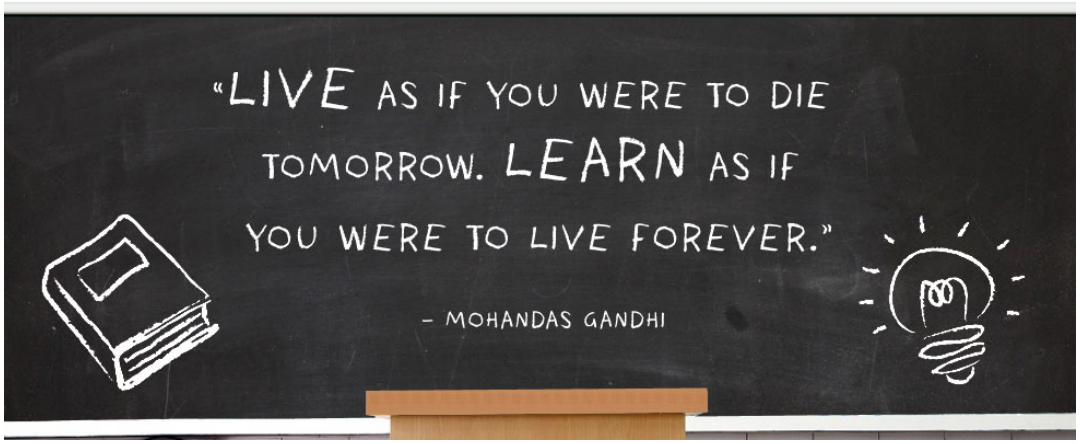
Vorlesung Softwaretechnik
SS 2019 / Teil II

Prof. Dr.-Ing. Samuel Kounev
M.Sc. Johannes Grohmann



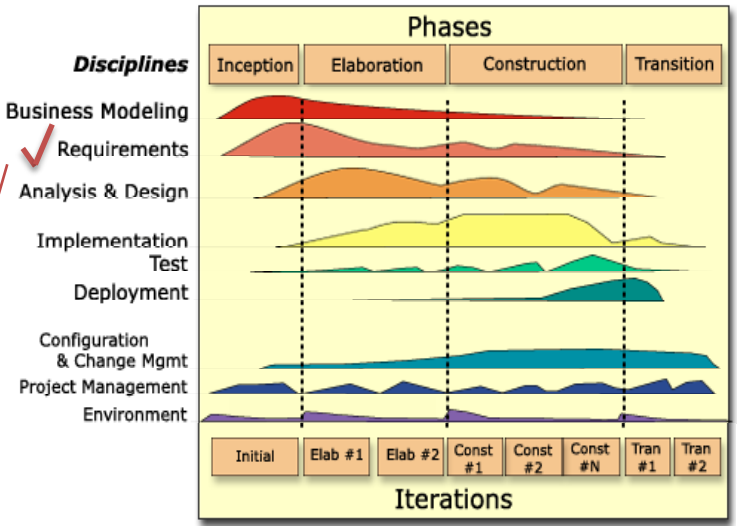
3. Domain Modeling

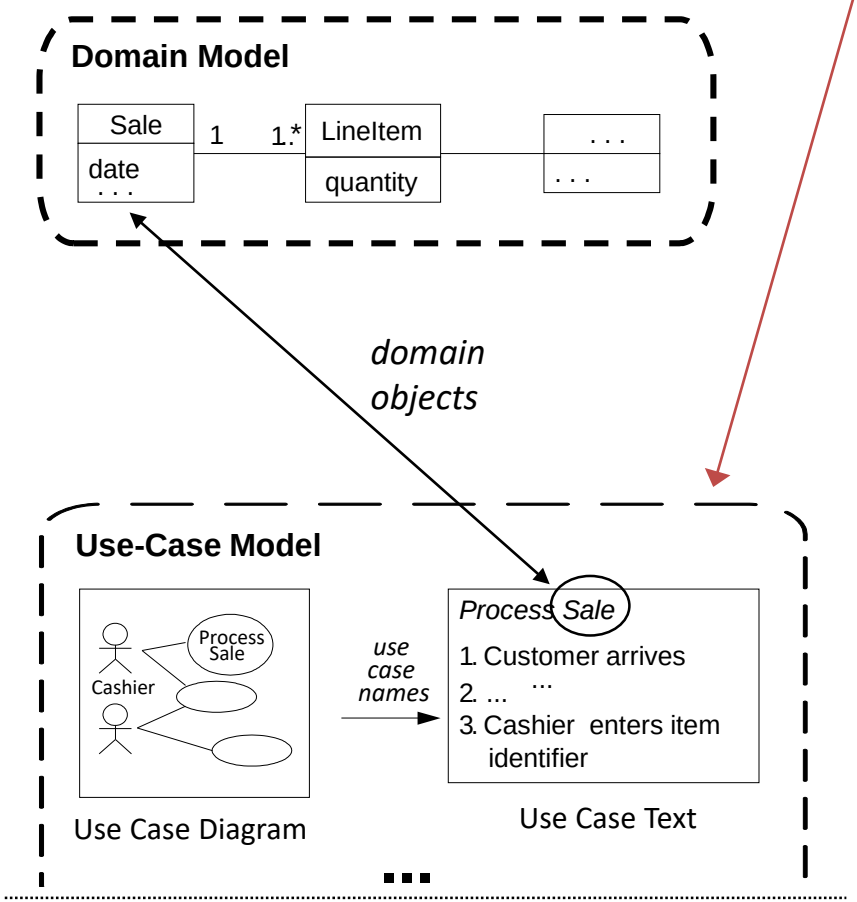
(a.k.a. Business Modeling)

Prof. Dr.-Ing. Samuel Kounev

Roadmap

- What's next?
  - Domain/business modeling





Analysis vs. Design ↔ What vs. How

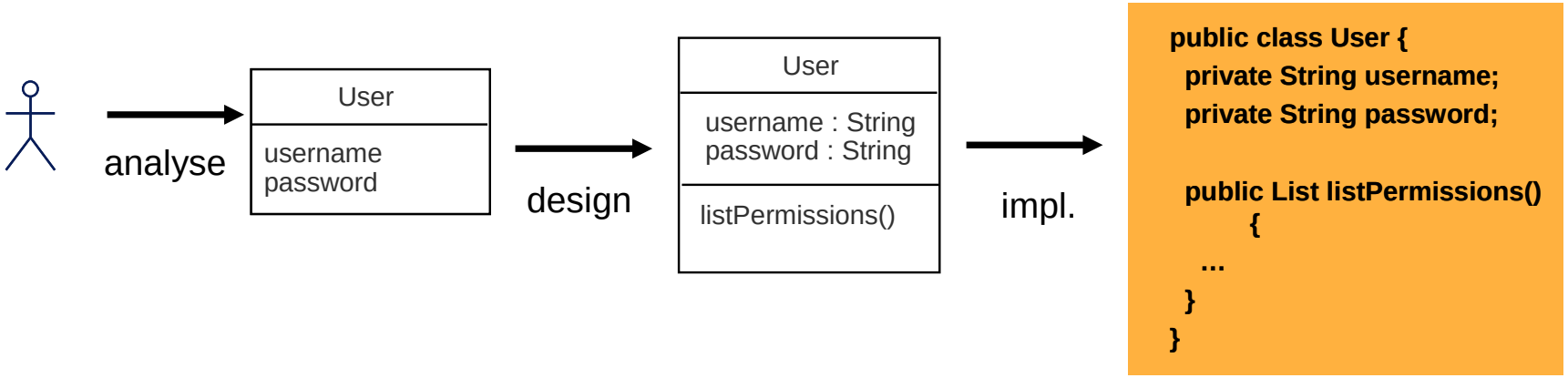
- **Analysis** emphasizes the investigation of **the problem** and requirements rather than a solution
  - broad term
    - requirements analysis → investigation of the requirements
    - object-oriented analysis → investigation of the domain objects
    - …

- **Design** emphasizes **a conceptual solution** that fulfills the requirements, rather than its implementation
  - broad term
    - object-oriented design
    - database design
    - …

"Do the right thing" (analysis) vs. "do the thing right" (design)

Object-Oriented Analysis & Design (OOA/D)

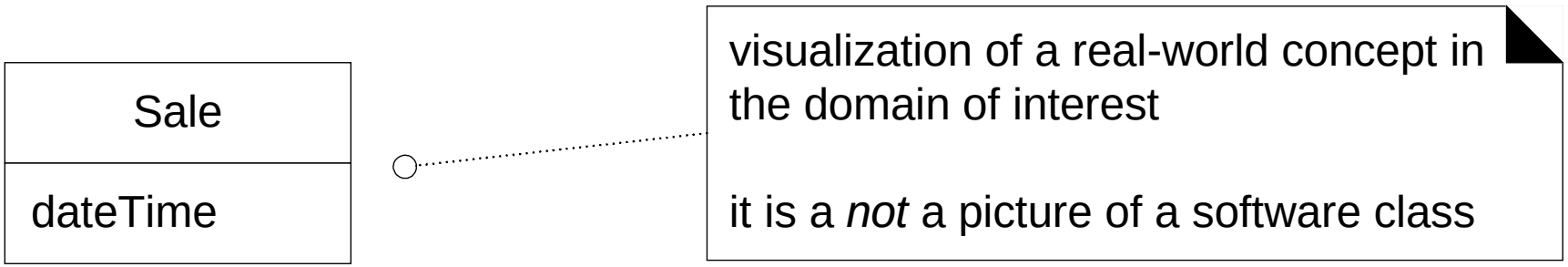
- Object-oriented **analysis** (OOA)
  - Finding and describing the objects (concepts) in the problem domain
- Object-oriented **design** (OOD)
  - Defining software objects and how they collaborate to fulfill requirements



Real World ***Analysis Level*** ***Design Level*** ***Implementation***

Divide and Conquer in Domain Analysis

- Since software problems can be complex, **decomposition** (**divide and conquer**) is a commonly applied strategy
- In traditional (structured analysis) techniques, the dimension of decomposition is the function
- In OOA, the dimension of decomposition is things or entities in the domain of interest (i.e., real-world concepts / objects)



Domain Model (1/4)

- Visual representation of **concepts** in the domain of interest
  - describes real world objects that are relevant / noteworthy
  - it does **not describe** software classes or objects
    - but provides a source of inspiration for designing software objects later
  - domain models most often focus on data
    - that needs to be represented in the system later
- Decomposes the domain of interest
  - into individual **conceptual classes** or objects in the real world
  - in order to be able to “rebuild” the real world in the object-oriented software system later

Domain Model (2/4)

- May be considered a **visual dictionary** of the:
  - noteworthy abstractions
  - domain vocabulary
  - and information content

  in the domain of interest

- Also referred to as
  - conceptual (object) model, domain object model, analysis object model

- Based on the notation of **UML class diagrams**
  - displays a partial view of the world and ignores uninteresting details

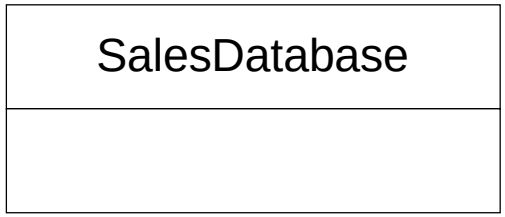
Domain Model (3/4)

- A domain model may show
  - domain objects or conceptual classes
  - associations between conceptual classes
  - attributes of conceptual classes

- Things that should **not** be in a domain model include –
  - software classes or objects
  - software artifacts such as a window or a database
    - (unless this is the domain being modeled)
  - object responsibilities (methods)
  - ….

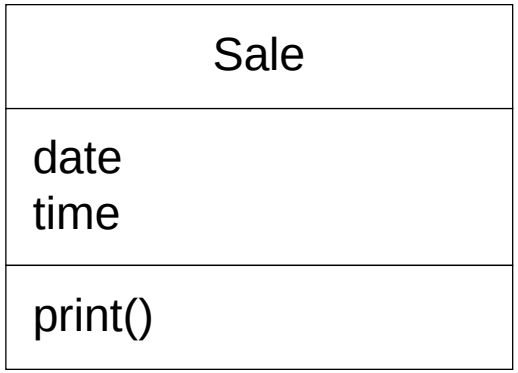
Domain Model (4/4)

avoid



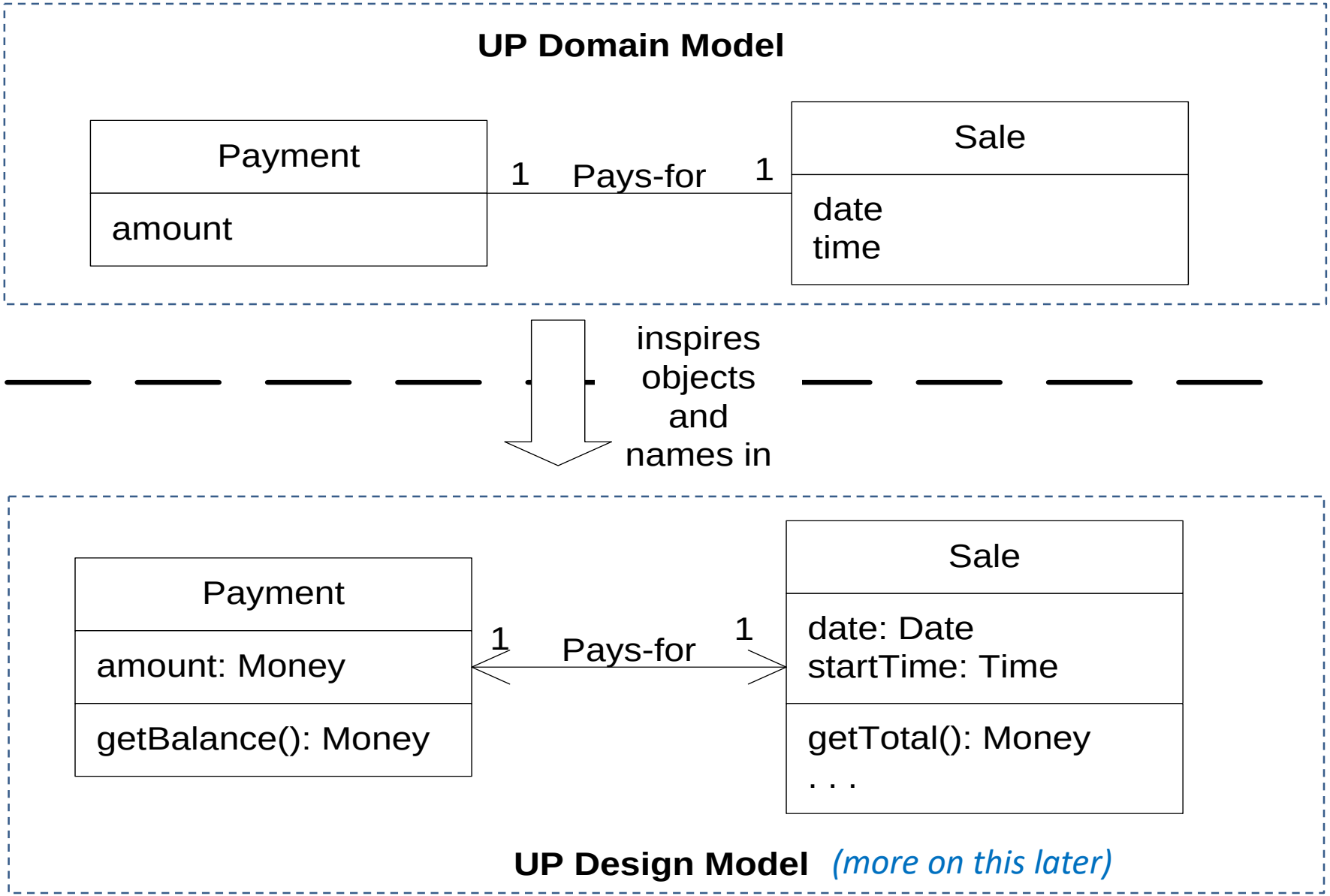
software artifact; not part of domain model

avoid



software class; not part of domain model

Low Representational Gap



How to Create a Domain Model

1. Find and list the candidate **conceptual classes**
2. Draw them in a domain model
3. Add **associations** to record relationships for which there is a need to preserve some memory
4. Add **attributes** to fulfill the information requirements

UP Context

Discipline	Artifact	Incep.	Elab.	Const.	Trans.
	Iteration →	I1	E1..En	C1..Cn	T1..T2
Business Modeling	***Domain Model***		s,r		
Requirements	***Use-Case Model***	s	r		
	Vision	s	r		
	Supplementary Specification	s	r		
	Glossary	s	r		
	Business Rules	s	r		

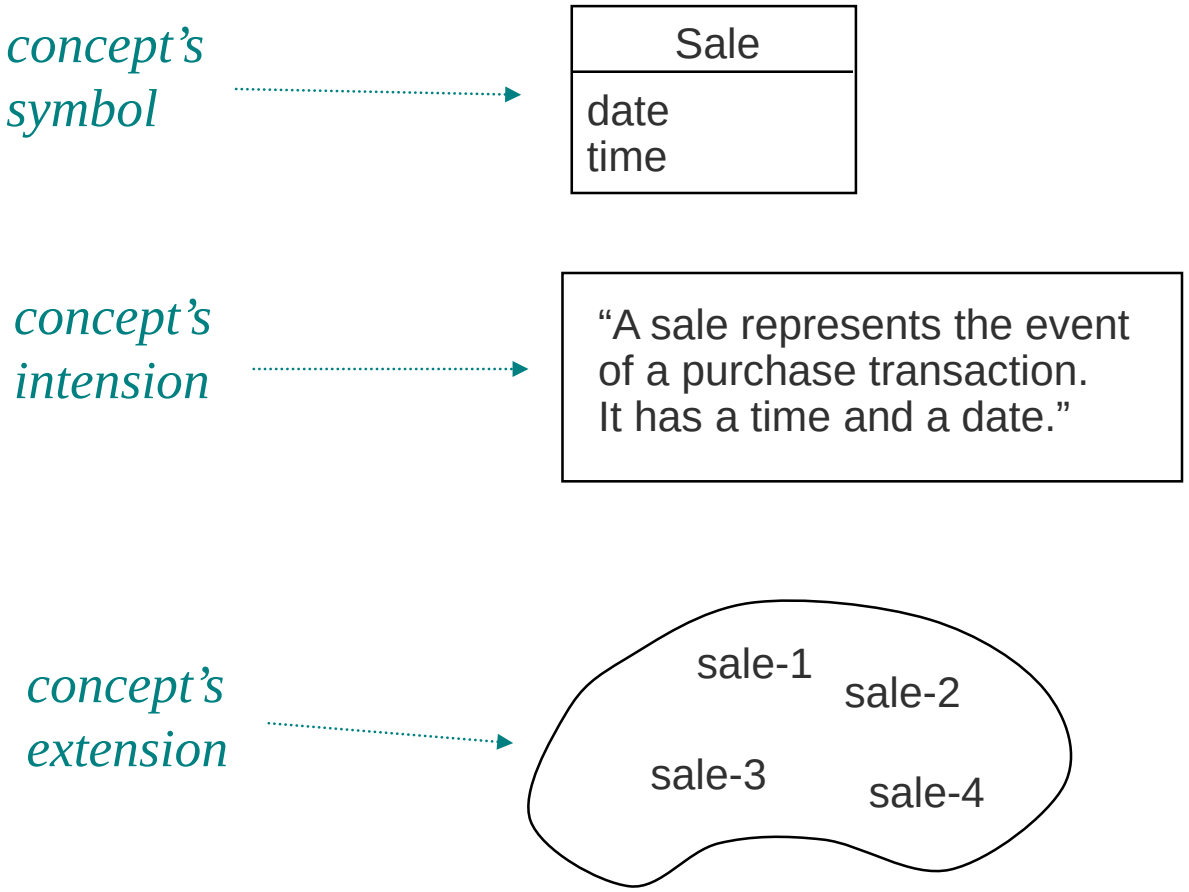
s = start
r = refine

Conceptual Classes

- Informally a conceptual class is an idea, thing or object
- More formally, a conceptual class comprises

1. **Symbol**
   - words or images representing a conceptual class
2. **Intension**
   - the definition of a conceptual class
3. **Extension**
   - the set of examples to which the conceptual class applies

Conceptual Class Example



Conceptual Class Identification

- The goal of domain modeling is to create a model of interesting or meaningful classes in the domain of interest
- Conceptual classes can be attributeless
  - have a purely behavioral role
- Three practical techniques for identifying conceptual classes
  1. reuse or modify existing models (**analysis patterns**)
  2. using a conceptual class category list
  3. identifying noun phrases

Conceptual Class Category List

Conceptual Class Category	Example
Physical or tangible objects	*Register, Airplane*
Specifications, designs or description of things	Product Specification, 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
Computers or machines external to the system	PaymentAuthorizationSystem, AirTrafficControl
Abstract noun concepts	Hunger, Acrophobia
Organizations	SalesDepartment, Airline
Events	Sale, payment, meeting, Flight, Crash, Landing
Processes (not always represented as a concept)	SellingAProduct, BookingASeat
Rules and policies	RefundPolicy, CancelationPolicy
Catalogs	ProductCatalog, PartsCatalog
Records of finances, work, contracts	Receipt, Ledger, EmploymentContract, Log
Financial instruments and services	LineOfCredit, Stock
Manuals, documents, references, papers, books	DailyPriceChangeList, RepairManual

Noun Phrase Identification

- A simple technique for identifying conceptual classes is based on the **linguistic analysis** of textual descriptions of the domain
  - identify the nouns and noun phrases and regard them as candidate conceptual classes or attributes
- Cannot be applied mechanically since words in natural language are ambiguous
  - essentially provides a source of inspiration
- **Fully dressed use cases** are an excellent basis to perform this kind of analysis

Noun Identification Example (1/2)

Main Success Scenario (or Basic Flow):

1. Customer arrives at 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 3-4 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 completed sale and sends sale and payment information to the external
   Accounting system (for accounting and commissions) and Inventory system (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.

Noun Identification Example (2/2)

Main Success Scenario (or Basic Flow):

1. ***Customer arrives*** at ***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 3-4 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 completed ***sale*** and sends sale and payment information to the external ***Accounting*** system (for ***accounting*** and ***commissions***) and ***Inventory*** system…
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.

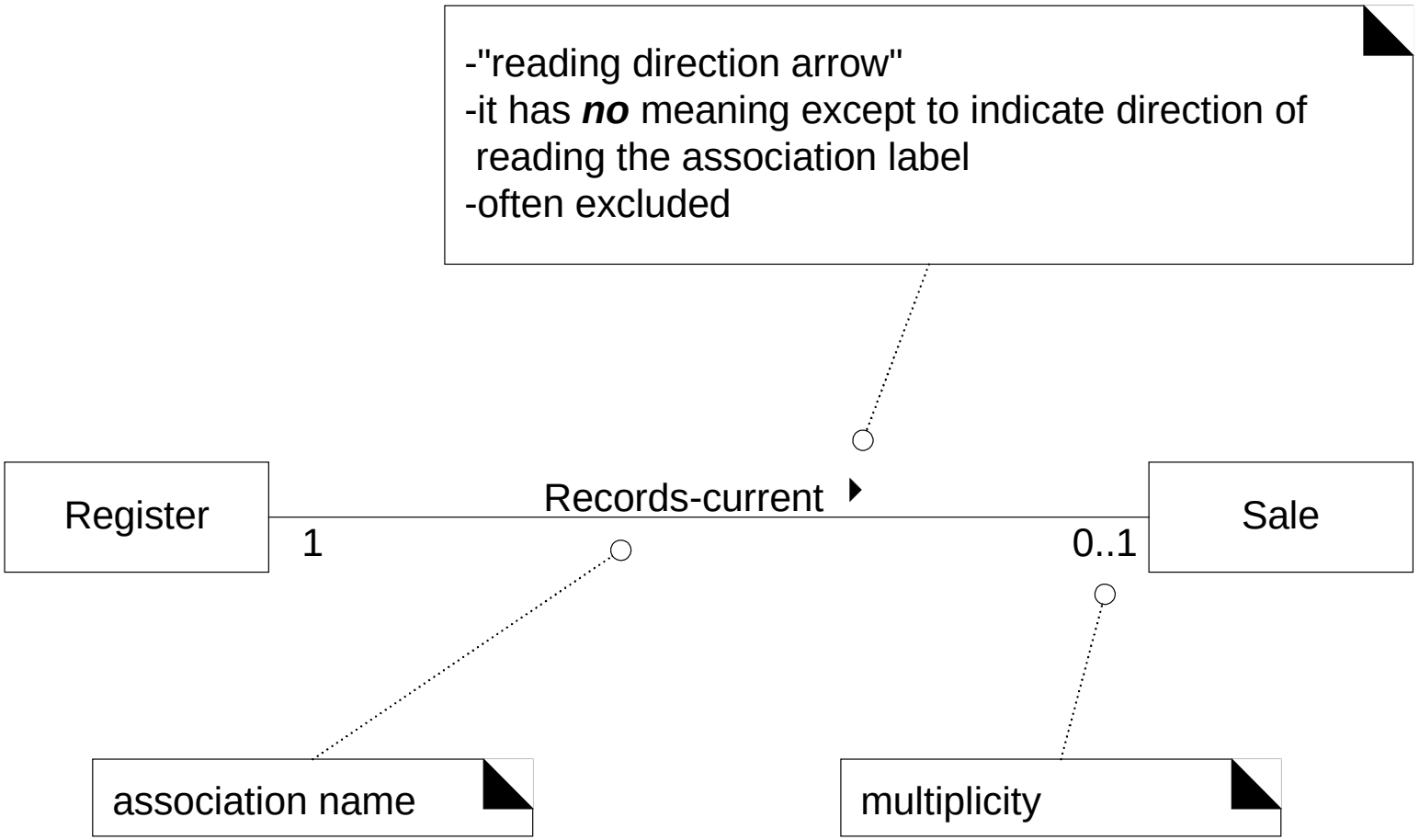
Candidate Conceptual Classes

- From the conceptual class category list and noun phrase analysis a list of candidate conceptual classes is generated
  - the list should be constrained to the requirements and simplifications currently under consideration
- There is no such thing as a “correct” list
  - A somewhat arbitrary collection of abstractions and domain vocabulary

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

Associations

Def: An **association** is a relationship between classes indicating some meaningful and interesting connection between instances of the classes



Criteria for Useful Associations

- On a domain model with **n** conceptual classes there are potential **n(n-1)/2** associations
  - many of these are "visual noise" recording no useful information
- Associations worth noting usually imply knowledge of a relationship that needs to be preserved for some duration
  - could be milliseconds or years depending on the context
- **Candidate associations**
  - associations for which knowledge of a relationship needs to be preserved for some duration ("need to know" associations)
  - associations from the Common Associations List

Common Associations List (1/2)

Category	Examples
A is a physical part of B	Drawer—Register Wing—Airplane
A is a logical part of B	SalesLineItem—Sale FightLeg—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 MaintenanceJob—Maintenance
A is known/logged/recorded/reported/captured in B	Sale—Register Reservation—FlightManifest
…..	…..

Common Associations List (2/2)

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

Representing Associations in UML

- Associations are inherently **bi-directional**
  - from instances of either class, logical traversal to the other is possible
  - such a traversal is purely abstract
    - it is not a statement about connections between software entities

- The end of an association may contain a **multiplicity** expression
  - indicating the numerical relationship between instances of the classes

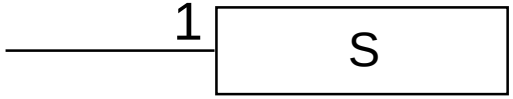
- An optional **"reading direction arrow"** indicates the direction in which the association should be read
  - it does not indicate direction of visibility or navigation
  - it has no meaning in terms of the model but is only an aid to the reader

Multiplicity…

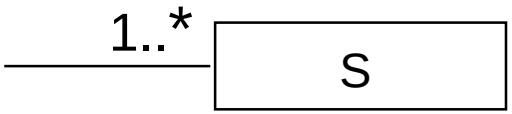
- …defines how many instances of class A can be associated with one instance of class B

- …indicates how many instances can be validly linked to another **at a particular moment** rather than over a span of time
  - a person can be married to several other people during the course of their lifetime, but at any one time can be married to only one
- …communicates a domain constraint that could be reflected in software

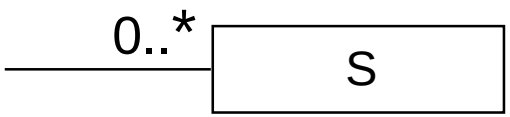
Multiplicity Values



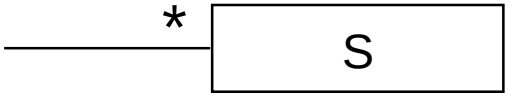
exactly one



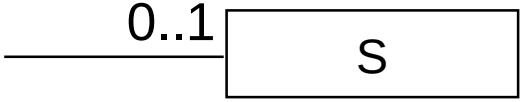
mandatory (one or more)



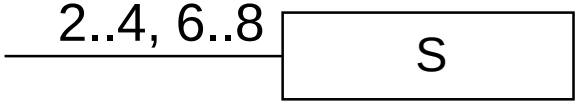
many (zero or more)



many (zero or more)



optional (zero or one)

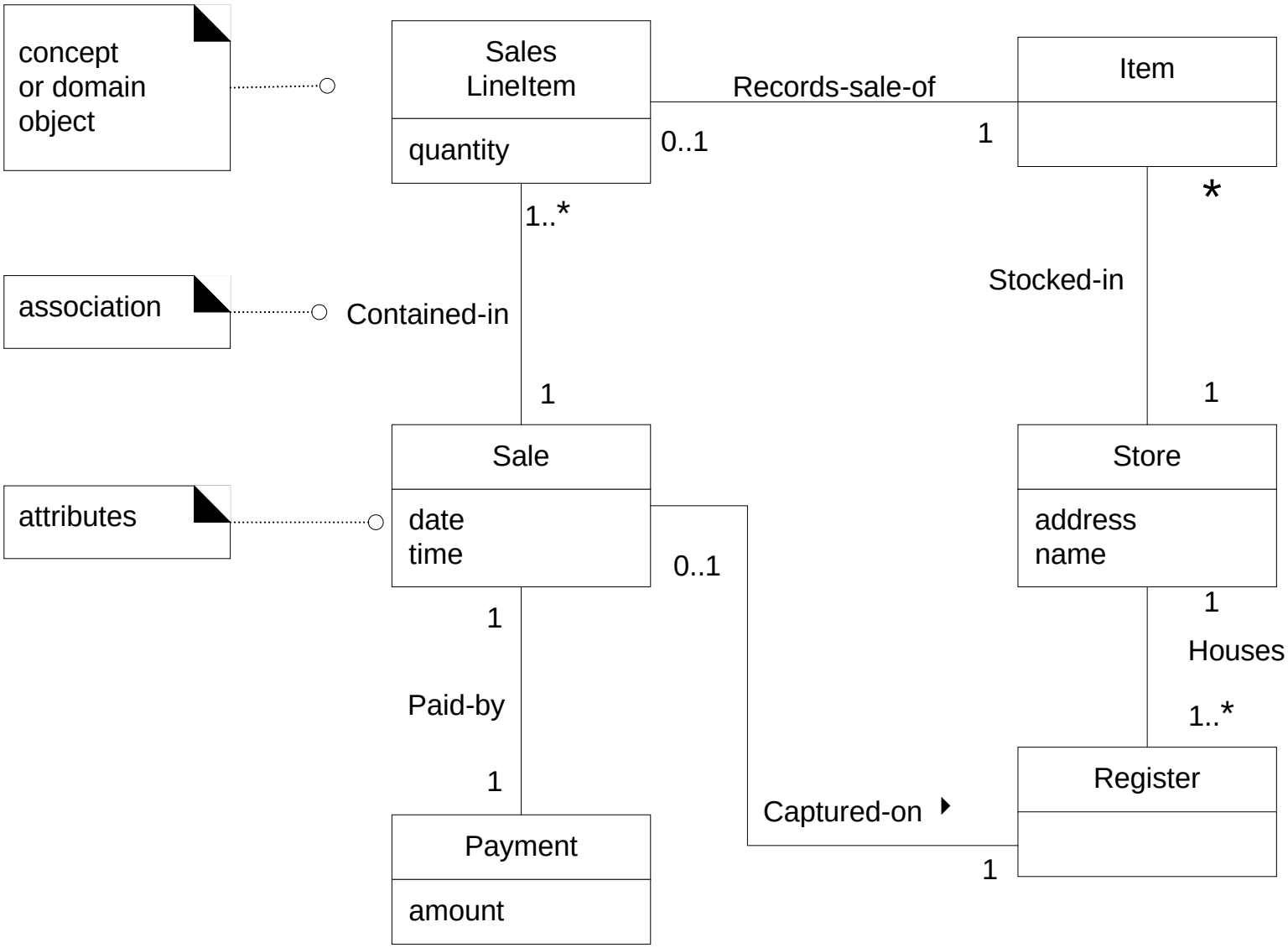


numerically specified

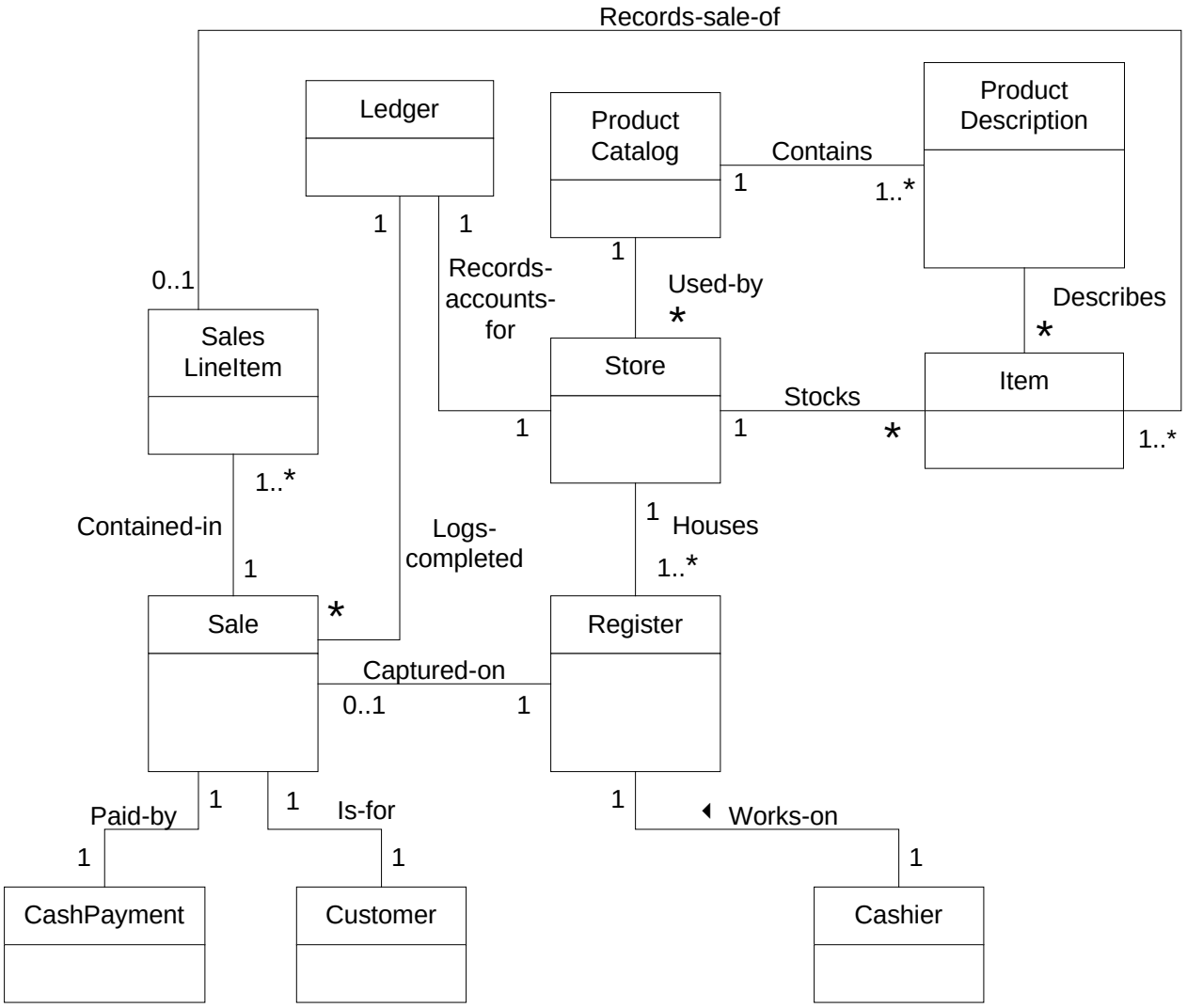
Associations Checklist Applied to POS System

Category	Examples
A is a physical part of B	Register-CashDrawer
A is a logical part of B	SalesLineItem—Sale
A is physically contained in/on B	Register—Store, Item—Store
A is logically contained in B	ProductSpecification—ProductCatalog ProductCatalog—Store
A is a description for B	ProductSpecification—Item
A is a line item of a transaction or report B	SalesLineItem—Sale
A is known/logged/recorded/reported/captured in B	Sales—Store Sale—Register
A is a member of B	Cashier—Store
A uses or manages B	Cashier—Register Manager—Register
A communicates with B	Customer—Cashier
A is related to a transaction B	Customer—Cashier Cashier—Payment
A is a transaction related to another transaction B	Payment—Sale
A is next to B	SalesLineItem—SalesLineItem
A is owned by B	Register—Store

POS System Partial Domain Model



POS System Domain Model (Initial Version)

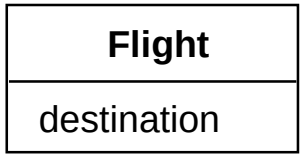


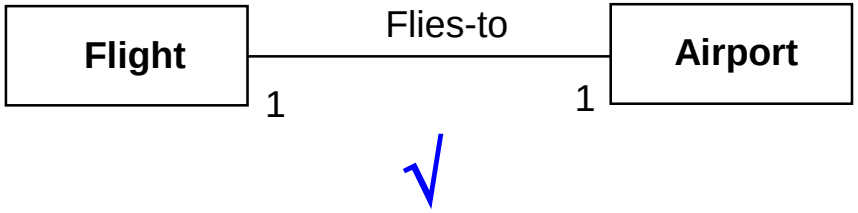
Attributes

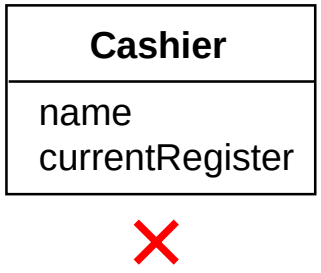
- An attribute is a logical data value of an object
- Include the following attributes in a domain model
  - those for which the requirements (e.g., use cases) suggest or imply **a need to remember information**
- There are some things that should not be represented as attributes, but rather as associations
  - namely when domain objects **have an identity**

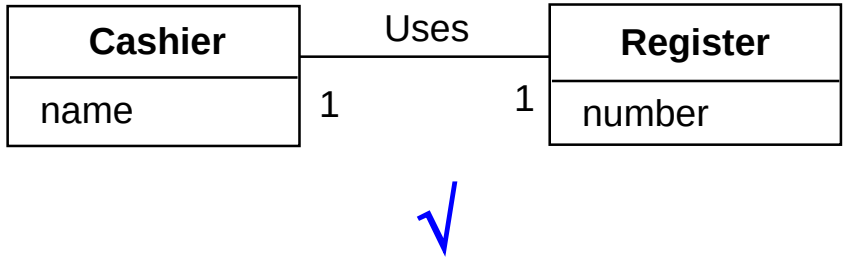
Do Not Model Relationships as Attributes!

- A common confusion is modeling complex domain concepts as attributes
  - e.g., a destination is not simply a string, it is a complex thing that occupies many square kilometers of space
    - therefore, Flight should be related to Airport via an association
  - **relate conceptual classes with associations, not attributes!**









Keep Attributes Simple

- Intuitively, simple attributes are generally thought of as primitive data types such as numbers
  - the type of an attribute should normally **not** be a complex domain concept, such as a Sale or Airport
  - the attributes in a domain model should generally be **data types**

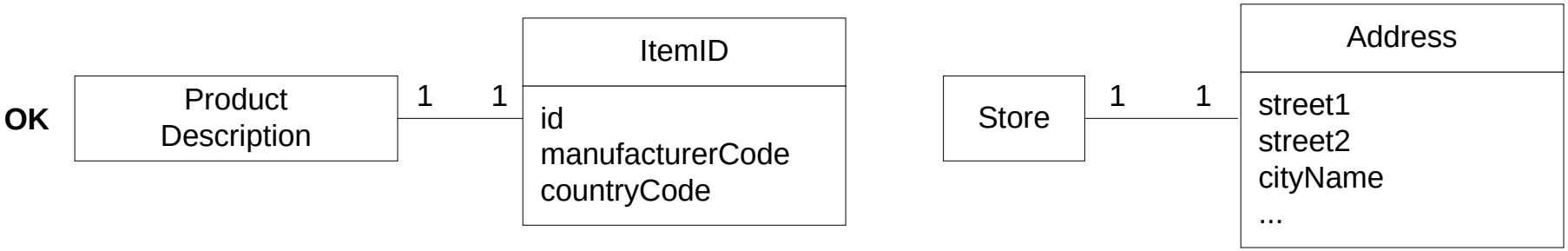
- **Def: Data types** (in UML)
  - A set of values for which **unique identity is not meaningful** (in the context of our model or system)
  - i.e., equality tests are *not* based on identity, but instead on value
  - Examples
    - Separate instances of the *Integer* 5
    - Separate instances of the *String* „cat“
    - Separate instances of the *Date* „June 1st, 1990“

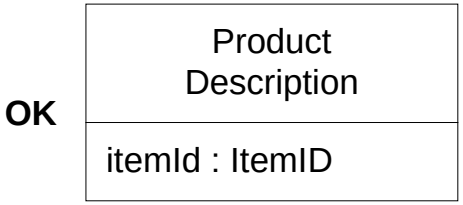
Common Data Types

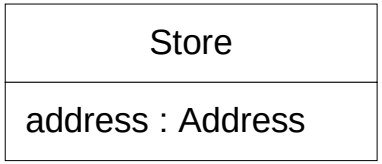
- Primitive types
  - Number, Boolean, Character, String (text), Date, Time
  - Enumerations (e.g., Size = [small, large])
- Other common types include
  - Date, Time
  - Address
  - Color
  - Geometrics (Point, Rectangle)
  - Phone Number
  - Social Security Number
  - Universal Product Code (UPC)
  - SKU
  - ZIP or postal codes

Data Type Classes

- Create an own **data type** for everything that –
  - consists of multiple parts, but has no identity
  - has quantity and a unit (e.g., currency)
- Two ways to indicate a data type attribute of an object



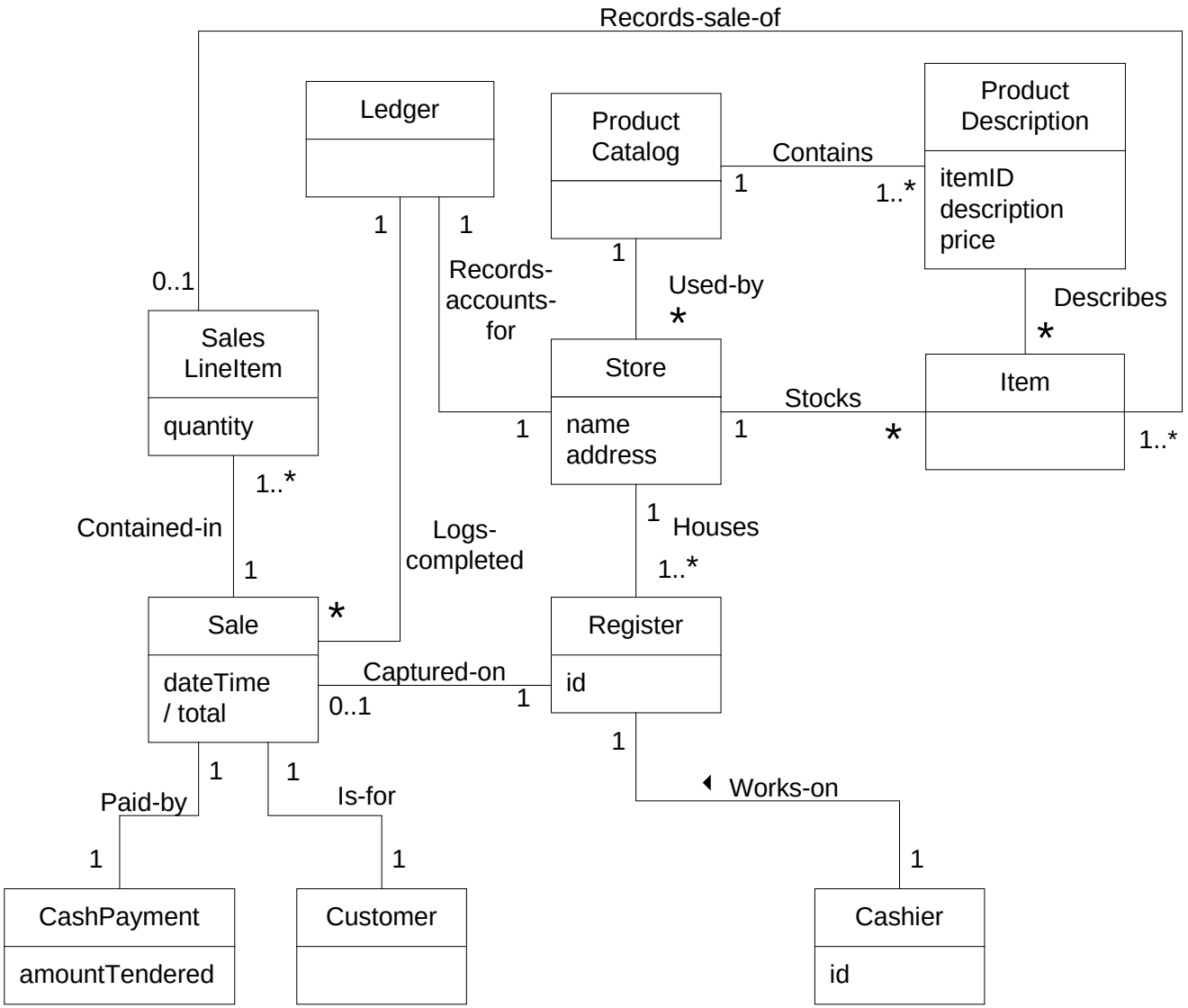




Attributes Indicated by ProcessSale Scenario

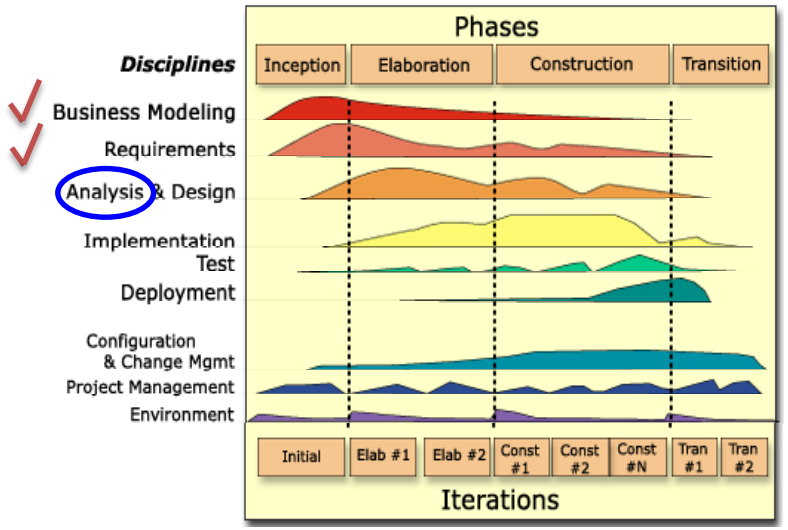
- **Payment**
  - **amount** - to determine if sufficient payment was provided and to calculate change, an amount (a.k.a amount tendered) must be captured
- **ProductSpecification**
  - **description** - to show the description on a display or a receipt
  - **id** - to look up a ProductSpecification, given an entered itemID, it is necessary to relate them to an id
  - **price** - to calculate the sales total and show the line item price
- **Sale**
  - **date, time** - a receipt is a paper of sale which normally shows date and time of sale
- **SalesLineItem**
  - **quantity** -to record the quantity entered when there is more than one item in the line item sale (e.g. 5 packets of crisps)
- **Store**
  - *address, name* - the receipt requires the name and address of the store

POS System Domain Model (Final Version)



Concluding Remarks

- There is no such thing as a single correct domain model
  - models are approximations of the domain we are trying to understand
- A good model captures essential abstractions and information
  - required to understand the domain in the context of the requirements
- It also aids people in understanding the domain
  - its concepts, terminology and relationships
- *Outlook: Analysis*



Appendices

User Stories Stock Manager

Nr.	User Story
1	Ich als Bankkunde möchte Aktien (an der Mannheimer Börse) kaufen können, um mein Depot zu vergrößern.
2	Ich als Bankkunde möchte Aktien verkaufen können, um mehr Kapitial verfügbar zu haben.
3	Ich als Bankkunde möchte ein Depot mit dem System anlegen können, um Aktien darüber zu handeln.
4	Ich als Bankkunde möchte den Inhalt meines Depots ansehen können, um mich über seinen Wert zu informieren.
5	Ich als Bankkunde möchte mich über aktuelle (und historische) Aktienkurse und Performance-Werte informieren können, um Kauf/Verkaufsentscheidungen zu treffen.
6	Ich als Bankkunde möchte eine "Watchlist" mit Aktien verwalten können, um bei interessanten Kursveränderungen vom laufenden Programm informiert zu werden.
7	Ich als Bankkunde möchte einen Depotauszug drucken können, um seinen aktuellen Stand "schwarz auf weiß" vor mir zu haben.
8	Ich als Bankkunde möchte den Inhalt meines Depots exportieren können, um die Daten in einer Tabellenkalkulation weiterzuverarbeiten.
9	Ich als Bank möchte, dass das System mit verschiedenen Depots umgehen kann, damit es von mehreren Bankkunden genutzt werden kann.
10	Ich als Bankkunde möchte mit dem System chart- und bewertungstechnische Analysen durchführen können, um fundiertere Kauf/Verkaufsentscheidungen treffen zu können.
11	Ich als Bankkunde möchte vom System Kauf- bzw. Verkaufsempfehlungen erhalten, um wichtige Trends nicht zu verpassen (optional lässt sich dieser Punkt zu einem automatisierten "Trading-Bot" erweitern).

State Models

- Complex domain objects may have user recognizable states
  - that need to be supported by the software objects later
    - in other words, operations should behave differently if the object's state is different
      - e.g., consider a task/ticket management system

- UML state charts can be used to capture this