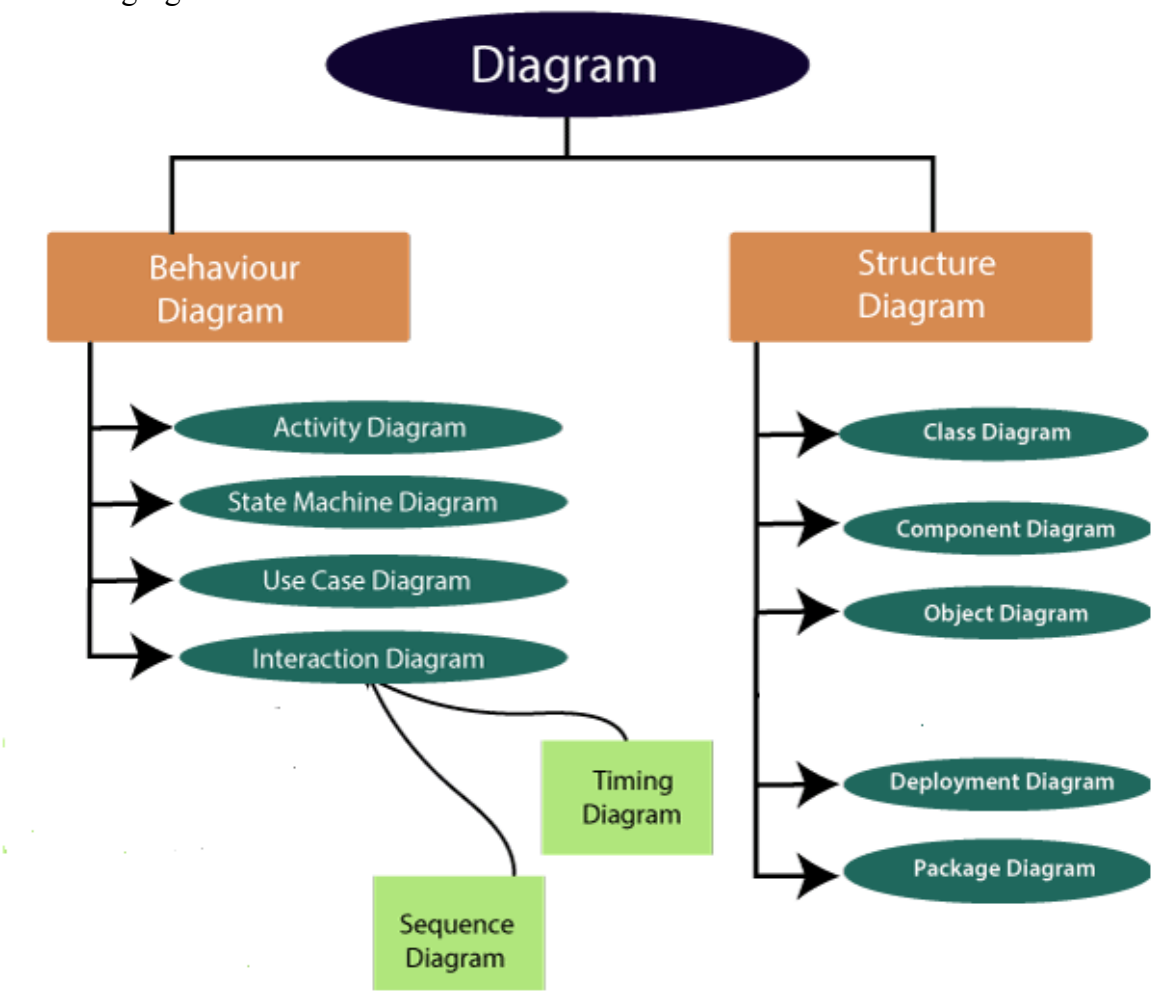
UML-Diagrams

The UML diagrams are categorized into **structural diagrams, behavioral diagrams,** and also interaction **overview diagrams.** The diagrams are hierarchically classified in the following figure:



1 Structural Diagrams

Structural diagrams depict a static view or structure of a system. It is widely used in the documentation of software architecture. It embraces class diagrams, composite structure diagrams, component diagrams, deployment diagrams, object diagrams, and package diagrams. It presents an outline for the system. It stresses the elements to be present that are to be modeled.

- **Class Diagram:** Class diagrams are one of the most widely used diagrams. It is the backbone of all the object-oriented software systems. It depicts the static structure of the system. It displays the system's class, attributes, and methods. It is helpful in recognizing the relation between different objects as well as classes.

- **Object Diagram:** It describes the static structure of a system at a particular point in time. It can be used to test the accuracy of class diagrams. It represents distinct instances of classes and the relationship between them at a time.
- **Component Diagram:** It portrays the organization of the physical components within the system. It is used for modeling execution details. It determines whether the desired functional requirements have been considered by the planned development or not, as it depicts the structural relationships between the elements of a software system.
- **Deployment Diagram:** It presents the system's software and its hardware by telling what the existing physical components are and what software components are running on them. It produces information about system software. It is incorporated whenever software is used, distributed, or deployed across multiple machines with dissimilar configurations.
- **Package Diagram:** It is used to illustrate how the packages and their elements are organized. It shows the dependencies between distinct packages. It manages UML diagrams by making it easily understandable. It is used for organizing the class and use case diagrams.

2 Behavioral Diagrams:

Behavioral diagrams portray a dynamic view of a system or the behavior of a system, which describes the functioning of the system. It includes use case diagrams, state diagrams, and activity diagrams. It defines the interaction within the system.

- **State Machine Diagram:** It is a behavioral diagram. it portrays the system's behavior utilizing finite state transitions. It is also known as the **State-charts** diagram. It models the dynamic behavior of a class in response to external stimuli.
- **Activity Diagram:** It models the flow of control from one activity to the other. With the help of an activity diagram, we can model sequential and concurrent activities. It visually depicts the workflow as well as what causes an event to occur.
- **Use Case Diagram:** It represents the functionality of a system by utilizing actors and use cases. It encapsulates the functional requirement of a system and its association with actors. It portrays the use case view of a system.

3 Interaction Diagrams

Interaction diagrams are a subclass of behavioral diagrams that give emphasis to object interactions and also depicts the flow between various use case elements of a system. In

simple words, it shows how objects interact with each other and how the data flows within them. It consists of communication, interaction overview, sequence, and timing diagrams.

- **Sequence Diagram:** It shows the interactions between the objects in terms of messages exchanged over time. It delineates in what order and how the object functions are in a system.
- **Timing Diagram:** It is a special kind of sequence diagram used to depict the object's behavior over a specific period of time. It governs the change in state and object behavior by showing the time and duration constraints.

UML - Modelling Types

It is very important to distinguish between the UML model. Different diagrams are used for different types of UML modelling. There are three important types of UML modelling.

Structural Modeling

Structural modelling captures the static features of a system. They consist of the following −

- Classes diagrams
- Objects diagrams
- Deployment diagrams
- Package diagrams
- Component diagram

Structural model represents the framework for the system and this framework is the place where all other components exist. Hence, the class diagram, component diagram and deployment diagrams are part of structural modelling. They all represent the elements and the mechanism to assemble them.

The structural model never describes the dynamic behaviour of the system. Class diagram is the most widely used structural diagram.

Behavioural Modeling

Behavioural model describes the interaction in the system. It represents the interaction among the structural diagrams. Behaviouralmodelling shows the dynamic nature of the system. They consist of the following −

- Activity diagrams
- Interaction diagrams
- Use case diagrams

All the above show the dynamic sequence of flow in a system.

Architectural Modeling

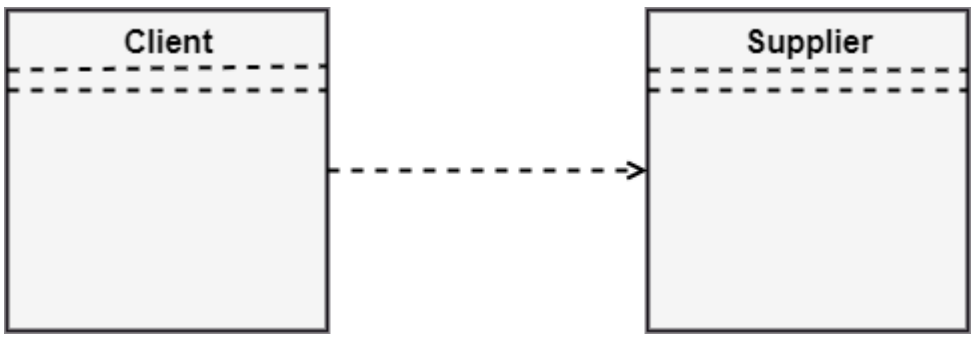
Architectural model represents the overall framework of the system. It contains both structural and behavioral elements of the system. Architectural model can be defined as the blueprint of the entire system. Package diagram comes under architectural modelling.

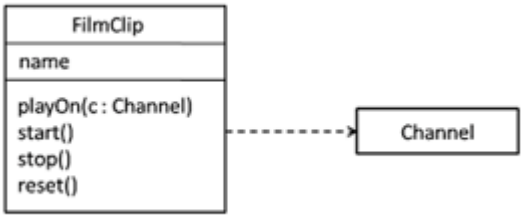
Relationships

A model is not complete unless the relationships between elements are described properly. The *Relationship* gives a proper meaning to a UML model. Following are the different types of relationships available in UML.

Dependency

A dependency is a using relationship. In this relationship one thing depends on the other thing. The change in independent thing will affect the dependent thing. Dependency is graphically represented as a dashed directed line. The arrow head points towards the independent thing. Dependency depicts how various things within a system are dependent on each other. It is used in class diagrams, component diagrams, deployment diagrams, and use-case diagrams, which indicates that a change to the supplier necessitates a change to the client. An example is given below:

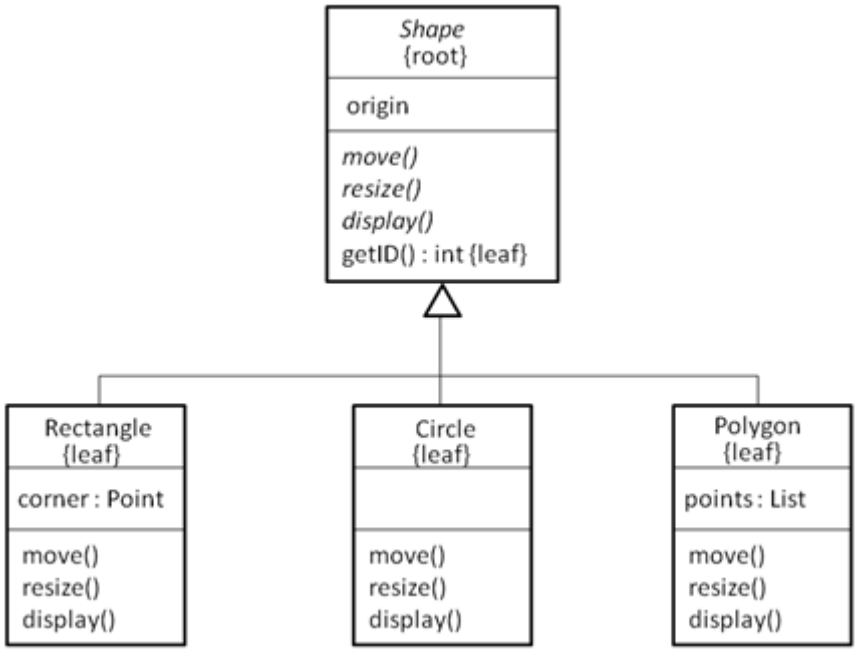




Generalization

A generalization relationship represents generalization-specialization relationship between classes. The class with the general structure and behavior is known as the parent or

superclass and the class with specific structure and behavior is known as the child or subclass. Consider the below class hierarchy:



Shape class is the parent or super class and the remaining three classes namely Rectangle, Circle and Polygon are the child or subclasses of the Shape class.

A subclass in the generalization relationship automatically inherits the state and behavior of the superclass. The generalization relationship is also known as the “is-a” relationship. If a class has only one parent, such inheritance is known as single inheritance and if a class has one or more parents, such inheritance is known as multiple inheritance.

Association

An association is a structural relationship which connects one thing with another. Given an association between two things, we can navigate from one thing to the other thing and vice versa. It is also common for a thing to have a self association.

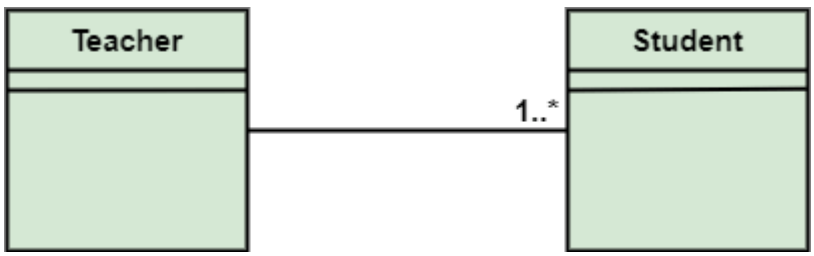
An association which connects exactly two classes is known as binary association. An association which connects n number of classes is known as n-nary association.

Name: An association can have a name which let’s you represent the meaning of the association. The name is written on top of the association. We can also represent the direction in which the name is read.

Role: A thing participating in an association will have a specific role. This role can be represented by writing it outside, near to the thing and below or above the association.

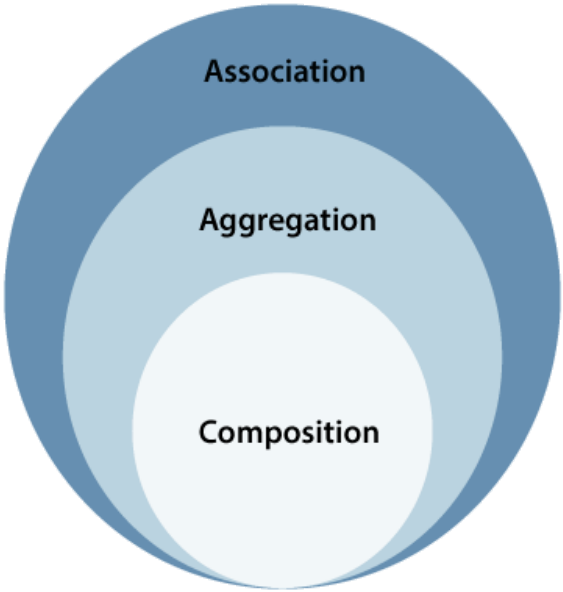
Multiplicity: When connecting things with association, we are concerned about how many objects of one class can connect across an instance of the association. This “how many” is known as multiplicity of an association’s role and is written as an expression.

Example:A single teacher has multiple students.



1) A single student can associate with many teachers.

The composition and aggregation are two subsets of association. In both of the cases, the object of one class is owned by the object of another class; the only difference is that in composition, the child does not exist independently of its parent, whereas in aggregation, the child is not dependent on its parent i.e., standalone. An aggregation is a special form of association, and composition is the special form of aggregation.



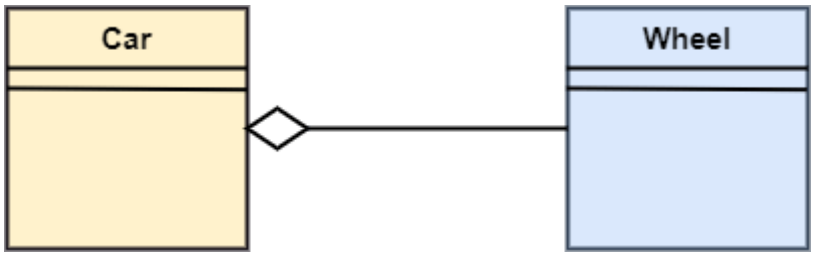
Aggregation

Aggregation is a subset of association, is a collection of different things. It represents has a relationship. It is more specific than an association. It describes a part-whole or part-of

relationship. It is a binary association, i.e., it only involves two classes. It is a kind of relationship in which the child is independent of its parent.

For example:

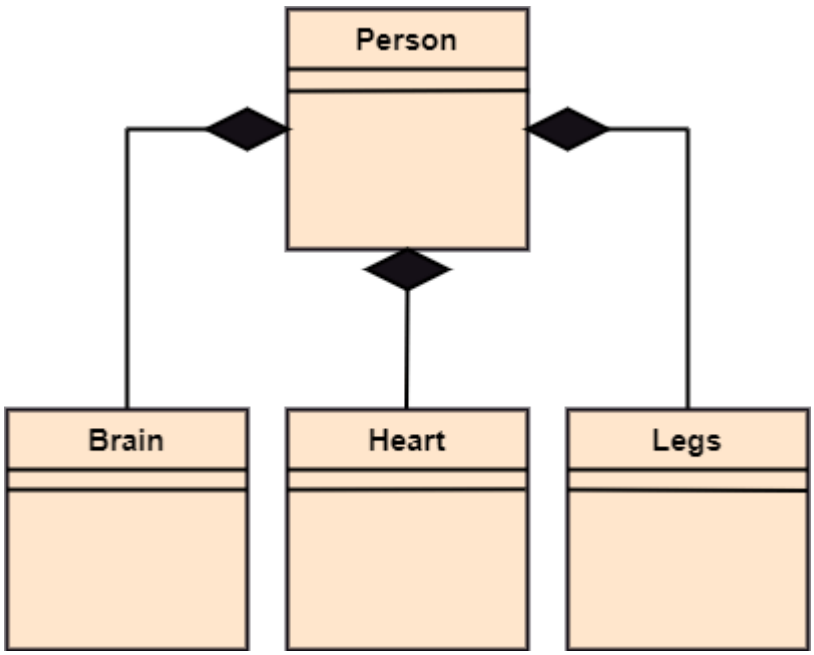
Here we are considering a car and a wheel example. A car cannot move without a wheel. But the wheel can be independently used with the bike, scooter, cycle, or any other vehicle. The wheel object can exist without the car object, which proves to be an aggregation relationship.



Composition

The composition is a part of aggregation, and it portrays the whole-part relationship. It depicts dependency between a composite (parent) and its parts (children), which means that if the composite is discarded, so will its parts get deleted. It exists between similar objects.

As you can see from the example given below, the composition association relationship connects the Person class with Brain class, Heart class, and Legs class. If the person is destroyed, the brain, heart, and legs will also get discarded.

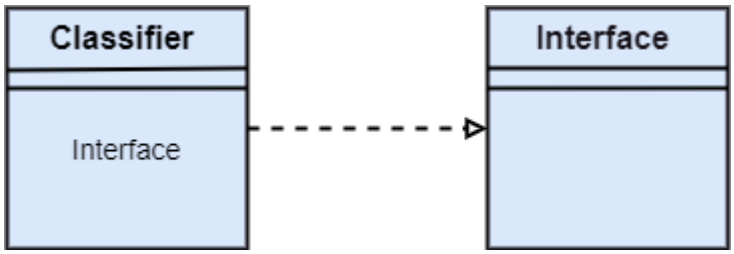


UML-Realization

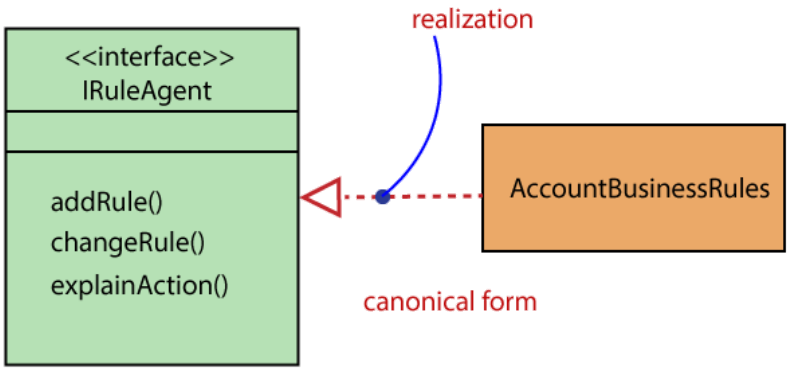
In UML modeling, the realization is a relationship between two objects, where the client (one model element) implements the responsibility specified by the supplier (another model element). The realization relationship can be employed in class diagrams and components diagrams.

It is mostly found in the interfaces. It is represented by a dashed line with a hollow arrowhead

at one end that points from the client to the server.



From the diagram given below, it can be seen that the object **Account Business Rules** realizes the interface **Iruleagent.**



UML Class Diagram

The class diagram depicts a static view of an application. It represents the types of objects residing in the system and the relationships between them. A class consists of its objects, and also it may inherit from other classes. A class diagram is used to visualize, describe, document various different aspects of the system, and also construct executable software code.

It shows the attributes, classes, functions, and relationships to give an overview of the software system. It constitutes class names, attributes, and functions in a separate compartment that helps in software development. Since it is a collection of classes, interfaces, associations, collaborations, and constraints, it is termed as a structural diagram.

Purpose of Class Diagrams

The main purpose of class diagrams is to build a static view of an application. It is the only diagram that is widely used for construction, and it can be mapped with object-oriented languages. It is one of the most popular UML diagrams. Following are the purpose of class diagrams given below:

1. It analyses and designs a static view of an application.
2. It describes the major responsibilities of a system.
3. It is a base for component and deployment diagrams.
4. It incorporates forward and reverse engineering.

The class diagram is made up of three sections:

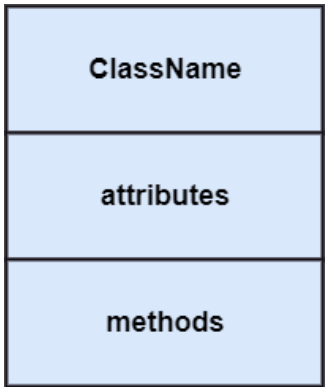
- **Upper Section:** The upper section encompasses the name of the class. A class is a representation of similar objects that shares the same relationships, attributes, operations, and semantics. Some of the following rules that should be taken into account while representing a class are given below:

    a. Capitalize the initial letter of the class name.
    b. Place the class name in the center of the upper section.
    c. A class name must be written in bold format.
    d. The name of the abstract class should be written in italics format.

Middle Section: The middle section constitutes the attributes, which describe the quality of the class. The attributes have the following characteristics:

a. The attributes are written along with its visibility factors, which are public (+), private (-), protected (#), and package (~).
    b. The accessibility of an attribute class is illustrated by the visibility factors.
    c. A meaningful name should be assigned to the attribute, which will explain its usage inside the class.

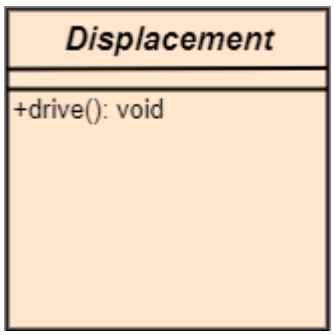
Lower Section: The lower section contain methods or operations. The methods are represented in the form of a list, where each method is written in a single line. It demonstrates how a class interacts with data.



Abstract Classes

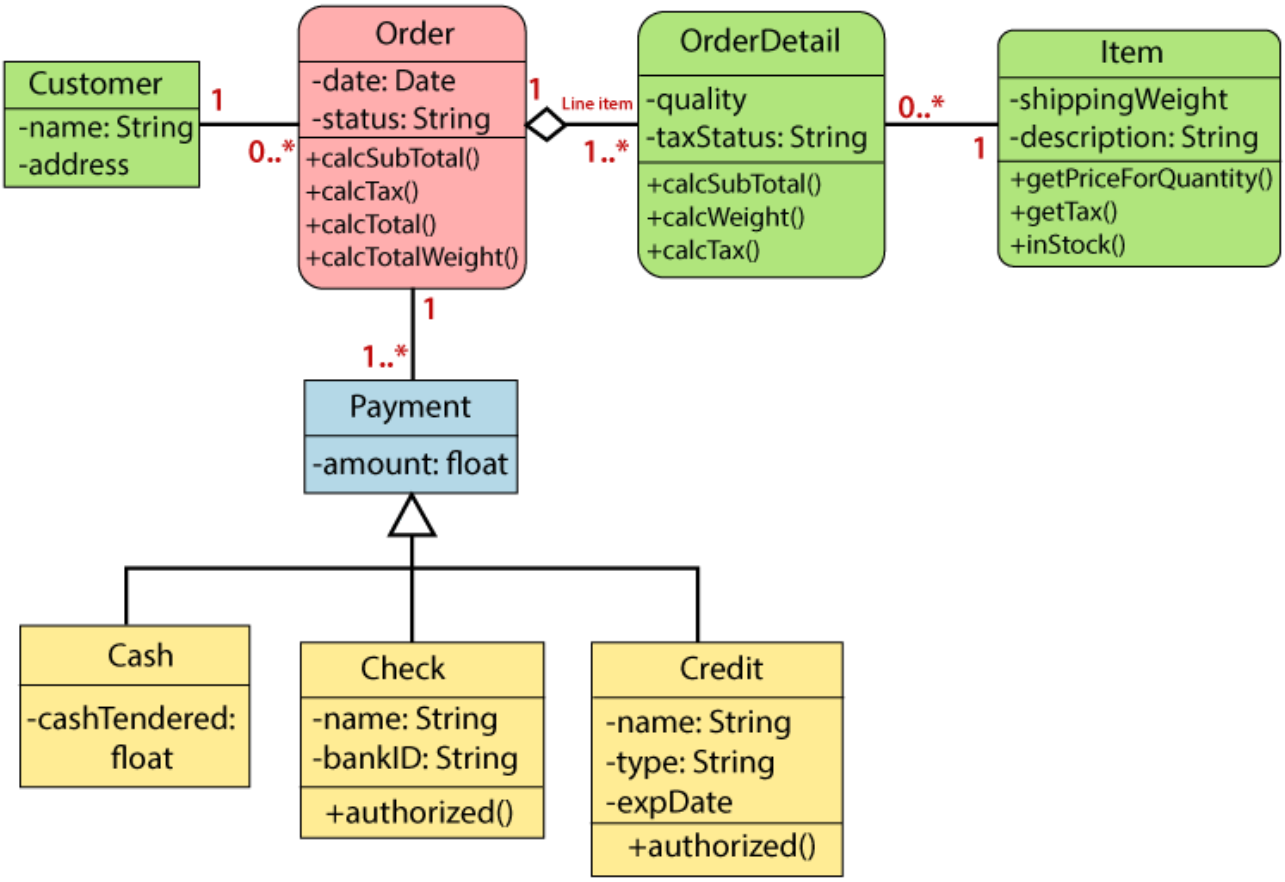
In the abstract class, no objects can be a direct entity of the abstract class. The abstract class can neither be declared nor be instantiated. It is used to find the functionalities across the classes. The notation of the abstract class is similar to that of class; the only difference is that the name of the class is written in italics. Since it does not involve any implementation for a given function, it is best to use the abstract class with multiple objects.

Let us assume that we have an abstract class named **displacement** with a method declared inside it, and that method will be called as a **drive ()**. Now, this abstract class method can be implemented by any object, for example, car, bike, scooter, cycle, etc.



Class Diagram Example

A class diagram describing the sales order system is given below

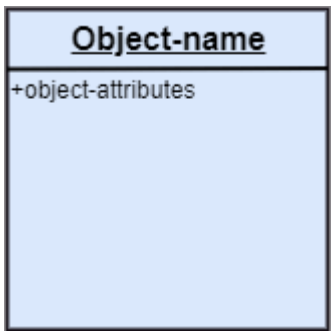


UML Object Diagram

Object diagrams are dependent on the class diagram as they are derived from the class diagram. It represents an instance of a class diagram. The objects help in portraying a static view of an object-oriented system at a specific instant.

Both the object and class diagram are similar to some extent; the only difference is that the class diagram provides an abstract view of a system. It helps in visualizing a particular functionality of a system.

Notation of an Object Diagram



Purpose of Object Diagram

The object diagram holds the same purpose as that of a class diagram. The class diagram provides an abstract view which comprises of classes and their relationships, whereas the object diagram represents an instance at a particular point of time.

The object diagram is actually similar to the concrete (actual) system behavior. The main purpose is to depict a static view of a system.

Following are the purposes enlisted below:

- It is used to understand object behavior and their relationships practically.
- It is used to get a static view of a system.
- It is used to represent an instance of a system.

Before drawing an object diagram, the following things should be remembered and understood clearly −

- Object diagrams consist of objects.
- The link in object diagram is used to connect objects.
- Objects and links are the two elements used to construct an object diagram.

The following diagram is an example of an object diagram. It represents the Order management system which we have discussed in the chapter Class Diagram. The following diagram is an instance of the system at a particular time of purchase. It has the following objects.

- Customer
- Order
- SpecialOrder
- NormalOrder

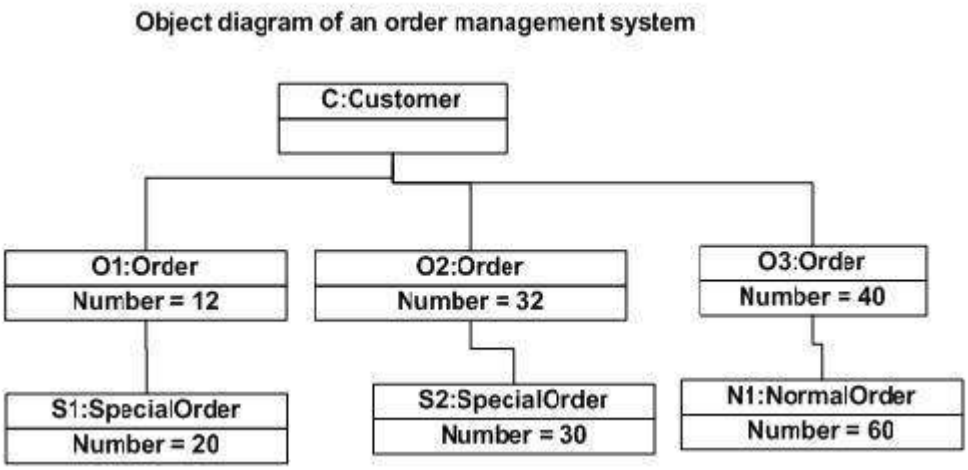
Now the customer object (C) is associated with three order objects (O1, O2, and O3). These order objects are associated with special order and normal order objects (S1, S2, and N1). The customer has the following three orders with different numbers (12, 32 and 40) for the particular time considered.

The customer can increase the number of orders in future and in that scenario the object diagram will reflect that. If order, special order, and normal order objects are observed then you will find that they have some values.

For orders, the values are 12, 32, and 40 which implies that the objects have these values for a particular moment (here the particular time when the purchase is made is considered as the moment) when the instance is captured

The same is true for special order and normal order objects which have number of orders as 20, 30, and 60. If a different time of purchase is considered, then these values will change accordingly.

The following object diagram has been drawn considering all the points mentioned above



UML Interaction Diagram

As the name suggests, the interaction diagram portrays the interactions between distinct entities present in the model. It amalgamates both the activity and sequence diagrams. The communication is nothing but units of the behavior of a classifier that provides context for interactions.

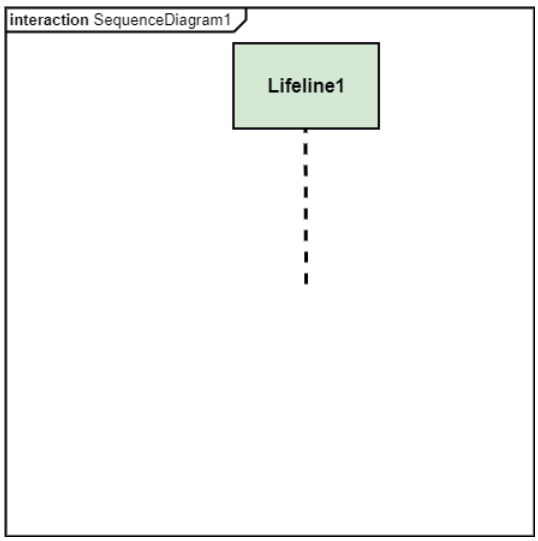
A set of messages that are interchanged between the entities to achieve certain specified tasks in the system is termed as interaction. It may incorporate any feature of the classifier of which it has access. In the interaction diagram, the critical component is the messages and the lifeline.

The message exchanged among objects is either to pass some information or to request some information. And based on the information, the interaction diagram is categorized into the sequence diagram, collaboration diagram, and timing diagram.

The sequence diagram envisions the order of the flow of messages inside the system by depicting the communication between two lifelines, just like a time-ordered sequence of events.

The collaboration diagram, which is also known as the communication diagram, represents how lifelines connect within the system, whereas the timing diagram focuses on that instant when a message is passed from one element to the other.

Notation of an Interaction Diagram



Purpose of an Interaction Diagram

The interaction diagram helps to envision the interactive (dynamic) behavior of any system. It portrays how objects residing in the system communicates and connects to each other. It also provides us with a context of communication between the lifelines inside the system.

Following are the purpose of an interaction diagram given below:

1. To visualize the dynamic behavior of the system.
2. To envision the interaction and the message flow in the system.
3. To portray the structural aspects of the entities within the system.
4. To represent the order of the sequenced interaction in the system.
5. To visualize the real-time data and represent the architecture of an object-oriented system.

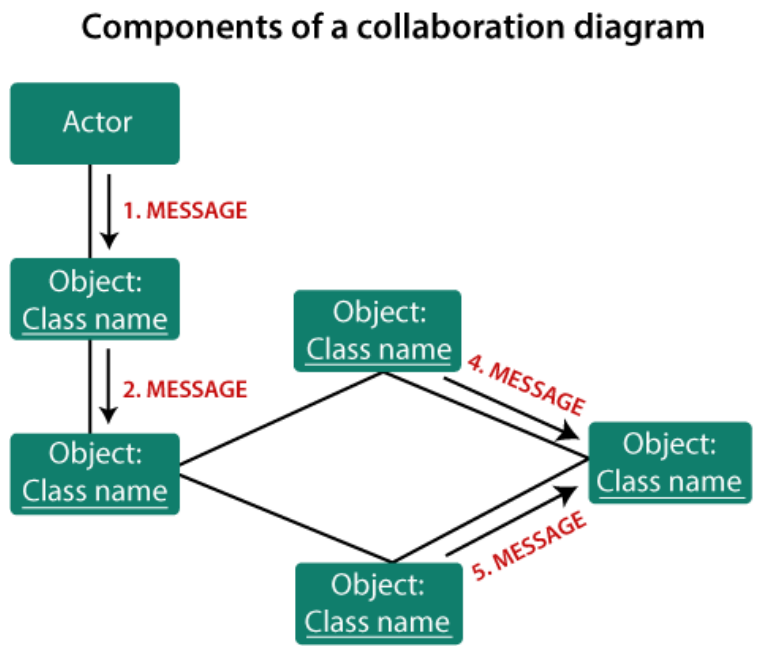
UML Collaboration Diagram

The collaboration diagram is used to show the relationship between the objects in a system. Both the sequence and the collaboration diagrams represent the same information but differently. Instead of showing the flow of messages, it depicts the architecture of the object residing in the system as it is based on object-oriented programming. An object consists of several features. Multiple objects present in the system are connected to each other. The collaboration diagram, which is also known as a communication diagram, is used to portray the object's architecture in the system.

Notations of a Collaboration Diagram

Following are the components of a component diagram that are enlisted below:

1. **Objects:** The representation of an object is done by an object symbol with its name and class underlined, separated by a colon.
   In the collaboration diagram, objects are utilized in the following ways:
   - The object is represented by specifying their name and class.
   - It is not mandatory for every class to appear.
   - A class may constitute more than one object.
   - In the collaboration diagram, firstly, the object is created, and then its class is specified.
   - To differentiate one object from another object, it is necessary to name them.
2. **Actors:** In the collaboration diagram, the actor plays the main role as it invokes the interaction. Each actor has its respective role and name. In this, one actor initiates the use case.
3. **Links:** The link is an instance of association, which associates the objects and actors. It portrays a relationship between the objects through which the messages are sent. It is represented by a solid line. The link helps an object to connect with or navigate to another object, such that the message flows are attached to links.
4. **Messages:** It is a communication between objects which carries information and includes a sequence number, so that the activity may take place. It is represented by a labeled arrow, which is placed near a link. The messages are sent from the sender to the receiver, and the direction must be navigable in that particular direction. The receiver must understand the message.



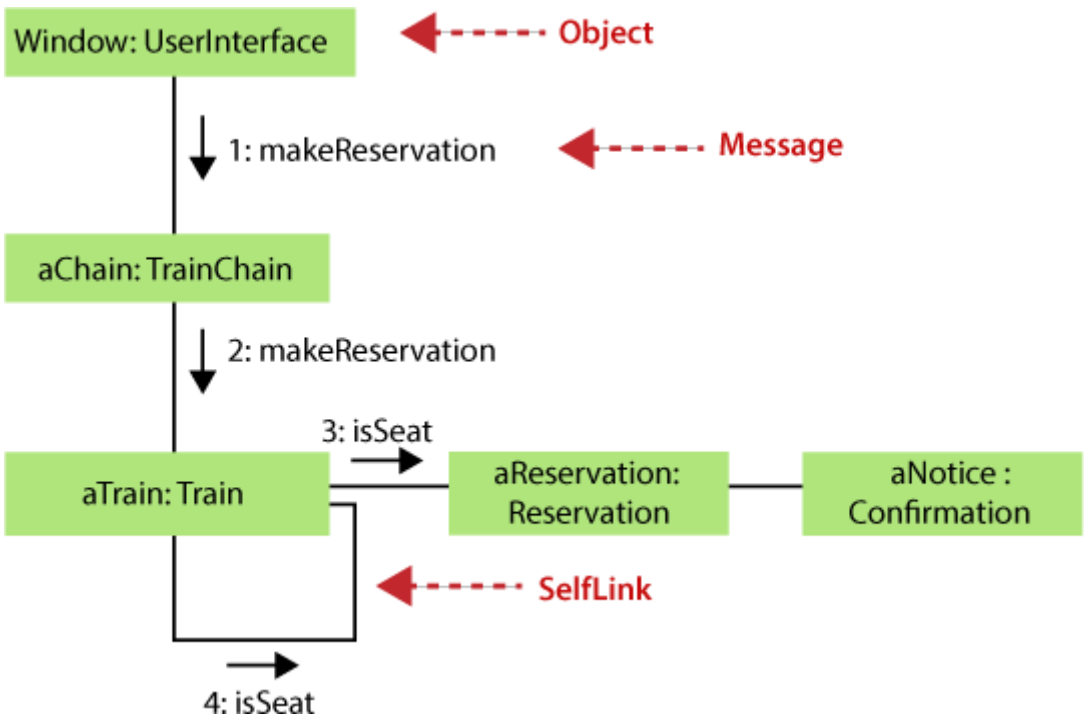
When to use a Collaboration Diagram?

The collaborations are used when it is essential to depict the relationship between the object. Both the sequence and collaboration diagrams represent the same information, but the way of portraying it quite different. The collaboration diagrams are best suited for analyzing use cases.

Following are some of the use cases enlisted below for which the collaboration diagram is implemented:

1. To model collaboration among the objects or roles that carry the functionalities of use cases and operations.
2. To model the mechanism inside the architectural design of the system.
3. To capture the interactions that represent the flow of messages between the objects and the roles inside the collaboration.
4. To model different scenarios within the use case or operation, involving a collaboration of several objects and interactions.
5. To support the identification of objects participating in the use case.
6. In the collaboration diagram, each message constitutes a sequence number, such that the top-level message is marked as one and so on. The messages sent during the same call are denoted with the same decimal prefix, but with different suffixes of 1, 2, etc. as per their occurrence.

Example of a Collaboration Diagram



Polymorphism in Collaboration Diagram:

Polymorphism bring great power to Object orientation. In structured design world we always knew : Call A means Call A. But if object oriented polymorphism is applied, the sender object may not be aware of the exact class of target object. So then what name should be given to the class of the target?

The answer to this is: Make the target's class the lowest class in the inheritance hierarchy i.e. the super class of all the classes to which the target object may possibly belong. For example : If the message is XYZ and the target object may be of class **Triangle, Rectangle or Hexagon**, then the class name on the target object would be **Shape** (assuming that Shape is the direct superclass of all the three classes)

One of the ways in which the exact class will be determined at the run time, is to show the target class name in the parenthesis as shown below:

Fig. 5.6: The method to implement **scale** *will be from* **Polygon** *or one of its subclasses and will be dynamically bound to the target object,* **icon**.

The Callback Mechanism

The callback mechanism is popular use of asynchronous message in which the following event occurs:

1. An object A registers an interest in some event type via an asynchronous message to the target object B
2. A continues with other activities while B monitors for the occurrence of an event of registered type.
3. When an event of that type occurs, object B sends a message (usually asynchronous) back to A to notify A of the occurrence. This is the callback. Object B can then continue with other activities.

Iterated Message

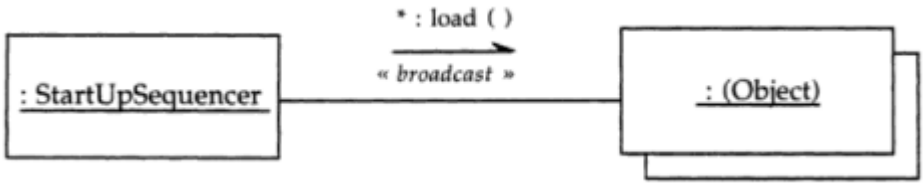
An iterated message is the one which is sent repeatedly, typically to each constituent of aggregate object.

- An asterisk (*) indicates that a message runs more than once
- Or the number of times message is repeated can be shown by numbers(1,2…).

Broadcast Message

In some cases, a sender may broadcast a message- that is treat every object in the system as a potential target. A copy of message goes into queue of every object in the system.

Below figure shows the UML for a broadcast message, where stereotype <<broadcast>> is used to indicate broadcast nature of this message.



Here the double symbol on the target indicates multiple objects.

Asynchronous message with priority

Once we allow concurrency, we allow a target object to be bombarded with messages from lots of concurrently executing sender objects. Messages waiting in the queue may be ordered by priority.

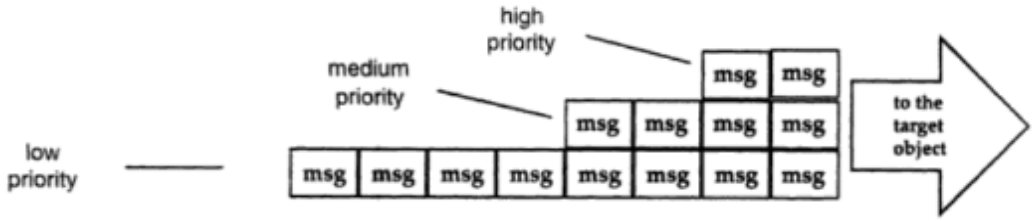
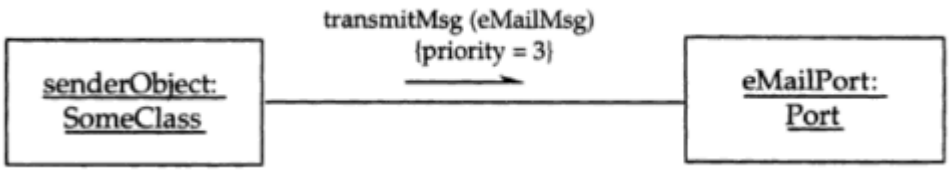


Fig. 5.17: Three parallel queues, each with its own priority.

For eg. In figure shown below asynchronous message is sent to the eMailPort object, asking it to transmit some outgoing email. Sometime email message arrive at eMailPort faster than they can leave, so they have to go in multiple priority queue. The priority {priority = 3} alongside the message arrow indicates that the message in this figure have a priority of 3



Sequence Diagram

A sequence diagram simply depicts interaction between objects in a sequential order i.e. the order in which these interactions take place. We can also use the terms event diagrams or event scenarios to refer to a sequence diagram. Sequence diagrams describe how and in what order the objects in a system function. These diagrams are widely used by businessmen and software developers to document and understand requirements for new and existing systems.

Sequence Diagram Notations –

1. **Actors –** An actor in a UML diagram represents a type of role where it interacts with the system and its objects. It is important to note here that an actor is always outside the scope of the system we aim to model using the UML diagram.

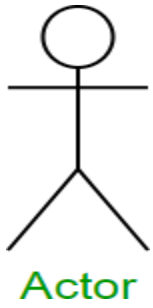


Figure – notation symbol for actor

We use actors to depict various roles including human users and other external subjects. We represent an actor in a UML diagram using a stick person notation. We can have multiple actors in a sequence diagram.
For example – Here the user in seat reservation system is shown as an actor where it exists outside the system and is not a part of the system.

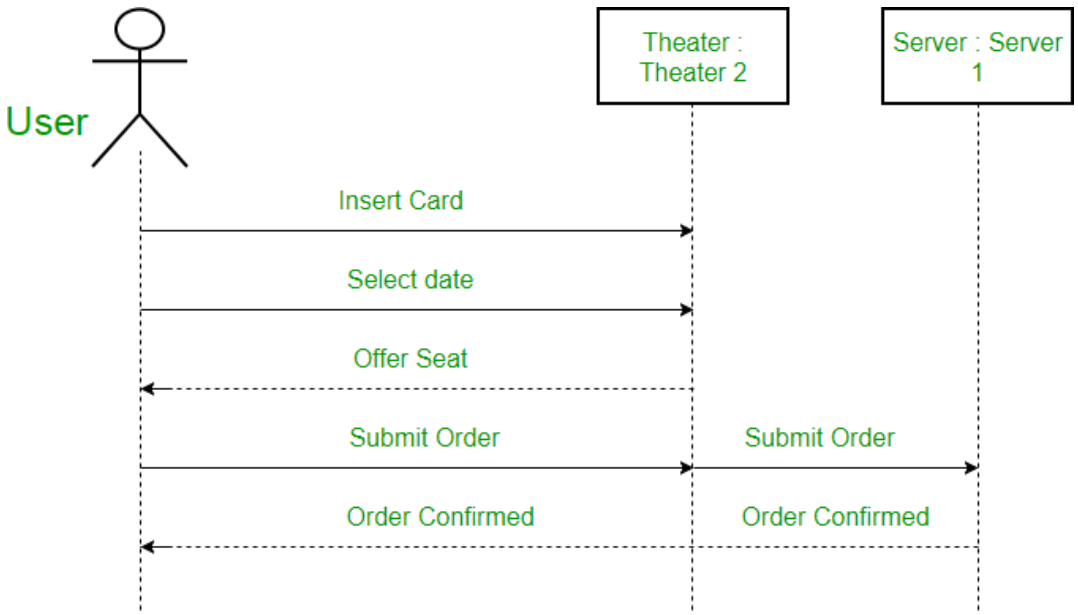


Figure – an actor interacting with a seat reservation system

2. **Lifelines –** A lifeline is a named element which depicts an individual participant in a sequence diagram. So basically each instance in a sequence diagram is represented by a lifeline. Lifeline elements are located at the top in a sequence diagram. The standard in UML for naming a lifeline follows the following format – Instance Name : Class Name

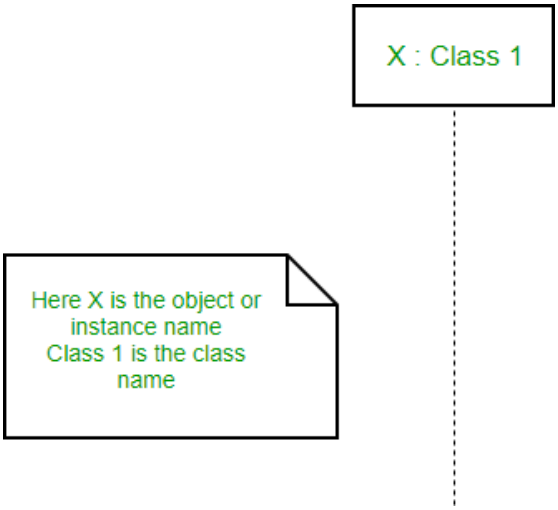


Figure – lifeline

We display a lifeline in a rectangle called head with its name and type. The head is located on top of a vertical dashed line (referred to as the stem) as shown above. If we want to model an unnamed instance, we follow the same pattern except now the portion of lifeline's name is left blank.

Difference between a lifeline and an actor – A lifeline always portrays an object internal to the system whereas actors are used to depict objects external to the system. The following is an example of a sequence diagram:

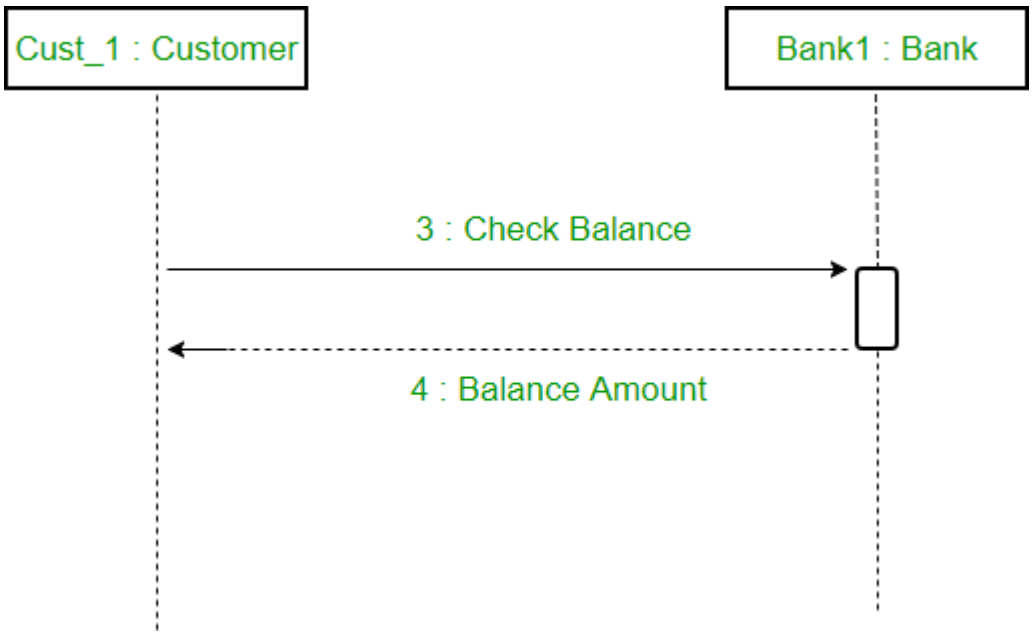


Figure – a sequence diagram

3. **Messages –** Communication between objects is depicted using messages. The messages appear in a sequential order on the lifeline. We represent messages using arrows. Lifelines and messages form the core of a sequence diagram.
   Messages can be broadly classified into the following **categories** :

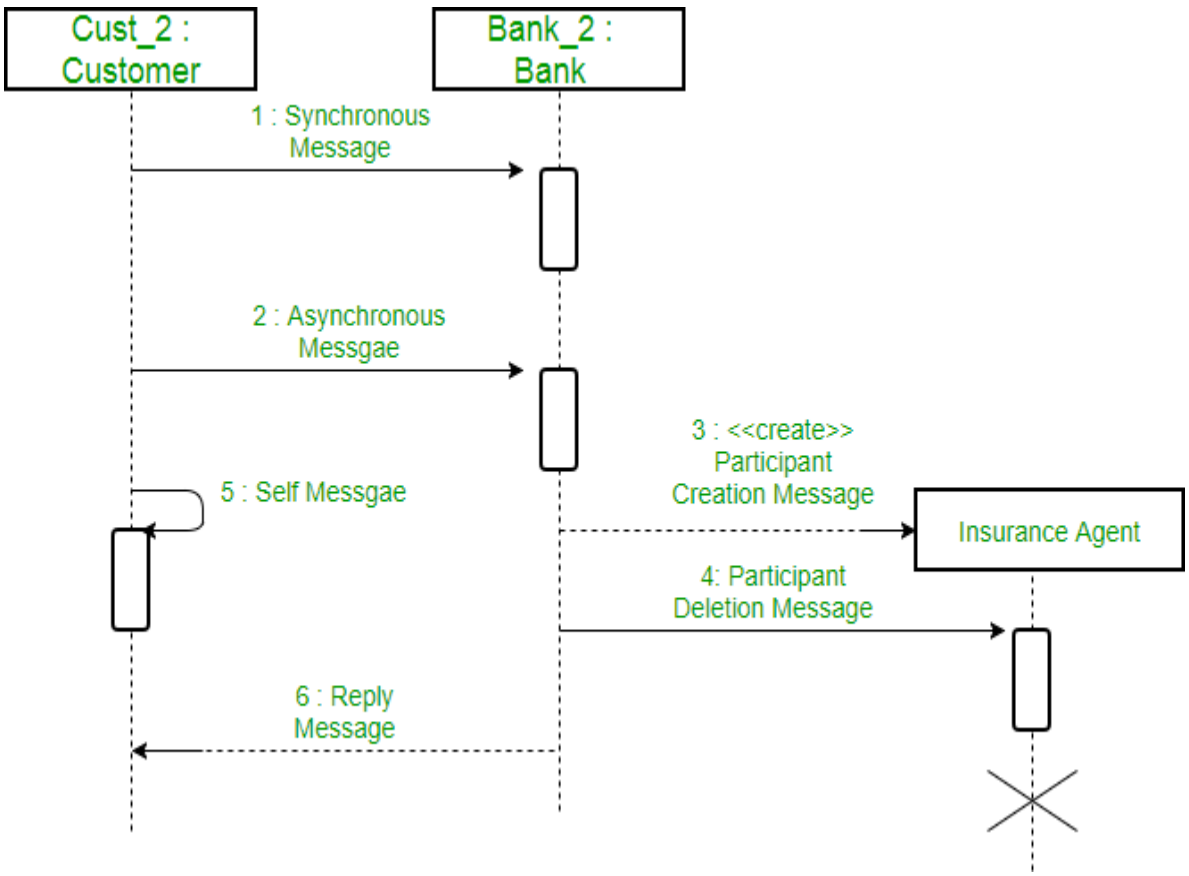


Figure – a sequence diagram with different types of messages

- **Synchronous messages –** 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 when it knows that the receiver has processed the previous message i.e. it receives a reply message. A large number of calls in object oriented programming are synchronous. We use a solid arrow head to represent a synchronous message.

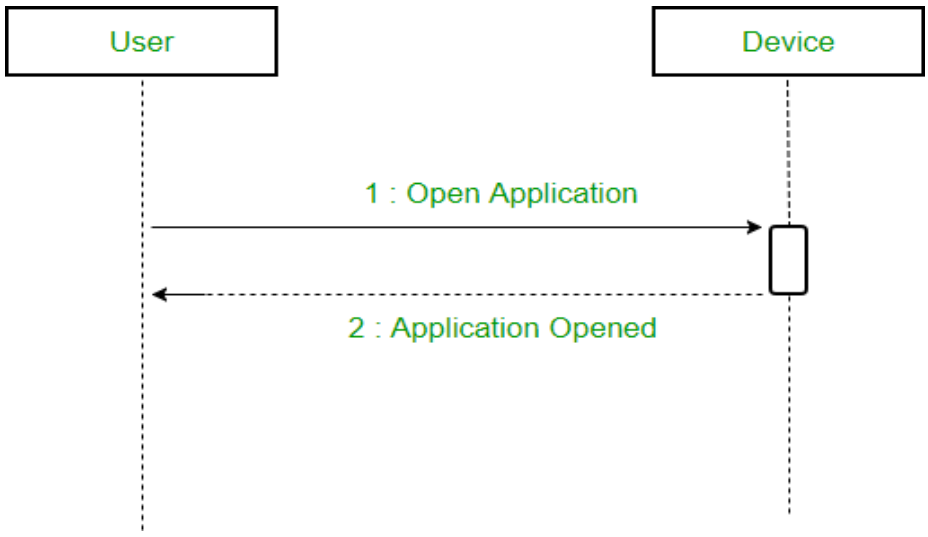
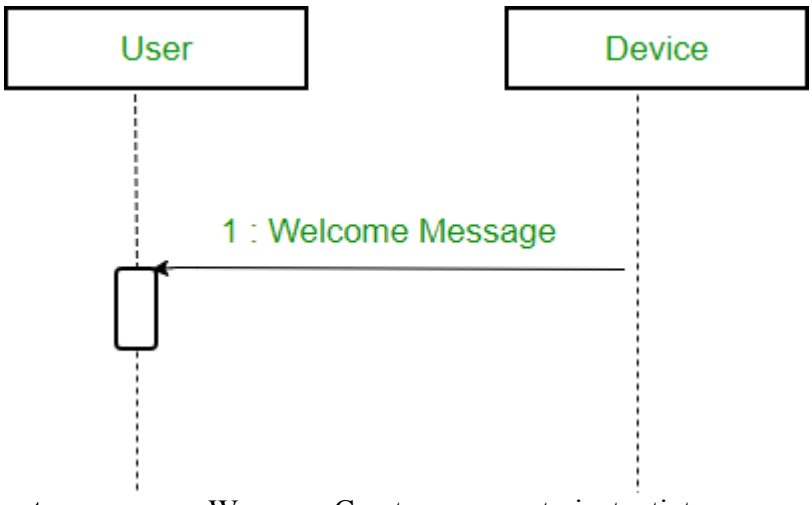


Figure – a sequence diagram using a synchronous message

- **Asynchronous Messages –** 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. We use a lined arrow head to represent an asynchronous message.



- **Create message –** We use a Create message to instantiate a new object in the sequence diagram. There are situations when a particular message call requires the creation of an object. It is represented with a dotted arrow and create word labelled on it to specify that it is the create Message symbol.
  For example – The creation of a new order on a e-commerce website would require a new object of Order class to be created.

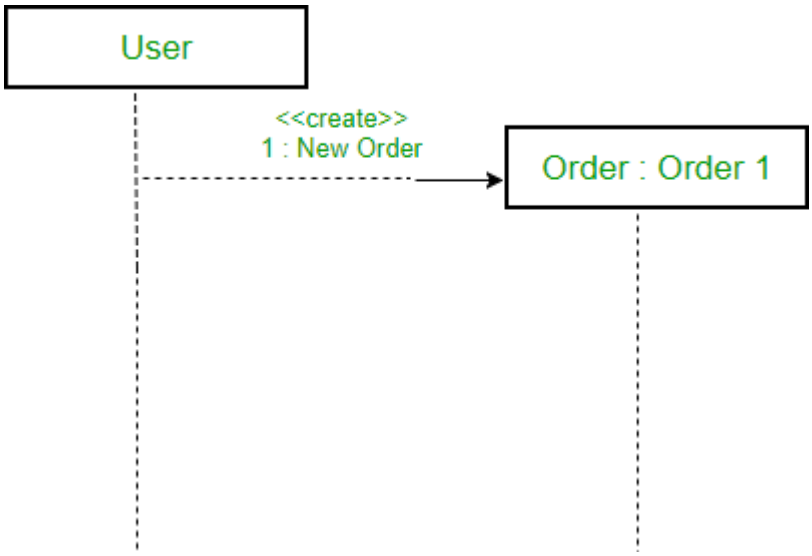


Figure – a situation where create message is used

- **Delete Message –** We use a Delete Message to delete an object. When an object is deallocated memory or is destroyed within the system we use the Delete Message symbol. It destroys the occurrence of the object in the system.It is represented by an arrow terminating with a x.
  For example – In the scenario below when the order is received by the user, the object of order class can be destroyed.

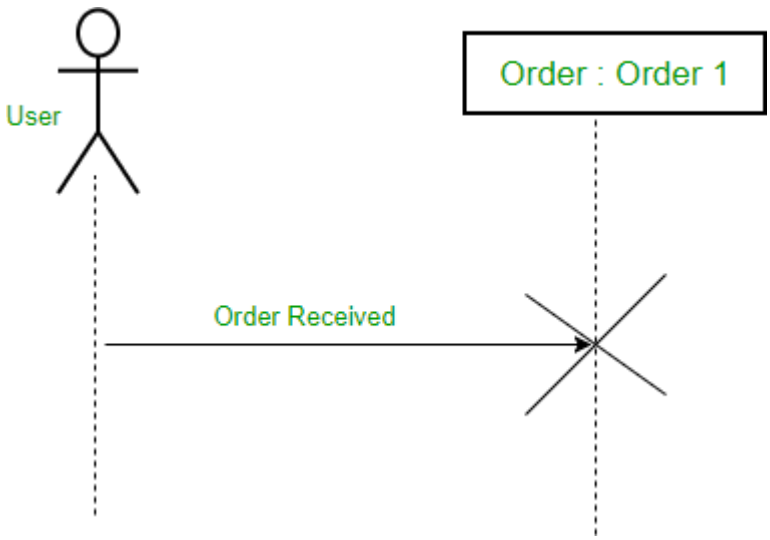


Figure – a scenario where delete message is used

- **Self Message –** Certain scenarios might arise where the object needs to send a message to itself. Such messages are called Self Messages and are represented with a U shaped arrow.

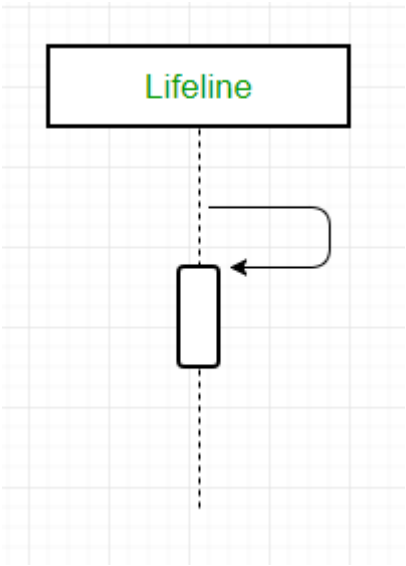


Figure – self message

For example – Consider a scenario where the device wants to access its webcam. Such a scenario is represented using a self message.

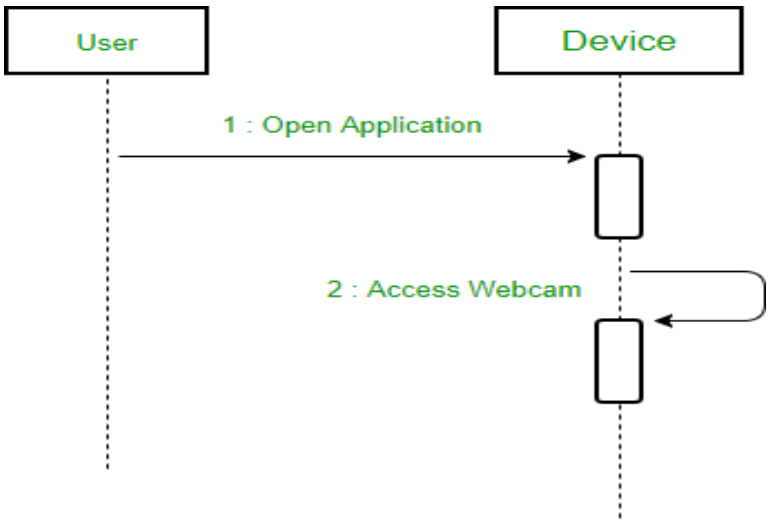


Figure – a scenario where a self message is used

- **Reply Message –** Reply messages are used to show the message being sent from the receiver to the sender. We represent a return/reply message using an open arrowhead with a dotted line. The interaction moves forward only when a reply message is sent by the receiver.

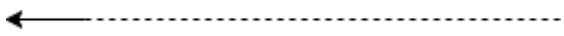


Figure – reply message

For example – Consider the scenario where the device requests a photo from the user. Here the message which shows the photo being sent is a reply message.

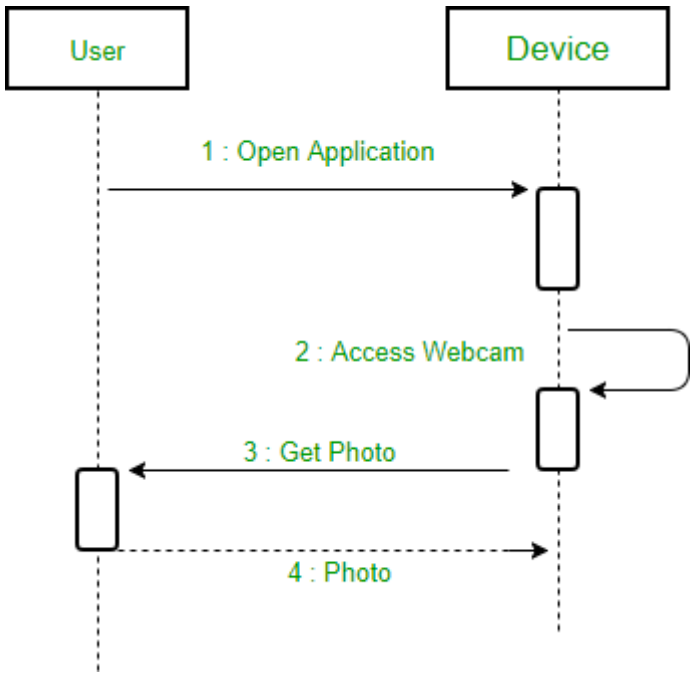


Figure – a scenario where a reply message is used

- **Found Message –** A Found message is used to represent a scenario where an unknown source sends the message. It is represented using an arrow directed towards a lifeline from an end point. For example: Consider the scenario of a hardware failure.

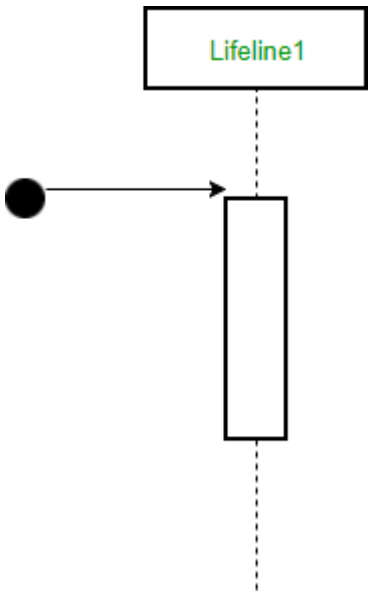


Figure – found message

It can be due to multiple reasons and we are not certain as to what caused the hardware failure.

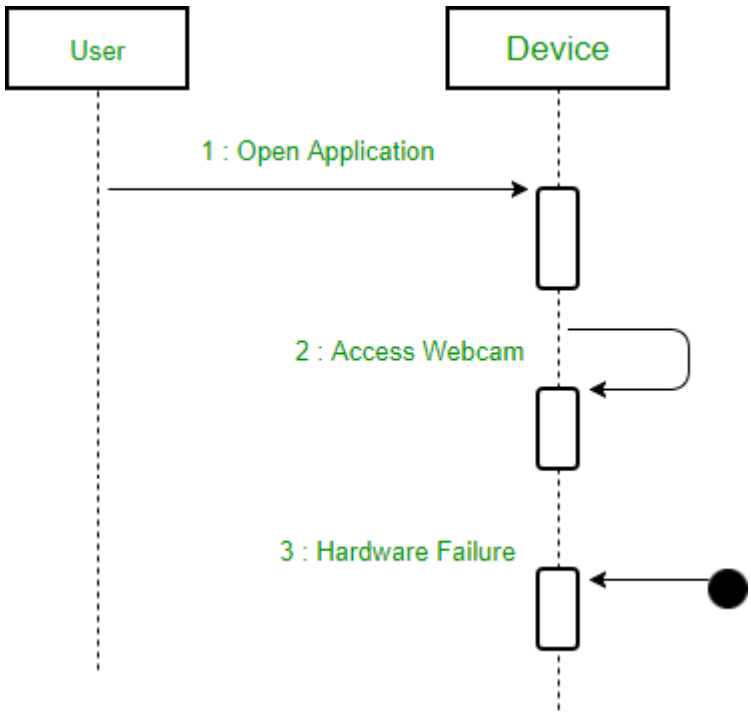


Figure – a scenario where found message is used

- **Lost Message –** A Lost message is used to represent a scenario where the recipient is not known to the system. It is represented using an arrow directed towards an end point from a lifeline. For example: Consider a scenario where a warning is generated.

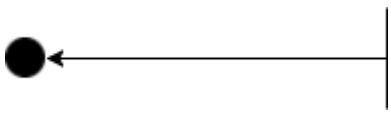


Figure – lost message

The warning might be generated for the user or other software/object that the lifeline is interracting with. Since the destination is not known before hand, we use the Lost Message symbol.

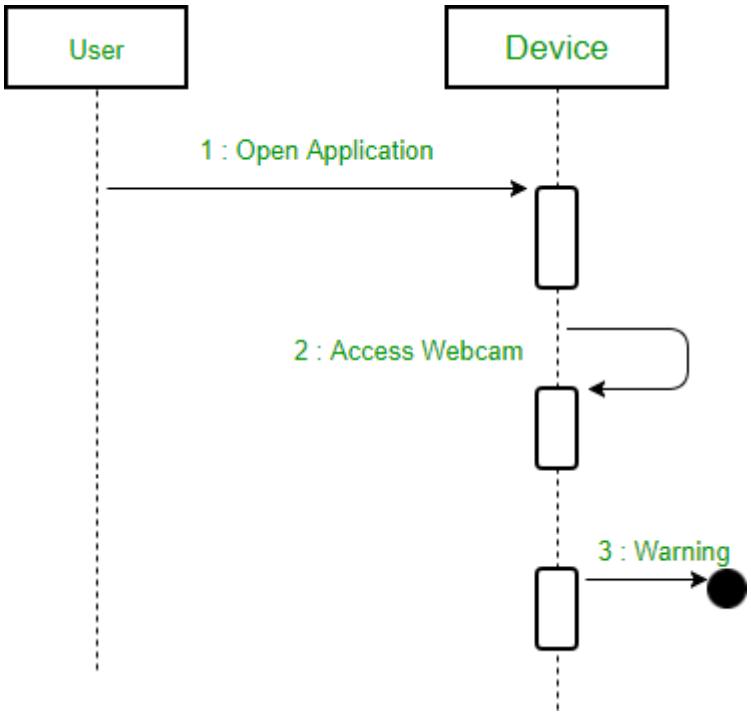


Figure – a scenario where lost message is used

4. **Guards –** To model conditions we use guards in UML. They are used when we need to restrict the flow of messages on the pretext of a condition being met. Guards play an important role in letting software developers know the constraints attached to a system or a particular process.
   For example: In order to be able to withdraw cash, having a balance greater than zero is a condition that must be met as shown below.

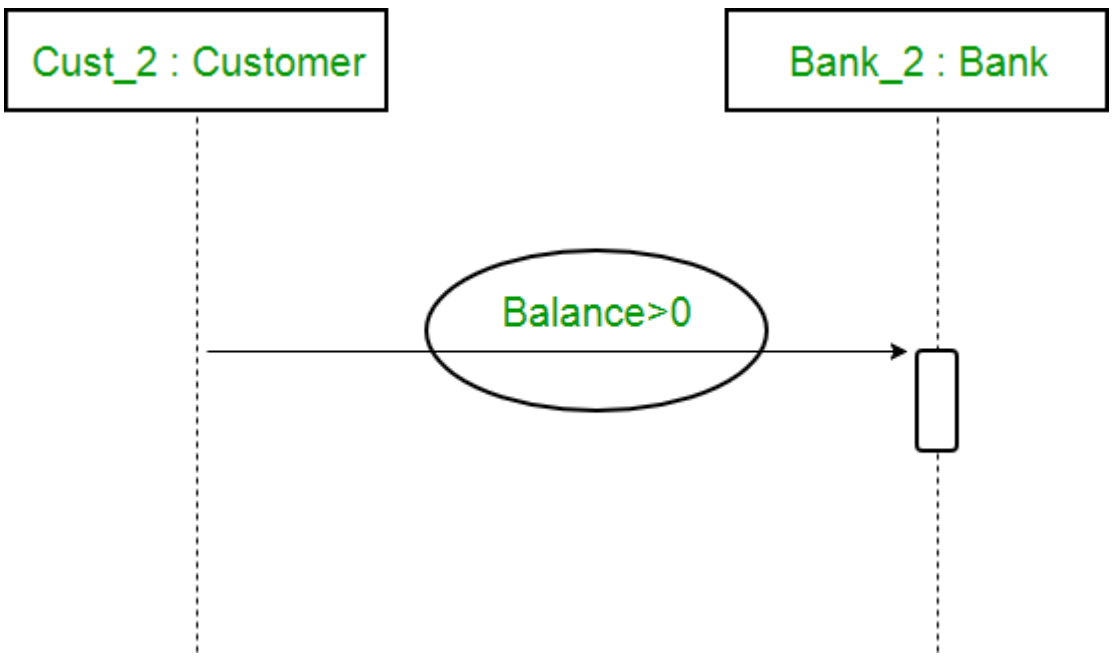


Figure – sequence diagram using a guard

A sequence diagram for an emotion based music player –

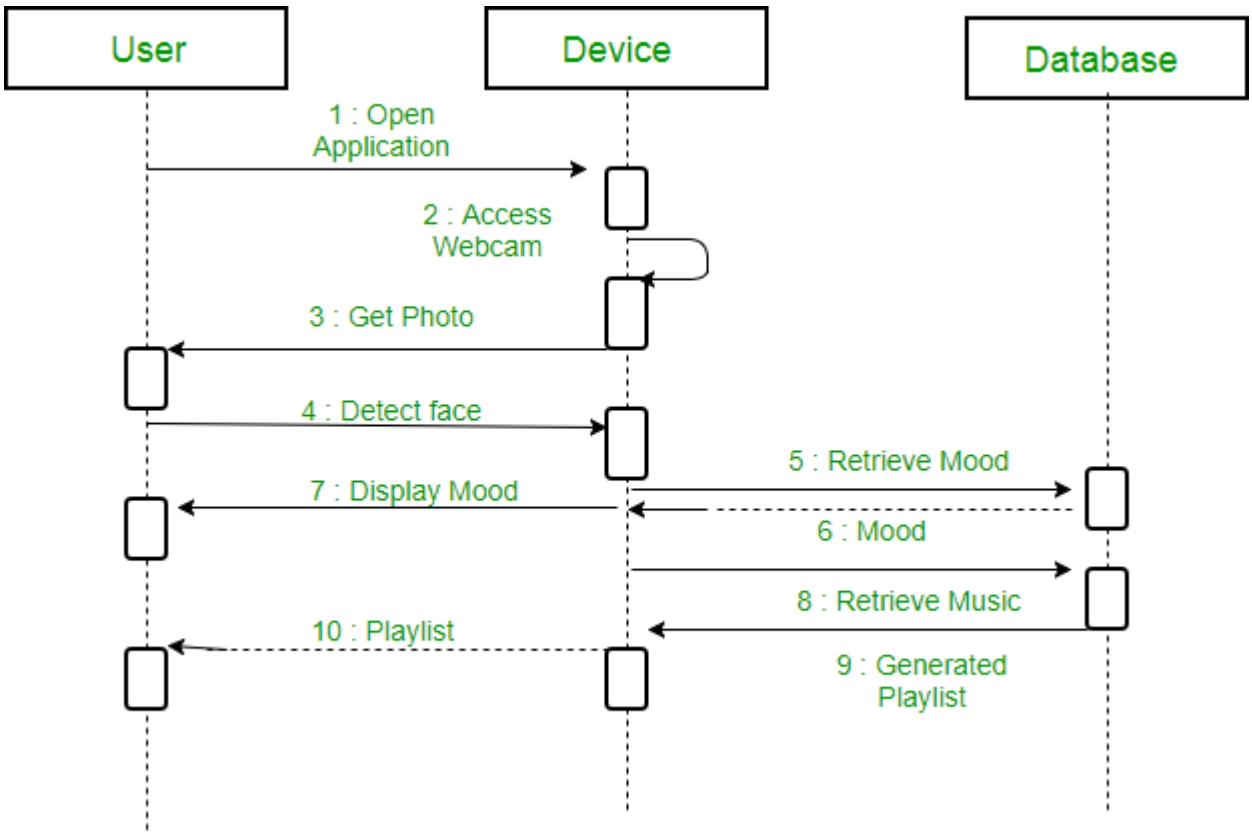


Figure – a sequence diagram for an emotion based music player

The above sequence diagram depicts the sequence diagram for an emotion based music player:

1. Firstly the application is opened by the user.
2. The device then gets access to the web cam.
3. The webcam captures the image of the user.
4. The device uses algorithms to detect the face and predict the mood.
5. It then requests database for dictionary of possible moods.
6. The mood is retrieved from the database.
7. The mood is displayed to the user.
8. The music is requested from the database.
9. The playlist is generated and finally shown to the user.

Uses of sequence diagrams –

- Used to model and visualise the logic behind a sophisticated function, operation or procedure.
- They are also used to show details of UML use case diagrams.
- Used to understand the detailed functionality of current or future systems.
- Visualise how messages and tasks move between objects or components in a system.

UML Use Case Diagram

A use case diagram is used to represent the dynamic behavior of a system. It encapsulates the system's functionality by incorporating use cases, actors, and their relationships. It models the tasks, services, and functions required by a system/subsystem of an application. It depicts the high-level functionality of a system and also tells how the user handles a system.

Purpose of Use Case Diagrams

The main purpose of a use case diagram is to portray the dynamic aspect of a system. It accumulates the system's requirement, which includes both internal as well as external influences. It invokes persons, use cases, and several things that invoke the actors and elements accountable for the implementation of use case diagrams. It represents how an entity from the external environment can interact with a part of the system.

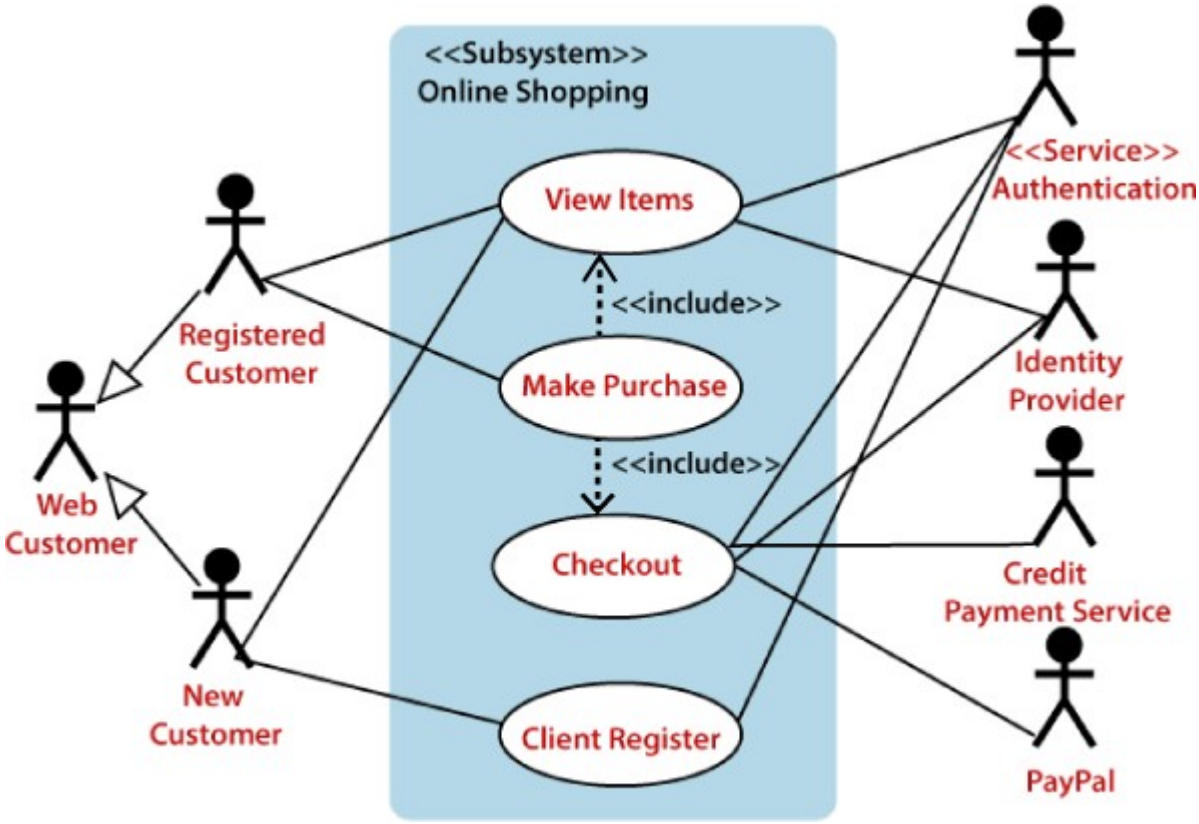
Following are the purposes of a use case diagram given below:

1. It gathers the system's needs.
2. It depicts the external view of the system.
3. It recognizes the internal as well as external factors that influence the system.
4. It represents the interaction between the actors.

Example of a Use Case Diagram

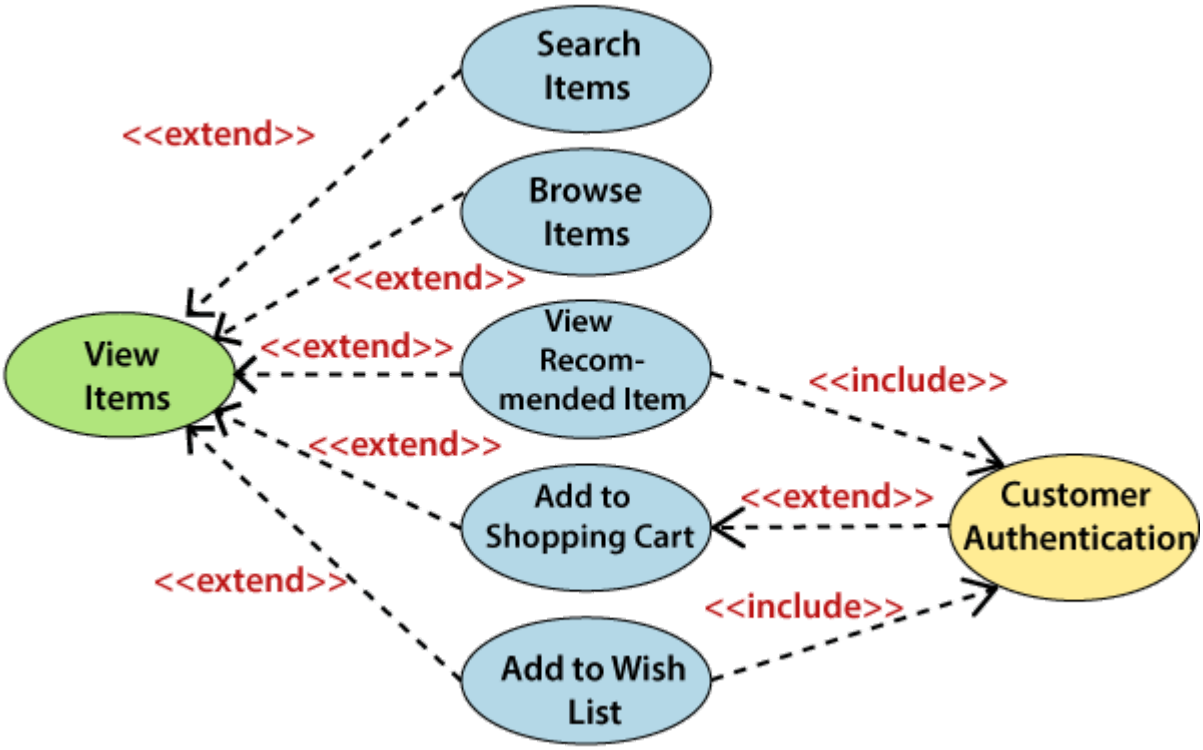
A use case diagram depicting the Online Shopping website is given below.

Here the Web Customer actor makes use of any online shopping website to purchase online. The top-level uses are as follows; View Items, Make Purchase, Checkout, Client Register. The **View Items** use case is utilized by the customer who searches and view products. The **Client Register** use case allows the customer to register itself with the website for availing gift vouchers, coupons, or getting a private sale invitation. It is to be noted that the **Checkout** is an included use case, which is part of **Making Purchase,** and it is not available by itself.



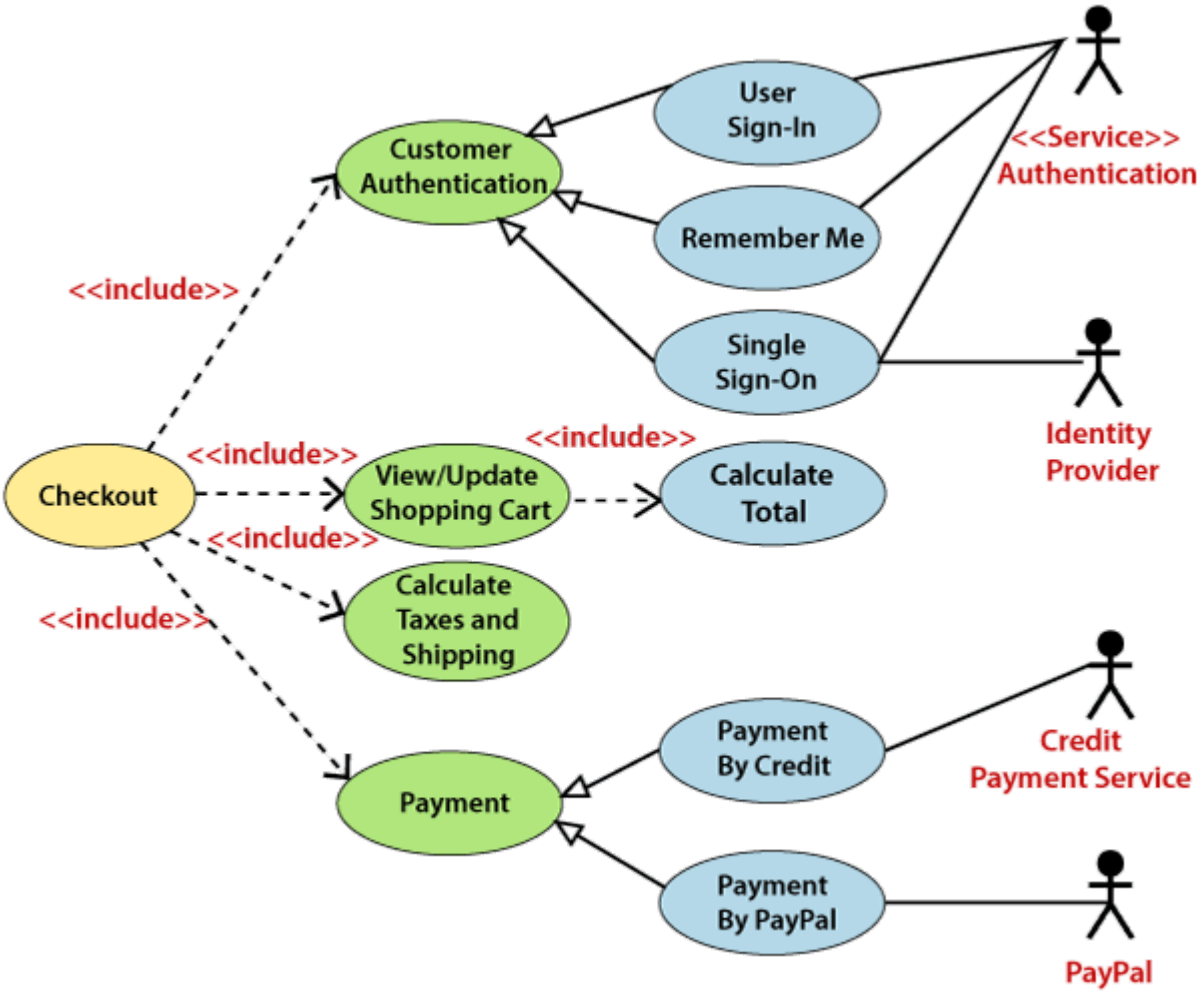
The **View Items** is further extended by several use cases such as; Search Items, Browse Items, View Recommended Items, Add to Shopping Cart, Add to Wish list. All of these extended use cases provide some functions to customers, which allows them to search for an item. The View Items is further extended by several use cases such as; Search Items, Browse Items, View Recommended Items, Add to Shopping Cart, Add to Wish list. All of these extended use cases provide some functions to customers, which allows them to search for an item.

Both **View Recommended Item** and **Add to Wish List** include the Customer Authentication use case, as they necessitate authenticated customers, and simultaneously item can be added to the shopping cart without any user authentication.

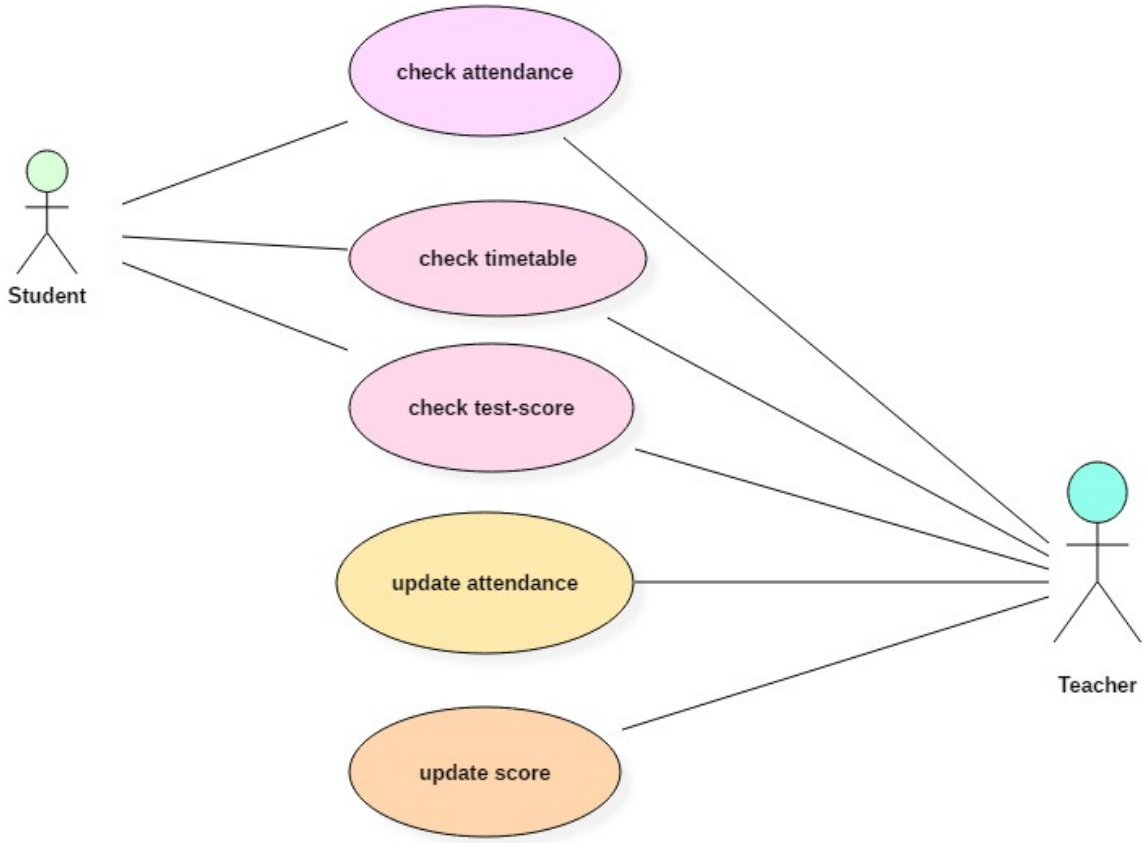


Similarly, the **Checkout** use case also includes the following use cases, as shown below. It requires an authenticated Web Customer, which can be done by login page, user authentication cookie ("Remember me"), or Single Sign-On (SSO). SSO needs an external identity provider's participation, while Web site authentication service is utilized in all these use cases.

The Checkout use case involves Payment use case that can be done either by the credit card and external credit payment services or with PayPal.



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



Activity Diagram

An activity diagram portrays the control flow from a start point to a finish point showing the various decision paths that exist while the activity is being executed. We can depict both sequential processing and concurrent processing of activities using an activity diagram. They are used in business and process modelling where their primary use is to depict the dynamic aspects of a system.

An activity diagram is very **similar to a flowchart**. So let us understand if an activity diagrams or a flowcharts are any different :

Difference between an Activity diagram and a Flowchart –

Flowcharts were typically invented earlier than activity diagrams. Non programmers use Flow charts to model workflows. For example: A manufacturer uses a flow chart to explain and illustrate how a particular product is manufactured. We can call a flowchart a primitive version of an activity diagram. Business processes where decision making is involved is expressed using a flow chart.

So, programmers use activity diagrams (advanced version of a flowchart) to depict workflows. An activity diagram is **used by developers** to understand the flow of programs on a high level. It also enables them to figure out constraints and conditions that cause particular

events. A flow chart converges into being an activity diagram if complex decisions are being made.
So an activity diagram helps people on both sides i.e. Businessmen and Developers to interact and understand systems

.**Difference between a Use case diagram and an Activity diagram**

An activity diagram is used to model the workflow depicting conditions, constraints, sequential and concurrent activities. On the other hand, the purpose of a Use Case is to just depict the functionality i.e. what the system does and not how it is done. So in simple terms, an activity diagram shows 'How' while a Use case shows 'What' for a particular system.
The levels of abstraction also vary for both of them. An activity diagram can be used to illustrate a business process (high level implementation) to a stand alone algorithm (ground level implementation). However, Use cases have a low level of abstraction. They are used to show a **high level** of implementation only.

The various components used in the diagram and the standard notations are explained below.

Activity Diagram Notations –

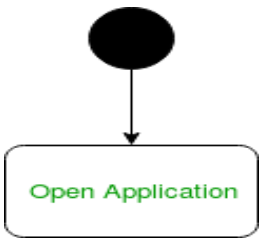
1. **Initial State –** The starting state before an activity takes place is depicted using the initial state.

   notation for initial state or start state

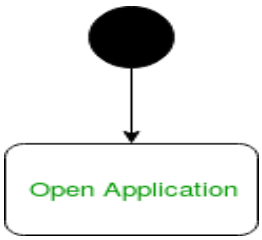
   A process can have only one initial state unless we are depicting nested activities. We use a black filled circle to depict the initial state of a system. For objects, this is the state when they are instantiated. The Initial State from the UML Activity Diagram marks the entry point and the initial Activity State.

   For example – Here the initial state is the state of the system before the application is opened.

2. **Action or Activity State –** An activity represents execution of an action on objects or by objects. We represent an activity using a rectangle with rounded corners. Basically any action or event that takes place is represented using an activity.

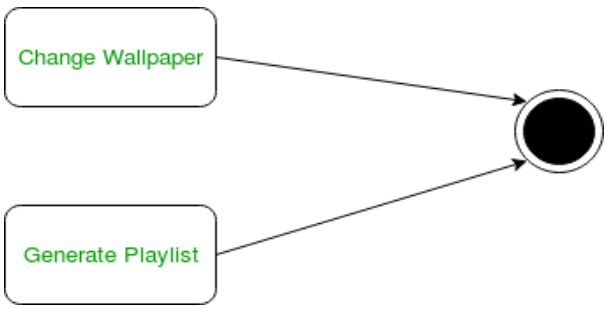
Figure – notation for an activity state
For example – Consider the previous example of opening an application opening the application is an activity state in the activity diagram.



3. **Action Flow or Control flows –** Action flows or Control flows are also referred to as paths and edges. They are used to show the transition from one activity state to another.

Figure – notation for control Flow
An activity state can have multiple incoming and outgoing action flows. We use a line with an arrow head to depict a Control Flow. If there is a constraint to be adhered to while making the transition it is mentioned on the arrow.

Consider the example – Here both the states transit into one final state using action flow symbols i.e. arrows.



4. **Decision node and Branching –** When we need to make a decision before deciding the flow of control, we use the decision node.

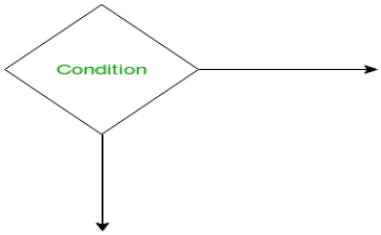
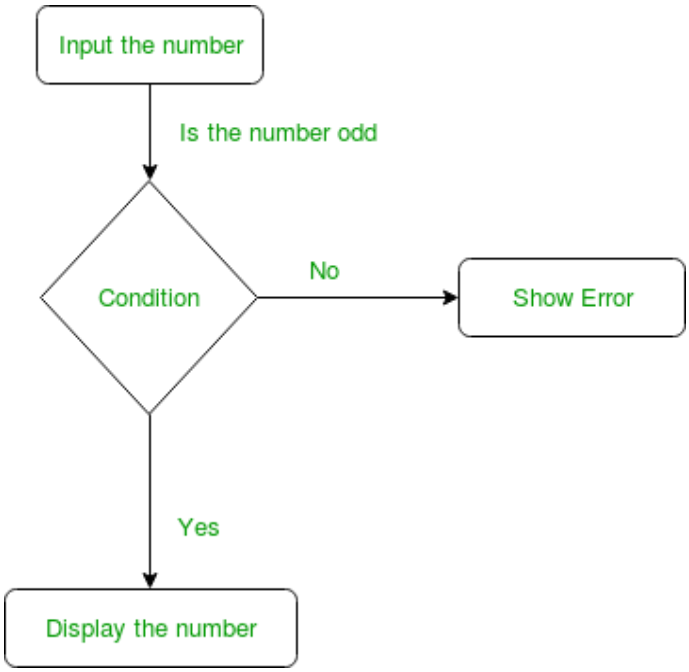
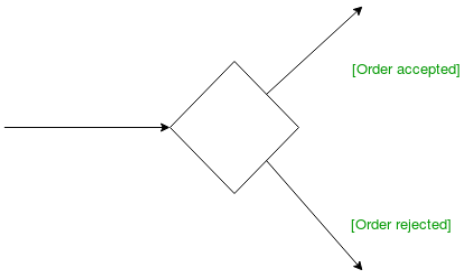


Figure – notation for decision node
The outgoing arrows from the decision node can be labelled with conditions or guard expressions.It always includes two or more output arrows.

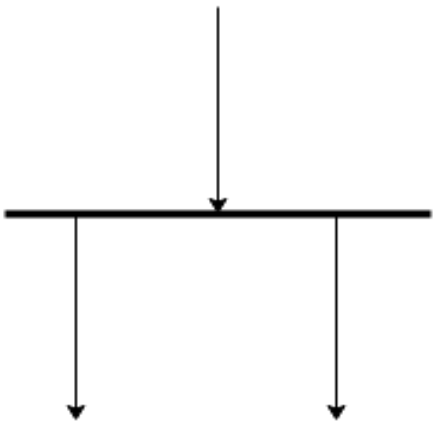


5. **Guards –** A Guard refers to a statement written next to a decision node on an arrow sometimes within square brackets.



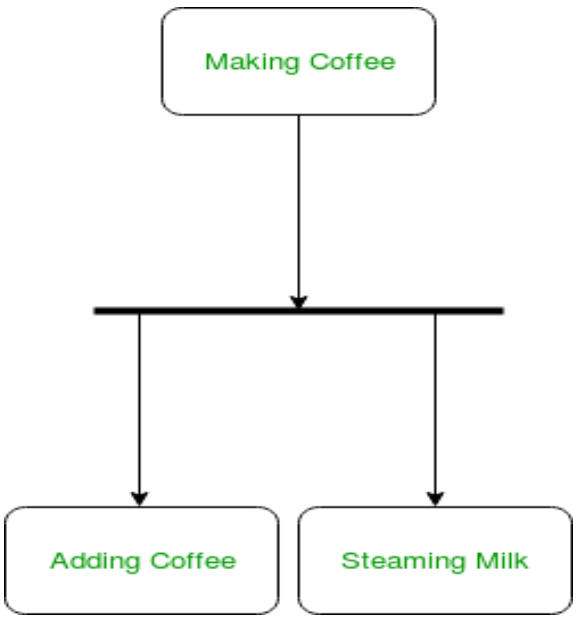
The statement must be true for the control to shift along a particular direction. Guards help us know the constraints and conditions which determine the flow of a process.

6. **Fork –** Fork nodes are used to support concurrent activities.



fork notation

When we use a fork node when both the activities get executed concurrently i.e. no decision is made before splitting the activity into two parts. Both parts need to be executed in case of a fork statement.
We use a rounded solid rectangular bar to represent a Fork notation with incoming arrow from the parent activity state and outgoing arrows towards the newly created activities.
For example: In the example below, the activity of making coffee can be split into two concurrent activities and hence we use the fork notation.



7. **Join –** Join nodes are used to support concurrent activities converging into one. For join notations we have two or more incoming edges and one outgoing edge.

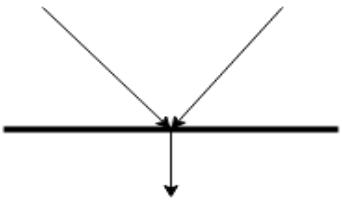
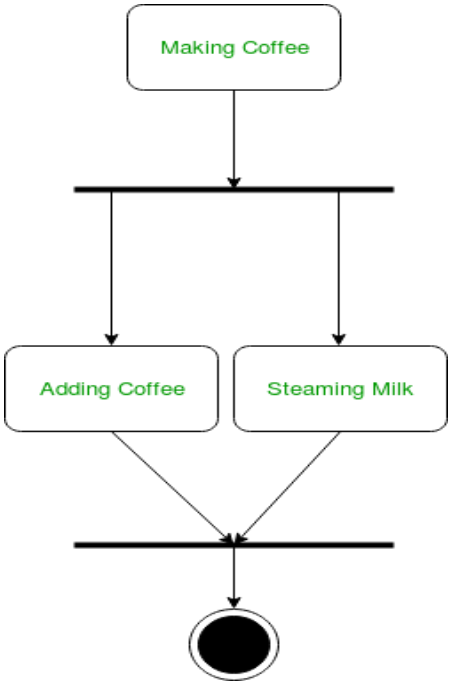


Figure – join notation

For example – When both activities i.e. steaming the milk and adding coffee get completed, we converge them into one final activity.



8. **Merge or Merge Event –** Scenarios arise when activities which are not being executed concurrently have to be merged. We use the merge notation for such scenarios. We can merge two or more activities into one if the control proceeds onto the next activity irrespective of the path chosen.

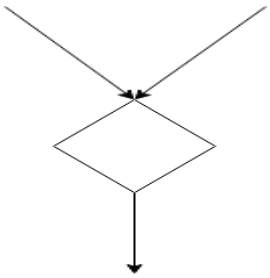
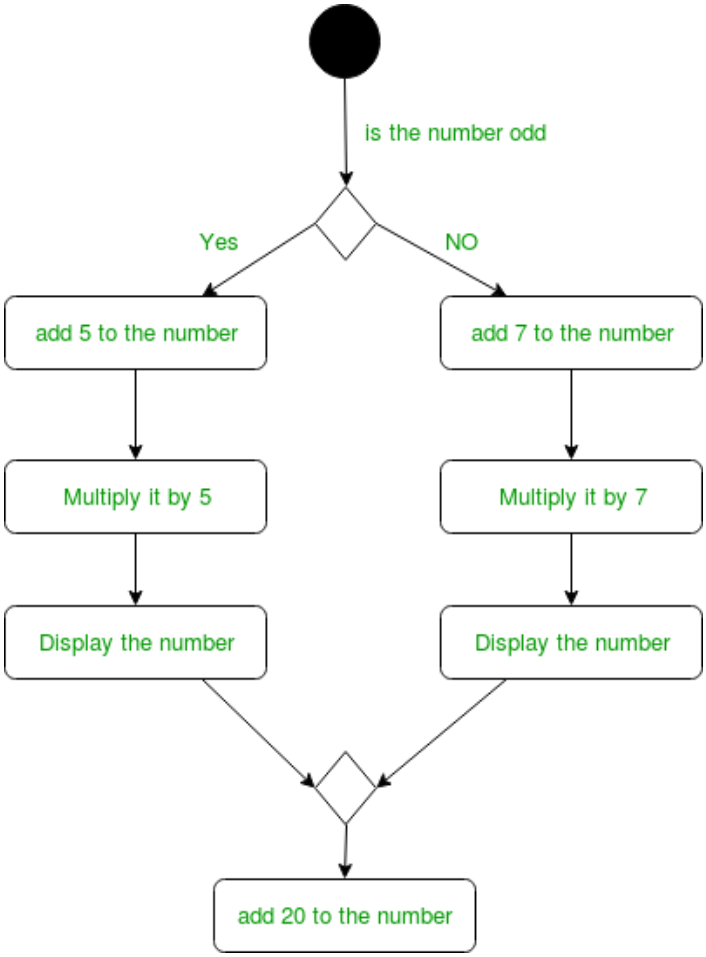


Figure – merge notation

For example – In the diagram below: we can't have both sides executing concurrently, but they finally merge into one. A number can't be both odd and even at the same time.



an activity diagram using merge notation

9. **Swimlanes –** We use swimlanes for grouping related activities in one column. Swimlanes group related activities into one column or one row. Swimlanes can be vertical and horizontal. Swimlanes are used to add modularity to the activity diagram. It is not mandatory to use swimlanes. They usually give more clarity to the activity diagram. It's similar to creating a function in a program. It's not mandatory to do so, but, it is a recommended practice.

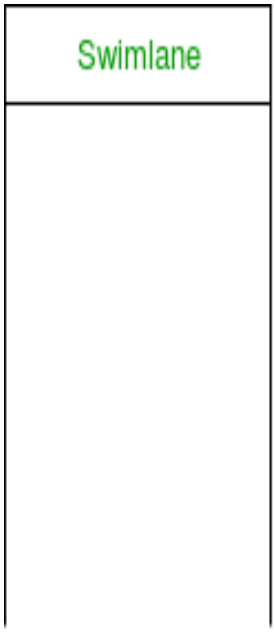
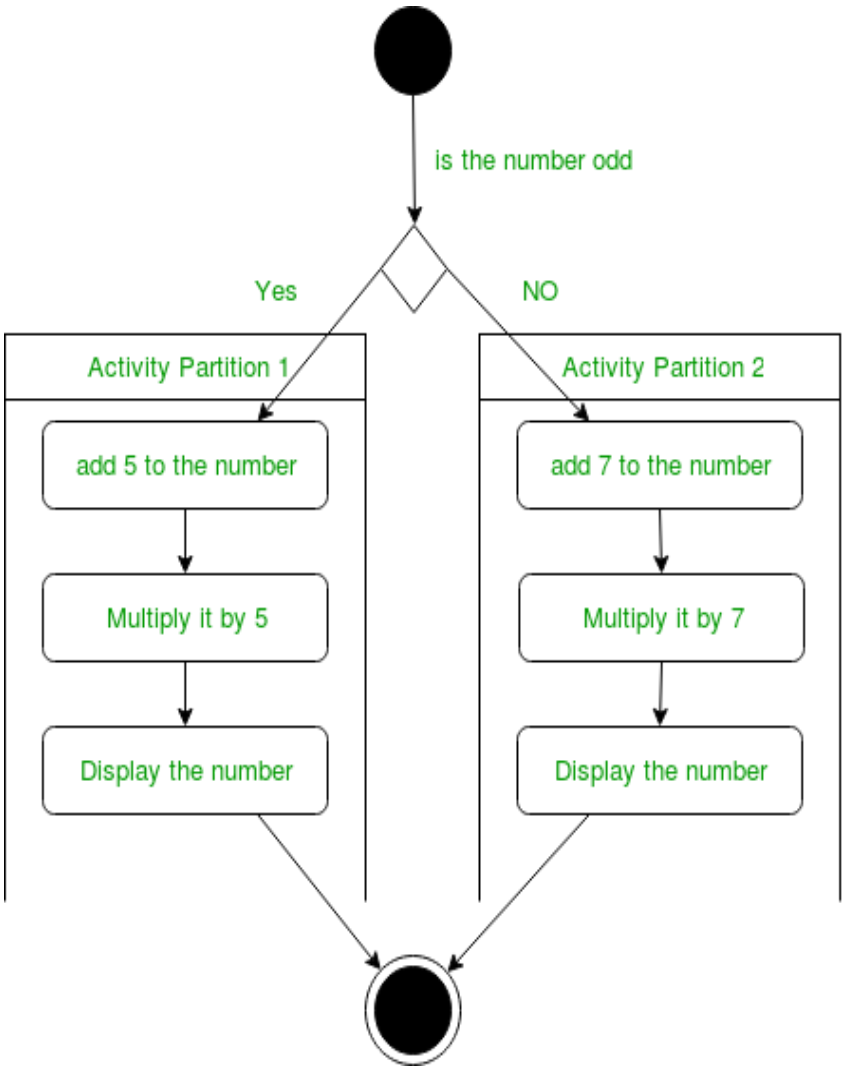


Figure – swimlanes notation

We use a rectangular column to represent a swimlane as shown in the figure above.

For example – Here different set of activities are executed based on if the number is odd or even. These activities are grouped into a swimlane.



an activity diagram making use of swimlanes

10. **Time Event –**

Figure – time event notation

We can have a scenario where an event takes some time to complete. We use an hourglass to represent a time event.
For example – Let us assume that the processing of an image takes takes a lot of time. Then it can be represented as shown below.

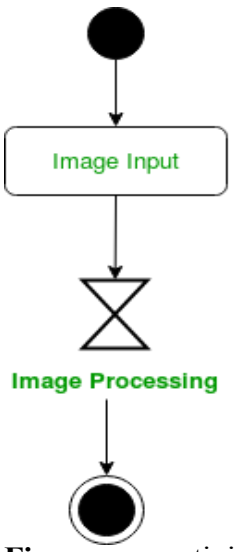


Figure – an activity diagram using time event

11. **Final State or End State –** The state which the system reaches when a particular process or activity ends is known as a Final State or End State. We use a filled circle within a circle notation to represent the final state in a state machine diagram. A system or a process can have multiple final states.

Figure – notation for final state

How to Draw an activity diagram –

1. Identify the initial state and the final states.
2. Identify the intermediate activities needed to reach the final state from he initial state.
3. Identify the conditions or constraints which cause the system to change control flow.
4. Draw the diagram with appropriate notations.

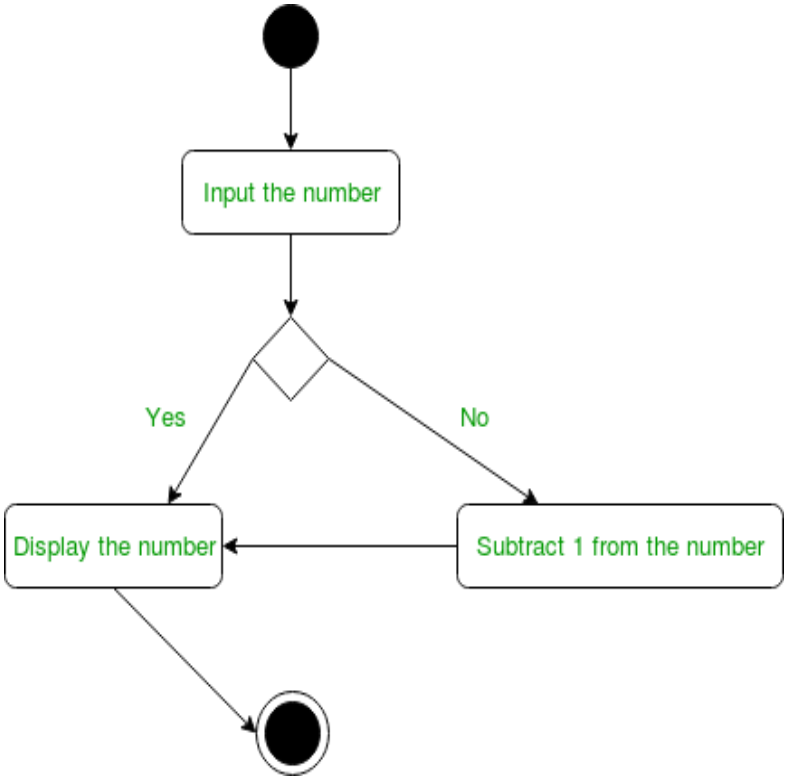


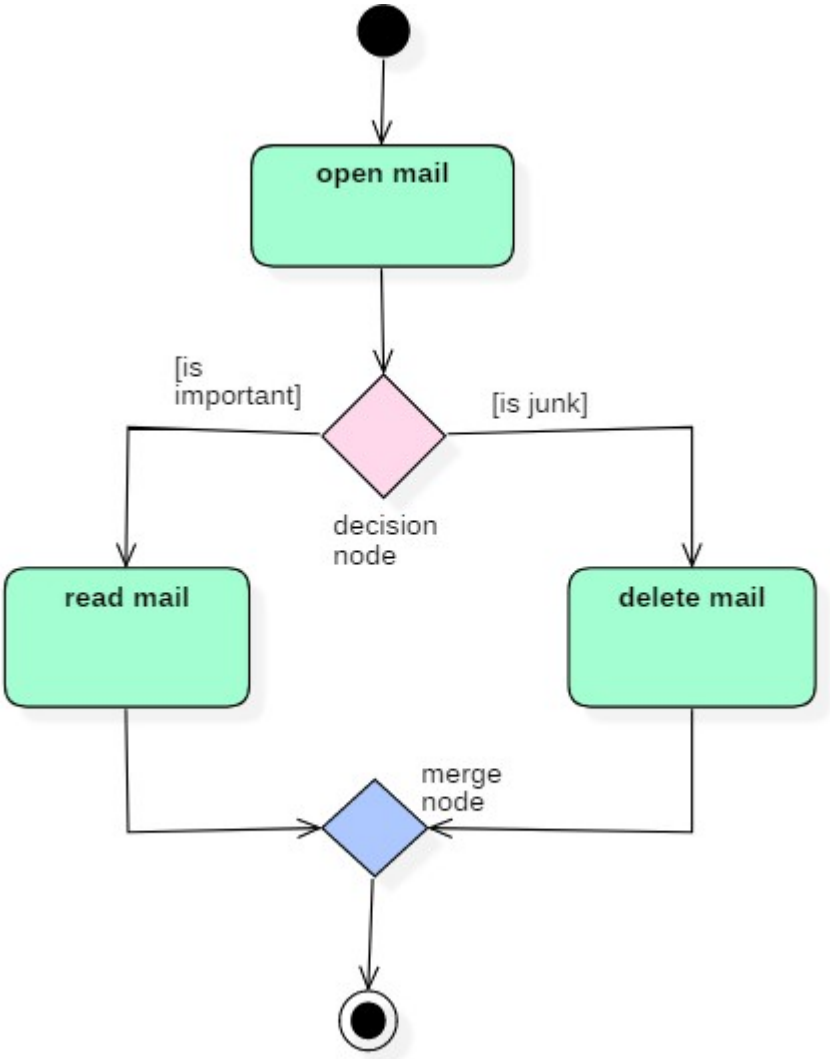
Figure – an activity diagram

The above diagram prints the number if it is odd otherwise it subtracts one from the number and displays it.

Uses of an Activity Diagram –

- Dynamic modelling of the system or a process.
- Illustrate the various steps involved in a UML use case.
- Model software elements like methods,operations and functions.
- We can use Activity diagrams to depict concurrent activities easily.
- Show the constraints, conditions and logic behind algorithms.

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


open mail
[is important]
[is junk]
decision node
read mail
delete mail
merge node