Object-Oriented Design using UML, Java and Patterns

Rajib Mall

Professor, CSE Department

IIT KHARAGPUR

About The Instructor

- RAJIB MALL

- B.E. , M.E., Ph.D from Indian Institute of Science, Bangalore

- Worked with Motorola (India)

- Shifted to IIT, Kharagpur in 1994
  - Currently Professor

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.**

Motivation

- Many learners start object-oriented programming by learning Java, C++, etc.
- With this they write woefully bad programs
- Often they write OO programs by intuitively extending procedural program design approaches.
- In this course, we discuss some essential concepts and techniques:
  - Should help develop good OO application code.

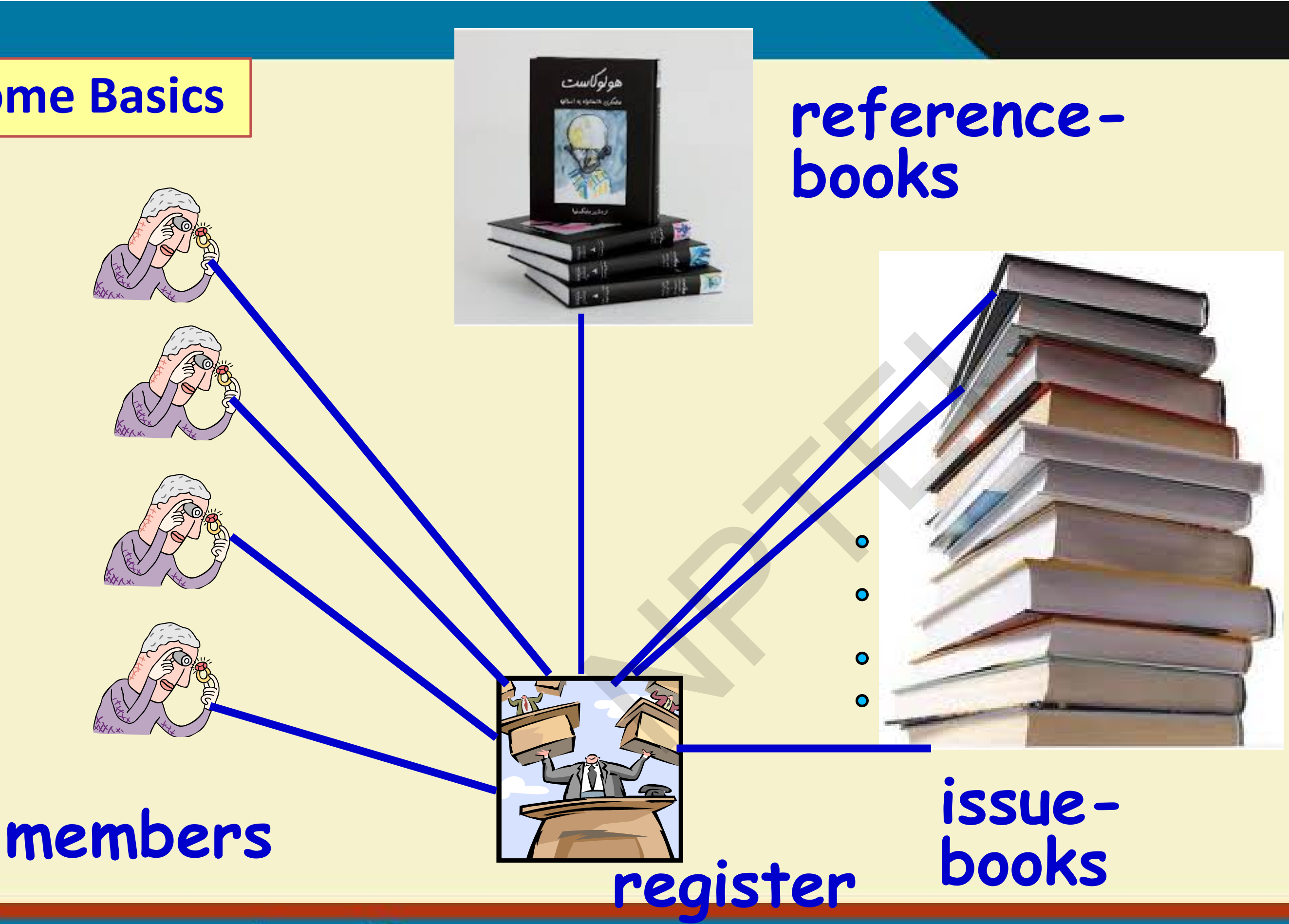
Course Plan

- UML (Unified Modelling Language)
- Object-oriented design process
- Arriving at better designs using design patterns

Assumption: You are familiar with basic Java or C++ programming.

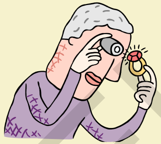
Some Basics

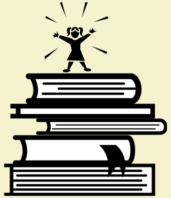
Schematic Object-Oriented Solution for LIS



Objects

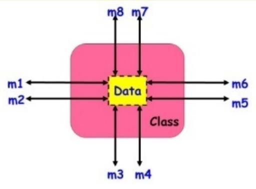
- A system is designed as a set of interacting objects.
- **Objects are often real-world entities:**
  - **Examples: an employee, a book etc.**





  - **Can also be conceptual objects :**
    - **Controller, manager, etc.**

- An object consists of data (attributes) and functions (methods) that operate on data.
  - **Encapsulation.**



Object Modelling Using UML

Caution: Learning UML can make you no more an expert OO designer, than learning English alphabet and grammar can make you an expert story writer…

For designing, need to learn object-oriented design methodology, patterns…

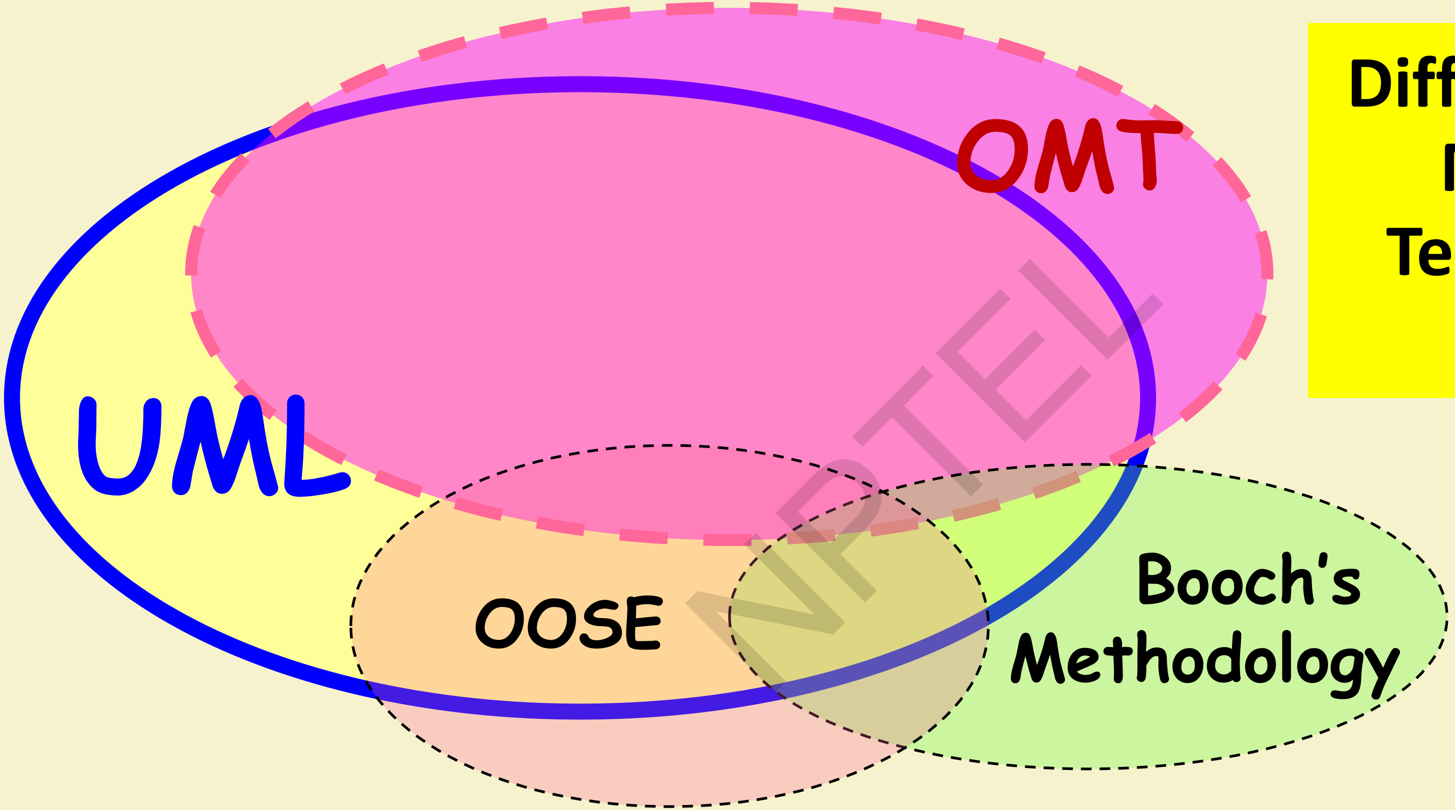
UML Origin

- 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]

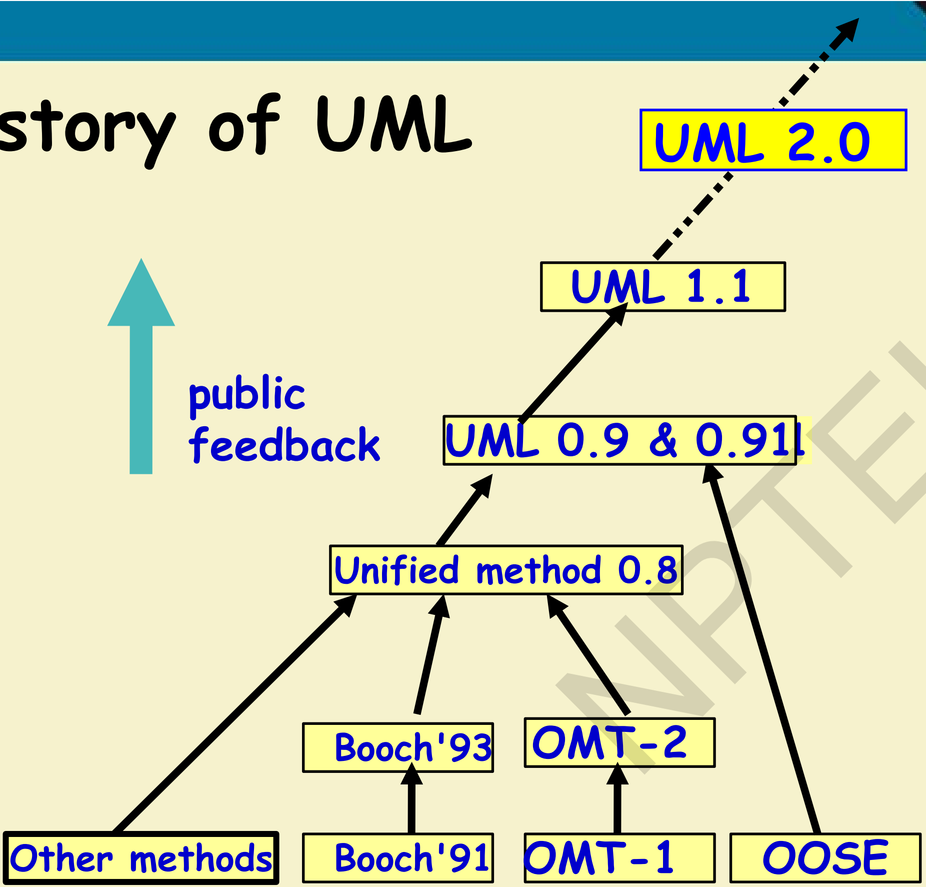
Different Object Modelling Techniques in UML

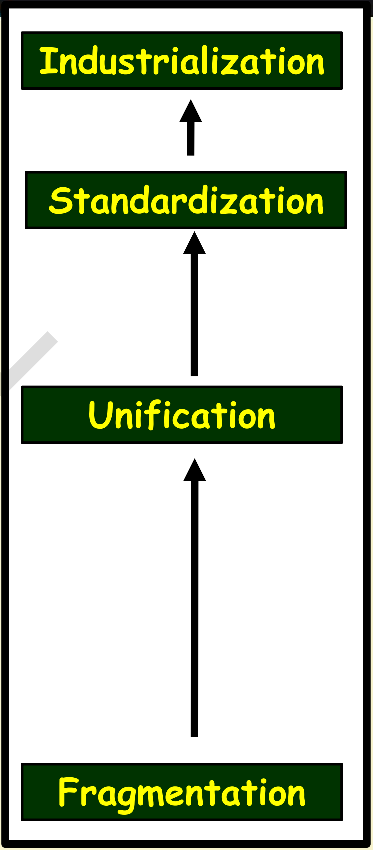


UML as A Standard

- 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 Build-to-order manufacturing

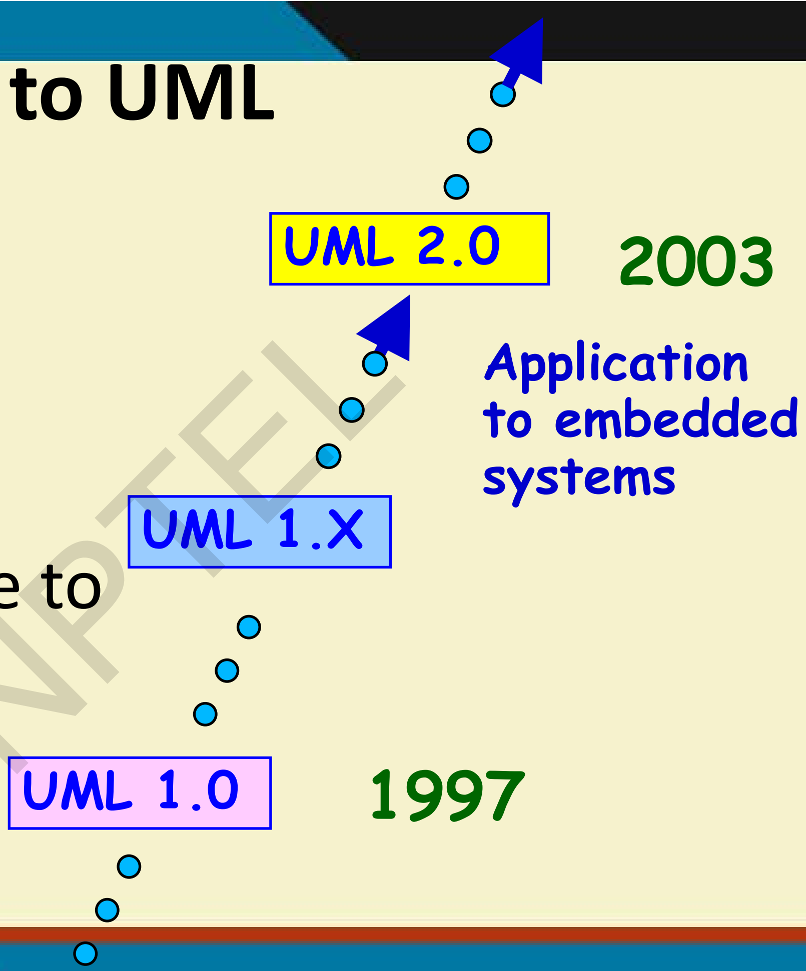
History of UML





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

Modelling vs. Designing

- Is Modelling the same as designing?

- A design is a model of the system.
  - But, every model is not a design of the system.

- From the requirements, an analysis model is created. (Analysis activity).

- Subsequently, the analysis model is refined into the design model.

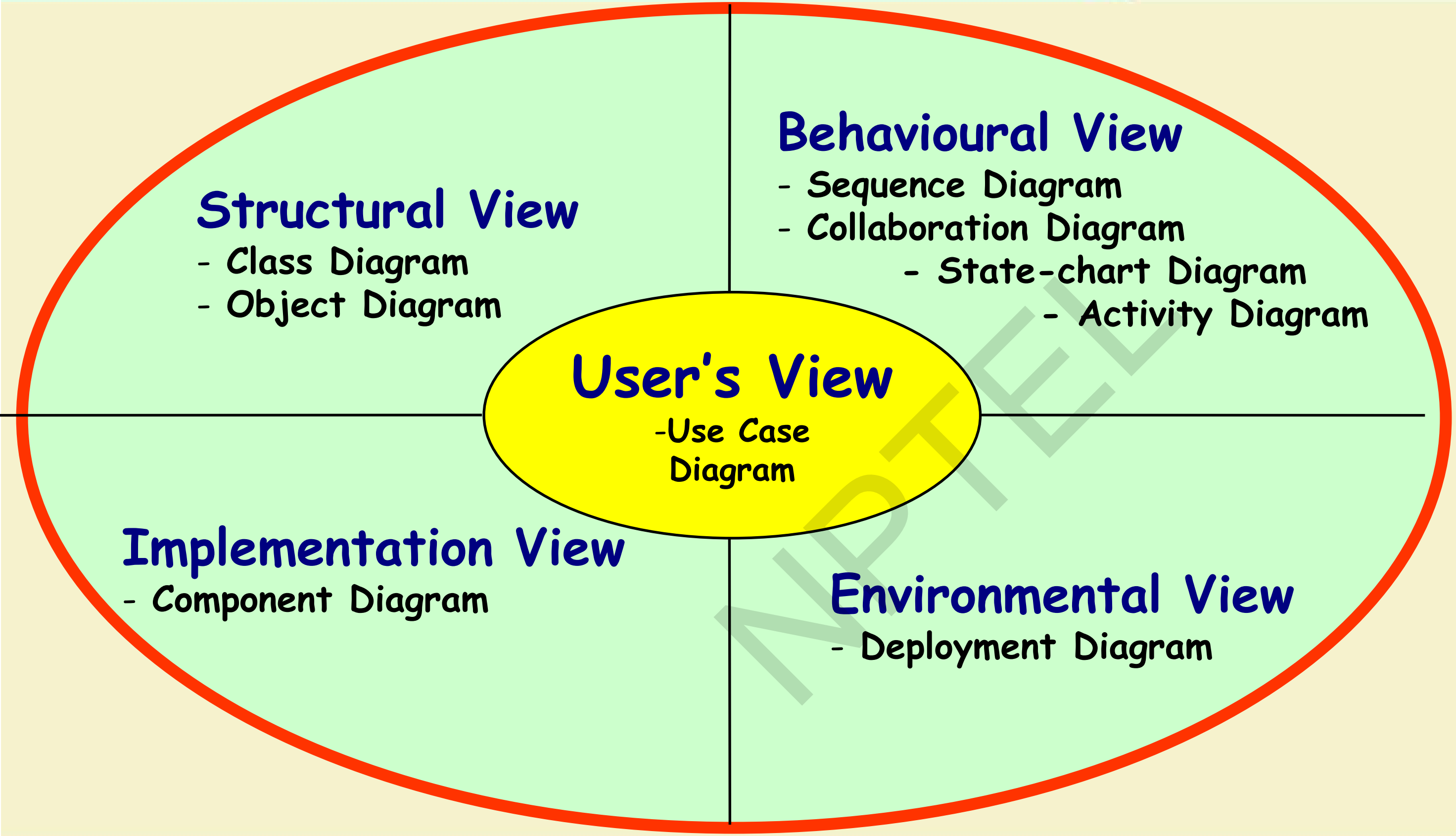
UML Diagrams

- Nine diagrams in UML 1.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.

UML Model Views

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

Diagrams and views in UML



Structural View

- Class Diagram
- Object Diagram

Behavioural View

- Sequence Diagram
- Collaboration Diagram
- State-chart Diagram
- Activity Diagram

User's View

-Use Case Diagram

Implementation View

- Component Diagram

Environmental View

- Deployment Diagram

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?

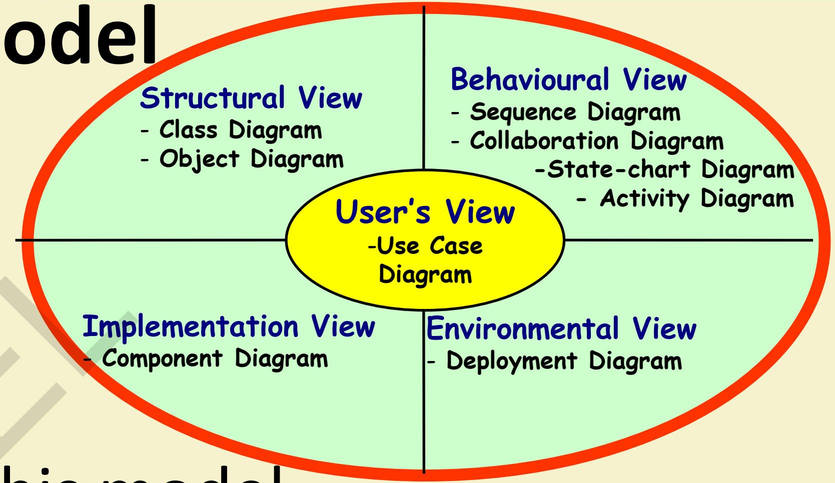
- NO
- **For a simple system:**
  - Use case diagram, class diagram and one of the interaction diagrams only.
- **State chart diagram:**
  - When class has significant states.
  - When states are only one or two, state chart model becomes trivial.
- **Deployment diagram:**
  - When system has many hardware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

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

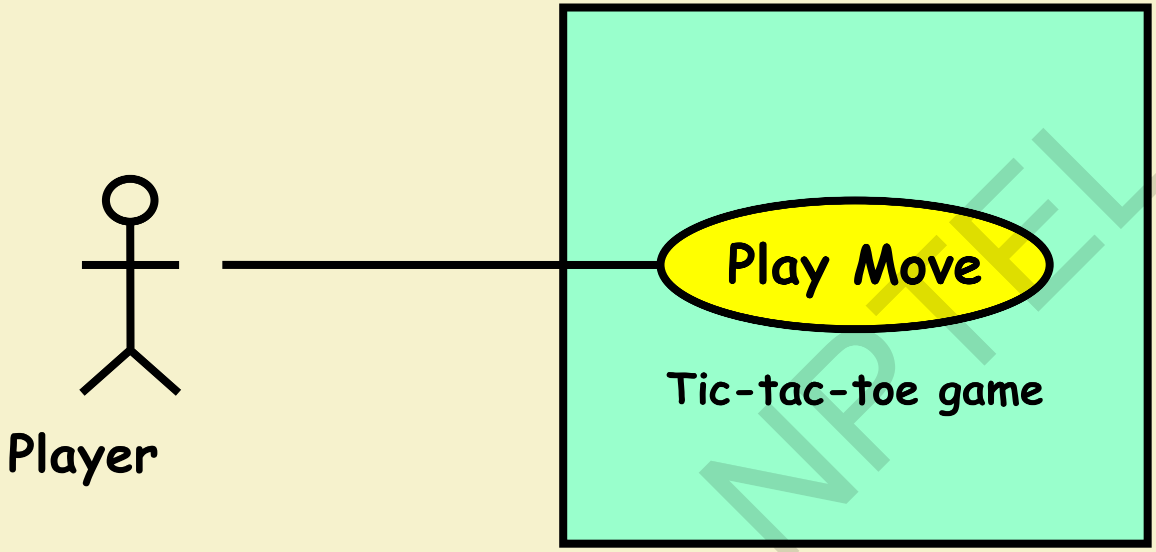
- **issue-book**
- **query-book**
- **return-book**
- **create-member**
- **add-book, etc.**

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.**

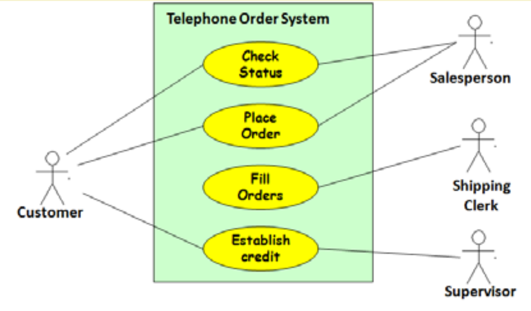
First Example: Use Case Diagram



Use case model

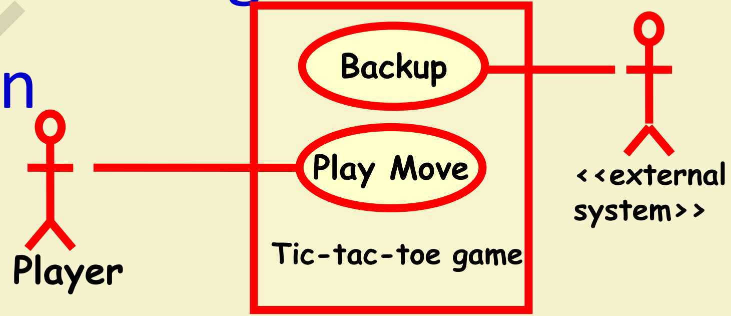
Why Develop A Use Case Diagram?

- Serves as requirements specification.
- How identifying actors helps 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).



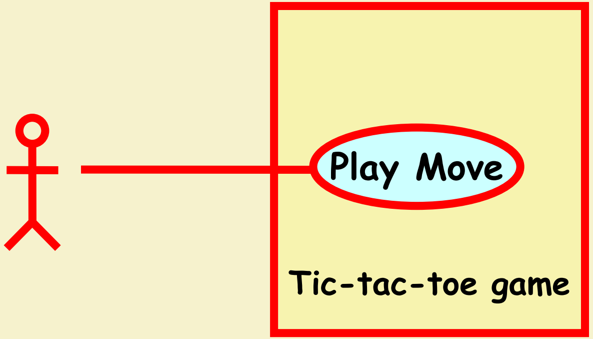
Representation of Use Cases

- 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?

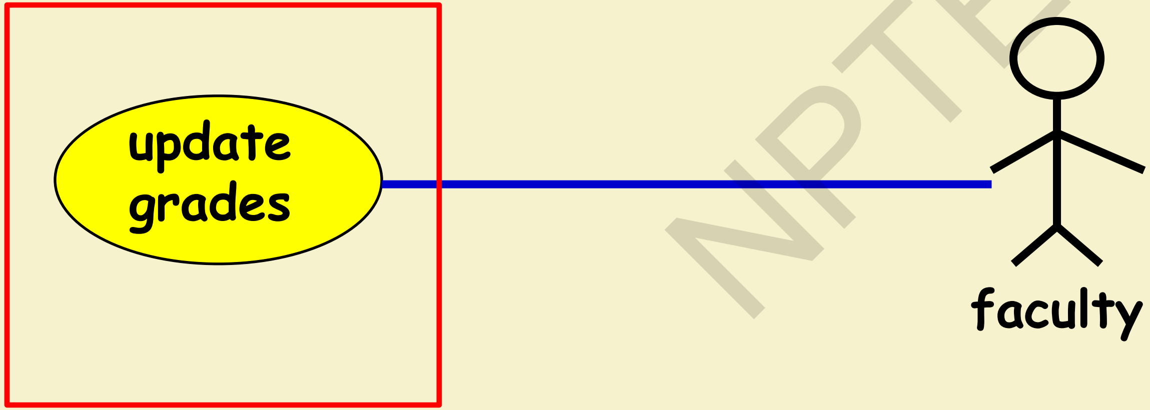
- A connection is an association between an actor and a use case.



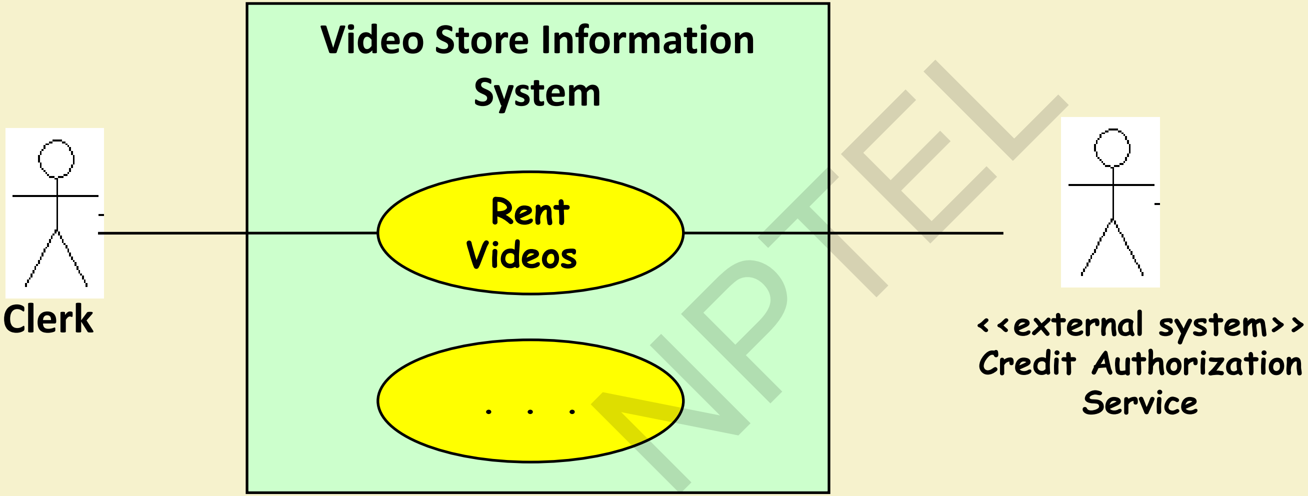
- Depicts a usage relationship
- Connection does not indicate data flow…

Relationships between Use Cases and Actors

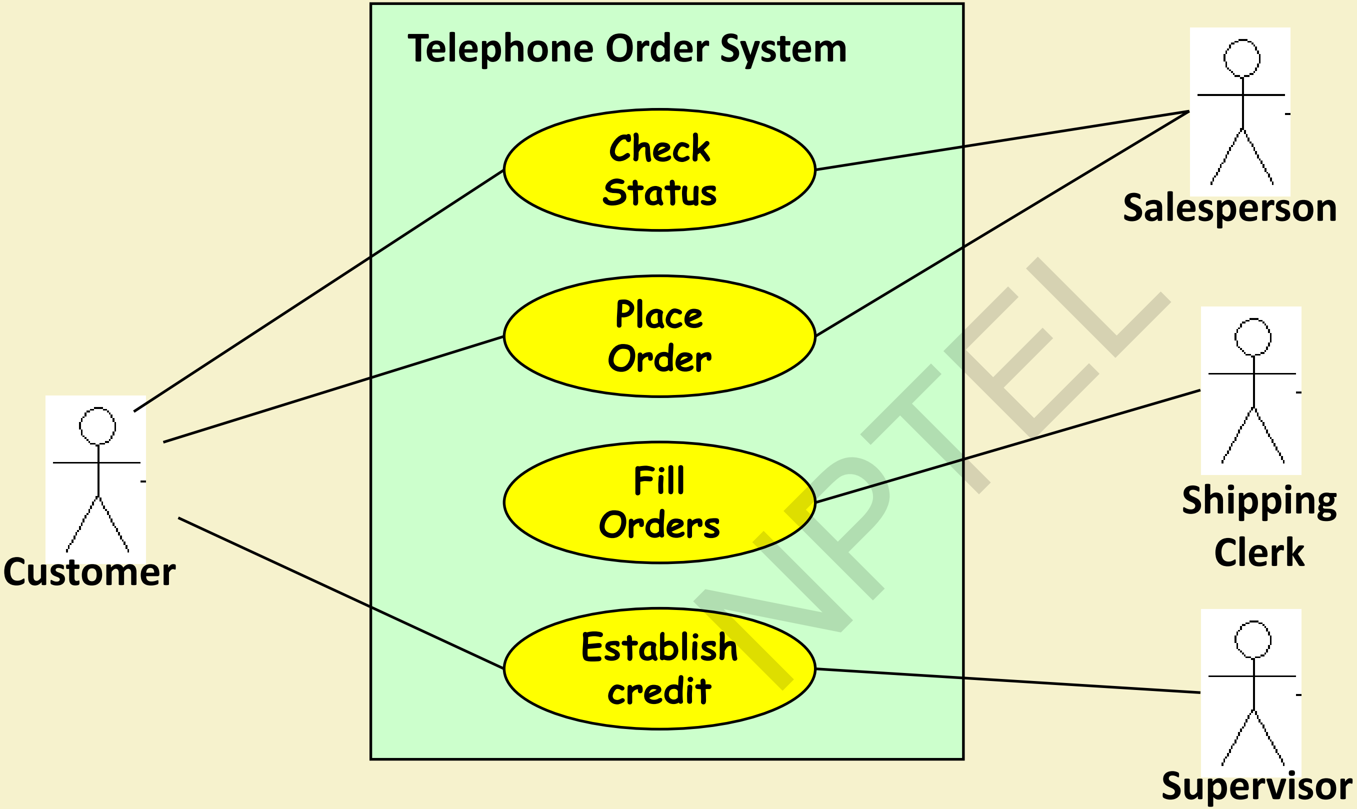
- Association relation indicates that the actor and the corresponding use case communicate with one another.



Another Example Use Case Diagram



Yet Another Use Case Example

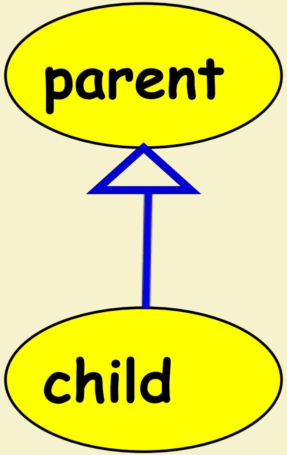


Factoring Use Cases

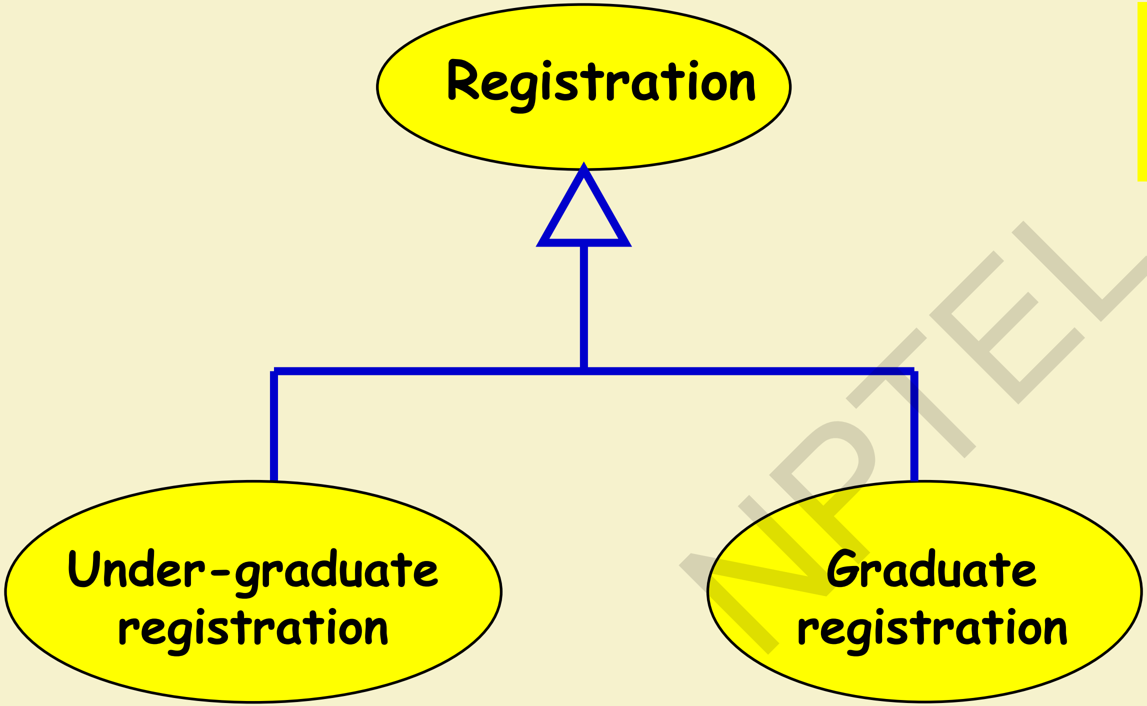
- Two main reasons for factoring:
  - Complex use cases need to be factored into simpler use cases
  - Helps represent common behavior across different use cases
- Three ways of factoring:
  - Generalization
  - Include
  - Extend

Generalization

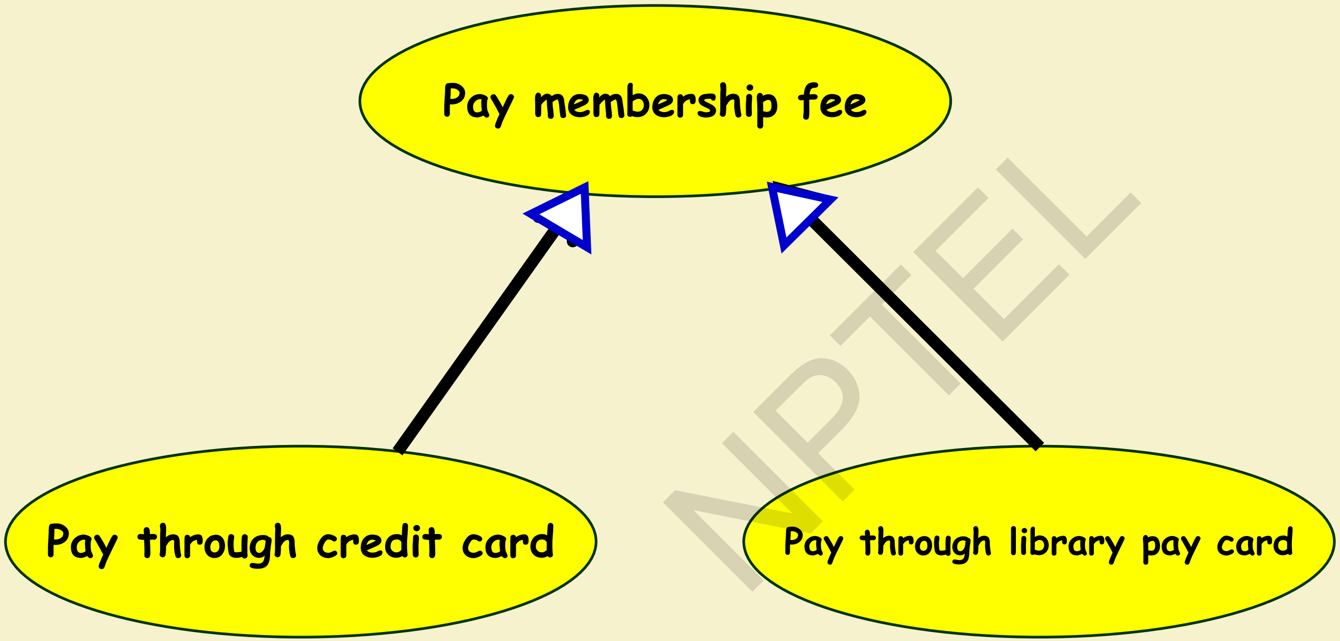
- The child use case inherits the behavior of the parent use case.
  - The child may add to or override some of the behavior of its parent.



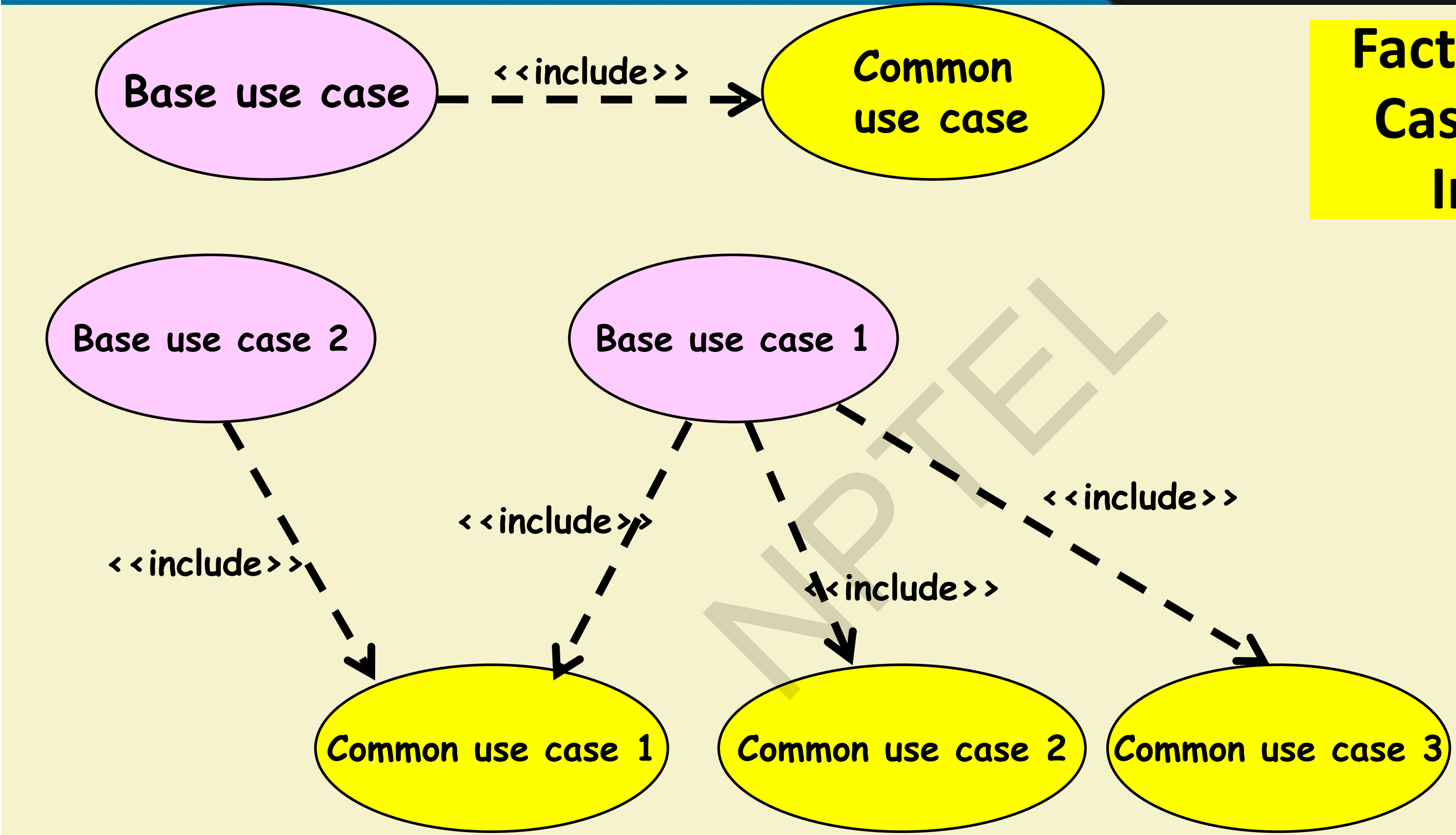
Generalization: Example 1



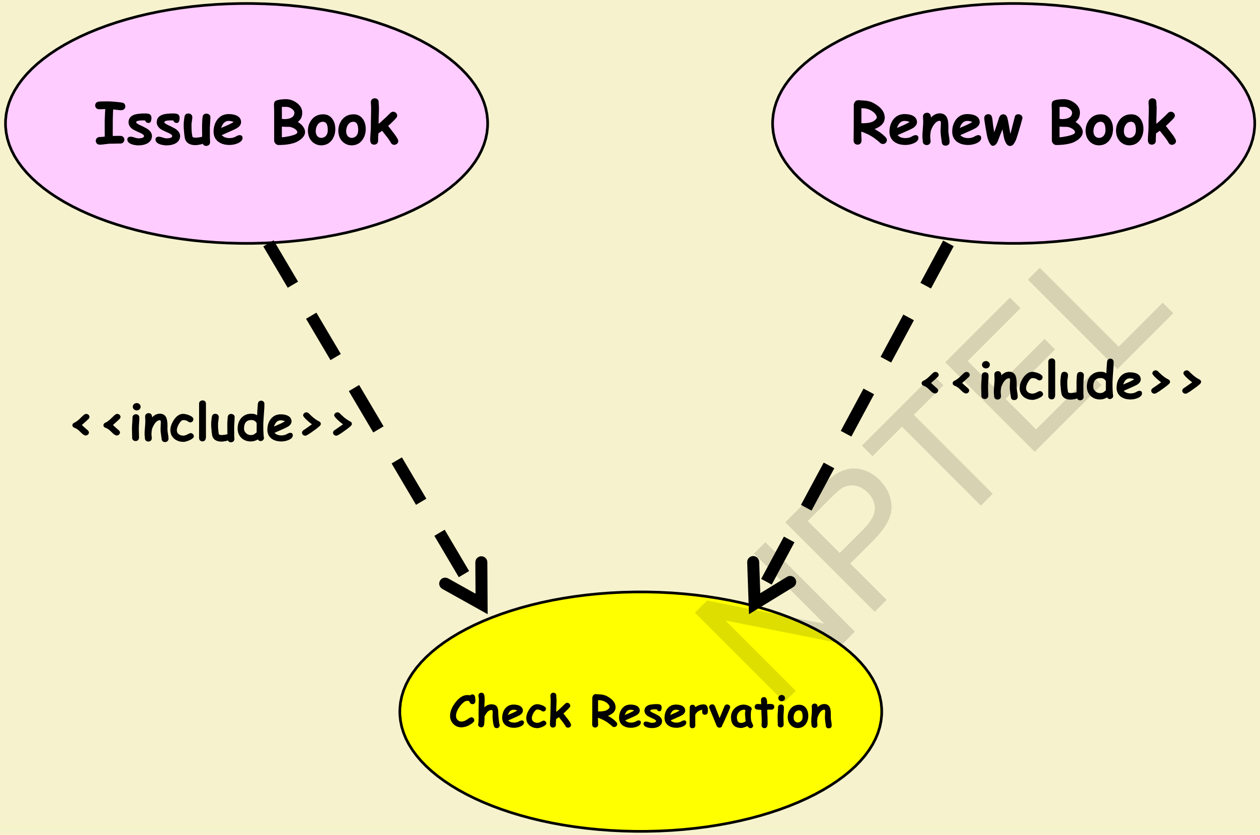
Generalization: Example 2



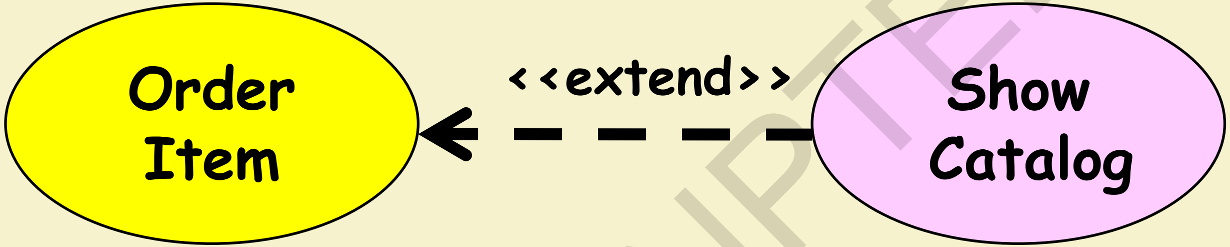
Factoring Use Cases Using Include



Factoring Use Cases Using Include: Example

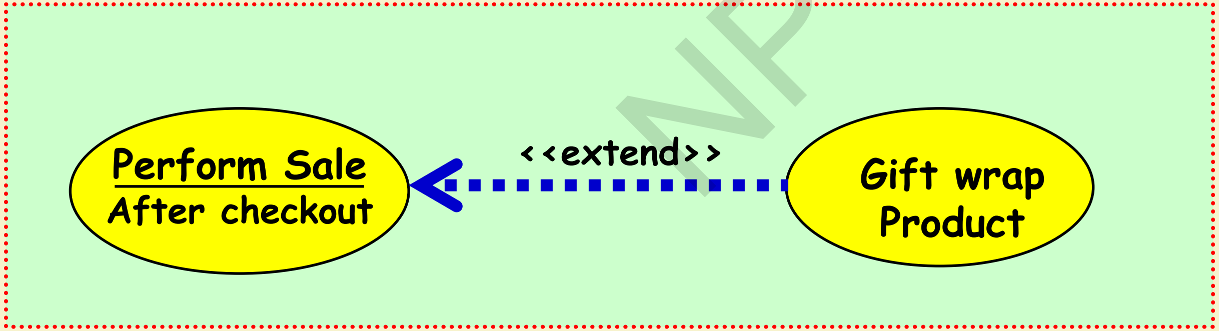


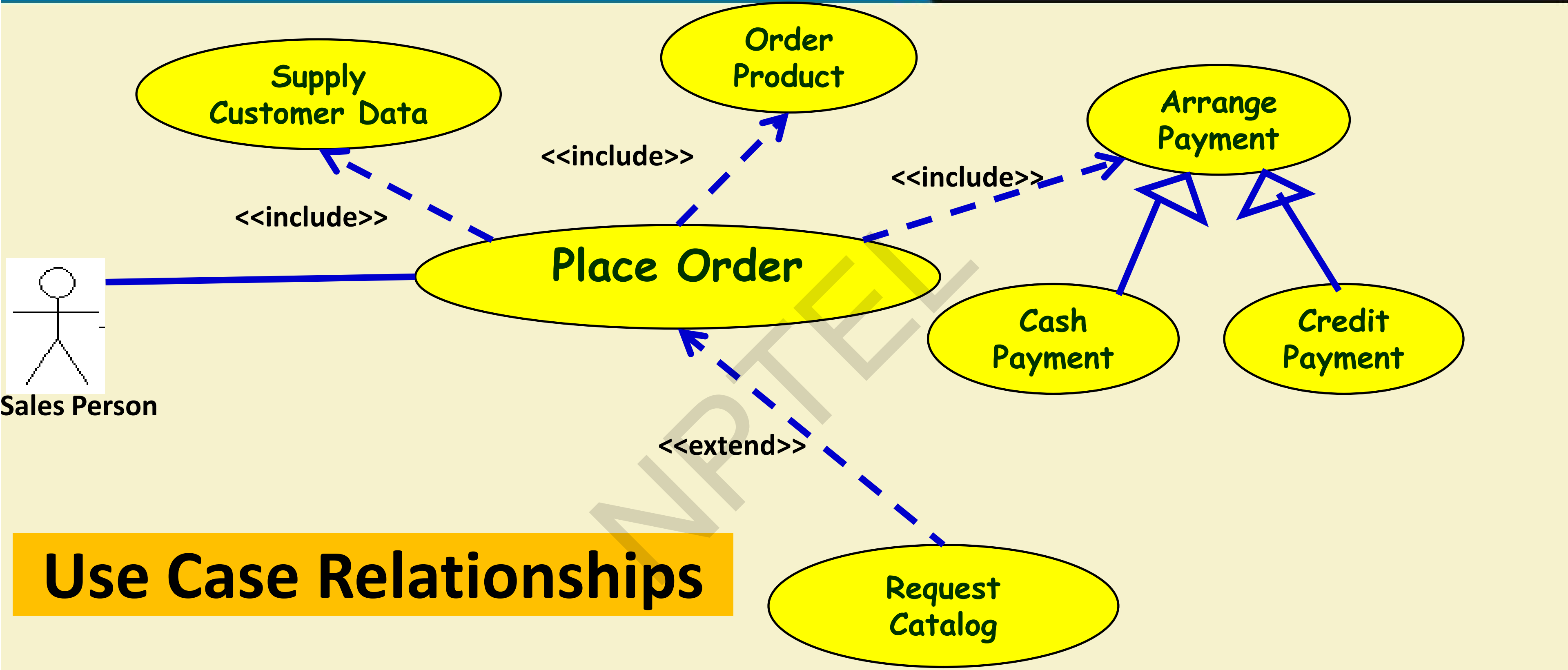
Factoring A Use Case Using Extend: Example



Extension Point

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





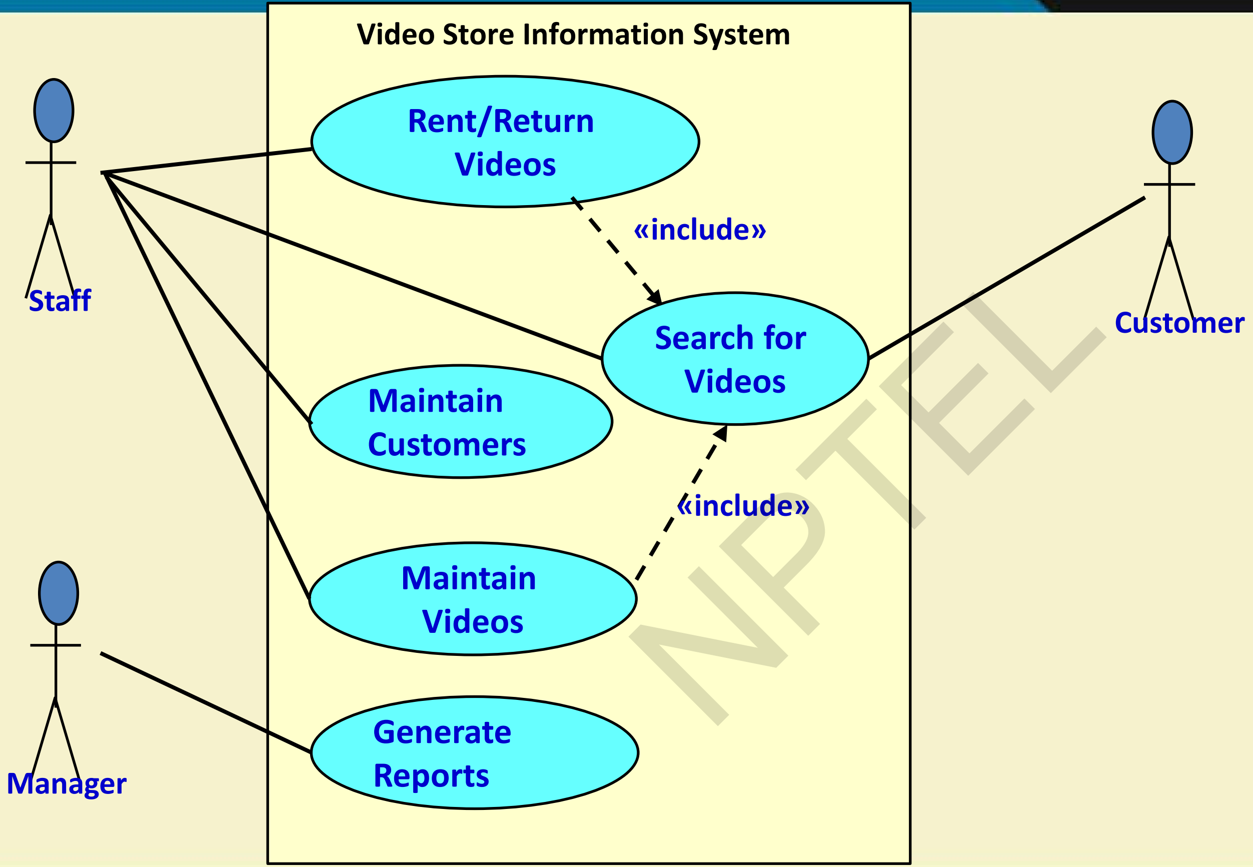
Use Case Relationships

- Video Store Information System supports the following business functions:

Example 1: Video Store Information System

  - Entering the information about all videos that the store owns
    - This database is searchable by staff and all customers
  - Store 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.
  - Store Manager can generate various reports.

Example 1: Solution

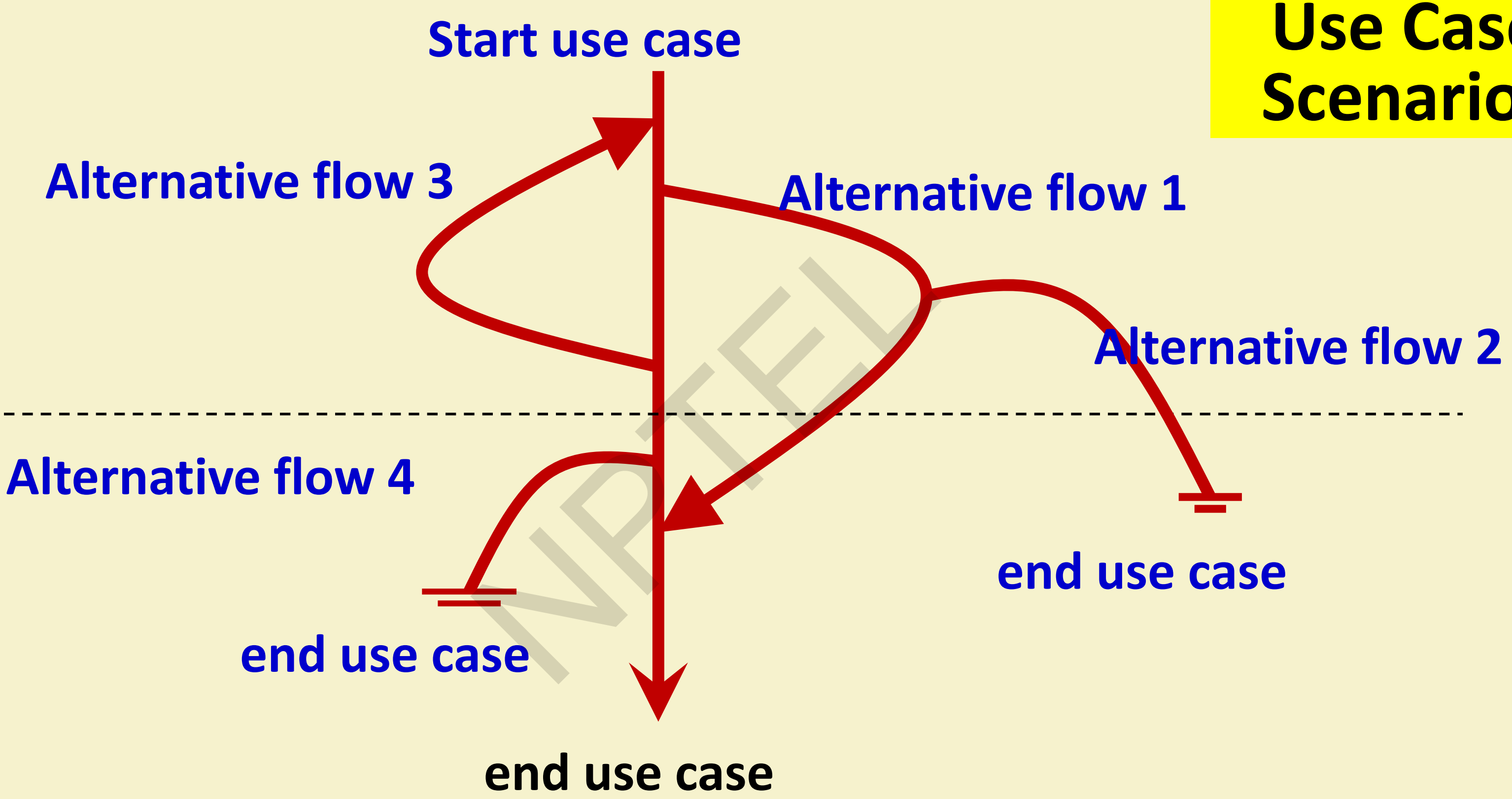


Use Case Description

- 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

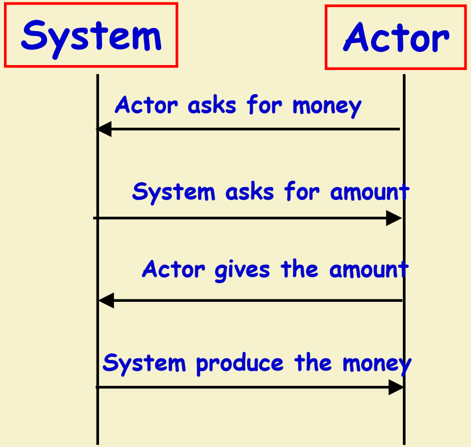
- **Actors:** traveler
- **Preconditions:**
  - Traveler has logged on to the system and selected 'change flight itinerary' option
- **Basic course**

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 mouse key”

Identification of Use Cases

1. Actor-based:

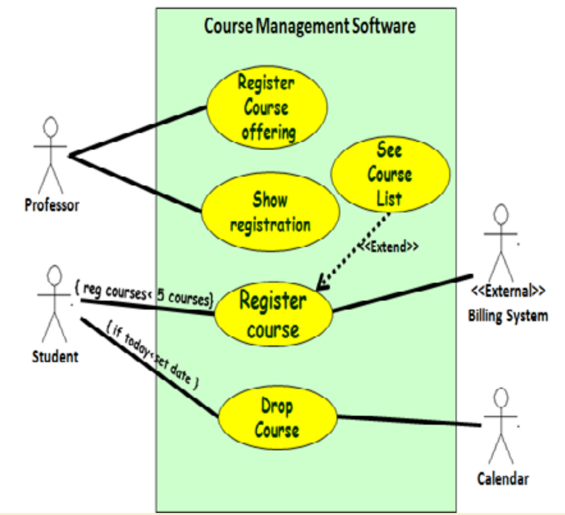
- Identify the actors.
- For each actor, identify the use cases they initiate or participate in.

2. Event-based

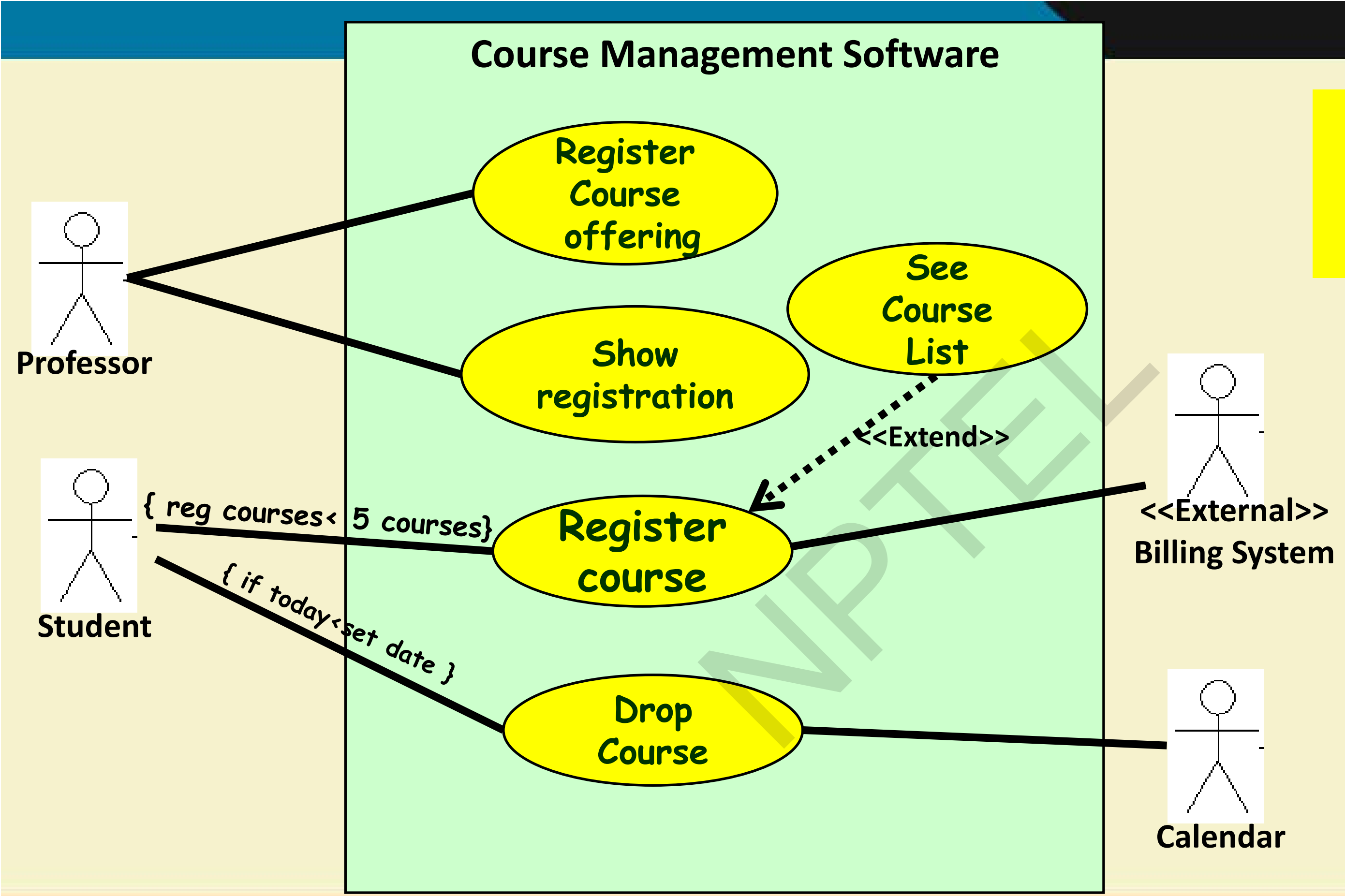
- Identify the external events that the system must respond to.
- Relate the events to actors and use cases.

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 certain 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


Course Management Software
Register Course offering
See Course List
Show registration
<<Extend>>
Register course
Drop Course
Professor
Student
{ reg courses< 5 courses}
{ if today<set date }
<<External>> Billing System
Calendar