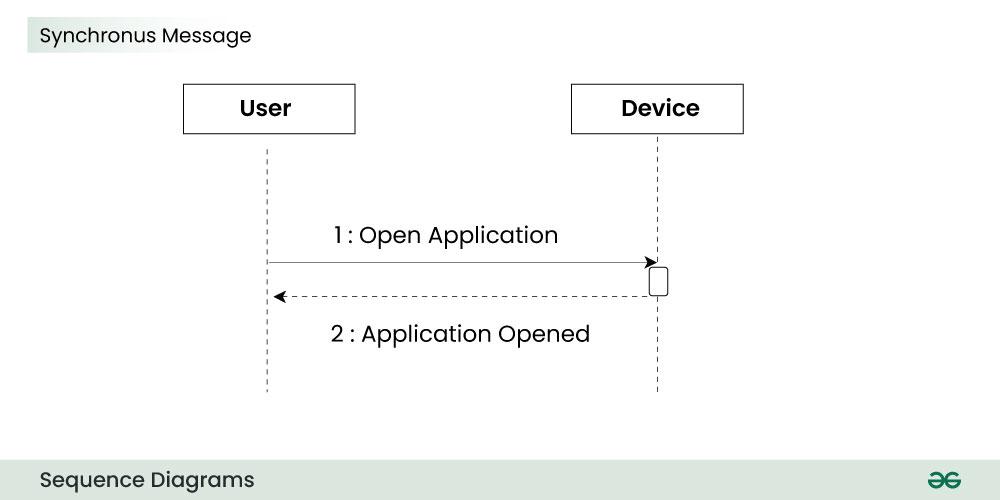


**Synchronous messages**

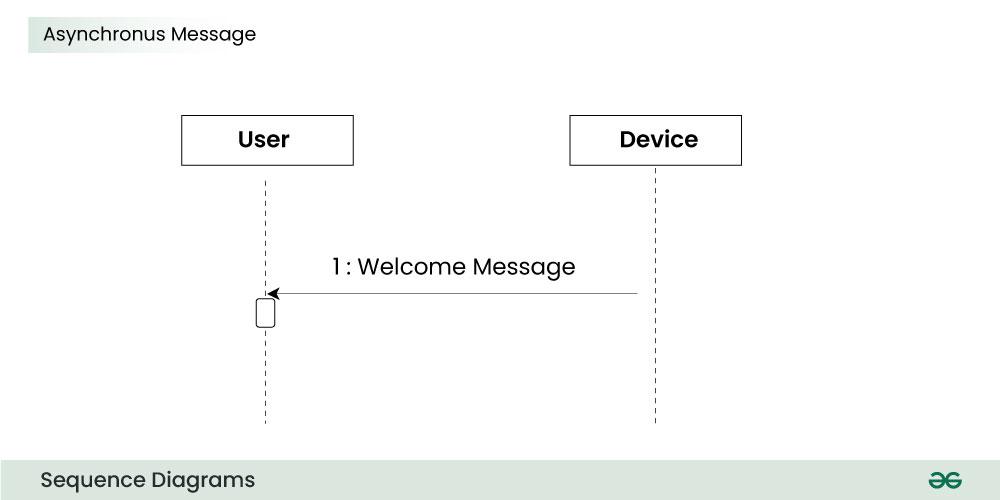
A synchronous message waits for a reply before the interaction can move forward. The sender waits until the receiver has completed the processing of the message. The caller continues only when it knows that the receiver has processed the previous message i.e. it receives a reply message.

* A large number of calls in object oriented programming are synchronous.
* We use a **solid arrow head** to represent a synchronous message.



**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.

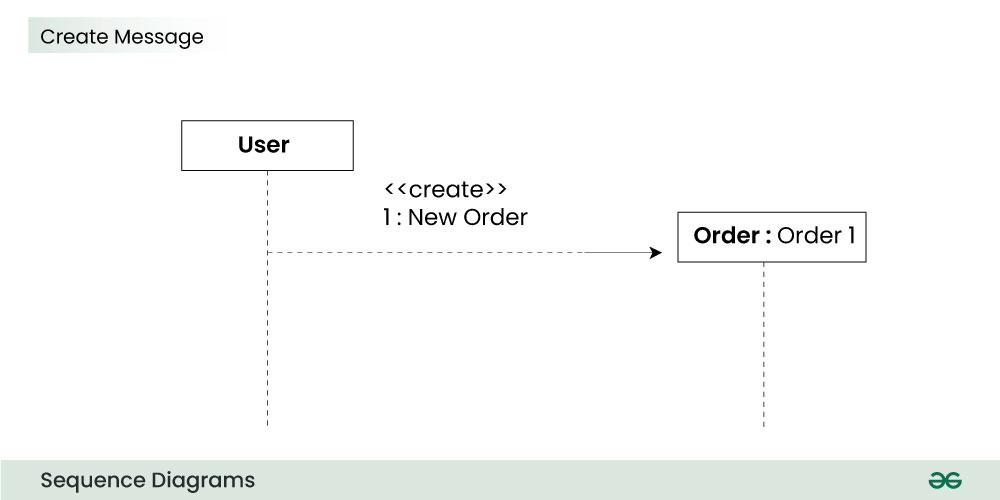


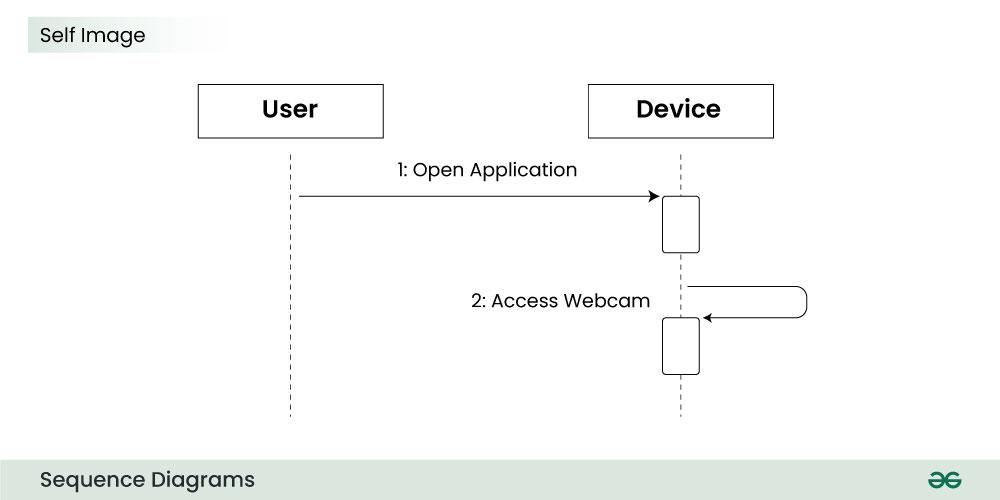
**Create message**

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.





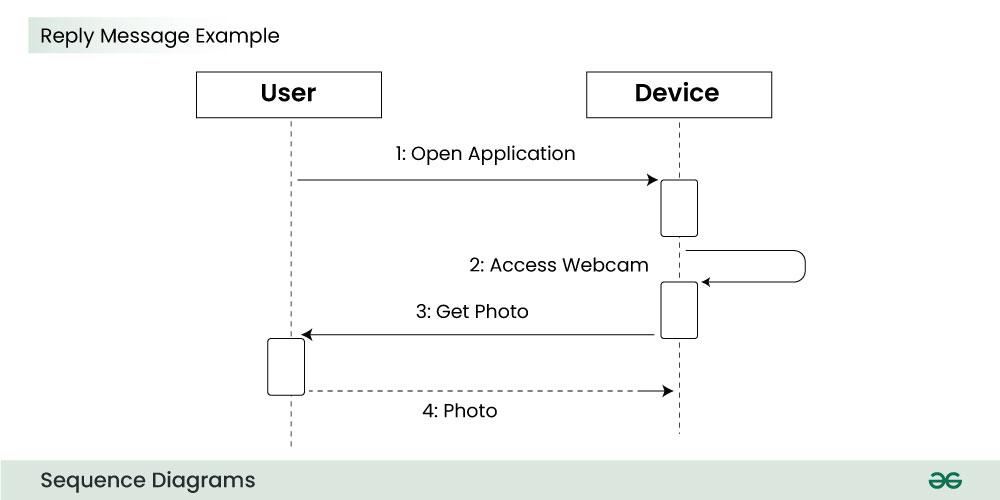
**Reply Message**

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 arrow head with a dotted line**. The interaction moves forward only when a reply message is sent by the receiver.



**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.

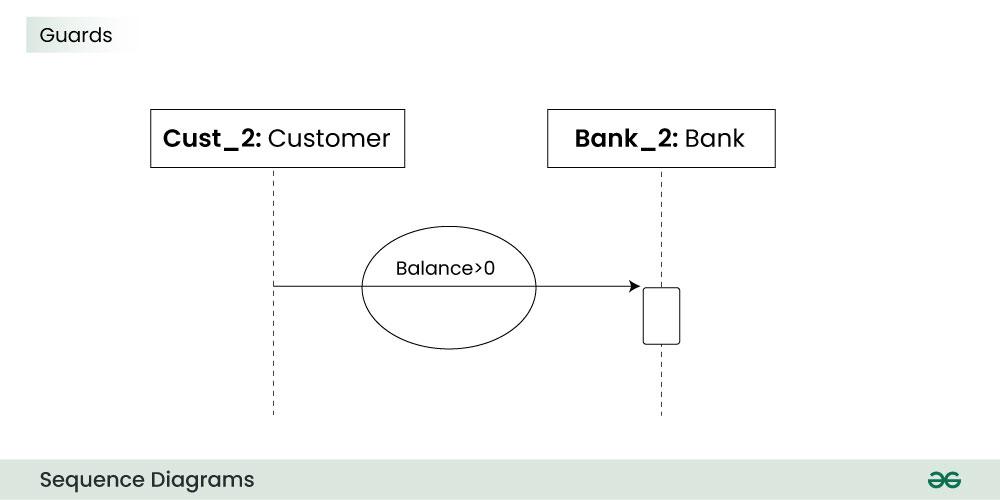


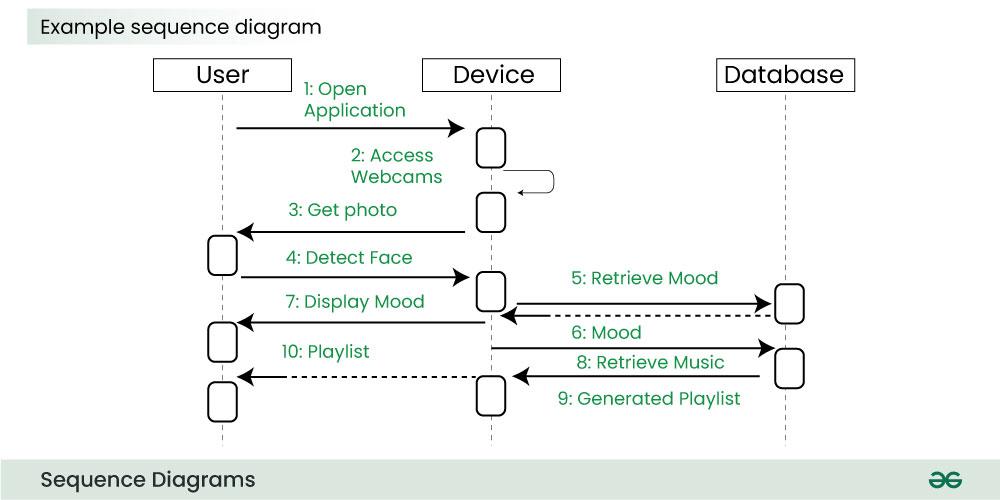
## ****Guards****

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.





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.

**How to create Sequence Diagrams?**

Creating a sequence diagram involves several steps, and it’s typically done during the design phase of software development to illustrate how different components or objects interact over time. Here’s a step-by-step guide on how to create sequence diagrams:

1. **Identify the Scenario:**
   * Understand the specific scenario or use case that you want to represent in the sequence diagram. This could be a specific interaction between objects or the flow of messages in a particular process.
2. **List the Participants:**
   * Identify the participants (objects or actors) involved in the scenario. Participants can be users, systems, or external entities.
3. **Define Lifelines:**
   * Draw a vertical dashed line for each participant, representing the lifeline of each object over time. The lifeline represents the existence of an object during the interaction.
4. **Arrange Lifelines:**
   * Position the lifelines horizontally in the order of their involvement in the interaction. This helps in visualizing the flow of messages between participants.
5. **Add Activation Bars:**
   * For each message, draw an activation bar on the lifeline of the sending participant. The activation bar represents the duration of time during which the participant is actively processing the message.
6. **Draw Messages:**
   * Use arrows to represent messages between participants. Messages flow horizontally between lifelines, indicating the communication between objects. Different types of messages include synchronous (solid arrow), asynchronous (dashed arrow), and self-messages.
7. **Include Return Messages:**
   * If a participant sends a response message, draw a dashed arrow returning to the original sender to represent the return message.
8. **Indicate Timing and Order:**
   * Use numbers to indicate the order of messages in the sequence. You can also use vertical dashed lines to represent occurrences of events or the passage of time.
9. **Include Conditions and Loops:**
   * Use combined fragments to represent conditions (like if statements) and loops in the interaction. This adds complexity to the sequence diagram and helps in detailing the control flow.
10. **Consider Parallel Execution:**
    * If there are parallel activities happening, represent them by drawing parallel vertical dashed lines and placing the messages accordingly.
11. **Review and Refine:**
    * Review the sequence diagram for clarity and correctness. Ensure that it accurately represents the intended interaction. Refine as needed.
12. **Add Annotations and Comments:**
    * Include any additional information, annotations, or comments that provide context or clarification for elements in the diagram.
13. **Document Assumptions and Constraints:**
    * If there are any assumptions or constraints related to the interaction, document them alongside the diagram.
14. **Tools:**
    * Use a UML modeling tool or diagramming software to create a neat and professional-looking sequence diagram. These tools often provide features for easy editing, collaboration, and documentation.

