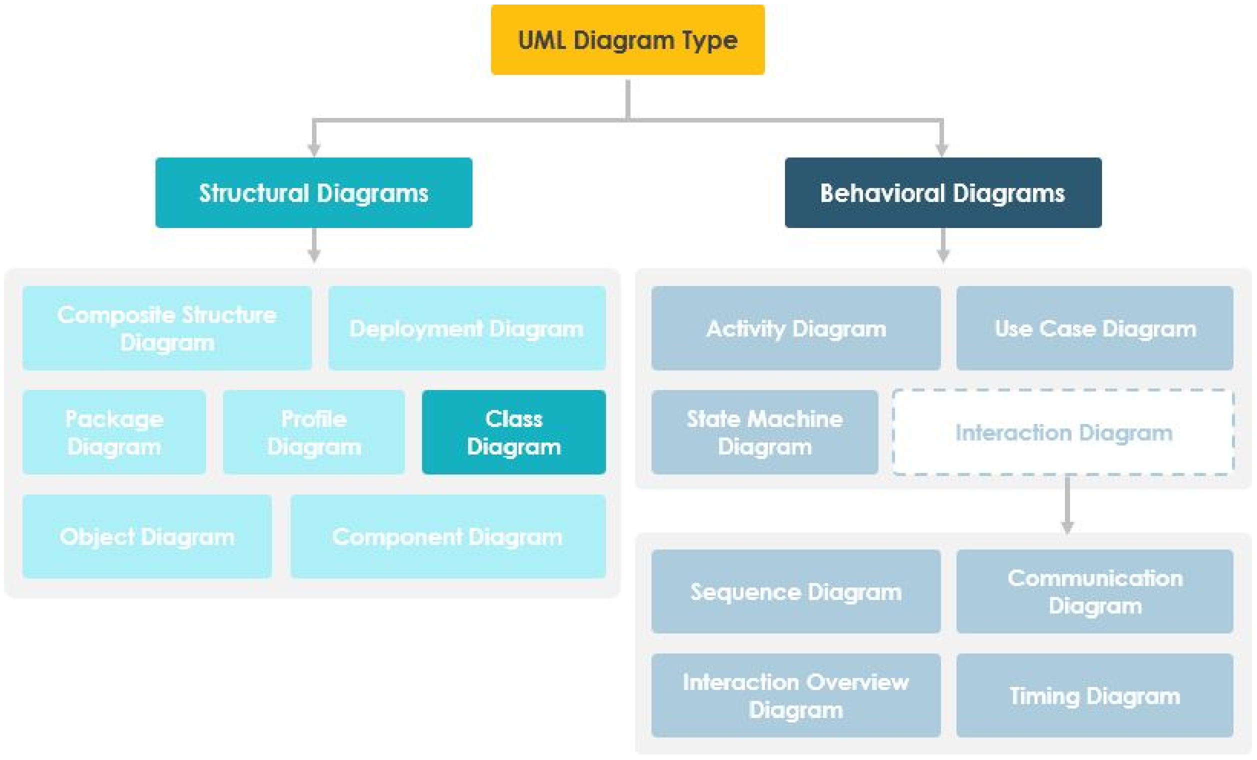


A UML class diagram is made up of:

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

What is a Class

A description of a group of objects all with similar roles in the system, which consists of:

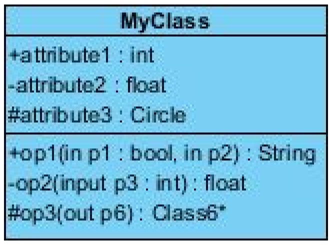
- Structural features (attributes) define what objects of the class "know"
    - Represent the state of an object of the class
    - Are descriptions of the structural or static features of a class
- Behavioral features (operations) define what objects of the class "can do"
    - Define the way in which objects may interact
    - Operations are descriptions of behavioral or dynamic features of a class

Class Notation

A class notation consists of three parts:

1. Class Name
    - The name of the class appears in the first partition.
2. Class Attributes
    - Attributes are shown in the second partition.
    - The attribute type is shown after the colon.
    - Attributes map onto member variables (data members) in code.
3. Class Operations (Methods)
    - Operations are shown in the third partition. They are services the class provides.

- The return type of a method is shown after the colon at the end of the method signature.
- The return type of method parameters are shown after the colon following the parameter name.
- Operations map onto class methods in code

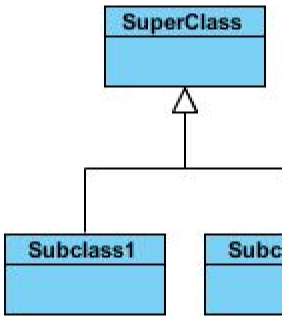
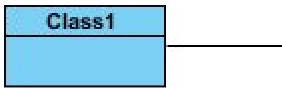


The graphical representation of the class - MyClass as shown above:

- MyClass has 3 attributes and 3 operations
- Parameter p3 of op2 is of type int
- op2 returns a float
- op3 returns a pointer (denoted by a *) to Class6

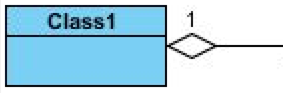
Class Relationships

A class may be involved in one or more relationships with other classes. A relationship can be one of the following types: (Refer to the figure on the right for the graphical representation of relationships).

Relationship Type	**Graphical Represen**
Inheritance (or Generalization): • Represents an "is-a" relationship. • An abstract class name is shown in italics. • SubClass1 and SubClass2 are specializations of Super Class. • A solid line with a hollow arrowhead that point from the child to the parent class	
Simple Association: • A structural link between two peer classes. • There is an association between Class1 and Class2 • A solid line connecting two classes	

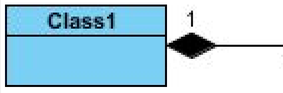
Aggregation:
A special type of association. It represents a "part of" relationship.

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



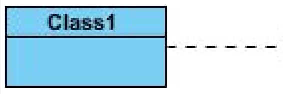
Composition:
A special type of aggregation where parts are destroyed when the whole is destroyed.

- Objects of Class2 live and die with Class1.
- Class2 cannot stand by itself.
- A solid line with a filled diamond at the association connected to the class of composite



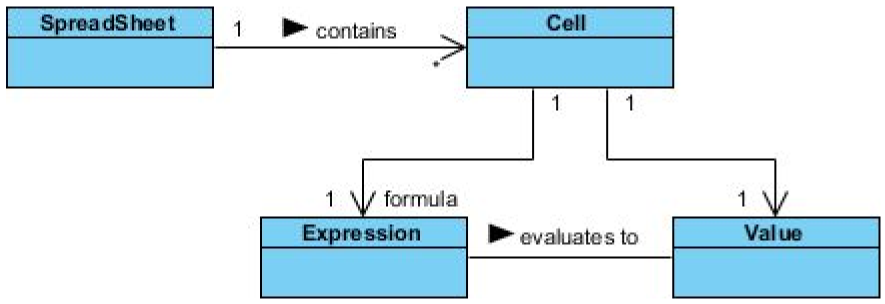
Dependency:

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



Relationship Names

- Names of relationships are written in the middle of the association line.
- Good relation names make sense when you read them out loud:
  - "Every spreadsheet contains some number of cells",
  - "an expression evaluates to a value"
- They often have a small arrowhead to show the direction in which direction to read the relationship, e.g., expressions evaluate to values, but values do not evaluate to expressions.



Relationship - Roles

- A role is a directional purpose of an association.
- Roles are written at the ends of an association line and describe the purpose played by that class in the relationship.
    - E.g., A cell is related to an expression. The nature of the relationship is that the expression is the formula of the cell.

Navigability

The arrows indicate whether, given one instance participating in a relationship, it is possible to determine the instances of the other class that are related to it.
The diagram above suggests that,

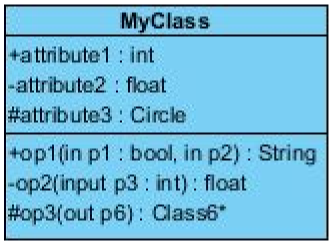
- Given a spreadsheet, we can locate all of the cells that it contains, but that
    - we cannot determine from a cell in what spreadsheet it is contained.
- Given a cell, we can obtain the related expression and value, but
    - given a value (or expression) we cannot find the cell of which those are attributes.

Visibility of Class attributes and Operations

In object-oriented design, there is a notation of visibility for attributes and operations. UML identifies four types of visibility: public, protected, private, and package.
The +, -, # and ~ symbols before an attribute and operation name in a class denote the visibility of the attribute and operation.

- + denotes public attributes or operations
- - denotes private attributes or operations
- # denotes protected attributes or operations
- ~ denotes package attributes or operations

Class Visibility Example



In the example above:

🎬 attribute1 and op1 of MyClassName are public

- attribute3 and op3 are protected.
- attribute2 and op2 are private.

Access for each of these visibility types is shown below for members of different classes.

Access Right	public (+)	private (-)	protected (#)	Package (
Members of the same class	yes	yes	yes	yes
Members of derived classes	yes	no	yes	yes
Members of any other class	yes	no	no	in same p

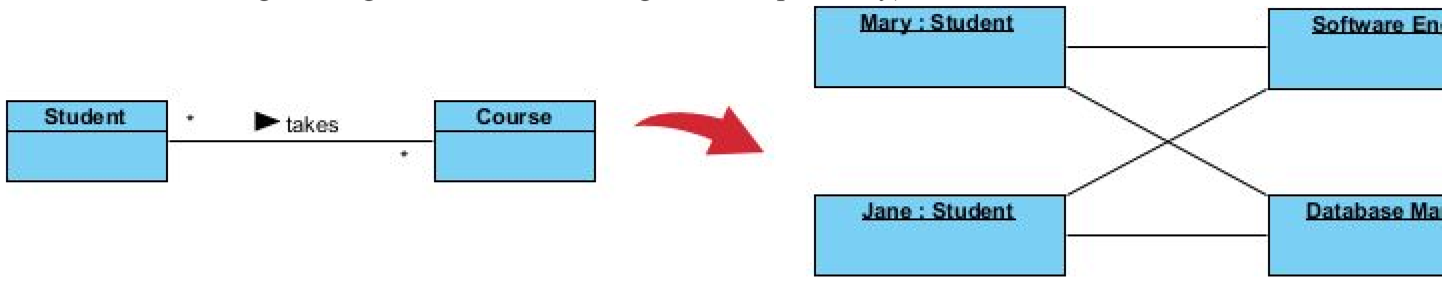
Multiplicity

How many objects of each class take part in the relationships and multiplicity can be expressed as:

- Exactly one - 1
- Zero or one - 0..1
- Many - 0..* or *
- One or more - 1..*
- Exact Number - e.g. 3..4 or 6
- Or a complex relationship - e.g. 0..1, 3..4, 6.* would mean any number of objects other than 2 or 5

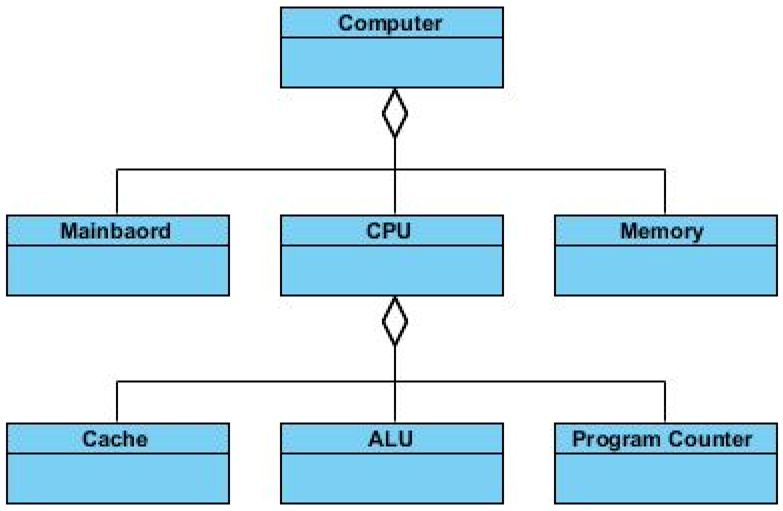
Multiplicity Example

- Requirement: A Student can take many Courses and many Students can be enrolled in one Course.
- In the example below, the class diagram (on the left), describes the statement of the requirement above for the static model while the object diagram (on the right) shows the snapshot (an instance of the class diagram) of the course enrollment for the courses Software Engineering and Database Management respectively)



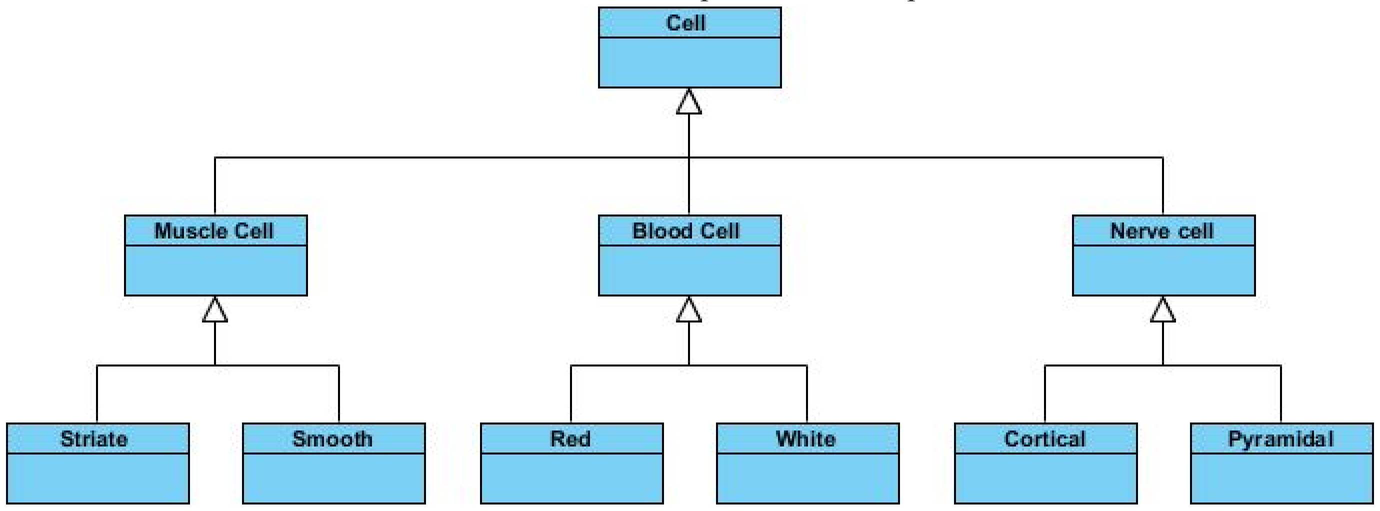
Aggregation Example - Computer and parts

- An aggregation is a special case of association denoting a "consists-of" hierarchy
- The aggregate is the parent class, the components are the children classes



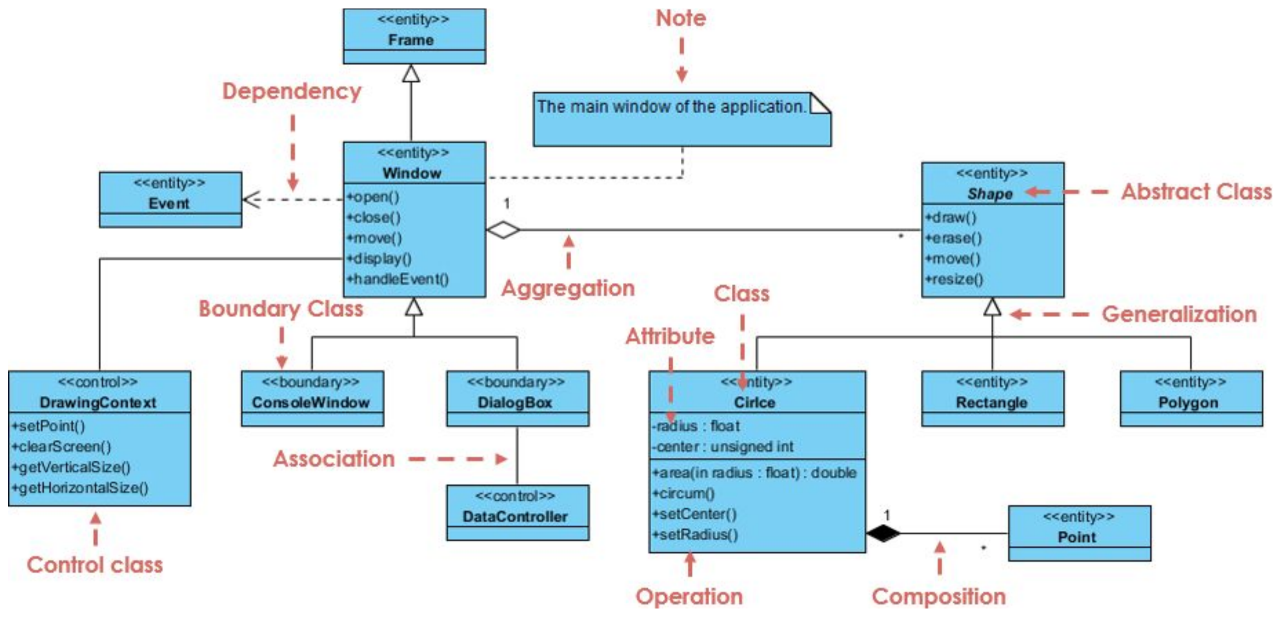
Inheritance Example - Cell Taxonomy

- Inheritance is another special case of an association denoting a "kind-of" hierarchy
- Inheritance simplifies the analysis model by introducing a taxonomy
- The children classes inherit the attributes and operations of the parent class.



Class Diagram - Diagram Tool Example

A class diagram may also have notes attached to classes or relationships. Notes are shown in grey.



In the example above:

We can interpret the meaning of the above class diagram by reading through the points as following.

1. Shape is an abstract class. It is shown in Italics.
2. Shape is a superclass. Circle, Rectangle and Polygon are derived from Shape. In other words, a Circle is-a Shape. This is a generalization / inheritance relationship.
3. There is an association between DialogBox and DataController.
4. Shape is part-of Window. This is an aggregation relationship. Shape can exist without Window.
5. Point is part-of Circle. This is a composition relationship. Point cannot exist without a Circle.
6. Window is dependent on Event. However, Event is not dependent on Window.
7. The attributes of Circle are radius and center. This is an entity class.
8. The method names of Circle are area(), circum(), setCenter() and setRadius().
9. The parameter radius in Circle is an in parameter of type float.
10. The method area() of class Circle returns a value of type double.
11. The attributes and method names of Rectangle are hidden. Some other classes in the diagram also have their attributes and method names hidden.

Dealing with Complex System - Multiple or Single Class Diagram?

Inevitably, if you are modeling a large system or a large business area, there will be numerous entities you must consider. Should we use multiple or a single class diagram for modeling the problem? The answer is:

- Instead of modeling every entity and its relationships on a single class diagram, it is better to use multiple class diagrams.
- Dividing a system into multiple class diagrams makes the system easier to understand, especially if each diagram is a graphical representation of a specific part of the system.

Perspectives of Class Diagram in Software Development Lifecycle

We can use class diagrams in different development phases of a software development lifecycle and typically by modeling class diagrams in three different perspectives (levels of detail) progressively as we move forward:
Conceptual perspective: The diagrams are interpreted as describing things in the real world. Thus, if you take the conceptual perspective you draw a diagram that represents the concepts in the domain under study. These concepts will naturally relate to the classes that implement them. The conceptual perspective is considered language-independent.

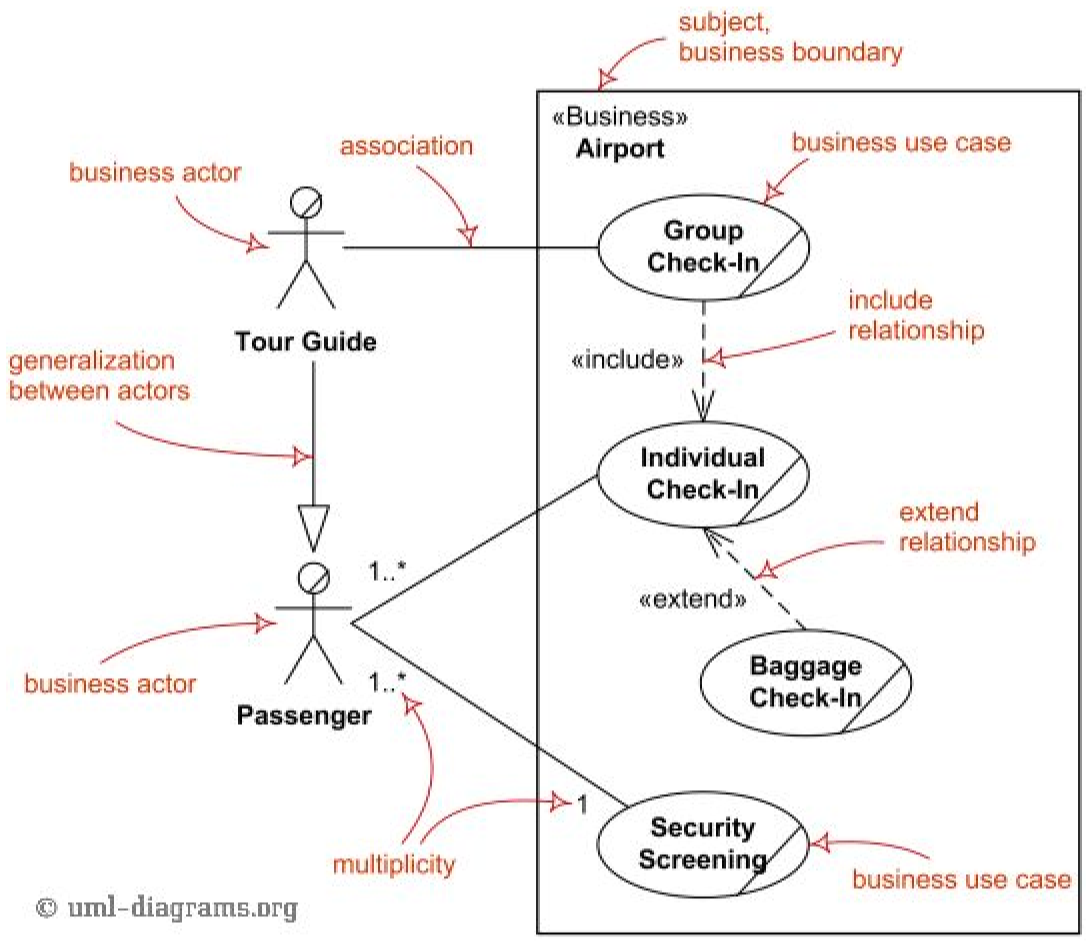
UML Use Case Diagrams

Ref: https://www.uml-diagrams.org/use-case-diagrams.html

Use case diagrams are usually referred to as **behavior diagrams** used to describe a set of actions (**use cases**) that some system or systems (**subject**) should or can perform in collaboration with one or more **external users** of the system (**actors**). Each use case should provide some observable and valuable result to the actors or other stakeholders of the system.
Note, that UML 2.0 to 2.4 specifications also described **use case diagram** as a specialization of a **class diagram**, and class diagram is a **structure diagram**.
Use case diagrams are in fact twofold - they are both **behavior diagrams**, because they describe behavior of the system, and they are also **structure diagrams** - as a special case of class diagrams where classifiers are restricted to be either **actors** or **use cases** related to each other with **associations**.
[UML 2.5 FTF - Beta 1] moved use cases out of behavior modeling to UML **supplementary concepts**. So, it is an unfortunate quandary what kind of UML diagrams use case diagrams are.

Business Use Case Diagrams

While support for **business modeling** was declared as one of the goals of the UML, UML specification provides **no notation** specific to business needs.
Business use cases were introduced in **Rational Unified Process** (RUP) to represent business function, process, or activity performed in the modeled **business**. A **business actor** represents a role played by some person or system external to the modeled business, and interacting with the business. Business use case should produce a result of observable value to a business actor.
Major elements of the business use case diagram are shown on the picture below. Note again, both business use case as well as business actor are not defined in UML standard, so you will either need to use some UML tool supporting those or create your own business modeling stereotypes.



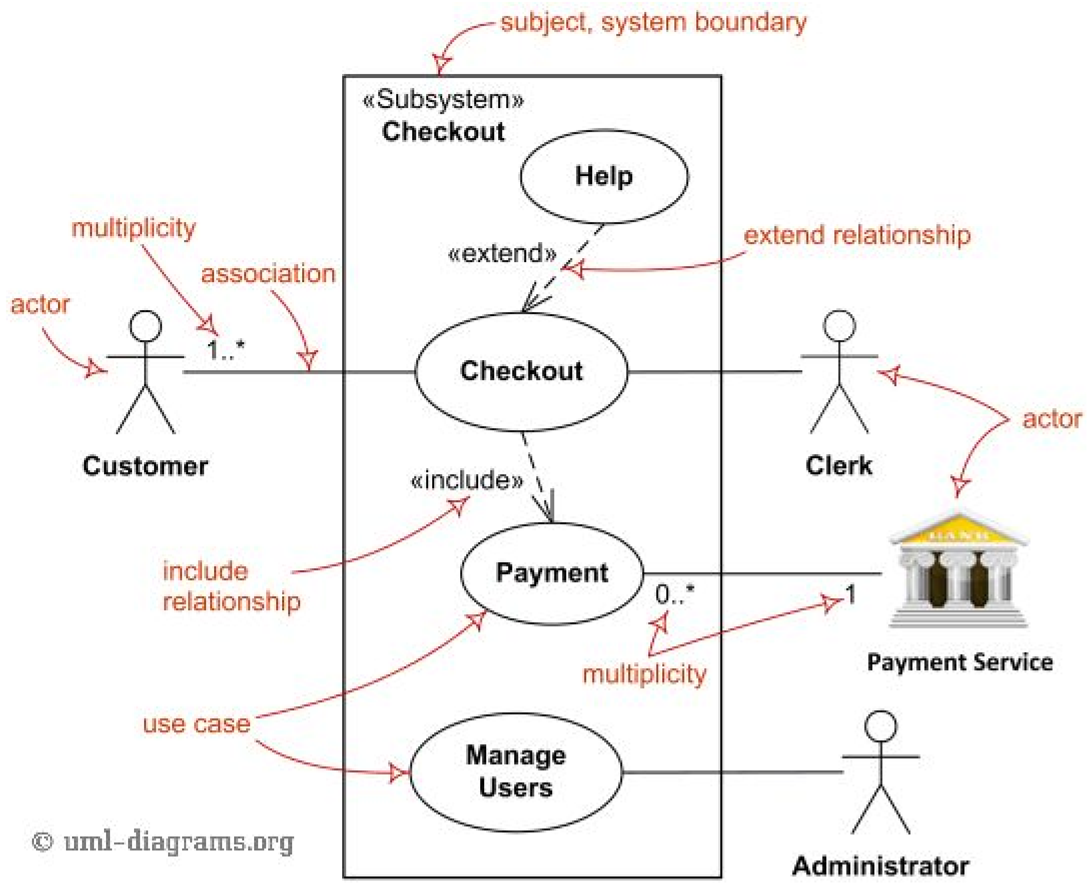
Major elements of business use case diagram - ***business actor*****,** ***business use case*****,**
business boundary**,** ***include*** *and* ***extend*** *relationships.*

System Use Case Diagrams

(System) Use case diagrams are used to specify:

- (external) **requirements**, required usages of a system under design or analysis (**subject**) - to capture what the system is supposed to do;
- the **functionality** offered by a subject – what the system can do;
- requirements the specified subject poses on its **environment** - by defining how environment should interact with the subject so that it will be able to perform its services.

Major elements of the UML use case diagram are shown on the picture below.



*Major elements of UML use case diagram - **actor**, **use case**, **subject**, **include** and **extend** relationships.*

UML - Activity Diagrams

Taken from: https://www.tutorialspoint.com/uml/uml_activity_diagram.htm

Activity diagram is another important diagram in UML to describe the dynamic aspects of the system.

Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system.

The control flow is drawn from one operation to another. This flow can be sequential, branched, or concurrent. Activity diagrams deal with all type of flow control by using different elements such as fork, join, etc

Purpose of Activity Diagrams

The basic purposes of activity diagrams is similar to other four diagrams. It captures the dynamic behavior of the system. Other four diagrams are used to show the message flow from one object to another but activity diagram is used to show message flow from one activity to another.

Activity is a particular operation of the system. Activity diagrams are not only used for visualizing the dynamic nature of a system, but they are also used to construct the executable system by using forward and reverse engineering techniques. The only missing thing in the activity diagram is the message part.

It does not show any message flow from one activity to another. Activity diagram is sometimes considered as the flowchart. Although the diagrams look like a flowchart, they are not. It shows different flows such as parallel, branched, concurrent, and single.

The purpose of an activity diagram can be described as −

- Draw the activity flow of a system.
- Describe the sequence from one activity to another.
- Describe the parallel, branched and concurrent flow of the system.

How to Draw an Activity Diagram?

Activity diagrams are mainly used as a flowchart that consists of activities performed by the system. Activity diagrams are not exactly flowcharts as they have some additional capabilities. These additional capabilities include branching, parallel flow, swimlane, etc.

Before drawing an activity diagram, we must have a clear understanding about the elements used in activity diagram. The main element of an activity diagram is the activity itself. An activity is a function performed by the system. After identifying the activities, we need to understand how they are associated with constraints and conditions.

Before drawing an activity diagram, we should identify the following elements −

- Activities
- Association
- Conditions
- Constraints

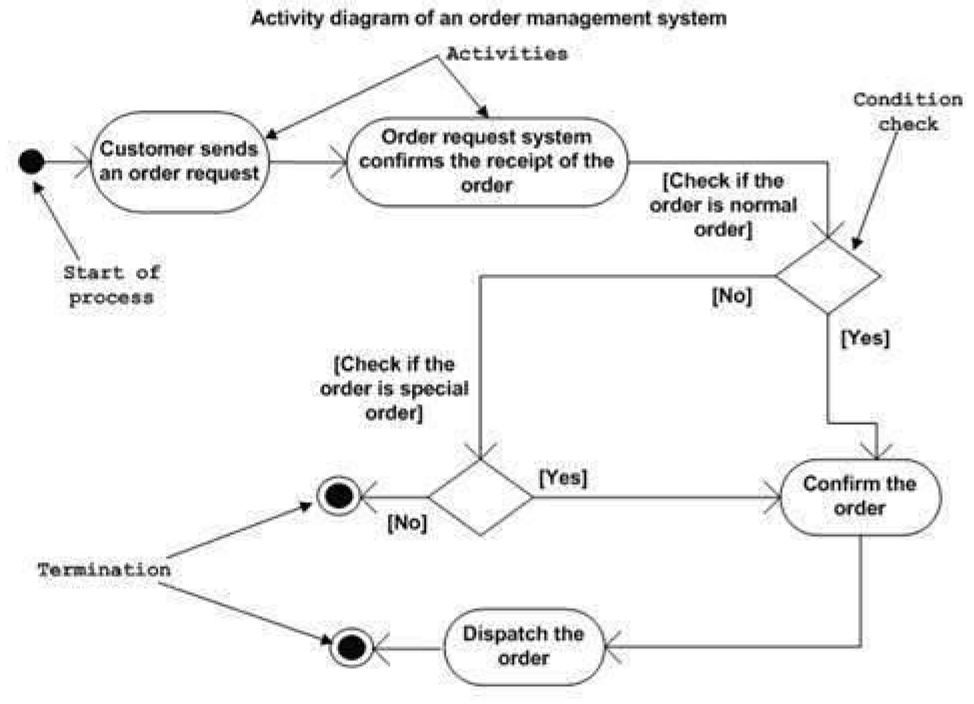
Once the above-mentioned parameters are identified, we need to make a mental layout of the entire flow. This mental layout is then transformed into an activity diagram.

Following is an example of an activity diagram for order management system. In the diagram, four activities are identified which are associated with conditions. One important point should be clearly understood that an activity diagram cannot be exactly matched with the code. The activity diagram is made to understand the flow of activities and is mainly used by the business users

Following diagram is drawn with the four main activities −

- Send order by the customer
- Receipt of the order
- Confirm the order
- Dispatch the order

After receiving the order request, condition checks are performed to check if it is normal or special order. After the type of order is identified, dispatch activity is performed and that is marked as the termination of the process.


Activity diagram of an order management system
Activities
Customer sends an order request
Order request system confirms the receipt of the order
Condition check
[Check if the order is normal order]
Start of process
[No]
[Yes]
[Check if the order is special order]
[Yes]
[No]
Confirm the order
Termination
Dispatch the order

Where to Use Activity Diagrams?

The basic usage of activity diagram is similar to other four UML diagrams. The specific usage is to model the control flow from one activity to another. This control flow does not include messages.

Activity diagram is suitable for modeling the activity flow of the system. An application can have multiple systems. Activity diagram also captures these systems and describes the flow from one system to another. This specific usage is not available in other diagrams. These systems can be database, external queues, or any other system.

We will now look into the practical applications of the activity diagram. From the above discussion, it is clear that an activity diagram is drawn from a very high level. So it gives high level view of a system. This high level view is mainly for business users or any other person who is not a technical person.

This diagram is used to model the activities which are nothing but business requirements. The diagram has more impact on business understanding rather than on implementation details.

Activity diagram can be used for −

- Modeling work flow by using activities.
- Modeling business requirements.
- High level understanding of the system's functionalities.
- Investigating business requirements at a later stage.

Collaboration diagrams (known as Communication Diagram in UML 2.x) are used to show how objects interact to perform the behavior of a particular use case, or a part of a use case. Along with sequence diagrams, collaboration are used by designers to define and clarify the roles of the objects that perform a particular flow of events of a use case. They are the primary source of information used to determining class responsibilities and interfaces.

What is a Collaboration?

- A Collaboration is a collection of named objects and actors with links connecting them. They collaborate in performing some task.
- A Collaboration defines a set of participants and relationships that are meaningful for a given set of purposes
- A Collaboration between objects working together provides emergent desirable functionalities in Object-Oriented systems
- Each object (responsibility) partially supports emergent functionalities
- Objects are able to produce (usable) high-level functionalities by working together
- Objects collaborate by communicating (passing messages) with one another in order to work together

Why Collaboration Diagram?

Unlike a sequence diagram, a collaboration diagram shows the relationships among the objects. Sequence diagrams and collaboration diagrams express similar information, but show it in different ways.

Because of the format of the collaboration diagram, they tend to better suited for analysis activities (see Activity: Use-Case Analysis). Specifically, they tend to be better suited to depicting simpler interactions of smaller numbers of objects. However, if the number of objects and messages grows, the diagram becomes increasingly hard to read. In addition, it is difficult to show additional descriptive information such as timing, decision points, or other unstructured information that can be easily added to the notes in a sequence diagram. So, here are some use cases that we want to create a collaboration diagram for:

- Model collaborations between objects or roles that deliver the functionalities of use cases and operations
- Model mechanisms within the architectural design of the system
- Capture interactions that show the messages passing between objects and roles within the collaboration
- Model alternative scenarios within use cases or operations that involve the collaboration of different objects and interactions
- Support the identification of objects (hence classes) that participate in use cases
- Each message in a collaboration diagram has a sequence number.
- The top-level message is numbered 1. Messages sent during the same call have the same decimal prefix but suffixes of 1, 2, etc. according to when they occur.

Notations of Collaboration Diagram

Objects

An object is represented by an object symbol showing the name of the object and its class underlined, separated by a colon:

<u>Object_name : class_name</u>

You can use objects in collaboration diagrams in the following ways:

- Each object in the collaboration is named and has its class specified
- Not all classes need to appear
- There may be more than one object of a class
- An object's class can be unspecified. Normally you create a collaboration diagram with objects first and specify their classes later.
- The objects can be unnamed, but you should name them if you want to discriminate different objects of the same class.

Actors

Normally an actor instance occurs in the collaboration diagram, as the invoker of the interaction. If you have several actor instances in the same diagram, try keeping them in the periphery of the diagram.

- Each Actor is named and has a role
- One actor will be the initiator of the use case

Links

Links connect objects and actors and are instances of associations and each link corresponds to an association in the class diagram

Links are defined as follows:

- A link is a relationship among objects across which messages can be sent. In collaboration diagrams, a link is shown as a solid line between two objects.
- An object interacts with, or navigates to, other objects through its links to these objects.
- A link can be an instance of an association, or it can be anonymous, meaning that its association is unspecified.
- Message flows are attached to links, see Messages.

Messages

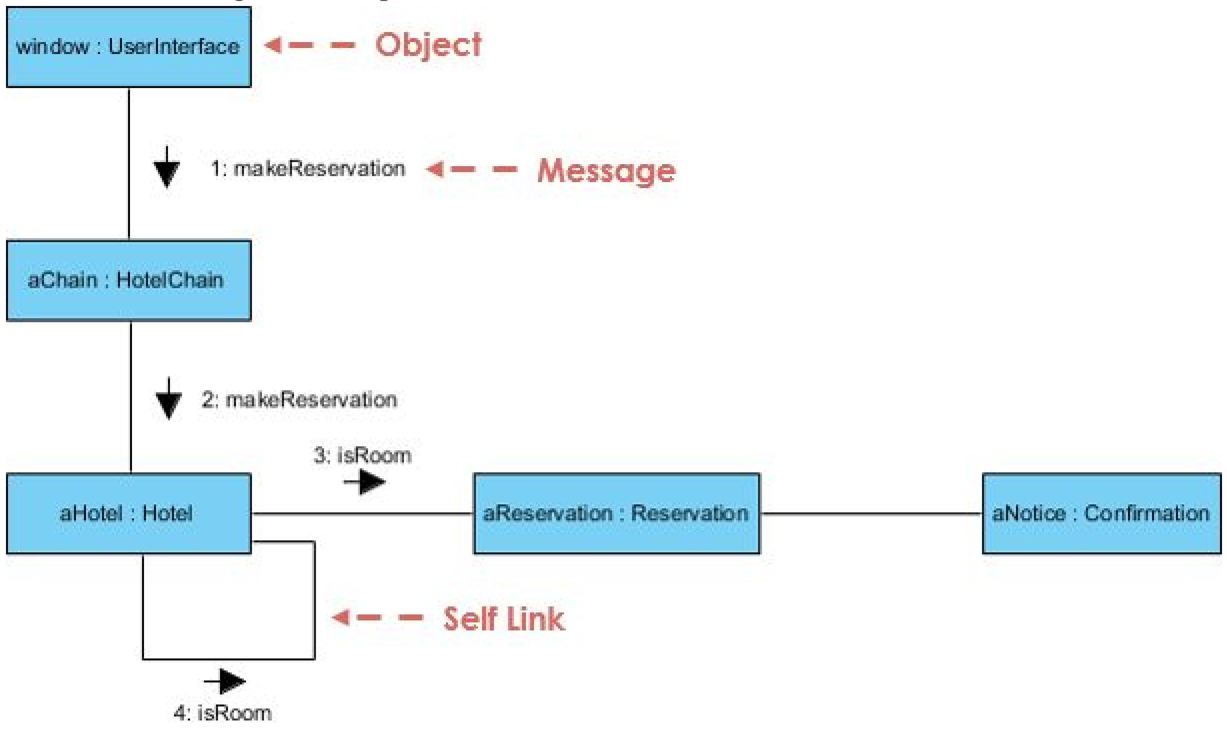
A message is a communication between objects that conveys information with the expectation that activity will ensue. In collaboration diagrams, a message is shown as a labeled arrow placed near a link.

- The message is directed from sender to receiver
- The receiver must understand the message
- The association must be navigable in that direction

Steps for Creating Collaboration Diagrams

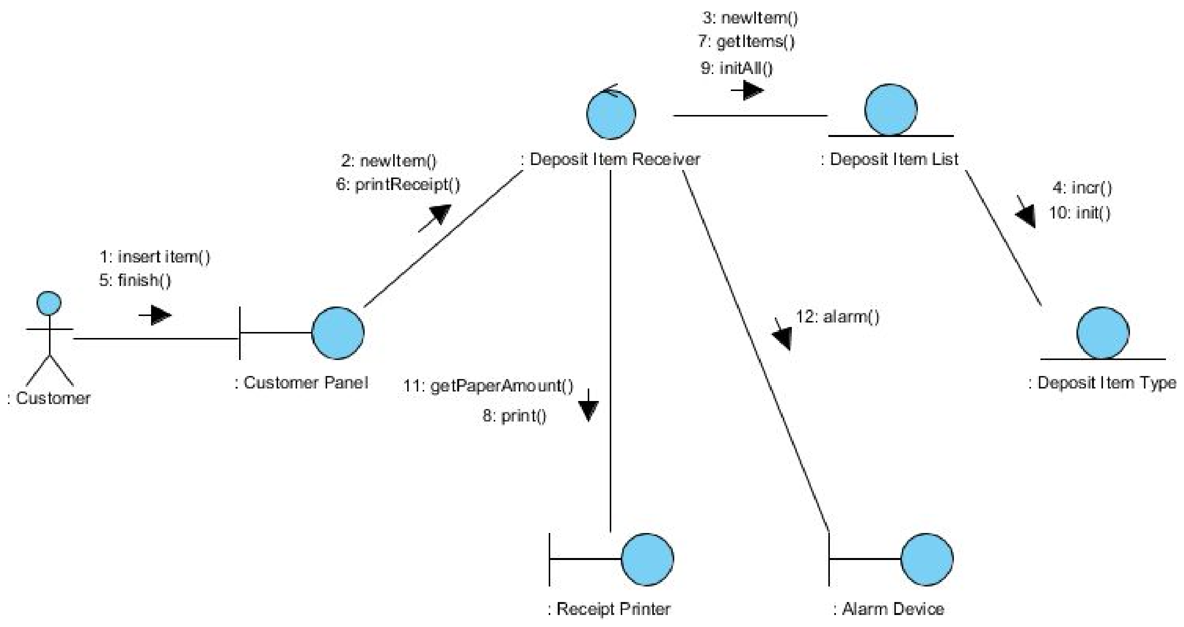
1. Identify behavior whose realization and implementation is specified
2. Identify the structural elements (class roles, objects, subsystems) necessary to carry out the functionality of the collaboration
    - Decide on the context of interaction: system, subsystem, use case and operation
3. Model structural relationships between those elements to produce a diagram showing the context of the interaction
4. Consider the alternative scenarios that may be required
    - Draw instance level collaboration diagrams, if required.
    - Optionally draw a specification level collaboration diagram to summarize the alternative scenarios in the instance level sequence diagrams

Collaboration Diagram Example



Collaboration Diagram in Robustness Diagram Format

You can have objects and actor instances in collaboration diagrams, together with links and messages describing how they are related and how they interact. The Receive Deposit Item in the Recycling-Machine System diagram shown below describes what takes place in the participating objects, in terms of how the objects communicate by sending messages to one another. You can make a collaboration diagram for each variant of a use cases flow of events.



UML - Deployment Diagrams

From: https://www.tutorialspoint.com/uml/uml_deployment_diagram.htm

Deployment diagrams are used to visualize the topology of the physical components of a system, where the software components are deployed.

Deployment diagrams are used to describe the static deployment view of a system. Deployment diagrams consist of nodes and their relationships.

Purpose of Deployment Diagrams

The term Deployment itself describes the purpose of the diagram. Deployment diagrams are used for describing the hardware components, where software components are deployed. Component diagrams and deployment diagrams are closely related.

Component diagrams are used to describe the components and deployment diagrams shows how they are deployed in hardware.

UML is mainly designed to focus on the software artifacts of a system. However, these two diagrams are special diagrams used to focus on software and hardware components.

Most of the UML diagrams are used to handle logical components but deployment diagrams are made to focus on the hardware topology of a system. Deployment diagrams are used by the system engineers.

The purpose of deployment diagrams can be described as −

- Visualize the hardware topology of a system.

- Describe the hardware components used to deploy software components.
- Describe the runtime processing nodes.

How to Draw a Deployment Diagram?

Deployment diagram represents the deployment view of a system. It is related to the component diagram because the components are deployed using the deployment diagrams. A deployment diagram consists of nodes. Nodes are nothing but physical hardware used to deploy the application.

Deployment diagrams are useful for system engineers. An efficient deployment diagram is very important as it controls the following parameters −

- Performance
- Scalability
- Maintainability
- Portability

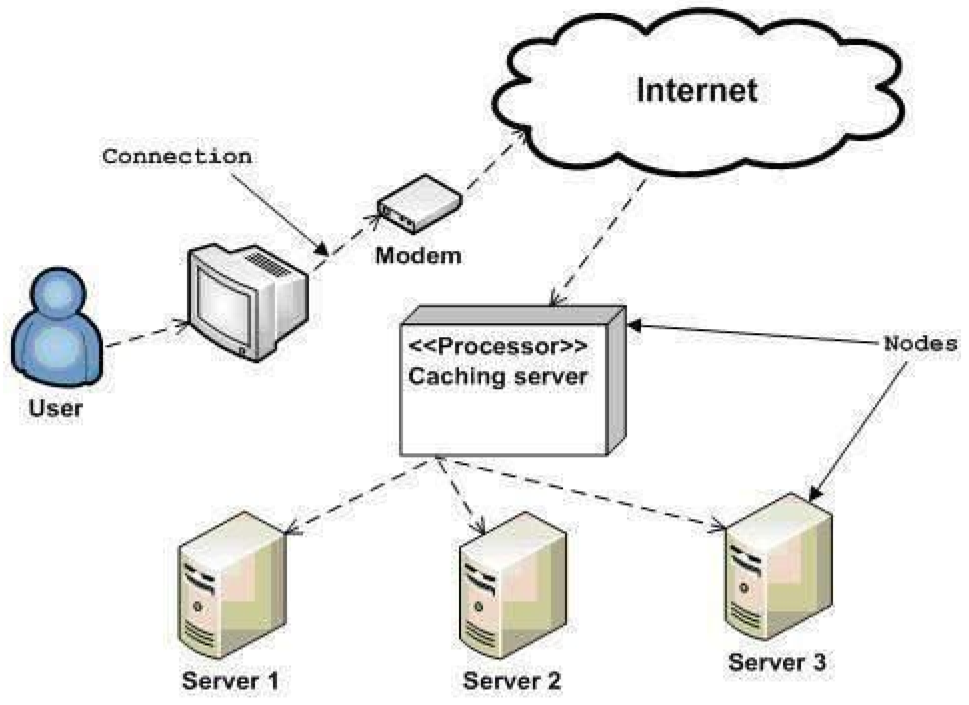
Before drawing a deployment diagram, the following artifacts should be identified −

- Nodes
- Relationships among nodes

Following is a sample deployment diagram to provide an idea of the deployment view of order management system. Here, we have shown nodes as −

- Monitor
- Modem
- Caching server
- Server

Deployment diagram of an order management system



The application is assumed to be a web-based application, which is deployed in a clustered environment using server 1, server 2, and server 3. The user connects to the application using the Internet. The control flows from the caching server to the clustered environment.

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

Where to Use Deployment Diagrams?

Deployment diagrams are mainly used by system engineers. These diagrams are used to describe the physical components (hardware), their distribution, and association.

Deployment diagrams can be visualized as the hardware components/nodes on which the software components reside.

Software applications are developed to model complex business processes. Efficient software applications are not sufficient to meet the business requirements. Business requirements can be described as the need to support the increasing number of users, quick response time, etc.

To meet these types of requirements, hardware components should be designed efficiently and in a cost-effective way.

Now-a-days software applications are very complex in nature. Software applications can be standalone, web-based, distributed, mainframe-based and many more. Hence, it is very important to design the hardware components efficiently.

Deployment diagrams can be used −

- To model the hardware topology of a system.
- To model the embedded system.
- To model the hardware details for a client/server system.
- To model the hardware details of a distributed application.
- For Forward and Reverse engineering.

Unified Modeling Language (UML) | Sequence Diagrams

In this post we discuss Sequence Diagrams. **Unified Modelling Language (UML)** is a modeling language in the field of software engineering which aims to set standard ways to visualize the design of a system. UML guides the creation of multiple types of diagrams such as interaction , structure and behaviour diagrams.
A **sequence diagram** is the most commonly used **interaction** diagram.
Interaction diagram –
An interaction diagram is used to show the **interactive behavior** of a system. Since visualizing the interactions in a system can be a cumbersome task, we use different types of interaction diagrams to capture various features and aspects of interaction in a system.
Sequence Diagrams –
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.

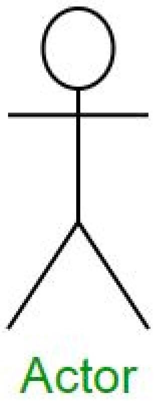


Figure – notation symbol for actor

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.

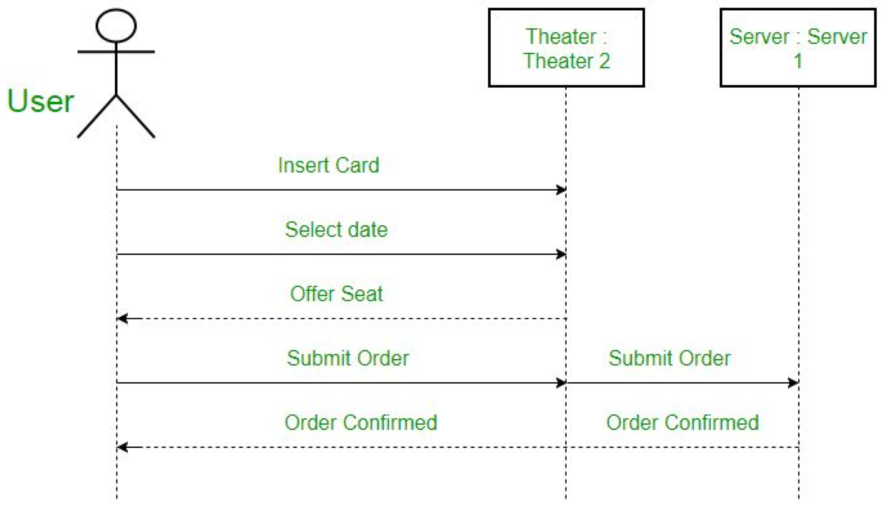


Figure – an actor interacting with a seat reservation system

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

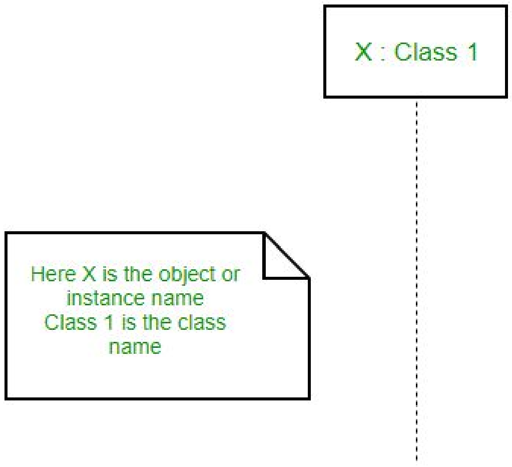

X : Class 1
Here X is the object or instance name
Class 1 is the class name

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:

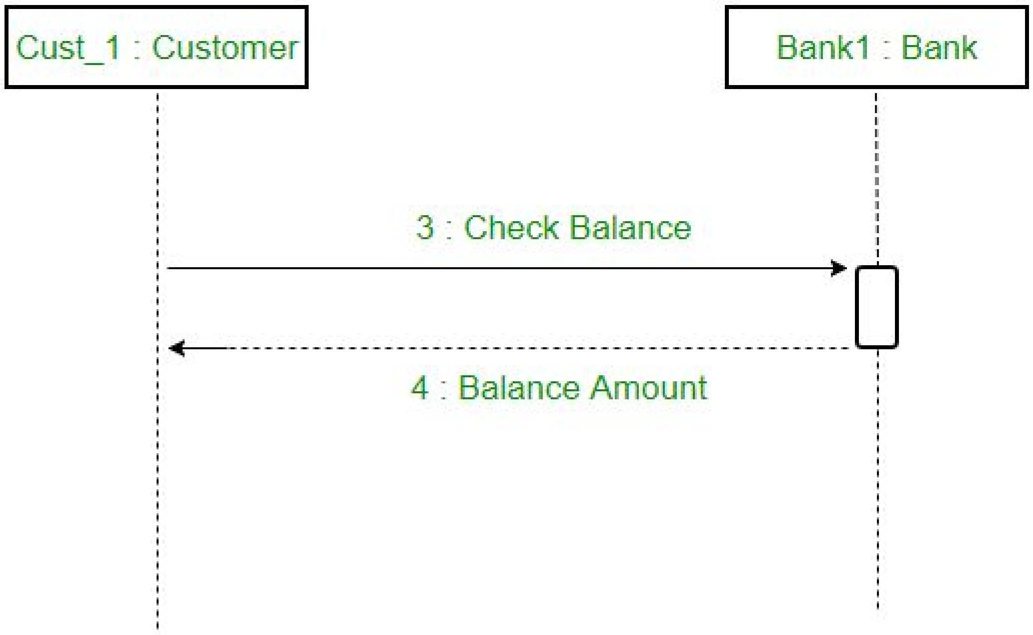


Figure – a sequence diagram

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** :

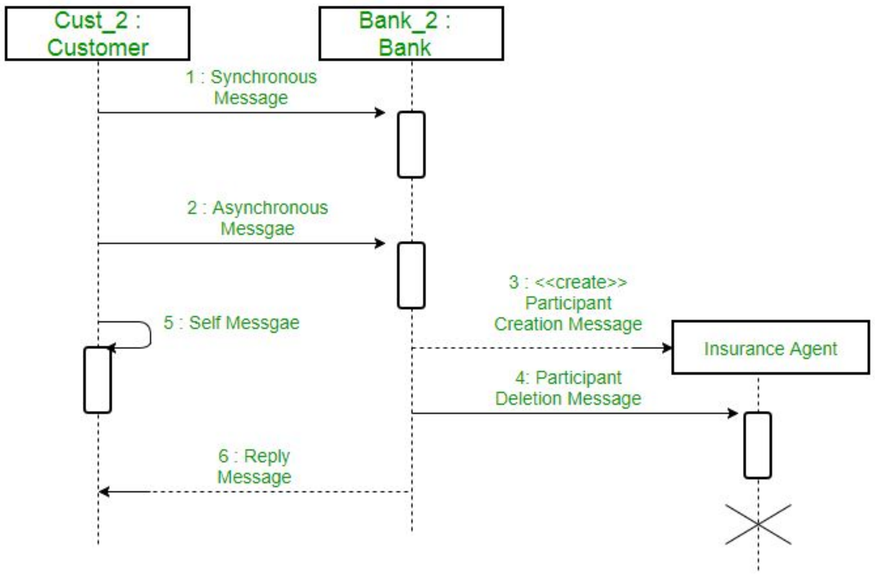


Figure – a sequence diagram with different types of messages

- **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.

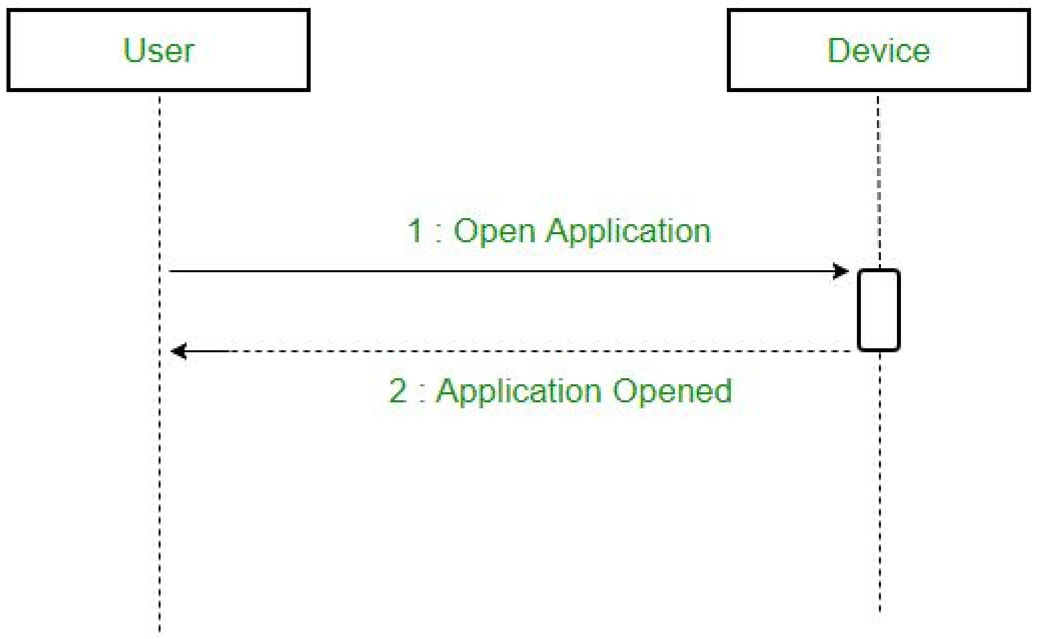
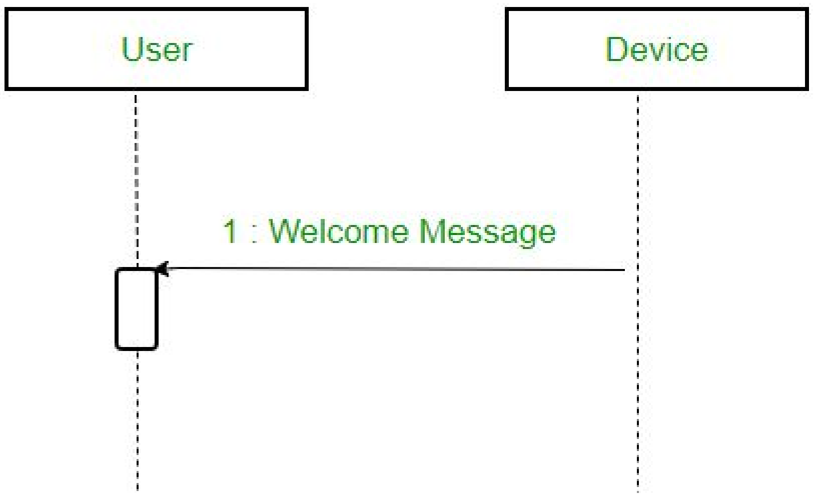


Figure – a sequence diagram using a synchronous message

- **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.

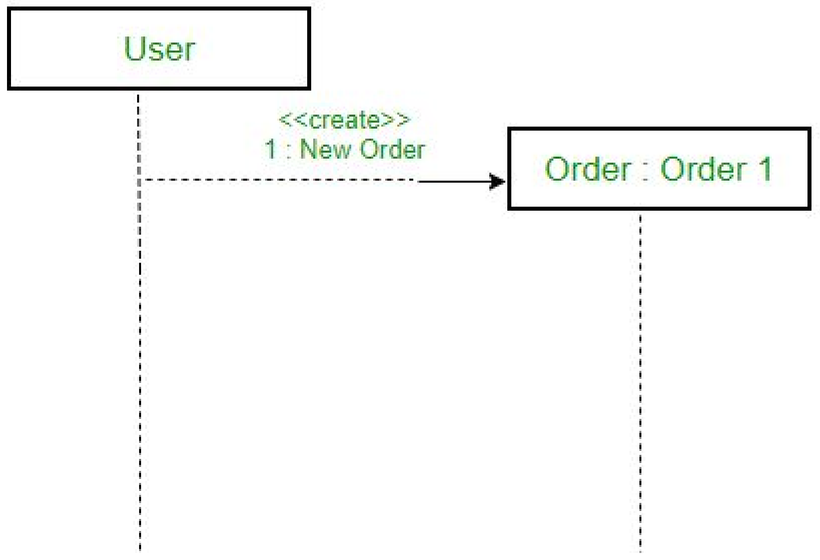


Figure – a situation where create message is used

- **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.

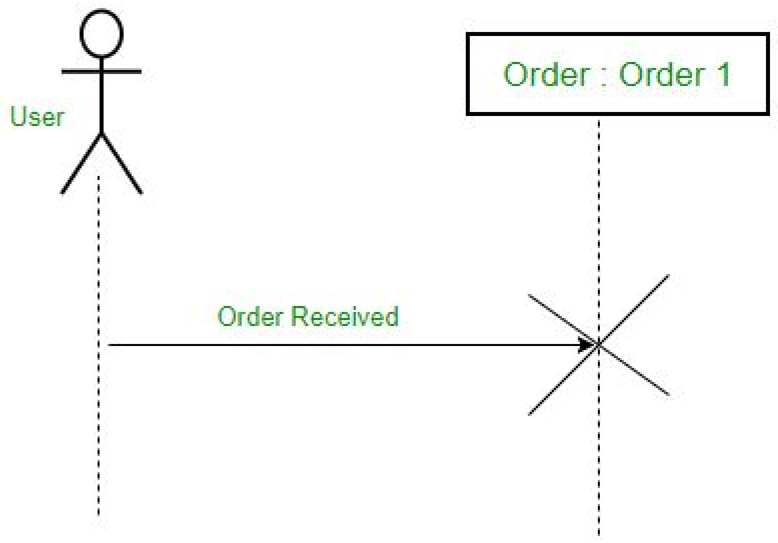


Figure – a scenario where delete message is used

- **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.

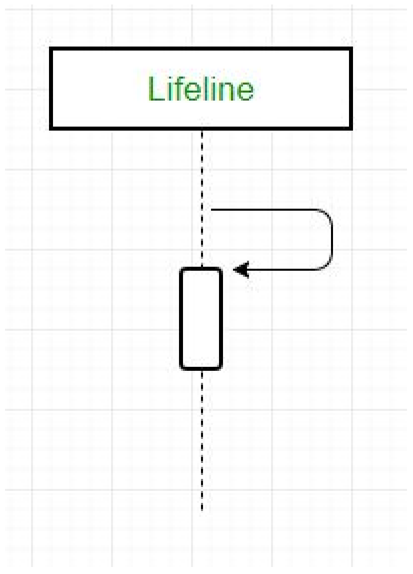


Figure – self message

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

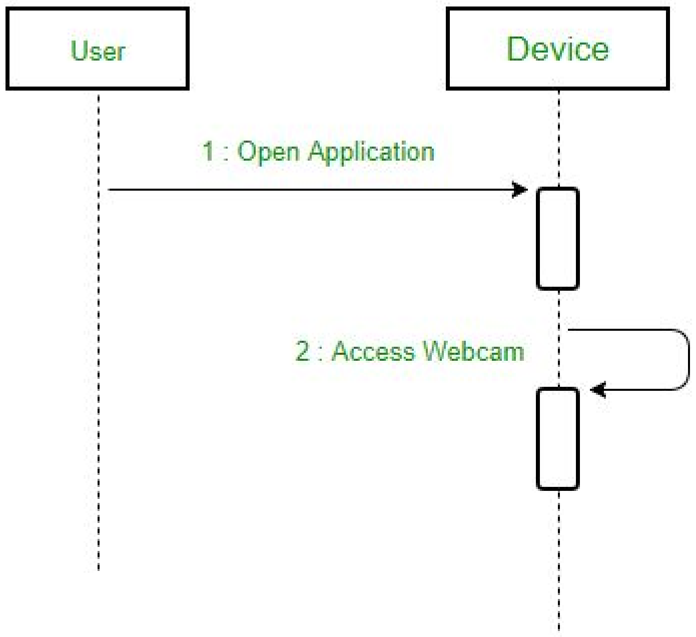


Figure – a scenario where a self message is used

- **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.

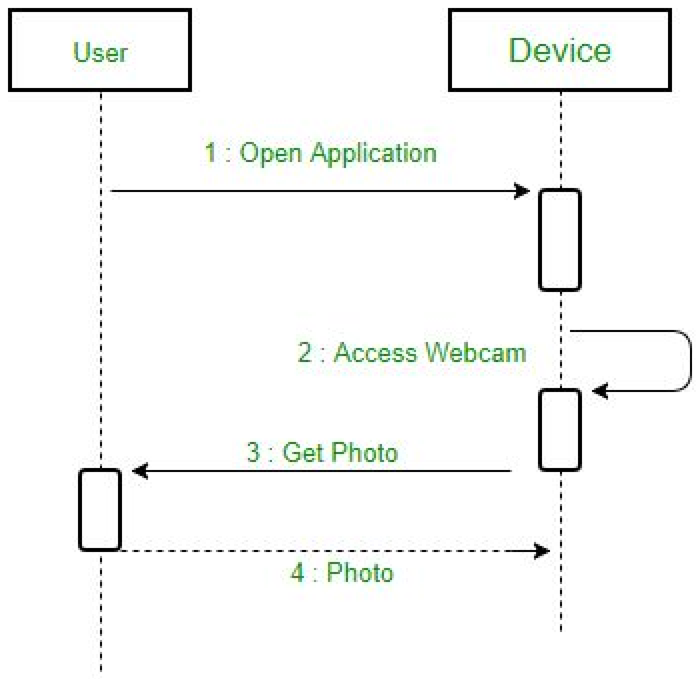


Figure – a scenario where a reply message is used

- **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.

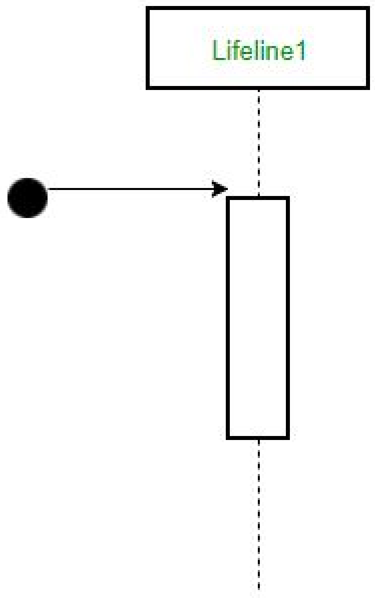


Figure – found message

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

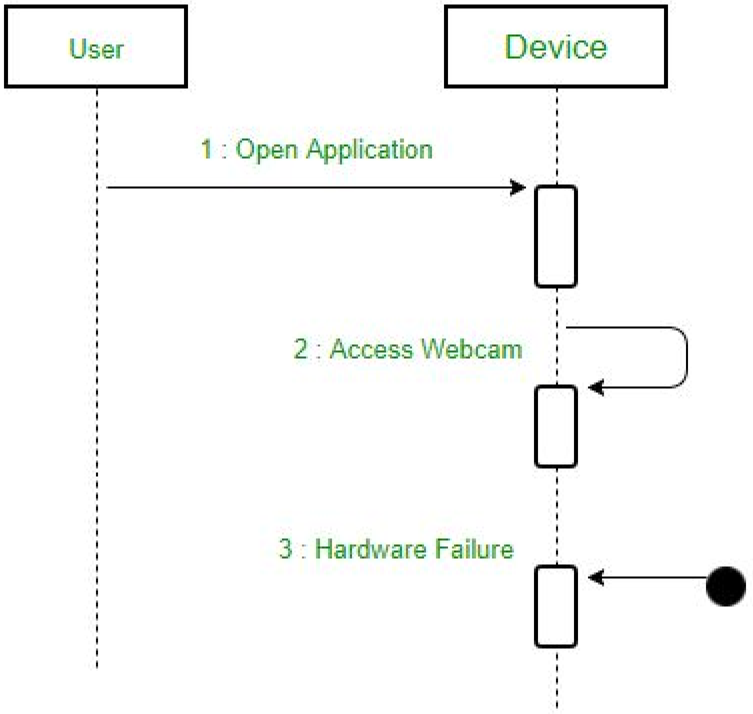


Figure – a scenario where found message is used

- **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.

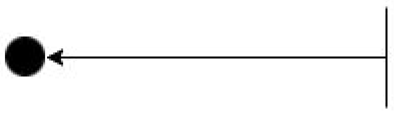


Figure – lost message

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.

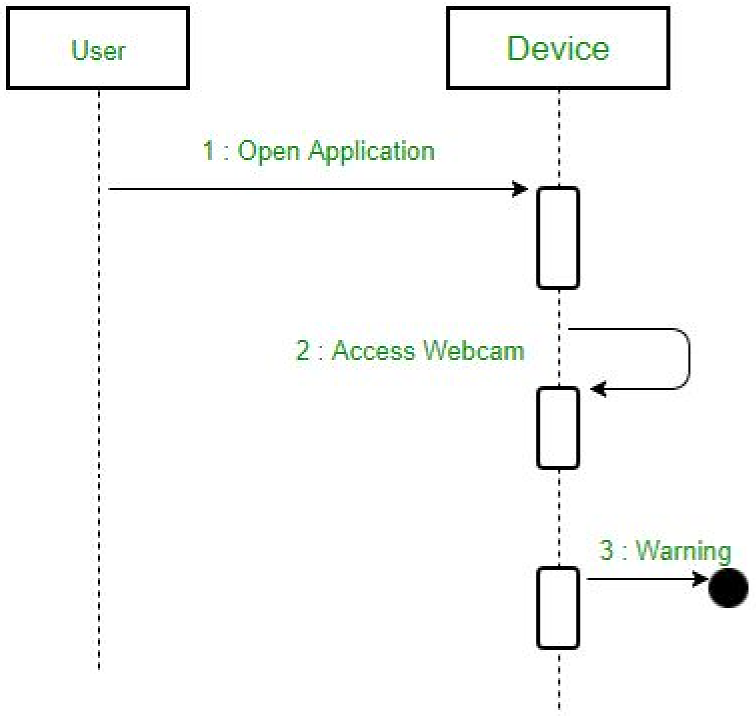


Figure – a scenario where lost message is used

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.

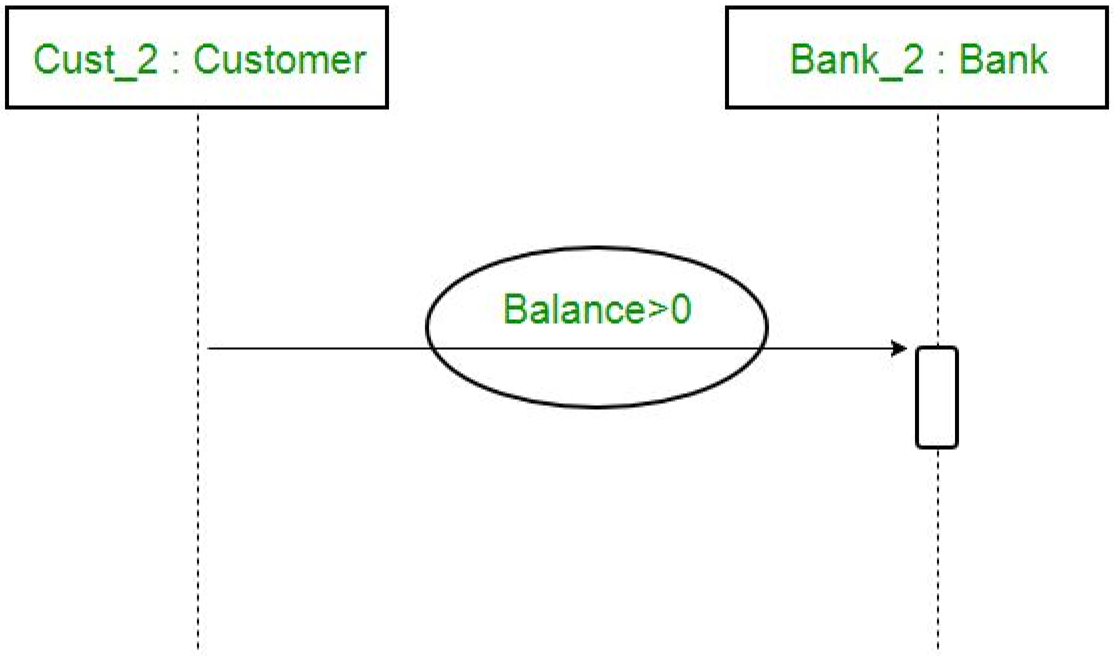


Figure – sequence diagram using a guard

A sequence diagram for an emotion based music player –

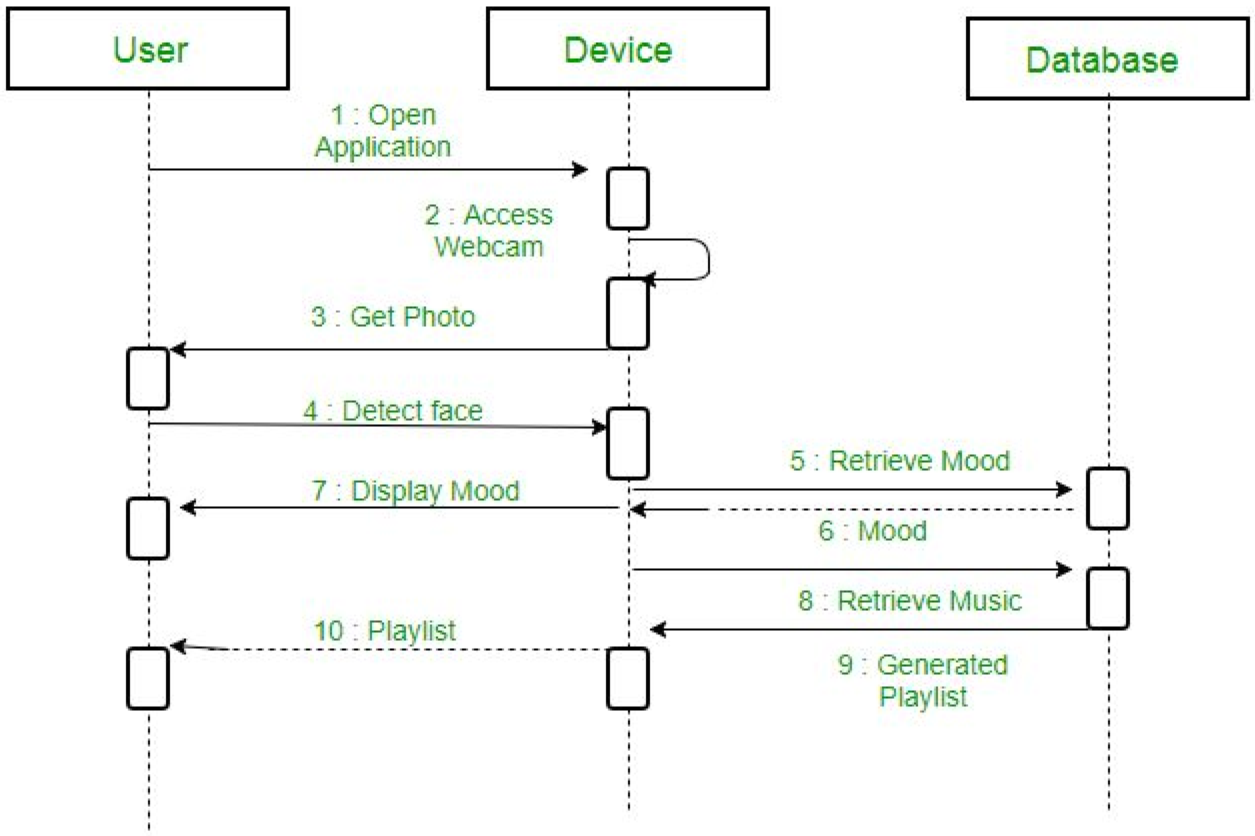


Figure – a sequence diagram for an emotion based music player

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.