

Principles of Software Construction: Objects, Design, and Concurrency

Part 1: Design for reuse

A formal design process: Responsibility assignment

Josh Bloch

Charlie Garrod



Administrivia

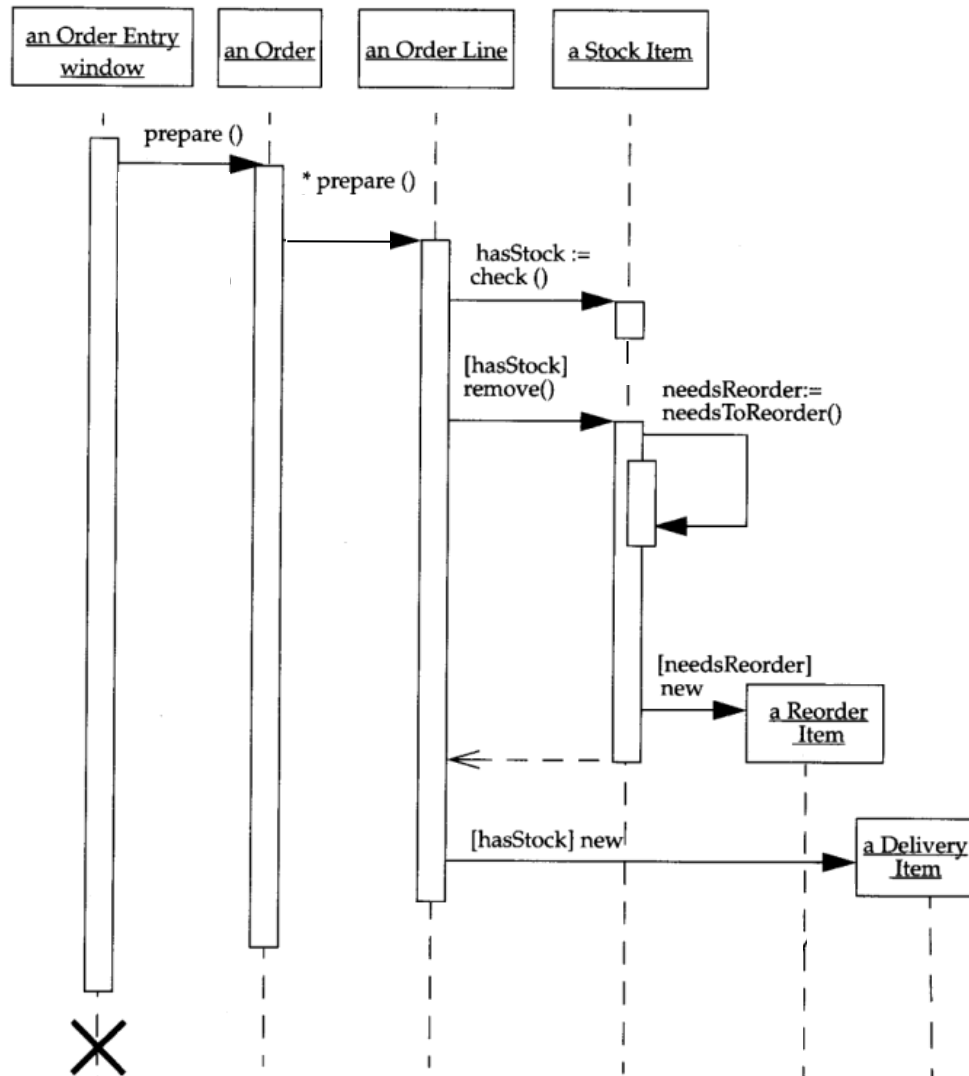
- Midterm exam ongoing
 - Due tonight at 11:59 p.m.
- Homework 4
 - Three parts, part A due next Thursday
 - Design review meetings next week



https://commons.wikimedia.org/wiki/File:1_carcassonne_aerial_2016.jpg


Key concepts from Tuesday

Sequence diagrams to visualize dynamic behavior




An object-oriented design process

- Model / diagram the problem, define objects
 - Domain model (a.k.a. conceptual model)
- Define system behaviors
 - System sequence diagram
 - System behavioral contracts
- Assign object responsibilities, define interactions
 - Object interaction diagrams
- Model / diagram a potential solution
 - Object model



Tuesday:
Understanding
the problem



Today:
Defining a
solution

Object-oriented programming

- Programming based on structures that contain both data and methods



```
public class Bicycle {  
    private int speed;  
    private final Wheel frontWheel, rearWheel;  
    private final Seat seat;  
    ...  
  
    public Bicycle(...) { ... }  
  
    public void accelerate() {  
        speed++;  
    }  
  
    public int speed() { return speed; }  
}
```

Responsibility in object-oriented programming

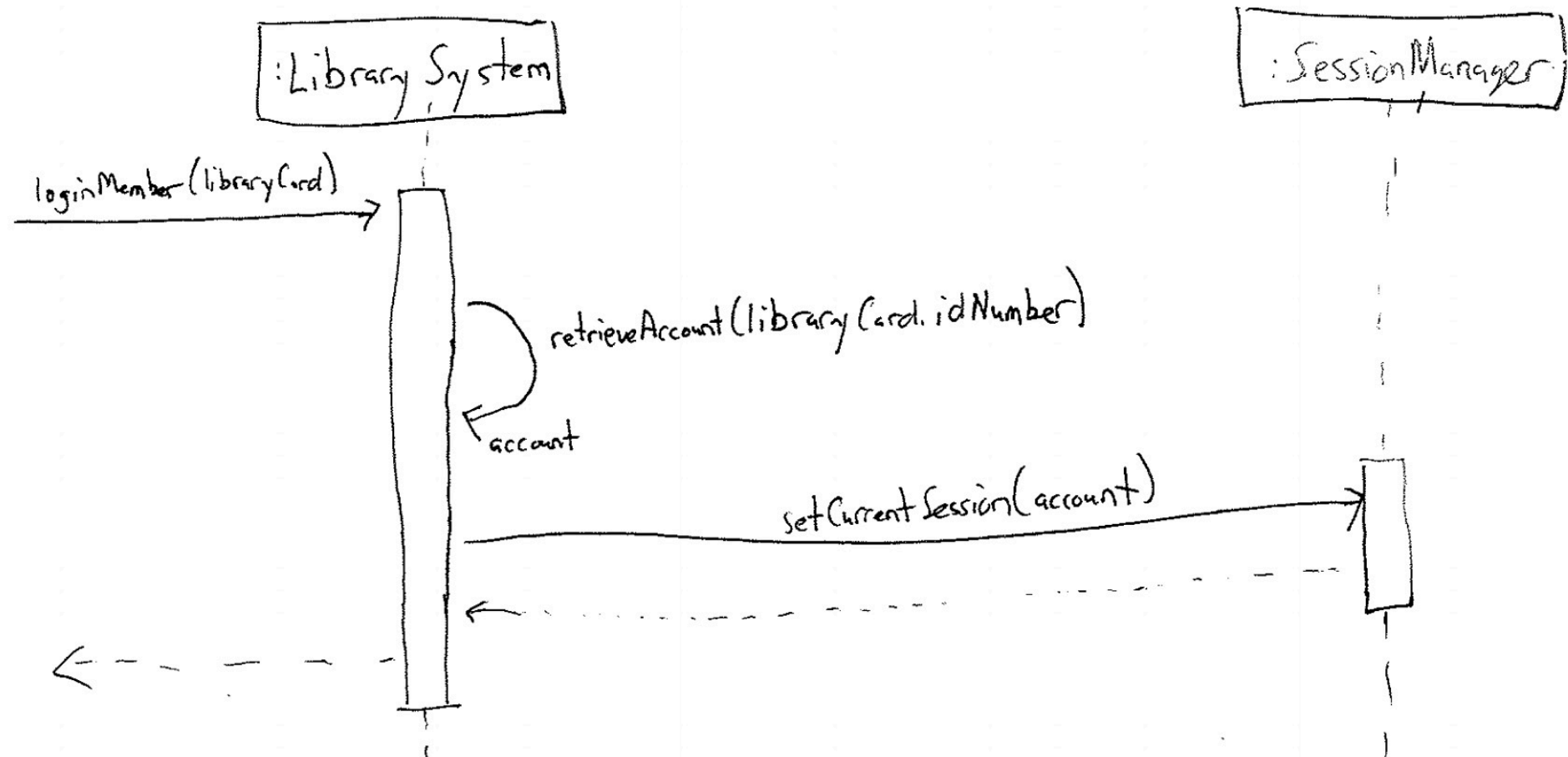
- Data:
 - Private or otherwise encapsulated data
 - Data in closely related objects
- Methods:
 - Private or otherwise encapsulated operations
 - Object creation, of itself or other objects
 - Initiating actions in other objects
 - Coordinating activities among objects

Using interaction diagrams to assign object responsibility

- For a given system-level operation, create an object interaction diagram at the *implementation-level* of abstraction
 - Implementation-level concepts:
 - Implementation-like method names
 - Programming types
 - Helper methods or classes
 - Artifacts of design patterns

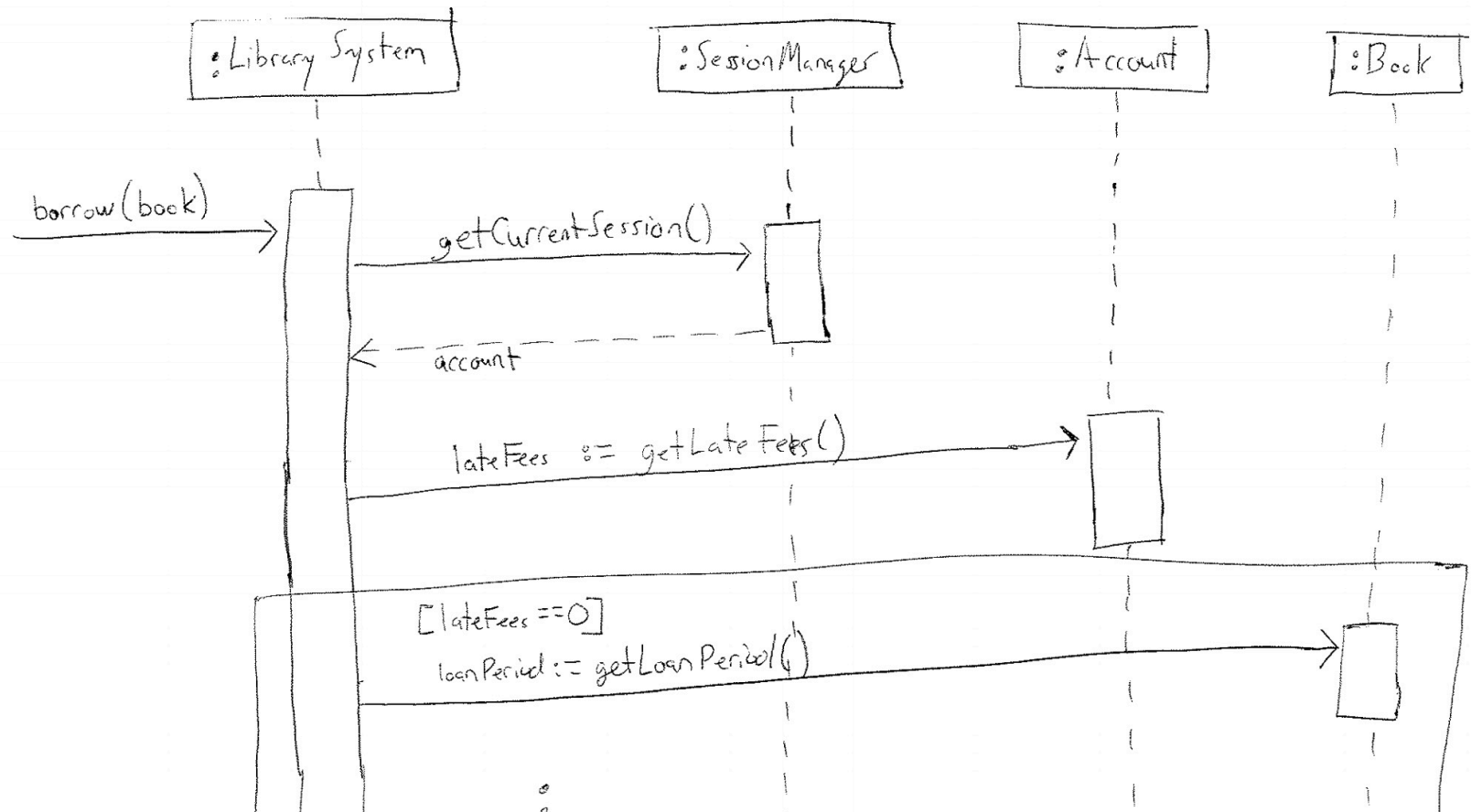
Example interaction diagram #1

Use case scenario: A library member should be able to use her library card to log in at a library system kiosk and ...



Example interaction diagram #2

Use case scenario: ...and borrow a book. After confirming that the member has no unpaid late fees, the library system should determine the book's due date by adding its loan period to the current day, and record the book and its due date as a borrowed item in the member's library account.



Interaction diagrams help evaluate design alternatives

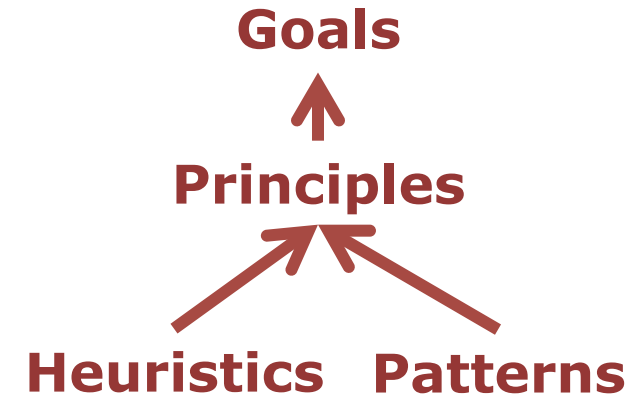
- Explicitly consider design alternatives
- For each, sketch the interactions implied by the design choice
 - Interactions correspond to the components' APIs

Interaction diagrams help evaluate design alternatives

- Explicitly consider design alternatives
- For each, sketch the interactions implied by the design choice
 - Interactions correspond to the components' APIs
- e.g., Create two diagrams that show the required interactions when solving a cryptarithm:
 1. First, assuming that a main method (or another external method or class) has the responsibility to solve the cryptarithm
 2. Instead, assuming that an instance of the cryptarithm class has the responsibility to solve itself

Heuristics for responsibility assignment

- Controller heuristic
- Information expert heuristic
- Creator heuristic



The controller heuristic

- Assign responsibility for all system-level behaviors to a single system-level object that coordinates and delegates work to other objects
 - Also consider specific sub-controllers for complex use-case scenarios
- Design process: Extract interface from system sequence diagrams
 - Key principles: Low representational gap and high cohesion

Information expert heuristic

- Assign responsibility to the class that has the information needed to fulfill the responsibility
 - Initialization, transformation, and views of private data
 - Creation of closely related or derived objects
- Design process: Assignment from domain model
 - Key principles: Low representational gap and low coupling

Responsibility in object-oriented programming

- Data:
 - Private or otherwise encapsulated data
 - Data in closely related objects
- Methods:
 - Private or otherwise encapsulated operations
 - Object creation, of itself or other objects
 - Initiating actions in other objects
 - Coordinating activities among objects

Creator heuristic: Who creates an object Foo?

- Assign responsibility of creating an object Foo to a class that:
 - Has the data necessary for initializing instances of Foo
 - Contains, aggregates, or records instances of Foo
 - Closely uses or manipulates instances of Foo
- Design process: Extract from domain model, interaction diagrams
 - Key principles: Low coupling and low representational gap

Challenges when using the creator heuristic

- In Homework 2, what object should have the responsibility for creating each instruction when parsing an assembly file?

There exist many heuristics

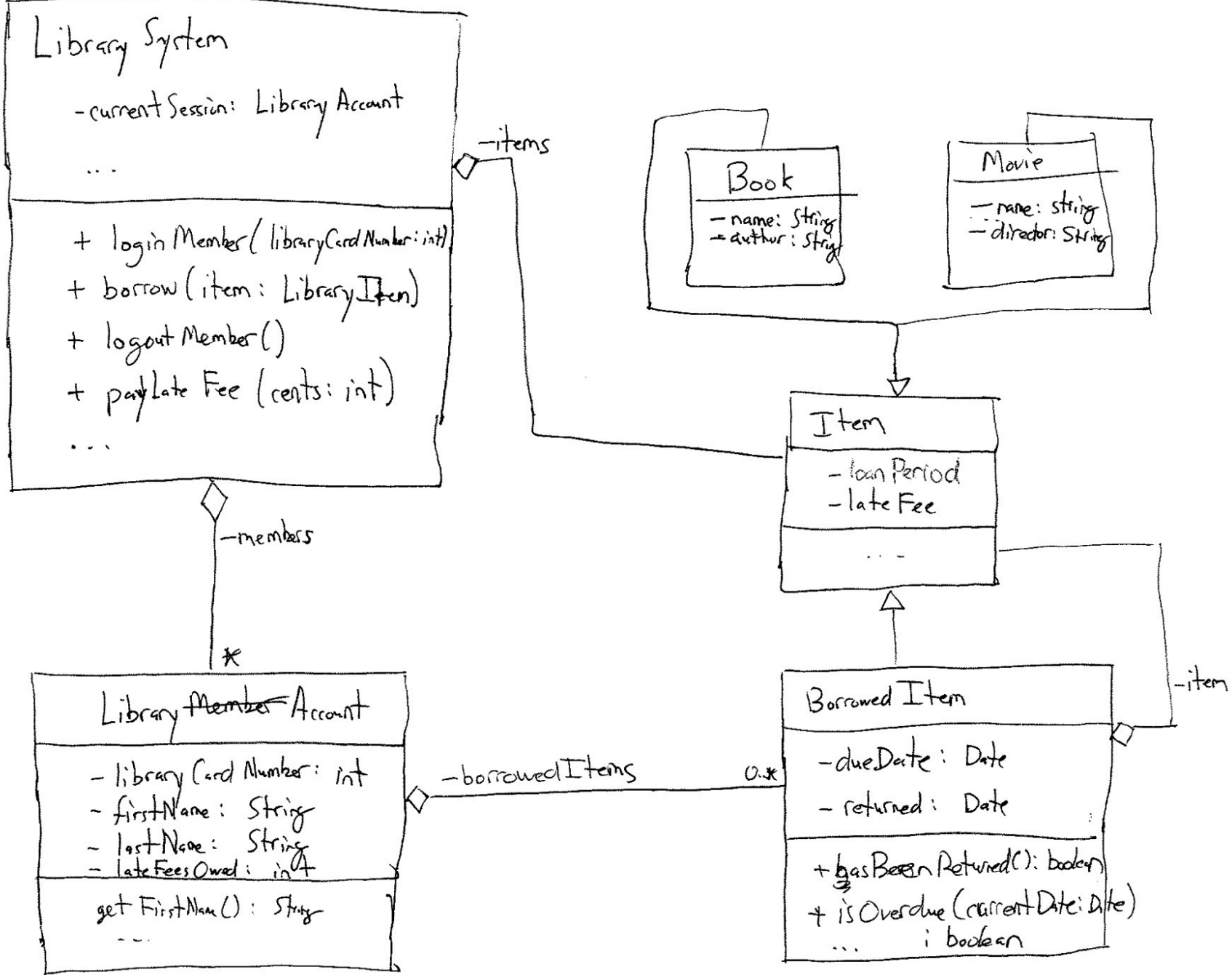
- Minimize mutability
- Minimize conceptual weight
- Favor composition/delegation over inheritance
- Use indirection to reduce coupling
- ...

Object-level artifacts of this design process

- **Object interaction diagrams** add methods to objects
 - Can infer additional data responsibilities
 - Can infer additional data types and architectural patterns
- **Object model** aggregates important design decisions
 - Is an implementation guide

Creating an object model

- Extract data, method names, and types from interaction diagrams
 - Include implementation details such as visibilities



Create an object model for Homework 3?

- Not today!

Summary:

- Object-level interaction diagrams and object model systematically guide the design process
 - Convert domain model, system sequence diagram, and contracts to object-level responsibilities
- Use heuristics to guide, but not define, design decisions
- Iterate, iterate, iterate...