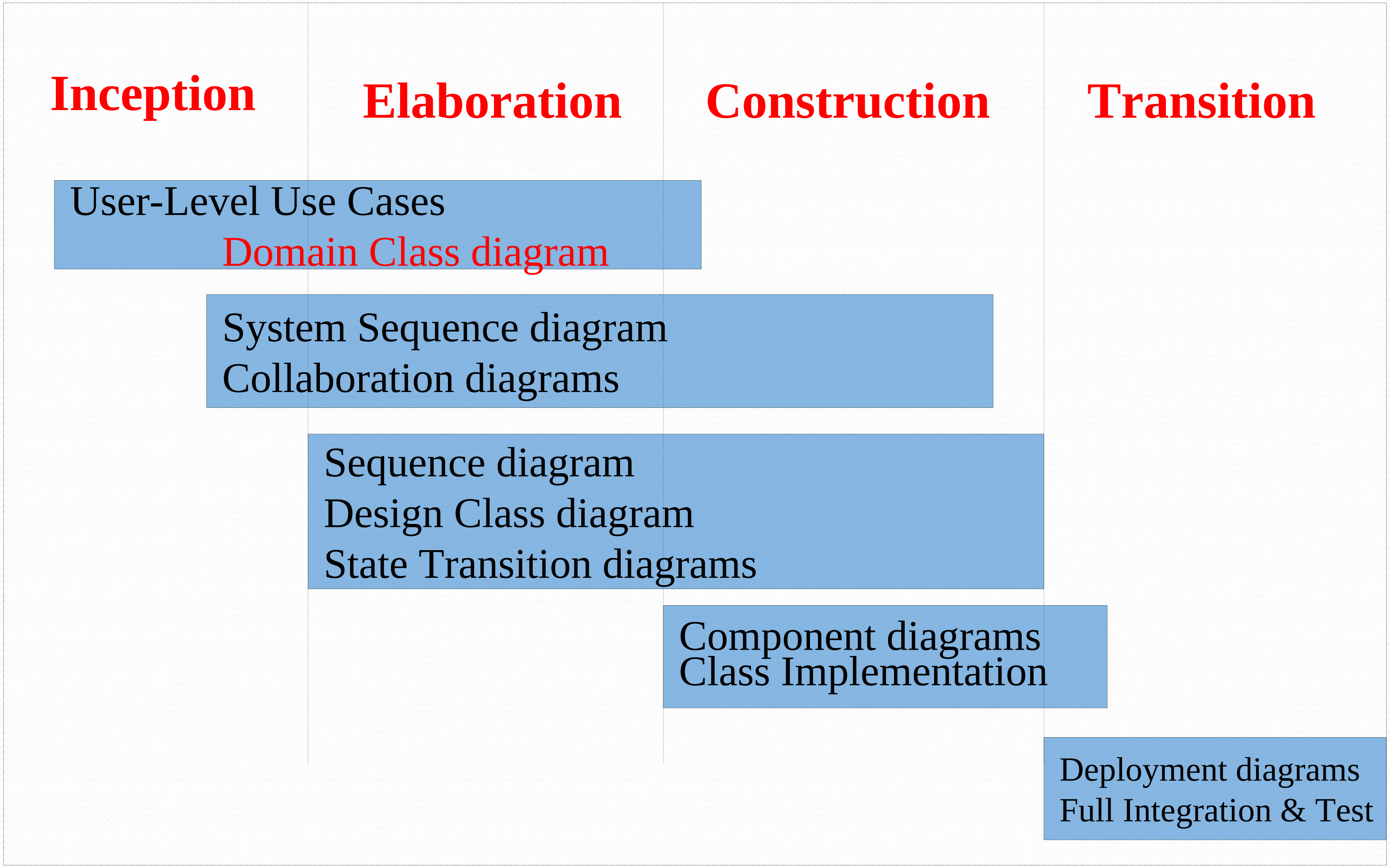
COMP 3711

(OOA and OOD)

Domain Model
Conceptual Class Relationships

Larman Chapter 9, 31

UML And UP



Inception

Elaboration

Construction

Transition

User-Level Use Cases

Domain Class diagram

System Sequence diagram

Collaboration diagrams

Sequence diagram

Design Class diagram

State Transition diagrams

Component diagrams

Class Implementation

Deployment diagrams

Full Integration & Test

Domain Model - Conceptual Class Relationships

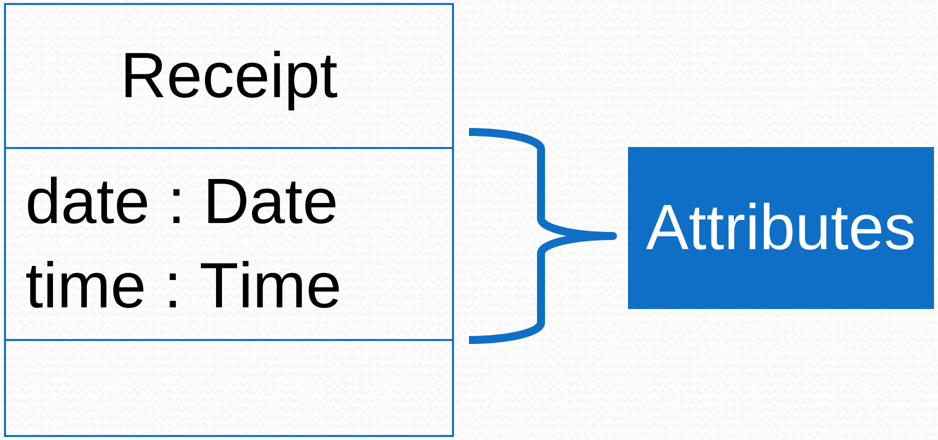
- In the Design Model, a software subclass inherits the attribute and operation definitions of its superclass through the inheritance hierarchies.

- The focus in the Domain model is the relationships between the conceptual classes, which may or may not be reflected in the Design Model.

Domain Model - Attributes

- An attribute is a data value which is part of an object
- Suggested or implied by requirements
- Collectively store the state of the object
- Attributes in Domain model preferably be simple attributes or data types (Boolean, Date, Number, Character, Sting, Time, Address, Colour, etc.)
- Attributes should not be used to relate conceptual classes in the Design Model

Domain Model - Attributes



Receipt

date : Date

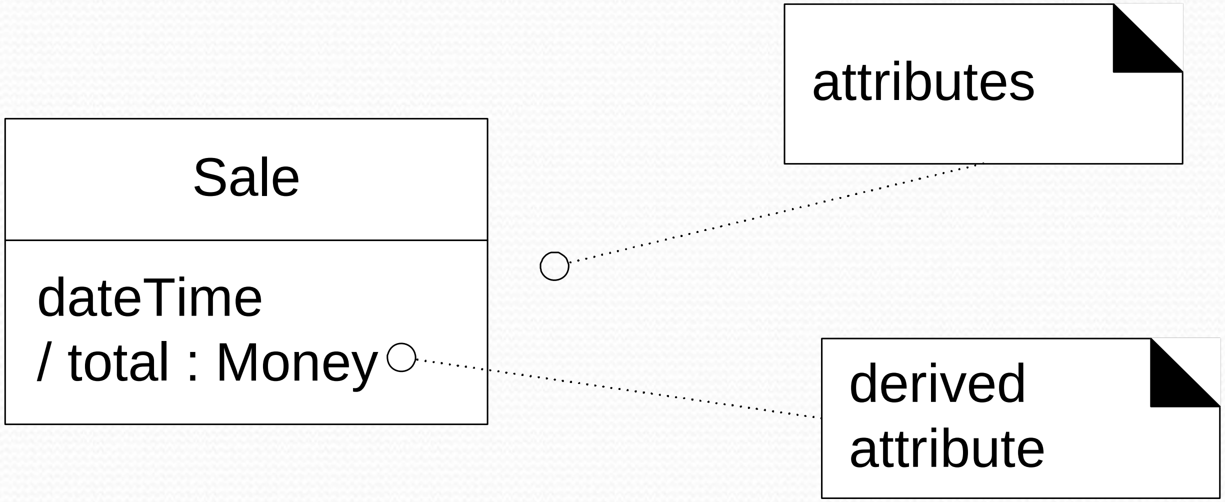
time : Time

Attributes

- Class name starts with a Capital letter and attribute name start is lowercase
- Attributes are shown in the second compartment of the class box
- Attribute type may be optionally shown
- In Domain Modeling the type is not normally shown

Derived Attributes

- A Derived Attribute is calculated or derived from information in another attribute
- Derived Attribute is prefixed by a /



Larman Fig 9.19

Domain Model Is Not

- An association in a Domain Model is not a statement about:
  - data flows
  - database foreign key relationships
  - instance variables
  - object connections in a software solution
- But about a relationship being meaningful in a purely conceptual perspective in the real domain.
- EG: Don’t stop to wonder how one class will distinguish another (key relations)

Identifying Associations

- More difficult than finding classes
- A relationship that needs to be preserved for some duration (need-to-know associations)
- Ask the question:
    - *Between what objects do we need some memory of a relationship?*
- Look at verbs and verb phrases in problem statement

Identifying Associations (continued ..)

- Any message between classes on a sequence or collaboration diagram requires a relationship between the classes
- Don’t worry about implementation details
- Consider deriving associations from the “Common Associations List”.

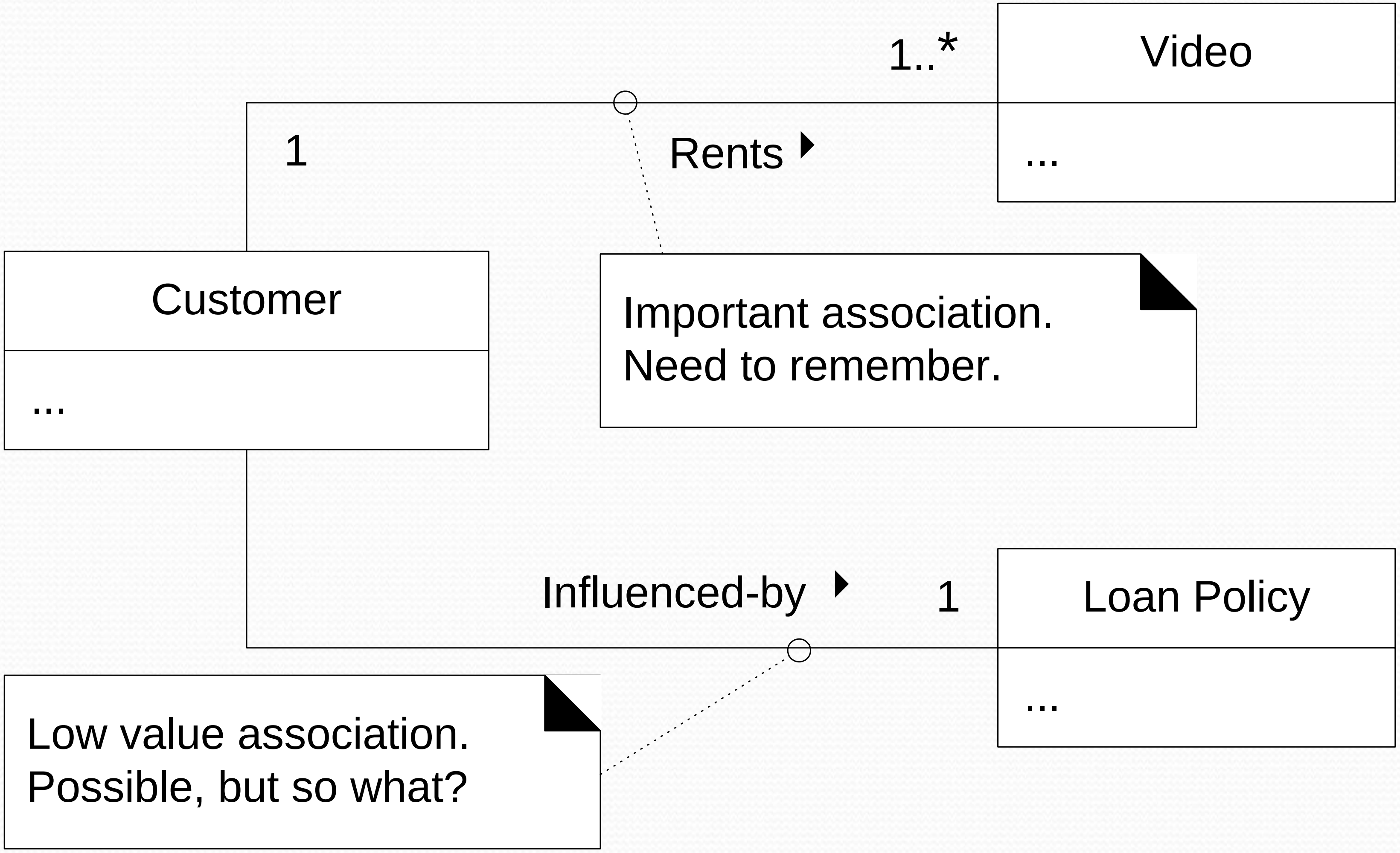
Common Association List - Example

Category	Examples
A is a physical part of B	Drawer←→Register; Wing←→Airplane
A is a logical part of B	SalesLineItem←→Sale; FlightLeg←→FlightRoute
A is physically contained in B	Register←→Store; Passenger← →Airplane
A is logically contained in B	ItemDescription←→Catalog; Flight←→FlightSchedule
A is a description for B	ItemDescription←→Item; FlightDescription←→Flight
A is a line item of a transaction or report in B	SalesLineItem←→Sale; MaintenanceJob←→MaintenanceLog
A is known/logged/recorded/reported/captured in B	Sale←→Register; Reservation←→FlightManifest
A is a member of B	Cashier←→Store; Pilot←→Airline
A is an organizational sub-unit of B	Department←→Store;Maintenance ←→Airline
A uses or manages B	Cashier←→Register; Pilot←→Airplane
A communicates with B	Customer←→Cashier; ReservationAgent←→Passenger
A is related to a transaction B	Customer←→Payment; Passenger←→Ticket
A is a transaction related to another transaction B	Payment←→Sale; Reservation←→Cancellation
A is next to B	SalesLineItem←→SalesLineItem; City←→City
A is owned by B	Register←→Store; Plane←→Airline
A is an event related to B	Sale←→Customer; Departure←→Flight

Association Guidelines

- Too many lines on a Domain Diagram will clutter it (visual noise)
- Diagram with n different conceptual classes can possibly have n(n-1)/2 associations
- Do not include associations that are not useful in the context of the requirements
- Focus on need-to-know associations
- It is more important to identify conceptual classes than to identify associations

Association Guidelines - Example



Important association.
Need to remember.

Low value association.
Possible, but so what?

Conceptual Class Relationships

- Four types of relationships:
  - Association
  - Aggregation (Composition)
  - Dependency
  - Generalization (Specialization)

Relationships

- Types of relationships:

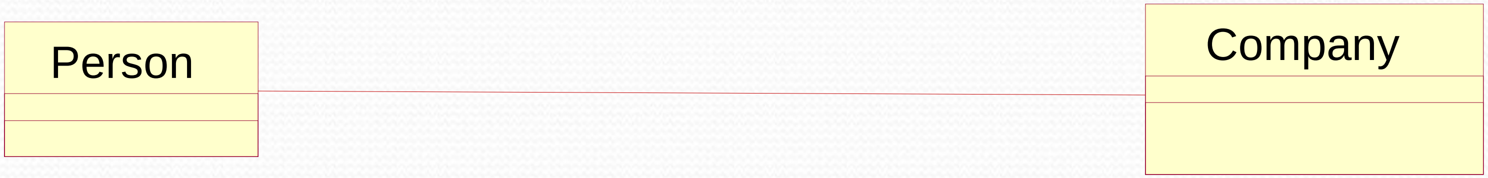
  - Association
  - Dependency
  - Aggregation (Composition)
  - Generalization (Specialization)

Relationship Association

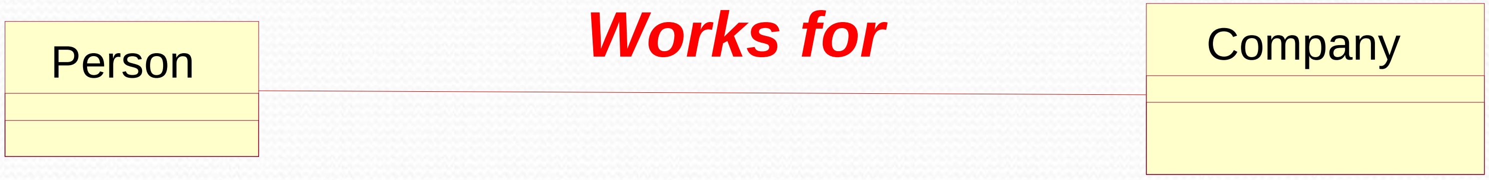
- A bi-directional connection between classes

- An association is shown as a line connecting the related classes

- It means there is a relationship between classes



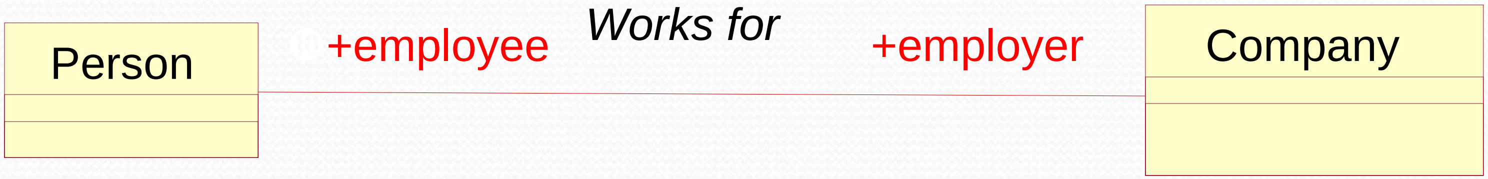
Naming Association



- Describes nature of relationship
- Association may or may not have name, and generally don’t have names.
- Association names should start with a *capital* letter.
- Name is read from left to right, top to bottom
- Name an association based on *TypeName-VerbPhrase-TypeName* format.
  - Example: *Person-Works for-Company*

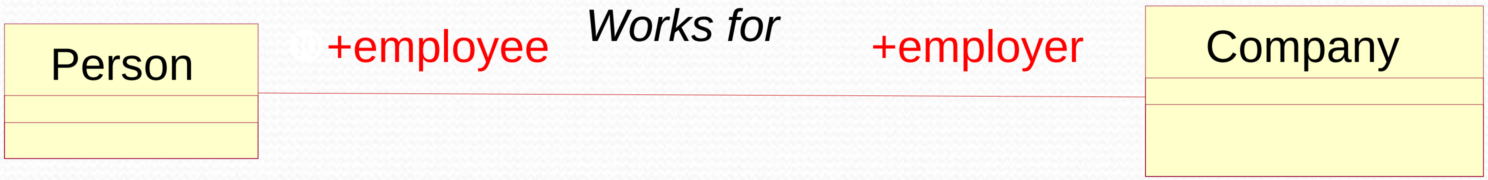
Roles

Each end of an association is called a Role.



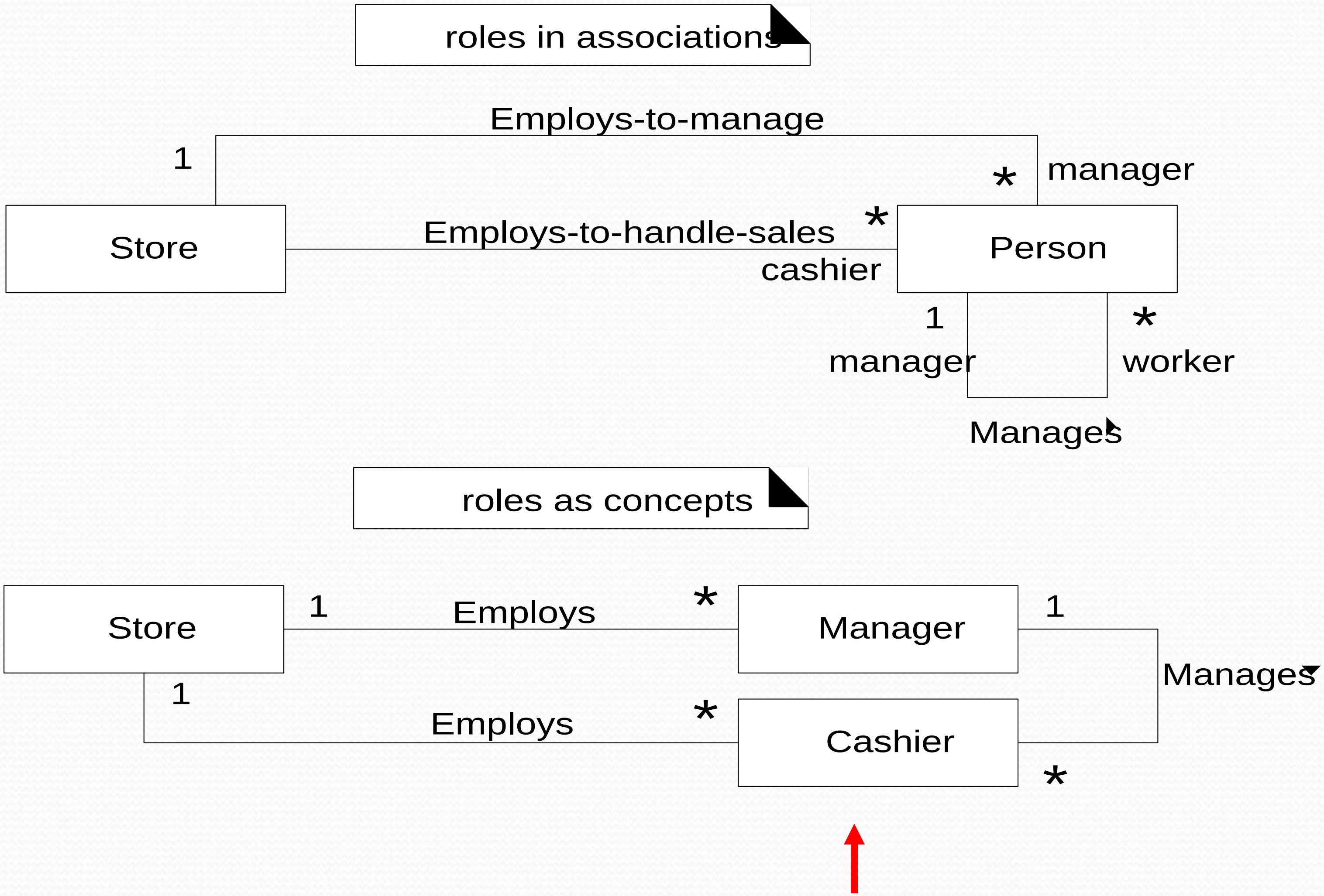
- Roles may optionally have:
  - Name
  - Multiplicity
  - Navigability
  - Type

Role Name



- Role name identifies an end of an association
- Describes the role played by objects in the association
- It is optional to indicate a role name
- Use it when the role of the object is not clear
- Plus sign on role indicates that they are public

Role Name – Another Examples



roles in associations

Employs-to-manage

1

*

manager

Store

Employs-to-handle-sales

*

cashier

Person

1

manager

*

worker

Manages

roles as concepts

Store

1

Employs

*

Manager

1

Manages

1

Employs

*

Cashier

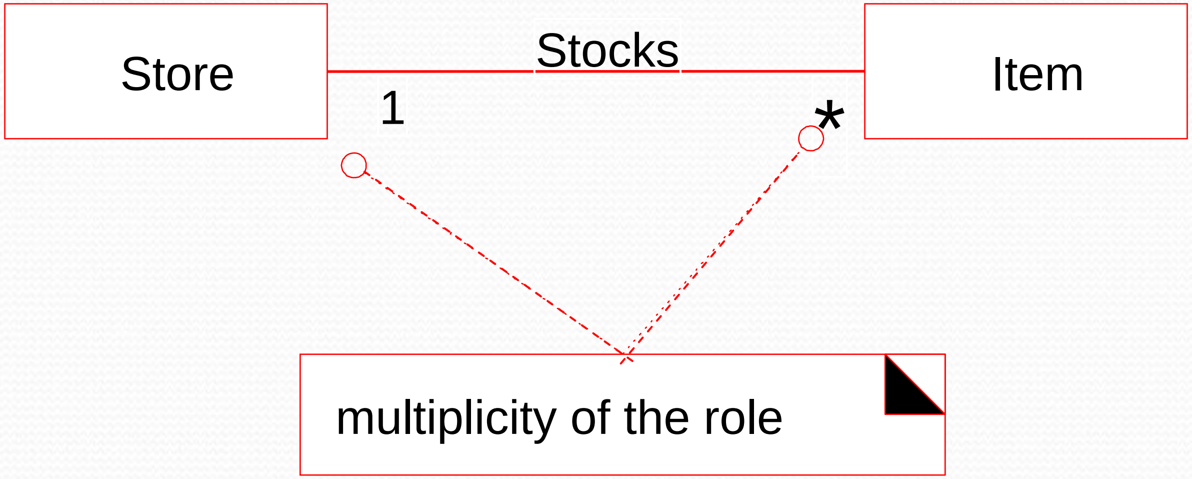
*

Larman Fig 31.21

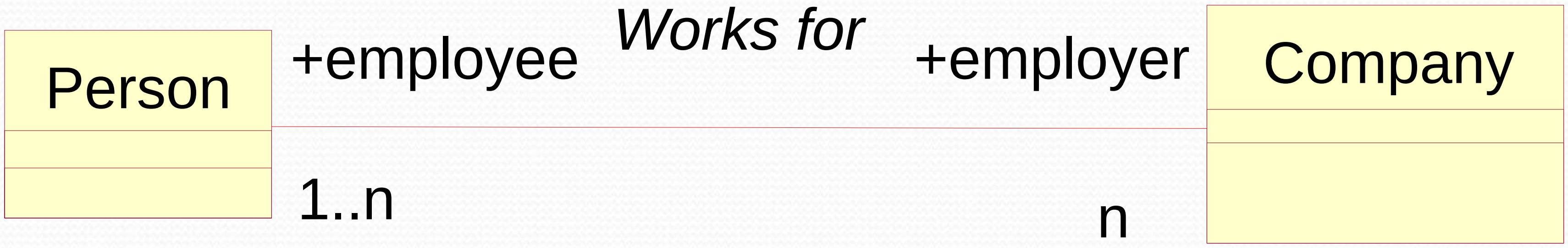
Flexibility as separate classes

Multiplicity

- Multiplicity defines how many objects participate in a relationship
  - Multiplicity is the number of instances of one class related to ONE instance of the other class



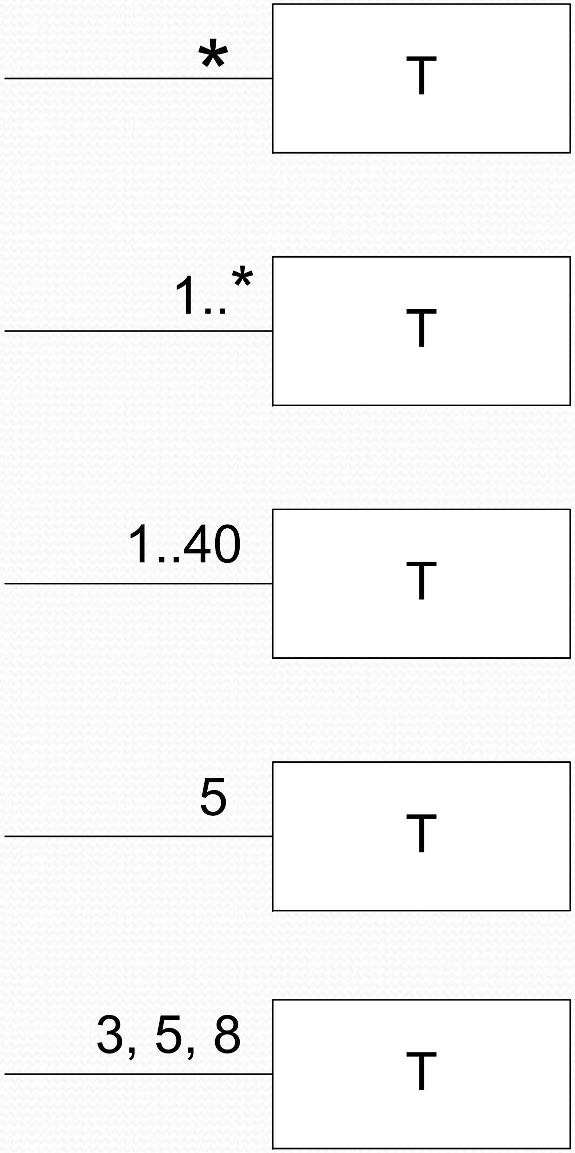
Example of Multiplicities



- Be aware that the UML uses * for many but the Rational Rose implementation uses n

- You can use n or put the * in yourself

Example of Multiplicities



* T

zero or more;
"many"

1..* T

one or more

1..40 T

one to 40

5 T

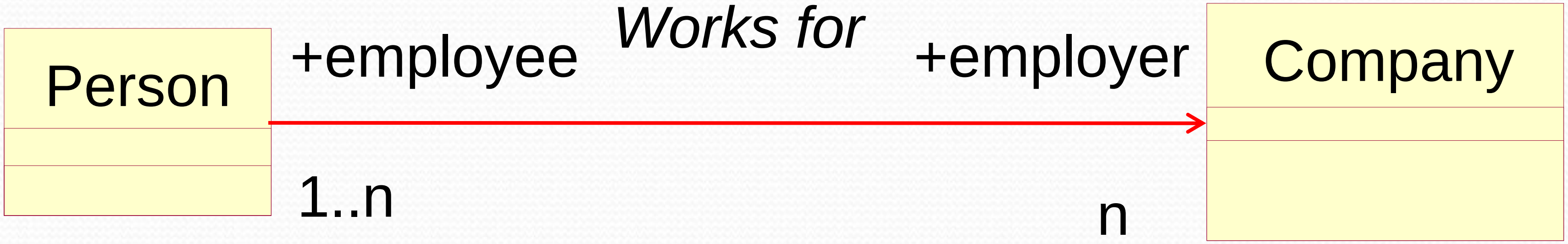
exactly 5

3, 5, 8 T

exactly 3, 5, or 8

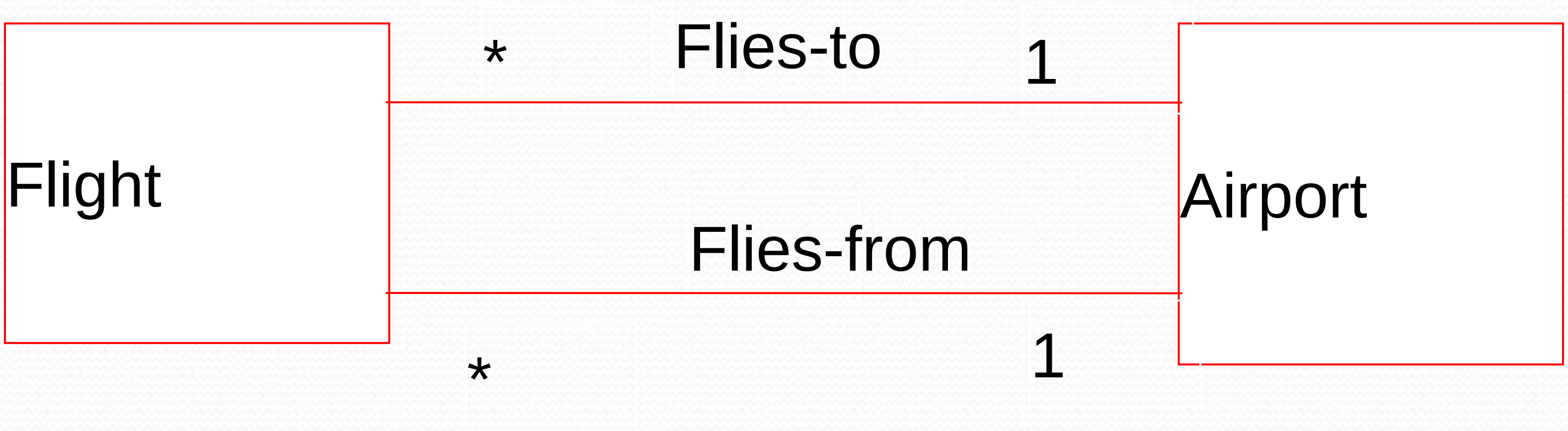
Navigability

- Although associations and aggregations are bi-directional by default, it is often desirable to restrict navigation to one direction
- If navigation is restricted, an arrowhead is added to indicate the direction of the navigation

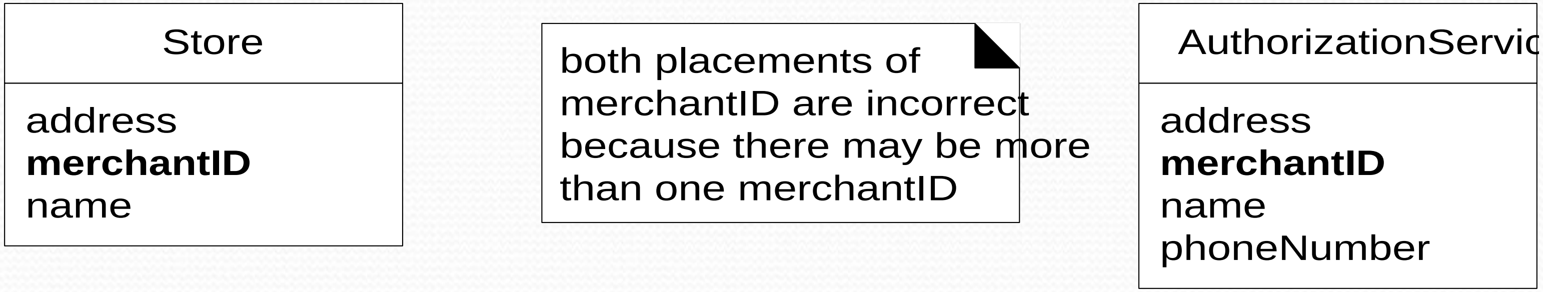


Multiple Associations

- It is not uncommon for two types to have multiple associations between them



Modeling Association As Class



Store

address
merchantID
name

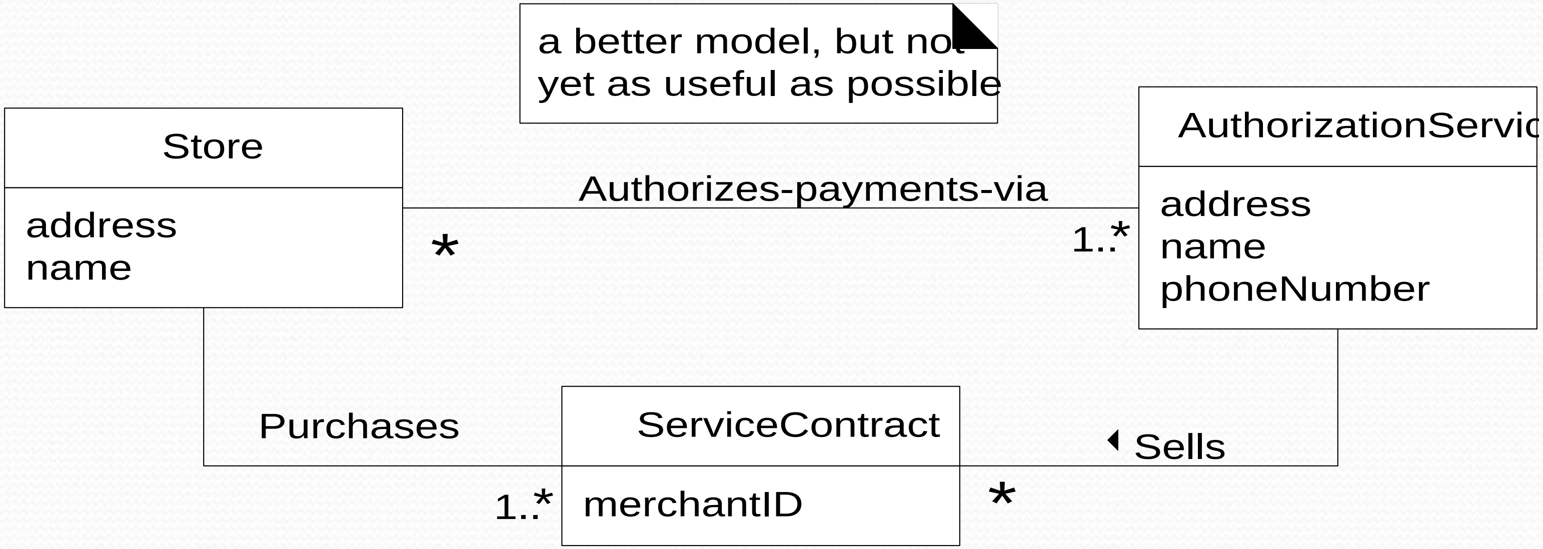
both placements of merchantID are incorrect because there may be more than one merchantID

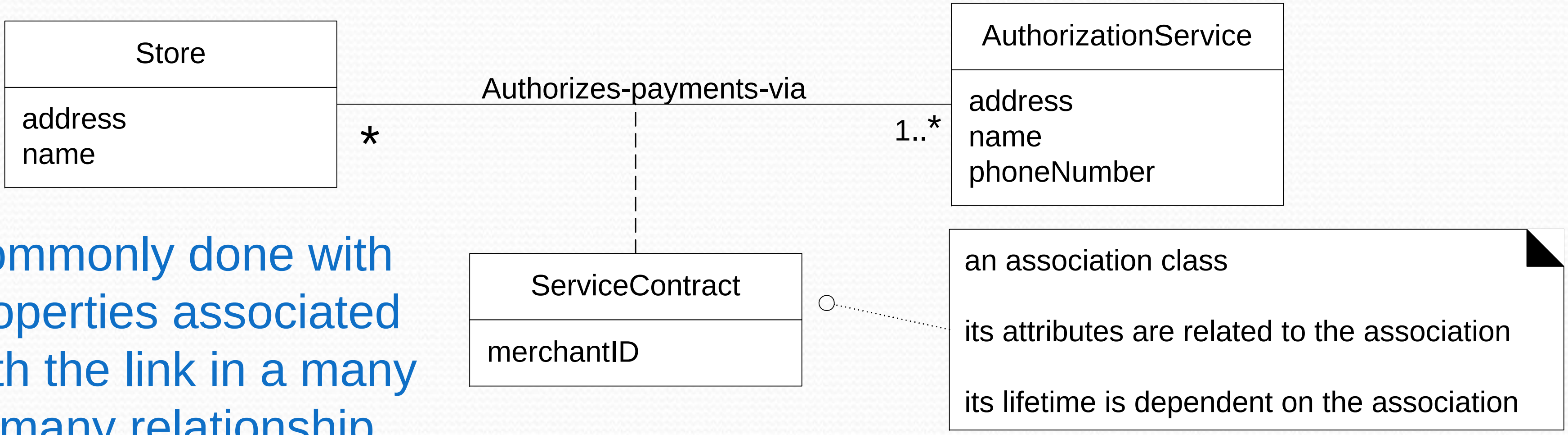
AuthorizationServic

address
merchantID
name
phoneNumber

- If a class C can simultaneously have many values for the same kind of attribute A, do not place attribute A in class C. Place A in another class associated with C.

Modeling Association As Class





Commonly done with properties associated with the link in a many to many relationship

Association Class Guideline

- If an attribute is related to an association
- Instance of the association class have a lifetime dependency on the association
- Many-to-many association between two concepts and information associated with the association itself

Relationships

- Four types of relationships:
  - Association
  - Dependency
  - Aggregation (Composition)
  - Generalization (Specialization)

Dependency

- If a class is depended on another class, its relationship is a dependent relationship

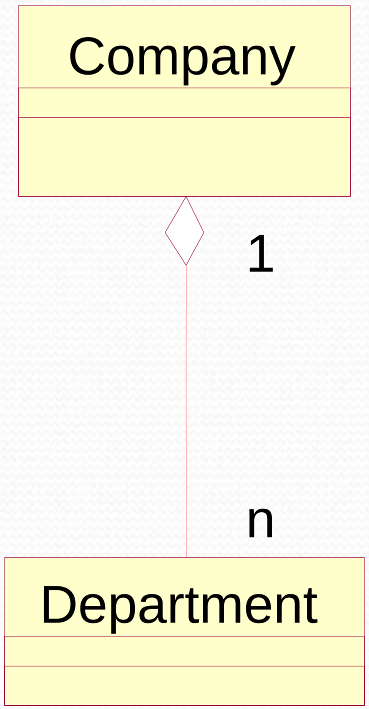
- Dependency relationship is represented by dashed line between the classes

- Example: relationship is a weaker form of relationship showing an interest between a client and a supplier
  - A dependency is shown as a dashed line pointing from the client to the supplier

Relationships

- Four types of relationships:
  - Association
  - Dependency
  - Aggregation (Composition)
  - Generalization (Specialization)

Aggregation

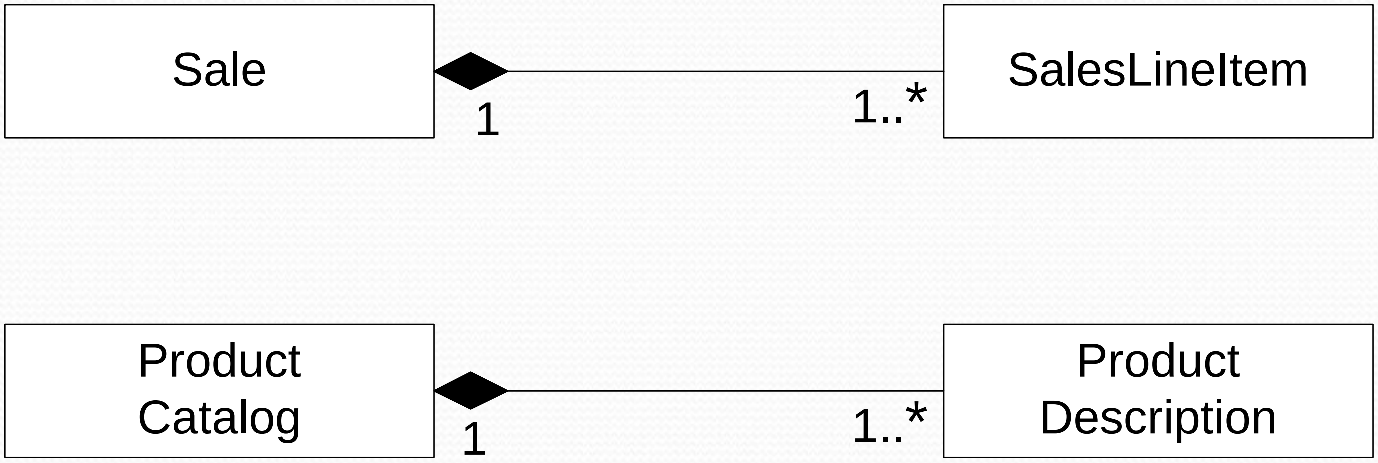


- Model a whole / part relationship
- Has-a relationship
- One class (the whole) consists of another class (the parts)
- An aggregation is represented as an open diamond with diamond on the aggregate end

Composition

- The lifetime of the part is bound within the lifetime of the composite – a stronger term

- Not that significant for domain modeling



Larman Fig 31.18

Relationships

- Four types of relationships:
  - Association
  - Dependency
  - Aggregation (Composition)
  - Generalization (Specialization)

Generalization

- Defines relationships between superclass (general concept) and subclasse(specialized concept)

- All members of a conceptual subclass set are members of their superclass set

Generalization

- The conceptual subclass must conform to 100% of the superclasses attributes and associations , thus termed the ***100% rule***

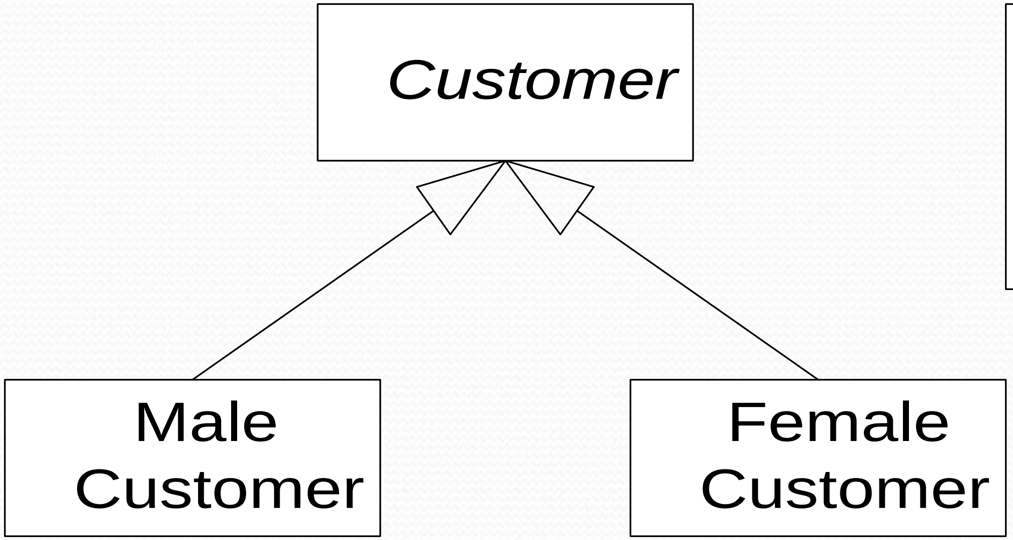
- The conceptual subclass ***is-a-kind of*** the superclass. Often is called the ***is-a rule***

Specialization

- When is it appropriate to define a conceptual subclass?

- When is it useful to show conceptual class partition?

Generalization / Specialization



Correct subclasses

But useful?

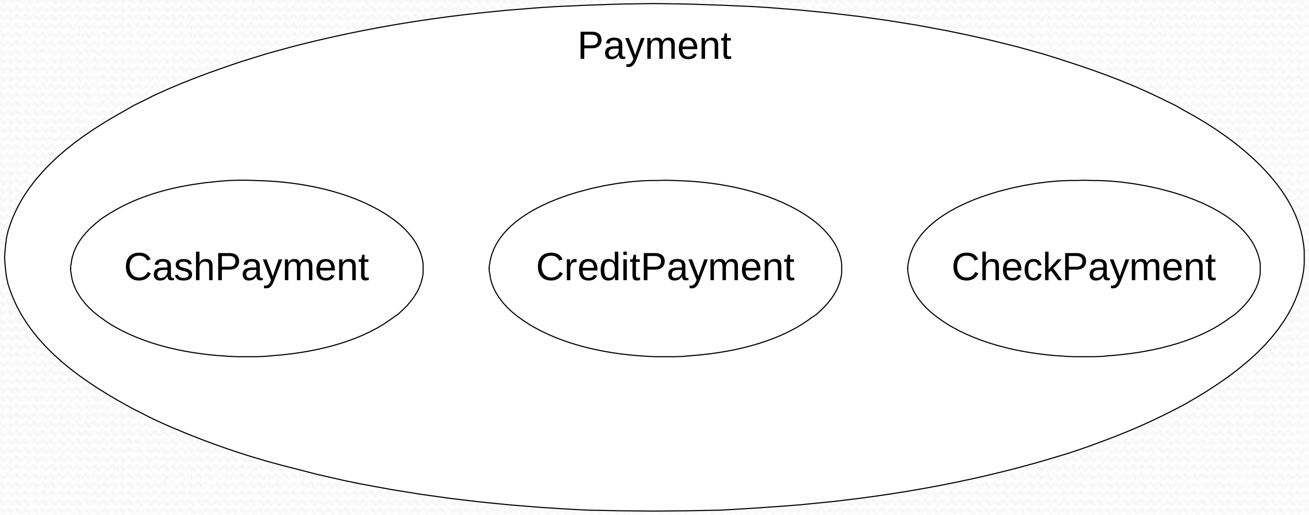
Larman Fig 31.6

Specialization

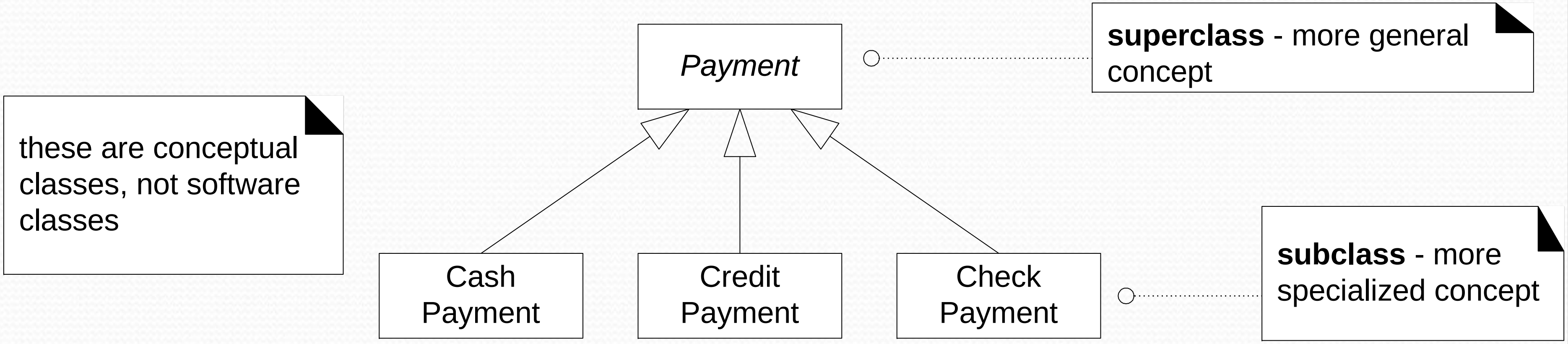
- When the subclass has additional attributes of interest.
- When the subclass has additional associations of interest.
- Adhere to setting :
  - Definition conformance - *100% rule*
  - Membership conformance - *is-a rule*.

Specialization

- What would be an appropriate Conceptual Diagram for a POS system that accepts the following three types of payments: cash payment, credit payment, cheque payment?



Generalization / Specialization



Payment

superclass - more general concept

these are conceptual classes, not software classes

Cash Payment

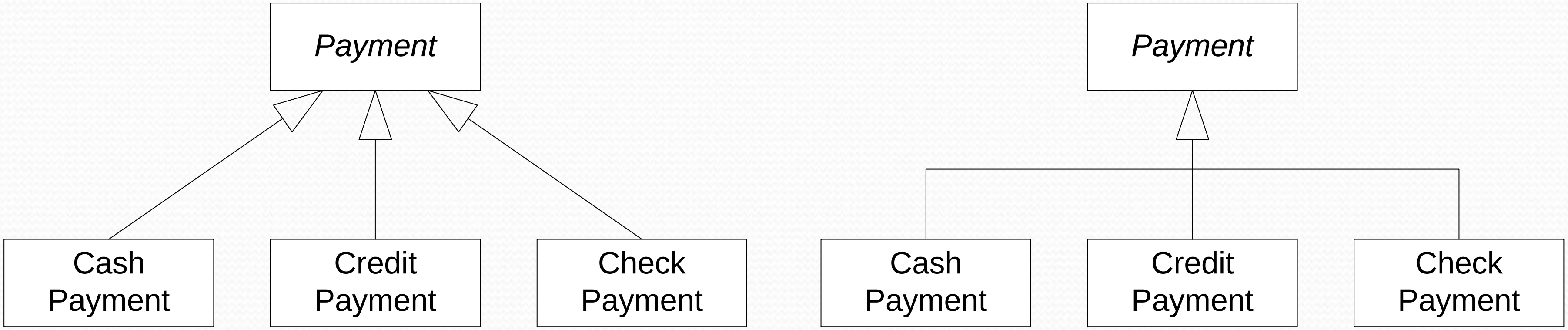
Credit Payment

Check Payment

subclass - more specialized concept

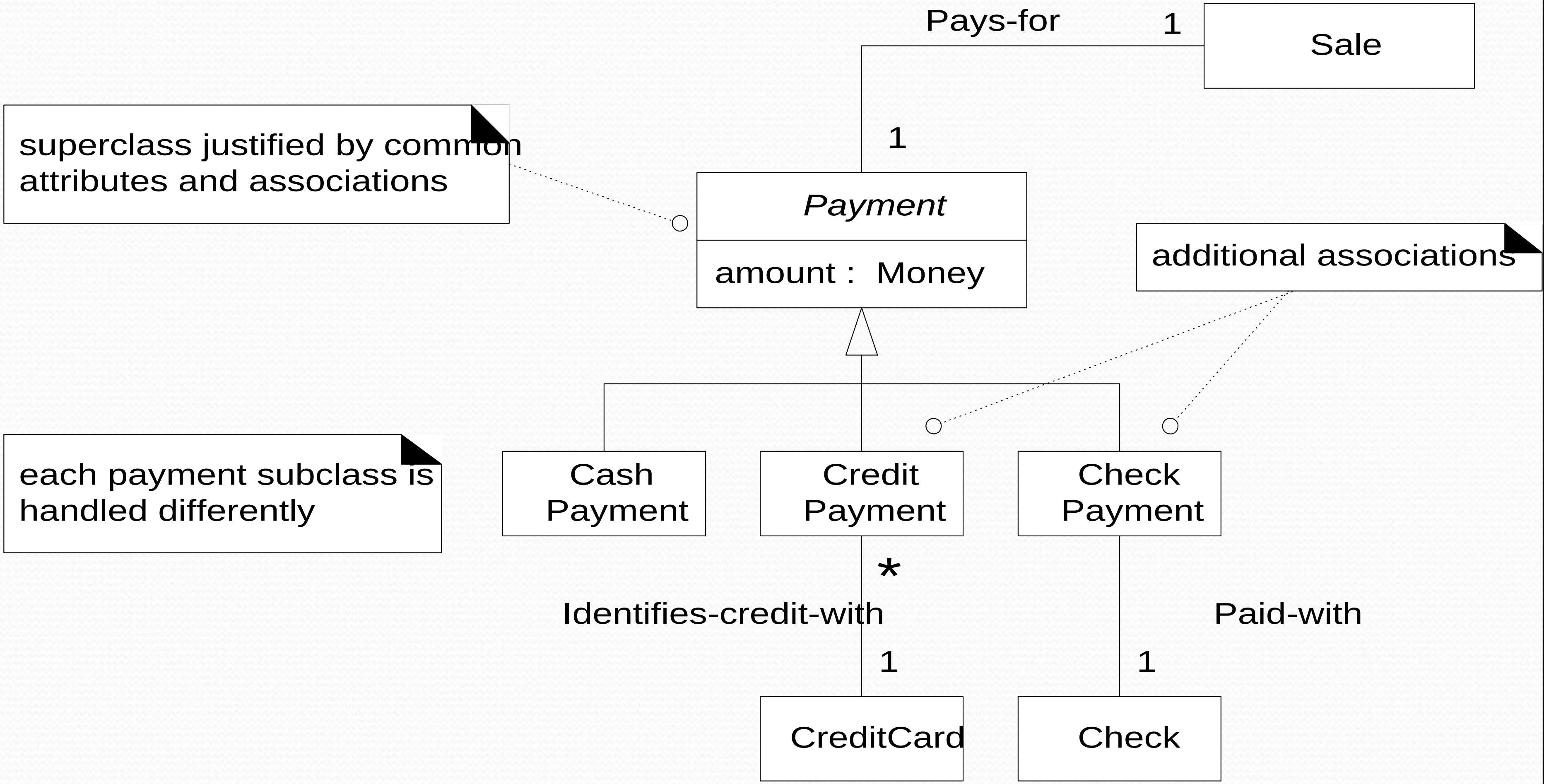
Larman Fig 31.1

Generalization / Specialization



Larman Fig 31.2

Generalization Justified



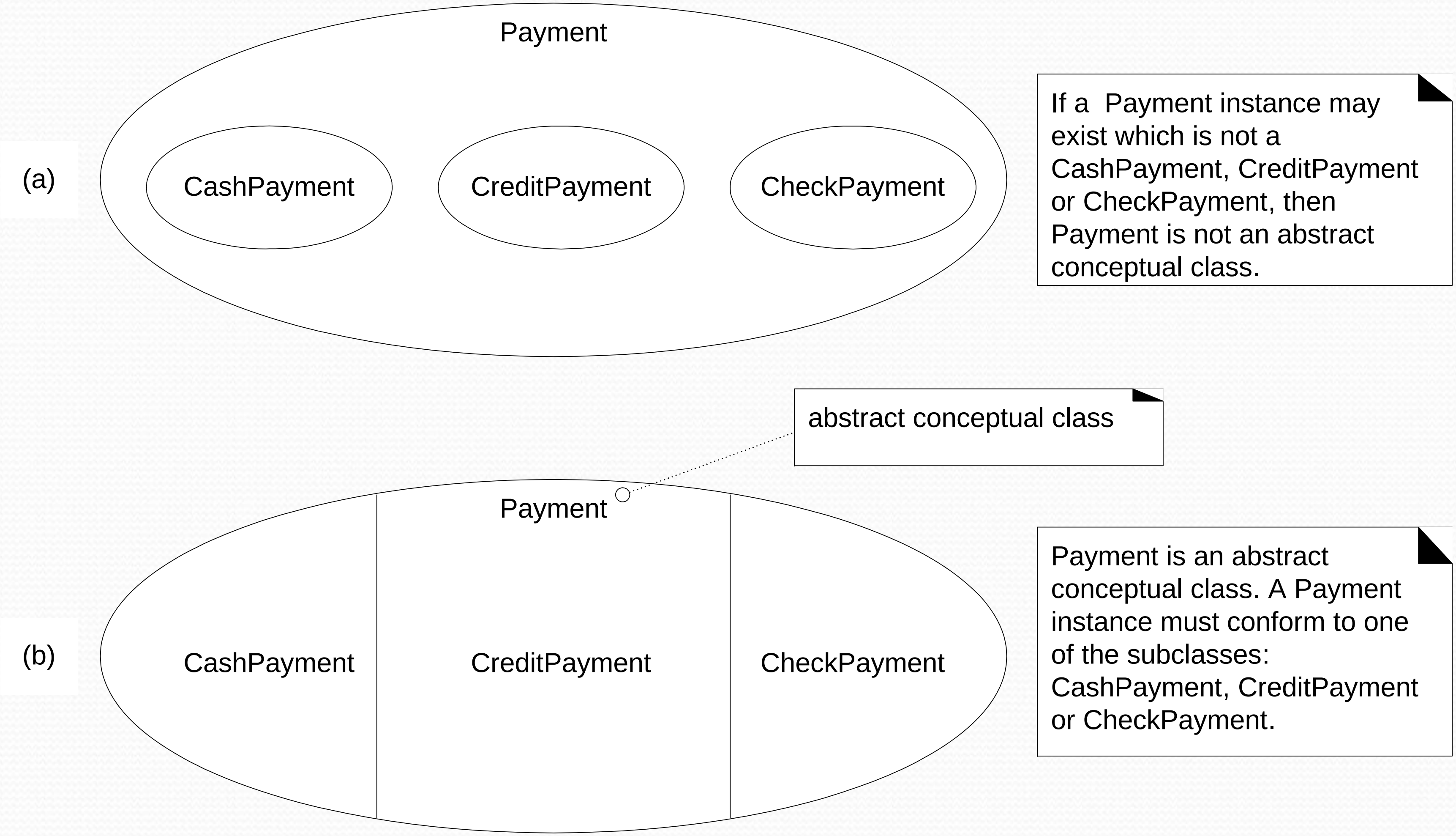
Larman Fig 31.7

Abstract Superclass

- It is useful to identify abstract classes in the domain model

- Every member of a class C must also be a member of a subclass, then class C is an abstract conceptual class.

Abstract Conceptual Class



(a)

Payment

CashPayment

CreditPayment

CheckPayment

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

abstract conceptual class

(b)

Payment

CashPayment

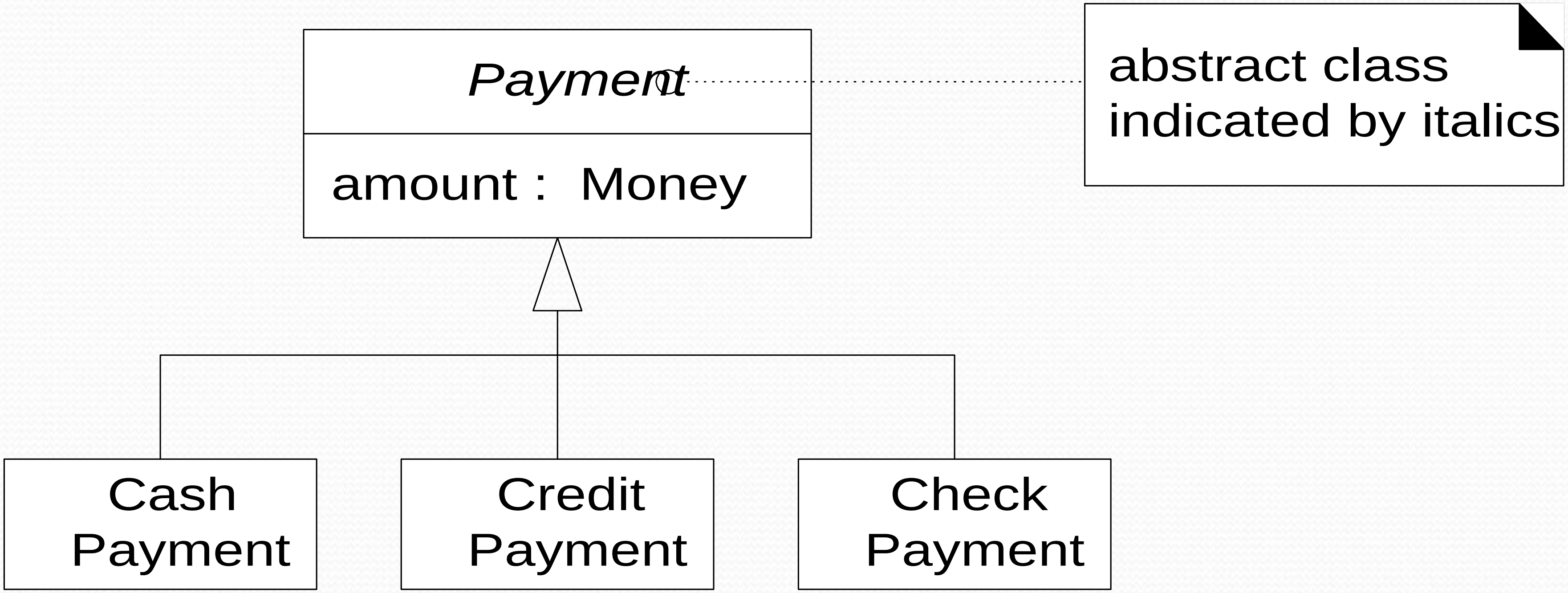
CreditPayment

CheckPayment

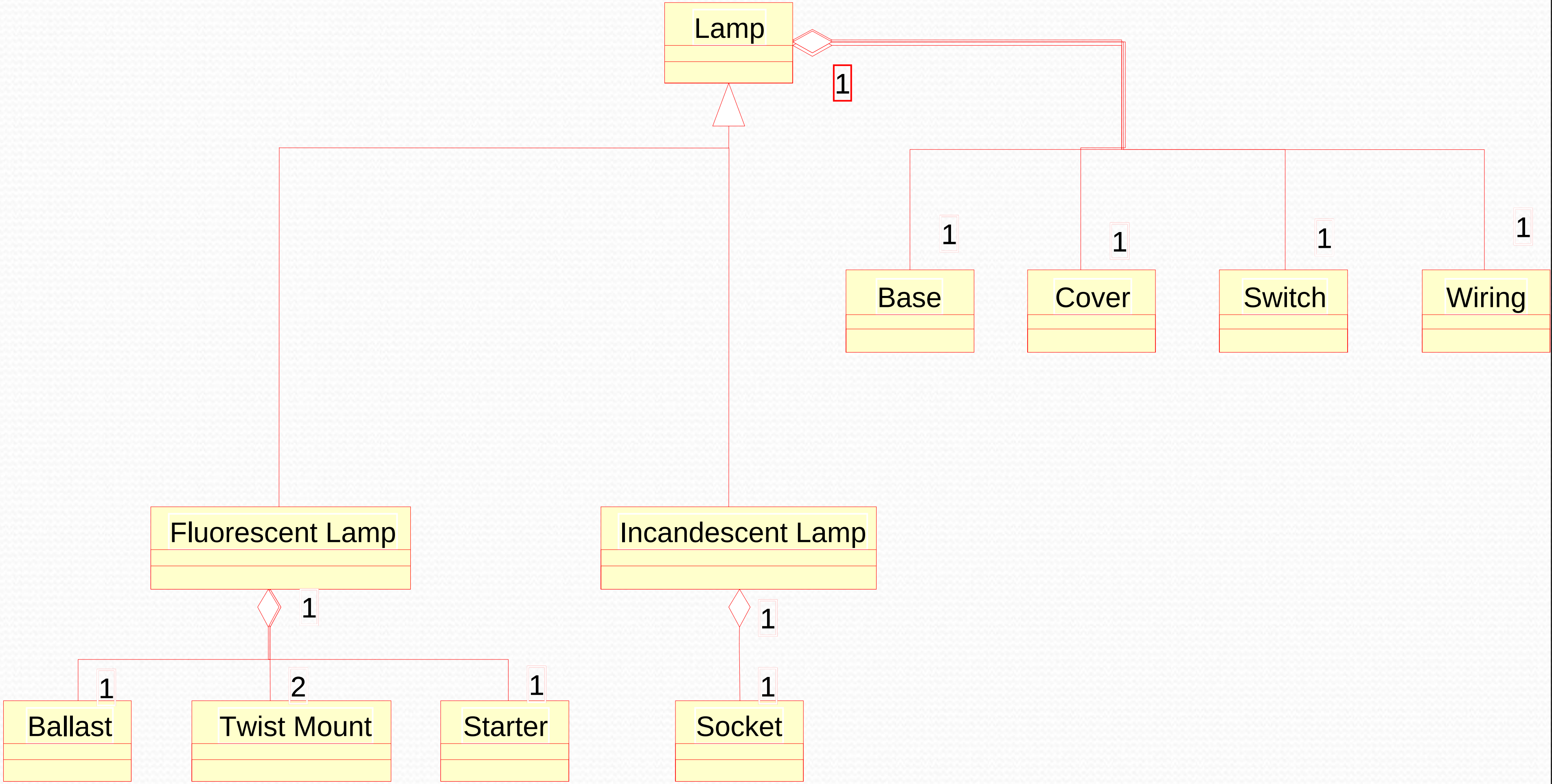
Payment is an abstract conceptual class. A Payment instance must conform to one of the subclasses: CashPayment, CreditPayment or CheckPayment.

Larman Fig 31.11

Abstract Conceptual Class



Aggregation vs Generalization



Summary

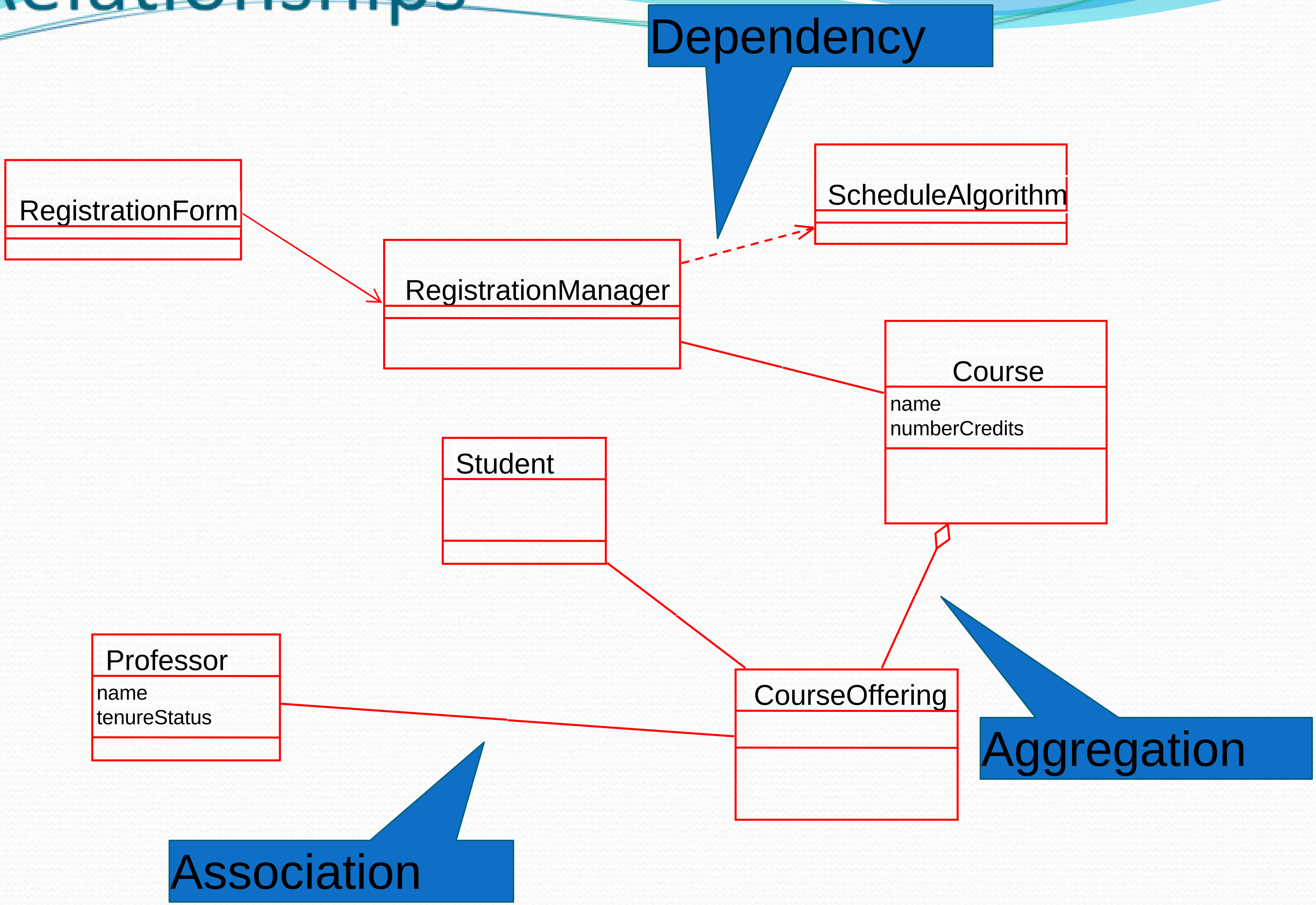
- **Association:** a bi-directional connection between classes
    - An association is shown as a line connecting the related classes

- **Aggregation:** a stronger form of relationship where the relationship is between a whole and its parts
    - An aggregation is shown as a line connecting the related classes with a diamond next to the class representing the whole

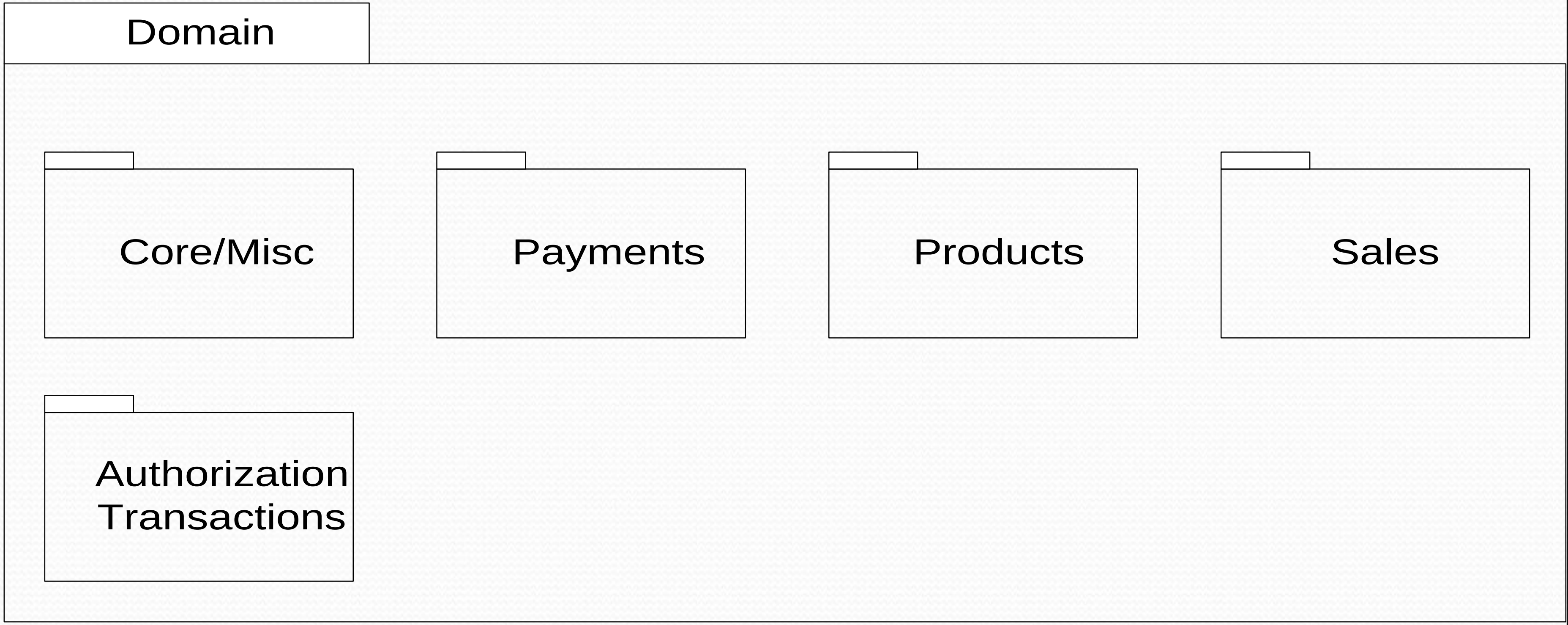
- **Dependency:** a weaker form of relationship showing an interest between two classes, shown as a dashed line

- **Generalization:** relationship in which one model element (the child) is based on another model element (the parent).

Relationships

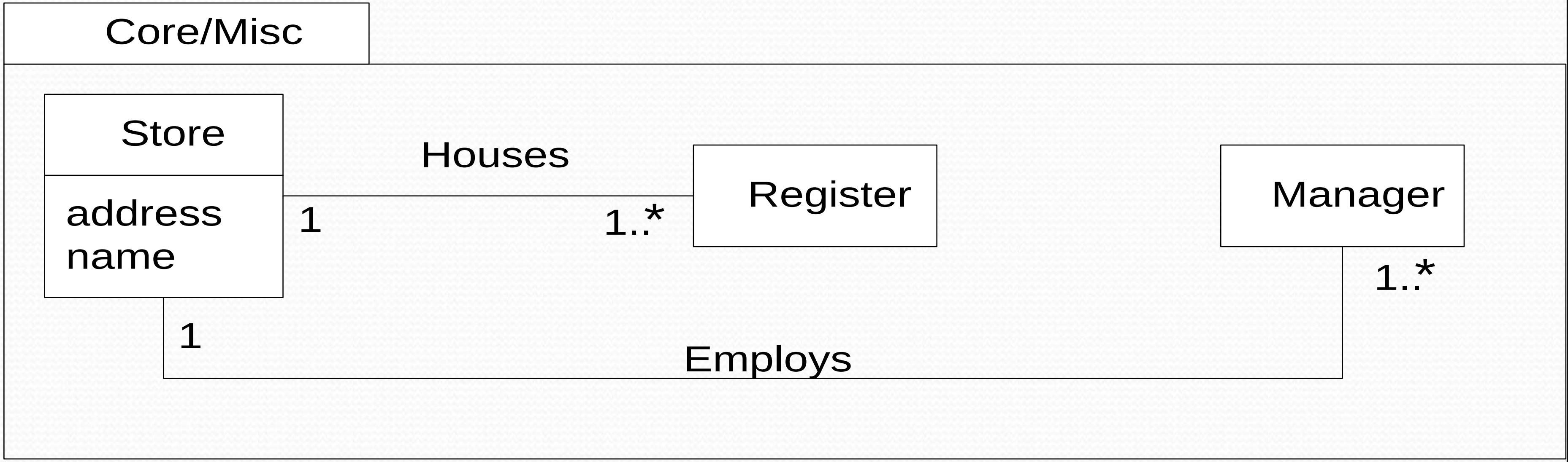


Domain Model Packages



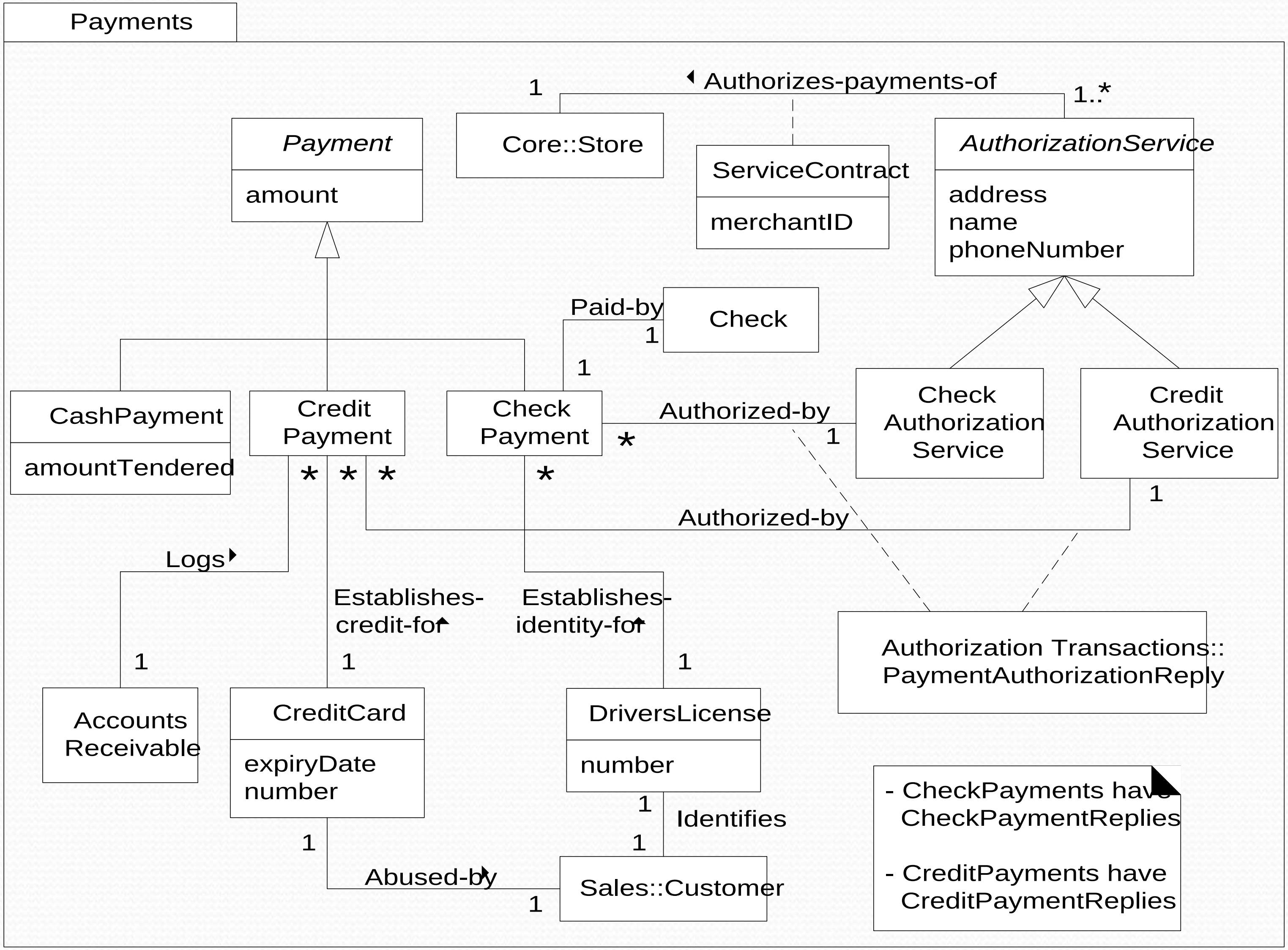
Larman Fig 31:29 – POS Domain Model Example

Domain Model Packages



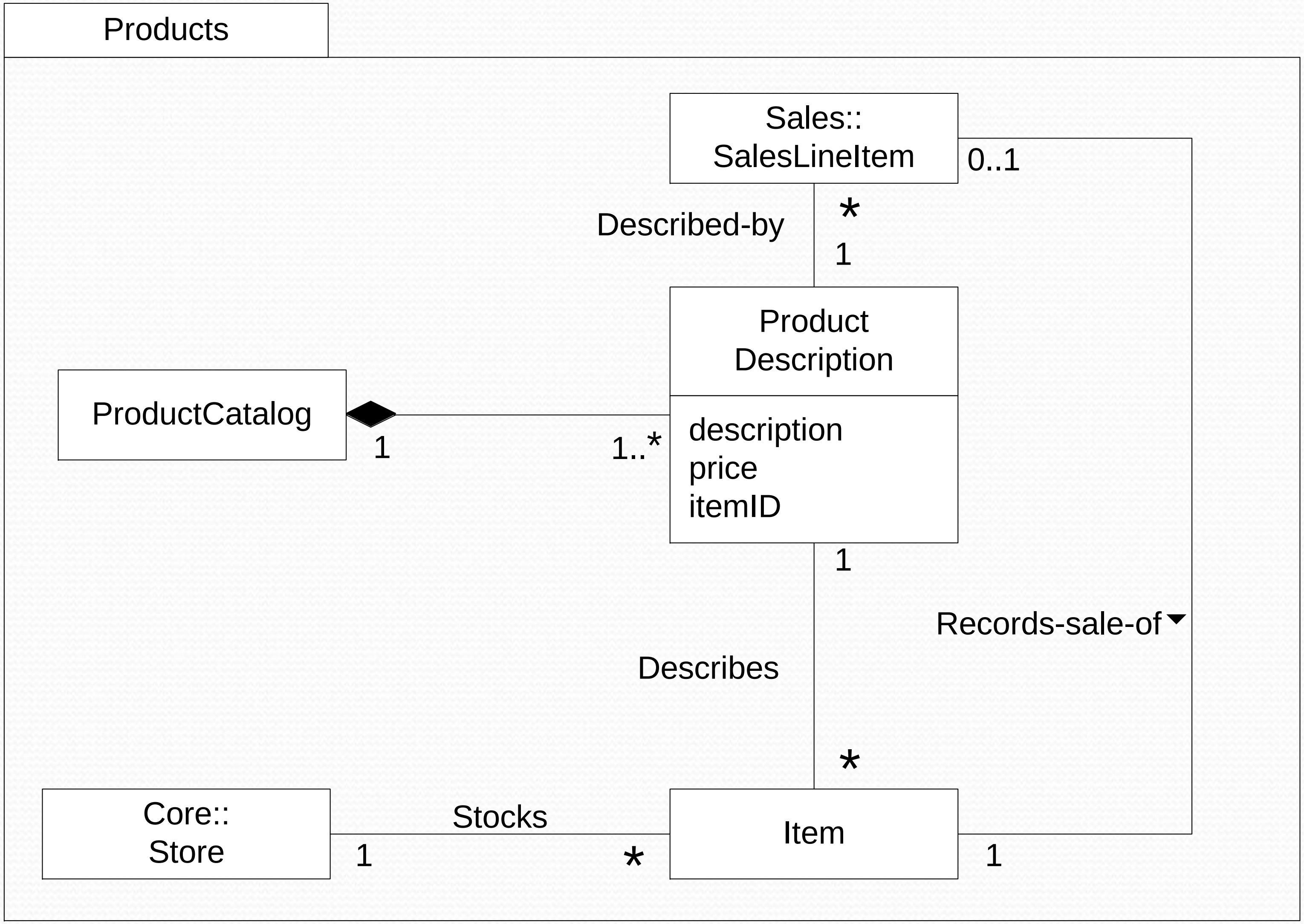
Larman Fig 31:30

Domain Model Packages



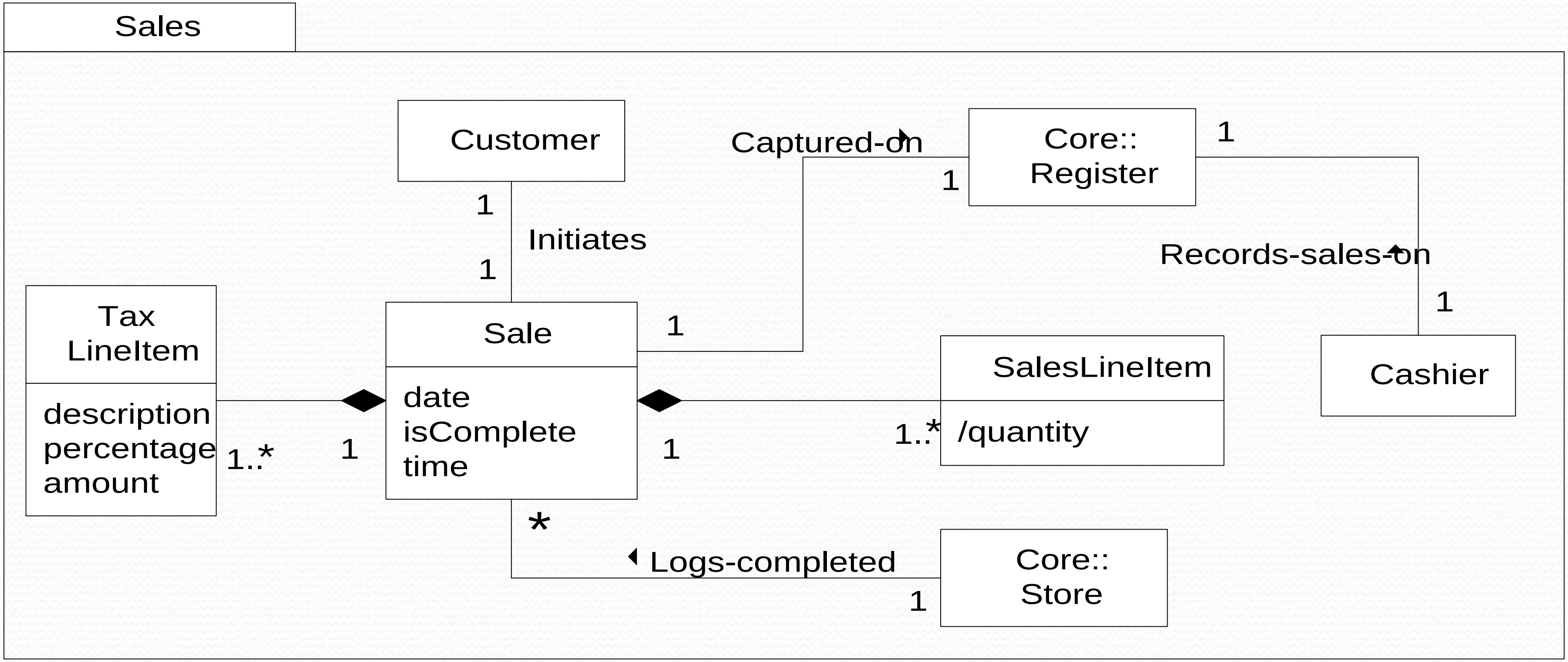
Larman Fig 31:31

Domain Model Packages



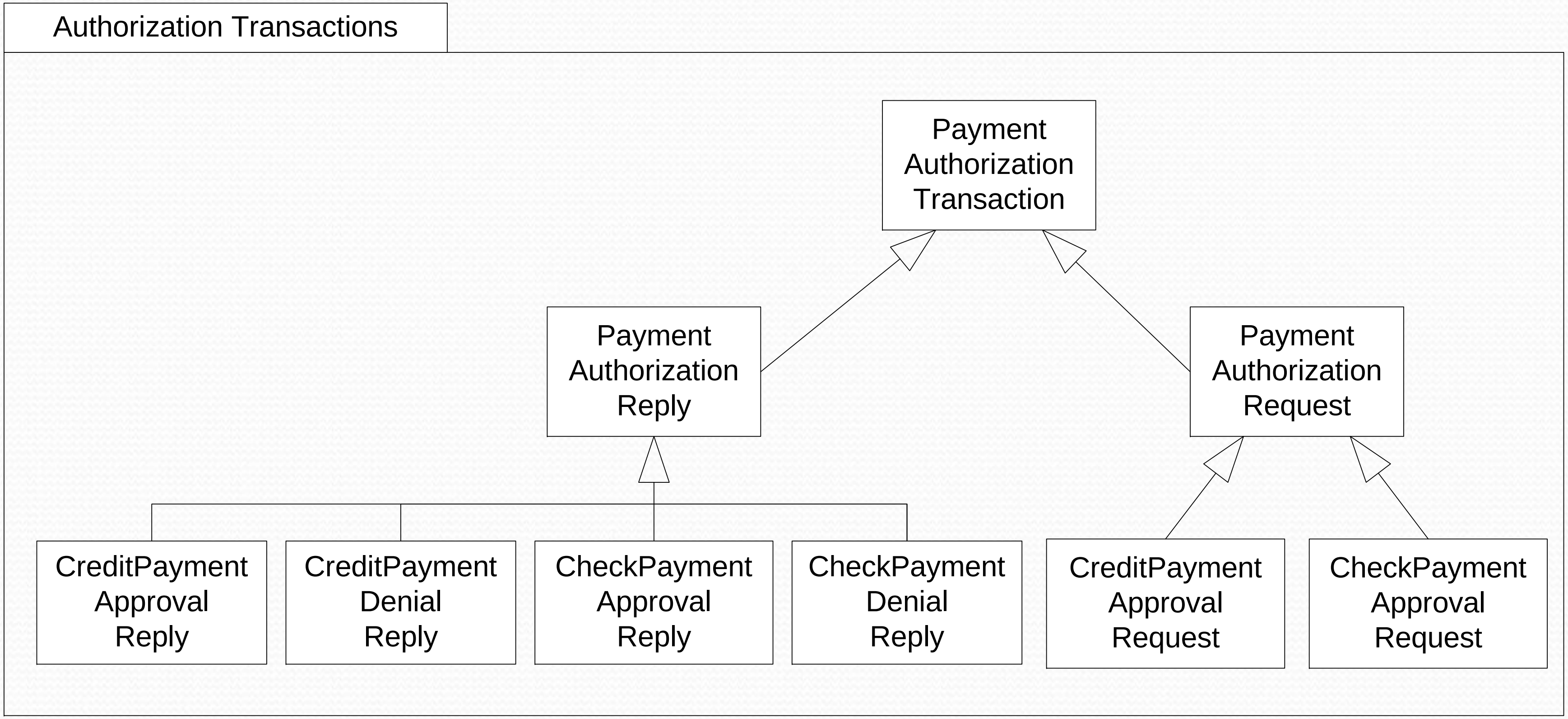
Larman Fig 31:32

Domain Model Packages



Larman Fig 31:33

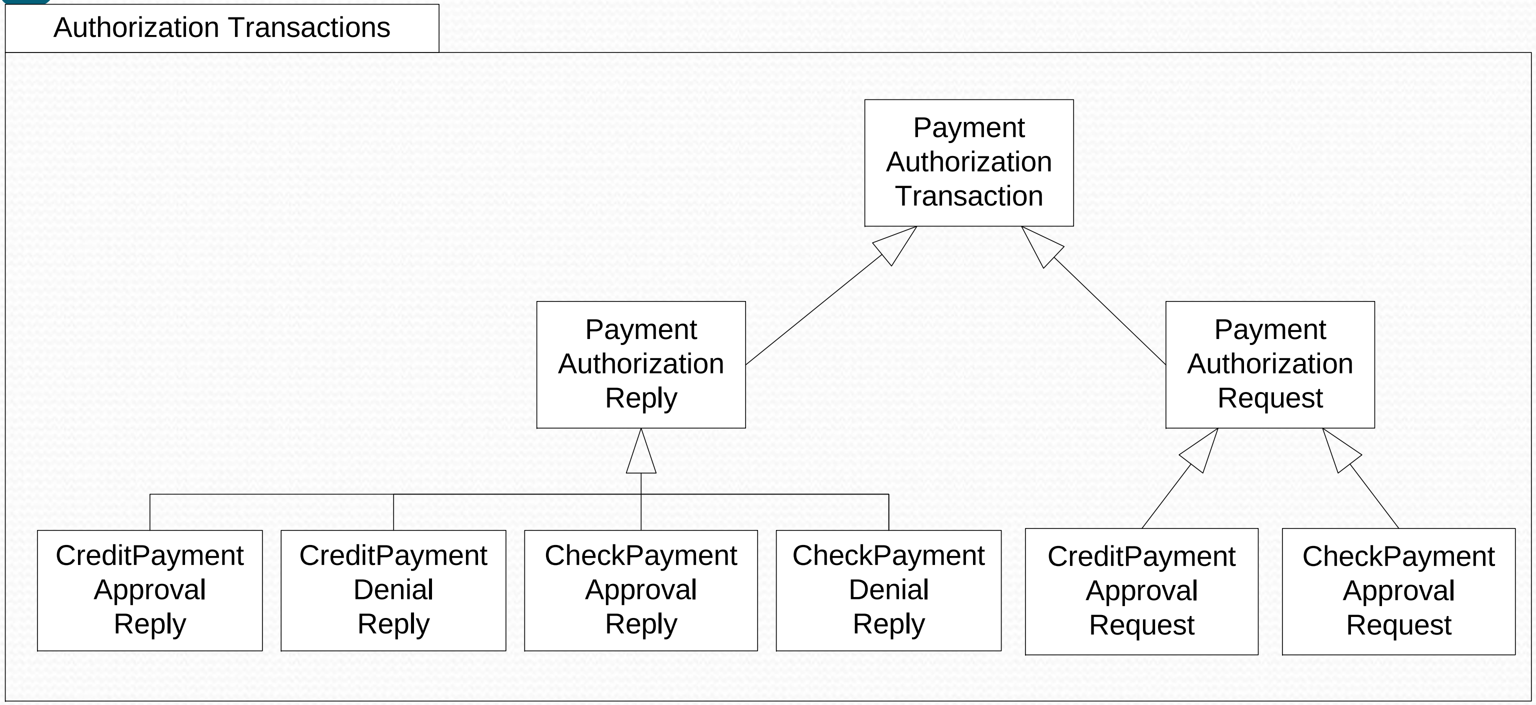
Domain Model Packages



Authorization Transactions
Payment Authorization Transaction
Payment Authorization Reply
Payment Authorization Request
CreditPayment Approval Reply
CreditPayment Denial Reply
CheckPayment Approval Reply
CheckPayment Denial Reply
CreditPayment Approval Request
CheckPayment Approval Request

Larman Fig 31:34

Fig. 32.34


Authorization Transactions
Payment Authorization Transaction
Payment Authorization Reply
Payment Authorization Request
CreditPayment Approval Reply
CreditPayment Denial Reply
CheckPayment Approval Reply
CheckPayment Denial Reply
CreditPayment Approval Request
CheckPayment Approval Request