Domain Model Refinement

When to make generalization hierarchies? Why is the following a good example?

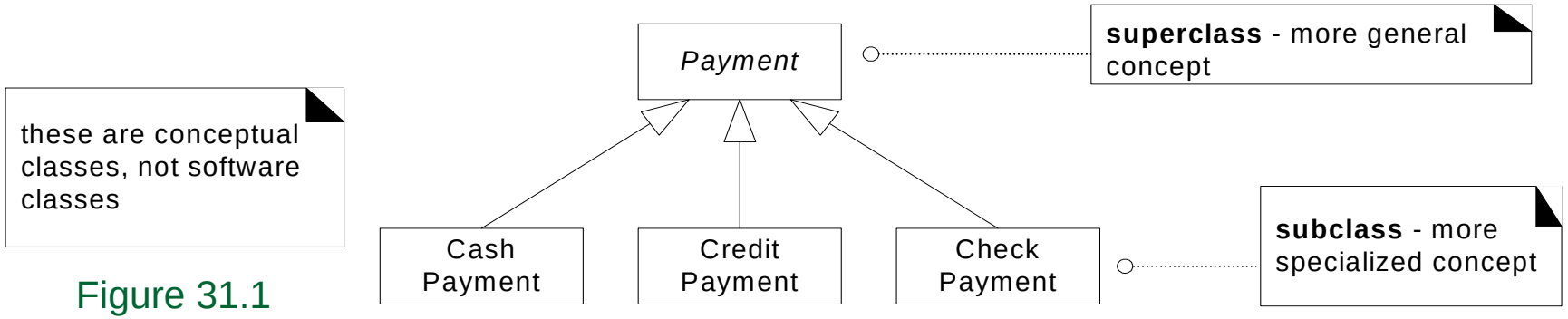
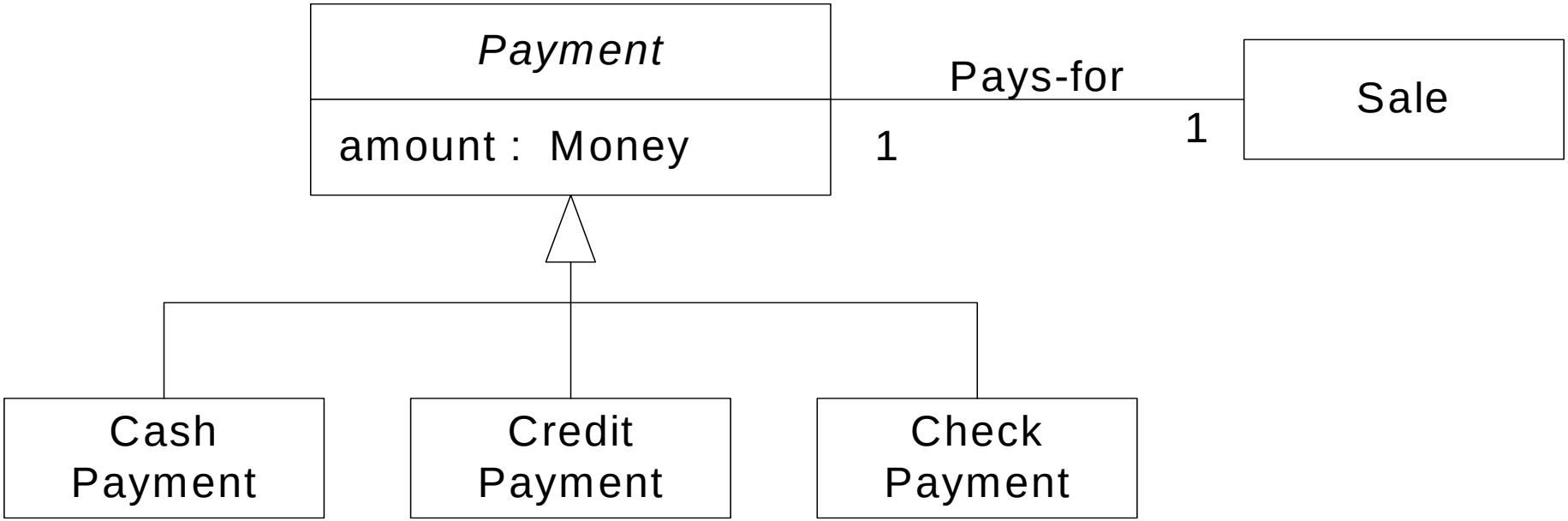


Figure 31.1

Guidelines:

- Identify superclasses and subclasses when they help us understand concepts in more general, abstract terms and reduce repeated information.
- Expand class hierarchy relevant to the current iteration and show them in the Domain Model.

100% and Is-a rules



100% rule: Subclasses must conform to 100% of superclass's attributes and associations. Do they above?

Is-a rule (informal test): "Subclass is a Superclass"

What about this hierarchy?

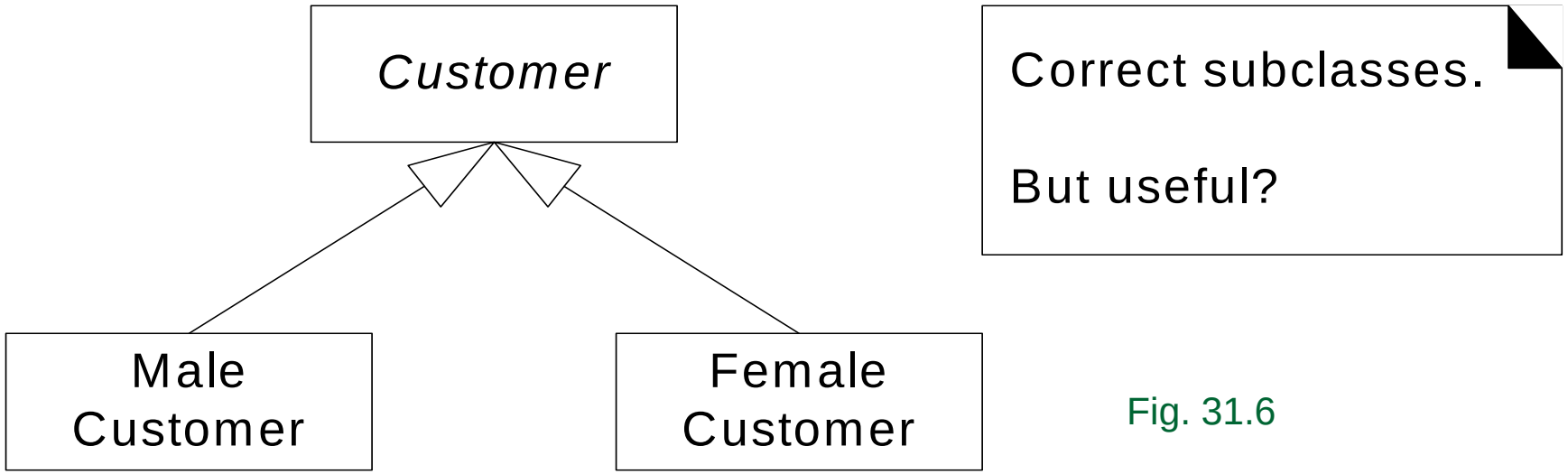


Fig. 31.6

Guidelines for creating conceptual subclasses:

- Subclass has additional attributes or associations of interest
- Subclass behaves or is operated on, or handled or manipulated differently than superclass or other subclasses

Is this hierarchy OK? What does it add to our understanding of the domain?

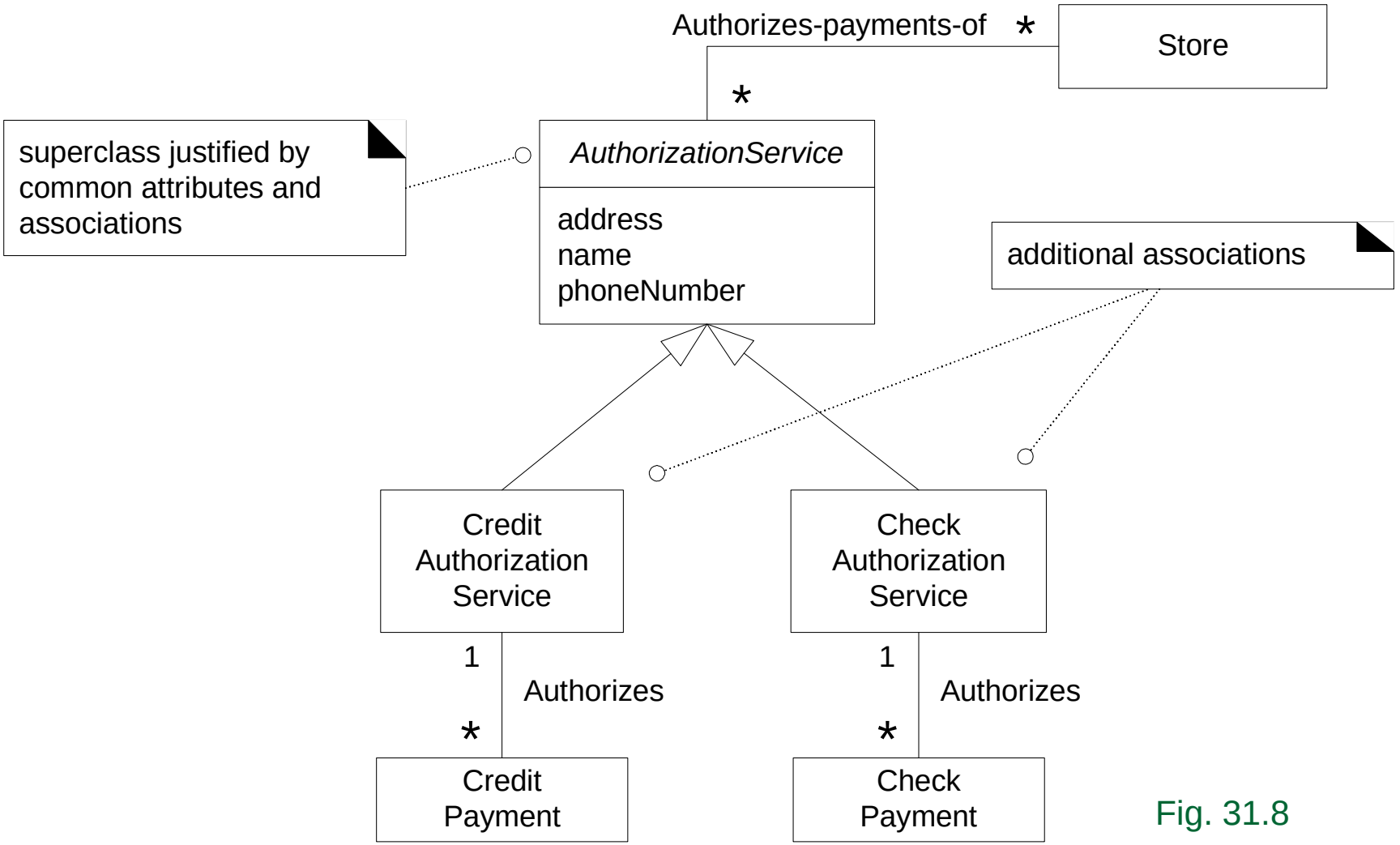


Fig. 31.8

Models transactions with external services which are important in POS domain

When to design Association classes?

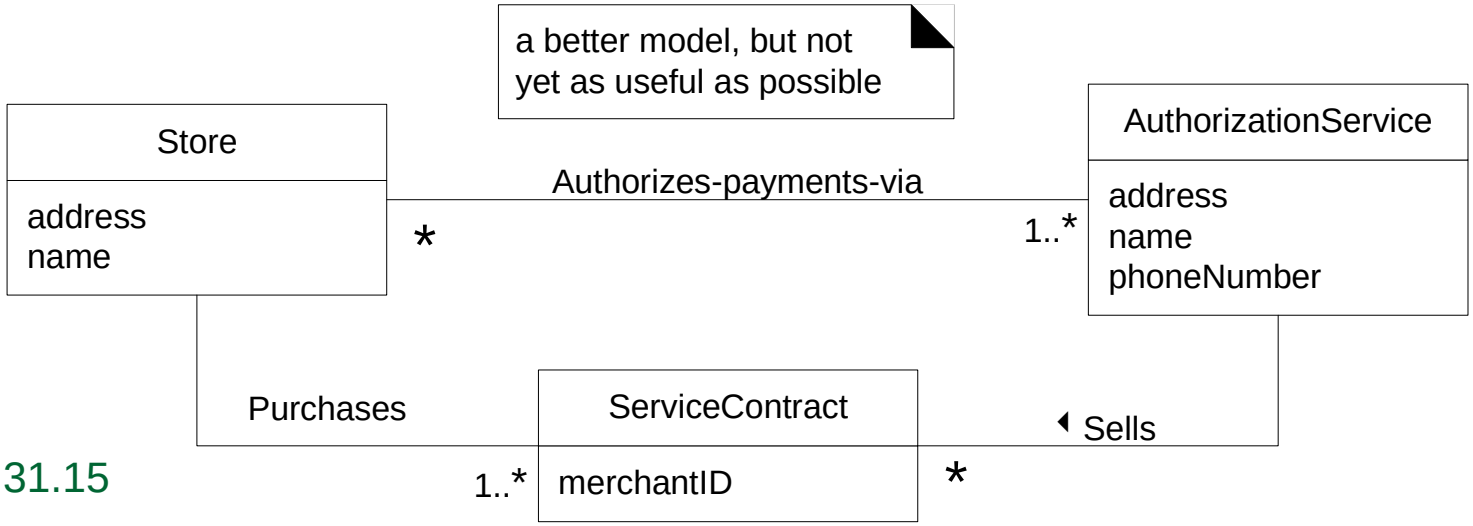


Fig. 31.15

ServiceContract records merchantID—good.
But ServiceContract is dependent on relationship between two classes

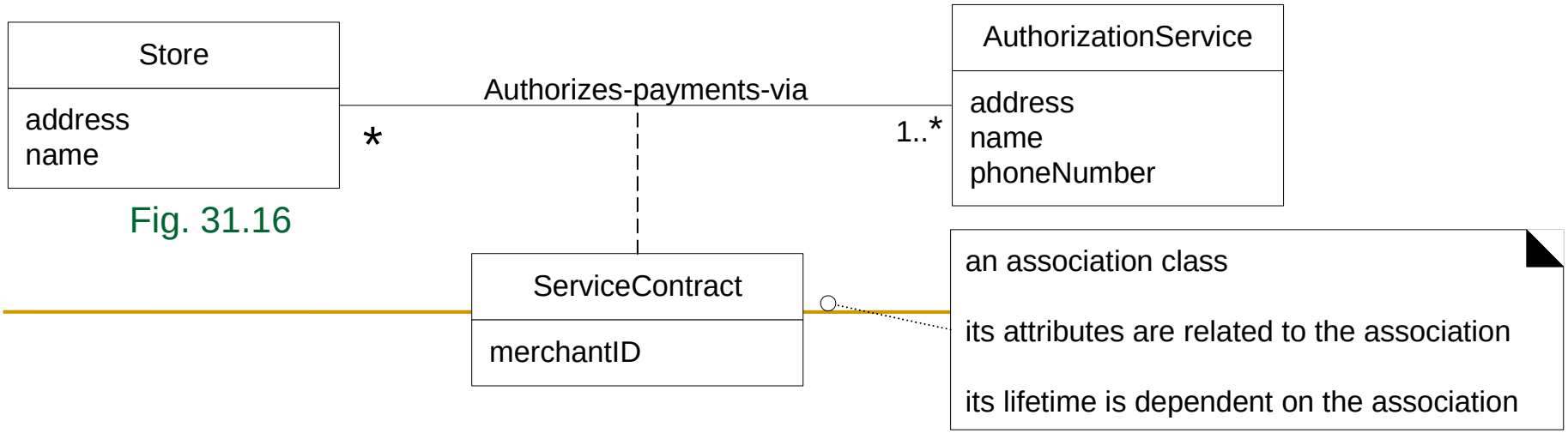
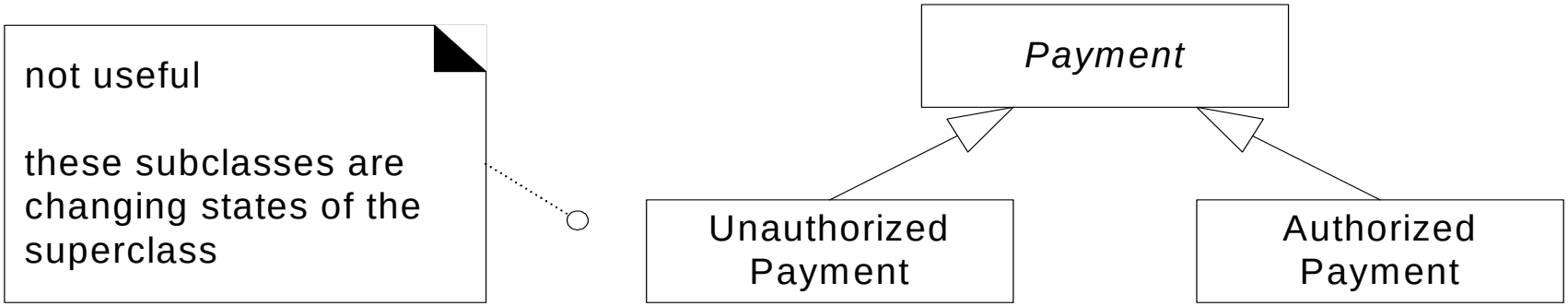


Fig. 31.16

What about this hierarchy? (Why not?)



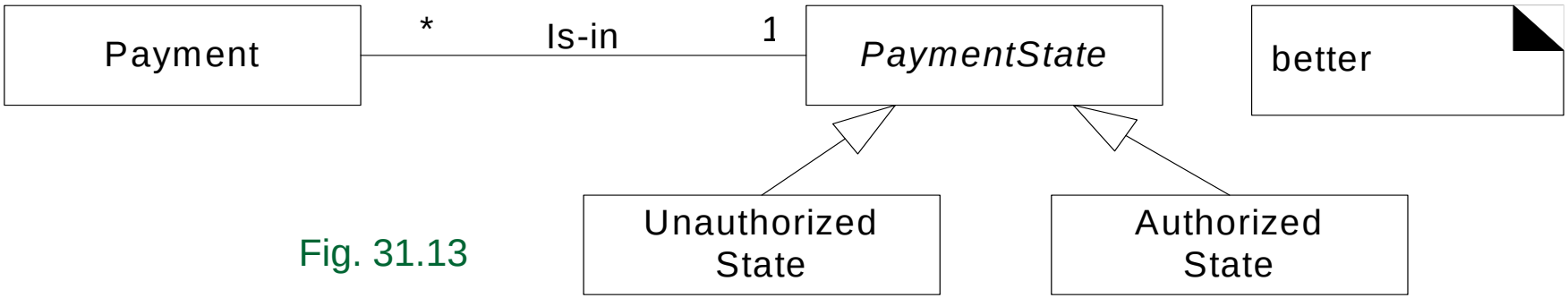


Fig. 31.13

- Classes should be invariant
- Classes can maintain state information as attributes
- Rather than subclasses, model changing states with state charts

More Association classes

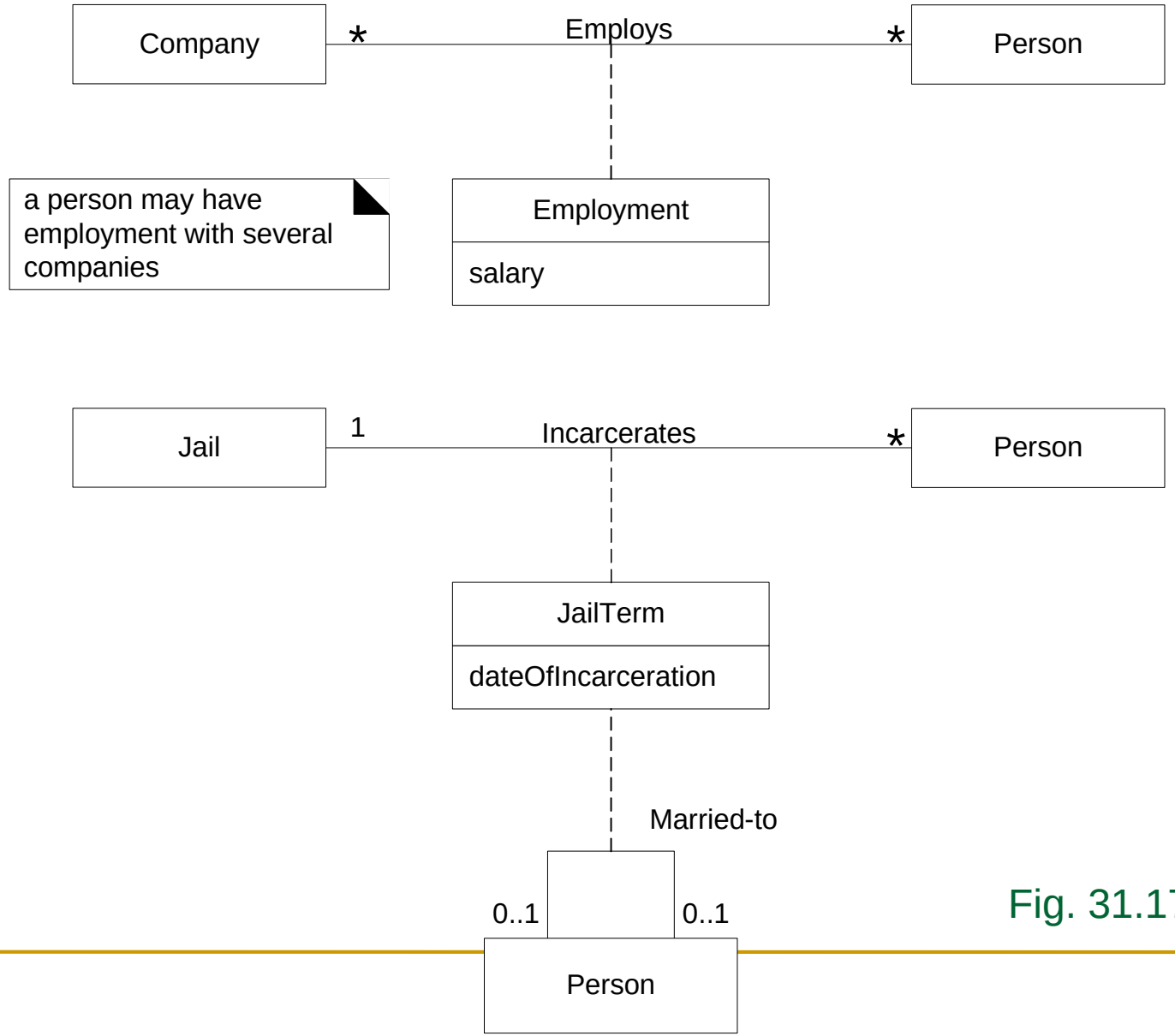


Fig. 31.17

When to add Composition notation?

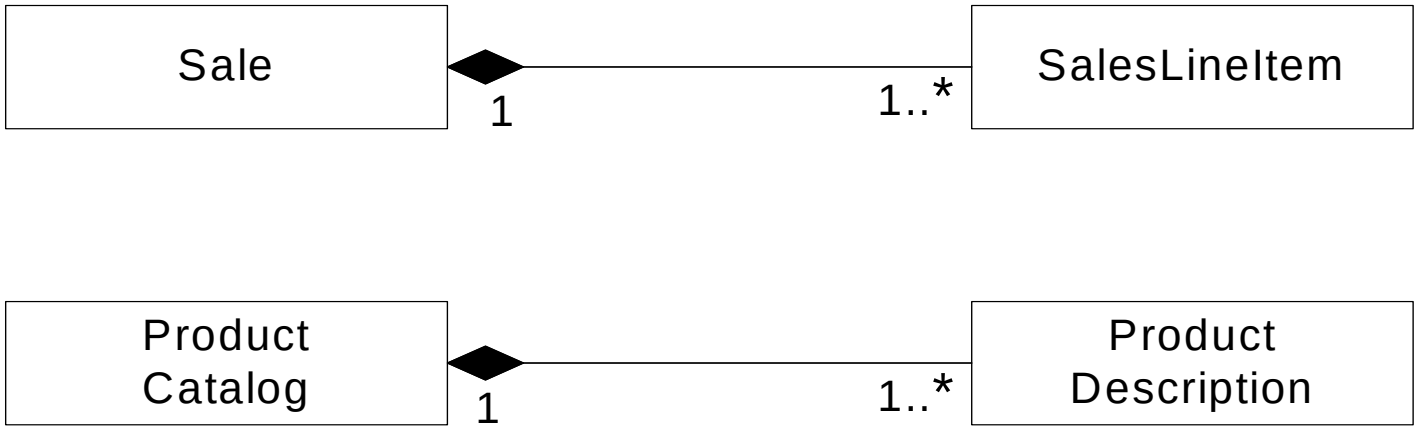


Fig. 31.18

Guidelines for drawing Composition (whole-part) relationships:

- Obvious whole-part physical or logical assembly
- Lifetime of part is bound within lifetime of whole (create-delete dependency)
- Some properties of whole propagate to parts, e.g., location
- Operations of whole propagate to parts, e.g., movement
- But: If in doubt, leave it out.

UML package diagrams

- UML packages for higher level structure than classes
- Denoted by box with smaller box on top
  - Note dependency arrows
  - A dependency indicates that changes to one element may cause changes to the other
- Guidelines for partitioning the domain model into packages:
  - Place elements together if they are in same class hierarchy, participate in the same use cases,
    or closely related by concept or purpose, or strongly associated
- Packages can be grouped in higher-order packages
  - Packages may include packages
  - Common package as <<global>> means all packages in system have dependency to this one
  - General package marked {abstract} means this package is an interface, with subtypes
- Guidelines: divide classes into packages;
  analyze dependencies; refactor to reduce dependencies

Higher order package for POS domain

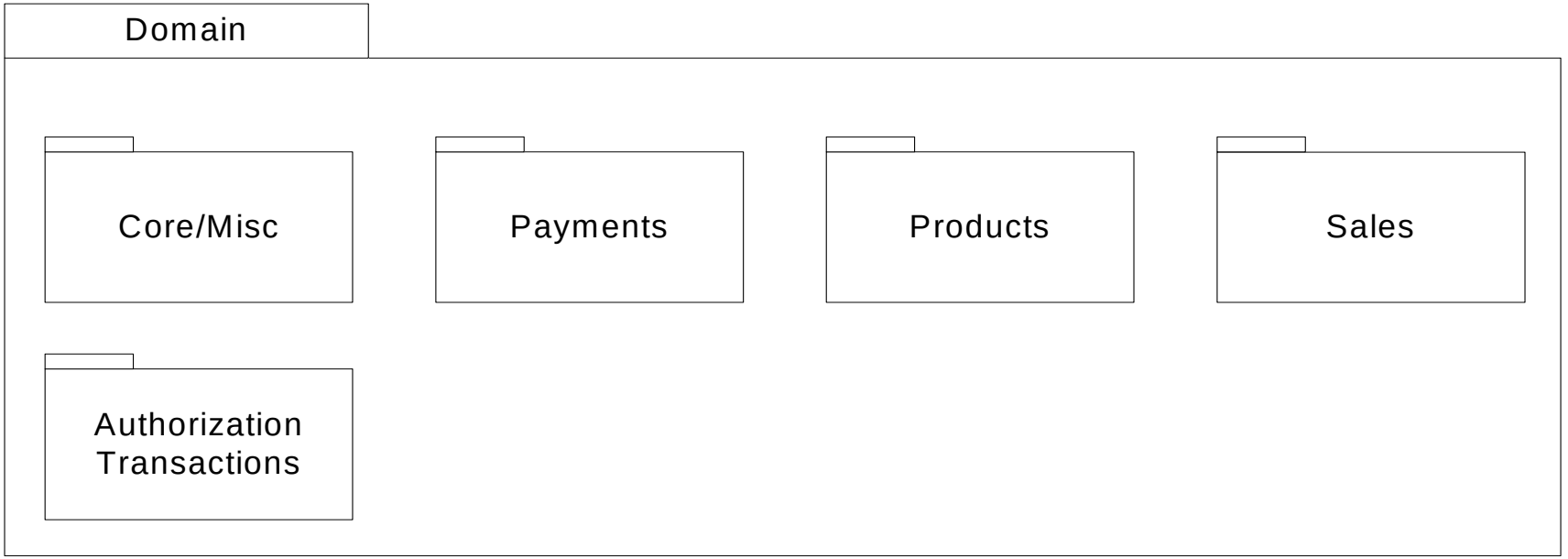


Fig. 31.29

What does this higher order package contain?

Core/Misc package

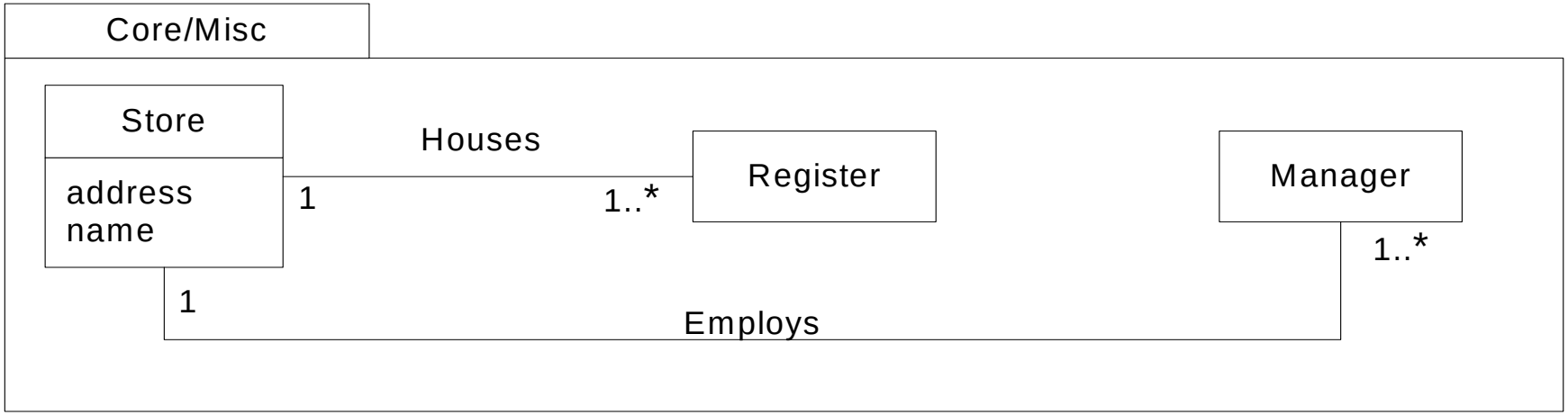


Fig. 31.30

Why call this package Core for the POS domain?

A rich package

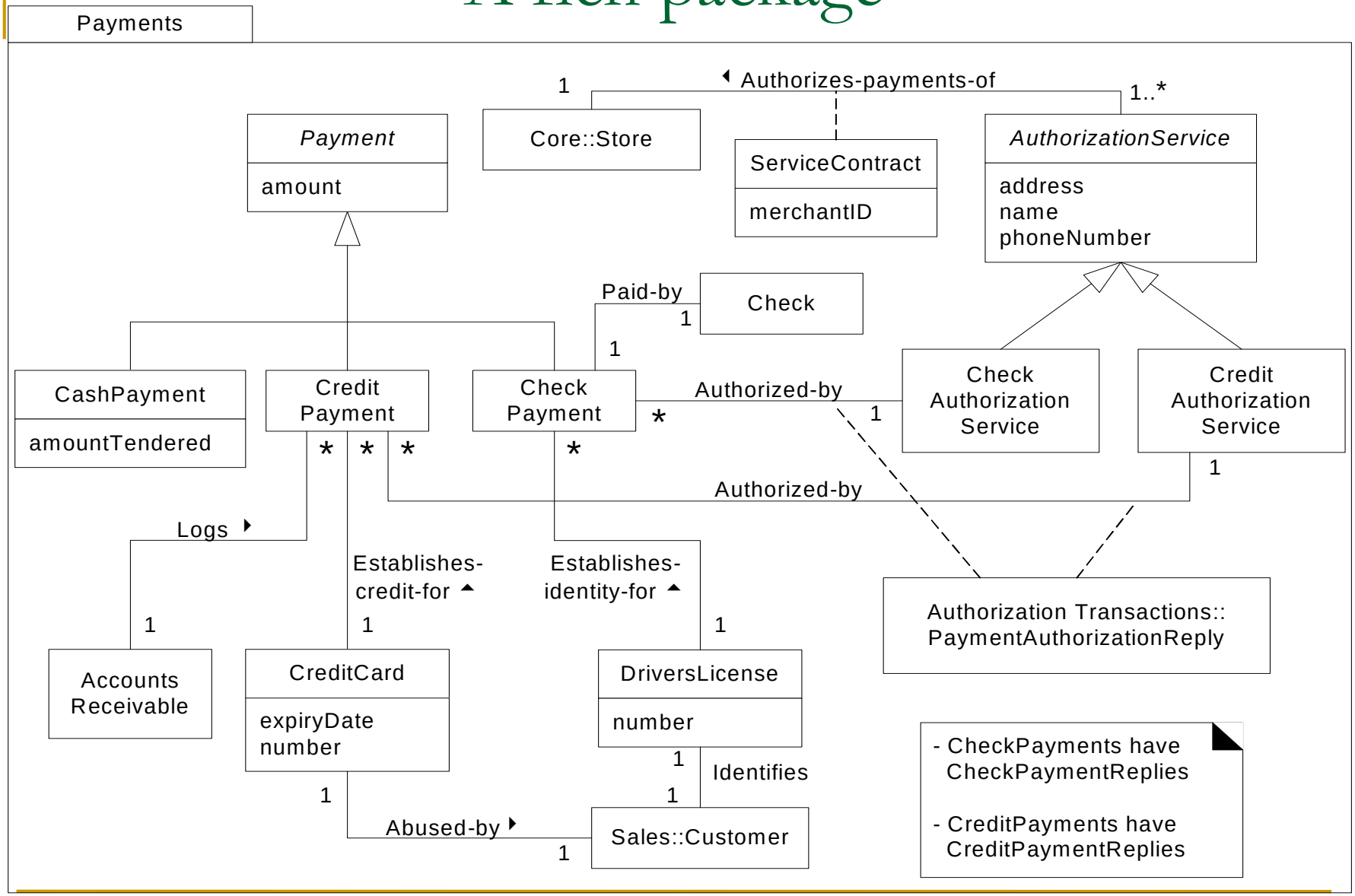


Fig. 31.31

Note: Composition and tie to Core package

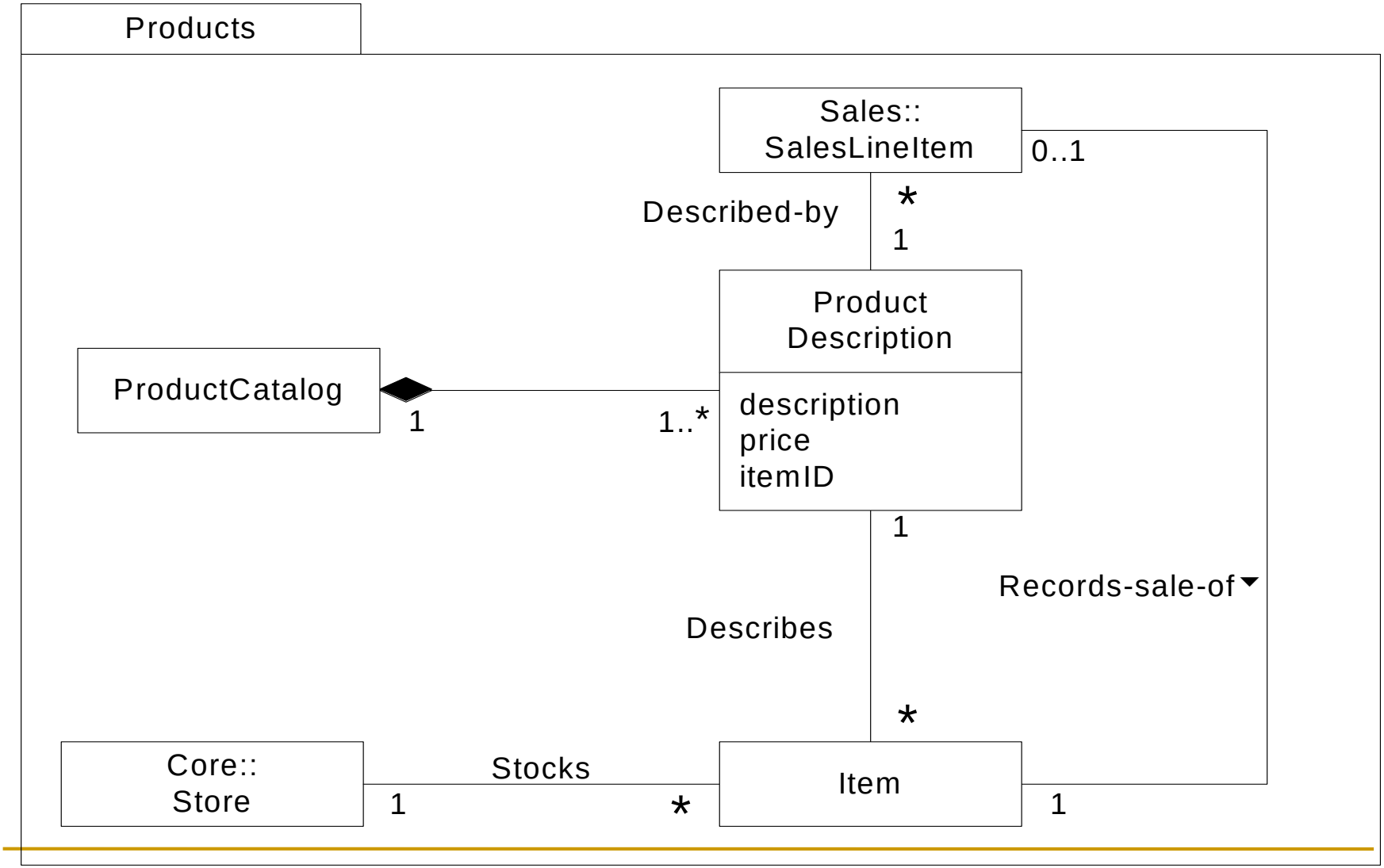
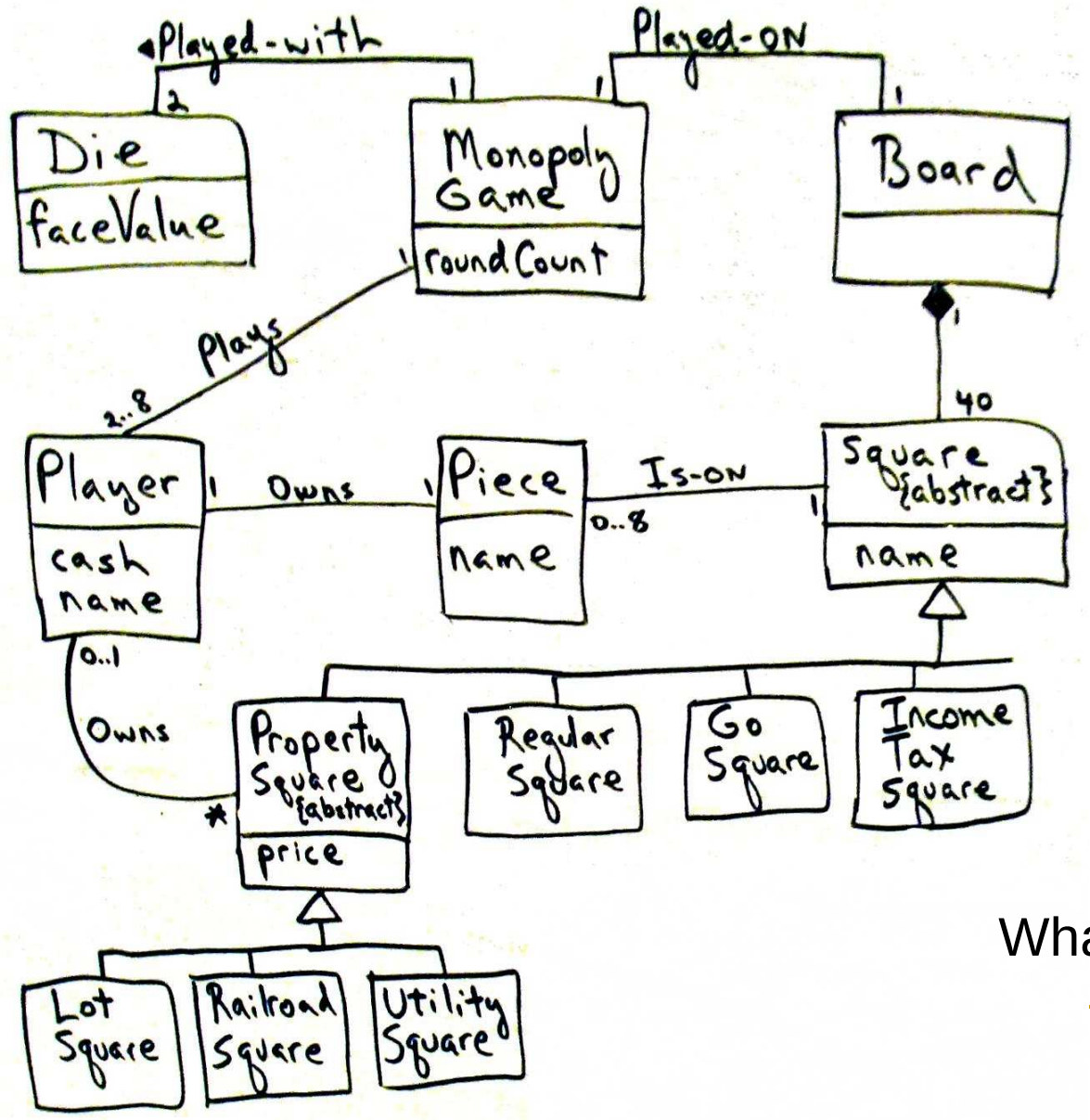


Fig. 31.32

Iteration-3 of Monopoly domain model



What's new?

Don’t forget to plan testing!

Plan testing for classes and system

- Write system and unit test plan descriptions as part of class design
- Unit test all public member functions
- Test for valid, invalid and boundary cases
- System testing follows thorough class testing
- See Junit tool for automated test generation support