

# Principles of Software Construction: Objects, Design, and Concurrency

Object Oriented Design- responsibility assignment

**Michael Hilton**

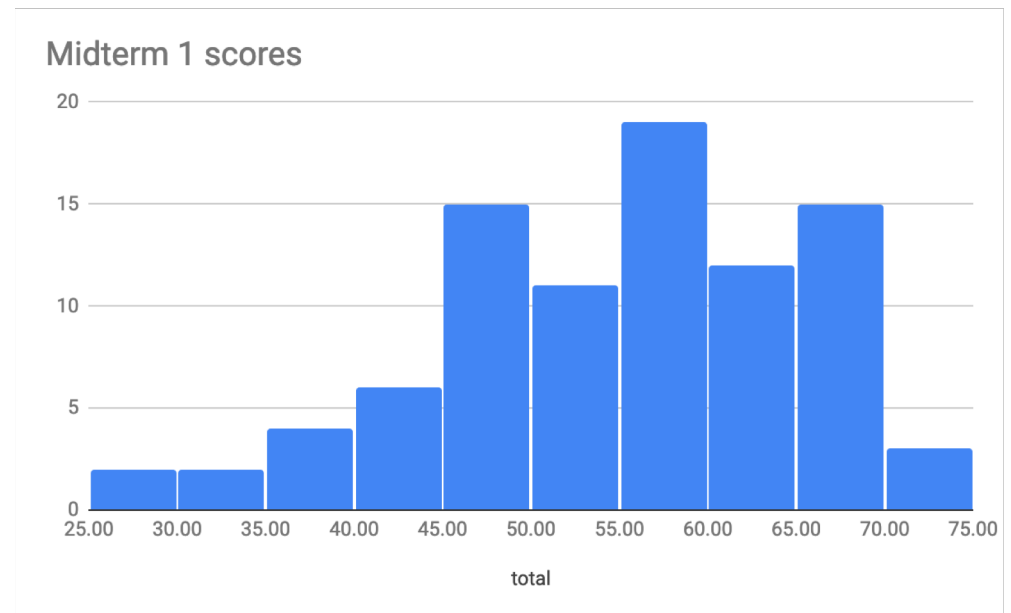
**Bogdan Vasilescu**

# Administrivia

- HW 4a design review meetings ongoing
- Midterms returned today

# Midterm

- Mean 54/74
- SD 10
- Max 71
- NOTE: This course does not have a fixed letter grade policy; i.e. the final letter grades will **not** be A=90-100%, B=80-90%, etc.



# Exam Review

- Node Iterator

# Design Pattern Question

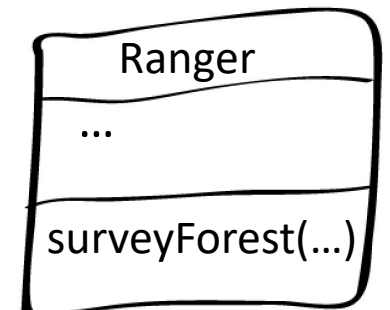
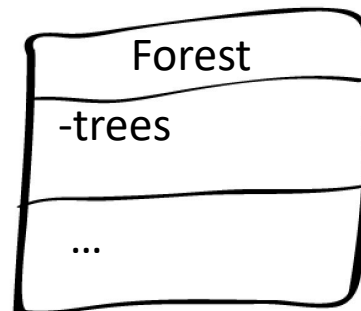
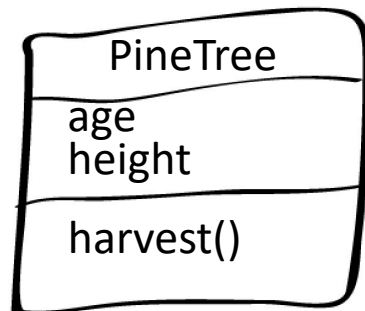
- Strategy vs Template Method Patterns

# Representational gap

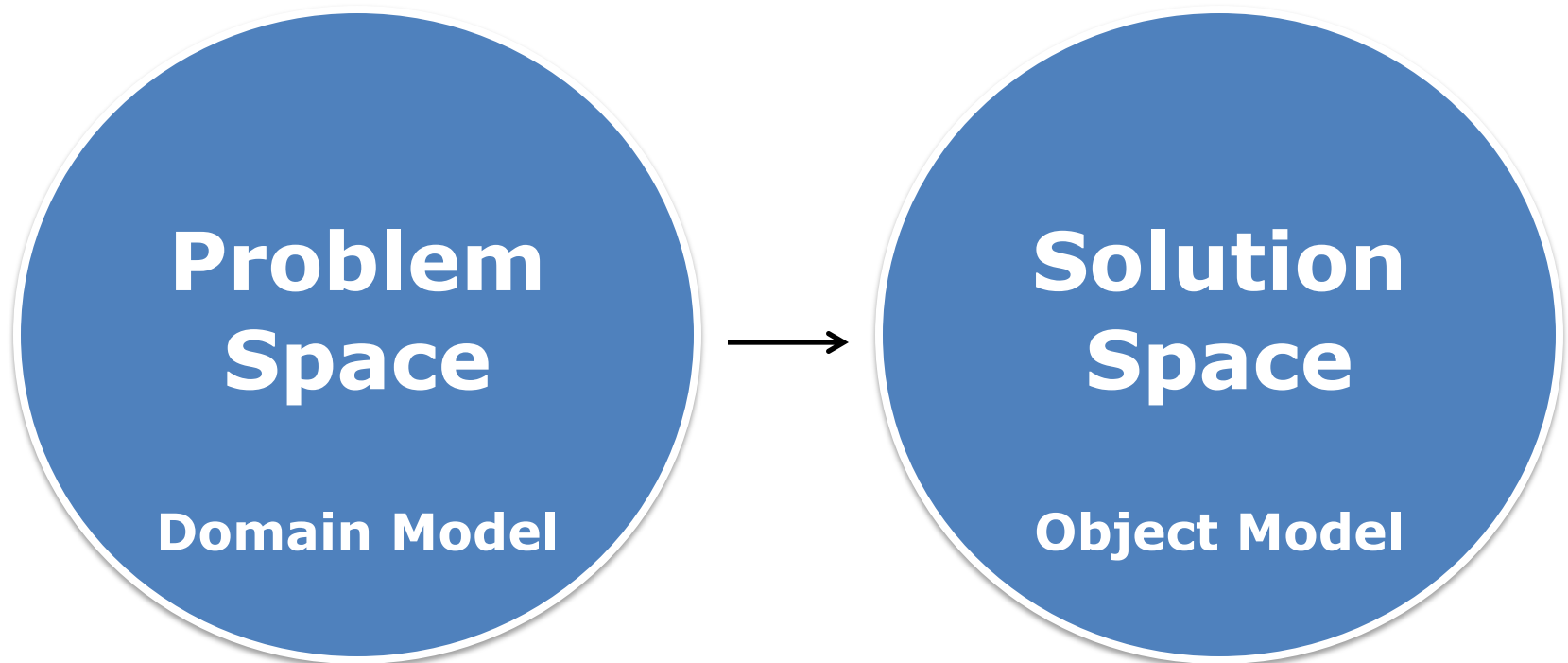
- Real-world concepts:



- Software concepts:




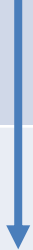
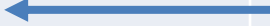
# Our path toward a more formal design process



- Real-world concepts
- Requirements, concepts
- Relationships among concepts
- Solving a problem
- Building a vocabulary

- System implementation
- Classes, objects
- References among objects and inheritance hierarchies
- Computing a result
- Finding a solution

# Design Process

	Modeling objects	Describing interaction
Understanding the Problem (Problem Level)	Domain Model 	System Sequence Diagram 
Defining a Solution (Code Level)	Object Model 	Object Interaction Diagrams

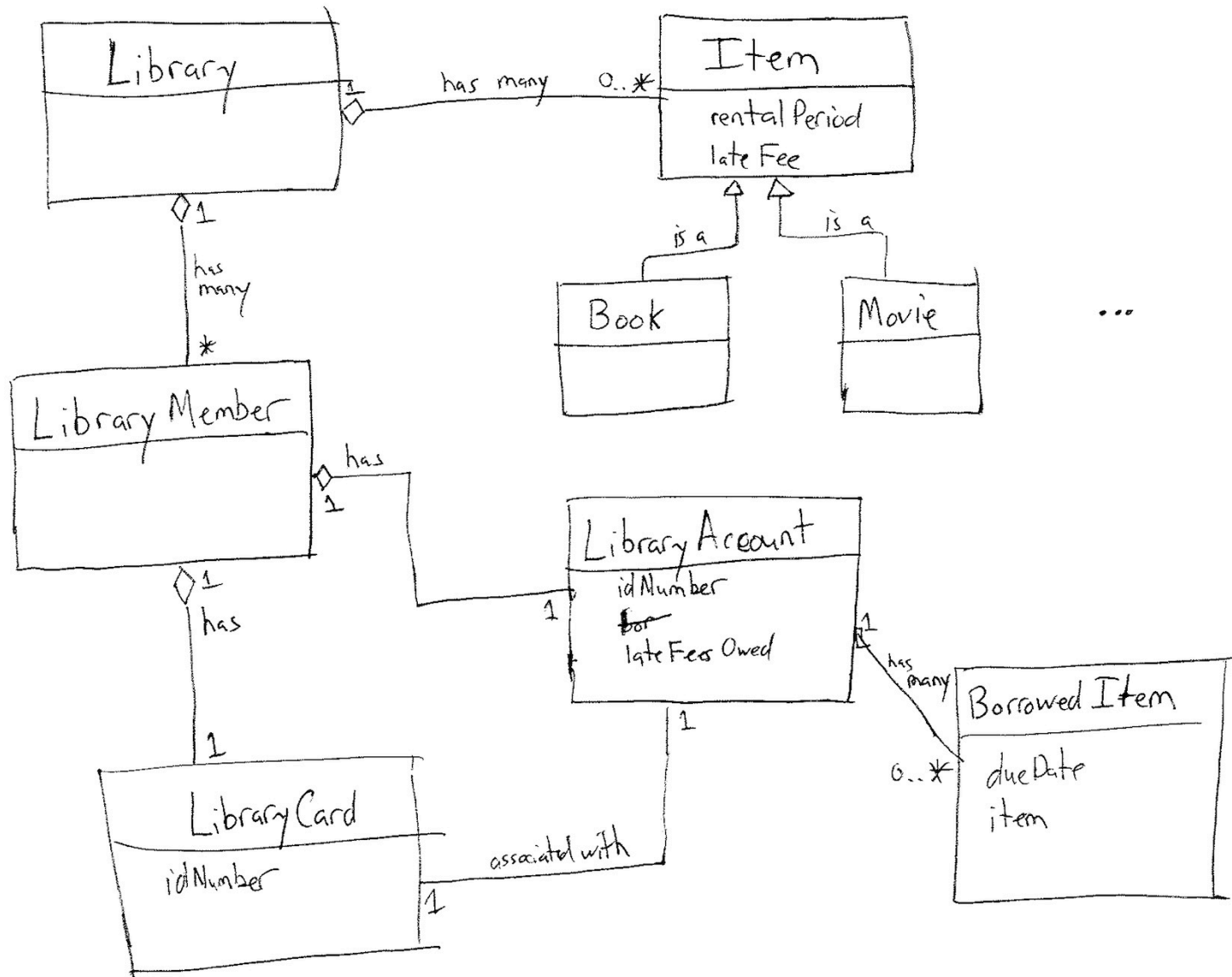


# Building a domain model for a library system

A public library typically stores a collection of books, movies, or other library items available to be borrowed by people living in a community. Each library member typically has a library account and a library card with the account's ID number, which she can use to identify herself to the library.

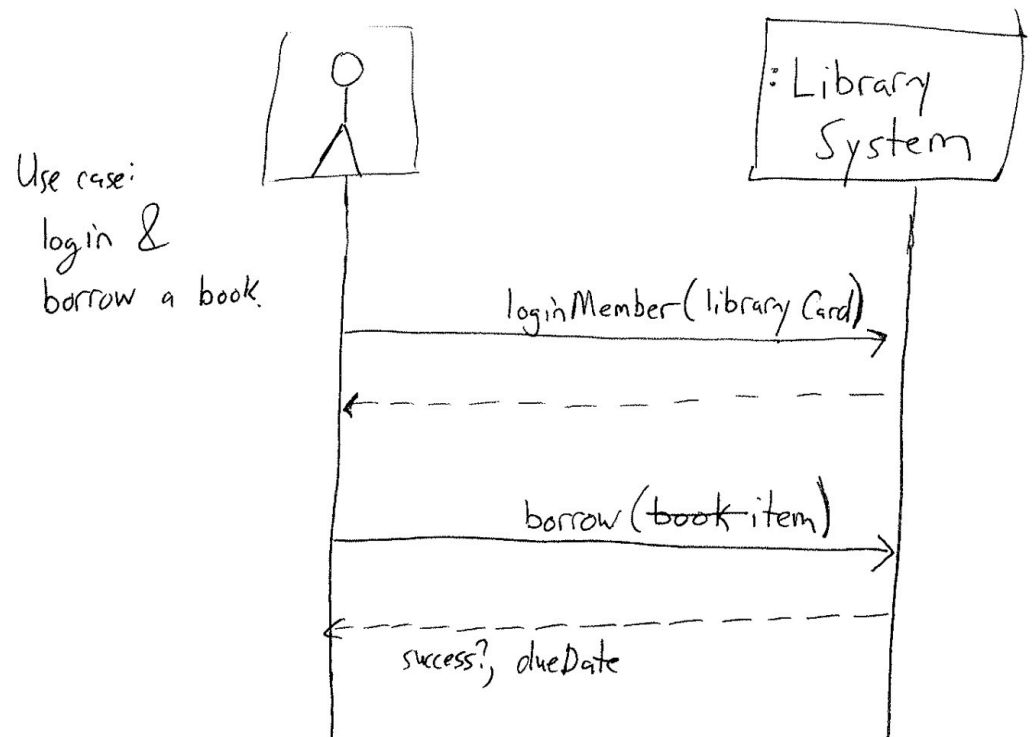
A member's library account records which items the member has borrowed and the due date for each borrowed item. Each type of item has a default rental period, which determines the item's due date when the item is borrowed. If a member returns an item after the item's due date, the member owes a late fee specific for that item, an amount of money recorded in the member's library account.

# One domain model for the library system



# One sequence diagram for the library system

Use case scenario: A library member should be able to use her library card to log in at a library system kiosk 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 rental period to the current day, and record the book and its due date as a borrowed item in the member's library account.



# A system behavioral contract for the library system

Operation:            borrow(item)

Pre-conditions:    Library member has already logged in to the system.  
Item is not currently borrowed by another member.

Post-conditions:   Logged-in member's account records the newly-borrowed item, or the member is warned she has an outstanding late fee.  
The newly-borrowed item contains a future due date, computed as the item's rental period plus the current date.

# Distinguishing domain vs. implementation concepts

- Domain-level concepts:
  - Almost anything with a real-world analogue
- Implementation-level concepts:
  - Implementation-like method names
  - Programming types
  - Visibility modifiers
  - Helper methods or classes
  - Artifacts of design patterns

# Summary: Understanding the problem domain

- Know your tools to build domain-level representations
  - Domain models
  - System sequence diagrams
  - System behavioral contracts
- Be fast and (sometimes) loose
  - Elide obvious(?) details
  - Iterate, iterate, iterate, ...
- Get feedback from domain experts
  - Use only domain-level concepts

# Artifacts of our 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
- 
- Understanding the problem
- 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

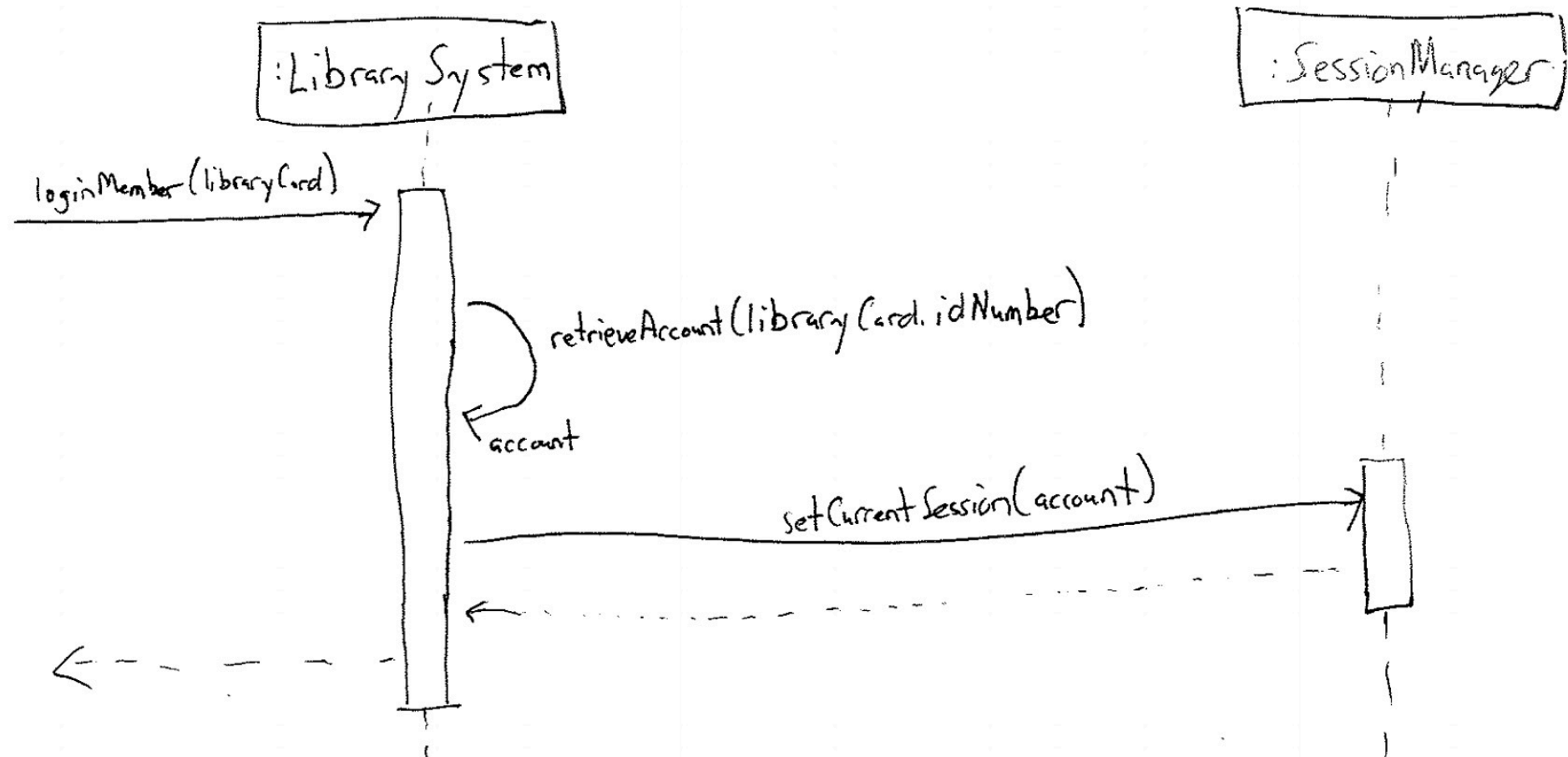
- Data:
  - Private or otherwise encapsulated data
  - Data in closely related objects
- Methods:
  - Private or otherwise encapsulated operations
  - Object creation, of itself or other objects
  - Initiating actions in other objects
  - Coordinating activities among objects

# 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 ...

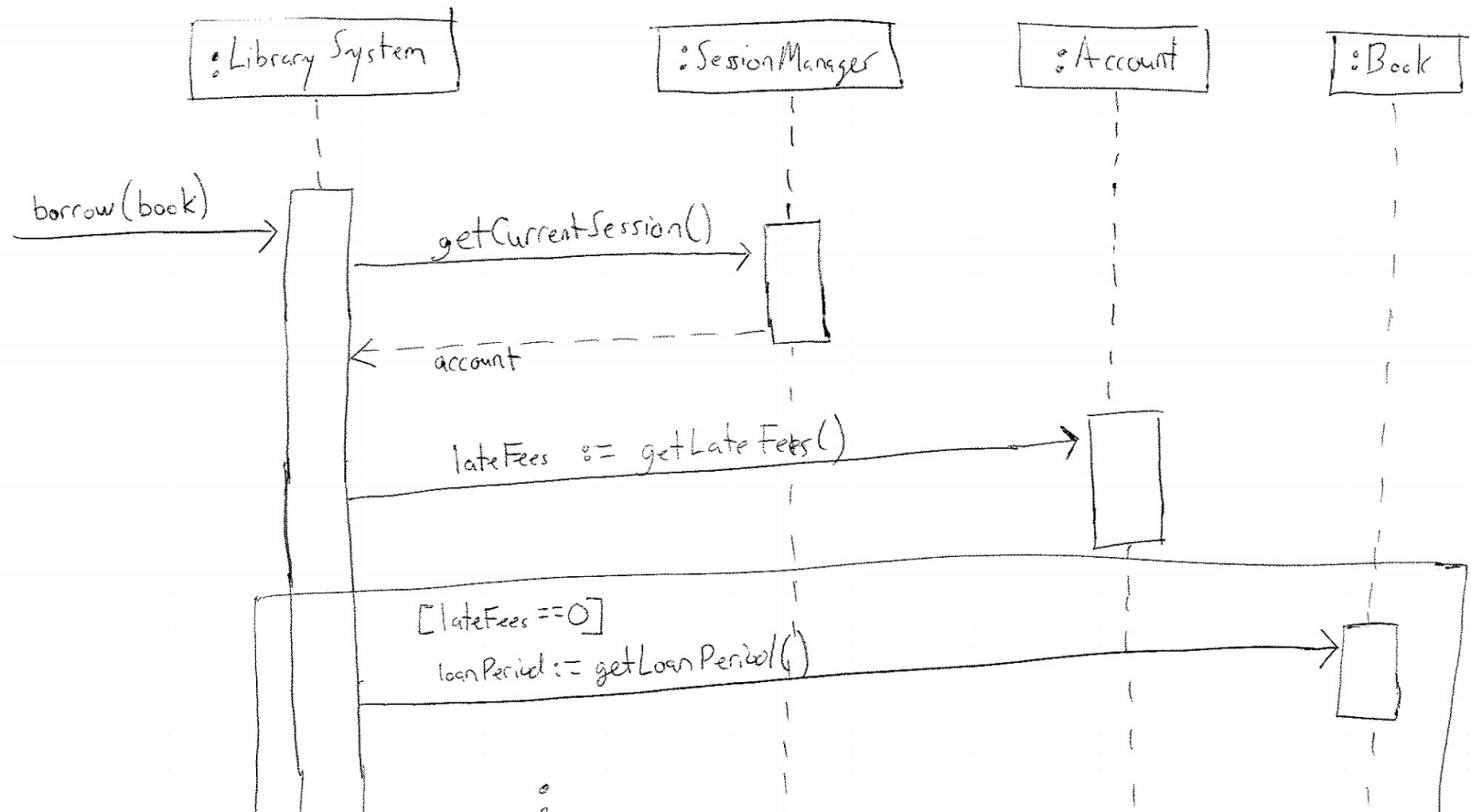


## Example interaction diagram #2

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.

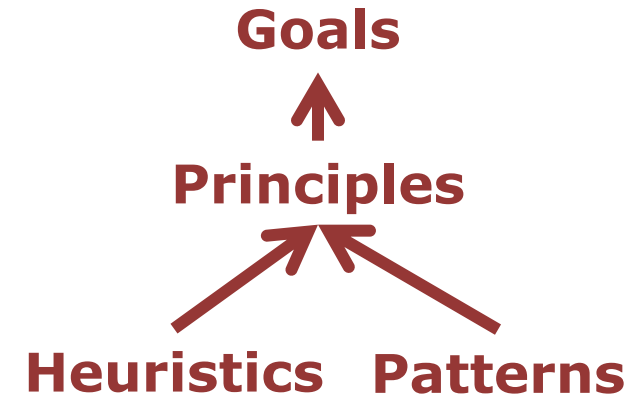
## Example interaction diagram #2

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.



# 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



# 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

# 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

## Another design principle: Minimize conceptual weight

- Label the concepts for a proposed object
  - Related to representational gap and cohesion

# Creator heuristic: Who creates an object Foo?

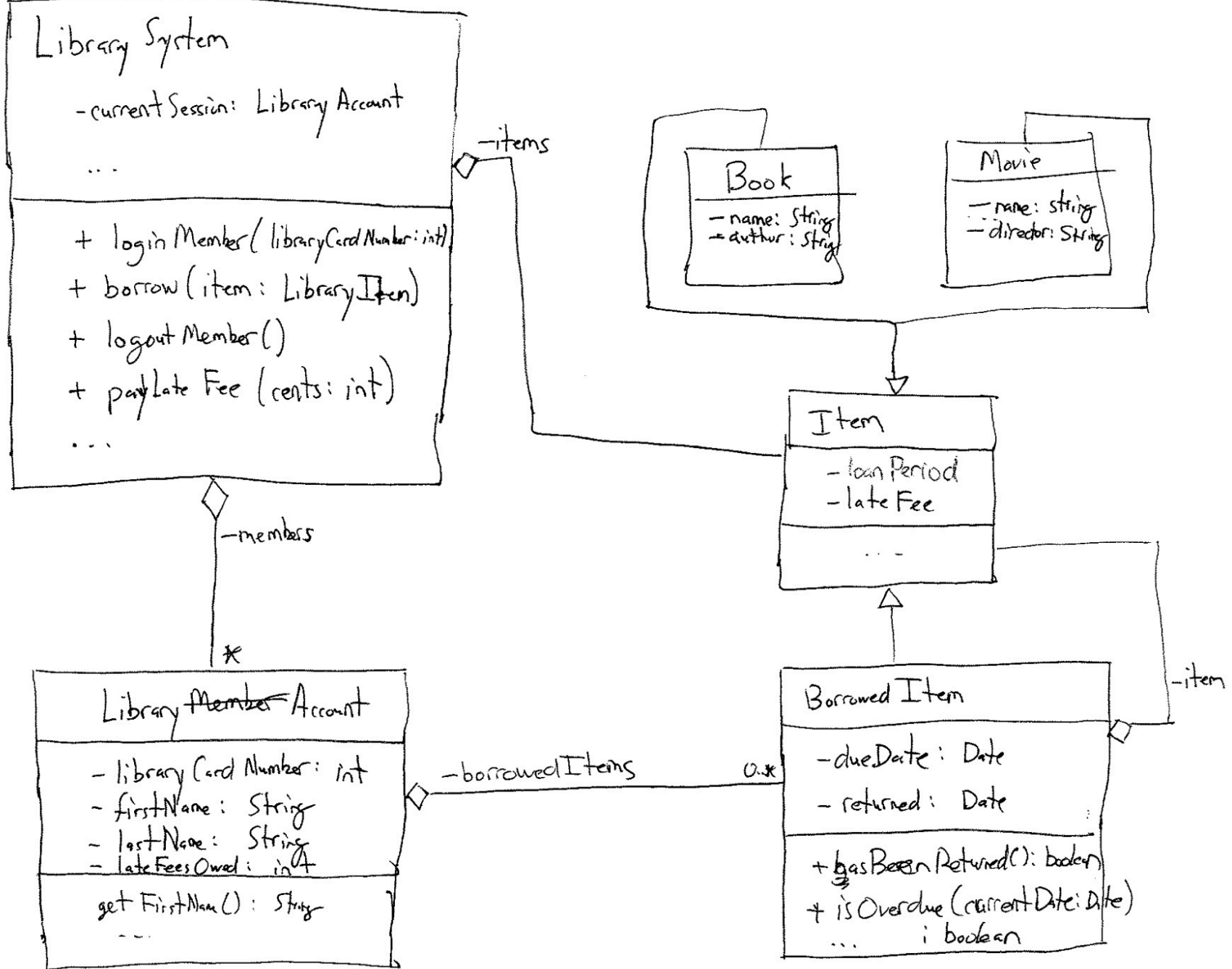
- Assign responsibility of creating an object Foo to a class that:
  - Has the data necessary for initializing instances of Foo
  - Contains, aggregates, or records instances of Foo
  - Closely uses or manipulates instances of Foo
- Design process: Extract from domain model, interaction diagrams
  -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

# 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



# 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...