Modelling using UML Class Diagrams

Let's revisit Process Sale Use case

Preconditions: Cashier is identified and authenticated on a sales terminal.

Main 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 sales 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, requests payment.
7. Customer pays and System handles payment.
8. System logs completed sale and sends sale and payment information to the external Accounting System and Inventory System.
9. System presents receipt.
10. Customer leaves with receipt and goods (if any).

Alternative flow:

3a. Customer asks cashier to remove an item from the purchase.
   1. Cashier enters item identifier for removal from sale.
   2. System displays updated running total.

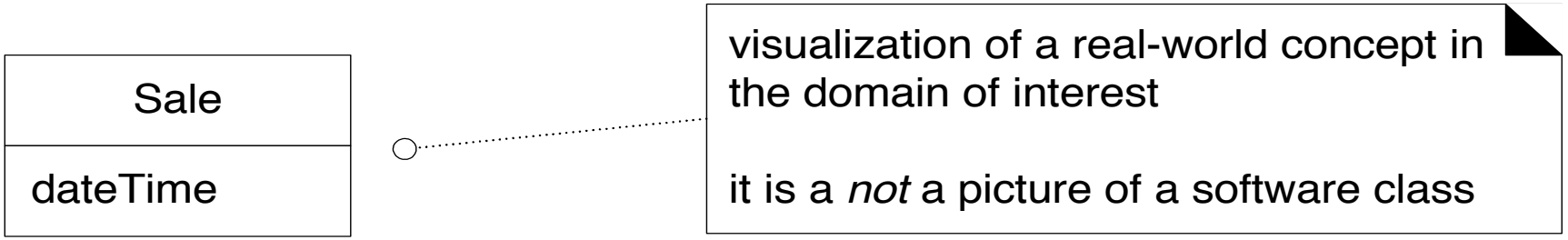
Moving from Use cases to solution design

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 ***sales 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, requests ***payment***.
7. Customer pays and System handles payment.
8. System logs completed ***sale*** and sends sale and payment information to the external ***Accounting*** System and ***Inventory*** System.
9. System presents ***receipt***.
10. Customer leaves with receipt and goods (if any).

Conceptual Classes

Note: These are not software classses

Representing Problem/Domain Concepts using class diagrams



Requirements class vs. design class

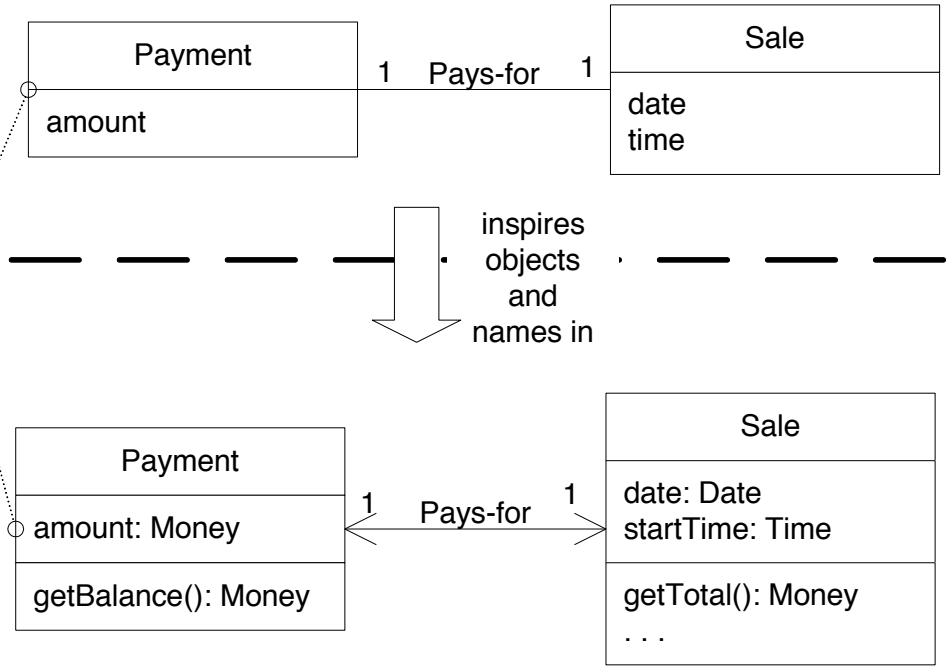
A Payment in the Domain Model is a concept, but a Payment in the Design Model is a software class. They are not the same thing, but the former *inspired* the naming and definition of the latter.

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.

Rendering Concepts

- A depicted class has the following structure:
  - Name compartment (mandatory)
  - Attributes compartment (optional)
- Every class must have a distinguishing name.

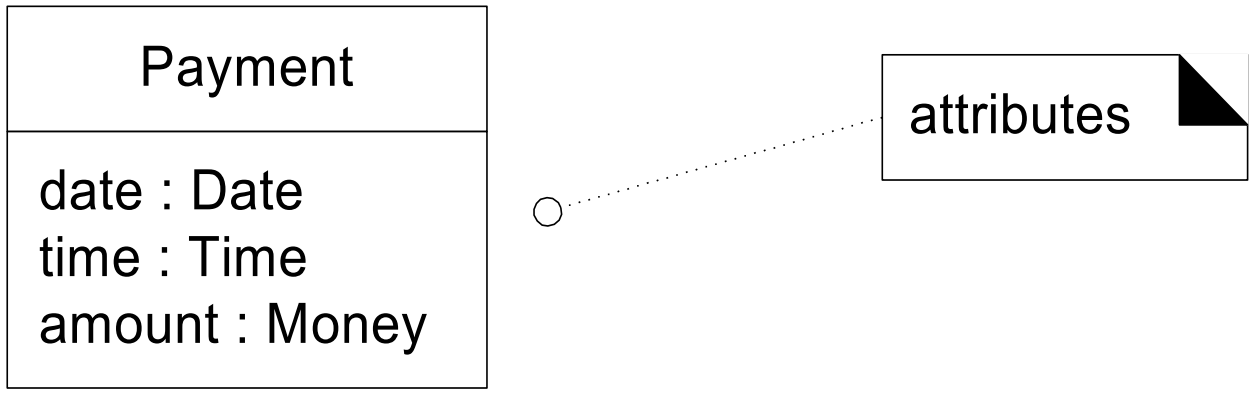
Attributes

- An attribute is a named property. Each concept instance associates value(s) with each attribute of a concept.
- *OOA syntax*

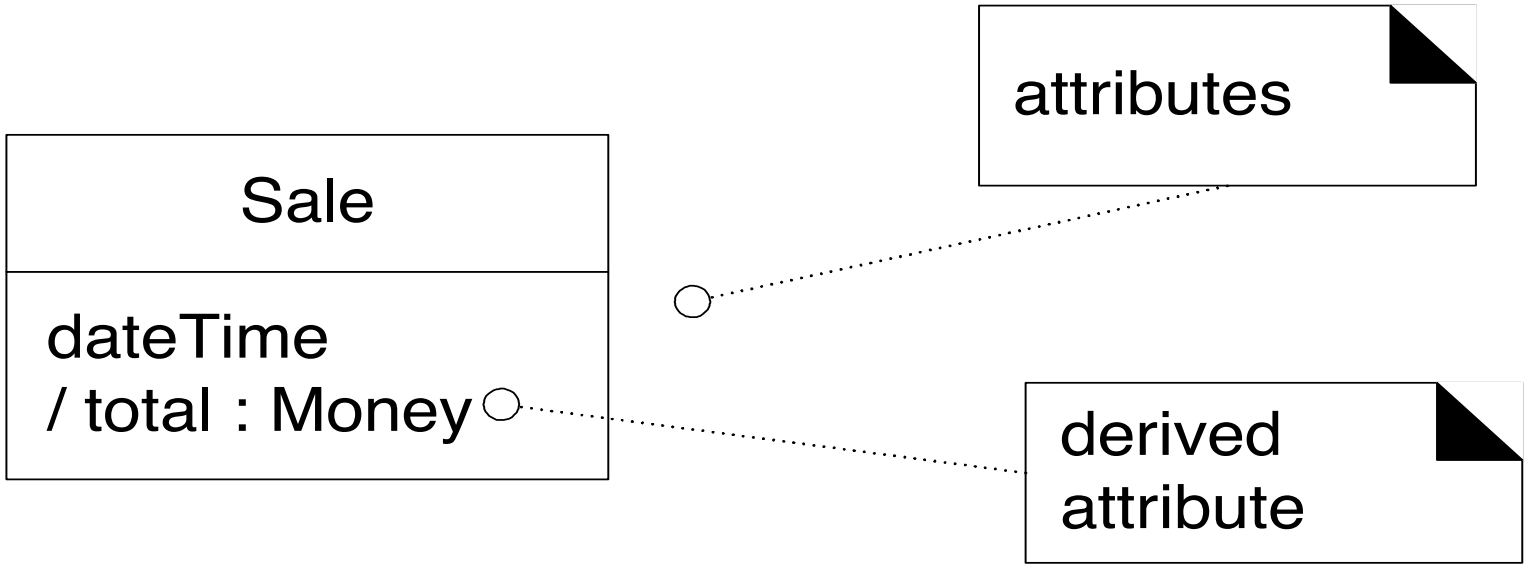
name [multiplicity] [: type] [{property-string}]

Attributes

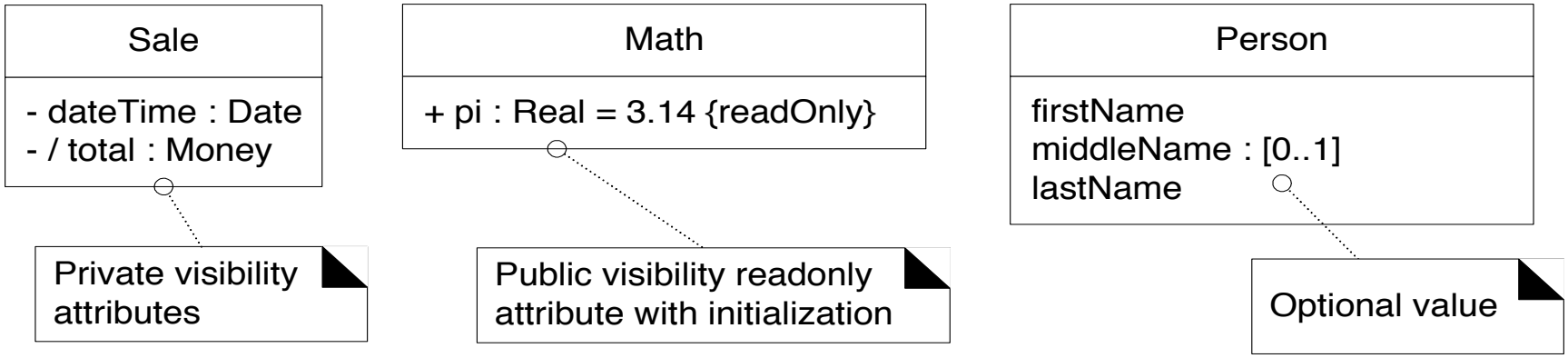
- Show only "simple" relatively primitive types as attributes.
- Connections to other concepts are to be represented as associations, not attributes.



Derived Attributes



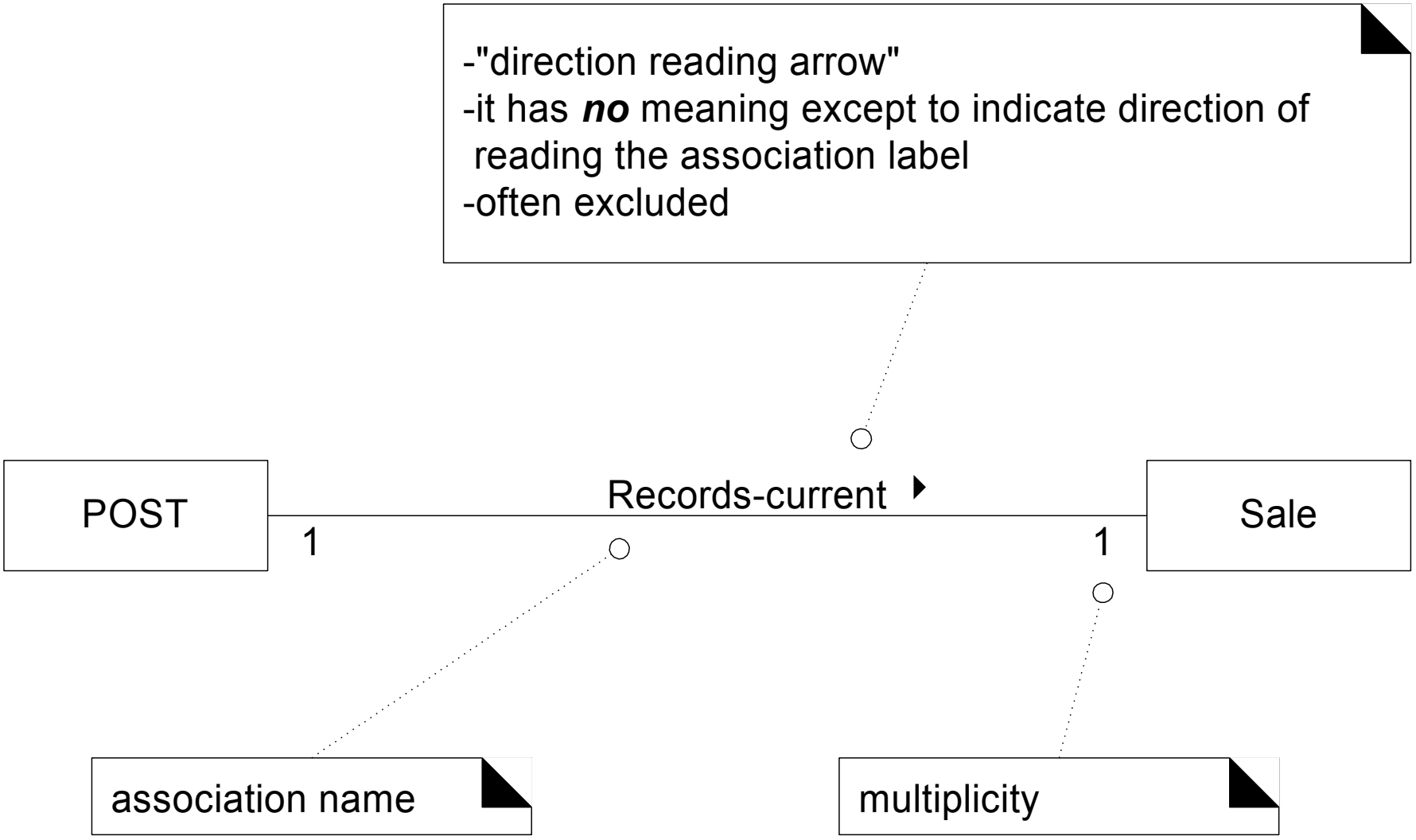
Access Modifiers



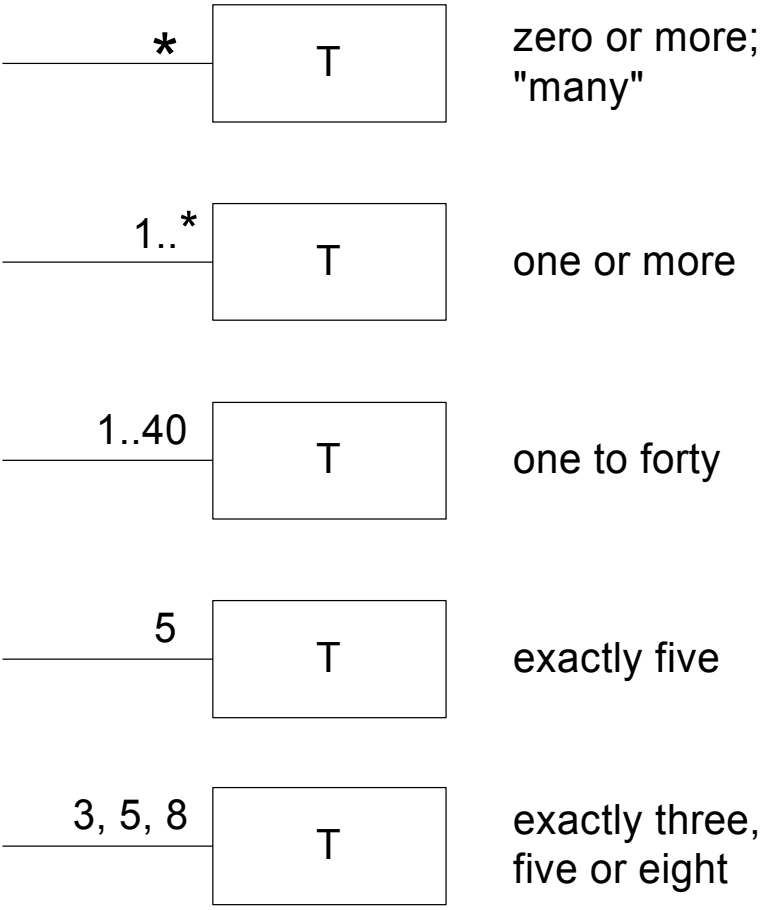
Modeling Static Relationships

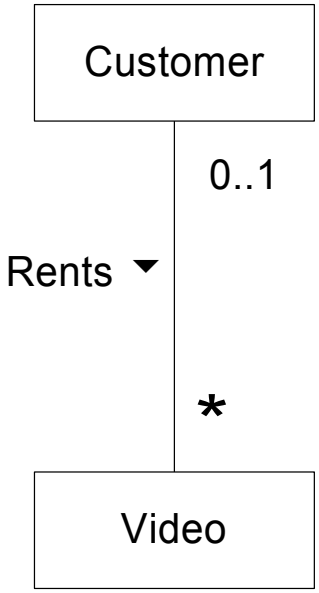
- Two kinds of static relationships:
  - Associations
    - Represent structural relationships among objects
  - Generalizations
    - Represent generalization/specialization class structures
- The two kinds of relationships are orthogonal

Associations



Multiplicity



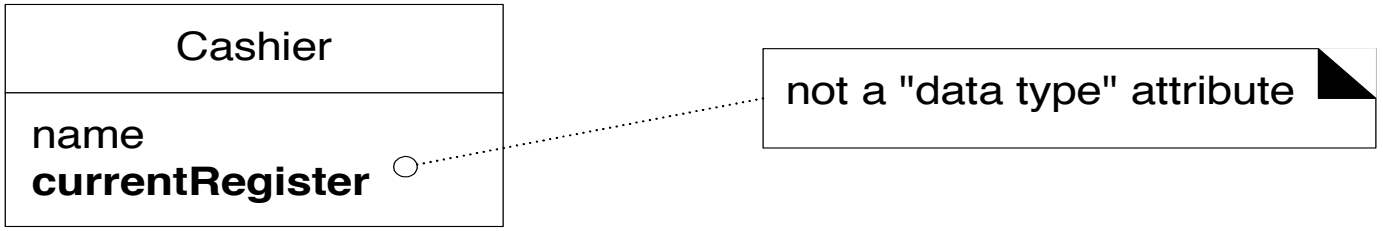


One instance of a Customer may be renting zero or more Videos.

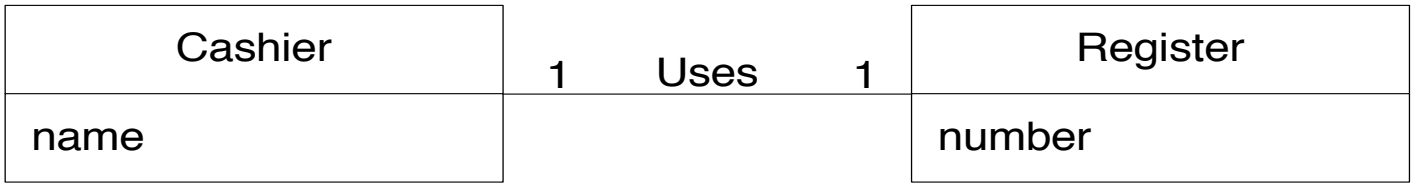
One instance of a Video may be being rented by zero or one Customers.

Association vs. Attribute

Worse

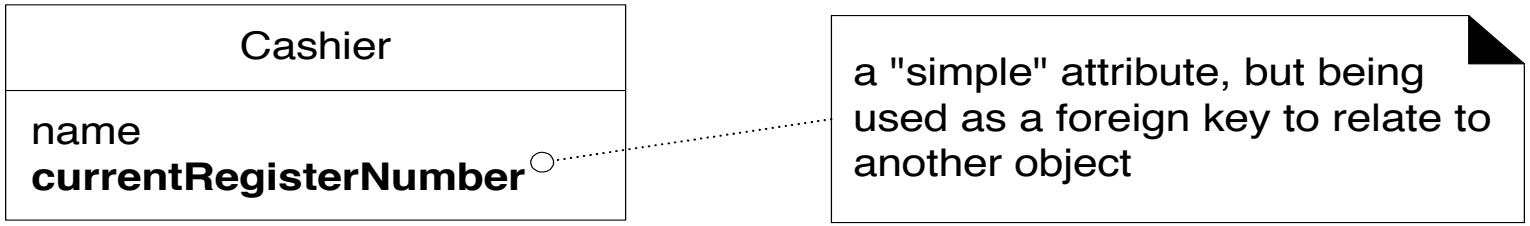


Better

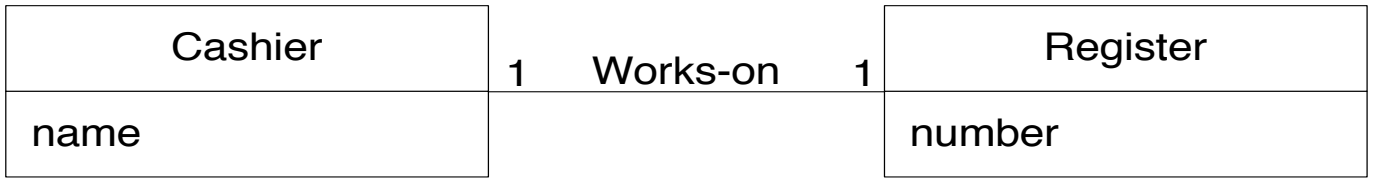


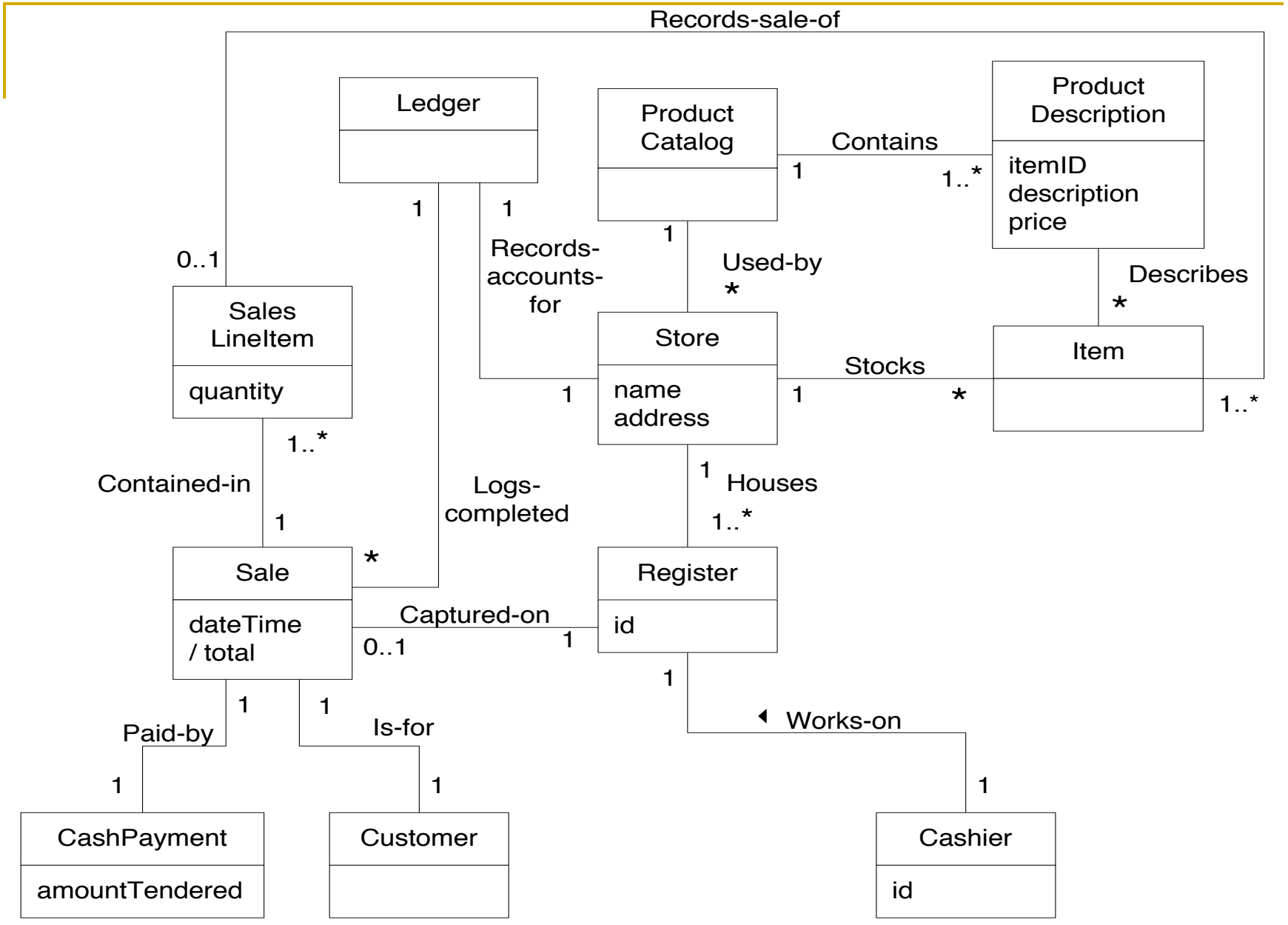
Do not use attributes to represent foreign keys

Worse



Better

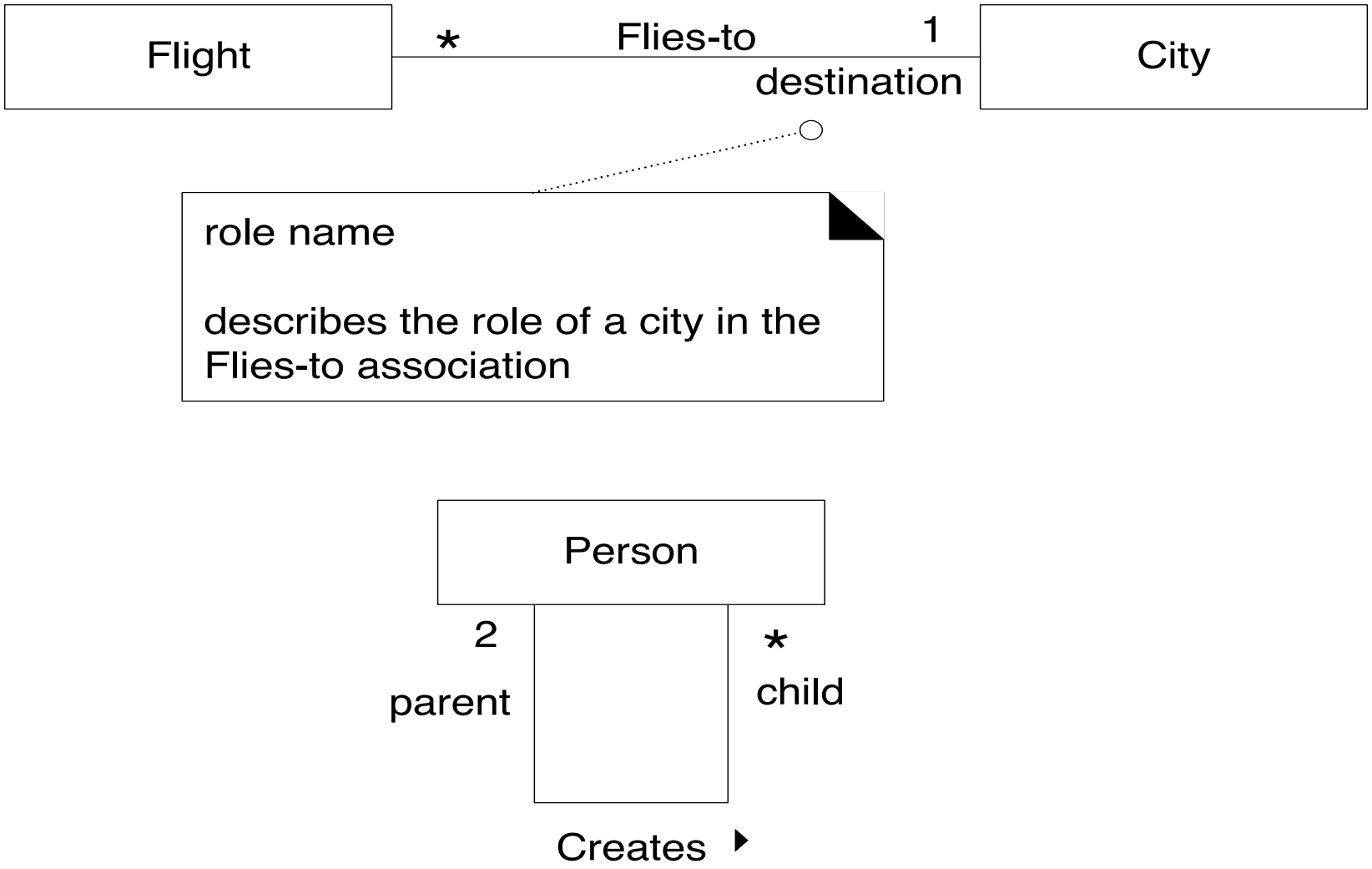



Records-sale-of
Ledger
Product Catalog
Contains
Product Description
itemID
description
price
1
1..*
1
1
1
Records-accounts-for
Used-by
*
Describes
*
0..1
Sales LineItem
quantity
Store
name
address
Item
Stocks
1
1
*
1..*
1..*
Contained-in
1
Logs-completed
1
Houses
1..*
Sale
dateTime
/ total
*
Register
id
Captured-on
0..1
1
1
1
Paid-by
Is-for
Works-on
1
1
1
CashPayment
amountTendered
Customer
Cashier
id

Association Roles

- When a class is part of an association it plays a *role* in the relationship.
- You can name the role that a class plays in an association by placing the name at the class's association end.
- Formally, a class role is the set of objects that are linked via the association.

Examples of Association Roles



Association Constraints

- Users can define constraints on how objects are linked via associations.
- Constraints can be expressed in the *Object Constraint Language* (OCL).

Aggregation

- Aggregation is a special form of association
  - reflect whole-part relationships
- The whole delegates responsibilities to its parts
  - the parts are subordinate to the whole
  - unlike associations in which classes have equal status

UML Forms of Aggregation

- Composition (strong aggregation)
  - parts are existent-dependent on the whole
  - parts are generated at the same time, before, or after the whole is created (depending on cardinality at whole end) and parts are deleted before or at the same time the whole dies
  - multiplicity at whole end must be 1 or 0..1

Composition

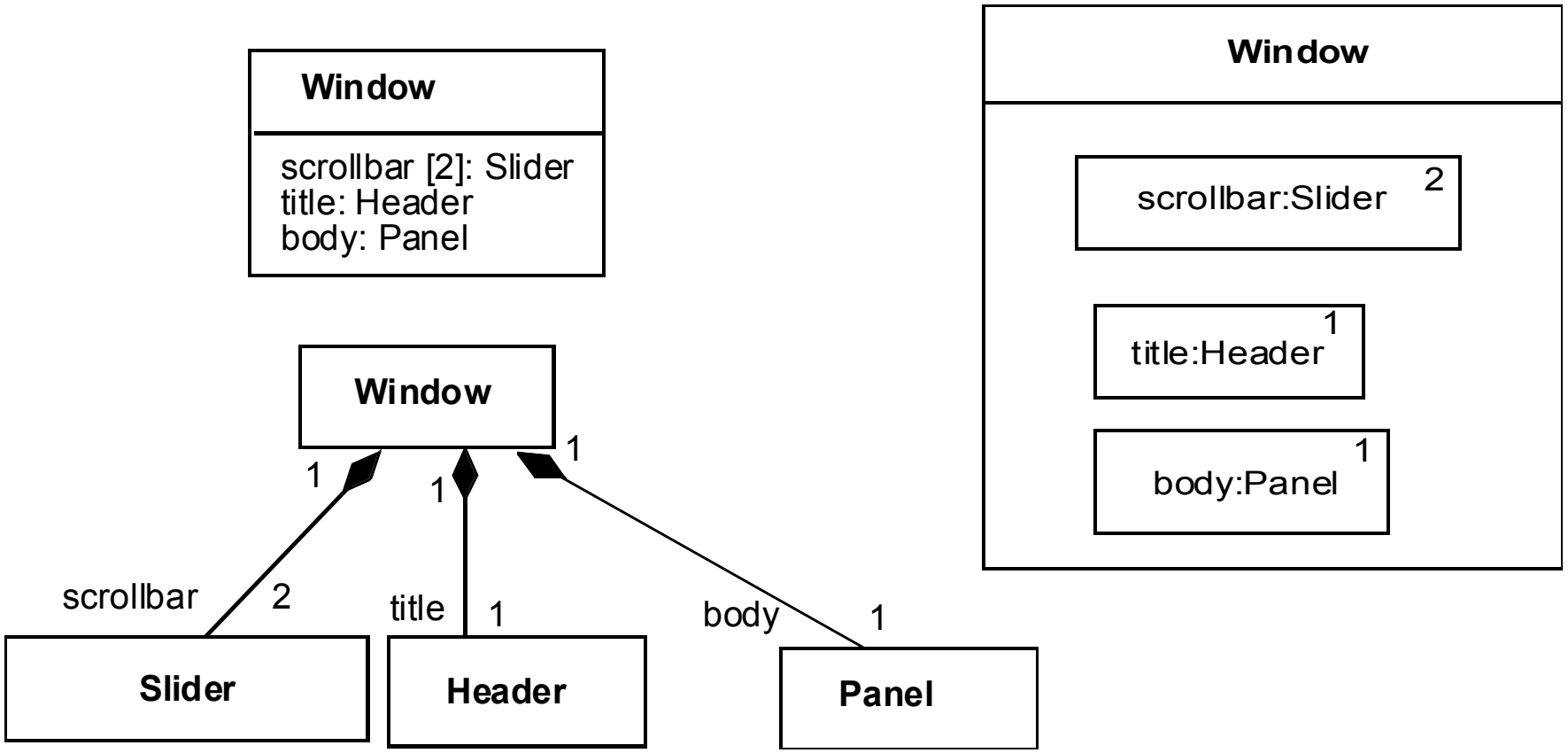


Fig. 3-45, *UML Notation Guide*

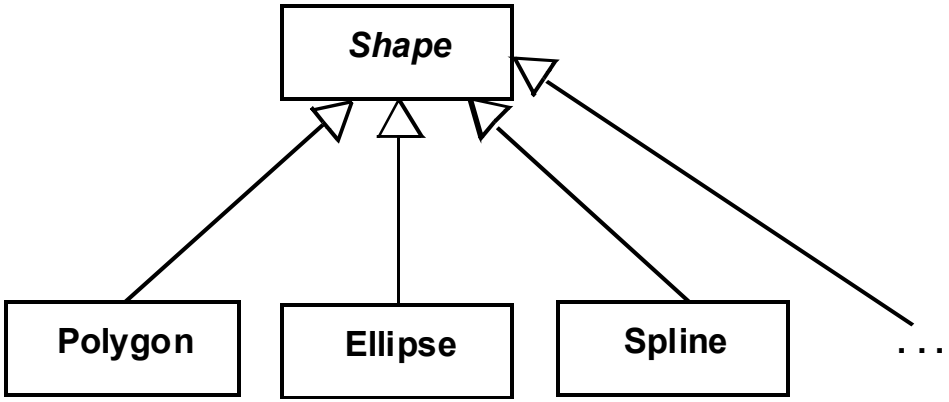
Generalization/Specialization

- A generalization (or specialization) is a relationship between a general concept and its specializations.
  - Objects of specializations can be used anywhere an object of a generalization is expected (but not vice versa).
- Example: *Mechanical Engineer* and *Aeronautical Engineer* are specializations of *Engineer*

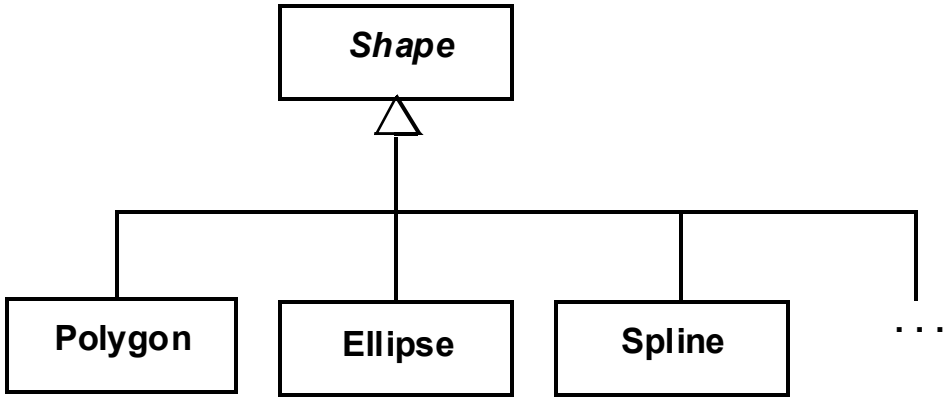
Rendering Generalizations

- Generalization is rendered as a solid directed line with a large open arrowhead.
  - Arrowhead points towards generalization
- A discriminator can be used to identify the nature of specializations

Generalization



Separate Target Style



Shared Target Style

Generalization

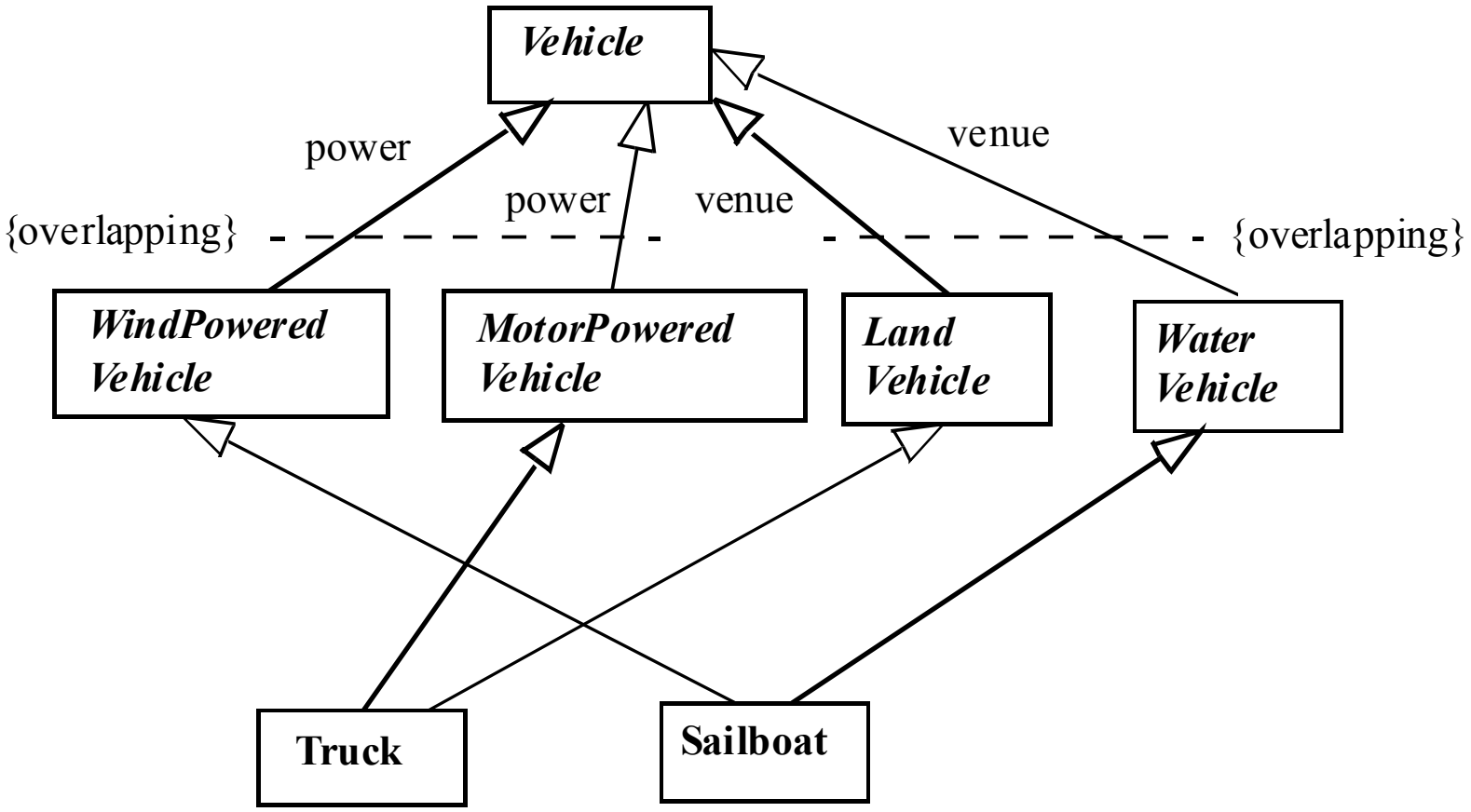
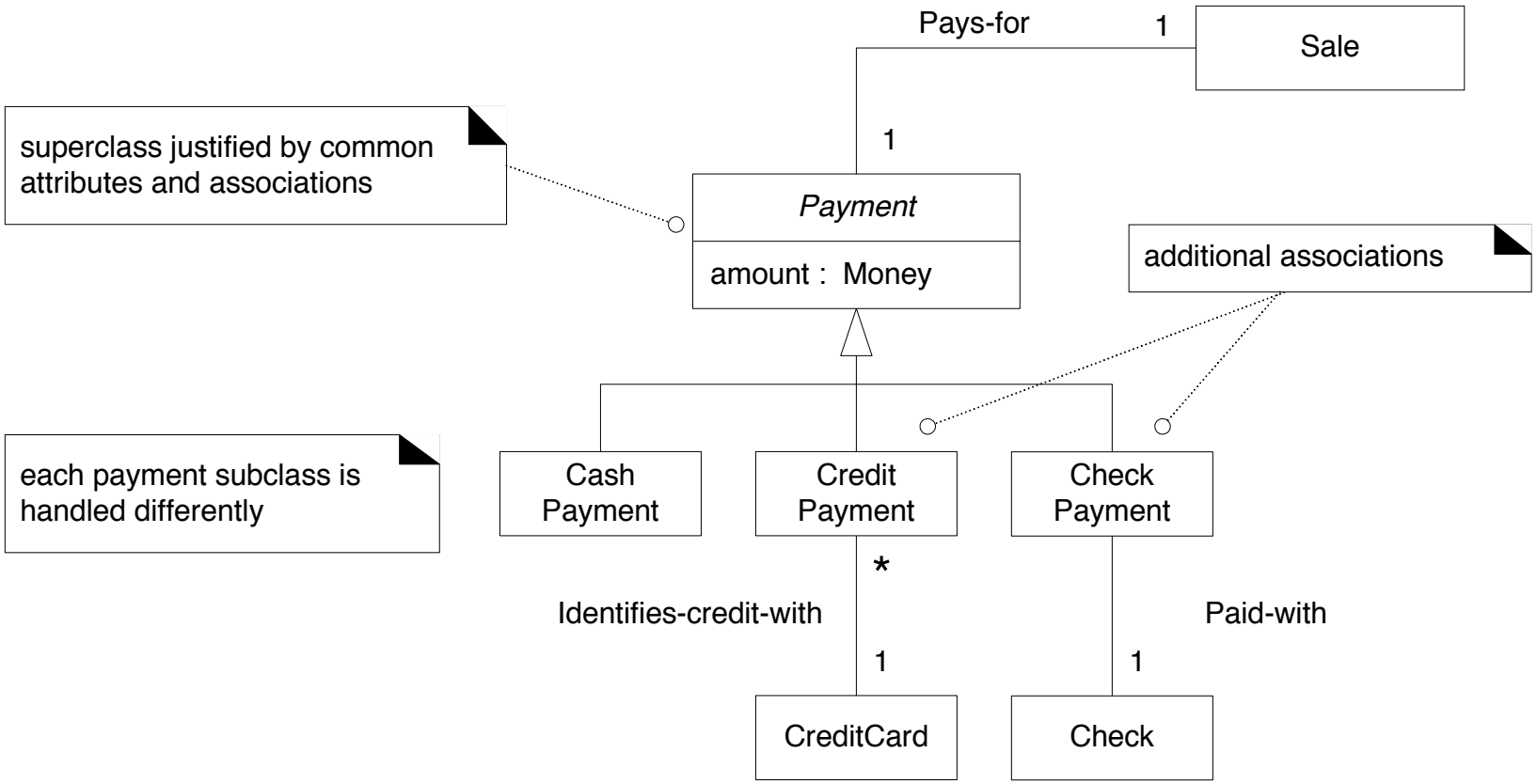
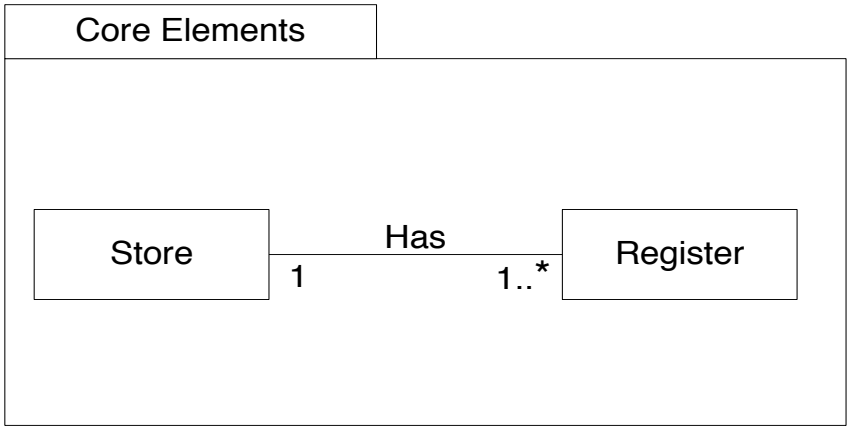


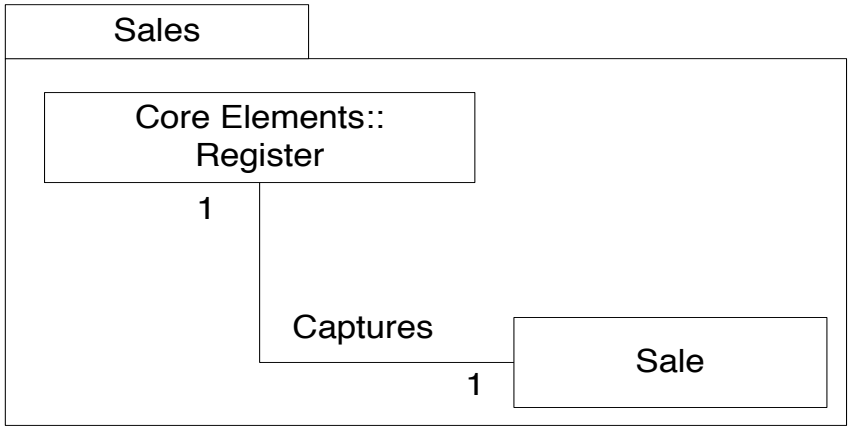
Fig. 3-48, *UML Notation Guide*

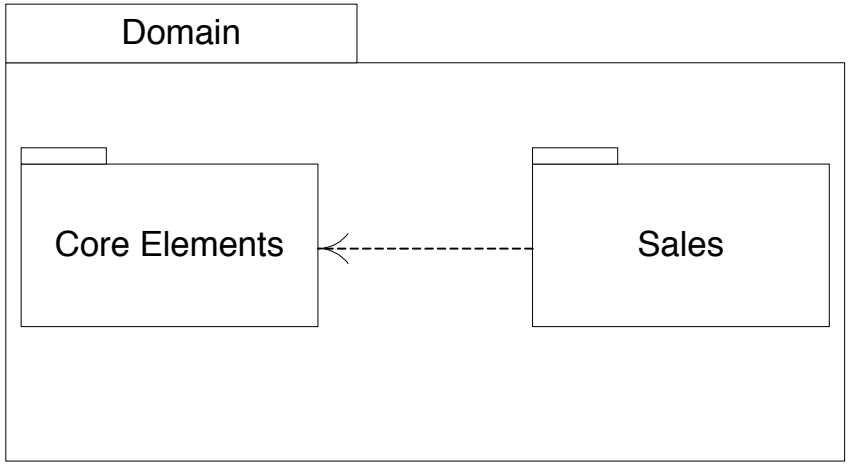
Inheritance of associations

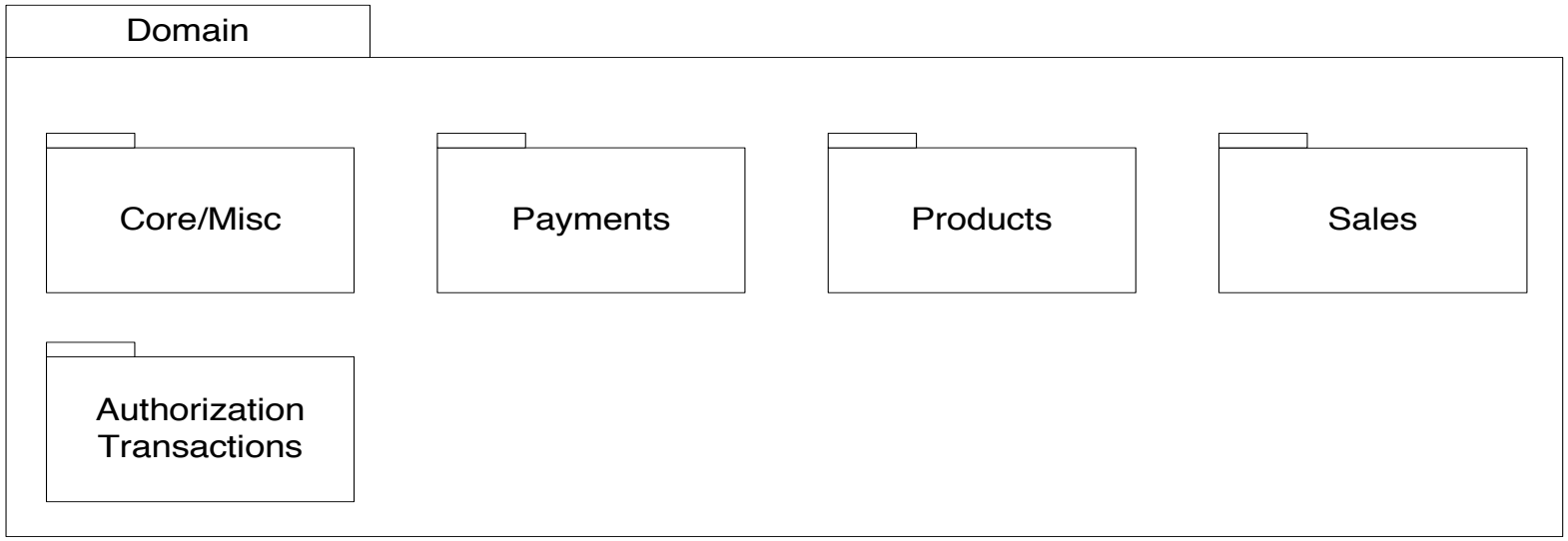


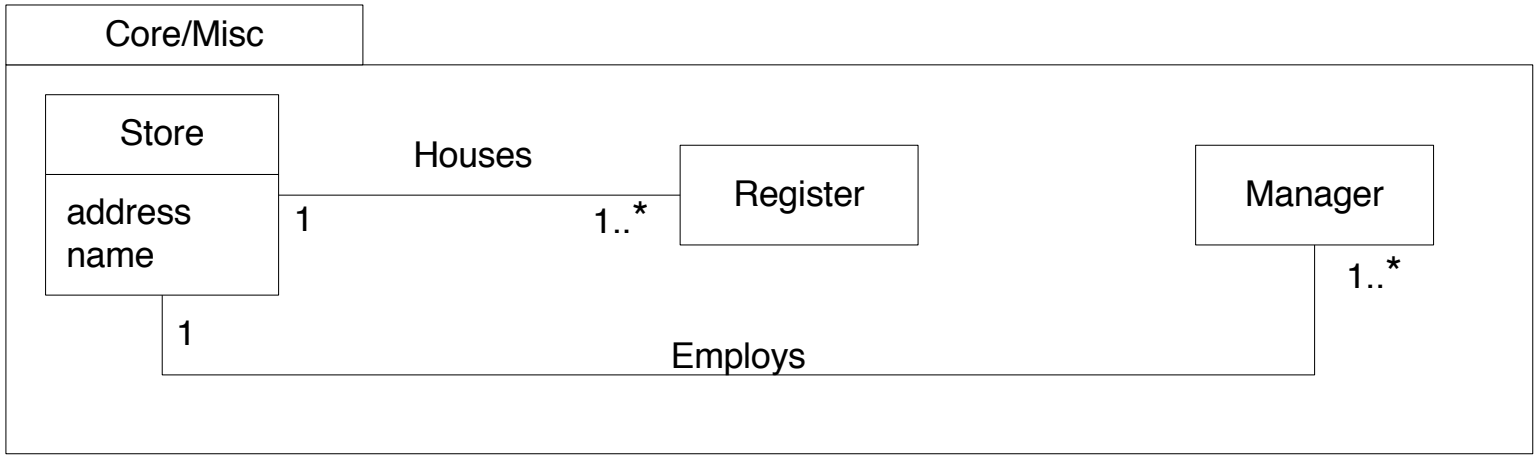
Handling Large Domain Models



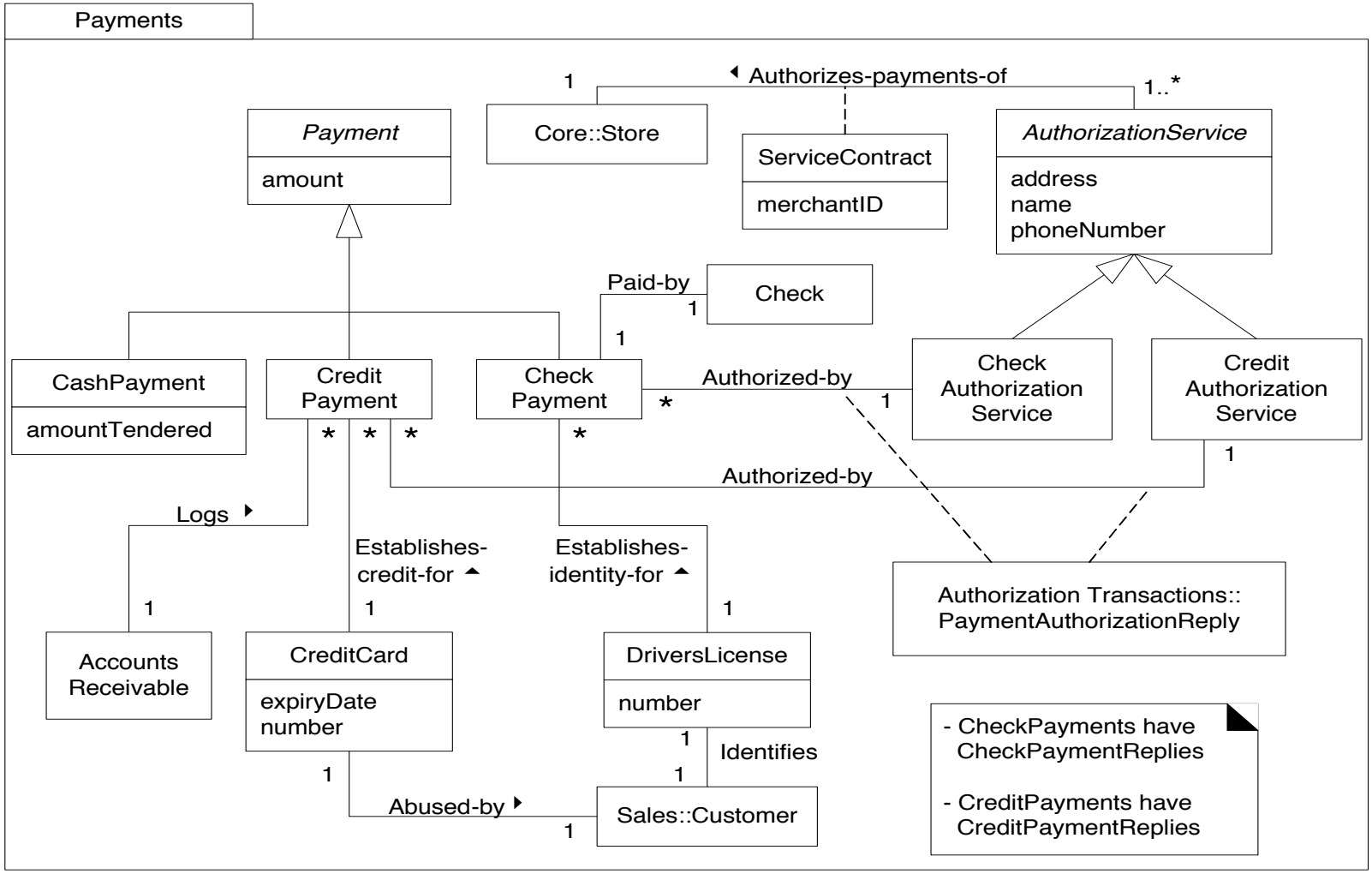




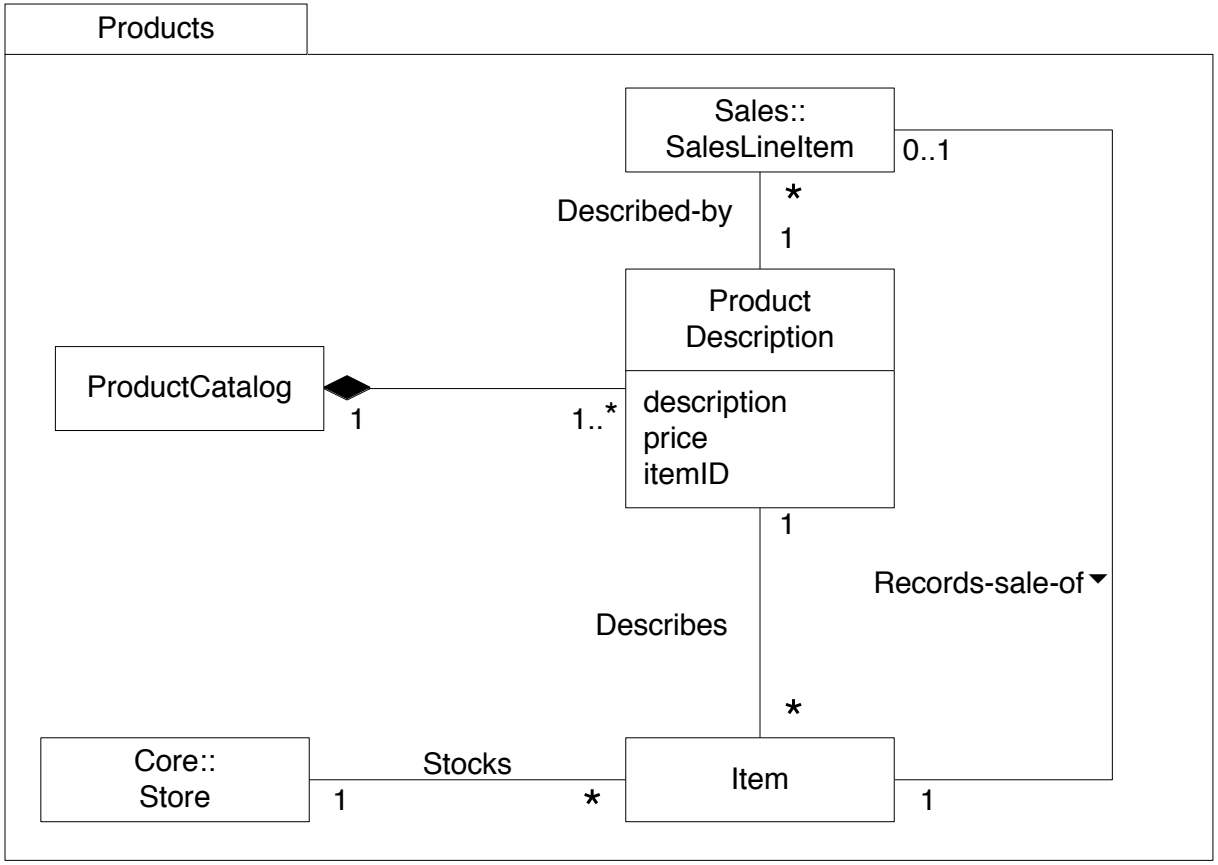

Domain
Core/Misc
Payments
Products
Sales
Authorization
Transactions


Core/Misc
Store
address
name
Houses
1
1..*
Register
Manager
1..*
1
Employs

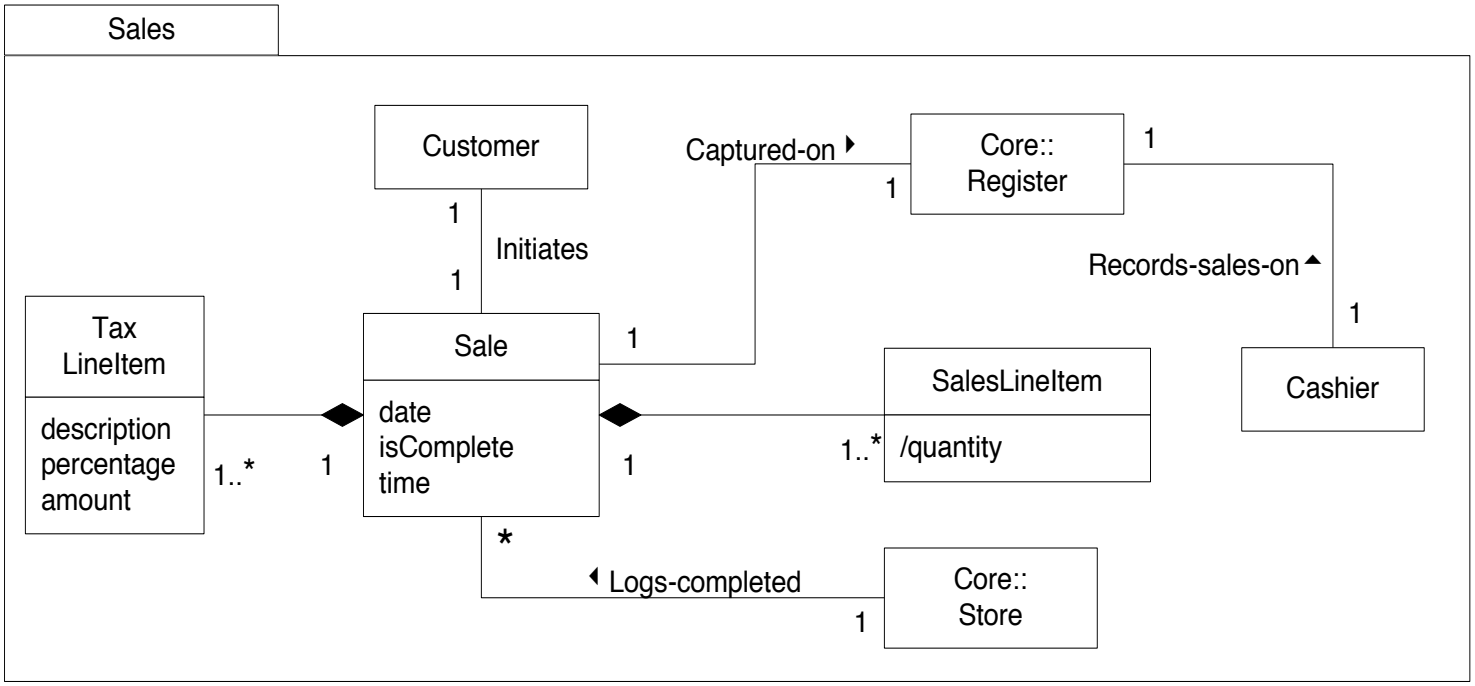
Payments Package



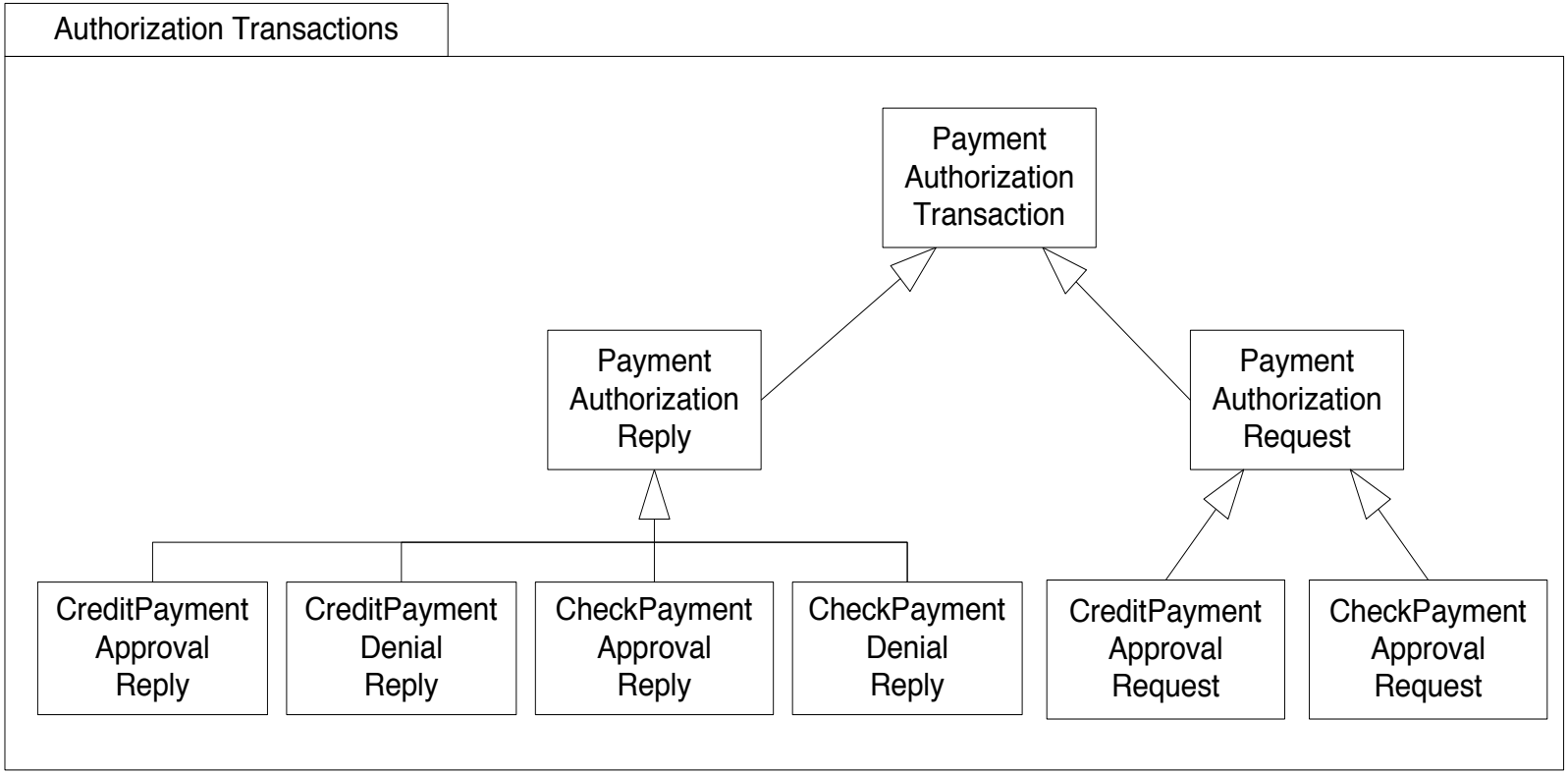
Products Package



Sales Package



Authorization Transactions Package



Summary

- Requirements models are used to represent conceptual elements and their relationships at the problem level
- UML may be used as a specification language. However UML is semi-formal

- How do we write formal specifications?
  - For example, constraints may be added to make it more formal. Object Constraint Language (OCL) may be used for this purpose