Principles of Software Construction:
Objects, Design, and Concurrency

Designing (sub-) systems

Responsibility assignment

Charlie Garrod   **Michael Hilton**

institute for
SOFTWARE
RESEARCH

Administrivia

- Reading for Today:
  - UML and Patterns Chapter 14, 15, and 16
- Midterm exam next Thursday (September 28th)
  - Exam review session: Hamerschlag Hall B103 **Wed 7-9pm**
  - Practice Exam posted on Piazza
- Muddiest Point Feedback

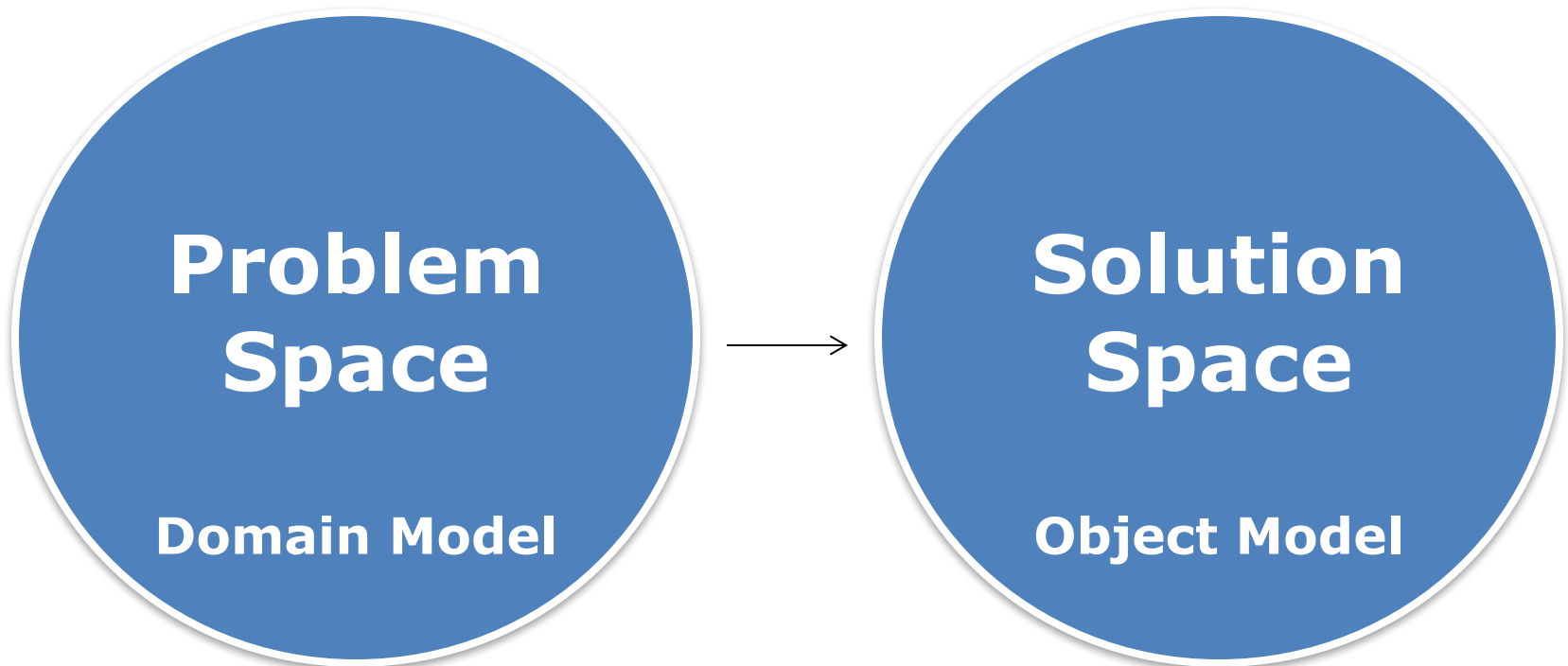institute for
SOFTWARE
RESEARCH

Homework 4

- Three parts, part A due Oct 5th
    - Design review meeting next week

institute for
SOFTWARE
RESEARCH

REVIEW: Sequence and communication diagrams
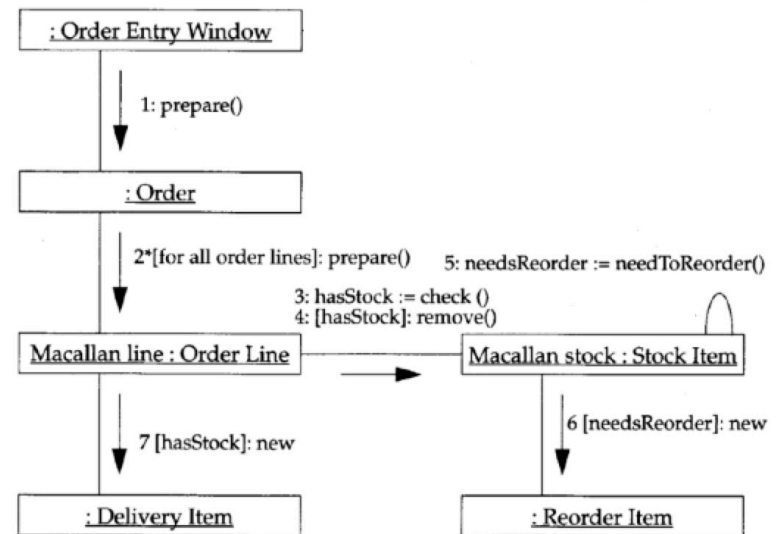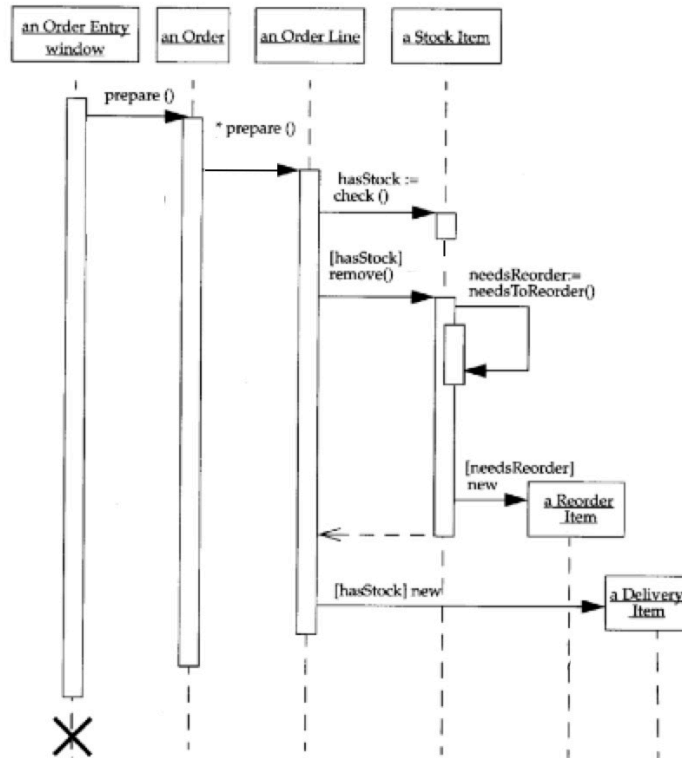
Our path toward a more formal design process

Problem Space

Domain Model

→

Solution Space

Object Model

- 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

isr institute for SOFTWARE RESEARCH

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**

Understanding the problem

- Assign object responsibilities, define interactions
  - Object interaction diagrams
- Model / diagram a potential solution
  - Object model

Defining a solution

isr institute for SOFTWARE RESEARCH

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
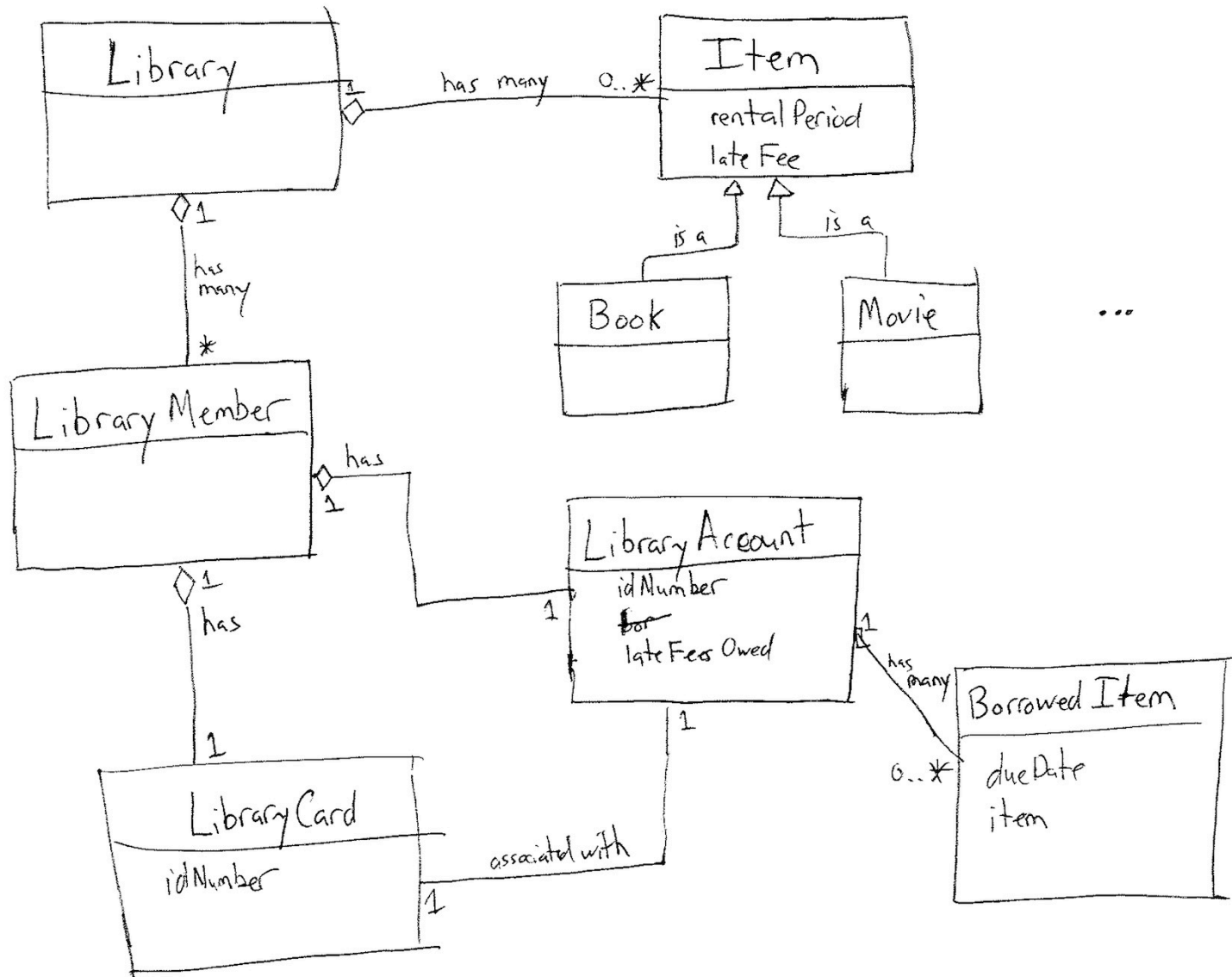
Notes on the library domain model

- All concepts are accessible to a non-programmer
- The UML is somewhat informal
  - Relationships are often described with words
- Real-world "is-a" relationships are appropriate for a domain model
- Real-word abstractions are appropriate for a domain model
- Iteration is important
  - This example is a first draft. Some terms (e.g. Item vs. LibraryItem, Account vs. LibraryAccount) would likely be revised in a real design.
- Aggregate types are usually modeled as classes
- Primitive types (numbers, strings) are usually modeled as attributes

Build a domain model for Monopoly

10

Build a domain model for Monopoly

Monopoly is a game in which each player has a piece that moves around a game board, with the piece's change in location determined by rolling a pair of dice. The game board consists of a set of properties (initially owned by a bank) that may be purchased by the players.

When a piece lands on a property that is not owned, the player may use money to buy the property from the bank for that property's price. If a player lands on a property she already owns, she may build houses and hotels on the property; each house and hotel costs some price specific for the property. When a player's piece lands on a property owned by another player, the owner collects money (rent) from the player whose piece landed on the property; the rent depends on the number of houses and hotels built on the property.

The game is played until only one remaining player has money and property, with all the other players being bankrupt.

Understanding system behavior with sequence diagrams

- A *system sequence diagram* is a model that shows, for one scenario of use, the sequence of events that occur on the system's boundary

- Design goal: Identify and define the interface of the system
  - Two components: A user and the overall system

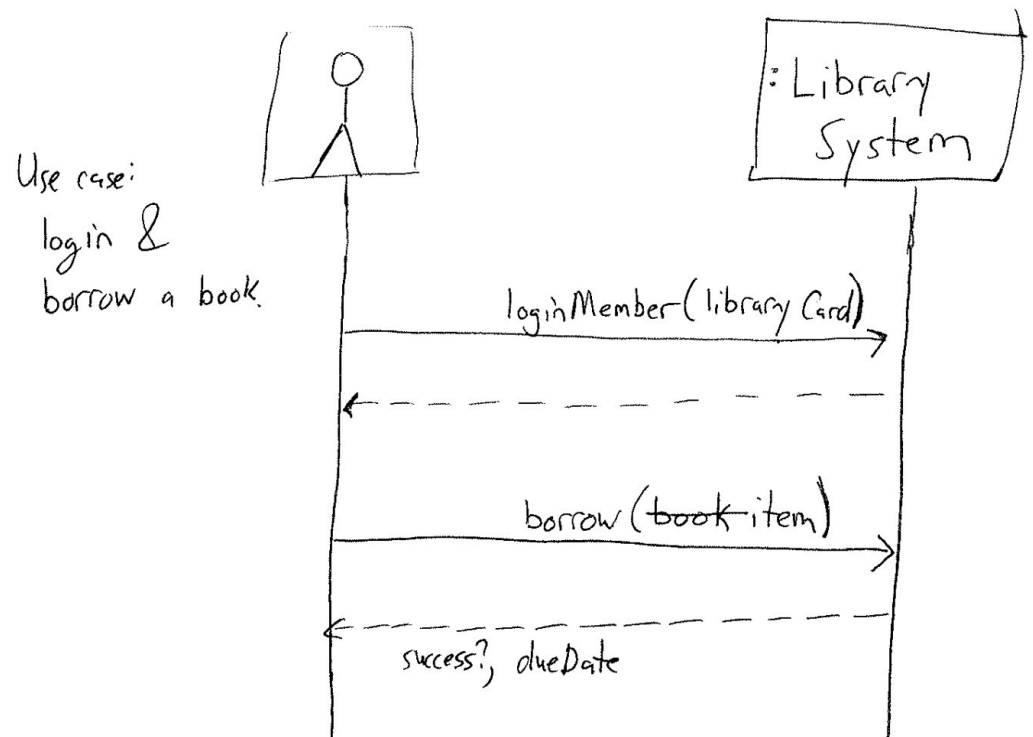# Understanding system behavior with sequence diagrams

- A *system sequence diagram* is a model that shows, for one scenario of use, the sequence of events that occur on the system's boundary

- Design goal: Identify and define the interface of the system
  - Two components: A user and the overall system

- Input: Domain description and one use case

- Output: A sequence diagram of system-level operations
  - Include only domain-level concepts and operations

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.

14

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.

Build one system sequence diagram for Monopoly

Use case scenario: When a player lands on an unowned property and has enough money to buy the property, she should be able to buy the property for the property's price. The property should no longer be purchasable from the bank by other players, and money should be moved from the player to the bank.

institute for
SOFTWARE
RESEARCH

Formalize system behavior with behavioral contracts

- A *system behavioral contract* describes the pre-conditions and post-conditions for some operation identified in the system sequence diagrams
  - System-level textual specifications, like software specifications

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.

A system behavioral contract for Monopoly

Operation: buy(property)

Pre-conditions: ?

Post-conditions: ?

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

institute for
SOFTWARE
RESEARCH

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 ✓

Understanding the problem

- Assign object responsibilities, define interactions
  - **Object interaction diagrams**
- Model / diagram a potential solution
  - Object model

Defining a solution

isr institute for SOFTWARE RESEARCH

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; }
}
```

institute for
SOFTWARE
RESEARCH

Object-Oriented Design

- Object-Oriented Design: "After identifying your requirements and creating a domain model, then add methods to the software classes, and define the messaging between the objects to fulfill the requirements."

- But how?
  - How should concepts be implemented by classes?
  - What method belongs where?
  - How should the objects interact?
  - This is a critical, important, and non-trivial task

institute for SOFTWARE RESEARCH

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

Responsibilities

- Responsibilities are related to the obligations of an object in terms of its behavior.

- Two types of responsibilities:
  - knowing
  - doing

- Doing responsibilities of an object include:
  - doing something itself, such as creating an object or doing a calculation
  - initiating action in other objects
  - controlling and coordinating activities in other objects

- Knowing responsibilities of an object include:
  - knowing about private encapsulated data
  - knowing about related objects
  - knowing about things it can derive or calculate

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 …
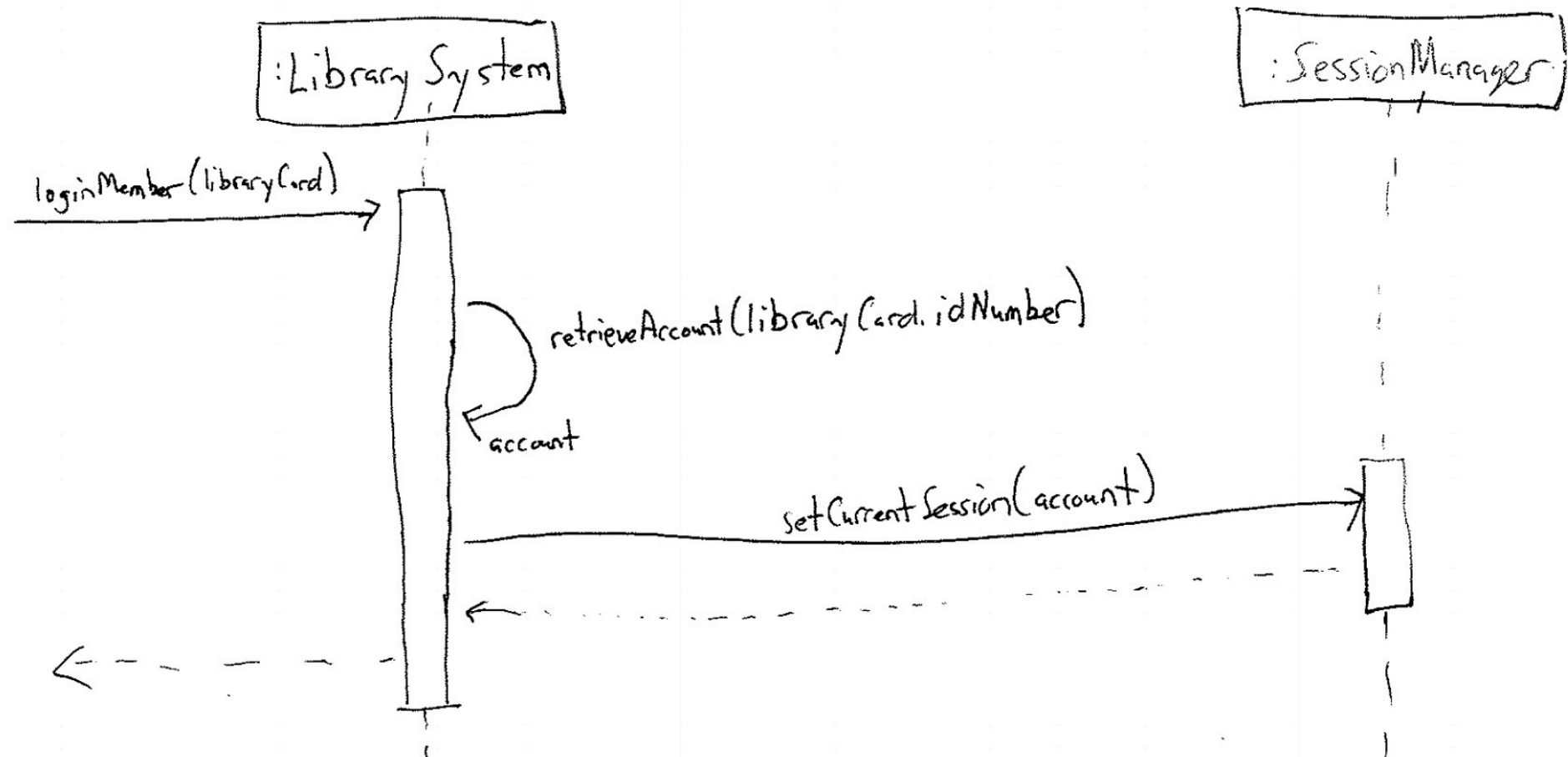
Example interaction diagram #2

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.
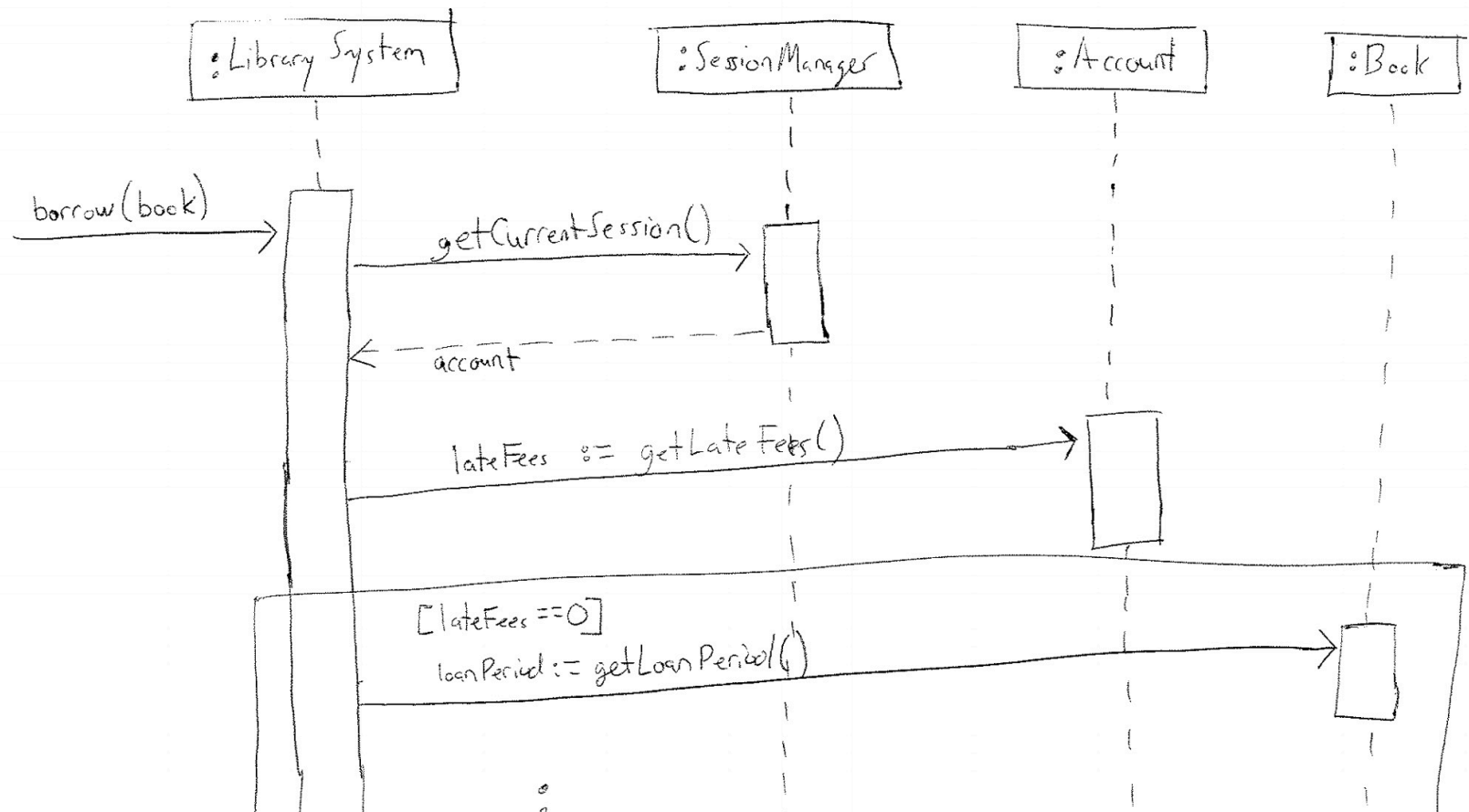
Interaction diagrams help evaluate design alternatives

Create two possible interaction diagrams:

1. Solving a cryptarithm, assuming that the cryptarithm class has responsibility for solving itself

2. Solving a cryptarithm, assuming that the main method (or a delegated method or class) has responsibility for solving the cryptarithm

Goals, Principles, Guidelines

- Design Goals
  - Desired quality attributes of software
  - Driven by cost/benefit economics
  - Examples: design for change, understanding, reuse, …
- Design Principles
  - Guidelines for designing software
  - Support one or more design goals
  - Examples: Information hiding, low repr. gap, low coupling, high cohesion, …
- Design Heuristics
  - Rules of thumb for **low-level** design decisions
  - Promote design principles, and ultimately design goals
  - Example: Creator, Expert, Controller
- Design Patterns
  - General solutions to recurring design problems
  - Promote design goals, but may add complexity or involve tradeoffs
  - Examples: Decorator, Strategy, Template Method
- Goals, principles, heuristics, patterns may conflict
  - Use high-level goals of project to resolve

Goals

↑

Principles

↖ ↗

Heuristics   Patterns
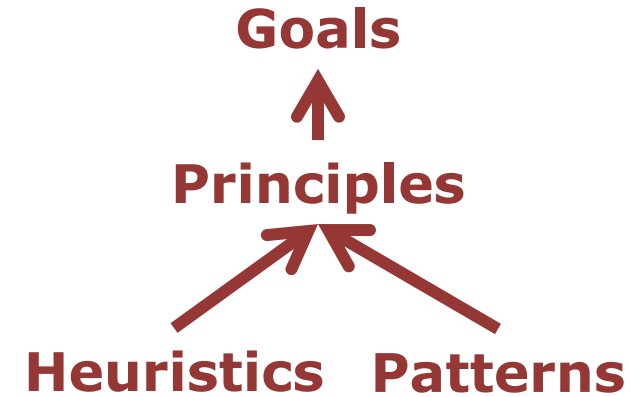
Heuristics for responsibility assignment

- Controller heuristic

- Information expert heuristic

- Creator heuristic

Goals

↑

Principles

Heuristics **Patterns**
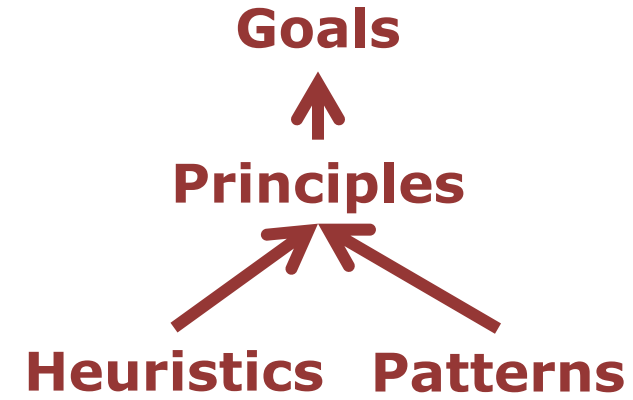
The controller heuristic

- 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

isr institute for SOFTWARE RESEARCH

Use the information expert heuristic

- In Homework 3, what object should have the responsibility to solve a cryptarithm?

- What is the relevant information?

institute for
SOFTWARE
RESEARCH

Use the information expert heuristic

- In Homework 3, what object should have the responsibility to solve a cryptarithm?

- What is the relevant information?
  - Who knows the # of digits (e.g. base 10) in the cryptarithm?
  - Who knows the letters of the cryptarithm?
  - Who can evaluate the cryptarithm expressions to check for equality?

institute for
SOFTWARE
RESEARCH

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

Use the creator heuristic

- In Homework 3, what object should have the responsibility for creating the permutation generator?

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

institute for
SOFTWARE
RESEARCH

Creating an object model

- Extract data, method names, and types from interaction diagrams
  - Include implementation details such as visibilities
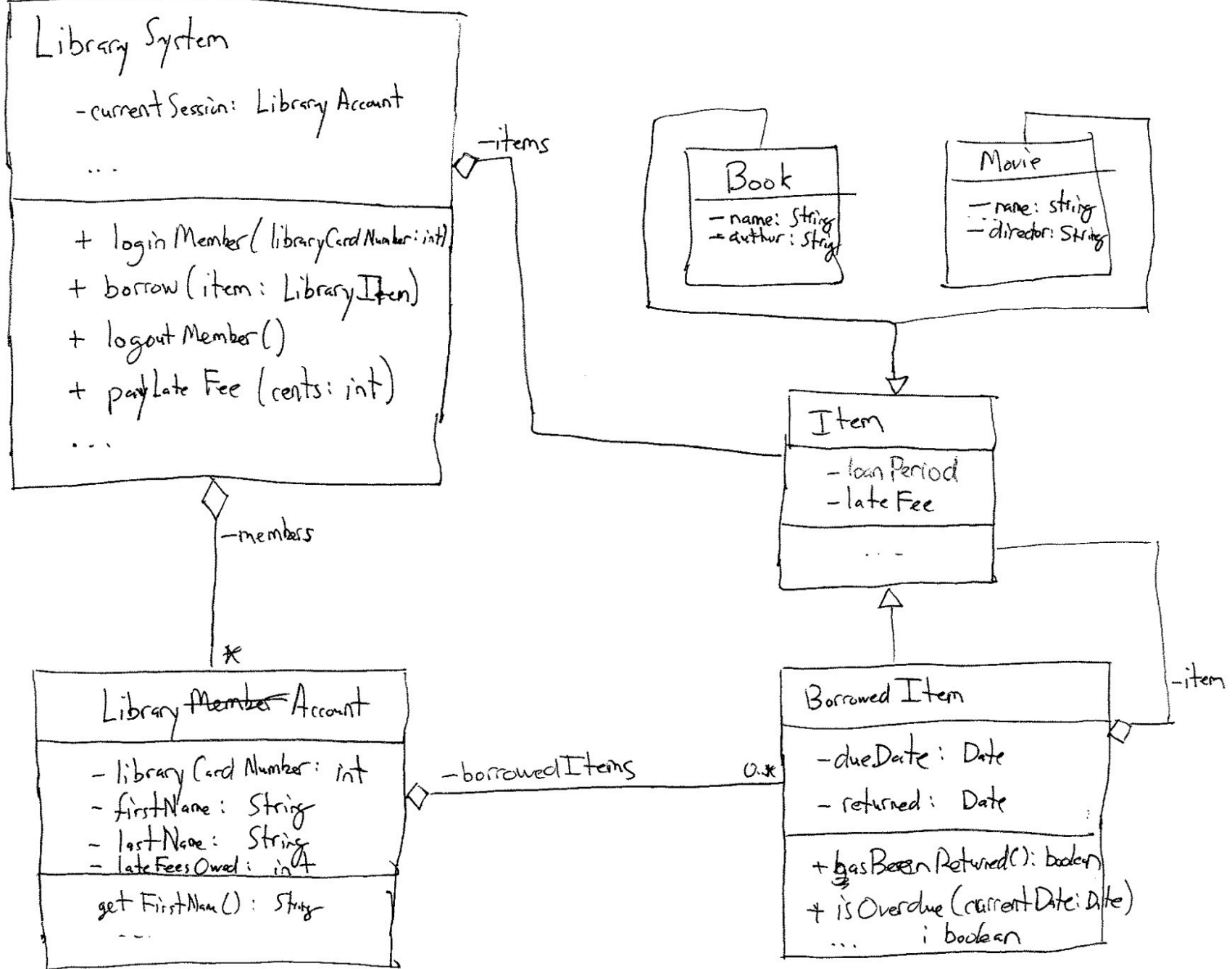
Create an object model for your cryptarithm solver

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…