Name: Khushal Dhanuka

Roll No: 2020300007

Batch A

TE COMPS

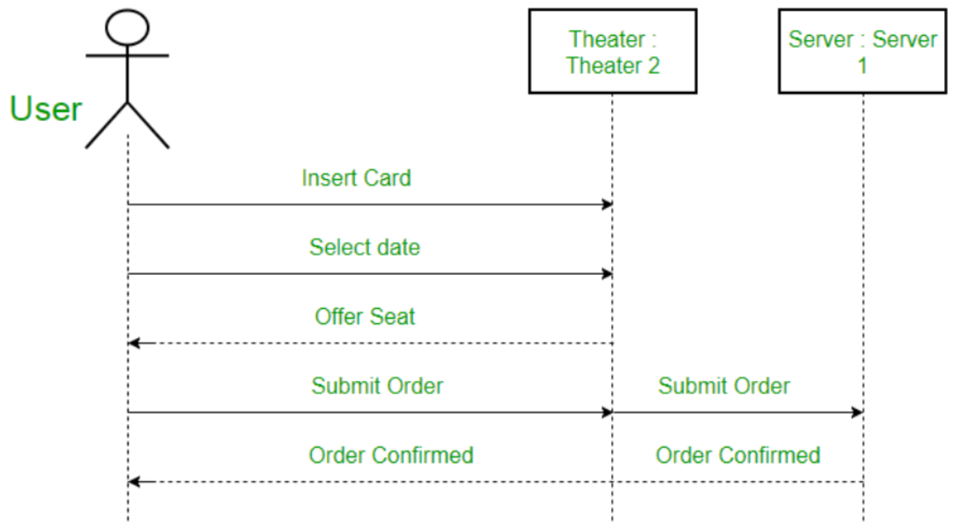
EXPERIMENT NO:3

SEQUENCE AND COLLABORATION DIAGRAMS

THEORY:

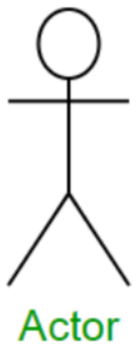
Sequence Diagram:

Unified Modeling 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 behavior 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.

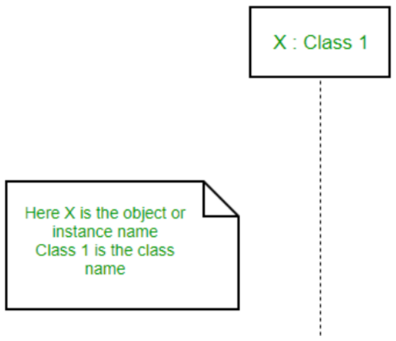


Sequence diagrams Notations:

- **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. 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.

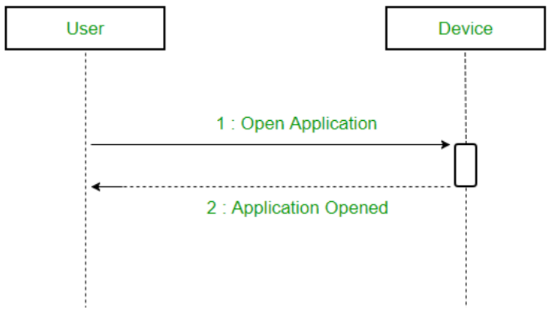


- **Lifelines** – A lifeline is a named element which depicts an individual participant in a sequence diagram. So basically each instance in a sequence diagram is represented by a lifeline. Lifeline elements are located at the top in a sequence diagram. The standard in UML for naming a lifeline follows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

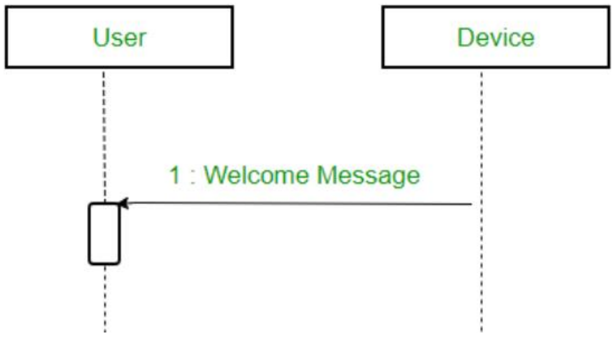

X : Class 1
Here X is the object or instance name Class 1 is the class name

- **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 :

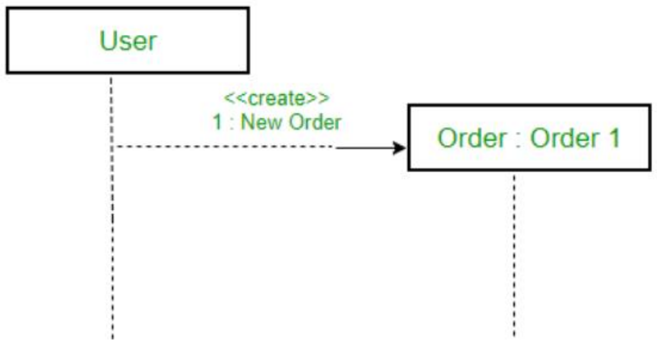
1) **Synchronous messages** – A synchronous message waits for a reply before the interaction can move forward. The sender waits until the receiver has completed the processing of the message. The caller continues only when it knows that the receiver has processed the previous message i.e. it receives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



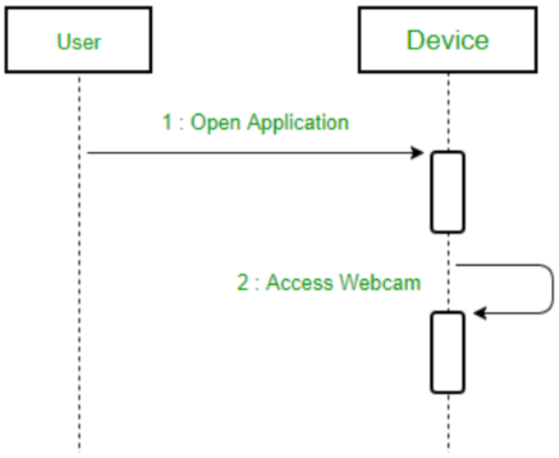
2) **Asynchronous Messages** – An asynchronous message does not wait for a reply from the receiver. The interaction moves forward irrespective of the receiver processing the previous message or not. We use a lined arrow head to represent an asynchronous message.



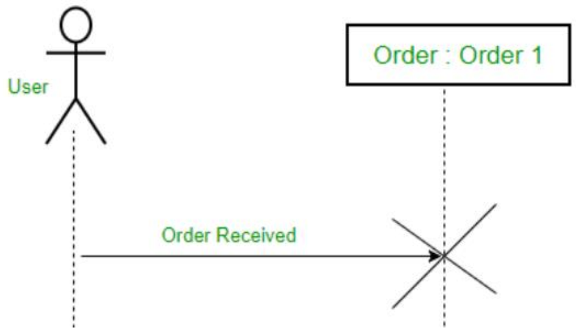
3) **Create message** – We use a Create message to instantiate a new object in the sequence diagram. There are situations when a particular message call requires the creation of an object. It is represented with a dotted arrow and create word labelled on it to specify that it is the create Message symbol.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



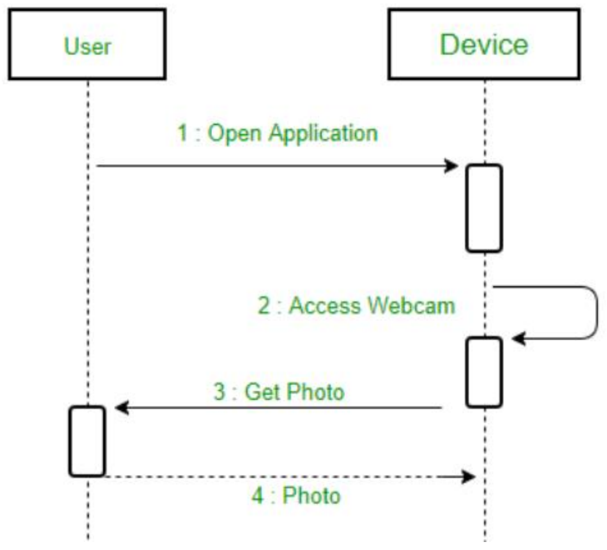
4) **Self Message** – Certain scenarios might arise where the object needs to send a message to itself. Such messages are called Self Messages and are represented with a U shaped arrow.



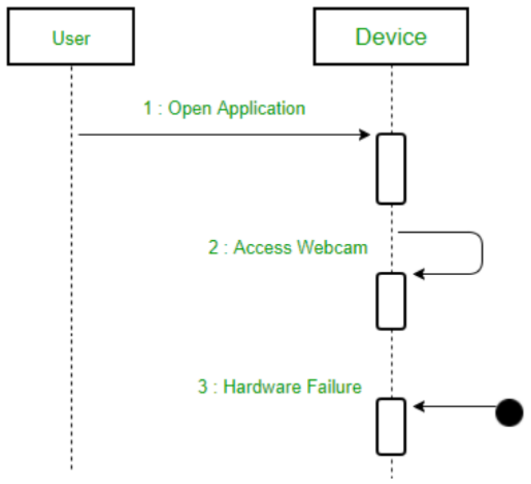
5) **Delete Message –** We use a Delete Message to delete an object. When an object is deallocated memory or is destroyed within the system we use the Delete Message symbol. It destroys the occurrence of the object in the system.It is represented by an arrow terminating with a x. For example – In the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



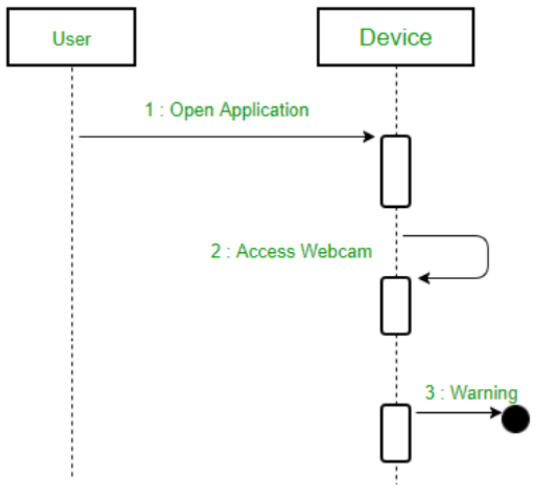
6) **Reply Message –** Reply messages are used to show the message being sent from the receiver to the sender. We represent a return/reply message using an open arrowhead with a dotted line. The interaction moves forward only when a reply message is sent by the receiver.



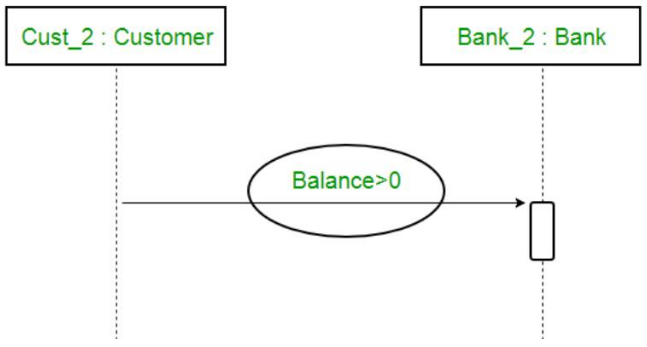
7) **Found Message –** A Found message is used to represent a scenario where an unknown source sends the message. It is represented using an arrow directed towards a lifeline from an end point.



8) **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.



- **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.



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.

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 Collaboration Diagram:

- **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.
- **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.
- **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.
- **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.

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:

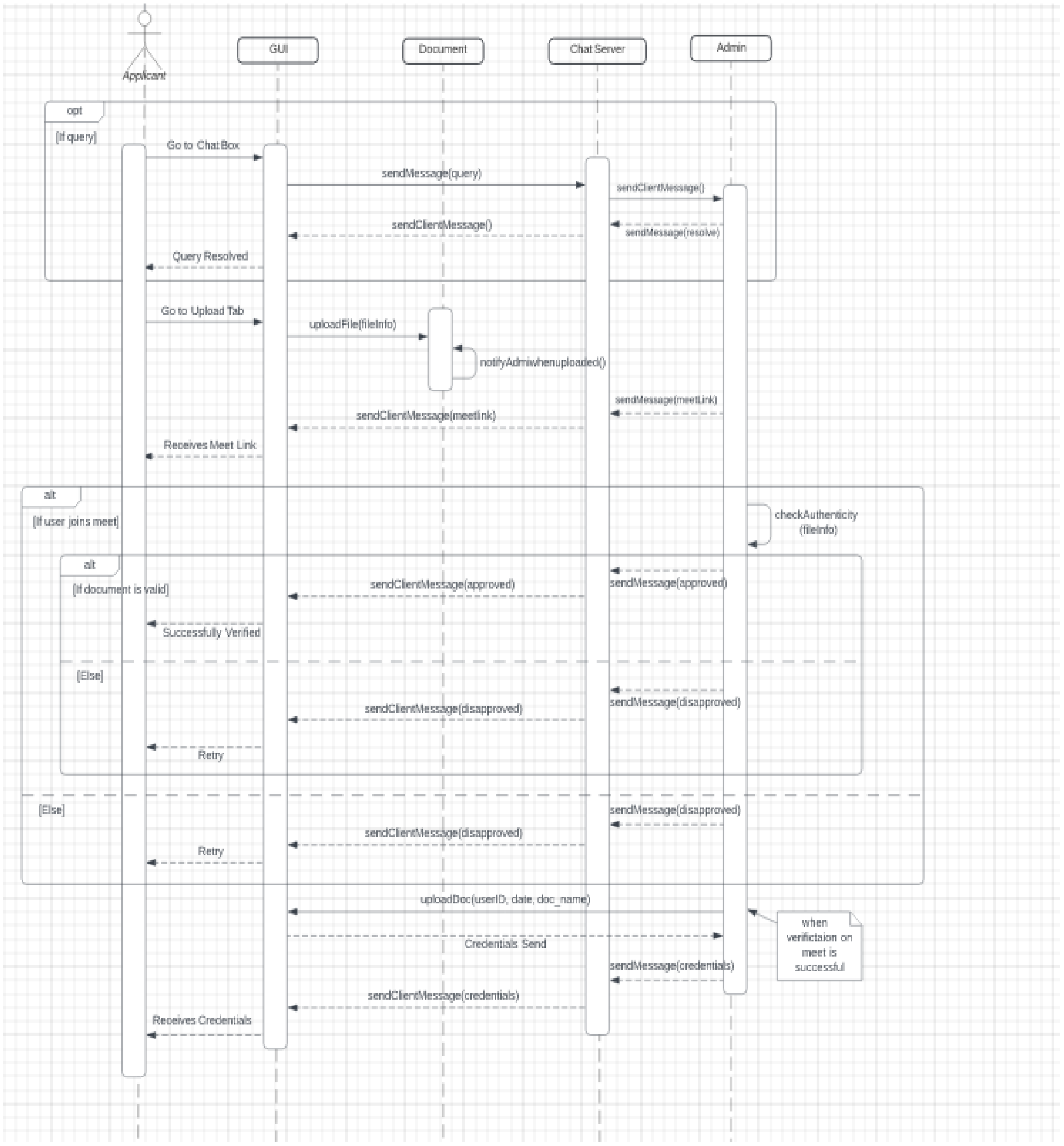
- It emphasizes the structural aspects of an interaction diagram – how lifeline connects.
- Its syntax is similar to that of sequence diagram except that lifeline don’t have tails.
- Messages passed over sequencing is indicated by numbering each message hierarchically.
- Compared to the sequence diagram, the communication diagram is semantically weak.
- Object diagrams are special cases of communication diagrams.
- 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:

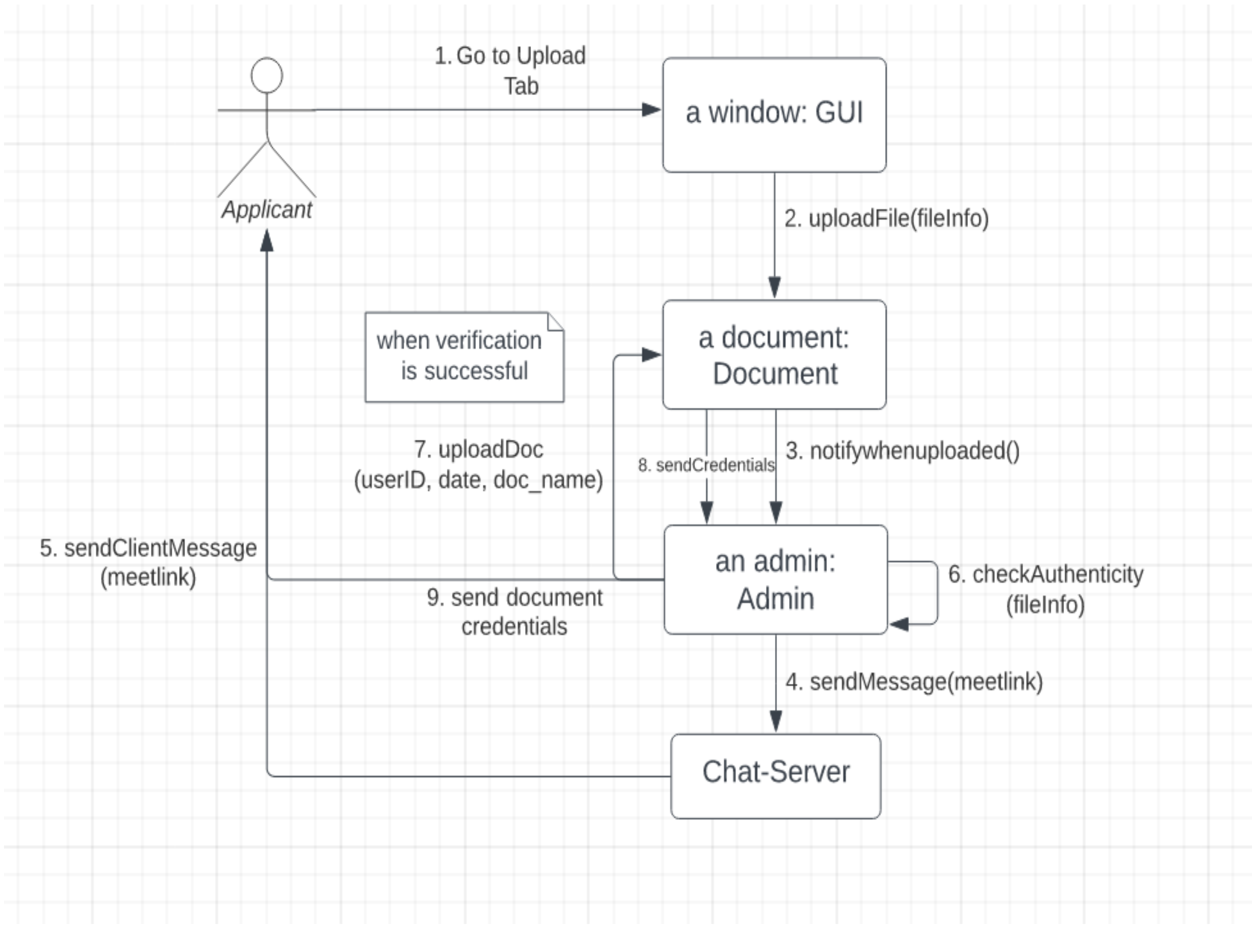
- Collaboration diagrams can become complex when too many objects are present within the system.
- It is hard to explore each object inside the system.
- Collaboration diagrams are time consuming.
- The object is destroyed after the termination of a program.
- The state of an object changes momentarily,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

IMPLEMENTATION:

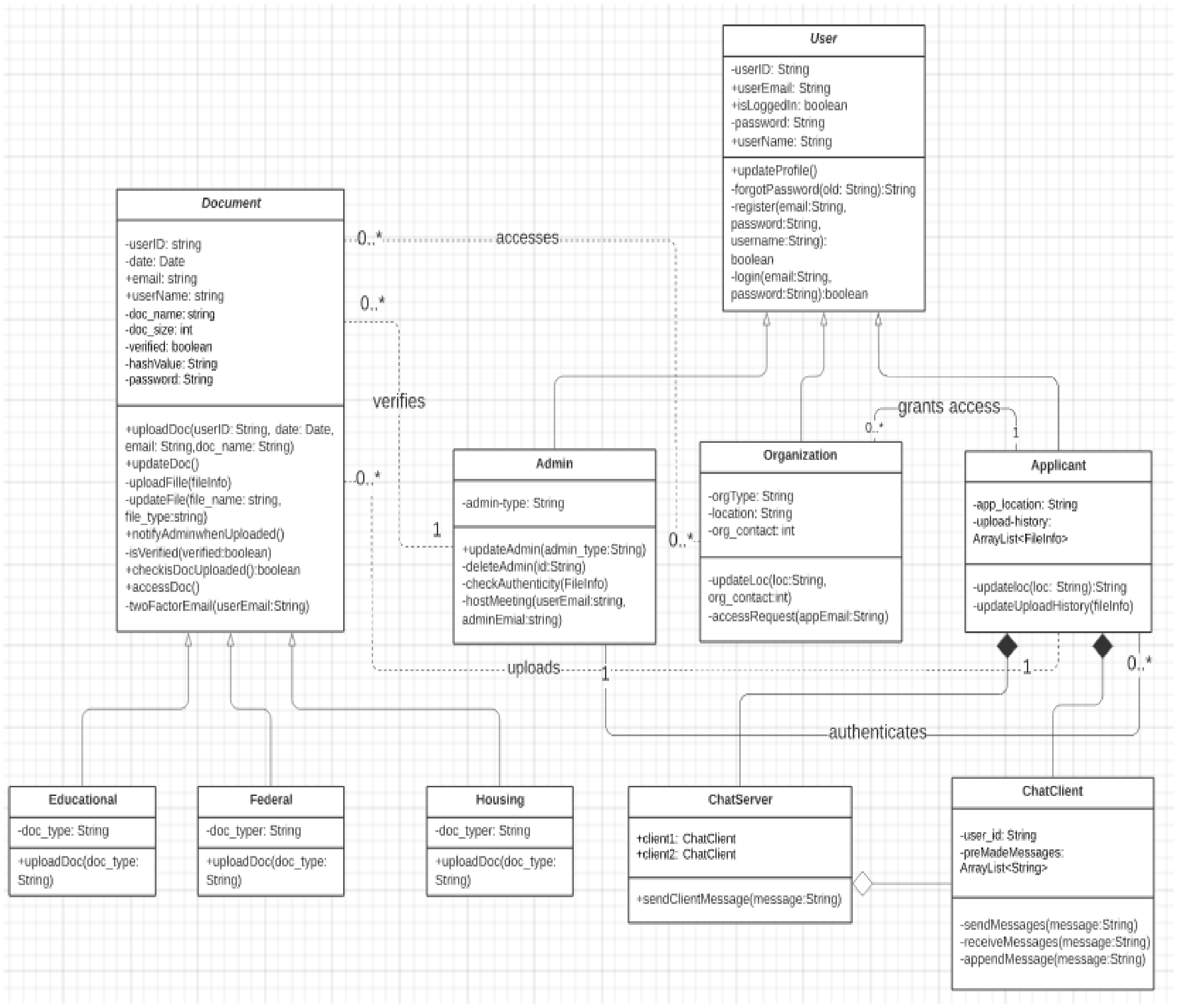
SEQUENCE DIAGRAM:



COLLABORATION DIAGRAM:



CLASS DIAGRAM (FOR REFERENCE):


User
-userID: String
+userEmail: String
+isLoggedIn: boolean
-password: String
+userName: String
+updateProfile()
-forgotPassword(old: String):String
-register(email:String, password:String, username:String): boolean
-login(email:String, password:String):boolean
Document
-userID: string
-date: Date
+email: string
+userName: string
-doc_name: string
-doc_size: int
-verified: boolean
-hashValue: String
-password: String
+uploadDoc(userID: String, date: Date, email: String,doc_name: String)
+updateDoc()
-uploadFille(fileInfo)
-updateFile(file_name: string, file_type:string)
+notifyAdminwhenUploaded()
-isVerified(verified:boolean)
+checkisDocUploaded():boolean
+accessDoc()
-twoFactorEmail(userEmail:String)
0..*
accesses
0..*
verifies
0..*
1
0..*
grants access
0..*
1
Admin
-admin-type: String
+updateAdmin(admin_type:String)
-deleteAdmin(id:String)
-checkAuthenticity(FileInfo)
-hostMeeting(userEmail:string, adminEmial:string)
Organization
-orgType: String
-location: String
-org_contact: int
-updateLoc(loc:String, org_contact:int)
-accessRequest(appEmail:String)
Applicant
-app_location: String
-upload-history: ArrayList<FileInfo>
-updateloc(loc: String):String
-updateUploadHistory(fileInfo)
uploads
1
1
0..*
authenticates
Educational
-doc_type: String
+uploadDoc(doc_type: String)
Federal
-doc_typer: String
+uploadDoc(doc_type: String)
Housing
-doc_typer: String
+uploadDoc(doc_type: String)
ChatServer
+client1: ChatClient
+client2: ChatClient
+sendClientMessage(message:String)
ChatClient
-user_id: String
-preMadeMessages: ArrayList<String>
-sendMessages(message:String)
-receiveMessages(message:String)
-appendMessage(message:String)

CONCLUSION: Successfully implemented the concepts of sequence diagram and collaboration diagram for our DigiNotary project. This helped us understand the concepts of sequence diagram and collaboration diagram, where they are used, their advantages etc.