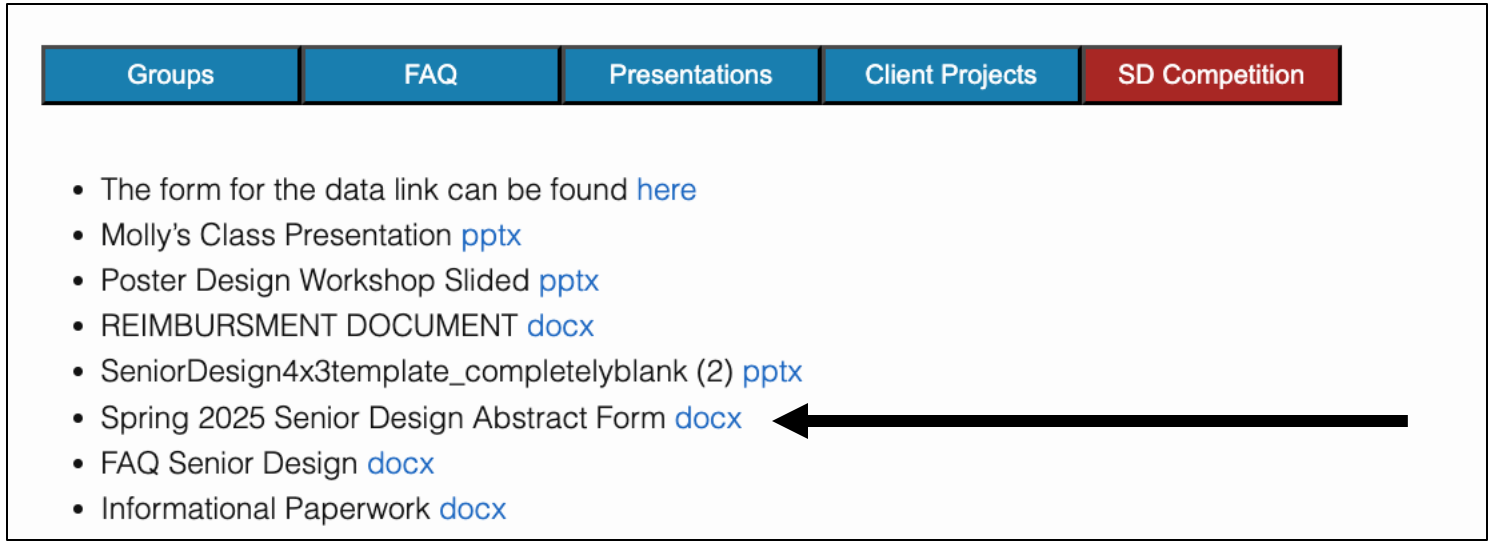
Software Analysis, Design and Implementation

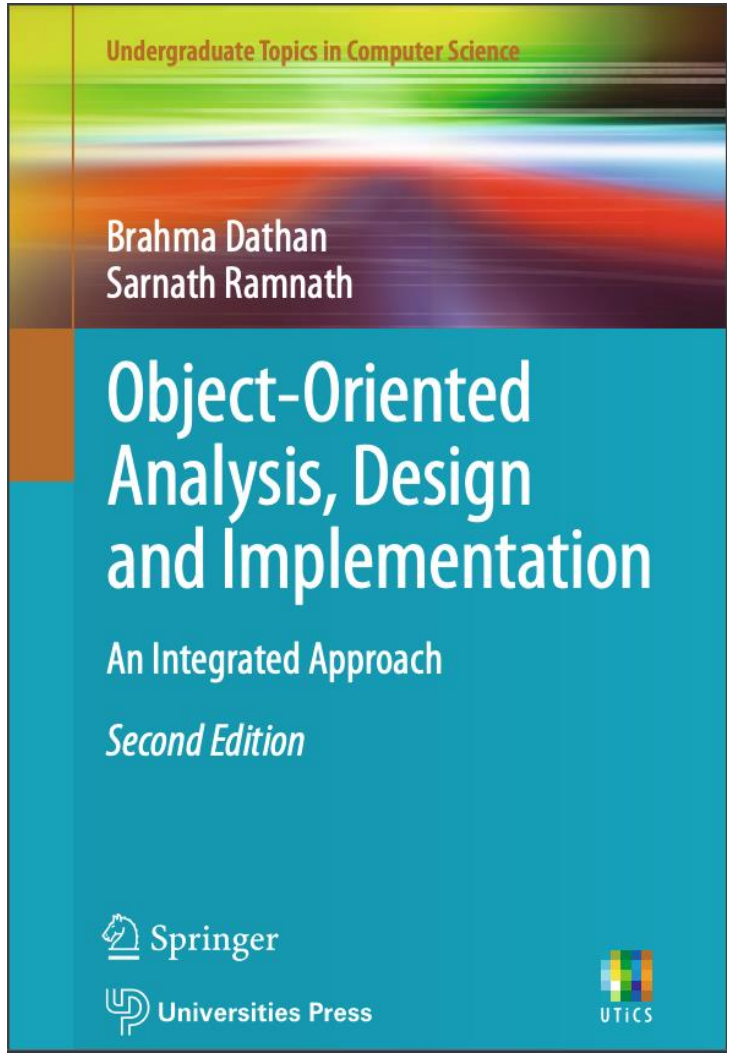
Dr. John Businge

john.businge@unlv.edu

Announcements

- Submit your DP I Tomorrow on Canvas and pin the Google doc on Discord
- Focus on DP II and III – Its both team and individual
  - Individual 60%
  - Team 40%

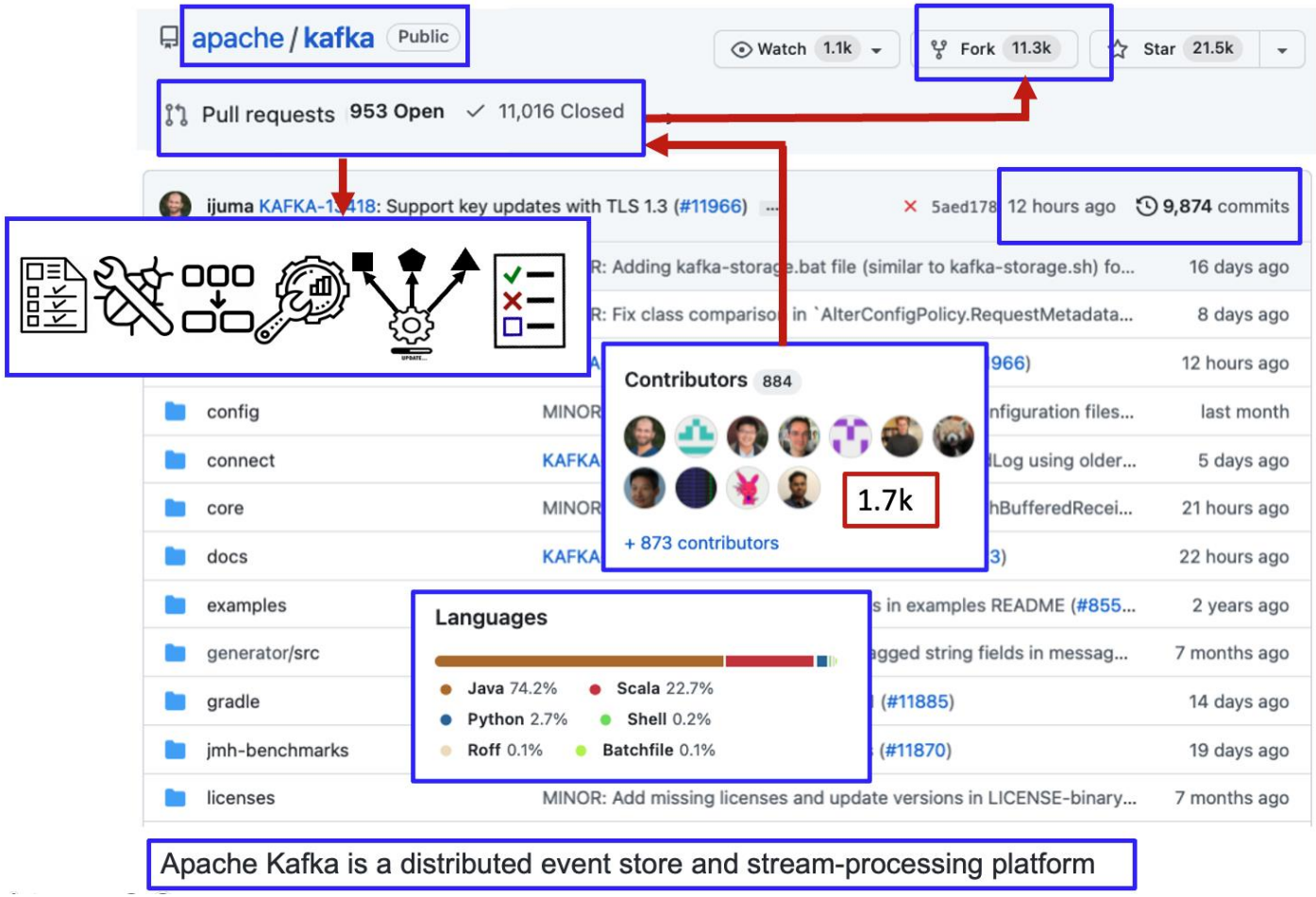




Chapters 6 - 7

- Chapter 6 - Analysing a System
  - https://link.springer.com/chapter/10.1007/978-3-319-24280-4_6
- Chapter 7 - Design and Implementation
  - https://link.springer.com/chapter/10.1007/978-3-319-24280-4_7

Free PDF version on Springer
https://doi.org/10.1007/978-3-319-24280-4_7

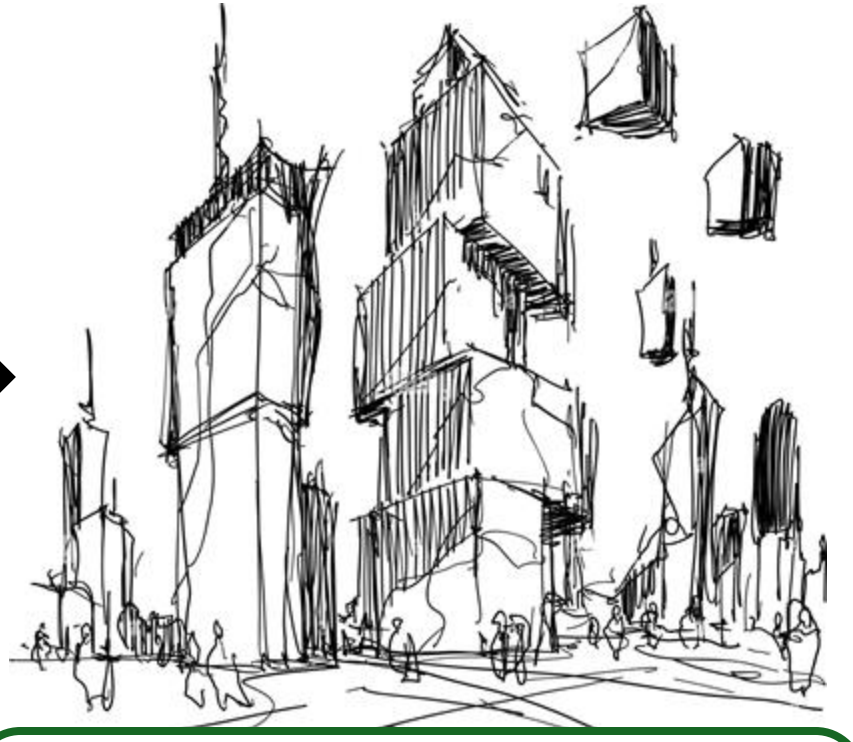


Consider this scenario.
You join a project that's been in development for a while.
You look at the code and become instantly overwhelmed.
If there is no high-level design of the code, your life would be a living hell for you to do anything on this project

Software Requirements, Conceptual and Technical Designs

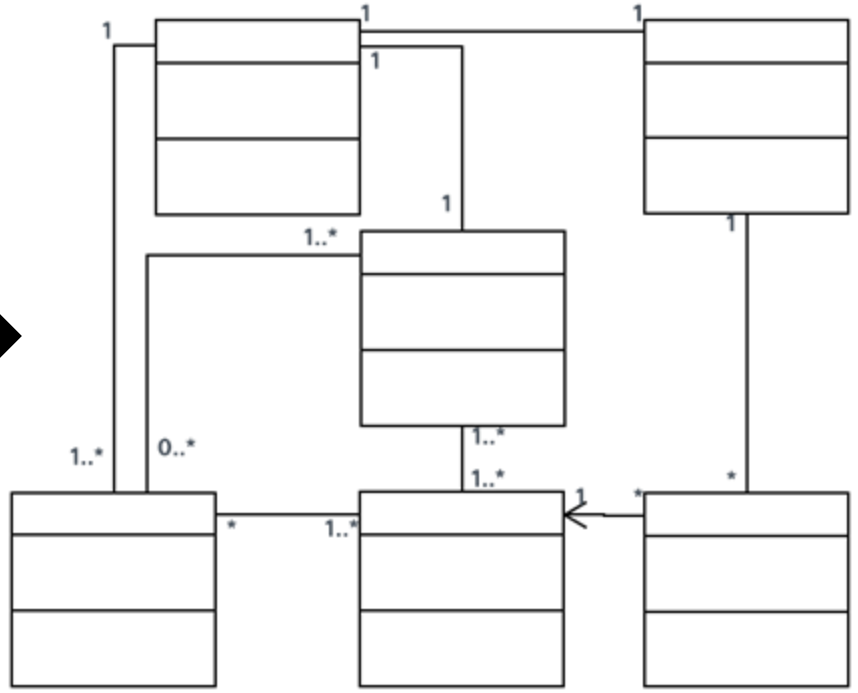
Requirements

Problem Space



Conceptual Design

Solution Space



Technical Design

Conceptual Design vs Technical Design

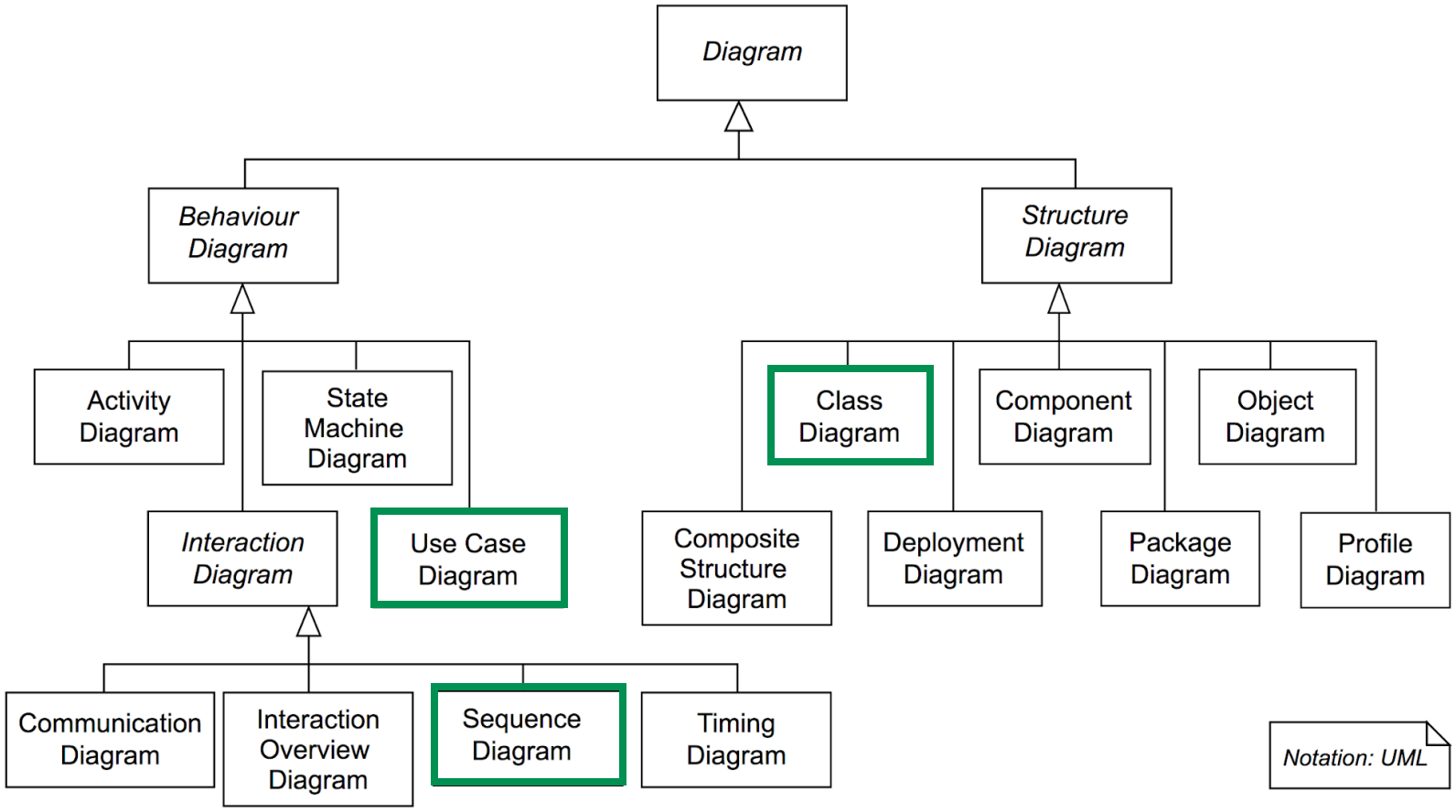
- **Conceptual design** – sets out the stage
  - Defining the system's requirements and functionalities
  - Overall design approach before diving into detailed technical specifications.
  - Data Model
  - User Interfaces design
  - Technological Considerations
- **Technical Design** – dives deeper into specifics
  - **Component Design**: Dividing the system into smaller components with clear roles and interfaces.
  - **Data Design**: Planning the data model, encompassing database schema, structures, flow diagrams, and access methods.
  - **Interface Design**: Specifying interactions between system components, such as APIs, protocols, and communication methods.
  - **Algorithm Design**: Crafting algorithms for data processing, computational logic, and optimization.

UML is the most popular modeling language and become the de-facto standard to design today's large object-oriented systems

There are many Kinds of UML Diagrams

- **Structure diagrams:** show the static architecture of the system irrespective of time.
  - University system - Student, Faculty, etc.
- **Behaviour diagrams**: depicts the behaviour of a system or business process.
- **Interaction diagrams**: show the methods, interactions and activities of the objects.
  - University system: show how a student registers for a course.

There are many Kinds of UML Diagrams

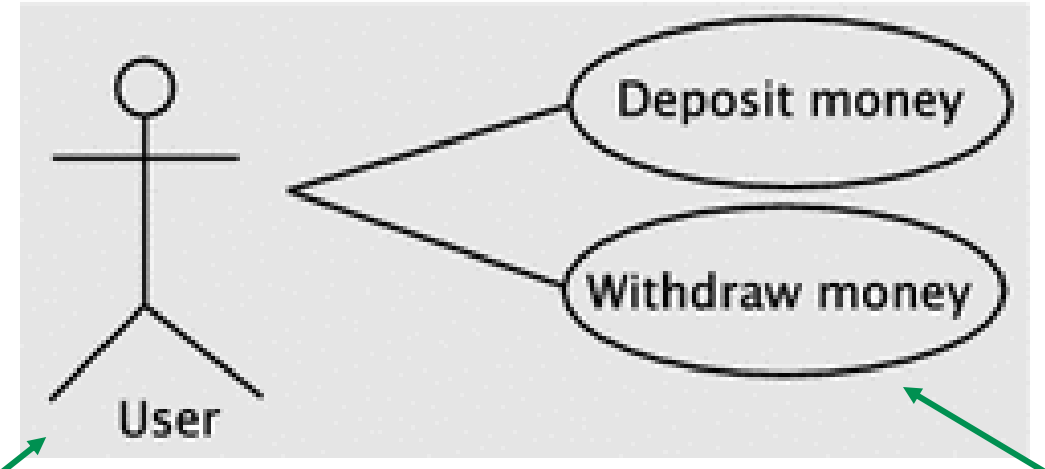


Component Design Overview

- Involves breaking down the system into smaller components or modules and designing their internal structure and interfaces.
  - **Use Case Diagrams**: Identify system functionalities or use cases, offering a high-level view of user/system interactions.
  - **Class Diagrams**: Crucial for designing internal component structure by representing classes, attributes, methods, and their relationships.
  - **Sequence Diagrams**: Detail interactions between components/objects, illustrating message flows and method calls.
- **Significance**: These UML diagrams serve as a blueprint for developers during implementation to ensure meeting requirements and design goals.

Use Case Diagram

Interaction between an actor and the systems



Actor – human or software

System

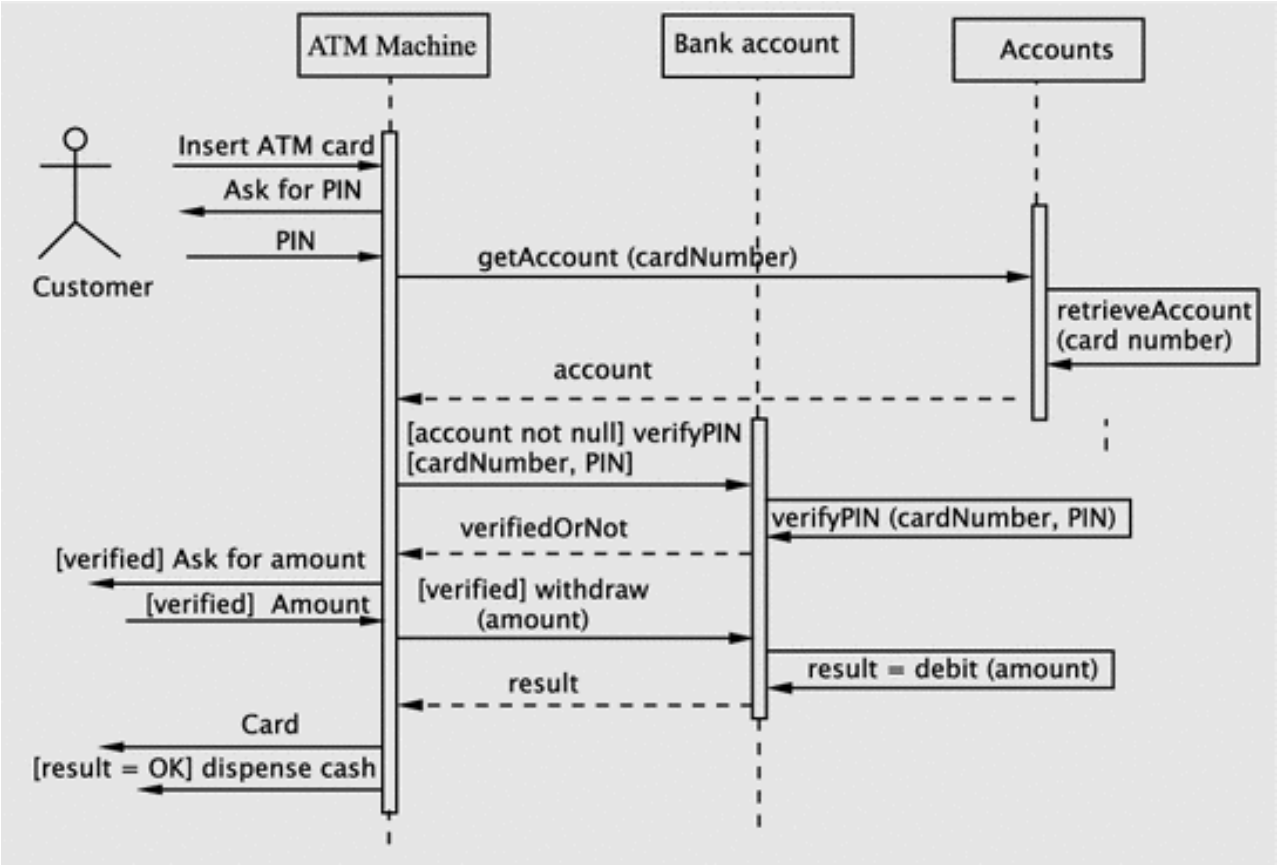
Use Case Diagram

Use case for withdrawing money

	Action performed by the actor		Responses from the system
1.	Inserts debit card into the 'Insert card' slot		
		2.	Asks for the PIN number
3.	Enters the PIN number		
		4.	Verifies the PIN. If the PIN is invalid, displays an error and goes to Step 8. Otherwise, asks for the amount
5.	Enters the amount		
		6.	Verifies that the amount can be with-drawn If not, display an error and goes to Step 8 Otherwise, dispenses the amount and updates the balance
7.	Takes the cash		
		8.	Ejects the card
9.	Takes the card		

Sequence Diagrams

Sequence Diagrams describes how objects in your system interact to complete a specific task.



Class Diagram

Student

- name : String
- address : String
- gpa : double

+ Student(studentName : String, studentAdress: String)
+ Student(studentName : String)
+ Student()
+ setName(studentName : String): void
+ setAdress(studentAdress : String): void
+ getName(): String
+ getGpa(): double
+ getAddress(): String
+ computeGpa(course : Course,grade:char) : void

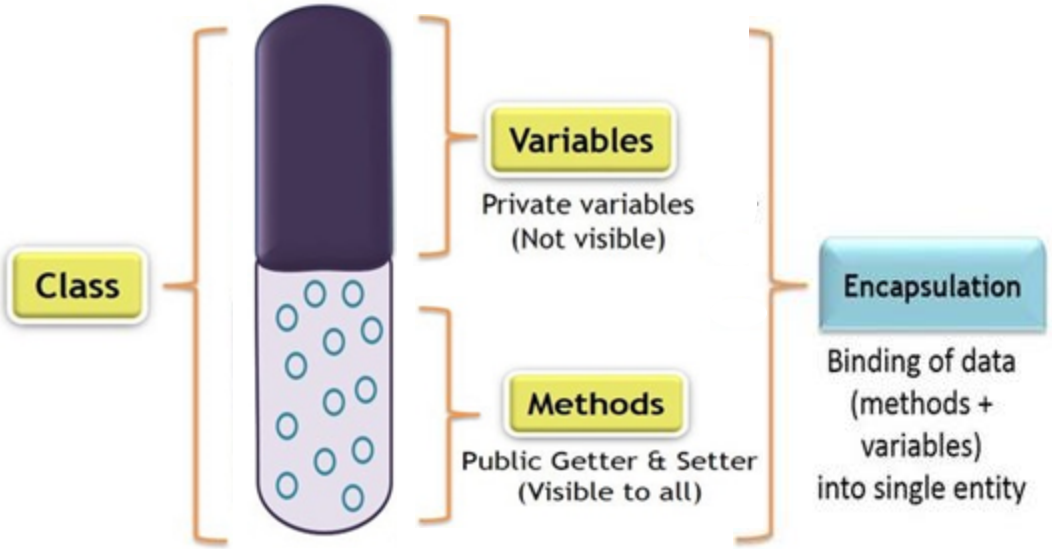
Name

-ve: private

Variables

+ve: public

Methods and their return types



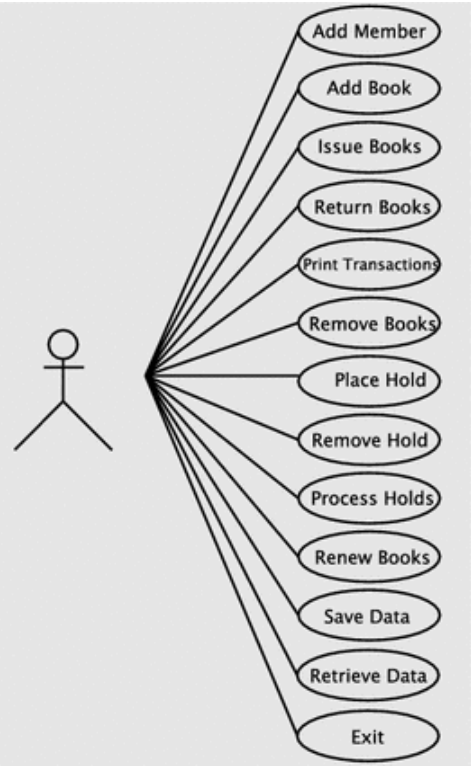
A well-structured class design typically consists of the following components: a name, private attributes, and public methods. This practice embodies the concept of encapsulation, wherein data (comprising methods and variables) is encapsulated within a singular entity.

Object Oriented Analysis, Design and Implementation

- We will examine steps in Object-oriented software development
  - Analysis
  - Design
  - Implementation
- To illustrate the process, let us study a relatively simple example
  - Software to manage a small library
  - Functions:
    - Lending books
    - Receiving back books
    - Doing operations such as: querrying, registering members, e.t.c, and keeping track of the transactions

Analysis phase overview

- Involves gathering, documenting the requirements, and developing a conceptual model of the system.
- Software to manage a small library – functional requirements
  - Register a members
  - Add books
  - Issue books
  - Return books
  - Remove books
  - ….



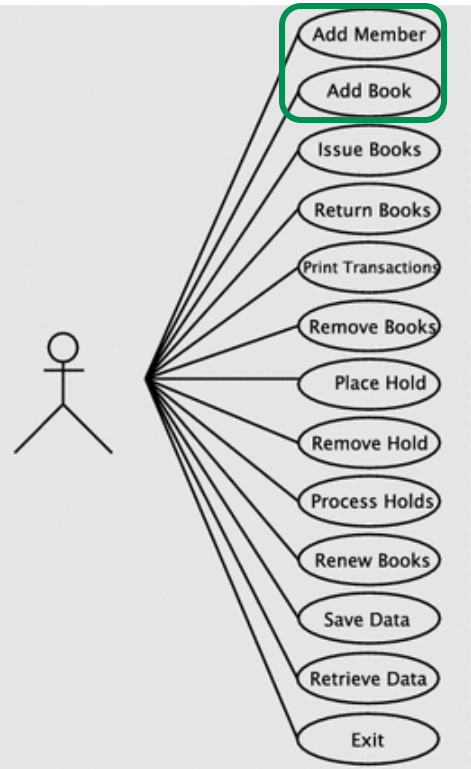
Use case diagram for the library system

Use case Register New Member

Actions performed by the actor	Responses from the system
1. The customer fills out an application form containing the customer's name, address, and phone number and gives this to the clerk	
2. The clerk issues a request to add a new member	
	3. The system asks for data about the new member
4. The clerk enters the data into the system	
	5. Reads in data, and if the member can be added, generates an identification number (which is not necessarily a number in the literal sense just as social security numbers and phone numbers are not actually numbers) for the member and remembers information about the member. Informs the clerk if the member was added and outputs the member's name, address, phone and id
6. The clerk gives the user his identification number	

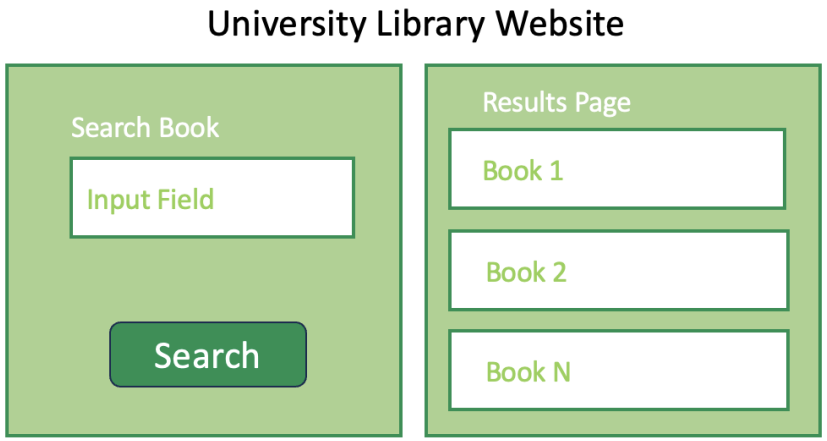
Use case Adding New Books

Actions performed by the actor	Responses from the system
1. Library receives a shipment of books from the publisher	
2. The clerk issues a request to add a new book	
	3. The system asks for the identifier, title, and author name of the book
4. The clerk generates the unique identifier, enters the identifier, title, and author name of a book	
	5. The system attempts to enter the information in the catalog and echoes to the clerk the title, author name, and id of the book. It then asks if the clerk wants to enter information about another book
6. The clerk answers in the affirmative or in the negative	
	7. If the answer is in the affirmative, the system goes to Step 3. Otherwise, it exits

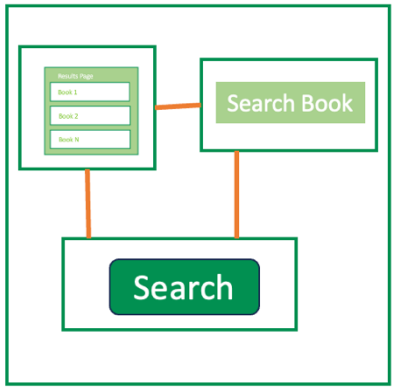


Use case diagram for the library system

Software Design - Identifying software classes



Conceptual Design



Components

Design

Goal

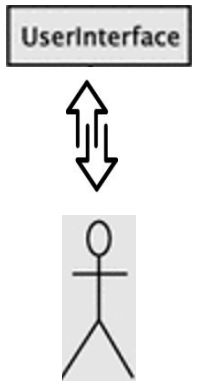
Classes
Functions
Components

Library Software Classes

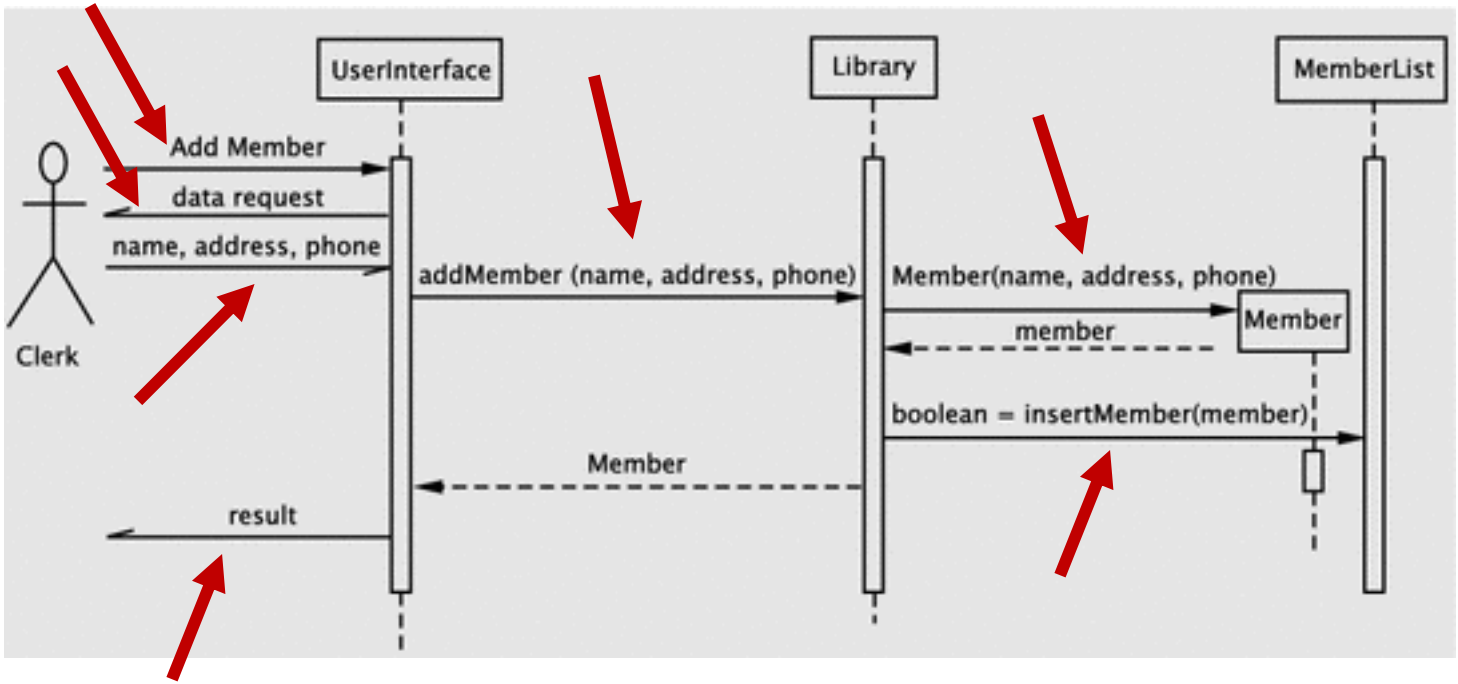
1. Member
2. Book
3. Library
4. Catalog
5. ….

Software Design - Assigning Responsibilities to Classes

Having decided on an adequate set of software classes, our next task is to assign responsibilities. The next step is, therefore, to spell out the details of how the system meets its responsibilities. Sequence Diagrams are a great UML tool for describing responsibilities of classes.

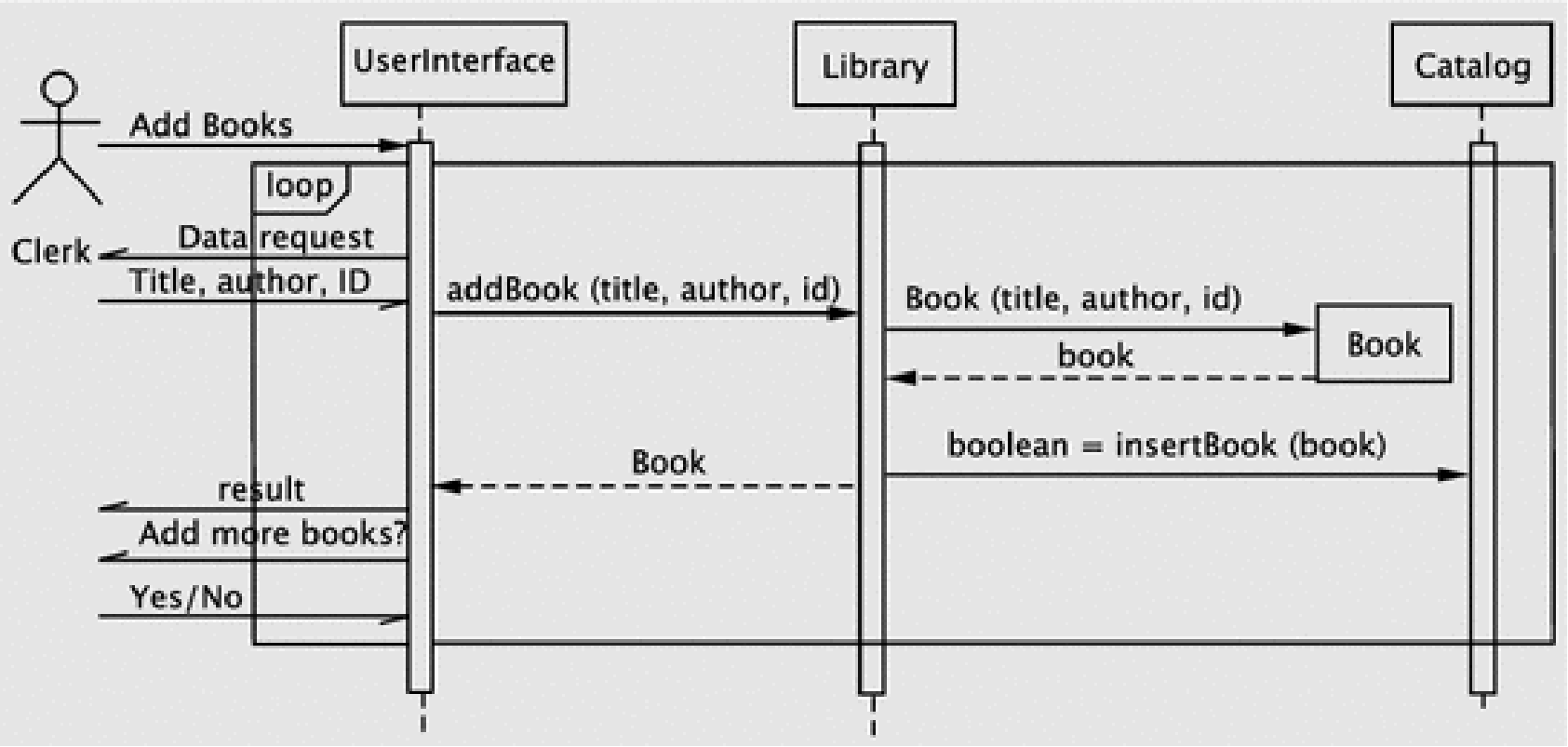


Sequence diagram for registering a new member



Software Design - Assigning Responsibilities to Classes

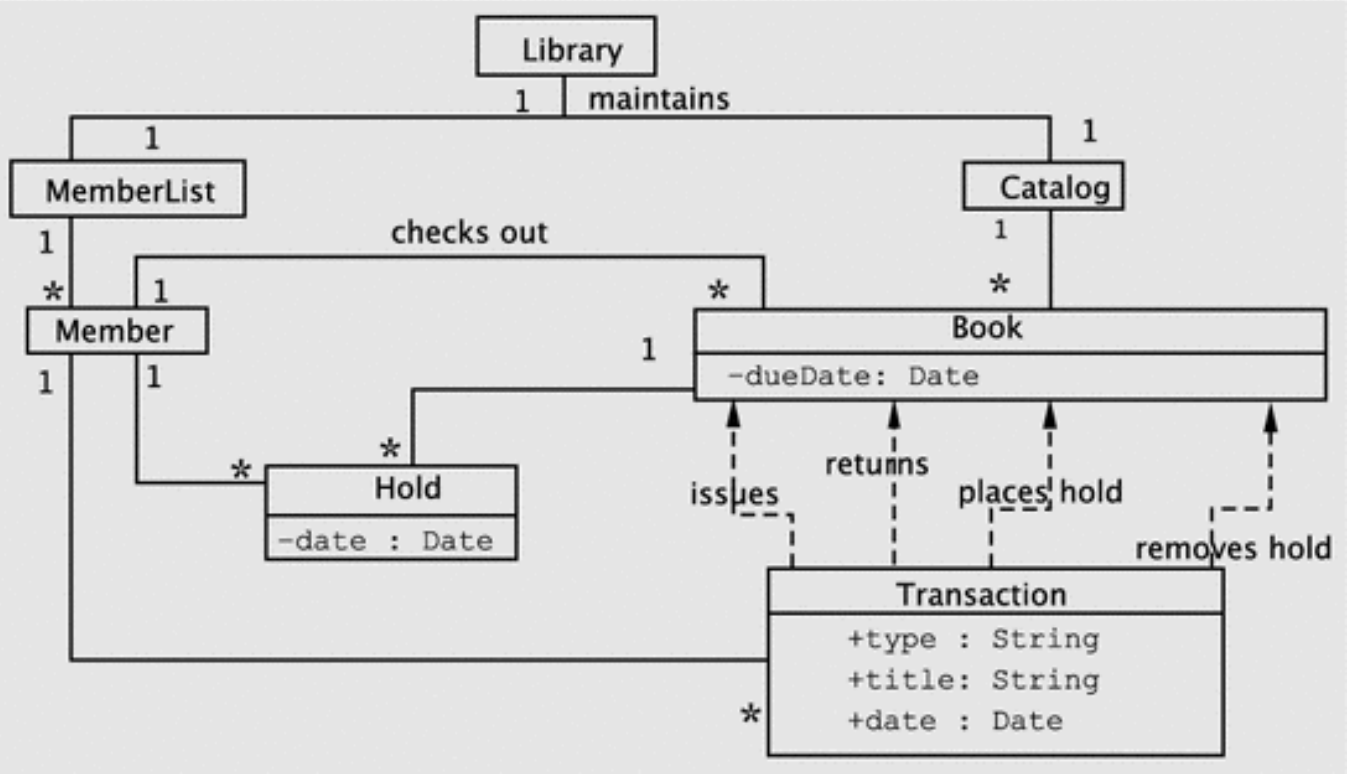
Sequence diagram for adding books



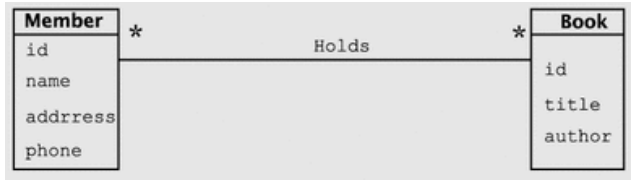
- In this case, when the request is made by the actor, the system enters a loop.
- Since the loop involves interacting repeatedly with the actor, the loop control mechanism is in the UI itself.

Class Diagrams – Relationships between classes

Library Software Classes
1. Library
2. MemberList
3. Catalog
4. Member
5. Book
6. Holds
7. Transaction
8. UserInterface

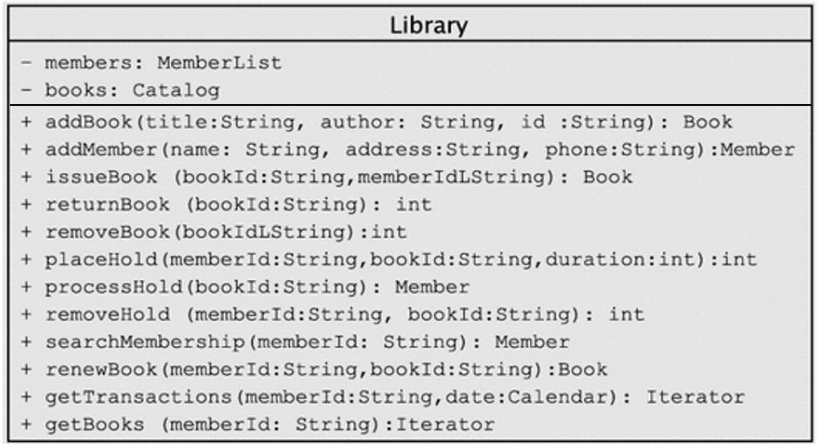


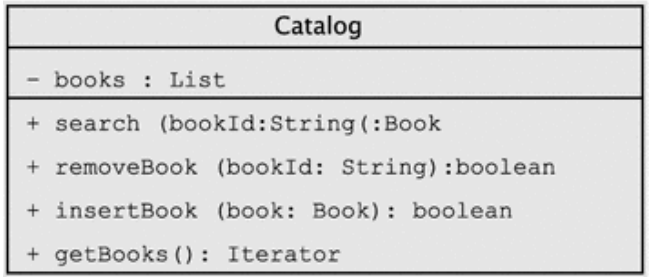
- Note that **Hold** is not shown as an association class, but an independent class that connects Member and Book.
- The new class **Transaction** is added to record transactions; this has a dependency on Book since it stores the title of the book.

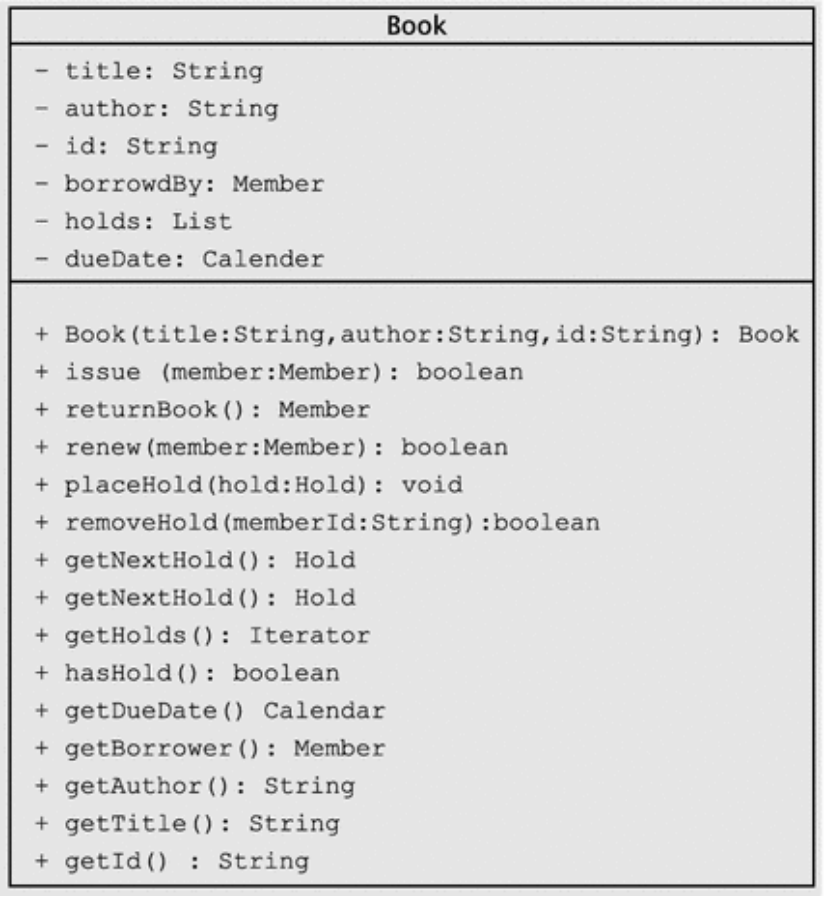


Class Diagrams Details

Library Software Classes
1. Library
2. MemberList
3. Catalog
4. Member
5. Book
6. Holds
7. Transaction

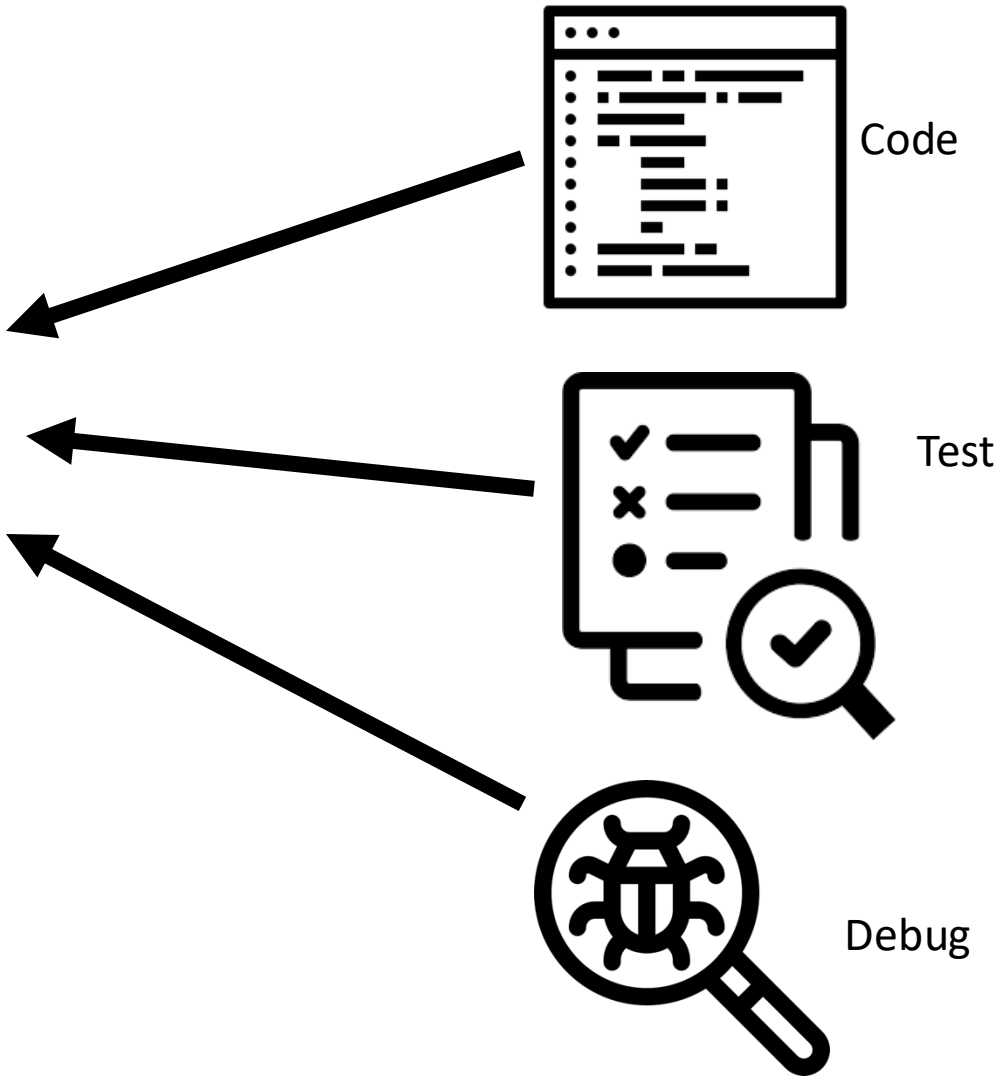






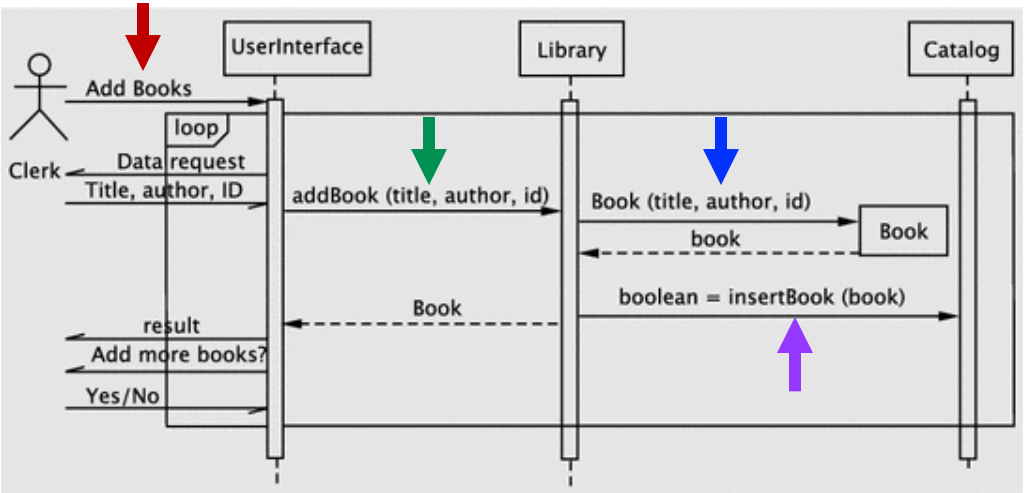
Implementing Our Design

Library Software Classes
1. Library
2. MemberList
3. Catalog
4. Member
5. Book
6. Holds
7. Transaction
8. UserInterface



Implementing Our Design

Adding New Books



addBooks method in the **UserInterface** class

```
public void addBooks() {
  Book result;
  do {
    String title = getToken("Enter book title");
    String author = getToken("Enter author");
    String bookID = getToken("Enter id");
    result = library.addBook(title, author, bookID);
    if (result != null) {
      System.out.println(result);
    } else {
      System.out.println("Book could not be added");
    }
    if (!yesOrNo("Add more books?")) {
      break;
    }
  } while (true);
}
```

addBook method inside the **Library** class

```
public Book addBook(String title, String author, String id) {
  Book book = new Book(title, author, id);
  if (catalog.insertBook(book)) {
    return (book);
  }
  return null;
}
```

insertBook method in the **Catalog** class

```
public class Catalog {
  private List books = new LinkedList();
  // some code not shown
  public boolean insertBook(Book book) {
    return books.add(book);
  }
}
```

Project

- For Design Portfolio II and III
- See - Instructions to the UML Diagrams and Design Process

- **UML Design Rubric (5 Points Total)**
  - [ ] 0 Points **(No Design Diagram)**: No design diagram is presented or linked in the PR.
  - [ ] 1 Point **(Minimal Design)**: A design diagram is presented, but it's incomplete, unclear, or lacks relevance to the feature/use case. Minimal effort shown.
  - [ ] 2 Points **(Basic Design)**: A design diagram is provided but may lack some key elements or clarity. It covers the basics but doesn't fully capture the interaction between components.
  - [ ] 3 Points **(Moderate Design)**: The design diagram is mostly complete and represents the interactions between components clearly. Shows a good understanding of the feature/use case but may be missing minor details.
  - [ ] 4 Points **(Strong Design)**: The design diagram is clear, detailed, and follows the guidelines well. It effectively captures the interactions between components and shows a solid understanding of the system.
  - [ ] 5 Points **(Exemplary Design)**: The design diagram is highly detailed, well-organized, and clearly linked in the PR. It provides a thorough representation of the system's interactions and reflects an excellent understanding of the feature/use case.