

Principles of Software Construction: Objects, Design, and Concurrency

A formal design process

Josh Bloch Charlie Garrod **Darya Melicher**



Administrivia

- Homework 2 feedback in your GitHub repository
- Homework 3 due today at 11:59 p.m.
- Homework 4 available end of this week
- Midterm exam next Thursday, September 27th
 - Review session: Tuesday, September 25th, 5–7 p.m. in Doherty 2315
 - There will be pizza!
 - Practice exam available by tomorrow on Piazza
- Optional reading due today: UML and Patterns Ch 17 & Effective Java Items 49, 54, and 69
 - Required reading due next Tuesday: UML and Patterns Ch 14, 15, and 16

Key concepts from Tuesday

Iterator design pattern

- Problem: Clients need uniform strategy to access all elements in a container, independent of the container type
 - Order is unspecified, but access every element once
- Solution: A strategy pattern for iteration
- Consequences:
 - Hides internal implementation of underlying container
 - Easy to change container type
 - Facilitates communication between parts of the program

Decorator design pattern

- Problem: You need arbitrary or dynamically composable extensions to individual objects
- Solution: Implement a common interface as the object you are extending, add functionality, but delegate primary responsibility to an underlying object
- Consequences:
 - More flexible than static inheritance
 - Customizable, cohesive extensions
 - Breaks object identity, self-references

Design principles are useful heuristics

- Reduce coupling to increase understandability, reuse
- Lower representational gap to increase understandability, maintainability
- Increase cohesion to increase understandability

High-level software design process

- Project
- Gather
- Define a
- 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
- Maintain

Assumption:
Somebody has gathered the
requirements (mostly text)

17-313

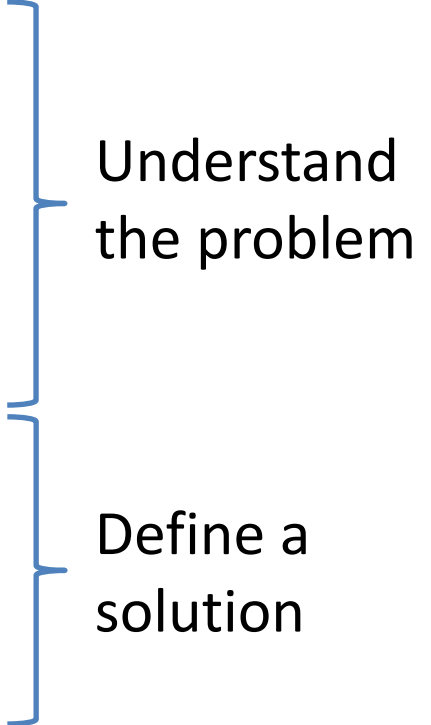
Social
aspects

17-214

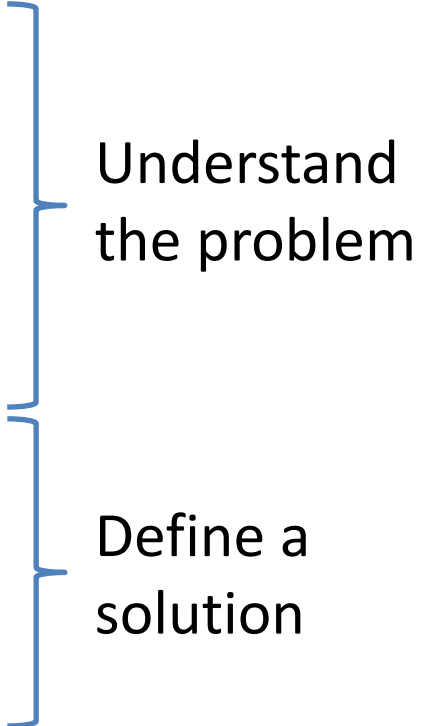
Challenges:
How do we implement them?
How do we cope with changes?

...

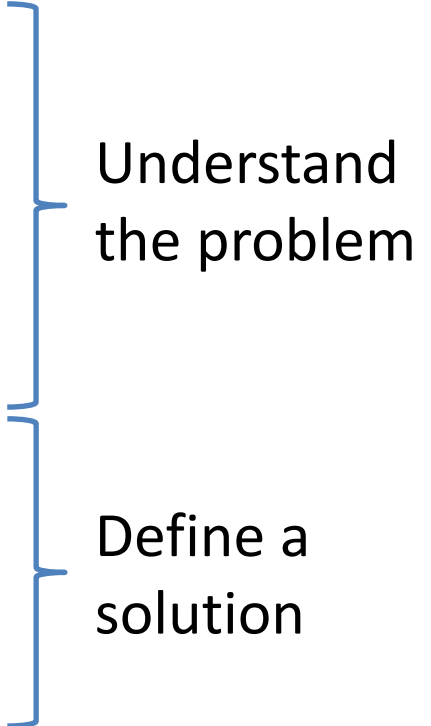
Artifacts of the design process

- 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
- 
- Understand the problem
- Define a solution

Today and next Tuesday you will...

- 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
- 
- Understand the problem
- Define a solution

Today and next Tuesday you will...

- **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
- 
- Understand the problem
- Define 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 rental period, which determines the item's due date when the item is borrowed. If a member renews a borrowed item, the library system should extend the item's due date by the rental period. The library system should also calculate and record late fees for borrowed items that are not returned by the due date. The library system should also calculate and record the total late fees for a 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.

Modeling a problem domain

1. 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

Build 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.



Read description carefully, look for nouns and **verbs**

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.

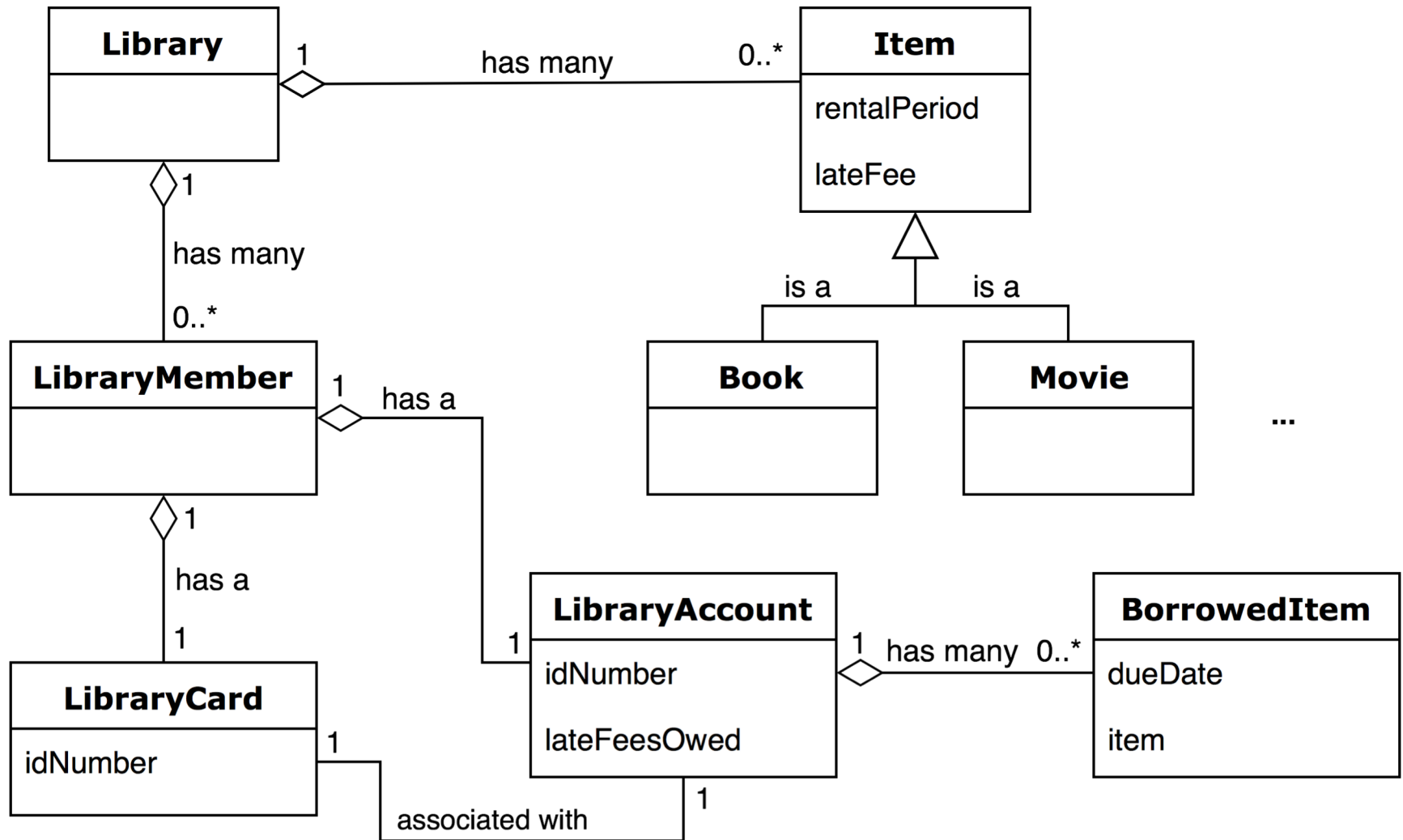
A member's library account **records** which items the member has borrowed and the due date for each borrowed item. Each type of item has a default rental period, which determines the item's due date when the item is borrowed. If a member **returns** an item after the item's due date, the member **owes**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



Modeling a problem domain

1. 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
2. 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
 - Include notes as needed
 - Expect revisions

An example domain model for a 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-world 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 an airport self-check-in system

An airport self-check-in system is used to check in passengers for their upcoming flights. Each passenger is assigned a seat on each of their flights. During the check-in procedure, the passenger may choose to check in one or more bags as their baggage. At the end of a check-in procedure, the passenger receives a boarding pass for each of the flights of their trip they checked into.



Build an airport self-check-in system

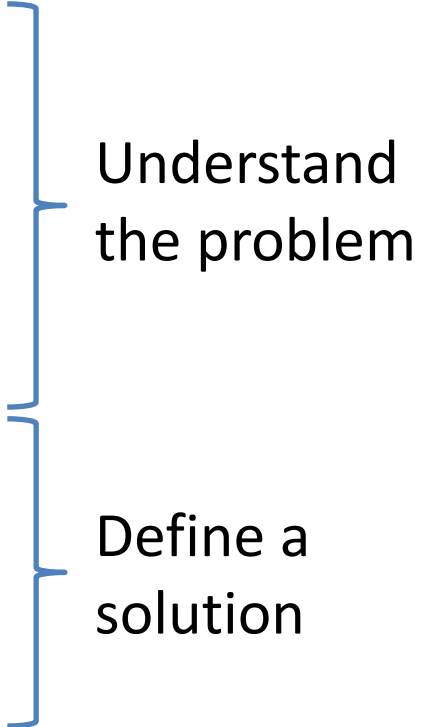
An airport self-check-in system is used to **check in** passengers for their upcoming flights. Each passenger **is assigned** a seat on each of their flights. During the check-in procedure, the passenger may choose to **check in** one or more bags as their baggage. At the end of a check-in procedure, the passenger **receives** a boarding pass for each of the flights of their trip they checked into.



Benefits of domain modeling

- Understand the domain
 - Details matter! E.g., how many items a library user may check out?
- Ensure completeness
 - e.g. library user's standing (fees due) affects the ability to check out items
- Agree on a common set of terms
 - e.g. library user vs. library member, rented item vs. checked-out item
- Prepare to design
 - Domain concepts are good candidates for OO classes (low representational gap)

Today and next Tuesday you will...

- 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
- 
- Understand the problem
- Define a solution

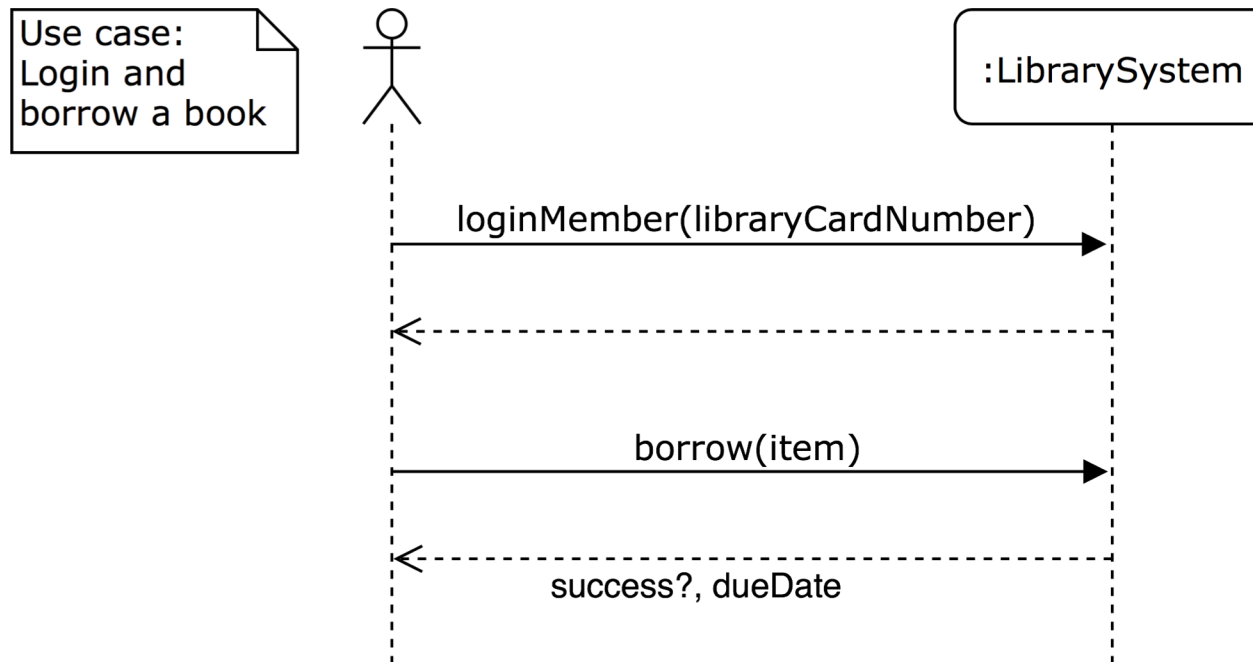
Understanding system behavior with sequence diagrams

- *A system sequence diagram* shows, for one use case scenario, 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
 - Useful for identifying tests later
- Input: Domain description and one use case
- Output: A sequence diagram of system-level operations
 - Include only domain-level concepts and operations

A 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.

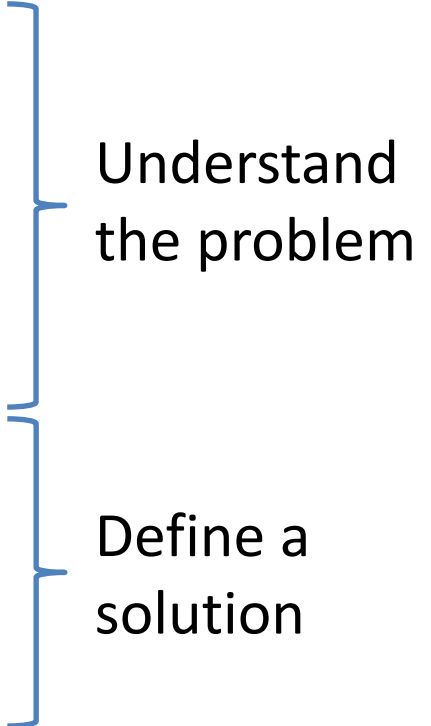


A sequence diagram for the airport self-check-in system

Use case scenario: After a passenger is authenticated with the system using their booking ID, they should be able to change a seat on any of their flights to any unoccupied seat in the cabin of the same type.



Today and next Tuesday you will...

- 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
- 
- Understand the problem
- Define a solution

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 behavior contract for the airport self-check-in

Use case scenario: After a passenger is authenticated with the system, they should be able to change a seat on any of their flights to any unoccupied seat in the same cabin.

Operation: `changeSeat(flight, newSeat)`

Pre-conditions:

Post-conditions:

