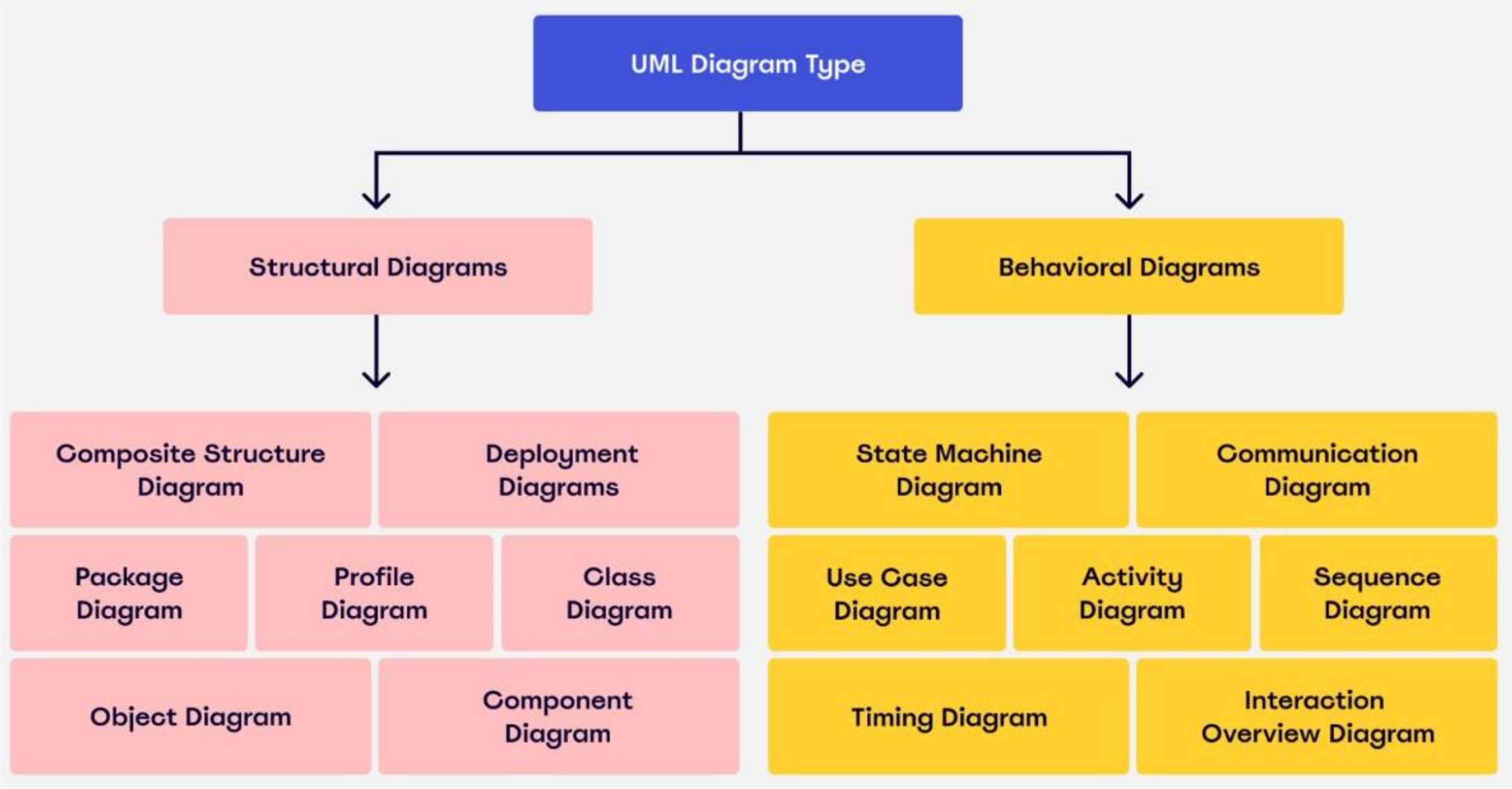
AMRITA
VISHWA VIDYAPEETHAM
श्रद्धावान् लभते ज्ञानम्
DEEMED TO BE UNIVERSITY

System Design using UML

UML Diagrams



UML Diagram Type

Structural Diagrams

Behavioral Diagrams

Composite Structure Diagram

Deployment Diagrams

Package Diagram

Profile Diagram

Class Diagram

Object Diagram

Component Diagram

State Machine Diagram

Communication Diagram

Use Case Diagram

Activity Diagram

Sequence Diagram

Timing Diagram

Interaction Overview Diagram

System Design

- For our project, the system design should include
  - Use Case diagram
  - Class diagram
  - Sequence Diagram - interactions between objects
  - Activity Diagram.

USE CASE DIAGRAM

- Use-case diagrams model the behavior of a system and help to capture the requirements of the system.
- Use-case diagrams describe the high-level functions and scope of a system.
- These diagrams also identify the interactions between the system and its actors.

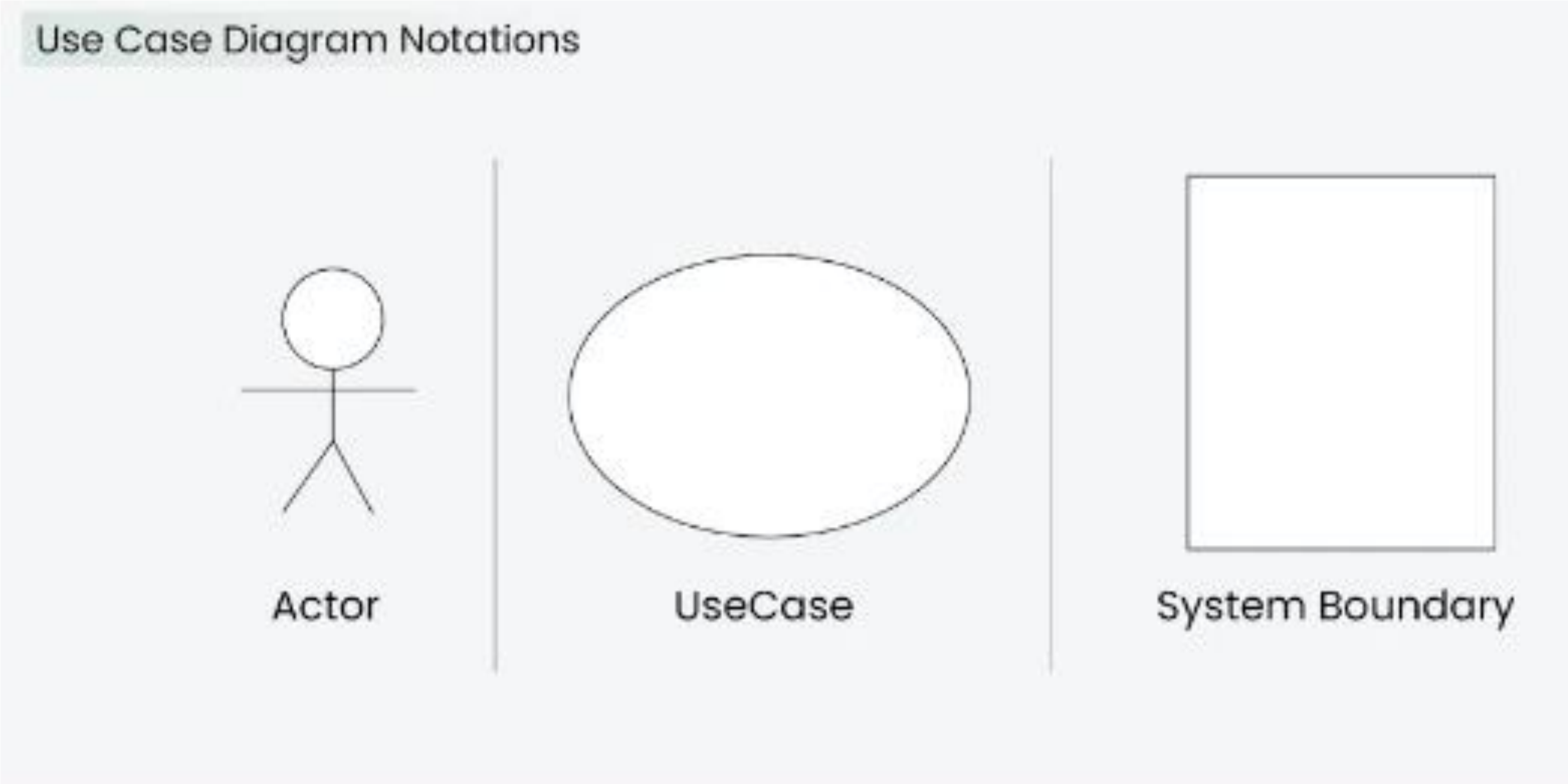
Use cases

- A use case describes a function that a system performs to achieve the user's goal. A use case must yield an observable result that is of value to the user of the system.

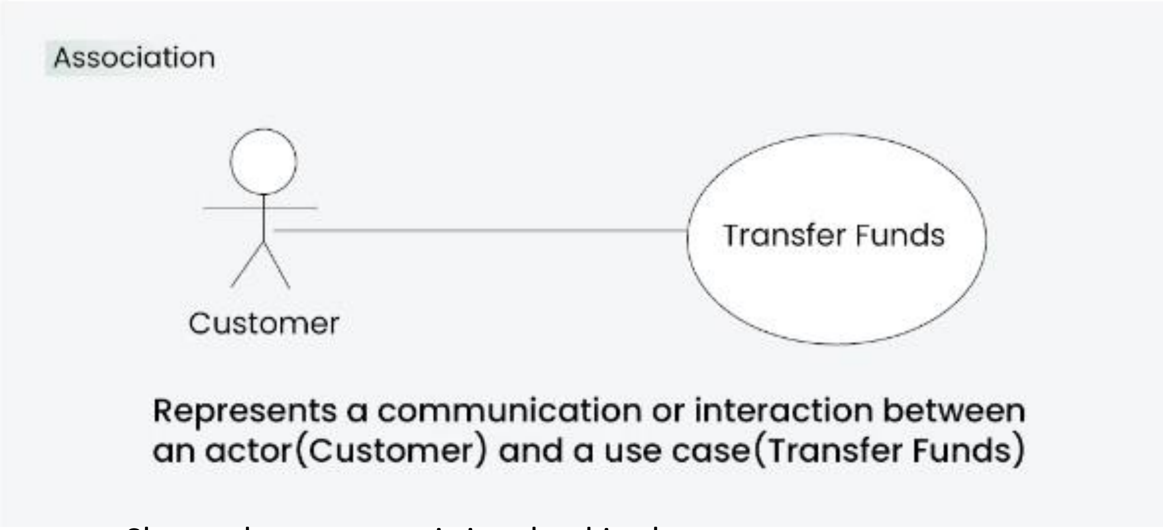
Actors

- An actor represents a role of a user that interacts with the system that you are modeling. The user can be a human user, an organization, a machine, or another external system.

Use Case Diagram - Components

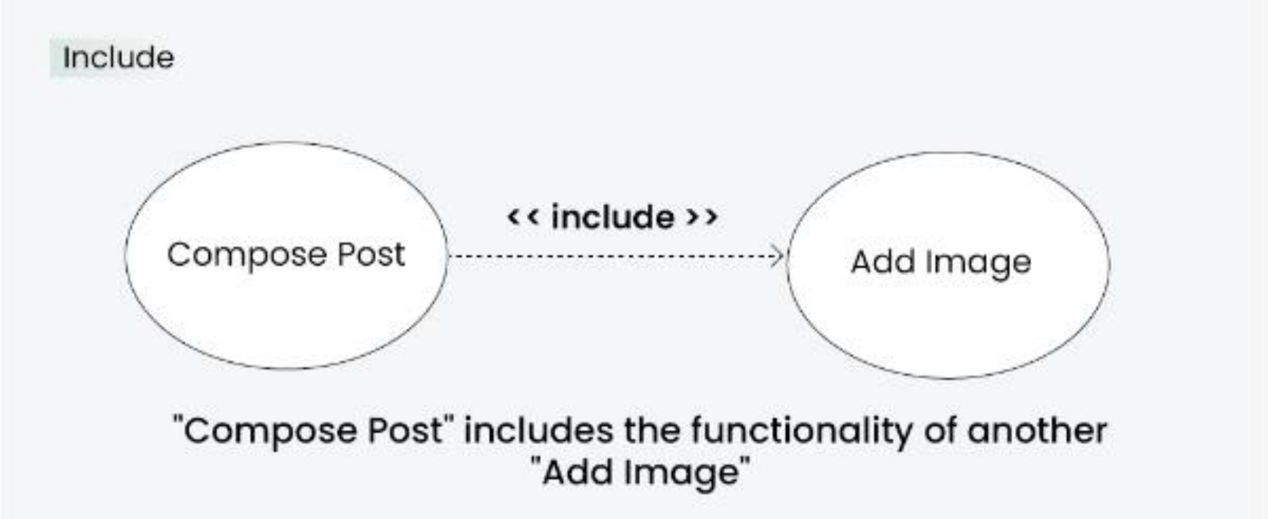


Use Case Diagram - Relationships



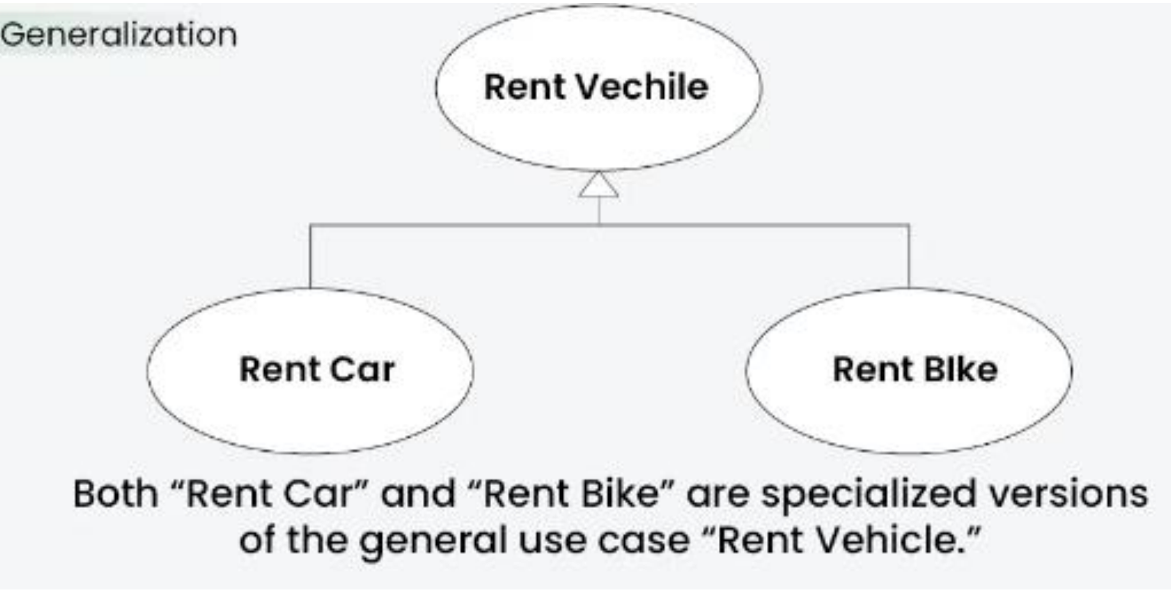
Represents a communication or interaction between an actor(Customer) and a use case(Transfer Funds)

Shows that an actor is involved in the use case

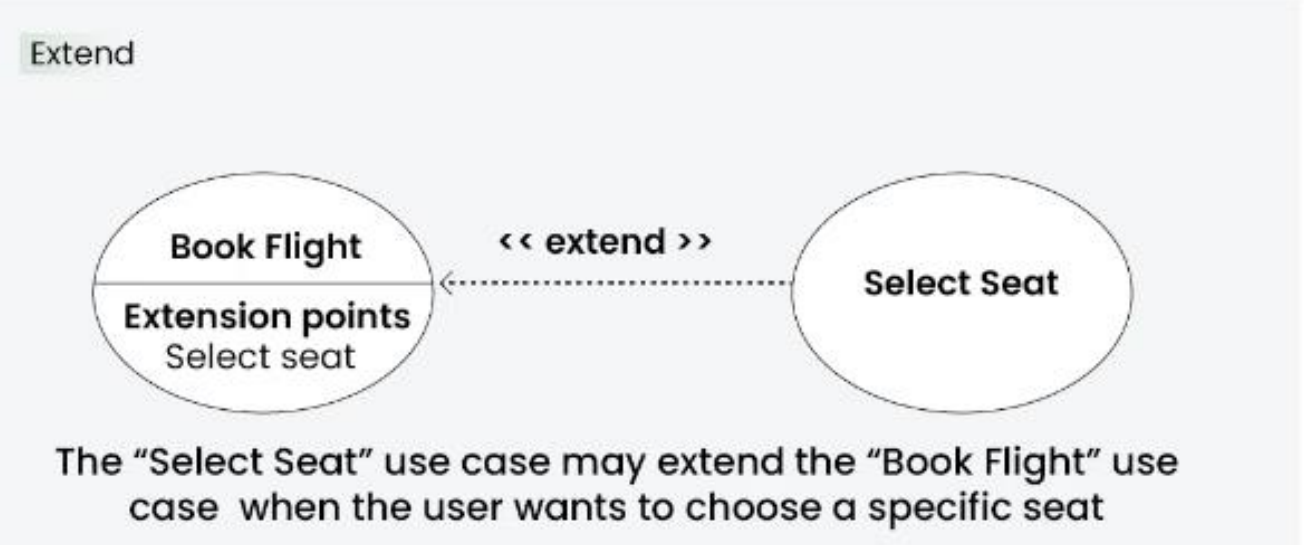


"Compose Post" includes the functionality of another "Add Image"

extract common functionality reused across multiple use cases



Both "Rent Car" and "Rent Bike" are specialized versions of the general use case "Rent Vehicle."



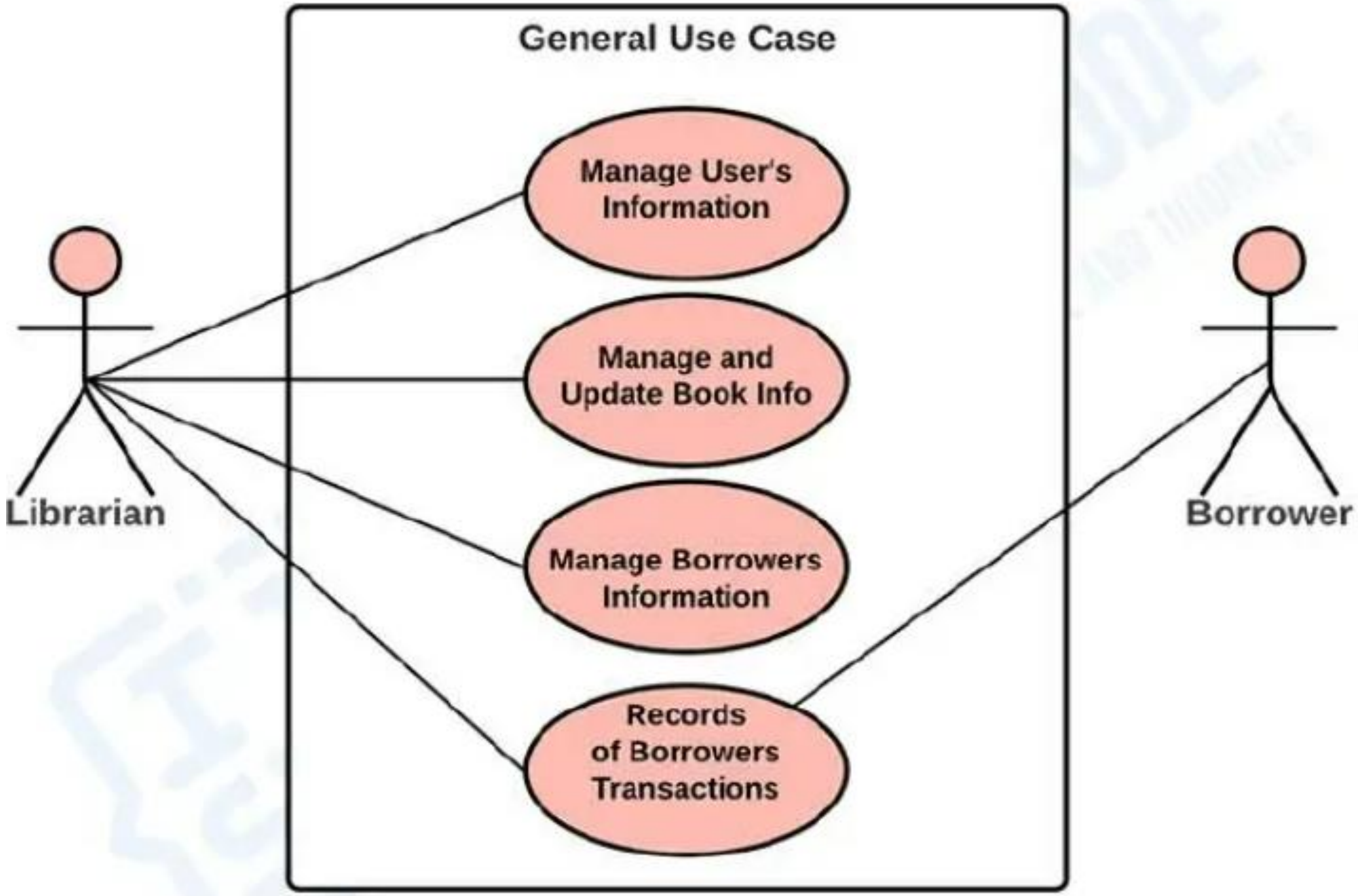
The "Select Seat" use case may extend the "Book Flight" use case when the user wants to choose a specific seat

One use case optionally extends the behavior of another.

One actor or use case inherits the behavior of another

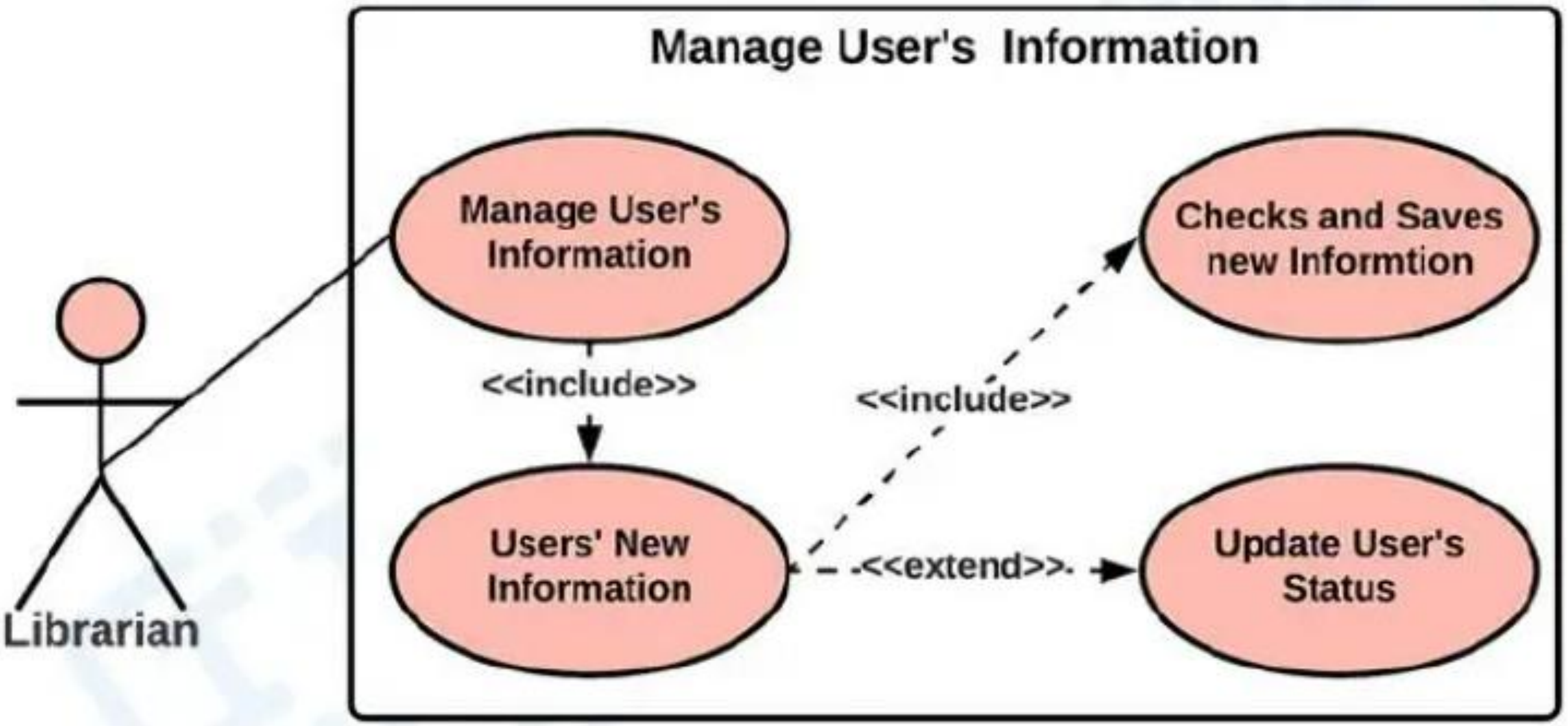
LIBRARY MANAGEMENT SYSTEM

- Use Case Diagram


General Use Case
Manage User's Information
Manage and Update Book Info
Manage Borrowers Information
Records of Borrowers Transactions
Librarian
Borrower

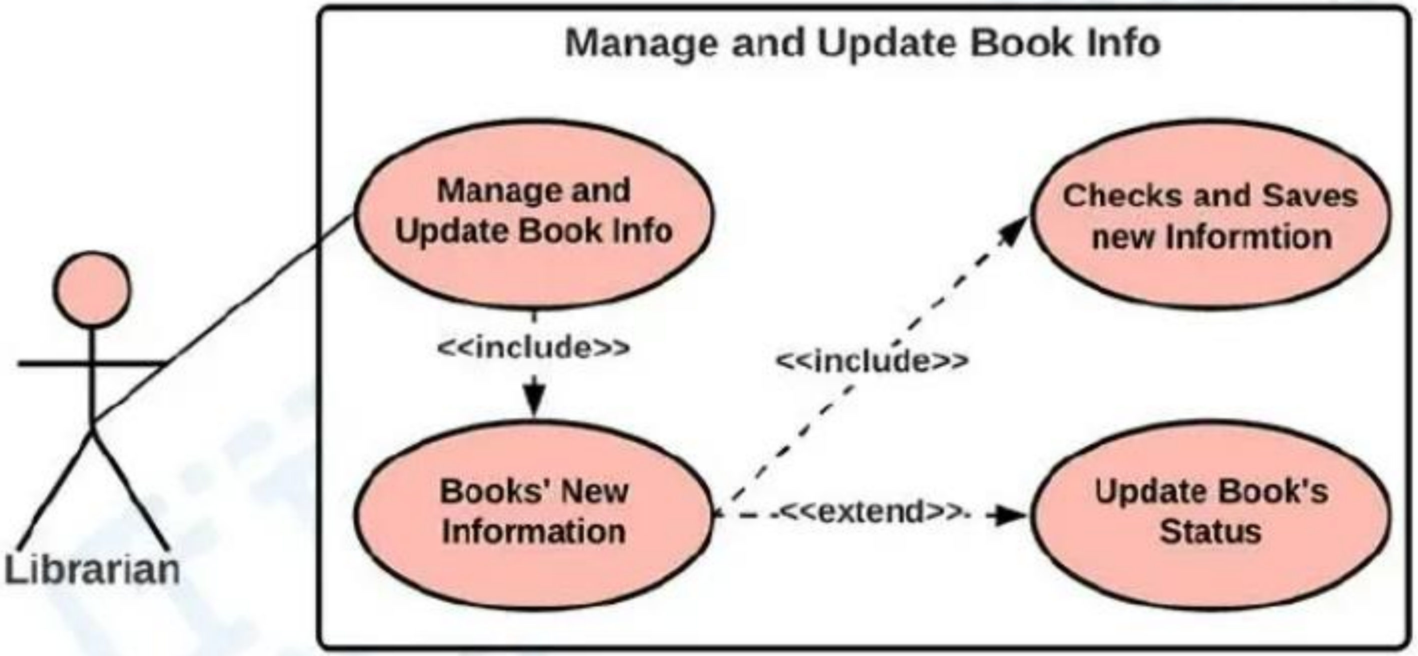
LIBRARY MANAGEMENT SYSTEM

- Use Case – Manage users’ Information



LIBRARY MANAGEMENT SYSTEM

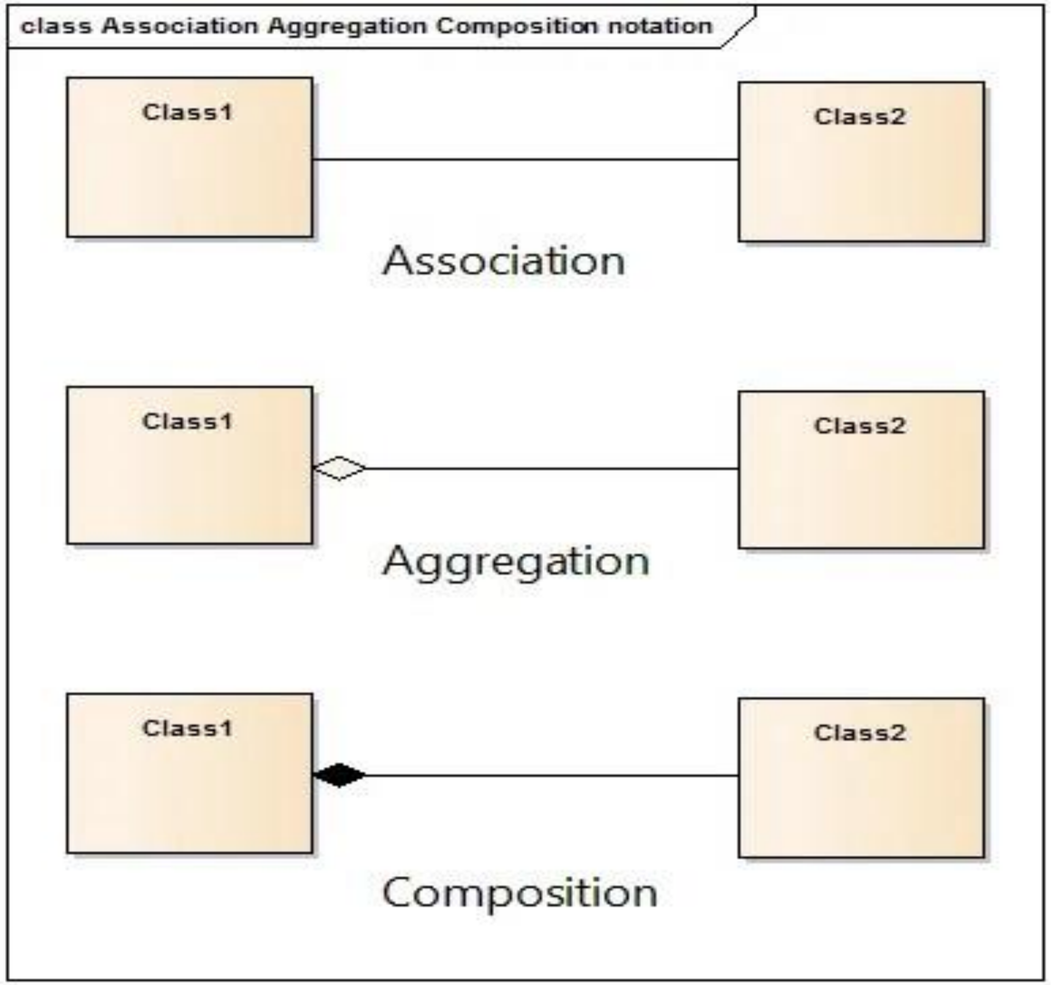
- Use Case – Manage & Update Book Info


Manage and Update Book Info
Librarian
Manage and Update Book Info
<<include>>
Books' New Information
<<include>>
Checks and Saves new Informtion
<<extend>>
Update Book's Status

CLASS DIAGRAM

Class Diagram

- class identification from project spec / requirements
  - nouns are potential classes, objects, fields
  - verbs are potential methods or responsibilities of a class
- Class diagrams are great for:
  - discovering related data and attributes
  - getting a quick picture of the important entities in a system
  - seeing whether you have too few/many classes
  - seeing whether the relationships between objects are too complex, too many in number, simple enough, etc.
  - spotting dependencies between one class/object and another


class Association Aggregation Composition notation
Class1
Class2
Association
Class1
Class2
Aggregation
Class1
Class2
Composition

Association relationships

- Interactions and dependencies between different entities in a system.

There are 3 types of associations:

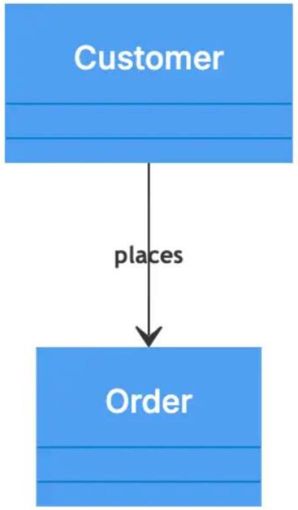
1. **Binary Associations**: The most common type, where two classes are linked. For example, a "Customer" class might be associated with an "Order" class, indicating that customers place orders.
2. **Unary Associations**: Also known as reflexive associations, these involve a single class having a relationship with itself. An example could be an "Employee" class where employees manage other employees.
3. **Ternary Associations**: These involve three classes and are used to represent more complex relationships where three entities are interconnected. For example, a "Student," "Course," and "Instructor" might be related to show which instructors teach which courses to which students.

Example..

Multiplicity

- Multiplicity defines the number of instances of one class that can be associated with a single instance of another class. It specifies the possible range of relationships (e.g., one-to-one, one-to-many, many-to-many).

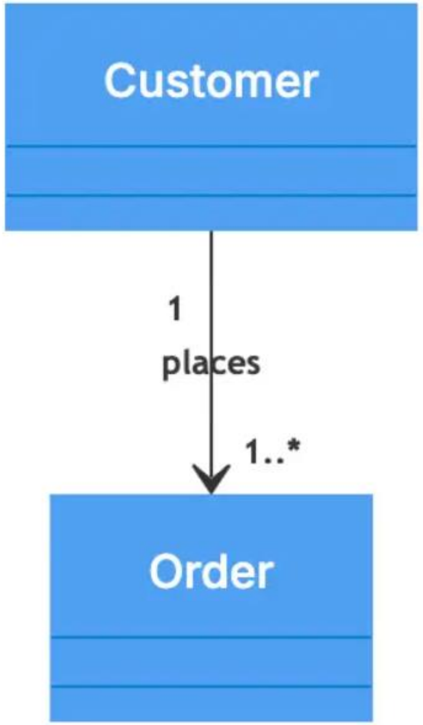
Customer **-places ->** Order



Example:

In the "Customer" and "Order" relationship:

- A customer can place multiple orders: **{1..*}**
- An order is placed by exactly one customer: **{1}**

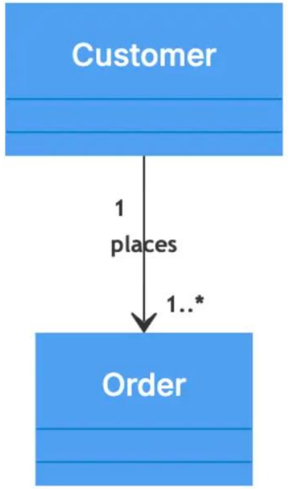


Customer **{1}-places->{1..*}** Order

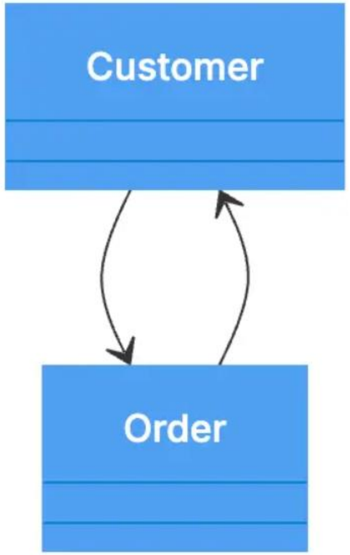
Direction

- Direction indicates whether the association is unidirectional or bidirectional. In a unidirectional association, only one class knows about the relationship with the other, whereas in a bidirectional association, both classes are aware of each other.

Unidirectional: A customer knows about the orders, but orders do not know about the customer



Bidirectional: Both the customer and the order are aware of each other



Clearly Define Roles

For an association between Author and Book:

Author **-writes->** Book

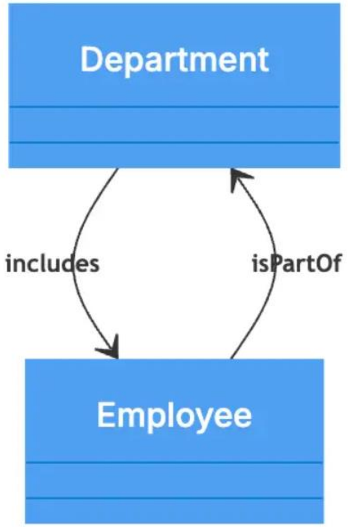
Book **-is written by->** Author

Correctly Specify Multiplicity

University **{1}-->{0..*}** Student.

Common multiplicities include 0..1, 1, 0..*, and `1..*’.

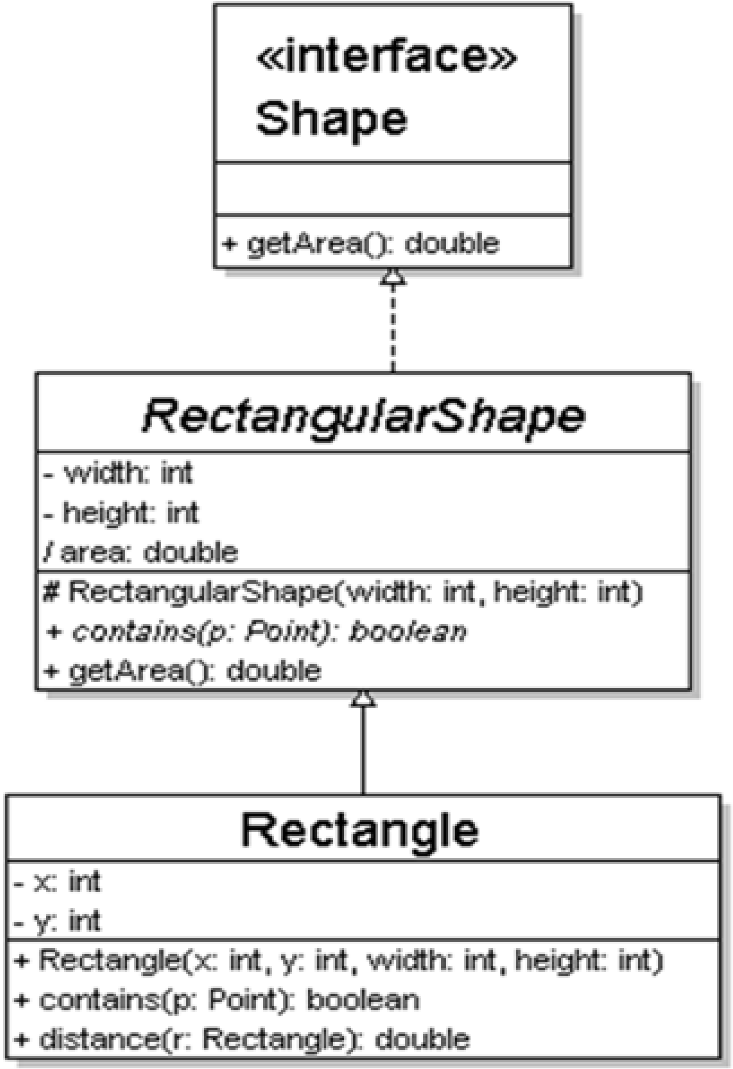
Use Direction and Navigability Appropriately



Generalization (inheritance) relationships

- hierarchies drawn top-down
- arrows point upward to parent
- line/arrow styles indicate whether parent is a(n):
  - class:
    solid line, black arrow
  - abstract class:
    solid line, white arrow
  - interface:
    dashed line, white arrow

- often omit trivial / obvious generalization relationships, such as drawing the Object class as a parent



«interface»
Shape

+ getArea(): double

RectangularShape

- width: int
- height: int
/ area: double

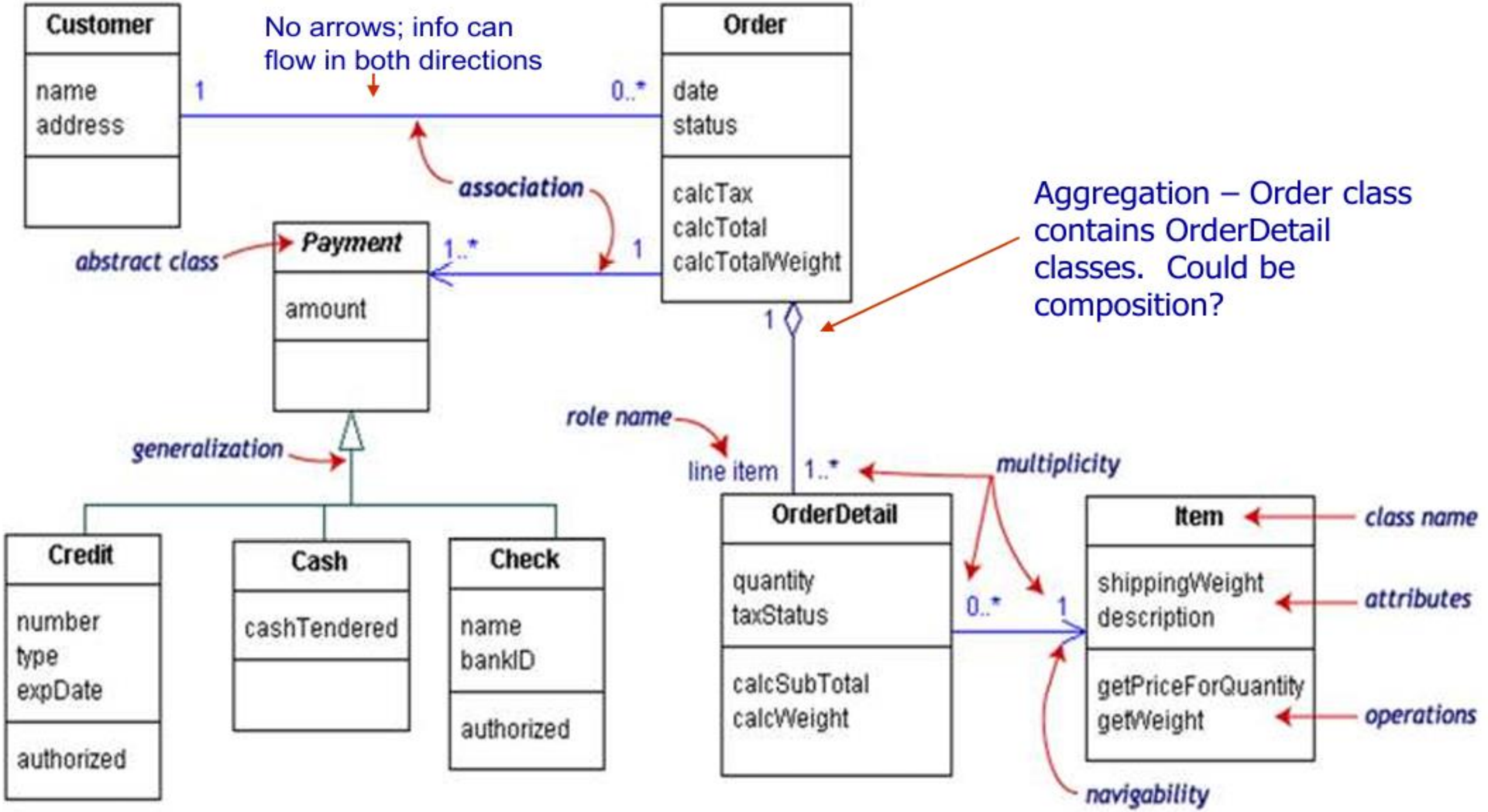
RectangularShape(width: int, height: int)
+ *contains(p: Point): boolean*
+ getArea(): double

Rectangle

- x: int
- y: int

+ Rectangle(x: int, y: int, width: int, height: int)
+ contains(p: Point): boolean
+ distance(r: Rectangle): double

Class diagram example



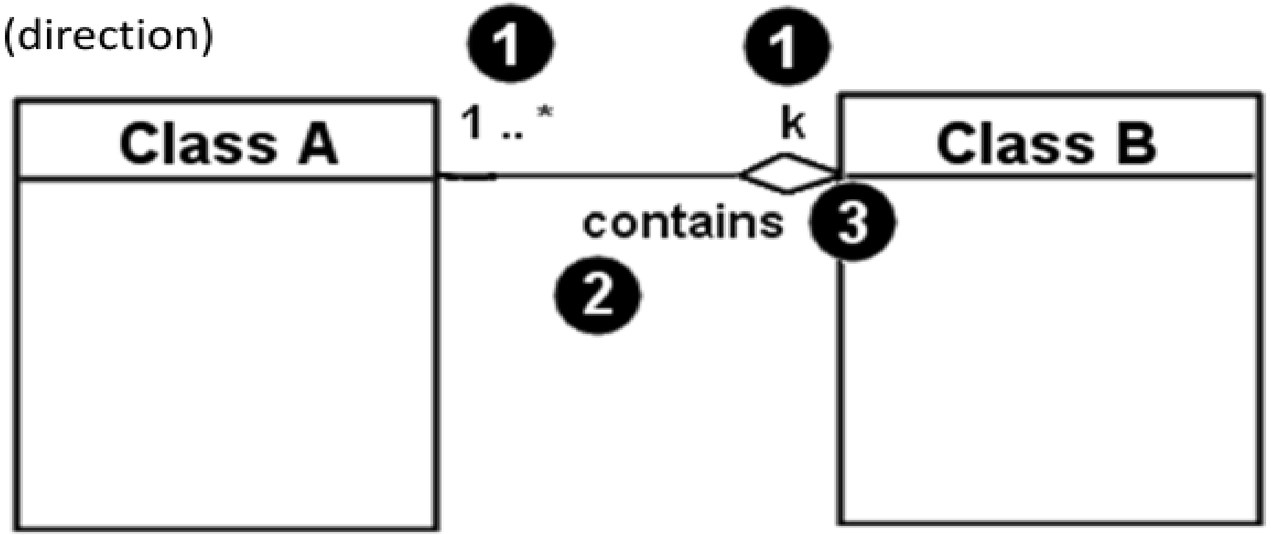
No arrows; info can flow in both directions

Aggregation – Order class contains OrderDetail classes. Could be composition?

Associational relationships

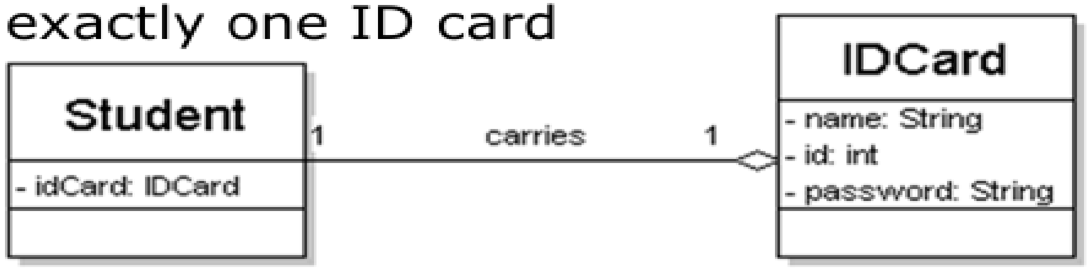
- associational (usage) relationships

  1. multiplicity (how many are used)
     - * ⇒ 0, 1, or more
     - 1 ⇒ 1 exactly
     - 2..4 ⇒ between 2 and 4, inclusive
     - 3..* ⇒ 3 or more (also written as “3..”)
  2. name (what relationship the objects have)
  3. navigability (direction)

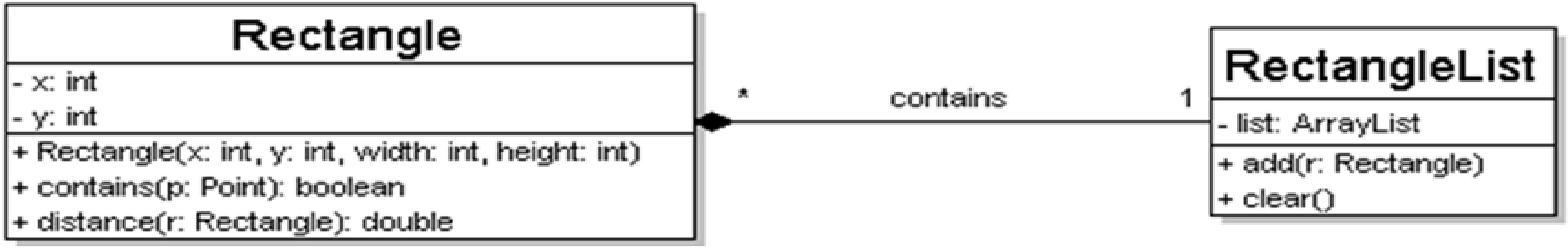


Multiplicity of associations

- one-to-one
  - each student must carry exactly one ID card



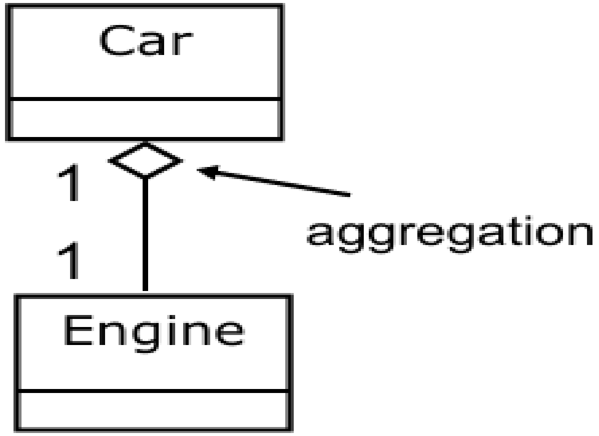
- one-to-many
  - one rectangle list can contain many rectangles



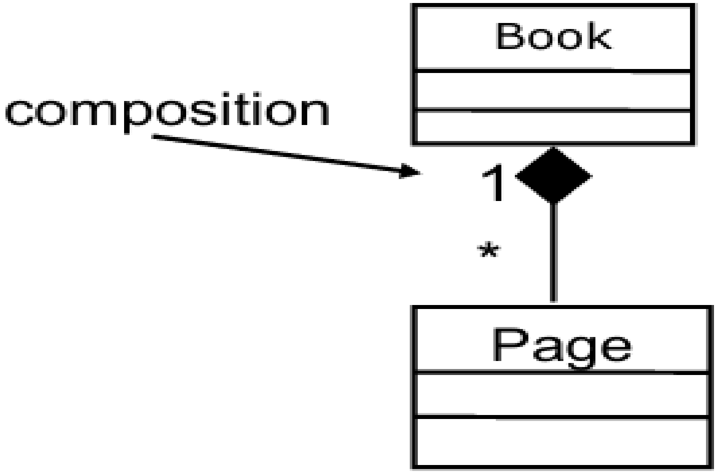
Rectangle
- x: int
- y: int
+ Rectangle(x: int, y: int, width: int, height: int)
+ contains(p: Point): boolean
+ distance(r: Rectangle): double
*
contains
1
RectangleList
- list: ArrayList
+ add(r: Rectangle)
+ clear()

Association types

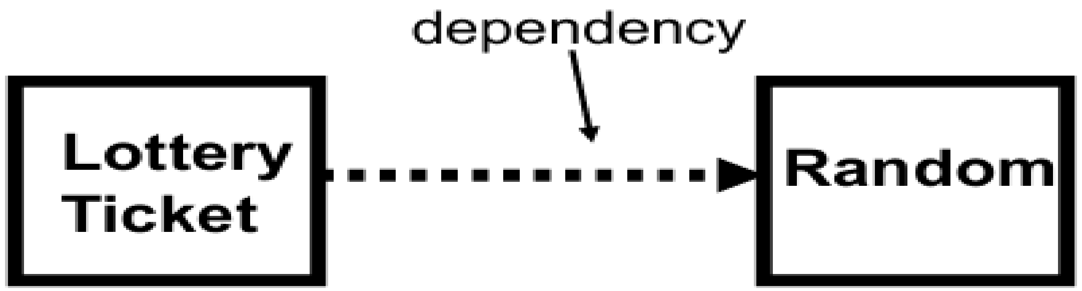
- **aggregation**: “is part of”
  - symbolized by a clear white diamond



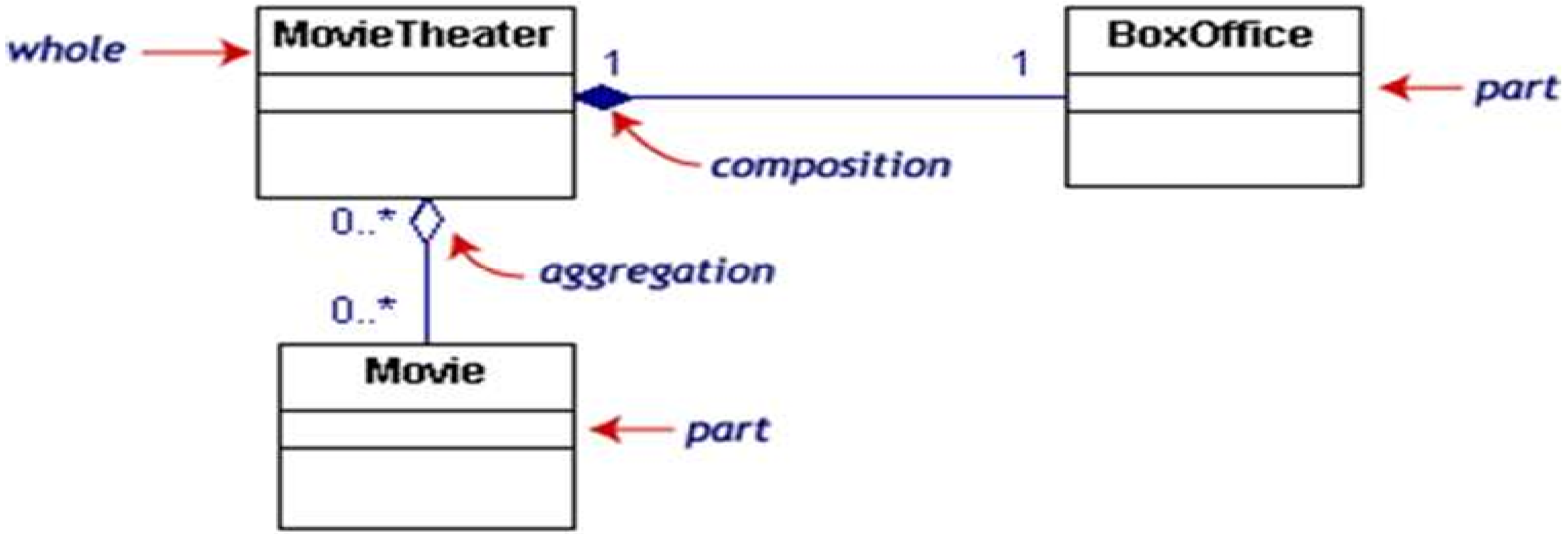
- **composition**: “is entirely made of”
  - stronger version of aggregation
  - the parts live and die with the whole
  - symbolized by a black diamond



- **dependency**: “uses temporarily”
  - symbolized by dotted line
  - often is an implementation detail, not an intrinsic part of that object's state



Composition/aggregation example



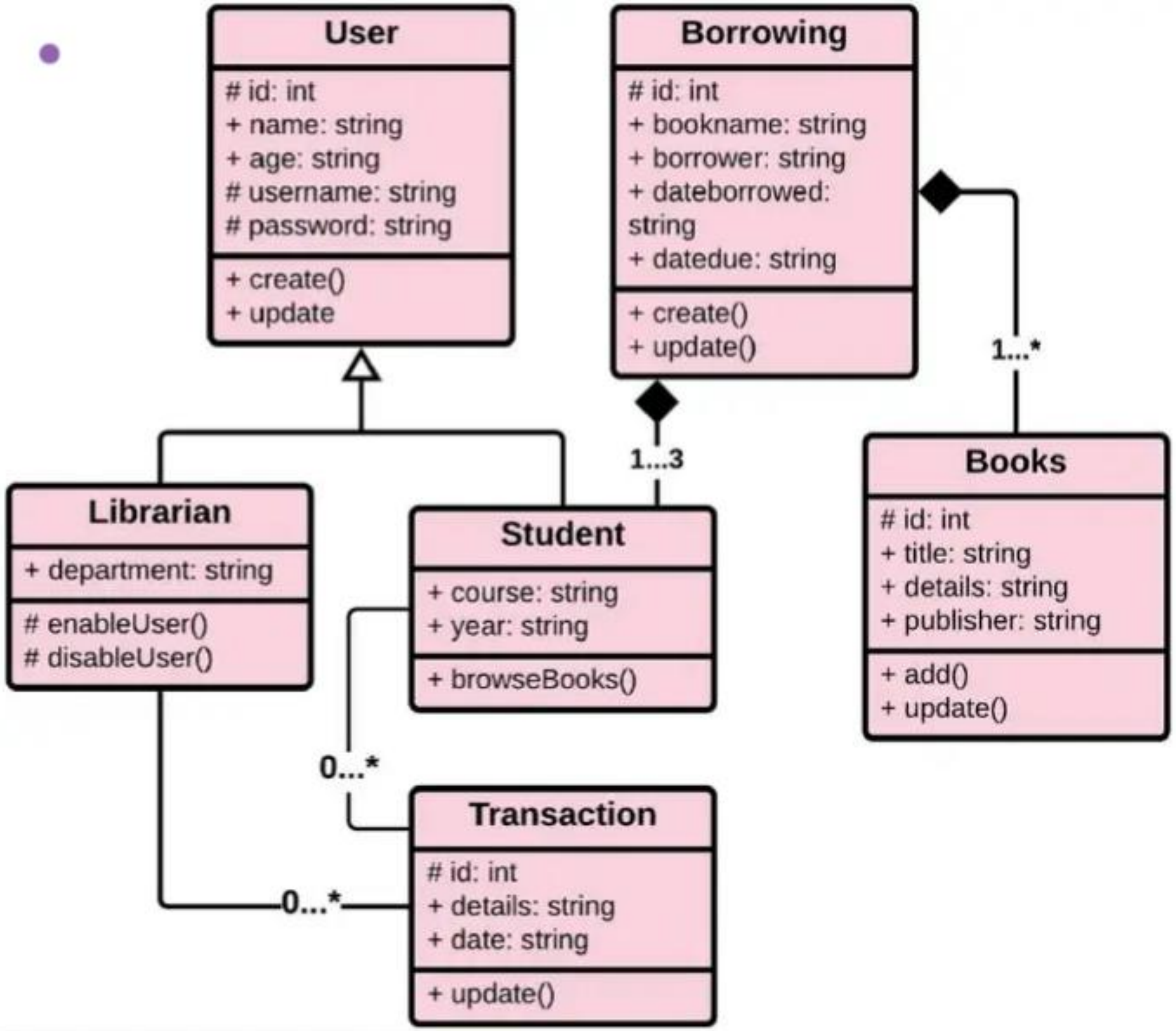
If the movie theater goes away

so does the box office => composition

but movies may still exist => aggregation

LIBRARY MANAGEMENT SYSTEM

- Class Diagram

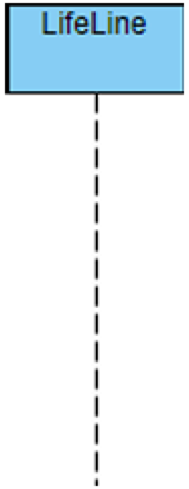


SEQUENCE DIAGRAM

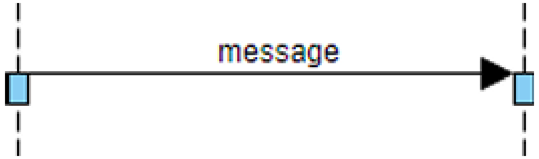
SEQUENCE DIAGRAM

- UML Sequence diagrams are a powerful tool for capturing and visualizing interactions between objects in a system.
- When to Use Sequence Diagrams?
    - Model high-level interactions between active objects in a system.
    - Model interactions within a collaboration that realizes a use case.
    - Model interactions within a collaboration that realizes an operation.
    - Capture either generic interactions (showing all possible paths) or specific instances of an interaction (showing just one path).

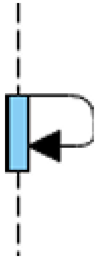
SEQUENCE DIAGRAM



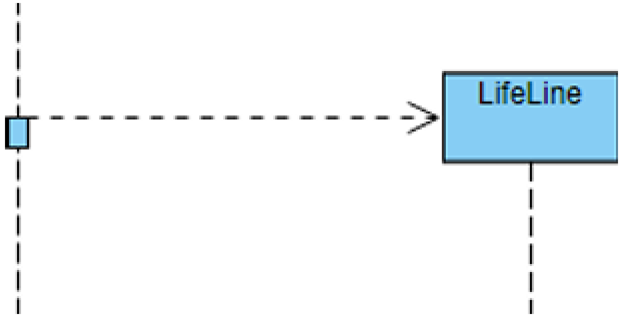
Object & Lifeline



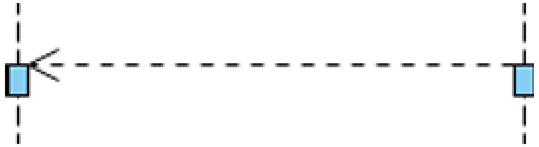
Call Message



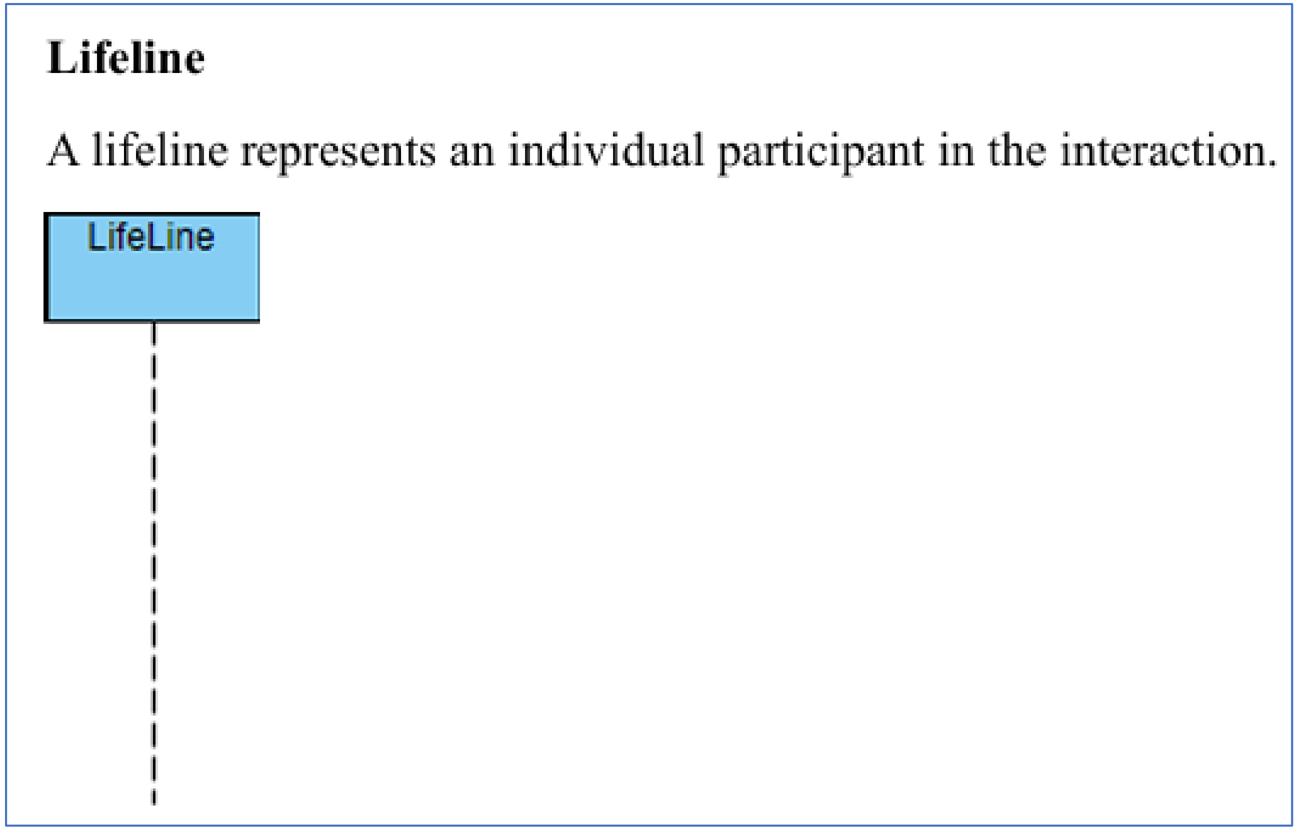
Self Message



Create Message

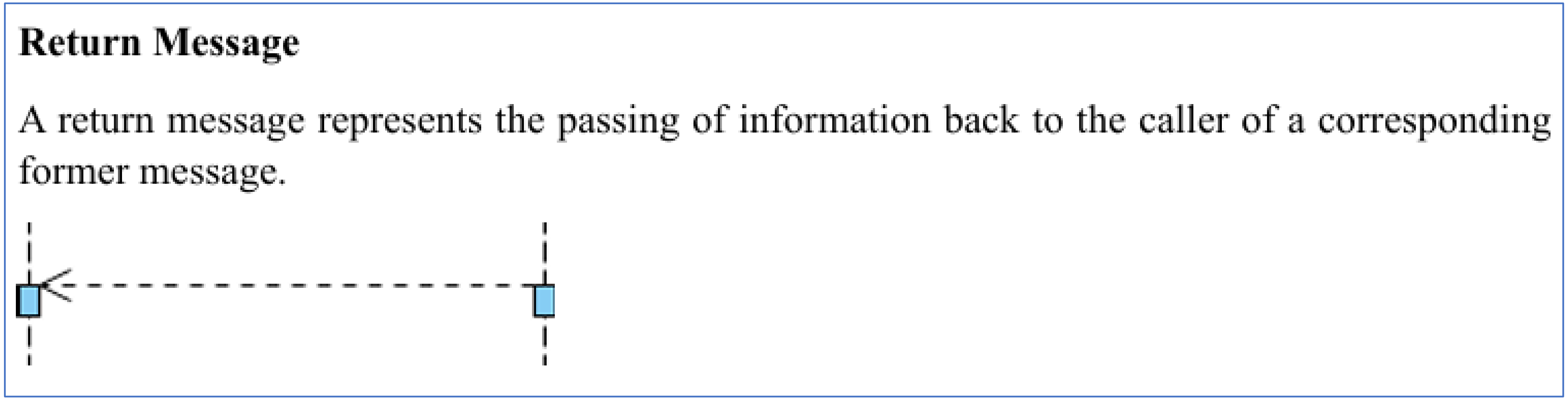


Return Message



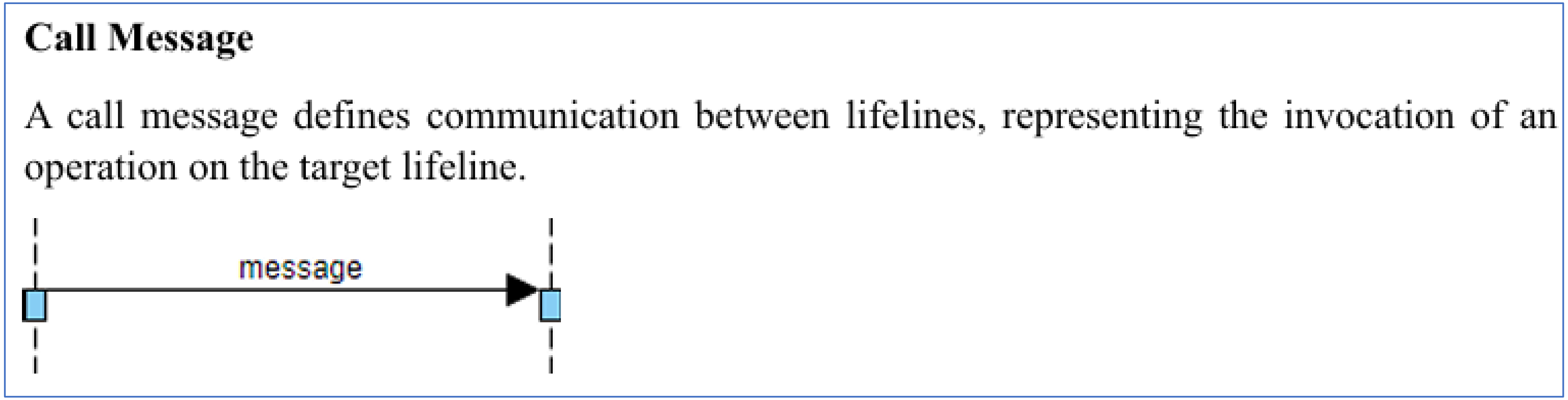
Lifeline

A lifeline represents an individual participant in the interaction.



Return Message

A return message represents the passing of information back to the caller of a corresponding former message.

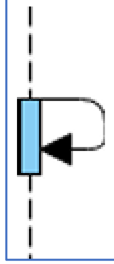


Call Message

A call message defines communication between lifelines, representing the invocation of an operation on the target lifeline.

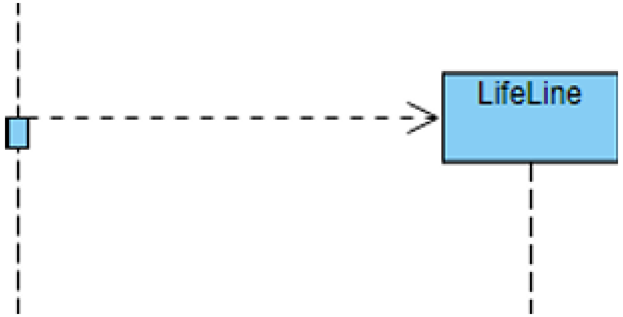
Self Message

A self message denotes communication within the same lifeline, representing the invocation of a message on itself.



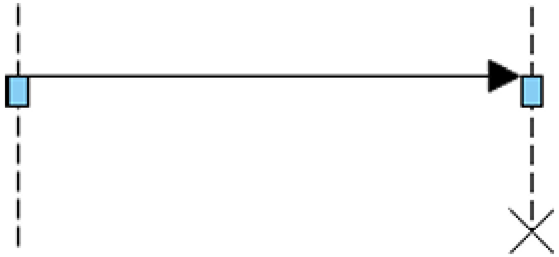
Create Message

A create message signifies the instantiation of a target lifeline.

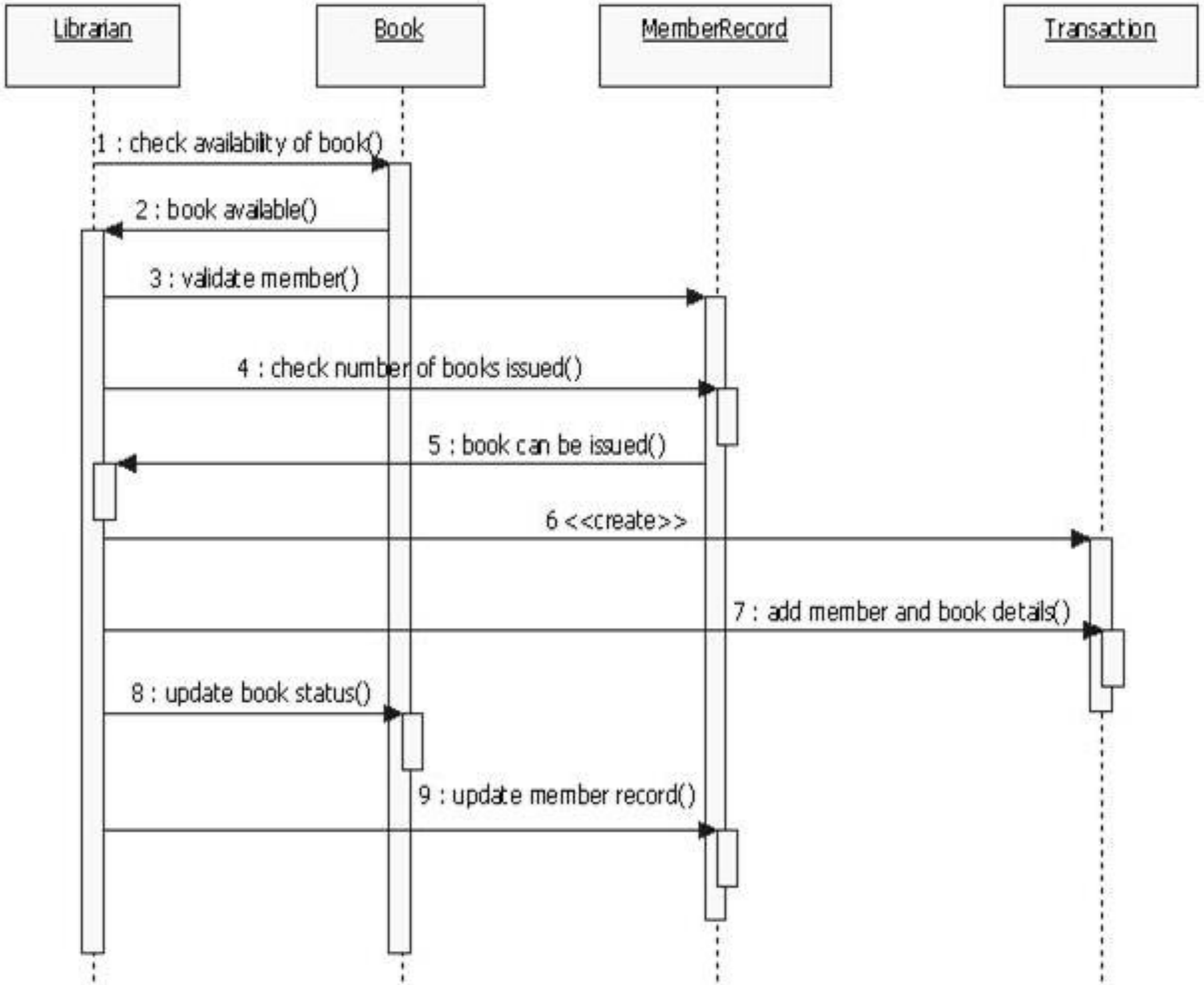


Destroy Message

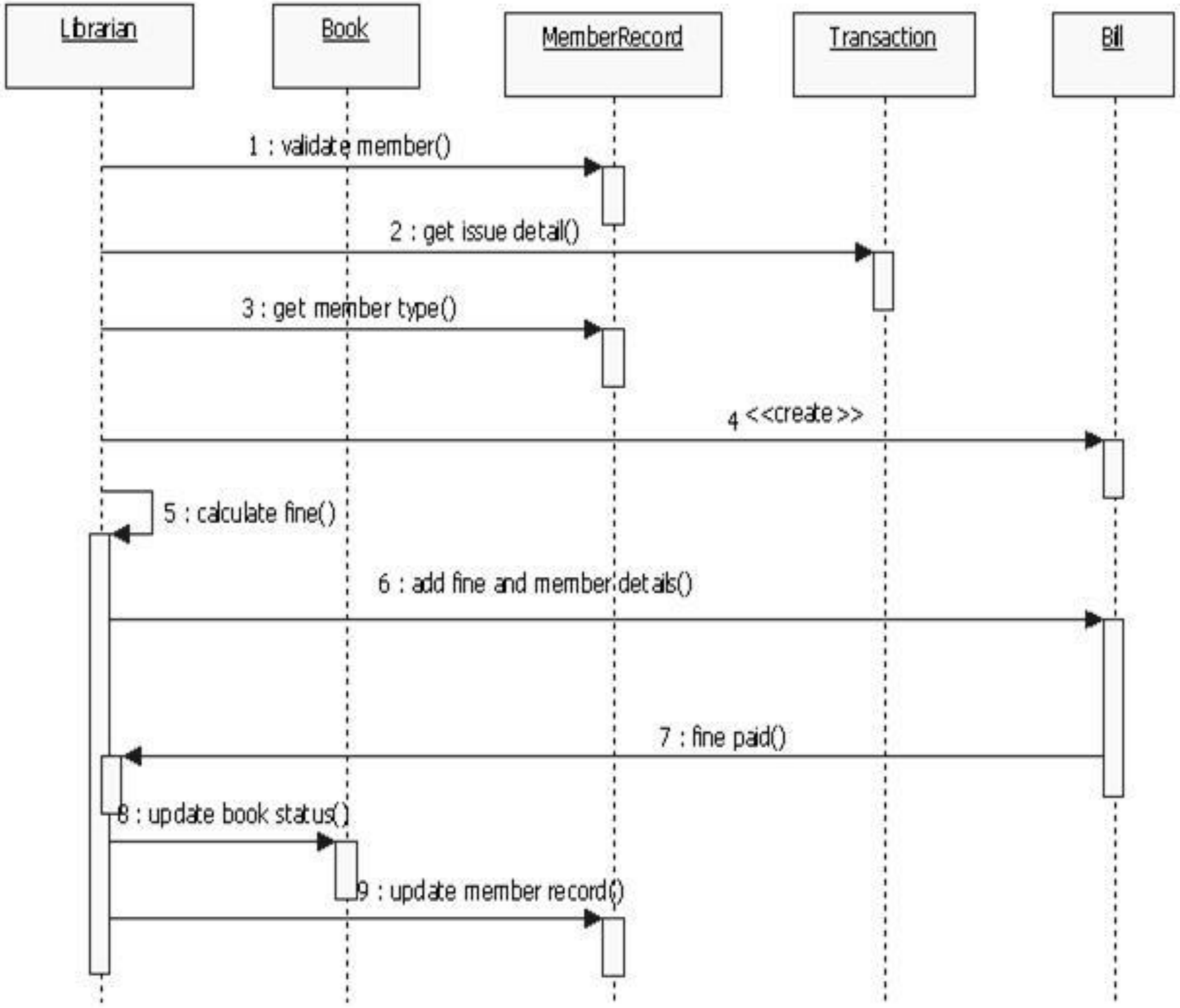
A destroy message represents the request to destroy the lifecycle of the target lifeline.



Sequence Diagram – Issue Book



Sequence Diagram – Return book



ACTIVITY DIAGRAM

ACTIVITY DIAGRAM

- An activity diagram provides a view of the behavior of a system by describing the sequence of actions in a process.
- In activity diagrams, you use activity nodes and activity edges to model the flow of control and data between actions.

Activities

- In UML, activities are container elements that describe the highest level of behavior in an activity diagram. Activities contain several activity nodes and activity edges that represent the sequence of tasks in a workflow that result in a behavior.

Actions

- In UML, an action represents a discrete unit of functionality in an activity.

Control nodes

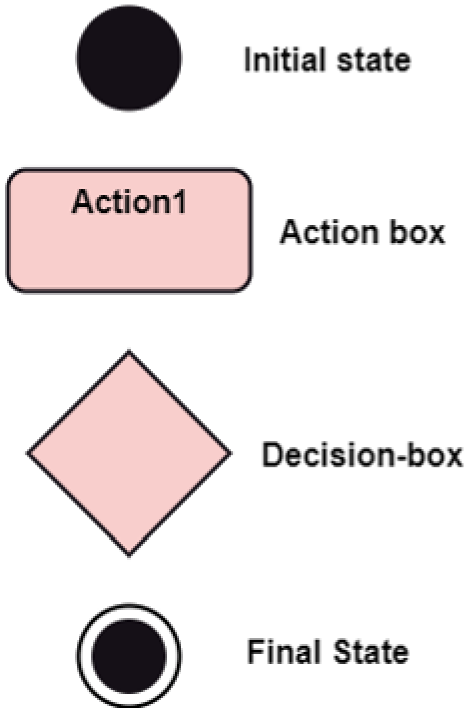
- In activity diagrams, a control node is an abstract activity node that coordinates the flow of control in an activity.

Object nodes

- In activity diagrams, an object node is an abstract activity node that helps to define the object flow in an activity. An object node indicates that an instance of a classifier might be available at a particular point in the activity.

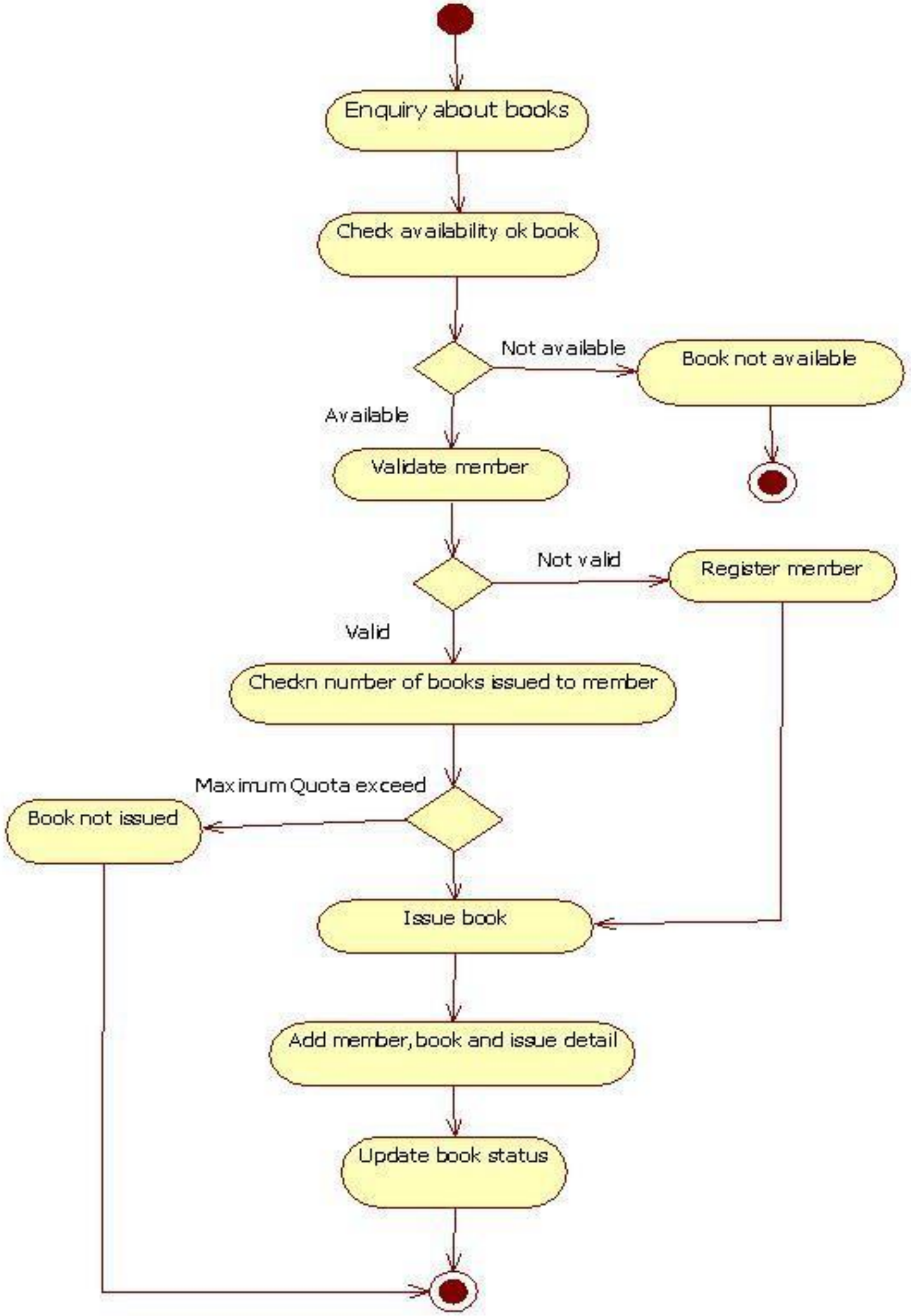
Activity edges

- In activity diagrams, an activity edge is a directed connection between two activity nodes. When a specific action in an activity is complete, the activity edge continues the flow to the next action in the sequence.

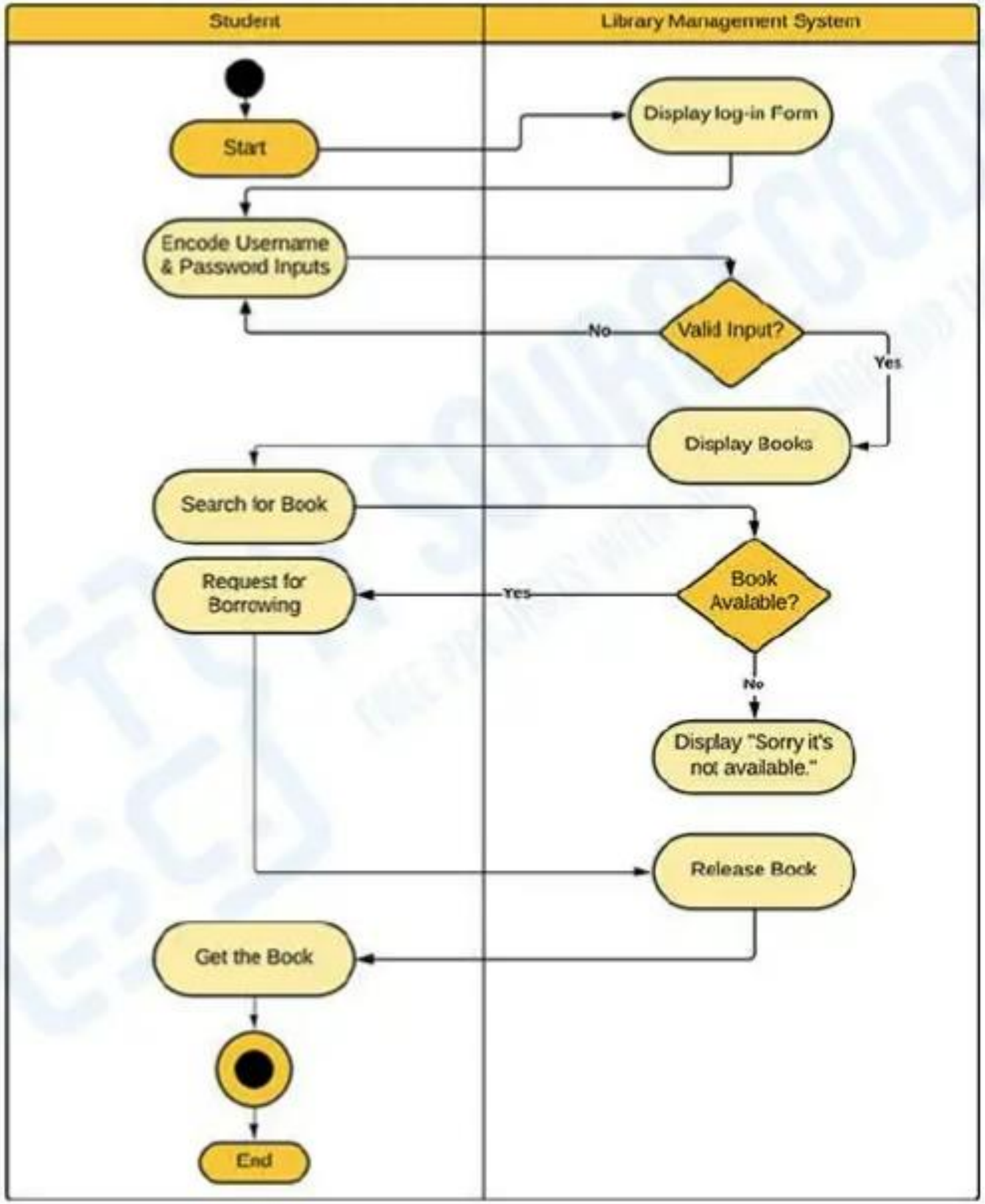

Initial state
Action1
Action box
Decision-box
Final State

Activity Diagram

- Issue Book



Activity Diagram


Student
Library Management System
Start
Display log-in Form
Encode Username & Password Inputs
Valid Input?
No
Yes
Display Books
Search for Book
Book Available?
Yes
Request for Borrowing
No
Display "Sorry it's not available."
Release Book
Get the Book
End