Object-Orientation Concepts, UML, and OOAD

2008

भारतीय प्रौद्योगिकी संस्थान भुवनेश्वर

INDIAN INSTITUTE OF TECHNOLOGY BHUBANESWAR

Advantages of Object-Oriented Development

- **Code and design reuse**
- **Increased productivity**
- **Elegant design:**
  - Loosely coupled, highly cohesive objects:
  - Essential for solving large problems.
  - Ease of testing and maintenance
  - Better understandability

Advantages of Object-Oriented Development cont…

- **Initially incurs higher costs**
  - After completion of some projects reduction in cost become possible
- **Using well-established OO methodology and environment:**
  - Projects can be managed with 20% -- 50% of traditional cost of development.

Object Modelling Using UML

- UML is a modelling language
- Used to document object-oriented analysis and design results.

UML Origin

- **OOD in late 1980s and early 1990s:**
  - Different software development houses were using different notations.
  - Methodologies were tied to notations.

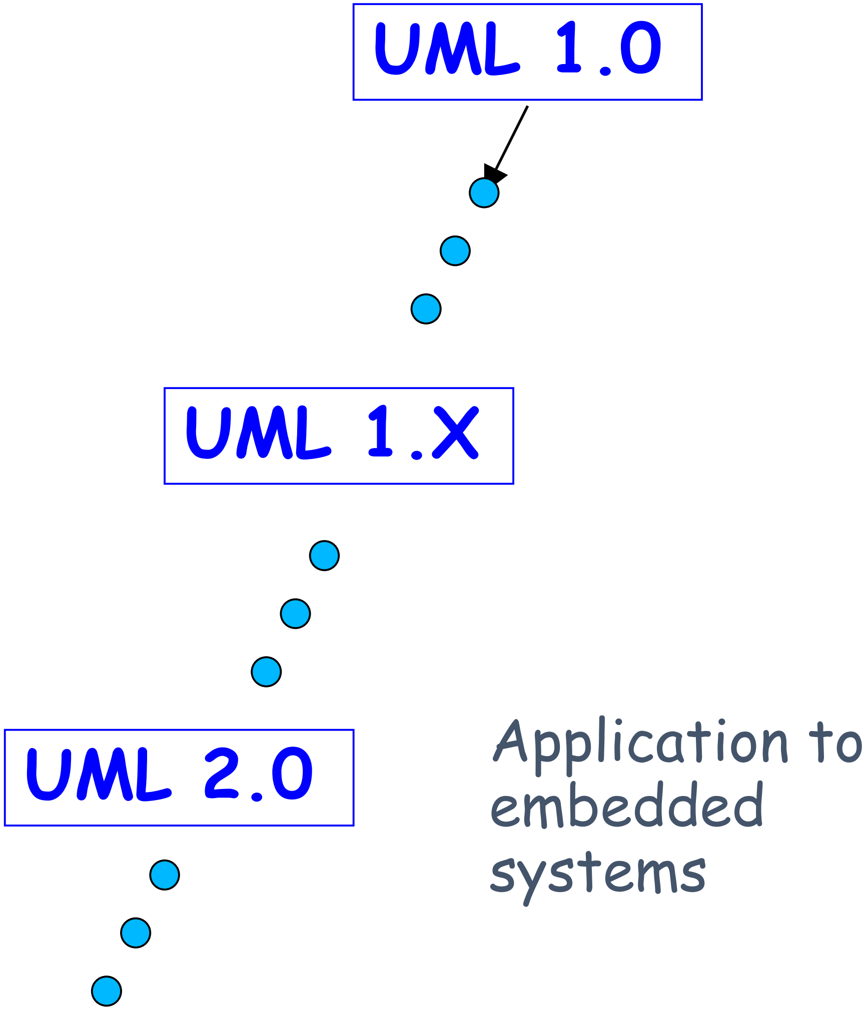
- **UML developed in early 1990s to:**
  - Standardize the large number of object-oriented modelling notations

UML as a Standard

- Adopted by Object Management Group (OMG) in 1997
- OMG is an association of industries
- Promotes consensus notations and techniques
- Used outside software development
  - Example car manufacturing

Developments to UML

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



UML 1.0

UML 1.X

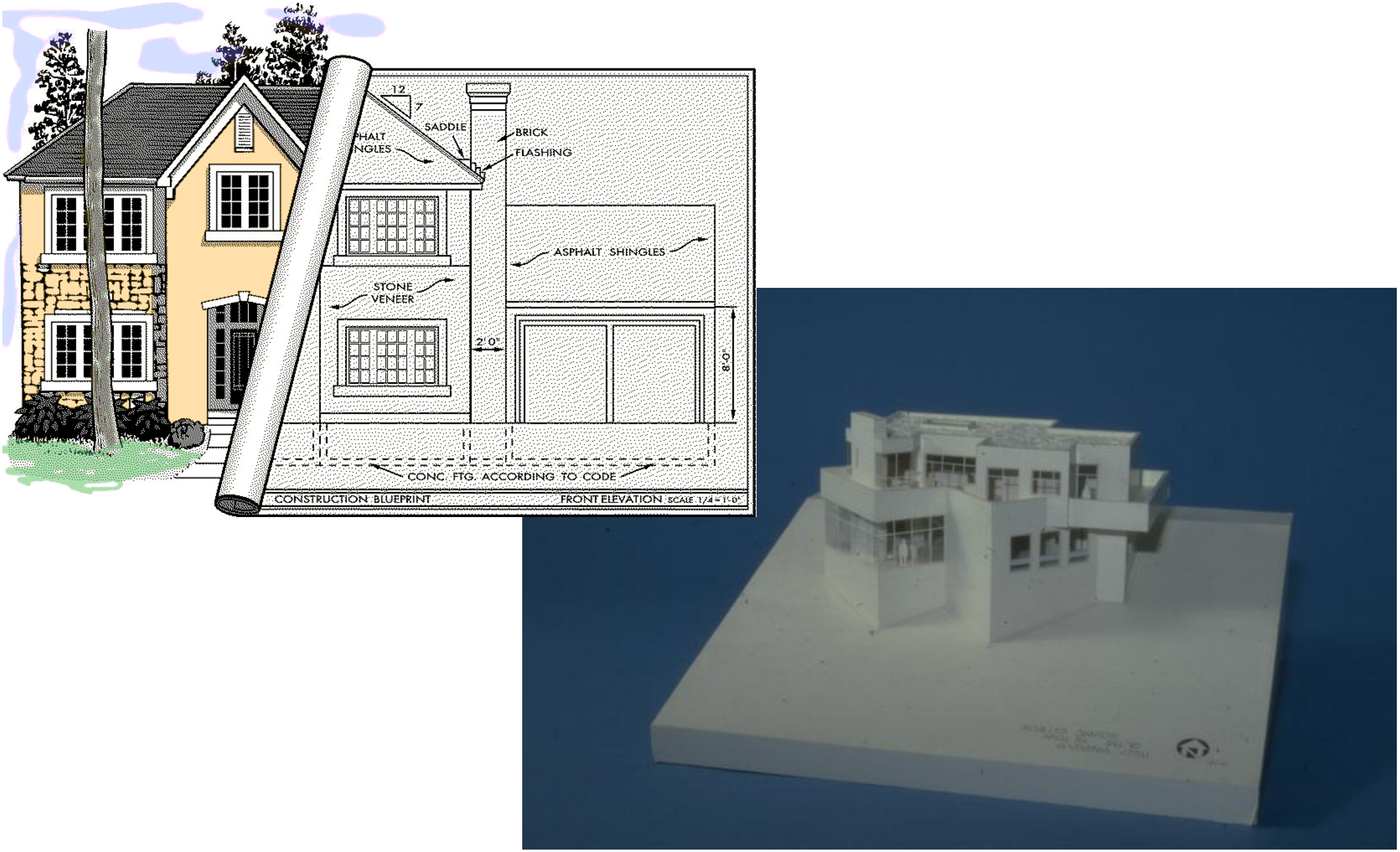
UML 2.0

Application to embedded systems

Why are UML Models Required?

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

Modeling a House



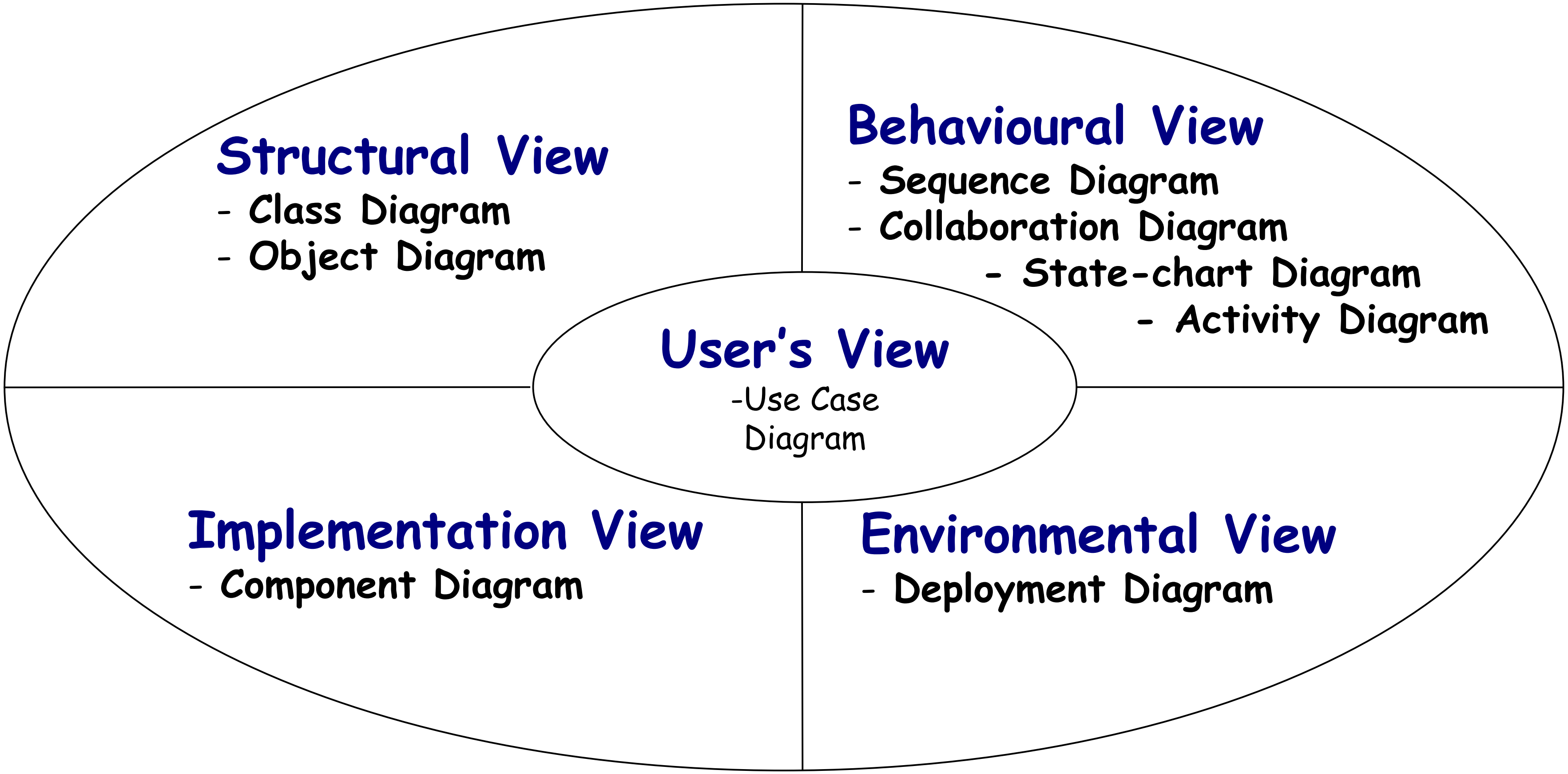
UML Diagrams

- **Nine diagrams are used to capture 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

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

Are All Views Required for Developing a Typical System?

- **NO**

- **Use case diagram, class diagram and one of the interaction diagram for a simple system**
- **When states are only one or two, state chart model becomes trivial**
- **Deployment diagram in case of large number of hardware components used to develop the system**

Use Case Model

- Consists of set of “use cases”
- An important analysis and design artifact
- The central model:
  - 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 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

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.

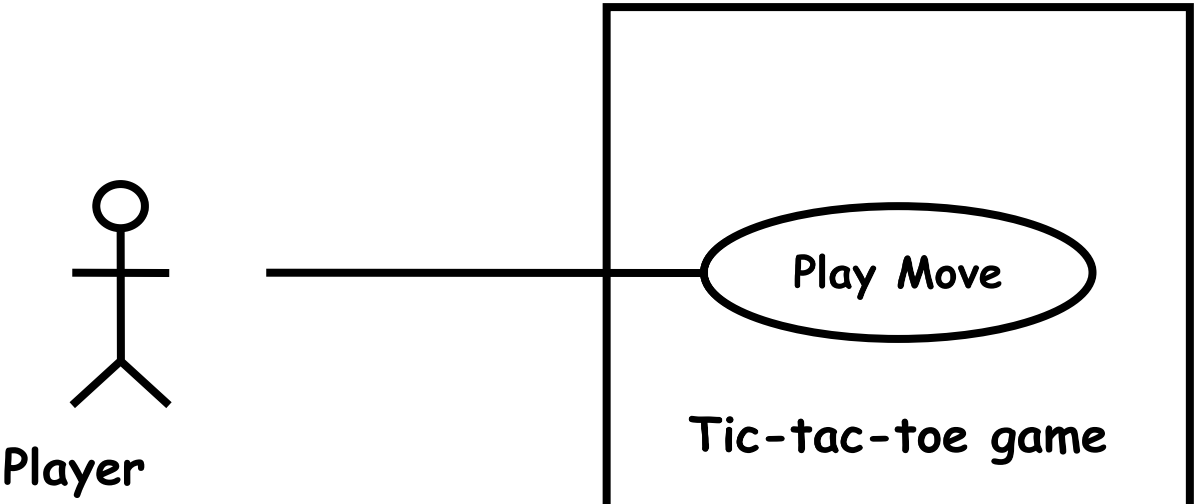
Example Use Cases

- For library information system
  - issue-book
  - query-book
  - return-book
  - create-member
  - add-book, etc.

Representation of Use Cases

- Represented by use case diagram
- 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

An Example Use Case Diagram



Use case model

Why Develop A Use Case Diagram?

- Serves as requirements specification

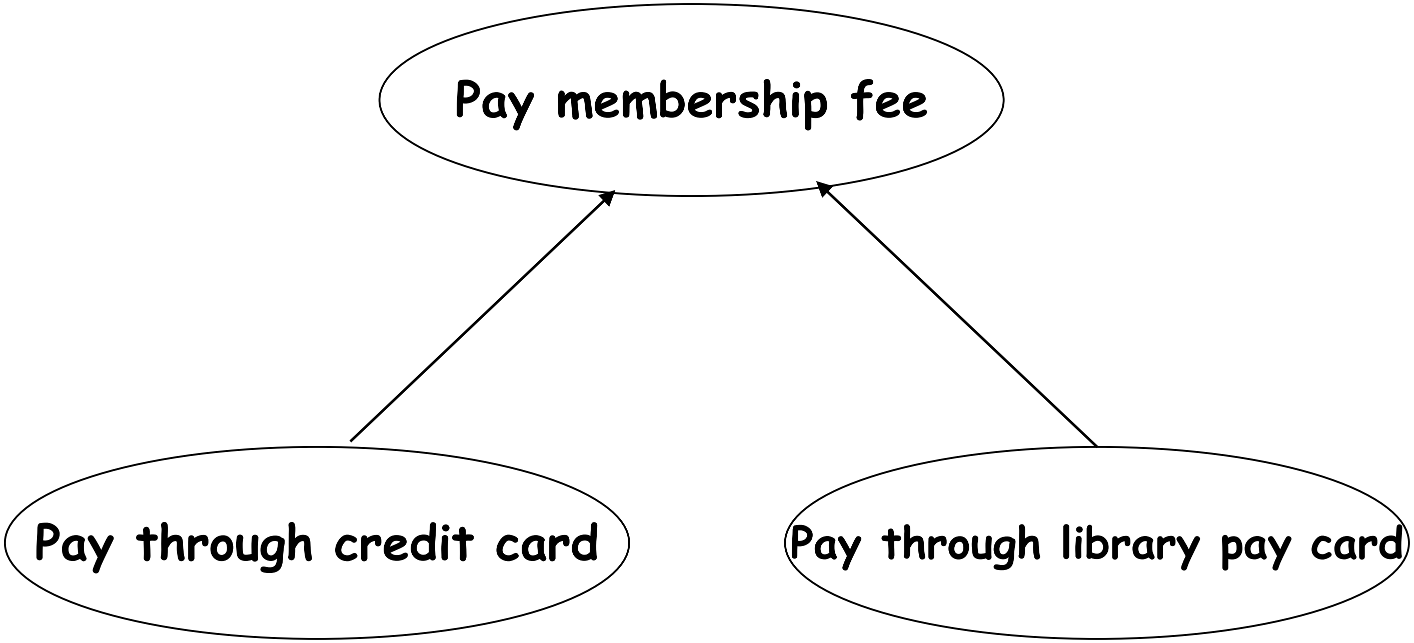
- **Actor identification** useful in software development:
  - User identification helps in implementing appropriate interfaces for different categories of users
  - Another use in preparing appropriate documents (e.g. **user’s manual**).

Factoring Use Cases

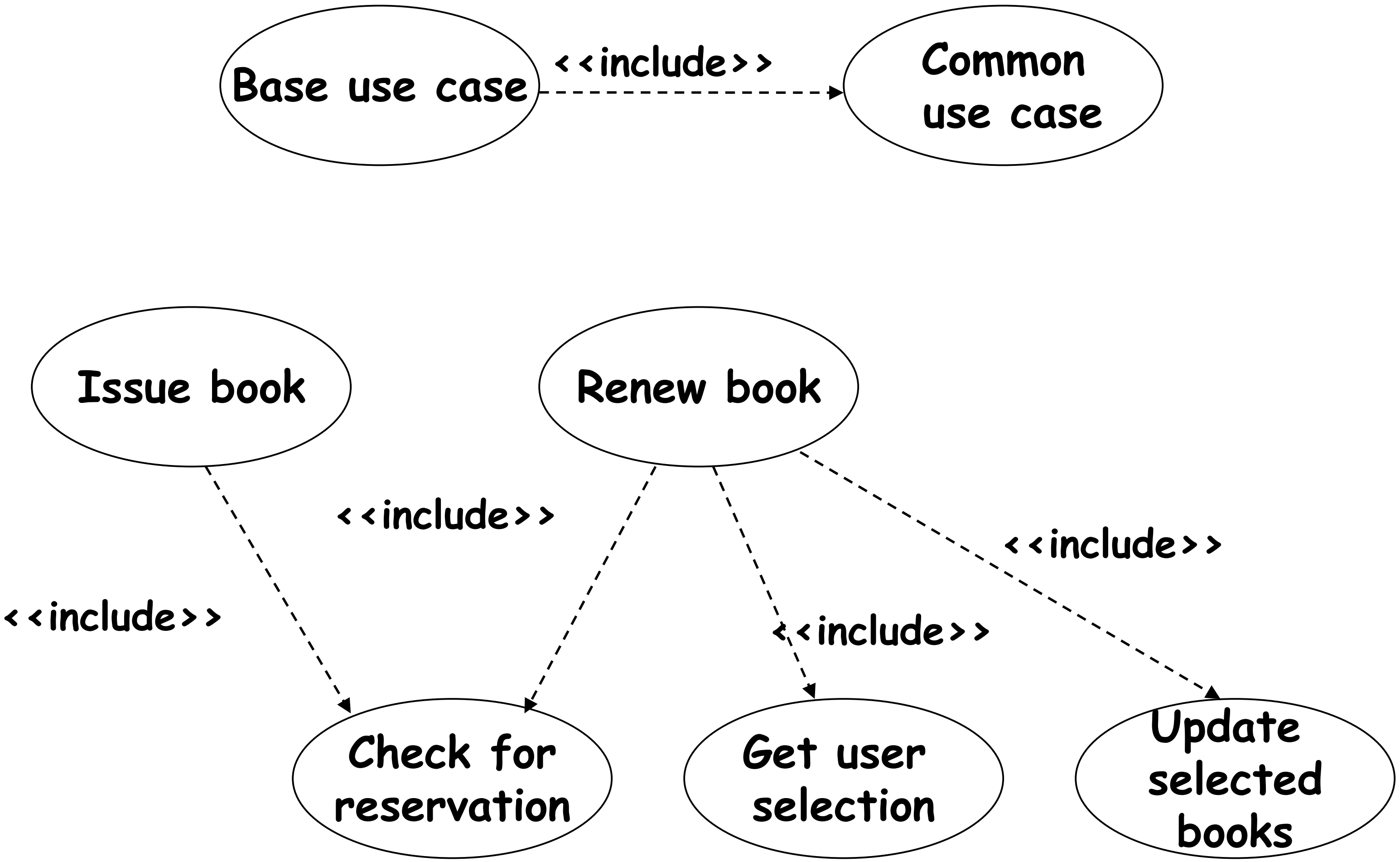
- Two main reasons for factoring:
  - **Complex use cases** need **to be factored** into simpler use cases
  - To represent common behavior across different use cases

- Three ways of factoring:
  - Generalization
  - Includes
  - Extends

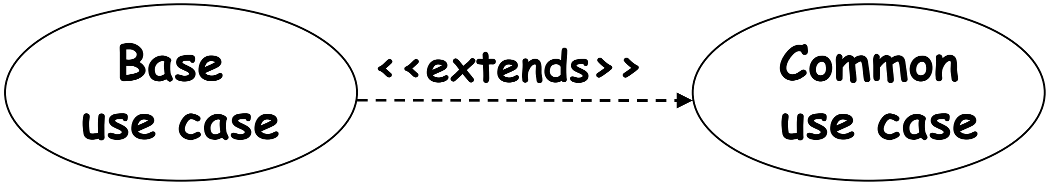
Factoring Use Cases Using Generalization



Factoring Use Cases Using Includes

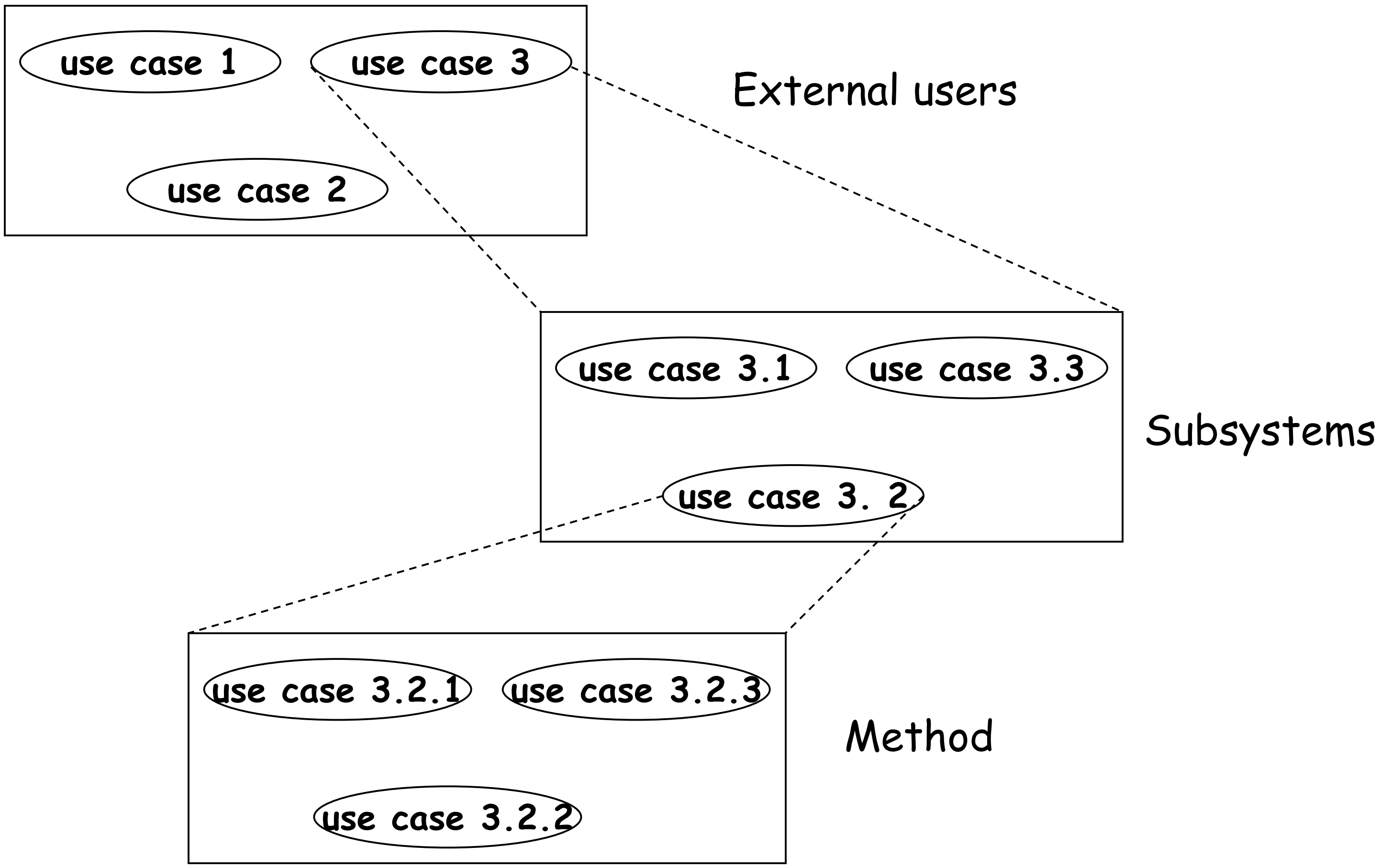


Factoring Use Cases Using Extends

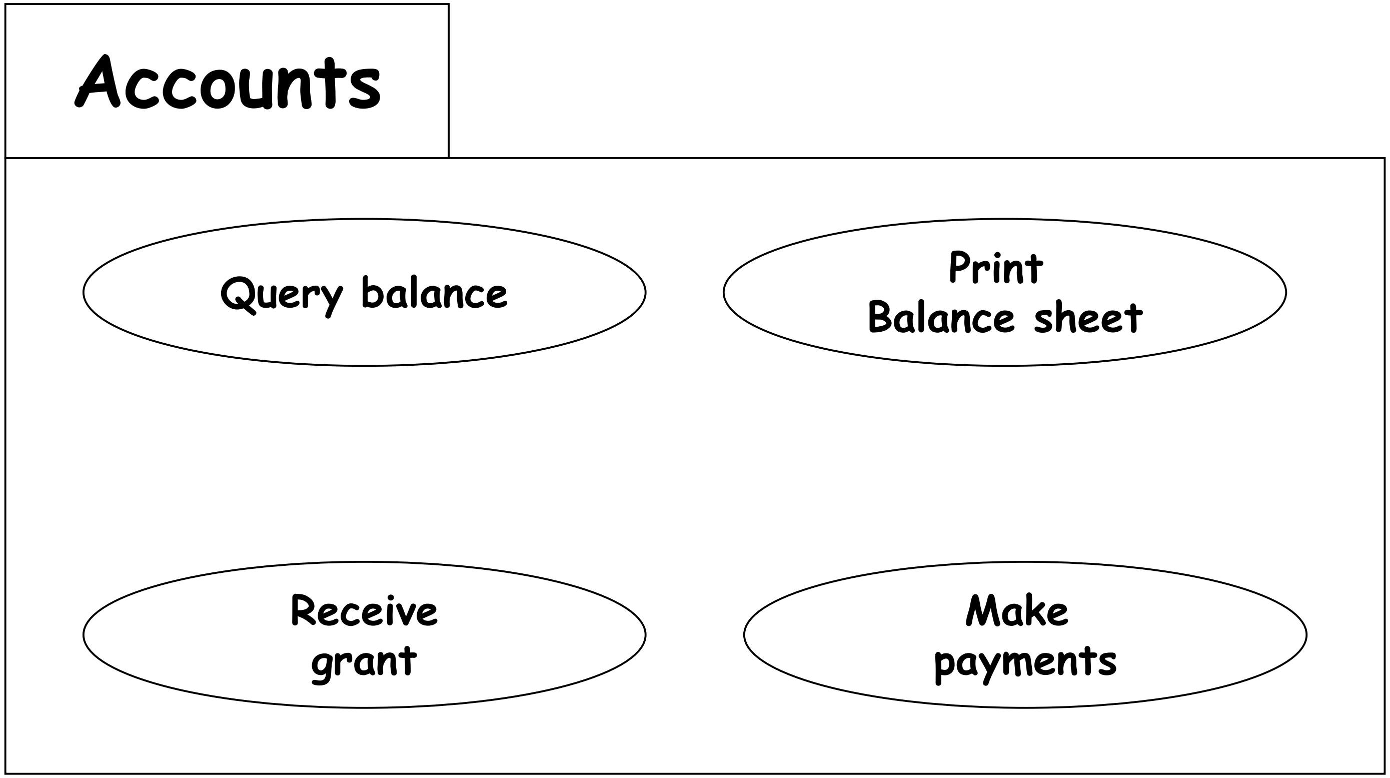


- Allows to show **optional** system behaviour.
- Optional behaviour executed if certain conditions hold.

Hierarchical Organization of Use Cases



Use Case Packaging



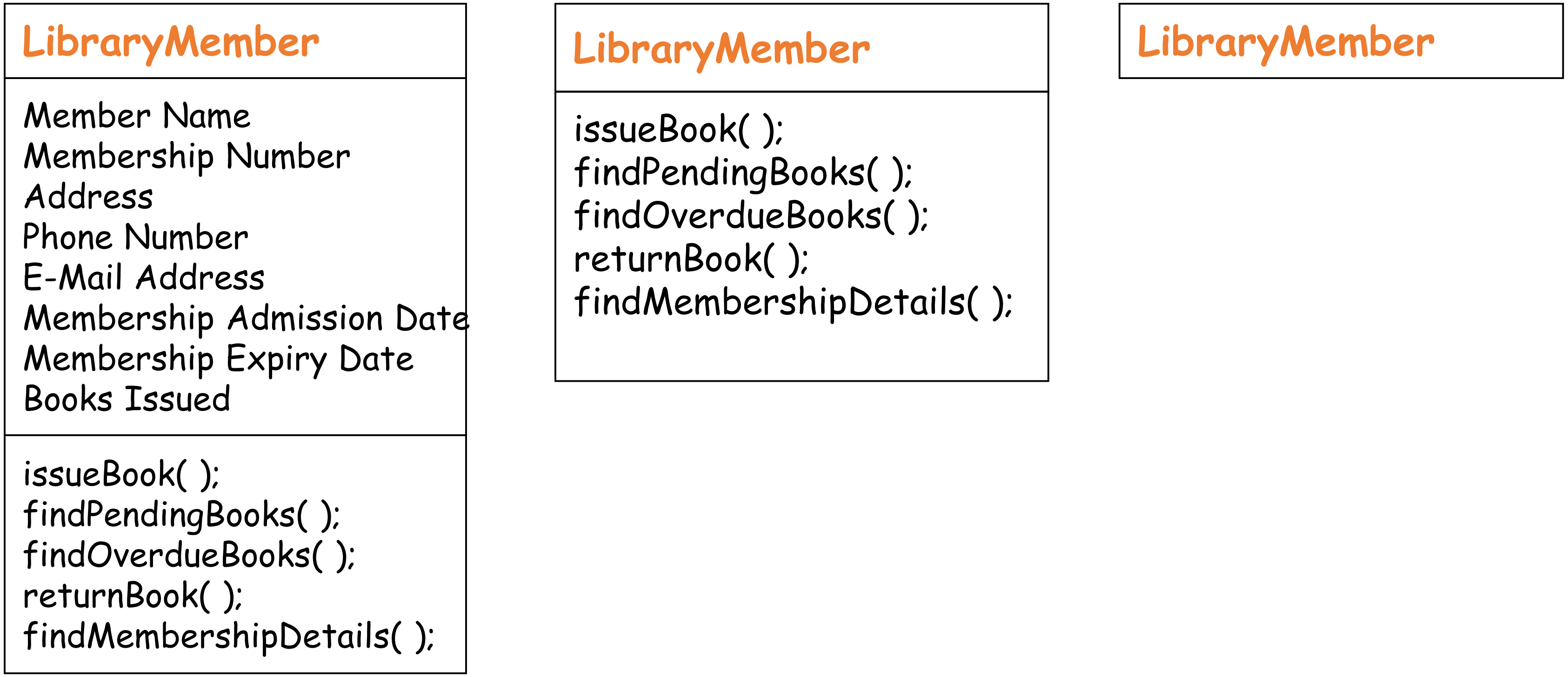
Class Diagram

- Describes **static structure** of a system
- Main constituents are classes and their relationships:
    - Generalization
    - 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, and operations.
- Attribute and operation compartments are optional depending on the purpose of a diagram.

Example Class Diagram



LibraryMember

Member Name
Membership Number
Address
Phone Number
E-Mail Address
Membership Admission Date
Membership Expiry Date
Books Issued

issueBook( );
findPendingBooks( );
findOverdueBooks( );
returnBook( );
findMembershipDetails( );

LibraryMember

issueBook( );
findPendingBooks( );
findOverdueBooks( );
returnBook( );
findMembershipDetails( );

LibraryMember

Different representations of the LibraryMember class

What are the Different Types of Relationships Among Classes?

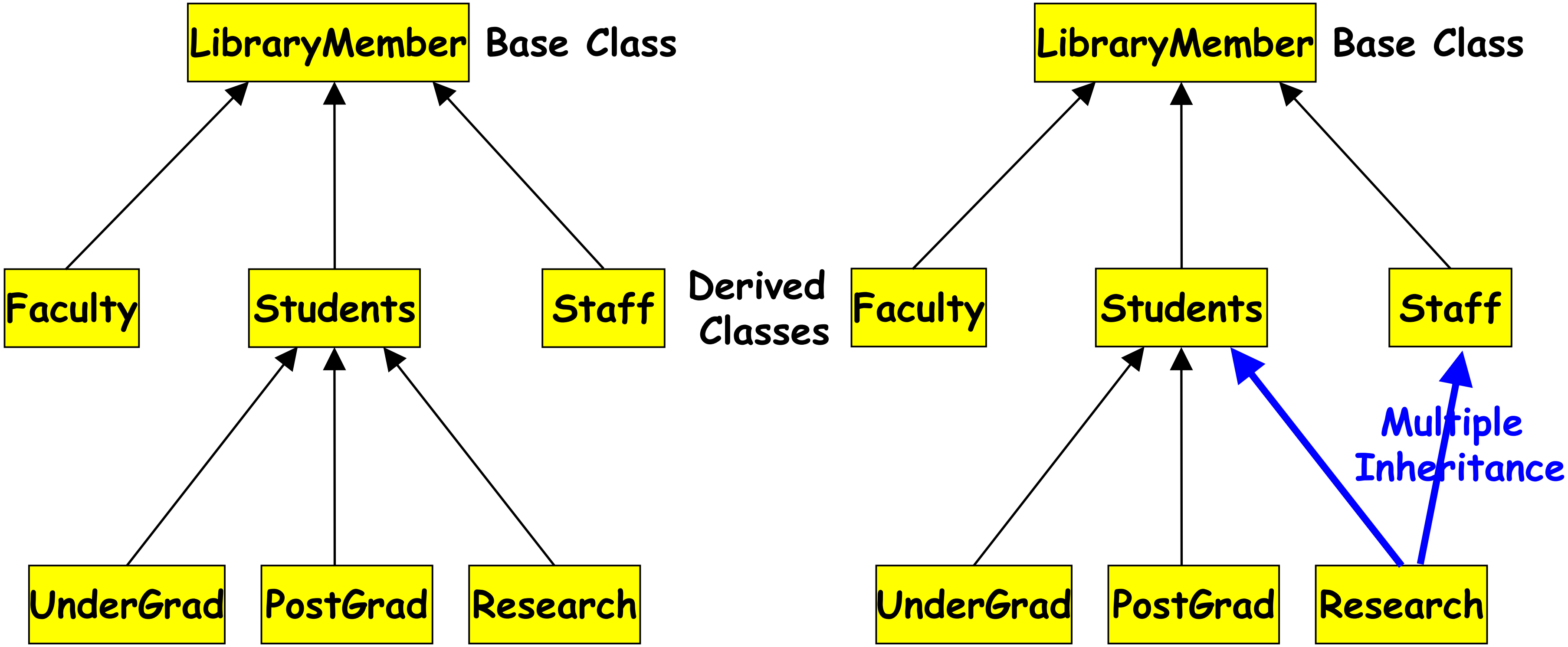
- Four types of relationships:
  - Inheritance
  - Association
  - Aggregation/Composition
  - Dependency

Inheritance

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

Multiple Inheritance

cont…



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

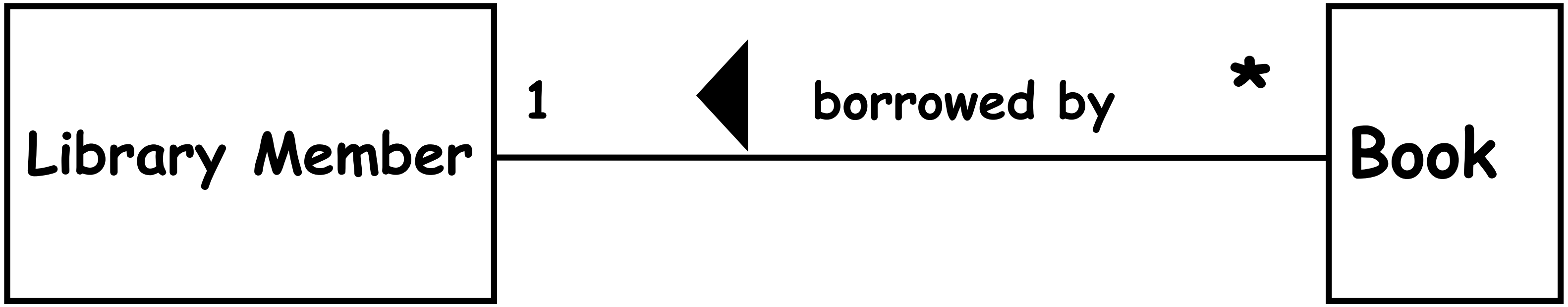
Association Relationship

- Enables objects to communicate with each other:
  - Thus one object must “know” the address of the corresponding object in the association.
- Usually binary:
  - But in general can be n-ary.

Association Relationship

- A class can be associated with itself (recursive association).

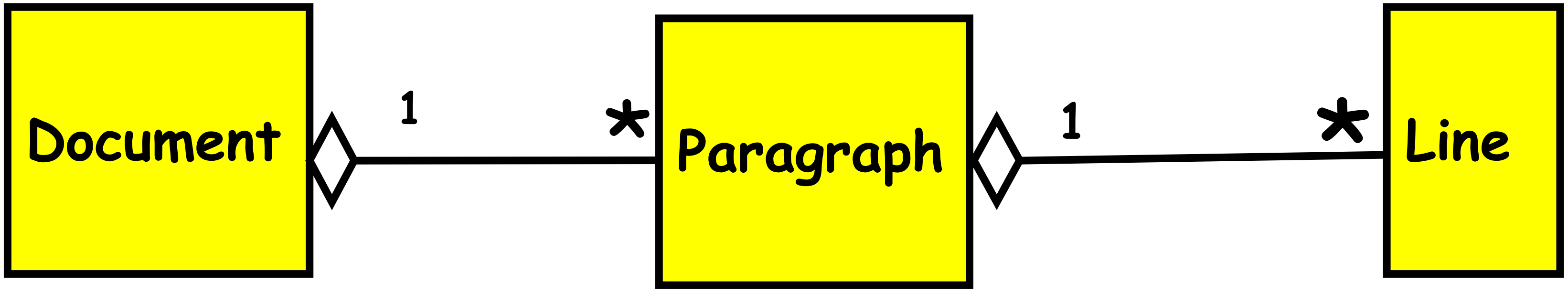
Association Relationship



Aggregation Relationship

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

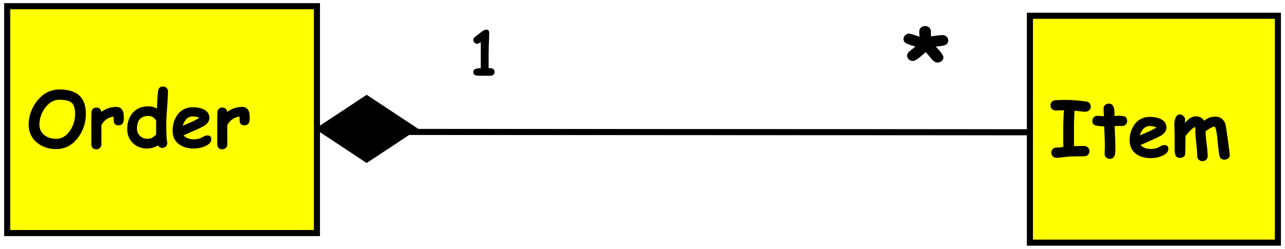
Aggregation Relationship



- Represents whole-part relationship
- Represented by a diamond symbol at the composite end

Composition Relationship

- Life of item is same as the order

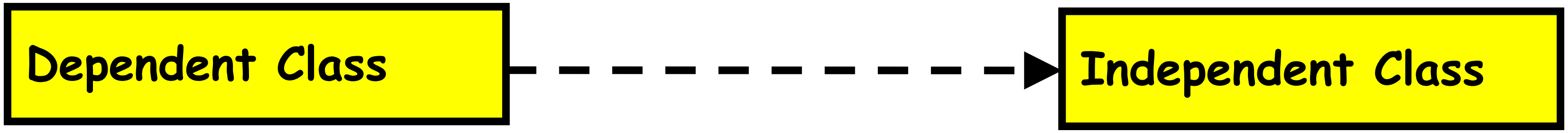


Aggregation

cont…

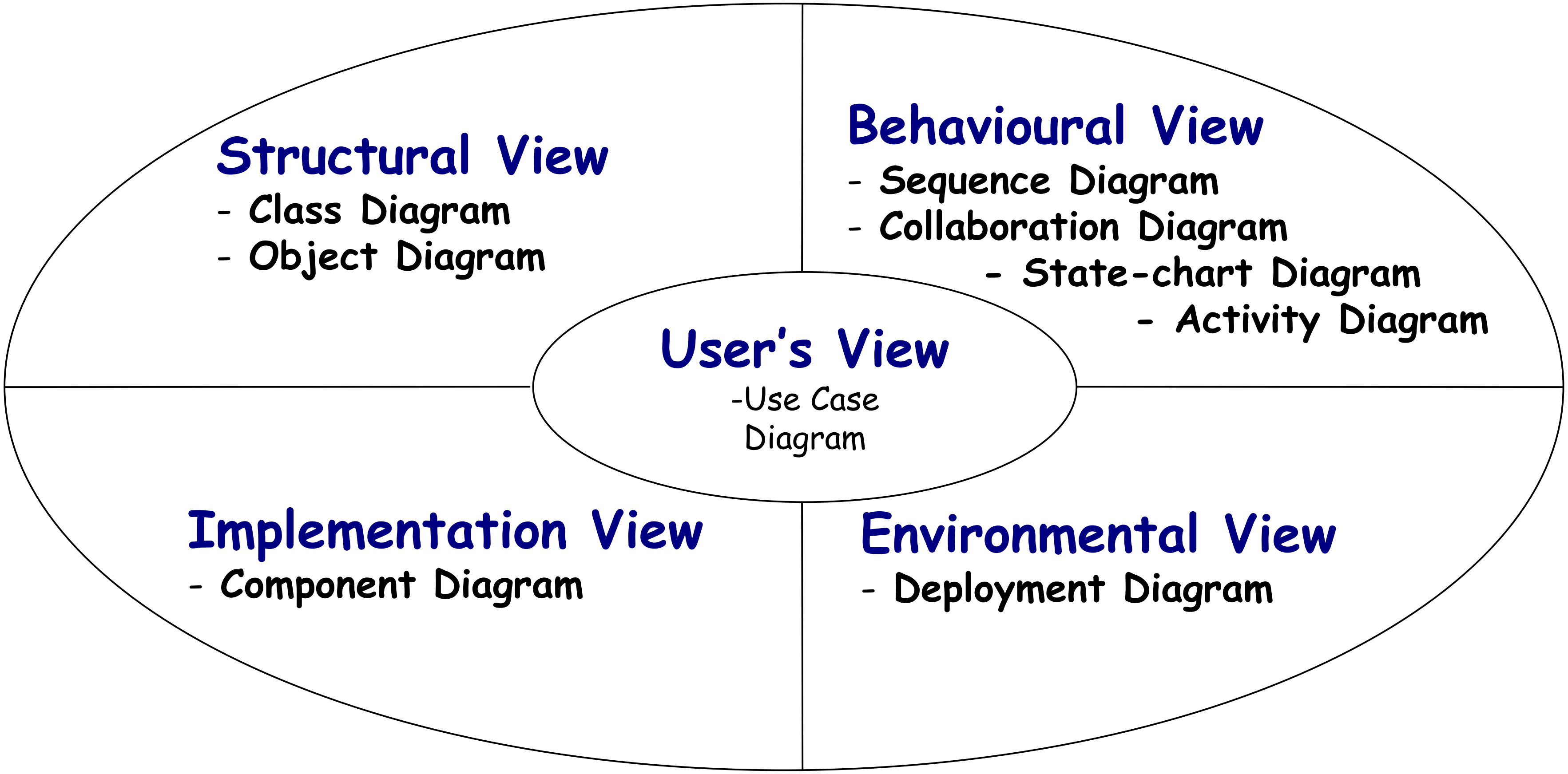
- **A aggregate object contains other objects.**
- **Aggregation limited to tree hierarchy:**
  - No circular inclusion relation.

Class Dependency



Representation of dependence between classes

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

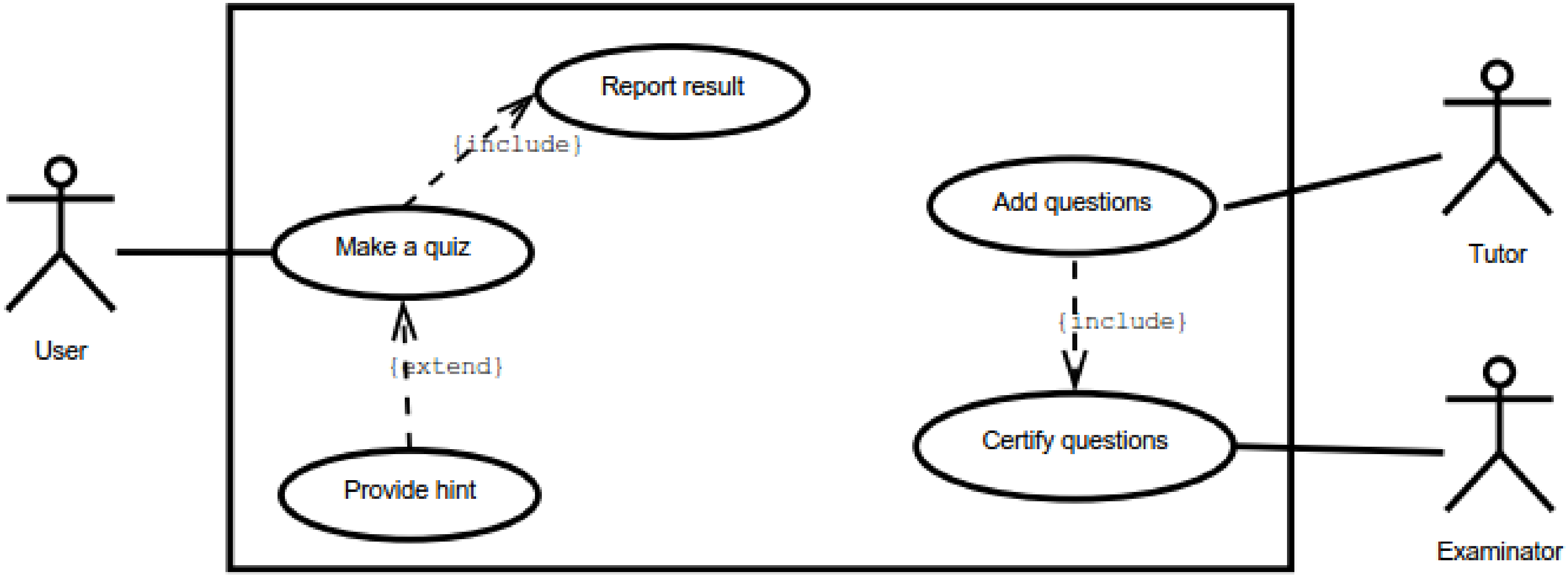
Diagrams and views in UML

Use case diagram: Example

- A user can request a quiz for the system. The system picks a set of questions from its database, and compose them together to make a quiz. It rates the user’s answers, and gives hints if the user requests it.

- In addition to users, we also have tutors who provide questions and hints. And also examinators who must certify questions to make sure they are not too trivial, and that they are sensical.

Use case diagram

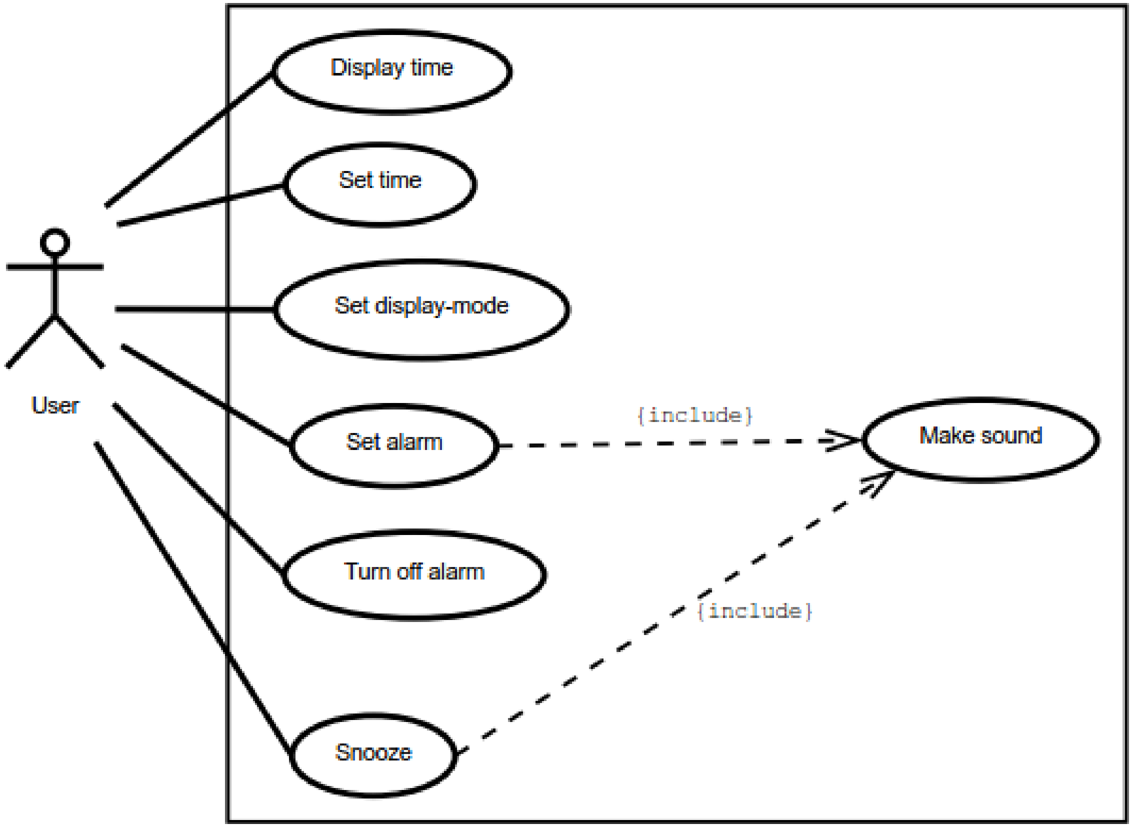


Use case diagram 2

- The clock shows the time of day. Using buttons, the user can set the hours and minutes fields individually, and choose between 12 and 24-hour display.
- It is possible to set one or two alarms. When an alarm fires, it will make some sound/. The user can turn it off, or choose to ’snooze’.
- If the user does not respond at all, the alarm will turn off itself after 2 minutes. ’Snoozing’ means to turn off the sound, but the alarm will fire again after some minutes of delay. This ’snoozing time’ is pre-adjustable.

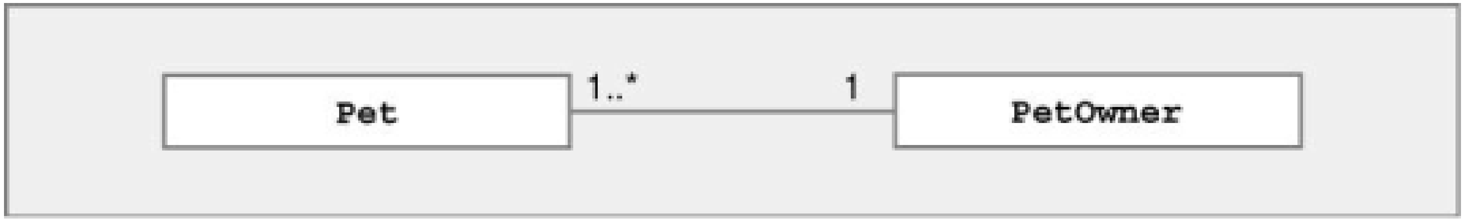
Identify the top-level functional requirements for the clock, and model it with a use case diagram.

Use case diagram 2



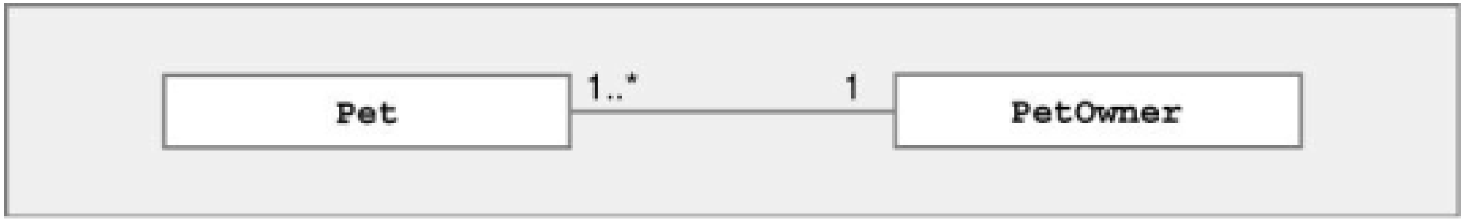
Class diagram 1

- Read and understand the following:

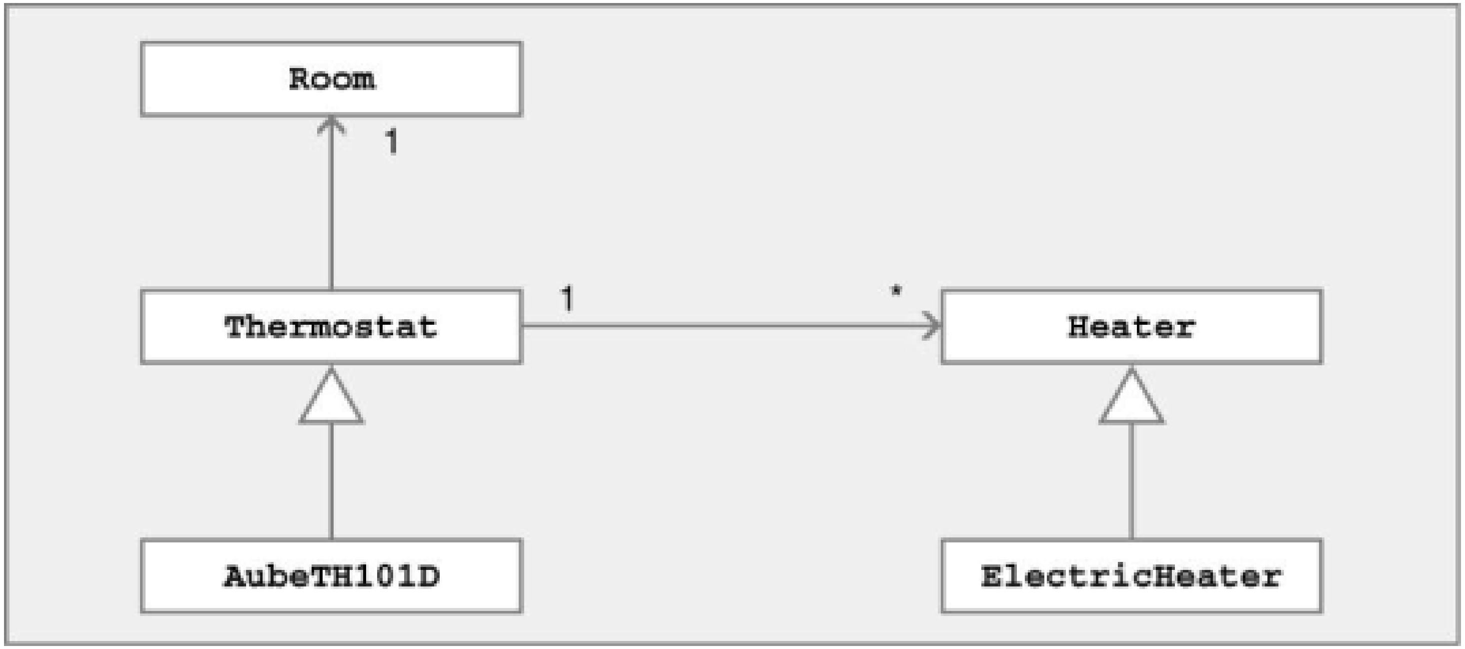


Class diagram 1

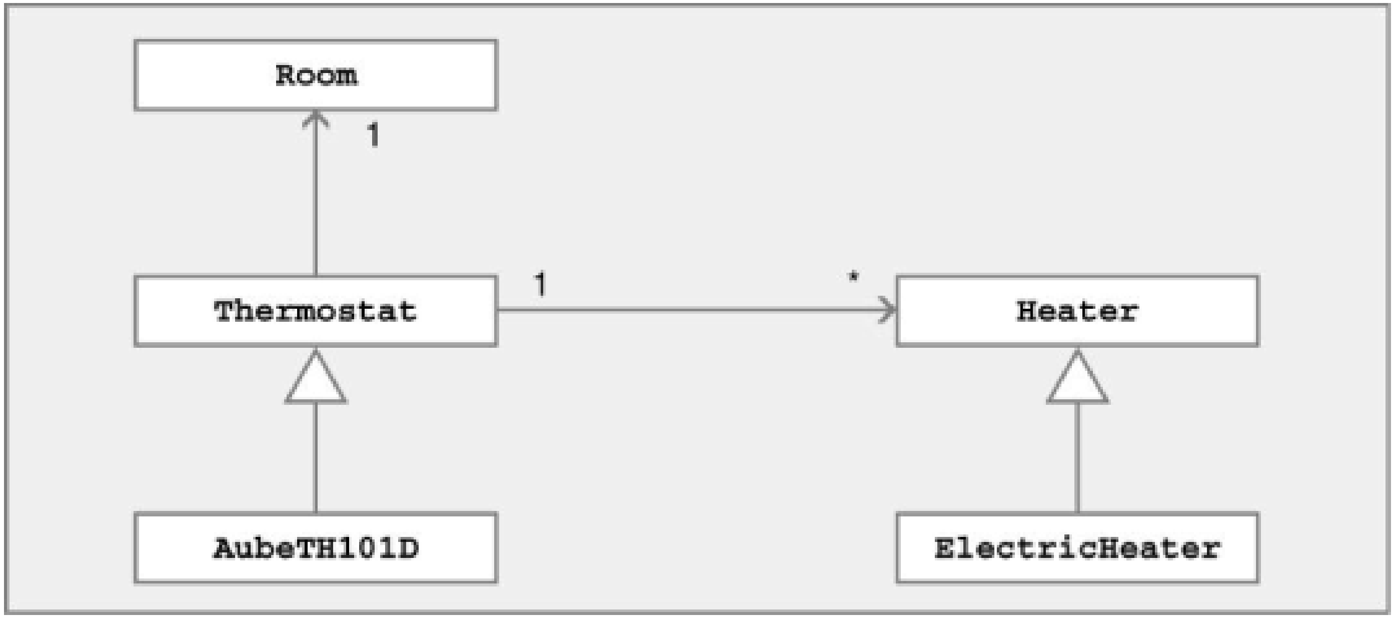
- 1 or more Pets associated with 1 PetOwner
- Each pet has exactly one PetOwner



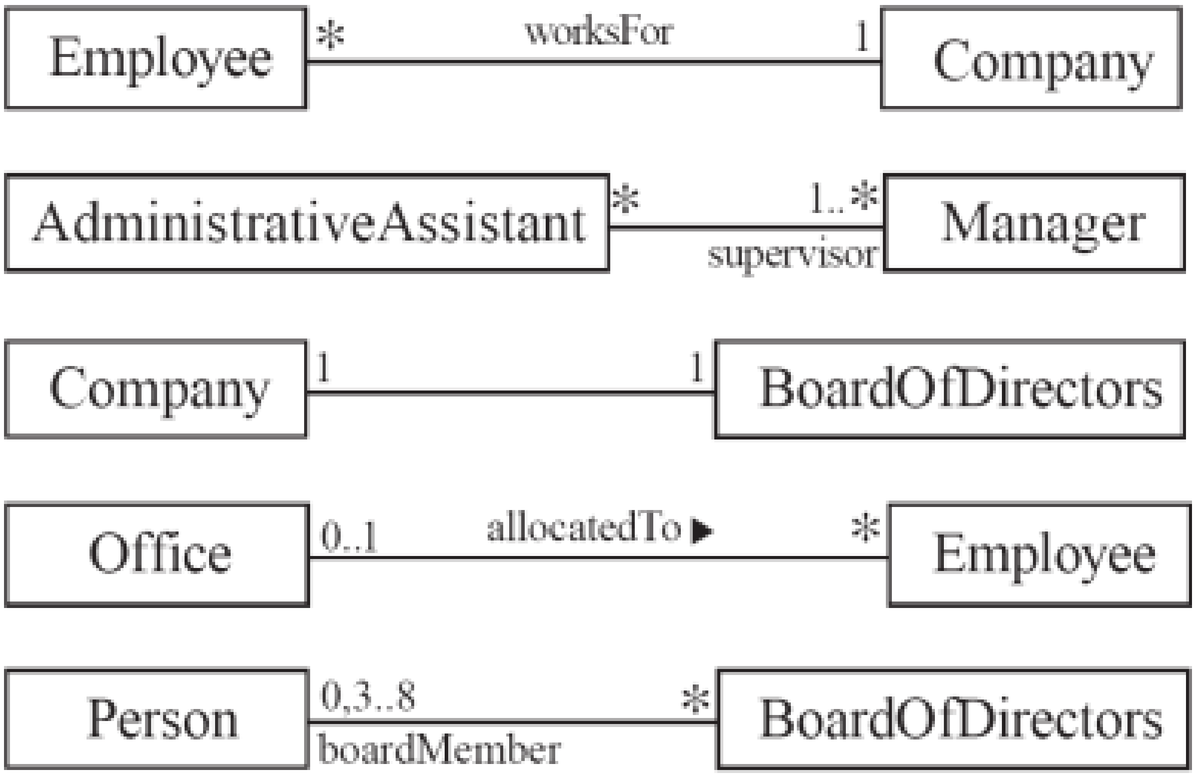
Home heating system

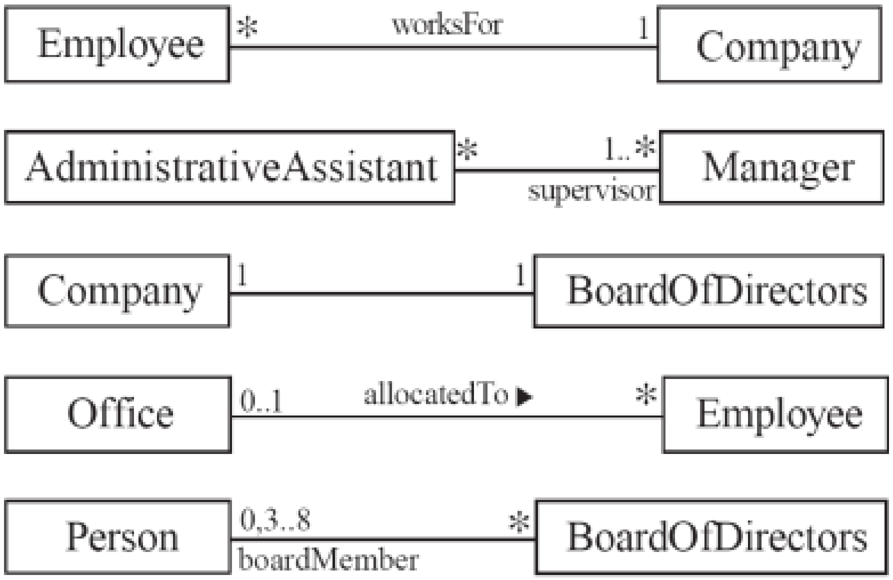


Home heating system



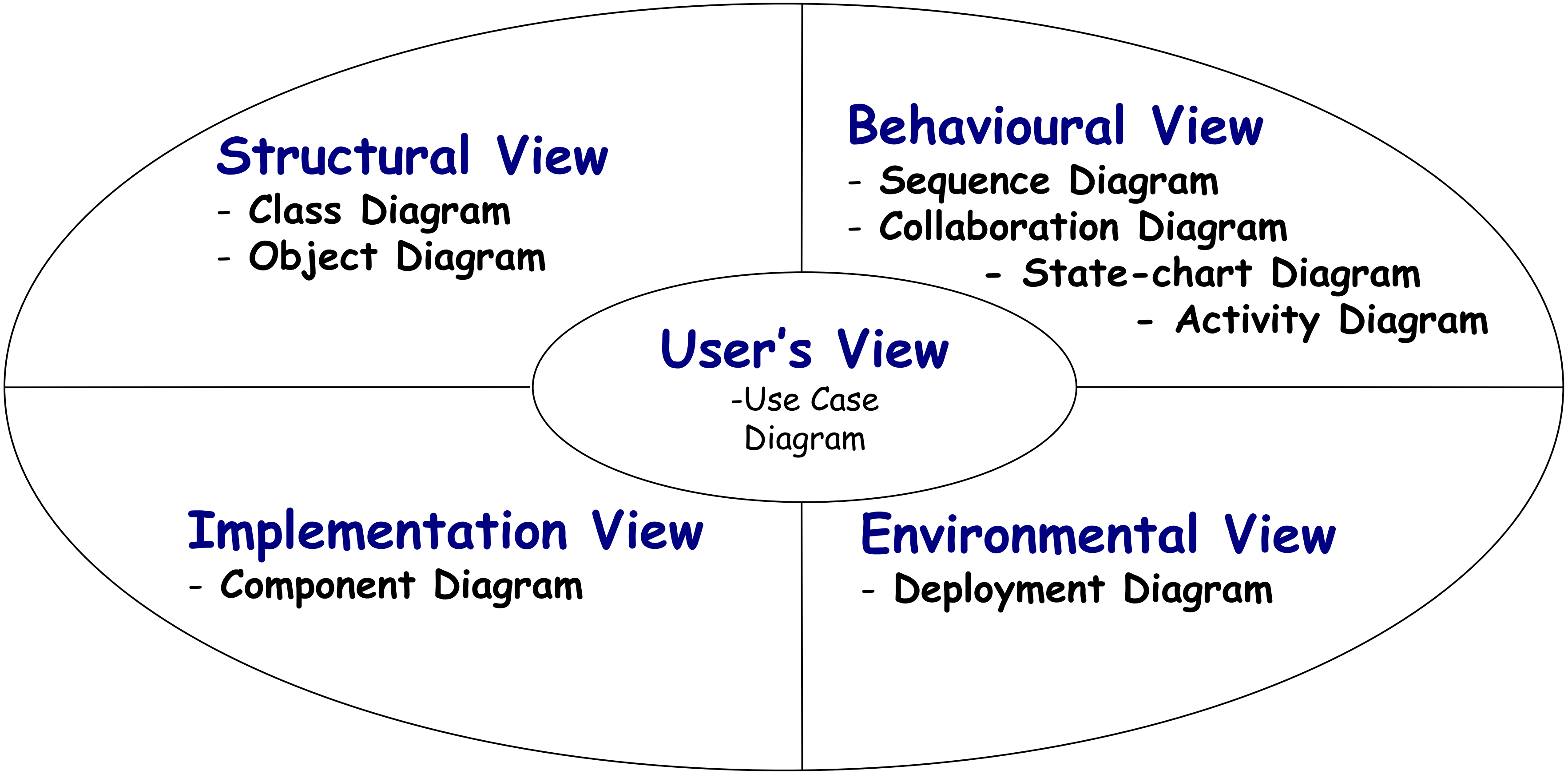
- Room has 1 Thermostat
- Each Thermostat is associated with 0 or more Heaters
- A Heather has exactly one Thermostat
- ElectricHeater is a specialized Heater
- AubeTH101D is a specialized Thermostat


Employee
*
worksFor
1
Company
AdministrativeAssistant
*
1..*
supervisor
Manager
Company
1
1
BoardOfDirectors
Office
0..1
allocatedTo ▶
*
Employee
Person
0,3..8
boardMember
*
BoardOfDirectors



- Each employee works for one company (which can have 0 employees)
- Each AdministrativeAssistant has one or more supervisors (who can have 0 or more employees)
- Each Company has exactly one BoardOfDirectors (and viceversa)
- Each Office is allocated to zero or more Employees (an Employee can have no office or at most one)
- A Person is boardMember of 0 or more BoardOfDirectors (each BoardOfDirectors has from 3 to 8 Persons)

UML Diagrams



Diagrams and views in UML

Interaction Diagrams

- Class diagram represent static structure of the system (classes and their rel)
- Do not model the *behavior of system*

- **Behavioral view** – shows how objects interact for performing actions (*typically* ***a use case***)
- Interaction is between objects, not classes

- Interaction diagram in two styles
  - Sequence diagram
  - Collaboration diagram
- Two are equivalent in power

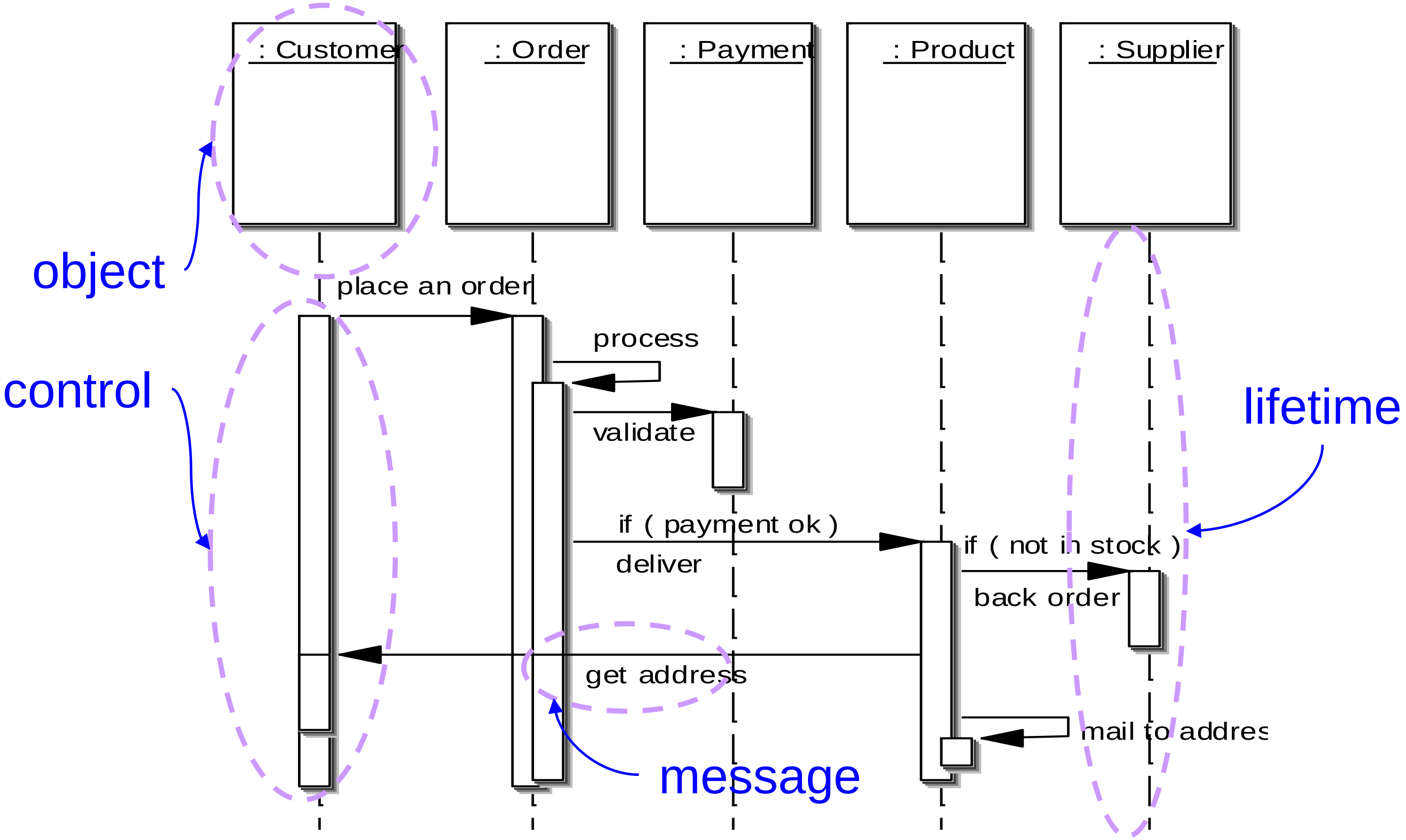
Interaction Diagram

- Typically **each interaction diagram** realizes behaviour of a **single use case**

Sequence Diagram

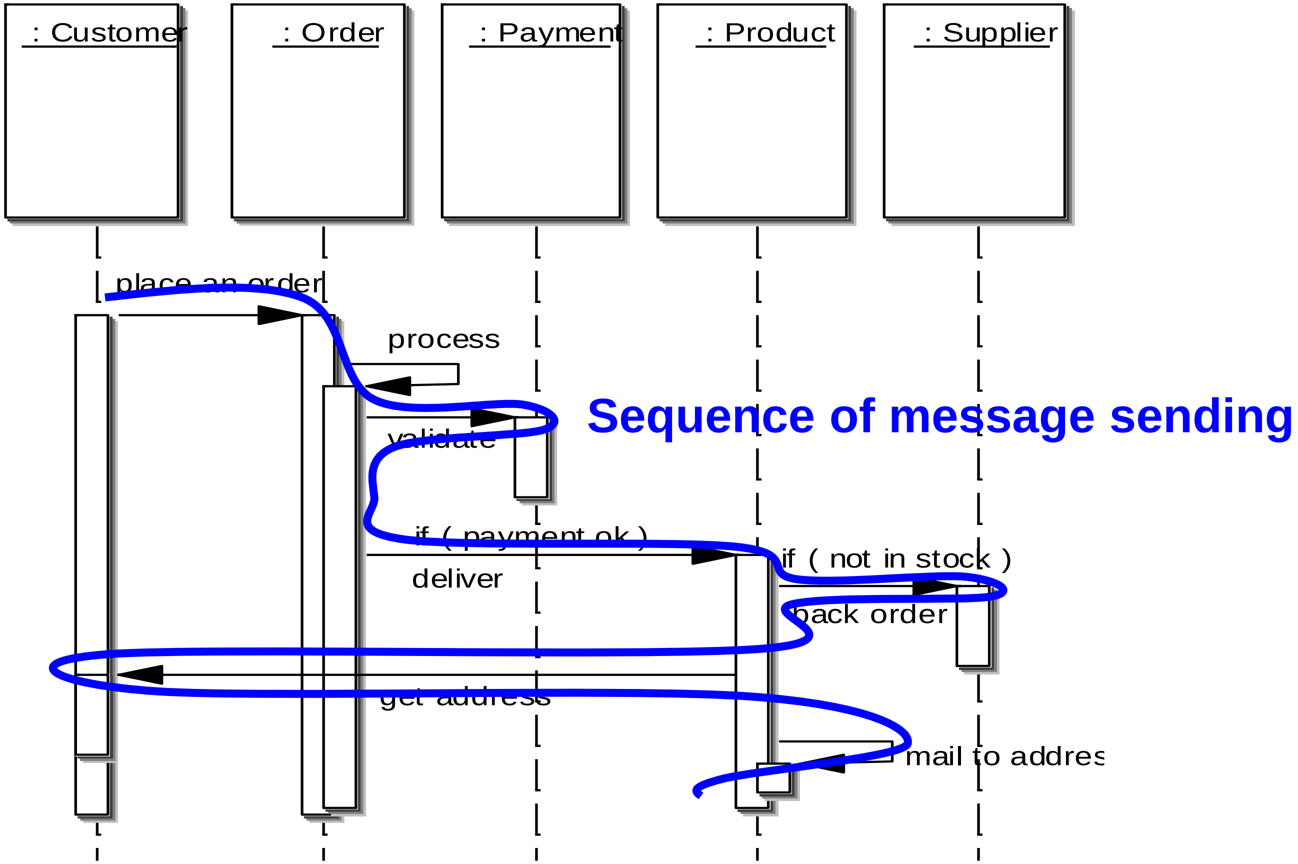
- Objects participating in an interaction are shown at the top
- For each object a vertical bar represents its lifeline
- Message from an object to another, represented as a labeled arrow
- If message sent under some condition, it can be specified in bracket
- Time increases downwards, **ordering of events is captured**

Elements of a Sequence Diagram

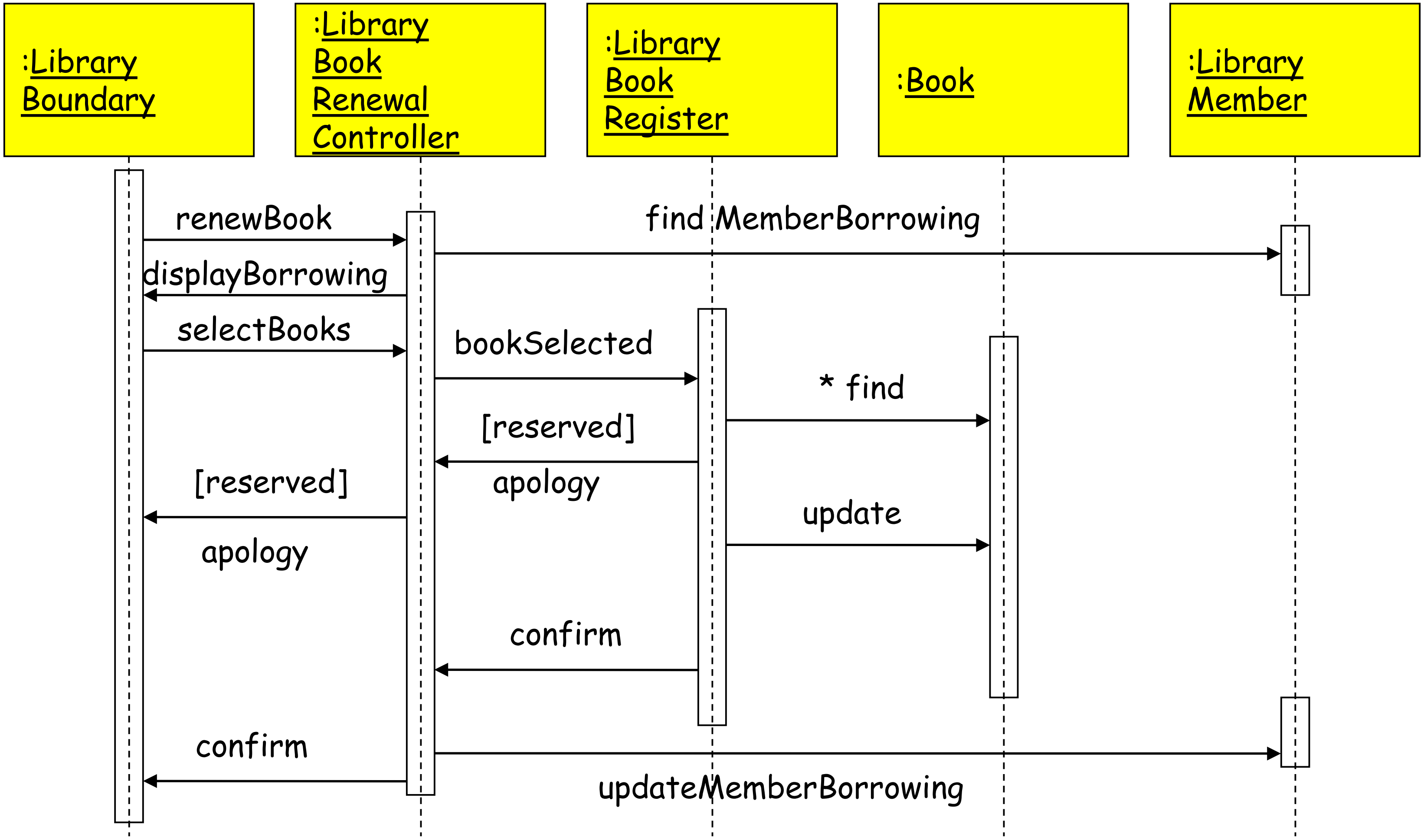


Example

Cont…



An Example of A Sequence Diagram

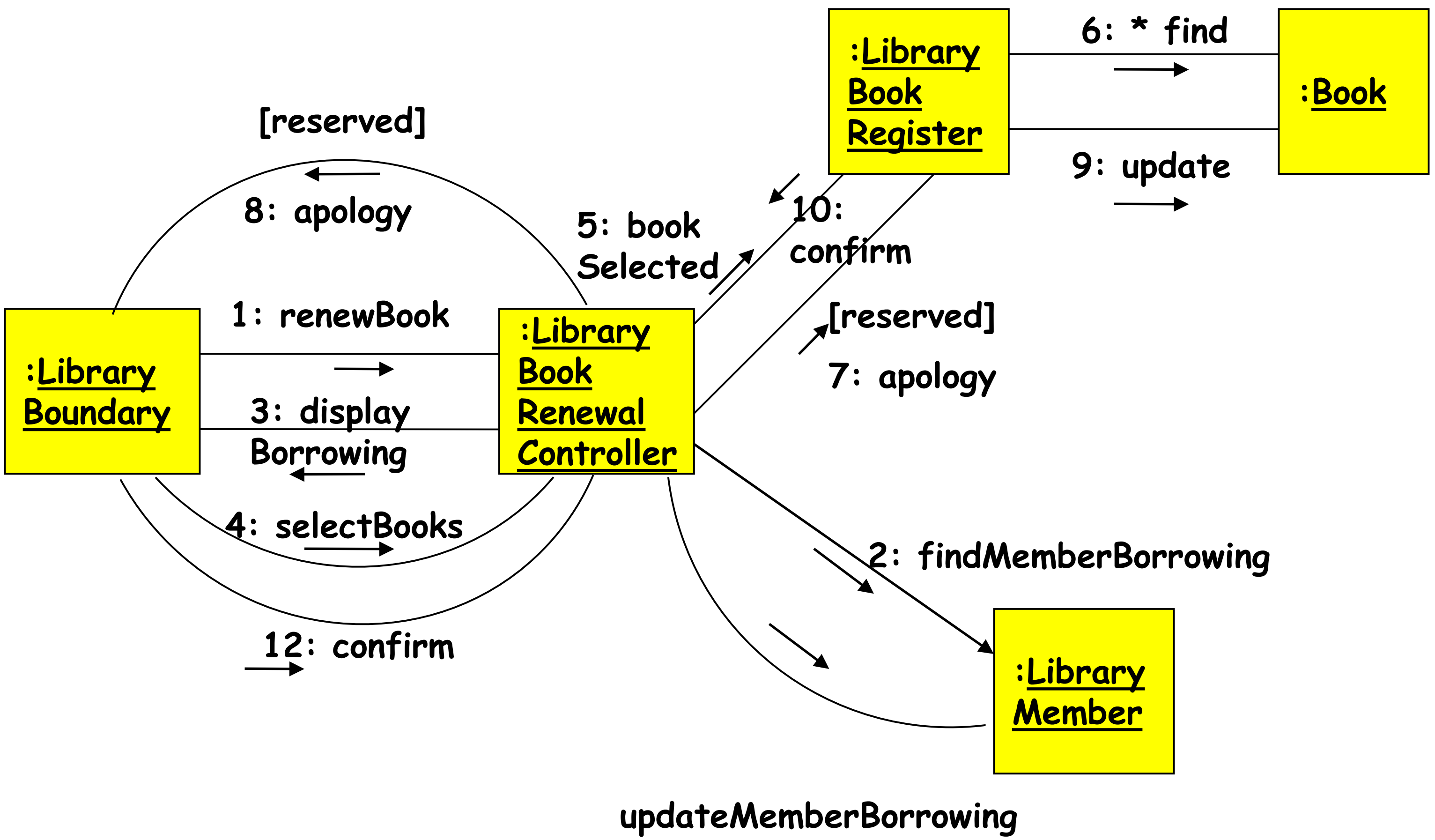


Sequence Diagram for the renew book use case

Collaboration Diagram

- Objects are collaborator, shown as boxes
- Messages between objects shown as a solid line
- A message is shown as a labelled arrow placed near the link
- Messages are prefixed with sequence numbers to show relative sequencing

An Example of A Collaboration Diagram



Collaboration Diagram for the renew book use case

Activity Diagram

- Not present in earlier modelling techniques
- Represents processing activity, **may not correspond to methods**
- Somewhat related to flowcharts

Activity Diagram vs Flow Chart

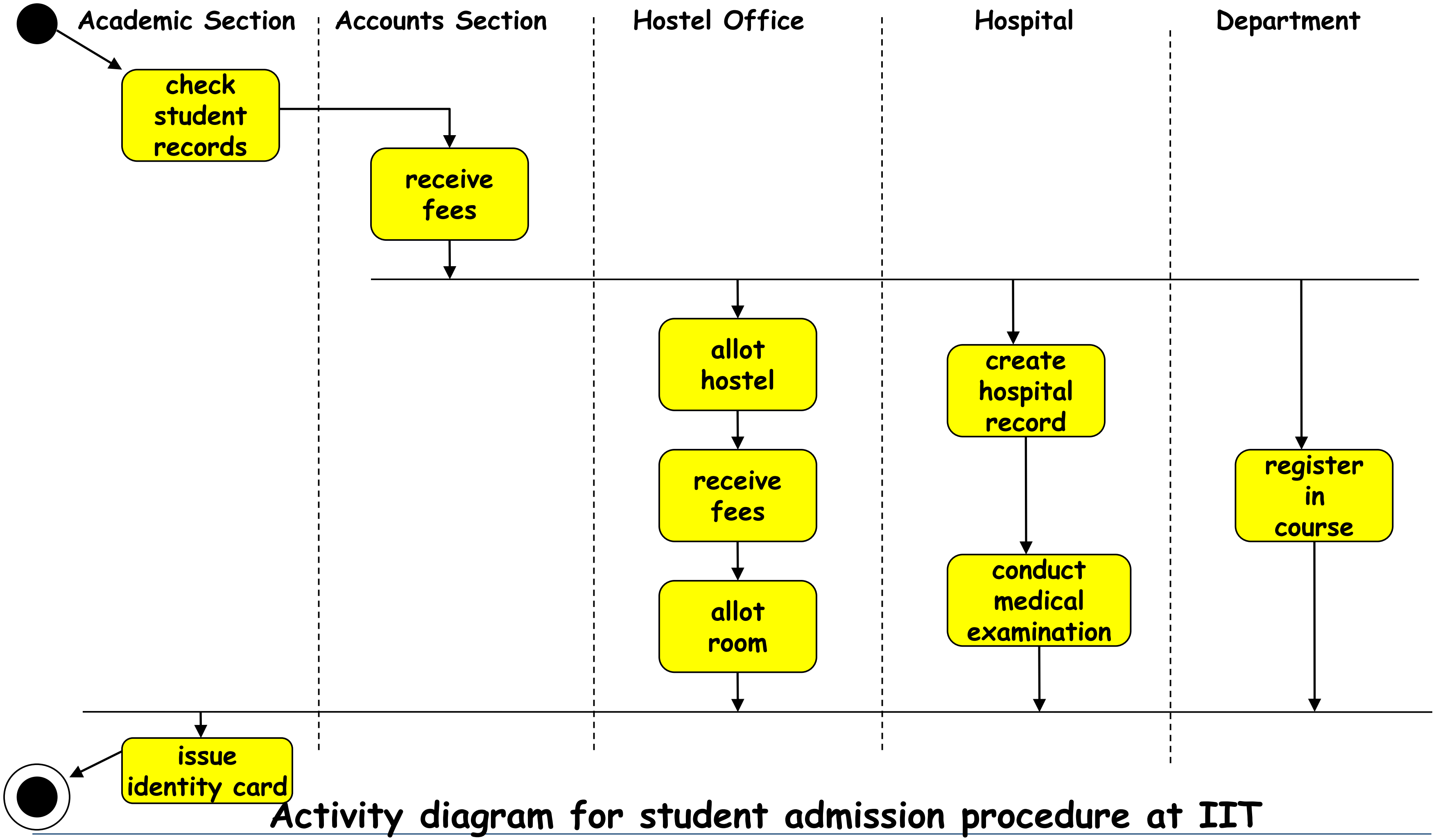
- **Swim lanes**
  - Can represent parallel activity and synchronization aspects
  - can be used to group activities based on who is performing them

- **Example**: academic department vs. hostel

Activity Diagram

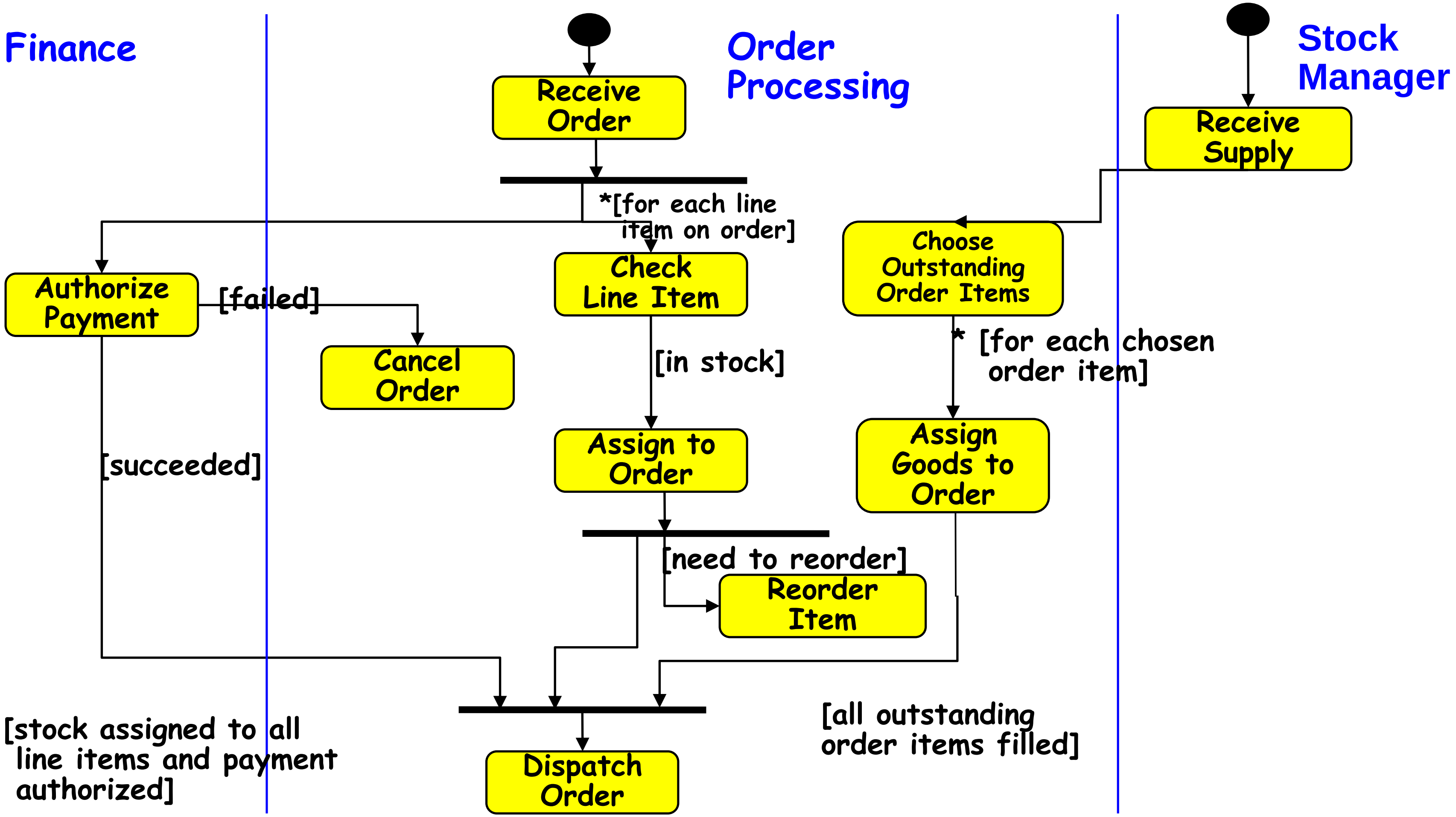
- Normally employed in business process modelling.
- Carried out **during analysis stage**.
- Can be used to develop interaction diagrams.

An Example of An Activity Diagram



Activity diagram for student admission procedure at IIT

Activity Diagram: Example 2



State Chart Diagram

- Based on the work of **David Harel** [1990]
- Model **how the state of an object changes** in its lifetime
- Based on **finite state machine** (FSM) formalism

State Chart Diagram

Cont…

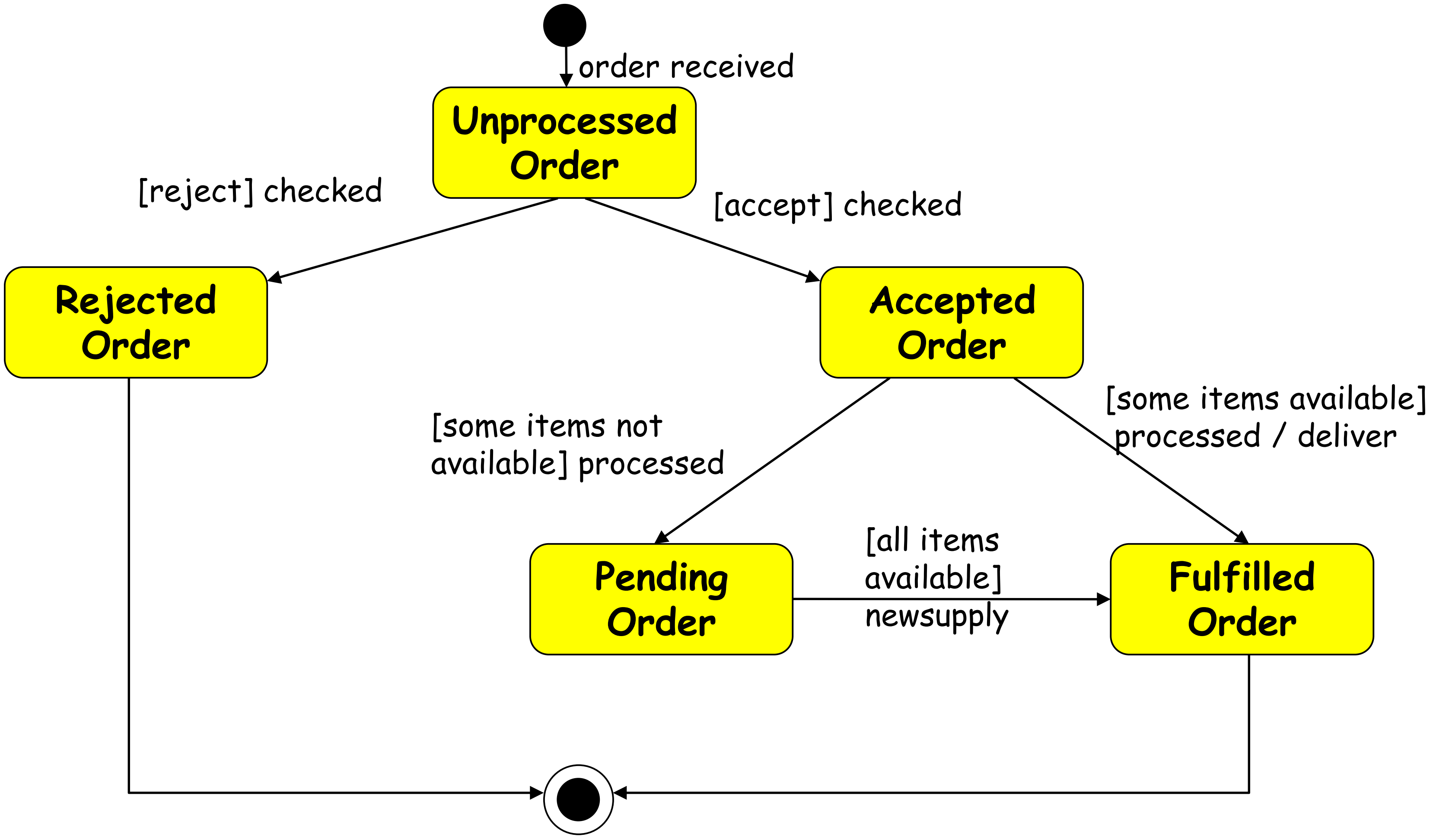
- State chart avoids the problem of state explosion of FSM.
- **Hierarchical model** of a system:
  - Represents **composite** **nested** **states**

State Chart Diagram

Cont…

- Elements of state chart diagram
  - Initial State: A filled circle
  - Final State: A filled circle inside a larger circle
  - State: Rectangle with rounded corners
  - Transitions: Arrow between states, also boolean logic condition (guard)

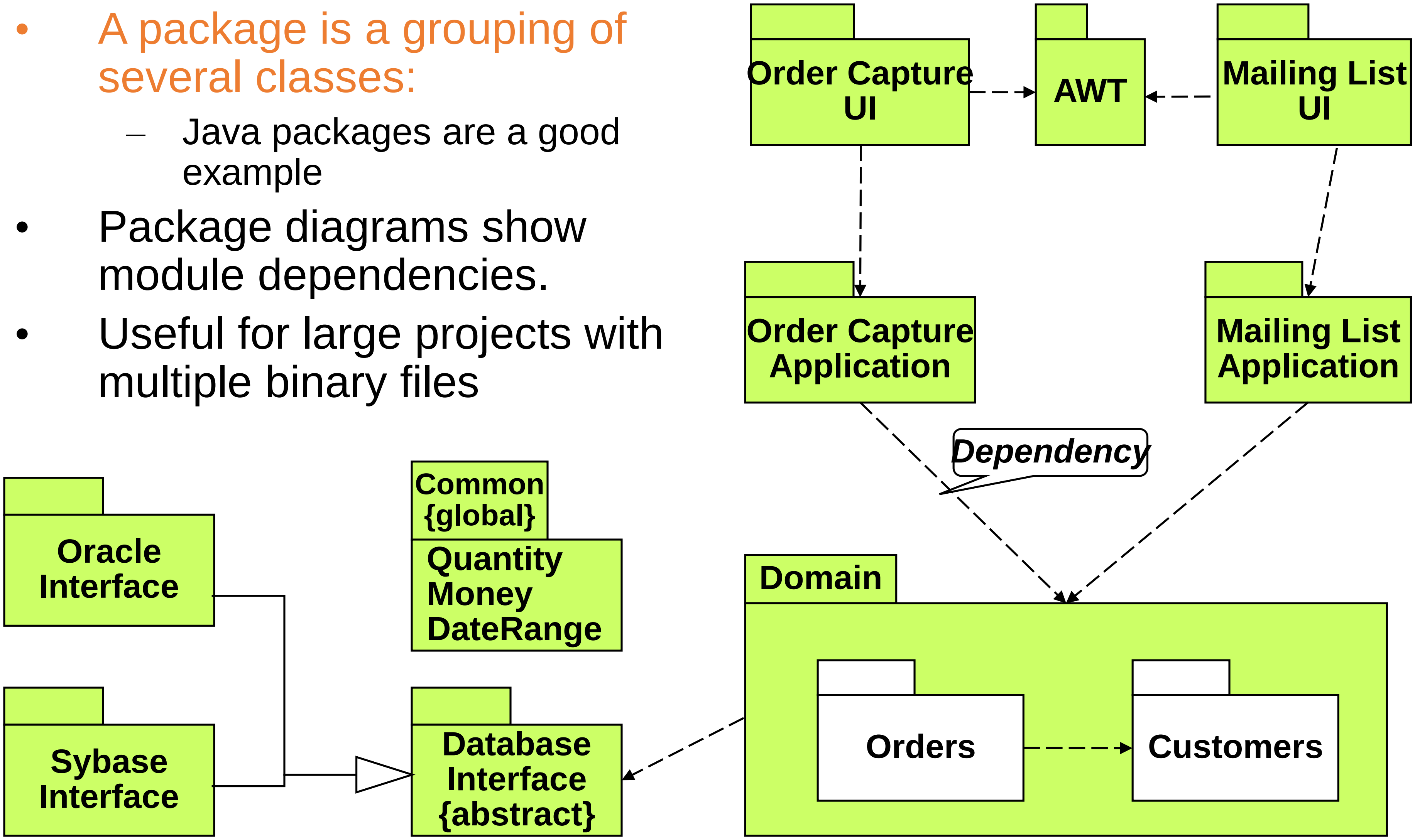
An Example of A State Chart Diagram



Example: State chart diagram for an order object

Package Diagrams

- A package is a grouping of several classes:
  - Java packages are a good example
- Package diagrams show module dependencies.
- Useful for large projects with multiple binary files

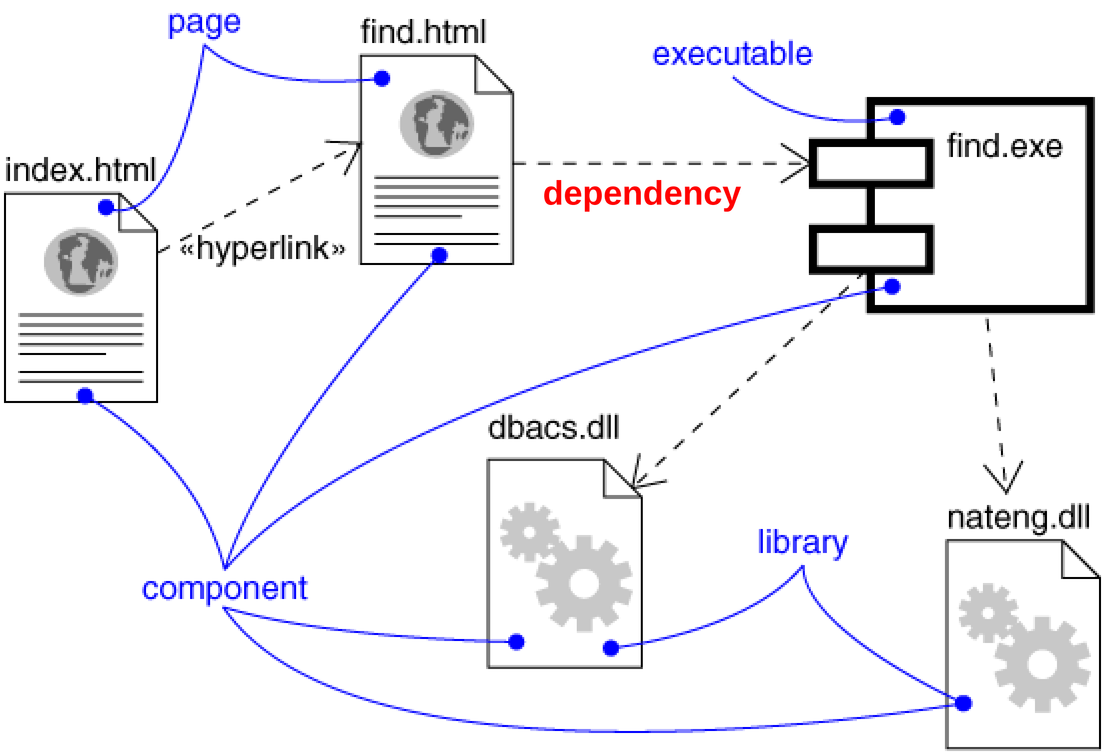


Component Diagram

- Describe the **physical artifacts of a system**

Components:

- **Executables**
- **Library**
- **Table**
- **File**
- **Document**

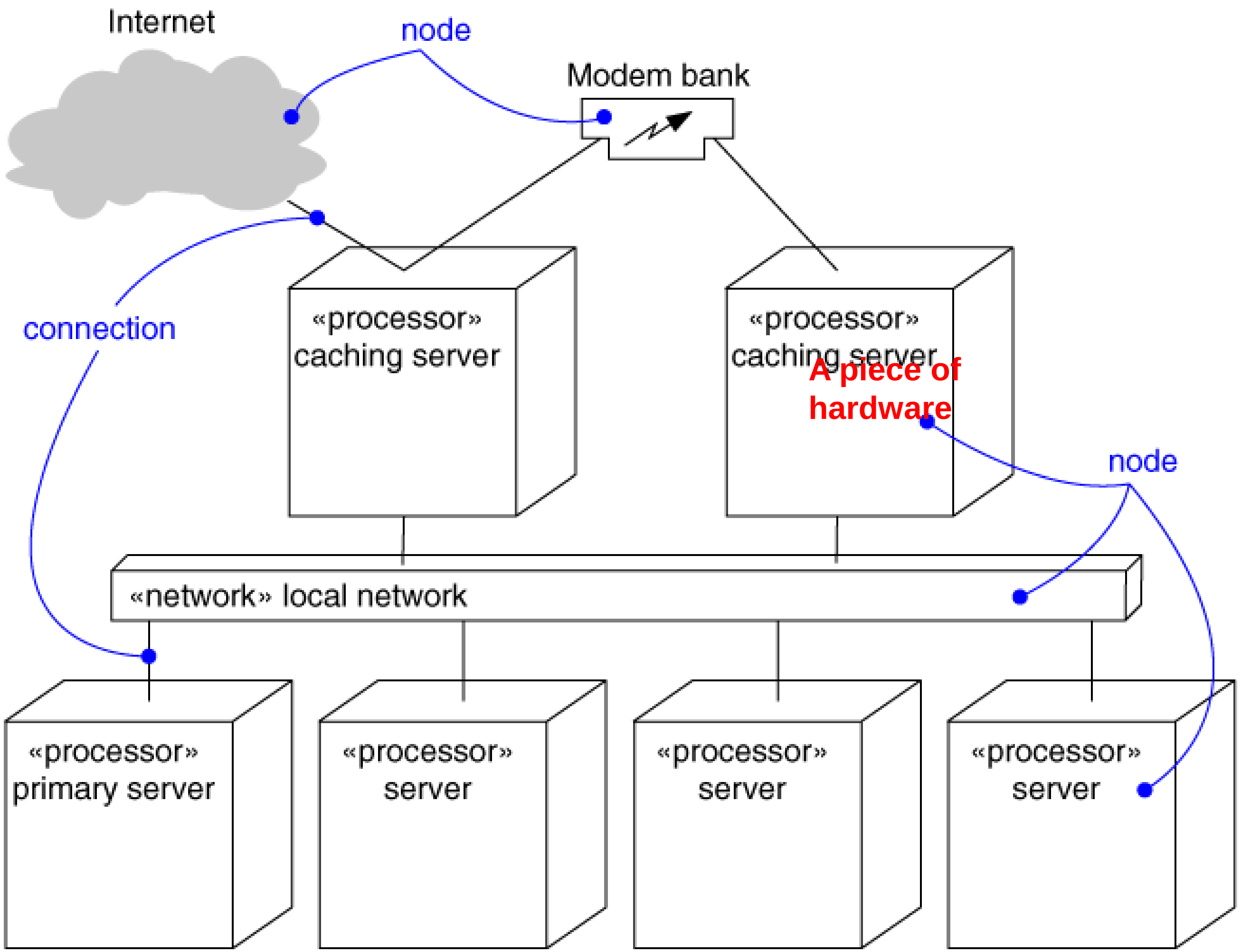


Component Diagram

- Captures the **physical structure** of the implementation
- Built as part of architectural specification
- Purpose
  - Organize source code
  - Construct an executable release
- Developed by architects and programmers

Deployment Diagram

- Captures the topology of a system's hardware

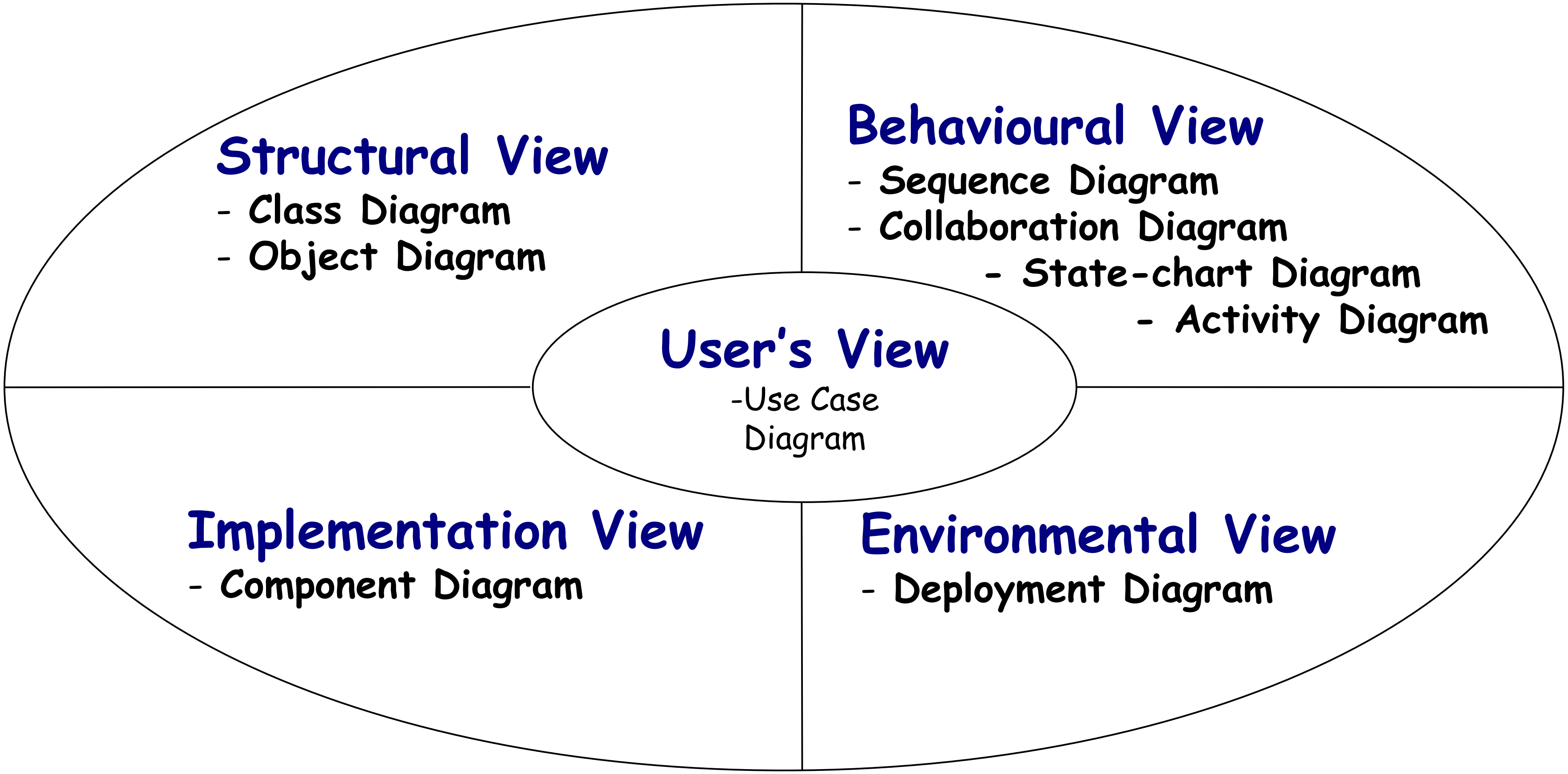


Deployment Diagram

Captures the topology of a system’s hardware

- Captures env in which the software solution is implemented.
- How a **software system will be physically deployed in the hardware environment**.
- **Which component will execute on which hardware**, how will they communicate etc.
- How diff components are distributed over diff hardware components of the system.

UML Diagrams



Diagrams and views in UML

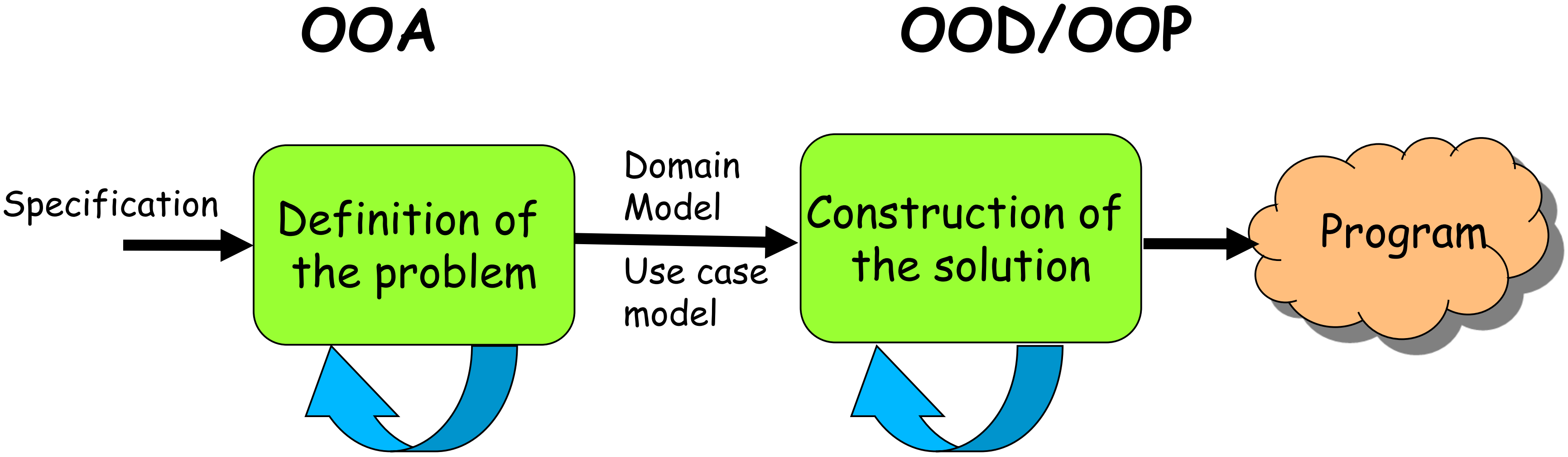
A Design Process

- From requirements specification, initial model is developed (OOA)
  - Analysis model is iteratively refined into a design model

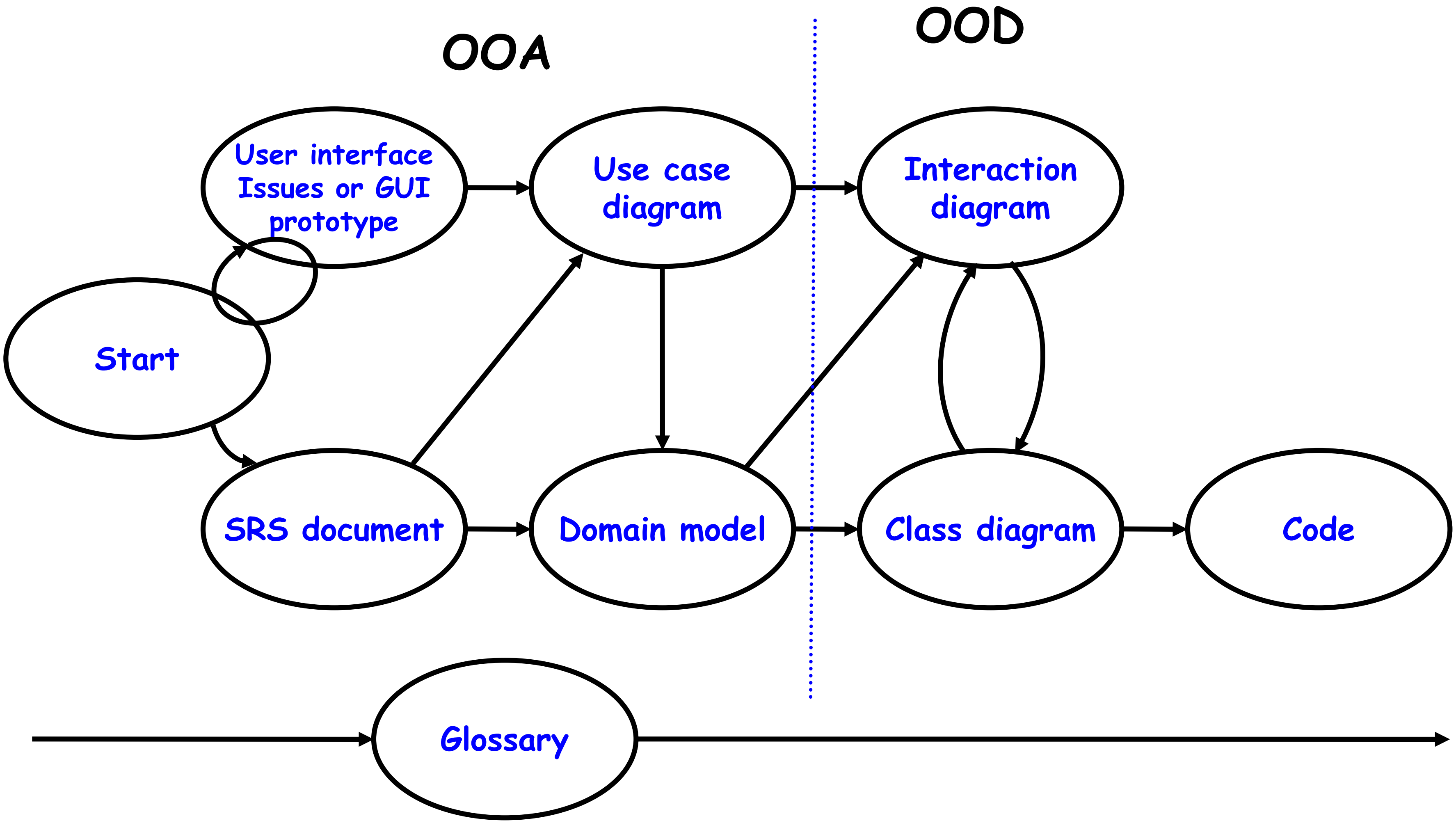
- Design model is implemented using OO concepts

OOAD

Iterative and Incremental



OOAD


OOA
OOD
User interface Issues or GUI prototype
Use case diagram
Interaction diagram
Start
SRS document
Domain model
Class diagram
Code
Glossary

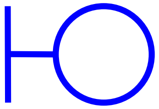
Domain Modelling

- Represents concepts or objects appearing in the problem domain.
- Also captures relationships among objects.
- Three types of objects are identified
  - Boundary objects
  - Entity objects
  - Controller objects

Class Stereotypes

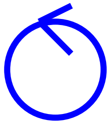
Three different stereotypes on classes are used: <<boundary>>, <<control>>, <<entity>>.

Boundary



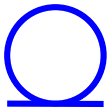
Cashier Interface

Control



Withdrawal

Entity



Account

Boundary Objects

- Interact with actors:
  - User interface objects
- Include screens, menus, forms, dialogs etc.
- Do not perform processing but validates, formats etc.

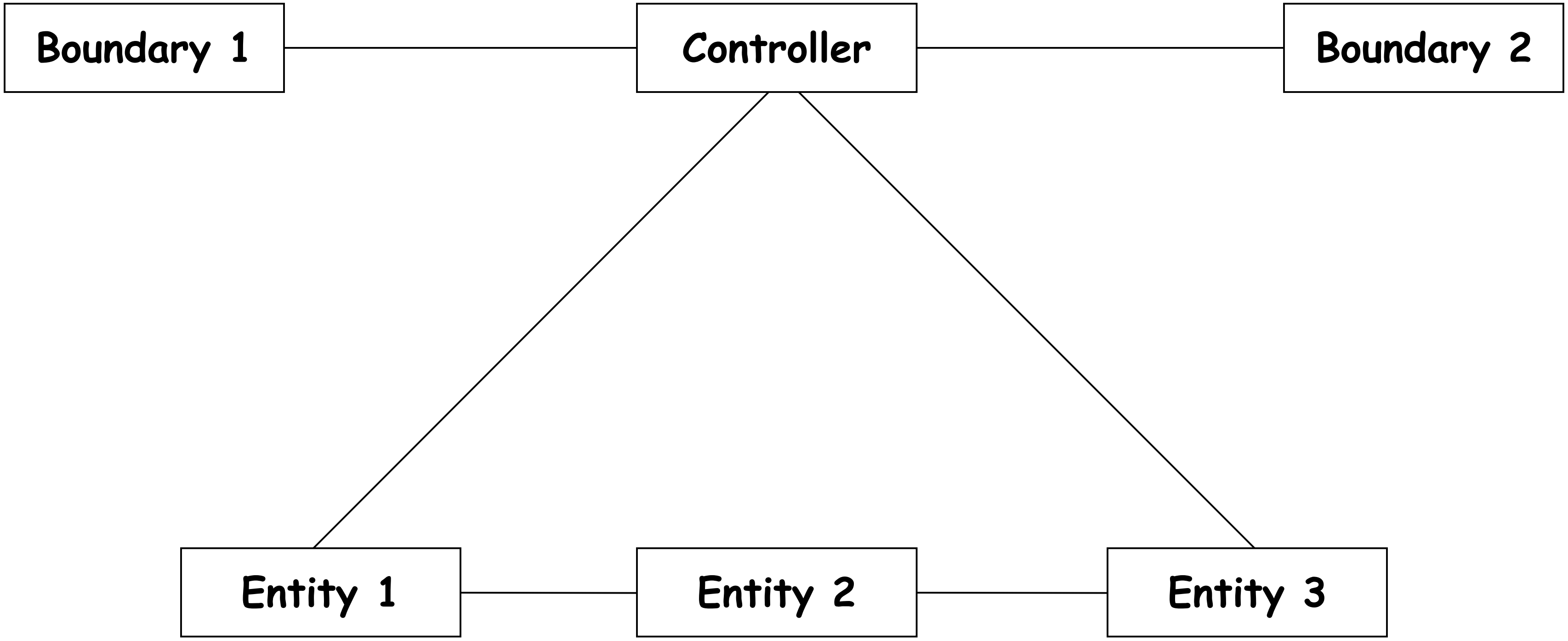
Entity Objects

- Hold information:
  - Such as data tables & files, e.g. Book, BookRegister
- Responsible for storing data, fetching data etc.
- Elementary operations on data such as searching, sorting, etc.
- Entity Objects are identified by examining nouns in problem description

Controller Objects

- Coordinate the activities of a set of entity objects
- Interface with the boundary objects
- Realizes use case behavior
- Embody most of the logic involved with the use case realization
- There can be more than one controller to realize a single use case

Use Case Realization



Realization of use case through the collaboration of Boundary, controller and entity objects

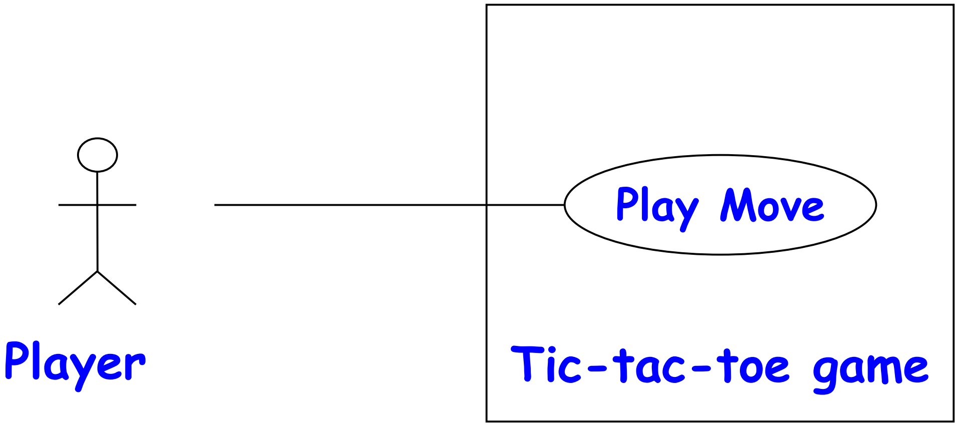
Example 1: Tic-Tac-Toe Computer Game

- 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.

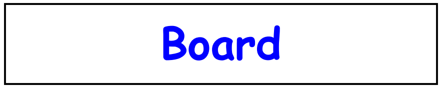
Example 1: Tic-Tac-Toe Computer Game

- 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: Initial and Refined Domain Model



Board

Initial domain model

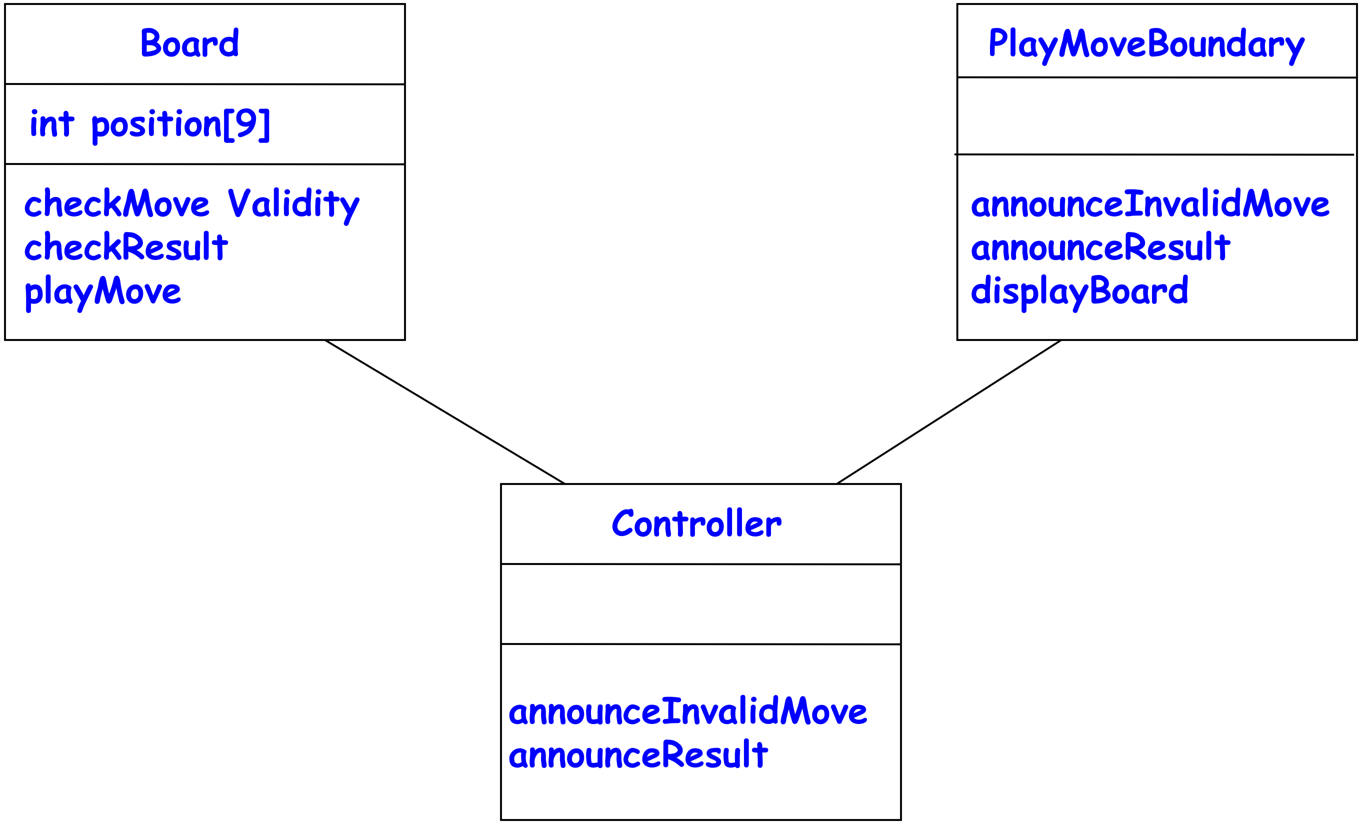
PlayMoveBoundary

PlayMoveController

Board

Refined domain model

Example 1: Class Diagram



Board

int position[9]

checkMove Validity
checkResult
playMove

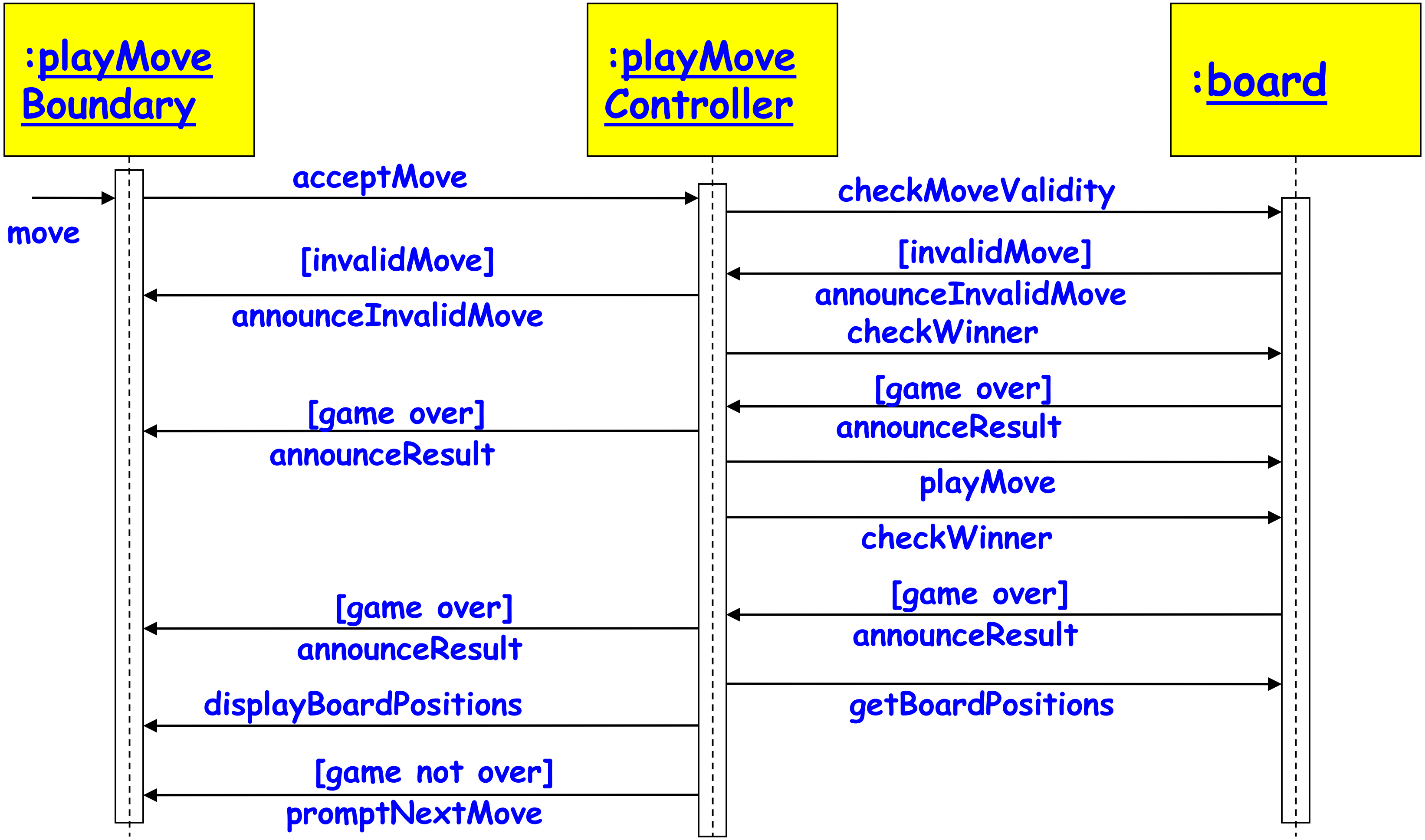
PlayMoveBoundary

announceInvalidMove
announceResult
displayBoard

Controller

announceInvalidMove
announceResult

Example 1: Sequence Diagram



Sequence Diagram for the play move use case

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.

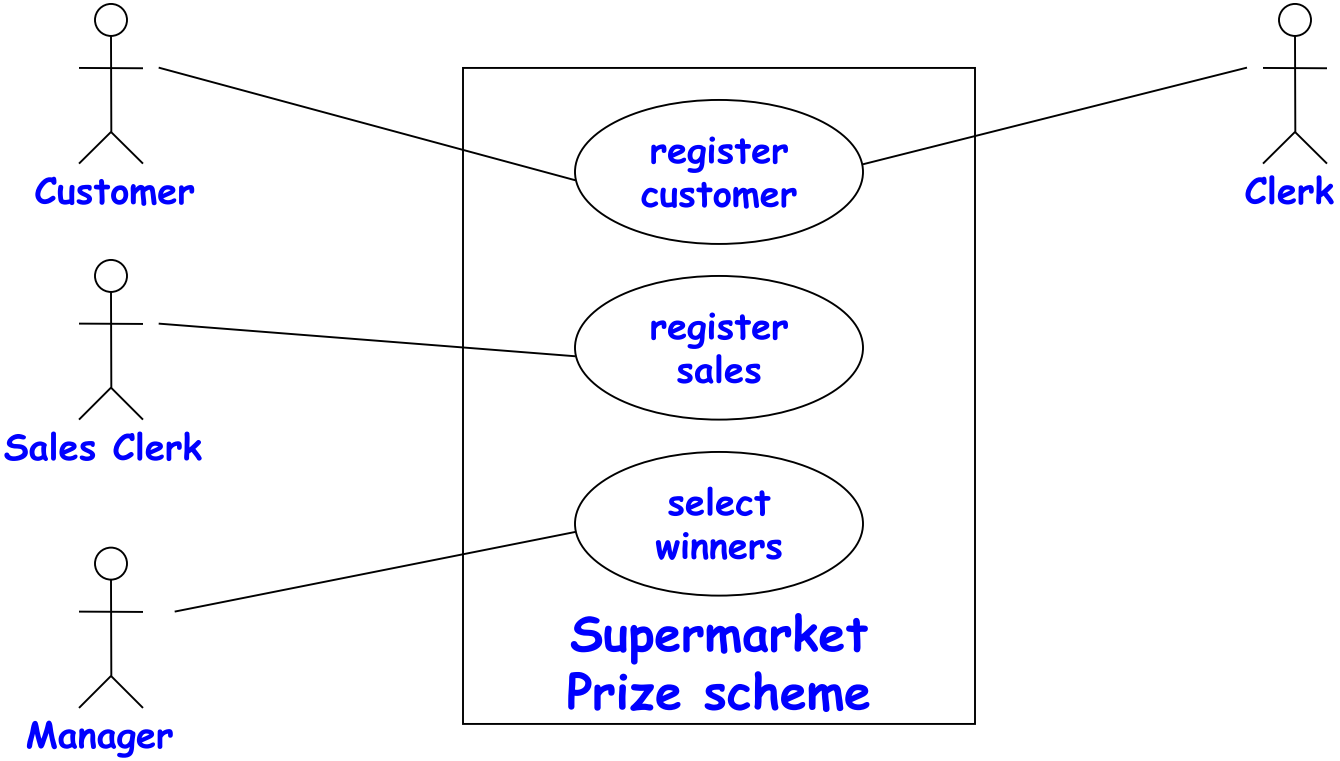
Example 2: Supermarket Prize Scheme

- 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.

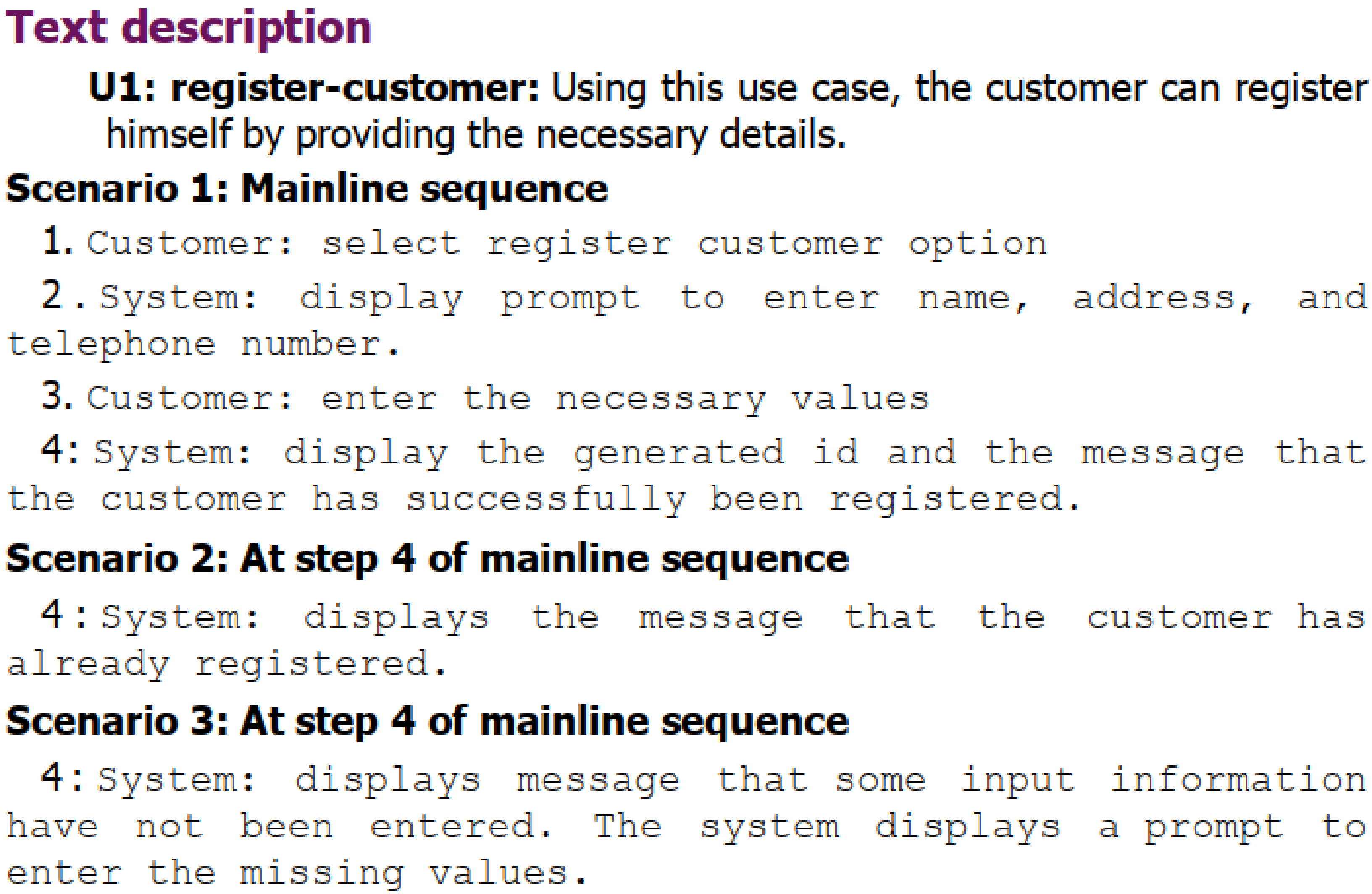
Example 2: Supermarket Prize Scheme

- Also, it awards a gold coin to every customer:
  - Whose purchases exceed Rs. 500,000.
- 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



Use case description- Register Customer



Text description

U1: register-customer: Using this use case, the customer can register himself by providing the necessary details.

Scenario 1: Mainline sequence

1. Customer: select register customer option

2. System: display prompt to enter name, address, and telephone number.

3. Customer: enter the necessary values

4: System: display the generated id and the message that the customer has successfully been registered.

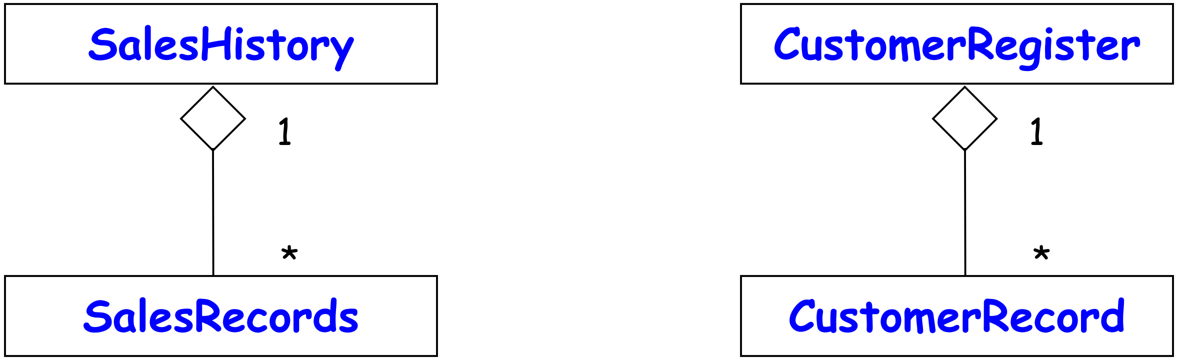
Scenario 2: At step 4 of mainline sequence

4: System: displays the message that the customer has already registered.

Scenario 3: At step 4 of mainline sequence

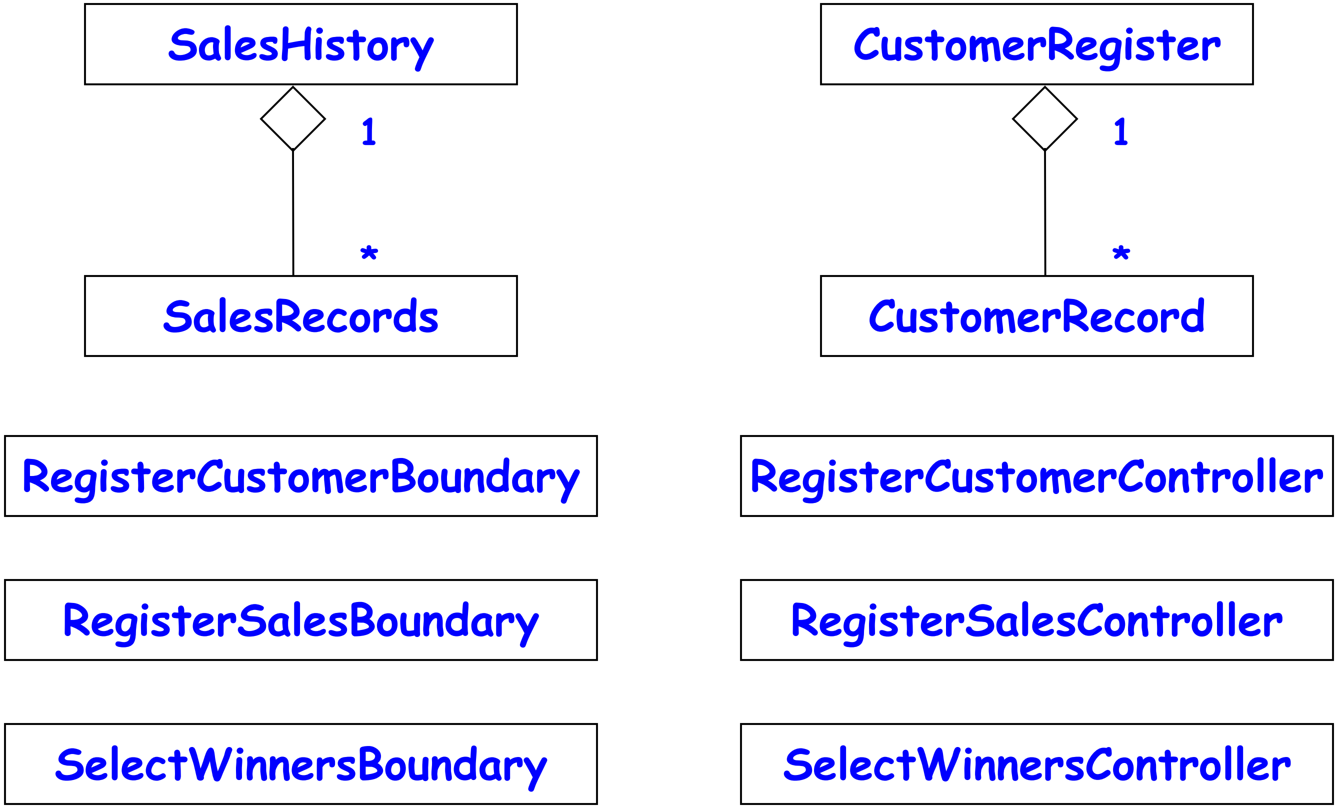
4: System: displays message that some input information have not been entered. The system displays a prompt to enter the missing values.

Example 2: Initial Domain Model



Initial domain model

Example 2: Refined Domain Model



SalesHistory

1

*

SalesRecords

CustomerRegister

1

*

CustomerRecord

RegisterCustomerBoundary

RegisterCustomerController

RegisterSalesBoundary

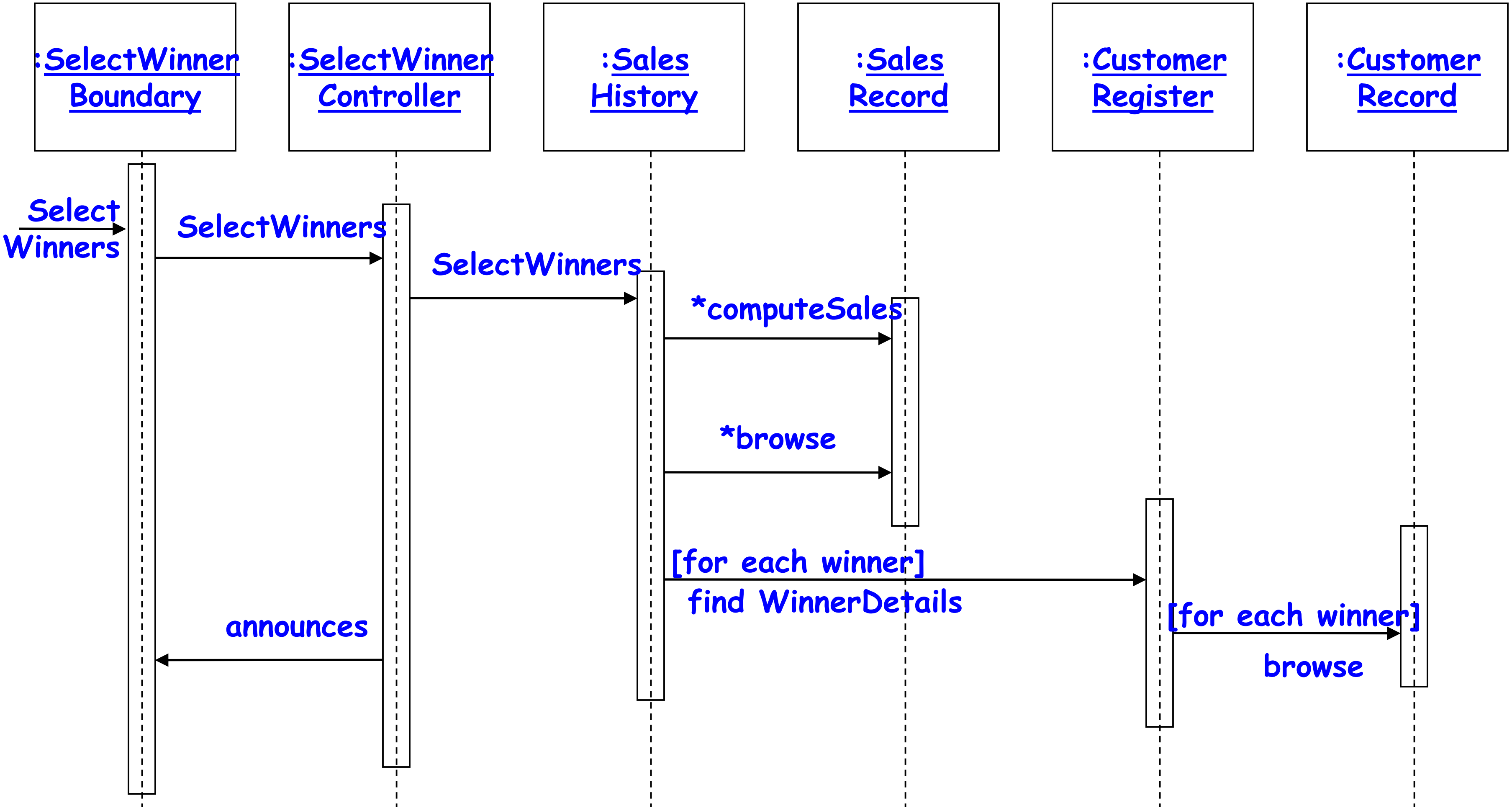
RegisterSalesController

SelectWinnersBoundary

SelectWinnersController

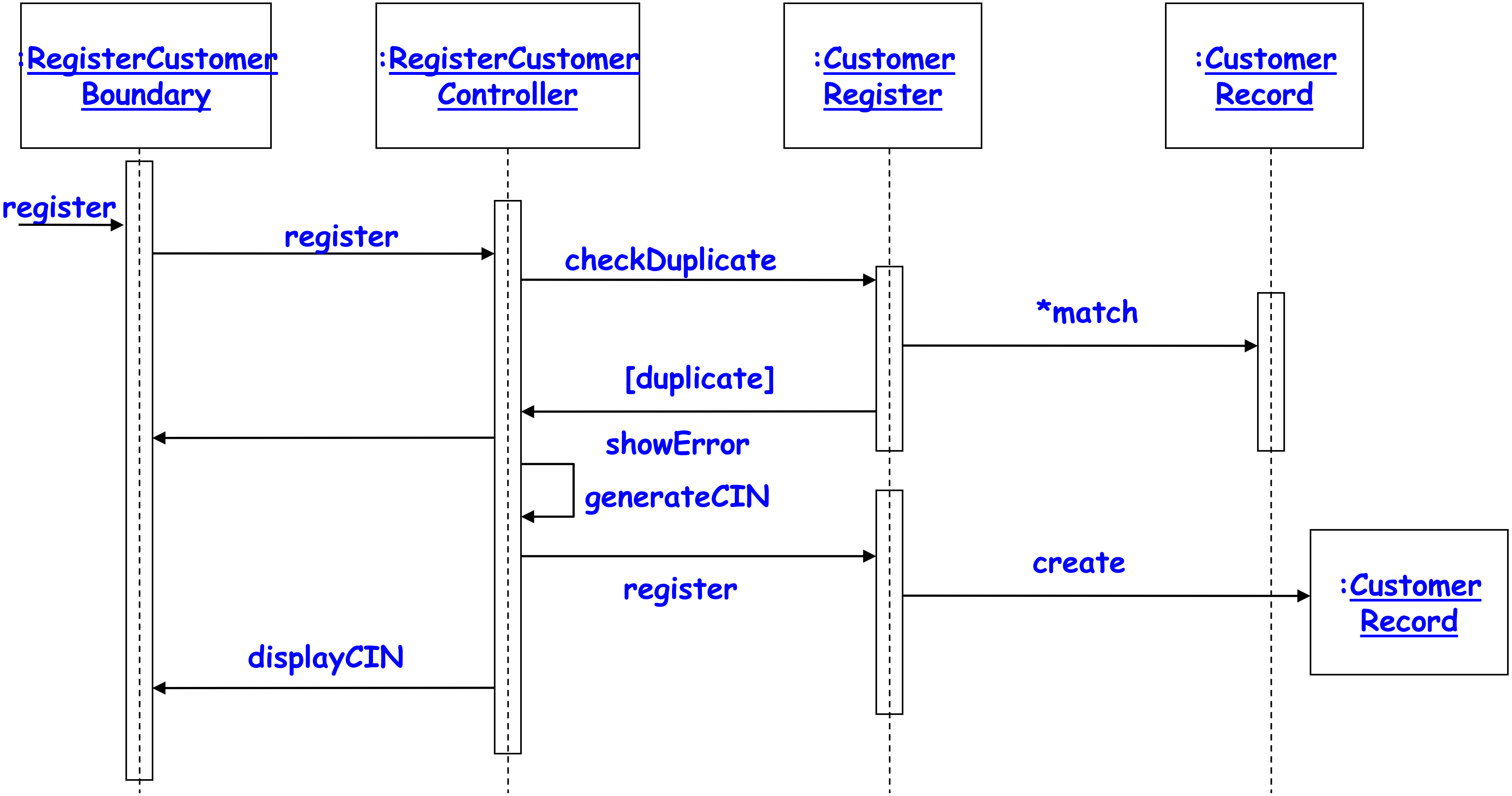
Refined domain model

Example 2: Sequence Diagram for the Select Winners Use Case



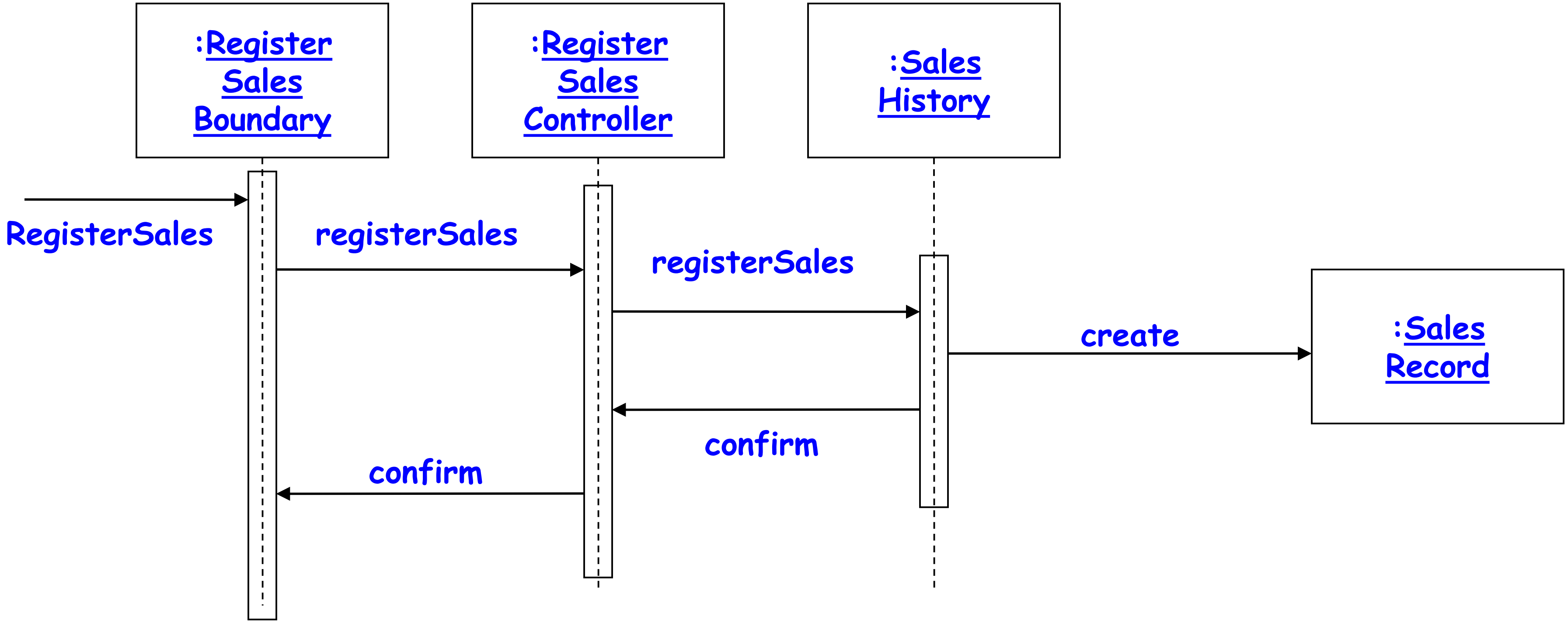
Sequence Diagram for the select winners use case

Example 2: Sequence Diagram for the Register Customer Use Case



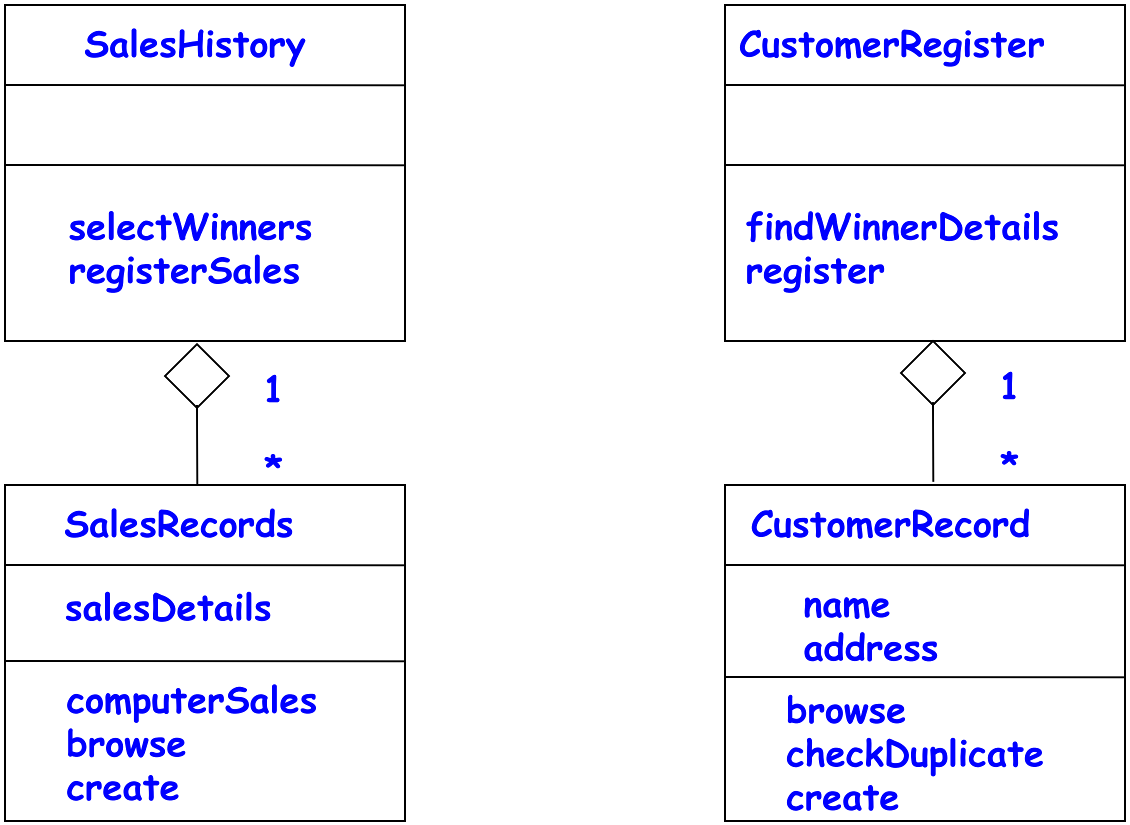
Sequence Diagram for the register customer use case

Example 2: Sequence Diagram for the Register Sales Use Case



Sequence Diagram for the register sales use case

Example 2: Class Diagram



SalesHistory

selectWinners
registerSales

1

*

SalesRecords

salesDetails

computerSales
browse
create

CustomerRegister

findWinnerDetails
register

1

*

CustomerRecord

name
address

browse
checkDuplicate
create

Summary

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

cont…

- Other UML diagrams:
  - Interaction diagrams (sequence and collaboration),
  - Activity diagrams,
  - State chart diagrams.
- We discussed OO software development process:
  - Use of patterns lead to increased productivity and good solutions.

Design Patterns

- Commonly accepted solutions to some problems that recur during designing different applications.

- Documented design solutions to certain problems that are *reusable* during design of different applications.

- Once a pattern is identified, we can reuse the documented pattern solution.

Patterns

- The essential idea:
  - If you can master a few important patterns, you can easily spot them in application development and use the pattern solutions.

Antipattern

- If a pattern represents a best practice:
  - Antipattern represents lessons learned from a bad design.
- Antipatterns help to recognise deceptive solutions:
  - That appear attractive at first, but turn out to be a liability later.

Example Patterns

- Creator Pattern
- Expert Pattern
- Facade Pattern
- MVC Pattern