SWEN30006
Software Design and Modelling

Domain Models &
System Sequence Diagrams

Textbook: Larman Chapter 1, 9 & 10

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

— anonymous management maxim

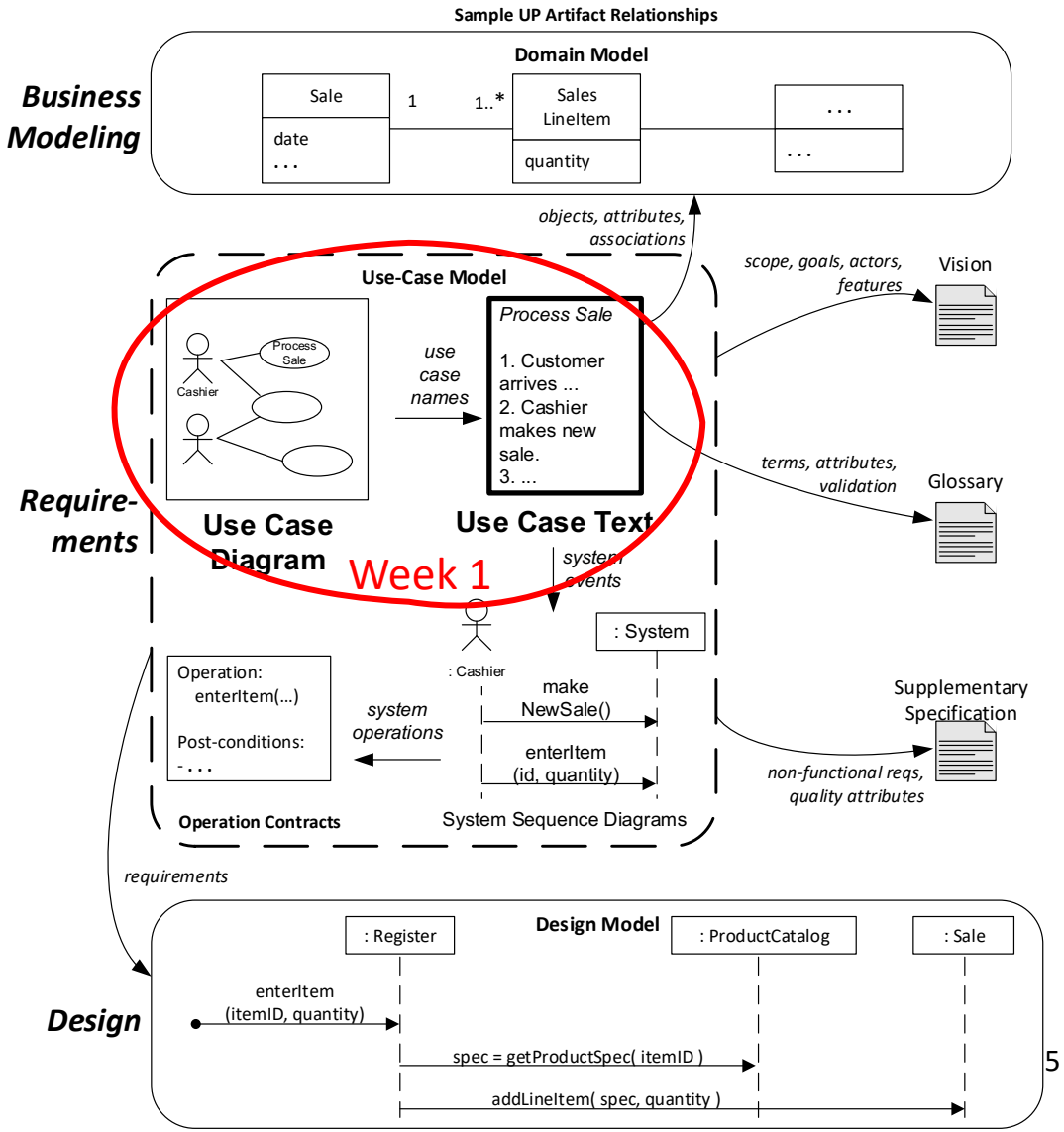
Learning Objectives

On completion of this topic, you should be able to:

- State the purpose of OO Analysis.
- Be familiar with static and dynamic OO models
  - Domain Class Diagrams
    - Identify conceptual classes
    - Create an initial domain model
    - Model appropriate attributes and associations
  - System Sequence Diagrams (SSD)
    - Identify system events
    - Create SSDs for use case scenarios

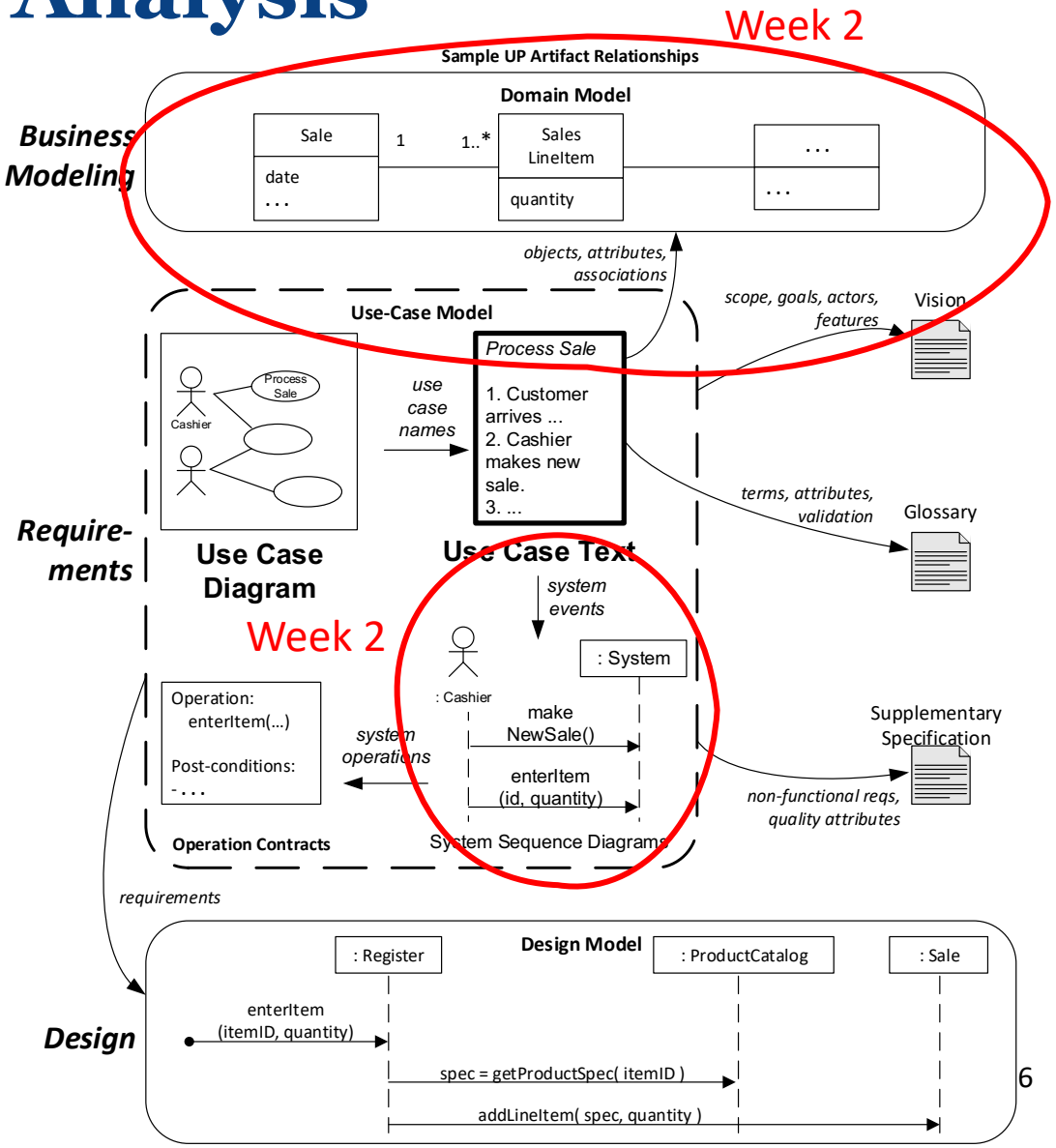
Week 1 Revisited: Use Cases

- **Use Cases** are text stories that describe how an actor *use* a system to meet goals
  - A tool to discover and record requirements
- **Use case diagrams** provides a context of a system and its environment
  - A tool to capture the names of use cases, actors, and their relationships



Week 2: Object-Oriented Analysis

- Week 2 focuses on object-oriented analysis based on use case text
- *Analysis* is investigation of the problem & requirements
- ***Object-oriented analysis*** is a process of creating a description of the domain from the perspective of *object*.
  - Analyze use cases and identify objects – or concepts – in the problem domain
  - The concepts and behaviours are captured in the form of *Domain Models* and *Sequence Diagrams*



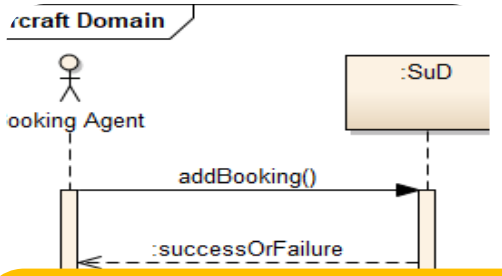
OO Analysis, Design, and Development

Problem domain

- Concepts
- Relationships

Use Cases

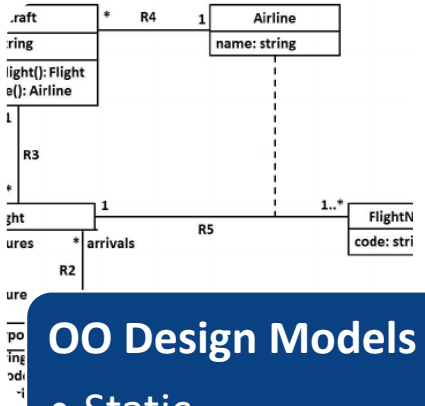
Text stories



OO Domain Models

- Static: Domain Class Diagram
- Dynamic: System Sequence Diagram

Conceptual Classes



OO Design Models

- Static
- Dynamic

Software Classes

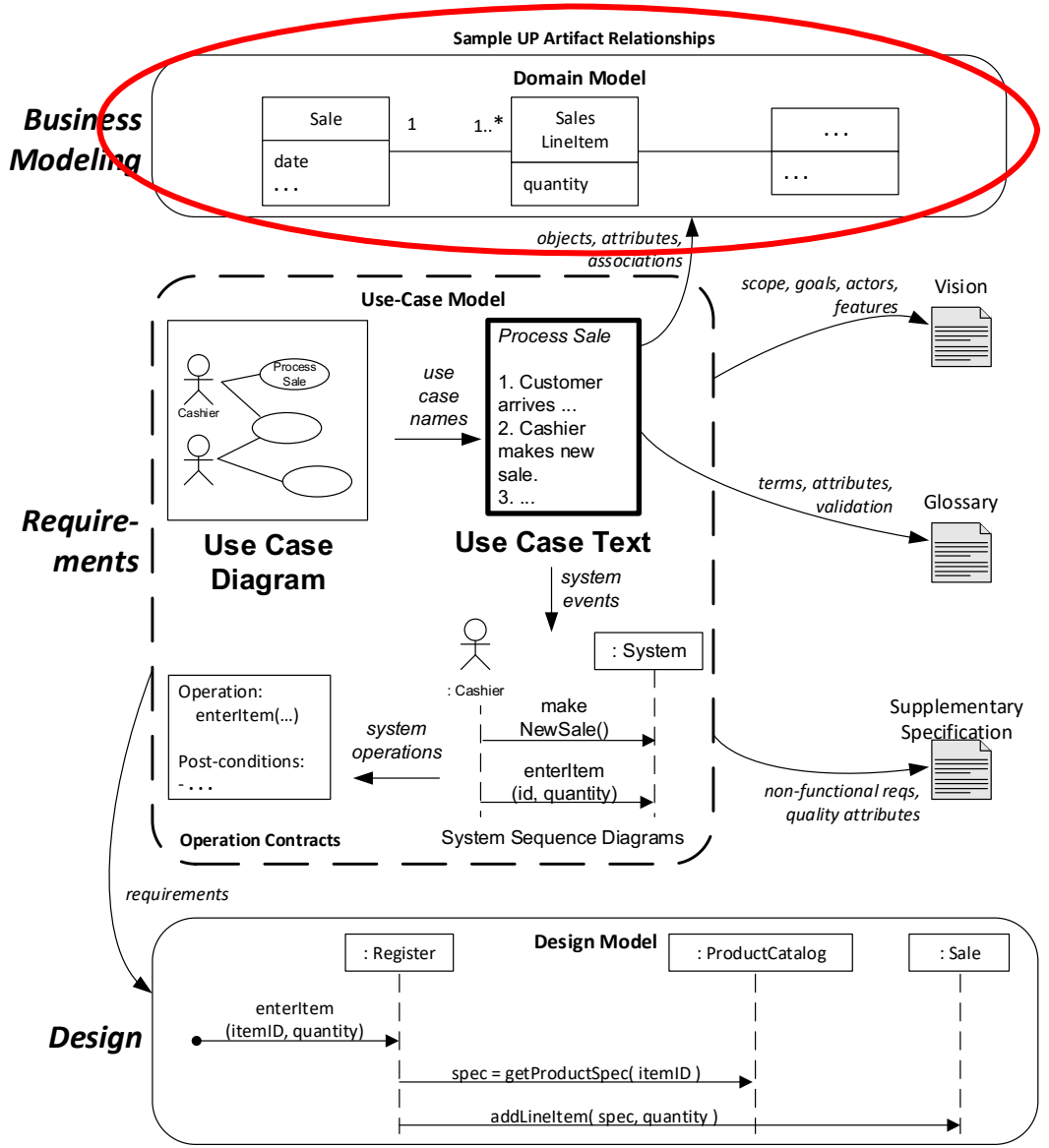
```
do_important_stuf
teZInputs();
t = zalg.apply_ZAlg(
stZOutput(zoutput))
eZOutput();
{
```

OO Programming

Classes in Programming

Domain Models

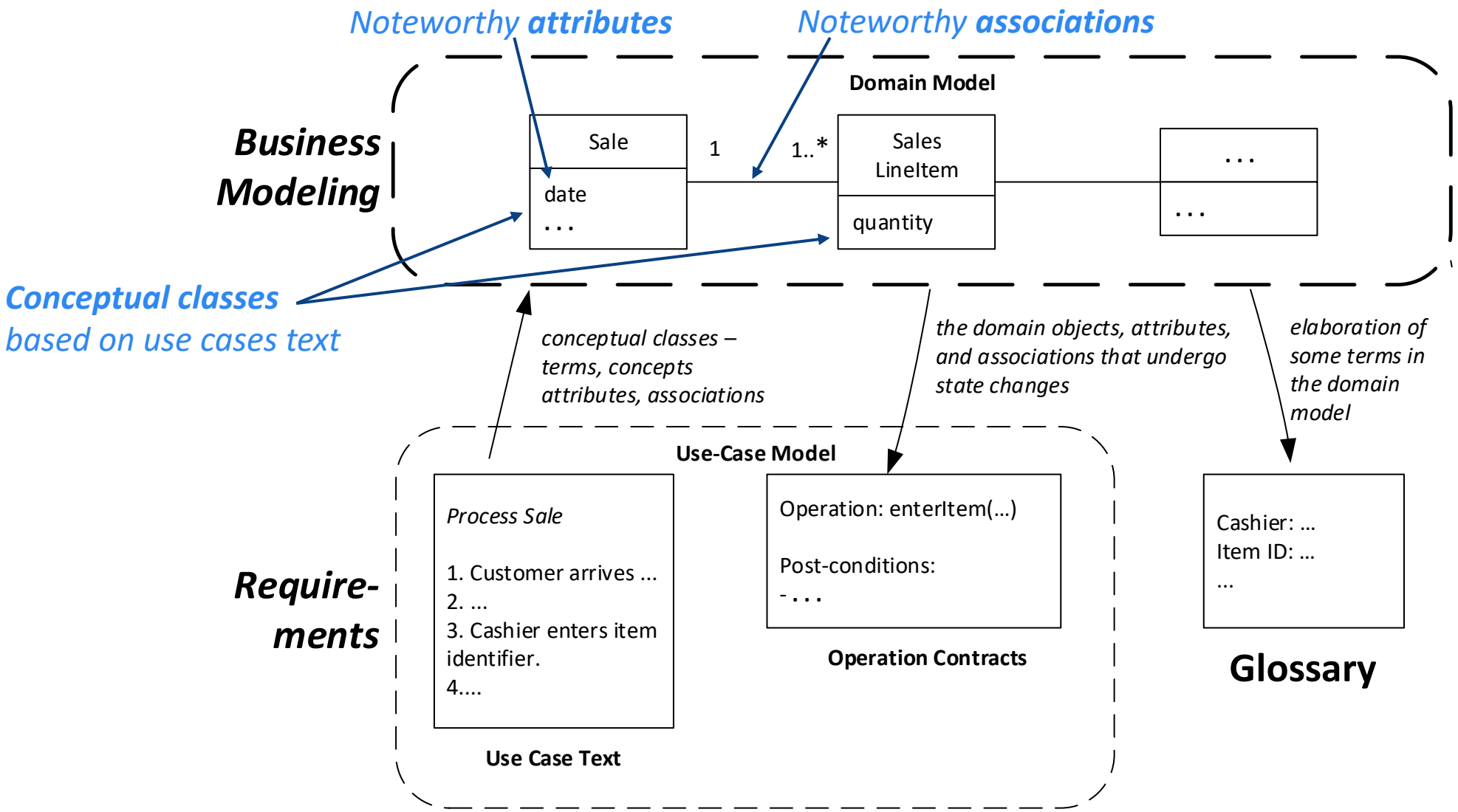
Textbook: Larman Chapter 9



Domain Models

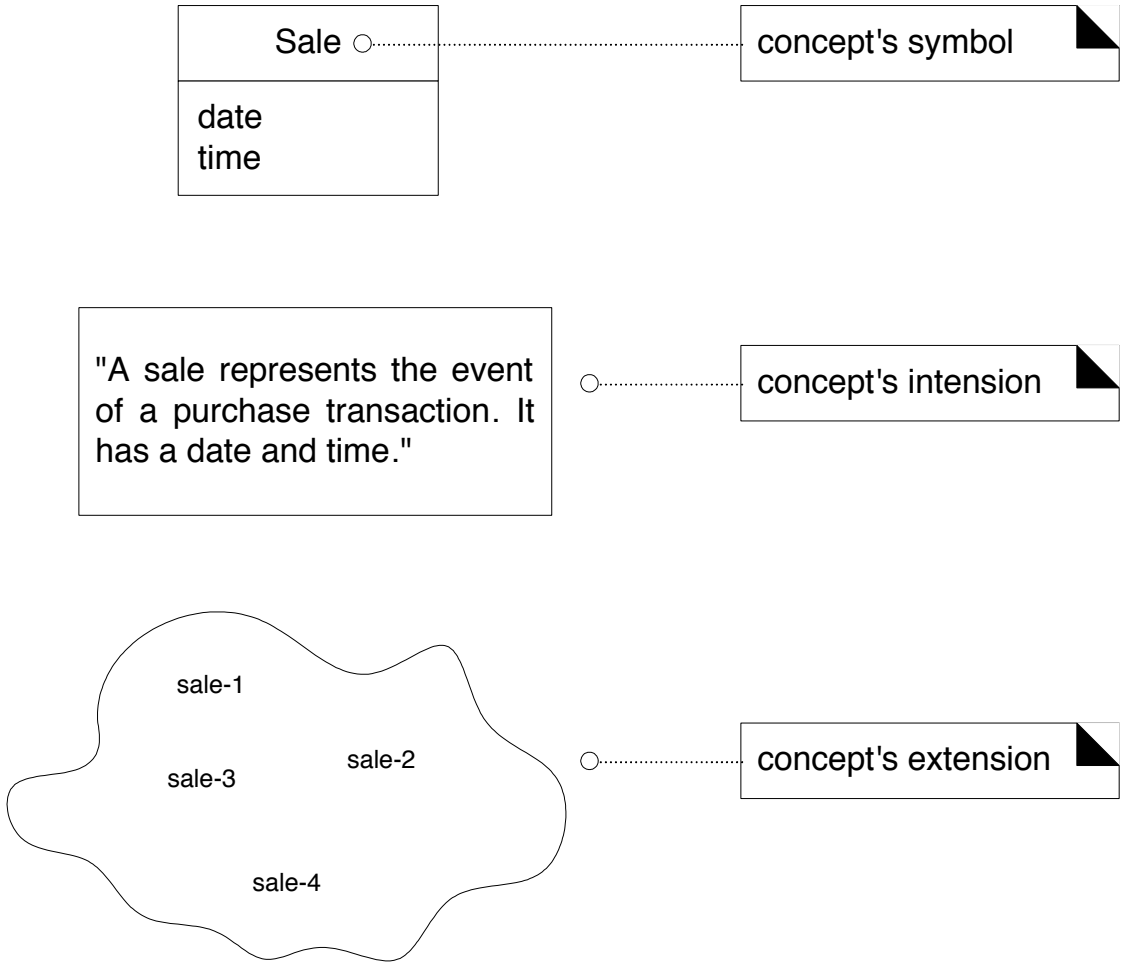
- OO Analysis includes an identification of the **concepts**, **attributes**, and **associations** that are considered noteworthy in the problem domain – These can be expressed by *a domain model*
- **Definition:** A domain model is a representation of *real-situation* conceptual classes
  - *Not* a software object
  - Shows the *noteworthy* domain concepts or object
  - One of OO artifacts
- The UP domain model = UP Business Object Model
  - Focus on explaining *things* and products important to a business domain
- **Visual** representation: UML class diagram is used to illustrate a domain model
  - A domain class diagram provides a *conceptual* perspective to show conceptual classes, their attributes and associations with other classes
  - *No* operations (method signatures) are defined in the class diagram

Domain Models in UP



A Conceptual Class: Definition

- **(Informal) Definition:** A conceptual class is an idea, thing, or object
  - It can be more than a data model. It can have a purely behavioural role in the domain instead of an information role
- **(More formal) Definition:** A conceptual class may be considered in terms of its symbol, intension, and extension
  - **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.



A Conceptual Class: How to identify it?

Three strategies to find a conceptual class from use cases

1. Identify based on the existing models for common domain problems, e.g., inventory, finance, etc.
   - Books: *Analysis Patterns by* Martin Fowler, *Data Model Patterns* by David Hay
2. Use a category list
   - See the conceptual class category list in Table 9.1 in the textbook (Larman Chapter 9)
3. Identify noun phrases
   - Care must be applied as words in natural languages are ambiguous
   - Do not use as a mechanical noun-to-class mapping

Conceptual Class Category	Examples
Business transactions Guideline: Critical (involve money)	Sale, Payment Reservation
Physical objects Guideline: esp. for device-controllers or simulations	Item, Register Board, Piece, Die Aeroplane
Containers of things (physical or information)	Store, Bin Board Aeroplane
Where are transactions recorded? Guideline: important	Register, Ledger FlightManifest, MaintenanceLog

Example: POS

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.

Identify conceptual classes based on a category list

- Role of people or organization:
  - Customer, Cashier
- Product or services:
  - Goods and/or services (rename to item)
- Business transaction:
  - Sale
- Transaction line items:
  - Sale line item
- Description of things:
  - Item description
- Where is the transaction recorded?
  - A POS checkout (Register)

Identify conceptual classes (Click the potential terms in the image)

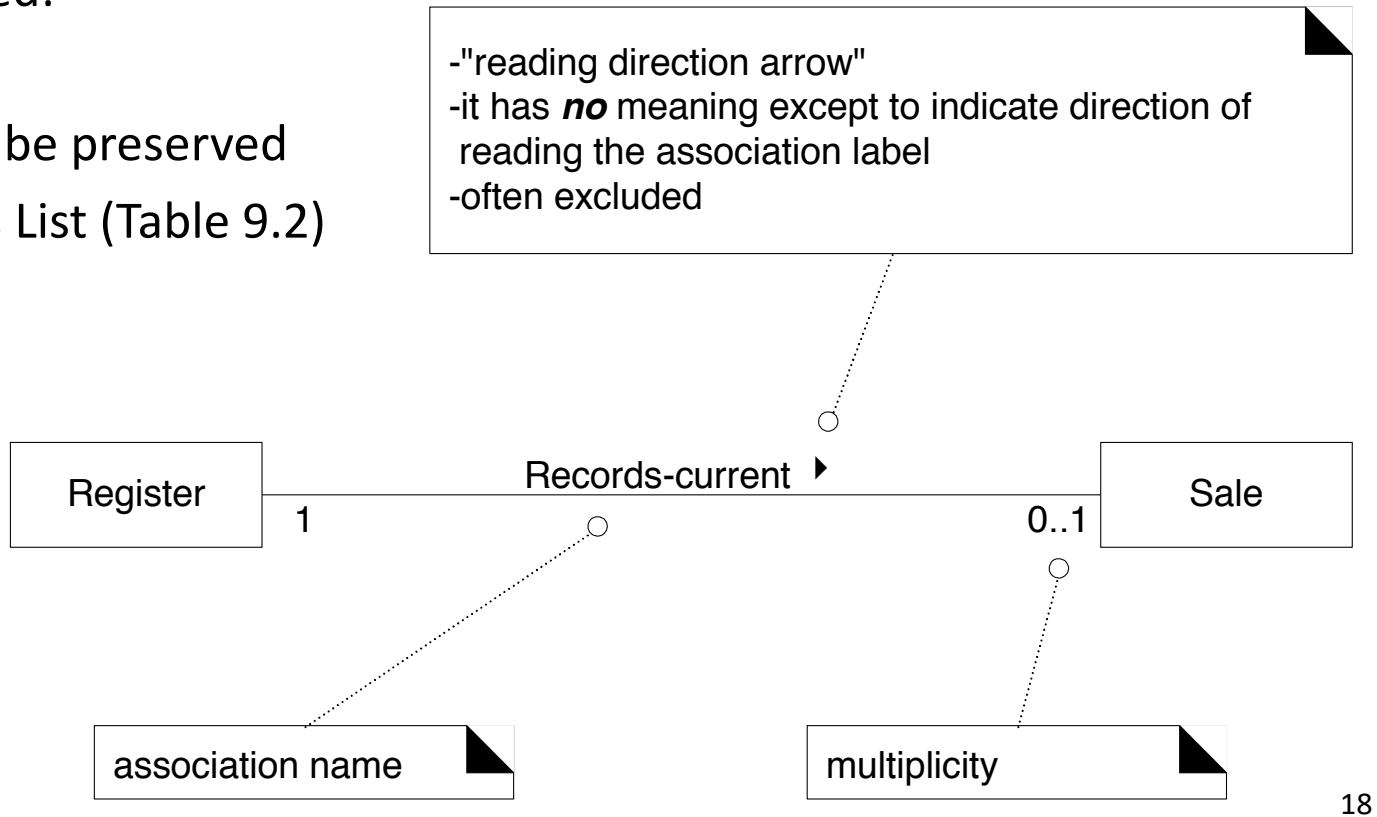
- The library has users who can borrow items.
- The library holds both books and videos: all items have a title and year of production (and a due date if they have been borrowed), while only books have authors and only videos have a duration
- There are different kinds of users: students, staff, and guests. The only difference between these user types is that they have a different borrowing limit (max number of items they can borrow) which is based on their user type.
- The library keeps track not only of which items are out on loan now, but the complete history of which users borrowed which items for which periods.

Example: Conceptual classes

- The library has **users** who can borrow **items**.
- The library holds both **books** and **videos**: all items have a title and year of production (and a due date if they have been borrowed), while only books have authors and only videos have a duration
- There are different kinds of users: students, staff, and guests. The only difference between these user types is that they have a different borrowing limit (max number of items they can borrow) which is based on their user type.
- The library keeps track not only of which items are out on **loan** now, but the complete **history** of which users borrowed which items for which periods.

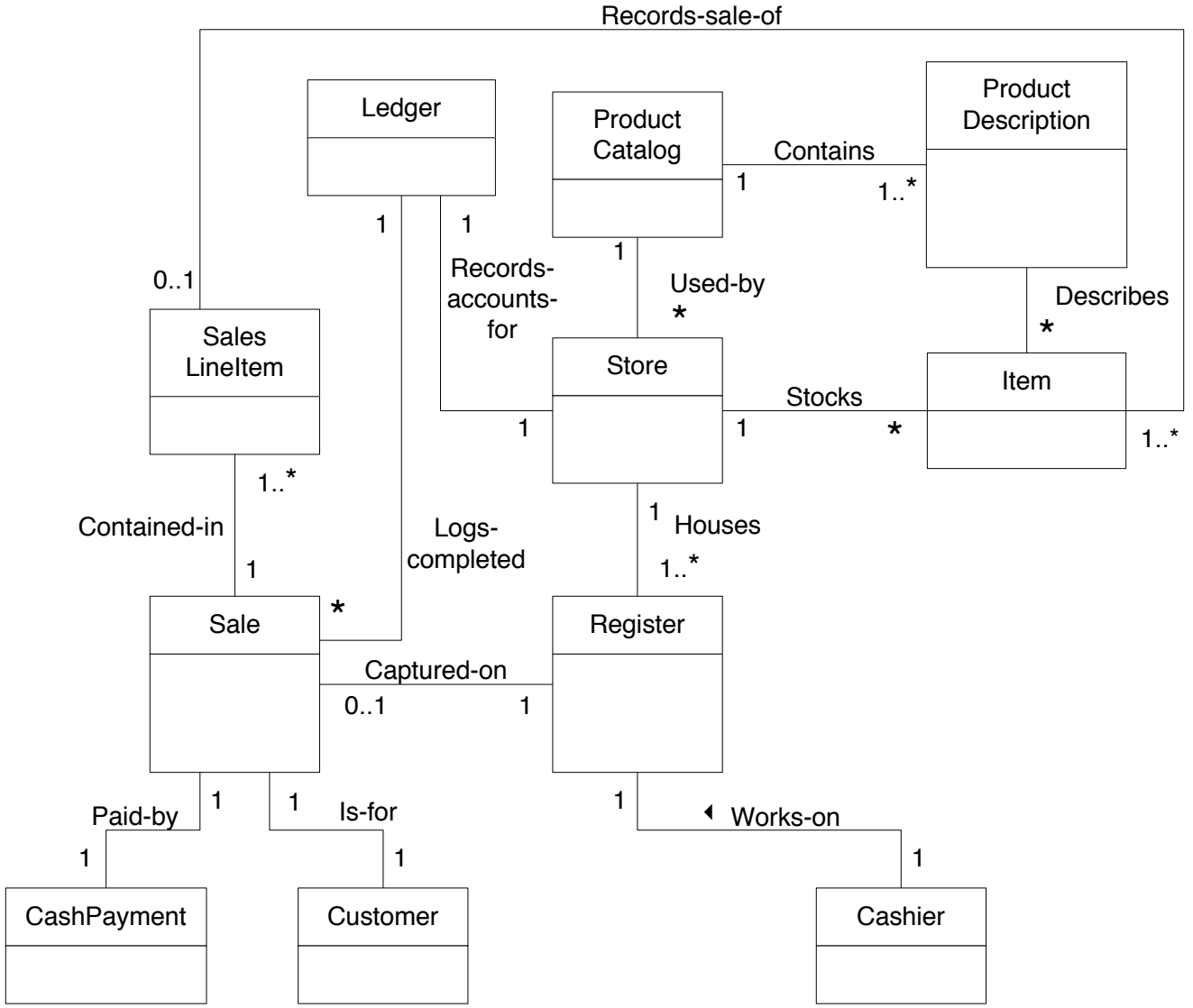
Associations between Conceptual Classes

- **Definition:** An *association* is a relationship between classes that indicate some meaningful and interesting connection
- The following associations should be included:
  - Significant in the domain
  - Knowledge of the relationship needs to be preserved
  - Derived from the Common Associations List (Table 9.2)



*See Worshop 2 supplementary materials

POS: Partial Domain Model

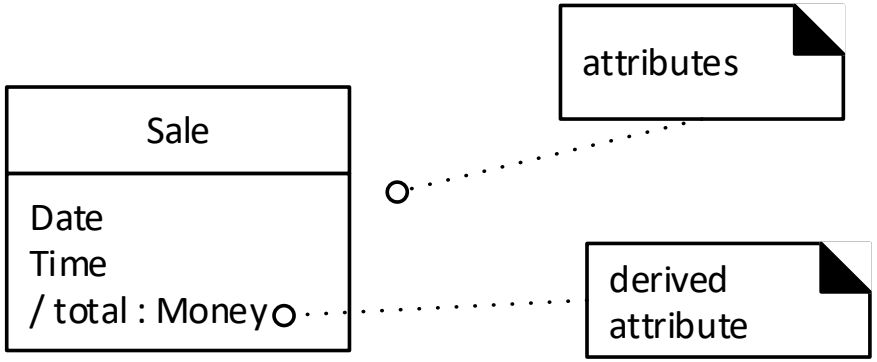


Attributes of Conceptual Classes

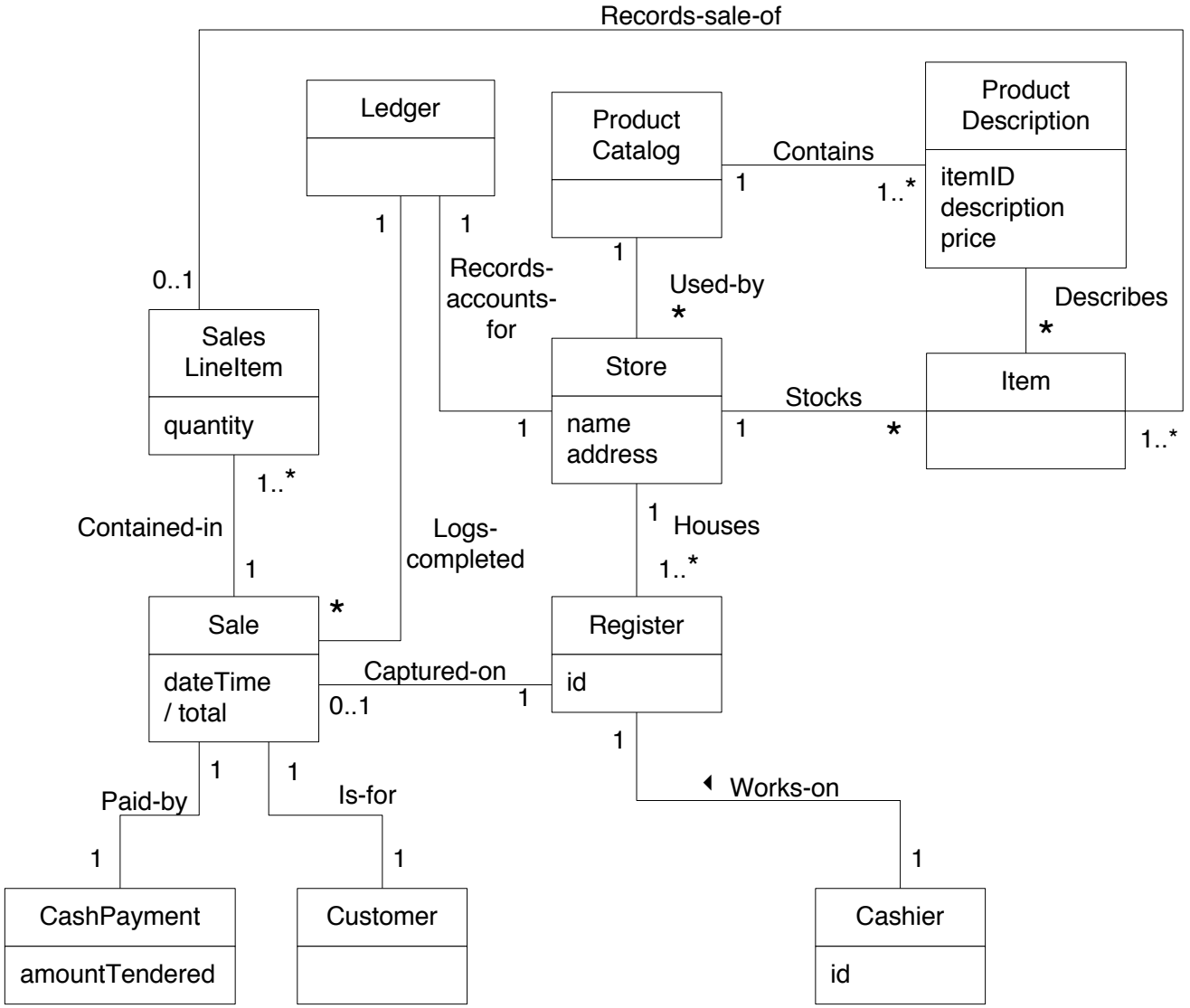
- **Definition**: An *attribute* is a logical data value of an object.
- Attributes should be included into conceptual classes when the requirements (e.g., use cases) suggest or imply *a need to remember information*

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.

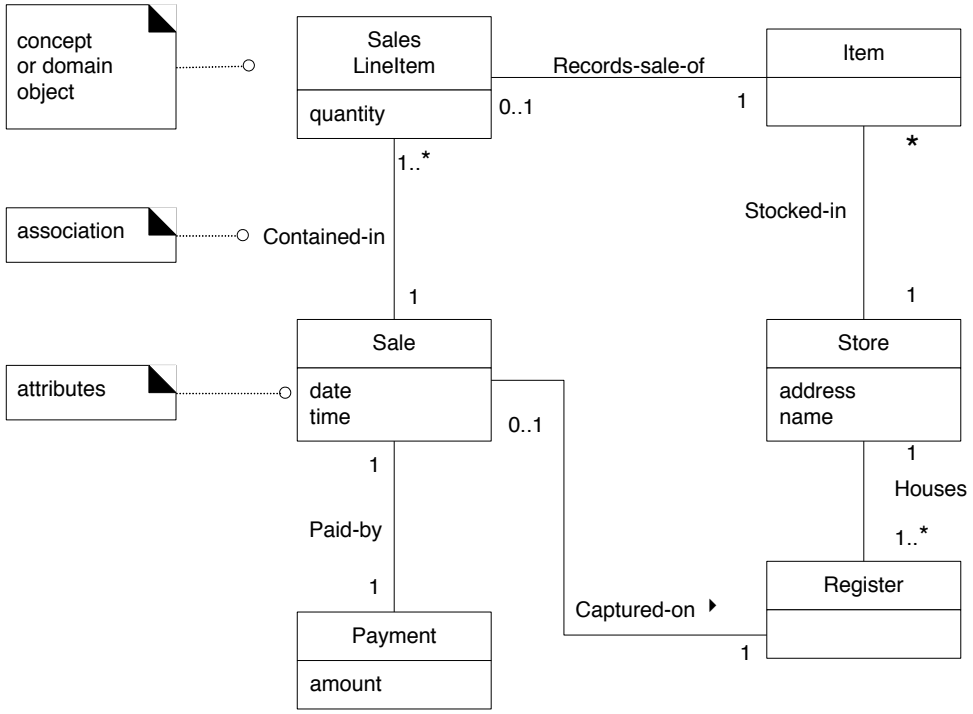


POS: Attributes in Model

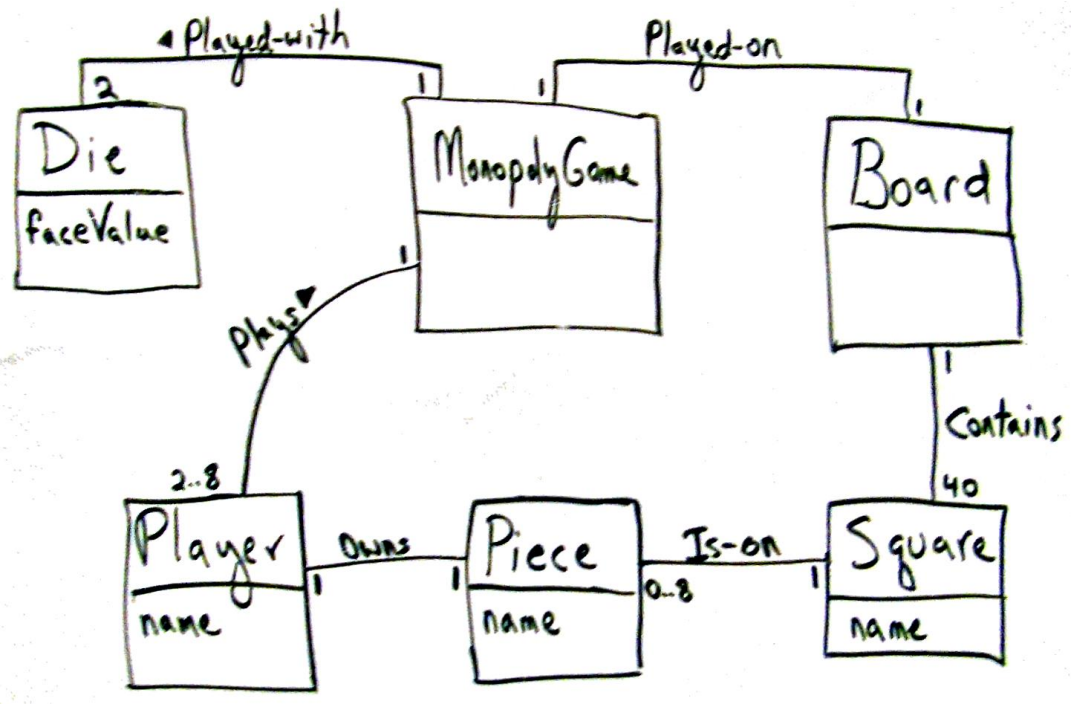


Domain Model: A Visual Dictionary

- The domain model is a *visual dictionary* of the noteworthy abstractions, domain vocabulary, and information content of the domain



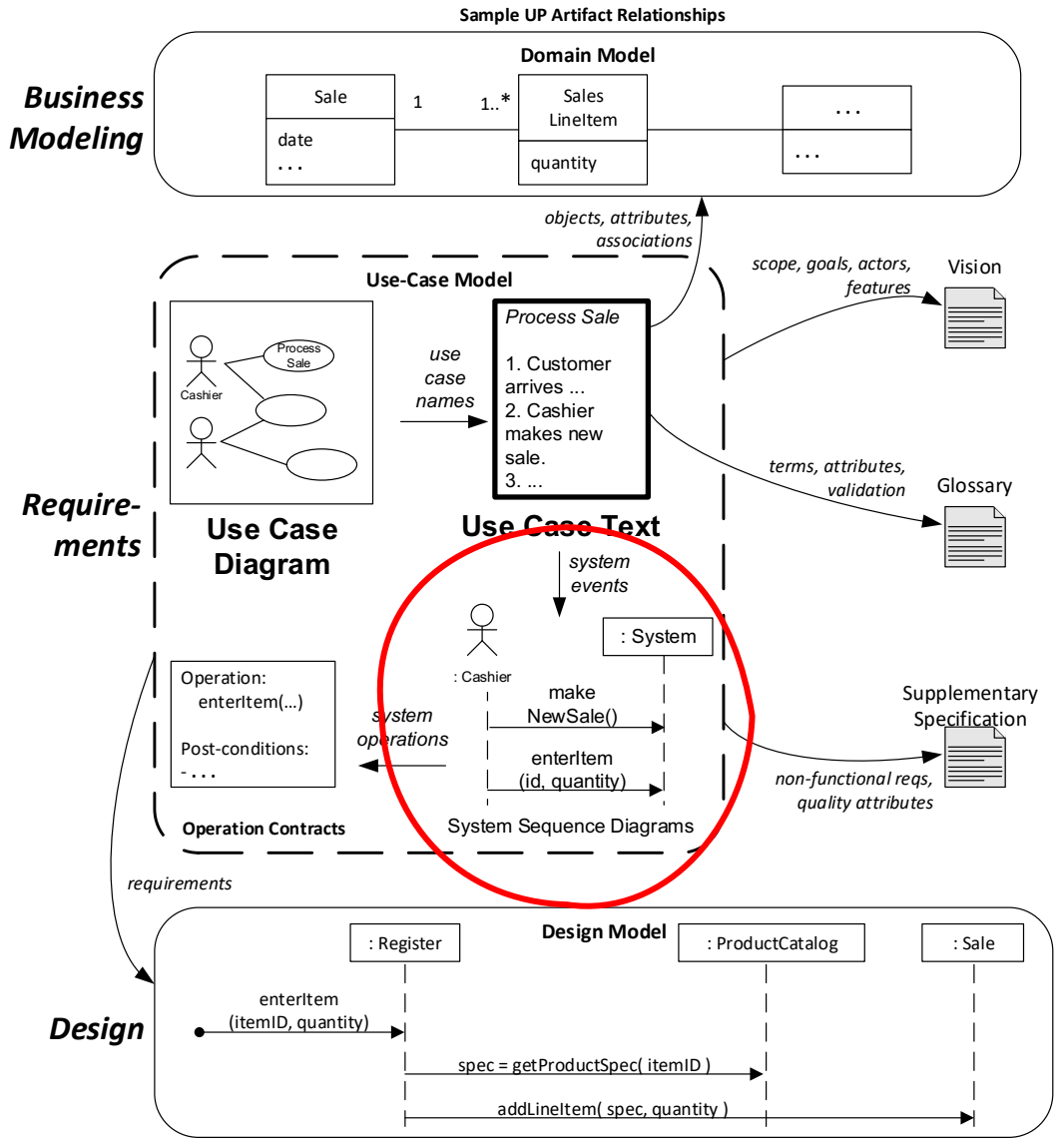
A partial domain model for POS



A partial domain model for Monopoly

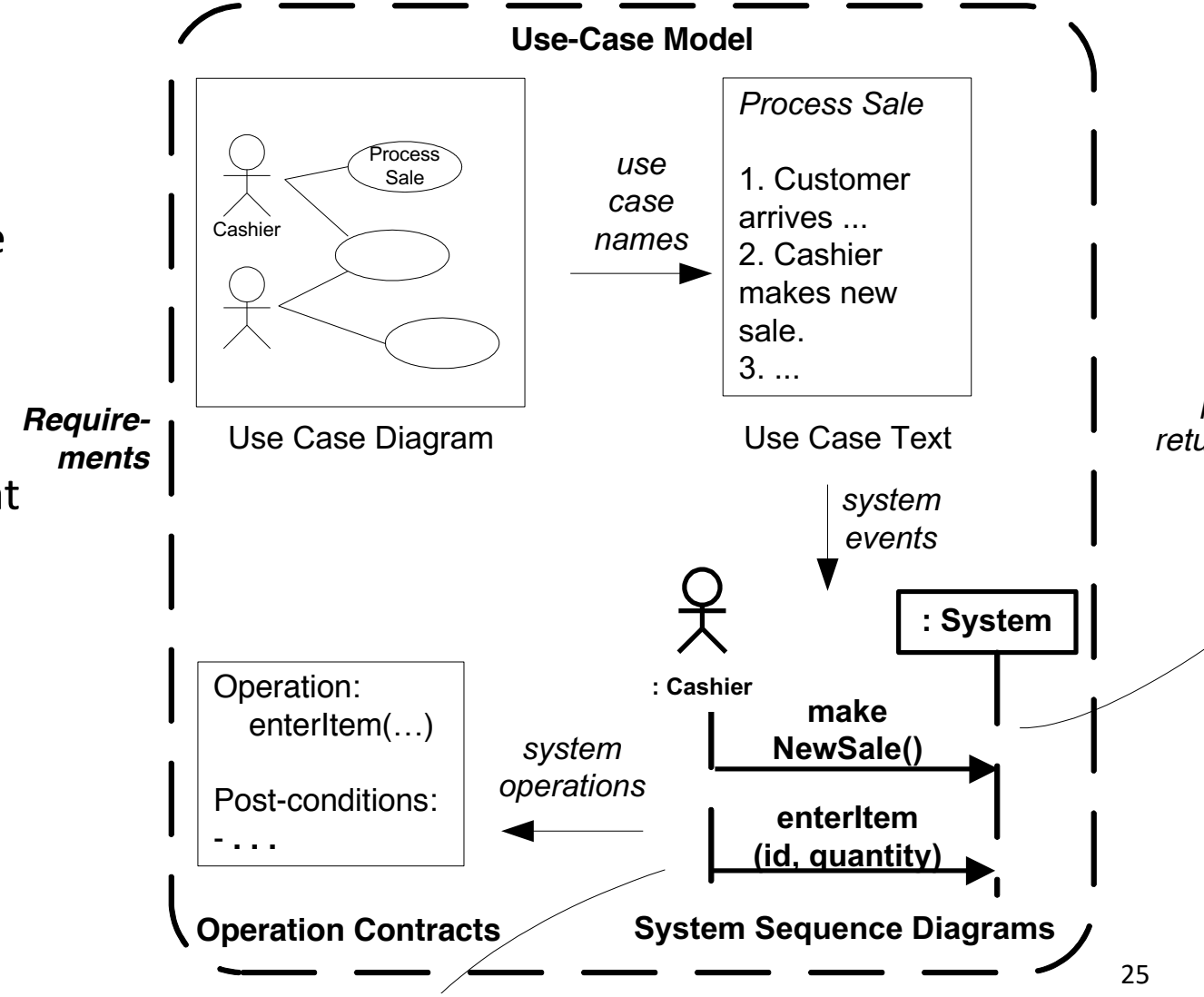
System Sequence Diagrams

Textbook: Larman Chapter 10



System Sequence Diagrams (SSDs)

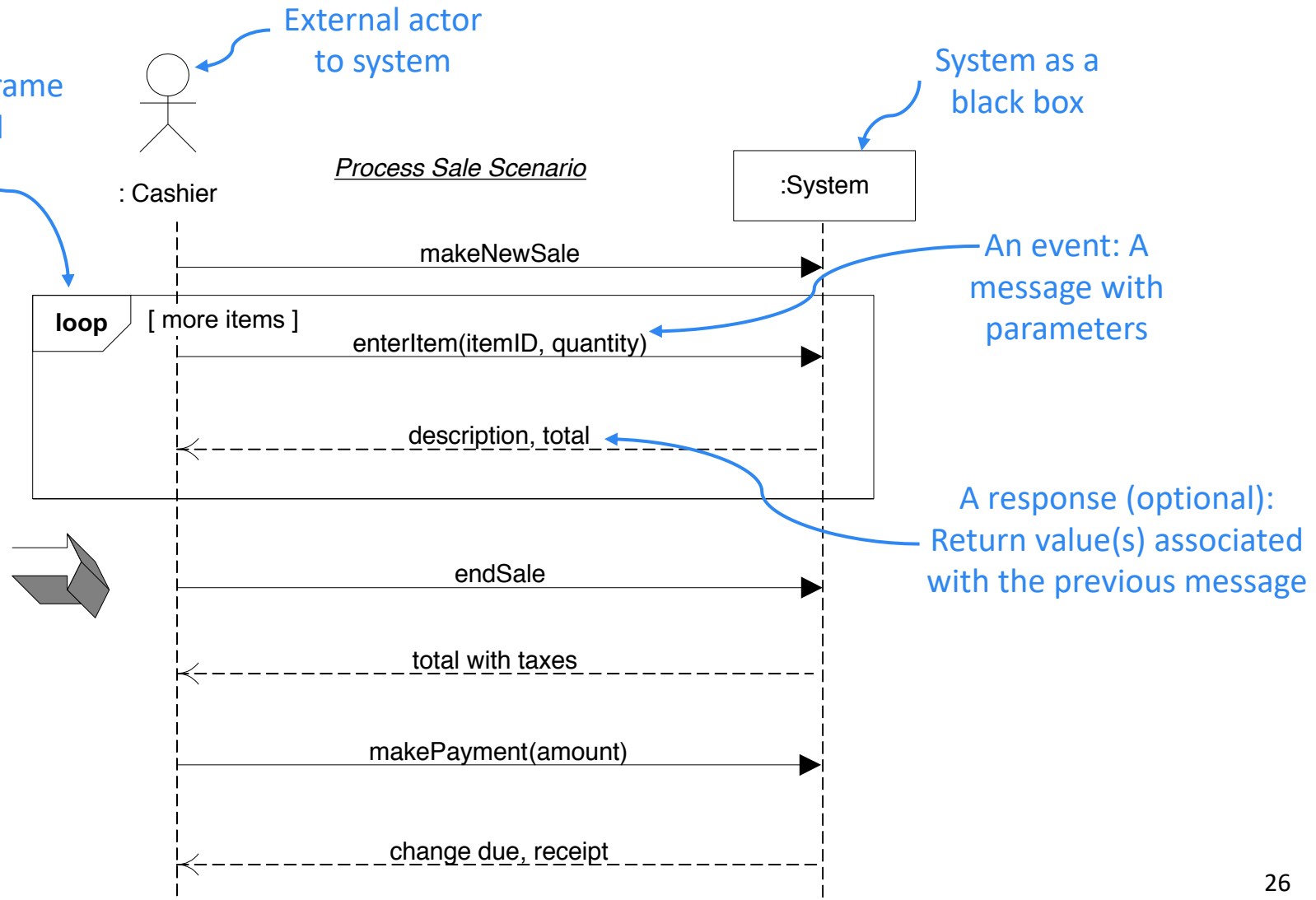
- **Definition:** A visualisation of the *system events* that external actors generate, the *order* of the events, and possible inter-system events
  - One SSD is for *one particular scenario* of a use case
  - Helps to identify the external input events to the system (the system events)
  - Treats system as a 'black box' to describe what the system does without explaining how it does it.



SSD: Process Sale scenario of POS use cases

Simple cash-only *Process Sale* scenario:

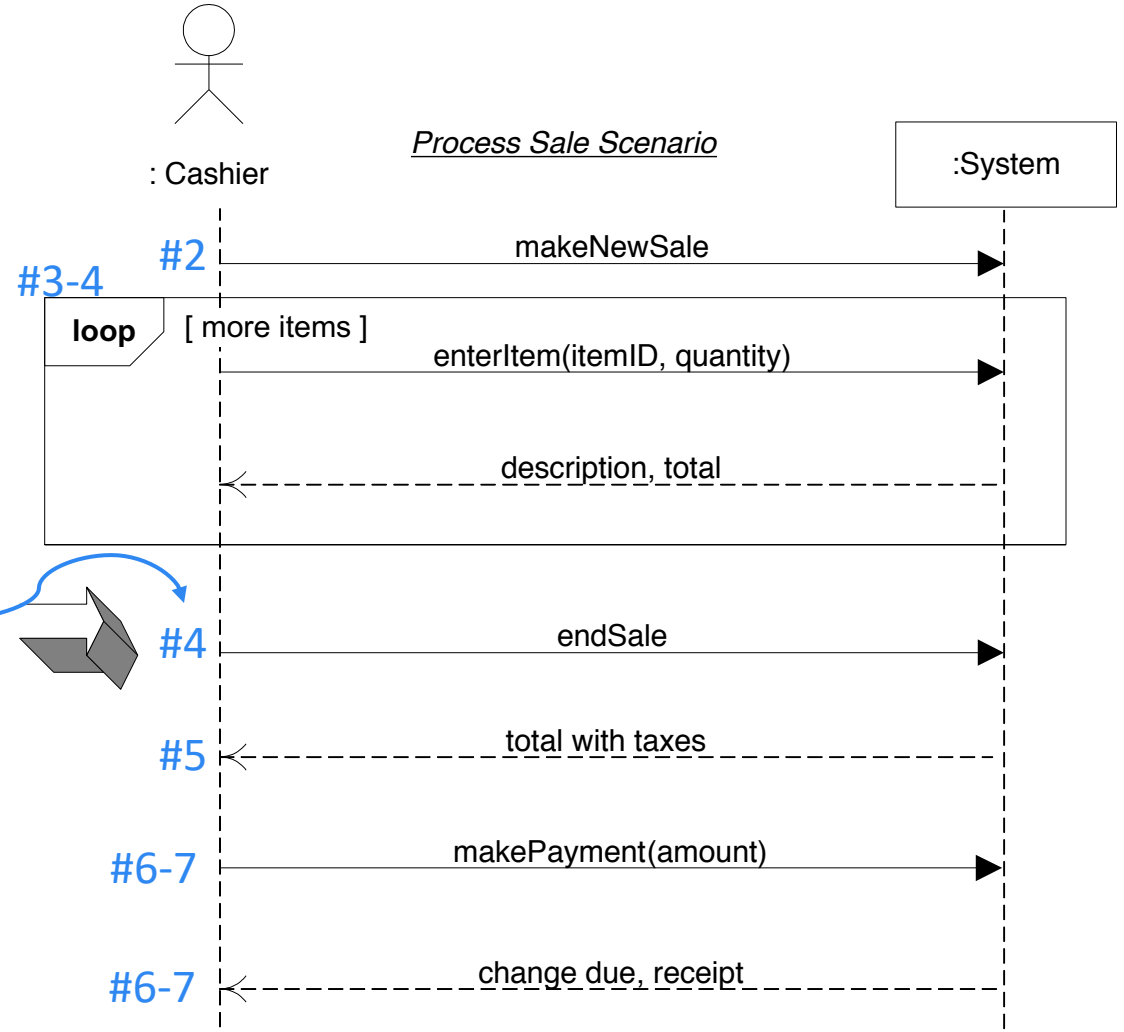
1. Customer arrives at a POS checkout with goods and/or services to purchase.
2. Cashier starts a new sale.
3. Cashier enters item identifier.
4. System records sale line item and presents item description, price, and running total.
Cashier repeats steps 3-4 until indicates done.
5. System presents total with taxes calculated.
6. Cashier tells Customer the total, and asks for payment.
7. Customer pays and System handles payment.
...



SSD: Process Sale scenario of POS use cases

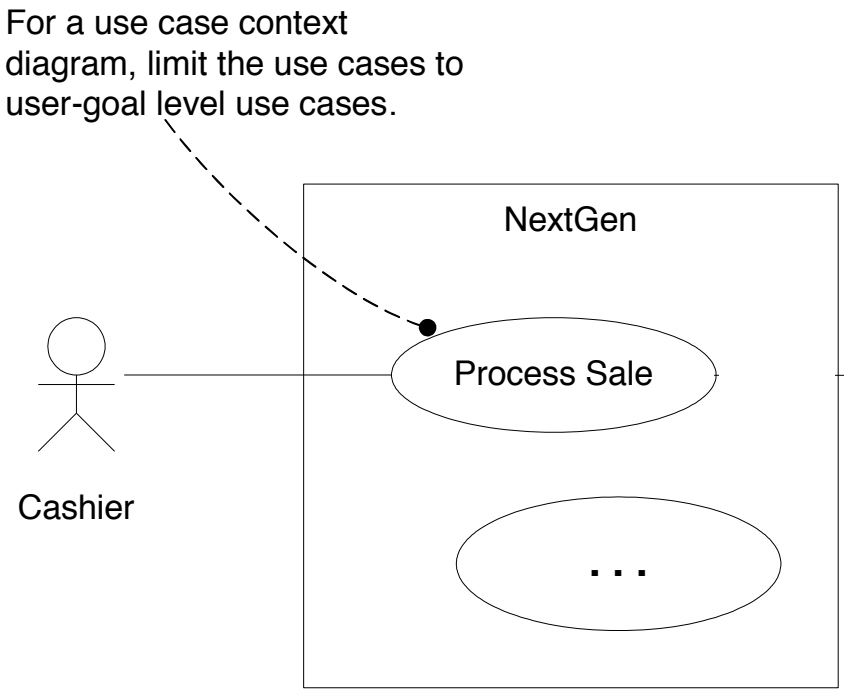
Simple cash-only *Process Sale* scenario:

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.
Cashier repeats steps 3-4 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.
...

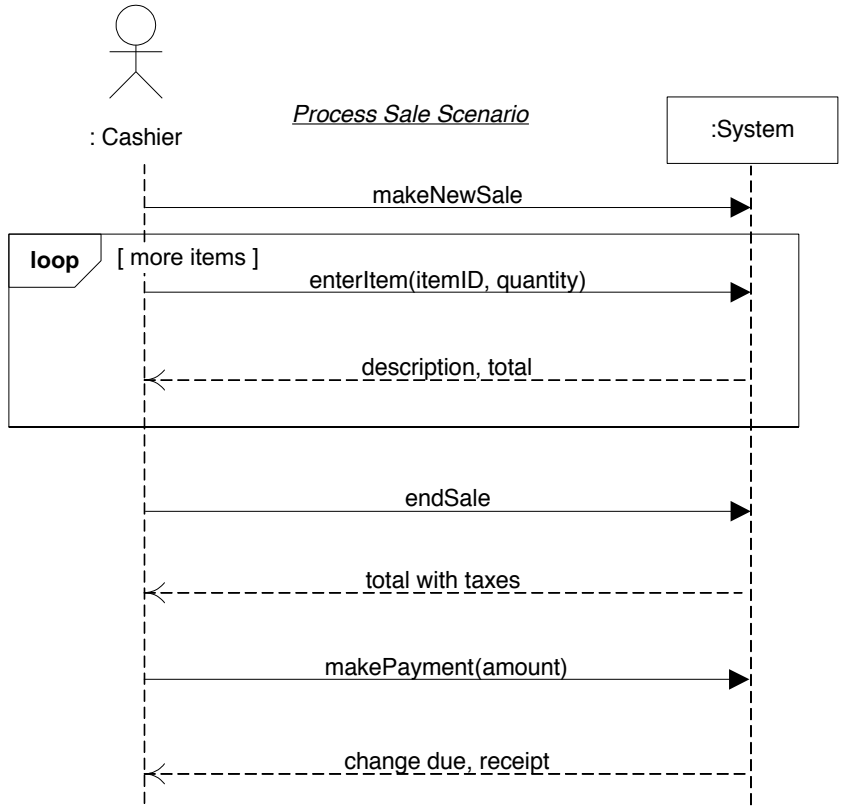


SSDs: Dynamic Context

- SSD captures dynamic context for system



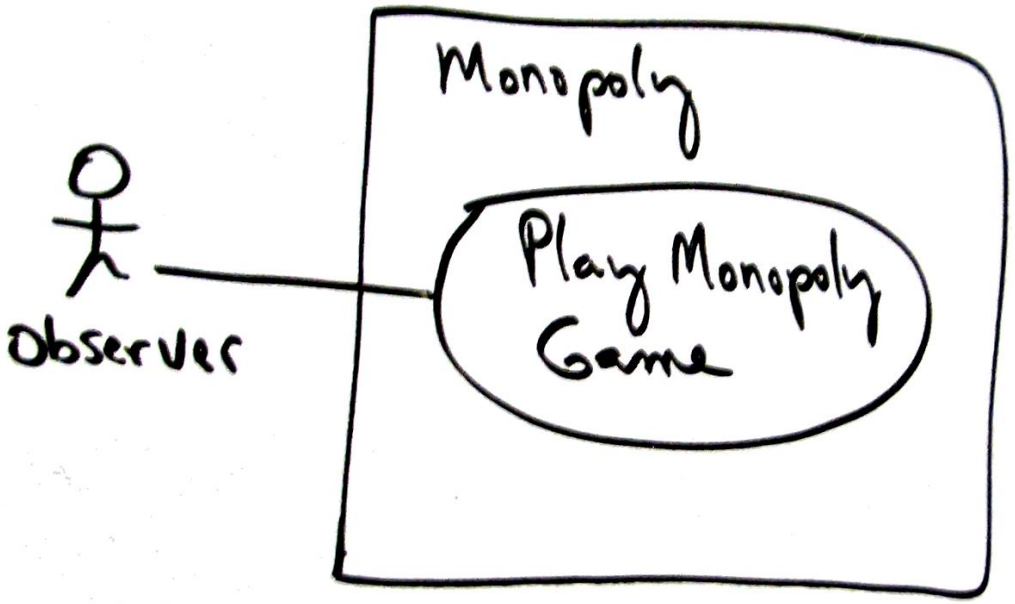
Use Case Diagram: POS



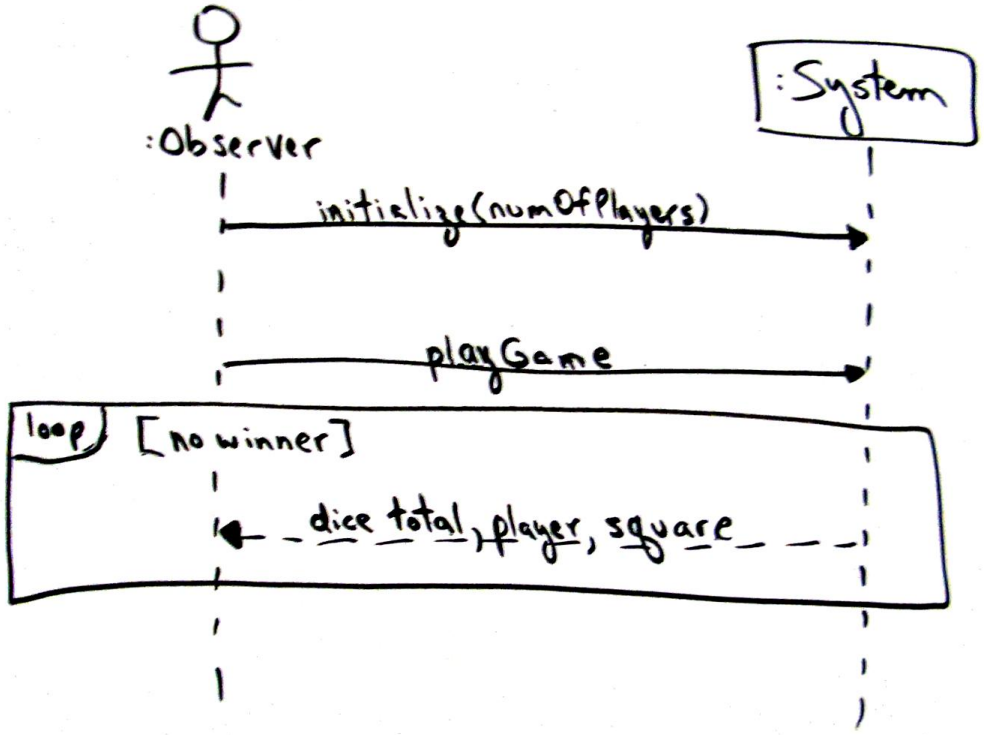
SSD for a Play Monopoly Game Scenario

SSDs: Dynamic Context

- SSD captures dynamic context for system



Use Case Diagram: Monopoly



SSD for a Play Monopoly Game Scenario

Exercise: SSD for Myki Quick Top-up Machine

Quick top-up scenario

1. A traveler taps a Myki card on the reader on the green panel. The machine display the balance of the card.
2. The traveler select the amount of top up from the displayed menu.
3. Then, the traveler taps a credit card to pay.
4. A traveler then taps the Myki card again to load the top up amount into the card. Finally, the machine displays the current balance of the card.

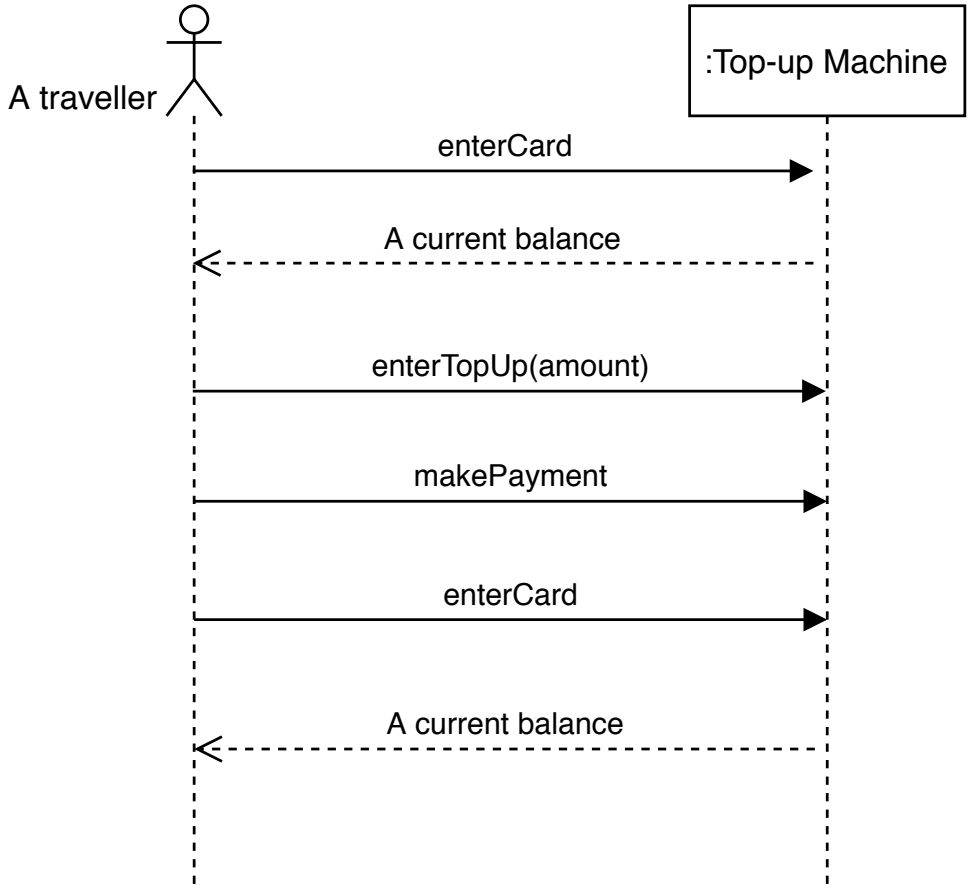
Myki: A smart card ticketing system

https://www.ptv.vic.gov.au/tickets/myki/travel-with-myki/myki-equipment/top-up-faster-with-a-quick-top-up-machine/

Exercise: SSD for Myki Quick Top-up Machine

Quick top-up scenario

1. A traveler taps a Myki card on the reader on the green panel. The machine display the balance of the card.
2. The traveler select the amount of top up from the displayed menu.
3. Then, the traveler taps a credit card to pay.
4. A traveler then taps the Myki card again to load the top up amount into the card. Finally, the machine displays the current balance of the card.

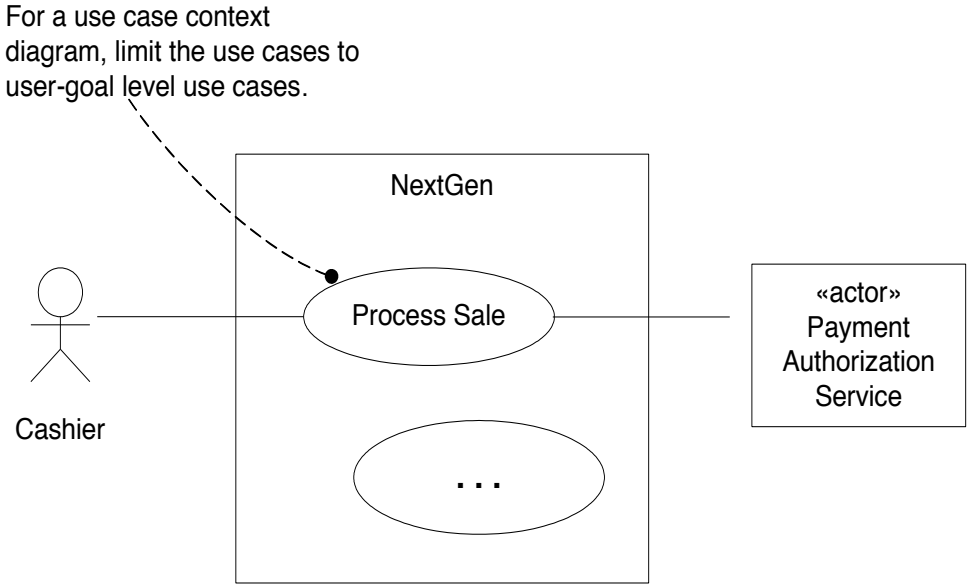


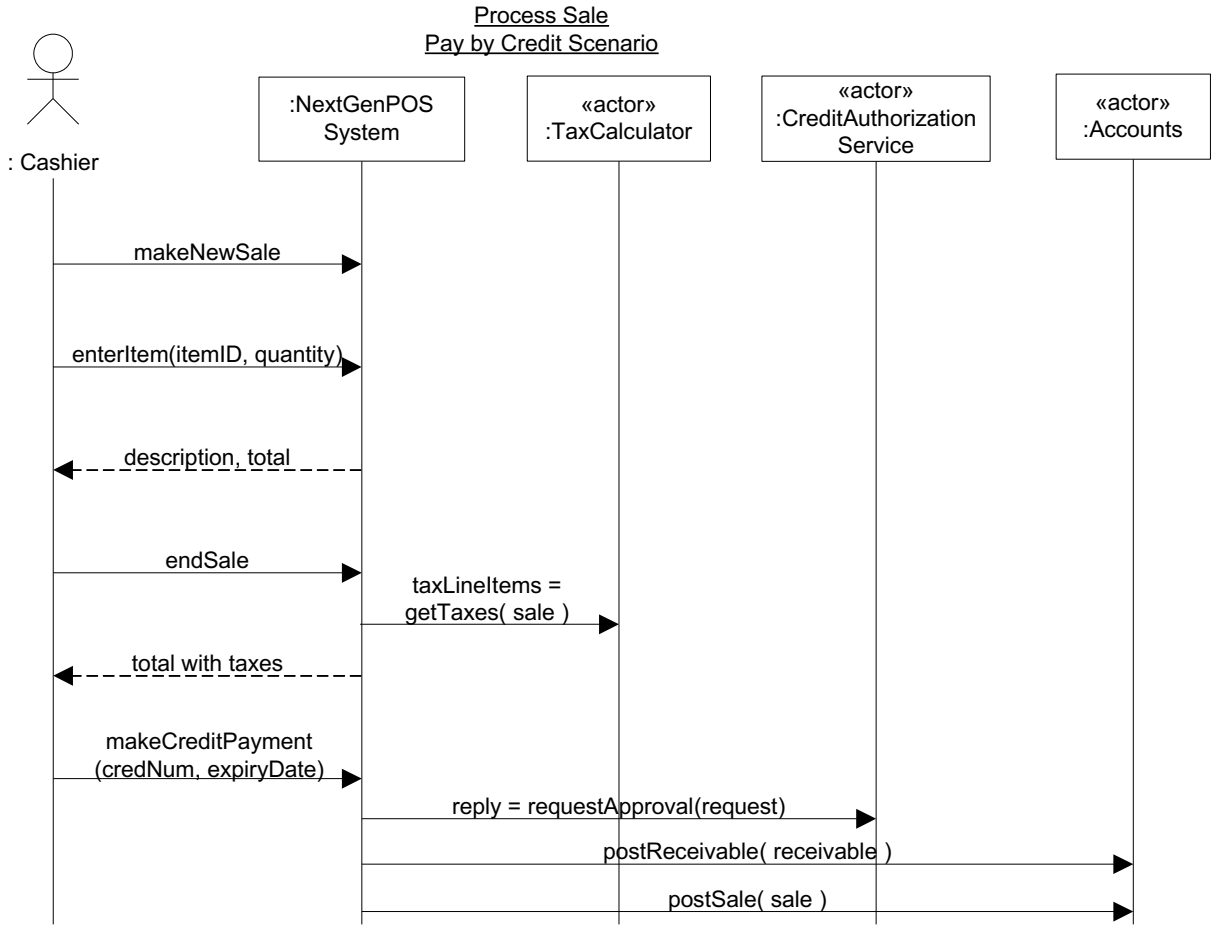
SSD for Myki Quick Top-up Machine

https://www.ptv.vic.gov.au/tickets/myki/travel-with-myki/myki-equipment/top-up-faster-with-a-quick-top-up-machine/

SSD: Inter-system events

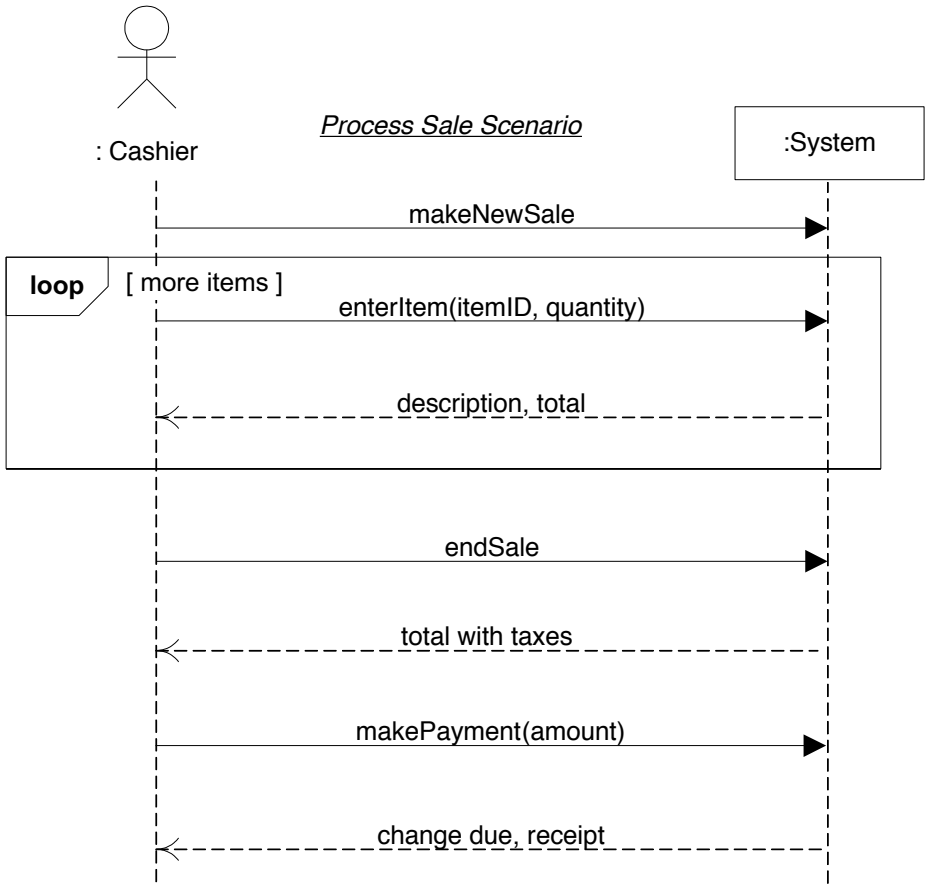
- SSDs can also illustrate interactions between systems, e.g., POS <--> External credit payment authorizer

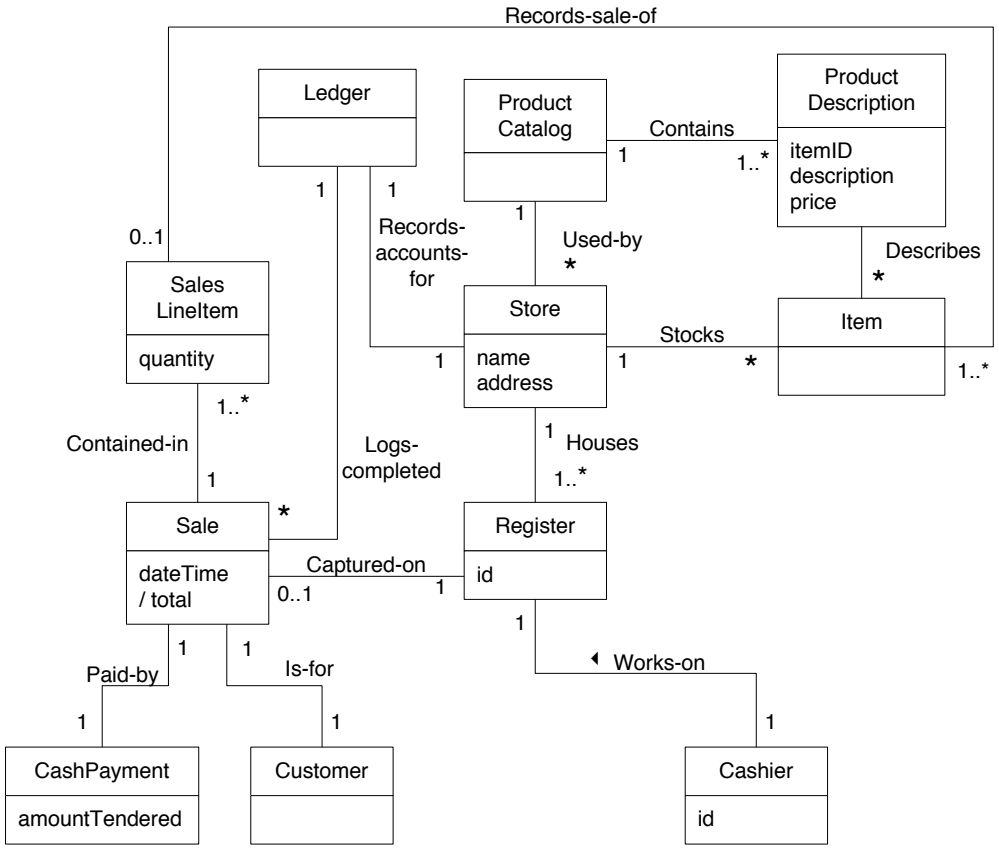




SSDs & Domain Models

- SSDs indicate events which design needs to handle and include associated information with reference to domain model





Summary & Remarks

- ***Object-oriented analysis*** concerns on how to describe the problem domain from the perspective of object
- **Domain models** capture the concepts, attributes, and associations in the domain
  - Provide *static* context of system: A visual dictionary of noteworthy abstractions, domain vocabulary, and information content of the domain
- **Sequence System Diagrams** capture the sequence of system events of one scenario in use cases
  - Provide *dynamic* context of system: How actors interact with system (as a black box) and how the system respond
- Domain models & SSDs should be expressed at the ***abstract level of intention***
  - Provide essential abstractions in information required to understand the domain in the context of the current requirements
  - Aid people in understanding the domain

Lecture Identification

Lecturer: Patanamon Thongtanunam

Semester 2, 2020

These slides include materials from:

Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, Third Edition, by Craig Larman, Pearson Education Inc., 2005.