Implementing a Design

Dr. Isaac Griffith Idaho State University

Outcomes

Idaho State University | Computer Science

After today's lecture you will be able to:

- How we generate implementation class diagrams from sequence diagrams
- How we convert class diagrams into code
- How we convert sequence diagrams into code

Class Diagrams

CS 2263

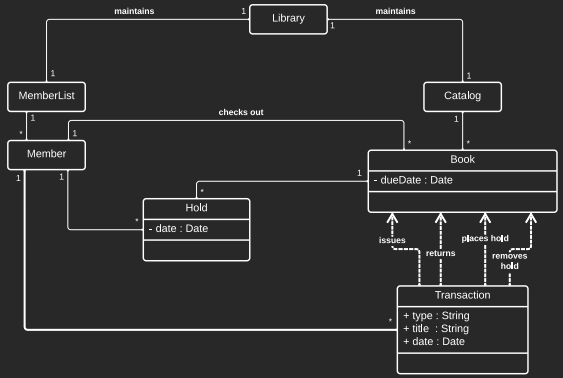
Class Diagrams

Idaho State University | Computer Science

- As we complete our sequence diagrams, we should also have identified all necessary software classes.
- For the Library System, the software classes are:
  - `Library`
  - `MemberList`
  - `Catalog`
  - `Member`
  - `Book`
  - `Hold`
  - `Transaction`
- We then need to flesh out the classes
  - We add methods by examining each sequence diagram
  - We add attributes by examining the methods

Overall

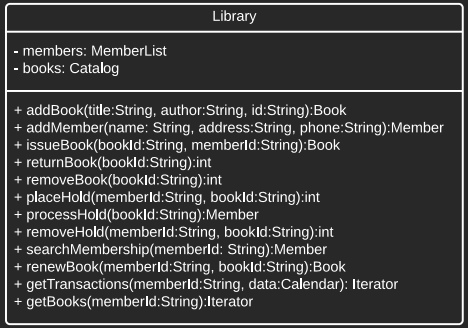
- We Note the following things
  - Holds is a class between `Book` and `Member`
  - `Transaction` is used to record transactions
    - with connections to book (as it holds the title)
  - `Library` is the **Facade** to the business logic
    - All updated are done by invoking methods on the Facade
    - rather than any class/method it controls



Library

Design

- All methods and parameters are extracted from the sequence diagrams
- Method return types are provided (not originally specified in diagrams)
- Connects to `MemberList` and `Catalog`

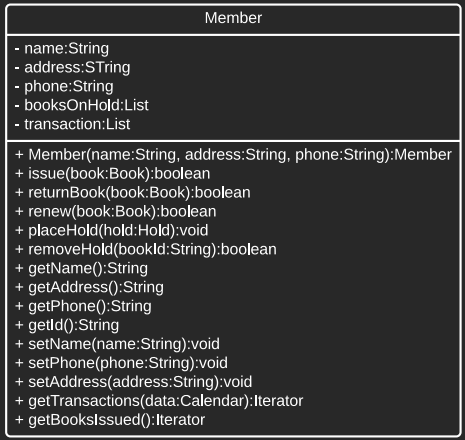


Member

Idaho State University | Computer Science

Design

- Sequence diagrams provide
  - Methods
  - Attributes
- Member generates its own ID
  - We need to ensure that these are unique among instances
  - Thus we will need static methods (not shown)
  - Will provide decentralized control with responsibilities close to the data



Book

Idaho State University | Computer Science

Design

- Use same approach to extract methods, attributes, parameters
- **Note:** No setters (we do not expect to alter data once entered)

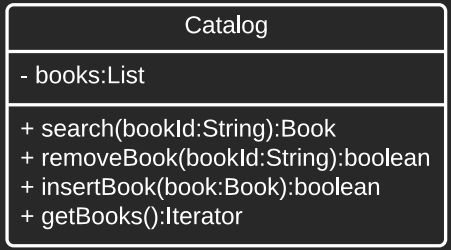
Book
- title:String - author:String - id:String - borrowedBy: Member - holds:List - dueDate:Calendar
+ Book(title:String, author:String, id:String):Book + issue(member:Member):boolean + returnBook():Member + renew(member:Member):boolean + placeHold(hold:Hold):void + removeHold(memberId:String):boolean + getNextHold():Hold + getHolds():Iterator + hasHold():boolean + getDueDate():Calendar + getBorrower():Member + getAuthor():String + getTitle():String + getId():String

Catalog

Idaho State University | Computer Science

Design

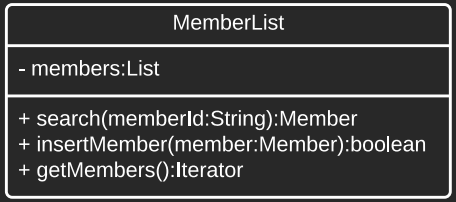
- Requires typical methods
  - `search`
  - `insert`
  - `remove`
- Only attribute
  - an internal collection of books



MemberList

Design

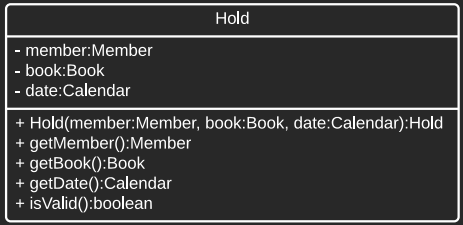
- Same as was done for `Catalog`



Hold

Design

- Basic accessors
- Adds in a method `isValid`
  - checks whether a certain hold is still valid

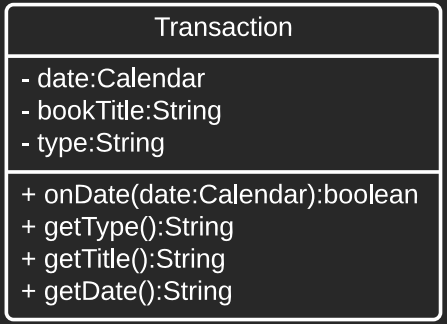


Transaction

Idaho State University | Computer Science

Design

- Handles data for each individual transaction
- Transaction dependencies to book represent the different member transactions:
  - Issues a book
  - Returns a book
  - Places a hold on a book
  - Removes a hold from a book



Importing/Exporting Objects

Idaho State University | Computer Science

- When implementing systems, we need to take care of managing references.
- It always seems convenient to return a reference to an object or collection of objects. But this can often be the wrong choice.
- This is where multiplicity comes into play.
  - When implementing our system we need to take care to ensure that the multiplicities are maintained
  - We also need to ensure that objects are not just sharing references around but rather encapsulate the data the contain
  - So remember not to export (share through a getter) contained collections or other internal objects

User Interface

Idaho State University | Computer Science

- Presents a menu to the user using a simple CLI:

0. Exit
1. Add a member
2. Add books
3. Issue books
4. Return books
5. Renew books
6. Remove books
7. Place a hold on a book
8. Remove a hold on a book
9. Process holds
10. Print a member's transactions on a given date
11. Save data for long-term storage
12. Retrieve data from storage
13. Help

- User can enter a number from 0 - 13
- Parameters will then be required for the selected operation
- Once complete the results will be displayed

Data Storage

- As we have noted before, most systems will need to store data on a long-term basis

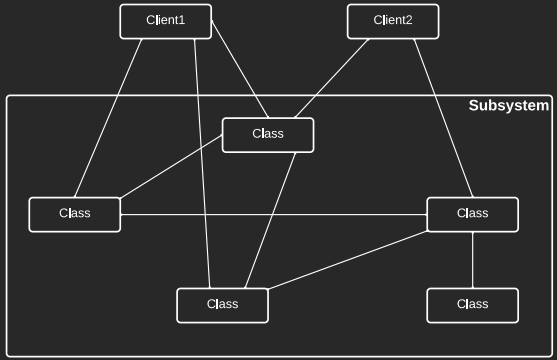
- Towards this end, we originally noted a need for the following two commands
  - A command to save the data on a long-term basis
    - data is copied to a secondary storage device
  - A command to load data from a long-term device
    - data is loaded from a secondary storage device to recreate objects

Implementation Walkthrough

CS 2263

Facade Pattern

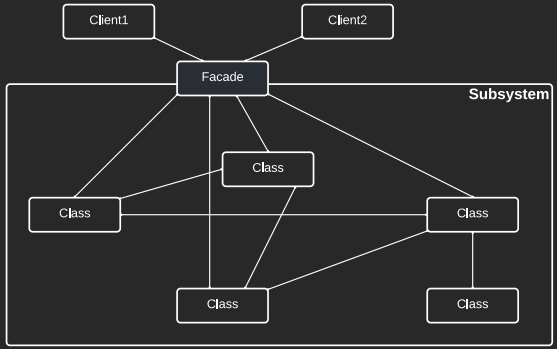
- **Problem Faced** - We have several clients accessing a subsystem
  - The clients have too much access to the underlying details of the subsystem
  - This creates tight coupling between the client and the subsystem
- Facade solves this by
  - Providing a single point of access to the subsystem
  - Adapts to changes in classes so that the client does not have to



Facade Pattern

Idaho State University | Computer Science

- The `Library Class` is an instance of a **Facade** pattern
- **Motivation** - To reduce complexity by minimizing communication and dependencies between a subsystem and its clients
  - Shields the client from the subsystem
  - Enables loose coupling between client and subsystem components
- **Using a Facade** - Employed in a situation where we have:
  1. A system with several individual classes, each with its own set of public methods
  2. An external entity interacting with the systems requires knowledge of the public methods of several classes



For Next Time

- Review Chapter 7
- Review this lecture
- Come to class

Are there any questions?