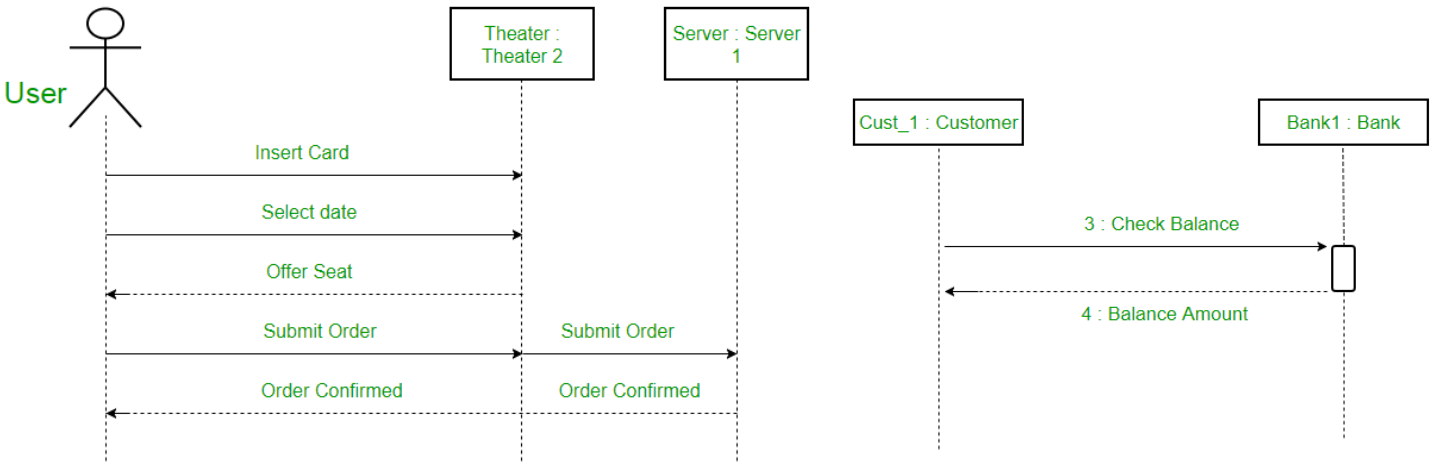
Name:	Rupin Malik and Kartik Menon
UID	2020300035 and 2020300040
Project Name	WeChange
Experiment Number	3
Aim	Sequence Diagram

Theory

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.

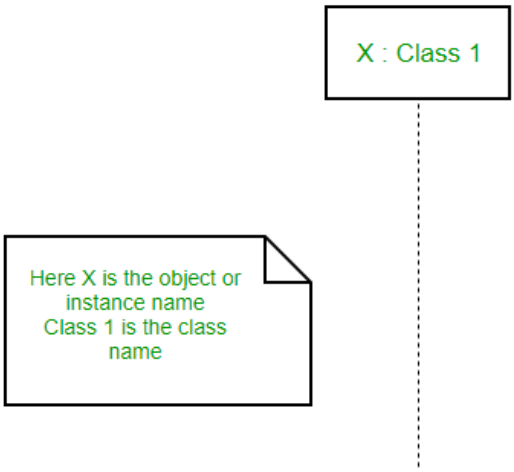


Sequence Diagram Notations –

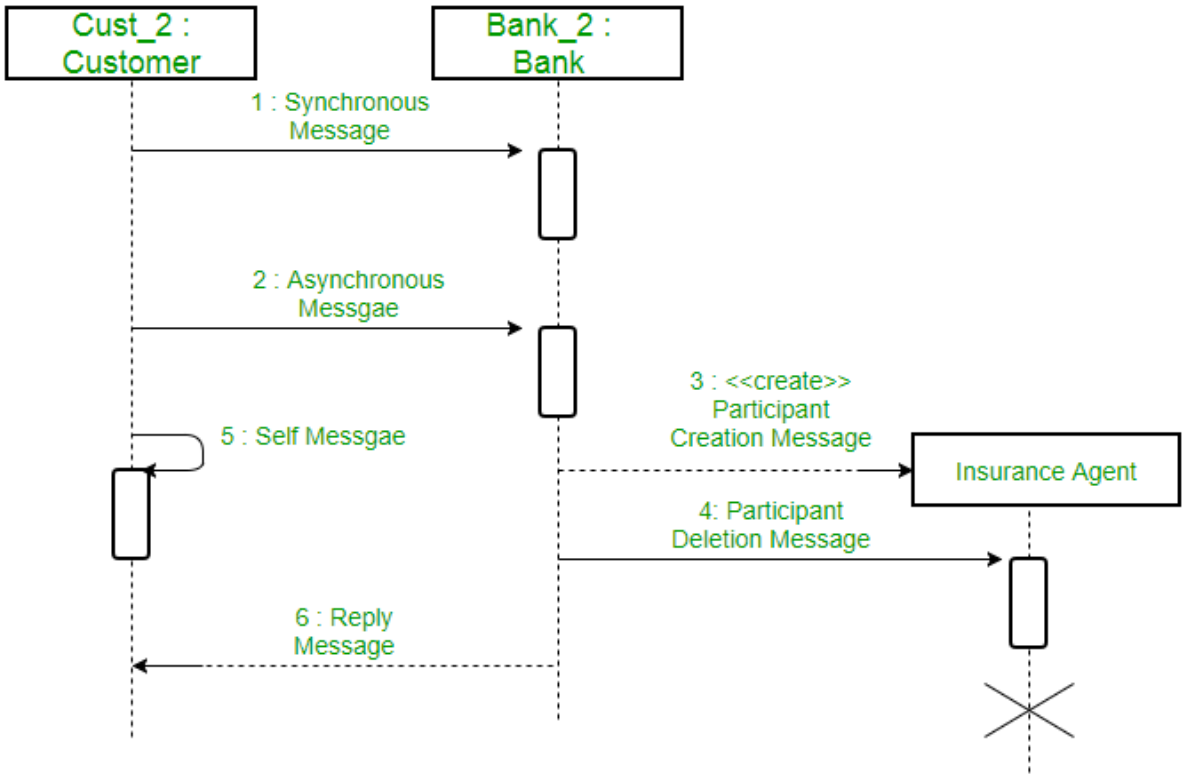
1. Actors – An actor in a UML diagram represents a type of role where it interacts with the system and its objects. It is important to note here that an actor is always outside the scope of the system we aim to model using the UML diagram. Figure – notation symbol for actor. We use actors to depict various 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 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.

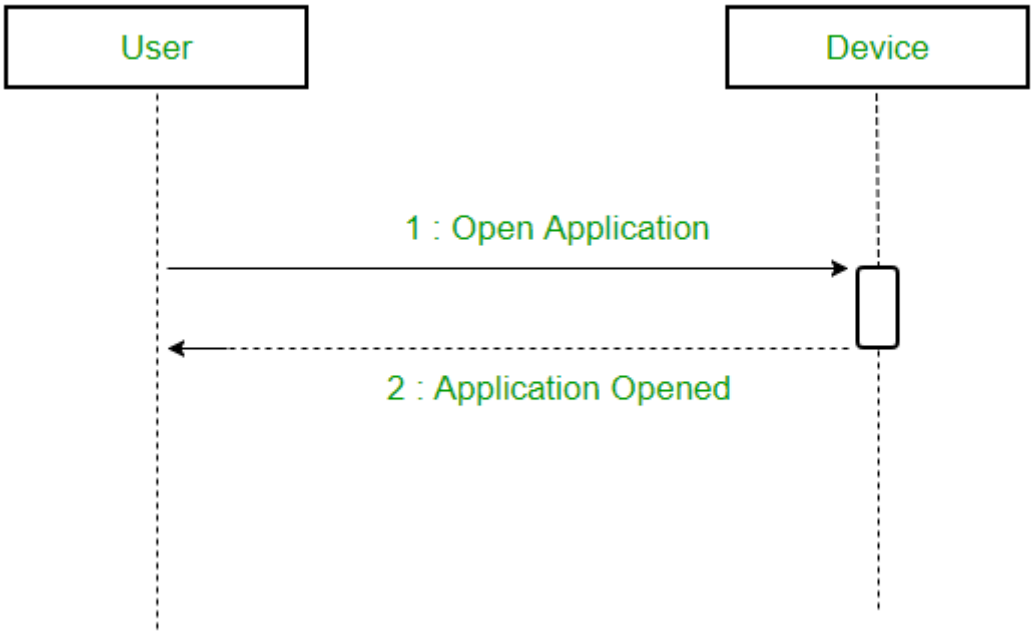
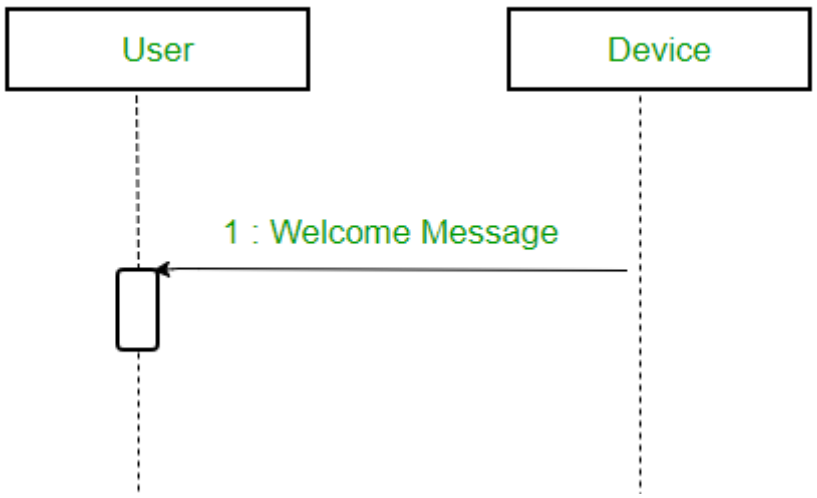


Figure – a sequence diagram using a synchronous message

- 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.

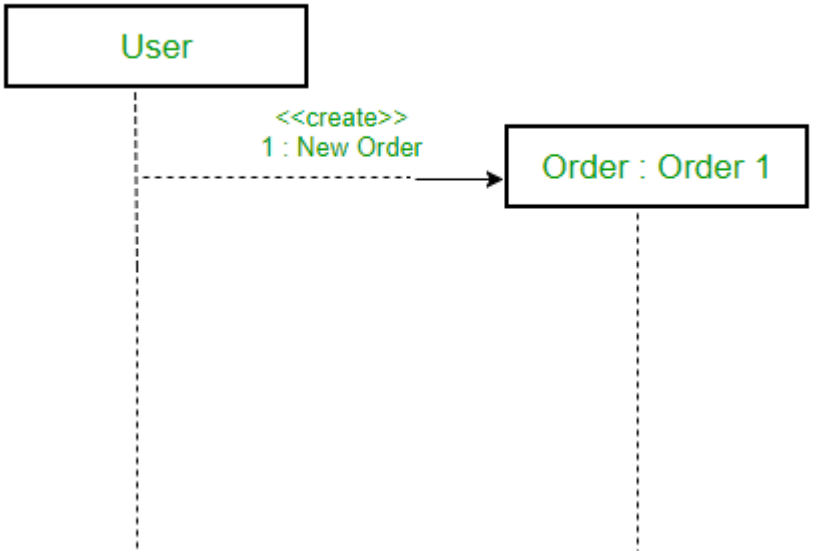


Figure – a situation where create message is used

- 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.

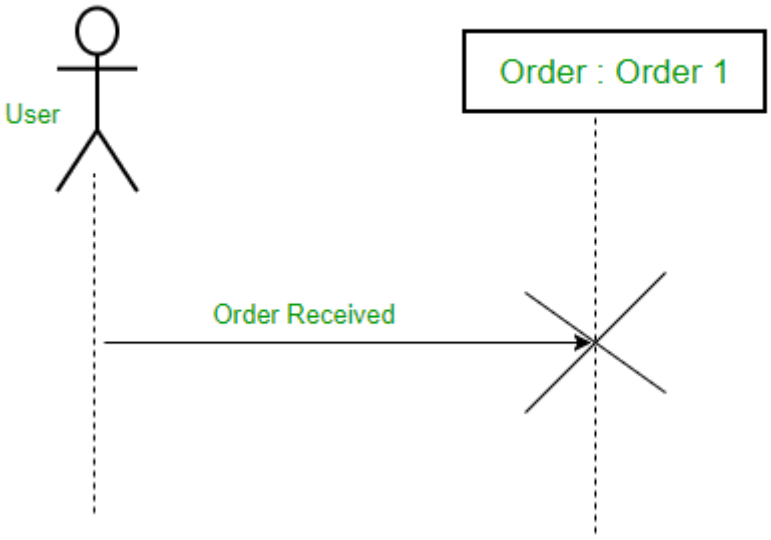


Figure – a scenario where delete message is used

- 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.

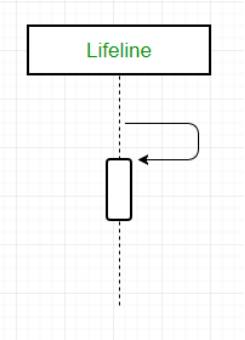


Figure – self message.

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

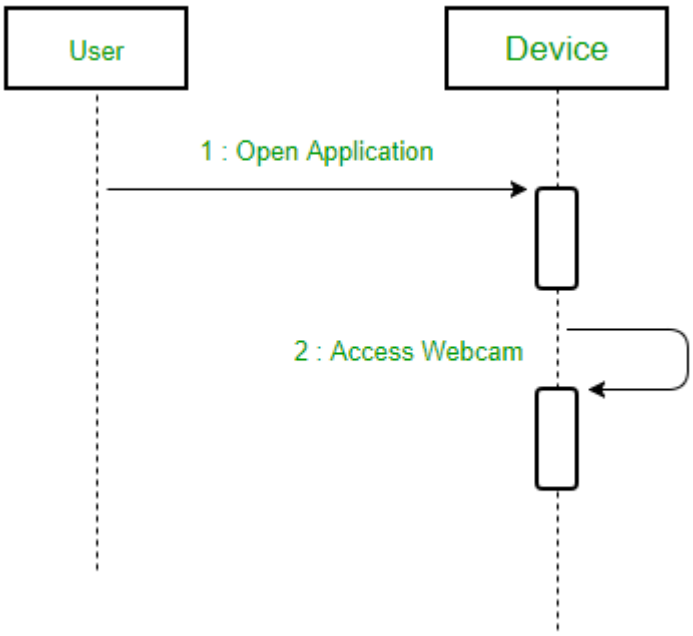


Figure – a scenario where a self message is used

- 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 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.

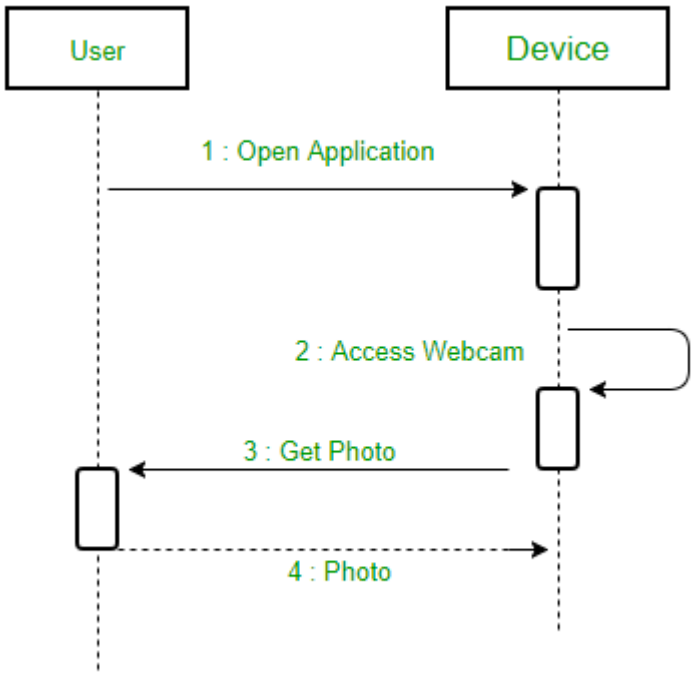


Figure – a scenario where a reply message is used

- 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.

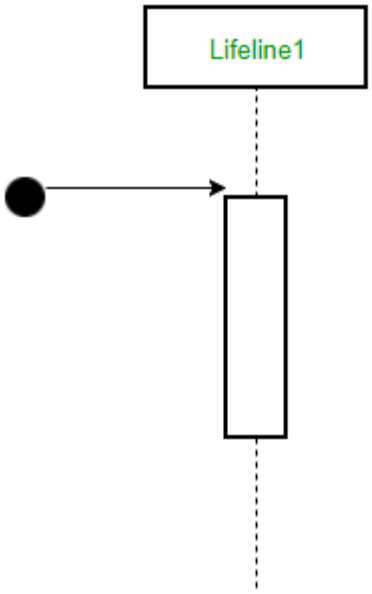


Figure – found message

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

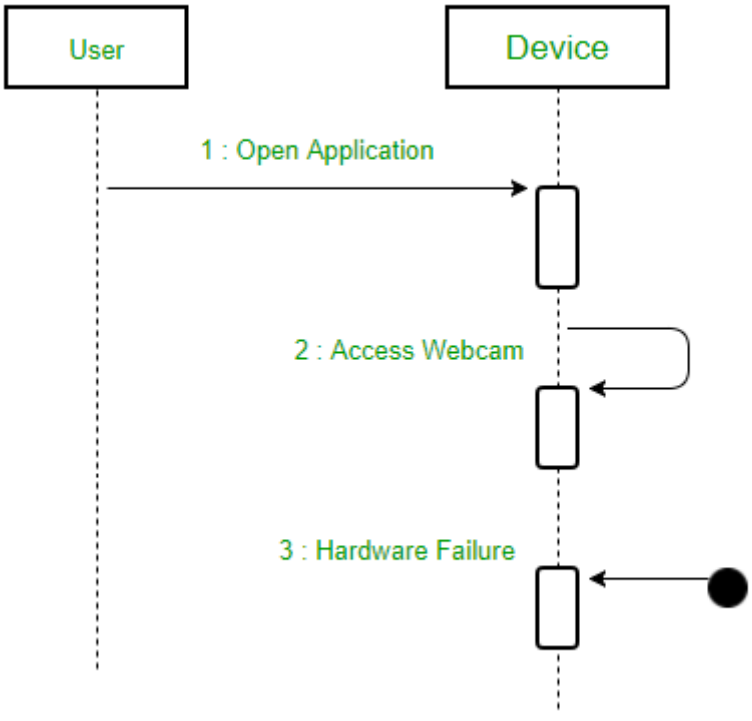


Figure – a scenario where found message is used

- 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. The 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.

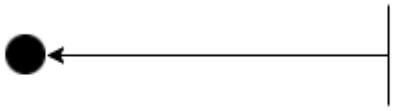


Figure – a scenario where lost message is used

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.

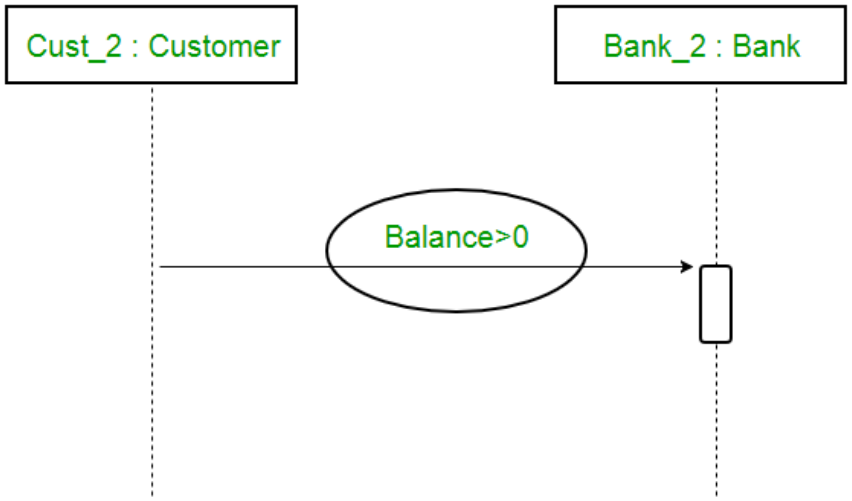


Figure – sequence diagram using a guard

Collaboration Diagram:

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

As per Object-Oriented Programming (OOPs), an object entity has various attributes associated with it. Usually, there are multiple objects present inside an object-oriented system where each object can be associated with any other object inside the system. Collaboration Diagrams are used to explore the architecture of objects inside the system. The message flow between the objects can be represented using a collaboration diagram.

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:

Object_name : class_name

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:

- Identify behavior whose realization and implementation is specified
- 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
- Model structural relationships between those elements to produce a diagram showing the context of the interaction

- 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

Benefits of Collaboration Diagram:

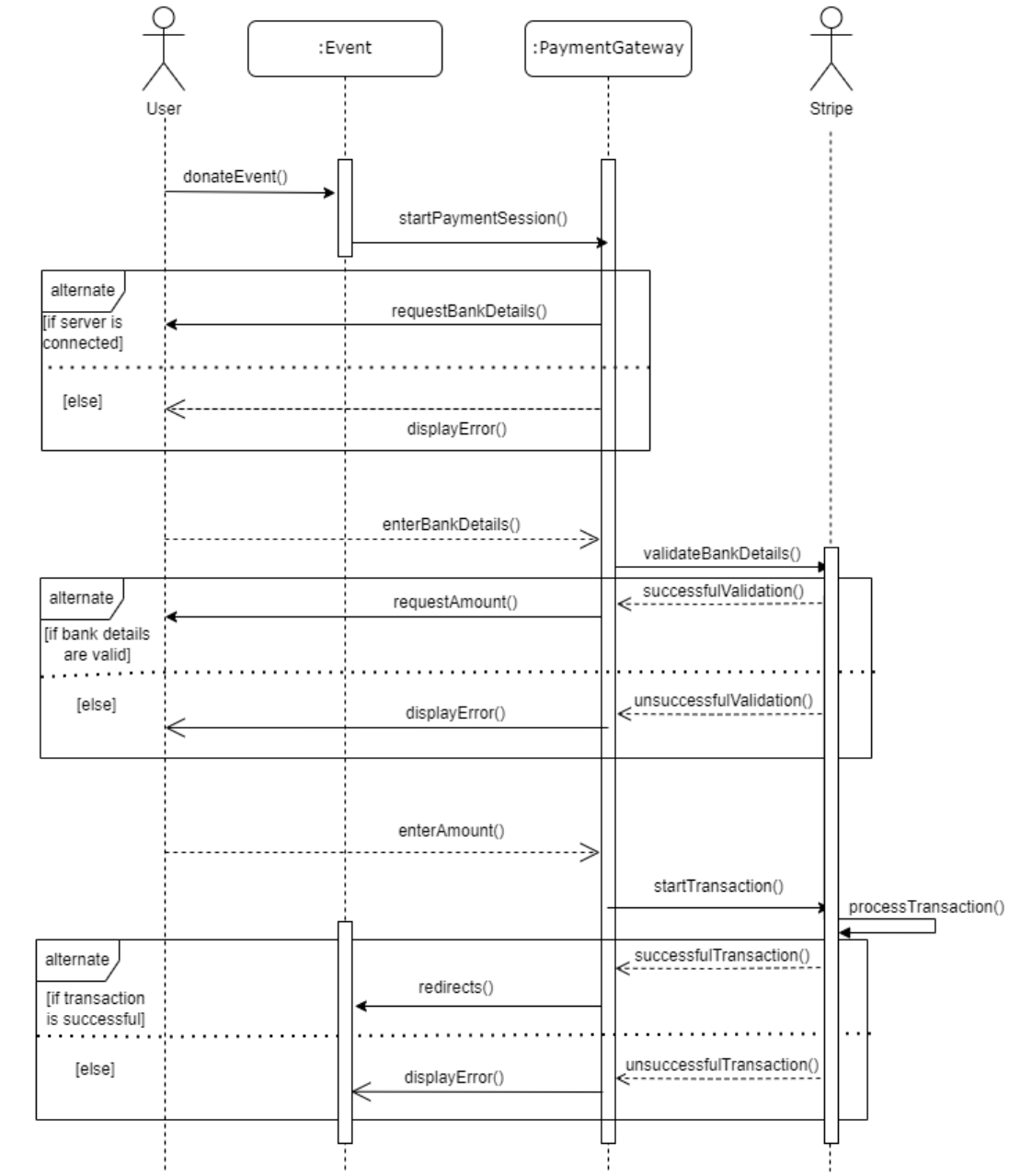
- It emphasizes the structural aspects of an interaction diagram – how lifeline connects.
- Its syntax is similar to that of sequence diagram except that lifeline don't have tails.
- Messages passed over sequencing is indicated by numbering each message hierarchically.
- Compared to the sequence diagram communication diagram is semantically weak.
- Object diagrams are special case of communication diagram.
- It allows you to focus on the elements rather than focusing on the message flow as described in the sequence diagram.
- Sequence diagrams can be easily converted into a collaboration diagram as collaboration diagrams are not very expressive.
- While modeling collaboration diagrams w.r.t sequence diagrams, some information may be lost.

Drawbacks of a Collaboration Diagram:

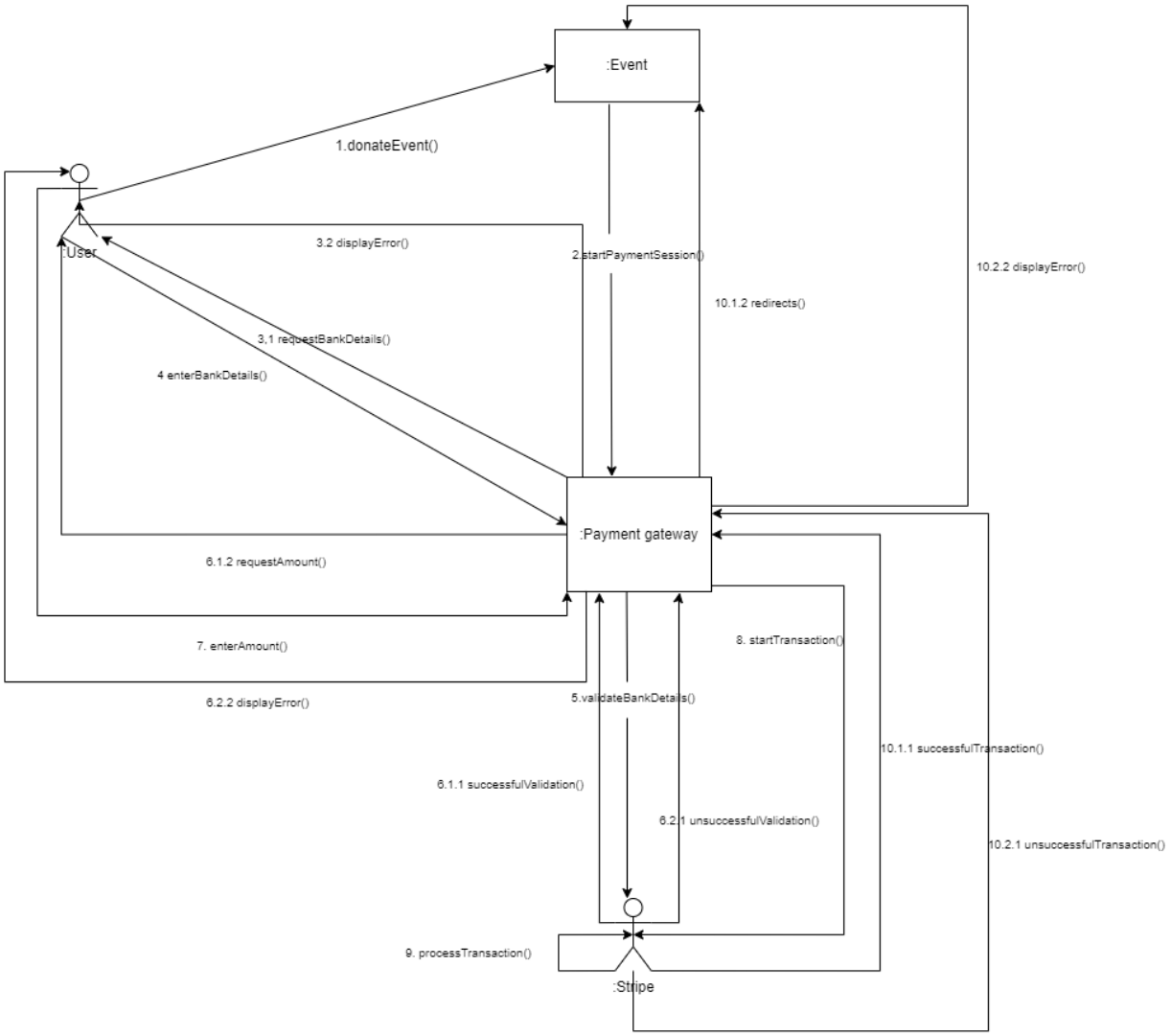
- Collaboration diagrams can become complex when too many objects are present within the system.
- It is hard to explore each object inside the system.
- Collaboration diagrams are time consuming.
- The object is destroyed after the termination of a program.
- The state of an object changes momentarily, which makes it difficult to keep track of every single change the occurs within an object of a system.

Implementation:

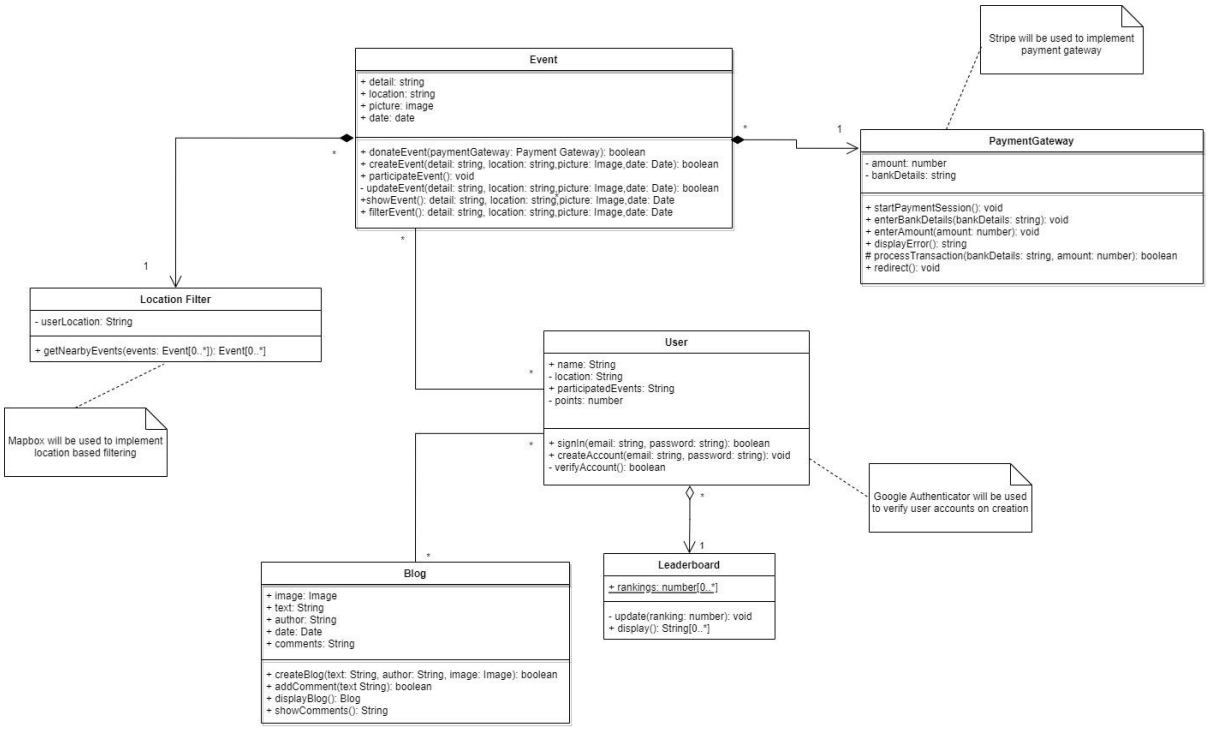
Sequence Diagram:



Collaboration Diagram:



Class Diagram (For ref):



Conclusion:

Successfully implemented the concepts of sequence diagram and collaboration diagram to make them for our project WeChange. This helped us understand the concepts of sequence diagram and collaboration diagram, where they are used, their advantages and limitations.