Advanced Software Engineering (CS6401)

Autumn Semester (2024-2025)

Dr. Judhistir Mahapatro
Department of Computer Science and Engineering
National Institute of Technology Rourkela

1

# Object-Oriented Software Design

Object-oriented Concepts

- Object-oriented (OO) design techniques are becoming popular:

  - Inception in early 1980's and nearing maturity.

  - Widespread acceptance in industry and academics

  - Unified Modelling Language (UML) poised to become a standard for modelling OO systems

Object-oriented Concepts

- Basic Mechanisms:
  - Objects:
    - A real-world entity.
    - A system is designed as a set of interacting objects.
    - Consists of data (attributes) and functions (methods) that operate on data
    - Hides organization of internal information (Data abstraction)
    - Examples: an employee, a book etc.

Object-oriented Concepts
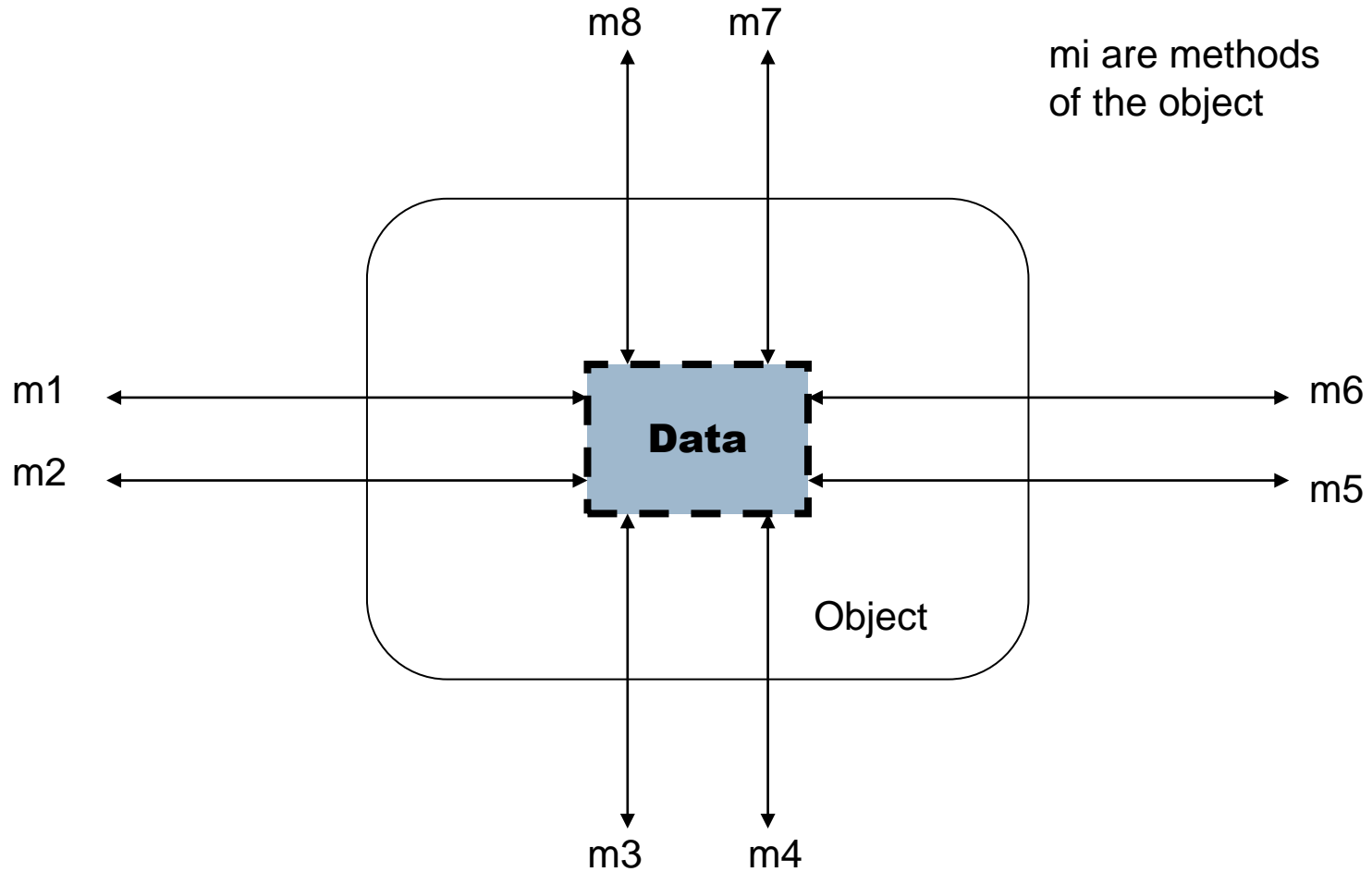
Model of an object

Object-oriented Concepts

- Class:
  - Instances are objects
  - Template for object creation
  - Sometimes not intended to produce instances (abstract classes)
  - Considered as abstract data type (ADT)
  - Examples: set of all employees, different types of book

Object-oriented Concepts

- Methods and message:
  - Operations supported by an object
  - Means for manipulating the data of other objects
  - Invoked by sending message
 - Examples: calculate_salary, issue-book, member_details, etc.

Object-oriented Concepts

- Inheritance:
    - Allows to define a new class (derived class) by extending or modifying existing class (base class)
    - Represents Generalization-specialization relationship
    - Allows redefinition of the existing methods (method overriding)

Object-oriented Concepts

- Multiple Inheritance:
  - Subclass can inherit attributes and methods from more than one base class

  - Multiple inheritance is represented by arrows drawn from the subclass to each of the base classes

Object-oriented Concepts
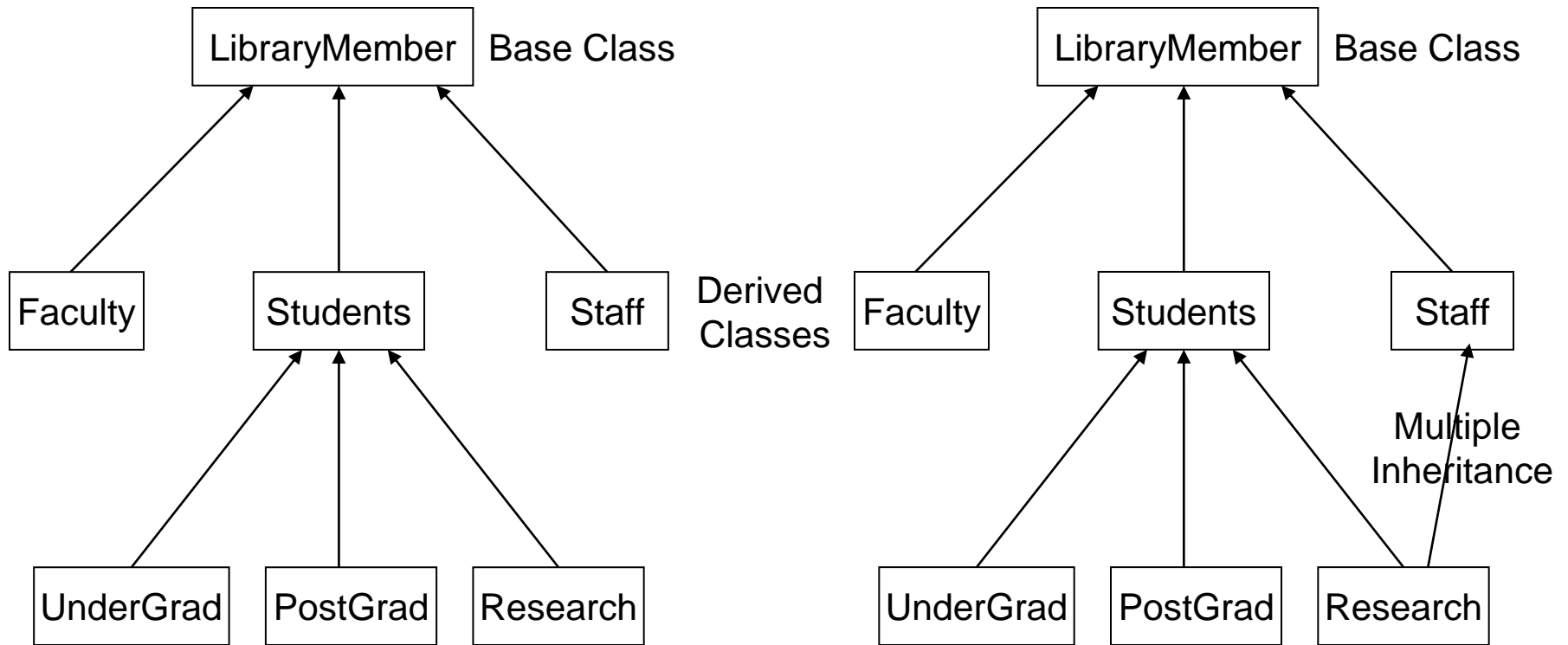
Object-oriented Concepts

- Key Concepts:
  - Abstraction:
    - Consider aspects relevant for certain purpose
    - Suppress non-relevant aspects
    - Supported at two levels i.e. class level where base class is an abstraction & object level where object is a data abstraction entity
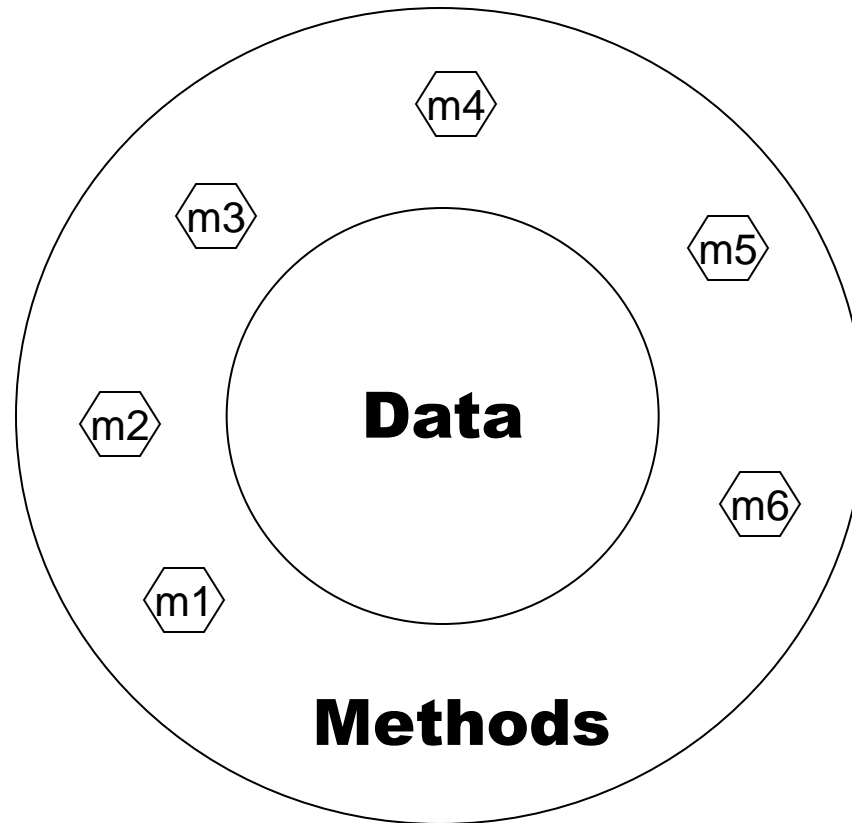
Object-oriented Concepts

- Advantages of abstraction:
    - Reduces complexity of software
    - Increases software productivity
- It is shown that software productivity is inversely proportional to software complexity

Object-oriented Concepts

- Encapsulation:
  - Objects communicate outside world through messages
  - Objects data encapsulated within its methods

Object-oriented Concepts

Concept of encapsulation

Object-oriented Concepts

- Polymorphism:
  - Denotes to poly (many) morphism (forms)
  - Same message result in different actions by different objects (static binding)

Object-oriented Concepts

- Example of static binding:

  Class Circle{

      private float x, y, radius;

      private int fillType;

      public create ();

      public create (float x, float y, float centre);

      public create (float x, float y, float centre, int fillType);

         }

Object-oriented Concepts

- In this example:
  - A class named Circle has three definitions for create operation

  - Without any parameter, default

  - Centre and radius as parameter

Object-oriented Concepts

- In this example:
  - Centre, radius and filltype as parameter

  - Depending upon parameters given, method will be invoked

  - Method create is overloaded

Object-oriented Concepts

- Dynamic binding:
  - In inheritance hierarchy, an object can be assigned to another object of its ancestor class

  - A method call to an ancestor object would result in the invocation of appropriate method of object of the derived class
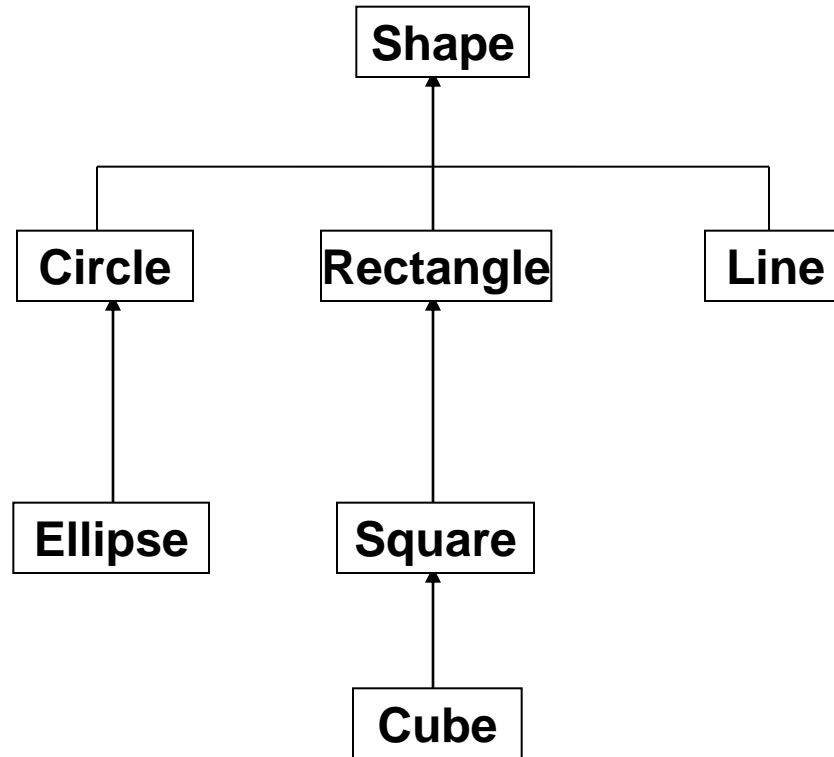
Object-oriented Concepts

- Dynamic binding:
    - Exact method cannot be known at compile time

    - Dynamically decided at runtime

Object-oriented Concepts

- Example of dynamic binding:
    - Consider class hierarchy of different geometric objects
    - Now display method is declared in the shape class and overridden in each derived class
    - A single call to the display method for each object would take care displaying appropriate element

Object-oriented Concepts

Class hierarchy of geometric objects

Object-oriented Concepts

Traditional code

If (shape == Circle) then
        draw_circle();
else if (shape == Rectangle) then
        draw_rectangle();

        -

        -

        -

        -

Object-oriented code

Shape.draw();

        -

        -

        -

        -

Traditional code and OO code using dynamic binding

Object-oriented Concepts

- Advantages of dynamic binding:
  - Leads to elegant programming
  - Facilitates code reuse and maintenance
  - New objects can be added with minimal changes to existing objects

Object-oriented Concepts

- **Composite objects:**
  - Object containing other objects

  - Composition limited to tree hierarchy i.e. no circular inclusion relation

  - Inheritance hierarchy different from containment hierarchy

Object-oriented Concepts

- **Composite objects:**
  - Inheritance hierarchy, different object types with similar features

  - Containment allows construction of complex objects

Object-oriented Concepts

- **Genericity:**
  - Ability to parameterize class definitions
  - Example: class stack of different types of elements such as integer, character and floating point stack
  - Generacity permits to define generic class stack and later instantiate as required

Advantages of Object-oriented design

- Code and design reuse

- Increased productivity

- Ease of testing & maintenance

- Better understandability

- Its agreed that increased productivity is chief advantage

Advantages of Object-oriented design

- Initially incur higher costs, but after completion of some projects reduction in cost become possible

- Well-established OO methodology and environment can be managed with 20-50% of traditional cost of development

Object modelling using UML

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

Unified Modelling Language (UML)

- Origin
  - In late 1980s and early 1990s different software development houses were using different notations
  - Developed in early 1990s to standardize the large number of object-oriented modelling notations

UML

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

UML
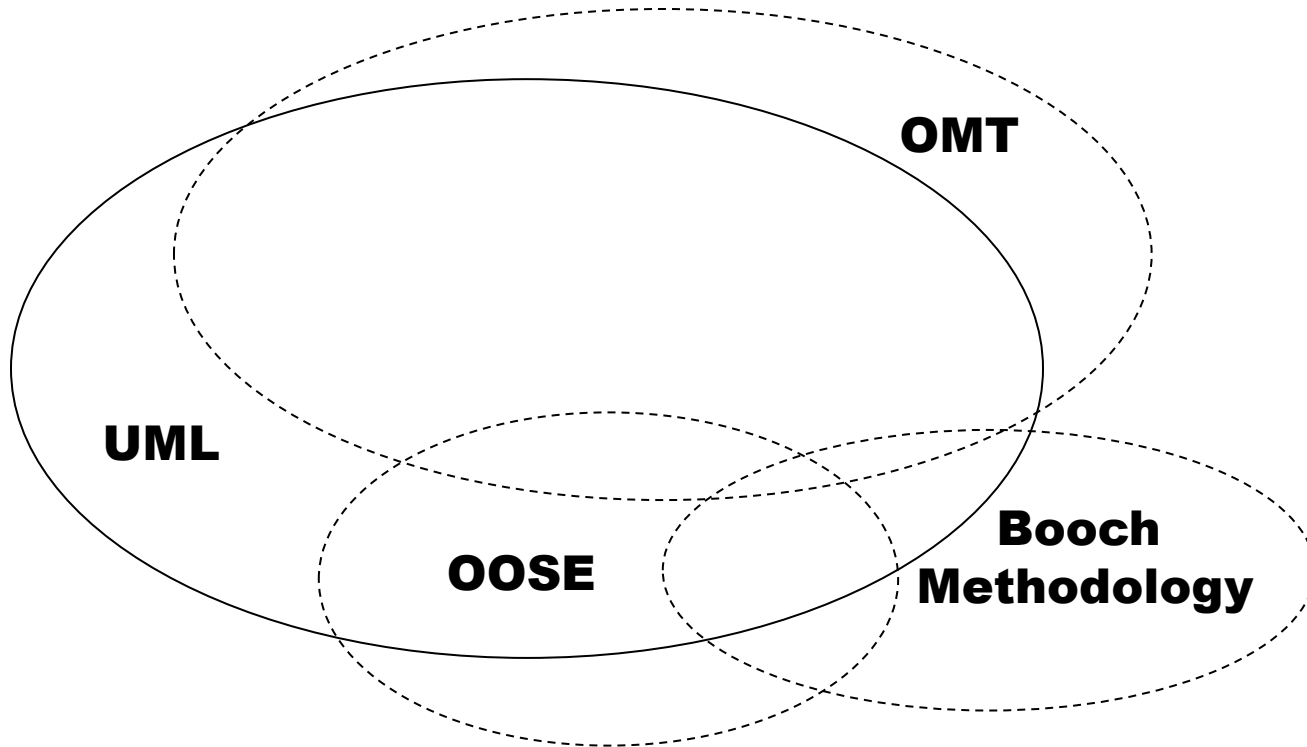
Different object modelling techniques in UML

UML

- As a Standard
  - Adopted by Object Management Group (OMG) in 1997
  - OMG an association of industries
  - Promote consensus notations and techniques
  - Used outside software development, example car manufacturing

Why UML is required?

- Model is required to capture only important aspects

- UML a graphical modelling tool, easy to understand and construct
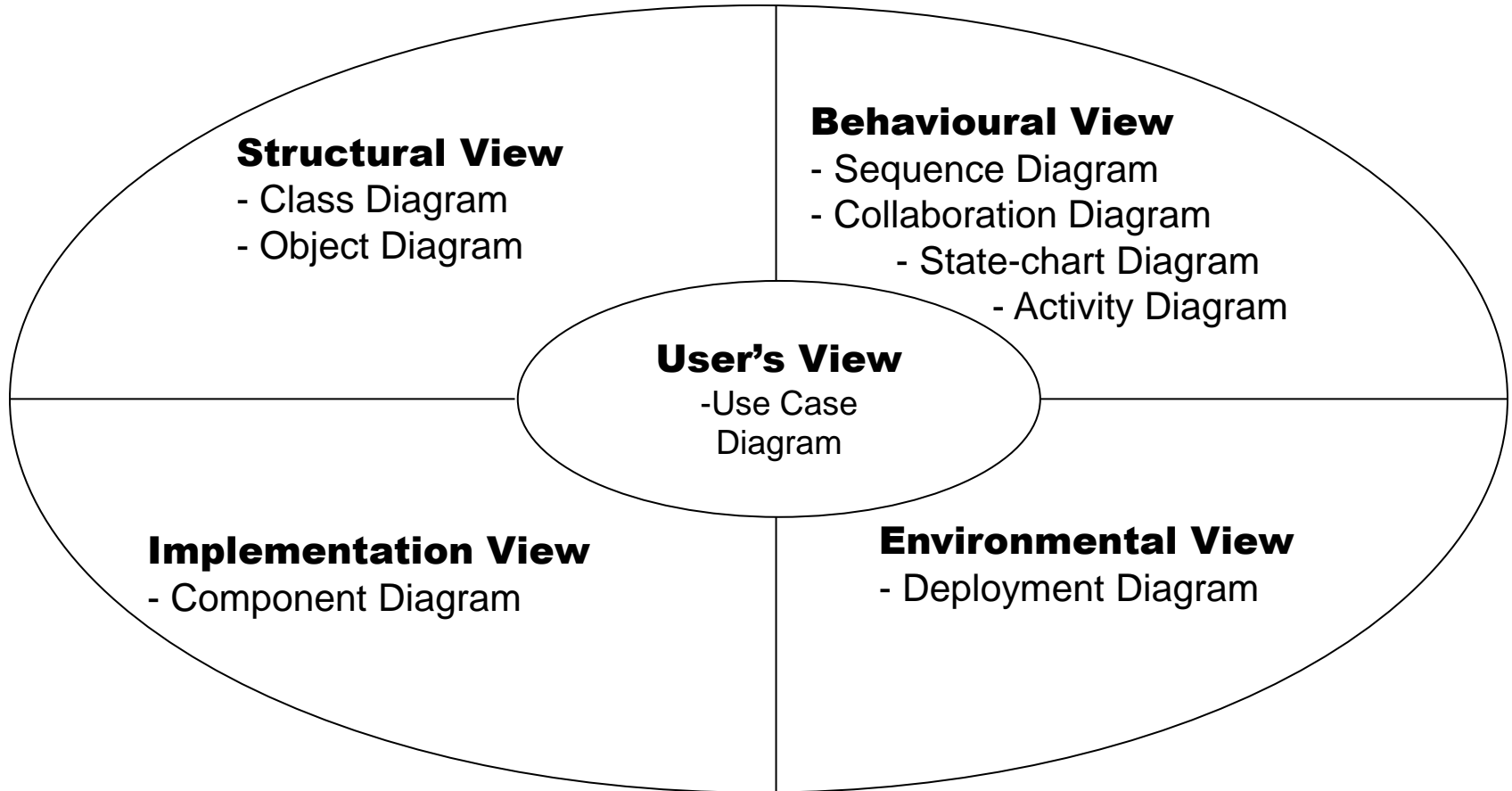
- Helps in managing complexity

UML diagrams

- Nine diagrams to capture different views of a system

- Provide different perspectives of the software system

- Diagrams can be refined to get the actual implementation of the system

UML diagrams

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

UML diagrams

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

Diagrams and views in UML

UML diagrams

- User's view
  - It captures the view of the system in terms of the functionalities offered by the system to its user
  - It is a black-box view of the system
  - Dynamic behavior of the components, the implementation etc are not captured.

UML diagrams

- Structural view
  - Defines the structure of the problem (or the solution) in terms of the objects (or classes) important to the understanding of the working of a system and its implementation.
  - It captures the relationship among the classes (or objects)
  - Called the static model, since the structure of a system does not change with time.

UML diagrams

- Behavioural view
  - Captures how objects interact with each other in time to realize the system behavior.
  - System behavior captures the time-dependent (dynamic) behavior.
  - It constitutes the dynamic model of the system

UML diagrams

- Implementation view
  - Captures important components of the system and their interdependencies.
  - Example, implementation view might show the GUI part, the middleware, and the database parts and would capture their interdependencies.

UML diagrams

- Environmental view
  - This view models how the different components are implemented on different pieces of hardware.

Are all views required?

- NO
- Use case model, class diagram and one of the interaction diagram for a simple system
- State chart diagram in case of many state changes
- Deployment diagram in case of large number of hardware components

Use Case model

- Consists of set of "use cases"

- An important analysis and design artifact

- Other models must confirm to this model

- Not really an object-oriented model

- Represents a functional or process model

Use Cases

- Different ways in which system can be used by the users

- Corresponds to the high-level requirements

- Represents transaction between the user and the system

- Define behavior without revealing internal structure of system

- Set of related scenarios tied together by a common goal

Use Cases

- 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.

Representation of Use Cases

- Represented by use case diagram

- Use case is represented by ellipse

- System boundary is represented by rectangle

- Users are represented by stick person icon (actor)

- Communication relationship between actor and use case by line

- External system by stereotype

Example of Use Cases

Player

Play Move

Tic-tac-toe game
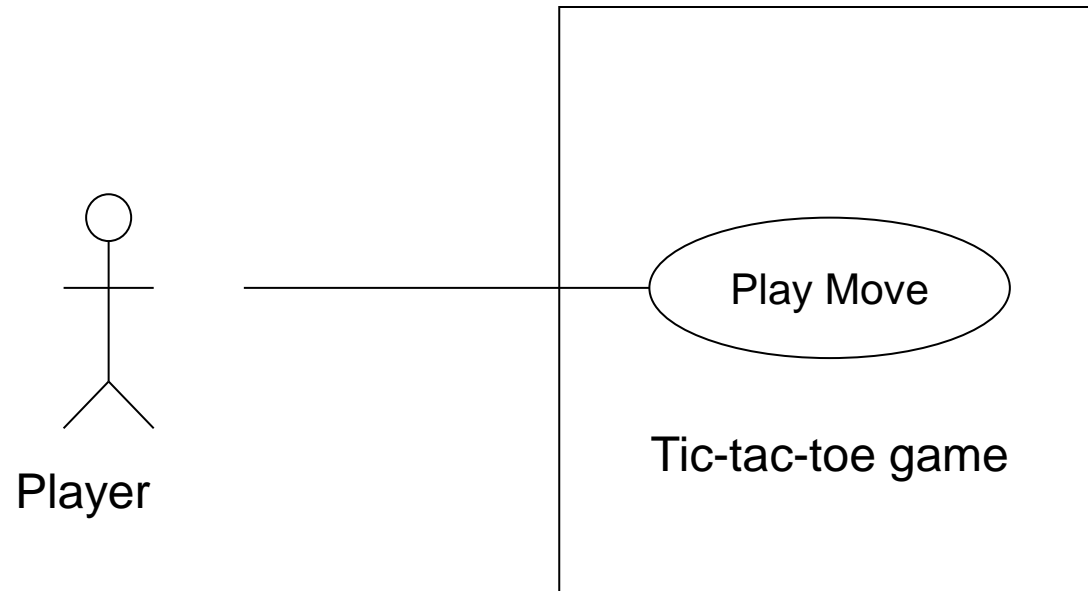
Use case model

Why develop Use Case diagram?

- Serves as requirements specification

- Users identification helps in implementing security mechanism through login system

- Another use in preparing the documents (e.g. user's manual)
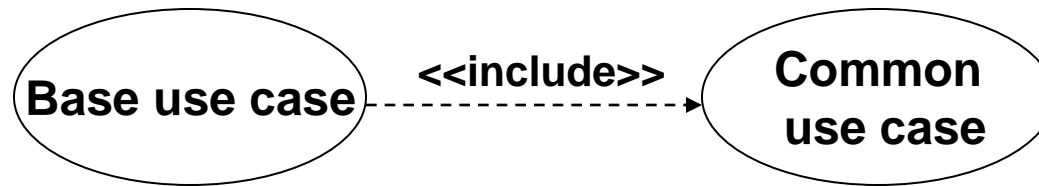
Factoring Use Cases

- Complex use cases need to be factored into simpler use cases

- Represent common behavior across different use cases

- Three ways of factoring
  - Generalization
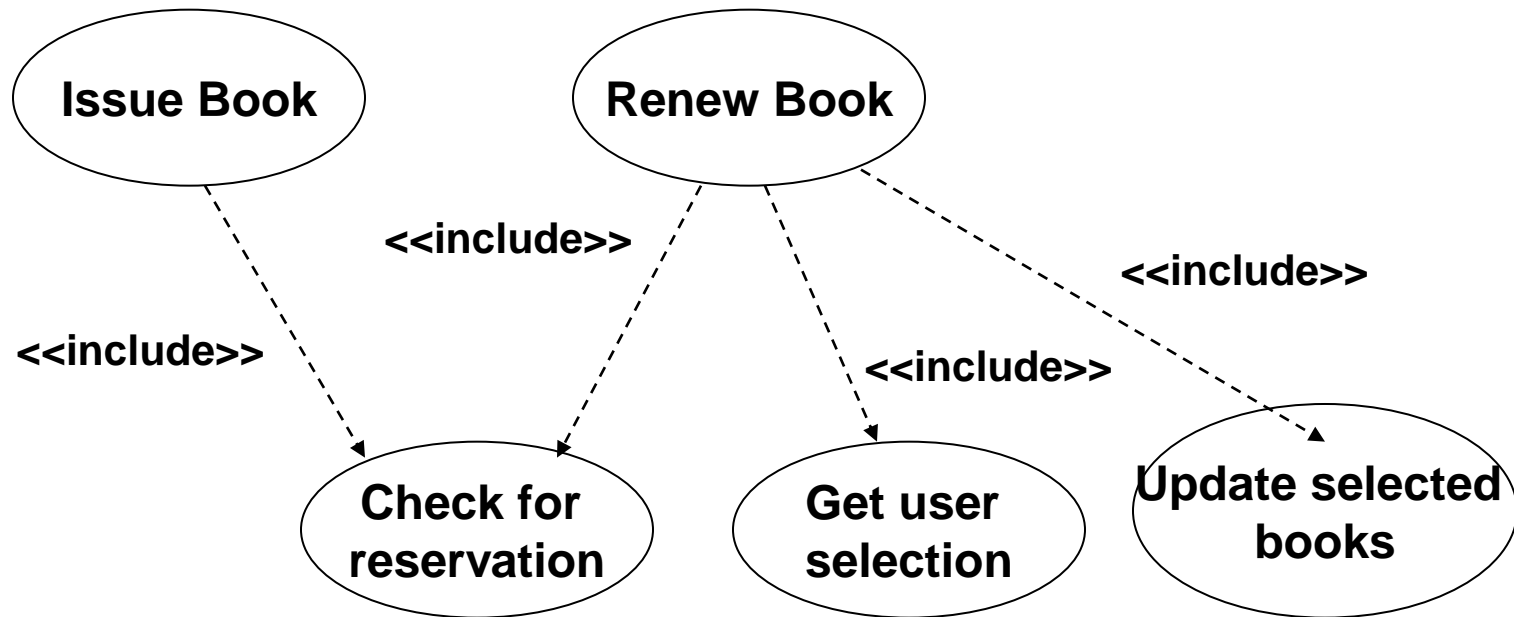 - Includes
  - Extends

Factoring Using Generalization
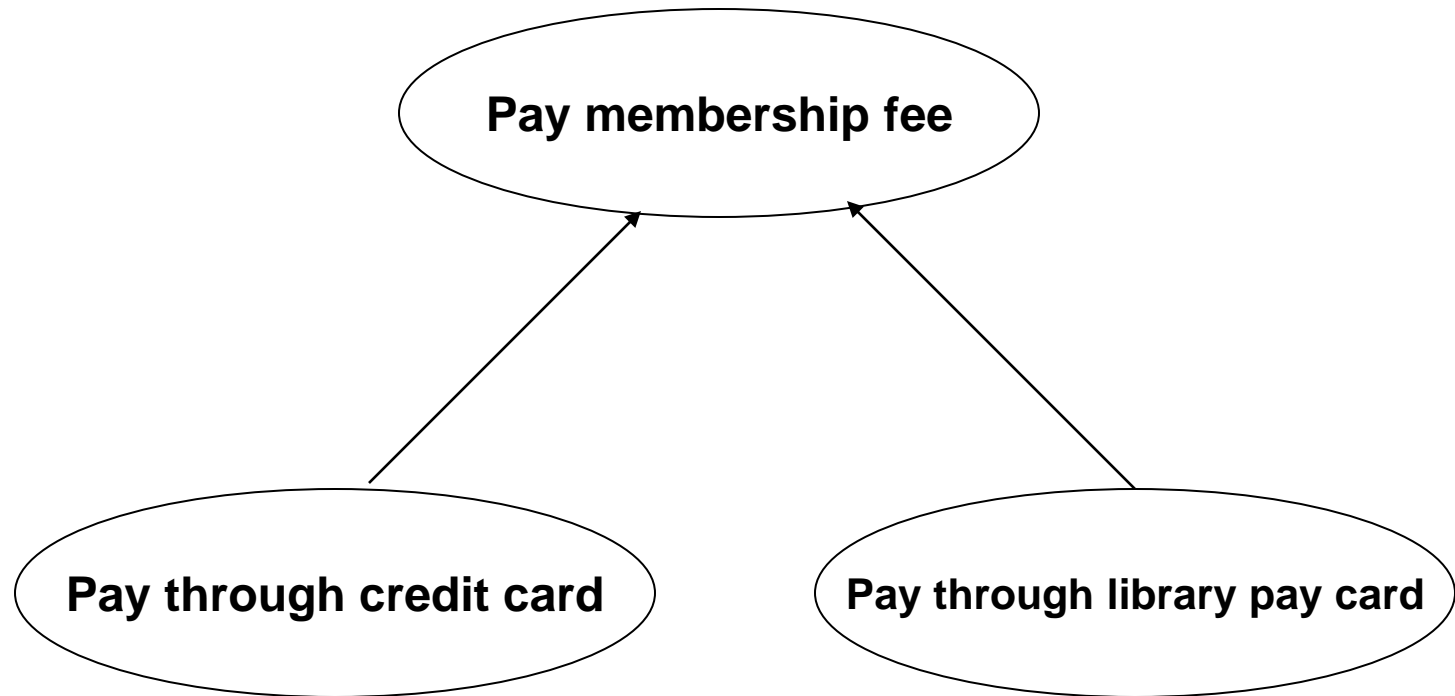
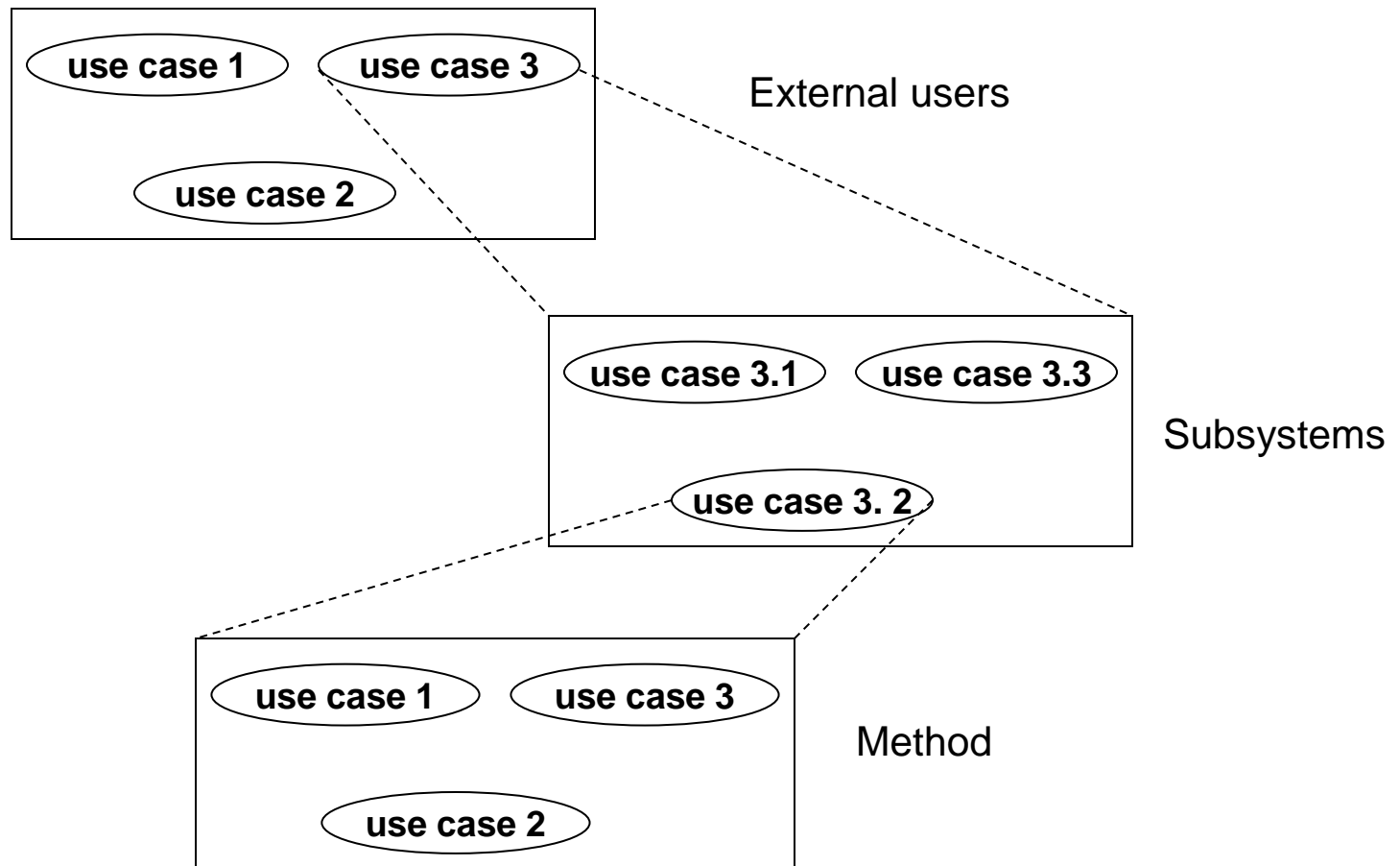**Use case generalization**

Factoring Using Includes

Use case inclusion

example

Factoring Using Extends

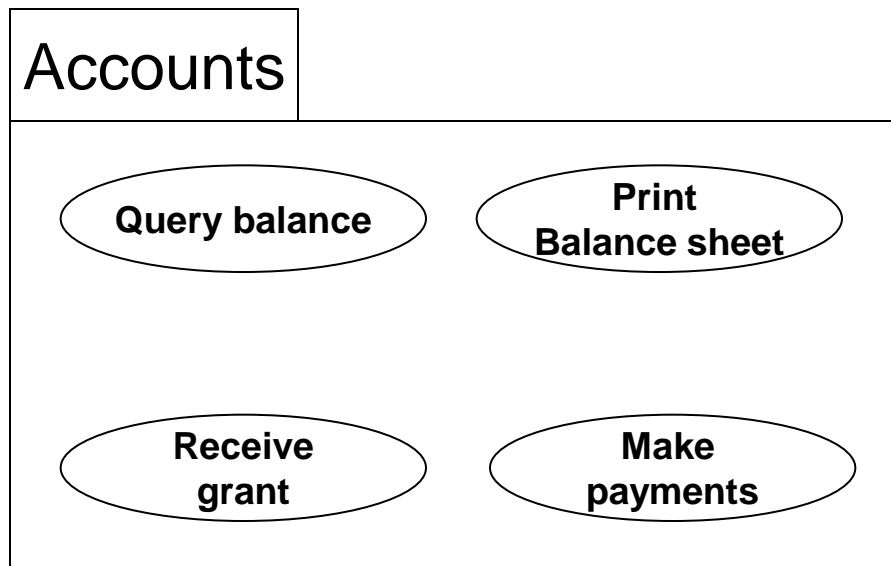**Use case extension**

Hierarchical Organization of Use Cases

Hierarchical organization of use cases

Use Case Packaging

Use case packaging

Reference

- R. Mall, *Fundamentals of Software Engineering*, Prentice Hall of India , 2014