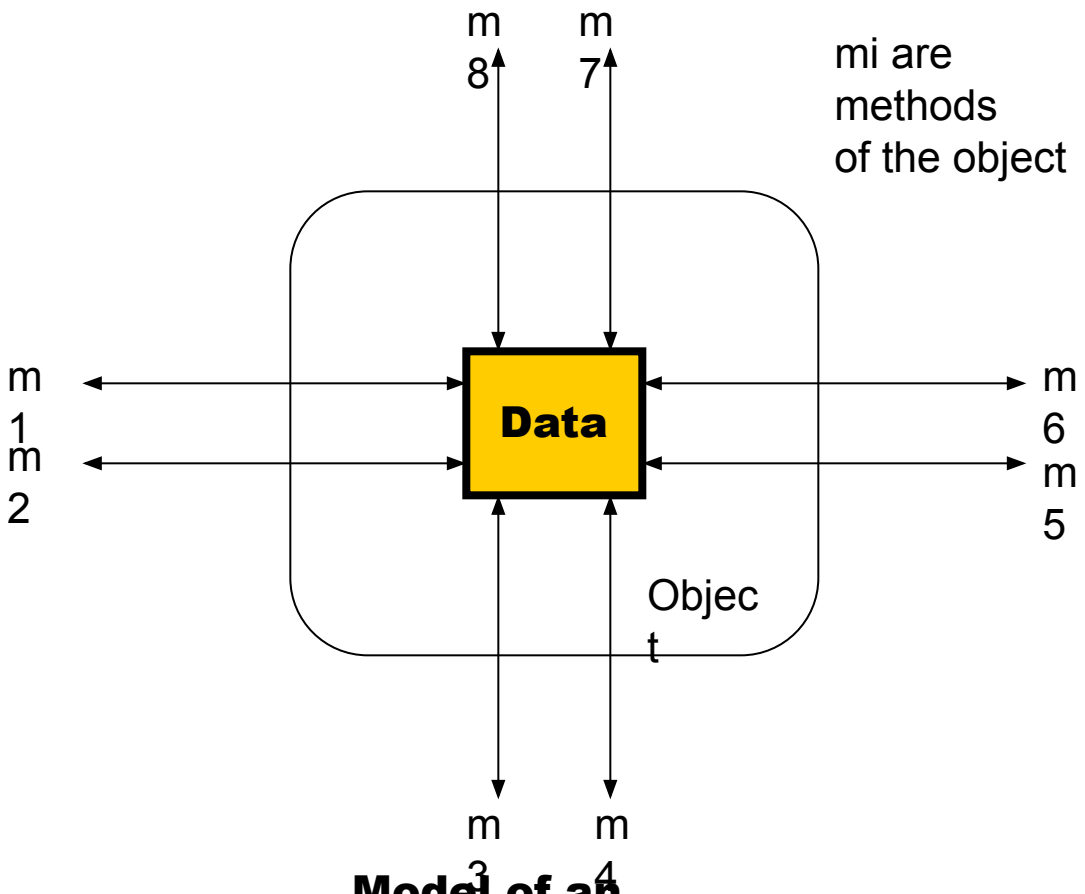
Object-Oriented Software Design (lecture 7)

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**
  - **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.**

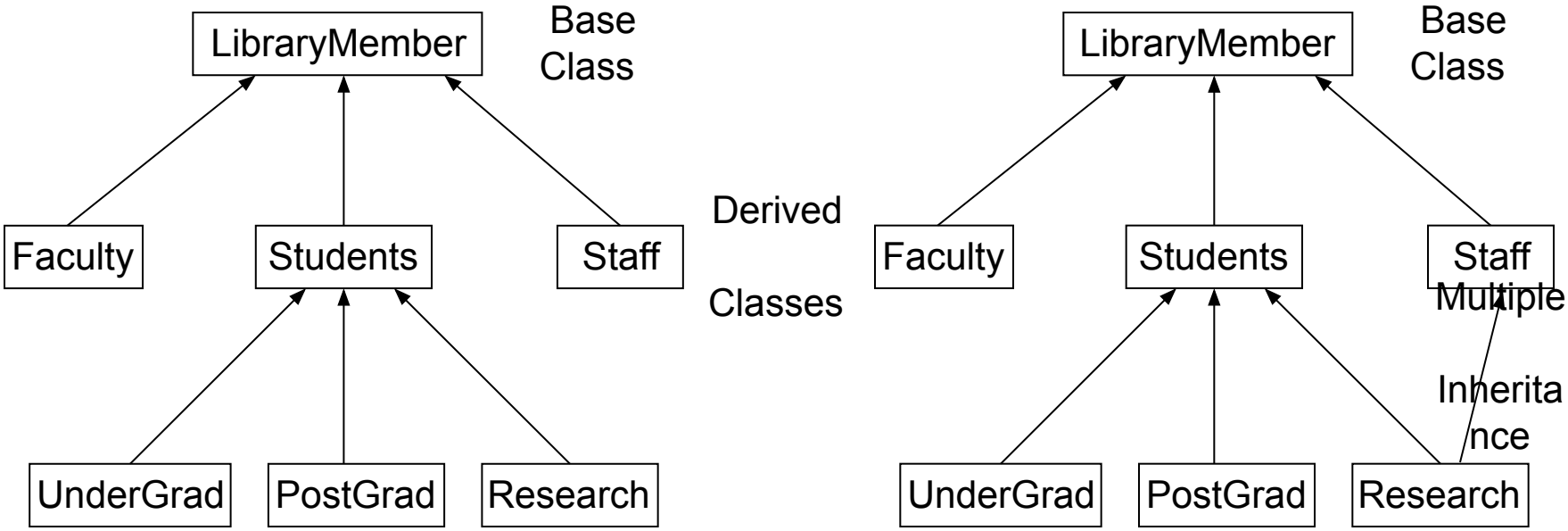
Object-oriented Concepts

- **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



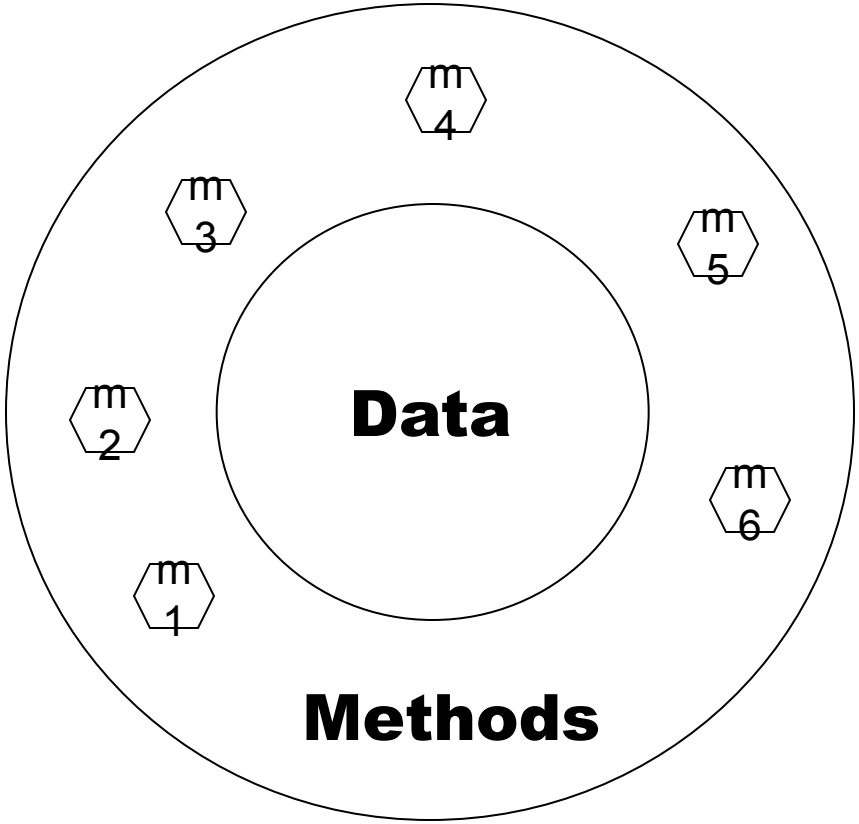
Abstraction

- Consider aspects relevant for certain purpose
- Suppress non-relevant aspects
- Supported at two-levels i.e. class level where base class is an abstraction and object level where object is a data abstraction entity.

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

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

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

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**

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

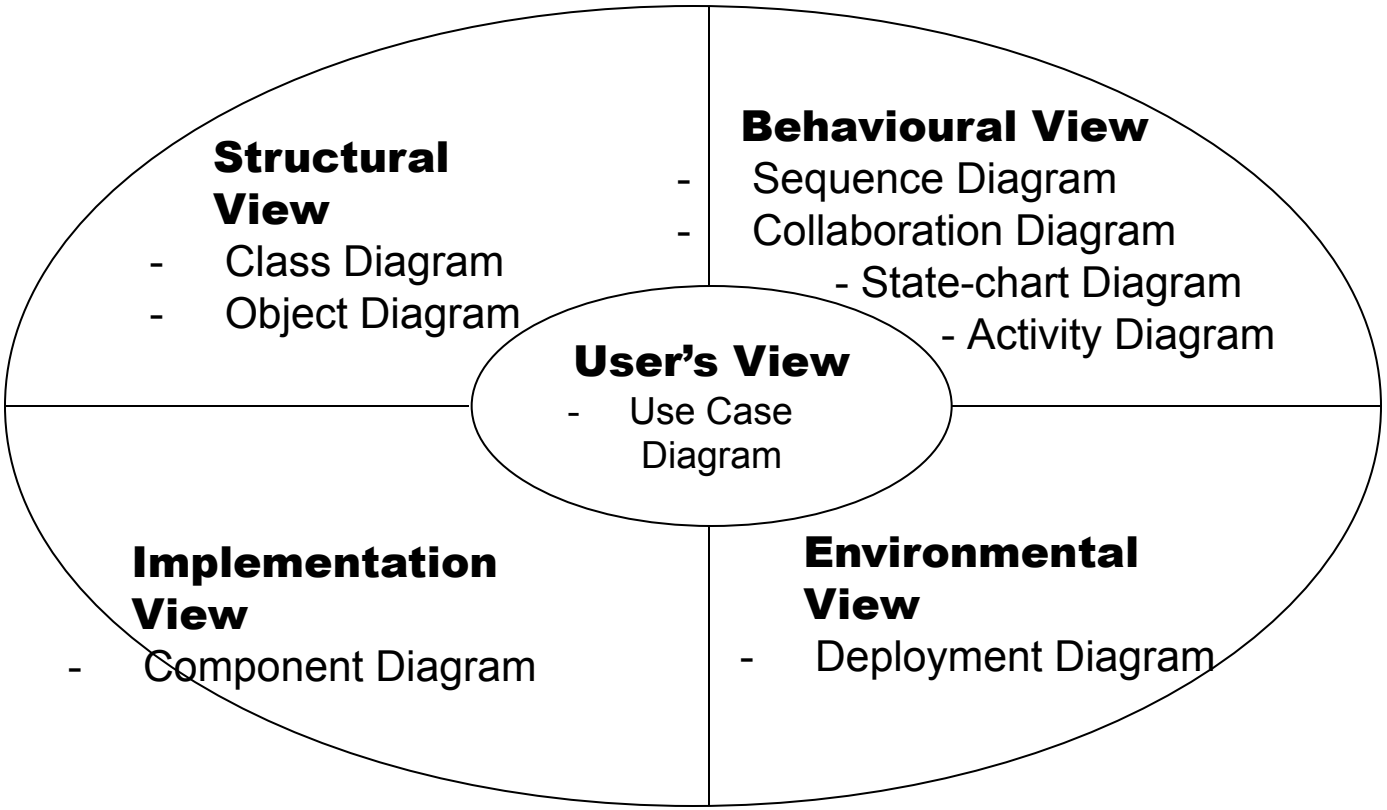
UML diagrams

- 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



Diagrams and views in UML

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.

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

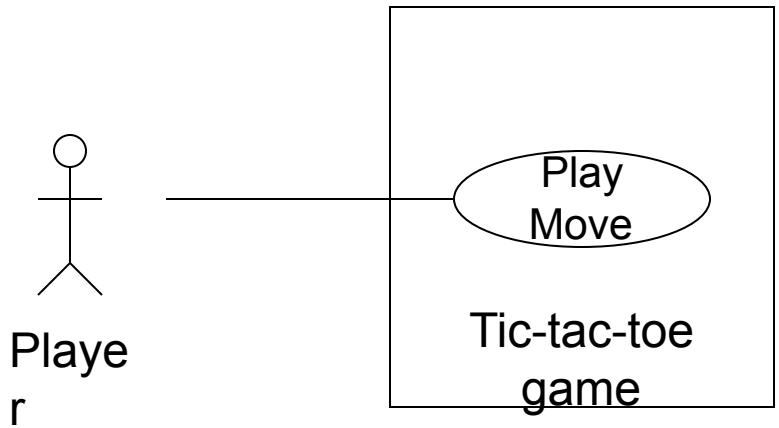
Example of Use Cases

- 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

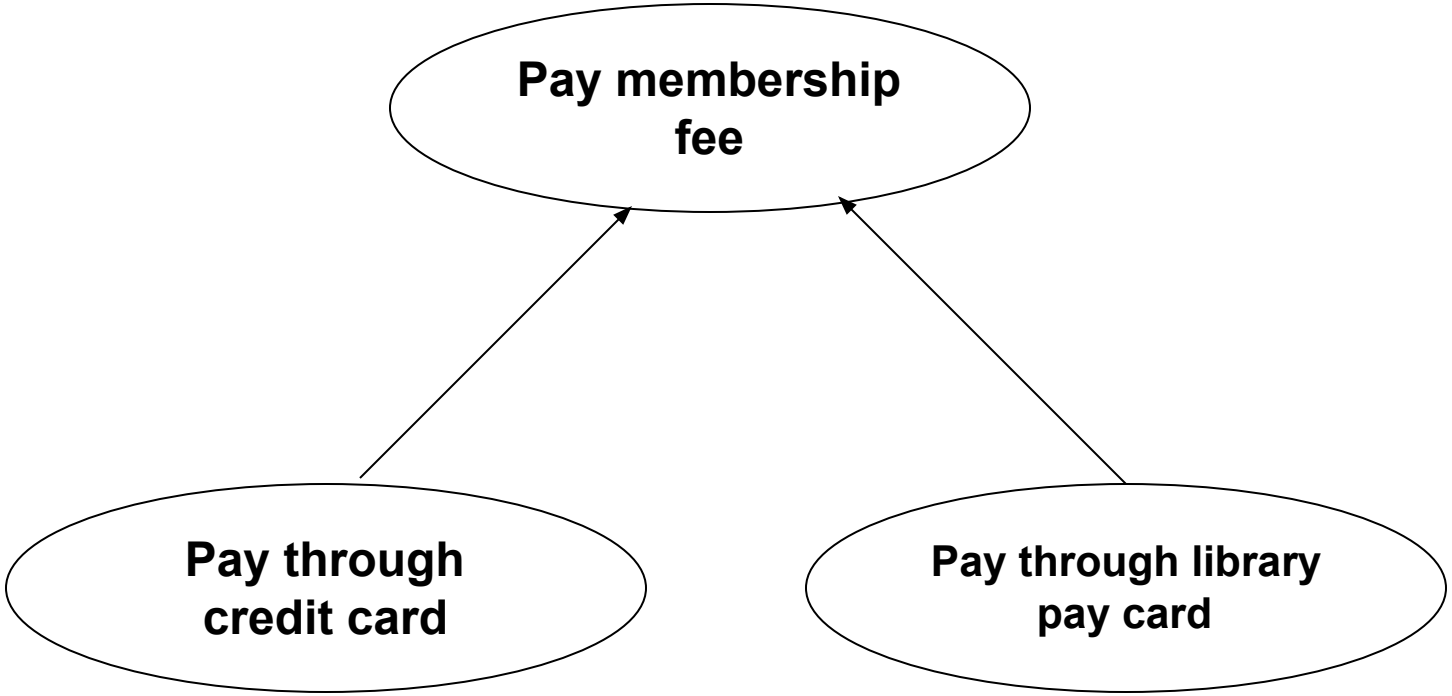


Use case model

Factoring Use Cases

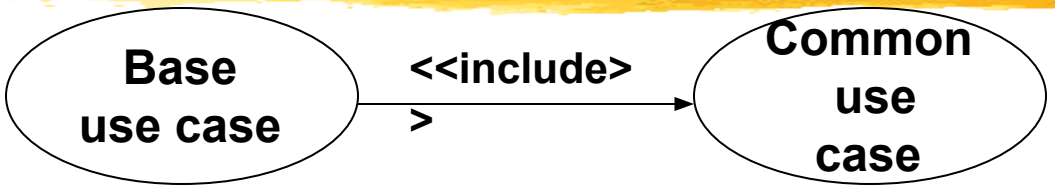
- Complex use cases need to be factored into simpler use cases
- Represent common behavior across different use cases
- Three ways of factoring
  - Generalization
  - Includes
  - Extends

Factoring Using Generalization

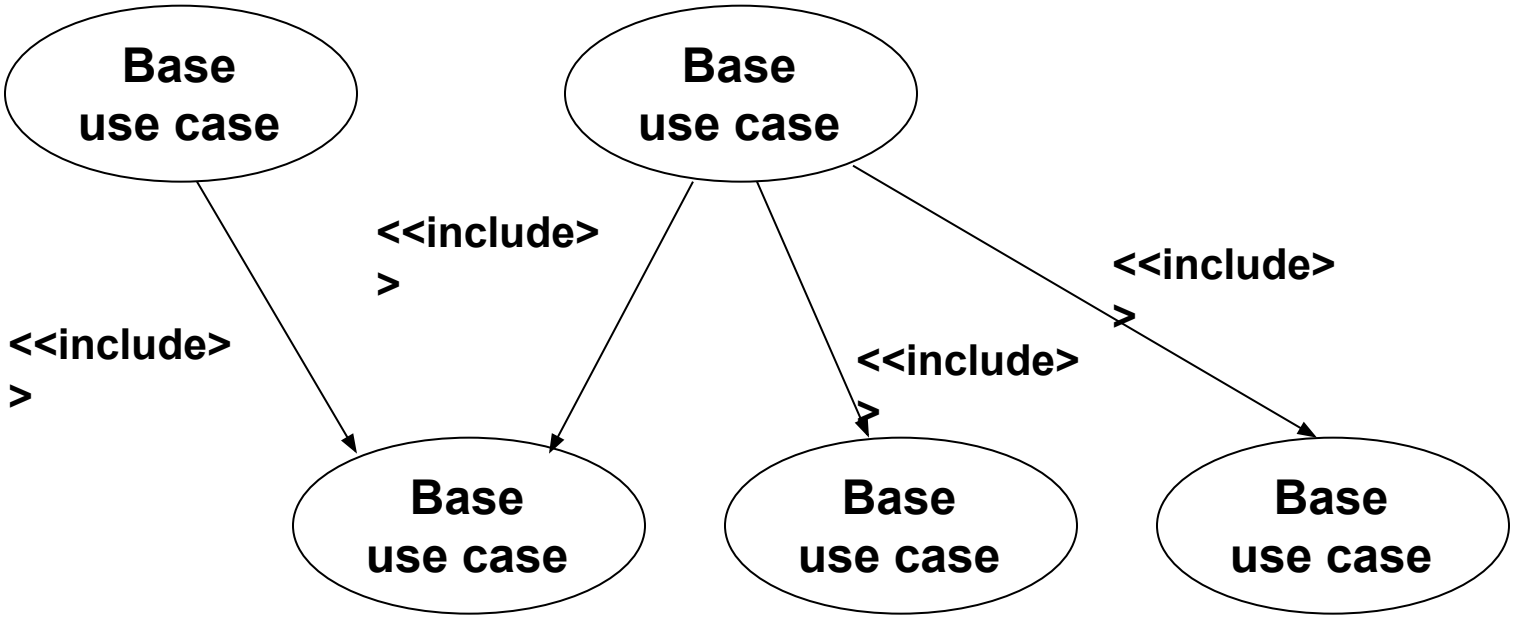


Use case generalization

Factoring Using Includes



Use case inclusion



Paralleling model

Factoring Using Extends

Use case extension

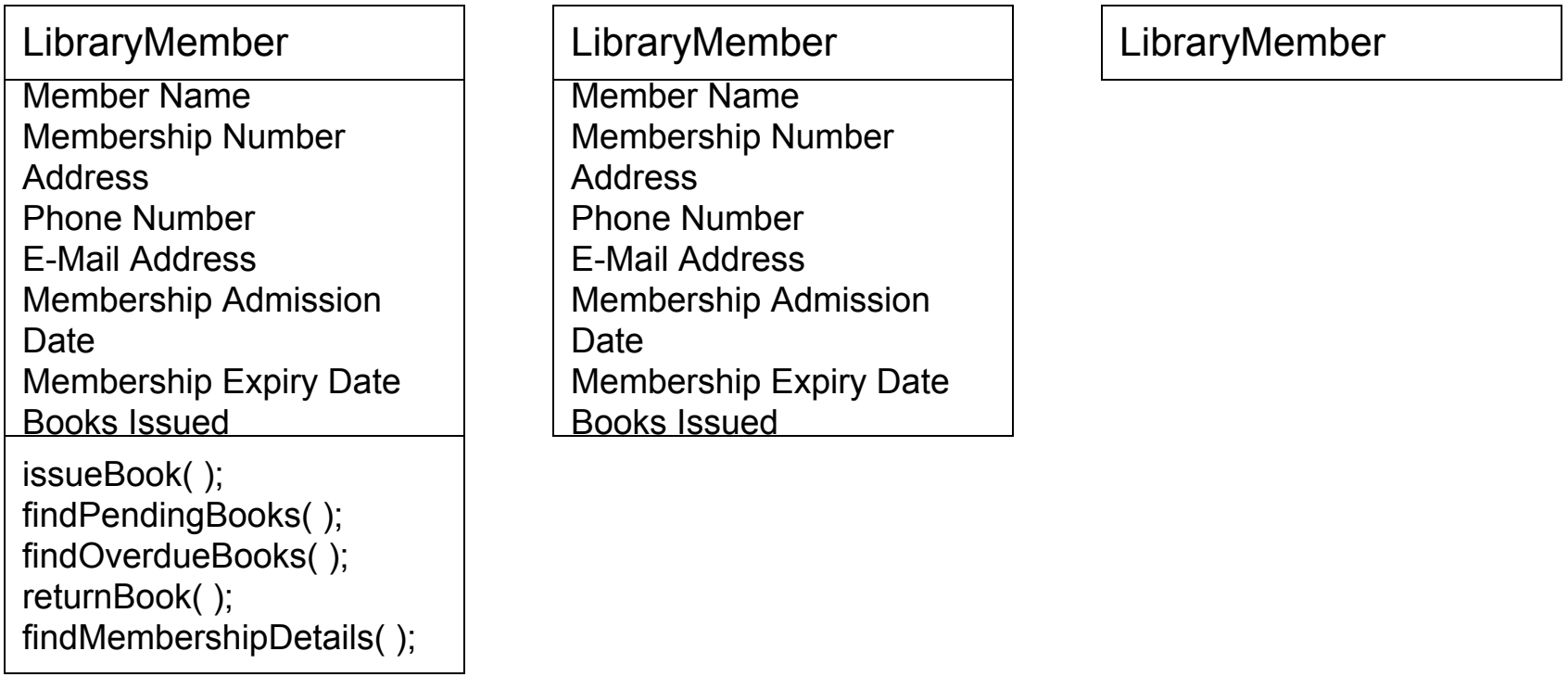
Class diagram

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

Class diagram

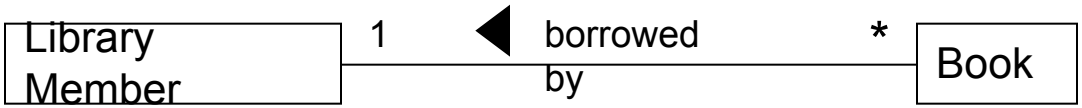
- Entities with common features, i.e. attributes and operations
- Classes are represented as solid outline rectangle with compartments
- Compartments for **name**, **attributes** & **operations**
- Attribute and operation compartment are optional for reuse purpose

Example of Class diagram



Different representations of the LibraryMember class

Association Relationship

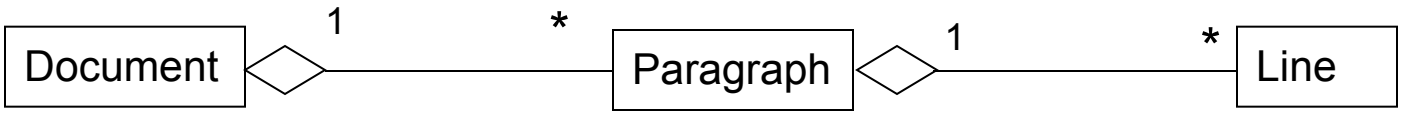


Association between two classes

Aggregation Relationship

- Represent a whole-part relationship
- Represented by **diamond** symbol at the composite end
- Cannot be reflexive(i.e. recursive)
- Not symmetric
- It can be transitive

Aggregation Relationship



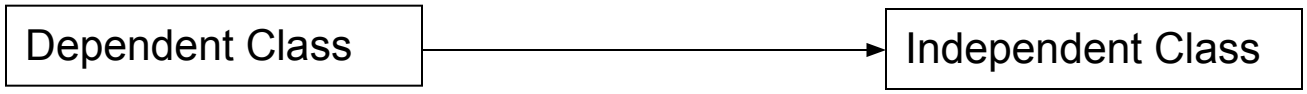
Representation of aggregation

Composition Relationship

- **Life of item is same as the order**

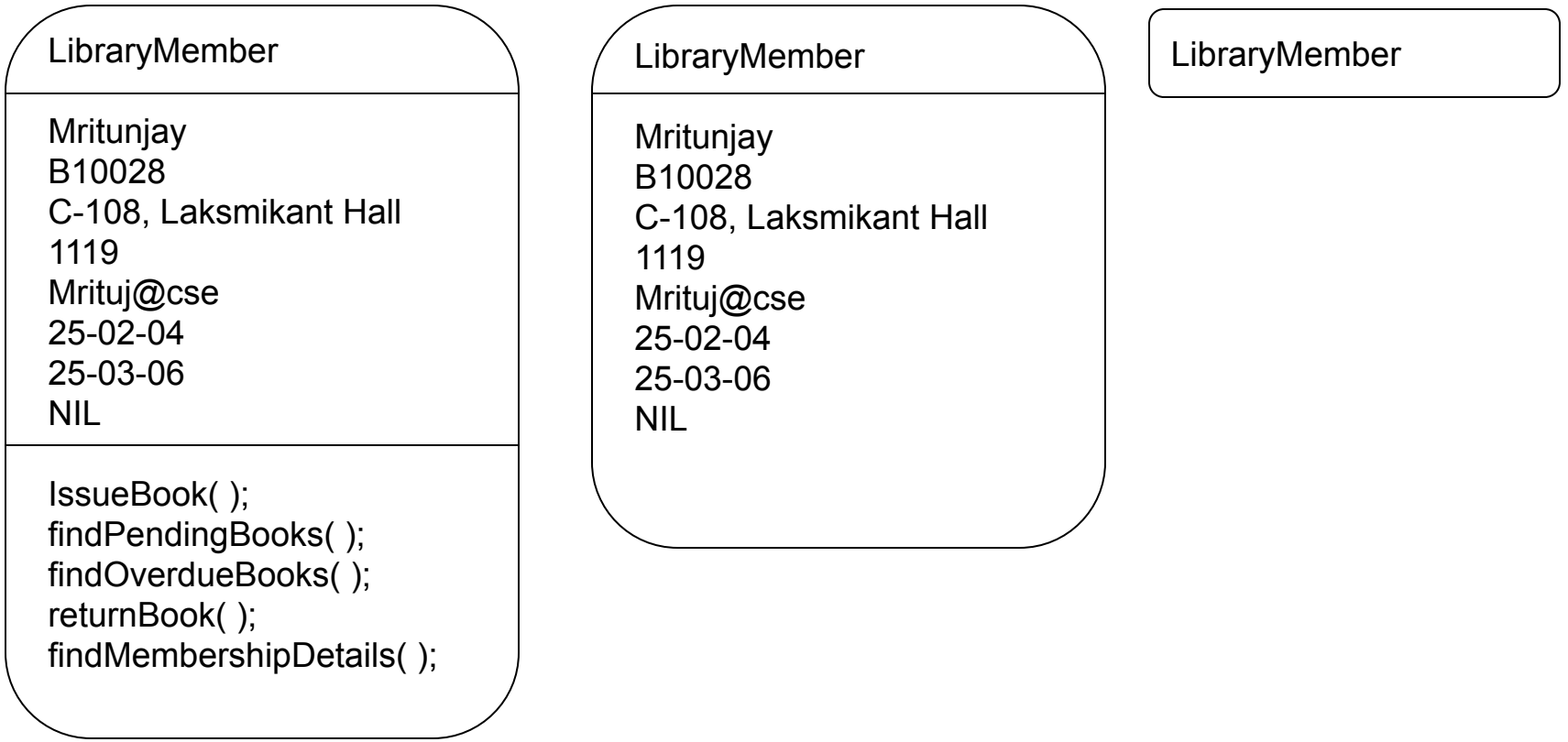
Representation of composition

Class Dependency



Representation of dependence between class

Object diagram



Different representations of the LibraryMember object

Interaction diagram

- **Models how groups of objects collaborate to realize some behaviour**
- **Typically each interaction diagram realizes behaviour of a single use case**

Interaction diagram

- Two kinds: **Sequence** & **Collaboration**
- Two diagrams are equivalent but portrays different perspective
- These diagram play a very important role in the design process

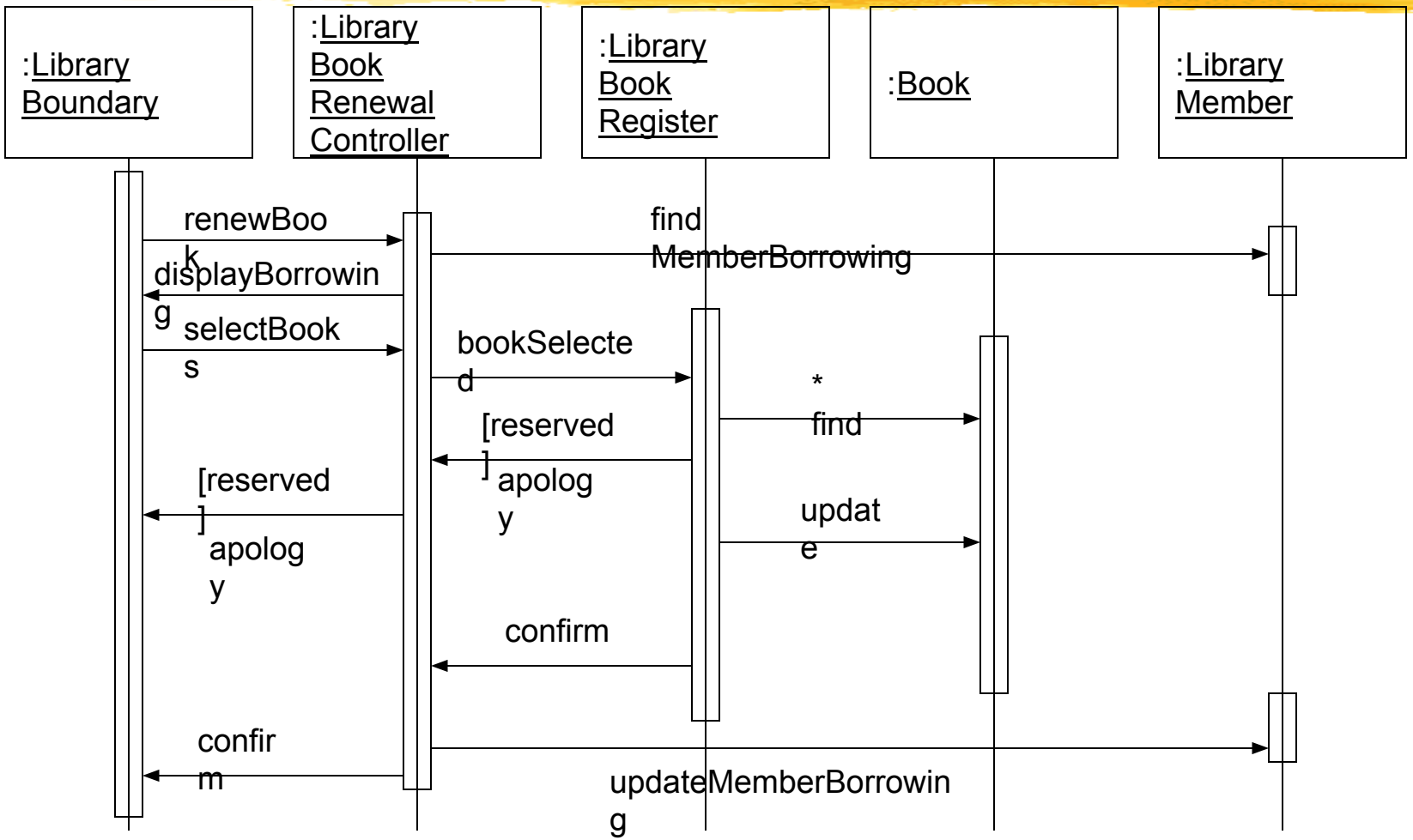
Sequence diagram

- Shows interaction among objects as two-dimensional chart
- **Objects** are shown as **boxes** at top
- **Objects existence** are shown as **dashed lines** (lifeline)
- **Objects activeness**, shown as **rectangle** on lifeline

Sequence diagram

- **Messages** are shown as **arrows**
- Message labelled with message name
- Message can be labelled with **control information**
- Two types of control information: **condition** ([]) & an **iteration** (*)

Example of Sequence diagram

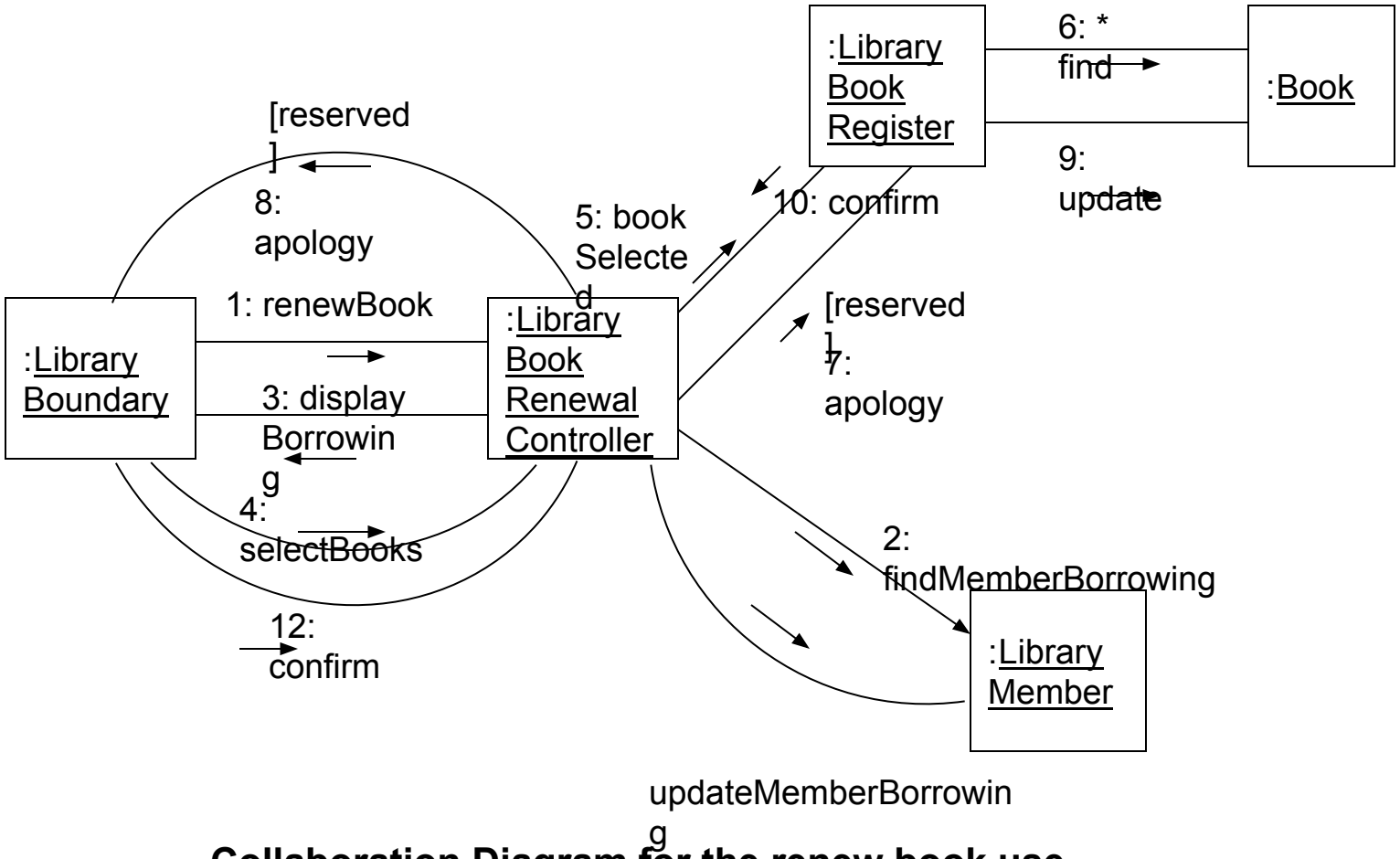


Sequence Diagram for the renew book use case

Collaboration diagram

- Shows both **structural** and **behavioural** aspects
- Objects are **collaborator**, shown as boxes
- Messages between objects shown as a **solid line**
- Message is shown as a **labelled arrow** placed near the link
- Messages are prefixed with **sequence numbers** to show relative sequencing

Example of Collaboration diagram



Collaboration Diagram for the renew book use case

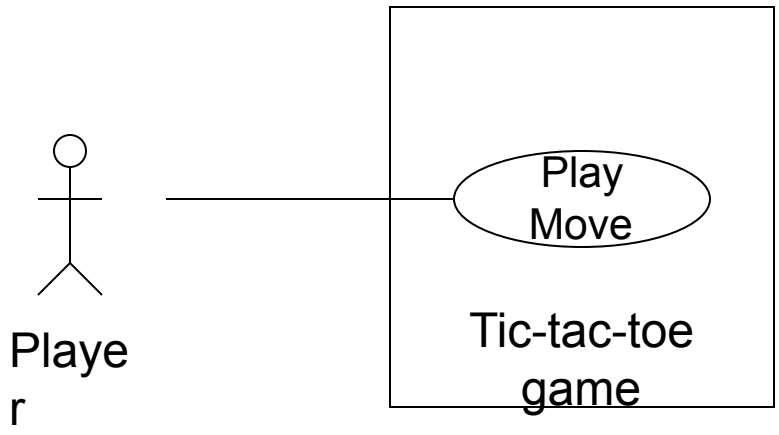
Example 1: Tic-Tac-Toe Computer Game

- A human player and the computer make alternate moves on a 3 3 square.
- A move consists of marking a previously unmarked square.
- The user inputs a number between 1 and 9 to mark a square
- Whoever is first to place three consecutive marks along a straight line (i.e., along a row, column, or diagonal) on the square wins.

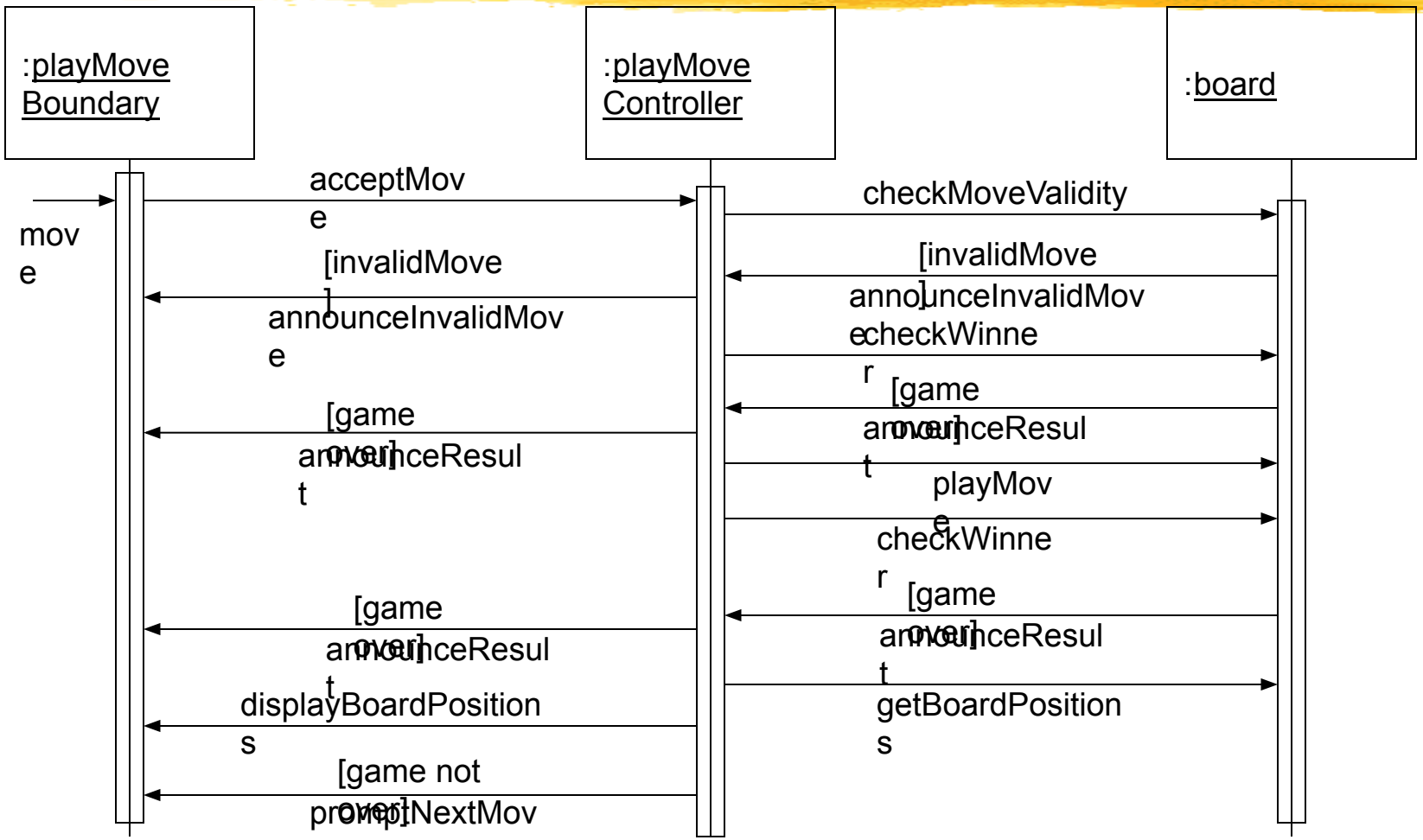
Example 1: Tic-Tac-Toe Computer Game

- As soon as either of the human player or the computer wins,
  - a message announcing 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.

Example 1: Use Case Model

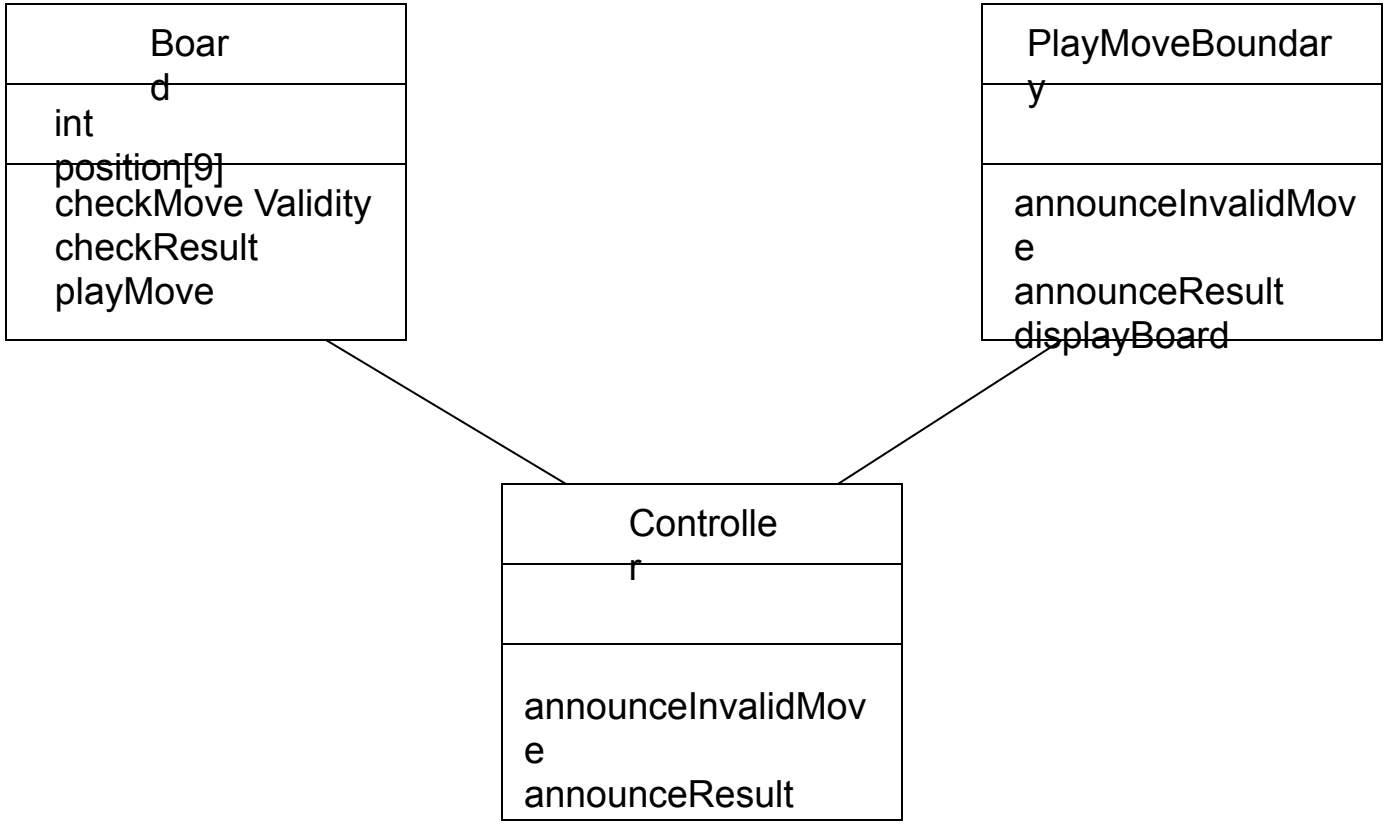


Example 1: Sequence Diagram



Sequence Diagram for the play move use case

Example 1: Class Diagram



Example 2: Supermarket Prize Scheme

- Supermarket needs to develop software to encourage regular customers.
- Customer needs to supply his residence address, telephone number, and the driving licence number.
- Each customer who registers is assigned a unique customer number (CN) by the computer.

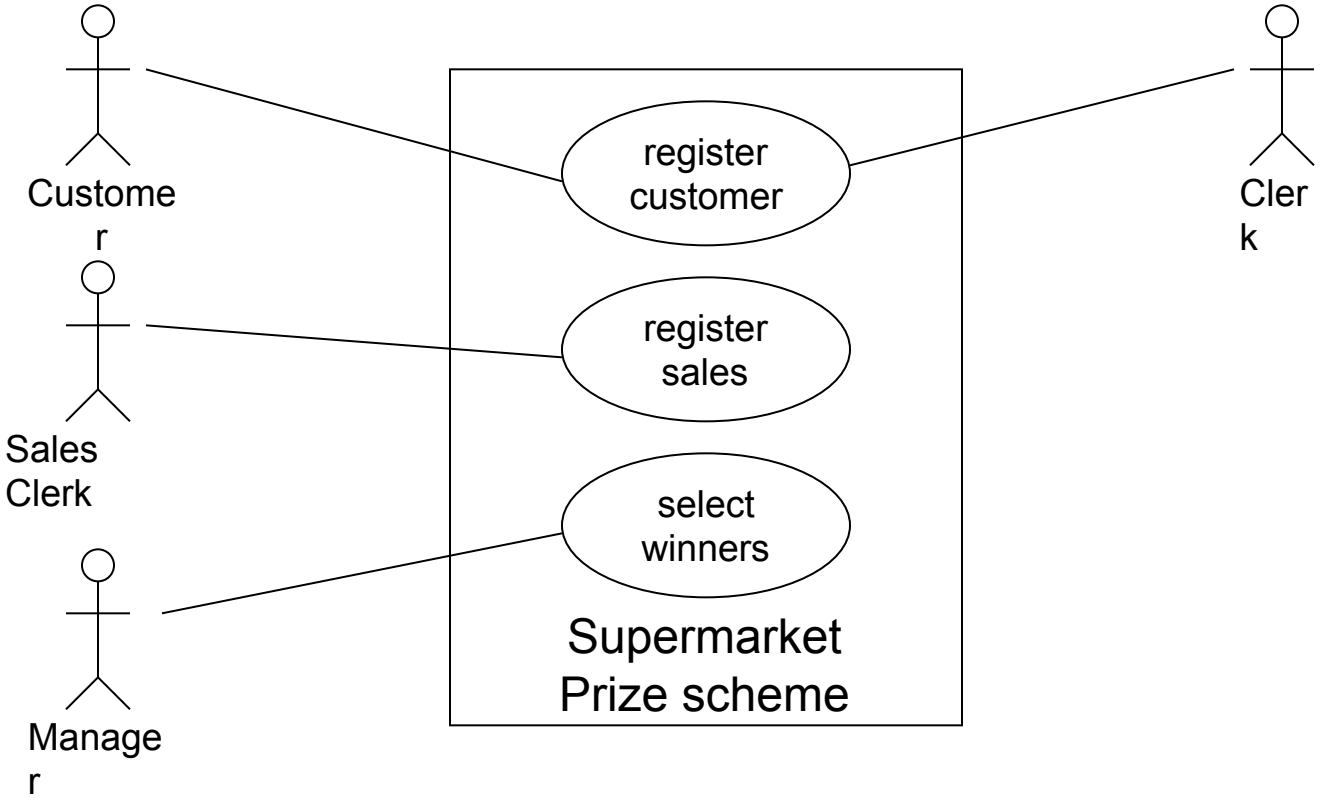
Example 2: Supermarket Prize Scheme

- A customer can present his CN to the staff when he makes any purchase.
- The value of his purchase is credited against his CN.
- At the end of each year, the supermarket awards surprise gifts to ten customers who make highest purchase.

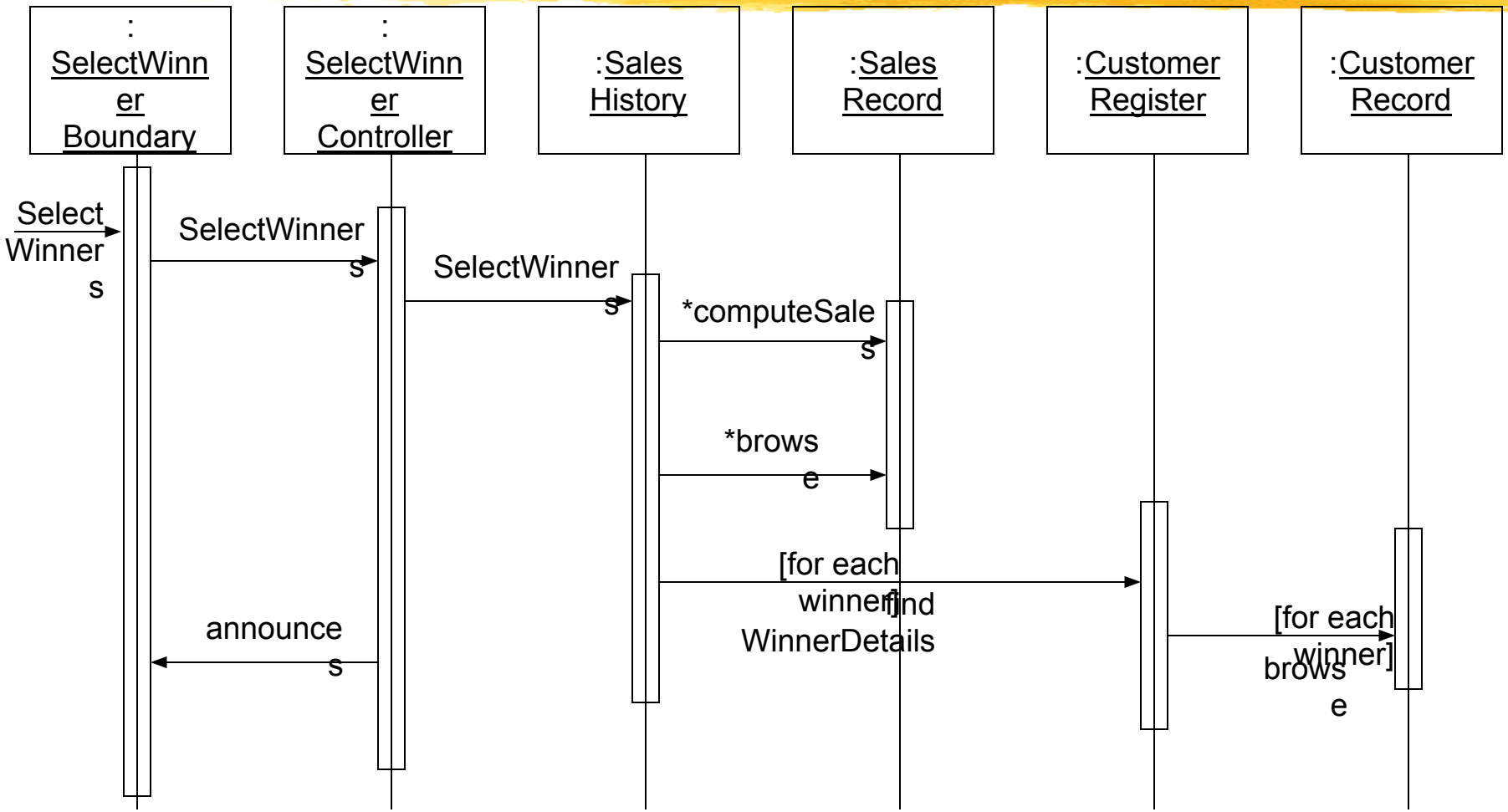
Example 2: Supermarket Prize Scheme

- Also, it awards a 22 carat gold coin to every customer whose purchases exceed Rs. 10,000.
- The entries against the CN are reset on the last day of every year after the prize winner's lists are generated.

Example 2: Use Case Model

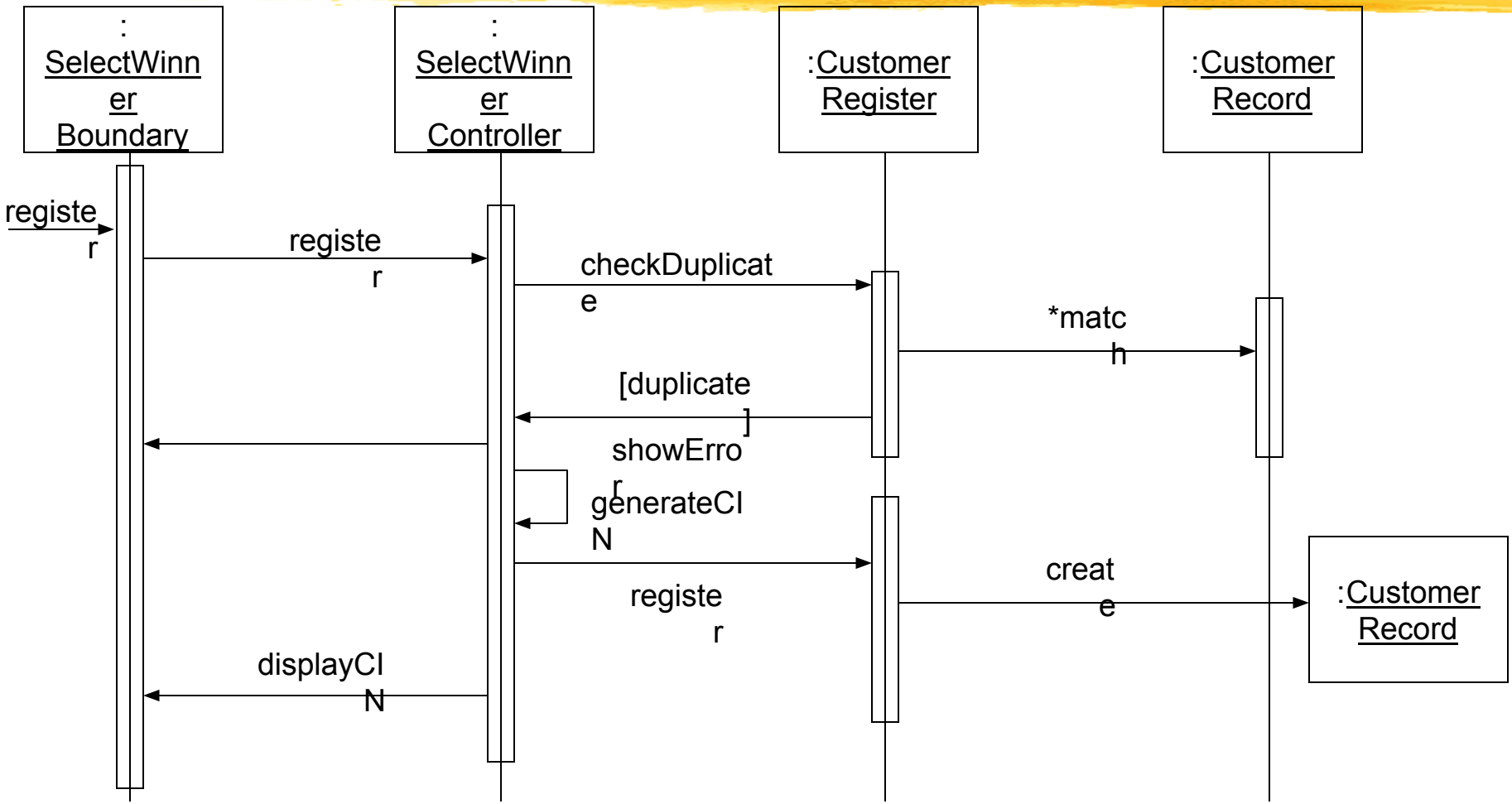


Example 2: Sequence Diagram for the Select Winners Use Case



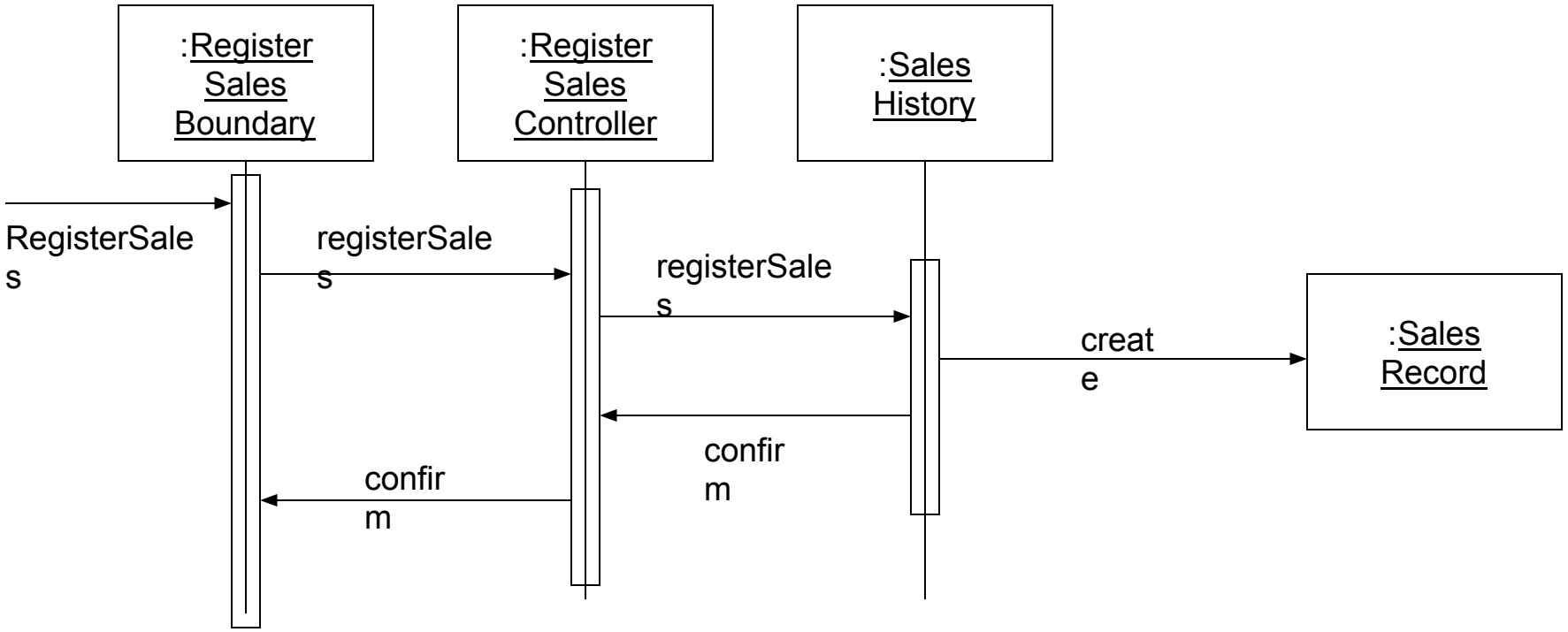
Sequence Diagram for the select winners use case

Example 2: Sequence Diagram for the Register Customer Use Case



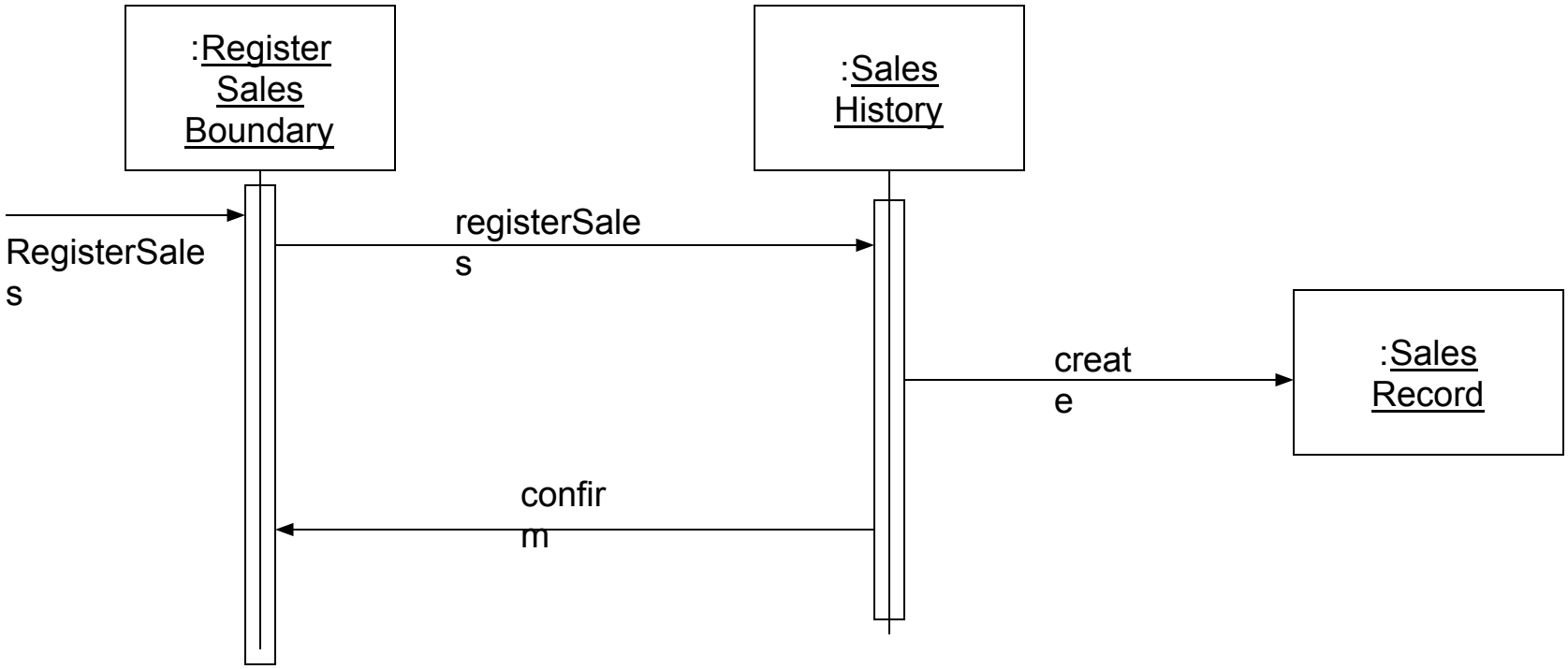
Sequence Diagram for the register customer use case

Example 2: Sequence Diagram for the Register Sales Use Case



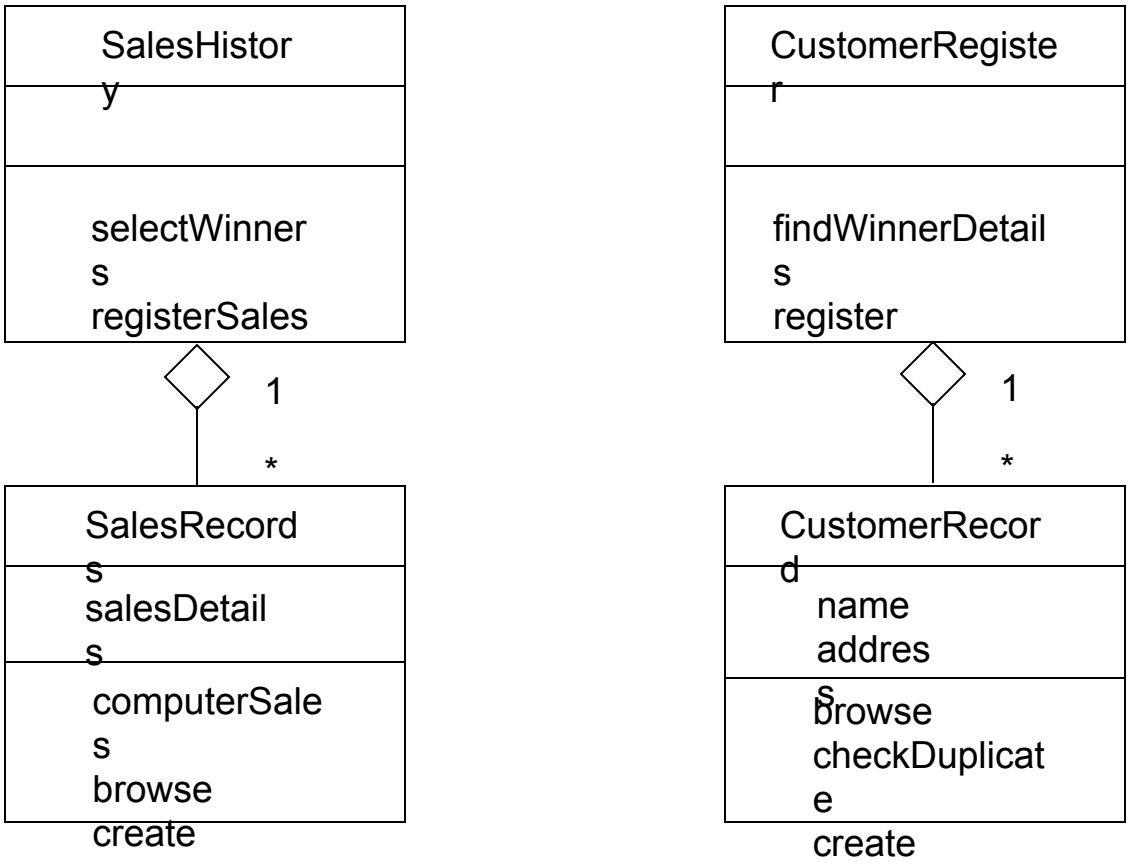
Sequence Diagram for the register sales use case

Example 2: Sequence Diagram for the Register Sales Use Case



Refined Sequence Diagram for the register sales use case

Example 1: Class Diagram



Summary

- **We discussed object-oriented concepts**
  - **Basic mechanisms:** **Such as objects, class, methods, inheritance etc.**
  - **Key concepts: Such as abstraction, encapsulation, polymorphism, composite objects etc.**

Summary

- We discussed an important OO language UML
  - Its origin, as a standard, as a model
  - Use case representation, its factorisation such as generalization, includes and extends
  - Different diagrams for UML representation
  - In class diagram we discussed some relationships association, aggregation, composition and inheritance

Summary

- Some more diagrams such as **interaction diagrams** (sequence and collaboration), **activity diagrams**, **state chart diagram**
- We discussed OO software development process and patterns
  - In this we discussed some **patterns** example and **domain modelling**