Analiza și Modelarea Sistemelor Software - Lecture 2: UML Class Diagrams

Traian-Florin Șerbănuță

2025

Agenda

1. Quick recap
2. What are class diagrams?
3. Core elements of class diagrams
4. Associations, multiplicity & composition
5. Advanced concepts (generalization, interfaces, dependencies)
6. **Interactive exercise 1:** Identify model elements
7. Design heuristics and good practices
8. **Interactive exercise 2:** Build a small class model
9. Wrap-up & next steps

Recap from Last Class

- Why modeling is crucial
- How abstraction helps communication
- First exposure to diagrams (morning routine exercise)

Today: move from informal to formal models.

What is a Class Diagram?

- **Purpose:** describes the *static structure* of a system
- **Main elements:** classes, attributes, operations, relationships
- **Used for:**
  - Domain modeling
  - Design-level documentation
  - Communication between stakeholders

Sample class diagram

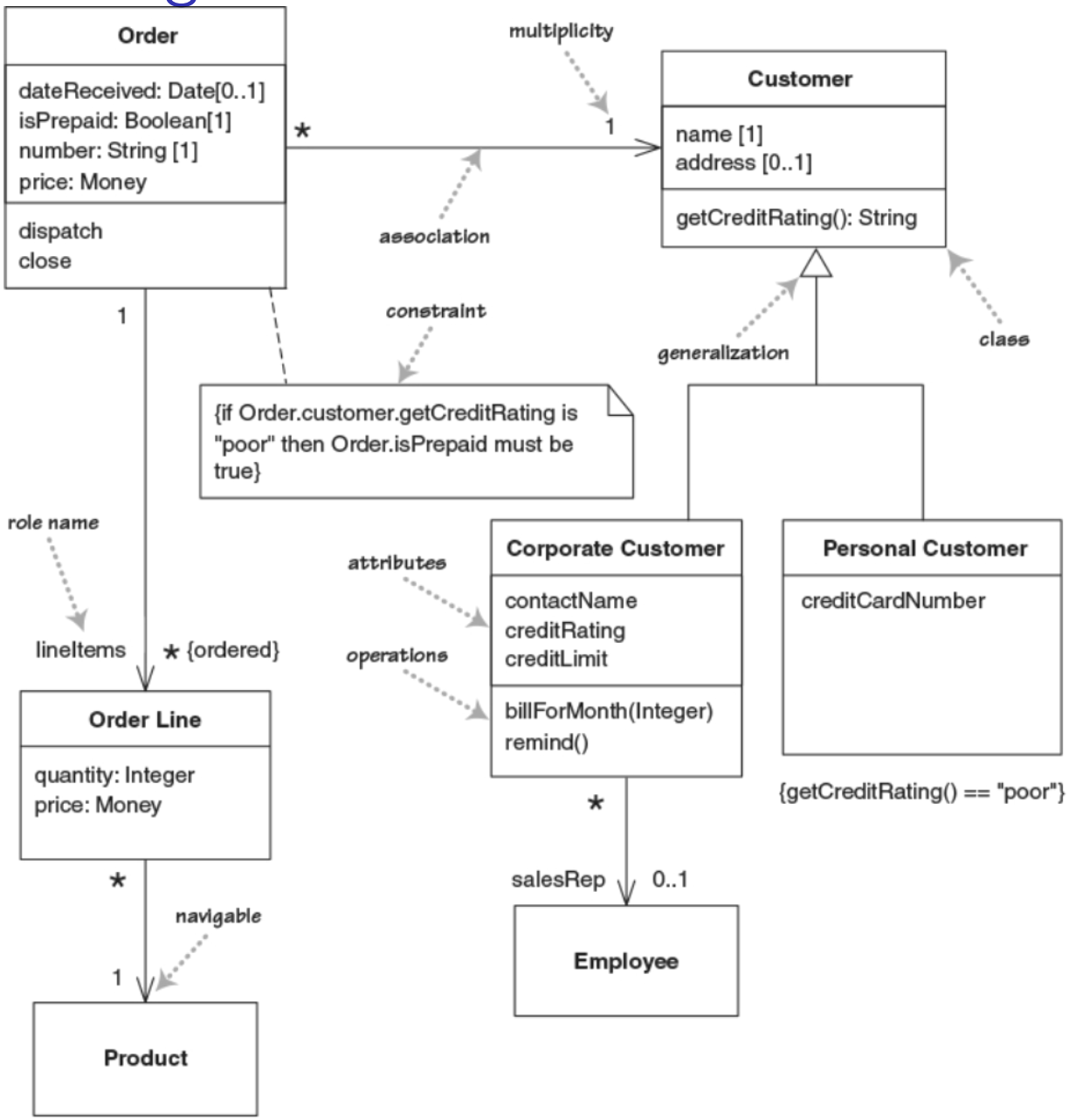


Figure 1: Order processing

Core Elements

Concept	Description	Example
Class	Blueprint for objects	`Book`, `Student`
Attribute	Property of a class	`title: String`, `age: Integer`
Operation	Behavior of a class	`borrowBook()`, `calculateFine()`

Visibility	Access modifier	`+ public`
		`- private`
		`# protected`
		`~ package`

Attribute ::= visibility name: type multiplicity = default {props}

Operation ::= visibility name(parameter-list): return-type {props}

Parameter ::= direction name: type = default_value

Associations

- Represent *relationships* between classes
- Can have:
  - **Roles**: names describing relationship ends
  - **Multiplicity**: number of instances
  - **Navigability**: direction of the relationship

Properties vs Associations:

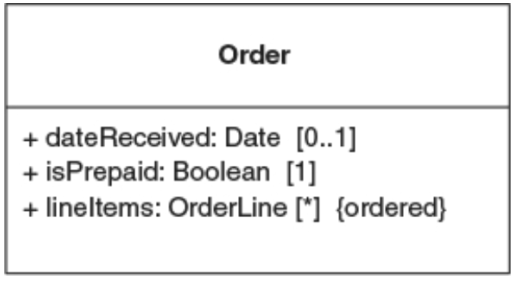


Figure 2: Properties

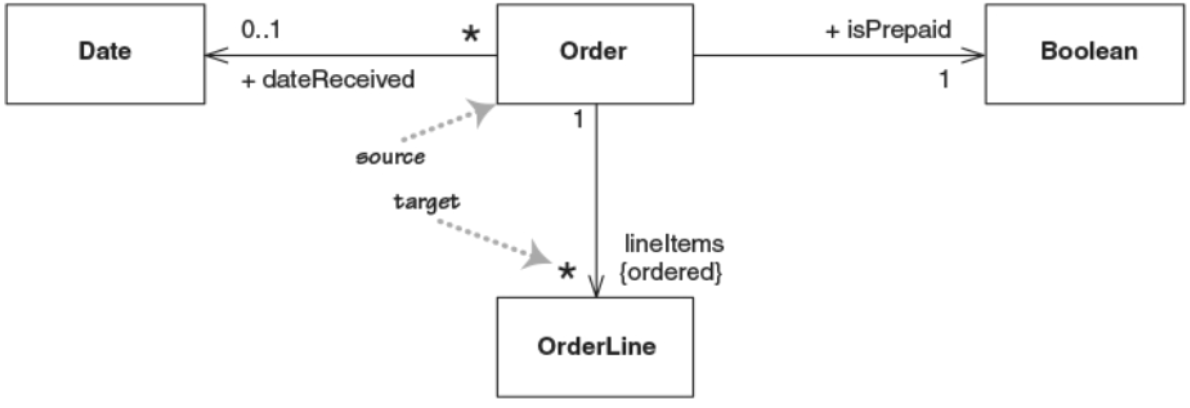


Figure 3: Associations

Bidirectional Associations

- Pair of properties which are linked together as inverses

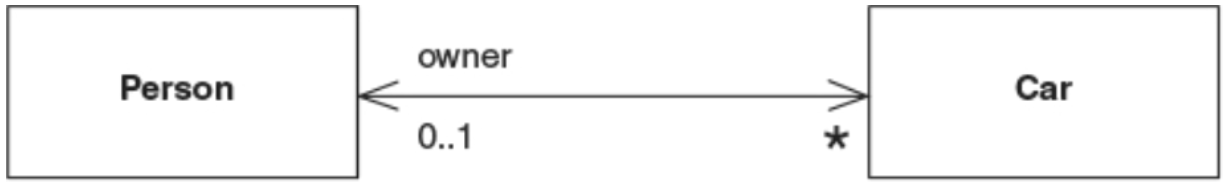


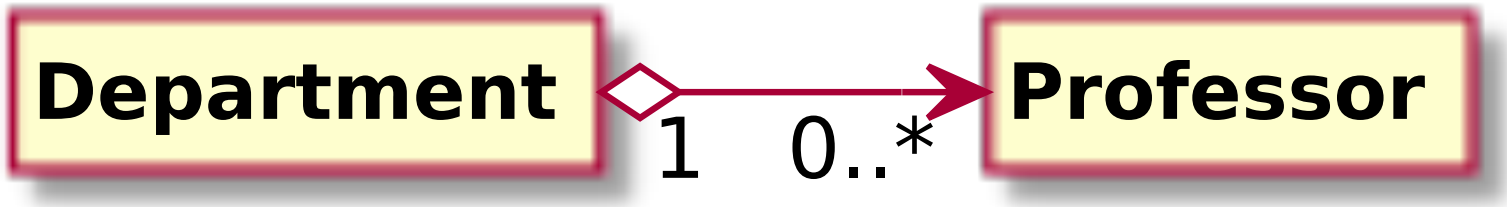
Figure 4: A bidirectional association

If I start with a car, take its `owner`, then take the `cars` property of the owner, then I should find the original car among those cars.

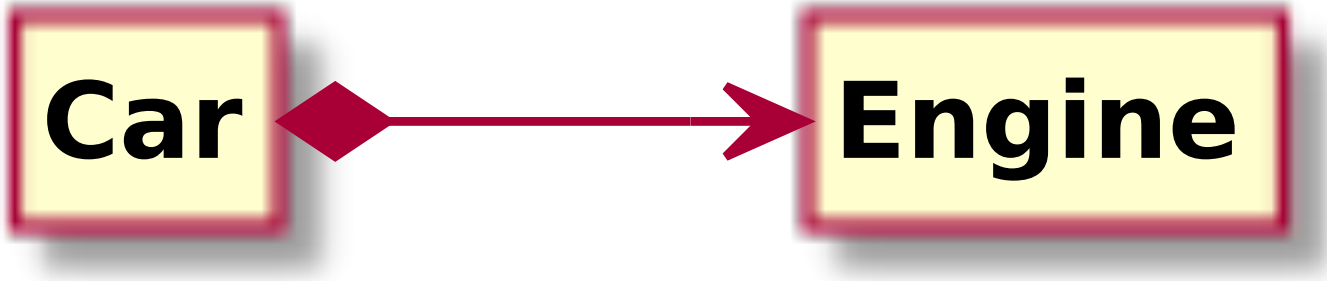
Figure 5: Another way to show a bidirectional association

Aggregation vs Composition

- **Aggregation (◊)**: "has-a" relationship, but parts can exist independently

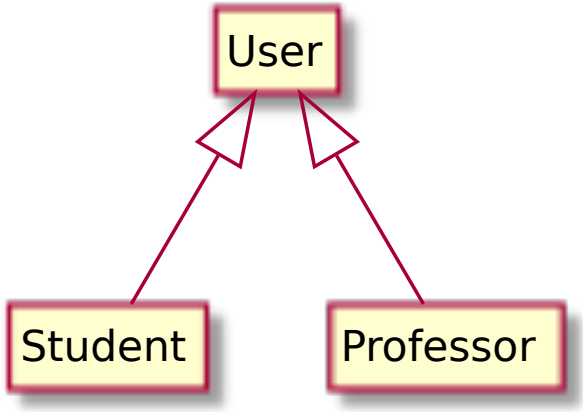


- **Composition (⧫)**: "owns-a" relationship, parts die with the whole

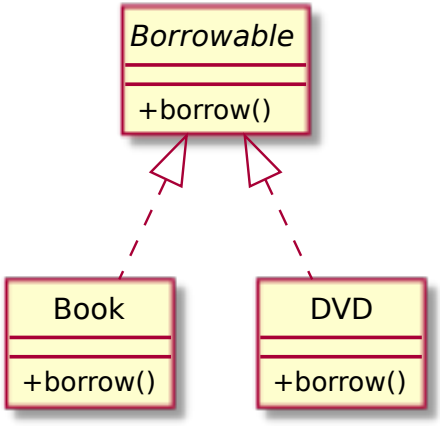


Generalization and Interfaces

- **Generalization:** inheritance between classes



- **Implementation** of an interface



Dependencies

- A dependency exists between two elements if
  - changes to the definition of one element (the supplier or target)
  - may cause changes to the other (the client or source).
- Indicated with a dashed arrow

Example – Interfaces and abstract classes in Java

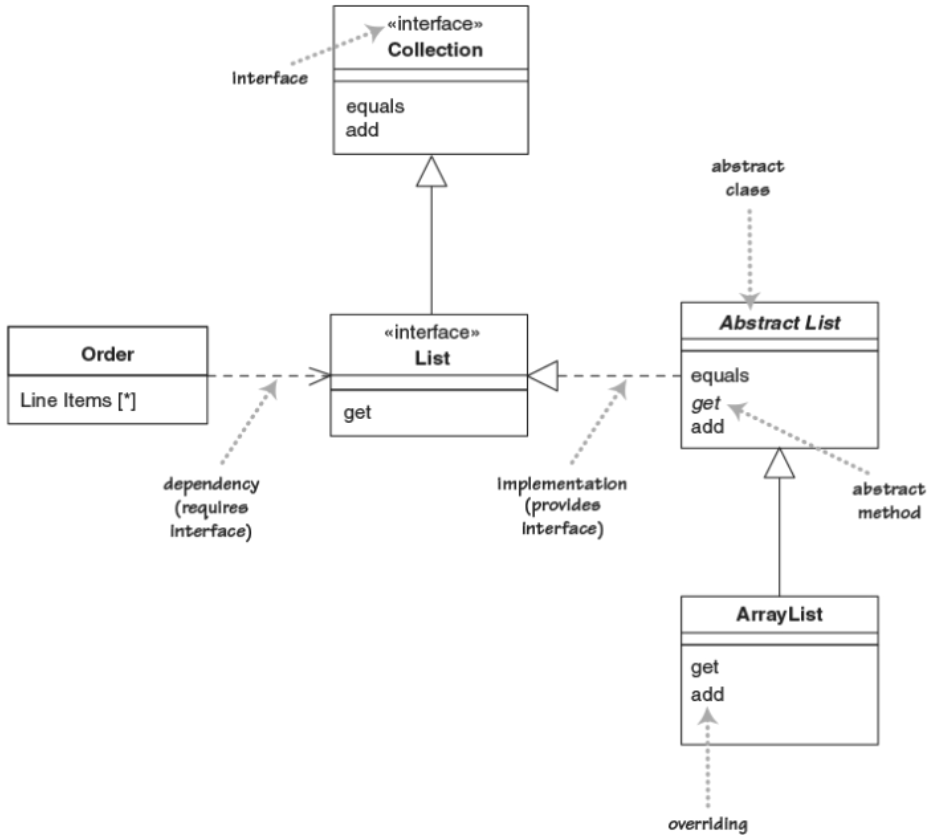


Figure 6: Interfaces and abstract classes in Java – expanded view

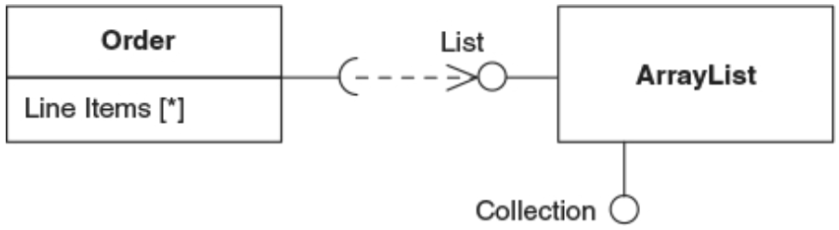


Figure 7: Interfaces and abstract classes in Java – ball-and-socket view

Interactive Exercise 1: Spot the Elements

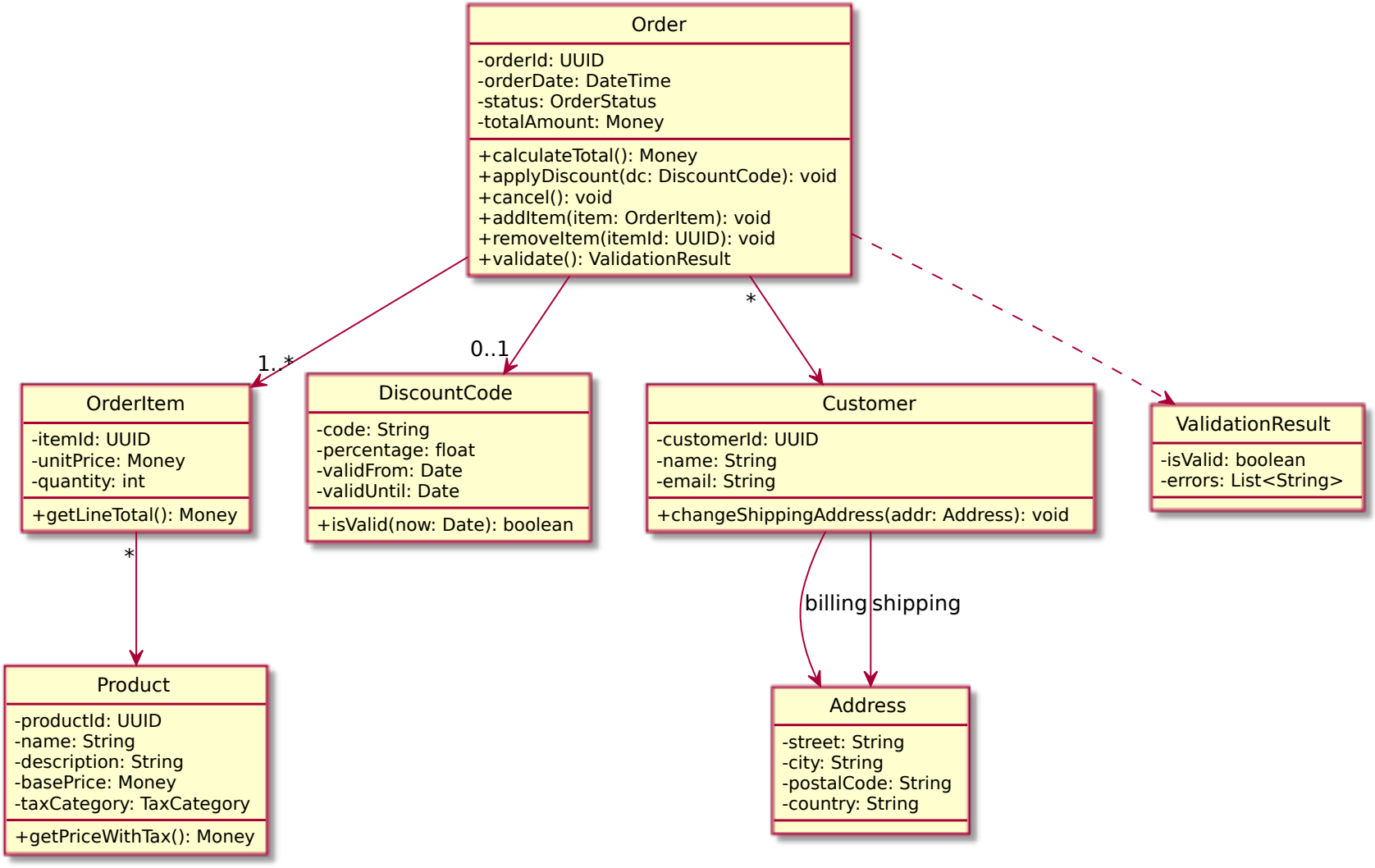
```
Class: Library
 - name: String
 - books: List<Book>
 + addBook(b: Book)
 + findBook(title: String): Book

Class: Book
 - title: String
 - author: String
 + borrow()
 + return()
```

Task (5 minutes): Identify classes, attributes, operations, and their relationships.

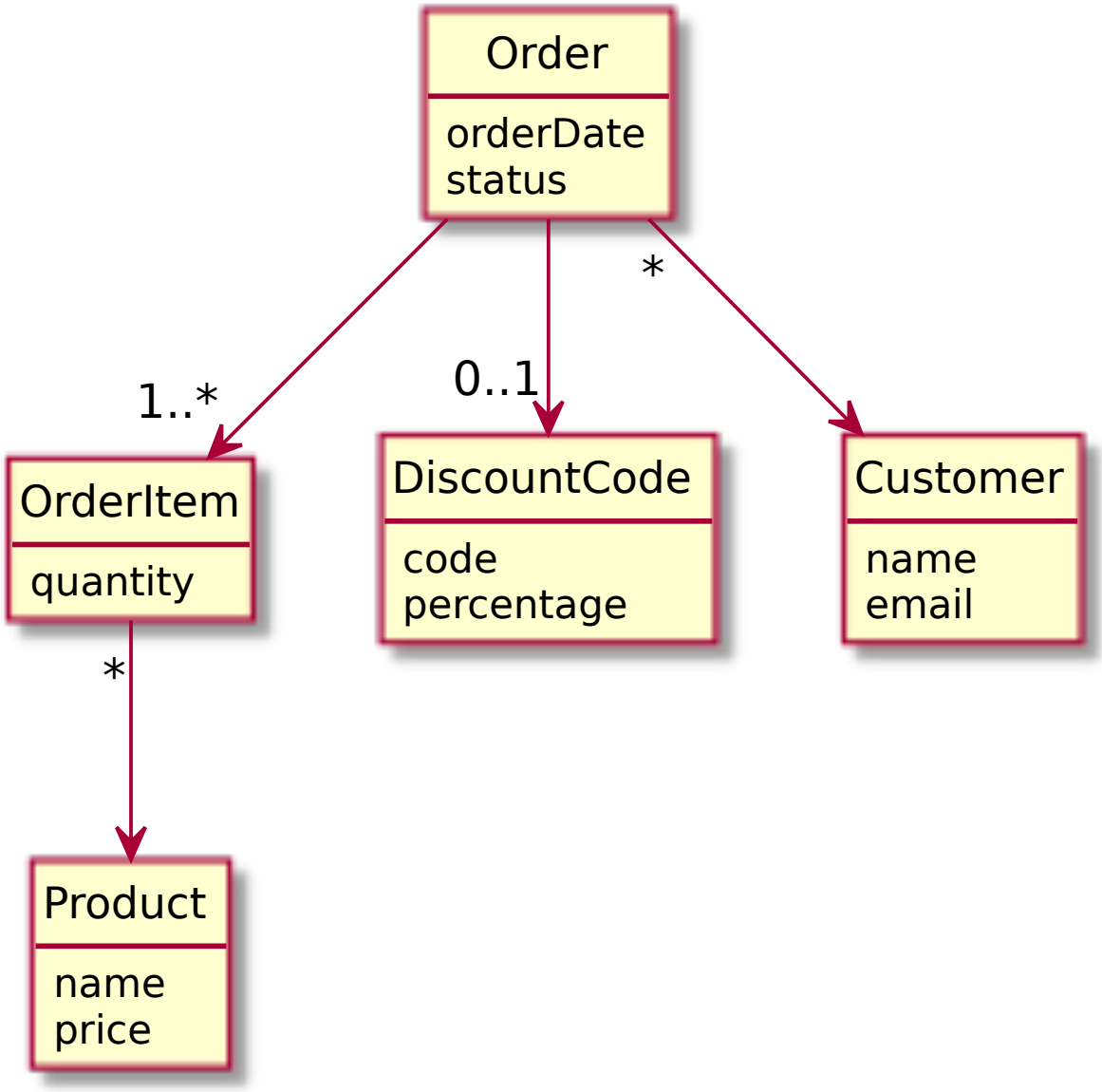
Design Heuristics & Good Practices

- Favor composition over inheritance
- Keep diagrams readable ($<$20 classes per diagram)
- Use consistent naming conventions
- Model only what's necessary
- Document assumptions and constraints

Example – over-complicated diagram



Example – Simplified (conceptual) diagram



Interactive Exercise 2: Build a Class Diagram

Scenario: Online food delivery system.

- Entities: `Customer`, `Restaurant`, `Order`, `MenuItem`, `DeliveryDriver`
- Operations: place order, assign driver, calculate total

Task (15 minutes):

- Form small groups
- Sketch a class diagram
- Show associations, multiplicities, and at least one inheritance relationship

Then: Present and discuss different design choices.

Wrap-Up

Today's takeaways:

- Class diagrams model the structure of systems
- Associations and multiplicities matter
- Composition, inheritance, and dependencies define relationships
- Modeling requires balance: detail vs. clarity

Next class: requirements analysis