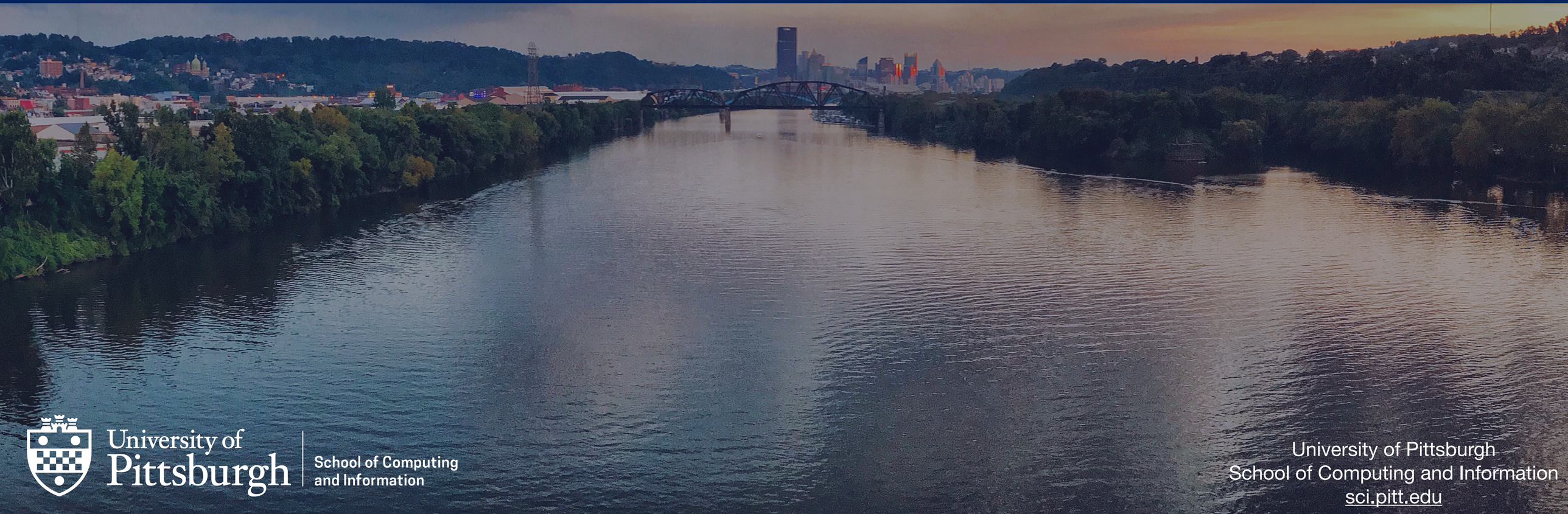
Fall 2025

L18 MVC & Exercise

CS 1530 Software Engineering

Nadine von Frankenberg



University of Pittsburgh
School of Computing and Information
sci.pitt.edu

Copyright

- These slides are intended for use by students in CS 1530 at the University of Pittsburgh only and no one else. They are offered free of charge and must not be sold or shared in any manner. Distribution to individuals other than registered students is strictly prohibited, as is their publication on the internet.
  - All materials presented in this course are protected by copyright and have been duplicated solely for the educational purposes of the university in accordance with the granted license. Selling, modifying, reproducing, or sharing any portion of this material with others is prohibited. If you receive these materials in electronic format, you are permitted to print them solely for personal study and research purposes.
  - Please be aware that failure to adhere to these guidelines could result in legal action for copyright infringement and/or disciplinary measures imposed by the university. Your compliance is greatly appreciated.
- Material from these notes is obtained from various sources, including, but not limited to, the following:
  - Bruegge, & Dutoit. Object–oriented software engineering. using UML, patterns, and Java. Pearson, 2009.
  - Gamma, Erich, Richard Helm, Ralph Johnson, and John Vlissides. Design Patterns. Pearson, 1994.
  - Sommerville, Ian. "Software Engineering" Pearson. 2011.
  - http://scrum.org/

Learning goals

- You can model a UML component diagram

Today's roadmap

- MVC
- Model a subsystem decomposition

MVC architectural style — Motivation

- **Problem:**
  - Systems with high coupling tend to require modifications to both boundary objects (presentation/user interface) and entity objects (data) when changes are made to either component
  - Developers cannot work in parallel (UI changes can be made affect the underlying data or logic)
- **Solution:** decouple entity objects (data) from boundary objects (presentation)

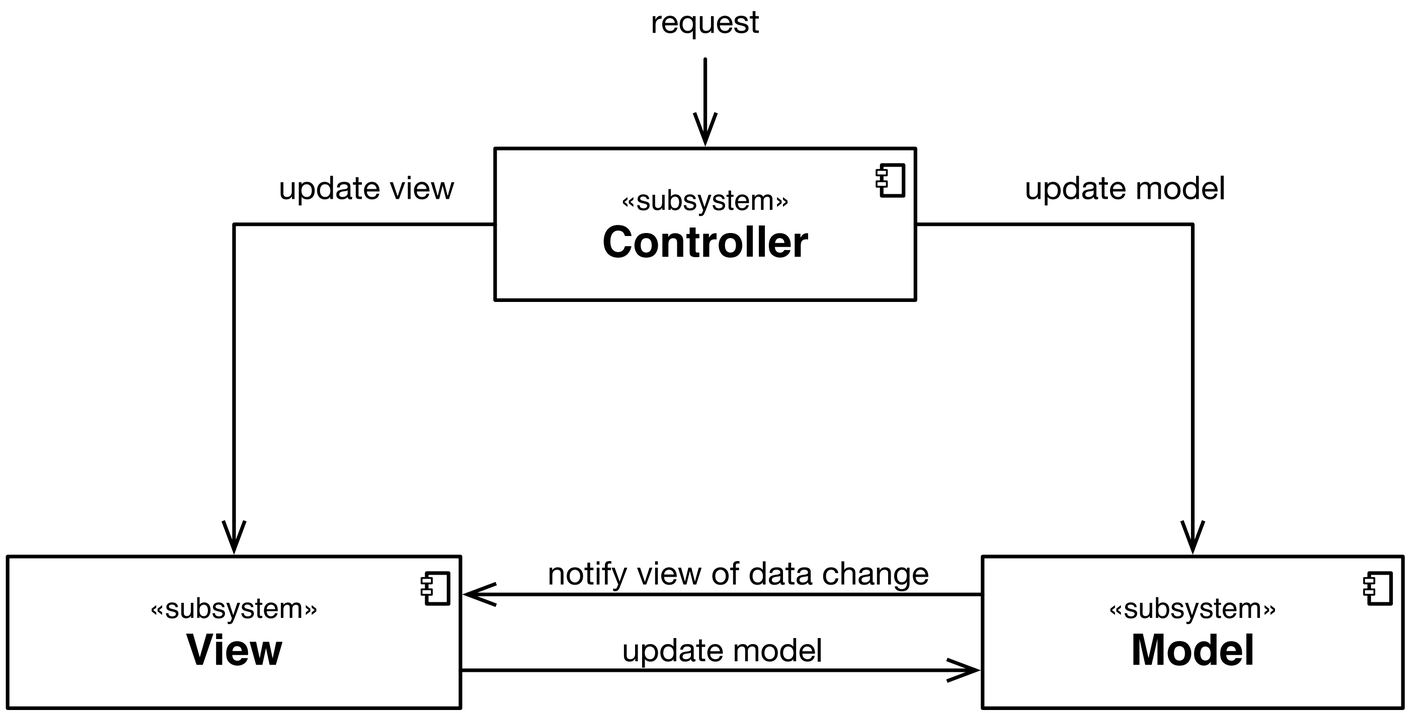
MVC architectural style — Definition

University of Pittsburgh

- Model-view-controller (MVC) architectural style
- MVC separates a system into three main components:
  - Model — processes and stores entity objects
  - View — displays data (boundary objects) to the user
  - Controller — Acts as an intermediary between Model and View
- MVC promotes a clear separation of concerns and modularity in application design
- Commonly used when developing user interfaces
  - Widely used in web development, desktop applications, mobile app development, …
- Enhances code maintainability and reusability

MVC architectural style — Overview

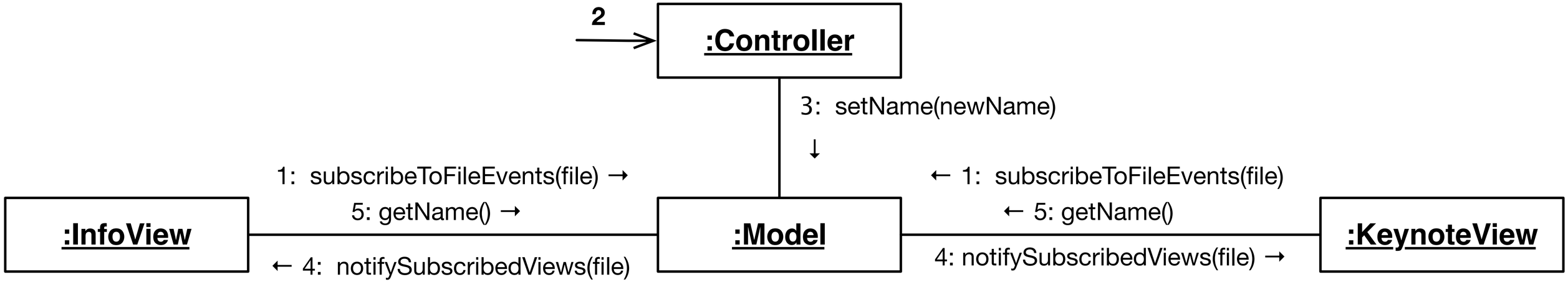
- MVC typically operates on a cycle of user interactions
- User input triggers actions within the Controller
- The Controller updates the Model based on these actions
- Changes in the Model are reflected in the View



[Example] MVC

- A website:
  - Model → HTML (handles knowledge of the website)
  - View → CSS (presentation)
  - Controller → browser (manipulates data through, e.g., user input forms)
- Mac:
  - Locate a file (keynote presentation) in the file browser (Finder)
  - Open the file's info view
  - Open the file in keynote
  - Change the file's name (the new name is synced across all views)

[Example] Update File name



Informal UML communication diagram: attributes & objects are missing

I18 - Order system (MVC)

⏳ 10min

👩🏽‍💻👨🏻‍💻 Pairs

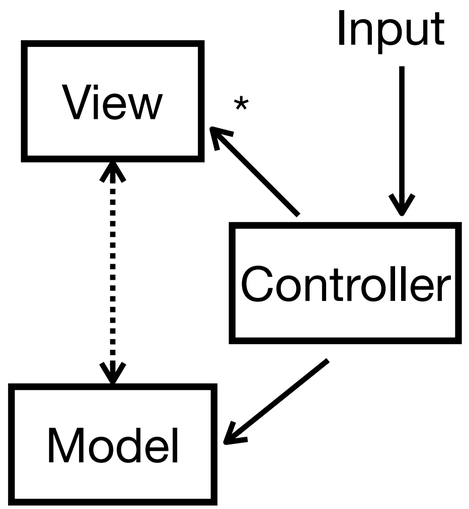
University of Pittsburgh

- Review: https://github.com/pitt-1530/order-app-mvc-refactor
- Describe how you would refactor the system to use the Model-View-Controller architecture

Deisgn goals

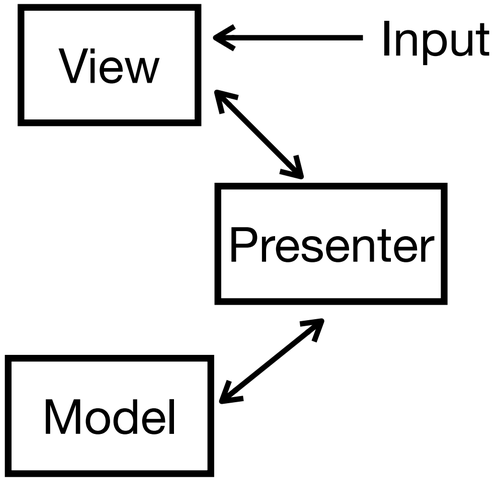
- Maintainability: Each component isolated
- Modifiability: UI changes don’t break logic
- Reusability: Same model works for different interfaces
- Parallel development: Teams work on different subsystems

Variations of MVC



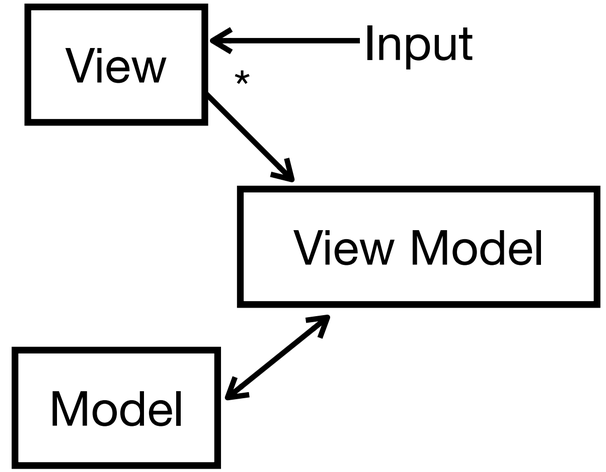
MVC
(Model-View-Controller)

Presenter replaces controller, View is passive



MVP
(Model-View-Presenter)

ViewModel exposes data, View updates automatically via data binding



MVVM
(Model-View-Viewmodel)

When to use MVC

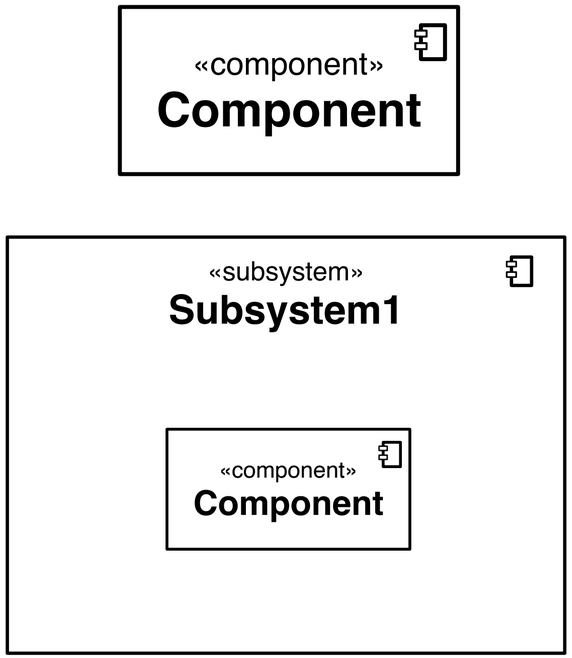
- Multiple synchronized views of the same model
- Exchangeable views and controllers
- Exchangeability of look and feel
- Considerations
  - Increased complexity
  - Potential for excessive number of updates
  - Close coupling of views and controllers to model

Today's roadmap

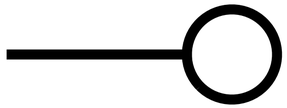
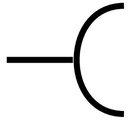
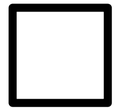
- MVC
- Model a subsystem decomposition

[Recap] UML component diagram

- High-level view of the system's design using components and dependencies
- A **UML component** represents a system part (such as classes, packages, or subsystems)
- UML component diagrams are used for:
  - **Architectural design and planning**
  - Identifying system components and their interactions
  - Representing the high-level structure of a software system
- Represented as a rectangle with a tabbed rectangle symbol inside



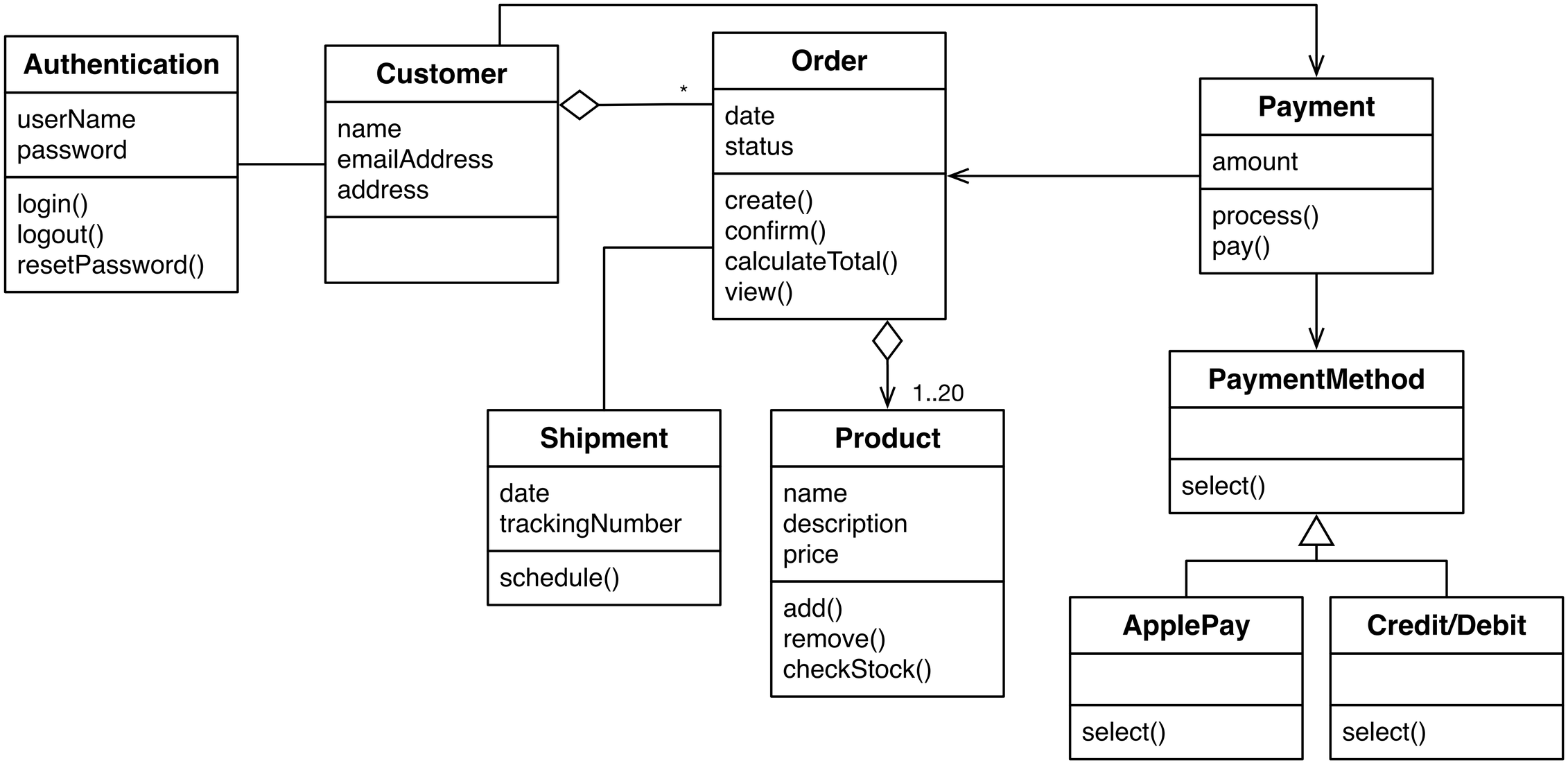
[Recap] UML interfaces

- Define how components interact with each other or with external entities
- Specify the required and provided services:
  - Lollipop: provided interface 
  - Socket: required interface 
  - Dependency 
    (a component depends on the implementation of another component)
  - A **port** specifies a distinct interaction point between the component and its environment 
    - Ports are depicted as small squares on the sides of classifiers
    - Ports allow to group interfaces

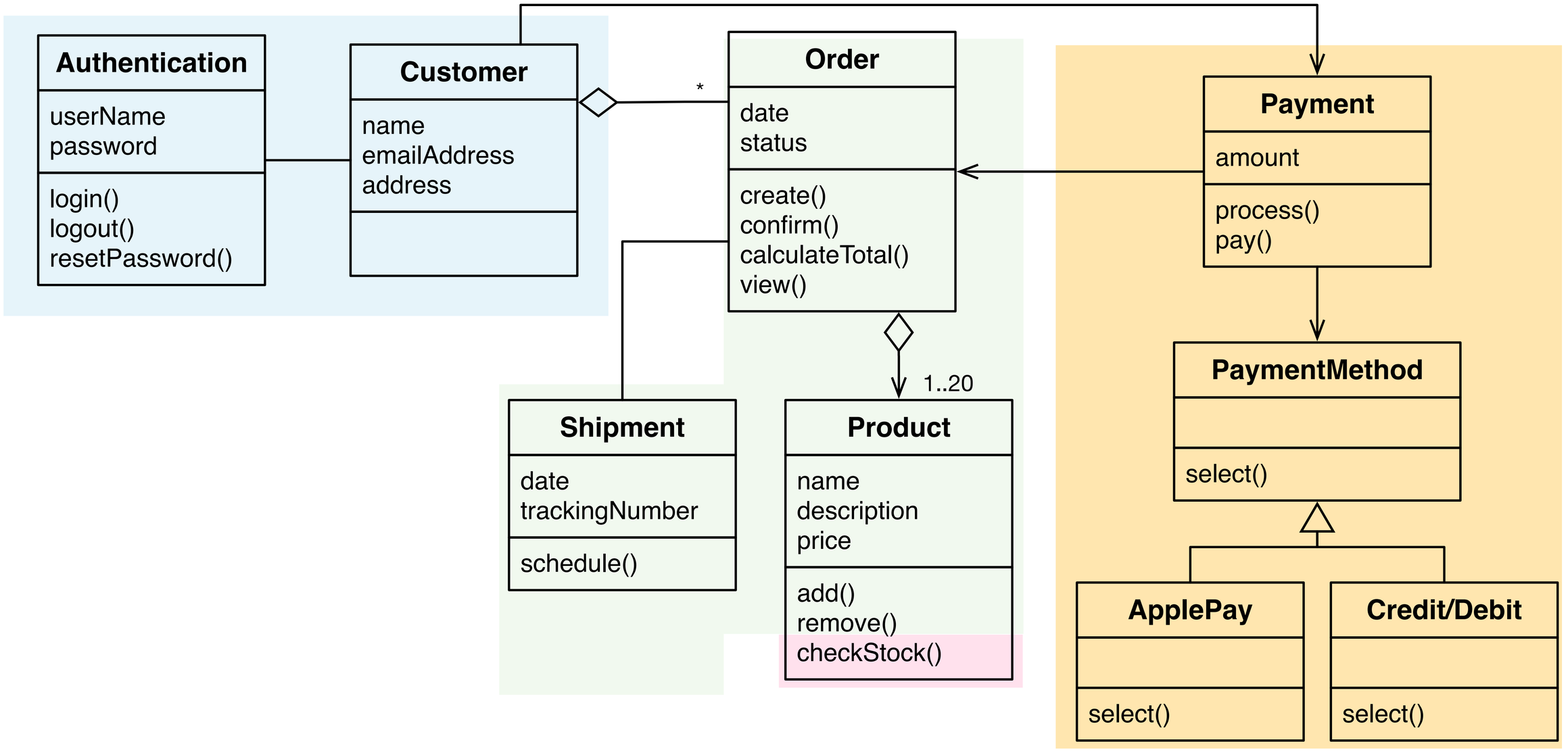
[Recap] Subsystem decomposition — Steps

1. **Identify functional components:** Break the system into functional components with distinct roles
2. **Define interfaces:** Clearly specify how subsystems interact and exchange data
3. **Organize subsystems:** E.g., organize subsystems hierarchically
4. **Assign responsibility**: Assign responsibilities to each subsystem
5. **Manage dependencies**: Minimize dependencies between subsystems

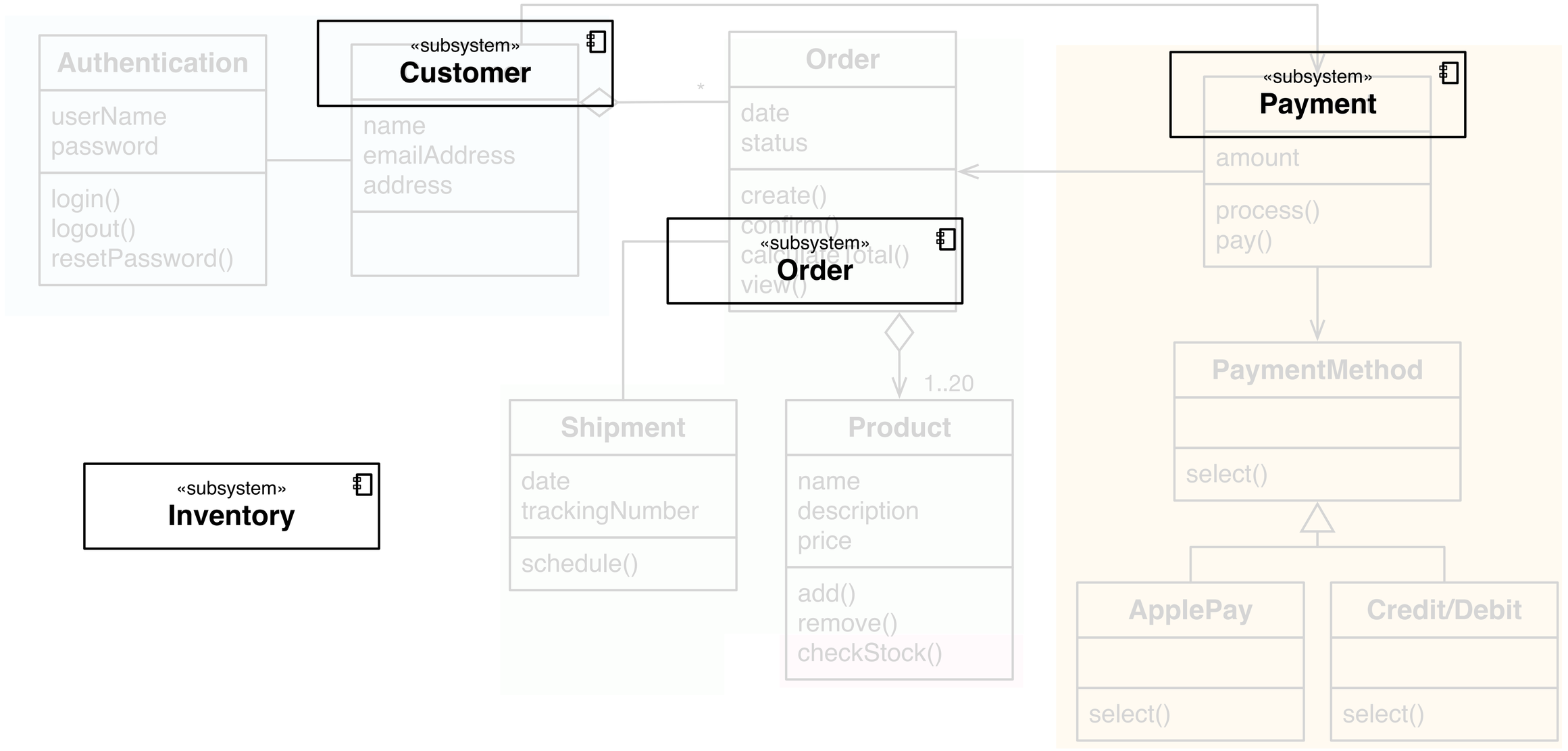
[Example] E-Commerce Ordering System



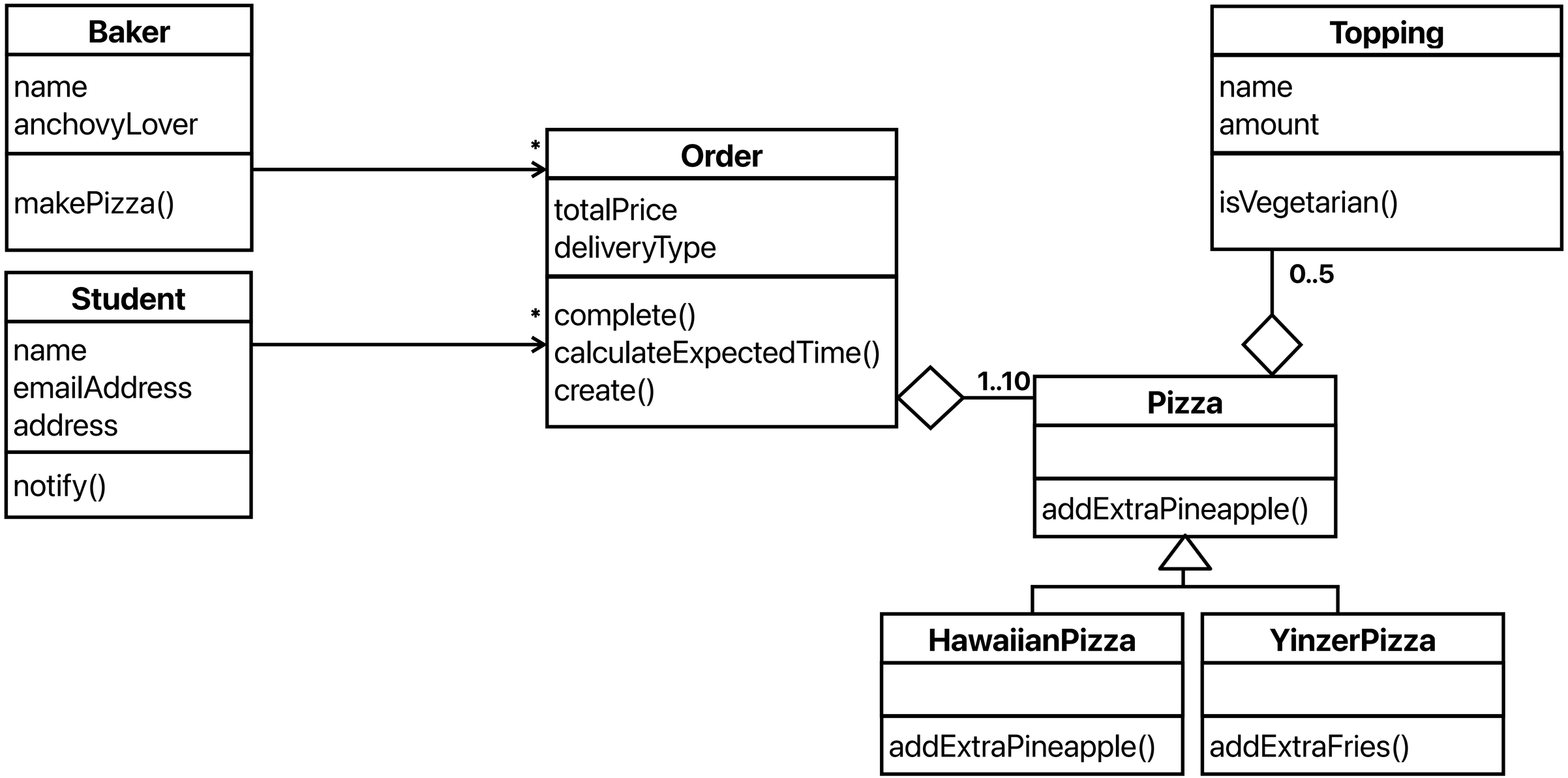
[Example] E-Commerce Ordering System - Option I



[Example] E-Commerce Ordering System - Option I



[Example] Pizza Ordering System



I18 In-class: Subsystem Decomposition

⏳ 15 min

👩🏽‍💻👨🏻‍💻 Pairs

University of Pittsburgh

Decompose your own or the provided pizza ordering system analysis object model into subsystems:

- Create a **UML component diagram**
- Decide on interfaces
- Remember the SOLID principles!
- **Review architectural styles: Can they solve a "problem"?**
- Add a discussion (3-6 sentences / short bullet points) how your architecture follows SOLID and which architectural style(s) you have chosen and why.
  - Describe at least 3 SOLID principles

References

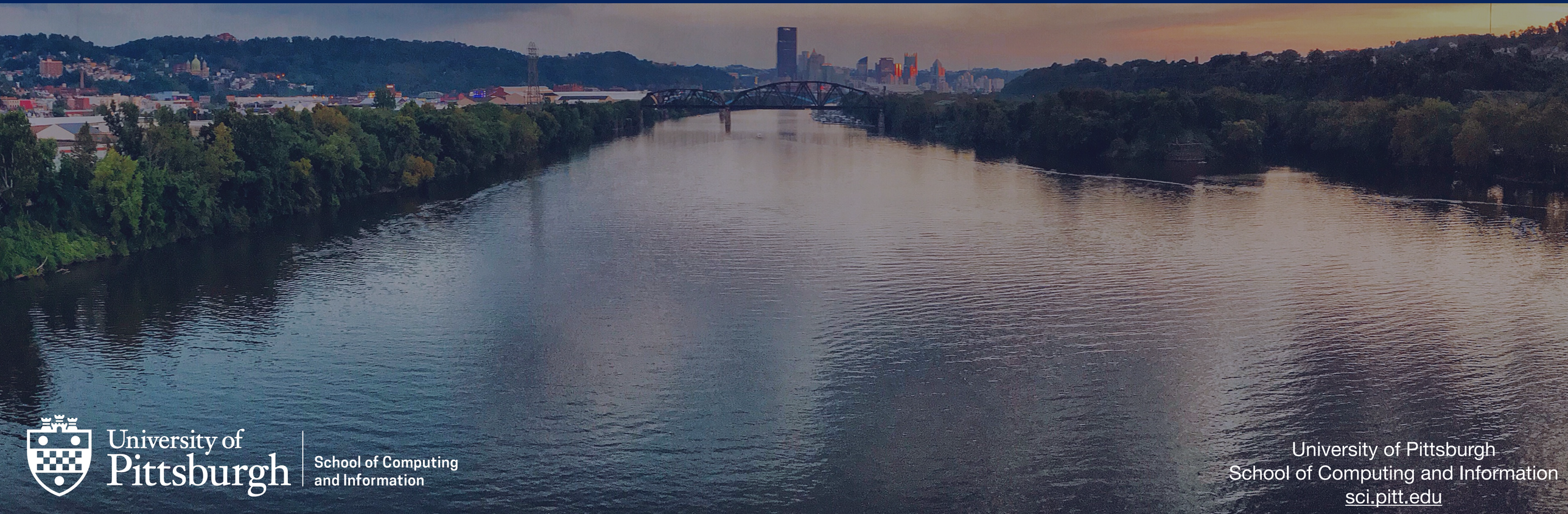
- Bruegge, & Dutoit. Object–oriented software engineering. using UML, patterns, and Java. Pearson, 2009.
- Object Management Group. Unified Modeling Language. Version 2.5.1, 2017

Fall 2025

L18 MVC & Exercise

CS 1530 Software Engineering

Nadine von Frankenberg



University of Pittsburgh
School of Computing and Information
sci.pitt.edu