THE UNIVERSITY OF MELBOURNE

SWEN30006
Software Modelling and Design

DOMAIN MODEL REFINEMENT

Larman Chapter 31

Crude classifications and false generalizations are the curse of the organized life.

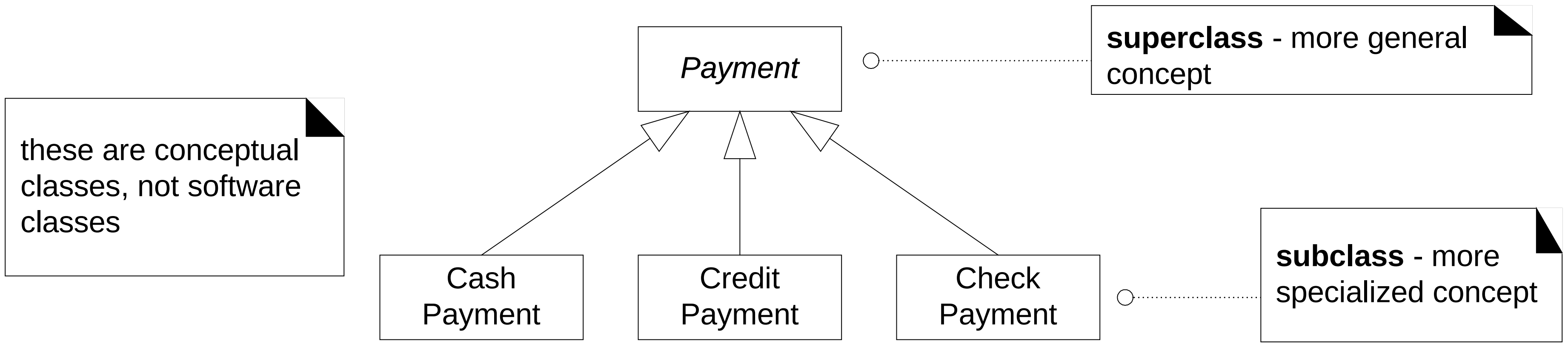
—A generalization by H.G. Wells

Objectives

On completion of this topic you should be able to:

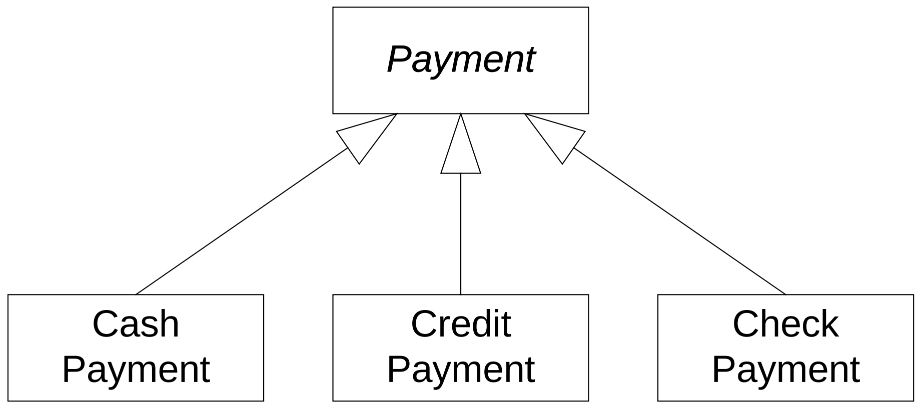
- Refine the domain model with generalizations, specializations, association classes, time intervals, composition, and packages.
- Identify when showing a subclass is worthwhile.

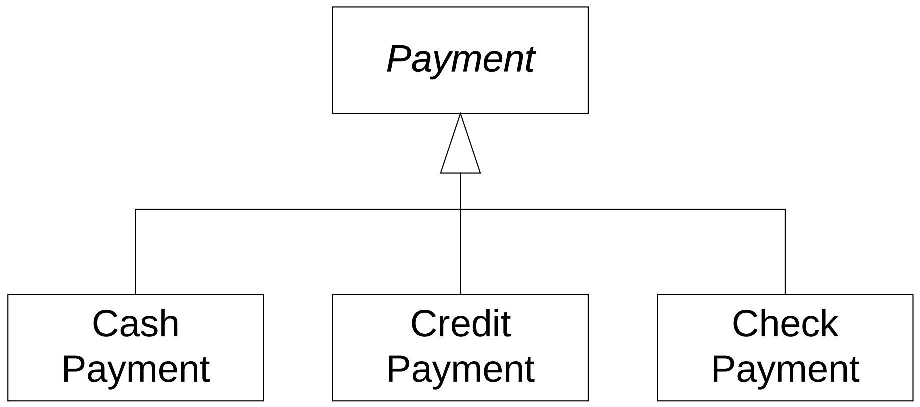
Generalization-Specialization Hierarchy



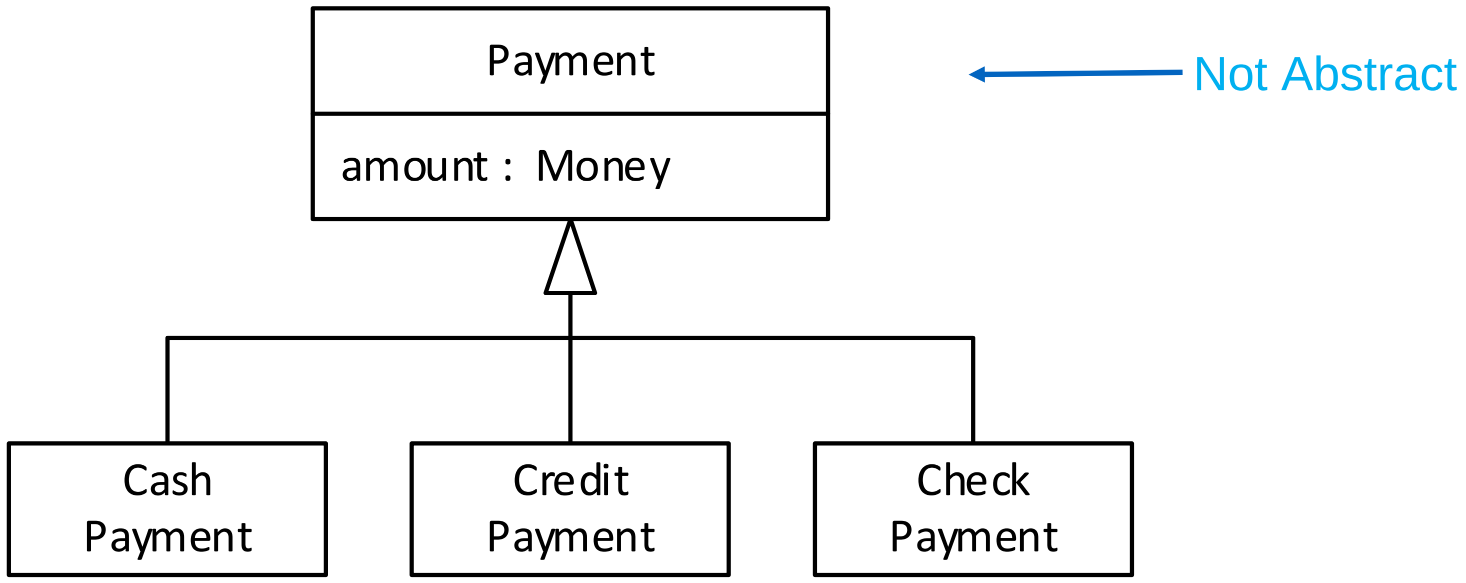
Payment
Cash Payment
Credit Payment
Check Payment
superclass - more general concept
subclass - more specialized concept
these are conceptual classes, not software classes

Notation: Separate/Shared Arrows



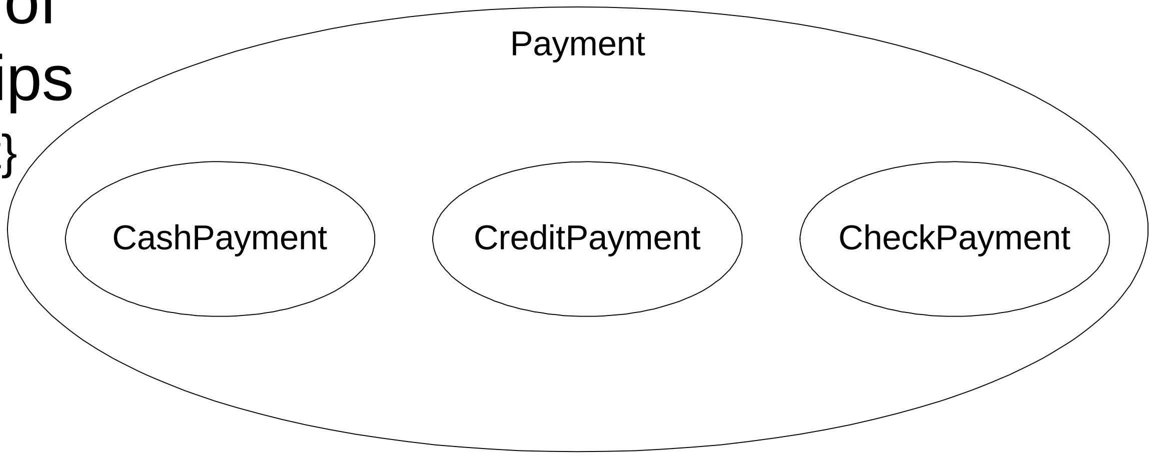


Payment Class Hierarchy

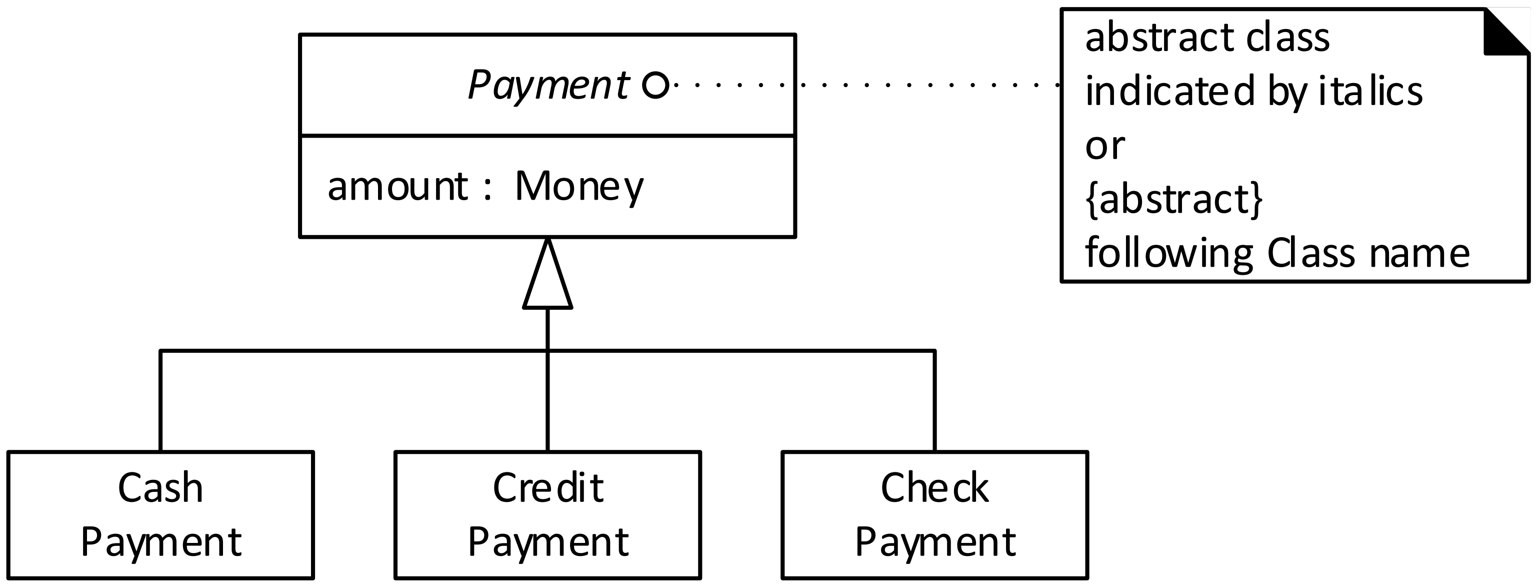


Venn Diagram of Set Relationships

{incomplete, disjoint}

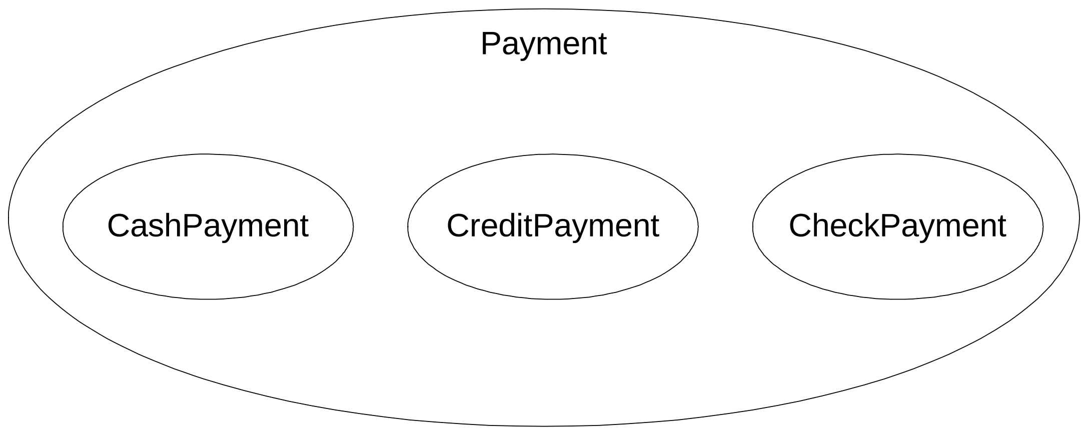


Abstract Class Notation



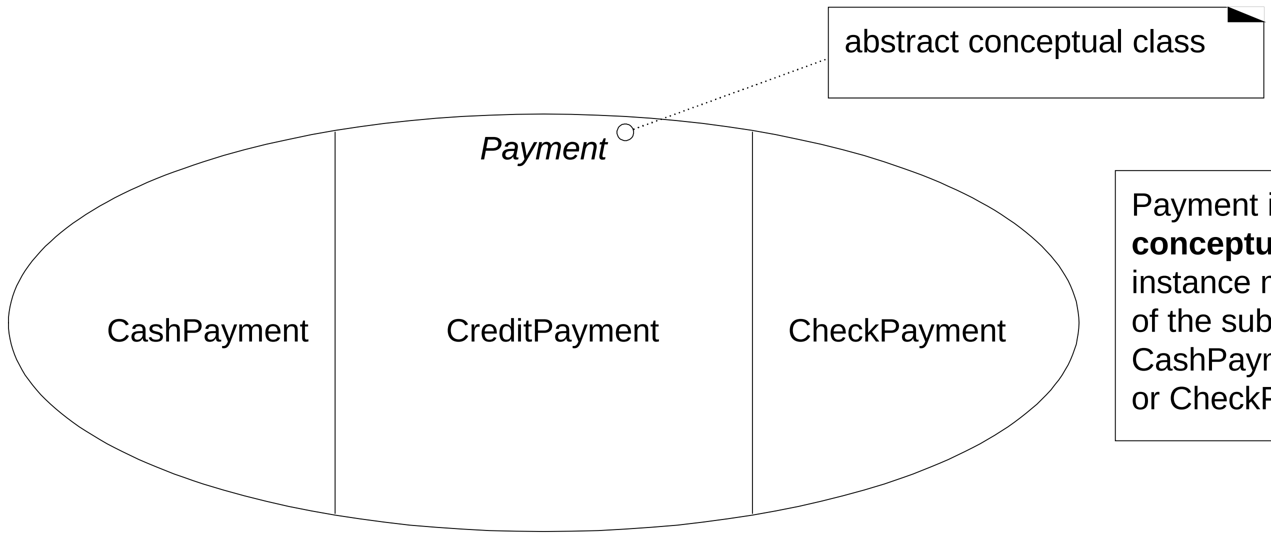
Abstract Conceptual Classes

(a)



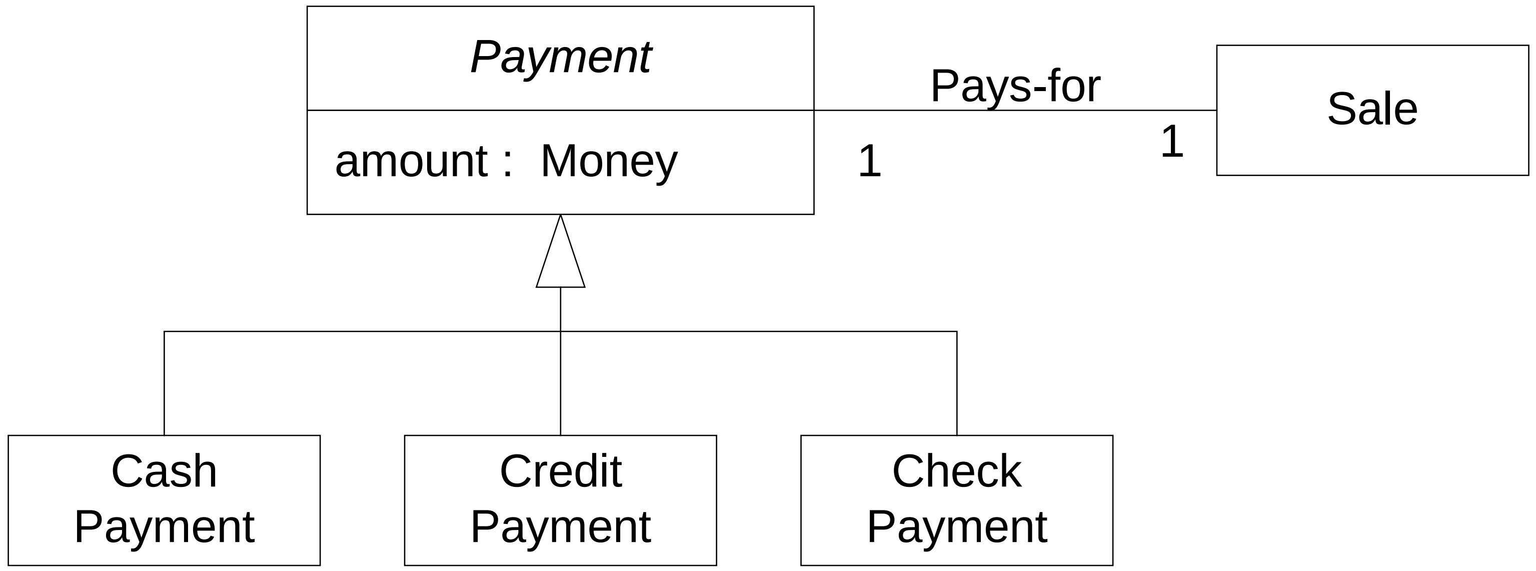
If a Payment instance may exist which is *not* a CashPayment, CreditPayment or CheckPayment, then Payment is not an abstract conceptual class.

(b)



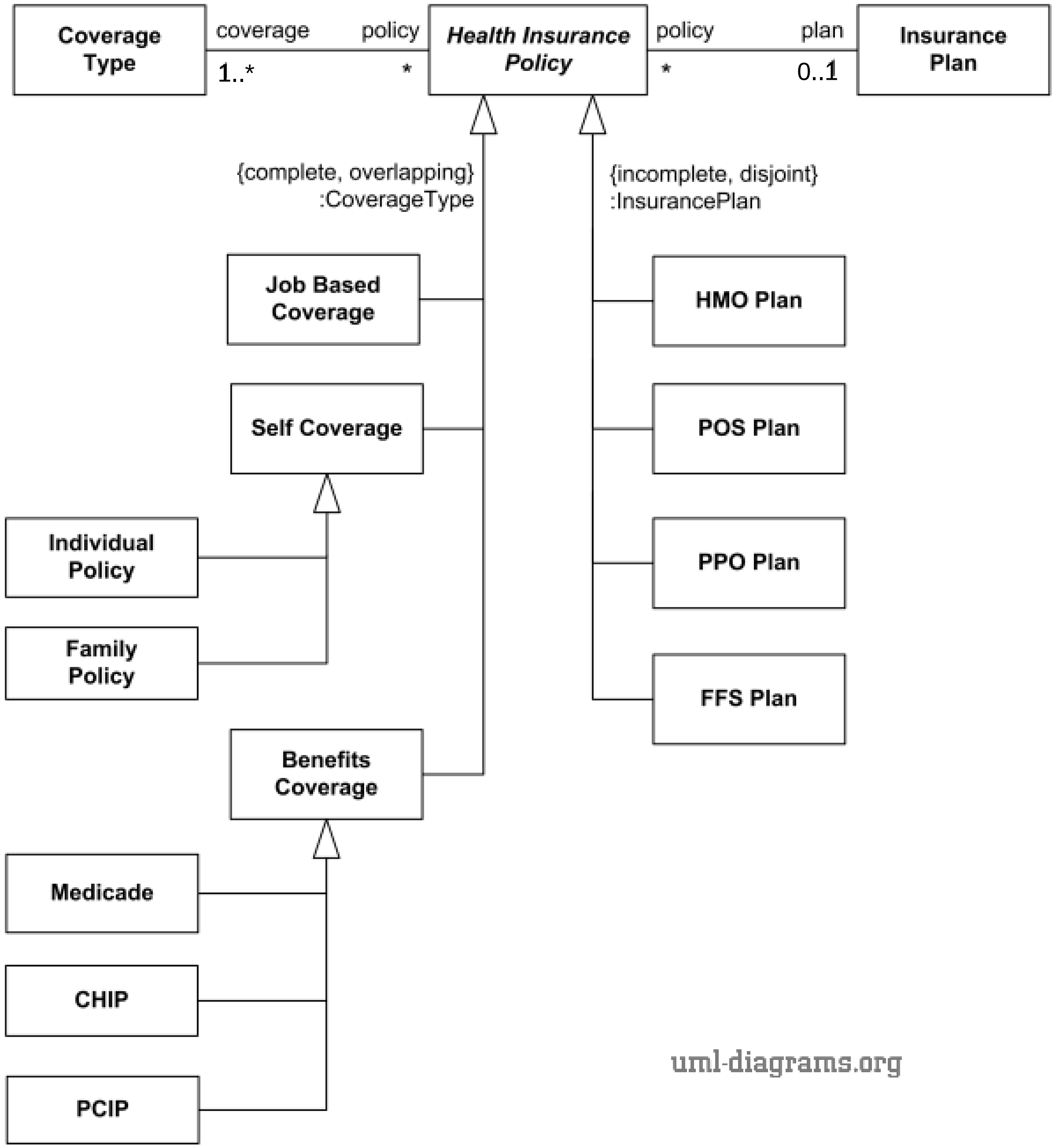
Payment is an **abstract conceptual class**. A Payment instance must conform to one of the subclasses: CashPayment, CreditPayment or CheckPayment.

Subclass Conformance

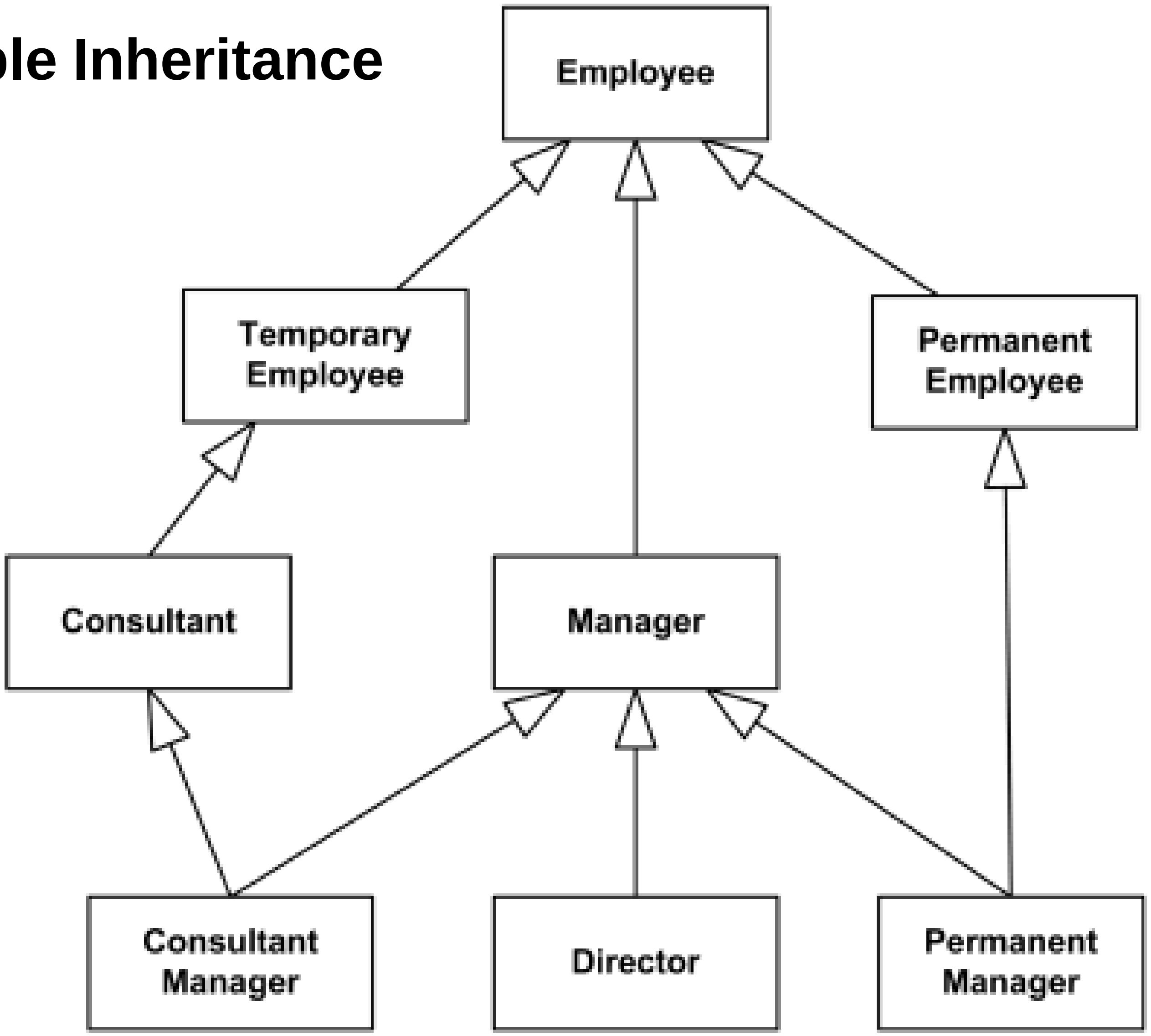


Generalization Sets

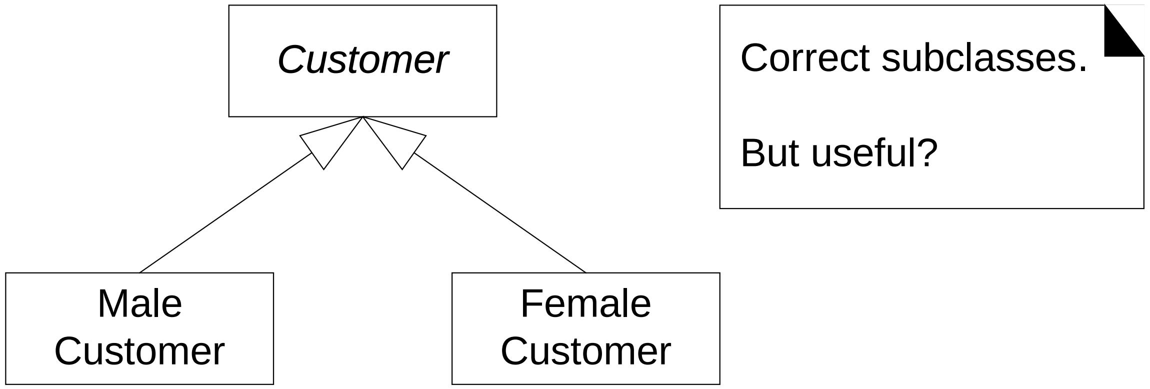
isCovering:
 complete or *incomplete*
isDisjoint:
 disjoint or *overlapping*
powerType (optional)
 - semantically
  equivalent association



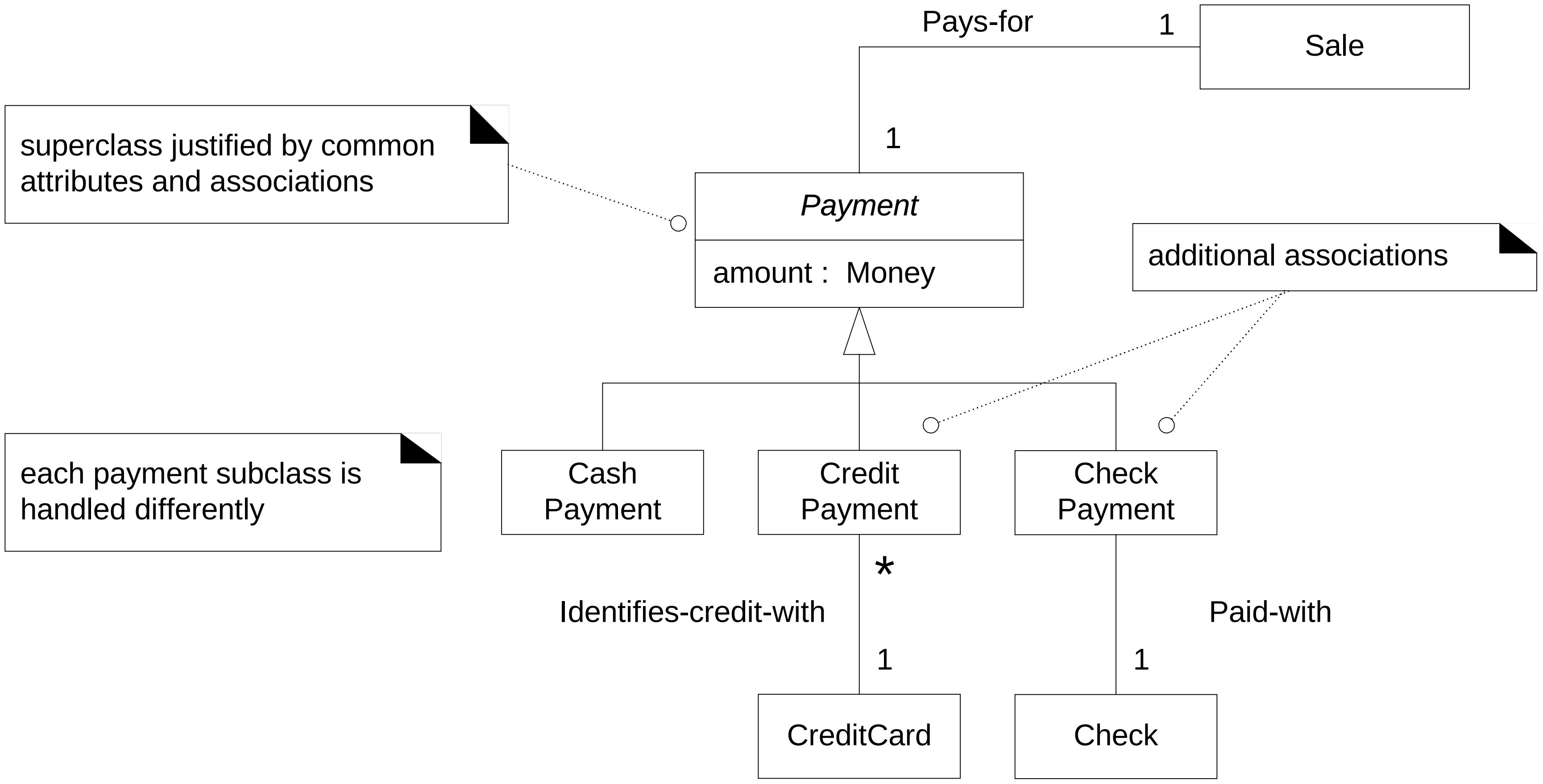
Multiple Inheritance



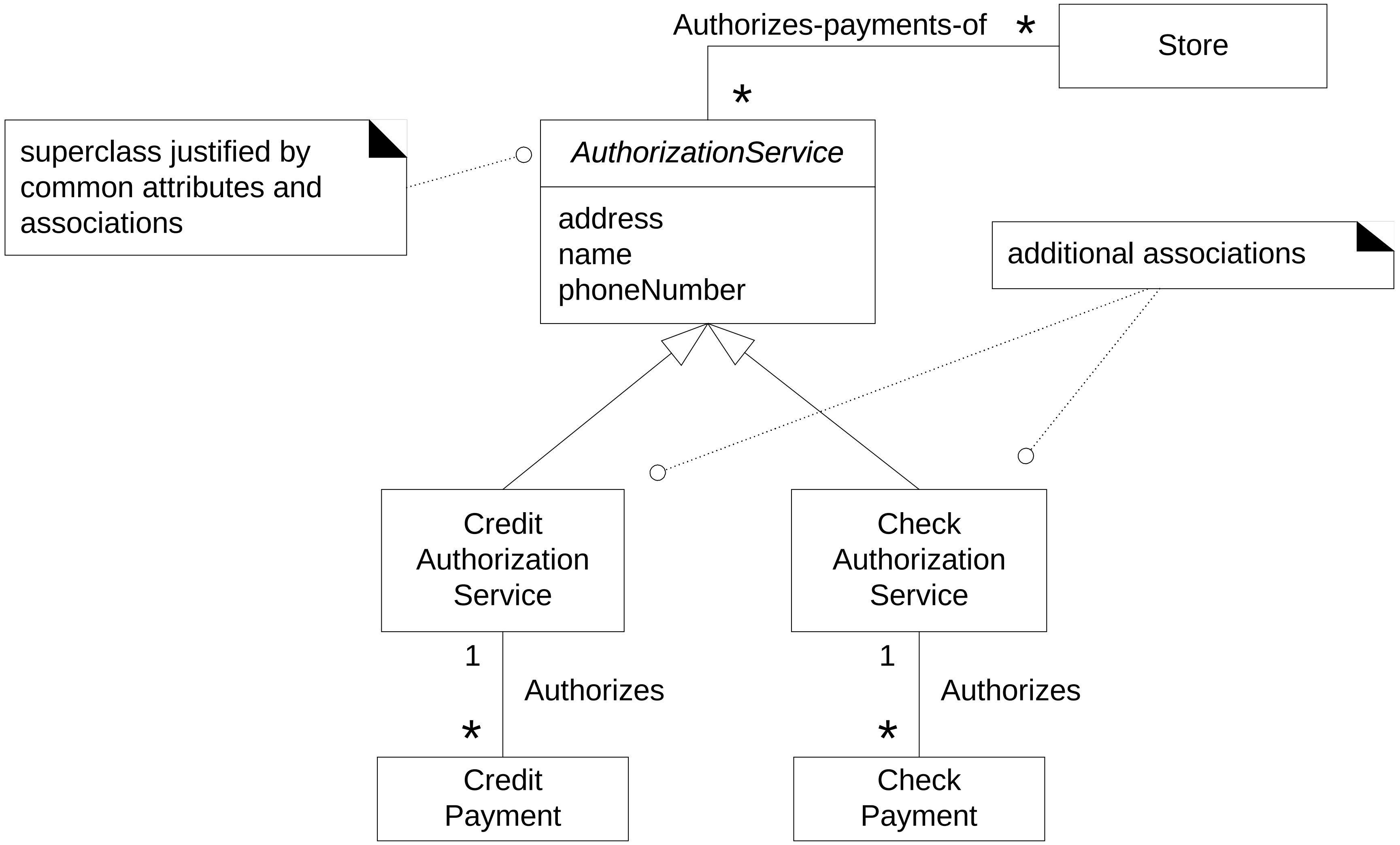
Legal, but is it Useful?



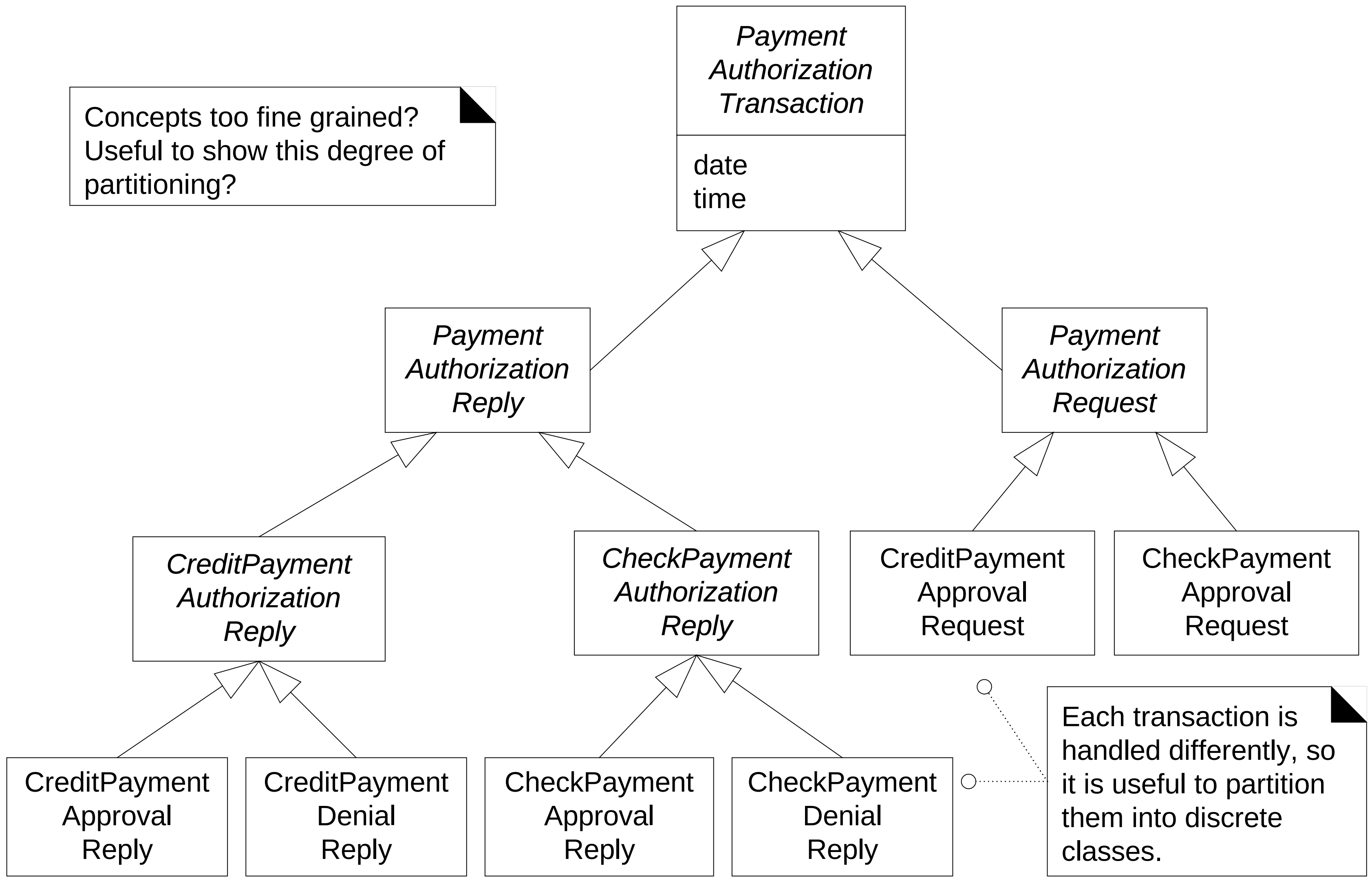
Justifying *Payment* Subclasses



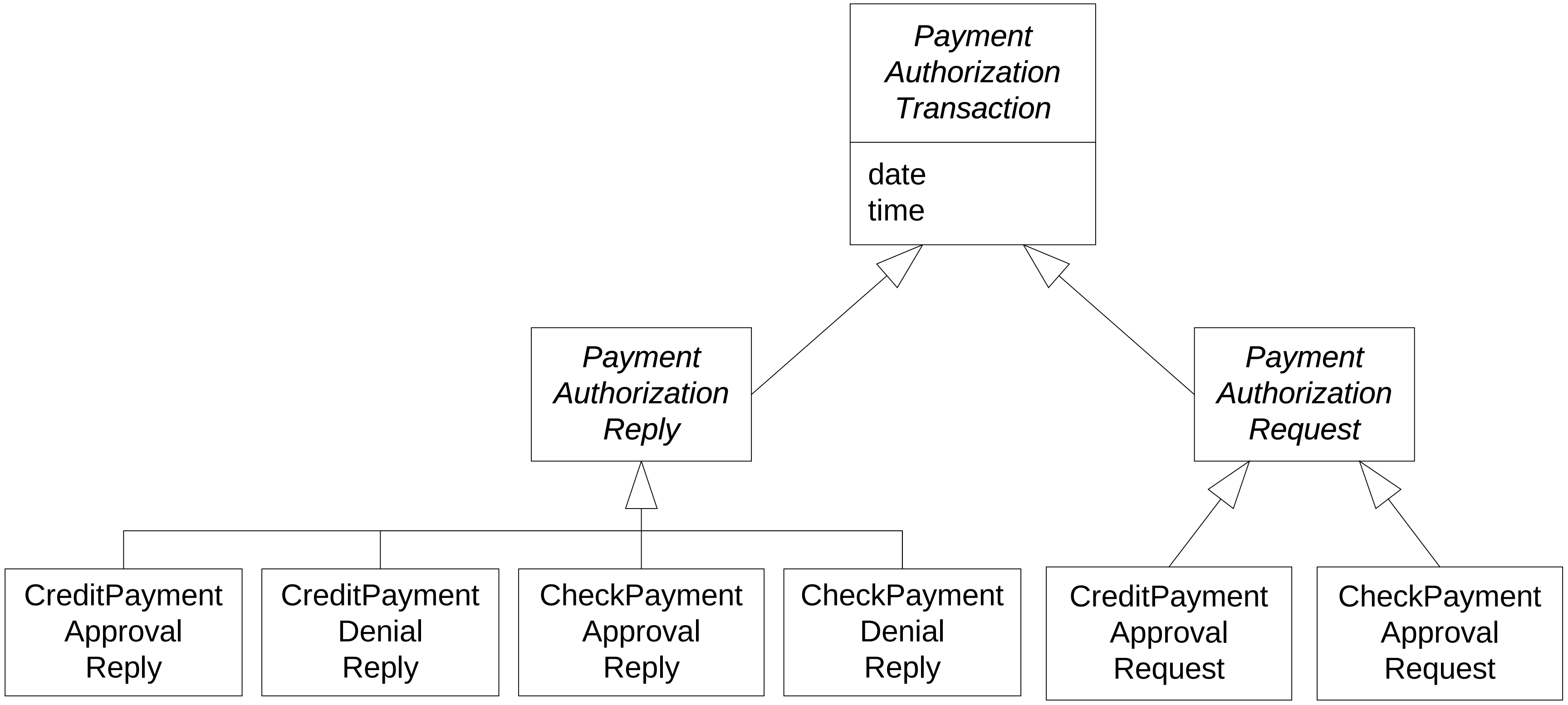
Justifying the AuthorizationService Hierarchy



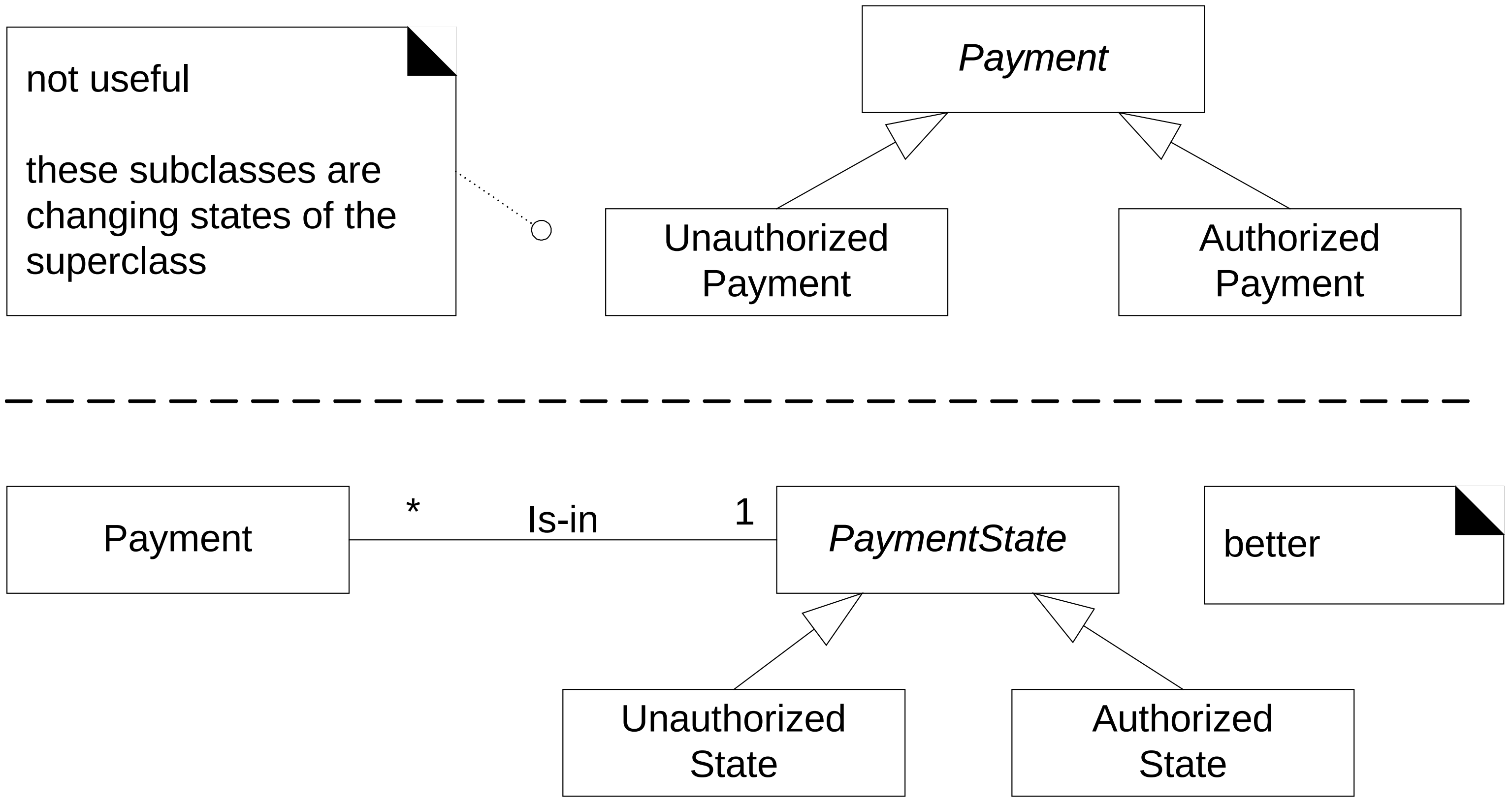
Possible Hierarchy: Transactions


Concepts too fine grained? Useful to show this degree of partitioning?
Payment Authorization Transaction
date
time
Payment Authorization Reply
Payment Authorization Request
CreditPayment Authorization Reply
CheckPayment Authorization Reply
CreditPayment Approval Request
CheckPayment Approval Request
CreditPayment Approval Reply
CreditPayment Denial Reply
CheckPayment Approval Reply
CheckPayment Denial Reply
Each transaction is handled differently, so it is useful to partition them into discrete classes.

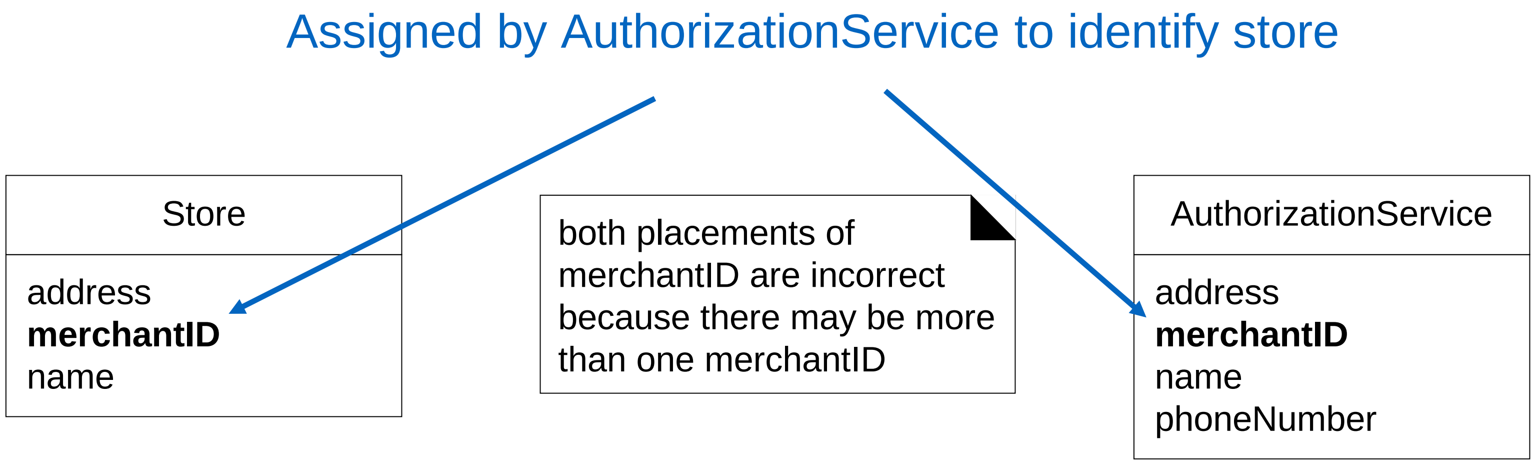
Alternate Hierarchy: Transactions


Payment Authorization Transaction
date
time
Payment Authorization Reply
Payment Authorization Request
CreditPayment Approval Reply
CreditPayment Denial Reply
CheckPayment Approval Reply
CheckPayment Denial Reply
CreditPayment Approval Request
CheckPayment Approval Request

Modelling Changing States



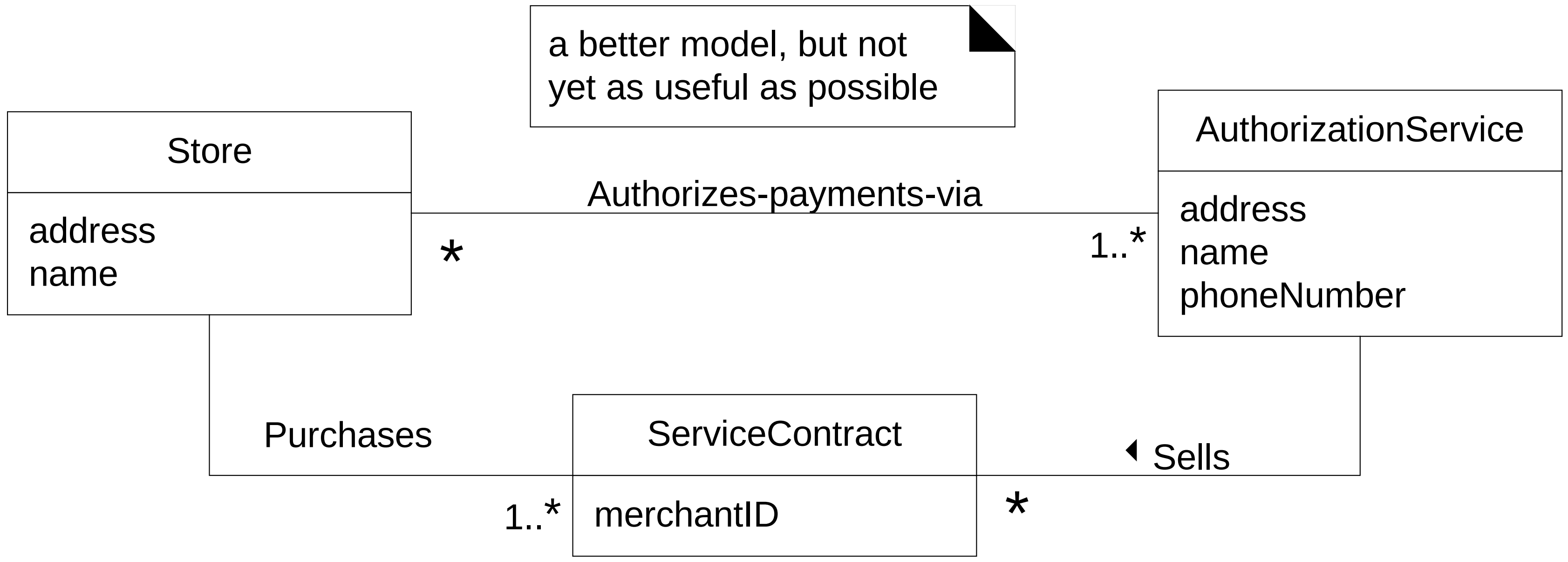
Inappropriate Use of an Attribute


Assigned by AuthorizationService to identify store
Store
address
merchantID
name
both placements of merchantID are incorrect because there may be more than one merchantID
AuthorizationService
address
merchantID
name
phoneNumber

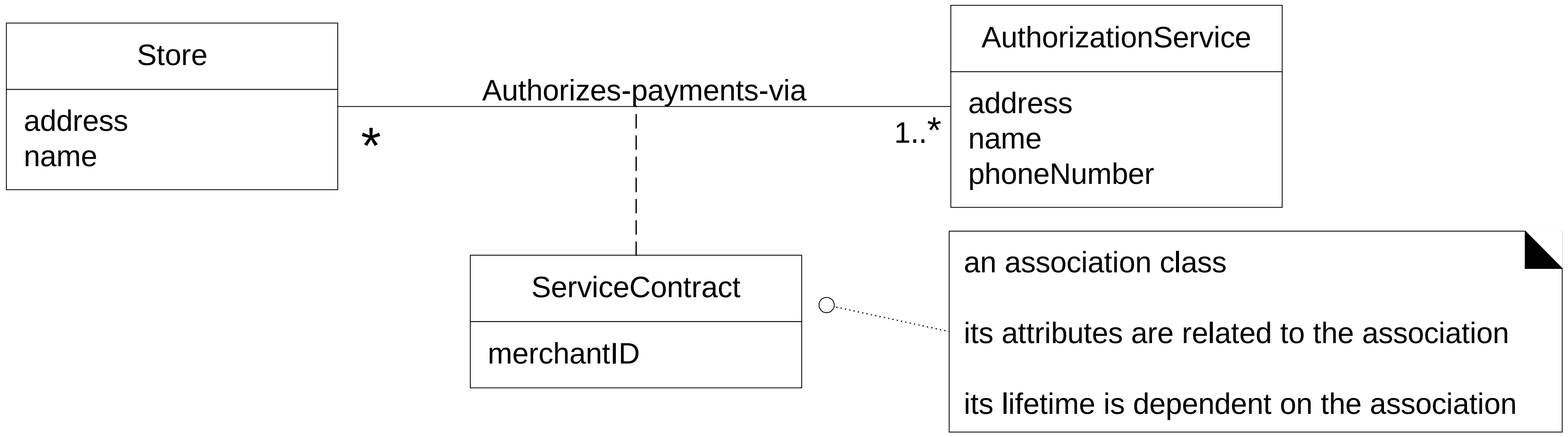
Guideline

In a domain model, if a class C can simultaneously have many values for the same kind of attribute A, do not place attribute A in C. Place attribute A in another class that is associated with C.

1st Attempt: *merchantID* Problem



Solution: *merchantID* Problem

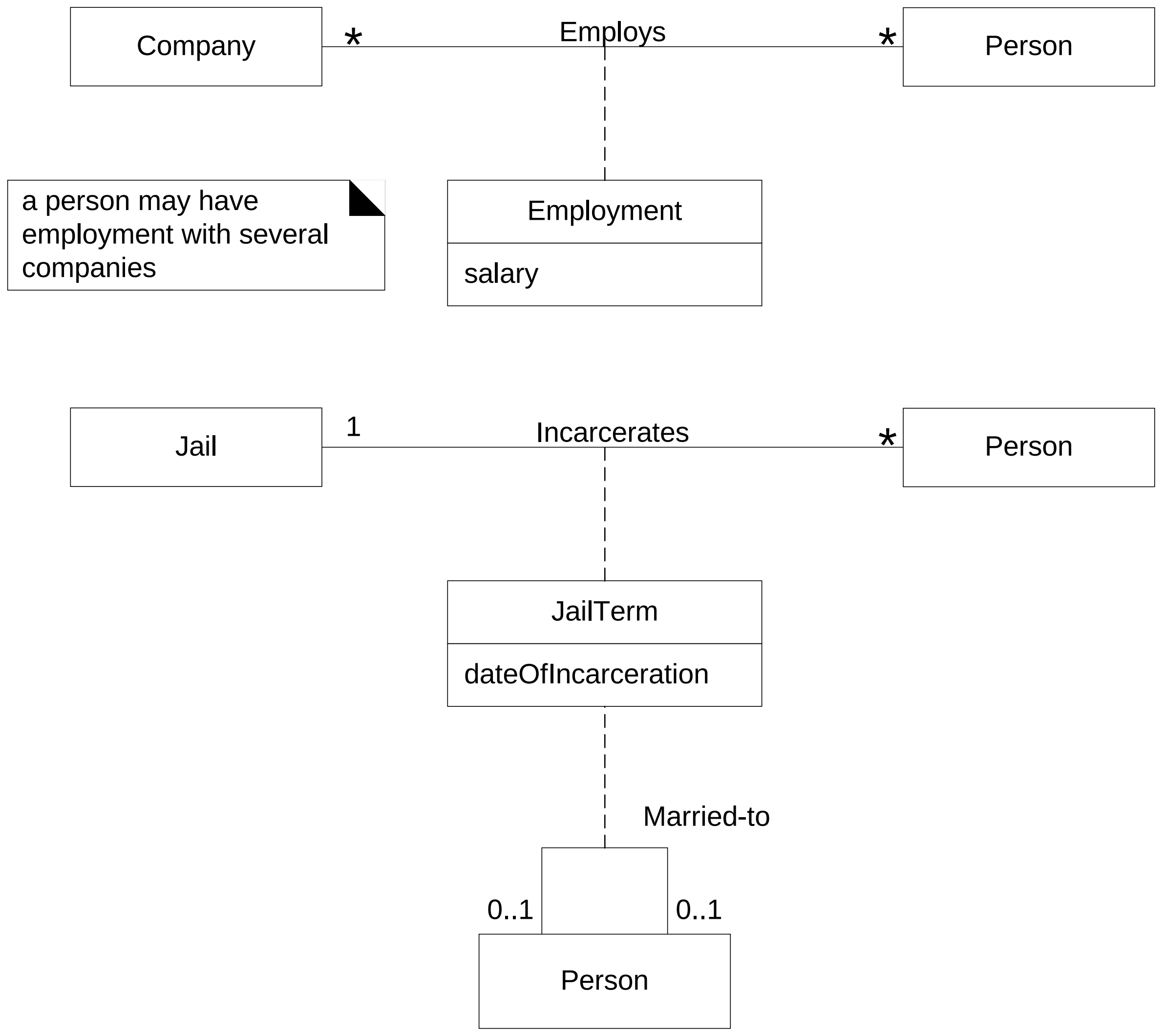


an association class

its attributes are related to the association

its lifetime is dependent on the association

Association Classes

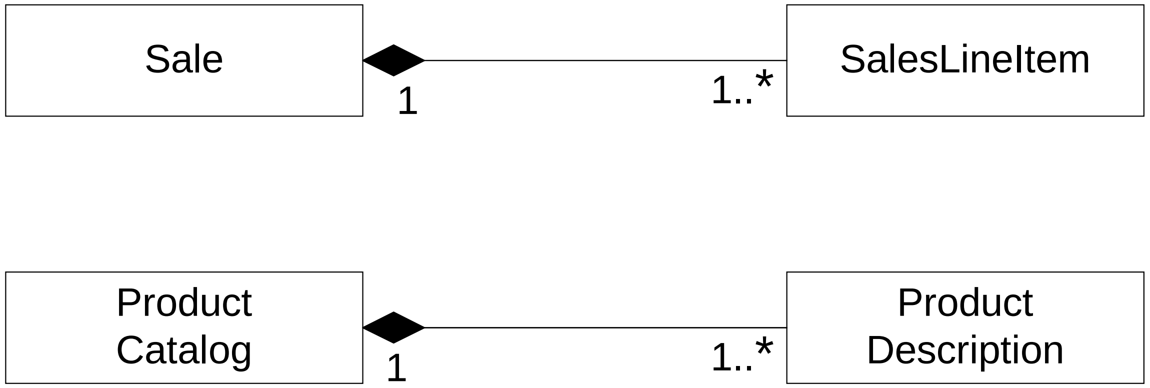


Guideline: Association Classes

Clues that a domain assoc. class might be useful:

1. An attribute is related to an association
2. Instances of (potential) assoc. class have lifetime dependency on the association
3. There is a many-to-many association between concepts, with other information to associate with the association.

Aggregation Examples: POS Application



Aggregation and Composition

Shared Aggregation (aka *Aggregation*)

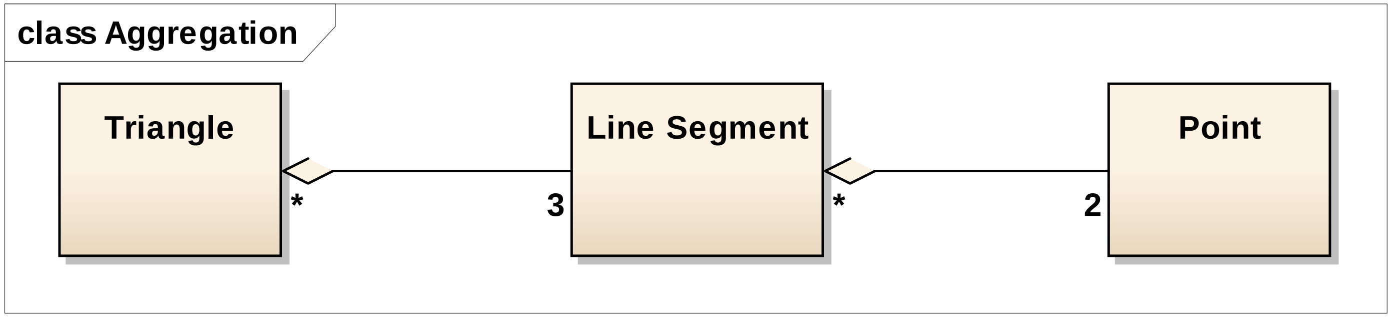
Composite Aggregation (aka *Composition*)

Common properties:

- binary associations
- asymmetric: only one end of association can be aggregation
- transitive: aggregation links should form a directed, acyclic graph; no composite instance can be indirect part of itself

Aggregation (shared)

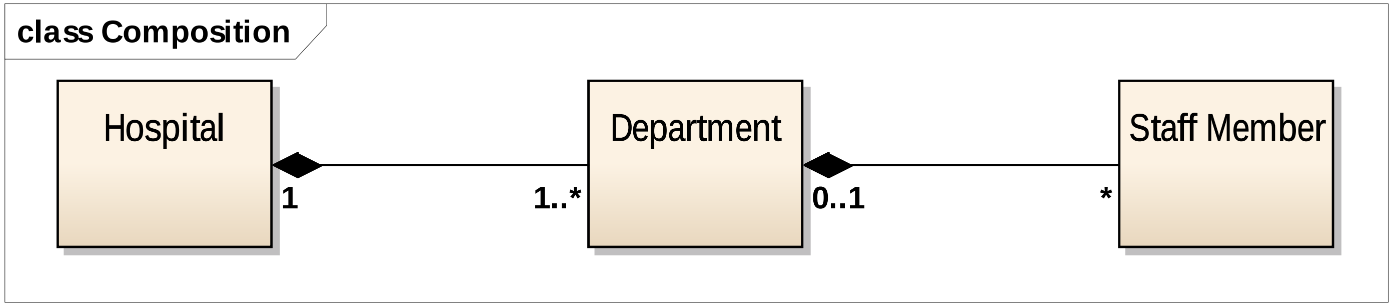
- shared part could be included in several composites; if some or all of the composites are deleted, shared part may still exist
- Kind of association that *loosely* suggests a whole-part relationship



- Guideline: *don't use shared aggregation*

Composition

- it is a *whole/part* relationship,
- a part could be included in *at most one* composite (whole) at a time, and
- if a composite (whole) is deleted, all of its composite parts may be deleted with it
  - Hospital: yes; Department: depends



Composition Guidelines

Consider composition when:

- Lifetime of part is bound to lifetime of composite
    - Create/Delete dependency
- Obvious whole-part physical or logical assembly
- Some properties of the composite propagate to the parts, e.g. location
- Operations applied to composite propagate to the parts, e.g. destruction, movement, recording

If in doubt, leave it out.

History

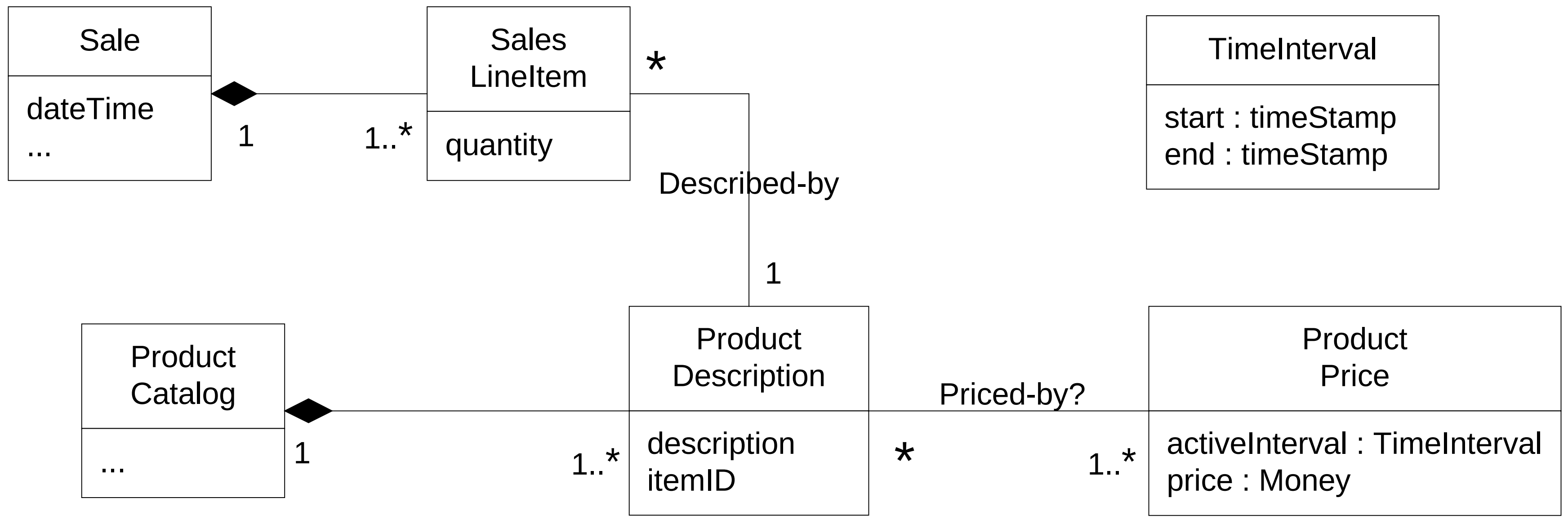
Scenarios:

1. I did not receive my tax receipt; please reissue.
2. What was the effect of pricing on sales?

History:

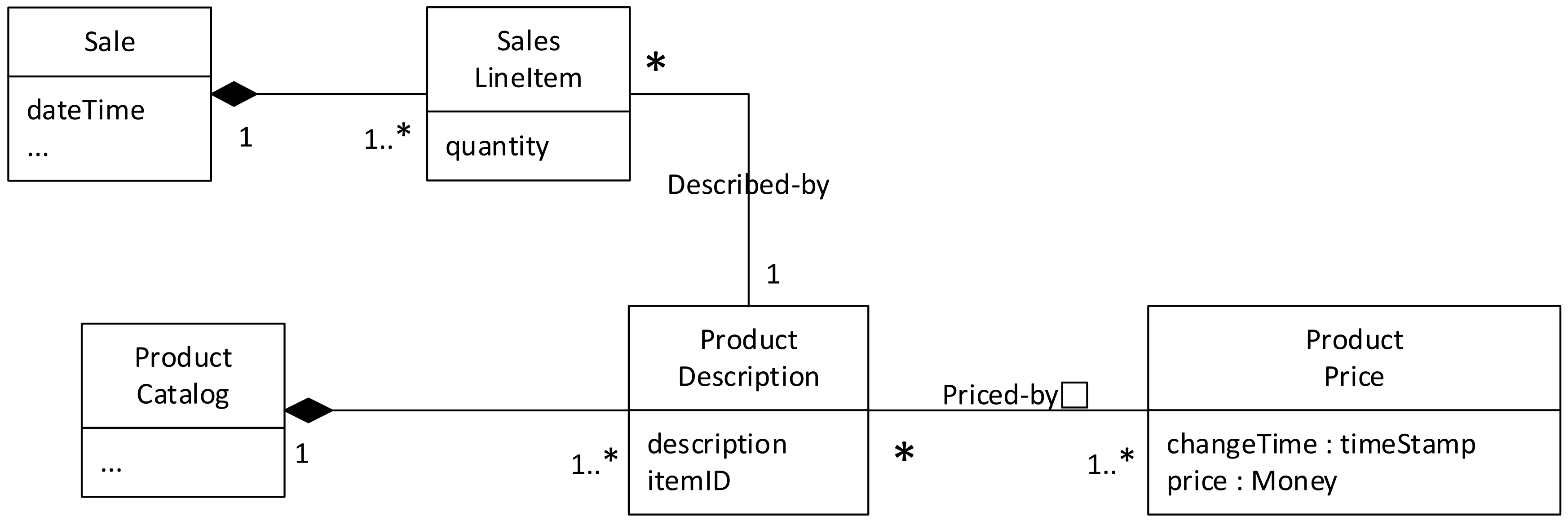
- Required when current information is insufficient
- May be a simple attribute (e.g. pricing) or may be a complex structure (catalogue, pricing, availability)
- Right structure for history depends on usage

Product Prices and Time Intervals



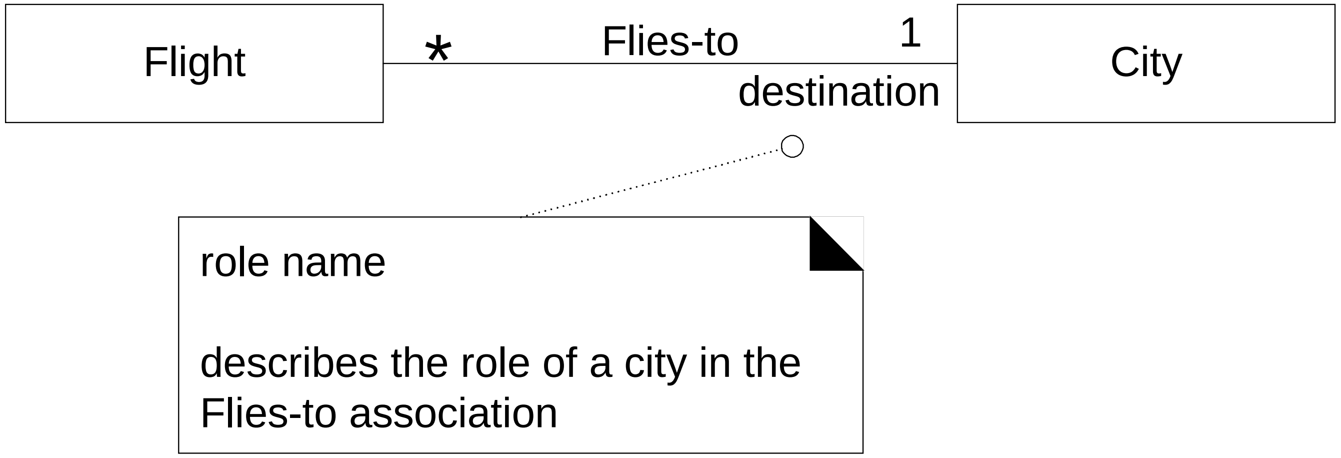
Product Price	activeInterval.start	activeInterval.end
$1.20	9:00am 1/1/2016	9:30am 1/1/2016
$1.50	9:30am 1/1/2016	10:00am 1/1/2016
$1.70	10:30am 1/1/2016	11:00am 1/1/2016
$1.20	10:45am 1/1/2016	11:30am 1/1/2016

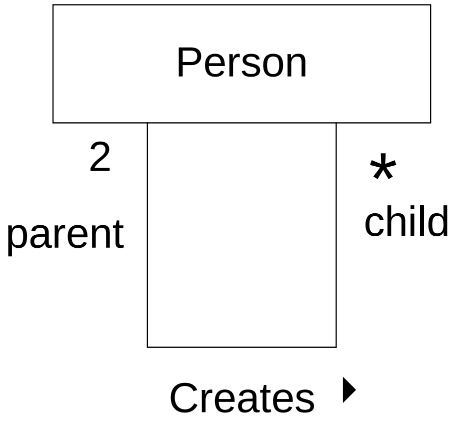
Product Prices and Change Time



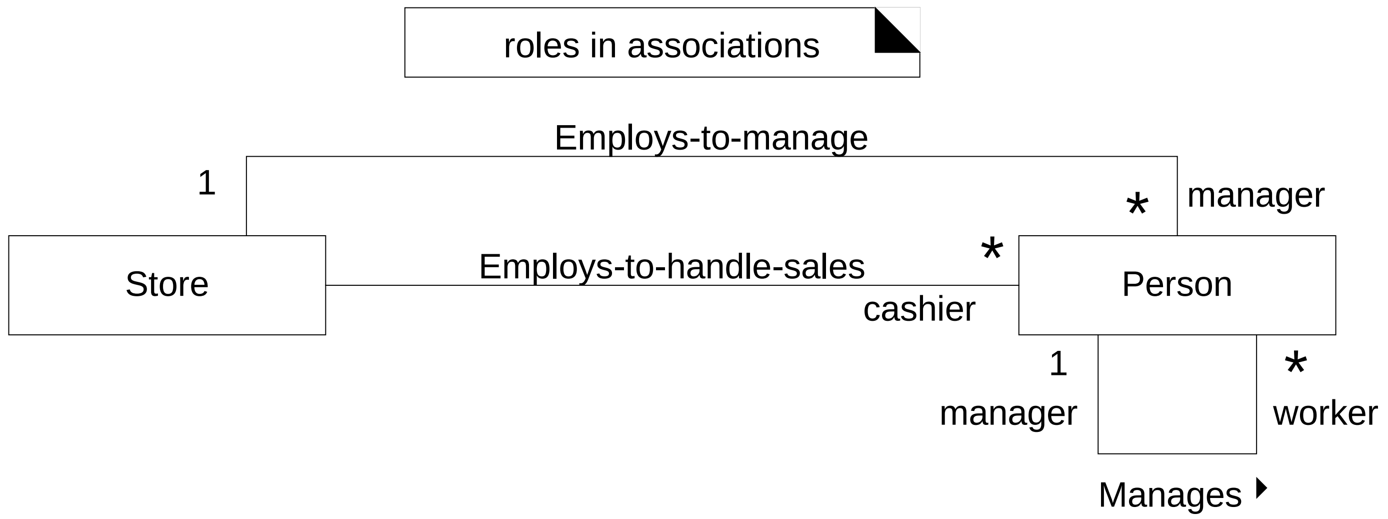
Product Price	changeTime
$1.20	9:00am 1/1/2016
$1.50	9:30am 1/1/2016
$1.70	10:30am 1/1/2016
$1.20	10:45am 1/1/2016

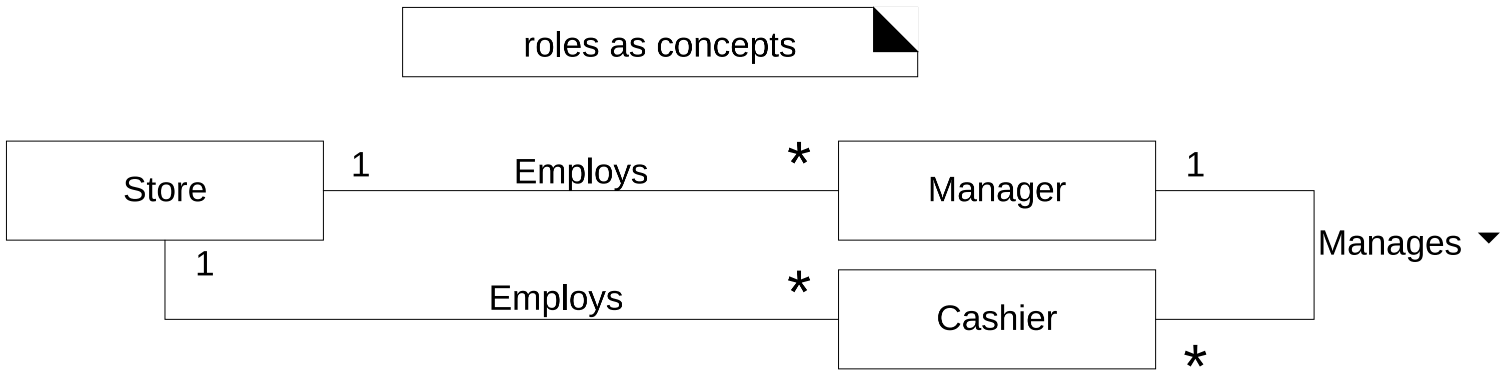
Role Names


Flight
*
Flies-to
1
destination
City
role name
describes the role of a city in the Flies-to association

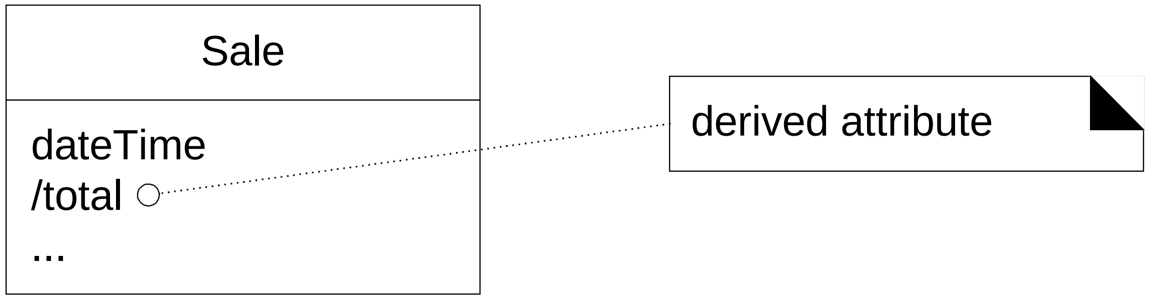

Person
2
parent
*
child
Creates

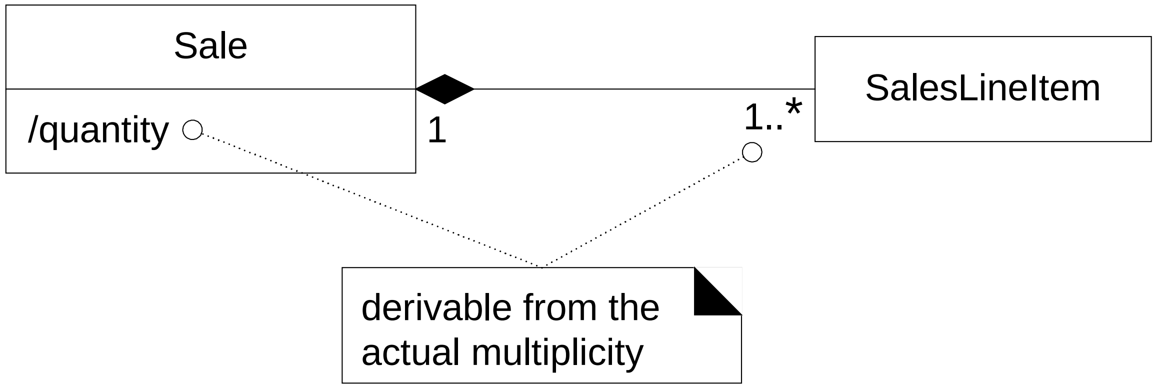
Two Ways to Model Human Roles





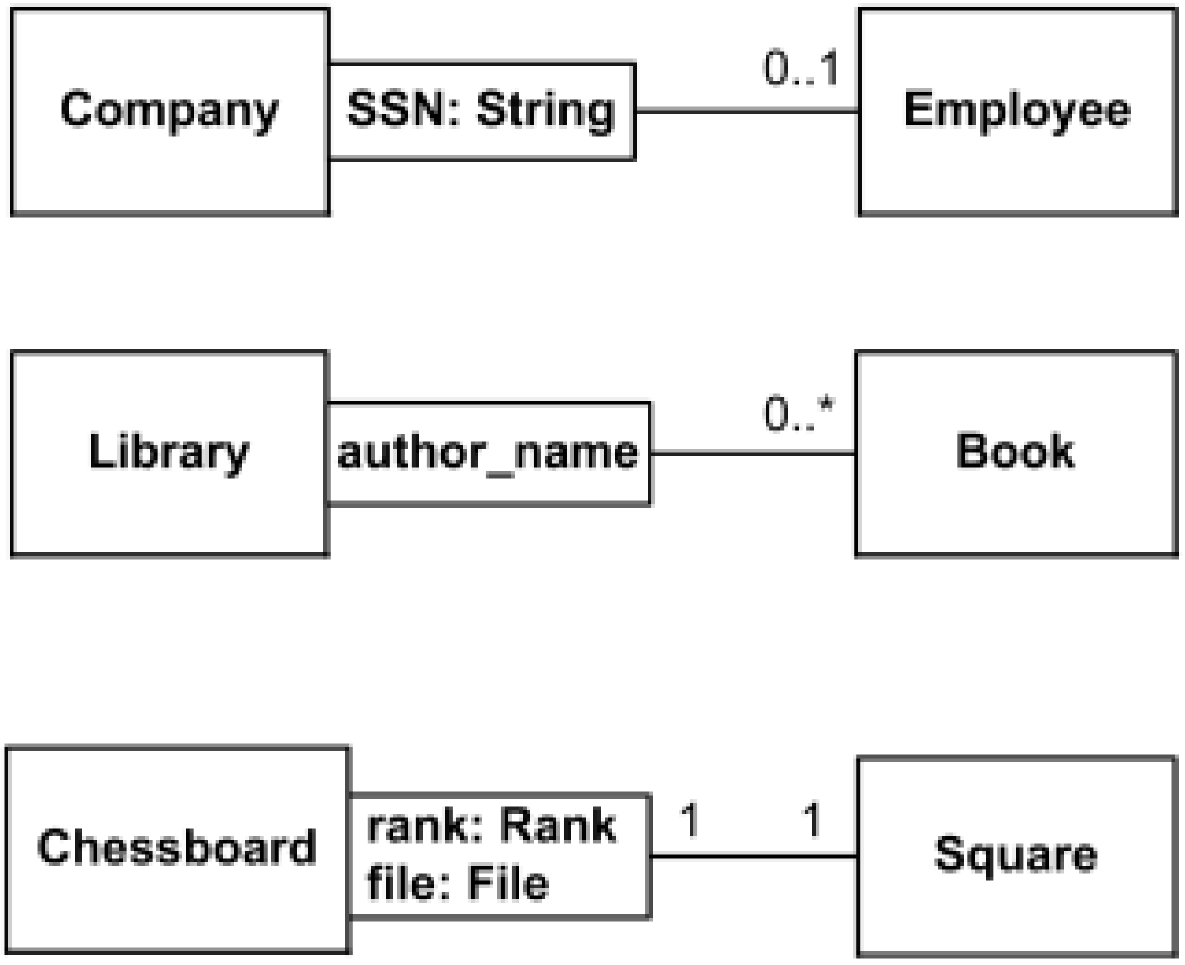
Derived Attributes



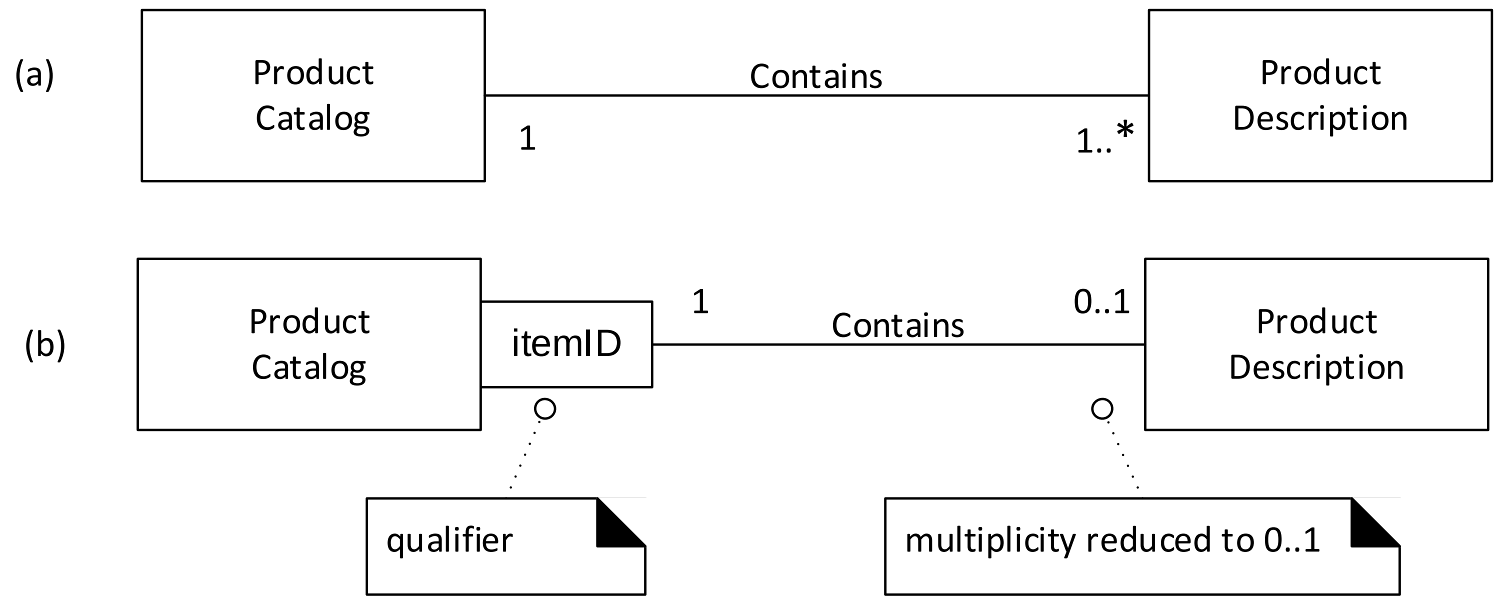


Guideline: Avoid unless defining prominent domain terminology

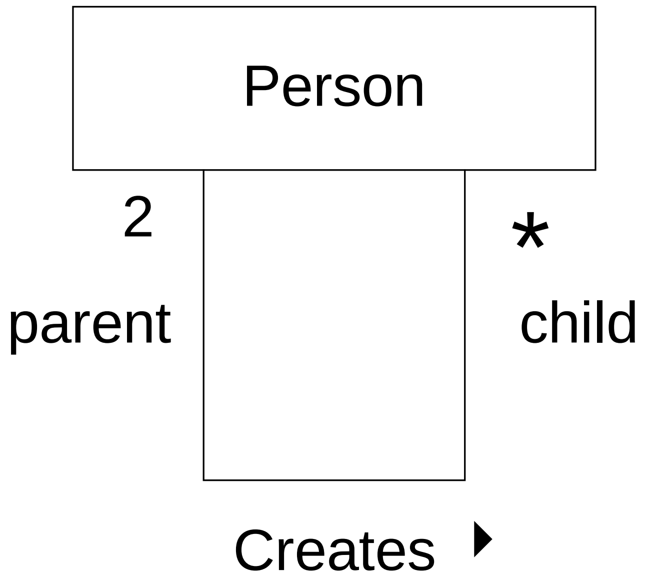
Qualified Association (1)



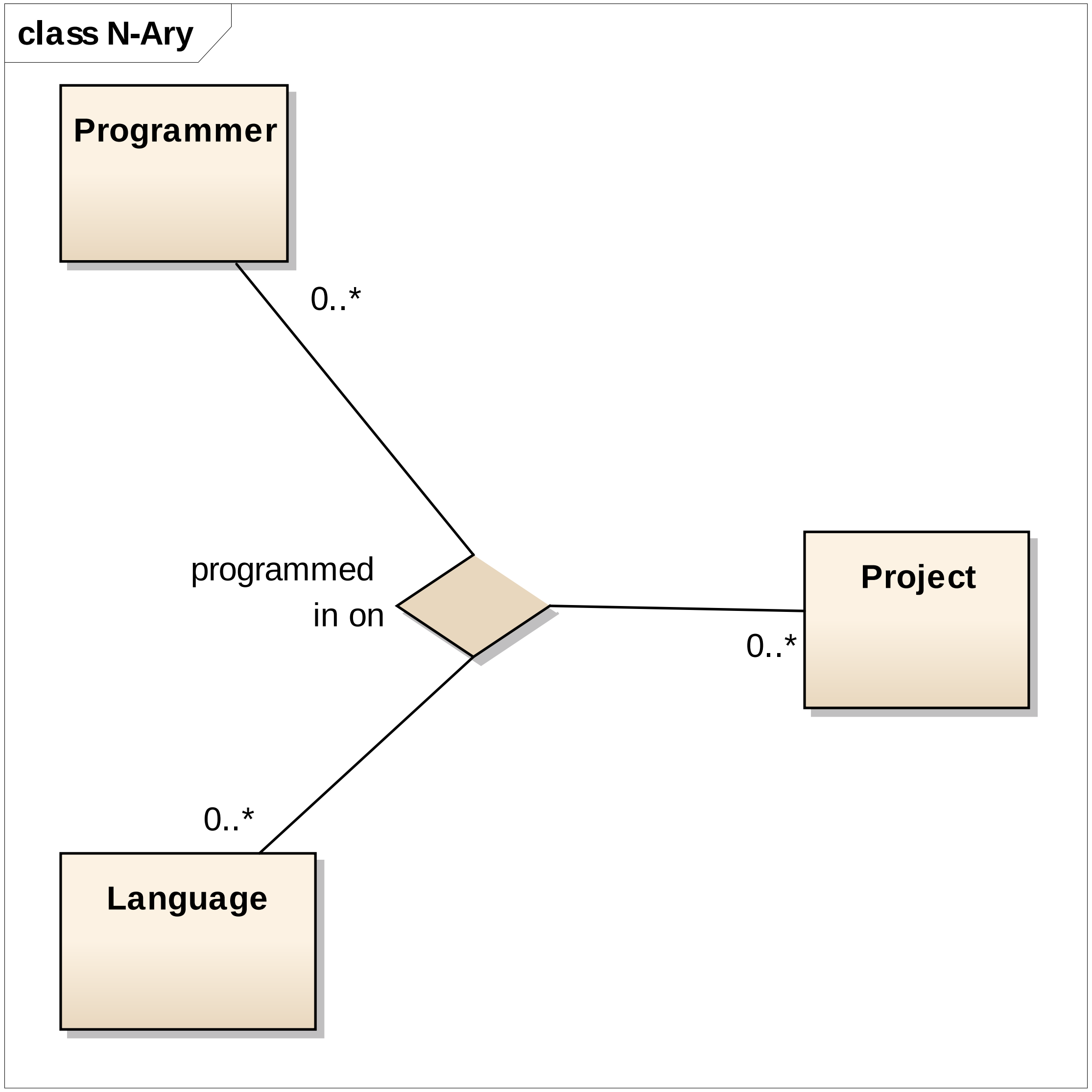
Qualified Association (2)



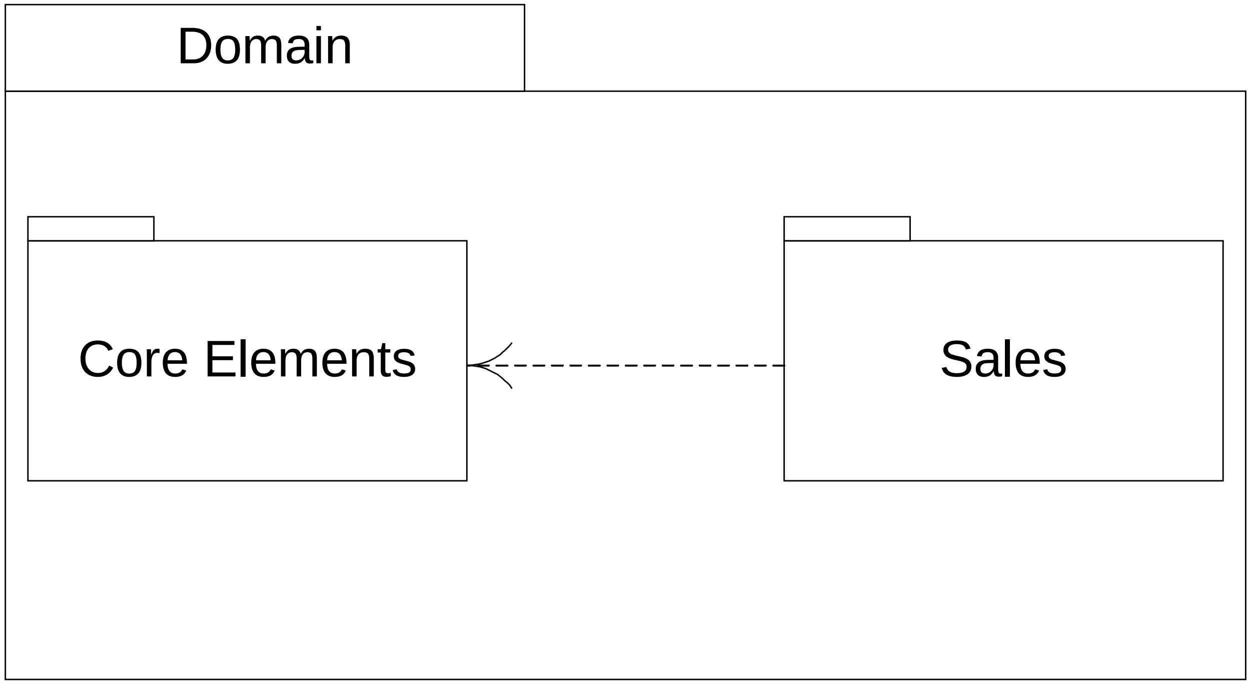
Reflexive Association



Ternary Association



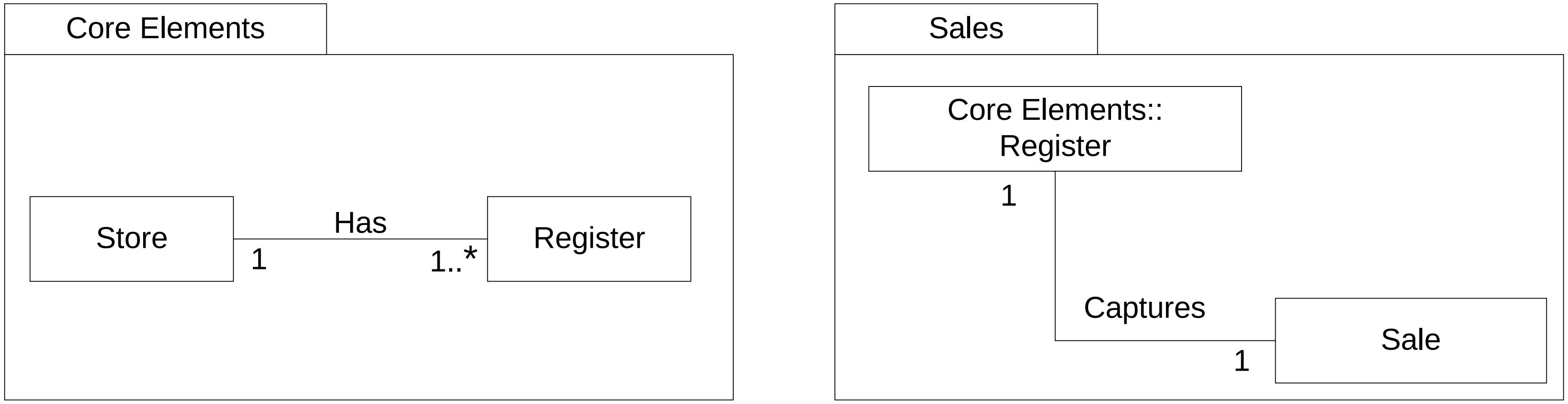
UML Packages: Organising the Domain Model



Package for top-level domain model with two subordinate packages.

Sales package shown to have a dependency on the Core Elements Package.

A Referenced Class in a Package



- Element is *owned* by the defining package.
- Element can be *referenced* in other packages
  - Qualified name: *PackageName::ElementName*
  - New associations but no other modifications

Partitioning the Domain Model

Guideline: Place elements together in a package that:

- are in the same subject area—closely related by subject or purpose
- are in a class hierarchy together
- participate in the same use cases
- are strongly associated

and:

- rooted in a package called *Domain*
- consider a package for widely shared concepts