Interaction Model:

Interaction model describes interaction among objects i.e. how individual objects collaborate to achieve the functionality of a system as a whole.

The class model describes classes and objects in a system and their relationships. The state model describes life cycle of the objects and the interaction model describes hoe the objects interact.

Interaction model is represented by

- Use-Case diagrams
- Sequence diagrams
- Activity diagrams.

Scenario:

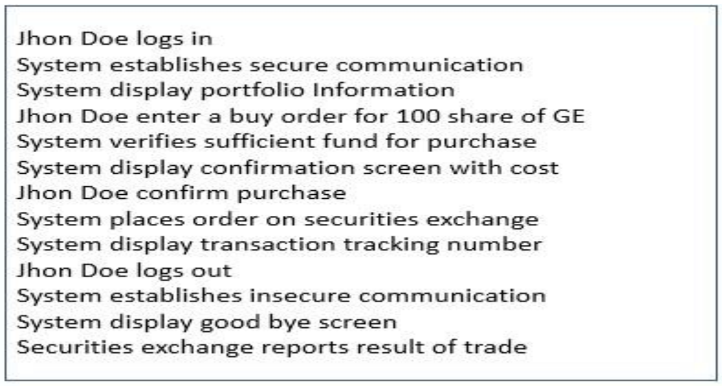
A scenario is a sequence of events that occurs during execution of a particular system functionality such as for a use case.

Scenario can be displayed as a list of text statements.

A scenario contains message between objects and activities performed by objects. Each message transmits information from one object to another.

The first step of writing a scenario is to identify the objects exchanging messages. Then the sender and receiver of the message will be determined and sequence of messages.

Consider the following example of a scenario .This represents a sequence of messages among objects like :customer, :StockBrokerSystem , :SecurityExchange. in a stock brokerage sytem



Jhon Doe logs in
System establishes secure communication
System display portfolio Information
Jhon Doe enter a buy order for 100 share of GE
System verifies sufficient fund for purchase
System display confirmation screen with cost
Jhon Doe confirm purchase
System places order on securities exchange
System display transaction tracking number
Jhon Doe logs out
System establishes insecure communication
System display good bye screen
Securities exchange reports result of trade

Scenario for Online Stock Broker Session

Senario doesn’t show clearly the sender and receiver of each message (especially if there are more than two objects).So,sequence diagram is useful to show interaction of objects through different types of messages.

Sequence Diagram:

A sequence diagram represents interaction between objects in a sequential order i.e. the order in which these interactions take place

Sequence diagrams model the interactions between objects in a single use case.

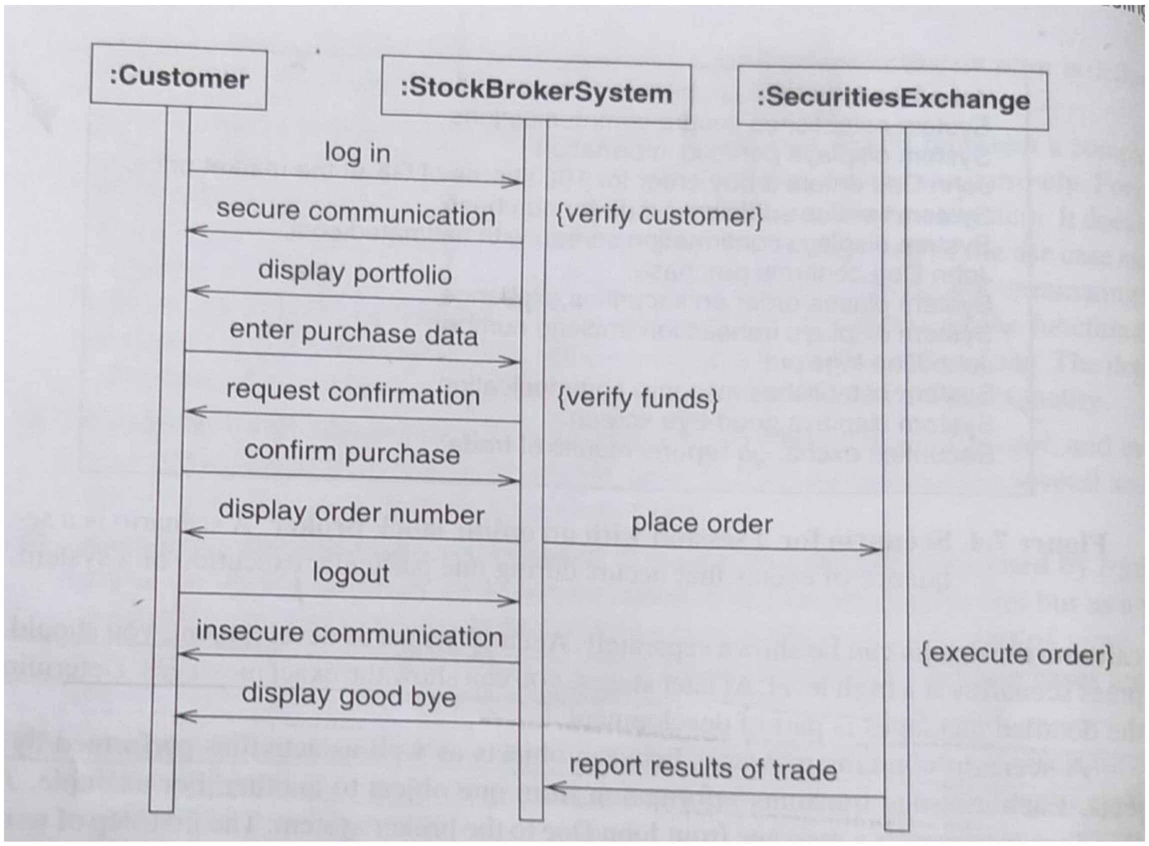
A Sequence diagram represents how the different parts of a system interact with each other to carry out functionality, and the order in which the interactions occur when a particular use case is executed.

The sequence diagram clearly displays the sender and receiver of the message and also the sequence of the messages exchanged which provide a clear vision of the interaction between them. Thereby the sequence diagram reveals how the actors interact with the system to execute all or part of the use case.

Sequence diagram has components like:

- The actor which are the entities that interact with the system or are external to the system.
- Lifeline shows the amount of time spent by an actor or system during the interaction.
- The message is the information sent by the sender to the receiver. The message is represented by a horizontal arrow in the sequence diagram where the arrowhead is towards the receiver of the message.
- Messages can be synchronous which means the sender waits for the receiver to process the sent message before proceeding further.
- Messages can be asynchronous where the sender proceeds without waiting for the receiver to complete the processing of the sent message.

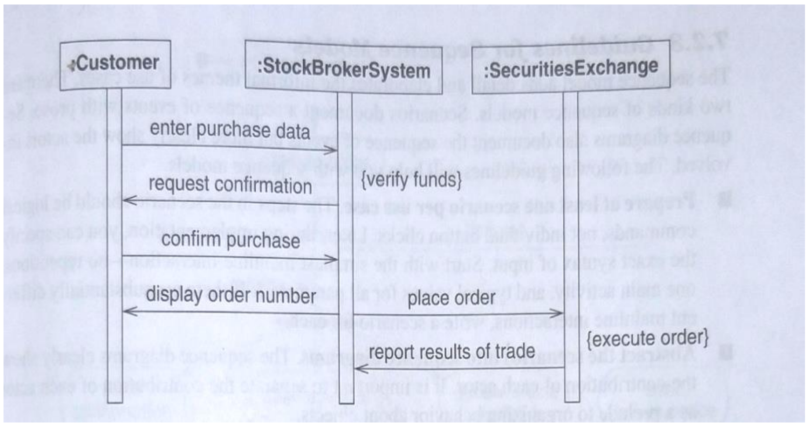
The figure below shows the sequence diagram for the scenario that we have discussed above of a stock broker.



(Sequence diagram for online stock brokerage system)

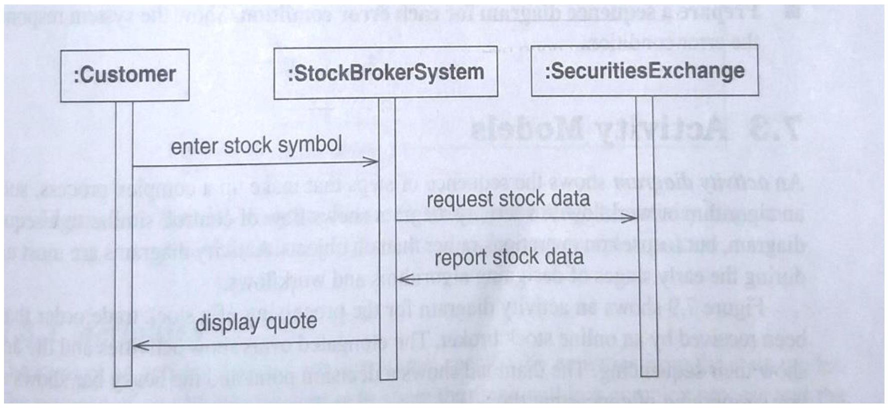
It is not compulsory that you should describe a use case in a single sequence diagram. It’s always better to show a portion of the use case, in a sequence diagram as large-scale interactions may have several independent tasks that could be combined in many ways.

So, instead of getting it messy by repeating the messages, we can implement a separate sequence diagram for each task. Like the figure below shows the sequence diagram for two independent tasks stock purchase and stock quote.



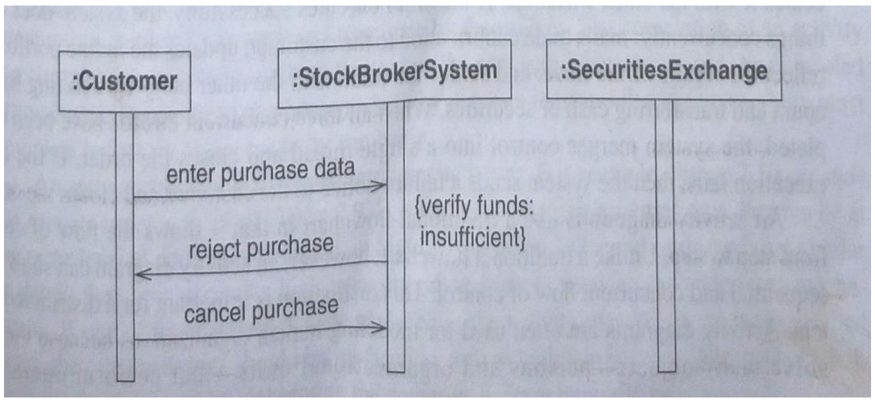
(Sequence diagram for stock purchase)

Sequence diagram for stock quote is as follows



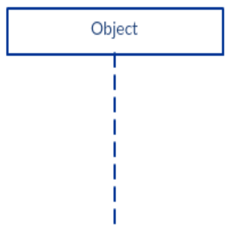
(Sequence diagram for stock quote)

Every use case would have an exceptional condition for which a separate sequence diagram should be implemented. Like the sequence diagram below shows the exceptional case where a purchasing of the stock fails.



Sequence diagram notations.

Lifeline Notation

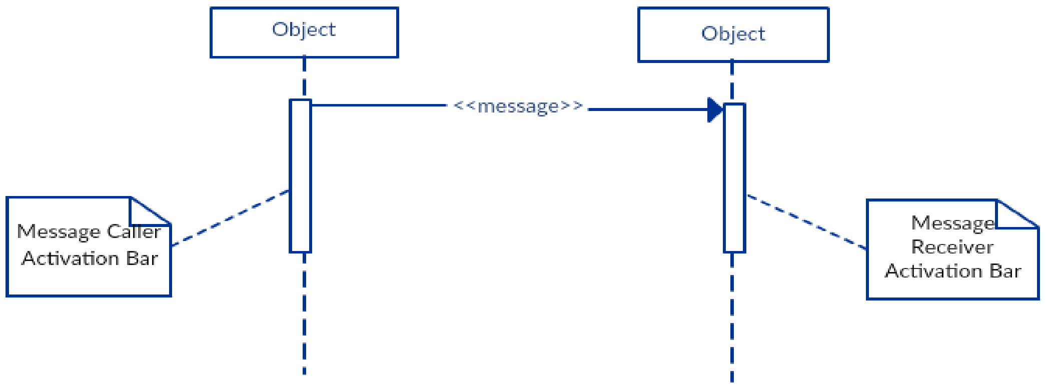


A sequence diagram is made up of several of these lifeline notations that should be arranged horizontally across the top of the diagram. No two lifeline notations should overlap each other. They represent the different objects or parts that interact with each other in the system during the sequence.

Activation Bars

Activation bar is the box placed on the lifeline. It is used to indicate that an object is active (or instantiated) during an interaction between two objects. The length of the rectangle indicates the duration of the objects staying active.

In a sequence diagram, an interaction between two objects occurs when one object sends a message to another. The use of the activation bar on the lifelines of the Message Caller (the object that sends the message) and the Message Receiver (the object that receives the message) indicates that both are active/is instantiated during the exchange of the message.


Object
Object
<<message>>
Message Caller Activation Bar
Message Receiver Activation Bar

Message Arrows

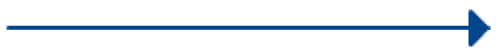
An arrow from the Message Caller to the Message Receiver specifies a message in a sequence diagram. A message can flow in any direction; from left to right, right to left or back to the Message Caller itself. While you can describe the message being sent from one object to the other on the arrow, with different arrowheads you can indicate the type of message being sent or received.

The message arrow comes with a description, which is known as a message signature, on it. The format for this message signature is below. All parts except the message_name are optional.

attribute = message_name (arguments): return_type

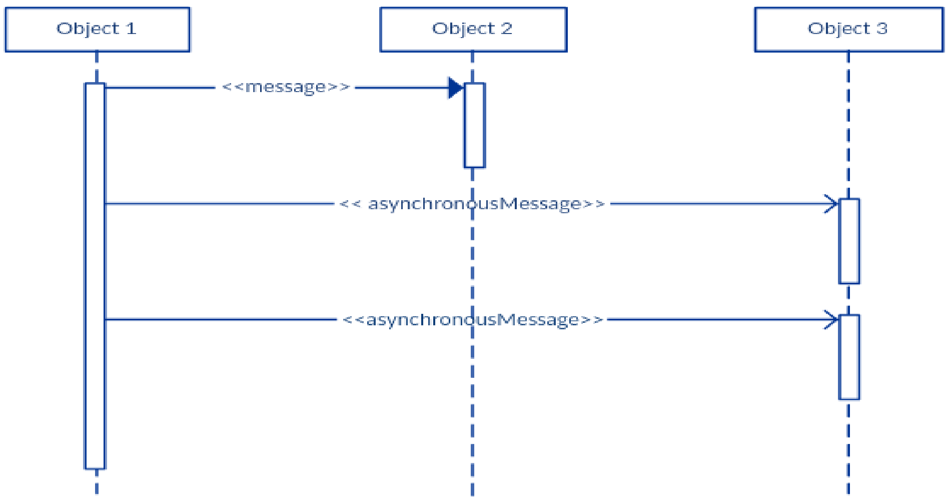
- ***Synchronous message***

As shown in the activation bars example, a synchronous message is used when the sender waits for the receiver to process the message and return before carrying on with another message. The arrowhead used to indicate this type of message is a solid one, like the one below.



- ***Asynchronous message***

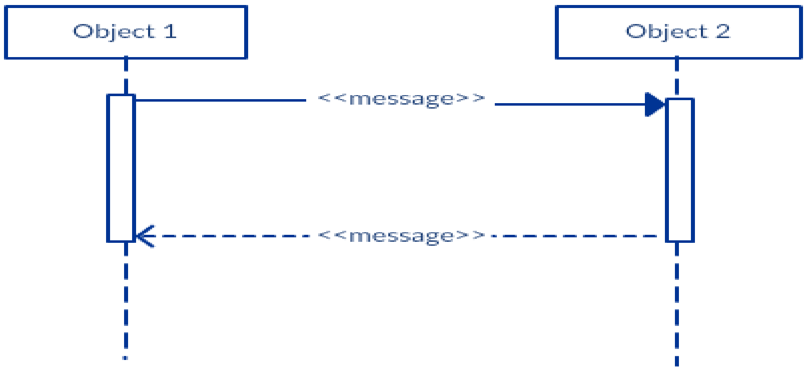
An asynchronous message is used when the message caller does not wait for the receiver to process the message and return before sending other messages to other objects within the system. The arrowhead used to show this type of message is a line arrow like shown in the example below.



- ***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 messages are optional notation pieces, for an activation bar that is triggered by a synchronous message always implies a return message.

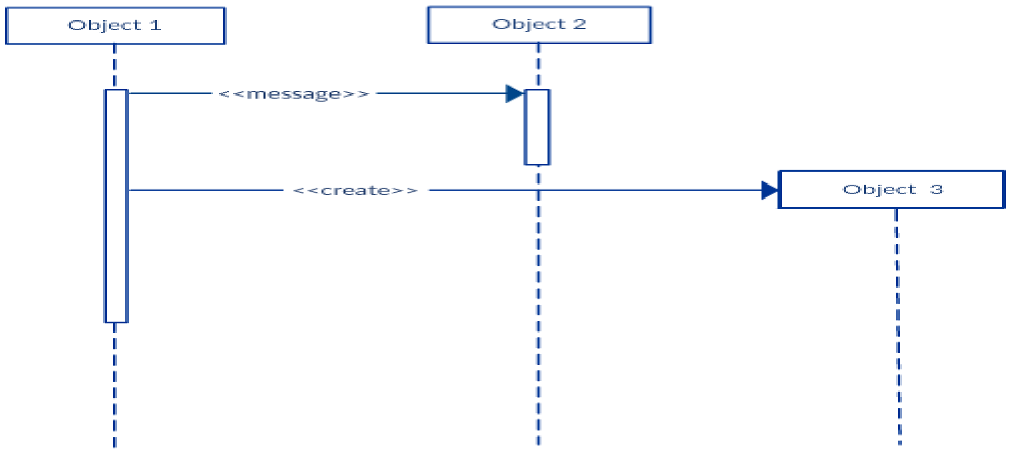
Tip: You can avoid cluttering up your diagrams by minimizing the use of return messages since the return value can be specified in the initial message arrow itself.



- ***Participant creation message***

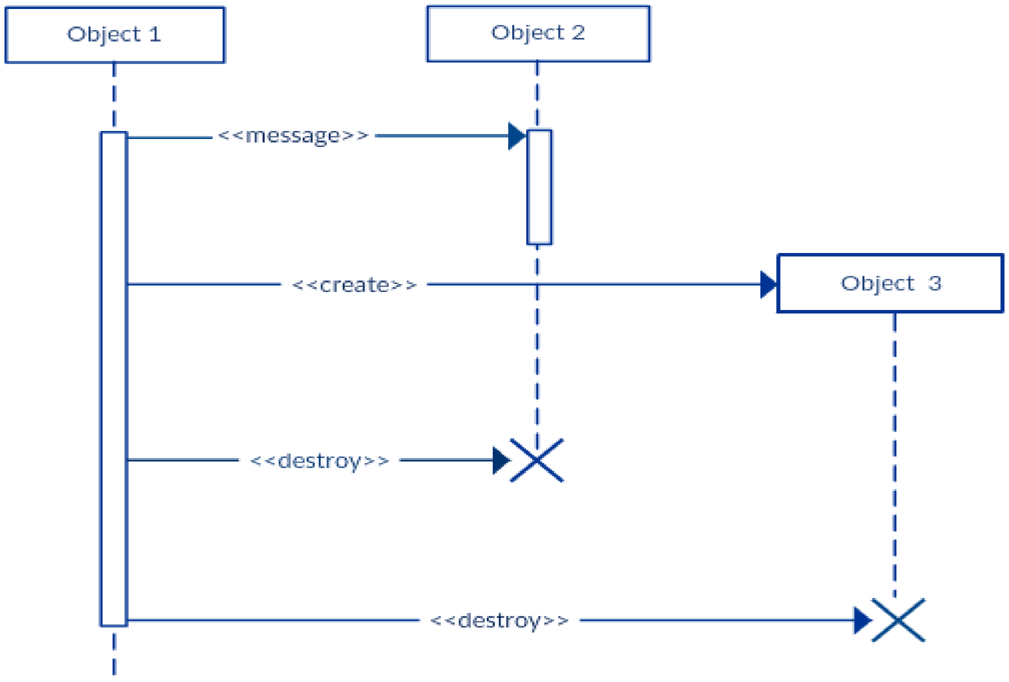
Objects do not necessarily live for the entire duration of the sequence of events. Objects or participants can be created according to the message that is being sent.

The dropped participant box notation can be used when you need to show that the particular participant did not exist until the create call was sent. If the created participant does something immediately after its creation, you should add an activation box right below the participant box.



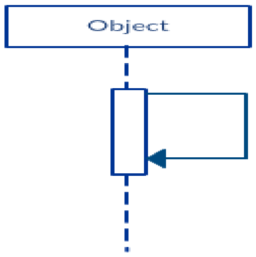
- ***Participant destruction message***

Likewise, participants when no longer needed can also be deleted from a sequence diagram. This is done by adding an ‘X’ at the end of the lifeline of the said participant.



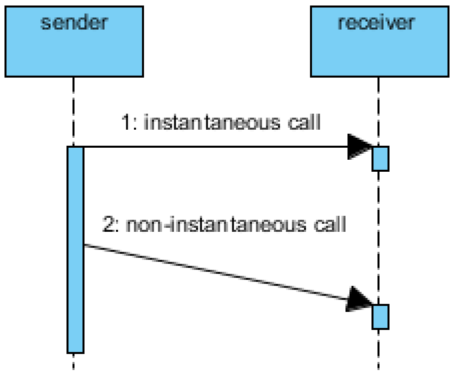
- ***Reflexive message***

When an object sends a message to itself, it is called a reflexive message. It is indicated with a message arrow that starts and ends at the same lifeline as shown in the example below.



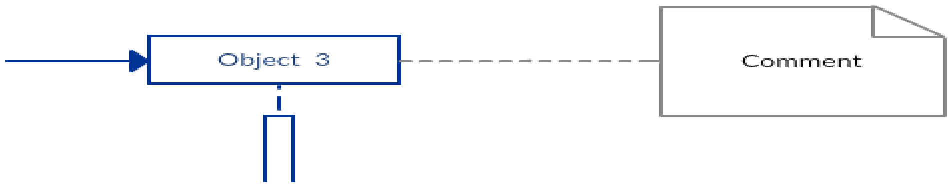
- ***Non instantaneous message***

Messages are often considered to be instantaneous, thus, the time it takes to arrive at the receiver is negligible. The messages are drawn as a horizontal arrow. To indicate that it takes a certain while before the receiver actually receives a message, a **slanted arrow is used**.



- ***Comment***

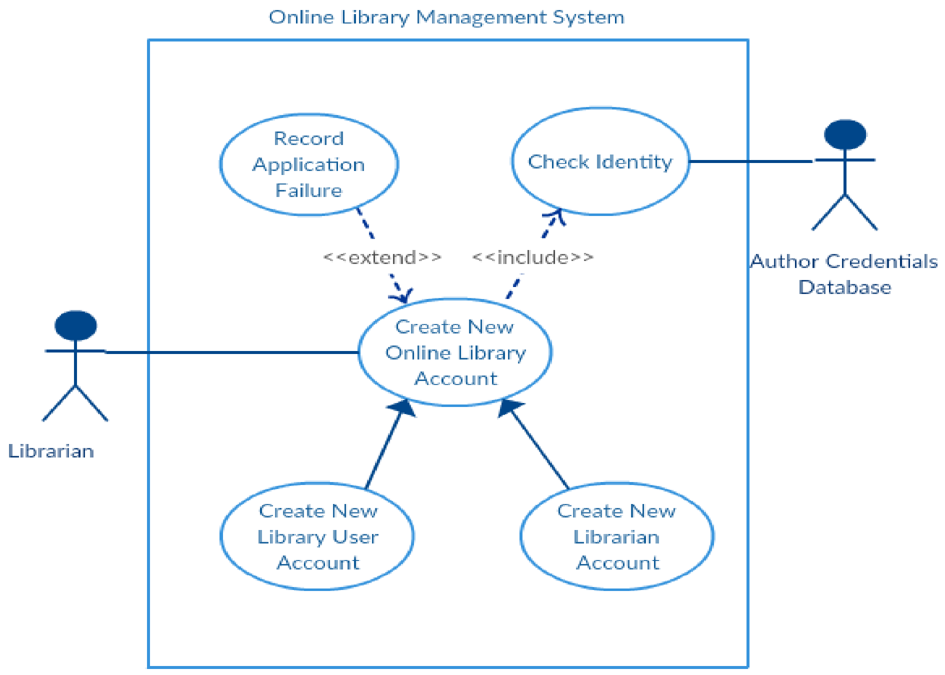
UML diagrams generally permit the annotation of comments in all UML diagram types. The comment object is a rectangle with a folded-over corner as shown below. The comment can be linked to the related object with a dashed line.



How to Draw a Sequence Diagram

A sequence diagram represents the scenario or flow of events in one single use case. The message flow of the sequence diagram is based on the narrative of the particular use case.

Then, before you start drawing the sequence diagram or decide what interactions should be included in it, you need to draw the use case diagram and ready a comprehensive description of what the particular use case does.


Online Library Management System
Record Application Failure
Check Identity
<<extend>>
<<include>>
Create New Online Library Account
Create New Library User Account
Create New Librarian Account
Librarian
Author Credentials Database

From the above use case diagram example of 'Create New Online Library Account', we will focus on the use case named 'Create New User Account' to draw our sequence diagram example.

Before drawing the sequence diagram, it's necessary to identify the objects or actors that would be involved in creating a new user account. These would be;

- Librarian
- Online Library Management system
- User credentials database
- Email system

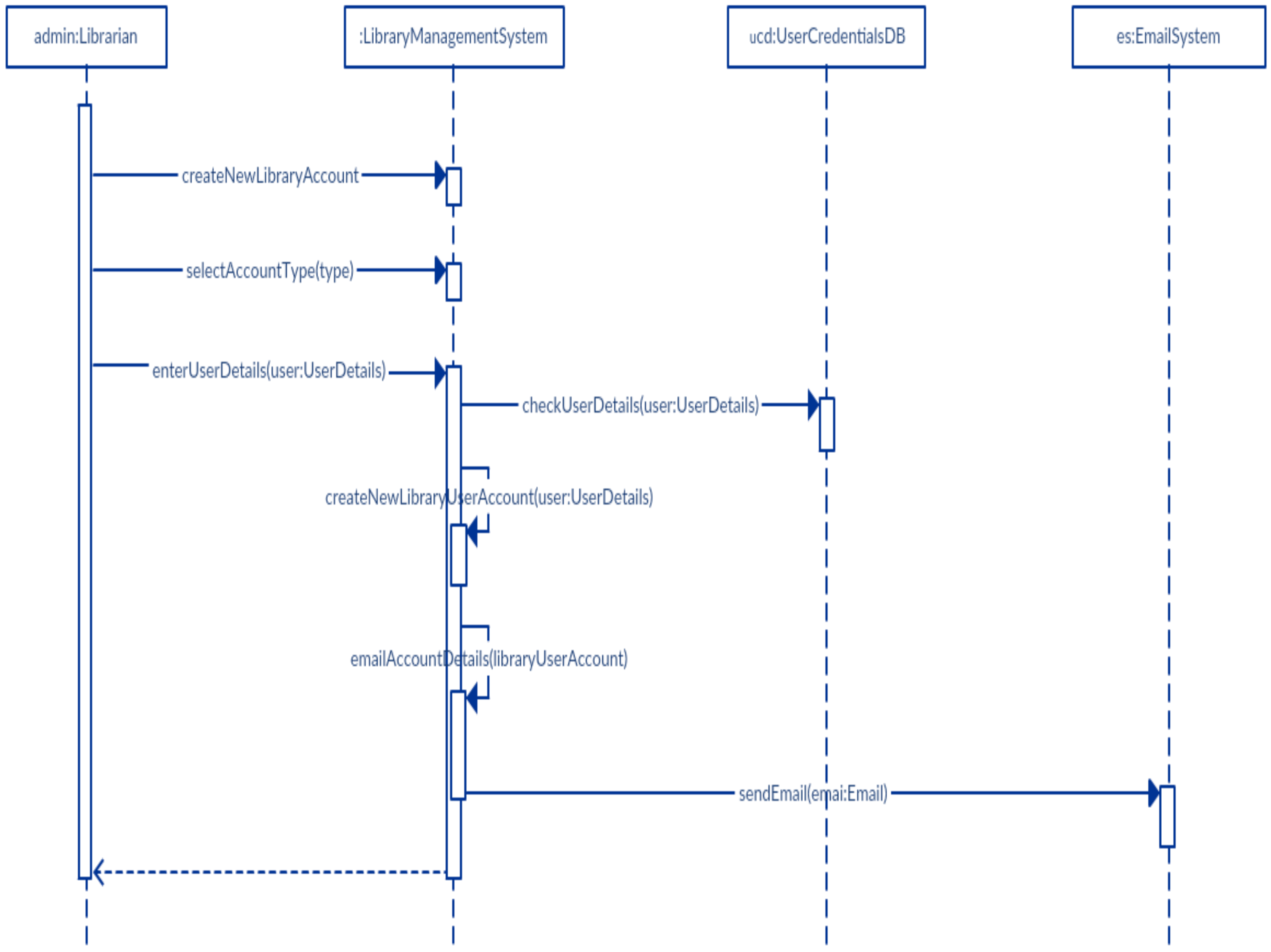
Once you identify the objects, it is then important to write a detailed description on what the use case does. From this description, you can easily figure out the interactions (that should go in the sequence diagram) that would occur between the objects above, once the use case is executed.

Here are the steps that occur in the use case named ‘Create New Library User Account’.

- The librarian request the system to create a new online library account
- The librarian then selects the library user account type
- The librarian enters the user’s details
- The user’s details are checked using the user Credentials Database
- The new library user account is created
- A summary of the of the new account’s details are then emailed to the user

From each of these steps, you can easily specify what messages should be exchanged between the objects in the sequence diagram. Once it’s clear, you can go ahead and start drawing the sequence diagram.

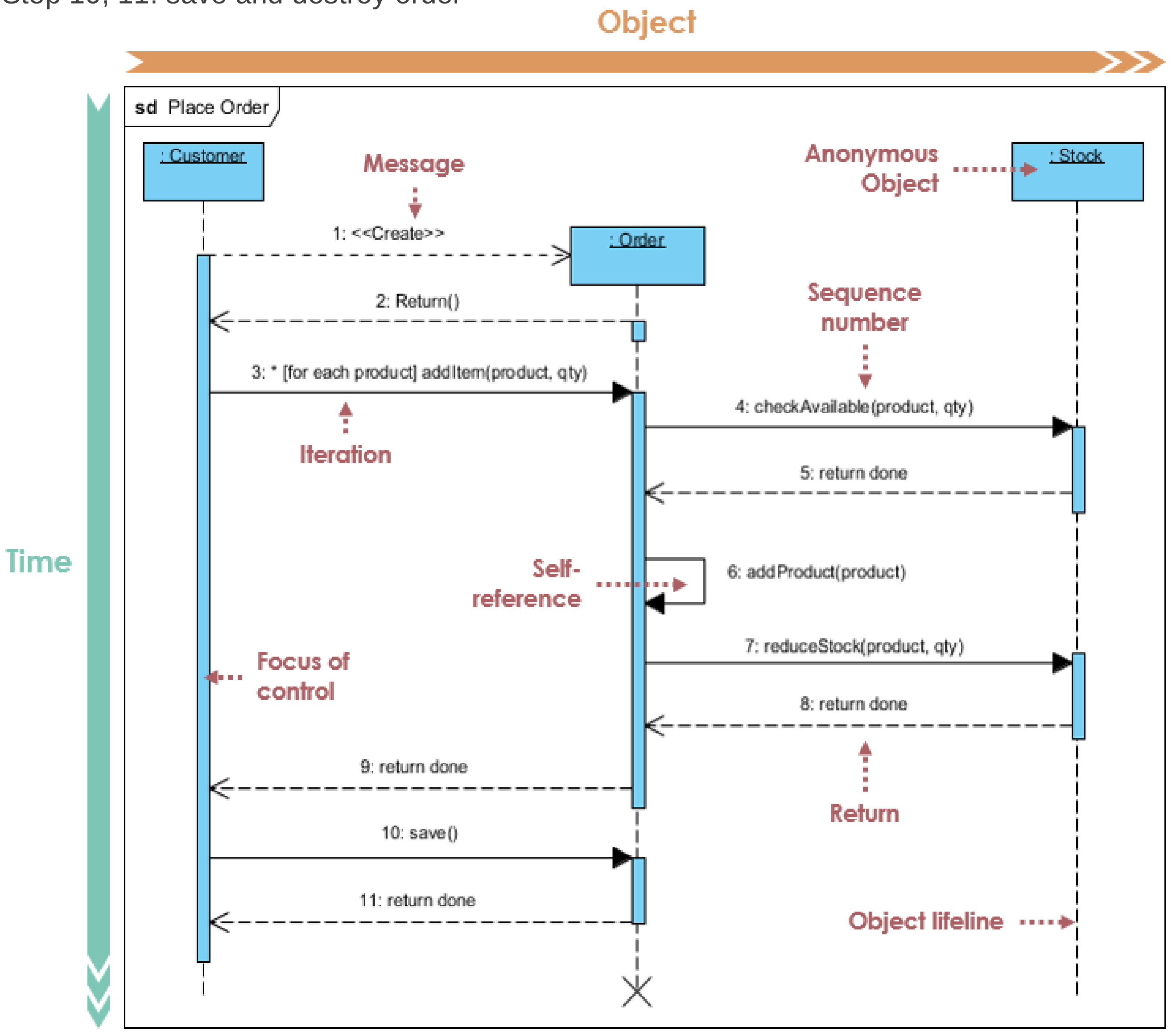
The sequence diagram below shows how the objects in the online library management system interact with each other to perform the function ‘Create New Library User Account’.


admin:Librarian
:LibraryManagementSystem
ucd:UserCredentialsDB
es:EmailSystem
createNewLibraryAccount
selectAccountType(type)
enterUserDetails(user:UserDetails)
checkUserDetails(user:UserDetails)
createNewLibraryUserAccount(user:UserDetails)
emailAccountDetails(libraryUserAccount)
sendEmail(emai:Email)

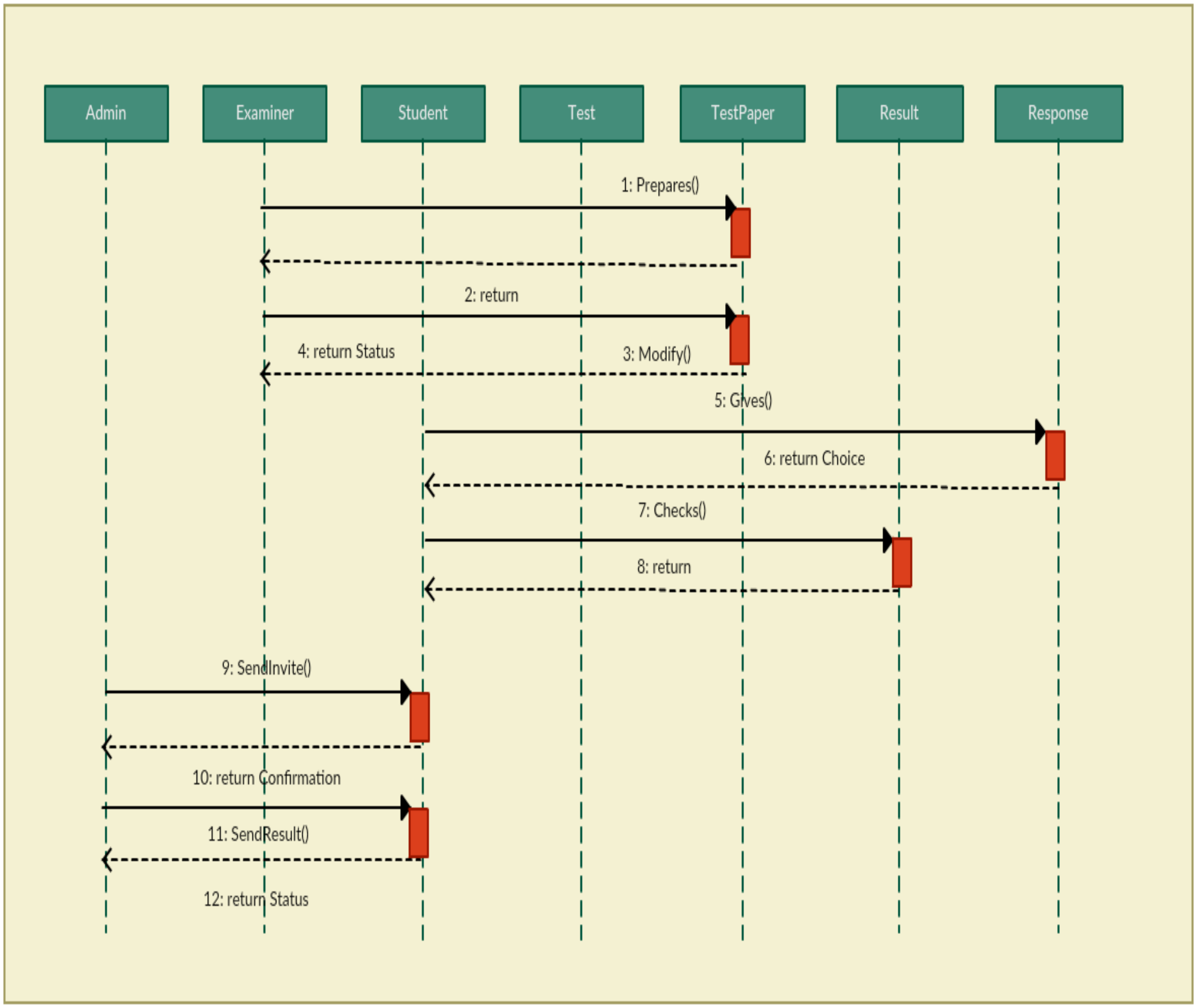
Example: Place Order for Stocks

The example shows a Sequence diagram with three participating objects: Customer, Order, and the Stock. Without even knowing the notation formally, you can probably get a pretty good idea of what is going on.

- Step 1 and 2: Customer creates an order.
- Step 3: Customer add items to the order.
- Step 4, 5: Each item is checked for availability in inventory.
- Step 6, 7, 8 : If the product is available, it is added to the order.
- Step 9 return
- Step 10, 11: save and destroy order

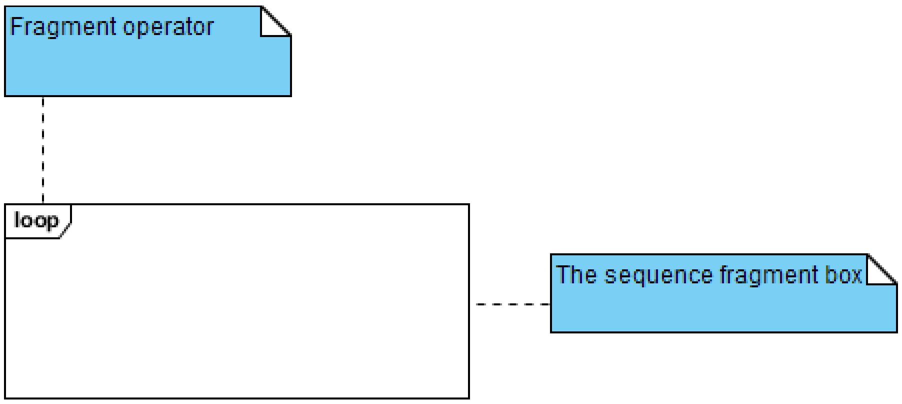


Example: Sequence Diagram of an Online Exam System



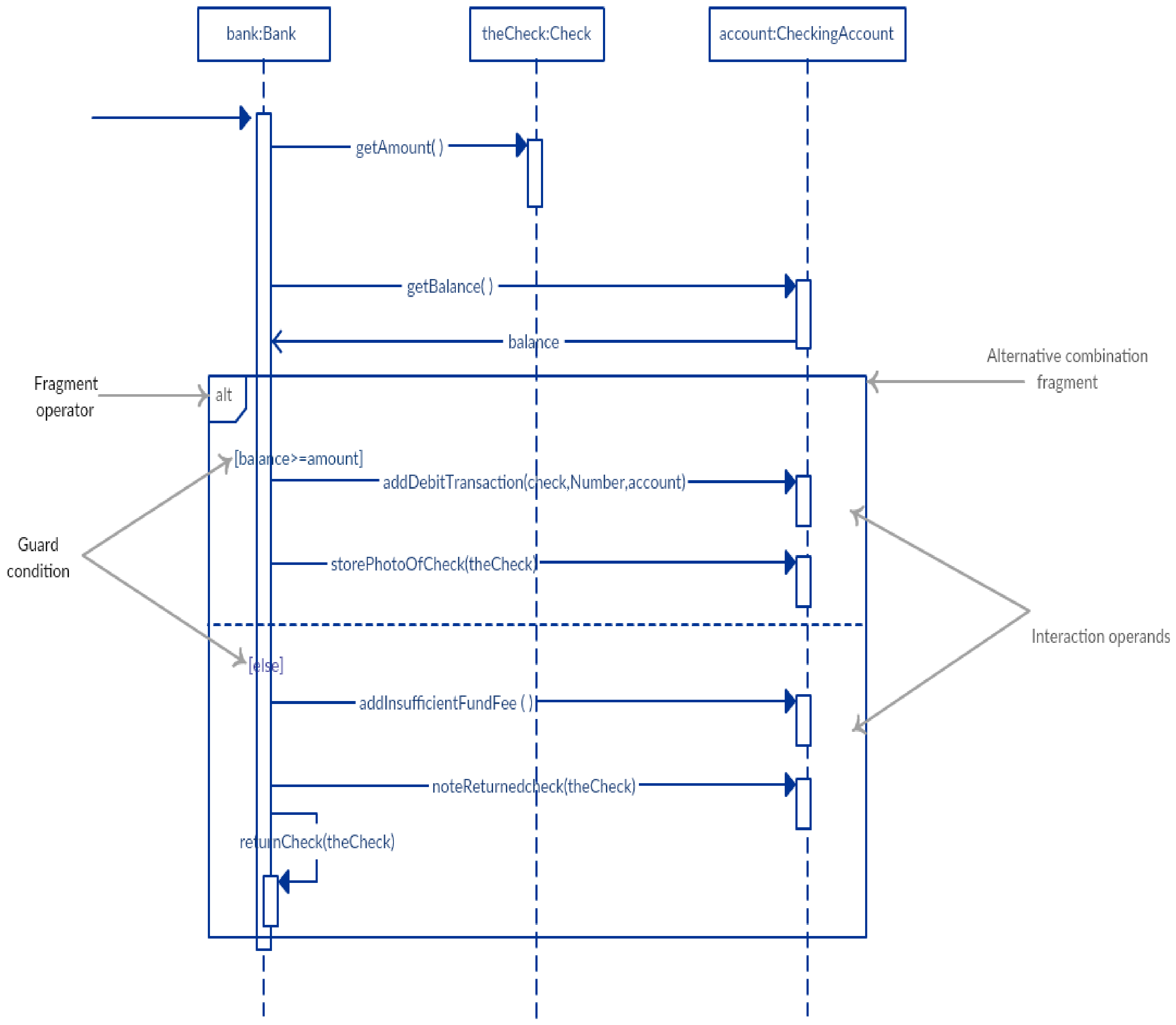
Sequence Fragments

- **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

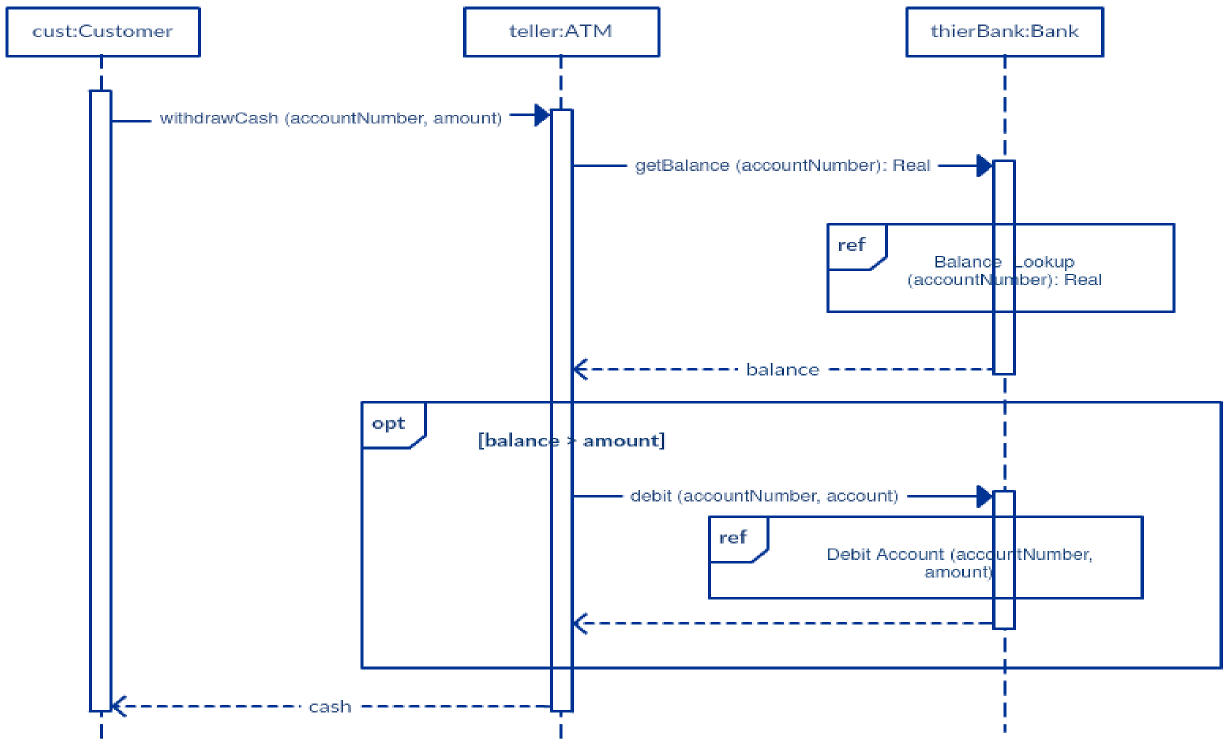


alt	Alternative multiple fragments: only the one whose condition is true will execute. *(if…then…else logic)*
opt	Optional: the fragment executes only if the supplied condition is true. Equivalent to an alt only with one trace. (*if…then logic*)
par	Parallel: each fragment is run in parallel.
loop	Loop: 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.

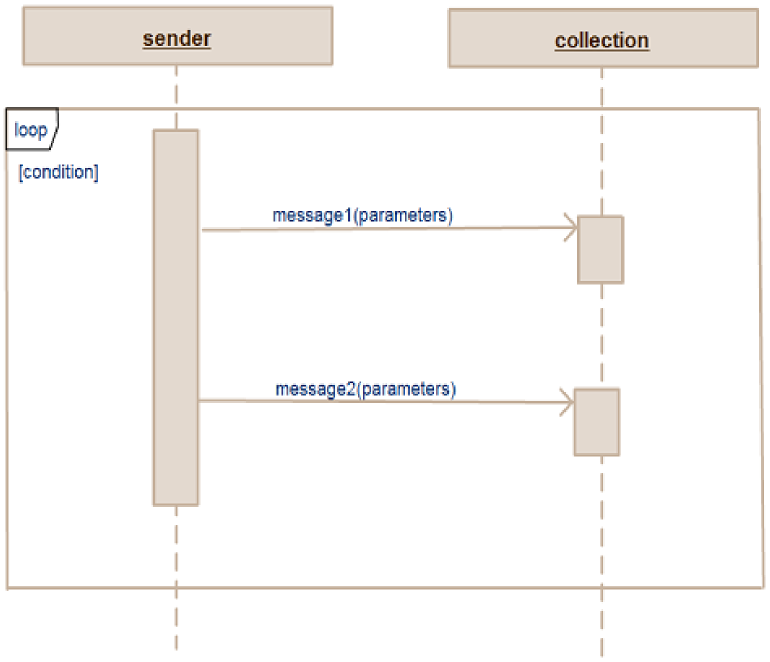
Example of sequence diagram with *alt* Fragments:

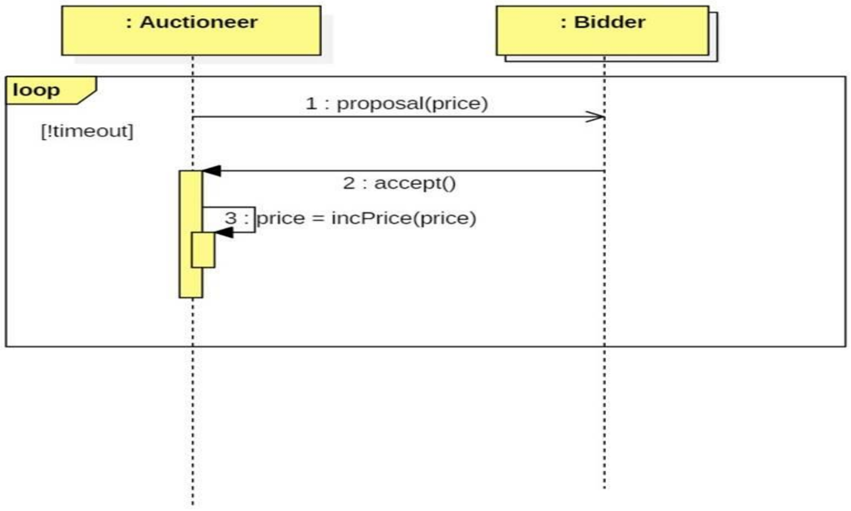


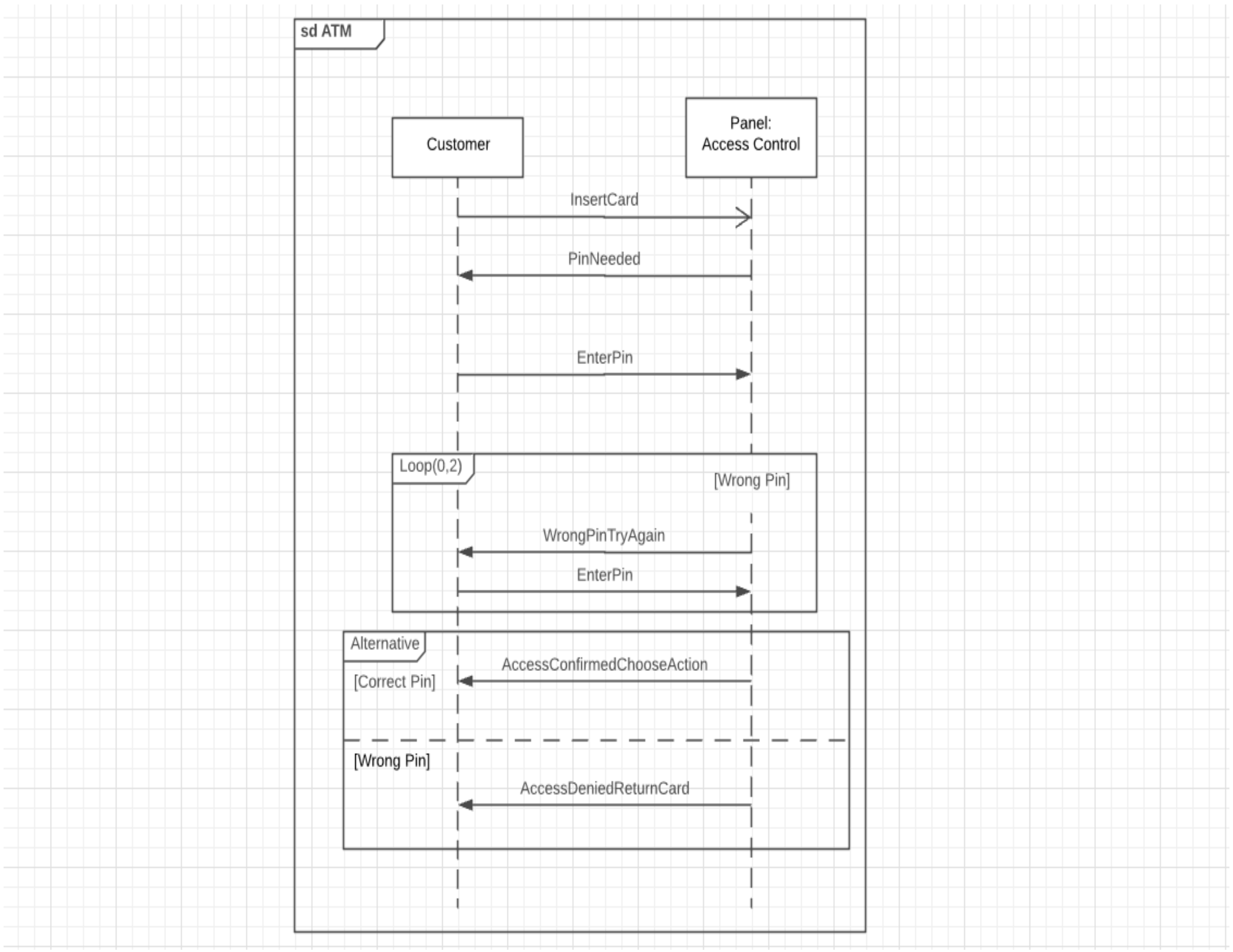
Example of sequence diagram with *opt and ref* Fragments:

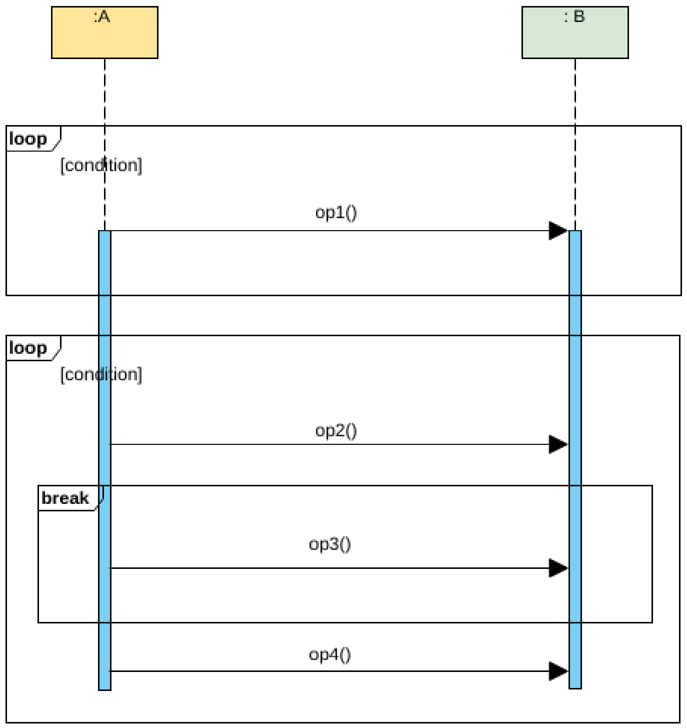

cust:Customer
teller:ATM
thierBank:Bank
withdrawCash (accountNumber, amount)
getBalance (accountNumber): Real
ref
Balance Lookup (accountNumber): Real
balance
opt
[balance > amount]
debit (accountNumber, account)
ref
Debit Account (accountNumber, amount)
cash

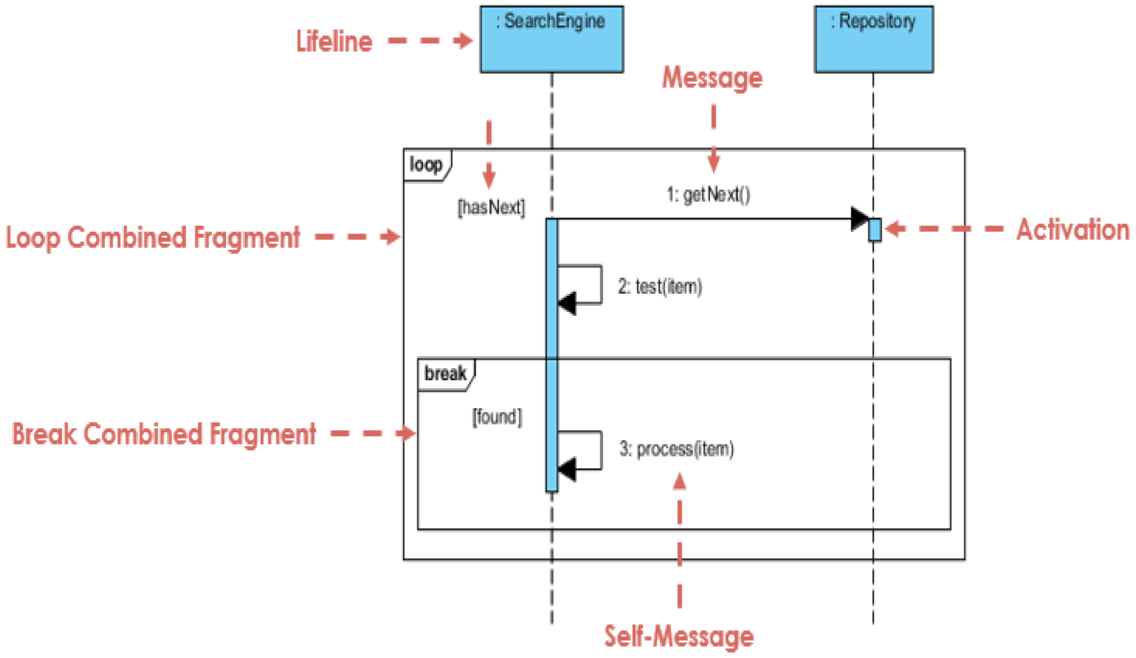
Loop Fragments Examples:

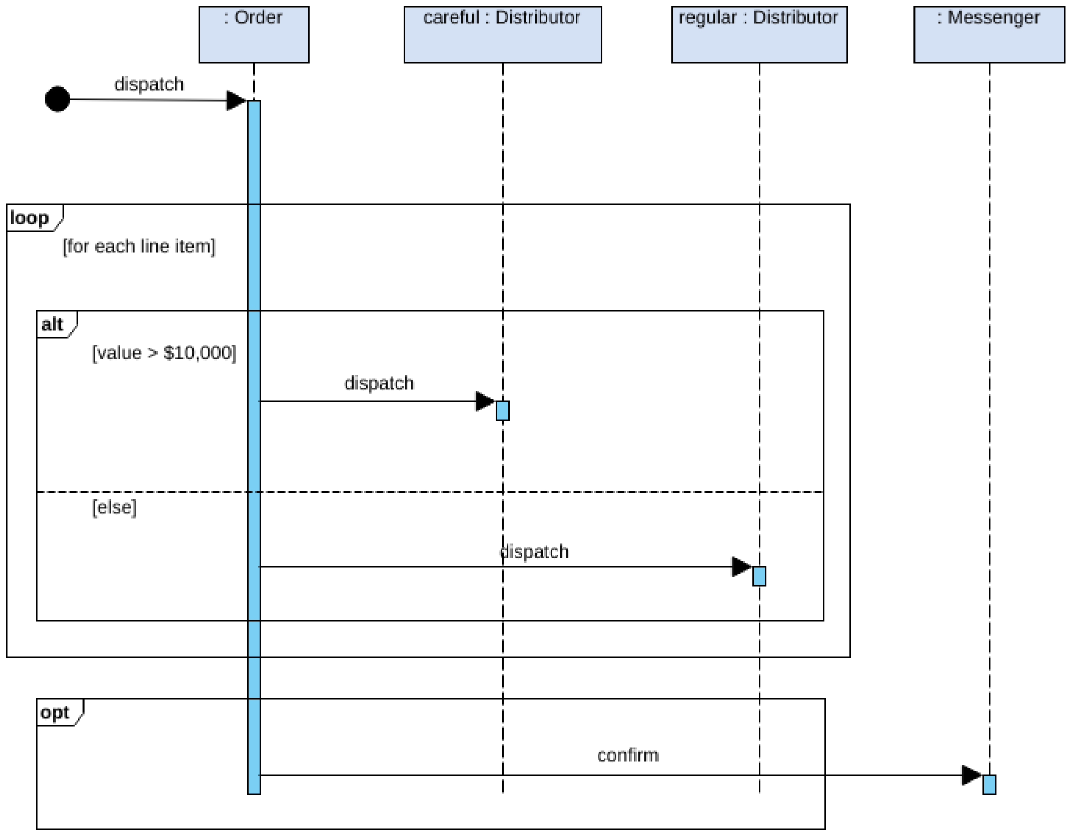



: Auctioneer
: Bidder
loop
[!timeout]
1 : proposal(price)
2 : accept()
3 : price = incPrice(price)

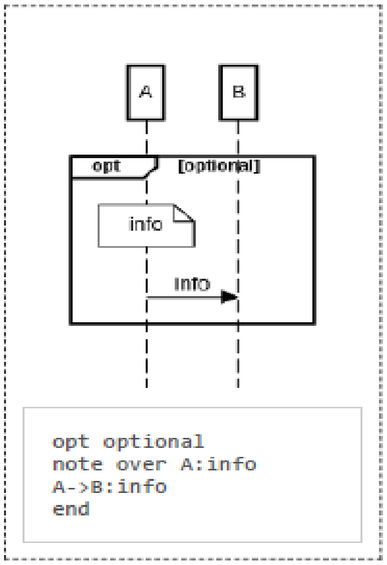

sd ATM
Customer
Panel:
Access Control
InsertCard
PinNeeded
EnterPin
Loop(0,2)
[Wrong Pin]
WrongPinTryAgain
EnterPin
Alternative
[Correct Pin]
AccessConfirmedChooseAction
[Wrong Pin]
AccessDeniedReturnCard


:A
: B
loop
[condition]
op1()
loop
[condition]
op2()
break
op3()
op4()

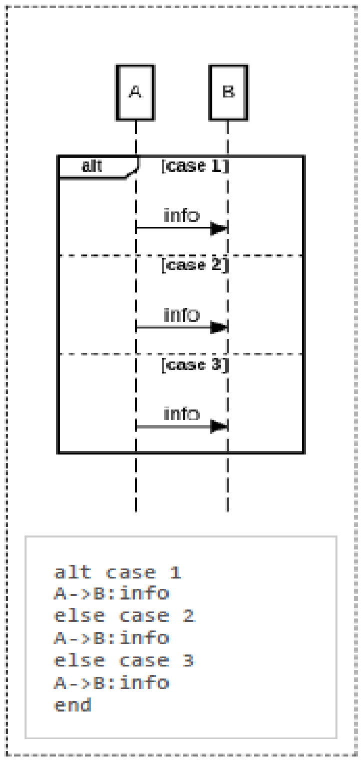

Lifeline
: SearchEngine
: Repository
Message
loop
[hasNext]
1: getNext()
Loop Combined Fragment
Activation
2: test(item)
break
[found]
Break Combined Fragment
3: process(item)
Self-Message


: Order
careful : Distributor
regular : Distributor
: Messenger
dispatch
loop
[for each line item]
alt
[value > $10,000]
dispatch
[else]
dispatch
opt
confirm

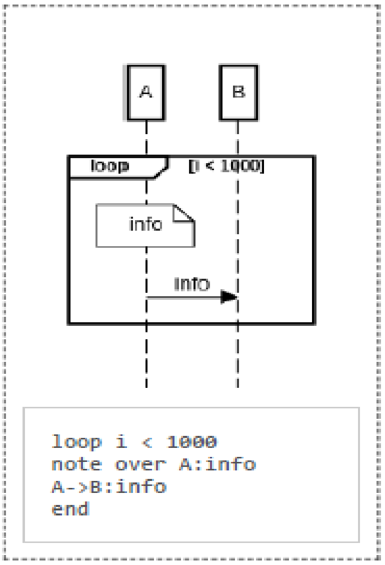
Summary of opt, alt anf loop fragments



```
opt optional
note over A:info
A->B:info
end
```



```
alt case 1
A->B:info
else case 2
A->B:info
else case 3
A->B:info
end
```



```
loop i < 1000
note over A:info
A->B:info
end
```