IIT KHARAGPUR

NPTEL ONLINE CERTIFICATION COURSES

NPTEL

Object-Oriented Design using UML

Rajib Mall

CSE Department

IIT KHARAGPUR

Introduction

- Object-oriented design (OOD) techniques are now extremely popular:
  - –Inception in early 1980’s and nearing maturity.
  - –Widespread acceptance in industry and academics.
  - –**Unified Modelling Language (UML) became an ISO standard (ISO/IEC 19501) in 2004.**

Object Modelling Using UML

- UML is a modelling language.
  - Not a system design or development methodology
- Used to document object-oriented analysis and design results.
- Independent of any specific design methodology.

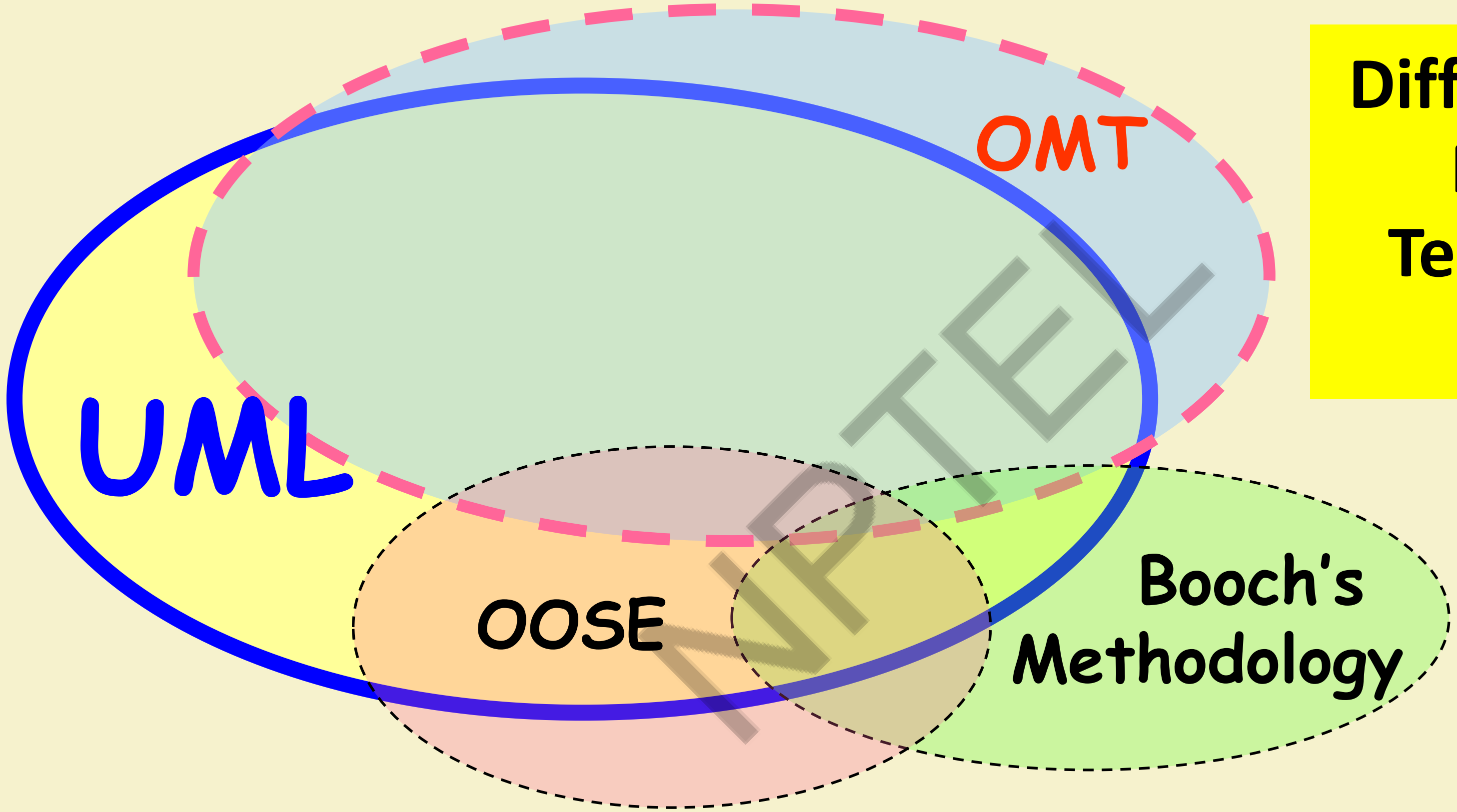
UML Origin

- OOD in late 1980s and early 1990s:
  - Different software development houses were using different notations.
  - **Methodologies were tied to notations.**
- UML developed in early 1990s:
  - To standardize the large number of object-oriented modelling notations that existed.

UML Lineology

- Based Principally on:
  - OMT [Rumbaugh 1991]
  - Booch's methodology[Booch 1991]
  - OOSE [Jacobson 1992]
  - Odell's methodology[Odell 1992]
  - Shlaer and Mellor [Shlaer 1992]

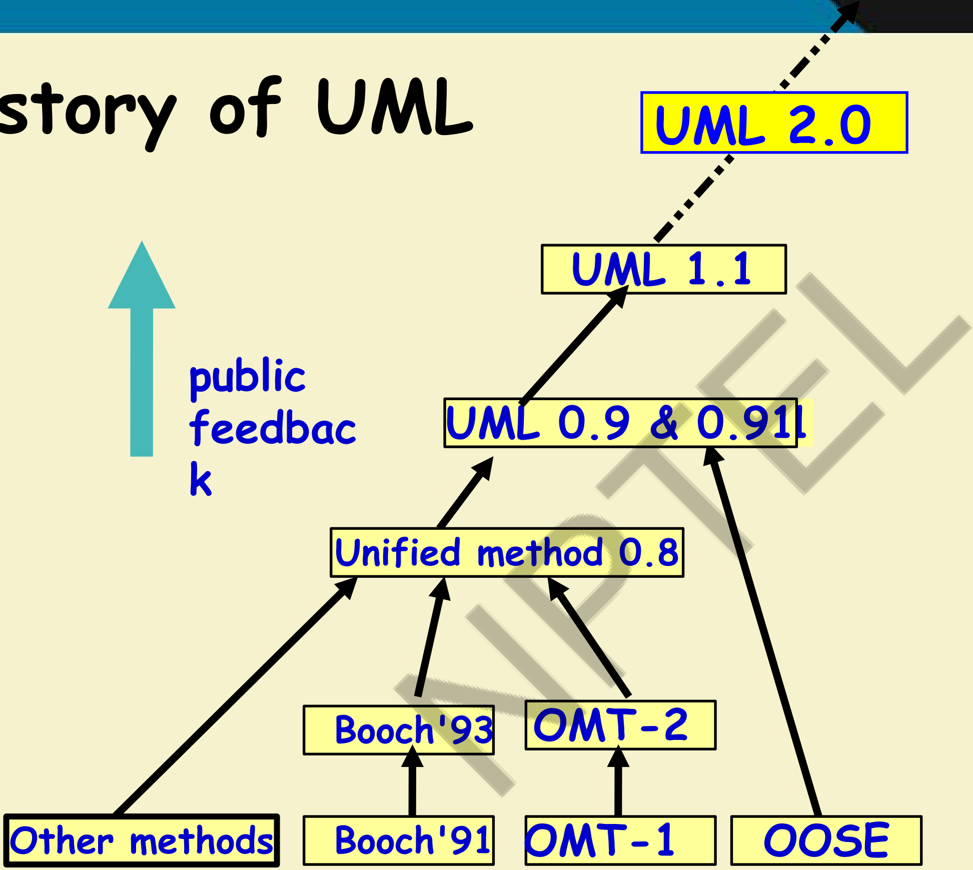
Different Object Modeling Techniques in UML

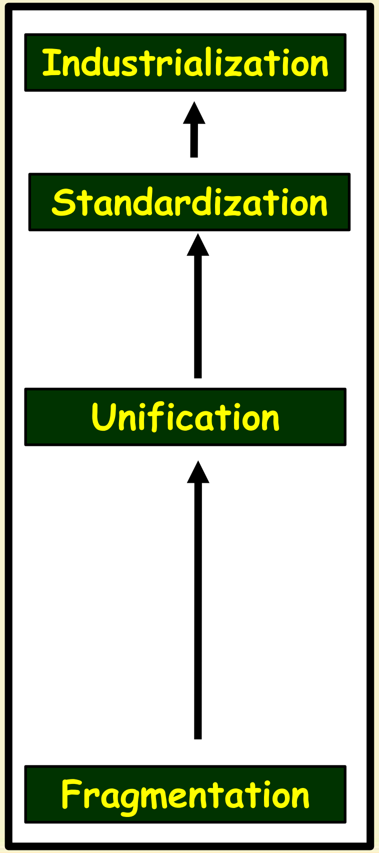


UML as A Standard

- Adopted by Object Management Group (OMG) in 1997.
- OMG is an association of industries
- Promotes consensus notations and techniques
- UML also being used outside software development area:
  - Example car manufacturing

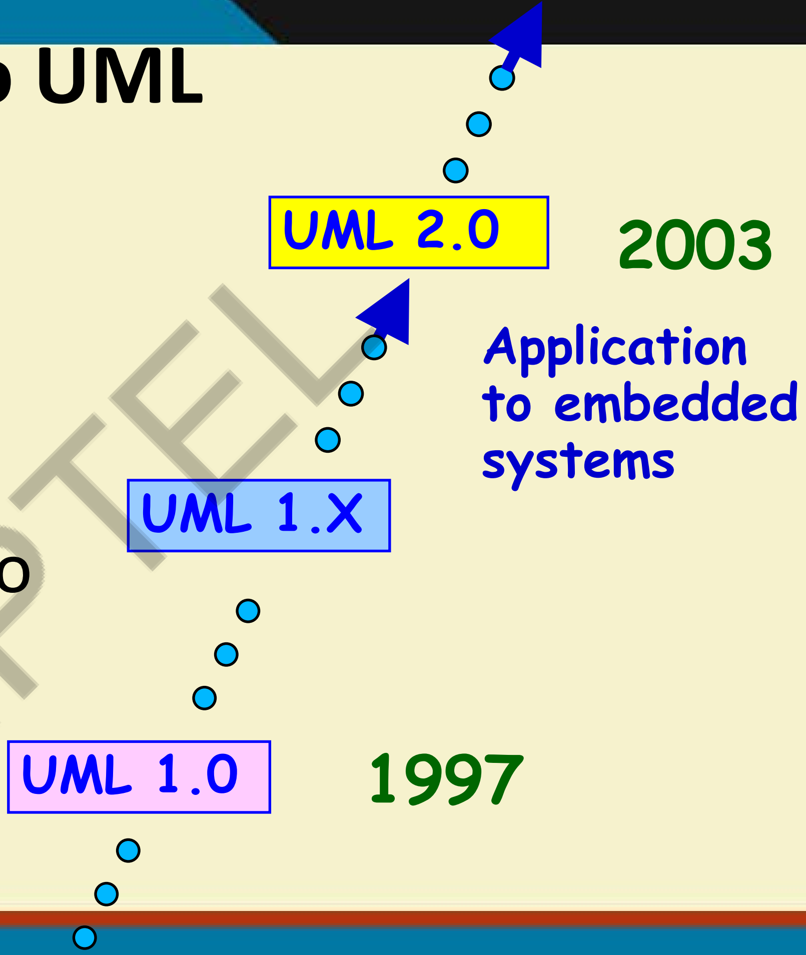
History of UML





Developments to UML

- UML continues to develop, due to:
  - Refinements
  - Making it applicable to new contexts



Why are UML Models Required?

- Modelling is an abstraction mechanism:
  - Capture only important aspects and ignores the rest.
  - Different models obtained when different aspects are ignored.
  - An effective mechanism to handle complexity.
- UML is a graphical modelling technique
- Easy to understand and construct

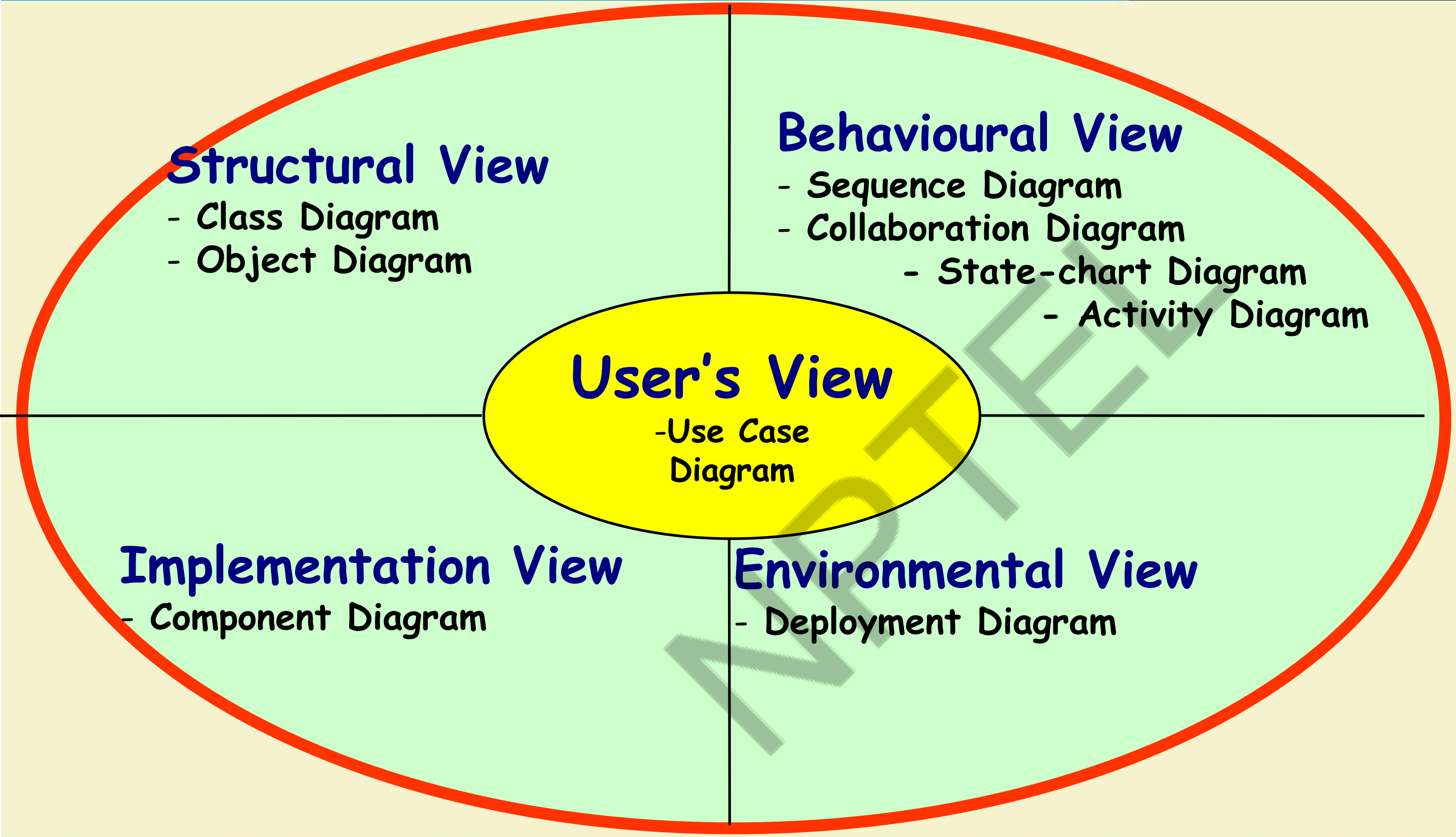
UML Diagrams

- Nine diagrams in UML1.x :
  - Used to capture 5 different views of a system.
- Views:
  - Provide different perspectives of a software system.
- Diagrams can be refined to get the actual implementation of a system.

- Views of a system:
  - User’s view
  - Structural view
  - Behavioral view
  - Implementation view
  - Environmental view

UML Model Views

Diagrams and views in UML



Structural Diagrams

- **Class Diagram**
  - set of classes and their relationships.
- **Object Diagram**
  - set of objects (class instances) and their relationships
- **Component Diagram**
  - logical groupings of elements and their relationships
- **Deployment Diagram**
  - set of computational resources (nodes) that host each component.

Behavioral Diagrams

- **Use Case Diagram**
  - high-level behaviors of the system, user goals, external entities: actors
- **Sequence Diagram**
  - focus on time ordering of messages
- **Collaboration Diagram**
  - focus on structural organization of objects and messages
- **State Chart Diagram**
  - event driven state changes of system
- **Activity Diagram**
  - flow of control between activities

Some Insights on Using UML

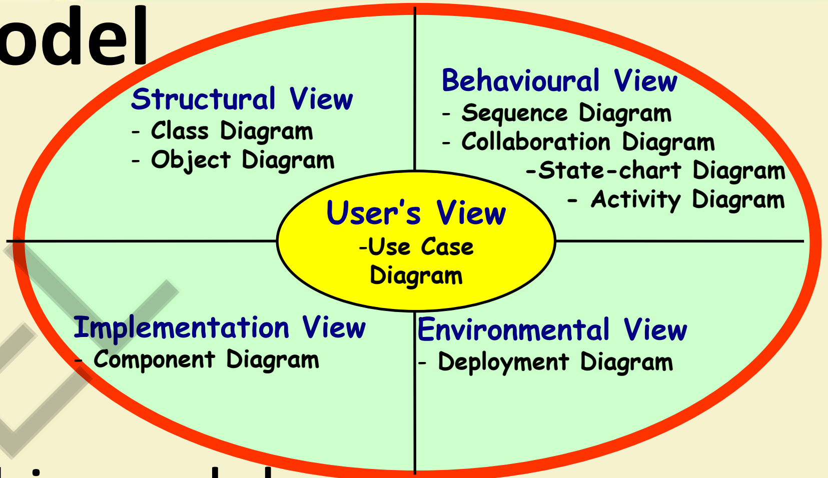
- “UML is a large and growing beast, but you don’t need all of it in every problem you solve…”
  – Martin Fowler
- “…when learning the UML, you need to be aware that certain constructs and notations are only helpful in detailed design while others are useful in requirements analysis …” Brian Henderson-Sellers

Are All Views Required for Developing A Typical System?

- NO
- For a simple system:
  - Use case diagram, class diagram and one of the interaction diagrams only.
- State chart diagram:
  - when class has significant states.
  - When states are only one or two, state chart model becomes trivial
- Deployment diagram:
  - In case several hardware components used to develop the system.

Use Case Modelling

Use Case Model



- Consists of a set of "**use cases**"
- It is the central model:
  - –Other models must conform to this model
  - –Not really an object-oriented model, it is a functional model of a system

A Use Case

- **A case of use**: A way in which a system can be used by the users to achieve specific goals
- Corresponds to a high-level requirement.
- Defines external behavior without revealing internal structure of system
- **Set of related scenarios tied together by a common goal**.

–Use cases for a Library information system

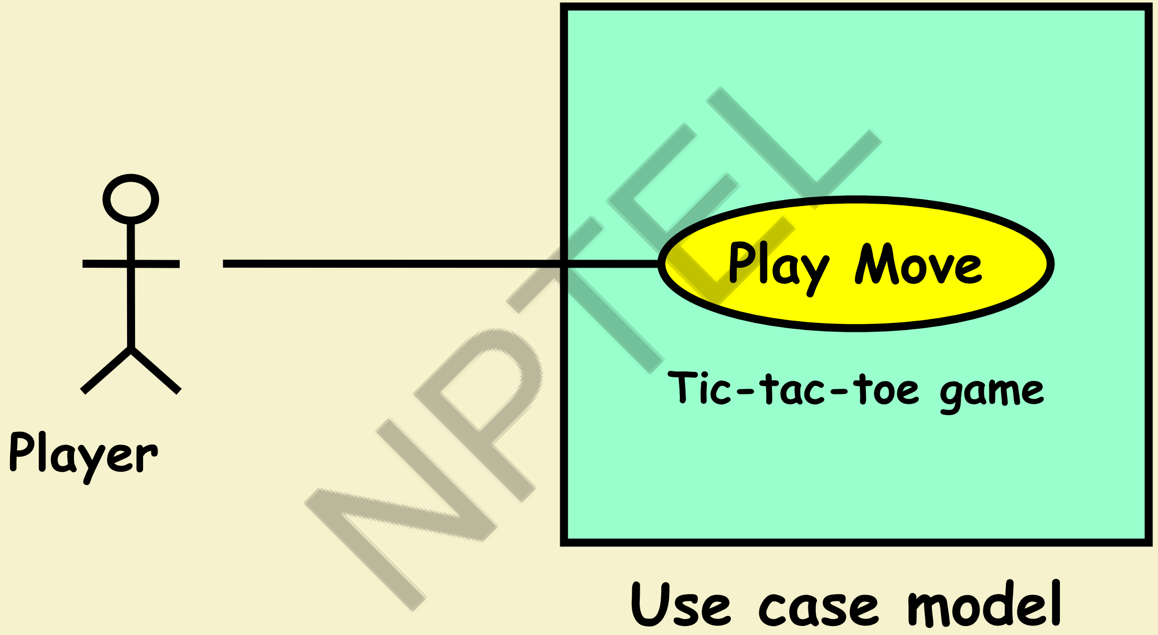
- issue-book
- query-book
- return-book
- create-member
- add-book, etc.

Example Use Cases

Are All Use Cases Independent?

- Use cases appear independent of each other
- However, Implicit dependencies may exist
- **Example:** In Library Automation System, renew-book and reserve-book are independent use cases.
  - But in actual implementation of renew-book--- **A check is made to see if any book has been reserved using reserve-book.**

An Example Use Case Diagram

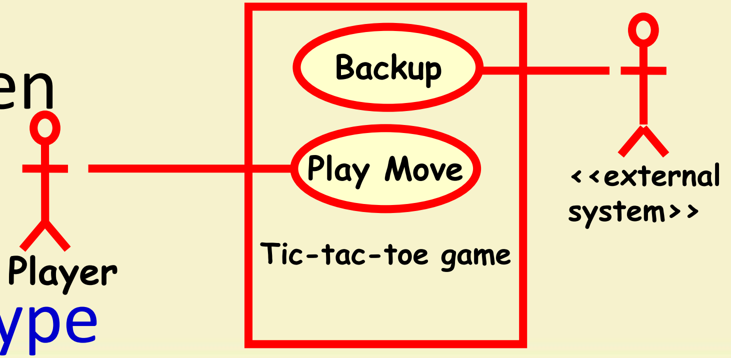


Why Develop A Use Case Diagram?

- Serves as requirements specification
- How are actor identification useful in software development?
  - –Identifies different categories of users:
    - **Helps in implementing appropriate interfaces for each category of users.**
    - **Helps in preparing appropriate documents (e.g.** users' manual**).**

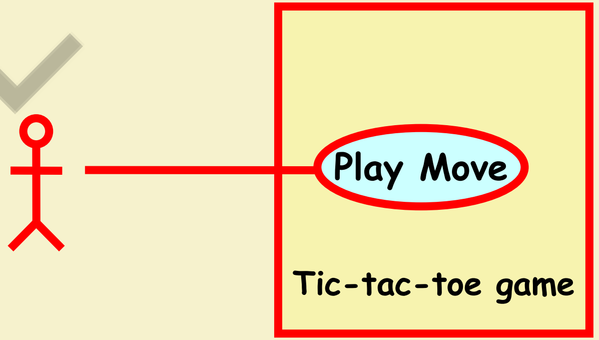
Representation of Use Cases

- Represented in a use case diagram
- A use case is represented by an ellipse
- System boundary is represented by a rectangle
- Users are represented by stick person icons (actor)
- Communication relationship between actor and use case by a line
- External system by adding a stereotype



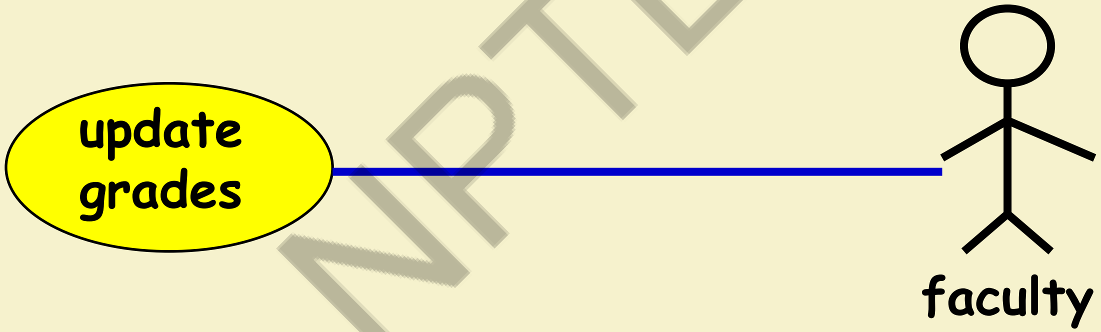
What is a Connection?

- A connection is an association between an actor and a use case.
- Depicts a usage relationship
- Connection does not indicate data flow

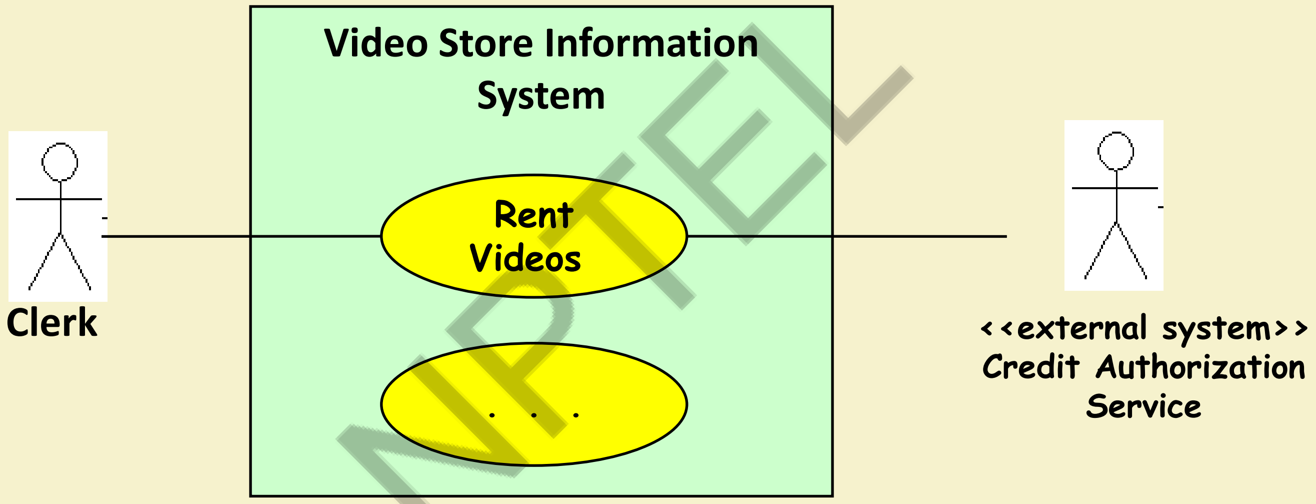


Relationships between Use Cases and Actors

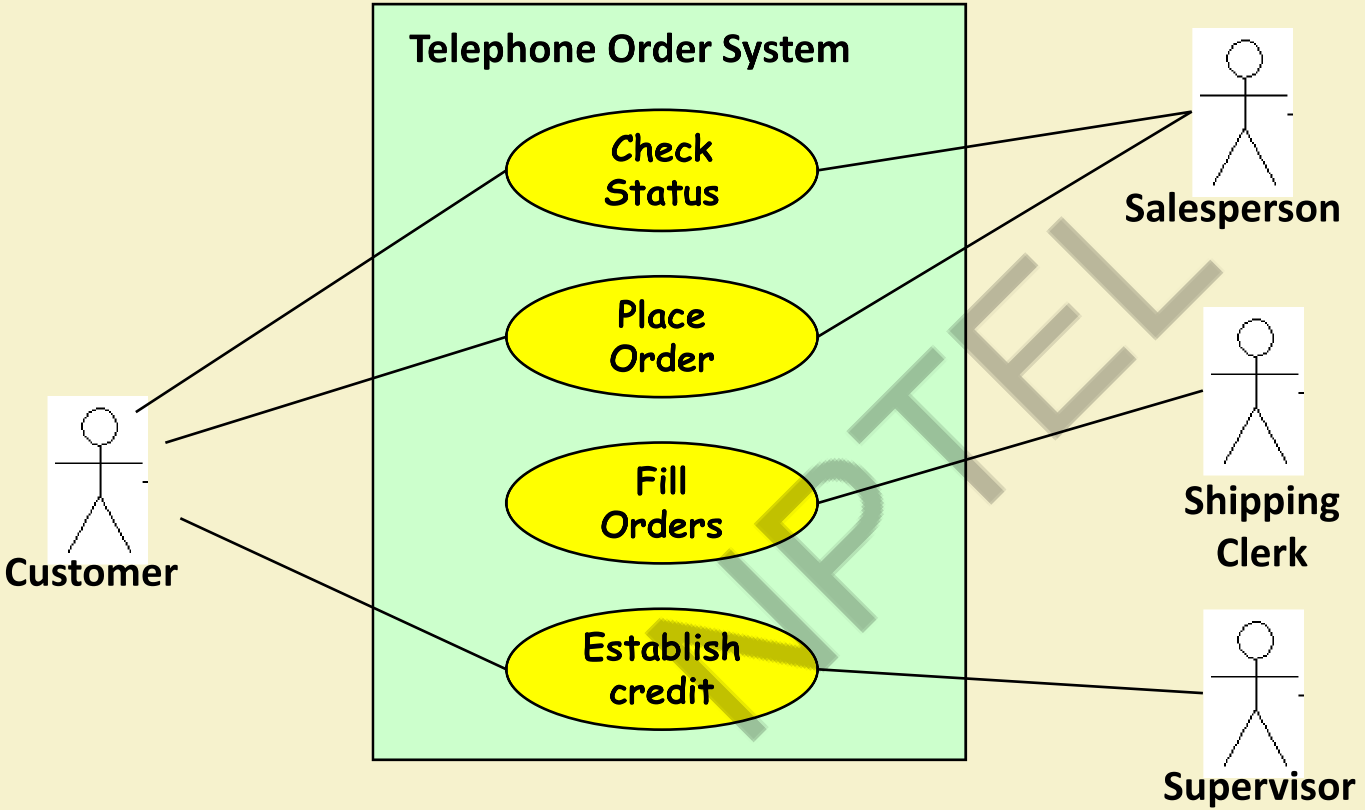
- Association relation indicates that the actor and the corresponding use case communicate with one another.



Another Example Use Case Diagram


Video Store Information System
Rent Videos
. . .
Clerk
<<external system>> Credit Authorization Service

Yet Another Use Case Example

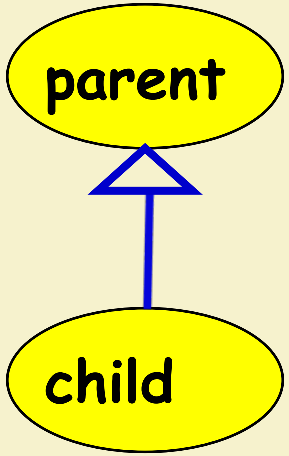


Factoring Use Cases

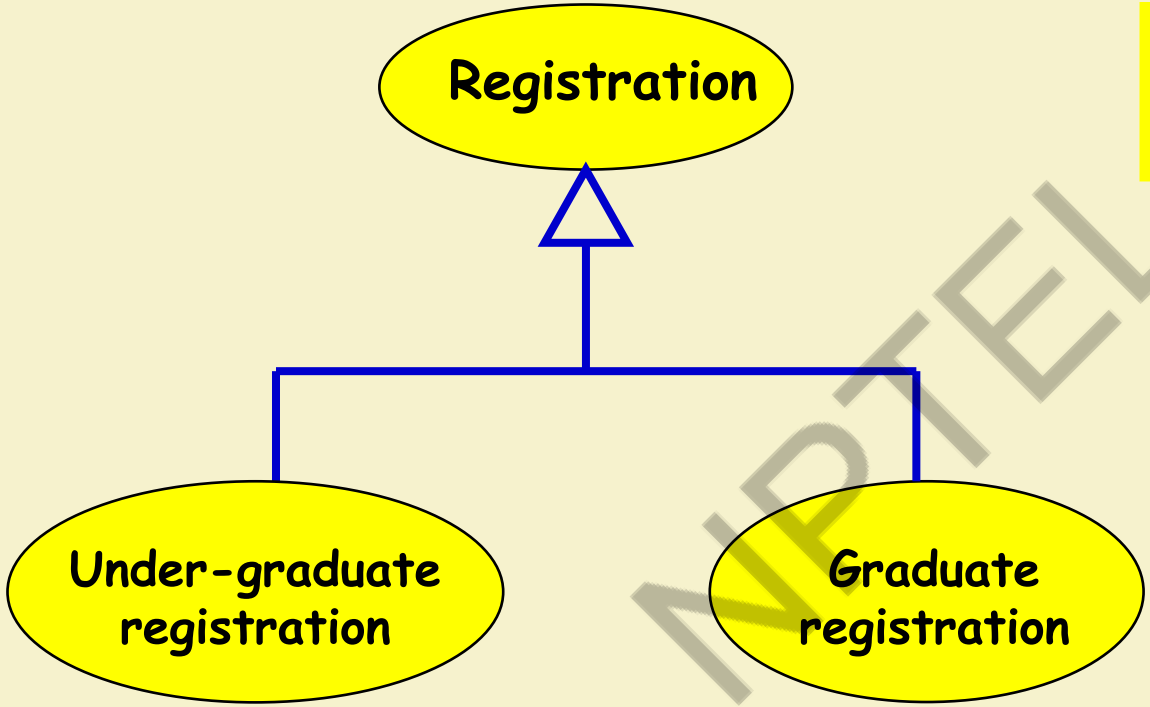
- Two main reasons for factoring:
  - **Complex use cases need to be factored into simpler use cases**
  - **Helps represent common behavior across different use cases**
- Three ways of factoring:
  - **Generalization**
  - **Include**
  - **Extend**

Generalization

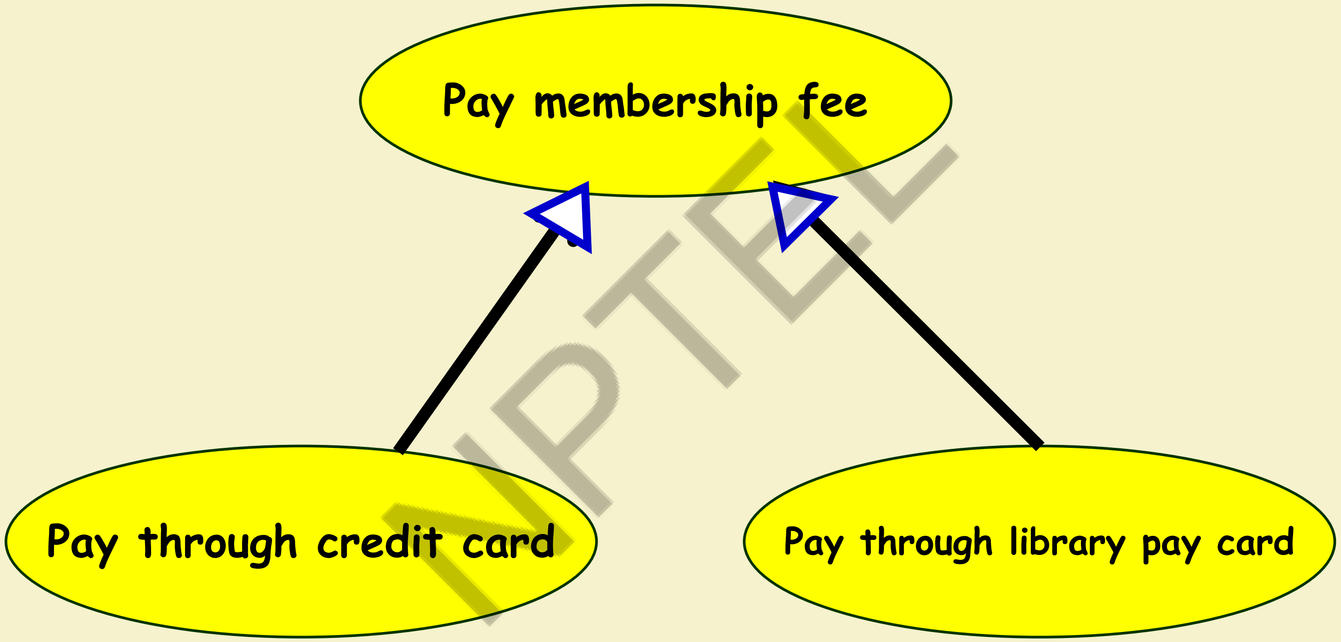
- The child use case inherits the behavior of the parent use case.
  - –The child may add to or override some of the behavior of its parent.



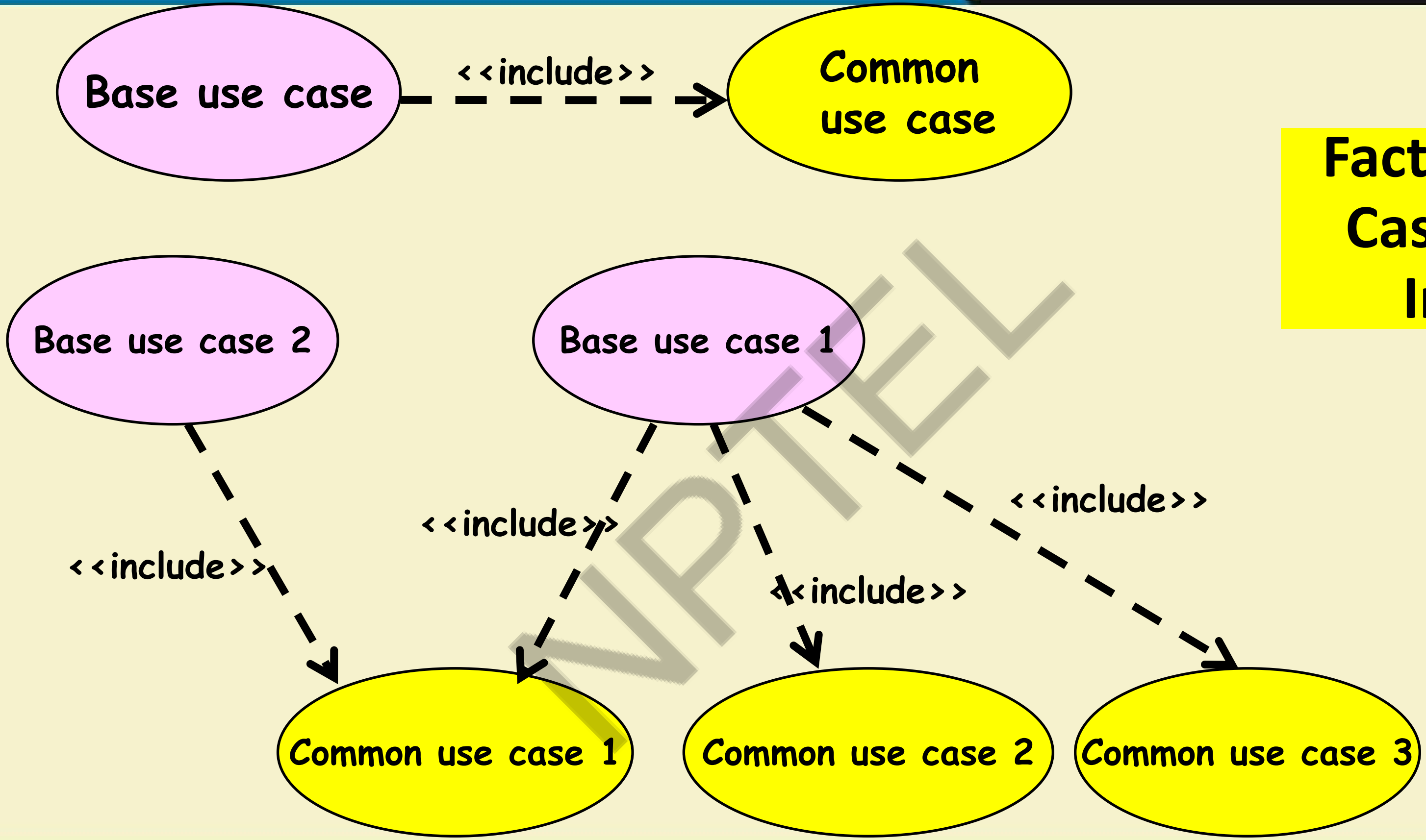
Generalization Example 1

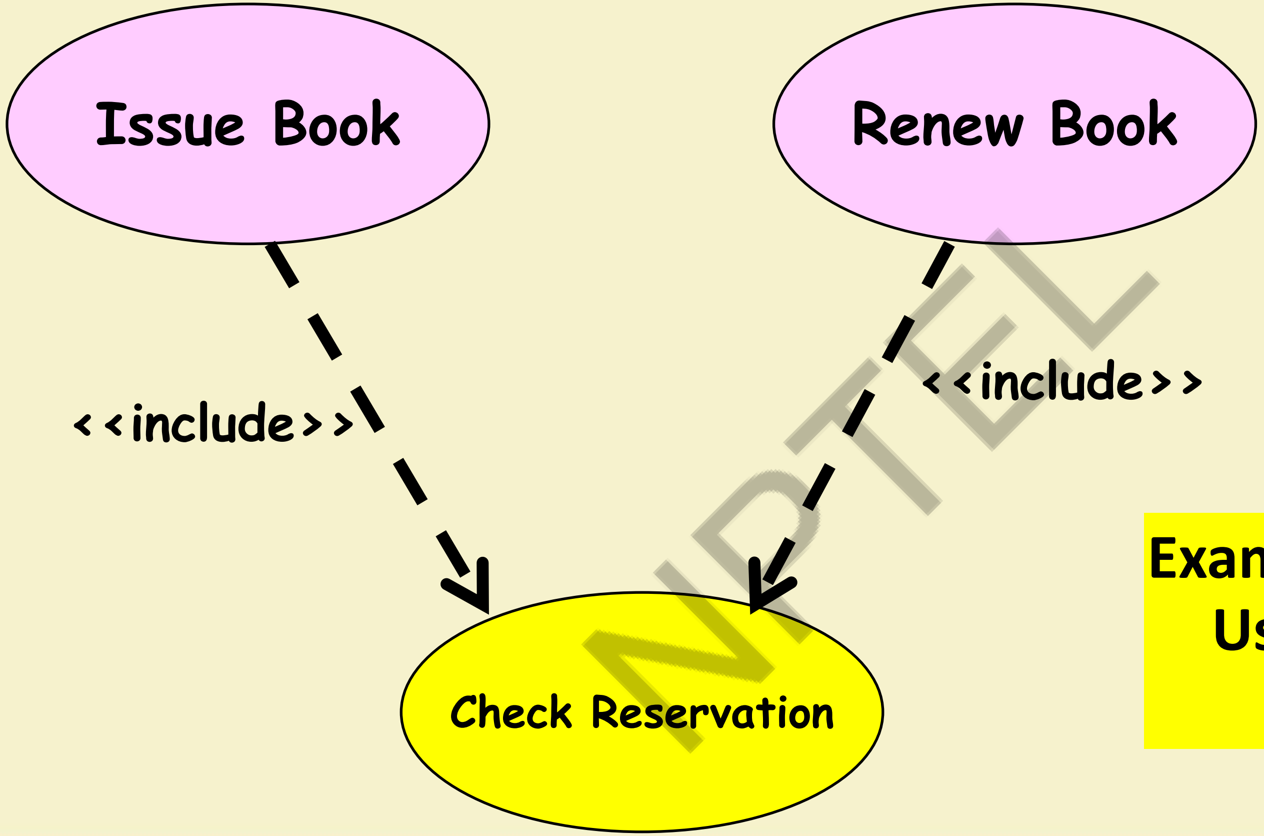


Factoring Use Cases Using Generalization



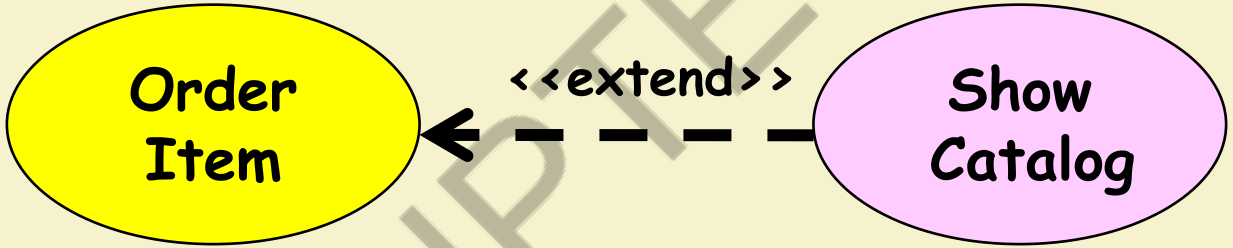
Factoring Use Cases Using Include





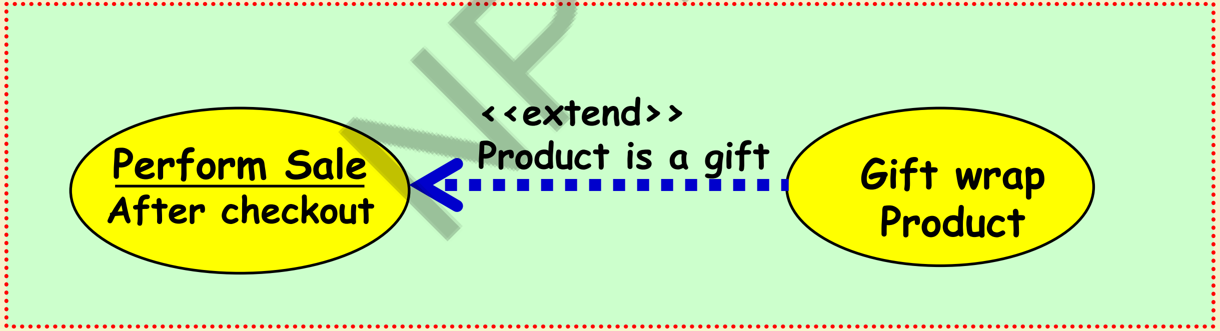
Example of Factoring Use Cases Using Include

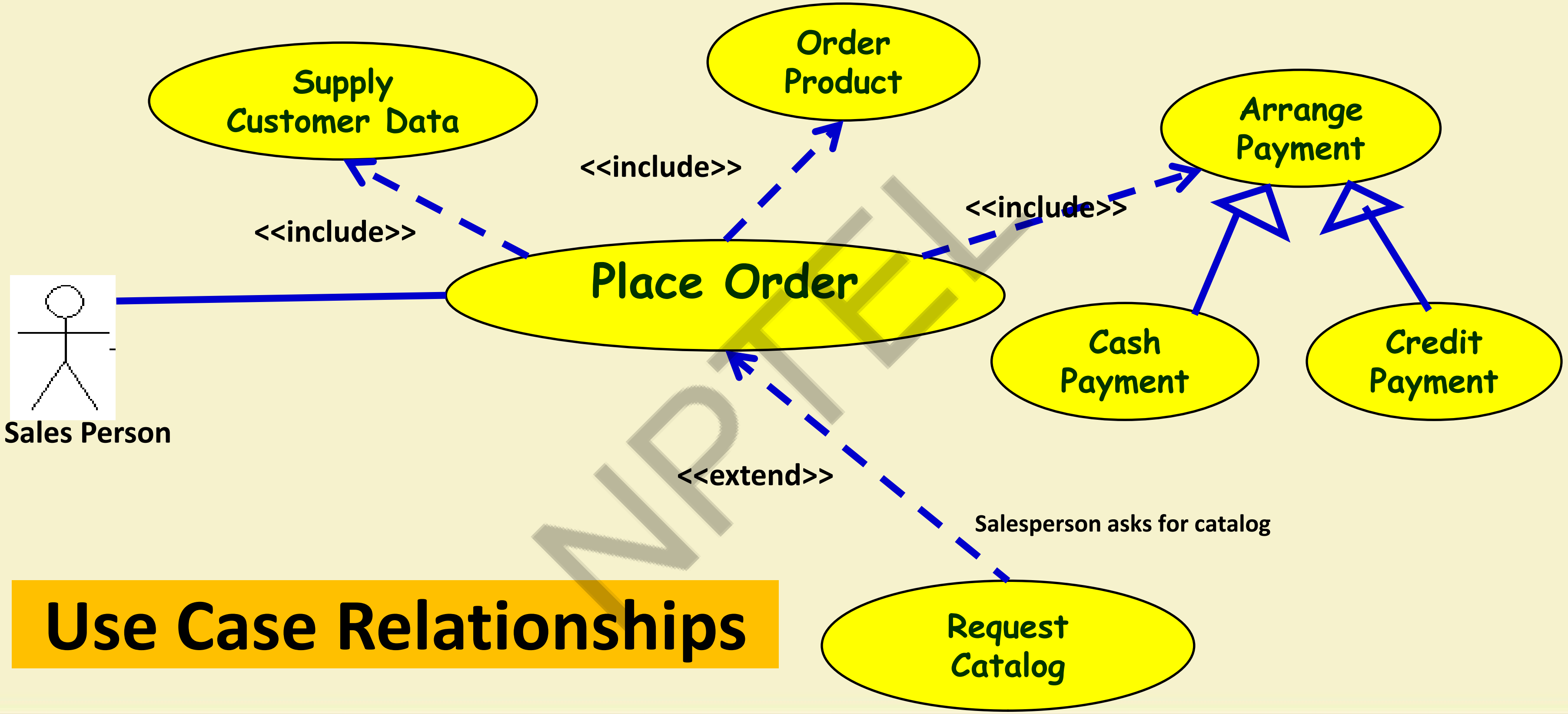
Example Factoring A Use Case Using Extend



Extension Point

- The base use case may include/extend other use cases:
  - **At certain points, called extension points.**
- Note the direction of the arrow



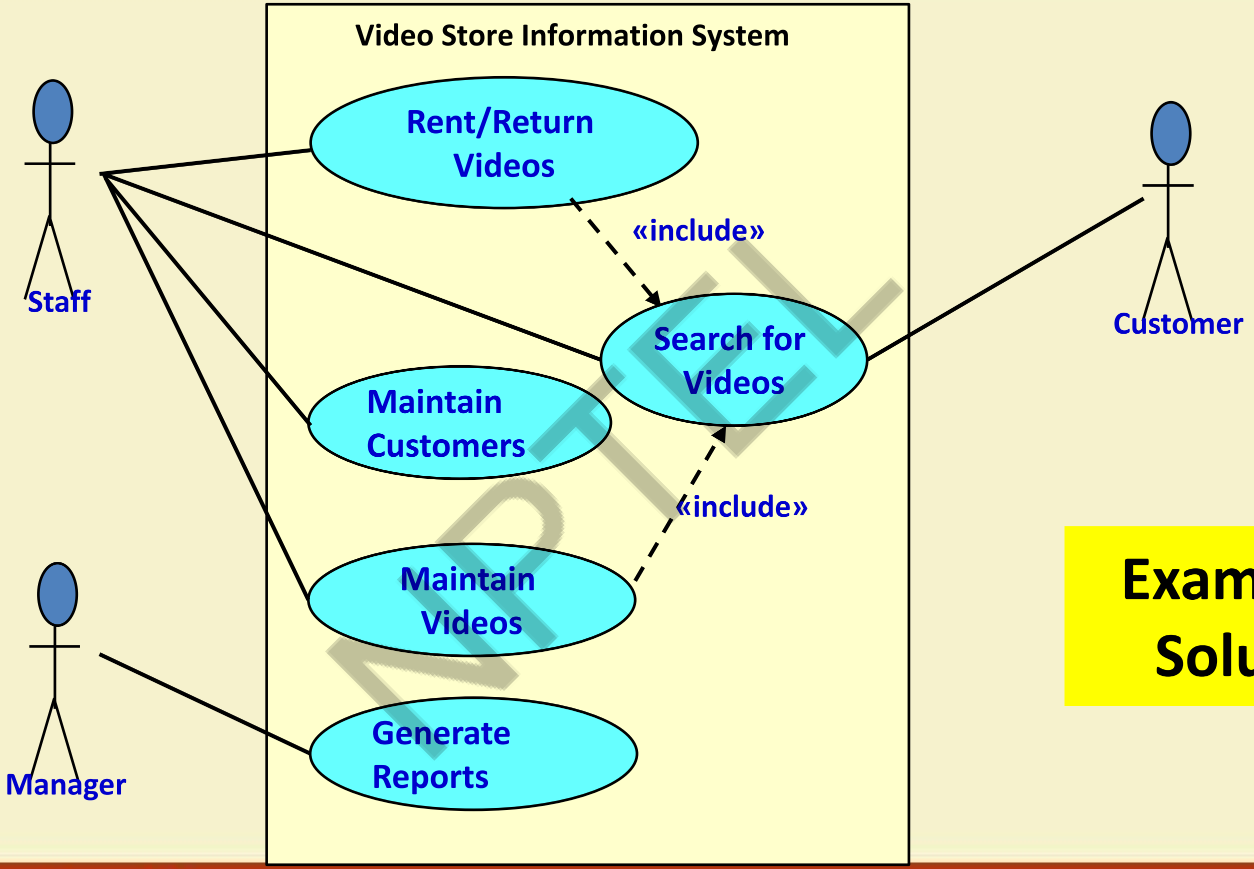


Use Case Relationships

- Video Store Information System supports the following business functions:

Example 1: Video Store Information System

  - Recording information about videos the store owns
    - This database is searchable by staff and all customers
  - Information about a customer’s borrowed videos
    - Access by staff and customer. It involves video database searching.
  - Staff can record video rentals and returns by customers. It involves video database searching.
  - Staff can maintain customer and video information.
  - Managers of the store can generate various reports.



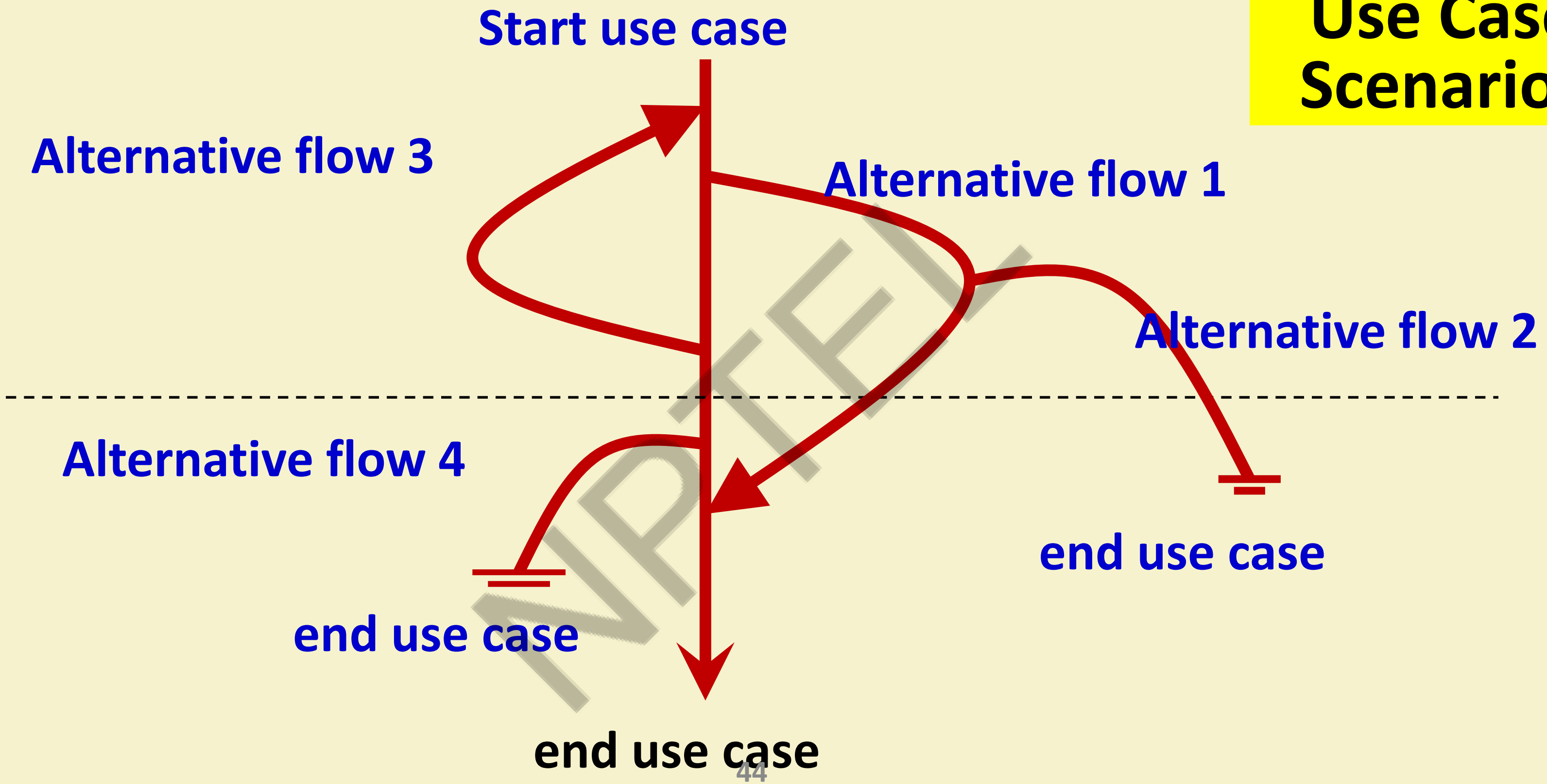
Example 1: Solution

Use Case Description

Name
Actors
Trigger
Preconditions
Post conditions
Mainline Scenario
Alternatives flows

Alistair Cockburn "Writing Effective Use Cases"

Use Case Scenarios



ATM Money Withdraw Example

- **Actors:** Customer
- **Pre Condition:**
  - ATM must be in a state ready to accept transactions
  - ATM must have at least some cash it can dispense
  - ATM must have enough paper to print a receipt
- **Post Condition:**
  - The current amount of cash in the user account is the amount before withdraw minus withdraw amount
  - A receipt was printed on the withdraw amount

ATM Money Withdraw Mainline Scenario

Actor Actions	System Actions
1. Begins when a Customer arrives at ATM	
2. Customer inserts a Credit card into ATM	3. System verifies the customer ID and status
5. Customer chooses "Withdraw" operation	4. System asks for an operation type
7. Customer enters the cash amount	6. System asks for the withdraw amount
	8. System checks if withdraw amount is legal
	9. System dispenses the cash
	10. System deduces the withdraw amount from account
	11. System prints a receipt
13. Customer takes the cash and the receipt	12. System ejects the cash card

ATM Money Withdraw (cont.)

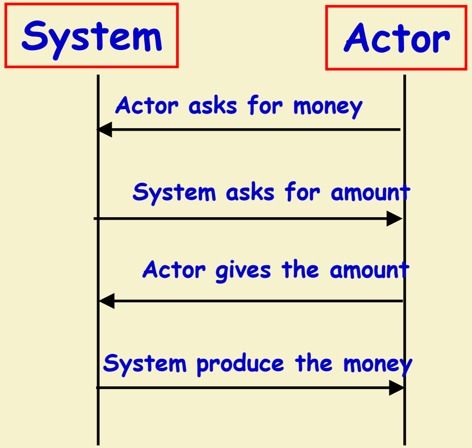
- **Alternative flow of events:**
  - **Step 3:** Customer authorization failed. Display an error message, cancel the transaction and eject the card.
  - **Step 8:** Customer has insufficient funds in its account. Display an error message, and go to step 6.
  - **Step 8:** Customer exceeds its legal amount. Display an error message, and go to step 6.
- **Exceptional flow of events:**
  - Power failure in the process of the transaction before step 9, cancel the transaction and eject the card.

Use Case Description: Change Flight

- **Actors:** traveler
- **Preconditions:**
  - Traveler has logged on to the system and selected 'change flight itinerary' option
- **Basic course**
  1. System retrieves traveler's account and flight itinerary from client account database
  2. System asks traveler to select itinerary segment she wants to change; traveler selects itinerary segment.
  3. System asks traveler for new departure and destination information; traveler provides information.
  4. If flights are available then
  5. …
  6. System displays transaction summary.
- **Alternative courses**
  4. If no flights are available then …

Guidelines for Effective Use Case Writing

- Use simple sentence
- Do not have both system and actor doing something in a single step
  - Bad: “Get the amount from the user and give him the receipt.”
- Any step should lead to some tangible progress:
  - Bad: “User clicks a key”



Identification of Use Cases

1. Actor-based:

- Identify the actors related to a system or organization.
- For each actor, identify the processes they initiate or participate in.

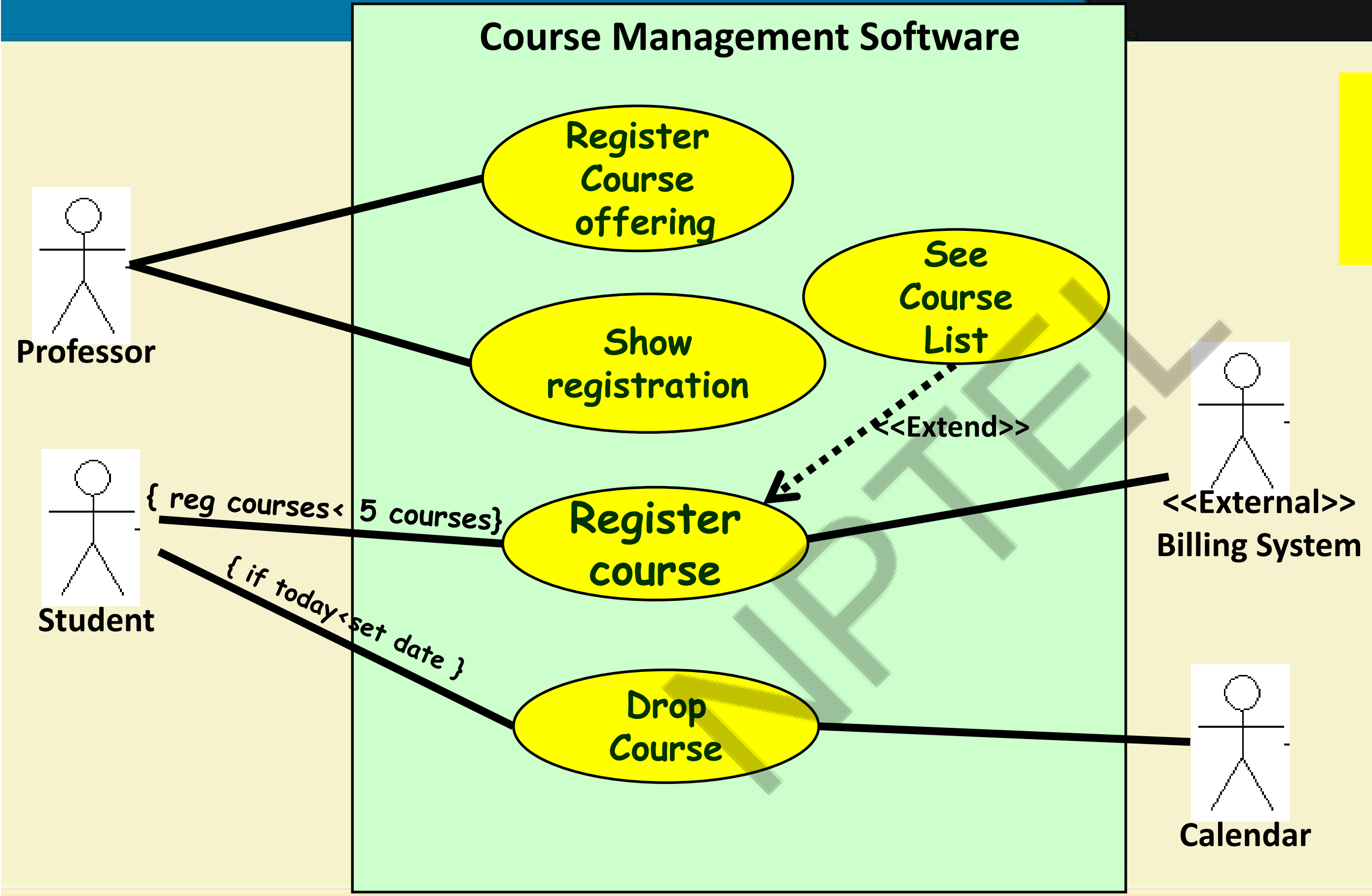
2. Event-based

- Identify the external events that the system must respond to.
- Relate the events to actors and use cases.

Example 2: Use Case Model for Course Management Software

- At the beginning of each semester,
  - Each professor shall register the courses that he is going to teach.
- A student can select up to four-course offerings.
  - **During registration a student can request a course catalogue showing course offerings for the semester.**
  - **Information about each course such as professor, department and prerequisites would be displayed.**
  - **The registration system sends information to the billing system, so that the students can be billed for the semester.**
- For each semester, there is a period of time during which dropping of courses is permitted.
- Professors must be able to access the system to see which students signed up for each of their course offerings.

Example 2: Model Solution



- Use case name should begin with a verb.
- While use cases do not explicitly imply timing:
  - Order use cases from top to bottom to imply timing -- it improves readability.
- **The primary actors should appear in the left.**
- Actors are associated with one or more use cases.
- Do not use arrows on the actor-use case relationship.
- **To initiate scheduled events include an actor called "time", or "calendar"**
- **Do not show actors interacting with each other.**
- <<include>> should rarely nest more than 2 levels deep.

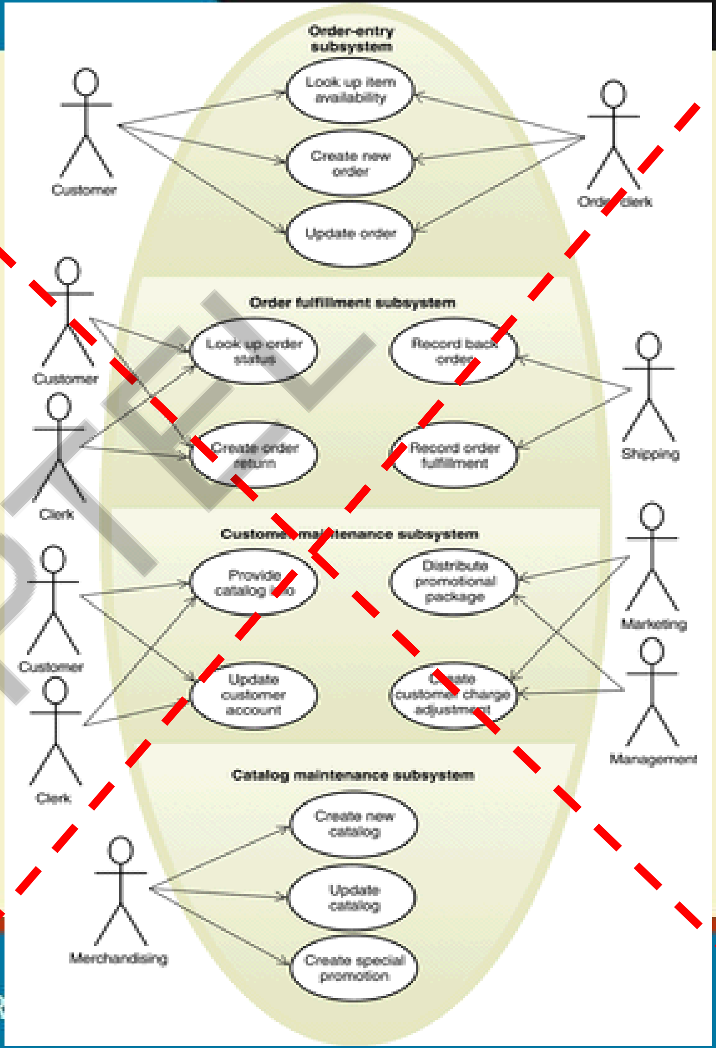
Style Notes (Ambler, 2005)

- Use cases should be named and organized from the perspective of the users.
- Use cases should start off simple and at as much high view as possible.
  - Can be refined and detailed further.

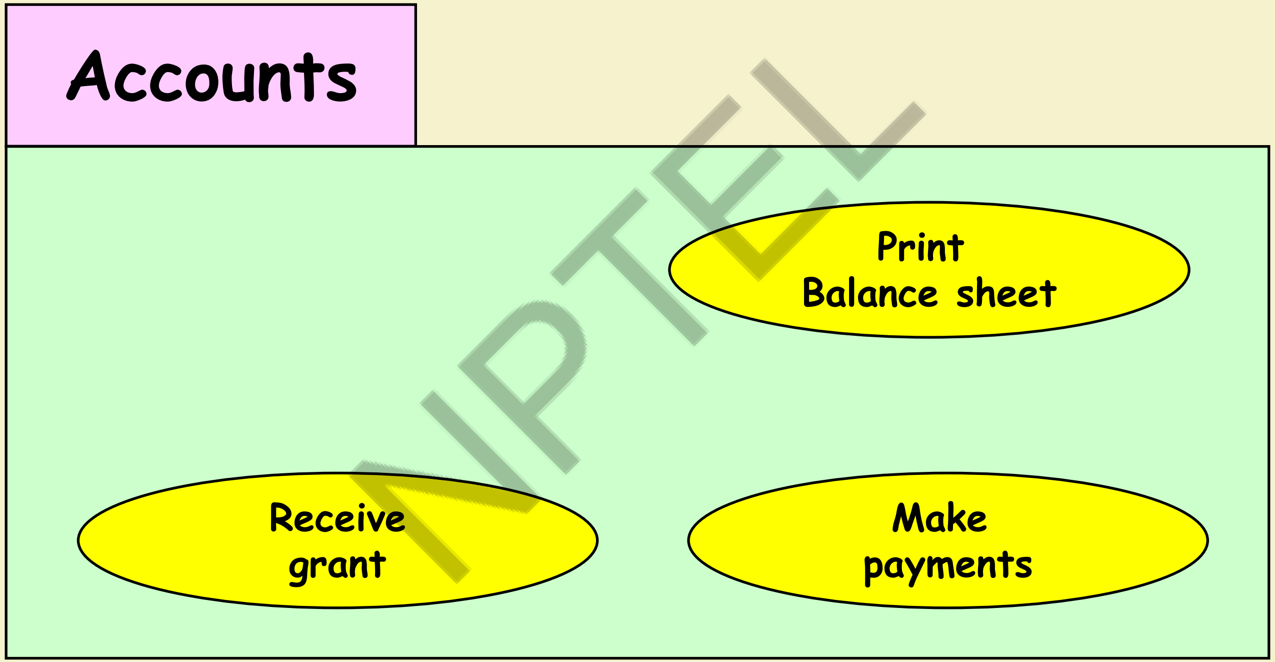
Effective Use Case Modelling

- Use case diagrams represent functionality:
  - **Should focus on the "what" and not the "how".**

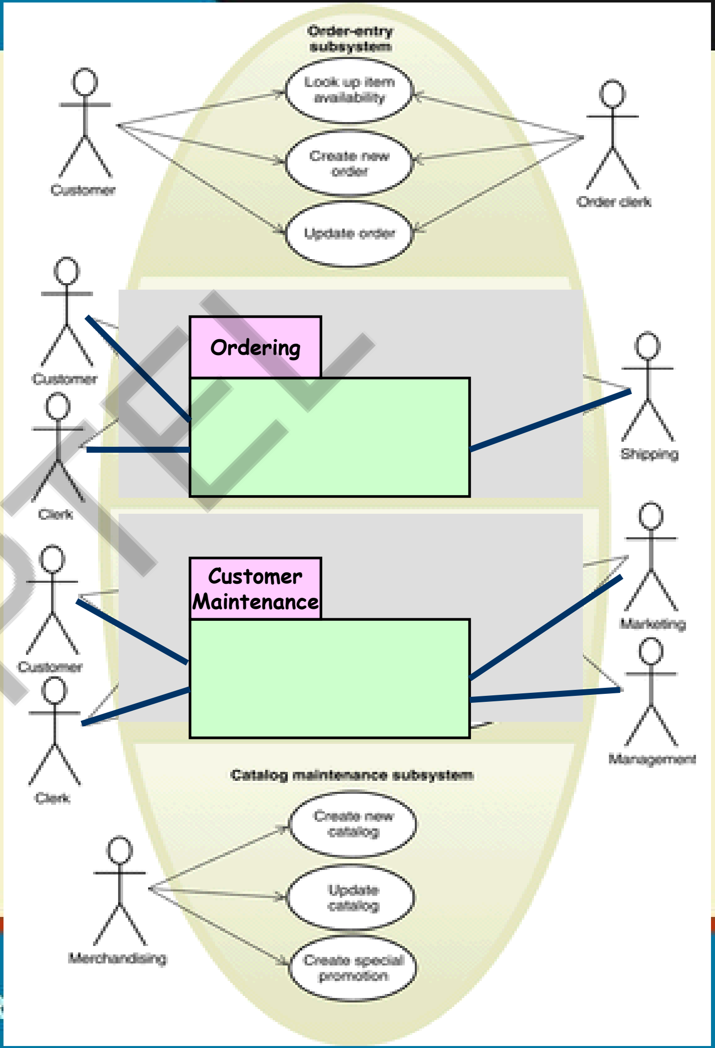
Too many use cases at any level should be avoided!



Use Case Packaging



More accetable!

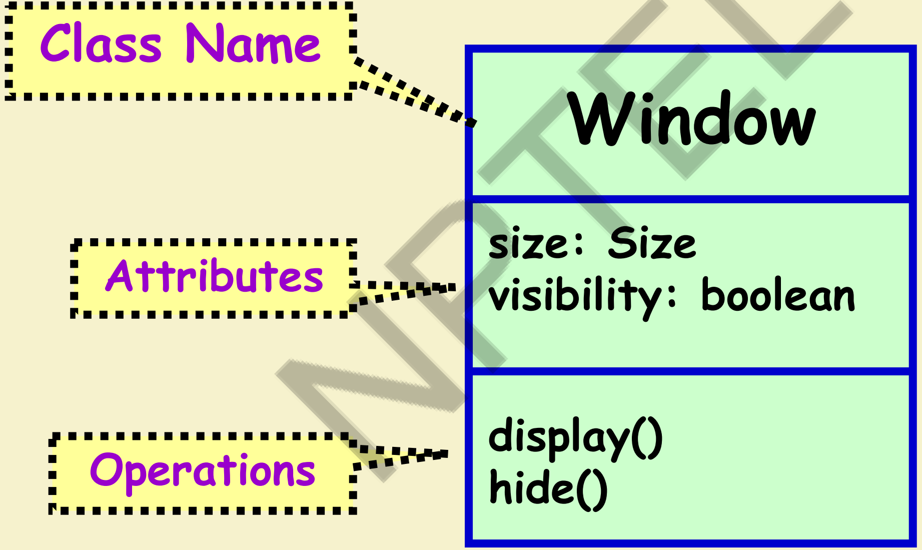


Class Diagram

- Classes:
  - Entities with common features, i.e. attributes and operations.
  - Represented as solid outline rectangle with compartments.
  - Compartments for **name, attributes, and operations.**
  - Attribute and operation compartments are optional depending on the purpose of a diagram.

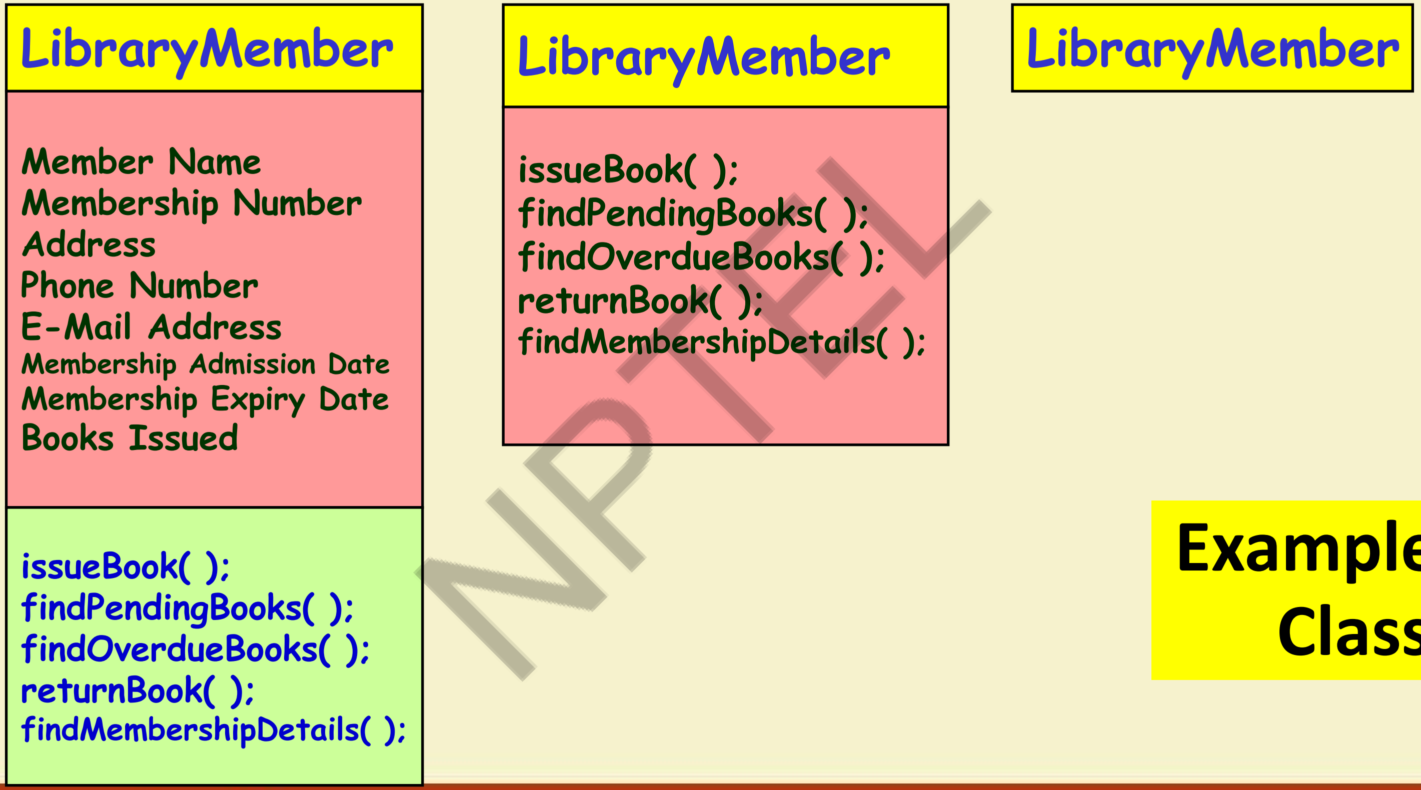
UML Class Representation

- A class represents a set of objects having similar attributes, operations, relationships and behavior.



A class can implicitly have a few association attributes

Different representations of the LibraryMember class



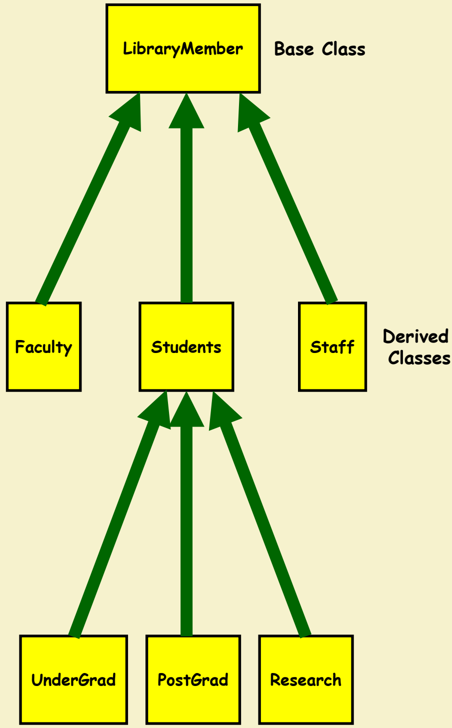
Example UML Classes

What are the Different Types of Relationships Among Classes?

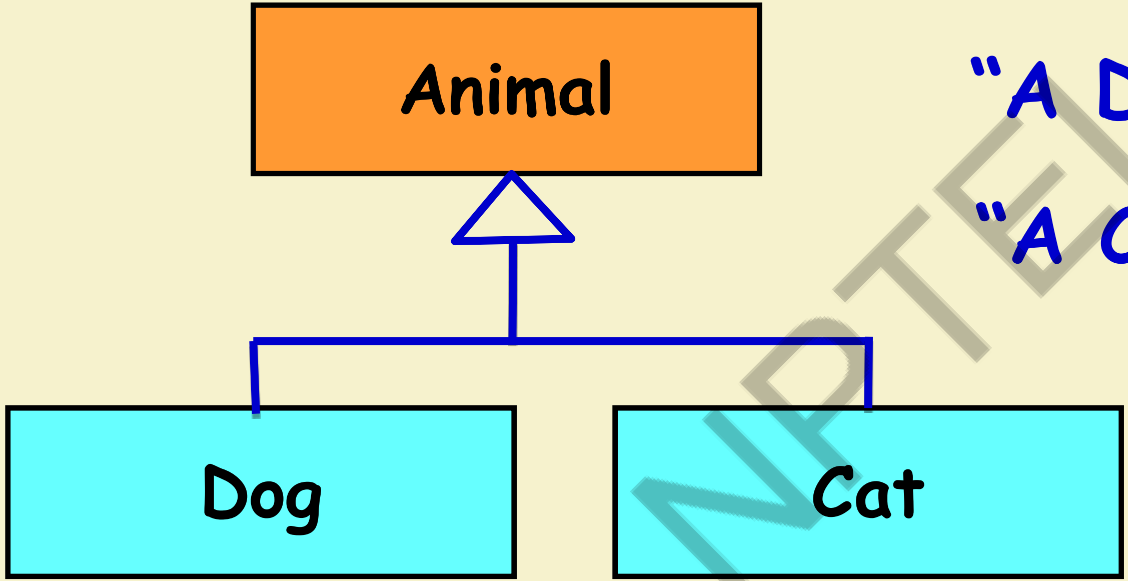
- Four types of relationships:
  - **Inheritance**
  - **Association**
  - **Aggregation/Composition**
  - **Dependency**

Inheritance

- Allows to define a new class (derived class) by extending an existing class (base class).
  - –Represents generalization-specialization
  - –Allows redefinition of the existing methods (method overriding).



Inheritance Example

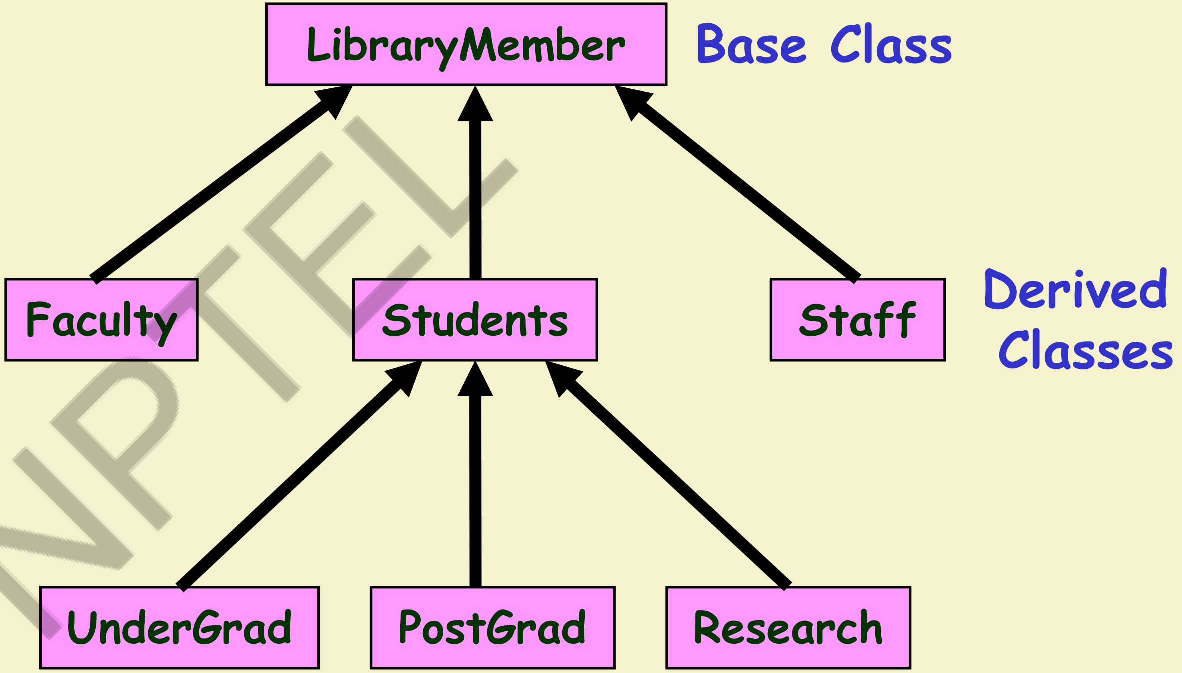


"A Dog <u>*ISA*</u> Animal"

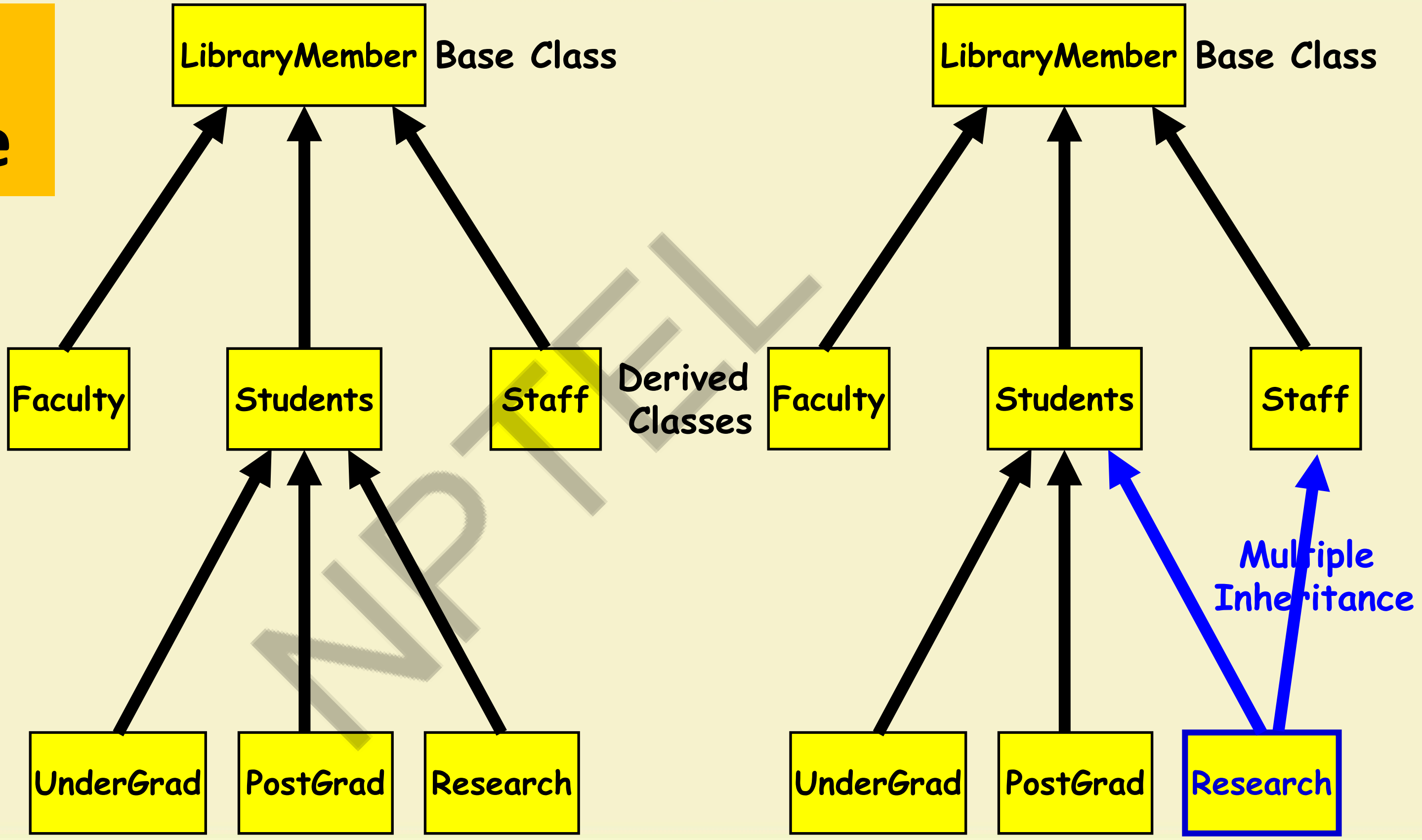
"A Cat <u>*ISA*</u> Animal"

Inheritance

- Lets a subclass inherit attributes and methods from a base class.



Multiple Inheritance

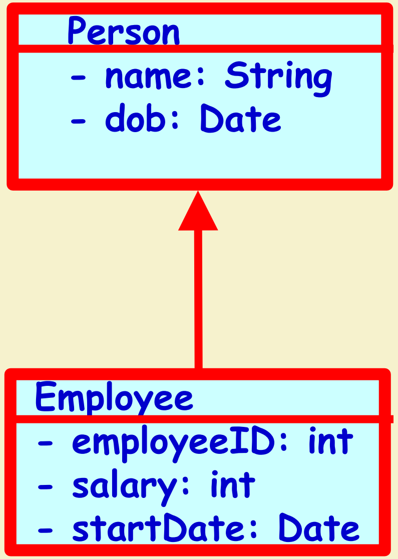


Inheritance Implementation in Java

- Inheritance is declared using the "extends" keyword
  - Even when no inheritance defined, the class implicitly extends a class called Object.

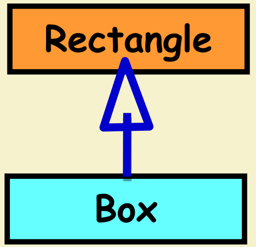
```
class Person{
   private String name;
   private Date dob;
   ...
}
```

```
class Employee extends Person{
   private int employeeID;
   private int salary;
   private Date startDate;
   ...
}
```

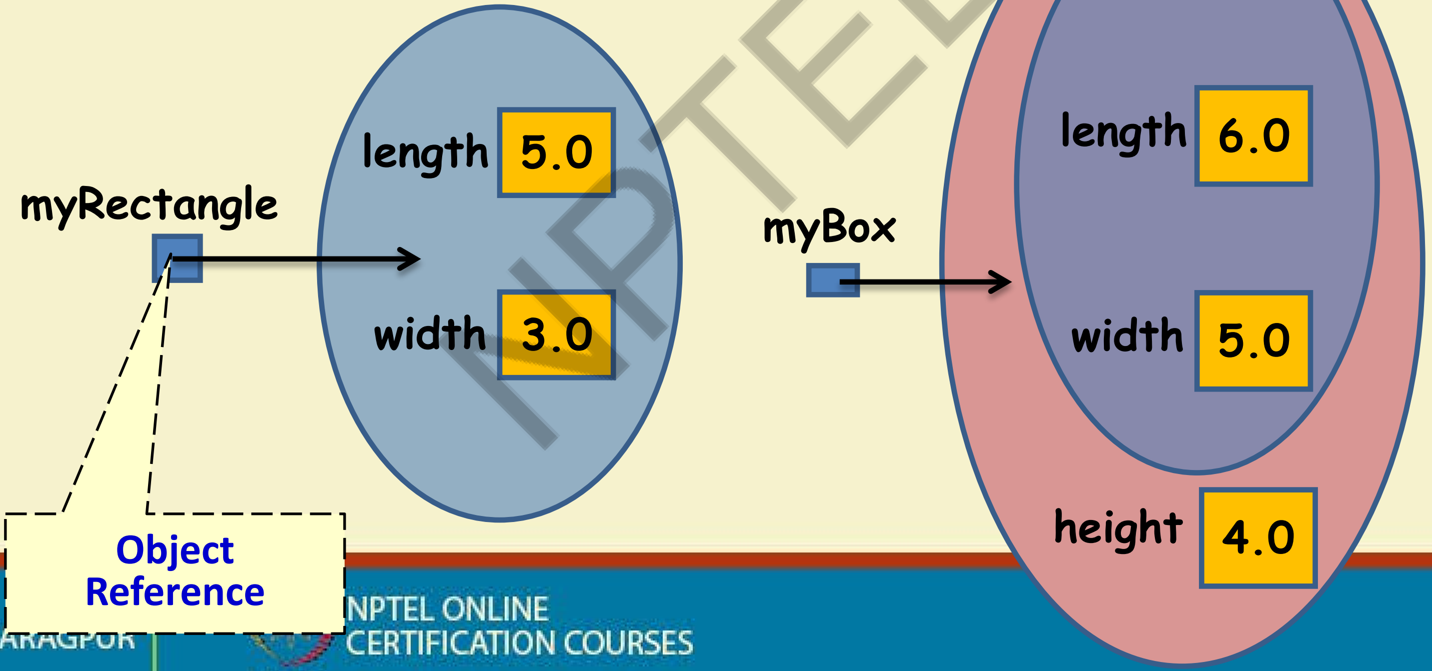


```
Employee anEmployee = new Employee();
```

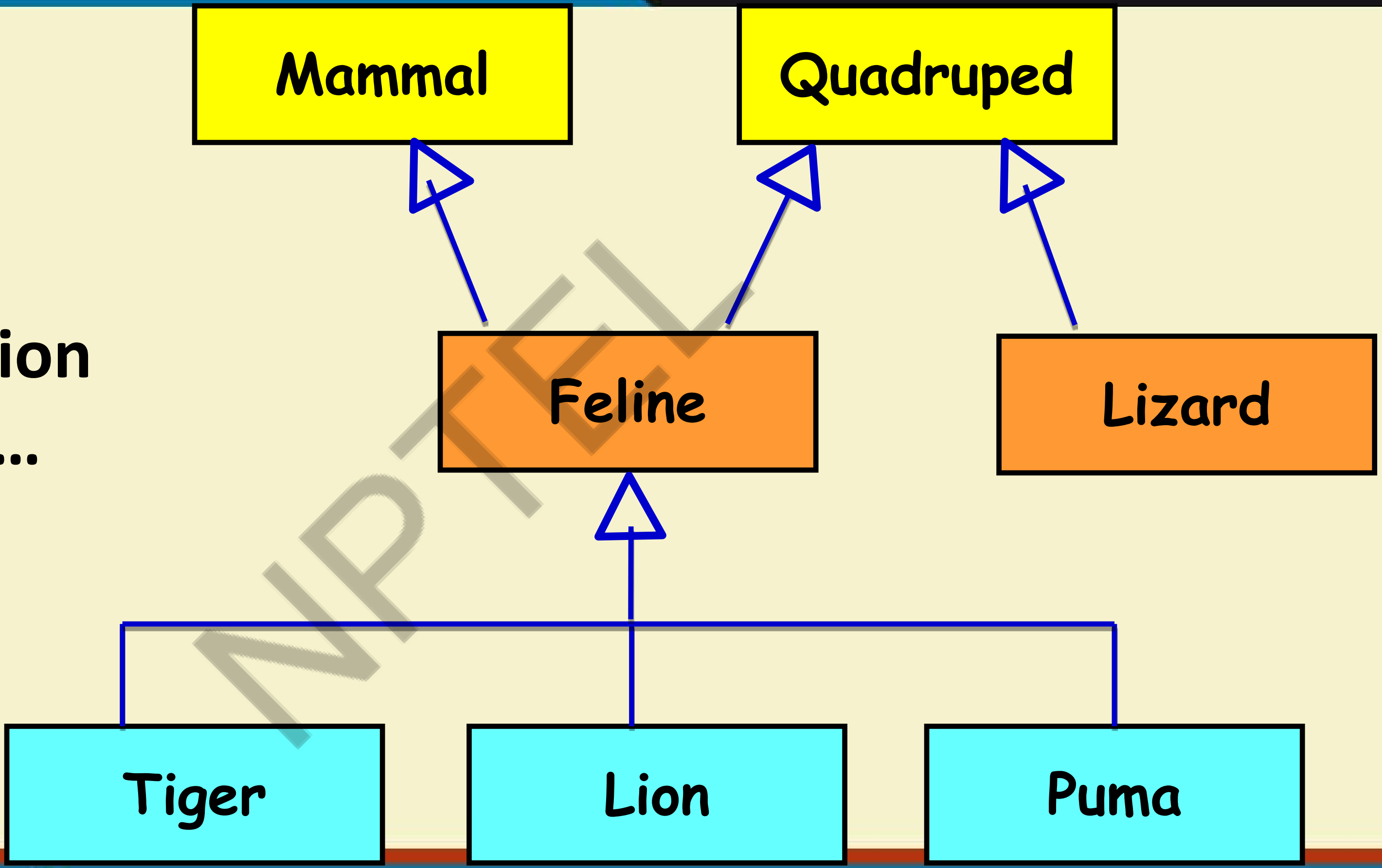
Objects myRectangle and myBox



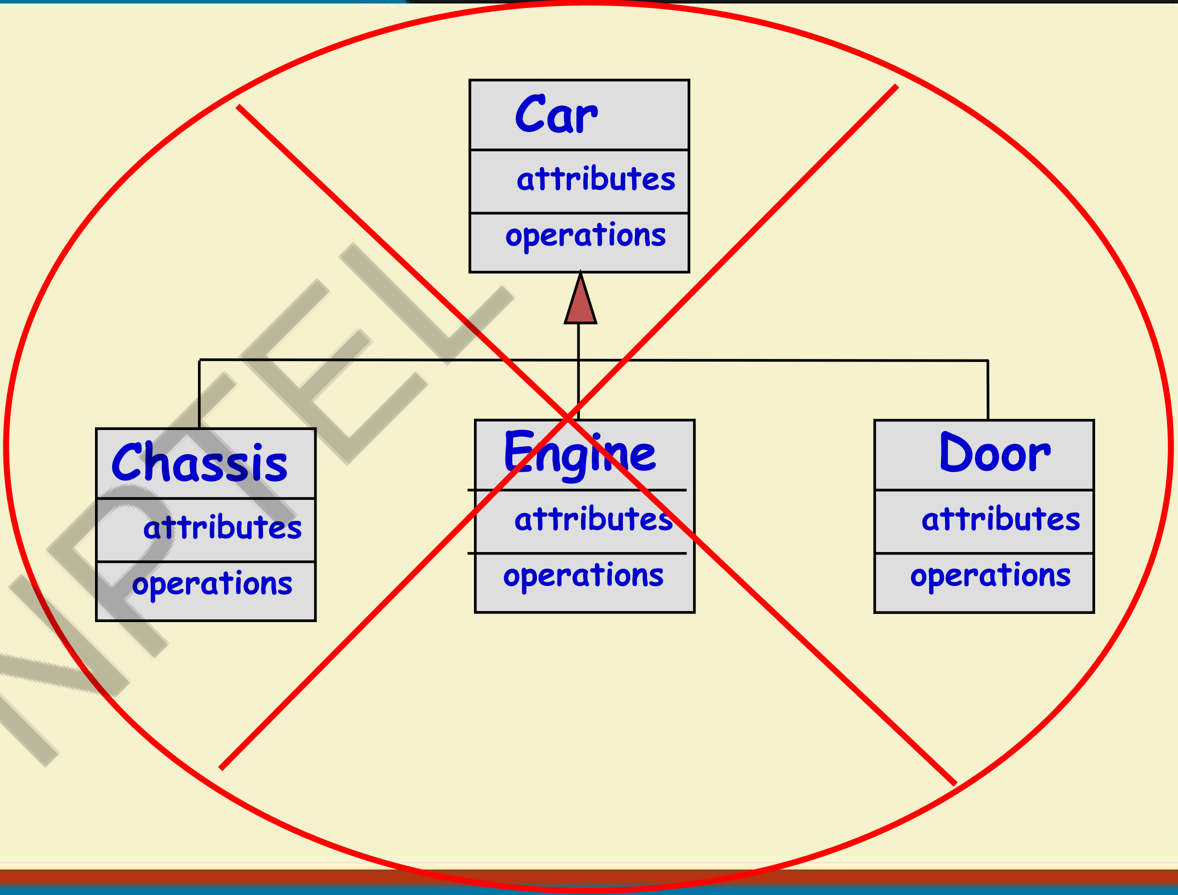
```
Rectangle myRectangle = new Rectangle(5, 3);
Box myBox = new Box(6, 5, 4);
```



More Generalization Examples...



Any problems?

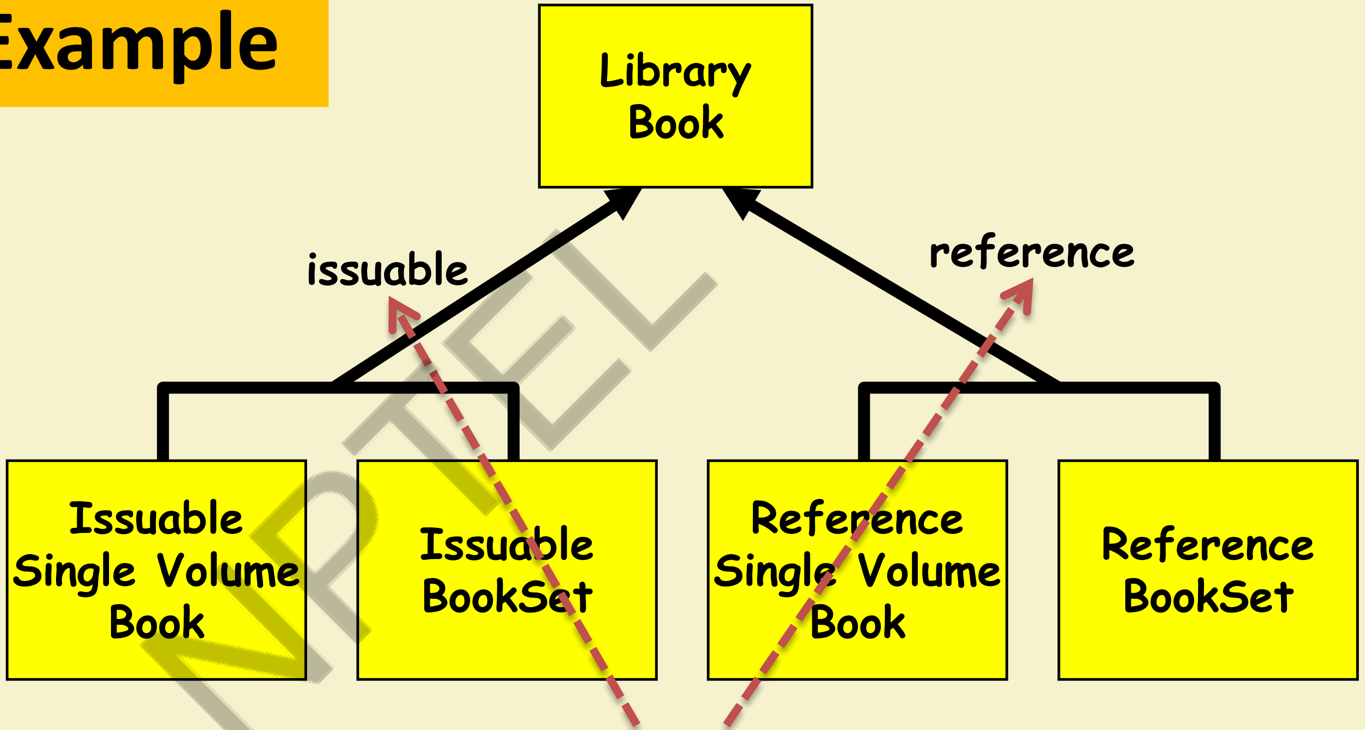


Wrong Generalization

---

violates "is a" or "is a kind of" heuristic

Inheritance Example



Discriminator: allows one to group subclasses into clusters that correspond to a semantic category.

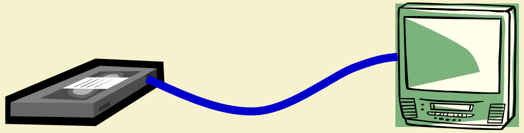
Inheritance Pitfalls

- Inheritance certainly promotes reuse.
- **Indiscriminate use can result in poor quality programs.**
- Base class attributes and methods visible in derived class…
  - Leads to tight coupling

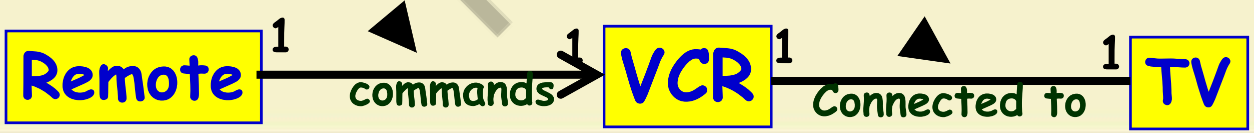
Association Relationship

- How implemented in program?
- Enables objects to communicate with each other:
  - –One object must "know" the ID of the corresponding object in the association.
- Usually binary:
  - –But in general can be n-ary.

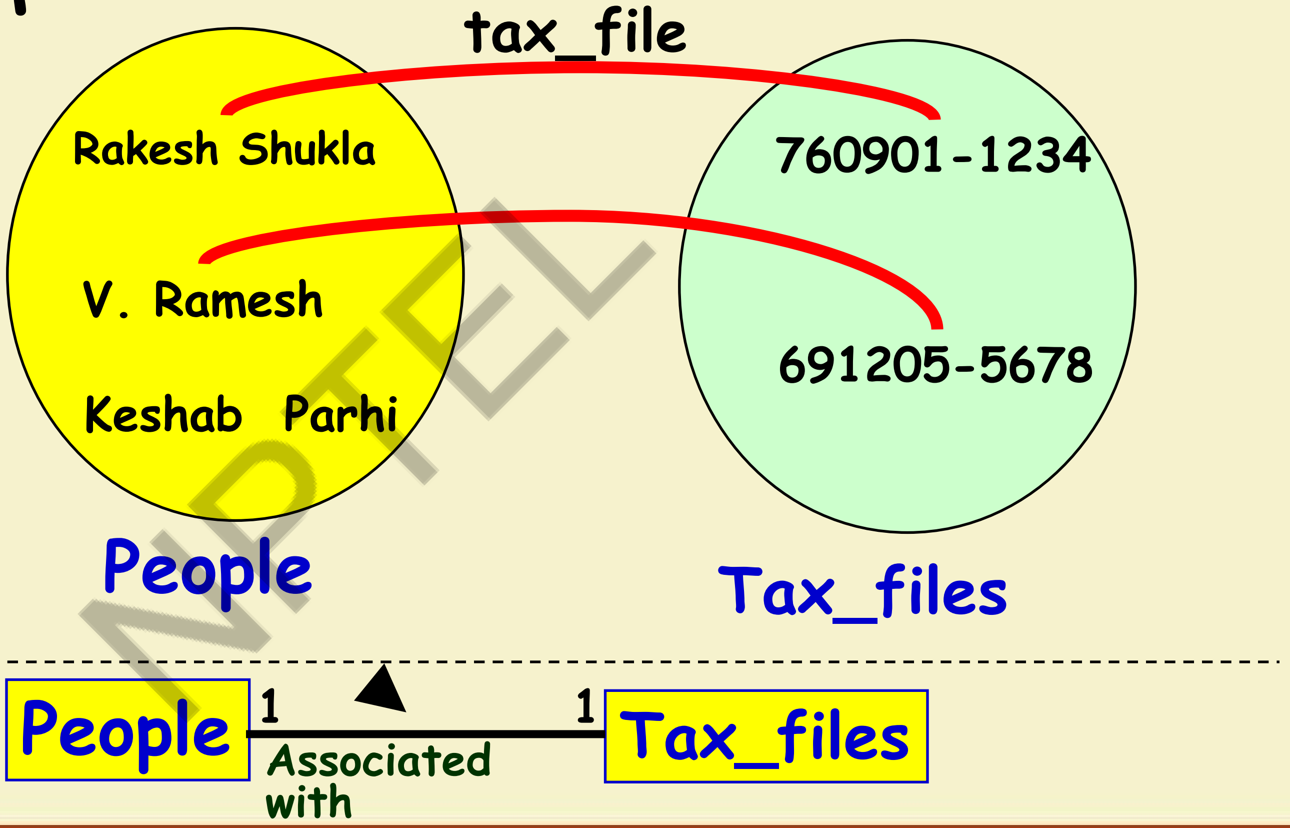
Association – example



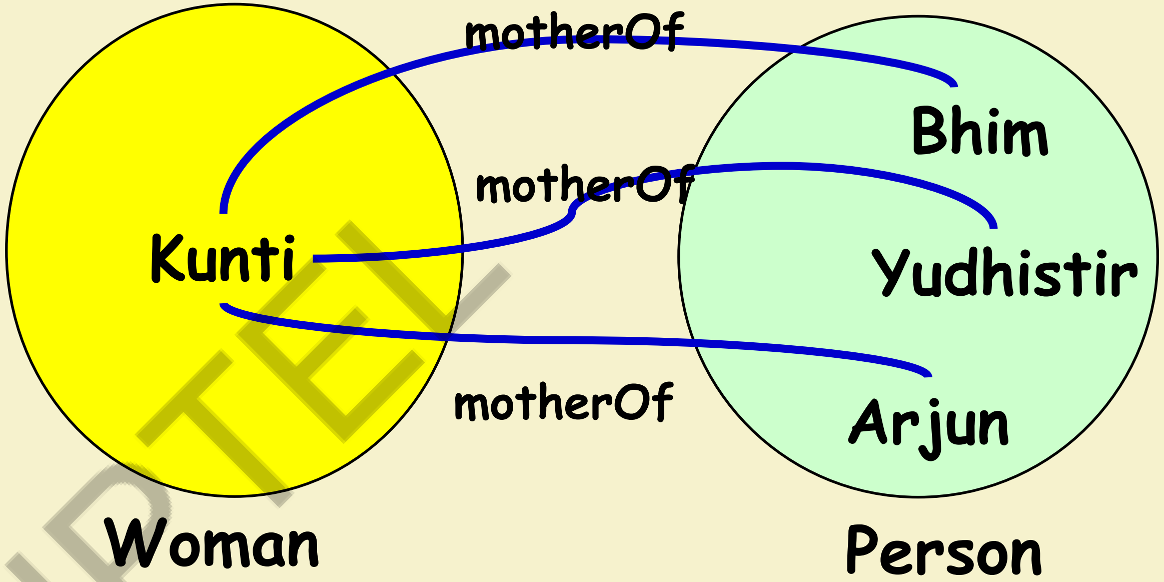
- In a home theatre system,
  - A TV object has an association with a VCR object
    - It may receive a signal from the VCR
  - VCR may be associated with remote
    - It may receive a command to record

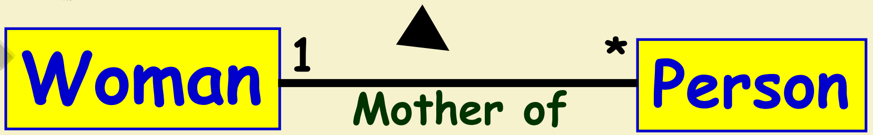


1-1 Association - example

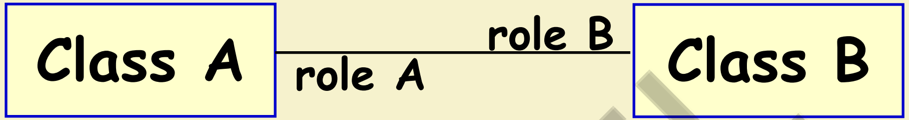


Multiple Association – example

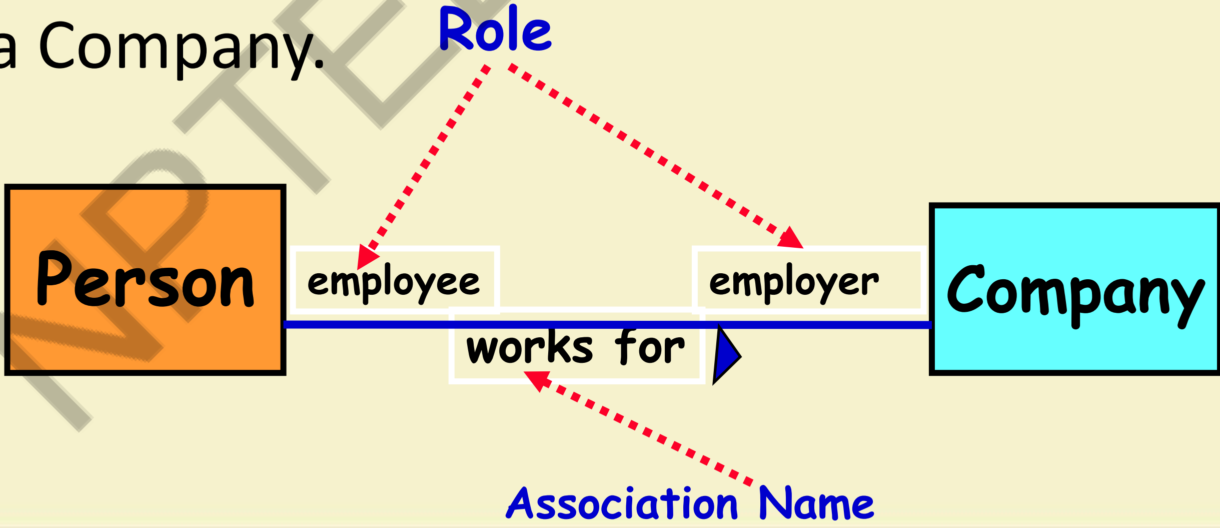




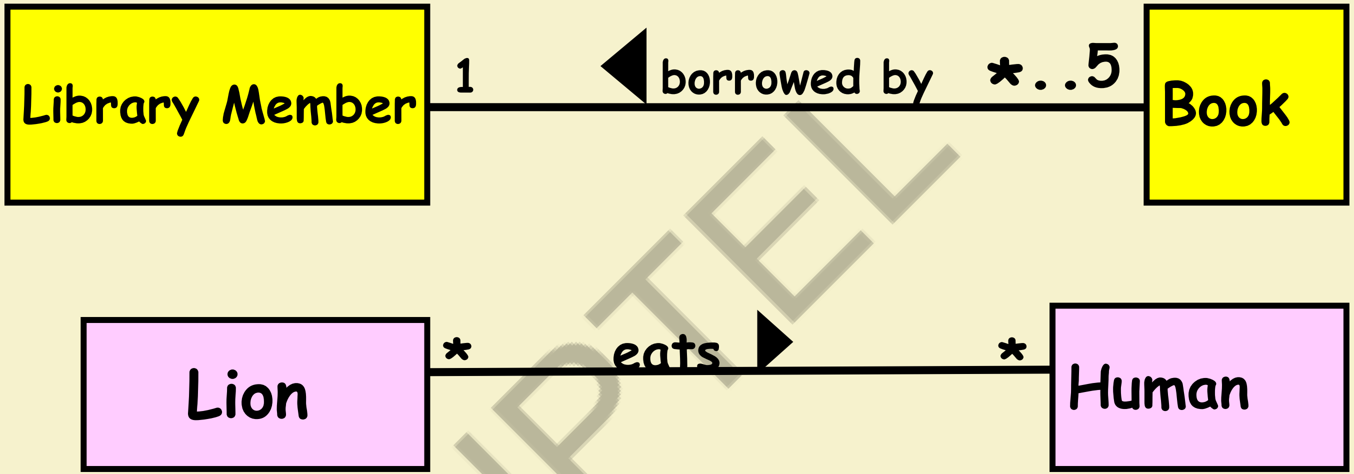
Association UML Syntax



- A Person works for a Company.

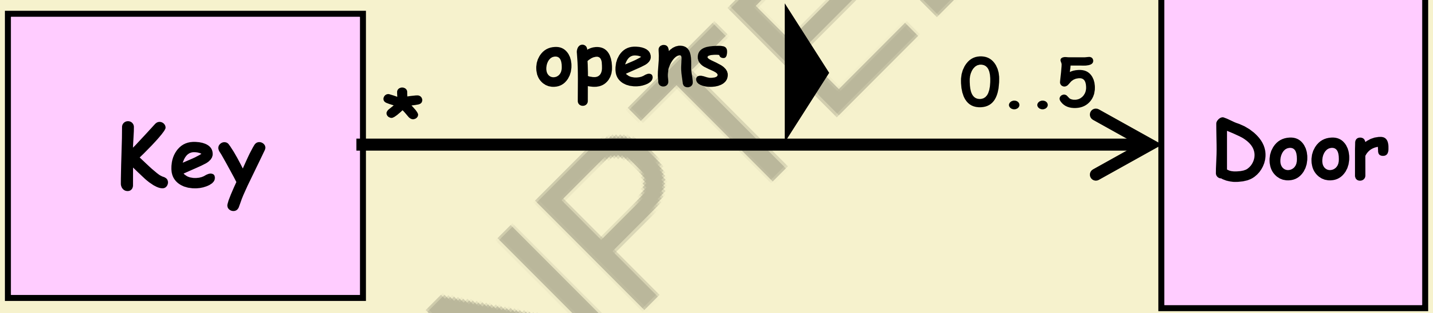


Association - More Examples



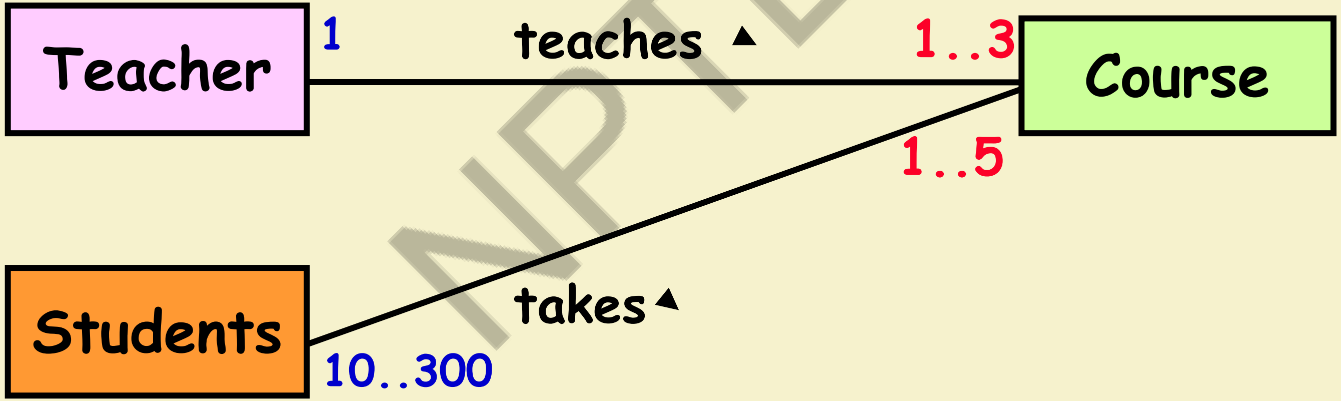
Multiplicity: The number of objects from one class that relate with a single object in an associated class.

Navigability



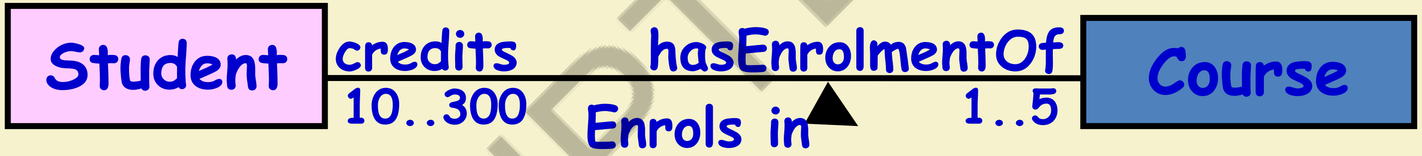
Association – Multiplicity

- A teacher teaches 1 to 3 courses (subjects)
- Each course is taught by only one teacher.
- A student can take between 1 to 5 courses.
- A course can have 10 to 300 students.

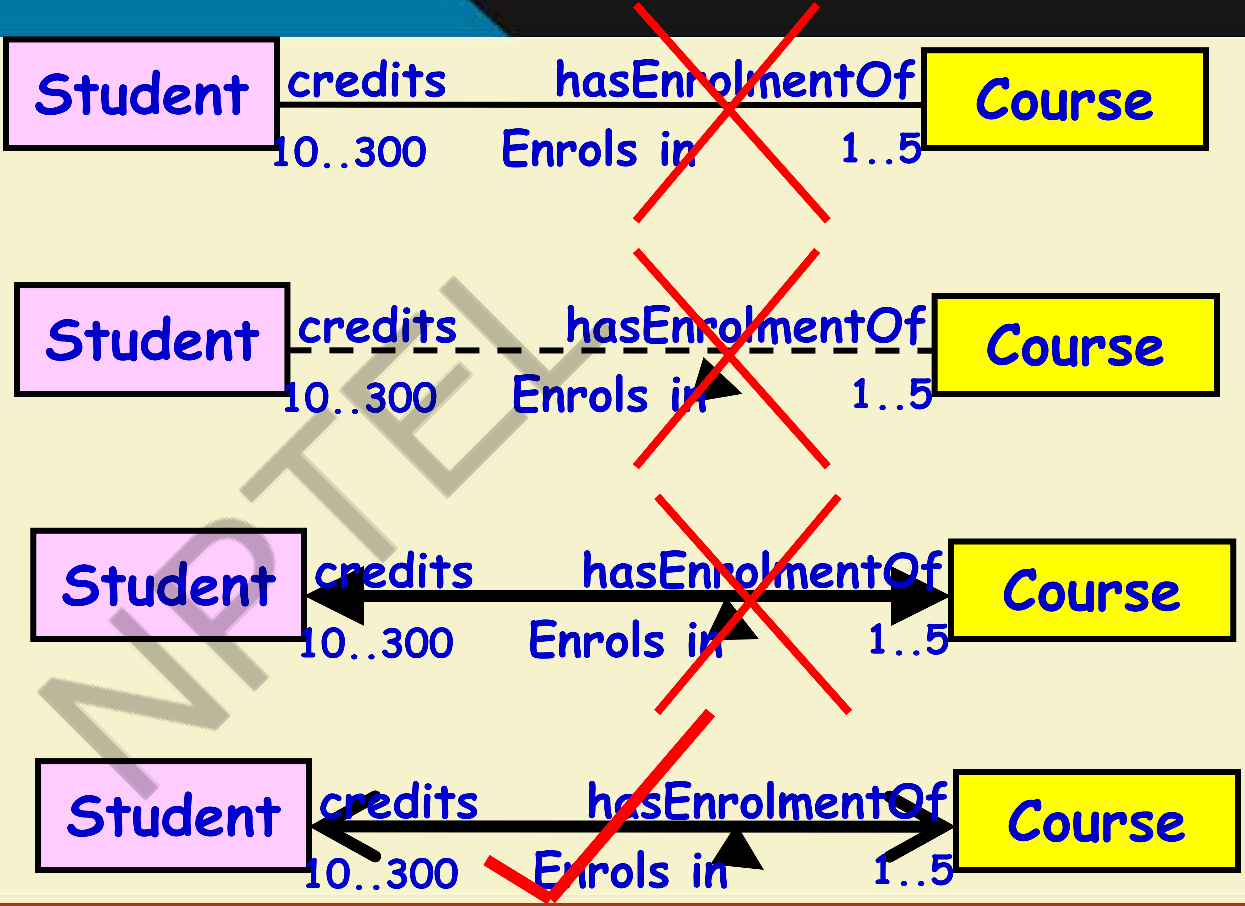


Quiz: Draw Class Diagram

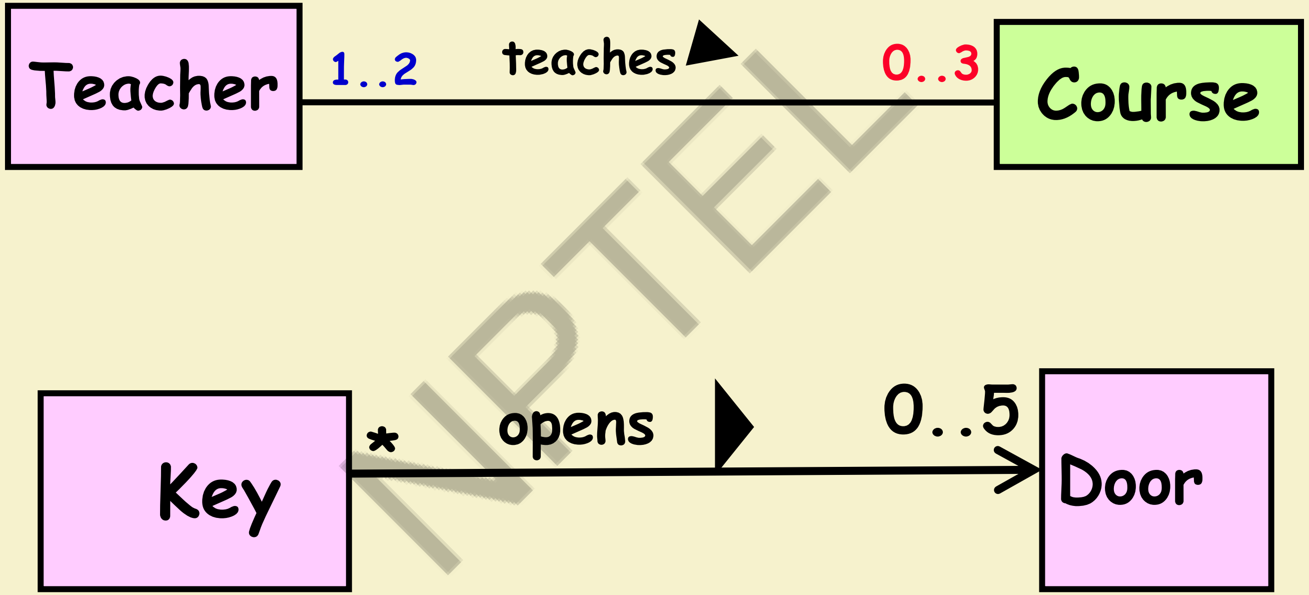
- A Student can take up to five Courses.
- A student needs to enroll in at least one course.
- Up to 300 students can enroll in a course.
- An offered subject in a semester should have at least 10 registered students.


Student
credits
10..300
hasEnrolmentOf
Enrols in
1..5
Course

Identify as Correct or Wrong



Quiz: Read the Diagram



Association and Link

- **A link:**
  - An instance of an association
  - Exists between two or more objects
  - **Dynamically created and destroyed as the run of a system proceeds**
- For example:
  - An employee joins an organization.
  - Leaves that organization and joins a new organization.