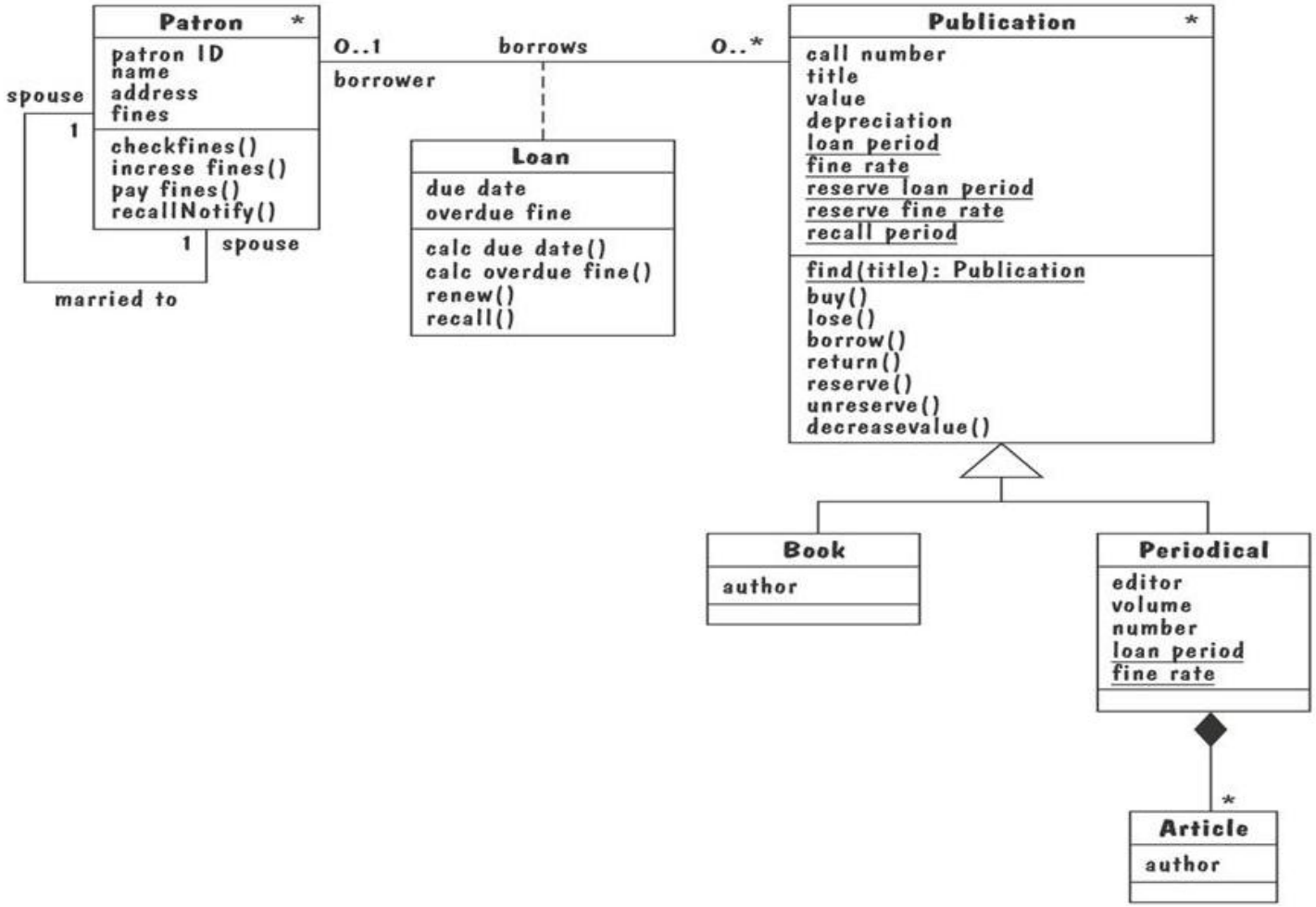
UML Modeling

Ashish Kumar Dwivedi

4.5 Modeling Notations

UML Class Diagram of Library Problem



4.5 Modeling Notations

UML Class Diagram (continued)

- Attributes and operations are associated with the class rather than instances of the class
- A **class-scope attribute** represented as an underlined attribute, is a data value that is shared by all instances of the class
- A **class-scope operation** written as underlined operation, is an operation performed by the abstract class rather than by class instances
- An **association**, marked as a line between two classes, indicates a relationship between classes' entities

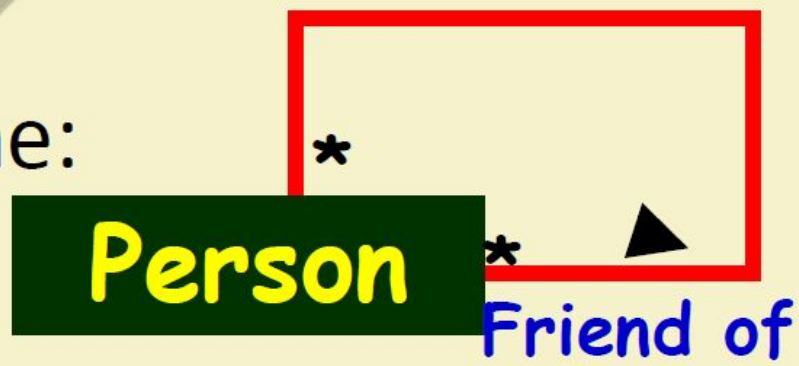
4.5 Modeling Notations

UML Class Diagram (continued)

- **Aggregate association** is an association that represents interaction, or events that involve objects in the associated (marked with white diamond)
  - "*has-a*" relationship
- **Composition association** is a special type of aggregation, in which instances of the compound class are physically constructed from instances of component classes (marked with black diamond)

Association Relationship

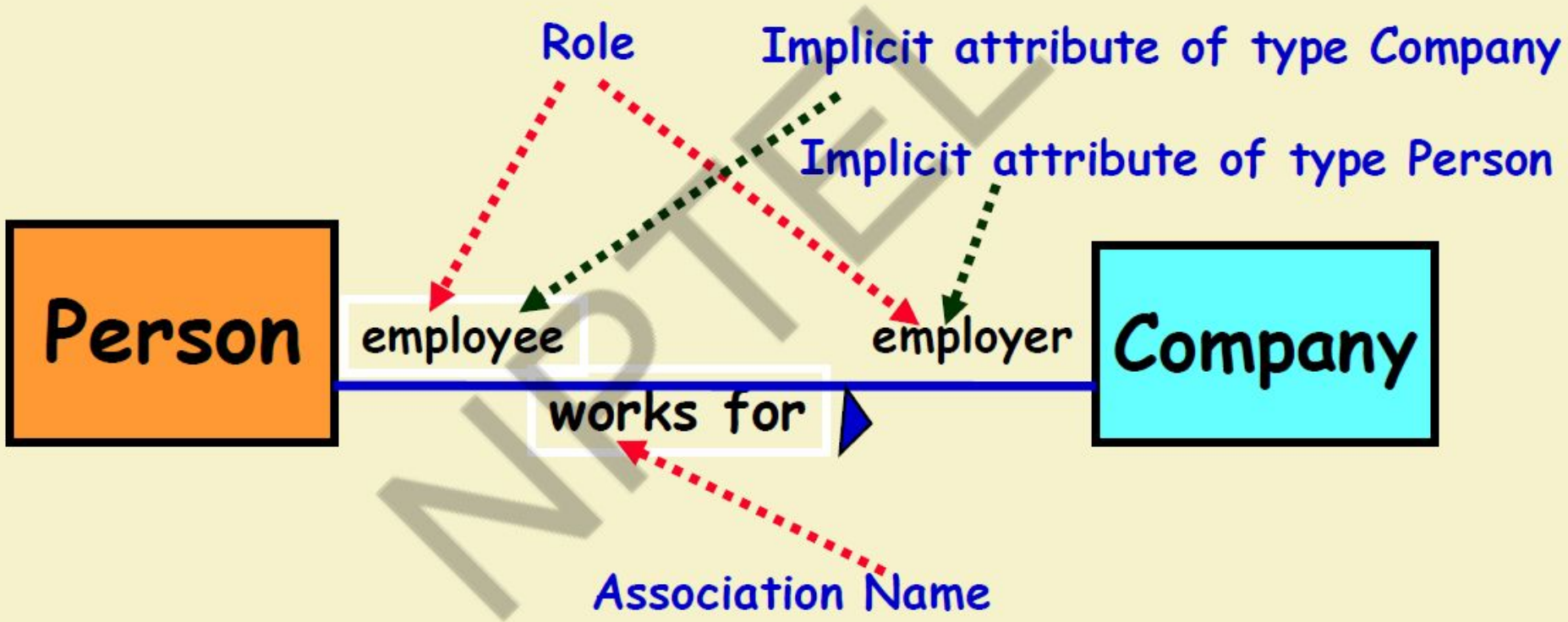
- A class can be associated with itself (unary association).
  - **Give an example?**
- An arrowhead used along with name:
  - Indicates direction of association.



- Multiplicity indicates # of instances taking part in the association.

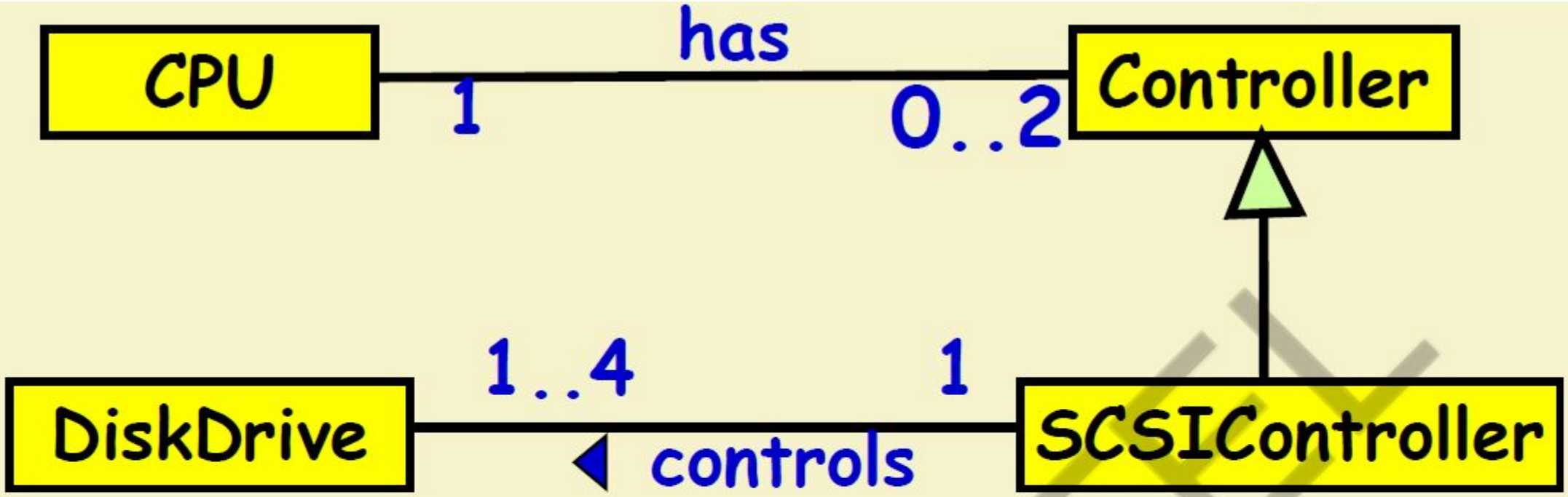
Association Exercise 1

- A Person works for a Company.



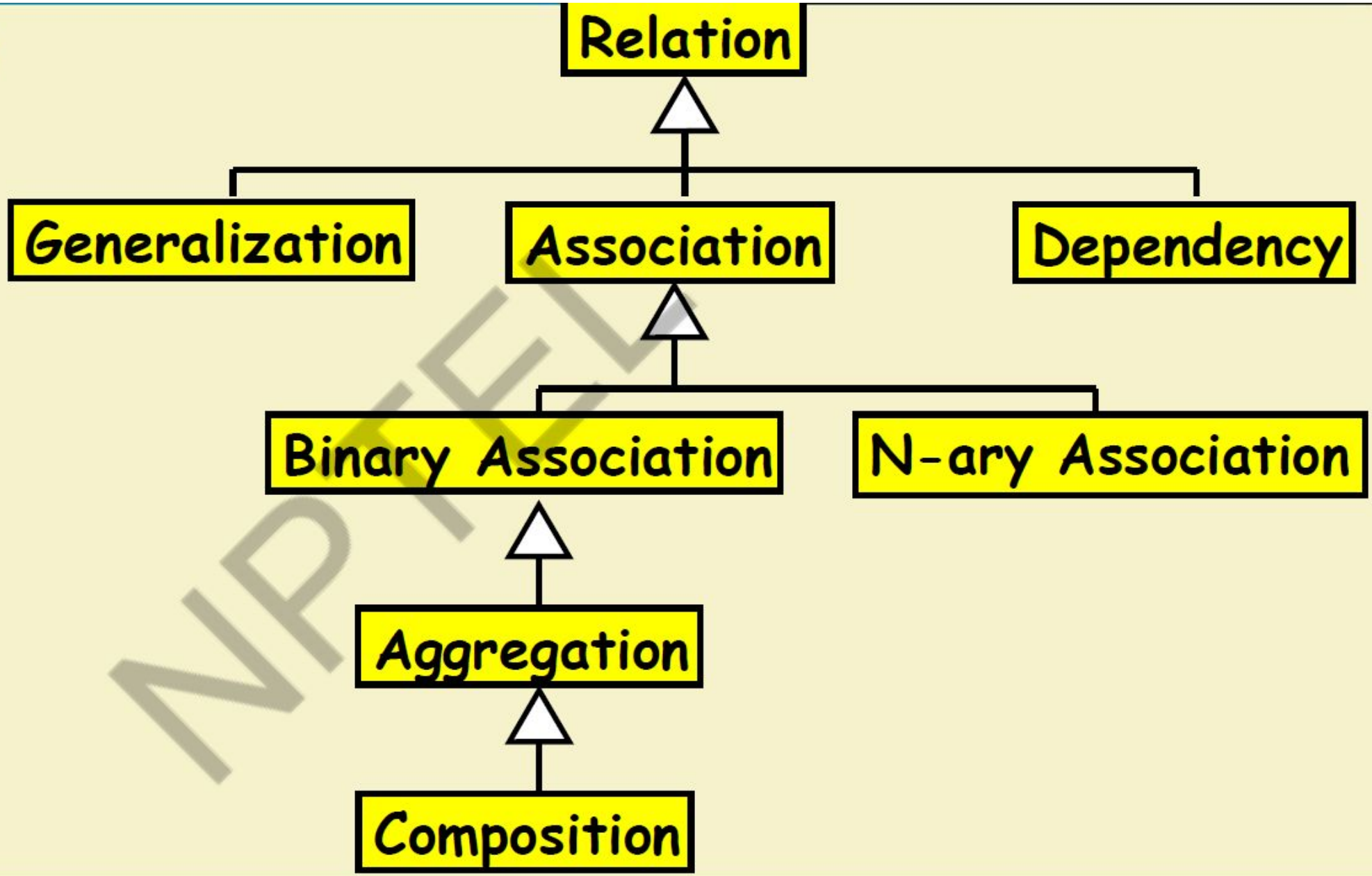
Observe: Implicit bidirectional navigation

Implementation?



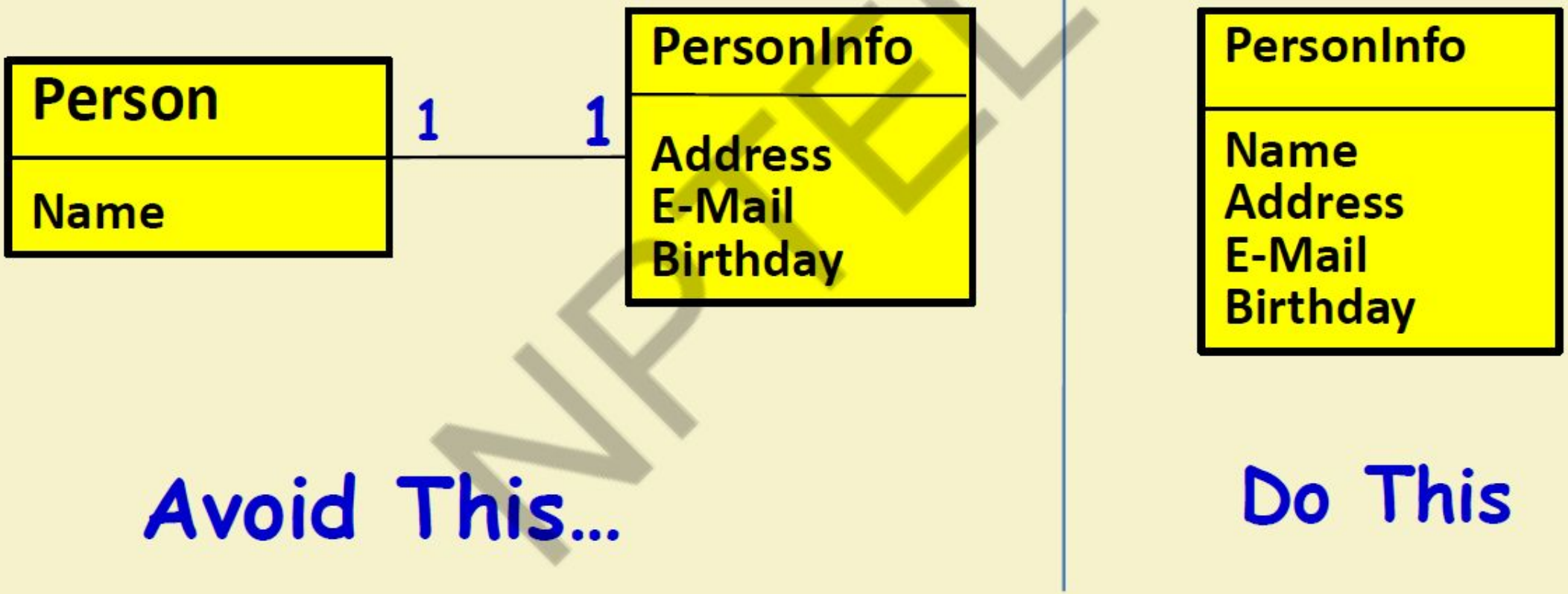
- 1 CPU has 0 to two Controllers
- 1-4 DiskDrives controlled by 1 SCSIController
- SCSIController is a (specialized) Controller

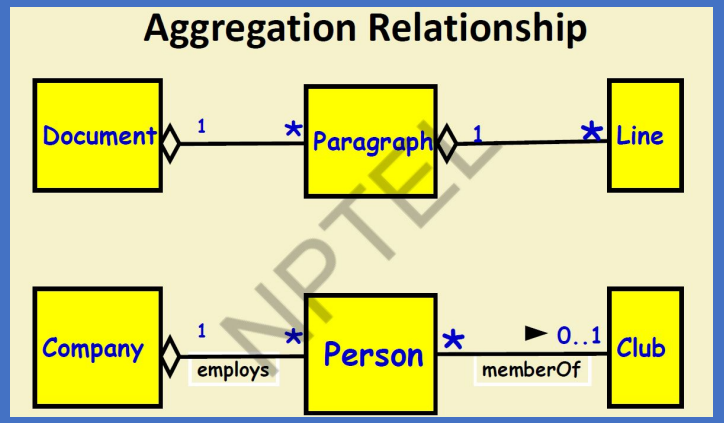
Types of Class Relationships



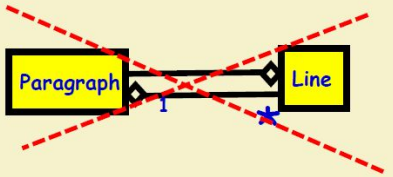
Overdoing Associations

Avoid unnecessary Associations





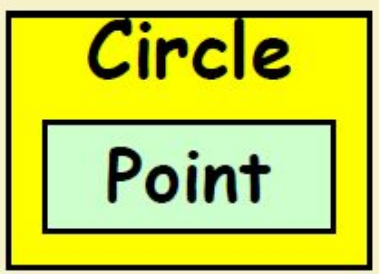
- An aggregate object contains other objects.
- Aggregation limited to tree hierarchy:
  - **No circular aggregate relation.**



Composition

A stronger form of aggregation

- The whole is the sole owner of its part.
  - A component can belong to only one whole
- The life time of the part is dependent upon the whole.
  - **The composite must manage the creation and destruction of its parts.**

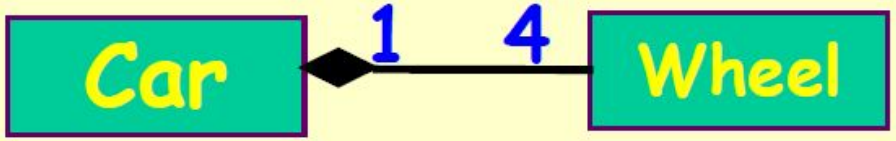


Aggregation vs. Composition

- Composition:
  - Composite and components have the same life line.
- Aggregation:
  - Lifelines are different.
- Consider an order object:
  - **Aggregation**: If order items can be changed or deleted after placing order.
  - **Composition**: Otherwise.

Implementing Composition

```
public class Car{
    private Wheel wheels[4];
    public Car (){
            wheels[0] = new Wheel();
            wheels[1] = new Wheel();
            wheels[2] = new Wheel();
            wheels[3] = new Wheel();
    }
}
```

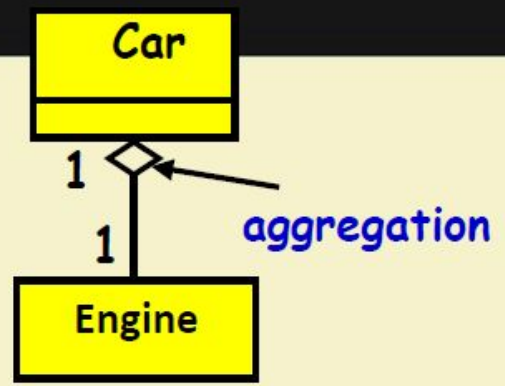


- Dependency relationship can arise due to a variety of reasons:
  - **Stereotypes are used to show the precise nature of the dependency.**

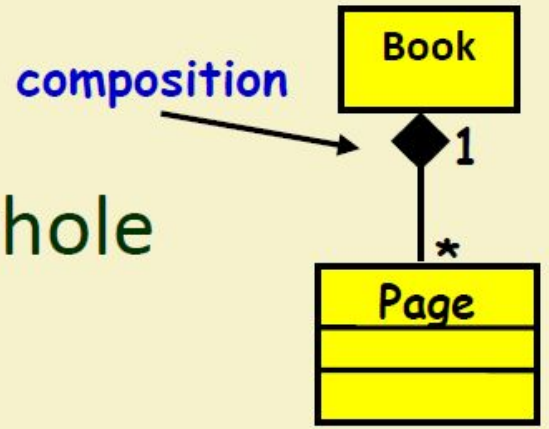
Type of dependency	Stereotype	Description
Abstraction	«abstraction»	Dependency of concrete class on its abstract class.
Binding	«bind»	Binds template arguments to create model elements from templates.
Realization	«realize»	Indicates that the client model element is an implementation of the supplier model element
Substitution	«substitute»	Indicates that the client model element takes place of the supplier.

Association Types

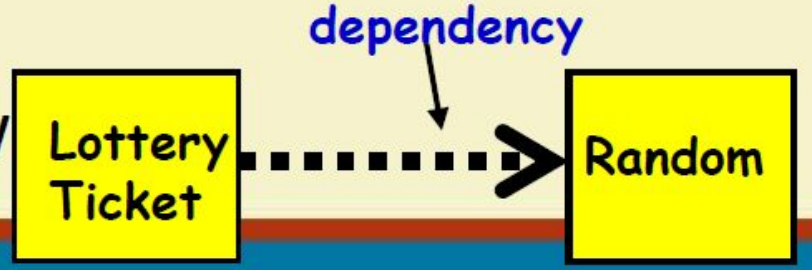
- **aggregation: "is part of"**
  - Symbolized by empty diamond



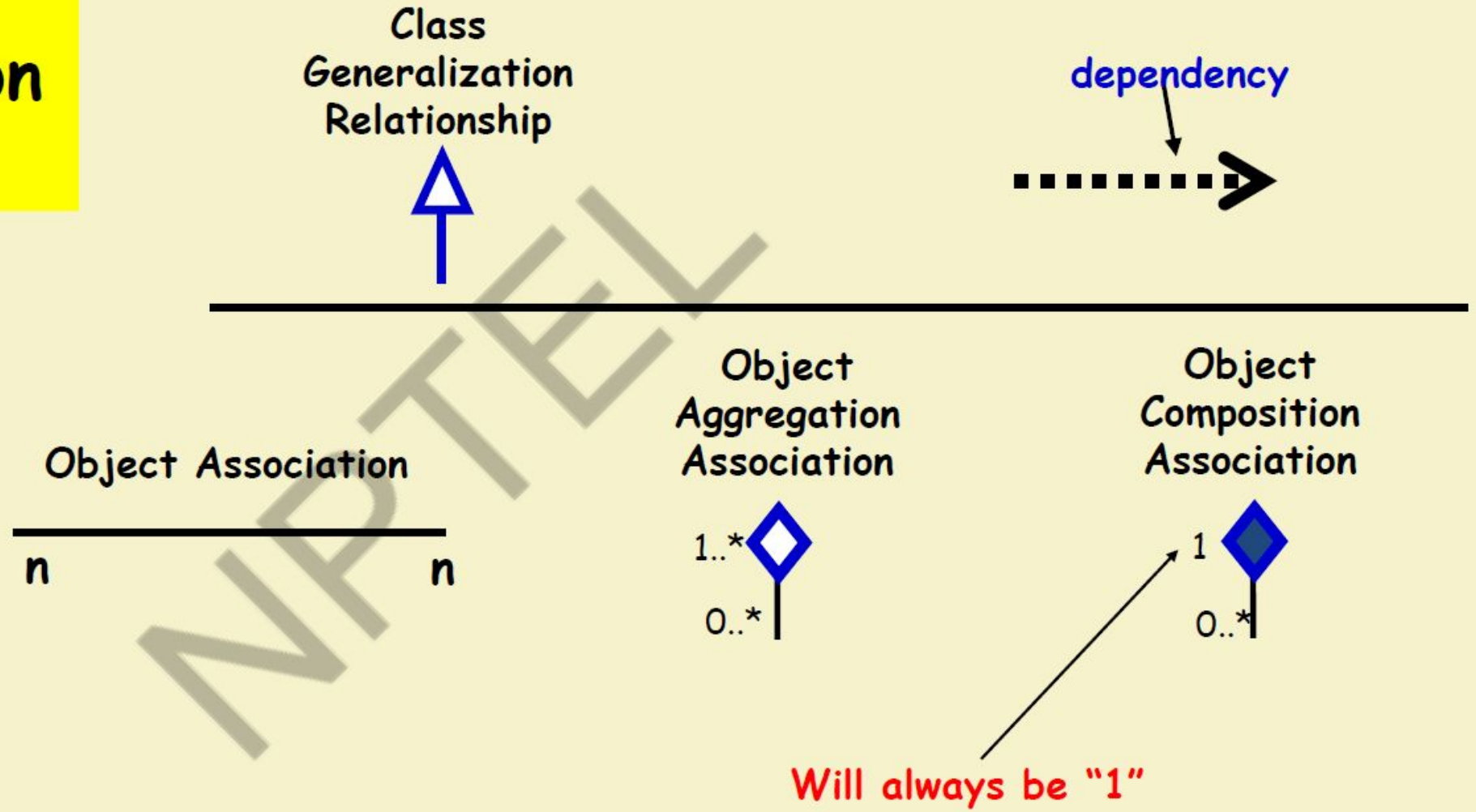
- **composition: is made of**
  - Stronger version of aggregation
  - The parts live and die with the whole
  - Symbolized by a filled diamond



- **dependency: Depends on**
  - Represented by dotted arrow

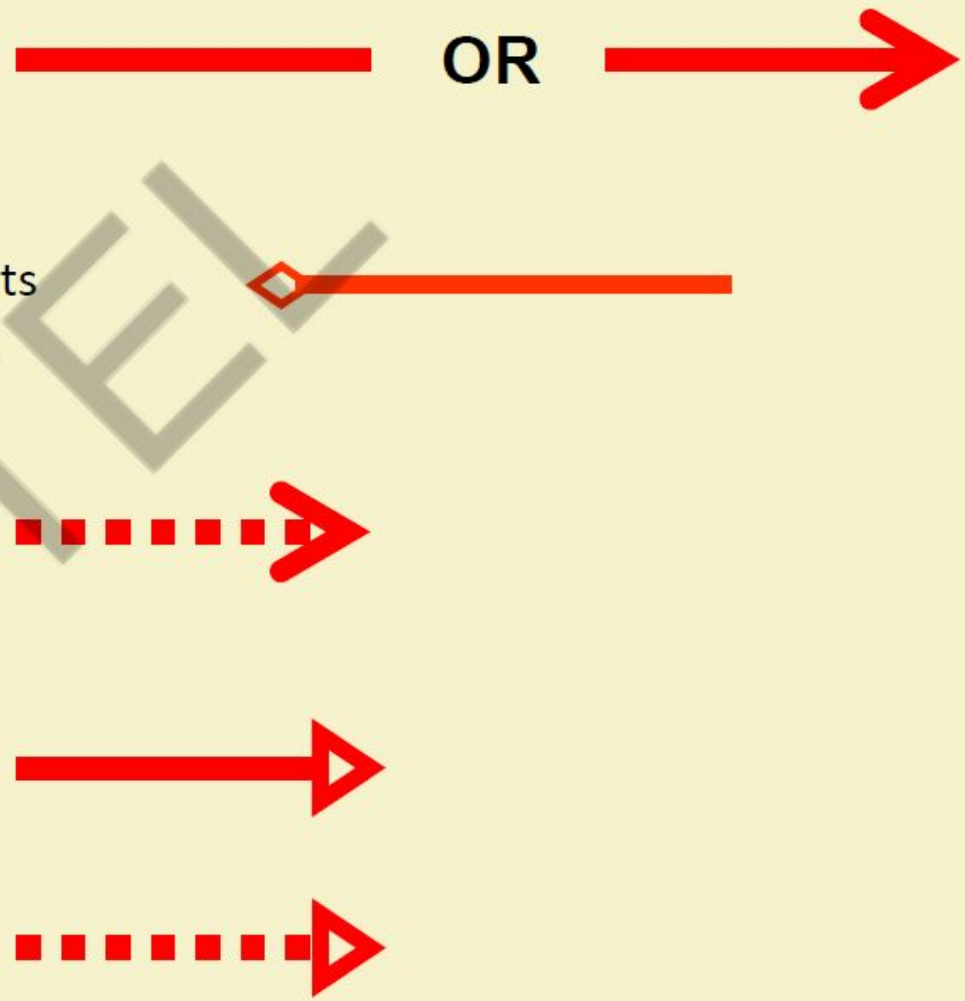


UML Class Relation Notation Summary



Summary of Relationships Between Classes

- **Association**
  - Permanent, structural, "has a"
  - Solid line (arrowhead optional)
- **Aggregation**
  - Permanent, structural, a whole created from parts
  - Solid line with diamond from whole
- **Dependency**
  - Temporary, "uses a"
  - Dotted line with arrowhead
- **Generalization**
  - Inheritance, "is a"
  - Solid line with open (triangular) arrowhead
- **Implementation**
  - Dotted line with open (triangular) arrowhead



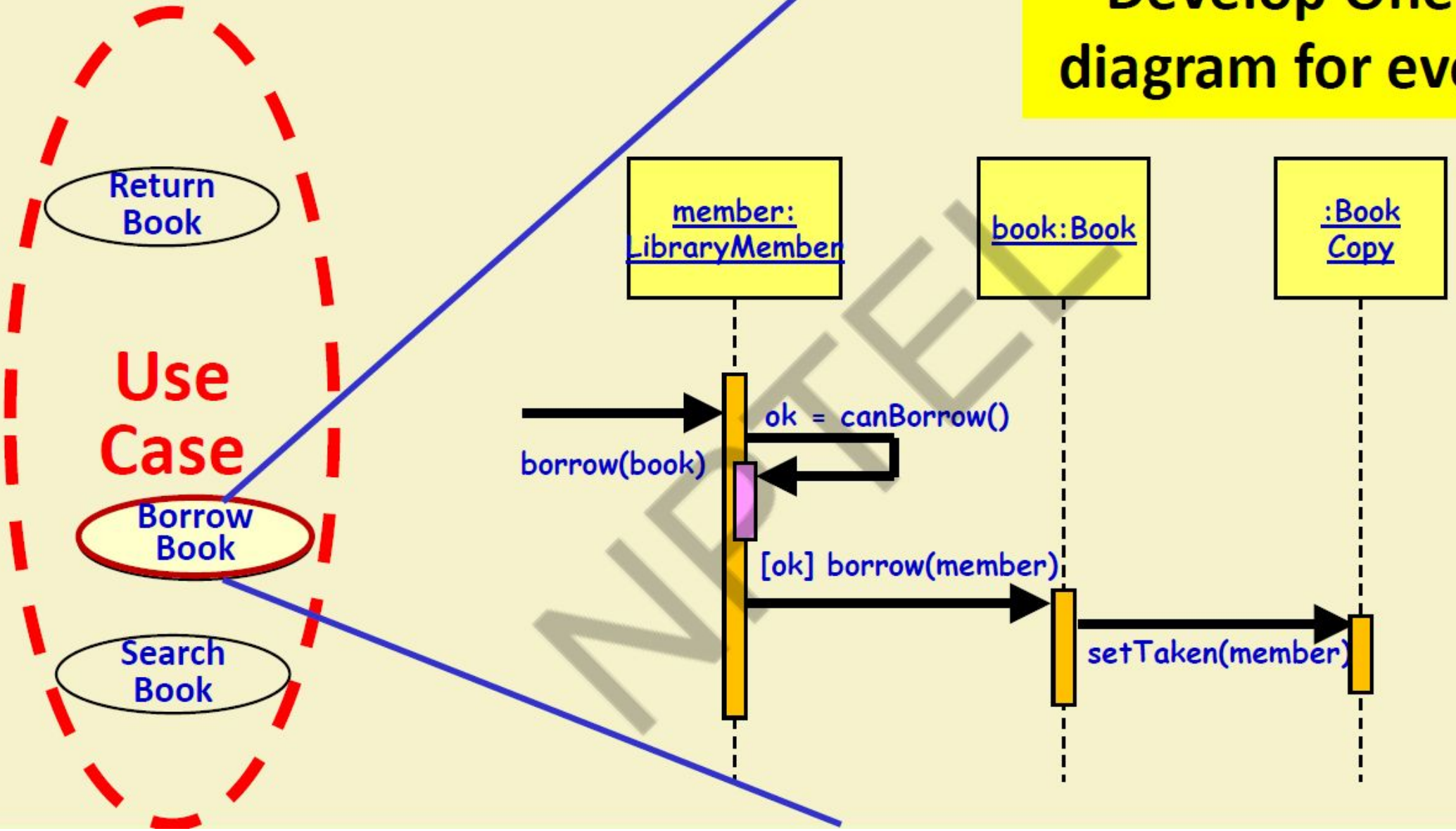
Quiz: Draw Class Diagram

- A Student can take up to five Courses.
- A student needs to enroll in at least one course.
- Up to 300 students can enroll in a course.
- An offered subject in a semester should have at least 10 registered students.

Interaction Diagram

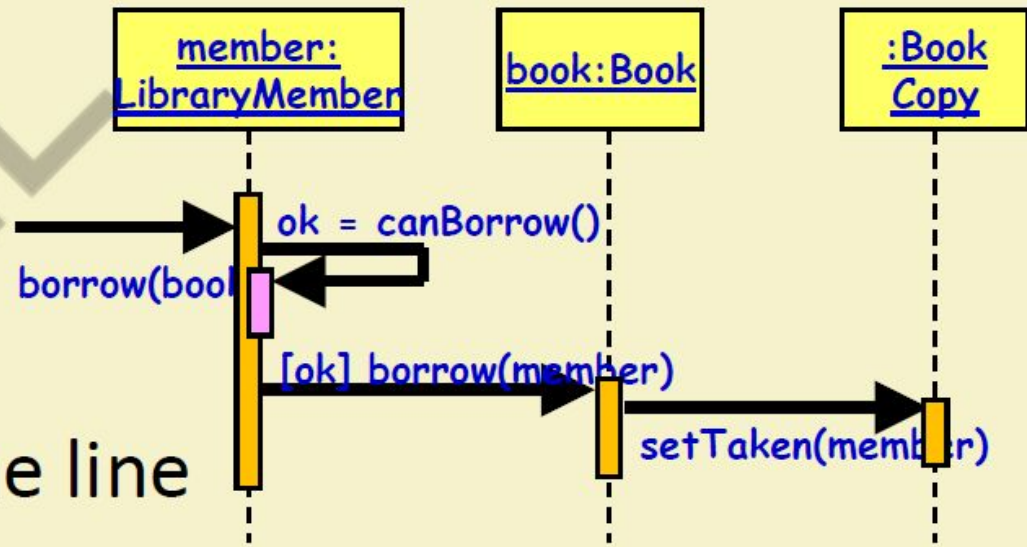
- **Can model the way a group of objects interact to realize some behaviour.**
- How many interaction diagrams to draw?
  - Typically each interaction diagram realizes behaviour of a single use case
  - **Draw one sequence diagram for each use case.**

Develop One Sequence diagram for every use case



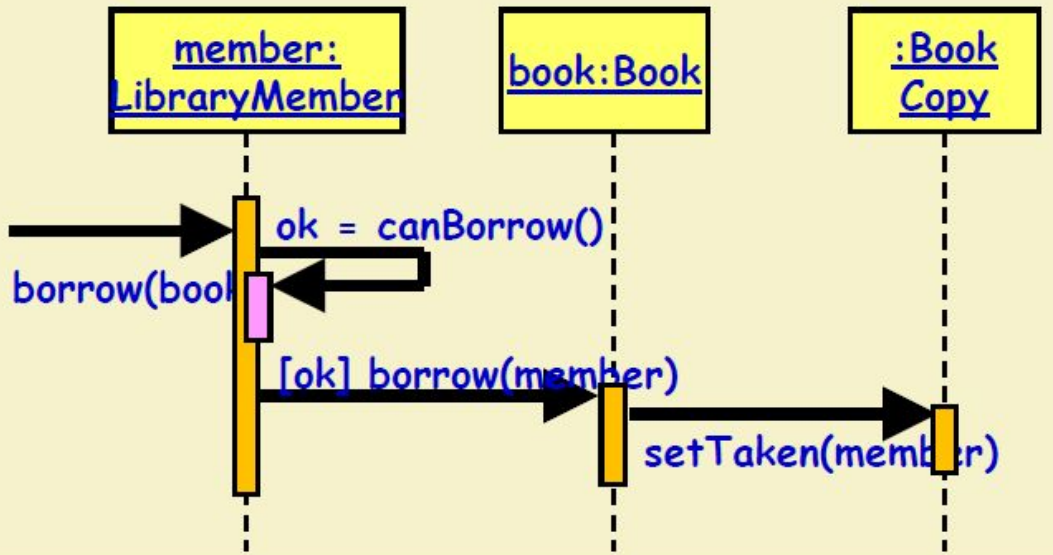
Sequence Diagram

- Shows interaction among objects in a two-dimensional chart
- Objects are shown as boxes at top
- If object created during execution:
  - Then shown at appropriate place in time line
- Object existence is shown as dashed lines (lifeline)
- Object activeness, shown as a rectangle on lifeline



Sequence Diagram Cont…

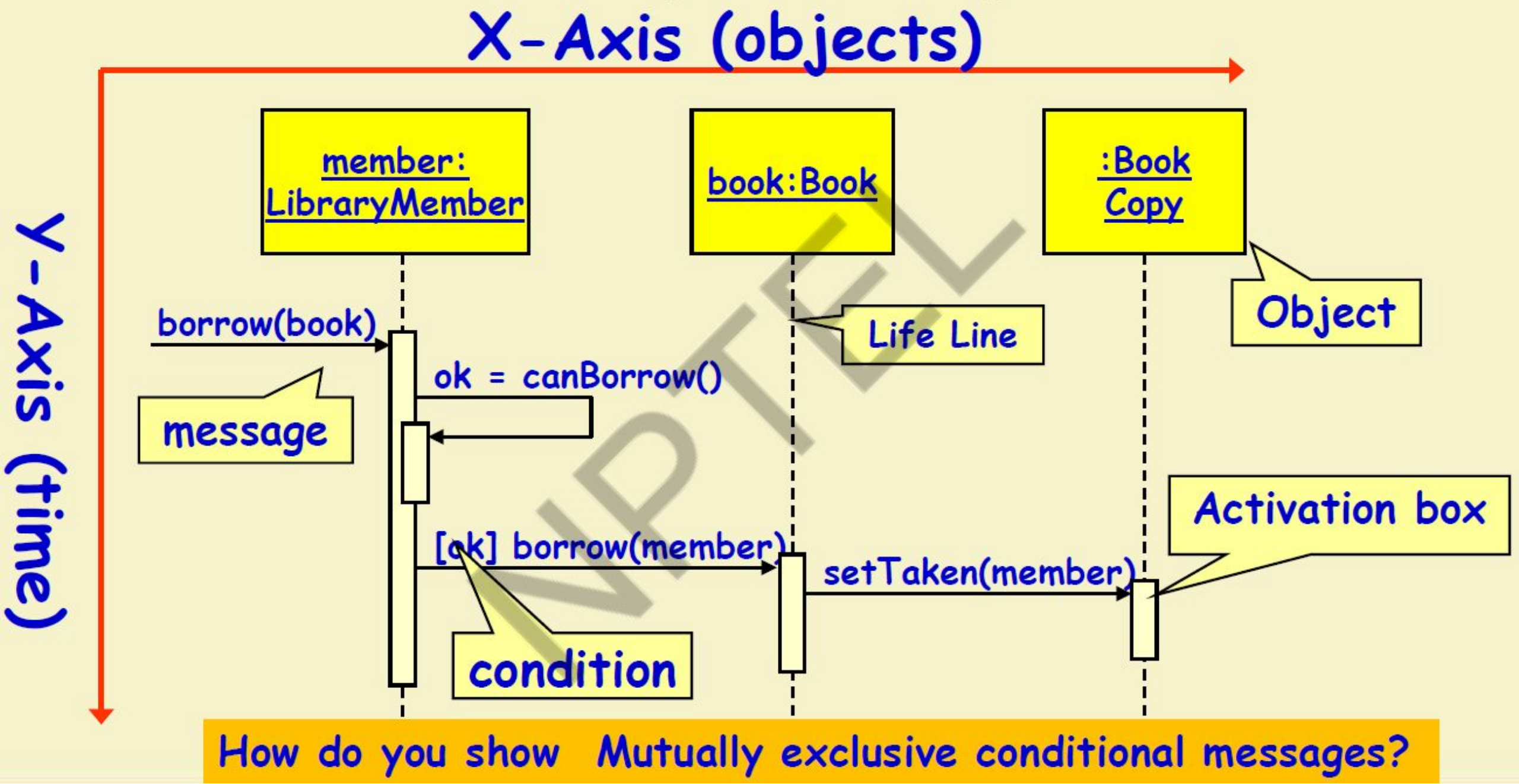
- Messages are shown as arrows.
- Each message labelled with corresponding message name.
- Each message can be labelled with some control information.
- Two types of control information:
  - –condition ([])
  - –iteration (*)

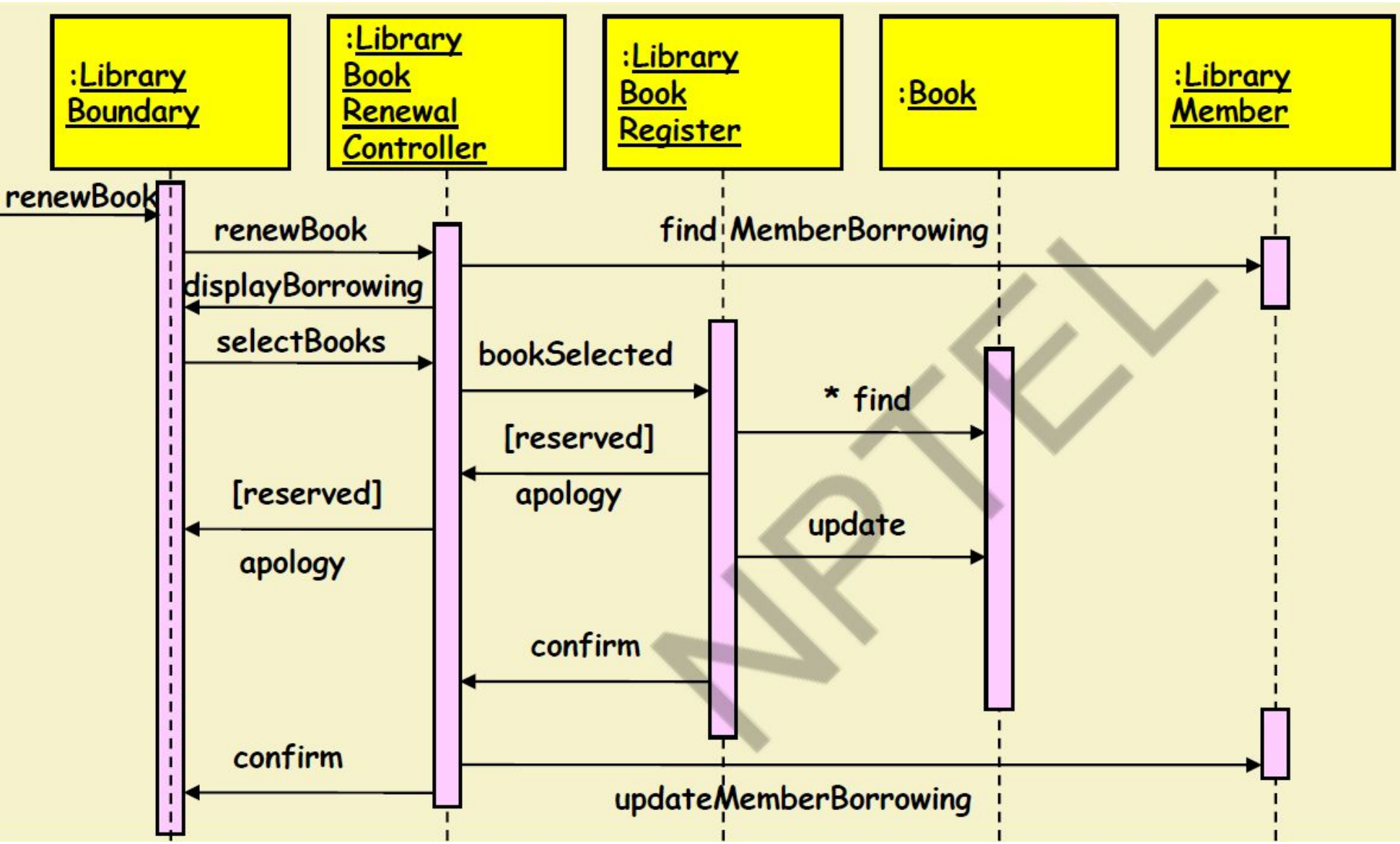


Control logic in Interaction Diagrams

- **Conditional Message**
  - [variable = value] message()
  - Message is sent only if clause evaluates to *true*
- **Iteration (Looping)**
  - * [i := 1..N] message()
  - "*" is required; [...] clause is optional
  - **The message is sent many times to possibly multiple receiver objects.**

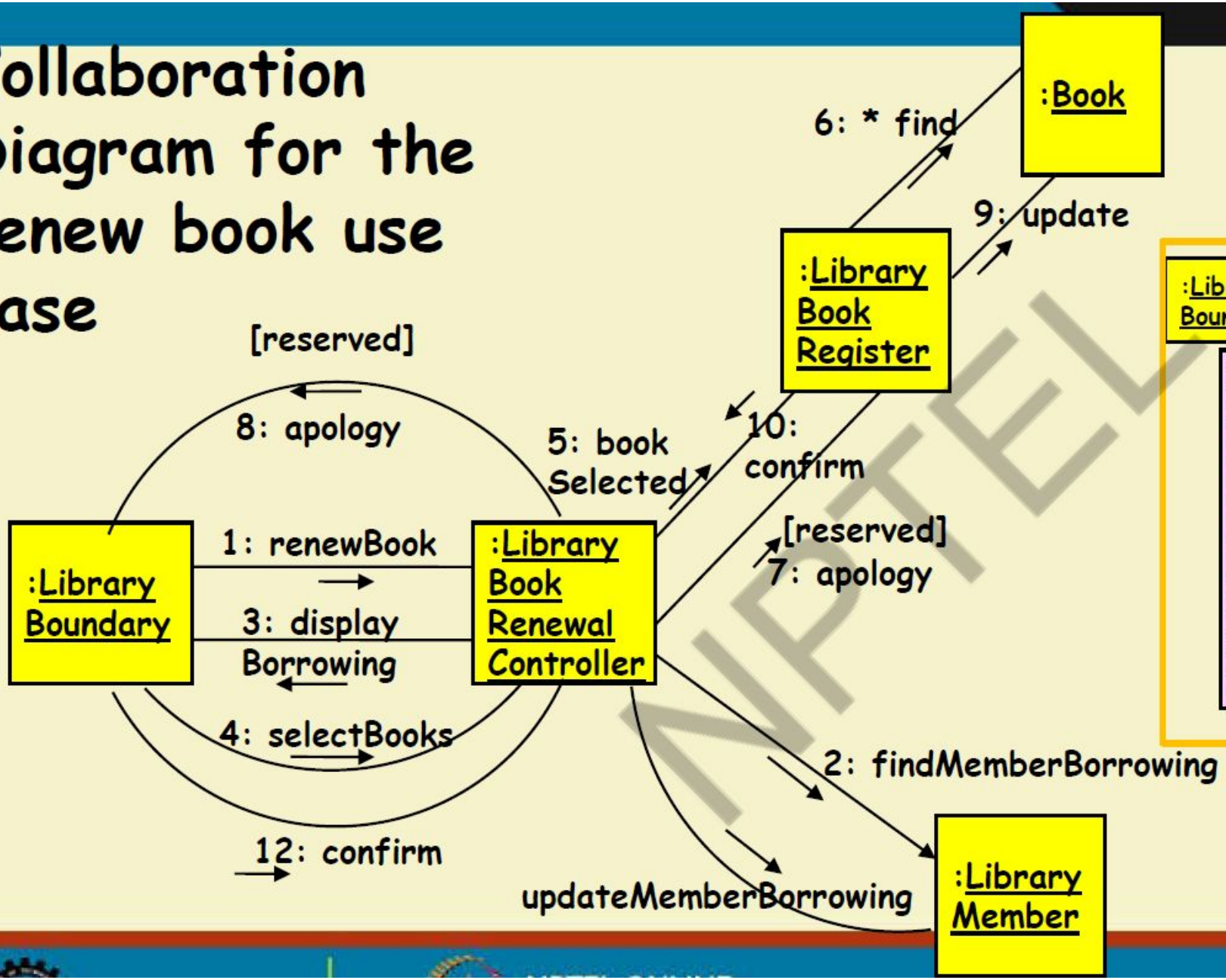
Elements of A Sequence Diagram

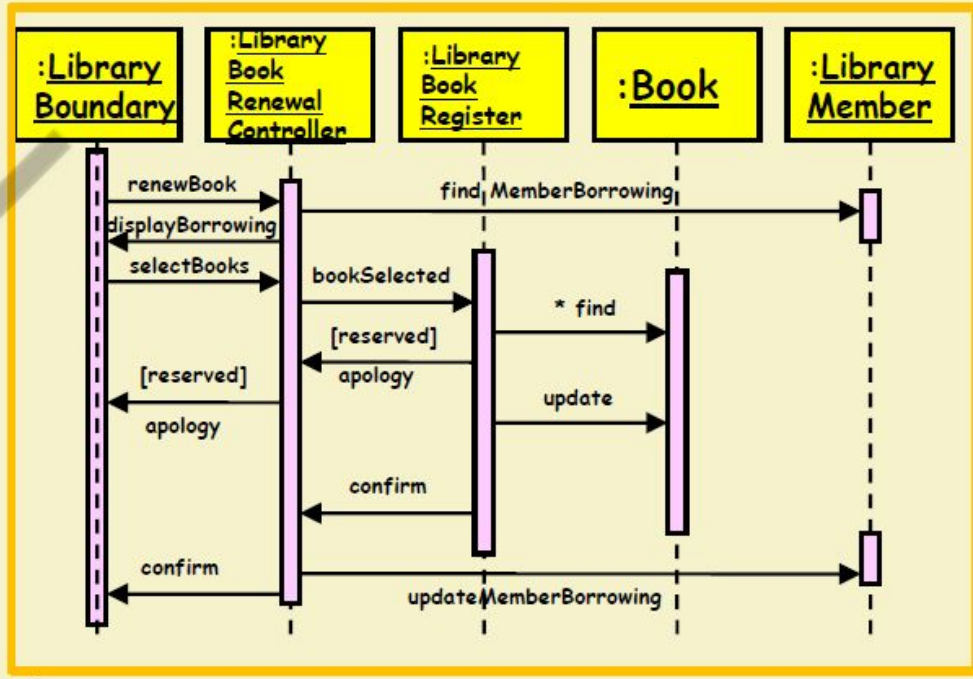




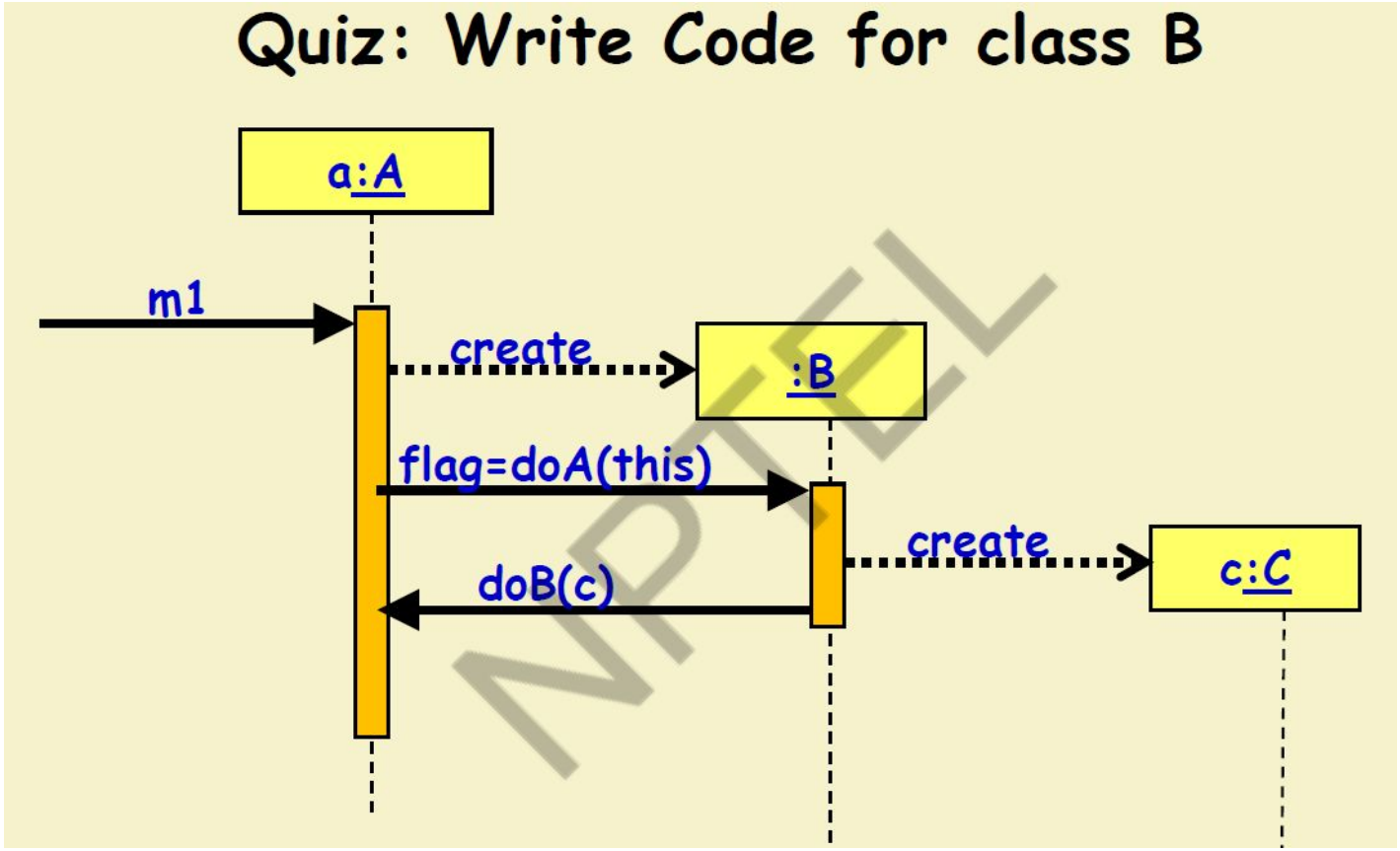
Sequence Diagram for the renew book use case

Collaboration Diagram for the renew book use case





Quiz: Write Code for class B



Quiz: Ans

```
public class B {
…
int doA(A a) {
      int flag;
      C c = new C();
      a.doB(c); …
      return flag; }
}
```

State Chart Diagram

- Elements of state chart diagram
- **Initial State:** A filled circle
- **Final State:** A filled circle inside a larger circle
- **State:** Rectangle with rounded corners
- **Transitions:** Arrow between states, also boolean logic condition (guard)
- UML extended state chart is called state machine

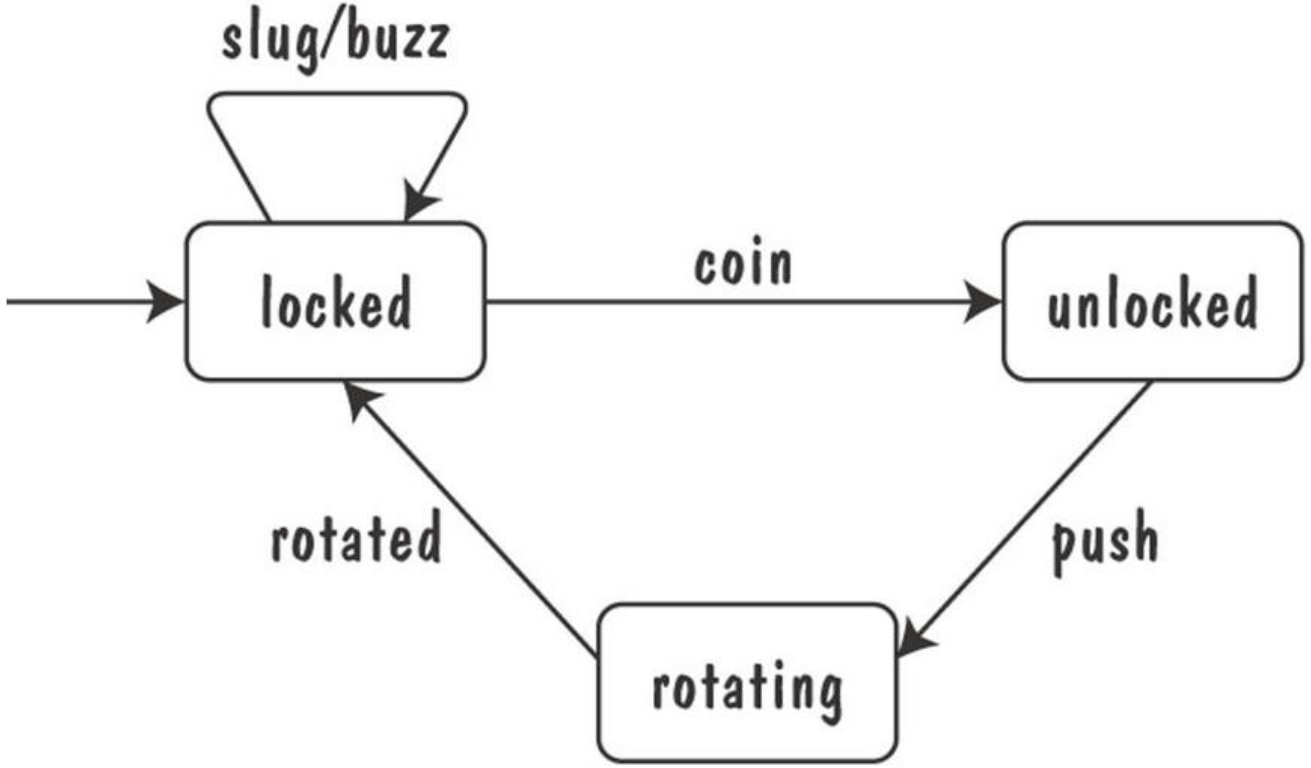
State Chart Diagram Cont...

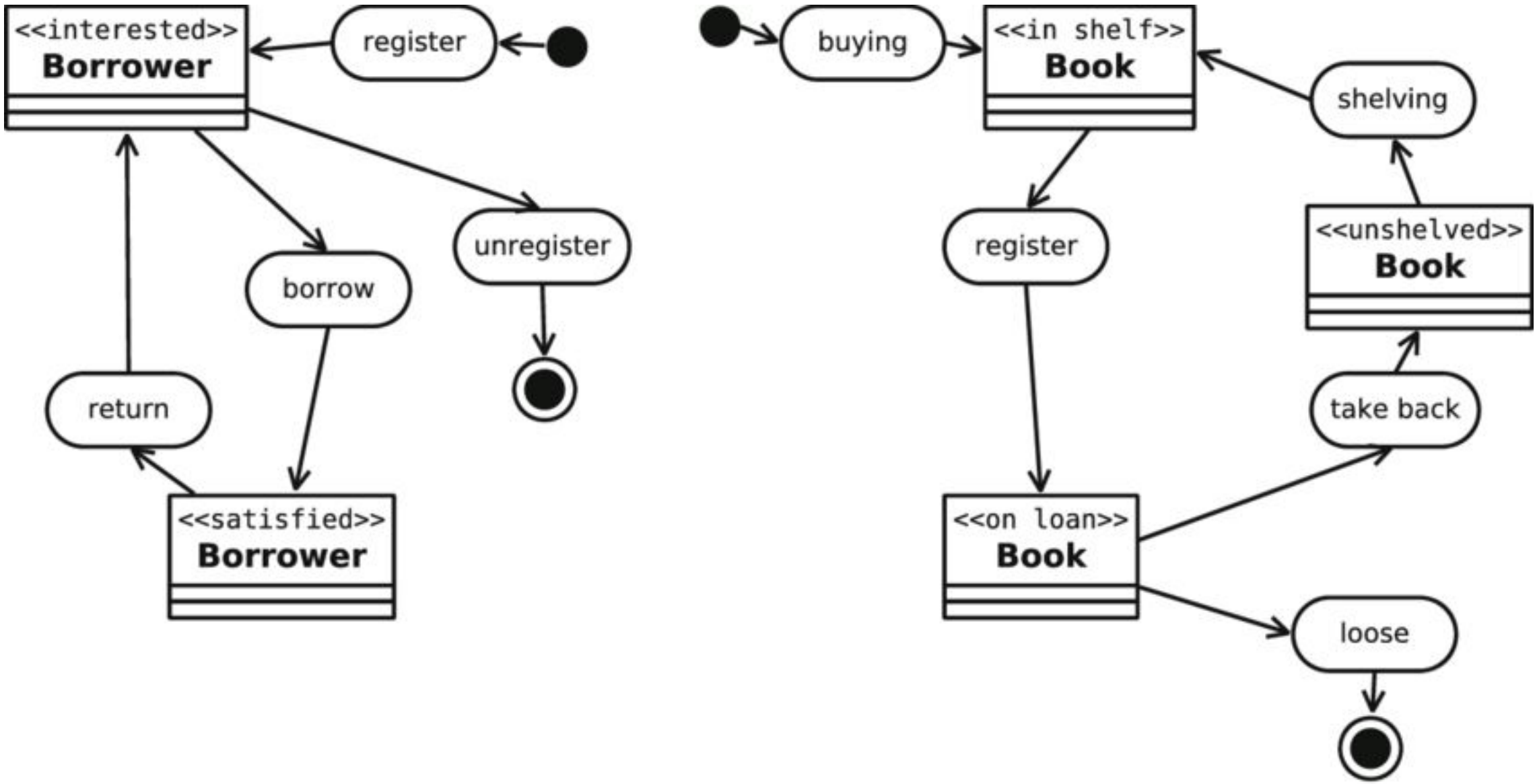
- State chart avoids two problems of FSM:
  - State explosion
  - Lack of support for representing concurrency
- A hierarchical state model:
  - Can have **composite states** --- OR and AND states.

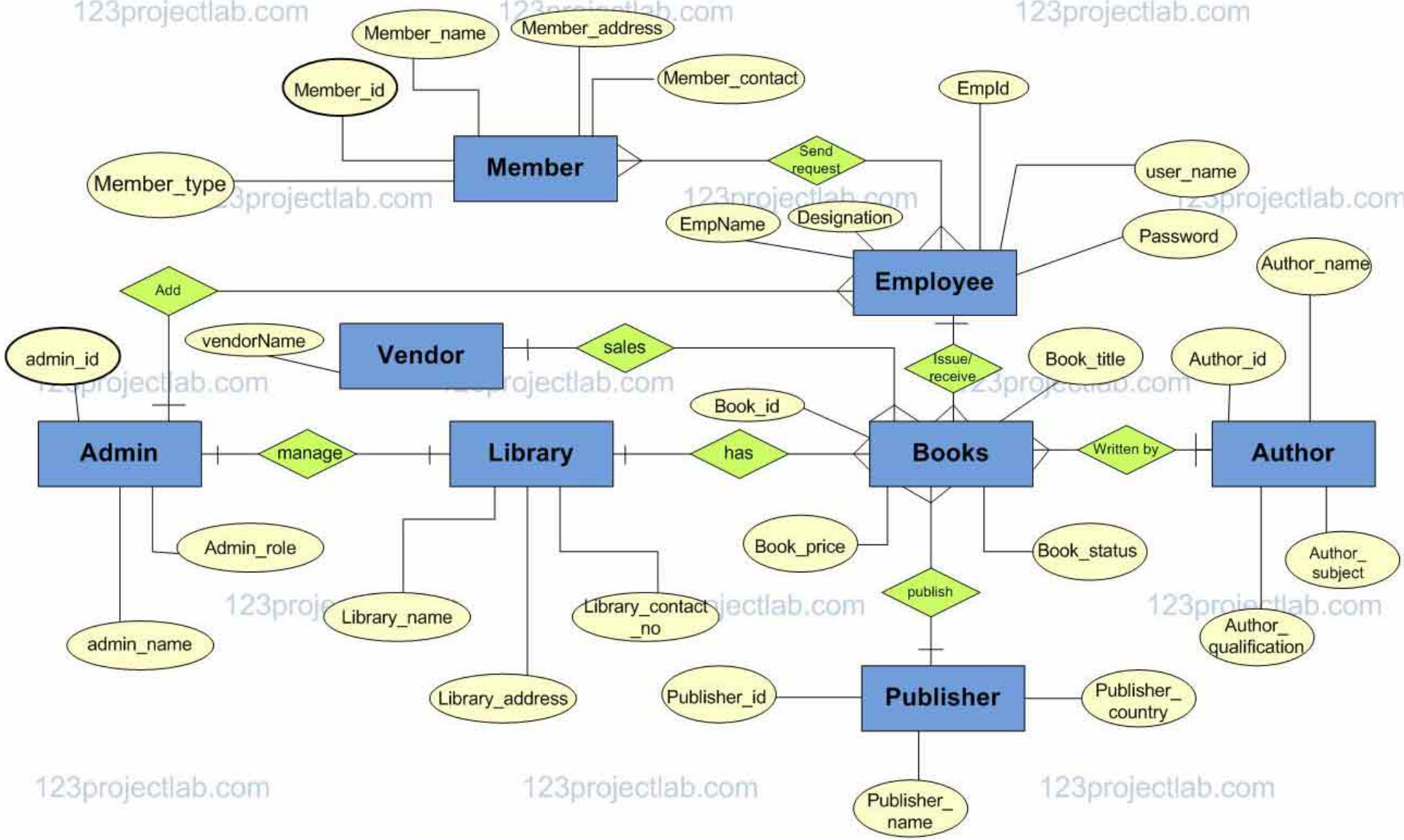
4.5 Modeling Notations

State Machines (continued)

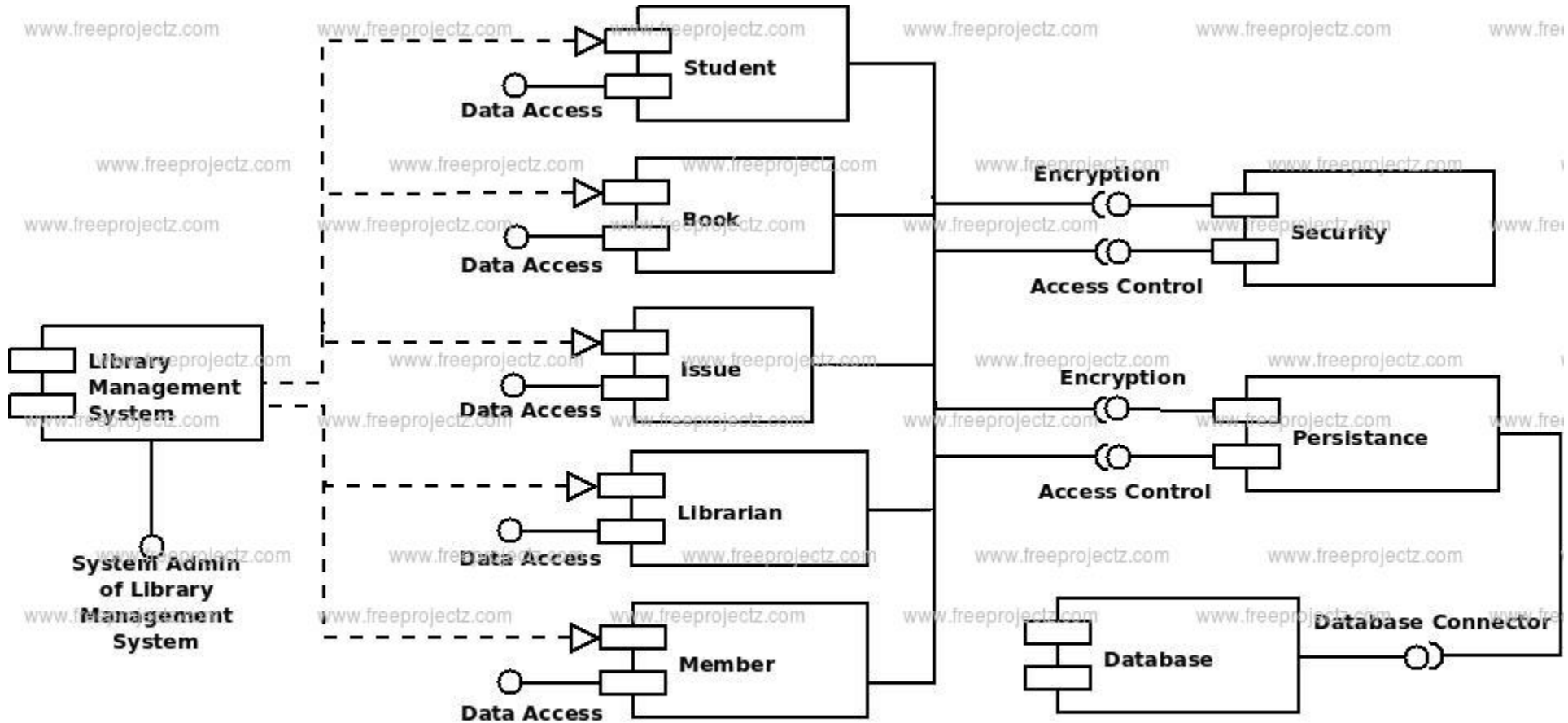
- Finite state machine model of the tunstile problem




<<interested>>
Borrower
register
unregister
borrow
return
<<satisfied>>
Borrower
buying
<<in shelf>>
Book
shelving
register
<<unshelved>>
Book
take back
<<on loan>>
Book
loose

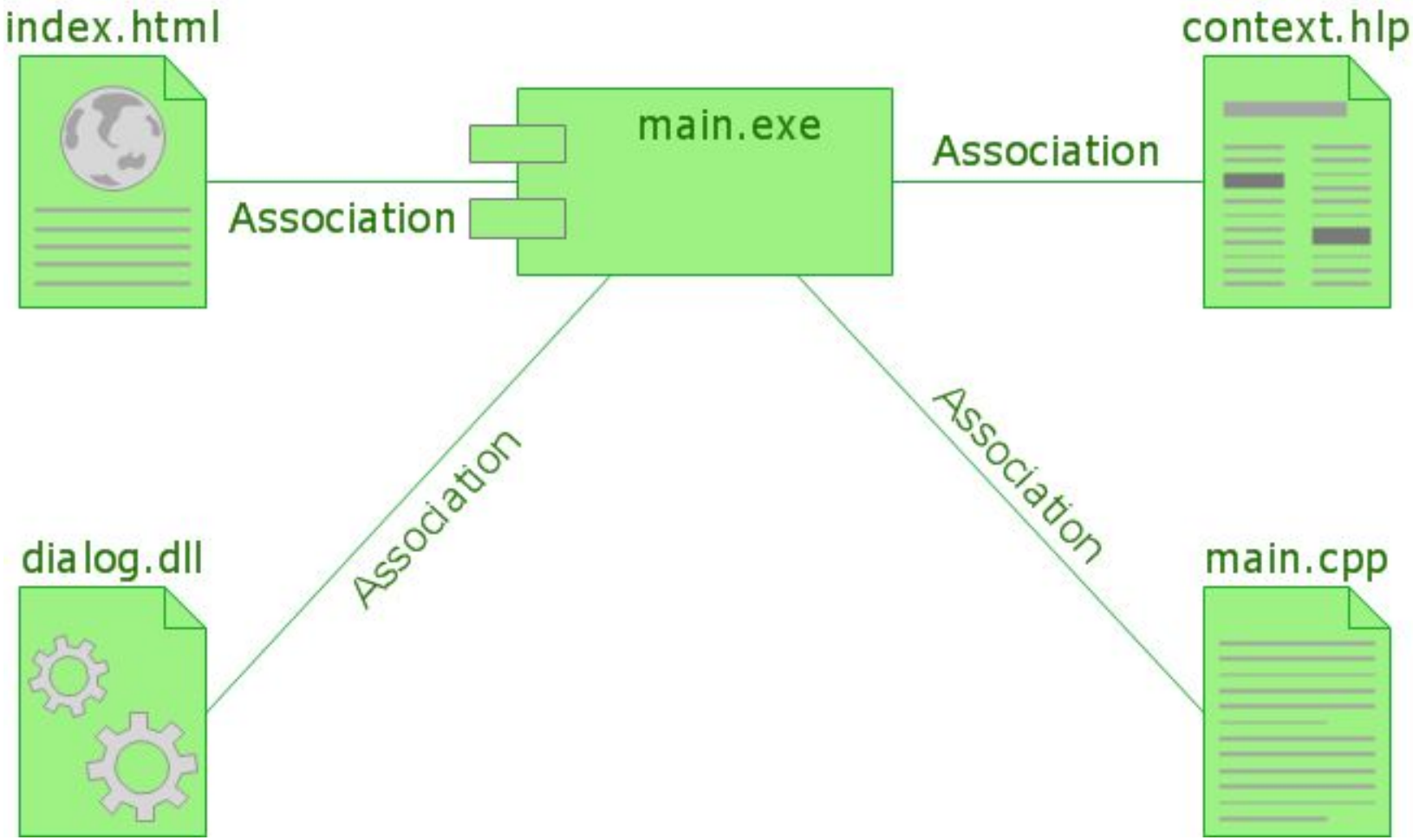


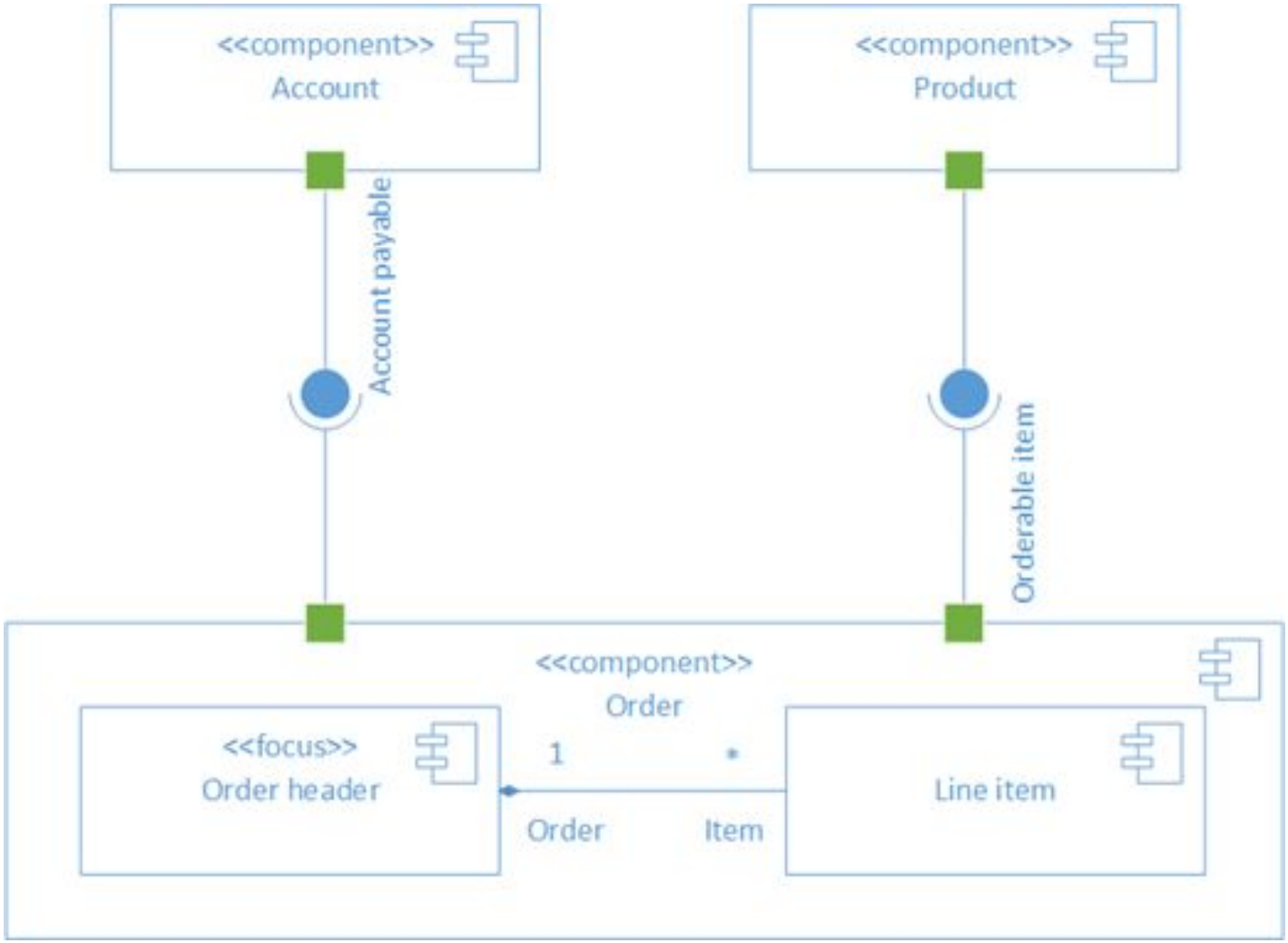
ER-Diagram for Library Management System

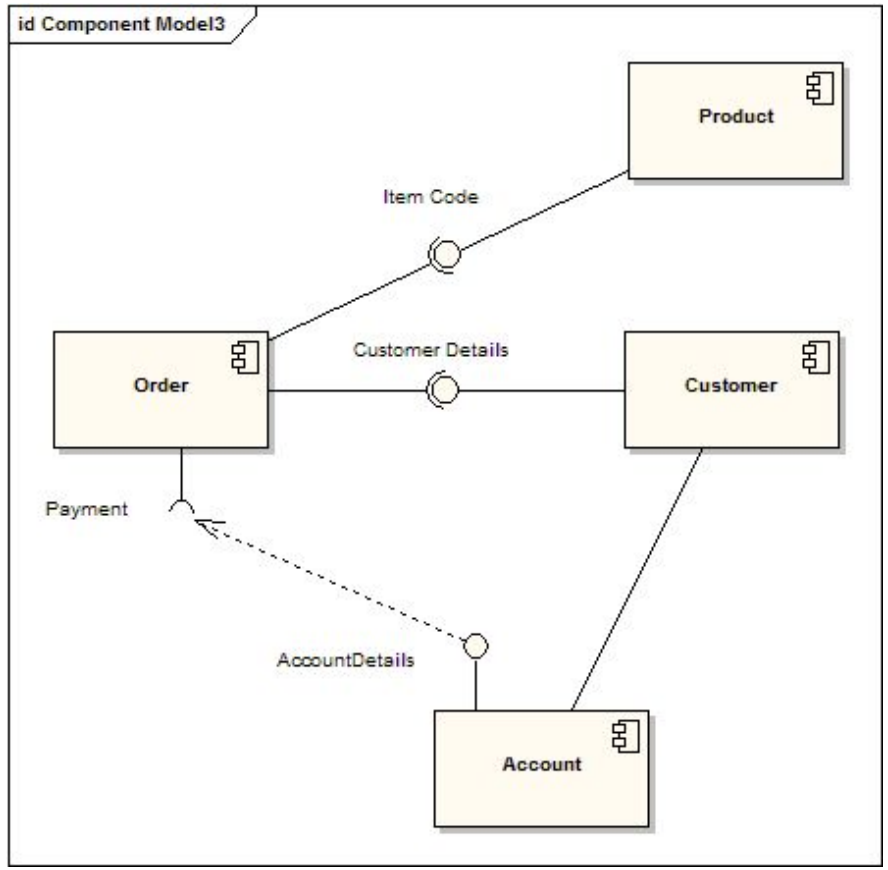


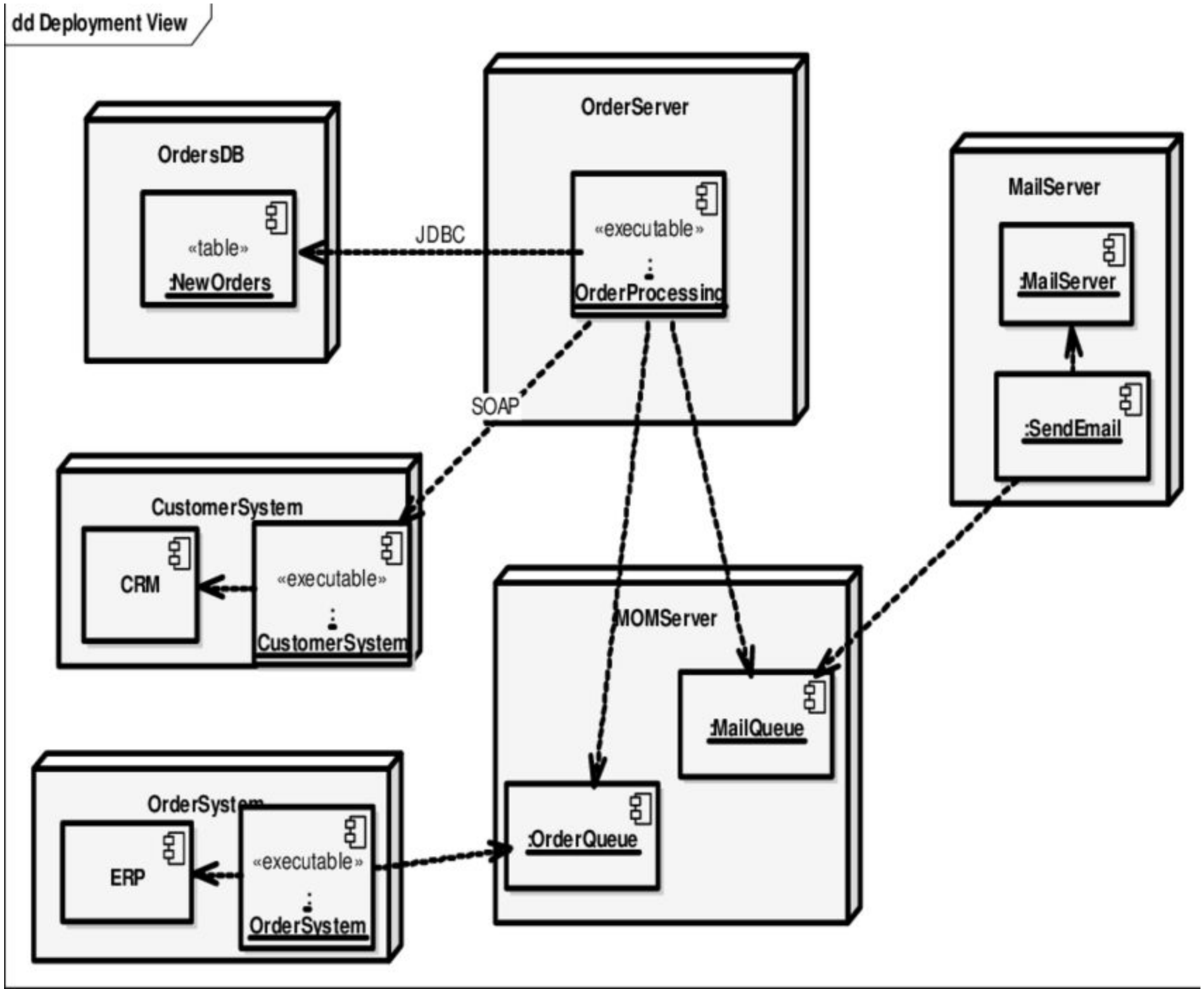
Component Diagram of Library Management System

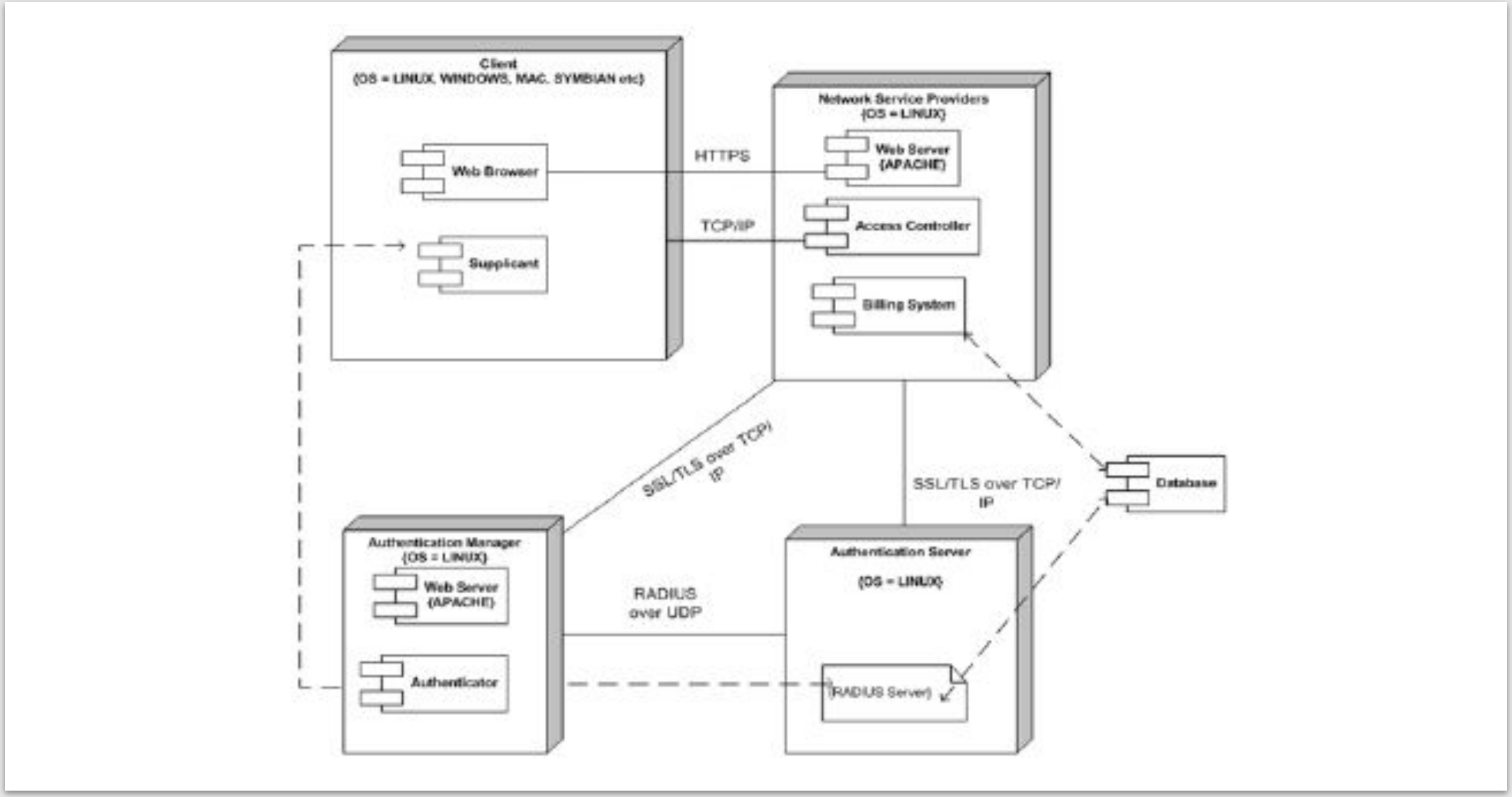
UML Component Diagram Template

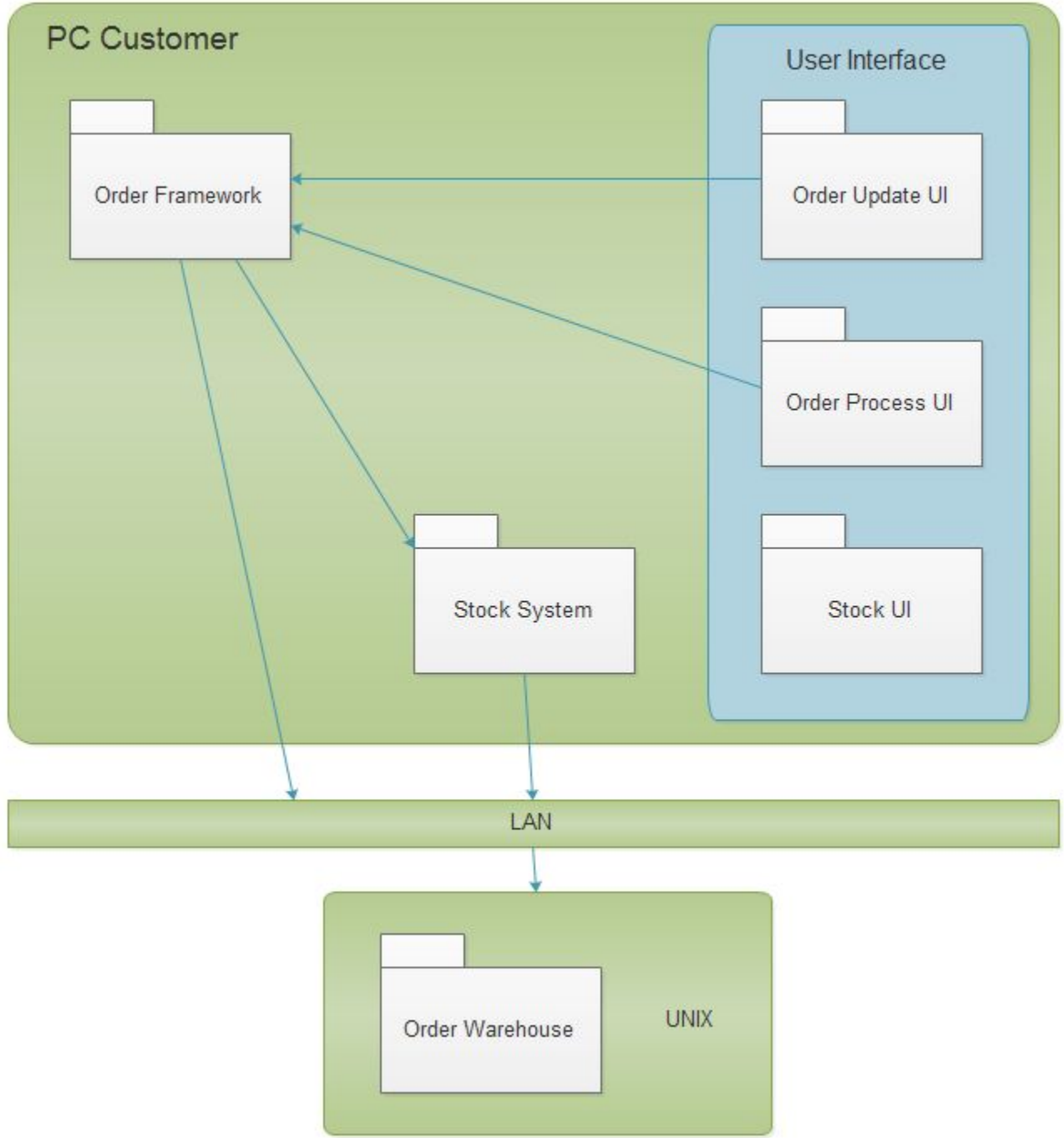



<<component>>
Account
<<component>>
Product
Account payable
Orderable item
<<component>>
Order
<<focus>>
Order header
1
*
Order
Item
Line item


id Component Model3
Product
Item Code
Order
Customer Details
Customer
Payment
AccountDetails
Account


dd Deployment View
OrderServer
OrdersDB
«table»
:NewOrders
JDBC
«executable»
OrderProcessing
MailServer
:MailServer
:SendEmail
SOAP
CustomerSystem
CRM
«executable»
CustomerSystem
MOMServer
:MailQueue
:OrderQueue
OrderSystem
ERP
«executable»
OrderSystem


Client
(OS = LINUX, WINDOWS, MAC, SYMBIAN etc)
Web Browser
Supplicant
HTTPS
TCP/IP
Network Service Providers
(OS = LINUX)
Web Server
(APACHE)
Access Controller
Billing System
SSL/TLS over TCP/
IP
SSL/TLS over TCP/
IP
Database
Authentication Manager
(OS = LINUX)
Web Server
(APACHE)
Authenticator
RADIUS
over UDP
Authentication Server
(OS = LINUX)
(RADIUS Server)


PC Customer
User Interface
Order Framework
Order Update UI
Order Process UI
Stock System
Stock UI
LAN
Order Warehouse
UNIX

4.9 Validation and Verification

- In requirements validation, we check that our requirements definition accurately reflects the customer's needs
- In verification, we check that one document or artifact conforms to another
- Verification ensures that we build the system right, whereas validation ensures that we build the right system

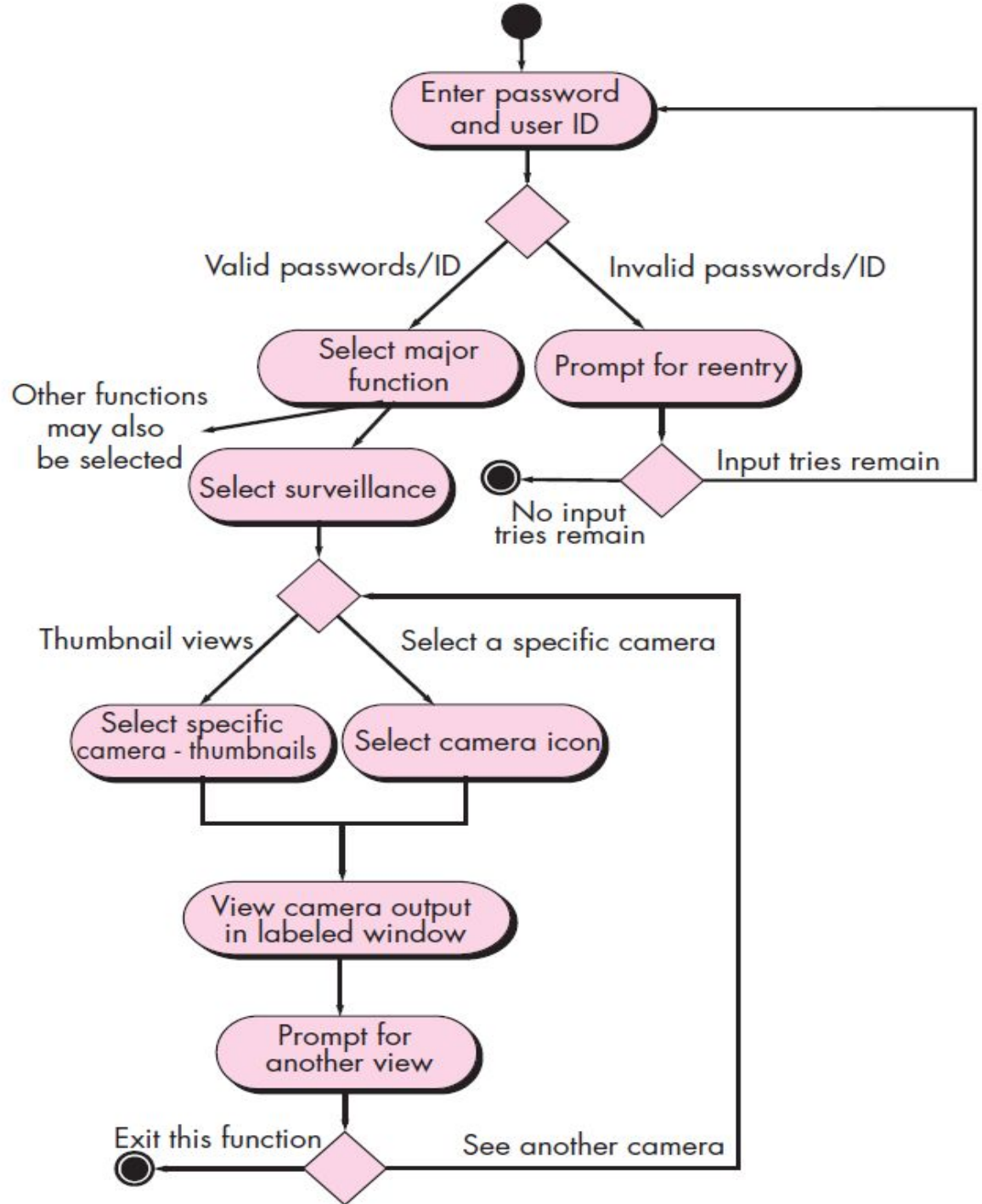
4.9 Validation and Verification

List of techniques to validate requirements

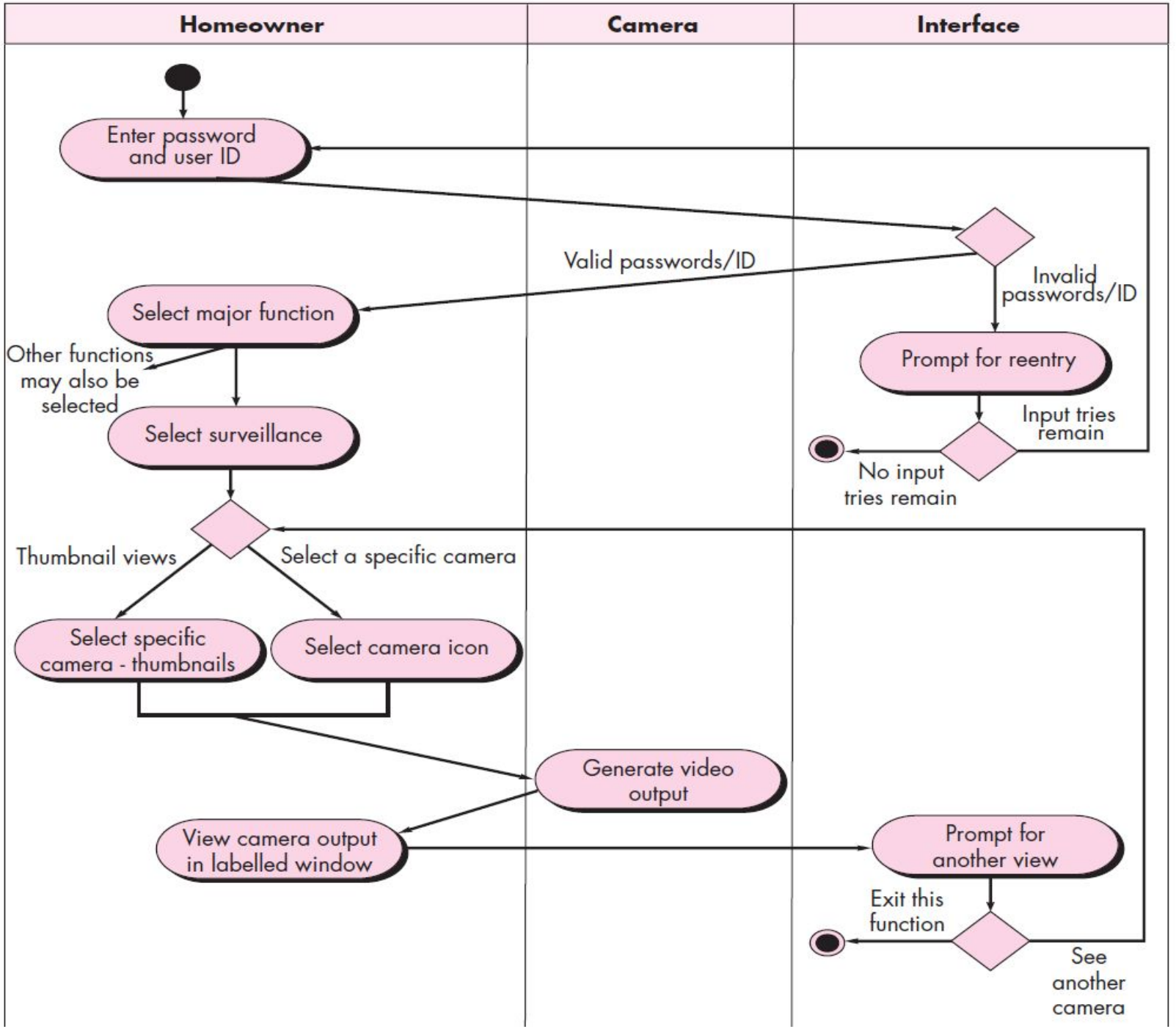
Validation	Walkthroughs Readings Interviews Reviews Checklists Models to check functions and relationships Scenarios Prototypes Simulation Formal inspections
Verification Checking	Cross- referencing Simulation Consistency checks Completeness checks Check for unreachable states of transitions Model checking Mathematical proofs

4.14 What This Chapter Means for You

- It is essential that the requirements definition and specification documents describe the problem, leaving solution selection to designer
- There are variety of sources and means for eliciting requirements
- There are many different types of definition and specification techniques
- The specification techniques also differ in terms of their tool support, maturity, understandability, ease of use, and mathematical formality
- Requirements questions can be answered using models or prototypes
- Requirements must be validated to ensure that they accurately reflect the customer's expectations


Enter password and user ID
Valid passwords/ID
Invalid passwords/ID
Select major function
Prompt for reentry
Other functions may also be selected
Select surveillance
Input tries remain
No input tries remain
Thumbnail views
Select a specific camera
Select specific camera - thumbnails
Select camera icon
View camera output in labeled window
Prompt for another view
Exit this function
See another camera

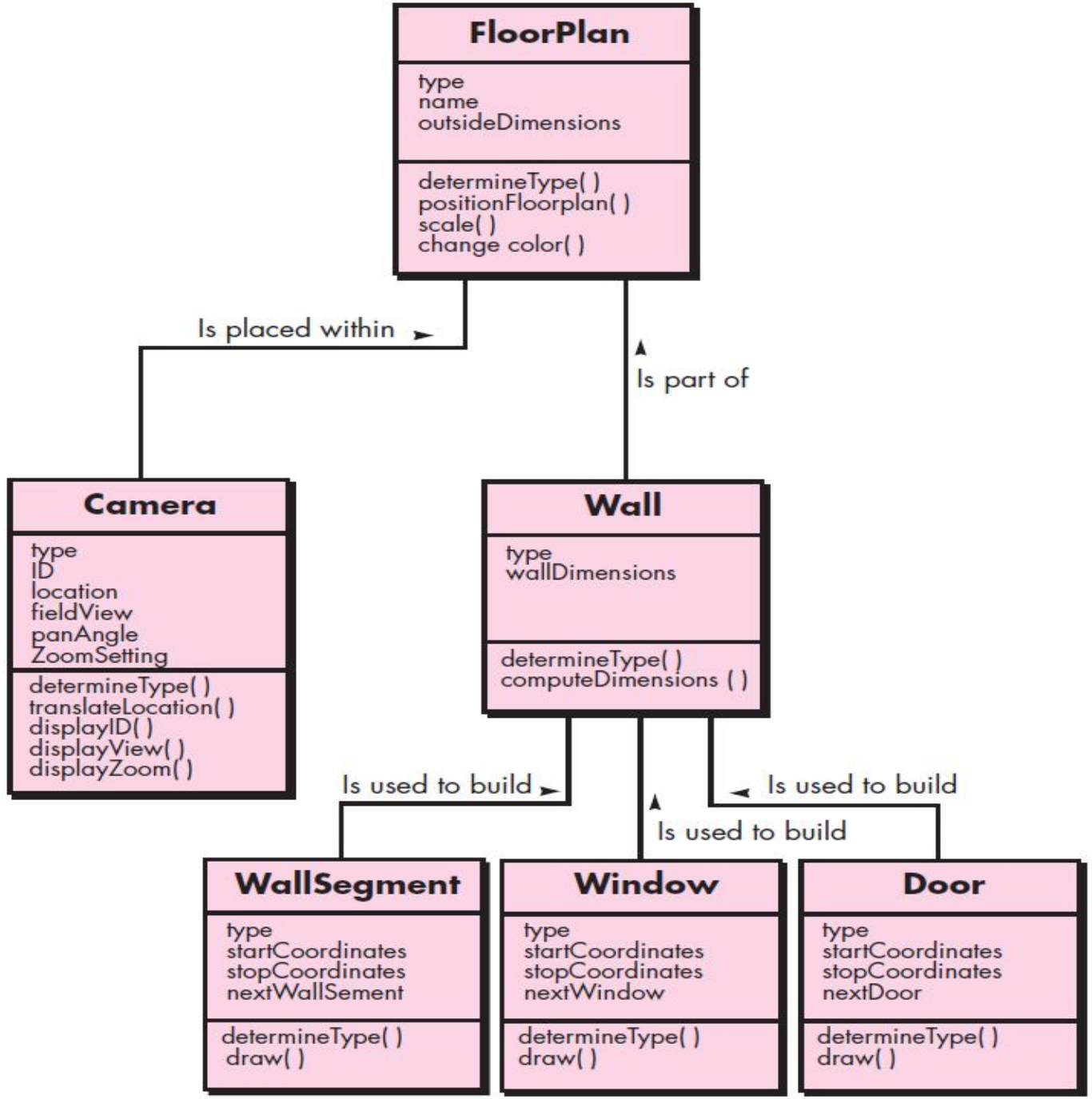
Swimlane Diagrams: Three analysis classes—**Homeowner, Camera,** and **Interface**—have direct or indirect responsibilities in the context of the activity diagram

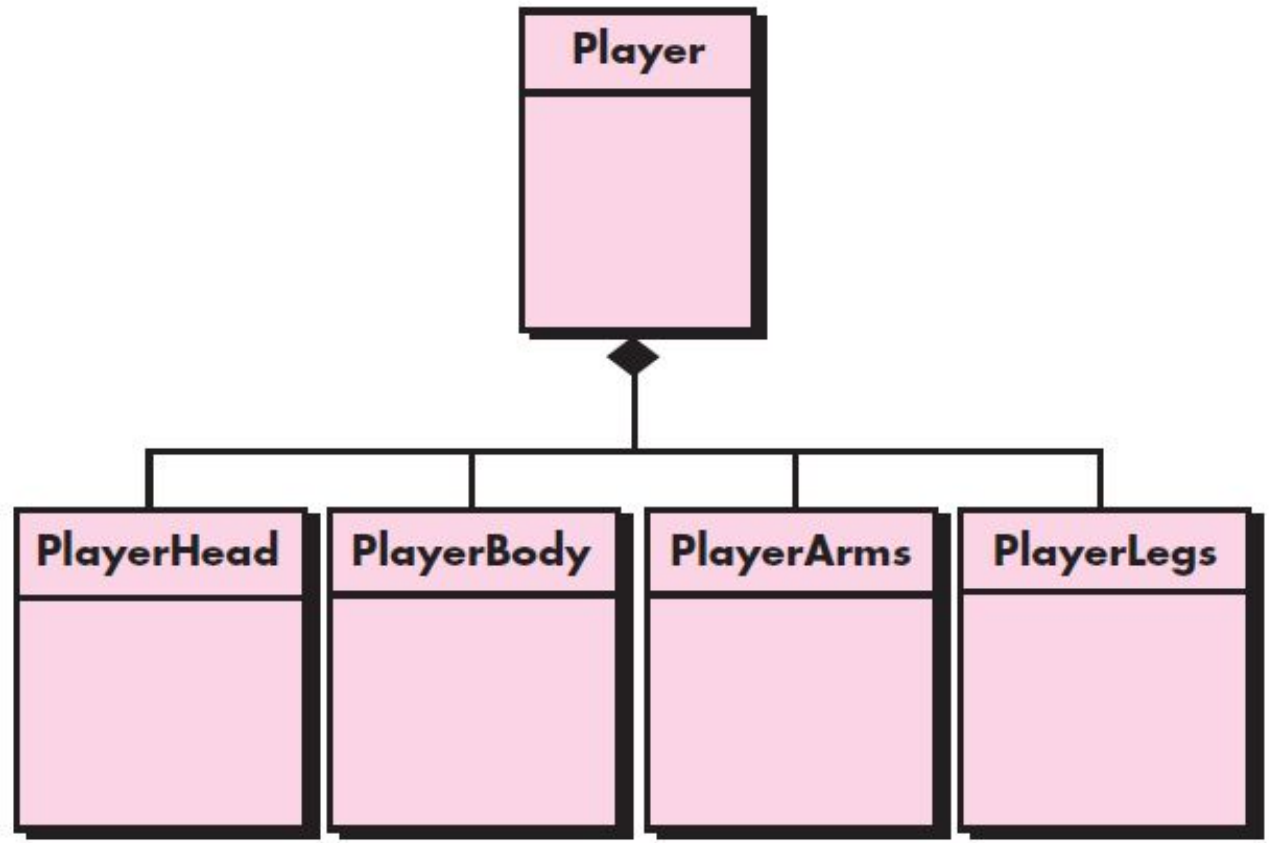


- The homeowner *receives* security information via a control panel, the PC, or a browser,
- collectively called an interface. The interface *displays* prompting messages and system
- status information on the control panel, the PC ,or the browser window. Homeowner interaction
- takes the following form . . .
- Extracting the nouns, we can propose a number of potential classes:

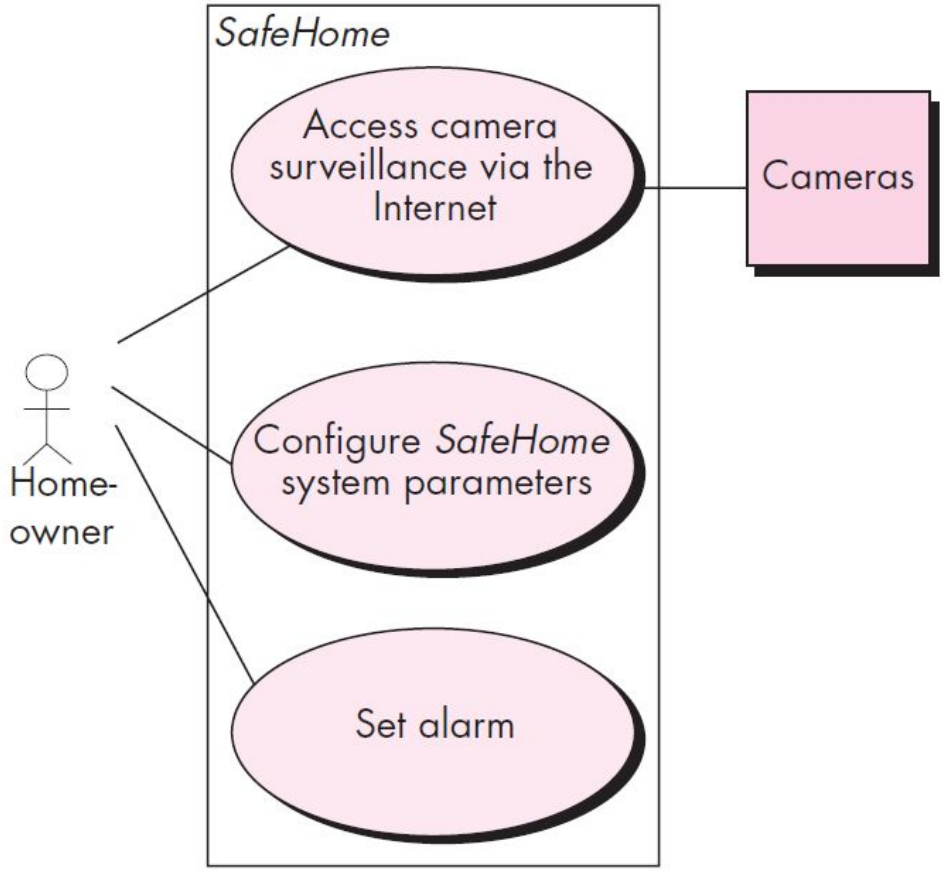
Potential Class	General Classification
homeowner	role or external entity
sensor	external entity
control panel	external entity
installation	occurrence
system (alias security system)	thing
number, type	not objects, attributes of sensor
master password	thing
telephone number	thing
sensor event	occurrence
audible alarm	external entity
monitoring service	organizational unit or external entity

Class: FloorPlan	
Description	
Responsibility:	**Collaborator:**
Defines floor plan name/type	
Manages floor plan positioning	
Scales floor plan for display	
Scales floor plan for display	
Incorporates walls, doors, and windows	**Wall**
Shows position of video cameras	**Camera**




Player
PlayerHead
PlayerBody
PlayerArms
PlayerLegs

Scenario-based Modeling



Input and output for domain analysis