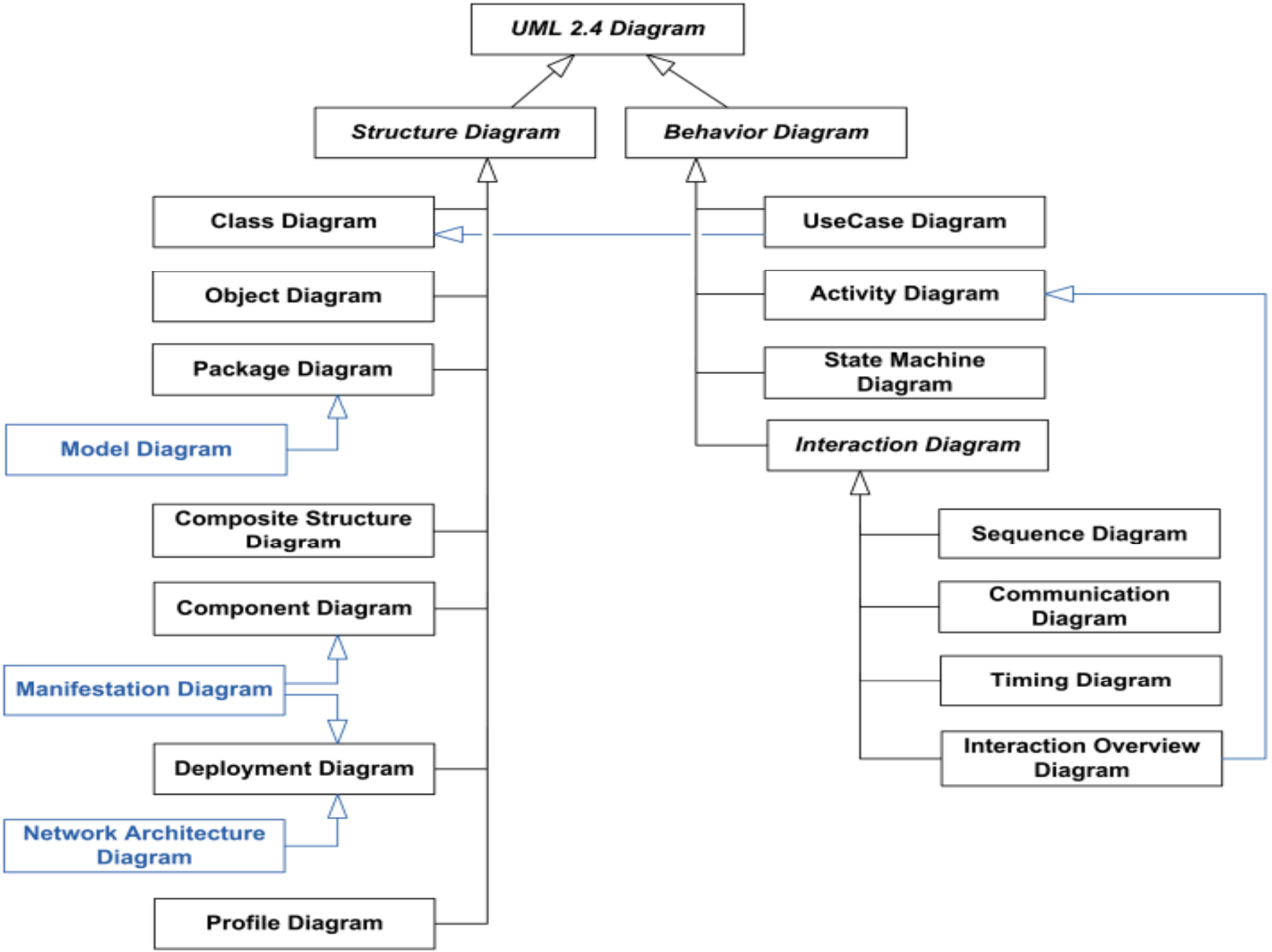
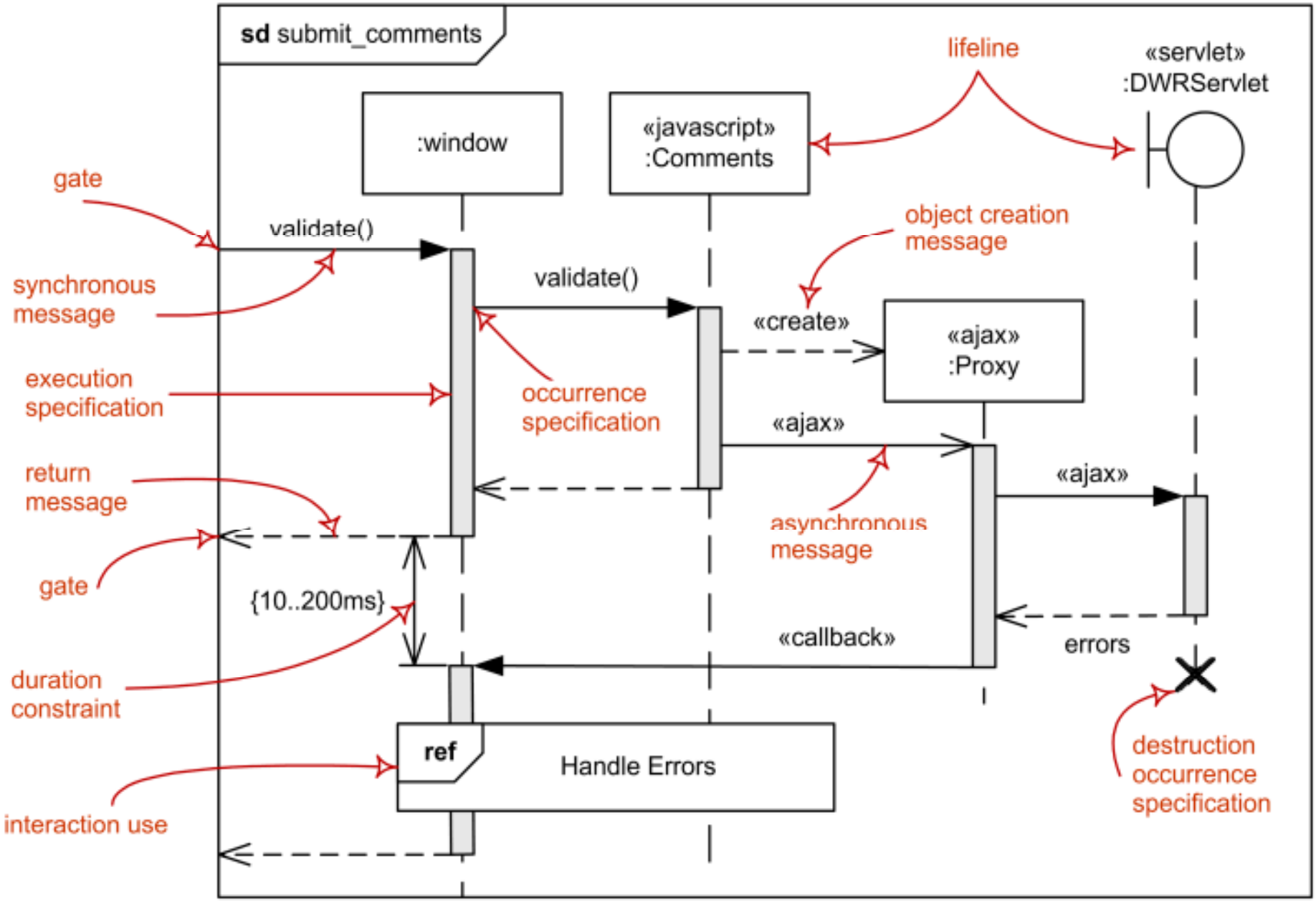
UML Sequence Diagrams

Deepali D. Londhe


UML 2.4 Diagram
Structure Diagram
Behavior Diagram
Class Diagram
Object Diagram
Package Diagram
Model Diagram
Composite Structure Diagram
Component Diagram
Manifestation Diagram
Deployment Diagram
Network Architecture Diagram
Profile Diagram
UseCase Diagram
Activity Diagram
State Machine Diagram
Interaction Diagram
Sequence Diagram
Communication Diagram
Timing Diagram
Interaction Overview Diagram

UML Diagrams

- Structure diagram
  - Class diagram
  - Object diagram
  - Component diagram
  - Deployment diagram
  - Package diagram
- Behaviour diagram
  - Use Case diagram
  - Activity diagram
  - **Interaction diagram**
    - **Sequence diagram**
    - Communication diagram


sd submit_comments
lifeline
«servlet»
:DWRServlet
:window
«javascript»
:Comments
gate
validate()
object creation
message
synchronous
message
validate()
«create»
«ajax»
:Proxy
execution
specification
occurrence
specification
«ajax»
return
message
«ajax»
asynchronous
message
gate
{10..200ms}
«callback»
errors
duration
constraint
ref
Handle Errors
destruction
occurrence
specification
interaction use

In a nutshell: what are interactions?

- A collection of communications between instances, including all ways to affect instances, like operation invocation, as well as creation and destruction of instances

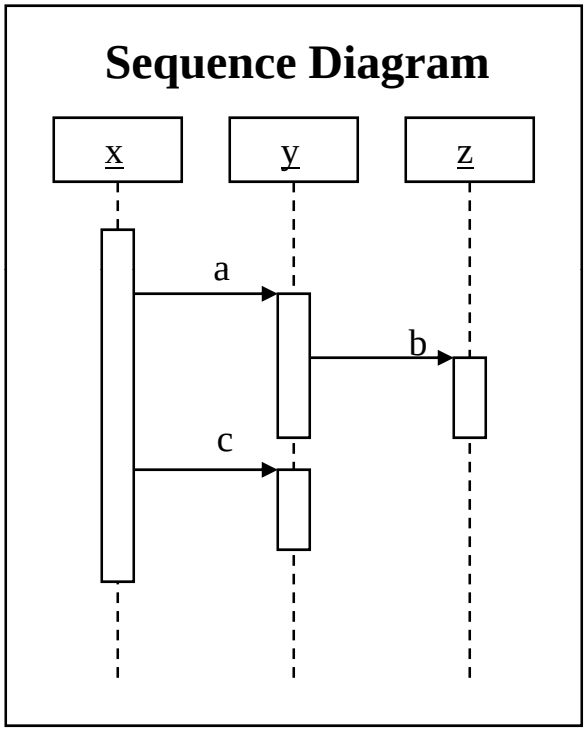
- The communications are partially ordered (in time)

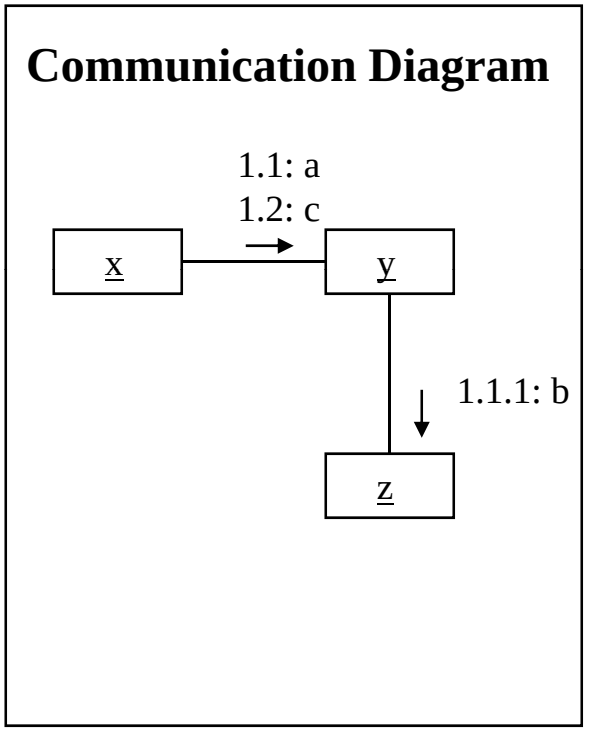
Interaction Diagram Guide

- Show interactions between instances in the model
  - graph of instances (possibly including links) and stimuli
  - existing instances
  - creation and deletion of instances

- Kinds
  - Sequence diagram (temporal focus)
  - Communication diagram (structural focus)

Interaction Diagrams





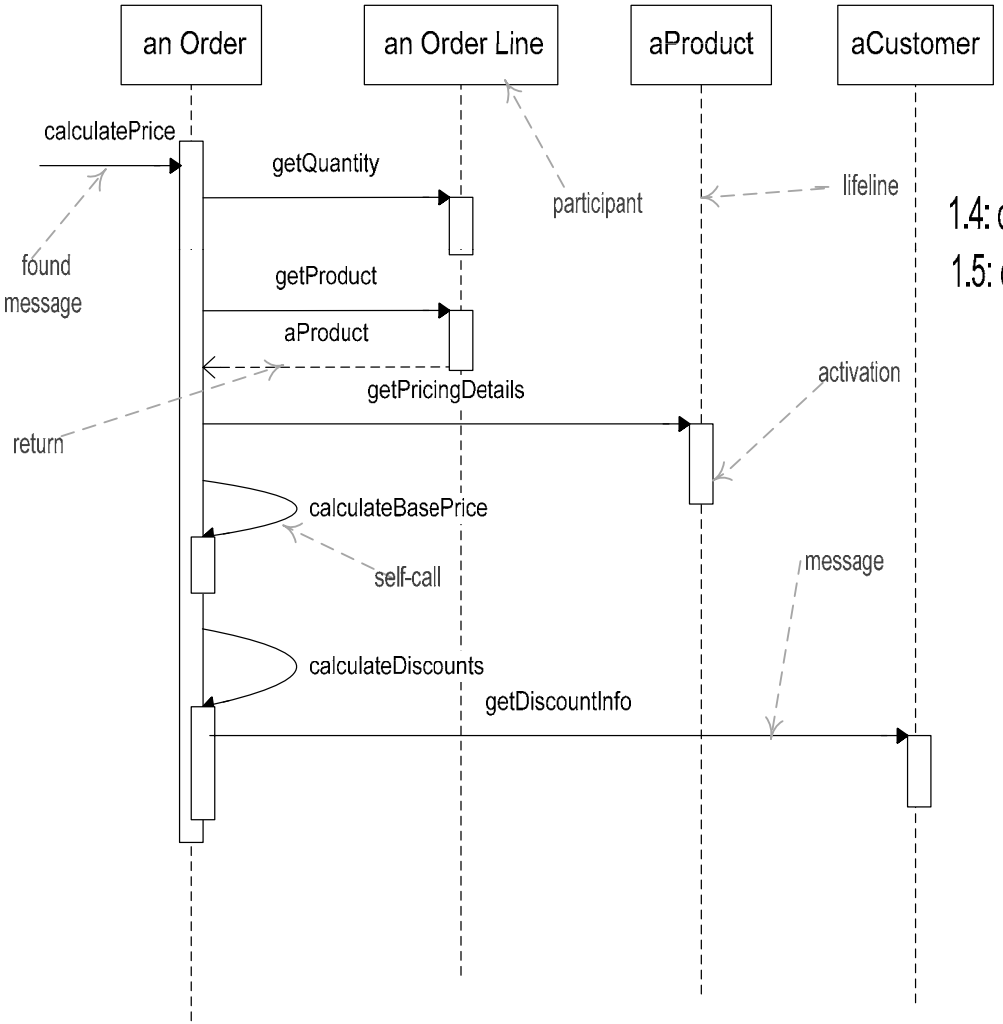
Sequence Diagram

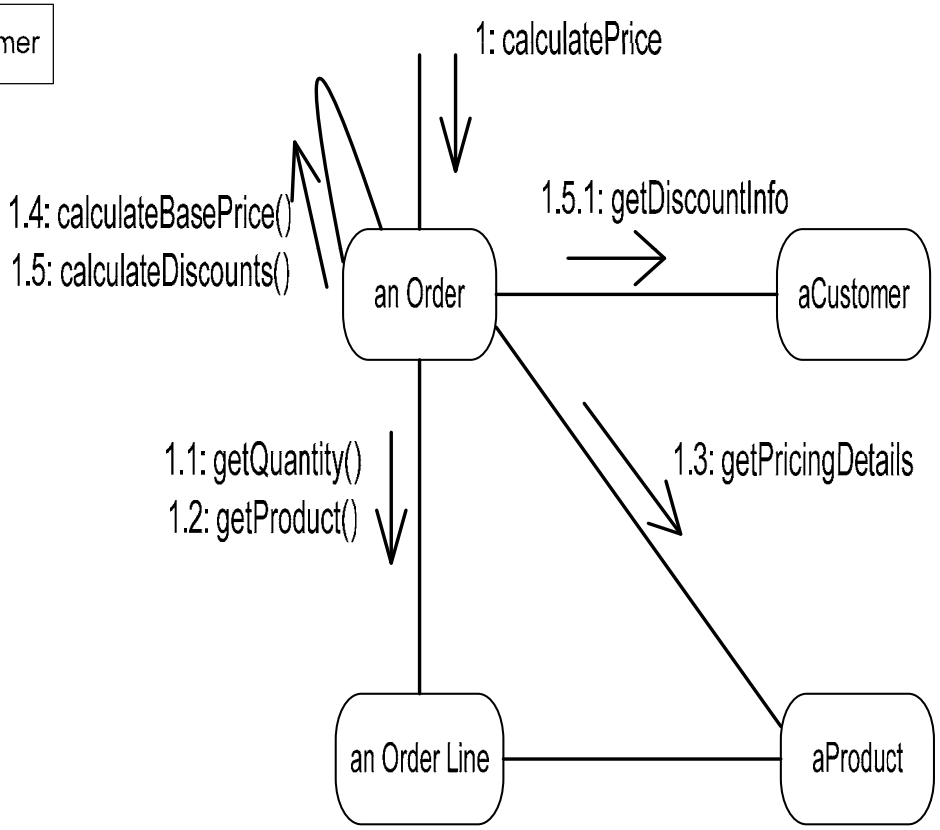
- **Sequence diagram** is the most common kind of interaction diagram, which focuses on the **message interchange between a number of lifelines**.
- Sequence diagram describes an interaction by focusing on the **sequence of messages that are exchanged, along with their corresponding occurrence specifications on the lifelines**.

Lifeline & Messages

- The vertical line is called the object's **lifeline**. The lifeline represents the object's life during the interaction.
- Each message is represented by an arrow between the lifelines of two objects.
- The order in which these messages occur is shown top to bottom.
- Each message is labeled at minimum with the message name; also arguments and some control information can be included.
- A **self-back** is a message that an object sends to itself, by sending the message arrow back to the same lifeline.

Sequence Diagram / Communication Diagram



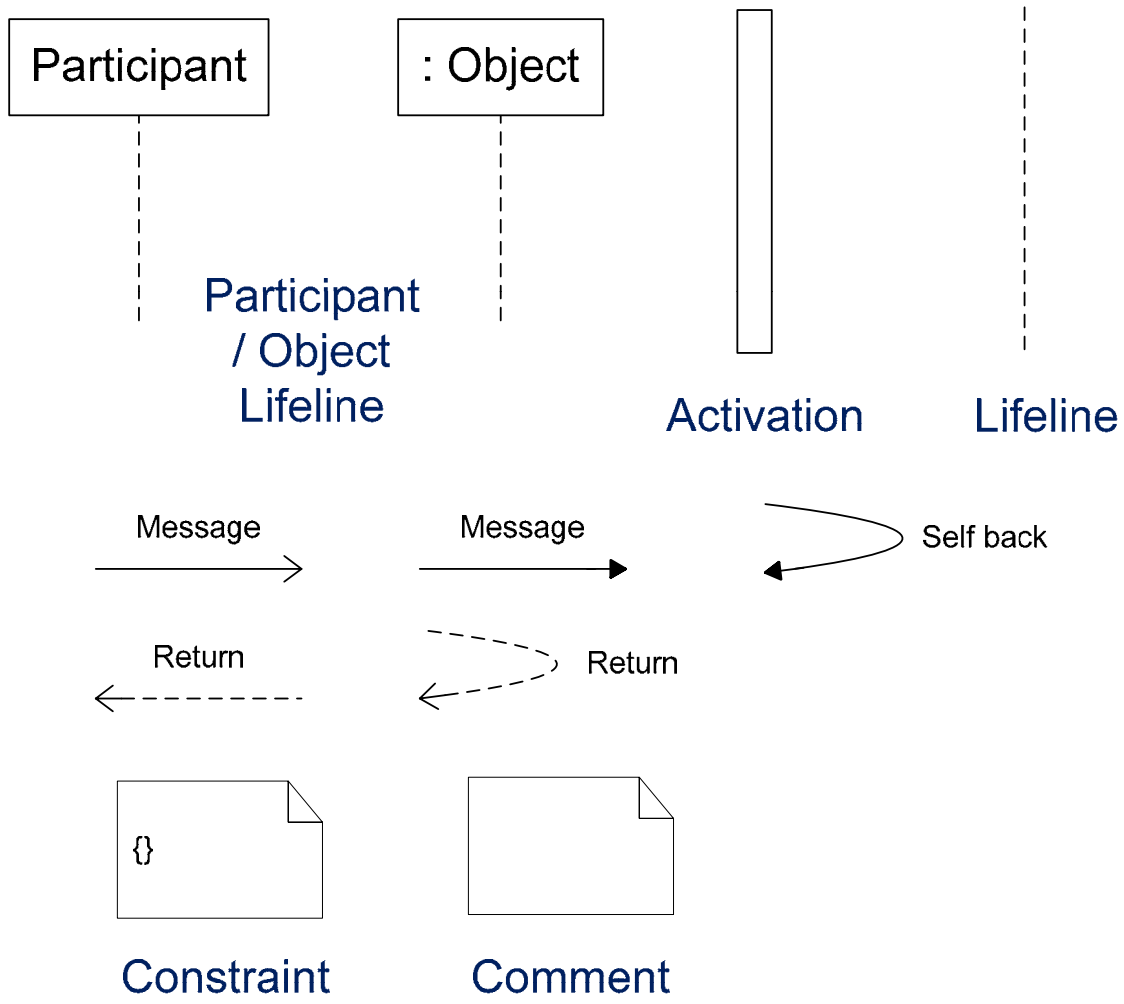


Sequence Diagram vs. Communication Diagram

- Sequence diagrams are used when you want to look at the behavior of several objects within a single use case. Sequence diagrams are good at showing collaborations among the objects.

- Communication diagrams emphasize the data links between the various participants in the interaction. With communication diagrams we can show how the participants are linked together.

Elements Sequence Diagrams



Message

- **Message** is a named element that defines one specific kind of communication between lifelines of an interaction. The message specifies not only the kind of communication, but also the sender and the receiver.
- Syntax for the message is:
  - message ::= [attribute '='] signal-or-operation-name [arguments] [':' return-value] | '*'
  - arguments ::= '(' [argument [',' argument]* ')‘
  - argument ::= [parameter-name '='] argument-value | attribute '=' out-parameter-name [':' argument-value] | ' -'
- Arguments of a message could only be:
  - attributes of the sending lifeline,
  - constants,
  - symbolic values (which are wildcard values representing any legal value),
  - explicit parameters of the enclosing interaction,
  - attributes of the class owning the interaction.

Different Kinds of Arrows

Procedure call or other kind of nested flow of control (synchronous message)

Return

Asynchronous message

Asynchronous message in UML 1.x

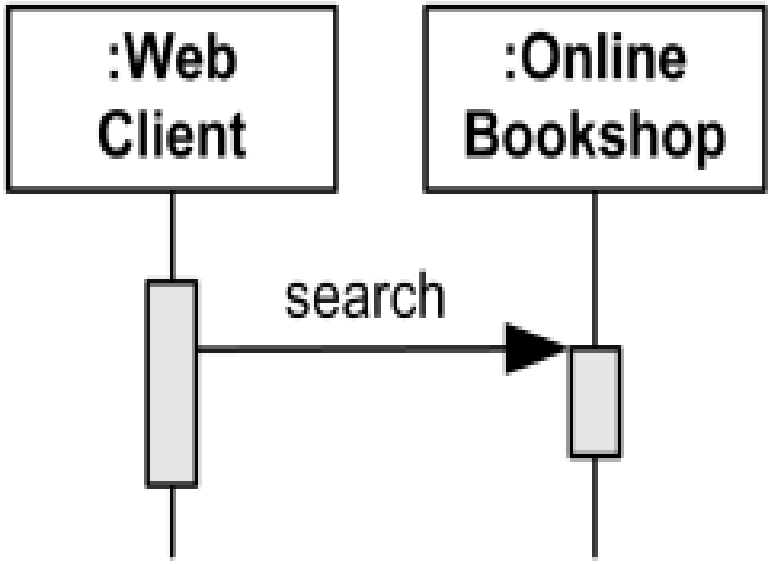
Messages by Action Type

- A message reflects either an operation call and start of execution or a sending and reception of a signal.
- When a message represents an **operation** call, the arguments of the message are the arguments of the operation. When a message represents a **signal**, the arguments of the message are the attributes of the signal.

- Depending on the type of action that was used to generate the message, message could be one of:
- synchronous call
- asynchronous call
- asynchronous signal
- create
- delete
- reply

Synchronous Call

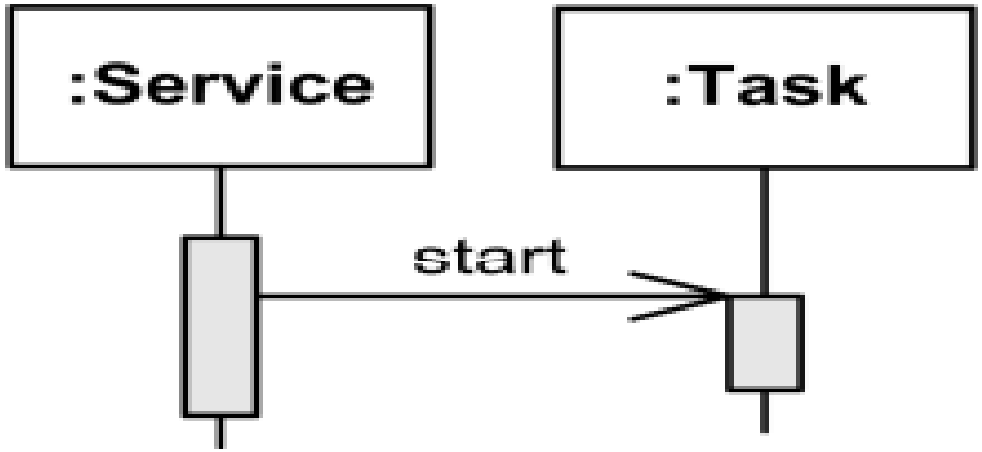
- **Synchronous call** typically represents operation call - send message and suspend execution while waiting for response. Synchronous call messages are shown with filled arrow head.



Web Client searches Online Bookshop and waits for results.

Asynchronous Call

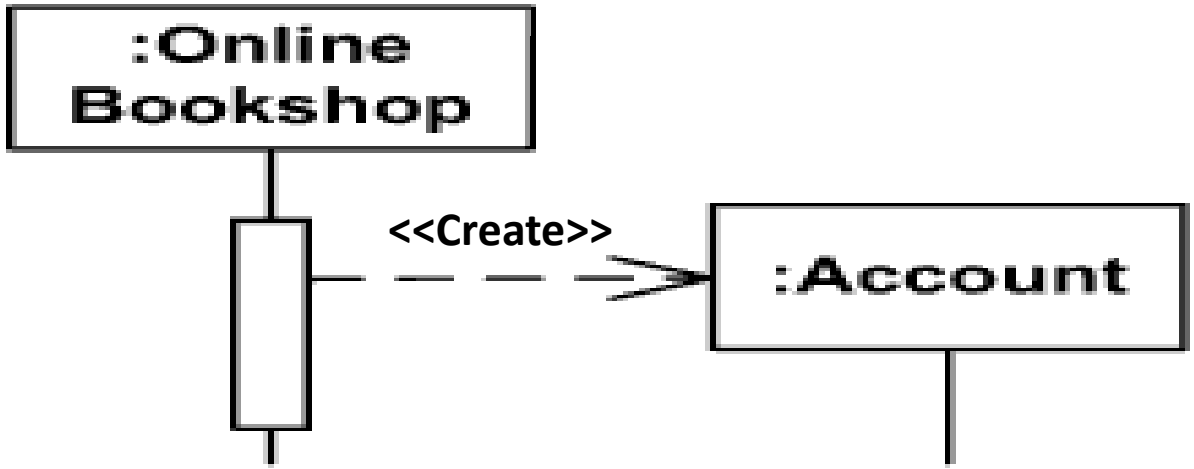
- **Asynchronous call** - send message and proceed immediately without waiting for return value. Asynchronous messages have an open arrow head.



Service starts Task and proceeds in parallel without waiting.

Create Message

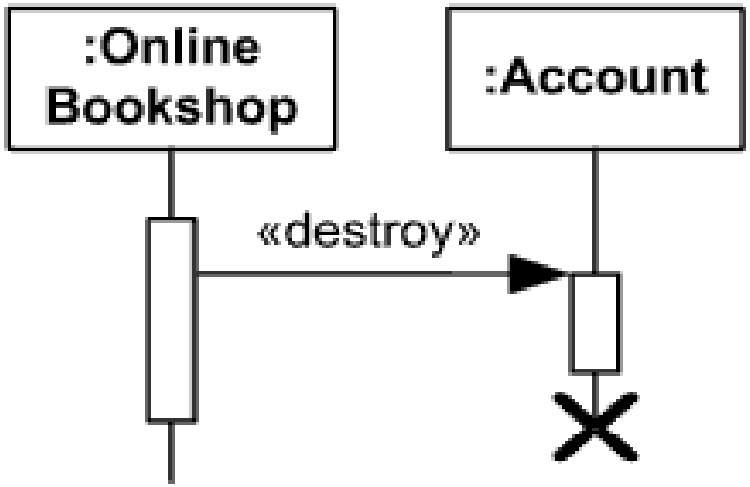
- **Create** message is sent to lifeline to create itself. Note, that it is weird but common practice in OOAD to send create message to a nonexisting object to create itself. In real life, create message is sent to some runtime environment.
- Create message is shown as a dashed line with open arrowhead (same as reply), and pointing to created lifeline's head.



Online Bookshop creates Account.

Delete Message

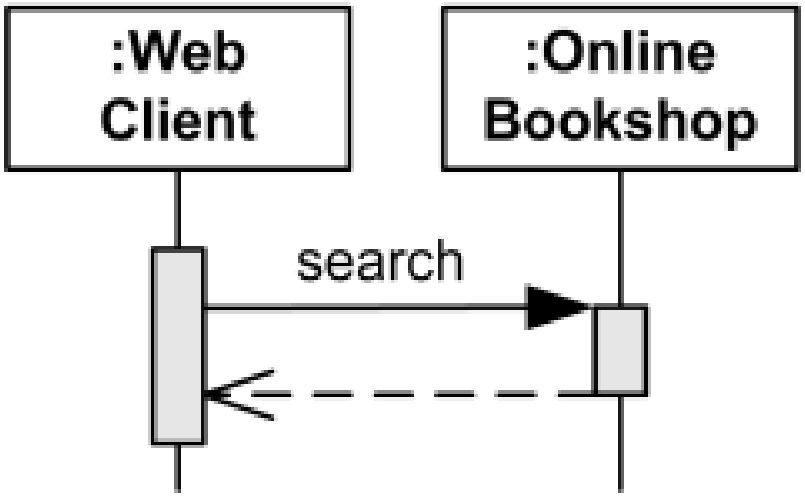
- **Delete** message (called **stop** in previous versions of UML) is sent to terminate another lifeline. The lifeline usually ends with a cross in the form of an **X** at the bottom denoting destruction occurrence.
- use custom «destroy» stereotype.



Online Bookshop terminates Account.

Reply Message

- **Reply** message to an operation call is shown as a dashed line with open arrow head (looks similar to **creation message**).

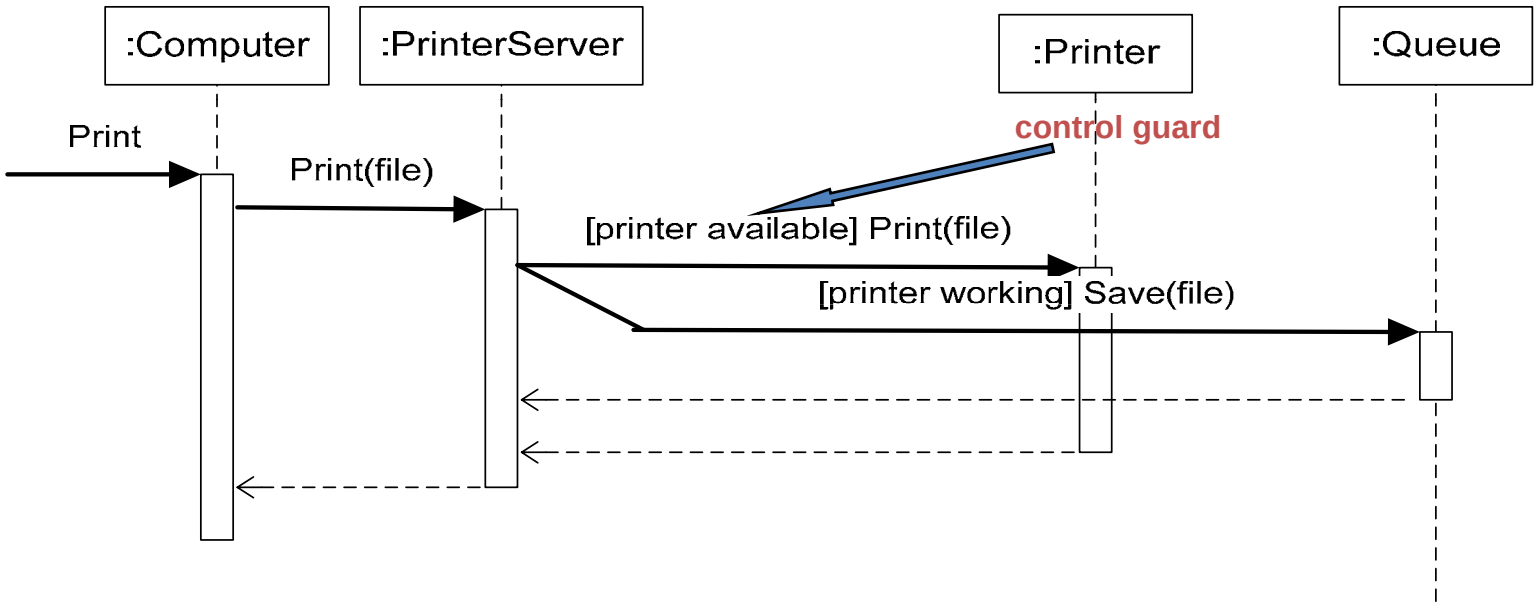


Web Client searches Online Bookshop and waits for results to be returned.

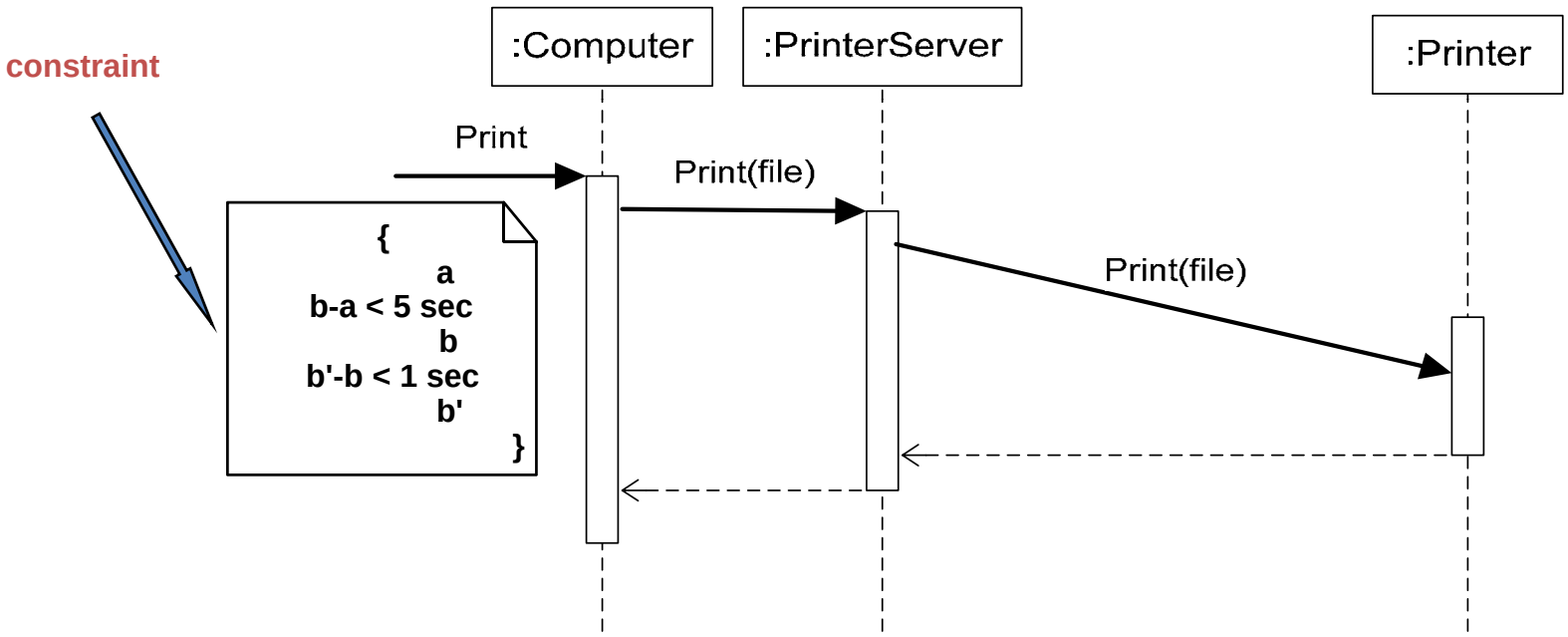
Messages by Presence of Events

- Depending on whether message send event and receive events are present, message could be one of:
- **complete message**
- lost message
- found message
- **unknown message** (default)
- The semantics of a **complete** message is the trace <sendEvent, receiveEvent>. Both sendEvent and receiveEvent are present.
- **Unknown message** - both sendEvent and receiveEvent are absent (should not appear).

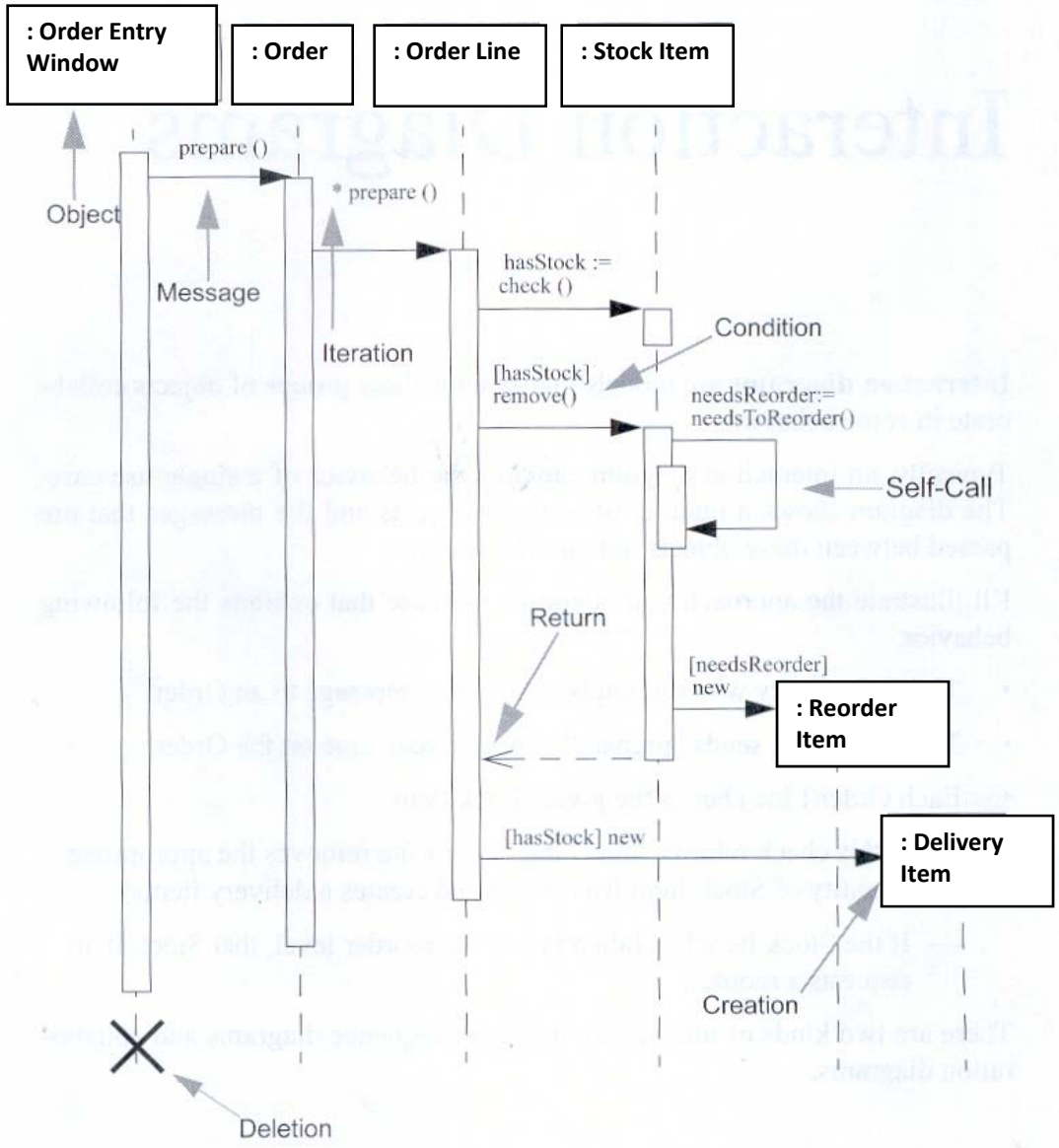
Next Concepts Sequence Diagrams 1of2



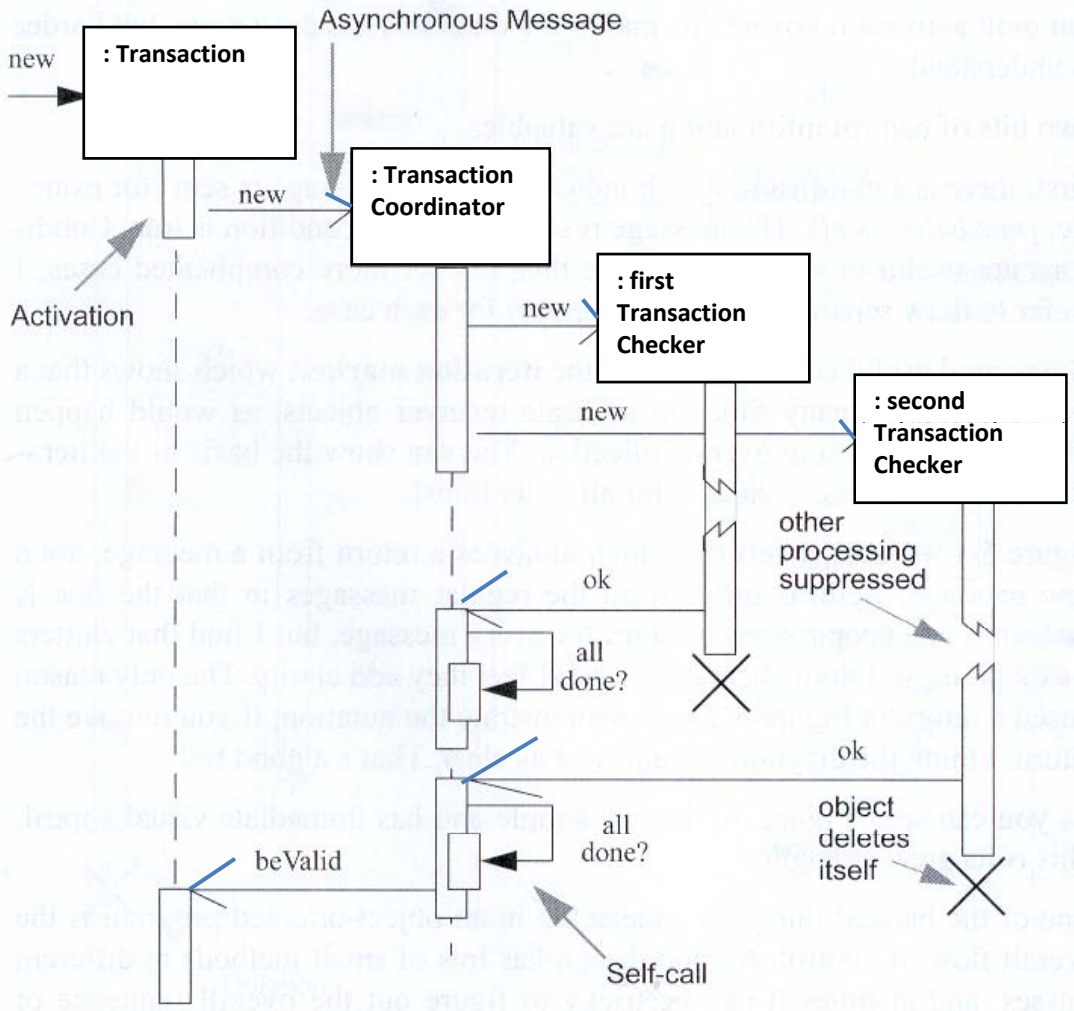
Next Concepts Sequence Diagrams 2of2



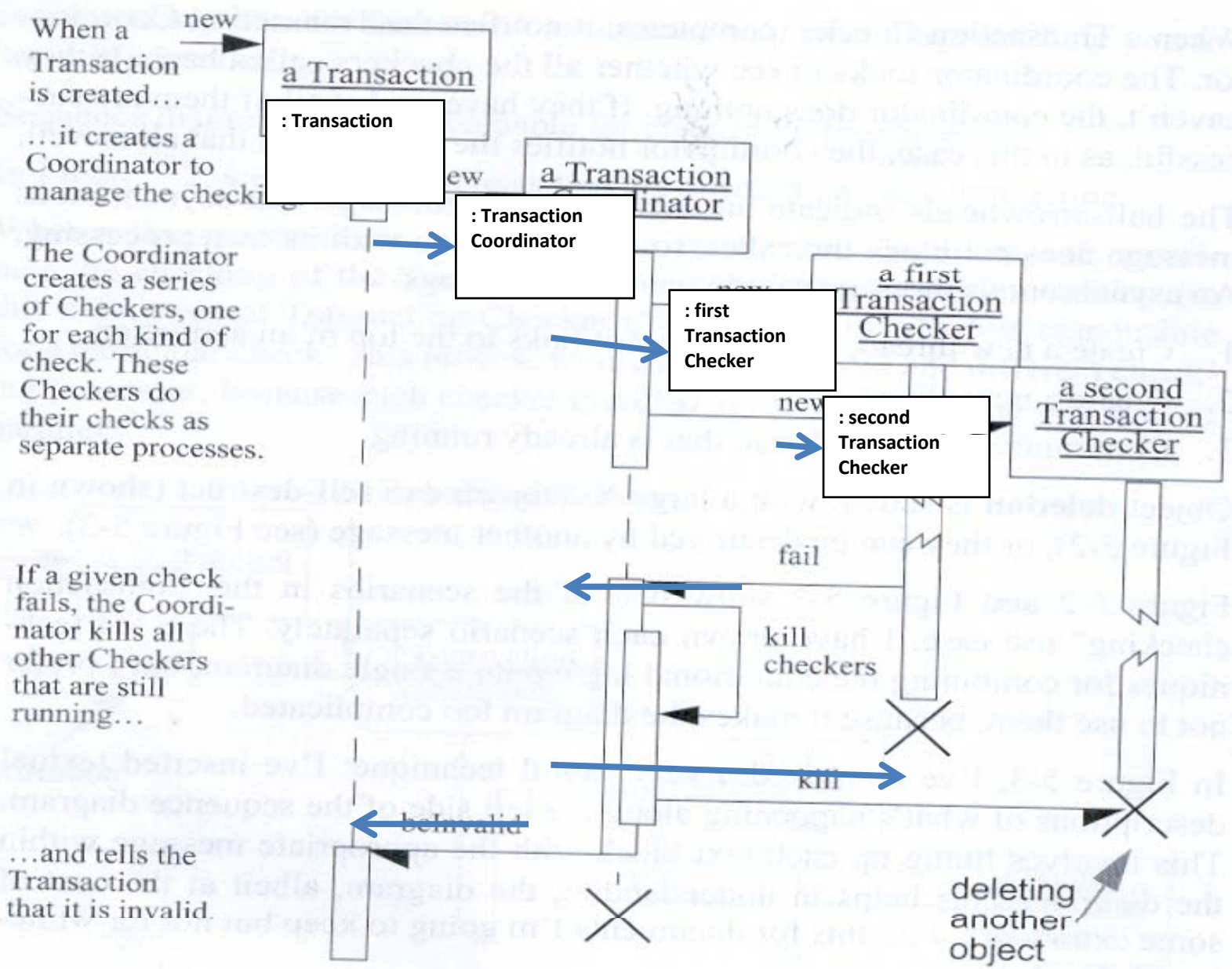
SEQUENCE DIAGRAM



Concurrent Processes and Activations



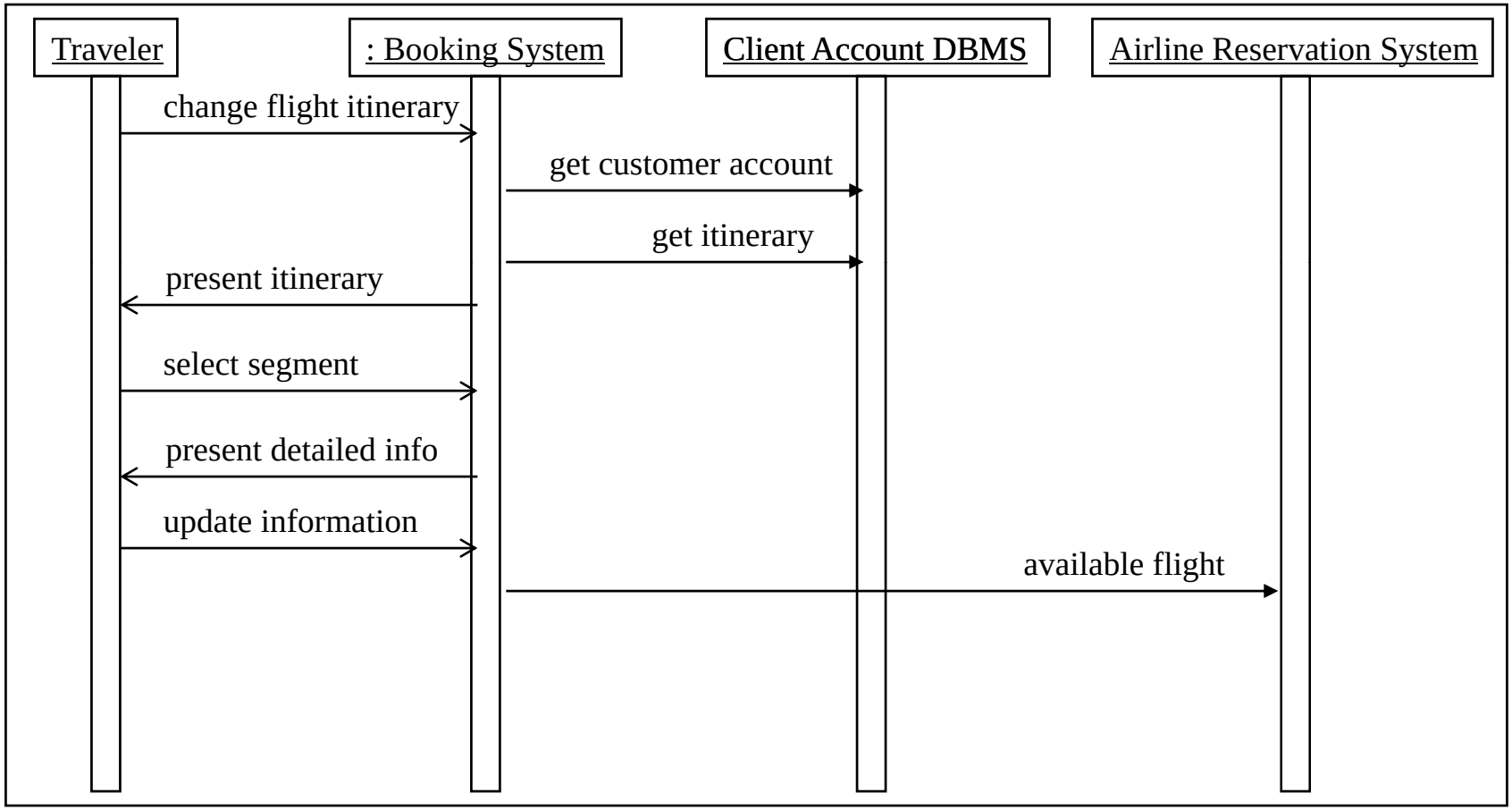
SEQUENCE DIAGRAM: Check Failure



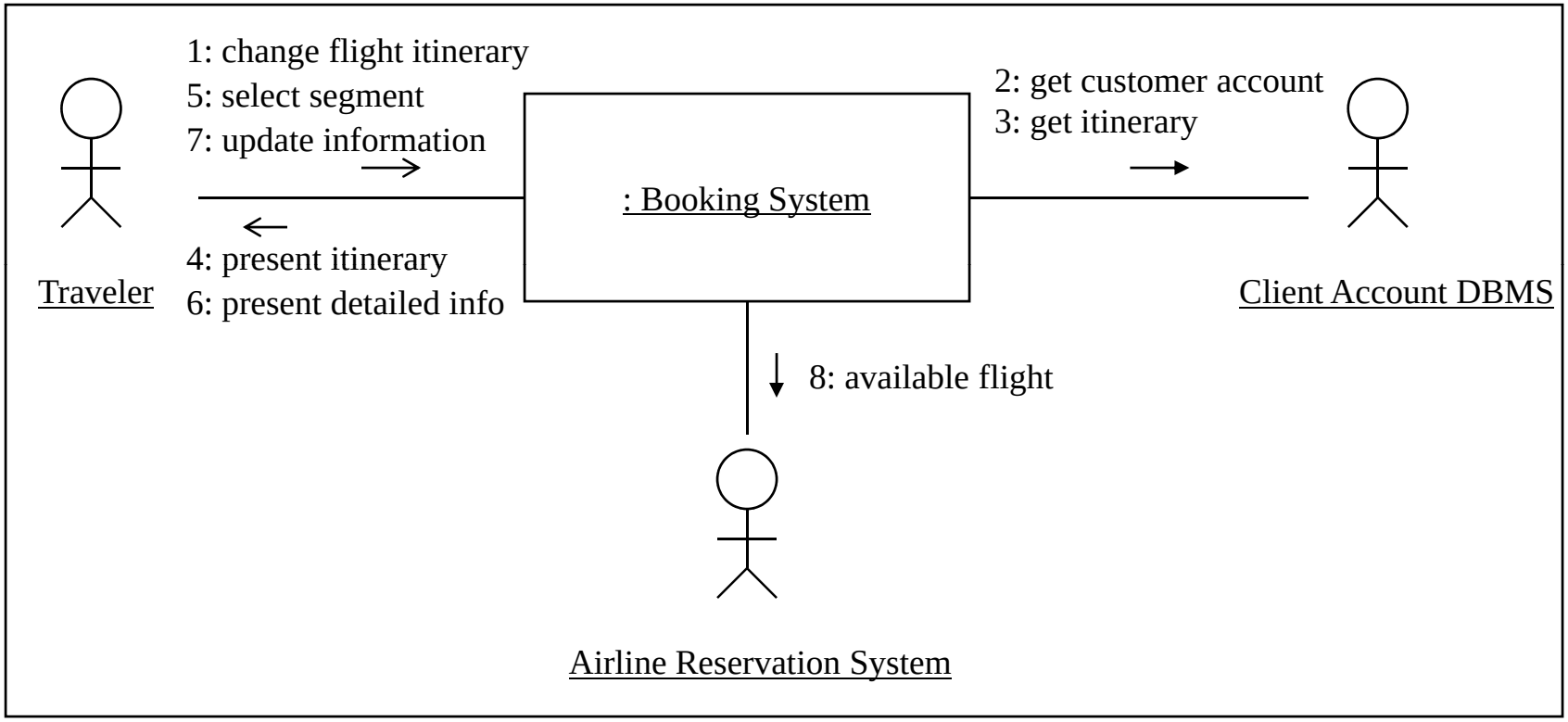
Example: A Booking System
Use Case Description: Change Flt Itinerary

- **Actors:** traveler, client account db, airline reservation system
- **Preconditions:** Traveler has logged in
- **Basic course:**
  - Traveler selects 'change flight itinerary' option
  - System retrieves traveler's account and flight itinerary from client account database
  - System asks traveler to select itinerary segment she wants to change; traveler selects itinerary segment.
  - System asks traveler for new departure and destination information; traveler provides information.
  - If flights are available then …
  - …
  - System displays transaction summary.
- **Alternative course:**
  - If no flights are available then…

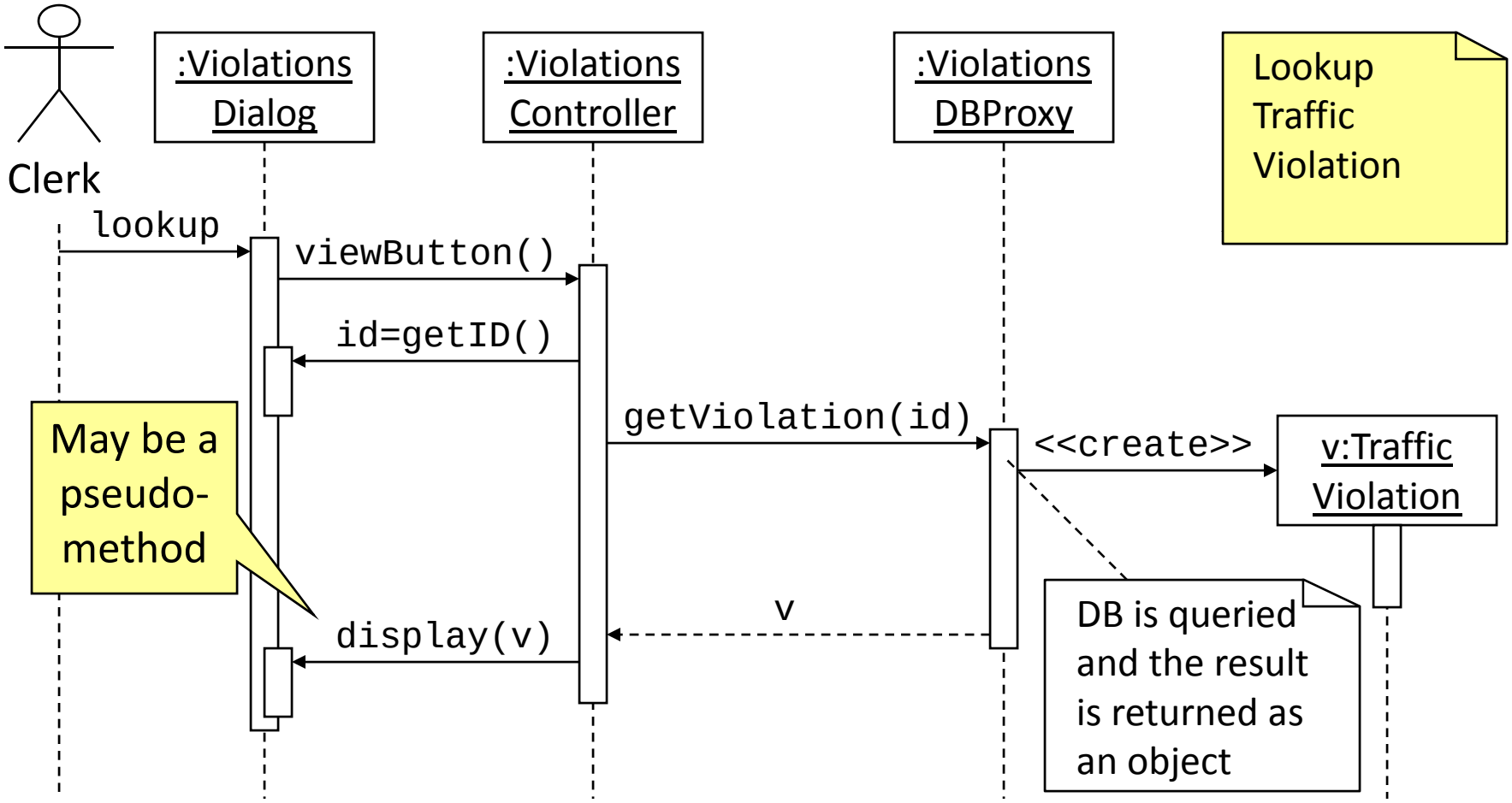
Sequence Diagram: Change Flight Itinerary



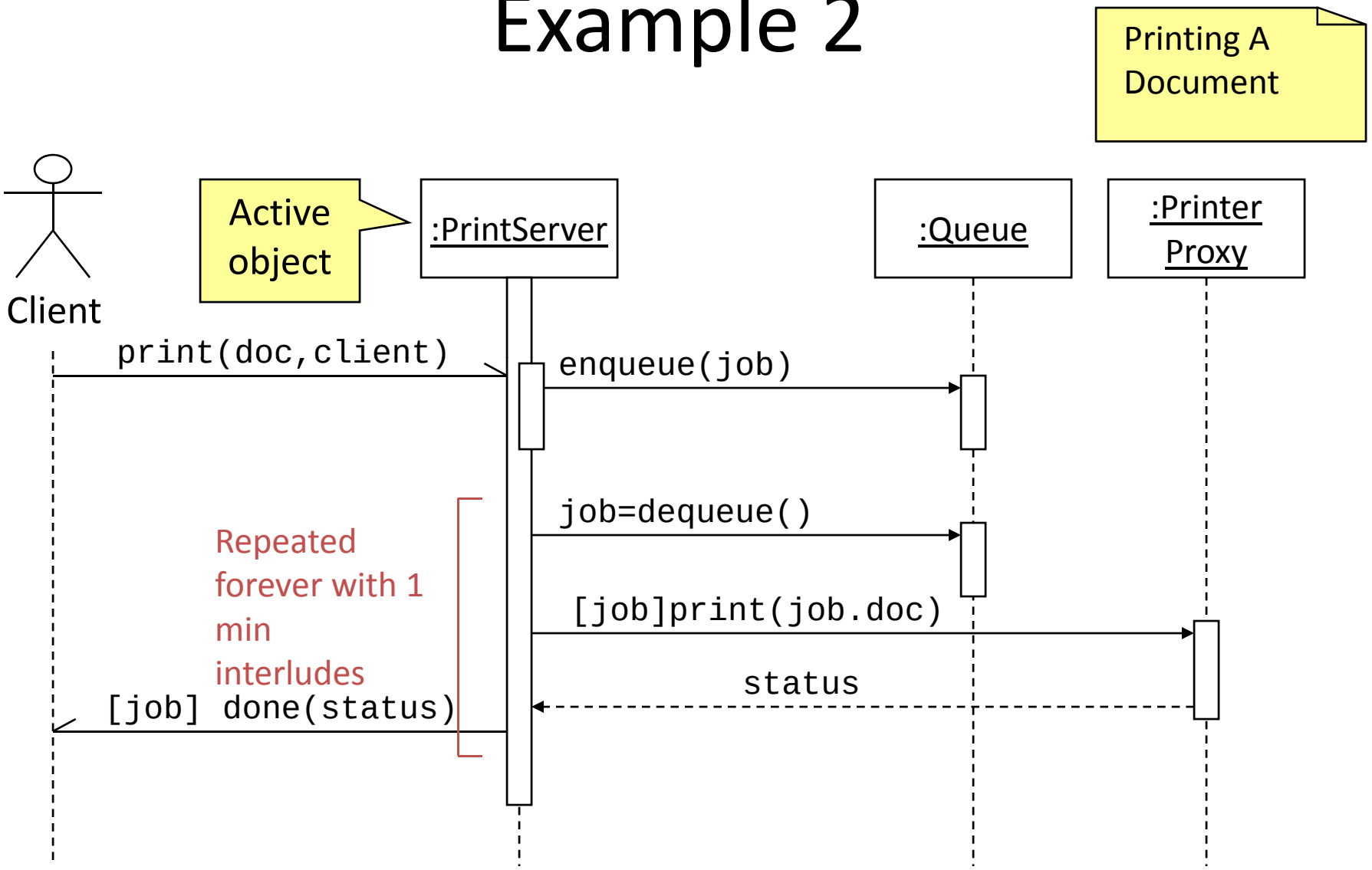
Collaboration Diagram: Change Flt Itinerary



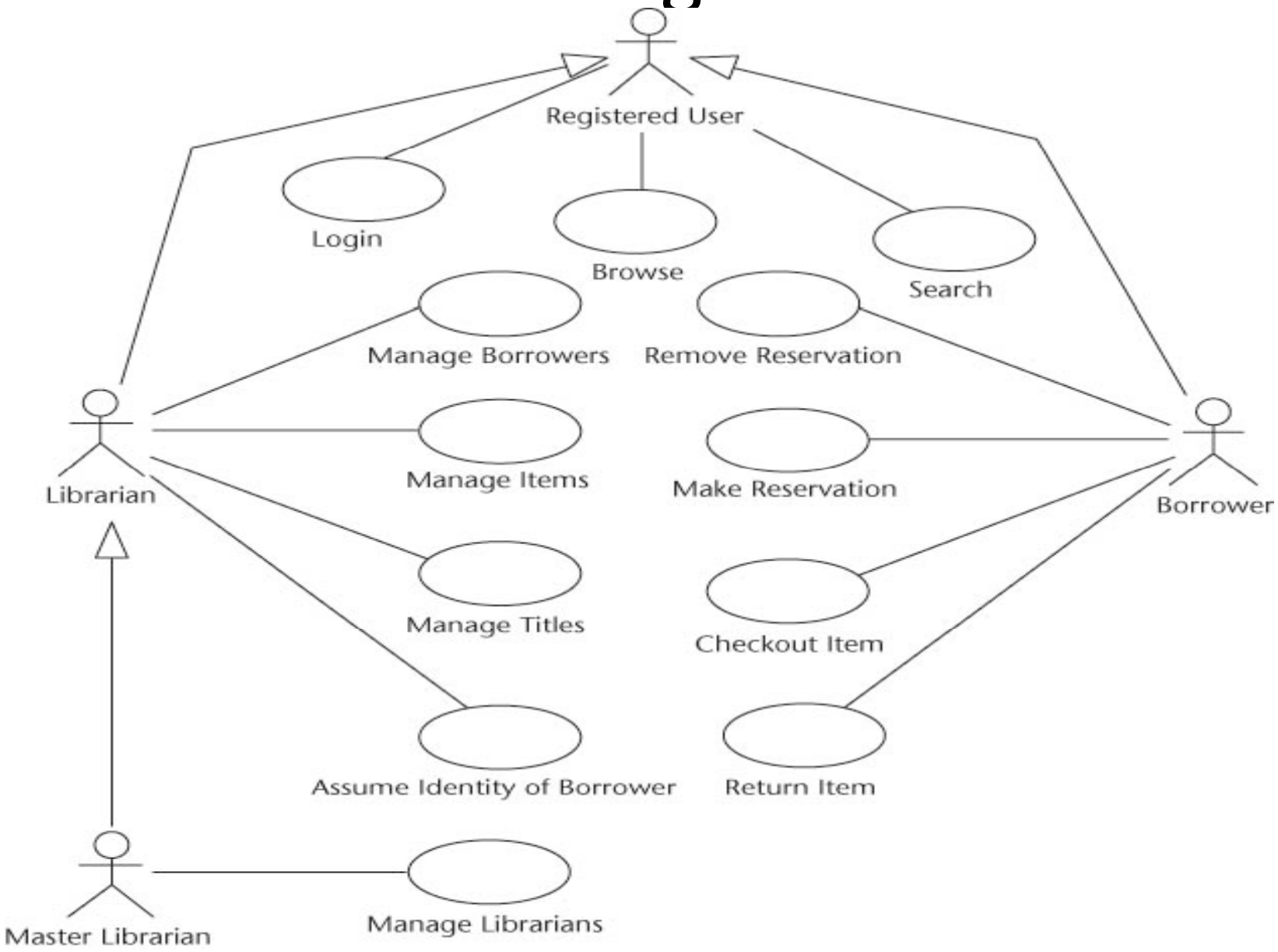
Example 1


Clerk
:Violations Dialog
:Violations Controller
:Violations DBProxy
Lookup Traffic Violation
lookup
viewButton()
id=getID()
May be a pseudo-method
getViolation(id)
<<create>>
v:Traffic Violation
v
display(v)
DB is queried and the result is returned as an object

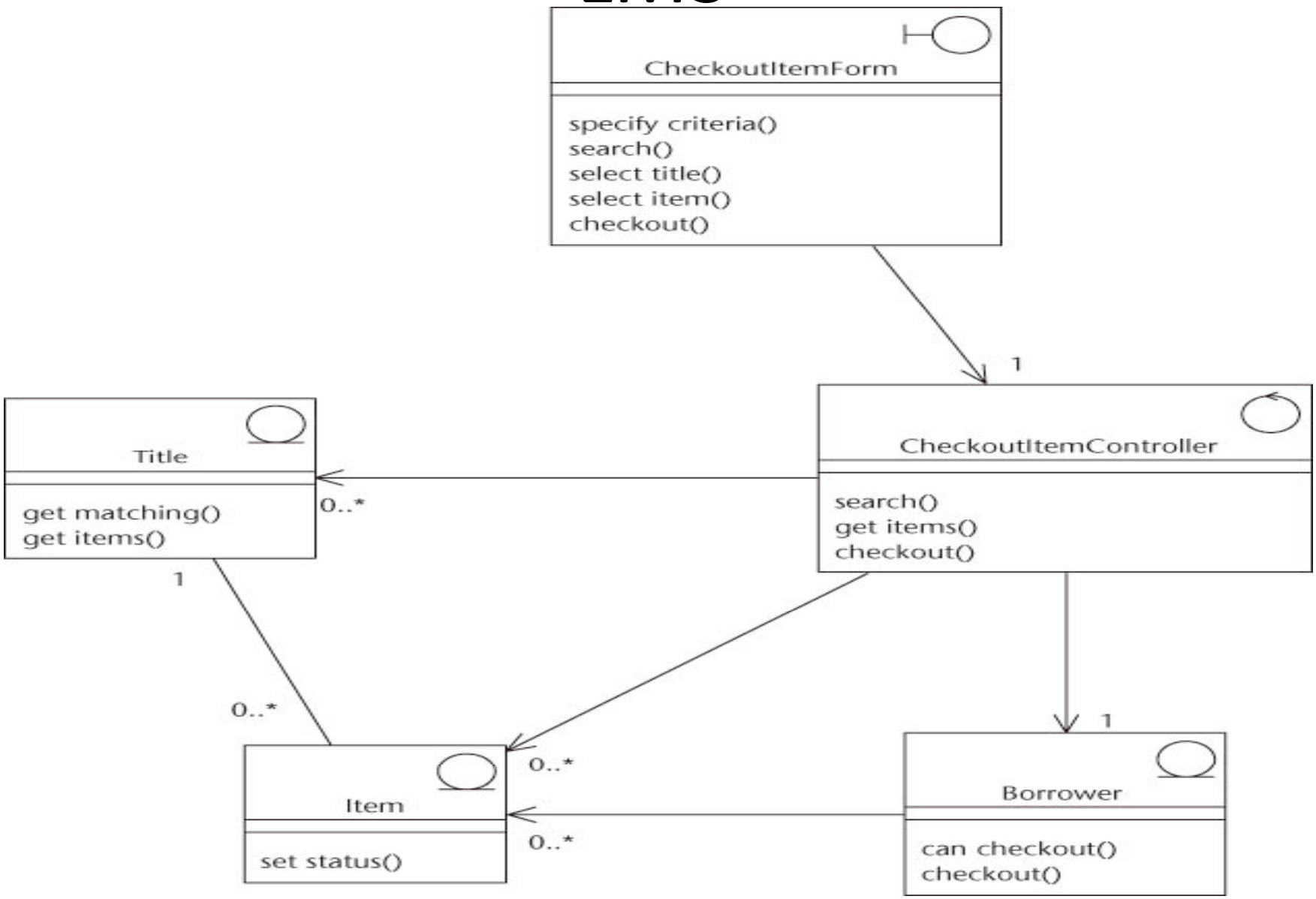
Example 2

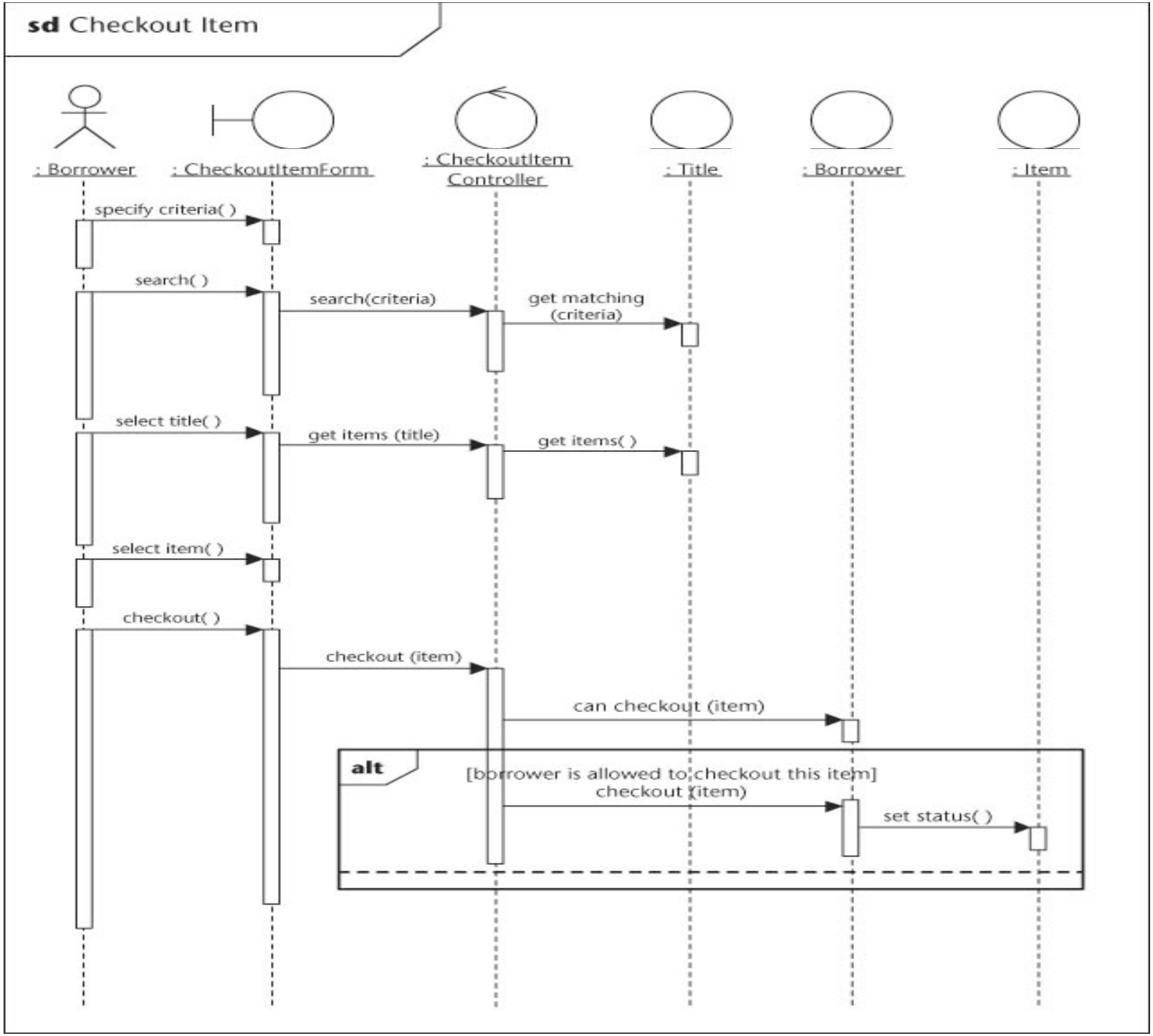

Printing A Document
Client
Active object
:PrintServer
:Queue
:Printer Proxy
print(doc,client)
enqueue(job)
job=dequeue()
Repeated forever with 1 min interludes
[job]print(job.doc)
status
[job] done(status)

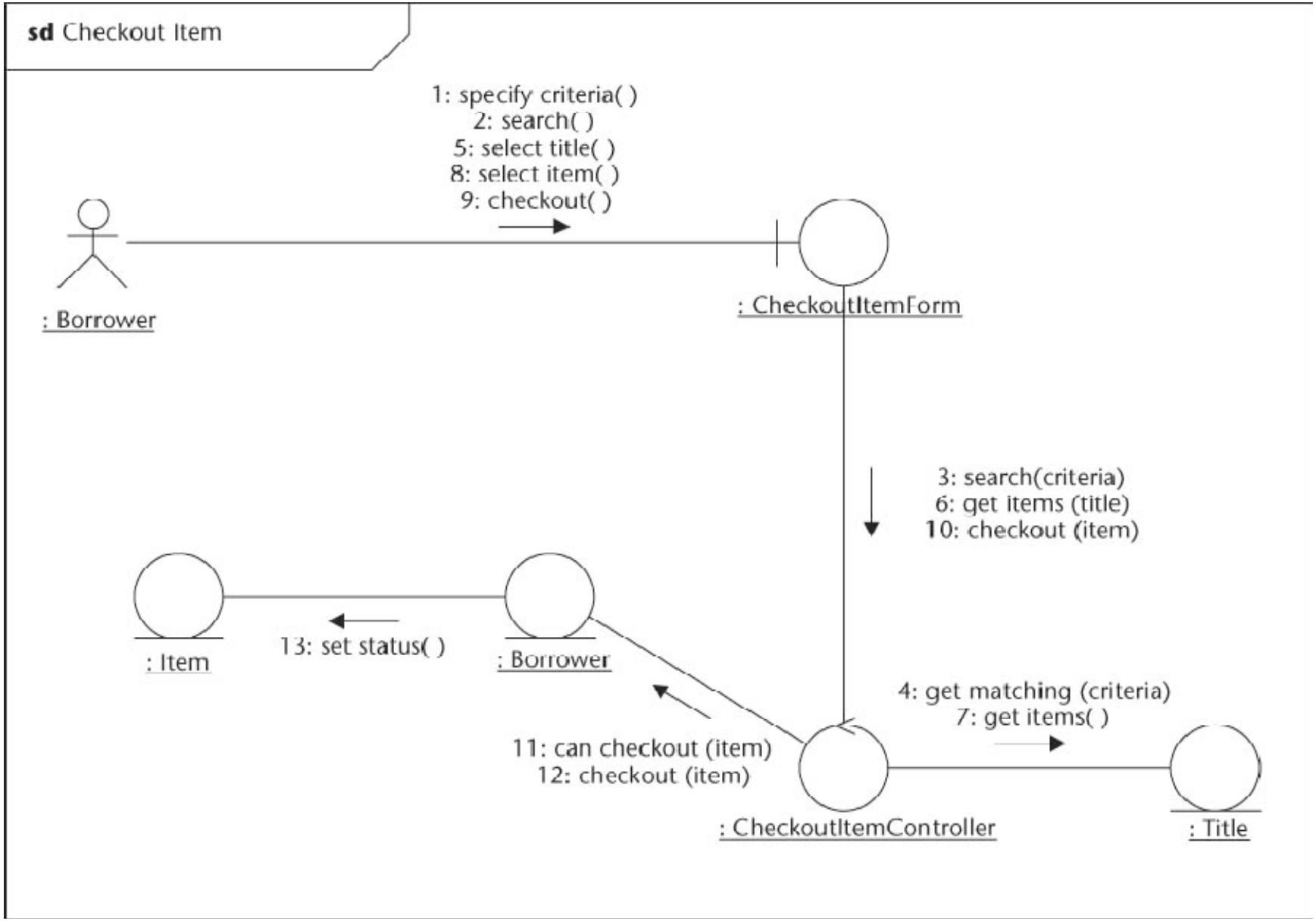
Use case Diagram for LMS



LMS


CheckoutItemForm
specify criteria()
search()
select title()
select item()
checkout()
1
Title
get matching()
get items()
0..*
CheckoutItemController
search()
get items()
checkout()
1
0..*
1
Item
set status()
0..*
0..*
Borrower
can checkout()
checkout()


sd Checkout Item
: Borrower
: CheckoutItemForm
: CheckoutItem Controller
: Title
: Borrower
: Item
specify criteria( )
search( )
search(criteria)
get matching (criteria)
select title( )
get items (title)
get items( )
select item( )
checkout( )
checkout (item)
can checkout (item)
alt
[borrower is allowed to checkout this item]
checkout (item)
set status( )


sd Checkout Item
1: specify criteria( )
2: search( )
5: select title( )
8: select item( )
9: checkout( )
: Borrower
: CheckoutItemForm
3: search(criteria)
6: get items (title)
10: checkout (item)
: Item
13: set status( )
: Borrower
4: get matching (criteria)
7: get items( )
11: can checkout (item)
12: checkout (item)
: CheckoutItemController
: Title

Java API provides a large number of classes grouped into different packages according to functionality.

java.lang: Language support classes. These are the classes that Java compiler itself uses and therefore they are automatically imported. They include classes of primitive types, strings, math functions, threads and exceptions.

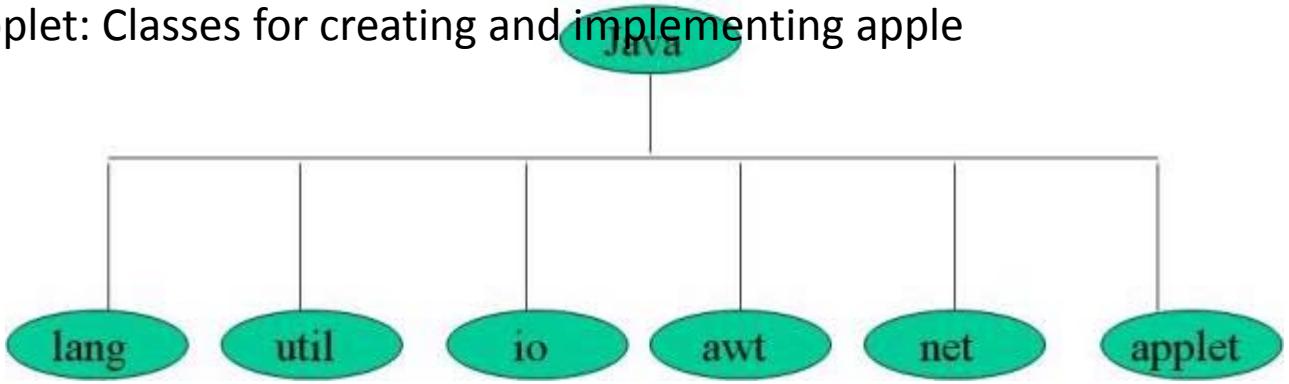
java.util: Language utility classes such as vector, hash tables, random numbers, data etc.

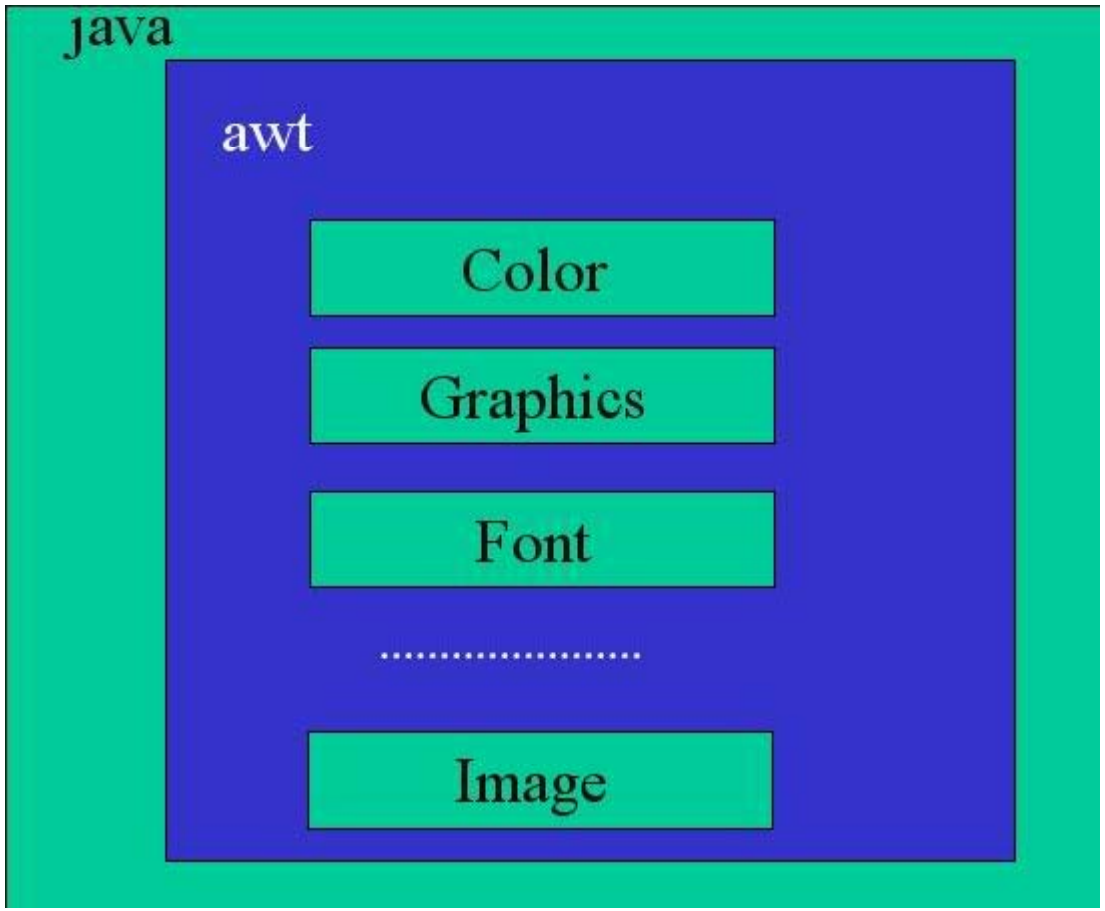
java.io: Input/output support classes. They provide facilities for the input and output of data

java.awt: set of classes for implementing graphical user interface. They include classes for windows, buttons, lists, menus and so on.

java.net: Classes for networking. They include classes for communicating with local computers as well as with internet servers.

java.applet: Classes for creating and implementing apple



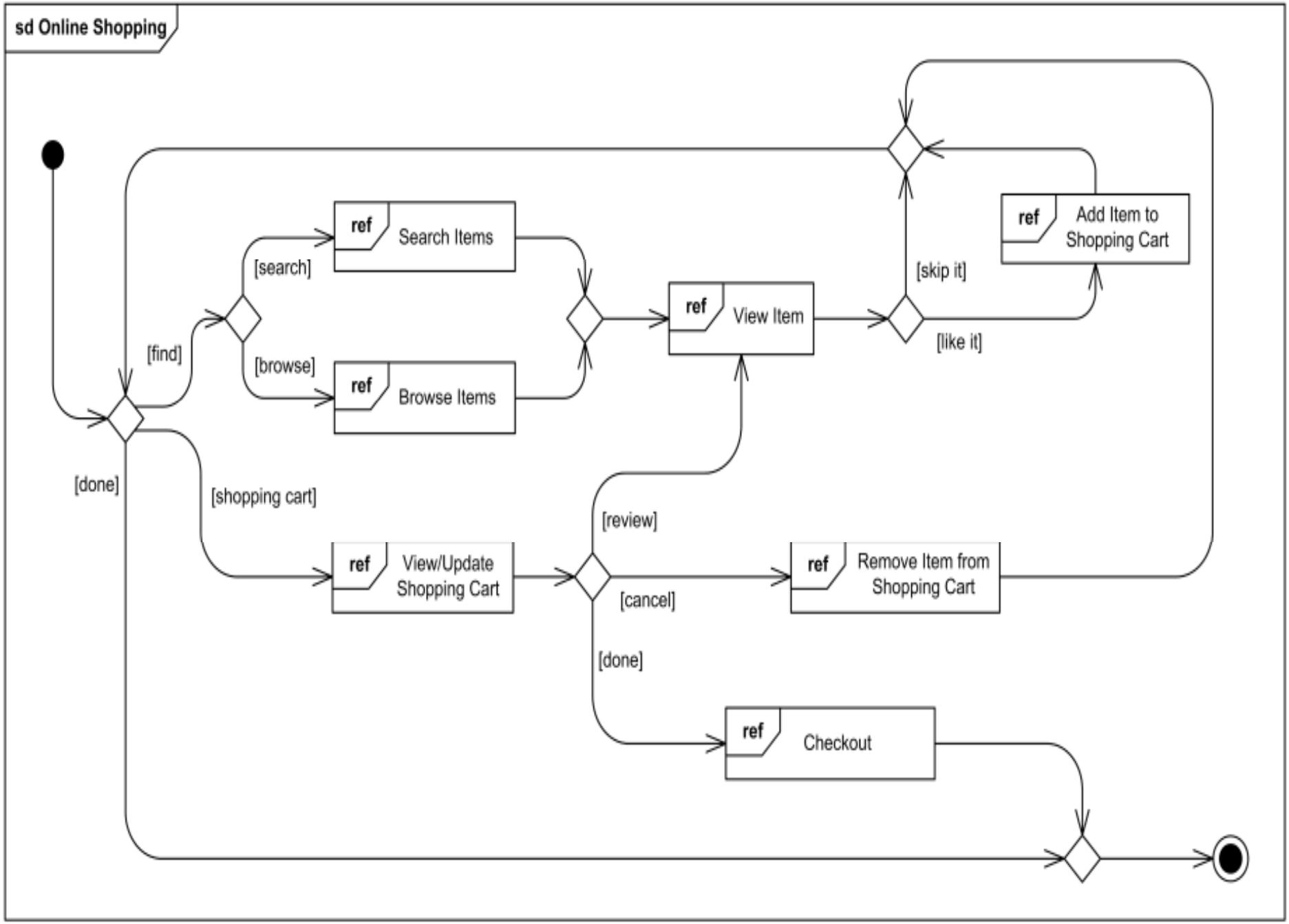

java
awt
Color
Graphics
Font
.....
Image

Interaction overview Diagram

- Interaction overview diagrams provide overview of the flow of control where nodes of the flow are **interactions** or interaction uses.
- specialization of activity diagrams.
- Interaction overview diagrams do look like activity diagrams that can only have inline interactions or interaction uses instead of **invocation actions**. The inline interactions and interaction uses are considered as special forms of **call behavior action**.

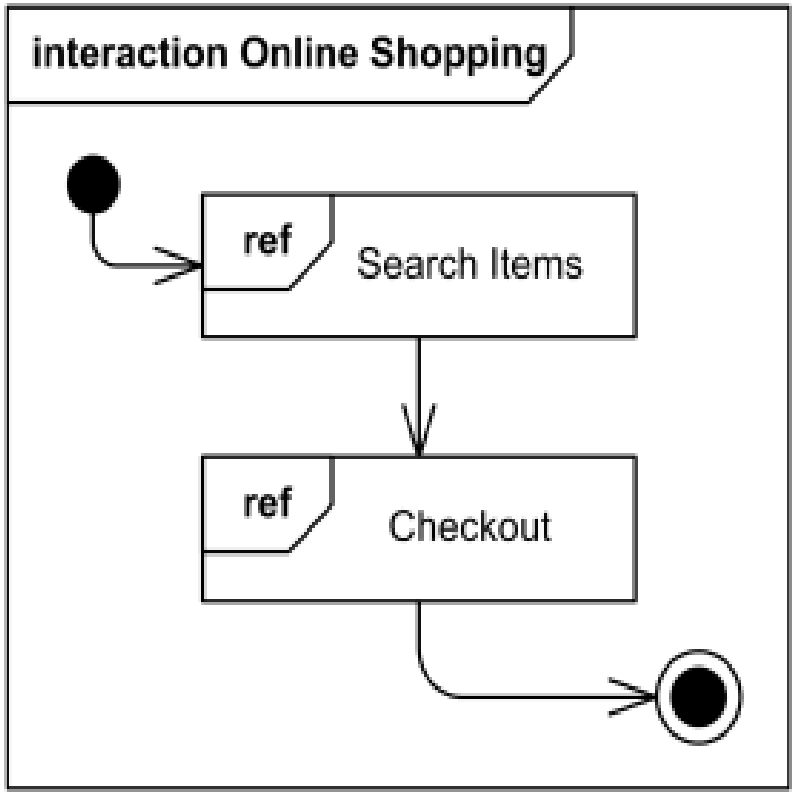
Common behaviors

- In UML specification, the common behaviors specify the core concepts required for **dynamic elements** and provide the infrastructure to support more detailed definitions of behavior.
- Elements of the common behaviors could be used when creating behavior diagrams.
- Common behaviors include: behavior, behaviored classifier, event, trigger, signal, reception, interval constraint, duration constraint, time constraint.


sd Online Shopping
ref
Search Items
ref
Add Item to
Shopping Cart
[search]
[skip it]
ref
View Item
[find]
[browse]
[like it]
ref
Browse Items
[done]
[shopping cart]
[review]
ref
View/Update
Shopping Cart
ref
Remove Item from
Shopping Cart
[cancel]
[done]
ref
Checkout

Frame

- Interaction overview diagrams are framed by the same kind of frame that encloses other forms of interaction diagrams - a rectangular frame around the diagram with a name in a compartment in the upper left corner. Interaction kind is **interaction** or **sd** (abbreviated form). Note, that UML has no **io** or **iod** abbreviation as some would expect.



Interaction overview diagram **Online Shopping**

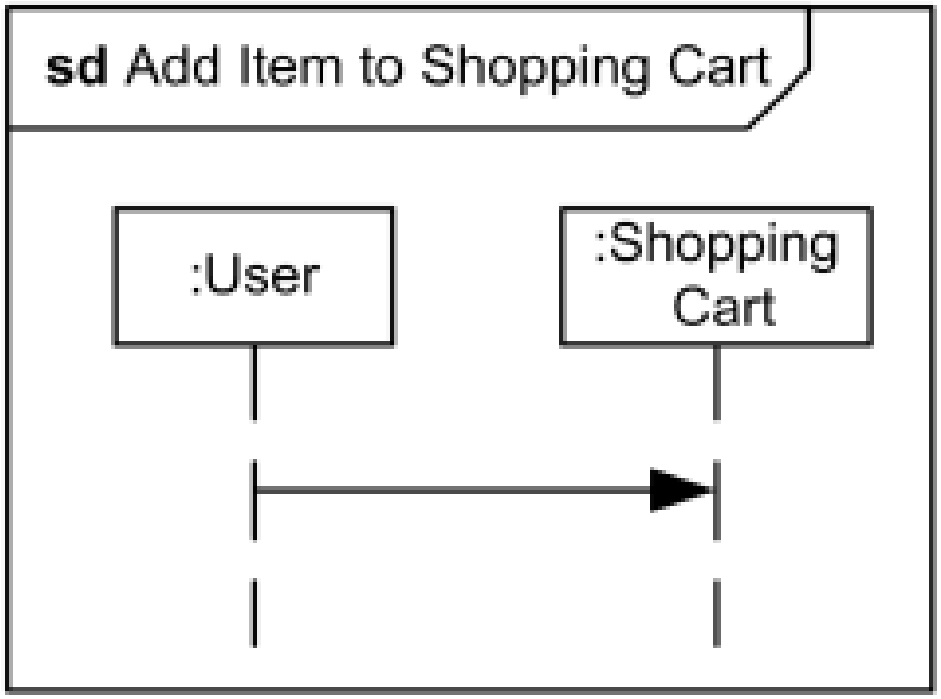
Elements of Activity Diagram

- **Interaction overview diagrams** are defined as specialization of activity diagrams and as such they inherit number of graphical elements.
- initial node
- flow final node
- activity final node
- decision node
- merge node
- fork node
- join node

Elements of Interaction Diagram

- interaction
- interaction use
- duration constraint
- time constraint

Interaction



Interaction Add Item to Shopping Cart may appear inline on some interaction overview diagram

Interaction Use

An interaction use may appear as an **invocation action**.