Principles of Software Construction: Objects, Design, and Concurrency

Object-oriented Analysis

Claire Le Goues   **Vincent Hellendoorn**

Carnegie Mellon University
School of Computer Science

institute for
SOFTWARE
RESEARCH

Quiz time

"Lecture 6 Quiz" on canvas. Password: "coverage"

This is a shortened version of a typical midterm question. It is still a bit involved for a quiz, so make sure you take your time to read it carefully. Here is the (buggy) code snippet with formatting for easier readability:

```typescript
function ratio(from: number, to: number): number {
    if (from != 0) {
        from = 1;
    }
    return to / from;
}
```

institute for SOFTWARE RESEARCH

Administrivia

- Homework 2 due next Monday
  - Homework 3 starts early next week – being right away
  - Santorini Demo (hopefully!)

Some Testing Hints

Code may be used in many contexts, don't make assumptions based on one client

Code always increments every value in arr.

Is this true for all users of IntArray?

```
arr = new IntArray(len);
while (...) {
 for (i=0; i<len; i++) {
  arr.add(i, 1);
 }
}
```

Some Testing Hints

Testing code with dependencies

```
@Test ...

Comparator x = myComplexImpl();

List l = loadFromFile();

l.sort(x);
```

If testing *sort*, avoid unnecessary dependencies. Simple implementations of other objects sufficient.

Learning Goals

- High-level understanding of requirements challenges

- Understand functional requirements

- Use basic UML notation to communicate designs

- Identify the key abstractions in a domain, model them as a **domain model**

- Identify the key interactions within a system, model them as **system sequence diagram**

- Discuss benefits and limitations of the 'low representational gap' design principle

User needs
(Requirements)

Miracle?

Code

isr institute for
SOFTWARE
RESEARCH

REQUIREMENTS

How the customer explained it
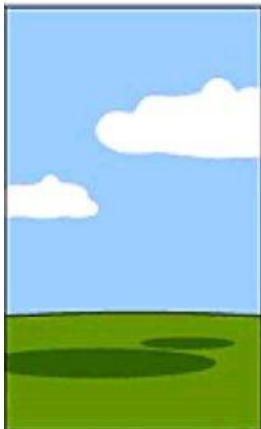
How the Project Leader understood it

How the Analyst designed it

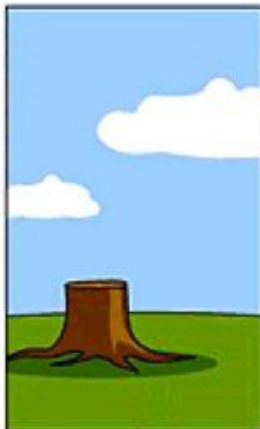How the Programmer wrote it.

How the Business Consultant sold it.

How the project was Documented

What operations installed

How the customer was billed

How it was supported

What the customer really wanted

Requirements

- What does the customer want?

- What is required, desired, not necessary? Legal, policy constraints?

- Customers often do not know what they really want; vague, biased by what they see; change their mind; get new ideas…

- Difficult to define requirements precisely

- (Are we building the right thing? Not: Are we building the thing right?)

Human and social issues beyond our scope (see 17-313)

isr institute for SOFTWARE RESEARCH

Requirements

- What are the requirements of [the system]?

- What are the customers requirements?

- Custo... ...sed by what they see; change their mind; get new ideas…

- Diffic...

- (Are ...ght?)

**Assumption in this course:
Somebody has gathered most
requirements (mostly text).**

**Challenges:
How do we start implementing them?
How do we cope with changes?**

**Human and social issues
beyond our scope (see 17-313)**

Input to the analysis process: Requirements and use cases

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 must pay a late fee, an amount of money recorded in the member's library account.

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.
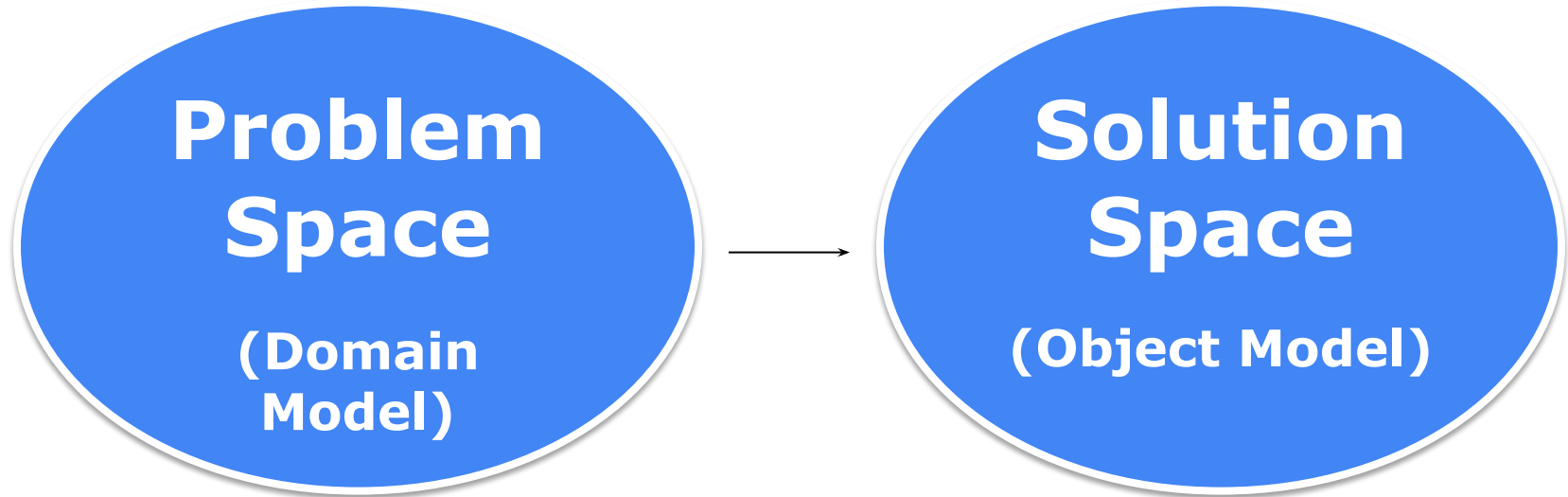
ISR institute for SOFTWARE RESEARCH

Input to the analysis process: Requirements and use cases

Time to start coding?

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 must pay a late fee, an amount of money recorded in the member's library account.

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.

Problem Space (Domain Model)

Solution Space (Object Model)

- Real-world things
- 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

An object-oriented design process

Model / diagram the problem, define concepts

- **Domain model** (a.k.a. conceptual model), **glossary**

Define system behaviors

- **System sequence diagram**
- **System behavioral contracts**

Assign object responsibilities, define interactions

- **Object interaction diagrams**

Model / diagram a potential solution

- **Object model**

OO Analysis:
Understanding
the problem

OO Design:
Defining a
solution

institute for SOFTWARE RESEARCH

DOMAIN MODELS

APPLYING UML AND PATTERNS
An Introduction to Object-Oriented Analysis and Design and Iterative Development
THIRD EDITION

CRAIG LARMAN
Foreword by Philippe Kruchten
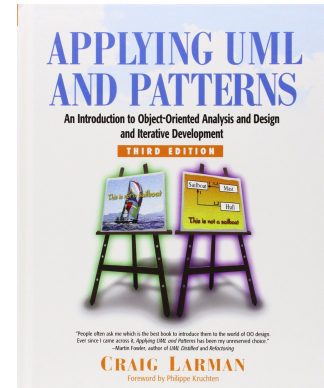
Chapter 9

Object-Oriented Analysis

- Find the <u>concepts</u> in the problem domain

  - Real-world abstractions, not necessarily software objects

- Understand the problem

  - Establish a common vocabulary

  - Common documentation, big picture

  - Main purpose is communication!

- Often using UML class diagrams as (informal) notation

- Starting point for finding classes later (low representational gap)

isr institute for SOFTWARE RESEARCH

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

Concepts in our 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.

Read description carefully, look for <u>nouns</u> and verbs

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

A member's library account records which items the member has borrowed and the <u>due date</u> for each <u>borrowed item</u>. Each type of item has a <u>default rental period</u>, 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 <u>late fee</u> specific for that item, an amount of money recorded in the member's library account.

Glossary

Identify and define key concepts

Ensure shared understanding between developers and customers

Define
potentially
ambiguous
concepts

Library item: Any item that is indexed and can be borrowed from the library
Library member: Person who can borrow from a library, identified by a card with an ID number
Book

No need to
expand on
obvious
concepts

ISR institute for SOFTWARE RESEARCH

Visual notation: UML

Name of
real-world
concept
(not software class)

Properties
of concept

Library Account
accountID lateFees

Visual notation: UML

Name of
real-world
concept
(not software class)

Properties
of concept

Library Account
accountID lateFees

Book
title author

Visual notation: UML

Name of
real-world
concept
(not software class)

Properties
of concept

Library Account
accountID lateFees

borrow

Book
title author

1   *

Associations
between
concepts

Multiplicities/cardinalities
indicate "how many"
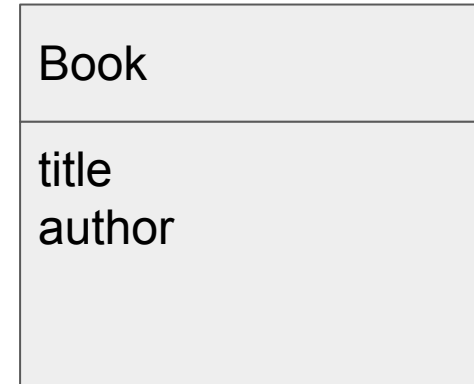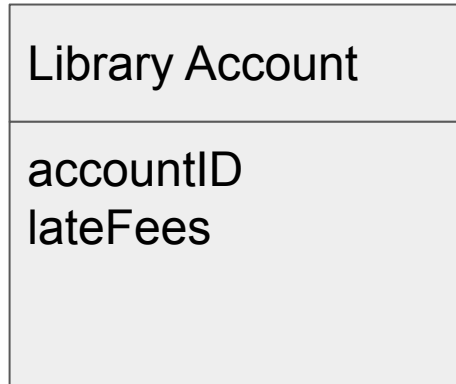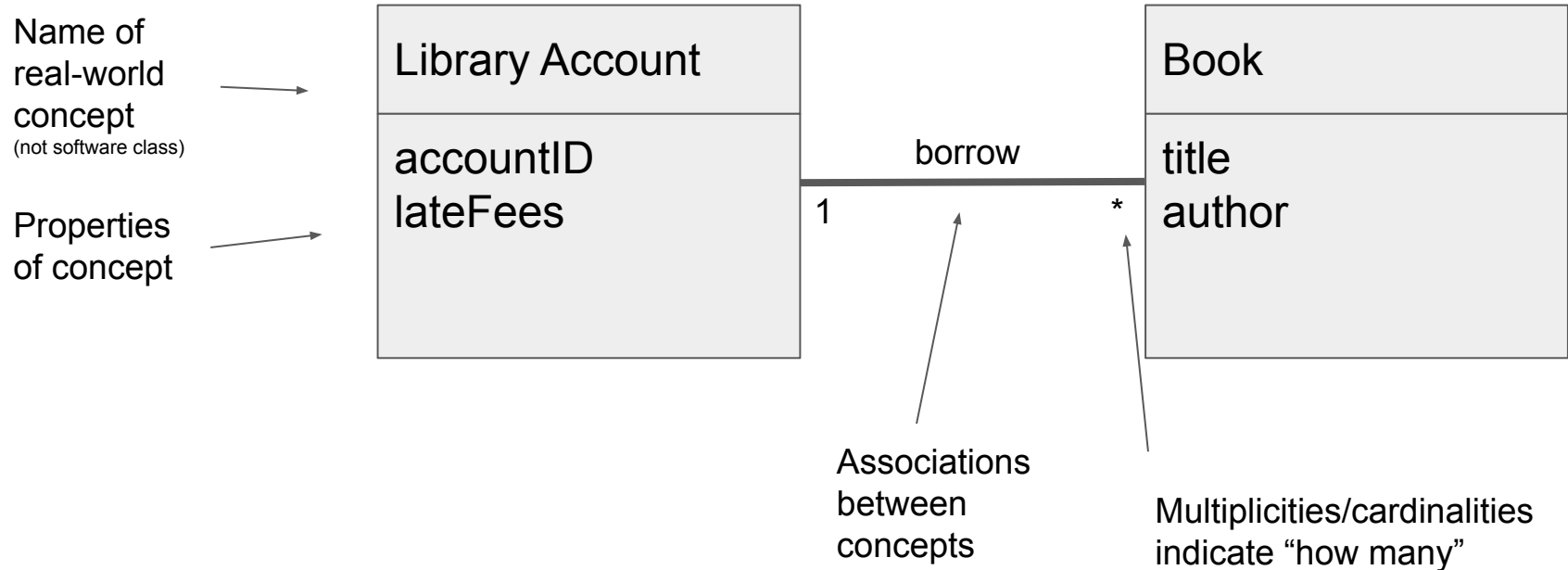
Reading associations

One **library account** can **borrow** *many* **book**s →

Library Account		Book
accountID lateFees	—— borrow —— 1 *	title author
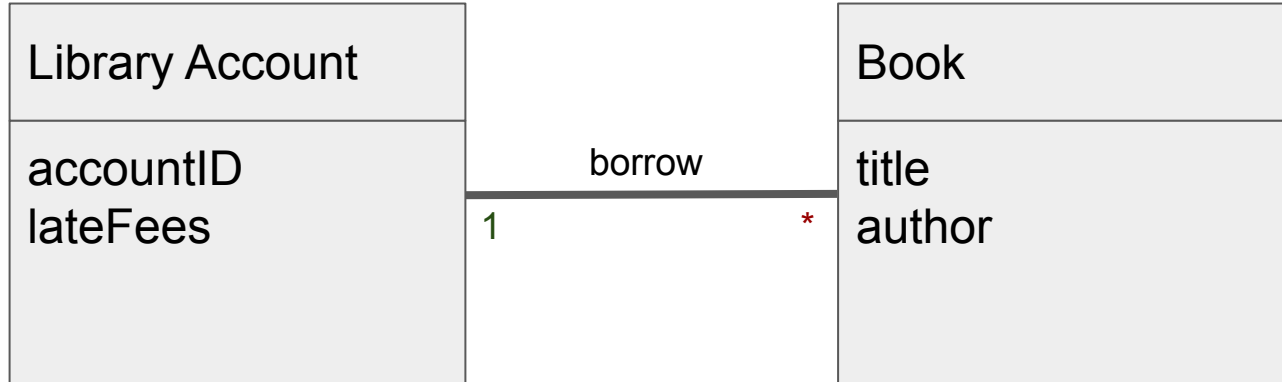
One **book** can be **borrowed** by *one* **library account**

isr institute for SOFTWARE RESEARCH

Reading associations

Specialization

Library Item

id

Book

title
author

Video

title
director

More specialized version of general concept.
Every video **is a** library item

institute for SOFTWARE RESEARCH

Concepts vs. Attributes

Library Account
accountID lateFees borrowedBooks

vs.

Library Account
accountID lateFees

borrow

1   *

Book
title author
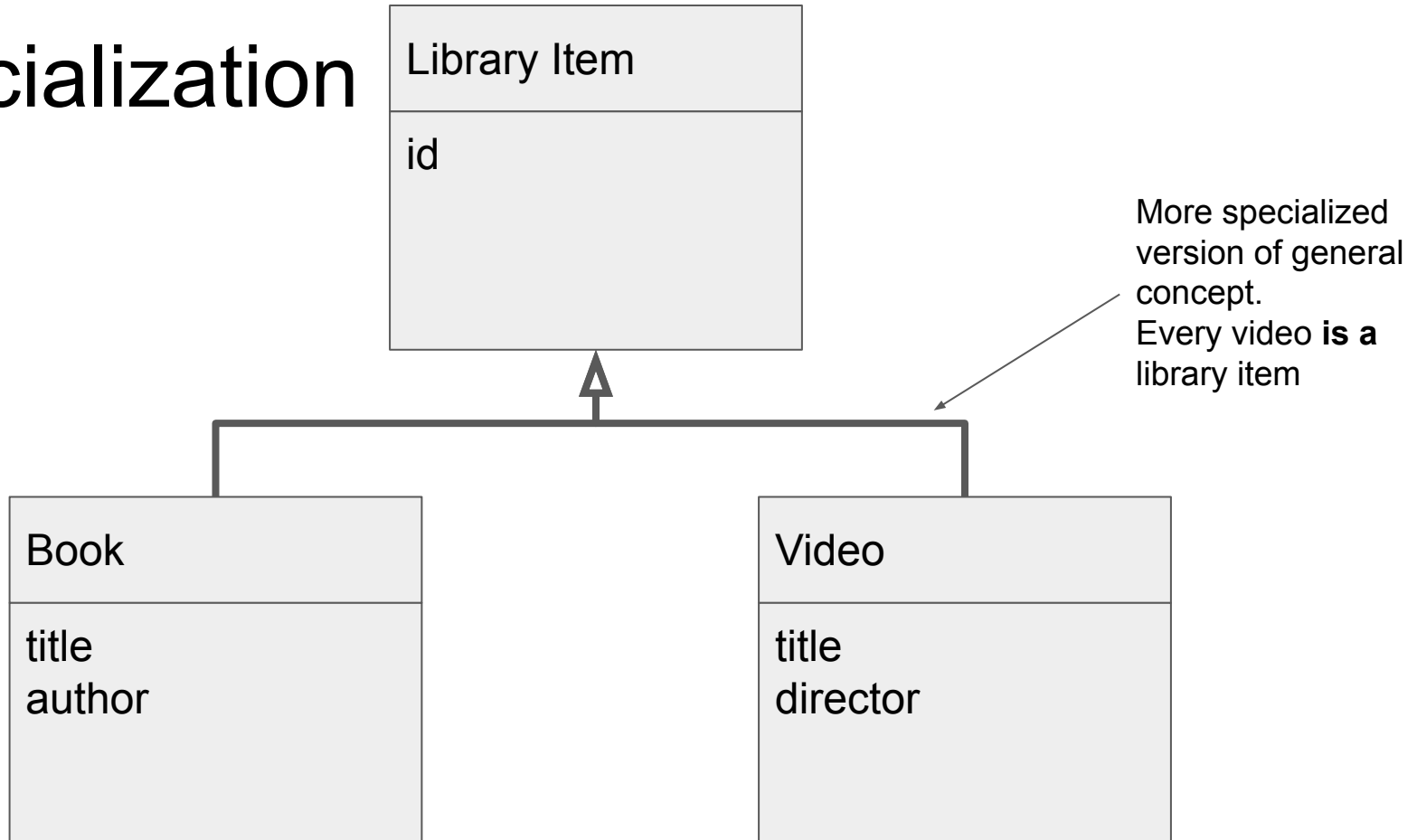
- "If we do not think of some conceptual class X as text or a number in the real world, it's probably a concept, not an attribute"

- Avoid type annotations

isr institute for SOFTWARE RESEARCH

Composition & Aggregation

Often, associations form a "has a" relationship

- Compositions: the parts are irrelevant* without the whole
- Aggregation: the parts meaningfully exist on their own
  - E.g., a library still exists without members
- In this class: no need to be very zealous about this

Figure source: https://phpmusings.blog/2021/03/04/uml-in-action-part-3-aggregation-composition/
* From the domain model's perspective: naturally, in some domains, organs are relevant on their own
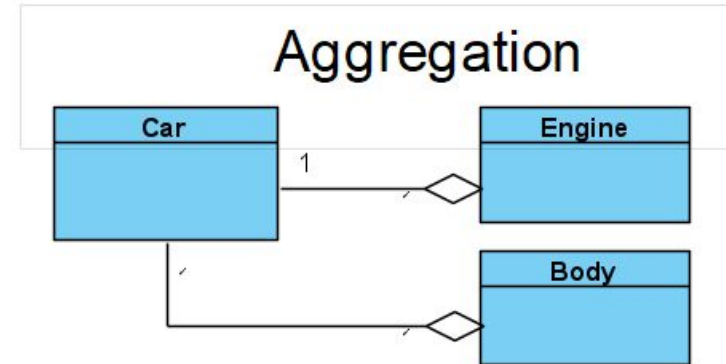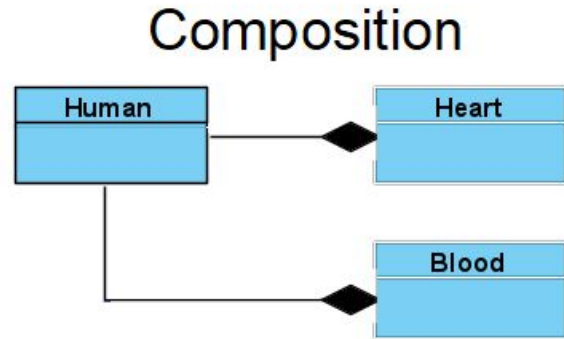
One possible domain model for the library system

...
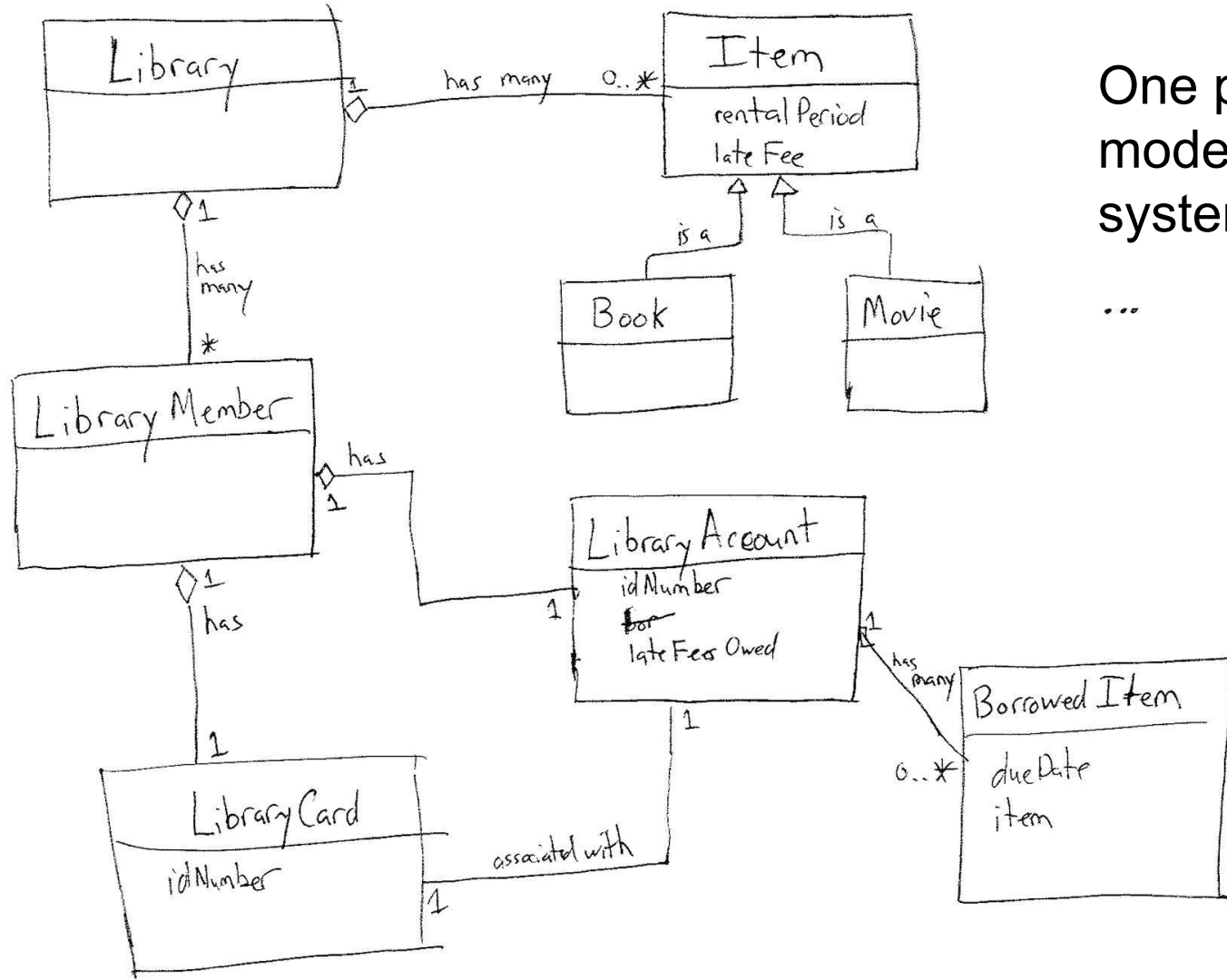
Notes on the library domain model

- Level of abstraction:
  - All concepts are accessible to a non-programmer
  - UML notation somewhat informal; relationships often described with words
  - Real-world "is-a" relationships are appropriate for a domain model
  - Real-word abstractions are appropriate for a domain model
- Design choices:
  - Aggregate types are usually modeled as separate concepts
  - Basic attributes (numbers, strings) are usually modeled as attributes
- 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.

institute for
SOFTWARE
RESEARCH

Why domain modeling?

- Understand the domain
  - Details matter! Are books different from videos for the system?
- Ensure completeness
  - Late fees considered?
- Agree on a common set of terms
  - Library item vs collection entry vs book
- Prepare to design
  - Domain concepts are good candidates for OO classes (-> low representational gap)

institute for SOFTWARE RESEARCH

Hints for Object-Oriented Analysis

- Use the domain model to agree on a vocabulary
  - For communication among developers, testers, clients, domain experts, …
- Focus on concepts, not software classes, not data
  - Ideas, things, objects
  - Give it a name, define it and give examples (symbol, intension, extension)
  - Add glossary
  - Some might be implemented as classes, other might not
- There are many choices, the model will never be perfectly correct
  - Start with a partial model, model what's needed, extend with additional information later
  - Communicate changes clearly
  - Otherwise danger of "analysis paralysis"

institute for SOFTWARE RESEARCH

User needs (Requirements) → *Miracle?* → Code

Miracle (work in progress):
1. Domain model
2. ???
3. Code!

isr institute for SOFTWARE RESEARCH

META NOTE

These slides were re-arranged since teaching this class in September 2022. If you are watching the recording, jump to slide 48 now and once you get to 56, return here.

Back to: requirements and **<u>use cases</u>**

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 must pay a late fee, an amount of money recorded in the member's library account.

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.

What about this second part?

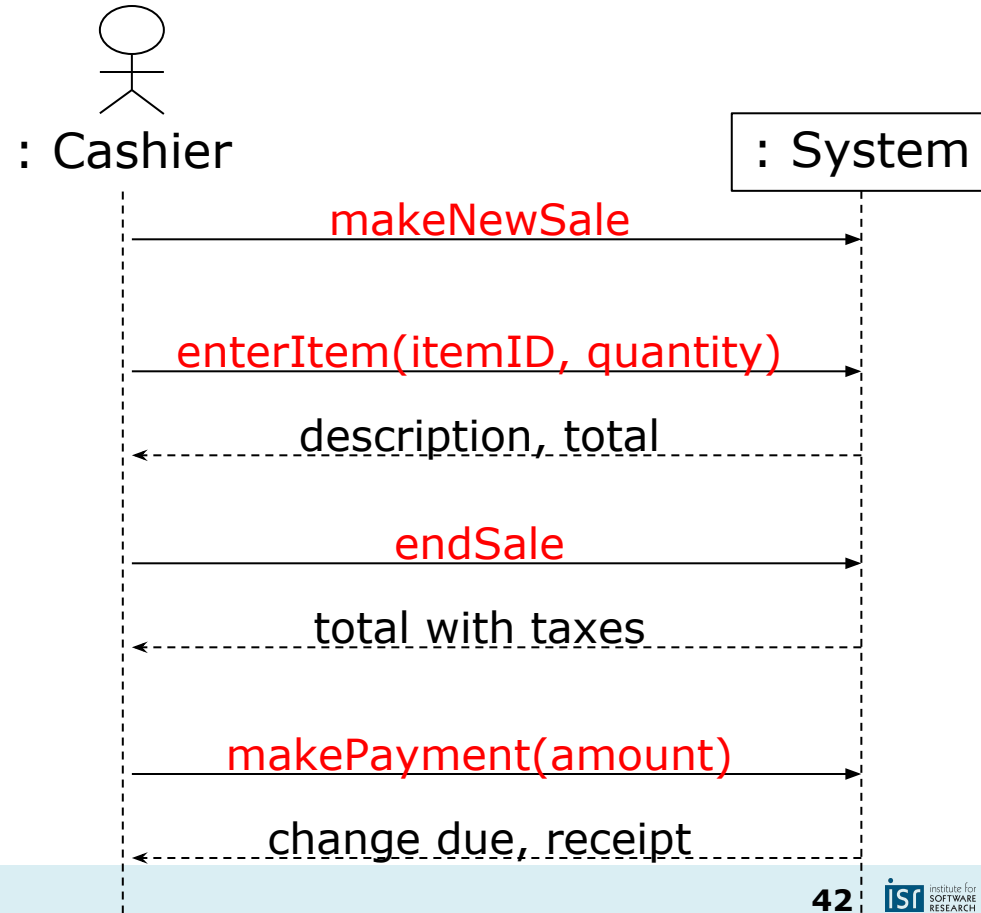# System Sequence Diagram
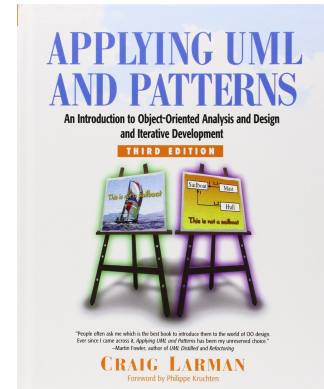
Chapter 10

Understanding system behavior

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**

- System-level components only: e.g., A user and the overall system

One example 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 example 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.
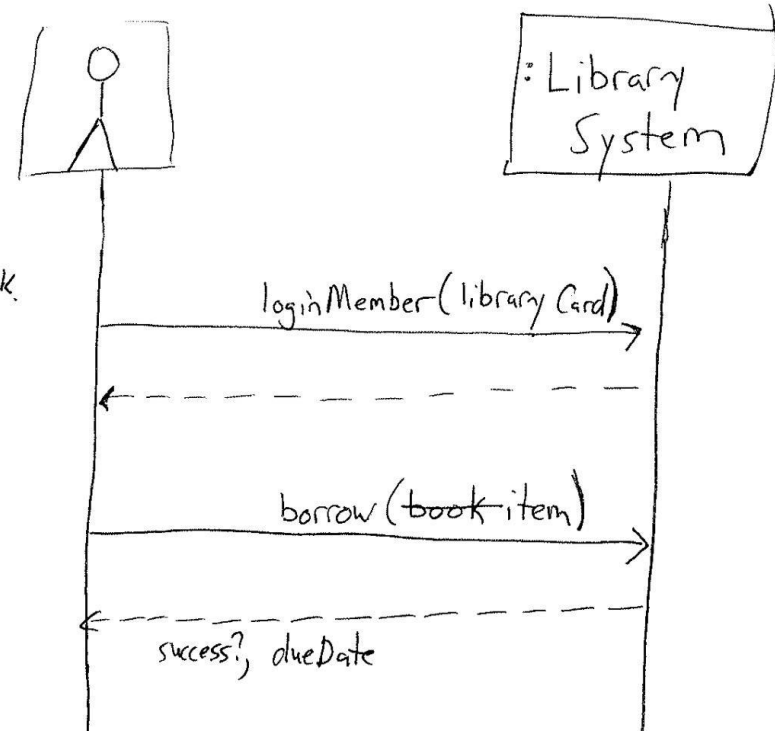
UML Sequence Diagram Notation

User

System

login(card)

borrow(book)

success?, due date

Actors in this use case (systems and real-world objects/people)

Time proceeds from top to bottom

Messages and responses for interactions, text describes what happens conceptually
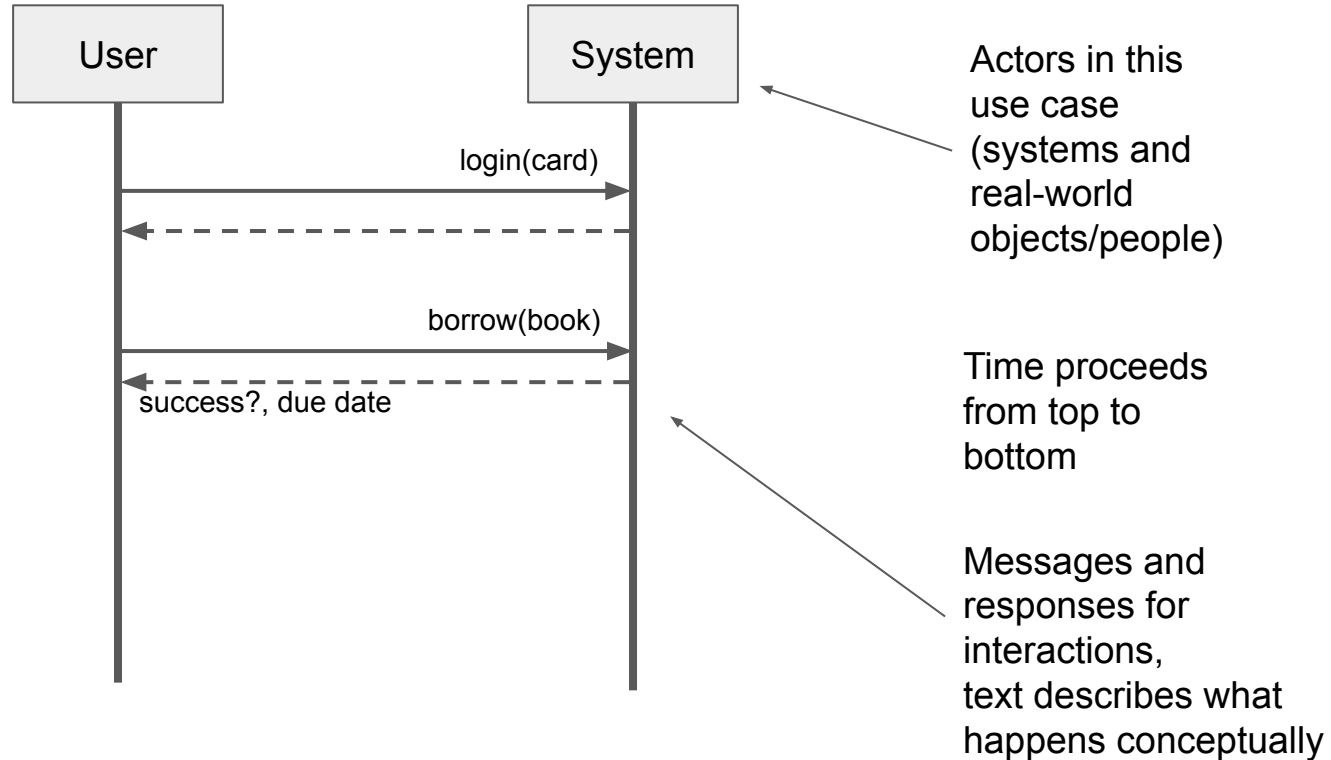
Outlook: System Sequence Diagrams to Tests

```
s = new System();

a = s.makeNewSale();

t = a.enterItem(…);

assert(50.30, t);

tt = a.endSale();

assert(52.32, tt);

…
```

: Cashier   : System

makeNewSale →

enterItem(itemID, quantity) →

← description, total

endSale →

← total with taxes

makePayment(amount) →

← change due, receipt

institute for SOFTWARE RESEARCH

Behavioral Contracts
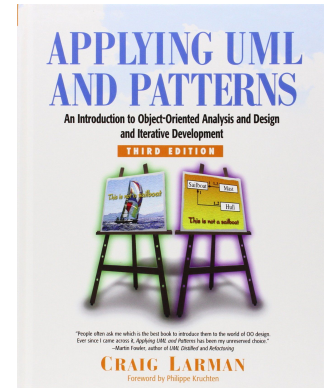
Chapter 11

Formalize system at boundary

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
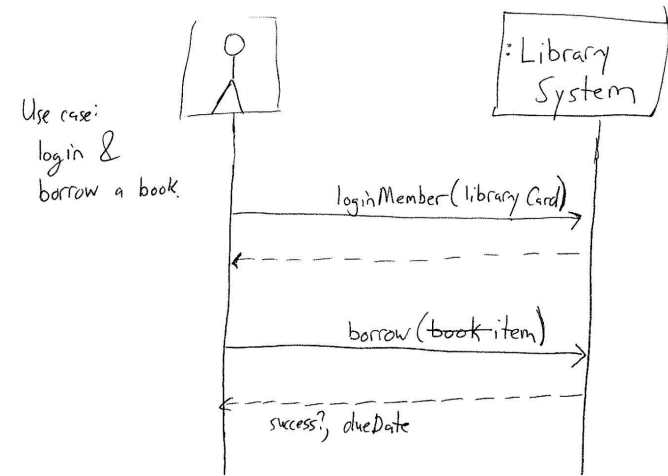
System behavioral contract example

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.

isr institute for SOFTWARE RESEARCH

Recommended Reading: Applying UML and Patterns

Detailed coverage of modeling steps

Explains UML notation

Many examples
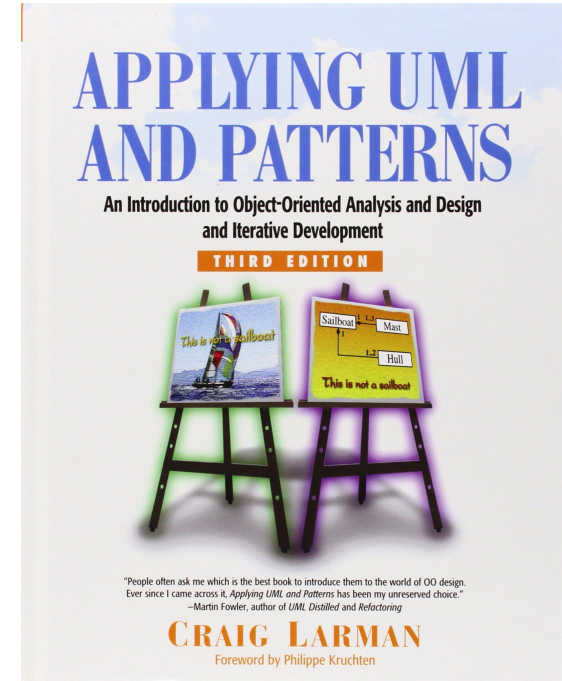
Chapter 9

Outlook: from concepts to objects

User needs
(Requirements)
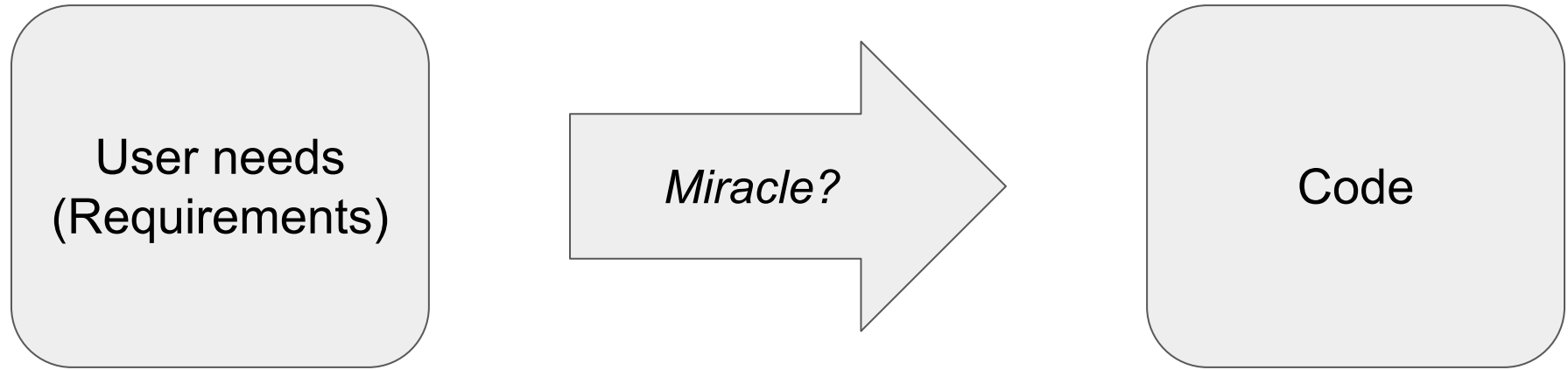
Miracle?

Code

Miracle (work in progress):
1. Domain model
2. System behavior
3. ???
4. Code!
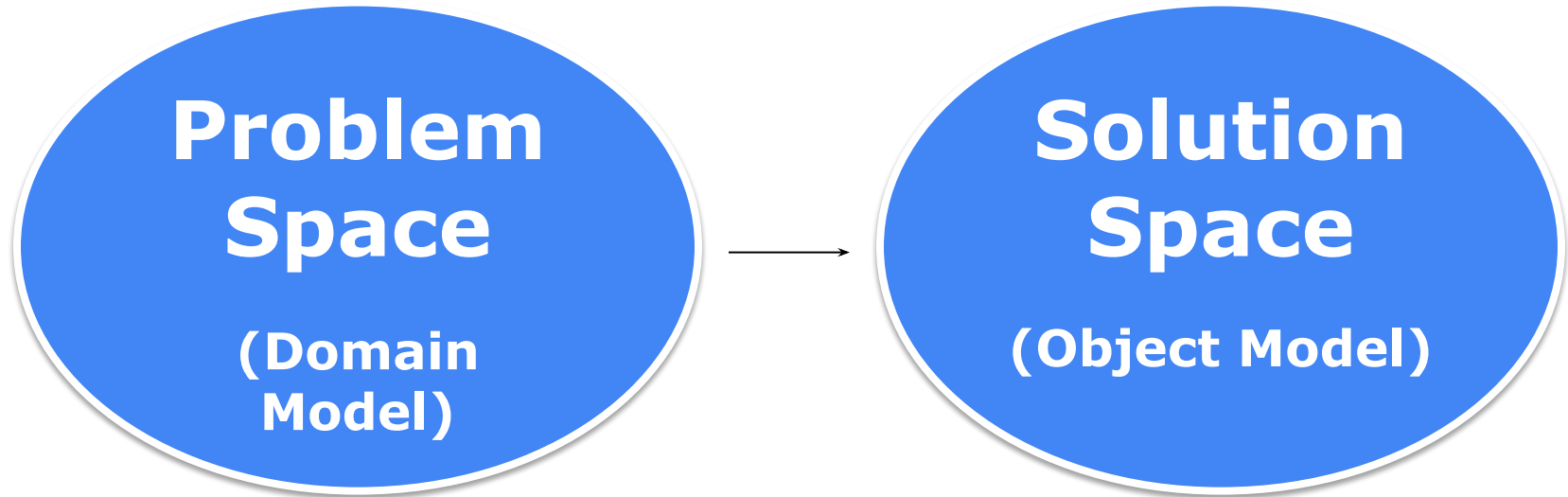
institute for
SOFTWARE
RESEARCH

Problem Space

(Domain Model)

→

Solution Space

(Object Model)

- Real-world things
- 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

Representational gap

- Real-world concepts:

- Software concepts:
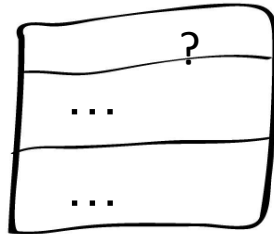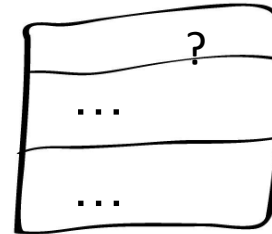
Low Representational Gap

Identified concepts provide inspiration for classes in the implementation

Classes mirroring domain concepts often intuitive to understand:

Low representational gap **principle**
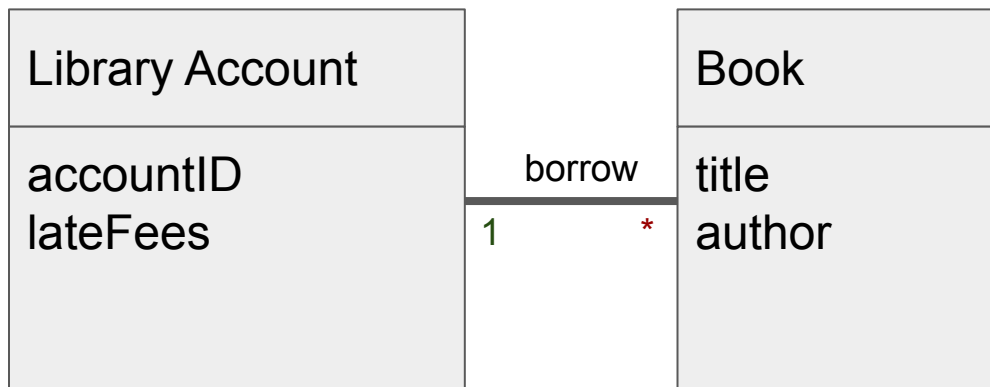

```
class Account {
    id: int;
    lateFees: int;
    borrowed: List<Book>;
    boolean borrow(Book) { … }
    void save();
}
class Book { … }
```

institute for SOFTWARE RESEARCH

Representational gap

- Real-world concepts:

- Software concepts:
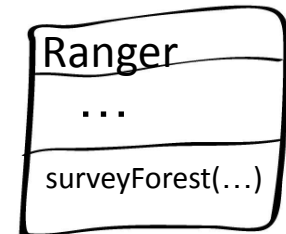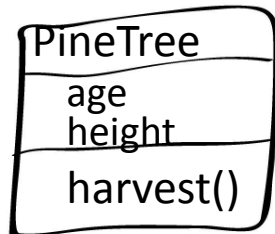
Benefits of low representational gap

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

Design Goals, Principles, and Patterns

Get familiar with design terminology – we'll see a lot of these

- Design <u>Goals</u>
  - Design for understanding, change
- Design <u>Principles</u>
  - Low representational gap
- Design <u>Heuristics</u>
  - Match concepts to classes

Goals

↑

Principles

Heuristics **Patterns**
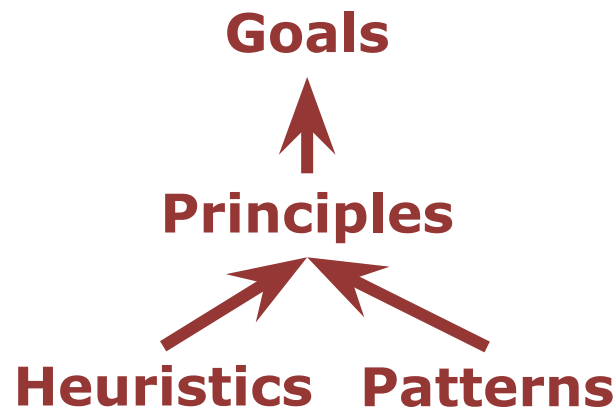
Distinguishing domain vs. implementation concepts

- 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

Towards Implementation

- Next week: how to move from <u>domain model</u> to <u>object model</u>
  - Some domain concepts become objects
  - Think about interface (methods), fields

Library Account
accountID lateFees

borrow

1   *

Book
title author

LibraryAccount
id: int lateFees: int
borrow(Book): bool returnItem(Book) payFees(int)

Outlook: Build a domain model for HW 3

Take-Home Messages

- To design a solution, problem needs to be understood
- Know your tools to build domain-level representations
  - Domain models – understand domain and vocabulary
  - System sequence diagrams + behavioral contracts – understand interactions with environment
- Be fast and (sometimes) loose
  - Elide obvious(?) details, iterate, iterate, iterate, …
- Domain classes often turn into Java classes
  - *Low representational gap* principle, design for understanding and change
  - Some domain classes don't need to be modeled in code; other concepts only live at the code level
- Get feedback from domain experts
  - Use only domain-level concepts

institute for
SOFTWARE
RESEARCH