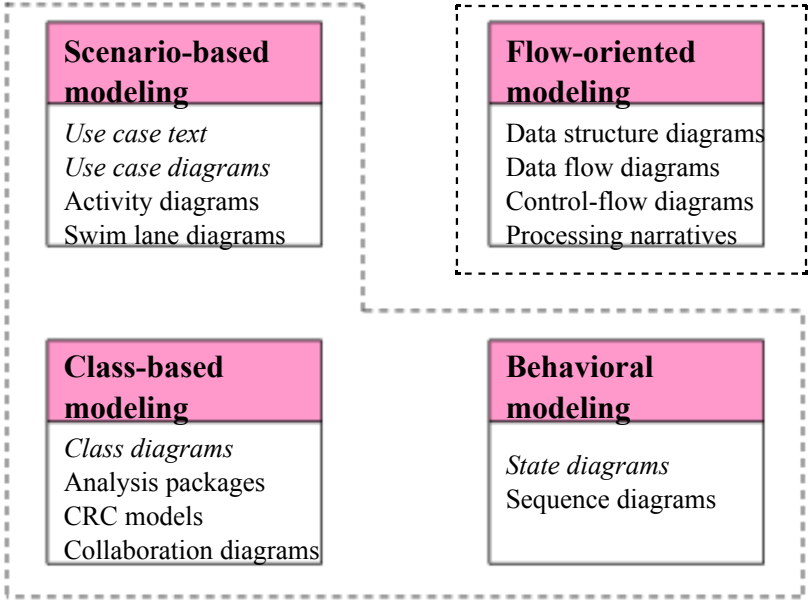
Elements of the Analysis Model

Object-oriented Analysis Procedural/Structured Analysis



Sequence Diagrams

Interaction Diagrams

- 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.

Interaction Diagrams (Cont.)

- 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.)

- UML
- Collaboration Diagrams
- Emphasizes structural relations between objects
- Sequence Diagram

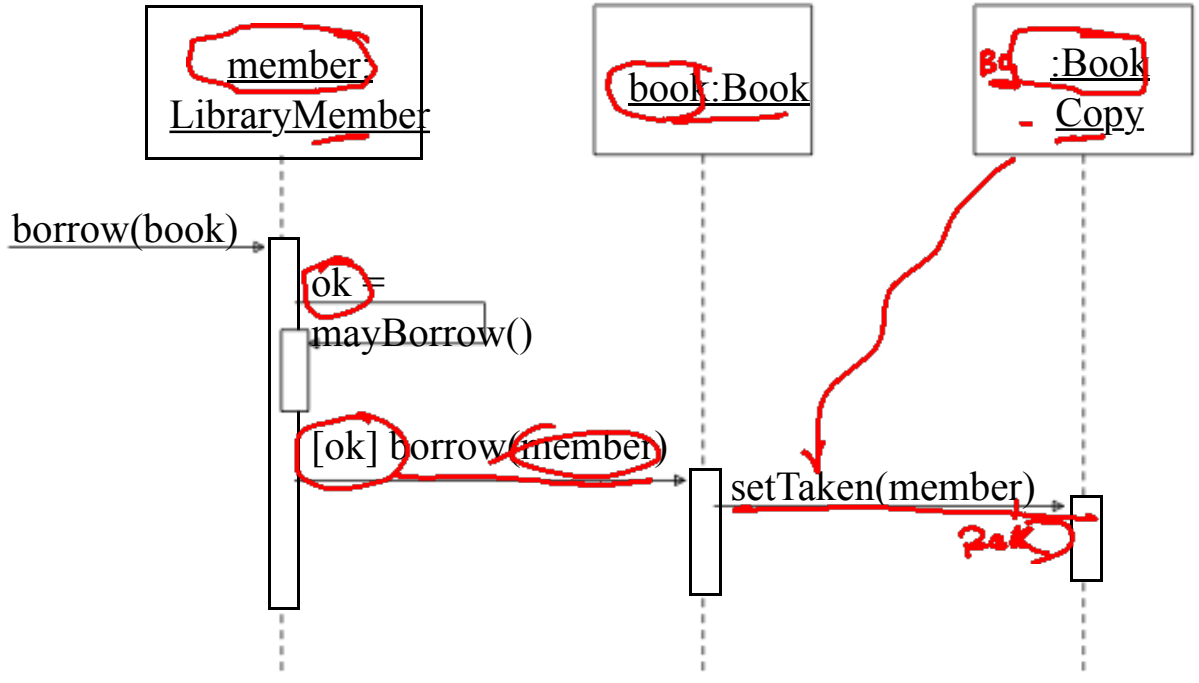
Importance of Sequence Diagrams

- Depict object interactions in a given scenario identified for a given Use Case
- Specify the messages passed between objects using horizontal arrows including messages to/from external actors
- Time increases from Top to bottom

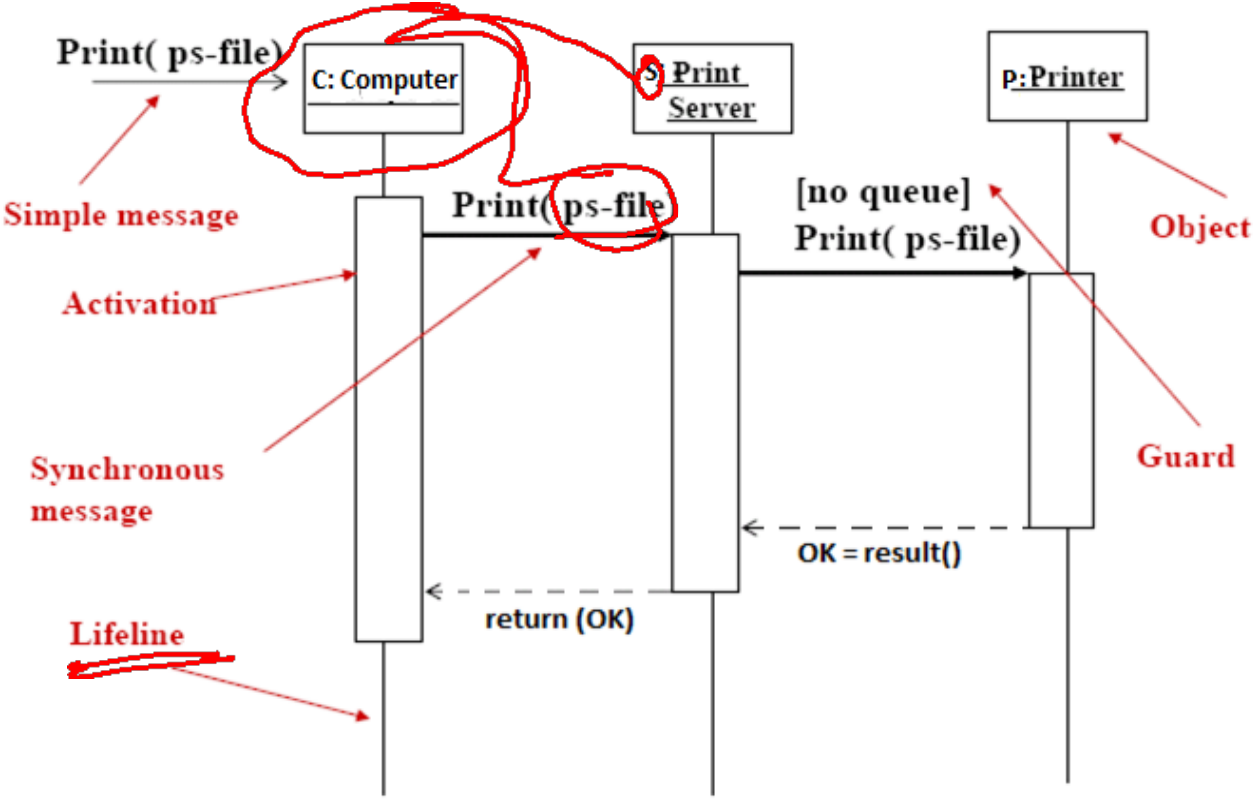
Sequence Diagrams

- Sequence diagrams illustrate how objects interact with each other.
- They focus on message sequences, that is, how messages are sent and received between a number of objects.
- The Branches, Conditions, and Loops may be included.
- Emphasizes time ordering of messages.
- Can model simple sequential flow, branching, iteration, recursion and concurrency.

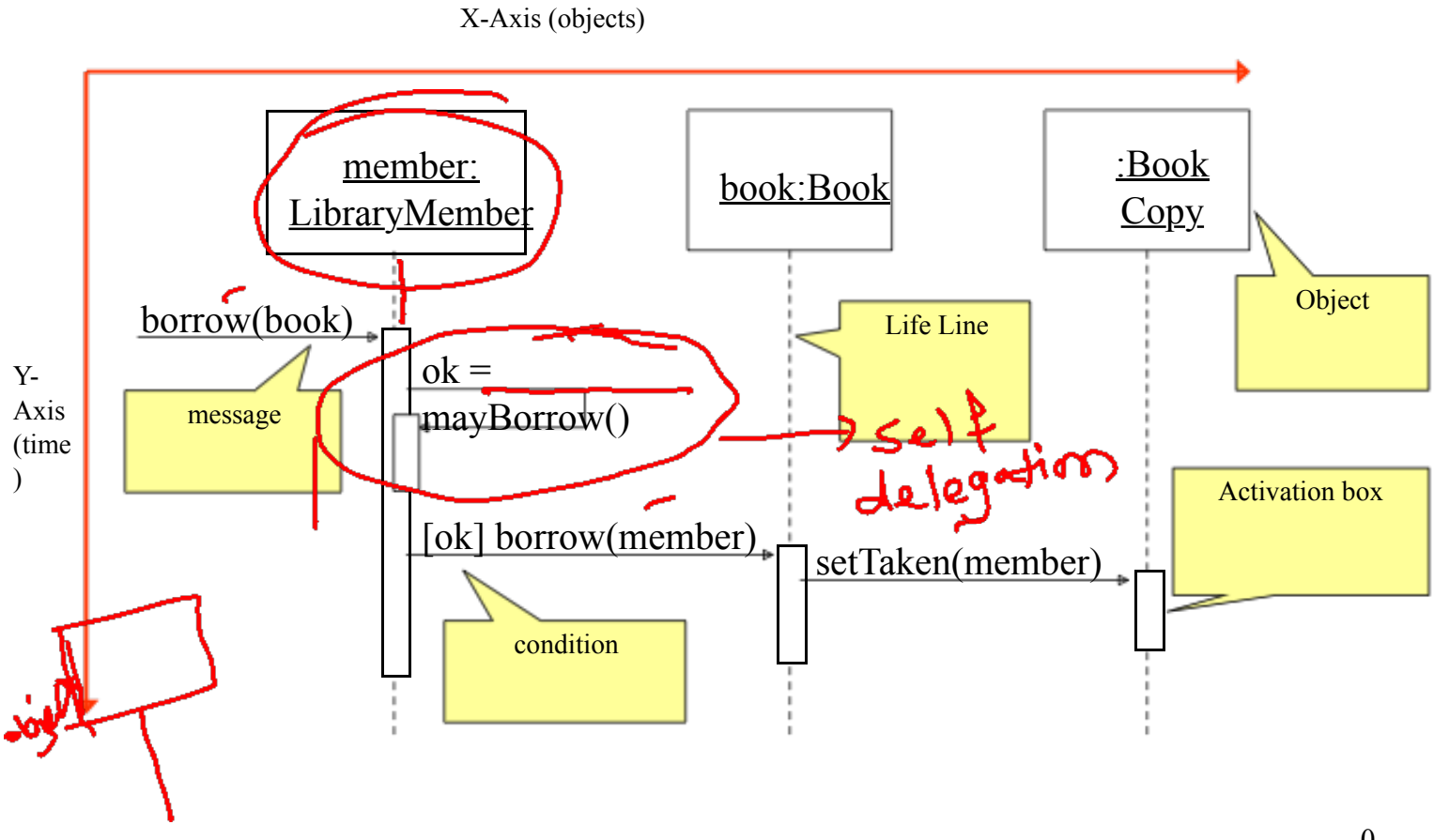
A Sequence Diagram



Sequence Diagram - Syntax



A Sequence Diagram



Object

- Object naming:
- syntax: *[instanceName][:className]*
- Name classes consistently with your class diagram (same classes).
- Include instance names when objects are referred to in messages or when several objects of the same type exist in the diagram.
- The *Life-Line* represents the object's life during the interaction

Messages

- An interaction between two objects is performed as a message sent from one object to another (simple operation call, Signaling, RPC)
- If object obj1 sends a message to another object obj2 some link must exist between those two objects (dependency, same objects)

Message types

- Synchronous messages
- Asynchronous messages

Messages (Cont.)

- A message is represented by an arrow between the lifelines of two objects.
- Self calls are also allowed
- 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.

Return Values

- Optionally indicated using a dashed arrow with a label indicating the return value.
- Don't model a return value when it is obvious what is being returned, e.g. getTotal()
- Model a return value only when you need to refer to it elsewhere, e.g. as a parameter passed in another message.
- Prefer modeling return values as part of a method invocation, e.g. `ok = isValid()`

Asynchronous messages

Asynchronous calls, which are associated with operations, typically have only a send message, but can also have a reply message.

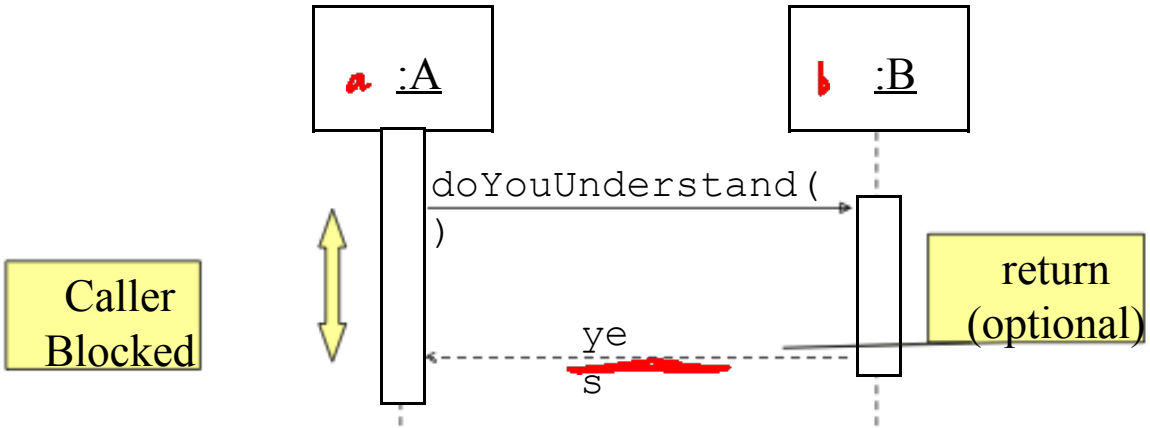
In contrast to a synchronous message, the source lifeline is not blocked from receiving or sending other messages.

You can also move the send and receive points individually to delay the time between the send and receive events.

You might choose to do this if a response is not time sensitive or order sensitive.

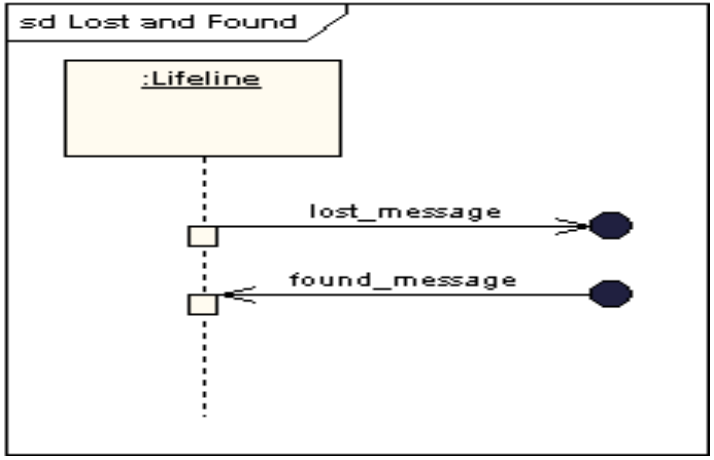
Synchronous Messages

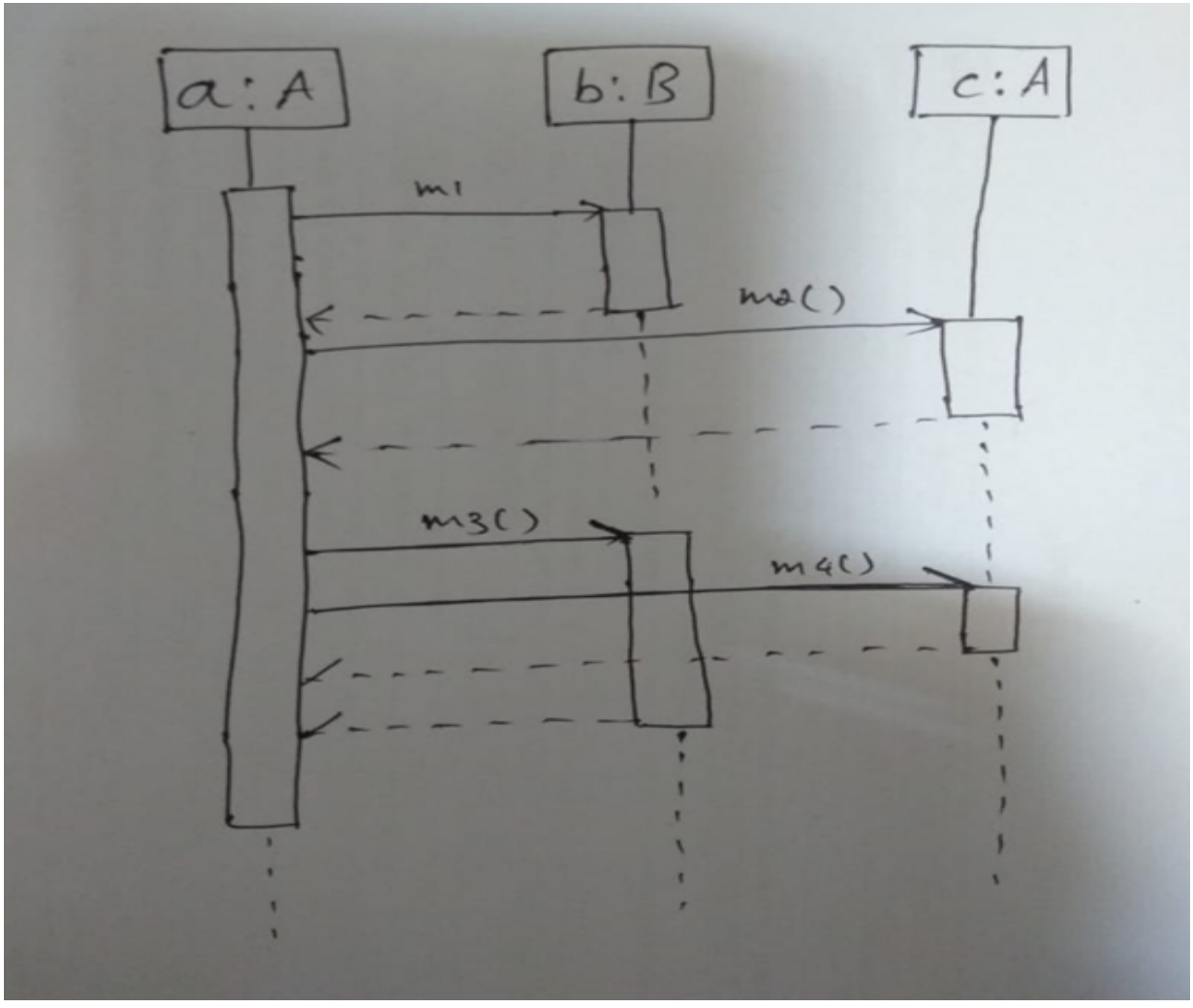
- Nested flow of control, typically implemented as an operation call.
- The routine that handles the message is completed before the caller resumes execution.



Lost and Found Messages

- Lost messages are those that are either sent but do not arrive at the intended recipient, or which go to a recipient not shown on the current diagram.
- Found messages are those that arrive from an unknown sender, or from a sender not shown on the current diagram. They are denoted going to or coming from an endpoint element.

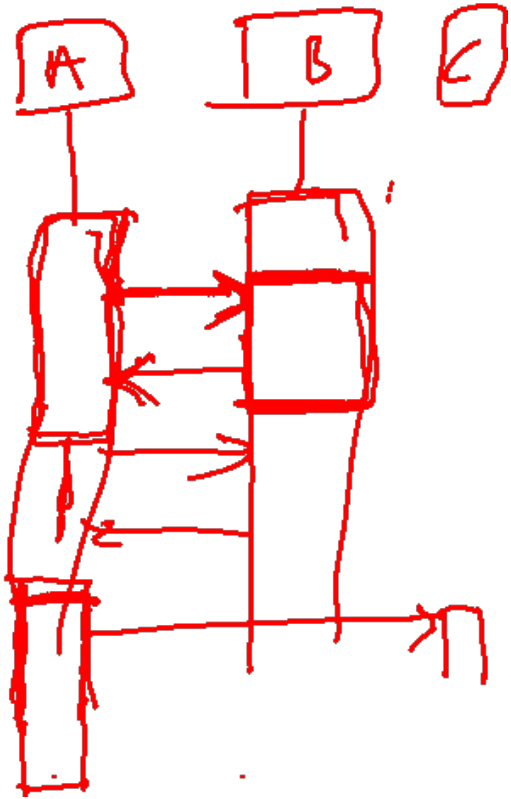



a:A
b:B
c:A
m1
m2()
m3()
m4()

Found & Lost Message

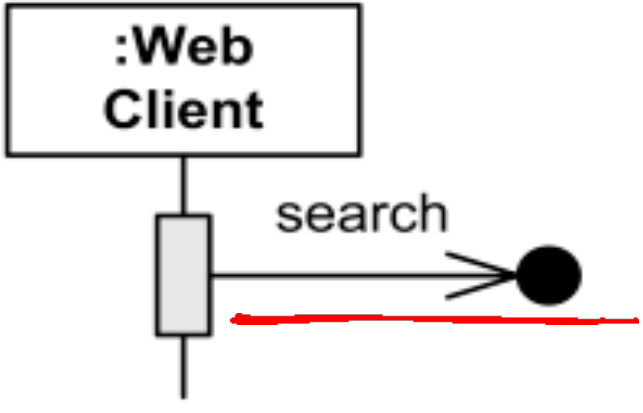
- **Found Message** is a message where the receiving event is known, but there is no (known) sending event

Online Bookshop gets search message of unknown origin.



Found & Lost Message

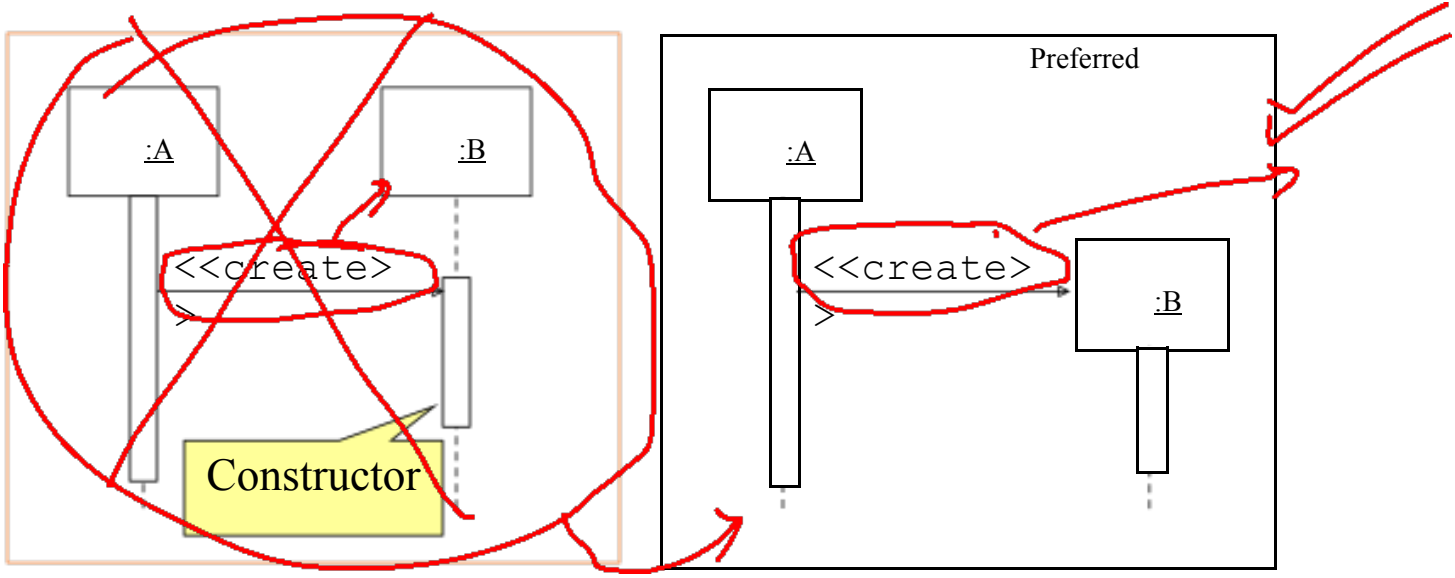
- **Lost Message** is a message where the sending event is known, but there is no receiving event



Web Client sent search message which was lost.

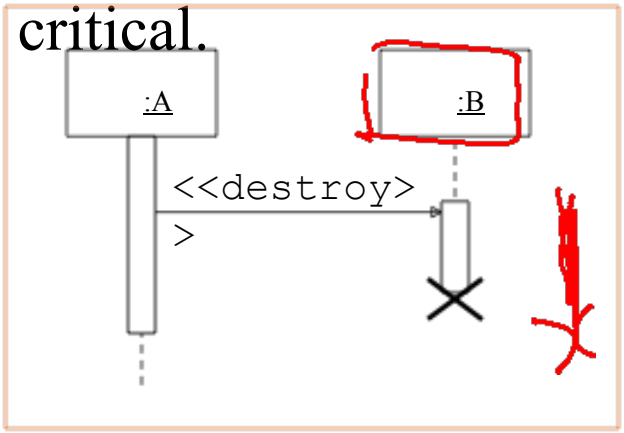
Object Creation

- An object may create another object via a `<<create>>` message.

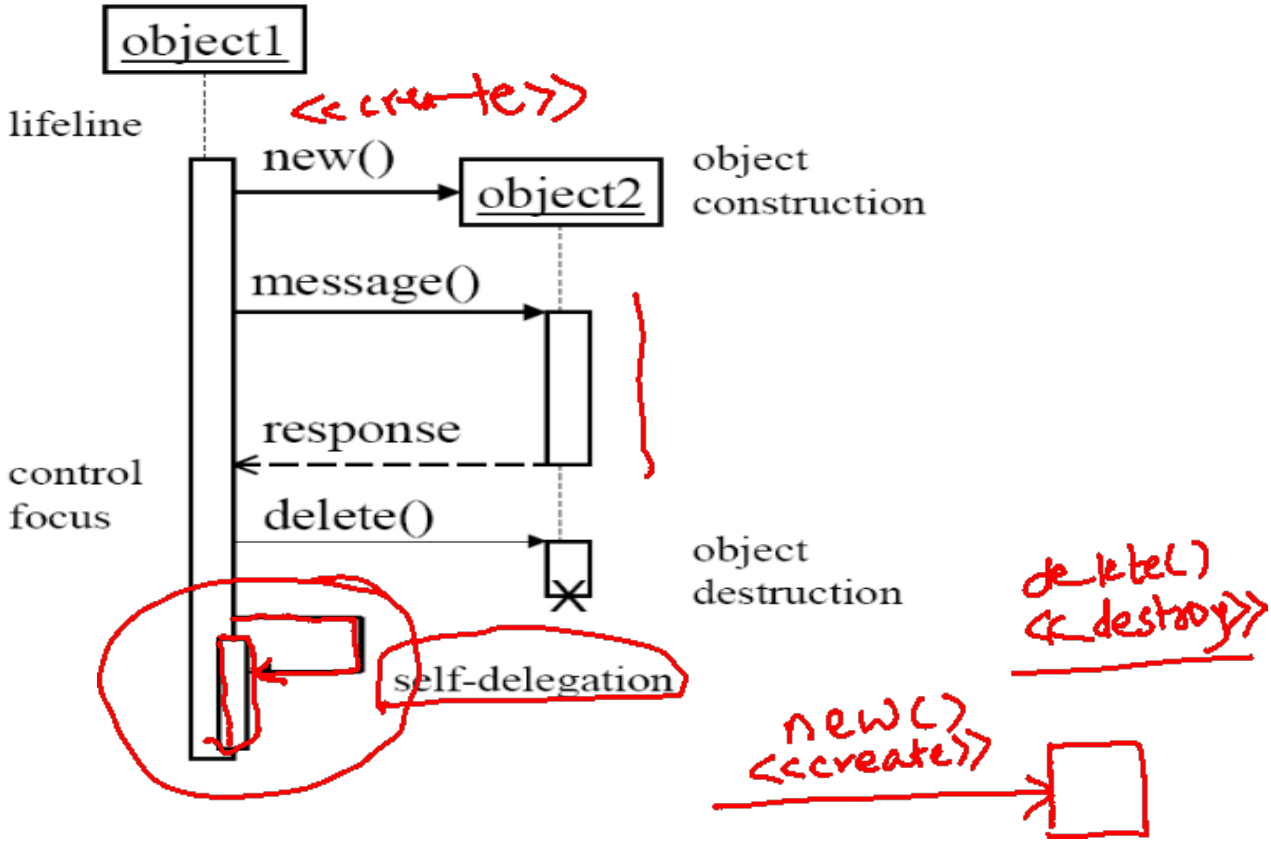


Object Destruction

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



Sequence Diagram - 2



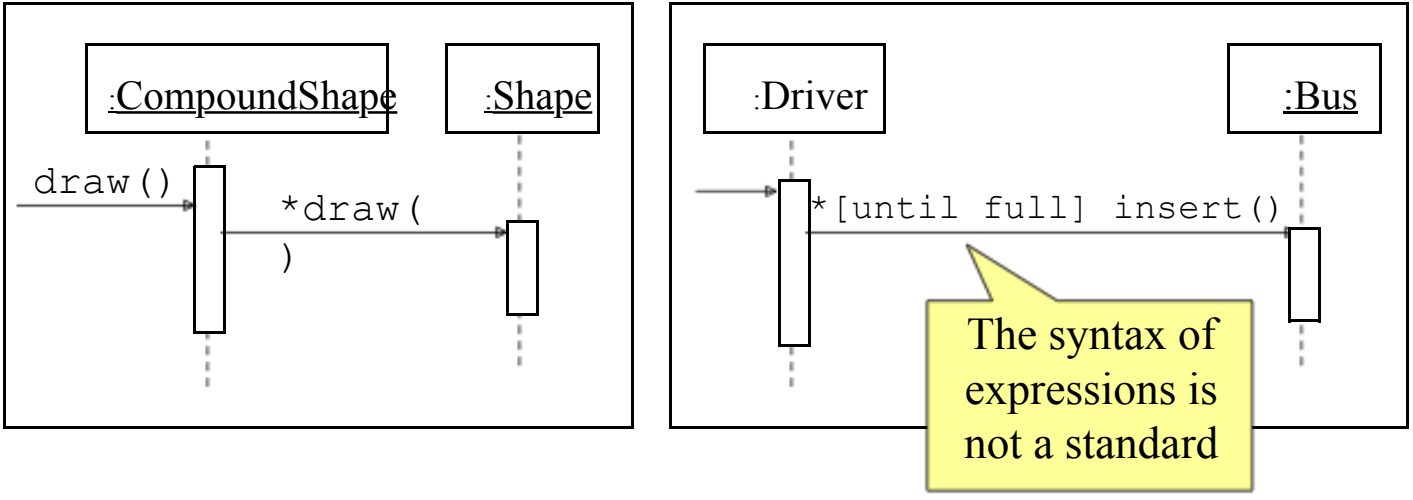
Control information

- Condition
- syntax: ‘[‘ expression ’]’ message-label
- The message is sent only if the condition is true
- example:
- Iteration
- syntax: * [‘[‘ expression ‘]’] message-label
- The message is sent many times to possibly multiple receiver objects.

[ok] borrow(member)

Control Information (Cont.)

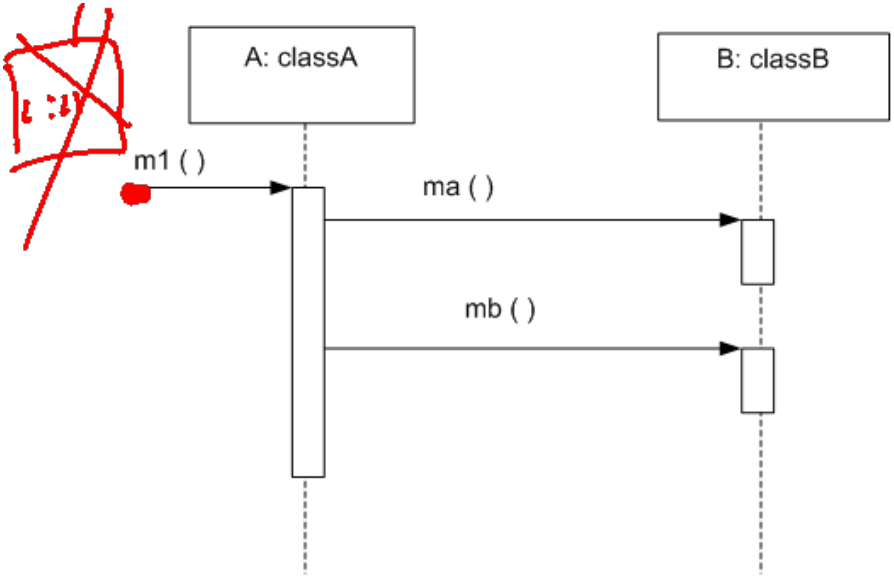
- **Iteration examples:**

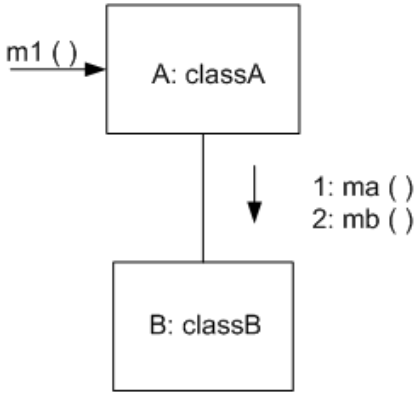


Control Information (Cont.)

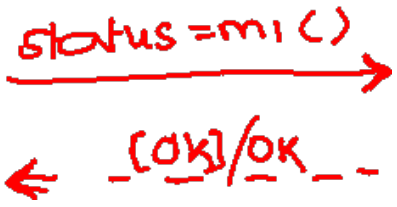
- The control mechanisms of sequence diagrams suffice only for modeling simple alternatives.
- Consider drawing several diagrams for modeling complex scenarios.
- Don't use sequence diagrams for detailed modeling of algorithms (this is better done using *activity diagrams, pseudo-code* or *state-charts*).

Sequence Diagram basic example

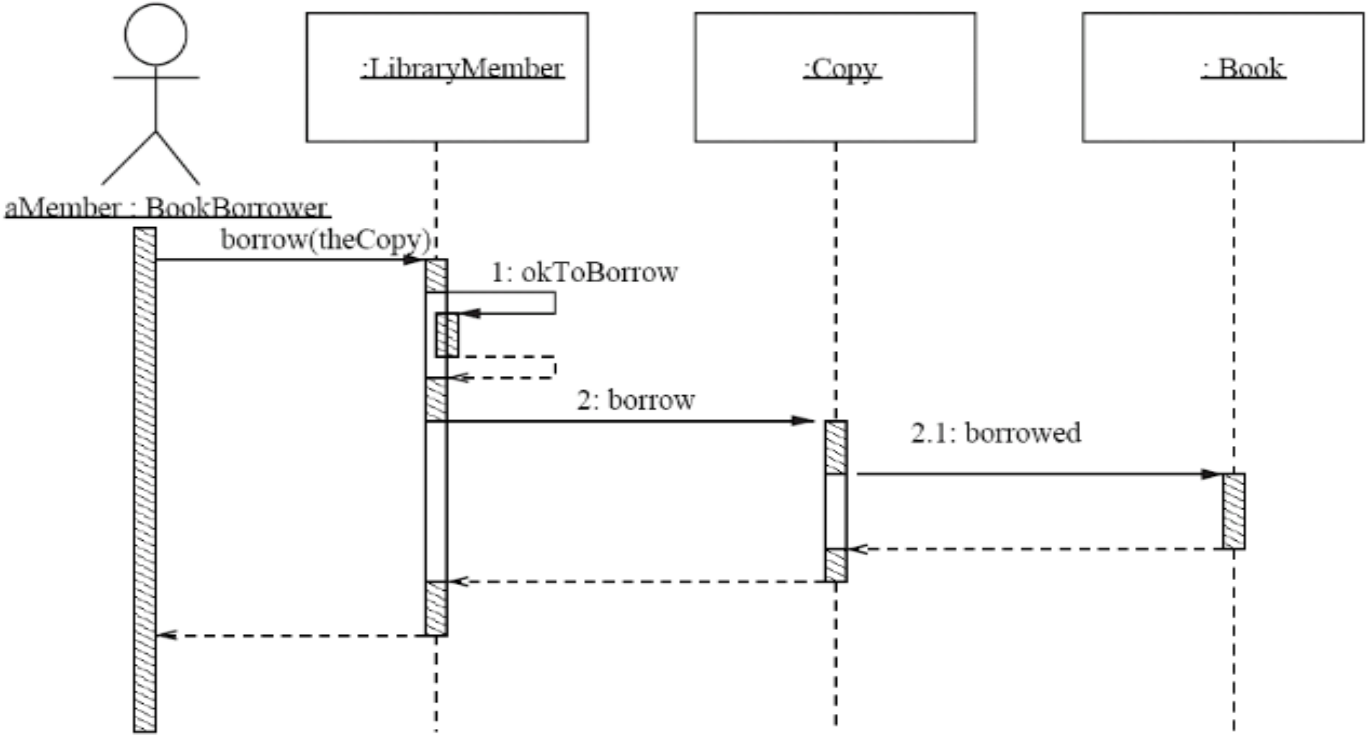




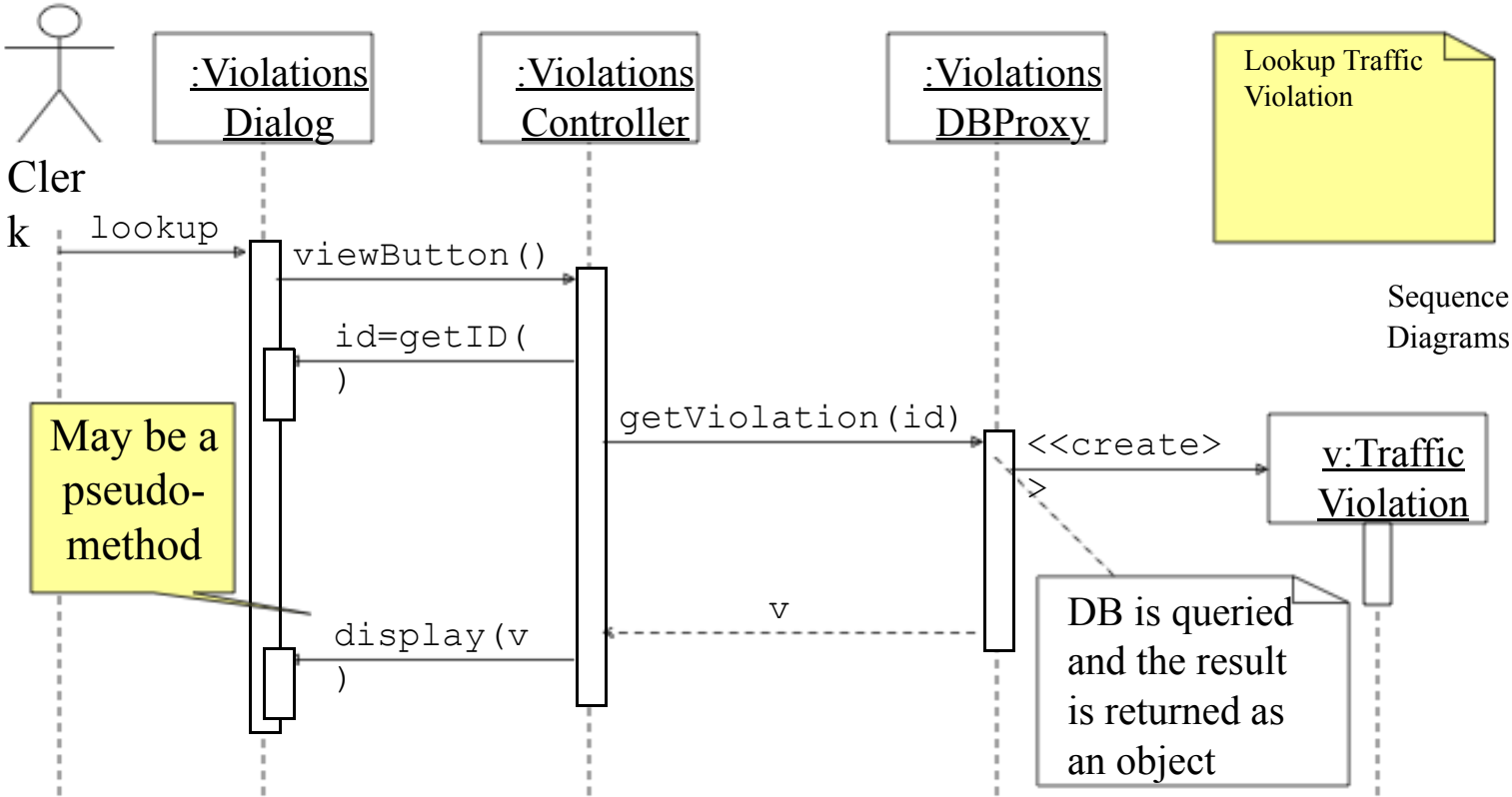

```
public class classA
private classB B = new ClassB();
public void m1(){
B. ma();
B.mb();
}
```



Nested activation

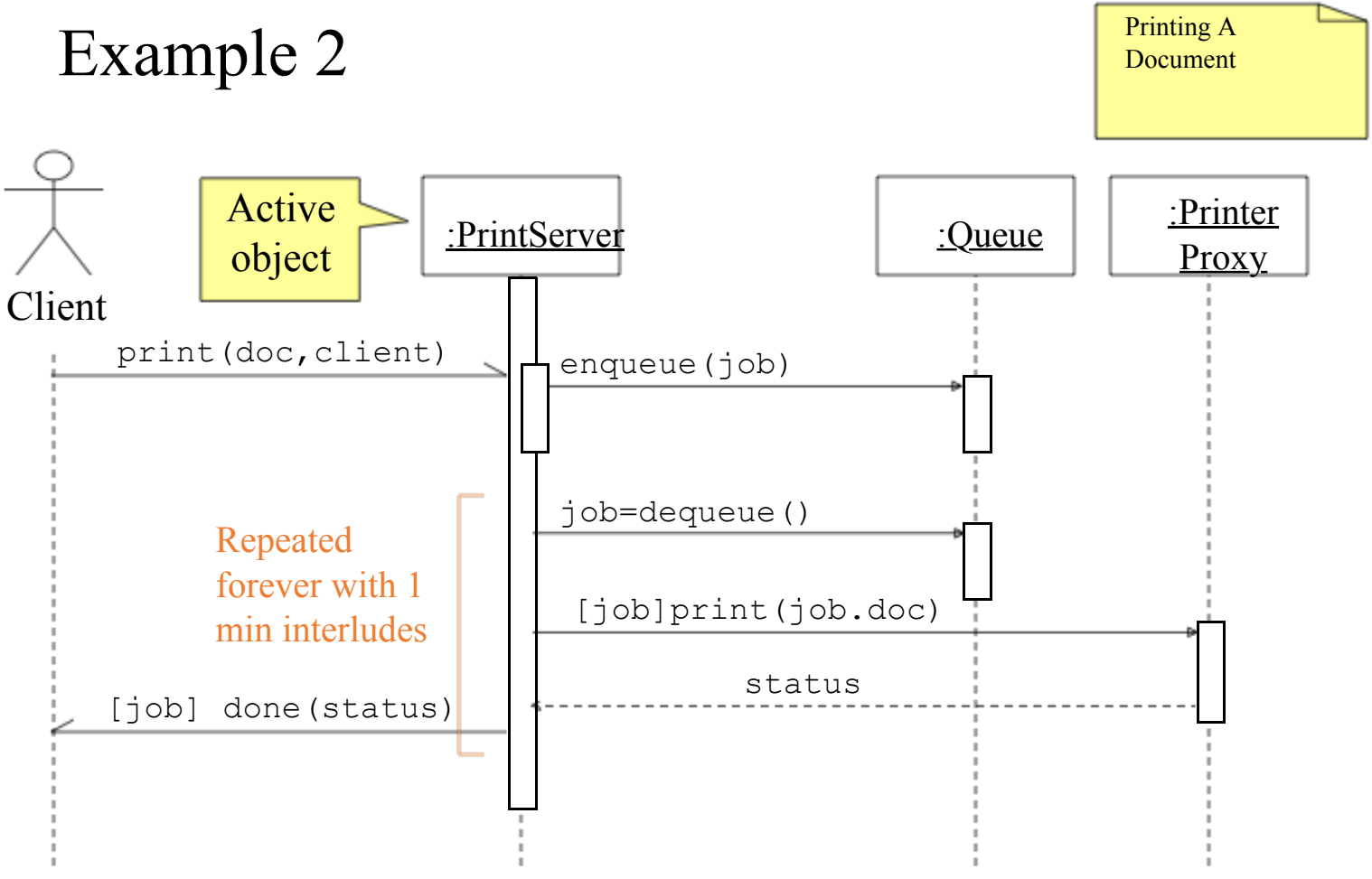


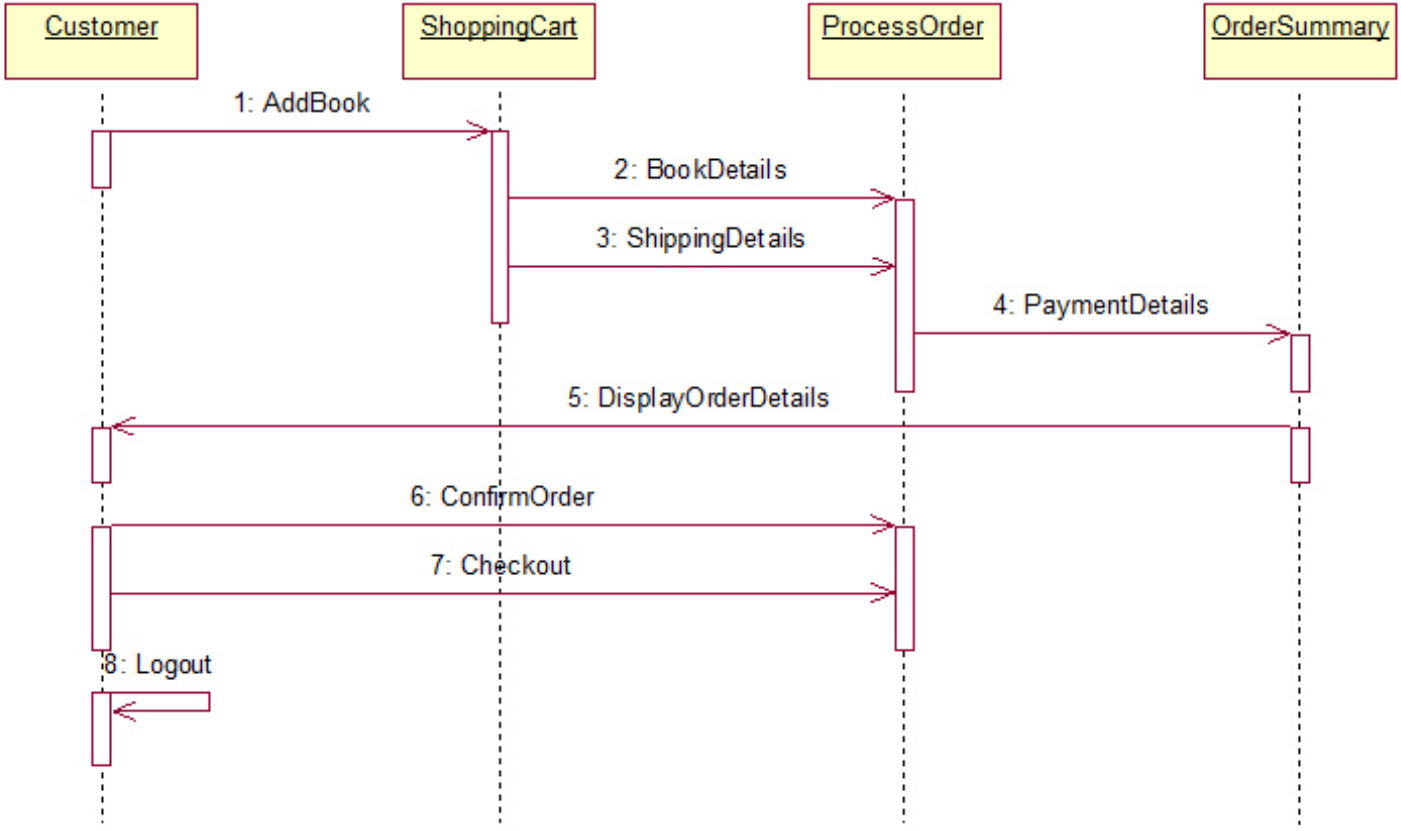
Example 1



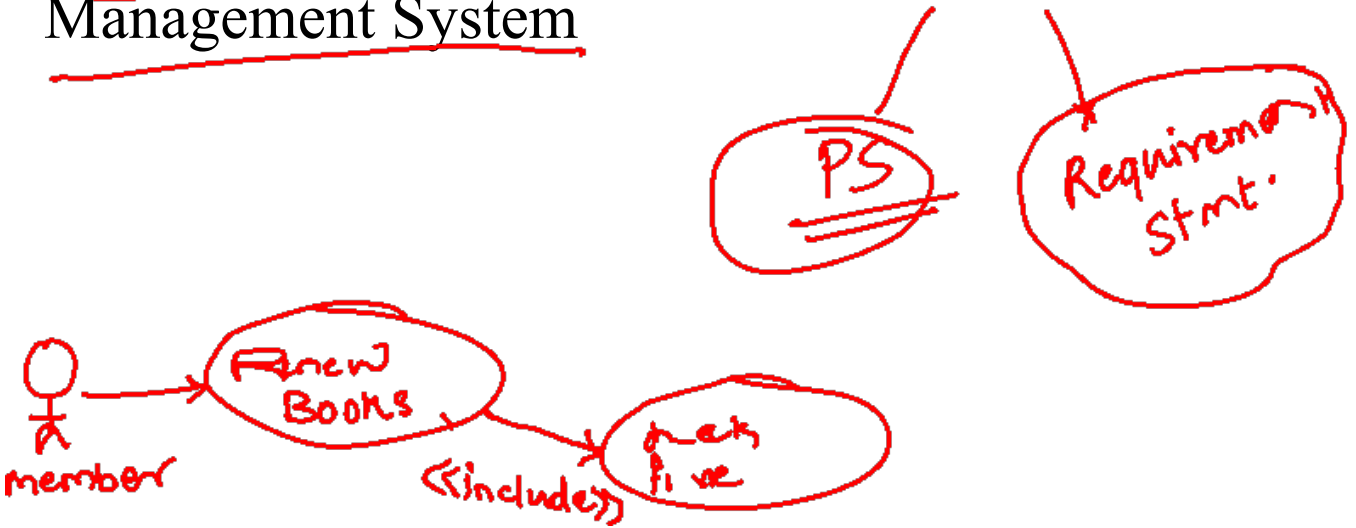
Sequence Diagrams

Example 2

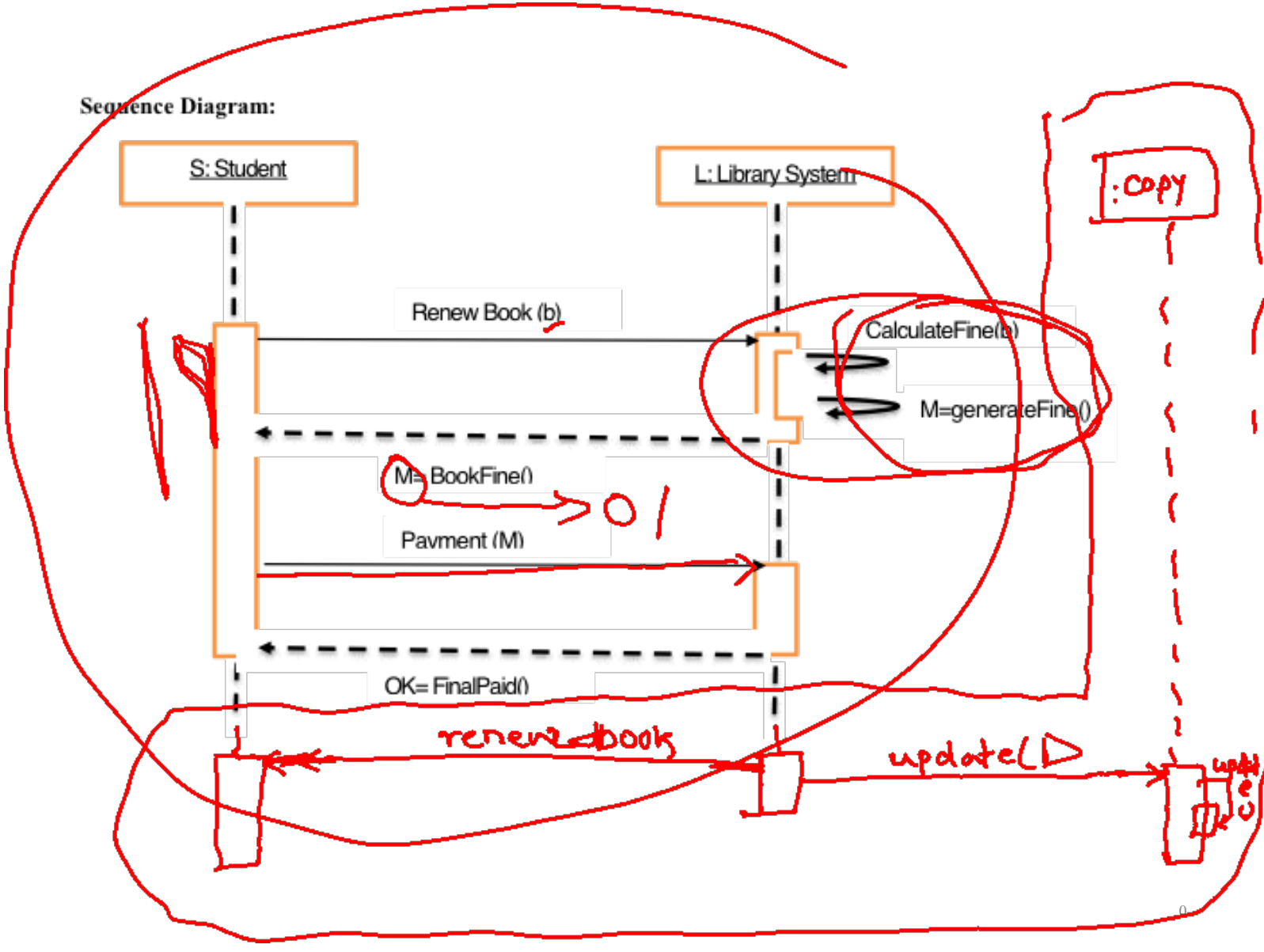

Printing A Document
Client
Active object
:PrintServer
:Queue
:Printer Proxy
print(doc,client)
enqueue(job)
job=dequeue()
Repeated forever with 1 min interludes
[job]print(job.doc)
status
[job] done(status)


Customer
ShoppingCart
ProcessOrder
OrderSummary
1: AddBook
2: BookDetails
3: ShippingDetails
4: PaymentDetails
5: DisplayOrderDetails
6: ConfirmOrder
7: Checkout
8: Logout

Draw sequence diagram for "Renew books with fine generation for late renewal" scenario in Library Management System



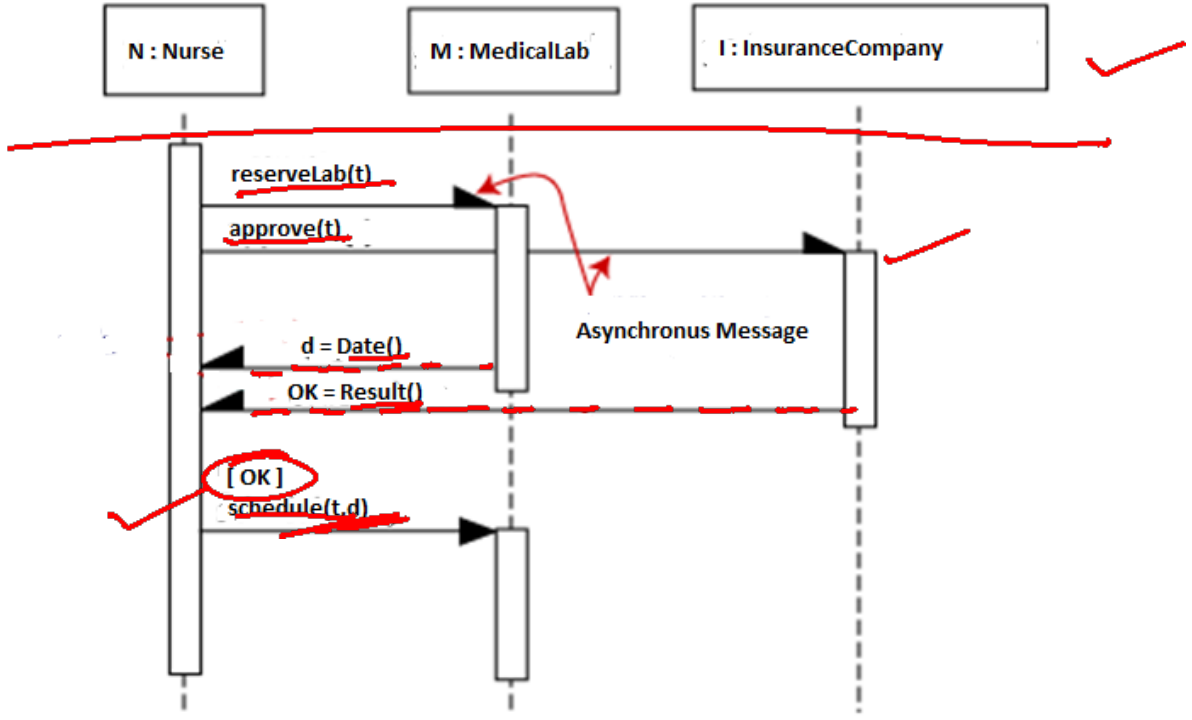
Sequence Diagram:



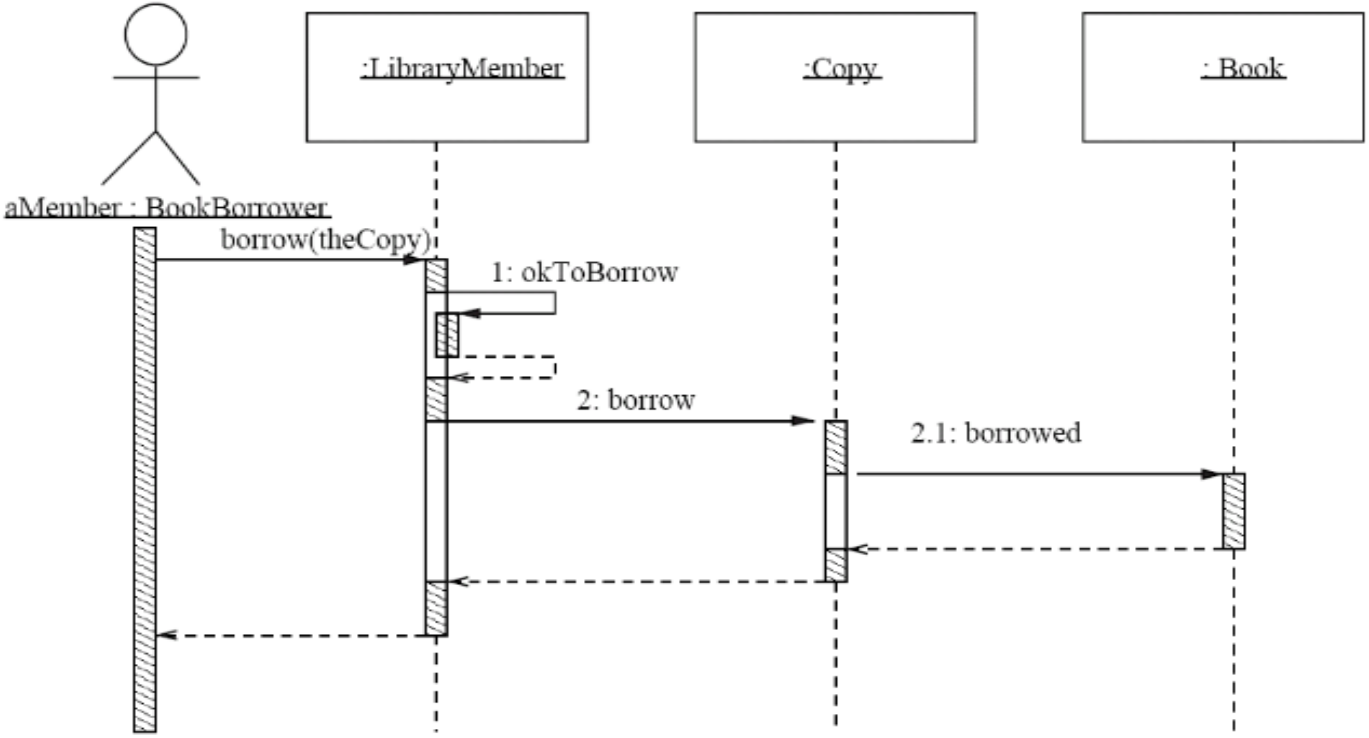
Example

In a hospital, nurse simultaneously requests the medical lab to reserve a date for the patient's diagnostic test (t) and the insurance company to approve the test. If the Insurance Company approves the test, then the Nurse will schedule the test on the date supplied by the Medical lab.

Asynchronous messages



Nested activation



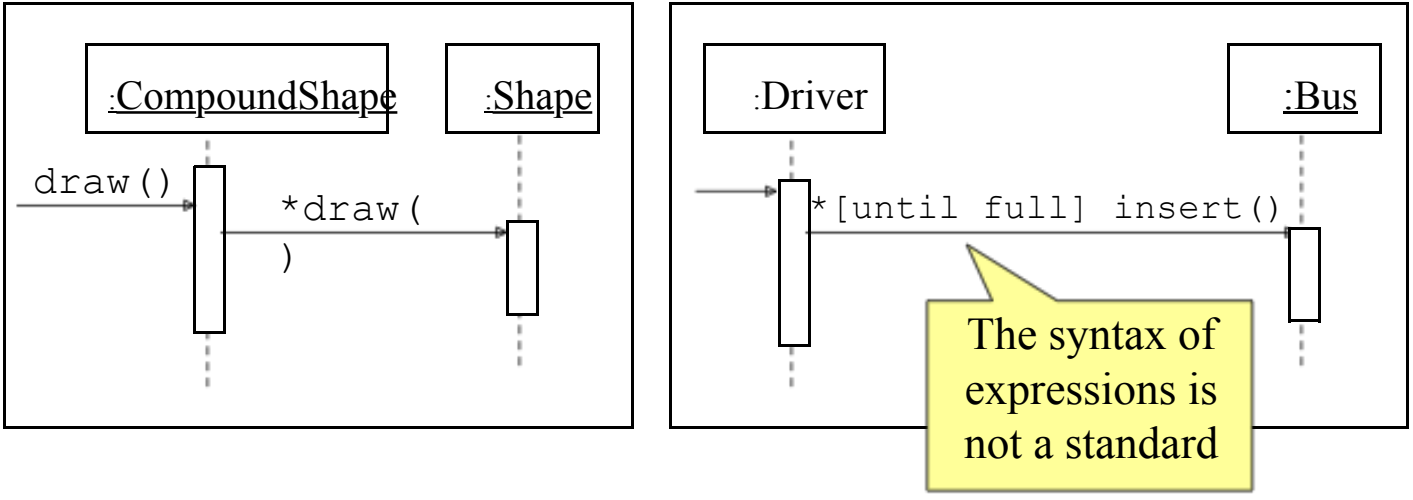
Control information

- Condition
- syntax: ‘[‘ expression ’]’ message-label
- The message is sent only if the condition is true
- example:
- Iteration
- syntax: * [‘[‘ expression ‘]’] message-label
- The message is sent many times to possibly multiple receiver objects.

[ok] borrow(member)

Control Information (Cont.)

- **Iteration examples:**



Control Information (Cont.)

- The control mechanisms of sequence diagrams suffice only for modeling simple alternatives.
- Consider drawing several diagrams for modeling complex scenarios.
- Don't use sequence diagrams for detailed modeling of algorithms (this is better done using *activity diagrams, pseudo-code* or *state-charts*).

Interaction Fragment in Sequence Diagram

- An Interaction fragment which defines an expression of interaction fragments.

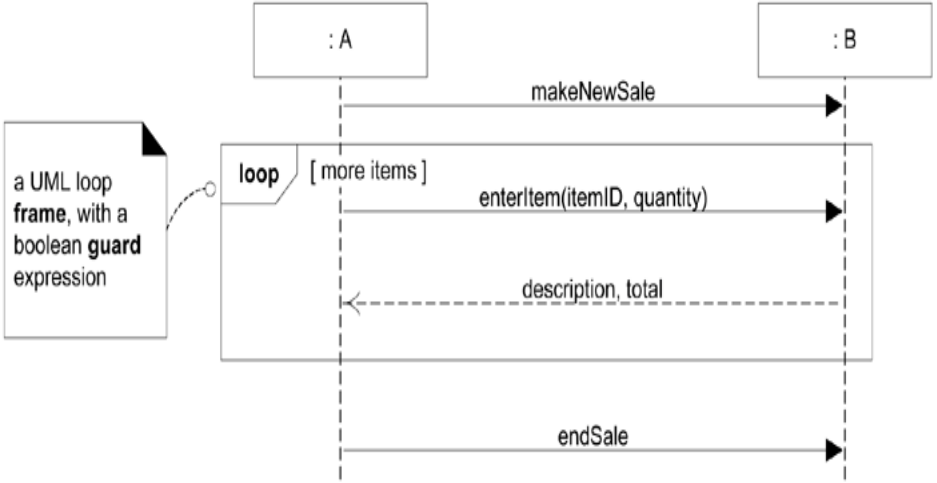
- An interaction fragment is defined by an interaction operator and corresponding interaction operands. Through the use of interaction fragments the user will be able to describe a number of traces in a compact and concise manner.

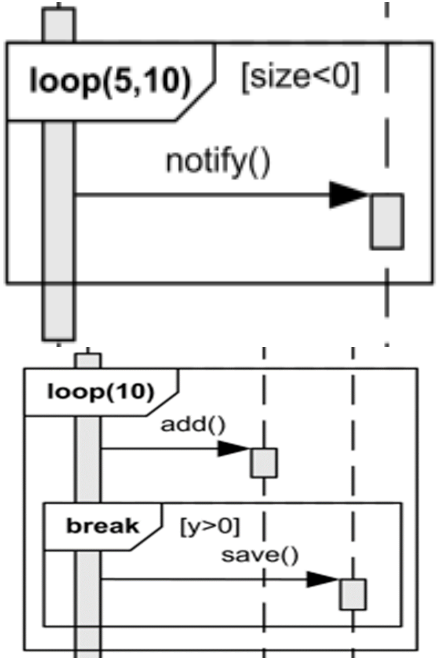
- interaction fragment may have constraints also called guards in UML

- Typical frame operators are:

Frame Operator	Semantics
alt	Alternative fragment for mutual exclusion conditional logic expressed in the guards.
loop	Loop fragment while guard is true. Can also write loop(n) to indicate looping n times.
opt	Optional fragment that executes if guard is true.
par	Parallel fragments that execute in parallel.
region	Critical region within which only one thread can run.

- Frames in UML Sequence Diagrams :
- To allow the visualization of complex algorithms, sequence diagrams support the notion of frames;
- Frames are regions of the diagrams that have an operator and a guard,. Figure.





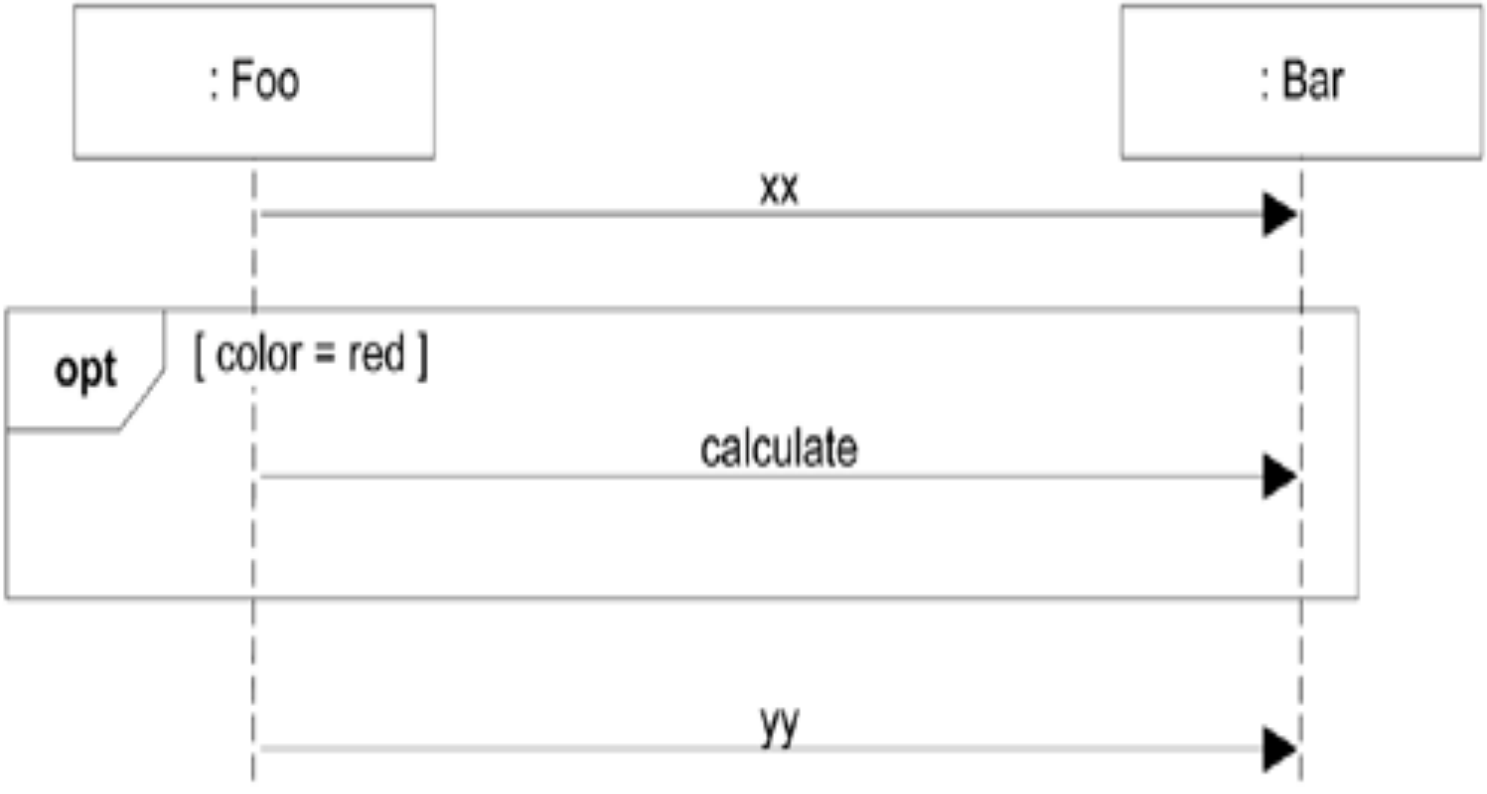


Figure Using the opt Frame

- Next Figure illustrate the use of the alt frame for mutually exclusive alternatives

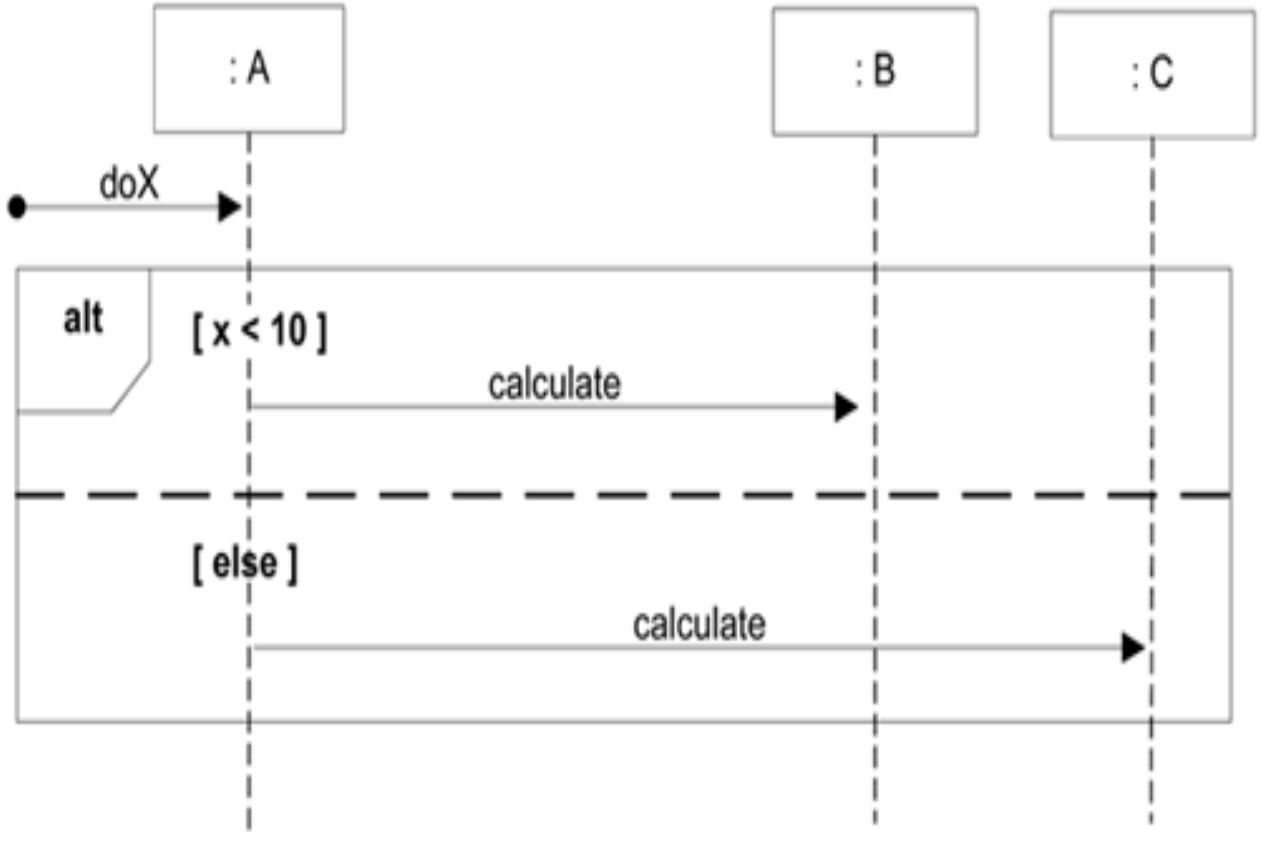
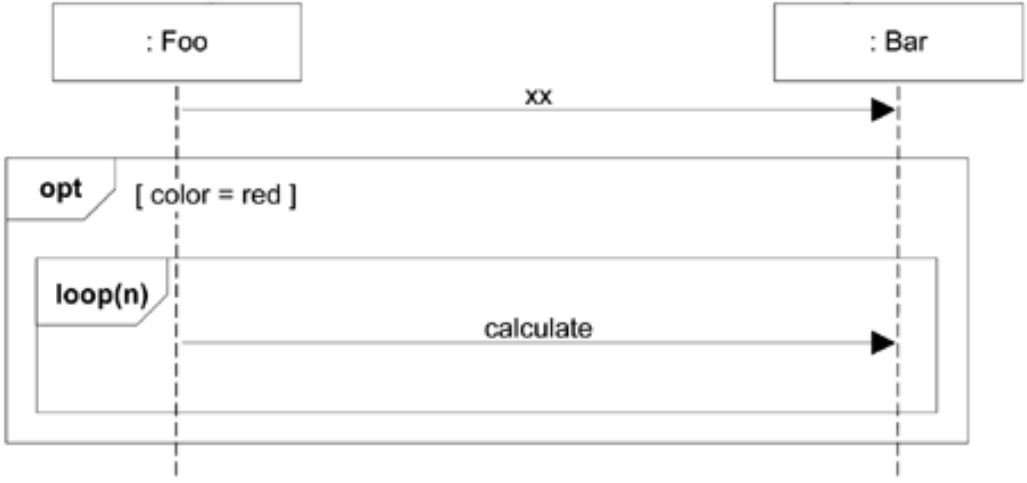
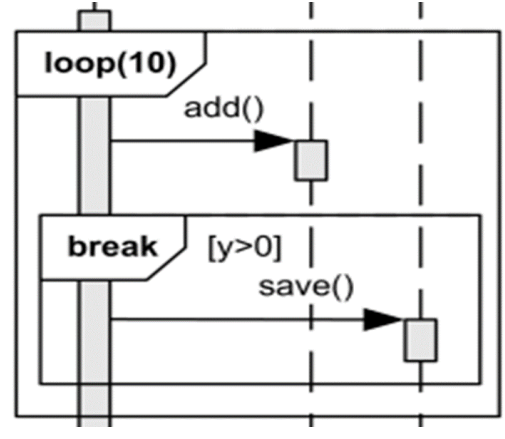


Figure Using the alt Frame

- Nesting of Frames :
- Frames can be nested:





- MyManipal is an online social networking service only for MIT students and staff. The users must register before using the site. Once the user is authenticated he can retrieve notification, updates and messages. Further, in the next step the user can change the user preferences as online/ hidden. Finally, the status will be displayed in users' wall. Unsuccessful login will lock the account for 1 minute to prevent from brute force attack.

- The user wants to boot a server with Linux operating system (OS). So whenever the user sends a signal *start* the operating system will load RAM and the booting process will be initiated. After a time interval, the OS will get its current date and time. Moreover, the user will be acknowledged with current date *d* and time *t* by the OS within a stipulated time period. Based on the user request the OS will spawn two processes simultaneously namely Pi and Pj. The order of creation of above two processes is random. Each process will return its *pid* to OS. Due to the scarcity of resources the OS is required to kill any one of the processes. The built-in functionality in OS called *Preemptive Algorithm* chooses Pi as victim process. OS sends a signal *kill* to kill the process Pi. If the OS is unsuccessful in killing the process Pi, then the OS will send *kill* message to kill the process Pj.