CS3300: Intro to Software Engineering Georgia Institute of Technology

L06: Use Case Diagram; Domain Model

Dr. Aibek Musaev

Milestone 2: Project Iteration 1

Overall Project Background

- You will be designing and building a digital board game over the next 5 milestones.
- You will have to implement some functionalities that one would expect to be in a digital board game. Examples of some of these functionalities are:
  - Game board
  - Multiple players
  - Dice rolling and moving players accordingly
  - Earning money; drawing chance cards, and more!
- All the milestones will build on top of each other.

Milestone 2 Deliverables

- In M2, you are going to be creating the base for the game.
- You will implement:
  - A welcome screen, allowing users to start the game
  - A configuration screen to allow game customization and allowing players to proceed to the main game screen
- You will need to write unit tests to cover meaningful functionality for your application. Each member of your team should write at least 1 JUnit test.

Milestone 2 Deliverables

- You will also need to create two Design Deliverables, a Use Case diagram and a Domain Model diagram, that describe a general idea of how your game will work.
- You must submit a brief writeup describing your testing process and how your tests prove the code functions as expected.

Milestone 2

- You will need to setup your project in a new repository using **public GitHub**. Make sure to add the Head TA and the instructor to the repository!
- Tags are a way of marking a specific commit in Git and is most commonly used to mark new versions of software. You will be required to tag your final commit before your demo.

Milestone 2 Deadline

- M2 is due ** **Sunday, October 3rd** **
- The Design Deliverables must be turned in by the due date, and your code must be tagged by the due date, but you are able to continue working on your code until your demo.

Use Cases

Use Cases

- Use cases are text stories used to discover and record requirements:
  - Brief
  - Casual
  - Fully-dressed
- Discuss how to write use cases
- Discuss how to draw use case diagram

Use Cases: Brief format

- **Process Sale:** A customer arrives at a checkout with items to purchase. The cashier uses the POS system to record each purchased item. The system presents a running total and line-item details. The customer enters payment information, which the system validates and records. The system updates inventory. The customer receives a receipt from the system and then leaves with the items.

Use Cases: Definitions

- **Actor:** something with behavior, such as a person, computer system, or organization, e.g. a cashier.
- **Scenario:** specific sequence of actions and interactions between actors and the system under discussion, e.g. the scenario of successfully purchasing items with cash.
- **Use case:** a collection of related success and failure scenarios that describe actors using a system to support a goal.

Use Case Modeling

- Use Case Model is defined within the Requirements discipline in UP
- Use Case Model is a set of all written use cases
- Use Case Model may optionally include a Use Case Diagram

Use Cases and adding value

- A key point is to focus on the question “how can using the system provide observable value to the user, or fulfill their goals?”
- Use cases mainly constitute functional requirements.

Three Kinds of Actors

- Primary actor
  - has user goals fulfilled through using services of the SuD
- Supporting actor
  - provides a service (for example, information) to the SuD
- Offstage actor
  - has an interest in the behavior of the use case

Use Cases: Casual format

Handle returns

Main success scenario: A customer arrives at a checkout with items to return. The cashier uses the POS system to record each returned item…

Alternate scenarios:

If the credit authorization is rejected, inform customer and ask for an alternative payment method.

If item identifier not found in the system, notify the Cashier and suggest manual entry of the identifier code.

…

Use Cases: Fully-dressed Style Template

Use Case Section	Comment
Use Case Name	Start with a verb.
Scope	The system under design.
Level	“user-goal” or “subfunction”
Primary Actor	Calls on the system to deliver its services.
Stakeholders and Interests	Who cares about this use case, and what do they want?
Preconditions	What must be true on start, *and* worth telling the reader?
Success Guarantee	What must be true on successful completion, *and* worth telling the reader.
Main Success Scenario	A typical, unconditional happy path scenario of success.
Extensions	Alternate scenarios of success or failure.
Special Requirements	Related non-functional requirements.
Technology and Data Variations List	Varying I/O methods and data formats.
Frequency of Occurrence	Influences investigation, testing, and timing of implementation.
Miscellaneous	Such as open issues.

Use Cases: Fully-dressed example: Process Sale

Use case UC1: Process Sale
Primary Actor: Cashier
Stakeholders and Interests:
-Cashier: Wants accurate and fast entry, no payment errors, …
-Salesperson: Wants sales commissions updated.
…
Preconditions: Cashier is identified and authenticated.
Success Guarantee (Postconditions):
-Sale is saved. Tax correctly calculated.
…
Main success scenario (or basic flow): [see next slide]
Extensions (or alternative flows): [see next slide]
Special requirements: Touch screen UI, …
Technology and Data Variations List:
-Identifier entered by bar code scanner,…
Open issues: What are the tax law variations? …

Use Cases: Fully dressed example: Process Sale (cont.)

Main success scenario (or basic flow):

The Customer arrives at a POS checkout with items to purchase.
The cashier records the identifier for each item. If there is more than one of the same item, the Cashier can enter the quantity as well.
The system determines the item price and adds the item information to the running sales transaction. The description and the price of the current item are presented.
On completion of item entry, the Cashier indicates to the POS system that item entry is complete.
The System calculates and presents the sale total.
The Cashier tells the customer the total.
The Customer gives a cash payment ("cash tendered") possibly greater than the sale total.

Extensions (or alternative flows):

If invalid identifier entered. Indicate error.
If customer didn't have enough cash, cancel sales transaction.

Goals and Scope of a Use Case

- At what level and scope should use cases be expressed?
- A: Focus on use cases at the level of **elementary business process** (EBP).
- EBP: a task performed by one person in one place at one time which adds measurable business value and leaves the data in a consistent state.
  - Approve credit order - OK.
  - Negotiate a supplier contract - not OK.
- It is usually useful to create separate "sub" use cases representing subtasks within a base use case.
  - e.g. Paying by credit

Finding primary actors, goals and use cases

- Choose the system boundary.
- Identify primary actors.
  - Those that have user goals fulfilled through using services of the system
- For each actor identify their user goals.
  - Tabulate findings in the Vision artifact.
- Define use cases that satisfy user goals; name them according to their goal
  - The EBP Test, The Boss Test, The Size Test

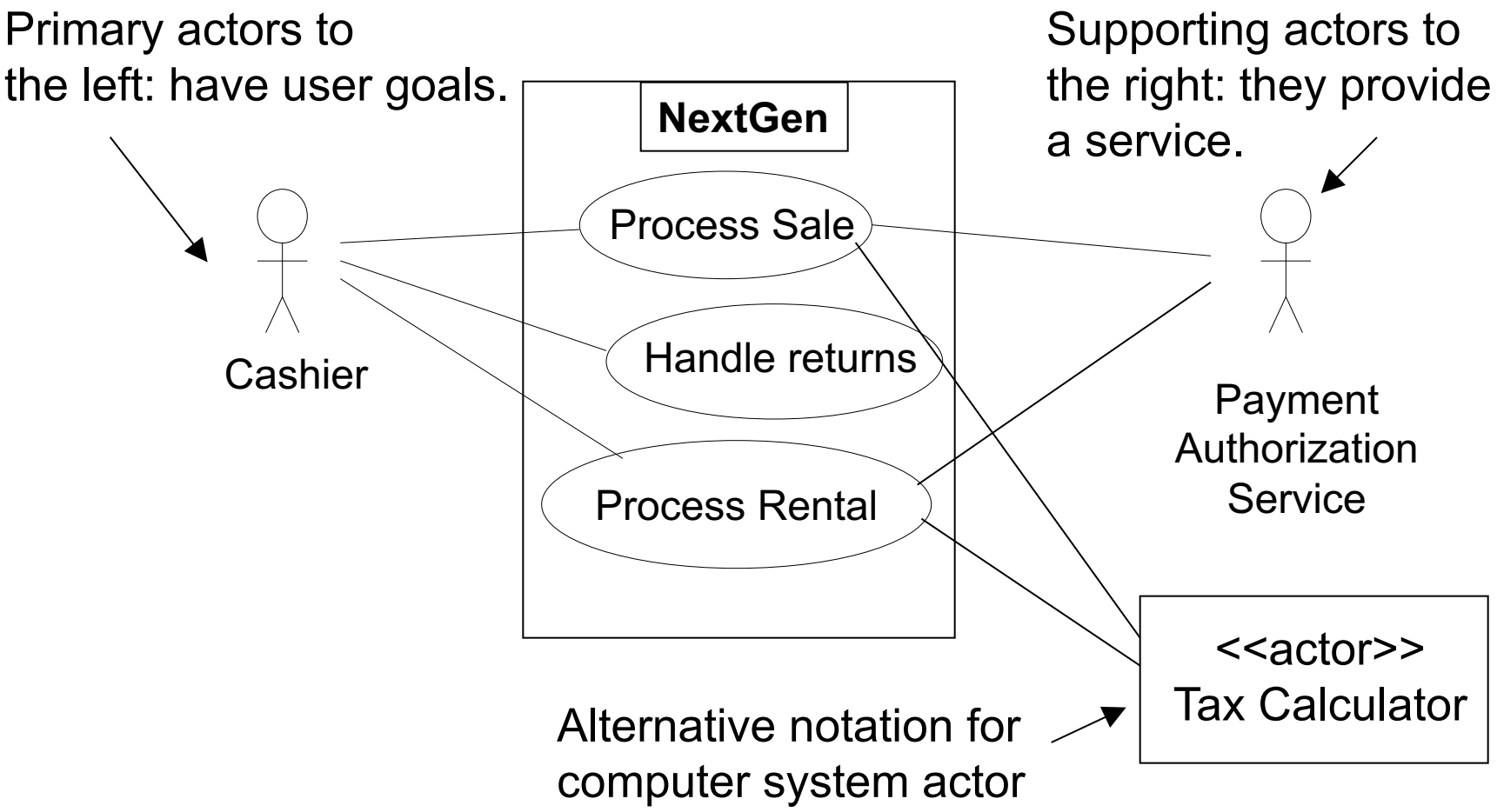
Determining validity of use cases

- Elementary Business Process (EBP) Test
  - A task performed by one person in one place at one time, in response to a business event, which adds measurable business value and leaves the data in a consistent state
- Boss Test
  - What have you been doing all day?
- Size Test
  - Use typically contains many steps as opposed to a single step within a series of steps

Essential vs. Concrete style

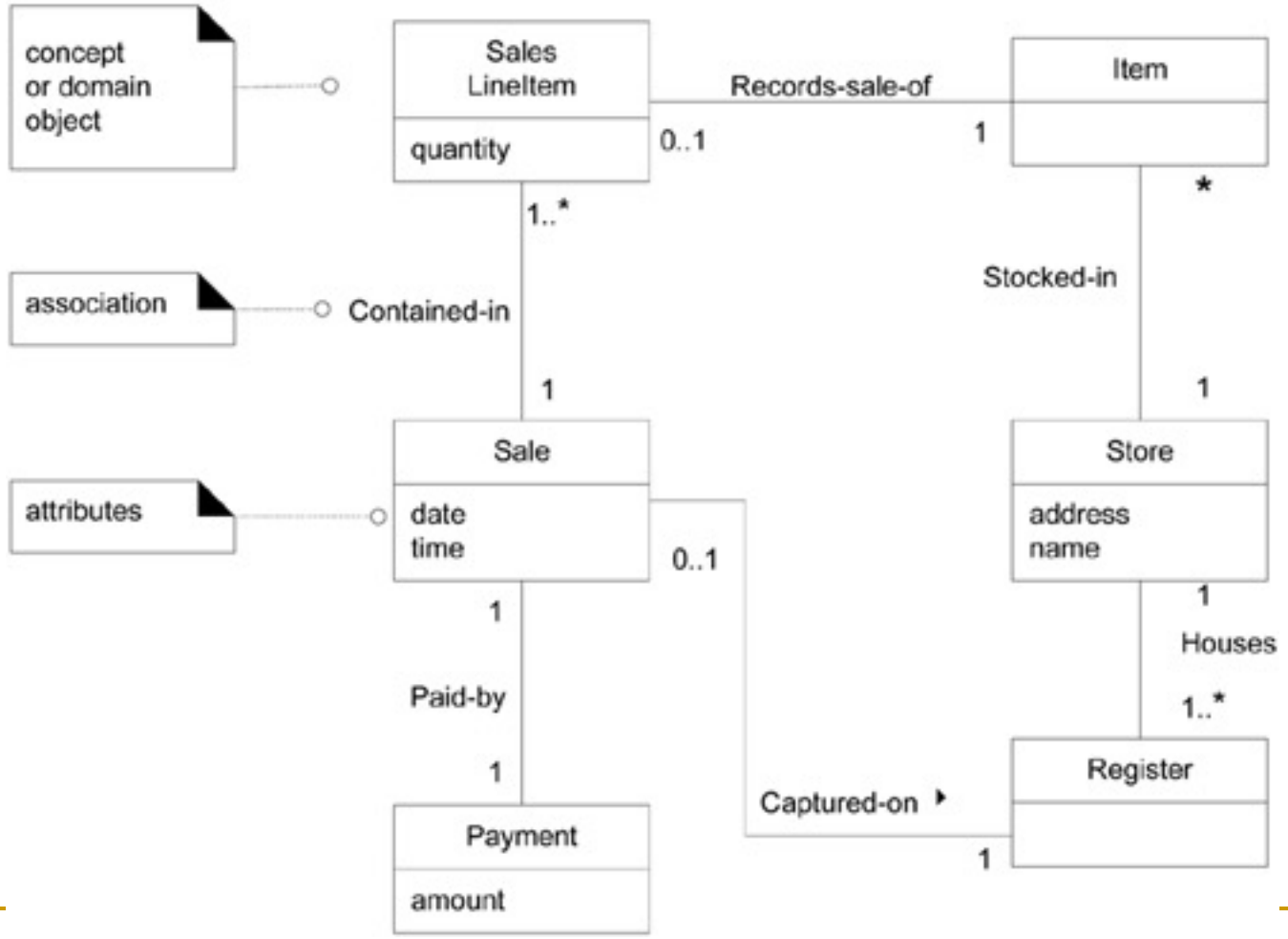
- Essential: Focus is on intent.
  - Avoid making UI decisions
- Concrete: UI decisions are embedded in the use case text.
  - e.g. “Admin enters ID and password in the dialog box (see picture X)”
  - Concrete style not suitable during early requirements analysis work.

Use Case Diagrams



Domain Models

Partial Domain Model as a Visual Dictionary

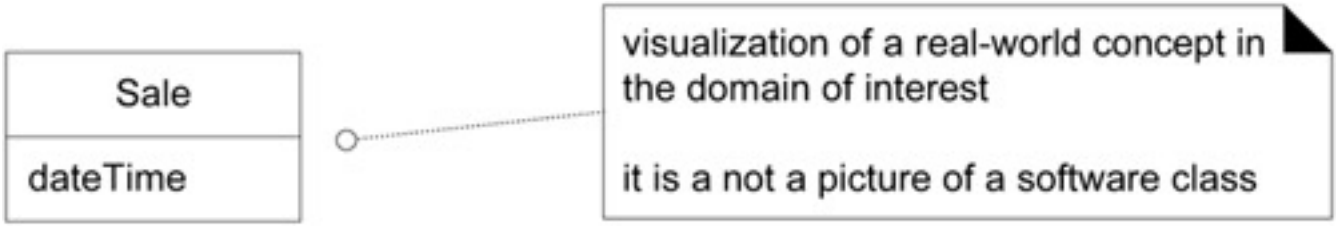


What is a Domain Model?

- A **domain model** is a visual representation of conceptual classes or real-situation objects in a domain
- A domain model may show:
  - domain objects or conceptual classes
  - associations between conceptual classes
  - attributes of conceptual classes

Is Domain Model a picture of software objects?

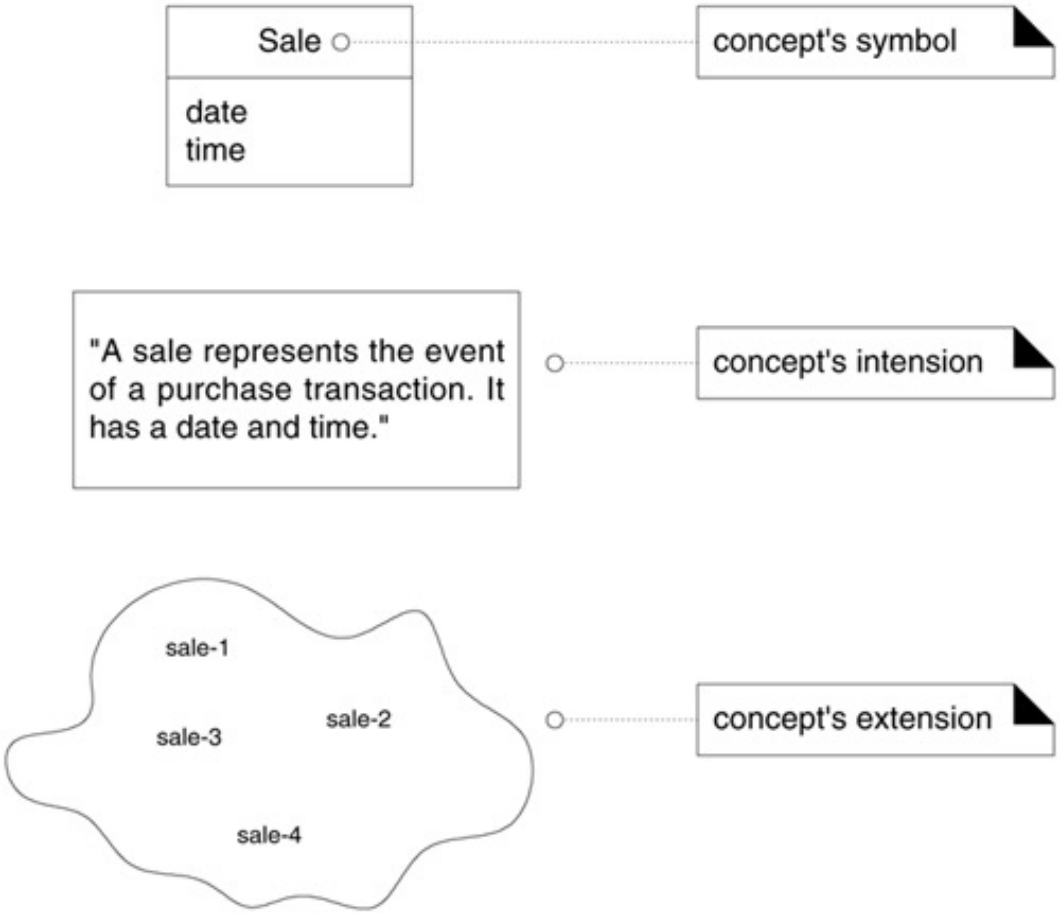
- A domain model shows conceptual classes, not software classes



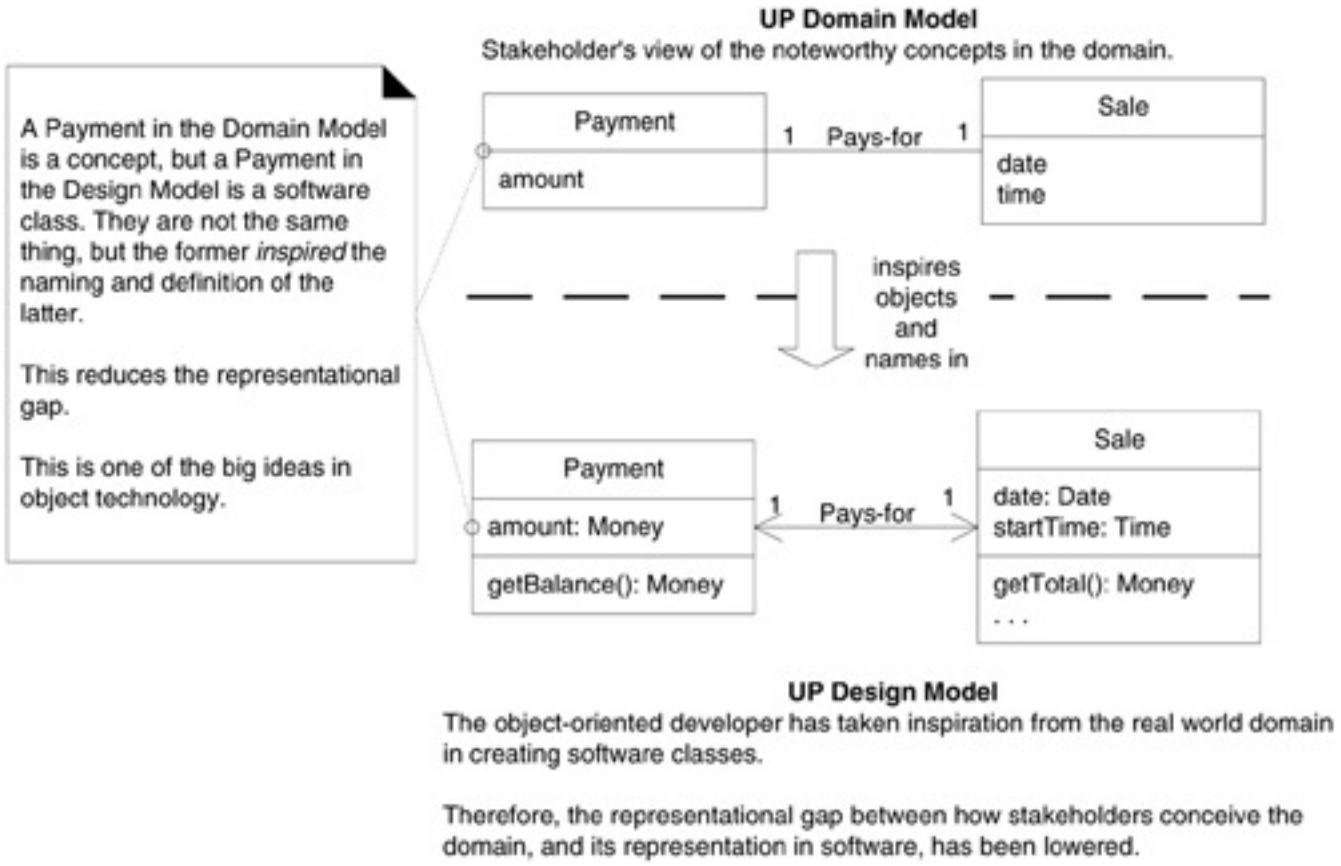
- The following elements are not suitable in a domain model:
  - Software artifacts, such as a window or a database
  - Responsibilities or methods

What are conceptual classes?

- A conceptual class has a symbol, intension, and extension



Lower representational gap with OO Modeling



How to create a Domain Model?

1. Find the conceptual classes
2. Draw them as classes in a UML class diagram.
3. Add associations and attributes

Three strategies for finding conceptual classes

- Reuse or modify existing models
  - Analysis Patterns by Martin Fowler
  - Data Model Patterns by David Hay
  - Data Model Resource Book by Len Silverston.
- Use a category list
- Identify noun phrases

Using noun phrase identification

- Identify the nouns and noun phrases and consider them as candidate conceptual classes or attributes
- Care must be applied
  - Mechanical noun-to-class mapping is not possible
  - Words may be ambiguous

Main Success 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**. Price calculated from a set of price rules.

Case Study: POS Domain

Guideline: include Report Objects in the Model?

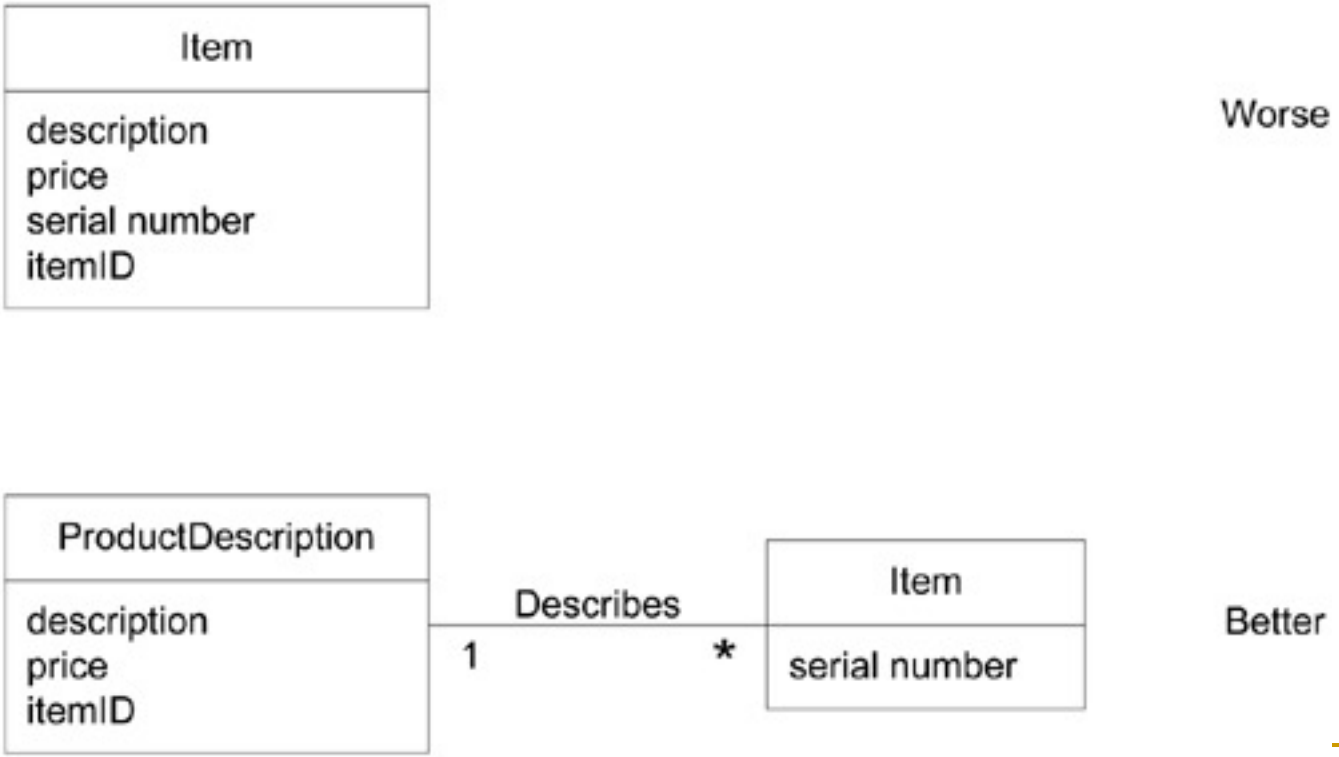
- *Receipt* is a noteworthy term in the POS domain. Should it be in the domain model?
- A report's information is derived or duplicated from other sources
- However, a receipt may be used when returning purchased items
- In this iteration, Receipt will be excluded

Attributes vs Classes?

- Guideline: 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.

What about 'Description' classes?

- **A description class** contains information that describes something else

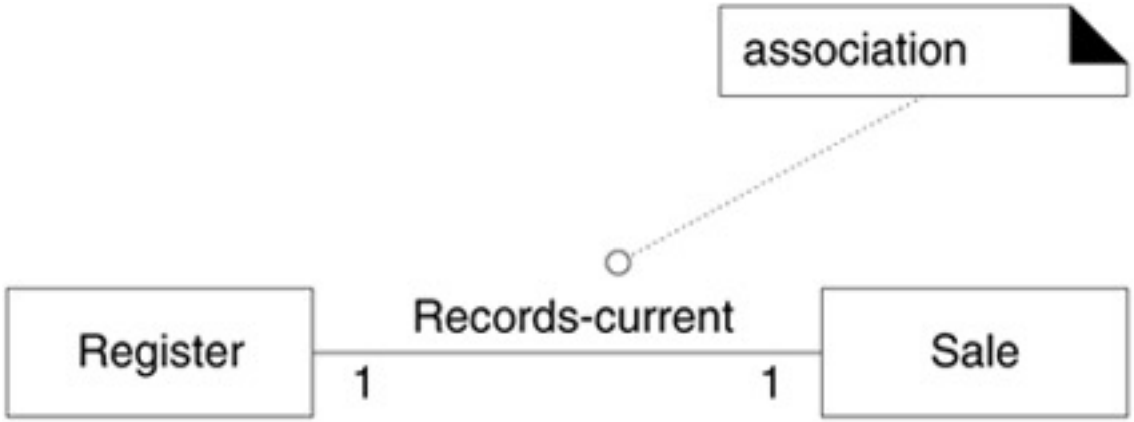


When are Description Classes Useful?

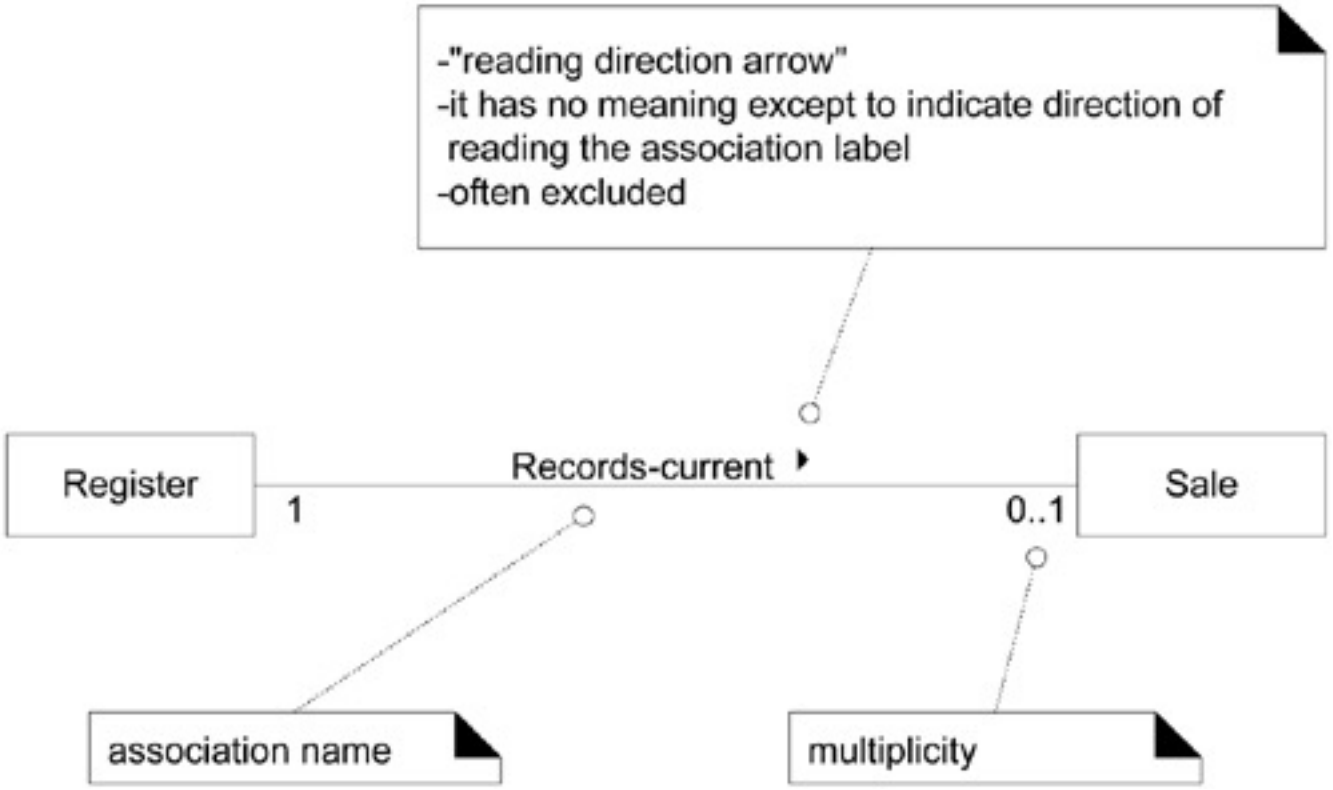
- Add a description class when:
  - There needs to be a description about an item or service, independent of the current existence of any examples of those items or services.
  - Deleting instances of things they describe (for example, Item) results in a loss of information that needs to be maintained, but was incorrectly associated with the deleted thing.
  - It reduces redundant or duplicated information.

Associations

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



Association Notation

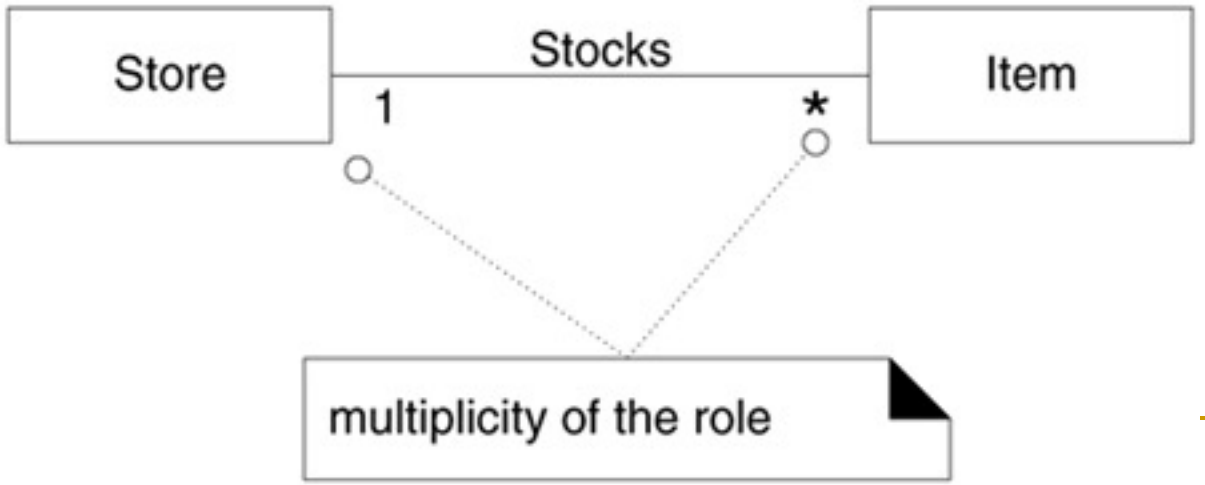


Naming associations in UML

- Name an association based on a ClassName-VerbPhrase-ClassName format where the verb phrase creates a sequence that is readable and meaningful
- *Sale Paid-by CashPayment*
  - bad example: Sale Uses CashPayment
- *Player Is-on Square*
  - bad example: Player Has Square

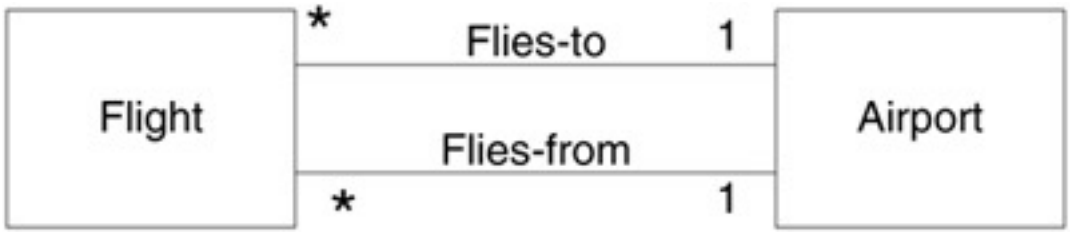
Roles and Multiplicity

- Each end of an association is called a role. Roles may optionally have:
  - multiplicity expression
  - name
  - navigability

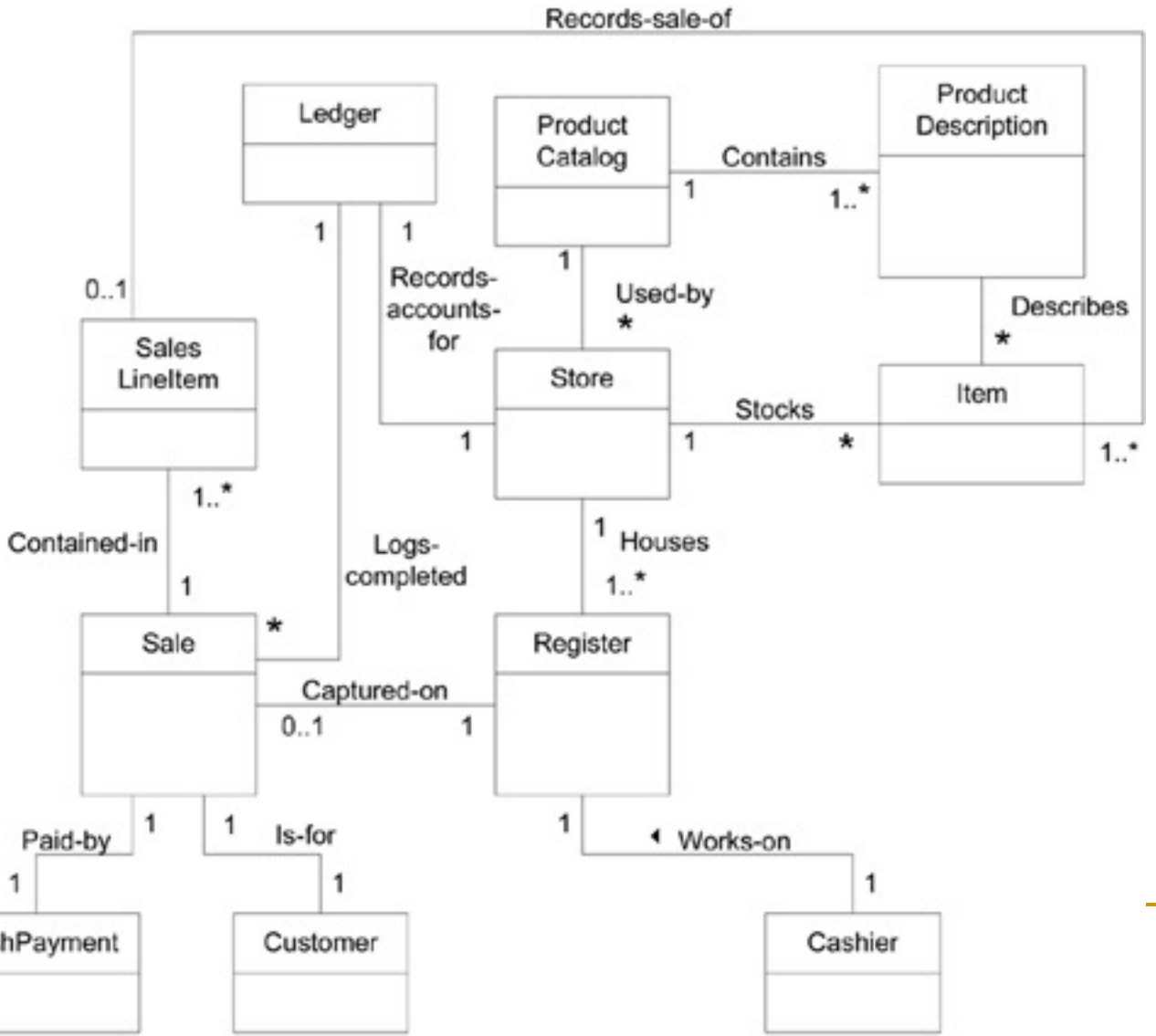


Multiple associations between two classes

- Two classes may have multiple associations between them in a UML class diagram



Case Study: NextGen POS

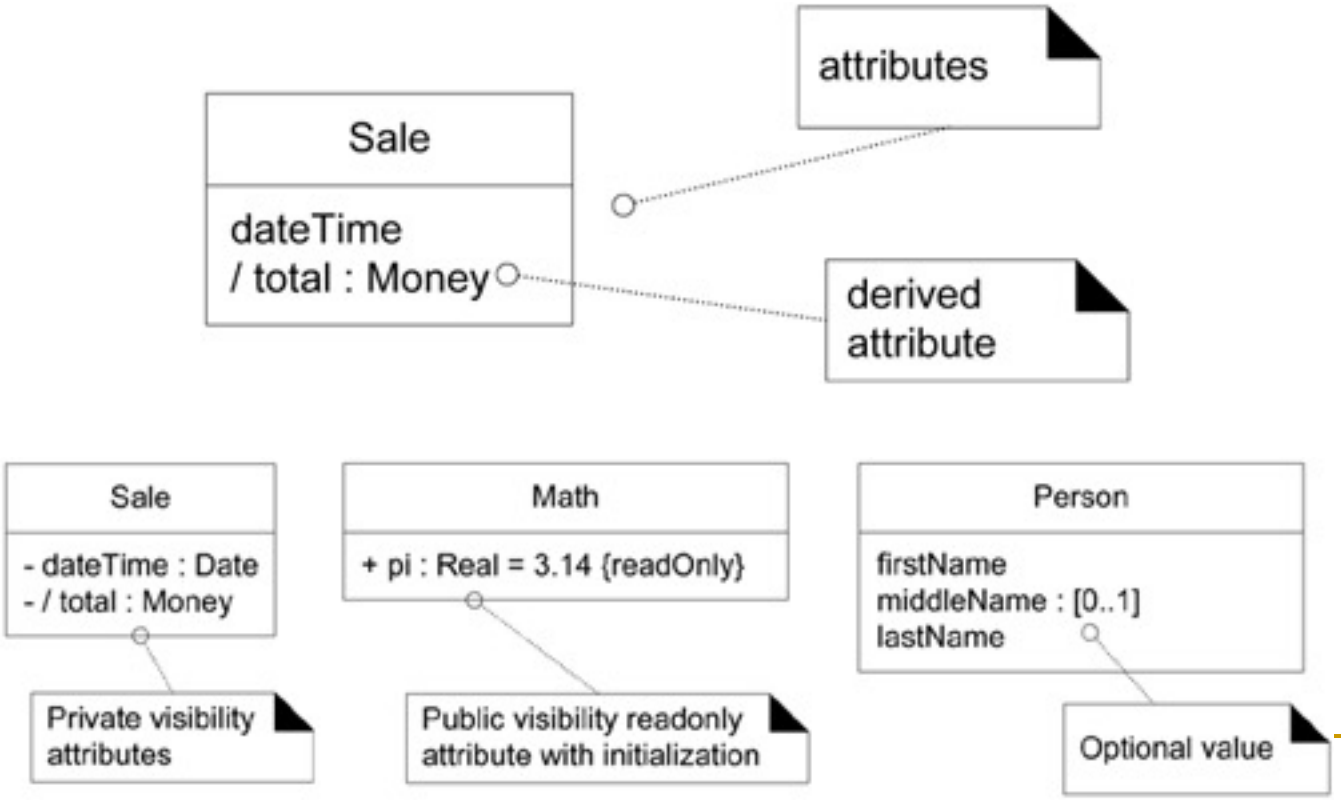


Attributes

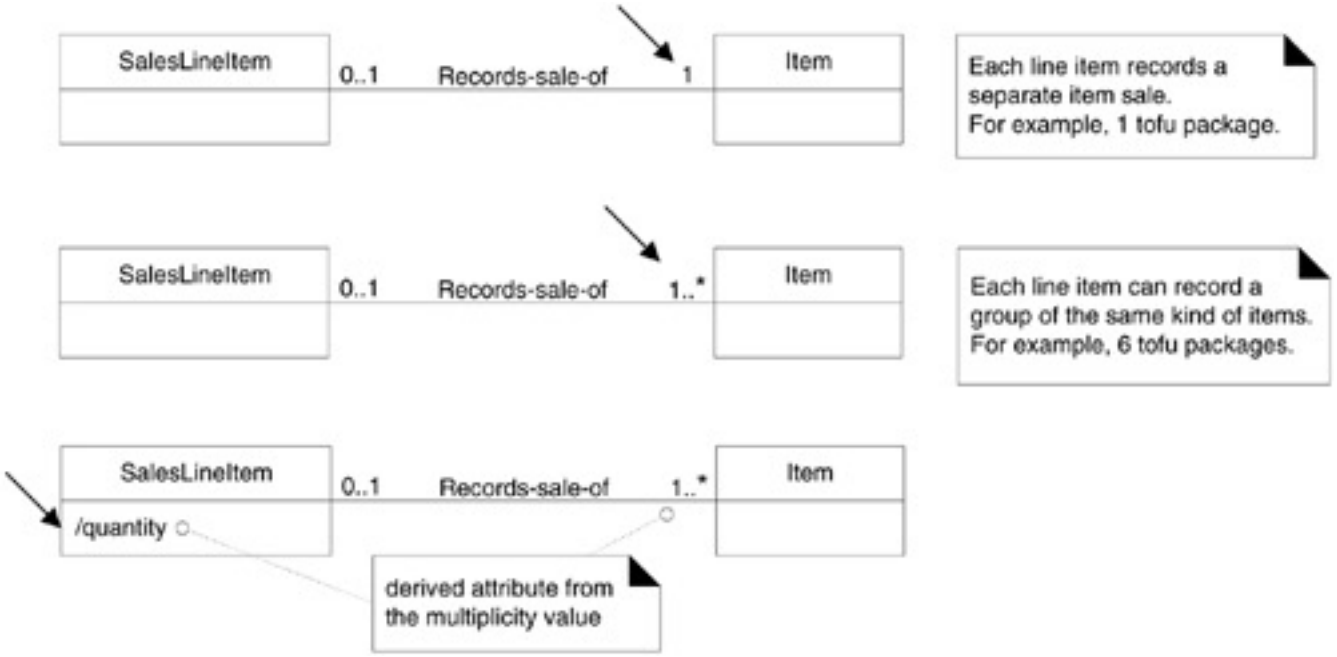
- **An attribute** is a logical data value of an object
- When to show attributes?
  - Include attributes that the requirements suggest or imply a need to remember information

Attribute notation in UML

- Attributes are shown in the second compartment of the class box



Derived attributes



Case Study: NextGen POS