Design

Dr. Isaac Griffith Idaho State University

Outcomes

Idaho State University | Computer Science

After today's lecture you will be able to:

- How we extract responsibilities from Use Cases
- How we select and identify classes that will be constructed in the software

Design

CS 2263

Introduction

- We now have the requirements of the system specified
- We next begin the task of designing our system
- **Goal:** design a system which behaves as specified by our model
  - **Tool:** UML Sequence Diagrams
- **Tasks:**
  - Breakdown system actions into specific tasks
  - Breakdown tasks into responsibilities
  - Assign these responsibilities to system entities
    - identify the public methods of each class
    - describe the function of each method
- **Outcome:** A design which specifies enough detail to be implemented in code.
  - Moving us into the implementation phase

Design Questions

Idaho State University | Computer Science

We must answer the following questions

1. On what platform(s) (hardware and software) will the system run?
   - Linux
   - MacOS
   - Windows
   - A combination thereof
2. What languages and paradigms will be used for implementation?
   - Functional vs. Imperative
   - Procedural vs. OO
   - Java, C++, Scala, Ruby, Go, C, etc.
   - Will depend on the needs of the project and the expertise of the engineers
3. What user interfaces will the system provide?
   - CLI, GUI, Web
4. What classes and interfaces need to be coded? What are their responsibilities?

Design Questions

Idaho State University | Computer Science

We must answer the following questions

5. How is data stored on a permanent basis?
   - What medium will be used?
   - What model will be used for data storage?
6. What happens if there is a failure?
   - We should strive to prevent data loss
   - What mechanisms are needed for realizing this?
7. Will the system use multiple computers.
   - What are the issues related to data and code distribution?
8. What kind of protection mechanisms will the system use?

Design Phase

CS 2263

Design Phase

Idaho State University | Computer Science

- In this phase we have several steps to complete
  1. Identify the major subsystems
  2. Create the Software Classes
  3. Assign Responsibilities to the Classes
  4. Transition from Software Classes to Implementation Classes
  5. User Interface
  6. Data Storage

Major Subsystems

Idaho State University | Computer Science

- Our first step is to identify the major subsystems of our system.
- In the Library Example, we have two major subsystems:
  - **Business Logic:**
    - Input data processing
    - Data creation
    - Queries
    - Data Updates
    - External storage, storing/retrieving data
  - **User Interface:**
    - Interacts with the user
    - Accepts and outputs information
- We should apply good design principles here
  - Separation of Concerns and Encapsulate what Varies
  - High Cohesion in the modules

Creating Software Classes

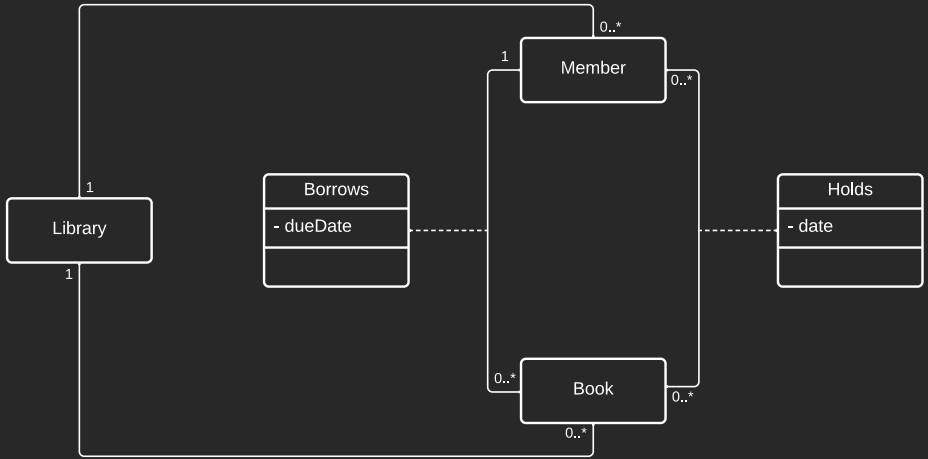
Idaho State University | Computer Science

- In the prior phase we created conceptual classes and class diagrams
  - Conceptual classes are from the “essential perspective”
  - In this phase we want transition to **Software Classes**
    - The conceptual classes act as a starting point, or first guess
- small **Software Classes** - are a more “concrete” set of classes which will correspond to system components.
- This transition is an iterative process:
  1. Come up with a set of classes
  2. Assign responsibilities to the classes and determine the necessary data structures and methods
     - Will most likely require several iterations, where classes may be
       - Added
       - Removed
       - Merged
       - Split

Which Classes

- Using our conceptual classes as a starting point
- We review the classes identified to see if the will remain going forward
- The conceptual classes were as follows:
  - Member
  - Book
  - Library
  - Borrows
  - Holds

Conceptual Model



Member, Book, Library

- Both `Member` and `Book` are key concepts to the system
  - These are definitely here to stay

- `Library` is a more difficult concept
  - This is where the key business logic will be implemented or called from
  - Additionally, just like a real library, it will need to track collections, specifically
    - A collection of members -> Which we will implement as the singleton class `MemberList`
    - A collection of books -> Which we will implement as the singleton class `Catalog`

Borrows and Holds

- These are the two association classes
  - Because association classes have no representation in code, we need to deal with them
- **Borrows**
  - This is on a one-to-many relationship, and thus we can simply move all of its attributes to the many side
- **Holds** is a bit different
  - Because it is on a many-to-many, we cannot just move it to one or the other side
  - Instead we need to create a class which is accessible to both `Members` and `Books`

Assigning Responsibilities

Idaho State University | Computer Science

- We now have a good set of software classes:
  - `Member`
  - `Book`
  - `Library`
  - `MemberList`
  - `Catalog`
  - `Hold`
- We now need to assign the responsibilities to these classes
  - This is done by expanding out the use cases
  - And devolving this expanded behavior into the respective classes
  - The tool of this devolution is the **sequence diagram**

Sequence Diagrams

Idaho State University | Computer Science

- To construct our sequence diagrams (and thus assign responsibilities) we have some work ahead of us
- For each system response in the right-hand column of each use case:
  - Specify the sequence in which the operations will occur
    - Requires a complete algorithm
  - Specify how each operation will be carried out
    - Specifies the classes involved in each step of the algorithm
    - This should fully specify these classes
    - Should specify the methods, their parameters, and return types
- Often, we will need to make design decisions along the way

Sequence Diagrams

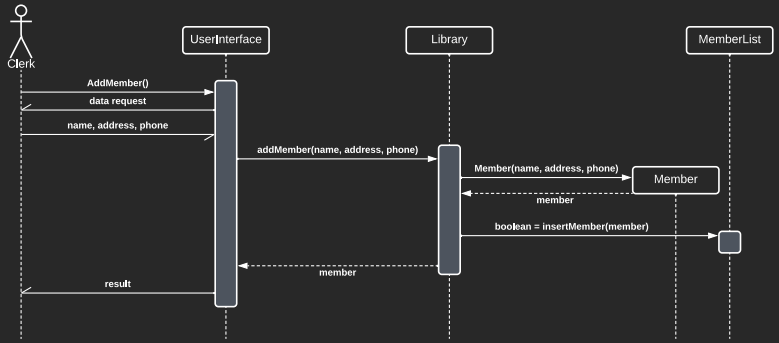
- For the Library System we have the following operations to describe:
  - Register Member
  - Add Books
  - Issue Books
  - Return Books
  - Remove Books
  - Member Transactions
  - Place Hold
  - Process Holds
  - Remove Hold
  - Renew Books

Register Member

Idaho State University | Computer Science

Algorithm

1. Create `Member` Object
2. Add the `Member` object to the list of members
3. Return the result of the operation



Design Decision

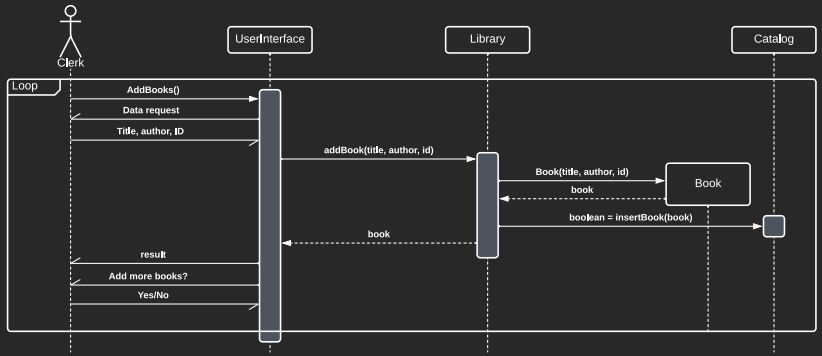
- To carry out the first two steps there are two options:
  1. Invoke `Member`'s constructor from within `addMember` of `Library` **(preferred)**
  2. Invoke `addNewMember` on `MemberList` passing all the needed info to create a new member there.
- Option 2 introduces unnecessary coupling between `MemberList` and `Member`

Add Books

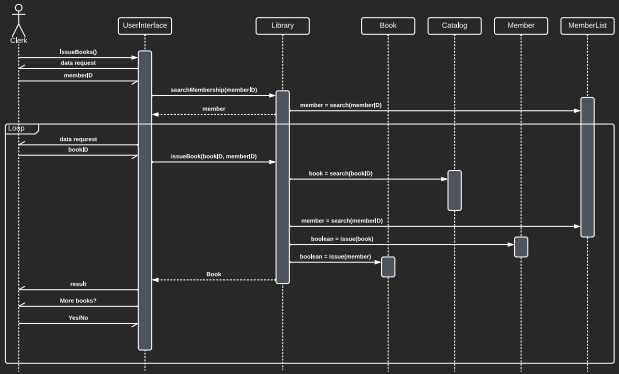
Idaho State University | Computer Science

Algorithm

1. Create a `Book` object
2. Add the `Book` object to the catalog
3. Return the result



Issue Books

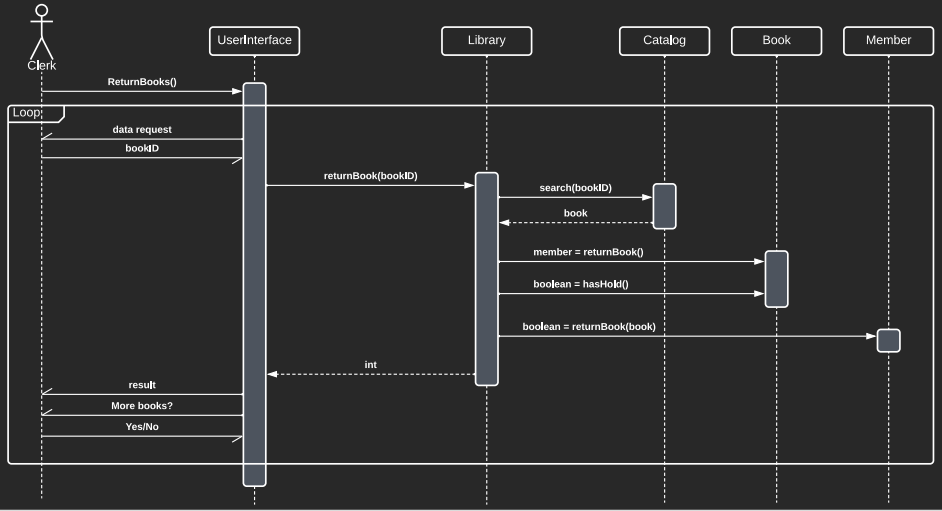


Design Decision

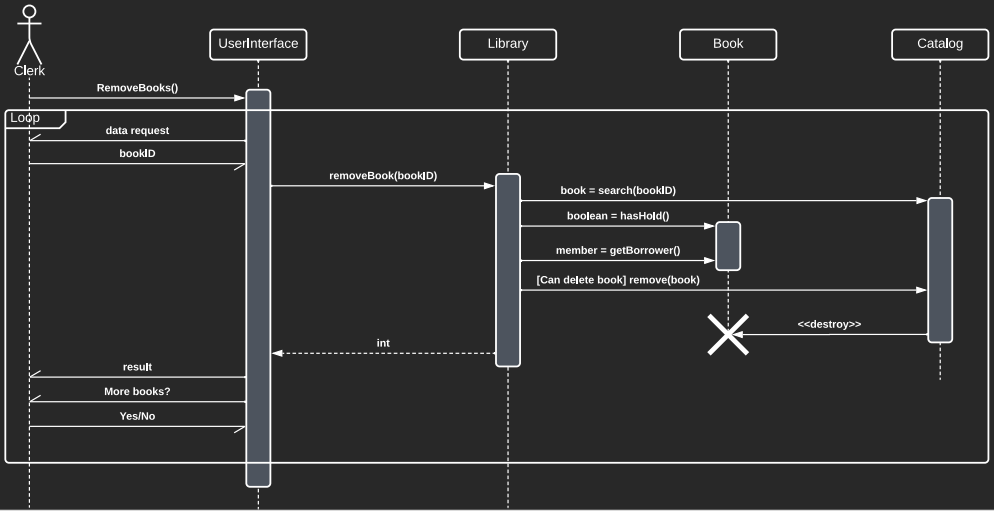
- We need to determine how to search for members
  1. Iterate across Member objects using an iterator from `MemberList`, and compare ID's to a target ID
  2. Delegate responsibility to `MemberList`

Return Books

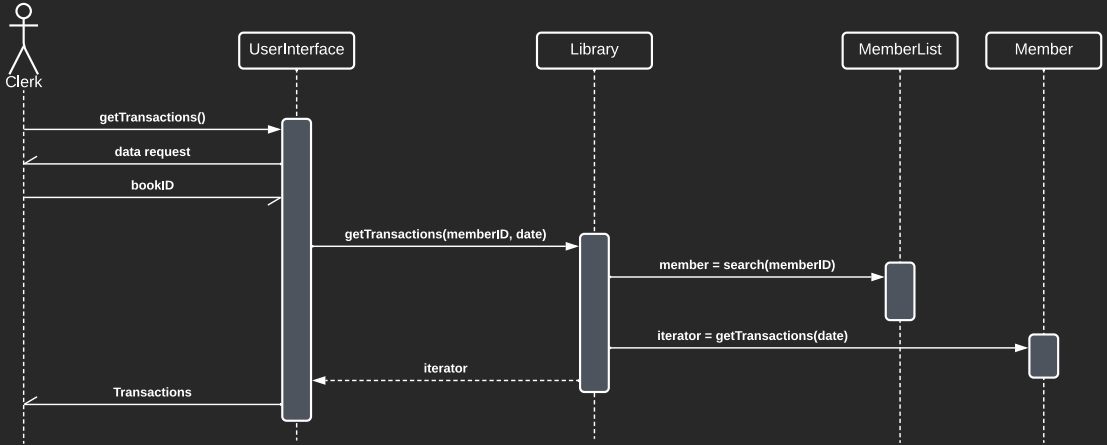
Idaho State University | Computer Science



Remove Books

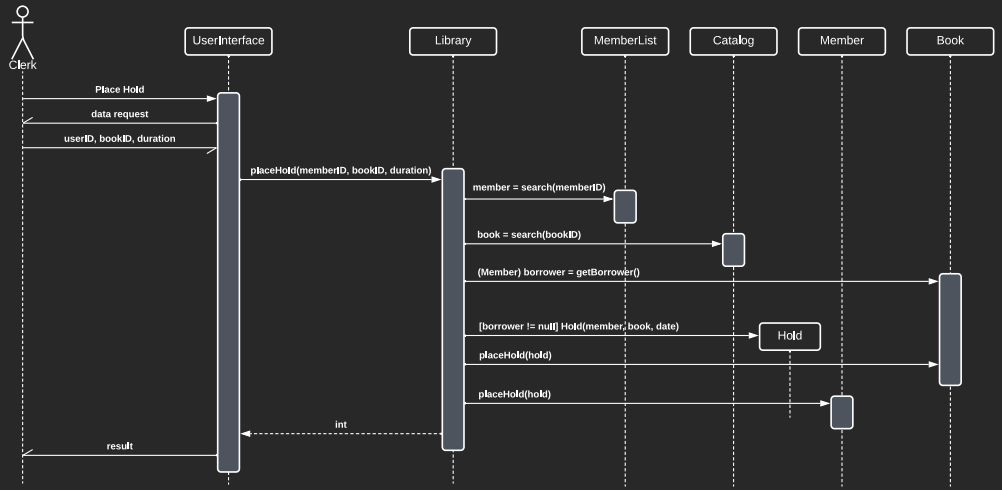


Member Transactions

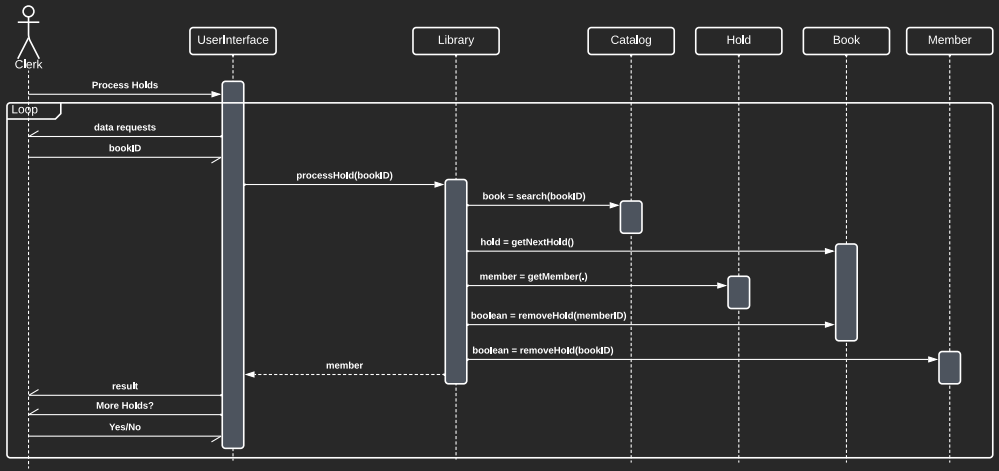


Place Hold

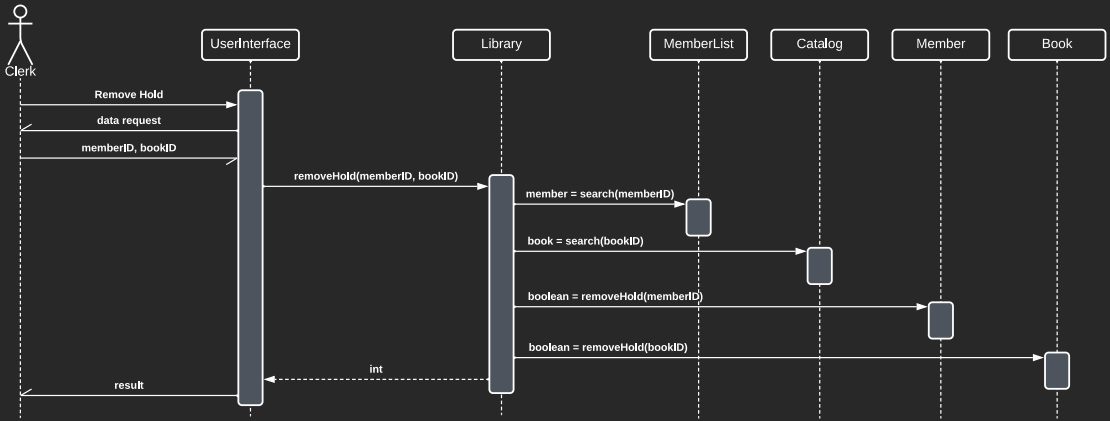
Idaho State University | Computer Science



Process Holds

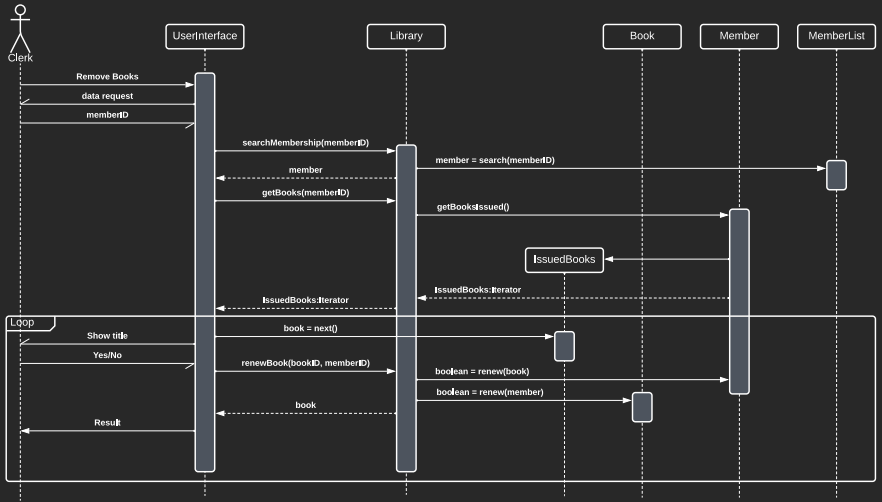


Remove Hold


Clerk
UserInterface
Library
MemberList
Catalog
Member
Book
Remove Hold
data request
memberID, bookID
removeHold(memberID, bookID)
member = search(memberID)
book = search(bookID)
boolean = removeHold(memberID)
boolean = removeHold(bookID)
int
result

Renew Books

Idaho State University | Computer Science



Coupling and Cohesion

Idaho State University | Computer Science

- We must keep in mind **coupling** and **cohesion** when designing systems.
  - We want **good cohesion** between entities grouped together or placed within a subsystem
  - We want **loose coupling** between these same entities
- This is why we separated out the two modules of our system:
  - User interface
  - Business Logic
- Additionally, we must keep in mind how we assign responsibilities to classes.
  - Classes should only be assigned those responsibilities necessary to utilize the data (fields) of the class
  - These responsibilities then become the class' methods

For Next Time

- Review Chapter 7.1
- Review this lecture
- Read Chapter 7.1 - 7.3
- Come to class

Are there any questions?