Programming Paradigm

Note : These slides are not study material, rather just a guide to study these topics. Students are suggested to go through books and refer class notes to understand these topics in detail.

Software Design

Object Modeling using Unified Modeling Language (UML)

What is a Model ?

A model is an abstraction of a real problem (or situation), and is constructed by leaving out unnecessary details.

- This reduces the problem complexity and makes it easy to understand the problem

Need of a Model ?

An important reason behind constructing a model is that it helps manage complexity.

Once models of a system have been constructed, these can be used for a variety of purposes during software development, including the following :

- Analysis
- Specification
- Code generation
- Design
- Visualize and understand the problem and the working of a system
- Testing, etc.

Need of a Model ? Contd…

Since a model can be used for a variety of purposes, it is expected that the model would vary depending on the purpose for which it is being constructed.

For example :- A model developed for initial analysis and specification should be very different from the one used for design

So it is a good idea to explicitly mention the purpose for which a model has been developed, along with the model.

Unified Modeling Language (UML)

UML, is a modelling language that may be used to visualize, specify, construct, and document the artifacts of a software system.

- Not a system design or development methodology

It provides a set of notations (e.g. rectangles, lines, ellipses, etc.) to create a visual model of the system – such that it has its own syntax (symbols or sentences) and semantics (meanings of symbols and sentences).

Used to document object-oriented analysis and design results. Independent of any specific design methodology.

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:

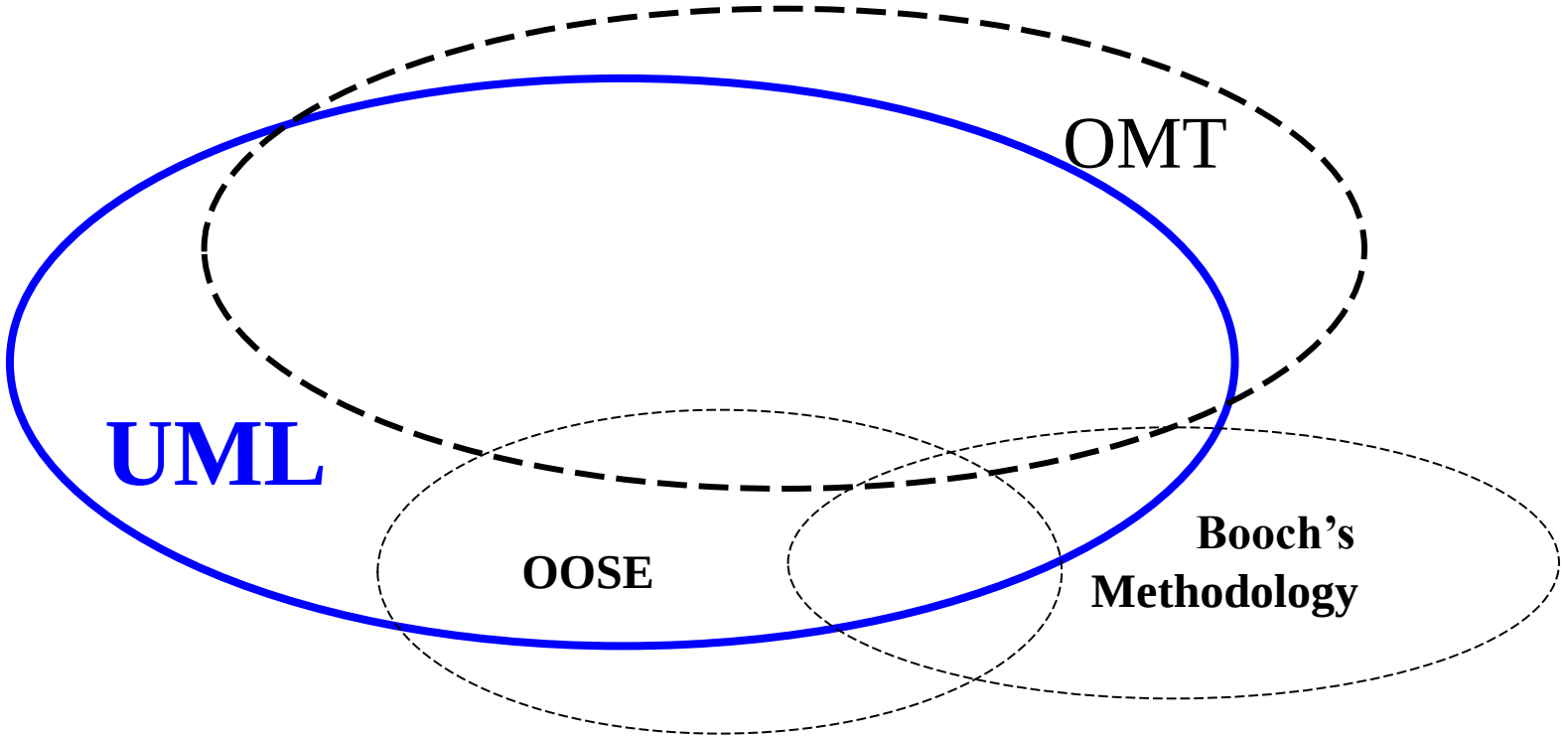
- Standardize the large number of object-oriented modeling notations

UML Origin Contd

Based on :

- Object Management Technology (OMT) [Rumbaugh 1991]
- Booch's methodology[Booch 1991]
- Object-Oriented Software Engineering (OOSE) [Jacobson 1992]
- Odell's methodology[Odell 1992]

Different Object Modeling Techniques in UML



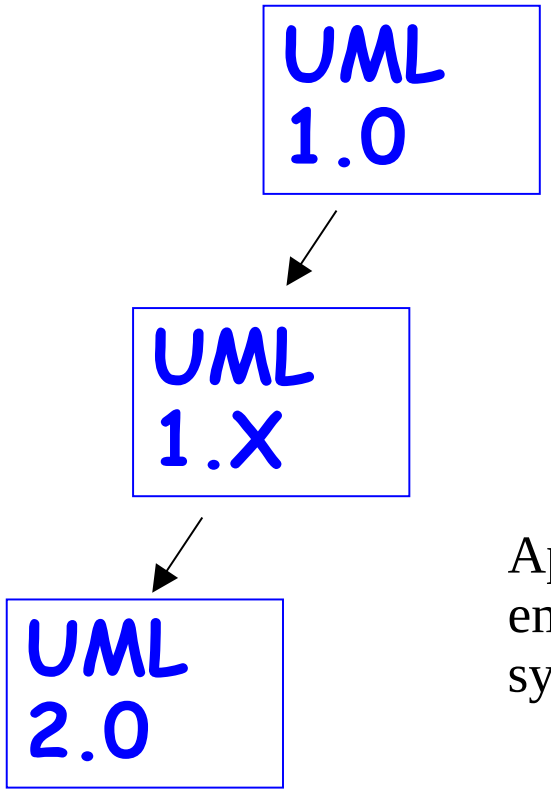
UML as a Standard

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

Developments to UML

UML continues to develop :

- Refinements
- Making it applicable to new contexts



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 modeling tool
- Easy to understand and construct

UML Diagrams

Nine diagrams are used to capture **Five** different **views** of a system.

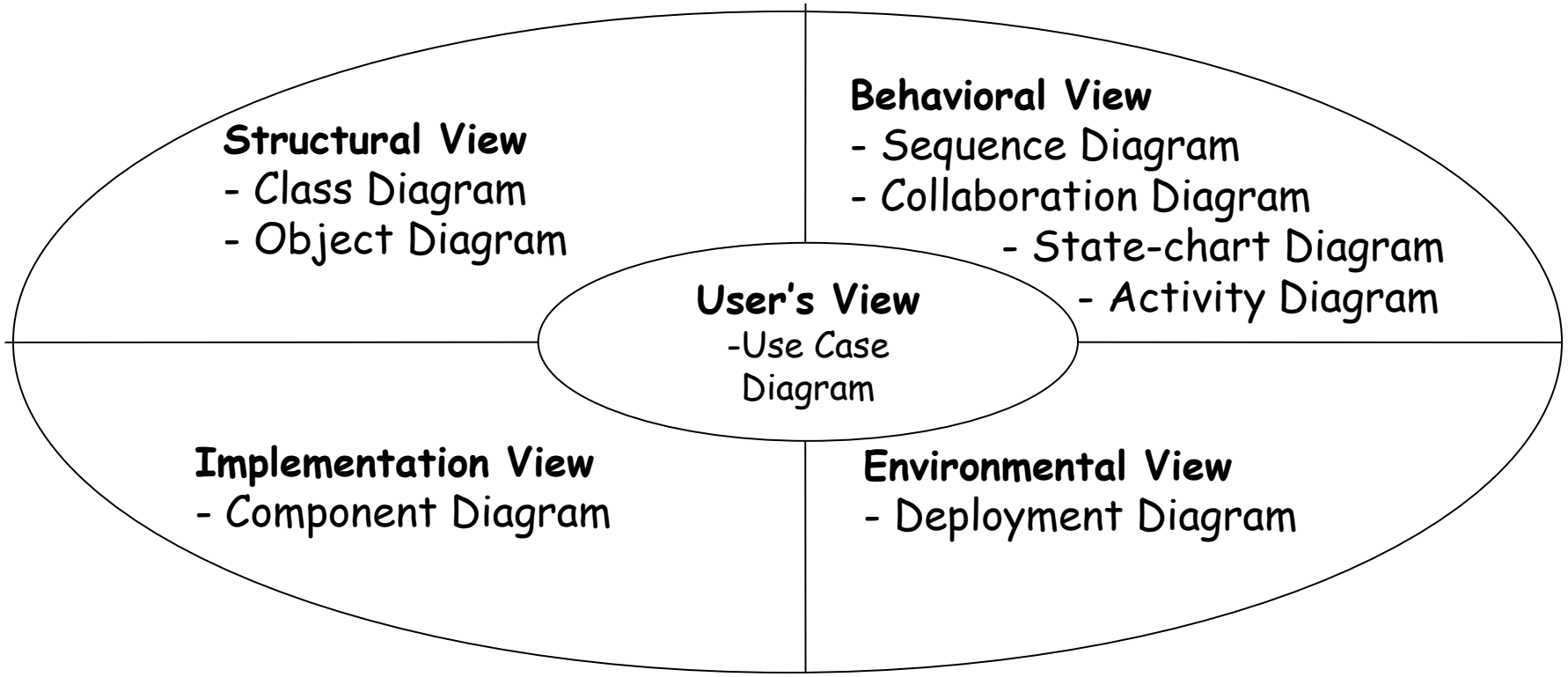
- **Views** provide different perspectives of a software system.
- **Diagrams** can be refined to get the actual implementation of a system.

UML Model Views

Views of a system:

- User's view
- Structural view
- Behavioral view
- Implementation view
- Environmental view

UML Diagrams



Diagrams and views in UML

User's view

- It defines the functionalities (facilities) made available by the system to its users
  - Captures the external users' view of the system in terms of the functionalities offered by the system.

- Is a black-box view of the system where the internal structure, the dynamic behavior of different system components, the implementation etc. are not visible.

- Is very different from all other views in the sense that it is a functional model compared to the object model of all other views.

- Can be considered as the central view and all other views are expected to conform to this view.

Structural view

- Defines the kinds of objects (classes) important to the understanding of the working of a system and to its implementation.
- Also captures the relationships among the classes (objects).
- The structural model is also called the static model, since the structure of a system does not change with time.

Other views

Behavioral view

- Captures how objects interact with each other to realize the system behavior.
- The system behavior captures the time-dependent (dynamic) behavior of the system.

Implementation view

- Captures the important components of the system and their dependencies.

Environmental view

- Models how the different components are implemented on different pieces of hardware.

Are all views required for developing a typical system?

Answer is NO

- Use case diagram, class diagram and one of the interaction diagram for a simple system
- State chart diagram required to be developed when a class state changes
- However, 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 View

Use Case Diagram

Use Case Model

- Consists of set of “use cases” - such as list of steps
- 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
- Set of related scenarios tied together by a common goal.

Use Cases Contd...

- 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

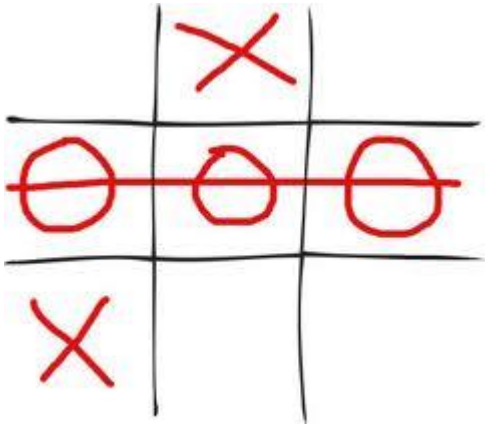
For library information system

- Issue-book
- Query-book
- Return-book
- Create-member
- Add-book, etc.

Representation of Use Cases

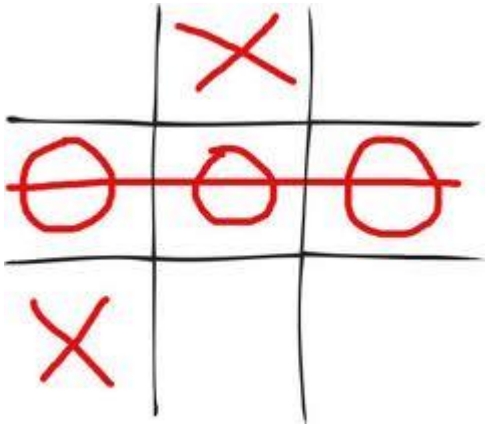
- Represented by use case diagram
- A use case is represented by an ellipse
- System boundary is represented by a rectangle
- Users are represented by stick person icons (actor)
- Communication relationship between actor and use case by a line

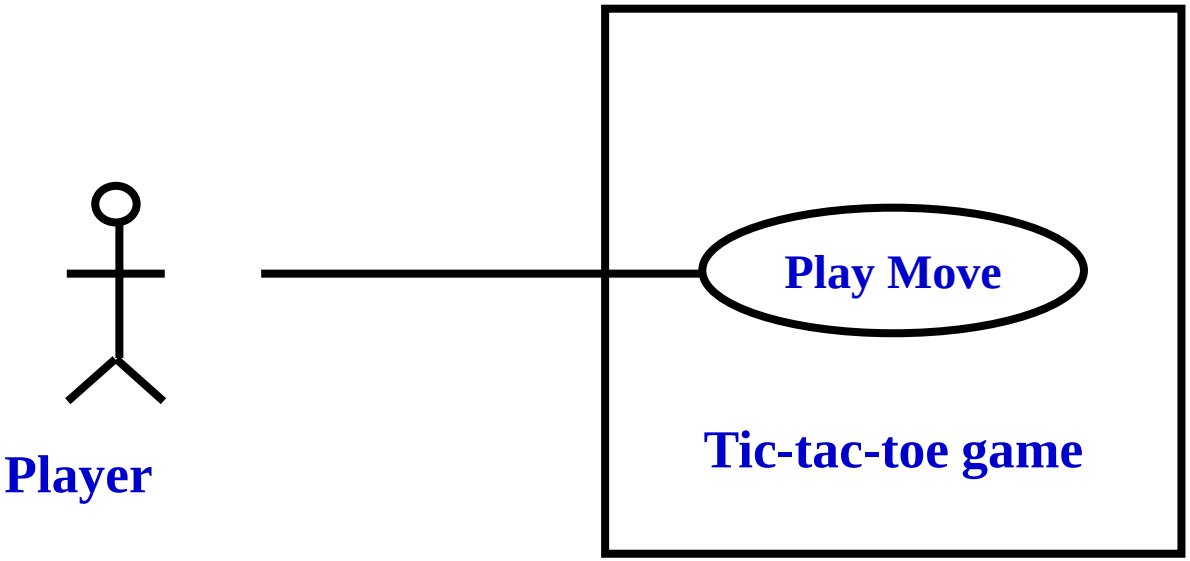
Example problem 1 : Tic-tac-toe



- Tic-tac-toe is a computer game in which a human player and the computer make alternative moves on a 3×3 square.
- A **move** consists of marking previously unmarked square.
- The **player plays** by placing three consecutive marks along a straight line on the square (i.e. along a row, column, or diagonal) wins the game.
- As soon as either 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, but all the squares on the board are filled up, then the game is drawn.
- The computer always tries to win a game.

Use Case Diagram



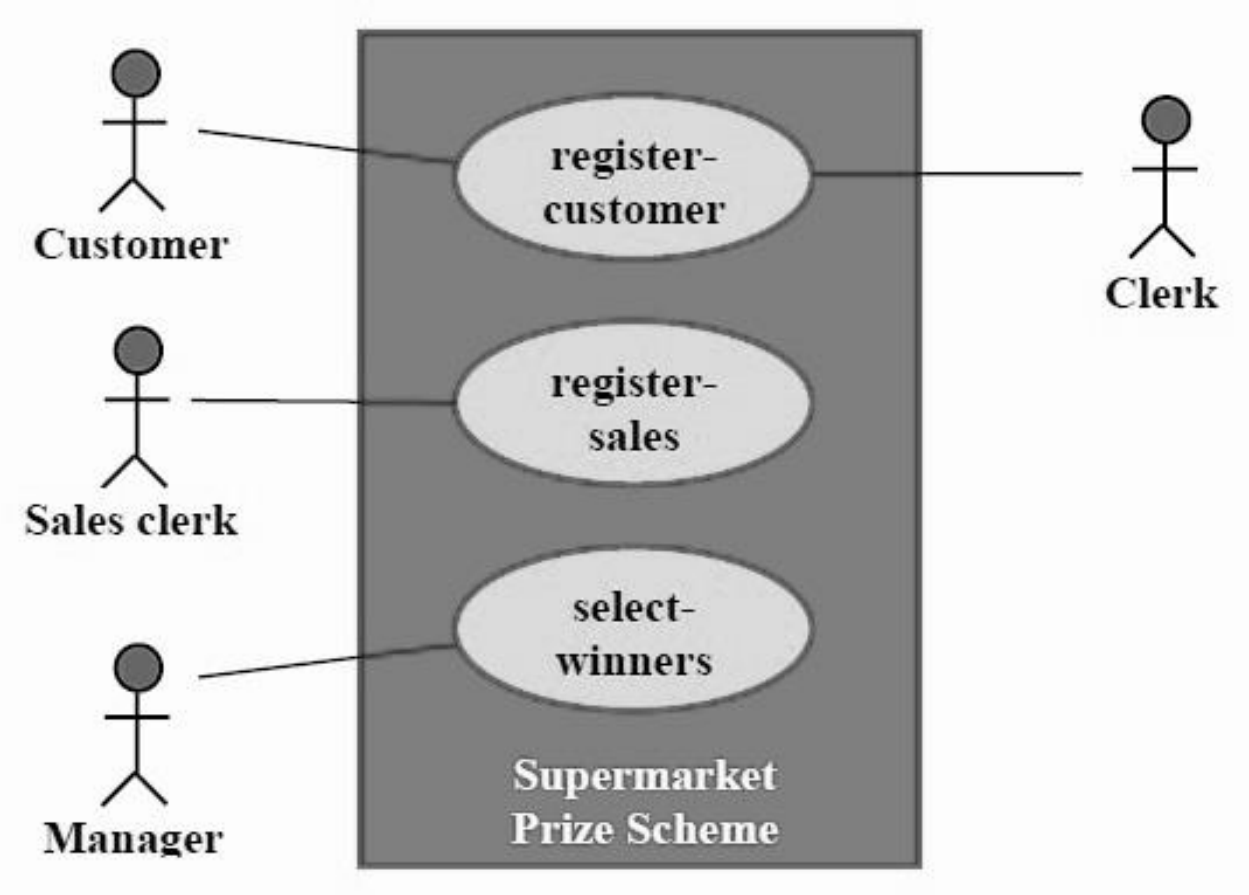


Use case model

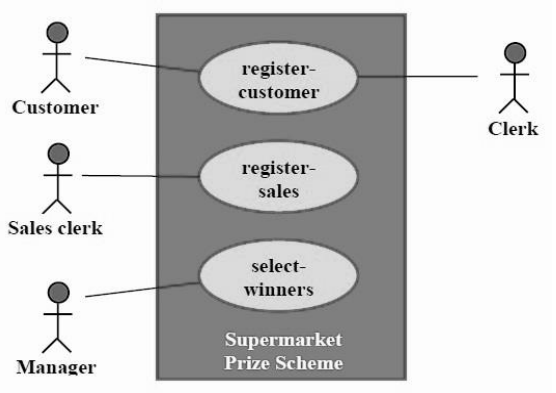
Example problem 2 : Supermarket Prize Scheme

- A supermarket needs to develop the following software to encourage regular customers.
- For this, the customer needs to supply his/her residence address,telephone number, and the driving license number. Each customer 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 each year, the supermarket intends to award surprise gifts to 10 customers who make the highest total purchase over the year.
- Also, it intends to award a gold coin to every customer whose purchase exceeded Rs.100,000.
- The entries against the CN are the reset on the day of every year after the prize winners' lists are generated.

Use Case Diagram



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/Clerk : select register customer option.
2. System: display prompt to enter name, address, and telephone number.
3. Customer/Clerk : enter the necessary values.
4. System: display the generated id and the message that the customer has been successfully registered.

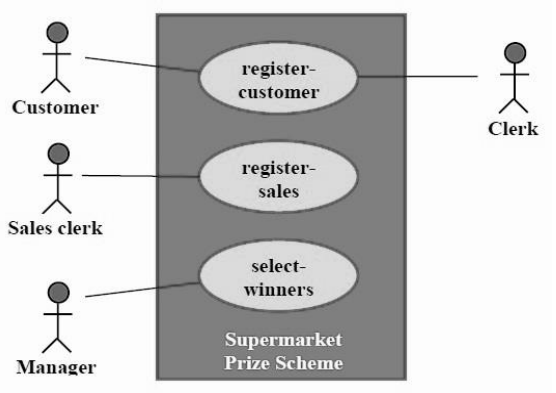
Scenario 2 : at **step 4** of mainline sequence

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

Scenario 3 : at **step 4** of mainline sequence

1. System: displays the message that some input information has not been entered. The system display a prompt to enter the missing value.

Text description Contd..



U2: register-sales : Using this use case, the clerk can register the details of the purchase made by a customer

Scenario 1 : Mainline sequence

1. Sales-Clerk: select register sale option.
2. System: display prompt to enter purchase details and the id of the customer
3. Sales-Clerk: enter the required detail.
4. System: display the message of having successfully registered the sale

U3: select-winner : Using this use case, the manager can generate the winner list

Scenario 1 : Mainline sequence

1. Manager: selects the select -winner option.
2. System: display the gold coin and the surprise gift winner list

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).

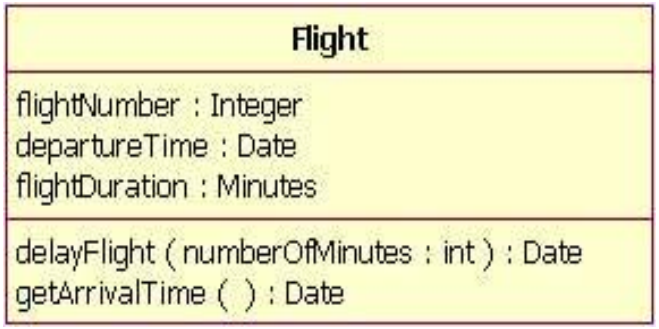
Structural View

Class Diagram

Class representation

Classes are represented with boxes which contain three parts:

- The top part contains the name of the class. It is printed in Bold, centered and the first letter capitalized.
- The middle part contains the attributes of the class. They are left aligned and the first letter is lower case.
- The bottom part gives the methods or operations of the class. They are also left aligned and the first letter is lower case.



Class representation
Visibility

To specify the visibility of a class member (i.e., any attribute or method) these are the following notations that must be placed before the member's name.

Mainly :

"+"	Public
"-"	Private
"#"	Protected
"_"	Static

That all ?

In the design of a system, a number of classes are identified and grouped together in a class diagram which helps to determine the static relations between those objects.

With detailed modeling, the classes of the conceptual design are often split into a number of subclasses.

In order to further describe the behavior of systems, we need to understand relationship among the class in the system

Class relationship

A relationship is a general term covering the specific types of logical connections found on class diagrams.

Four types of Class relationships among the classes in a software system :

- Inheritance
- Association
- Aggregation/Composition
- Dependency

Inheritance
Generalization and Specialization

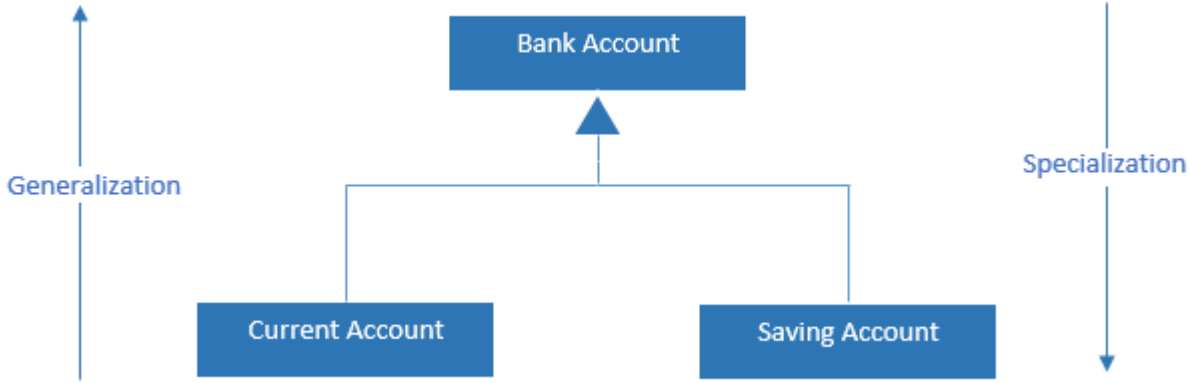
The process of extracting common characteristics from two or more classes and combining them into a generalized base/super class, is called Generalization. The common characteristics can be attributes or methods.

Generalization is represented by a **triangle followed by a line**.

Specialization is the reverse process of Generalization means creating new derive/sub classes from an existing class.

Inheritance
Generalization and Specialization

Example of Bank Account

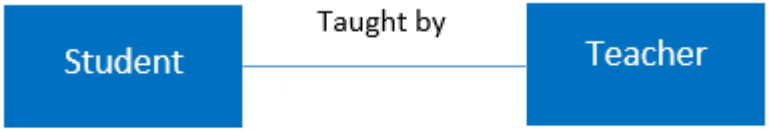


A Bank Account is of two types – Current Account and Saving Account. Current Account and Saving Account inherits the common/generalized properties like Account Number, Account Balance etc. from a Bank Account and also have their own specialized properties like interest rate etc.

Association

It represents a relationship between two or more objects where all objects have their **own lifecycle** and there is **no owner**.

The name of an association specifies the nature of relationship between objects. This is represented by a solid **line**.



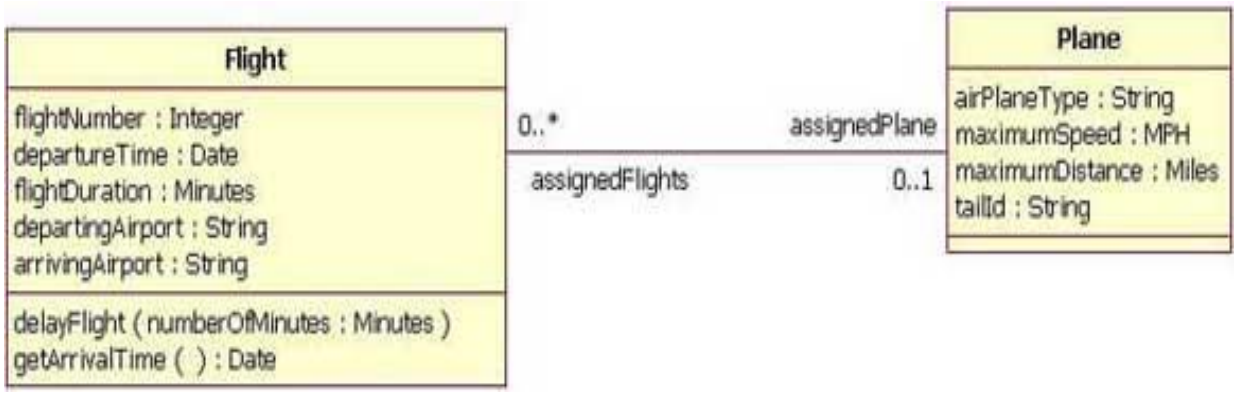
Association

Multiple students can associate with a single teacher and a single student can associate with multiple teachers. But there is no ownership between the objects and both have their own lifecycle. Both can be created and deleted independently.

Bi-directional (standard) association

Associations are always assumed to be bi-directional; this means that both classes are aware of each other and their relationship, unless qualify the association as some other type

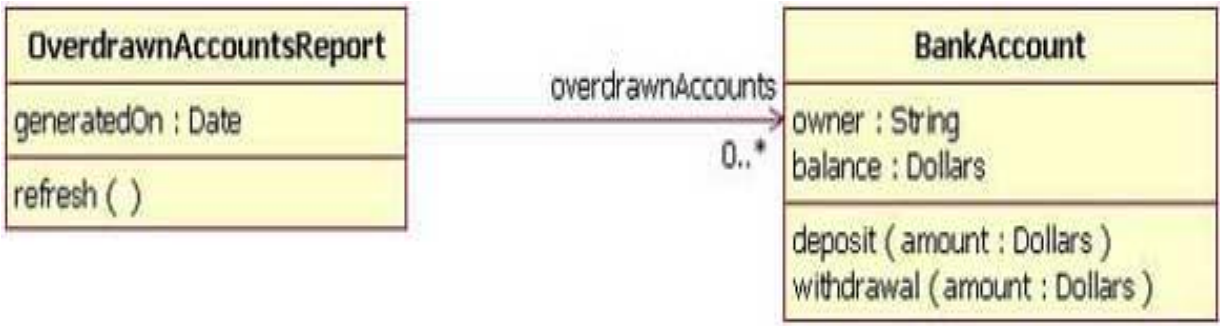
A bi-directional association is indicated by a solid line between the two classes. At either end of the line, you place a role name and a multiplicity value.



Uni-directional association

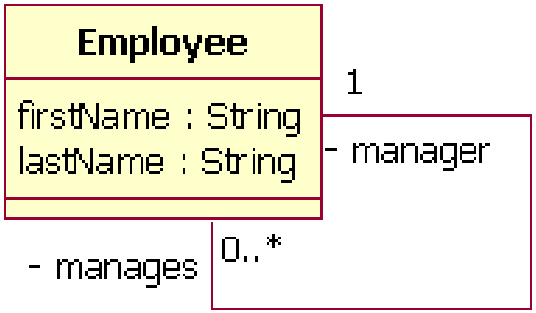
In a uni-directional association, two classes are related, but only one class knows that the relationship exists.

A uni-directional association is drawn as a solid line with an **open arrowhead** (not the closed arrowhead, or triangle, used to indicate inheritance) pointing to the known class



Reflexive (recursive) association

A class can have association with itself.



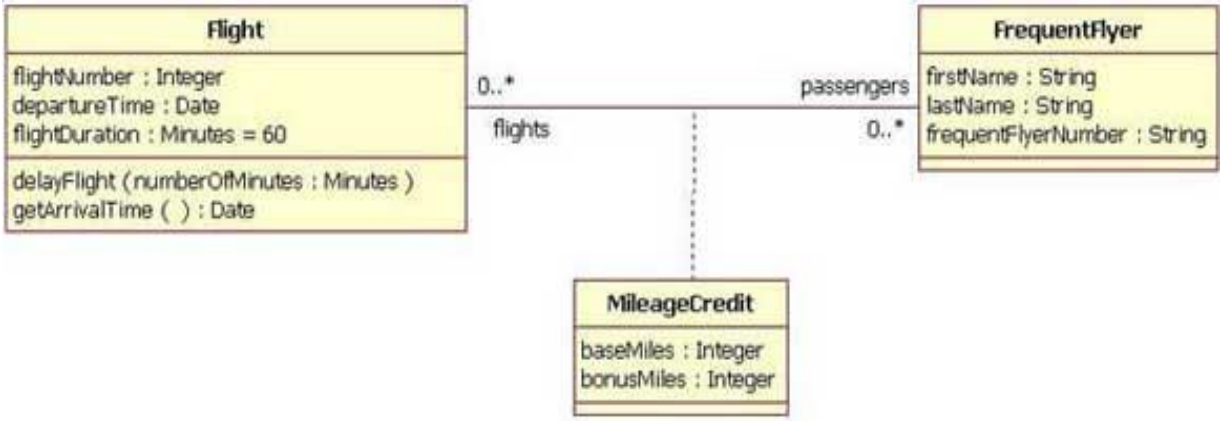
An instance of Employee can be the manager of another Employee instance. However, because the relationship role of "manages" has a multiplicity of 0..; an Employee might not have any other Employees to manage.*

N-ary association

There are times when it is needed to include another class because it includes valuable information about the relationship.

For this an association class is used to tie to the primary association. An association class is represented like a normal class.

The difference is that the association line between the primary classes intersects a dotted line connected to the association class.



Aggregation

It is a specialized form of Association where all object have their **own lifecycle but there is ownership**. Such that the child class instance can outlive its parent class.

This represents “whole-part or a-part-of” relationship. This is represented by a **hollow diamond followed by a line**.

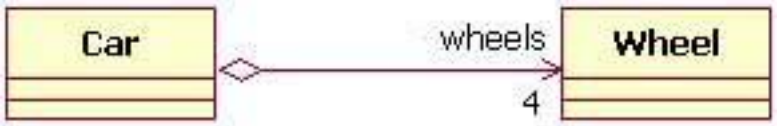
Aggregation

A Teacher may belongs to multiple departments. Hence Teacher is a part of multiple departments. But if we delete a Department, Teacher Object will not destroy.

Aggregation
Another example

Relationship between a Car and a Wheel.

Car as a whole entity and Car Wheel as part of the overall Car. The wheel can be created weeks ahead of time, and it can sit in a warehouse before being placed on a car during assembly. The Wheel class's instance clearly lives independently of the Car class's instance. A wheel of one car can be fit into another car.



Composition

It is a specialized form of Aggregation. It is a strong type of Aggregation.

In this relationship **child objects does not have their lifecycle without Parent object**. If a parent object is deleted, all its child objects will also be deleted.

This represents “death” relationship. This is represented by a **solid diamond followed by a line**.

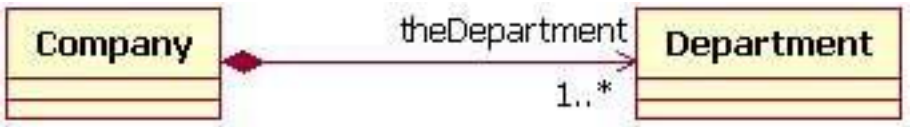
Composition

House can contain multiple rooms there is no inaependent life of room and any room cannot belongs to two different house if we delete the house room will automatically delete.

Composition
Another example

Relationship between a Company and a Department.

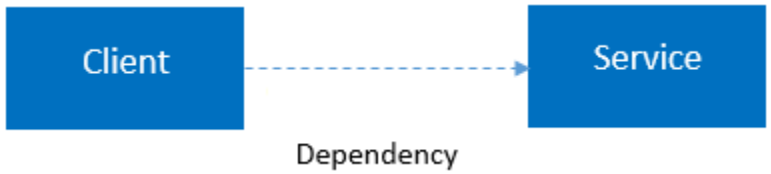
A department cannot exist before a company exists. Company class instance will always have at least one Department class instance. When the Company instance is removed/destroyed, the Department instance is automatically removed/destroyed as well.



Dependency

It represents a relationship between two or more objects where an object is dependent on another object(s) for its specification or implementation.

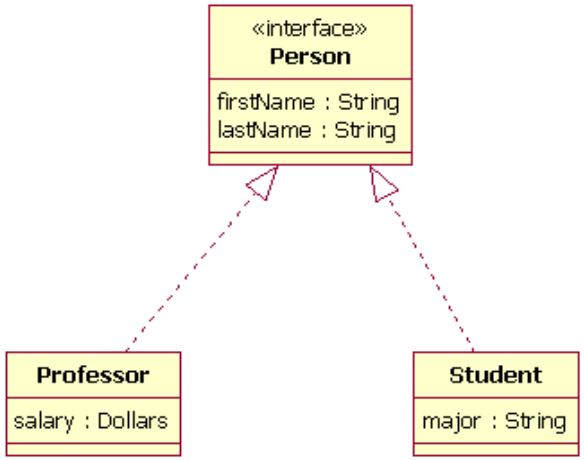
This is represented by a **dashed arrow**.



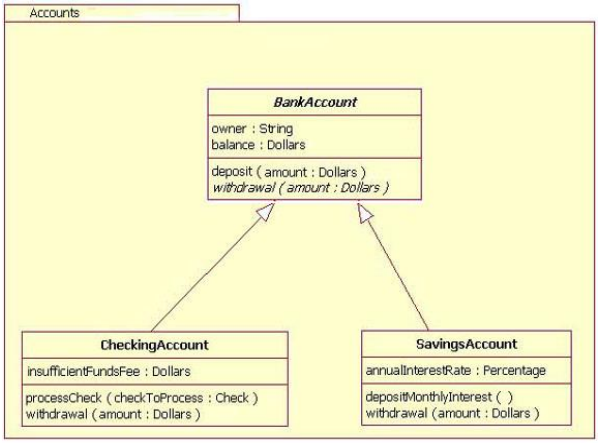
A client is dependent on the service for implementing its functionalities.

Few more

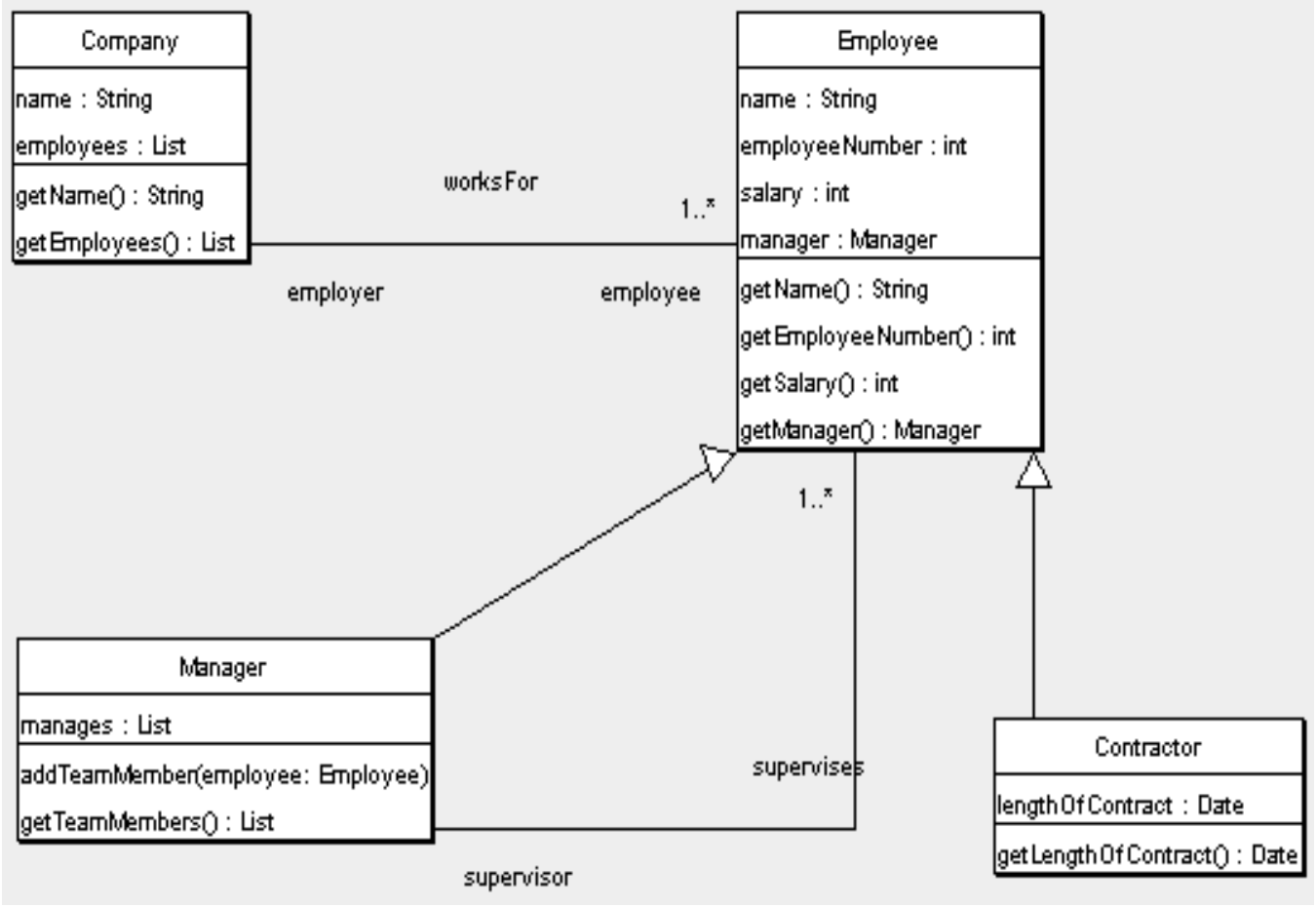
Interface :



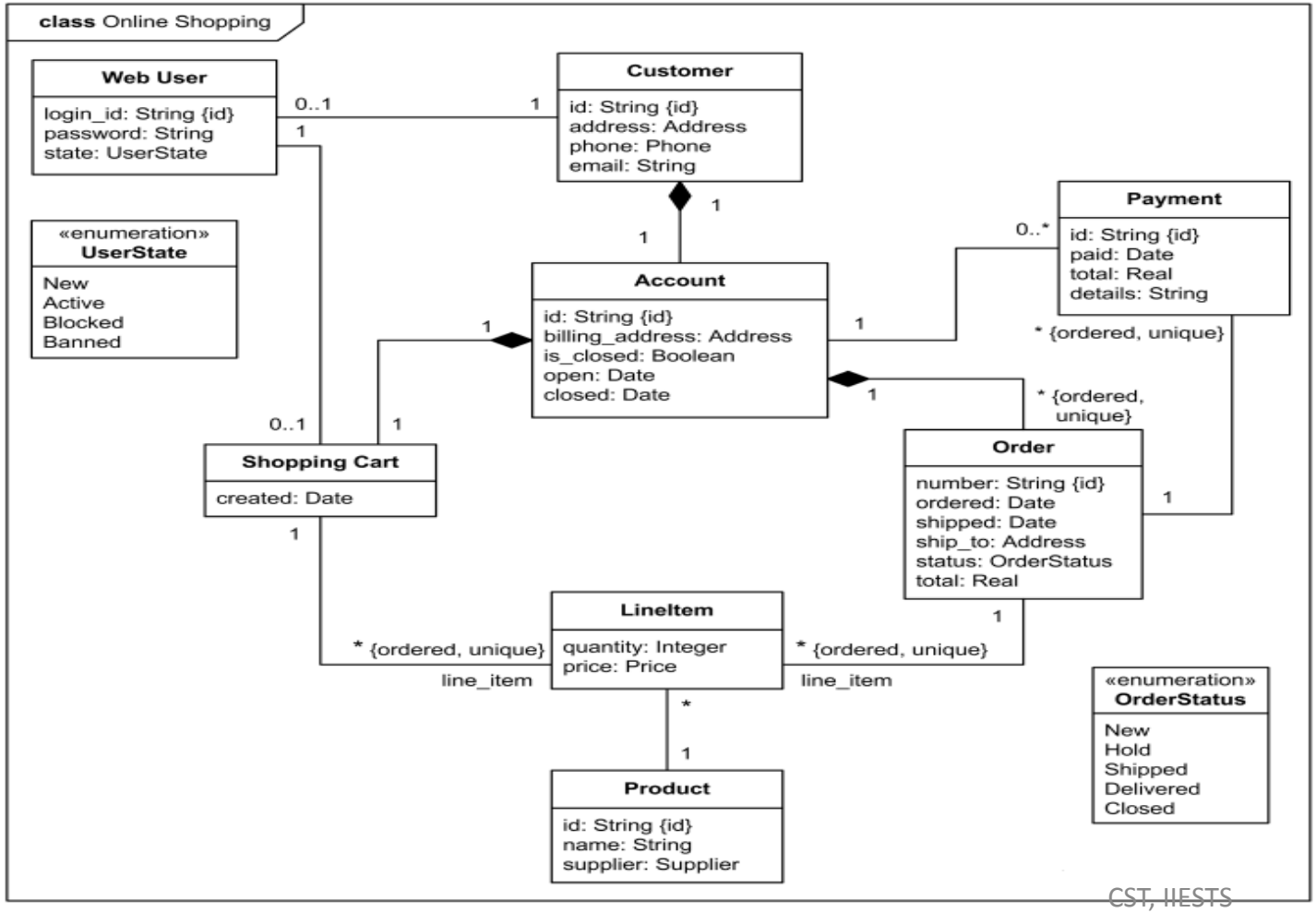
Package :



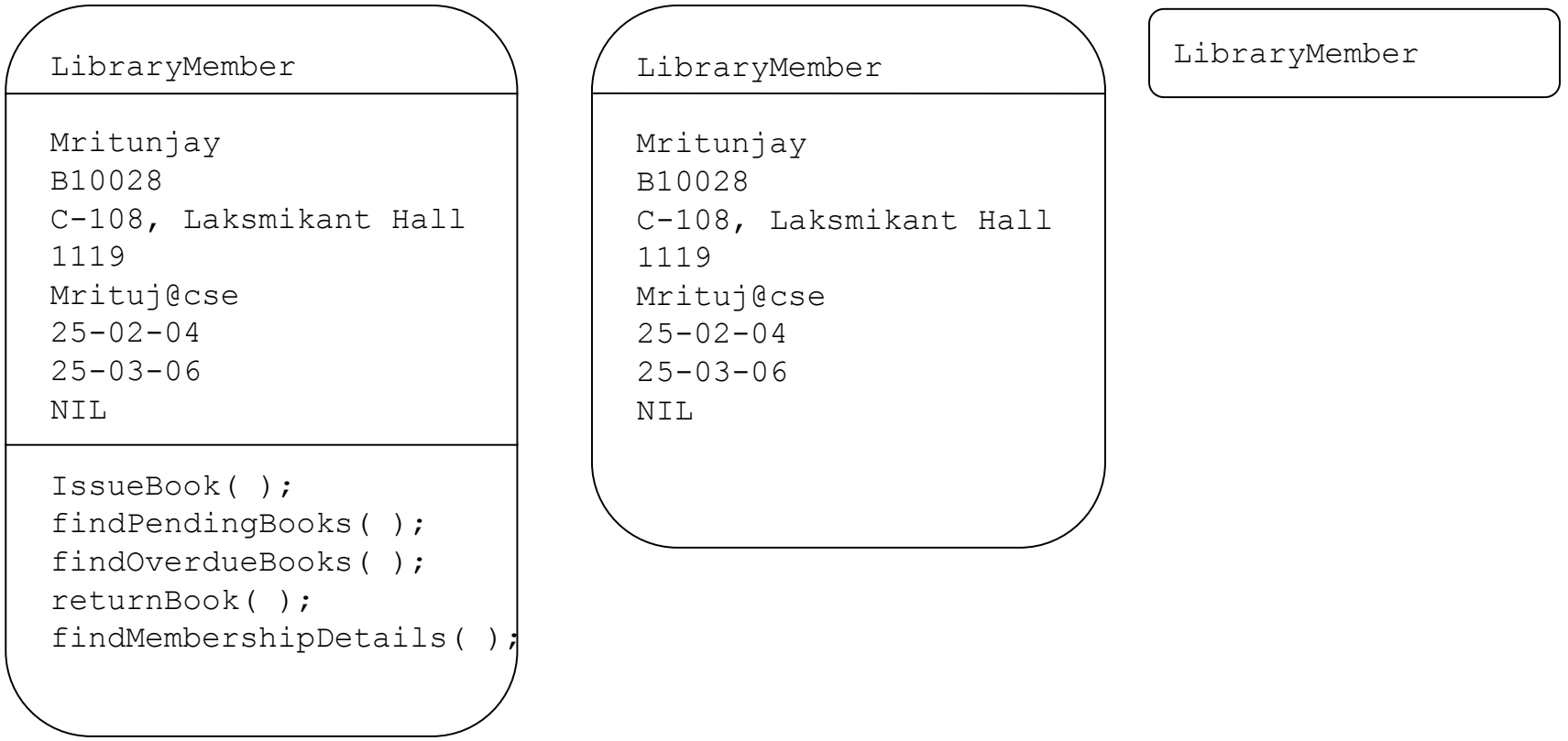
Class Diagram Example 1 : Company, Employee, Manager



A typical Class Diagram Example : Online Shopping



Object Diagram



Different representations of the LibraryMember object

References

Book :

Fundamental of Software Engineering – Rajib Mall, PHI

URL :

https://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/