reported/captured in B	SaleRegister, PieceSquare, ReservationFlightManifest
A is a member of B	CashierStore, PlayerMonopolyGame, PilotAirline
A is an organizational subunit of B	DepartmentStore, MaintenanceAirline
A uses or manages or owns B	CashierRegister, PlayerPiece, PilotAirplane
A is next to B	SalesLineItemSalesLineItem, SquareSquare, CityCity

Relate with Association

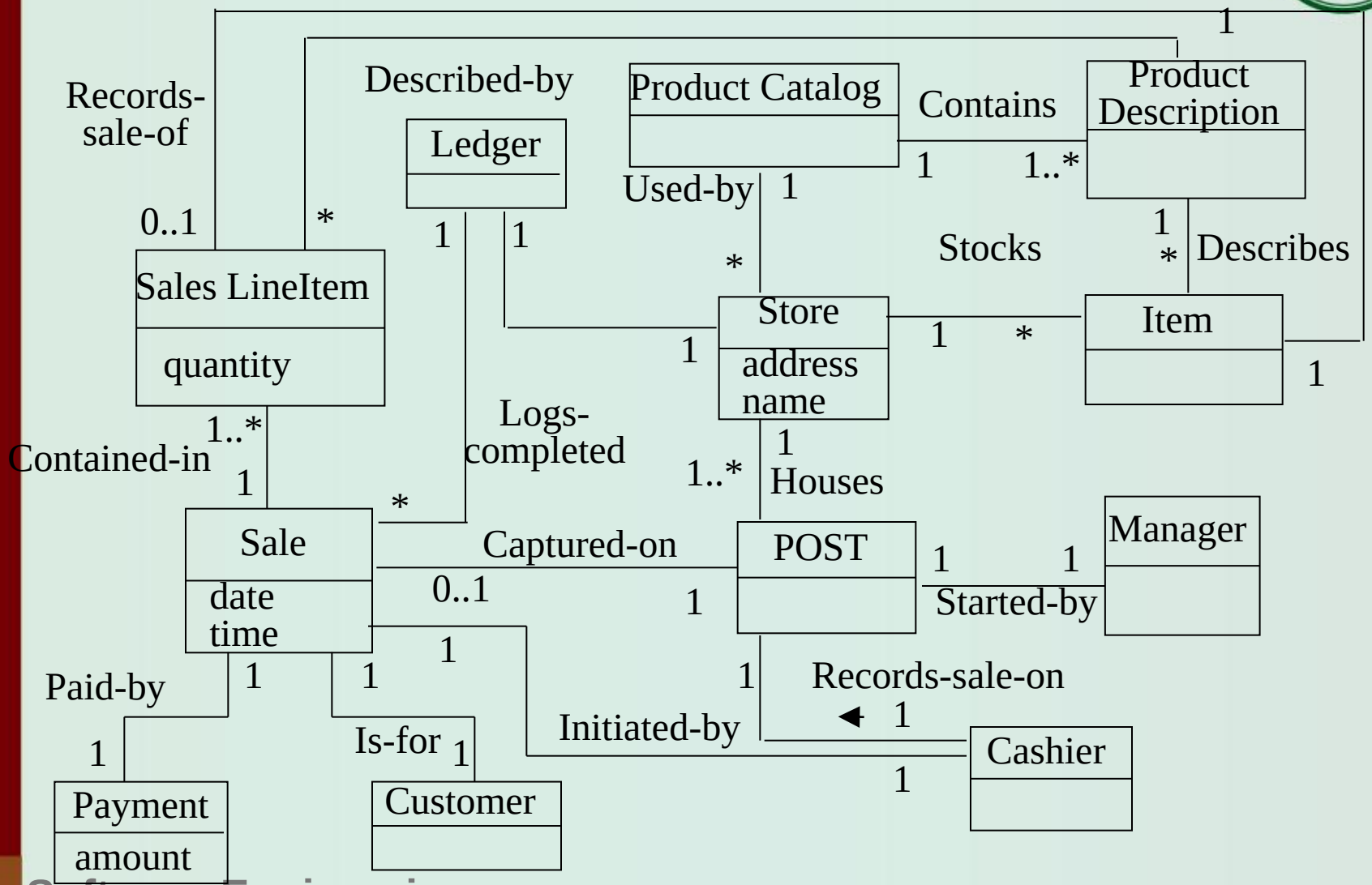


Modeling Quantities



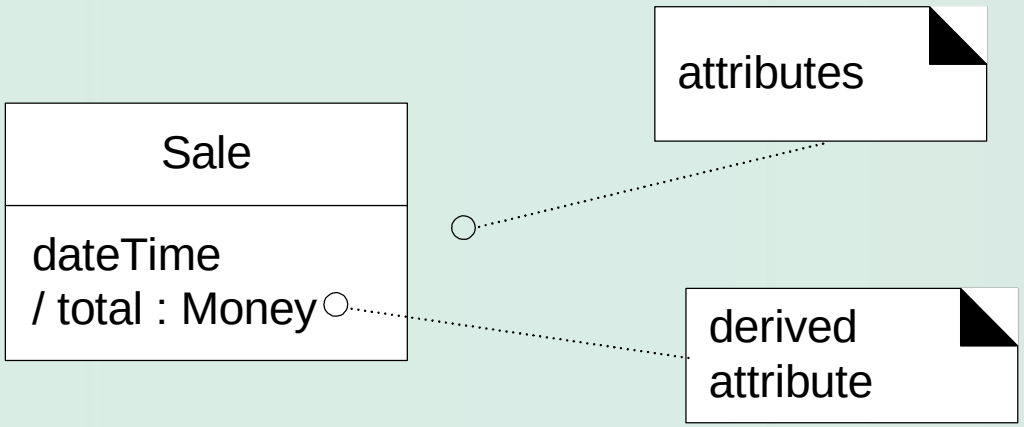


POS Partial Domain Model



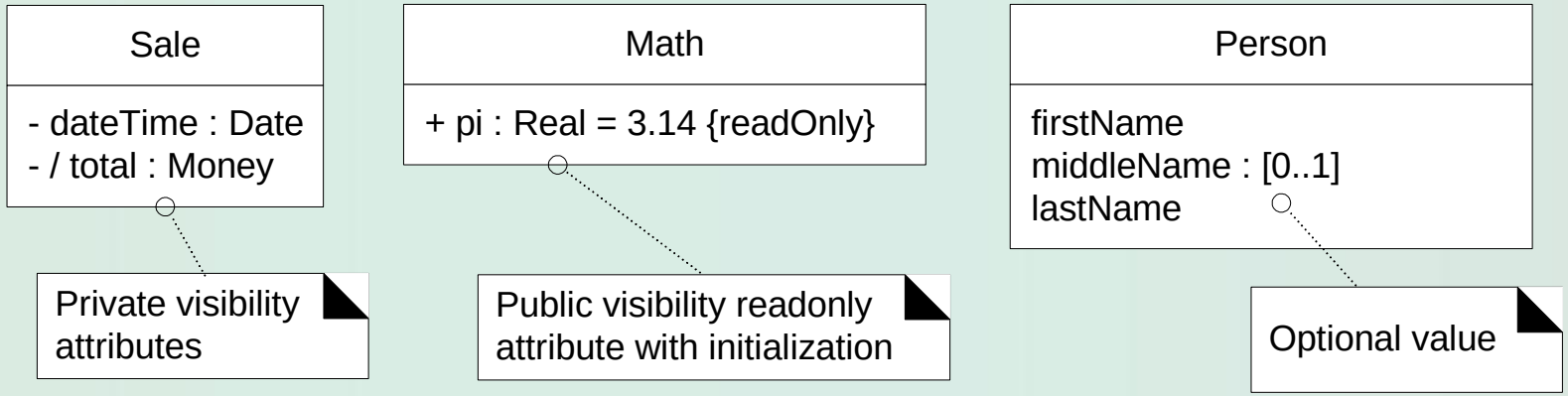
Attributes 1

- An **attribute** is a logical data value of an object.
- **Guideline: When to Show Attributes?**
  - Include attributes that the requirements (e.g., use cases) ***suggest or imply a need*** to remember information.
  - e.g., a receipt (which reports the information of a sale) in the Process Sale use case normally includes a date and time, the store name and address, and the cashier ID,



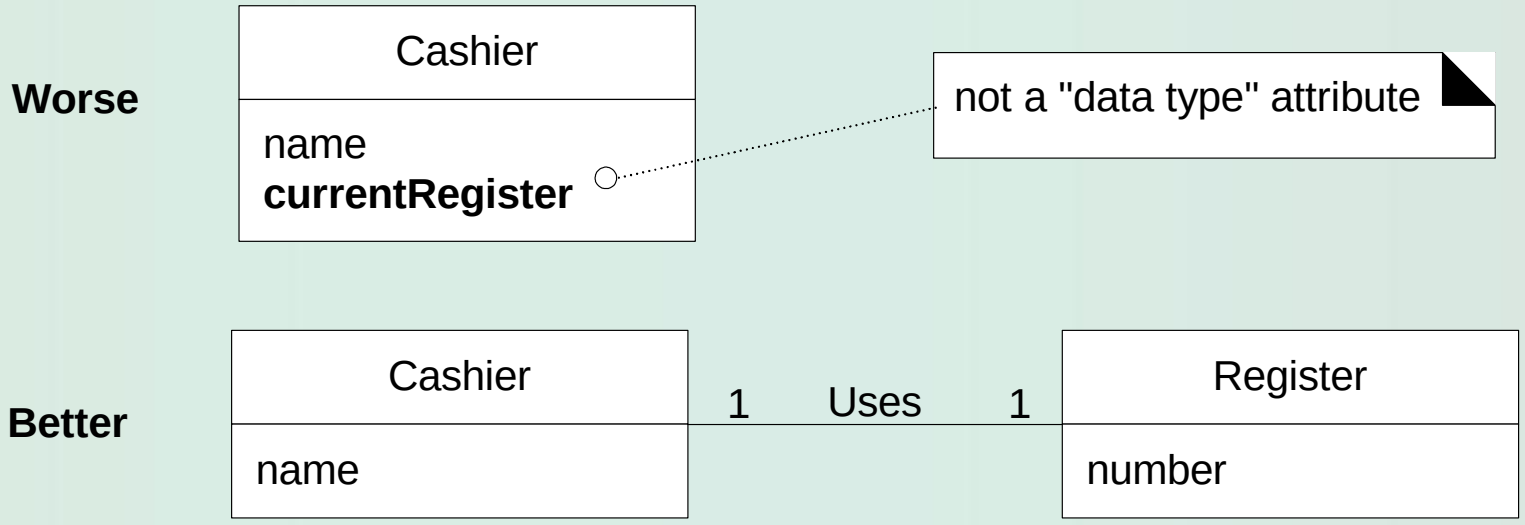
Attributes 2

- Guideline: Where to Record Attribute Requirements?
  - to use a tool that integrates UML models with a data dictionary; then all attributes will automatically show up as dictionary elements.
- ***Derived Attributes***
  - The total attribute in the Sale can be calculated or derived from the information in the SalesLineItems.



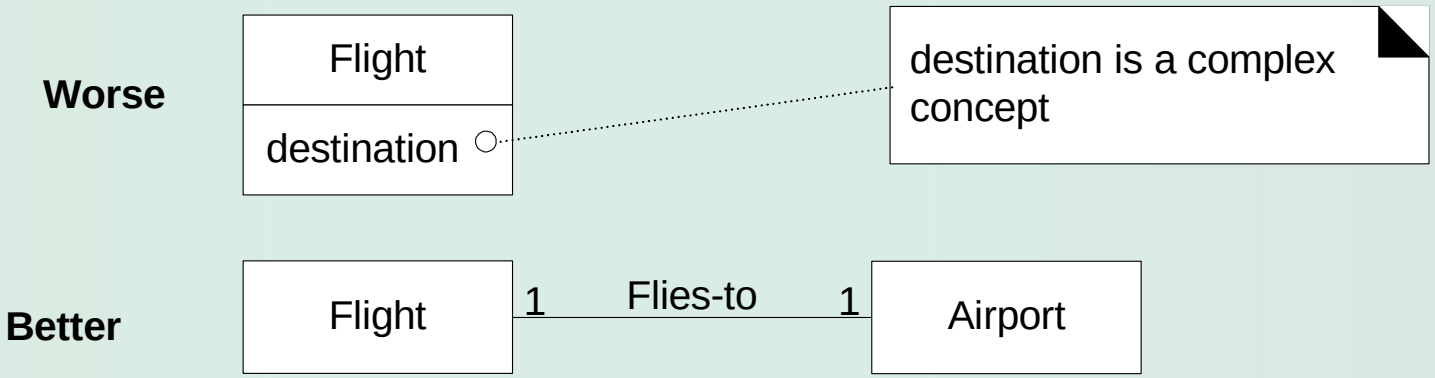
Attributes 3

- Guideline: Focus on Data Type Attributes in the Domain Model.
  - most attribute types should be primitive data types, such as numbers and booleans.



Attributes 4

- Guideline: Don't show complex concepts as attributes; use associations

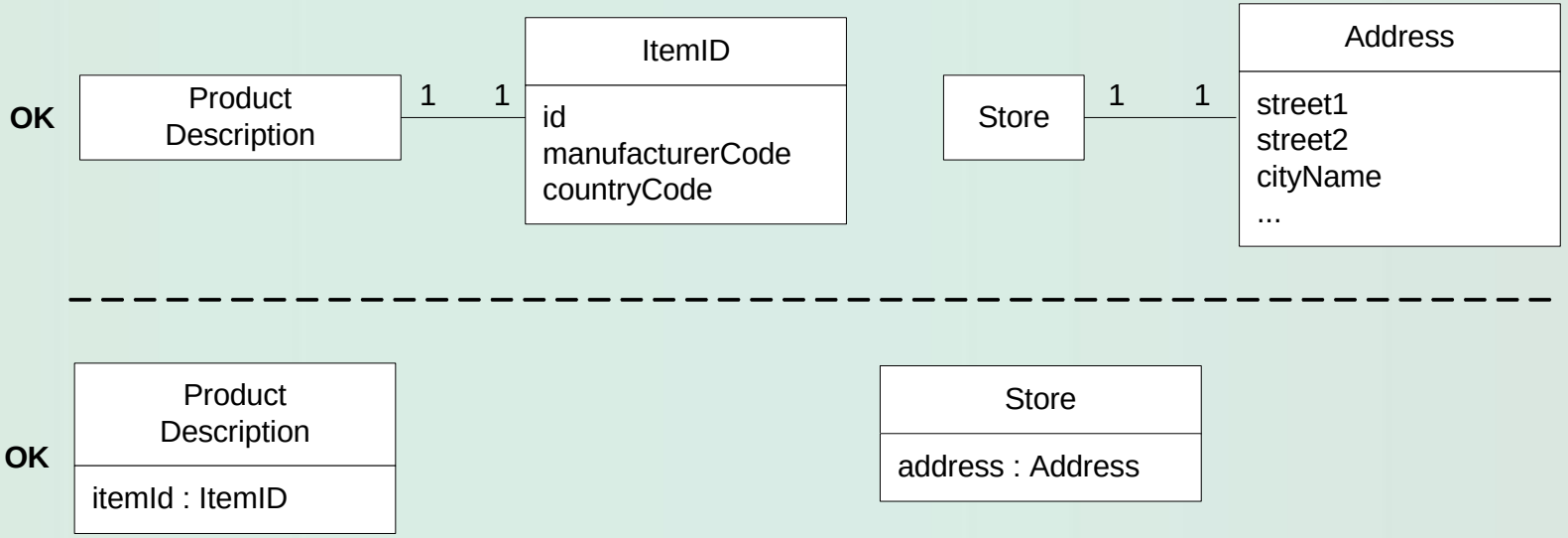


Attributes 5

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

Attributes 6

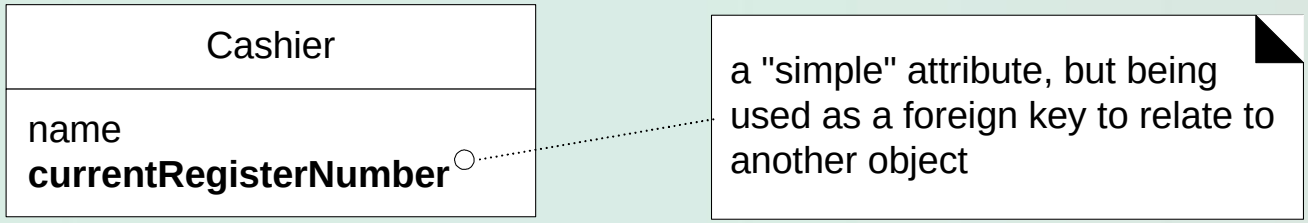
- Applying UML: Two ways to indicate a data type property of an object.



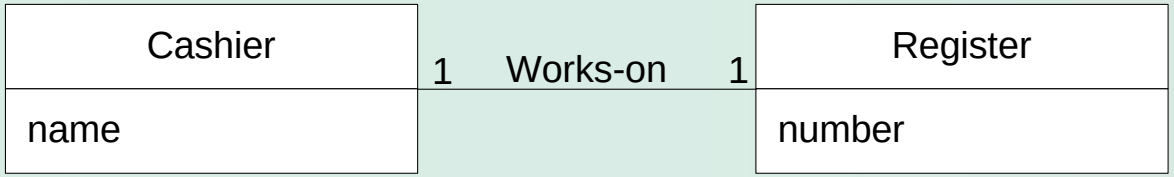
Attributes 7

- Guideline: No Attributes Representing Foreign Keys.

Worse

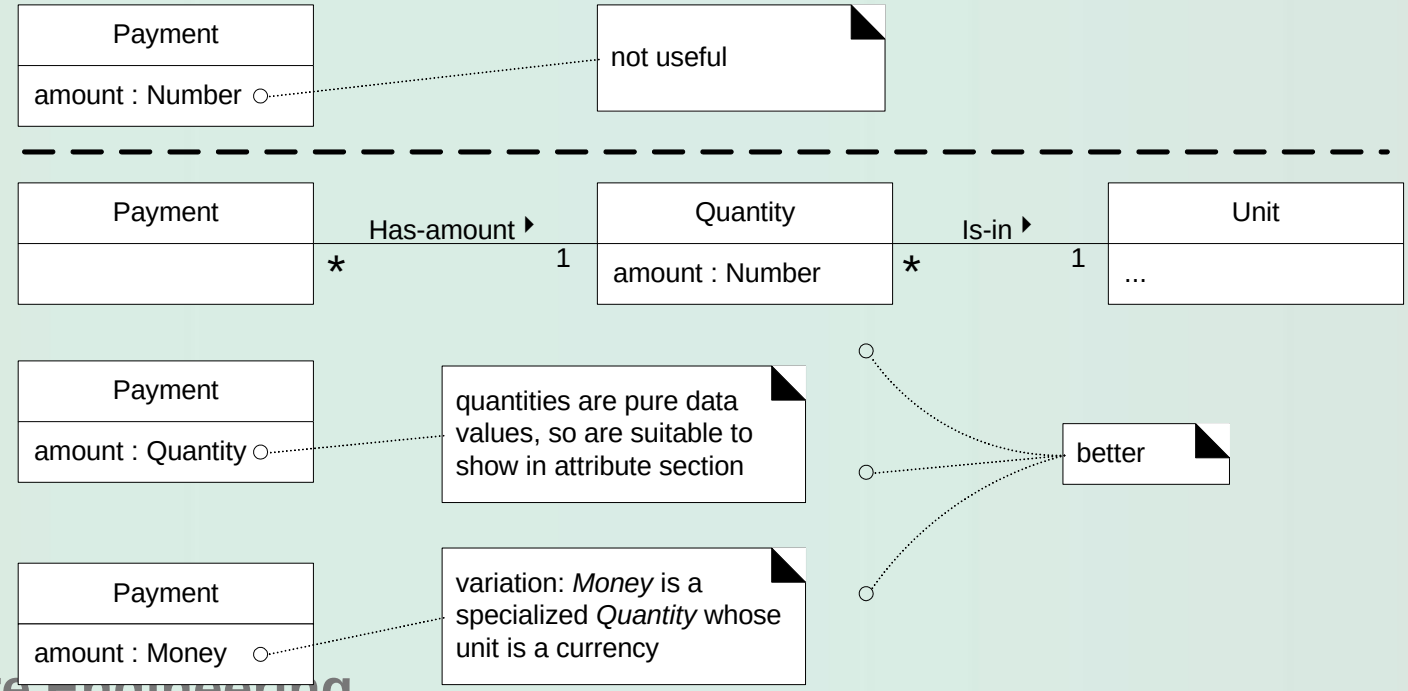


Better

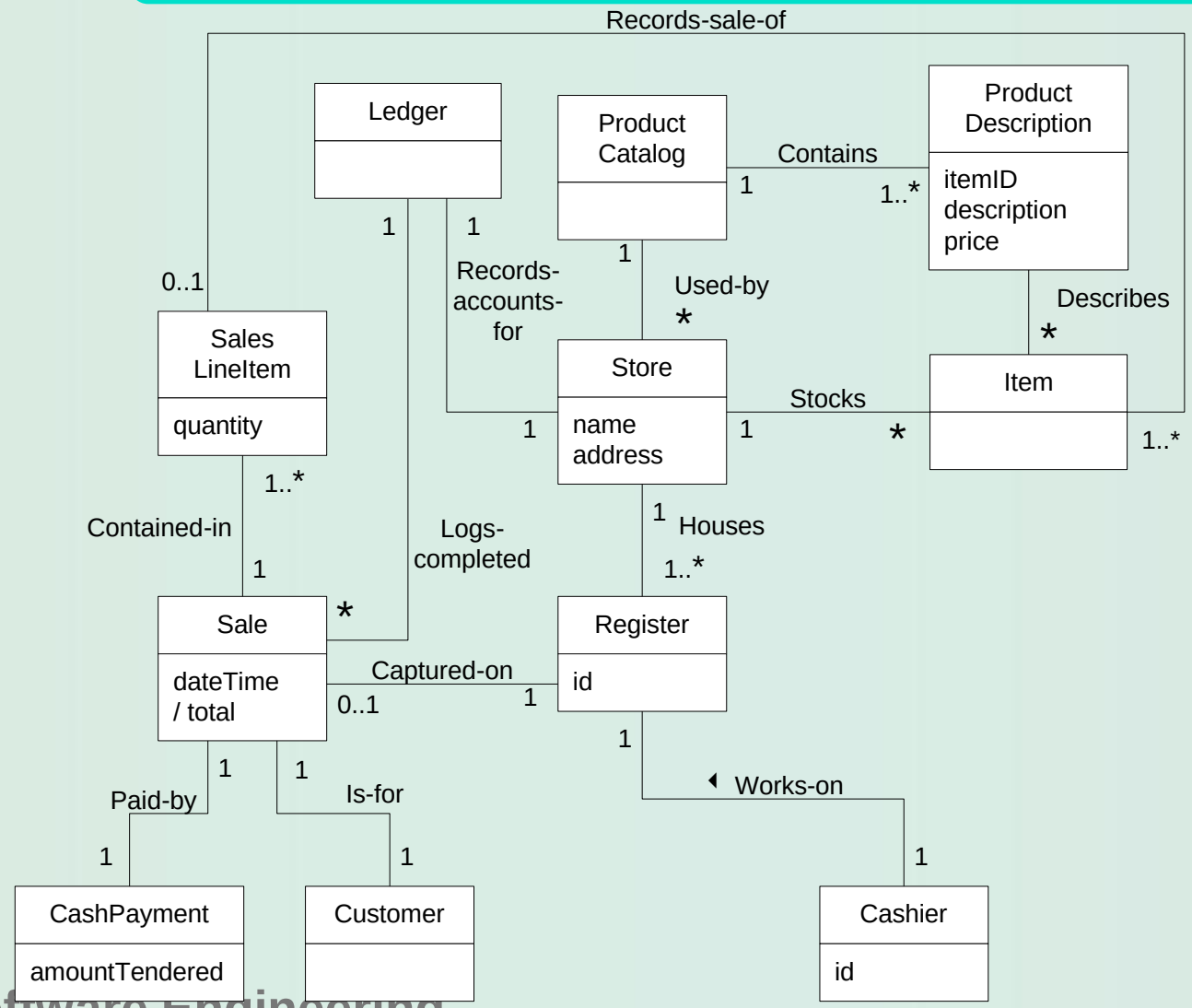


Attributes 8

- Guideline: Modeling Quantities and Units.
  - Most numeric quantities should not be represented as plain numbers.
  - To represent Quantity as a distinct class, with an associated Unit.
  - To show Quantity specializations. Money is a kind of quantity whose units are currencies. Weight is a quantity with units such as kilograms or pounds.



POS Partial Domain Model



案例与实践

- 牧师与魔鬼

Iterative and Evolutionary Domain Modeling

Discipline	Artifact	Incep.	Elab.	Const.	Trans.
	Iteration	**I1**	**E1..En**	**C1..Cn**	**T1..T2**
Business Modeling	Domain Model		s		
Requirements	Use-Case Model (SSDs)	s	r		
	Vision	s	r		
	Supplementary Specification	s	r		
	Glossary	s	r		
Design	Design Model		s	r	
	SW Architecture Document		s		
	Data Model		s	r	