**UML**

UML is composed of three main building blocks, i.e., things, relationships, and diagrams. Building blocks generate one complete UML model diagram by rotating around several different blocks. It plays an essential role in developing UML diagrams. The basic UML building blocks are enlisted below:

1. Things
2. Relationships
3. Diagrams

### Things

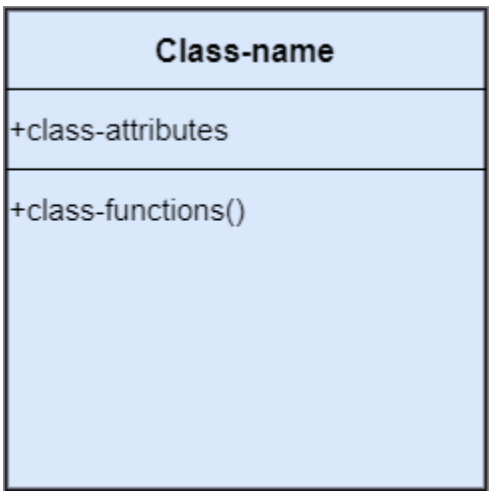
Anything that is a real world entity or object is termed as things. It can be divided into several different categories:

- Structural things
- Behavioral things
- Grouping things

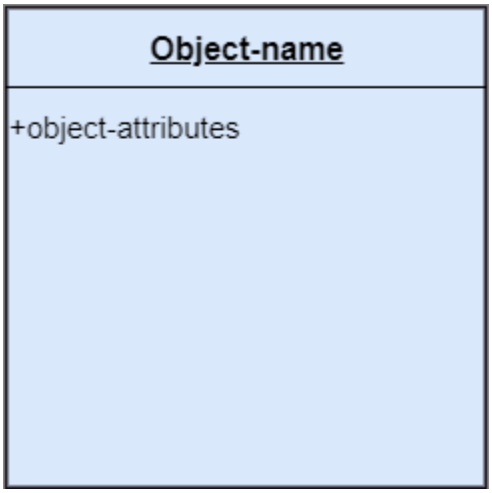
### Structural things

Nouns that depicts the static behavior of a model is termed as structural things. They display the physical and conceptual components. They include class, object, interface, node, collaboration, component, and a use case.

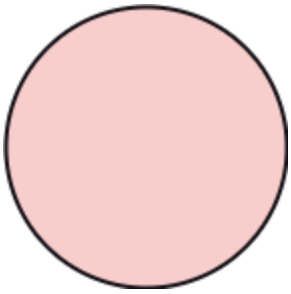
**Class:** A Class is a set of identical things that outlines the functionality and properties of an object. It also represents the abstract class whose functionalities are not defined. Its notation is as follows;




**Object:**: An individual that describes the behavior and the functions of a system. The notation of the object is similar to that of the class; the only difference is that the object name is always underlined and its notation is given below;

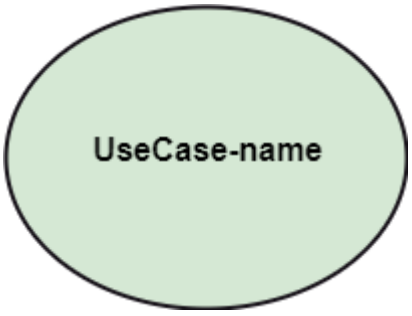



**Interface:** A set of operations that describes the functionality of a class, which is implemented whenever an interface is implemented.

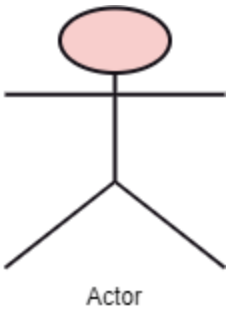


**Collaboration:** It represents the interaction between things that is done to meet the goal. It is symbolized as a dotted ellipse with its name written inside it.

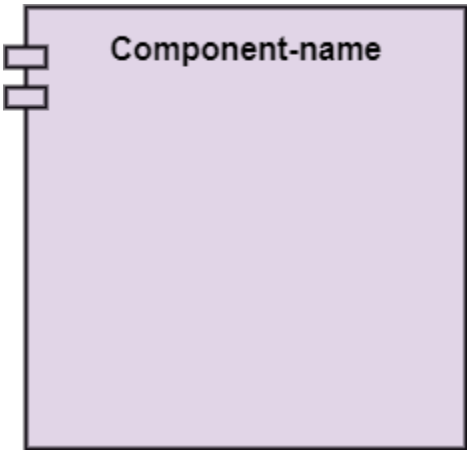

**Use case:** Use case is the core concept of object-oriented modeling. It portrays a set of actions executed by a system to achieve the goal.



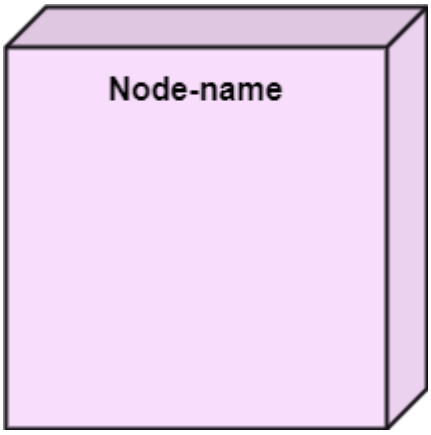

**Actor:** It comes under the use case diagrams. It is an object that interacts with the system, for example, a user.




**Component:** It represents the physical part of the system.



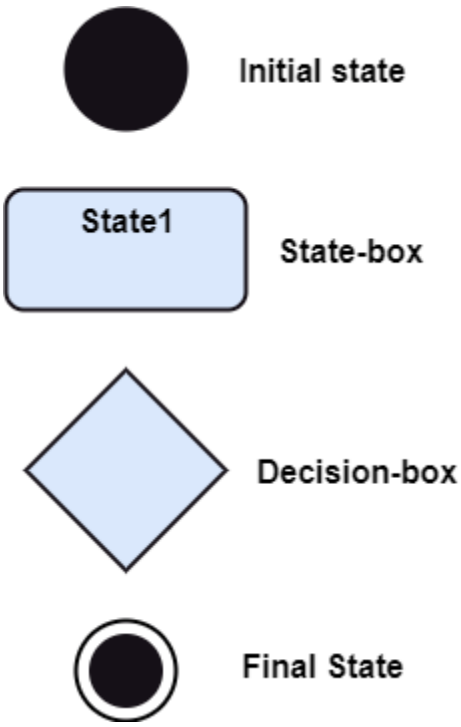

**Node:** A physical element that exists at run time.



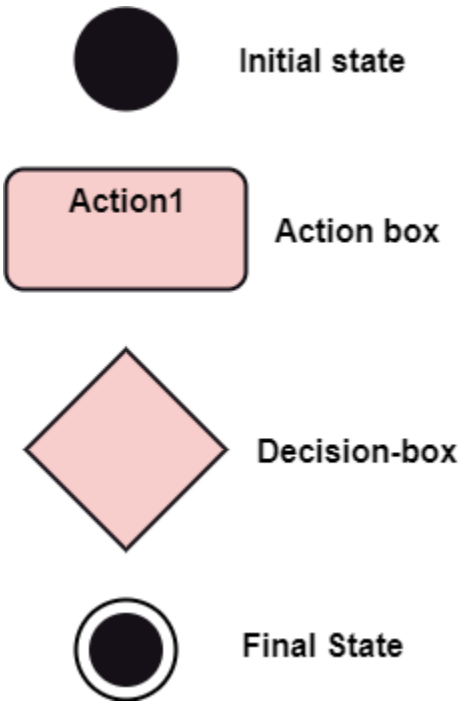

Behavioral Things

They are the verbs that encompass the dynamic parts of a model. It depicts the behavior of a system. They involve state machine, activity diagram, interaction diagram, grouping things, annotation things

**State Machine:** It defines a sequence of states that an entity goes through in the software development lifecycle. It keeps a record of several distinct states of a system component.




**Activity Diagram:** It portrays all the activities accomplished by different entities of a system. It is represented the same as that of a state machine diagram. It consists of an initial state, final state, a decision box, and an action notation.



# How to draw an activity diagram?

Activity diagram is a flowchart of activities. It represents the workflow between various system activities. Activity diagrams are similar to the flowcharts, but they are not flowcharts. Activity diagram is an advancement of a flowchart that contains some unique capabilities.

Activity diagrams include swim lanes, branching, parallel flow, control nodes, expansion nodes, and object nodes. Activity diagram also supports exception handling.

To draw an activity diagram, one must understand and explore the entire system. All the elements and entities that are going to be used inside the diagram must be known by the user. The central concept which is nothing but an activity must be clear to the user. After analyzing all activities, these activities should be explored to find various constraints that are applied to activities. If there is such a constraint, then it should be noted before developing an activity diagram.

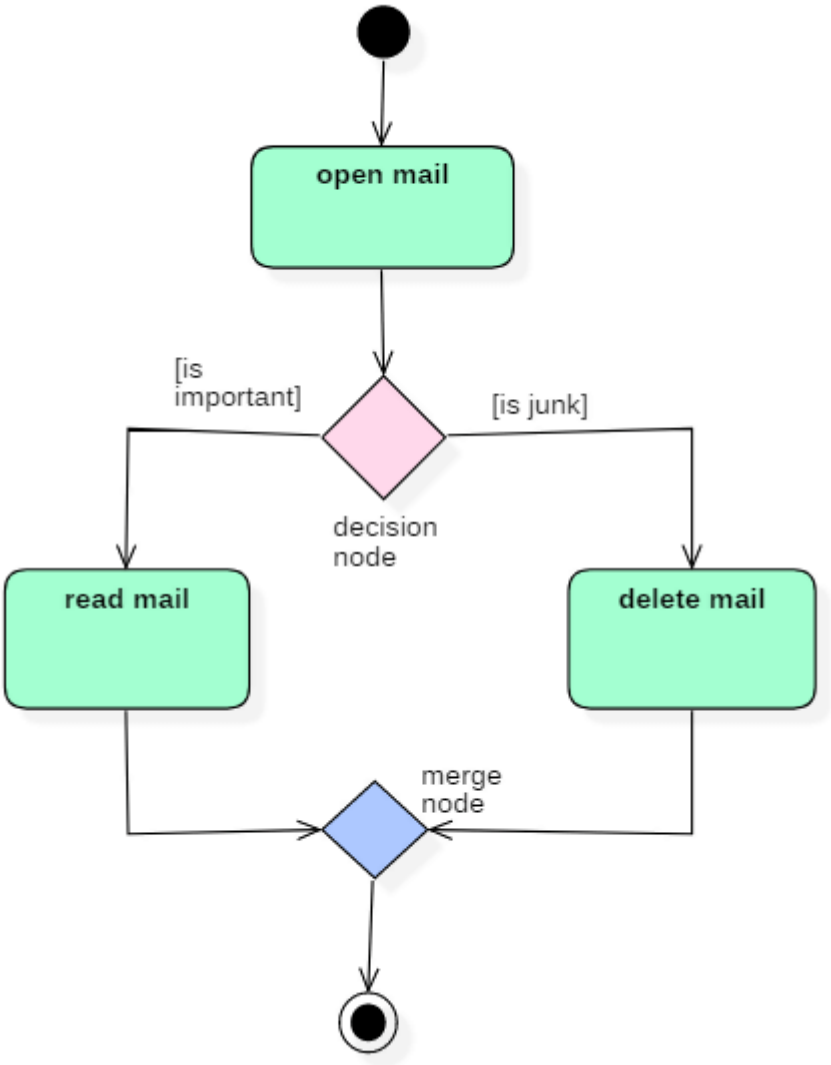
All the activities, conditions, and associations must be known. Once all the necessary things are gathered, then an abstract or a prototype is generated, which is later converted into the actual diagram.

Following rules must be followed while developing an activity diagram,

1. All activities in the system should be named.
2. Activity names should be meaningful.
3. Constraints must be identified.
4. Activity associations must be known.

# Example of Activity Diagram

Let us consider mail processing activity as a sample for Activity Diagram. Following diagram represents activity for processing e-mails.




activity diagram

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.

## When Use Activity Diagram

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.

## Summary

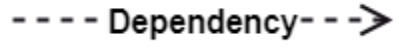
- Activity diagram is also called as **object-oriented flowcharts**.
- Activity diagrams consist of activities that are made up of smaller actions.
- Activity is a behavior that is divided into one or more actions.
- It uses action nodes, control nodes and object nodes.
- An activity partition or a swim lane is a high-level grouping of a set of related actions.
- Fork and join nodes are used to generate concurrent flows within an activity.
- Activity diagram is used to model business processes and workflows.

Relationships

It illustrates the meaningful connections between things. It shows the association between the entities and defines the functionality of an application. There are four types of relationships given below:

**Dependency:** Dependency is a kind of relationship in which a change in target element affects the source element, or simply we can say the source element is dependent on the target element. It is one of the most important notations in UML. It depicts the dependency from one entity to another.

It is denoted by a dotted line followed by an arrow at one side as shown below,



**Association:** A set of links that associates the entities to the UML model. It tells how many elements are actually taking part in forming that relationship.

It is denoted by a dotted line with arrowheads on both sides to describe the relationship with the element on both sides.


**Generalization:** It portrays the relationship between a general thing (a parent class or superclass) and a specific kind of that thing (a child class or subclass). It is used to describe the concept of inheritance.

It is denoted by a straight line followed by an empty arrowhead at one side.


**Realization:** It is a semantic kind of relationship between two things, where one defines the behavior to be carried out, and the other one implements the mentioned behavior. It exists in interfaces.

It is denoted by a dotted line with an empty arrowhead at one side.


Diagrams

The diagrams are the graphical implementation of the models that incorporate symbols and text. Each symbol has a different meaning in the context of the UML diagram. There are thirteen different types of UML diagrams that are available in UML 2.0, such that each diagram has its own set of a symbol. And each diagram manifests a different dimension, perspective, and view of the system.

UML diagrams are classified into three categories that are given below:

1. Structural Diagram
2. Behavioral Diagram
3. Interaction Diagram

**Structural Diagram:** It represents the static view of a system by portraying the structure of a system. It shows several objects residing in the system. Following are the structural diagrams given below:

- Class diagram
- Object diagram
- Package diagram
- Component diagram
- Deployment diagram

**Behavioral Diagram:** It depicts the behavioral features of a system. It deals with dynamic parts of the system. It encompasses the following diagrams:

- Activity diagram
- State machine diagram
- Use case diagram

**Interaction diagram:** It is a subset of behavioral diagrams. It depicts the interaction between two objects and the data flow between them. Following are the several interaction diagrams in UML:

- Timing diagram
- Sequence diagram
- Collaboration diagram

## What is Class?

A Class is a blueprint that is used to create Object. The Class defines what object can do.

## What is Class Diagram?

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 defines the types of objects in the system and the different types of relationships that exist among them. It gives a high-level view of an application. 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.

Class Diagram helps construct the code for the software application development.

## Benefits of Class Diagram

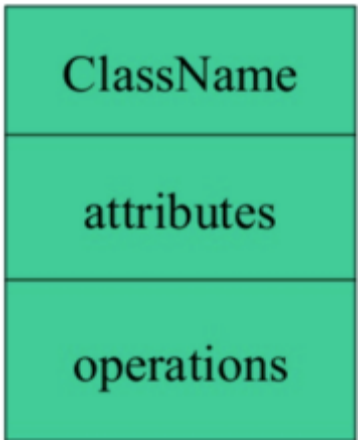
- Class Diagram Illustrates data models for even very complex information systems
- It provides an overview of how the application is structured before studying the actual code. This can easily reduce the maintenance time
- It helps for better understanding of general schematics of an application.
- Allows drawing detailed charts which highlights code required to be programmed
- Helpful for developers and other stakeholders.

## Essential elements of A UML class diagram

Essential elements of UML class diagram are:

1. Class Name
2. Attributes
3. Operations

## Class Name



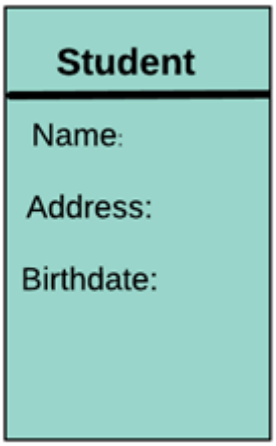

The name of the class is only needed in the graphical representation of the class. It appears in the topmost compartment. A class is the blueprint of an object which can share the same relationships, attributes, operations, & semantics. The class is rendered as a rectangle, including its name, attributes, and operations in sperate compartments.

Following rules must be taken care of while representing a class:

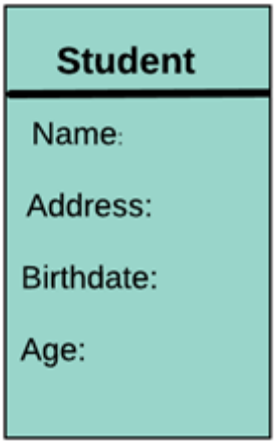
1. A class name should always start with a capital letter.
2. A class name should always be in the center of the first compartment.
3. A class name should always be written in **bold** format.
4. An abstract class name should be written in italics format.

## Attributes:

An attribute is named property of a class which describes the object being modeled. In the class diagram, this component is placed just below the name-compartment.




A derived attribute is computed from other attributes. For example, an age of the student can be easily computed from his/her birth date.




Attributes characteristics

- The attributes are generally written along with the visibility factor.
- Public, private, protected and package are the four visibilities which are denoted by +, -, #, or ~ signs respectively.
- Visibility describes the accessibility of an attribute of a class.
- Attributes must have a meaningful name that describes the use of it in a class.

## Relationships

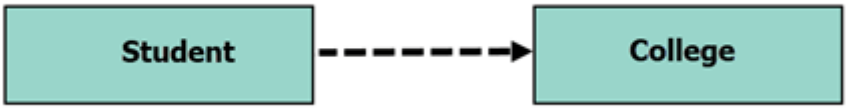
There are mainly three kinds of relationships in UML:

1. Dependencies
2. Generalizations
3. Associations

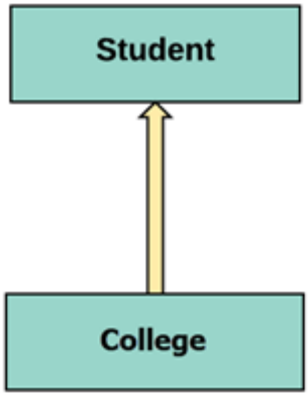
### Dependency

A dependency means the relation between two or more classes in which a change in one may force changes in the other. However, it will always create a weaker relationship. Dependency indicates that one class depends on another.

In the following example, Student has a dependency on College




### Generalization:




A generalization helps to connect a subclass to its superclass. A sub-class is inherited from its superclass. Generalization relationship can't be used to model interface implementation. Class diagram allows inheriting from multiple super classes.

In this example, the class Student is generalized from Person Class.

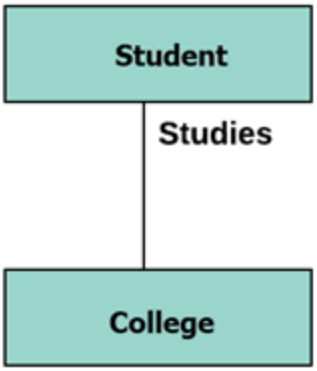
## Association:

This kind of relationship represents static relationships between classes A and B. For example; an employee works for an organization.

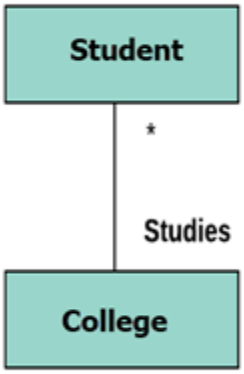
Here are some rules for Association:

- Association is mostly verb or a verb phrase or noun or noun phrase.
- It should be named to indicate the role played by the class attached at the end of the association path.
- Mandatory for reflexive associations

In this example, the relationship between student and college is shown which is studies.




## Multiplicity

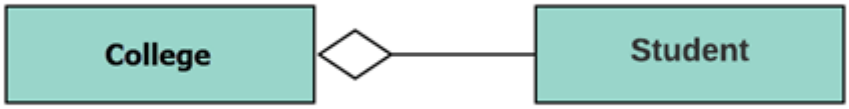


A multiplicity is a factor associated with an attribute. It specifies how many instances of attributes are created when a class is initialized. If a multiplicity is not specified, by default one is considered as a default multiplicity.

Let's say that that there are 100 students in one college. The college can have multiple students.

## Aggregation

Aggregation is a special type of association that models a whole- part relationship between aggregate and its parts.




For example, the class college is made up of one or more student. In aggregation, the contained classes are never totally dependent on the lifecycle of the container. Here, the college class will remain even if the student is not available.

## Composition:


The composition is a special type of aggregation which denotes strong ownership between two classes when one class is a part of another class.

For example, if college is composed of classes student. The college could contain many students, while each student belongs to only one college. So, if college is not functioning all the students also removed.

## Aggregation vs. Composition

| Aggregation | Composition |
| --- | --- |
| Aggregation indicates a relationship where the child can exist separately from their parent class. Example: Automobile (Parent) and Car (Child). So, If you delete the Automobile, the child Car still exist. | Composition display relationship where the child will never exist independent of the parent. Example: House (parent) and Room (child). Rooms will never separate into a House. |

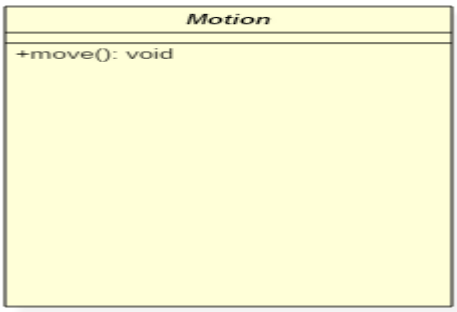
## Abstract Classes

It is a class with an operation prototype, but not the implementation. It is also possible to have an abstract class with no operations declared inside of it. An abstract is useful for identifying the functionalities across the classes. Let us consider an example of an abstract class. Suppose we have an abstract class called as a motion with a method or an operation declared inside of it. The method declared inside the abstract class is called a **move ()**.

This abstract class method can be used by any object such as a car, an animal, robot, etc. for changing the current position. It is efficient to use this abstract class method with an object because no implementation is provided for the given function. We can use it in any way for multiple objects.

In UML, the abstract class has the same notation as that of the class. The only difference between a class and an abstract class is that the class name is strictly written in an italic font.

An abstract class cannot be initialized or instantiated.




Abstract Class Notation

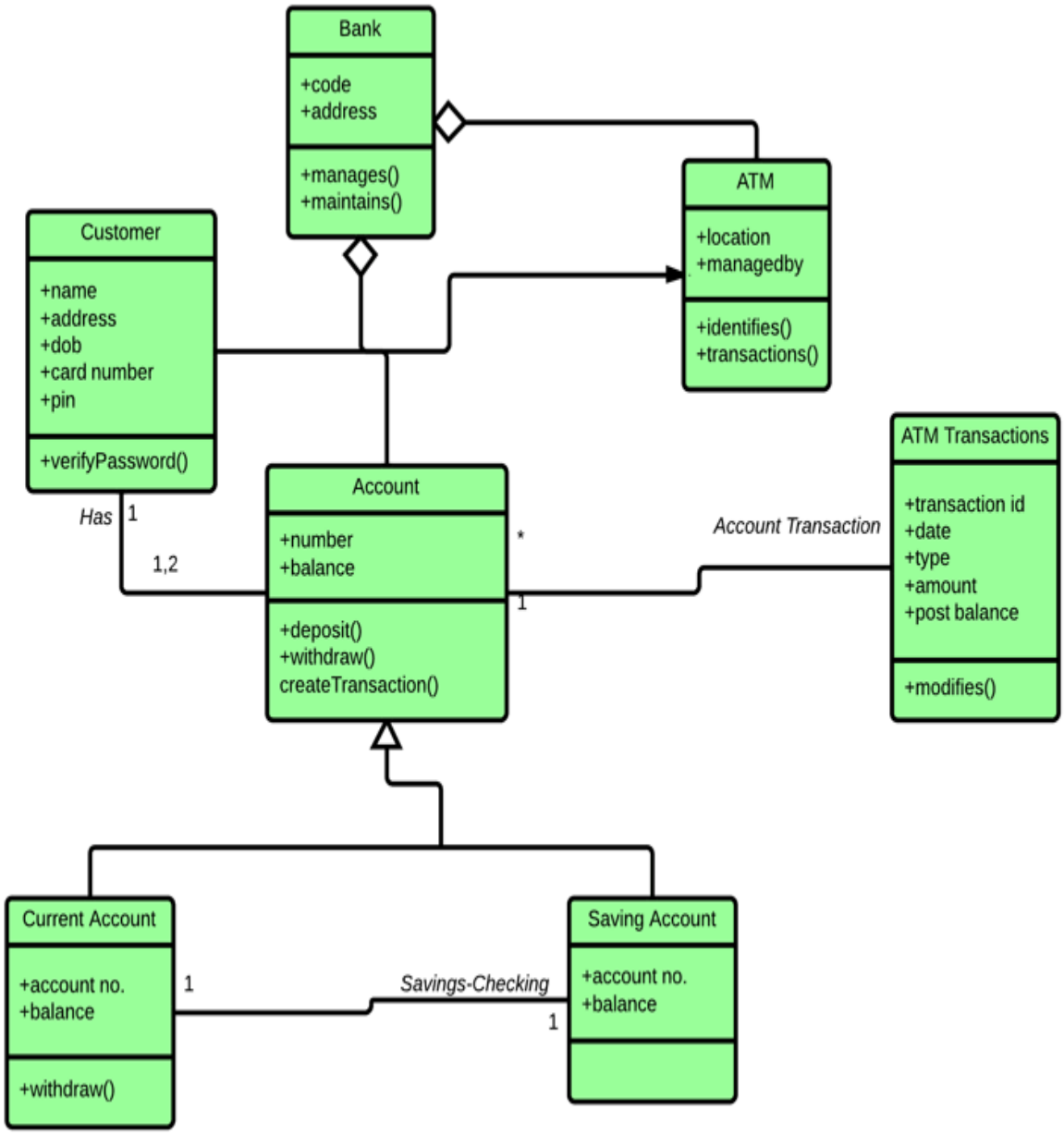
In the above abstract class notation, there is the only a single abstract method which can be used by multiple objects of classes.

## Example of UML Class Diagram

Creating a class diagram is a straightforward process. It does not involve many technicalities. Here, is an example:

ATMs system is very simple as customers need to press some buttons to receive cash. However, there are multiple security layers that any ATM system needs to pass. This helps to prevent fraud and provide cash or need details to banking customers.

Below given is a UML Class Diagram example:




# Class Diagram in Software Development Lifecycle

Class diagrams can be used in various software development phases. It helps in modeling class diagrams in three different perspectives.

**1. Conceptual perspective:** Conceptual diagrams are describing things in the real world. You should draw a diagram that represents the concepts in the domain

under study. These concepts related to class and it is always language-independent.

**2. Specification perspective:** Specification perspective describes software abstractions or components with specifications and interfaces. However, it does not give any commitment to specific implementation.

**3. Implementation perspective:** This type of class diagrams is used for implementations in a specific language or application. Implementation perspective, use for software implementation.

## Best practices of Designing of the Class Diagram

Class diagrams are the most important UML diagrams used for software application development. There are many properties which should be considered while drawing a Class Diagram. They represent various aspects of a software application.

Here, are some points which should be kept in mind while drawing a class diagram:

- The name given to the class diagram must be meaningful. Moreover, It should describe the real aspect of the system.
- The relationship between each element needs to be identified in advance.
- The responsibility for every class needs to be identified.
- For every class, minimum number of properties should be specified. Therefore, unwanted properties can easily make the diagram complicated.
- User notes should be included whenever you need to define some aspect of the diagram. At the end of the drawing, it must be understandable for the software development team.
- Lastly, before creating the final version, the diagram needs to be drawn on plain paper. Moreover, It should be reworked until it is ready for final submission.

# What is the 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-cases are the core concepts of Unified Modelling language modeling.

# Why Use-Case diagram?

A Use Case consists of use cases, persons, or various things that are invoking the features called as actors and the elements that are responsible for implementing the use cases. 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. Use case diagrams are responsible for visualizing the external things that interact with the part of the system.

# Use-case diagram notations

Following are the common notations used in a use case diagram:

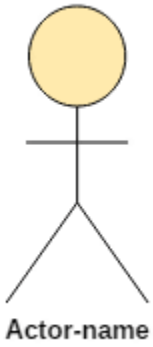
**Use-case:**

Use cases are used to represent high-level functionalities and how the user will handle the system. A use case represents a distinct functionality of a system, a component, a package, or a class. It is denoted by an oval shape with the name of a use case written inside the oval shape. The notation of a use case in UML is given below:


UML UseCase Notation

### Actor:

It is used inside use case diagrams. The actor is an entity that interacts with the system. A user is the best example of an actor. An actor is an entity that initiates the use case from outside the scope of a use case. It can be any element that can trigger an interaction with the use case. One actor can be associated with multiple use cases in the system. The actor notation in UML is given below.




UML Actor Notation

## How to draw a use-case diagram?

To draw a use case diagram in UML first one need to analyse the entire system carefully. You have to find out every single function that is provided by the system. After all the functionalities of a system are found out, then these functionalities are converted into various use cases which will be used in the use case diagram.

A use case is nothing but a core functionality of any working system. After organizing the use cases, we have to enlist the various actors or things that are going to interact with the system. These actors are responsible for invoking the

functionality of a system. Actors can be a person or a thing. It can also be a private entity of a system. These actors must be relevant to the functionality or a system they are interacting with.

After the actors and use cases are enlisted, then you have to explore the relationship of a particular actor with the use case or a system. One must identify the total number of ways an actor could interact with the system. A single actor can interact with multiple use cases at the same time, or it can interact with numerous use cases simultaneously.

Following rules must be followed while drawing use-case for any system:

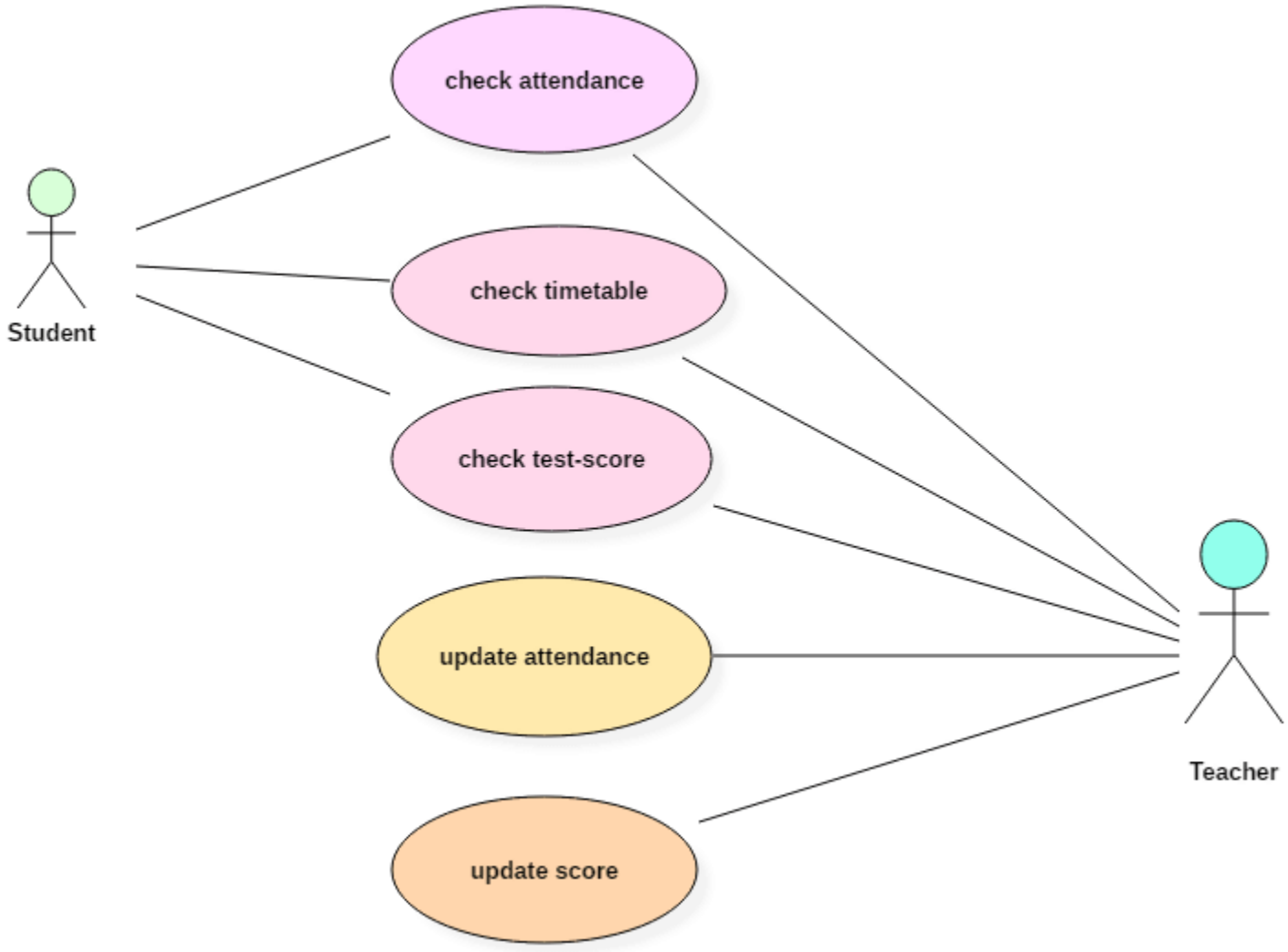
1. The name of an actor or a use case must be meaningful and relevant to the system.
2. Interaction of an actor with the use case must be defined clearly and in an understandable way.
3. Annotations must be used wherever they are required.
4. If a use case or an actor has multiple relationships, then only significant interactions must be displayed.

## Tips for drawing a use-case diagram

1. A use case diagram should be as simple as possible.
2. A use case diagram should be complete.
3. A use case diagram should represent all interactions with the use case.
4. If there are too many use cases or actors, then only the essential use cases should be represented.
5. A use case diagram should describe at least a single module of a system.
6. If the use case diagram is large, then it should be generalized.

## An example of a use-case diagram

Following use case diagram represents the working of the student management system:




UML UseCase Diagram

In the above use case diagram, there are two actors named student and a teacher. There are a total of five use cases that represent the specific functionality of a student management system. Each actor interacts with a particular use case. A student actor can check attendance, timetable as well as test marks on the application or a system. This actor can perform only these interactions with the system even though other use cases are remaining in the system.

It is not necessary that each actor should interact with all the use cases, but it can happen.

The second actor named teacher can interact with all the functionalities or use cases of the system. This actor can also update the attendance of a student and

marks of the student. These interactions of both student and a teacher actor together sums up the entire student management application.

## When to use a use-case diagram?

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.

In general use case diagrams are used for:

1. Analyzing the requirements of a system
2. High-level visual software designing
3. Capturing the functionalities of a system
4. Modeling the basic idea behind the system
5. Forward and reverse engineering of a system using various test cases.

Use cases are intended to convey desired functionality so the exact scope of a use case may vary according to the system and the purpose of creating UML model.

## Summary

- Use case diagrams are a way to capture the system's functionality and requirements in UML diagrams.
- It captures the dynamic behavior of a live system.
- A use case diagram consists of a use case and an actor.
- A use case represents a distinct functionality of a system, a component, a package, or a class.

- An actor is an entity that initiates the use case from outside the scope of a use case.
- The name of an actor or a use case must be meaningful and relevant to the system.
- A purpose of use case diagram is to capture the core functionalities of a system.

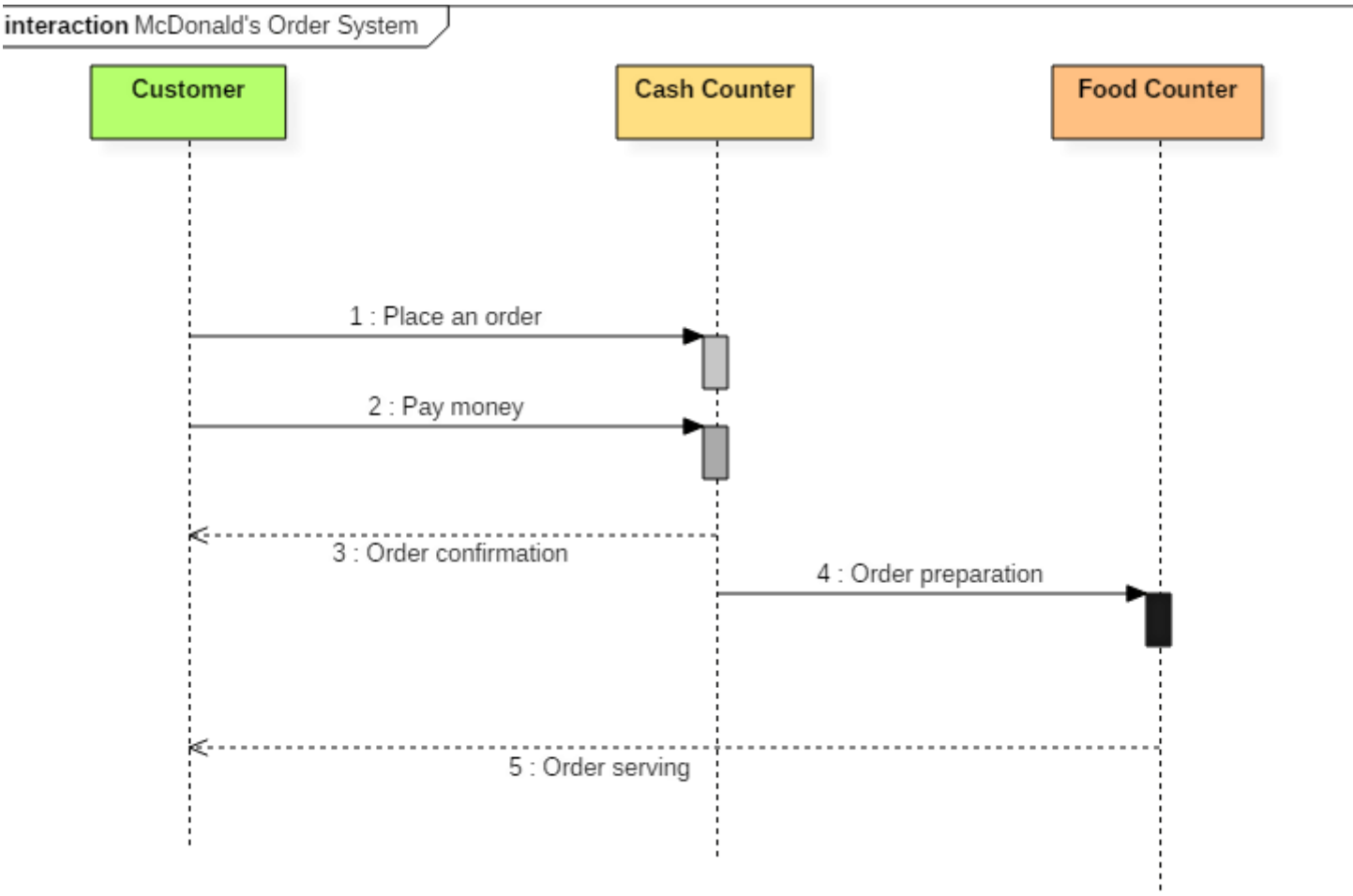
# What is a Sequence Diagram?

A **SEQUENCE DIAGRAM** simply depicts interaction between objects in a sequential order. 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 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.
- In a sequence diagram, different types of messages and operators are used which are described above.
- In a sequence diagram, iteration and branching are also used.

## Sequence diagram example

The following sequence diagram example represents McDonald's ordering system:




Sequence diagram of 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.

If one changes the order of the operations, then it may result in crashing the program. It can also lead to generating incorrect or buggy results. Each sequence in the above-given sequence diagram is denoted using a different type of message. One cannot use the same type of message to denote all the interactions in the diagram because it creates complications in the system.

You must be careful while selecting the notation of a message for any particular interaction. The notation must match with the particular sequence inside the diagram.

### Benefits of a Sequence Diagram

- Sequence diagrams are used to explore any real application or a system.
- Sequence diagrams are used to represent message flow from one object to another object.
- Sequence diagrams are easier to maintain.
- Sequence diagrams are easier to generate.
- Sequence diagrams can be easily updated according to the changes within a system.
- Sequence diagram allows reverse as well as forward engineering.

## What is the Collaboration diagram?

COLLABORATION DIAGRAM depicts the relationships and interactions among software objects. They are used to understand the object architecture within a system rather than the flow of a message as in a sequence diagram. They are also known as “Communication Diagrams.”

As per Object-Oriented Programming (OOPs), an object entity has various attributes associated with it. Usually, there are multiple objects present inside an object-oriented system where each object can be associated with any other object inside the system. Collaboration 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.

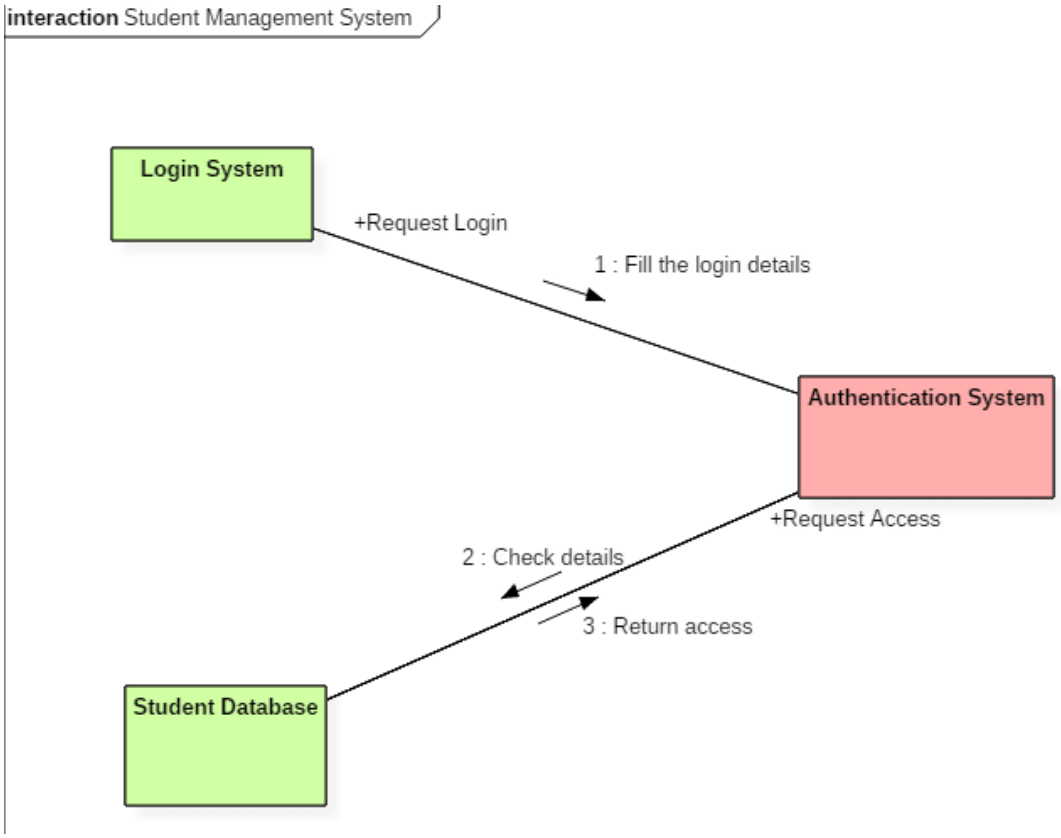
### Benefits of Collaboration Diagram

- 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.
- While modeling collaboration diagrams w.r.t sequence diagrams, some information may be lost.

## Collaboration diagram Example

Following diagram represents the sequencing over student management system:




Collaboration diagram for student management system

The above 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.

## What is Timing diagram?

TIMING DIAGRAM is a waveform or a graph that is used to describe the state of a lifeline at any instance of time. It is used to denote the transformation of an object from one form into another form. Timing diagram does not contain notations as required in the sequence and collaboration diagram. The flow between the software program at various instances of time is represented using a waveform.

- It is a proper representation of interactions that focuses upon the specific timings of messages sent between various objects.
- Timing diagrams are used to explain the detailed time processing of a particular object.
- Timing diagrams are used to explain how an object changes within its lifetime.
- Timing diagrams are mostly used with distributed and embedded systems.
- In UML, timing diagrams are read from left to right according to the name of a lifeline specified at the left edge.
- Timing diagrams are used to represent various changes that occur within a lifeline from time to time.
- Timing diagrams are used to display a graphical representation of various states of a lifeline per unit time.
- UML provides various notations to simplify the transition state between two lifelines per unit time.