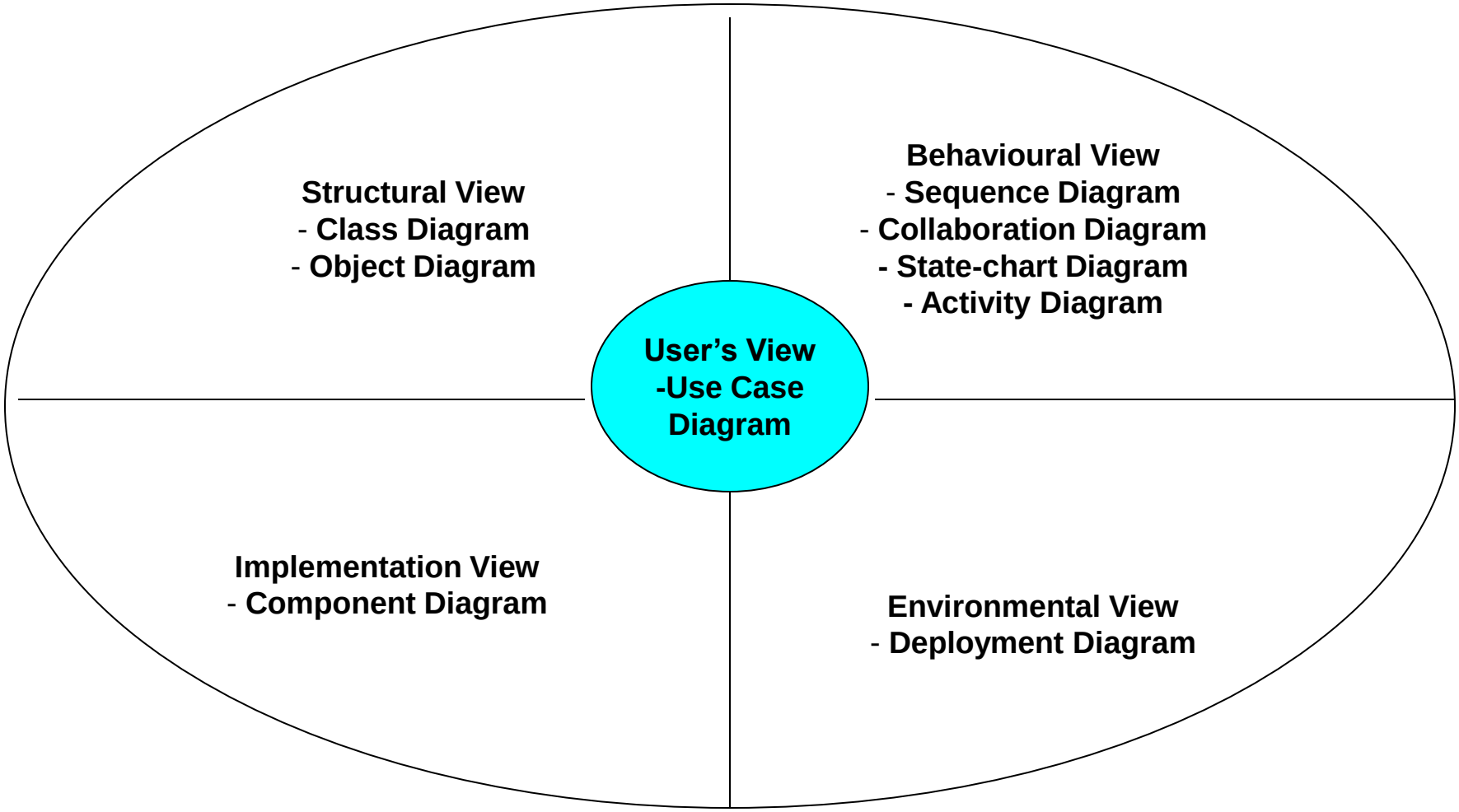
Chapter 3

User's view

UML Diagrams



Diagrams and views in UML

Use Case Diagram

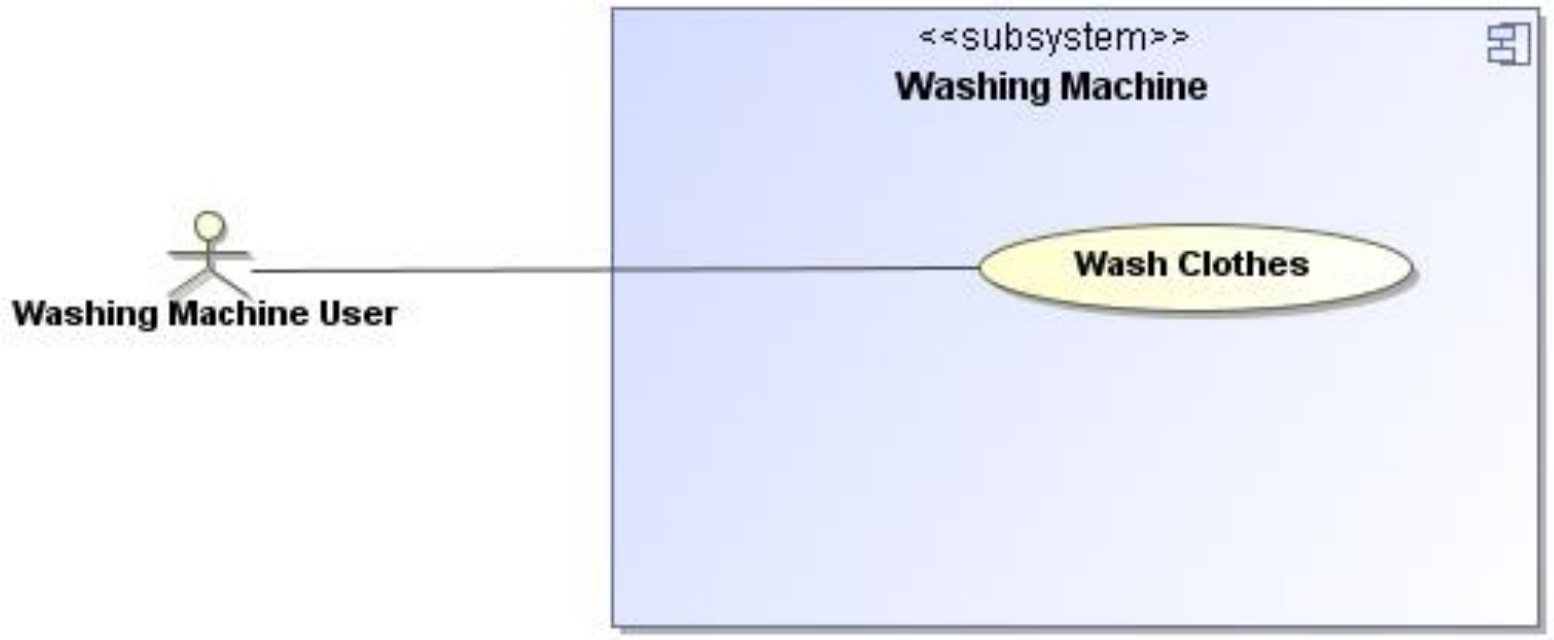
- A Use Case Model describes the proposed functionality of a new system.
- A Use Case represents a discrete unit of interaction between a user (human or machine) and the system.
- This interaction is a single unit of meaningful work.

- Create Account
- View Account Details.

Representation of Use Cases

- **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 a **stereotype**

An Use Case Diagram



Use Cases

- Different ways in which a system can be used by the users
- Corresponds to the high-level requirements
- Represents transaction between the user and the system
- Defines external behavior without revealing internal structure of system
- Set of related scenarios tied together by a common goal

Use Cases cont...

- Normally, use cases are independent of each other
- Implicit dependencies may exist

Example: In Library Automation System,

renew-book & 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.

Example of Use Cases

- For library information system

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

Components

- **Actors**: An actor portrays any entity (or entities) that performs certain roles in a given system.
- **Use case:** A use case in a use case diagram is a visual representation of a distinct business functionality in a system.
- **System boundary:** A system boundary defines the scope of what a system will be.

Components cont…

- **Include:** When a use case is depicted as using the functionality of another use case in a diagram, this relationship between the use cases is named as an *include* relationship.
- **Extend:** In an *extend* relationship between two use cases, the child use case adds to the existing functionality and characteristics of the parent use case.
- **Generalizations:** A *generalization* relationship is also a parent-child relationship between use cases.

Components cont…

- General comments and notes describing the use case.
- Requirements - The formal functional requirements of things that a Use Case must provide to the end user.

- ability to update order

- These correspond to the functional specifications found in structured methodologies.
- Form a contract that the Use Case performs some action.

Components cont…

- Constraints :The formal rules and limitations a Use Case operates under, defining what can and cannot be done.

1. Pre-conditions that must have already occurred or be in place before the use case is run.

- <create order> must precede <modify order>

2. Post-conditions that must be true once the Use Case is complete;

-<order is modified and consistent>

3. Invariants that must always be true throughout the time the Use Case operates;

- an order must always have a customer number.

Components cont…

- Scenarios: Formal, sequential descriptions of the steps taken to carry out the use case
- Scenario diagrams: Sequence diagrams to depict the workflow
- Additional attributes:

- implementation phase
- version number
- complexity rating
- Stereotype
- status.

Coverage of Use Case

- A use case specification document should cover the following areas:
- **Actors:** List the actors that interact and participate in this use case.
- **Pre-conditions:** Pre-conditions that need to be satisfied for the use case to perform.
- **Post-conditions:** Define the different states in which you expect the system to be in, after the use case executes.

Coverage of Use Case cont...

- **Basic Flow:** List the basic events that will occur when this use case is executed. Include all the primary activities that the use case will perform. Be fairly descriptive when defining the actions performed by the actor and the response of the use case to those actions. This description of actions and responses are your functional requirements. These will form the basis for writing the test case scenarios for the system.
- **Alternative flows:** Any subsidiary events that can occur in the use case should be listed separately. Each such event should be completed in itself to be listed as an alternative flow. A use case can have as many alternative flows as required. But remember, if there are too many alternative flows, you need to revisit your use case design to make it simpler and, if required, break the use case into smaller discrete units.

Coverage of Use Case cont...

- **Special Requirements:** Business rules for the basic and alternative flows should be listed as special requirements in the use case narration. These business rules will also be used for writing test cases. Both success and failure scenarios should be described here.
- **Use case relationships:** For complex systems, it is recommended that you document the relationships between use cases. If this use case extends from other use cases or includes the functionality of other use cases, these relationships should be listed here. Listing the relationships between use cases also provides a mechanism for traceability.

Courseware Management System

The organization offers a variety of courses in a variety of areas such as learning management techniques and understanding different software languages and technologies. Each course is made up of a set of topics. Tutors in the organization are assigned courses to teach according to the area that they specialize in and their availability. The organization publishes and maintains a calendar of the different courses and the assigned tutors every year. There is a group of course administrators in the organization who manage the courses including course content, assign courses to tutors, and define the course schedule. The training organization aims to use the Courseware Management System to get a better control and visibility to the management of courses as also to streamline the process of generating and managing the schedule of the different courses.

Terms and entities

- Courses and Topics that make up a course
- Tutors who teach courses
- Course administrators who manage the assignment of the courses to tutors
- Calendar or Course Schedule is generated as a result
- Students who refer to the Course schedule or Calendar to decide which courses they wish to take up for study

Actors

- Tutors, Course administrators, Students

students are not the potential active participants for this system, we will drop them from the list of actors. Similarly, tutors are not active participants from our system's perspective, and hence, we will exclude tutors from our list if roles. Yet, we will still record them in our use case model since we do not wish to lose this business information. Our final list of primary actors has now come down to only one:

- Course administrators

Use Cases

- Manage courses
- Manage course assignments

Within the "Manage courses" use case, we can identify the following sub processes:

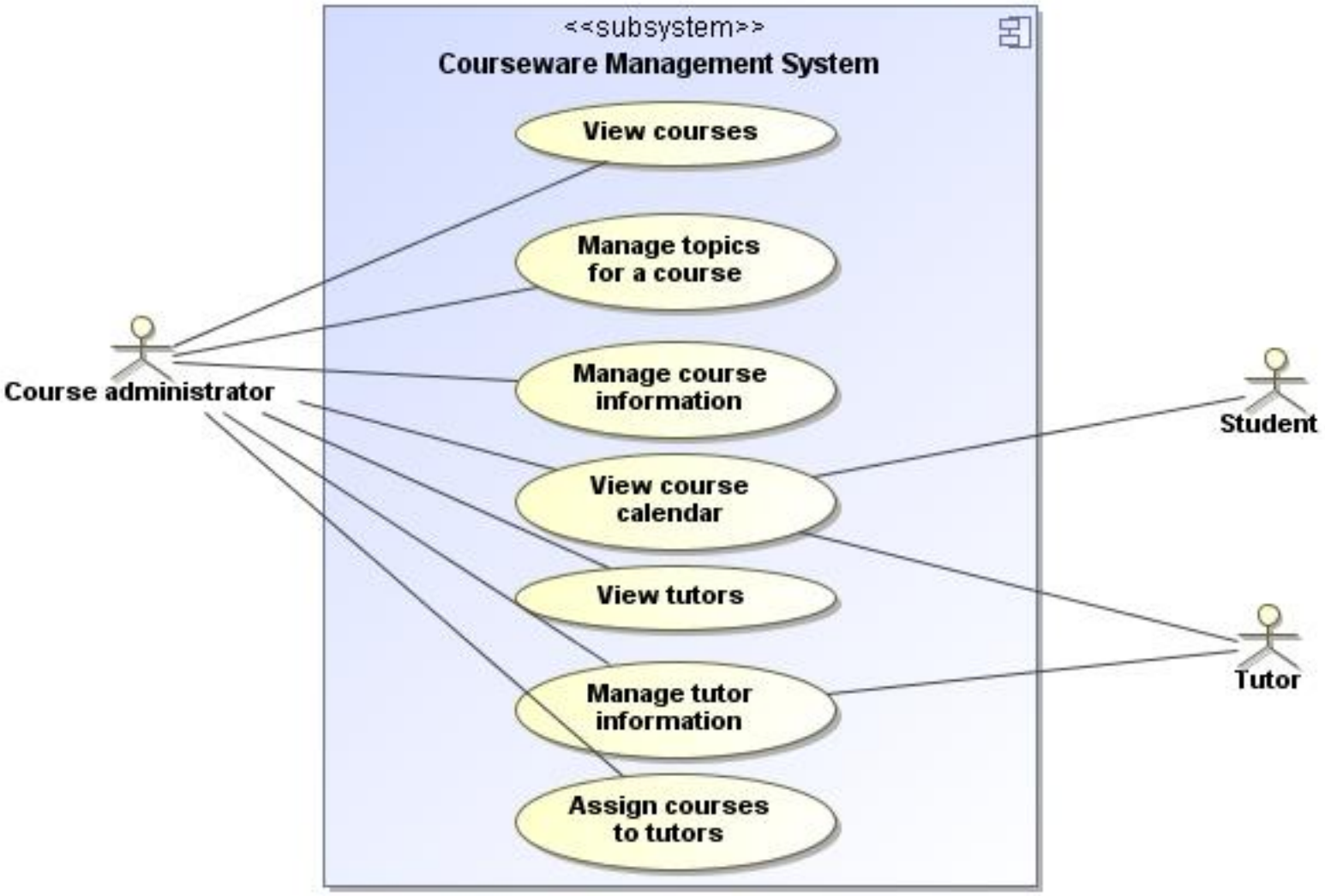
- View courses
- Manage topics for a course
- Manage course information

Use Cases cont…

Within the "Manage course assignment" use case are:

- View course calendar
- View tutors
- Manage tutor information
- Assign courses to tutors

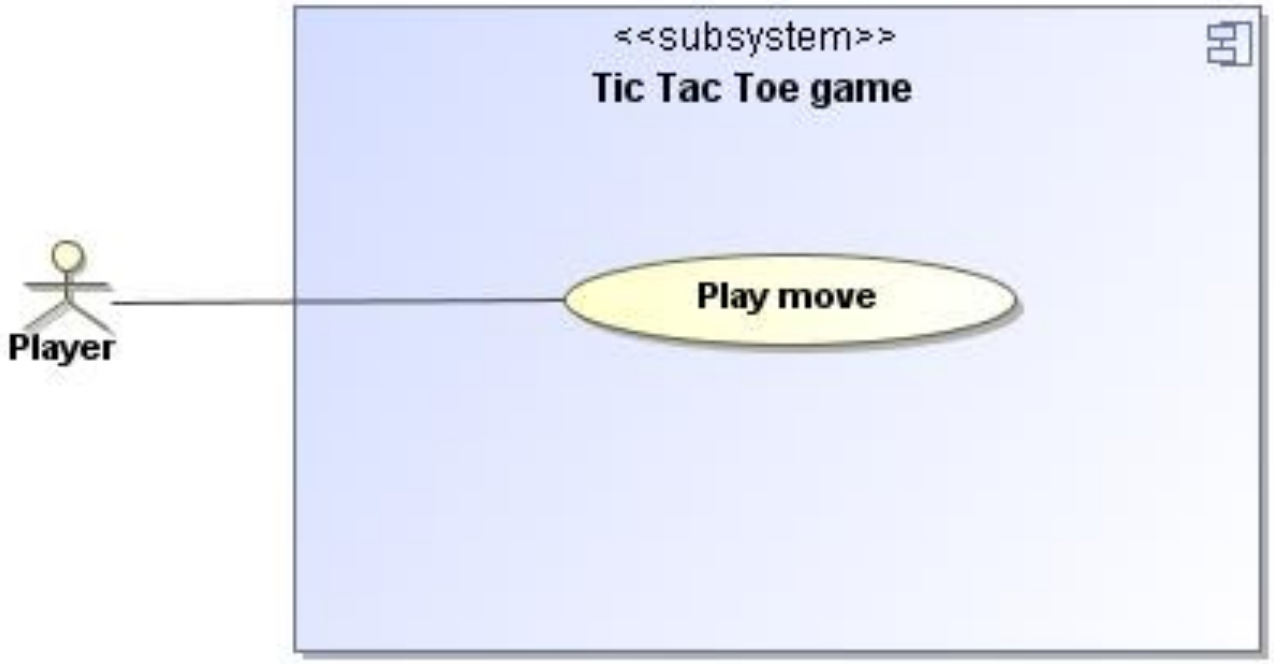
Use Case Diagram



Tic-Tac-Toe Computer Game:

- Tic-tac-toe is a computer game in which a human player and the computer make alternate moves on a 3 × 3 square. A move consists of marking a previously unmarked square. The player who is first to place three consecutive marks along a straight line (i.e., along a row, column, or diagonal) on the square wins. As soon as either of the human player or the computer wins, a message congratulating the winner should be displayed. If neither player manages to get three consecutive marks along a straight line, and all the squares on the board are filled up, then the game is drawn. The computer always tries to win a game.

An Example Use Case Diagram



Why Develop A Use Case Diagram?

- Serves as requirements specification
- How are actor identification useful in software development:

− User identification helps in implementing appropriate interfaces for different categories of users

− Another use in preparing appropriate documents (e.g. **user's manual**).

Factoring Use Cases

- Complex use cases need to be factored into simpler use cases
- Represent common behavior across different use cases
- Three ways of factoring:

– **Generalization**

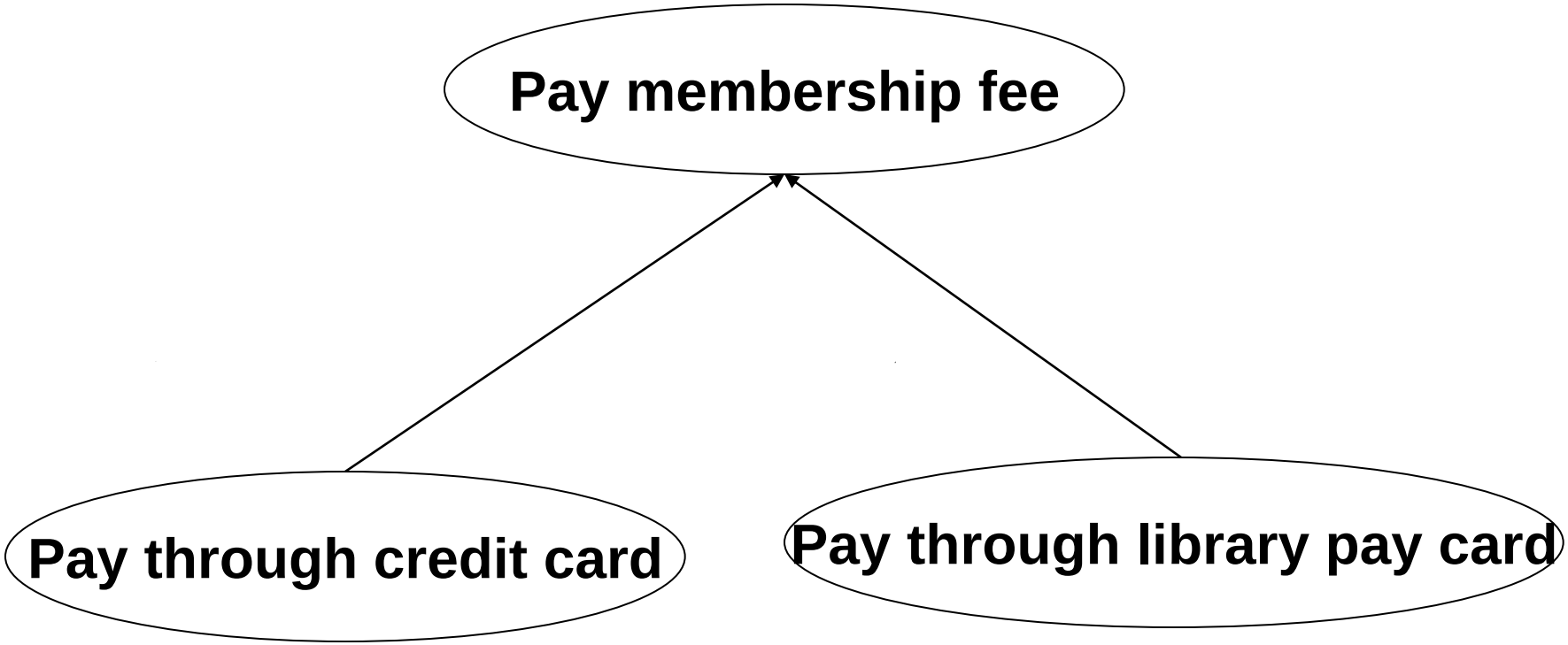
– **Includes**

– **Extends**

Generalization

- When one use case is slightly different from other.
- Works the same way with use cases as it does with classes.
- Child use case inherits the behavior and meaning of the perent use case.

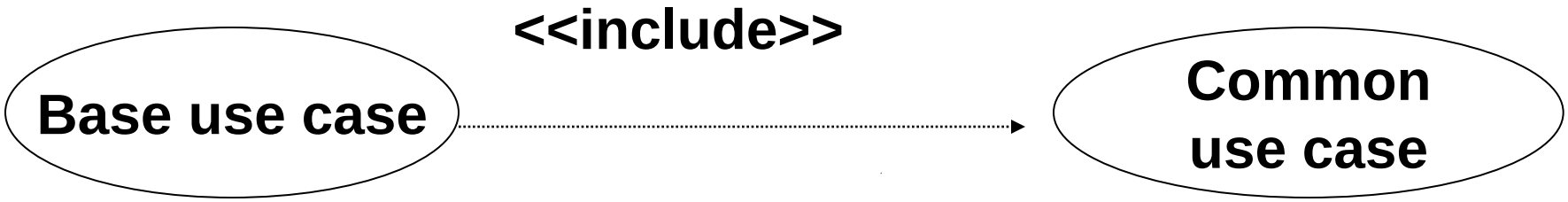
Factoring Use Cases Using Generalization



Includes

- Uses in the older versions of UML (prior to UML 1.1)
- One use case including the behavior of another use case in its sequence of events and actions.

Factoring Use Cases Using Includes



Personal Library System:

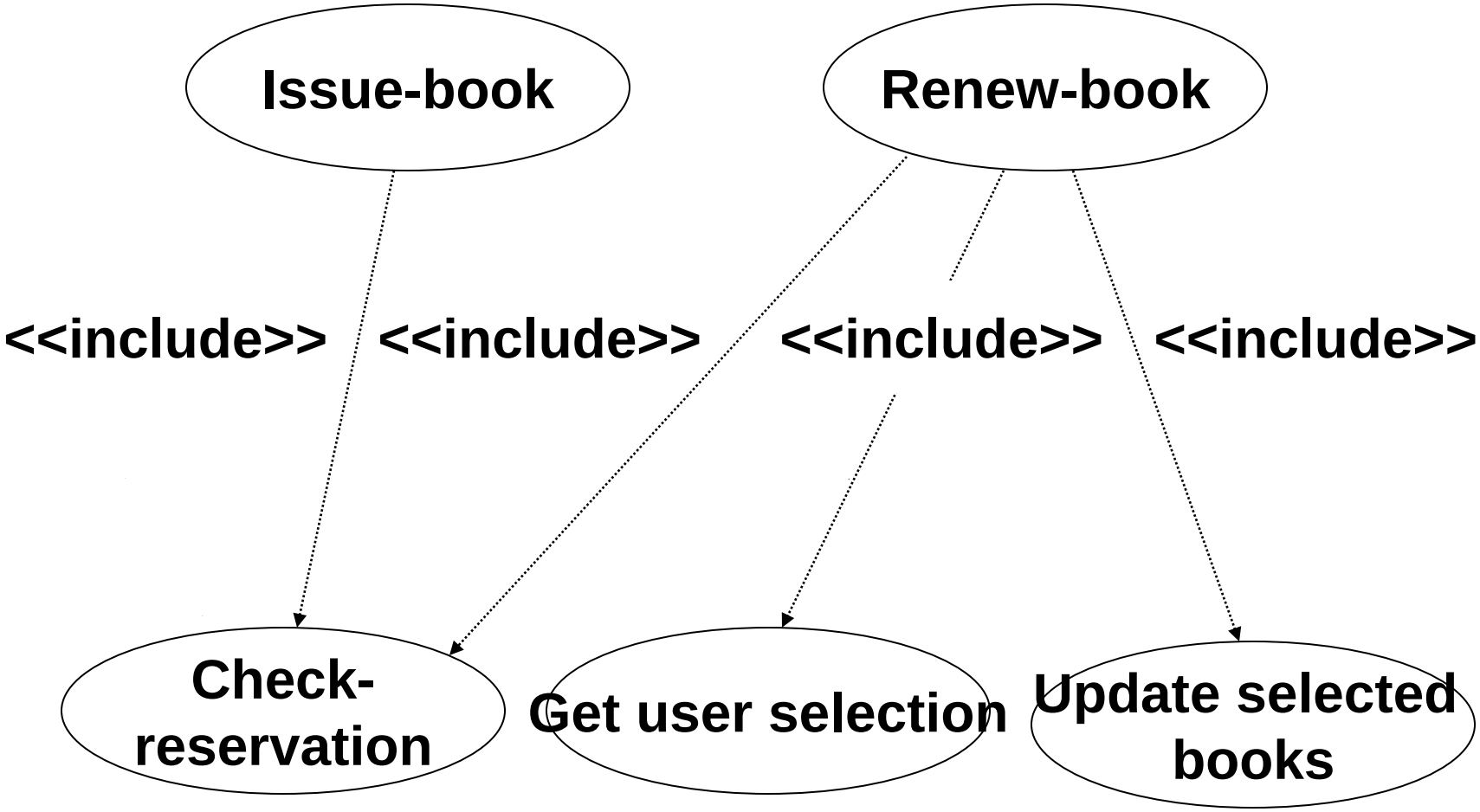
It is required to develop a software to manage the collection of books by individuals. A person can have a few hundreds of books. The details of all the books such as name of the book, year of publication, date of purchase, price, and publisher must be entered. A book is to be given a unique serial number by the computer which written by pen on the book. Before a friend can be lended a book, he must be registered. The registration data would include name of the friend, address, land line number, and mobile number. Before a friend is issued a book, the various books outstanding against him also with the date borrowed are displayed. The date of issue and the title of the book are stored. When a friend returns a book, the date of return is stored and the book is removed from his borrowing list. Up on query, the software should display the name, address, and telephone numbers of each friend against whom books are outstanding along with the titles of the outstanding books and the date issued.

Personal Library System: cont…

The owner of the library software, when he borrows books from his friends, enters the details regarding the title of books borrowed, and the date borrowed. It should be able display all the books borrowed from various friends.

It should be query about the availability a particular book, the total number of books in the personal library, and the total capital invested in the library.

Factoring Use Cases Using Includes cont...

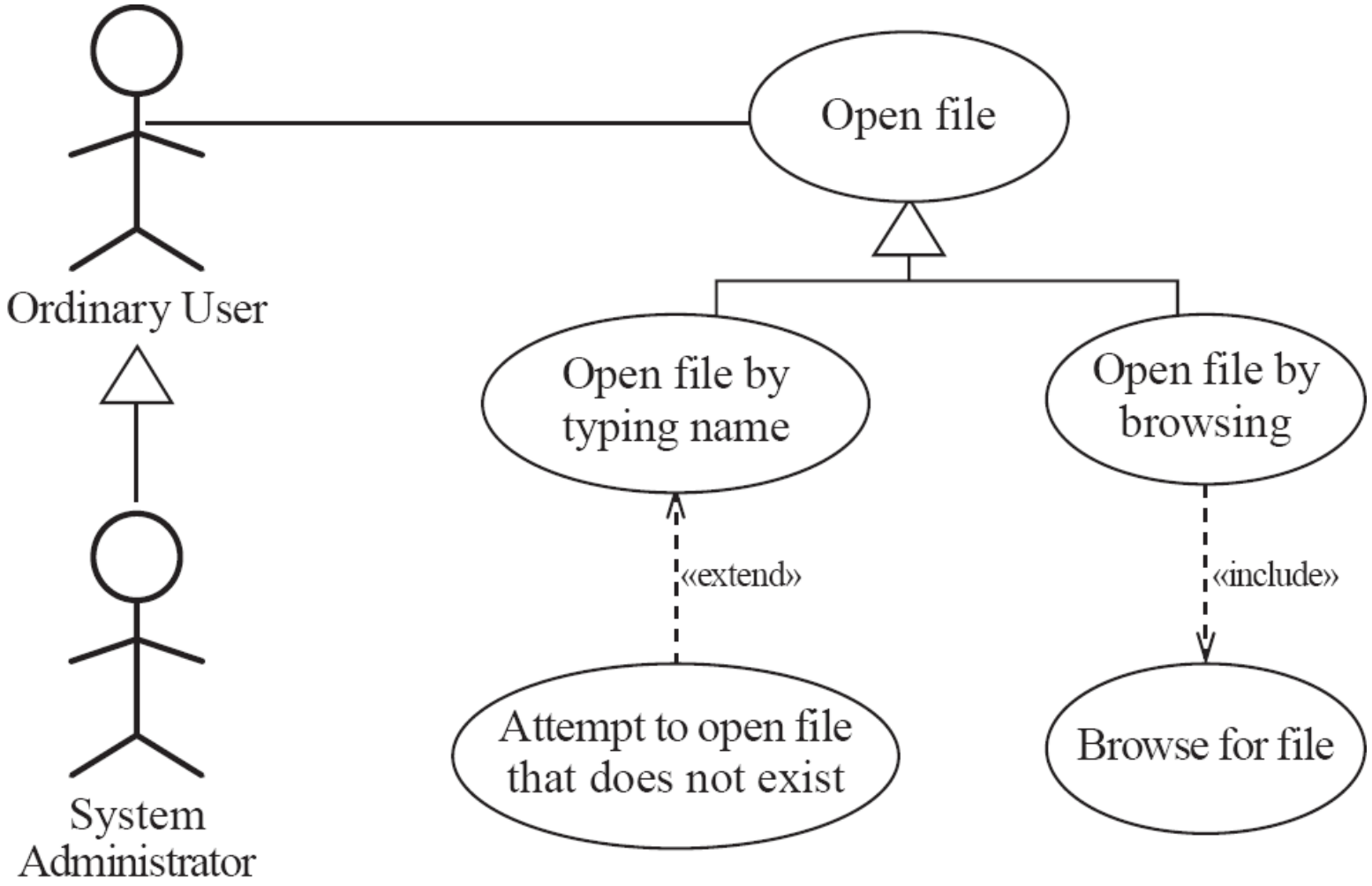


Extends

- It allows to show optional system behavior.
- An optional system behavior is executed only under certain conditions.
- Unlike generalization, the extend use case can add additional behavior only at an extension point when certain conditions specified.

Factoring Use Cases Using Extends

Example of generalization, extension and inclusion



Example description of a use case

Use case: **Open file**

Related use cases:
Generalization of:
- Open file by typing name
- Open file by browsing

Steps:

Actor actions	System responses
1. Choose ‘Open…’ command	2. File open dialog appears
3. Specify filename	
4. Confirm selection	5. Dialog disappears

Example cont…

Use case: **Open file by typing name**

Related use cases:
Specialization of: Open file

Steps:

Actor actions	System responses
1. Choose ‘Open…’ command	2. File open dialog appears
3a. Select text field	
3b. Type file name	
4. Click ‘Open’	5. Dialog disappears

Example cont…

Use case: **Open file by browsing**

Related use cases:
Specialization of: Open file
Includes: Browse for file

Steps:

Actor actions	System responses
1. Choose ‘Open…’ command	2. File open dialog appears
3. Browse for file	
4. Confirm selection	5. Dialog disappears

Example cont…

Use case: **Attempt to open file that does not exist**

Related use cases:
Extension of: Open file by typing name

Actor actions	**System responses**
1. Choose ‘Open…’ command	2. File open dialog appears
3a. Select text field	
3b. Type file name	
4. Click ‘Open’	5. System indicates that file does not exist
6. Correct the file name	
7. Click ‘Open’	8 Dialog disappears

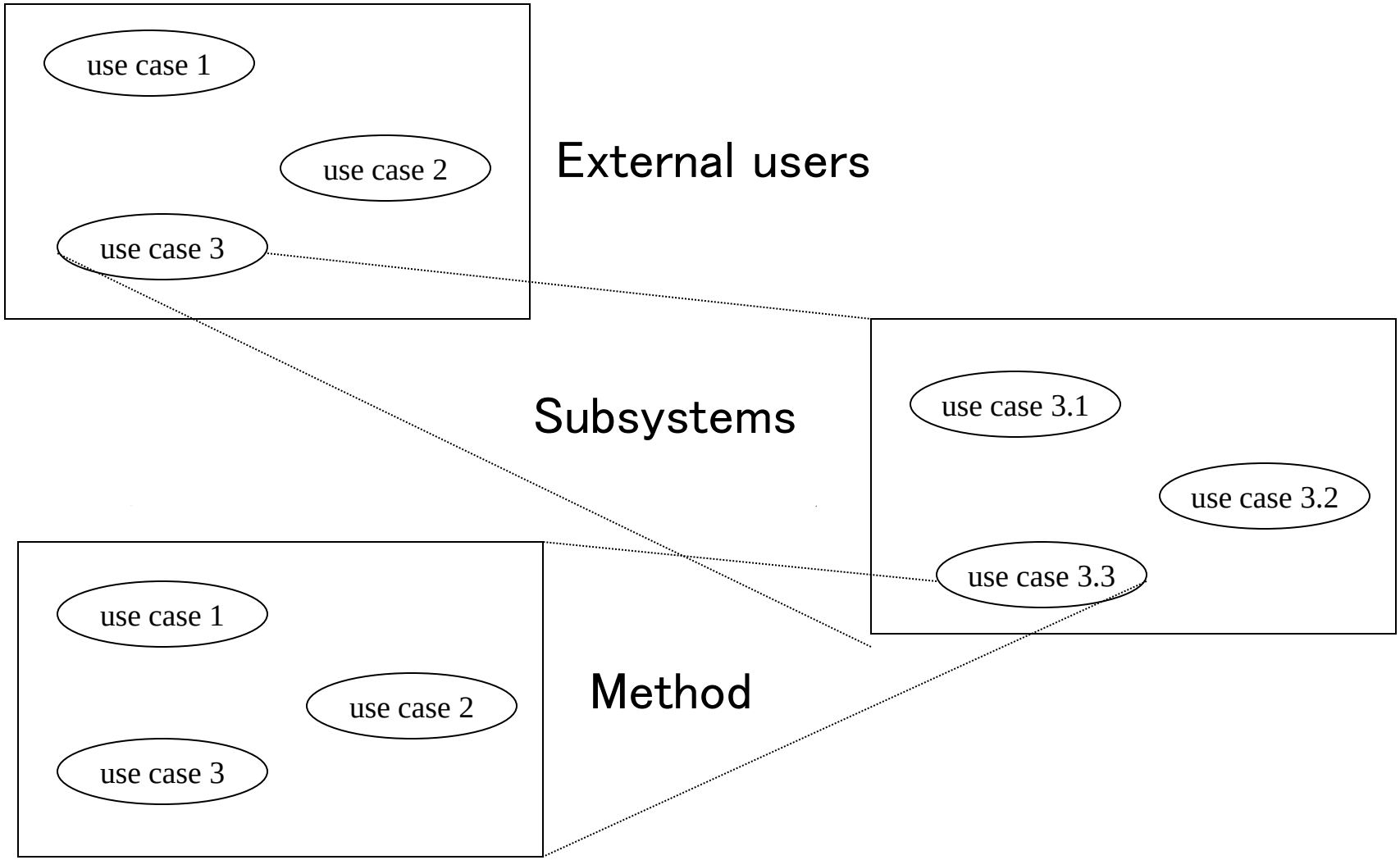
Example cont…

Use case: **Browse for file (inclusion)**

Steps:

Actor actions	System responses
1. If the desired file is not displayed, select a directory	2. Contents of directory is displayed
3. Repeat step 1 until the desired file is displayed	
4. Select a file	

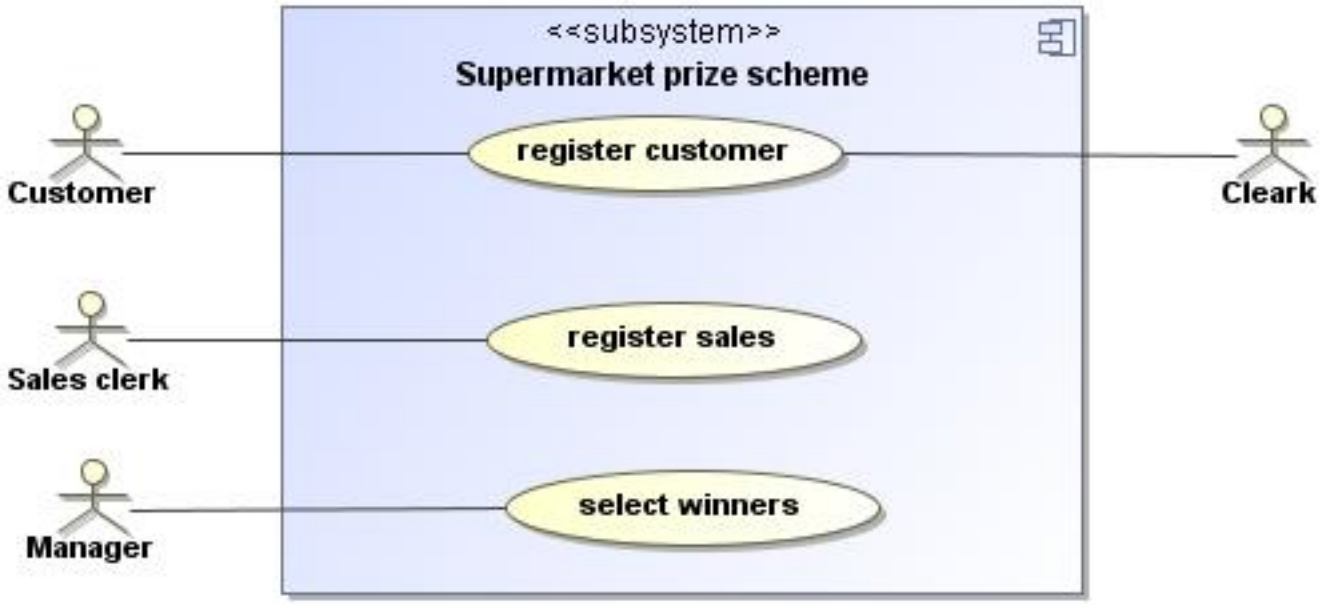
Hierarchical Organization of Use Cases



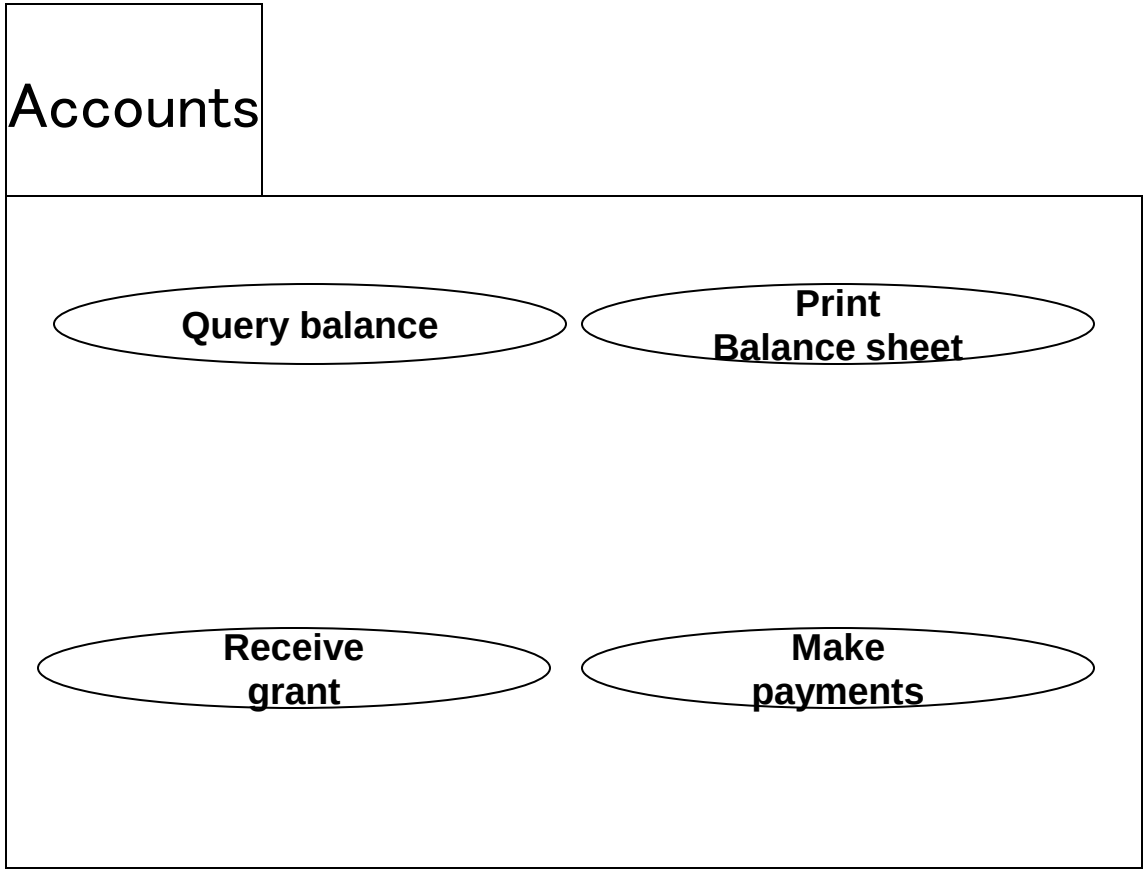
Supermarket Software

A supermarket needs to develop the following software to encourage regular customers. For this, the customer needs to supply his residence address, telephone number, and the driving license number. Each customers who registers for this scheme is assigned a unique customer number(CN) by the computer. A customer can present his CN to the check-out staff when he makes any purchase. In this case, the value of his purchase is credited against his CN. At the end of the year, the supermarket awards surprise gifts to 10 customers who make the highest total purchases over the year. Also, it awards a 22 carat gold coin to every customer whose purchases exceeds Rs. 10,000. The entries against the CN are reset on the last day of every year after the prize winners' lists are generated.

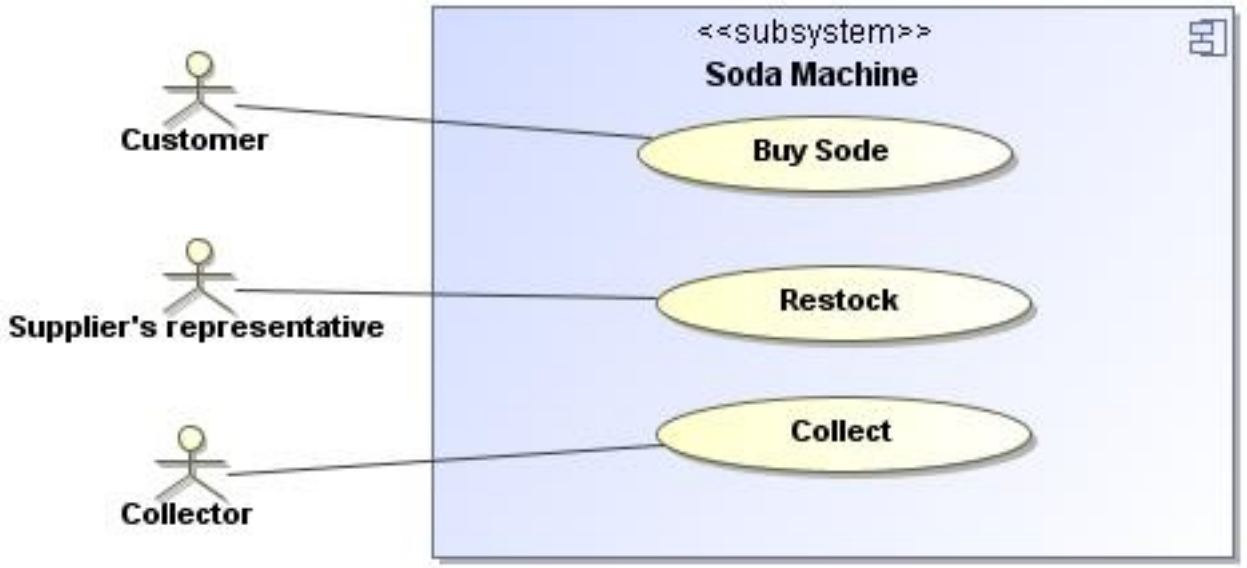
Supermarket Software cont...



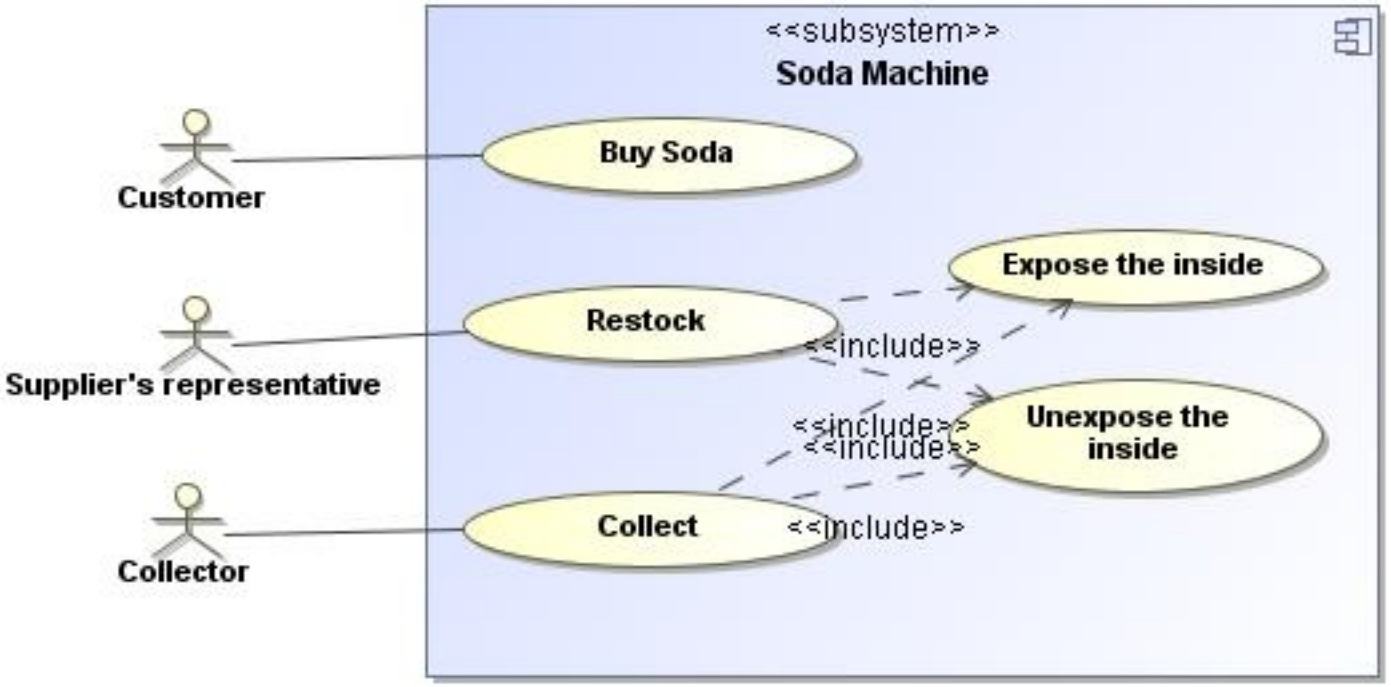
Use Case Packaging



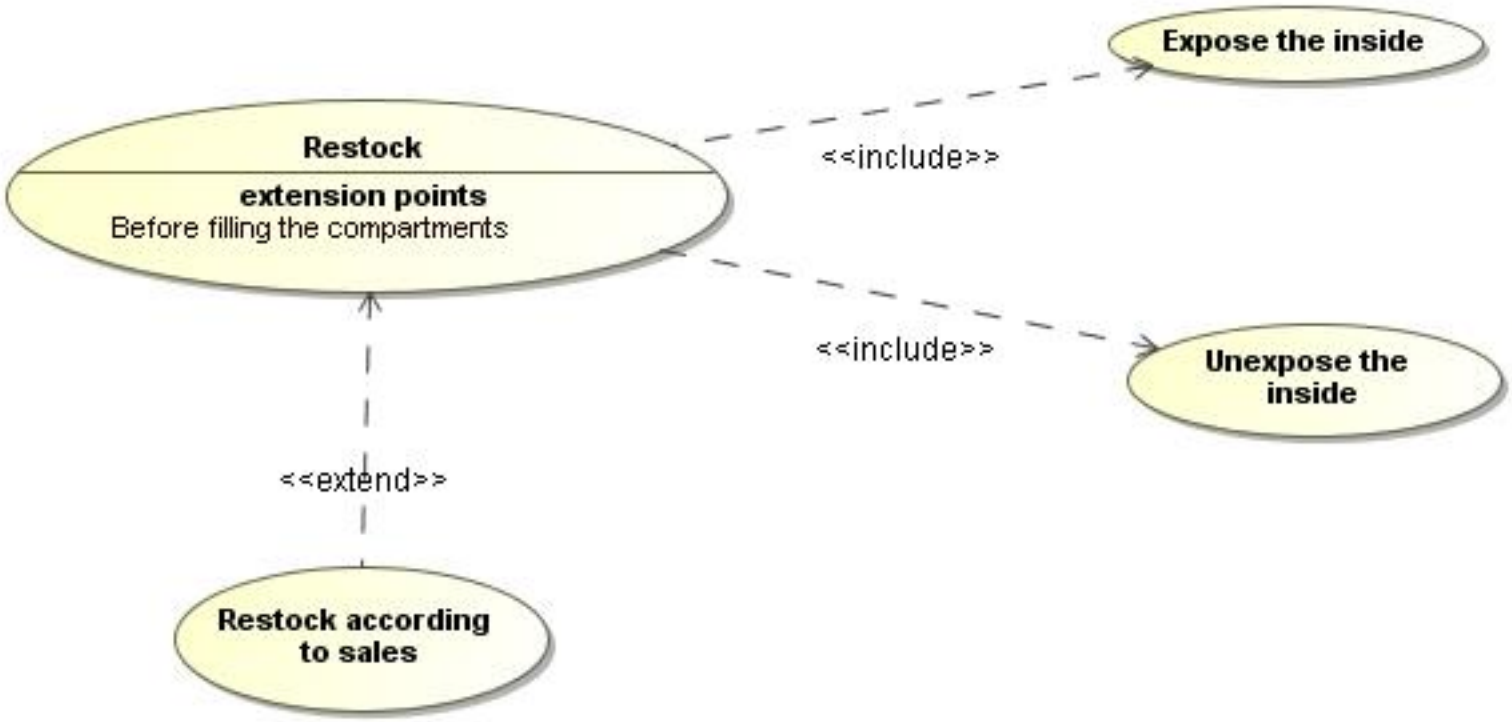
Soda Machine



Soda Machine cont…



Soda Machine cont…



The modeling processes: Choosing use cases on which to focus

- Often one use case (or a very small number) can be identified as *central* to the system
  - The entire system can be built around this particular use case
- There are other reasons for focusing on particular use cases:
  - Some use cases will represent a high *risk* because for some reason their implementation is problematic
  - Some use cases will have high political or commercial value

The benefits of basing software development on use cases

- They can
  - Help to define the *scope* of the system
  - Be used to *plan* the development process
  - Be used to both develop and validate the requirements
  - Form the basis for the definition of test cases
  - Be used to structure user manuals

Use cases must not be seen as a panacea

- The use cases themselves must be validated
  - Using the requirements validation methods.
- Some aspects of software are not covered by use case analysis.
- Innovative solutions may not be considered.

Thank you

Questions?