OOAD

Interaction Diagram

Interaction Diagram

- From the name *Interaction* it is clear that the diagram is used to describe some type of interactions among the different elements in the model.
- Interaction diagrams are models that describe how a group of objects interact / collaborate in some behavior - typically a single use-case.

Interaction Diagram

- **Purpose**
  - To capture dynamic behaviour of a system.
  - To describe the message flow in the system.
  - To describe structural organization of the objects.
  - To describe interaction among objects.

Forms of Interaction Diagram

- This interactive behaviour is represented in UML by two diagrams known as
  - ***Sequence diagram*** **and**
  - ***Collaboration diagram*****.**

Drawing the interaction diagram

- The purpose of interaction diagrams are to capture the dynamic aspect of a system.
- Dynamic aspect can be defined as the snap shot of the running system at a particular moment.
- So the following things are to identified clearly before drawing the interaction diagram:
  - Objects taking part in the interaction.
  - Message flows among the objects.
  - The sequence in which the messages are flowing.
  - Object organization.

Sequence Diagram

- Describe the flow of messages, events, actions between objects .
- An important characteristic of a sequence diagram is that time passes from top to bottom and model important runtime interactions between the parts that make up the system.
- Typically used to document and understand the logical flow of the system .

Sequence Diagram Key Parts

- **participant: object or entity that acts in the diagram**

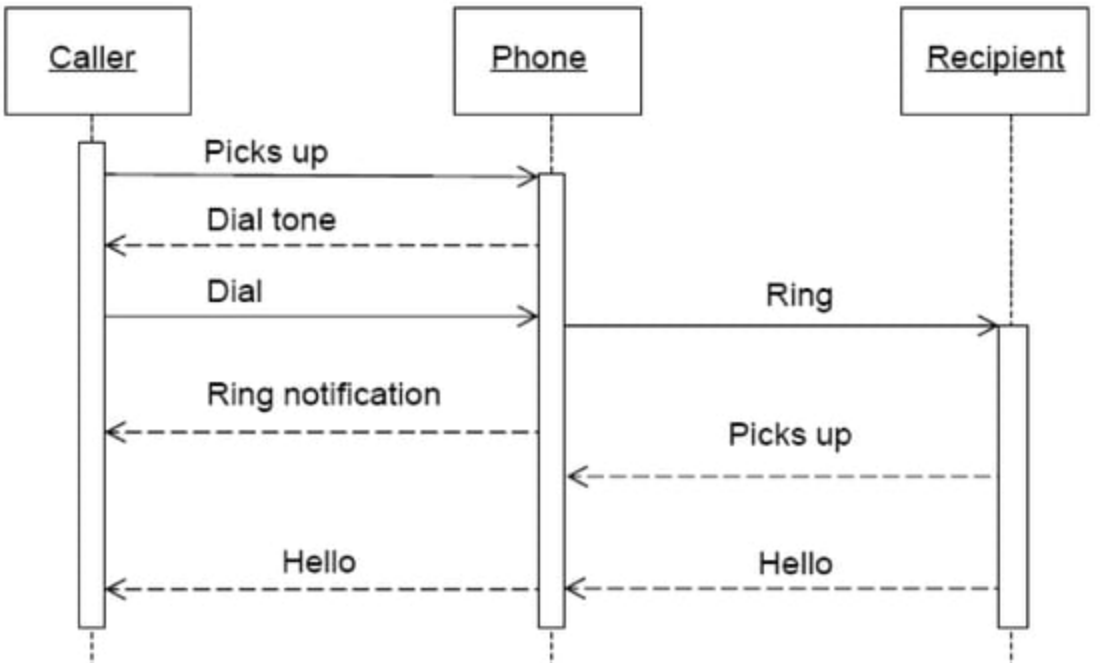
  message: communication between participant objects

- the **axes in a sequence diagram:**
- –**horizontal:** which object/participant is acting
- –**vertical:** time (down -> forward in time)

Sequence Diagrams 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.

Sequence Diagram (make a phone call)

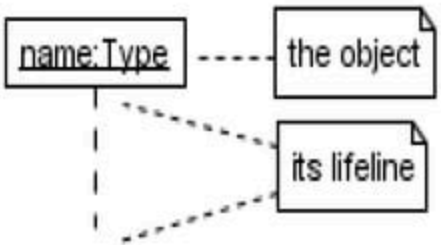


Object

- Objects as well as classes can be **targets** on a sequence diagram, which means that messages can be sent to them.
- A **target** is displayed as a rectangle with some text in it. Below the target, its lifeline extends for as long as the target exists. The lifeline is displayed as a vertical dashed line.

Object

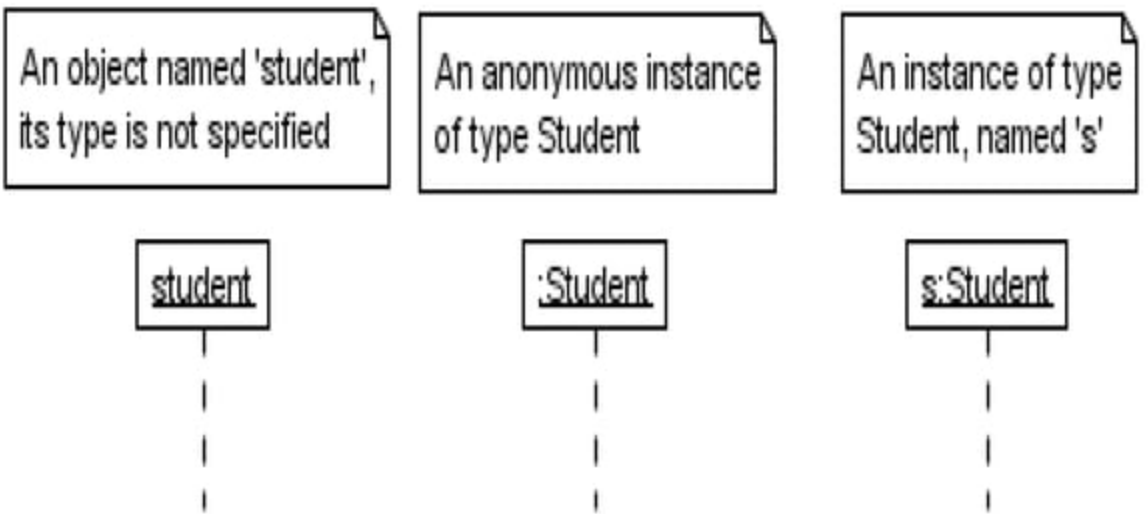
- The basic notation for an object is



where 'name' is the name of the object in the context of the diagram and 'Type' indicates the type of which the object is an instance.

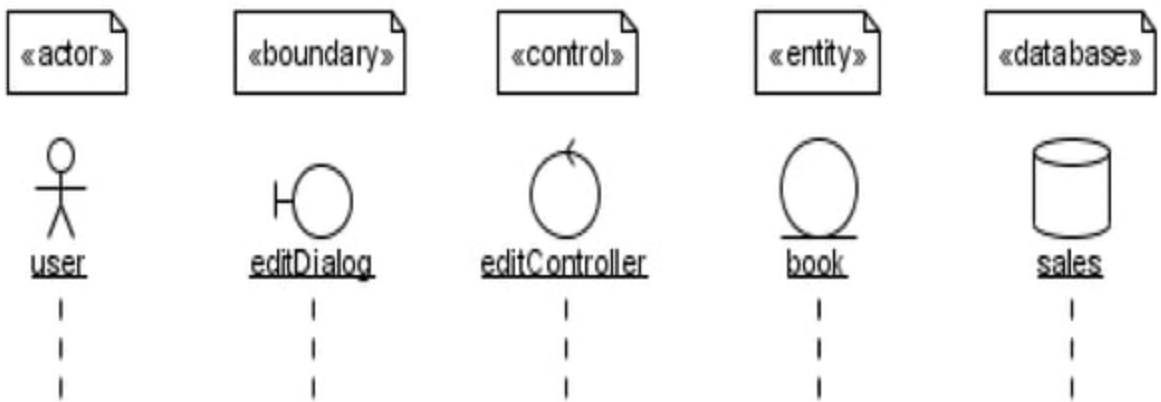
- Both name and type are optional, but at least one of them should be present.

Object (Examples)



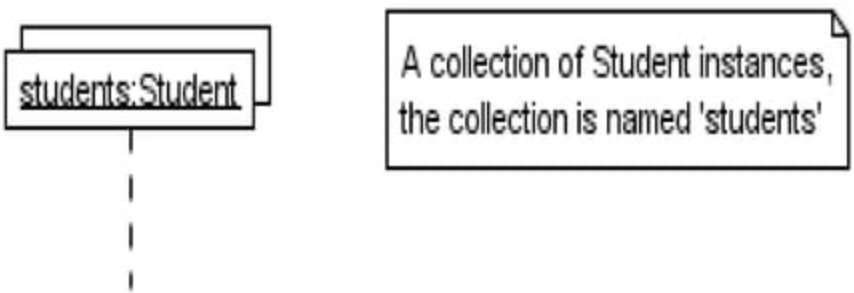
Object (Examples)

- As with any UML-element, we can add a stereotype to a target. Some often used stereotypes for objects are «actor», «boundary», «control», «entity» and «database».



MultiObject

- When we want to show how a client interacts with the elements of a collection, we can use a multiobject. Its basic notation is:

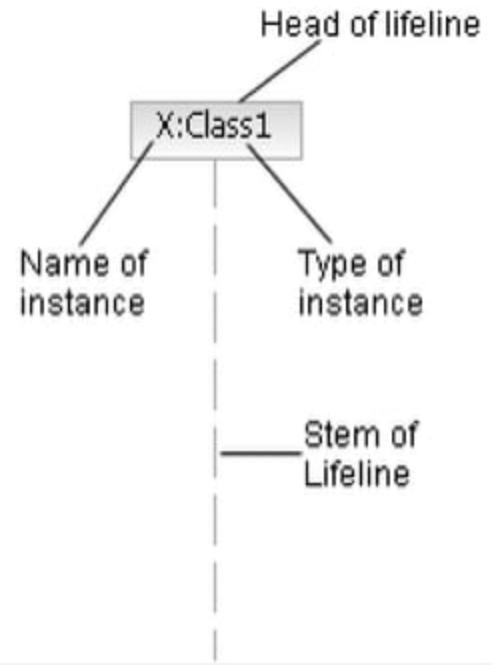


Lifelines in UML diagrams

- In UML diagrams, such as sequence or communication diagrams, lifelines represent the objects that participate in an interaction.
- **Lifeline** is a named element which represents an **individual participant** in the interaction.
- For example, in a banking scenario, lifelines can represent objects such as a bank system or customer.
- Each instance in an interaction is represented by a lifeline.

Lifelines in Sequence diagrams

- Lifeline in a sequence diagram is displayed with its name and type in a rectangle, which is called the head. The head is located on top of a vertical dashed line, called the stem, which represents the timeline for the instance of the object.

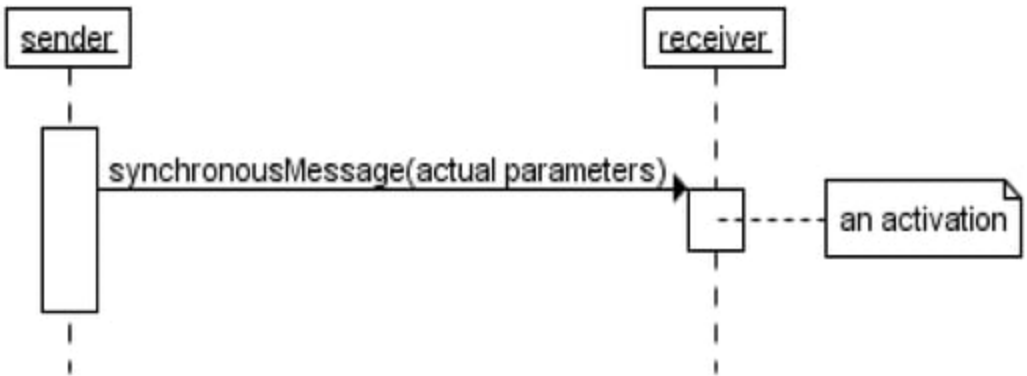


Messages

- 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.
- Near the arrow, the name and parameters of the message are shown.

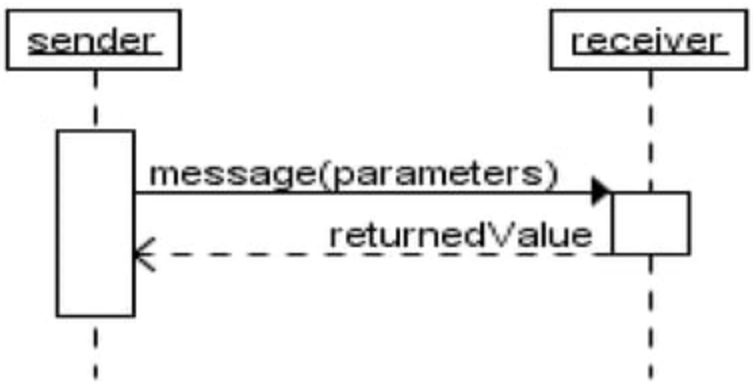
Synchronous message

- A synchronous message is used when the sender waits until the receiver has finished processing the message, only then does the caller continue.
- A closed and filled arrowhead signifies that the message is sent synchronously.



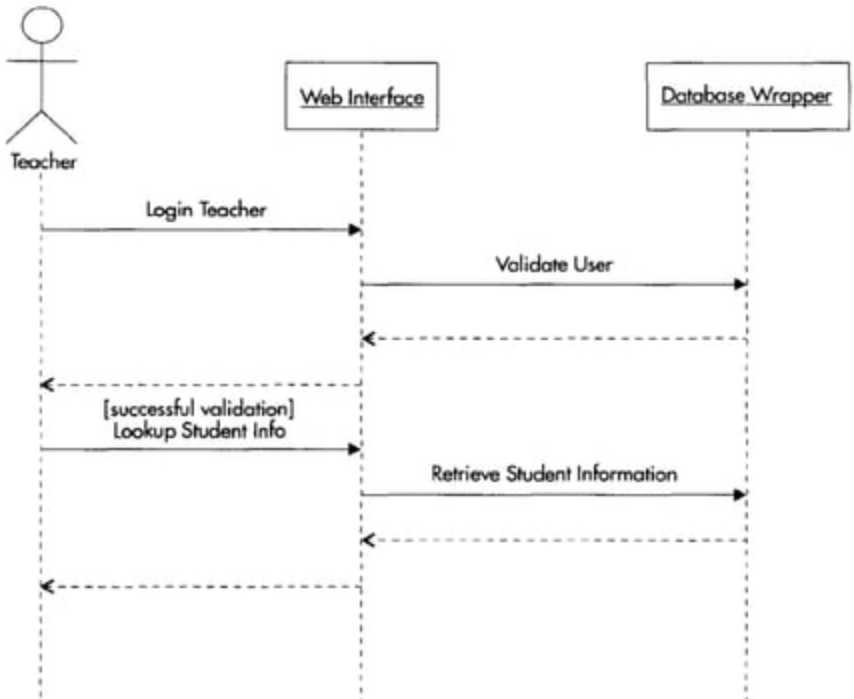
Synchronous message

- If you want to show that the receiver has finished processing the message and returns control to the sender, draw a dashed arrow from receiver to sender. Optionally, a value that the receiver returns to the sender can be placed near the return arrow.



- If you want your diagrams to be easy to read, only show the return arrow if a value is returned. Otherwise, hide it.

Synchronous message

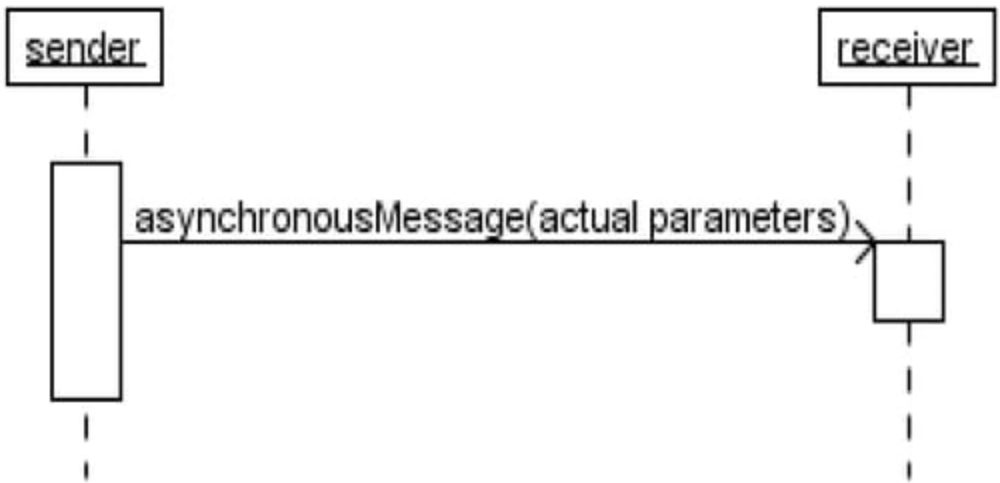


Asynchronous messages

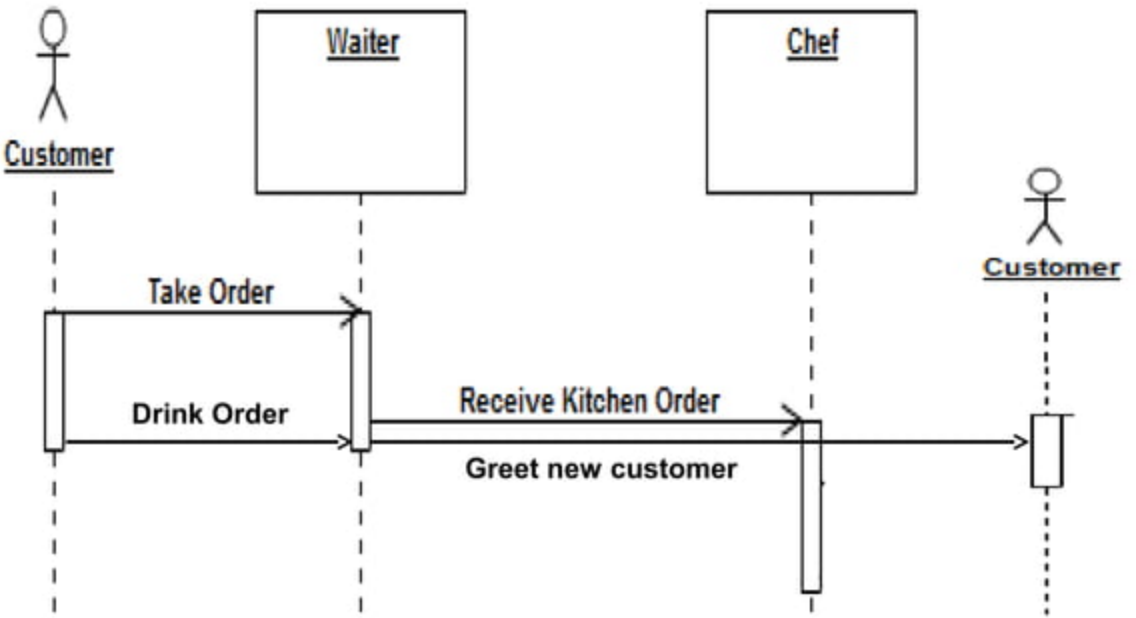
- With an asynchronous message, the sender does not wait for the receiver to finish processing the message, it continues immediately.
- Messages sent to a receiver in another process or calls that start a new thread are examples of asynchronous messages.

Asynchronous messages

- An open arrowhead is used to indicate that a message is sent asynchronously.

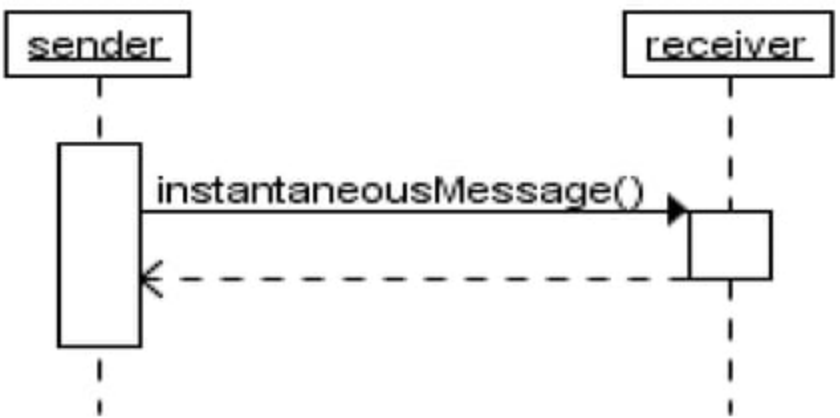


Asynchronous messages



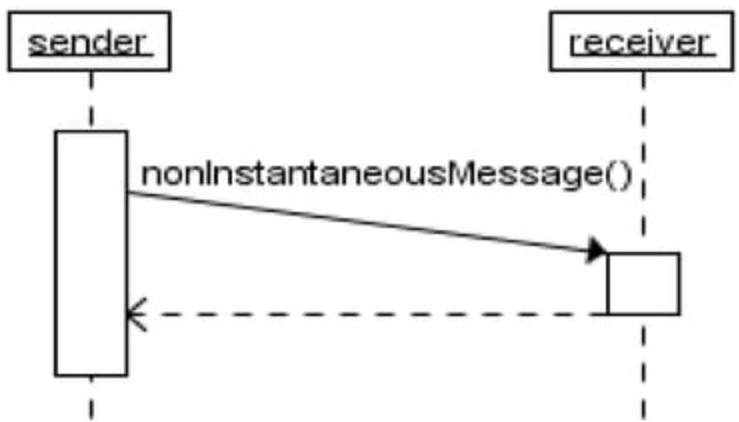
Instantaneous message

- Messages are often considered to be instantaneous, i.e. the time it takes to arrive at the receiver is negligible.
- Such messages are drawn as a horizontal arrow.

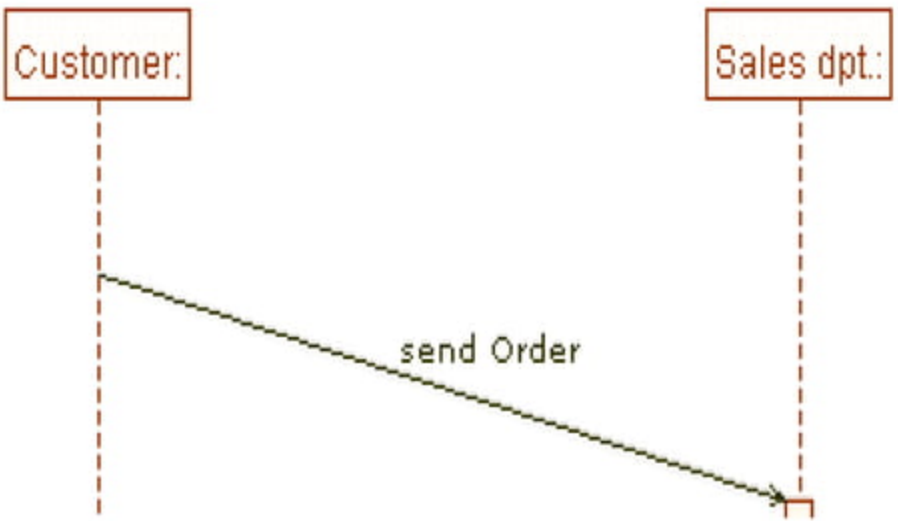


Non-instantaneous message

- Sometimes the message takes a considerable amount of time to reach the receiver.
- For example, a message across a network.
- Such a non-instantaneous message is drawn as a slanted arrow.



Non-instantaneous message

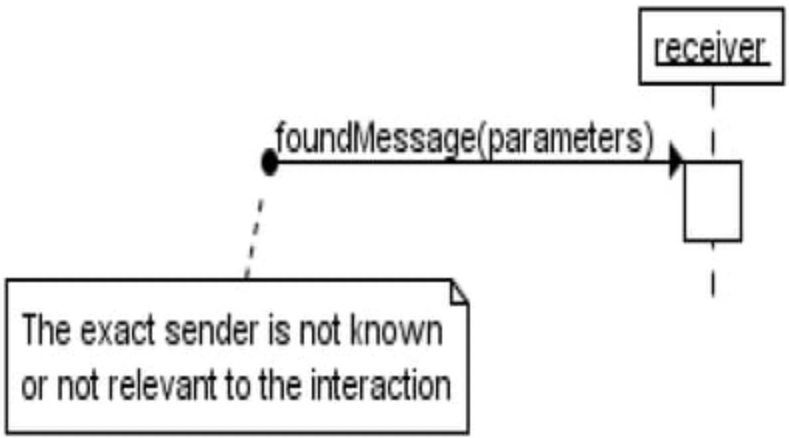


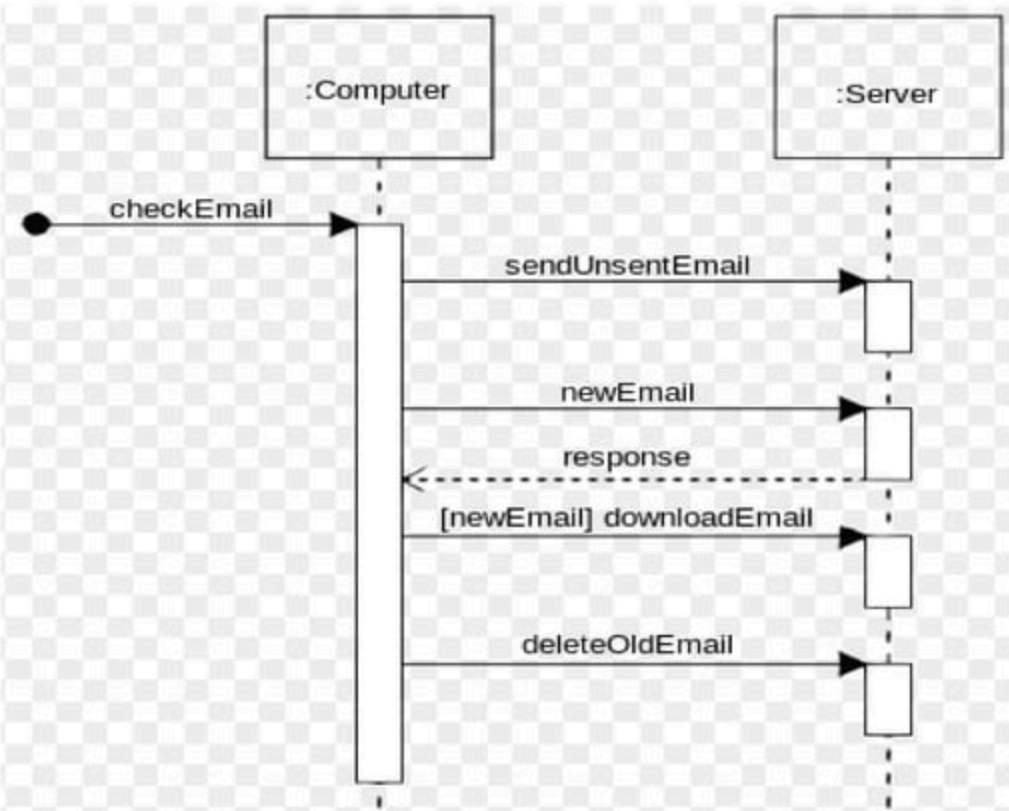
Found message

- A found message is a message of which the caller is not shown.
- Depending on the context, this could mean that either the sender is not known, or that it is not important who the sender was.

Found message

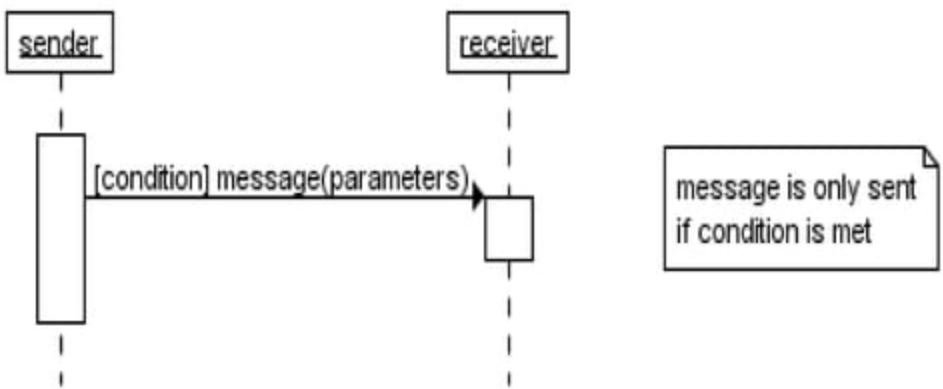
- The arrow of a found message originates from a filled circle.



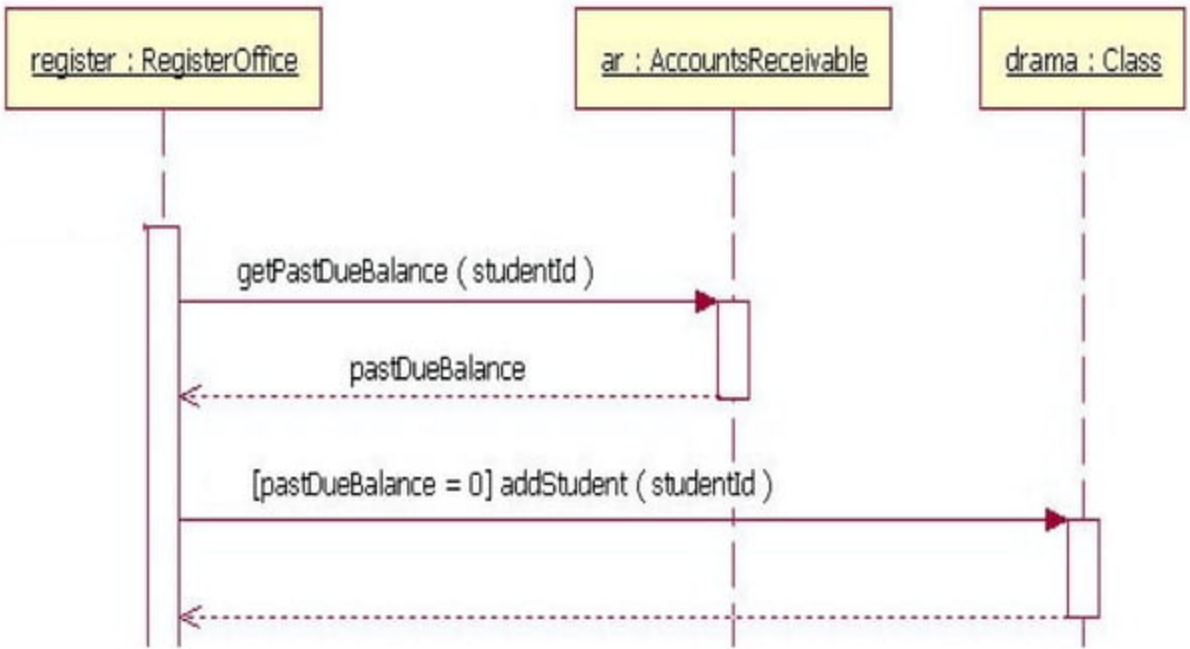

:Computer
:Server
checkEmail
sendUnsentEmail
newEmail
response
[newEmail] downloadEmail
deleteOldEmail

Guards

- A message can include a guard, which signifies that the message is only sent if a certain condition is met.
- The guard is simply that condition between brackets.
- Guards are used throughout UML diagrams to control flow.

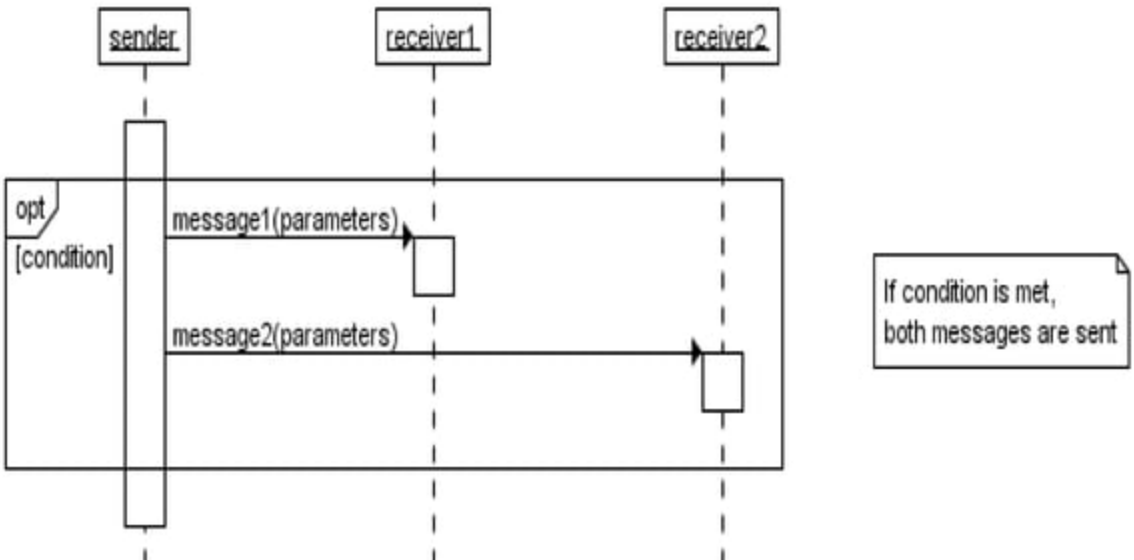


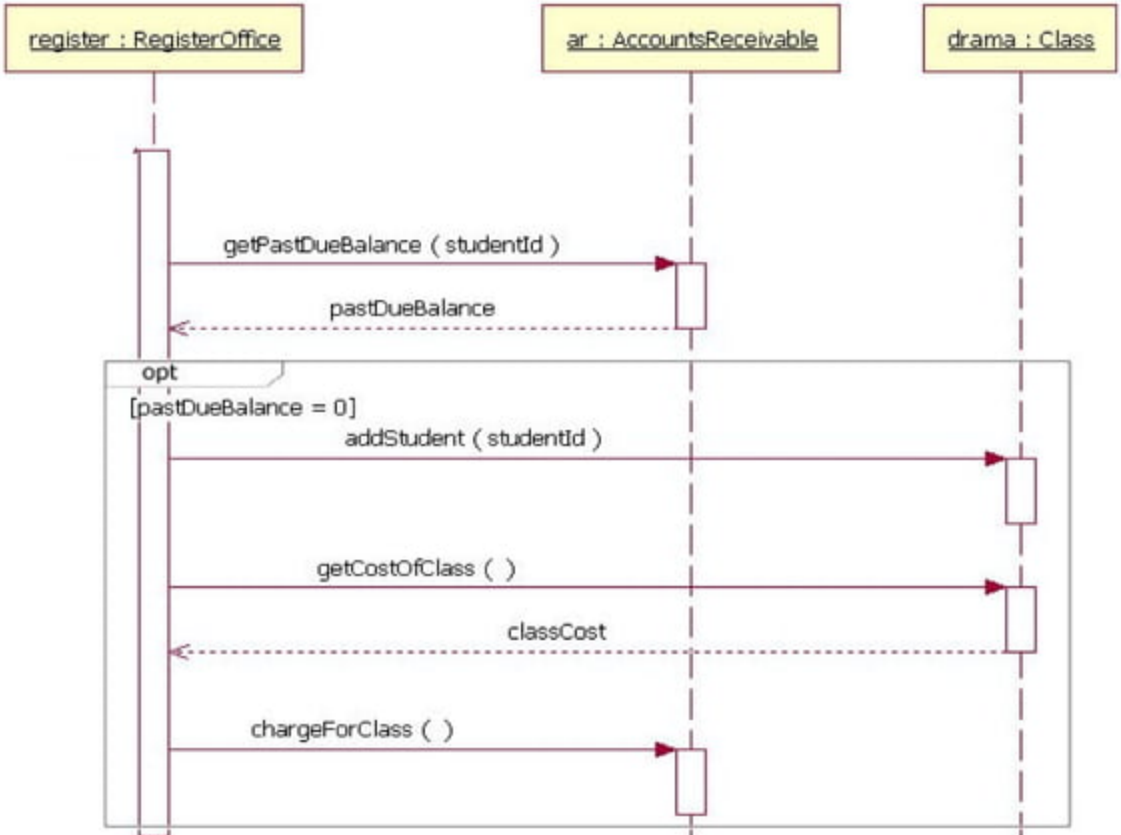
Guards


register : RegisterOffice
ar : AccountsReceivable
drama : Class
getPastDueBalance (studentId)
pastDueBalance
[pastDueBalance = 0] addStudent (studentId)

Combined fragments (alternatives, options, and loops)

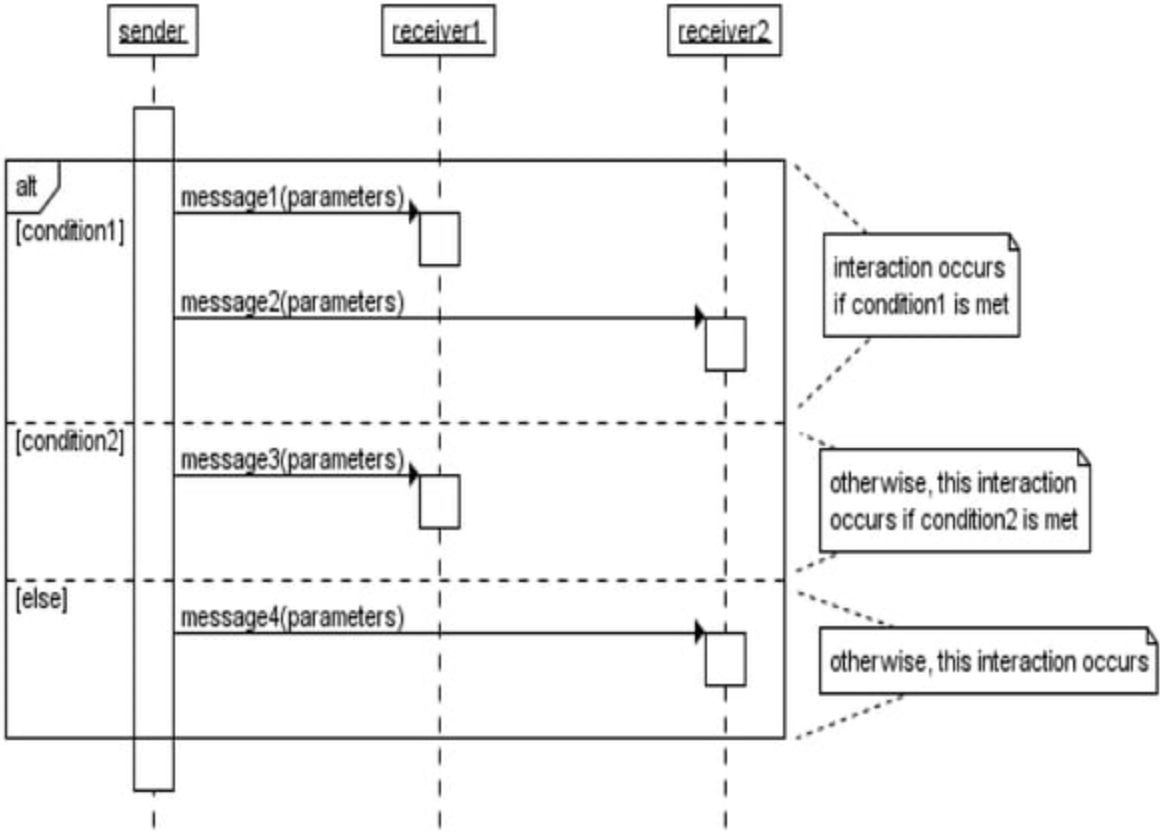
- If you want to show that several messages are conditionally sent under the same guard, you'll have to use an '**opt**' combined fragment.
- The combined fragment is shown as a large rectangle with an 'opt' operator plus a guard, and contains all the conditional messages under that guard.

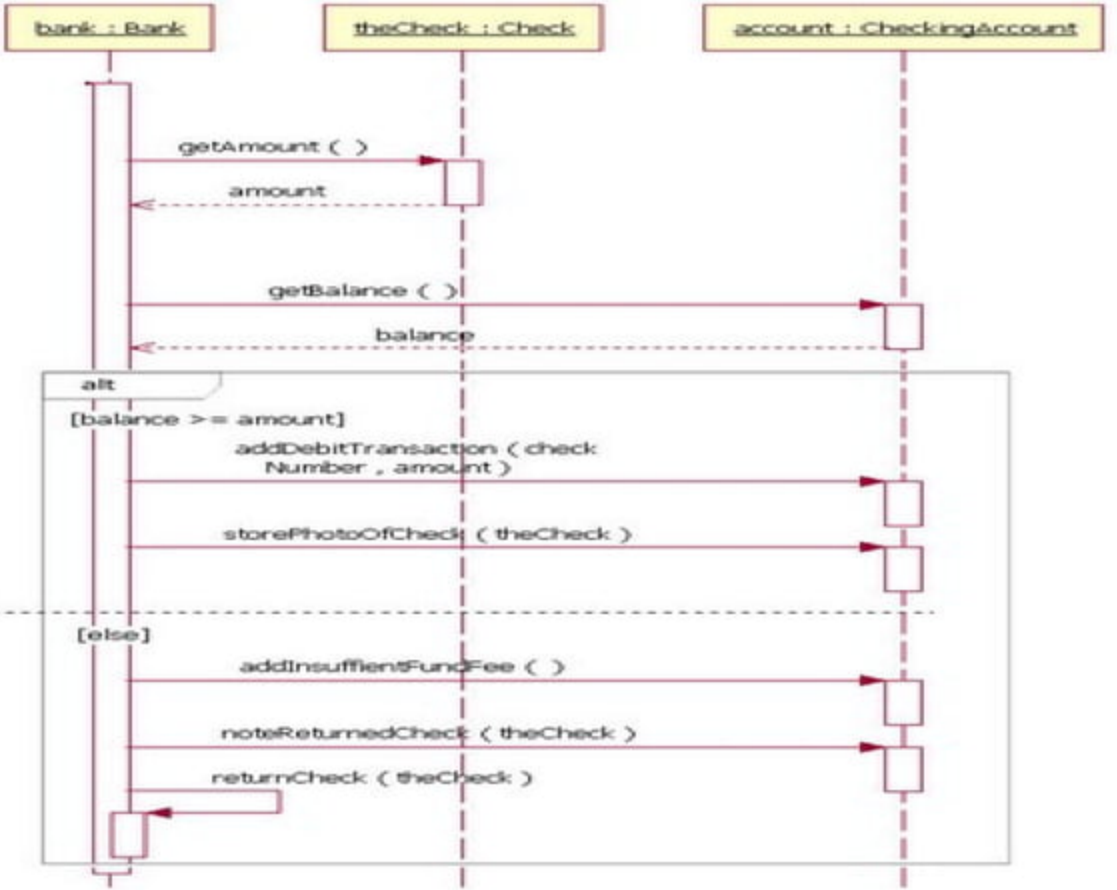

sender
receiver1
receiver2
opt
[condition]
message1(parameters)
message2(parameters)
If condition is met,
both messages are sent

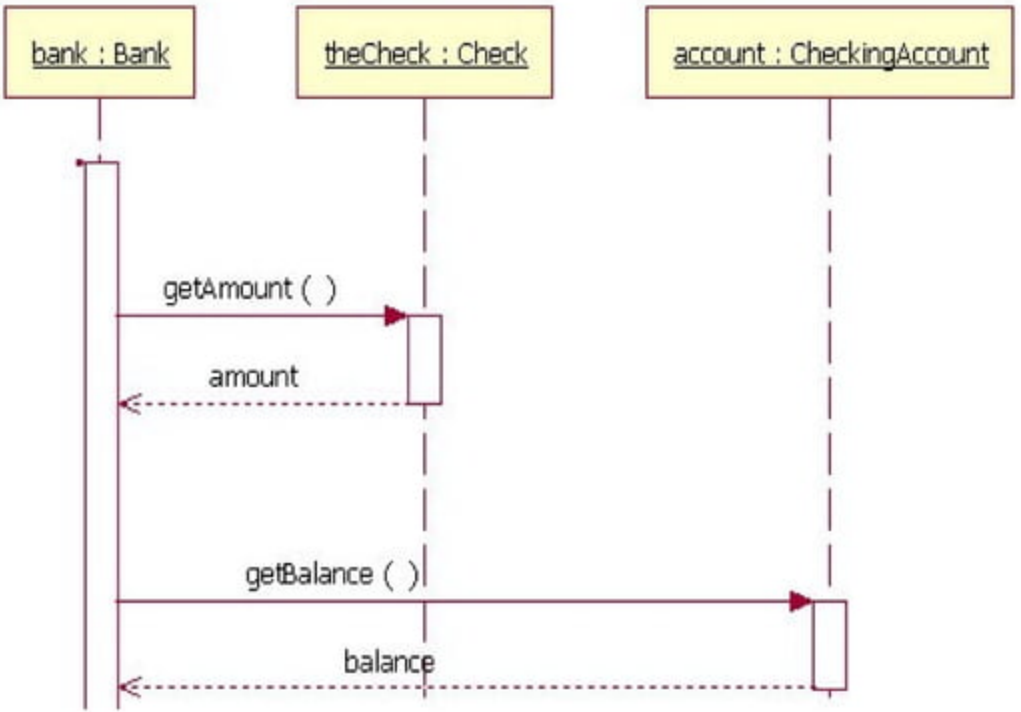

register : RegisterOffice
ar : AccountsReceivable
drama : Class
getPastDueBalance (studentId)
pastDueBalance
opt
[pastDueBalance = 0]
addStudent (studentId)
getCostOfClass ( )
classCost
chargeForClass ( )

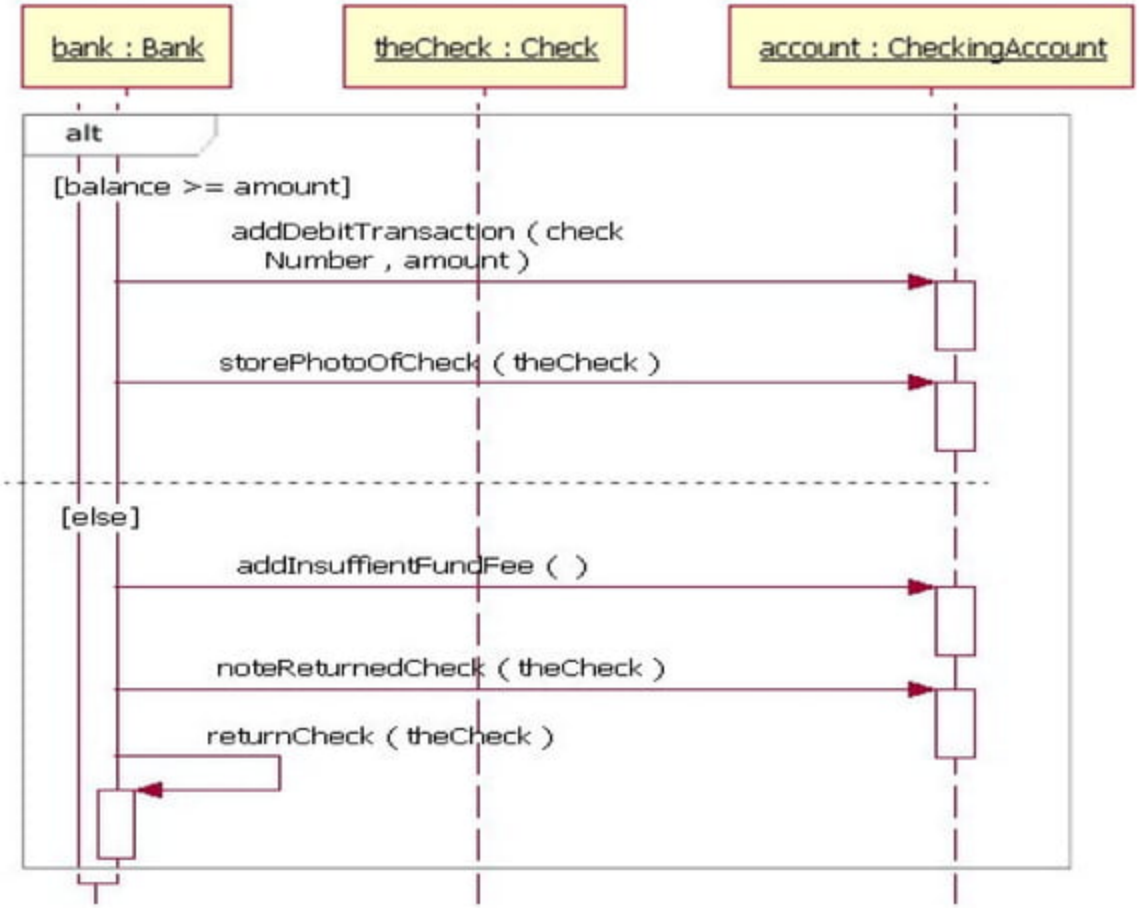
alt

- If you want to show several alternative interactions, use an '**alt**' combined fragment.
- The combined fragment contains an operand for each alternative. Each alternative has a guard and contains the interaction that occurs when the condition for that guard is met.


sender
receiver1
receiver2
alt
[condition1]
message1(parameters)
message2(parameters)
interaction occurs if condition1 is met
[condition2]
message3(parameters)
otherwise, this interaction occurs if condition2 is met
[else]
message4(parameters)
otherwise, this interaction occurs

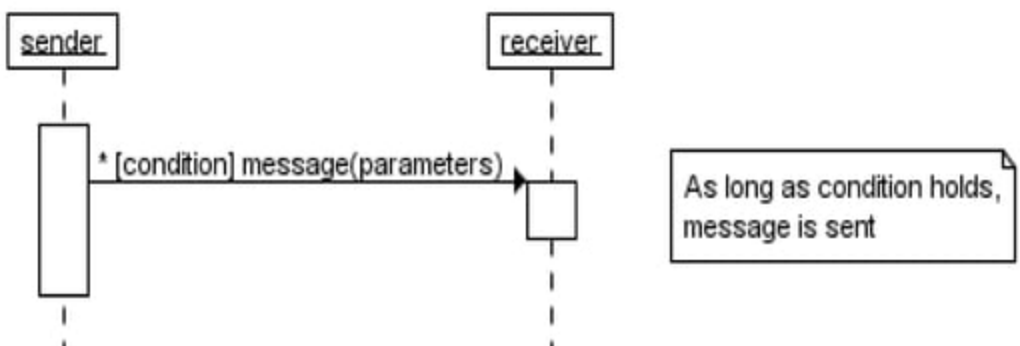

bank : Bank
theCheck : Check
account : CheckingAccount
getAmount ( )
amount
getBalance ( )
balance
alt
[balance >= amount]
addDebitTransaction (check Number , amount)
storePhotoOfCheck (theCheck)
[else]
addInsuffientFundFee ( )
noteReturnedCheck (theCheck)
returnCheck (theCheck)

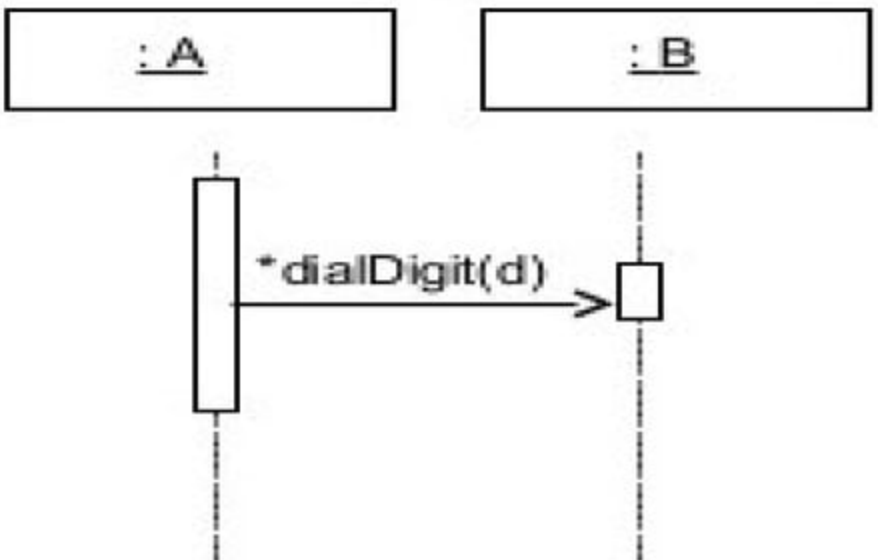

bank : Bank
theCheck : Check
account : CheckingAccount
getAmount ( )
amount
getBalance ( )
balance


bank : Bank
theCheck : Check
account : CheckingAccount
alt
[balance >= amount]
addDebitTransaction (check Number , amount)
storePhotoOfCheck (theCheck)
[else]
addInsuffientFundFee ( )
noteReturnedCheck (theCheck)
returnCheck (theCheck)

Repeated interaction

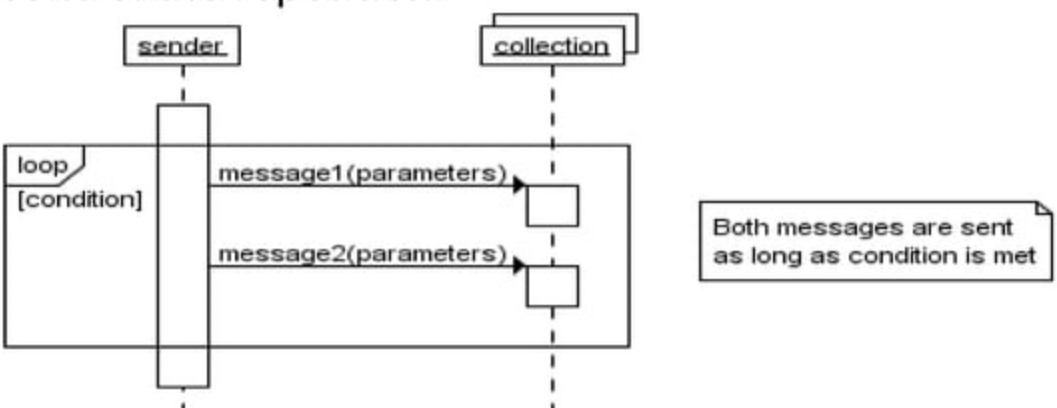
- When a message is prefixed with an asterisk (the '*'-symbol), it means that the message is sent repeatedly. A guard indicates the condition that determines whether or not the message should be sent (again). As long as the condition holds, the message is repeated.

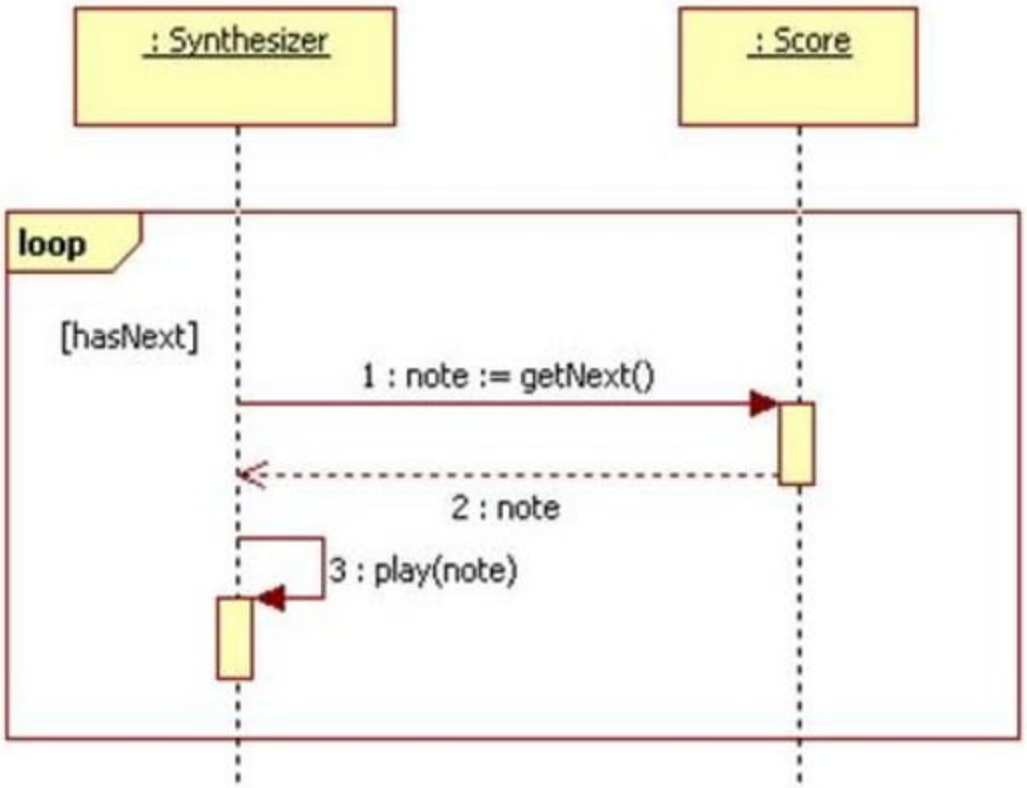


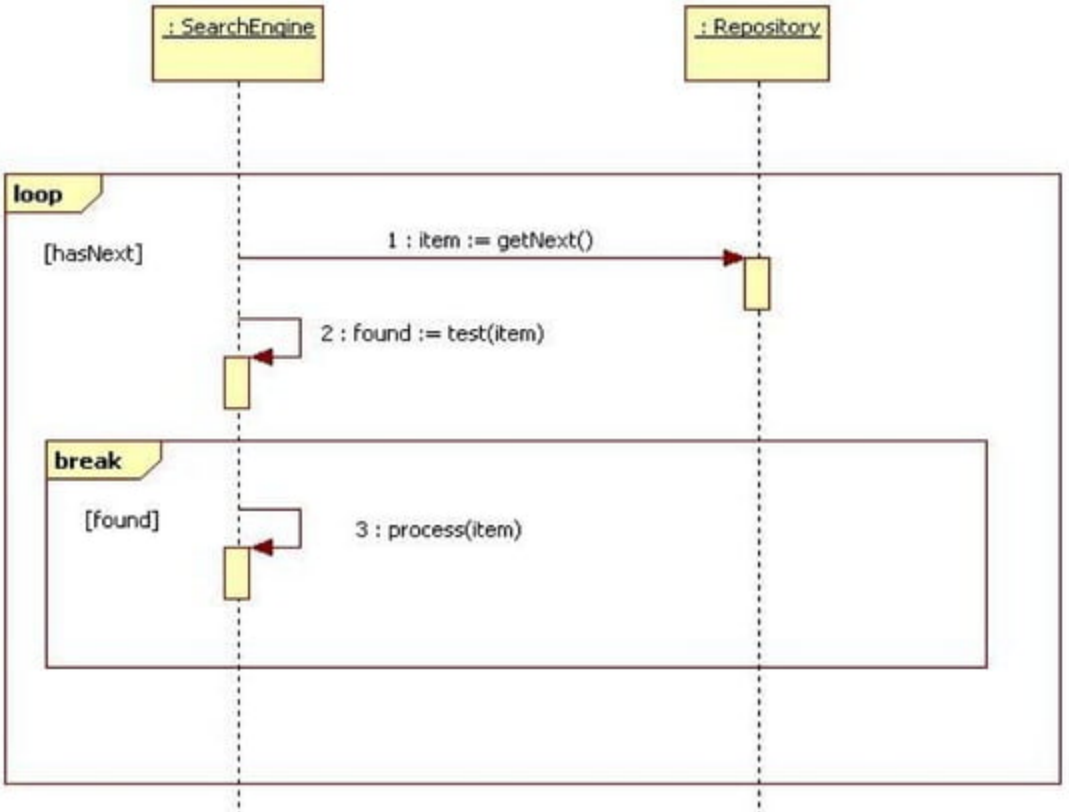

: A
: B
*dialDigit(d)

Repeated interaction

- If you want to show that multiple messages are sent in the same iteration, a '**loop**' combined fragment can be used. The operator of the combined fragment is 'loop' and the guard represents the condition to control the repetition.

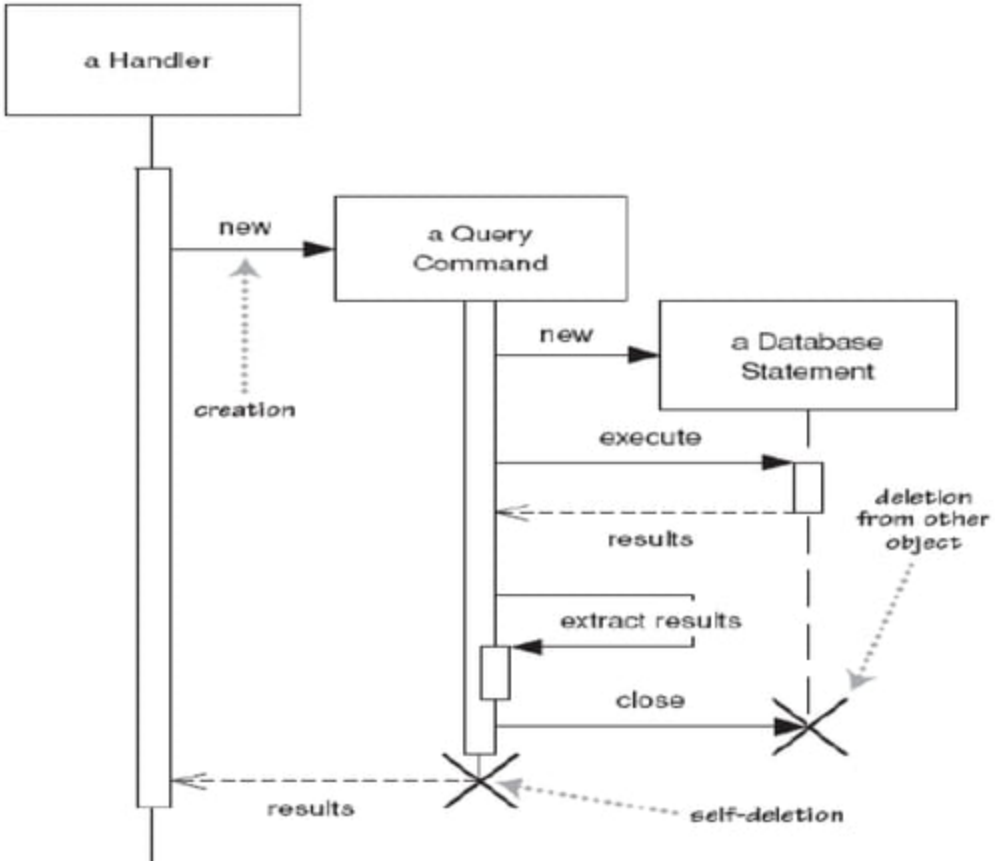



: Synthesizer
: Score
loop
[hasNext]
1 : note := getNext()
2 : note
3 : play(note)

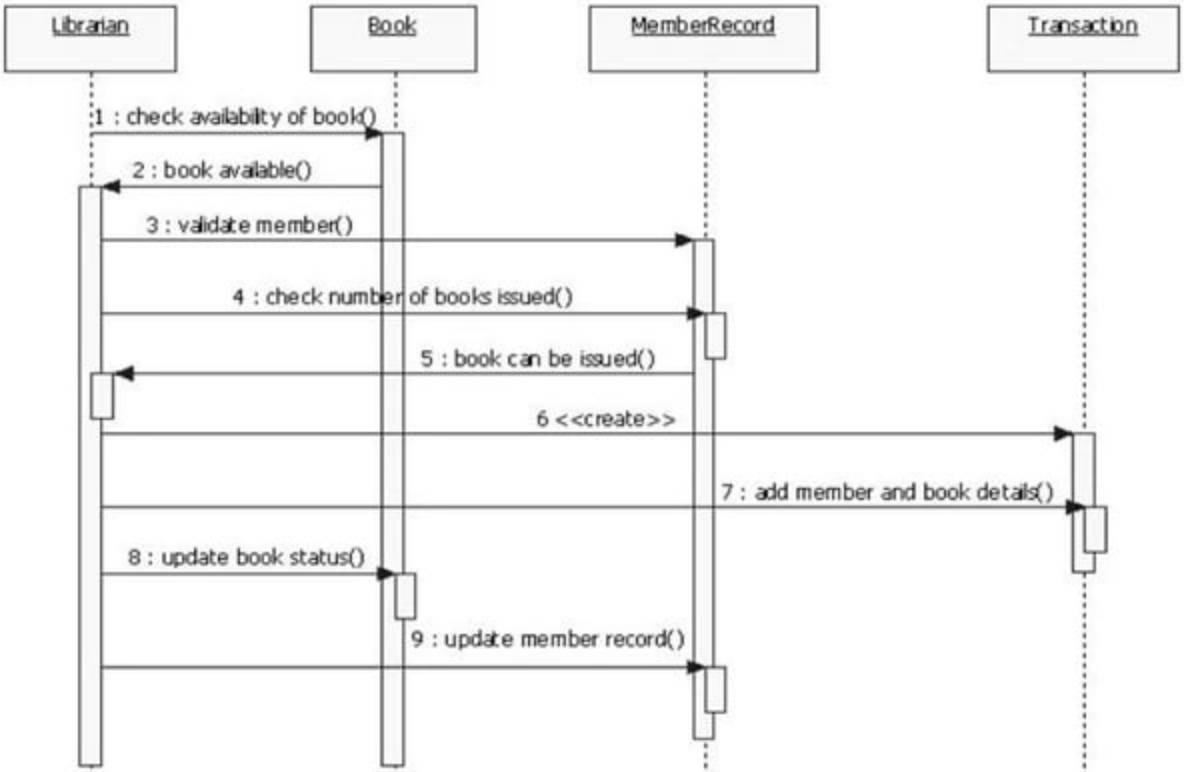

: SearchEngine
: Repository
loop
[hasNext]
1 : item := getNext()
2 : found := test(item)
break
[found]
3 : process(item)

Create and Destroy message

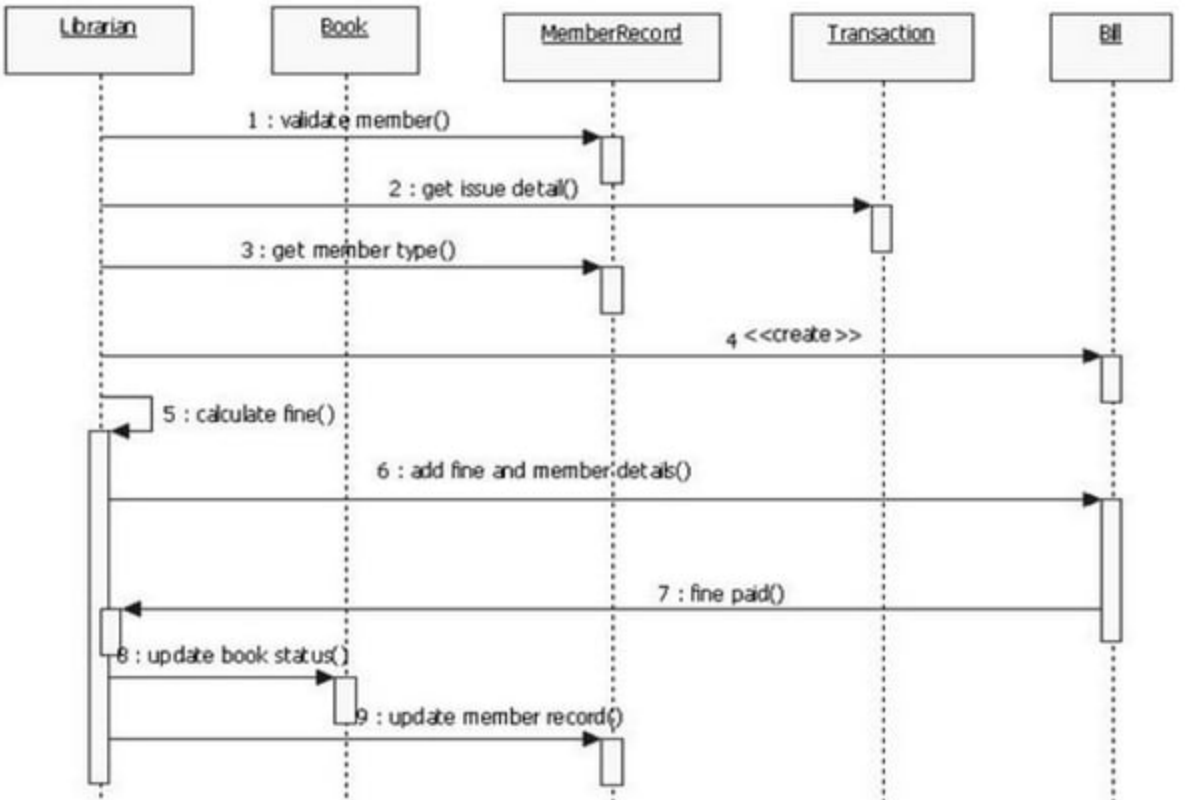
- A create message represents the creation of an instance in an interaction. The target lifeline begins at the point of the create message.
- A destroy message represents the destruction of an instance in an interaction. The target lifeline ends at the point of the destroy message, and is denoted by an X.

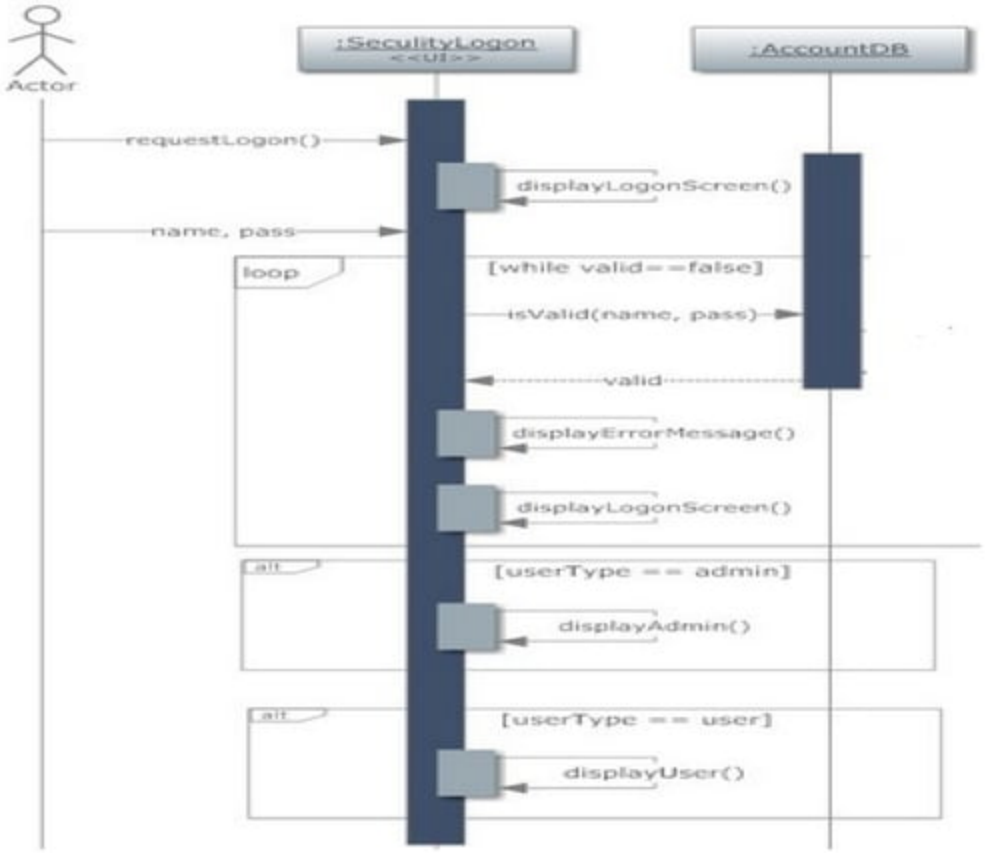

a Handler
new
a Query Command
creation
new
a Database Statement
execute
deletion from other object
results
extract results
close
results
self-deletion

Sequence diagram for issuing book



Sequence diagram for returning book



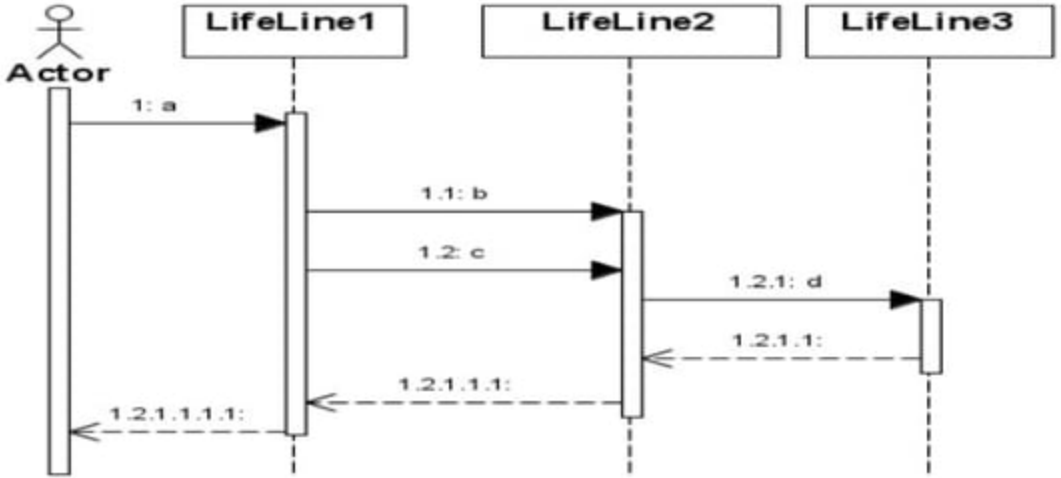

Actor
:SeculityLogon
<<UI>>
:AccountDB
requestLogon()
displayLogonScreen()
name, pass
loop
[while valid==false]
isValid(name, pass)
valid
displayErrorMessage()
displayLogonScreen()
alt
[userType == admin]
displayAdmin()
alt
[userType == user]
displayUser()

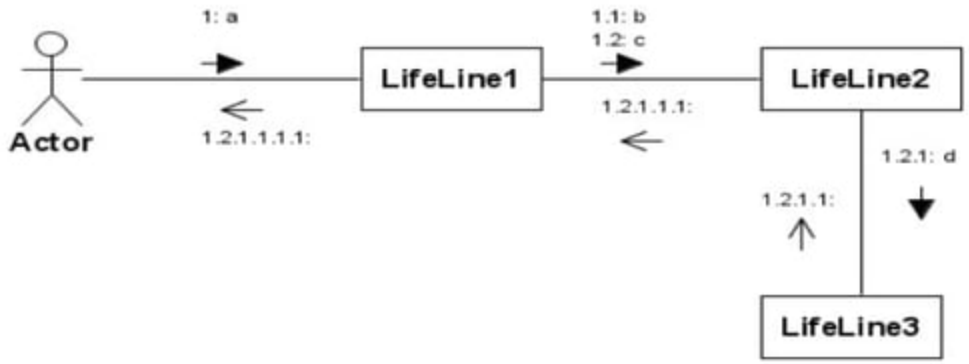
Communication diagram

- A communication diagram, is an interaction diagram that shows similar information to sequence diagrams but its primary focus is on object relationships.
- Formerly known as collaboration diagram.
- Shows interactions between objects and/or parts (represented as lifelines) using sequenced messages in a free-form arrangement.

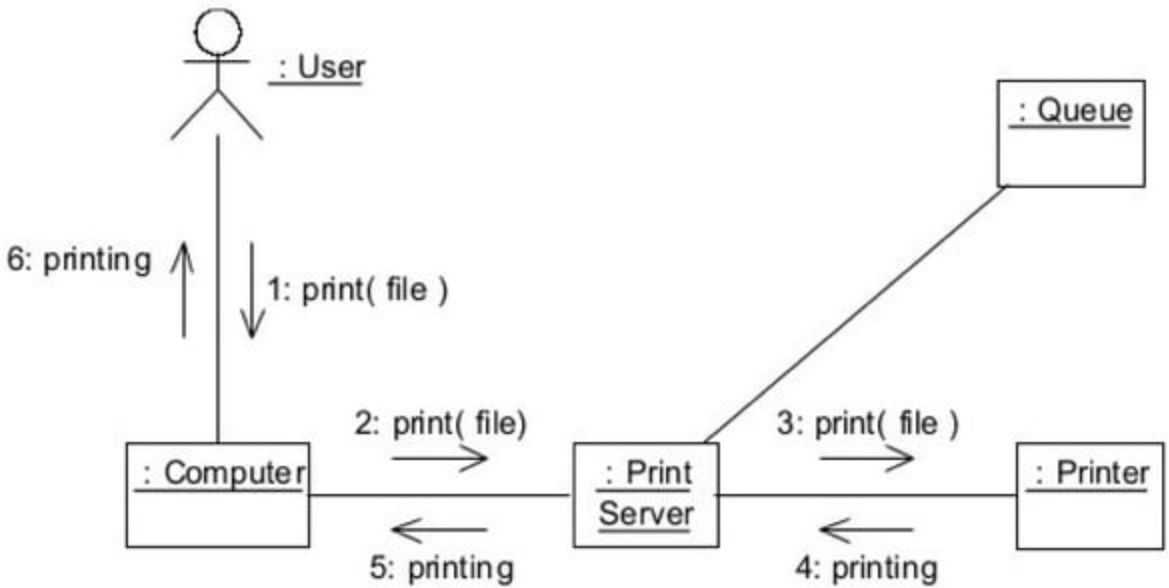
Communication diagram

- Model collaborations between objects or roles that deliver the functionalities of use cases and operations.
- Capture interactions that show the passed messages between objects and roles within the collaboration.
- Support the identification of objects (hence classes) that participate in use cases.

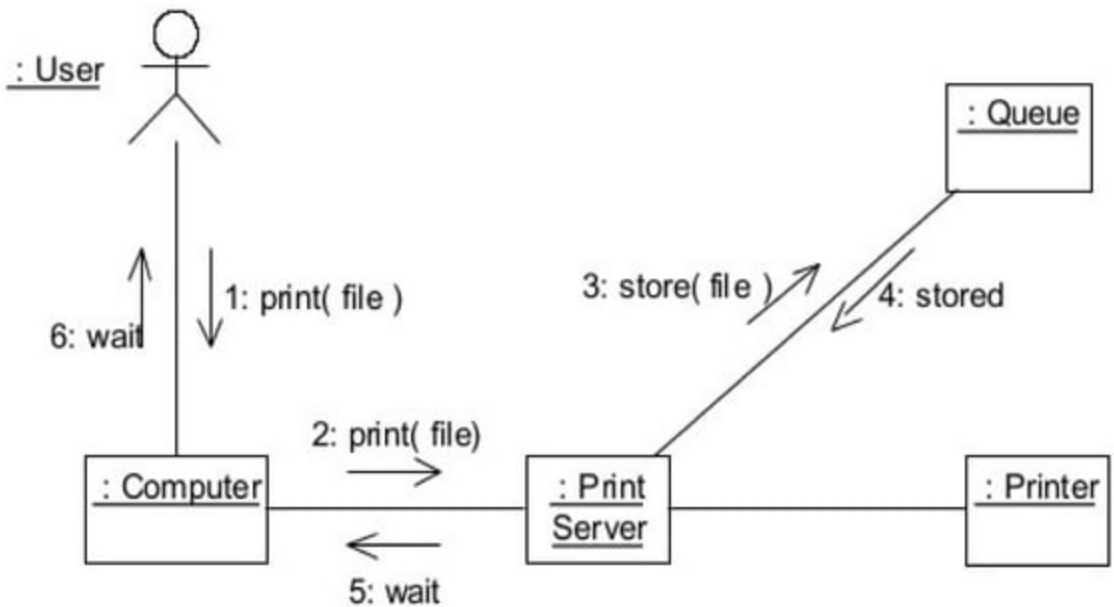

Actor
LifeLine1
LifeLine2
LifeLine3
1: a
1.1: b
1.2: c
1.2.1: d
1.2.1.1:
1.2.1.1.1:
1.2.1.1.1.1:


Actor
1: a
1.2.1.1.1.1:
LifeLine1
1.1: b
1.2: c
1.2.1.1.1:
LifeLine2
1.2.1: d
1.2.1.1:
LifeLine3

Communication diagram



Communication diagram



Which to Use?

- Choose sequence diagram when only the sequence of events needs to be shown and collaboration among the objects are priority.
- Choose a communication diagram when the objects and their links facilitate understanding the interactions (you don't have to put all objects at the top and make the lines all vertical or horizontal).
- Sequence diagrams emphasize the time ordering of messages, whereas communication diagrams depict more of an organizational structure and are more space efficient.

Which to Use?

- Although it is possible to derive the sequencing of messages in the communication diagram from the numbering scheme, it isn't immediately visible.
- What the communication diagram does show quite clearly though, is the full set of messages passed between adjacent objects.
- Communication diagrams offer a better view of a scenario than a Sequence diagram when the modeler is trying to understand all of the effects on a given object