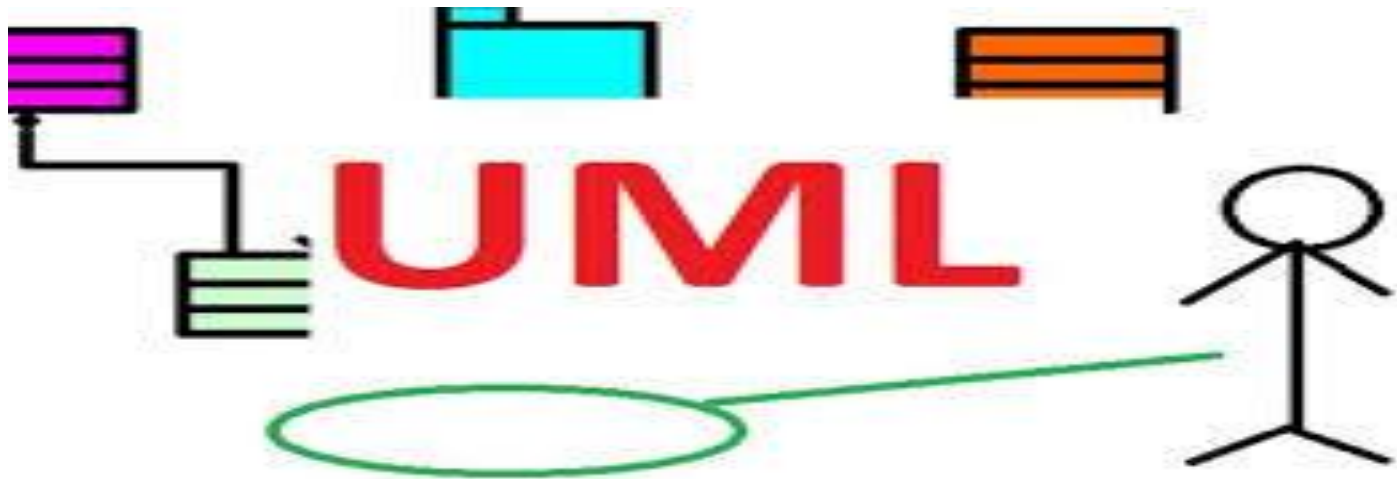




Introduction to UML

Kiran Waghmare
Session VII



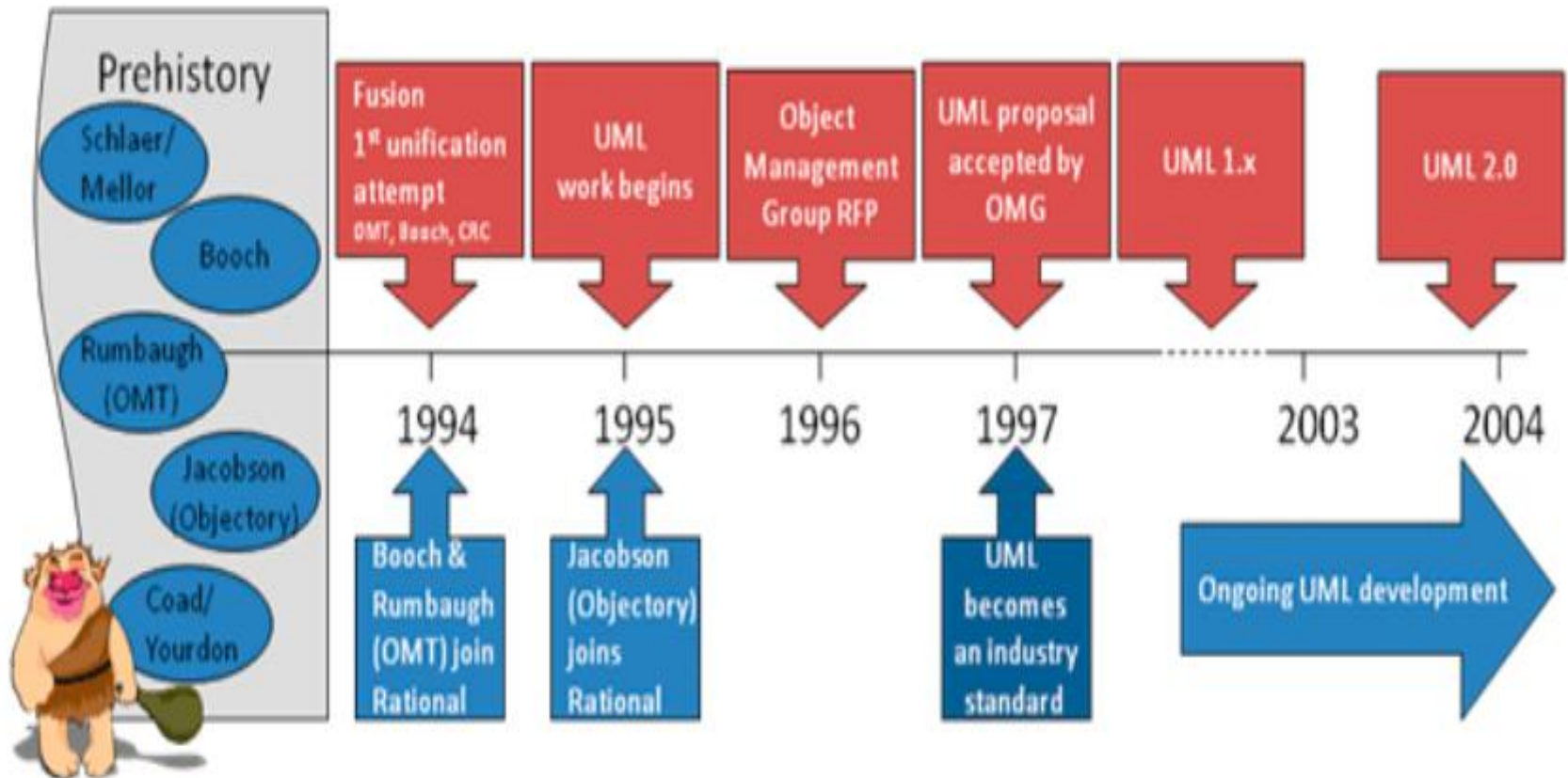
UML

- UML is **Unified Modeling Language**.
- **Graphical language** for visualizing artifacts of the system.
- Allow to create a **blue print** of all the aspects of the system.

What is UML?

- UML stands for “Unified Modeling Language”
- It is a industry-standard graphical language for
 - specifying,
 - visualizing,
 - constructing, and
 - documenting the artifacts of software systems
- The UML uses mostly graphical notations to
 - express the **OO analysis and design** of software projects.
- Simplifies the complex process of software design

History of UML



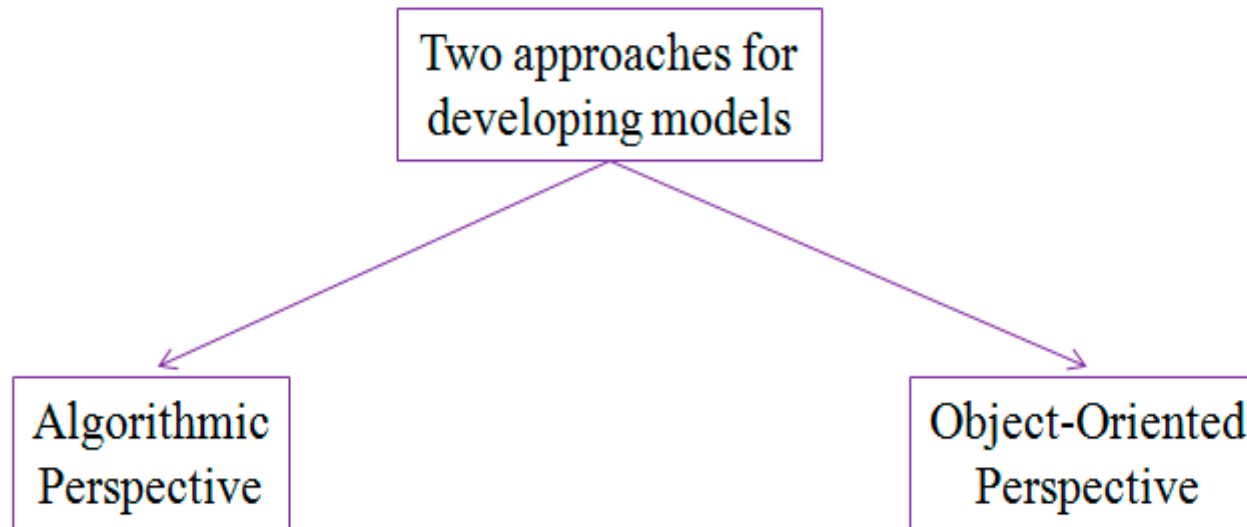
Characteristics of UML

- The UML has the following features:
 1. It is a **generalized** modeling language.
 2. It is **distinct** from other programming languages like C++, Python, etc.
 3. It is **interrelated** to object-oriented analysis and design.
 4. It is **used to visualize** the workflow of the system.
 5. It is a **pictorial language**, used to generate powerful modeling artifacts.

Conceptual Modeling

- Before moving ahead with the concept of UML, we should first **understand the basics** of the conceptual model.
- A conceptual model is composed of several interrelated concepts.
- It makes it easy to understand the objects and how they interact with each other.
- This is the first step before drawing UML diagrams.

UML Approaches

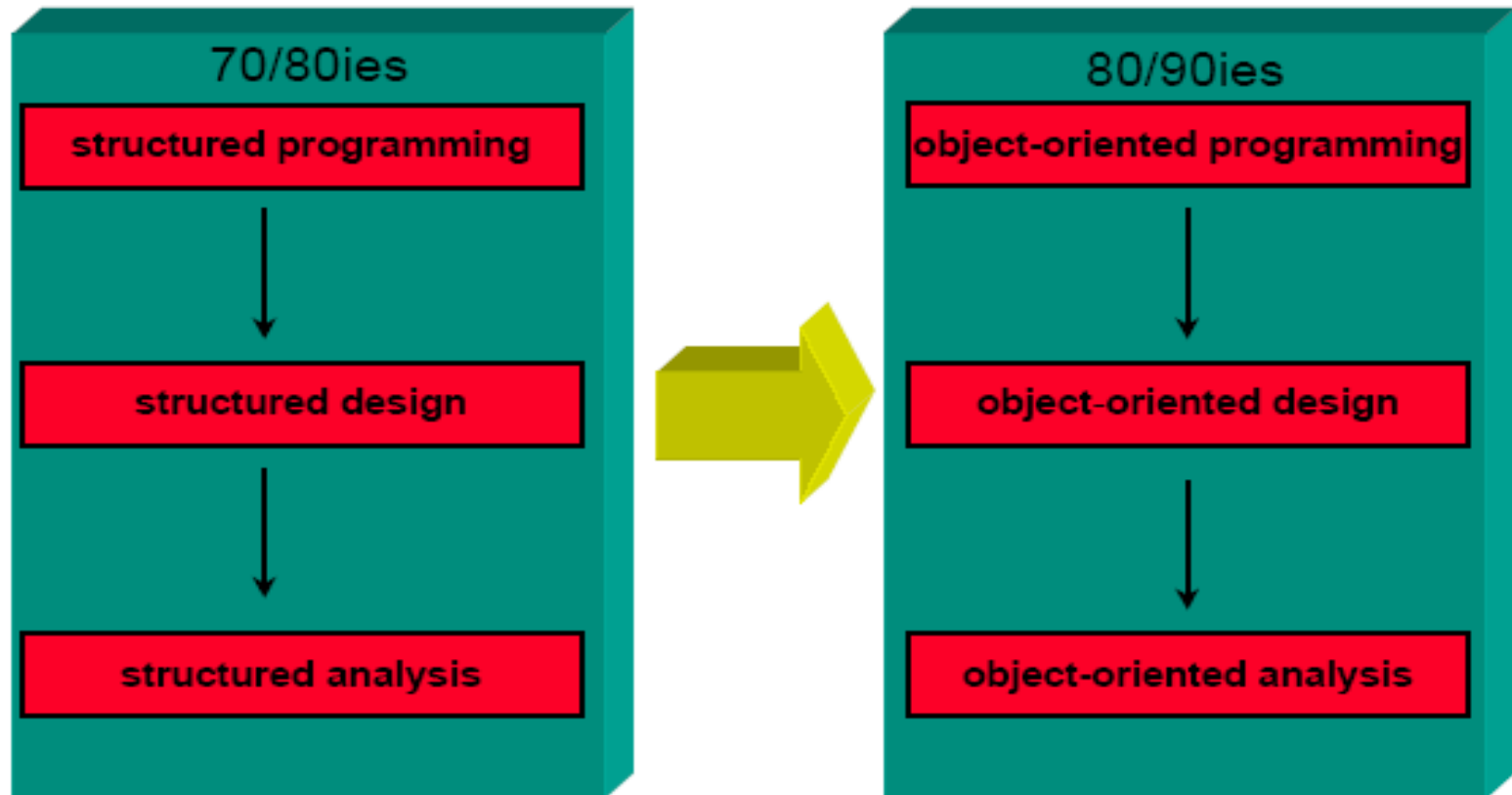


- Traditional approach
- Fundamental unit is function
- Functional decomposition
- Difficult to maintain

- Modern approach
- Fundamental unit is object or class
- Communication through messages
- Easy to maintain and enhance

Object Oriented Modeling

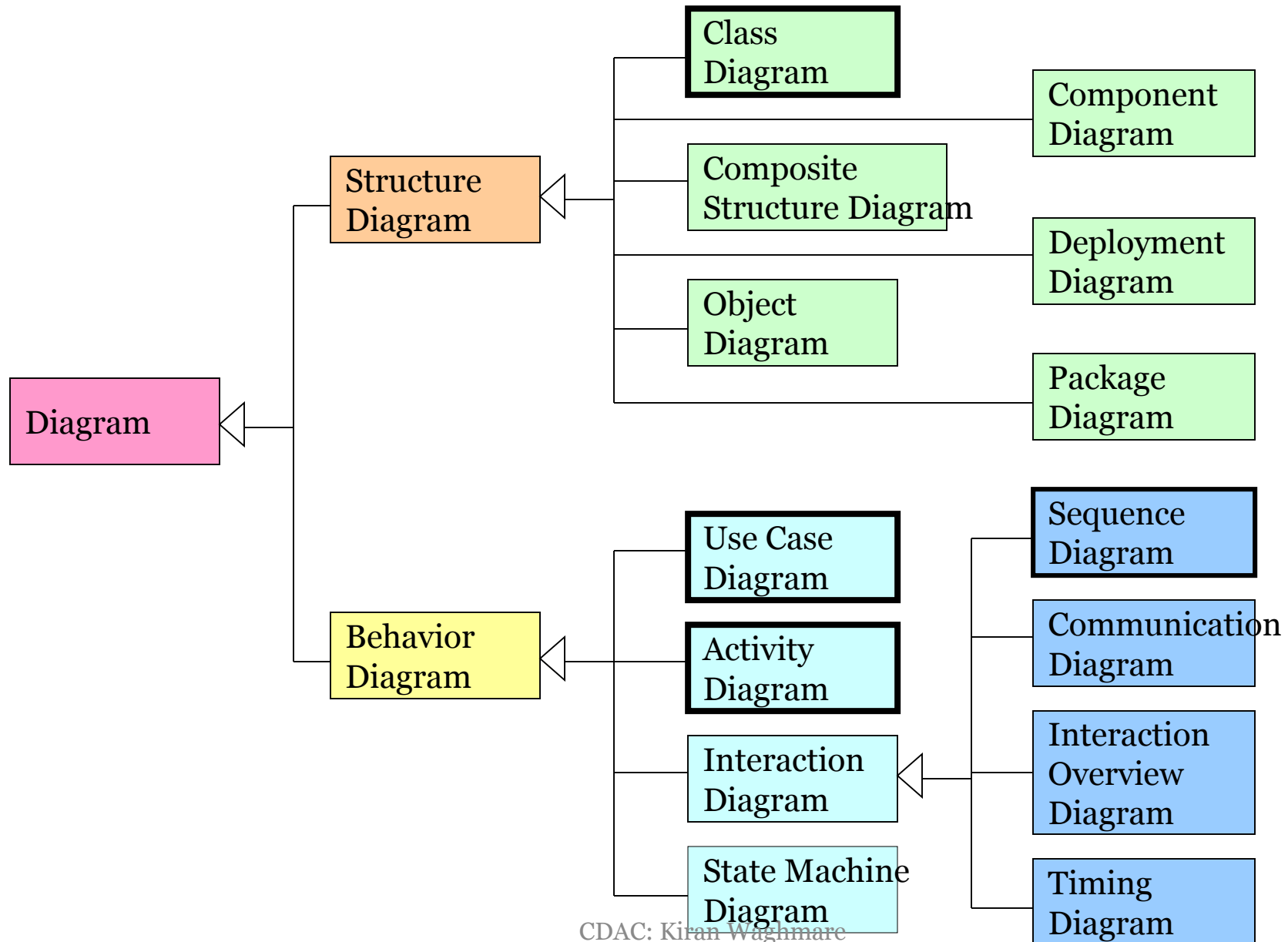
Evolution of OO Development Methods



Following are some object-oriented concepts that are needed to begin with UML:

- **Object:** An object is a real world entity. There are many objects present within a single system. It is a fundamental building block of UML.
- **Class:** A class is a software blueprint for objects, which means that it defines the variables and methods common to all the objects of a particular type.
- **Abstraction:** Abstraction is the process of portraying the essential characteristics of an object to the users while hiding the irrelevant information. Basically, it is used to envision the functioning of an object.
- **Inheritance:** Inheritance is the process of deriving a new class from the existing ones.
- **Polymorphism:** It is a mechanism of representing objects having multiple forms used for different purposes.
- **Encapsulation:** It binds the data and the object together as a single unit, enabling tight coupling between them.

Classification of Diagram Types



- UML diagrams are divided into three different categories such as,
 1. Structural diagram
 2. Behavioral diagram
 3. Interaction diagram

Overview of UML Diagrams

Structural

: element of spec. irrespective of time

- Class
- Component
- Deployment
- Object
- *Composite structure*
- *Package*

Behavioral

: behavioral features of a system / business process

- Activity
- State machine
- Use case
- *Interaction*

Interaction

: emphasize object interaction

- Communication(collaberati on)
- Sequence
- *Interaction overview*
- *Timing*

UML Tools

- There are **many tools available** in the market to generate UML diagrams.
- Some are desktop based while others can be used **online**.
- Following is a curated list of tools which can be used for the creation of UML models:
 1. Star UML
 2. Argo UML
 3. Dia
 4. Visual Paradigm
 5. U-Model
 6. UML lab
 7. Enterprise Architect

UML Building Blocks

- Conceptual model of UML can be mastered by learning the following three major elements:
 1. UML building blocks
 2. Rules to connect the building blocks
 3. Common mechanisms of UML
- The building blocks of UML can be defined as:
 1. Things
 2. Relationships
 3. Diagrams

Architecture

View	Stakeholders	Static Aspects	Dynamic Aspects
Use case view	End users Analysts Testers	Use case diagrams	Interaction diagrams Statechart diagrams Activity diagrams
Design view	End users	Class diagrams	Interaction diagrams Statechart diagrams Activity diagrams
Implementation view	Programmers Configuration managers	Component diagrams	Interaction diagrams Statechart diagrams Activity diagrams
Process view	Integrators	Class diagrams (active classes)	Interaction diagrams Statechart diagrams Activity diagrams
Deployment view	System Engineer	Deployment diagrams	Interaction diagrams Statechart diagrams Activity diagrams

Structural things:

The **Structural things** define the static part of the model. They represent physical and conceptual elements. Following are the brief descriptions of the structural things.

Class:

Class represents set of objects having similar responsibilities.



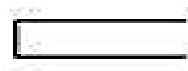
Interface:

Interface defines a set of operations which specify the responsibility of a class.



Collaboration:

Collaboration defines interaction between elements.



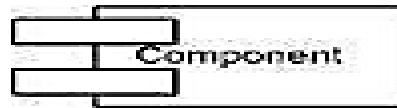
Use case:

Use case represents a set of actions performed by a system for a specific goal.



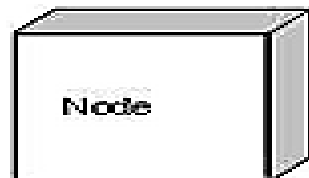
Component:

Component describes physical part of a system.



Node:

A node can be defined as a physical element that exists at run time.



Behavioral things:

A behavioral thing consists of the dynamic parts of UML models. Following are the behavioral things:

Interaction:

Interaction is defined as a behavior that consists of a group of messages exchanged among elements to accomplish a specific task.



State machine:

State machine is useful when the state of an object in its life cycle is important. It defines the sequence of states an object goes through in response to events. Events are external factors responsible for state change.



Grouping things:

Grouping things can be defined as a mechanism to group elements of a UML model together. There is only one grouping thing available:

Package:

Package is the only one grouping thing available for gathering structural and behavioral things.



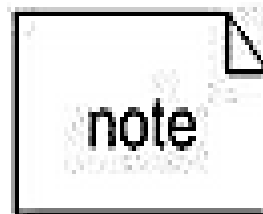
Annotational things:

Annotational things can be defined as a mechanism to capture remarks, descriptions, and comments of UML model elements.

Note is the only one Annotational thing available.

Note:

A note is used to render comments, constraints etc of an UML element.



(2) Relationship :

Relationship is another most important building block of UML. It shows how elements are associated with each other and this association describes the functionality of an application.

There are four kinds of relationships available.

Dependency:

Dependency is a relationship between two things in which change in one element also affects the other one.



Association:

Association is basically a set of links that connects elements of an UML model. It also describes how many objects are taking part in that relationship.



Generalization:

Generalization can be defined as a relationship which connects a specialized element with a generalized element. It basically describes inheritance relationship in the world of objects.



Realization:

Realization can be defined as a relationship in which two elements are connected. One element describes some responsibility which is not implemented and the other one implements them. This relationship exists in case of interfaces.



Types of UML Diagrams

- 1. Use Case Diagram**
- 2. Class Diagram**
- 3. Sequence Diagram**
- 4. Collaboration Diagram**
- 5. State Diagram**

This is only a subset of diagrams ... but are most widely used

UseCase Diagram

Use Case Diagram

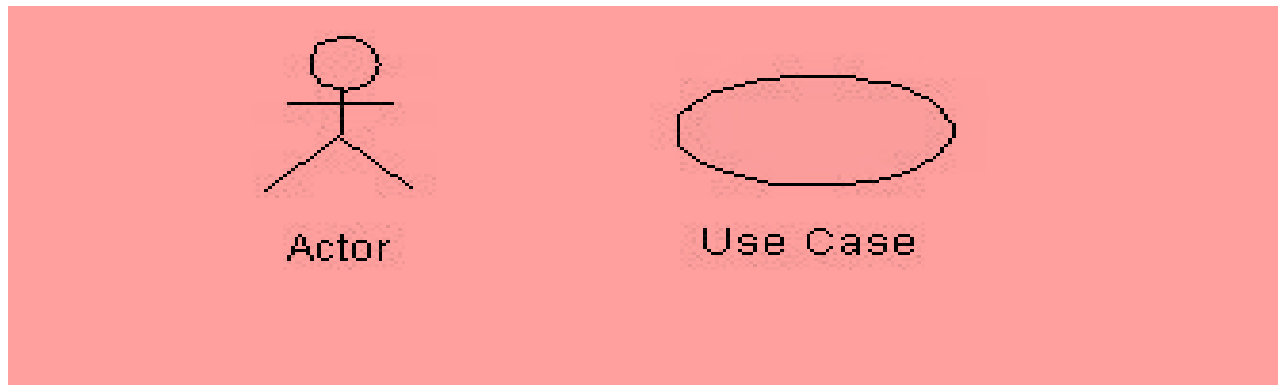
- Used for describing a set of user scenarios
- Mainly used for capturing user requirements
- Work like a **contract** between end user and software developers

Use Case Diagram (core components)

Actors: A role that a user plays with respect to the system, including human users and other systems.

e.g., inanimate physical objects (e.g. robot); an external system that needs some information from the current system.

Use case: A set of scenarios that describing an interaction between a user and a system, including alternatives.

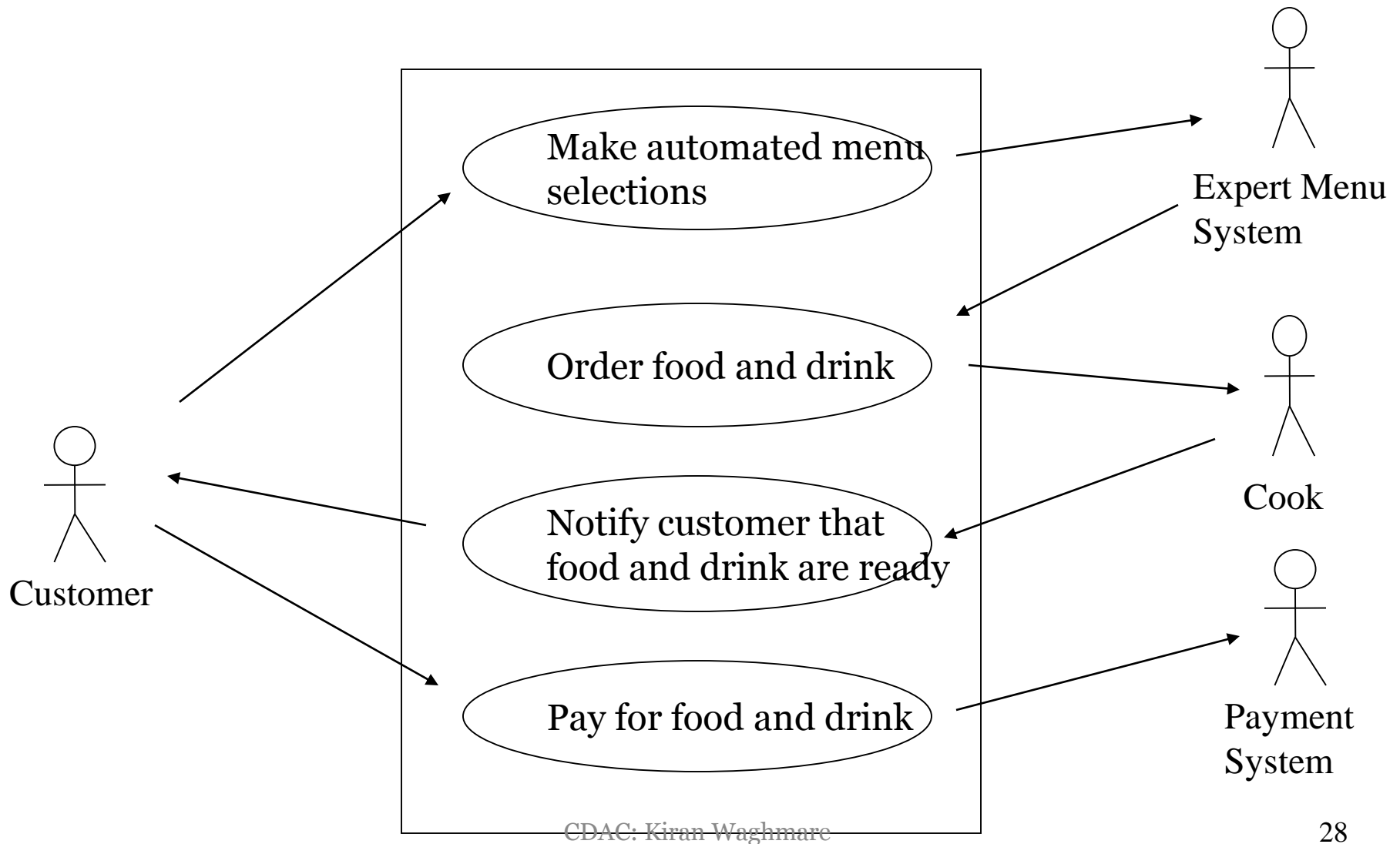


System boundary: rectangle diagram representing the boundary between the actors and the system.

Use cases diagram

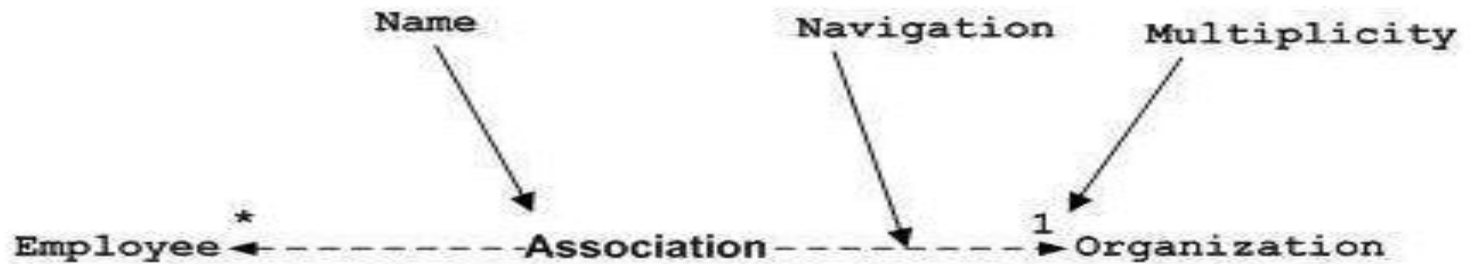


Example Use Case Diagram



Use Case Diagram(core relationship)

Association: communication between an actor and a use case; Represented by a solid line.

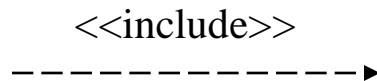


Generalization: relationship between one general use case and a special use case (used for defining special alternatives)
Represented by a line with a triangular arrow head toward the parent use case.

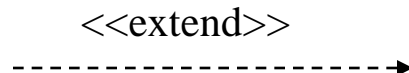


Use Case Diagram(core relationship)

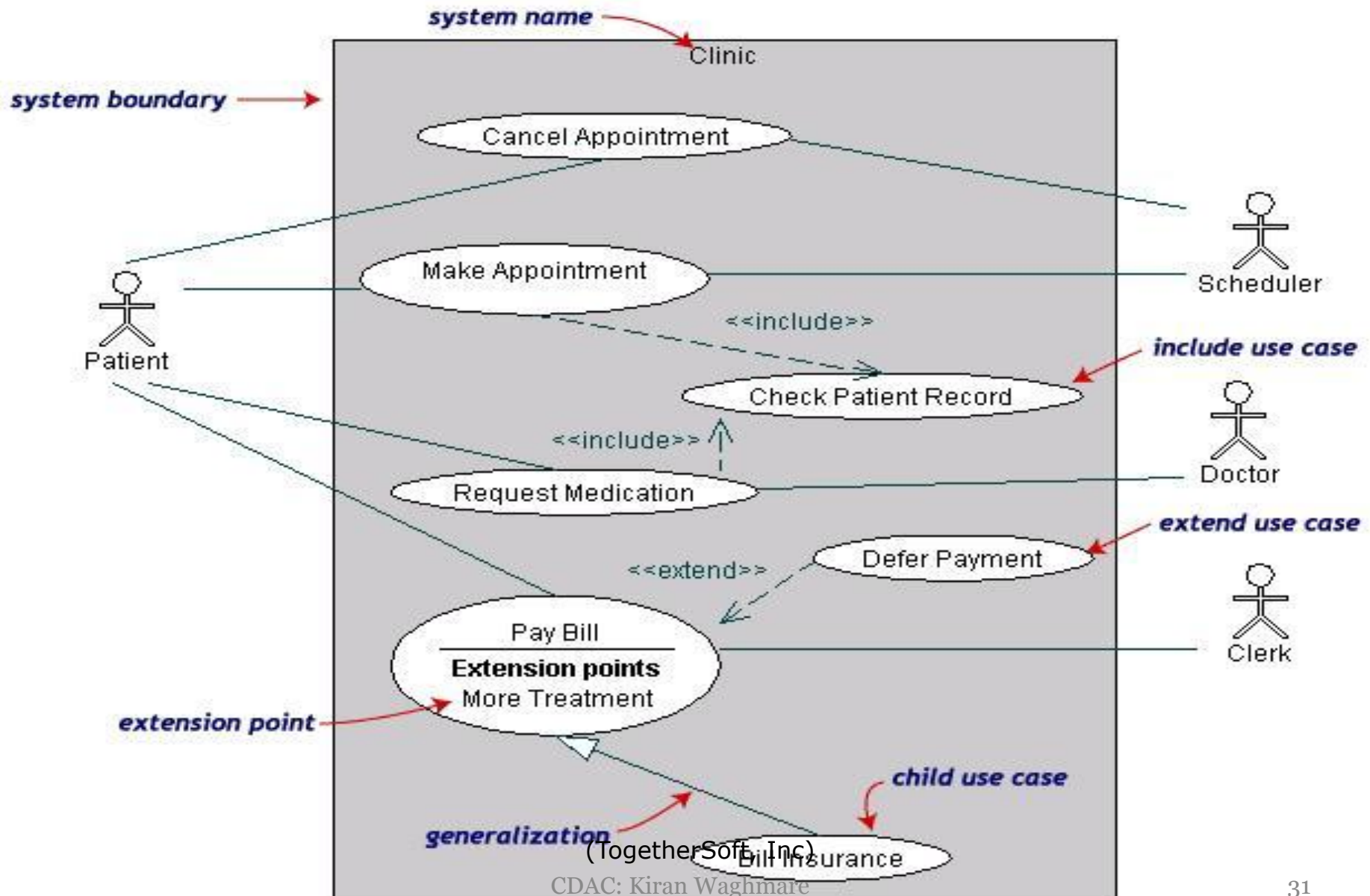
Include: a dotted line labeled <<include>> beginning at base use case and ending with an arrows pointing to the include use case. The include relationship occurs when **a chunk of behavior is similar across more than one use case**. Use “include” in stead of copying the description of that behavior.



Extend: a dotted line labeled <<extend>> with an arrow toward the **base case**. The extending use case **may add behavior to the base use case**. The base class declares “extension points”.



Use Case Diagrams(cont.)



Class Diagram

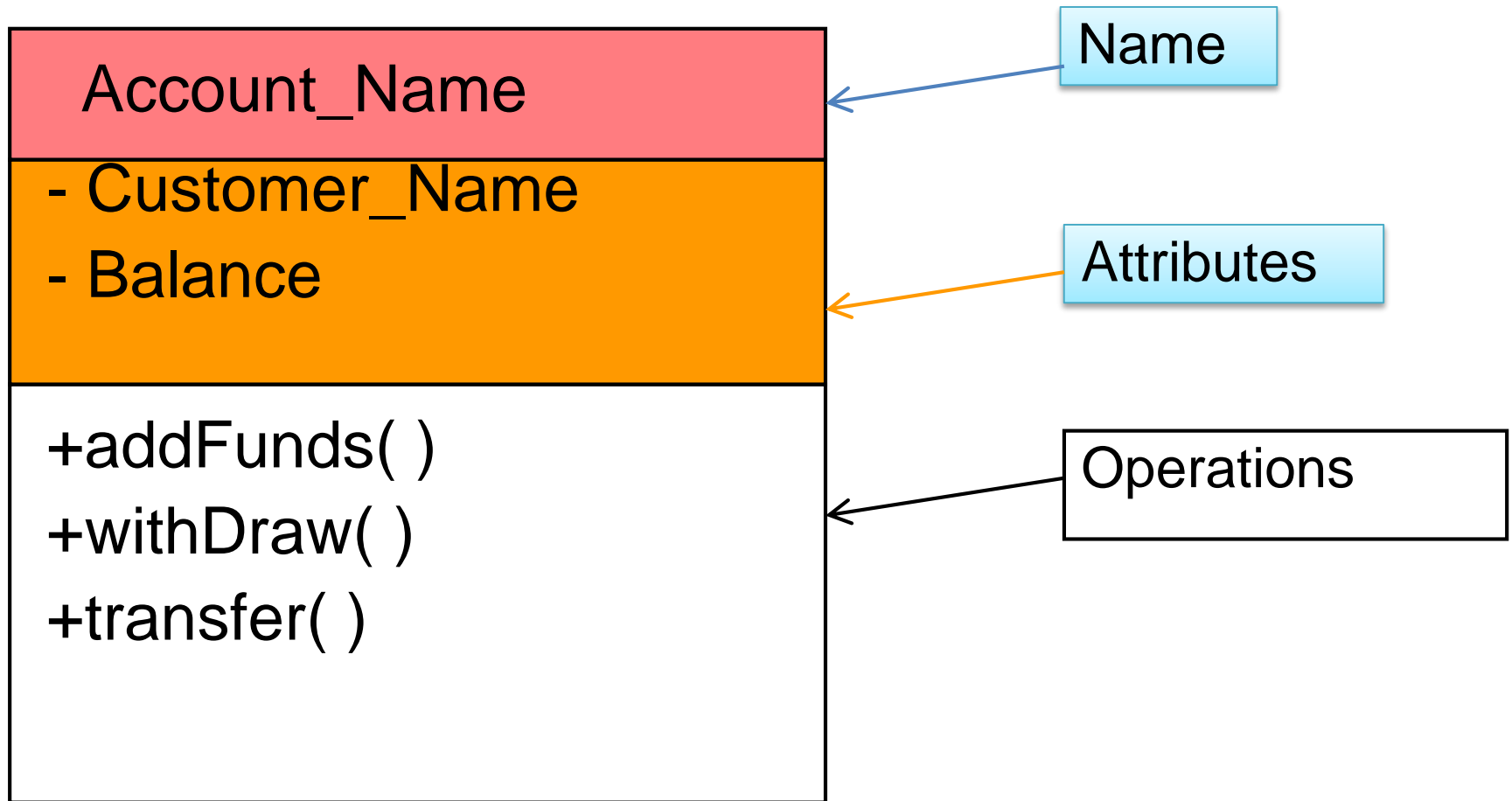
Class diagram

- Used for describing **structure and behavior** in the use cases
- Provide a conceptual model of the system in terms of entities and their relationships
- Used for requirement capture, end-user interaction
- Detailed class diagrams are used for developers

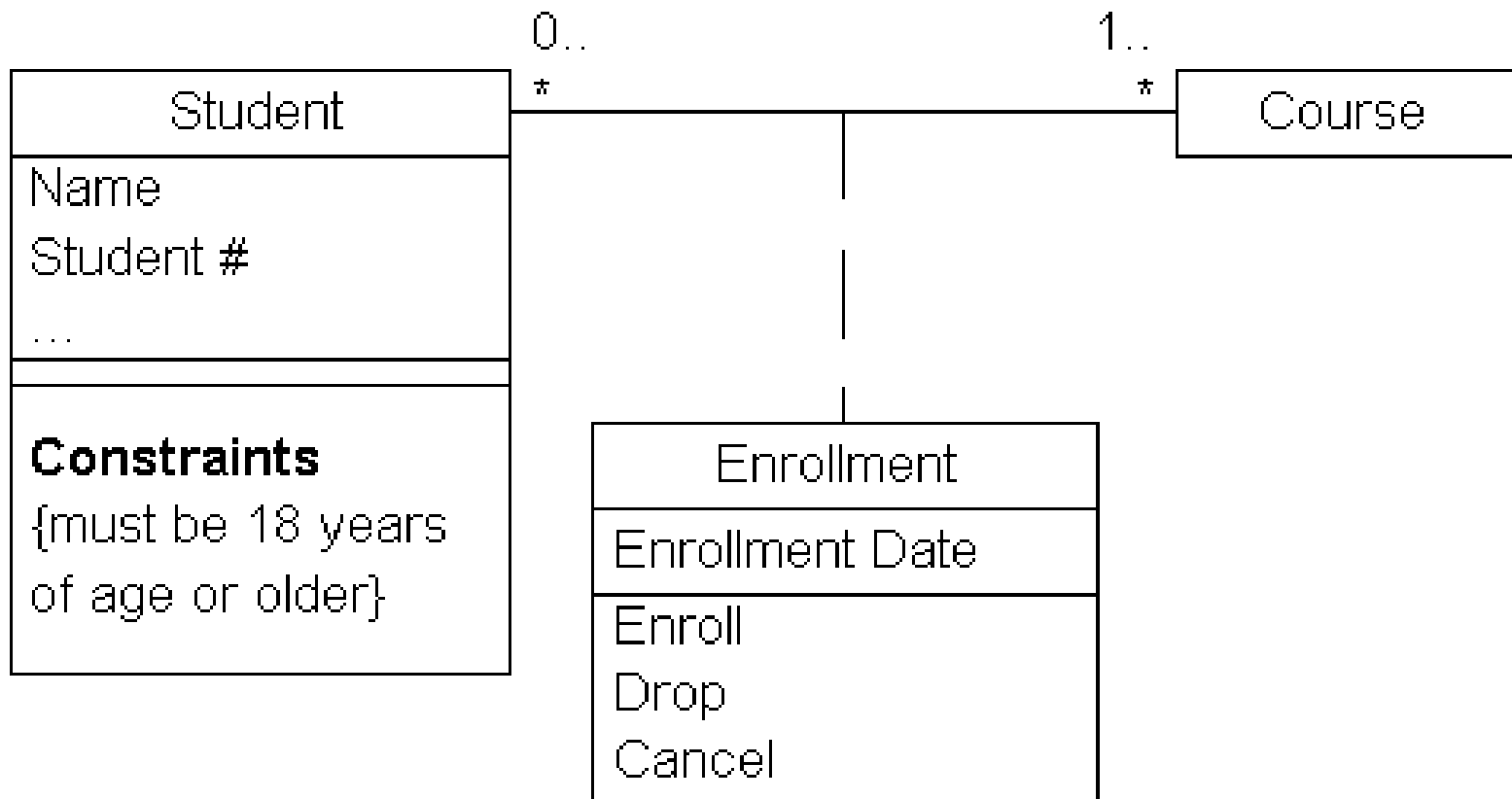
Class representation

- Each class is represented by a rectangle subdivided into three compartments
 - Name
 - Attributes
 - Operations
- Modifiers are used to indicate visibility of attributes and operations.
 - ‘+’ is used to denote *Public* visibility (everyone)
 - ‘#’ is used to denote *Protected* visibility (friends and derived)
 - ‘-’ is used to denote *Private* visibility (no one)
- By default, attributes are hidden and operations are visible.

An example of Class



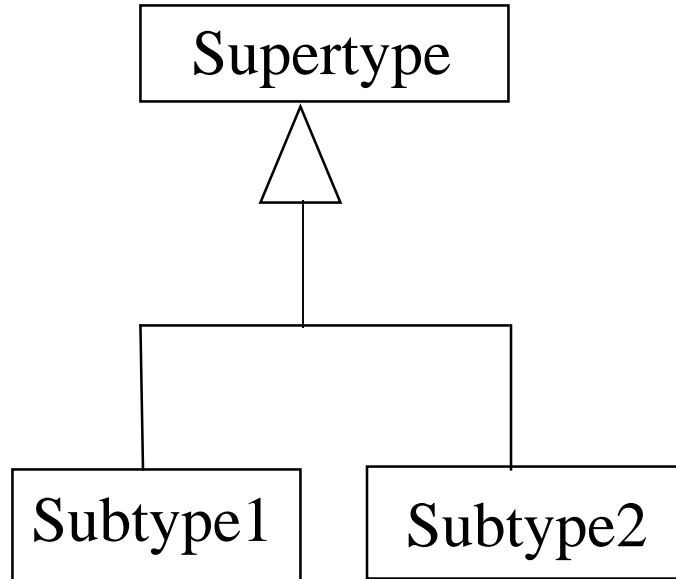
Class diagram



OO Relationships

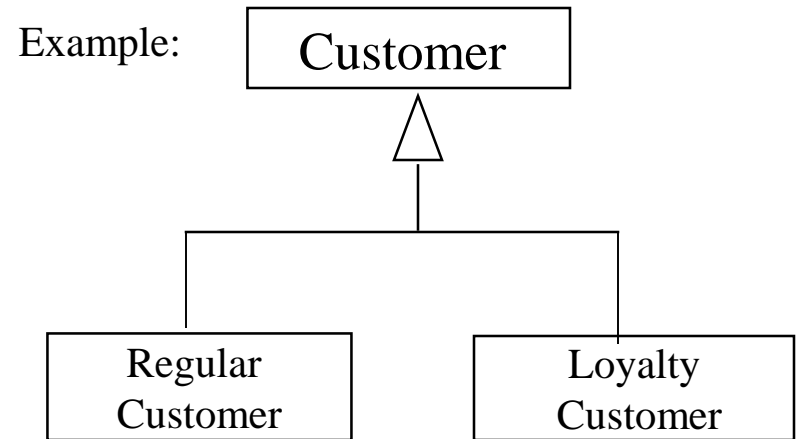
- There are two kinds of Relationships
 - Generalization (parent-child relationship)
 - Association (student enrolls in course)
- Associations can be further classified as
 - Aggregation
 - Composition

OO Relationships: Generalization

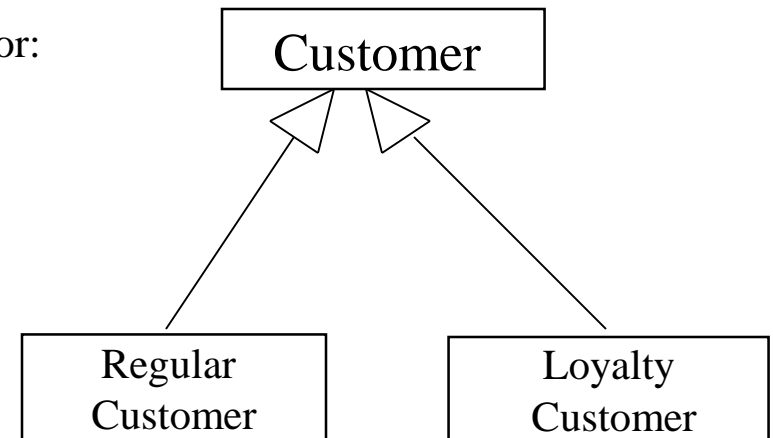


- Generalization expresses a **parent/child relationship** among related classes.

- **Used for abstracting details** in several layers



or:



OO Relationships: Association

- Represent relationship between instances of classes
 - Student enrolls in a course
 - Courses have students
 - Courses have exams
 - Etc.
- Association has two ends
 - Role names (e.g. enrolls)
 - Multiplicity (e.g. One course can have many students)
 - Navigability (unidirectional, bidirectional)

Association: Multiplicity and Roles



Multiplicity

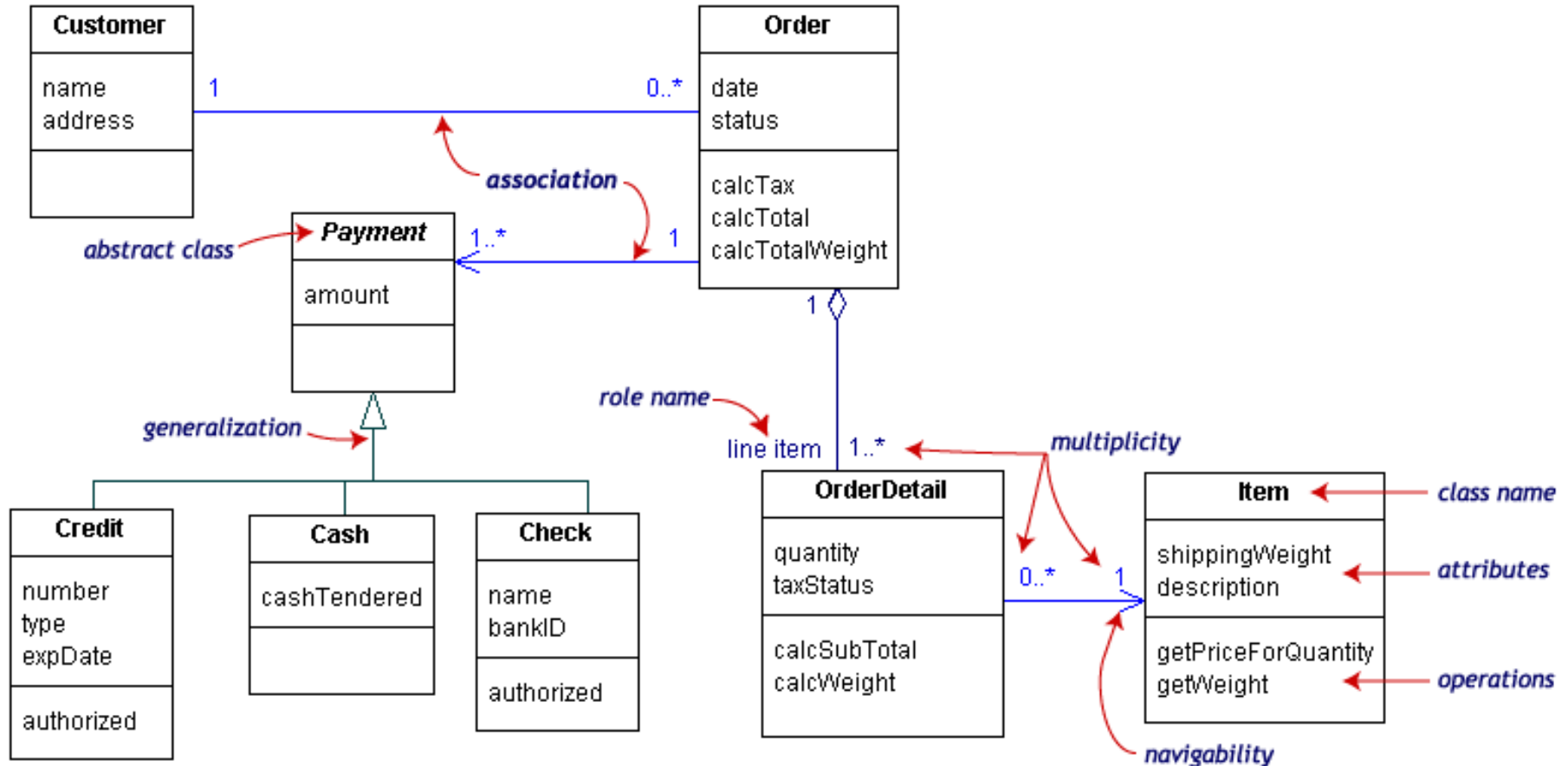
Symbol	Meaning
1	One and only one
0..1	Zero or one
M..N	From M to N (natural language)
*	From zero to any positive integer
0..*	From zero to any positive integer
1..*	From one to any positive integer

Role

Role

“A given university groups many people; some act as students, others as teachers. A given student belongs to a single university; a given teacher may or may not be working for the university at a particular time.”

UML Class Example



Sequence Diagram

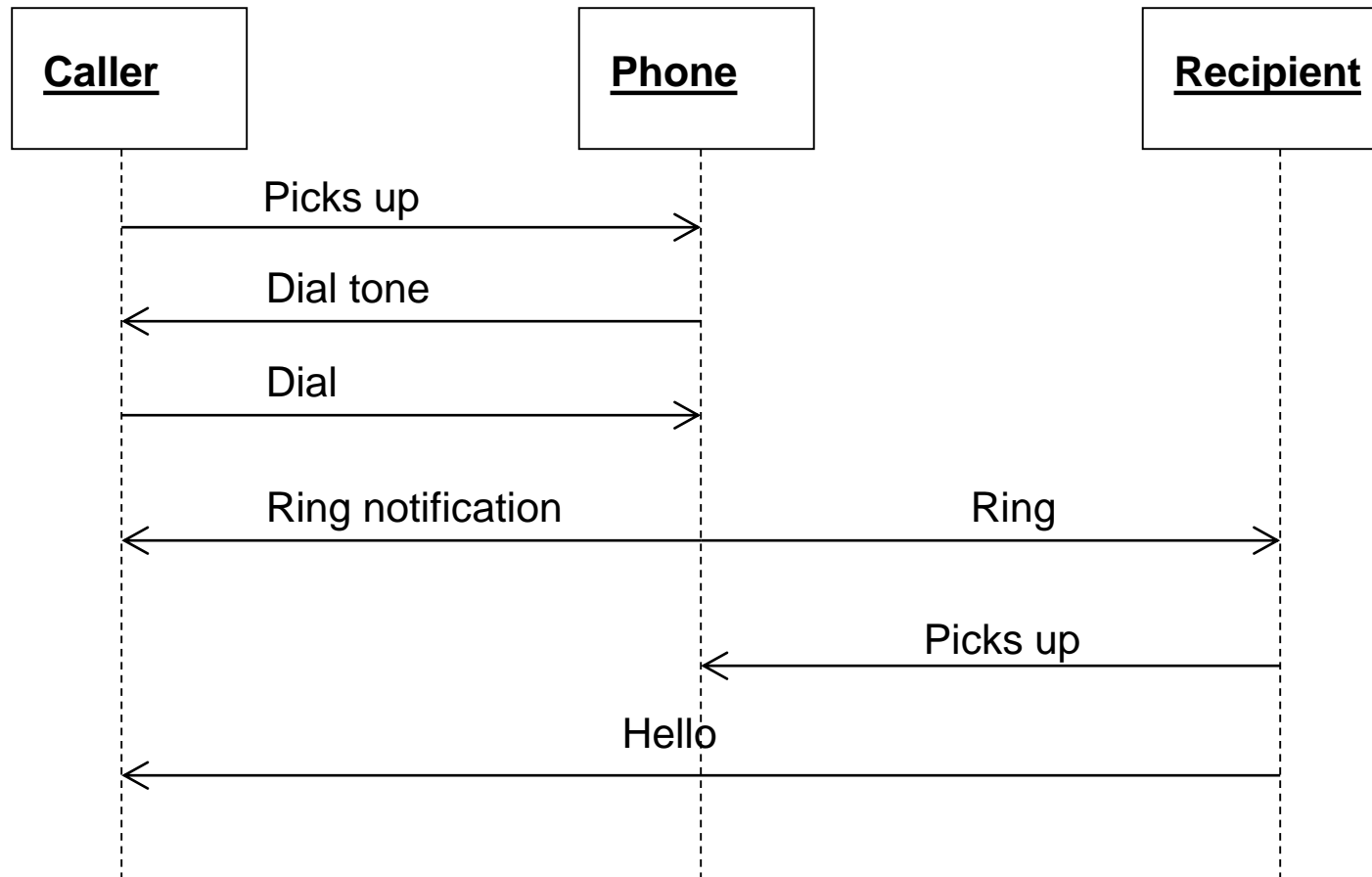
Sequence Diagram

- Captures the behavior of a single scenario in a use case
- Shows a number of example objects and messages that are passed between those objects within the use case
- The columns : Represent each object involved in the use case
- The life time : An object progresses from the top of the diagram to the bottom
- Clearly illustrates the calls between participants and the sequence of those calls
- Gives a good picture about which participants are doing which processing

Sequence Diagram

- Can exhibit centralized control or distributed control
 - In centralized control, one participant does all of the processing
 - In distributed control, processing is split among many participants
 - Distributed control gives more opportunities for using polymorphism rather than using conditional logic
- Use a sequence diagram when you want to look at the behavior of several objects within a single use case
- When not to use a sequence diagram
 - If you want to look at the behavior of a single object across many use cases, use a state diagram
 - If you want to look at the behavior of several objects across many scenarios, use an activity diagram

Sequence Diagram(make a phone call)

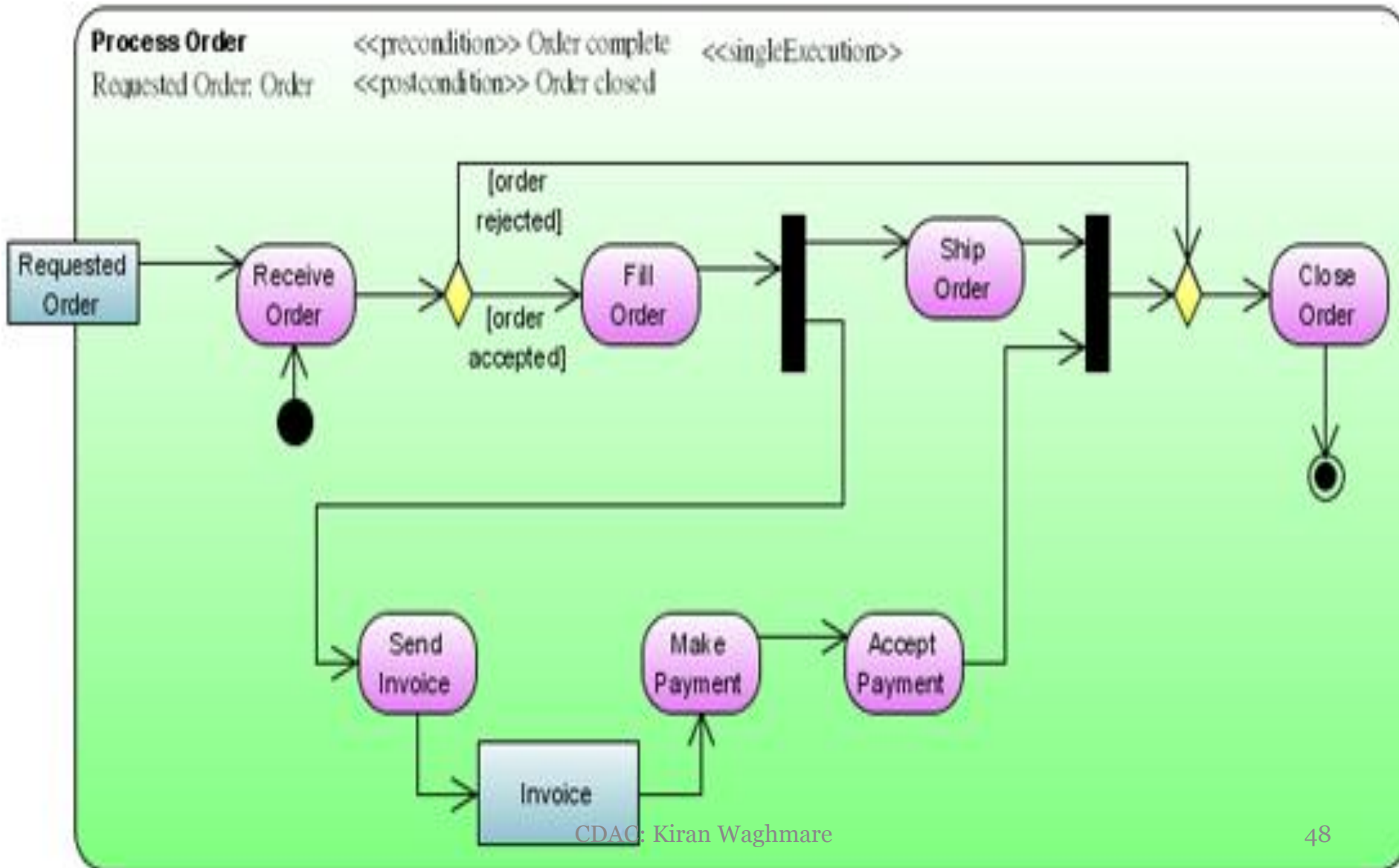


Activity Diagram

Activity diagram

- [UML 2 Activity diagrams](#) helps to describe the flow of control of the target system, such as the exploring complex business rules and operations, describing the use case also the business process. It is object-oriented equivalent of flow charts and data-flow diagrams (DFDs).

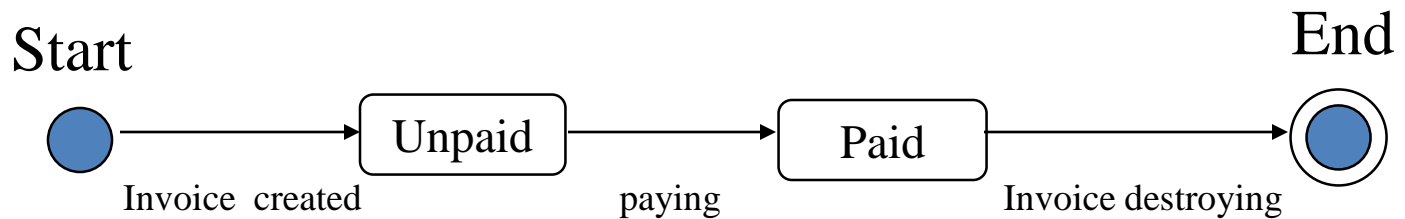
Activity diagram



State Diagrams

(Billing Example)

State Diagrams show the sequences of states an object goes through during its life cycle in response to stimuli, together with its responses and actions; an abstraction of all possible behaviors.



References

- <http://www.agilemodeling.com/>
- <http://www.visual-paradigm.com/VPGallery/diagrams/index.html>
- <http://bdn.borland.com/article/0,1410,31863,00.html>
- http://en.wikipedia.org/wiki/Unified_Modeling_Language
- http://pigseye.kennesaw.edu/~dbraun/csis4650/A&D/UML_tutorial/index.htm

UML Modeling Tools

- Rational Rose (www.rational.com) by IBM
- TogetherSoft Control Center, Borland
(<http://www.borland.com/together/index.html>)
- **ArgoUML** (free software) (<http://argouml.tigris.org/>)
OpenSource; written in java
- Others (http://www.objectsbydesign.com/tools/umltools_byCompany.html)

Reference

1. **UML Distilled:** A Brief Guide to the Standard Object Modeling Language
[Martin Fowler](#), [Kendall Scott](#)
2. IBM Rational
<http://www-306.ibm.com/software/rational/uml/>
3. Practical UML --- A Hands-On Introduction for Developers
http://www.togethersoft.com/services/practical_guides/umlonlinecourse/
4. Software Engineering Principles and Practice. Second Edition; Hans van Vliet.
5. <http://www-inst.eecs.berkeley.edu/~cs169/>

Online Movie Booking System

Example

FUNCTION SPECIFICATION

ADMIN :

In this module The Administrator is maintain the user Details, Movie details, Theater details and Check the number of seats available.

BOOKING TICKETS :

In this module The Users will search for the movie and then go for theater then booking the tickets online.

MEMBER :

In this modules The user can first Registration in enter the Personal details and User login and If you also want to update personal details. The user collect all information like upcoming Movies details, in advance ticket booking and Theater details.

BOOKING :

If you are book ticket by tell so you collect ticket before start show 30 minute otherwise cancel your booking

HARDWARE & SOFTWARE REQUIREMENT

Minimum Server Side Requirements :

Software	
Operating System :	Window7 or compatible
Web Server :	IE 6.0 +

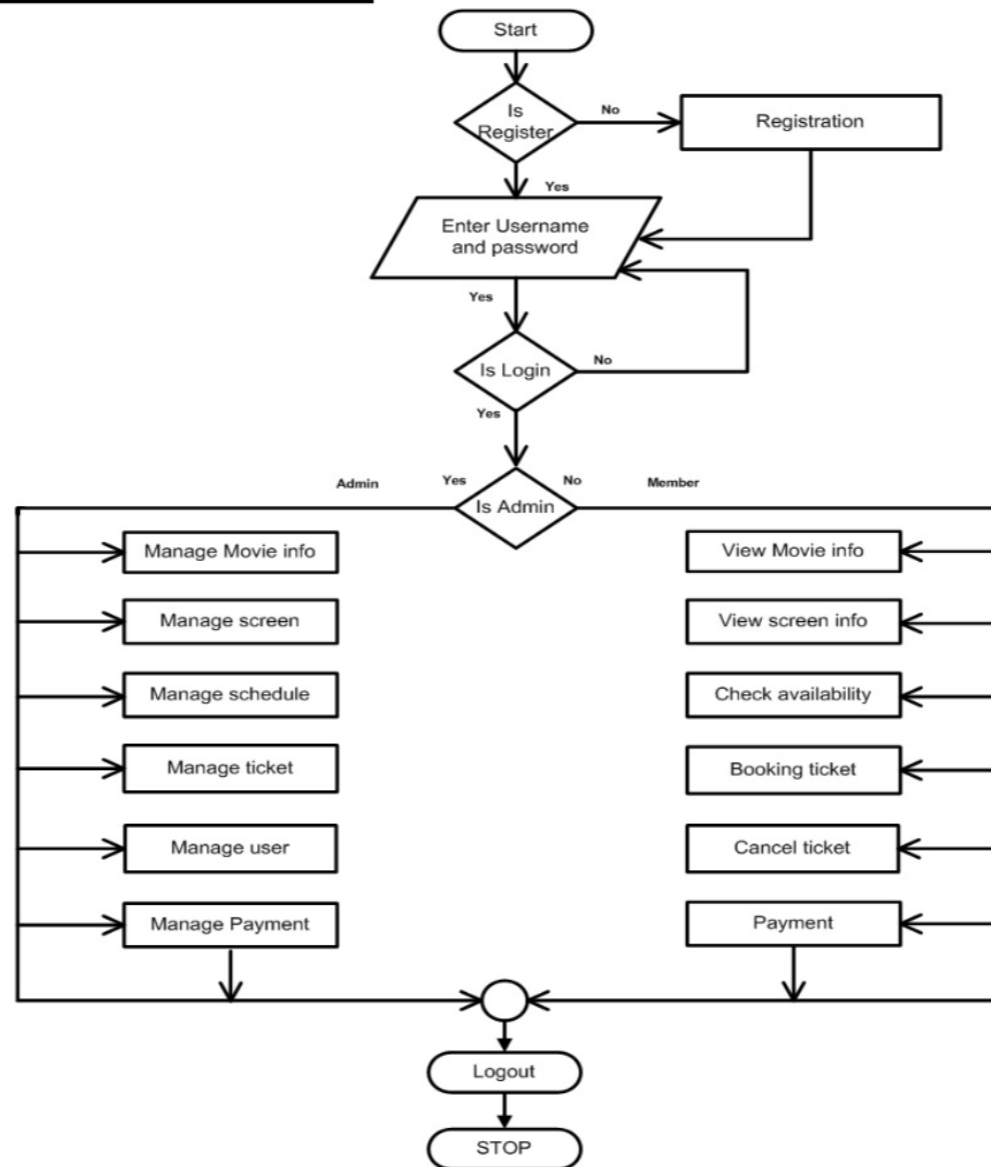
Hardware	
Processor :	1.5 MHz or above
Ram :	1 GB
Hard disk :	5 GB of free space on hard disk

Minimum Client Side Requirements :







Software	
Operating System :	Window XP or compatible
Web Server :	IE 6.0 +

Hardware	
Processor :	1.5MHz
Ram :	512 MB
Hard disk :	2 GB (only if downloading is needed)

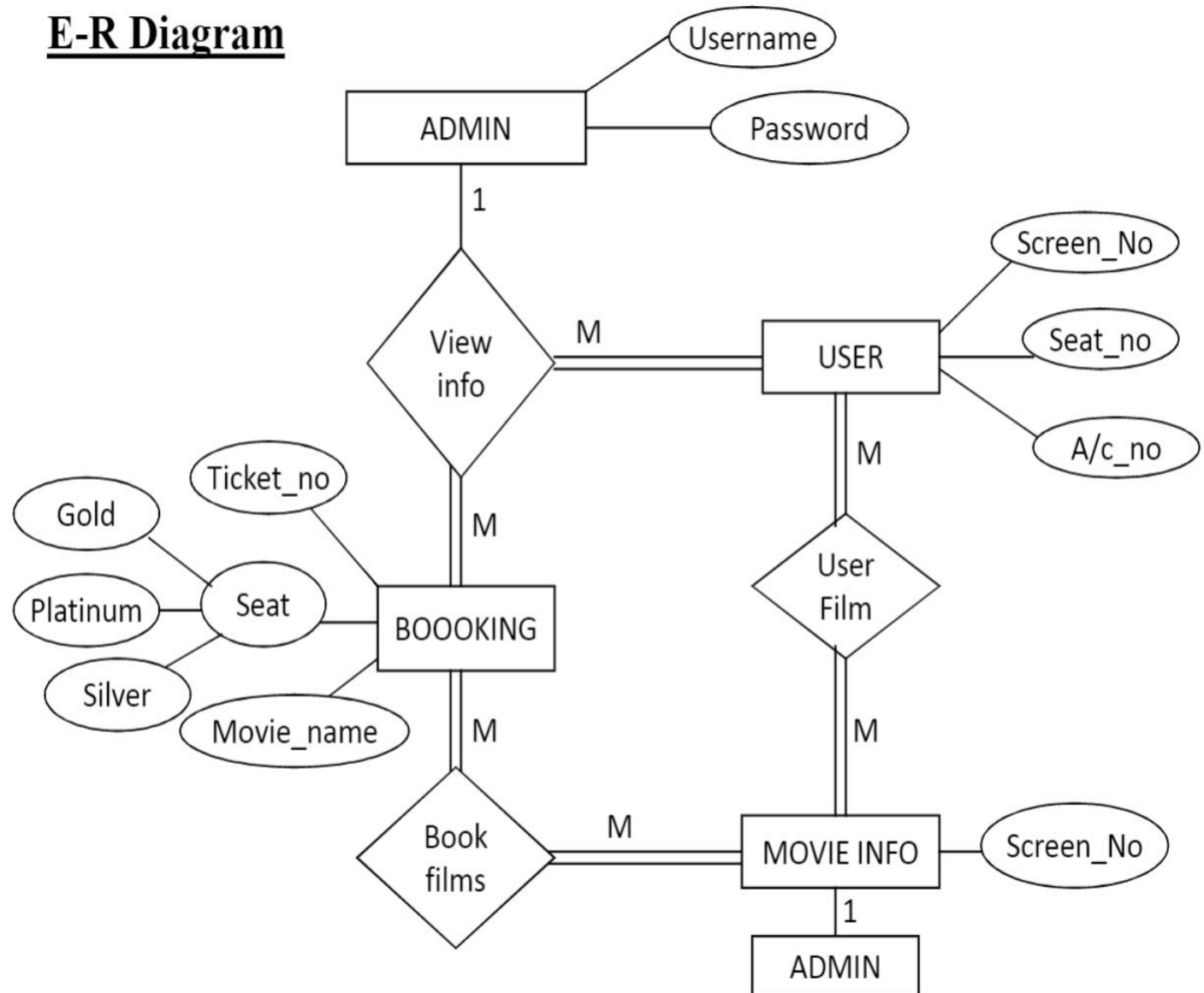
SYSTEM FLOW CHART



TIME LINE CHART

Task	Time Duration (In Days)						Total Days
	15	30	45	60	75	90	
Requirement Gathering & Analysis							15
Designing							15
Coding							35
Testing							15
Deployment & Implementation							10
Documentation							90

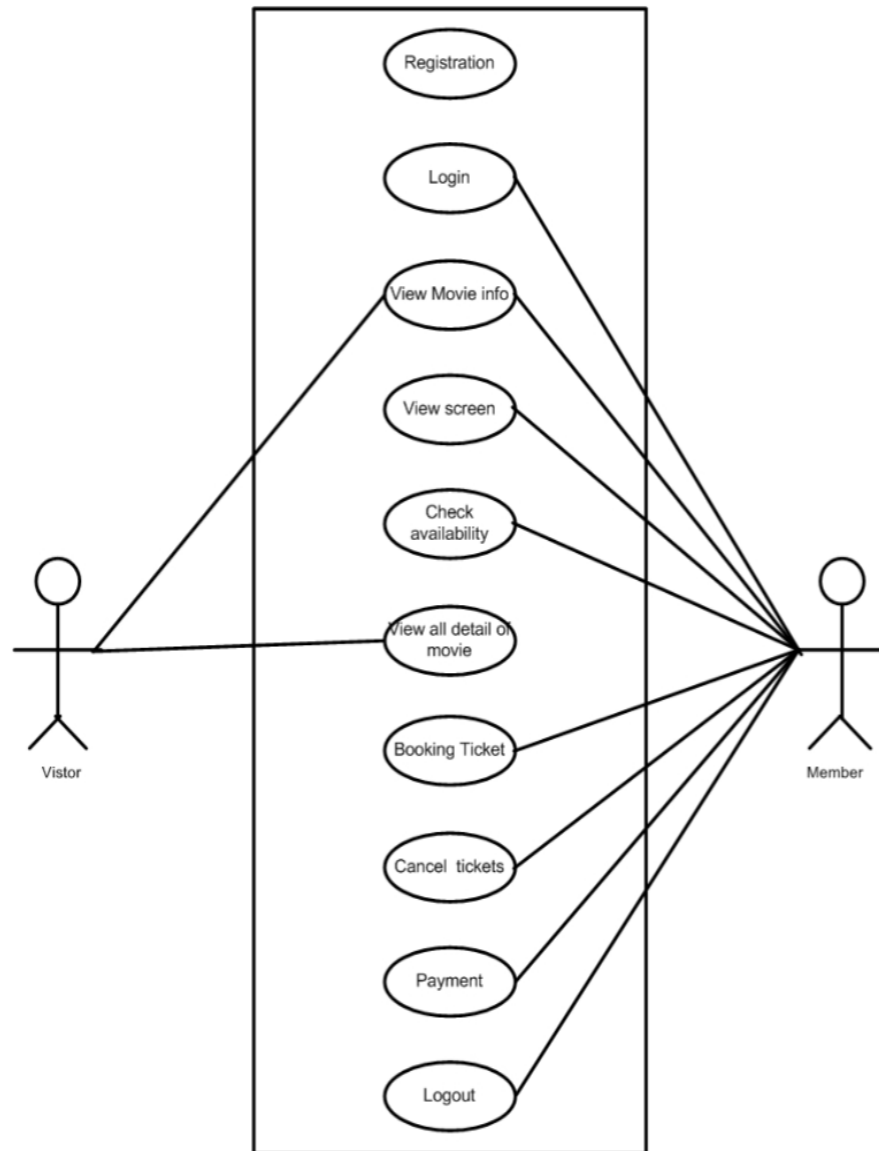
E-R Diagram



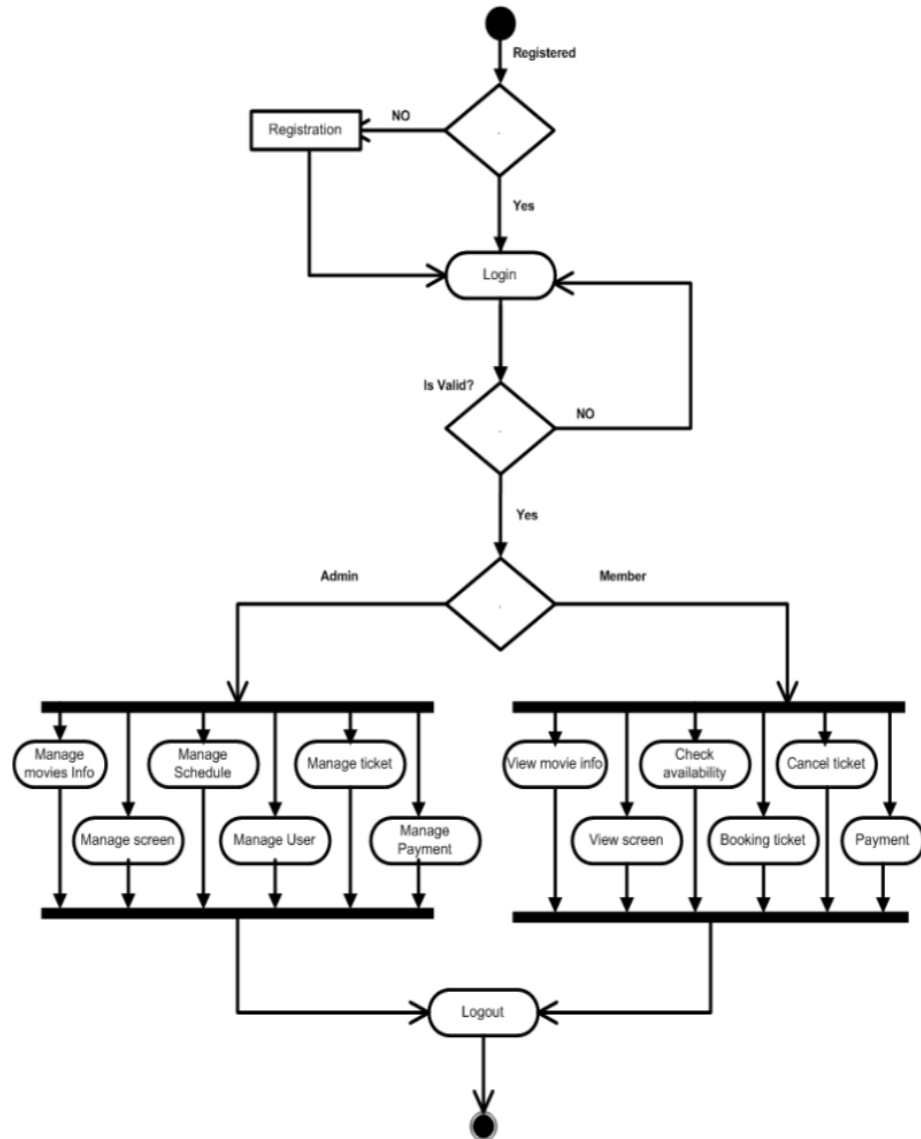
UML Diagram

❖ Use Case Diagram

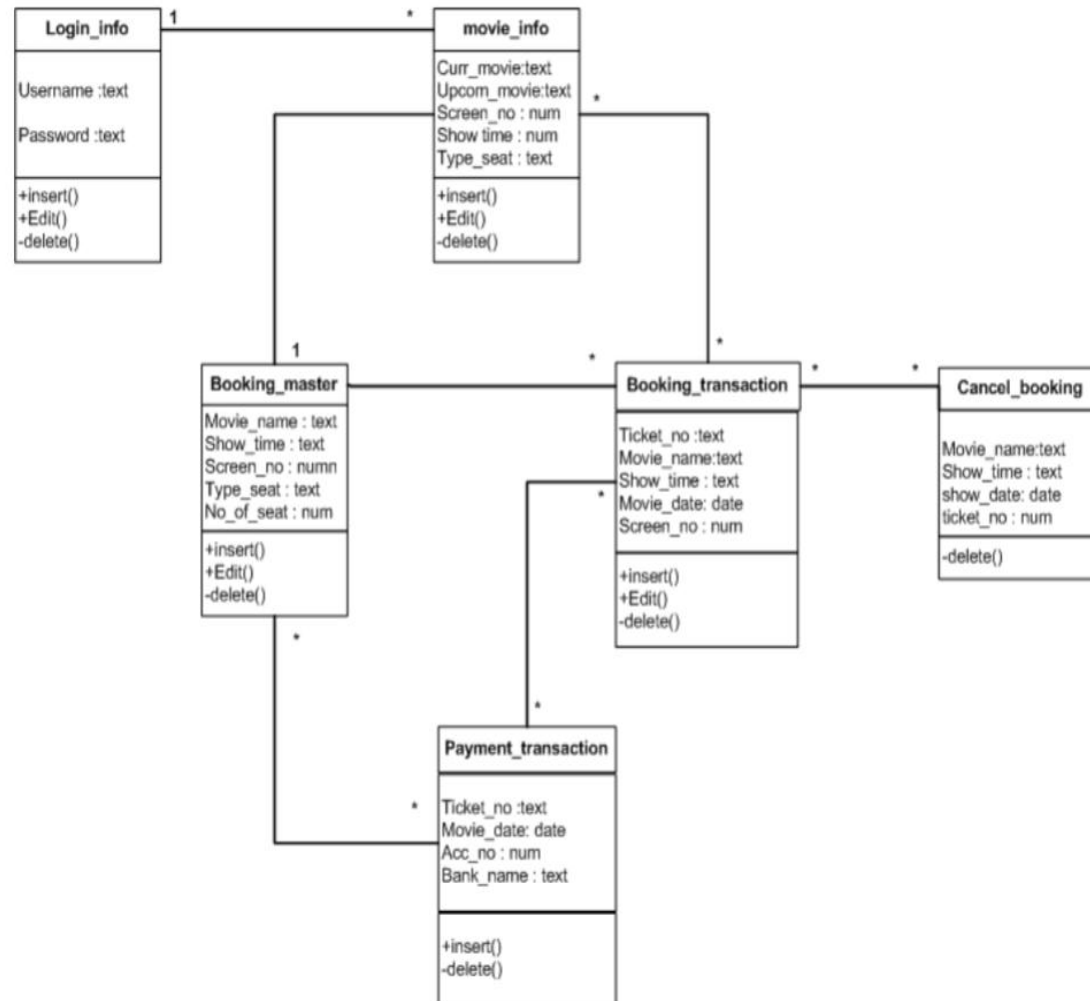




❖ Activity Diagram

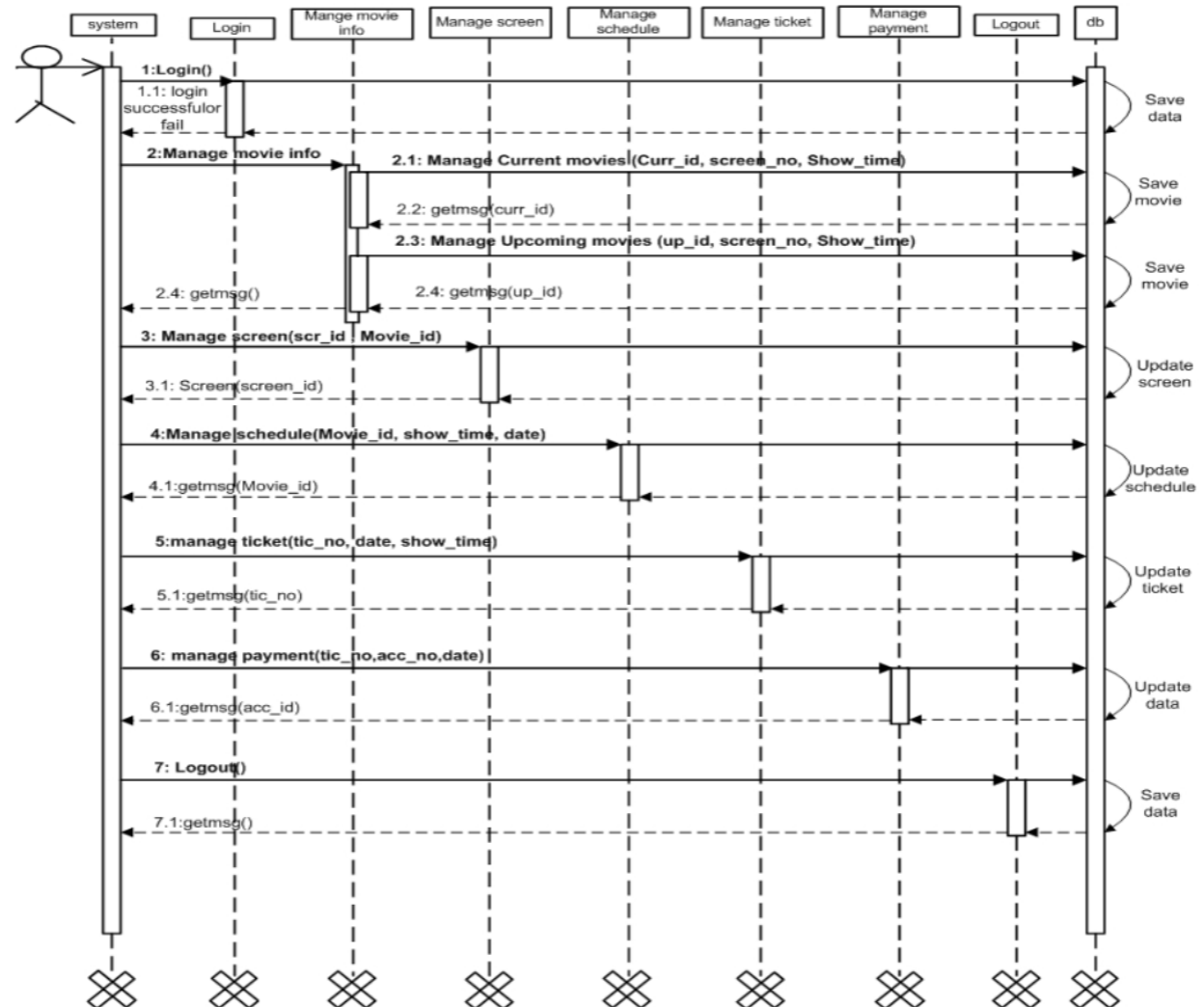


❖ Class Diagram



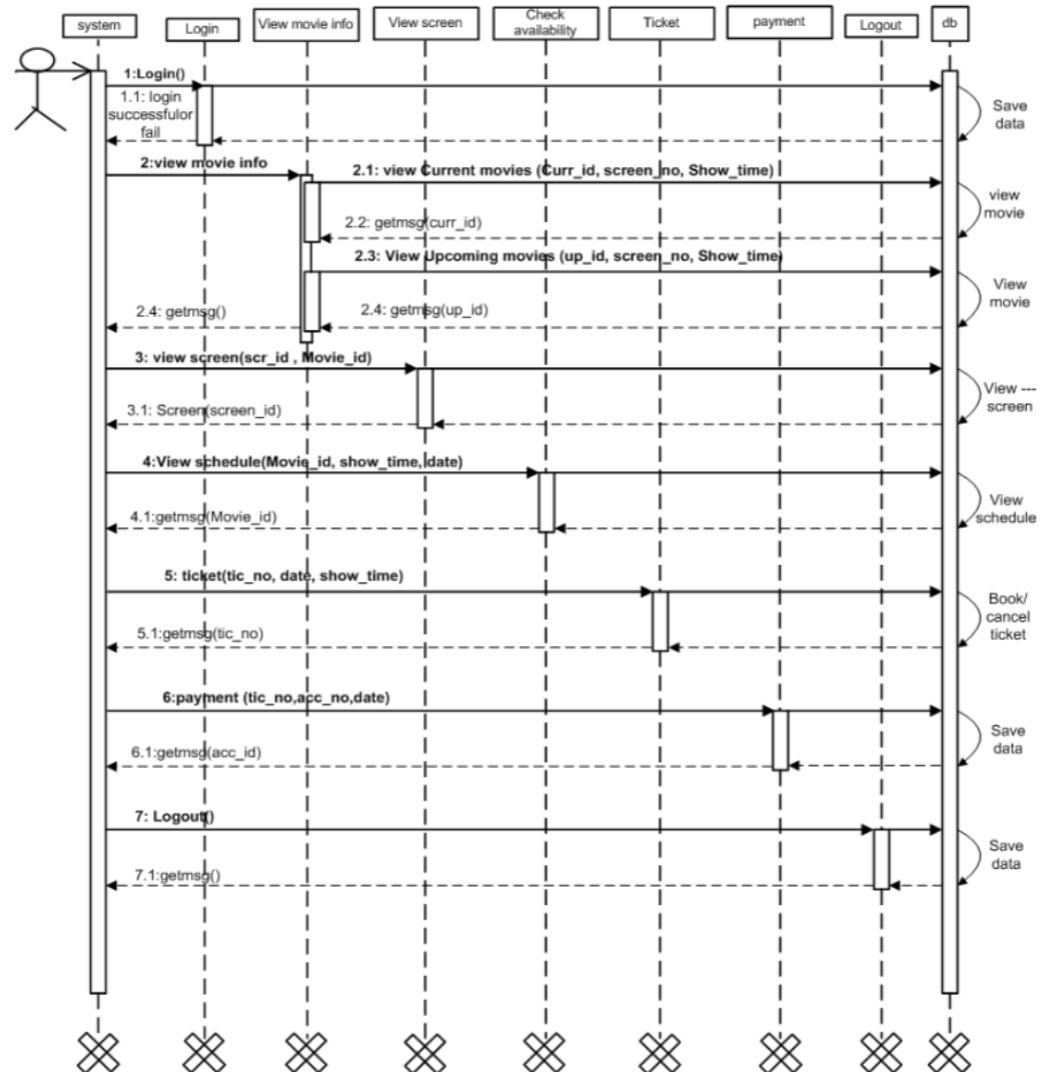
❖ SEQUENCE DIAGRAM

➤ Admin



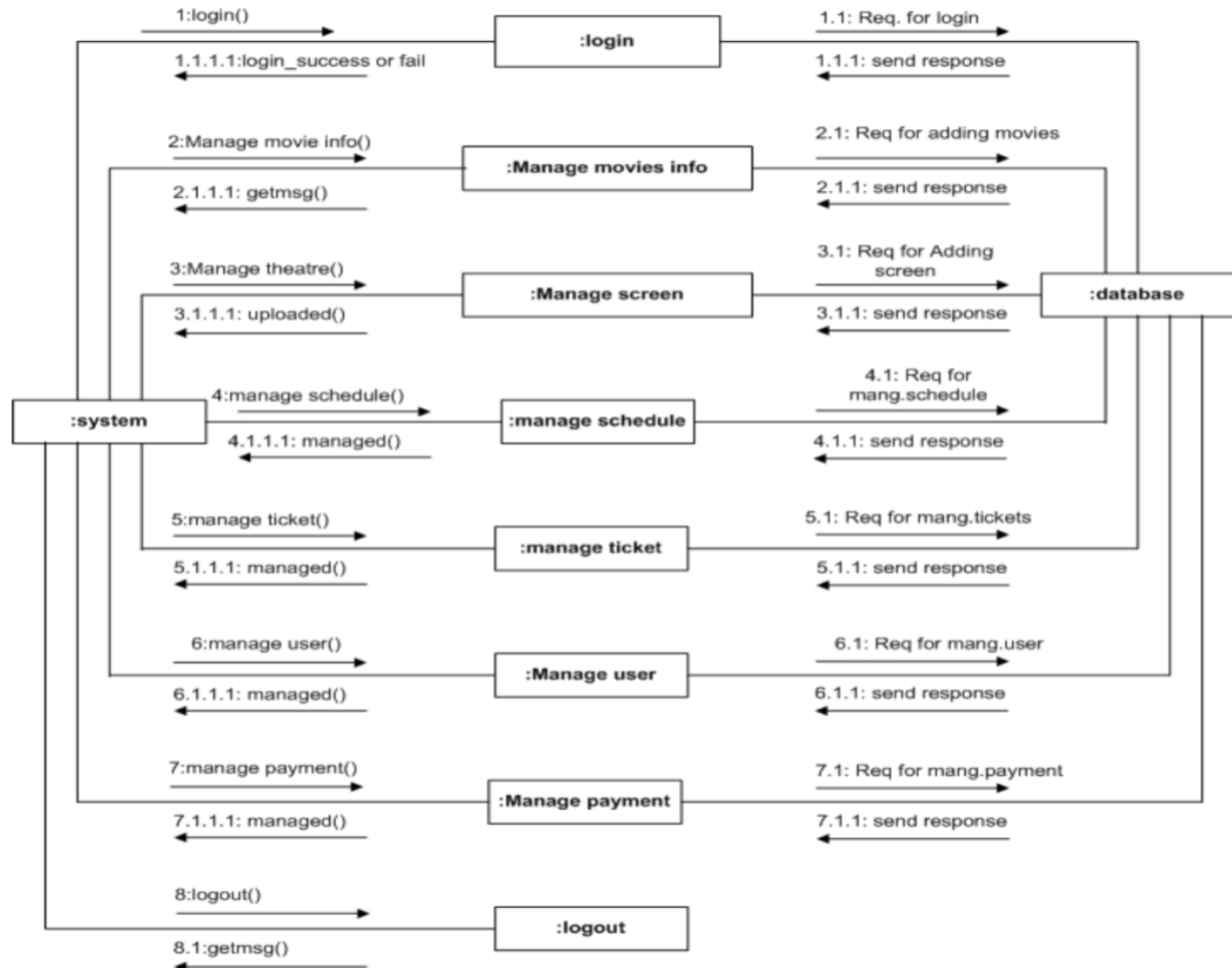
❖ SEQUENCE DIAGRAM

➤ Member



❖ COLLABORATION DIAGRAM

➤ Admin



❖ COLLABORATION DIAGRAM

➤ Member

