Domain Model: Visualizing concepts

Objectives

- Identify conceptual classes related to the current iteration requirements
- Create an initial domain model
- Distinguish between correct and incorrect attributes
- Add *specification* conceptual classes, when appropriate
- Compare and contrast conceptual and implementation views

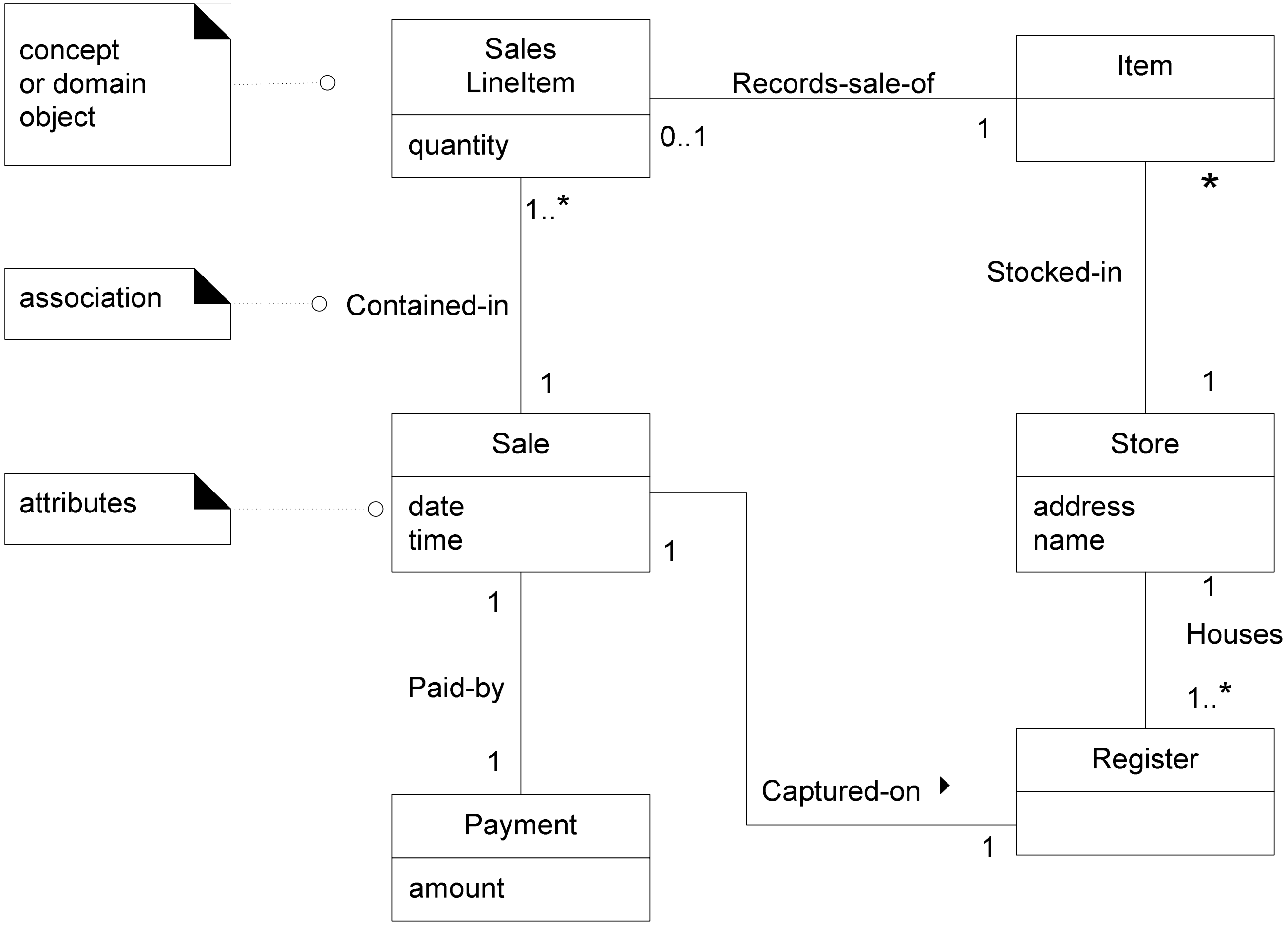
Domain Model

- The Domain Model is the most important artifact to create during object oriented *analysis* – *(*a visual dictionary)
- The domain model is a representation of real-world conceptual classes **NOT** of software components
- The domain model is used as a source of inspiration for designing software objects.

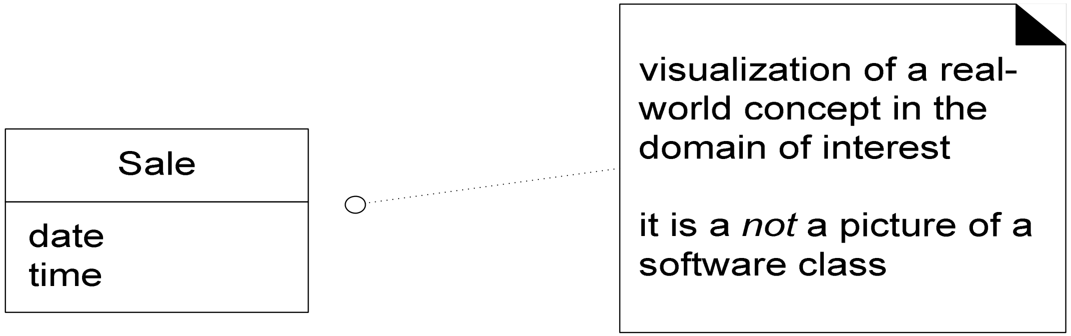
Domain Model

- A domain model is illutrated with a set of *class diagrams* (in which no operations are defined)
- It may show:
  - Domain objects or conceptual classes
  - Associations between conceptual classes
  - Attributes of conceptual classes

Domain Model



Domain Model



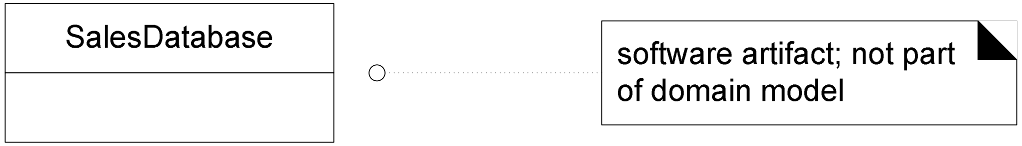
Sale

date
time

visualization of a real-world concept in the domain of interest

it is a *not* a picture of a software class

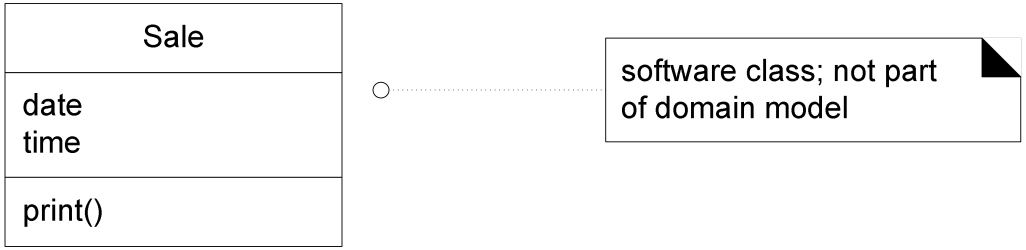
avoid



SalesDatabase

software artifact; not part of domain model

avoid



Sale

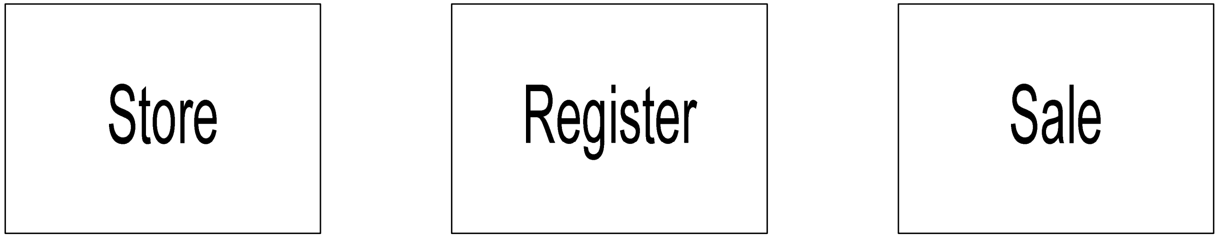
date
time

print()

software class; not part of domain model

Domain Model

- The primary analysis task is to identify different concepts in the problem domain and document the result in a domain model
- Conceptual classes in the Sale Domain:



Store

Register

Sale

Domain Model

guideline

- It is better to overspecify a domain model with lots of fine-grained conceptual classes than to underspecify it !
- A Domain Model is not absolutely correct or wrong, but more or less useful, it is a tool of communication

Domain Model

strategies to identify conceptual classes

- Use a conceptual class category list (p 134-135) → list of candidate conceptual classes
- Identify noun phrases → based upon fully dressed use cases (p 135-136)
- (Use analysis patterns)

Domain Model

Candidate conceptual classes for the Sales Domain:

- Register
- Item
- Store
- Sale
- Payment
- ProductCatalog
- ProductSpecification
- SalesLineItem
- Cashier
- Customer
- Manager
- Receipt???

Domain Model

Candidate conceptual classes for the Sales Domain:

- Receipt???
  - – Report of a sale – duplicates information found elsewhere → exclude receipt
  - – Special role in terms of business rules (the right to return bought items) → include it (since returns are not considered in this iteration, receipt is excluded)

Domain Model

Modeling guidelines

1. List the candidate conceptual classes using CCC-List and noun phrase identification technique related to the current requirements under consideration
2. Draw them in a domain model
3. Add the associations necessary to record relationships for which there is a need to preserve some memory
4. Add the attributes necessary to fulfill the information requirements

Domain Model

’the mapmaker’

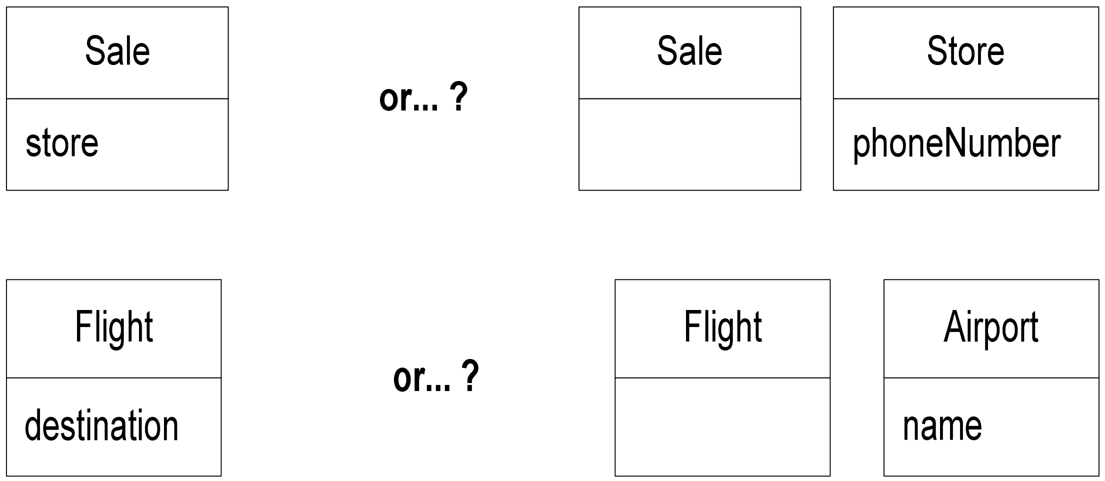
- Use the existing names of the territory
- Exclude irrelevant features
- Do not add things that are not there

Use the Domain Vocabulaty strategy

Domain Model

common mistakes

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



Sale

store

or... ?

Sale

Store

phoneNumber

Flight

destination

or... ?

Flight

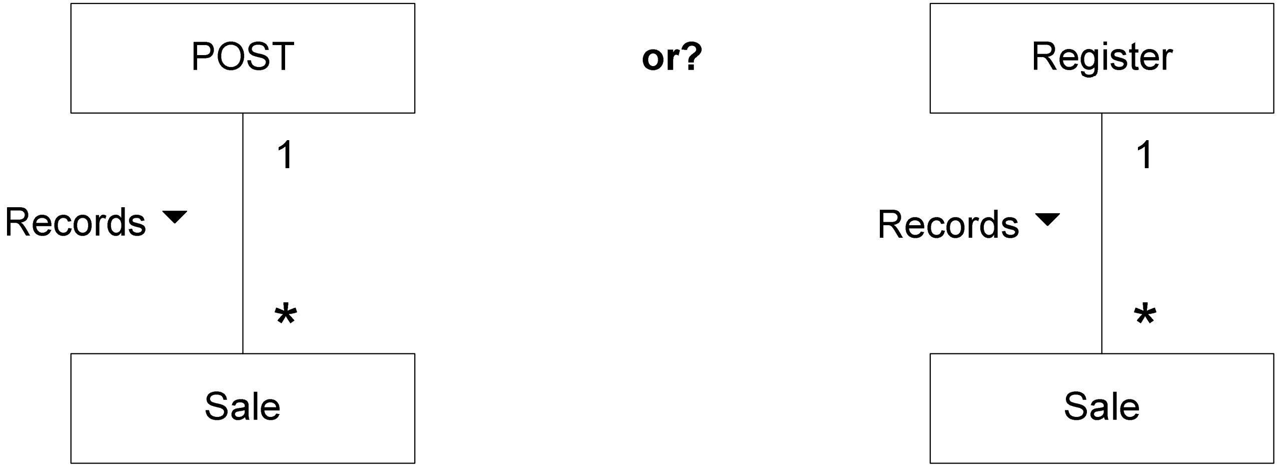
Airport

name

Domain Model

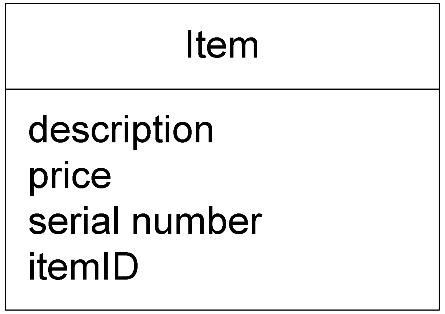
similar conceptual classes

similar concepts with different names



Domain Model

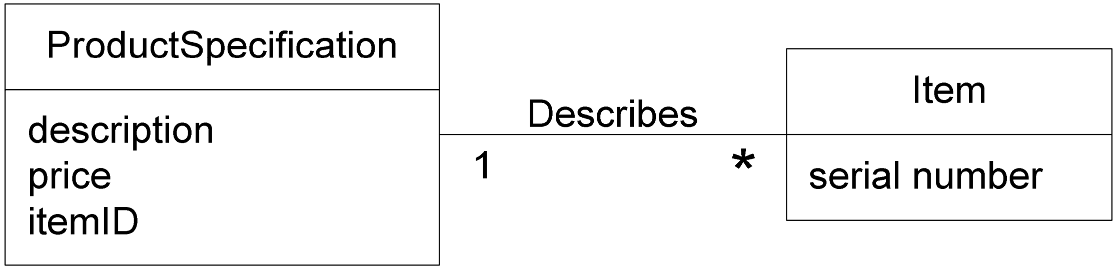
’specification conceptual classes’



Item

description
price
serial number
itemID

Worse



ProductSpecification

description
price
itemID

Describes

1

*

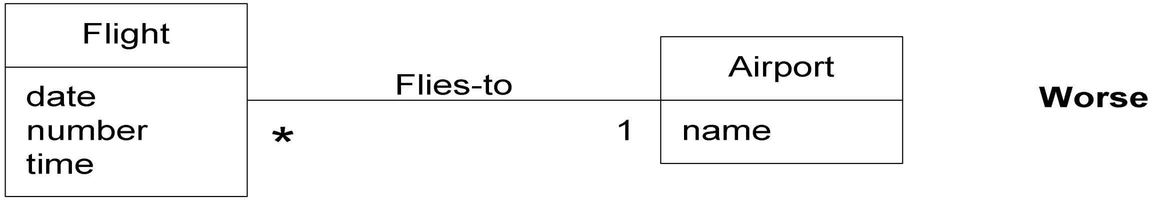
Item

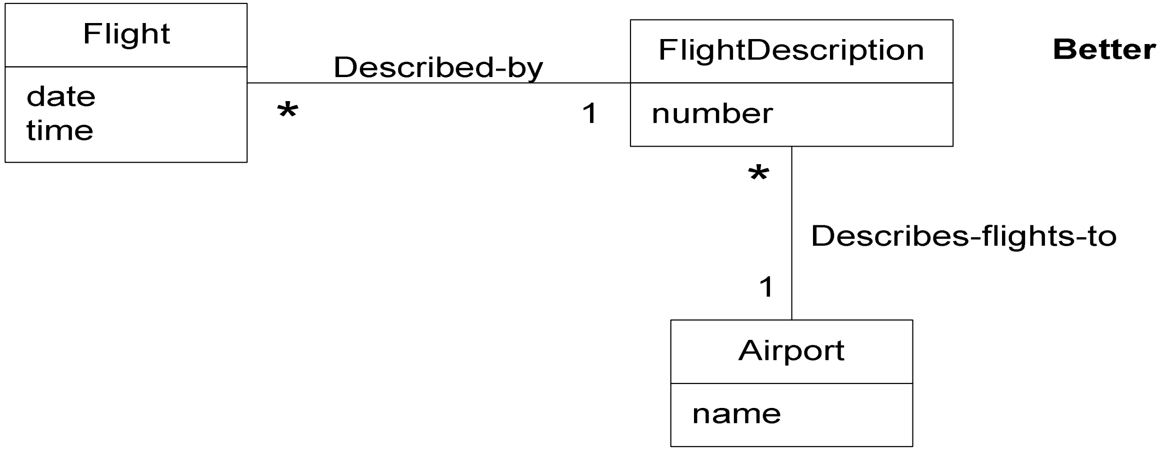
serial number

Better

Domain Model

’specification conceptual classes’

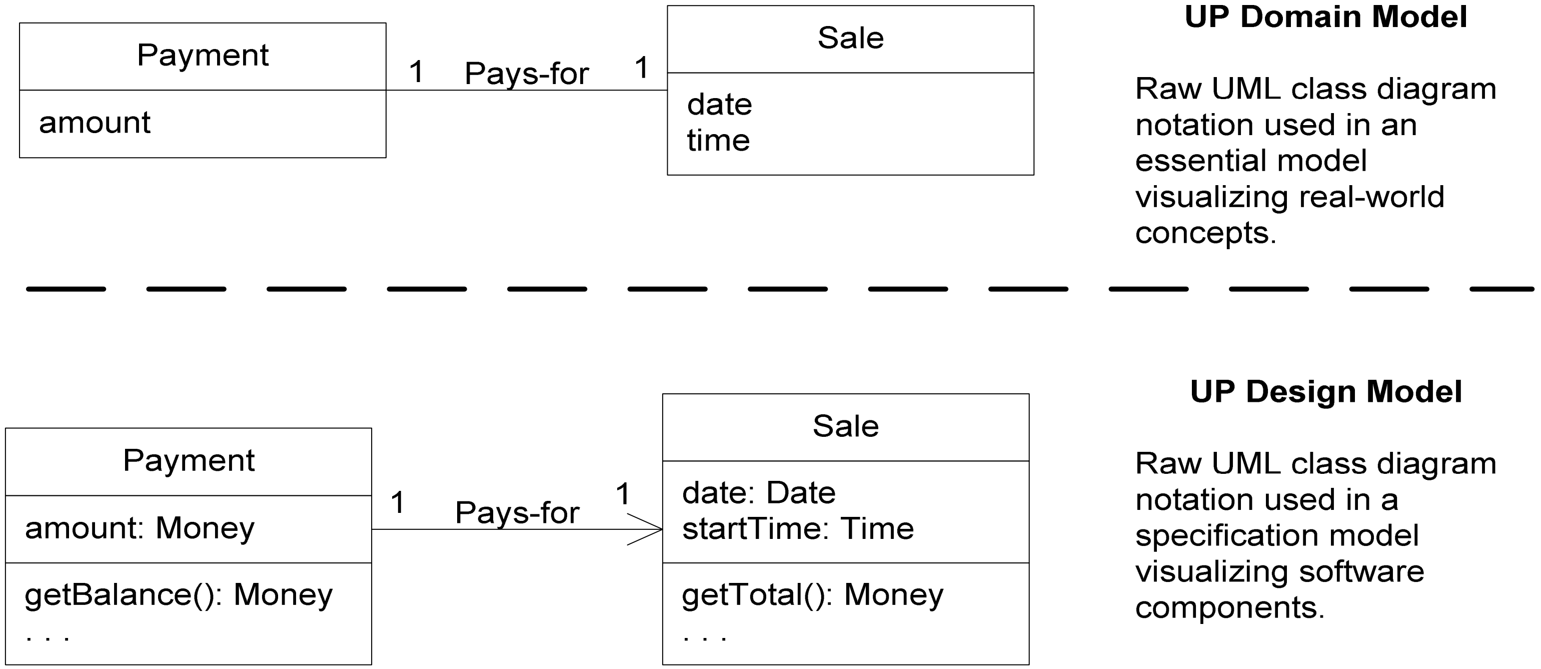




UML Notation

multiple perspectives

- Essential or conceptual perspective
- Specification perspective
- Implementation perspective



UP Domain Model

Raw UML class diagram notation used in an essential model visualizing real-world concepts.

UP Design Model

Raw UML class diagram notation used in a specification model visualizing software components.

UML Notation

terminology

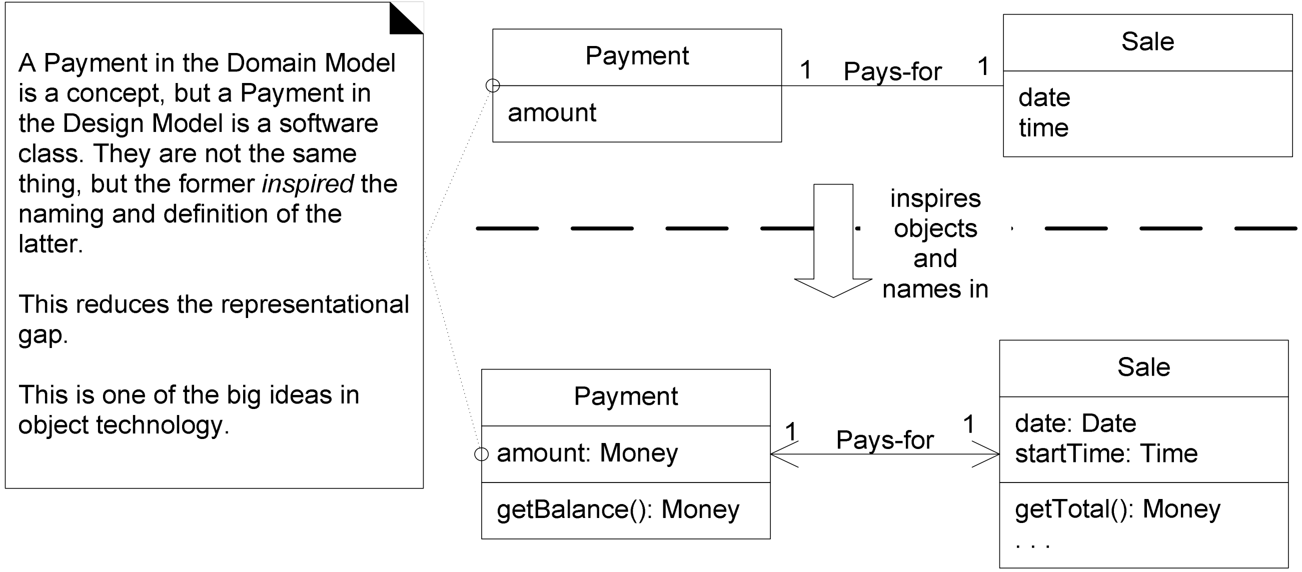
- **Conceptual class** – real world thing/concept
- **Software class** – a class representing a specification or implementation perspective of a software component
- **Design class** – a member of the UP Design Model – synonym for sw class
- **Implementation class** – a class implemented in an objectoriented language
- **Class** – the general term representing either a real world thing or a software thing

Domain Model

representational gap

UP Domain Model

Stakeholder's view of the noteworthy concepts in the domain.



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.

Payment

amount

1 Pays-for 1

Sale

date
time

inspires objects and names in

Payment

amount: Money

getBalance(): Money

1 Pays-for 1

Sale

date: Date
startTime: Time

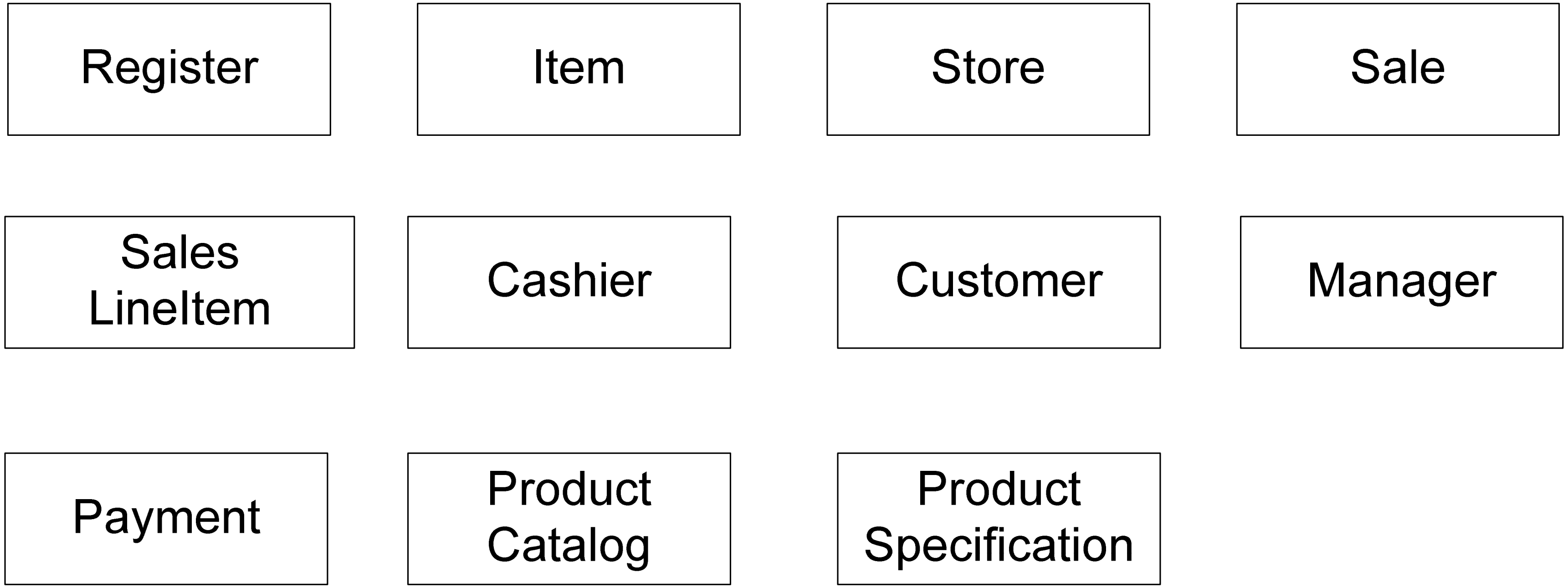
getTotal(): Money
. . .

UP Design Model

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.

The NextGen POS Domain Model



Register

Item

Store

Sale

Sales LineItem

Cashier

Customer

Manager

Payment

Product Catalog

Product Specification

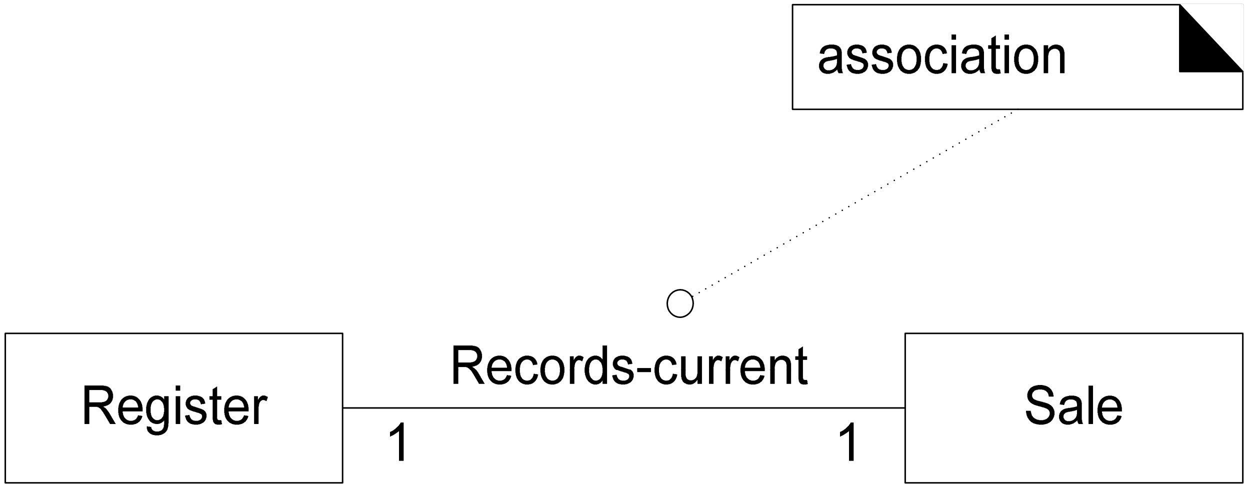
Objectives

- Identify associations for a Domain Model
- Distinguish between need-to-know and comprehension-only associations

Domain Model

associations

- An ***association*** is a relationship between types (or more specifically, instances of those types) that indicates some meaningful and interesting connection



association

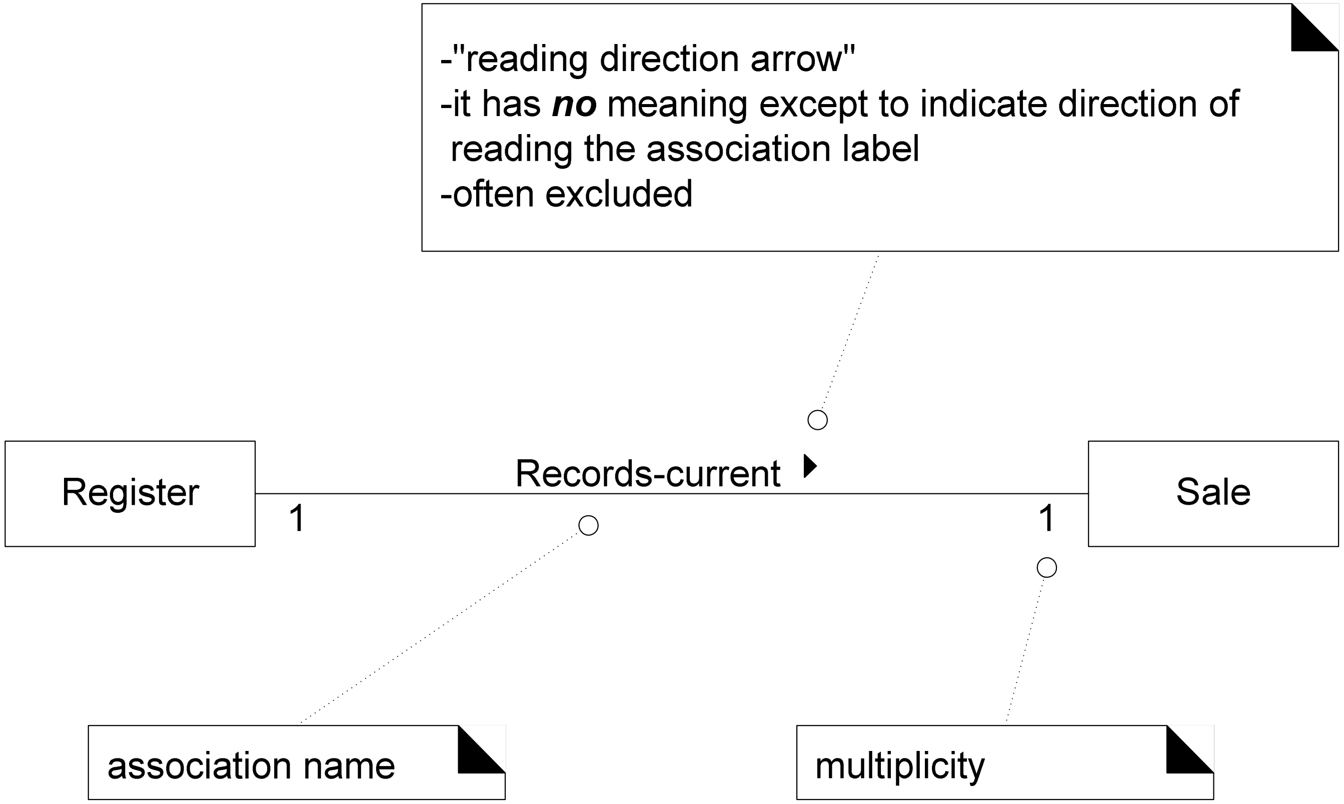
Register

Records-current

Sale

Domain Model

UML notation for associations



-"reading direction arrow"
-it has ***no*** meaning except to indicate direction of
 reading the association label
-often excluded

association name

multiplicity

Domain Model

Finding associations

- Common Associations List (p 156-157)
- High-Priority Associations:
  - A is a physical part of B
  - A is physically or logically contained in B
  - A is recorded in B

Domain Model

Association guidelines

- Focus on those associations for which knowledge of the relationship needs to be preserved for some duration (”need to know” associations)
- It is more important to identify conceptual classes than to identify associations
- Too many associations tend to confuse the domain model rather than illuminate it. Their discovery can be time-consuming, with marginal effect
- Avoid showing redundant or derivable associations

Domain Model

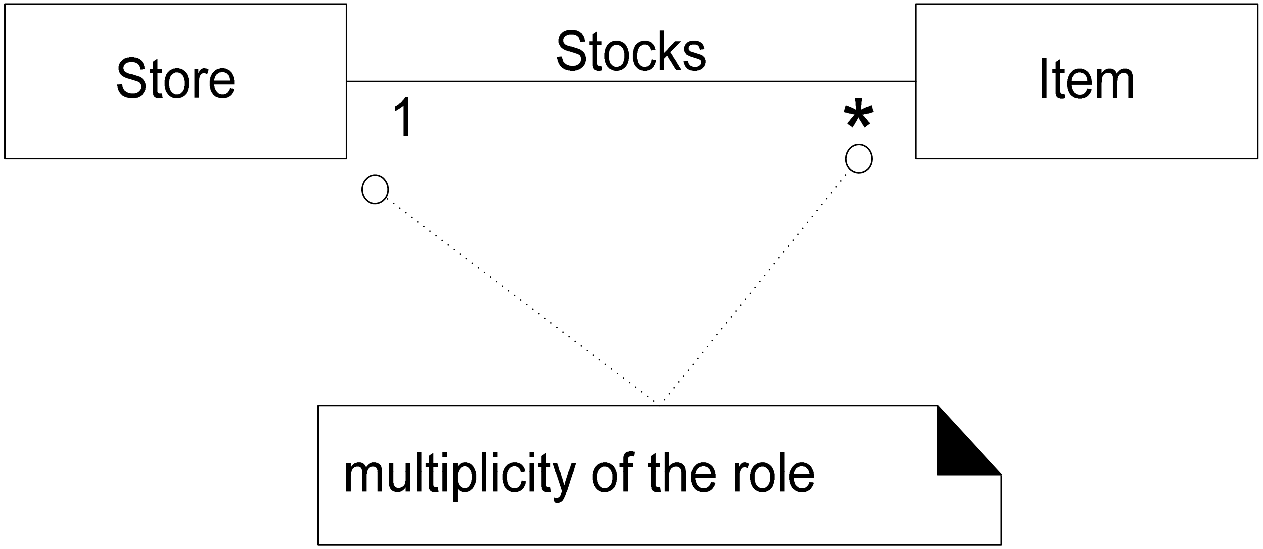
associations - roles

- Each end of an association is called a *role*
- Roles may optionally have:
  - Name
  - Multiplicity expression
  - Navigability (later)

Domain Model

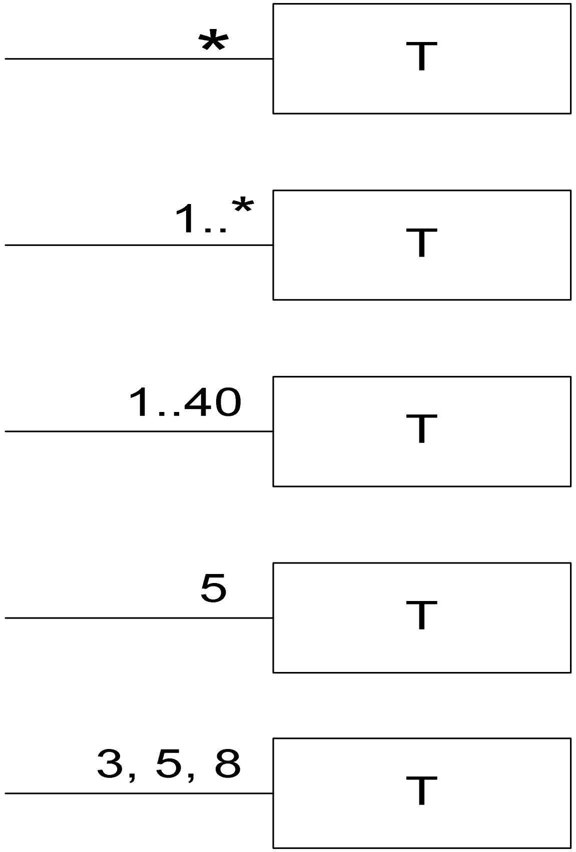
associations - multiplicity

- Multiplicity defines how many instances of a class A can be associated with one instance of class B



Domain Model

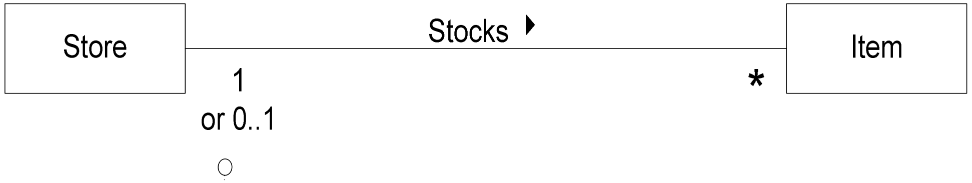
associations - multiplicity



*	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

Domain Model

associations - multiplicity



Multiplicity should "1" or "0..1"?

The answer depends on our interest in using the model. Typically and practically, the muliplicity communicates a domain constraint that we care about being able to check in software, if this relationship was implemented or reflected in software objects or a database. For example, a particular item may become sold or discarded, and thus no longer stocked in the store. From this viewpoint, "0..1" is logical, but ...

Do we care about that viewpoint? If this relationship was implemented in software, we would probably want to ensure that an *Item* software instance would always be related to 1 particular *Store* instance, otherwise it indicates a fault or corruption in the software elements or data.

This partial domain model does not represent software objects, but the multiplicities record constraints whose practical value is usually related to our interest in building software or databases (that reflect our real-world domain) with validity checks. From this viewpoint, "1" may be the desired value.

Domain Model

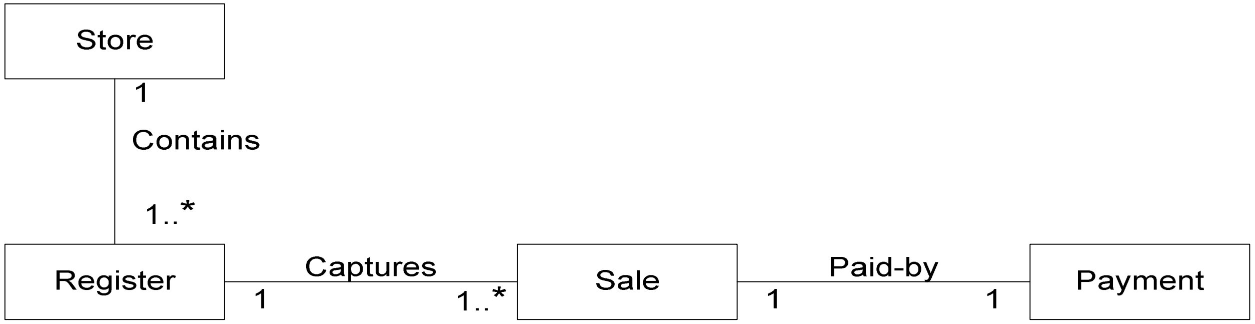
Naming associations

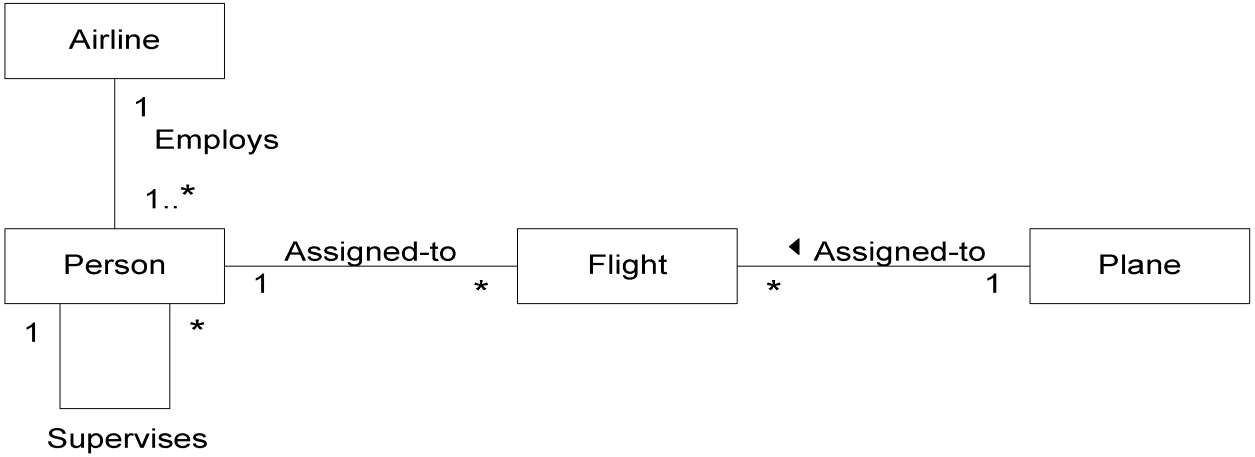
- Name an association based on **TypeName-VerbPhrase-TypeName** format where the verb-phrase creates a sequence that is readable and meaningful in the model context

- Paid-by or PaidBy

Domain Model

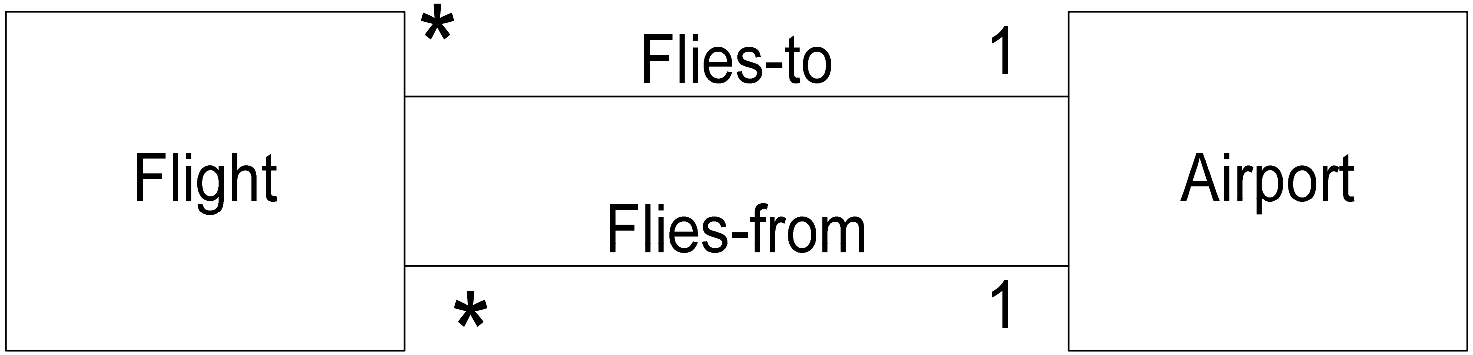
Naming associations





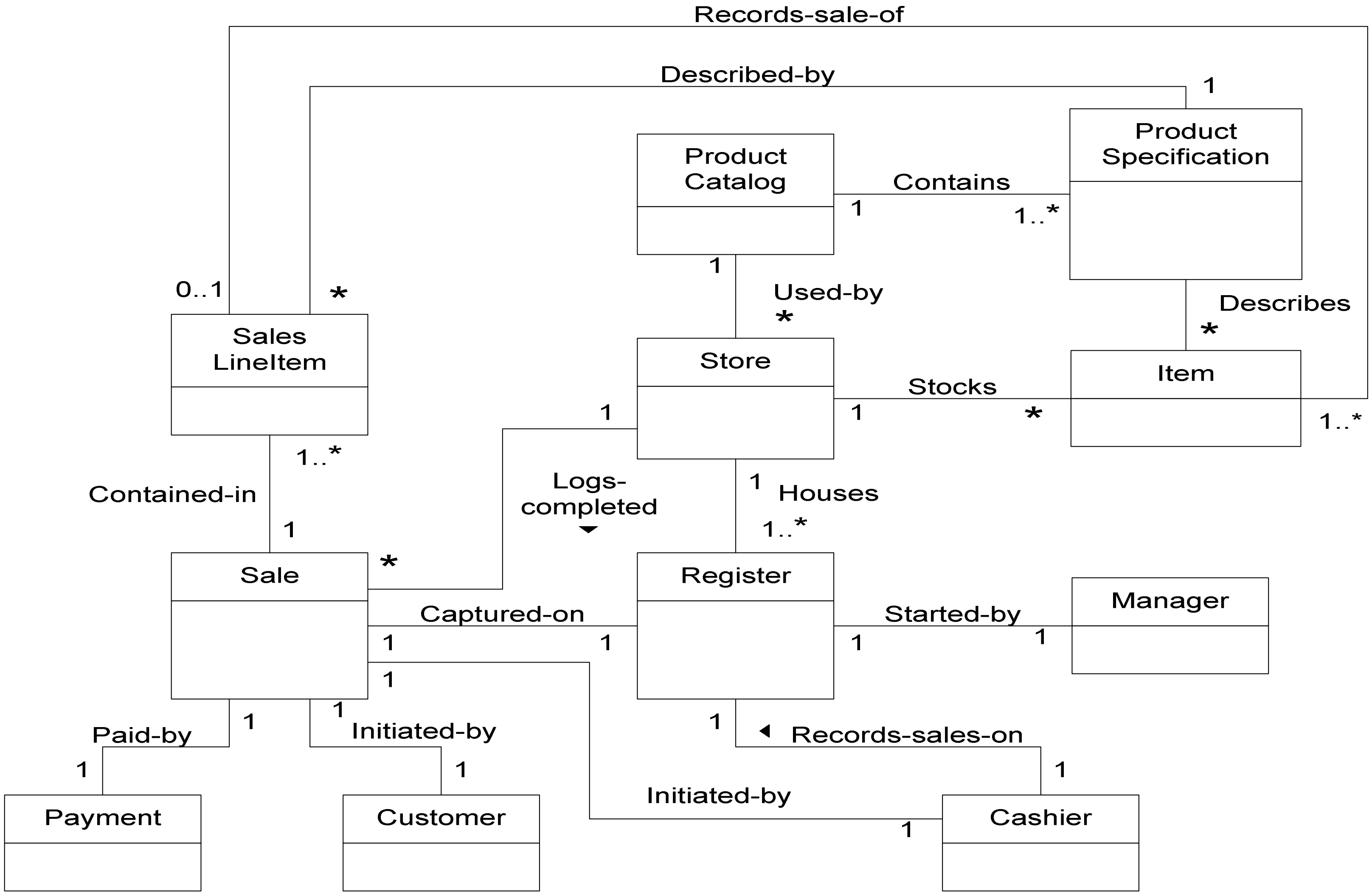
Domain Model

Multiple associations



NextGen POS Domain Model

Associations



Domain Model

Need-to-know vs. comprehension

- Emphasize need-to-know associations, but add choise comprehension-only associations to enrich critical understanding of the domain

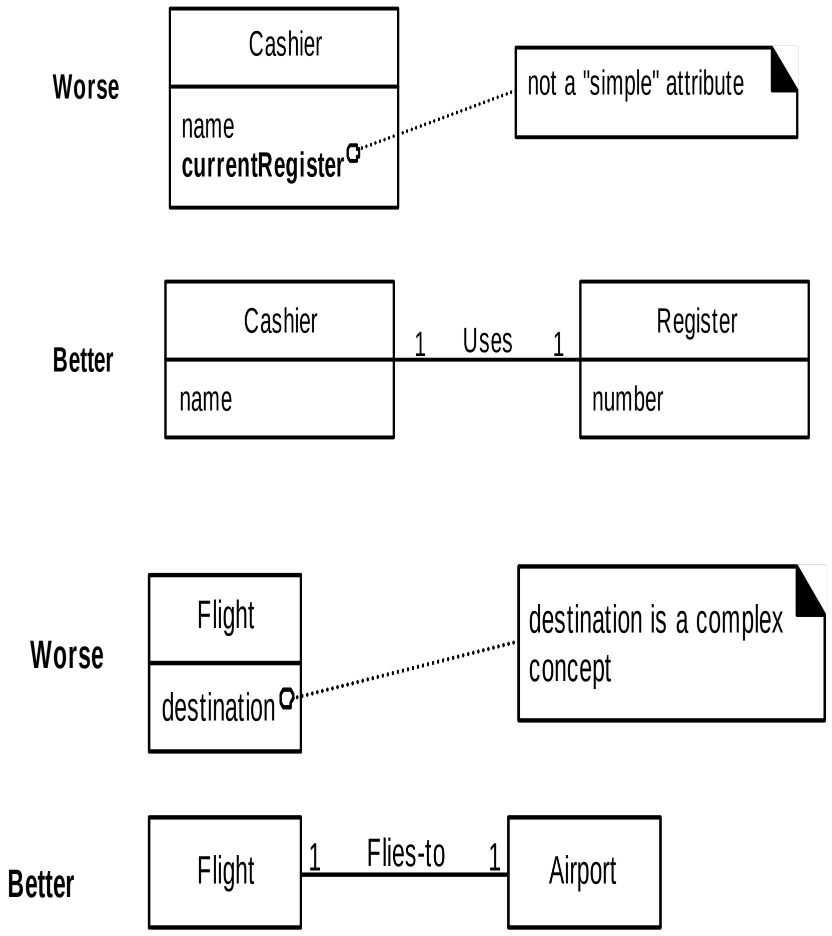
Attributes

- Attribute = a logical data value of an object.

- Include the following attributes in a domain model :
  - Those for which the requirements (ex. Use cases) suggest or imply a need to remember information.
  - Ex. A receipt normally has a date and time.

Valid attribute types

- Keep it simple
  - Prefer simple attributes or data types
    - Boolean, Date, Number, String….
- Relate conceptual classes with an association
  - not with an attribute



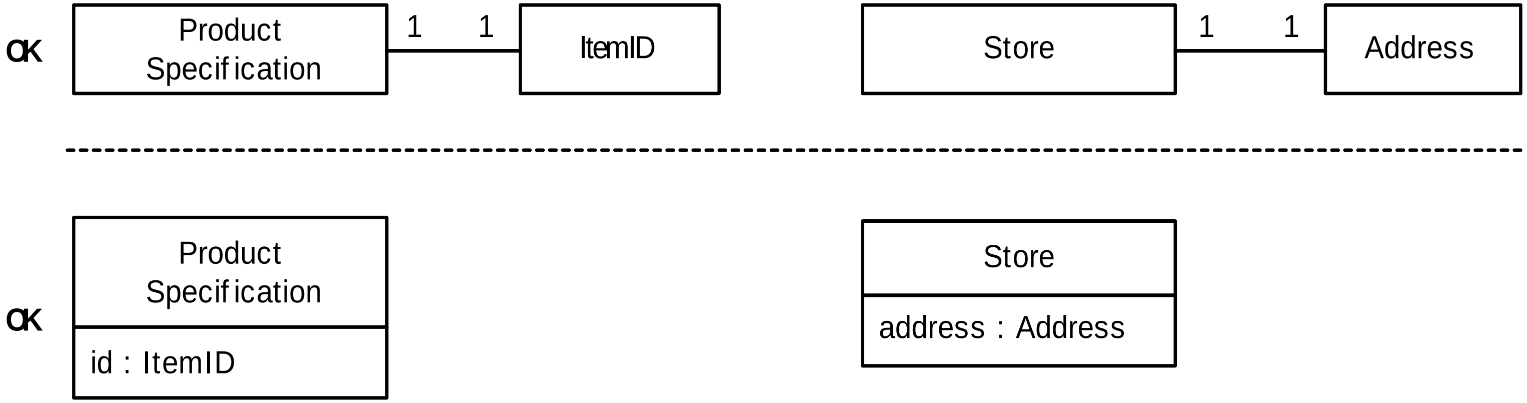
Simple attributes

- Make it an attribute if it is naturrally thought of as number, string, boolean, time….
  - Otherwise represent it as a separate conceptual class rather than as an attribute

- If in doubt, define something as a sepate conceptual class rather than as an attribute.

Non-primitive data type classes

- Represent as a non-primitive class if :
  - It is composes of separate sections
    - Phone number, name, …
  - There are operations associated with it.
  - It has other attributes
  - It is a quantity with a unit
- That means :
  - Address and Quantity are data types that can be considered as separate classes.
  - But it depends on what you wants to emphasize in the diagram
    - It may be shown in the attribute of the class box.
  - A domain model is a tool of communication



OK

Product Specification 1 — 1 ItemID

Store 1 — 1 Address

OK

Product Specification

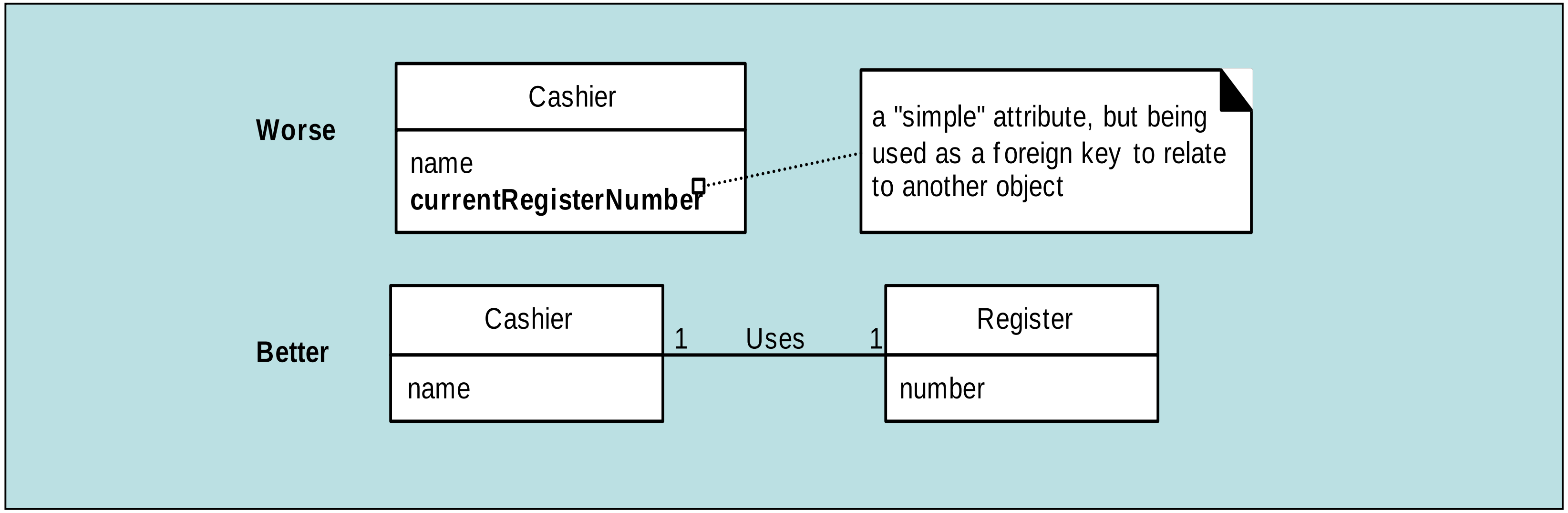
id : ItemID

Store

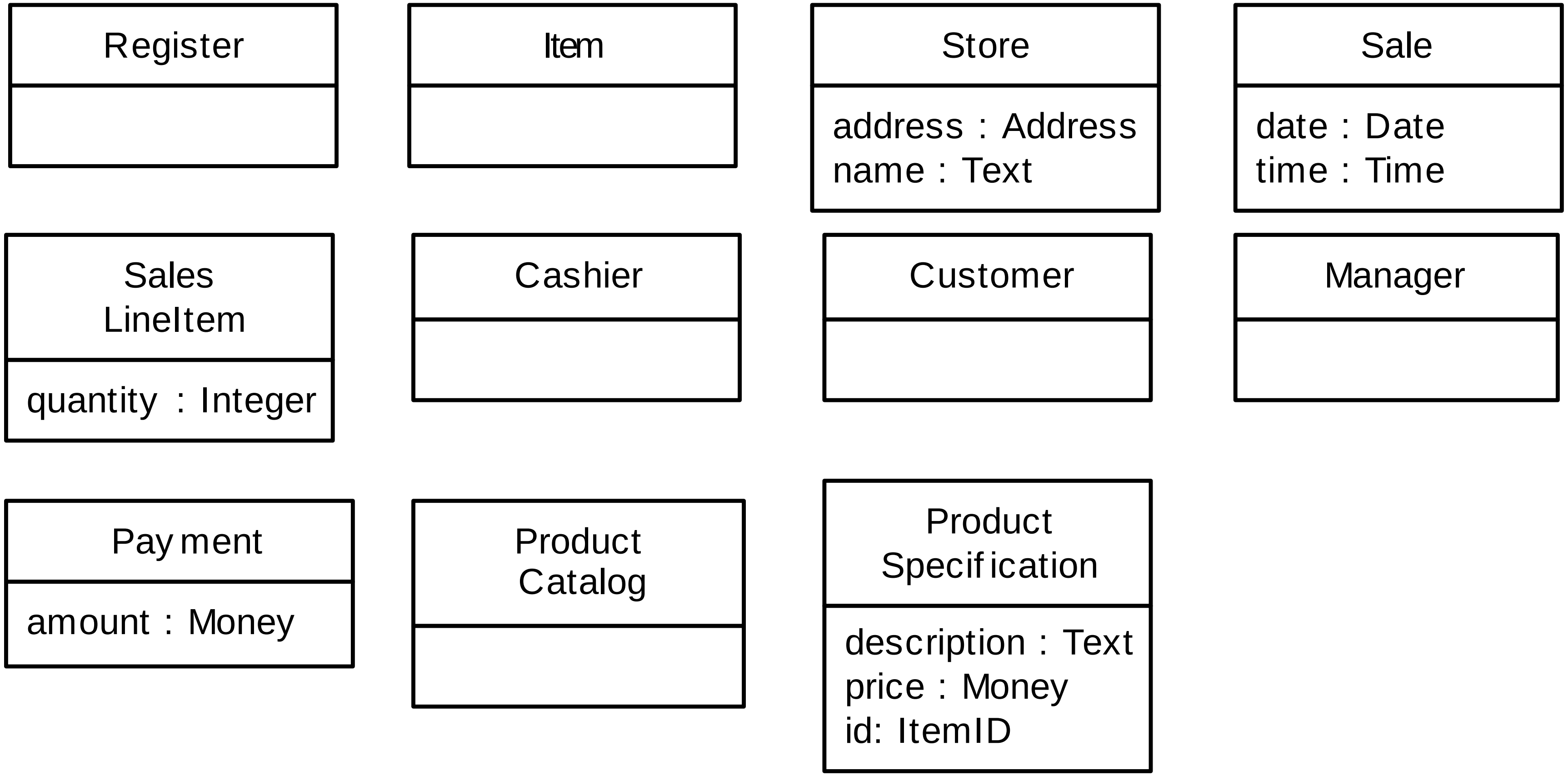
address : Address

No attributes as foreign keys

- Do not use attributes as foreign keys.
  - – Use associations to relate an object to another object.


Worse
Cashier
name
currentRegisterNumber
a "simple" attribute, but being used as a foreign key to relate to another object
Better
Cashier
name
1
Uses
1
Register
number

Attributes in the NextGen Domain Model



Register

Item

Store
address : Address
name : Text

Sale
date : Date
time : Time

Sales LineItem
quantity : Integer

Cashier

Customer

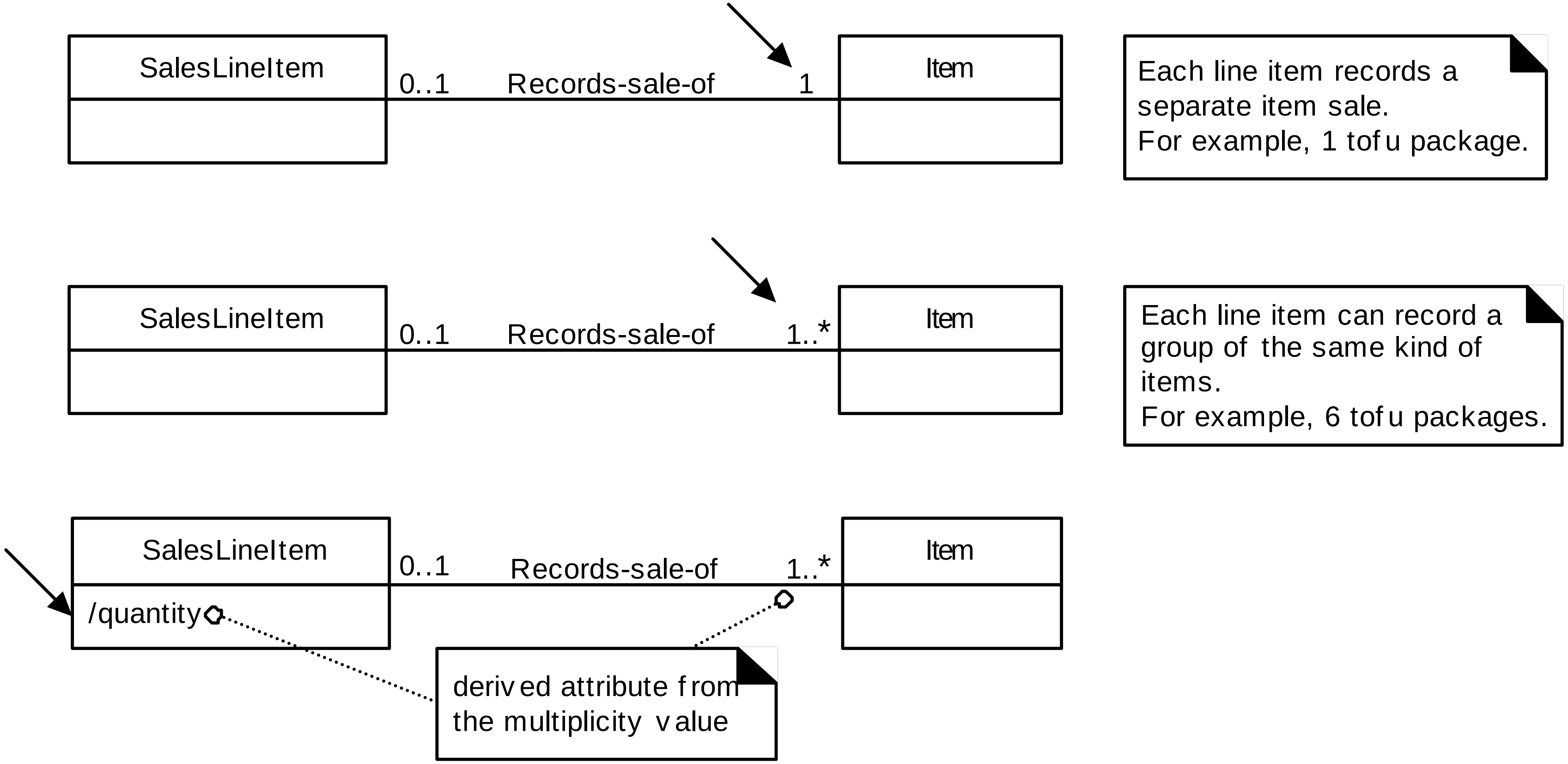
Manager

Payment
amount : Money

Product Catalog

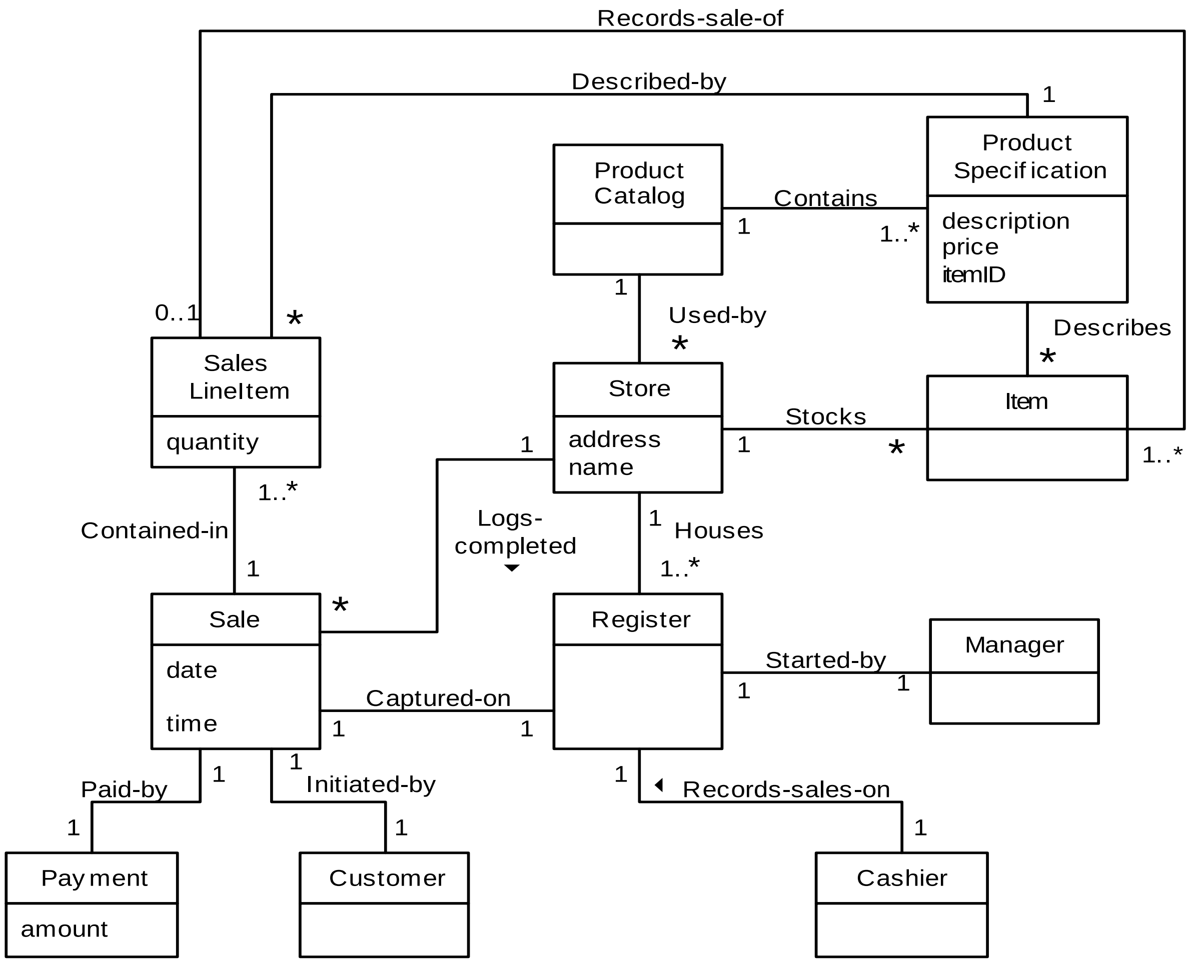
Product Specification
description : Text
price : Money
id: ItemID

Derived attributes



SalesLineItem
0..1
Records-sale-of
1
Item
Each line item records a separate item sale.
For example, 1 tofu package.
SalesLineItem
0..1
Records-sale-of
1..*
Item
Each line item can record a group of the same kind of items.
For example, 6 tofu packages.
SalesLineItem
/quantity
0..1
Records-sale-of
1..*
Item
derived attribute from the multiplicity value

A partial domain model


Records-sale-of
Described-by
1
Product Catalog
Contains
1
1..*
Product Specif ication
description
price
itemID
1
Used-by
*
Describes
*
0..1
*
Sales LineItem
quantity
Store
address
name
Stocks
1
*
Item
1..*
1
1..*
Contained-in
1
Logs-completed
1
Houses
1..*
Sale
date
time
*
Register
Started-by
1
1
Manager
Captured-on
1
1
1
1
Paid-by
Initiated-by
1
Records-sales-on
1
1
1
Pay ment
amount
Customer
Cashier