

Principles of Software Construction

Design (sub-)systems

A formal design process, part 2

Josh Bloch

Charlie Garrod

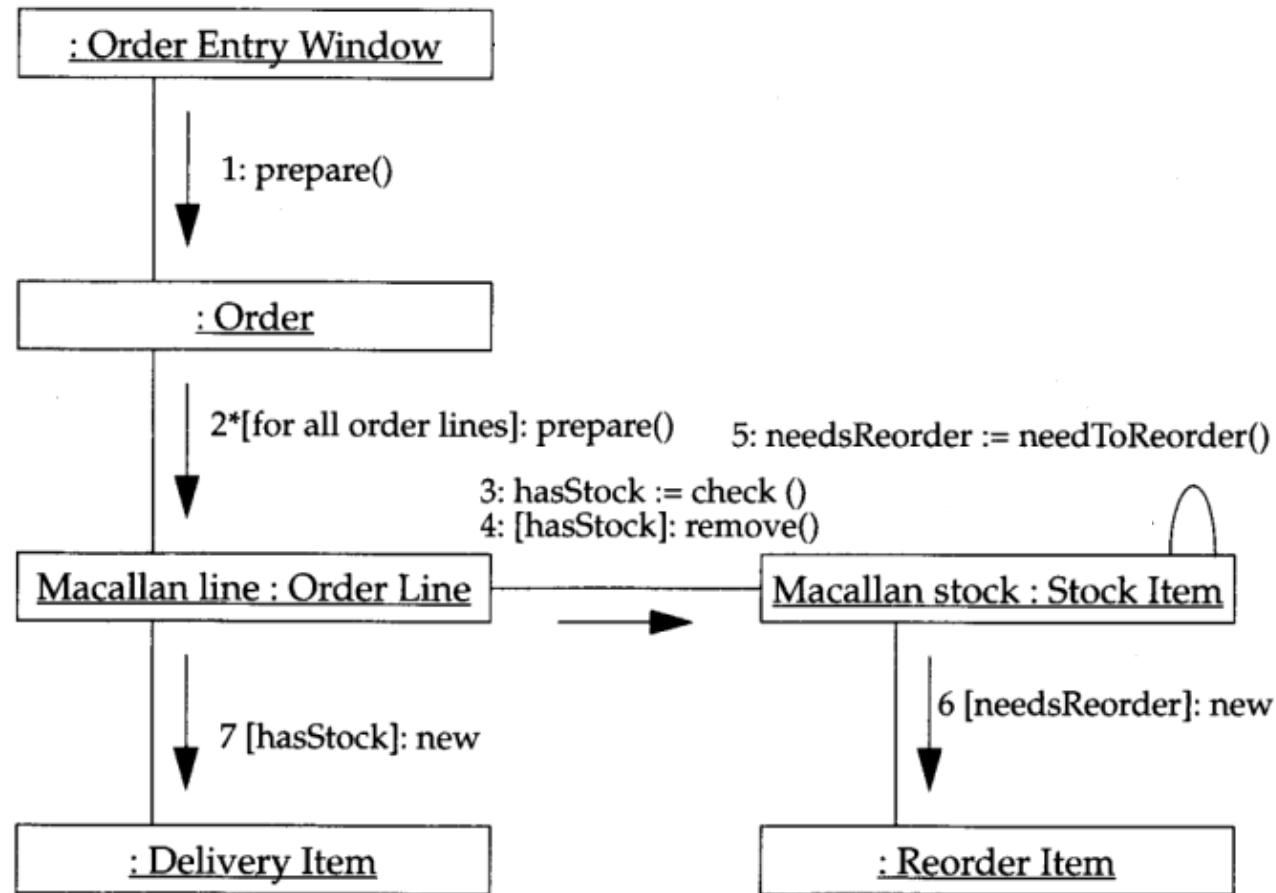
Administrivia

- Midterm exam Thursday
 - Review session Wednesday, Feb 10th, 7-9 p.m. DH 1212
- Homework 4 coming soon
 - Three parts, part A due Feb 18th
 - Design review meetings next week
- Collaboration policy...

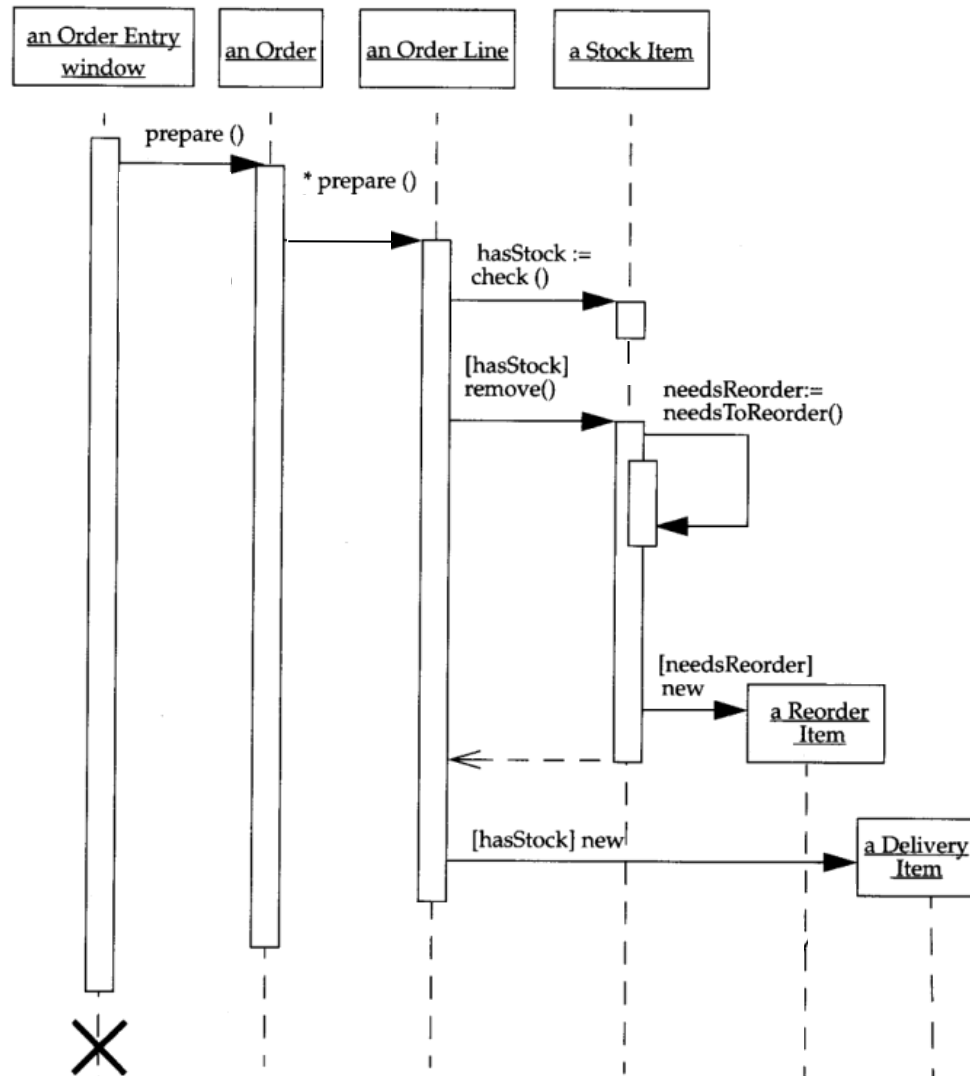


Key concepts from Thursday...

Communication diagrams to visualize dynamic behavior



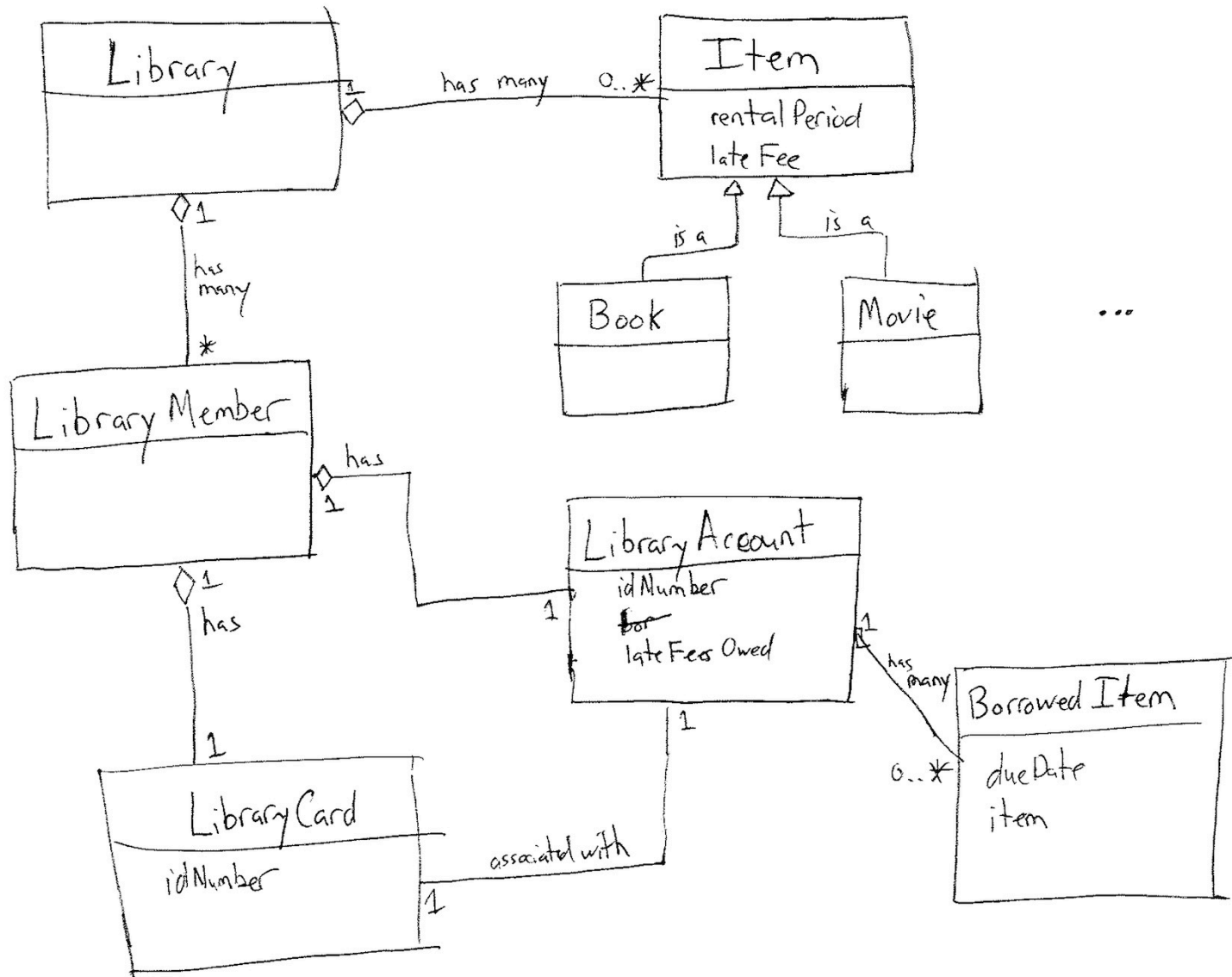
Sequence diagrams to visualize dynamic behavior



Design principles

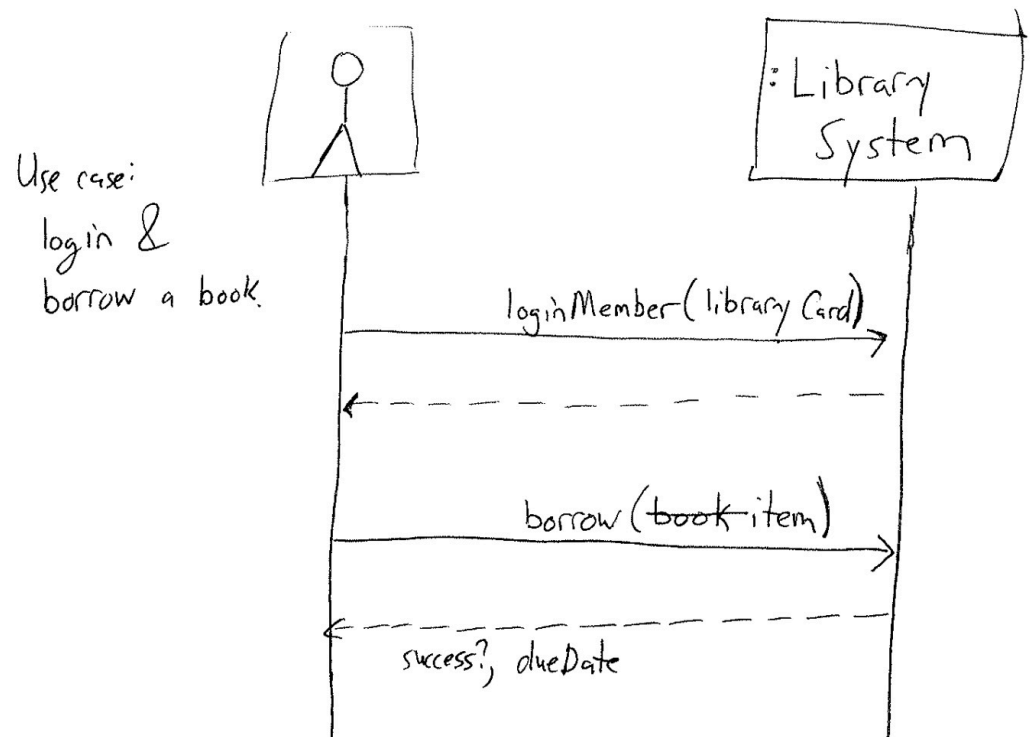
- Low coupling
- Low representational gap
- High cohesion

A 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.

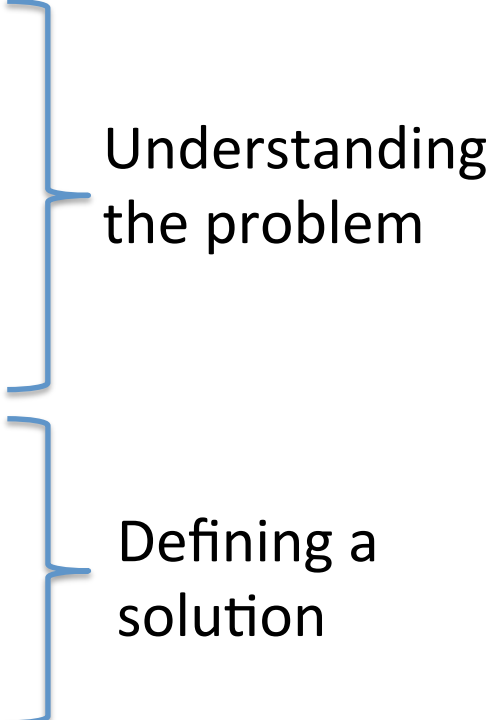
Post-conditions: Logged-in member's account records the newly-borrowed item, or the member is warned she has an outstanding late fee.
The newly-borrowed item contains a future due date, computed as the item's rental period plus the current date.

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

Draw a domain model for cryptarithm solving

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

Learning goals for today

Today:

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

Object-oriented programming

- Programming based on structures that contain both data and methods



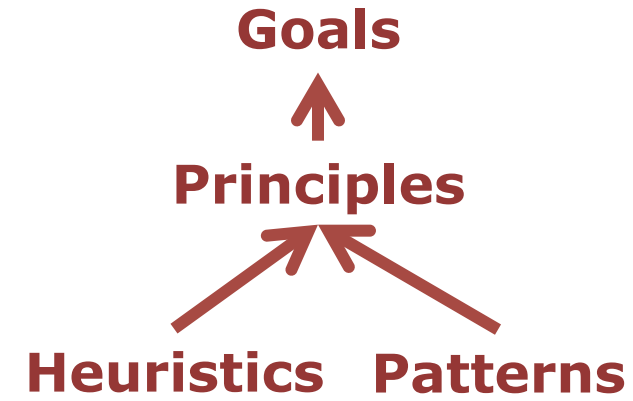
```
public class Bicycle {  
    private int speed;  
    private final Wheel frontWheel, rearWheel;  
    private final Seat seat;  
    ...  
  
    public Bicycle(...) { ... }  
  
    public void accelerate() {  
        speed++;  
    }  
  
    public int getSpeed() { return speed; }  
}
```

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

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 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 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: Straightforward assignment from domain model
 - Key principles: Low representational gap and low coupling

Use the information expert heuristic

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

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?

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

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) has responsibility for solving the cryptarithm

A key 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?

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 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...