

# Object Oriented Modeling, Analysis and Design using UML

**UE17MC603**

# Course description

- **OBJECTIVE:**

To understand the Unified Modeling Language and orient towards Object Oriented methodology using UML for modeling software systems.

- **TARGET AUDIENCE:**

In particular, it is intended for **software professionals** who have sound knowledge of object concepts and some experience towards analysis and design.

- **PREREQUISITES:**

Good understanding of object concepts.

Sound knowledge of any object oriented language.

Knowledge of software engineering process.

# OOMAD

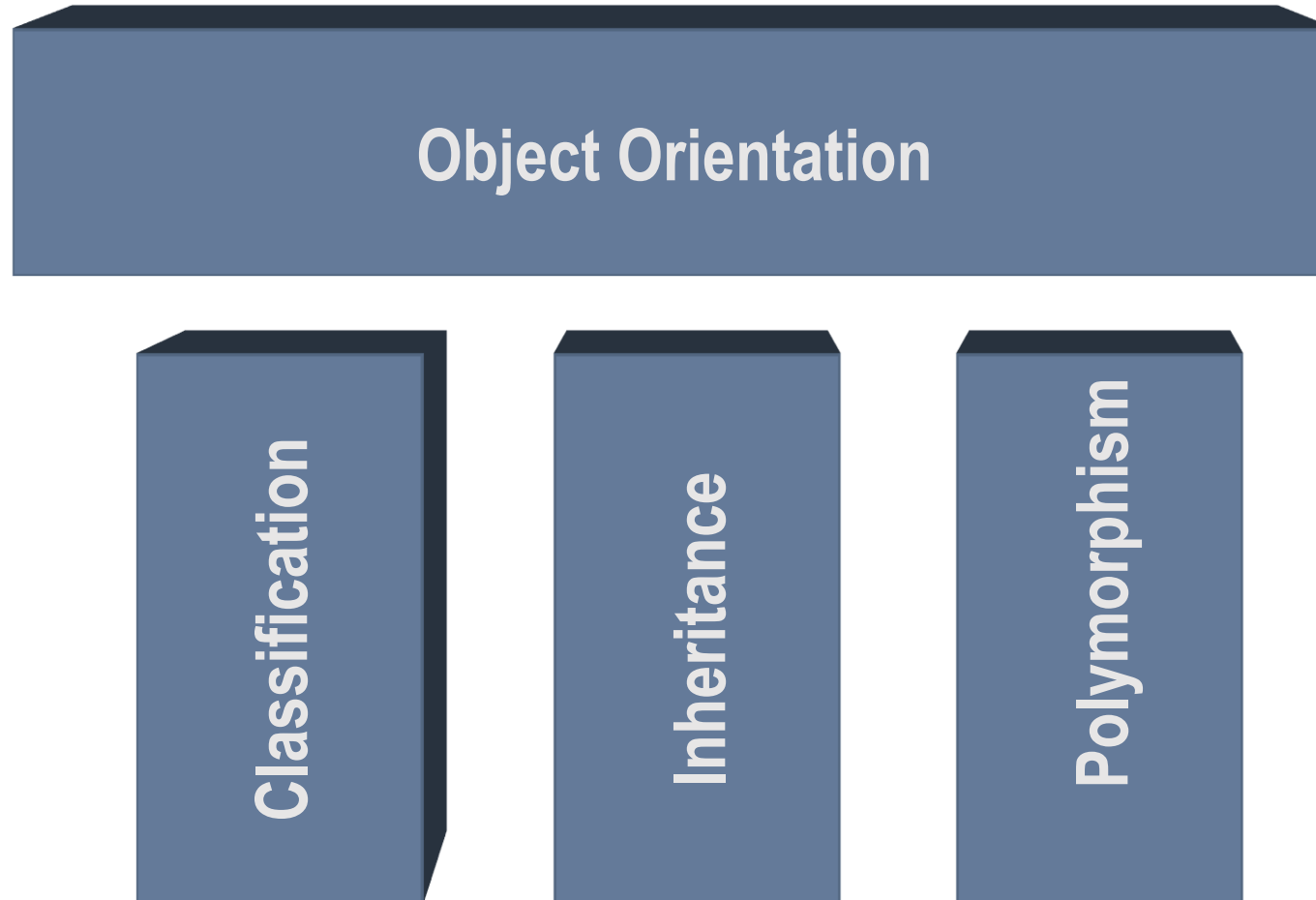
- Text Book1:
  - Michael Blaha, James Rumbaugh; Object-Oriented Modeling and Design with UML, 2<sup>nd</sup> Edition, Pearson Education/PHP Publication, 2005.  
**(Unit 2 to 5)**
- Text Book 2:
  - Dr. IKVinderpal Singh, ER Sangeeta Bhandari, Object Oriented Modeling, Analysis and Design, Khanna Publishing  
**(Unit 1)**

# Object Oriented Approach

# Introduction to Object Orientation Topics

- ★ 1. Basic Principles of Object Orientation
- 2. OO Methodology
- 3. OO Themes

# 1. Basic Principles of Object Orientation



- OO approach include the following aspects or characteristics
  1. Identity & Classification
  2. Inheritance
  3. polymorphism

**Identity:** Data is quantized into discrete, distinguishable entities called objects.

- Each object has its own inherent identity.
- Ex: PC, Bicycle, queen in chess etc., File in a file system, Scheduling policy in a multiprocessing operation.

# Classification

- It means that objects with same data structure(attributes) and behaviour(operations) are grouped into a class.
- For example:

## Objects



## Class

### **Attributes(members)**

Frame size

Wheel size

### **Operations(methods)**

move, repair



# Inheritance

**Inheritance:** is sharing of attributes and operations among classes based on a hierarchical relationship.

- Ex: Scrolling window & Fixed window are sub classes of Window class.

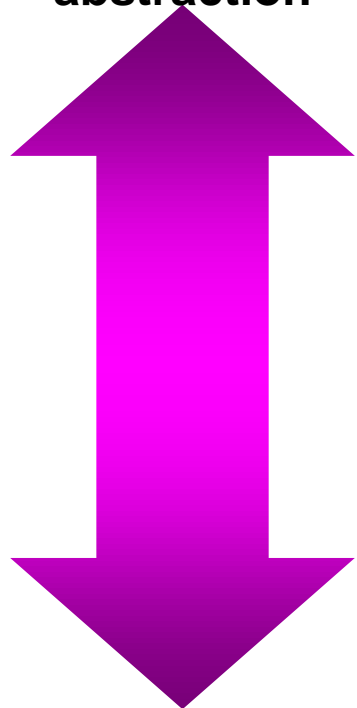
Elements at the same level of the hierarchy should be at the same level of abstraction

# Inheritance...

**Asset**

Levels of abstraction

Increasing  
abstraction



Decreasing  
abstraction

BankAccount

Securities

RealEstate

Savings

Checking

Stock

Bond

Property

***Elements at the same level of the hierarchy  
should be at the same level of abstraction***

# Polymorphism

- It means that the same operation may behave differently for different classes.
- For example, in Chess game **move operation** behaves differently for different pawns



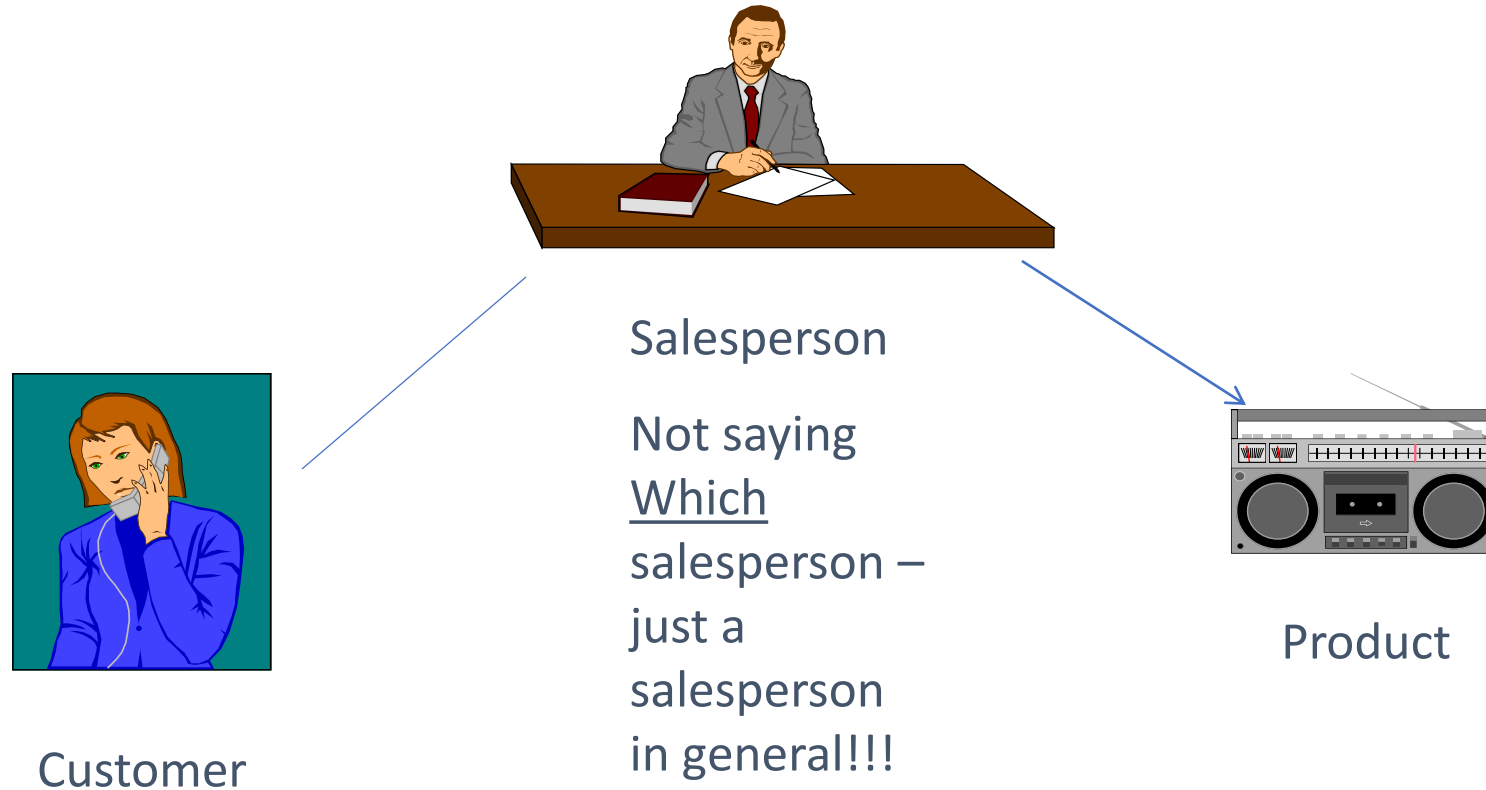
## 2. Object Oriented Methodology

- The process consists of building a model of an application and the adding details to it during design.
- The methodology has following stages
  - System Conception
  - Analysis
  - System Design
  - Class design
  - Implementation

# 3. Object Oriented Themes

- Several Themes pervade OO Technology
- They are,
  - ❖ Abstraction
  - ❖ Encapsulation
  - ❖ Combining data and behaviour
  - ❖ Sharing

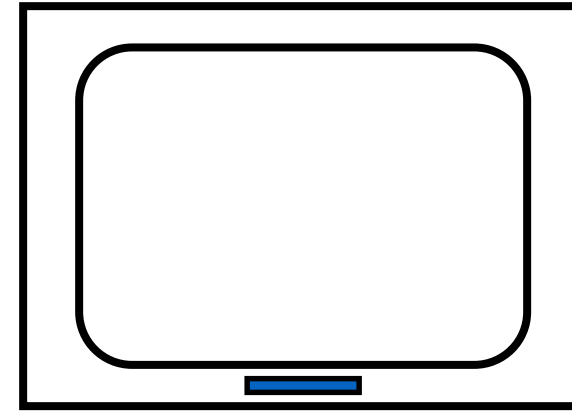
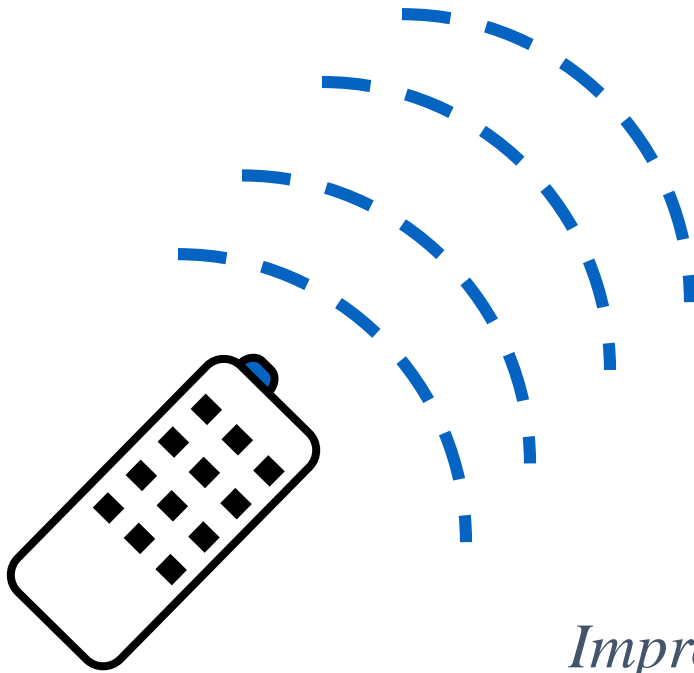
# 1. Abstraction



*“It emphasis on only essential things in application while ignoring rest of the details”*

## 2. Encapsulation

- Hide implementation from clients
  - Clients depend on interface

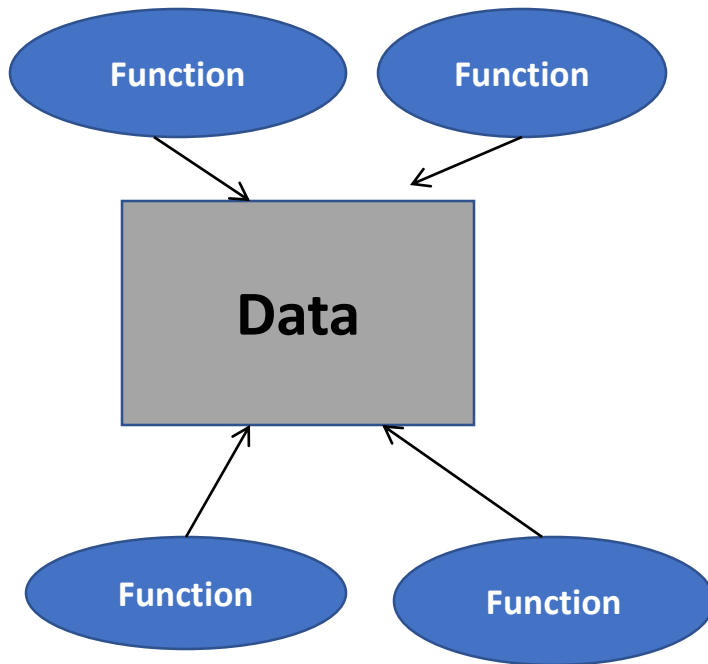


How does an object encapsulate?  
What does it encapsulate?

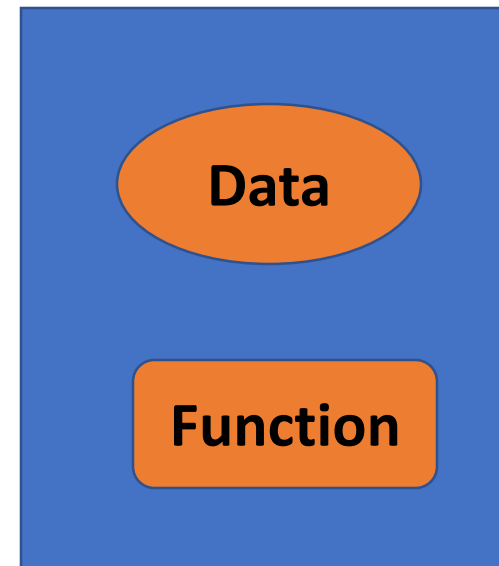
*Improves Resiliency*

# 3. Combining Data and Behaviour

- **Old approach**



## **OO approach**





# 4. Sharing

Contents

View



Edit



Rules

Sharing

## Sharing for “Documentation”

You can control who can view and edit your item using the list below.

Name	Can add	Can edit	Can view	Can review
 Logged-in users	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
 Jayne Smythe (smythe)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

☒ **Inherit permissions from higher levels**  
By default, permissions from the container of this item are inherited. If you disable this, only the explicitly defined sharing permissions will be valid. In the overview, the symbol  indicates an inherited value. Similarly, the symbol  indicates a global role, which is managed by the site administrator.



# Object Oriented Modeling using UML

# Introduction

- A model is a **simplification of reality**, which provides the **blueprints** of a system.
- We use models in almost everything that we do, for example:
  - When we get up in the morning and get ready to leave for work, we have a rough plan of the day ready in our mind – *model of the day*.
  - Similarly, an engineer prepare outline of building- *blueprint*
  - Authors prepare rough Table of Content(TOC) for book- *contents of book*

# Why do we use modeling?

- Modeling helps us in having an abstraction of something for the purpose of understanding it before building it.
- It is easier to work with.
- It is easy to understand.
- We can identify potential problems early and find solutions.
- The end user/customer gets early indications of what to expect.

# Modeling

- A model provides a means for **conceptualization** and **communication of ideas** in a precise and unambiguous form.
- Most modelling techniques used for analysis and design involve **graphic languages**, which are **made up of sets of symbols**. These symbols are used according to certain rules of methodology for communicating complex relationships of information.
- Modeling is used in many phases of software life cycle such as analysis, design, and implementation and it is iterative in nature
- Models serves several purpose:
  1. Testing a physical entity before building it.
  2. Communication with customers
  3. Visualisation
  4. Reduction of complexity.

# Abstraction

- Abstraction is the selective examination of certain aspects of a problem.
- The goal of abstraction is “*to isolate those aspects that are important and suppress those aspects that are unimportant*”.

## Importance of Modeling

Through modelling we can achieve **four aims**

1. Models help us to **visualize** a system as it is or as we want it to be.
2. Models permit us to *specify* the **structure** or **behaviour** of a system.
3. Models give us a **template** that guides us in constructing a system.
4. Models **document** the decisions we have made.

# Principles of Modeling

- There are **four principles** of modelling:
  1. The choice of **what to create** has a profound influence on **how a problem is attached** and **how a solution is shaped**.
  2. Every model may be expressed at different **level of precision**.
  3. The best model is **connected to reality**.
  4. No single model is sufficient. Every nontrivial system is best approached through small set of nearly **independent models**.

# Unified Modeling Language (UML)

- The UML is a **visual language** for specifying, constructing and documenting the artifacts of system.
- It is a standard language for writing **software blueprints**.

## Overview of UML

- The UML is used to
  - **visualize,**
  - **specify,**
  - **construct** and
  - **document** the artifacts of software system.



## 1. The UML is a language for Visualizing:

- Writing models in UML addresses the third issue: an explicit model facilitates communication.
- Some things are best modelled textually; while the others are modelled graphically.

## 2. The UML is a language for specifying:

- Specifying means building models that are precise, unambiguous and complete.
- The UML addresses the specification of all the important analysis, design and implementation **decisions that must be made in developing and deploying** a software-intensive system.

### 3. The UML is a language for Constructing:

- The UML models can be **directly connected to a variety of programming languages**. As a result, it is possible to map from a model in the UML to a programming language such as Java, C++ or Visual Basic or even to tables in a relational database.

### 4. The UML is a language for Documenting:

- A healthy software organization produces all sorts of artifacts in addition to raw executable code. These artifacts include:
  - Requirements
  - Architecture
  - Design
  - Source code
  - Project plans
  - Tests
  - Prototypes
  - Releases

# A Brief Historical Perspective

- The UML is the primary modelling language used to analyse, specify and design software systems.
  - ▶ In 1991 at GE R&D led to development of Object Modelling Technique(OMT).
  - ▶ In 1994 **Jim Rumbaugh** joined Rational and began working with **Grady Booch** on unifying Booch notations.
  - ▶ In 1995 **Ivar Jacobson** joined this team.
  - ▶ In 1996 Object Management Group(OMG) accepted proposals from Rational team which results the UML(Unified Modelling Language).

# Usage of UML:

- Primarily for software-intensive systems. Also used effectively for domains like:
  - Enterprise information systems
  - Banking and financial services
  - Telecommunications
  - Transportation
  - Defence / aerospace
  - Retail
  - Medical electronics
  - Scientific
  - Distributed Web-based services
- UML is not limited to modelling software. In fact it is expressive enough to **model non-software systems** such as **workflow** in the **legal system**, the structure and behaviour of a patient **healthcare system** and design of hardware.

# Three ways to apply UML

- There are **3 ways** in which people apply UML:
  - 1. UML as sketch:** to explore difficult parts of problem or solution space
  - 2. UML as blueprint:** detailed design diagrams are used either
    - for reverse engineering or
    - for forward engineering.
  - 3. UML as programming Language:** executable code is automatically generated, but is not normally seen or modified by developers; one works only in the UML “programming language”.

# Three perspectives to apply UML

1. **Conceptual perspective:** the diagrams are interpreted as describing **things in a situation** of the real world or domain of interest.
2. **Specification (software) perspective:** the diagrams describe **software abstraction or components with specifications and interfaces**, but no commitment to a particular implementation.
3. **Implementation (software) perspective:** the diagrams describe software implementations in a particular technology (such as java).

# Role of UML in OO Design

- UML is a modelling language used to model software and non-software systems.
- The relation between OO Design and UML is very important, since the **OO Design is transformed into UML diagrams** according to the requirement.
- Once OO Analysis and Design is done then the next step is very easy. Since the input from the OOAD is the input to the UML diagrams

# UML Modeling Constructs/Diagrams

## Static vs. Dynamic Perspectives

- A Diagram is a View to a Model
  - Presented from the aspect of a particular Stakeholder
  - Provides a partial representation of the System
  - Is Semantically Consistent with other views
- In the UML, There Are **Nine Standard Diagrams**
  - **Static Views:** Use Case, Class, Object, Component, Deployment
  - **Dynamic Views:** Sequence, Collaboration, State chart, Activity



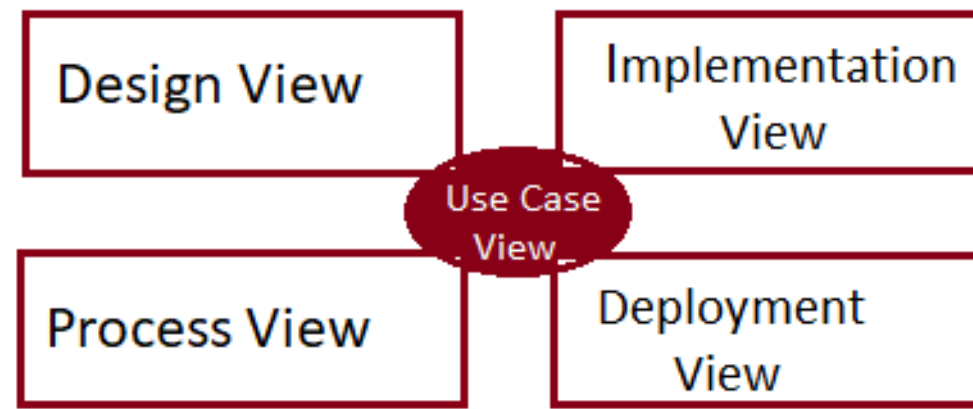
# UML Modeling Constructs/Diagrams Classification

- **Use-Case Diagrams**
- **Class and Object Diagrams**
- **Behavior Diagrams**
  - State chart Diagrams
  - **Activity Diagrams**
- **Interaction Diagrams**
  - **Sequence Diagram**
  - Collaboration Diagram
- **Implementation Diagrams**
  - Component Diagram
  - Deployment Diagram

# UML Architecture

- Any real world system is used by different users – **developers, testers, business people, analysts and many more.**
- So before designing a system the **architecture is made with different perspectives** in mind.
- UML plays an important role in defining different perspectives of a system. These **perspectives** are:
  - Design
  - Implementation
  - Process
  - Deployment

And the centre is the Use Case view which connects all these four.



**1. Use Case View:** encompasses the use cases that **describe the behavior of the system** as seen by its **end users, analysts** and **testers**.

- This view exists to specify the forces that shape the system's architecture.
- With UML, the **static aspects** of this view are captured in **use case diagrams**; the **dynamic aspects** of view are captured in **interaction diagrams, statechart diagrams, and activity diagrams**.

**2. Design view:** encompasses the classes, interfaces and collaborations that form the vocabulary of the problem and its solution.

- This view primarily supports the functional requirements of the system, meaning the services that the system should provide to its end users.
- With UML, **static aspects** of view are captured in **class diagram** and **object diagram**, the **dynamic aspects** are captured in **interaction diagrams, statechart diagrams, and activity diagrams**.

- **Process View:** encompasses the **threads** and **processes** that form the system's concurrency and synchronization mechanisms.
- This view primarily addresses the performance, scalability, and throughput of the system.
- With UML, the **static** and **dynamic aspects** of this view are captured in the **same** kinds of diagrams as for the **design view**, but **with a focus on the active classes** that represent these threads and processes.
- **Implementation View:** encompasses the **components** and **files** that are used to assemble and release the physical system.
- This view primarily addresses the configuration management of the system's releases, made up of independent components and files that can be assembled in various ways to produce a running system.
- With UML, the static aspects of this view are captured in component diagrams; the dynamic aspects are captured in interaction diagrams, statechart diagrams and activity diagrams.

- **Deployment View:** encompasses the **nodes** that form the system's **hardware topology** on which the system executes.
- This view primarily addresses the distribution, delivery and installation of the parts that make up the physical system.
- With UML, the **static aspects** of this view are captured in **deployment diagrams**, the **dynamic aspects** of this view are captured in **interaction diagrams, statechart diagrams and activity diagrams**.

# A conceptual model of UML

- A conceptual model can be defined as a **model which is made of concepts and their relationships.**
- This model is the **first step** before drawing a UML diagram. It helps to understand the **entities in the real world** and how they **interact** with each other.
- Conceptual model of UML can be mastered by learning the following major elements:
  - UML building blocks
  - Rules to connect the building blocks
  - Common mechanisms of UML

# UML Building Blocks

The building blocks of UML can be defined as:

- **Things (entities)**
- **Relationships**
- **Diagrams**

# Things (**Entites**)

- **Entities are the elements of models.**
- All UML entities can be divided into:
  - **Structural entities** - **nouns** UML models, such as class, interface, cooperation, precedent, the active class, component, unit.
  - **Behavioral entities** - **verbs** UML models, such as interaction, activity, machines; grouping the essence of the package that is used for grouping semantically related elements of the model in forming a single whole modules;
  - **Summary entity** - **note**, which can be added to the model to record specific information, very much like the **sticker**.

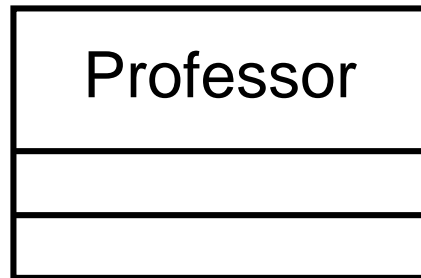


# Structural entities

- Structural entities are the **nouns** of UML models. These are the static parts of a model, representing elements that are either conceptual or physical.
  - Class
  - Interface
  - Collaboration
  - Use case
- **Class:** a class is a set of objects that share the same attributes, operations, relationships and semantics.

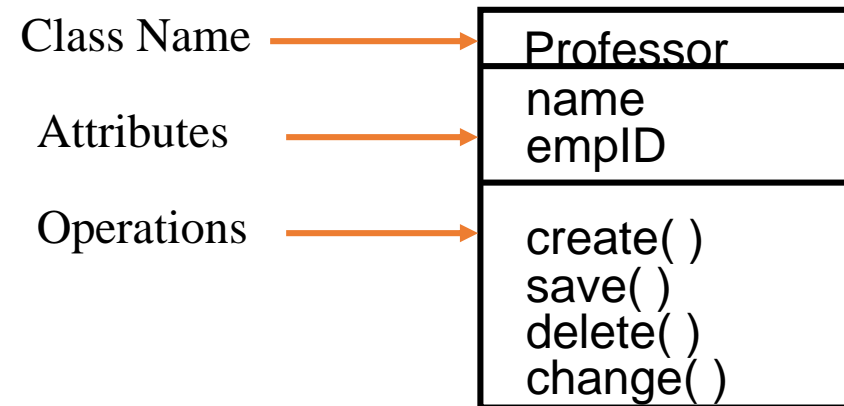
# Representing Classes

- A class is represented using a compartmented rectangle

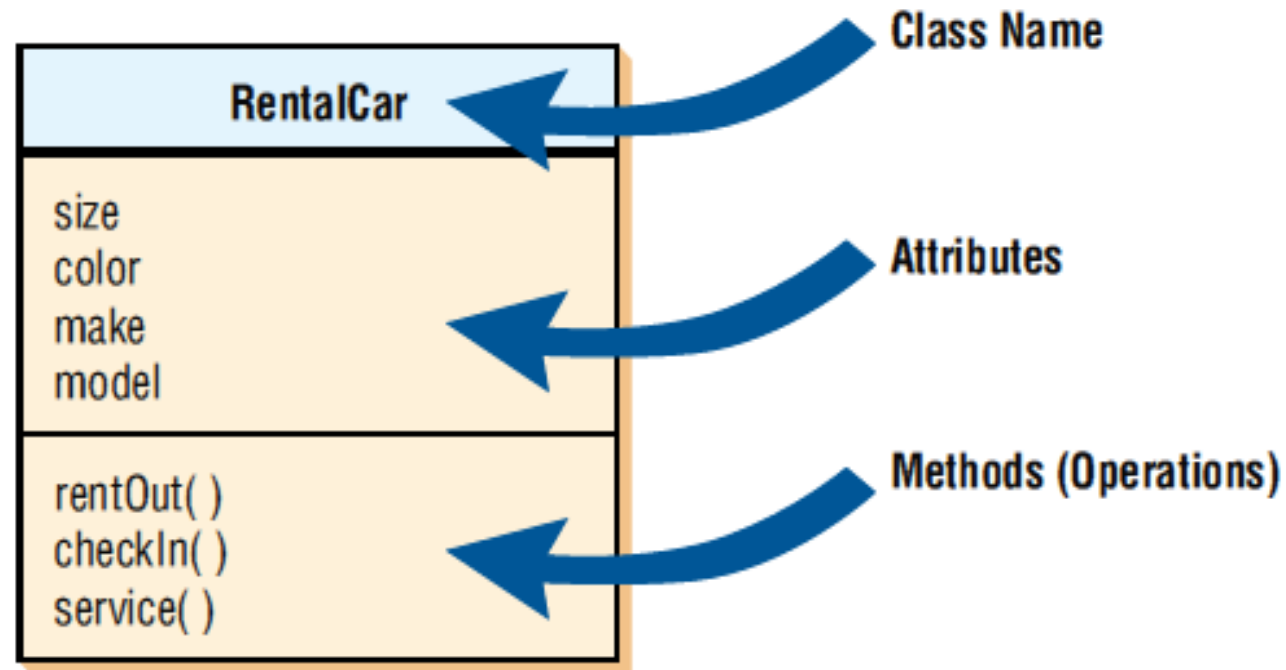


# Class Compartments

- A class is comprised of three sections
  - The **first section** contains the **class name**
  - The **second section** shows the structure (**attributes**)
  - The **third section** shows the behavior (**operations**)



# Example1: for Class diagram

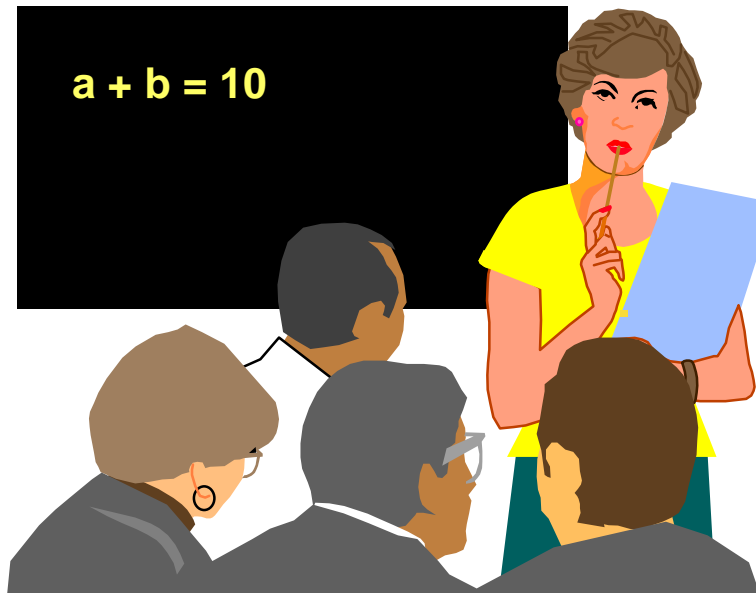


# Example 2:

## Class Course

### Properties

Name  
Location  
Days offered  
Credit hours  
Start time  
End time



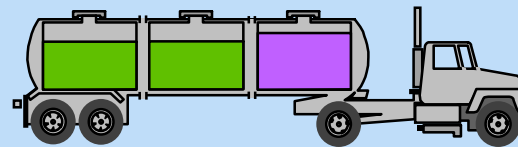
### Behavior

Add a student  
Delete a student  
Get course roster  
Determine if it is full

# What is an Object?

- Informally, an object represents an entity, either physical, conceptual, or software

- Physical entity



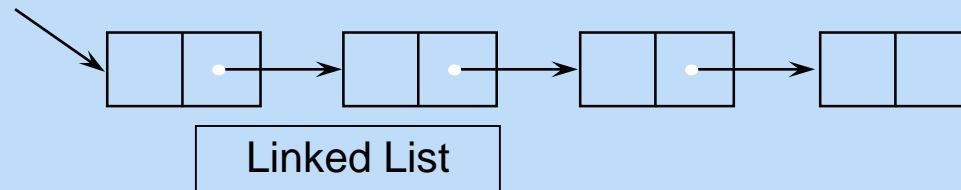
Truck

- Conceptual entity



Chemical  
Process

- Software entity



# Representing Objects

- An object is represented as rectangles with underlined names

: Professor

**Class Name Only**

ProfessorClark :  
Professor

**Class and Object Name**

ProfessorClark

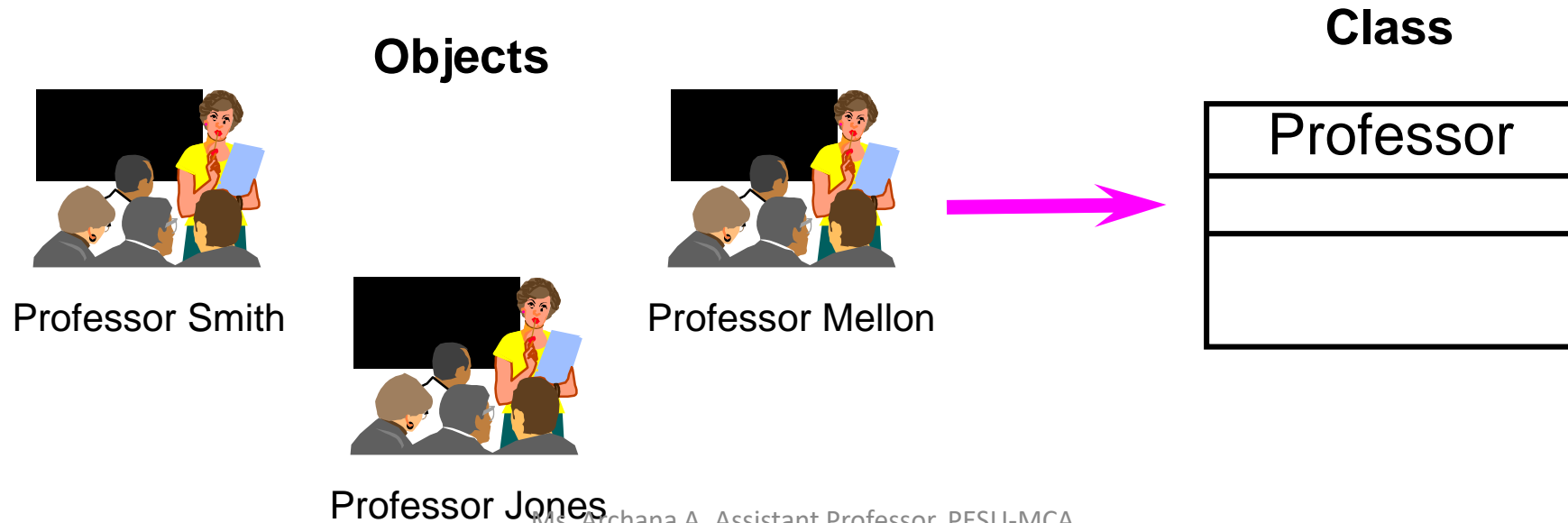
**Object Name Only**



*(stay tuned for classes)*

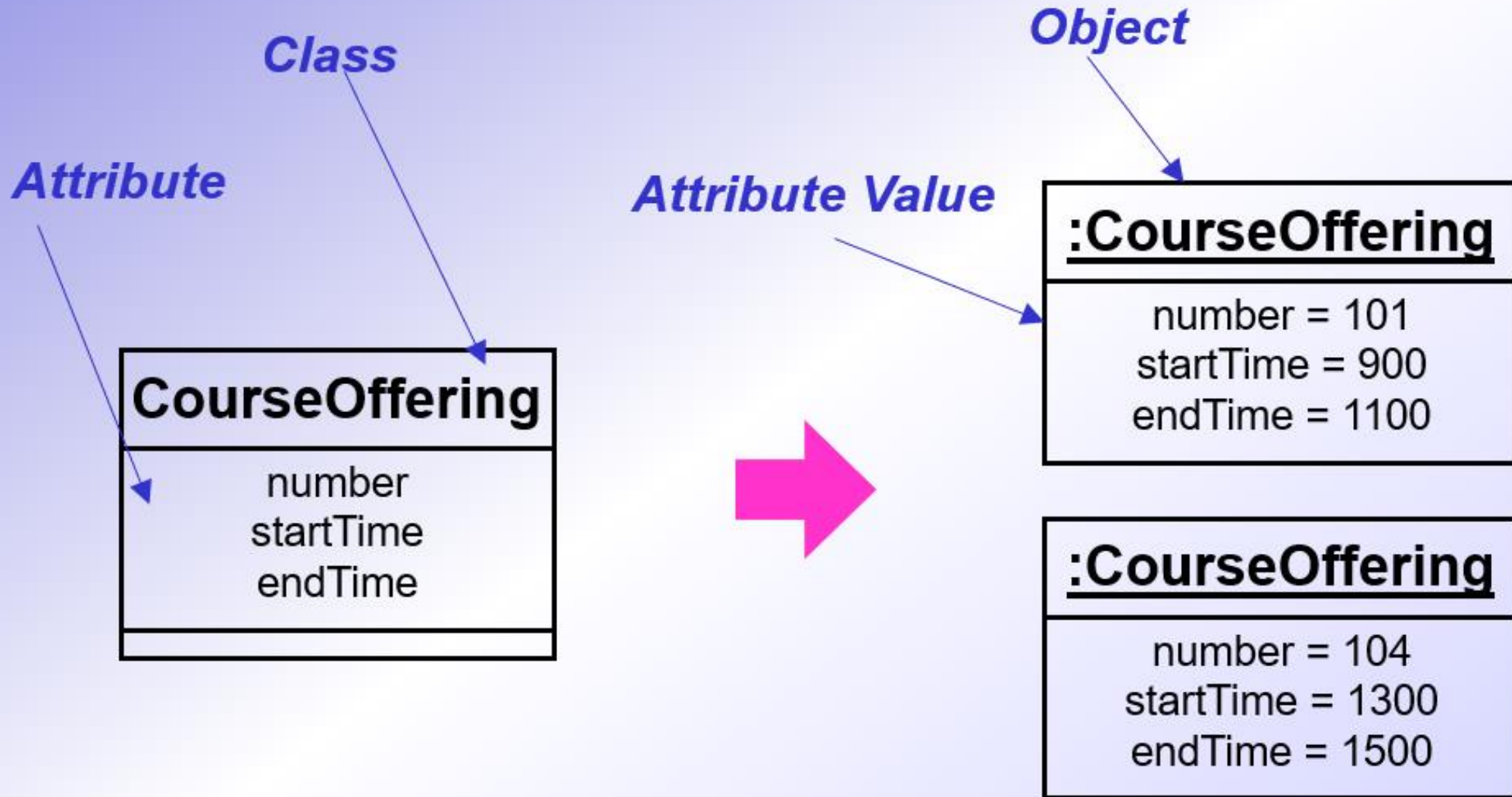
# The Relationship Between Classes and Objects

- A class is an abstract definition of an object
  - It defines the structure and behavior of each object in the class
  - It serves as a template for creating objects
- Objects are grouped into classes

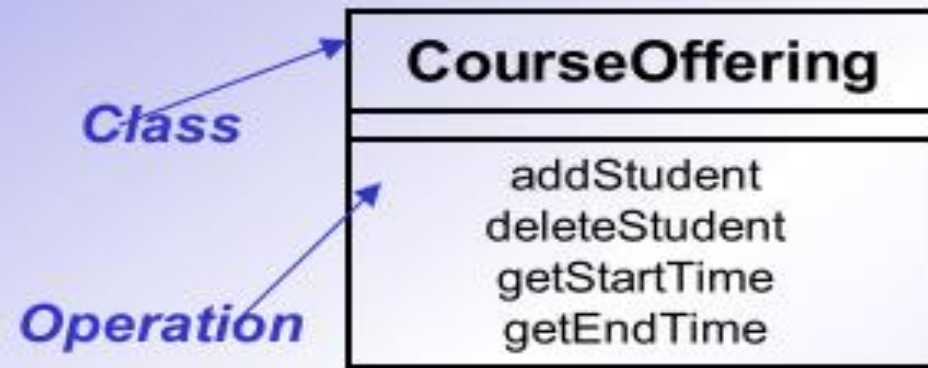




# What is an Attribute?

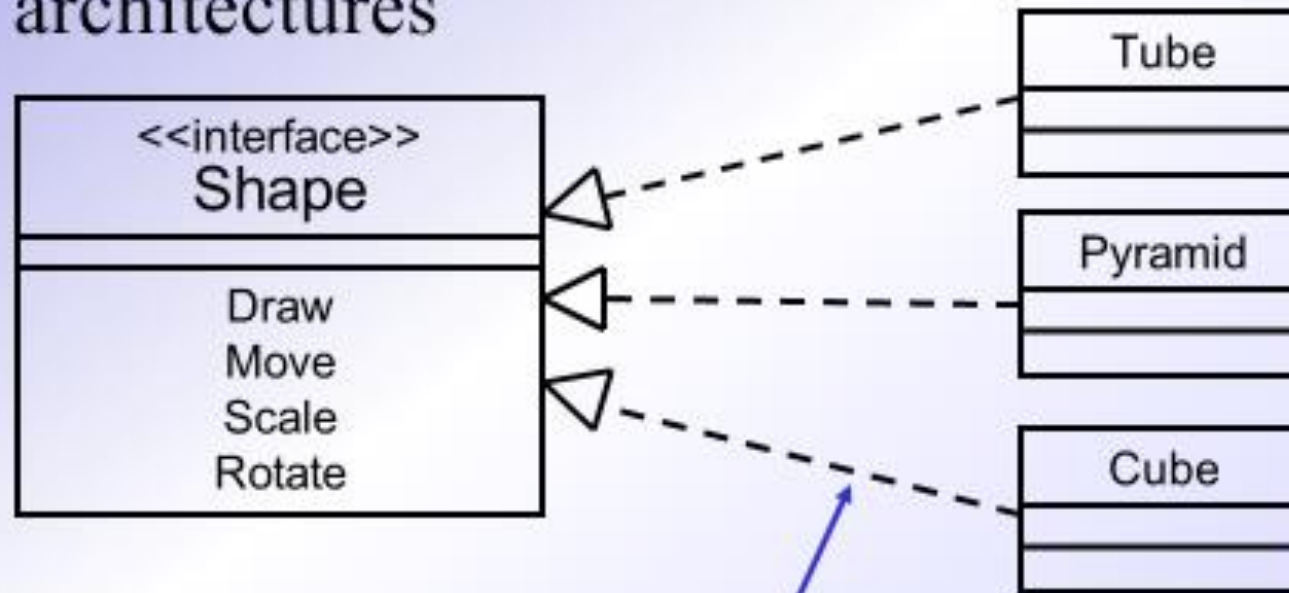


# What is an Operation?



# What is an Interface?

- Interfaces formalize polymorphism
- Interfaces support “plug-and-play” architectures

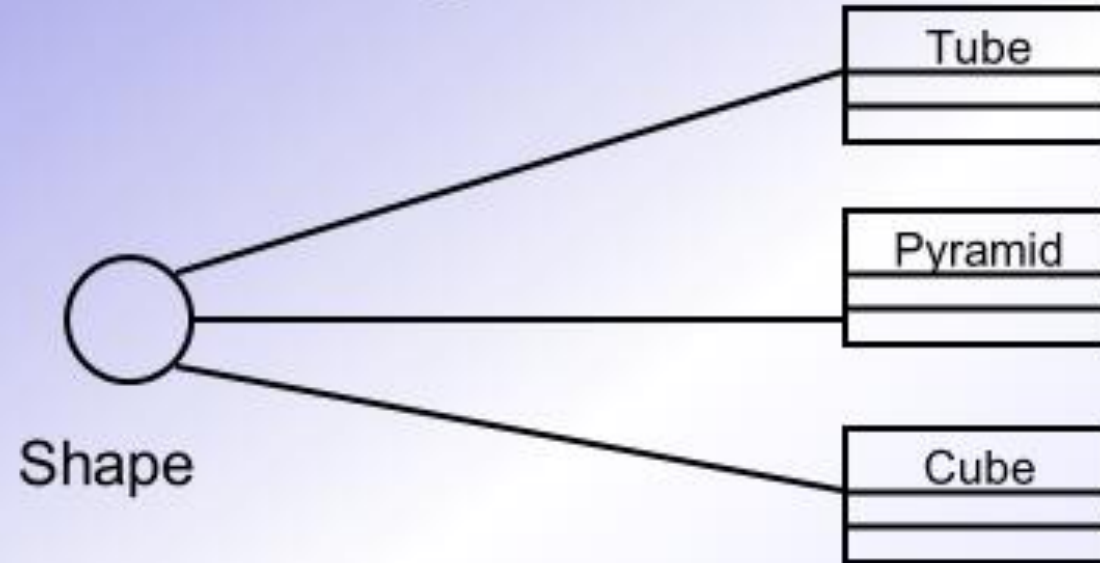


Realization relationship

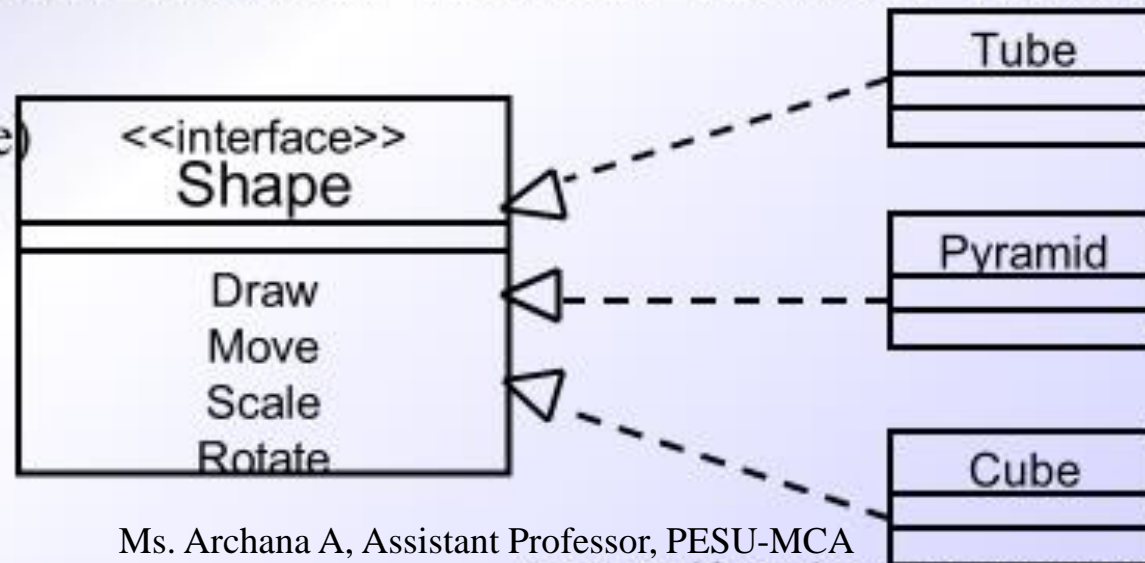
*(stay tuned for realization relationships)*

# Interface Representations

Elided/Iconic  
Representation  
("lollipop")



Canonical  
(Class/Stereotype)  
Representation

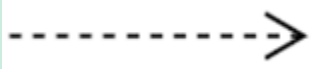





Ms. Archana A, Assistant Professor, PESU-MCA




*(stay tuned for realization relationships)*

# Relationships in UML

- Dependency
- Association
- Generalization
- Realization

Type of relationship	UML syntax Source – Target	Semantics
<b>Dependency</b>		Source element depends on the target element and change last may affect the first.
<b>Association</b>		Description of the set of relationships between objects.
<b>Aggregation</b>		The target element is a part of the source element.
<b>Composition</b>		Strict (more limited) form of aggregation/



Type of relationship	UML syntax Source – Target	Semantics
<b>Inclusion</b>		Source element contains the target element.
<b>Generalization</b>		The source element is a specialization of a more generalized the target element, and can replace him.
<b>Realization</b>		The original item is guaranteed to perform the contract, a certain target element.

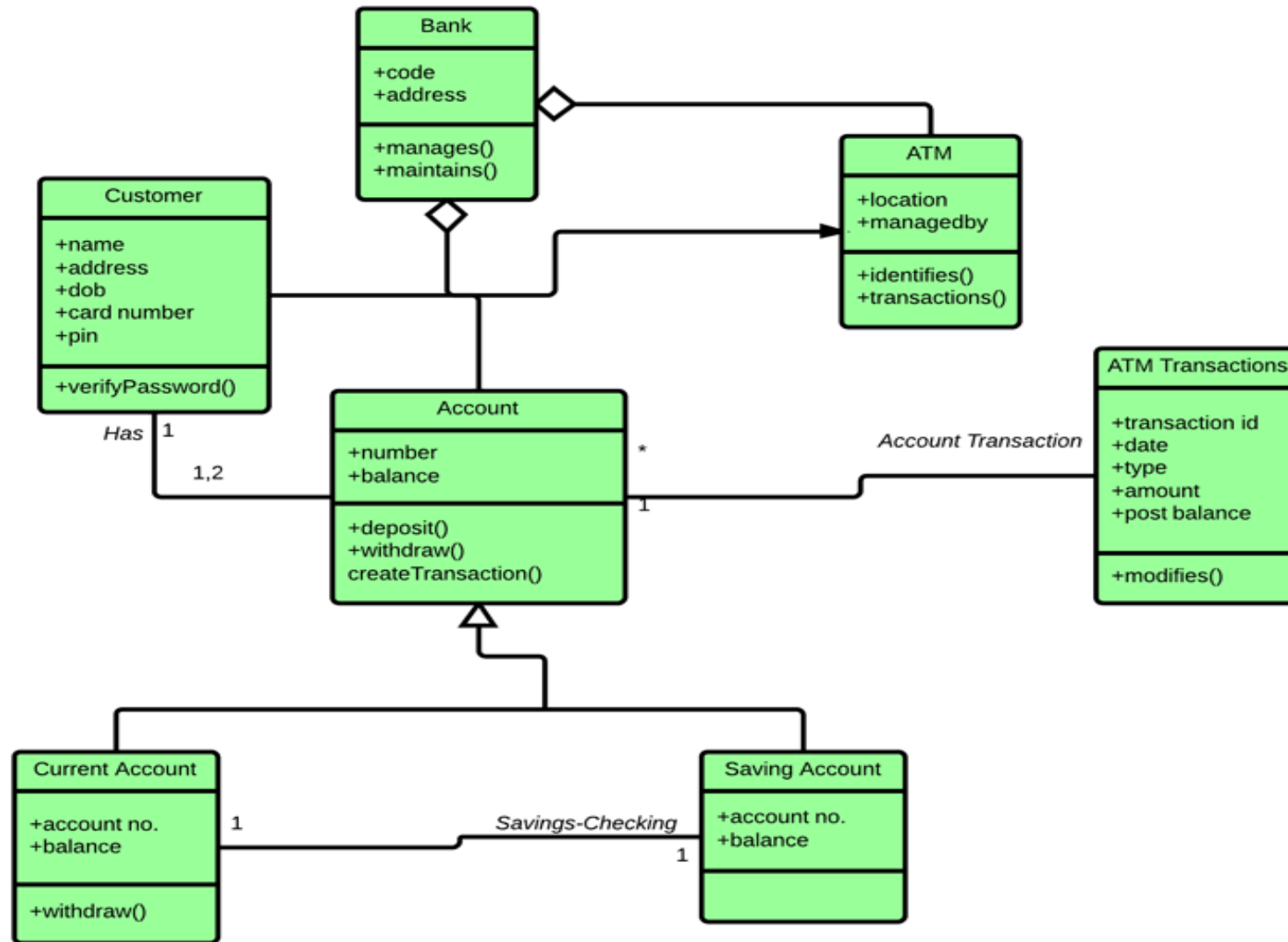
# Categories of UML diagrams



# 1. Class Diagram

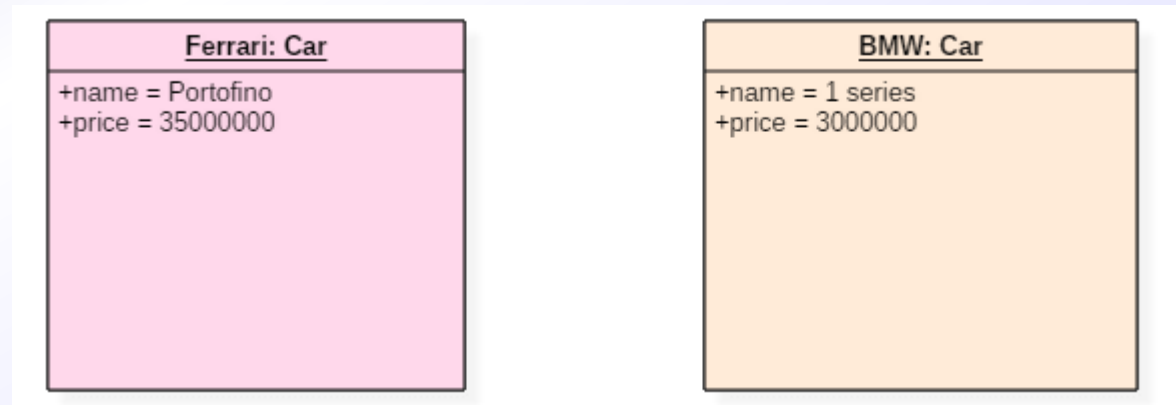
- Class Diagram gives the static view of an application. A class diagram describes the types of objects in the system and the different types of relationships that exist among them. This modeling method can run with almost all Object-Oriented Methods. A class can refer to another class. A class can have its objects or may inherit from other classes.
- UML Class Diagram gives an overview of a software system by displaying classes, attributes, operations, and their relationships. This Diagram includes the class name, attributes, and operation in separate designated compartments.
- Class Diagram helps construct the code for the software application development.
- Essential elements of UML class diagram are:
  - Class Name
  - Attributes
  - Operations

# Class diagram example



## 2. Object Diagram:

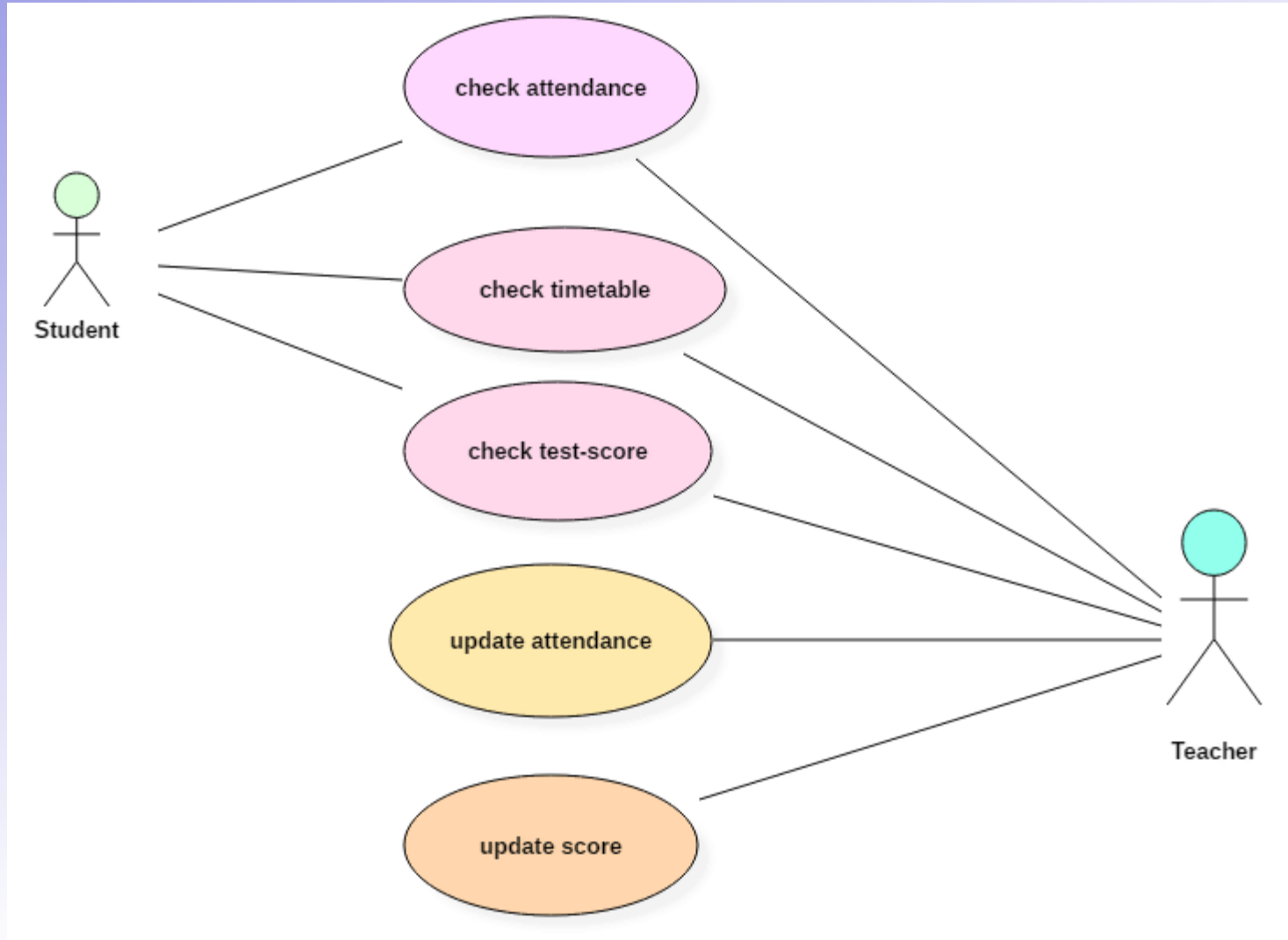
- Objects are the real-world entities whose behavior is defined by the classes. Objects are used to represent the static view of an object-oriented system. We cannot define an object without its class. Object and class diagrams are somewhat similar.
- The difference between the class and object diagram is that the class diagram mainly represents the bird's eye view of a system which is also referred to as an abstract view. An object diagram describes the instance of a class. It visualizes the particular functionality of a system.
- Object diagram example:



# 3. Use Case Diagram

- Use Case diagrams are defined as diagrams that capture the system's functionality and requirements in UML. Use-cases are the core concepts of Unified Modelling language modeling.
- A use case is a unique functionality of a system which is accomplished by a user. A purpose of use case diagram is to capture core functionalities of a system and visualize the interactions of various things called as actors with the use case. This is the general use of a use case diagram.
- The use case diagrams represent the core parts of a system and the workflow between them. In use case, implementation details are hidden from the external use only the event flow is represented.
- With the help of use case diagrams, we can find out pre and post conditions after the interaction with the actor. These conditions can be determined using various test cases.

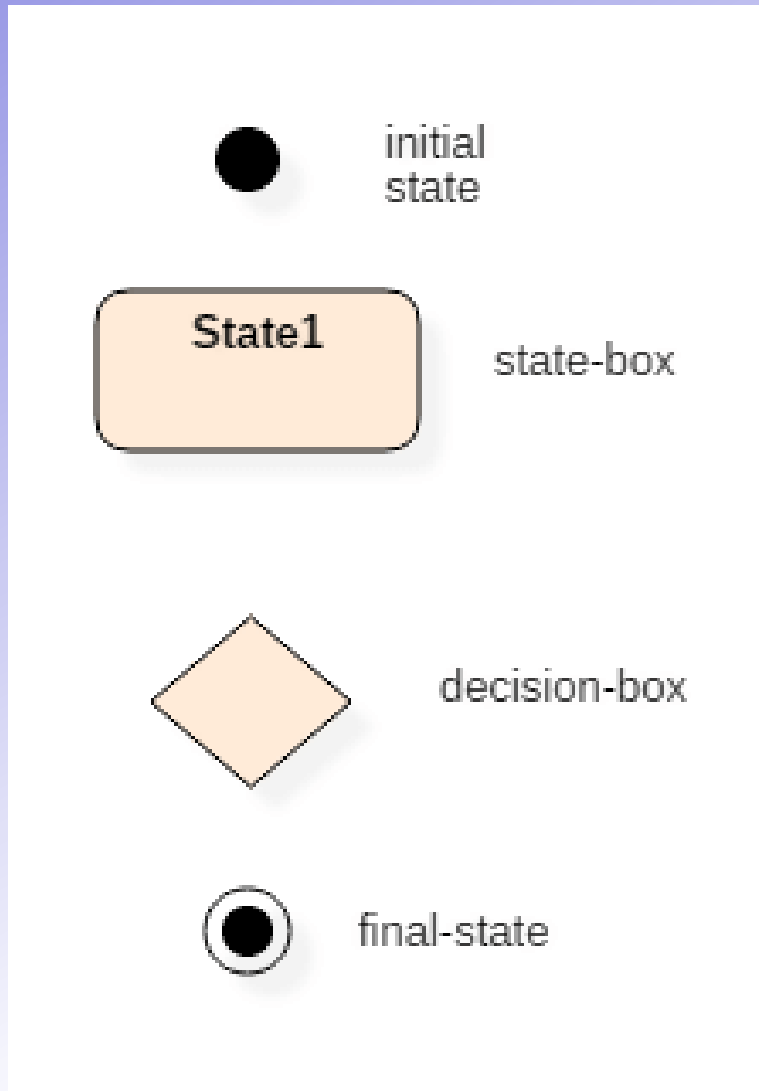
# Use Case diagram example



# 4. State Chart diagram

- State chart diagrams are used to capture the behavior of a software system. UML State machine diagrams can be used to model the behavior of a class, a subsystem, a package, or even an entire system. It is also called a State machine or State Transition diagram.
- Statechart diagrams provide us an efficient way to model the interactions or communication that occur within the external entities and a system. These diagrams are used to model the event-based system. A state of an object is controlled with the help of an event.
- Statechart diagrams are used to describe various states of an entity within the application system.

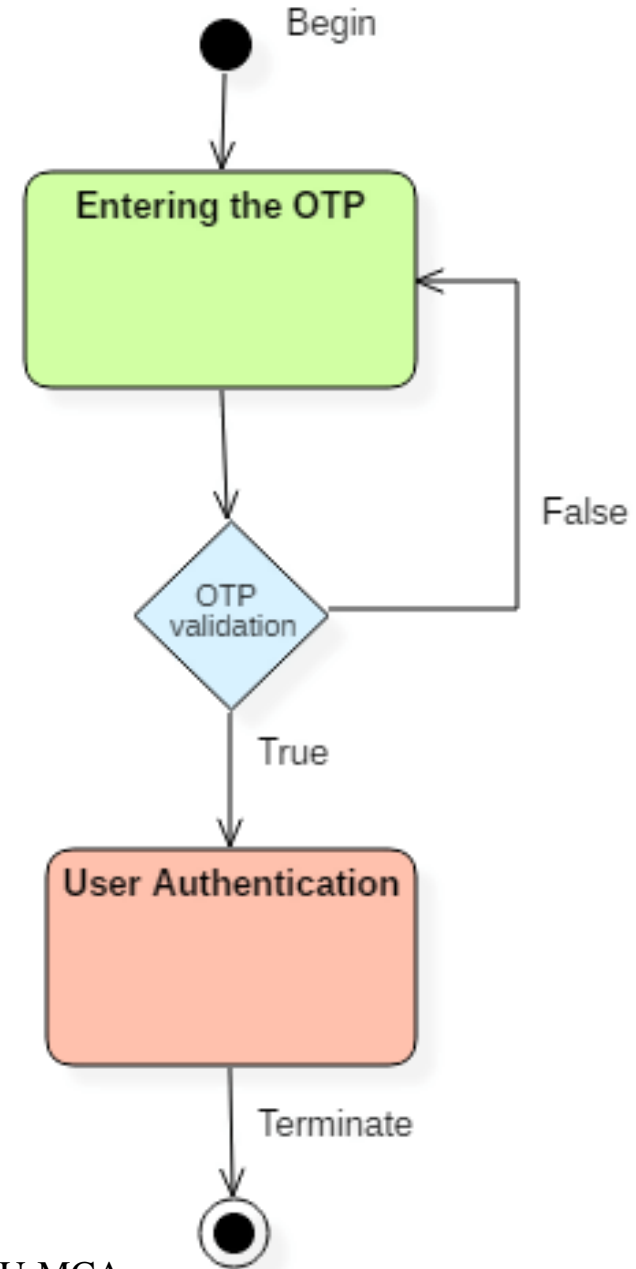
# Notation and Symbol for State Machine





# example

- There are a total of two states, and the first state indicates that the OTP has to be entered first. After that, OTP is checked in the decision box, if it is correct, then only state transition will occur, and the user will be validated. If OTP is incorrect, then the transition will not take place, and it will again go back to the beginning state until the user enters the correct OTP.





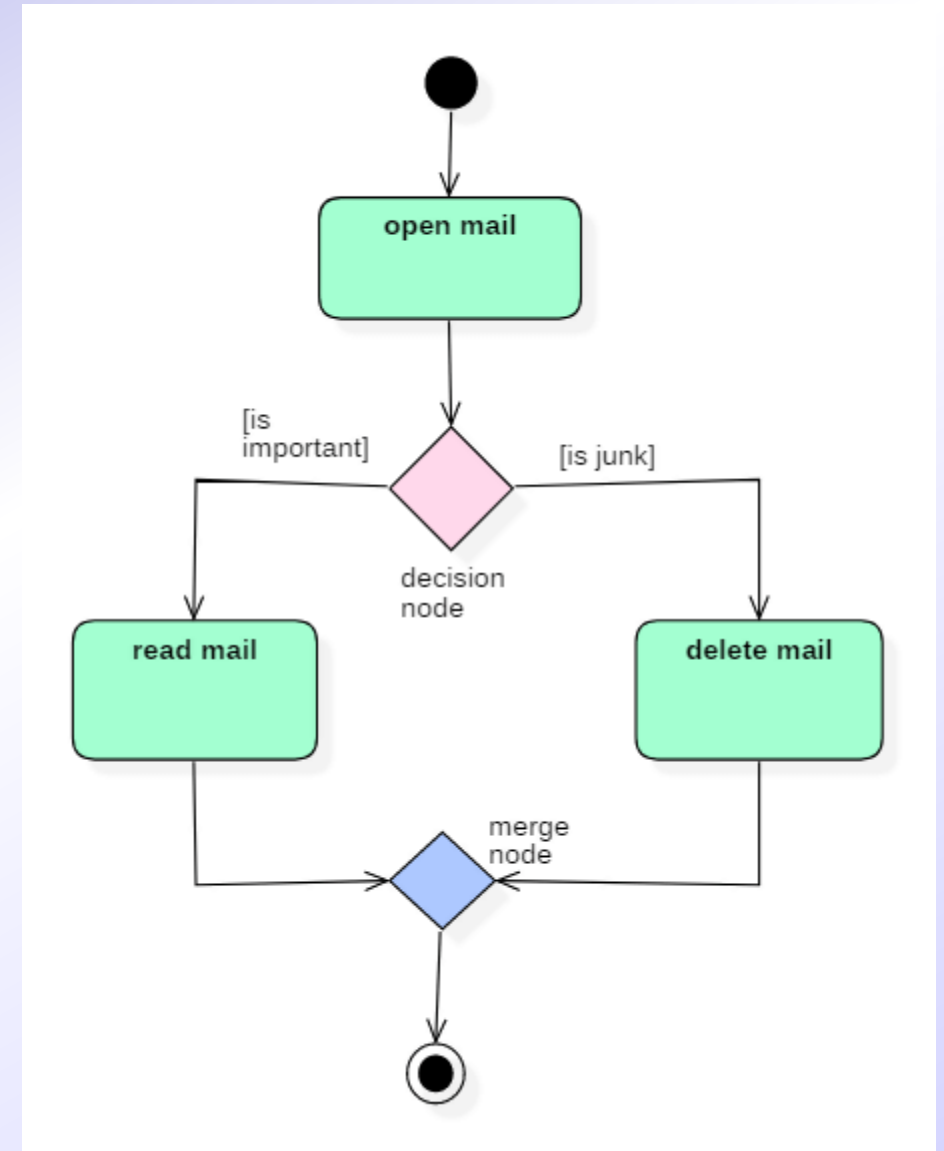
# 5. Activity Diagram

- Activity diagram is defined as a UML diagram that focuses on the execution and flow of the behavior of a system instead of implementation. It is also called **object-oriented flowchart**. Activity diagrams consist of activities that are made up of actions which apply to behavioral modeling technology.
- Activity diagram is used to model business processes and workflows. These diagrams are used in software modeling as well as business modeling.

- Most commonly activity diagrams are used to,
  1. Model the workflow in a graphical way, which is easily understandable.
  2. Model the execution flow between various entities of a system.
  3. Model the detailed information about any function or an algorithm which is used inside the system.
  4. Model business processes and their workflows.
  5. Capture the dynamic behavior of a system.
  6. Generate high-level flowcharts to represent the workflow of any application.
  7. Model high-level view of an object-oriented or a distributed system.

# example

- In the above activity diagram, three activities are specified. When the mail checking process begins user checks if mail is important or junk. Two guard conditions [is essential] and [is junk] decides the flow of execution of a process. After performing the activity, finally, the process is terminated at termination node.



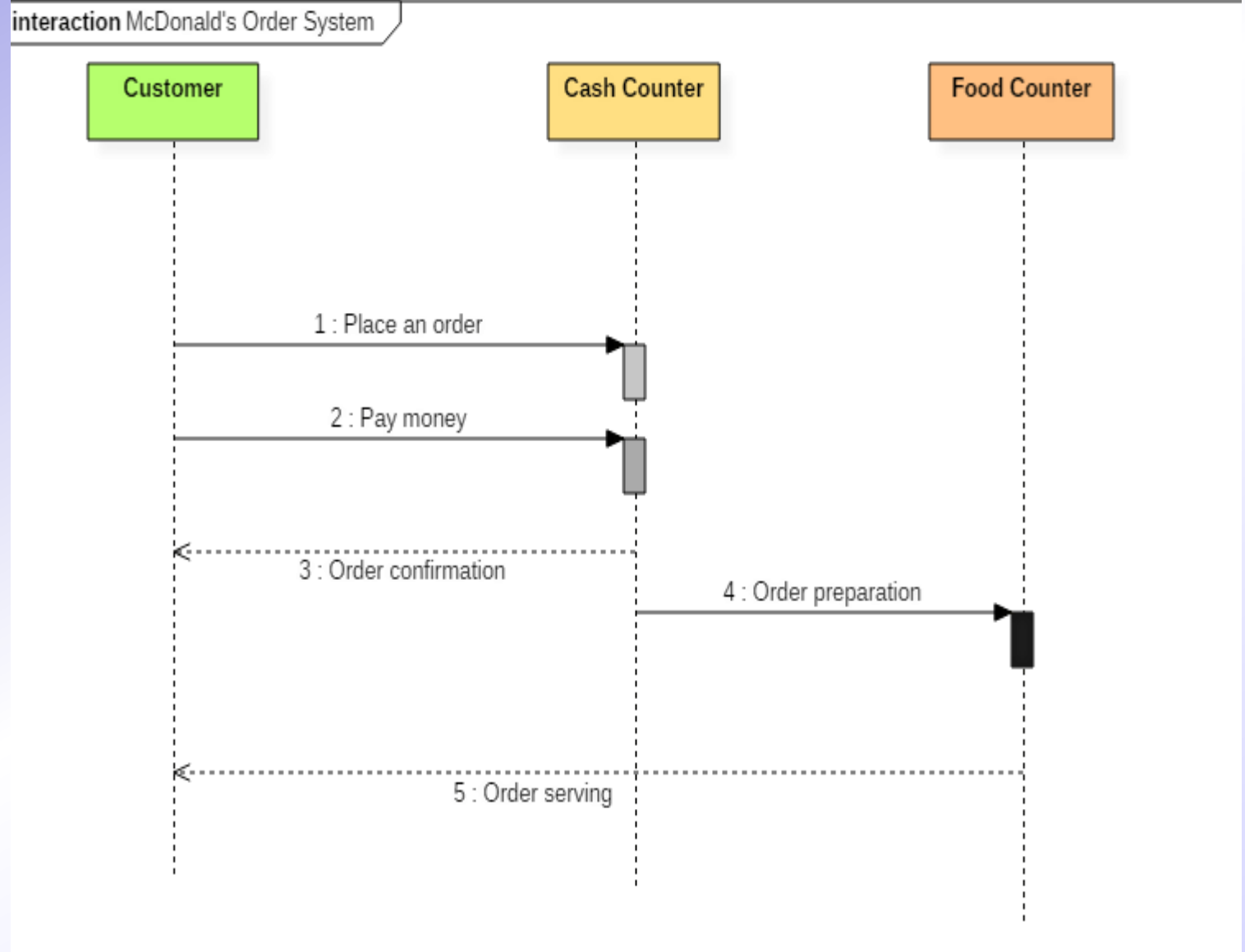
# 6. Sequence Diagram

- The purpose of a sequence diagram in UML is to visualize the sequence of a message flow in the system.
- The sequence diagram shows the interaction between two lifelines as a time-ordered sequence of events.
- A sequence diagram is used to capture the behavior of any scenario.
- A sequence diagram shows an implementation of a scenario in the system. Lifelines in the system take part during the execution of a system.
- In a sequence diagram, a lifeline is represented by a vertical bar.
- A message flow between two or more objects is represented using a vertical dotted line which extends across the bottom of the page.

# example

- sequence diagram example representing McDonald's ordering system.
- The ordered sequence of events in a given sequence diagram is as follows:

1. Place an order.
2. Pay money to the cash counter.
3. Order Confirmation.
4. Order preparation.
5. Order serving.



# 7. Collaboration Diagram

- A collaboration diagram is required to identify how various objects make up the entire system.
- They are used to understand the object architecture within a system rather than the flow of a message in a sequence diagram.
- An object an entity that has various attributes associated with it. It is a concept of object-oriented programming.
- There are multiple objects present inside an object-oriented system where each object can be associated with any other object inside the system.
- Collaboration or communication diagrams are used to explore the architecture of objects inside the system. The message flow between the objects can be represented using a collaboration diagram.

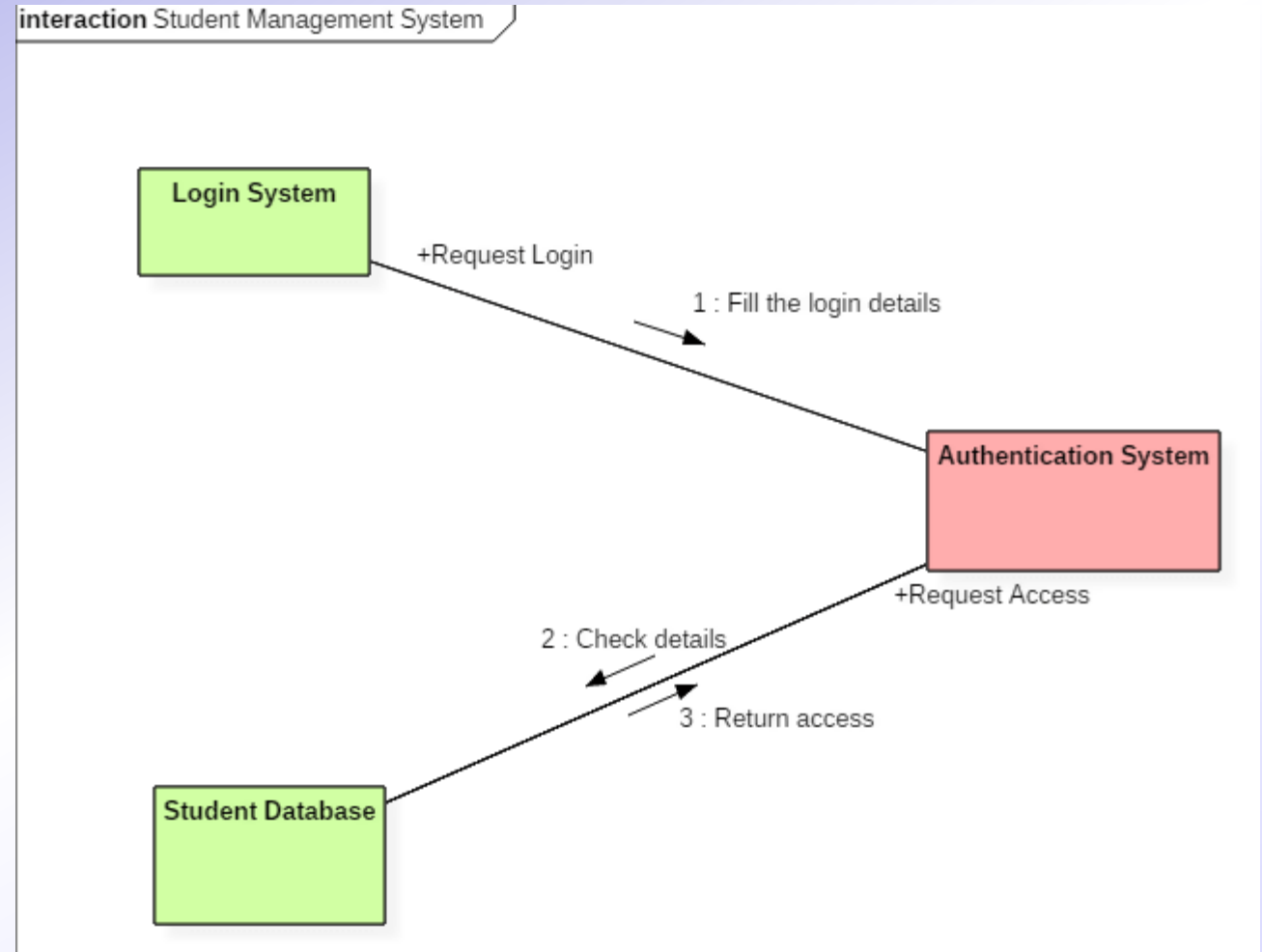


# Benefits of collaboration diagram

- It is also called as a communication diagram.
- It emphasizes the structural aspects of an interaction diagram - how lifeline connects.
- Its syntax is similar to that of sequence diagram except that lifeline don't have tails.
- Messages passed over sequencing is indicated by numbering each message hierarchically.
- Compared to the sequence diagram communication diagram is semantically weak.
- Object diagrams are special case of communication diagram.
- It allows you to focus on the elements rather than focusing on the message flow as described in the sequence diagram.
- Sequence diagrams can be easily converted into a collaboration diagram as collaboration diagrams are not very expressive.

# Example:

- collaboration diagram represents a student information management system. The flow of communication in the above diagram is given by,
1. A student requests a login through the login system.
  2. An authentication mechanism of software checks the request.
  3. If a student entry exists in the database, then the access is allowed; otherwise, an error is returned.



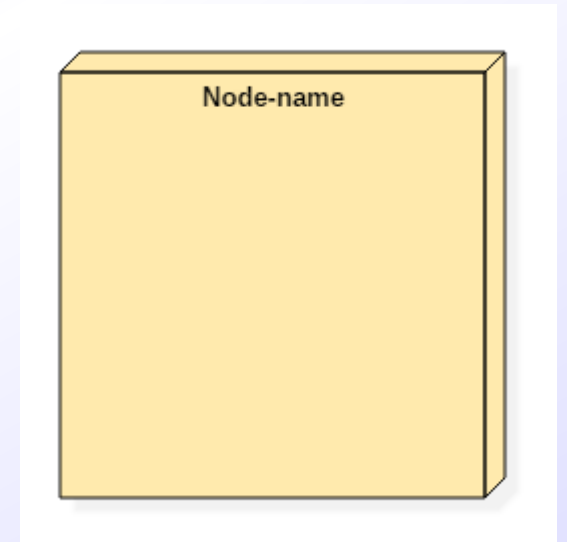
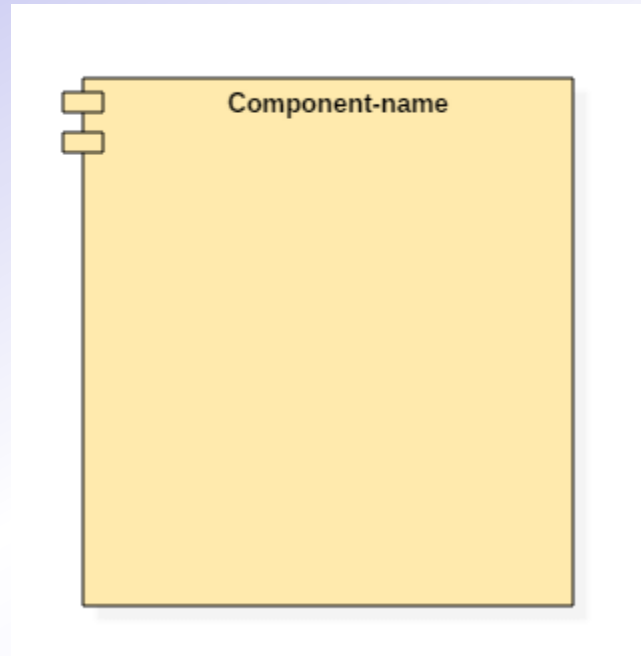


# 8. Component diagram

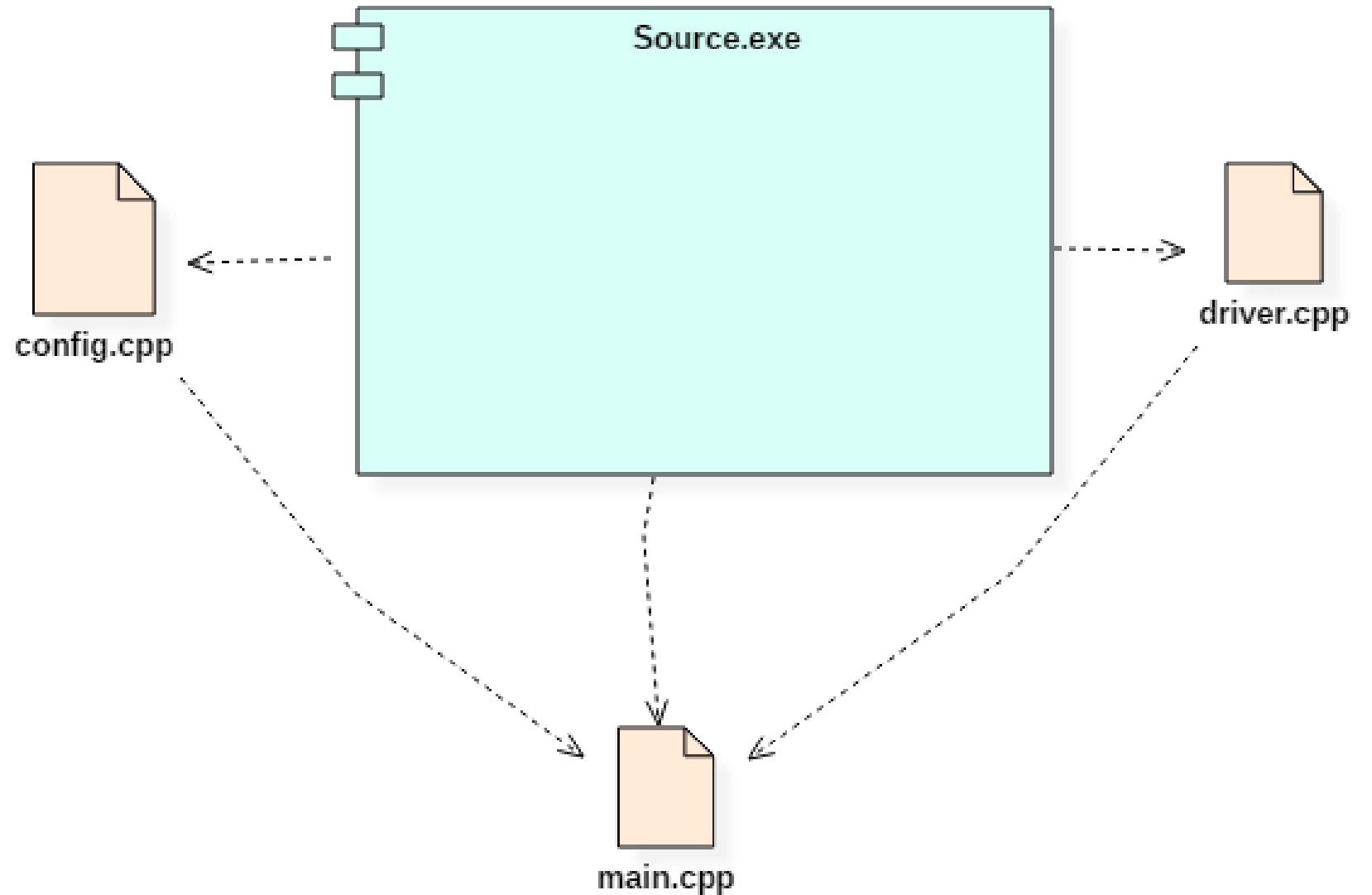
- When modeling large object-oriented systems, it is necessary to break down the system into manageable subsystems. UML component diagrams are used for modeling large systems into smaller subsystems which can be easily managed.
- A component is a replaceable and executable piece of a system whose implementation details are hidden. A component provides the set of interfaces that a component realizes or implements. Components also require interfaces to carry out a function.

- Component diagrams are different from any other diagrams in UML. Component diagrams are used to display various components of a software system as well as subsystems of a single system. They are used to represent physical things or components of a system. It generally visualizes the structure and an organization of a system.
- It describes how various components together make a single, fully functional system. We can display each component individually or collectively as a single unit.
- Component diagrams are used to model the component organization of a system.
- They are used to divide a single system into various subsystems as per the functionality

# Notations used in component diagram



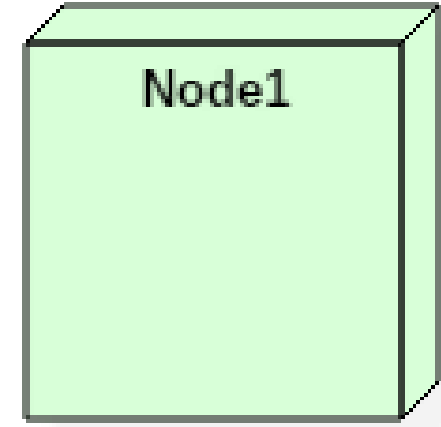
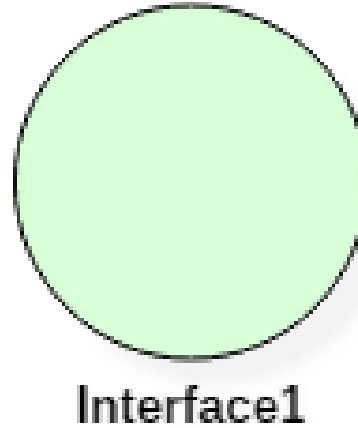
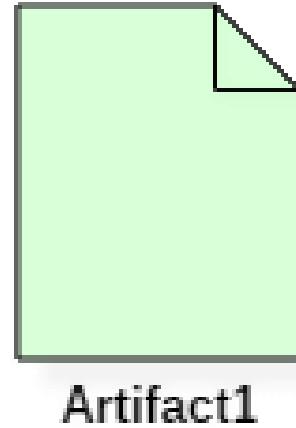
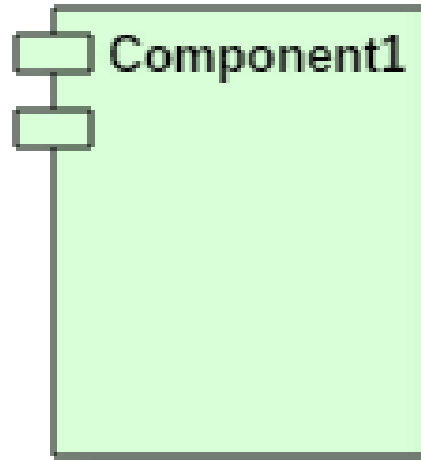
# example



# 9. Deployment diagram

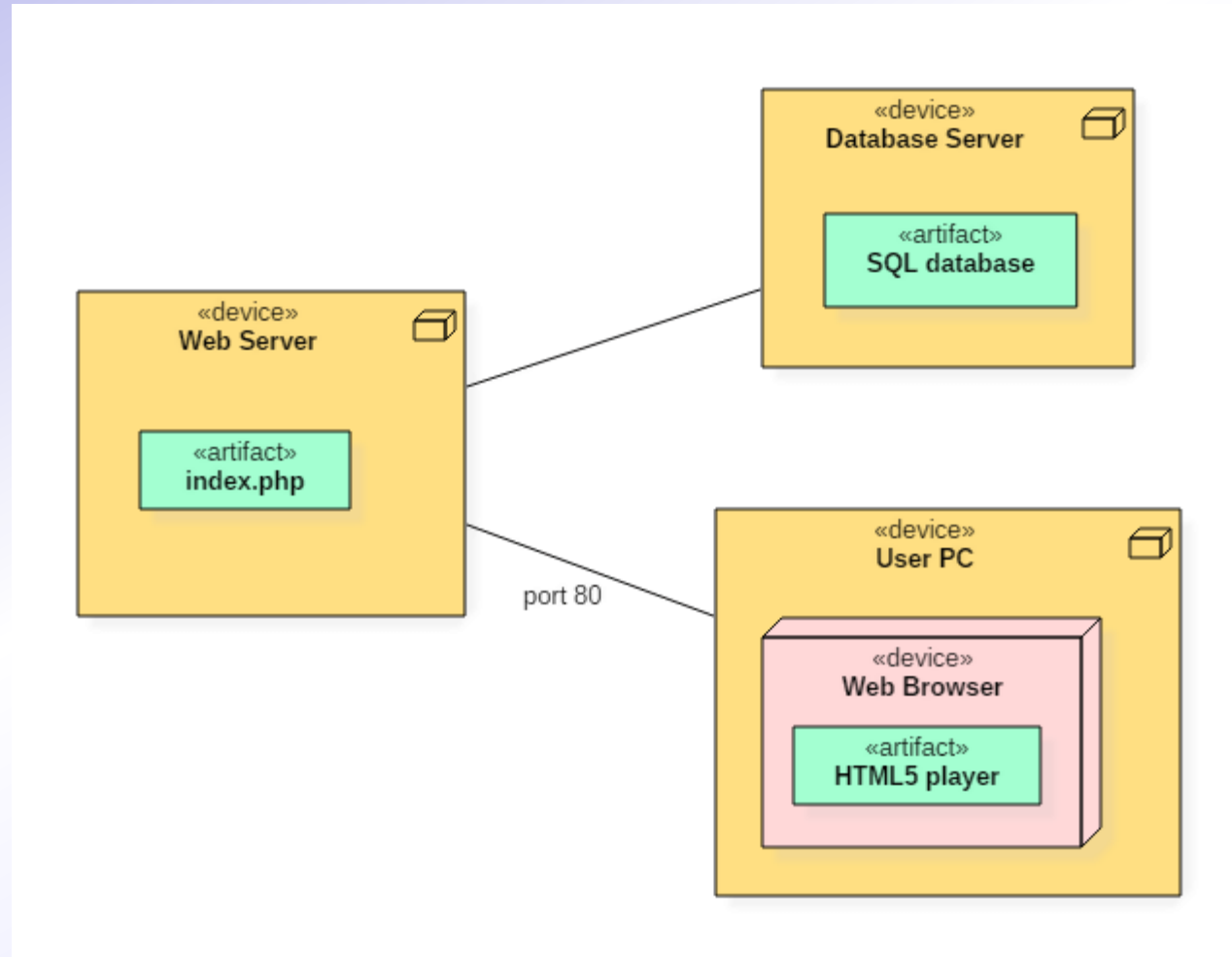
- Deployment Diagram is a type of diagram that specifies the physical hardware on which the software system will execute. It also determines how the software is deployed on the underlying hardware. It maps software pieces of a system to the device that are going to execute it.
- The deployment diagram maps the software architecture created in design to the physical system architecture that executes it. In distributed systems, it models the distribution of the software across the physical nodes.
- The software systems are manifested using various **artifacts**, and then they are mapped to the execution environment that is going to execute the software such as **nodes**. Many nodes are involved in the deployment diagram; hence, the relation between them is represented using communication paths.

# Deployment Diagram Symbol and notations



# example

- Following deployment diagram represents the working of HTML5 video player in the browser:



Deployment diagrams can be used for,

- Modeling the network topology of a system.
- Modeling distributed systems and networks.
- Forward and reverse engineering processes.

### Summary:

- The deployment diagram maps the software architecture created in design to the physical system architecture that executes it.
- It maps software pieces of a system to the hardware that are going to execute it.
- Deployment diagram visualizes the topological view of an entire system.
- Nodes and artifacts are the essential elements of deployment.
- Node and artifacts of a system participate in the final execution of a system.



- **End of Unit-1**