SOFTWARE ENGINEERING CS-UH 2012, SPRING 2025

Lecture 5: Domain Modeling

Dr. Mohamad Kassab, m.kassab@nyu.edu

LESSON PLAN

- How to Construct Domain Model from Use Cases.
  - Identifying Domain Classes
  - Identifying Attributes
  - Identifying Associations
    - Generalizations
    - Association Classes
    - Composition / Aggregation

What is a Domain Model?

- The domain model captures the most important types of objects in the context of the system.

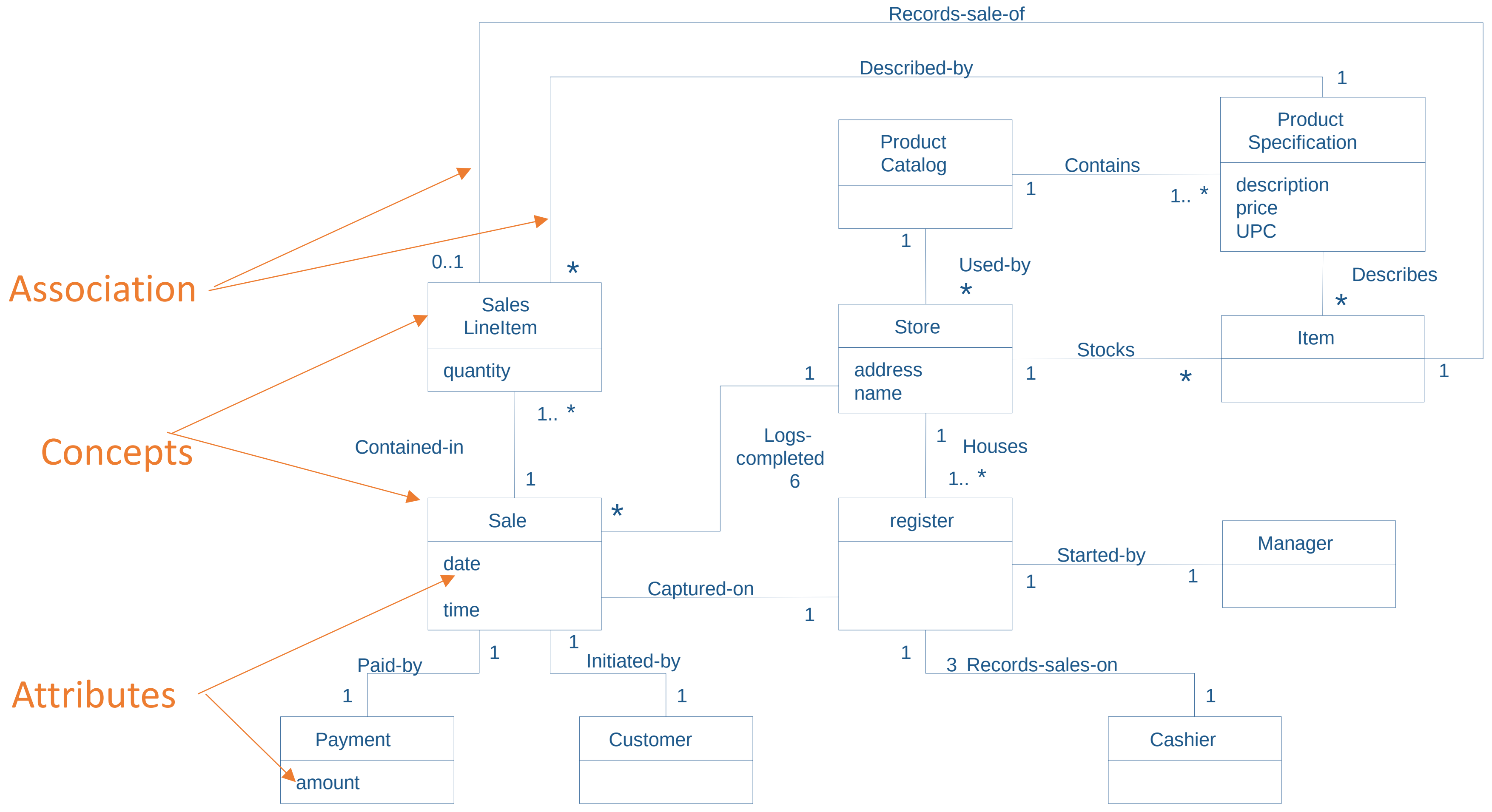
- Domain objects represent the "**things**" that exist or events that transpire in the environment in which the system works.

- The domain model is bounded by the use case scenario under development. It is not a description of software objects.

- The domain model is a **visualization of the concepts**.

- It comes from a specification of conceptual **classes**, their **attributes** and **associations** described via UML Class diagrams.

Hoes does a Domain Model Looks Like?



What is a Domain Model?

- The domain model illustrates the domain **classes** and how they are related to one another by **association**.

- It is a static model; therefore, no sequence of events or information flow is shown in the domain model.

How to Construct a Domain Model?

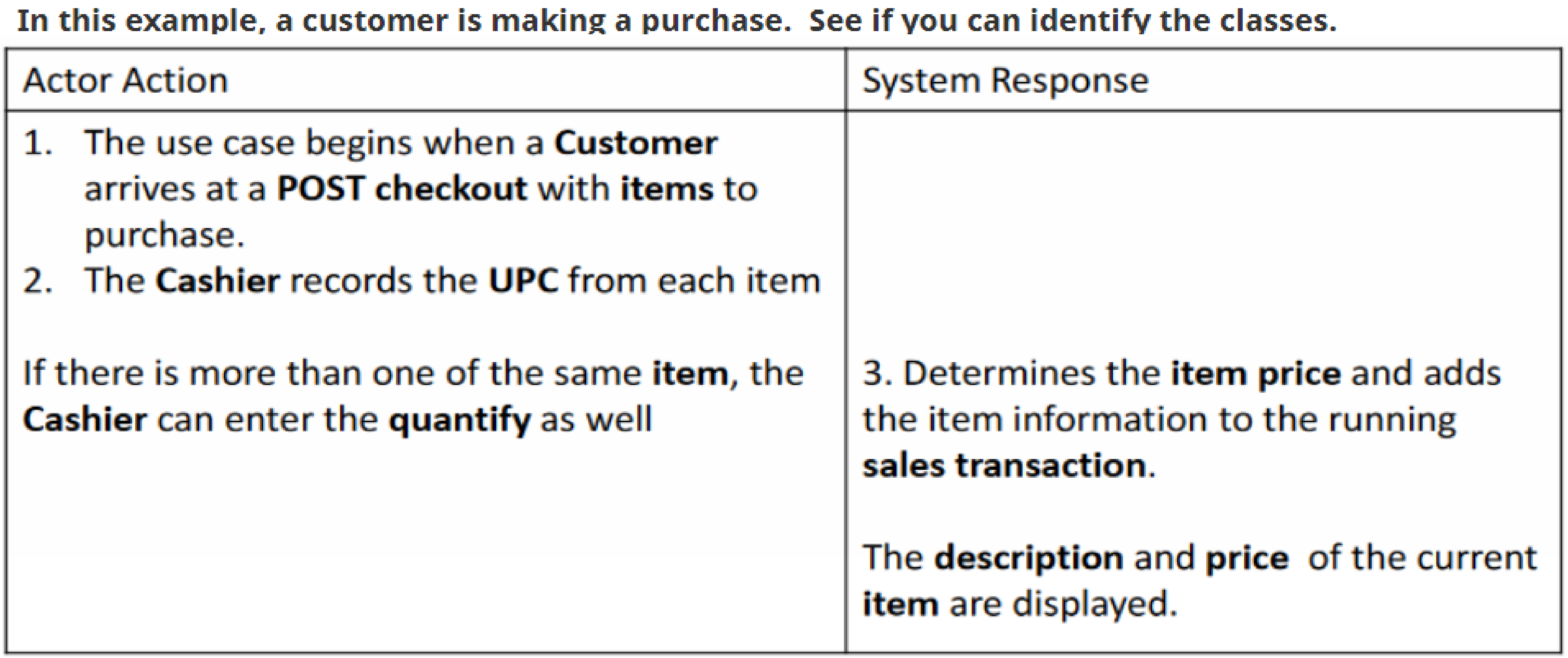
- 1- We must identify classes
- 2- We must identify Attributes
- 3- We must identify Associations

Identifying Classes

- The textual description of the use case; create a list of potential classes.

- Use the nouns to identify potential classes or attributes.

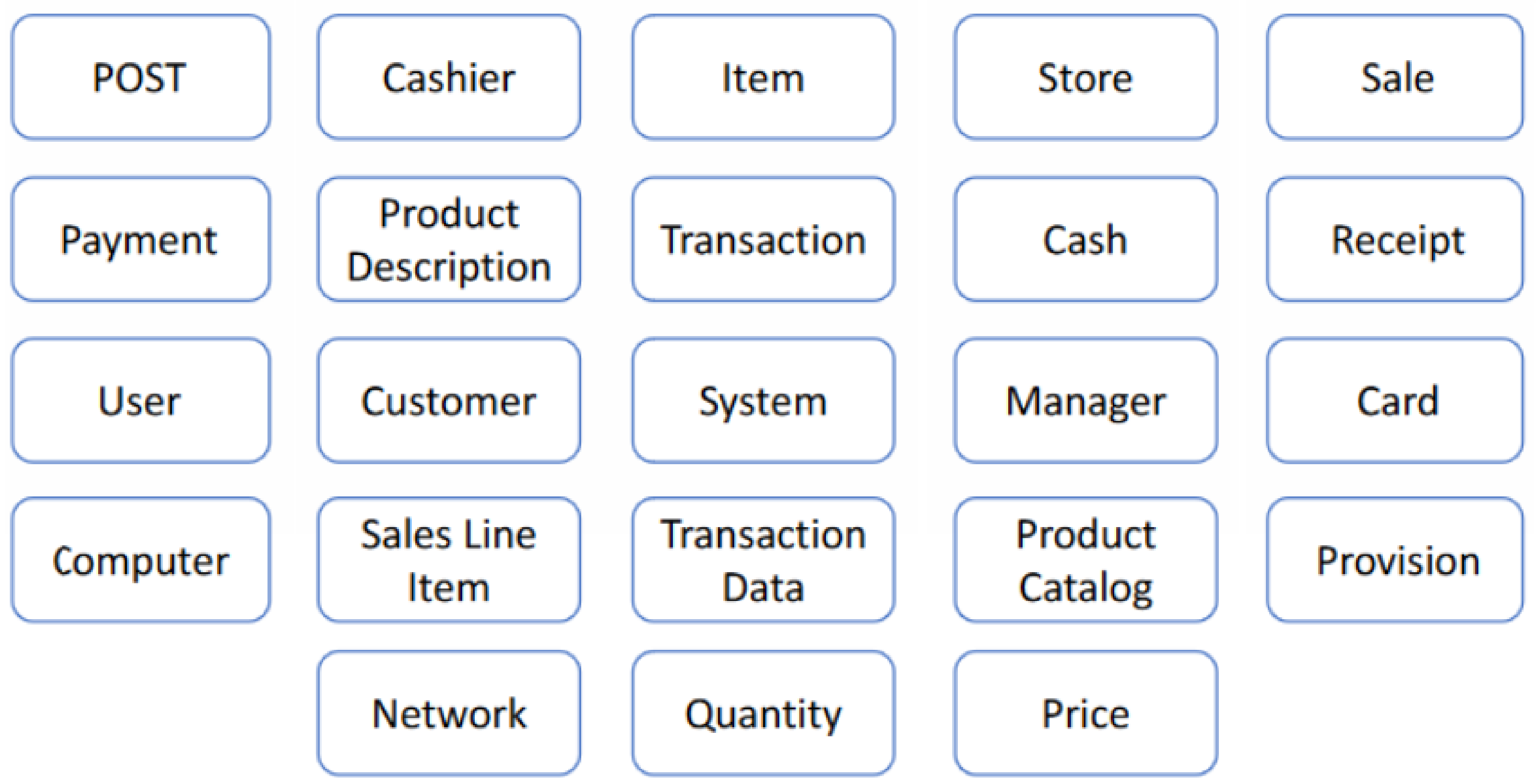
Identifying Classes



In this example, a customer is making a purchase. See if you can identify the classes.

Actor Action	System Response
1. The use case begins when a **Customer** arrives at a **POST checkout** with **items** to purchase. 2. The **Cashier** records the **UPC** from each item	
If there is more than one of the same **item**, the **Cashier** can enter the **quantify** as well	3. Determines the **item price** and adds the item information to the running **sales transaction**. The **description** and **price** of the current **item** are displayed.

Did you identify the classes shown below?

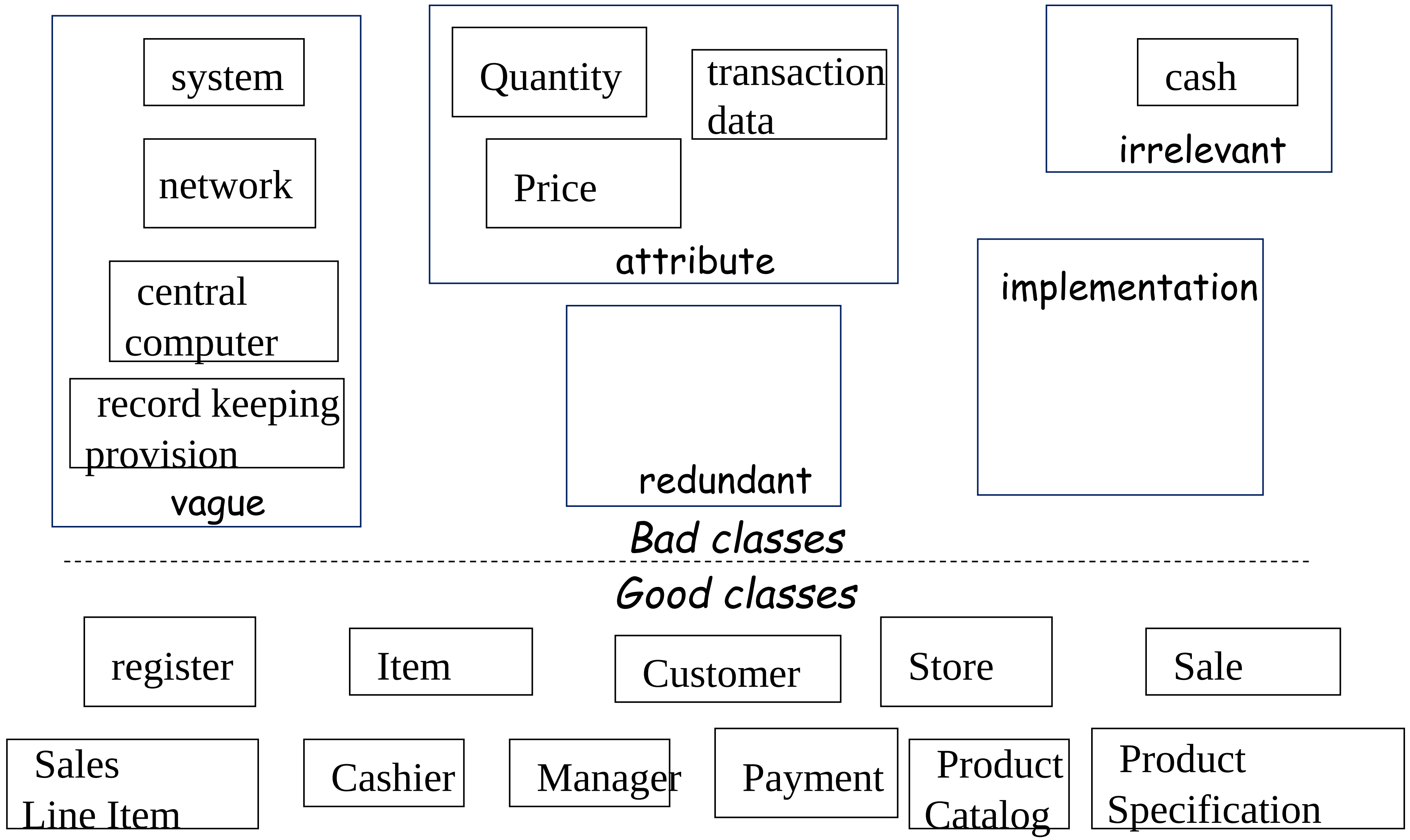


POST	Cashier	Item	Store	Sale
Payment	Product Description	Transaction	Cash	Receipt
User	Customer	System	Manager	Card
Computer	Sales Line Item	Transaction Data	Product Catalog	Provision
	Network	Quantity	Price	

Pruning the List of Classes

- Examine list of potential classes & eliminate
  - *Redundant Classes:* Two classes that express the same information
  - *Irrelevant Classes:* Classes that have little or nothing to do with the problem
  - *Vague Classes:* Classes with ill-defined boundaries or too broad in scope
  - *Implementation Specific Classes:* Classes that represent software or processes

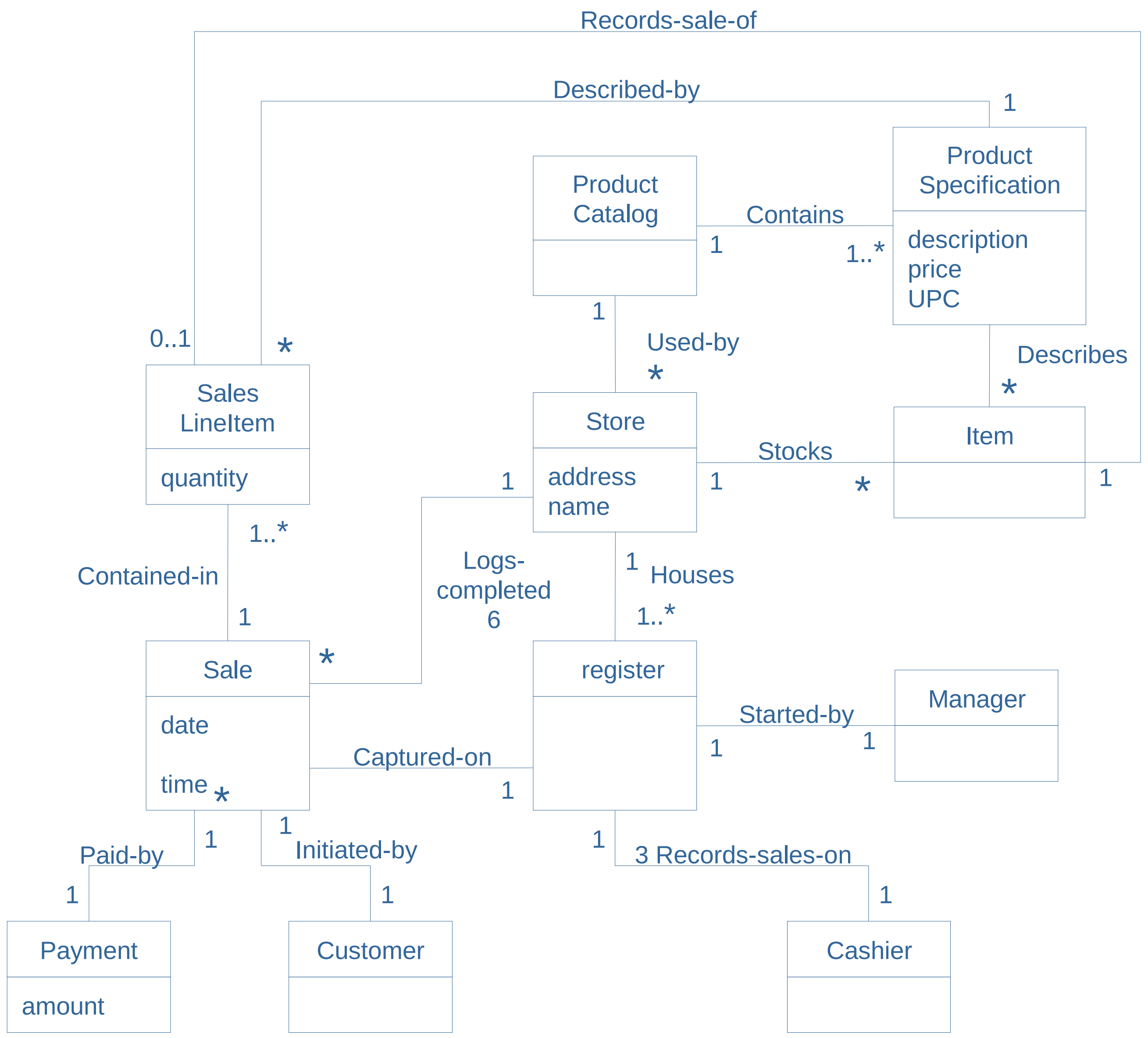
Example: Pruning the List



Taking this example further, the "good classes" were put into a domain model *diagram*.

This diagram shows the multiplicity which defines how many instances of **class A** can be associated with **class B.**

For example, a single instance of *Store* can be associated with "many" (zero or more, indicated by the *) Sale instances.

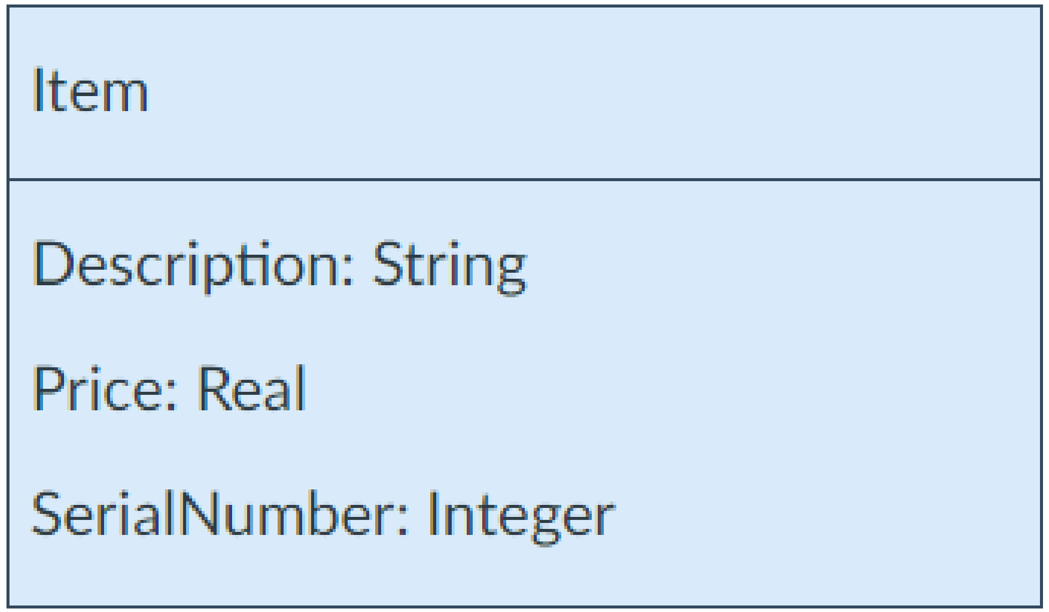


How to Construct a Domain Model?

- 1- We must identify classes
- 2- We must identify Attributes
- 3- We must identify Associations

What is a Class Attribute?

- After deciding on the list of domain classes to include, we need to add the essential attributes to each class.
- A class encapsulates the state (**attributes**) of an object. These are the logical data values of the objects.
- Attributes should be included when requirements suggest or imply a need to remember information.



Attributes

- Include attributes when requirements suggest or imply a need to remember information
- The syntax of an attribute is

[visibility] name [multiplicity] [:type] [= initial value] [{property}]

Attributes (Contd.)

- Examples:

• height	Name only
• + height	Visibility and name (public)
• height : Integer	Name and type
• name [0..1]: String	Name, multiplicity, and type
• width : Real = 0.0	Name, type, and initial value
• id: Integer {frozen}	Name, type, and property

Attributes (Contd.)

- **A property of an attribute can be:**
  - **Changeable** No restrictions on modifying the attribute's value.
  - **addOnly** For attributes with a multiplicity greater than one, additional values may be added, but once created, a value may not be removed or altered.
  - **frozen** The attribute’s value may not be changed after the object is initialized.

Attributes (Cont.)

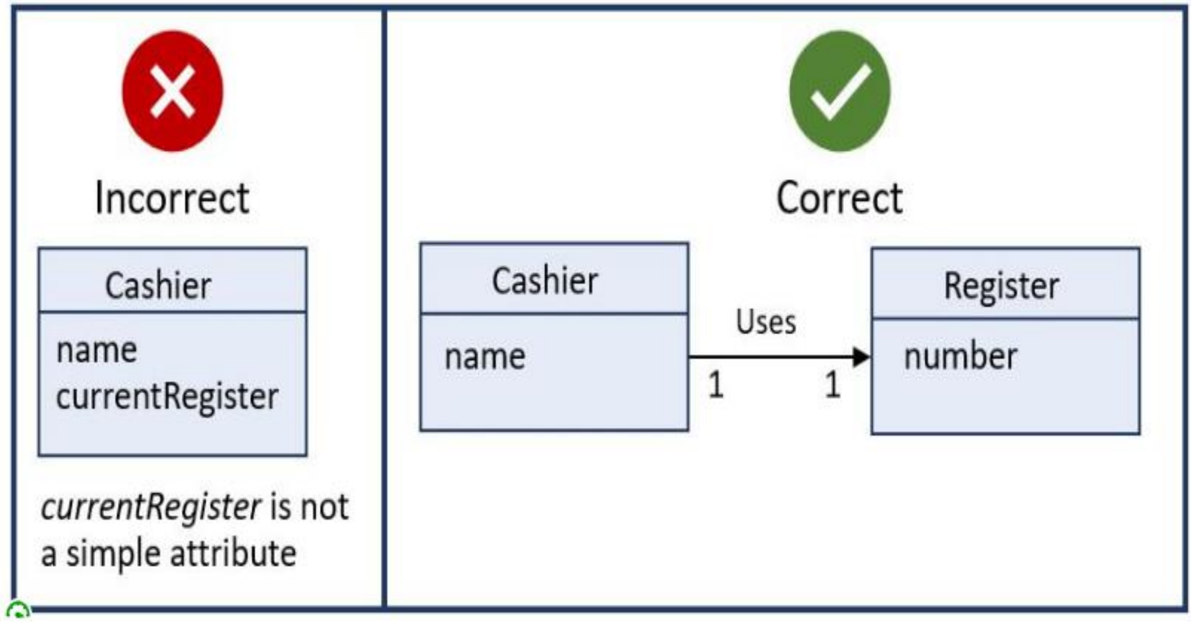
- Attributes should preferably be **simple attributes** or **data types**.

- In the domain mode, attributes should be data types generally where the types are very common: Boolean, Date, Number, String/text, time.

- Other common types are: Address, color, phone number, SS#, UPC, SKU, ZIP, enumerated types.

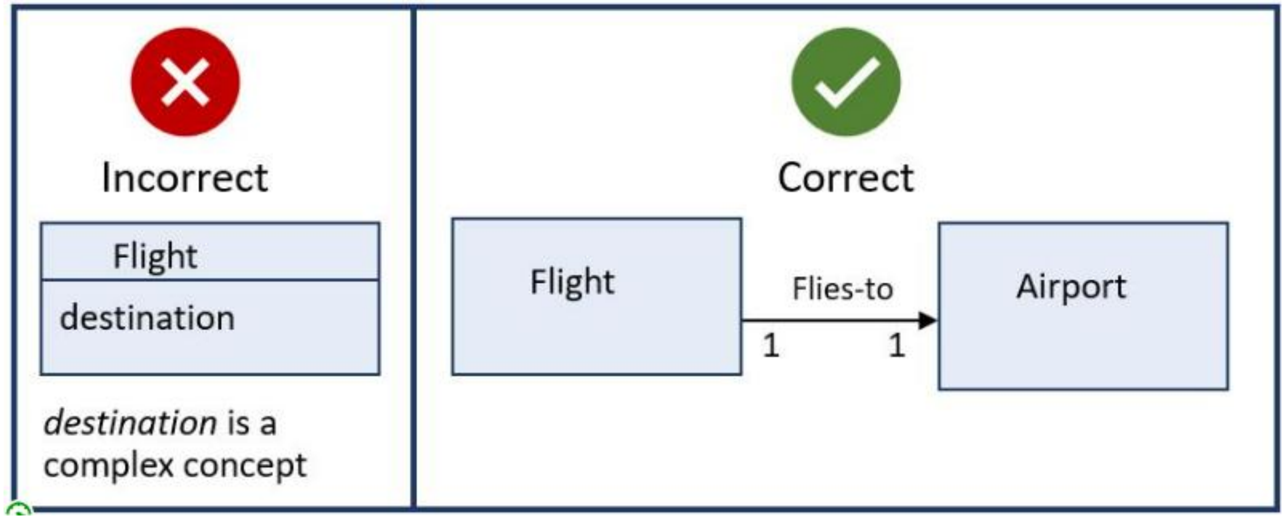
- ***Tip 1*****:** Stick to simple types.
- ***Tip 2*****:** If in doubt, define something as a **concept**, rather than an attribute.

Example 1:



Register is a complex concept, not a simple one. If you want to relate the Cashier to Register, it is advisable to relate the two concepts through an association rather than having Register an attribute in the class Cashier.

Example 2:



Here is another example of wrongly representing complex domain concepts as attributes. Again, you want to use associations.

In this example, the airport "destination" is not just a string. It is much more complex, therefore a flight should be related to the airport via an **association**.

Attributes vs. Concepts

- Does it make sense to distinguish between two separate instances of a *Person* whose names are John Doe?
- In attributes, identity is determined by the value

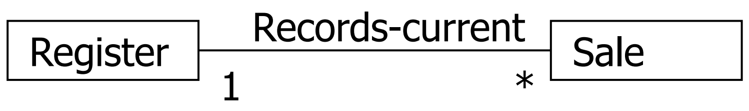
- **Tip:**
  - Stick to simple types
  - If in doubt, define something as concept, rather than attribute

How to Construct a Domain Model?

- 1- We must identify classes
- 2- We must identify Attributes
- 3- We must identify Associations

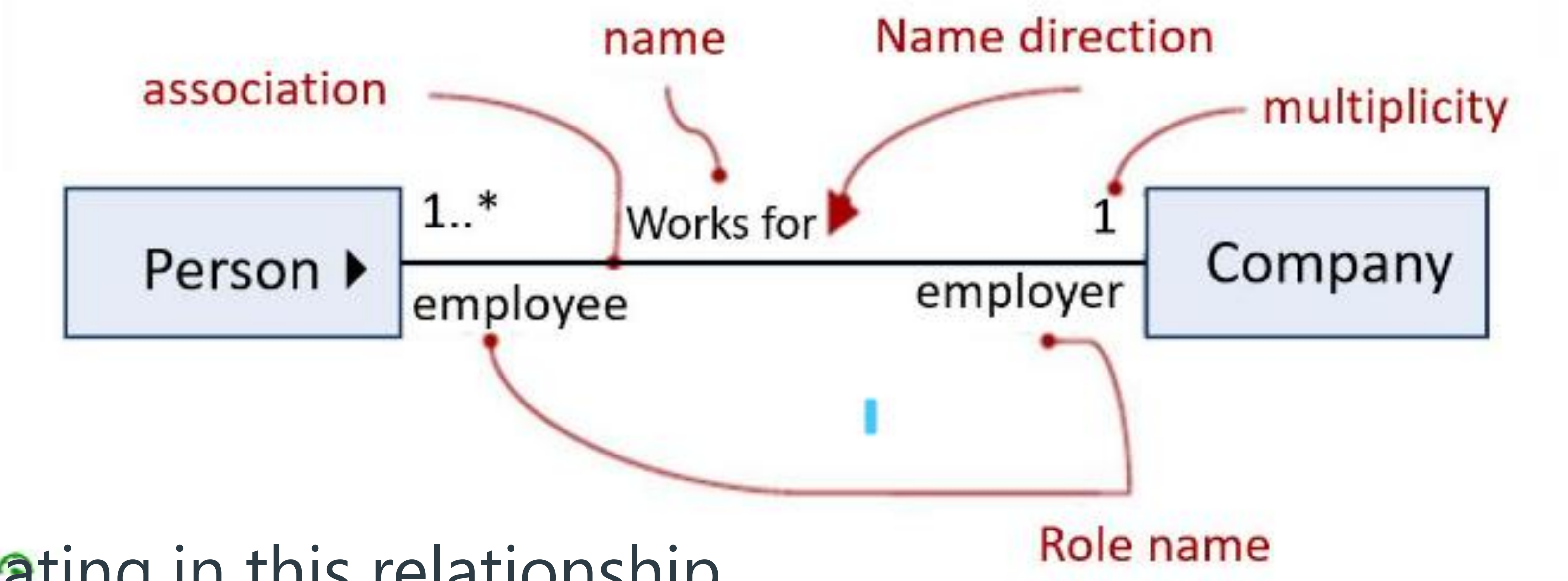
Association

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



Associations

The illustration below shows 4 basic adornments that apply to an association:



One *Person* participating in this relationship works for **ONE** *Company*

One *Company* in this relationship may have "**ONE or MORE**" *Persons* working for it.

Association Adornments

- There are 3 basic adornments that apply to an association:
  - ***name*** - describes the nature of the relationship. To avoid ambiguity about its meaning, the direction of the association can be shown with a triangle.
  - ***role*** - when a class participates in an association, it has a specific role that it plays in that relationship: a role can be thought as the face the class at the near end presents to the class at the other end.
  - ***multiplicity*** – cardinality of the relationship. Indicates how many instances of the classes participate in the given relationship.

How to Identify an Association?

- Examine the concepts for the following relationships:

  - A is *physical* or *logical part* of B
  - A is *physically* or *logically contained* in/on B
  - A is *recorded* in B

Association Guidelines

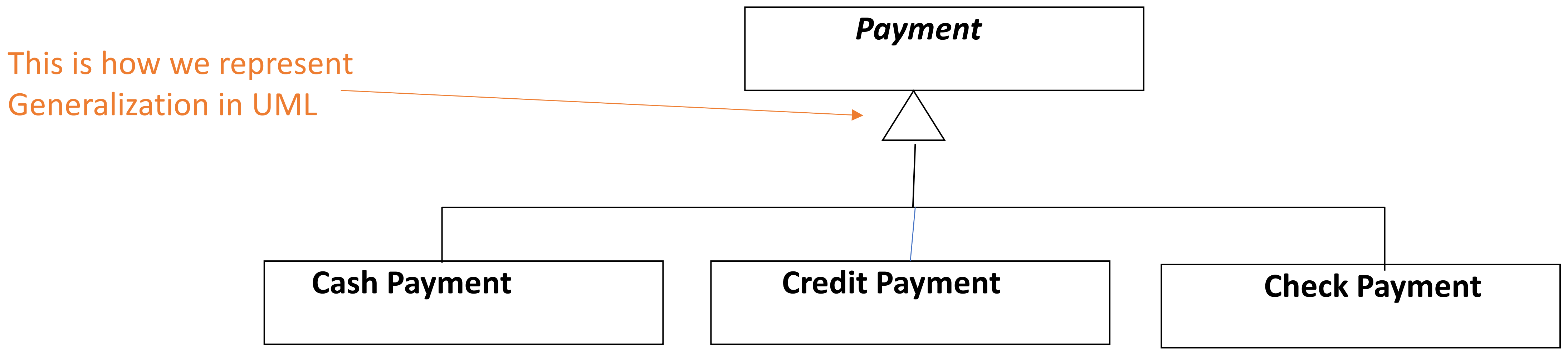
- Focus on “need-to-know” associations
  - High priority associations are strongly “need-to-know”
- It is more important to identify concepts than associations
- Showing too many associations **at this phase** can confuse a domain model

Special Considerations for Associations

- Generalizations
- Association Classes
- Aggregation and Composition

Generalization

- One type of association is Generalization. This type aims to organize concepts to identify commonality among them.)
- The superclass of ***payment*** and the subclasses of *cash payment*, *credit payment*, *check payment*



Generalization (Contd.)

- Generalization creates a conceptual class hierarchy when
  - Subclasses have additional attributes of interest
  - Subclasses have additional associations of interest
  - The subclass concept is manipulated differently than the superclass and other subclasses

Subclass Definition: 100% Rule

- 100% of *superclass'* definition should be applicable to *subclass*

- This includes:
  - Attributes, and
  - Association

- Example:
  - *Payment* has an *amount* and pays for a *sale*
  - This is true for *CreditPayment*, *CashPayment*, *CheckPayment*

When to define subclass

- **Subclass has additional attributes of interest**
  - Library*: Book*, a subclass of *Loanable* resource, has ISBN attribute
- **Subclass has additional associations of interest**
  - Register: *CreditPayment*, a subclass of *Payment*, is associated with a *CreditCard*
- **Subclass is operated, handled or acted differently than superclass**
  - Register*: CreditPayment*, a subclass of *Payment*, is handled differently than other kinds of payments
- **Subclass behaves differently than superclass**
  - Market Research: *Child*, subclass of *Human*, behaves differently than *Parents* with respect to sleep habits

Abstract Conceptual Classes

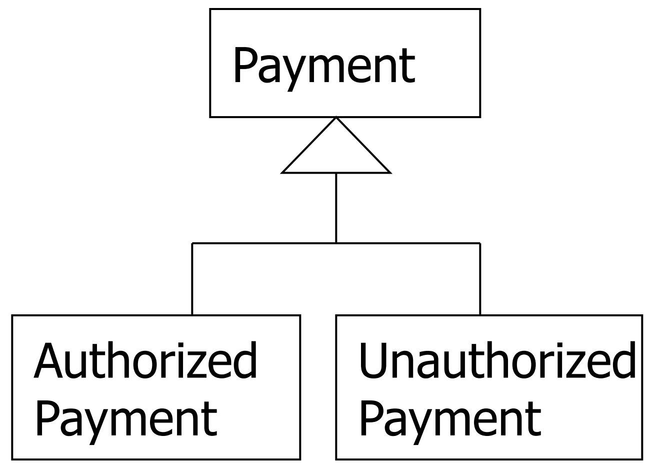
- If every member of a superclass must also be a member of a subclass then that superclass is an abstract superclass
  - An abstract class cannot be instantiated
- Payment class in our previous example is an abstract class
- In UML, the name of an abstract class is italicized

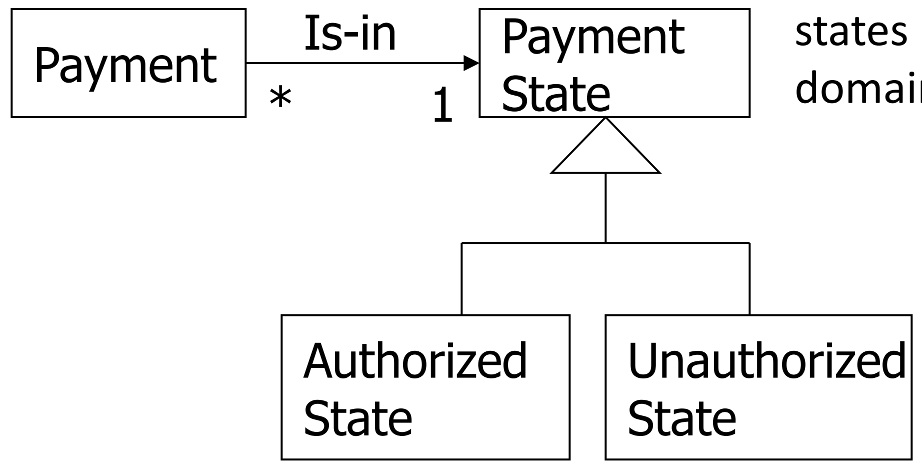
Modeling Changing States

- Do not model states of a concept as subclass of X. Rather:
  - Define a state hierarchy and associate the states with X, or
  - Ignore showing states of a concept in the conceptual model; show them in state diagrams

Example: Payment States

Avoid This Situation

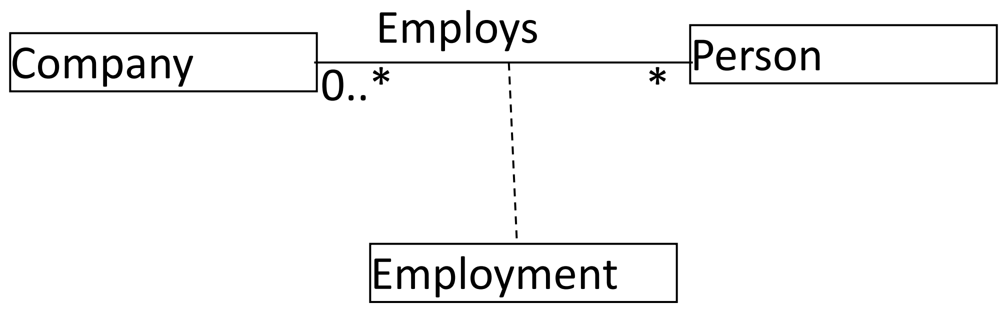




This is a better way to represent states in a domain model. Or ignore showing the states in the domain model.

Association Classes

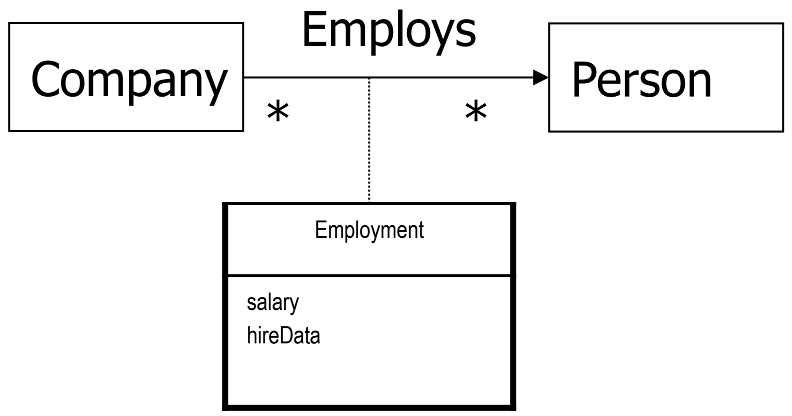
- Let's consider an example using a "Person" and "Company" scenario”.
- Assume you have a system where individuals can be employed by multiple companies, and for each employment, there is additional information such as the role or position and the start date.
- In this case, you can use an association class to represent the relationship between "Person" and "Company" with the added information about the employment.

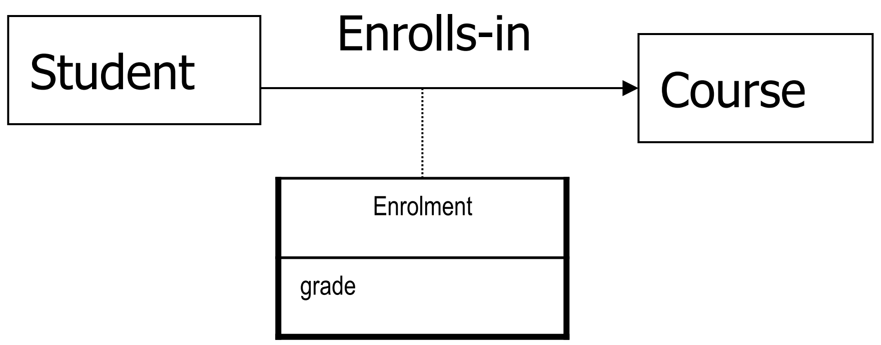


Association Classes (Contd.)

- Create an association class if
  - An attribute is related to an association.
  - Instances of the association class have a life-time dependency on the association.
  - There is a many-to-many association between two concepts, and information associated with the association itself

Associative Classes: Examples





Aggregation

- It is a kind of association used for modeling whole-part relationships
  - Shown in UML as a hollow diamond symbol at the “whole end” of the whole-part association
  - No semantic difference to a regular association, which is why Larman avoids it.

Composition

- It is a specialized form of aggregation
  - Shown in UML as a filled diamond symbol at the “whole end” of the whole-part association

  - More meaningful than aggregation.
    - The parts are “owned” by the whole…they are often created together, they definitely die together, and operations applied to the whole propagate to the parts.

Recommended Exercise

Create a UML domain model for the Library System Use Case (available in Supplementary Reading)

Readings , Assignments, Project Tasks

Read:

- Chapters 7, 8 of the textbook.

Assignments & Project Tasks:

- Assignment 1 is open (Due on Friday 7/2). This is an <u>Individual Submission</u>.

Questions?