Software Engineering

Lecture 9

RIPHAH INTERNATIONAL UNIVERSITY

Agenda

- Sequence Diagram

Sequence Diagram

- Sequence Diagrams are interaction diagrams that detail how operations are carried out.
- Interaction diagrams model important runtime interactions between the parts that make up the system.
- Interactions Diagrams
  1. **Sequence diagrams**
  2. Interaction overview diagrams
  3. Timing diagrams
  4. Communication diagrams

What do Sequence Diagrams model?

- Capture the interaction between objects in the context of a collaboration
- Show object instances that play the roles defined in a collaboration
- Show the order of the interaction visually by using the vertical axis of the diagram to represent time what messages are sent and when
- Show elements as they interact over time, showing interactions or interaction instances
- Do not show the structural relationships between objects

Participants in a Sequence Diagram

- A sequence diagram is made up of a collection of participants
- Participants - the system parts that interact each other during the sequence
- Classes or Objects - each class (object) in the interaction is represented by its named icon along the top of the diagram

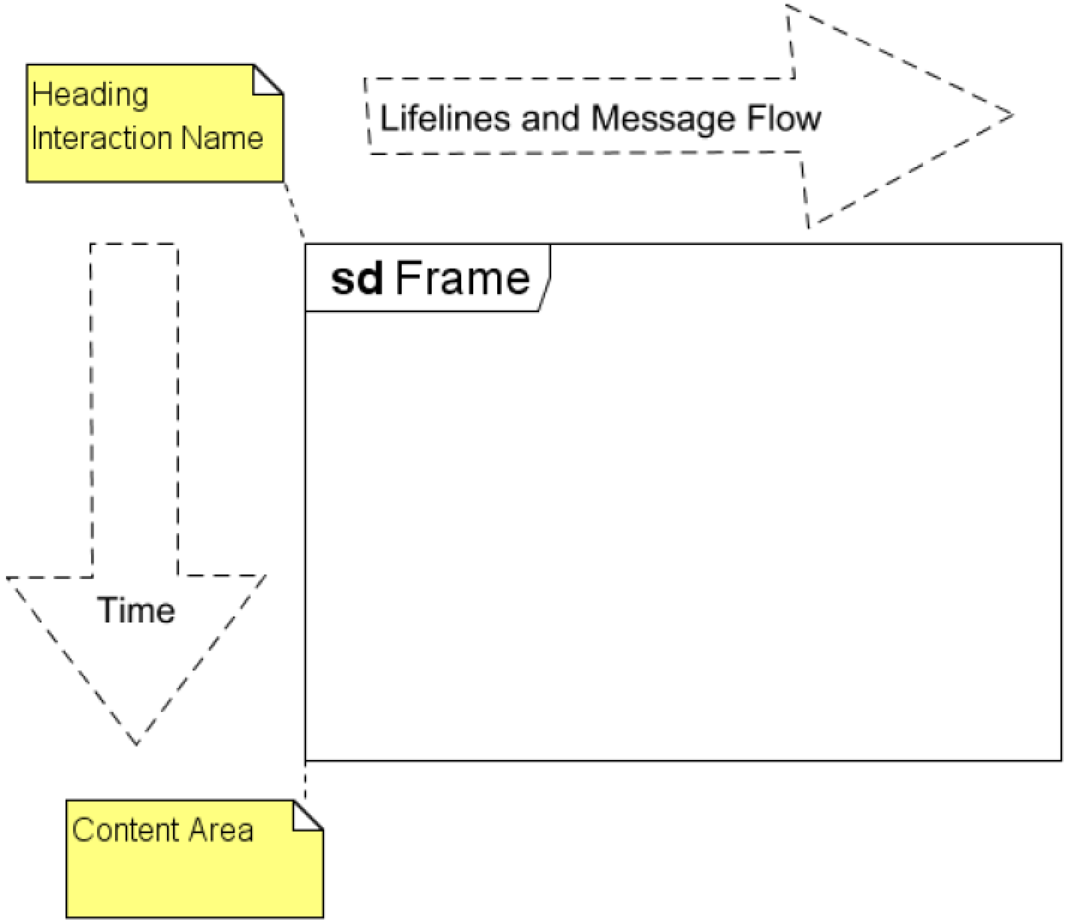
Participants in a Sequence Diagram

- In UML 1.x, participants were usually software objects (instances of classes) in object-oriented programming sense.
- In UML 2.0, as general modeling language, participants are also at the level of system parts.

Sequence Diagrams

- Frames
- Lifelines
- Messages and Focus Control
- Combined Fragments
- Interaction Occurrences
- States
- Textual Annotation
- Tabular Notation

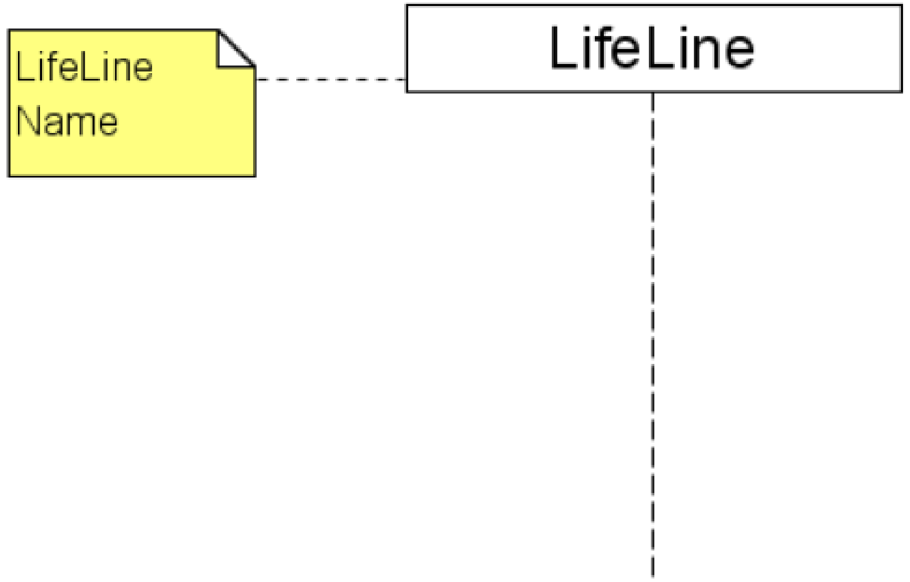
Frames


Heading
Interaction Name
Lifelines and Message Flow
sd Frame
Time
Content Area

Sequence Diagram Dimensions

- Time.
  - The vertical axis represents time proceedings (or progressing) down the page. Note that Time in a sequence diagram is all a about ordering, not duration. The vertical space in an interaction diagram is not relevant for the duration of the interaction.
- Objects.
  - The horizontal axis shows the elements that are involved in the interaction. Conventionally, the objects involved in the operation are listed from left to right according to when they take part in the message sequence. However, the elements on the horizontal axis may appear in any order.

Lifelines

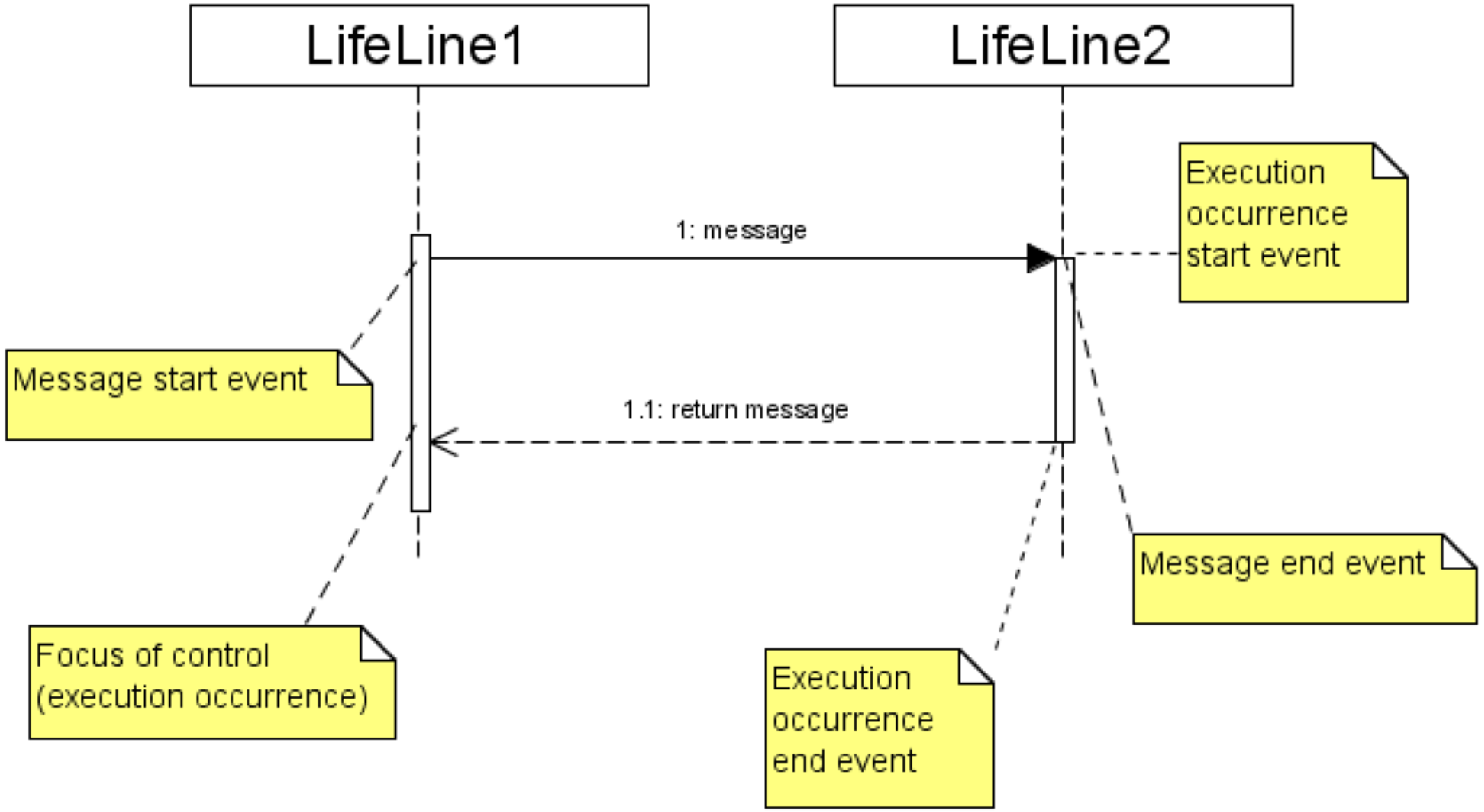


- Sequence diagrams are organised according to time
- Each participant has a corresponding lifeline
- Each vertical dotted line is a lifeline, representing the time that an object exists
- Lifeline name:
- [connectable-element-name][`[`selector']'][:class-name][decomposition]

Lifeline Names

Syntax	Explanation
selecturer	An object named selecturer
selecturer:Lecture r	An object named selecturer of class Lectuer
:Lecturer	An anonymous object of class Lecturer
lecturer[i]	The object lecturer that is selected by the index value I

Messages and Focus of Control


LifeLine1
LifeLine2
1: message
1.1: return message
Execution occurrence start event
Message start event
Focus of control (execution occurrence)
Execution occurrence end event
Message end event

Messages and Focus of Control

- Focus of control (execution occurrence): an execution occurrence (shown as tall, thin rectangle on a lifeline) represents the period during which an element is performing an operation. The top and the bottom of the of the rectangle are aligned with the initiation and the completion time respectively.
- An Event is any point in an interaction where something occurs.

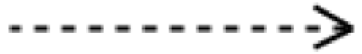
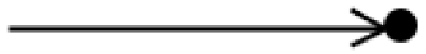
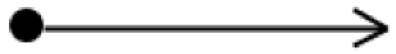
Messages

- Messages (or signals) on a sequence diagram are specified using an arrow from the participant (message caller) that wants to pass the message to the participant (message receiver) that is to receive the message
- A Message (or stimulus) is represented as an arrow going from the sender to the top of the focus of control (i.e., execution occurrence) of the message on the receiver's lifeline

Message Type Notations

Message	Description
───────▶	**Synchronous:** A synchronous message between active objects indicates wait semantics; the sender waits for the message to be handled before it continues. This typically shows a method call.
───────>	**Asynchronous:** With an asynchronous flow of control, there is no explicit return message to the caller. An asynchronous message between objects indicates no-wait semantics; the sender does not wait for the message before it continues. This allows objects to execute concurrently.
<------------	**Reply:** This shows the return message from another message.

Message Type Notations

Message	Description
	**Create:** This message results in the creation of a new object. The message could call a constructor for a class if you are working with Java, for example.
	**Lost:** A lost message occurs whet the sender of the message is known but there is no reception of the message. This message allows advanced dynamic models to built up by fragments without complete knowledge of all the messages in the system. This also allows the modeler to consider the possible impact of a message's being lost.
	**Found:** A found message indicates that although the receiver of the message is known in the current interaction fragment, the sender of the message is unknown.

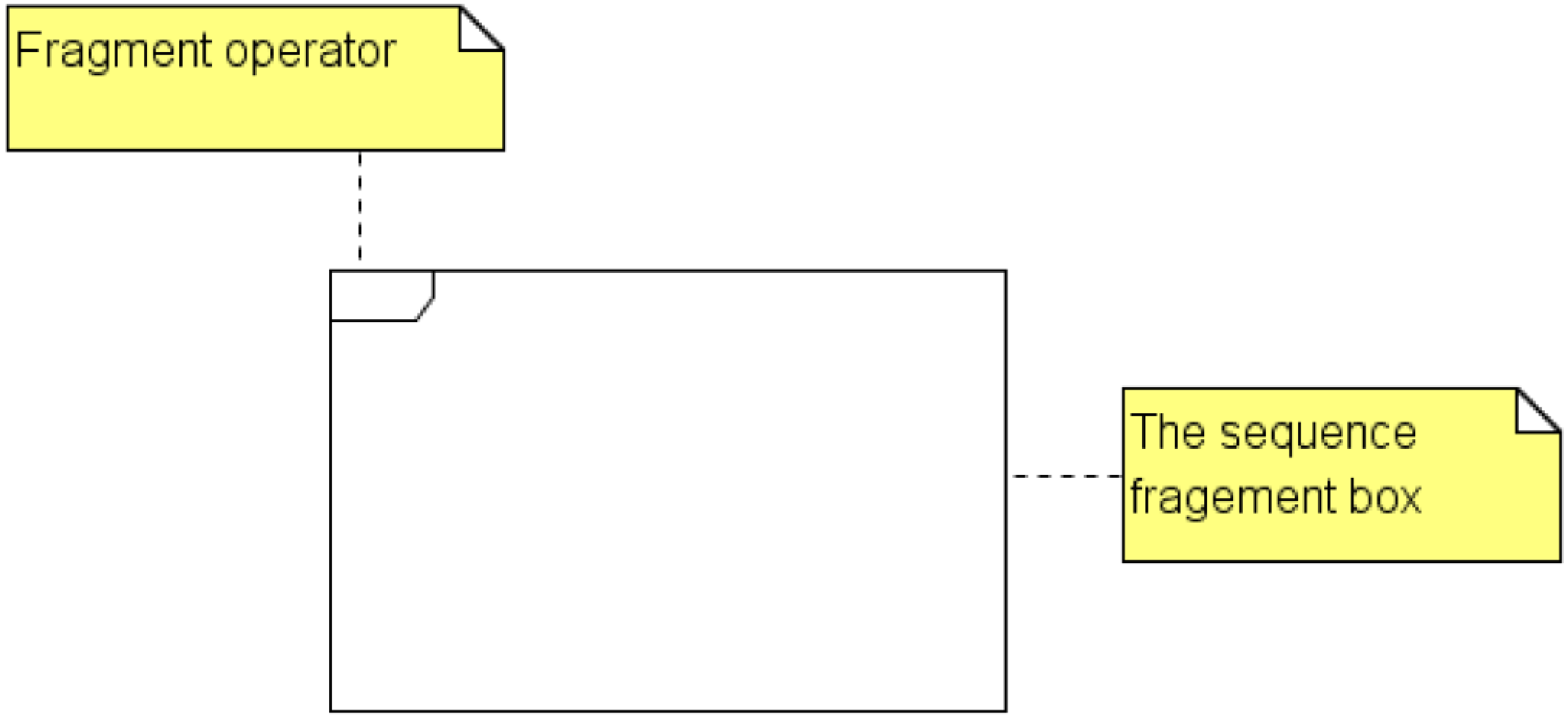
Types of Communications

- **Reflexive Communications:** similar to a reflexive association or link, an element may communicate with itself where communication is sent from the element to itself. Sending messages to itself means an object has two activations simultaneously.
- **Repetitions:** involve repeating a set of messages or stimuli within a generic-form interaction. Messages are grouped together in a rectangle. The expression in square brackets, [ ], is a condition. The asterisk *" means iteration.
- **Conditionality:** branching results in a choice of two different messages (or operation calls) being sent to the same object, the lifeline of the object splits with two activations. The separate lifelines merge back together after the completion of different actions in response to the different messages.
- **Return Values:** often worthwhile to label the return value because it may be used later in the interaction.

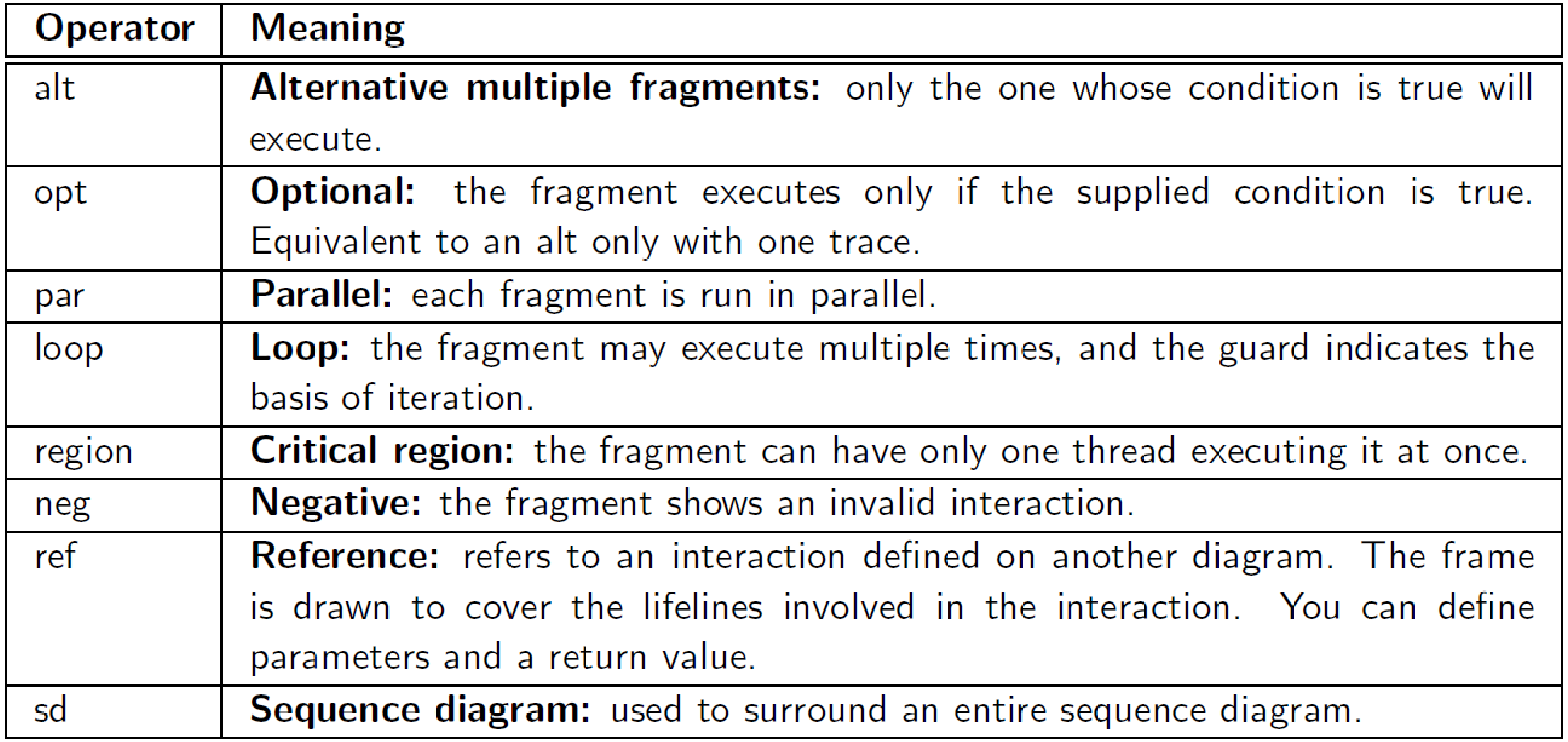
Sequence Fragments

- UML 2.0 introduces sequence (or interaction) fragments
- Sequence fragments make it easier to create and maintain accurate sequence diagrams
- A sequence fragment is represented as a box, called a combined fragment, which encloses a portion of the interactions within a sequence diagram
- The fragment operator (in the top left cornet) indicates the type of fragment
- Fragment types: ref, assert, loop, break, alt, opt, neg

Sequence Fragments



Common Operators for Interaction Frames

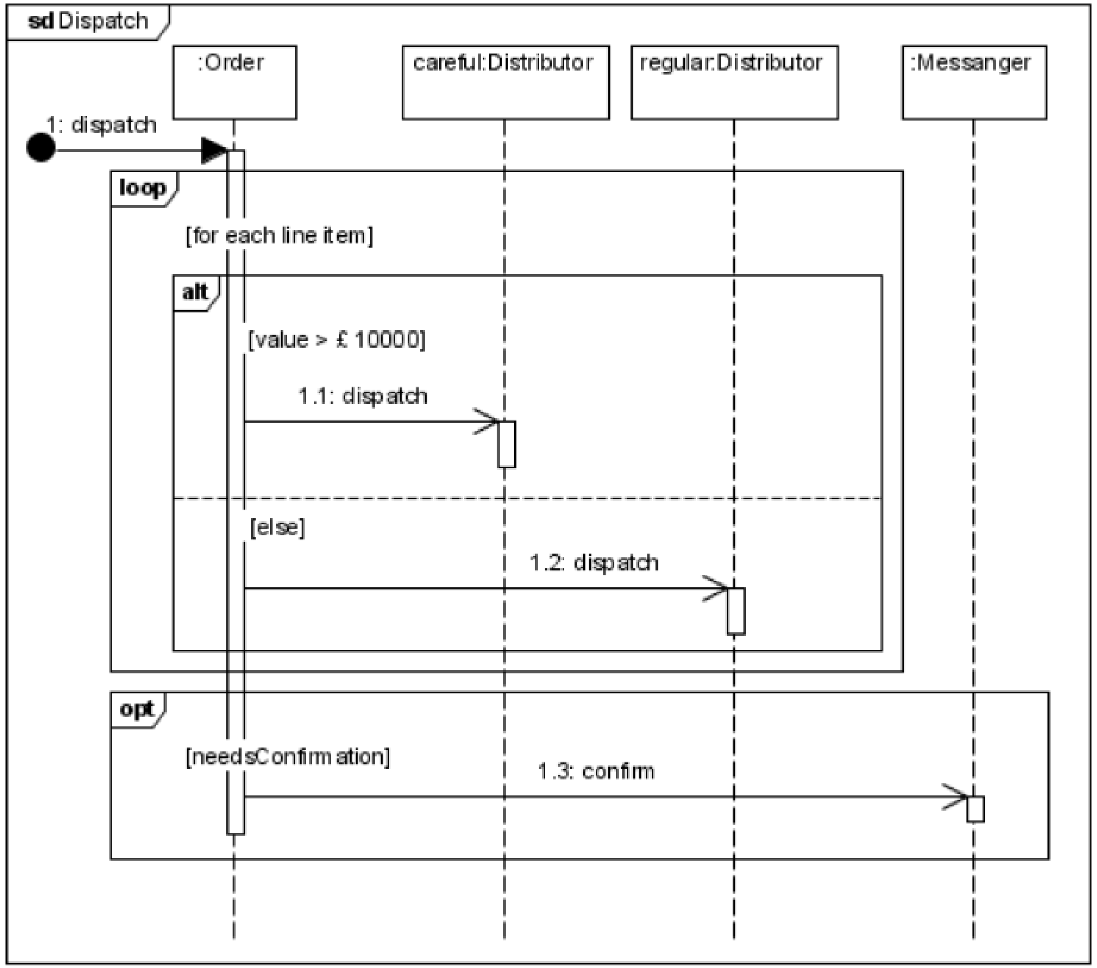


Operator	Meaning
alt	**Alternative multiple fragments:** only the one whose condition is true will execute.
opt	**Optional:** the fragment executes only if the supplied condition is true. Equivalent to an alt only with one trace.
par	**Parallel:** each fragment is run in parallel.
loop	**Loop:** the fragment may execute multiple times, and the guard indicates the basis of iteration.
region	**Critical region:** the fragment can have only one thread executing it at once.
neg	**Negative:** the fragment shows an invalid interaction.
ref	**Reference:** refers to an interaction defined on another diagram. The frame is drawn to cover the lifelines involved in the interaction. You can define parameters and a return value.
sd	**Sequence diagram:** used to surround an entire sequence diagram.

Combined Frames

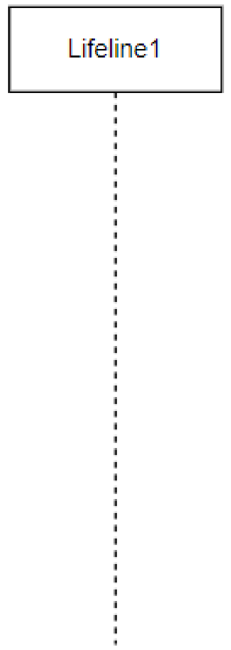
- It is possible to combine frames in order to capture, e.g., loops or branches.
- Combined fragment keywords: alt, opt, break, par, seq, strict, neg, critical, ignore, consider, assert and loop
- Other ways in UML 2.0 of hiding information are by interaction occurrences and continuations

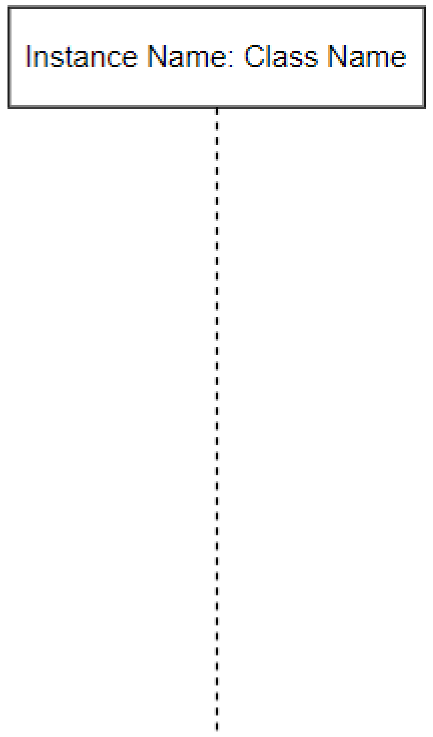
Interaction Frames



Lifeline

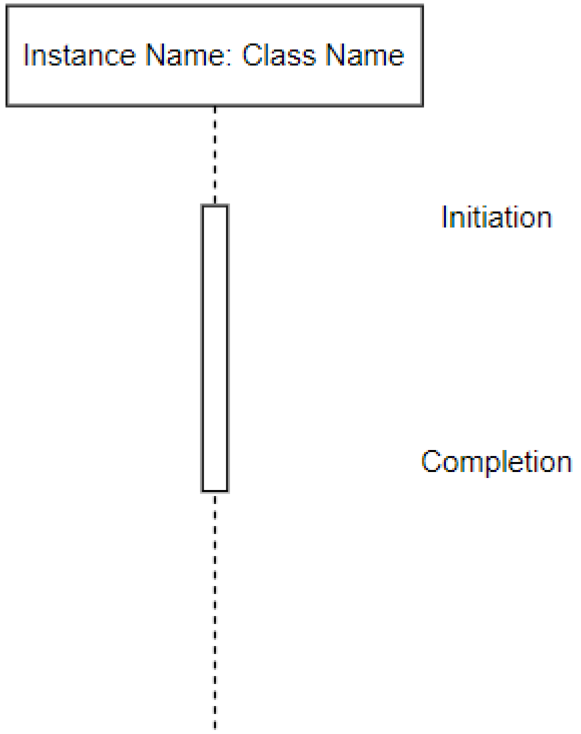
- A lifeline represents an individual participant in the interaction.




Instance Name: Class Name

Activations

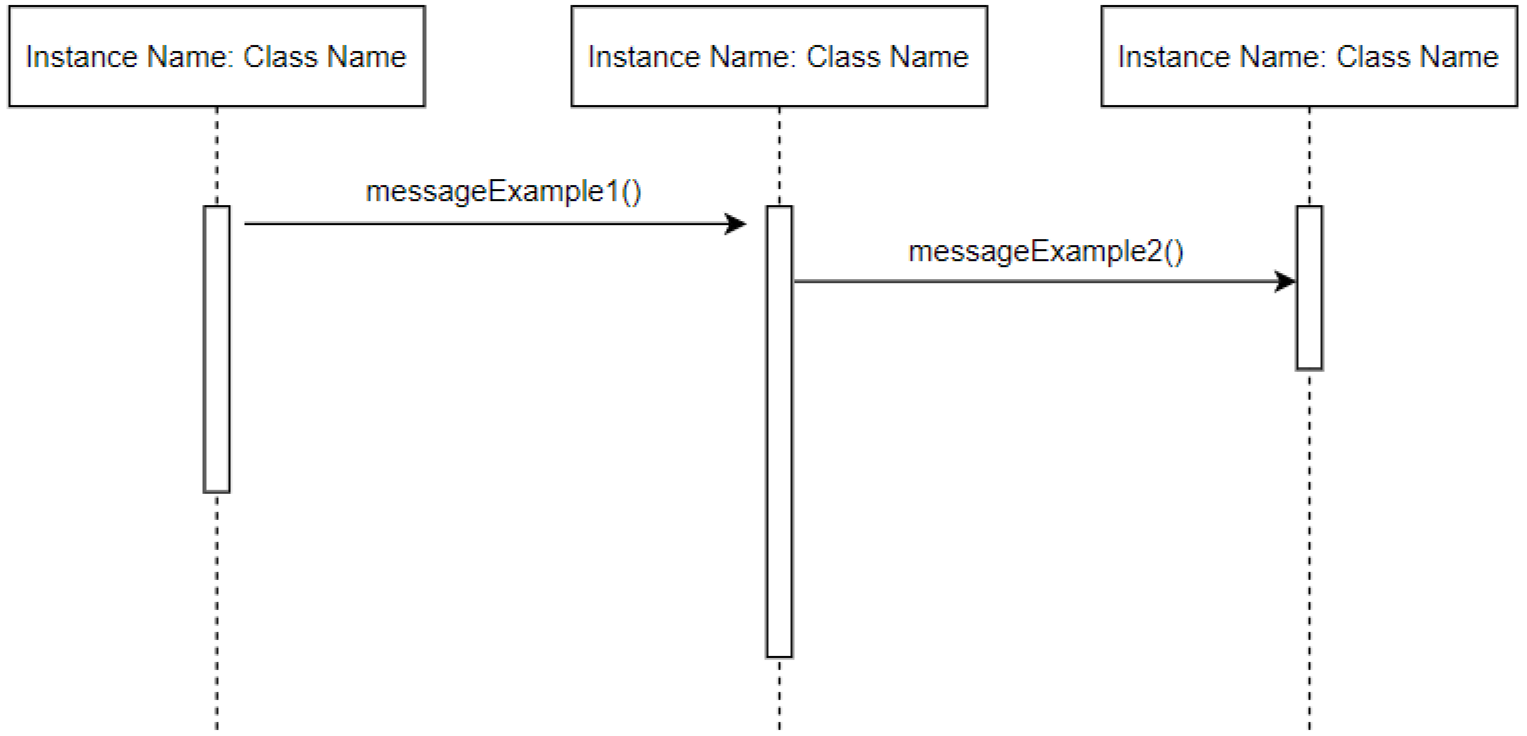
- A thin rectangle box on a lifeline represents the time needed for an object to complete a task.
  - Execution of an operation.


Instance Name: Class Name
Initiation
Completion

Message

- A message defines a particular communication between lifelines of an interaction which means showing how the objects interact with each other by sending and receiving messages.

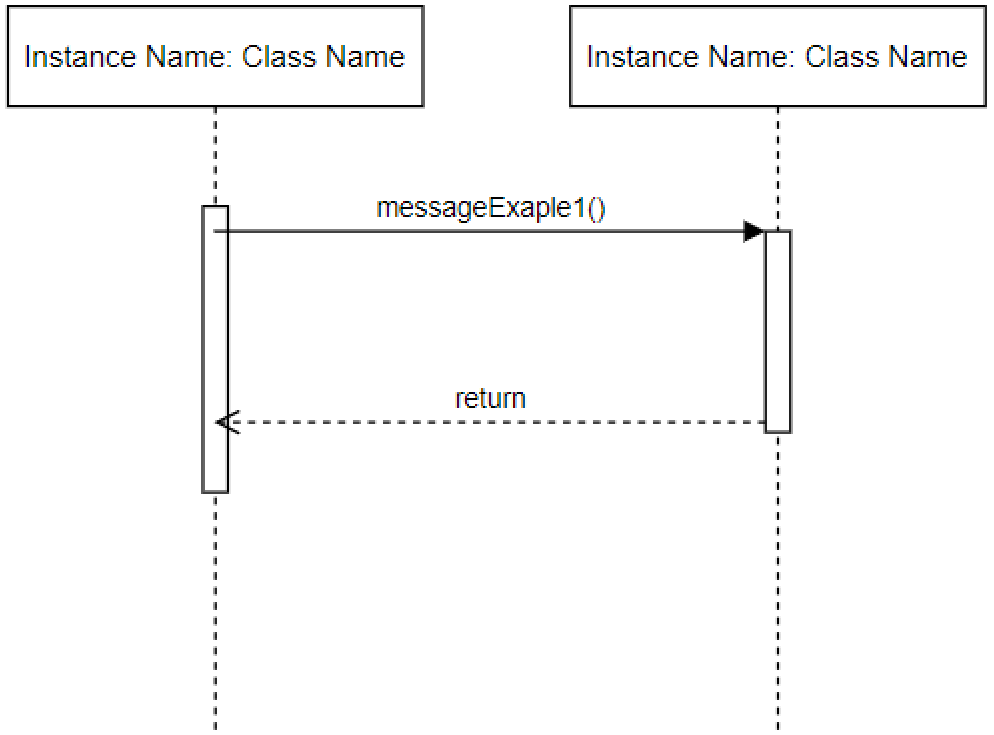
Message


Instance Name: Class Name
Instance Name: Class Name
Instance Name: Class Name
messageExample1()
messageExample2()

Synchronous Messages

- Represented by a solid line with a solid arrowhead. This symbol is used when a sender must wait for a response to a message before it continues.

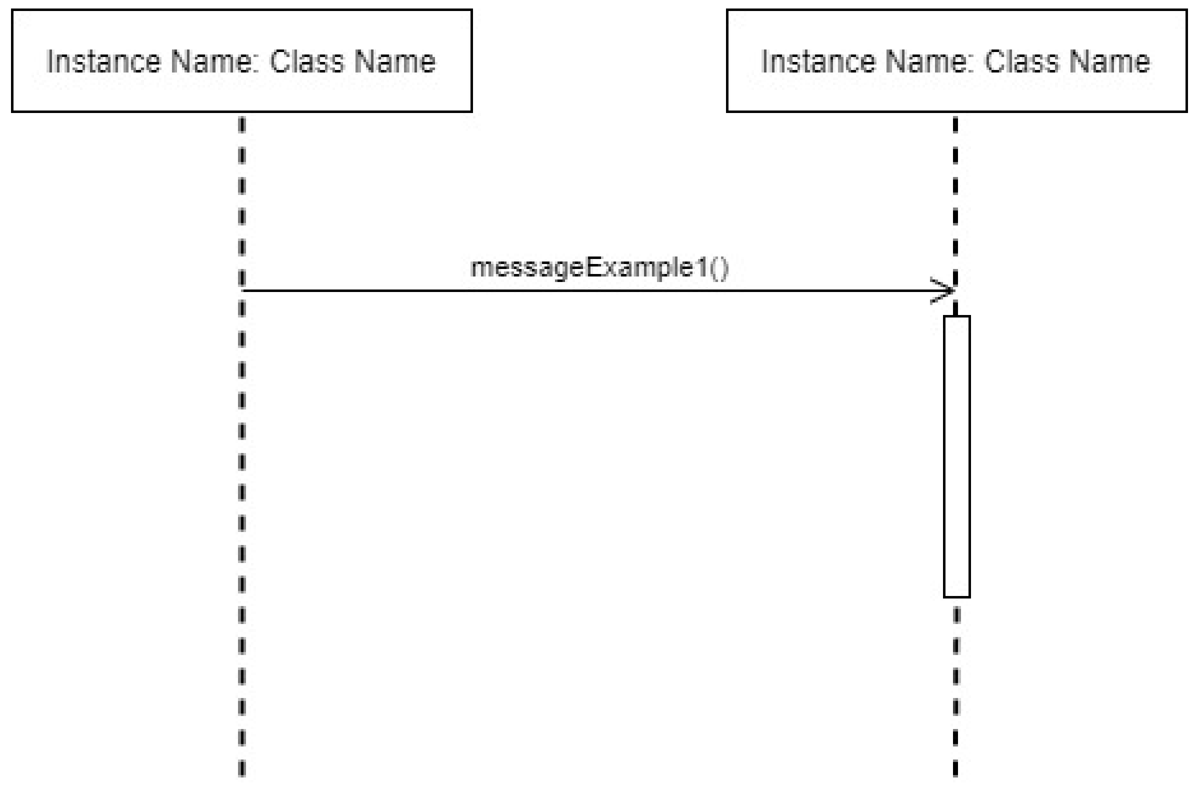
Synchronous Messages



Asynchronous Messages

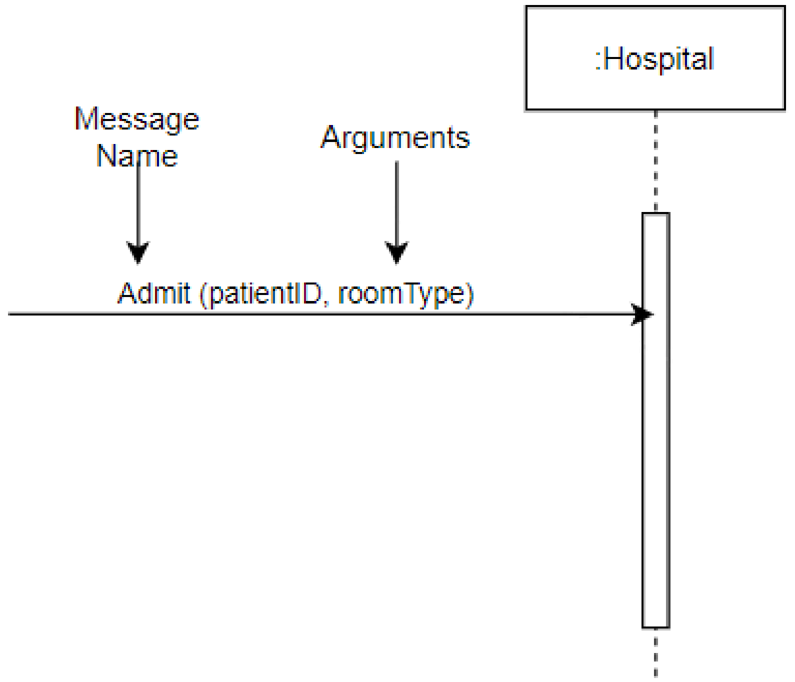
- An asynchronous message does not wait for a response from the receiver before the sender continues

Asynchronous Messages

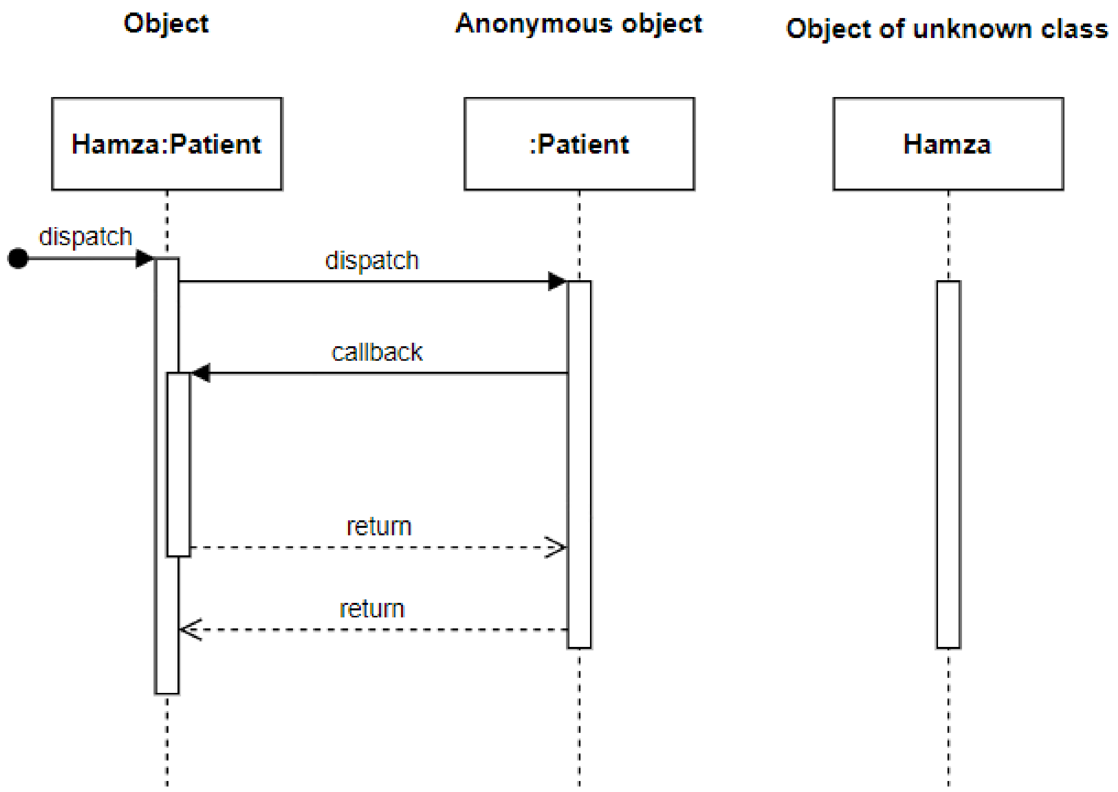


Messages Between Objects

- messages (method calls) indicated by arrow to another object
- –write message name and arguments above arrow



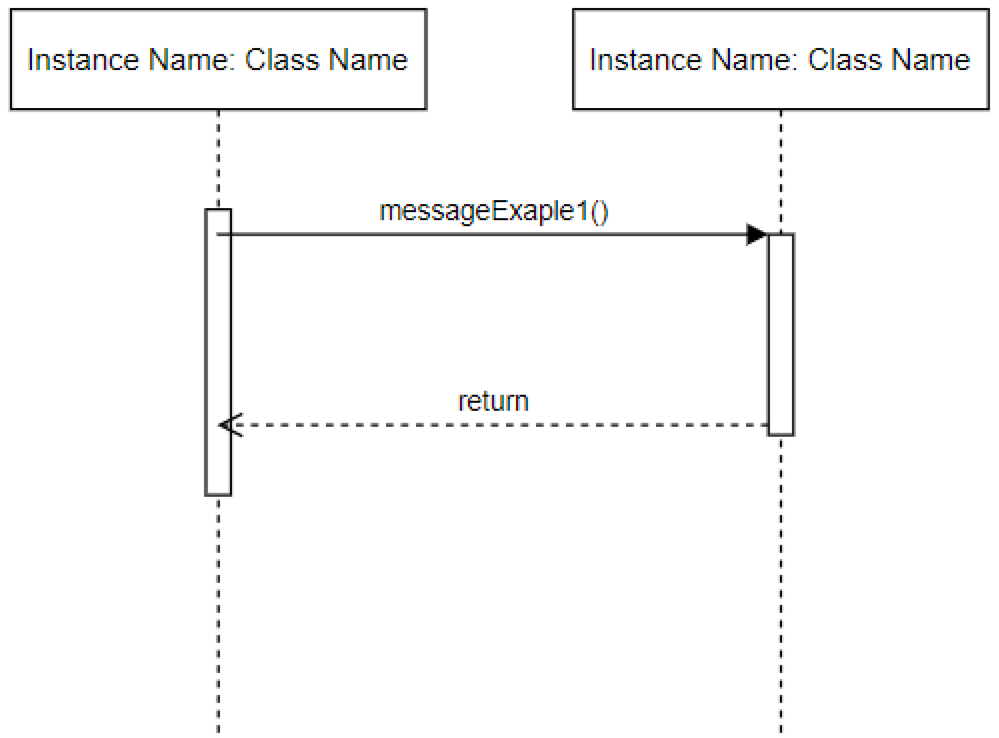
Squares with object type, optionally preceded by "name :"
–write object's name if it clarifies the diagram



Return Message

- A return message is used to indicate that the message receiver is done processing the message and is returning control over to the message caller

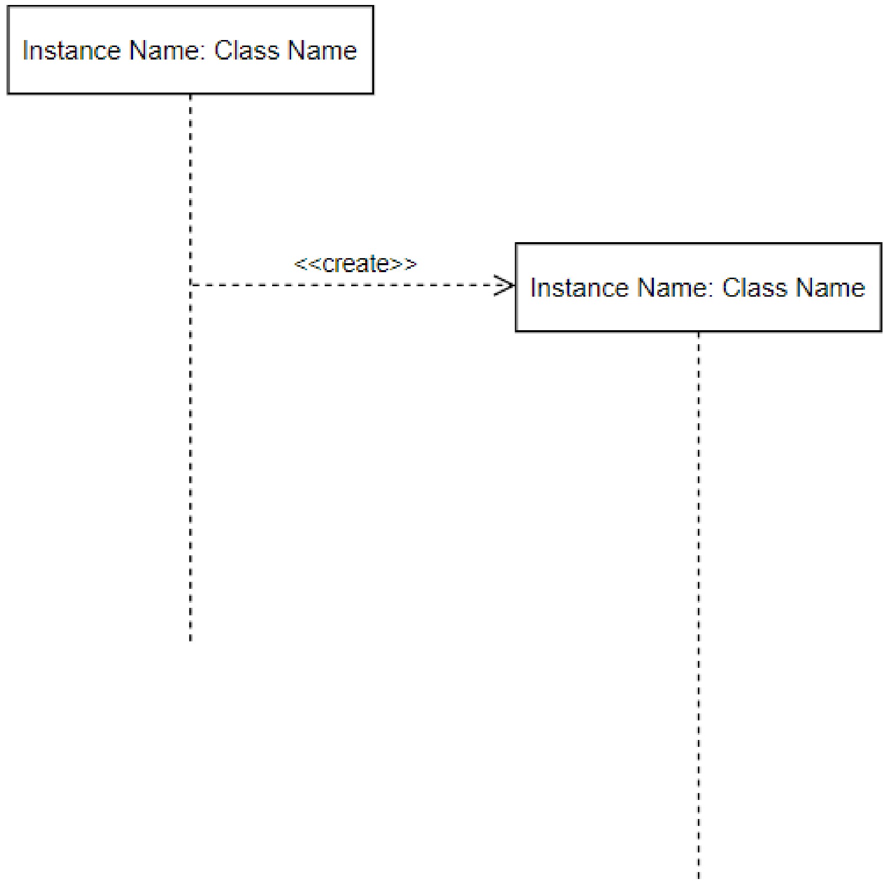
Return Message



Create Message

- To represent the new object created in the process of the execution, create a message is used

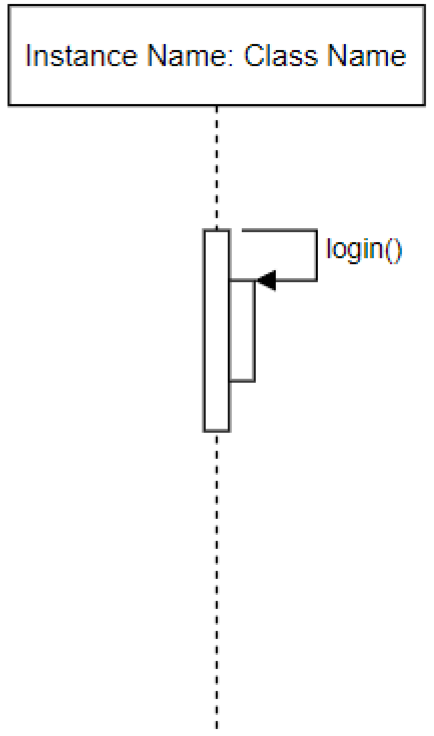
Create Message



Reflexive Message

- Sometimes an object needs to communicate with the same class methods that are when an object sends a message to itself, it is called the reflexive message.

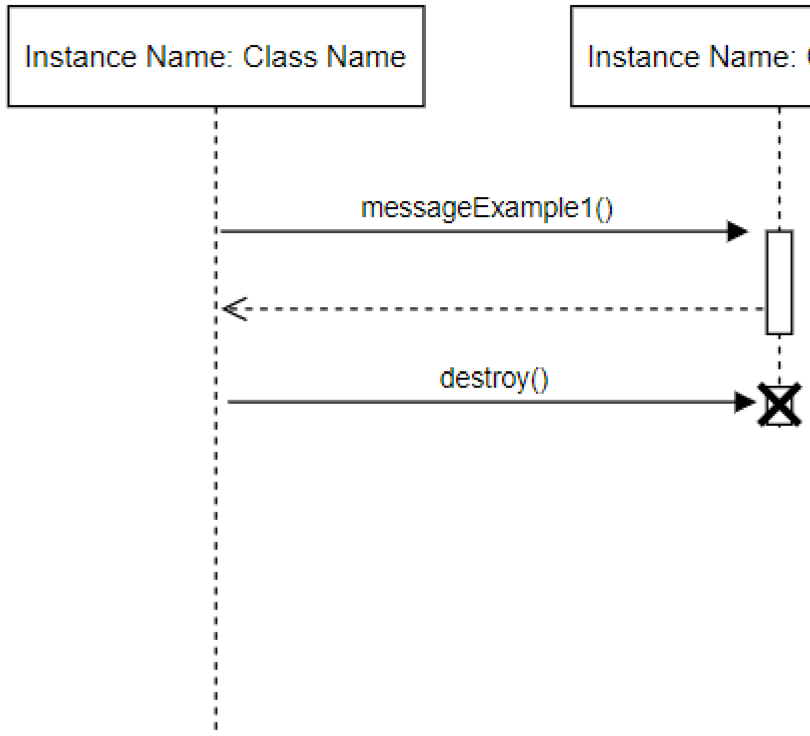
Reflexive Message



Object Destruction

- As soon as the object does not need anymore in the further process object destruction is called which is deleting the object.
- Object destruction represents the request of destroying the lifecycle of the target lifeline.

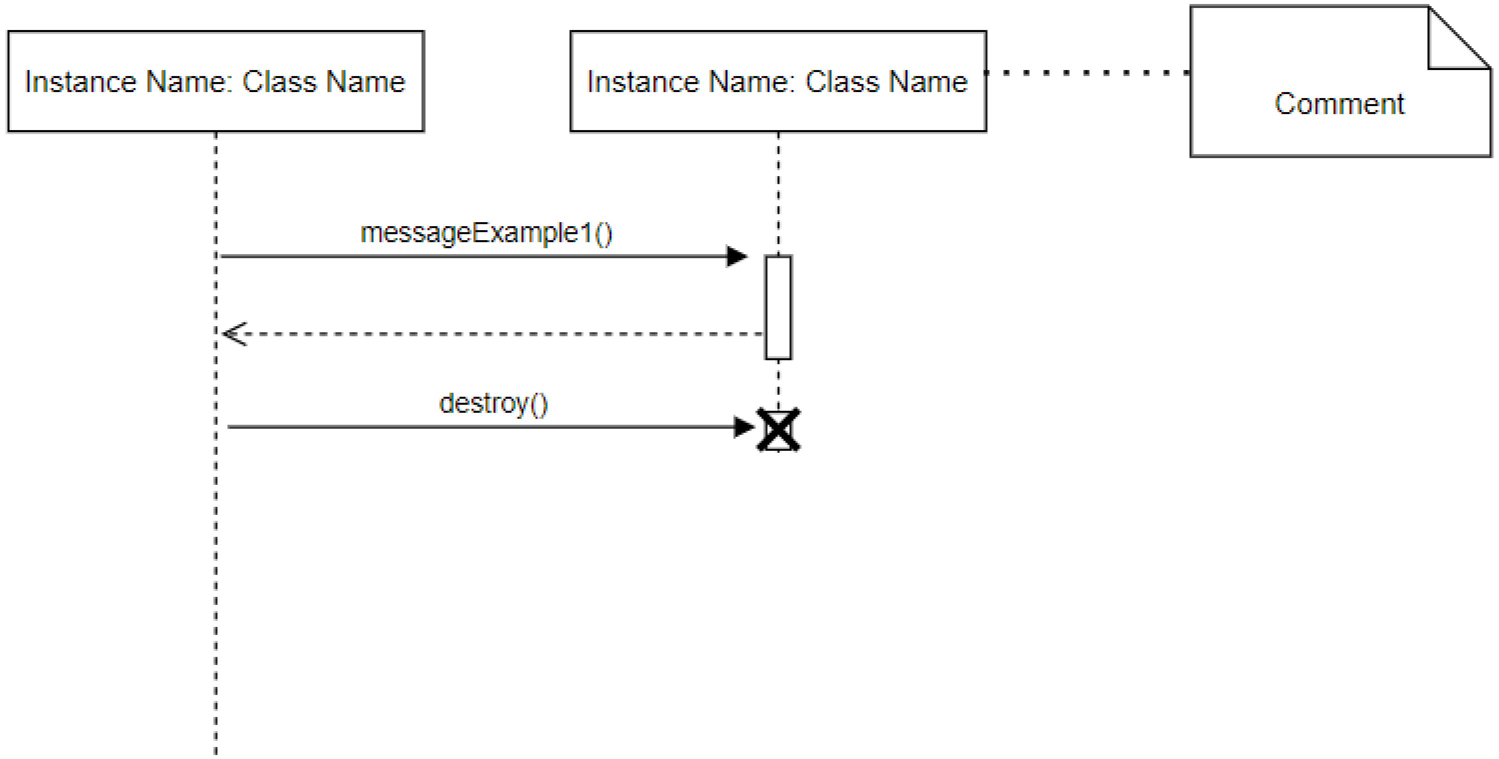
Object Destruction



Comment

- A comment or note can be used to reflect the various remarks to the elements.

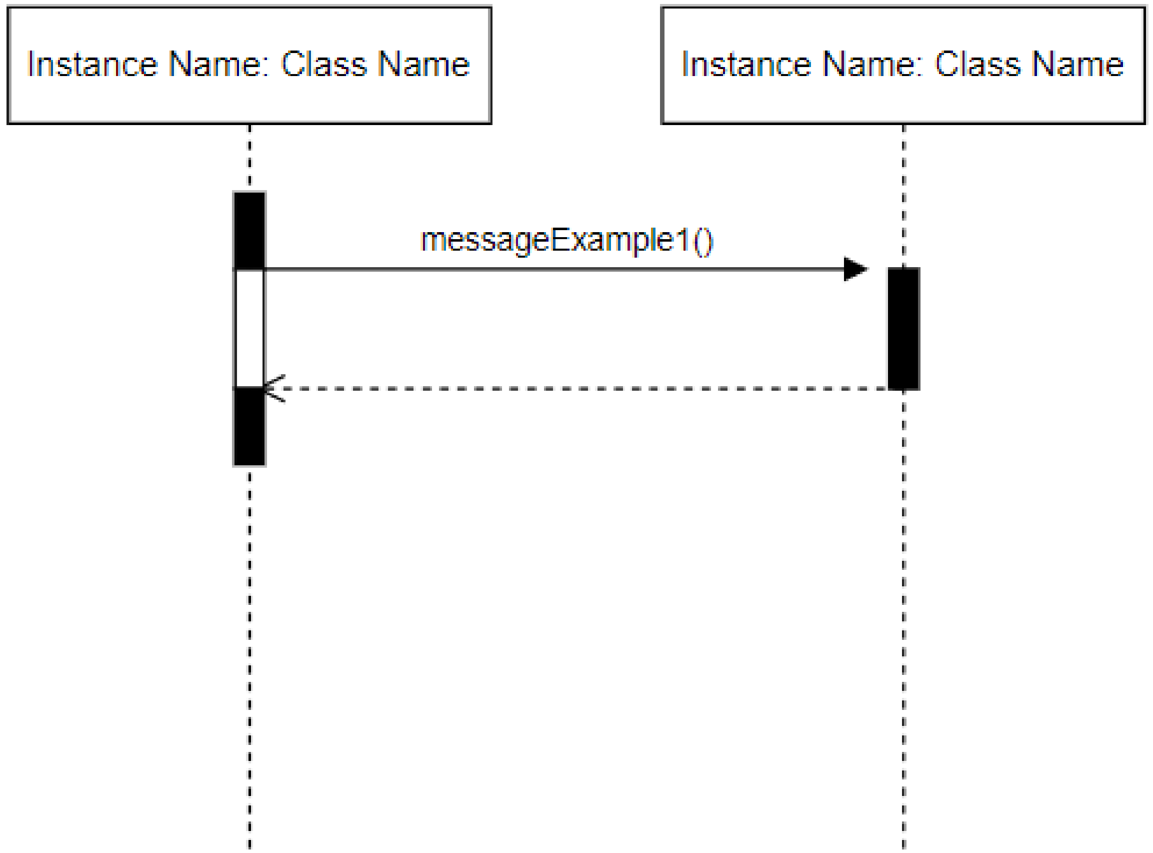
Comment


Instance Name: Class Name
Instance Name: Class Name
Comment
messageExample1()
destroy()

Focus of Control

- The focus of control indicates times during activation when processing is taking place withing that object.

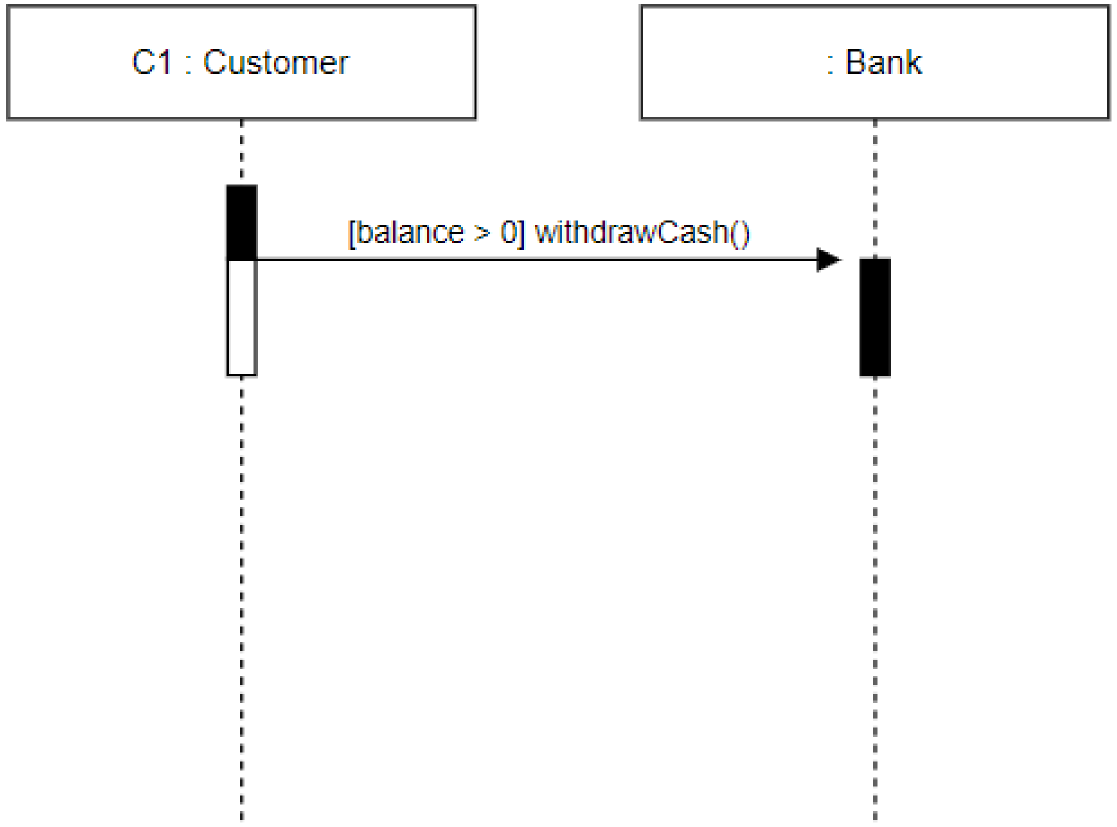
Focus of Control



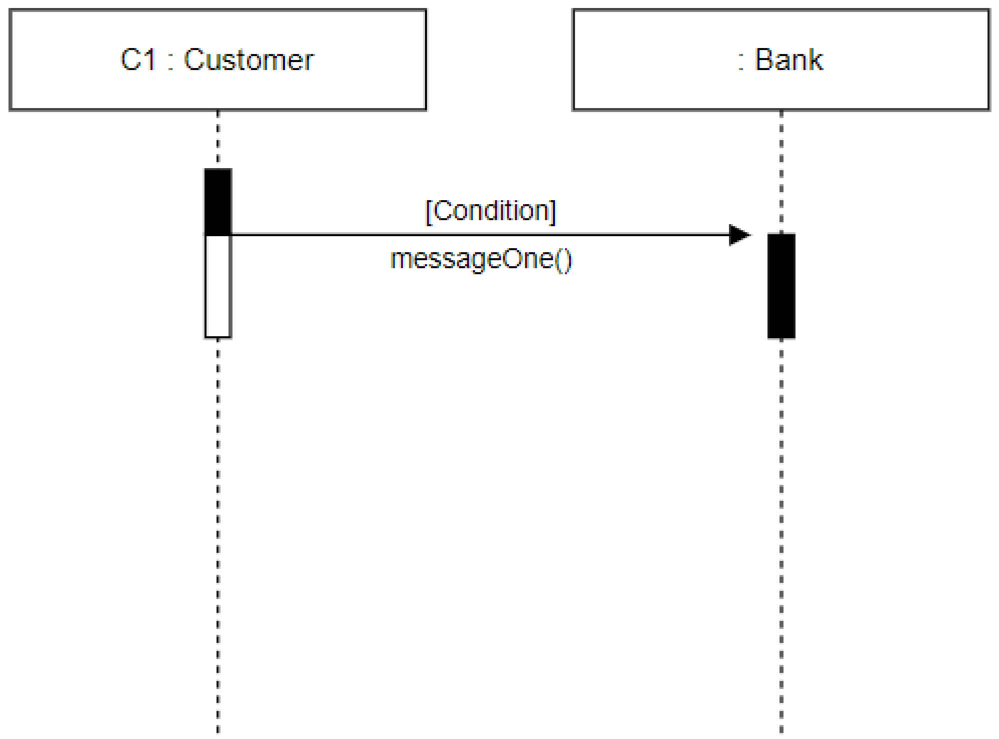
Guards

- To model conditions, like if condition in programming, we use guards in sequence diagrams.
- Guards play an important role in letting software developers know the constraints attached to the systems or a particular process.
    - In order to withdraw cash from the bank, the account balance should be greater than zero so a balance greater than zero is a condition that must be made for this message to be called.

Guards



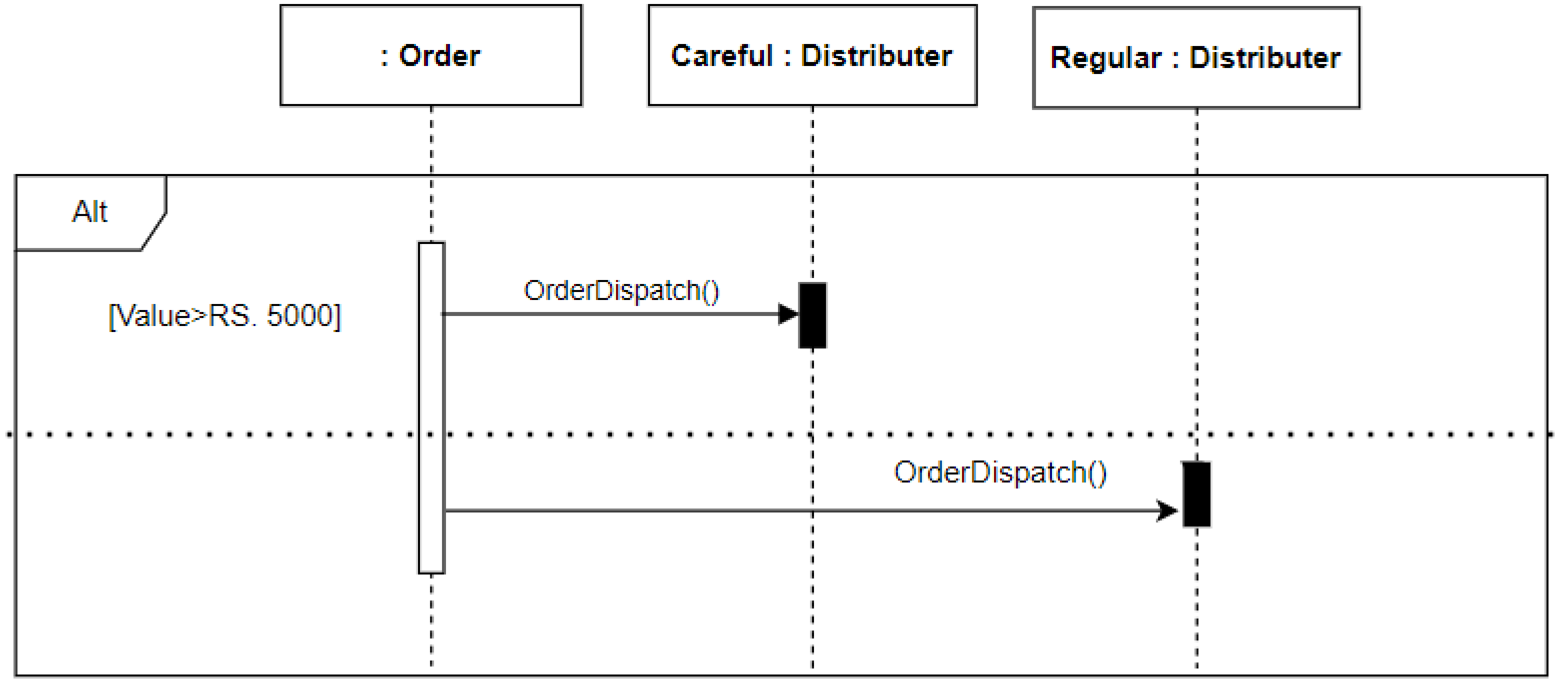
Guards



Alternatives

- Alternative in sequence diagram symbolizes a choice between two or more message sequences.

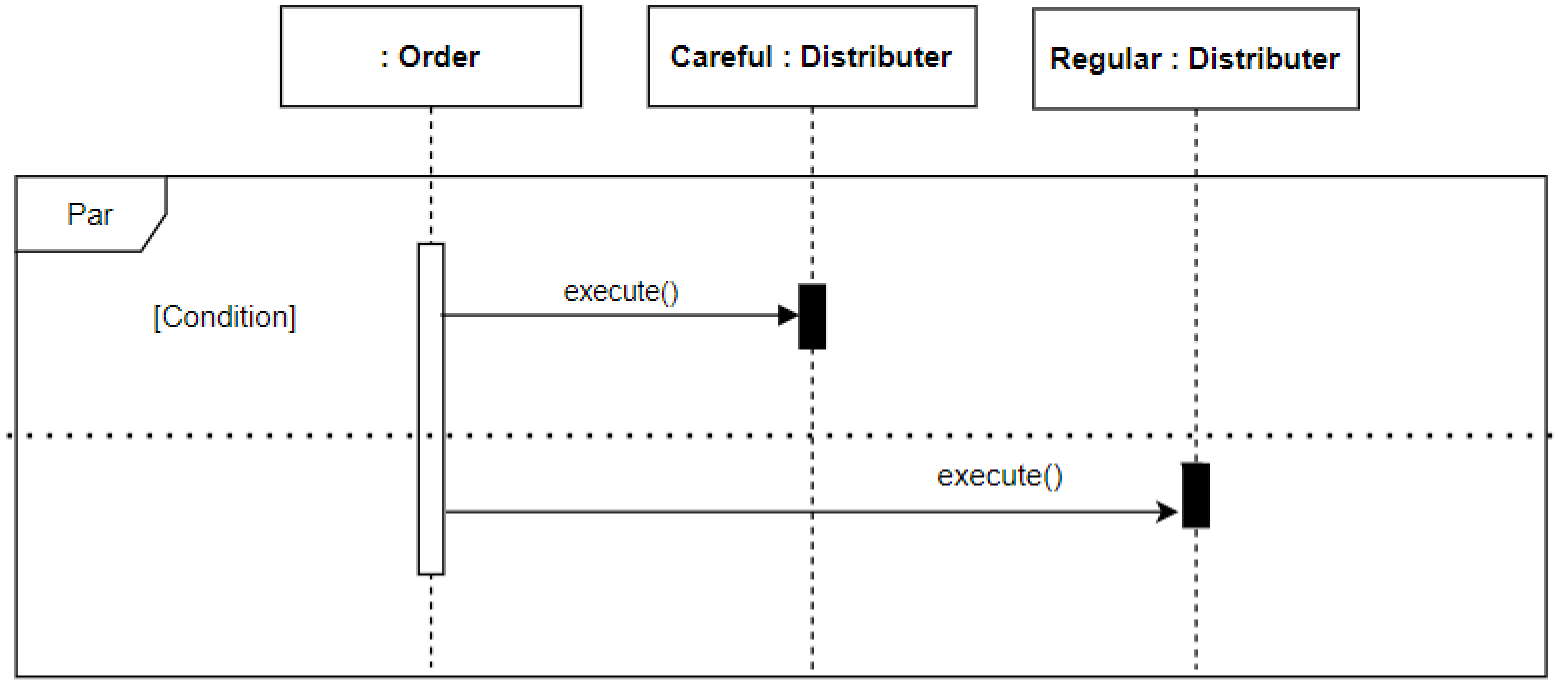
Alternatives



Parallel

- If the condition is match, each fragment or the message will execute parallel.

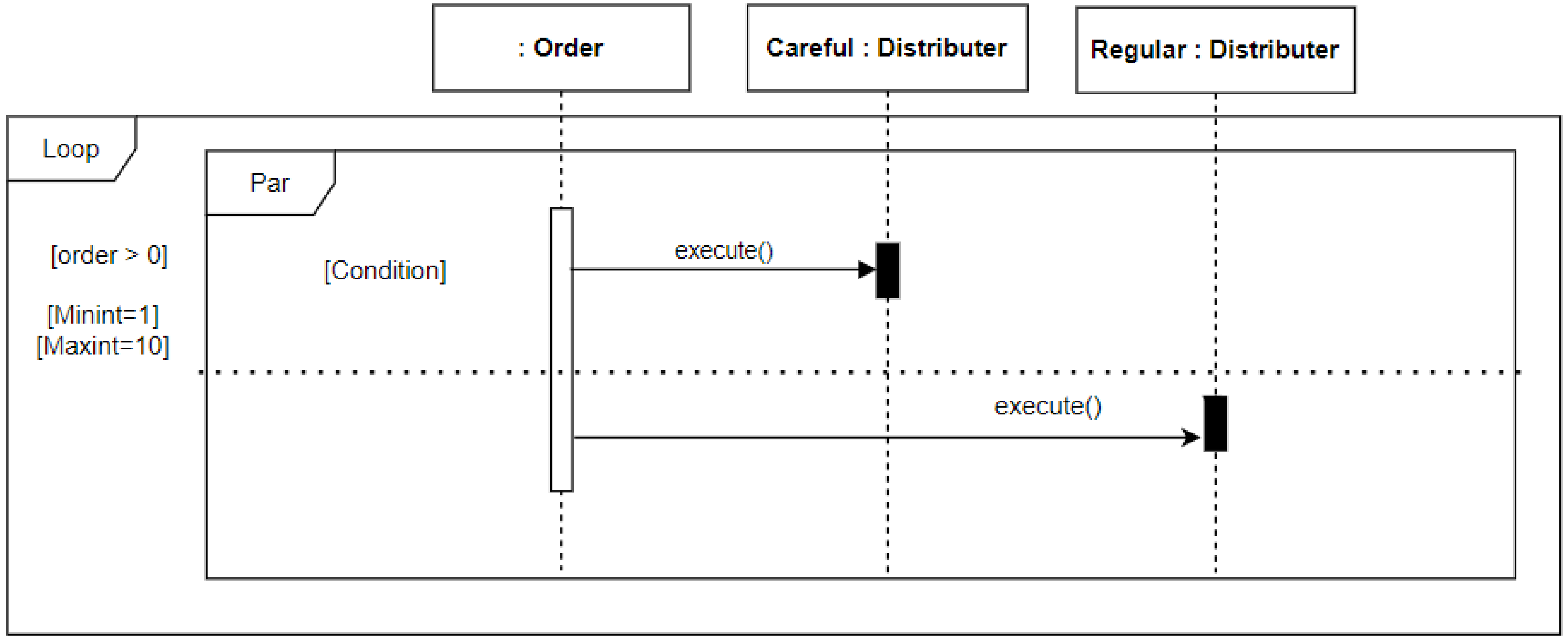
Parallel


: Order
Careful : Distributer
Regular : Distributer
Par
[Condition]
execute()
execute()

Loops

- Loops fragment is used to represent a repetitive sequence.

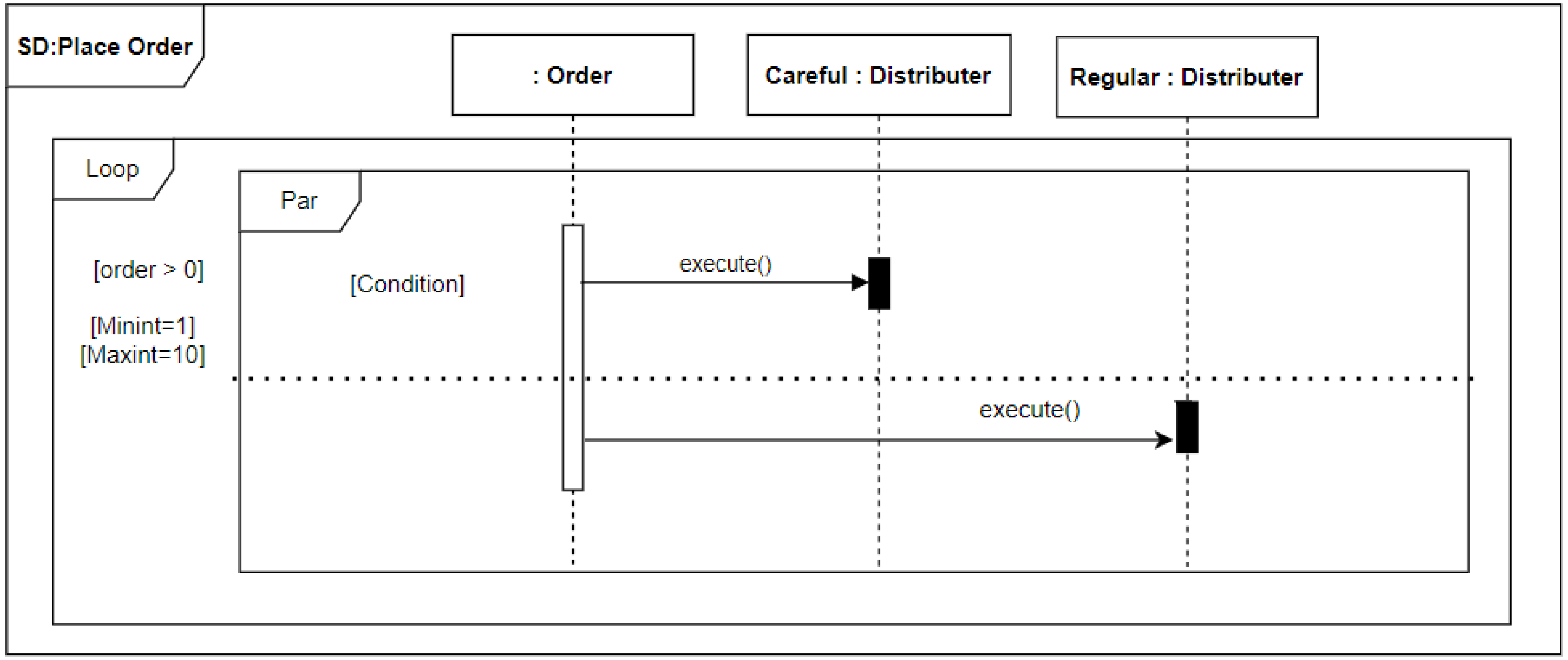
Loops

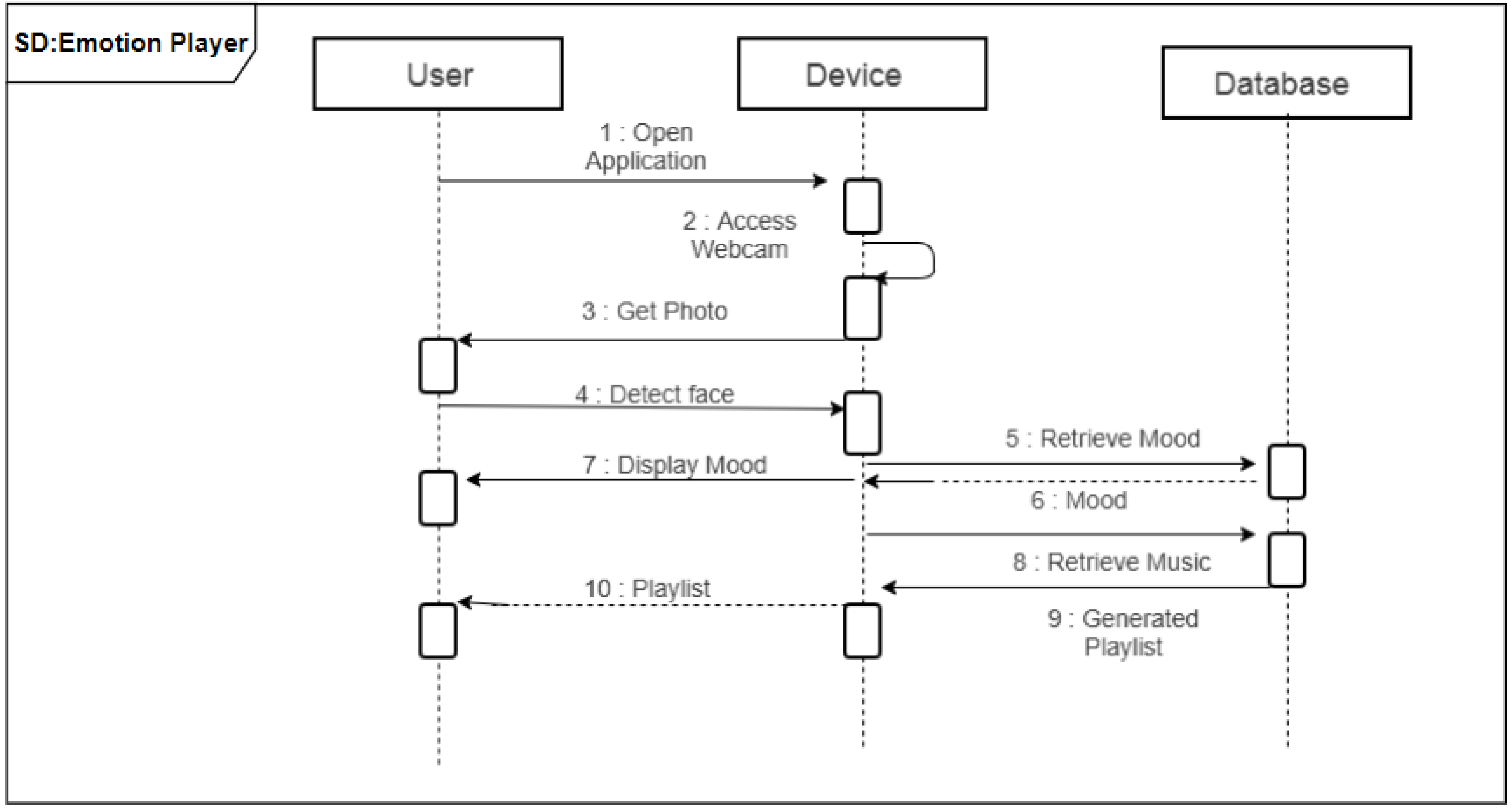

: Order
Careful : Distributer
Regular : Distributer
Loop
Par
[order > 0]
[Minint=1]
[Maxint=10]
[Condition]
execute()
execute()

Sequence Diagram

- The entire sequence diagram is surround by the box or frame.

Sequence Diagram




SD:Emotion Player
User
Device
Database
1 : Open Application
2 : Access Webcam
3 : Get Photo
4 : Detect face
5 : Retrieve Mood
6 : Mood
7 : Display Mood
8 : Retrieve Music
9 : Generated Playlist
10 : Playlist

Questions?

RIPHAH
INTERNATIONAL
UNIVERSITY