CS-3004
SOFTWARE DESIGN AND ANALYSIS

RUBAB JAFFAR
RUBAB.JAFFAR@NU.EDU.PK

Interaction diagrams
Lecture # 25, 26,27

TODAY'S OUTLINE

- Interaction Diagrams
  - Sequence Diagram
  - Collaboration Diagram
- Sequence Diagram
- Sequence Diagram Notations
- Sequence Diagram Example
- Sequence Diagram Advanced Notations
- Sequence Diagram Example

INTERACTION DIAGRAMS

- A type of behavior diagram
- A series of diagrams describing the *dynamic behavior* of an object-oriented system.
  - A set of messages exchanged among a set of objects within a context to accomplish a purpose.
- Often used to model the way a use case is realized through a sequence of messages between objects.
- The purpose of Interaction diagrams is to:
  - Model interactions between objects
  - Assist in understanding how a system (a use case) actually works
  - Verify that a use case description can be supported by the existing classes
  - Identify responsibilities/operations and assign them to classes

INTERACTION DIAGRAMS (CONT.)

- Two types of interaction diagrams
  - Collaboration Diagrams
    - Emphasizes structural relations between objects
  - Sequence Diagram
    - The subject of today

WHAT IS SEQUENCE DIAGRAM?

- A sequence diagram shows how objects interact in a specific situation. Sequence diagrams provide an approximation of time and the general sequence of these interactions by reading the diagram from top to bottom.
- They are also called event diagrams.
- Illustrates how objects interacts with each other.
- Emphasizes time ordering of messages.
- Can model simple sequential flow, branching, iteration, recursion and concurrency.

SEQUENCE DIAGRAMS

- They focus on message sequences, that is, how messages are sent and received between a number of objects.
- Sequence diagrams have two axes:
  - the vertical axis shows time and
  - the horizontal axis shows a set of objects.
- A sequence diagram also reveals the interaction for a specific scenario—a specific interaction between the objects that happens at some point in time during the system's execution (for example, when a specific function is used).

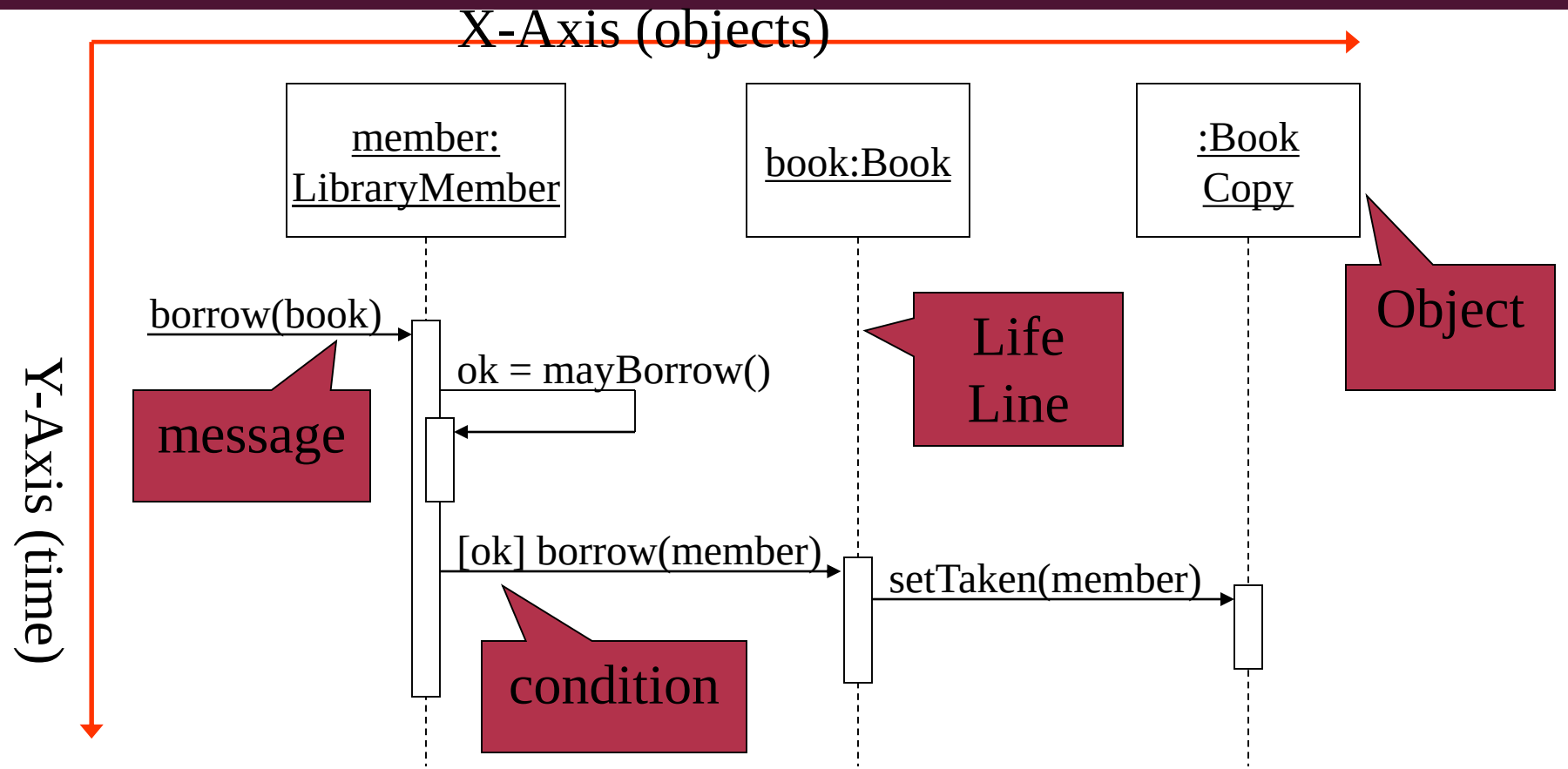
EXAMPLE

- Here are the steps that occur in the use case named 'Borrow Book'.
- The librarian enters the borrower id in the system through UI to get the details about the number of books borrower can have.
- The UI calls the transaction object to calculate the amount that can be borrowed.
- Transactions objects creates and calculate the fine for borrower id .
- Borrowers checked out media and overdue media is also checked by transaction object and then returned to UI to displays this information for the librarian.
- If borrower's amount is equal to zero then UI displays the invalid message.

SEQUENCE DIAGRAMS

- A sequence diagram is enclosed by a rectangular frame with the name of the diagram shown in a pentagram in the upper-left corner prefixed by *sd*.
- *On the* horizontal axis are the objects involved in the sequence. Each is represented by an object rectangle with the object and/or class name underlined.
- The rectangle along with the vertical dashed line, called the object's lifeline, indicates the object's execution during the sequence (that is, messages sent or received and the activation of the object).
- Communication between the objects is represented as horizontal message lines between the objects' lifelines.
- To read the sequence diagram, start at the top of the diagram and read down to view the exchange of messages taking place as time passes.

A SEQUENCE DIAGRAM



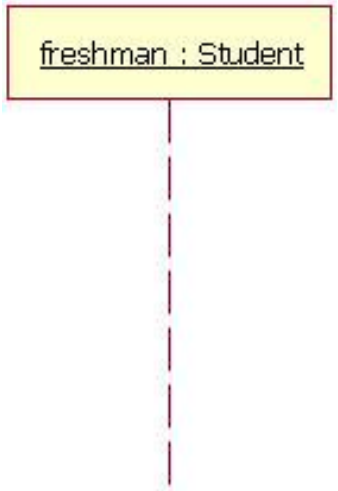
GENERIC AND INSTANCE FORM

- Sequence diagrams can be used in two forms:
  - the generic form and
  - the instance form.
- The instance form describes a specific scenario in detail; it documents one possible interaction. The instance form does not have any conditions, branches, or loops; it shows the interaction for just the chosen scenario.(e.g. Successful opening of an account)
- The generic form describes all possible alternatives in a scenario; therefore branches, conditions, and loops may be included .(e.g. Opening an account)

BASIC NOTATIONS

CLASS ROLES OR PARTICIPANTS

Class roles describe the way an object will behave in context. Use the UML object symbol to illustrate class roles, but don't list object attributes.



An example of the Student class whose instance name is freshman

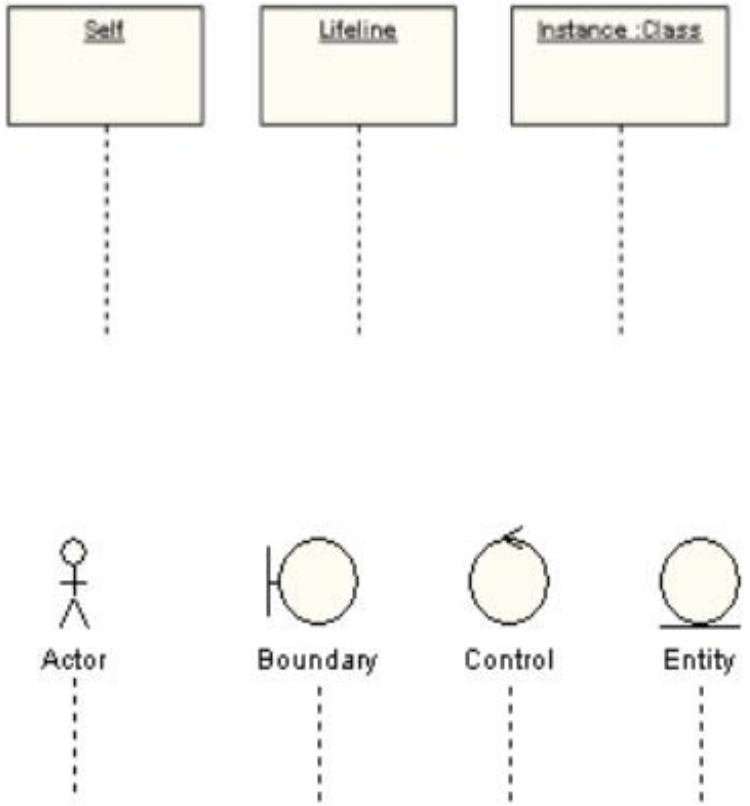
Instance Name : Class Name

ACTOR SYMBOL

- Represented by a stick figure, actors are ent  t are both interactive with and external to the system.

LIFELINES

- An activated object is either executing its own code or is waiting for the return of another object to which it has sent a message.
- The lifeline represents the existence of an object at a particular time; it is drawn as an object icon with a dashed line extending down to the point at which the object stops existing.
- Lifelines indicate the object's presence over time.



ACTIVATION OR EXECUTION OCCURRENCE

- Activation boxes represent the time an object needs to complete a task. When an object is busy executing a process or waiting for a reply message, use a thin gray rectangle placed vertically on its lifeline.

Activation or Execution Occurrence

MESSAGE

- A message is a communication between objects that conveys information with the expectation that action will be taken.
- Messages can be signals, operation invocations, or something similar (for example, remote procedure calls)
- In the sequence diagram, communication between the objects can be shown with distinct message types.

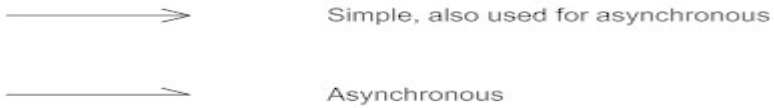
MESSAGES (CONT.)

- A message is represented by an arrow between the life lines of two objects.
  - The time required by the receiver object to process the message is denoted by an *activation-box*.
- A message is labeled at minimum with the message name.
  - Arguments and control information (conditions, iteration) may be included.

TYPES OF MESSAGES

- **Synchronous Message**

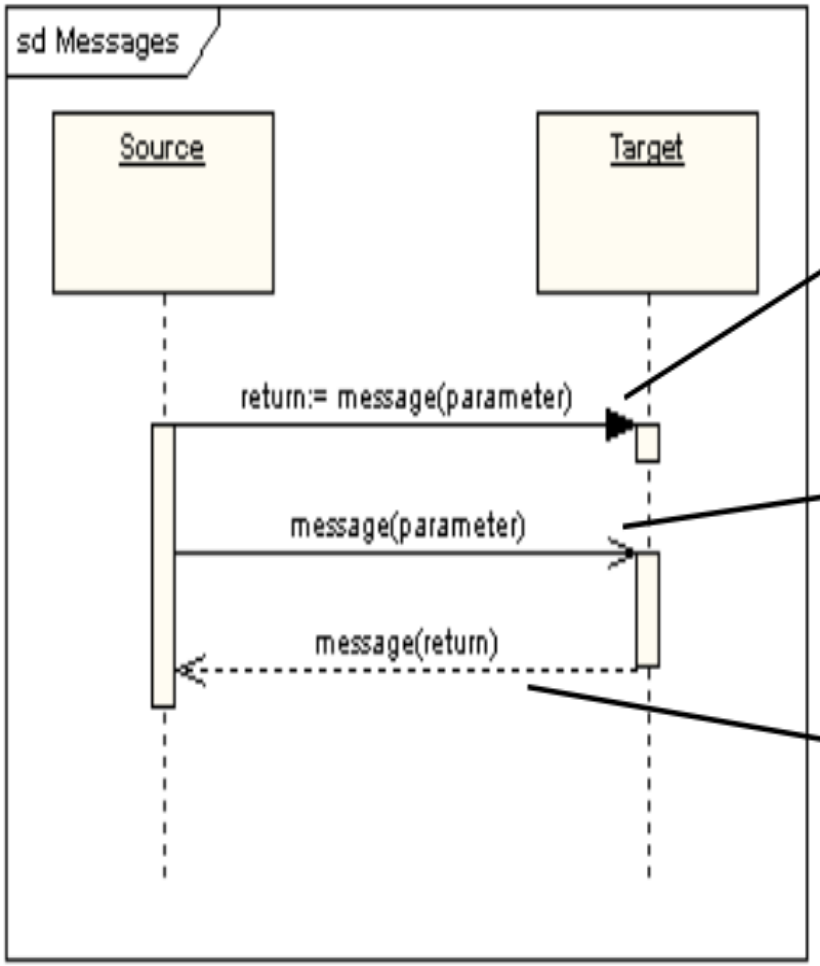
  - A synchronous message indicates wait semantics.
  - A synchronous message requires a response before the interaction can continue. It's usually drawn using a line with a solid arrowhead pointing from one object to another.



- **Asynchronous message**
  - An asynchronous message reveals that the sending object does not wait, but continues to execute immediately after having sent the message (any result is typically sent back as an asynchronous message as well).

- **Reply or Return Message**
  - A reply message is drawn with a dotted line and an open arrowhead pointing back to the original lifeline.

EXAMPLE



synchronous message line
denoted by the solid arrowhead

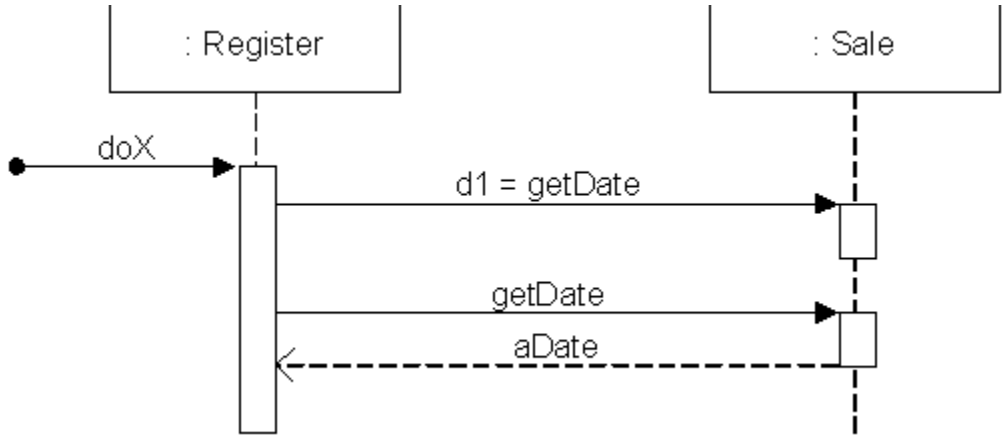
asynchronous message line
denoted by line arrowhead

return message line
denoted by dashed line

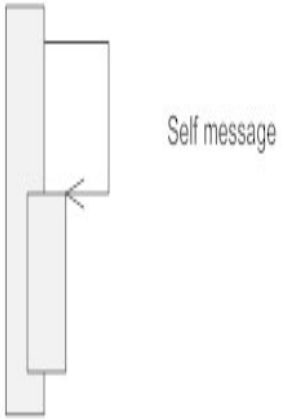
REPLY OR RETURN

There are two ways to show return result from a message:

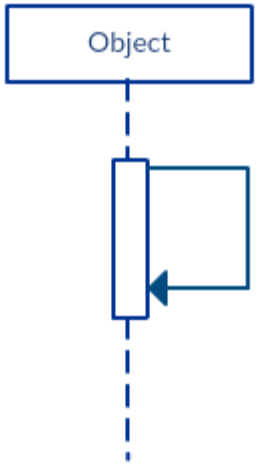
- Using the message syntax returnVar=message(parameter)
- Using a reply(or return) message line at the end of an activation bar.



TYPES OF MESSAGES



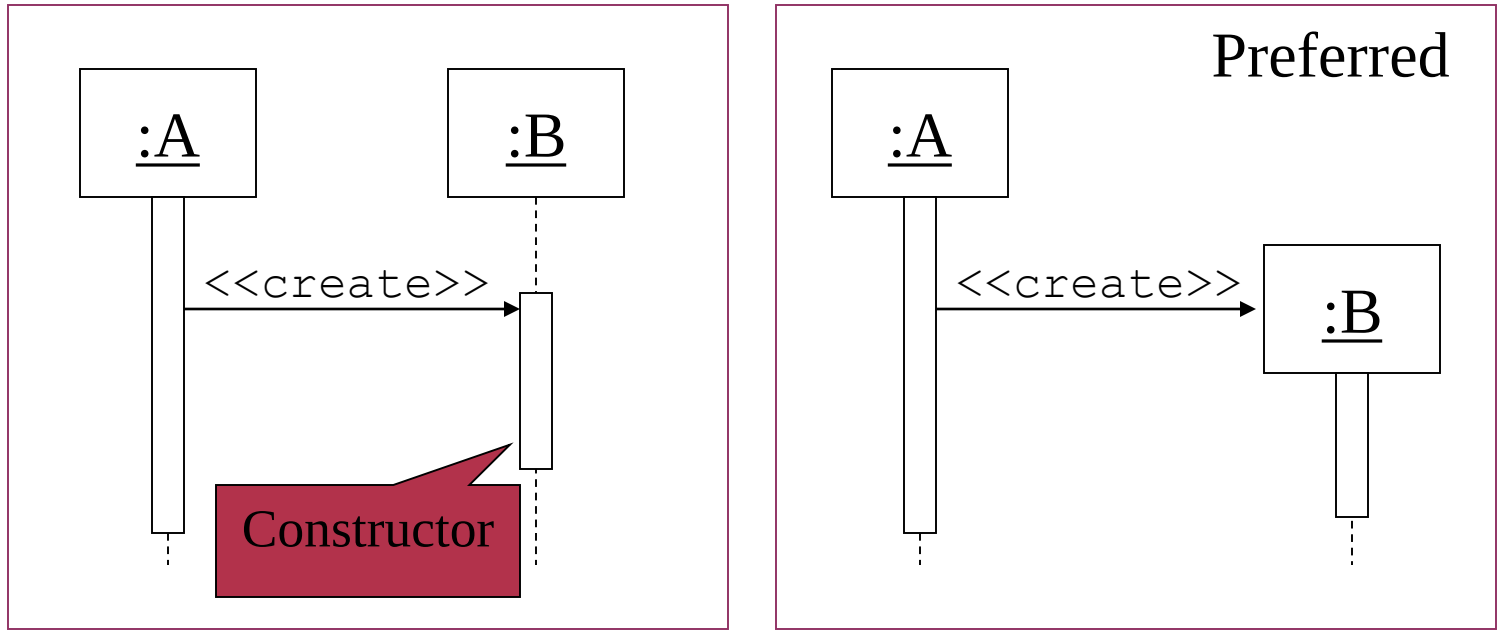
- **Self Message/*Reflexive message***



- A message an object sends to itself, usually shown as a U shaped arrow pointing back to itself

TYPES OF MESSAGES

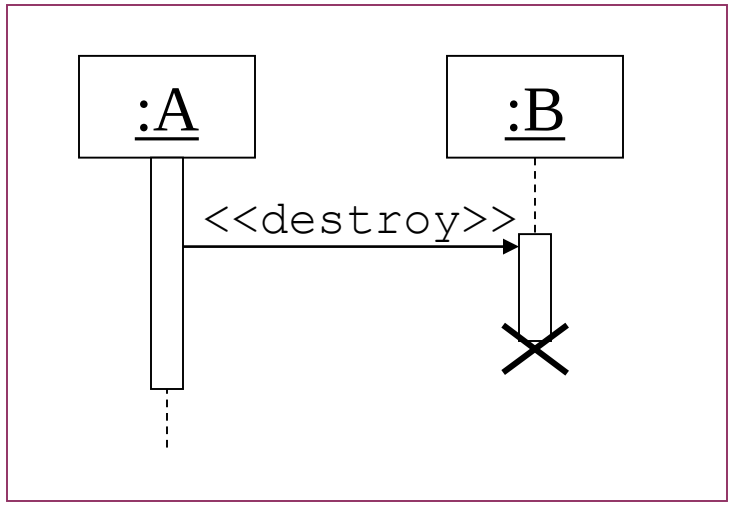
- Create Message
- An object may create another object via a **<<create>>** message.



TYPES OF MESSAGES

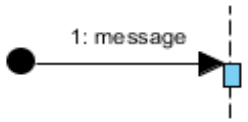
- An object may destroy another object via a **<<destroy>>** message.
  - An object may destroy itself.
  - Avoid modeling object destruction unless memory management is critical.

Objects can be terminated early using an arrow labeled "<< destroy >>" that points to an X. This object is removed from memory. When that object's lifeline ends, you can place an X at the end of its lifeline to denote a destruction occurrence



TYPES OF MESSAGES

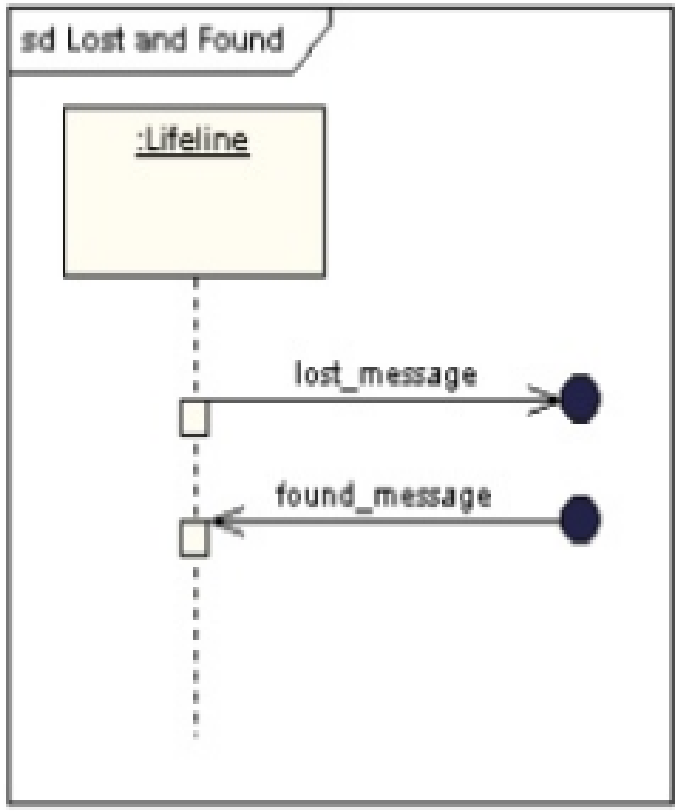
Found Message



A message sent from an unknown recipient or from a sender not shown on the current diagram, shown by an arrow from an endpoint to a lifeline.

- **Lost Message**

A message sent to an unknown recipient that is not shown on the diagram. It's shown by an arrow going from a lifeline to an endpoint, a filled circle.



That is all