So Far …

Part 1: OOAD Intro

- SE
- OOAD
- UML
- Iterative, Evolutionary, and Agile
- Case Studies

Part 2: Inception

- Inception is Not the Requirements Phase
- Evolutionary Requirements
- Use Cases
- Other Requirements

Part 3: Elaboration—Iteration 1

- Iteration 1—Basics
- Domain Models
- System Sequence Diagrams
- Operation Contracts
- Requirements to Design—Iteratively
- Logical Architecture and UML Package Diagrams
- On to Object Design

Elaboration

Elaboration Iteration 1 —Basics

Abdulkareem Alali

ack Dale Haverstock

Based on Larman’s Applying UML and Patterns Book, 3d

Iteration—1

Iteration-1 of the elaboration phase **emphasizes a range of fundamental and common OOA/D skills** used in building object systems

Other skills—such as database design, usability engineering, and UI design—but they are out of scope here

Focus is on:

- **More Requirements**,
- **OOA/D** and
- **Applying UML**

Iteration—1 POS Requirements (Where to Start?)

- Implement a basic, key scenario of the **Process Sale** use case
- Implement a **Start Up** (**cash-only scenario**) use case as necessary to support the initialization needs of the iteration
- Happy path scenario, and the design and implementation to support it
- No collaboration with external services
  - e.g. tax calculator or product database
- No complex pricing rules are applied

All the Requirements are Not Implemented at Once, POS

The above requirements for iteration-1 are **subsets** of the complete requirements or use cases

They are a simplified version of the complete Process Sale use case, one simple cash-only scenario

All the requirements analysis for the POS system has not been done, only the Process Sale use case has been done in detail, many others are not yet analyzed

Iteration—1 Monopoly Requirements (Where to Start?)

Implement a basic, key scenario of the Play Monopoly Game use case:

- Players moving around the squares of the board

Implement a Start Up use case as necessary to support the initialization needs of the iteration

Two to eight players can play, a game is played as a series of rounds

Iteration—1 Monopoly Requirements

- During a round: Each player takes one turn, a player advances his/her piece clockwise around the board a number of squares equal to the sum of the number rolled on two six-sided dice
- Play the game for only 20 rounds
- After the dice are rolled, the name of the player and the roll are displayed
- When the player moves and lands on a square, the name of the player and the name of the square that the player landed on are displayed

Iteration—1 Monopoly Requirements

- In iteration-1 there is:

  no money,

  no winner or loser,

  no properties to buy or rent to pay, and

  no special squares of any kind

Iteration—1 Monopoly Requirements

- Each square has a name .. **Go**, **Square1**, **Square2**, … **Square39**

- Every player begins the game with their piece located on the square named Go

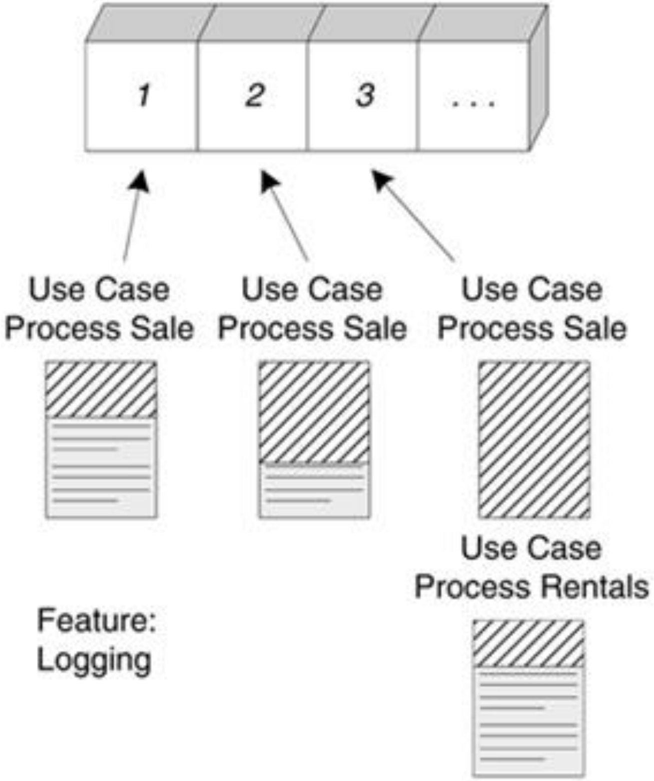
- Run the game as a simulation requiring no user input, other than the number of players

All the Requirements are Not Implemented at Once

Scenarios of the same use case are commonly worked on over several iterations

And gradually extend the system until it finally handles all the required functionality

Use Case Implementation May Be Spread Across Iterations



A use case or feature is often too complex to complete in one short iteration.

Therefore, different parts or scenarios must be allocated to different iterations.

Inception Activities and Artifacts Summary

- Inception may last only one week
- Some artifacts begun but incomplete
- Inception is a short step to elaboration
- Short requirements workshop were most actors, goals, use cases named
- Brief format most vs. 10–20% use cases are fully dressed to improve understanding of the scope and complexity
- Most influential and risky quality requirements are identified
- The Vision, Glossary, Domain Rules and Supplementary Specification has an initial version
- Technical proof-of-concept prototypes
  - Does Java UI work properly on touch-screen displays?
- Risk list—e.g. demo for a trade show in Hamburg, in 18 months
- UI-oriented prototypes clarify vision of functional requirements
- Buy/build/reuse components recommendations, candidate tools list
- High-level candidate architecture have been proposed
- A plan for the iteration—1 has been created

Elaboration Activities and Artifacts Summary

In initial elaboration iterations:

- Core architecture is programed and tested, Majority of requirements will be discovered and stabilized, Major risks will be resolved

In the UP, “risk” includes use cases of business value

2 or more iterations of 2 - 6 week time-boxed (fixed) duration, do short time-boxed risk-driven iterations

Estimate the overall schedule and resources

No prototypes, code and design are production-quality portions of the final system

Executable architecture or architectural baseline

Start programming and testing early, adaptively design, implement, and test realistically the core and risky parts of the architecture

Adapt based on feedback from tests, users, developers

Write most of the use cases and other requirements in detail, through a series of workshops, once per elaboration iteration

Planning the next iteration is begun in iteration—1

Requirements are organized and ranked according to:

- Risk, Coverage (major parts of the system are at least touched), Criticality (Business reasons)

Plan of iterations is adaptive also, it is not written in stone

Elaboration Briefly

Elaboration in a sentence

- "Build the core architecture,
- resolve the high-risk elements,
- define most requirements,
- and estimate the overall schedule and resources." [Larman]

Domain Model

Abdulkareem Alali

ack Dale Haverstock

Based on Larman’s Applying UML and Patterns Book, 3d

It's all very well in practice, but it will never work in theory.

—anonymous management maxim

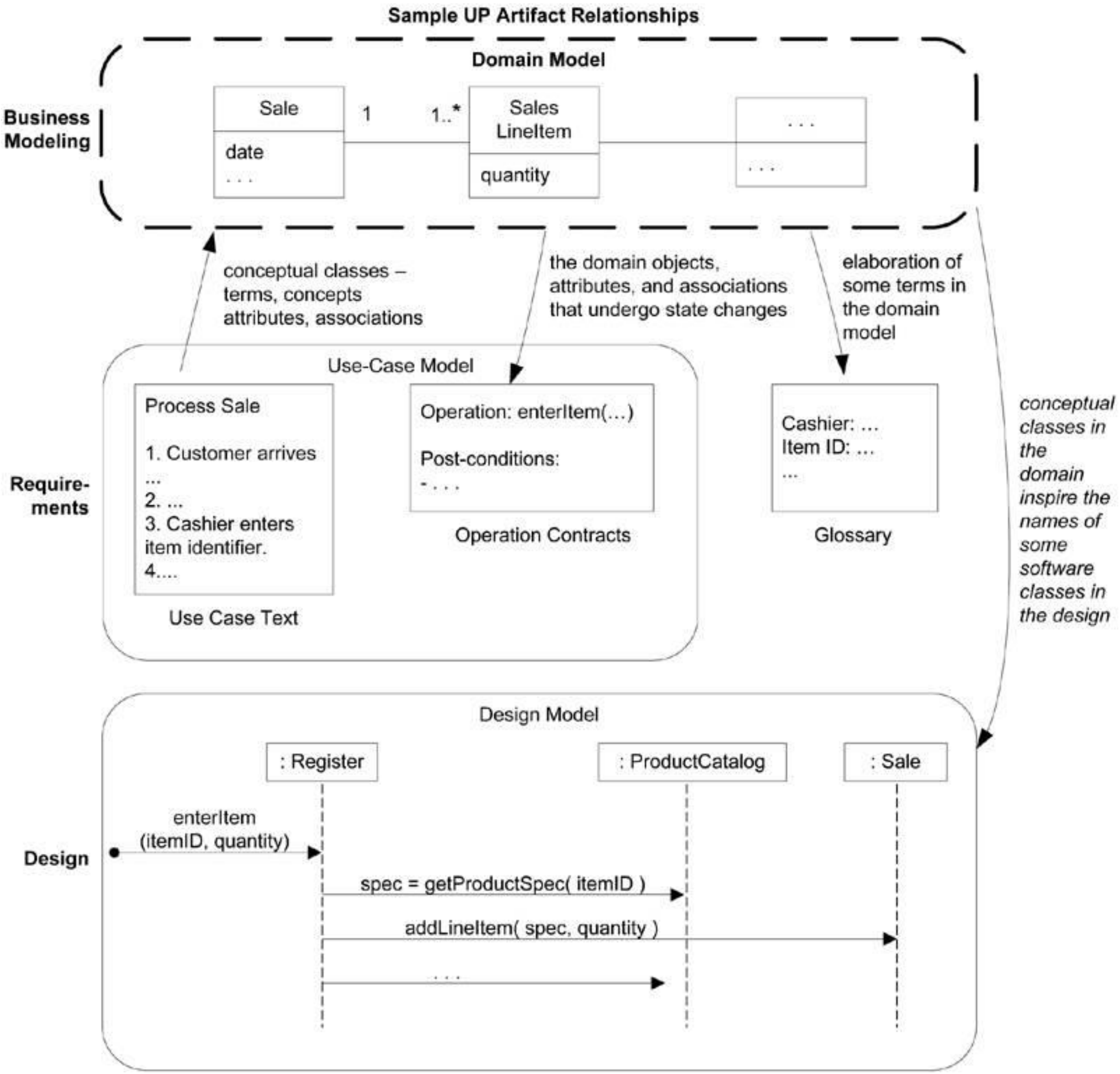
Domain Model

Business Modeling Discipline
Applying OOA produces the Domain Model artifact

Bounded by the use case scenarios (requirement analysis) which is also the input to the Domain Model

Illustrates noteworthy concepts in a domain

Will act as a source of inspiration for software object design, and input to other artifacts

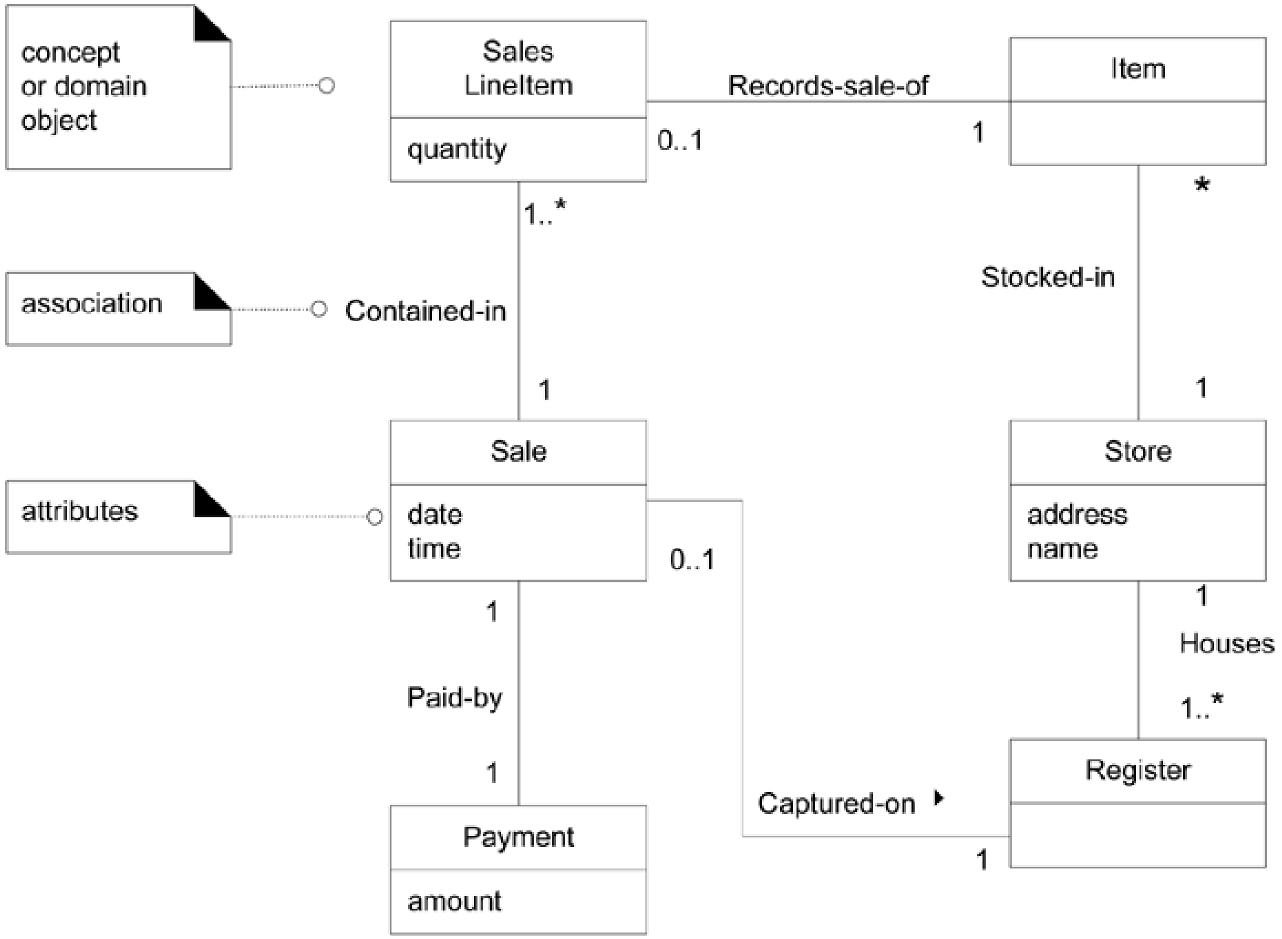

Sample UP Artifact Relationships
Business Modeling
Domain Model
Sale
date
. . .
1
1..*
Sales LineItem
quantity
. . .
. . .
conceptual classes – terms, concepts attributes, associations
the domain objects, attributes, and associations that undergo state changes
elaboration of some terms in the domain model
Require-ments
Use-Case Model
Process Sale
1. Customer arrives
...
2. ...
3. Cashier enters item identifier.
4....
Use Case Text
Operation: enterItem(…)
Post-conditions:
- . . .
Operation Contracts
Cashier: …
Item ID: …
...
Glossary
conceptual classes in the domain inspire the names of some software classes in the design
Design
Design Model
: Register
: ProductCatalog
: Sale
enterItem (itemID, quantity)
spec = getProductSpec(itemID)
addLineItem(spec, quantity)
. . .

Domain Model

Domain model includes a UML class diagram showing important objects in the problem domain

Guideline: "Avoid a waterfall-mindset big-modeling effort to make a thorough or "correct" domain model - it won't ever be either, and such over-modeling efforts lead to analysis paralysis, with little or no return on the investment.“ Although we are tempted to!

Partial POS Domain Model —Visual Dictionary



What is a Domain Model?

In OOA noteworthy objects in the problem domain are identified

These noteworthy objects are called **Conceptual Classes**

A visual representation (UML) of the objects is created

What is a Domain Model?

The diagram will show:

1. **Domain objects or Conceptual classes**
2. **Associations**
3. **Attributes**

Not software classes, no functionalities assigned, that is **OOD**

The Domain Object Model as a Visual Dictionary

Notice how the diagram visualizes and relates words and concepts in the domain

The same information represented textually (in the UP Glossary) would not be as clear

The human brain is good at understanding **picture-type information**

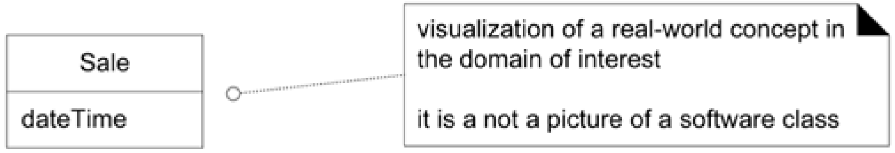
Domain Model —Avoid

Depiction of the elements of the problem

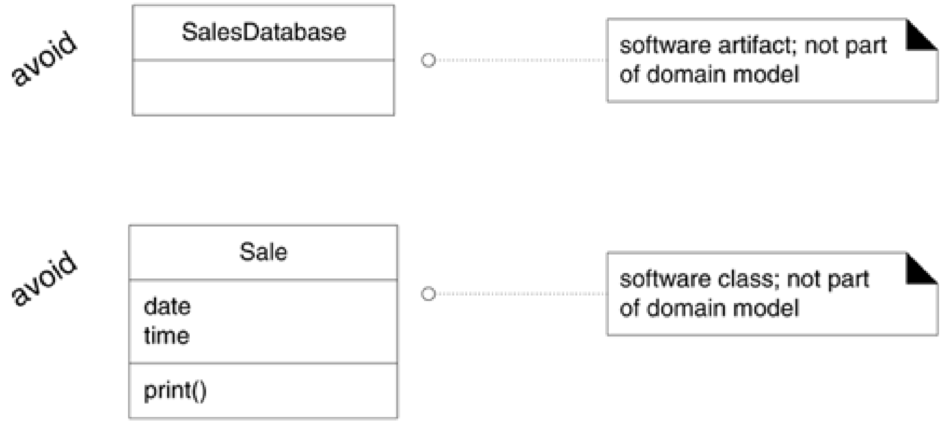
We don't want:

- Software artifacts, such as a window or a database
    - Unless that's the domain being modeled
- Responsibilities or methods of the objects

Real-situation conceptual classes, not software classes


Sale
dateTime
visualization of a real-world concept in the domain of interest
it is a not a picture of a software class

Does not show software artifacts or classes


avoid
SalesDatabase
software artifact; not part of domain model
avoid
Sale
date
time
print()
software class; not part of domain model

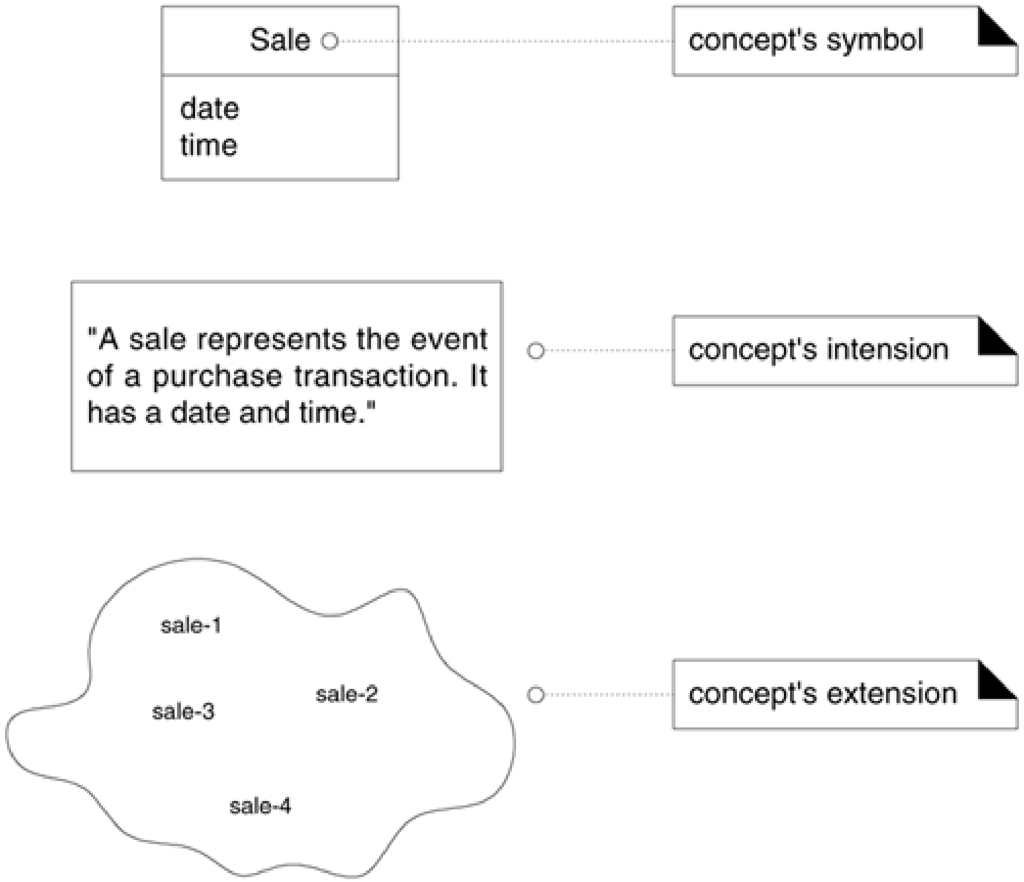
1. Conceptual Classes

Informally, a conceptual class is an idea, thing, or object in the problem domain

A conceptual class expressed as:

- Symbol—Words or images representing a conceptual class
- Intension—The definition of a conceptual class
- Extension—The set of examples to which the conceptual class applies, instances

A conceptual class has a symbol, intension, and extension



Domain Models ≠ Data Models

A data model shows persistent data to be stored somewhere

Domain model objects may or may not record data

Don't include or exclude objects from the domain model based on data they may store

Low Representational Gap (LRG)

Conceptual class names are derived from names in the problem domain

Some of the conceptual class names will be used for software classes

Results in a low representational gap (LRG) between the mental and the software models, a more comprehensible software

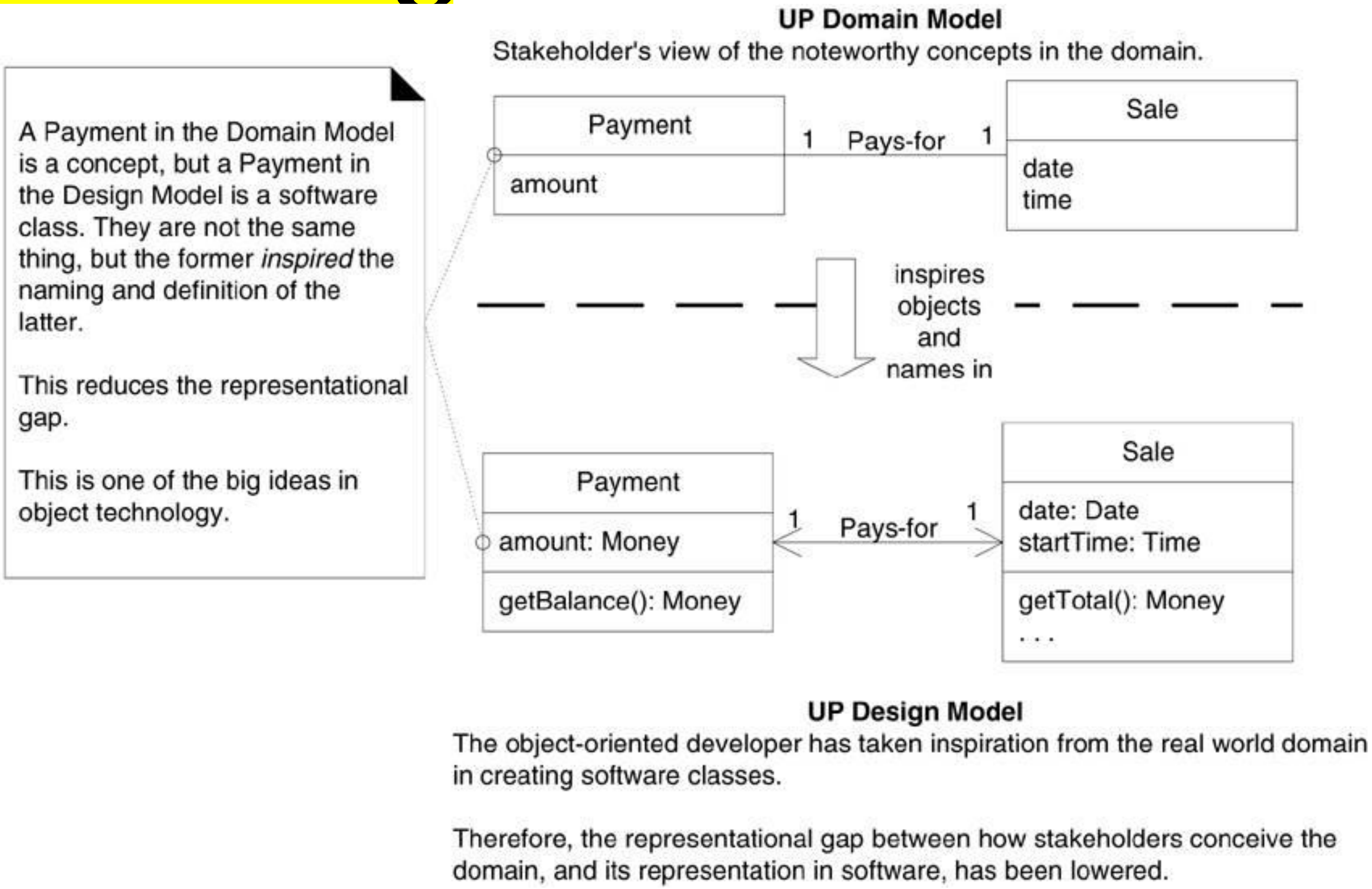
Low Representational Gap (LRG)

e.g. Source-code payroll program written in 1953:

10000101010001110101010101000101010101010111010101 ...

The gap between this software representation and our mental model of the payroll domain is HUGE

Lower Representational Gap with OO Modeling



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.

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.

How to Create a Domain Model?

To create a domain model object diagram:

A. **Find the conceptual classes (How?)**

B. **Draw the conceptual classes as classes in a UML class diagram**

C. **Add associations and attributes**

A. Find Conceptual Classes

1. **Reuse or modify existing models**

2. **Use a category list**

3. **Identify noun phrases**

1. Reuse/Modify Existing Models

There are published, well-crafted domain models for many common domains, such as inventory, finance, health, etc.

This is an excellent, and often the easiest approach

This approach is outside of our scope

2. Use a Category List

Kick-start, make a list of candidate conceptual classes

Common categories that are usually worth considering, with an emphasis on business information system needs

e.g. **Transactions** are fundamental in most business applications and the best place to start

Examples next are drawn from the 1) **POS**, 2) **Monopoly**, and 3) **Airline Reservation** domains

Conceptual Class Category List

Conceptual Class Category	**Examples**
business transactions These are critical (they involve money), so start with transactions.	*Sale, Payment, Reservation*
transaction line items Transactions often come with related line items, so consider these next.	*SalesLineItem*
product or service related to a transaction or transaction line item Transactions are *for* something (a product or service). Consider these next.	*Item, Flight, Seat, Meal*
where is the transaction recorded? Important.	*Register, Ledger, FlightManifest*
roles of people or organizations related to the transaction; actors in the use case We usually need to know about the parties involved in a transaction.	*Cashier, Customer, Store, MonopolyPlayer, Passenger, Airline*
place of transaction; place of service	*Store, Airport, Plane, Seat*
noteworthy events, often with a time or place we need to remember	*Sale, Payment, MonopolyGame, Flight*
physical objects This is especially relevant when creating device-control software, or simulations.	*Item, Register, Board, Piece, Die, Airplane*
descriptions of things	*ProductDescription, FlightDescription*
Catalogs Descriptions are often in a catalog.	*ProductCatalog, FlightCatalog*
containers of things (physical or information)	*Store, Bin, Board, Airplane*
things in a container	*Item, Square (in a Board), Passenger*
other collaborating systems	*CreditAuthorizationSystem, AirTrafficControl*
records of finance, work, contracts, legal matters	*Receipt, Ledger, MaintenanceLog*
financial instruments	*Cash, Check, LineOfCredit, TicketCredit*
schedules, manuals, documents that are regularly referred to in order to perform work	*DailyPriceChangeList, RepairSchedule*

3. Identify Noun Phrases

Linguistic Analysis: Identify the **nouns** and **noun phrases** in textual descriptions of a domain, and consider them as candidate conceptual classes or attributes

A mechanical noun-to-class mapping is not possible, words in natural languages are ambiguous, different nouns may represent the same concept

Identify Noun Phrases

A noun may represent an attribute

The concept the noun represents may not be something we want to consider at this point

A fully-dressed use case is a good place to start

Check other documents or get from discussions with experts

Nouns to Classes —POS Domain

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)

9. System presents **receipt**

10. Customer leaves with receipt and goods (if any)

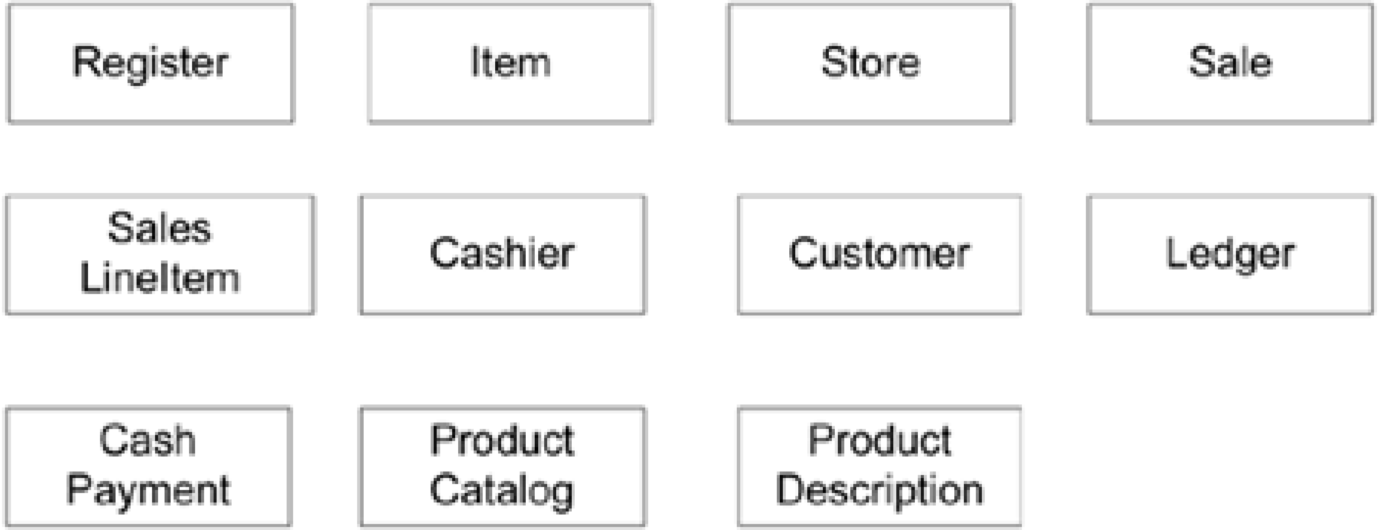
Extensions (or Alternative Flows):

. . .

7a. Paying by cash:

1. Cashier enters the cash **amount tendered**
2. System presents the **balance due**, and releases the **cash drawer**
3. Cashier deposits cash tendered and returns balance in cash to Customer
4. System records the cash payment

Nouns to Classes —POS Domain



Register

Item

Store

Sale

Sales LineItem

Cashier

Customer

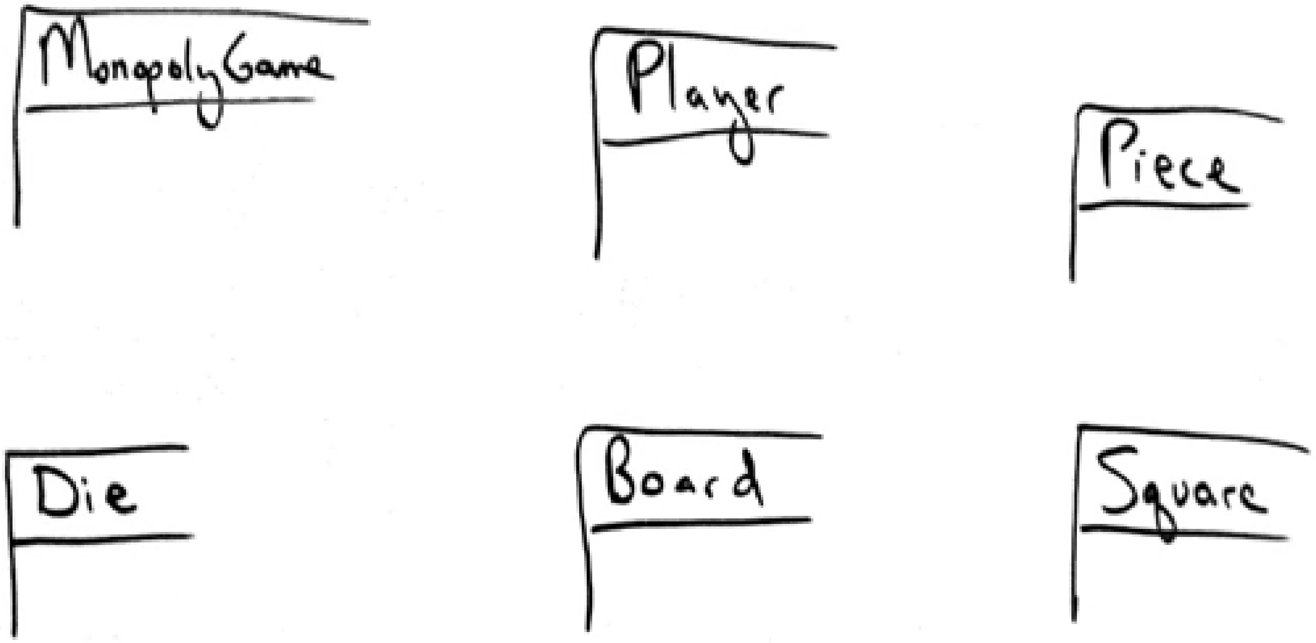
Ledger

Cash Payment

Product Catalog

Product Description

Nouns to Classes —MonopolyGame Domain



Model the Unreal World

Software for domains with very little analogy in natural or business domains; e.g. telecommunications software

High degree of abstraction

Candidate conceptual classes related to the domain of a e.g. a telecommunication switch:

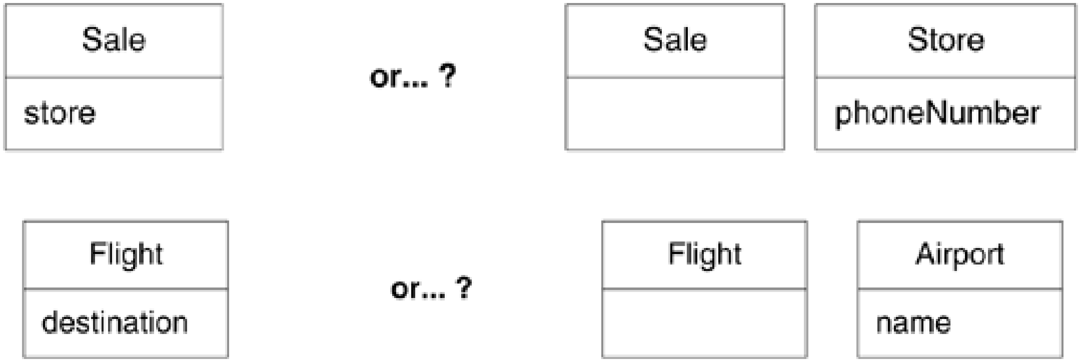
- Message, Connection, Port, Dialog, Route, Protocol.

Attributes vs. Conceptual Classes

It is easy to represent something as an attribute that should be a class

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.



Consider:

Should the store be an attribute of Sale?

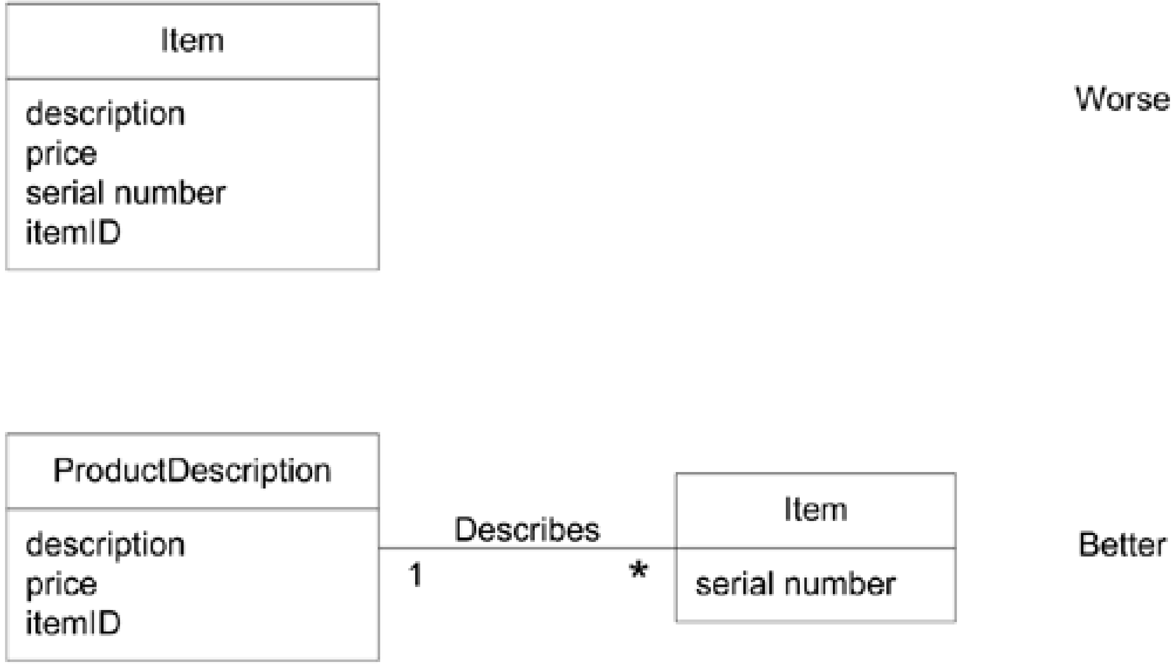
Should the destination be an attribute of Flight?

Description Classes

A description class contains information that describes something else

Consider an Item in the POS, a **ProductDescription** that records the price, a picture, and a text description of the Item. Why?

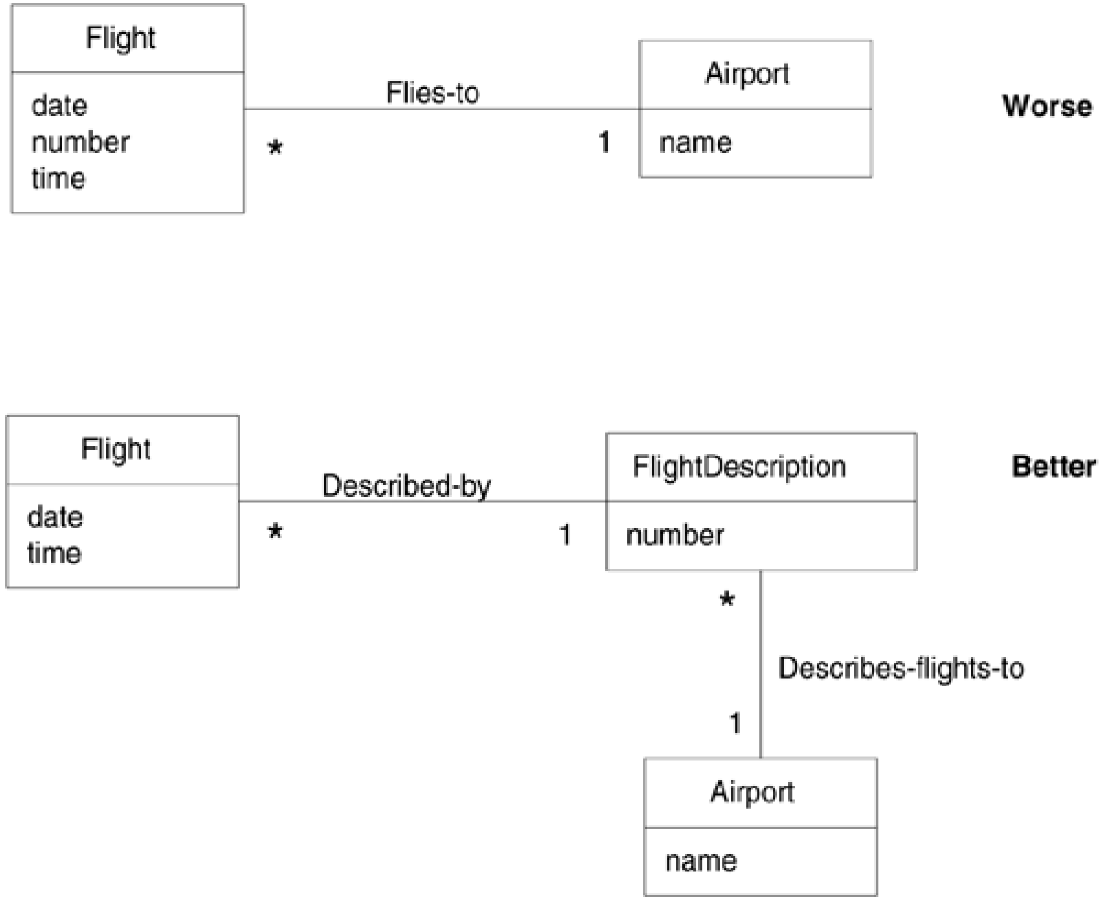
Descriptions About Other Things



Description Classes?

- Values of attribute would be the same for all Item objects. No need to duplicate the information in all Item objects (1:M)
- If a picture of some Item was needed but that Item was sold out, i.e., there are no instances of the Item object? No image would be available
- If a picture of some Item needed to be updated? All instances of the Item would need to be updated

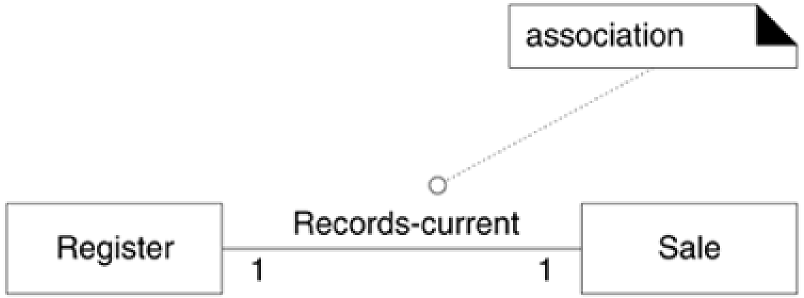
Descriptions About Other Things



2. Associations

An association is a relationship between classes that indicates some meaningful and interesting **connection**

Showing associations clarifies the analysis model by making relationships between objects explicit



When to Show an Association?

Implies knowledge of a relationship that needs to be **preserved for some duration**,

it could be for milliseconds or indefinitely, depending on context, **need-to-remember**

When to Show an Association?

- Need-to-remember what **SalesLineItem** instances are associated with a **Sale** instance
  - Would not be possible to reconstruct a sale without it, print a receipt, or sale total

- Need-to-remember completed **Sales** in a **Ledger**, for accounting and legal purposes
- Need-to-remember what **Square** a **Piece** (or **Player**) is on—game won’t work without it
- Need-to-remember what **Piece** is owned by a particular **Player**
- Need-to-remember what **Squares** are part of a particular **Board**

- No need-to-remember **Dice** *total* (**Square** to move to)
- No need-to-remember a **Cashier** may look up **ProductDescriptions**

ClassName VerbPhrase ClassName

Simple association names such as Has or Uses are usually poor, seldom enhance understanding domain

Sale Paid-by **CashPayment** vs. **Sale** Uses **CashPayment**

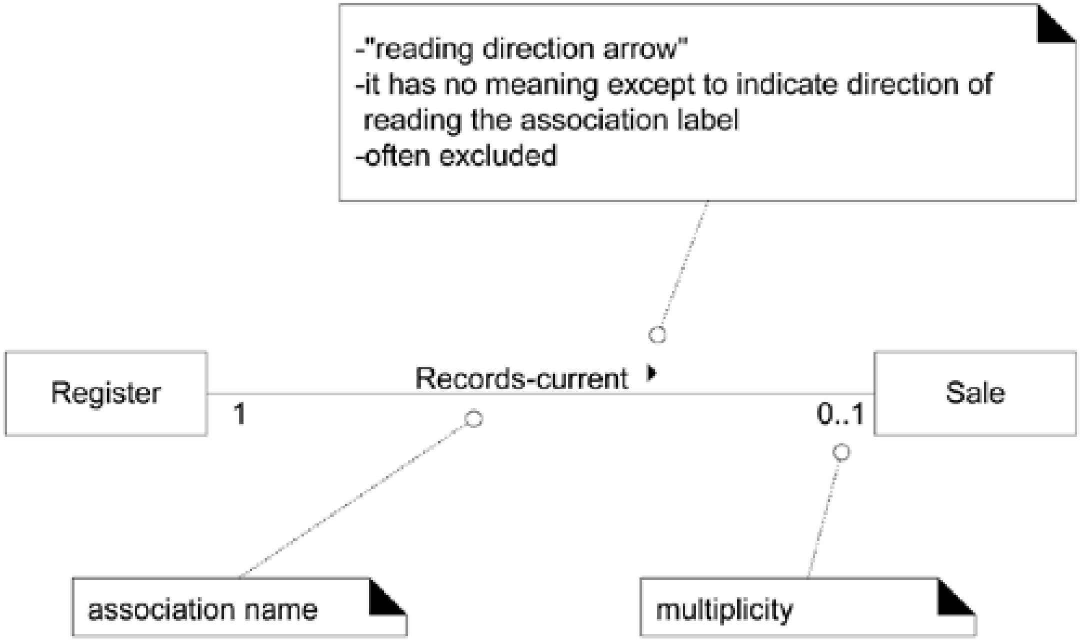
Player Is-on **Square** vs. **Player** Has **Square**

Association names start with a capital letter

Compound association Legal formats:

Records-current or RecordsCurrent

If arrow is not present, the convention is to read the association from left to right or top to bottom


-"reading direction arrow"
-it has no meaning except to indicate direction of reading the association label
-often excluded
Register
Records-current
1
0..1
Sale
association name
multiplicity

Identifying Associations

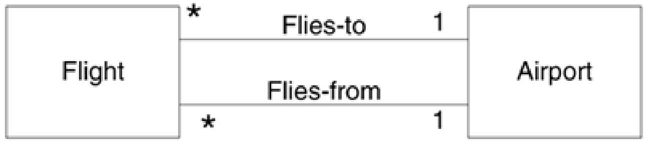
Heuristic for identifying associations:

- Examine verb phrases,
  - e.g.: has, is part of, manages, reports to, is triggered by, is contained in, talks to, includes

- Use a category list
  - of common associations, as with objects

Identifying Associations

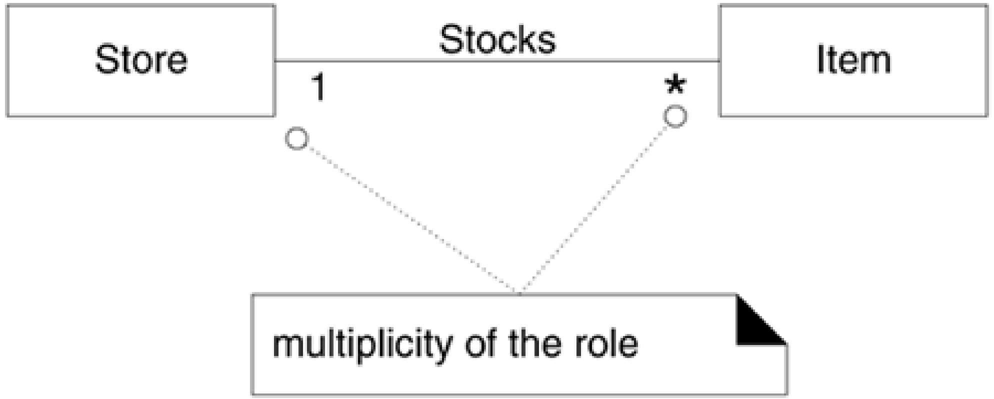
- Two classes may have multiple associations between them, this is not uncommon

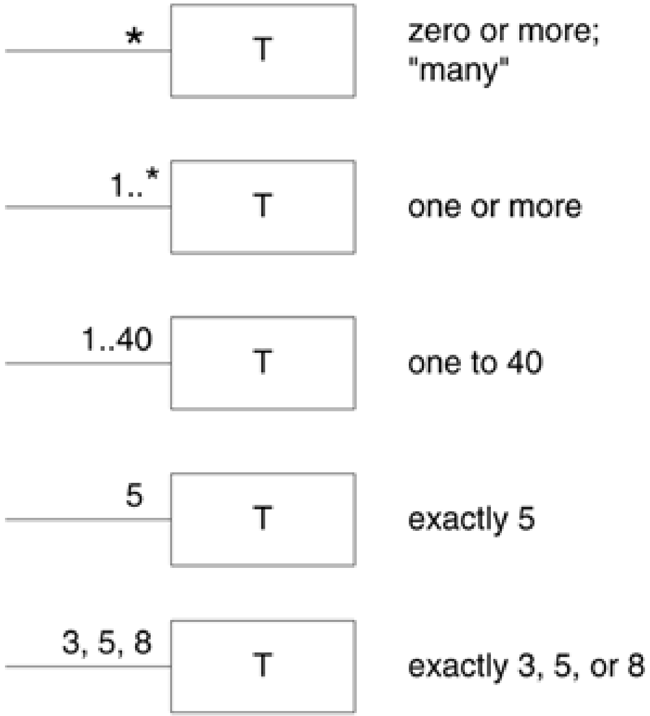


- Too many associations make a model unreadable

- Do not worry about multiplicity until the set of associations is stable

Multiplicity on an Association

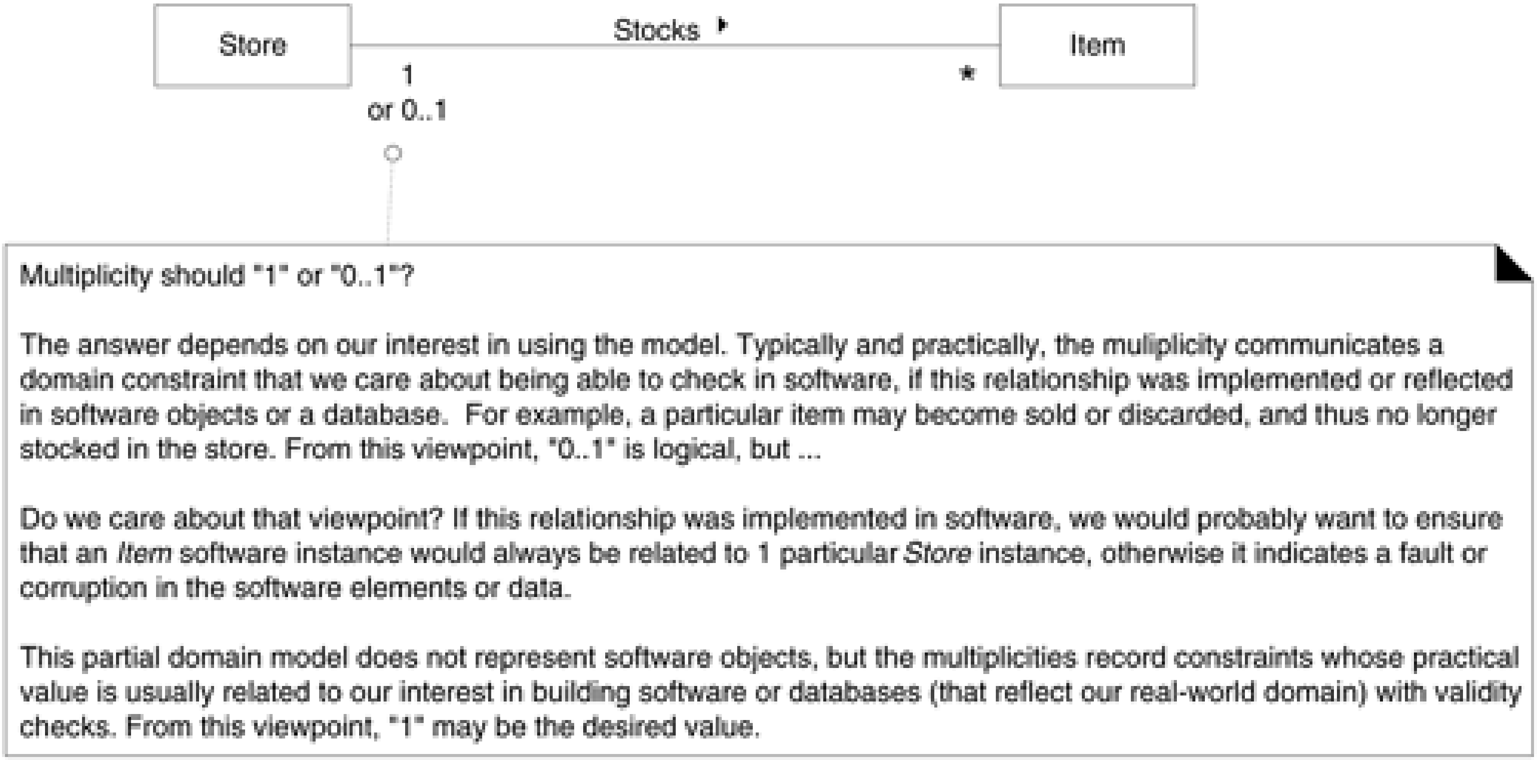




How many instances can be validly associated with another, at a particular moment, rather than over a span of time

Car could be repeatedly sold back to used car dealers over time. But at any particular moment, the car is only Stocked-by one dealer

Multiplicity on an Association



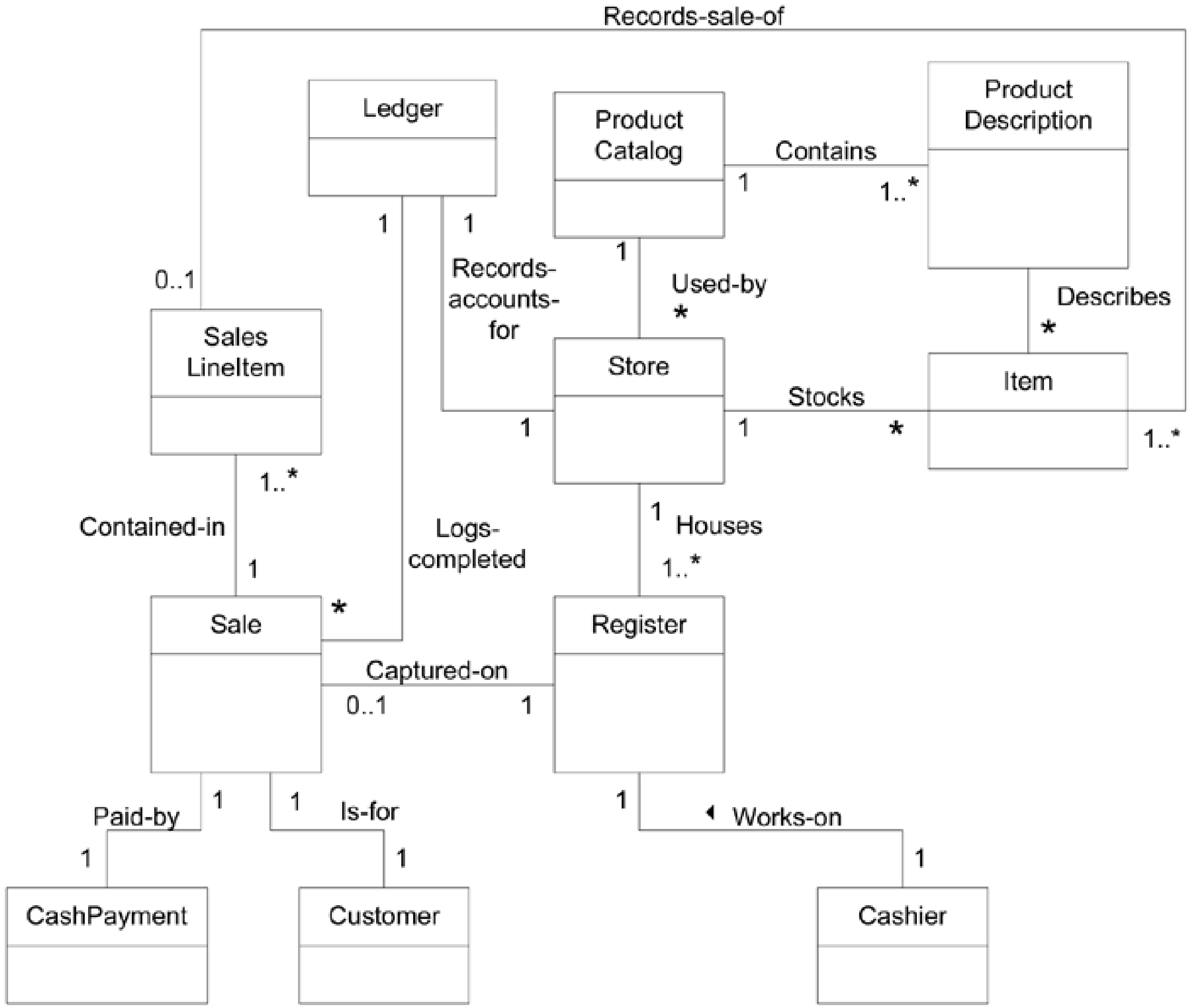
Multiplicity should "1" or "0..1"?

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 ...

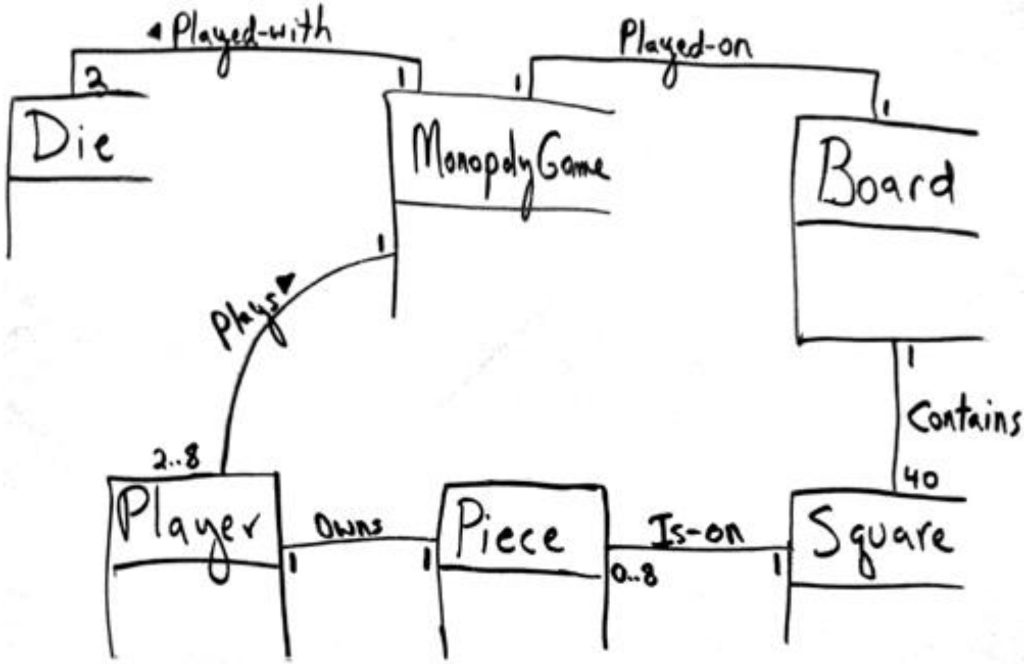
Do we care about that viewpoint? If this relationship was implemented in software, we would probably want to ensure that an *Item* software instance would always be related to 1 particular *Store* instance, otherwise it indicates a fault or corruption in the software elements or data.

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.

Nextgen POS Partial Domain Model, — A Story, A Visual Dictionary!



Monopoly Partial Domain Model



3. Attributes

A named property of a classifier that describes a range of values that instances of the property may hold

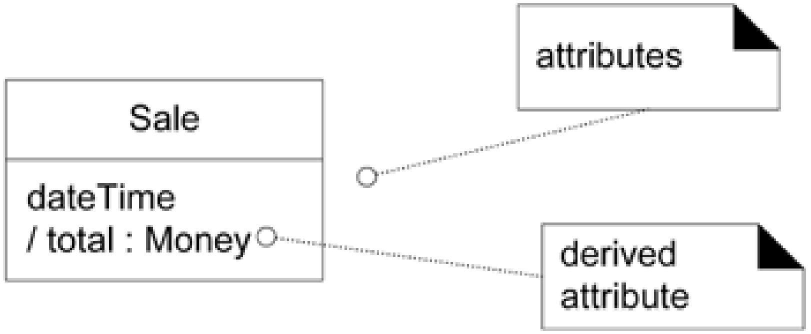
The full syntax for an attribute in the UML is:

visibility name : type multiplicity = default {property-string}

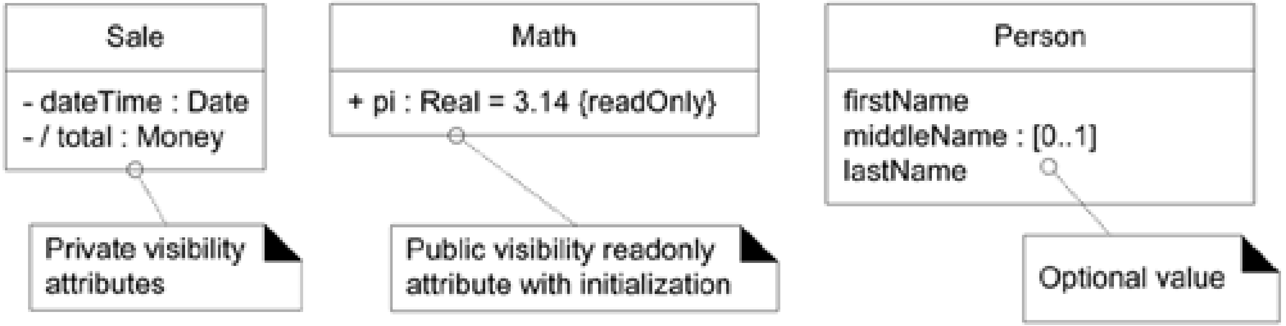
Show attributes that the requirements suggest or imply are needed to remember information

UML Attribute Notation

Class and attributes



Attribute notation in UML


Sale
- dateTime : Date
- / total : Money
Private visibility attributes
Math
+ pi : Real = 3.14 {readOnly}
Public visibility readonly attribute with initialization
Person
firstName
middleName : [0..1]
lastName
Optional value

Recording Attributes

A requirement or domain rule, embedded in the domain model

middleName : [0..1]

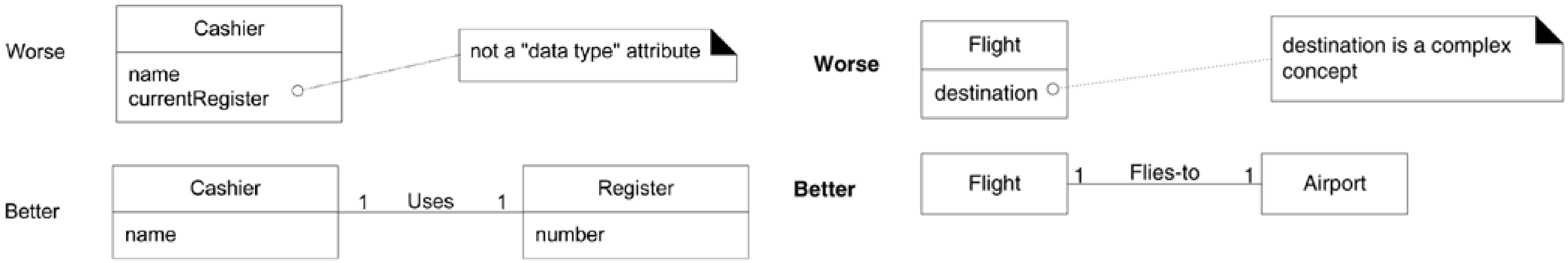
It is easy for this requirement to get overlooked in the UML diagram

Recommended placing all such attribute requirements in the UP Glossary, which serves as a data dictionary

Data Type vs. Associations

Attribute types should be what are often primitive data types (numbers, Booleans, etc.) not complex domain concept, such as a **Sale** or **Airport**

DM Rule: Relate conceptual classes with an association, not with an attribute



Data Types

The attributes in a domain model should preferably be **data type**.

e.g: Boolean, Date, Number, Character, String (Text), Time, Address, Color, Point, Phone Number, Social Security Number, Universal Product Code (UPC), SKU, ZIP or postal codes, enumerated types

A **value test** is done with the equals method, and an identity test with the == operator, Integer 5, String 'cat'., and Date “Nov. 13, 1990”, identified by value.

Data Types

Distinguish (by object identity), two separate **Person** instances whose names are both “Jill Smith” represent separate individuals

Data type values are usually immutable.

- Instance '5' of Integer is immutable;

  “Nov. 13, 1990” Date is immutable.

- A **Person** instance may have its lastName changed for various reasons

Software Data Types

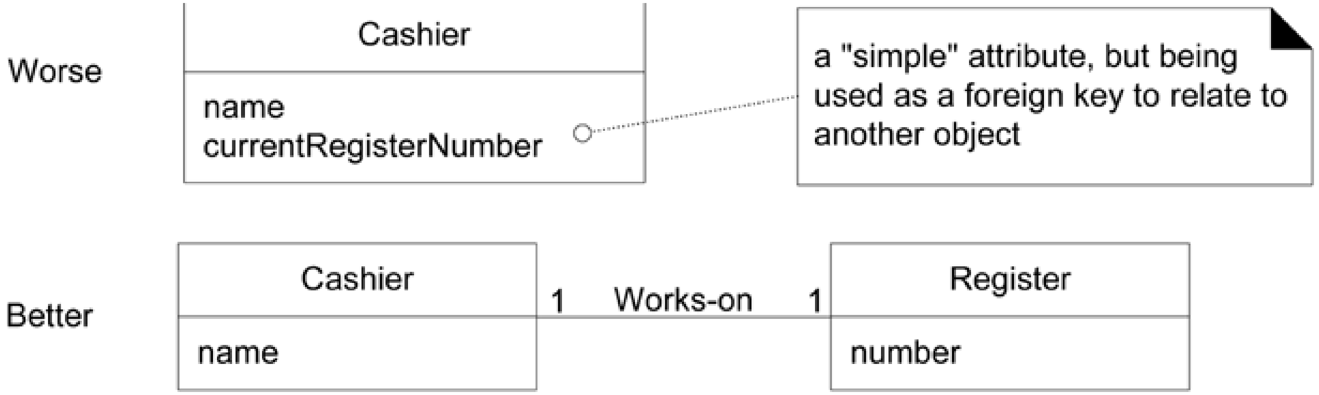
Few situations where one would compare the memory addresses (identity) of instances of Integer or Date; only **value-based** comparisons are relevant

Memory addresses of **Person** instances could conceivably be compared and distinguished, even if they had the same attribute values, because their unique identity is important

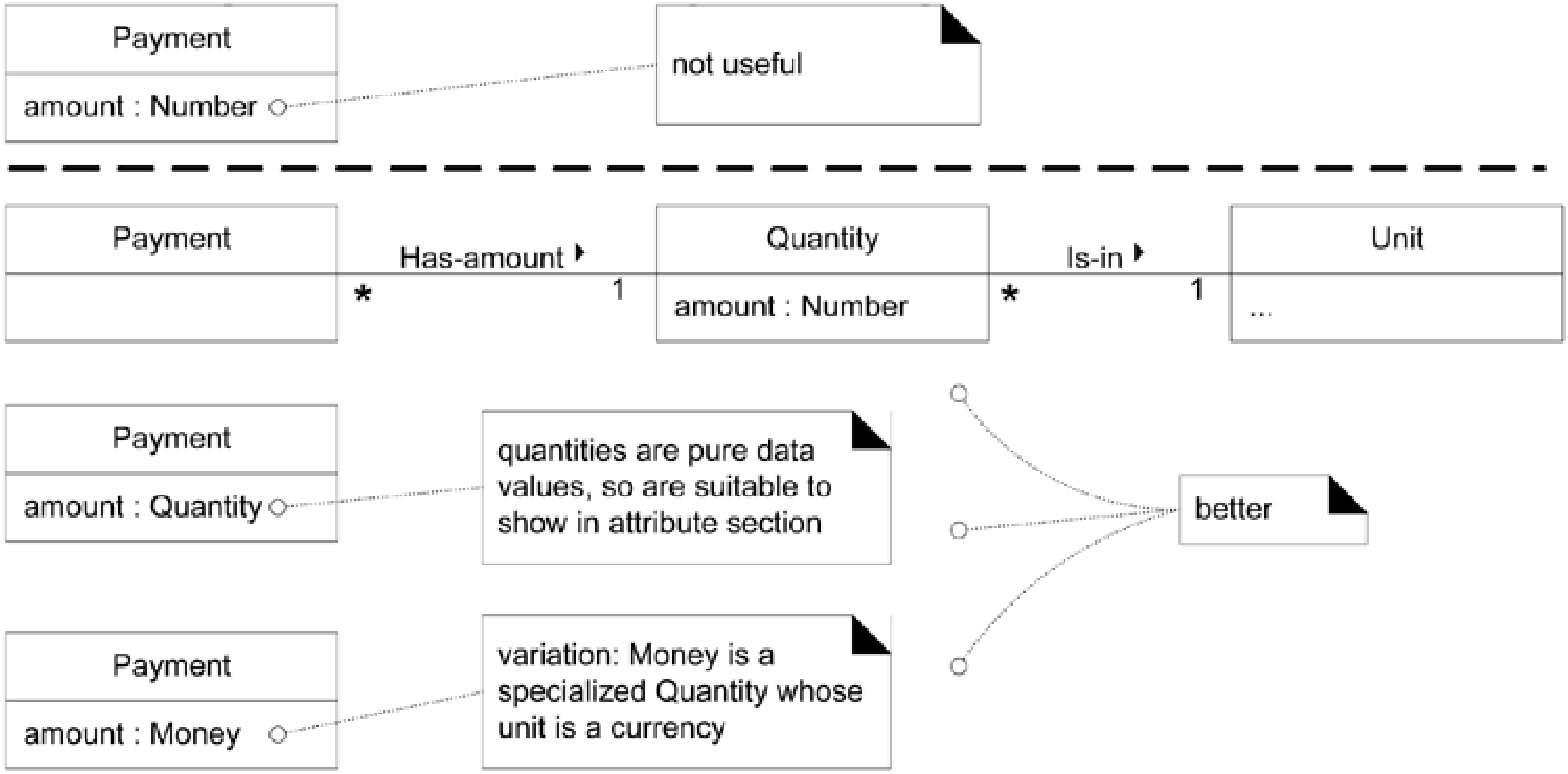
Attributes in the domain model be mainly data types does not imply that C# or Java attributes must only be of simple, primitive data types

Domain model is a conceptual perspective, not a software one. In the Design Model, attributes may be of any type, only primitive types!

Do Not Use Attributes As Foreign Keys


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

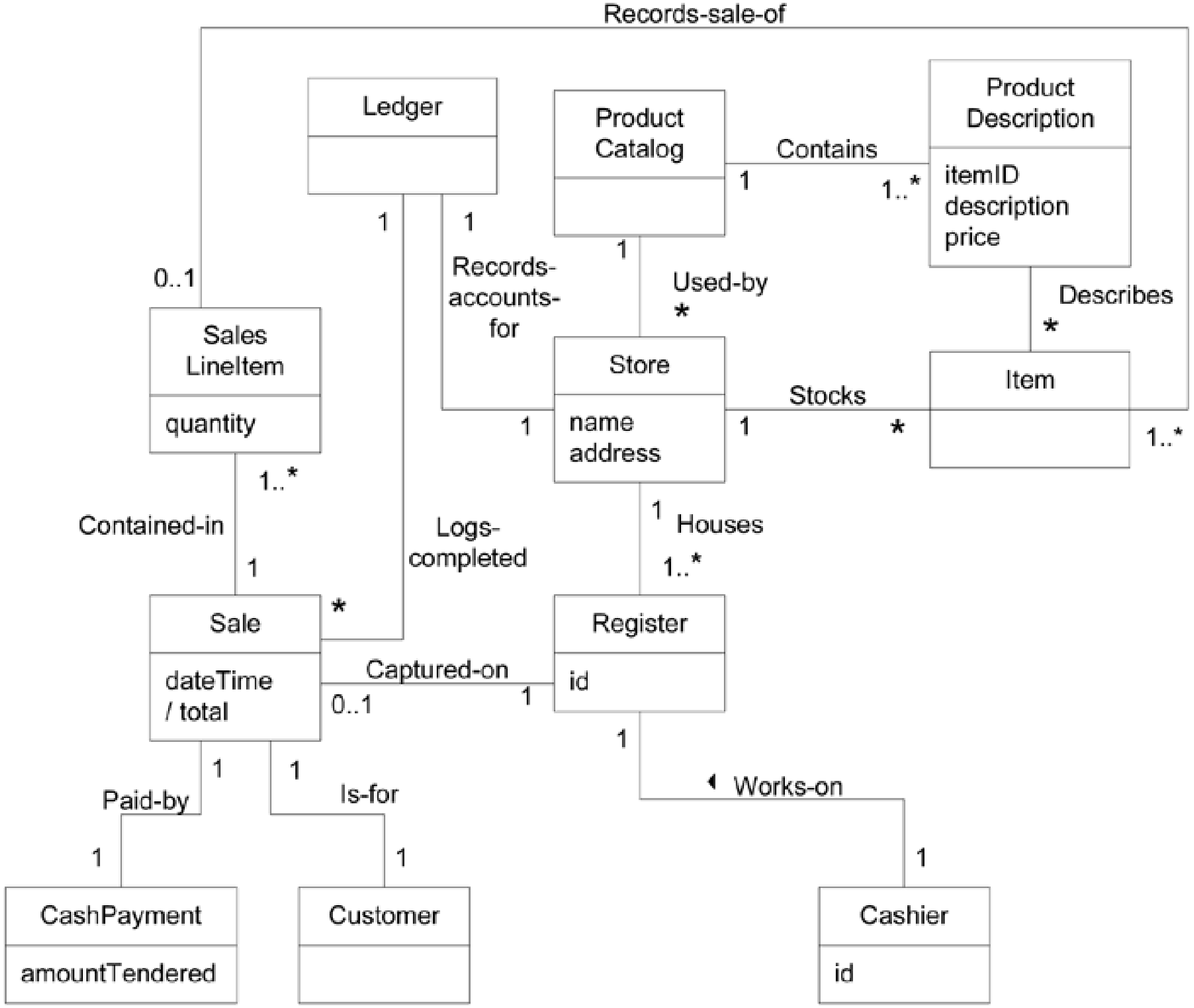
Modeling Quantities


Payment
amount : Number
not useful
Payment
Has-amount ▸
*
1
Quantity
amount : Number
Is-in ▸
*
1
Unit
...
Payment
amount : Quantity
quantities are pure data values, so are suitable to show in attribute section
better
Payment
amount : Money
variation: Money is a specialized Quantity whose unit is a currency

Case Study Attributes —NextGen POS

CashPayment	amountTendered— To determine if sufficient payment was provided, and to calculate change, an amount (also known as “amount tendered”) must be captured.
ProductDescription	description— To show the description on a display or receipt. itemId— To look up a **ProductDescription**. price— To calculate the sales total, and show line item price.
Sale	dateTime— A receipt normally shows date and time of sale, and this is useful for sales analysis.
SalesLineItem	quantity— To record the quantity entered, when there is more than one item in a line item sale (for example, fivepackages of tofu).
Store	address, name— The receipt requires the name and address of the store.

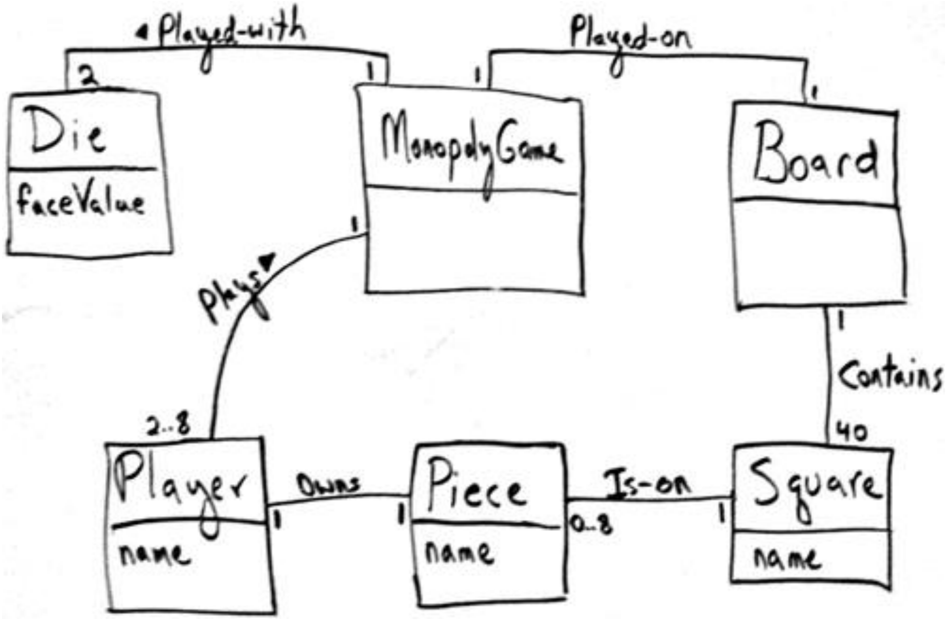
Nextgen POS Partial Domain Model, Updated!



Case Study: Monopoly

Die	faceValue— After rolling the dice, needed to calculate the distance of a move.
Square	name— To print the desired trace output.

Monopoly Partial Domain Model, Updated!



Domain Model —Correctness, and Why?

No single correct domain model, approximations of the domain attempting to understand

A tool of understanding and communication

Captures essential abstractions and information that is required to understand the domain with regard to current requirements

Aid people in understanding the domain;

its concepts,

terminology, and

relationships

Domain Model —Understand Domain!

Captures essential abstractions and information that is required to understand the domain regarding current requirements

Aid people in understanding the domain;

its concepts,

terminology, and

relationships

The Domain Model and Iterative Development

Domain modeling will not take a long time to do, a few hours per iteration

The domain model will evolve across iterations

No domain modeling is done during Inception

Most domain modeling is done during Elaboration