Software Specification & Design

Lecture 2

Today topics

- Requirements to design
- Design to UML
- Design Pattern
- Command Pattern
- Half of the material will not be in the slides

From requirements to design

- What are requirements?
- What are types of requirements?
    - Functional
    - Usability
    - Reliability
    - Performance
    - Supportability

Requirements - Functional

- Features & capabilities
- Describe what the system can do
- Example:
    - Users can register/login to the system
    - Users can like their content

Requirements - Usability

- Human factors
- Understandability, Learnability, Operability, Attractiveness
- Example:
    - The text must be visible from 2 meters.
    - All error messages in the system describe how to fix it.

Requirements - Reliability

- Frequency of failure
- Recoverability
- Predictability
- Example:
    - The system can boot up after 1 minutes after failure.
    - The system will fail less than 3 hours a week.

Requirements - Performance

- Response times
- Throughput
- Accuracy
- Example
    - The system can handle up to 10k concurrent users
    - The new feed calculation is always done within 31.5 seconds

Requirements - Supportability

- Adaptability
- Maintainability
- Internationalization
- Example
    - The app supports Thai and English

From requirements to design

- What should be the main requirements for designing software?
- There are many
- For this class, let’s go with Use case for functional requirements

Use case

- Text stories
- Discover and record requirements
- 3 types, brief, casual, fully dressed

Example of use case (1)

- Dice game use case

  1. Player roll two dice
  2. The system displays results
  3. The player win if the sum of two faces is 7. Otherwise, he lose.

Example of use case (2)

- POS - Process Sale :
    1. A customer arrives at a checkout with items to purchase.
    2. The cashier uses the POS system to record each purchased item.
    3. The system presents a running total and line-item details.
    4. The customer enters payment information
    5. The system validates and records.
    6. The system updates inventory.
    7. The customer receives a recipe from the system and then leaves with the item.

Use case

- How to handle branch scenario (alternate flows)
- Example (from the last page)
    - 3a. Invalid item id
        - 1. System signal error and reject entry
        - 2. Cashier handle error

Use case

- How to handle exception
- Example (from the last 2 page)
    - *a At anytime, System crashes
        - 1. Cashier restart the system, logs in, and recover the last state
        - 2. The system resume the last state

From requirements to design

- Now, we have use cases, what next?
- We need some thing to translate use cases into design language.

Domain model

- A visual representation of conceptual classes or real-situation objects in a domain
- Conceptual models, Domain Object Models, Analysis Object Model
- Are not software objects or software classes
- [domain objects] [associations] [attributes]

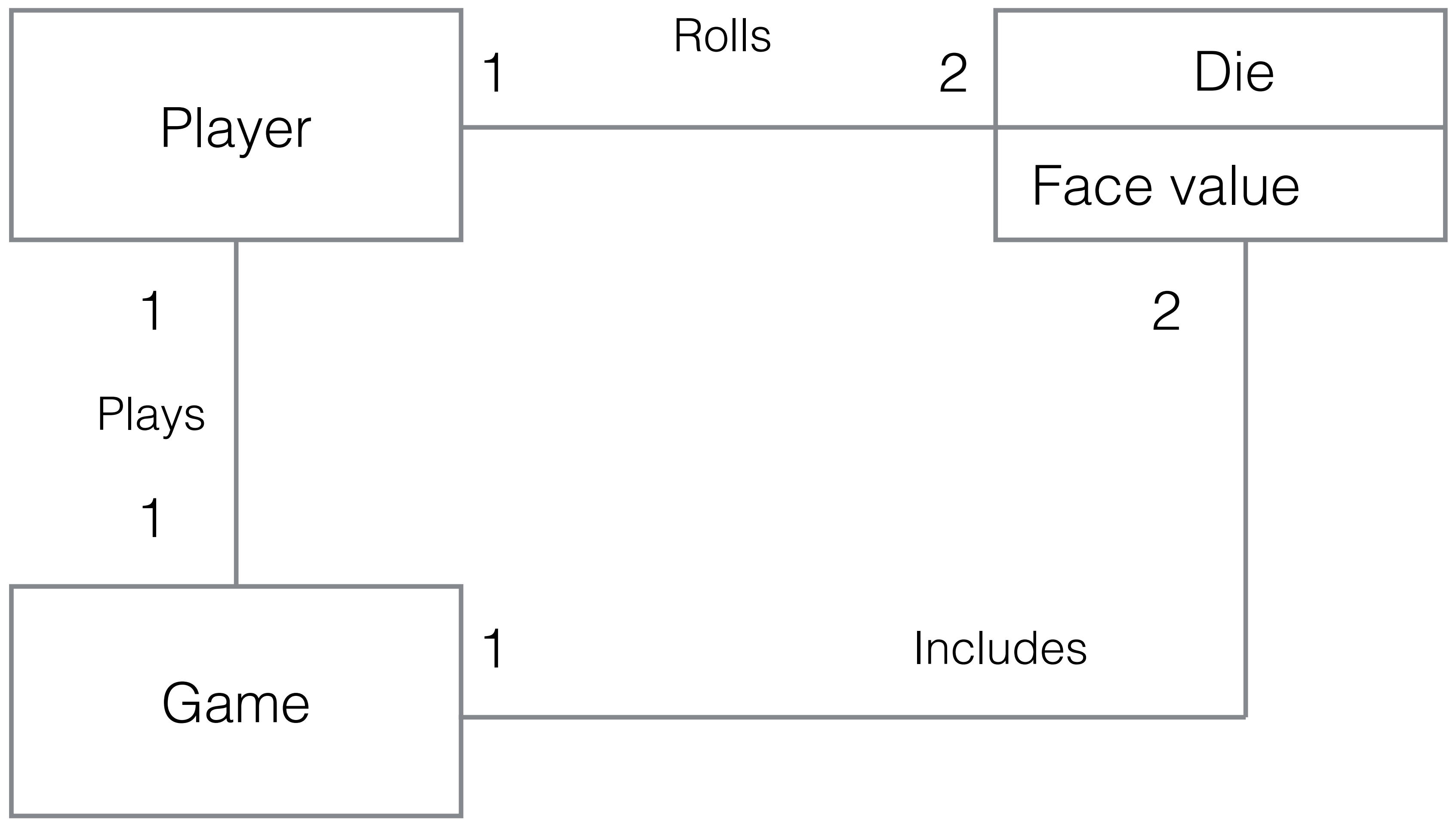
Example use case

- Dice game

Dice Game

- Dice game use case
    - Player roll two dice
    - The system displays results
    - The player win if the sum of two faces is 7. Otherwise, he lose.

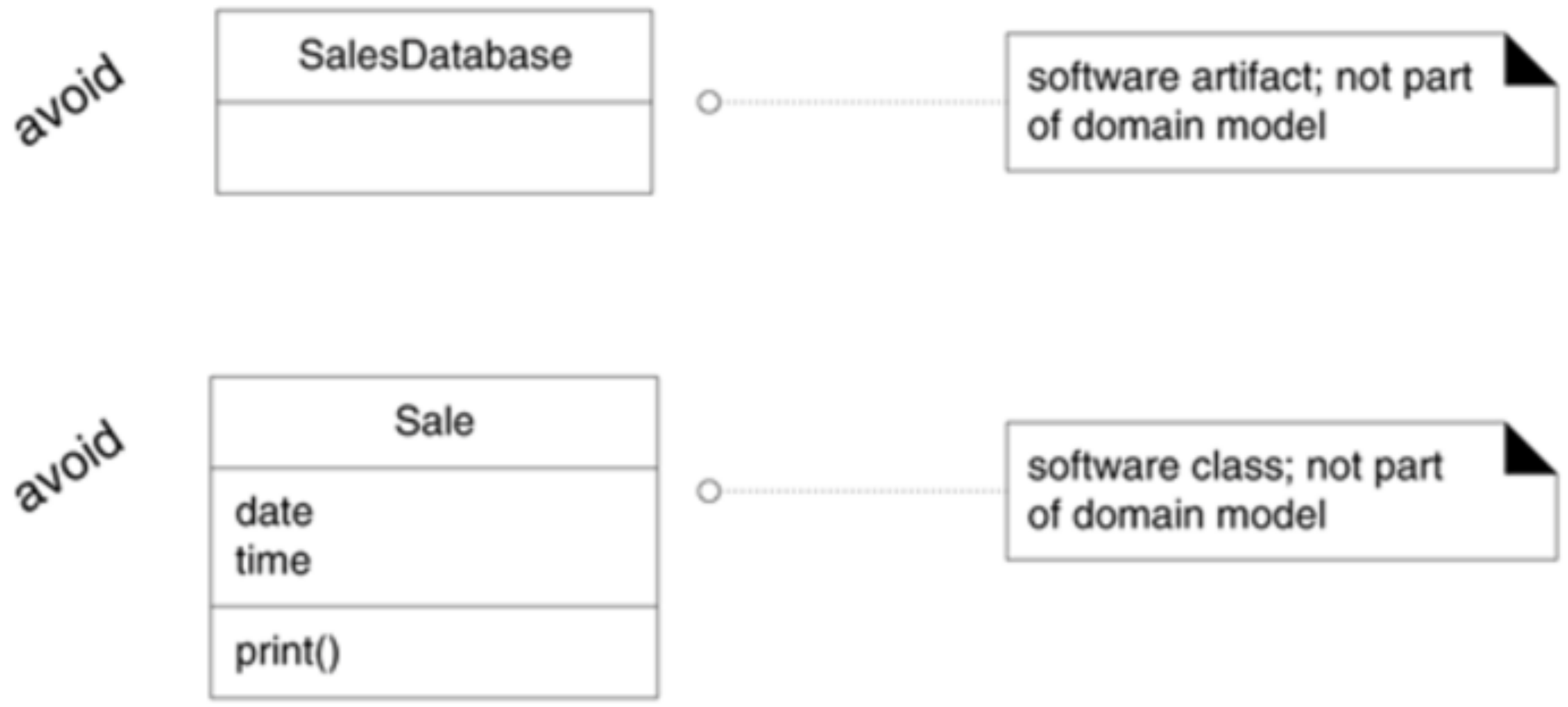
Domain model



Domain model - Things to avoid

- Describe software artifacts like Window, Database
- Specify method to a model
- See examples

Domain model - Things to avoid


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

Domain model - conceptual class

- Symbol - words or images representing the model
- Intension - the definition of the model
- Extension - the set of examples to of the model

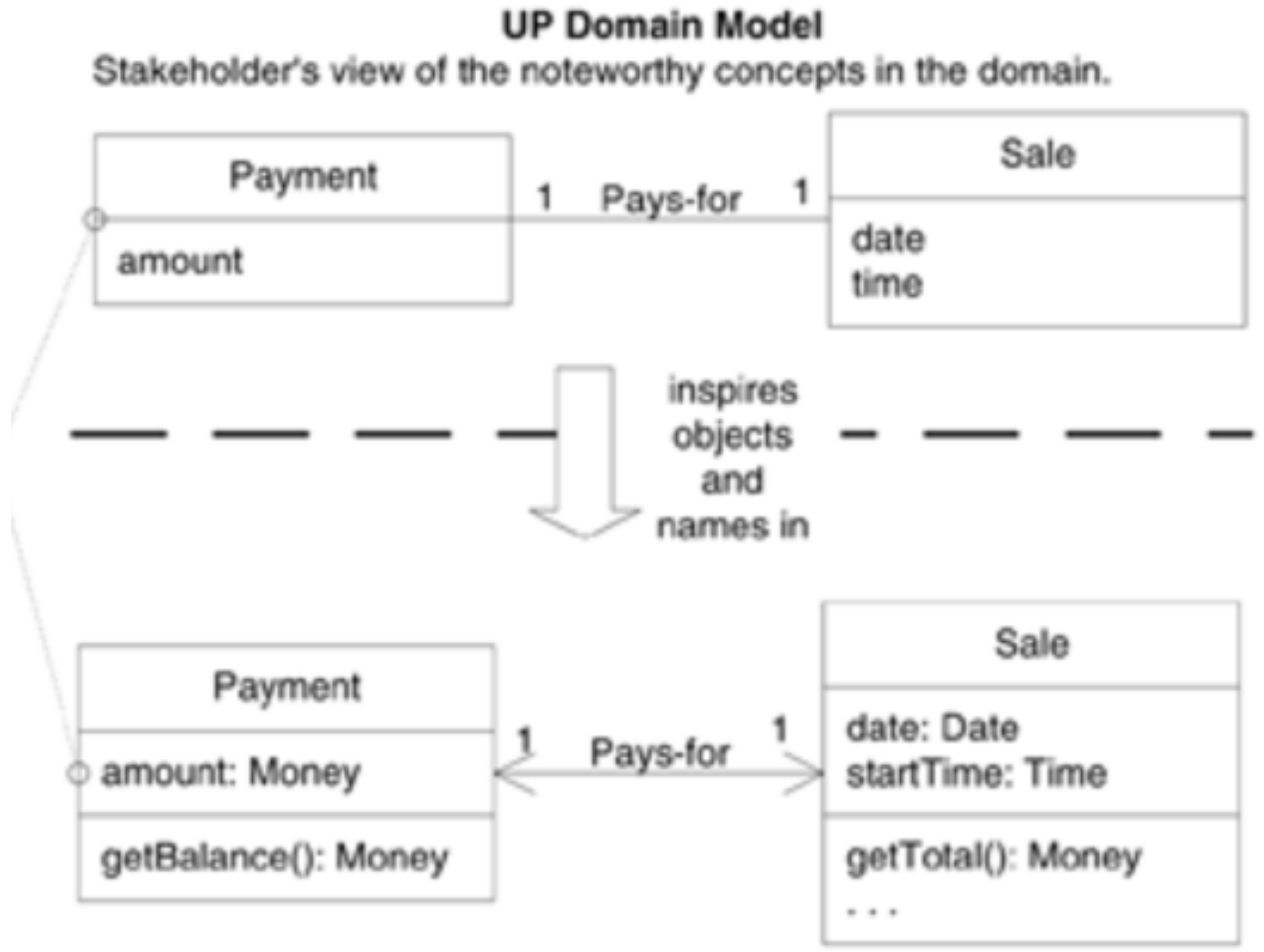
Domain model - conceptual class

- Symbol - Sale
- Intension - A sale represent the event of a purchase transaction. It has a date and time
- Extension - { sale-1, sale-2, sale-3 }

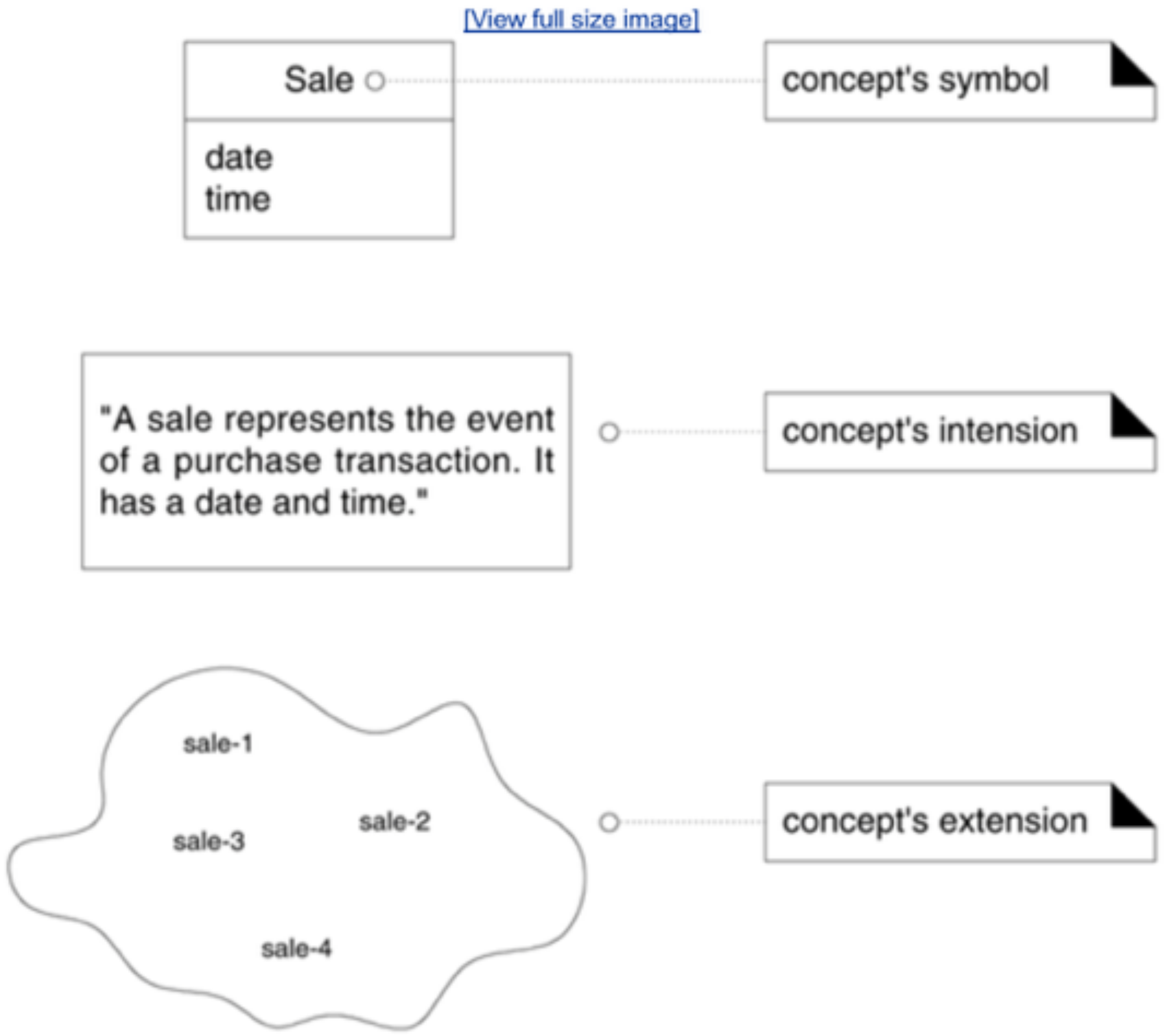
Why create a domain model?

- To understand key concepts of the business
- Get the big picture without worrying about the software details
- Domain model acts as inspirational to create software classes

Why create a domain model?



Domain model - conceptual class



Domain model - From use case

Preconditions: Cashier is identified and authenticated

Postconditions: Sale is saved. Tax is correctly calculated. Account and inventory updated. Commission recorded. Receipt is generated. Payment authorization approvals are record

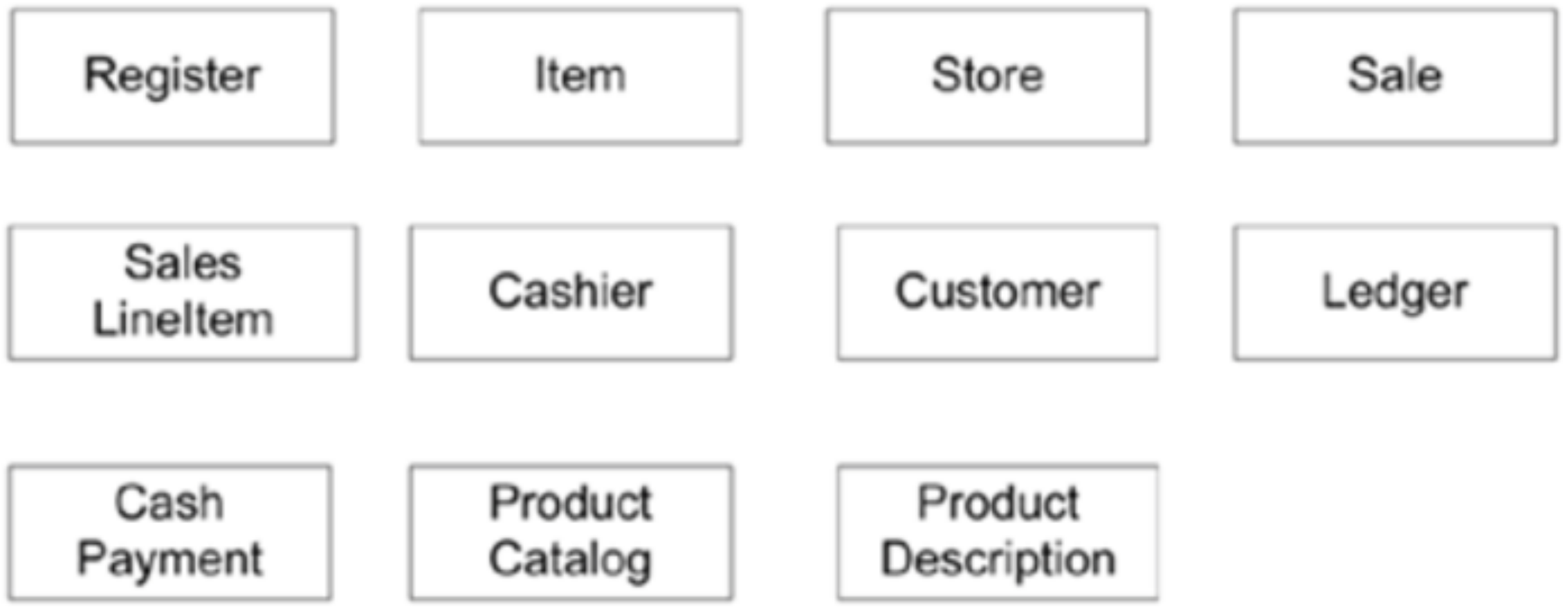
Main Success Scenario:

1. Customer arrives at POS with items
2. Cashier starts a new sale
3. Cashier enters item id
4. System records sale line item and present item description, price, total

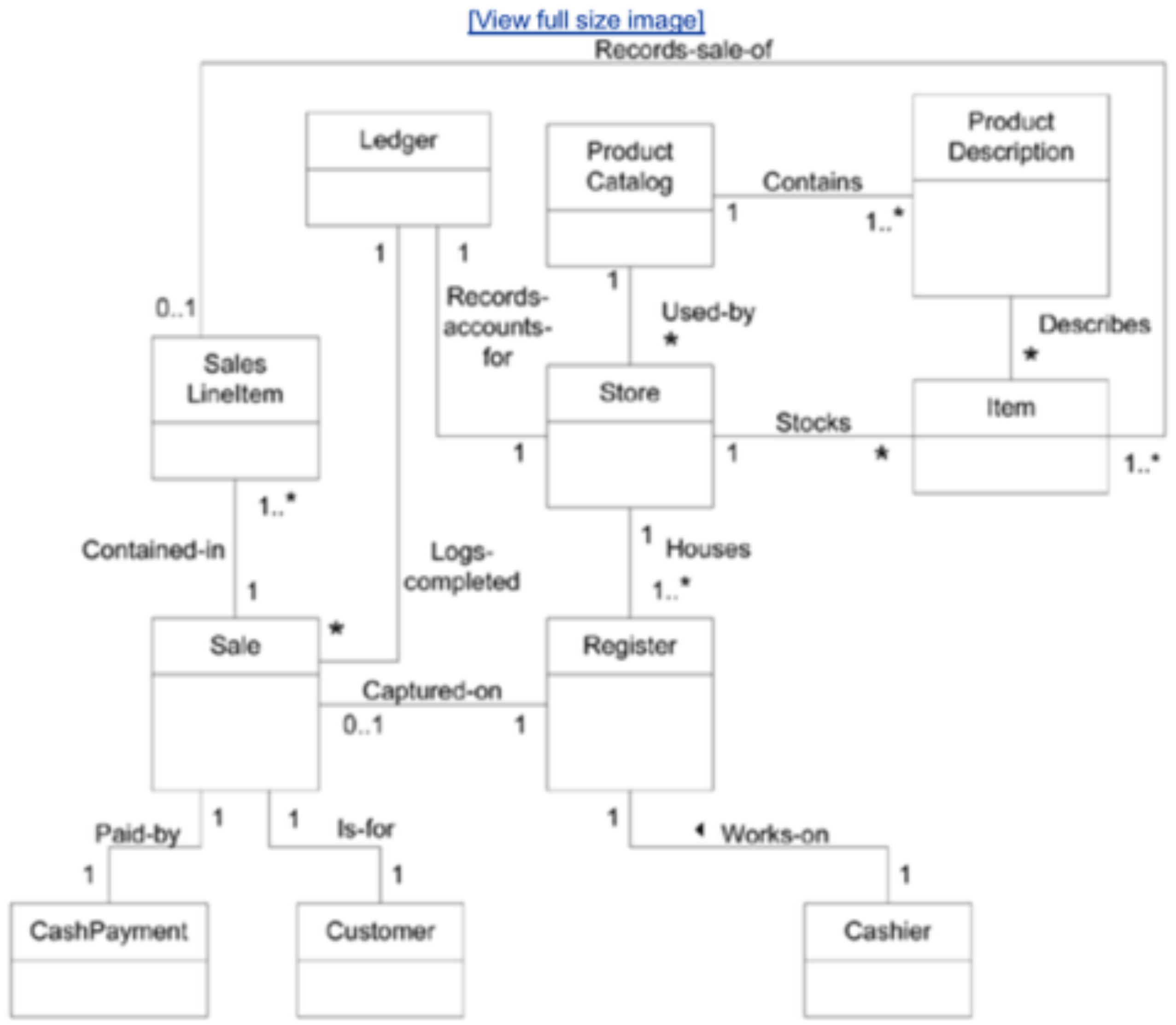
- - - Cashier repeat steps 3-4 until done

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 completed sale and sends sale and payment information to the external Accounting and inventory system
9. System presents receipt
10. Customer leaves with items

Domain model - candidates



Domain model - POS



Domain model - Attributes vs classes

- If that thing is raw number or text in the real world it might be an attribute
- In the previous model, What is store?

Description class

- Contains a information that describe something else
- If we don’t have description class, what happen when items are sold out?
- Reduce redundancy

Association

- When to show association?
- Why too many association is bad?
- Will the association be implemented in software?
- See examples

Association

- How should we name association?
- Has and Use are not very good.
- Sale ‘Use’ CashPayment => Bad
- Sale ‘Paid-by’ CashPayment => Better

Association

- Multiplicity, see examples
- Multiple associations are also possible

Attributes

- When to show attributes?
- No foreign keys

Command Pattern

- Let’s see example in together