Domain Models

CSE 3311 & 5324
Christoph Csallner
University of Texas at Arlington (UTA)

Motivation

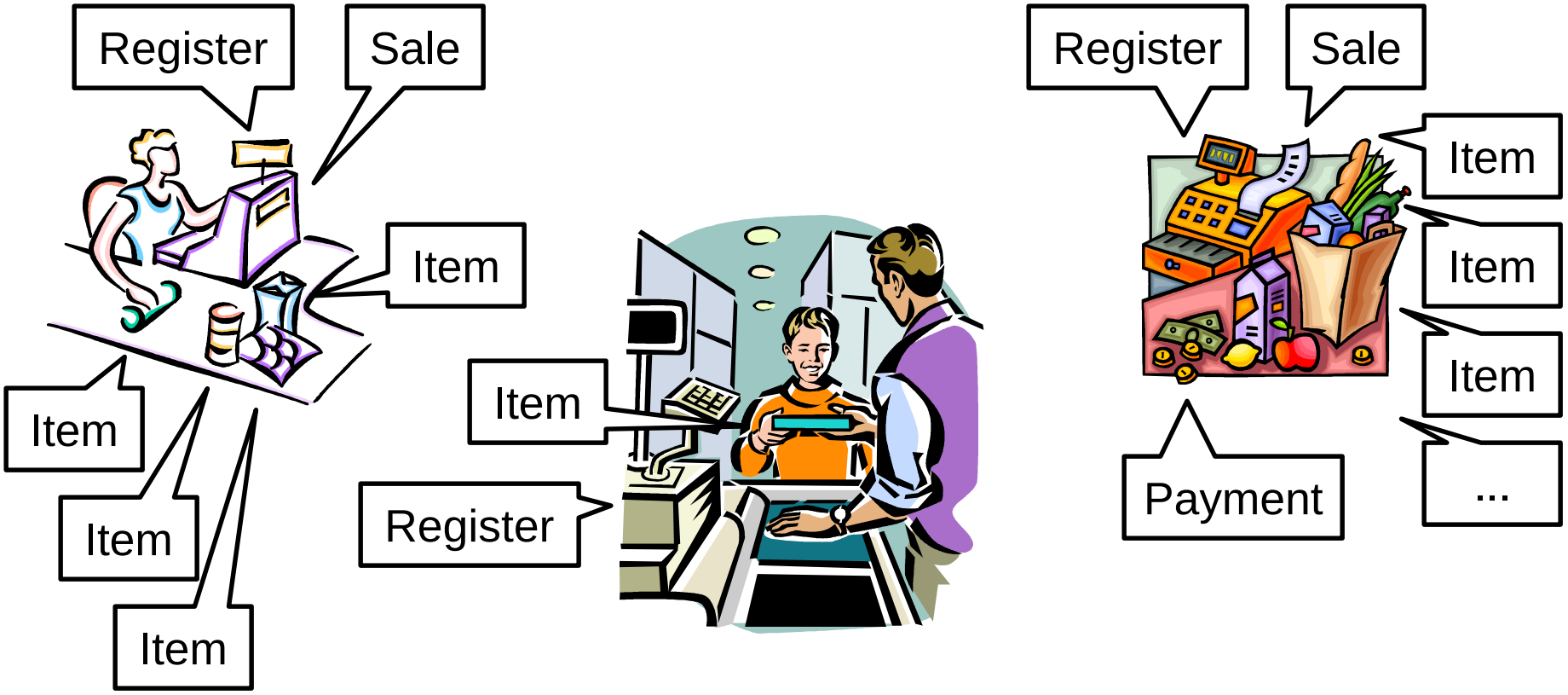
- We expect application to interact with our world

- Application has to maintain an **up-to-date model of the current state** of our world, e.g.:
  - POS system has to sum up the customer's actual items the customer wants to buy today
    - Don't add in items from earlier shopping trips
  - POS system has to calculate with current tax rate
    - Don't use old (outdated) tax rate

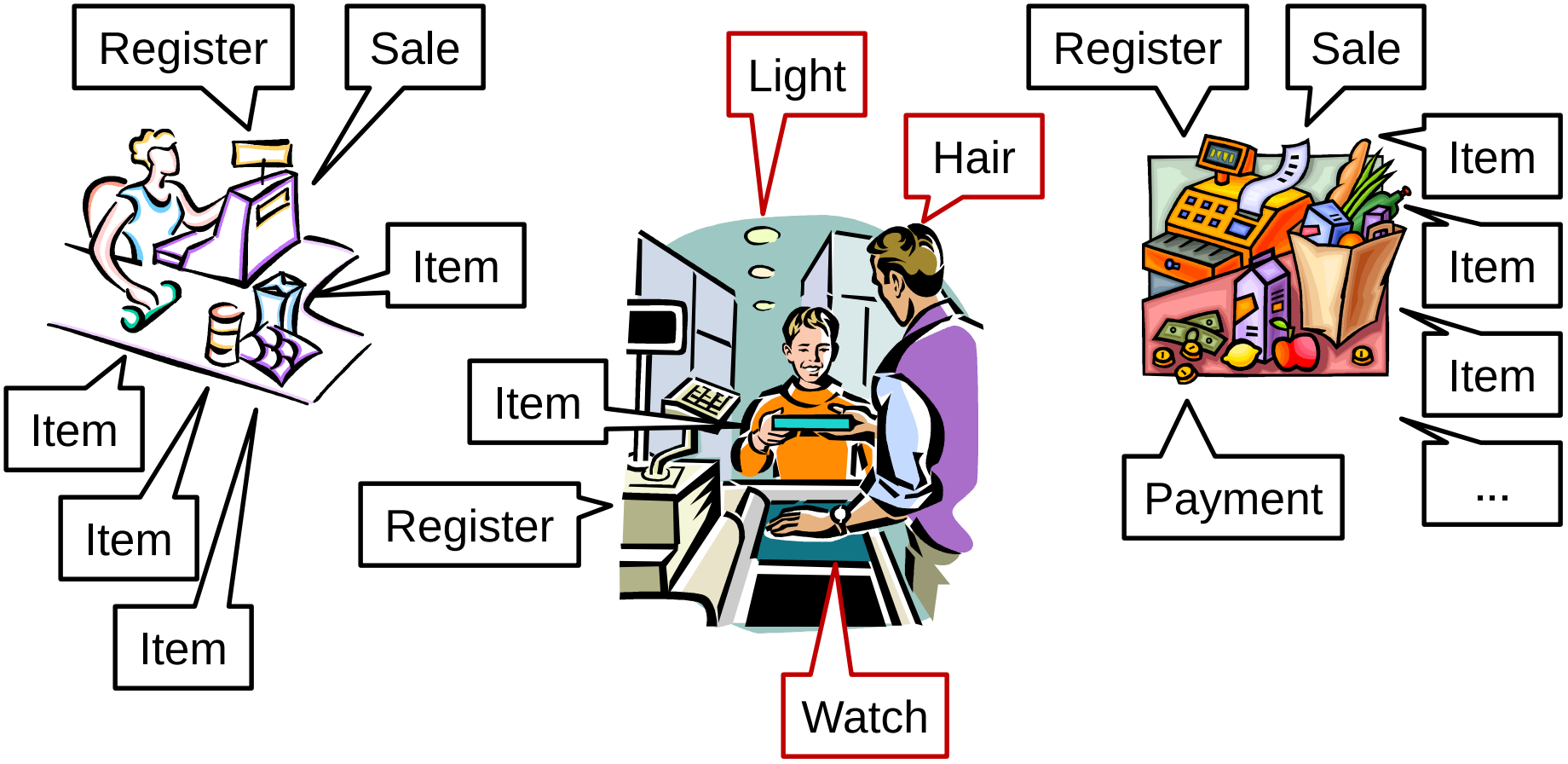
Domain Models: CL, Chapter 9

- Many names for “domain model” (CL, pp. 133-134)
  - Conceptual model, conceptual perspective model
  - Conceptual **entity-relationship model**
  - Domain object model, analysis object model
- Simple idea:
  1. Concepts / things / **entities** of problem / project domain
  2. Relationships between (**relations** on) problem / project domain concepts / things / entities
  3. Constraints on entities and relations
  4. **That’s it!**
     - At this point we don’t think about implementation / code

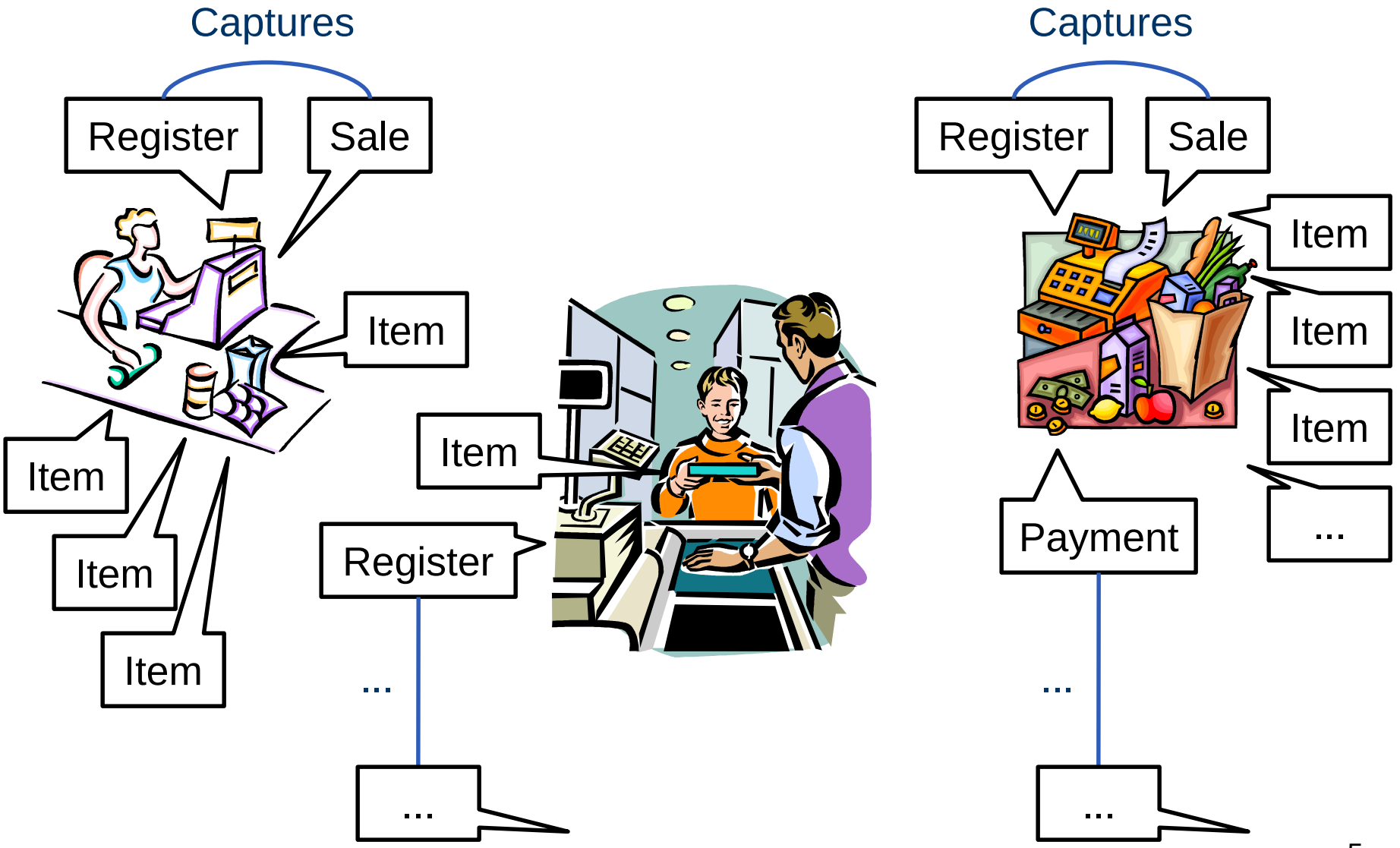
Example: Things in Point of Sale (POS) Domain



Which Things Do We Care About in a Given Project?



Some Relationships Between Things



Example POS Domain Model

- Represent all registers by Register set
- Represent all sales by Sale set
- **Represent each set with a box**

- Represent all “register—sale” relationships by “Captures” relation
- **Represent relation with a connection**

UML Class Diagram

- Class (box)
  - Named, e.g.: “Sale”
  - Here we don’t think of it as a Java, C#, C++, etc. class

- Association (line)
  - Named, e.g.: “records-current”
  - Connects two boxes
    - Or one box with itself
    - Other lines rare in UML

UML Class Diagram

- Remember: A diagram is just a diagram
  - Just boxes and connections

- Use a class diagram to represent, e.g.:
  - **Now: Concepts in requirements analysis**
  - Later use: Classes in object-oriented programs

- **Do not automatically think “code” when you hear “class diagram”**
  - We do not talk about Java / C# / etc. classes here

Recap: Map World to Class Diagram

- Map real world problem / project domain to UML class diagram
  - **Project domain → class diagram**

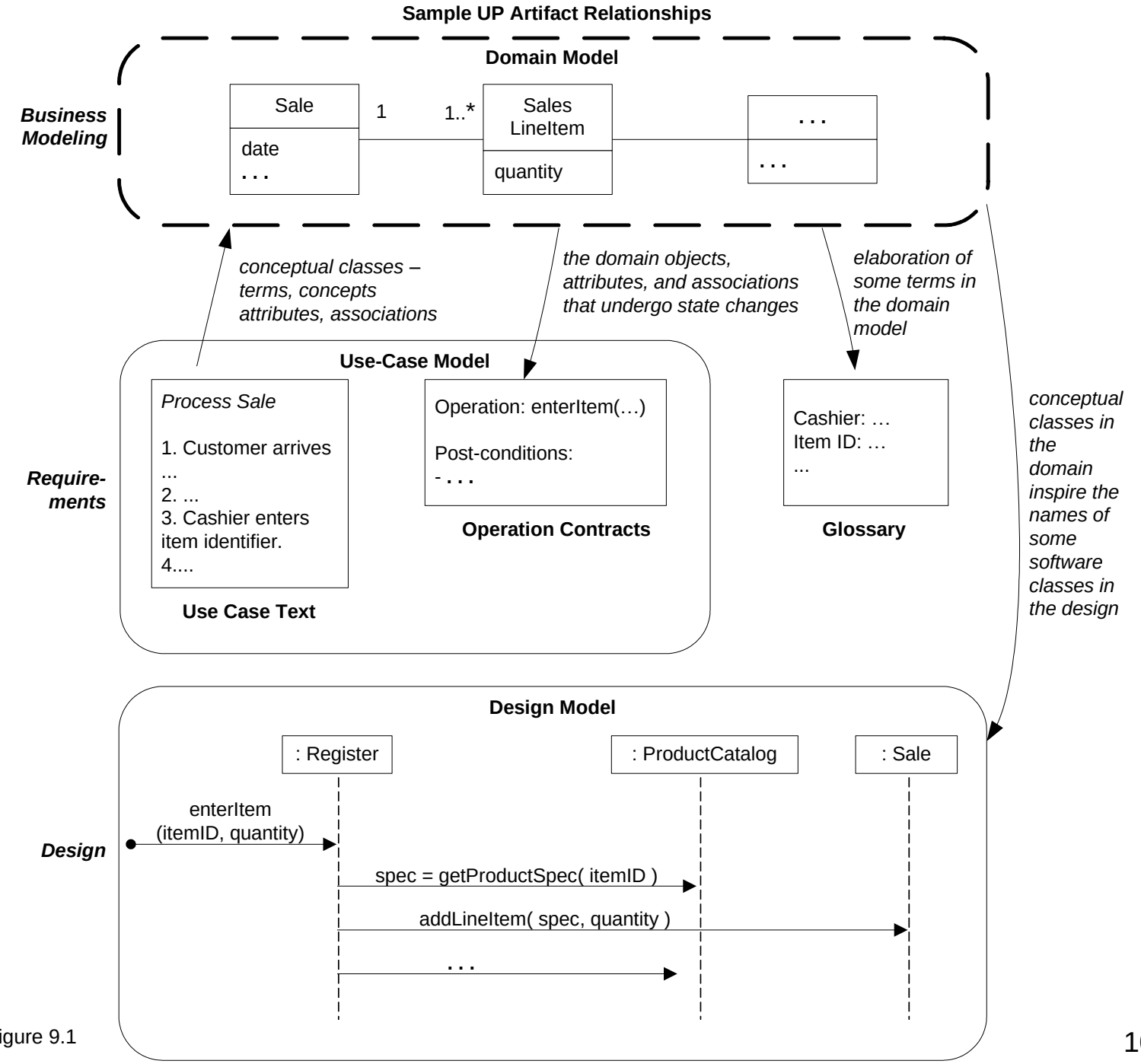
- Problem / project domain concepts / things / entities
  - **Set of similar items in domain → box**

- Relationships between (relations on) problem / project domain concepts / things / entities
  - **Relationship between sets of domain items → line**

Class Diagram

Use-Cases

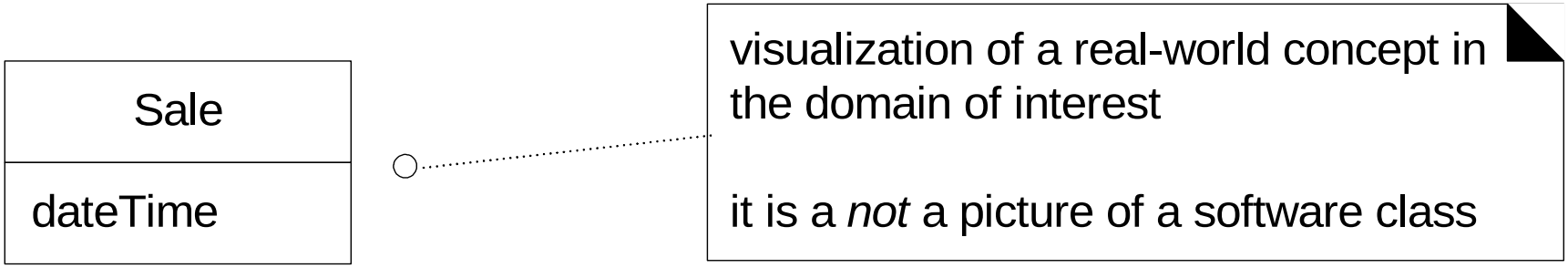
Sequence Diagram



CL Figure 9.1

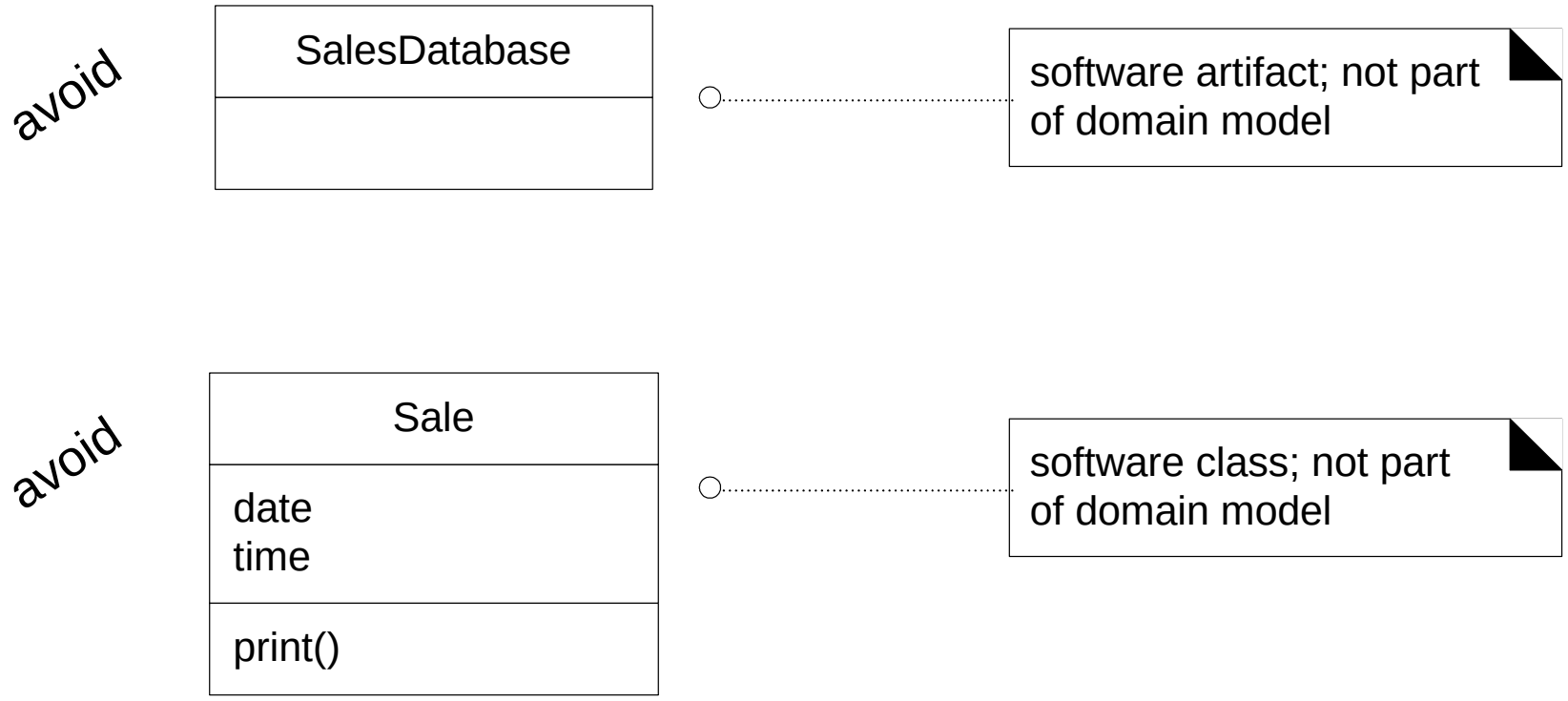
Documenting a Set / Concept Box

- Example from the text book:
  - **Name**: Sale
  - **Definition**: The event of a purchase transaction, has a date and time
  - **Examples**: Example sales in the domain
    - (CL talks about all sales in the universe, which is confusing, unless they mean the universe of the domain)



CL Figure 9.3

Examples of Sets / Concepts to Avoid



CL Figure 9.4

Later: Map Domain Model to Code

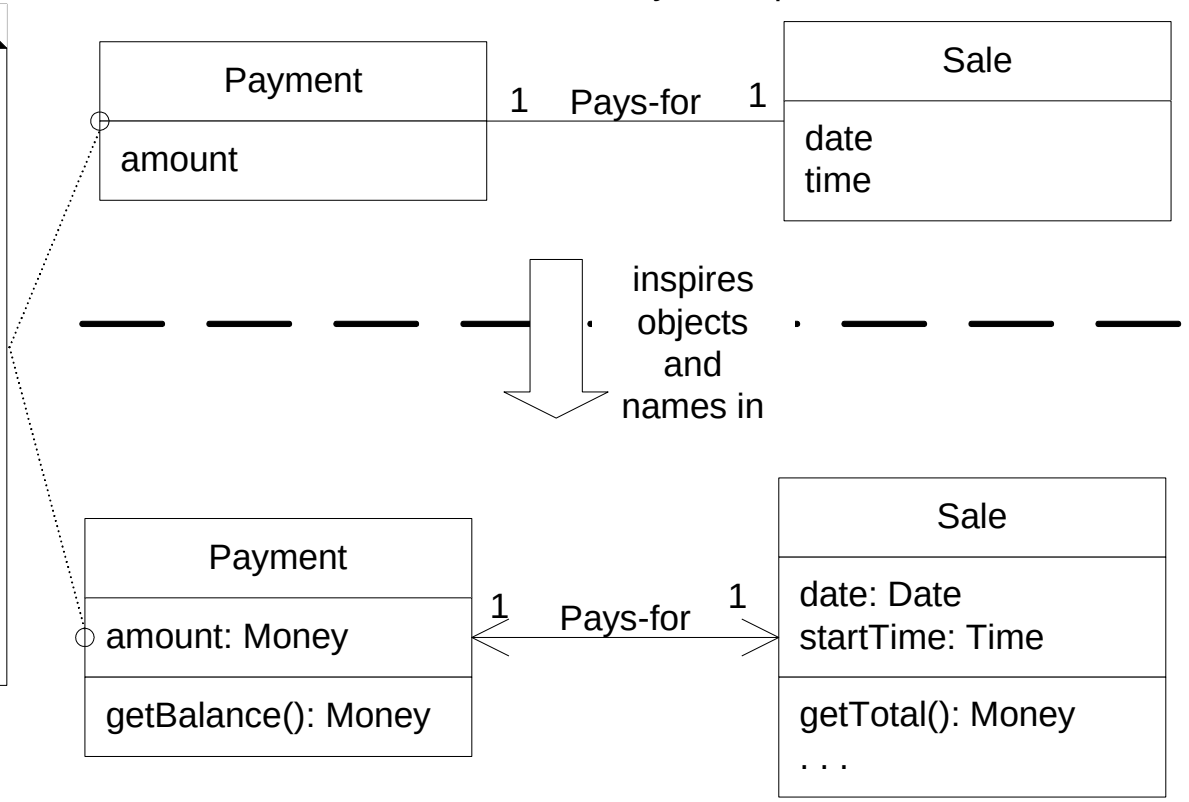
A Payment in the Domain Model is a concept, but a Payment in the Design Model is a software class. They are not the same thing, but the former *inspired* the naming and definition of the latter.

This reduces the representational gap.

This is one of the big ideas in object technology.

UP Domain Model

Stakeholder's view of the noteworthy concepts in the domain.



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.

CL Figure 9.5

How to Create a Domain Model?

1. Find the conceptual classes
   - Reuse or modify existing models
   - Use books on domain model patterns (CL, page 139)
   - Use a category list (next slides)
   - Identify noun phrases (next slides)

2. Draw the conceptual classes in a UML class diagram

3. Add associations and constraints

Use a Category List

- Look at an existing list of concept categories, e.g.:
  - Business transactions
  - Transaction line items
  - Product / service related to transaction (line item)
  - Where is the transaction recorded?
  - Roles of people or organizations related to the transaction
  - Actors in the use case
  - Place of transaction / service
  - Events, with time & place, we need to remember
  - Physical objects

- This is the example list from CL, pp. 140—141

Use a Category List

- Example list: CL, pp. 140—141 (continued):
  - Description of things
  - Catalogs
  - Containers of things (physical or information)
  - Things in a container
  - Other collaborating systems
  - Records of finance, work, contracts, legal matters
  - Financial instruments
  - Schedules, manuals, documents that are regularly referred to in order to perform work

Identify Noun Phrases

- Nouns and noun phrases in domain descriptions
- Roughly:
  Noun in use-case / other requirement document
  →
  Conceptual class in domain model

- Deal with imprecision of natural language
  - Two nouns in use-cases may refer to same concept
    → Detect and fix use-case
  - Same noun in two use-cases may refer to different concepts
    → Detect and fix use-case

IN-CLASS EXERCISE: IDENTIFY DOMAIN CONCEPTS

Identify Domain Concepts / Entities

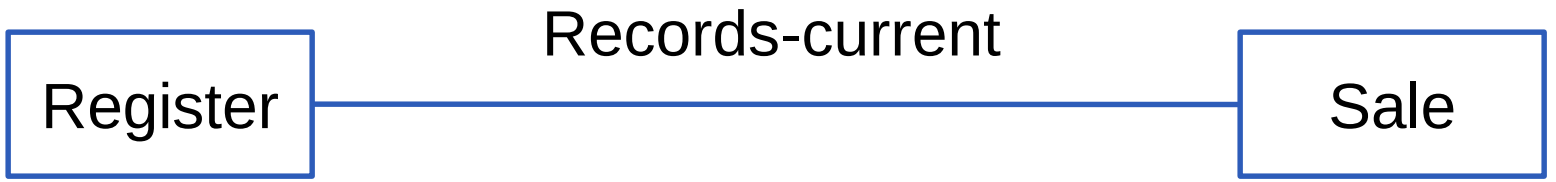
- Get together with your team
- Identify the entities / conceptual classes in your project's domain model
  - Use case actors
  - Nouns in uses cases
  - Business transactions
  - Events we need to remember
  - Physical objects, collaborating systems
- Draw each conceptual class as a box in a UML class diagram
- Be prepared to present your results

Guideline: Avoid Attributes

- Avoid attributes
- Avoid attributes
- **Avoid attributes**
- Why?
  - Because it will simplify your life
- If you add an attribute A to a conceptual class C ...
  - ... you implicitly add a new kind of relationship between A and C
  - This relationship is similar to, but different from, the explicit “line” relationship between conceptual classes
  - **Why two kinds of “relationship” when one is enough?**

Association

- Relationship between instances of one or more classes
- Example:



Association Name

- Name = descriptive **verb phrase**
  - **Class name – *verb phrase* – class name**
- Example: **Sale *paid by* cash payment**
  - Bad (why?) example: Sale uses cash payment

- Name reading direction
  - By default left-to-right
  - Arrow is optional
  - At this point: No code meaning such as field access etc.

How to Find Associations?

- Look at list of common association categories, e.g.:
  - Two transactions are related
  - A is a line item of transaction B
  - A is a product or service for transaction (line item) B
  - A is a role related to transaction B
  - A is a physical or logical part of B
  - A is physically or logically contained in B
  - A is a description of B
  - A is known/logged/recorded/reported/captured in B

- Above is example list from CL, page 155

Common Association Categories

- (Continued)
  - A is a member of B
  - A is an organizational subunit of B
  - A uses or manages or owns B
  - A is next to B

Guideline: Include Association X?

- Is X a **“Need-to-Remember Association”**?

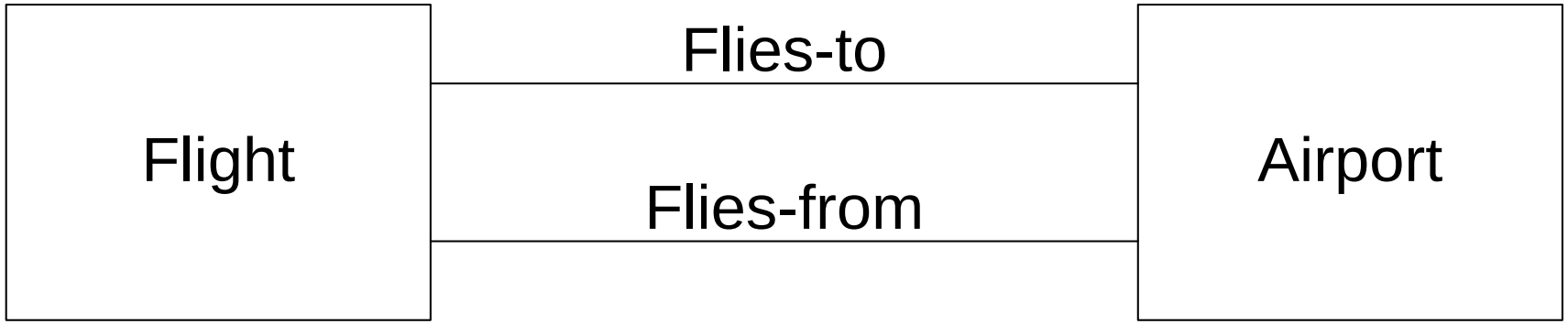
- Does system need to **preserve knowledge of the relationship for some time?**
  - Include those associations in the domain model

- Avoid adding other relationships that may exist in the domain but are less important
  - Don’t want to flood your models

Multiple Relationships between A, B

- More than one association between conceptual classes A and B

- Example:



CL Figure 9.16

IN-CLASS EXERCISE: CREATE YOUR TEAM'S DOMAIN MODEL

Create Your Team's Domain Model

- Get together with your team
- Create a draft of your domain model
  1. Find your conceptual classes (e.g.: "Register", "Sale")
  2. Find the associations between your conceptual classes (e.g.: "Register records current Sale")
  3. Draw a UML class diagram that contains one box per conceptual class and one line per association
- Be prepared to present your results

For the following, both the CL textbook and the UML User Guide 2nd edition may be confusing. I recommend you

- Ask questions
- Look up Entity-Relationship Models in a database book
- Read **Program Development in Java** by Barbara Liskov [BL]

WARNING

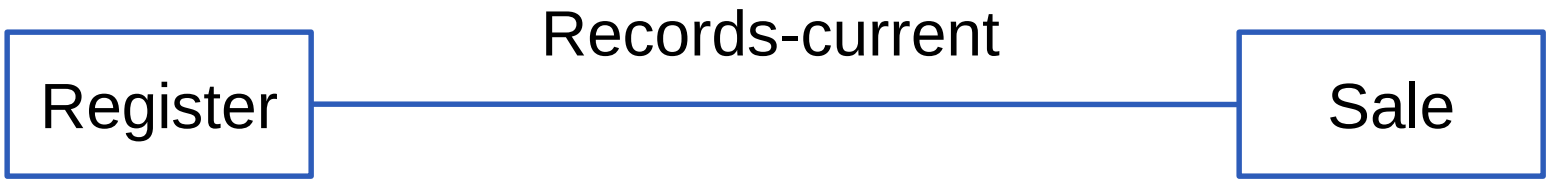
Association Arity

- By default, an association is binary
  - Relationship between instances of one or two classes

- N-ary (n>2) associations exist, but very rare in UML
  - UML User Guide 2nd edition (400+ pages) [UMLUG2] mentions it very briefly
    - 2 lines on page 65
    - 1 line on page 457
    - UMLUG2 claims to be the “definite guide to the use of the UML”

Association = Relation on Classes

- Relationship between instances of one or more classes
- Example:



- Recall that a relation is just a set of tuples
  - Association ≈ Table in a relational database
  - Conceptual class ≈ Column in a relational database
  - Tuple ≈ Row in a relational database

Revisit Definitions: Conceptual Class

- **Set** of similar things, e.g., employees
- Important: Set contains all items **that exist at a particular moment**

  – BL, page 272
- Example: Employee set does not include:
  - Past or future employees
  - "All employees in the universe"
  - Otherwise "Employee" becomes more like "Person"
- May need separate relation for former employees
  - So better to not mix them together into same class

Revisit Definitions: Association

- **Relation** on conceptual classes, e.g., reports-to
- Important: Relation contains exactly the tuples **that exist at a particular moment**

- Example: Reports-to relation does not include:
  - Past employee-manager relationships
  - Future employee-manager relationships
- Relation may have had different constraints in the past (e.g., no employee can have more than one manager)
  - So better to not mix old and current relation together

Revisit Definitions: Domain Model

- We said “concept=set” and “relationship=relation”
- Classic set and binary relation on that set:
  - Natural numbers: {0, 1, 2, 3, ...}
  - Less-than: {(0,1), (0,2), (0,3), ...}

Domain Model

- We said “concept=set” and “relationship=relation”
- Classic set and binary relation on that set:
  - Natural numbers: {0, 1, 2, 3, ...}
  - Less-than: {(0,1), (0,2), (0,3), ...}
- Business-related concept (=set) and relation (-ship):
  - Employee: {Amy, Belinda, Carl}
  - Reports-to: {(Amy, Carl), (Carl, Belinda)}
- Corresponding domain model as a UML class diagram:

Domain Model

- What do we mean with this domain model?
- Should our software only work for the employees Amy, Belinda, and Carl?

Domain Model

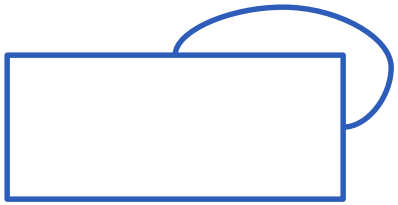
- What do we mean with this domain model?
- Should our software only work for the employees Amy, Belinda, and Carl? No!
  - Employee is not one concrete set, but a **set variable**
  - Allows state to change over time, e.g., to {Amy, Carl}
- Should our software only allow Amy to report to Carl and Carl to report to Belinda? No!
  - Similar to above, Reports-to is a **relation variable**
  - Allows state to change over time, e.g., to {(Carl, Amy)}

- See BL, Section 12.1

Concept

- Put two definitions together:
  - Concept = set variable
  - Set contains all items that exist at a particular moment
- Items in the set may change over time
- Set does not contain all past and future items

- Compare with column in relational database
- Compare with a set variable in a program
- Compare with instances of a class in an OO app



Reports-to

Association

- Put two definitions together:
  - Association = relation variable
  - Relation contains all tuples that exist at a particular moment
- Relationship may change over time
- Relationship does not contain all past and future tuples
- Compare with a table in a relational database
- Compare with an instance field in an OO program
  - At any point during program execution, the field contains a reference to the current field value

Intuitive in Code

```
public class Employee {  // snippet of object-oriented program, e.g., in Java
    protected String name = null;
    protected Employee reportsTo = null;
    public Employee(String id, Employee manager) {
      //..
    }
}
```

- {Amy, Belinda, Carl} means that we currently have three Employee instances
  - May change, e.g., by calling “new Employee(..)”
- {(Carl, Amy)} means that: carl.reportsTo==amy
  - May change, e.g., by setting: carl.reportsTo = null

Other Class Diagram Elements

- Several other elements of UML class diagram are essentially different kinds of constraints on
  - (Concept) set variables
  - Relation variables
- Example: Multiplicity constraint on association
  - 1:N relation, ..

- The important assumption is that
  Each such constraint has to evaluate to true at each point in time

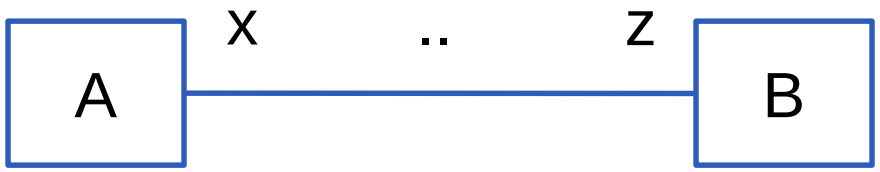
This is a Common Notion

- Each constraint has to evaluate to true at each point in time

- Compare with relational database:
  - Multiplicity constraint: Key constraint, rule, etc.
  - We expect key constraints to hold at any point in time

Important: Multiplicity Constraints

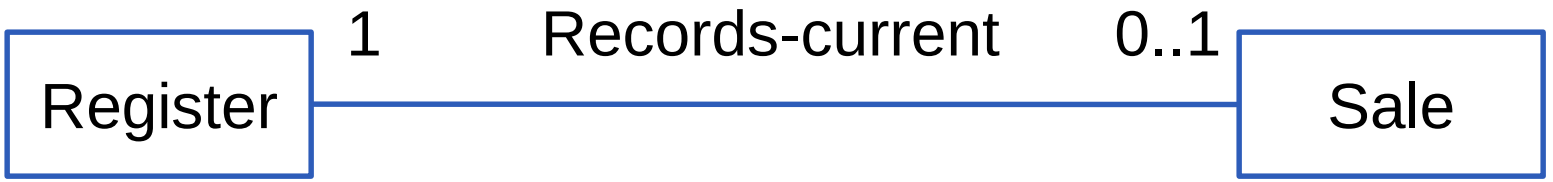
- Use it to restrict (constrain) an association
  - To a function or a special relation
- If present, must be present at each end of line



- **Each item in A maps to z items in B**
- **Each item in B maps to x items in A**
- Each annotation x, z is a range, e.g.: 1..5
  - Collapse single element ranges, e.g.: “1..1” to “1”
- Star (*) means “zero or more”

Example: Register—Sale

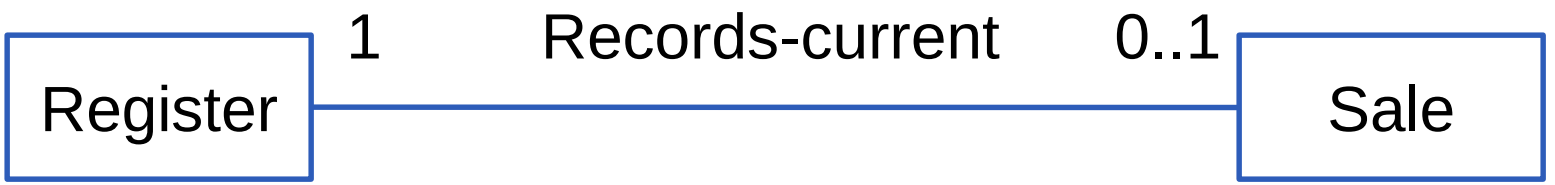
- We want to capture the currently recorded sales
- Records-current relation with multiplicity constraint:



- At any particular moment, a register may record zero or one sales (but not more) → 0..1
- A sale is recorded by exactly one register → 1

Example: Legal Relation State 1

- Arlington Walmart, Feb.16, 4:00:00 am
  - Registers: {1, 2, 3, .., 25}
  - Sales: {s1, s2}

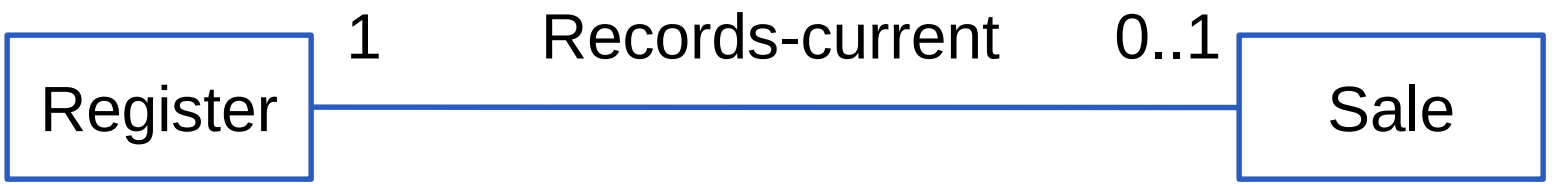


- Records-current:

{
(1, s1),
(3, s2)
}

Example: Legal Relation State 2

- Arlington Walmart, Feb. 16, 4:30:00 am
  - Registers: {1, 2, 3, .., 25}
  - Sales: {s3, s4, s5}

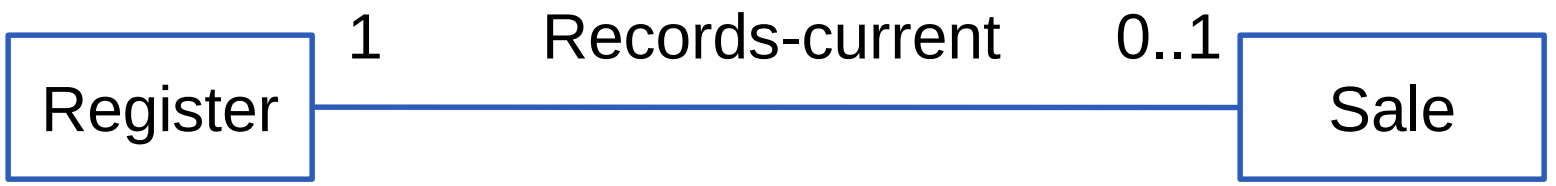


- Records-current:

{
(1, s3),
(3, s4),
(9, s5)
}

Counter-Examples: Illegal States

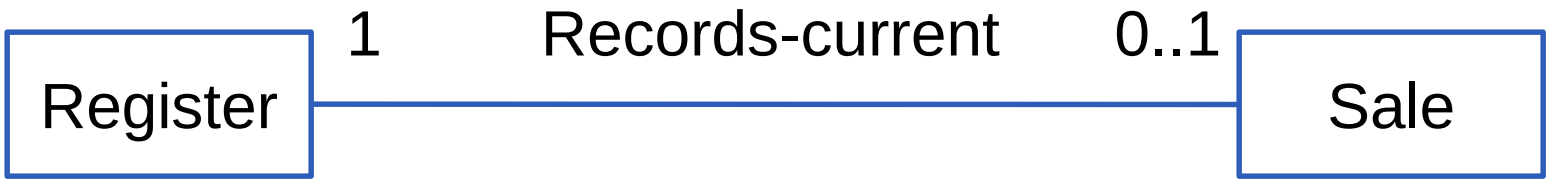
- Arlington Walmart, never
  - Registers: {1, 2, 3, .., 25}
  - Sales:{sa, sb}



- Records-current: Why?
  - {(1, sa), (1, sb)}
  - {(1, sa)}
  - {(1, sa), (2, sa)}

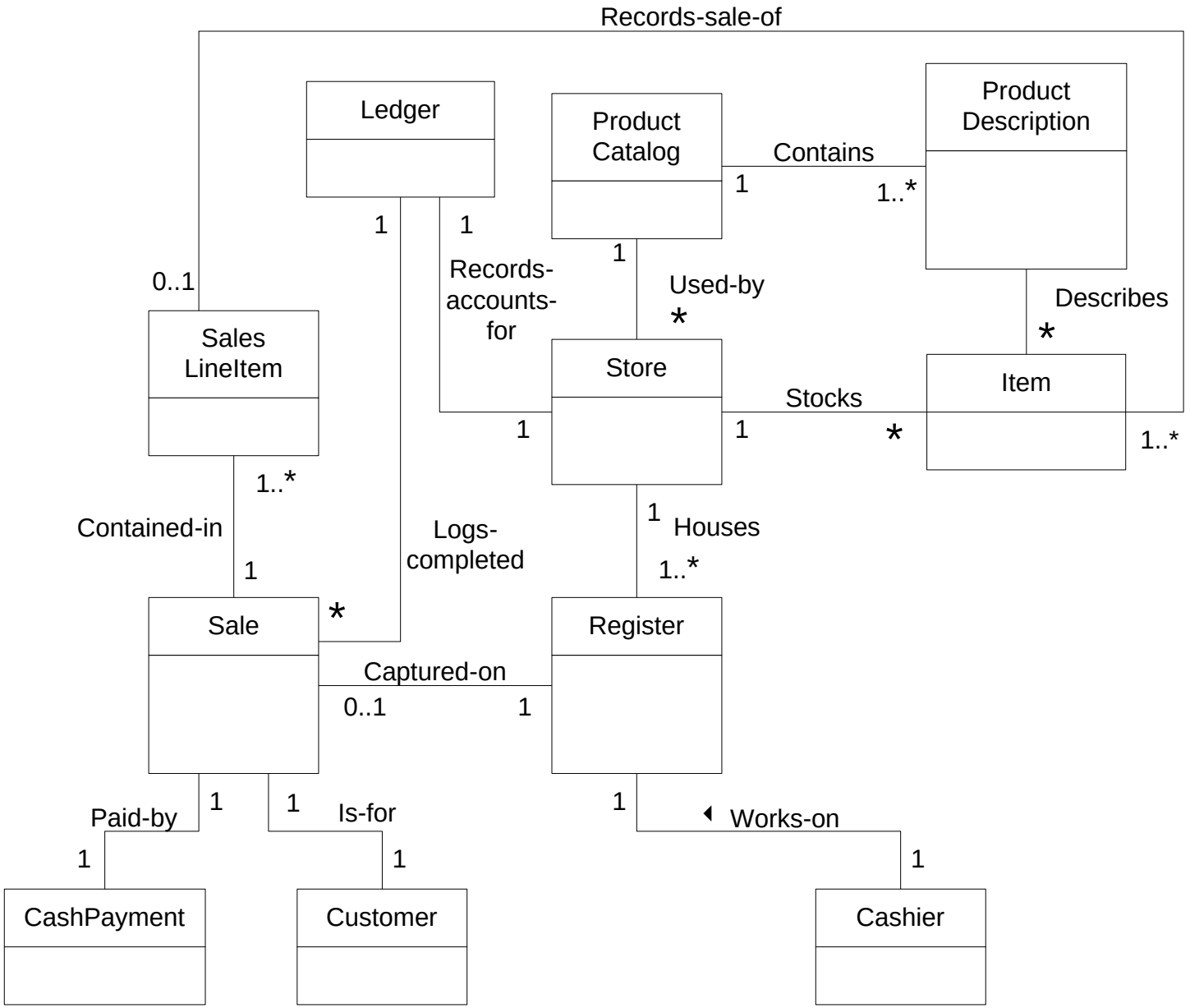
Counter-Examples: Illegal States

- Arlington Walmart, never
  - Registers: {1, 2, 3, .., 25}
  - Sales:{sa, sb}



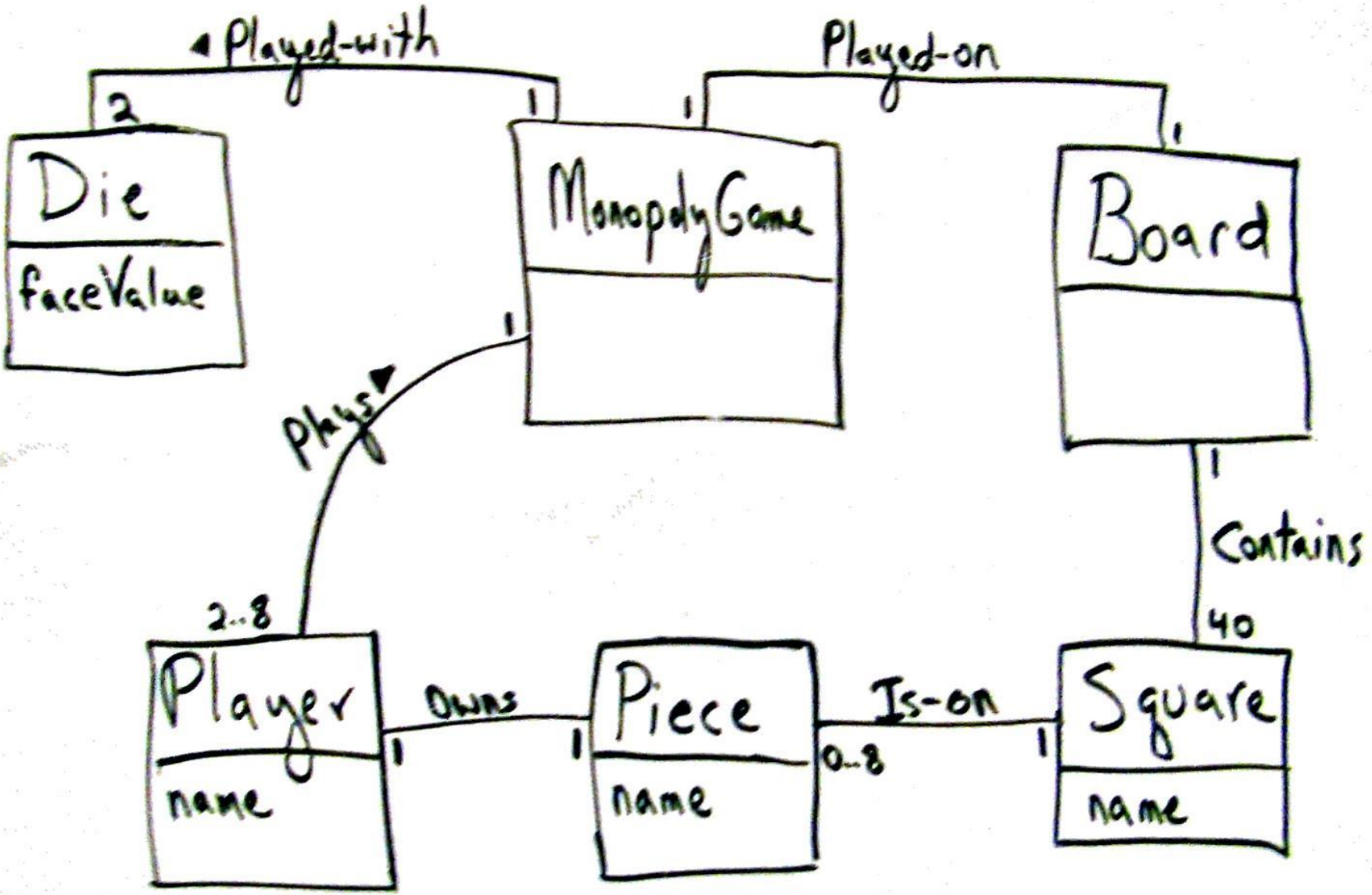
- Records-current: Why?
  - {(1, sa), (1, sb)} Register 1 has to map to 0..1 sales
  - {(1, sa)} Sale sb has to map to 1 register
  - {(1, sa), (2, sa)} Sale sb has to map to 1 register
- Would all be legal without multiplicity constraint!

Example: Point of Sale (POS)



CL Figure 9.17

Example: Monopoly

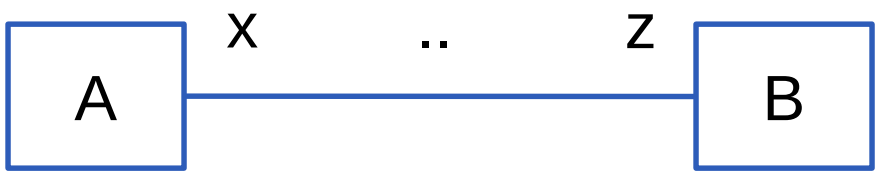


CL Figure 9.18

IN-CLASS EXERCISE: ADD MULTIPLICITY CONSTRAINTS

Add Multiplicity Constraints

- Get together with your team
- Add multiplicity constraints to your domain model:



- **Each item in A maps to z items in B**
- **Each item in B maps to x items in A**
- Each annotation x, z is a range, e.g.: 1..5

- Be prepared to present your results

Things That Should be in the UML

- We have seen a relation constraint: multiplicity
- Where are corresponding constraints on concepts?
  - They are in BL, Section 12.1.1 ;-)
  - Not sure if they are in the UML

- Examples:
  - Fixed: Set membership is fixed (constant set)
  - Size: How many elements the set contains
    - Singleton (1), Java-style enumeration, etc.

CL, Chapter 31, but warning still applies

DOMAIN MODEL REFINEMENT

Subset

- Recall: Conceptual class = Set of similar things
- Sometimes, want to structure set
- Classic way to structure a set: Define subsets

Subset: A ⊆ B

- Sets A, B
- Every element of A is also an element of B
  - A is a subset of B
  - B is a superset of A

  - A is included in B
  - B includes A

  - A and B may be the same

Subset: Is-A Test

- Natural language formulation of subset relation
  - Informal, easy
- Quick first check if two conceptual classes are related by subtype relation

- Examples:
  - CashPayment is-a Payment
  - CreditPayment is-a Payment
  - CheckPayment is-a Payment

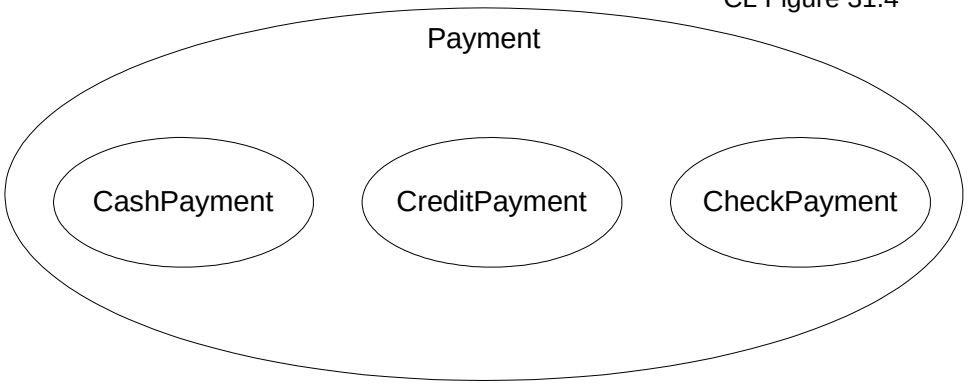
Example: Cash payment ⊆ Payment

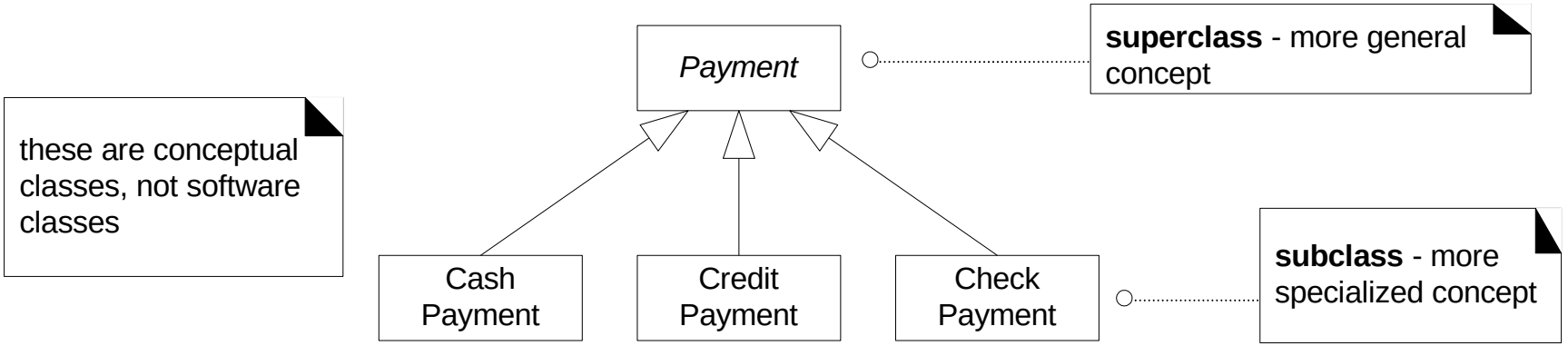
- Conceptual classes: Payment, Cash payment
- Every Cash payment is also a Payment
- Cash payment is a subset of Payment
- Etc.

Class Diagram Terminology

- (Generalization-specialization) class hierarchy
  - Subset relation on conceptual classes
- Superclass
  - Superset
  - Example: Payment
- Subclass
  - Subset
  - Example: Cash payment

CL Figure 31.4





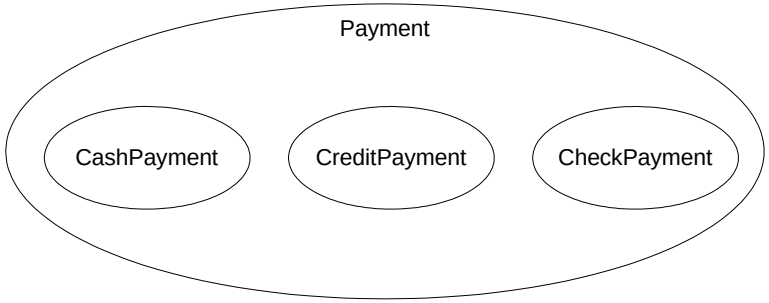
CL Figure 31.1

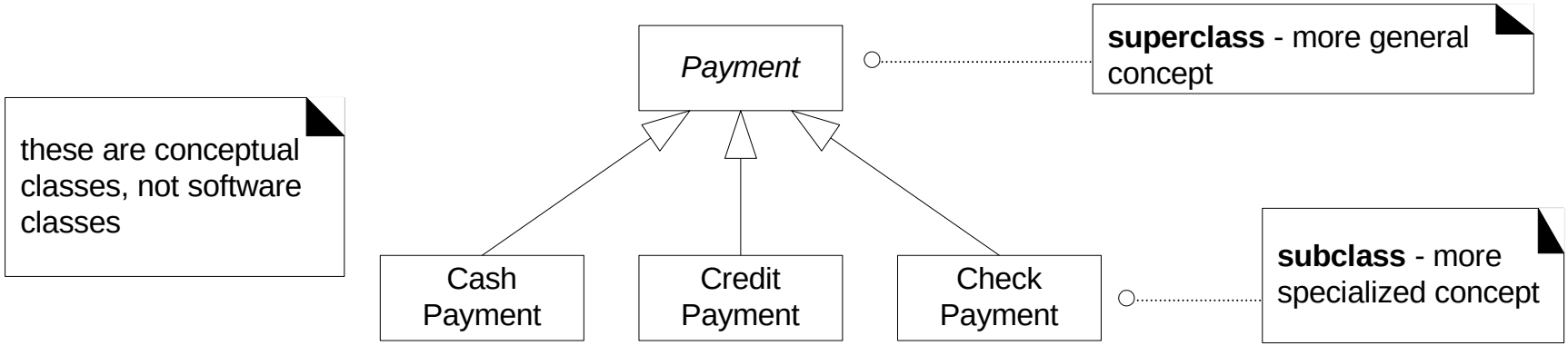
Class Hierarchy

- Subset relation on conceptual classes
- Example:

{
(CashPayment, Payment),
(CreditPayment, Payment),
(CheckPayment, Payment)
}

CL Figure 31.4





CL Figure 31.1

Subset Relation – OO Subtype Relation

- Compare with subtype polymorphism in object-oriented programming languages, e.g.:
- Cash payment ⊆ Payment
  - Conceptual classes: Cash payment, Payment
  - Every Cash payment is also a Payment
  - **Cash payment is a subset of Payment**
- Java: class CashPayment extends Payment {..}
  - Types (classes): CashPayment, Payment
  - Every CashPayment instance is also a Payment instance
  - **CashPayment is a subtype (subclass) of Payment**

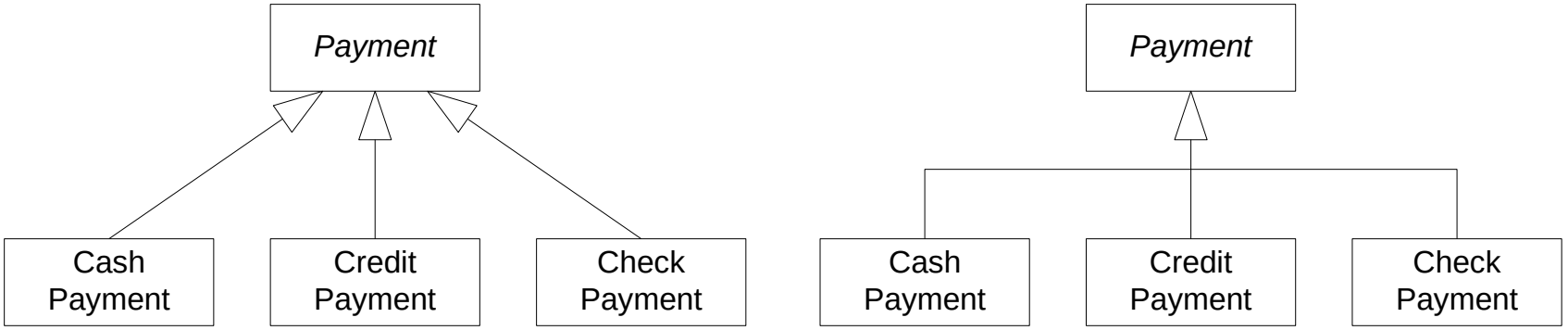
Summary

- Conceptual class = set
- Subset relation
- Subset relation → Class hierarchy

- OO class defines a set of instances
- Subtype relation can be seen as subset relation
  - Subtype polymorphism
- Subtype relation → Type hierarchy

UML Class Diagram Notation

- $A \subseteq B$
  - A is a subset of B
  - Directed edge from A to B
  - Large hollow triangle
  - Edge may be shared with $C \subseteq B$ etc.
  - Edge sharing just layout (unlike BL, see BL page 273)
- Example:



CL Figure 31.2

IN-CLASS EXERCISE: ADD SUBCLASSES

Add Subclasses

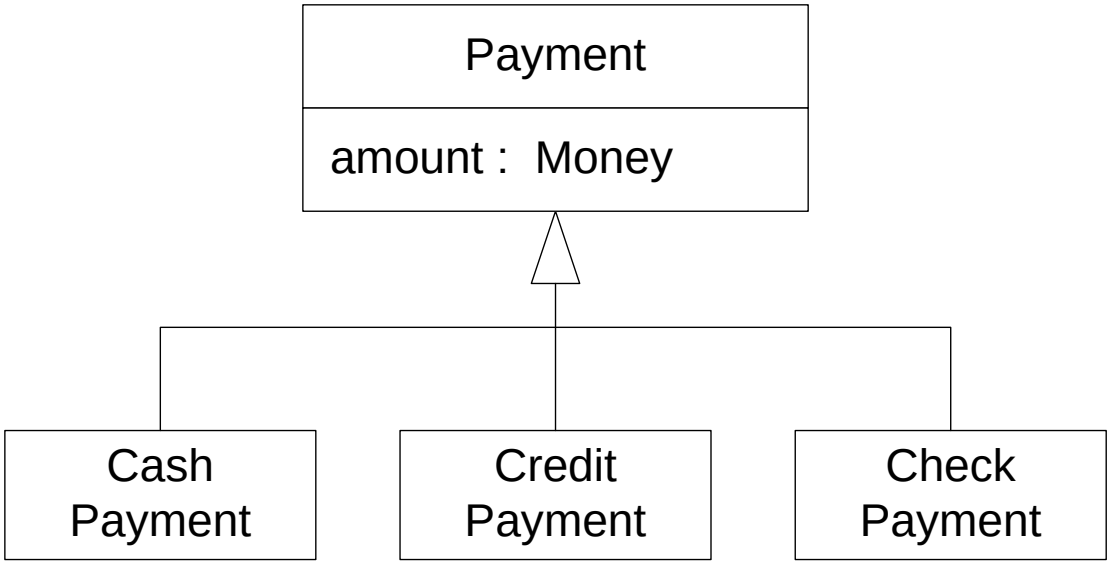
- Get together with your team

- Refine your domain model by adding conceptual subclasses
  - Pick a class (box) that seems complex and plays different roles
  - Just for the purpose of this exercise -- it does not commit your team project to adapt it

- Be prepared to present your results

Subclass “Inherits” Attributes

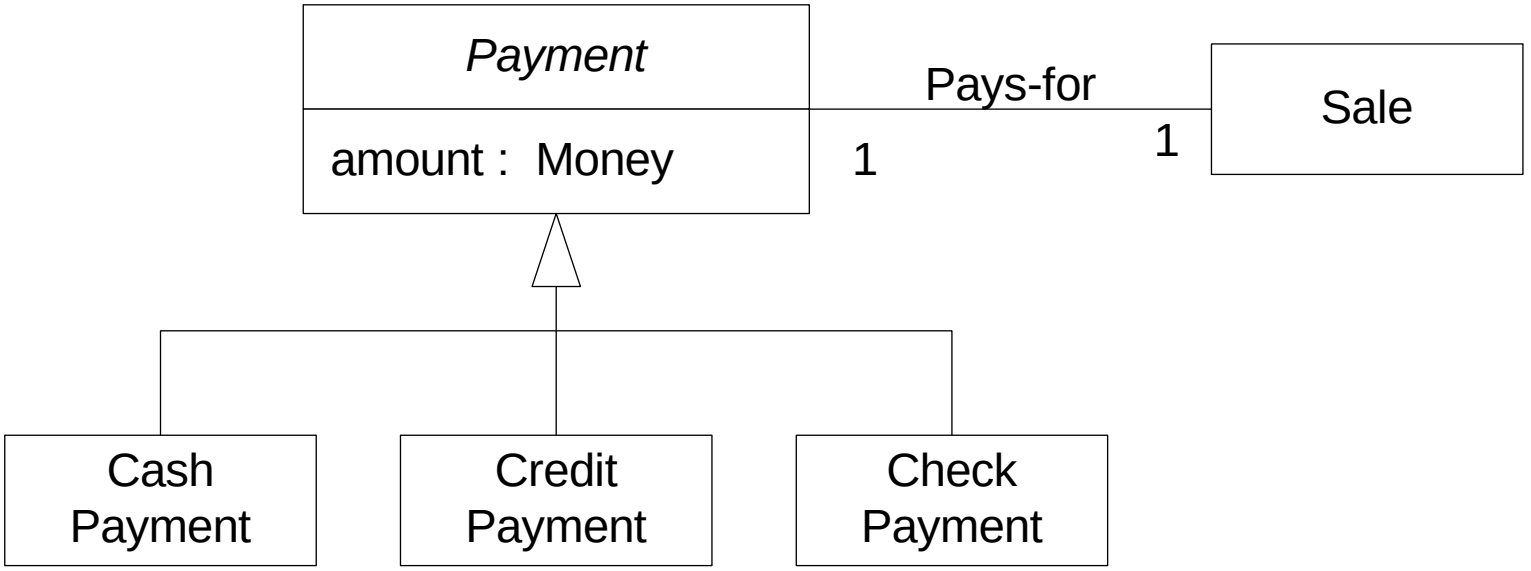
- Recall: Class hierarchy = Subset relation
  - Each element of subclass is an element of superclass
- Each payment has an amount attribute → Each element of each subclass of Payment has an amount attribute



CL Figure 31.5

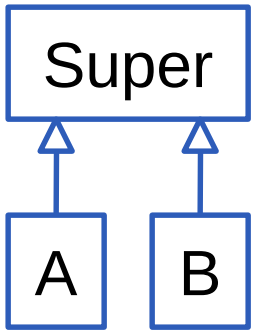
Subclass “Inherits” Associations

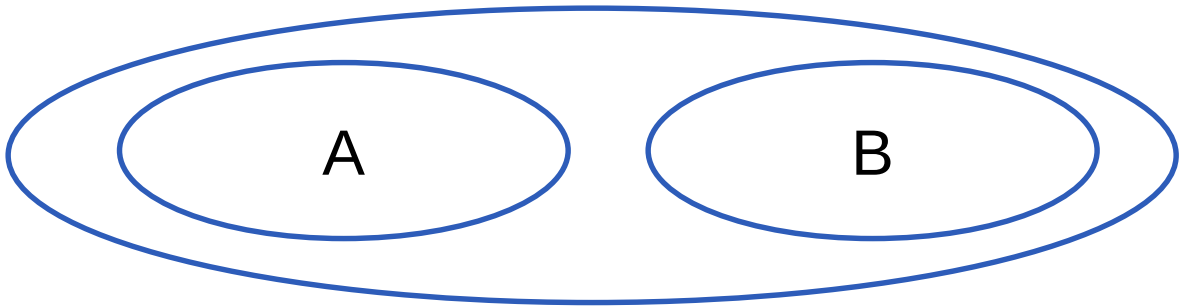
- Recall: Class hierarchy = Subset relation
  - Each element of subclass is an element of superclass
- Each payment is associated with one sale → Each element of each subclass of Payment is associated with one element of Sale



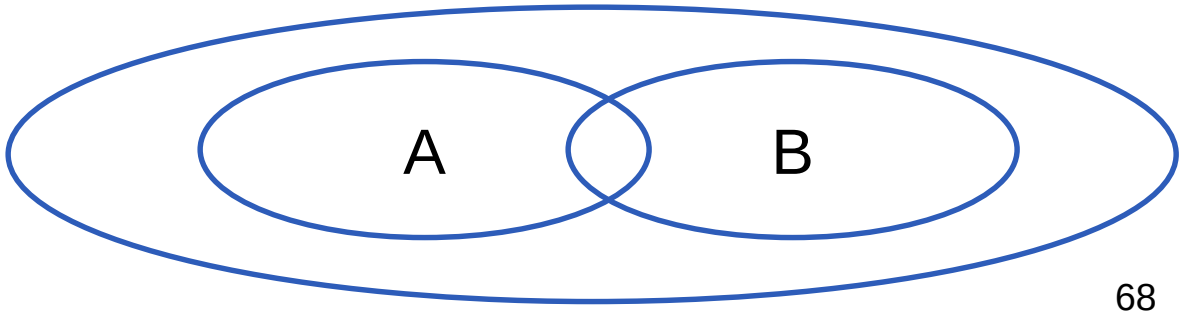
CL Figure 31.5

Are Subsets Mutually Exclusive?



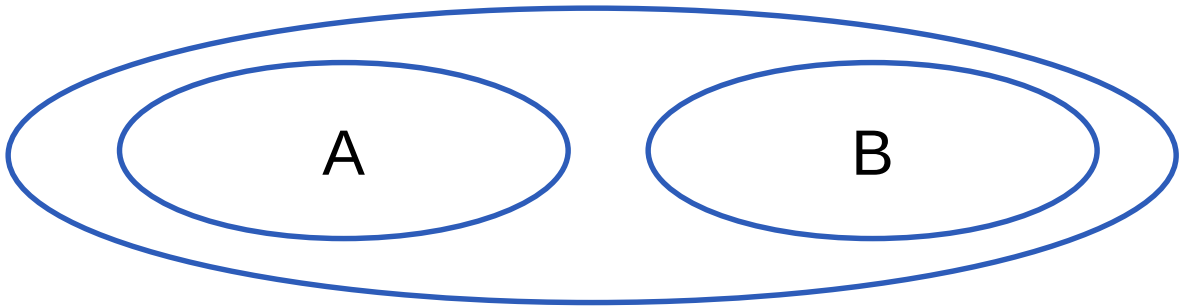


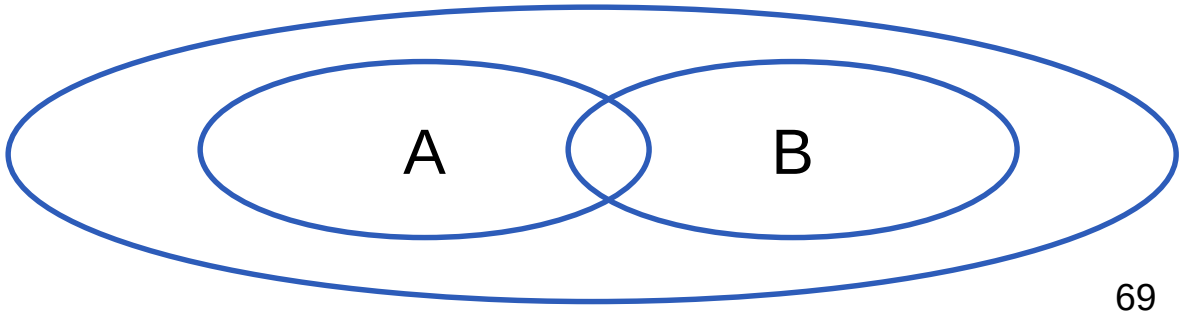
- Or:



Are Subsets Mutually Exclusive?

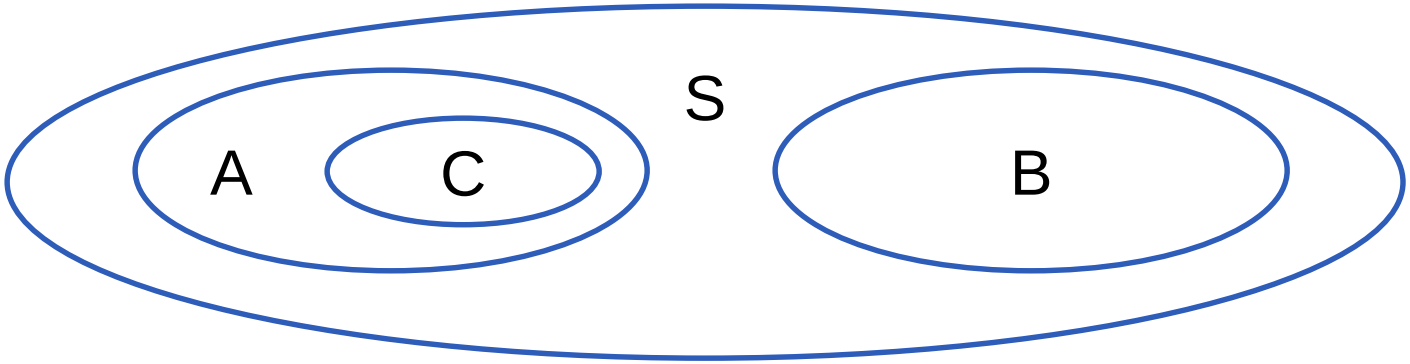
- Both possible in Java
- How?





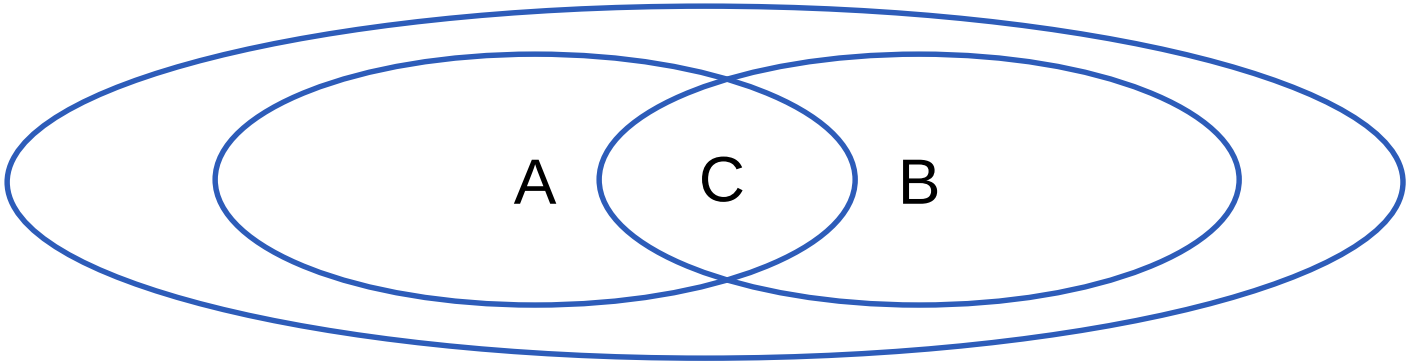
Mutually Exclusive Subsets

- Possible in Java
- Single class inheritance (sub-classing)
  - class A extends S {..}
  - class B extends S {..}
  - Instance of A cannot be an instance of B
  - Member of set A cannot be a member of class B
  - class C extends A {..} // but not B



Overlapping Subsets

- Possible in Java
- Multiple interfaces inheritance
  - interface A extends S {..}
  - interface B extends S {..}
  - Instance of A can also be an instance of B
  - Member of set A can also be a member of set B
  - class C implements A, B {..}

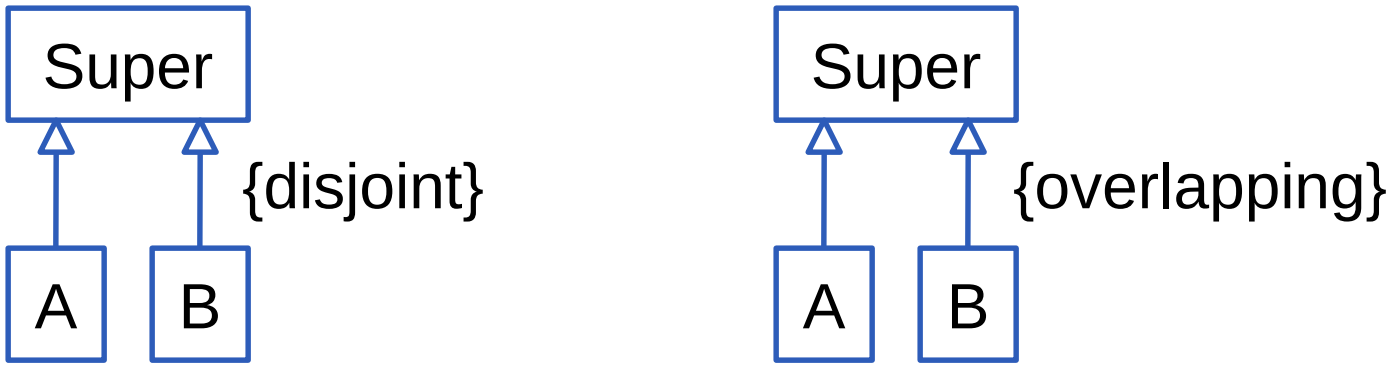


Are Subsets Mutually Exclusive?

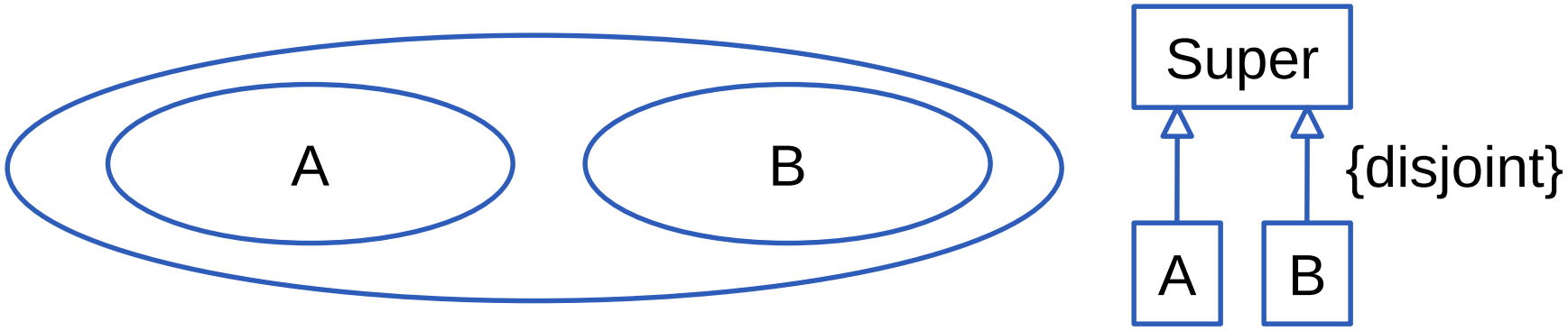
- Both possible in Java
- How about UML?

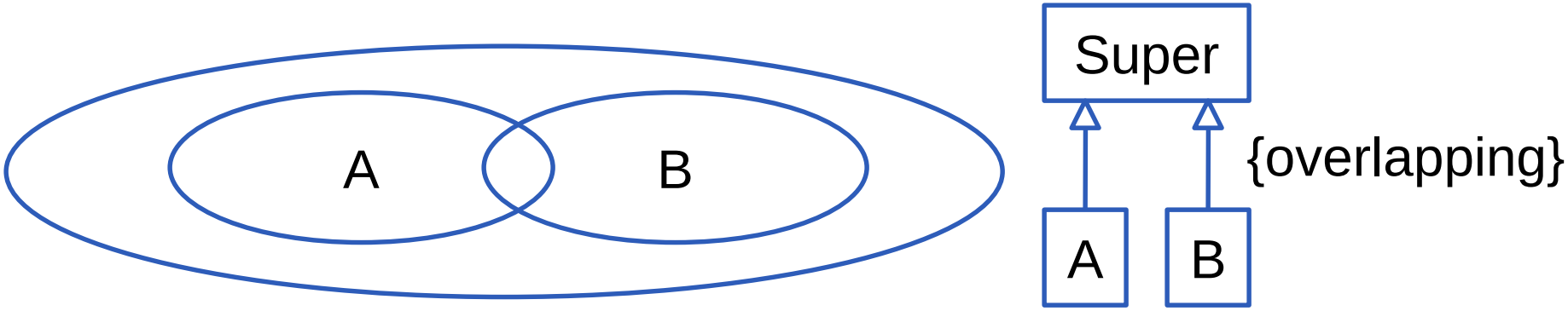
Are Subsets Mutually Exclusive?

- Both possible in Java
- Both possible in UML
- Default: disjoint (mutually exclusive)
  - CL, page 514: Subclasses disjoint (mutually exclusive)
  - UMLSS version 2.1.2, Section 7.3.21
    - UMLSS = UML “Superstructure” Specification
    - http://www.omg.org/spec/UML/2.1.2/
- Constraint on subclass arrow:



Are Subsets Mutually Exclusive?

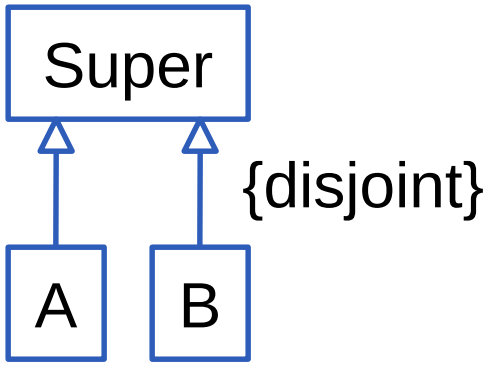


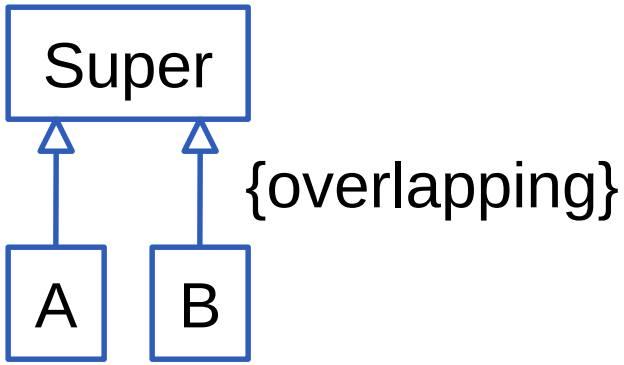


IN-CLASS EXERCISE: REFINE SUBCLASS RELATION

Refine Subclass Relation

- Get together with your team
- Refine your domain model's subclasses by adding constraints that classify your subtype relations as **{disjoint}** or **{overlapping}**
- Be prepared to present your results

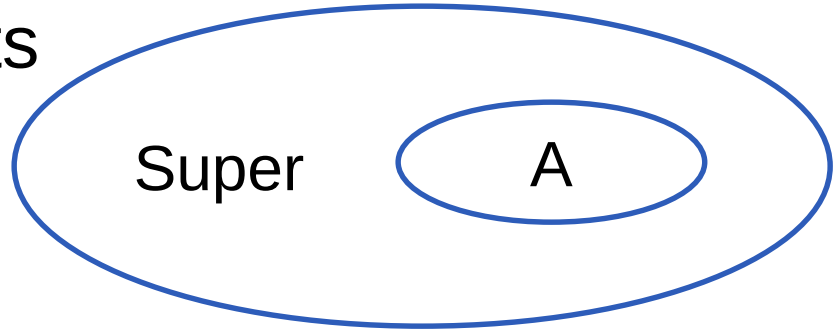




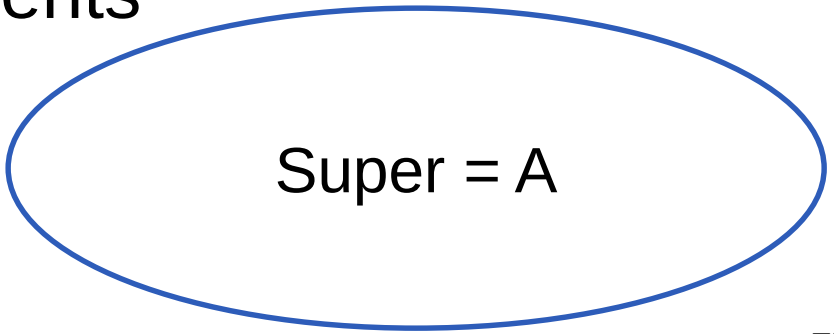
Exclusively Super Elements?

- Does an element exist that is an element of a superclass but not an element of any subclass?

- Exclusively super elements
  - A ⊆ Super
  - e element of Super
  - e not element of A



- No exclusively super elements
  - A ⊆ Super
  - e element of Super
  - e element of A

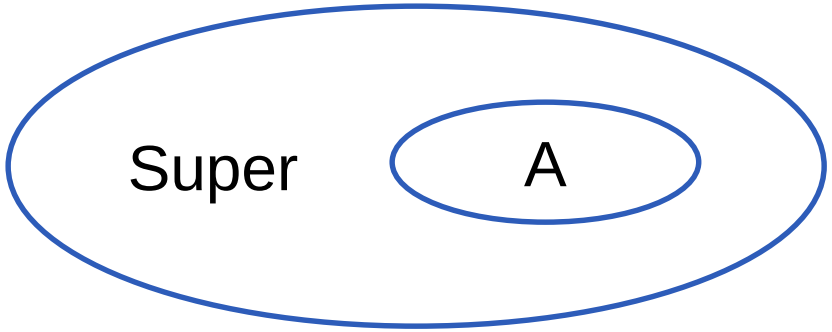


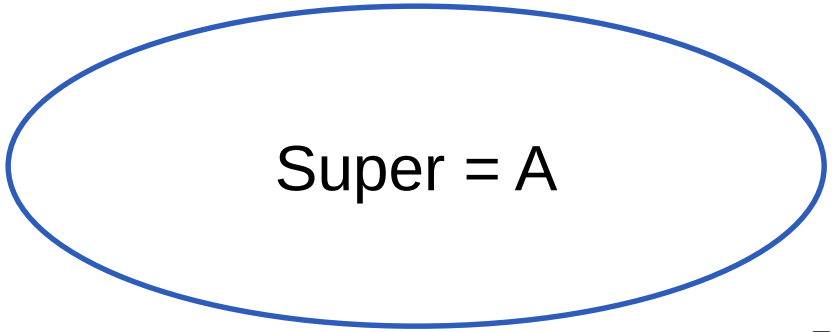
Exclusively Super Elements?

- Does an element exist that is an element of a superclass but not an element of any subclass?

- Both possible in Java

- How?





Exclusively Super Elements

- There may exist elements of the superclass that are not element of any subclass
- Concrete classes in Java
  - class Super {..}
  - class A extends Super {..}
- It is possible that an instance of Super is not an instance of any subtype of Super
  - It is possible that an element of the Super set is not an element of any subset of Super

No Exclusively Super Elements

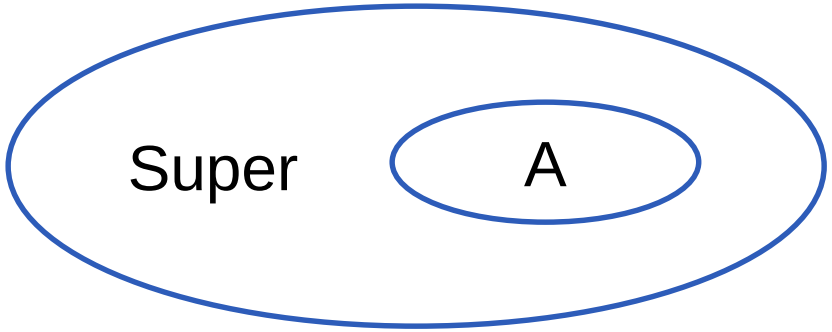
- Each element of a superclass is also an element of a subclass
- Abstract types in Java
  - interface Super {..}
  - abstract class Super {..}
- Each instance of abstract type Super also has to be an instance of a subtype of Super
  - Each element of set Super also has to be an element of a subset of Super

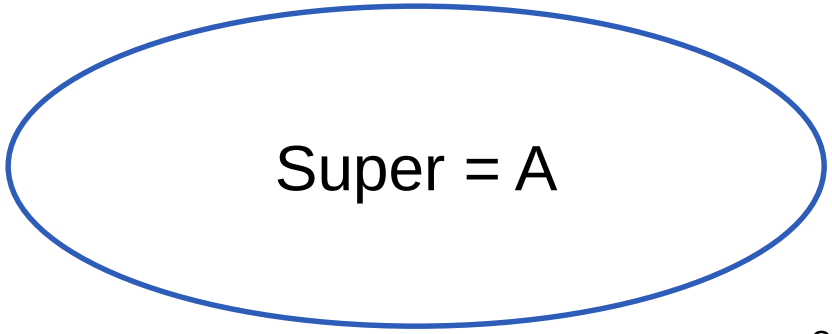
Exclusively Super Elements?

- Does an element exist that is an element of a superclass but not an element of any subclass?

- Both possible in Java

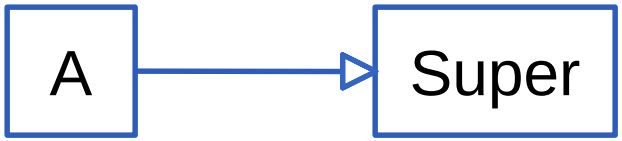
- How about UML?

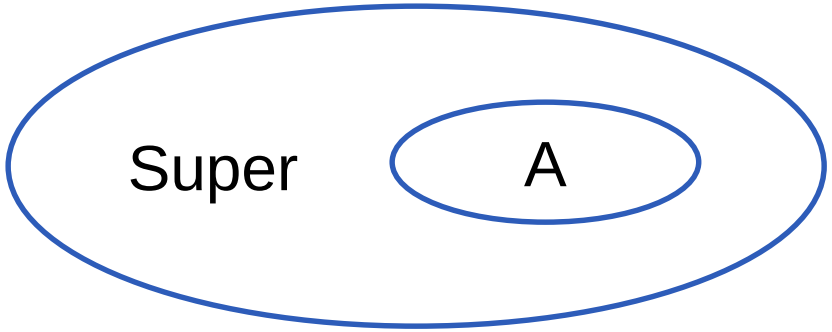




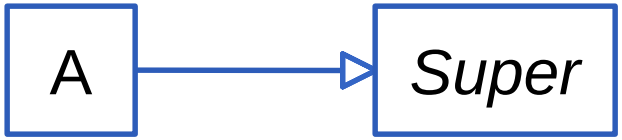
Exclusively Super Elements?

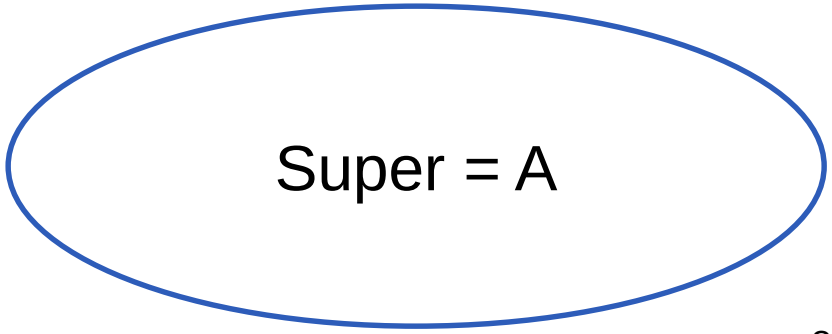
- Does an element exist that is an element of a superclass but not an element of any subclass?
- Both possible in UML
- Default:



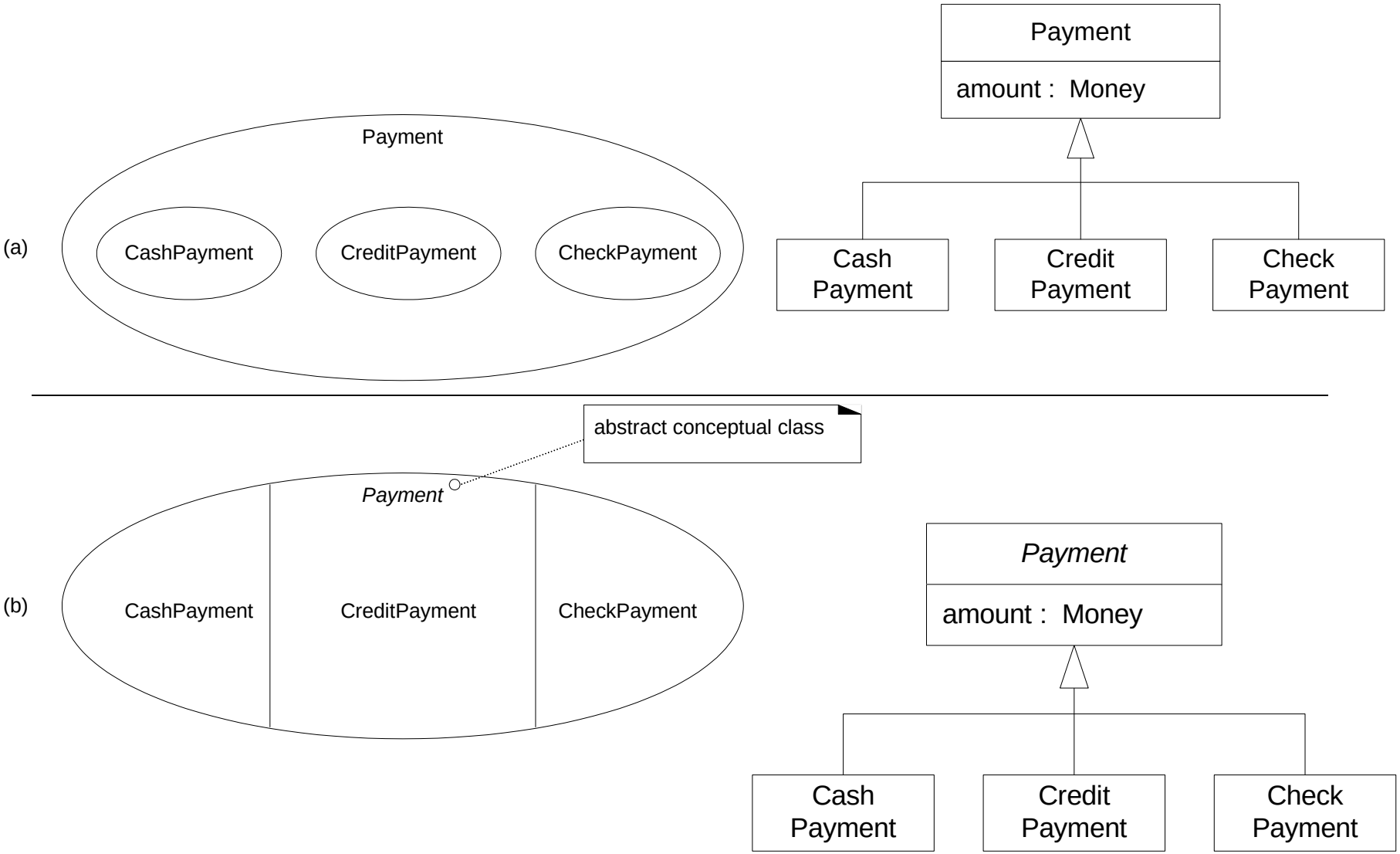


- Abstract conceptual superclass in *italic*





Payment Example



IN-CLASS EXERCISE: REFINE SUBCLASS RELATION

Refine Subclass Relation

- Get together with your team
- Refine your domain model's super-classes by allowing or disallowing elements to be exclusively members of the super-class
- Be prepared to present your results