Modeling Generalization

Software Design and Analysis

CSCI 2040

Objectives

- Create generalization-specialization hierarchies.
  - Identify when showing a subclass is worthwhile.
  - Apply the "100%" and "Is-a" tests to validate subclasses.
  - Add aggregation relationships.
  - Choose how to model roles.

Generalization

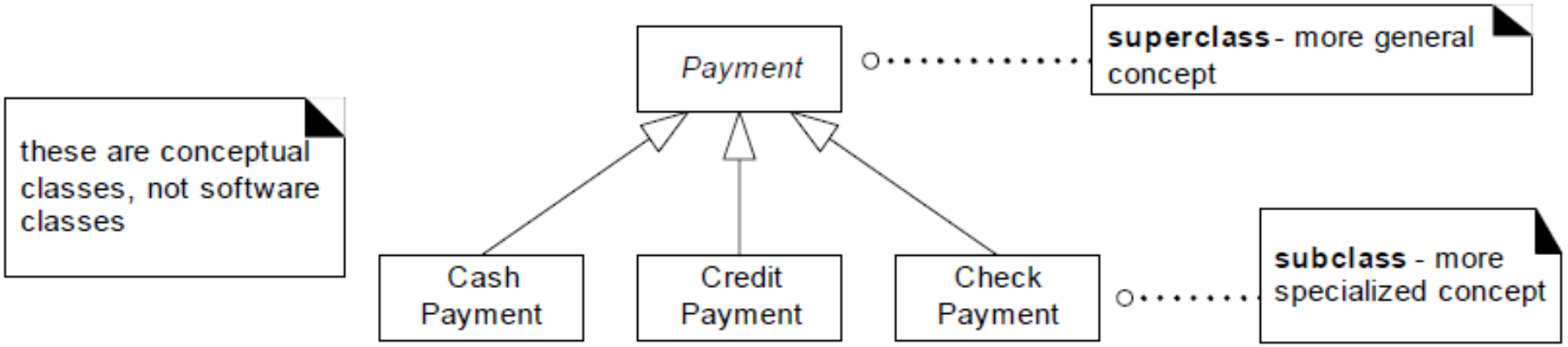
- **Generalization** is the activity of identifying commonality among concepts
  - defining superclass (general concept) and
  - subclass (specialized concept) relationships.
- It is a way to construct taxonomic classifications among concepts which are then illustrated in class hierarchies.

Introduction

- **Generalization and specialization** are fundamental concepts in domain modeling
  - that exploit inheritance and reduce duplication of code.

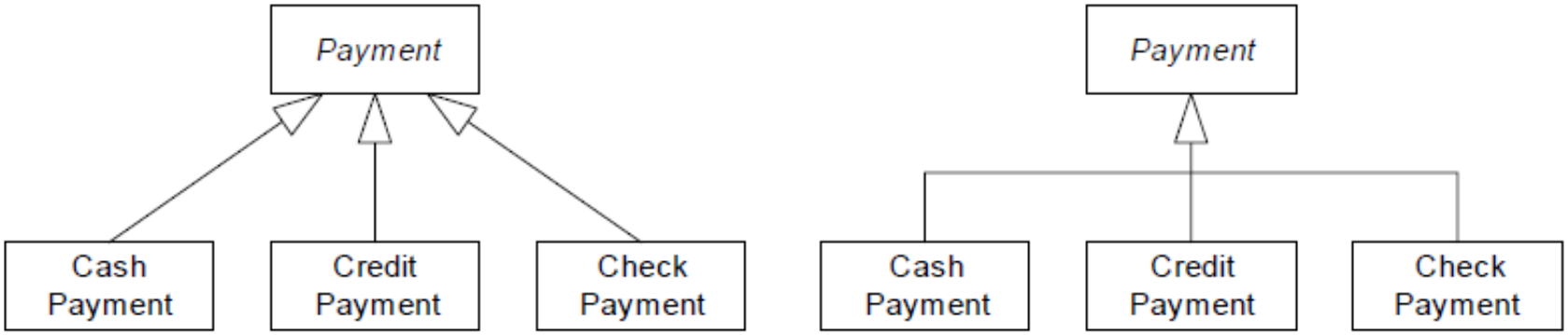
Generalization

- The concepts *CashPayment, CreditPayment,* and *Check Payment* are all very similar.
  - It is possible to organize them into a **generalization-specialization class hierarchy**
    - in which the **superclass** *Payment* represents a more general concept, and the **subclasses** more specialized ones.



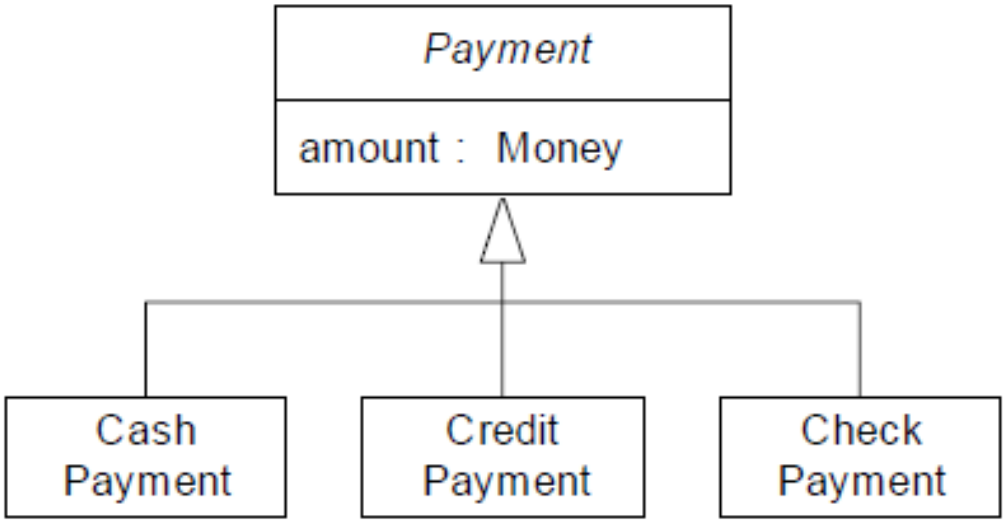
Separate and Shared Arrow Notations

- Either a separate target or shared target arrow style may be used.



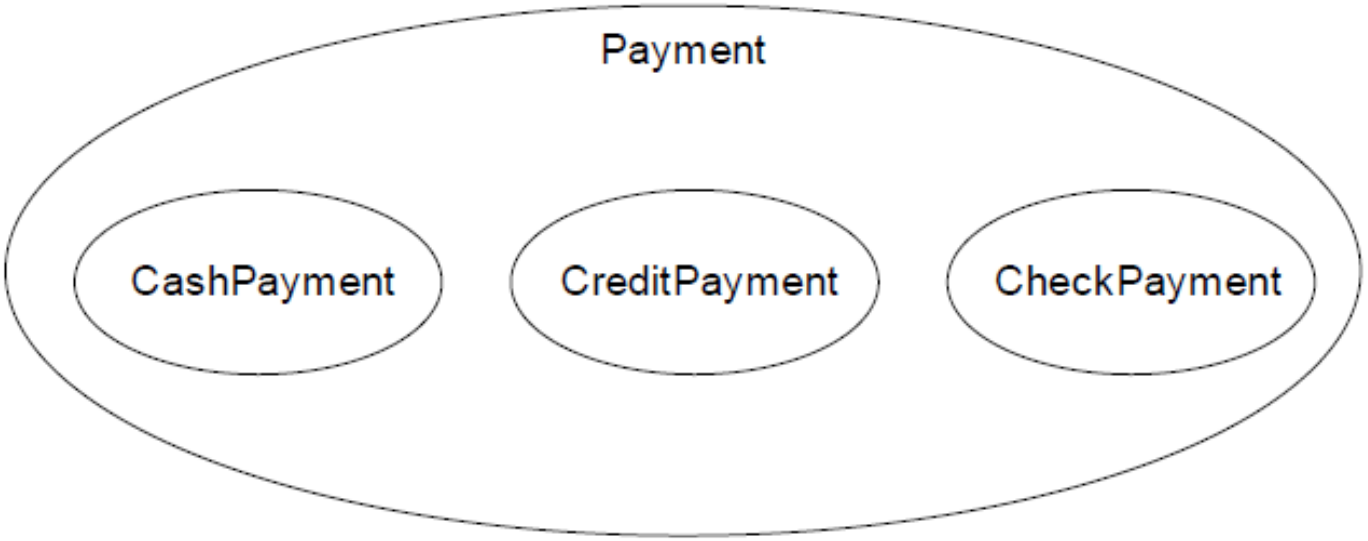
Payment Class Hierarchy

- All payments have an amount of money transferred.



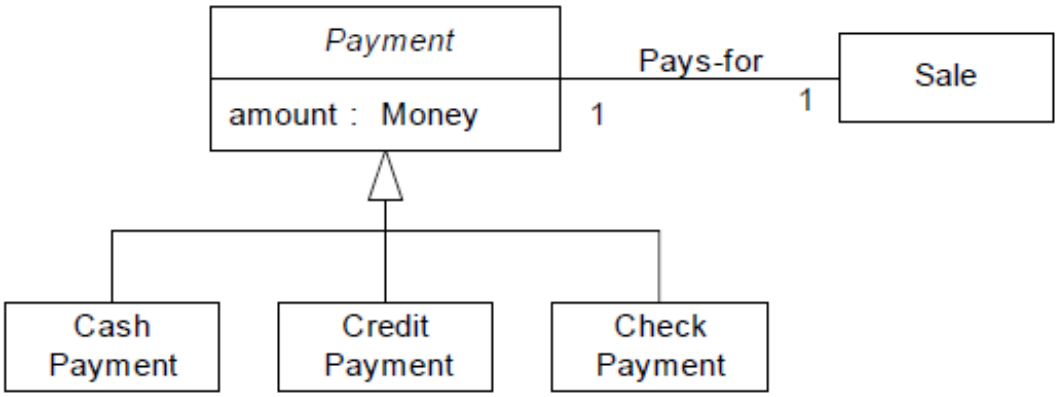
Venn Diagram

- Conceptual subclasses and superclasses are related in terms of set membership.
- **All the members of a conceptual subclass set are members of their superclass set.**

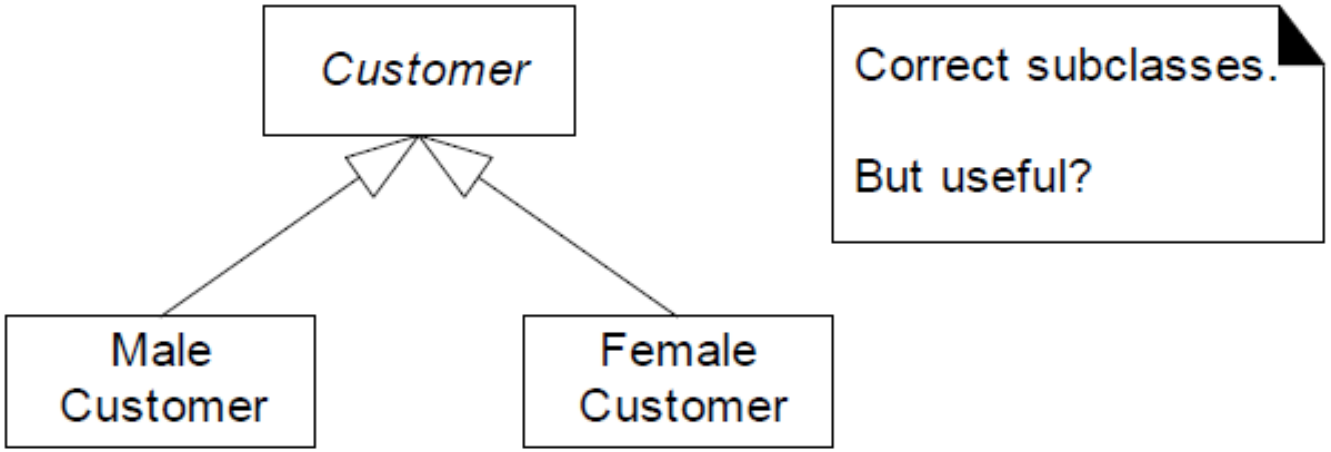


Subclass Conformance

- ***100% Rule***
  - 100% of the conceptual superclass's definition should be applicable to the subclass.
    - **The subclass must conform to 100% of the superclass's: attributes and associations.**
- ***Is-a Test***
  - (Credit Payment is-a Payment)

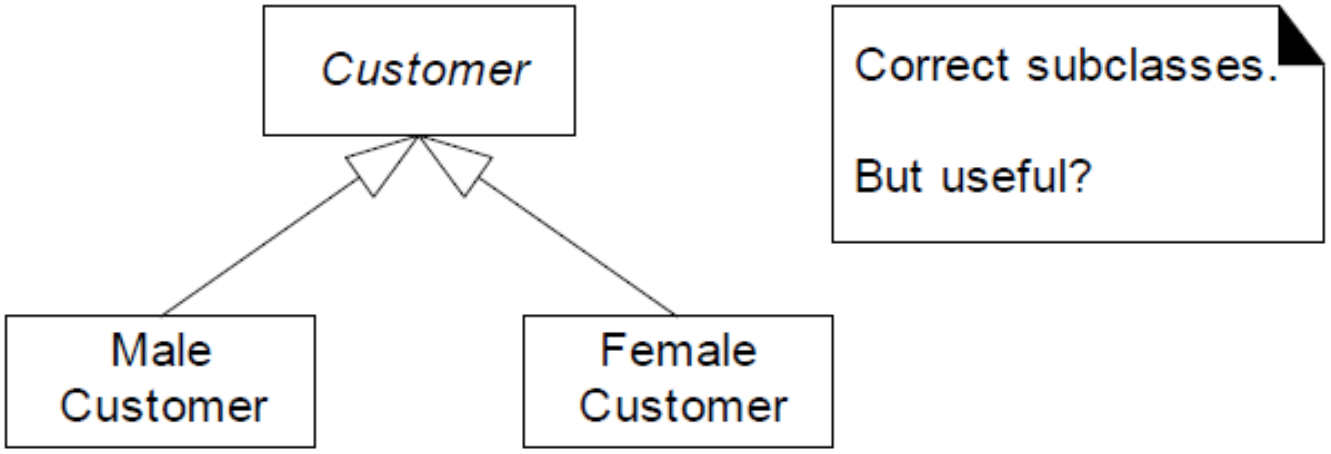


Is this subclass useful?

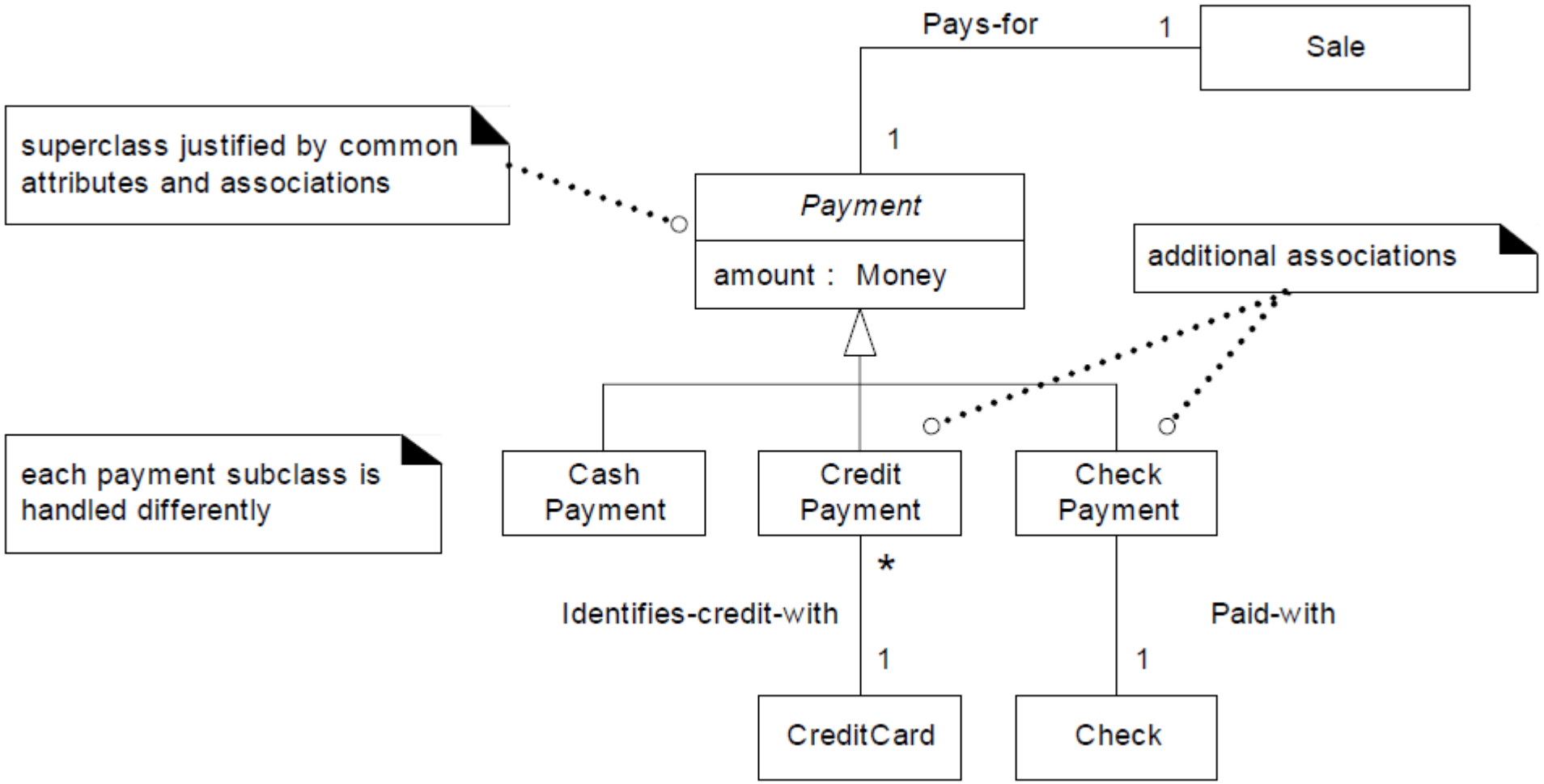


Is this subclass useful?

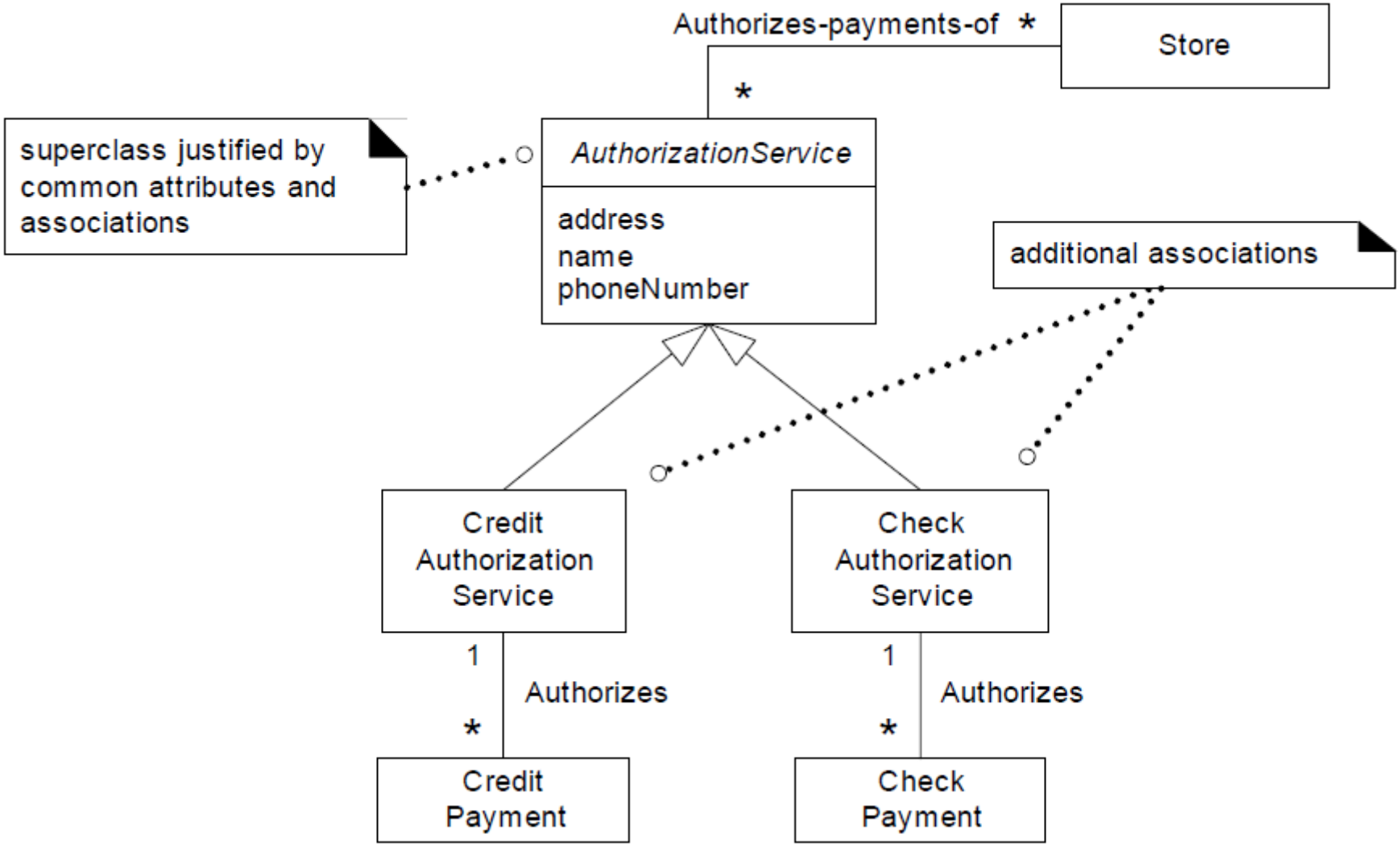
- This partition is not useful for our domain;
- Male and Female Customers are not operated (or treated) differently.



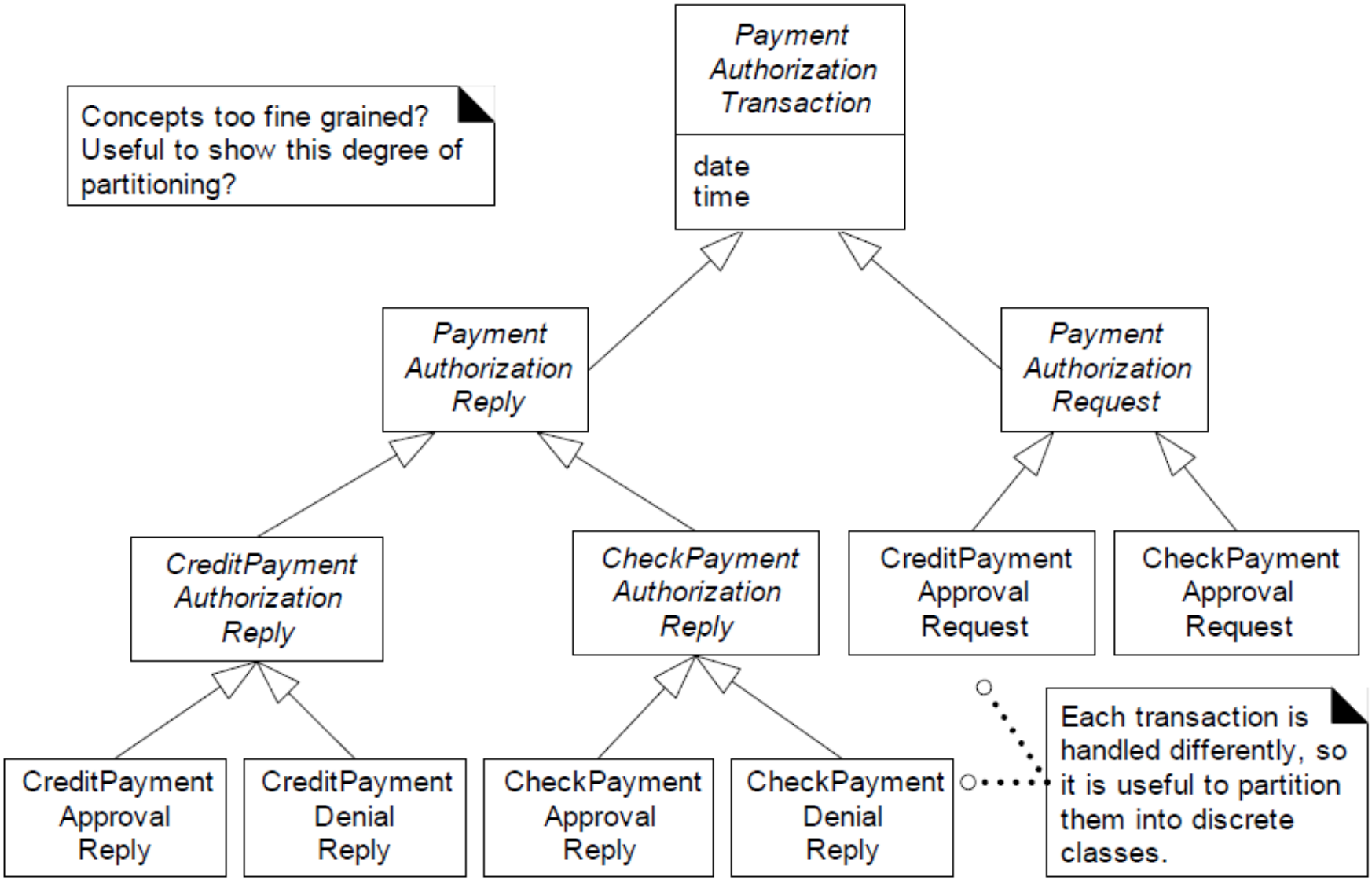
Justifying Payment Subclasses



Justifying Authorization Service Hierarchy

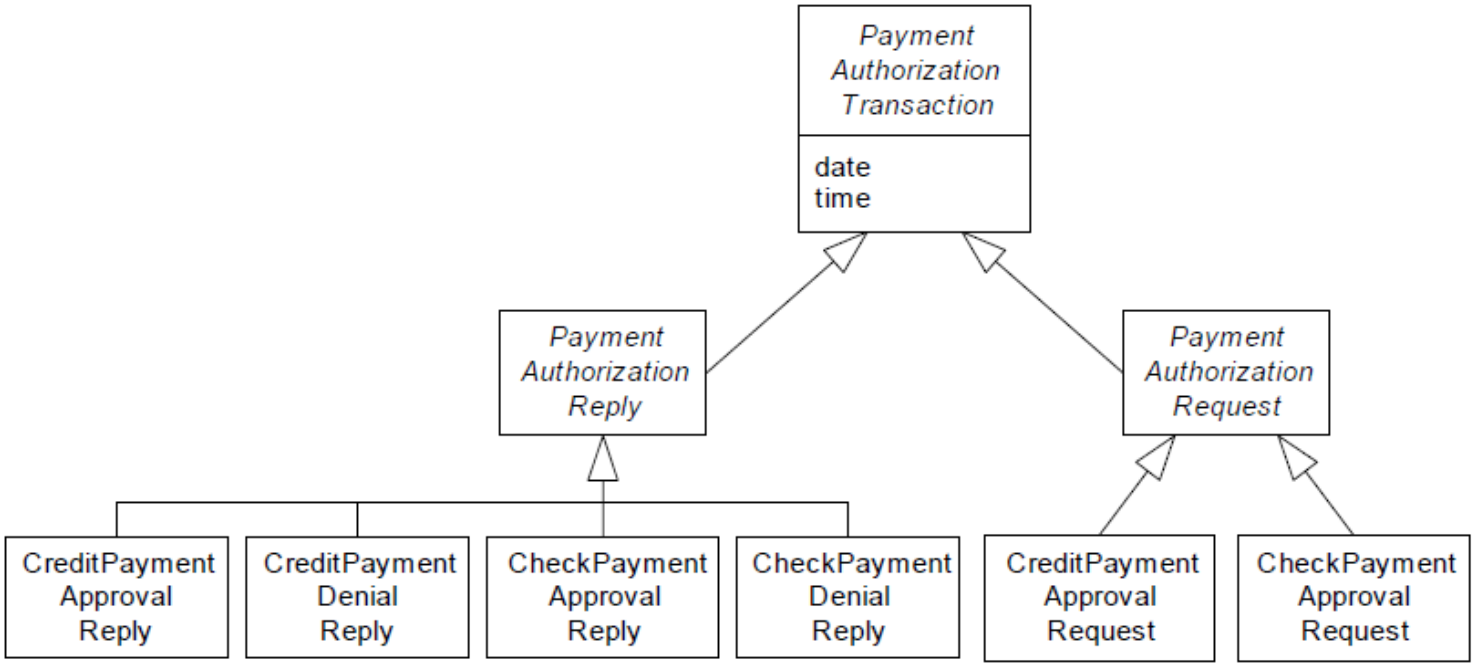


One Possible Transaction Class Hierarchy



Alternate Transaction Class Hierarchy

- The class hierarchy shown is sufficiently useful in terms of generalization,
  - because the additional generalizations in the previous diagram do not add obvious value.

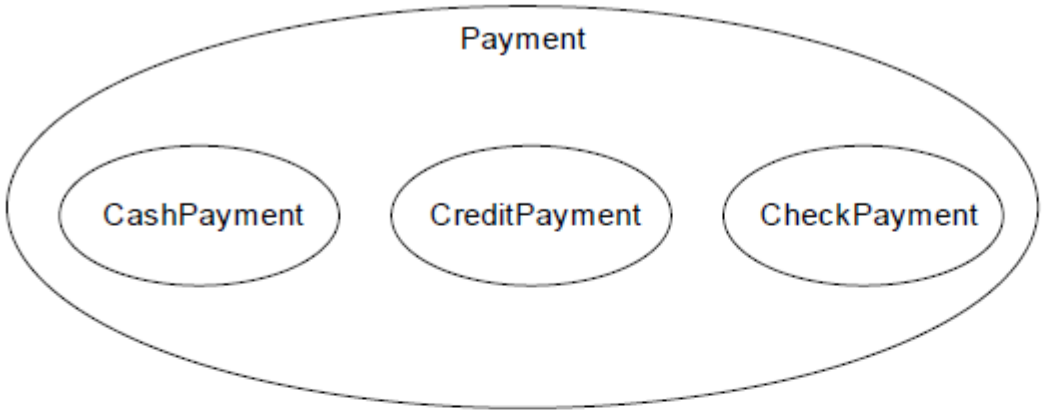


Abstract Conceptual Classes

- If every member of a class C must be a member of a subclass, then class C is called an **abstract conceptual class**.
- For example, assume that every *Payment* instance must more specifically be an instance of the subclass *CreditPayment, CashPayment,* or *CheckPayment.*

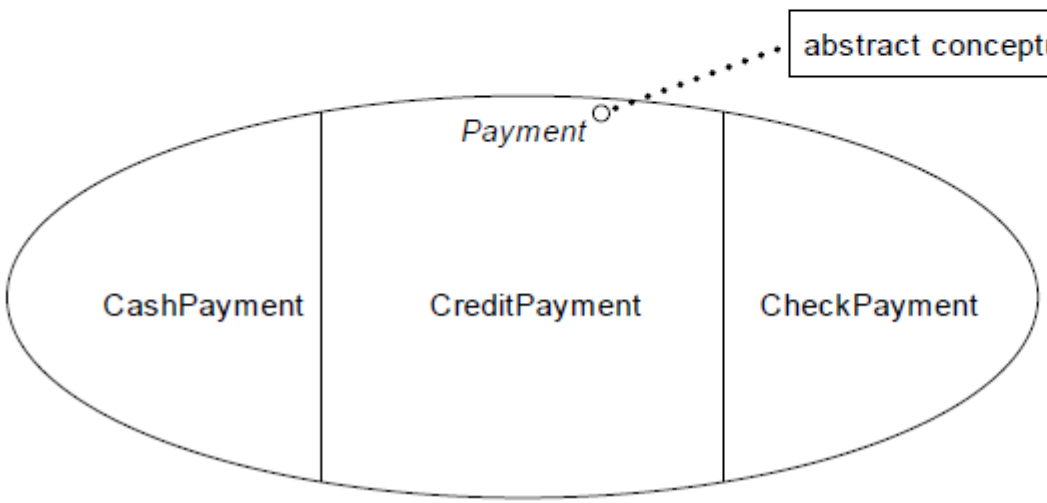
Abstract Conceptual Classes

(a)



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

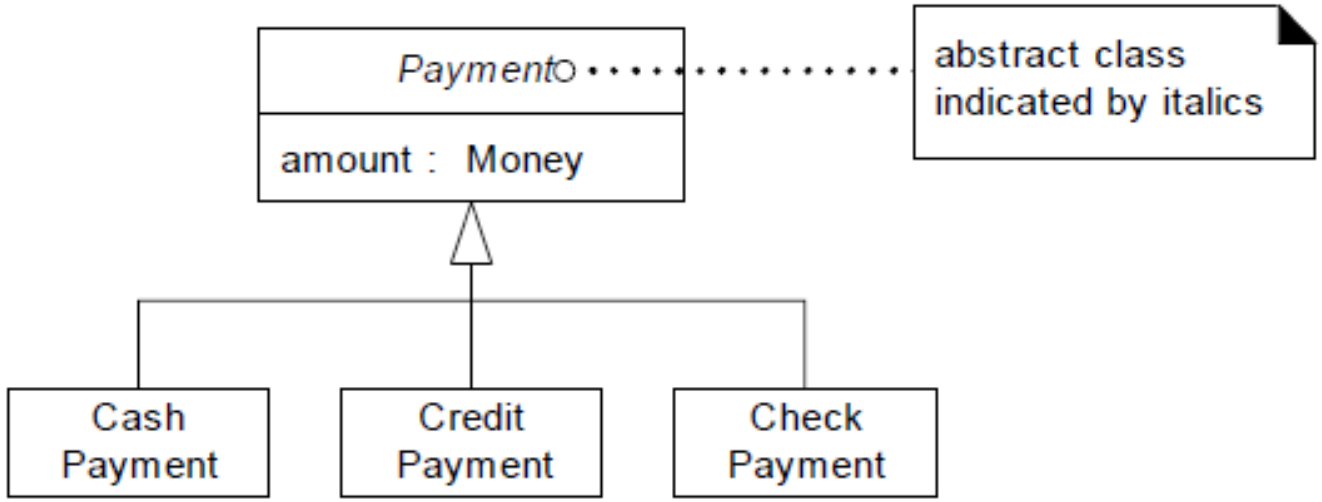
(b)



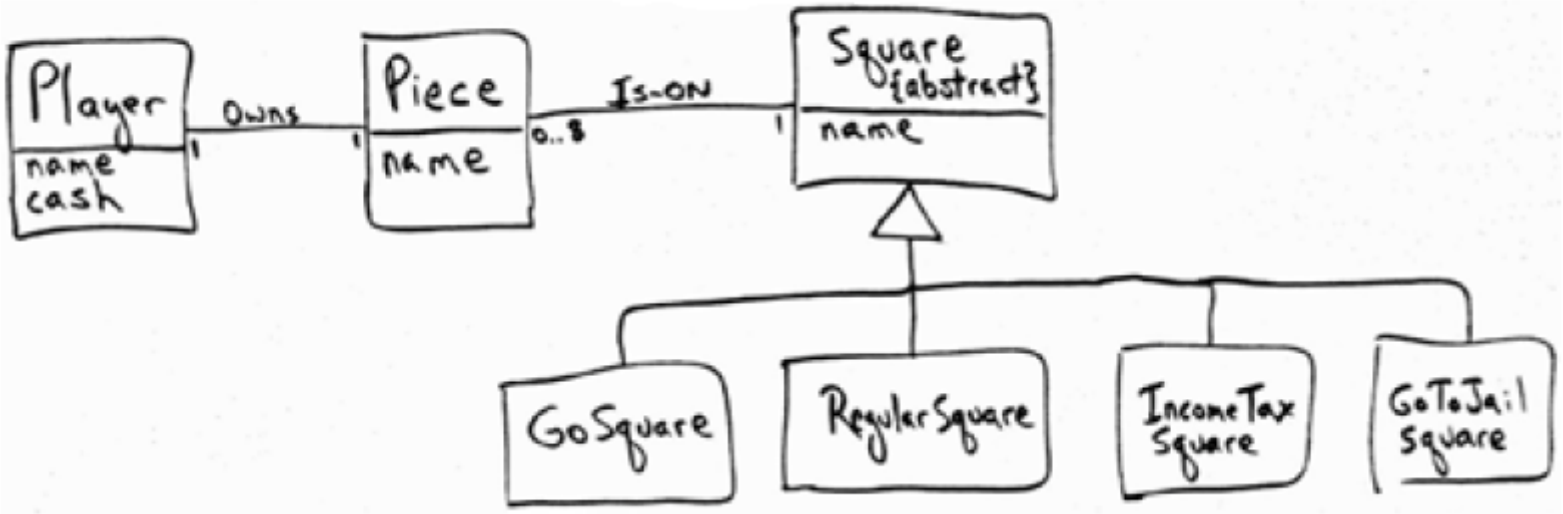
Payment is an **abstract conceptual class** A Payment instance must conform to one of the subclasses: CashPayment, CreditPayment or CheckPayment.

Abstract Class Notation in UML

- Identify abstract classes and illustrate them with an italicized name in the Domain Model, or use the *{abstract}* keyword.

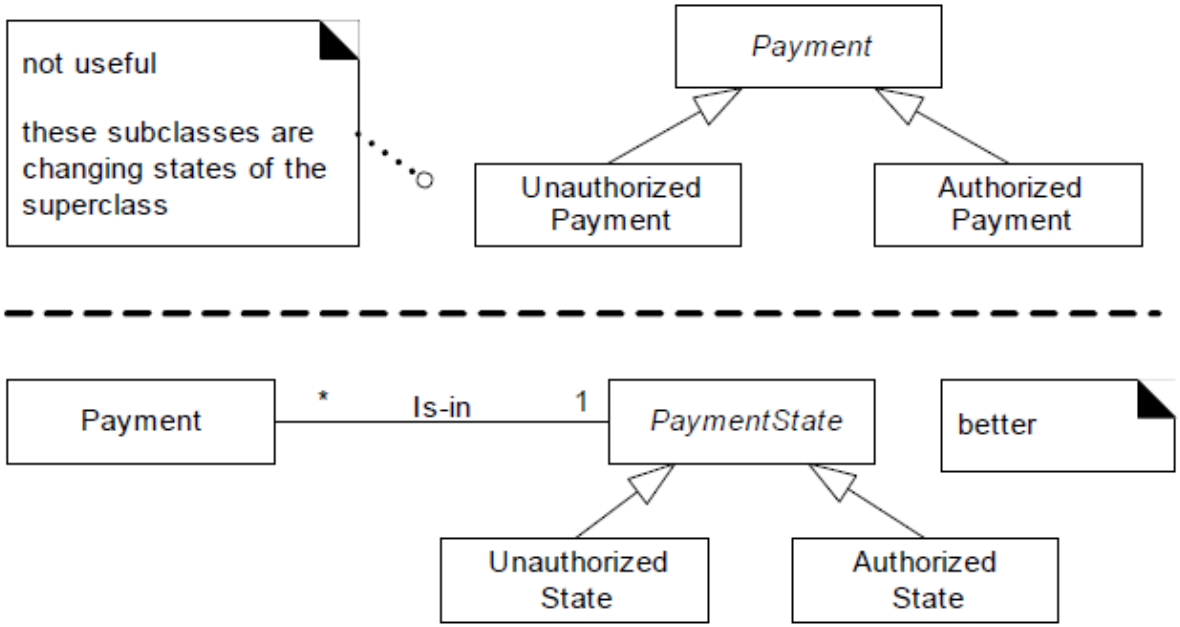


Monopoly Domain Model Changes for Iteration-2



Modeling Changing States

- Note that a payment does not stay in one of these states;
  - it typically transitions from unauthorized to authorized.

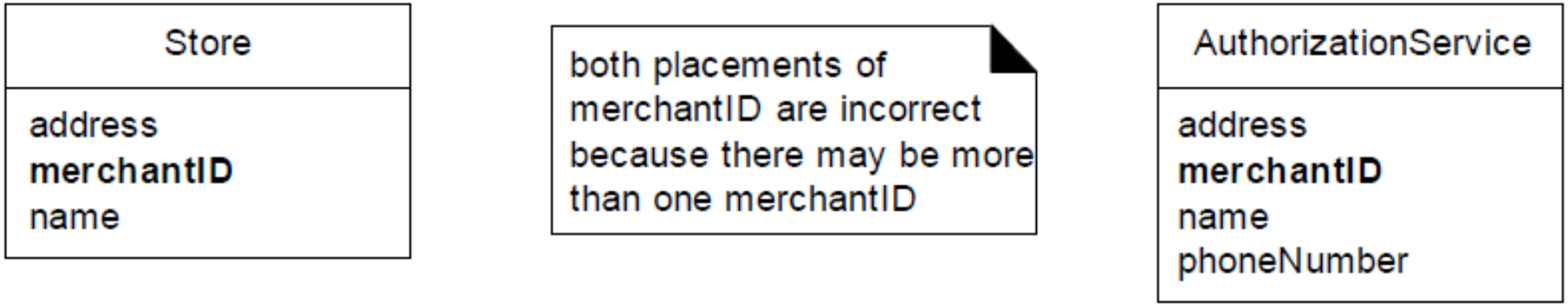


Modeling Changing States

- Do not model the states of a concept X as subclasses of X. Rather, either:
  - Define a state hierarchy and associate the states with X, or
  - Ignore showing the states of a concept in the domain model;
    - show the states in State Diagrams instead.

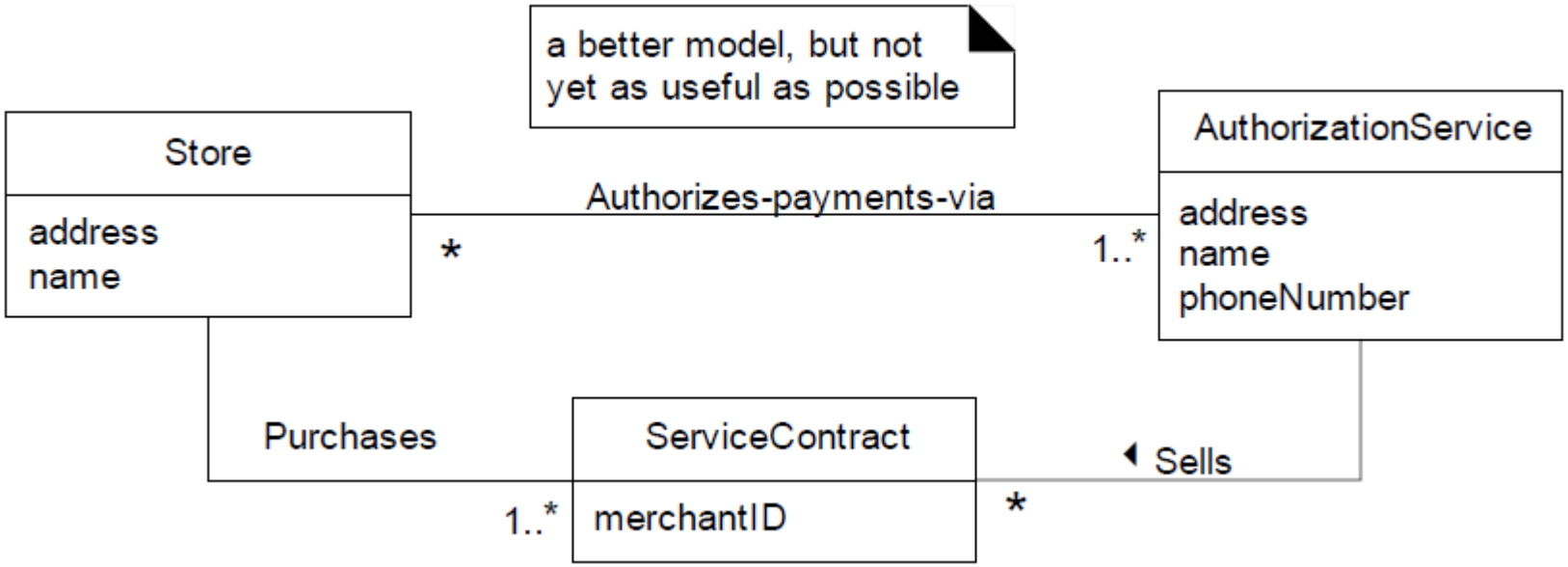
Inappropriate Use of Attribute

- Placing *merchantID* in *Store* is incorrect because a *Store* can have more than one value for *merchantID.*
- The same is true with placing it in *Authorization-Service*



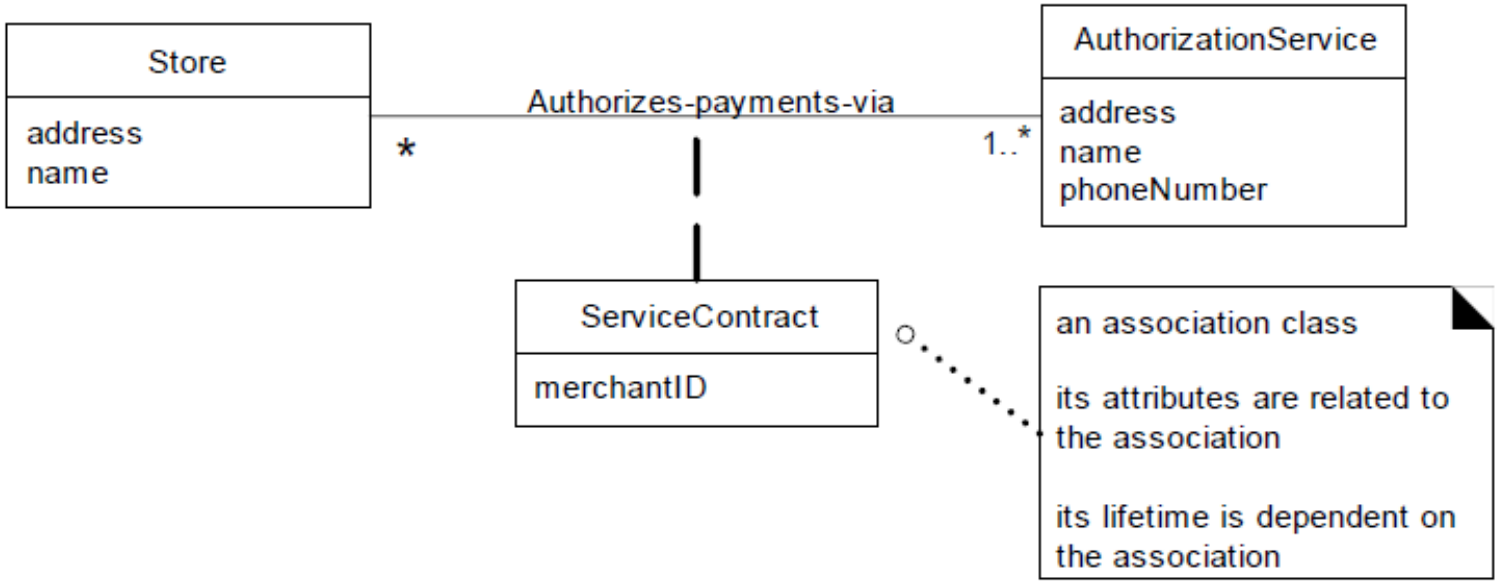
Modeling MerchantID Problem

- First attempt at modeling the MerchantID problem

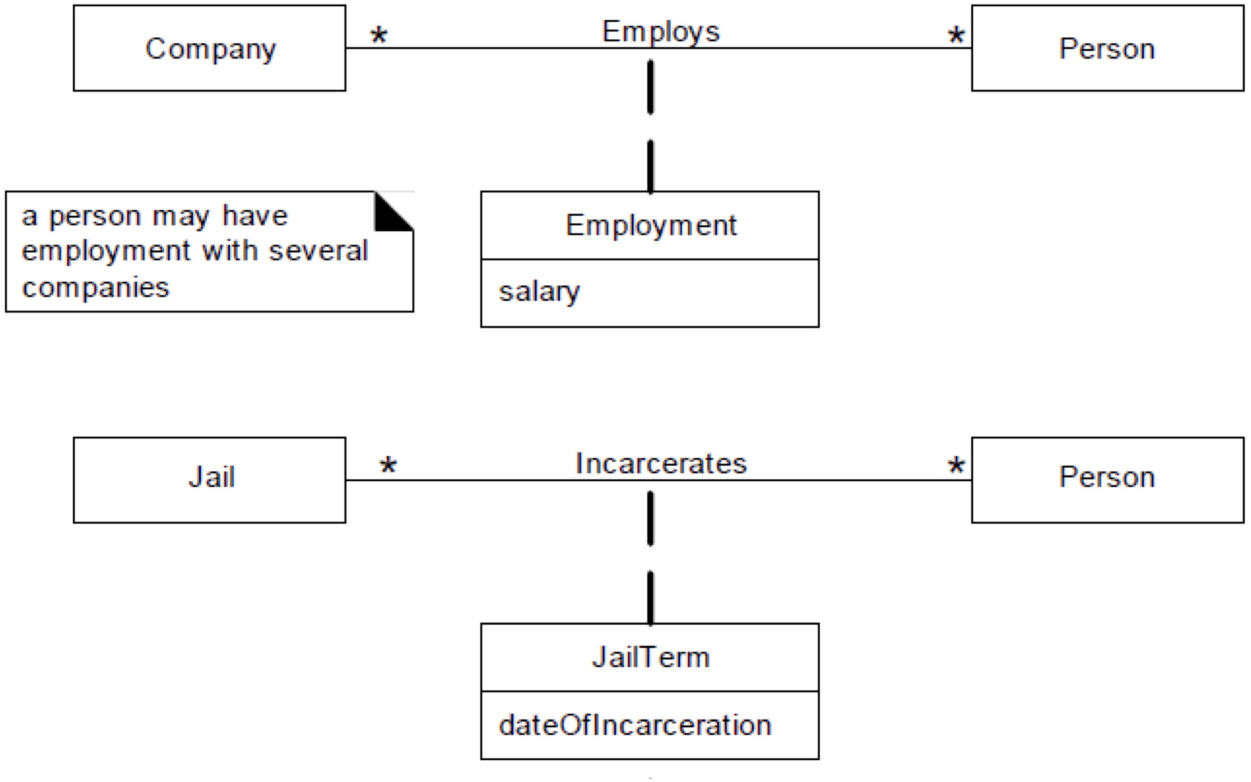


Association Class

- This leads to the notion of an **association class**, in which we can add features (e.g., attributes) to the association itself.
  - Common if there is a many-to-many association between two concepts

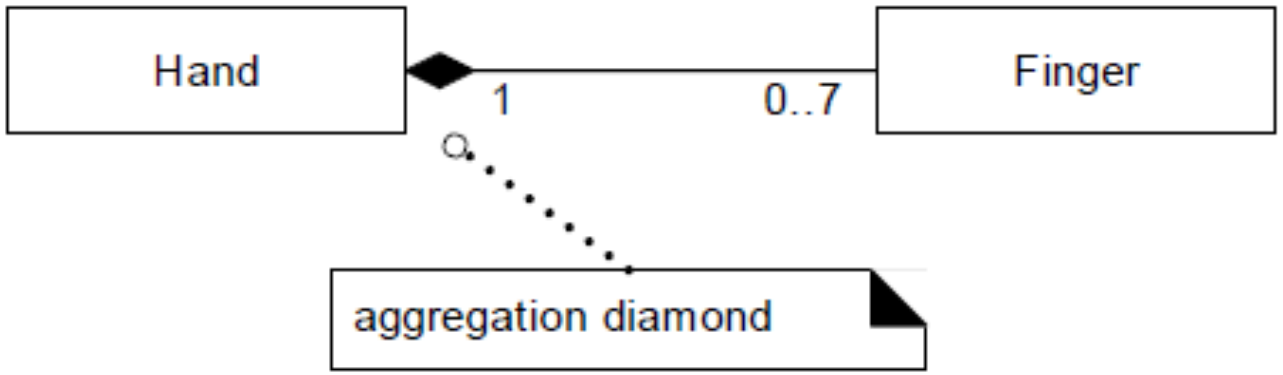


Other Examples of Association Classes



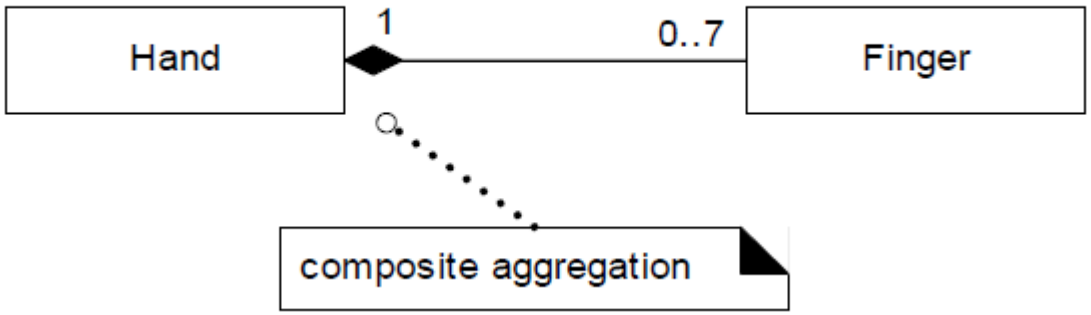
Aggregation in UML

- **Aggregation** is a kind of association used to model whole-part relationships between things.



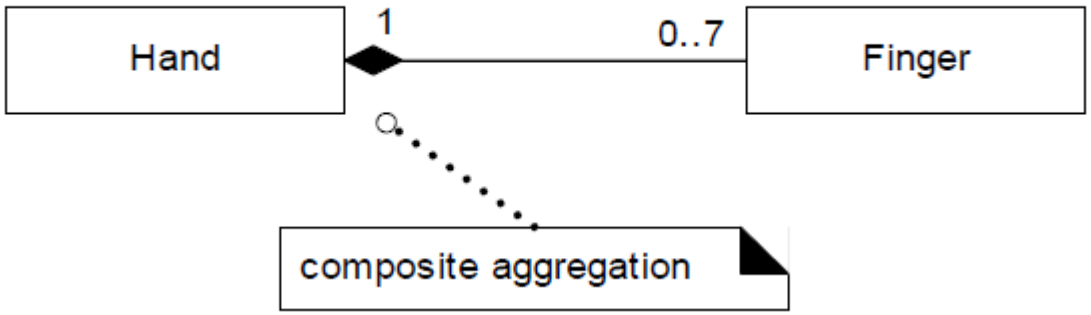
Association Name

- The association name is often excluded in aggregation relationships since it is typically thought of as *Has-part.*



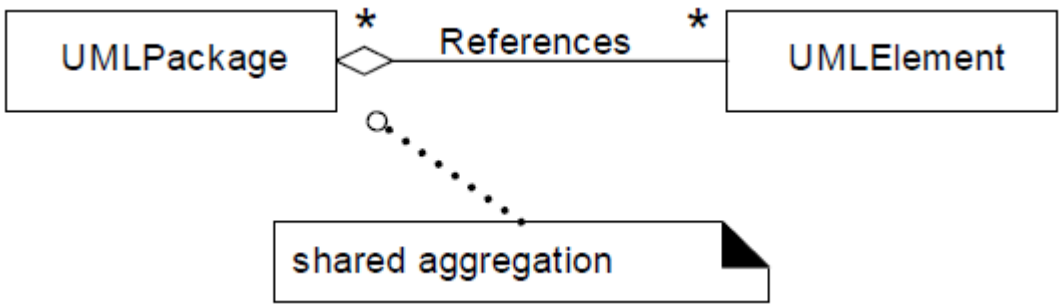
Composite Aggregation

- **Composite aggregation, or composition,** means that the part is a member of only one composite object
  - For example, a hand is in a composition relationship to a finger.

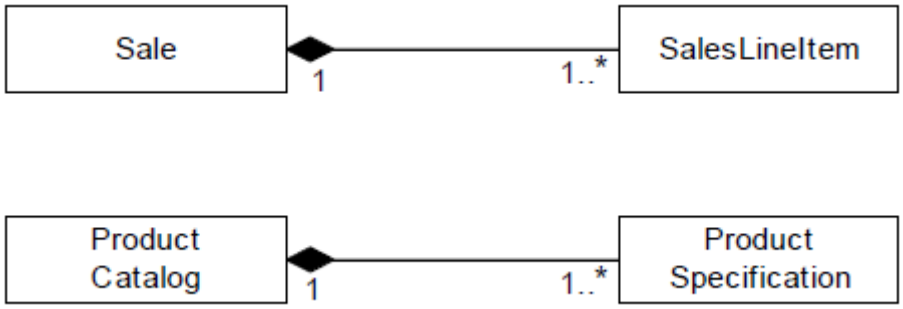


Shared Aggregation

- **Shared aggregation** means that the multiplicity at the composite end may be more than one, and is signified with a hollow diamond.
- It implies that the part may be simultaneously in many composite instances.

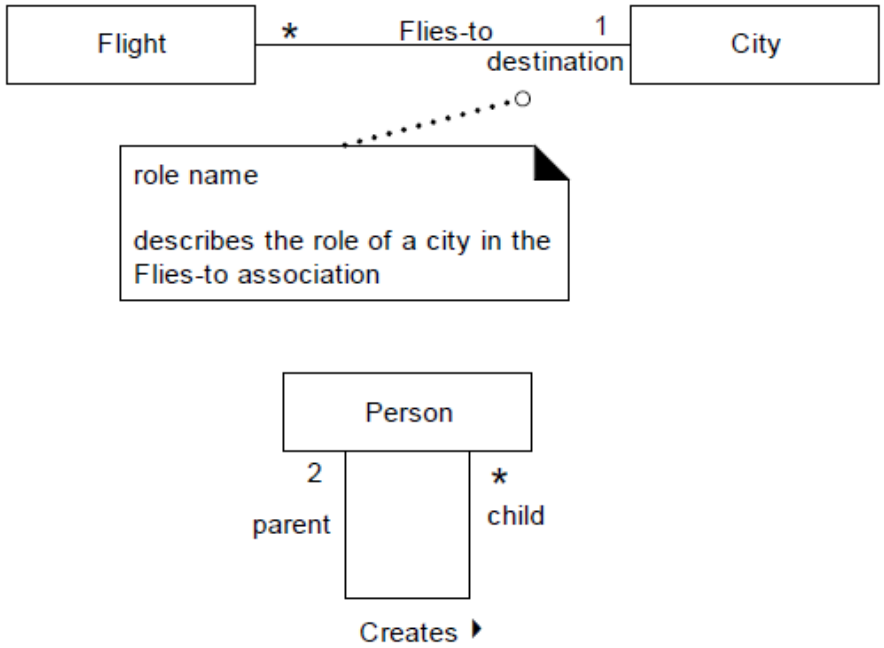


Aggregation in POS Domain Model

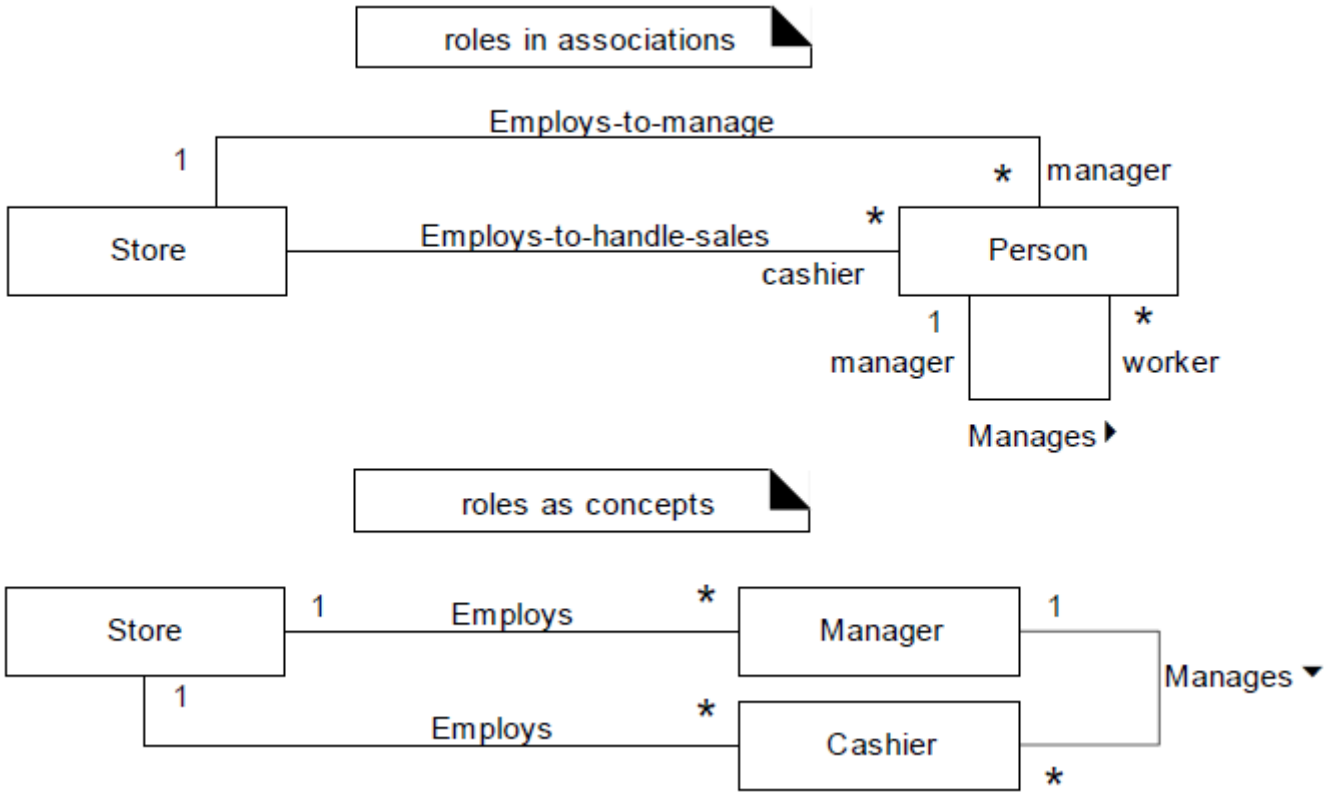


Role Names

- In a domain model, a real-world role—especially a human role—may be modeled in a number of ways,
  - expressed as a **role** in an association.

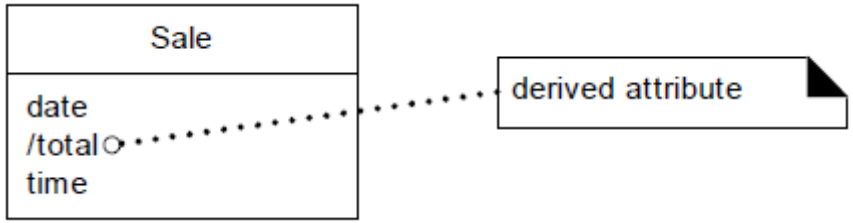


Two Ways to Model Human Roles

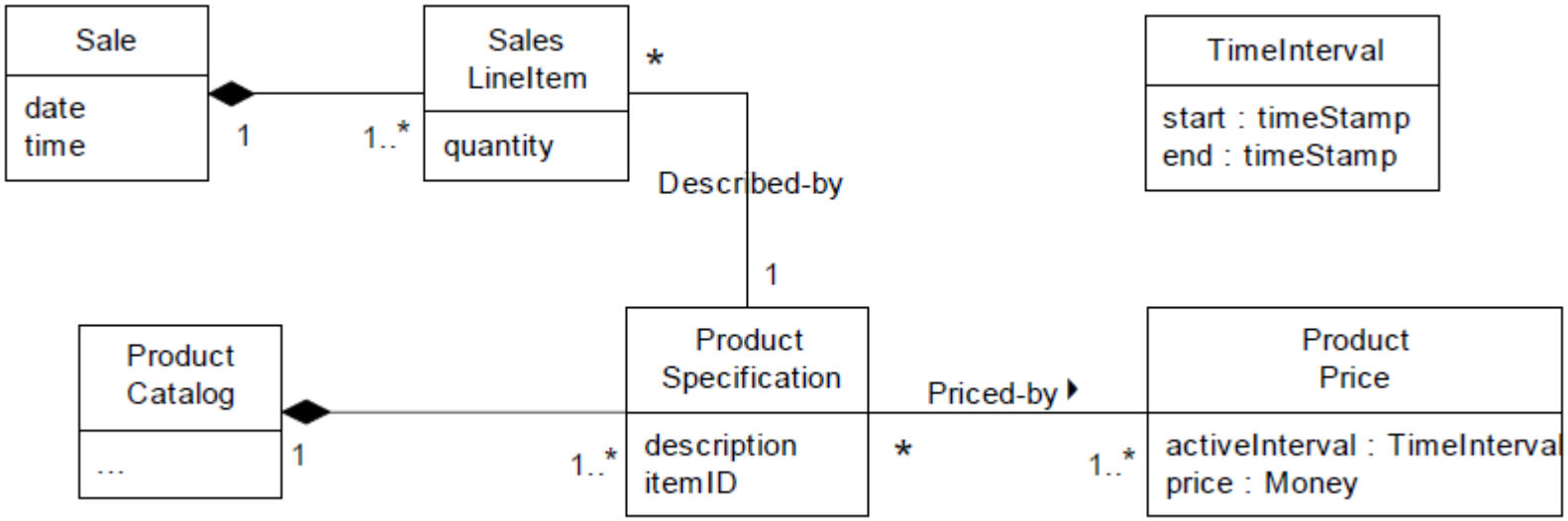


Derived Attributes

- A **derived element** can be determined from others.

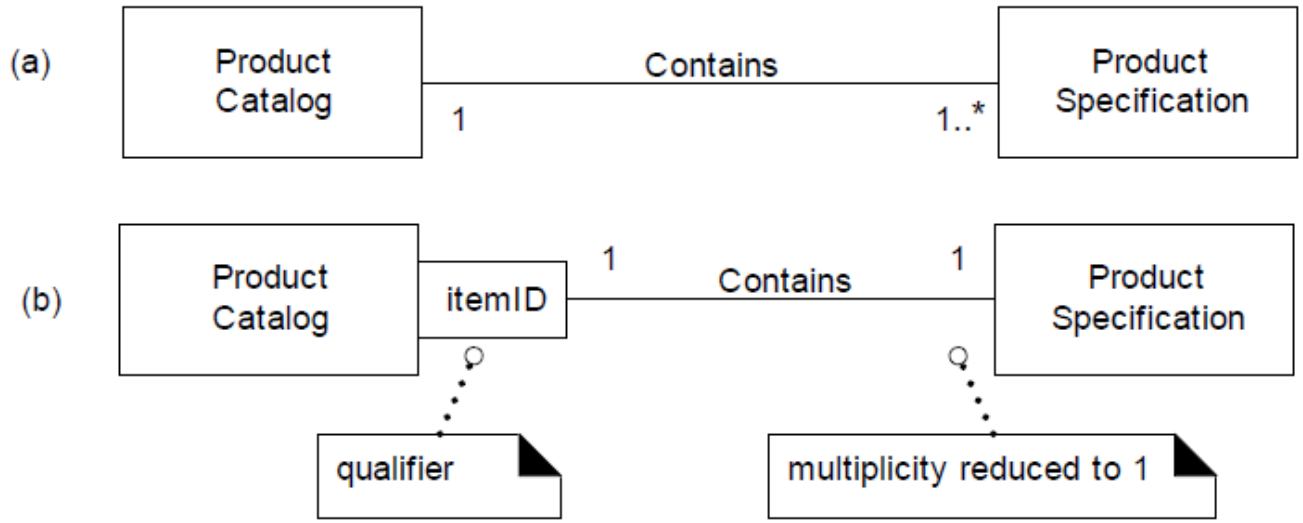


Product Prices and Time Intervals



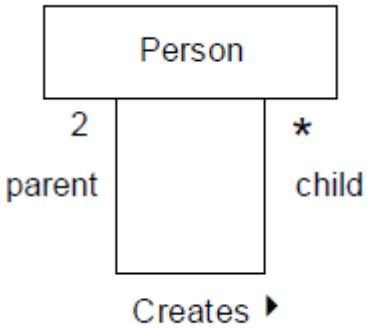
Qualified Association

- A qualifier may be used in an association;
  - it distinguishes the set of objects at the far end of the association based on the qualifier value.
  - An association with a qualifier is a **qualified association.**
  - For example, *ProductSpecifications* may be distinguished in a *ProductCatalog* by their *itemID.*

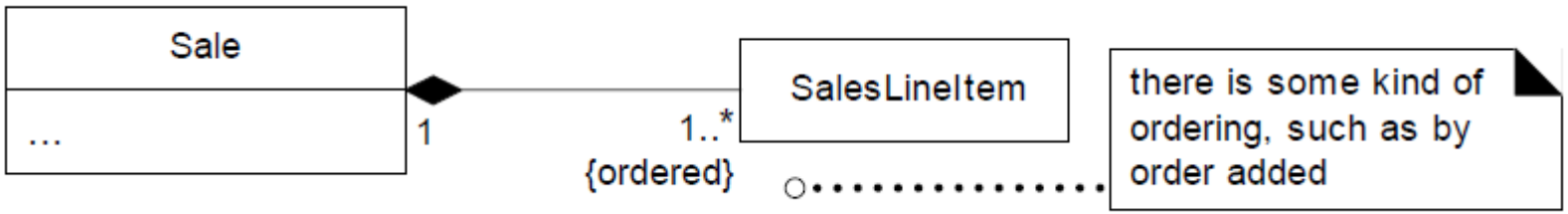


Reflexive Association

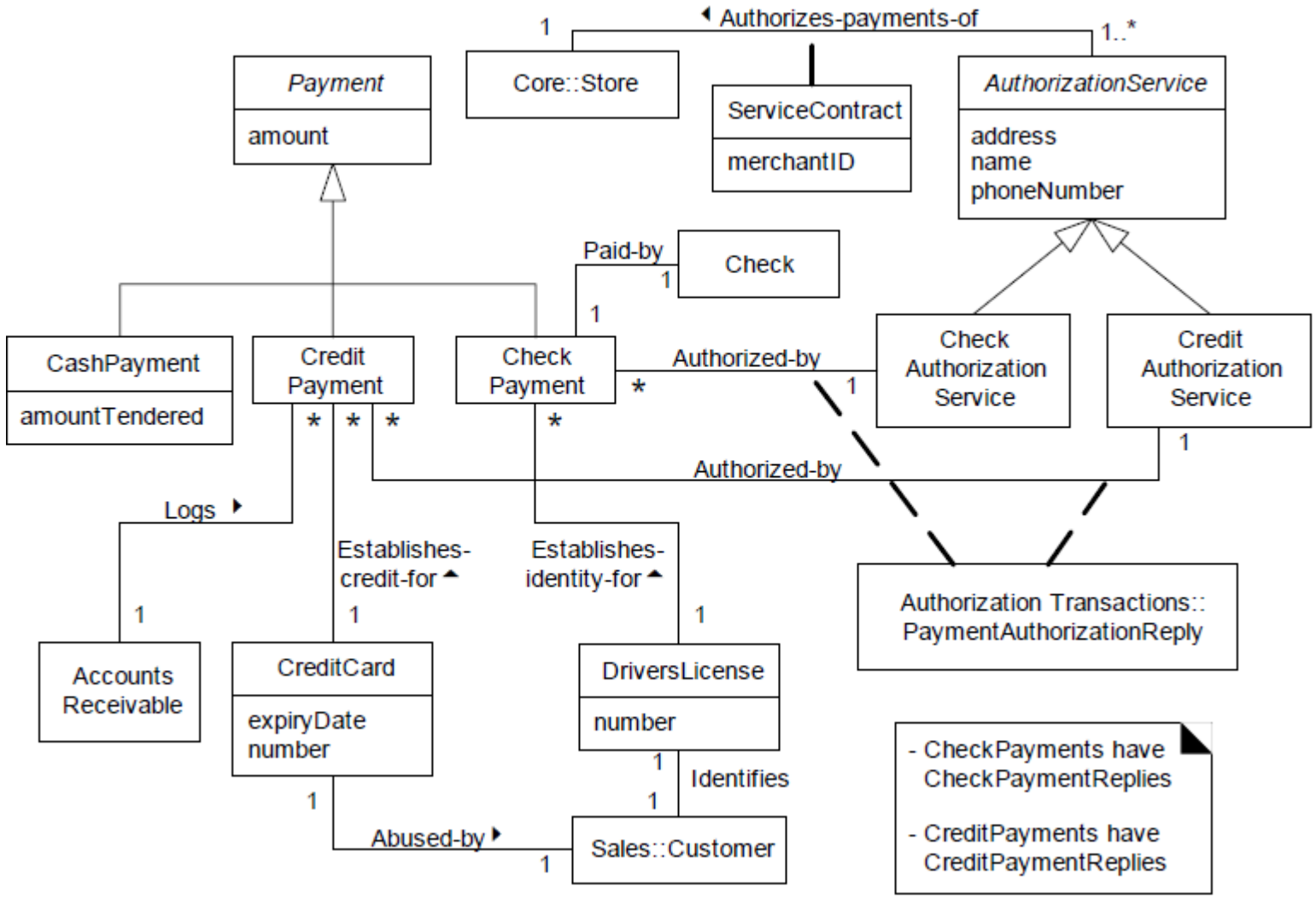
- A concept may have an association to itself;
  - this is known as a **reflexive association**.



Ordered Elements



Payments UML Class Diagram



Quiz

- Provide an example of composite aggregation and shared aggregation.
- What are the advantages of generalization from programming point of view?
- What is reflexive association? Provide an example of it.
- What is 100% rule and Is-a Test?
- What are two ways to model roles? Provide an example.

Actions

- Review Slides.
- Read Chapter 26 (Modeling Generalization)
  - *Applying UML and Patterns*, Craig Larman