Domain Modeling

CSE 5324, Summer 2017

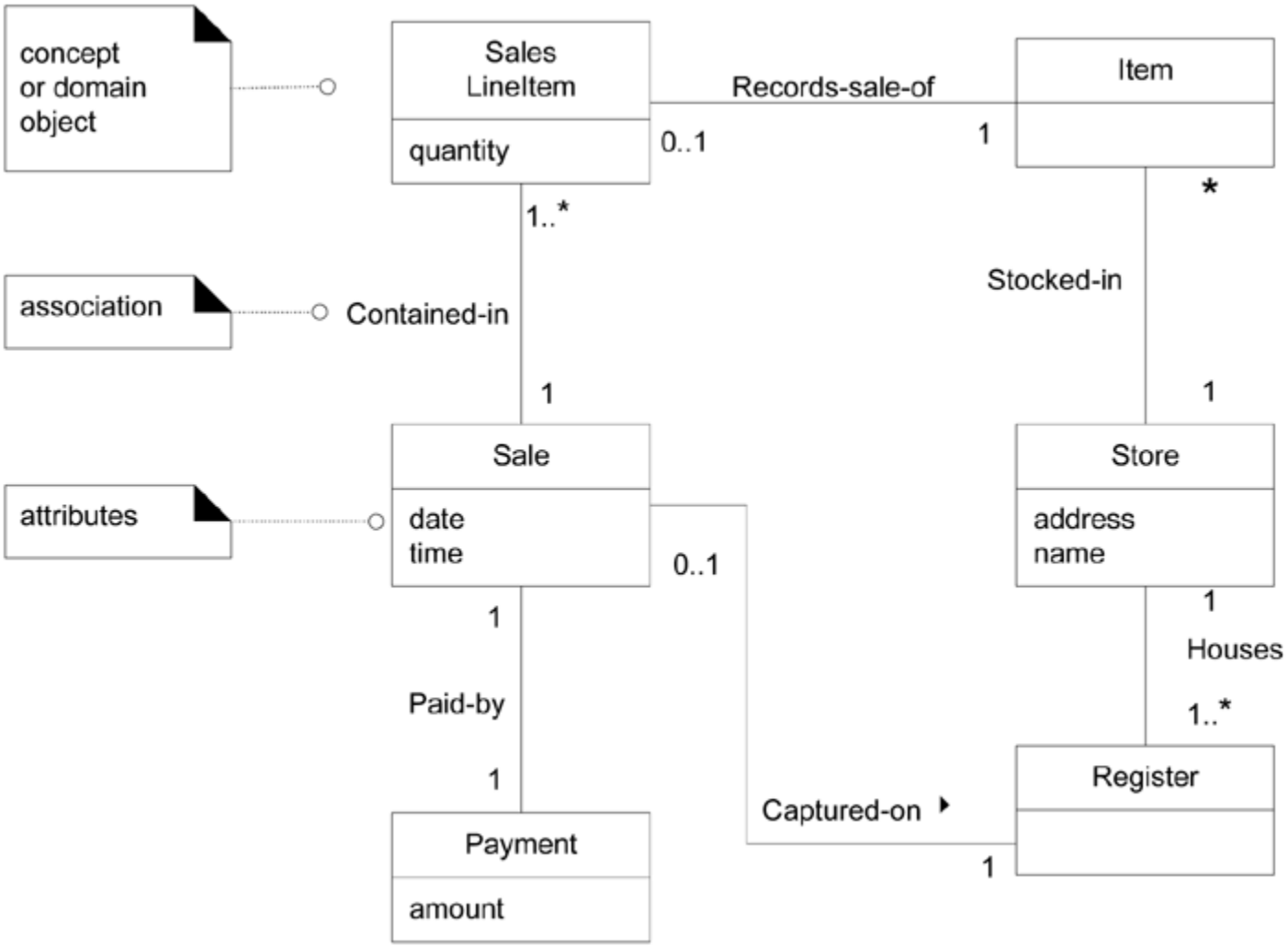
Outline

- What and Why?
- How to Find Conceptual Classes
- Conceptual Super- and Sub-Classes
- Associations and Association Classes
- Package Diagrams

Domain Model

- An abstract representation of the key entities (physical or conceptual) and their relationships in a problem domain.
- NOT a model of software objects
  - Typically no software artifacts such as a window or database
  - No responsibilities or methods
- Should be independent from the design or implementation concerns

Example


concept
or domain
object
association
attributes
Sales
LineItem
quantity
0..1
Records-sale-of
1
Item
*
1..*
Stocked-in
Contained-in
1
1
Sale
date
time
Store
address
name
0..1
1
Houses
1..*
Paid-by
1
Register
Captured-on
Payment
amount
1

Why?

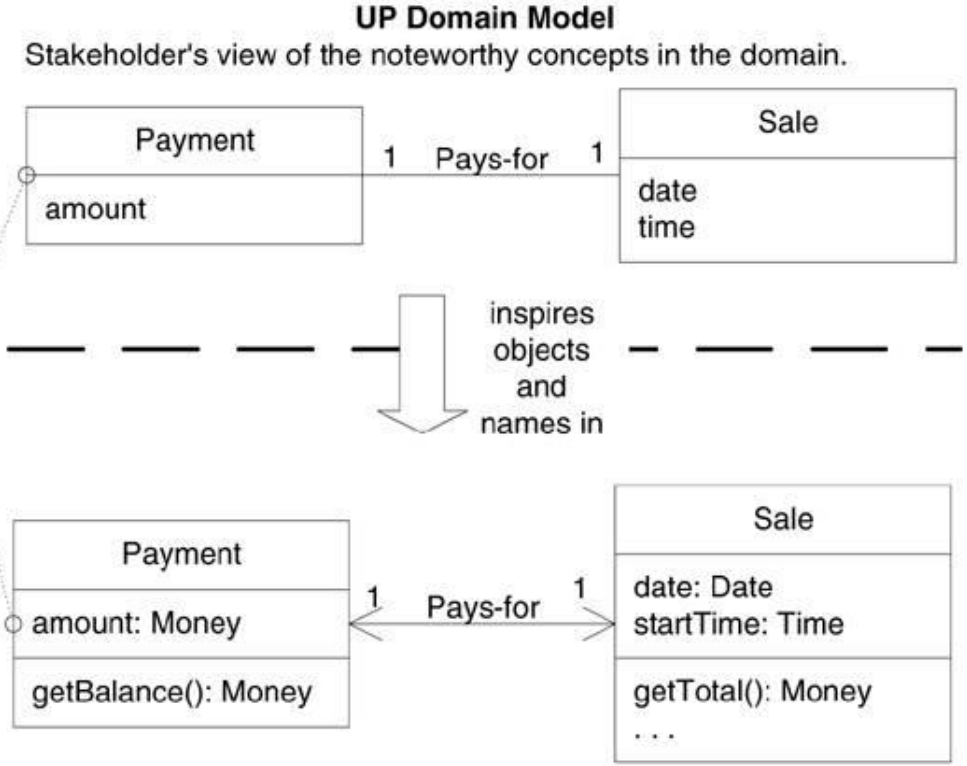
- Helps to clarify and validate the understanding of the problem domain
- Defines a vocabulary that can be used to facilitate communication
- Reduces the representational gap between the mental model and the OO model

Domain vs Design Model

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.



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.

Major Steps

- Find the conceptual classes
- Draw them as classes in a UML class diagram
- Add associations and attributes

How to Find Conceptual Classes

- Identify the nouns and noun phrases in textual descriptions of a domain
- Guideline: A mechanical noun-to-class mapping isn't possible. Words in natural languages may be ambiguous.
  - Different nouns may represent the same conceptual classes
  - Some nouns may be attributes, instead of classes
  - Some nouns may not need to be represented

Example

Main Success Scenario (or Basic Flow):

1. **Customer** arrives at a **POS checkout** with **goods** and/or **services** to purchase.
2. **Cashier** starts a new **sale**.
3. **Cashier** enters **item identifier**.
4. System records **sale line item** and presents **item description, price**, and running **total**. Price calculated from a set of price rules.

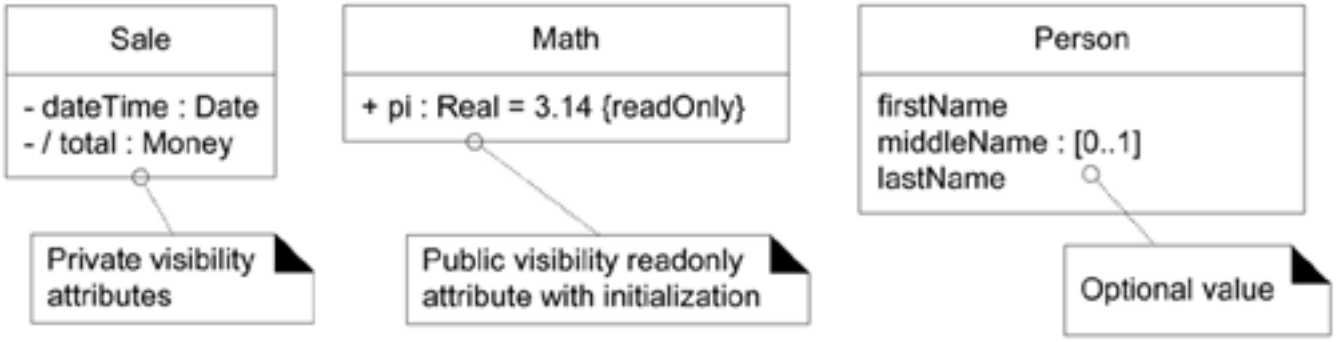
Cashier repeats steps 2-3 until indicates done.

5. System presents total with **taxes** calculated.
6. Cashier tells Customer the total, and asks for **payment**.
7. Customer pays and System handles payment.
8. System logs the completed **sale** and sends sale and payment information to the external **Accounting** (for accounting and **commissions**) and **Inventory** systems (to update inventory).

Attributes

- A logic data value of an object
  - Identified when there is a need to remember information
- For example, a receipt in the Process Sale use case normally includes a date and time, the store name and address, and the cashier ID, among other things.
  - What attributes can be identified?

UML Notation



Attribute Types

- Attributes are preferably data types:
  - Boolean, Date, Number, Character, String, Time
  - Address, Color, Geometrics (Point, Rectangle), Phone Number, Social Security Number, Universal Product Code, SKU, ZIP or postal codes, enumerated types
- In contrast, attributes in software implementation can be of any type.

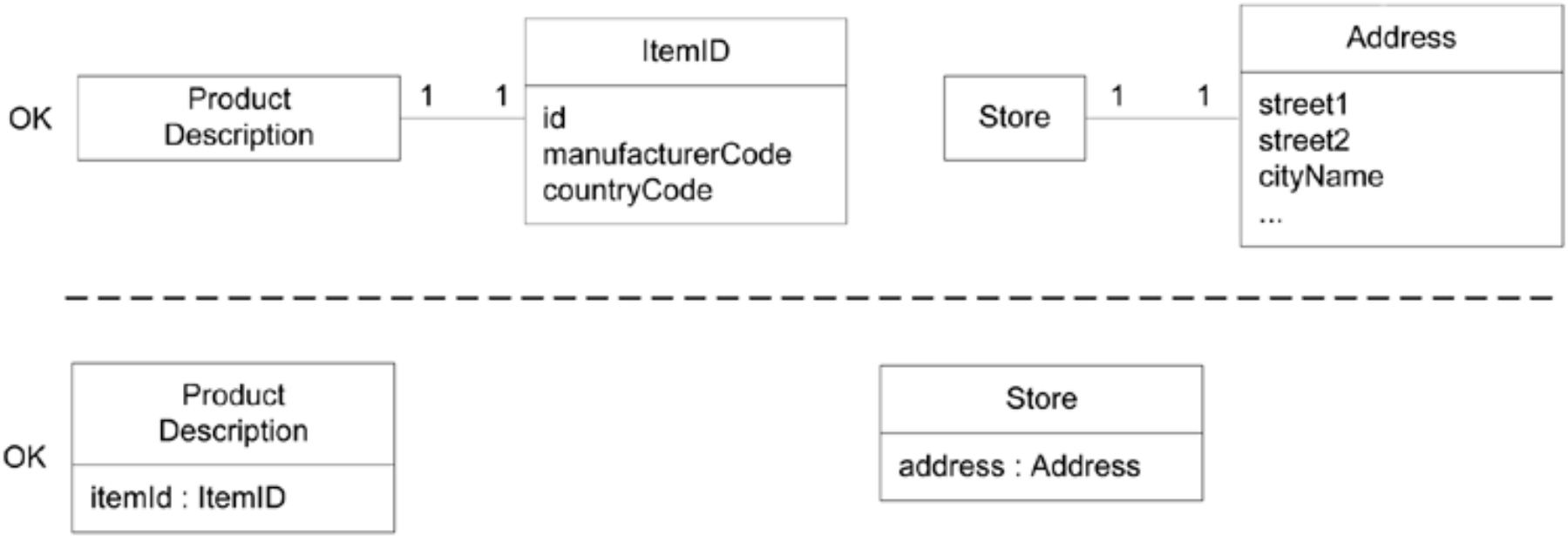
Data Types

- Equality tests are not based on identity, but on value.
- Data types are typically immutable.
  - If you change it, it becomes a different value.

When to Define New Data Types?

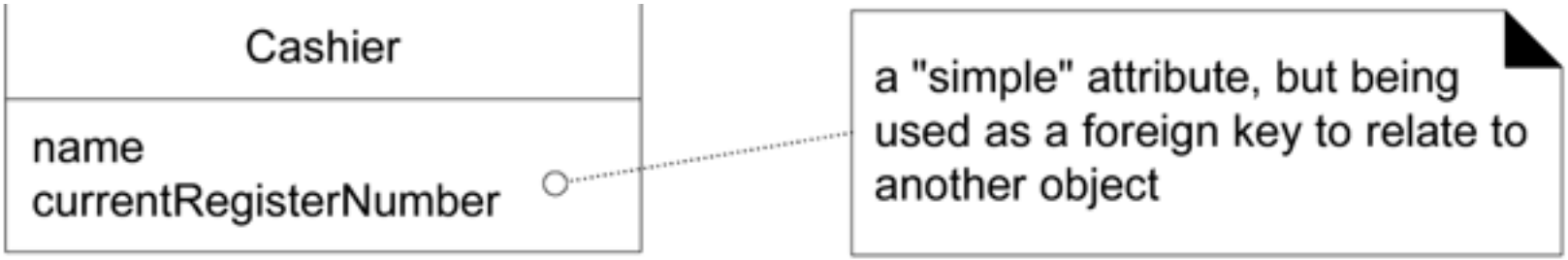
- Represent a number or string as a new data type class if
  - It is composed of separate sections, e.g., phone number, name of person
  - There are operations associated with it, such as parsing or validation
  - It has other attributes, e.g., promotional price may have a start and/or end date.
  - It is a quantity with a unit, e.g., payment amount

New Data Types

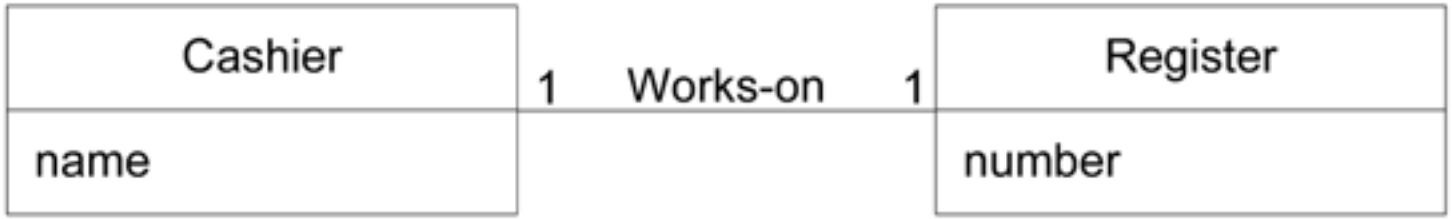


No Foreign Key Attributes

Worse



Better



Attributes vs Classes

- If we do not think of some conceptual class X as a number or text in the real world, X is probably a conceptual class, not an attribute.
- For example, a student is probably a conceptual class, not an attribute.

Guideline

- Avoid a waterfall-mindset big-modeling effort to make a thorough or correct model
- Limit domain modeling to no more than a few hours per iteration.
- The domain model is primarily created during elaboration iterations.
  - Usually no domain modeling during inception

Generalization

- The activity of identifying commonality among concepts and defining superclass and subclass relationships.
- Helps to understand concepts in more general, refined, and abstract terms.
- Leads to better understanding and a reduction in repeated information

Superclass vs Subclass

- Is-A Rule: All the members of a subclass set must be members of their super class set
- 100% Rule: 100% of the conceptual superclass's definition, i.e., attributes and associations, should be applicable to the subclass.

When to Create a Subclass?

Conceptual Subclass Motivation	Examples
The subclass has additional attributes of interest.	Payments— not applicable. Library— *Book*, subclass of *LoanableResource*, has an ISBN attribute.
The subclass has additional associations of interest.	Payments— *CreditPayment*, subclass of *Payment*, is associated with a CreditCard. Library— *Video*, subclass of *LoanableResource*, is associated with Director.
The subclass concept is operated upon, handled, reacted to, or manipulated differently than the superclass or other subclasses, in ways that are of interest.	Payments— *CreditPayment*, subclass of *Payment*, is handled differently than other kinds of payments in how it is authorized. Library— *Software*, subclass of *LoanableResource*, requires a deposit before it may be loaned.
The subclass concept represents an animate thing (for example, animal, robot) that behaves differently than the superclass or other subclasses, in ways that are of interest.	Payments— not applicable. Library— not applicable. Market Research— *MaleHuman*, subclass of *Human*, behaves differently than *FemaleHuman* with respect to shopping habits.

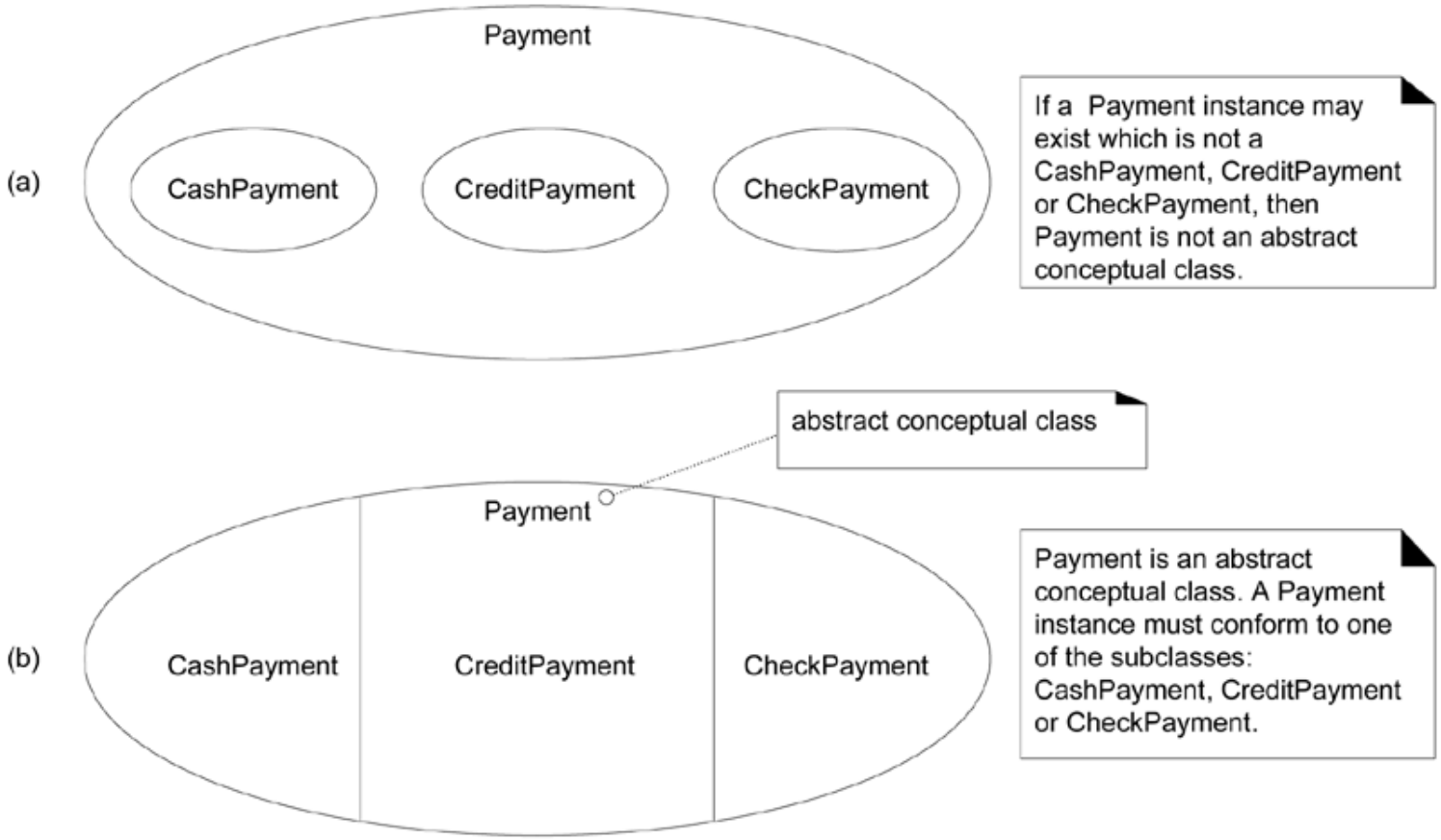
When to Create a Superclass?

- Create a superclass in a generalization relationship to subclasses when:
  - The potential conceptual subclasses represent variations of a similar concept.
  - The subclasses will conform to the 100% and Is-A rules
  - All subclasses have the same attribute that can be factored out and expressed in the superclass
  - All subclasses have the same association that can be factored out and related to the superclass.

Abstract Conceptual Classes

- If every member of a class C must also be a member of a subclass, then class C is called an abstract class.

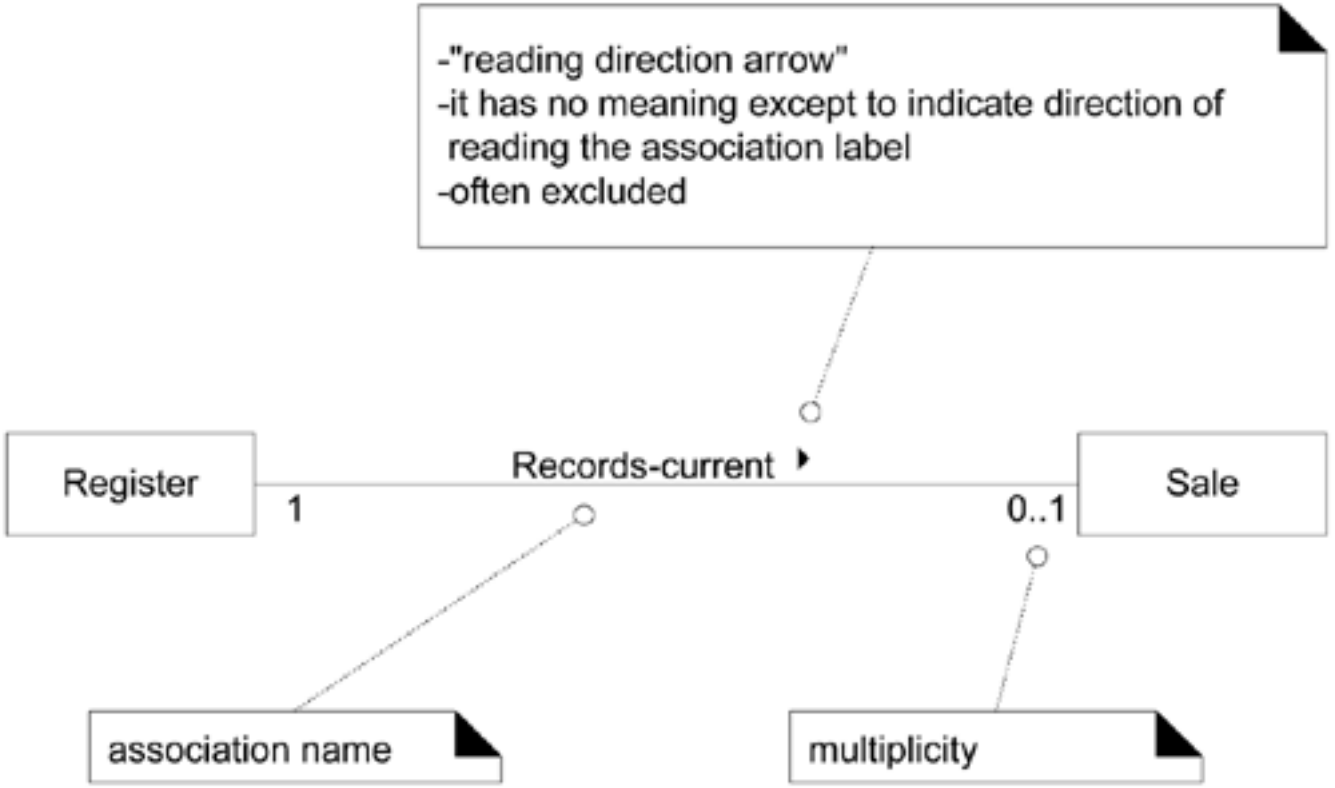
Abstract Conceptual Classes (2)



Association

- A relationship between classes that indicates some meaningful and interesting connection.
  - A conceptual, not software, perspective
- Name an association based on a ClassName-VerbPhrase-ClassName format.
  - Sale Paid-By CashPayment
  - Player Is-On Square

UML Notation



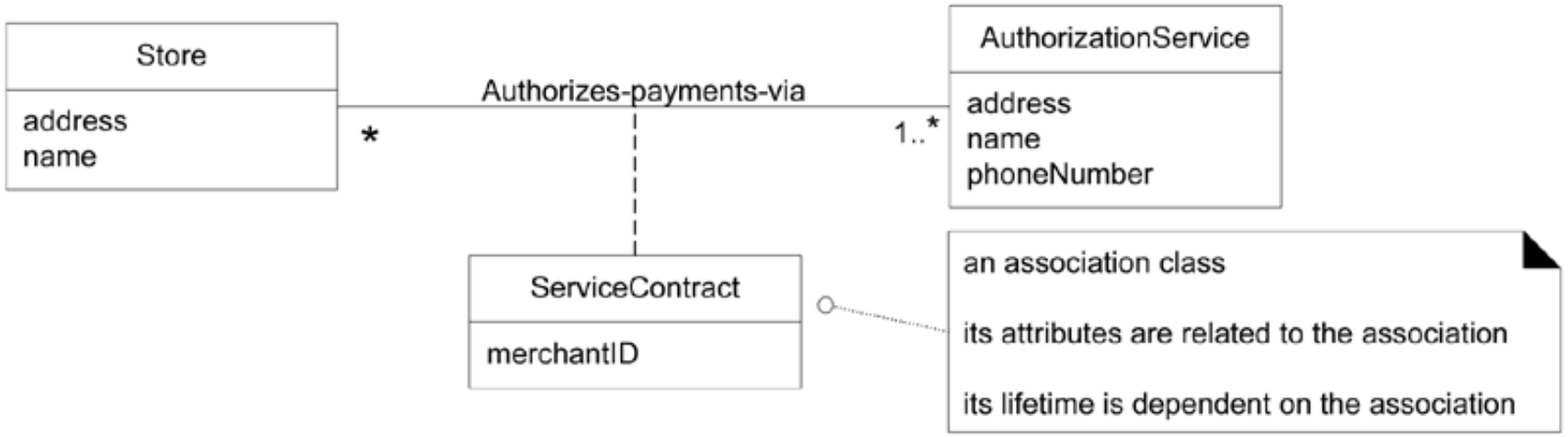
The Memory Test

- Knowledge of a relationship that needs to be preserved for some duration
  - In a POS system, do we need to remember what SalesLineItem instances are associated with a Sale instance?
  - Do we need to remember that a particular Cashier looking up particular ProductDescriptions?

Association Classes

- If a class C can simultaneously have many values for the same kind of attribute A, do not place attribute A in C. Instead, place it in another class that is associated with C.
- Clues that an association class might be useful:
  - An attribute is related to an association.
  - Instances of the association class have a lifetime dependency on the association.
  - The presence of a many-to-many association between two concepts.

Example


Store
address
name
Authorizes-payments-via
*
1..*
AuthorizationService
address
name
phoneNumber
ServiceContract
merchantID
an association class
its attributes are related to the association
its lifetime is dependent on the association

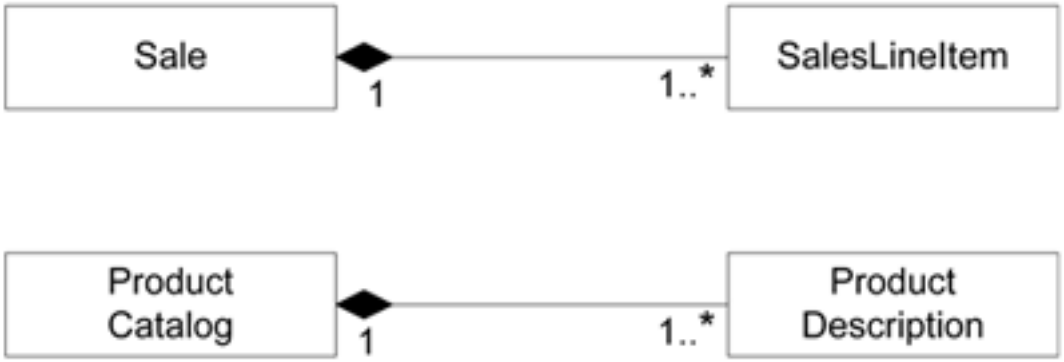
Composition

- An instance of the part belongs to only one composite instance at a time.
- The part always belongs to a composite
- The composite is responsible for the creation and deletion of its parts

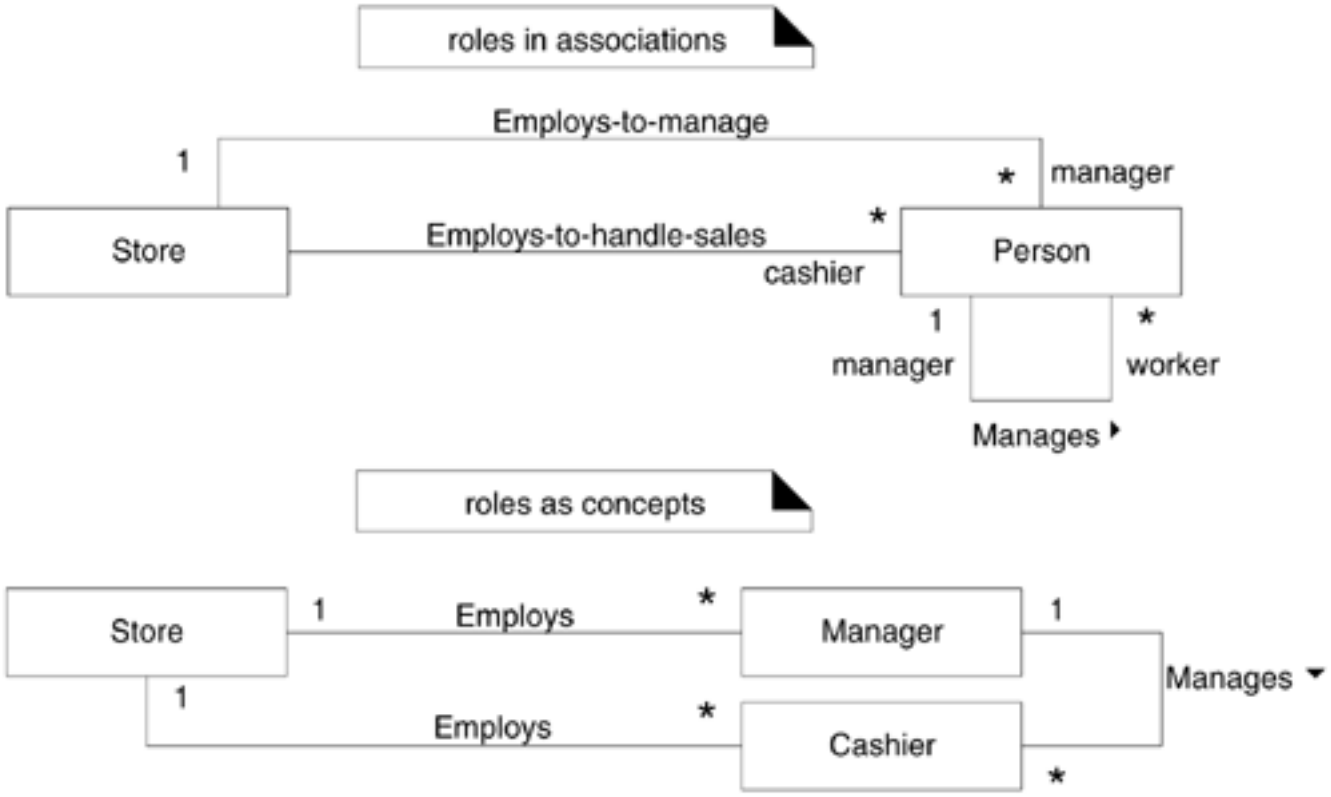
Guideline

- The lifetime of the part is bound with the lifetime of the composite.
- There is an obvious whole-part physical or logical assembly.
- Some properties and operations of the composite propagate to the part

Example

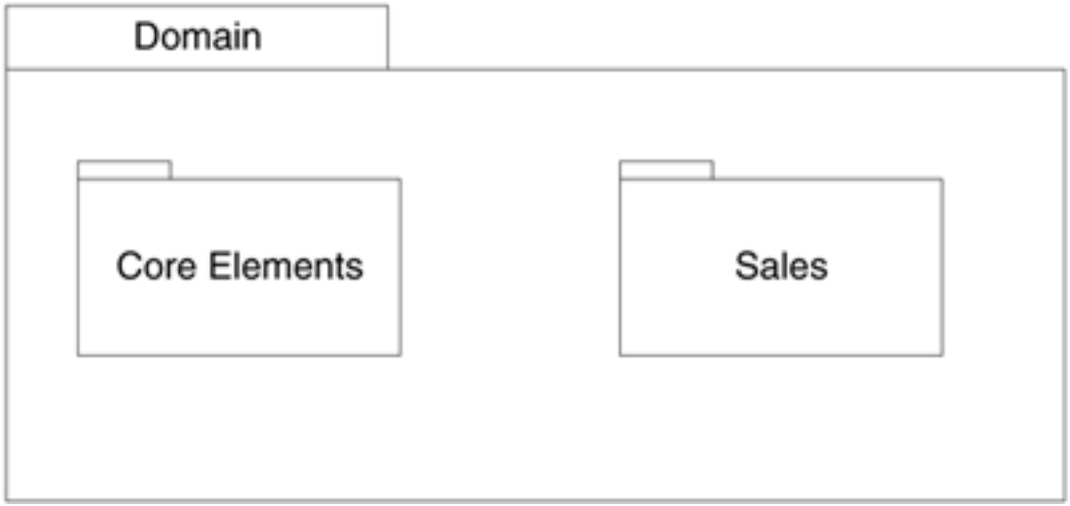

Sale
1
1..*
SalesLineItem
Product Catalog
1
1..*
Product Description

Roles as Concepts vs Roles in Associations



Why Package Diagrams?

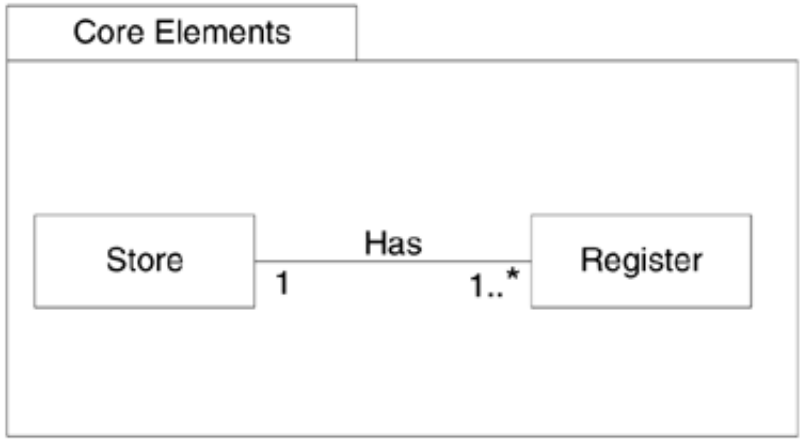
- Used to manage domain models that grow too big.
- Grouping related concepts together helps to understand them.

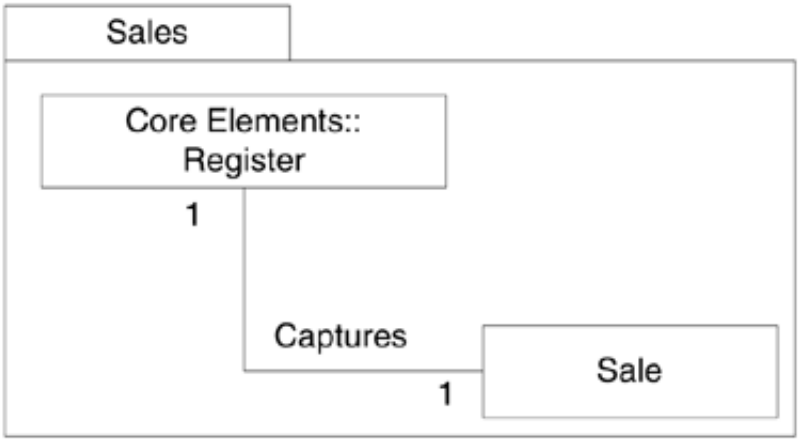


Guidelines: How to Partition the Domain Model

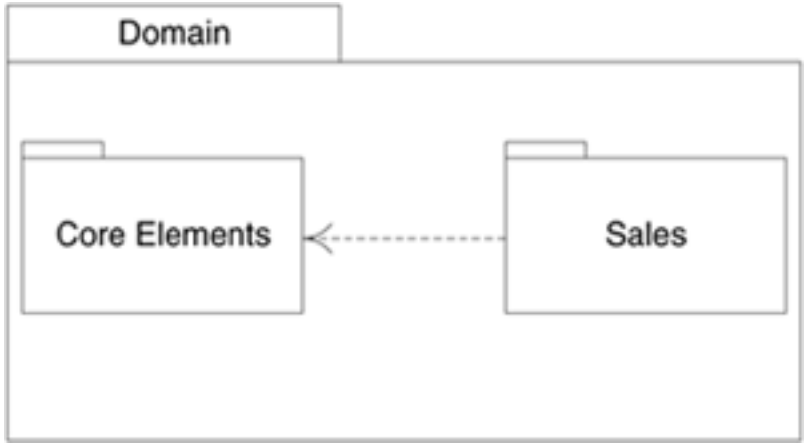
- To partition the domain model into packages, place elements together that
  - are in the same subject area, i.e., closely related by concept or purpose
  - are in a class hierarchy together
  - participate in the same use cases
  - are strongly associated

Ownership and Reference

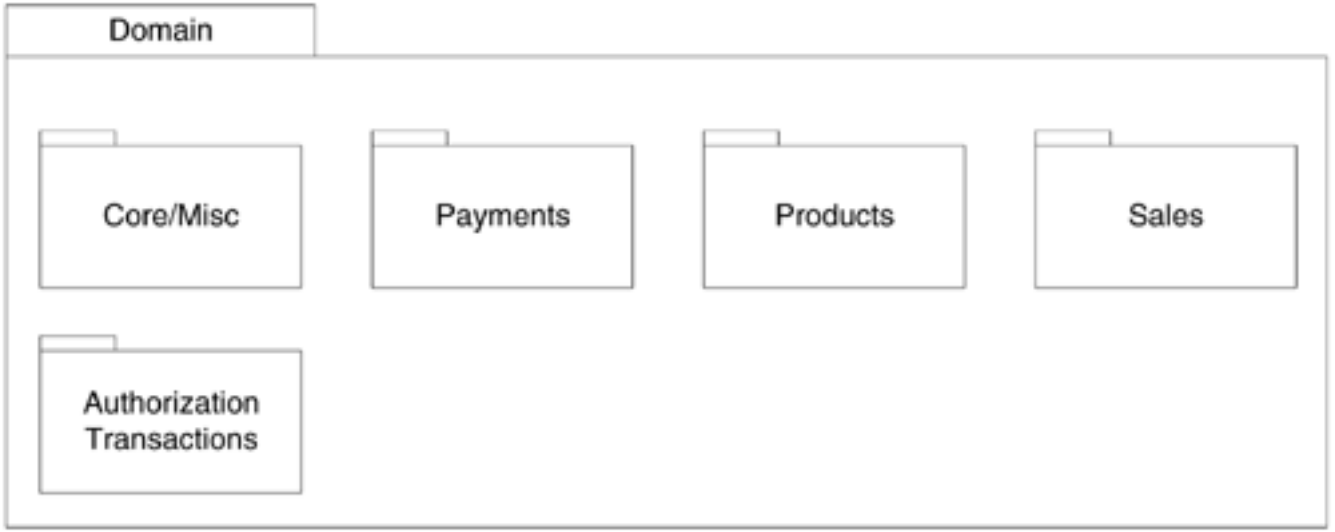




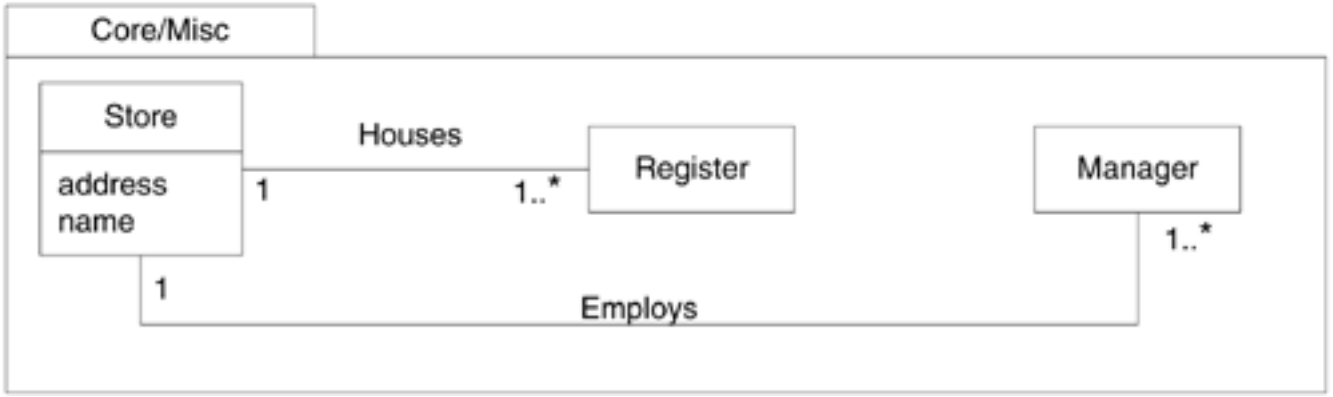
Dependencies



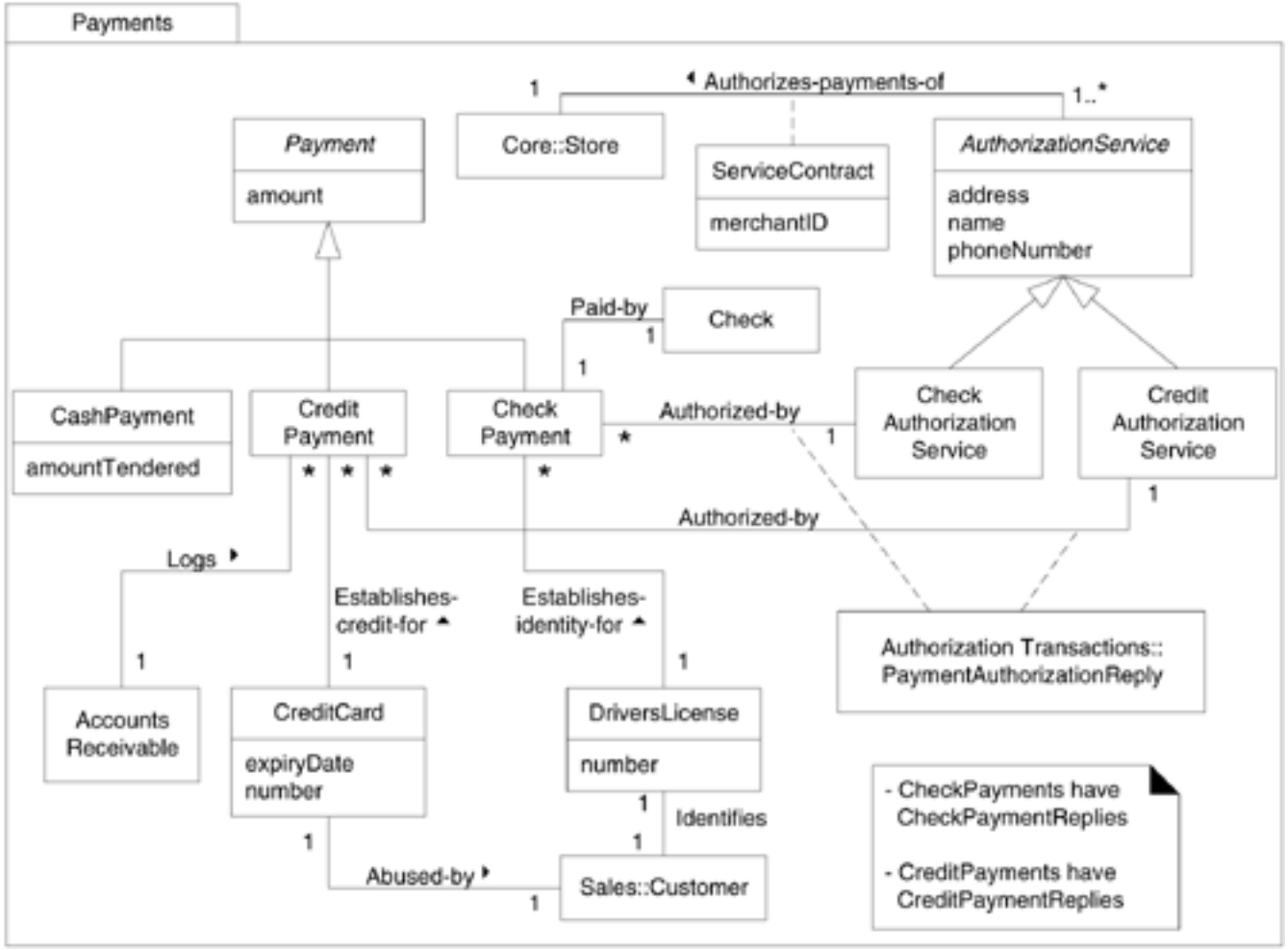
Example


Domain
Core/Misc
Payments
Products
Sales
Authorization
Transactions

Example (2)


Core/Misc
Store
address
name
Houses
1
1..*
Register
Manager
1..*
1
Employs

Example (3)


Payments
Authorizes-payments-of
1
1..*
Payment
amount
Core::Store
ServiceContract
merchantID
AuthorizationService
address
name
phoneNumber
Paid-by
Check
CashPayment
amountTendered
Credit Payment
Check Payment
Authorized-by
Check Authorization Service
Credit Authorization Service
Logs
Establishes-credit-for
Establishes-identity-for
Authorization Transactions:: PaymentAuthorizationReply
Accounts Receivable
CreditCard
expiryDate
number
DriversLicense
number
Identifies
Abused-by
Sales::Customer
- CheckPayments have CheckPaymentReplies
- CreditPayments have CreditPaymentReplies