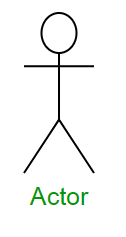
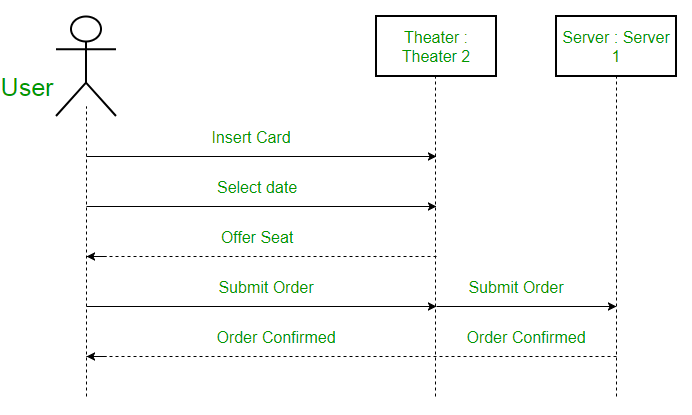
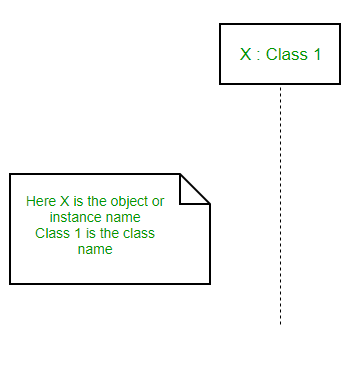
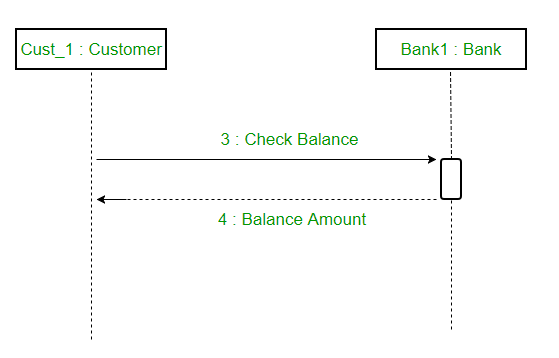
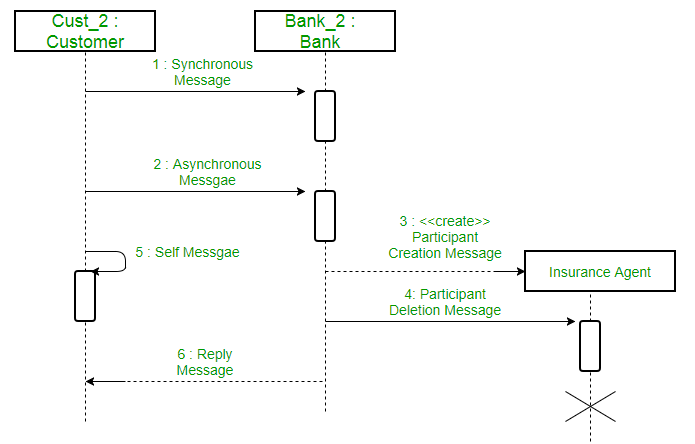
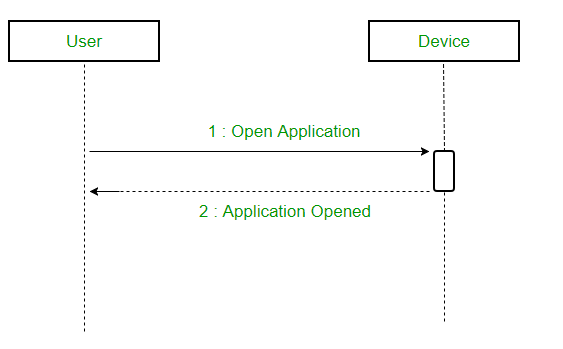
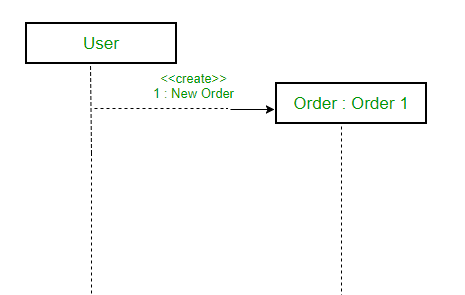
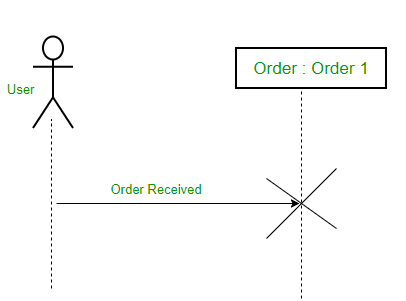
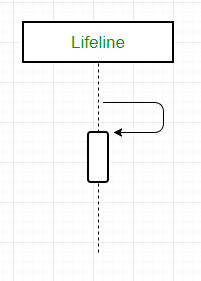
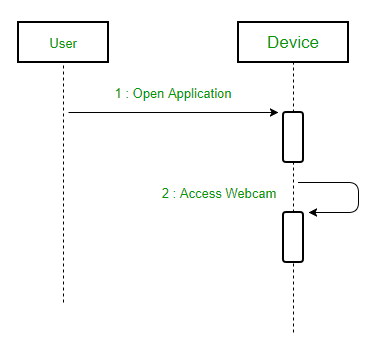
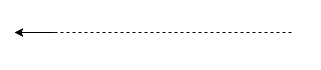
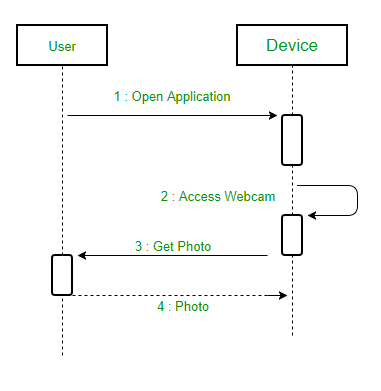
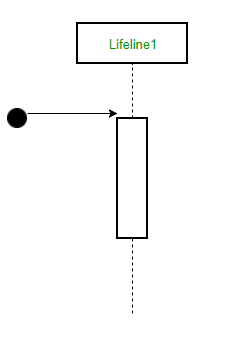
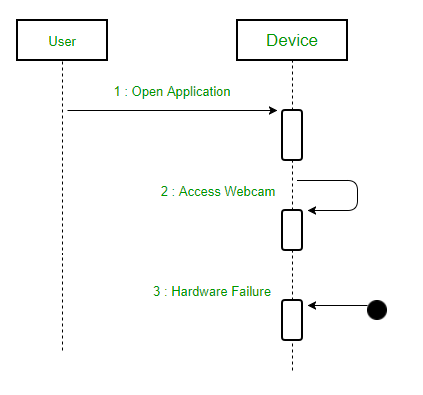
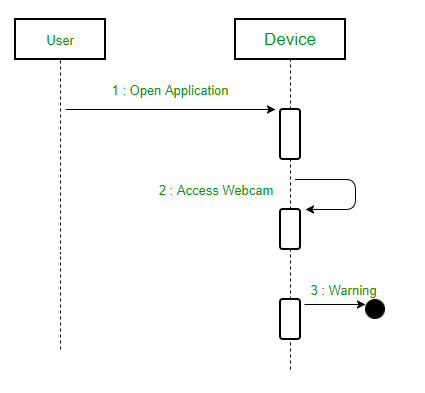
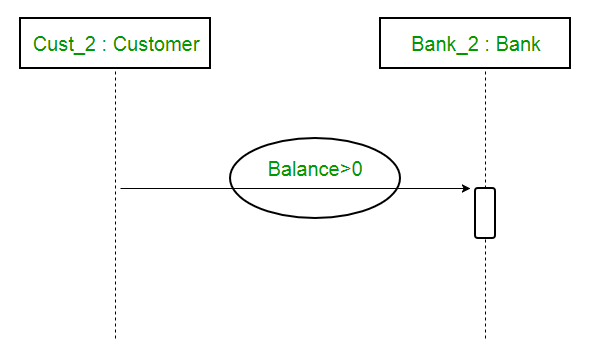


**Unified Modelling Language (UML)** is a modeling language in the field of software engineering which aims to set standard ways to visualize the design of a system. UML guides the creation of multiple types of diagrams such as interaction , structure and behaviour diagrams. A **sequence diagram** is 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 actorWe 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 –** lifelineWe 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 messageFor 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 messageFor 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 messageIt 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 messageThe warning might be generated for the user or other software/object that the lifeline is interacting 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 playerThe 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.

**Unified Modeling Language (UML) | State Diagrams**

A **state diagram** is used to represent the condition of the system or part of the system at finite instances of time. It’s a **behavioral** diagram and it represents the behavior using finite state transitions. State diagrams are also referred to as **State machines** and **State-chart Diagrams**. These terms are often used interchangeably. So simply, a state diagram is used to model the dynamic behavior of a class in response to time and changing external stimuli. We can say that each and every class has a state but we don’t model every class using State diagrams. We prefer to model the states with three or more states.

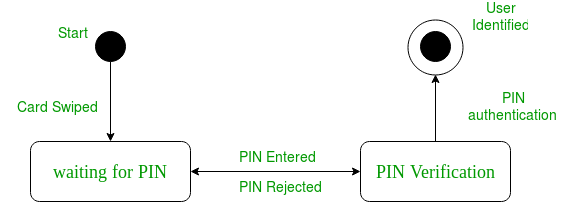
**Uses of statechart diagram –**

* We use it to state the events responsible for change in state (we do not show what processes cause those events).
* We use it to model the dynamic behavior of the system .
* To understand the reaction of objects/classes to internal or external stimuli.

Firstly let us understand what are **Behavior diagrams**? There are two types of diagrams in UML :

1. **Structure Diagrams –** Used to model the static structure of a system, for example- class diagram, package diagram, object diagram, deployment diagram etc.
2. **Behavior diagram –** Used to model the dynamic change in the system over time. They are used to model and construct the functionality of a system. So, a behavior diagram simply guides us through the functionality of the system using Use case diagrams, Interaction diagrams, Activity diagrams and State diagrams.

**Difference between state diagram and flowchart –**  
The basic purpose of a **state diagram** is to portray various changes in state of the class and not the processes or commands causing the changes. However, a **flowchart** on the other hand portrays the processes or commands that on execution change the state of class or an object of the class.



**Figure –** a state diagram for user verification

The state diagram above shows the different states in which the verification sub-system or class exist for a particular system.

**Basic components of a statechart diagram –**

1. **Initial state –** We use a black filled circle represent the initial state of a System or a class.

UML-State-Diagram

**Figure –** initial state notation

1. **Transition –** We use a solid arrow to represent the transition or change of control from one state to another. The arrow is labelled with the event which causes the change in state.

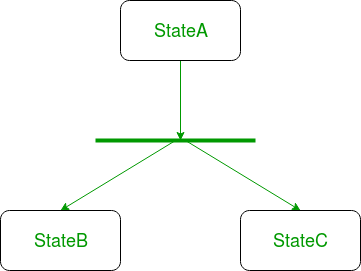
UML-State-Diagram

**Figure –** transition

1. **State –** We use a rounded rectangle to represent a state. A state represents the conditions or circumstances of an object of a class at an instant of time.  
   UML-State-Diagram

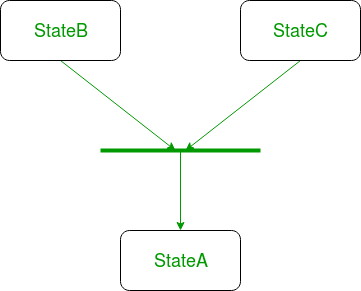
**Figure –** state notation

1. **Fork –** We use a rounded solid rectangular bar to represent a Fork notation with incoming arrow from the parent state and outgoing arrows towards the newly created states. We use the fork notation to represent a state splitting into two or more concurrent states.



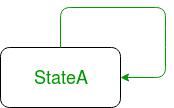
**Figure –** a diagram using the fork notation

1. **Join –** We use a rounded solid rectangular bar to represent a Join notation with incoming arrows from the joining states and outgoing arrow towards the common goal state. We use the join notation when two or more states concurrently converge into one on the occurrence of an event or events.



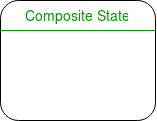
**Figure –** a diagram using join notation

1. **Self transition –** We use a solid arrow pointing back to the state itself to represent a self transition. There might be scenarios when the state of the object does not change upon the occurrence of an event. We use self transitions to represent such cases.



**Figure –** self transition notation

1. **Composite state –** We use a rounded rectangle to represent a composite state also.We represent a state with internal activities using a composite state.



**Figure –** a state with internal activities

1. **Final state –** We use a filled circle within a circle notation to represent the final state in a state machine diagram.

UML-State-Diagram

**Figure –** final state notation

**Steps to draw a state diagram –**

1. Identify the initial state and the final terminating states.
2. Identify the possible states in which the object can exist (boundary values corresponding to different attributes guide us in identifying different states).
3. Label the events which trigger these transitions.

**Example –** state diagram for an online order –



**Figure –** state diagram for an online order

The UMl diagrams we draw depend on the system we aim to represent. Here is just an example of how an online ordering system might look like :

1. On the event of an order being received, we transit from our initial state to Unprocessed order state.
2. The unprocessed order is then checked.
3. If the order is rejected, we transit to the Rejected Order state.
4. If the order is accepted and we have the items available we transit to the fulfilled order state.
5. However if the items are not available we transit to the Pending Order state.
6. After the order is fulfilled, we transit to the final state. In this example, we merge the two states i.e. Fulfilled order and Rejected order into one final state.