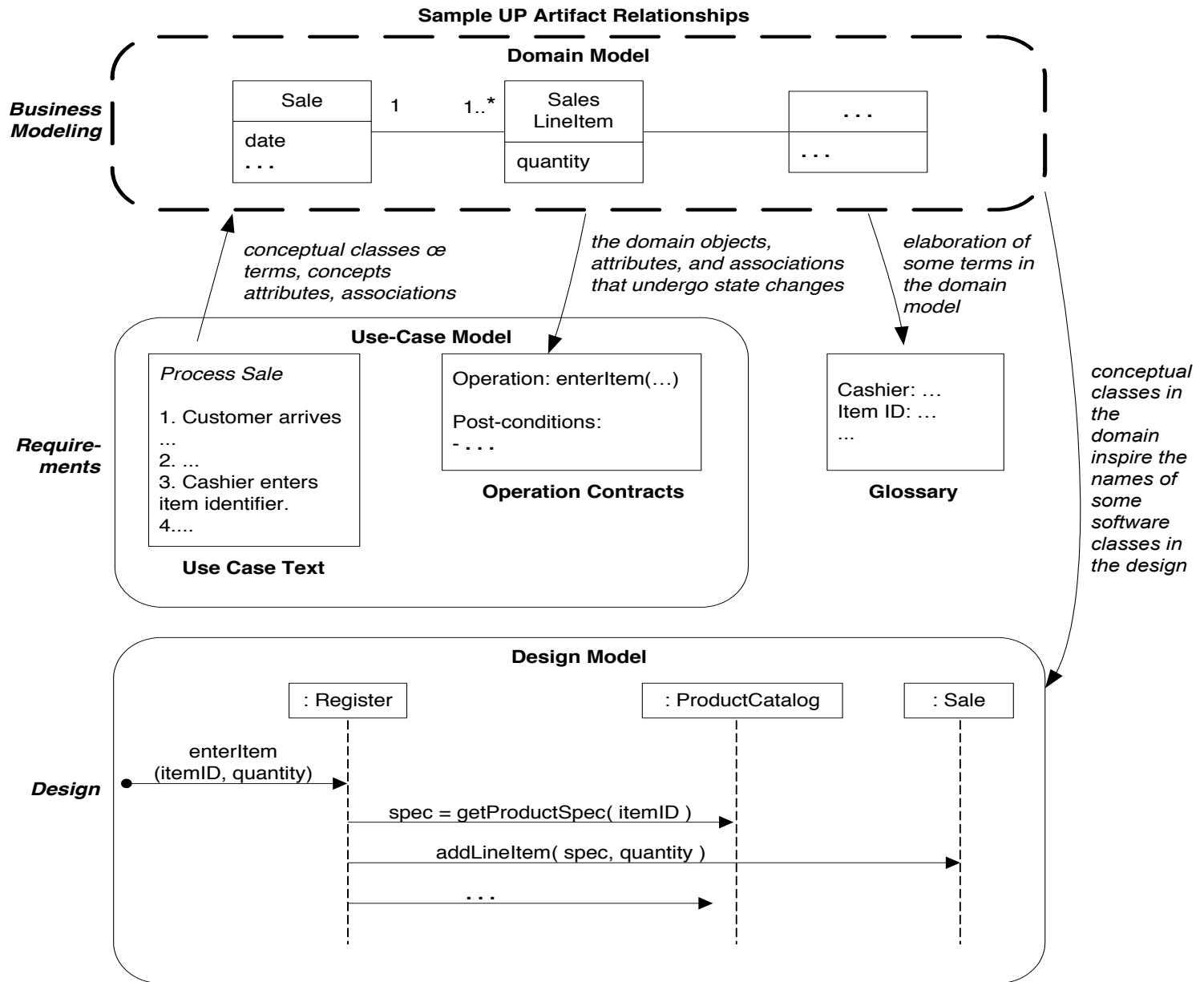


Requirements Modeling (Requirements Class Diagrams)



What is a Domain Model?

- Structuring of domain concepts
 - Identifies problem concepts and their relationships
- Structural models (for example, UML class diagrams) used to depict structure
- Key Question: What are the concepts of interest in this domain?
 - their attributes?
 - their relationships?
- **IMPORTANT:** This is a model of problem concepts; these concepts are **NOT** software objects, but a “visual dictionary” of domain concepts.

Representing Problem/Domain Concepts



visualization of a real-world concept in the domain of interest

it is a *not* a picture of a software class

Avoid Design/Solution/Computer-based concepts

avoid



software artifact; not part of domain model

avoid



software class; not part of domain model

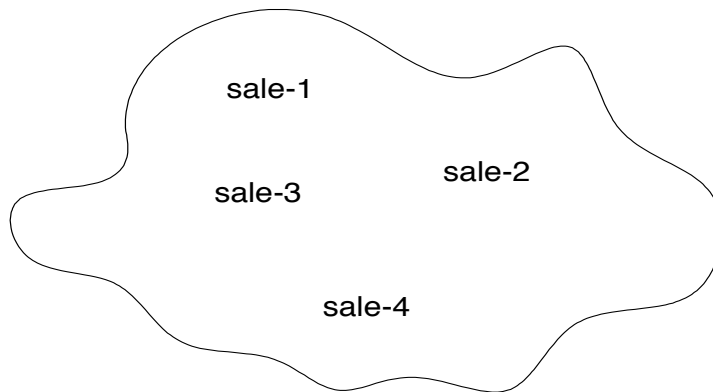
Extensional vs. Intensional Descriptions



concept's symbol

"A sale represents the event of a purchase transaction. It has a date and time."

concept's intension



concept's extension

Requirements class vs. design class

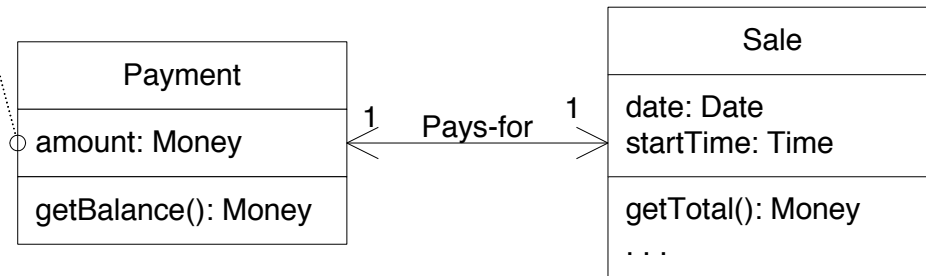
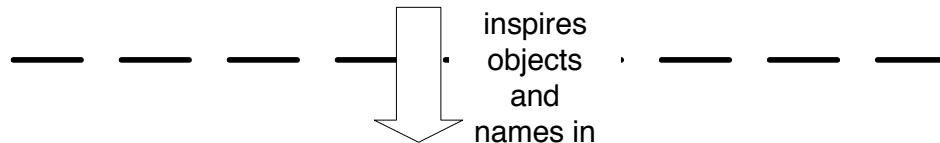
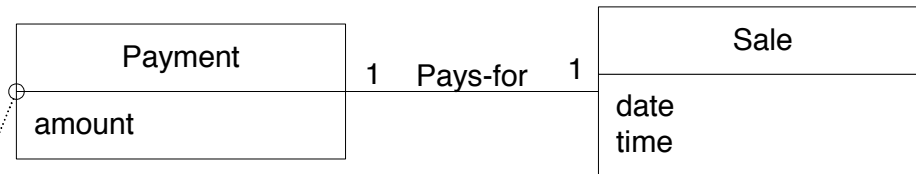
A Payment in the Domain Model is a concept, but a Payment in the Design Model is a software class. They are not the same thing, but the former *inspired* the naming and definition of the latter.

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

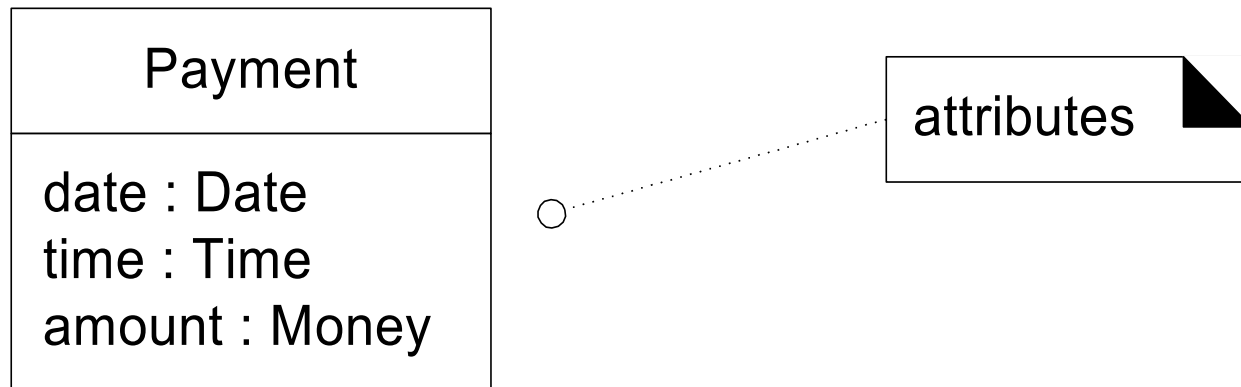
- An attribute is a named property. Each concept instance associates value(s) with each attribute of a concept.

- *OOA syntax*

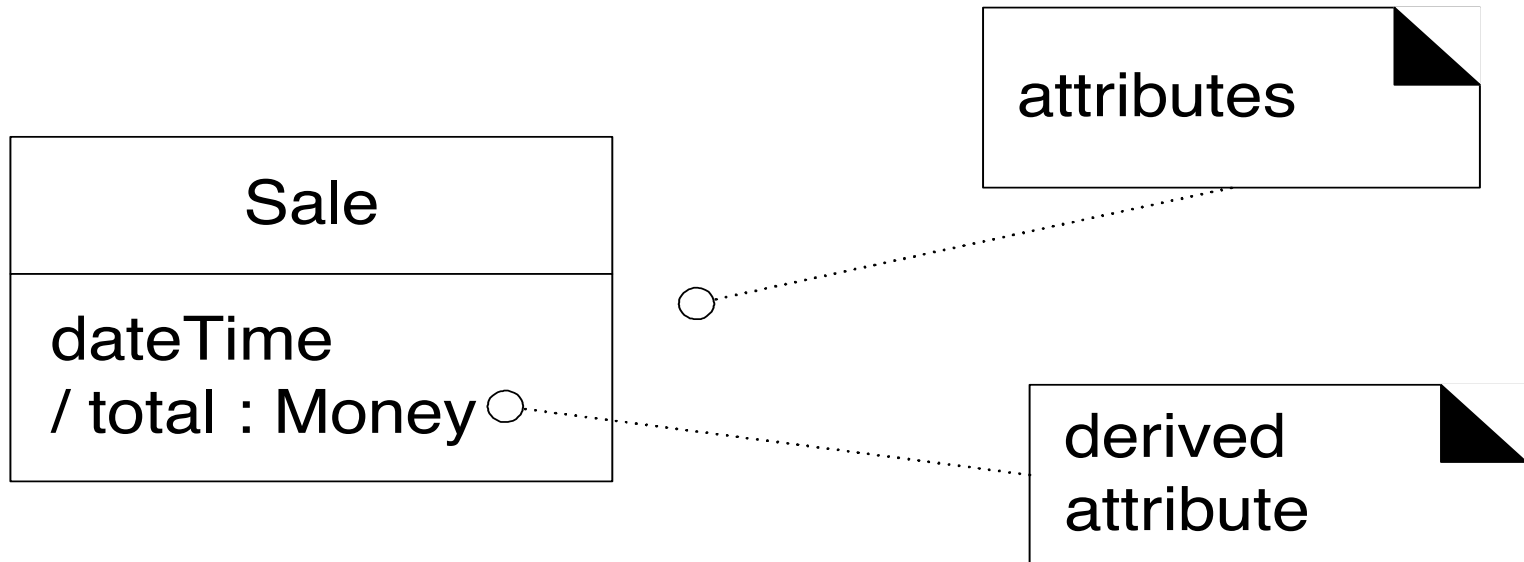
name [multiplicity] [: type] [{property-string}]

Attributes

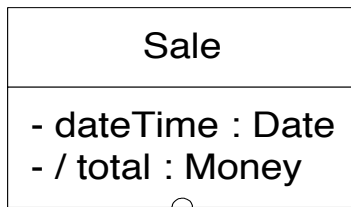
- Show only “simple” relatively primitive types as attributes.
- Connections to other concepts are to be represented as associations, not attributes.



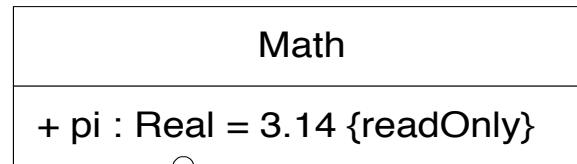
Derived Attributes



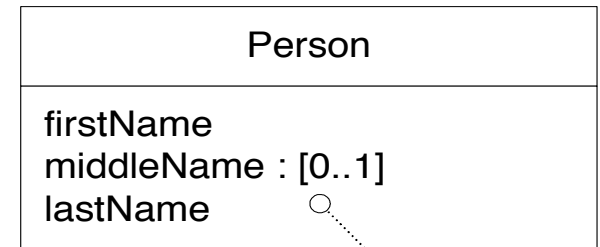
Access Modifiers



Private visibility
attributes



Public visibility readonly
attribute with initialization



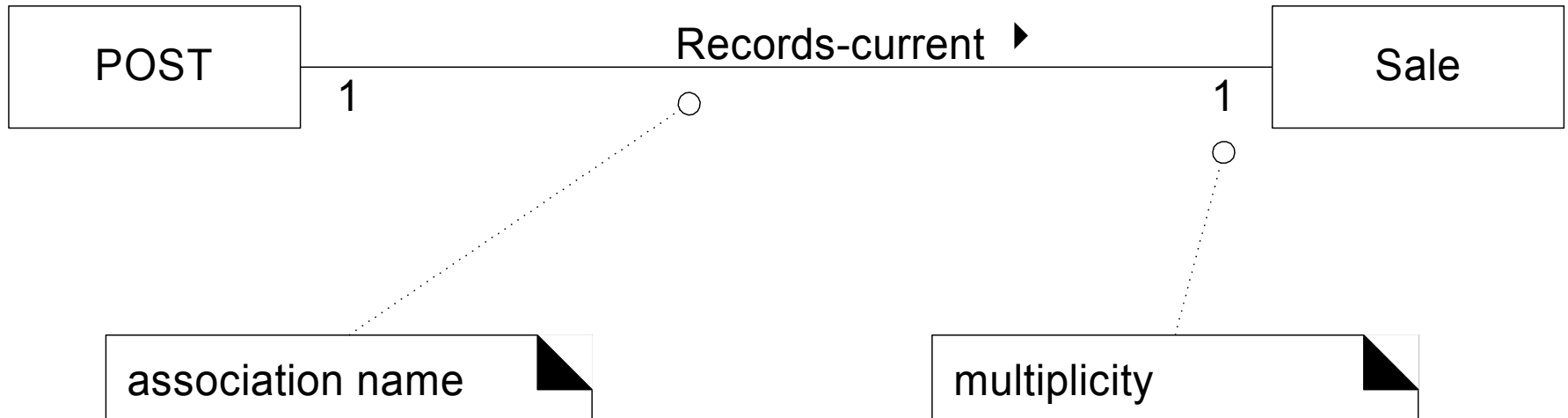
Optional value

Modeling Static Relationships

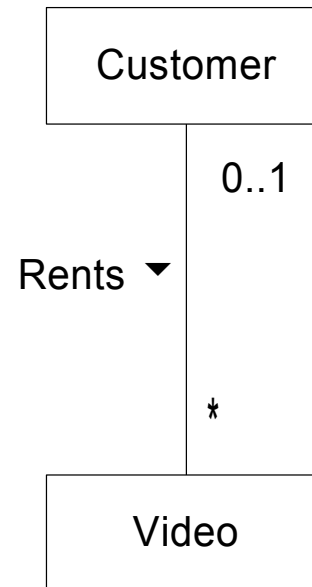
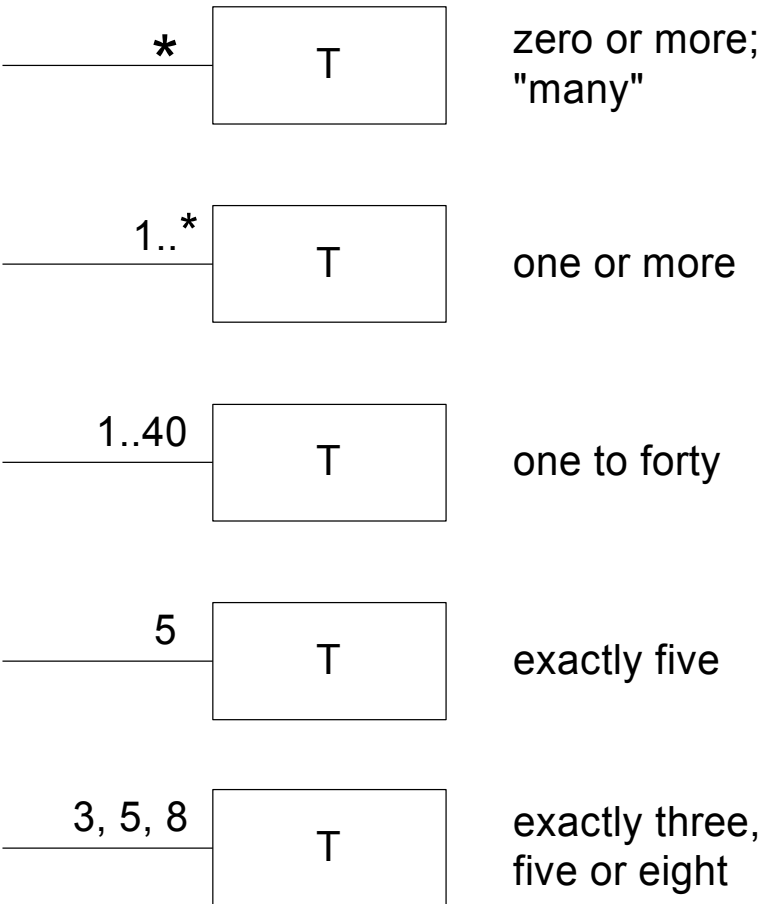
- Two kinds of static relationships:
 - Associations
 - Represent structural relationships among objects
 - Generalizations
 - Represent generalization/specialization class structures
- The two kinds of relationships are orthogonal

Associations

- "direction reading arrow"
- it has **no** meaning except to indicate direction of reading the association label
- often excluded



Multiplicity

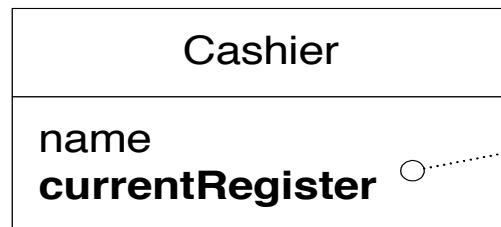


One instance of a Customer may be renting zero or more Videos.

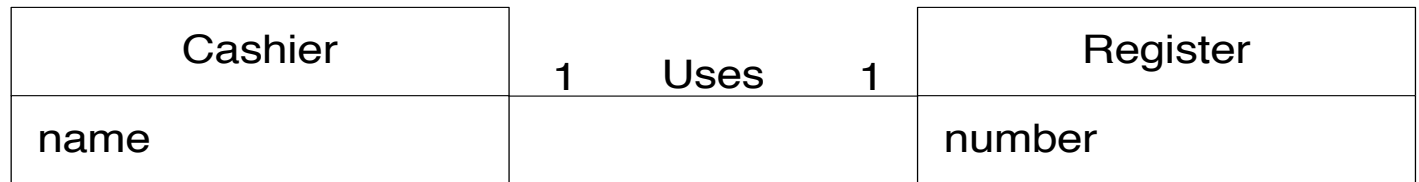
One instance of a Video may be being rented by zero or one Customers.

Association vs. Attribute

Worse

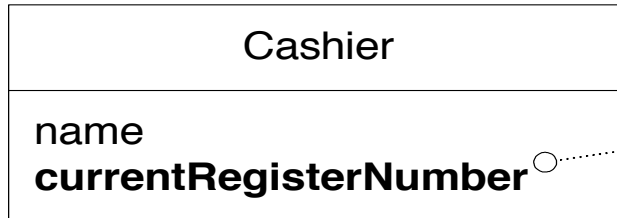


Better



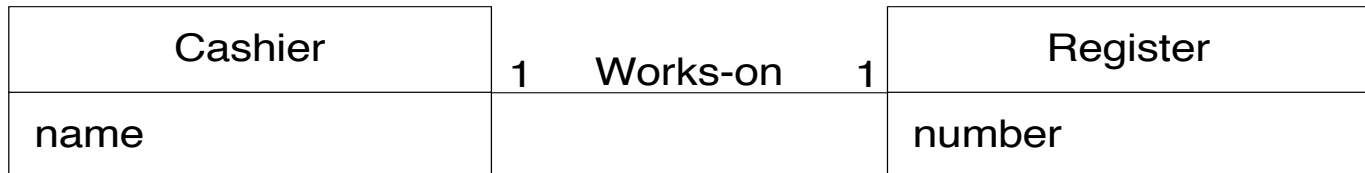
Do not use attributes to represent foreign keys

Worse

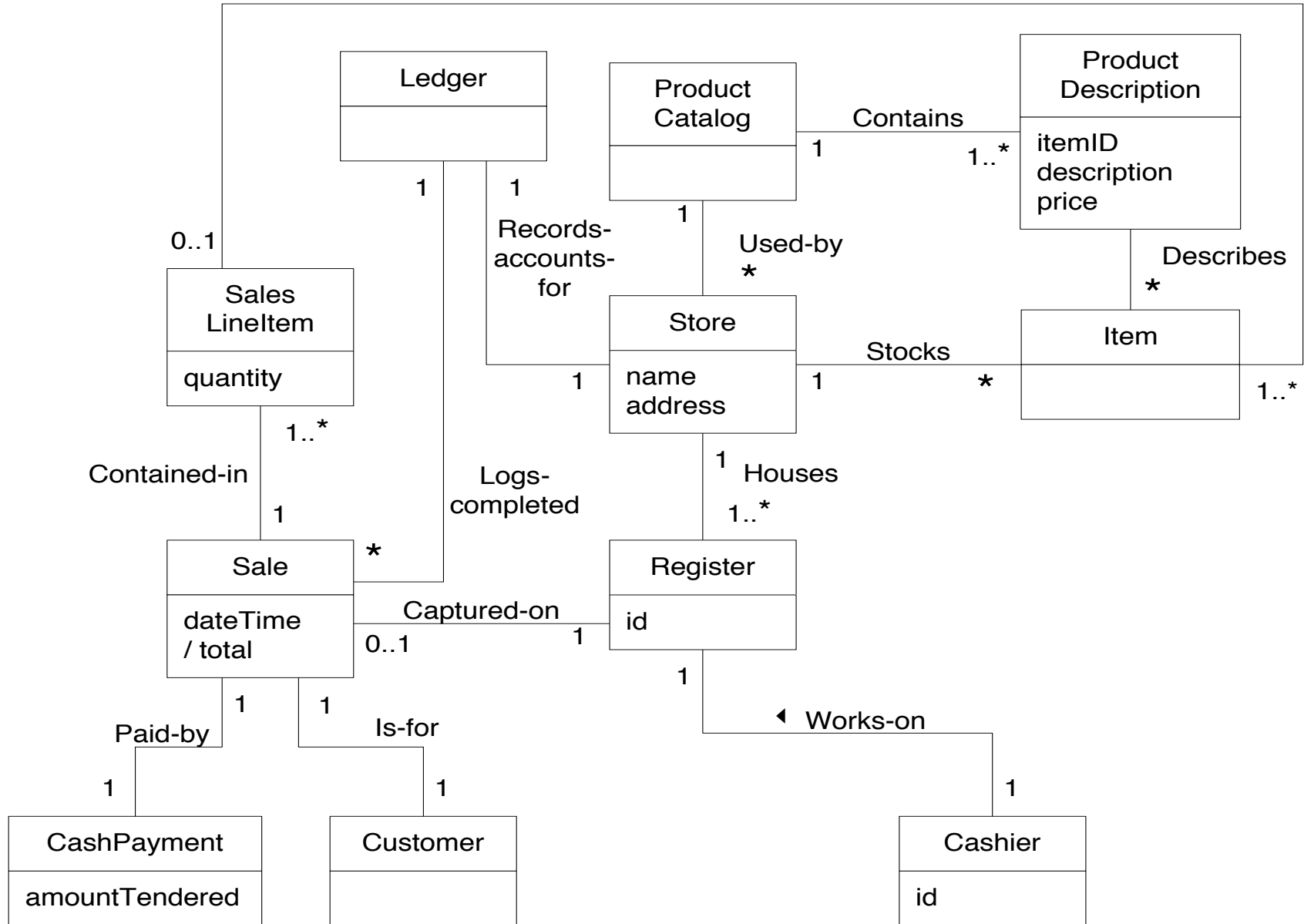


a "simple" attribute, but being used as a foreign key to relate to another object

Better



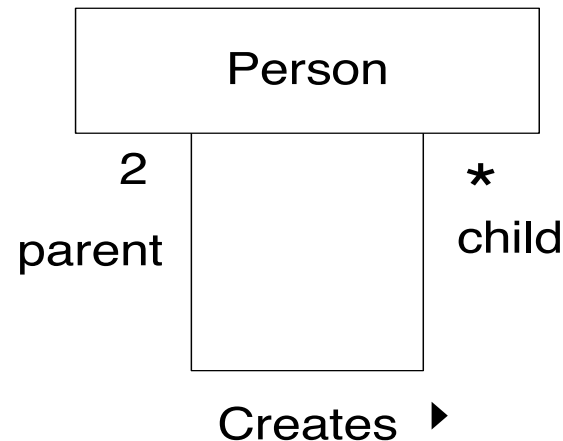
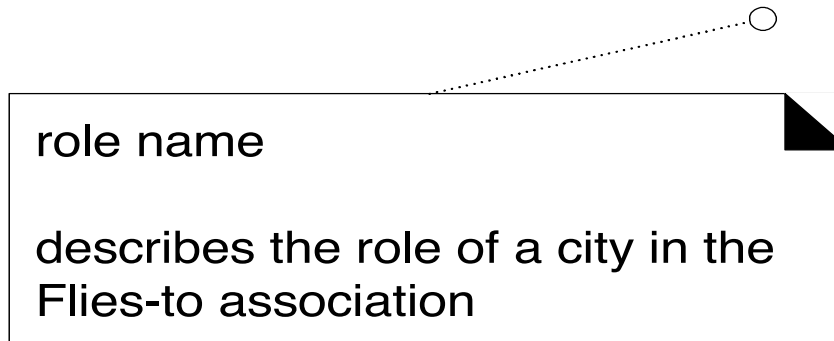
Records-sale-of



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

Domain Models

- Requirements Class Diagrams

(Module 9)

(weak) Aggregation

Composition

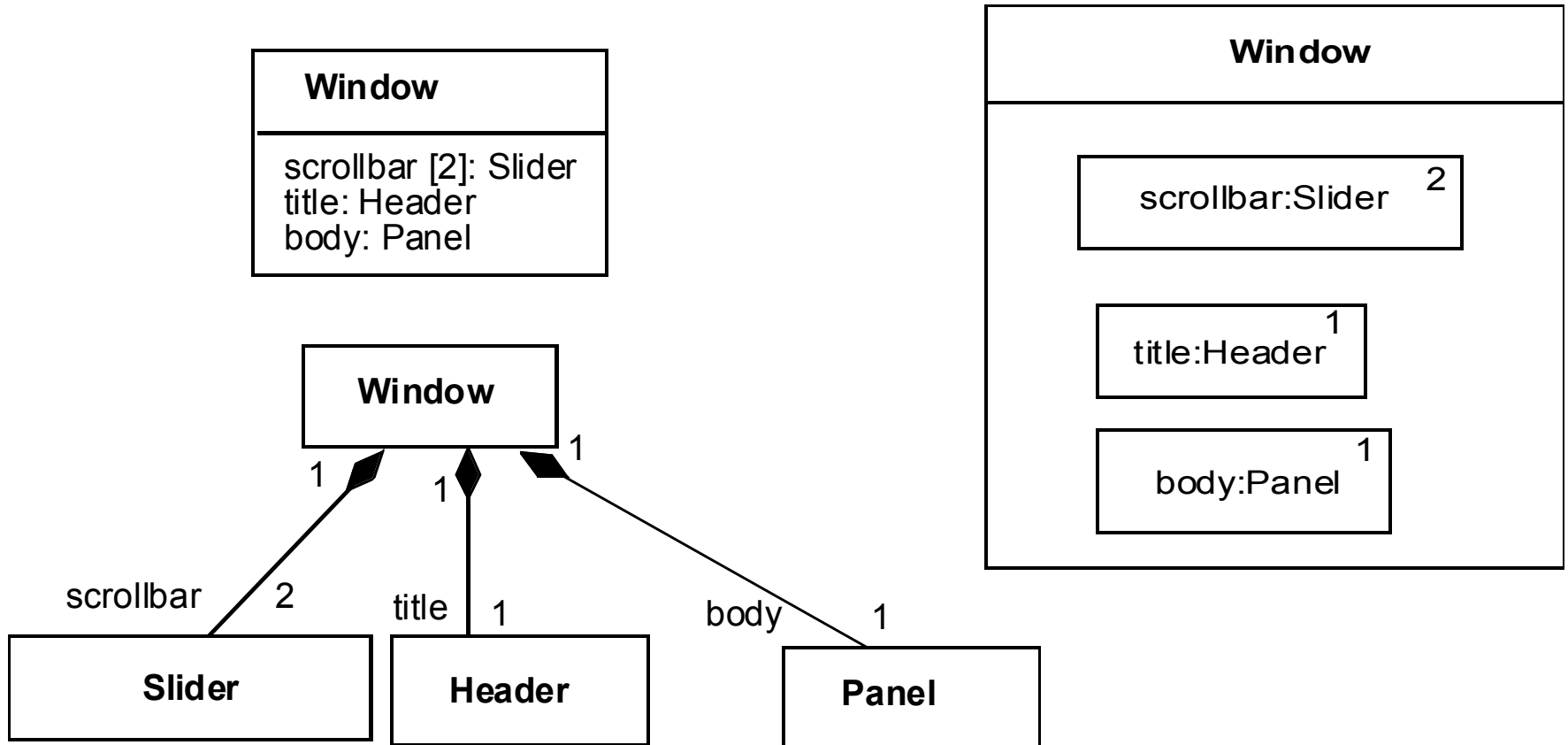


Fig. 3-45, *UML Notation Guide*

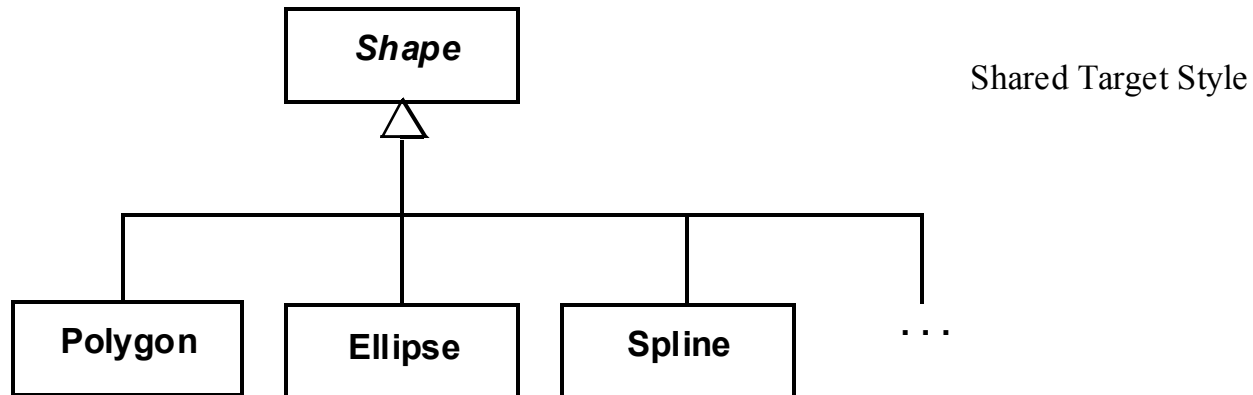
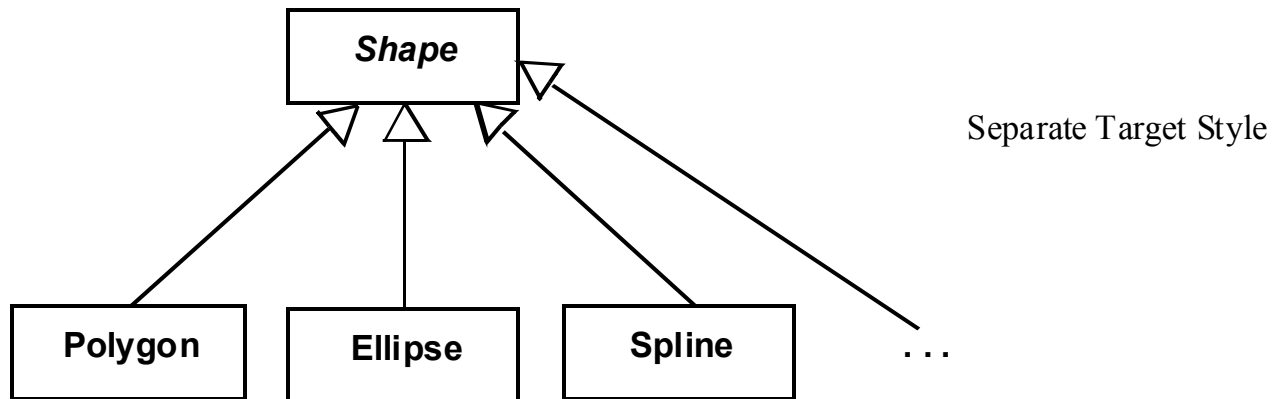
Generalization/Specialization

- 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



Generalization

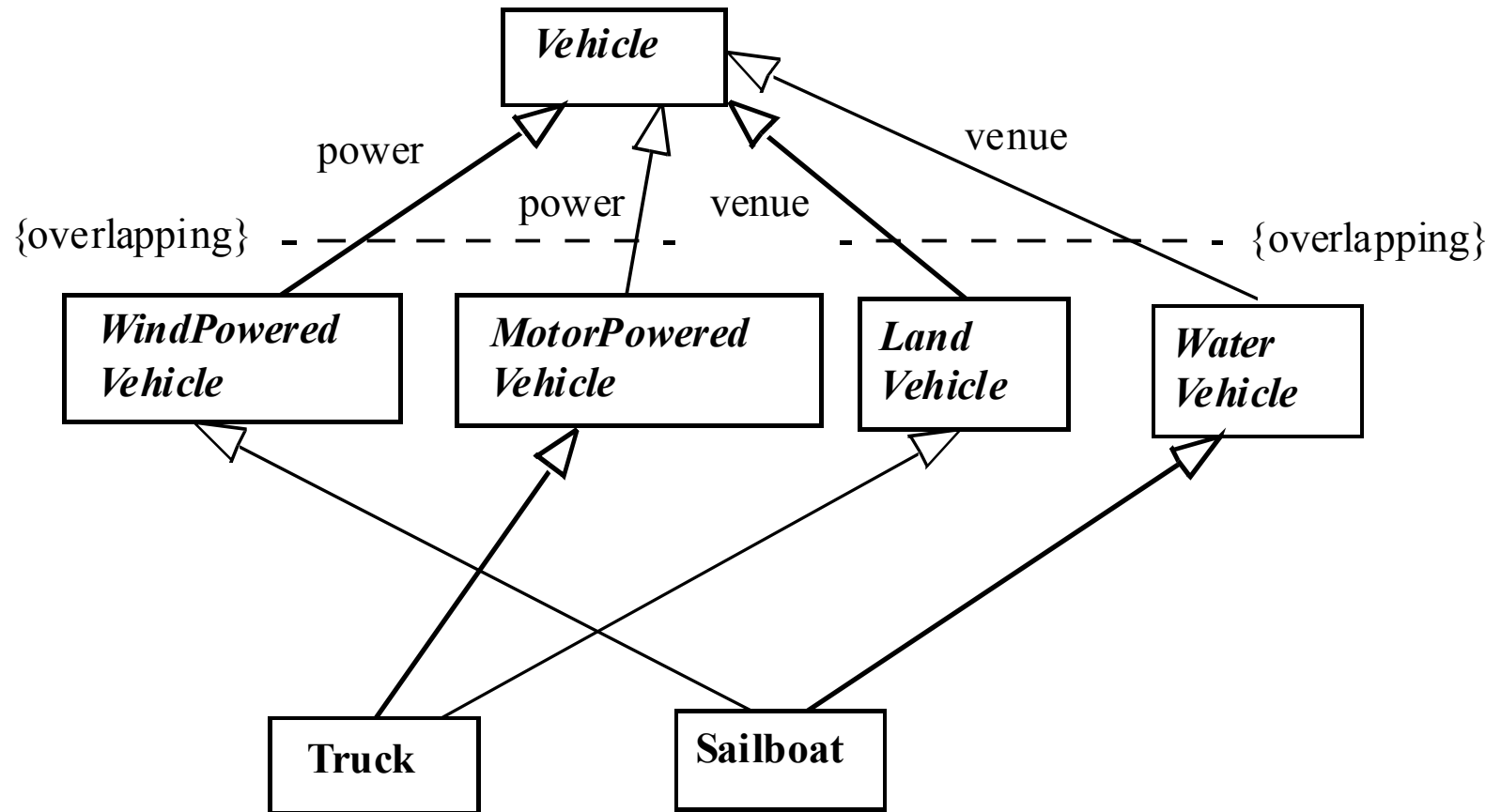
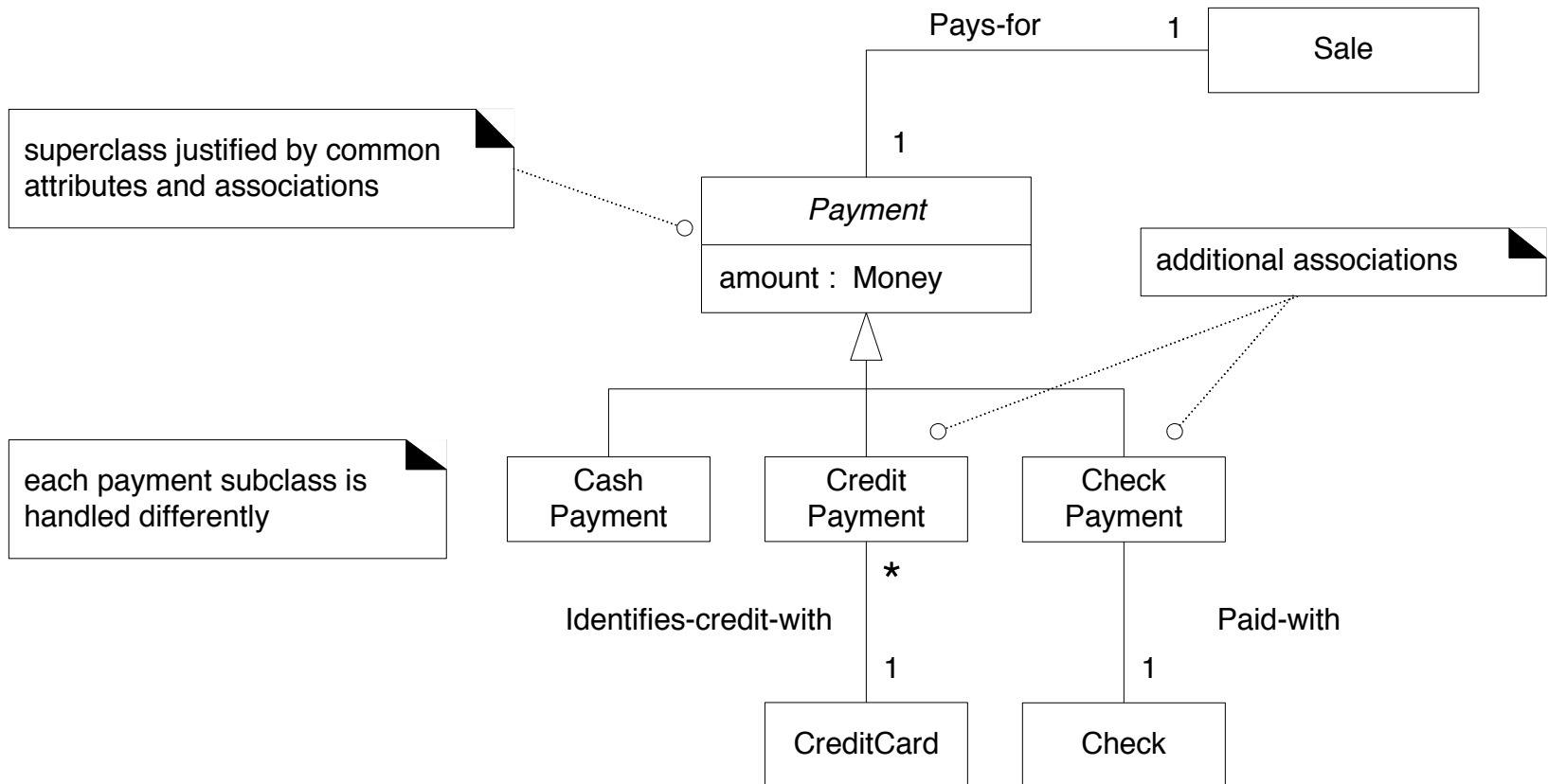
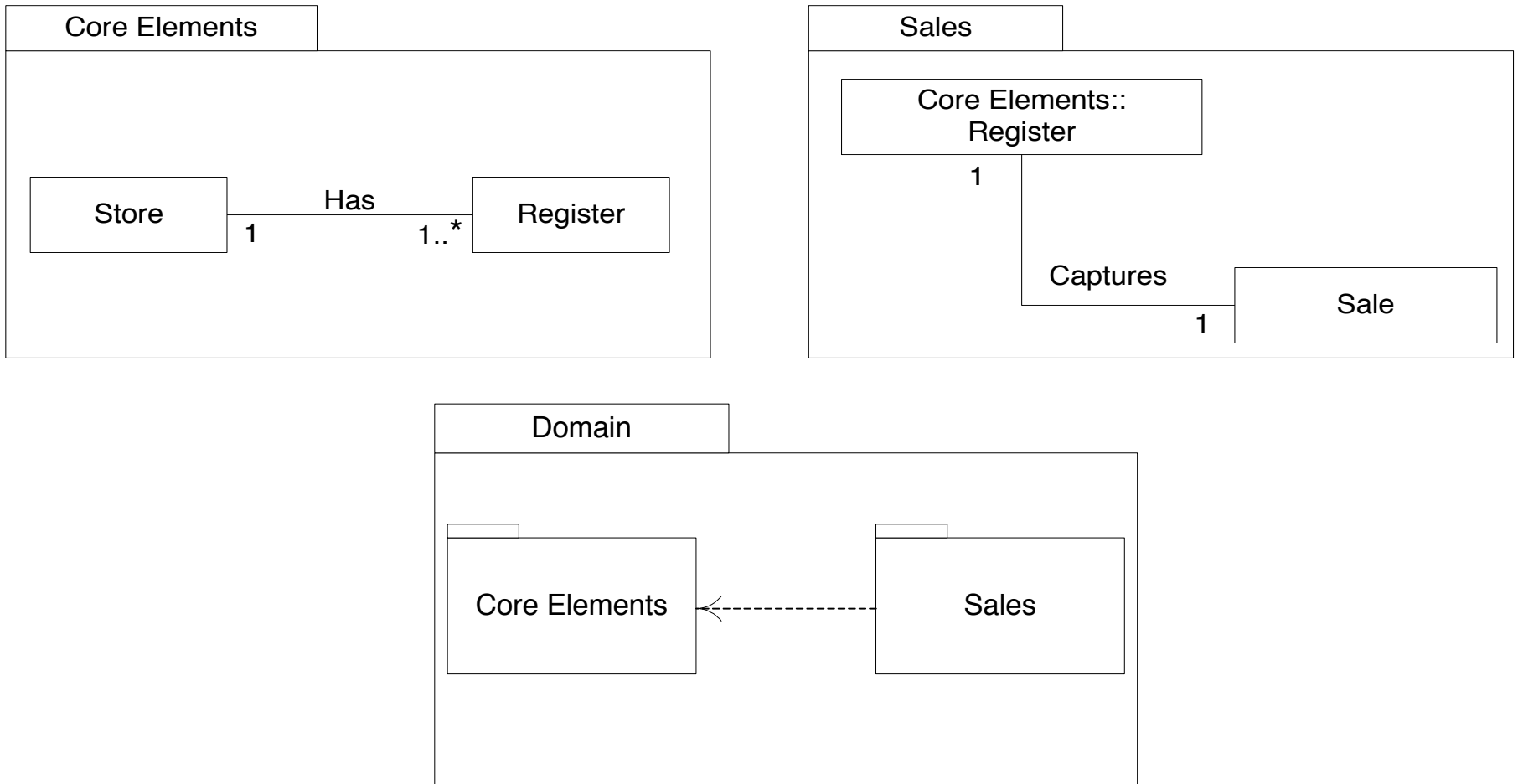


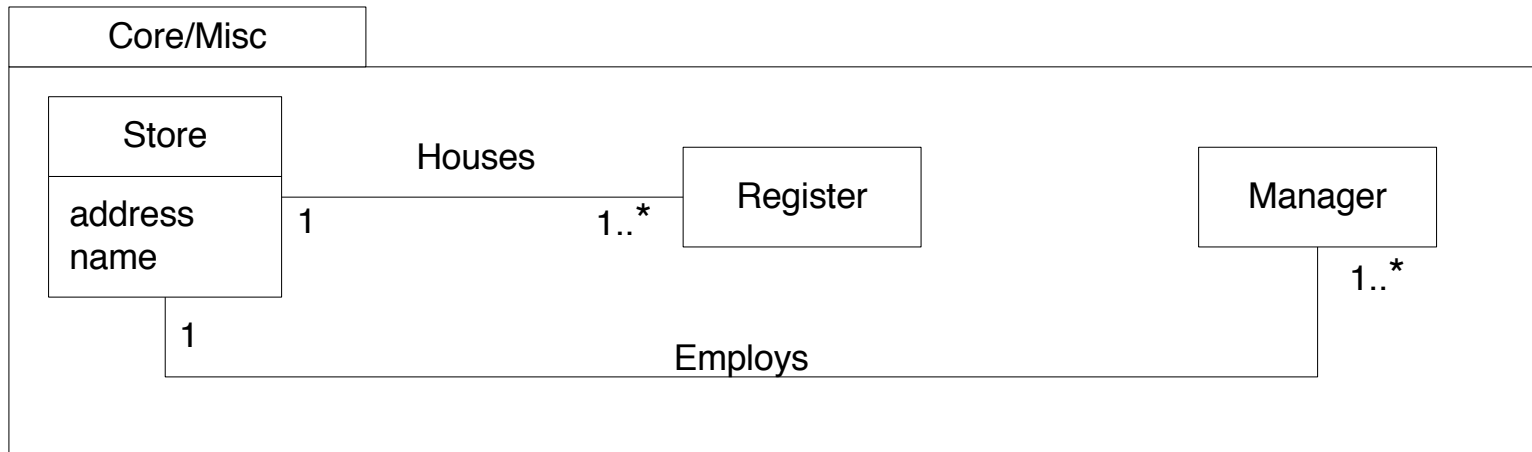
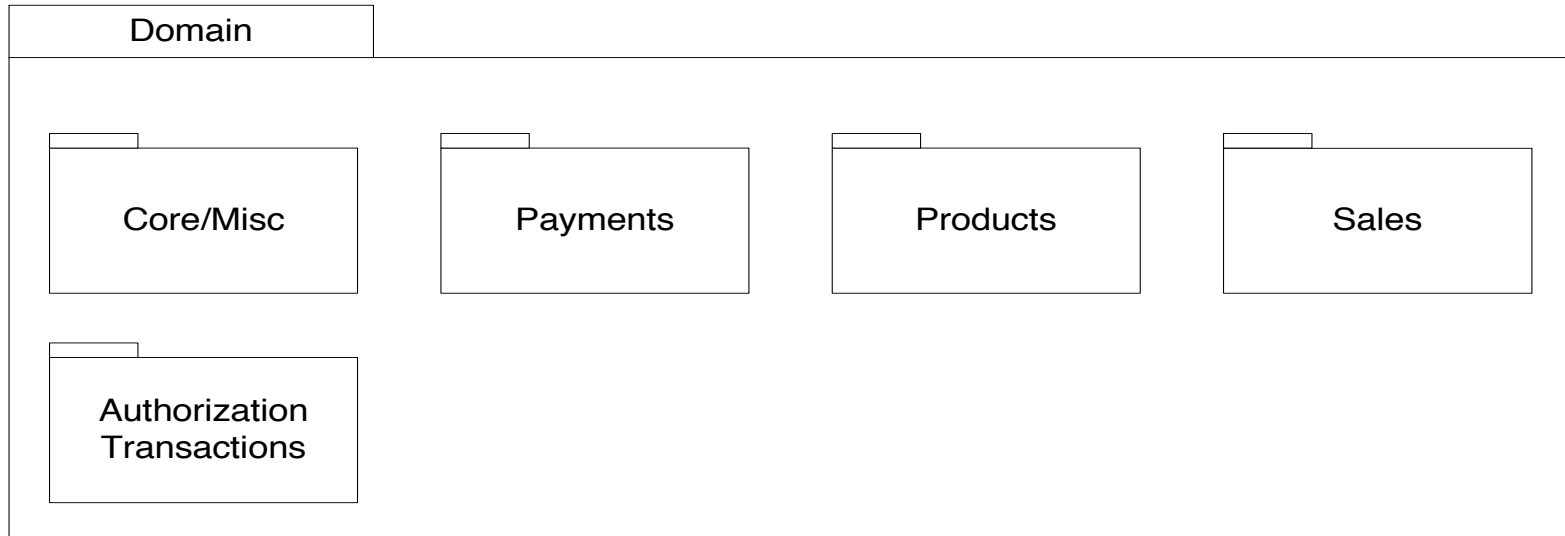
Fig. 3-48, *UML Notation Guide*

Inheritance of associations

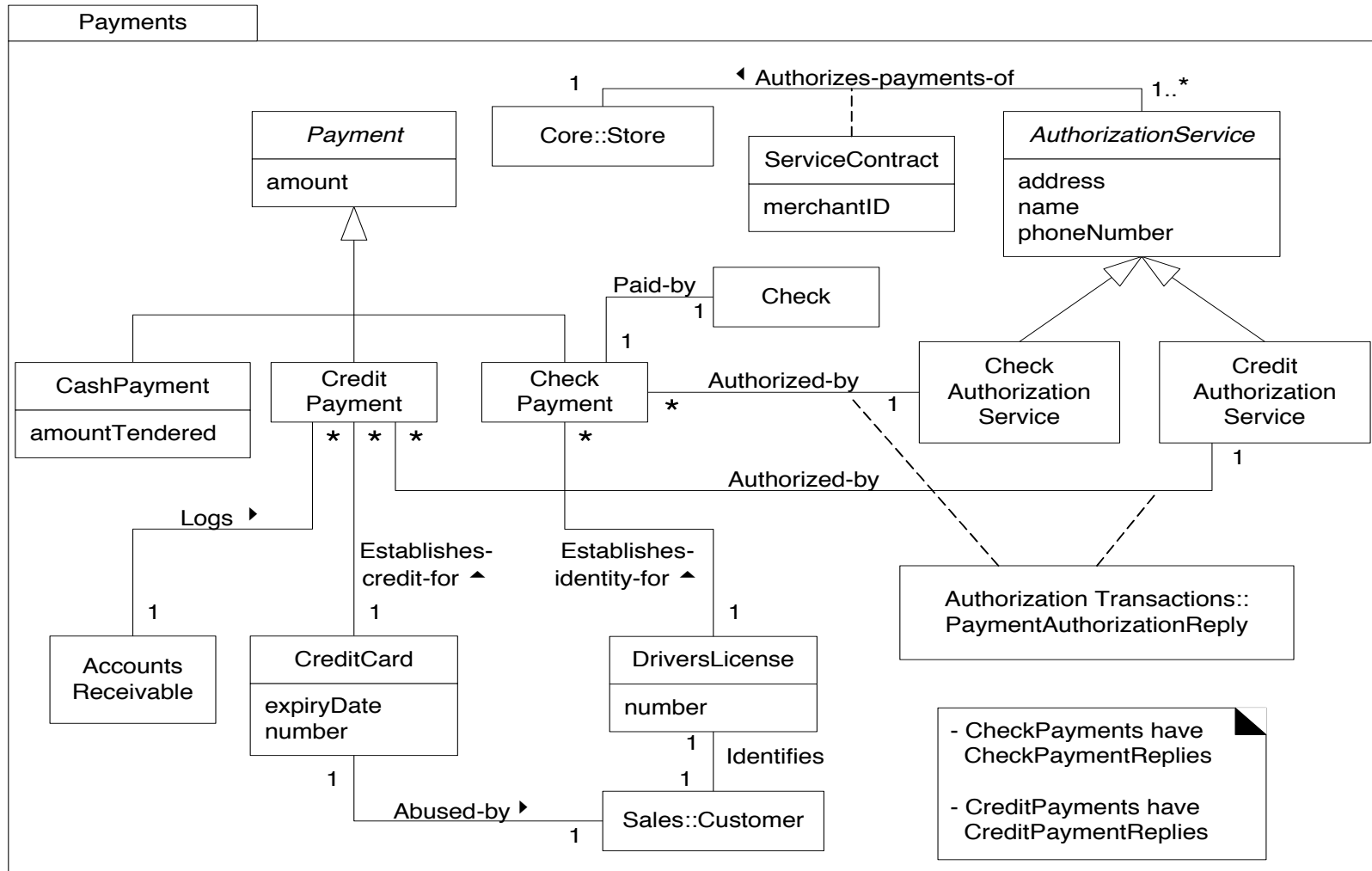


Handling Large Domain Models

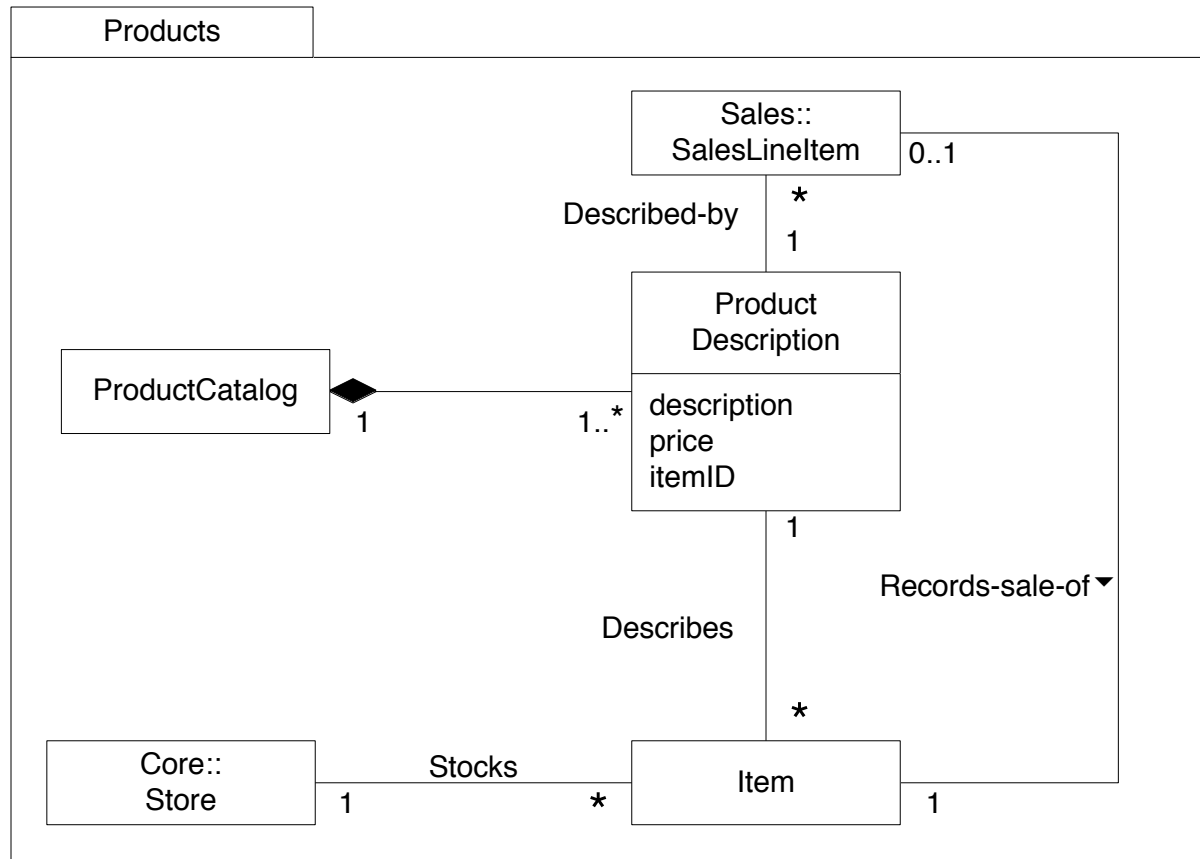




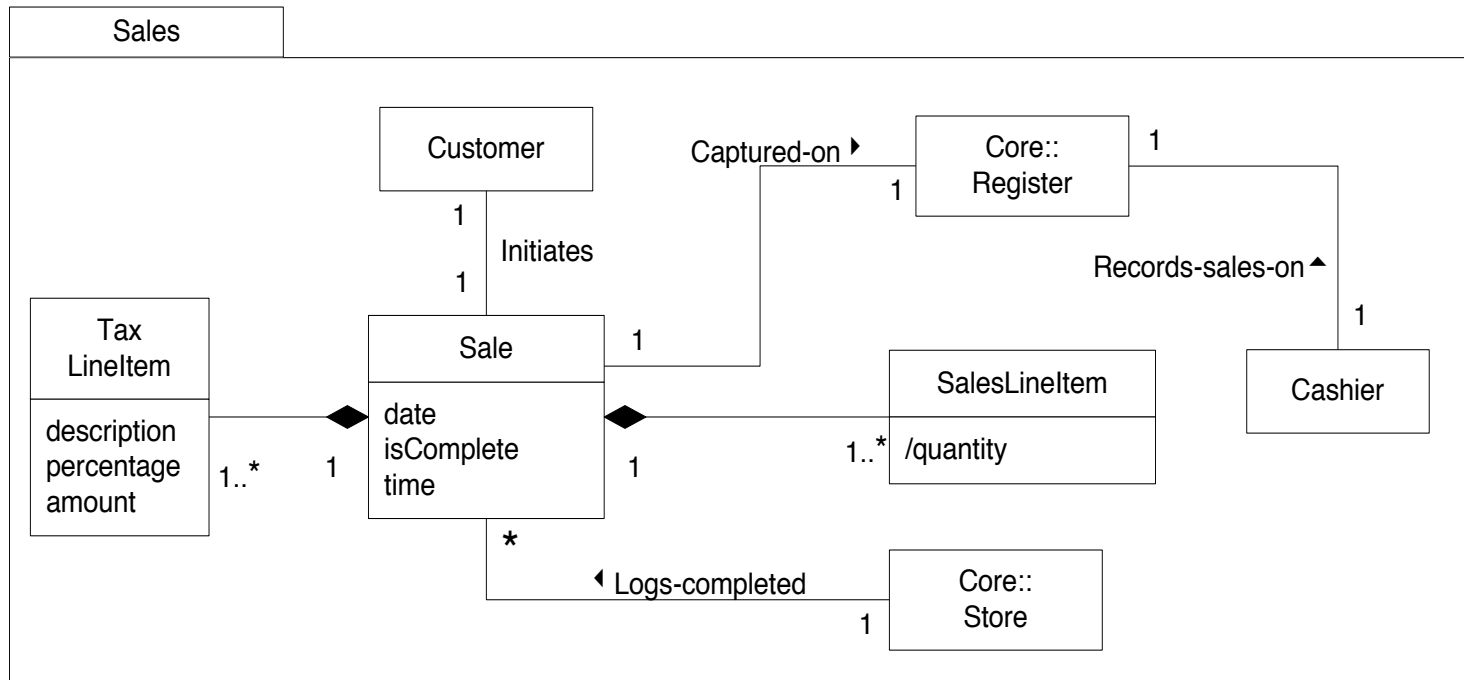
Payments Package



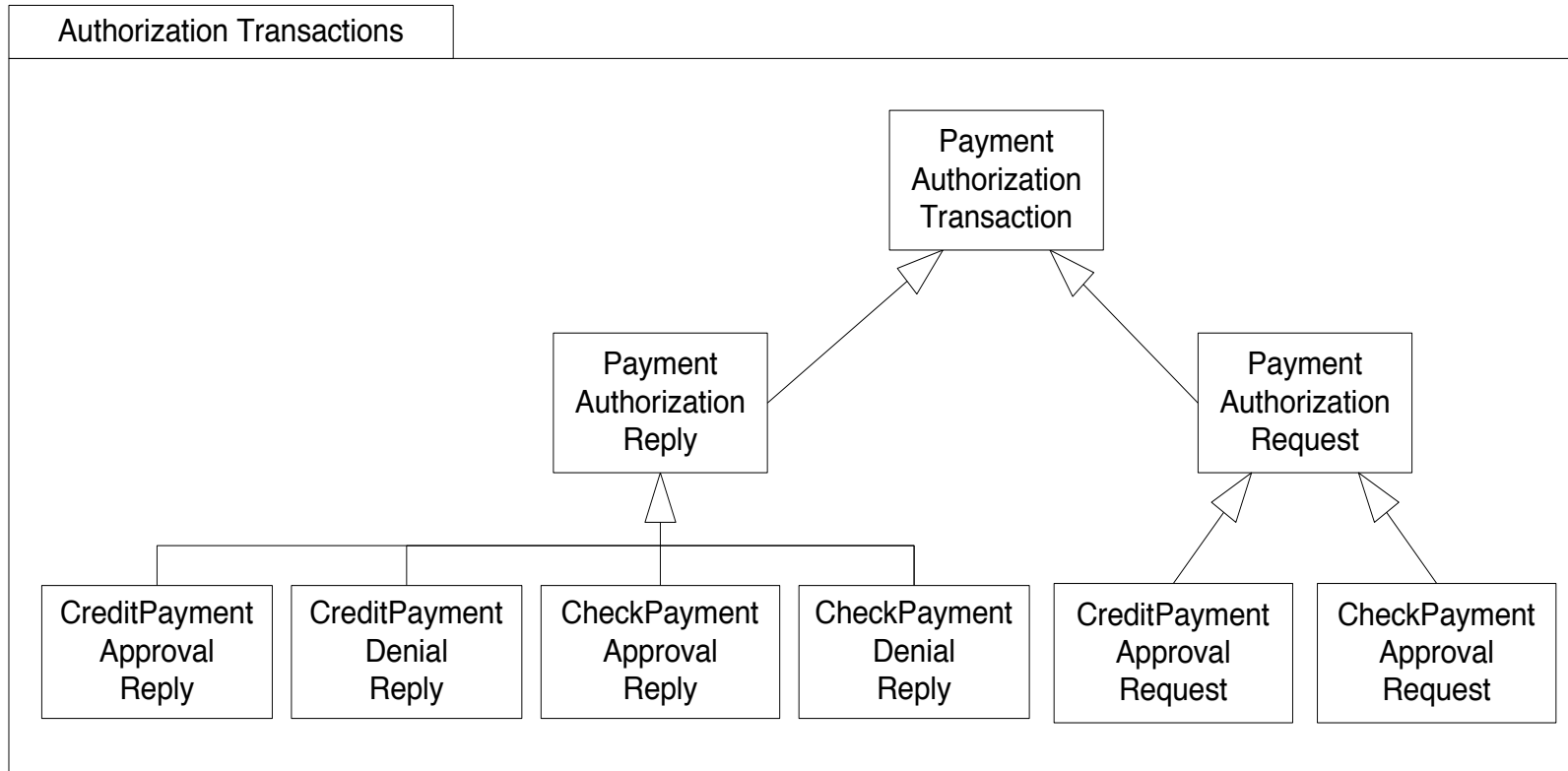
Products Package



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