

Principles of Software Construction: Objects, Design, and Concurrency

Designing (sub-) systems

A formal design process

Michael Hilton

Bogdan Vasilescu

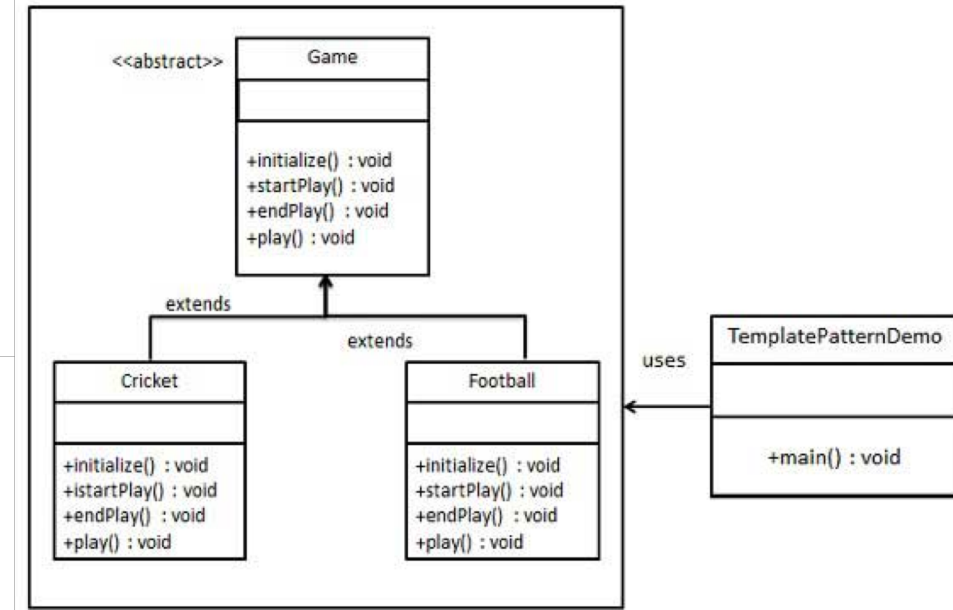
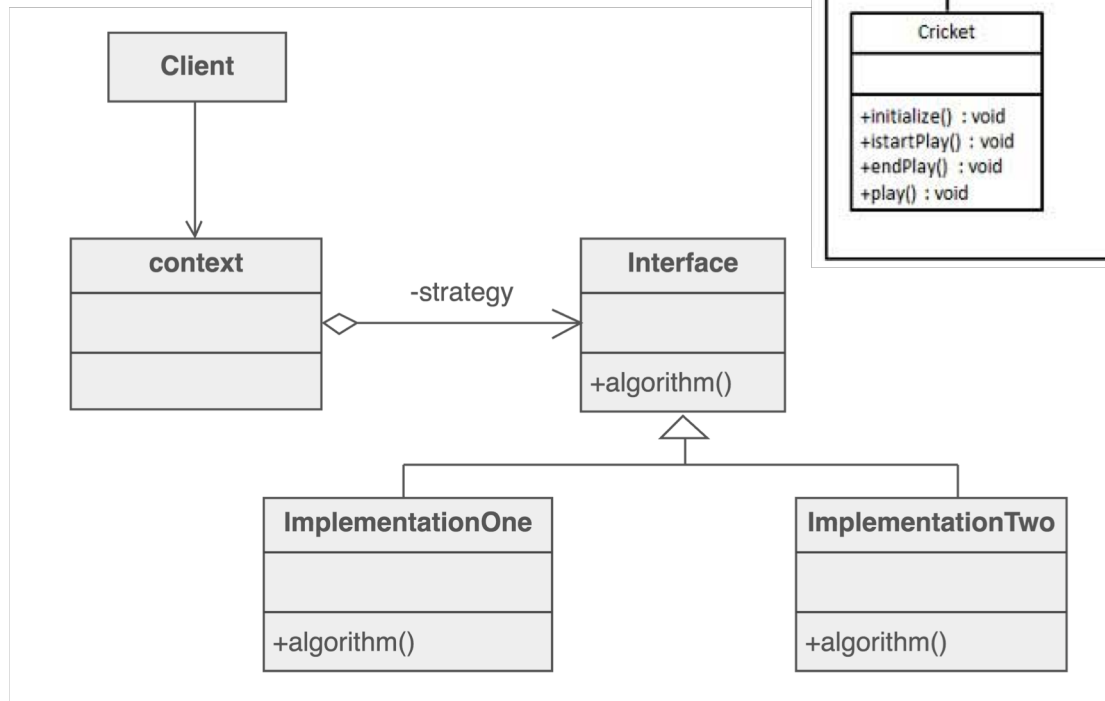


Administrivia

- HW 3 deadline Today instead of Thu, Feb 7
- Reading Today: UML and Patterns, Ch. 14, 15, and 16
- Midterm 1 on Thu, Feb 14
 - Review meeting: Wed, Feb 13, 6-8pm, Scaife Hall 125
- Recitation tomorrow: Review the rules of blackjack the card game

Key concepts from last Tuesday

- Template Method Pattern
- Strategy Pattern
- Iterator Pattern
- Decorator Pattern



Inspired by Design Patterns



Inspired by Design Patterns



Today

- Design goals and design principles

Metrics of software quality, i.e., *design goals*

Functional correctness Adherence of implementation to the specifications

Robustness Ability to handle anomalous events

Flexibility Ability to accommodate changes in specifications

Reusability Ability to be reused in another application

Efficiency Satisfaction of speed and storage requirements

Scalability Ability to serve as the basis of a larger version of the application

Security Level of consideration of application security

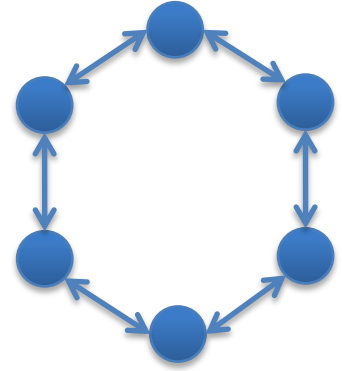
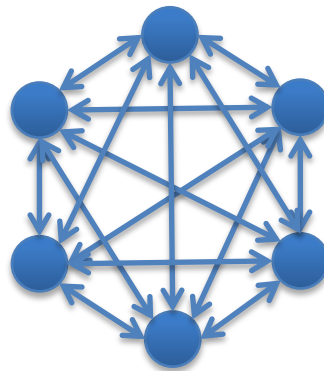
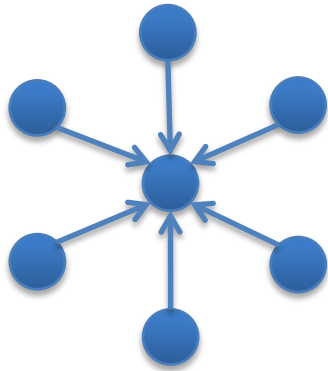
**Source: Braude, Bernstein,
Software Engineering. Wiley 2011**

Design principles: heuristics to achieve design goals

- Low coupling
- Low representational gap
- High cohesion

A design principle for reuse: *low coupling*

- Each component should depend on as few other components as possible



- Benefits of low coupling:
 - Enhances understandability
 - Reduces cost of change
 - Eases reuse

Law of Demeter

- "Only talk to your immediate friends"

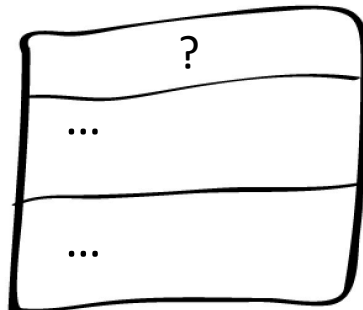
~~foo.bar().baz().quz(42)~~

Representational gap

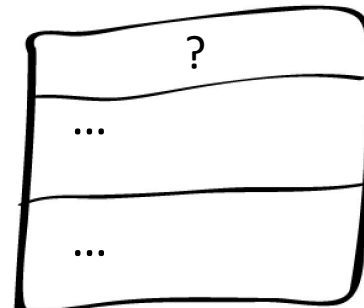
- Real-world concepts:



- Software concepts:



...

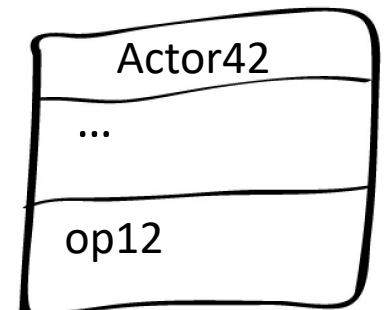
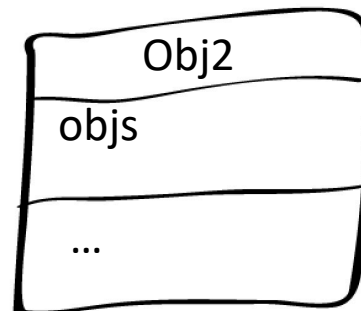
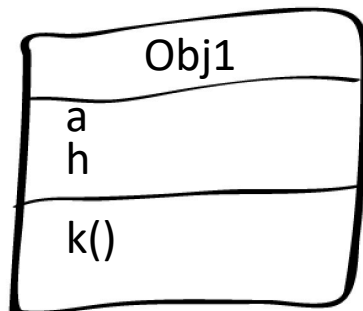


Representational gap

- Real-world concepts:



- Software concepts:

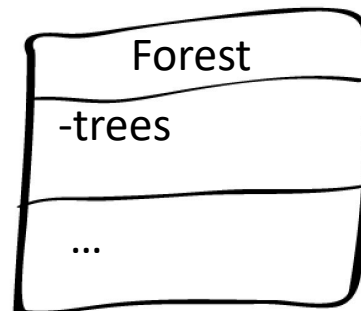
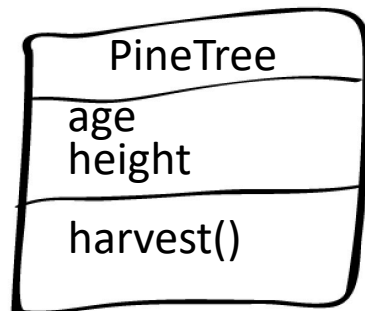


Representational gap

- Real-world concepts:



- Software concepts:

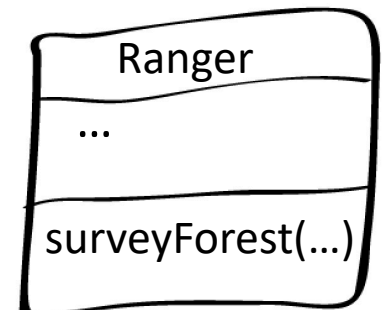
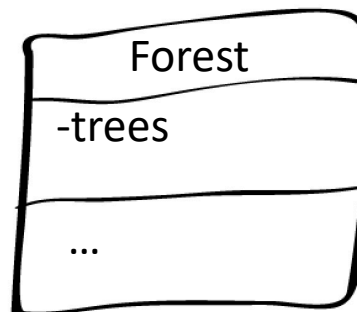
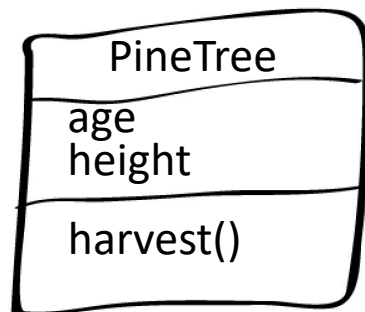


Benefits of low representational gap

- Facilitates understanding of design and implementation
- Facilitates traceability from problem to solution
- Facilitates evolution

A related design principle: high cohesion

- Each component should have a small set of closely-related responsibilities
- Benefits:
 - Facilitates understandability
 - Facilitates reuse
 - Eases maintenance



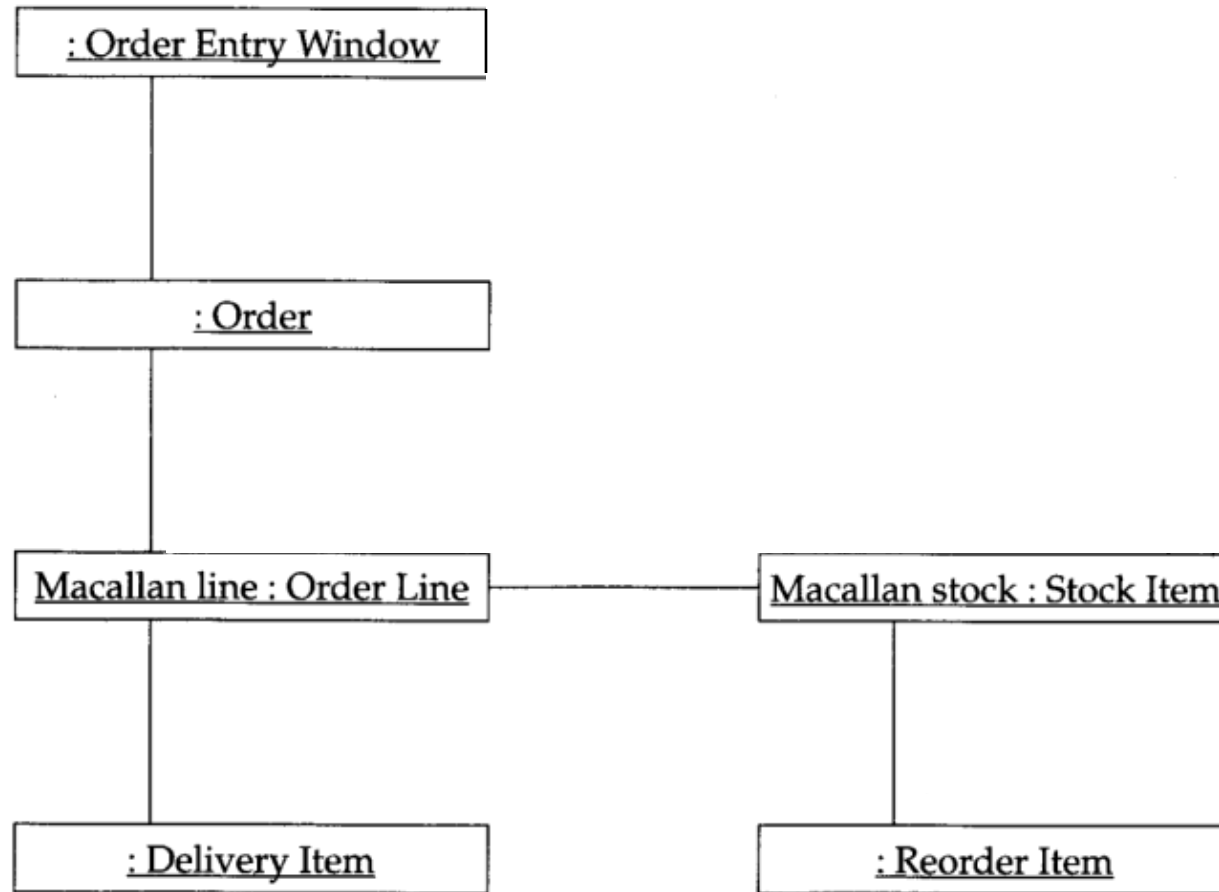
Coupling vs. cohesion

- All code in one component?
 - Low cohesion, low coupling
- Every statement / method in a separate component?
 - High cohesion, high coupling

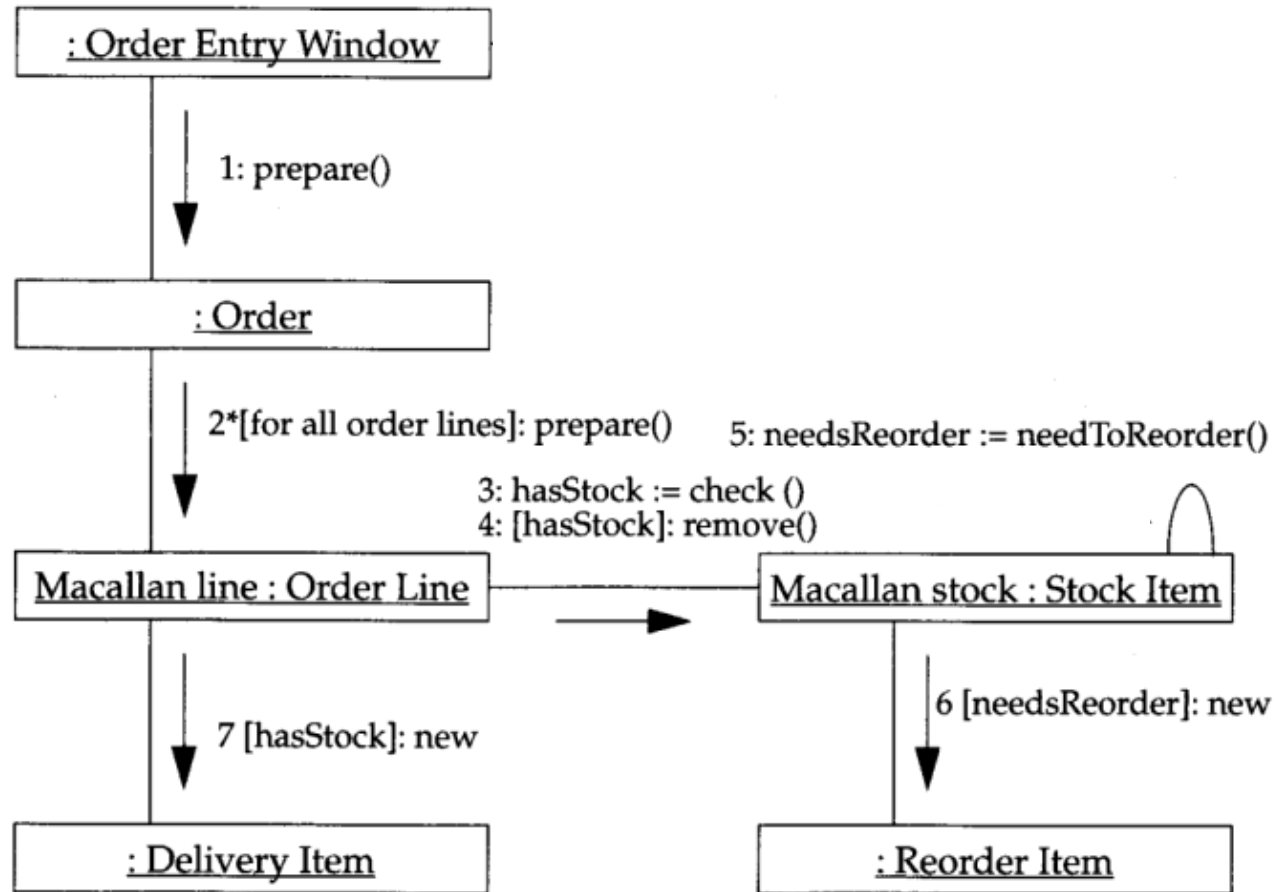
Visualizing dynamic behavior: Interaction diagrams

- An *interaction diagram* is a picture that shows, for a single scenario of use, the events that occur across the system's boundary or between subsystems
- Clarifies interactions:
 - Between the program and its environment
 - Between major parts of the program
- For this course, you should know:
 - Communication diagrams
 - Sequence diagrams

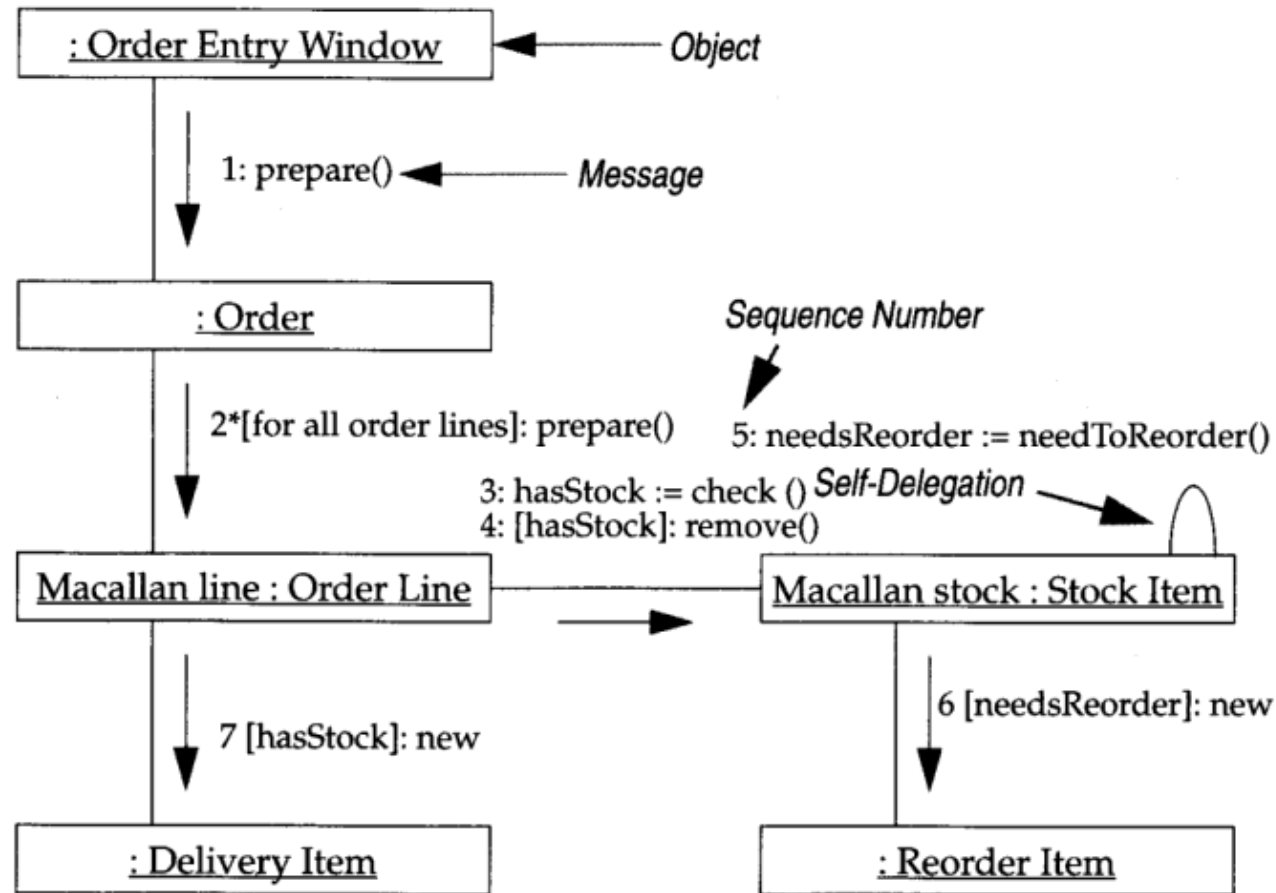
Creating a communication diagram



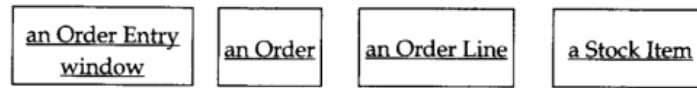
An example communication diagram



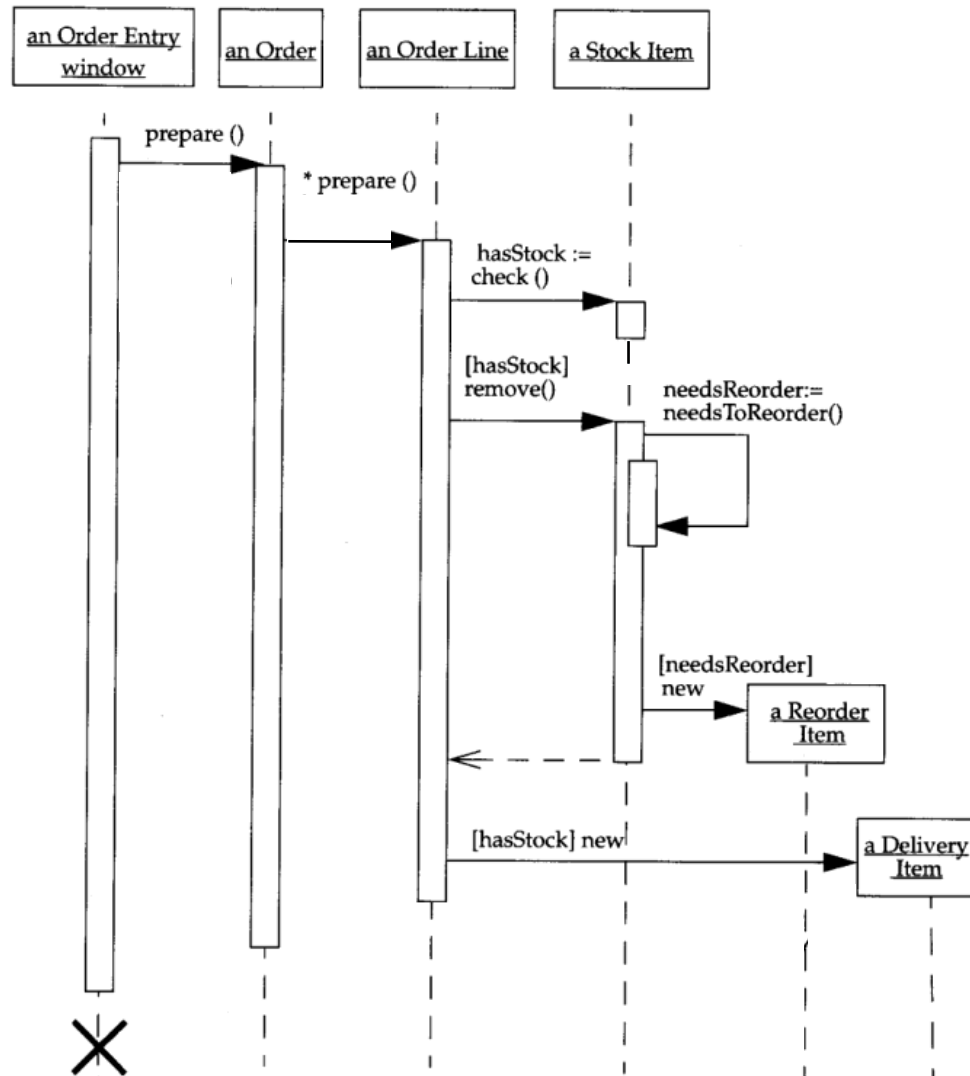
(Communication diagram with notation annotations)



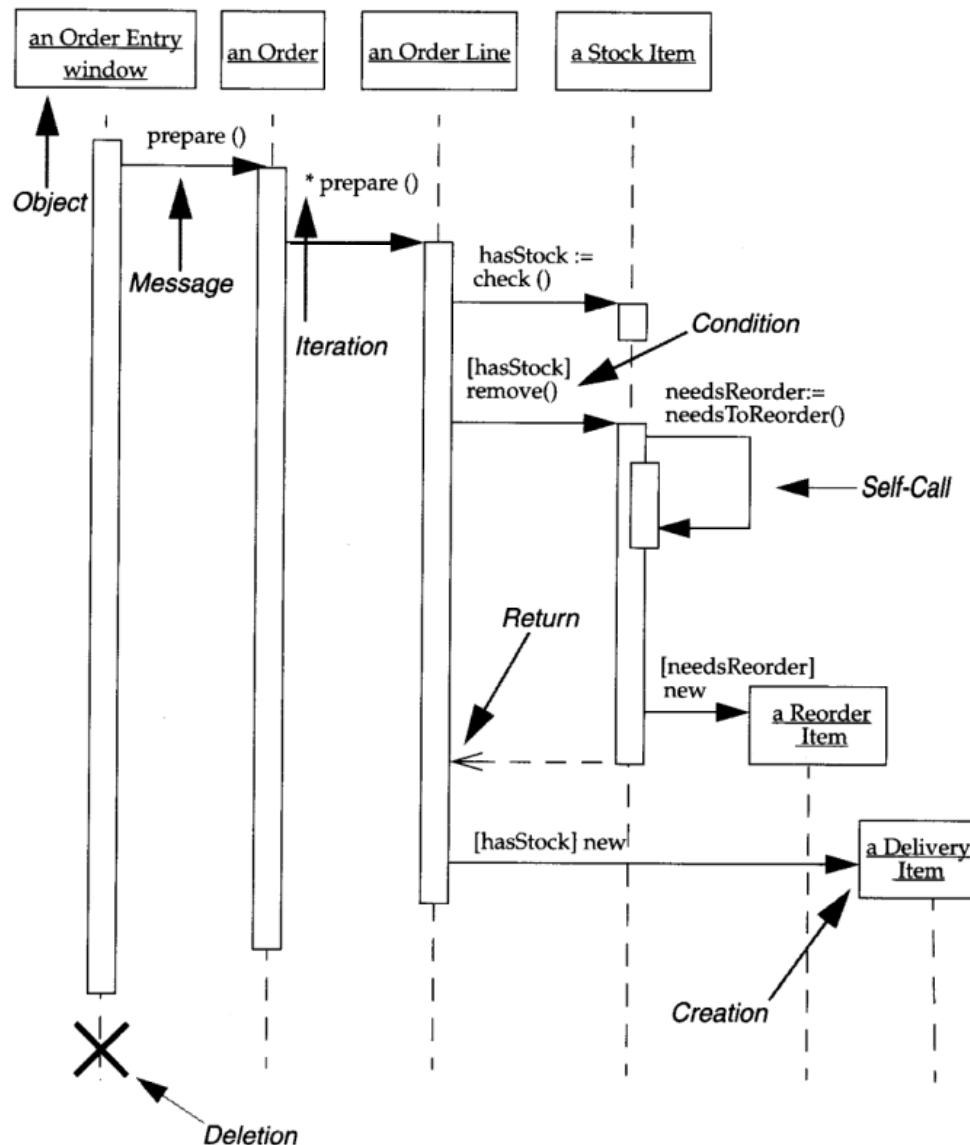
Constructing a sequence diagram



An example sequence diagram

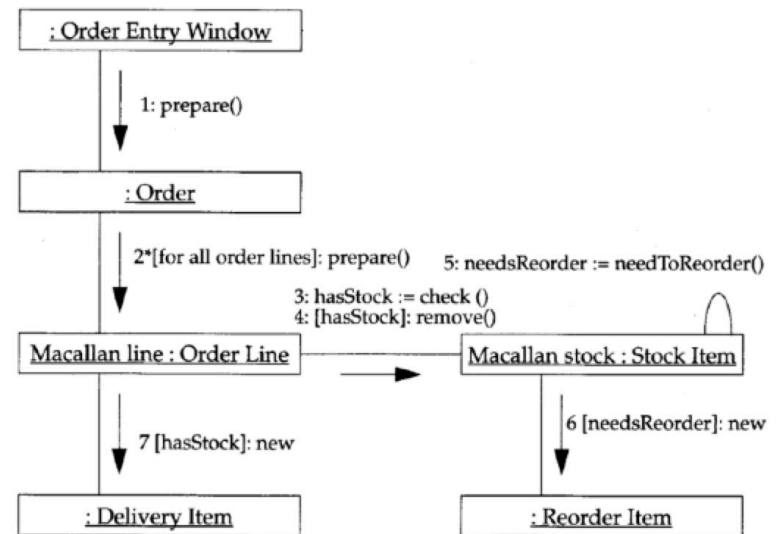
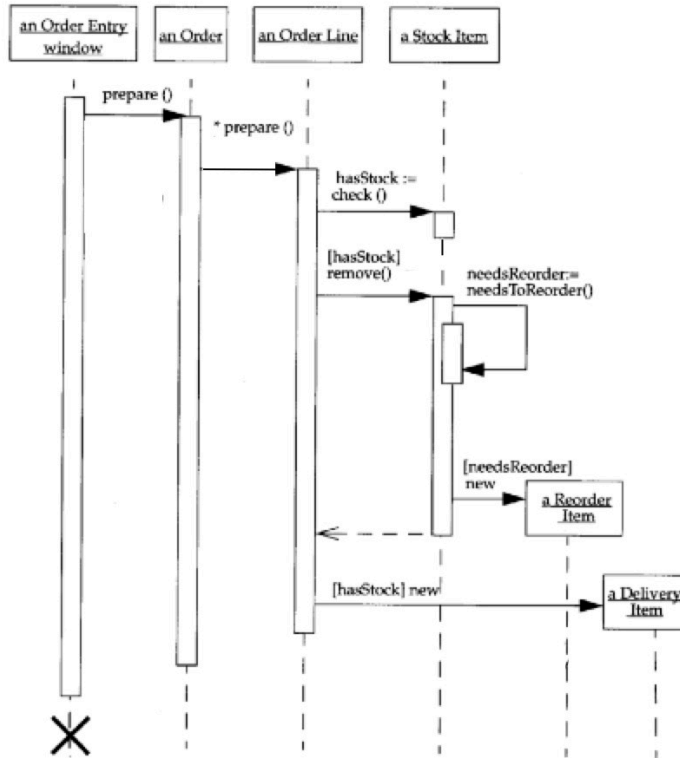


(Sequence diagram with notation annotations)



Sequence vs. communication diagrams

- Relative advantages and disadvantages?



DESIGN PROCESS

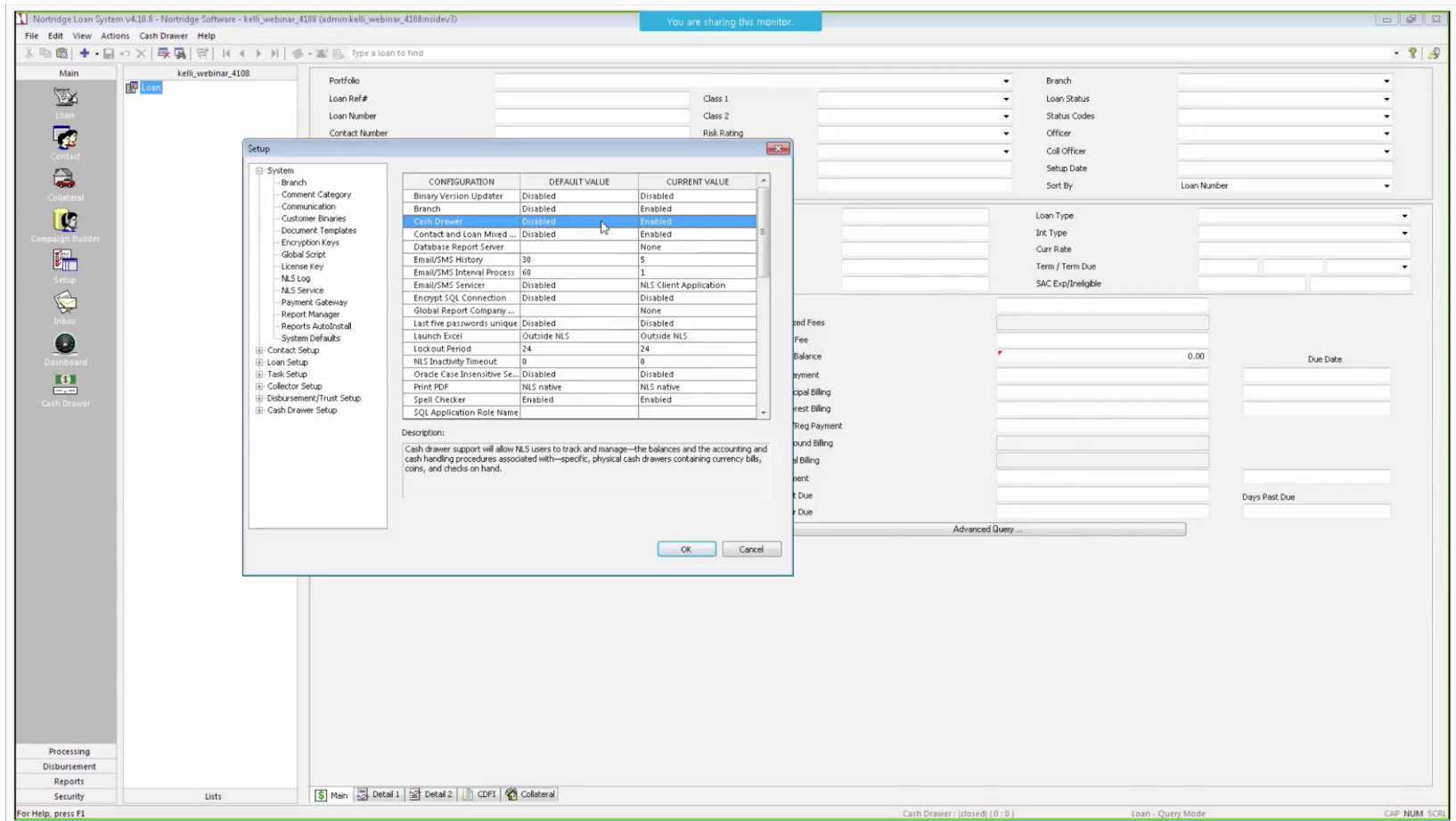
Tactical Data Radios



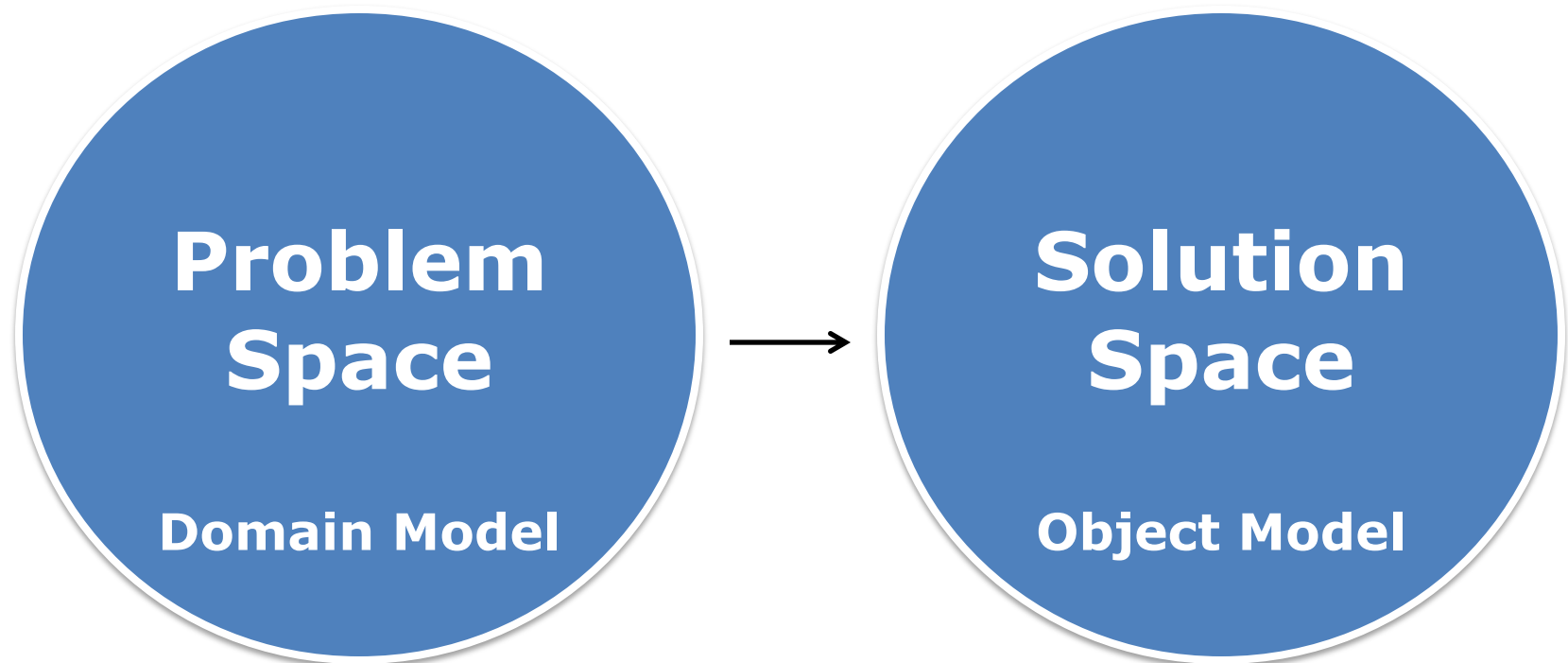
Coast Guard SAFE Boats



Loan Management Systems



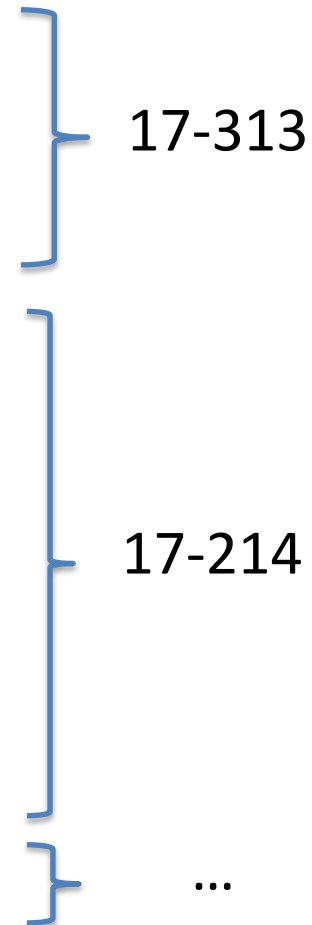
Our path toward a more formal design process



- 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

A high-level software design process

- Project inception
 - Gather requirements
 - Define actors, and use cases
 - Model / diagram the problem, define objects
 - Define system behaviors
 - Assign object responsibilities
 - Define object interactions
 - Model / diagram a potential solution
 - Implement and test the solution
 - Maintenance, evolution, ...
- 
- The diagram shows a list of 10 steps in a software design process. The first three steps are grouped by a blue bracket on the right, labeled '17-313'. The next five steps are grouped by another blue bracket on the right, labeled '17-214'. The final step is not grouped and is followed by an ellipsis '...'. The brackets are positioned to the right of the list, with the first bracket spanning the first three steps and the second bracket spanning the next five steps.
- 17-313
- 17-214
- ...



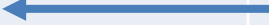
Artifacts of this design process

- 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

Artifacts of this 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

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

Input to the design process: Requirements and use cases

- Typically prose:

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. member must pay a fee for the member's library card.

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.

Modeling a problem domain

- Identify key concepts of the domain description
 - Identify nouns, verbs, and relationships between concepts
 - Avoid non-specific vocabulary, e.g. "system"
 - Distinguish operations and concepts
 - Brainstorm with a domain expert

Modeling a problem domain

- Identify key concepts of the domain description
 - Identify nouns, verbs, and relationships between concepts
 - Avoid non-specific vocabulary, e.g. "system"
 - Distinguish operations and concepts
 - Brainstorm with a domain expert
- Visualize as a UML class diagram, a *domain model*
 - Show class and attribute concepts
 - Real-world concepts only
 - No operations/methods
 - Distinguish class concepts from attribute concepts
 - Show relationships and cardinalities

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.

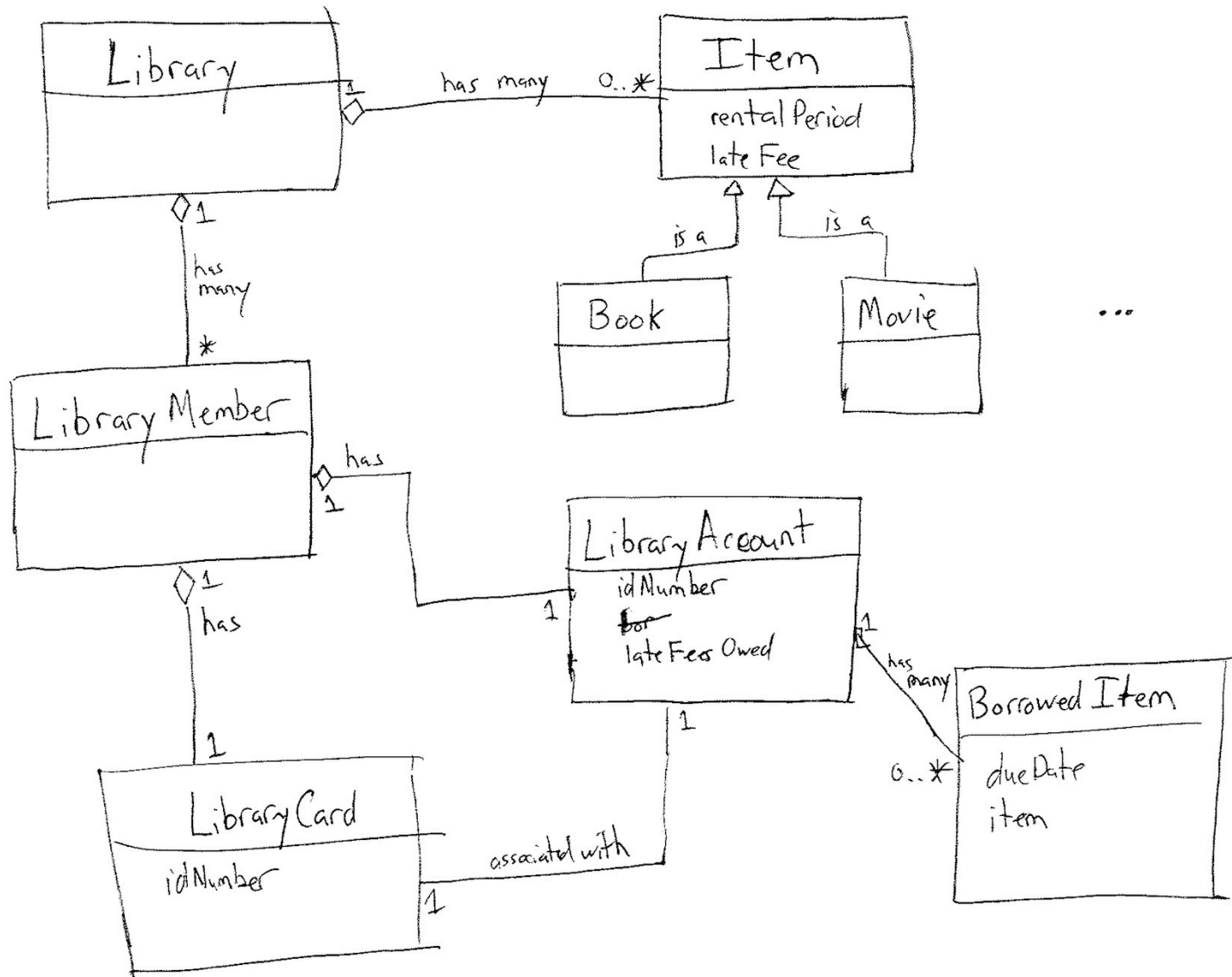
A member's library account records which items the member has borrowed and the due date for each borrowed item. Each type of item has a default rental period, which determines the item's due date when the item is borrowed. If a member returns an item after the item's due date, the member owes a late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

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.

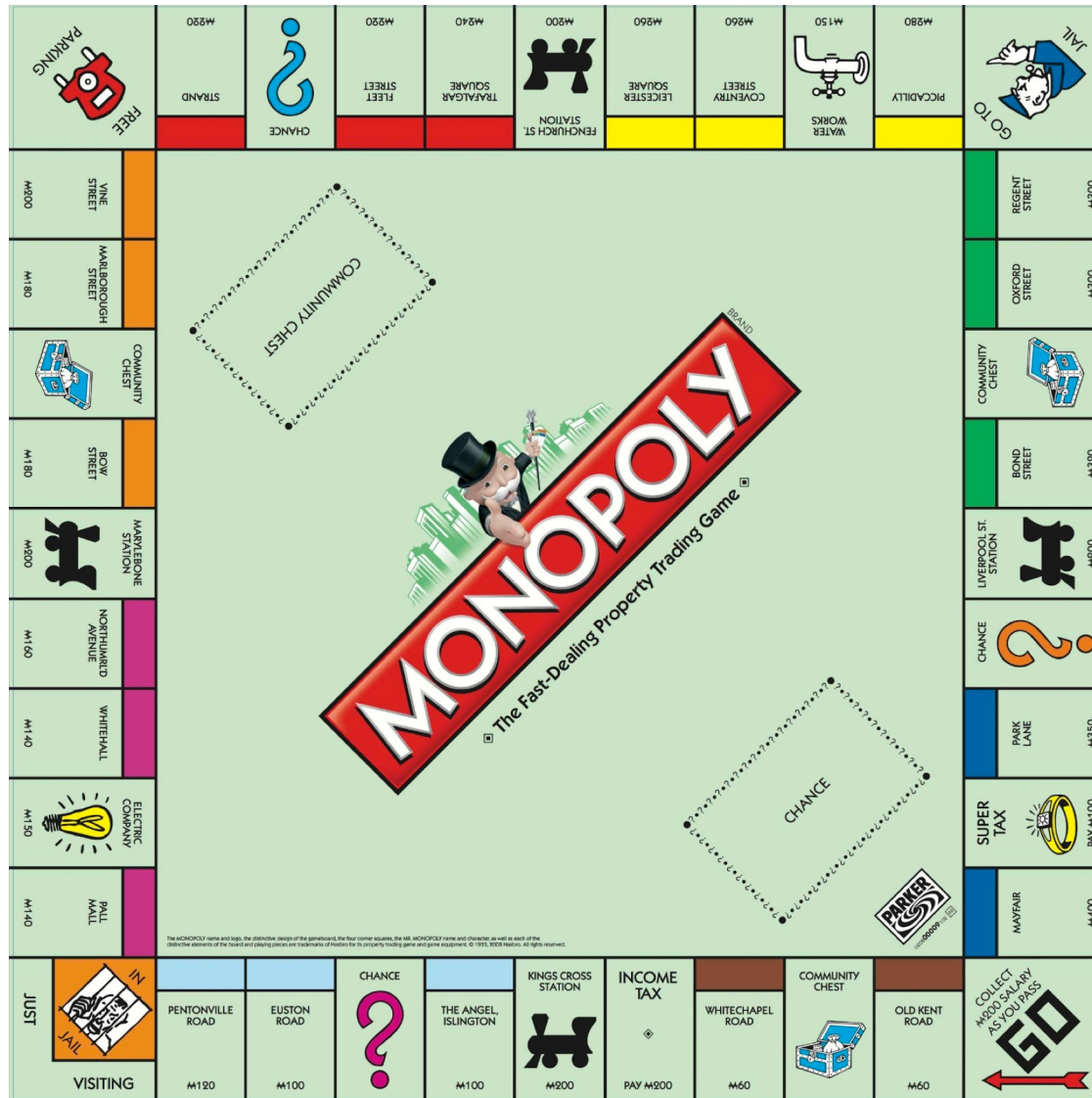
A member's library account records which items the member has borrowed and the due date for each borrowed item. Each type of item has a default rental period, which determines the item's due date when the item is borrowed. If a member returns an item after the item's due date, the member owes a late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

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

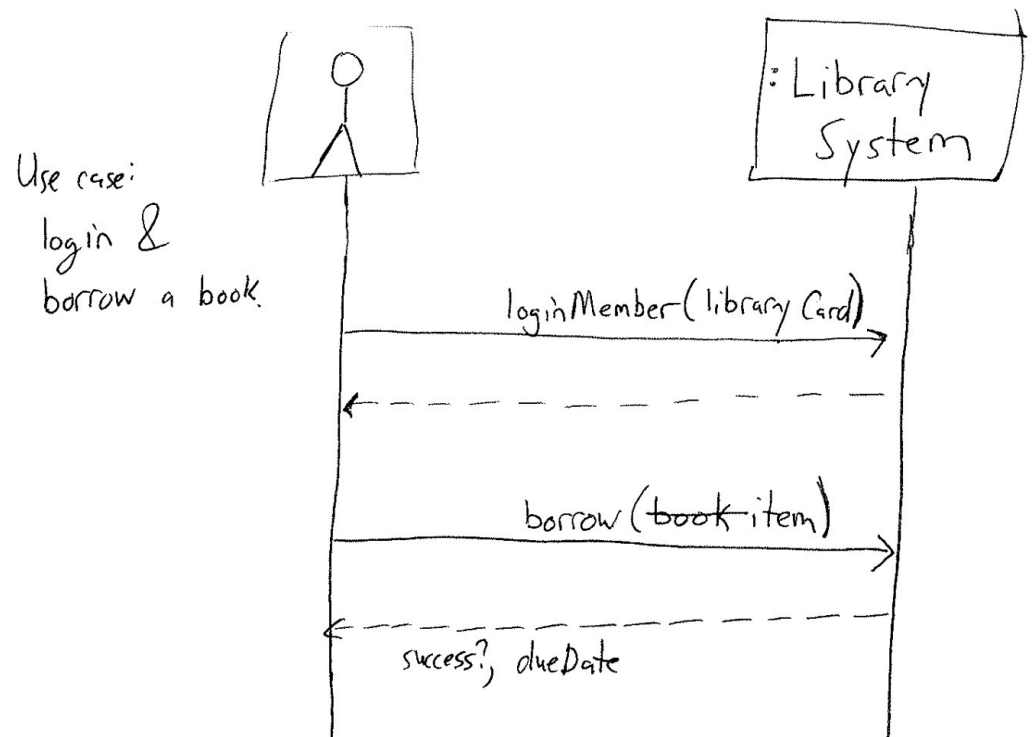
- A *system sequence diagram* is a model that shows, for one scenario of use, the sequence of events that occur on the system's boundary
- Design goal: Identify and define the interface of the system
 - Two components: A user and the overall system
- Input: Domain description and one use case
- Output: 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.

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.

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.

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

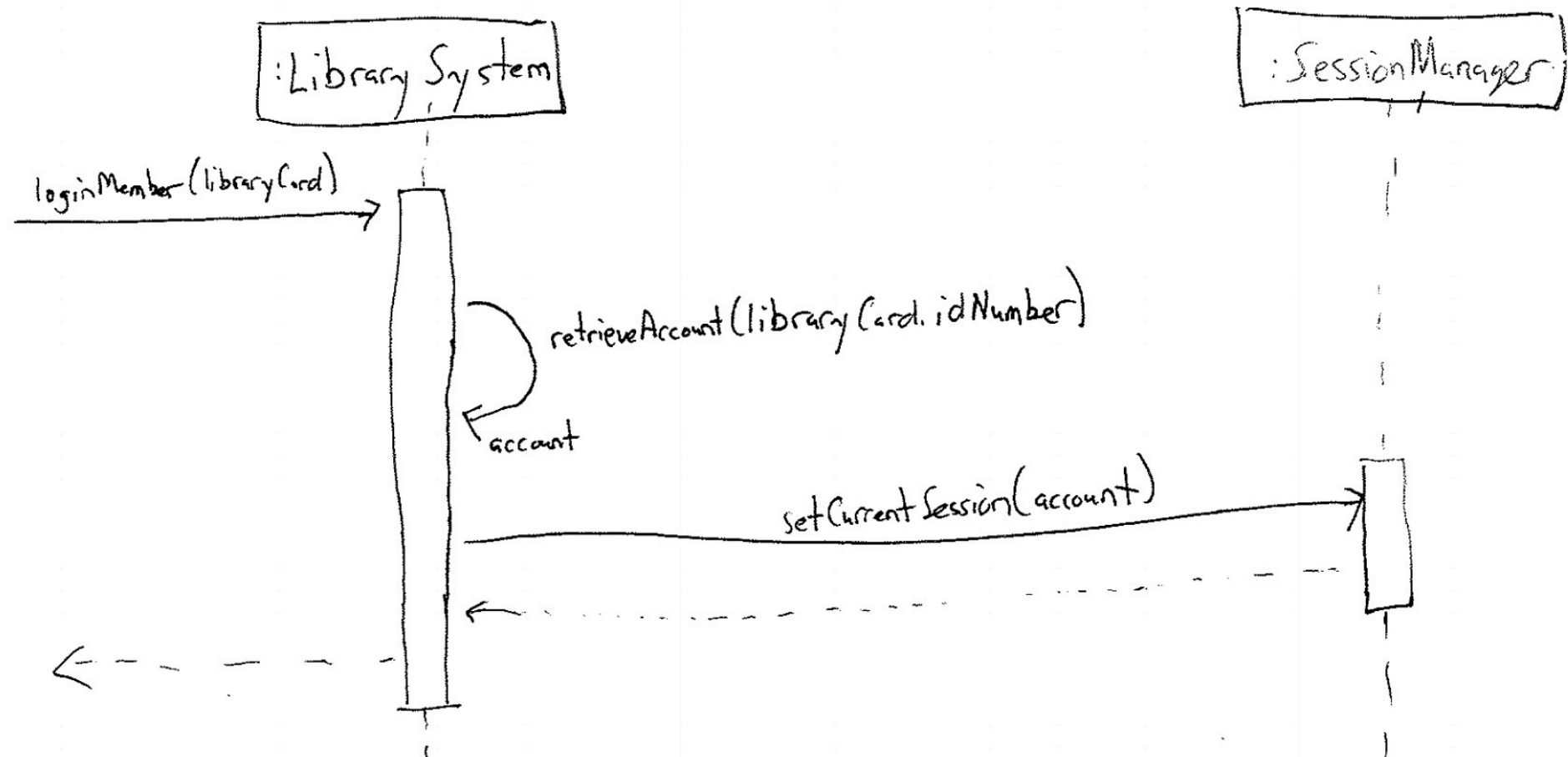
- Data:
 - Private or otherwise encapsulated data
 - Data in closely related objects
- Methods:
 - Private or otherwise encapsulated operations
 - Object creation, of itself or other objects
 - Initiating actions in other objects
 - Coordinating activities among objects

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

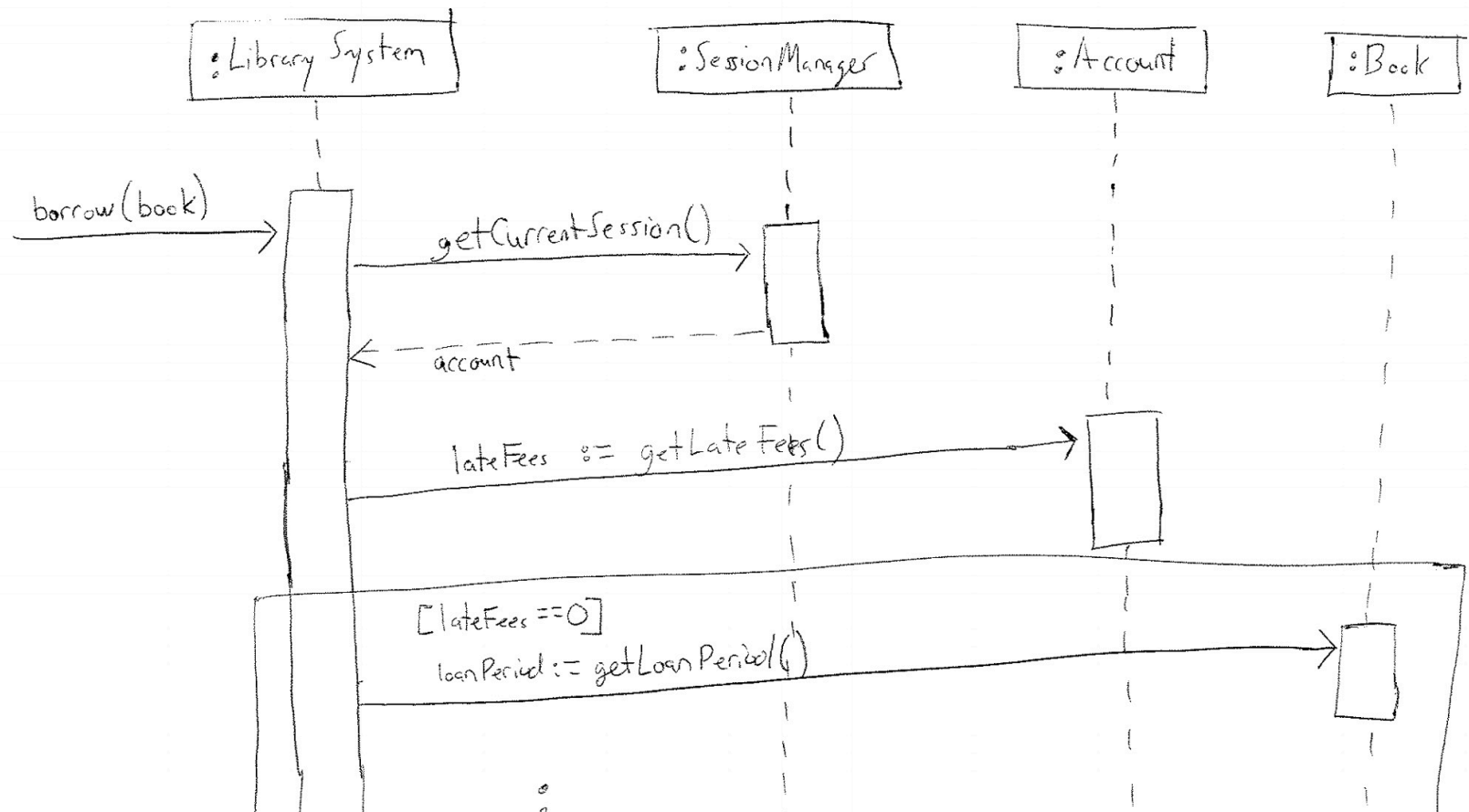
Example interaction diagram #1

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



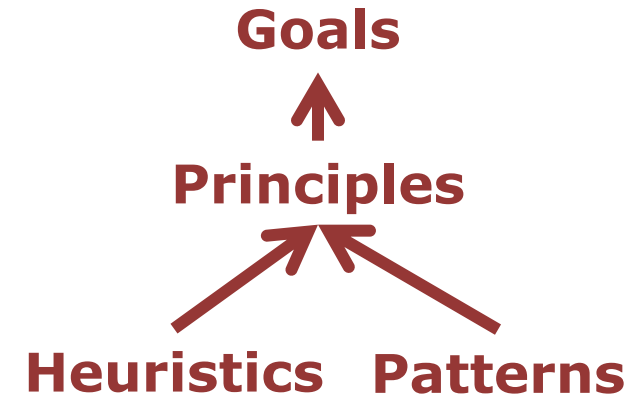
Example interaction diagram #2

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.



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?

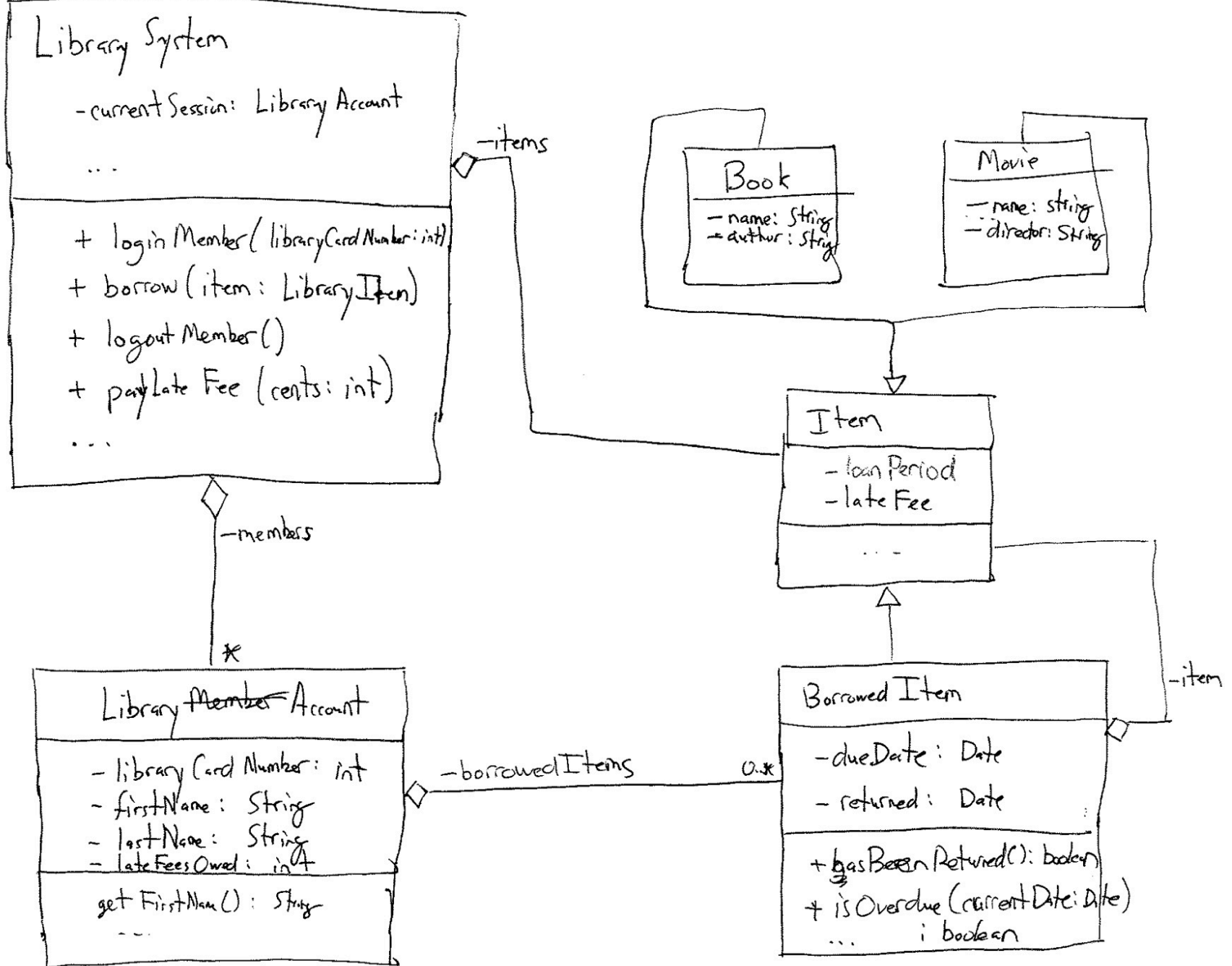
- Assign responsibility of creating an object Foo to a class that:
 - Has the data necessary for initializing instances of Foo
 - Contains, aggregates, or records instances of Foo
 - Closely uses or manipulates instances of Foo
- Design process: Extract from domain model, interaction diagrams
 - Key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



Create an object model for your sudoku 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...