Name : Kahar Ankit Shriram
Enroll: 2301030400014
Roll: CE-A2-064
Sub : Software engineering
Pr: 05

Case Study on Sequence Diagram

What is a Sequence Diagram ?

A **sequence diagram** is one of the diagrams used in **UML (Unified Modeling Language)**. It visually represents how **objects or components interact with each other in a particular sequence of time** to carry out a process.

Why use Sequence Diagrams?

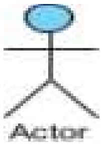
Sequence diagrams are used because they offer a clear and detailed visualization of the interactions between objects or components in a system, focusing on the order and timing of these interactions.Here are some key reasons for using sequence diagrams:

- **Visualizing Dynamic Behavior**: Sequence diagrams depict how objects or systems interact with each other in a sequential manner, making it easier to understand dynamic processes and workflows.
- **Clear Communication**: They provide an intuitive way to convey system behavior, helping teams understand complex interactions without diving into code.
- **Use Case Analysis**: Sequence diagrams are useful for analyzing and representing use cases, making it clear how specific processes are executed within a system.
- **Designing System Architecture**: They assist in defining how various components or services in a system communicate, which is essential for designing complex, distributed systems or service-oriented architectures.

- **Documenting System Behavior**: Sequence diagrams provide an effective way to document how different parts of a system work together, which can be useful for both developers and maintenance teams.
- **Debugging and Troubleshooting**: By modeling the sequence of interactions, they help identify potential bottlenecks, inefficiencies, or errors in system processes.

Sequence Diagram Notations :-

1. Actor

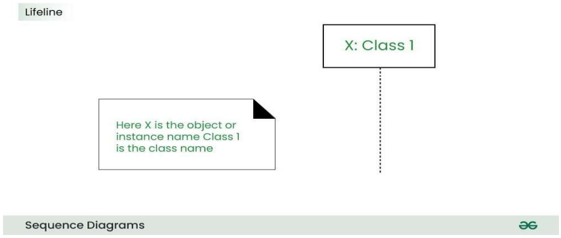


Definition: An **actor** represents a **user or external system** that interacts with the system being modeled.

Example:

➔ In an ATM system: **Customer** is an actor.

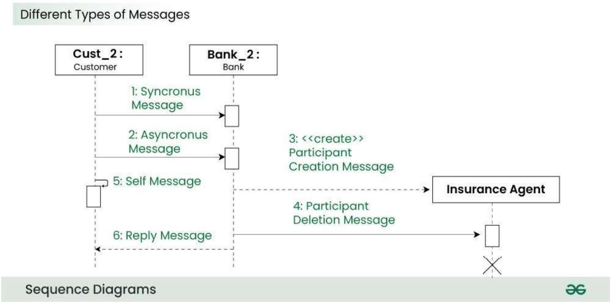
2. Lifelines / Objects


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

Definition: Represent **objects, classes, or system components** that take part in the interaction.
Notation: A **rectangle** with the object's name on top.
A **dashed vertical line** extends downward from it called a **lifeline**.
Lifeline: Represents the existence of that object during the sequence.

Example: ATM, Bank Server, Database , all are lifelines in the ATM withdrawal process.

3. Messages



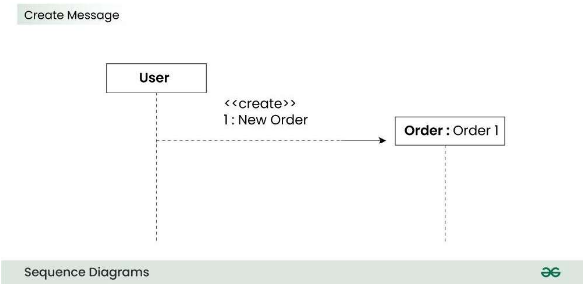
Definition: Messages are arrows that show **communication between lifelines**. Represent **method calls, signals, or data exchanges**.

Types of Messages:

1. **Synchronous message (solid line, filled arrowhead):** Sender waits until the receiver completes the task.
   ➢ **Example**: ATM → Bank Server: *verifyPIN()*

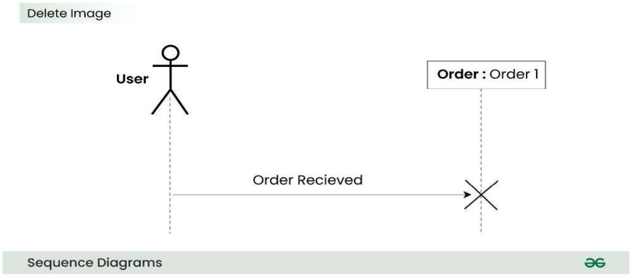
2. **Asynchronous message (solid line, open arrowhead):** Sender doesn't wait for a response.
   ➢ **Example:** User → Notification Service: *sendEmail()*

4. Create message



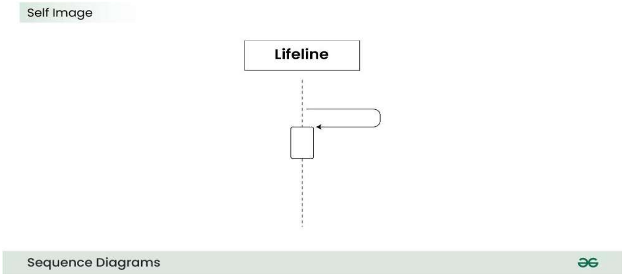
Definition: Represents the creation of a new object during the sequence.
Notation: Solid line with a filled arrowhead pointing to the lifeline box of the new object.

5. Delete Message



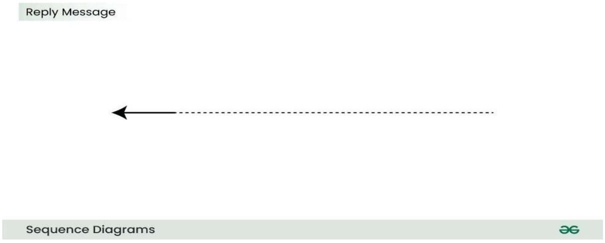
Definition: Represents the destruction of **an object.**
Notation: Message arrow leading to the end of a lifeline, which ends **with an ‘X’** symbol.

6. Self Message



Definition: When an object sends a message **to itself**.
Notation: Arrow starts and ends on the same lifeline, drawn as a **U-shaped arrow**.

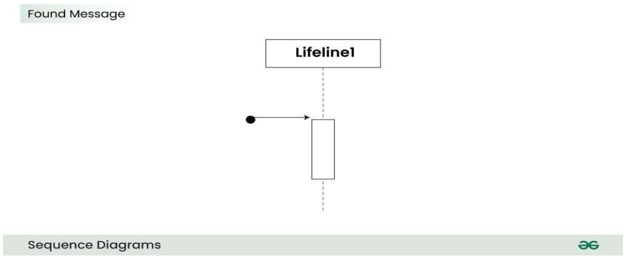
7. Reply Message



Definition: A **Reply Message** (also called **Return Message**) shows the **response** sent back from the receiver object to the sender object **after processing a request**. Drawn as a **dashed horizontal arrow** (instead of solid).

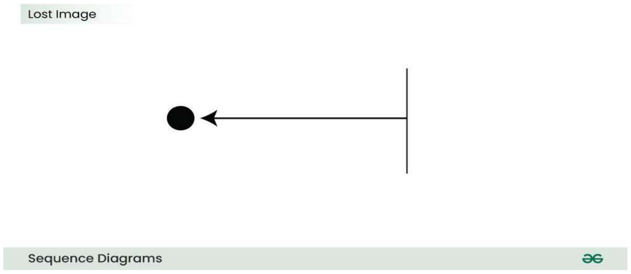
Notation: Arrowhead is **open** (not filled).
It goes from the **receiver’s lifeline back to the sender’s lifeline**.

8. Found Message



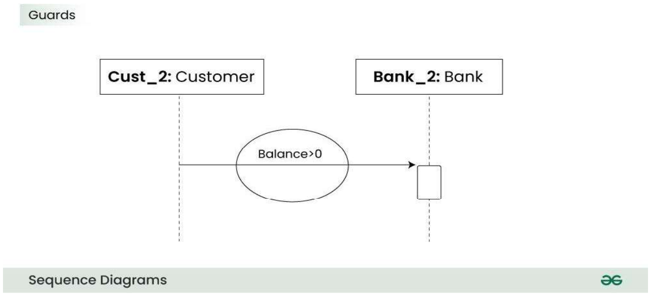
Definition: A message **that starts from outside the system**, where the sender is not known.
Notation: Arrow that starts with a **black dot** at the top of the lifeline.

9. Lost Message



Definition: A message that is **sent but never received**.
Notation: Arrow ending with a **black dot** at the end of the lifeline

10. Guards.



Definition: A **Guard** is a **condition** written inside **square brackets [ ]** that must be **true** for a message or interaction to take place.
Notation: Written inside **[ ]** brackets.
Placed next to a **message arrow** or inside an **interaction fragment (alt/opt/loop)**.

How to create Sequence Diagrams?

1. Identify the Scenario: Choose a specific use case or interaction to model.

2. List Participants: Identify all actors or objects involved (users, systems, components).

3. Define Lifelines: Draw vertical dashed lines for each participant to show their lifespan.

4. Arrange Lifelines: Place participants left to right in order of interaction.

5. Add Activation Bars: Draw narrow rectangles on lifelines to show when participants are active.

6. Draw Messages: Use arrows to represent interactions:

- **Solid = synchronous**
- **Dashed = asynchronous or return**
- **Looping arrows = self-calls**

7. Include Return Messages: Show responses with dashed arrows pointing back.

8. Indicate Order: Number messages or keep them top-to-bottom chronologically.

9. Add Conditions/Loops: Use frames (e.g., alt, loop, opt) for control structures.

10. Represent Parallel Actions (if needed): Use parallel lifelines and messages within a par frame.

11. Review & Refine: Check for accuracy, clarity, and completeness.

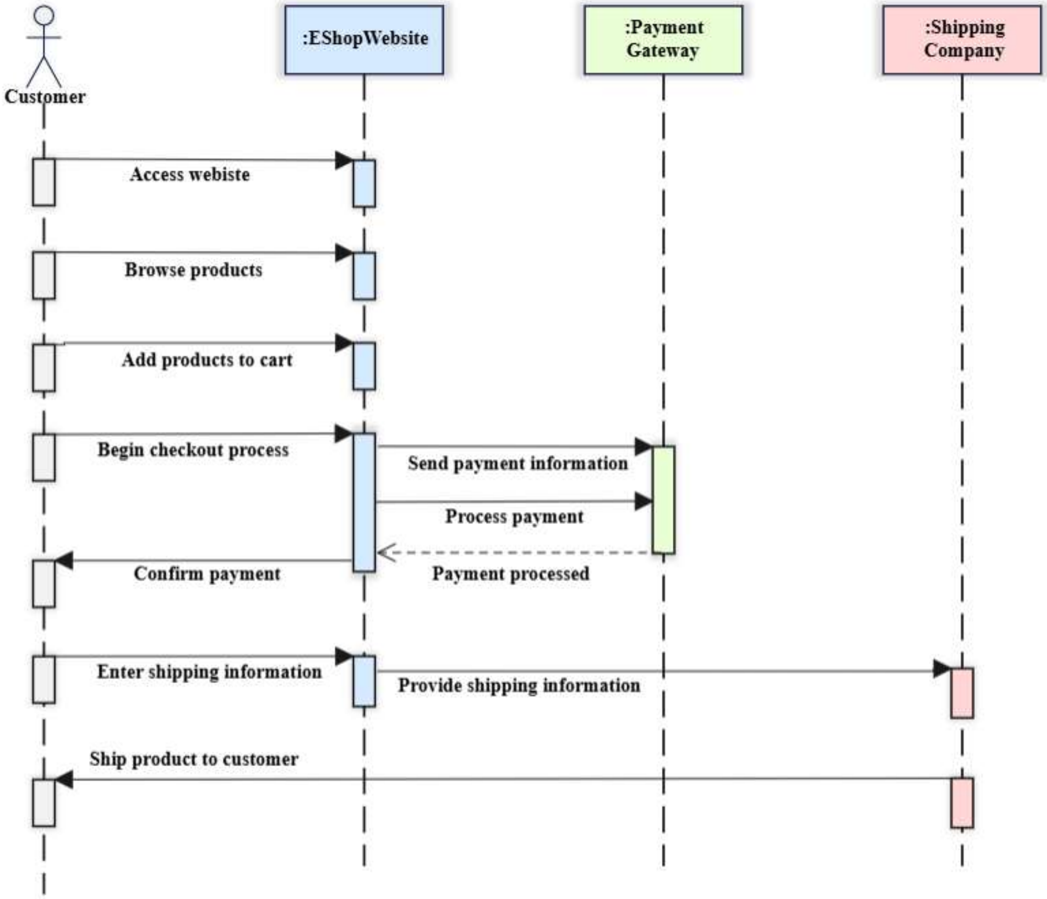
12. Use Tools: Try tools like Lucidchart, Draw.io, PlantUML, or StarUML for clean diagrams.

Benefits of Sequence Diagrams :-

- Show clear **step-by-step interactions** between objects.
- Help in **understanding and clarifying requirements**.

- Useful for **system design** and assigning responsibilities.
- Detect **errors early** before coding.
- Support **test case creation**.
- Serve as good **documentation** for maintenance.

Sequence Diagram for Ecommerce (Purchase) :-


Customer
:EShopWebsite
:Payment Gateway
:Shipping Company
Access webiste
Browse products
Add products to cart
Begin checkout process
Send payment information
Process payment
Confirm payment
Payment processed
Enter shipping information
Provide shipping information
Ship product to customer