

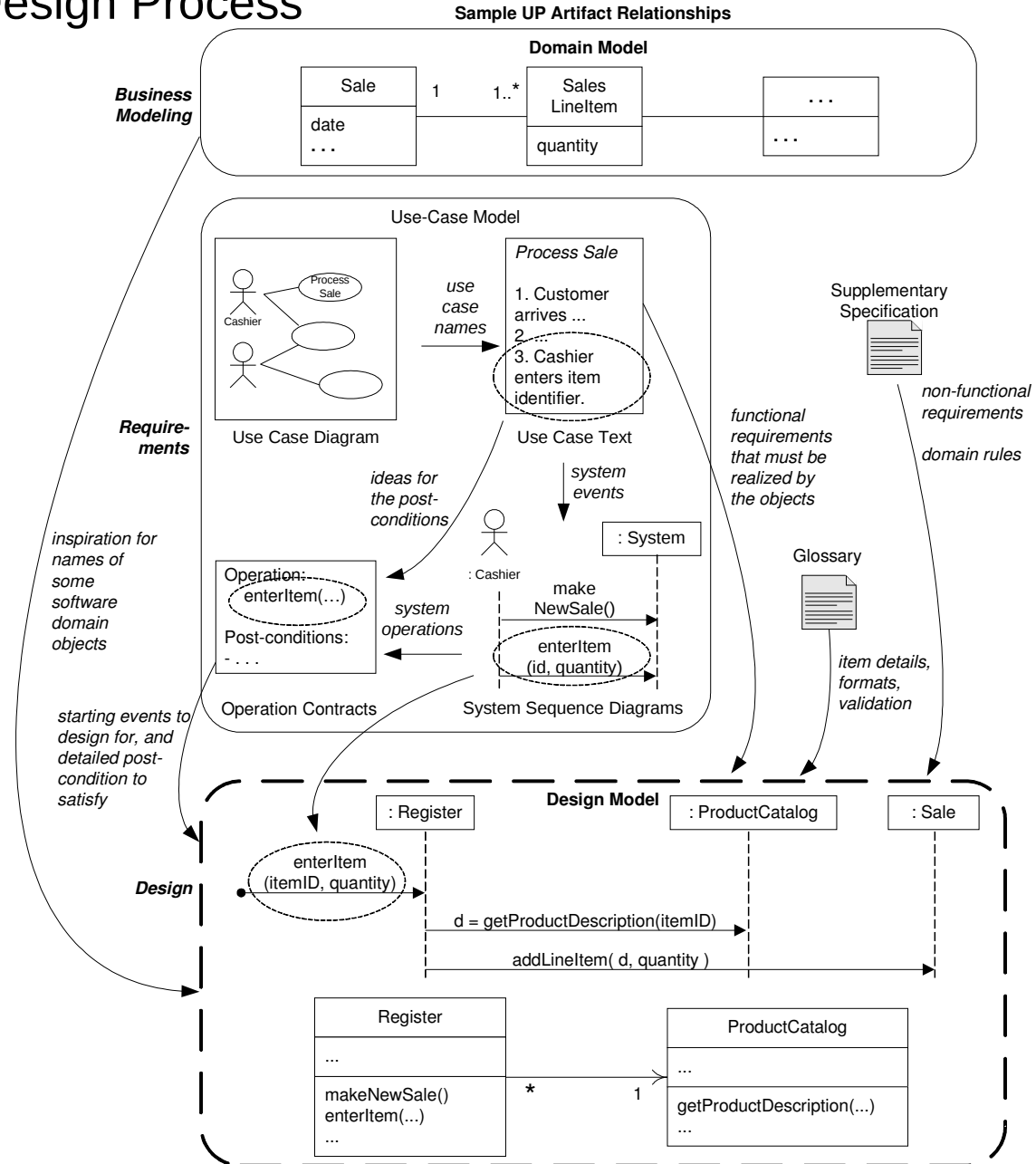
# SOEN 6461 Domain Model

Greg Butler

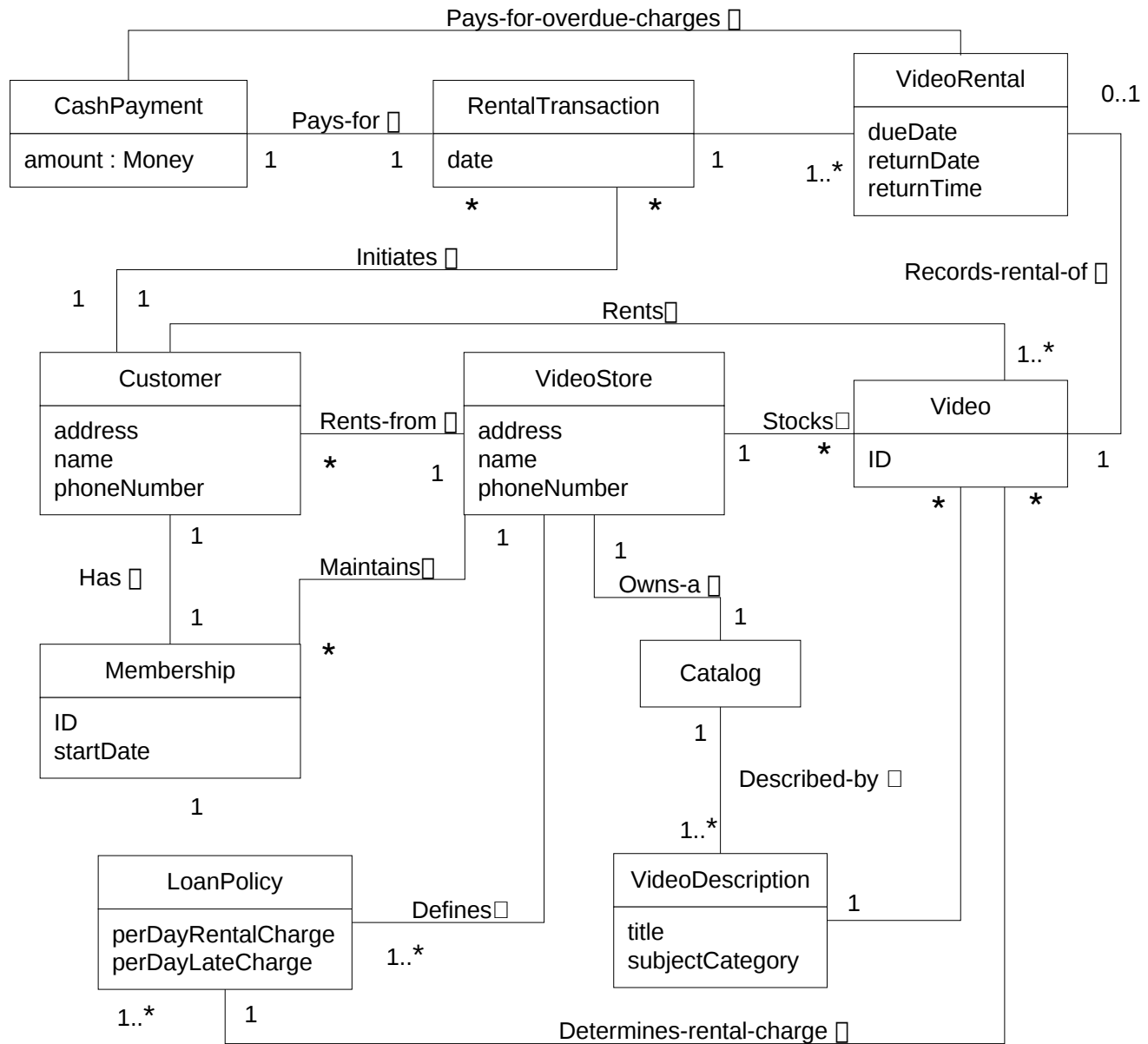
Computer Science and Software Engineering  
Concordia University, Montreal, Canada

Email: [gregb@cs.concordia.ca](mailto:gregb@cs.concordia.ca)

# Larman's Design Process



# Domain Model



# *Domain Model: Visualizing Concepts*

# *Domain Models*

- ◆ A Domain Model illustrates meaningful concepts in a problem domain.
- ◆ It is a representation of real-world things, not software components.
- ◆ It is a set of static structure diagrams; no operations are defined.
- ◆ It may show:
  - concepts
  - associations between concepts
  - attributes of concepts

# *Domain Modelling*

After domain analysis the next step is to create a model of the domain.

The idea is to capture the business and understand it — don't just concentrate on *this* application, study *all* applications in this business first.

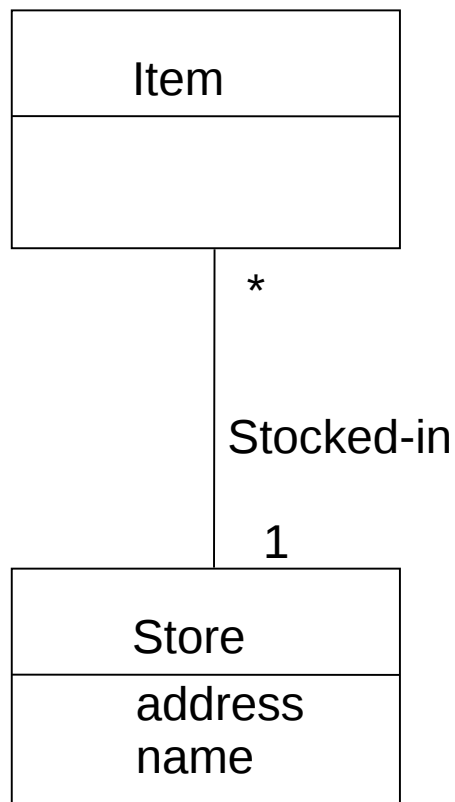
Together, they describe the environment in which the software has to live and work — and the context that will exert pressure on it to change.

In an object-oriented world, we capture the objects, relationships and processes in the business:

- the people and roles in the organization
- the tasks they need to perform
- the interactions between these tasks

This provides good documentation for the business.

# Domain Models



- ◆ A Domain Model is a description of things in the real world.
- ◆ A Domain Model is not a description of the software design.
- ◆ A concept is an idea, thing, or object.

## *Conceptual Classes in the POS Domain*

Store

Register

Sale

Partial Domain Model.

A central distinction between object-oriented and structured analysis:

division by concepts (objects) rather than division by functions.



# *Strategies to Identify Conceptual Classes*

- ◆ Use a conceptual class category list.
  - Make a list of candidate concepts.
- ◆ Use noun phrase identification.
  - Identify noun (and noun phrases) in textual descriptions of the problem domain, and consider them as concepts or attributes.
  - Use Cases are an excellent description to draw for this analysis.

# *Use a Conceptual Class Category List*

<u>Concept Category</u>	<u>Example</u>
physical or tangible objects	Register
specifications, designs, or descriptions of things	ProductDescription
places	Store
transactions	Sale, Payment
transaction line items	SalesLineItem
roles of people	Cashier
containers of other things	Store, Bin

(See pp. 140-141 in Larman 3<sup>rd</sup> ed.)

# *Finding Conceptual Classes with Noun Phrase Identification*

1. This use case begins when a **Customer** arrives at a **cash register** with items to purchase.
2. The **Cashier** starts a new sale.
3. **Cashier** enters an **item identifier**.
- ...

- ◆ Fully addressed Use Cases are an excellent description to draw for this analysis.
- ◆ Some of these noun phrases are candidate concepts; some may be attributes of concepts.
- ◆ A mechanical noun-to-concept mapping is not possible, as words in a natural language are (sometimes) ambiguous.

# *The Need for Specification or Description Conceptual Classes*

- ◆ What's wrong with this picture?

Item
description price serial number itemID

# *The Need for Specification or Description Conceptual Classes*

Item
description price serial number itemID

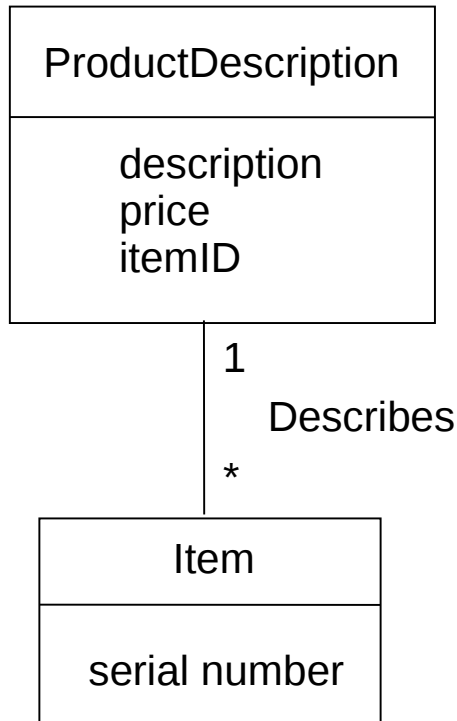
- ◆ What's wrong with this picture?
- ◆ Consider the case where all items are sold, and thus deleted from the computer memory.
- ◆ How much does an item cost?

# *The Need for Specification or Description Conceptual Classes*

Item
description price serial number itemID

- ◆ The memory of the item's price was attached to inventoried instances, which were deleted.
- ◆ Notice also that in this model there is duplicated data (description, price, itemID).

# *The Need for Specification or Description Conceptual Classes*



Add a specification or description concept when:

- Deleting instances of things they describe results in a loss of information that needs to be maintained, due to the incorrect association of information with the deleted thing.
- It reduces redundant or duplicated information.

# *The NextGen POS (partial) Domain Model*

Register

Item

Store

Sale

Sales  
LineItem

Cashier

Customer

Ledger

Cash  
Payment

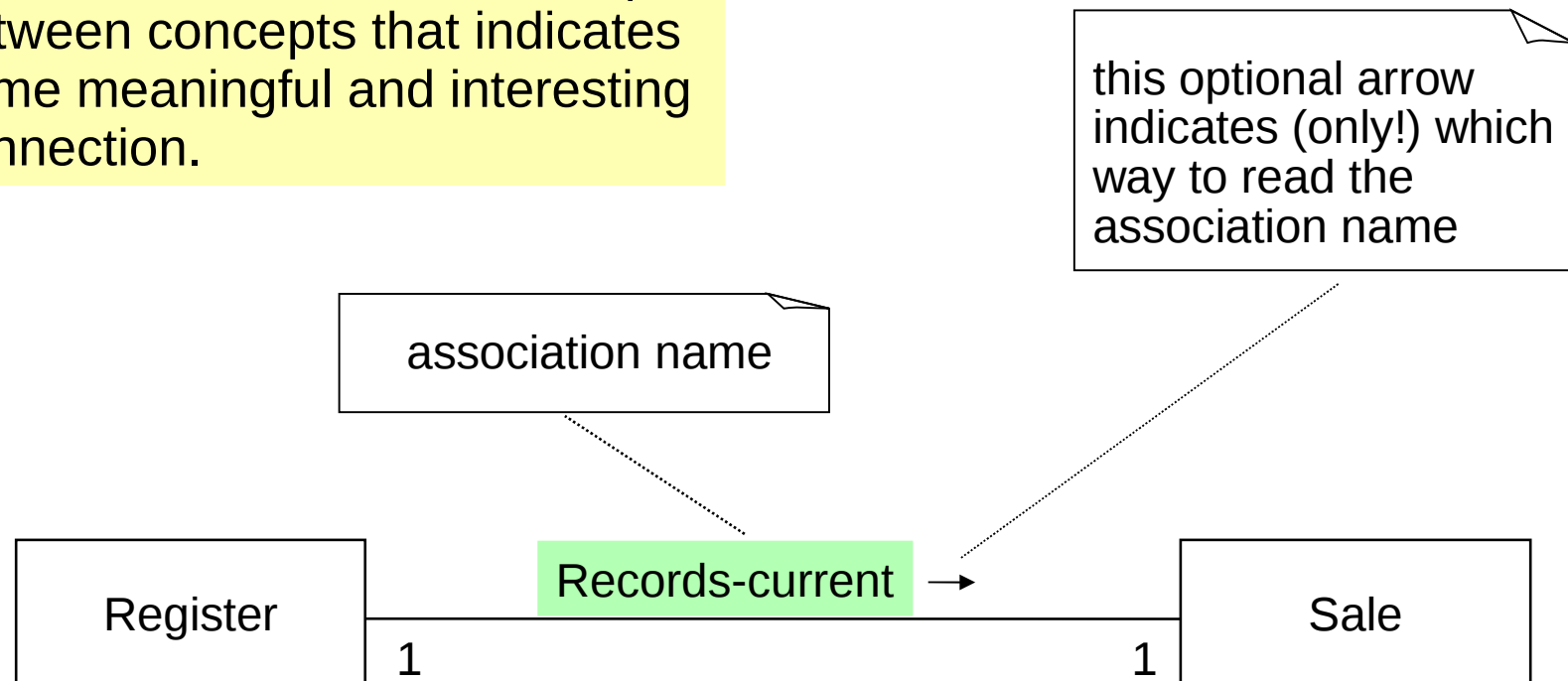
Product  
Catalog

Product  
Description



# Adding Associations

An association is a relationship between concepts that indicates some meaningful and interesting connection.



# *Finding Associations – Common Associations List*

## Category

***A is a physical part of B***

***A is a logical part of B***

***A is physically contained in/on B***

***A is logically contained in B***

A is a description of B

A is a line item of a transaction  
or report B

***A is known/logged/recorded/  
captured in B***

A is a member of B

...

## Examples

Drawer - Register

SalesLineItem - Sale

Register - Store

ItemDescription - Catalog

ItemDescription - Item

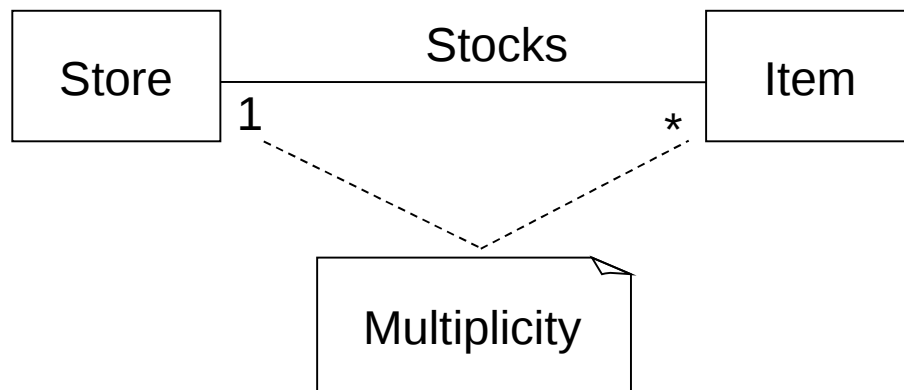
SalesLineItem - Sale

Sale - Register

Cashier - Store

(See pp. 155-156 in Larman 3<sup>rd</sup> ed.)

# Multiplicity

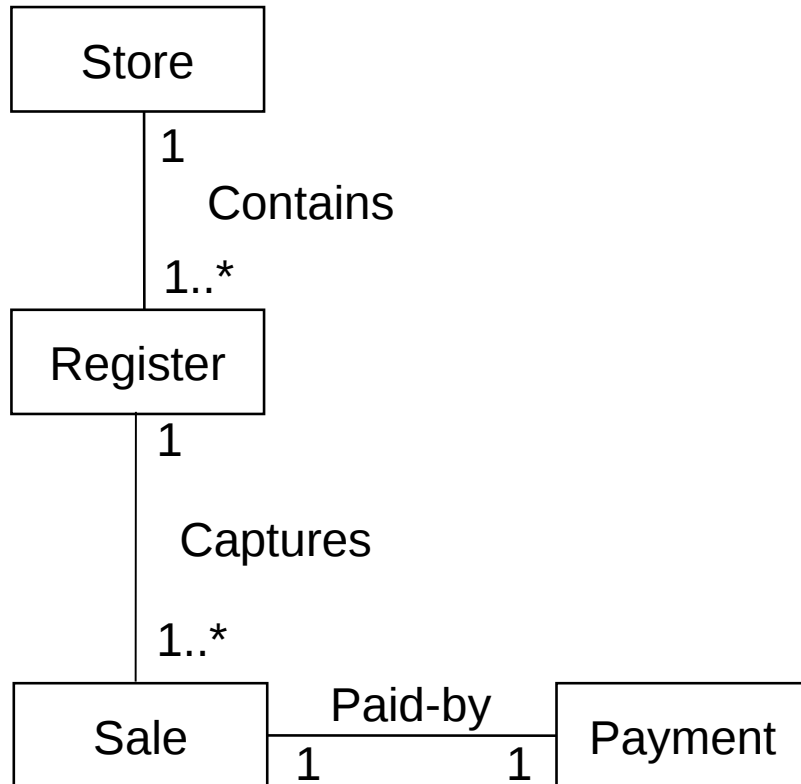


- ◆ Multiplicity defines how many instances of a type A can be associated with one instance of a type B, at a particular moment in time.
- ◆ For example, a single instance of a Store can be associated with “many” (zero or more) Item instances.

# *Multiplicity*

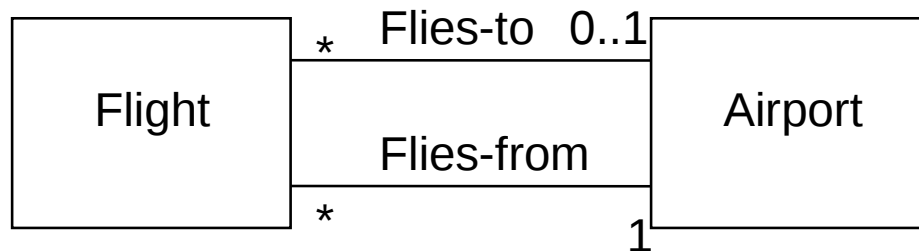
*	Thing	Zero or more; “many”
1..*	Thing	One or more
1..40	Thing	One to forty
5	Thing	Exactly five
3, 5, 8	Thing	Exactly three, five or eight.

# Naming Associations



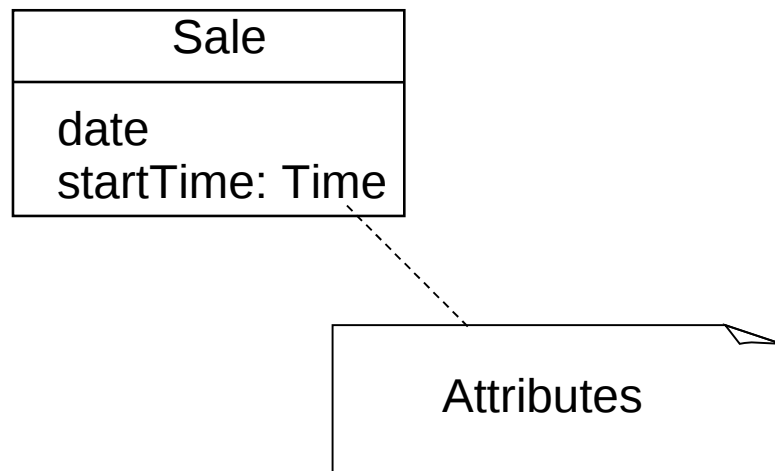
- ◆ Name an association based on a `ClassName - VerbPhrase - ClassName` format. Use specific terms rather than general ones (e.g. "Paid-by" instead of "Uses")
- ◆ Association names should start with a capital letter.
- ◆ A verb phrase should be constructed with hyphens.
- ◆ The default direction to read an association name is left to right, or top to bottom.

## *Multiple Associations Between Two Types*



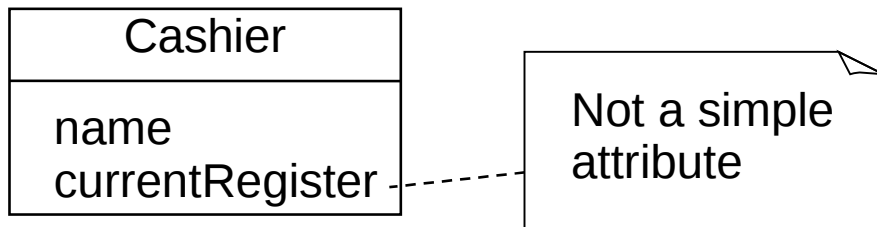
- ◆ It is not uncommon to have multiple associations between two types.
- ◆ In the example, not every flight is guaranteed to land at an airport.

# *Adding Attributes*

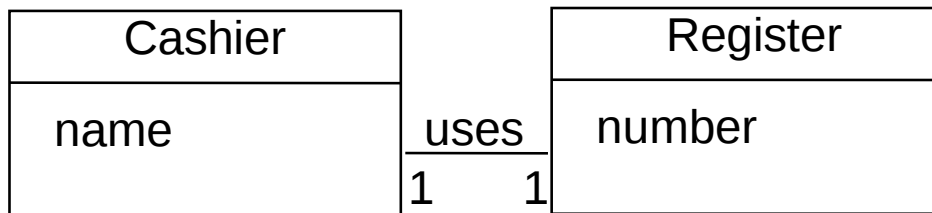


- ◆ An attribute is a logical data value of an object.
- ◆ An attribute should be added when requirements suggest or imply a need to remember information.
- ◆ For example, a sales receipt normally includes a date and time.
- ◆ The Sale concept would need date and time attributes.

# Valid Attribute Types

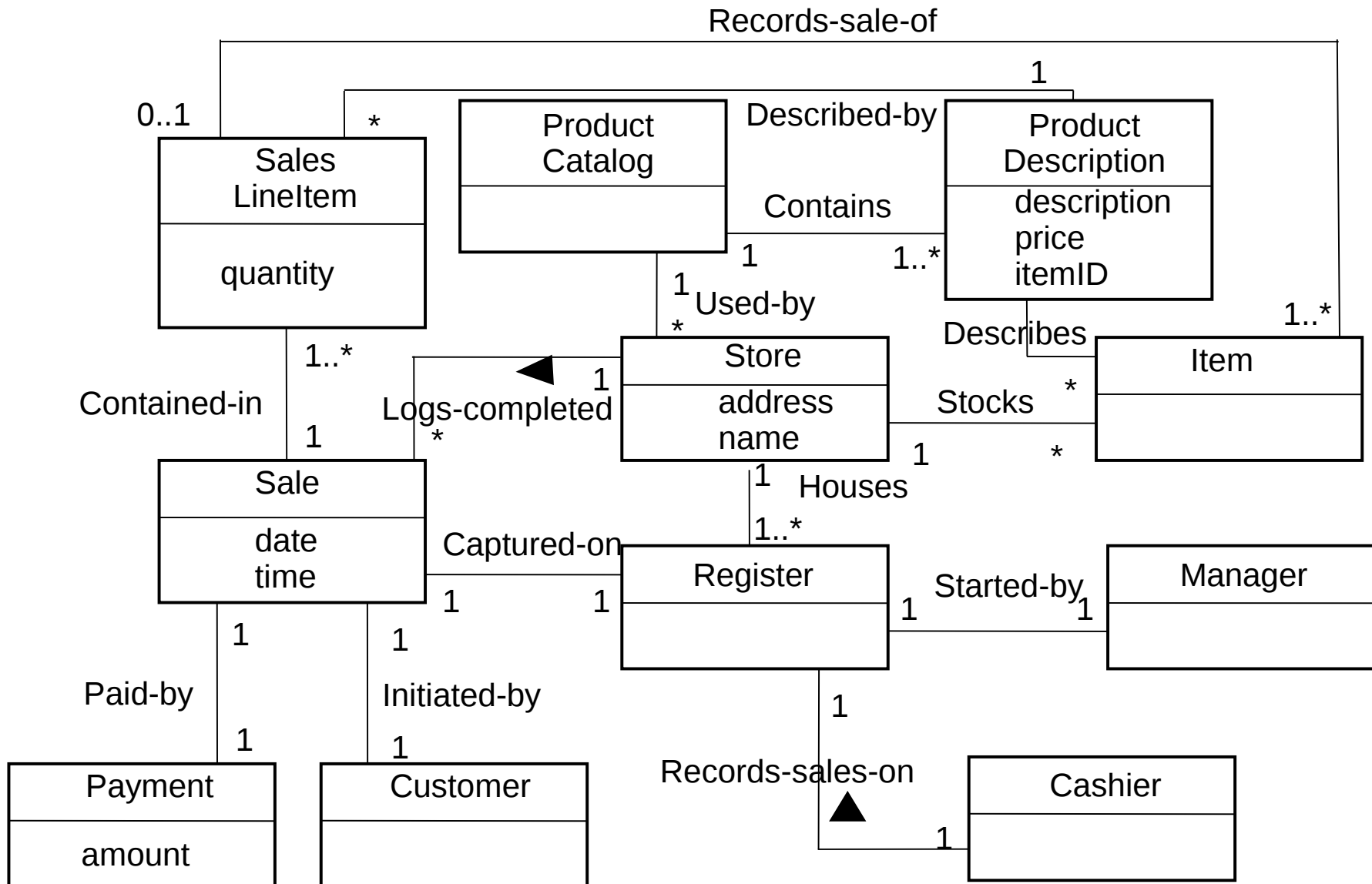


- ◆ Keep attributes simple.
- ◆ The type of an attribute should not normally be a complex domain concept, such as Sale or Airport.
- ◆ Attributes in a Domain Model should preferably be
  - pure data values: Boolean, Date, Number, String, ...
  - simple attributes: color, phone number, zip code, universal product code (UPC), ...





# Domain Model Conclusion



# *Refining the Domain Model*

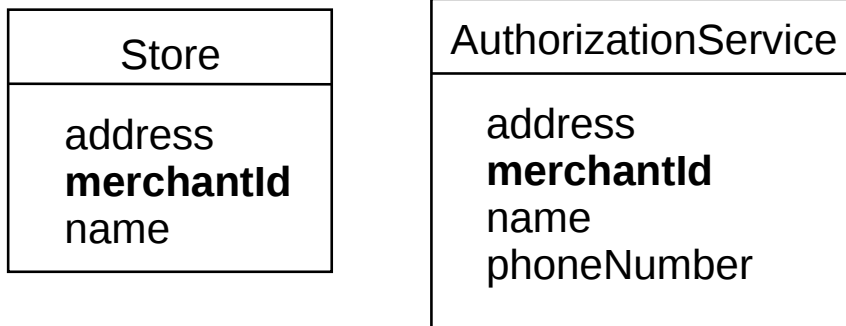
*(a brief return to Week 2, Chapter 16 and Chapter 31)*

# Association Classes

## Example:

- Authorization services assign a merchant ID to each store for identification during communications.
- A payment authorization request from the store to an authorization service requires the inclusion of the merchant ID that identifies the store to the service.
- Consider a store that has a different merchant ID for each service (e.g. ID for Visa is XXX, ID for MC is YYY, etc.).
- Question: Where in the conceptual model should the merchant ID attribute reside?

# Association Classes

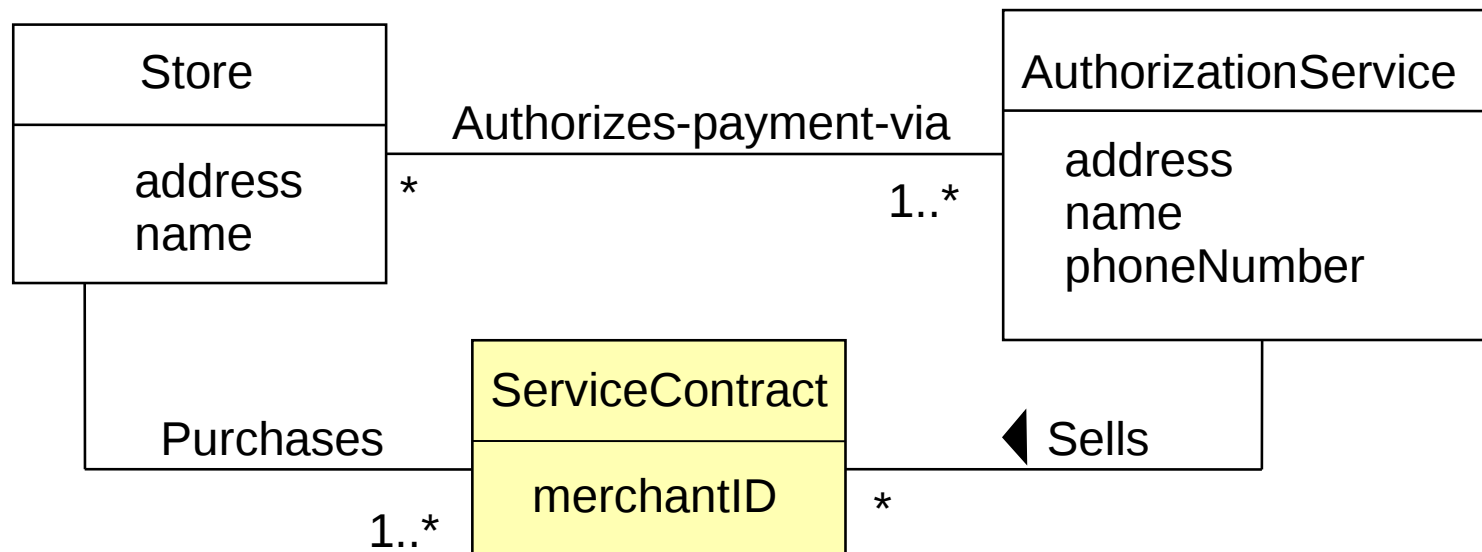


- Where in the conceptual model should the merchant ID attribute reside?
- Placing the merchantID in the Store is incorrect, because a Store may have more than one value for merchantID.
- ...but the same is true with placing it in the AuthorizationService

# Association Classes

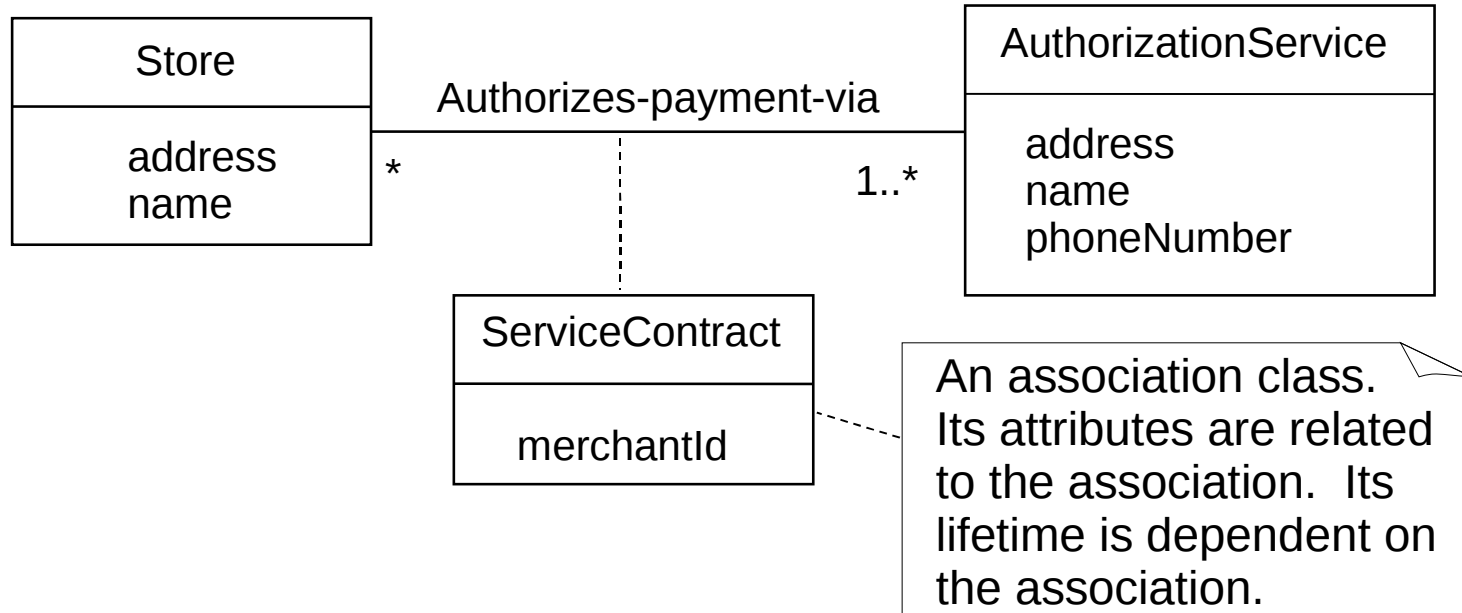
principle:

In a conceptual model, 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 type that is associated with C.



# Association Classes

- ◆ The merchantID is an attribute related to the association between the Store and AuthorizationService; it depends on their relationship.
- ◆ ServiceContract may then be modeled as an association class.



# Association Classes

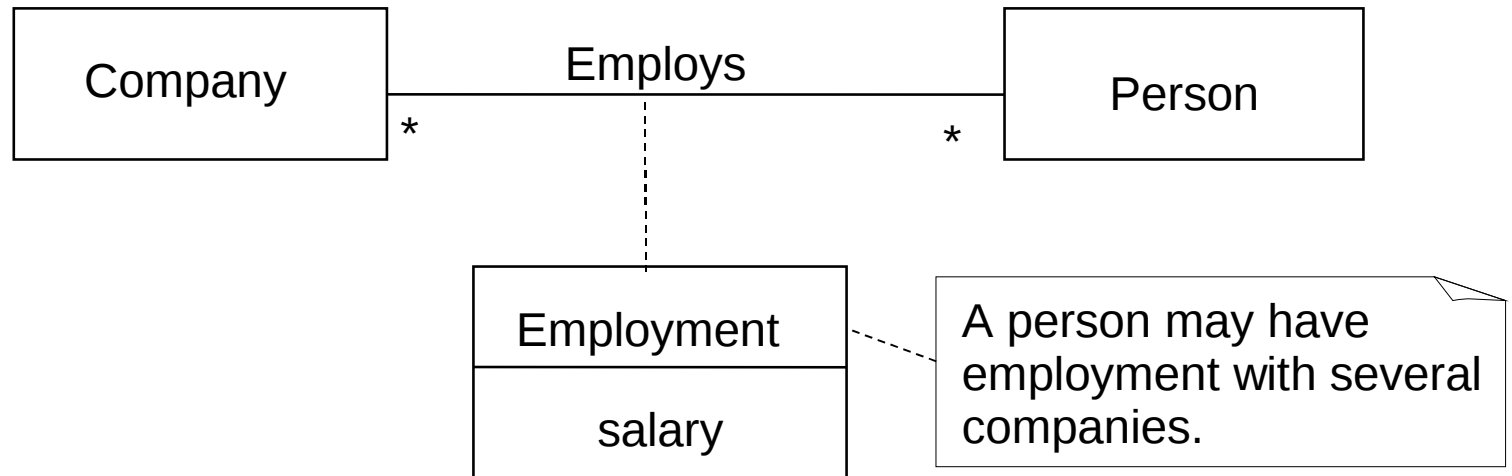
- ◆ Association classes are in some sense similar to description classes, as discussed in week 2.
- ◆ The difference is in what they represent: a description class models the description of an item (e.g. the price of an item in the POS system), while an association class models an association between two or more other classes.

# Guidelines for Association Classes

Consider using an association class when:

- An attribute is related to an association.
- Instances of the association class have a lifetime dependency on the association.
- There is a many-to-many association between two concepts.

The presence of a many-to-many association between two concepts is a clue that a useful associative type may exist.





# Roles as Concepts vs. Roles in Associations

In a conceptual model, a real-world role may be

- modeled as a discrete concept

or

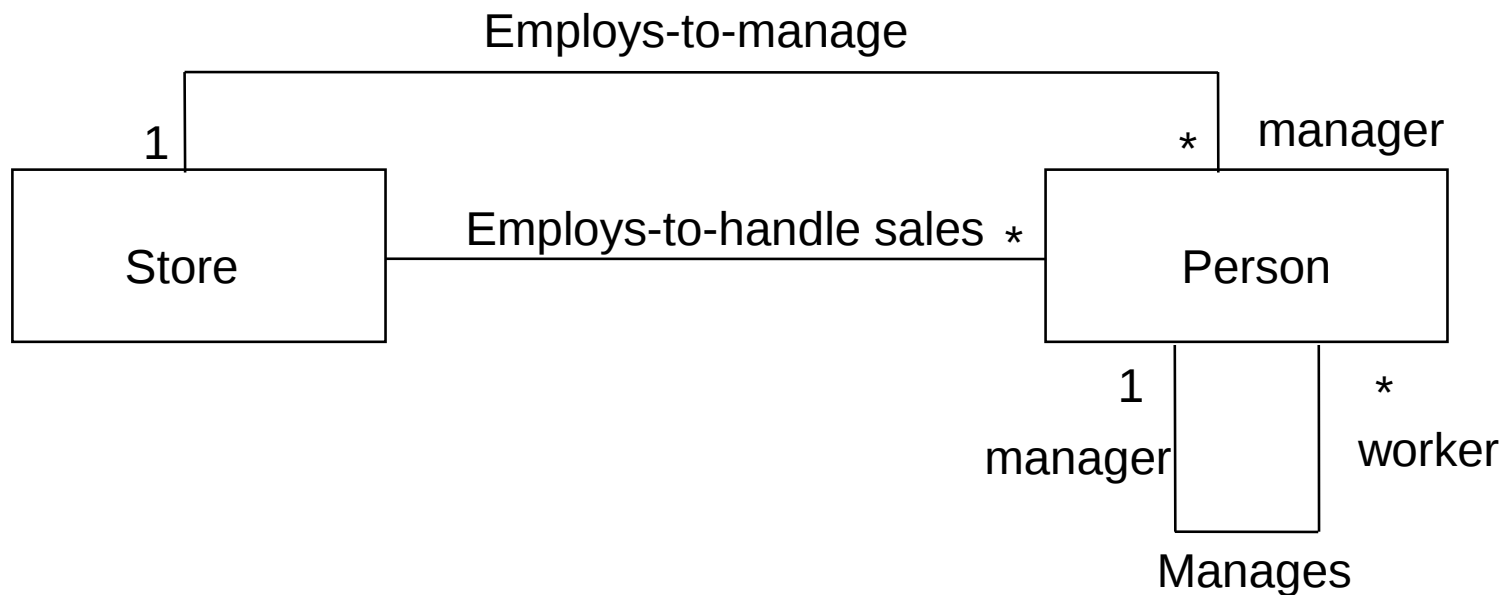
- expressed as a role in an association

Each approach has its own advantages.

(Recall that a "role" in modelling terms is the name given to one end of an association.)

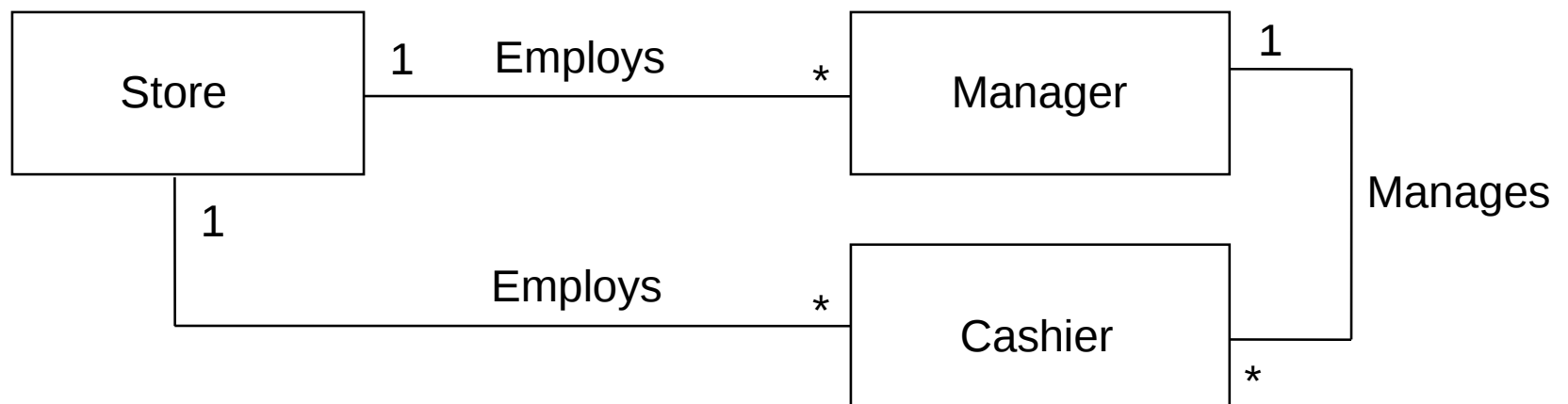
# Roles in Associations

a relatively accurate way to express the notion that the same instance of a person takes on multiple (and dynamically changing) roles in various environments



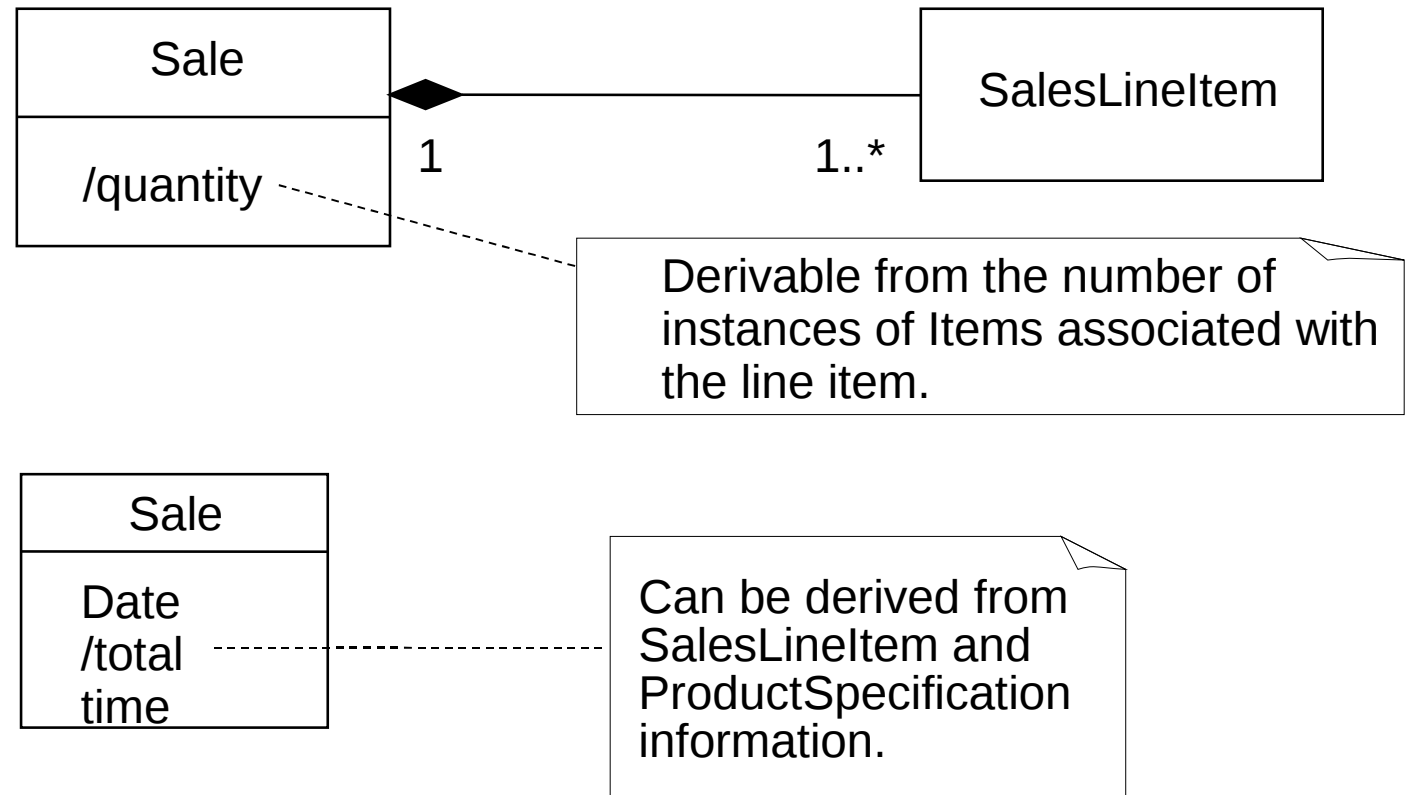
# Roles as Concepts

- ♦ modeling roles as concepts provides ease and flexibility in adding unique attributes, associations, and additional semantics
- ♦ also easier, because Java, C++ etc. don't provide an easy way to dynamically change the class of an existing object



# Derived Elements

- a derived element can be computed from existing attributes
- derived elements should be shown (only) when doing so makes the diagram easier to understand



# Recursive or Reflexive Associations

A concept may have an association to itself; this is known as a recursive or reflective association.

