# Software Engineering(SE)

## CSC 601


**Subject Incharge**

Varsha Nagpurkar

Assistant Professor

Room No. 407

email: varshanagpurkar@sfit.ac.in

# Module-2 Syllabus

| | | | |
|---|---|---|---|
| 2.0 | | **Requirements Analysis and Modelling** | 08 |
| | 2.1 | Requirement Elicitation, Software requirement specification (SRS), Developing Use Cases (UML) | |
| | 2.2 | Requirement Model – Scenario-based model, Class-based model, Behavioural model. | |

# Key Ideas

The goal of the **analysis phase** is to truly understand the requirements of the new system and develop a system that addresses them.

The first challenge is collecting and integrating the information

The second challenge is finding the right people to participate.

# Elicitation & Analysis

The requirements process consists of two activities:

- Requirements **Elicitation**:
  - Definition of the system in terms understood by the ***customer*** ("Problem Description")
- Requirements **Analysis**:
  - *Technical* specification of the system in terms understood by the ***developer*** ("Problem Specification")

# Analysis Phase

This phase takes the general ideas in the system request and

- refines them into a detailed requirements definition
- functional models,
- structural models and
- behavioral models

**Final deliverable**: system proposal

- Includes revised project management deliverables,
  - feasibility analysis and
  - work plan

# Requirement Specification

- a statement of **what**
  - the system must do or
  - characteristics it must have
  - Written from business person perspective (what the system does?)
  - Later requirements become more technical (how the system will be implemented?)

# Functional vs. Nonfunctional

A ***functional requirement*** relates directly to a process the system has to perform or information it needs to contain (functions the system needs to have).

***Nonfunctional requirements*** refer to behavioral properties that the system must have, such as performance and usability.

# Functional Requirements

**Interface requirements**

- Field 1 accepts numeric data entry.
- Field 2 only accepts dates before the current date.

**Business Requirements**

- Data must be entered before a request can be approved.
- Clicking the Approve button moves the request to the Approval Workflow.

# Nonfunctional Requirements

D. Nonfunctional Requirements

1. Operational Requirements
   1.1. The system will operate in Windows and Macintosh environments
   1.2. The system will be able to read and write Word documents, RTF, and HTML
   1.3. The system will be able to import Gif, Jpeg, and BMP graphics files

2. Performance Requirements
   2.1. Response times must be less than 7 seconds
   2.2. The Inventory database must be updated in real time

3. Security Requirements
   3.1. No special security requirements are anticipated

4. Cultural and Political Requirements
   4.1. No special cultural and political requirements are anticipated

# Nonfunctional Requirements

- Performance – for example Response Time, Throughput, Utilization, Static Volumetric
- Scalability ,Capacity, Availability, Reliability, Recoverability, Maintainability, Serviceability
- Security ,Regulatory, Manageability, Environmental, Data Integrity, Usability ,Interoperability

YOUR TURN

## 3-1 Identifying Requirements

One of the most common mistakes made by new analysts is to confuse functional and nonfunctional requirements. Pretend that you received the following list of requirements for a sales system:

Requirements for Proposed System:
The system should...

1. be accessible to Web users.
2. include the company standard logo and color scheme.
3. restrict access to profitability information.
4. include actual and budgeted cost information.
5. provide management reports.
6. include sales information that is updated at least daily.
7. have 2-second maximum response time for predefined queries and 10-minute maximum response time for ad hoc queries.
8. include information from all company subsidiaries.
9. print subsidiary reports in the primary language of the subsidiary.
10. provide monthly rankings of salesperson performance.

**Questions:**

1. Which requirements are functional business requirements? Provide two additional examples.
2. Which requirements are nonfunctional business requirements? What kind of nonfunctional requirements are they? Provide two additional examples.

# Purpose Of requirement analysis

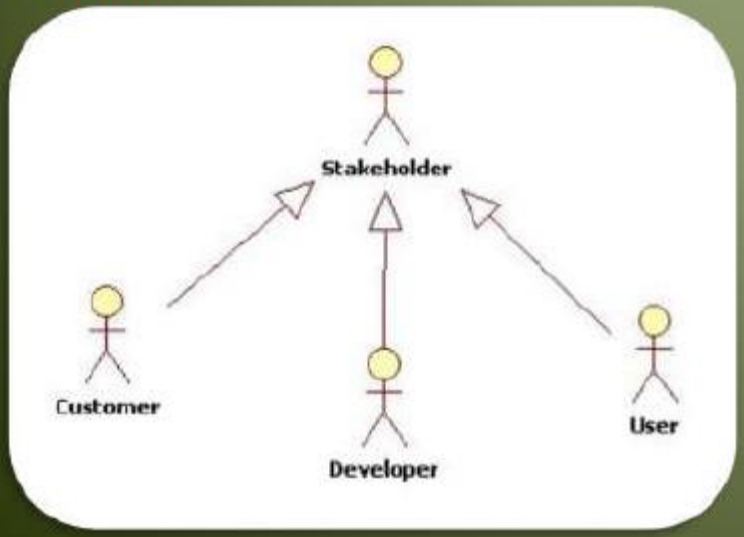
- *Scope of project*
- *Establish the user's expectations for system*

# Requirements Gathering

| | |
|---|---|
| Stakeholder Analysis | • Ensure that all stakeholder will considered<br>• Identifies all users and stakeholders who will influence system |
| Brain Storming | • It is used in identifying all possible solutions<br>• It will give good number of ideas from group |
| One to One Interview | • Sit with clients and ask questions<br>• In depth ideas from stakeholders |
| Group Interview | • Interview with more persons<br>• Can get hidden requirements |
| Document Analysis | • Prepare documentation<br>• Can help in As Is process design analysis |
| Prototyping | • Modern technique for gathering requirements<br>• Build the prototype ,its more practical and reduce risk |
| Joint Application Devlopment | • Arrange workshops for gathering requirements<br>• Contain two analyst facilitator and Scriber |

# Stakeholder Analysis

Stakeholder analysis identifies all the users and stakeholders who may influence or be impacted by the system. This helps ensure that the needs of all those involved are taken into account




## Benefits

1. Ensures that all relevant stakeholders are considered

1. All important stakeholders are captured, and yet that irrelevant actors are not included

## Drawbacks

There is a danger that too much time is spent on identifying roles and relationships, and the team is swamped with data.

# Brainstorming

**It is utilized in requirements elicitation to gather good number of ideas from a group of people. Usually brainstorming is used in identifying all possible solutions to problems and simplifies the detail of opportunities. It casts a broad net, determining various discreet possibilities.**


## Basic Rules

1. Start out by clearly stating the objective of the brainstorming session.
2. Generate as may ideas as possible.
3. Let your imagination soar.
4. Do not allow criticism or debate while you are gathering information.
5. Once information is gathered, reshape and combine ideas.

# Brainstorming

## Benefits

1. Generate a variety of ideas in a short time
2. Produce new and creative ideas

## Risks

1. The risk of having a bad session
2. Making staff scared to voice their ideas because they were criticized in the session
3. Problems normally occur if
   you only use traditional b rainstorming techniques.

# One On One Interview

The most common technique for gathering requirements is to sit down with the clients and ask them what they need. The discussion should be planned out ahead of time based on the type of requirements you're looking for

## Benefits

- Privacy of everyone
- in-depth a stakeholder's thoughts and get his or her perspective

## Risks & Drawbacks

- Time Consuming
- Misunderstandings

# Group Interview

If there are more then one person during interview usually 2 or 4 these people must be on some level must be on some level less time required

## Benefits

- we can get hidden requirements
- uncover a richer set of requirements in a shorter period of time
- Uncover ambiguities

## Risks & Drawbacks

- Not relaxed environment
- Conflicts
- The allotted time have been exhausted

# Requirements Elicitation Process

- Elicitation is a task that helps the customer to define what is required
- Requirement Elicitation Methods
  - Collaborative Requirements Gathering
  - Quality function deployment
  - User scenarios
  - Elicitation Work Products

## Collaborative Requirements Gathering

- In this collaborative,team-oriented approach to requirements gathering,a team of stakeholders and developers work together
  - To identify the problem
  - Propose elements of the solution
  - Negotiate different approaches
  - Specify a preliminary set of solution requirements
- The meeting for collaborative requirements gathering is conducted to discuss all above issues

*

# Guidelines for conducting a collaborative requirements gathering meeting are

- Meetings are conducted and attended by both software engineer and customers(along with other interested stakeholders)
- Rules for preparation and participation are established.
- An agenda is suggested that is formal enough to cover all important points but informal enough to encourage the free flow of ideas.
- A "facilitator" (can be a customer, a developer, or an outsider) controls the meeting.
- A "definition mechanism" (can be work sheets, flip charts, or wall stickers or an electronic bulletin board, chat room, or virtual forum) is used.

*

# Quality Function Deployment

- *Quality function deployment* (QFD) is a quality management technique that translates the needs of the customer into technical requirements for software. QFD "concentrates on maximizing customer satisfaction from the software engineering process"

# 3 types of QFD requirements

- Normal requirements:-*Normal requirements* identify the objectives and goals that are stated for a product or system during meetings with the customer. If these requirements are present, the customer is satisfted.
- Examples of normal requirements might be requested types of graphical displays,specific system functions,and defined levels of performance

*

# Expected requirements

- These requirements are implicit to the product or system and may be so fundamental that the customer does not explicitly state them
- Their absence will be a cause for significant dissatisfaction
- Examples of expected requirements are ease of human/machine interaction,overall operational correctness and reliability,and ease of software installation

*

# Exciting requirements

- These requirements reflect features that go beyond the customer's expectations and prove to be very satisfying when present
- For example,word processing software is requested with standard features
- The delivered product contains a number of page layout capabilities that are quite pleasing and unexpected

*

# User Scenaios

- It is difficult to move into more technical software engineering activities until the software team understands how these functions and features will be used by different classes of end users
- To accomplish this,developers and users can create a set of scenarios that identify a thread of usage for the system to be constructed
- The scenarios,often called use-cases

# Elicitation Work product

- The work product produced by requirement elicitation depend upon the size of the system or the system to be built
- The information produced as a consequence of requirements gathering includes
  - a statement of need and feasibility
  - a bounded statement of scope for the system or product
  - a list of customers, users, and other stakeholders who participated in requirements elicitation,
  - a descrip- tion of the system's technical environment

# Elicitation Work product

- The information produced as a consequence of requirements gathering includes
  - a list of requirements (prefera- bly organized by function) and the domain constraints that applies to each
  - a set of usage scenarios that provide insight into the use of the system or product under different operating conditions,
  - any prototypes developed to better define requirements

# Software Requirements Specification(SRS)

- SRS is the official statement of what the system developers should implement.
- SRS is a complete description of the behaviour of the system to be developed
- SRS should include both a definition of user requirements and a specification of the system requirements.
- The SRS fully describes what the software will do and how it will be expected to perform.

# Purpose of SRS

- The SRS precisely defines the software product that will be built.
- SRS used to know all the requirements for the software development and thus that will help in designing the software.
- It provides feedback to the customer.

# SRS Format

- All the requirements for the system have to be included in a document that is clear and concise
- For this, it is necessary to organize the requirements document as sections and subsection.
- Here, we discuss the organization proposed in the IEEE guide to software requirements specifications [IEEE87, IEEE94].

# SRS Format

- The details which are included in SRS depend on the type of system that is being developed and the type of the development process
- A number of large organizations,such as IEEE have defined standards for software requirements document
- The most widely used standard is IEEE/ANSI 830-1998

# SRS Format

**1. Introduction**

- 1.1 Purpose
- 1.2 Scope
- 1.3 Definitions, Acronyms and Abbreviations
- 1.4 Reference
- 1.5 Overview

**2. Overall Description**

- 2.1 Product Perspective
- 2.2 Product Functions
- 2.3 User Characteristics
- 2.4 General Constraints
- 2.5 Assumption and dependencies

# SRS Format

**3. Specific Requirements**

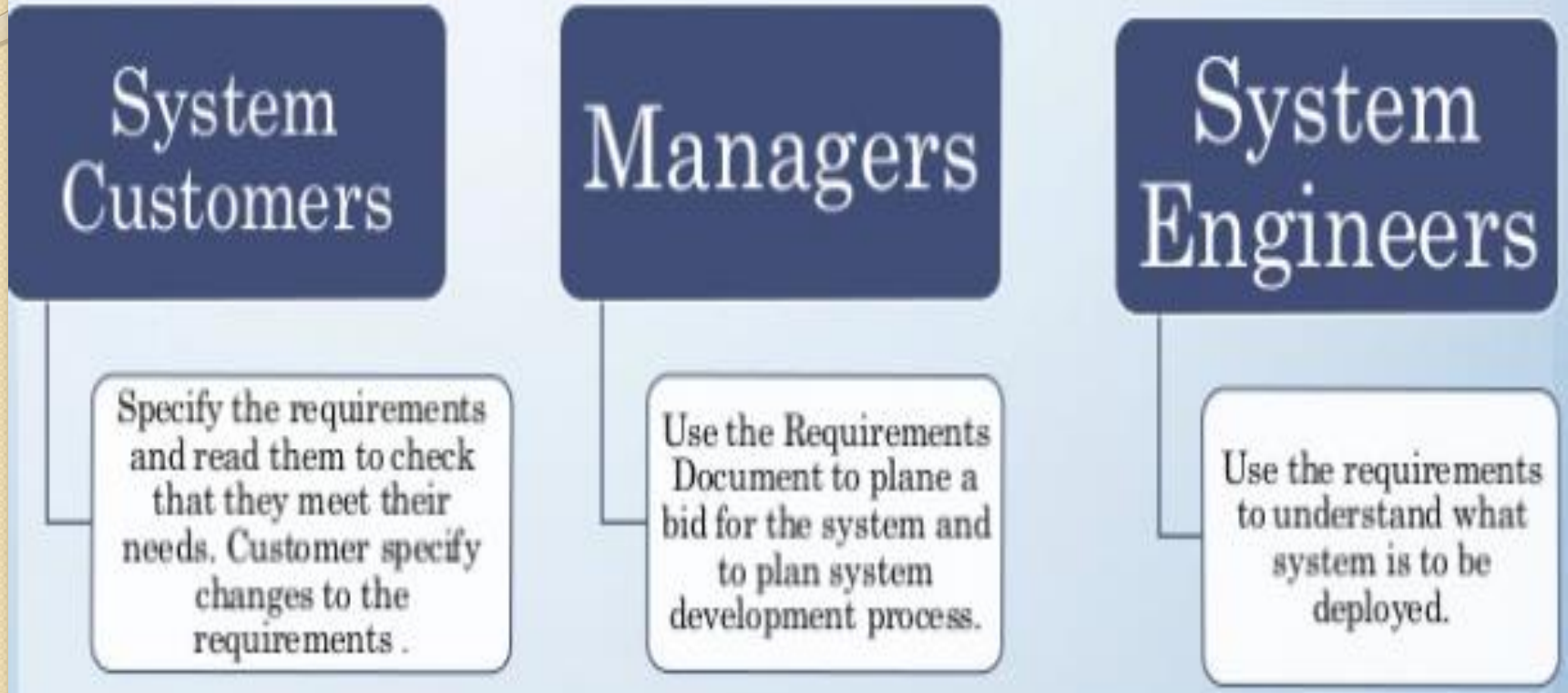
- 3.1 External interface Requirements.
  - 3.1.1 User interfaces.
  - 3.1.2 Hardware interfaces.
  - 3.1.3 Softwaer interfaces.
  - 3.1.4 Communications interfaces.
- 3.2 Functional Requirements
- 3.3 Performance Requirements
- 3.4 Design Constraints
- 3.5 Attributes
- 3.6 Other Requirements

*

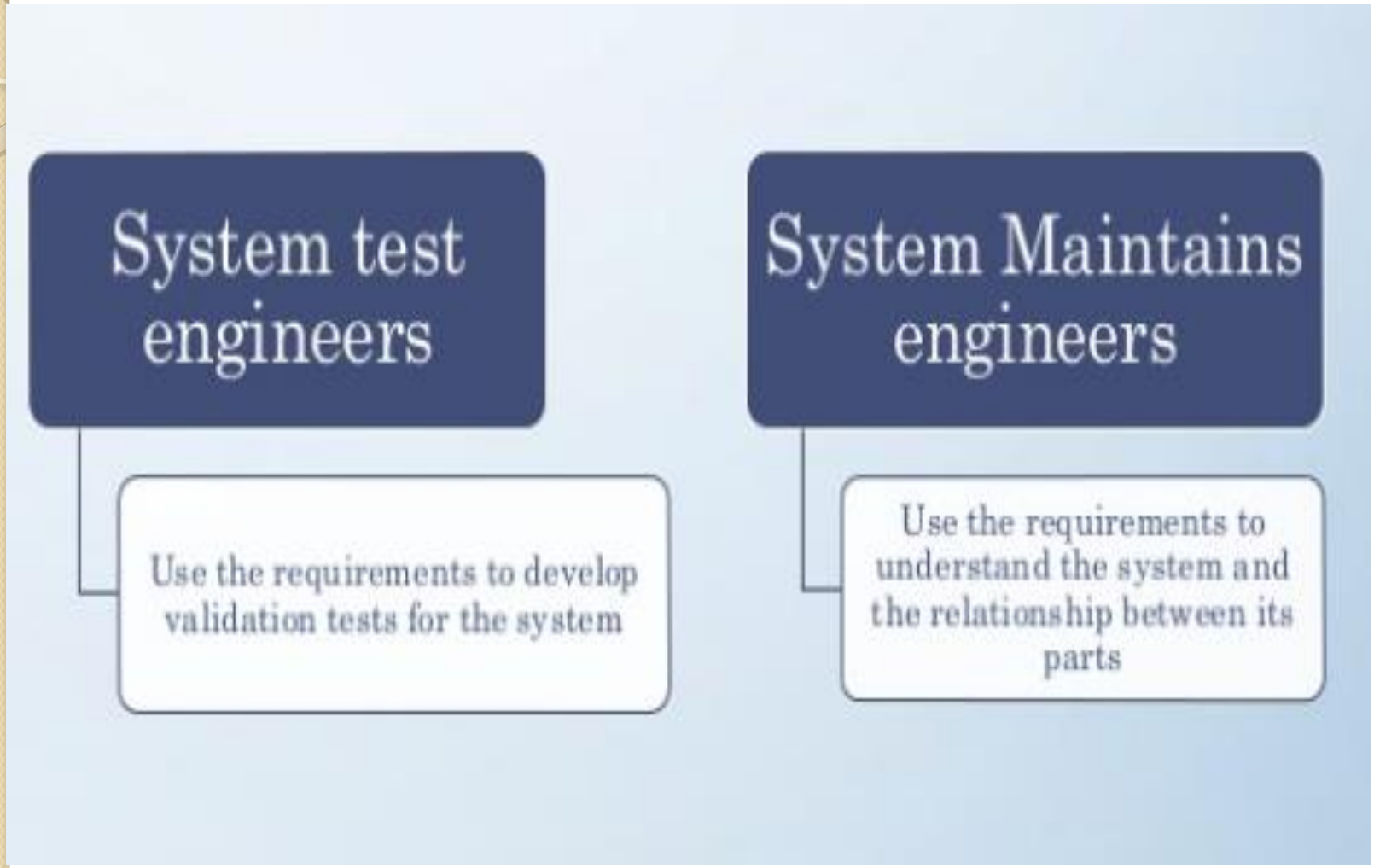
# Product Perspective

- As an external view on the **product**, the **product perspective** defines how the **product** contributes to fulfilling stakeholder needs and adjacent systems assumptions.
- They define the **product's** functional behavior, qualities, and constraints in response to stakeholder needs or adjacent systems' assumptions.

# Types of reader for requirement specification



# Types of Reader for Requirement Specification



# Characteristics of an SRS

- Complete-All the project functionalities should be recorded by SRS
- Consistent-SRS should be consistent
- Accurate-SRS defines systems functionality in real world.we need to record requirement very carefully
- Modifiable-Logical and hierarchical modification should be allowed in SRS

# Characteristics of an SRS

- Ranked-Requirement should be ranked using different factors like stability,security,ease or difficulty
- Testable-The requirement should be realistic and able to implement
- Traceable-Every requirement we must be able to uniquely identify
- Unambiguous-Statement in the SRS document should have only one meaning
- Valid-All the requirements should be valid

# **U**nified **M**odeling **L**anguage

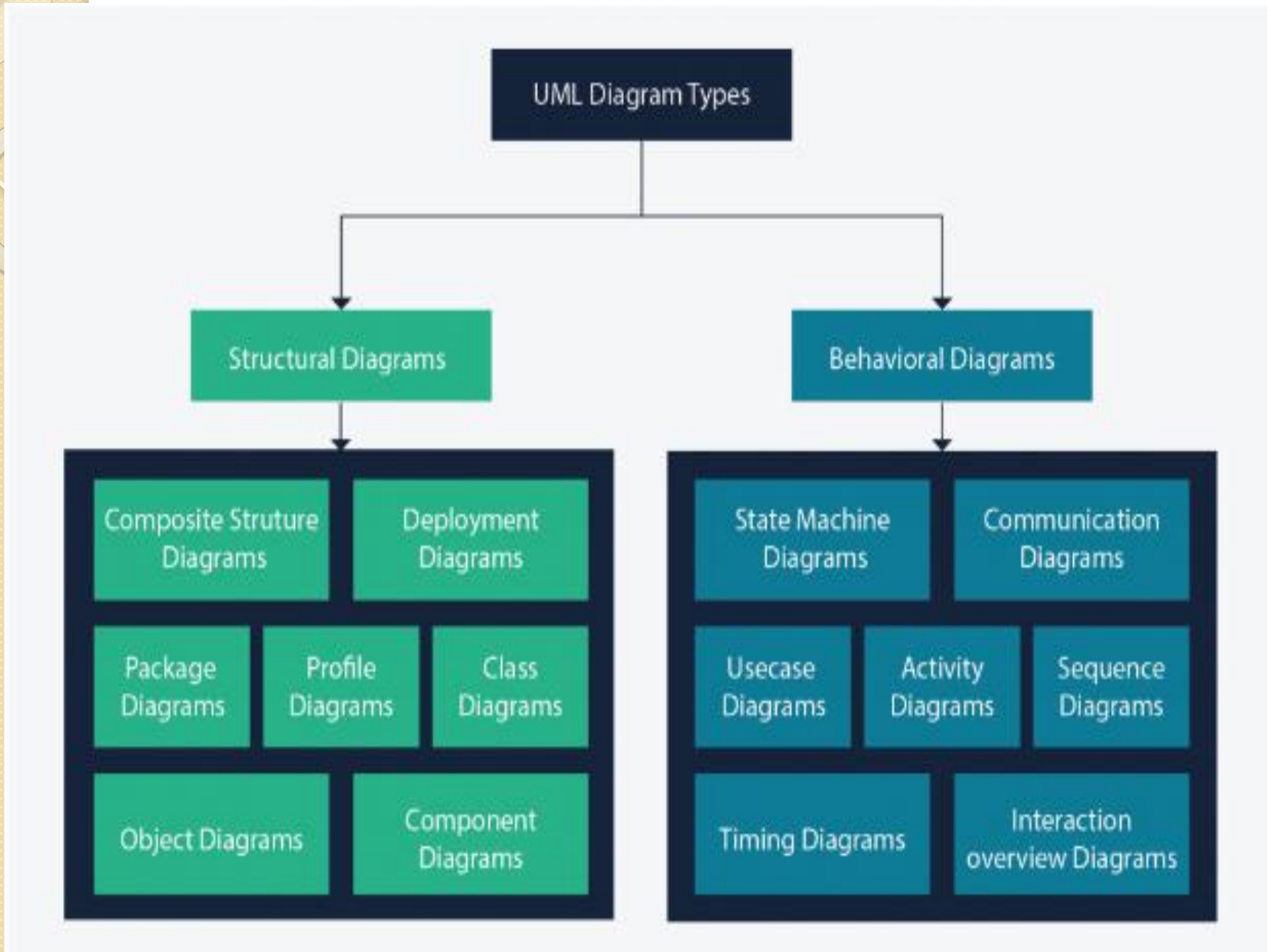
# UML

- UML stands for **U**nified **M**odeling **L**anguage.
- It's a rich language to model software solutions, application structures, system behavior and business processes.

# UML Diagram Types

- There are two main categories;
  - **Structure diagrams**

    **These diagrams** show different objects in a system
  - **Behavioral diagrams**.
    - **These diagrams** show what should happen in a system.
    - They describe how the objects interact with each other to create a functioning system.


UML Diagram Types
Structural Diagrams
Behavioral Diagrams
Composite Struture Diagrams
Deployment Diagrams
Package Diagrams
Profile Diagrams
Class Diagrams
Object Diagrams
Component Diagrams
State Machine Diagrams
Communication Diagrams
Usecase Diagrams
Activity Diagrams
Sequence Diagrams
Timing Diagrams
Interaction overview Diagrams

# Use Case Diagram

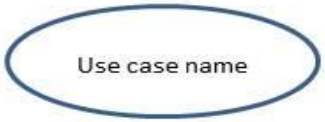
- **Use Case Diagram** captures the system's functionality and requirements by using actors and use cases.
- Use Cases model the services, tasks, function that a system needs to perform.
- Use cases represent high-level functionalities and how a user will handle the system.

# Use Case Diagram

- Use case diagrams capture the dynamic behaviour of a live system.
- It models how an external entity interacts with the system to make it work.
- It's a great starting point for any project discussion because you can easily identify the main actors involved and the main processes/functionalities of the system

# Symbols used in Use Case Diagram


Actor

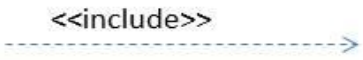



Use case

Generalization symbol used between actors and between use cases



Association between actor and use case




Include relationship between use cases


Extend relationship between use cases

**Symbols in a use case diagram**

# Use Cases

What is a Use Case

-A formal way of representing how a business system interacts with its environment

-Illustrates the activities that are performed by the users of the system

-A **scenario-based** technique in the UML

Scenario is A sequence of actions a system performs that yields a valuable result for a particular actor.

# Actor

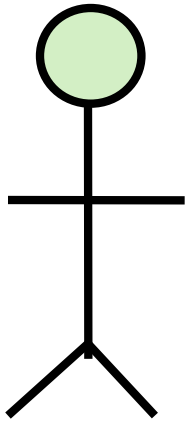
What is an Actor?

- A **user** or **outside** system that interacts with the system being designed in order to obtain some value from that interaction

Use Cases describe **scenarios** that describe the interaction between users of the system (the actor) and the system itself.

# Actors

- An Actor is outside or external the system.
- It can be a:
  - Human
  - Peripheral device (hardware)
  - External system or subsystem
  - Time or time-based event
- Represented by stick figure



# Use Case Diagram

- **Use case diagrams** describe what a system does from the standpoint of an external observer. **The emphasis is on *what* a system does rather than *how*.**

- Use case diagrams are closely connected to scenarios. A **scenario** is an example of what happens when someone interacts with the system.

# Use Cases

The picture below is a **Make Appointment** use case for the medical clinic.

The actor is a **Patient**. The connection between actor and use case is a **communication association** (or **communication** for short).

**Actors are stick figures. Use cases are ovals.**
**Communications are lines that link actors to use cases.**

# Use Case Relationships

| Relationship | Symbol | Meaning |
|---|---|---|
| Communicates |  | An actor is connected to a use case using a line with no arrowheads. |
| Includes |  | A use case contains a behavior that is common to more than one other use case. The arrow points to the common use case. |
| Extends |  | A different use case handles exceptions from the basic use case. The arrow points from the extended to the basic use case. |
| Generalizes |  | One UML "thing" is more general than another "thing." The arrow points to the general "thing." |

# Use Case Diagram

## Other Types of Relationships for Use Cases

- Generalization
- Include
- Extend

# Use Case Diagram for student management system:

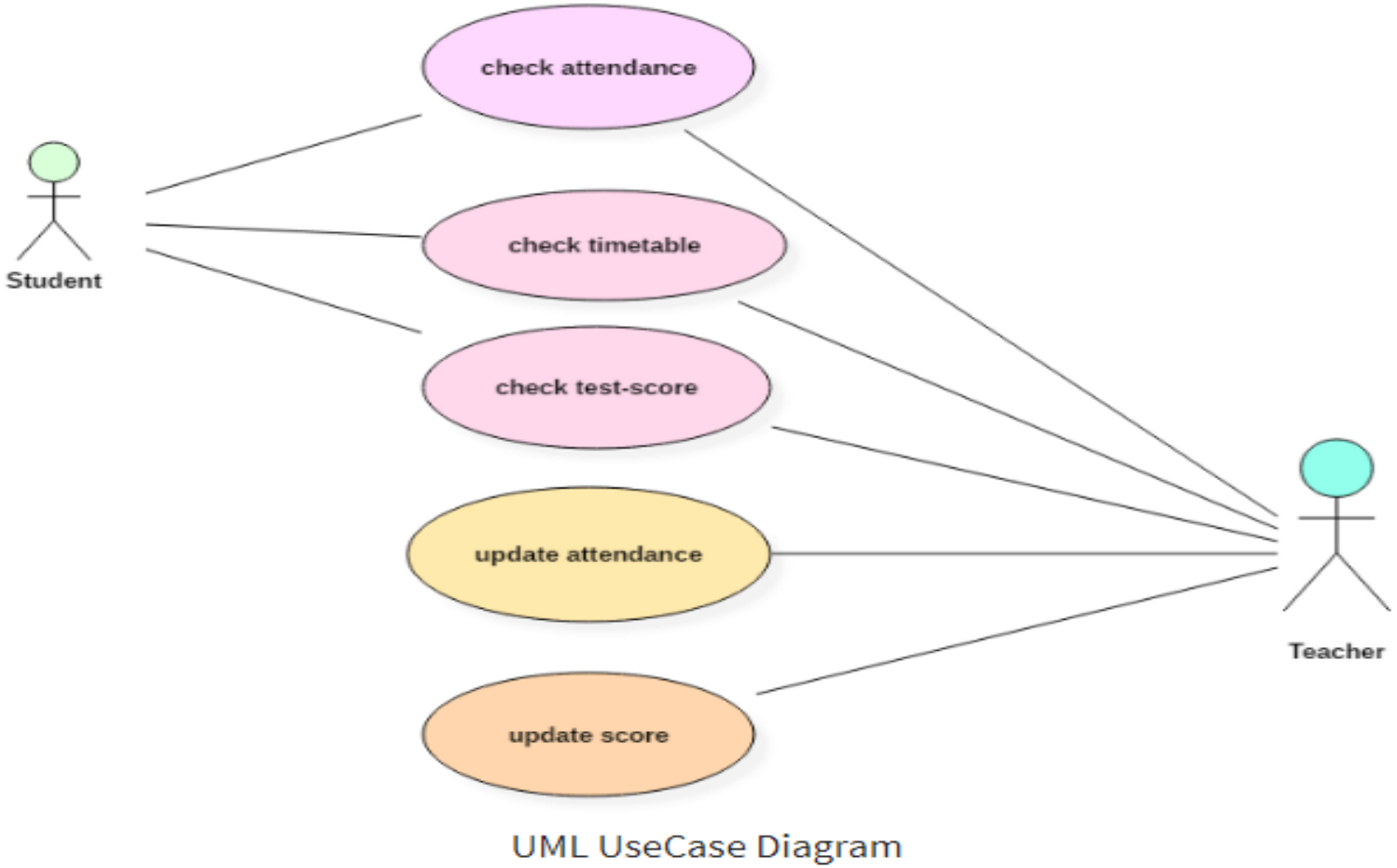



Fig.1 Use Case Diagram for Student Management System

# Use Case Diagram for Bank ATM System

- Step-1

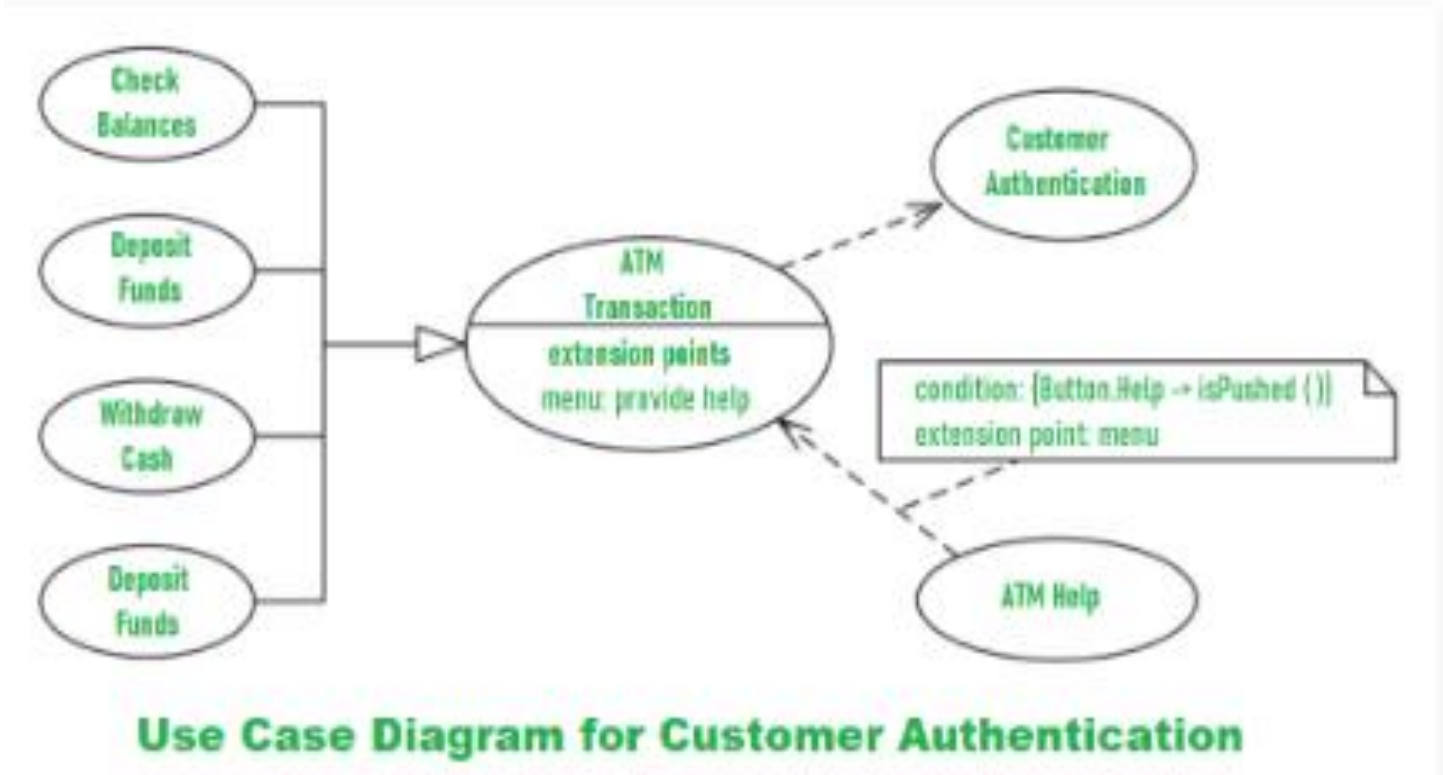
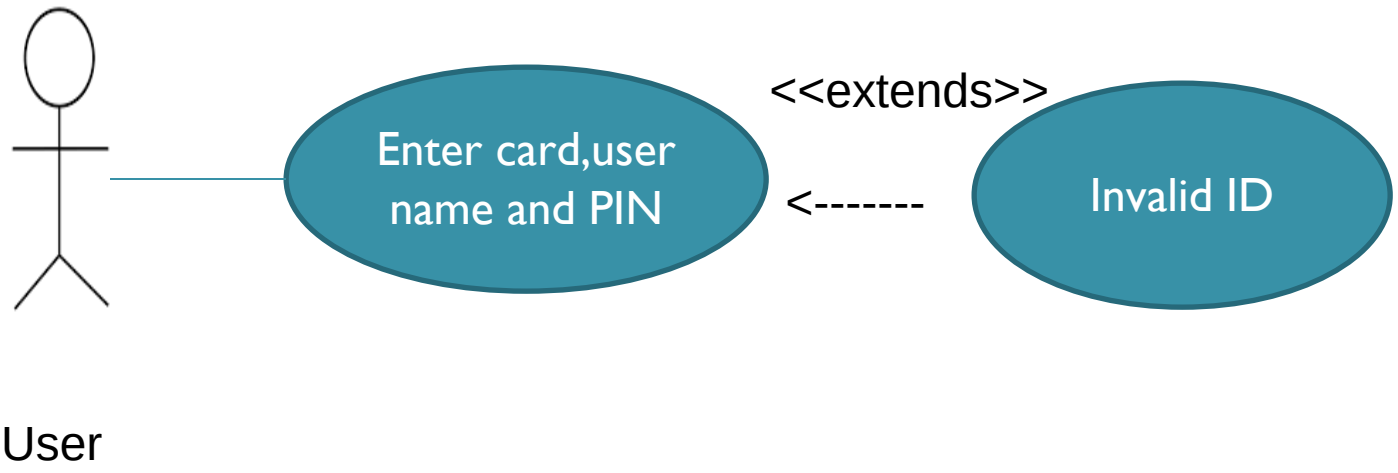



Fig.2 Use Case Diagram for Customer Authentication

# Step-1

- The user is authenticated when enters the plastic ATM card in a Bank ATM. Then enters the user name and PIN (Personal Identification Number).
- For every ATM transaction, a Customer Authentication use case is required and essential. So, it is shown as include relationship.
- Example of use case diagram for Customer Authentication is shown in Fig.No-2

# Extends Relationship



# Extends Relationship

- The extends relationship describes the situation in which one use case allows the new use case to handle a exception or variation from the basic use case.
- The arrow goes from the extended to the basic use case.

# Use Case Diagram for Bank ATM System

- Step-2

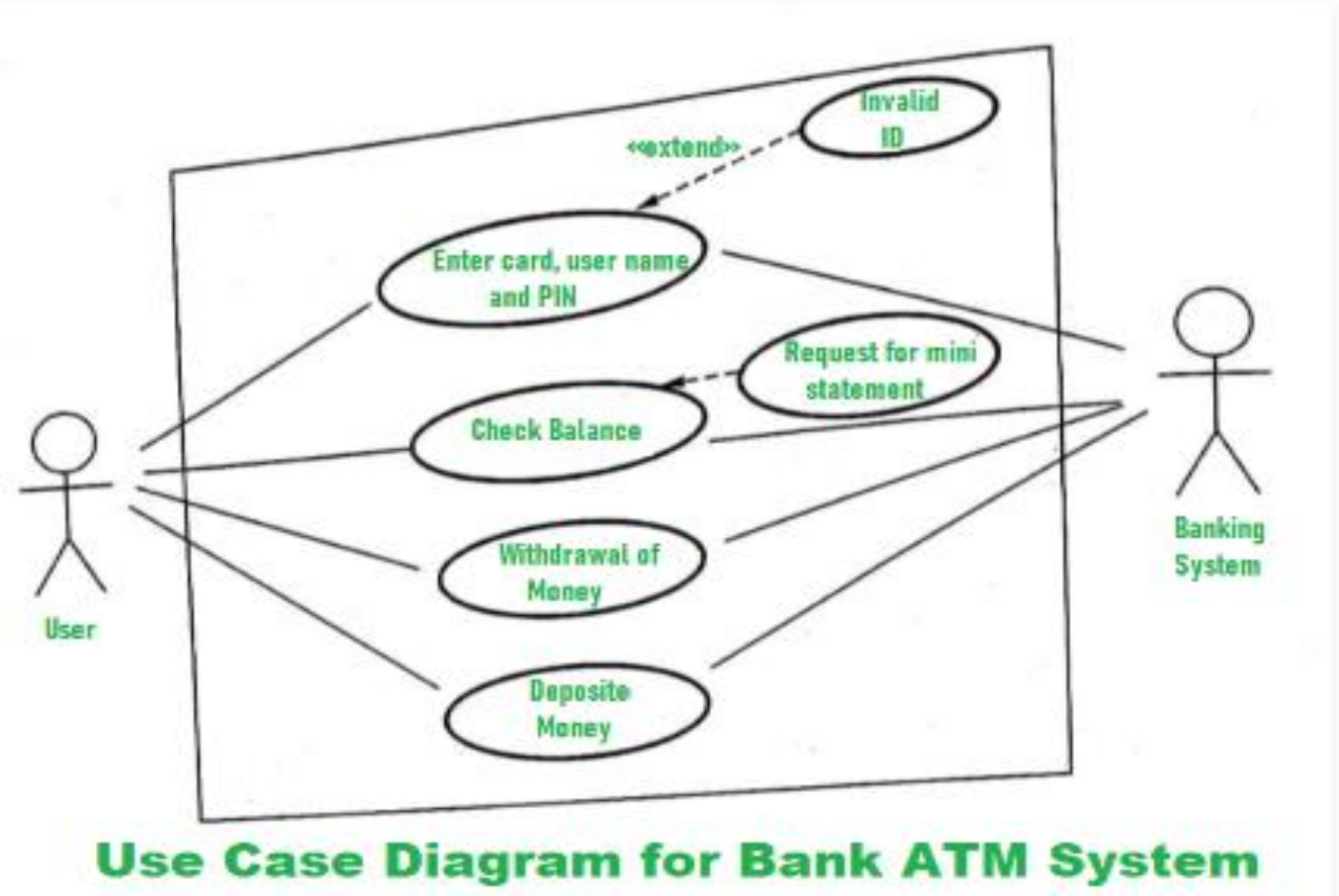



Fig.3 Use Case Diagram for Bank ATM System

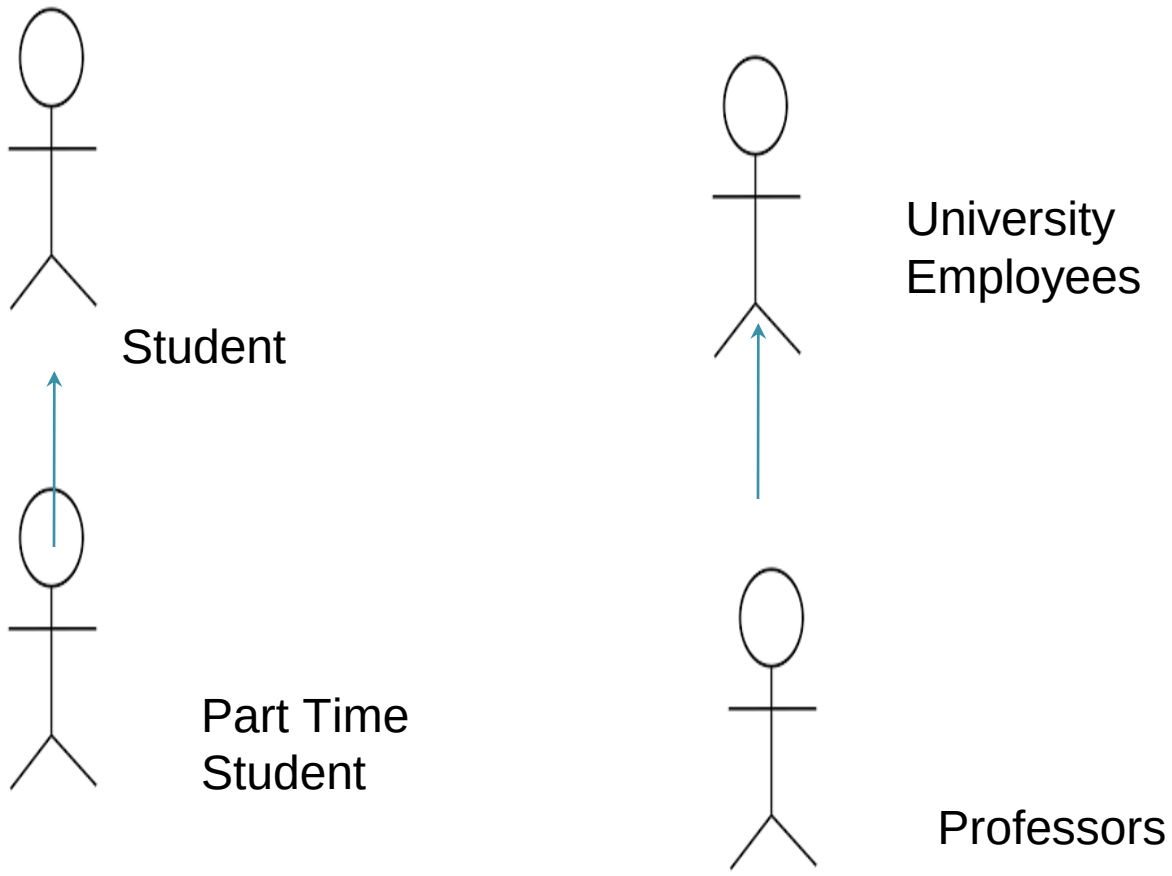
# Step-2

- User checks the bank balance as well as also demands the mini statement about the bank balance if they want.
- Then the user withdraws the money as per their need. If they want to deposit some money, they can do it. After complete action, the user closes the session. Example of the use case diagram for Bank ATM system is shown in fig. no-3:

# Generalization Relationship

- It implies that one thing is more typical than the other thing.
- The relationship may exist between 2 actors or 2 use cases.
- For example,
  - A part time student generalizes a student.
  - Some of the university employees are professors.
- The arrow points to the general thing.

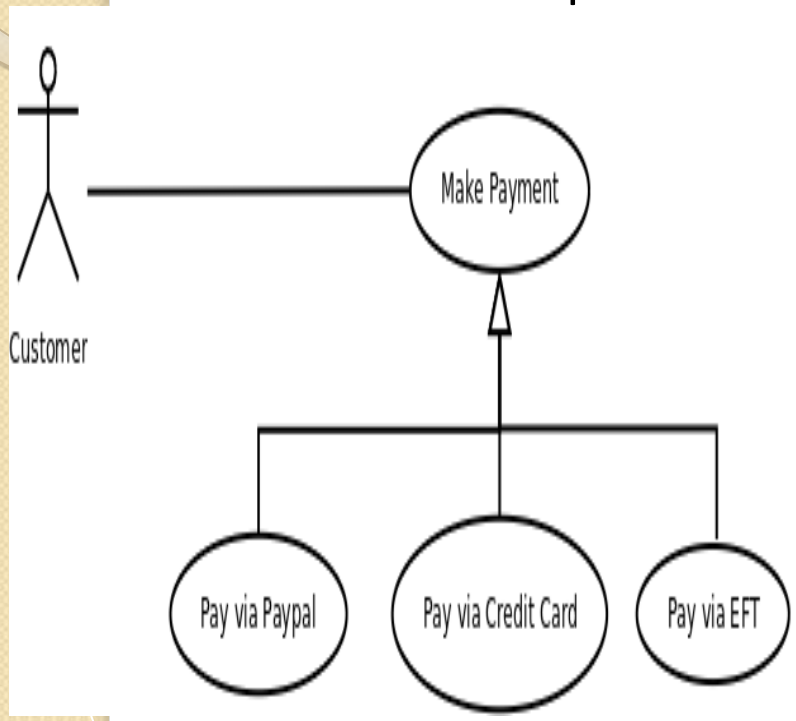
# Generalization Example

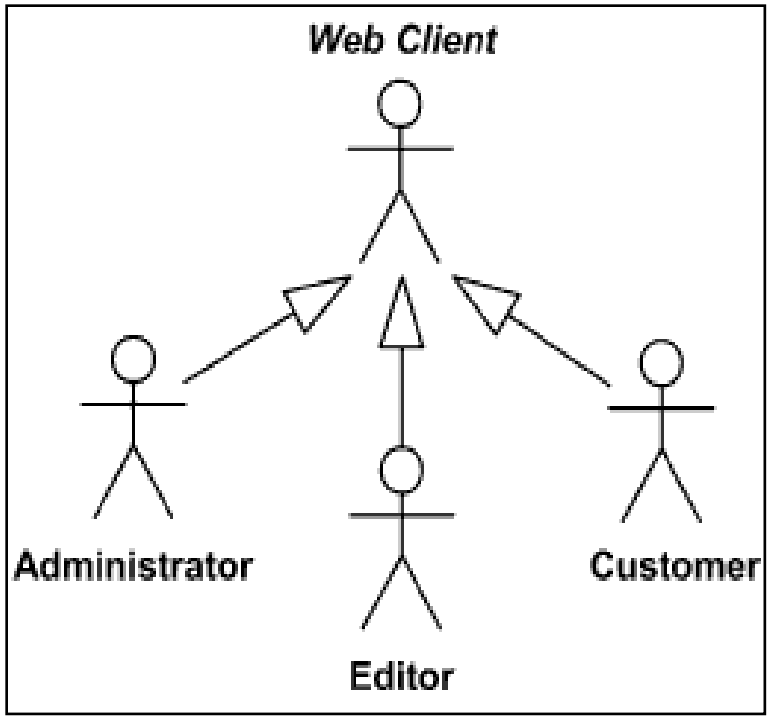


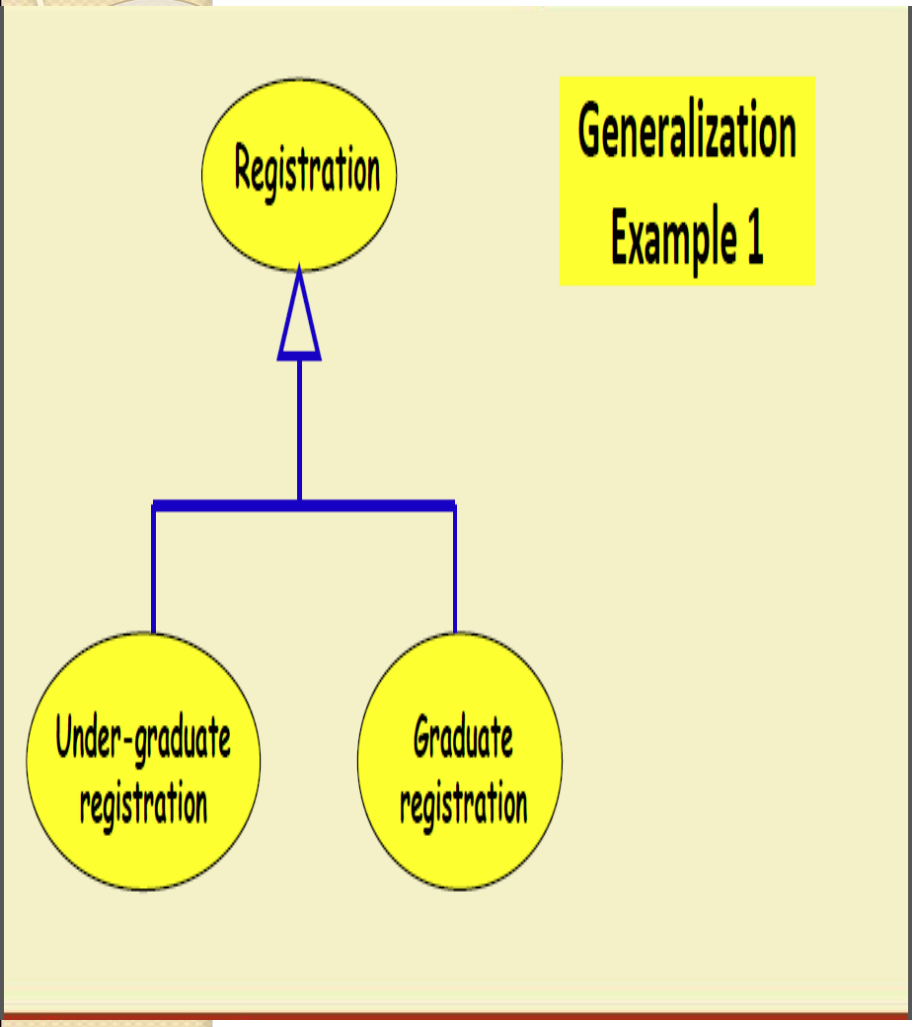
# Components of Use Case Diagram

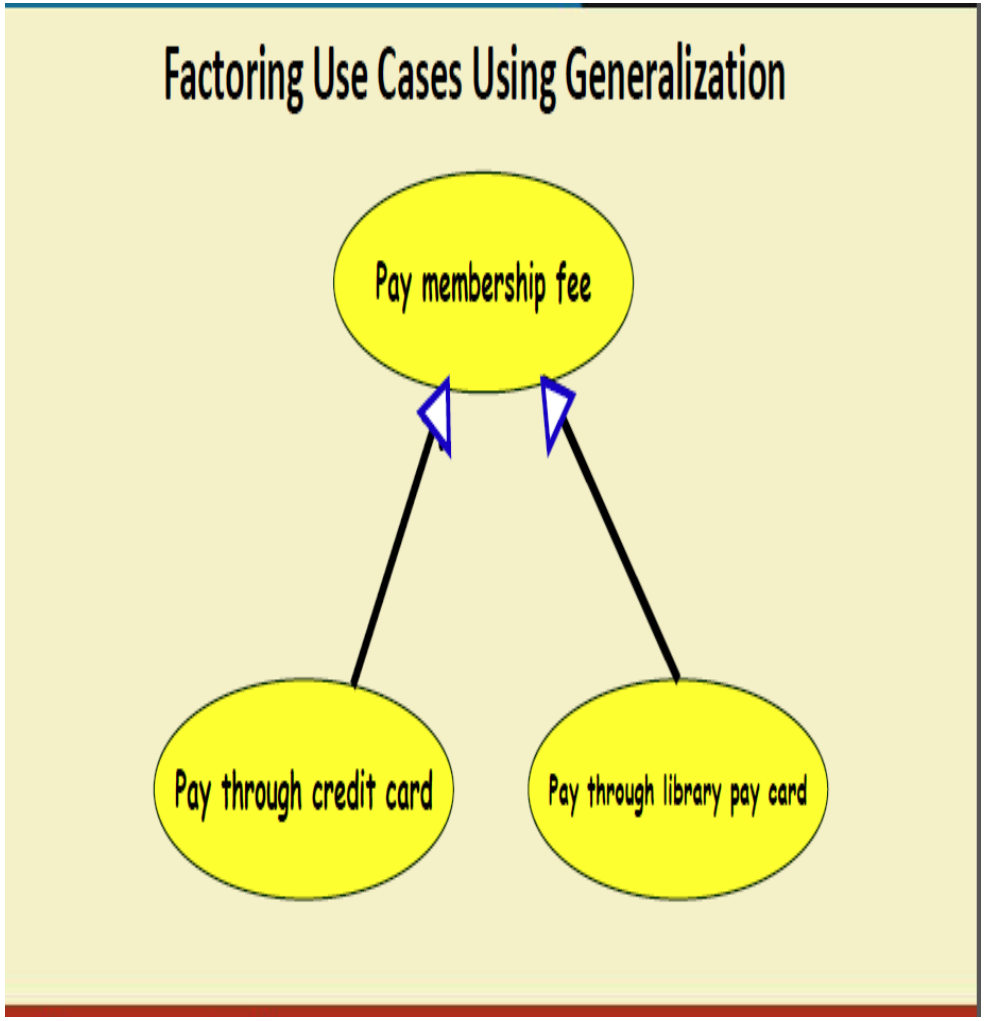
## Generalization Relationship

- Represented by a line and a hollow arrow
  - From child to parent




Registration
Generalization
Example 1
Under-graduate registration
Graduate registration


Factoring Use Cases Using Generalization
Pay membership fee
Pay through credit card
Pay through library pay card

# Use Case Diagram for Bank ATM System

- Step-3

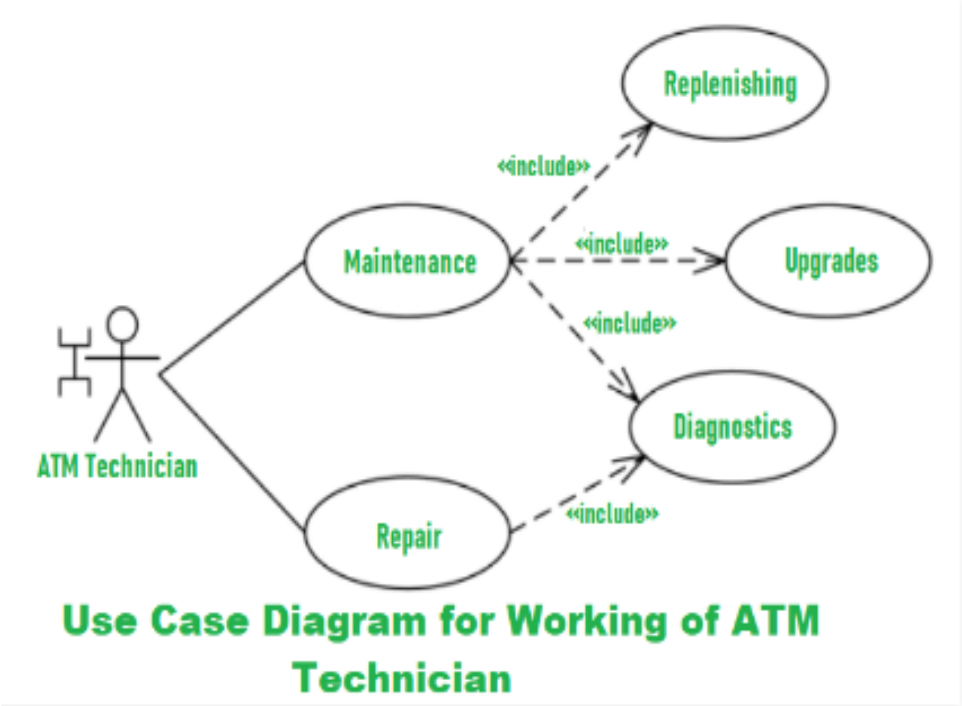
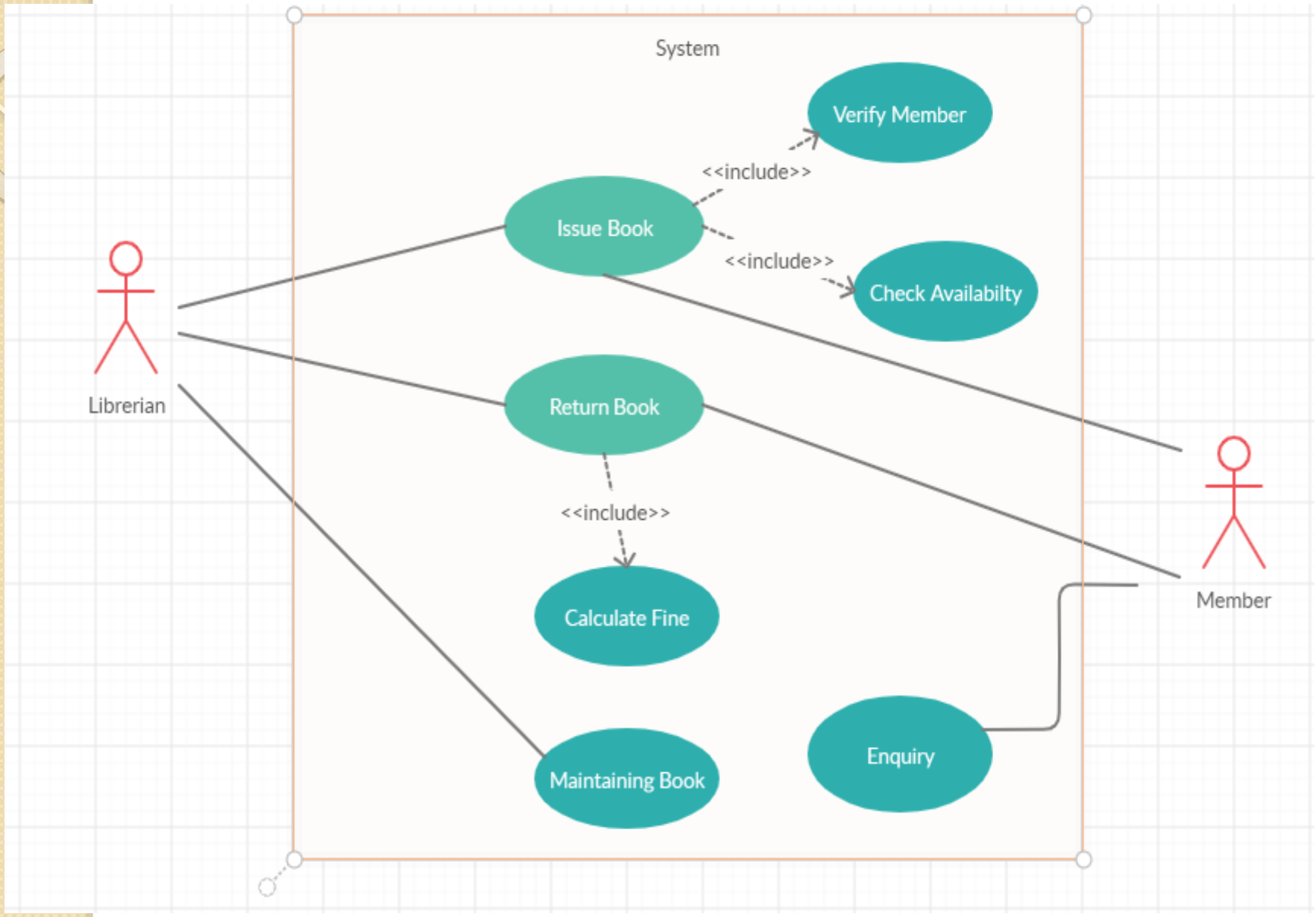



Fig.4 Use Case Diagram for Working of ATM Technician

# Step-3:

- If there is any error or repair needed in Bank ATM, it is done by an ATM technician.
- ATM technician is responsible for the maintenance of the Bank ATM, upgrades for hardware, firmware or software, and on-site diagnosis.
- Example of use case diagram for working of ATM technician is shown in fig no-4

# Library Management System



# Scenario Based Model :
# Use Cases,Activity Diagrams and Swim lane Diagrams

# Activity Diagram

- The UML activity diagram supplements the use case by providing a graphical representation of the flow of interaction within a specific scenario
- Used to document workflow of business process activities for each use case or scenario
- Similar to the flowchart, an activity diagram uses rounded rectangles to imply a specific system function,arrows to represent flow through the system,decision diamonds to depict a branching decision and solid horizontal lines to indicate parallel activities

# Activity Diagrams Symbols

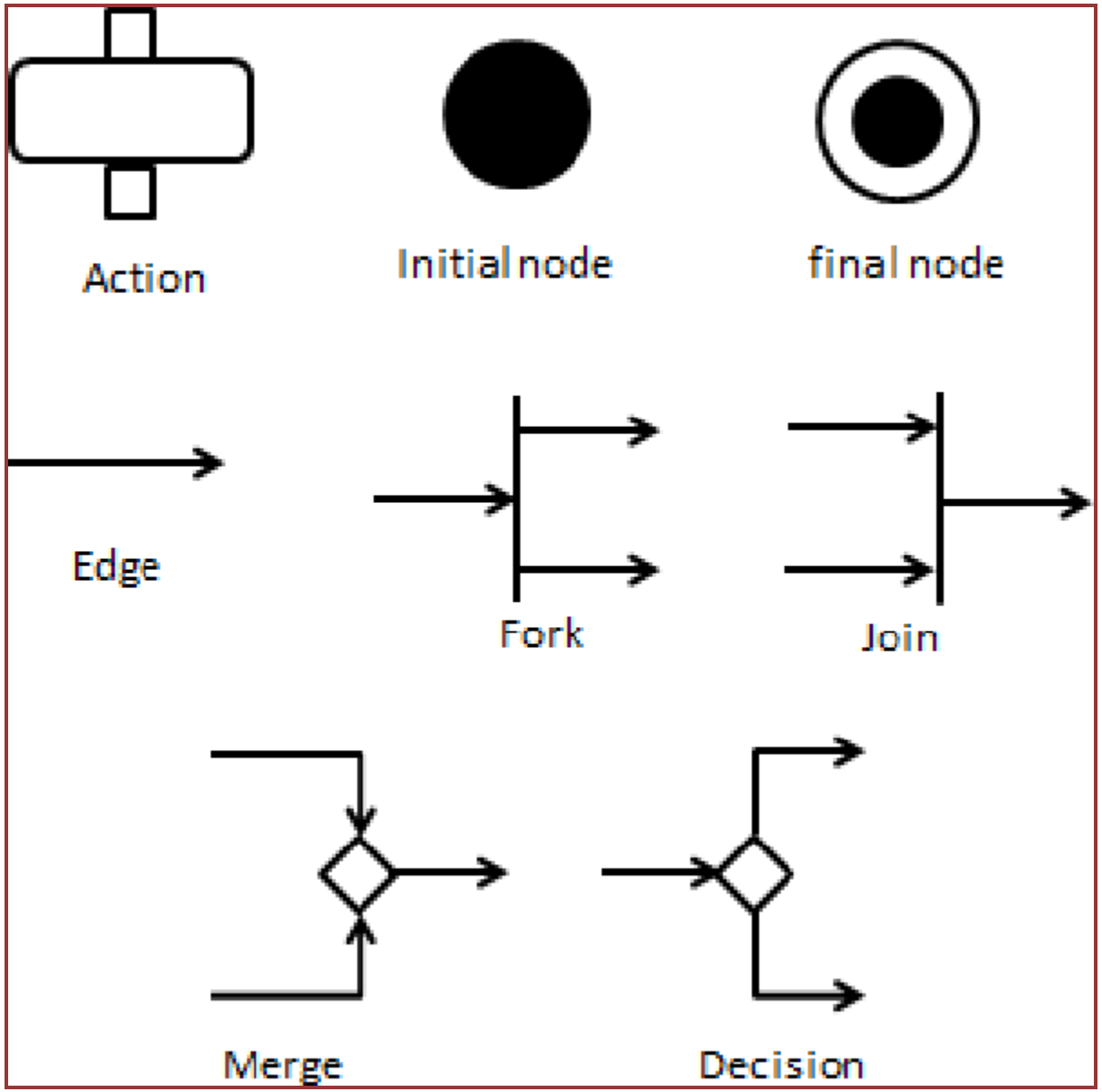
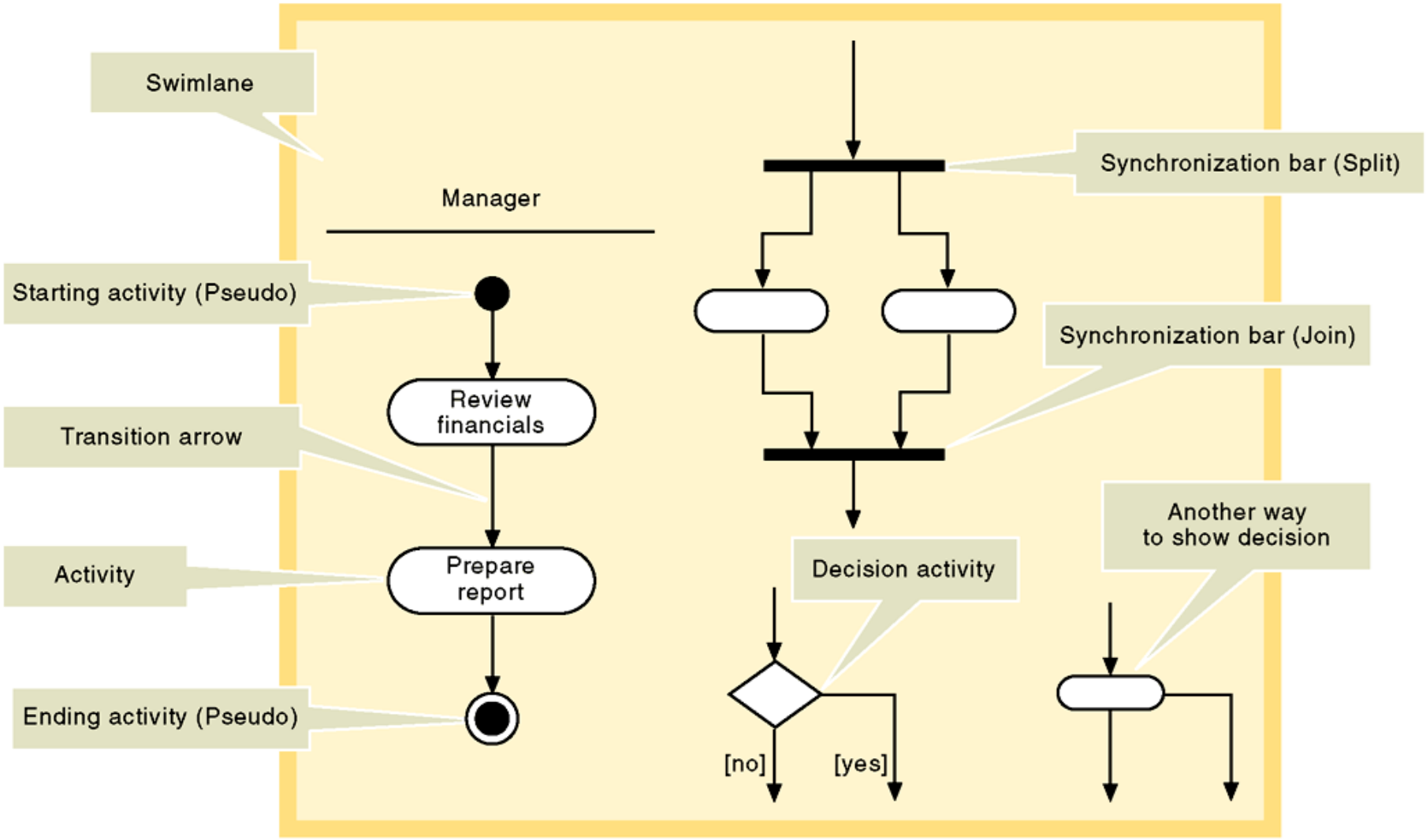


FIGURE 4-11

Activity diagram symbols.

# Activity Diagram Symbols



# Basic Components in an Activity Diagram

**Initial node**

The filled circle is the starting point of the diagram

*Activity initial node.*

**Final node**

The filled circle with a boarder is the ending point. An activity diagram can have zero or more activity final state.

*Activity final node.*

**Activity**

The rectangle with rounded ends represents activities that occur. An activity is not necessarily a program, it may be a manual thing also

**Flow/ edge**

The arrows in the diagram. No label is necessary

# Basic Components in an Activity Diagram

**Fork**

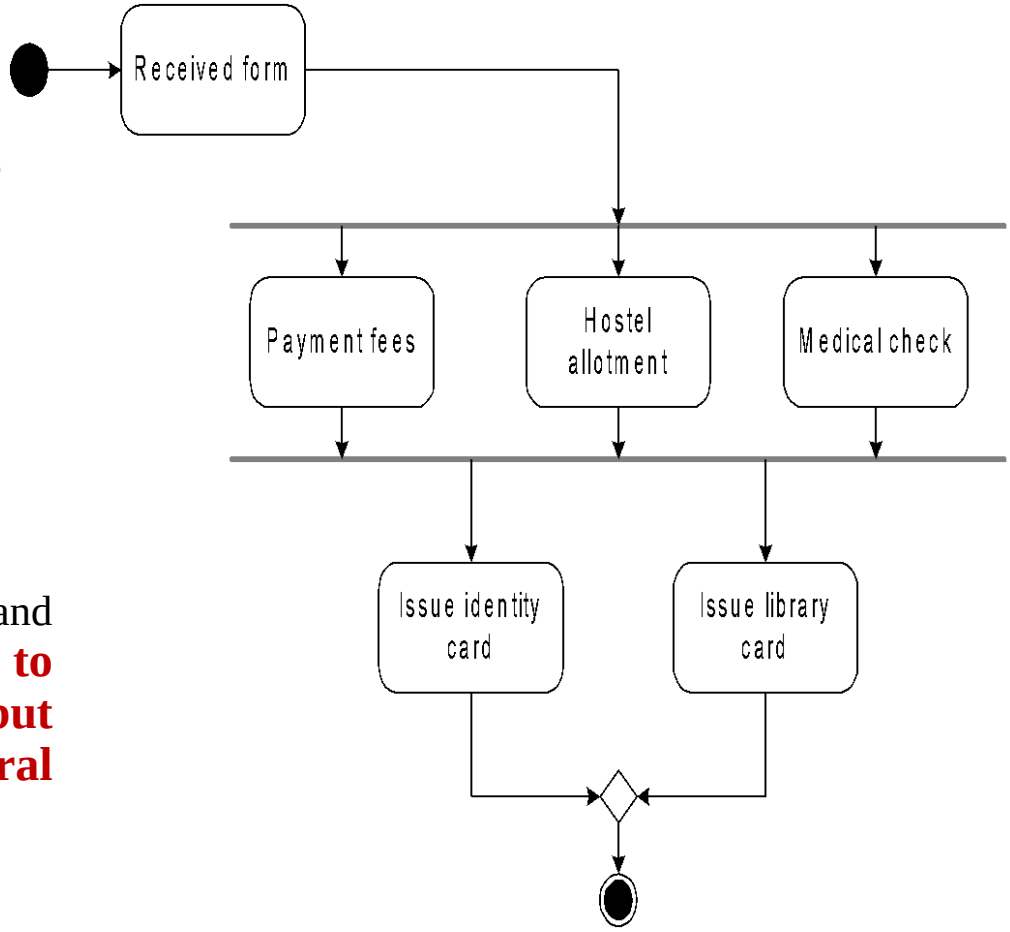
A black bar ( horizontal/vertical ) with one flow going into it and several leaving it. This denotes the **beginning of parallel activities**

**Join**

A block bar with several flows entering it and one leaving it. **this denotes the end of parallel activities**

**Merge**

A diamond with several flows entering and one leaving. **It is not used to synchronize concurrent flows but to accept one among several alternate flows.**



# How to Draw an Activity Diagram

- Diagrams are read from top to bottom and have branches and forks to describe conditions and parallel activities.
  - A fork is used when multiple activities are occurring at the same time.
  - A branch describes what activities will take place based on a set of conditions.
  - All branches at some point are followed by a merge to indicate the end of the conditional behavior started by that branch.
  - After the merge all of the parallel activities must be combined by a join before transitioning into the final activity state.

# Basic Components in an Activity Diagram

Difference between Join and Merge

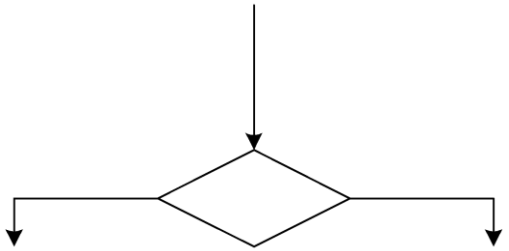
- **A join is different from a merge in that the join synchronizes two inflows and produces a single outflow. The outflow from a join cannot execute until all inflows have been received**

- **A merge passes any control flows straight through it. If two or more inflows are received by a merge symbol, the action pointed to by its outflow is executed two or more times**

# Basic Components in an Activity Diagram

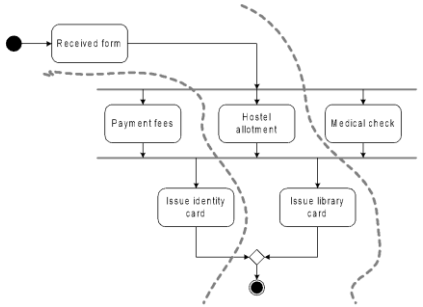
**Decision**

A diamond with one flow entering and several leaving. The flow leaving includes conditions as yes/ no state

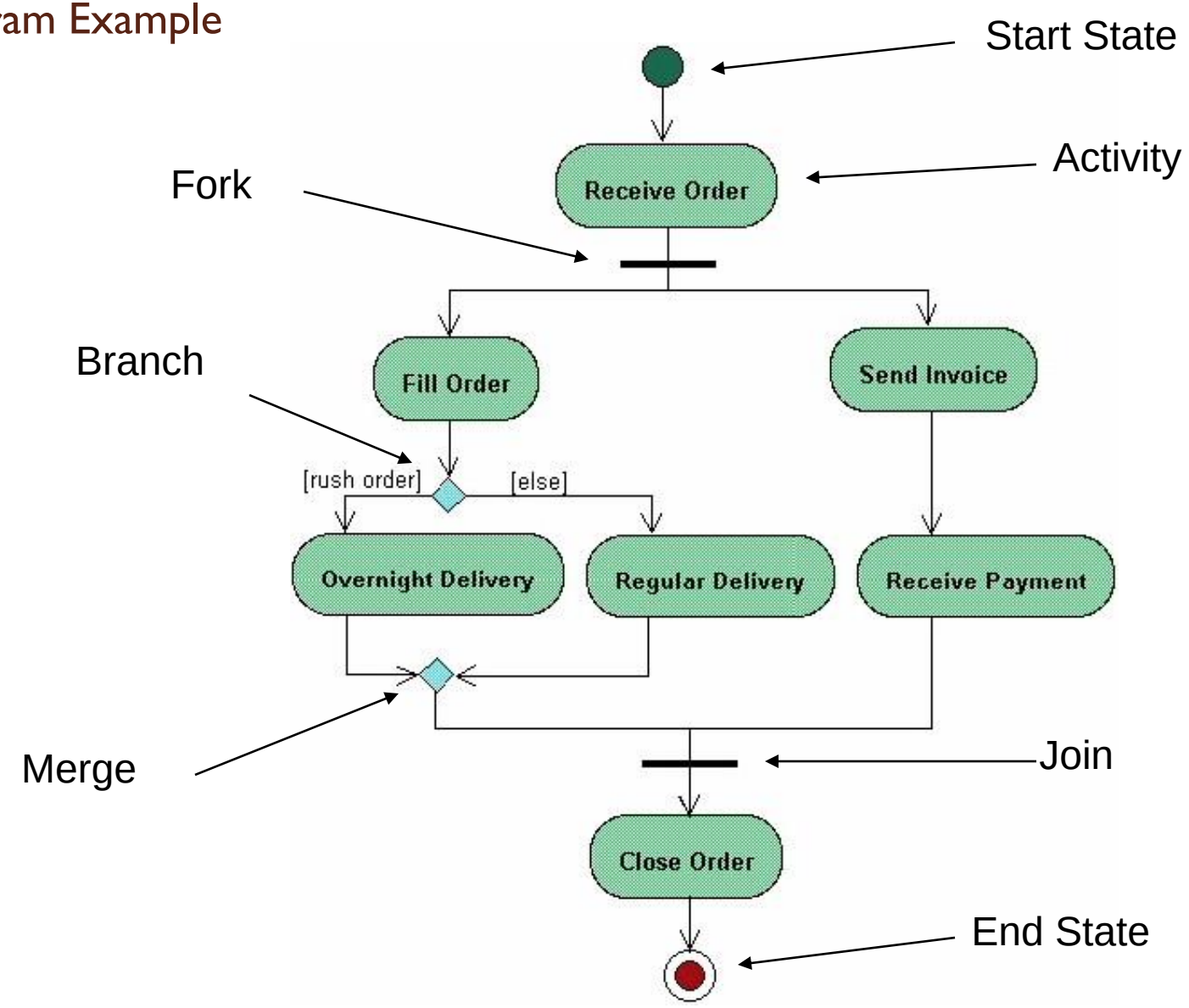


**Swim lane**

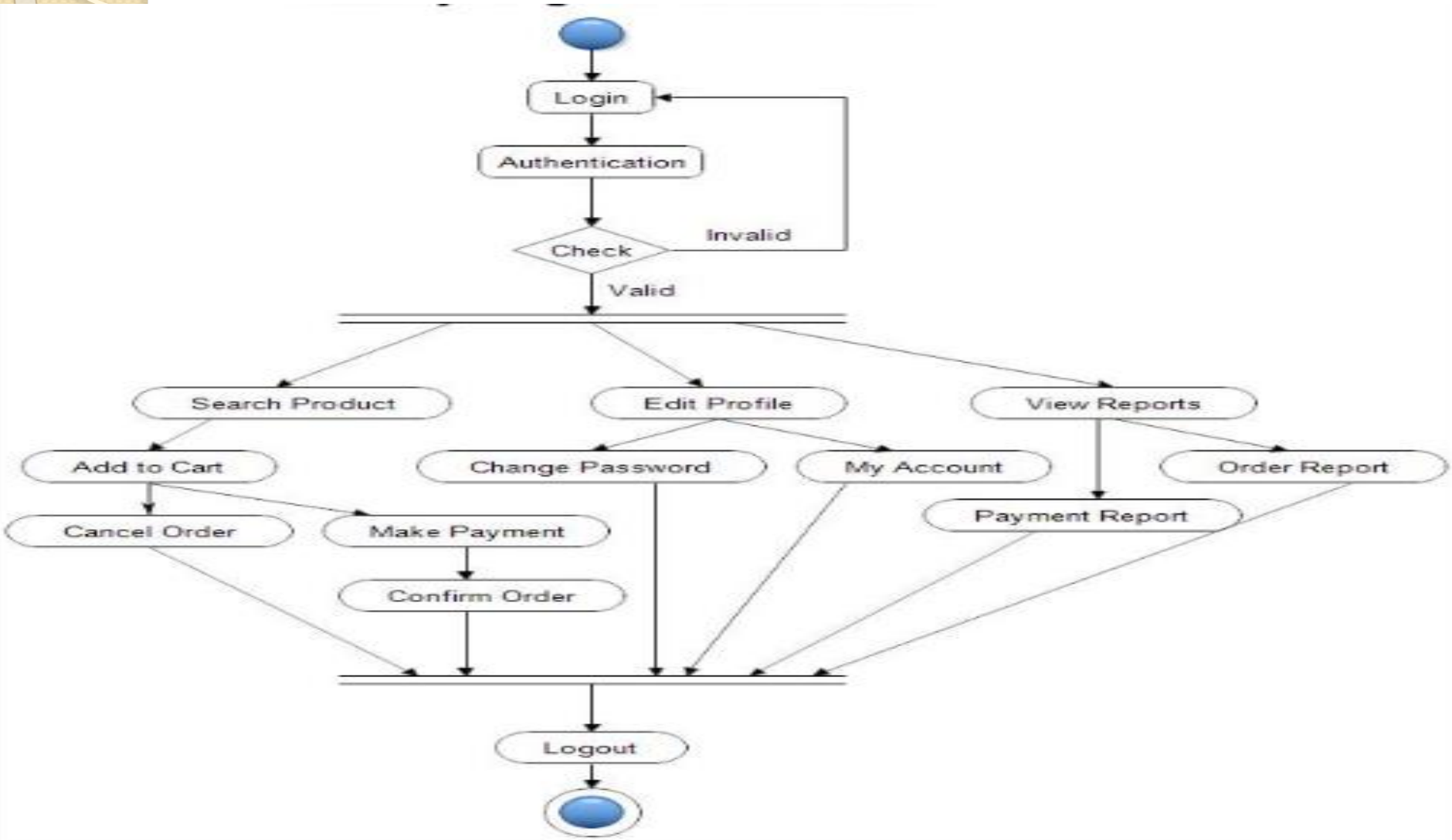
A partition in activity diagram by means of line, called swim lane. This swim lane may be horizontal or vertical



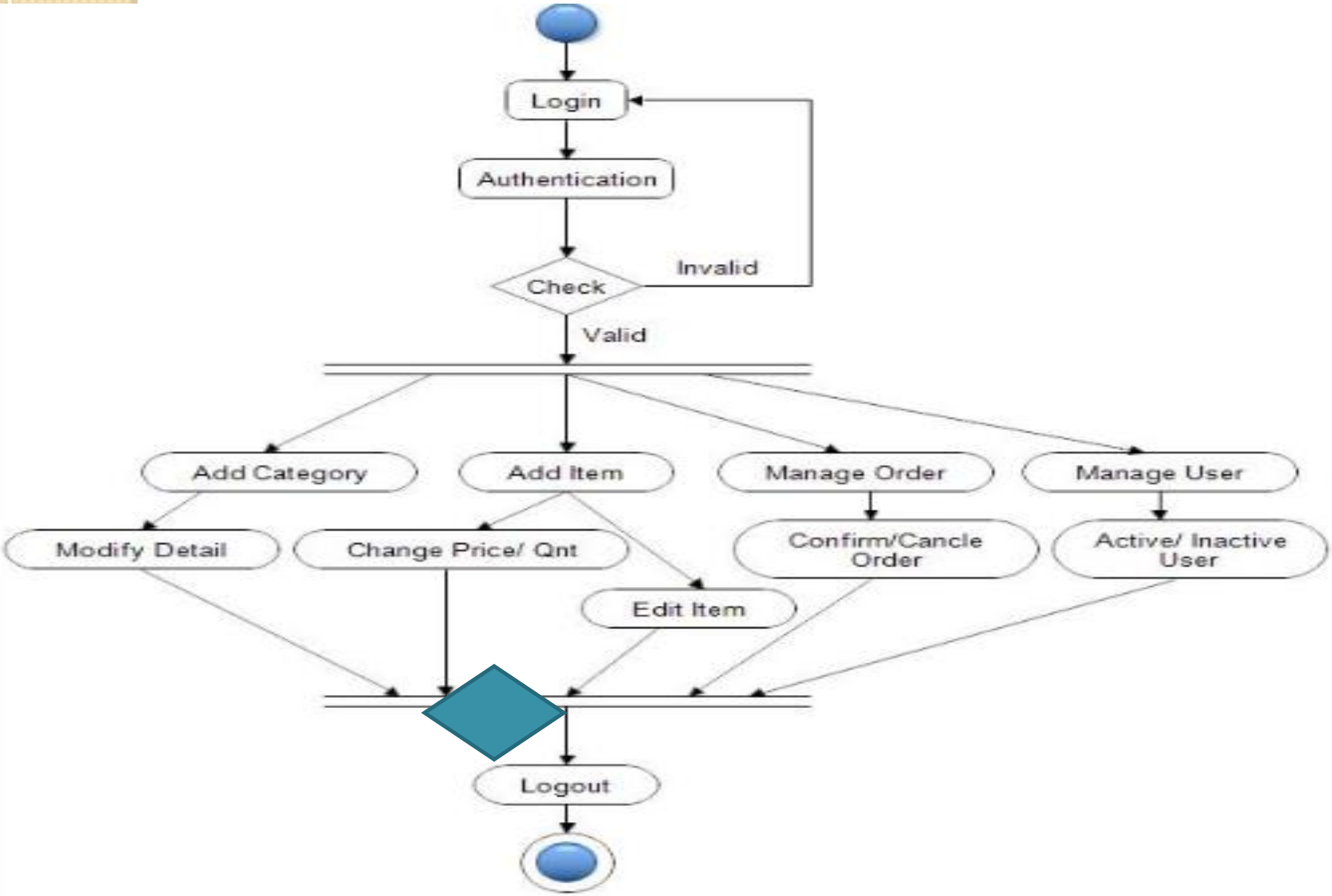
# Activity Diagram Example



# Activity Diagram for Online Shopping Website-Activity Diagram for User Side



# Activity Diagram for Admin Side

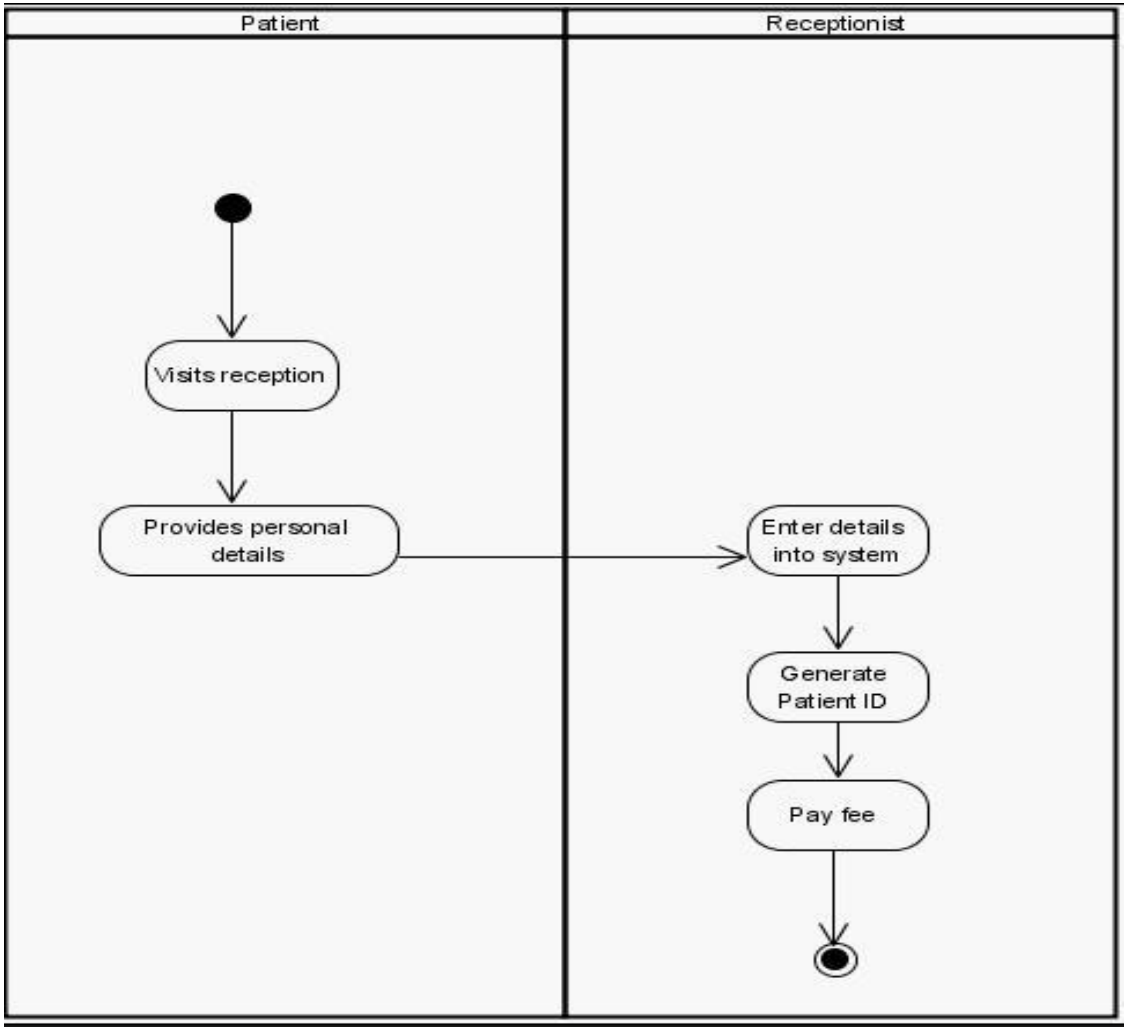


# Swim lane Diagram

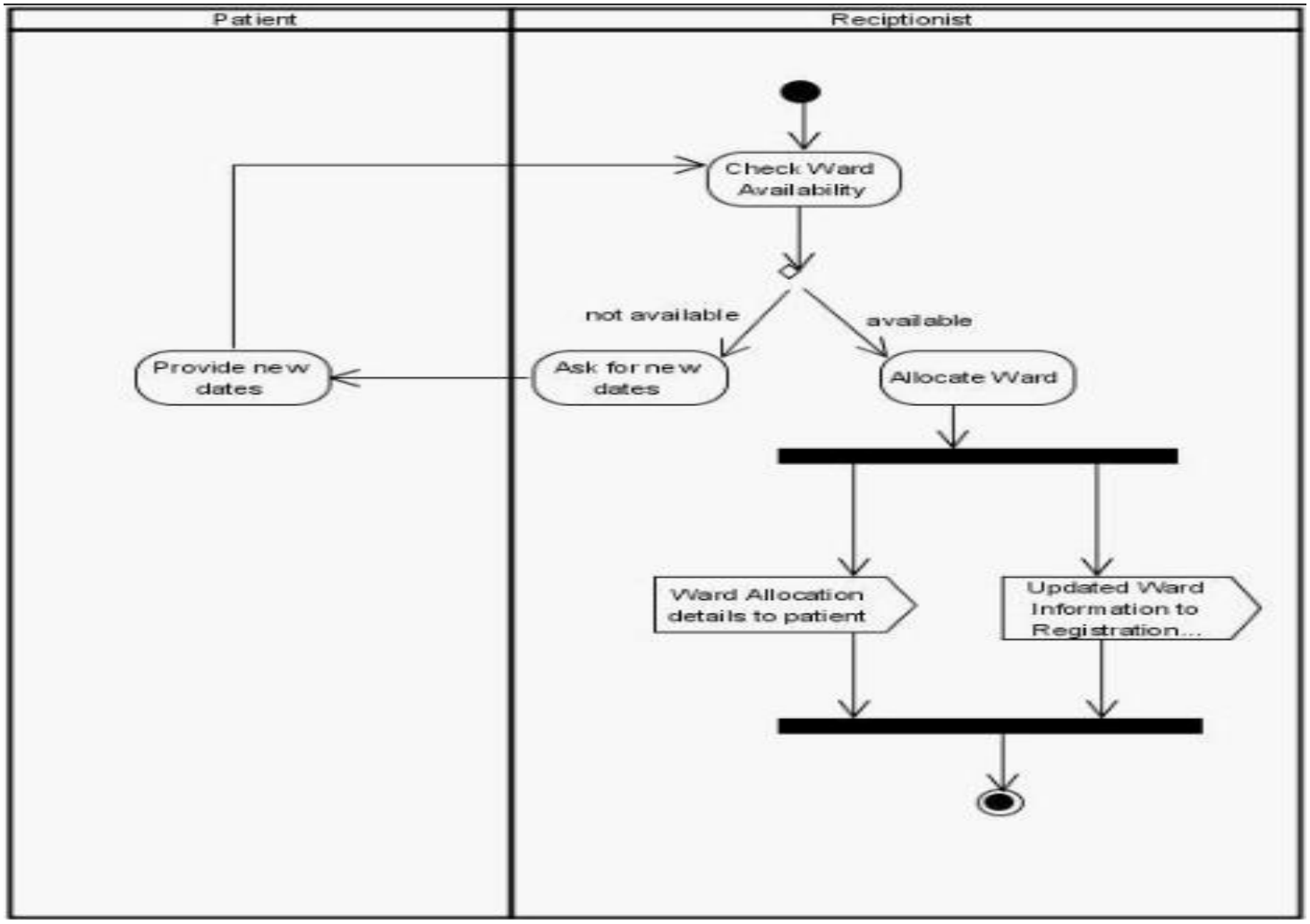
- Swim lane diagram is a useful variation of the activity diagram
- It allows the modeler to represent the flow of activities described by the use case
- It also indicate which actor(if there are multiple actors involved in a specific function) or analysis class has responsibility for the action described by an activity rectangle
- Here,the interaction between different activities shown in different lanes
- To show parallel activities,diagram is divided vertically into lanes,similar to lanes of swimming pool.

# Swim lane Diagram: Hospital Management System

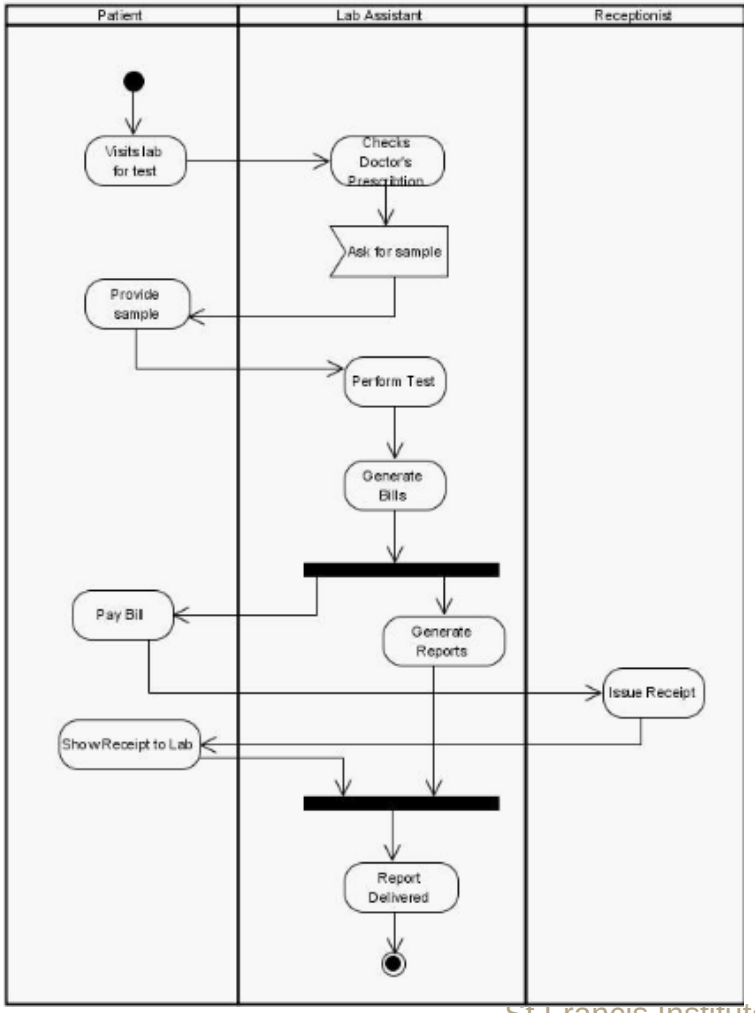
# Swim lane Diagram:Registration



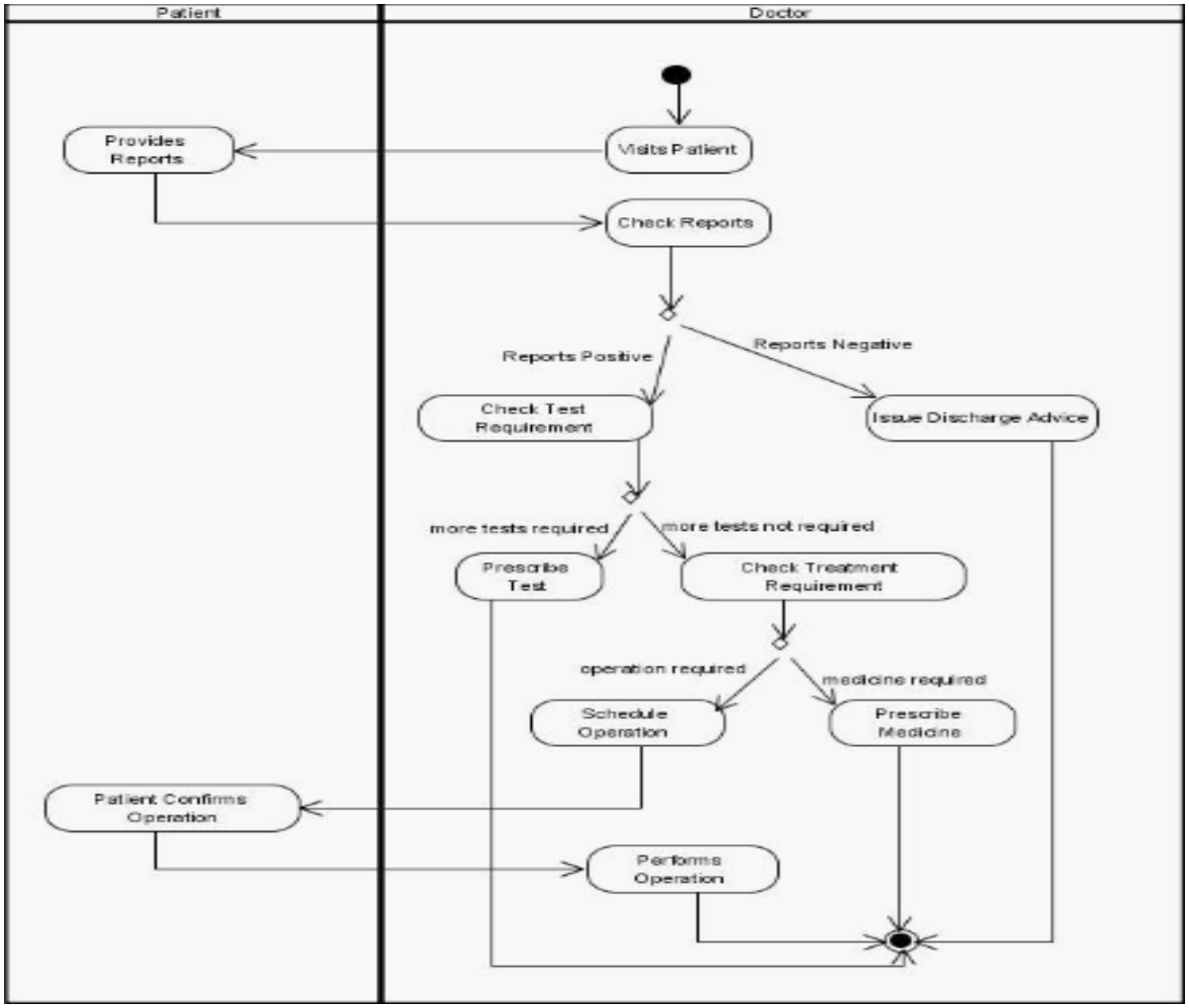
# Swim lane Diagram for Ward Allocation:-



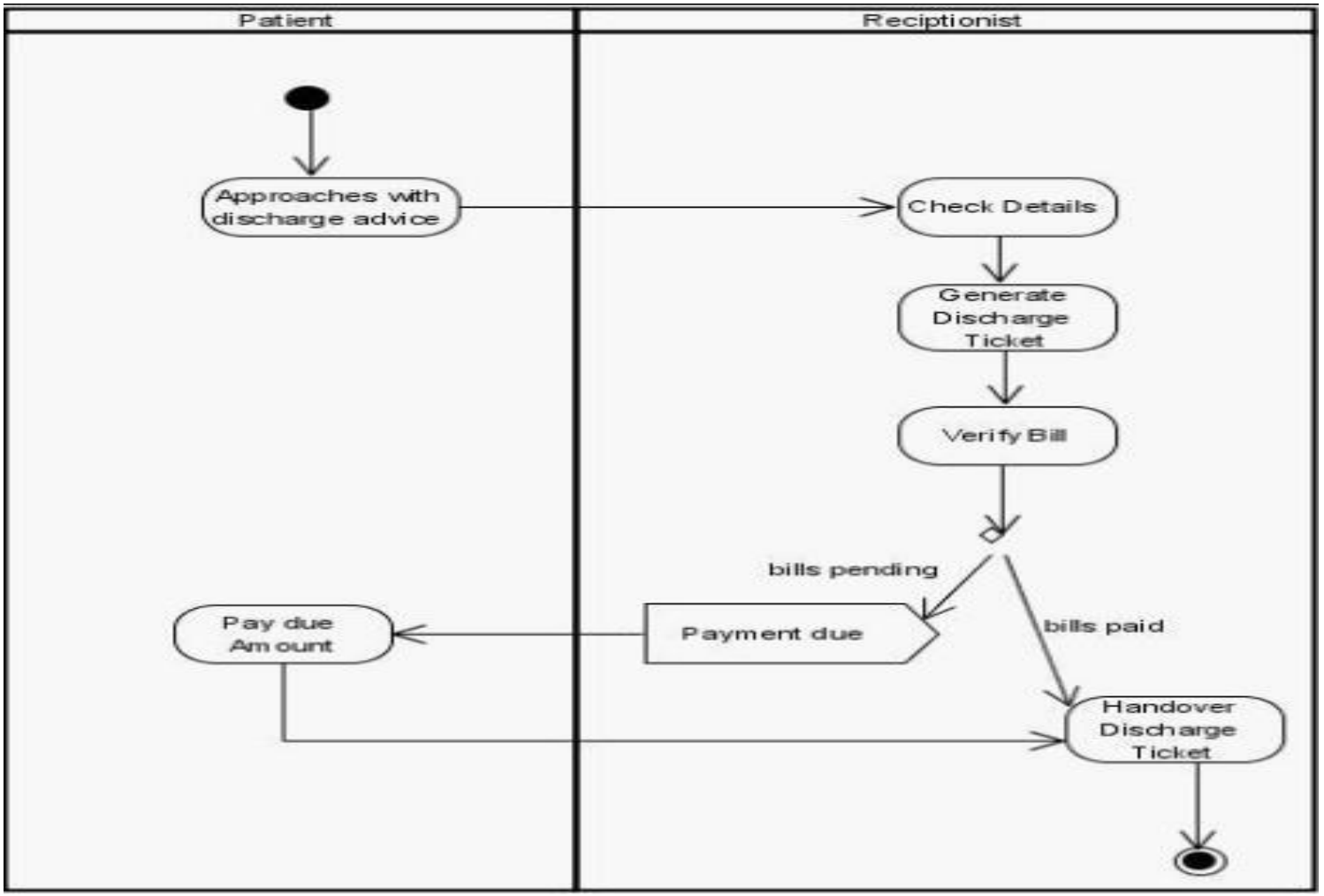
# Swim lane Diagram for Tests to Perform:-



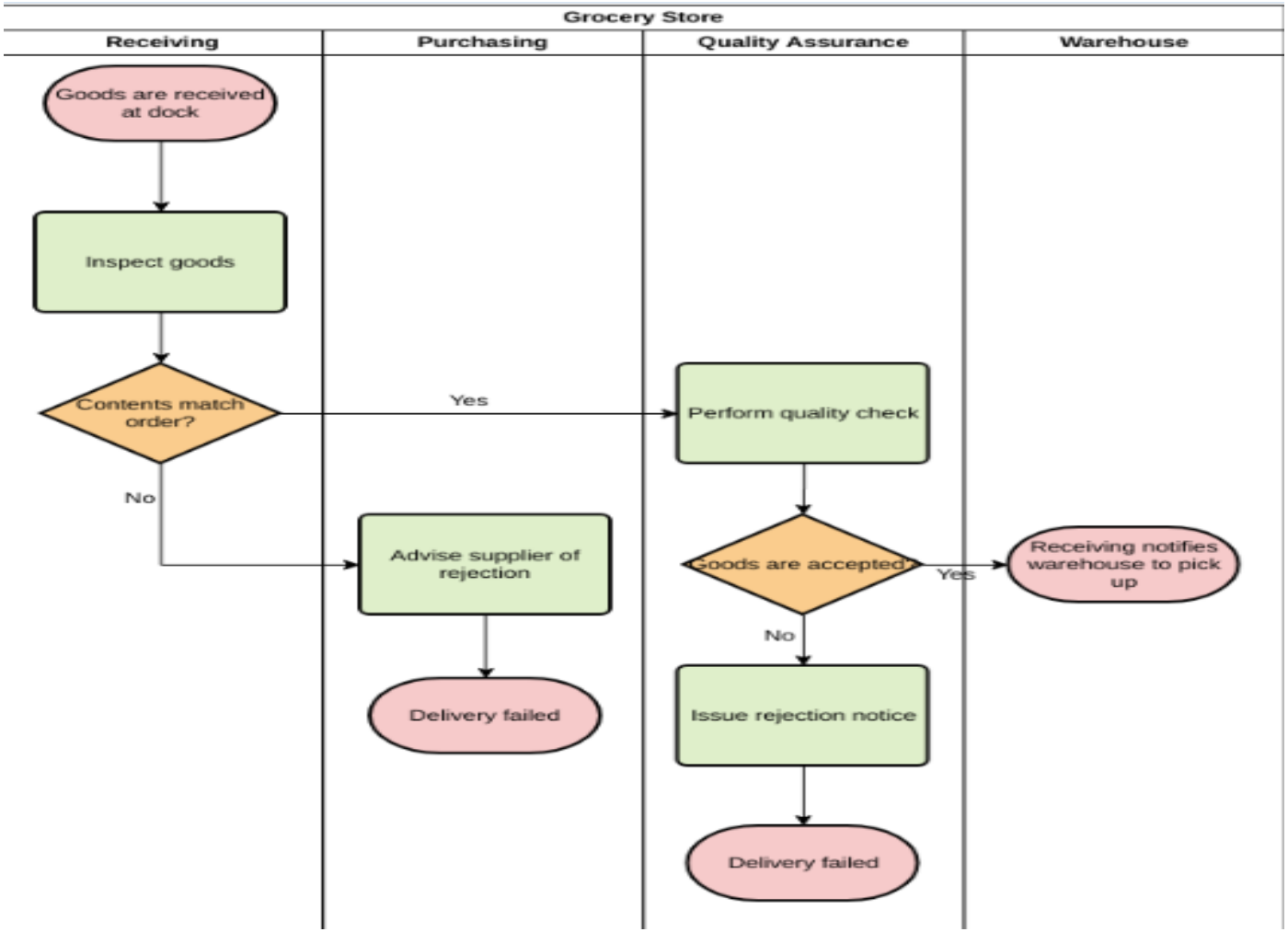
# Swim lane Diagram for Treatment and Operations:-



# Swim lane Diagram Discharge:-



# Swim Lane Diagram for Receiving Goods



# Swim Lane Diagram for Login System

**Swimlane Diagram:**

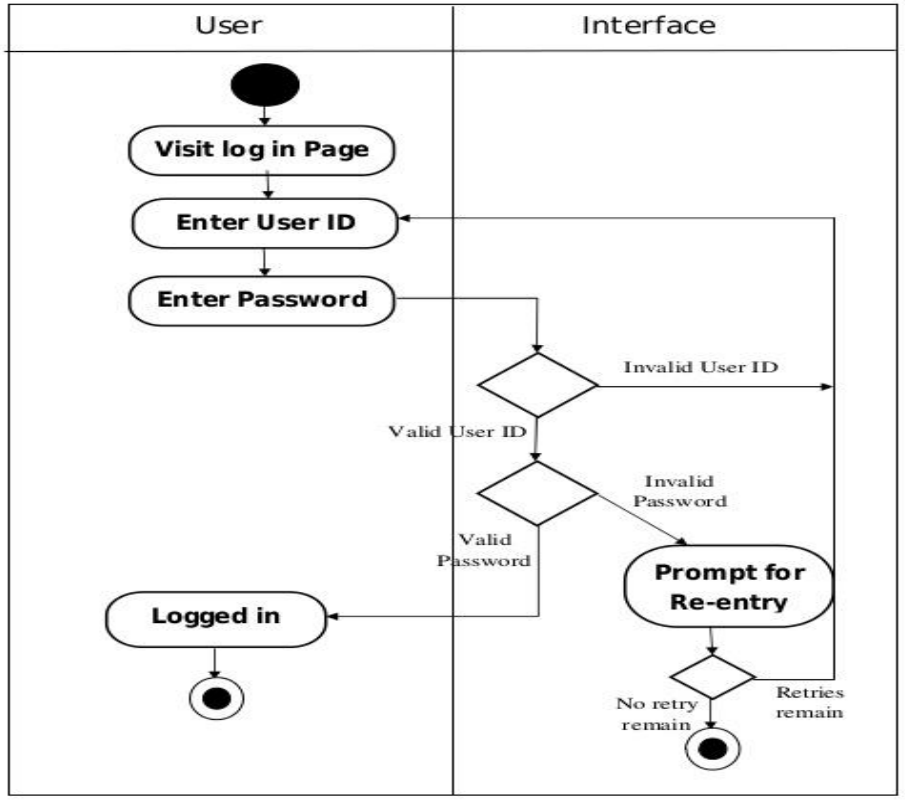
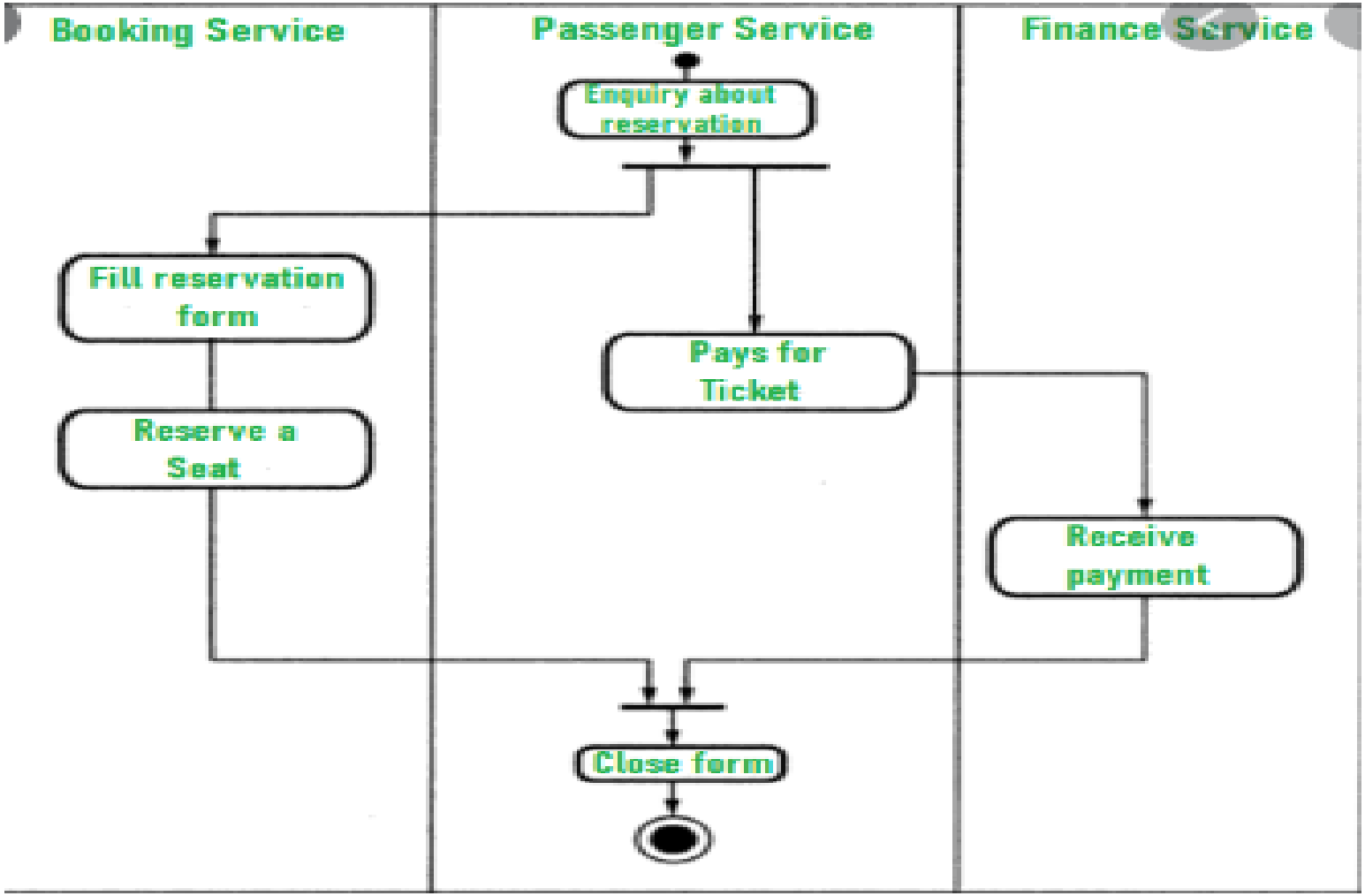



Fig 11: Swimlane Diagram of Sign In

# Swimlane Diagram for Reserving a Ticket




*

# Behavioural Model

- Sequence Diagram
- State Diagram

# Introduction to Sequence diagram

- Sequence diagram is a type of behavioral representations in UML
- It indicates how events cause transitions from object to object
- Once events are identified by examining a use-case,the modeler creates a sequence diagram-a representation of how events cause flow from one object to another as a function of time
- Sequence diagram is an interaction diagram that shows the **set of objects and messages sent and received by those object.**
- It mainly emphases on **time ordering and messages**.
- Sequence diagram is a shorthand version of the use case
- It represents key classes and the events that cause behavior to flow from class to class

# Activity Diagram And Sequence Diagram

The **main difference** between activity diagram and sequence diagram is that the **activity diagram represents the flow of activities one after the other in a system** **while** **the sequence diagram represents the sequence of messages flowing from one object to another.**

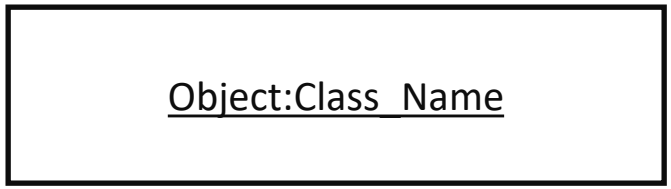
**Activity diagram is focused on Actions within the behavior**.
**Sequence diagram is focused on Interactions** (communication between objects) within the behavior.

The main focus in an **activity diagram is the flow of activities** whereas the main focus in a **sequence diagram is the interaction between objects over a specific period of time.**

# Terms and Concepts

**Objects or Participants** :-

- The sequence diagram is made up of collection of **participants or objects**.
- Participants are system parts that interacts with each other during sequence diagram.
- The participants interact with each other by sending and receiving message
- The object is represented as shown below

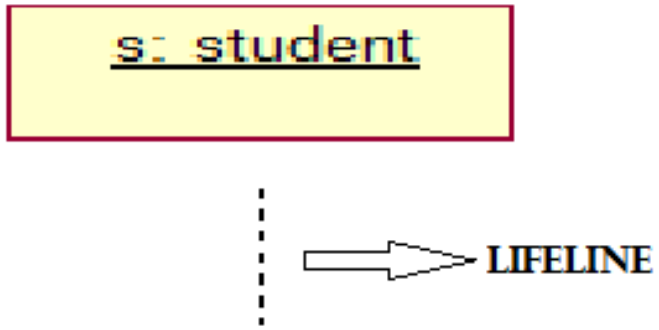


# Terms and Concepts

**Lifeline:-**

Lifeline represents the existence of  an object over a period of time.

It is represented by vertical dashed line.

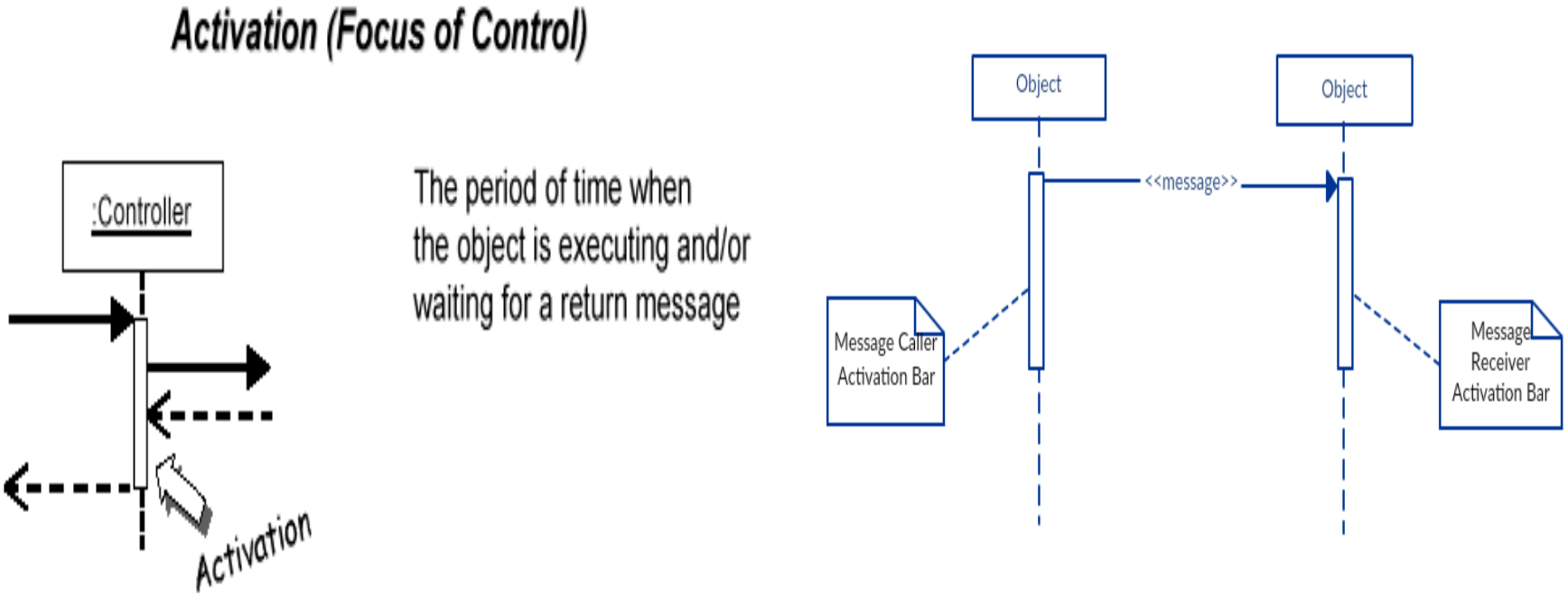


# Terms and Concepts

## Activation bar:-

It is also called as focus of control. It shows the period of time during which an object is performing an action.

It is represented by tall thin rectangle:



# Types of Messages

- Synchronous message
- Asynchronous message
- A return message
- *Participant  creation message*
- *Participant destruction message*
- *Reflexive message*

# *Synchronous message*

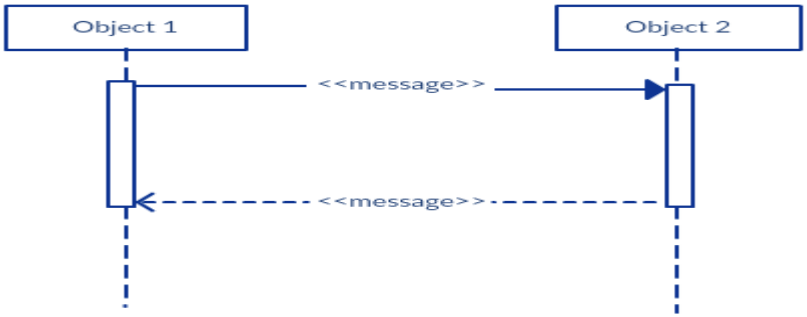
- A synchronous message waits for a reply before the interaction can move forward.
- The sender waits until the receiver has completed the processing of the message.
- The caller continues only if it receives a reply message(when it knows that the receiver has processed the previous message)

# *Asynchronous message*

- An asynchronous message does not wait for a reply from the receiver.
- The interaction moves forward irrespective of the receiver processing the previous message or not.

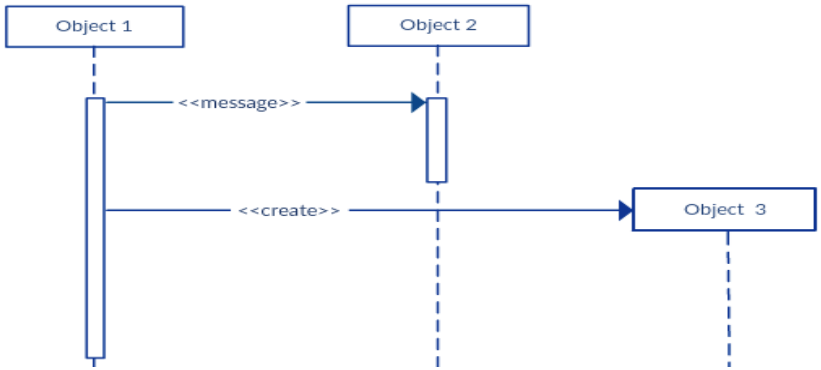
# *Types of Messages*

**A return message** is used to indicate that the message receiver is done processing the message and is returning control over to the message caller.




***Participant creation message***

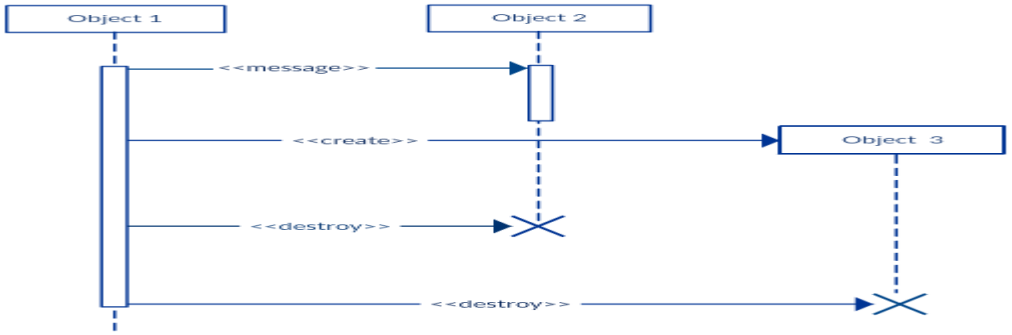
Objects do not necessarily live for the entire duration of the sequence of events. Objects or participants can be created according to the message that is being sent.



# Types Of Messages

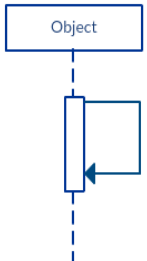
***Participant destruction message***

participants when no longer needed can also be deleted from a sequence diagram. This is done by adding an 'X' at the end of the lifeline of the said participant.




***Reflexive message***

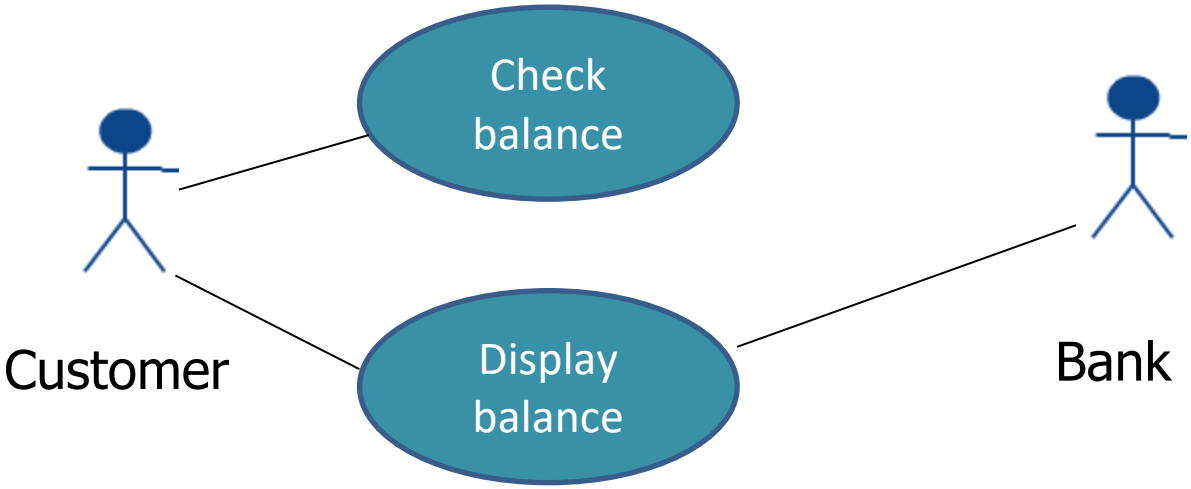
When an object sends a message to itself, it is called a reflexive message. It is indicated with a message arrow that starts and ends at the same lifeline as shown in the example below.



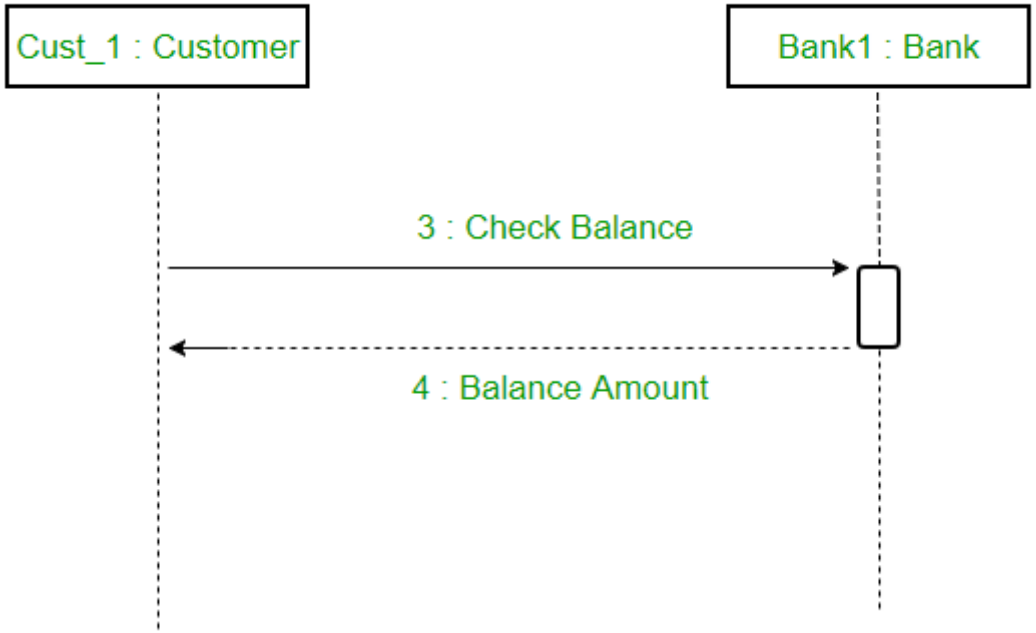
# Exercise:
# To generate sequence diagram using use case

## Steps:-

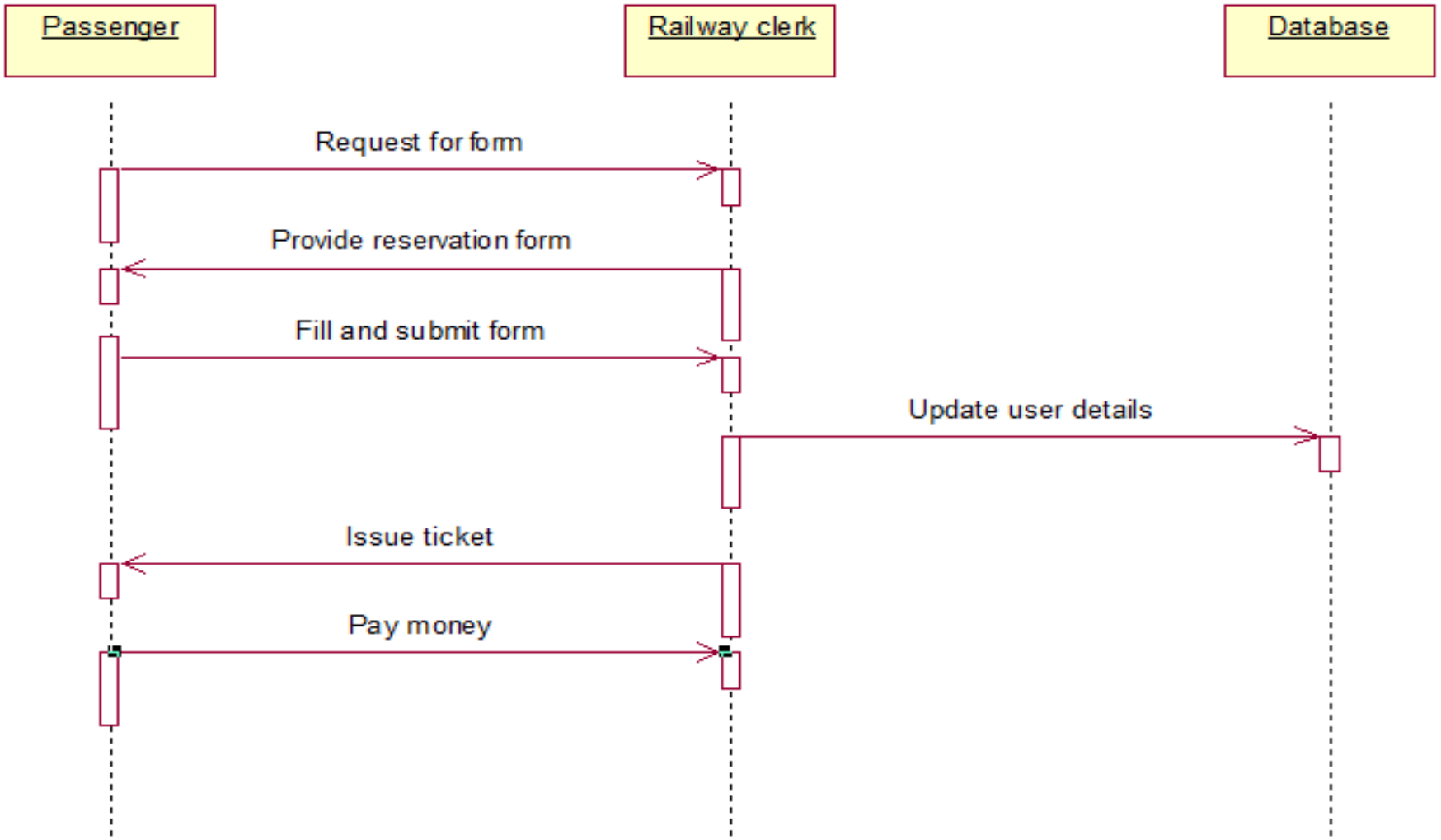
1. Designate actors and business system—Who is taking part?
2. Designate initiators—Who starts interactions?
3. Describe the message exchange between actors and business system—Which messages are being exchanged?
4. Identify the course of interactions—What is the order?
5. Insert additional information—What else is important?
6. Verify the view—Is everything correct?


Check balance
Customer
Display balance
Bank

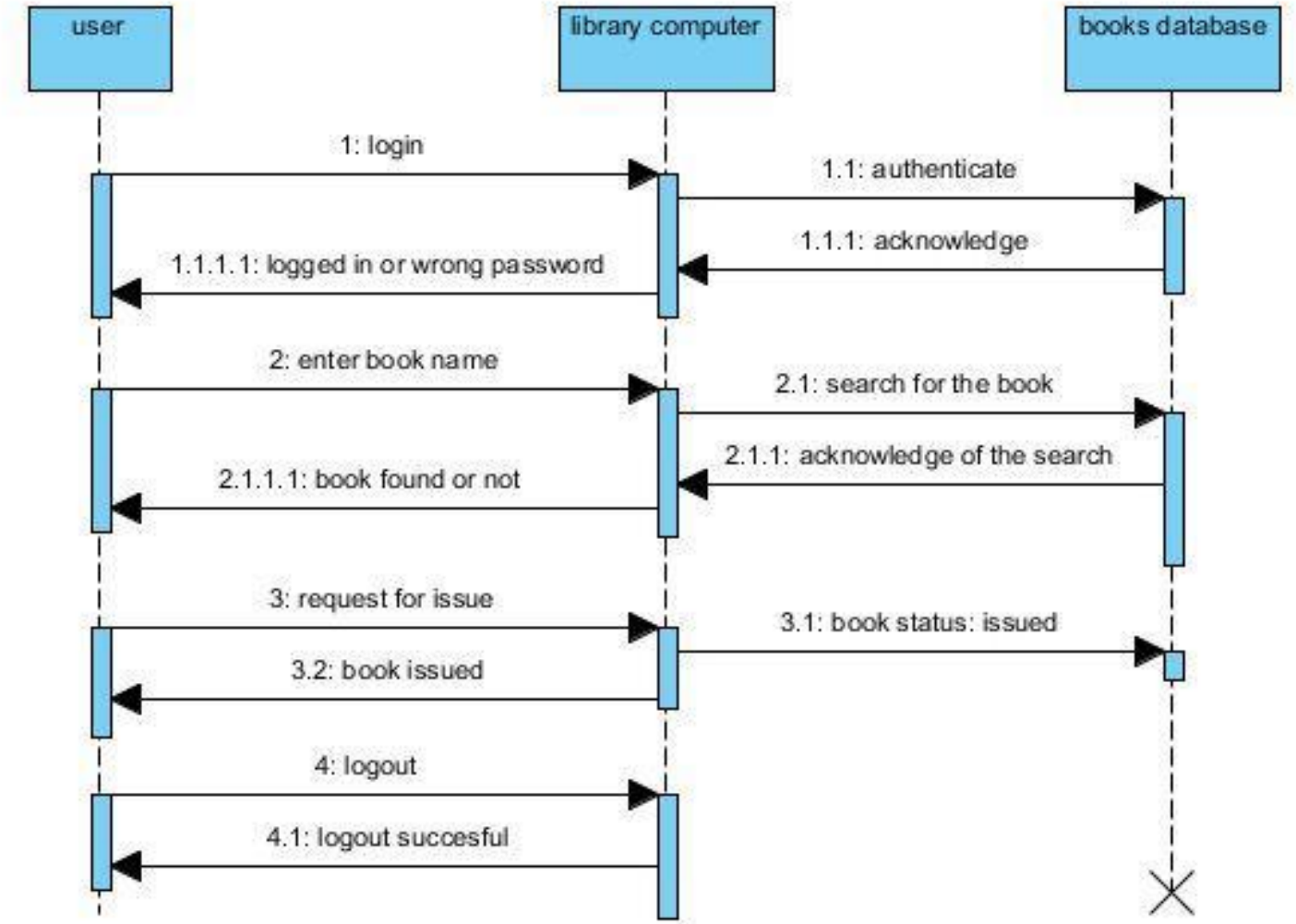
# example



# Sequence diagram of Railway reservation system

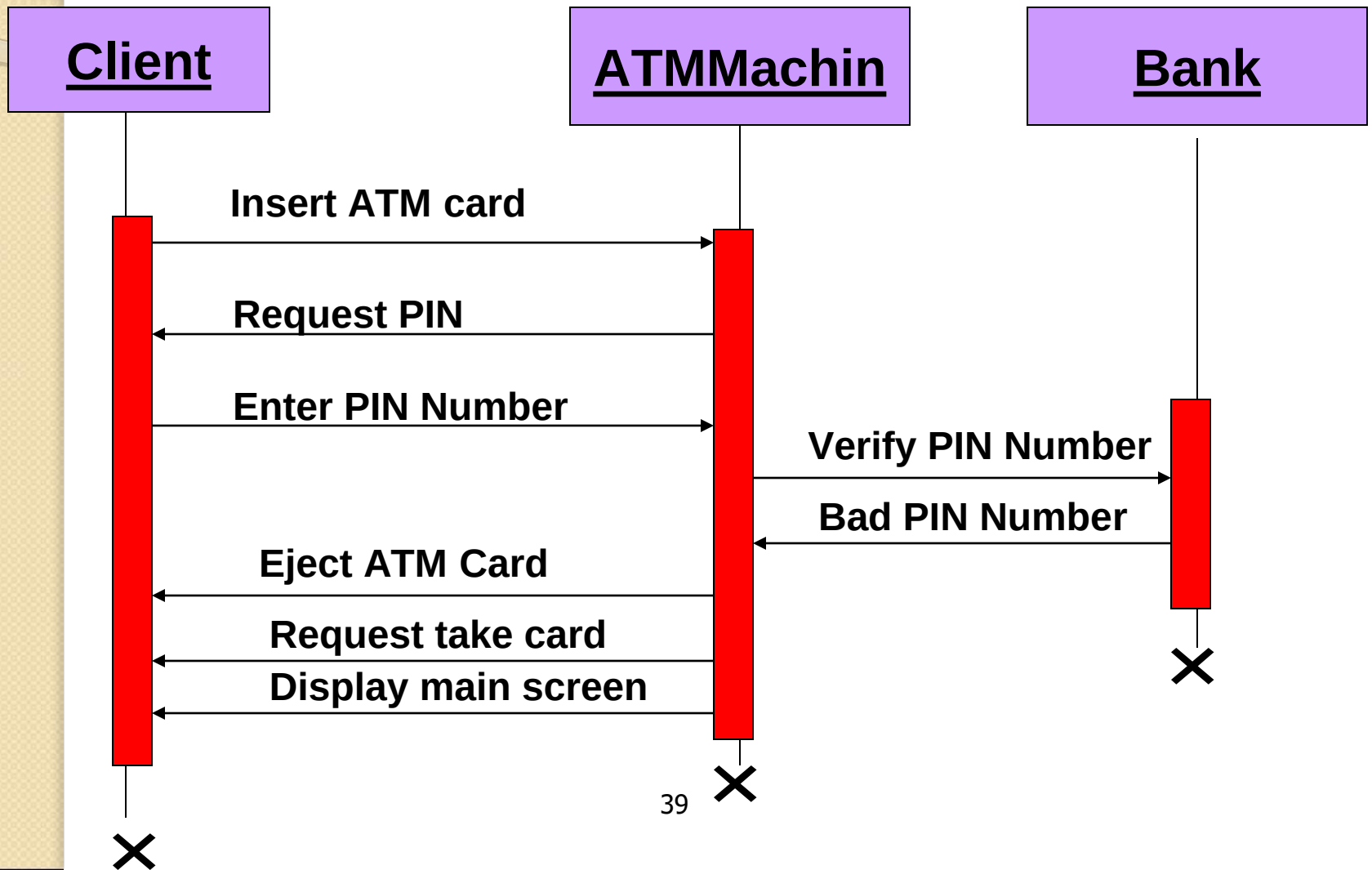


# Library

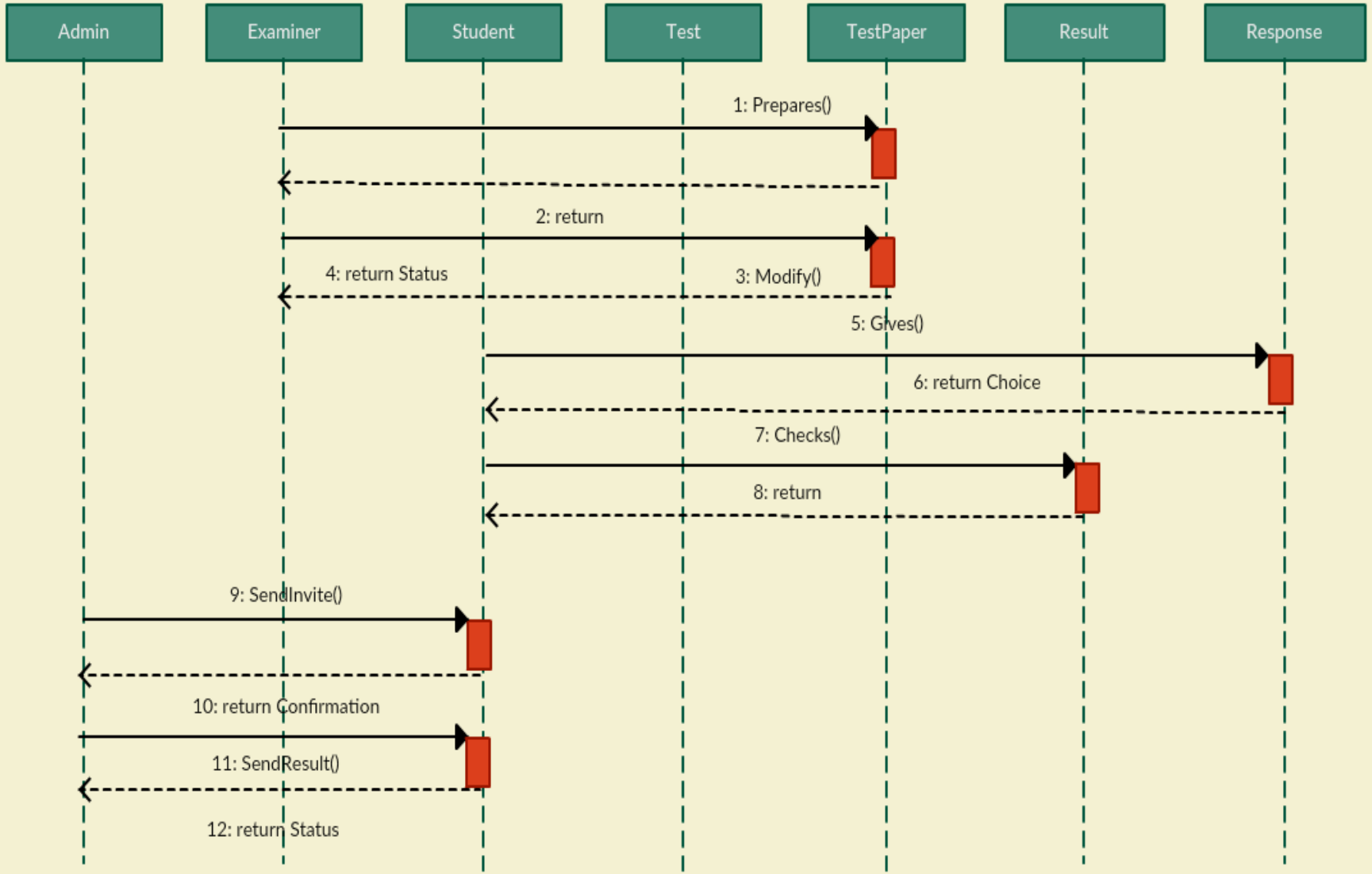

user
library computer
books database
1: login
1.1: authenticate
1.1.1: acknowledge
1.1.1.1: logged in or wrong password
2: enter book name
2.1: search for the book
2.1.1: acknowledge of the search
2.1.1.1: book found or not
3: request for issue
3.1: book status: issued
3.2: book issued
4: logout
4.1: logout succesful

# Sequence Diagram

Scenario: Invalid Pin Number



# *Sequence Diagram of an Online Exam System*



| Sequence Diagram | Activity Diagram |
| --- | --- |
| The Sequence diagram represents the UML, which is used to visualize the sequence of calls in a system that is used to perform a specific functionality. | The Activity diagram represents the UML, which is used to model the workflow of a system. |
| The Sequence diagram shows the message flow from one object to another object. | The Activity diagram shows the message flow from one activity to another. |
| Sequence diagram is used for the purpose of dynamic modelling. | Activity diagram is used for the purpose of functional modelling. |
| Sequence diagram is used to describe the behavior of several objects in a single use case | Activity diagrams is used to describe the general sequence of actions for several objects and use cases. |
| Sequence diagram is mainly used to represent the time order of a process. | Activity diagram is used to represent the execution of the process. |

# Collaboration Diagram

# Collaboration Diagram

- Collaboration diagrams are like sequence diagrams because of interaction and behavior factors.
- They are more concerned about object organization rather than sequence diagram that are more focused on a time sequence.
- They are also known as "Communication Diagram."
- These are used to represent the flow of messages between the objects.

# Collaboration Diagram

- To create a **collaboration diagram** from a **sequence diagram**: Right-click in the background of the **sequence diagram** and select Create Default **Collaboration Diagram** in the contextual menu.
- Select Tools→Create Default **Collaboration Diagram**.

# Collaboration Diagram

- The collaboration diagram and sequence diagram shows similar information but in a distinct form.
- It can portray the architecture of an object inside the system.
- It can be used to depict the relationship among various objects within the system.
- The collaboration diagram shows the nature of a specific use case.
- They are used to determine the interfaces and class responsibilities.
- Objects collaborate through passing messages to each other.

# Notations

- Objects
- Links
- Messages

# Objects

- It is represented through an object symbol depicting the object's name and their class underlined, isolated by a colon.
- You can apply the objects within the collaboration diagram as follows:
- Every object in the collaboration has a name and has a specified class.

# Links

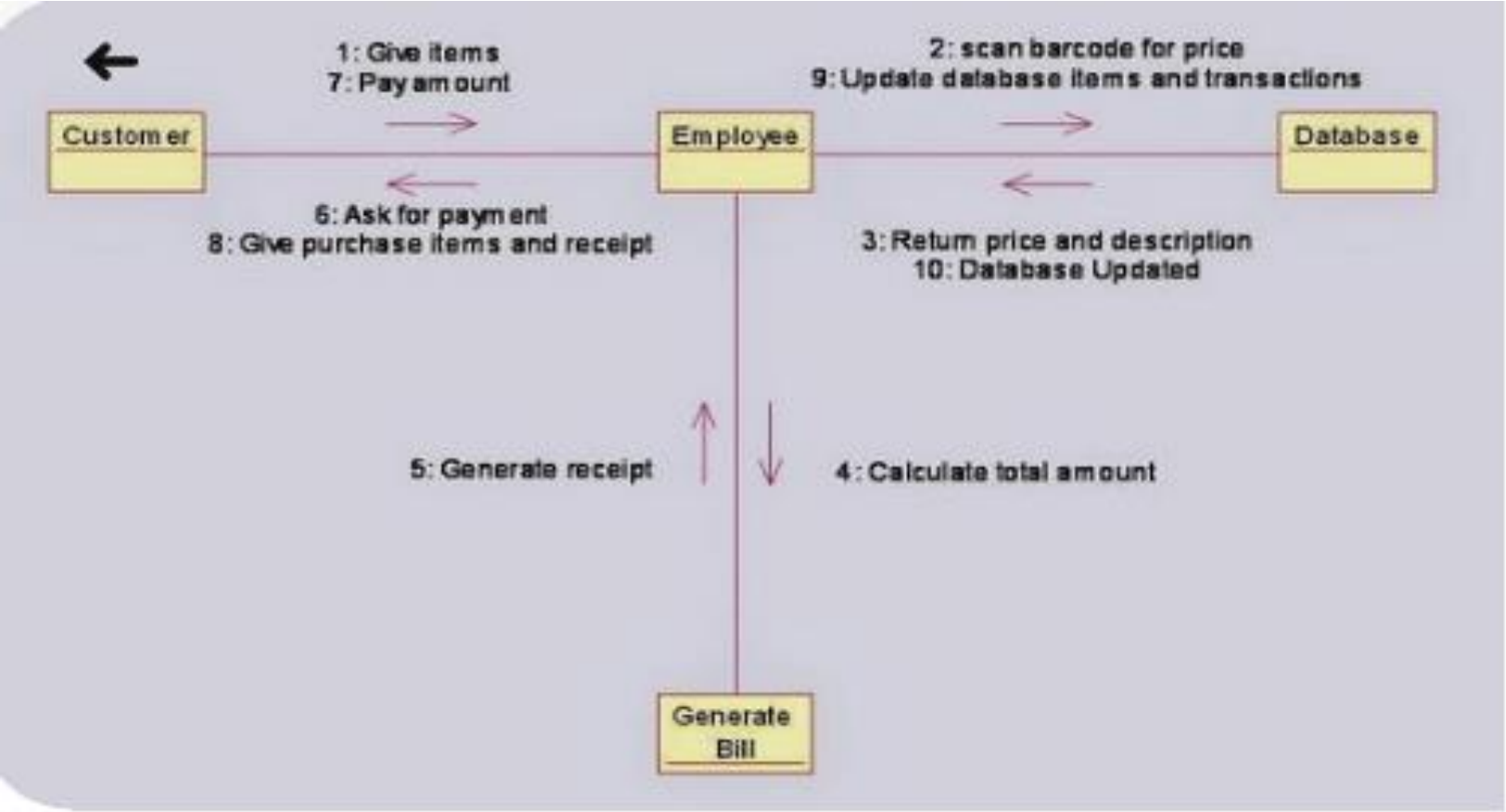
- Links connect actors and objects. Every link correlates to the association in a class diagram.
- A link can be a relation between objects for which messages are transferred. In these diagrams, a link can be displayed just like a sturdy line among two objects.

*

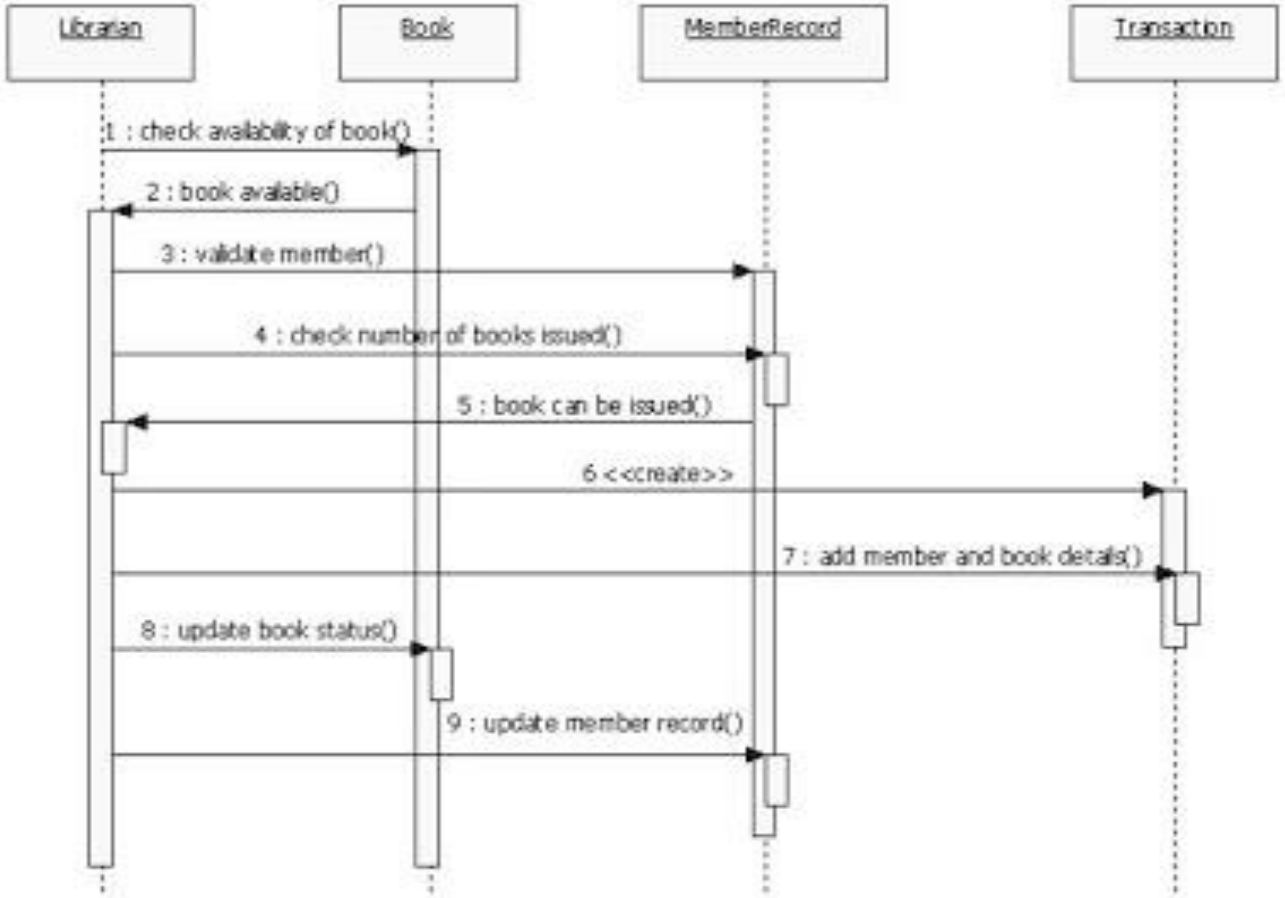
# Messages

- It is communication among objects that transmits information with an expectation that action will arise.
- A message can be represented as a specified arrow positioned near the link in the collaboration diagram.

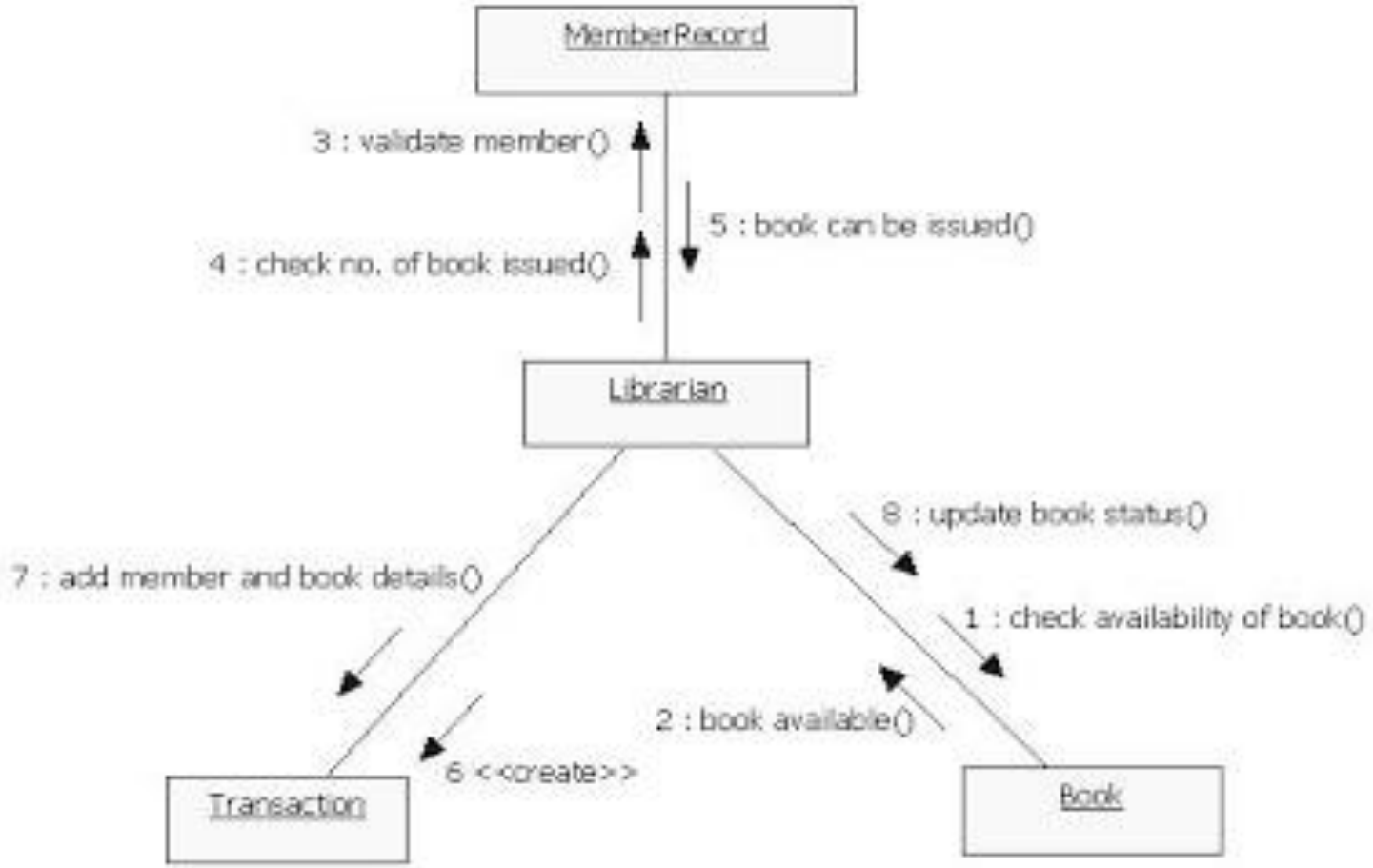
# Collaboration diagram for Online Shopping System



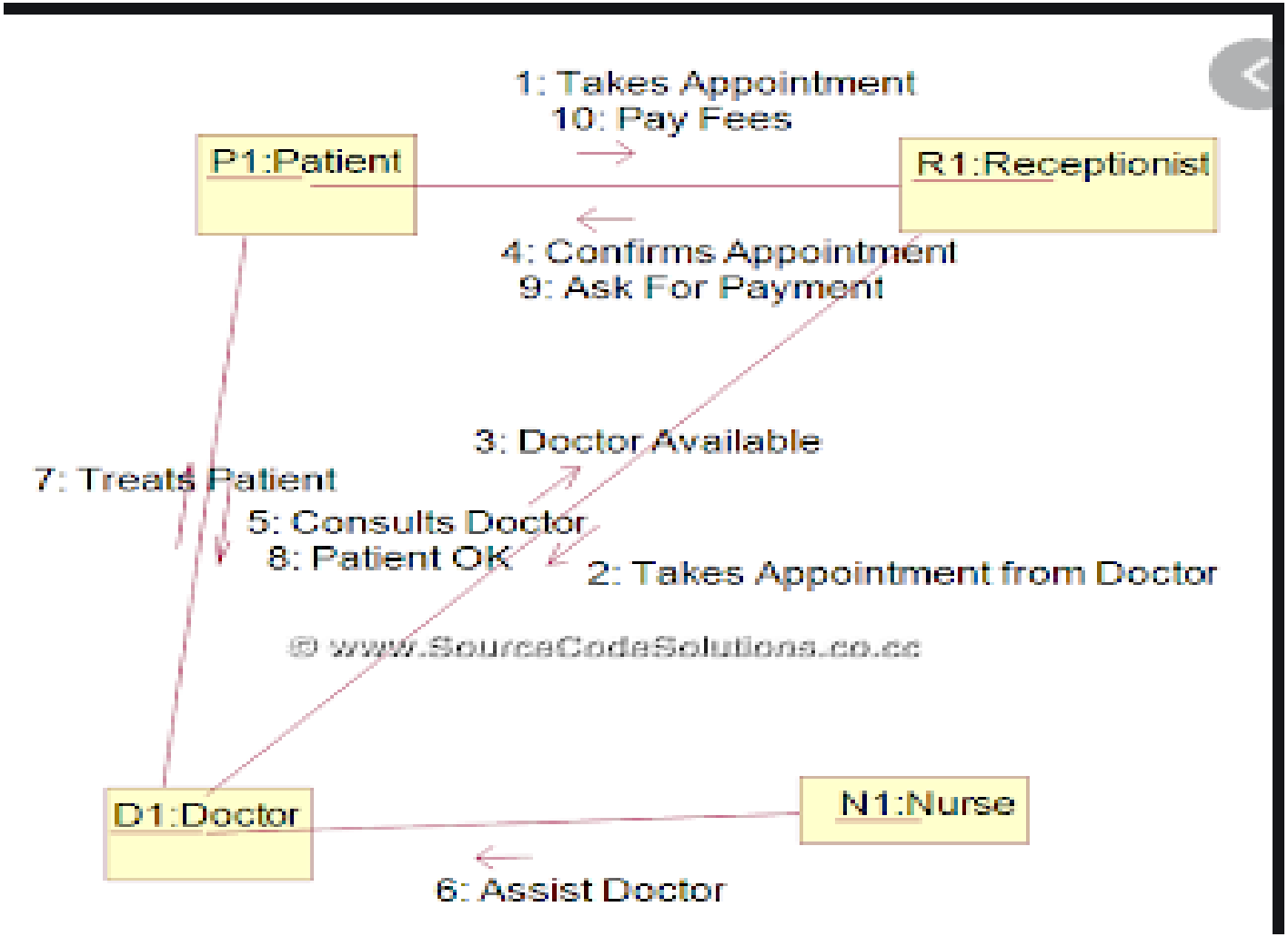
# Sequence diagram for Issue Book



# Collaboration diagram for Issue book



# Collaboration diagram for Online Hospital Management System



# Behavioural Model : State Chart Diagram

# State Chart Diagram

- State chart diagram describes different states of a component in a system. The states are specific to a component/object of a system.
- A Statechart diagram describes a state machine. State machine can be defined as a machine which defines different states of an object and these states are controlled by external or internal events.

# Purpose of Statechart Diagrams

- Statechart diagram is one of the UML diagrams used to model the dynamic nature of a system. They define different states of an object during its lifetime and these states are changed by events.
- Statechart diagrams are useful to model the reactive systems. Reactive systems can be defined as a system that responds to external or internal events.

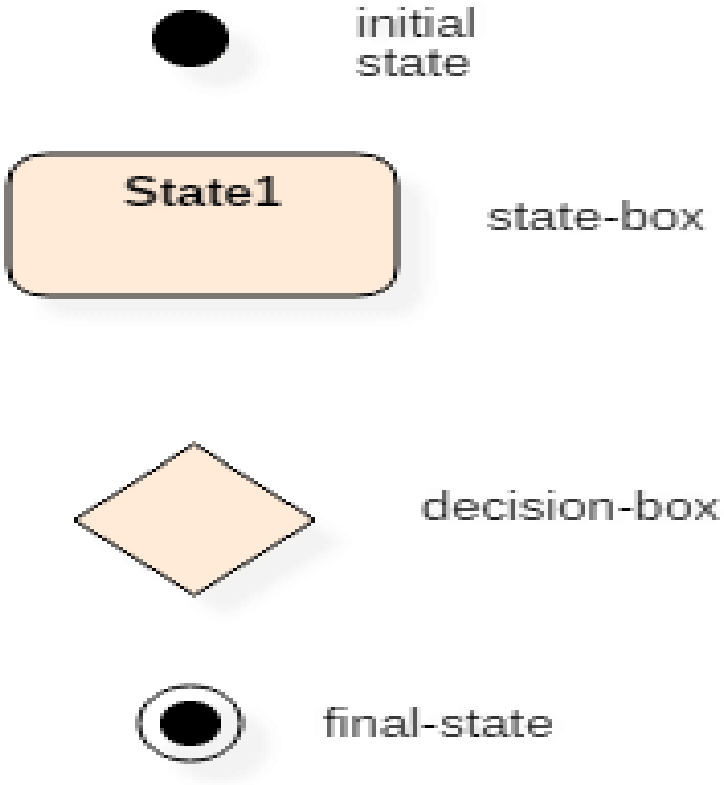
# Purpose of Statechart Diagrams

- Statechart diagram describes the flow of control from one state to another state. States are defined as a condition in which an object exists and it changes when some event is triggered.
- The most important purpose of Statechart diagram is to model lifetime of an object from creation to termination.

# Following are the main purposes of using Statechart diagrams

- To model the dynamic aspect of a system.
- To model the life time of a reactive system.
- To describe different states of an object during its life time.
- Define a state machine to model the states of an object.

# Notation and Symbol for State Machine



# How to Draw a Statechart Diagram?

- Statechart diagram is used to describe the states of different objects in its life cycle. Emphasis is placed on the state changes upon some internal or external events. These states of objects are important to analyze and implement them accurately.
- Statechart diagrams are very important for describing the states. States can be identified as the condition of objects when a particular event occurs.

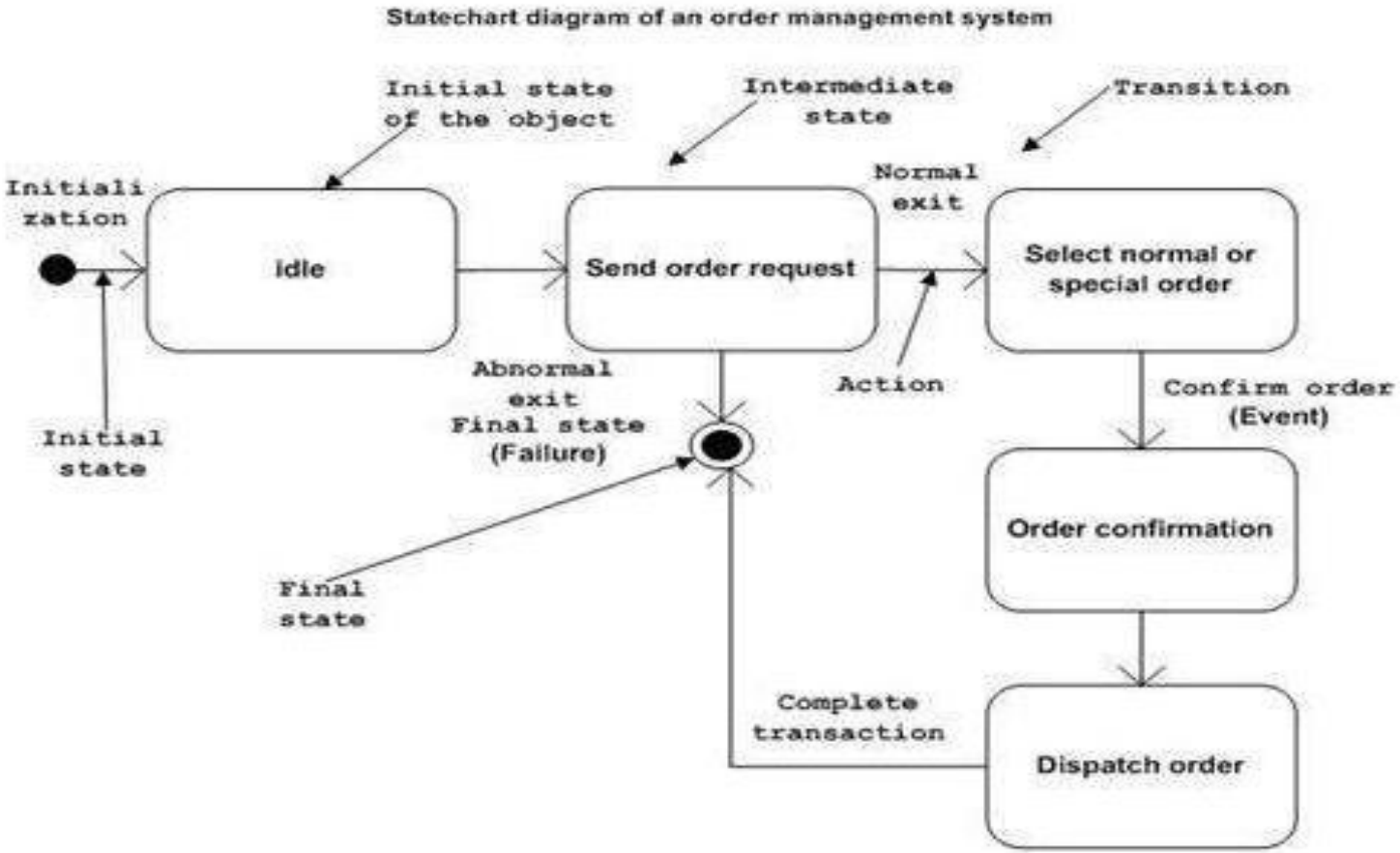
# How to Draw a Statechart Diagram?

- Before drawing a Statechart diagram we should clarify the following points −
- Identify the important objects to be analyzed.
- Identify the states.
- Identify the events.

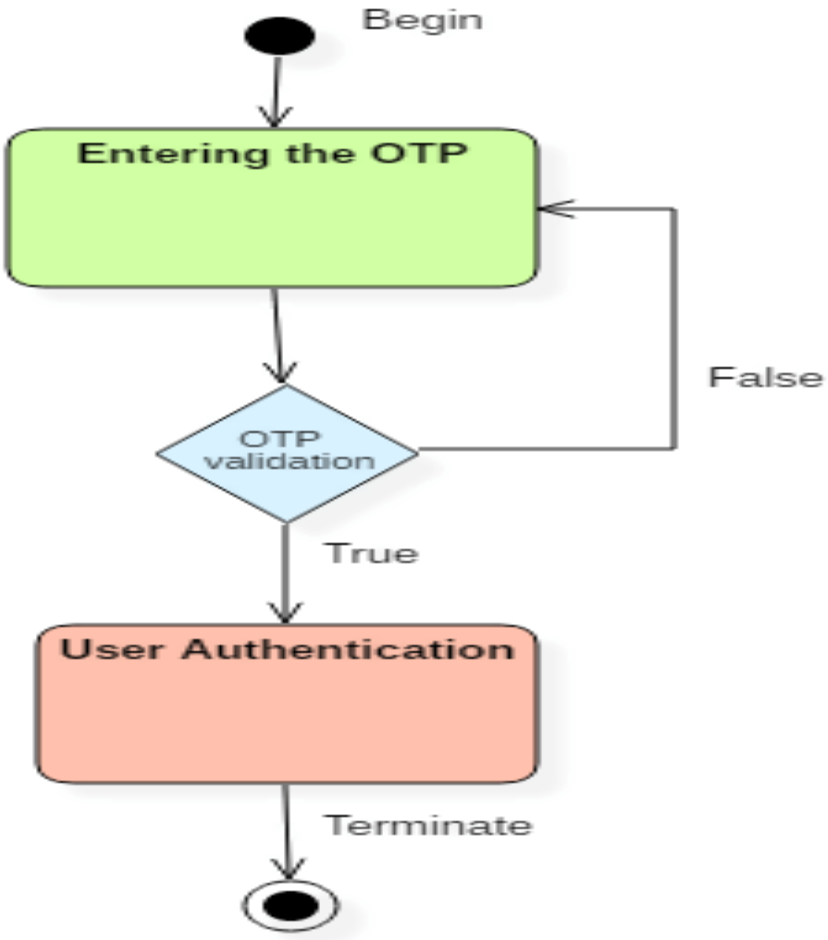
# Following is an example of a Statechart diagram where the state of Order object is analyzed

- The first state is an idle state from where the process starts. The next states are arrived for events like send request, confirm request, and dispatch order. These events are responsible for the state changes of order object.
- During the life cycle of an object (here order object) it goes through the following states and there may be some abnormal exits. This abnormal exit may occur due to some problem in the system. When the entire life cycle is complete, it is considered as a complete transaction as shown in the following figure. The initial and final state of an object is also shown in the following figure.

# State chart diagram of an order management system



# State chart diagram represents the user authentication process.



# Class-based Model

## Class Diagram

# Class Diagram

- Class diagram is a static diagram.
- It represents the static view of an application.
- Class diagram is not only used for visualizing, describing, and documenting different aspects of a system but also for constructing executable code of the software application.
- Class diagram describes the attributes and operations of a class and also the constraints imposed on the system.

# A UML class diagram is made up of:

- A set of classes and
- A set of relationships between classes

# What is a Class

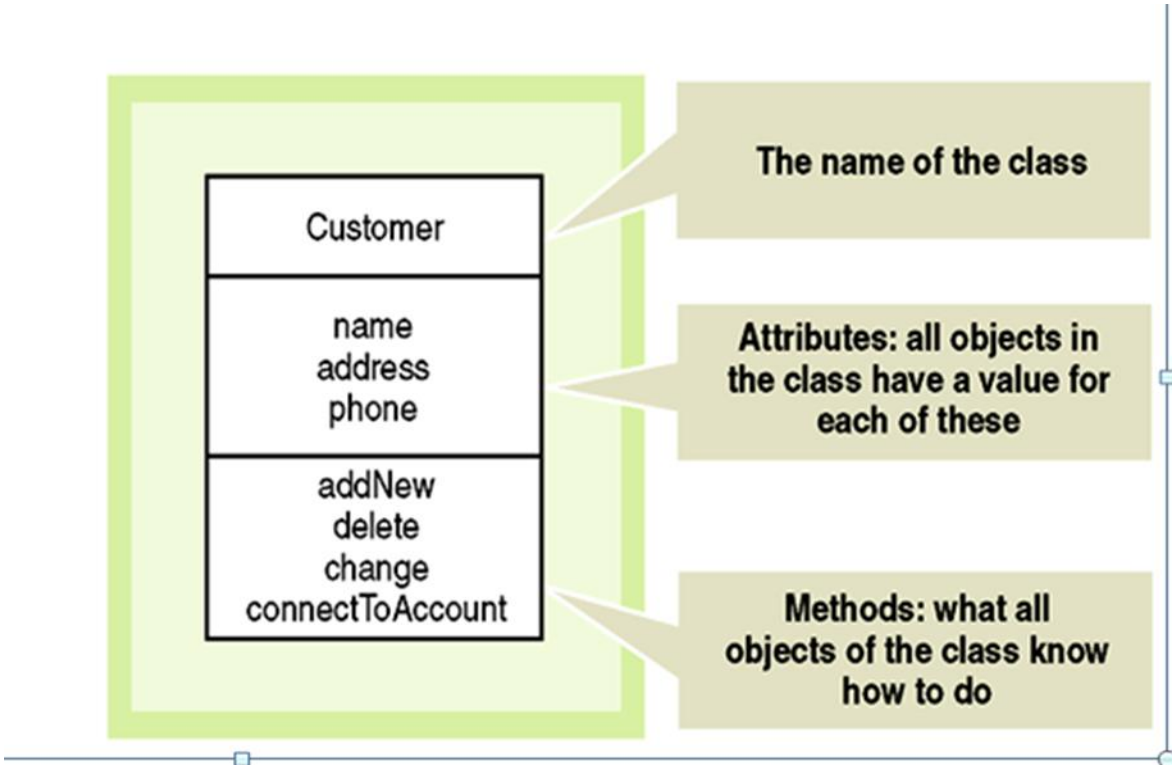
- A description of a group of objects all with similar roles in the system, which consists of:
- **Structural features** (attributes) define what objects of the class "know"
  - Represent the state of an object of the class
  - Are descriptions of the structural or static features of a class
- **Behavioral features** (operations) define what objects of the class "can do"
  - Define the way in which objects may interact
  - Operations are descriptions of behavioral or dynamic features of a class

# Class Diagram

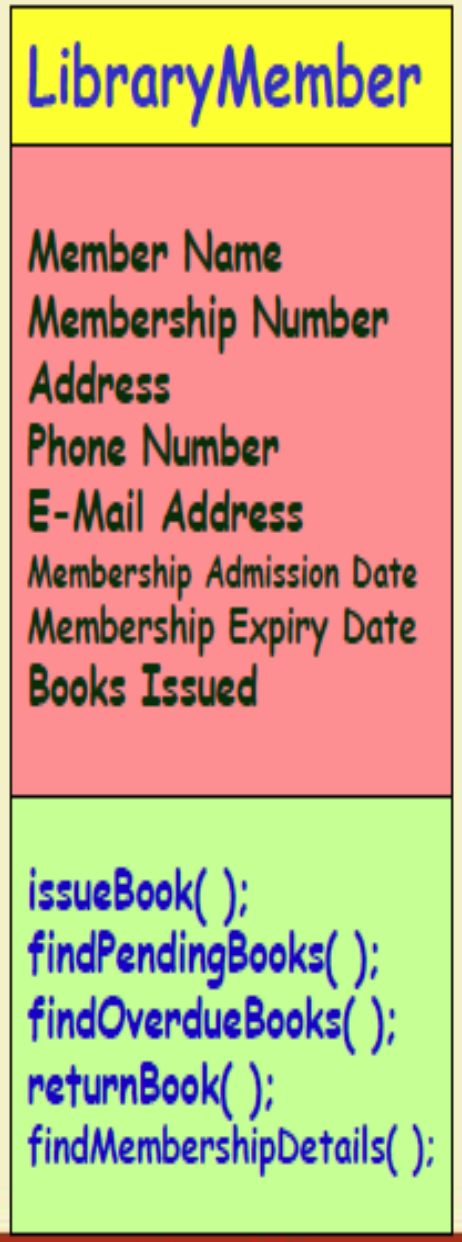
- Classes: Entities with common features, i.e. attributes and operations.
- Represented as solid outline rectangle with compartments.
- Compartments for **name, attributes, and operations.**
- Attribute and operation compartments are optional depending on the purpose of a diagram.

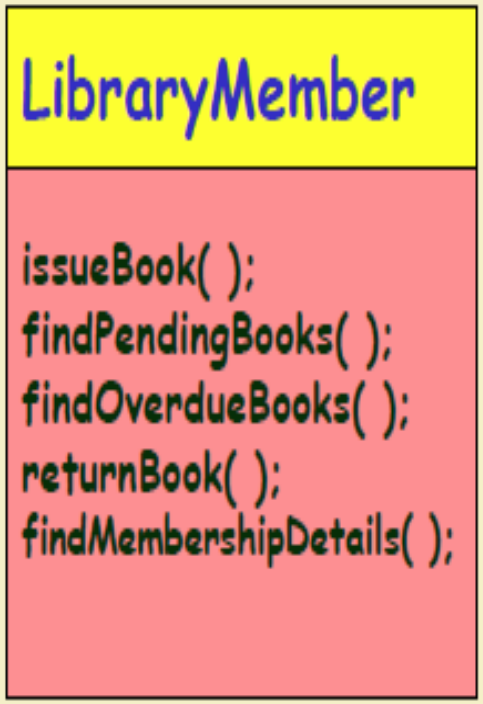
# UML Class Representation

- A class represents a set of objects having similar attributes, operations, relationships and behavior.



# Different representations of the LibraryMember class





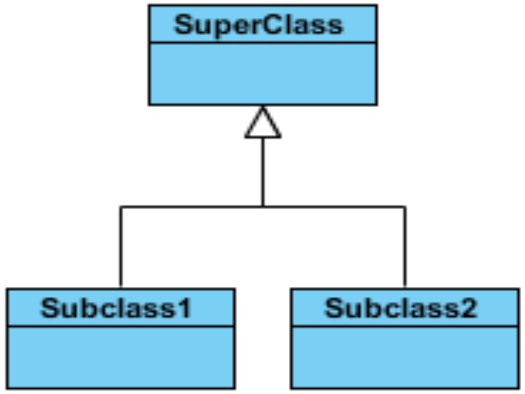
**Example UML Classes**

# What are the Different Types of Relationships Among Classes?

- Inheritance(Generalization ,Specialization)
- Simple Association
- Aggregation/Composition
- Dependency

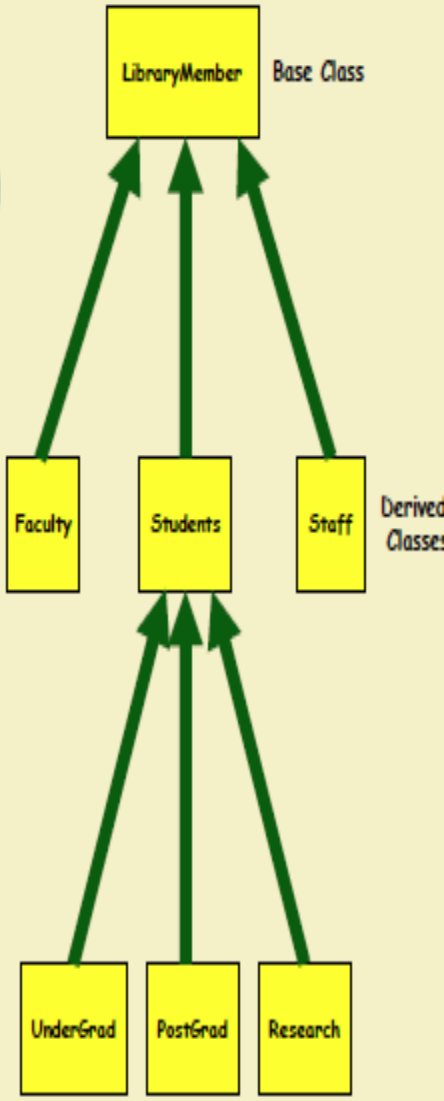
# **Inheritance** (or Generalization):

- Represents an "is-a" relationship.
- SubClass1 and SubClass2 are specializations of Super Class.
- A solid line with a hollow arrowhead that point from the child to the parent class

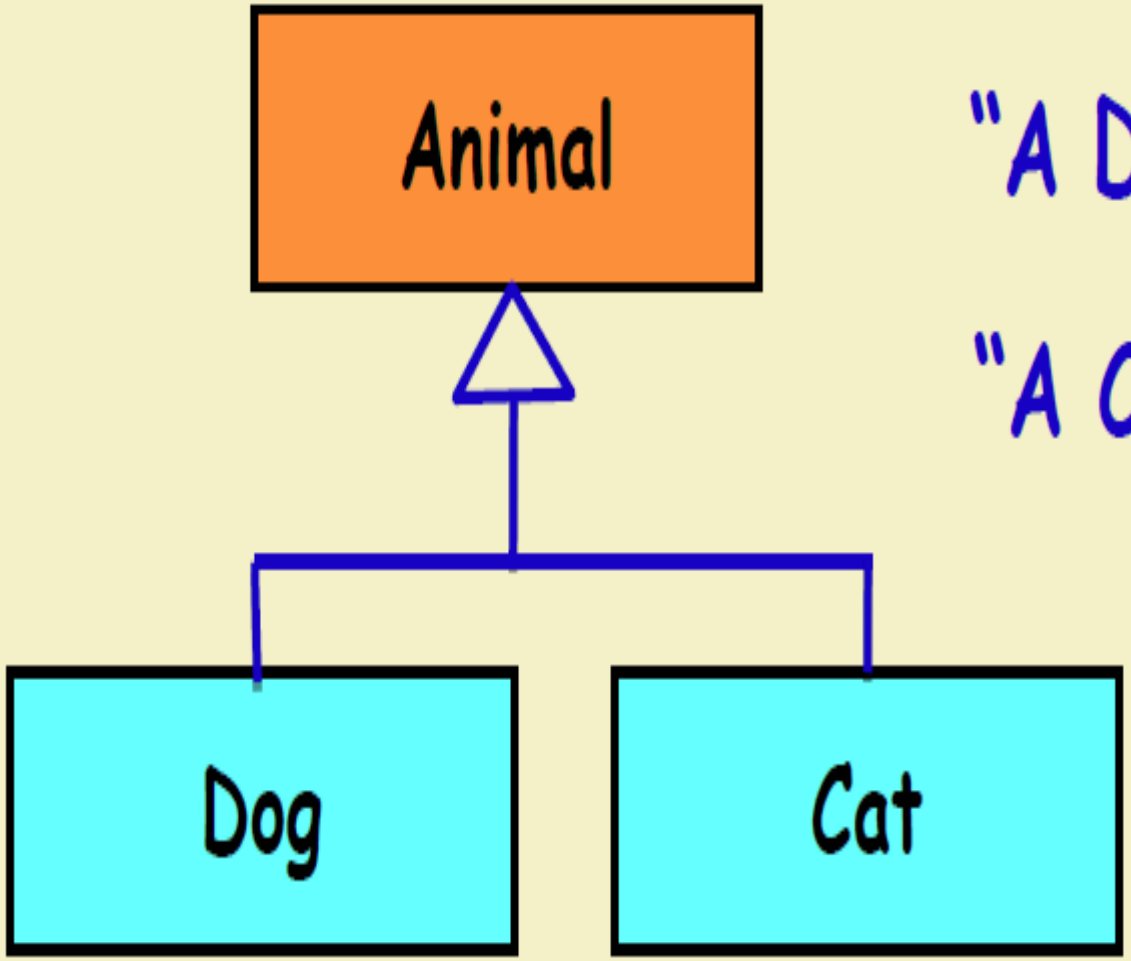


# Inheritance

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



# Inheritance Example

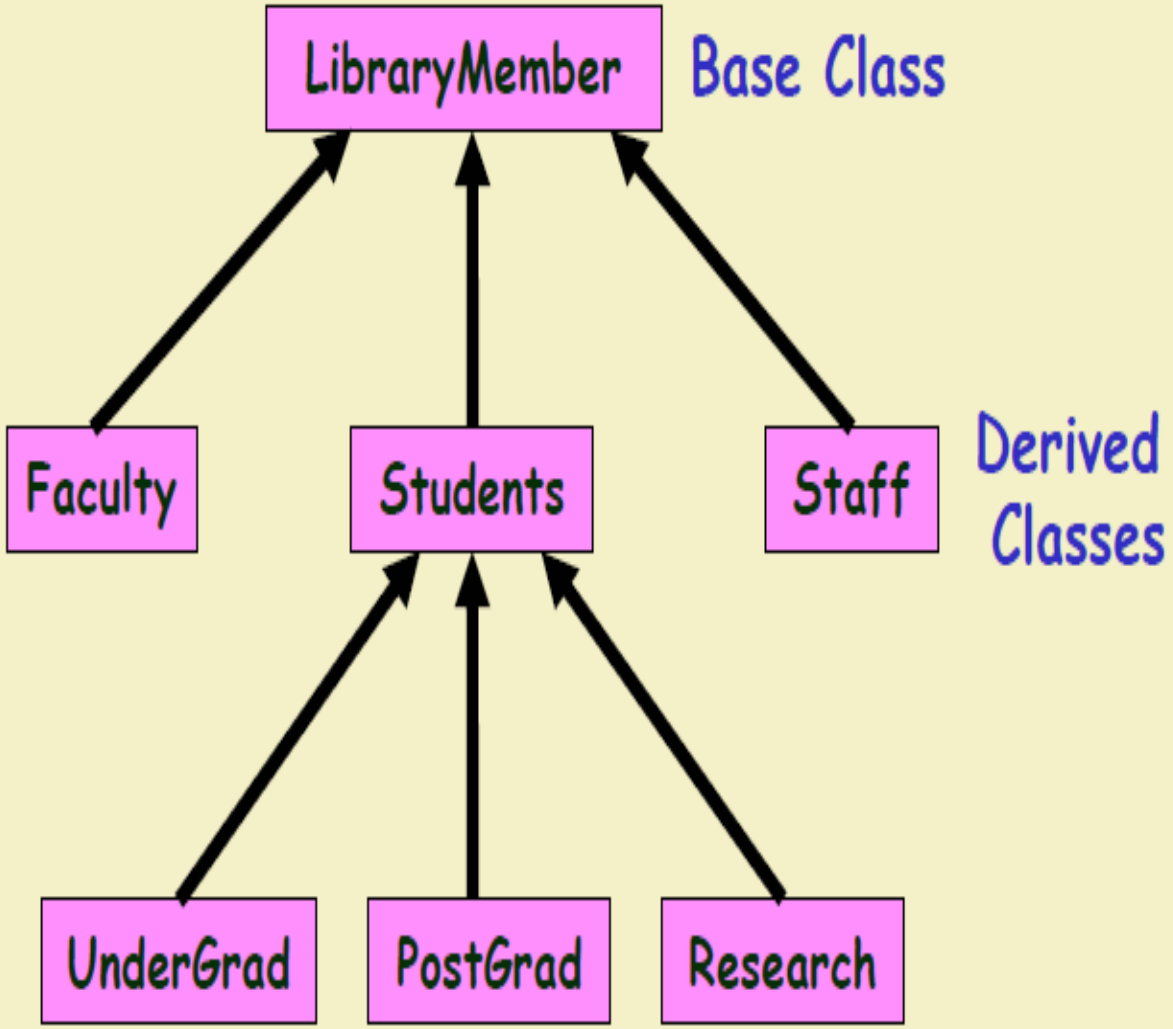



"A Dog *ISA* Animal"

"A Cat *ISA* Animal"

# Inheritance

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



# A Generalization/Specialization Hierarchy for Motor Vehicles

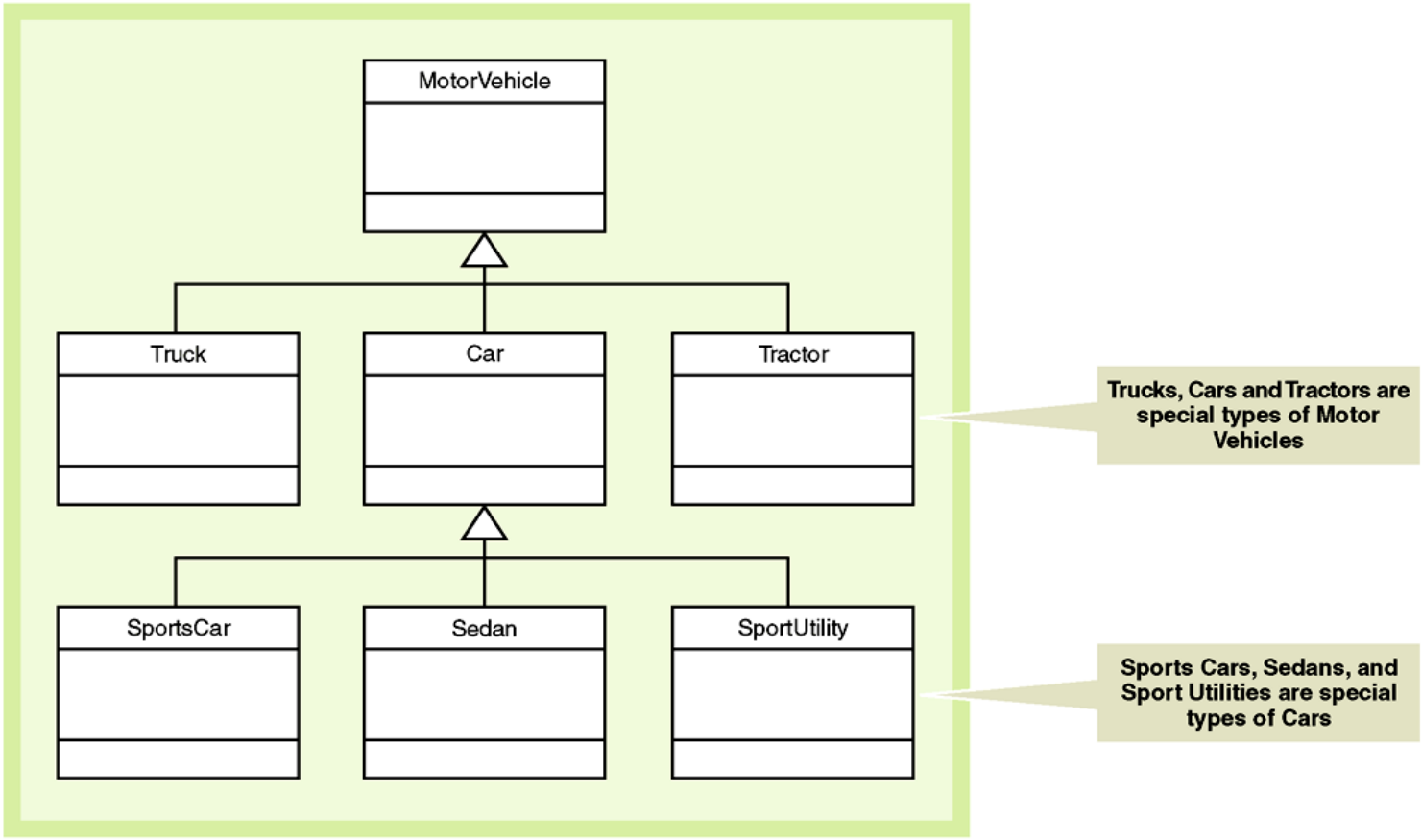


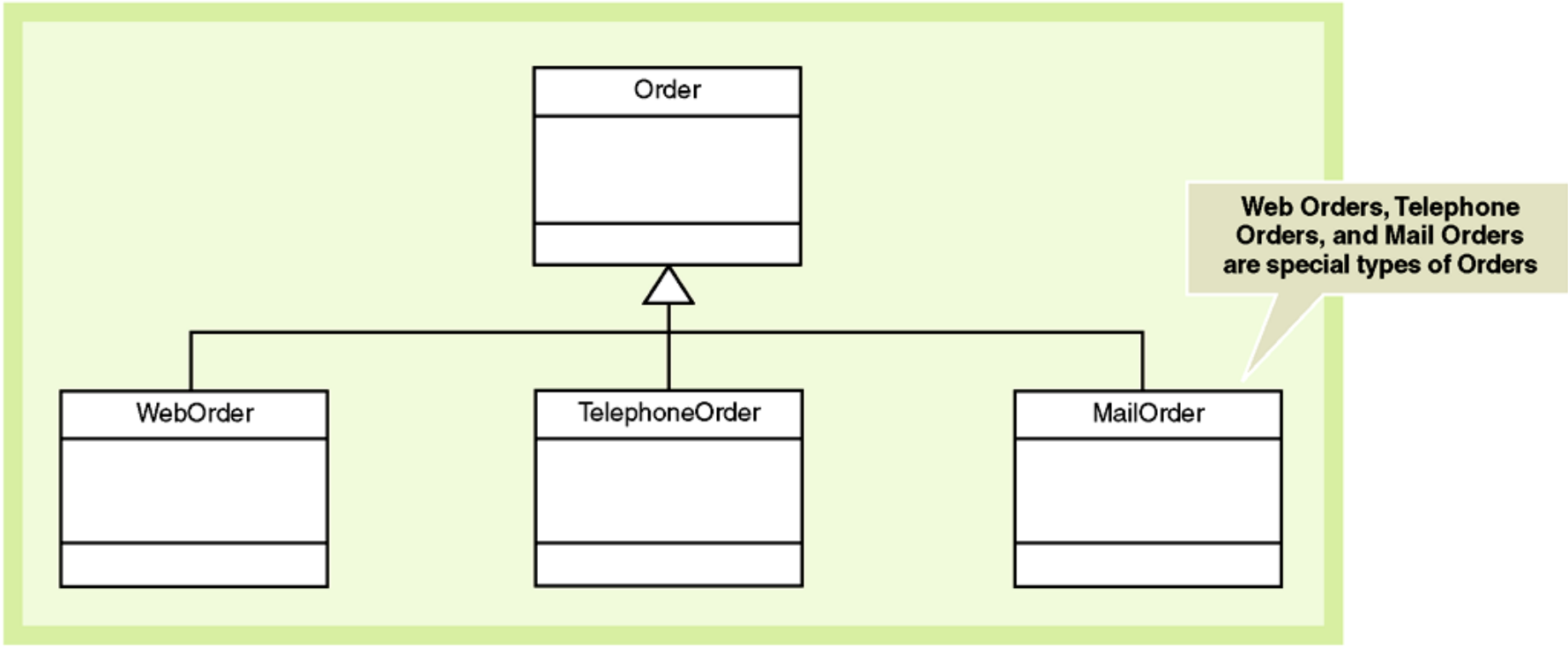

FIGURE 5-29

A generalization/specialization hierarchy for motor vehicles.

# A Generalization/Specialization Hierarchy for Orders

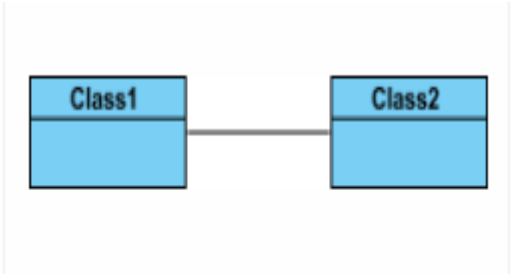
FIGURE 5-30

A generalization/specialization hierarchy for orders.

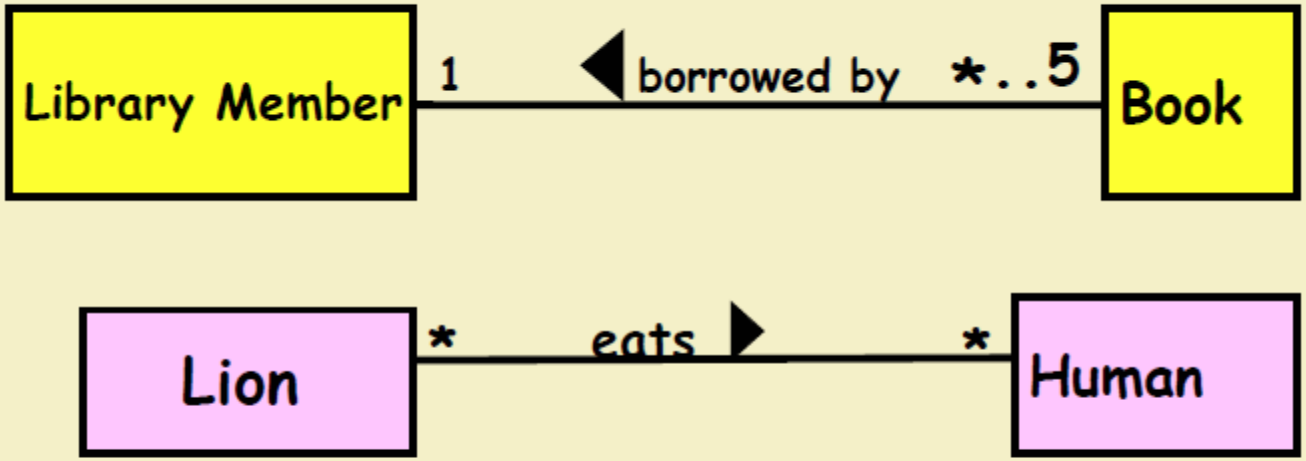


# Simple Association:

- A structural link between two peer classes.
- There is an association between Class1 and Class2
- A solid line connecting two classes



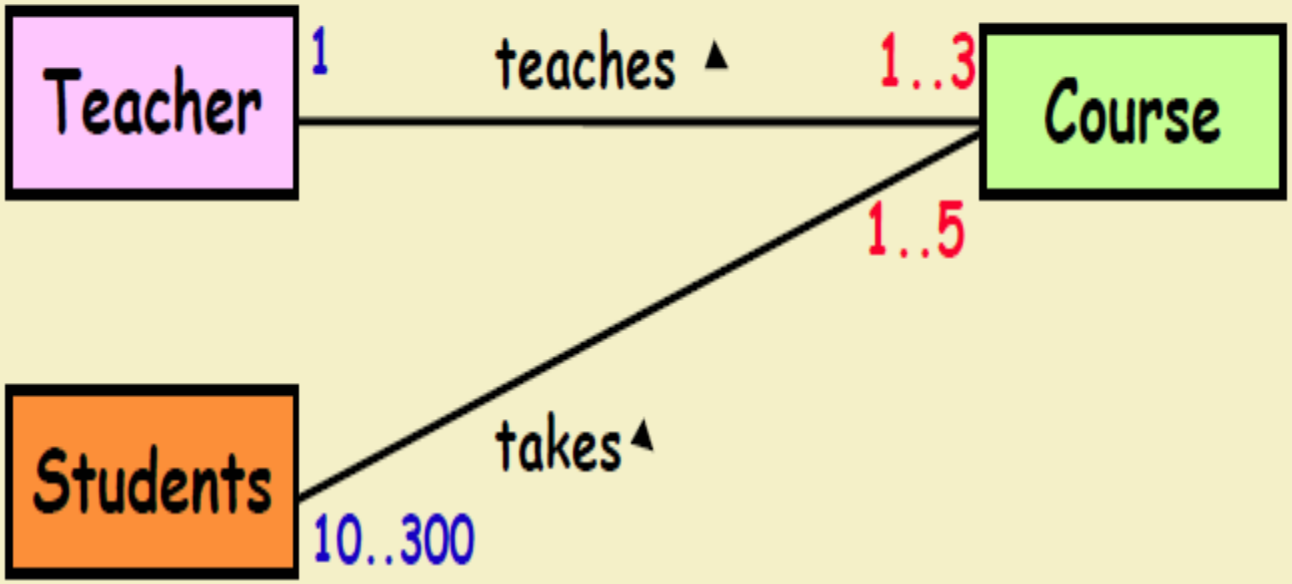
# Association - More Examples




**Multiplicity:** The number of objects from one class that relate with a single object in an associated class.

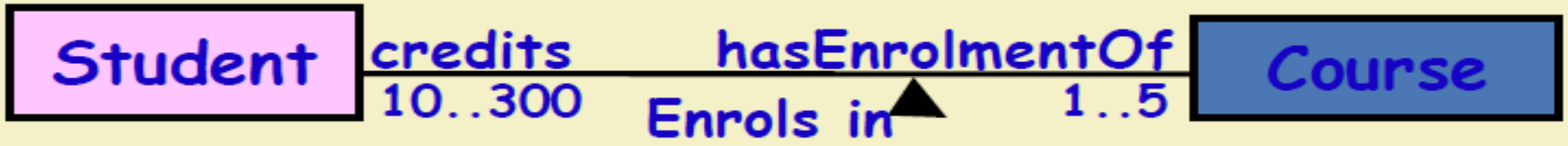
# Association - Multiplicity

- A teacher teaches 1 to 3 courses (subjects)
- Each course is taught by only one teacher.
- A student can take between 1 to 5 courses.
- A course can have 10 to 300 students.

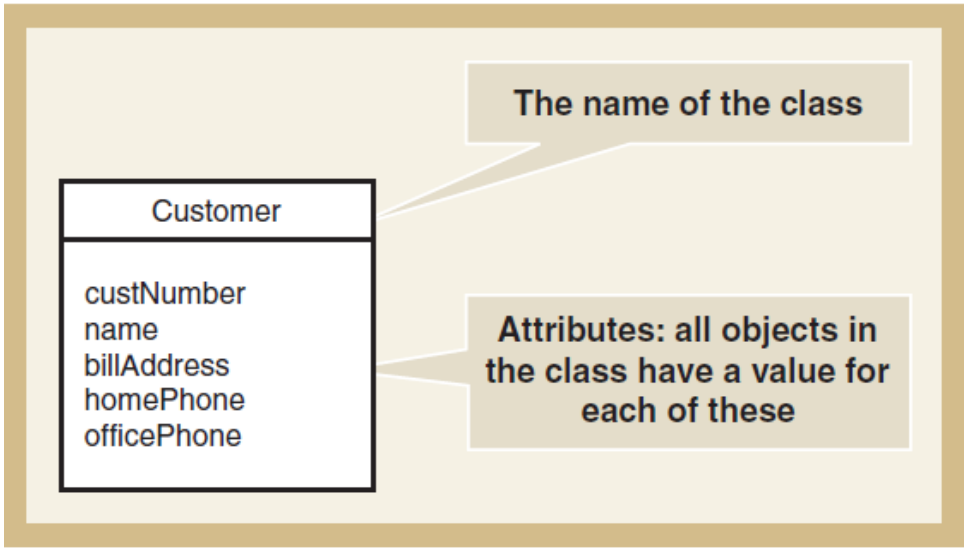


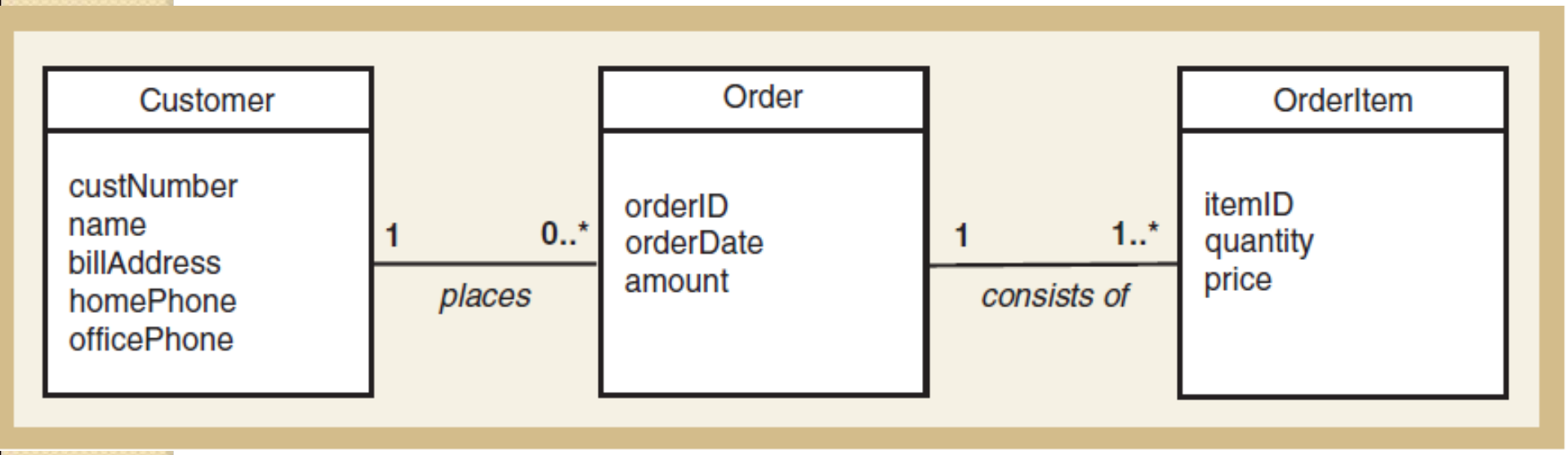
# Quiz: Draw Class Diagram

- A Student can take up to five Courses.
- A student needs to enroll in at least one course.
- Up to 300 students can enroll in a course.
- An offered subject in a semester should have at least 10 registered students.



# Domain model class diagram



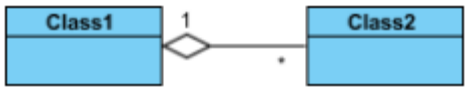



# Multiplicity of association

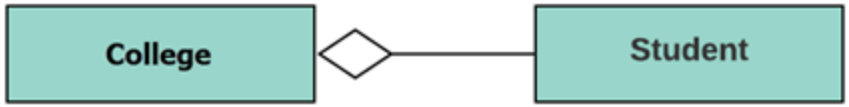
| Indicator | Meaning |
|---|---|
| 0..1 | Zero or one |
| 1 | One only |
| 0..* | 0 or more |
| 1..* \| * | 1 or more |
| n | Only n (where n > 1) |
| 0..n | Zero to n (where n >1) |
| 1..n | One to n (where n > 1) |

# Aggregation:

- A special type of association. It represents a "part of" relationship.
- Class2 is part of Class1.
- Many instances (denoted by the *) of Class2 can be associated with Class1.
- Objects of Class1 and Class2 have separate lifetimes.
- A solid line with an unfilled diamond at the association end connected to the class of composite

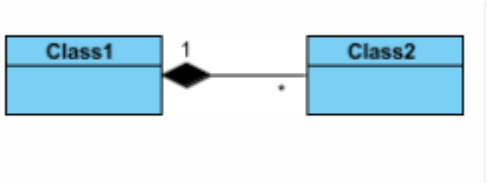


# Aggregation



# Composition:

- A special type of aggregation where parts are destroyed when the whole is destroyed.
- Objects of Class2 live and die with Class1.
- Class2 cannot stand by itself.
- A solid line with a filled diamond at the association connected to the class of composite

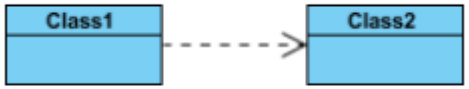


# Aggregation & Composition

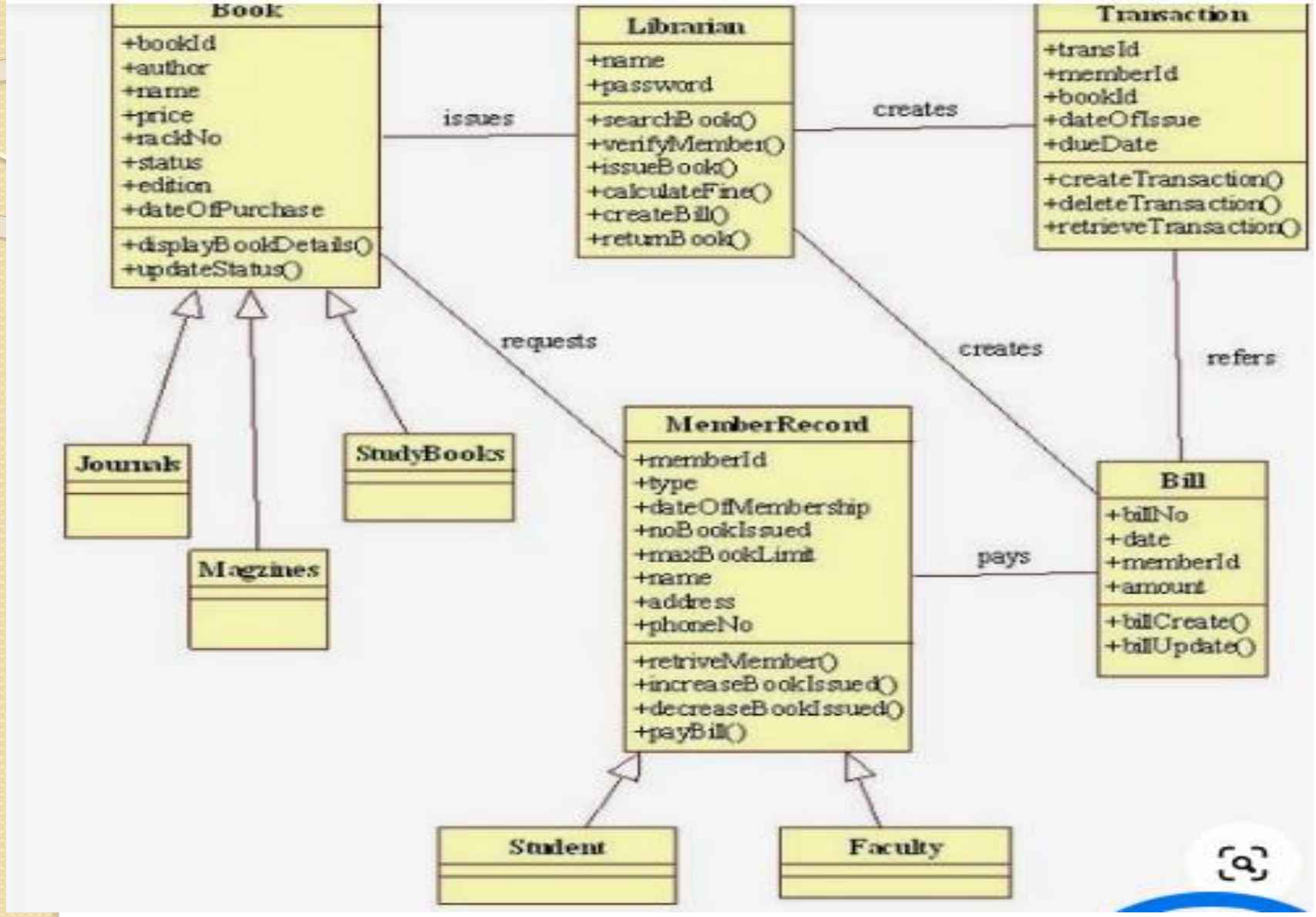
- **Aggregation** implies a relationship where the child can exist independently of the parent. Example: **Class** (parent) and **Student** (child). Delete the Class and the Students still exist.
- **Composition** implies a relationship where the child cannot exist independent of the parent. Example: **House** (parent) and **Room** (child). Rooms don't exist separate to a House.

# Dependency:

- Exists between two classes if the changes to the definition of one may cause changes to the other (but not the other way around).
- Class1 depends on Class2
- A dashed line with an open arrow



# Class Diagram for Library Management System



# Hospital Management System