Principles of Software Construction: Objects, Design, and Concurrency

Designing (sub-) systems

A formal design process

Charlie Garrod **Michael Hilton**

School of Computer Science

institute for SOFTWARE RESEARCH

Administrivia

- Optional reading for Today:
  - UML and Patterns Chapter 17
  - Effective Java items 39, 43, and 57
- Homework 3 due Thursday at 11:59 p.m.
- Midterm exam next Thursday (September 28th)
  - Exam review session: Hamerschlag Hall B103 **Wed 7-9pm**
  - Practice Exam coming soon

Participation opportunity

- Write down your “muddiest point”
  - What concept or idea do you feel you understand the least, or need more practice with?

Today

- Design goals and design principles

Metrics of software quality

Source: Braude, Bernstein, Software Engineering. Wiley 2011

- Sufficiency / functional correctness
  - Fails to implement the specifications … Satisfies all of the specifications
- Robustness
  - Will crash on any anomalous event … Recovers from all anomalous events
- Flexibility
  - Must be replaced entirely if spec changes … Easily adaptable to changes
- Reusability
  - Cannot be used in another application … Usable without modification
- Efficiency
  - Fails to satisfy speed or storage requirement … satisfies requirements
- Scalability
  - Cannot be used as the basis of a larger version … is an outstanding basis…
- Security
  - Security not accounted for at all … No manner of breaching security is known

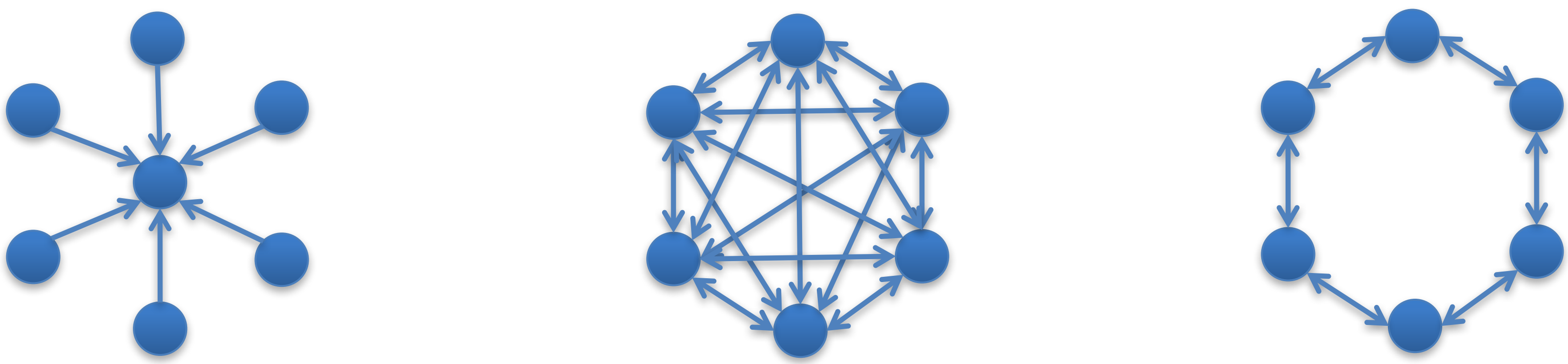
Design challenges/goals

Design principles

- 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

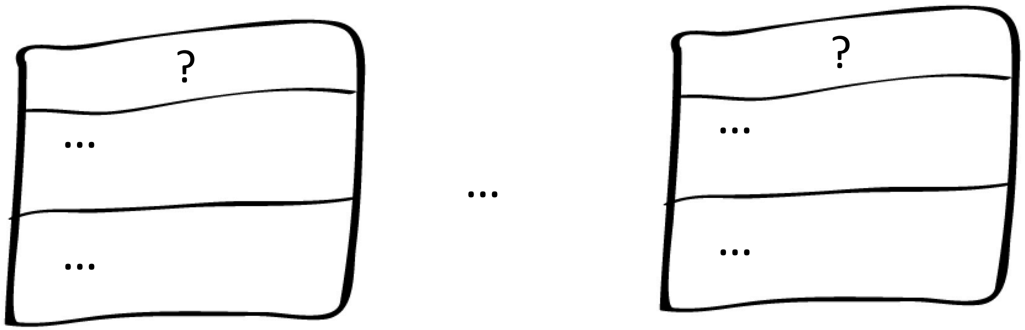
- "Only talk to your immediate friends"

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

Representational gap

- Real-world concepts:

- Software concepts:

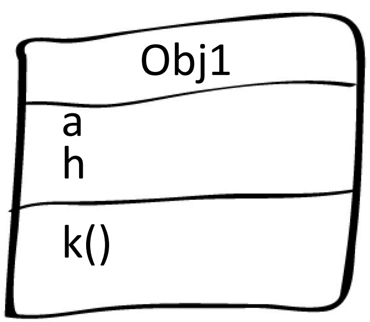


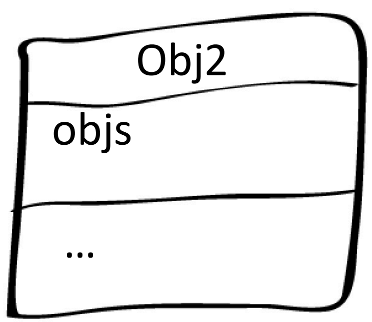
Representational gap

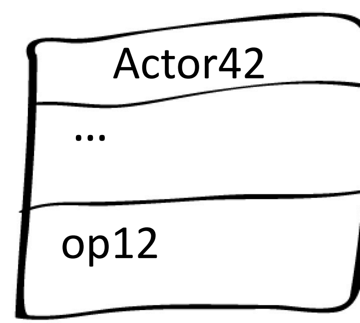
- Real-world concepts:



- Software concepts:



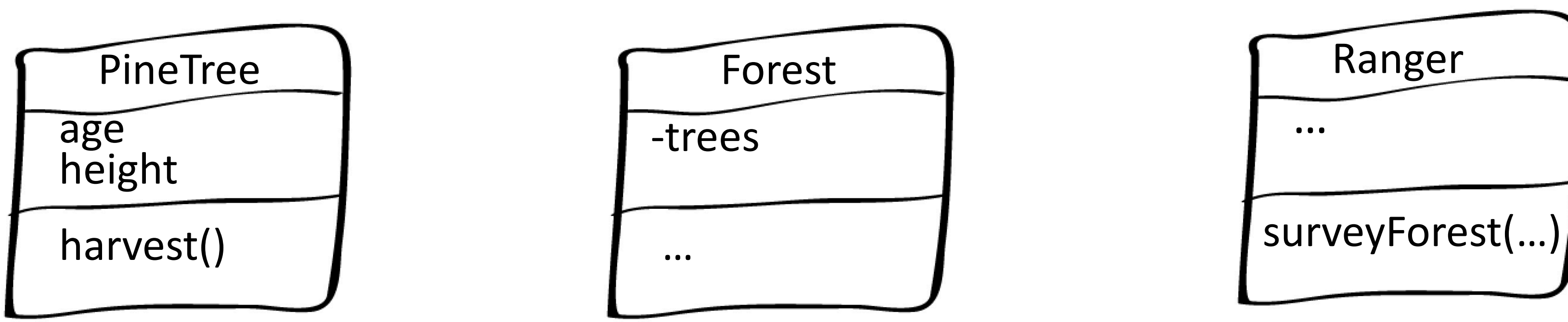




Representational gap

- Real-world concepts:

- Software concepts:

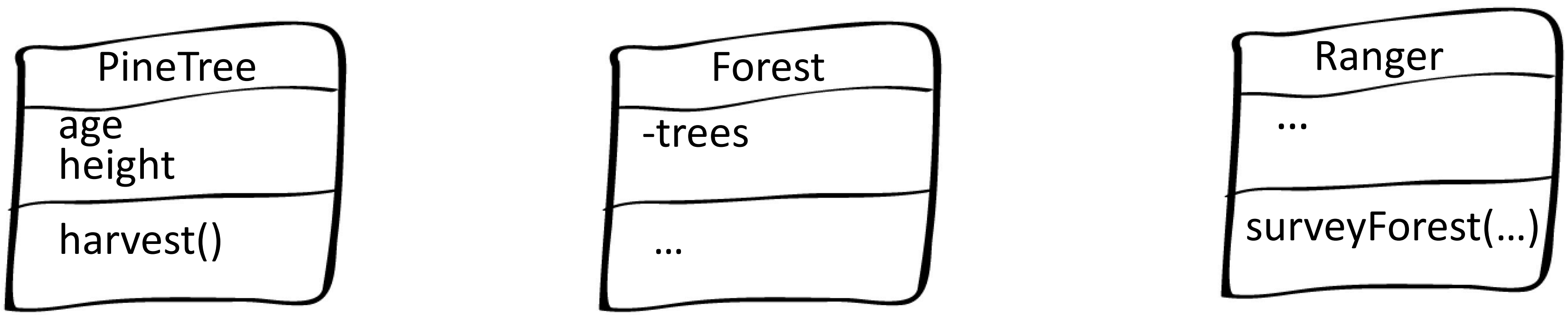


Benefits of low representational gap

- 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 (SRP)
- 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

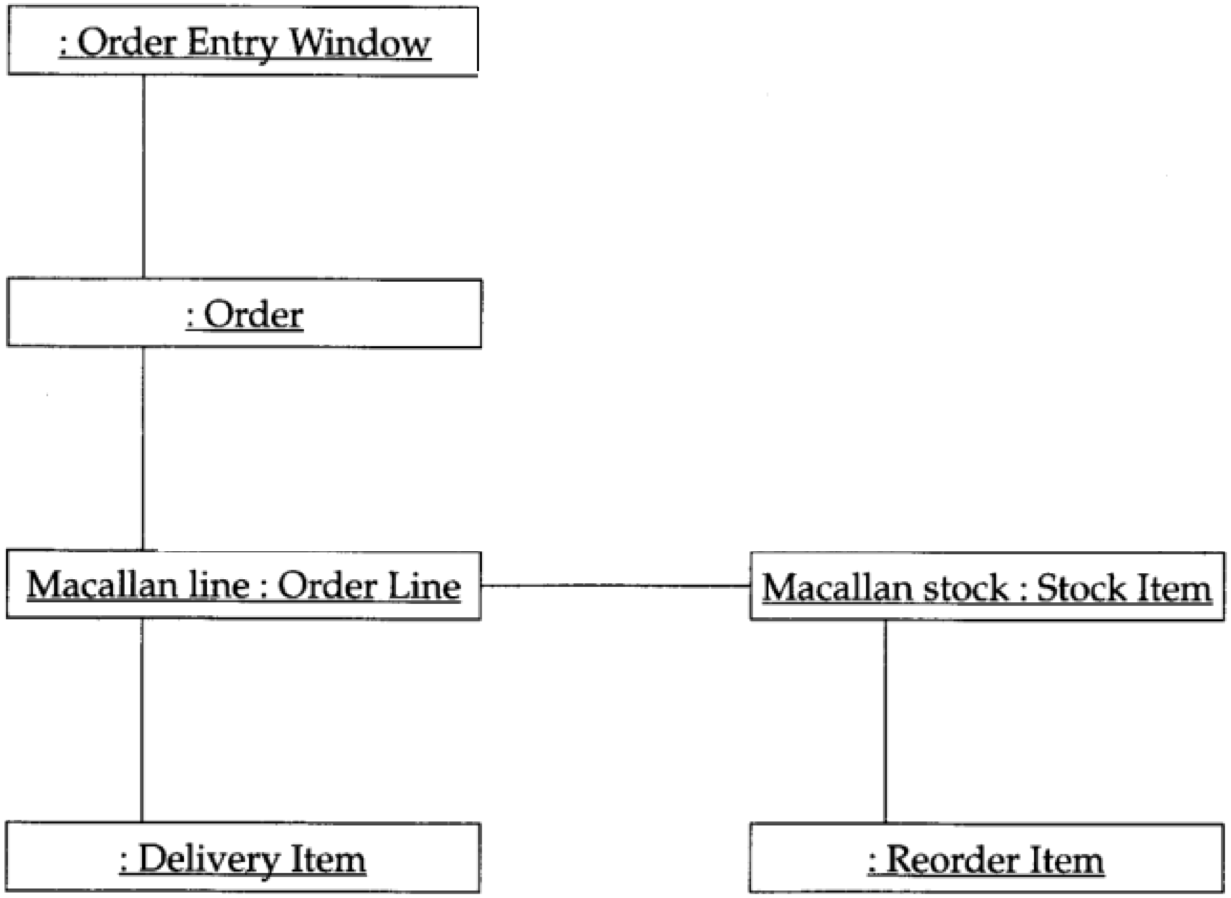
Today: Tools, goals, and understanding the problem space

- Visualizing dynamic behavior with interaction diagrams
- Design goals and design principles
- Understanding a design problem: Object oriented analysis

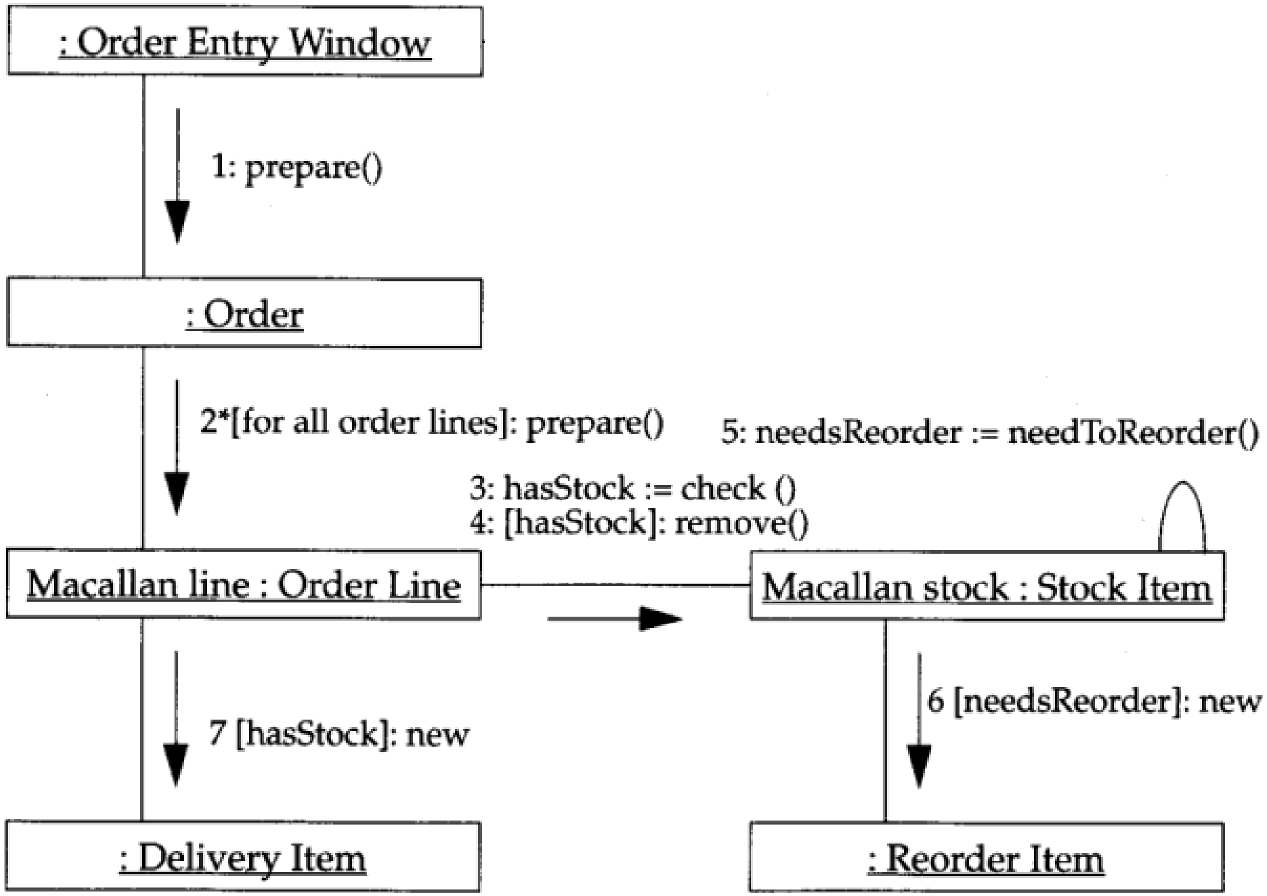
Visualizing dynamic behavior: Interaction diagrams

- 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

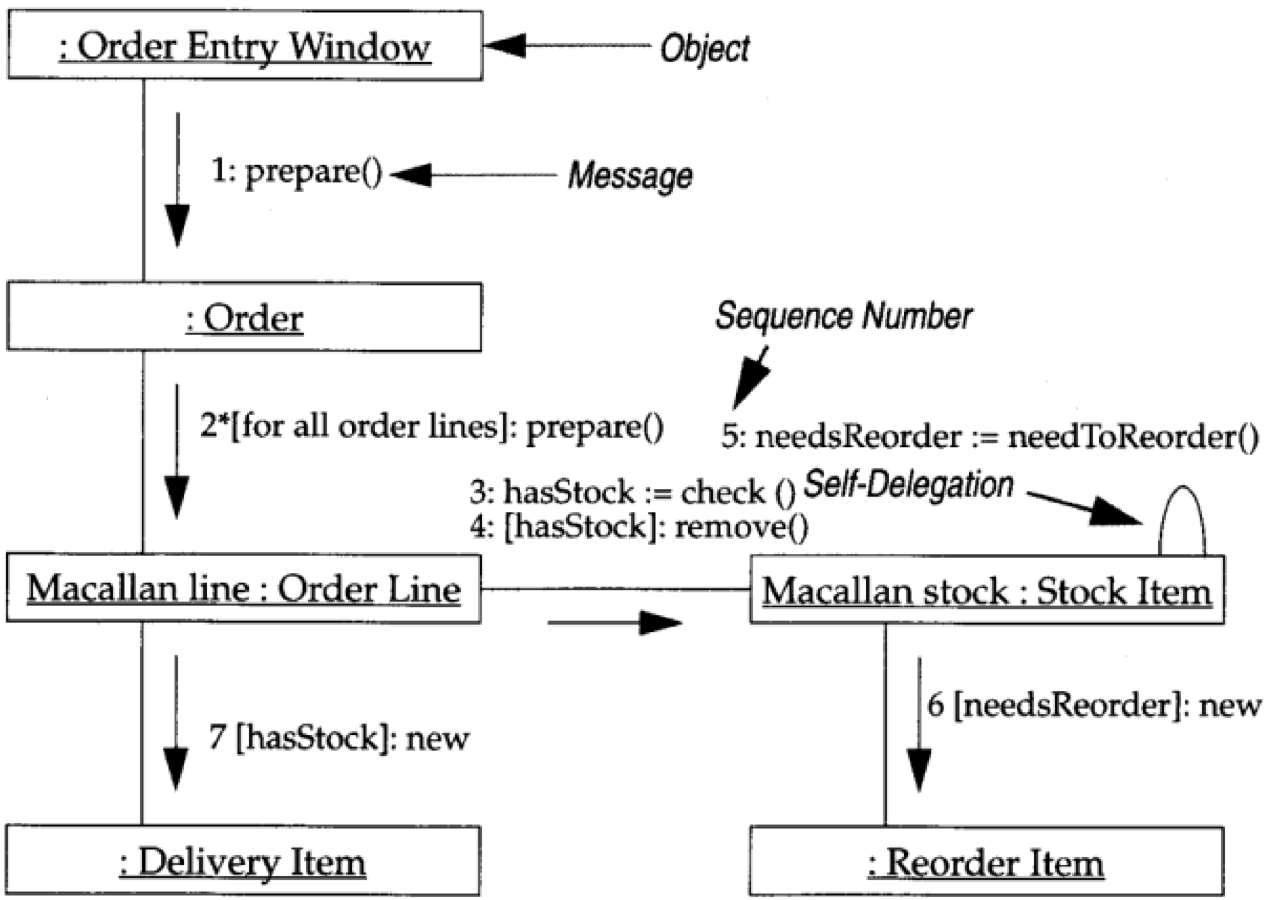
Creating a communication diagram



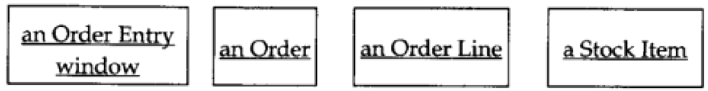
An example communication diagram



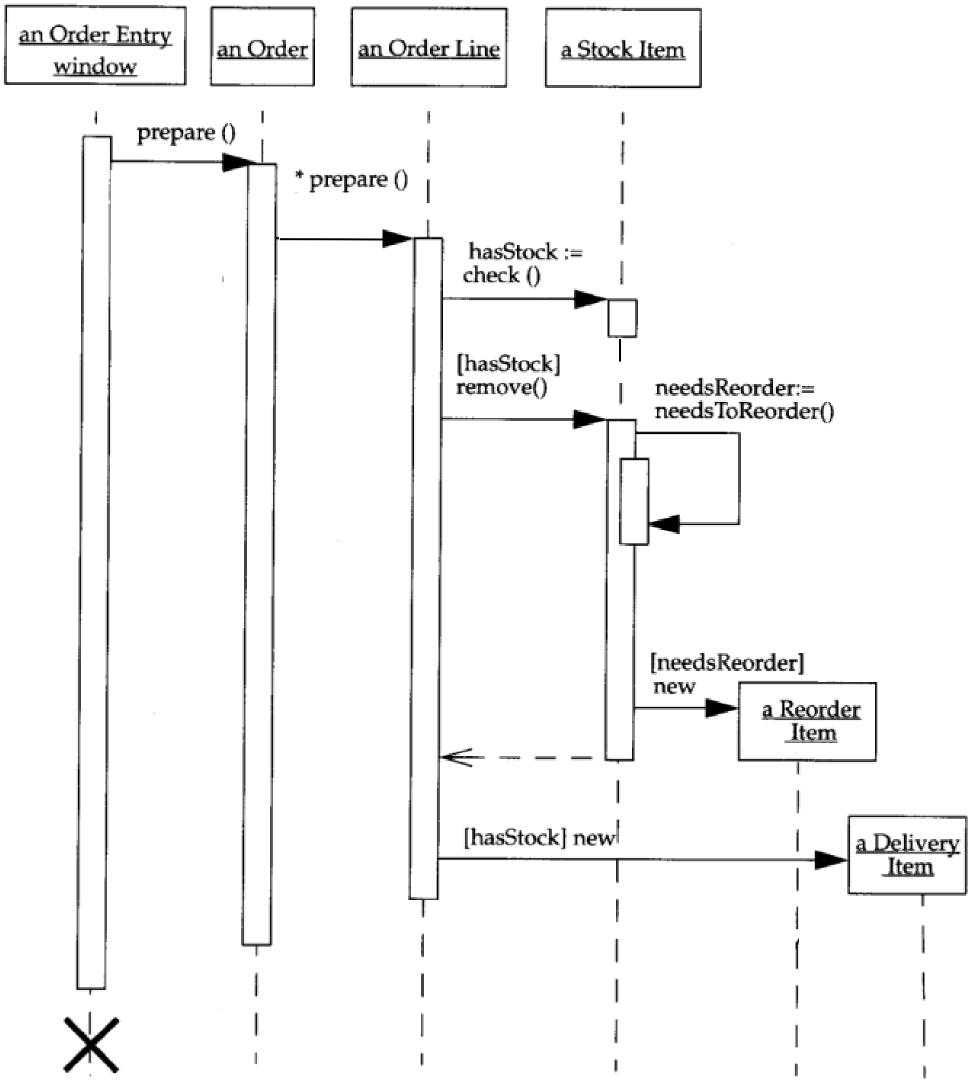
(Communication diagram with notation annotations)



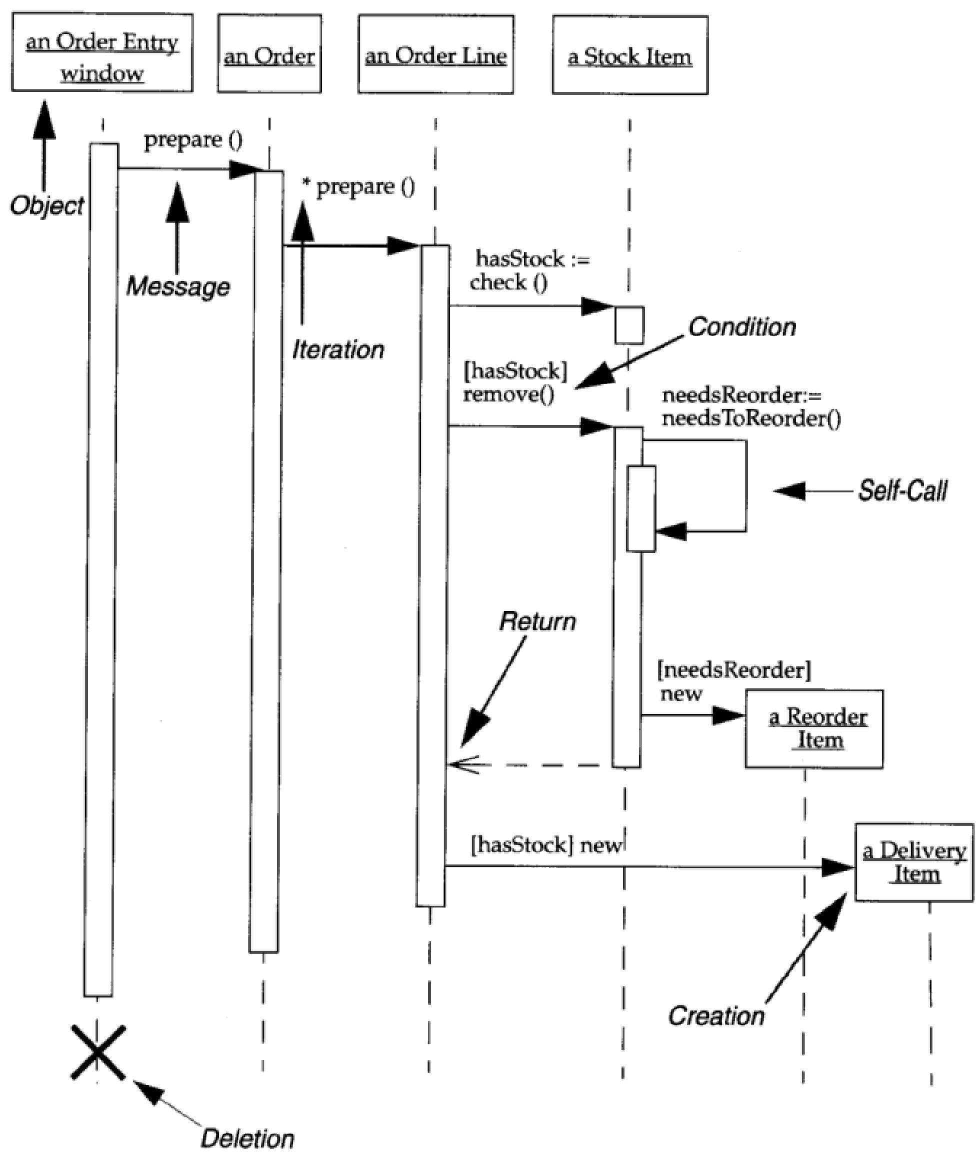
Constructing a sequence diagram



An example sequence diagram



(Sequence diagram with notation annotations)

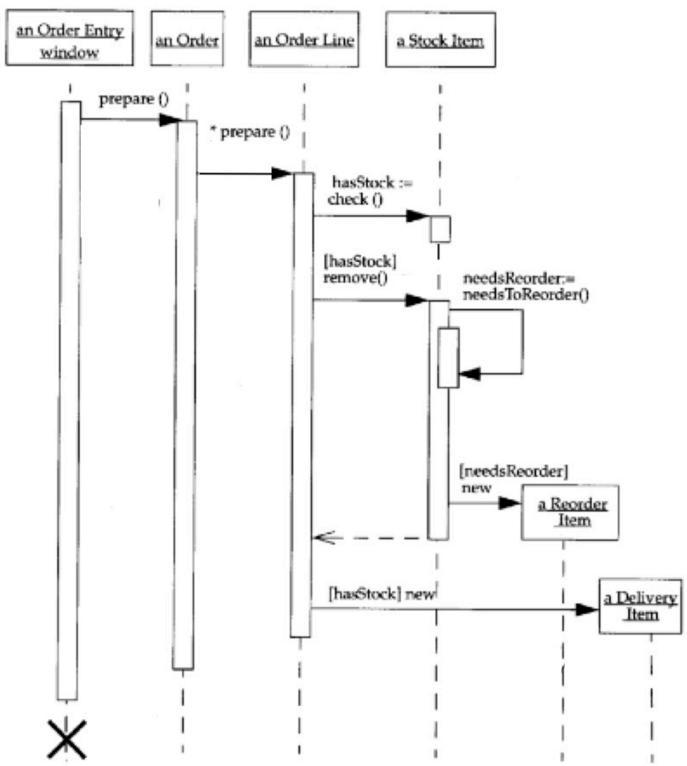


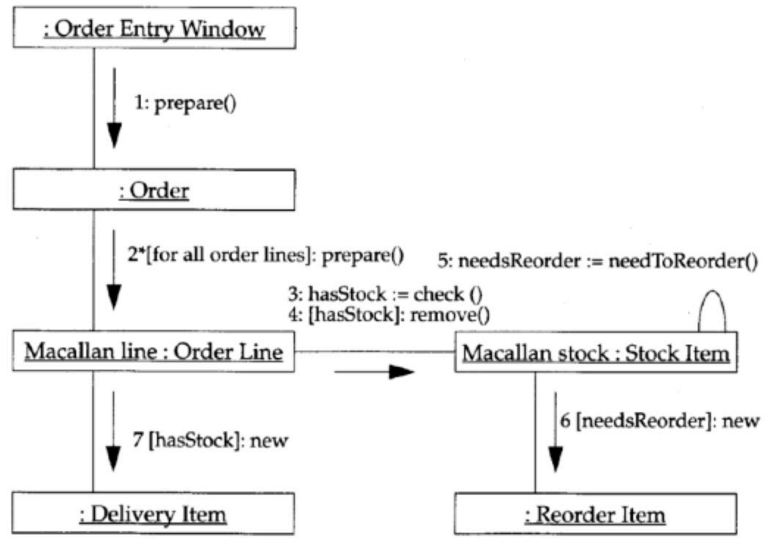
Draw a sequence diagram for a call to `LoggingList.add`:

```
public class LoggingList<E> implements List<E> {
  private final List<E> list;
  public LoggingList<E>(List<E> list) { this.list = list; }
  public boolean add(E e) {
      System.out.println("Adding " + e);
      return list.add(e);
  }
  public E remove(int index) {
      System.out.println("Removing at " + index);
      return list.remove(index);
  }
  …
```

Sequence vs. communication diagrams

- Relative advantages and disadvantages?





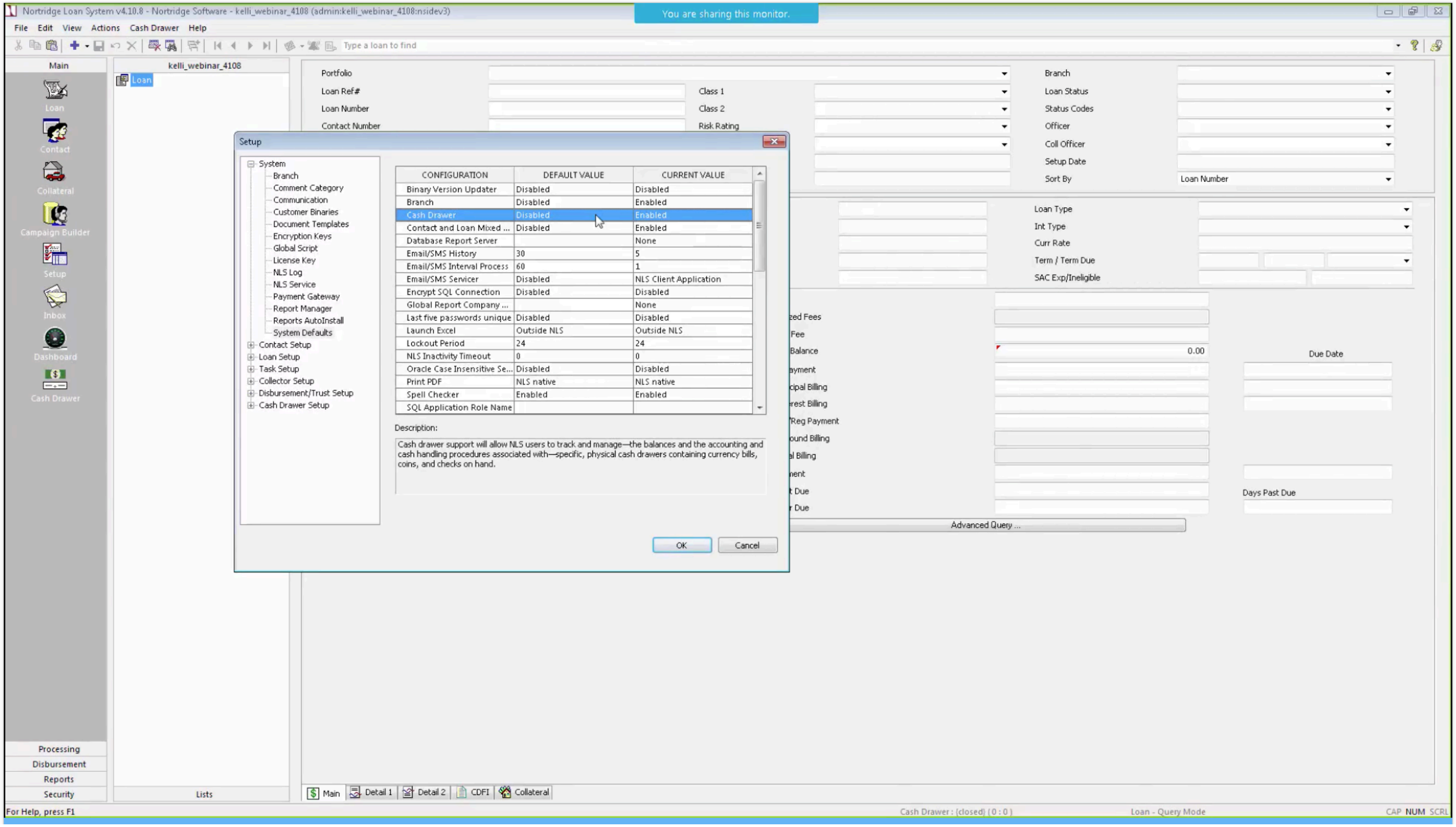
Today: Tools, goals, and understanding the problem space

- Visualizing dynamic behavior with interaction diagrams
- Design goals and design principles
- Understanding a design problem: Object oriented analysis

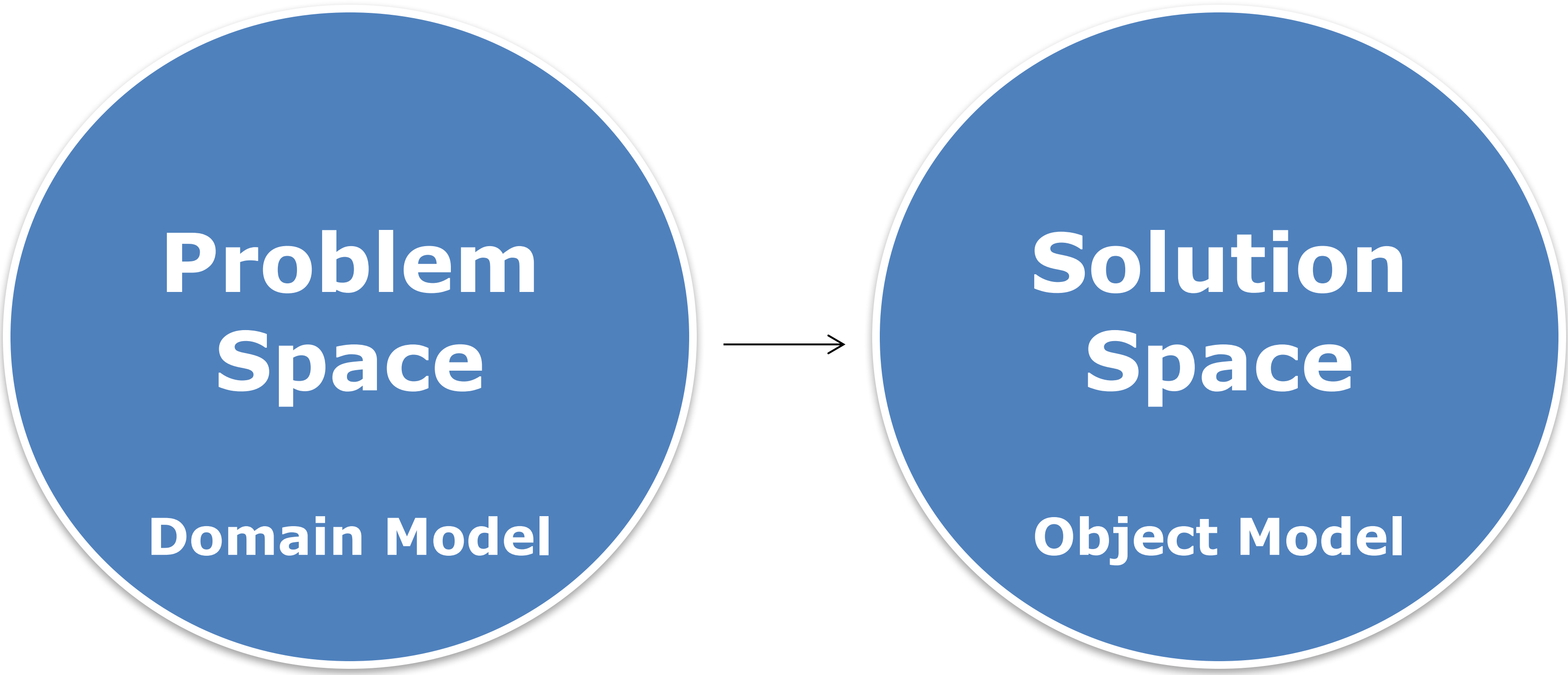
Tactical Data Radios

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, …

15-313

15-214

…

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

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 re
is borrowed
member mu
member's li

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.

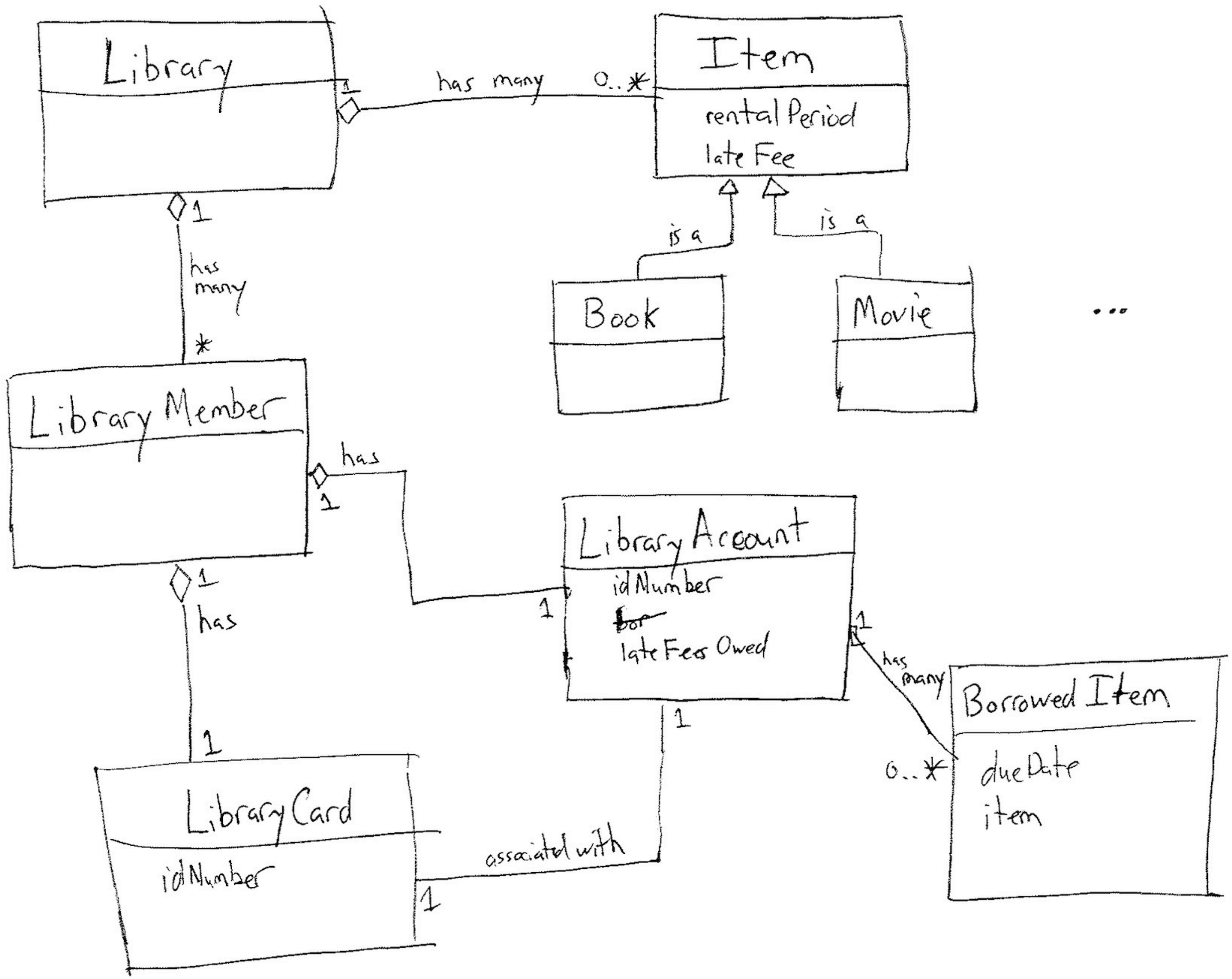
A member's library account records which items the member has borrowed and the due date for each borrowed item. Each type of item has a default rental period, which determines the item's due date when the item is borrowed. If a member returns an item after the item's due date, the member owes a late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

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.