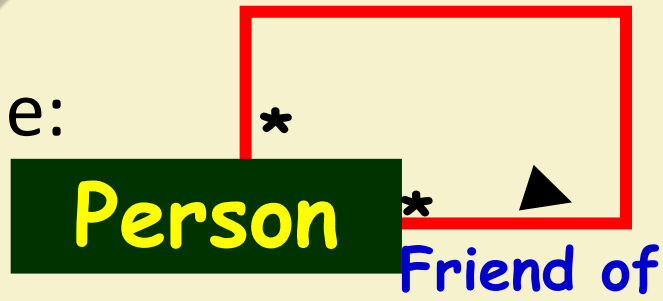
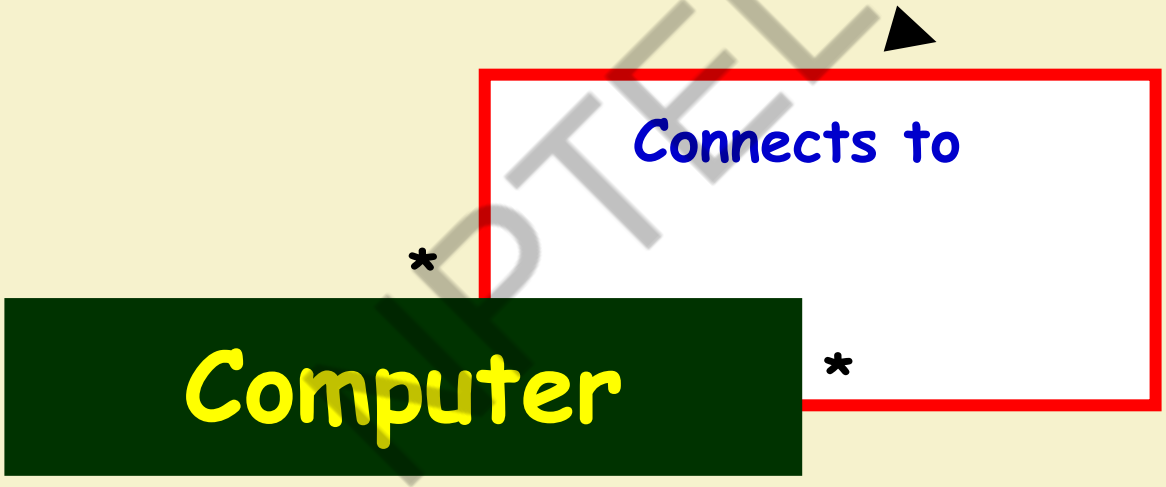
Association Relationship

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

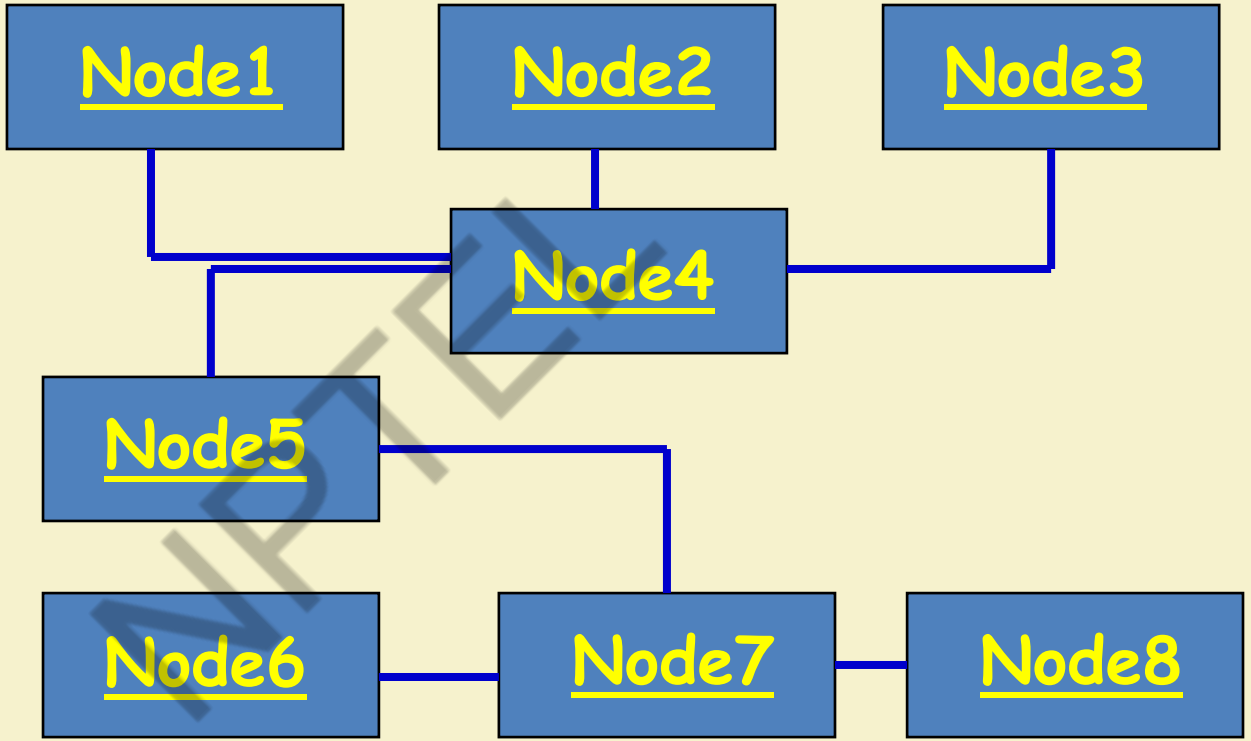


- Multiplicity indicates # of instances taking part in the association.

Self Association: Example of Computer Network

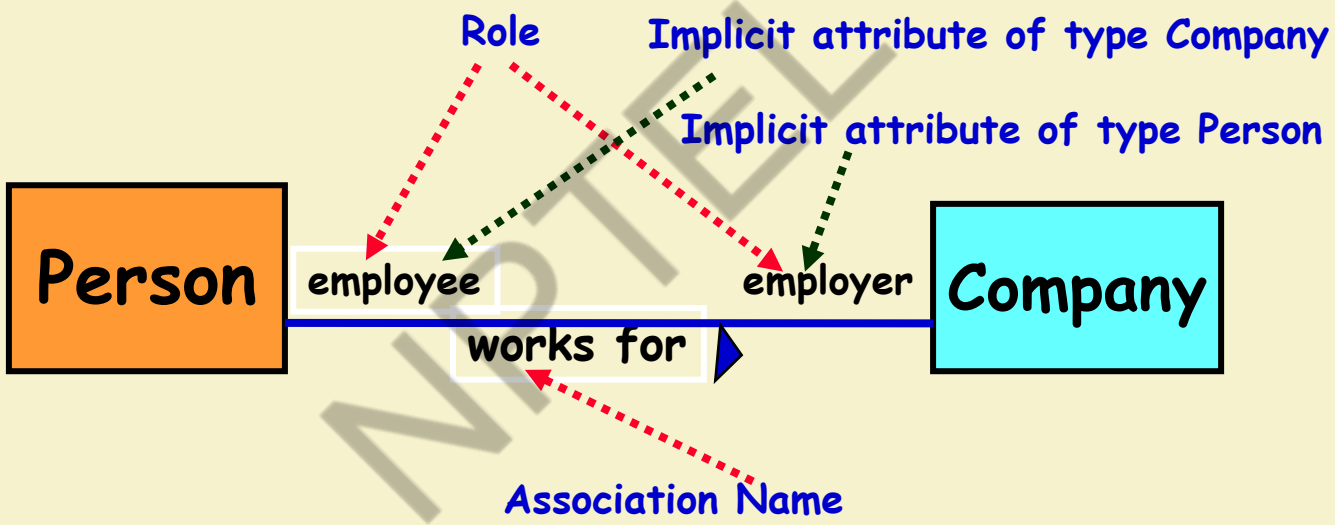


Computer Network: Object Diagram



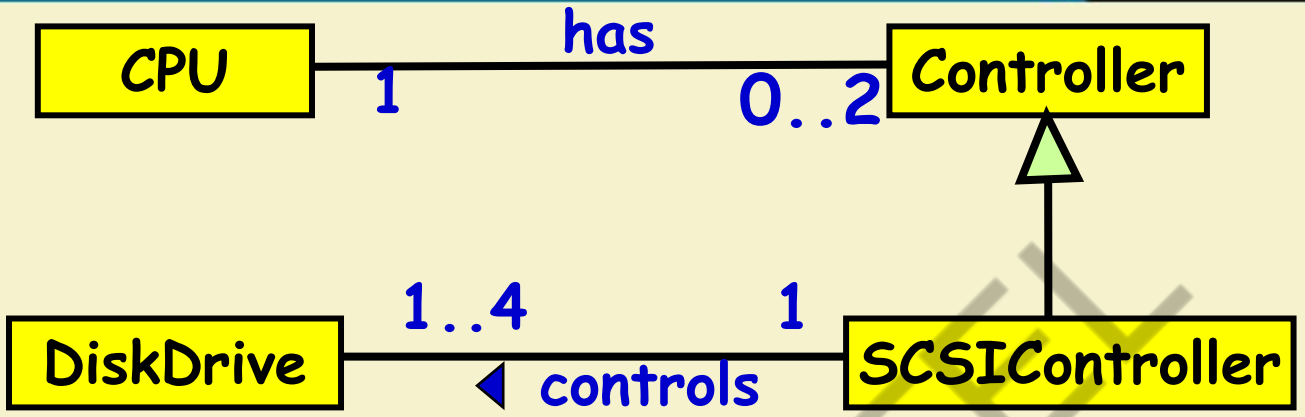
Association Exercise 1

- A Person works for a Company.



Observe: Implicit bidirectional navigation

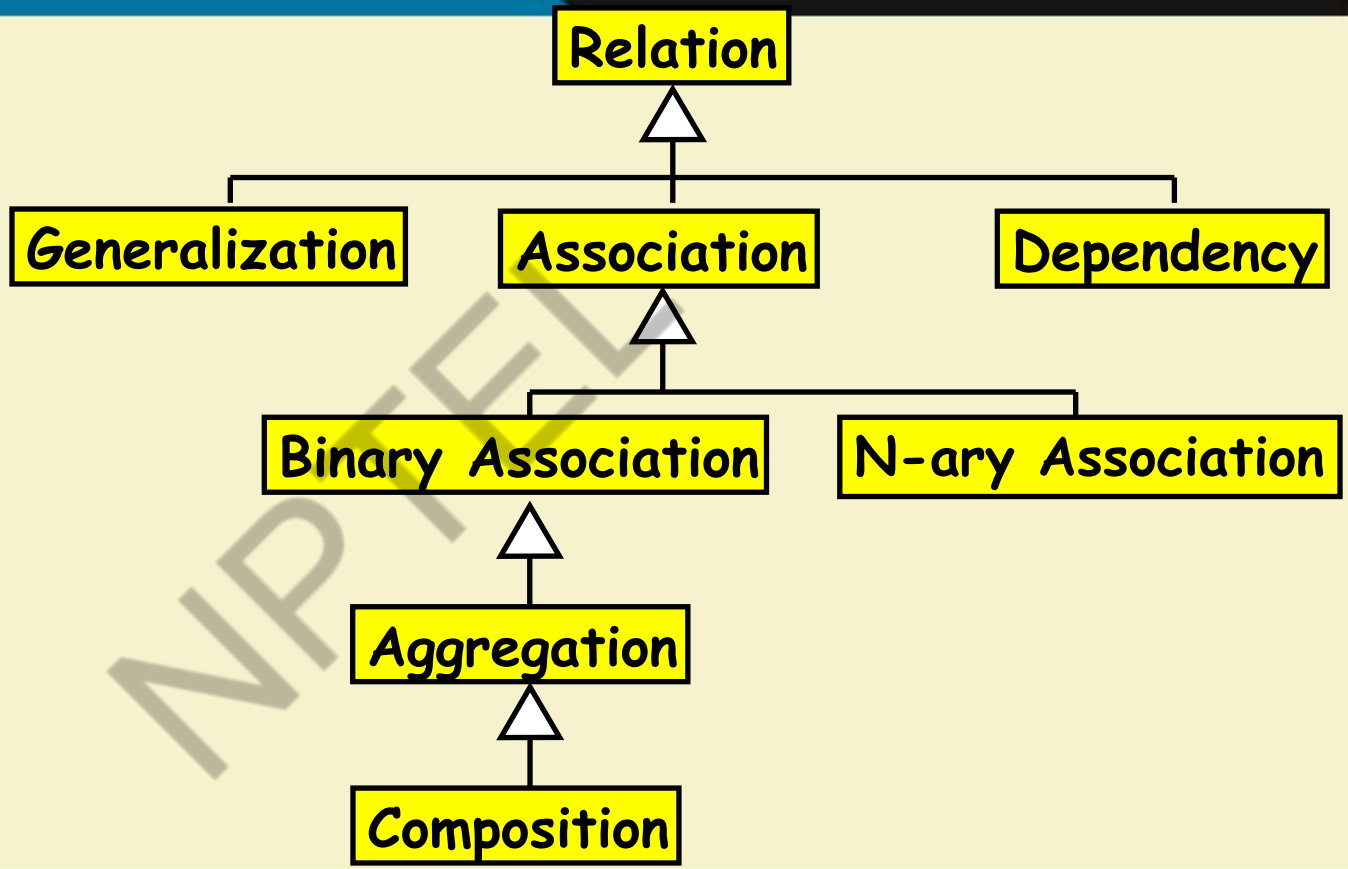
Implementation?



Quiz: Read and understand UML class diagram

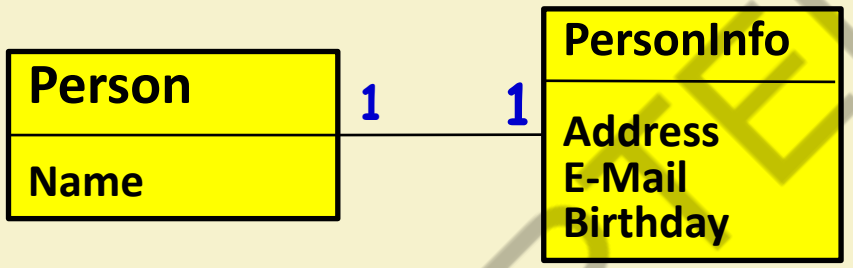
- 1 CPU has 0 to two Controllers
- 1-4 DiskDrives controlled by 1 SCSIController
- SCSIController is a (specialized) Controller

Types of Class Relationships

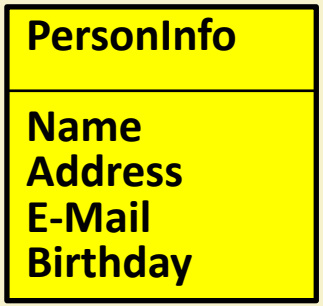


Overdoing Associations

- Avoid unnecessary Associations



Avoid This…

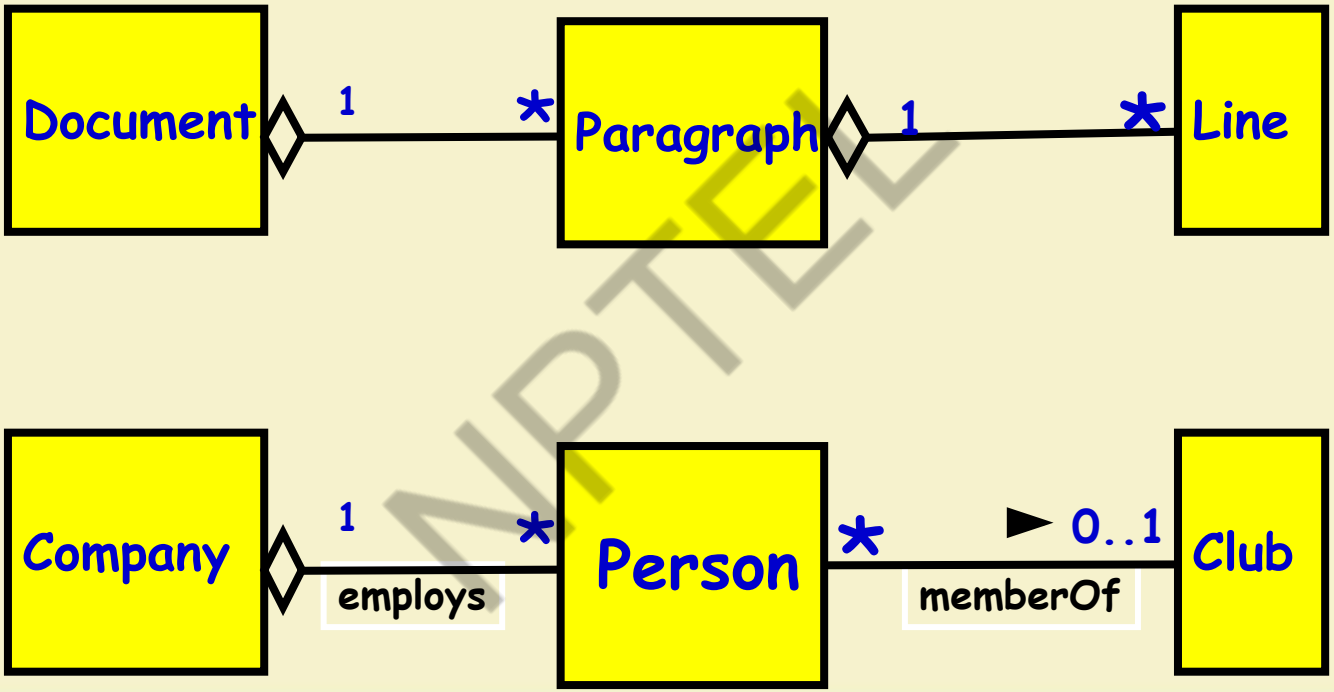


Do This

Aggregation Relationship

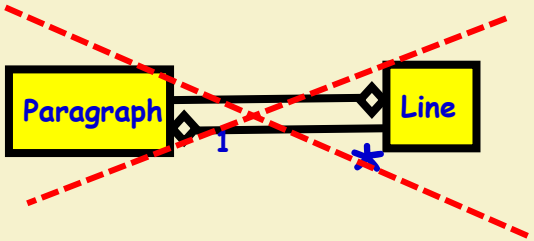
- Represents whole-part relationship
- Represented by a diamond symbol at the composite end.
- Usually creates the components:
  - Also, **aggregate usually invokes the same operations of all its components.**
  - This is in contras to plain association
- Usually owner of the components:
  - But can share with other classes

Aggregation Relationship



Aggregation cont...

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



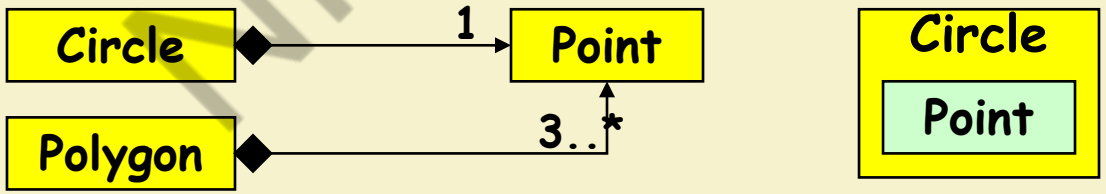
Aggregation vs. Inheritance

Cont…

- **Inheritance:**
  - Different object types with similar features.
  - Necessary semantics for similarity of behavior is in place.
- **Aggregation:**
  - Containment allows construction of complex objects.

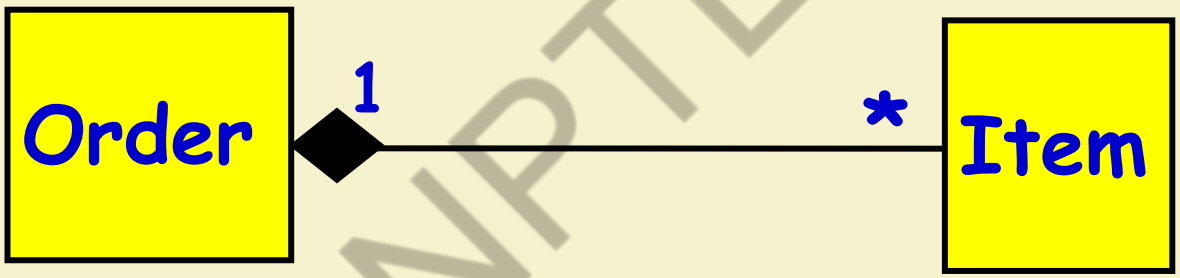
Composition

- 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.**

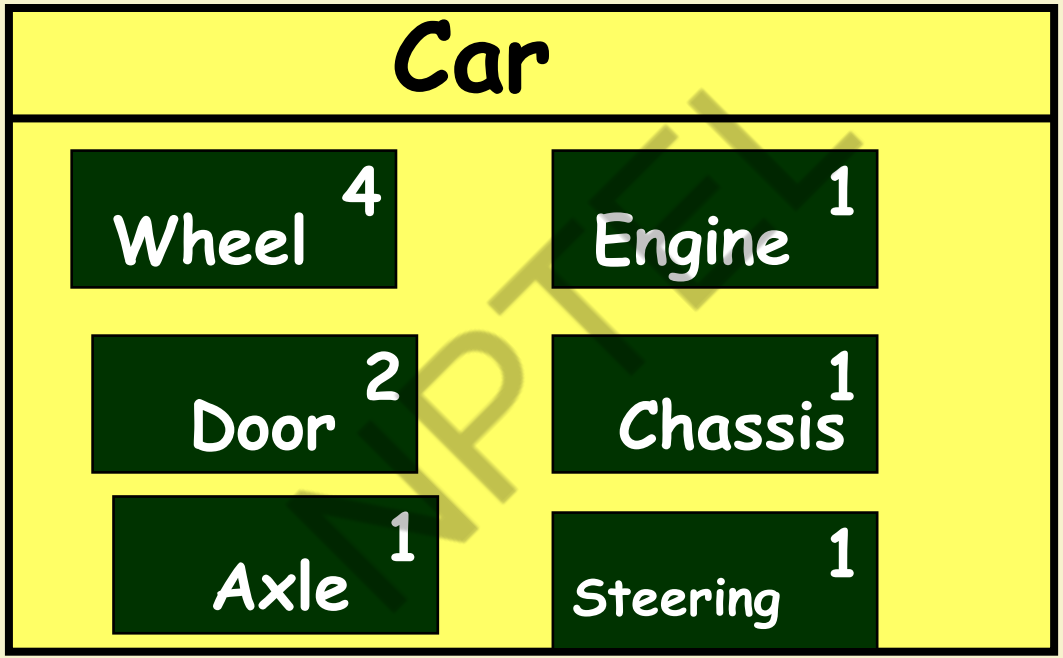


Composition Relationship

- Life of item is same as that of order

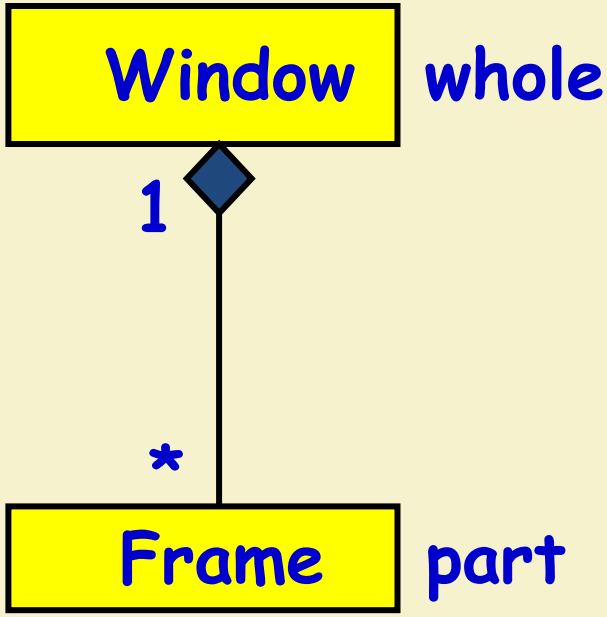


Composition: Alternate Notation



Composition

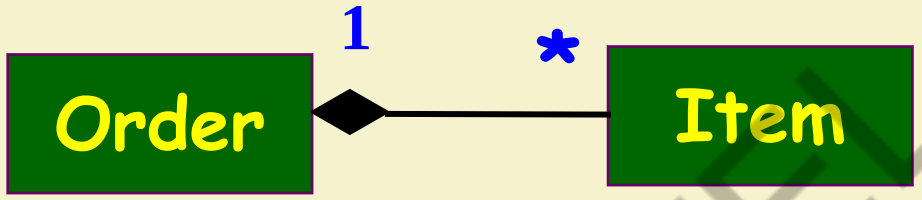
- An object may be a part of ONLY one composite at a time.
  - Whole is responsible for the creation and disposition of its parts.



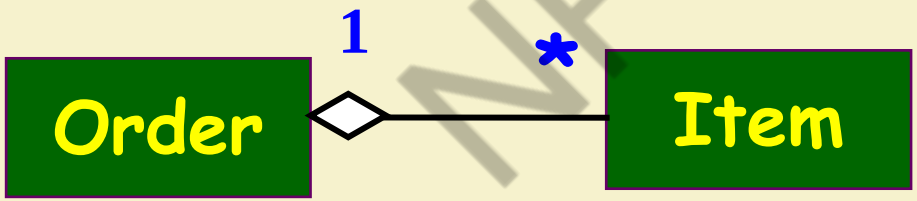
Aggregation vs. Composition

- 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.

Composition versus Aggregation



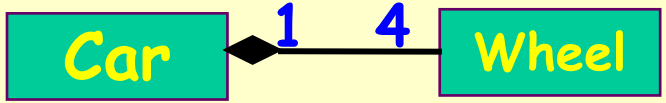
Composition



Aggregation

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();
    }
}
```



How to identify aggregation/composition?

- Lifetime of part is bound within lifetime of composite
  - –There is a create-delete dependency
- There is an obvious whole-part physical or logical assembly
- Some properties of composite propagate to parts (e.g., location)
- Operations applied to composite propagate to parts (e.g., destruction, movement, recording)

Class Dependency

Dependency

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

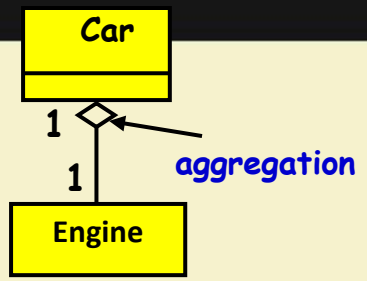
Type of dependency	Stereotype	Description
Abstraction	«abstraction»	Dependency of concrete class on its abstract class.
Binding	«bind»	Binds template arguments to create model elements from templates.
Realization	«realize»	Indicates that the client model element is an implementation of the supplier model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

Association Vs. Aggregation

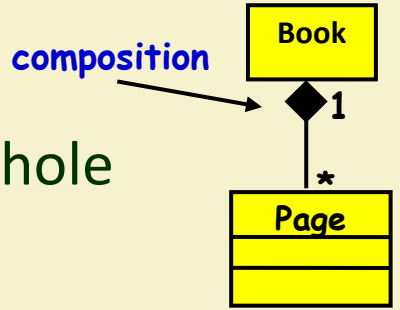
- Is aggregation an association?
- Is composition an aggregation?

Association Types

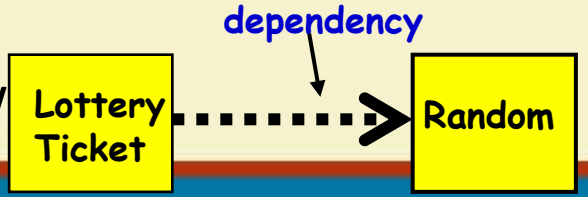
- **aggregation:** "is part of"
  - Symbolized by empty diamond



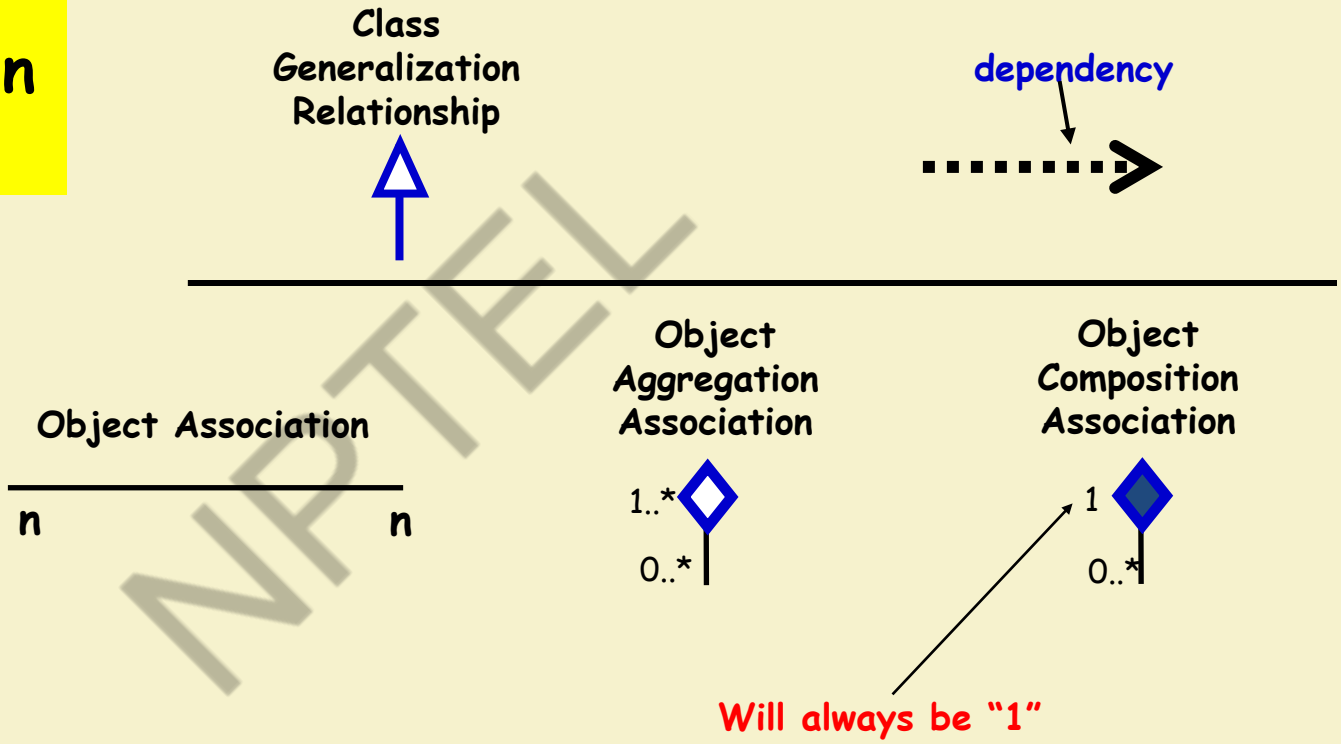
- **composition:** is made of
  - Stronger version of aggregation
  - The parts live and die with the whole
  - Symbolized by a filled diamond



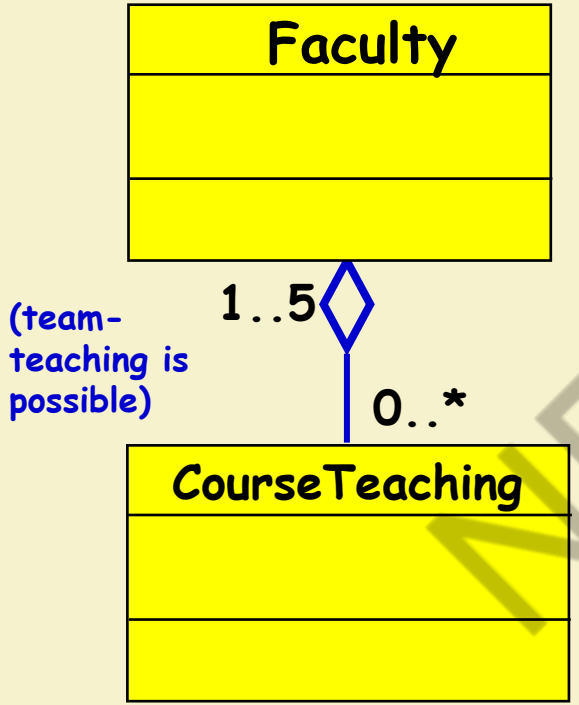
- **dependency:** Depends on
  - Represented by dotted arrow



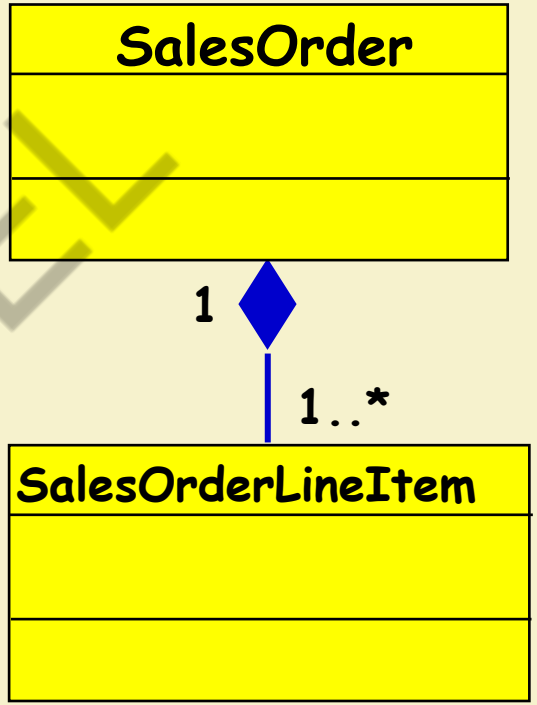
UML Class Relation Notation Summary



Aggregation



Composition



Class Relation Hints

- **Composition**
  - B is a permanent part of A
  - A contains B
  - A is a permanent collection of Bs
- **Subclass / Superclass**
  - A is a kind of B
  - A is a specialization of B
  - A behaves like B
- **Association (Collaboration)**
  - A delegates to B
  - A needs help from B
  - A and B are peers.

Class Diagram Inference Based on Text Analysis (based on Dennis, 2002)

- A common noun implies a class e.g. Book
- A proper noun implies an object (instance of a class): CSE Dept, OOSD, etc.
- An adjective implies an attribute e.g. price of book
- A "doing" verb implies an operation
  - Can also imply a relationship e.g. student issues Book
- A "having" verb implies an aggregation relationship
- A predicate or descriptive verb phrase elaborates an operation e.g. ISBN numbers are integers
- An adverb implies an attribute of an operation e.g. fast loading of image…

- Faculty & student
- Hospital & doctor
- Door & Car
- Member & Organization
- People & student
- Department & Faculty
- Employee & Faculty
- Computer Peripheral & Printer
- Account & Savings account

Identify Class Relations

- A square is a polygon
- Shyam is a student
- Every student has a name
- 100 paisa is one rupee
- Students live in hostels
- Every student is a member of the library
- A student can renew his borrowed books
- The Department has many students

Identify Classes & Relations

- A country has a capital city
- A dining philosopher uses a fork
- A file is an ordinary file or a directory file
- Files contain records
- A class can have several attributes
- A relation can be association or generalization
- A polygon is composed of an ordered set of points
- A person uses a computer language on a project

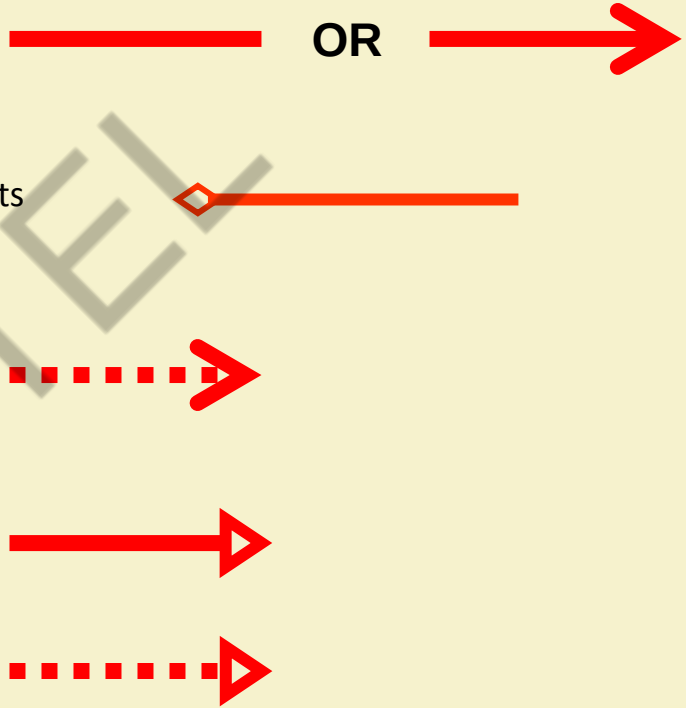
Identify Classes & Relations

- Describes static structure of a system
- Main constituents are classes and their relationships:
  - –**Generalization**
  - –**Aggregation**
  - –**Association**
  - –Various kinds of **dependencies**

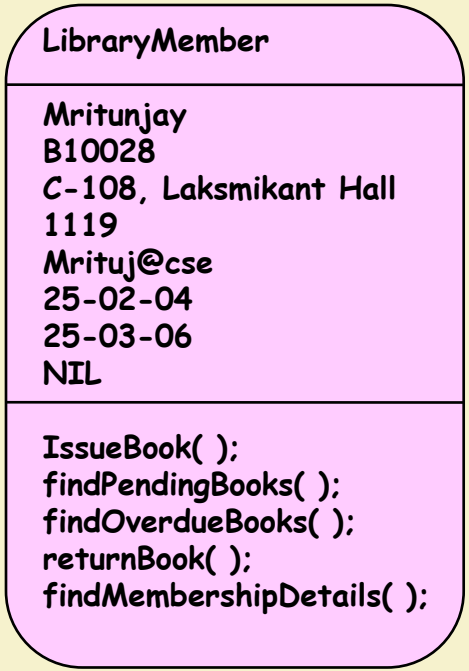
Class Diagram: Recap

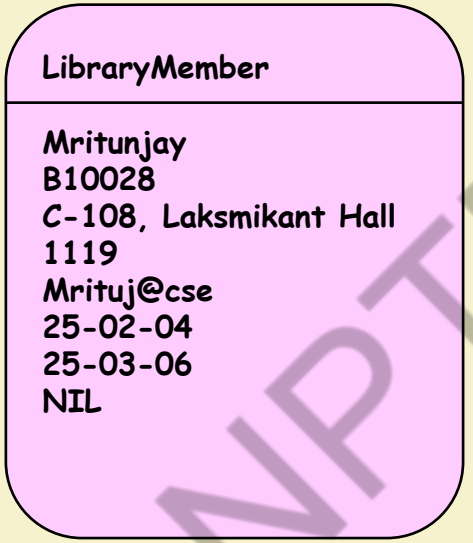
Summary of Relationships Between Classes

- **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



Object Diagram





Different representations of the LibraryMember object

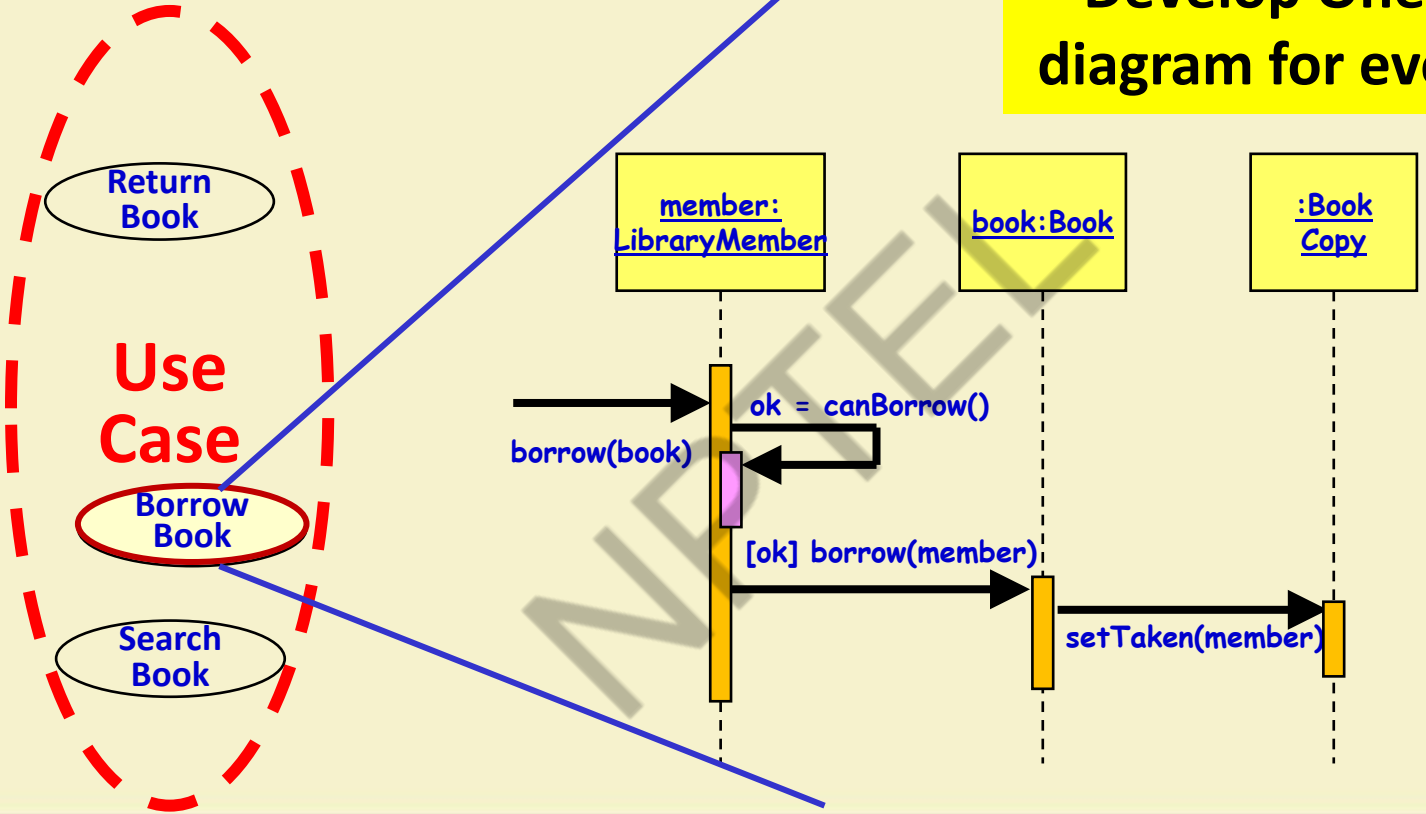
Interaction Diagram

- **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.**

A First Look at Sequence Diagrams

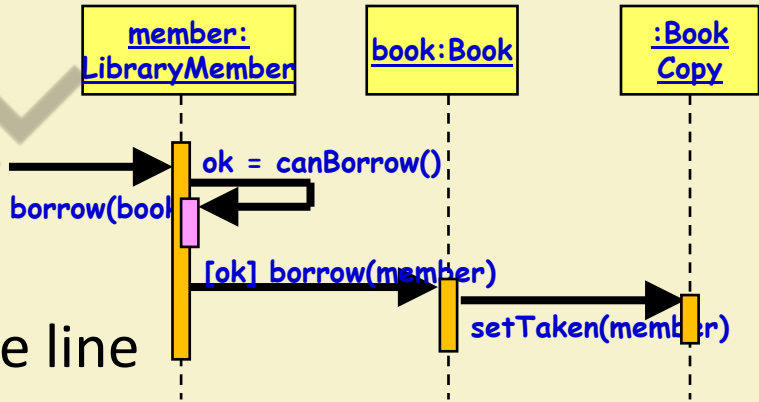
- **Captures how objects interact with each other:**
  - **To realize some behavior (use case execution).**
- Emphasizes time ordering of messages.
- Can model:
  - Simple sequential flow, branching, iteration, recursion, and concurrency.

Develop One Sequence diagram for every use case



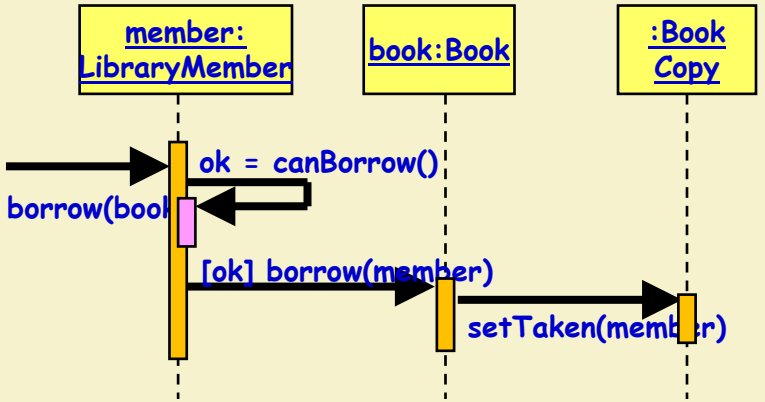
Sequence Diagram

- 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

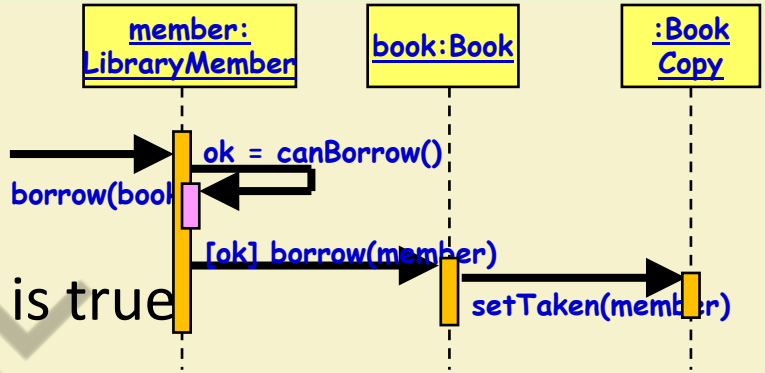


Sequence Diagram Cont…

- 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 (*)



- **iteration marker** *[for all objects]
- **[condition]**
  - message is sent only if the condition is true
- **self-delegation**
  - a message that an object sends to itself
- **Loops and conditionals:**
  - **UML2 uses a new notation called interaction frames to support these**

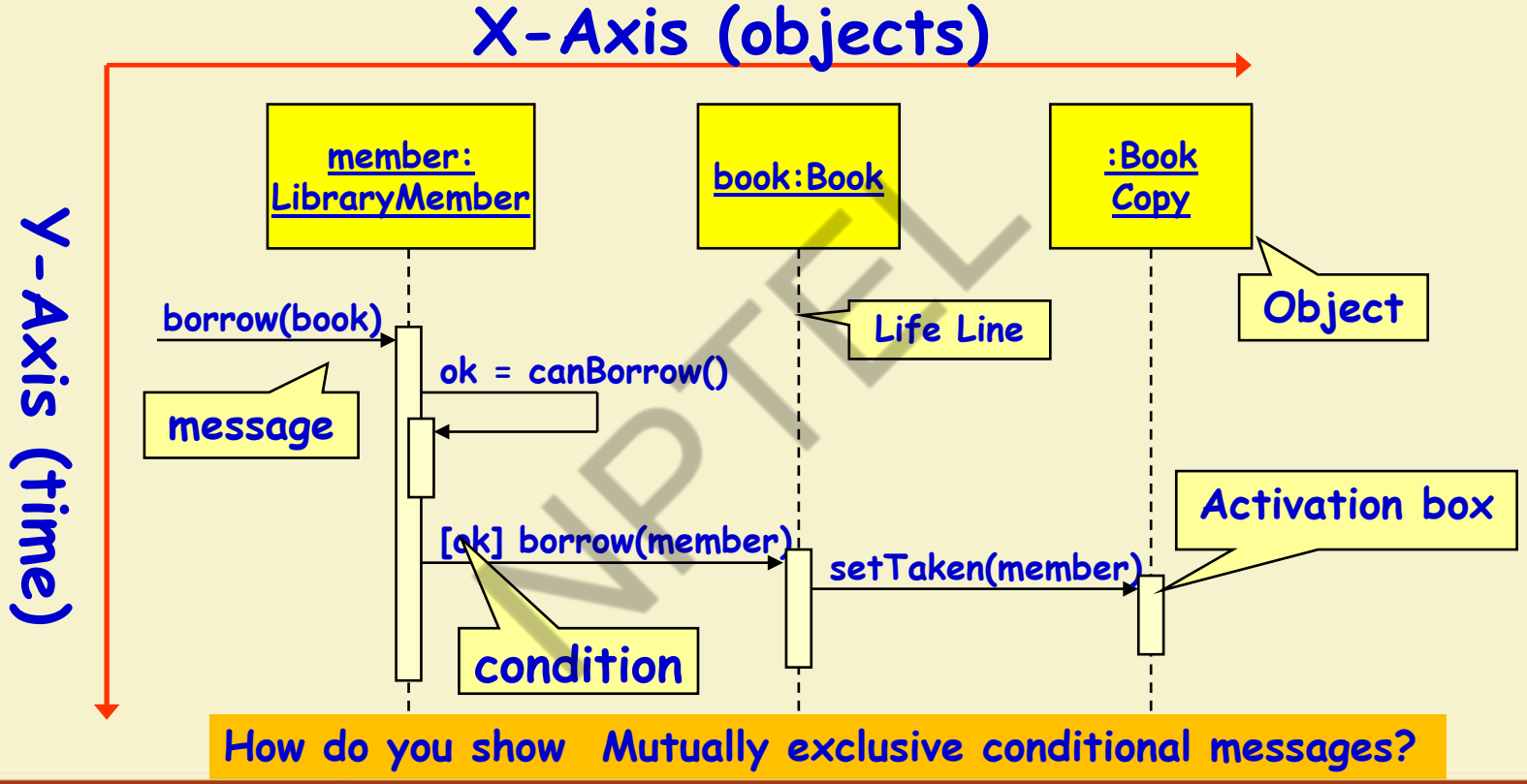


Gist of Syntax

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.**

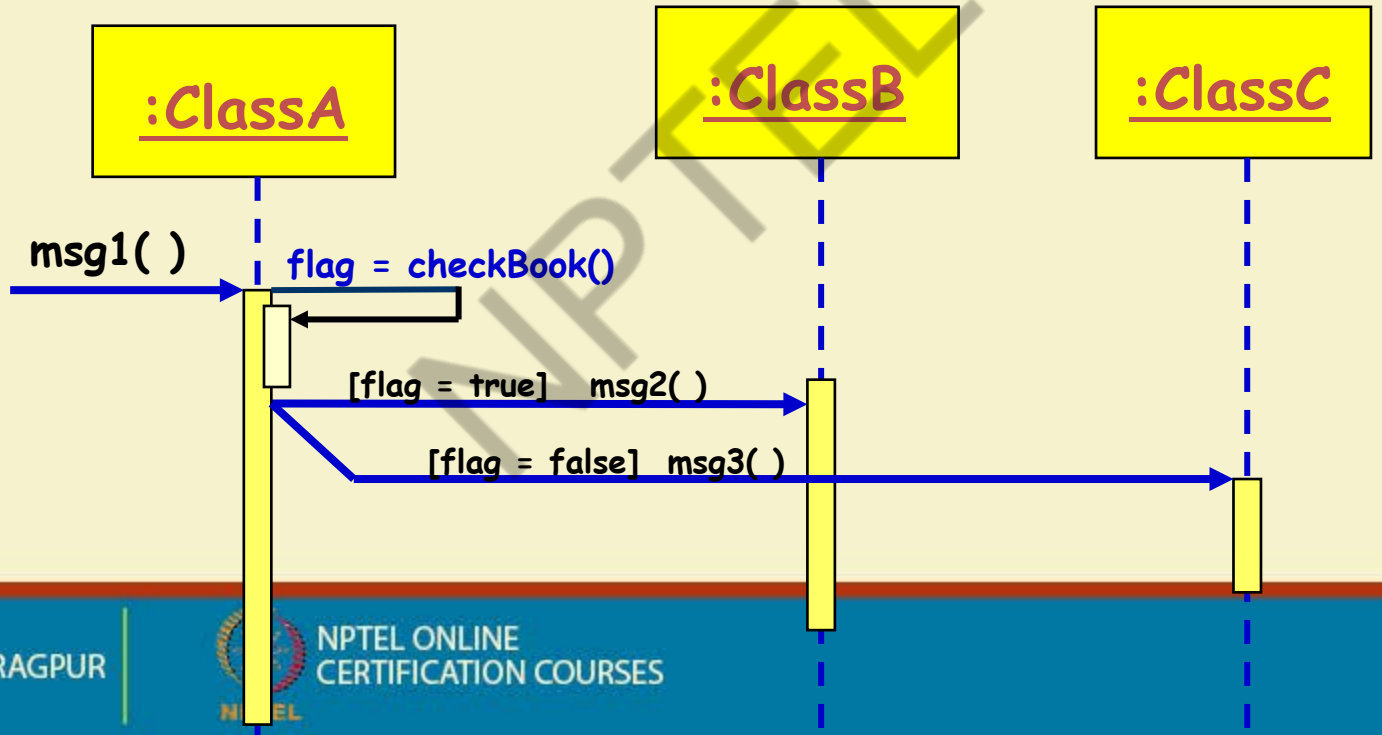
Elements of A Sequence Diagram

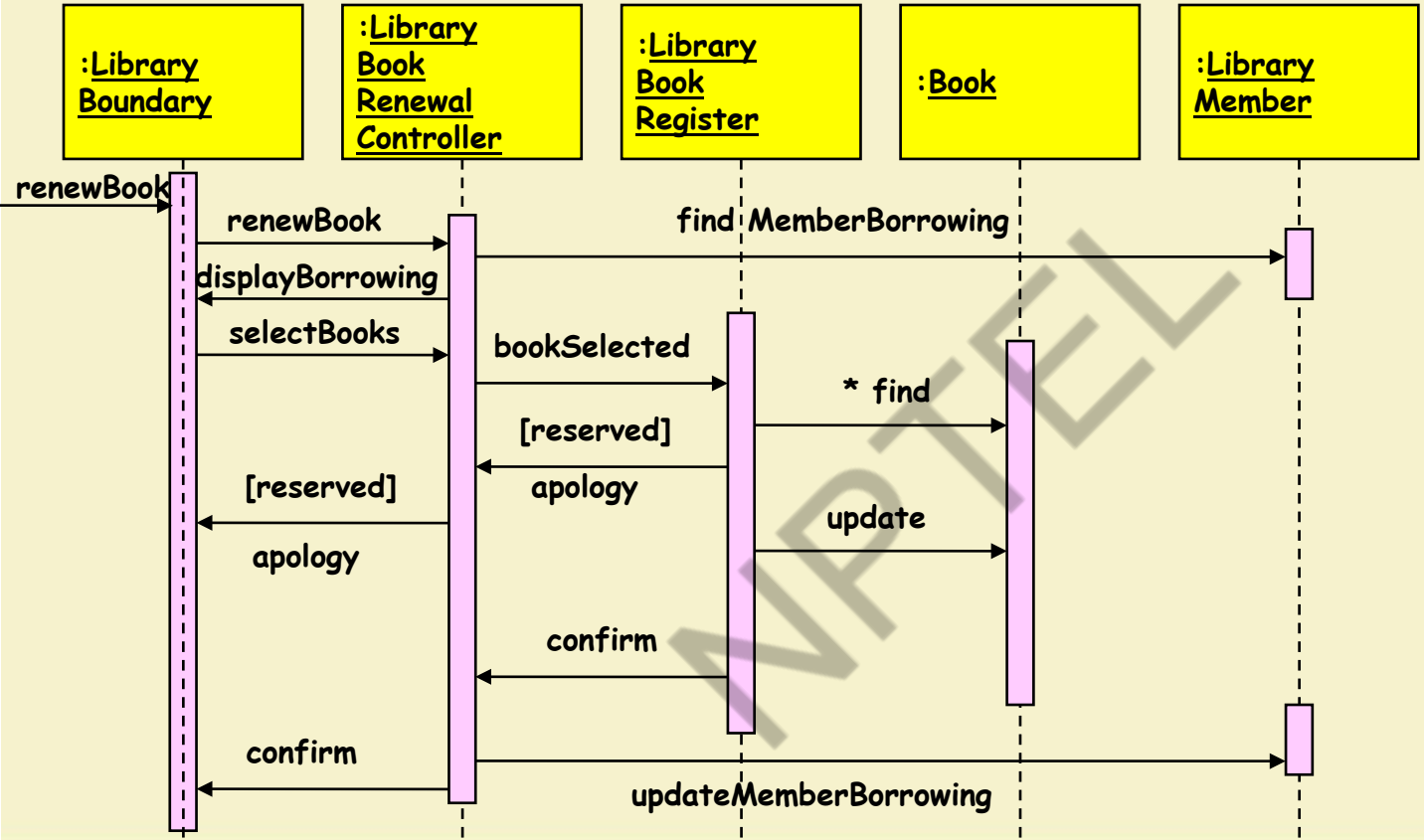


How do you show Mutually exclusive conditional messages?

Sequence Diagrams

How to represent Mutually exclusive conditional invocations? If book is available, invoke msg2 on ClassB else invoke msg3 on classC,



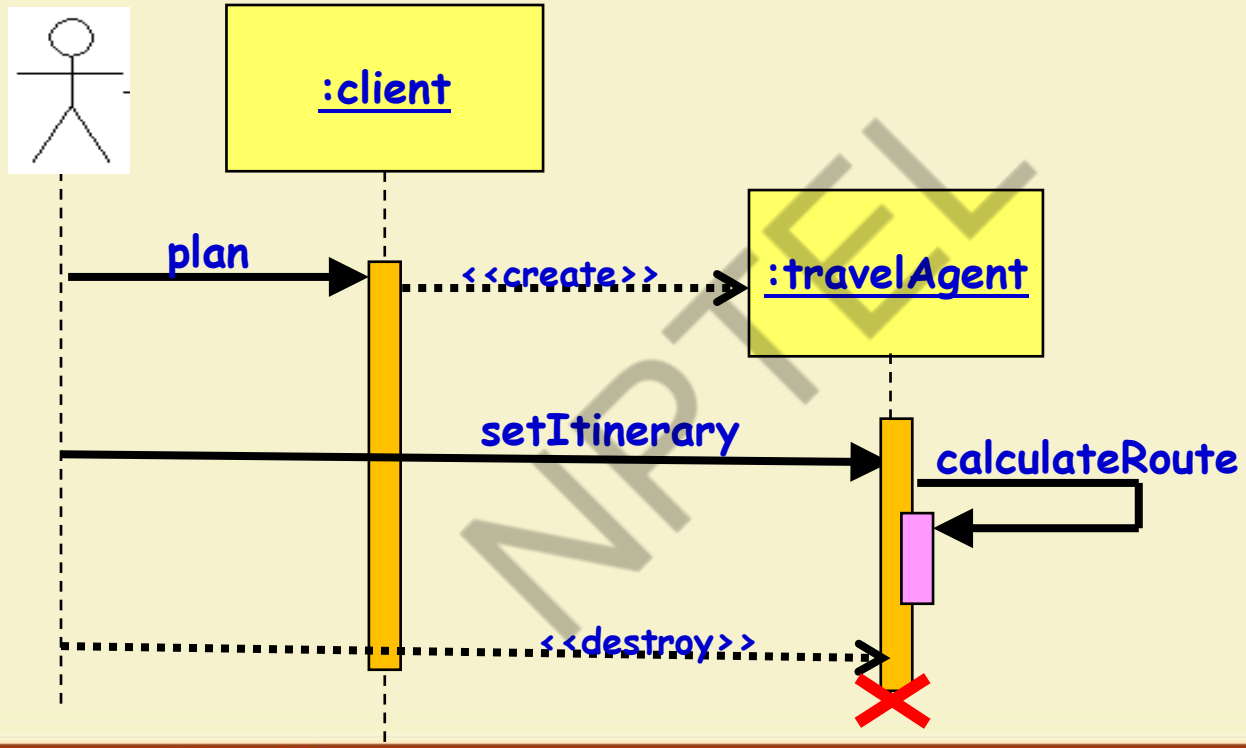


Sequence Diagram for the renew book use case

Example: Develop Sequence Diagram

- A user can use a travel portal to plan a travel
- When the user presses the plan button, a travel agent applet appears in his window
- Once the user enters the source and destination,
  - The travel agent applet computes the route and displays the itinerary.
  - Travel agent widget disappears when user presses close button

Example: Solution

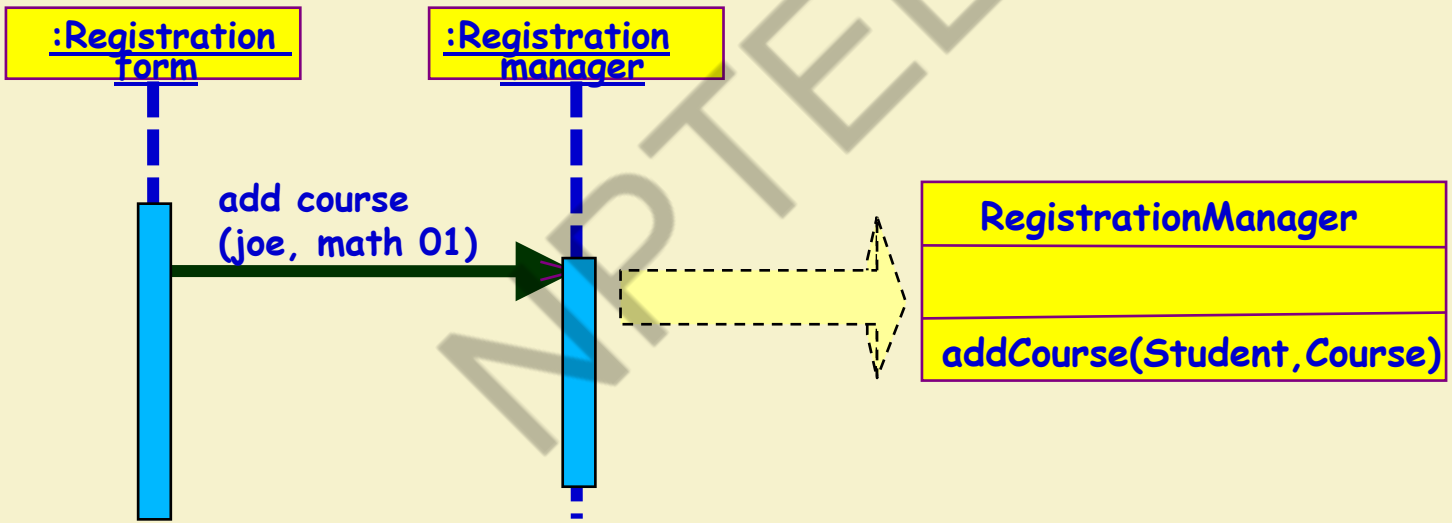


Return Values

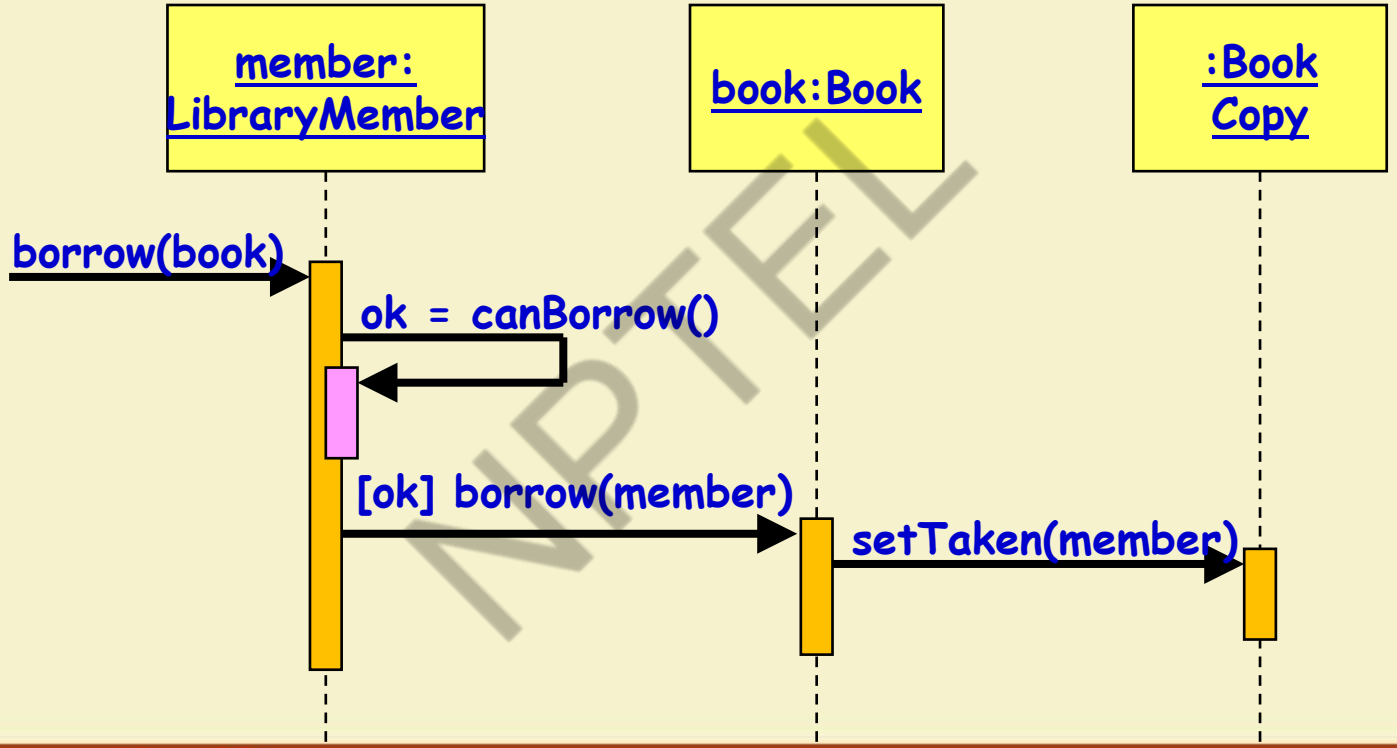
- Optionally indicated using a dashed arrow:
  - Label indicates the return value.
  - Don't need when it is obvious what is being returned, e.g. getTotal()
- **Model a return value only when you need to refer to it elsewhere:**
  - **Example:** A parameter passed to another message.

Method Population in Classes

- Methods of a class are determined from the interaction diagrams…

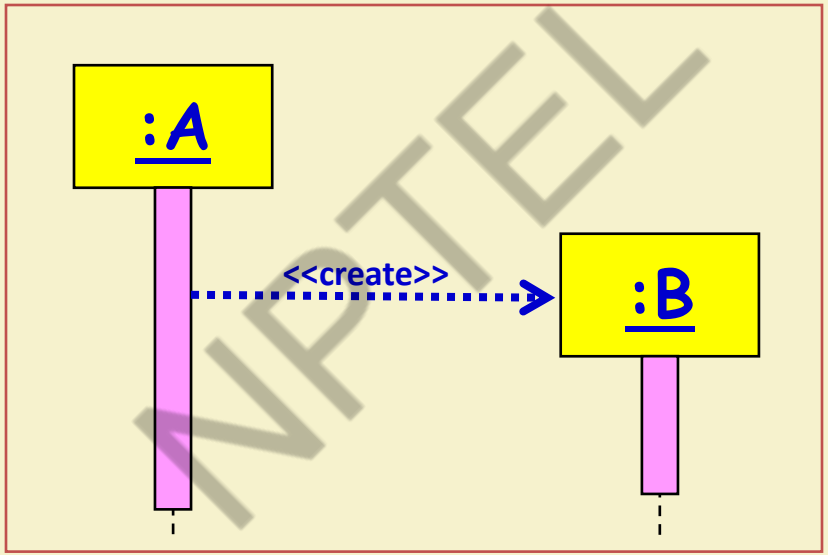


Example Sequence Diagram: Borrow Book Use Case



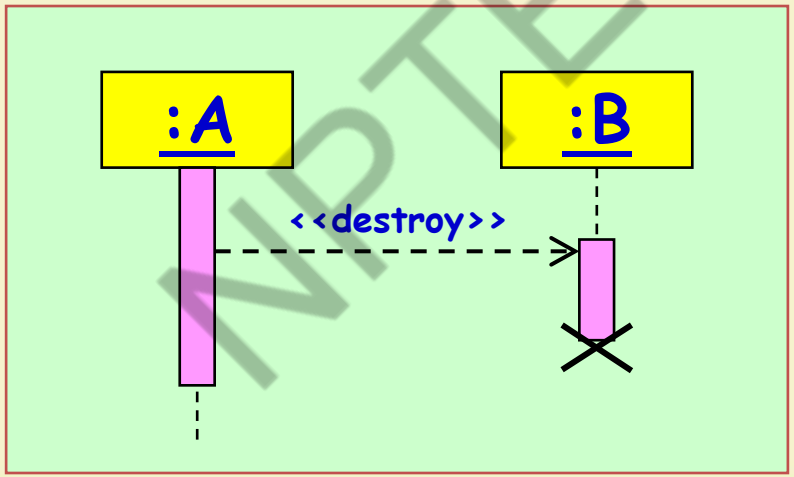
Object Creation

- An object may create another object via a **<<create>>** message.



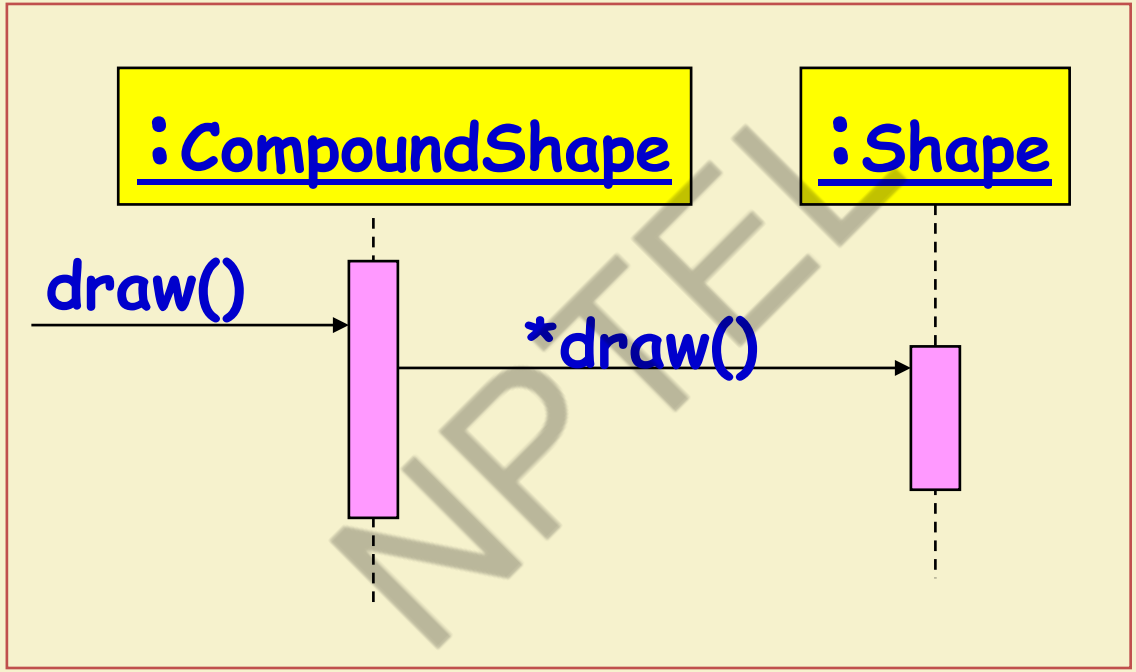
Object Destruction

- An object may destroy another object via a <<destroy>> message.
  - An object may also destroy itself.
- **But, how do you destroy an object in Java?**
  - **Avoid modeling object destruction unless memory management is critical.**

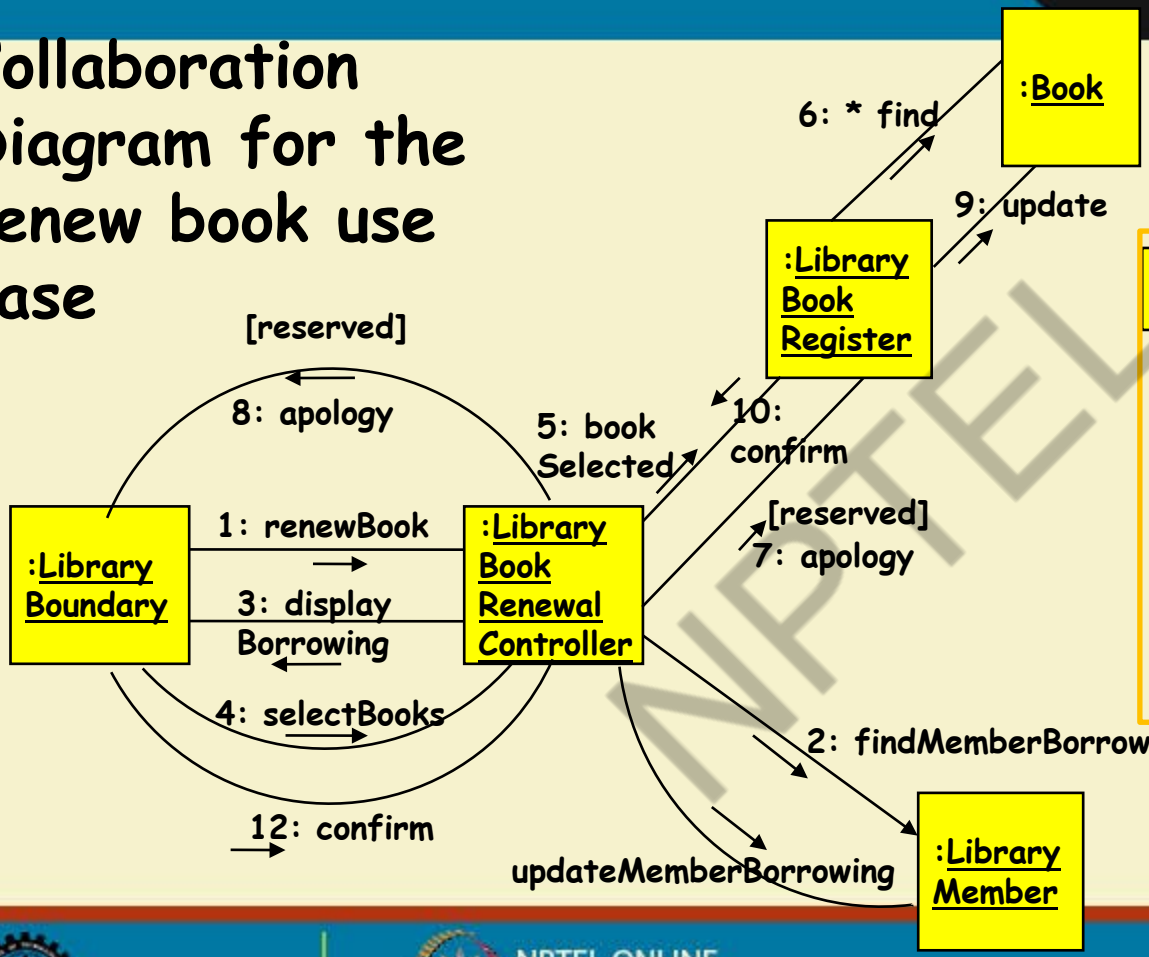


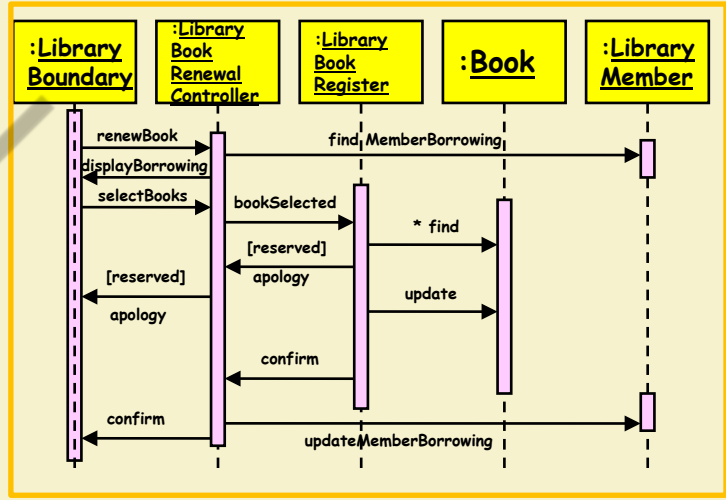
Control Information

Iteration example UML 1.x:

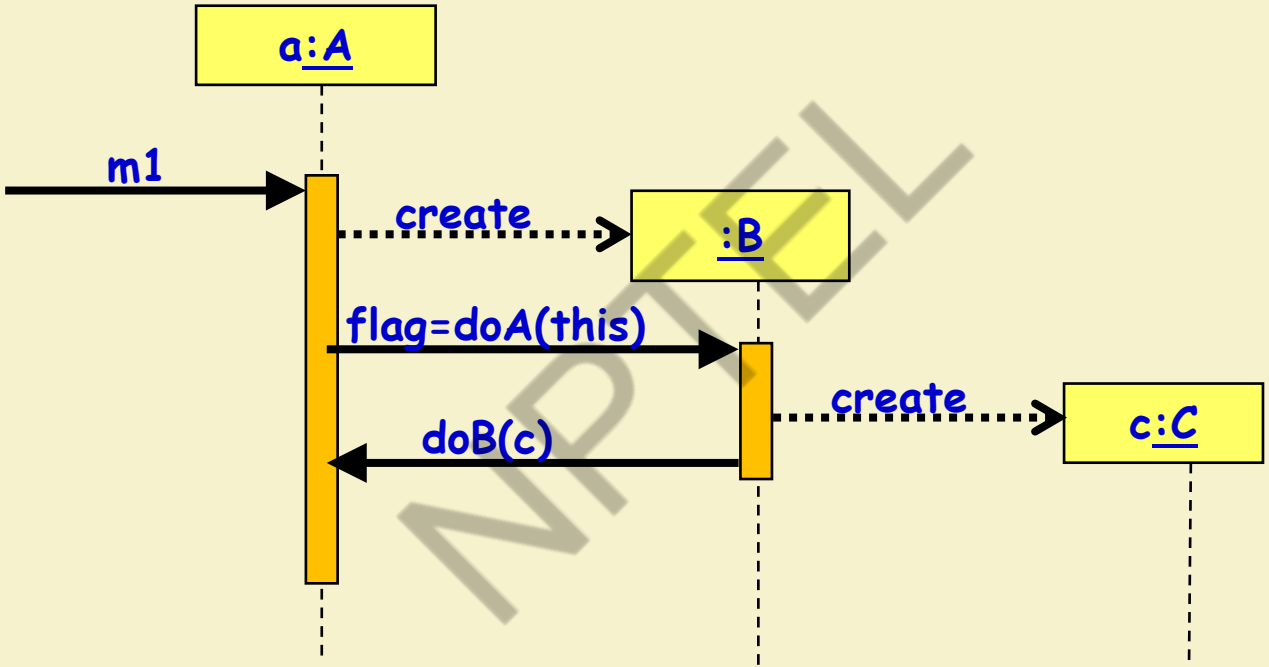


Collaboration Diagram for the renew book use case





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; }
}
```

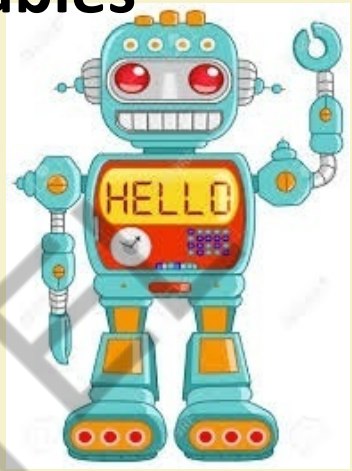
State Machine Diagrams

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.

Robot: State Variables

- Movement: On, OFF
- Direction: Forward, Backward, left, Right
- Left hand: Raised, Down
- Right hand: Raised, down
- Head: Straight, turned left, turned right
- Headlight: On, Off
- Turn: Left, Right, Straight

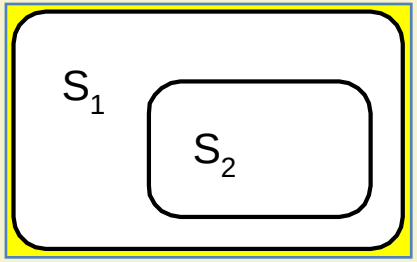


How many states in the state machine model?

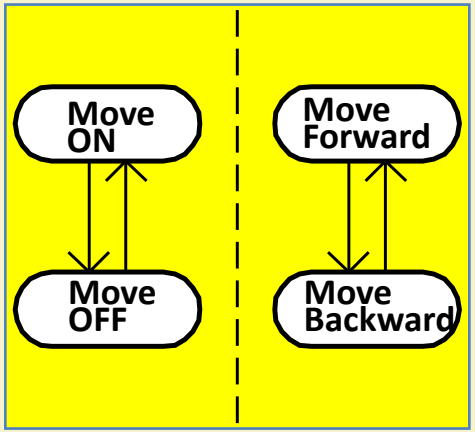
FSM: exponential rise in number of states with state variables

Features of Statecharts

- Two major features are introduced :
  - **nested** states
  - **concurrent** states
- Many other features have also been added:
  - History state
  - broadcast messages
  - actions on state entry, exit
  - …



Nested State Diagrams



Concurrent State Diagrams

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

How to Encode an FSM?

- Three main approaches:
  - **Doubly nested switch in loop**
  - **State table**
  - **State Design Pattern**

3 Principal Ways

- **Doubly nested switch in loop:**
  - Global variables store state --- Used as switch
  - Event type is discriminator in second level switch
  - Hard to handle concurrent states, composite state, history, etc.
- **State table:** Straightforward table lookup
- **Design Pattern:**
  - States are represented by classes
  - Transitions as methods in classes

Doubly Nested Switch Approach

```
int state, event; /* state and event are variables */
while(TRUE){
    switch (state){
        Case state1: switch(event){
                        case event1:
                                    state=state2; etc…; break;
                        case event2:…
                        default:
                                    }
        Case state2: switch(event){…
                                    }
    ....
    }}
```

State Table Approach

- From the state machine, we can set up a state transition table with a column for the actions associated with each transition

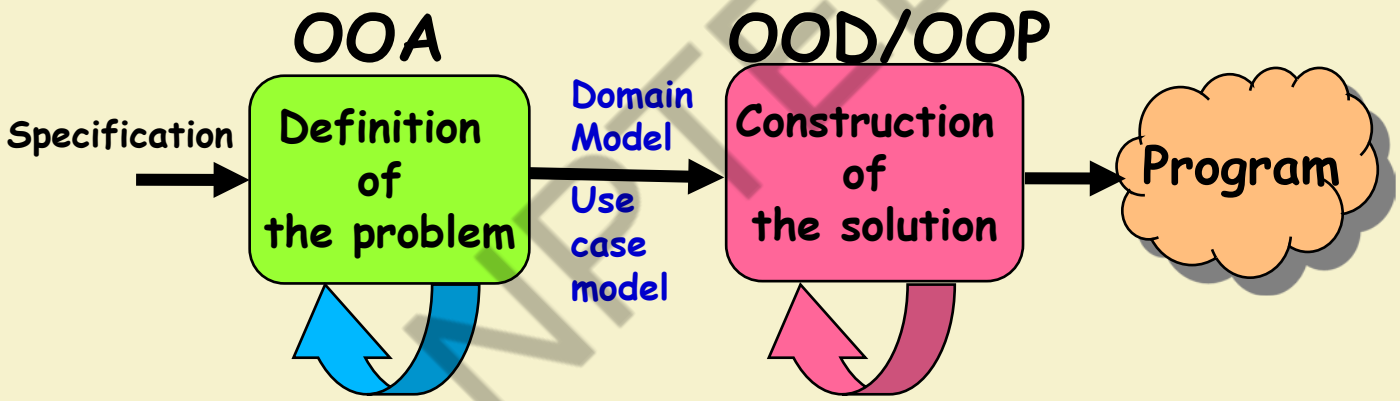
Present state	Event	Next state	Actions
Light_off	e1	Light_off	none
	e2	Light_on	set red LED flashing
Light_on	e1	Light_on	none
	e2	Light_off	reset red LED flashing

An Object-Oriented Design Process

- We discuss a design process based to a large extent on Larman's approach:
  - Also synthesizes features from various other methodologies.
- From requirements specification, an initial model is developed (OOA):
  - **Analysis model is iteratively refined into a design model**
- Design model is implemented using an OO language.

OOAD Process

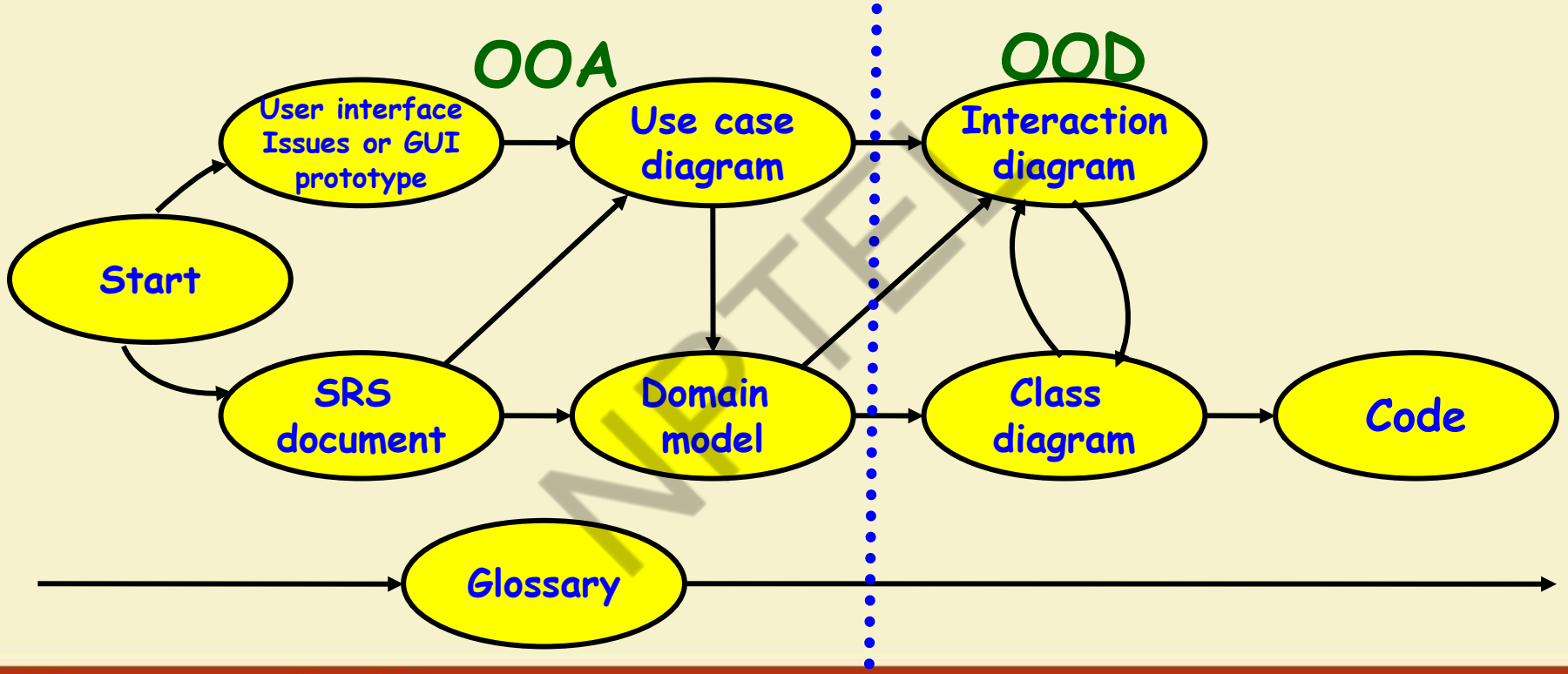
Iterative and Incremental



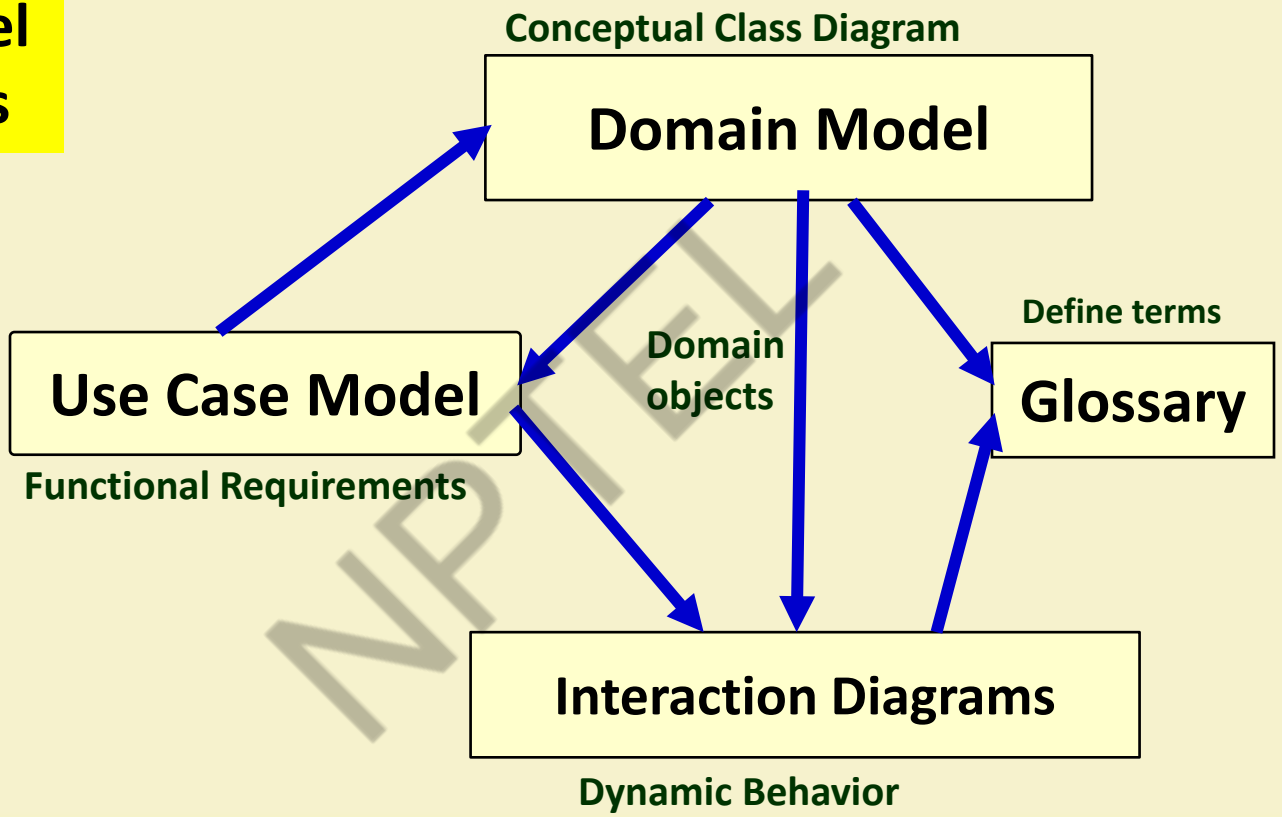
OOA versus OOD?

- **Analysis:**
  - An elaboration of requirements.
  - Independent of any specific implementation
- **Design:**
  - A refinement of the analysis model.
  - Takes implementation constraints into account

Design Process



Domain Model Relationships



Domain Modelling

- Represent concepts or objects appearing in the problem domain.
  - Also capture object relationships.
- Three types of objects are identified:
  - **Boundary objects**
  - **Entity objects**
  - **Controller objects**

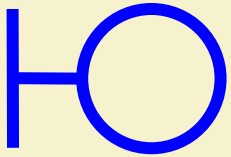
Three different stereotypes are used to represent classes :

<<boundary>>, <<control>>, <<entity>>.

Class Stereotypes

Boundary

Cashier Interface

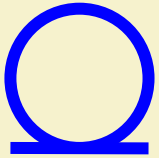


Control

Withdrawal manager

Entity

Account



Boundary Objects

- Handle interaction with actors:
  - **User interface objects**
- Often implemented as screens, menus, forms, dialogs etc.
- Do not perform processing:
  - But may validate input, format output, etc.

Entity Objects

- **Hold information over long term:**

  –e.g. Book, BookRegister

- Normally are dumb servers:

  – **Responsible for storing data, fetching data etc.**

  – **Elementary operations on data such as searching, sorting, etc.**

- Often appear as **nouns** in the problem description...

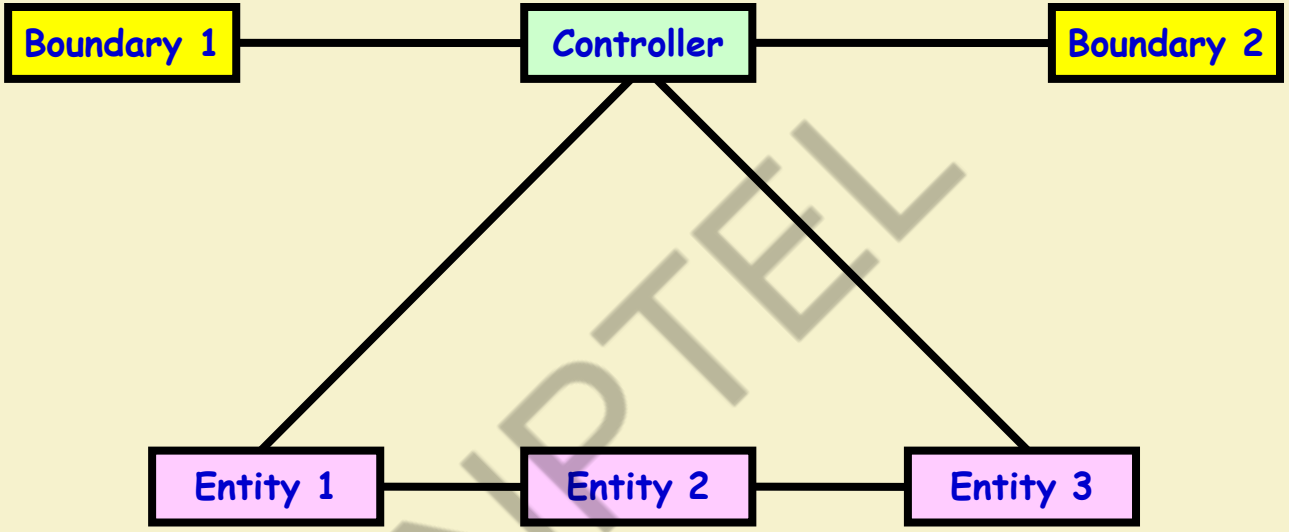
Controller Objects

- **Overall responsibility to realize use case behavior:**
  - Interface with the boundary objects
  - Coordinate the activities of a set of entity objects
- **Embody most of the business logic required for use case execution:**
- There can be more than one controller to realize a single use case

Controller Classes

- Controller classes coordinate, sequence, transact, and otherwise control other objects…
- In Smalltalk MVC mechanism:
  - These are called controllers

Use Case Realization



Realization of use case through the collaboration of Boundary, controller and entity objects