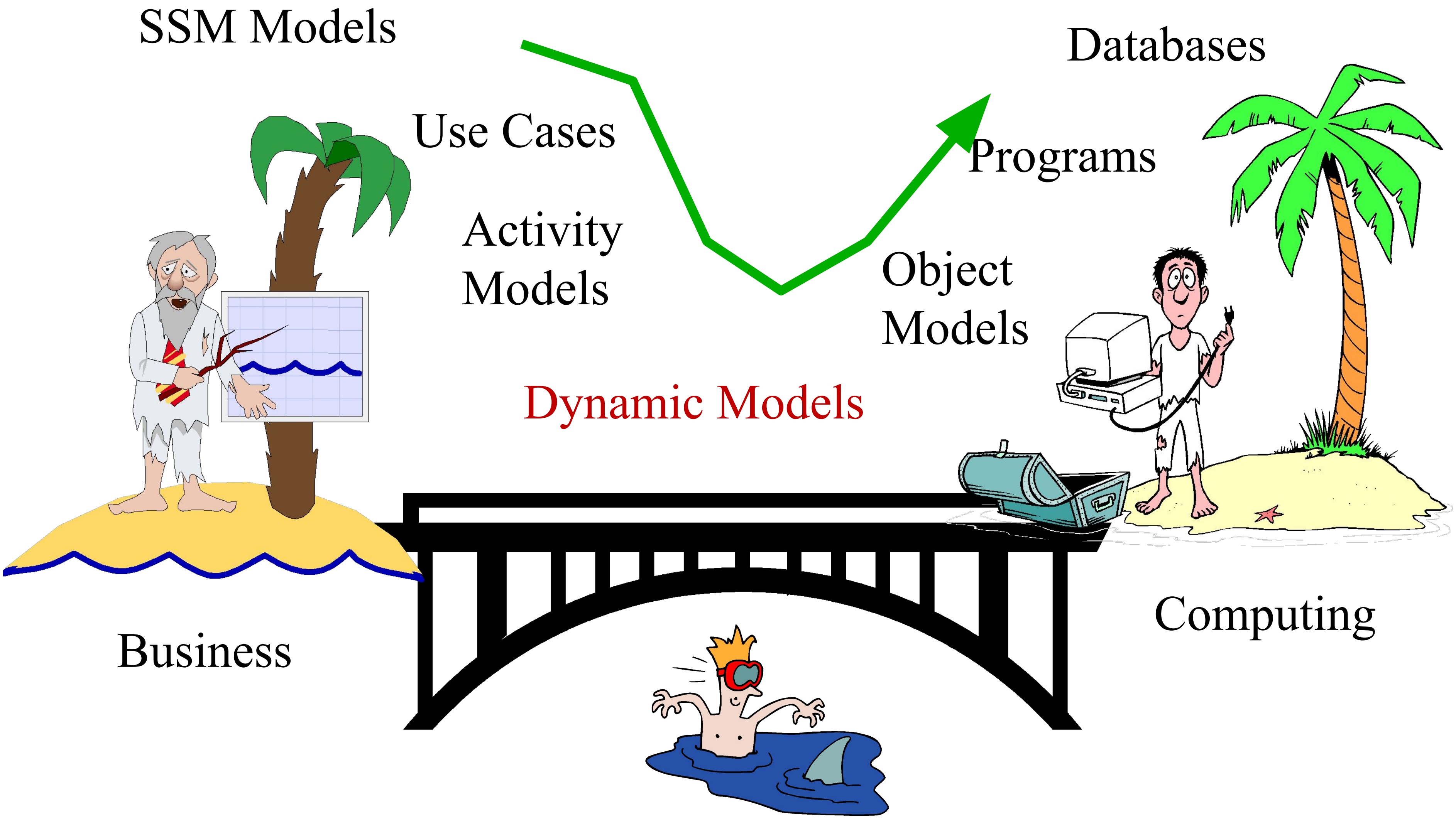
The Vision



SSM Models

Use Cases

Activity Models

Dynamic Models

Object Models

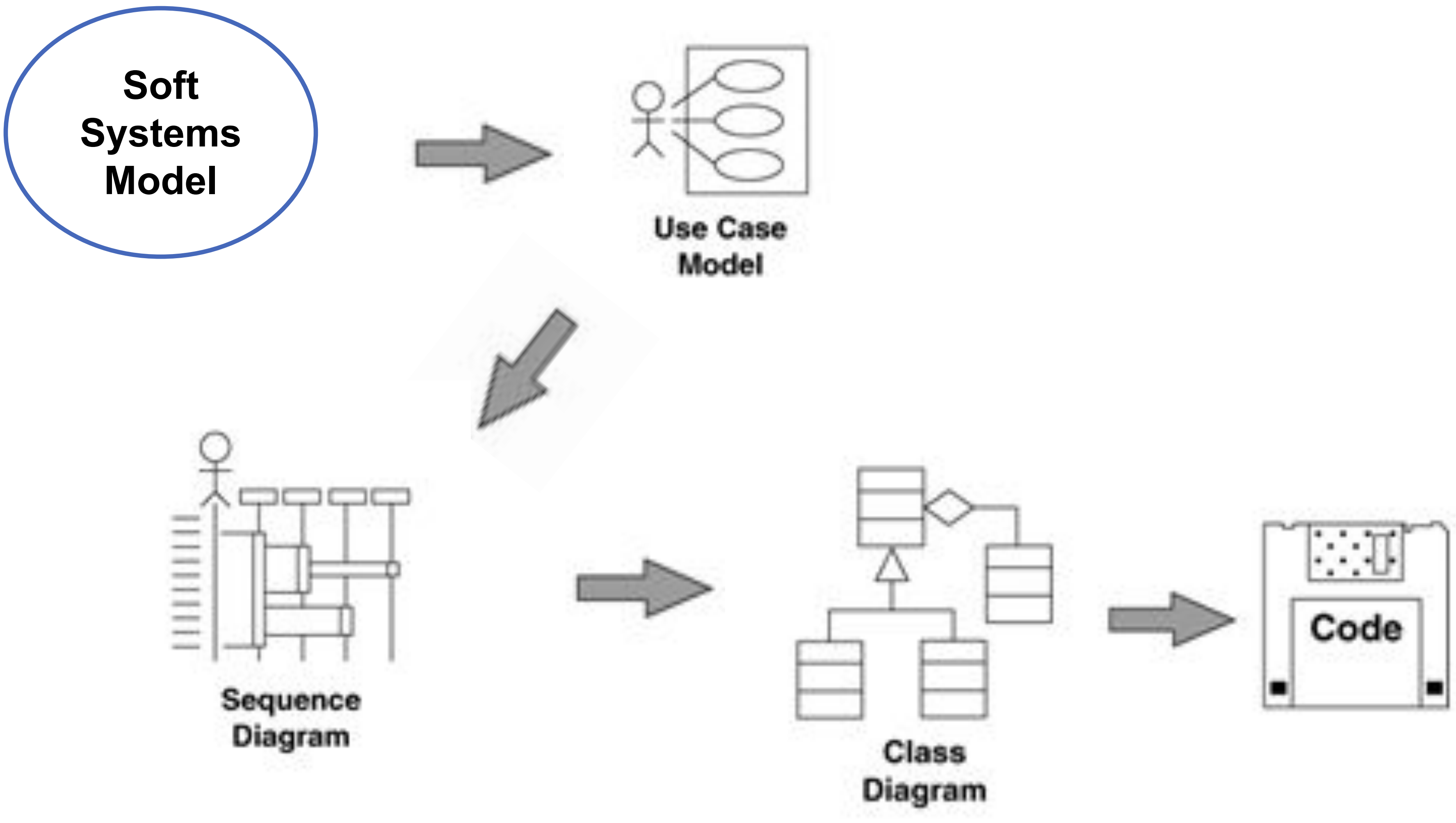
Programs

Databases

Business

Computing

Beginnings of a Method



Soft Systems Model

Use Case Model

Sequence Diagram

Class Diagram

Code

Types of Diagrams

- **Structural Diagrams – focus on static aspects of the software system**
  - **Class, Object, Component, Deployment**

- **Behavioral Diagrams – focus on dynamic aspects of the software system**
  - **Use-case, Interaction, State Chart, Activity**

Structural Diagrams

- Class Diagram – set of classes and their relationships. Describes interface to the class (set of operations describing services)
- Object Diagram – set of objects (class instances) and their relationships
- Component Diagram – logical groupings of elements and their relationships
- Deployment Diagram - set of computational resources (nodes) that host each component.

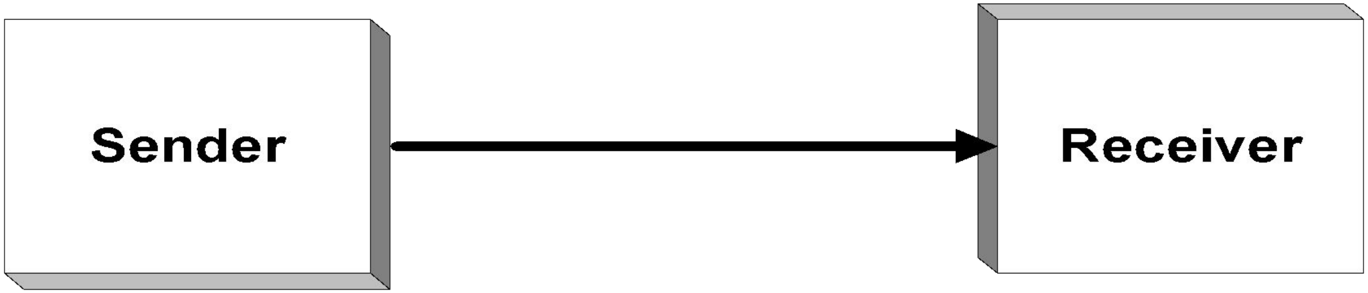
Behavioral Diagram

- **Use Case Diagram – high-level behaviors of the system, user goals, external entities: actors**
- **Sequence Diagram – focus on time ordering of messages**
- **Collaboration Diagram – focus on structural organization of objects and messages**
- **State Chart Diagram – event driven state changes of system**
- **Activity Diagram – flow of control between activities**

Systems Activities

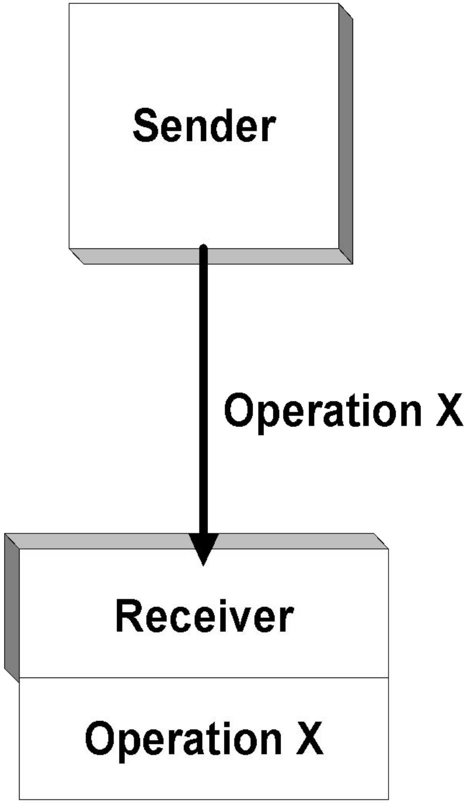
- The systems functionality is represented as a number of Use Cases
- The functionality of each use case will be realised through objects collaborating with each other
- Collaboration is achieved through message passing

Message Connections



- **The arrow indicates that:**
  - **The sender sends a message**
  - **The receiver receives the message**
  - **The receiver takes some action, returning the result to the sender**

Message Connections



- The message must activate an operation in the receiving object

Message Connections

There are two ways of knowing which object to send a message to:

(1) An association exists between sender and receiver in the object model

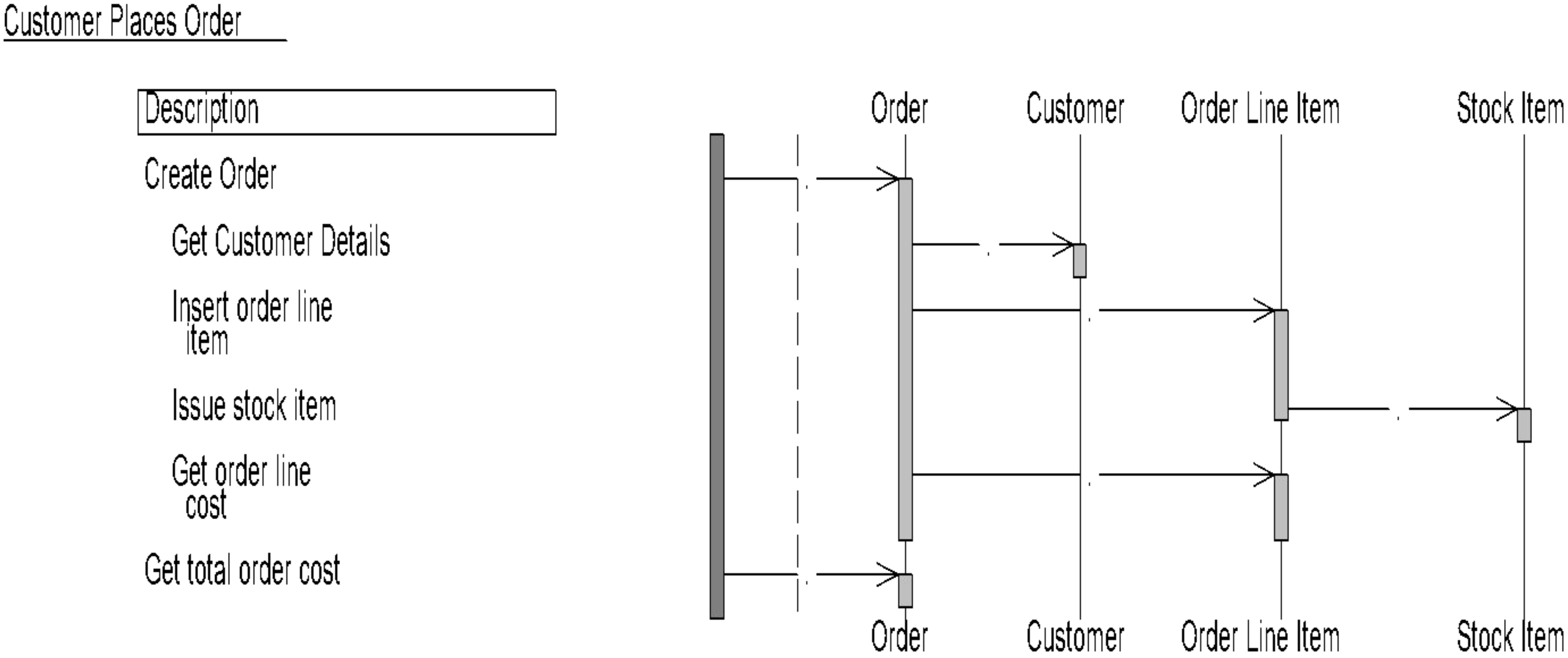
(2) The receiver’s *object id* is passed as part of the message (i.e. as a parameter)

Message Connections

Sequence Diagrams allow us to describe object communication associated with a specific use case

- **Can be used:**
  - **during analysis to help define an object's responsibilities**
  - **as documentation for the final implemented software**

Sequence Diagram for Placing an Order



Placing an Order

- **A message is sent to the *order* class to create a new "order"**
  - ***Customer No., Stock items* and *Quantities* are passed as parameters**
  - **Customer details are retrieved from the appropriate customer object**
  - **For each *stock item* an *order line* object is created**
    - **» Details are extracted from the stock item object**

Library Example

We have identified three objects: *Borrower, Book* and *Librarian* and the following relationship:

Issuing a Loan

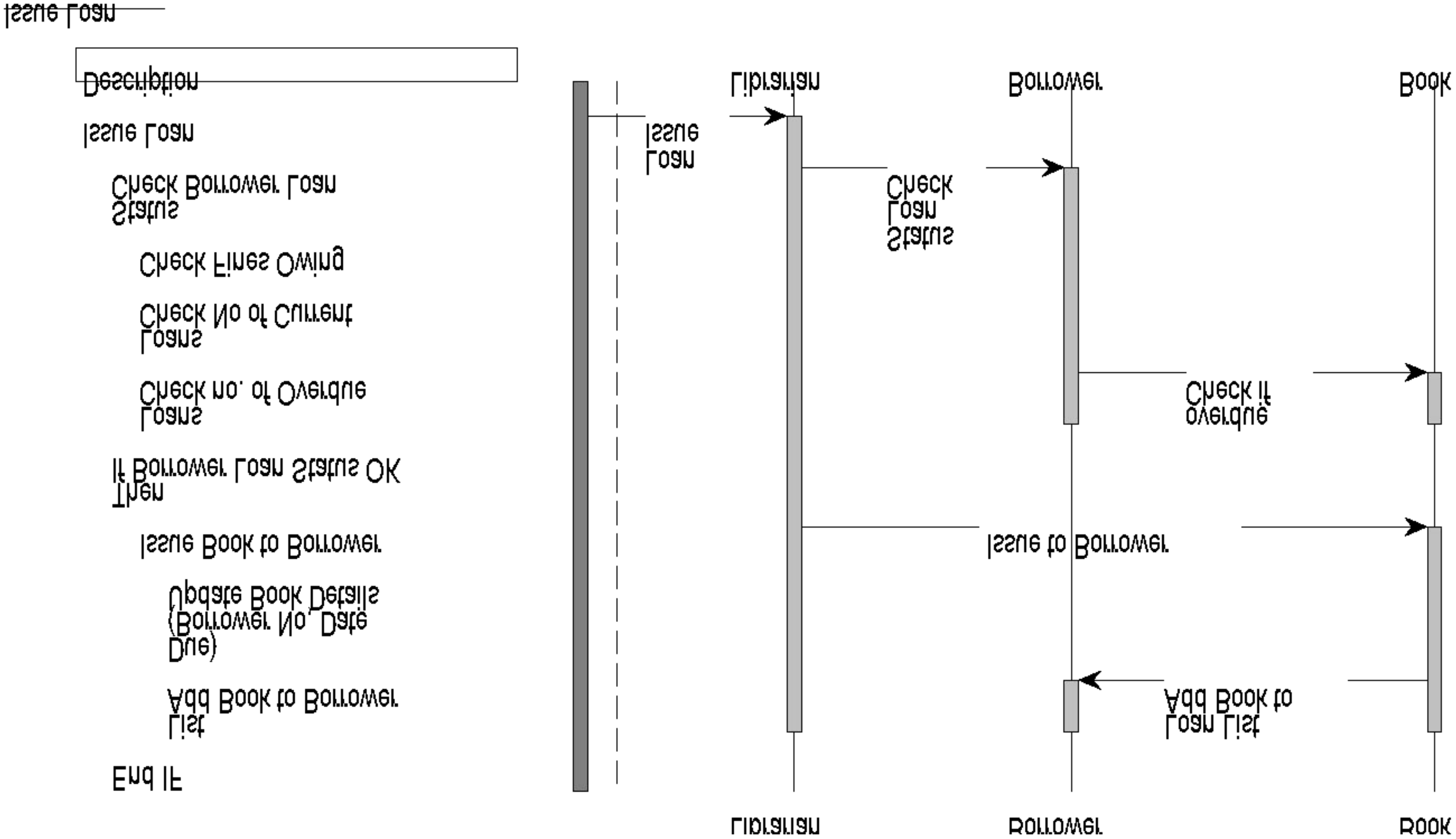
Triggered by a request from a Borrower for the loan of a Book.

Library Example

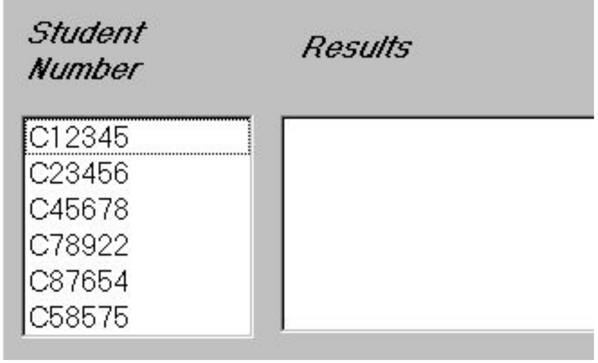
Before issuing the loan we need to check:

1) The borrower has no overdue fines
2) The borrower has not already reached the maximum number of loans that they are allowed.
3) None of the borrower’s current loans are overdue

Issue Loan - Sequence Diag.



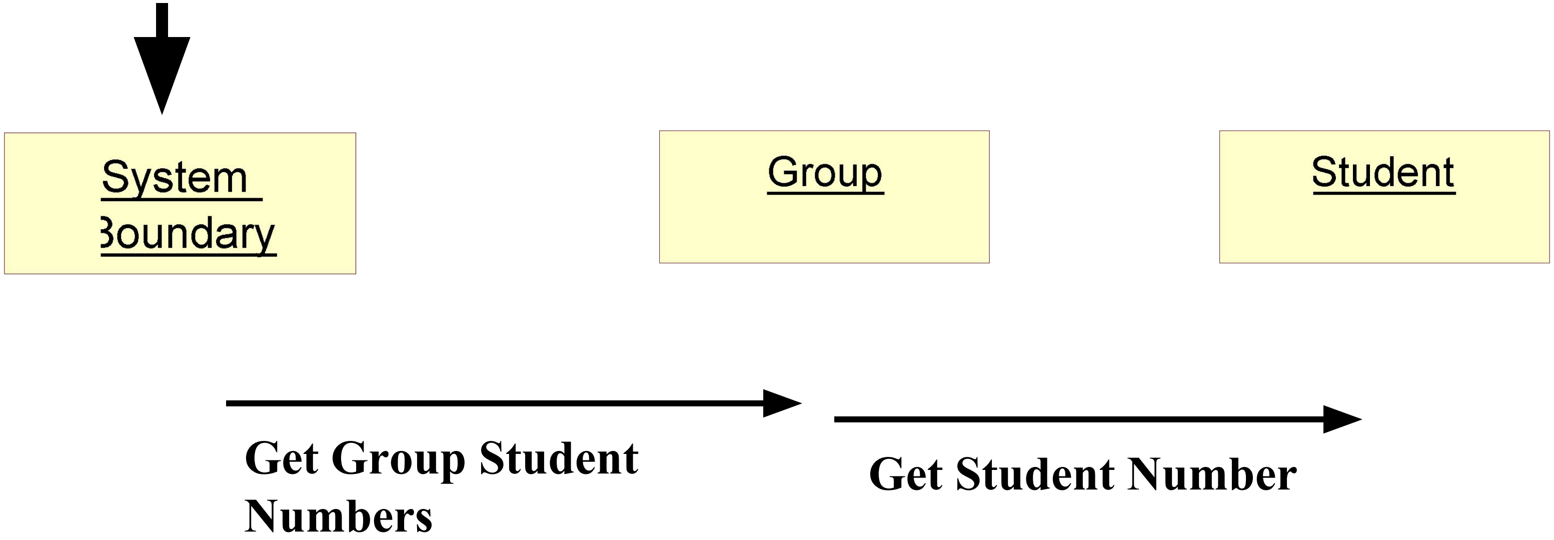
```
Issue Loan

Description
Issue Loan
    Check Borrower Loan Status
        Check Fines Owing
        Check No of Current Loans
        Check no. of Overdue Loans
    If Borrower Loan Status OK Then
        Issue Book to Borrower
            Update Book Details (Borrower No, Date Due)
            Add Book to Borrower List
    End IF
```



Student Number

C12345
C23456
C45678
C78922
C87654
C58575

Results



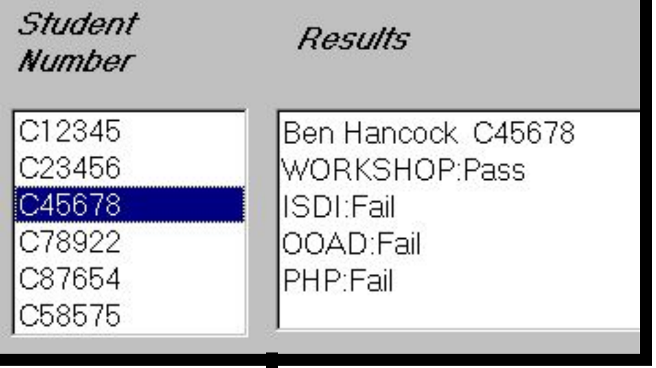
System Boundary

Group

Student

Get Group Student Numbers

Get Student Number



Student Number

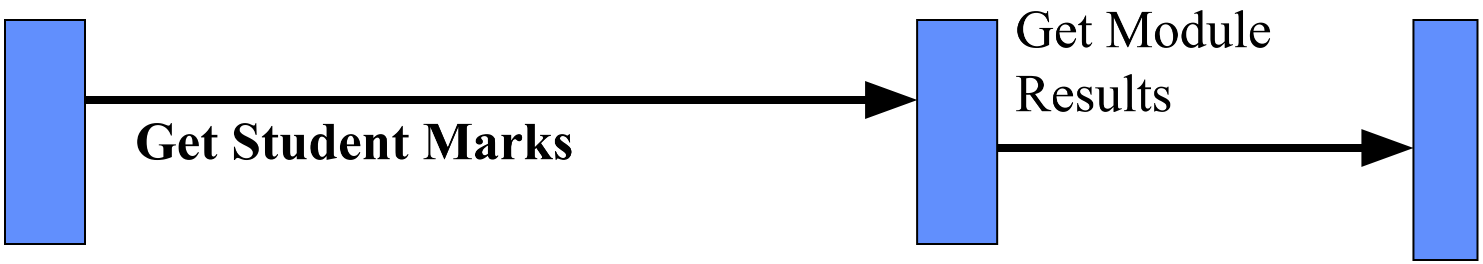
C12345
C23456
C45678
C78922
C87654
C58575

Results

Ben Hancock C45678
WORKSHOP:Pass
ISDI:Fail
OOAD:Fail
PHP:Fail

System Boundary

Module Results



Get Student Marks

Get Module Results

Exercise

Each Instructor has a name, address and telephone number and is qualified to present one or more courses.

We store the date when the instructor became qualified to teach the course

A course has at least one instructor qualified to teach it but it may have many

Each course has a number, title and a date of next revision

Each course will have several scheduled presentations

Details of the date, duration and location are recorded for each presentation

Each presentation will be given by only one instructor but one instructor may give many presentations

The “Reschedule Presentation” use case

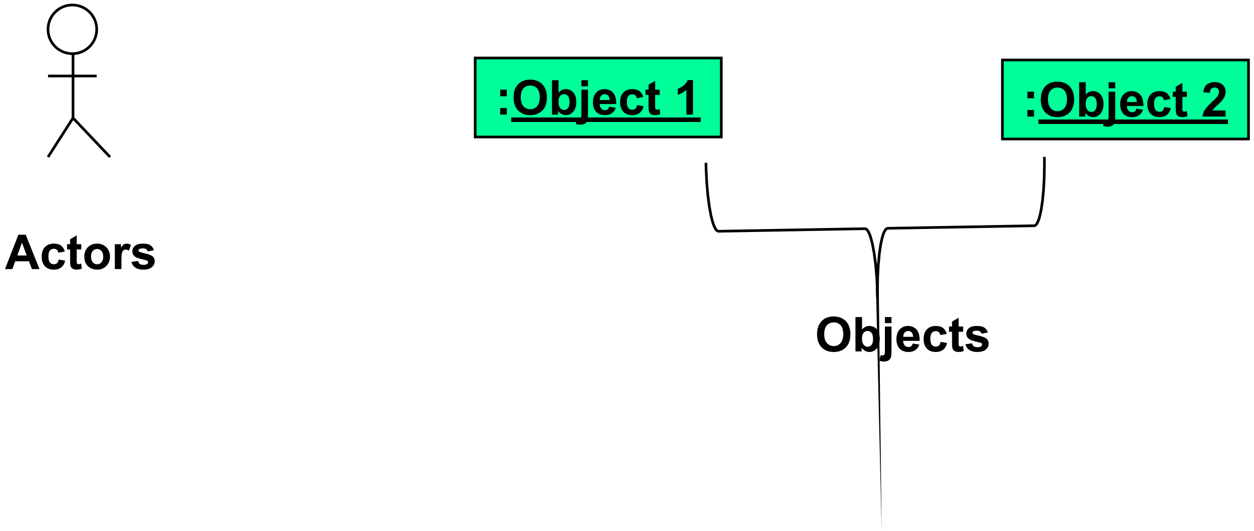
Sometimes a presentation needs to be rescheduled when this happens the availability of the existing instructor needs to be checked. If they are available they are assigned to the presentation on the new date. If not we need to release the current instructor, find all other qualified instructors and check their availability to identify a replacement.

System Boundary

Developing Sequence Diagrams

- Identify the relevant objects involved in the computation
- Establish the role of each object
- Identify the controller
- Identify the collaborators
- Decide on the messages between objects

Sequence diagram notation



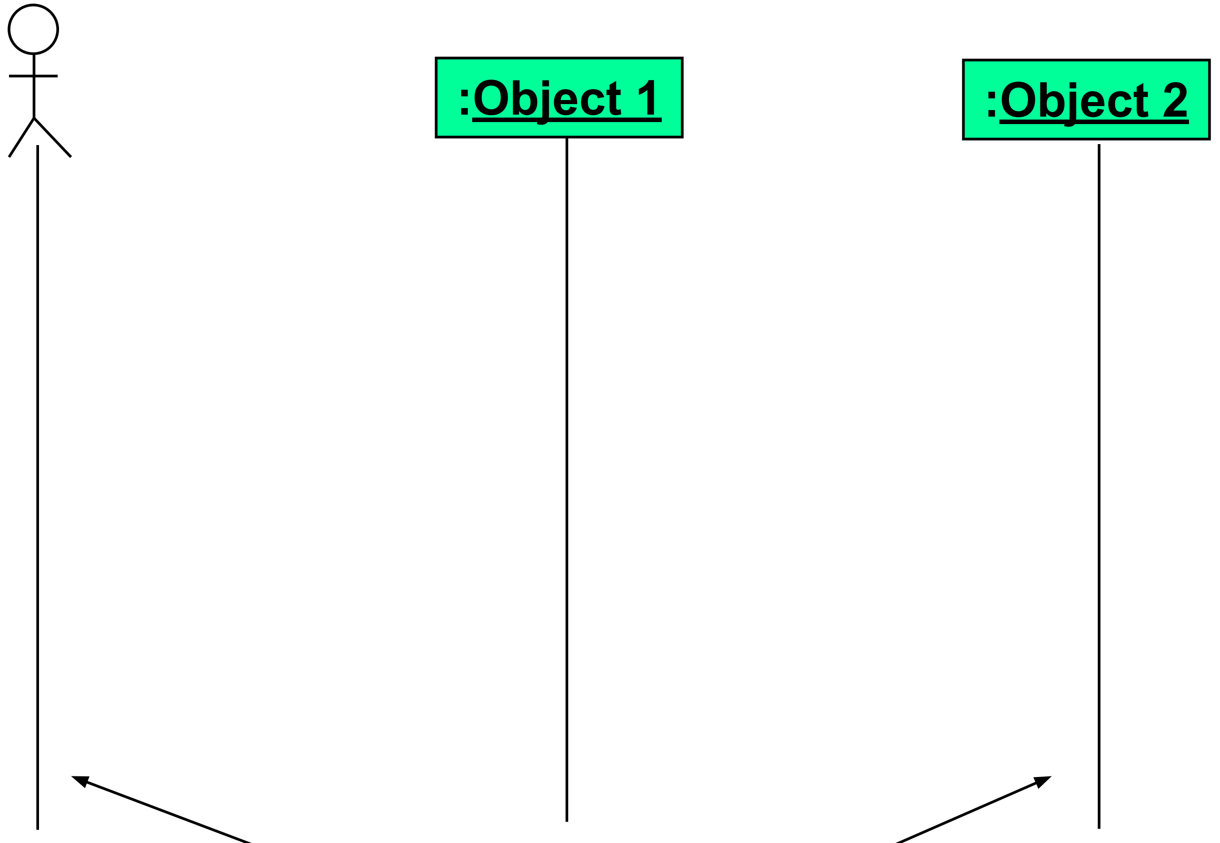
Actors

:Object 1

:Object 2

Objects

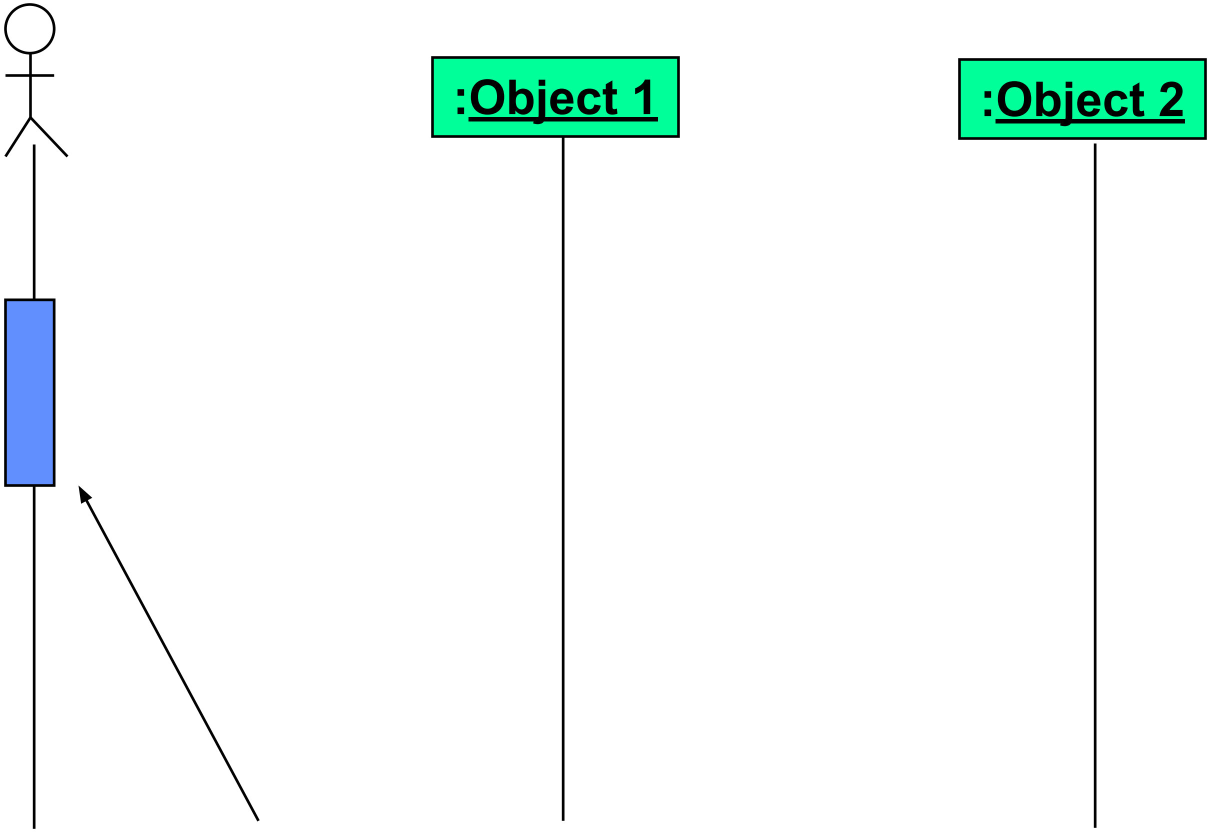
Sequence diagram notation



Lifelines

Identify the existence of the object over time.

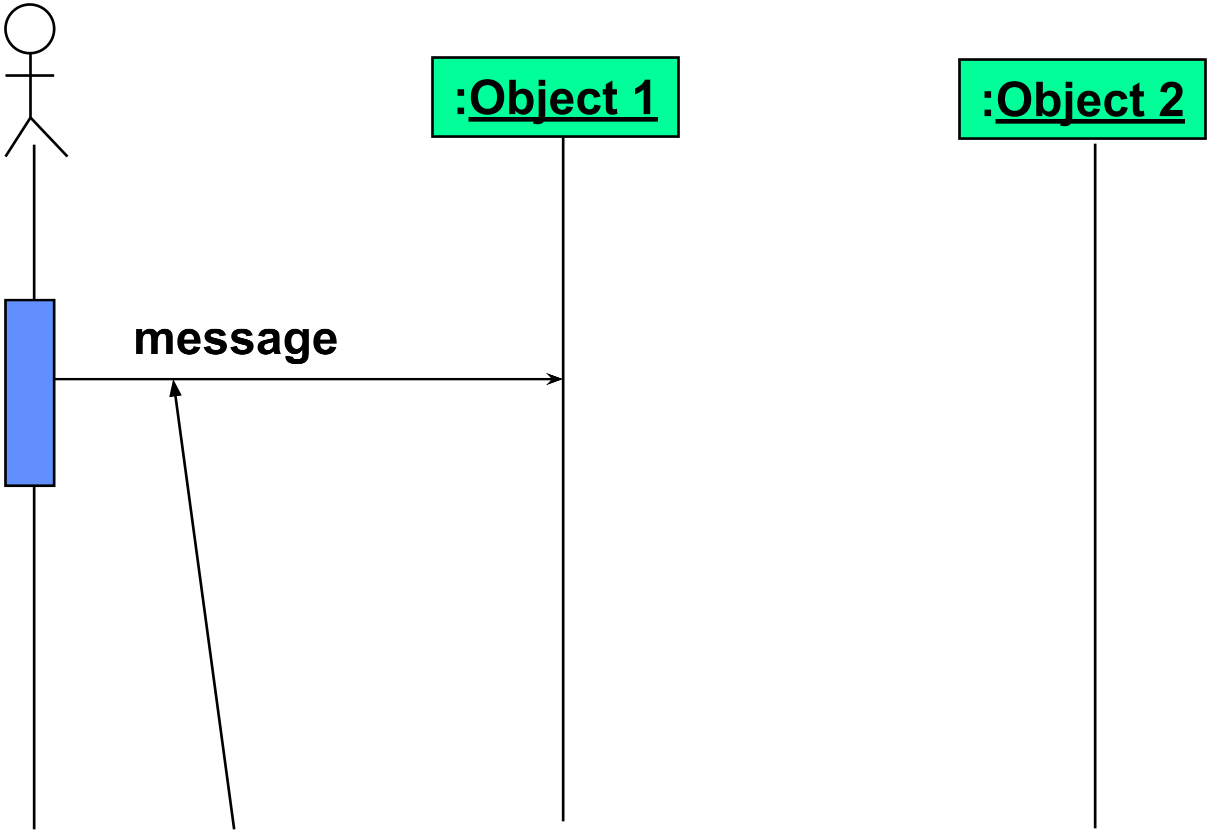
Sequence diagram notation



Activations

Indicate when an object is performing an action

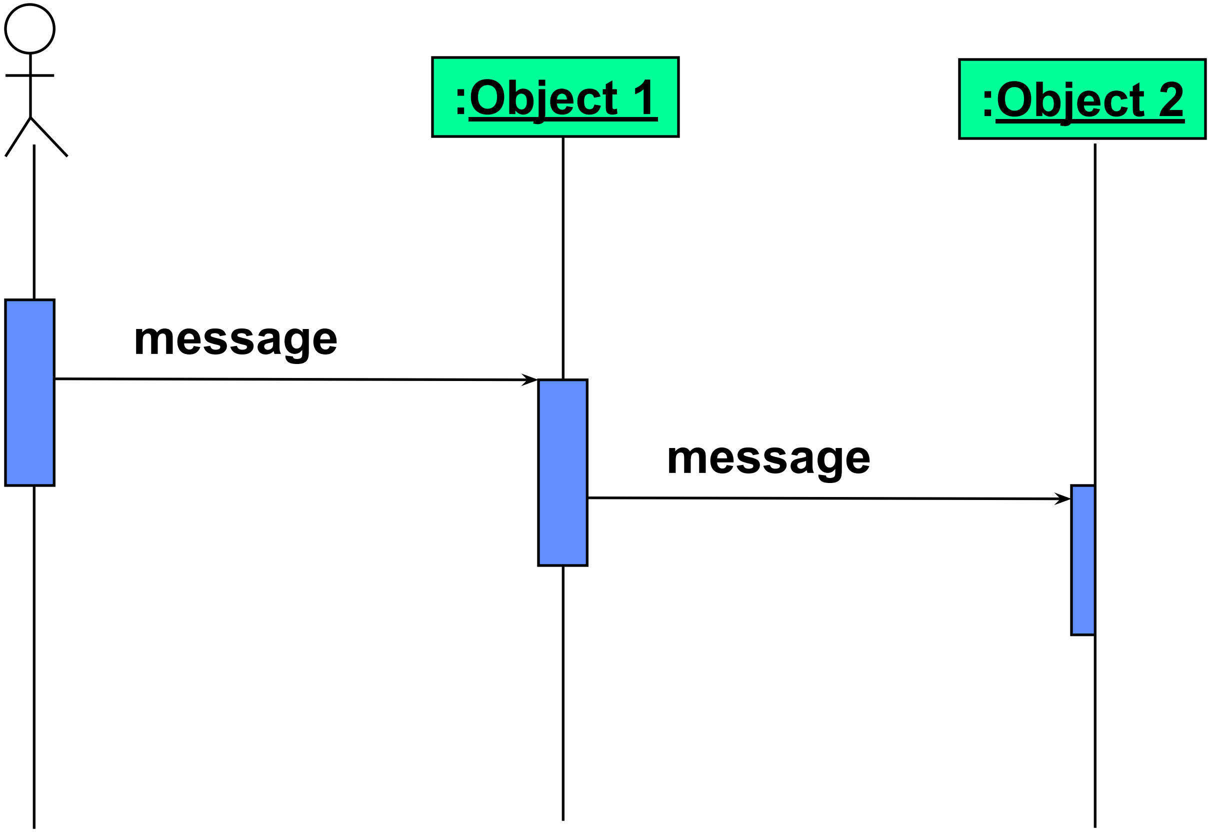
Sequence diagram notation



Messages

Indicate the communications between objects

Sequence diagram notation

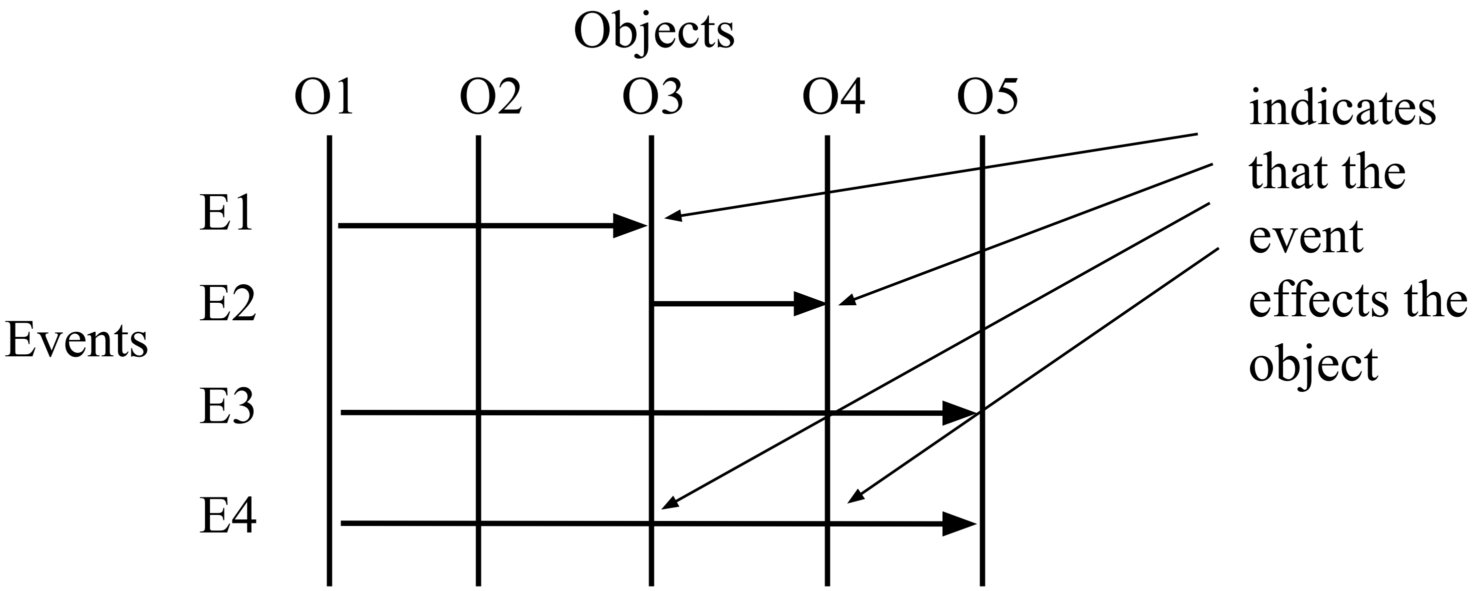


Sequence

Vertical position signifies sequence – earlier messages appear nearer the top.

Sequence Diagram

- Tracks a sequence of events in a scenario
- Identifies all objects involved



Sequence modelling

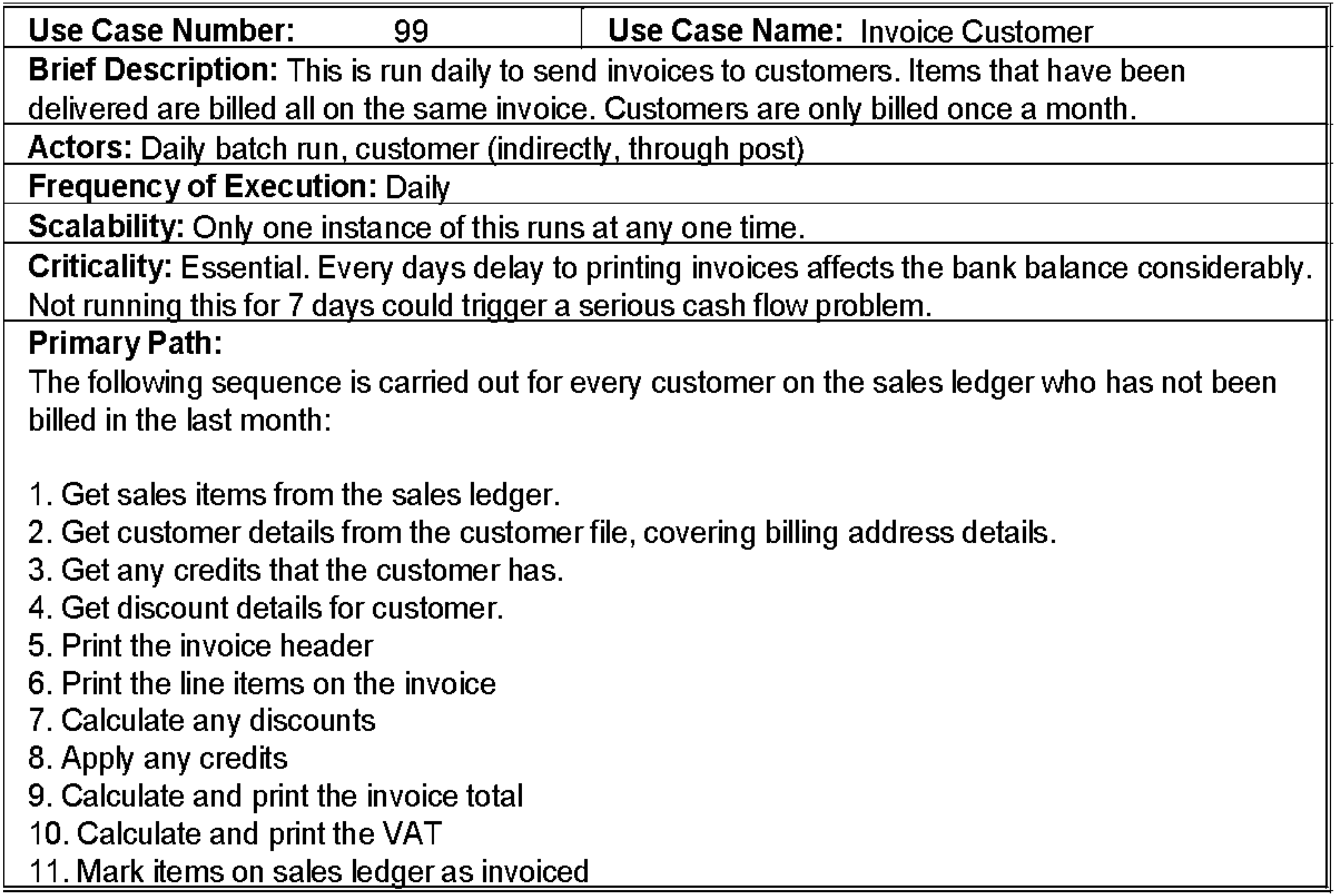
For each event, ask “what objects does this involve?”

- Used to identify new classes
- Determines how classes interact

Use case elaboration

- We define use cases as sequences - primary and alternative paths
- Now we take sample sequences and build sequence diagrams
- This gives us the objects
- And it gives us the relationships
- And it gives us the operations

Invoicing use case (1)



Use Case Number: 99	**Use Case Name:** Invoice Customer

Brief Description: This is run daily to send invoices to customers. Items that have been delivered are billed all on the same invoice. Customers are only billed once a month.

Actors: Daily batch run, customer (indirectly, through post)

Frequency of Execution: Daily

Scalability: Only one instance of this runs at any one time.

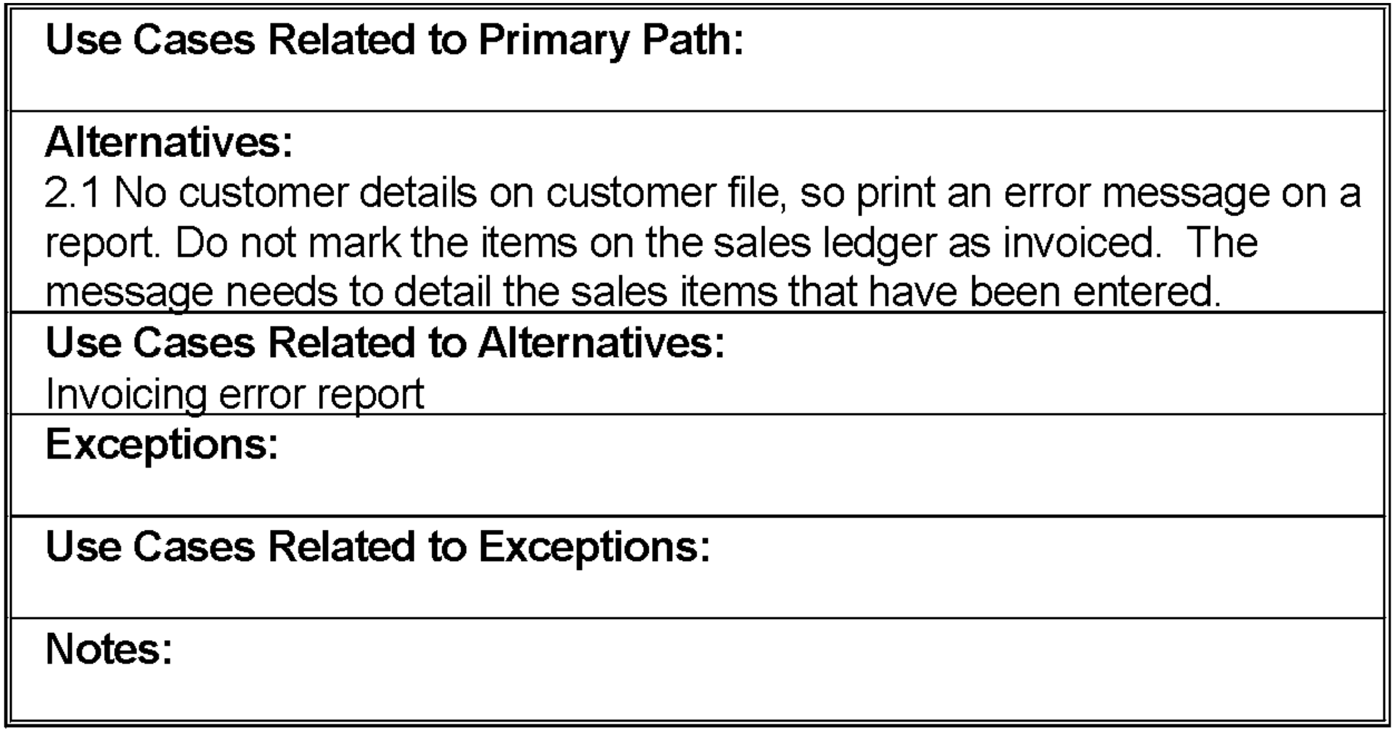
Criticality: Essential. Every days delay to printing invoices affects the bank balance considerably. Not running this for 7 days could trigger a serious cash flow problem.

Primary Path:

The following sequence is carried out for every customer on the sales ledger who has not been billed in the last month:

1. Get sales items from the sales ledger.
2. Get customer details from the customer file, covering billing address details.
3. Get any credits that the customer has.
4. Get discount details for customer.
5. Print the invoice header
6. Print the line items on the invoice
7. Calculate any discounts
8. Apply any credits
9. Calculate and print the invoice total
10. Calculate and print the VAT
11. Mark items on sales ledger as invoiced

Invoicing use case (2)



Use Cases Related to Primary Path:
Alternatives: 2.1 No customer details on customer file, so print an error message on a report. Do not mark the items on the sales ledger as invoiced. The message needs to detail the sales items that have been entered.
Use Cases Related to Alternatives: Invoicing error report
Exceptions:
Use Cases Related to Exceptions:
Notes:

The Primary Path

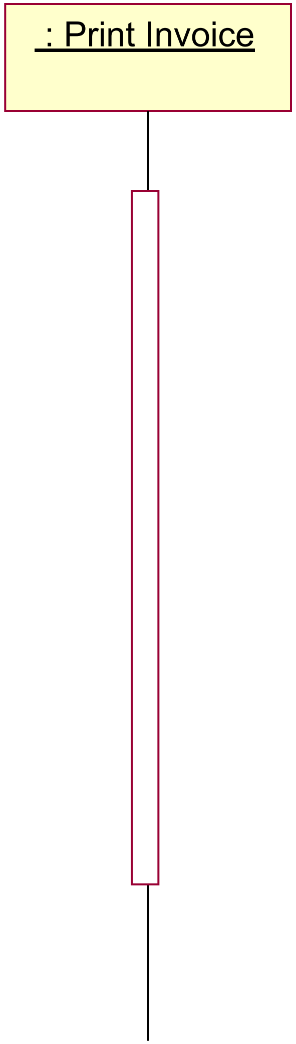
The following sequence is carried out for every customer on the sales ledger who has not been billed in the last month:

1. Get sales items from the sales ledger.
2. Get customer details from the customer file, covering billing address details.
3. Get any credits that the customer has.
4. Get discount details for customer.
5. Print the invoice header
6. Print the line items on the invoice
7. Calculate any discounts
8. Apply any credits
9. Calculate and print the invoice total
10. Calculate and print the VAT
11. Mark items on sales ledger as invoiced

Now we can realise the use case

- Elaborate the scenario with sequence diagrams
- Find objects
- Add operations to objects
- Add attributes to objects

Sequence diagram for Print Invoice use case



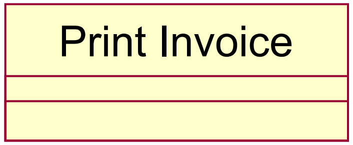
In recent years many OO gurus have suggested that we should introduce a control class for each Use Case.

The control class drives the processing.

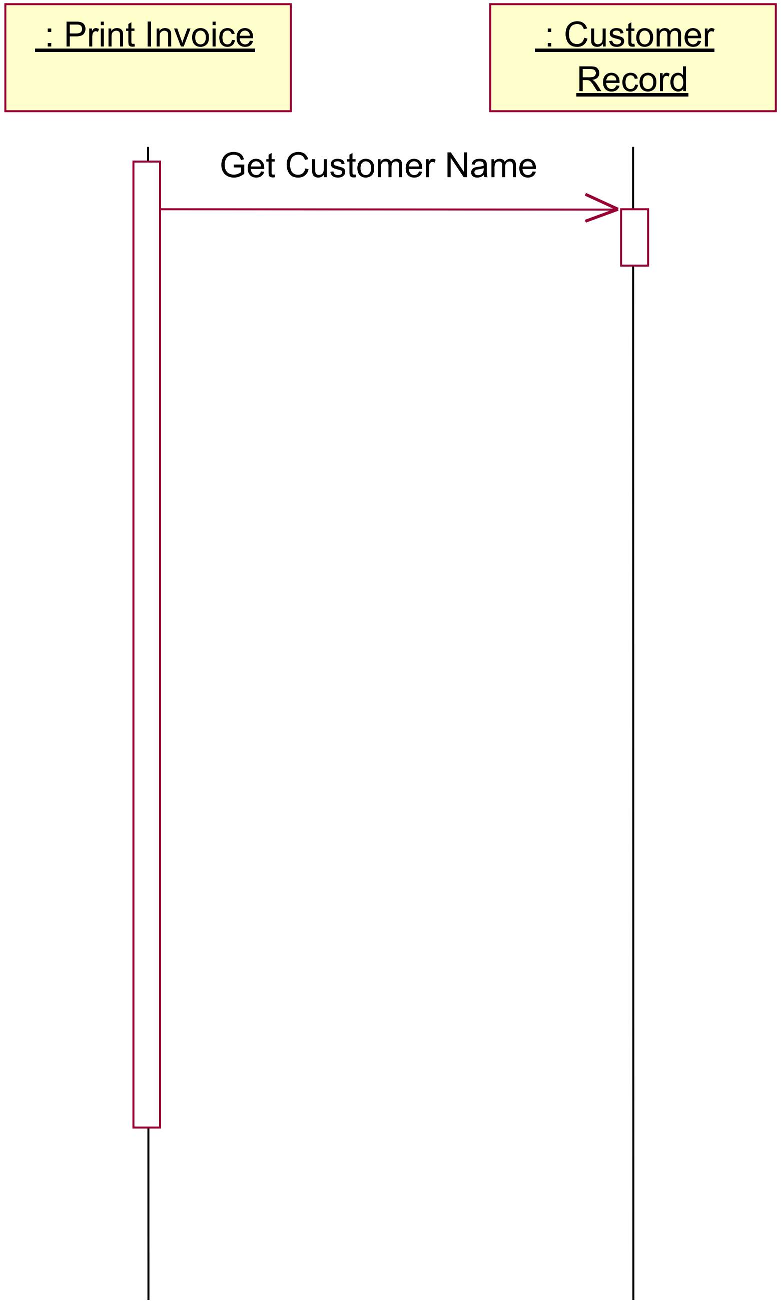
For interactive use cases there is usually a boundary class too.

We can put this object in a class diagram

Print Invoice - class diagram



From our sequence diagram, we find our first object!

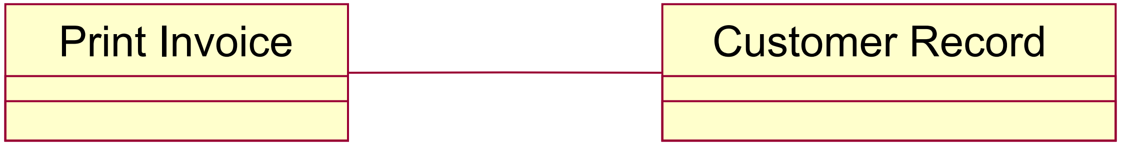


Now we implement the first step of the scenario by getting the Print Invoice control class to send a message.

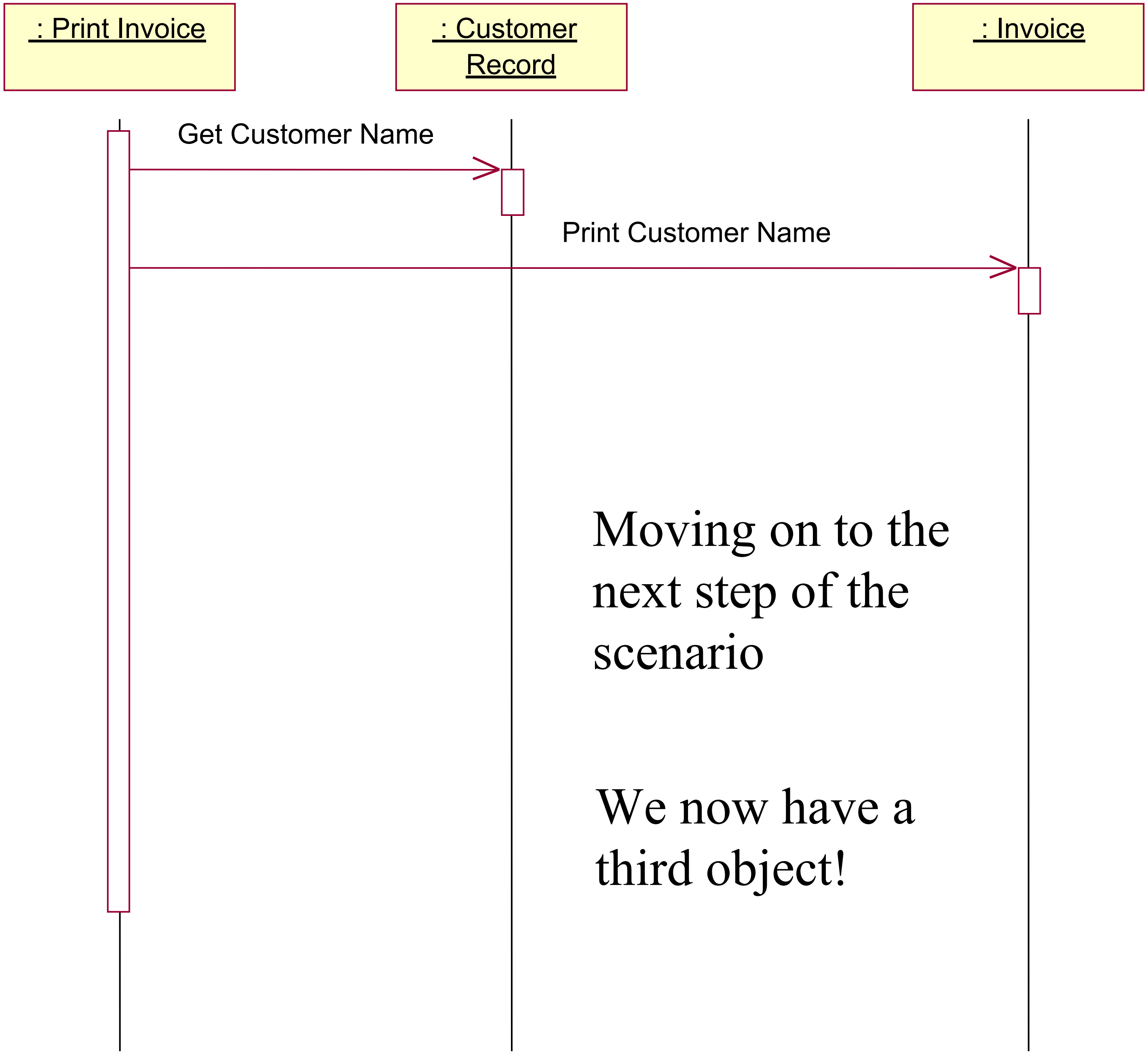
And then we need a recipient of the message.

So we have found another object

Print Invoice - class diagram



We can see that as the objects communicate we need a relationship between them.



: Print Invoice

: Customer Record

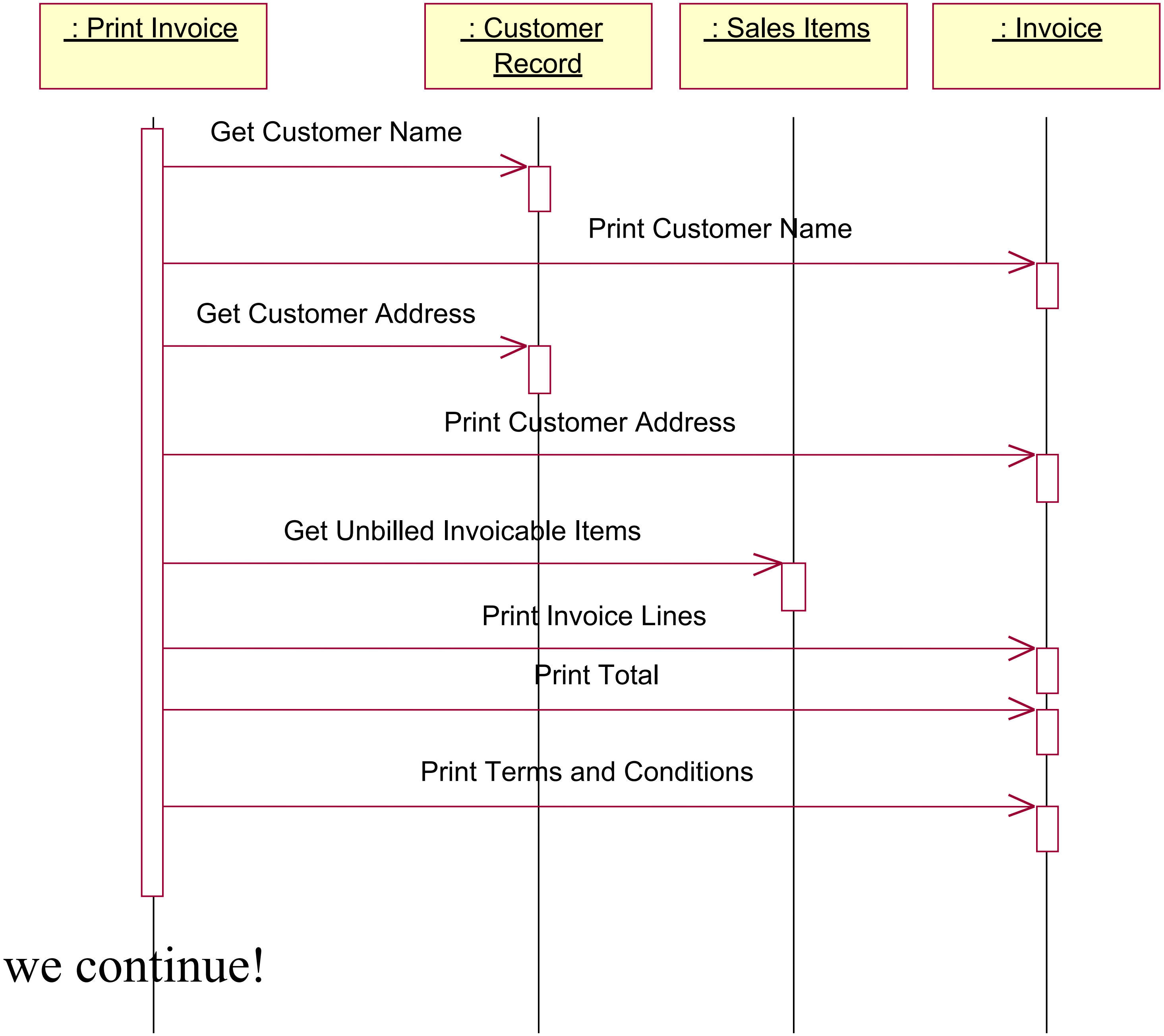
: Invoice

Get Customer Name

Print Customer Name

Moving on to the next step of the scenario

We now have a third object!



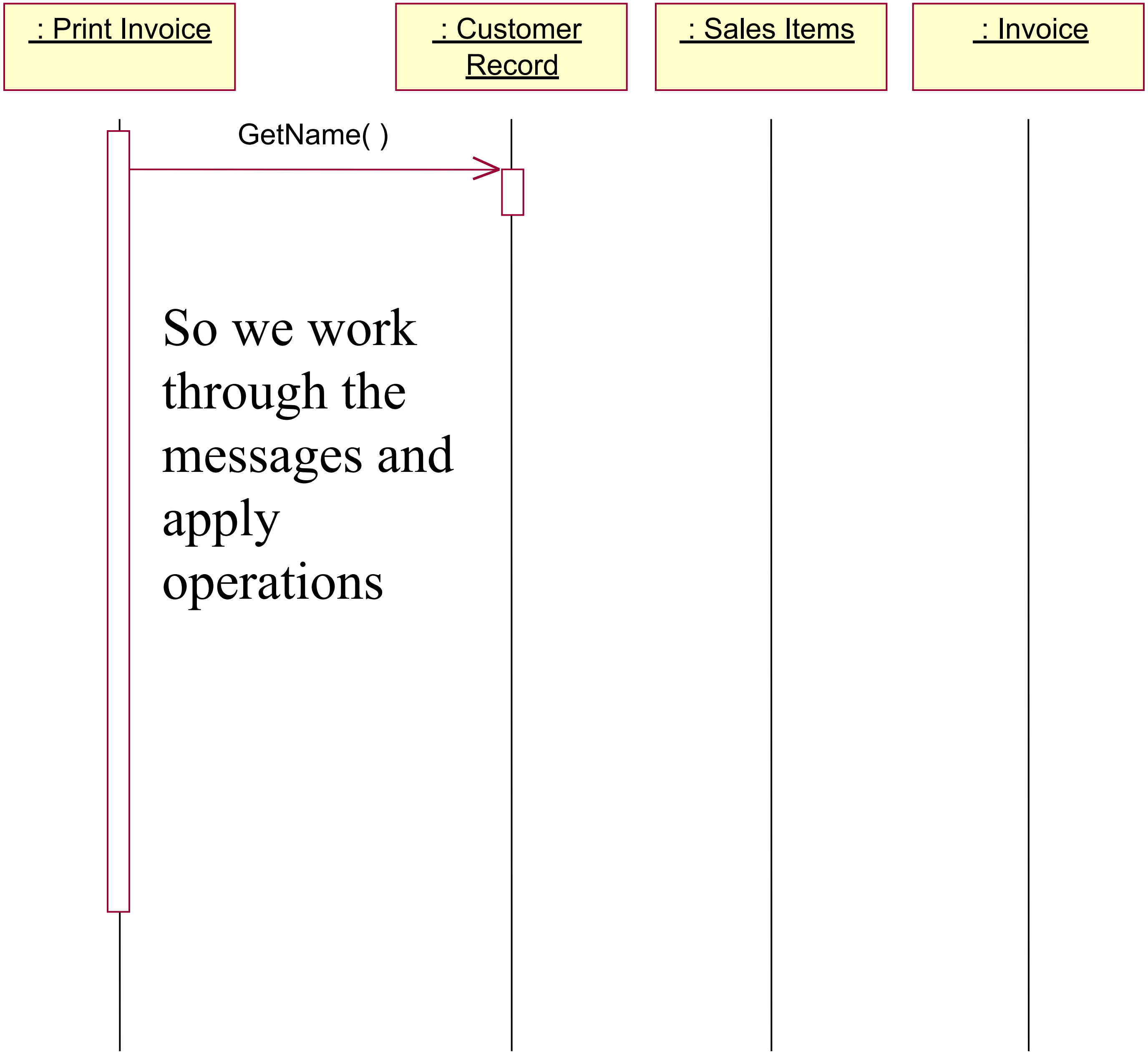
So we continue!

So what have we done?

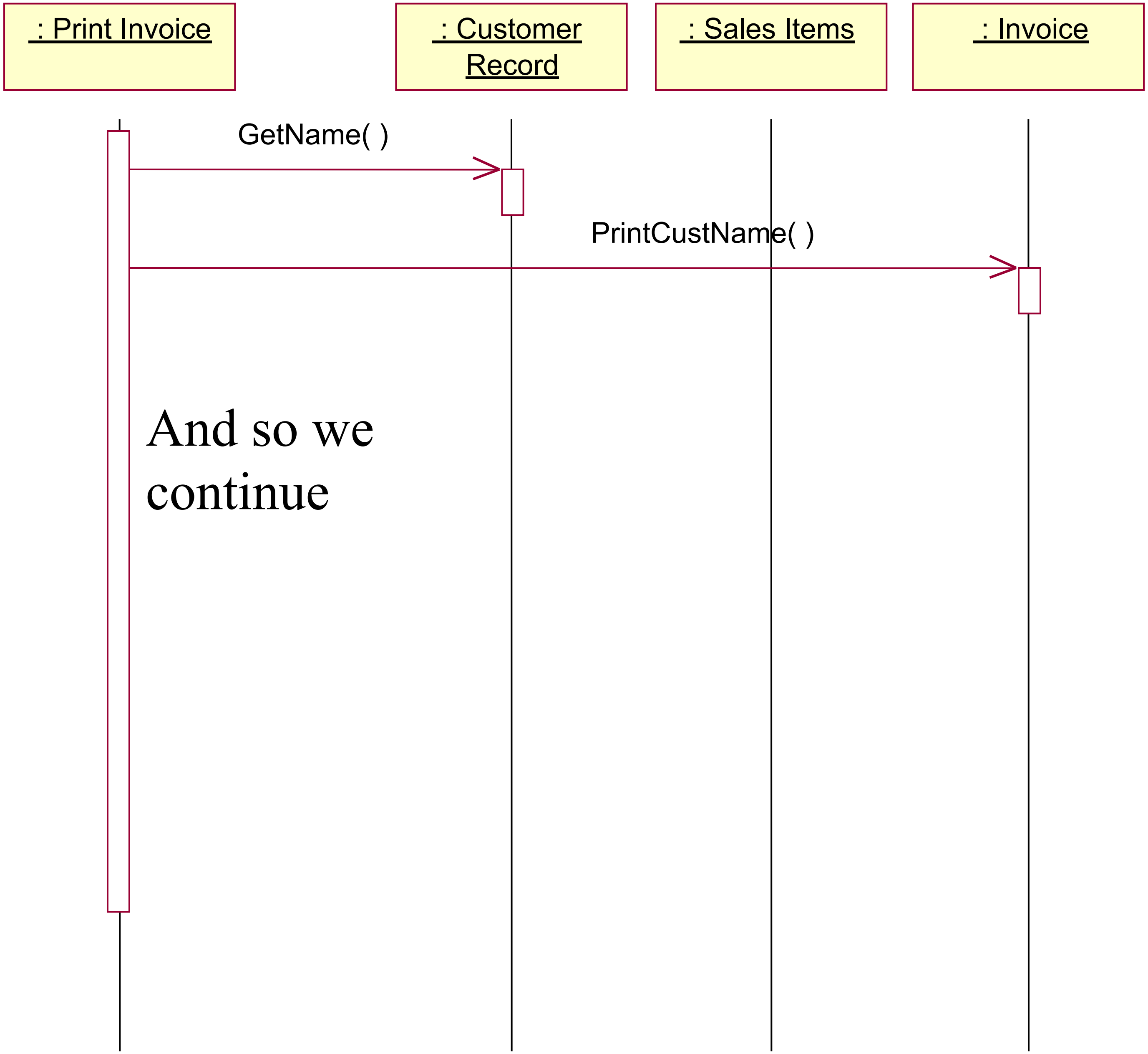
- Worked through a Use Case scenario step by step
- Introduced a controller object to drive things
- Sent messages from one object to another
- Found objects to deal with the messages

What have we got left to do?

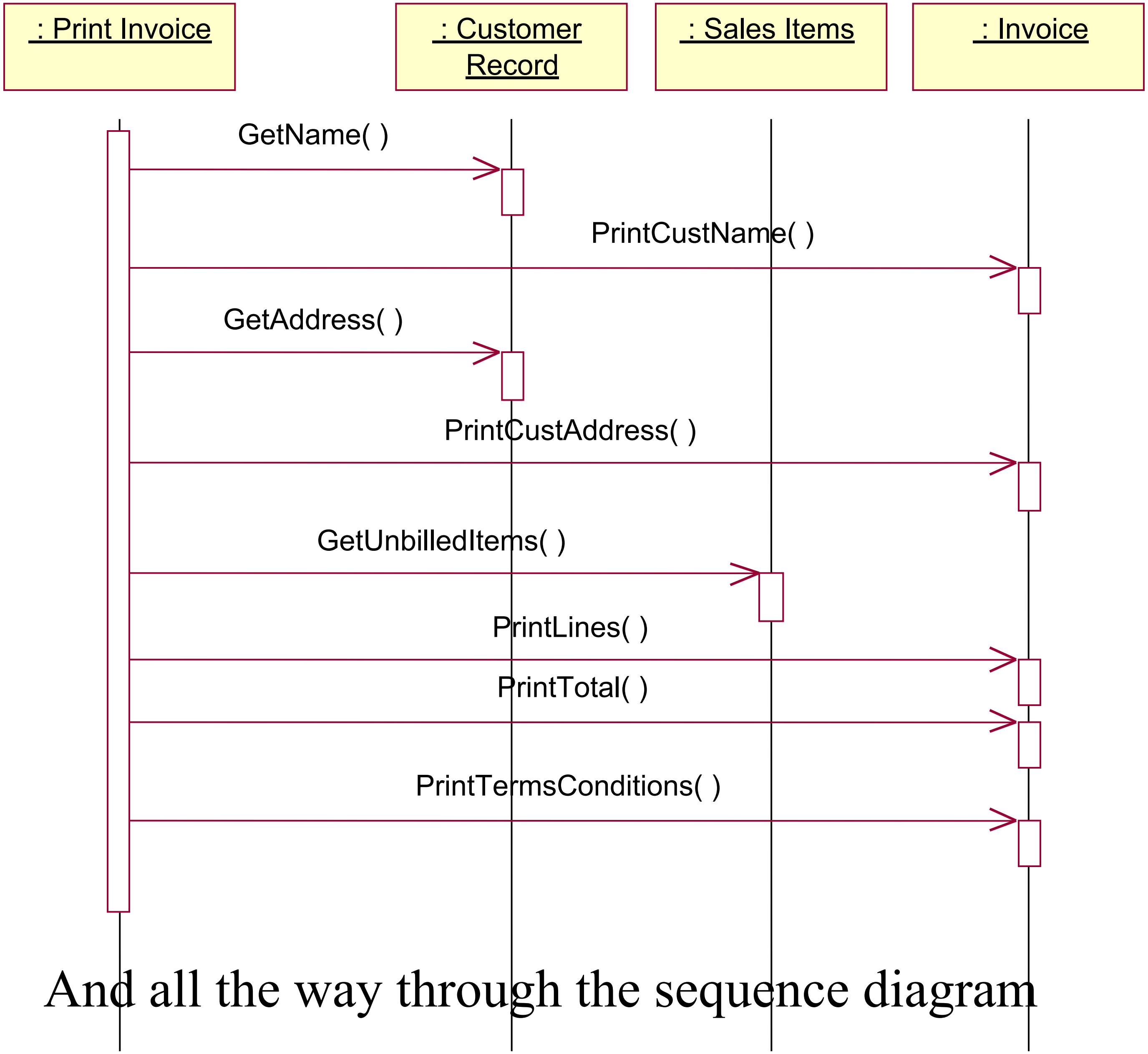
- Find operations on objects to support the messages
- Find attributes to support the objects



So we work through the messages and apply operations

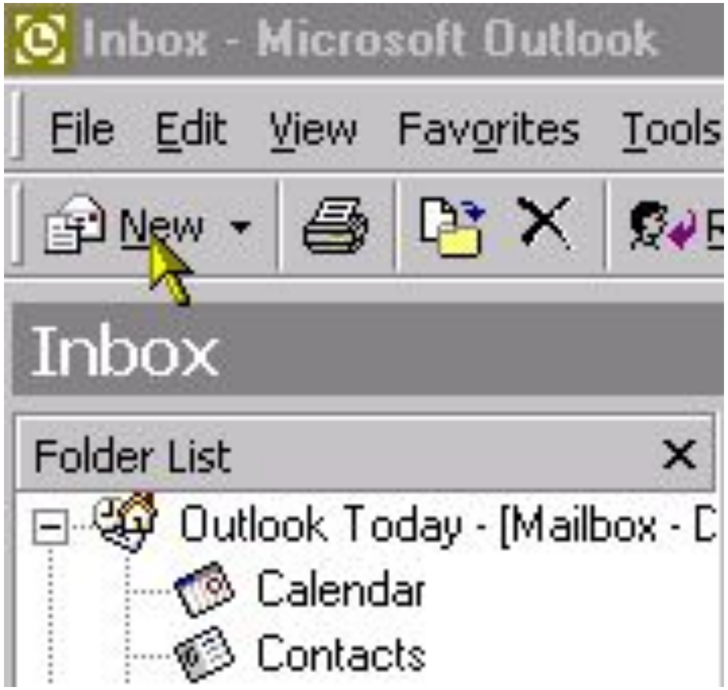


And so we continue

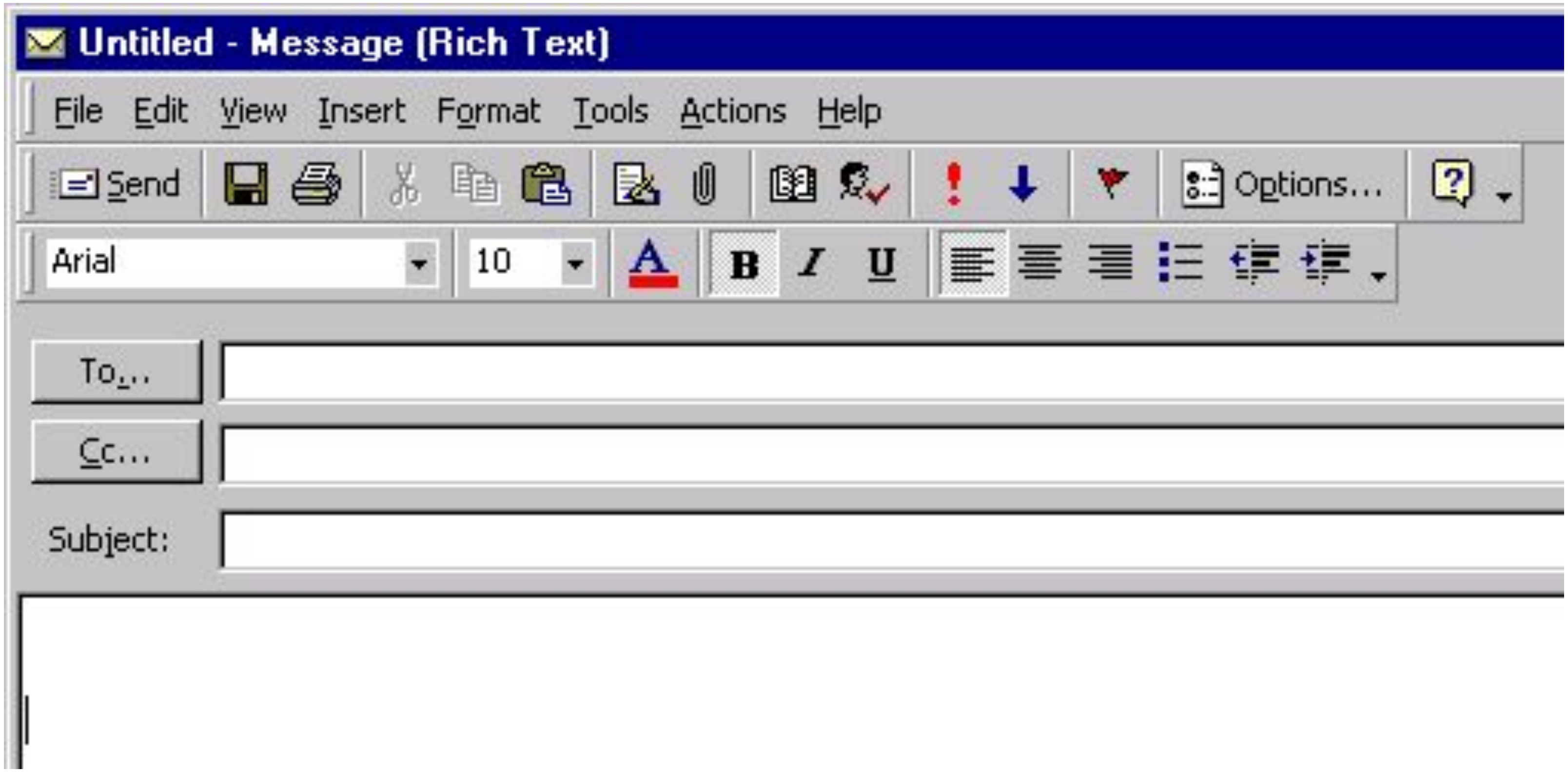


And all the way through the sequence diagram

A Simpler Example - Sending an email



E-mail interface



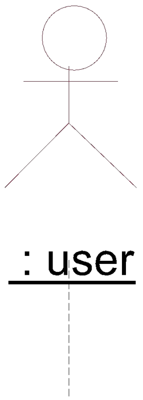
Working from a scenario

Sending an email

1. Press “New ” email icon
2. Enter person’s name in “To” section
3. Type subject
4. Type contents
5. Press Send button
6. System looks up email address in address book
7. System submits the email to the email server

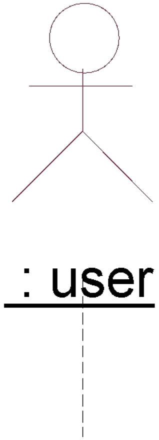
Starting the diagram

- If this is an interactive scenario, we always have an actor driving it, so we put one on the sequence diagram



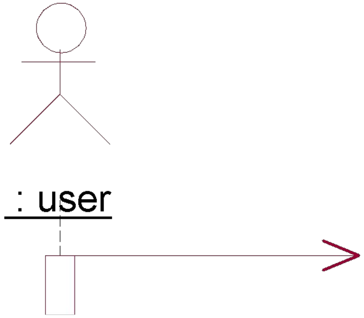
Add objects

The first interaction is with the icon bar, which we can treat as an object

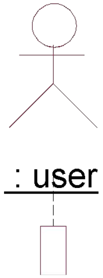


Add message

The user talks to the icon bar



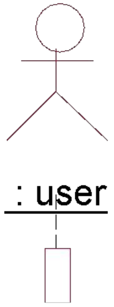
Label the communication



Remember that actors can only communicate with interface objects such as screens, menus and icon bars.

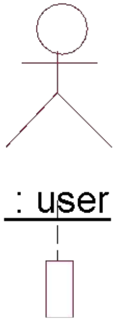
The icon bar has some work to do.

It creates an email page.

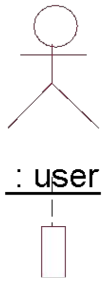


Now the user can see the email page and use it.

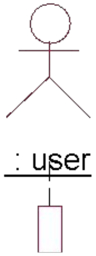
The next three steps are filling in the details on the email page



The user then clicks Send

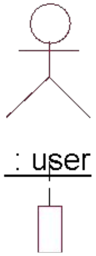


Now consider how to do the sending



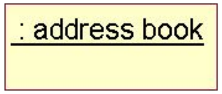
We can choose to get the email page to look up the email address from an address book object

The arrow allows information to return



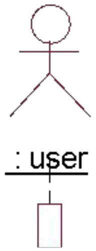
So we don’t need to put a return arrow with the email address going back to the email page

We can choose to get the email page to submit the email to the email server

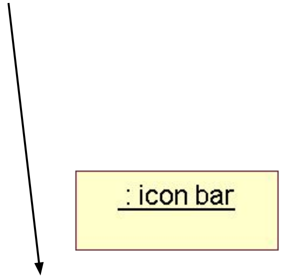


: address book

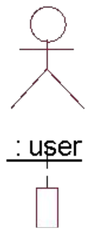
And if we think carefully, the email page always closes after the send.



Now we go through and change the messages to operations on the object



And so on, all the way through



Developing a Sequence Diagram

Work through a scenario step by step

Make actors communicate with screens, icons, menus

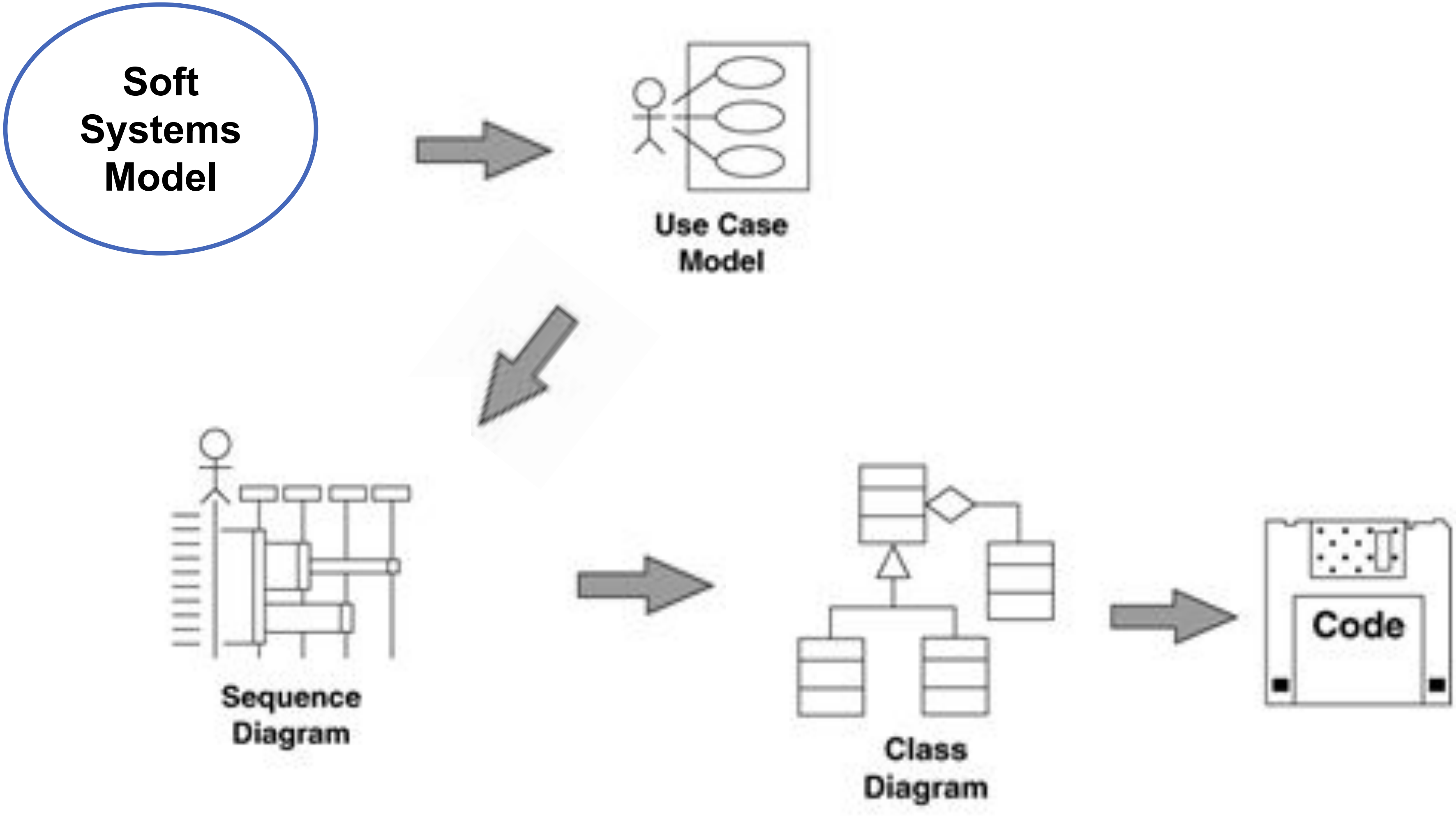
Make the screen actions (etc) trigger actions with objects

Convert the actions to operations

Sequence Diagrams – why bother?

- Tie use cases and object models together
- Use the sequences in use cases
- Identify objects
- Identify relationships
- Identify operations

Beginnings of a Method



Soft Systems Model

Use Case Model

Sequence Diagram

Class Diagram

Code

Sequence Diagrams

- Used to model *object interactions* on a time axis.
  - Dynamic aspects of a system.
  - How objects collaborate to realize a use case.
- Distribute use case behavior to classes.
  - Starting to look at *how* the system does something rather that just what is done.
- For now, high level interactions
  - Look at more detailed level later

Why do sequence diagrams?

- Add detail to use cases.
- Specify how objects collaborate.
- Move closer to design.

Sequence Diagram Example

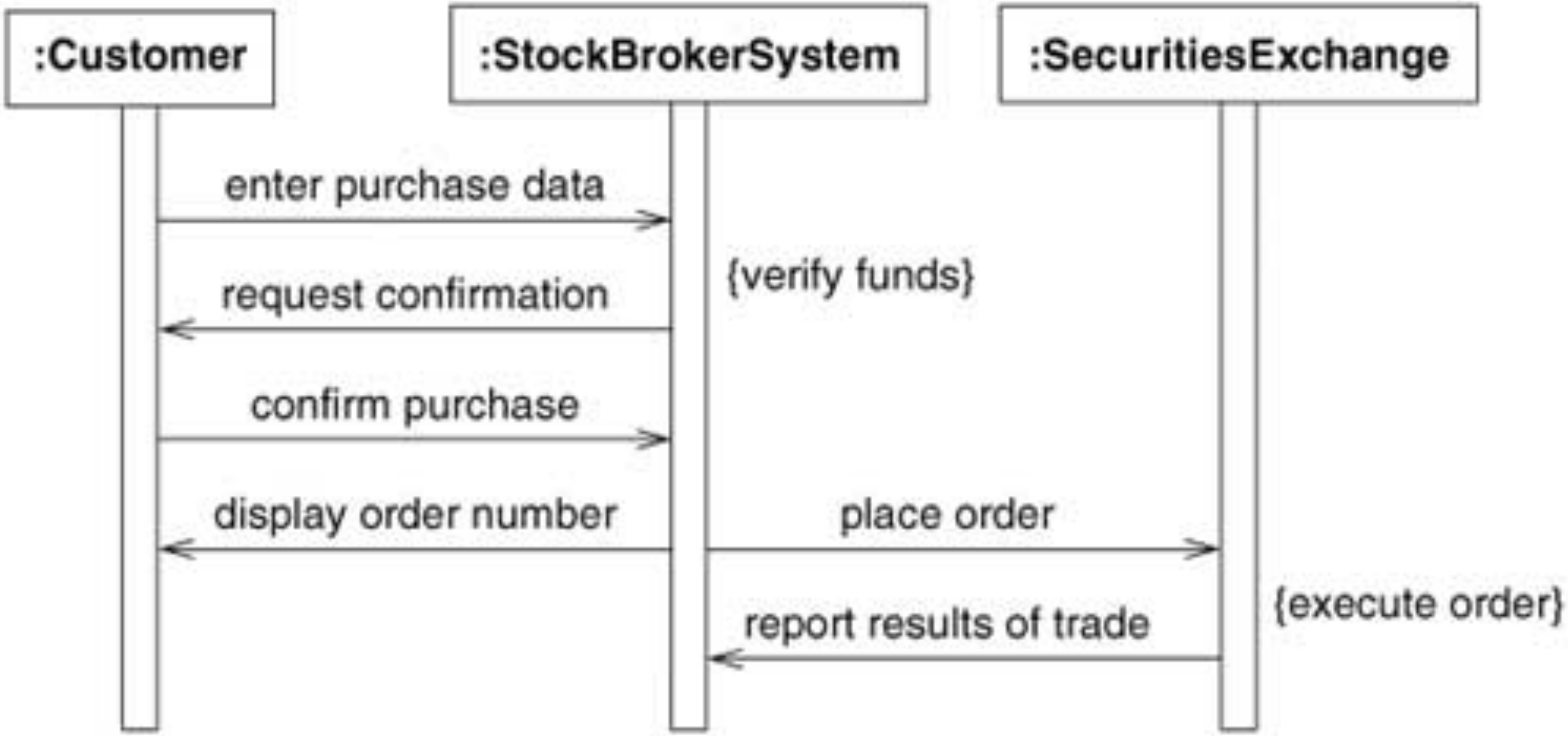


Figure 7.6 Sequence diagram for a stock purchase. Sequence diagrams can show large-scale interactions as well as smaller, constituent tasks.

Object-Oriented Modeling and Design with UML, Second Edition by Michael Blaha and James Rumbaugh. ISBN 0-13-1-015920-4. © 2005 Pearson Education, Inc., Upper Saddle River, NJ. All rights reserved.

Sequence Diagram Example

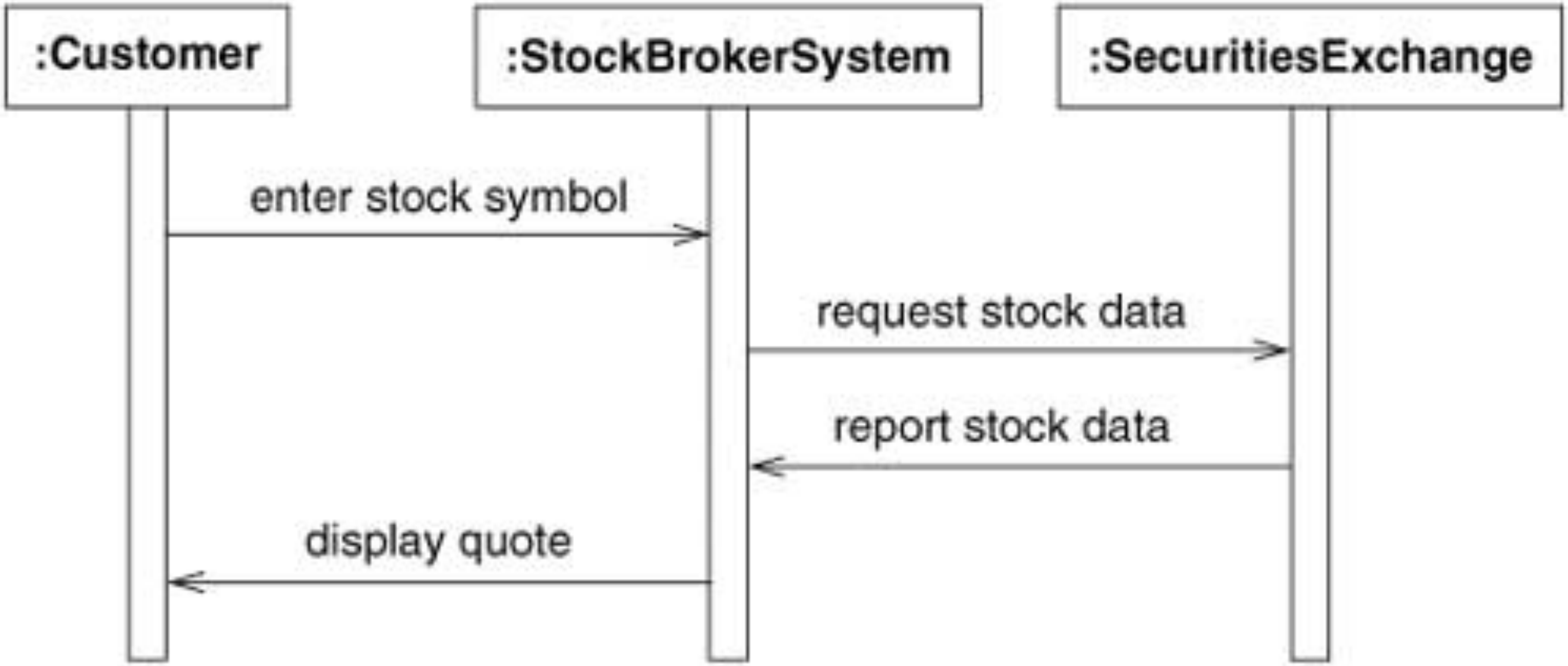


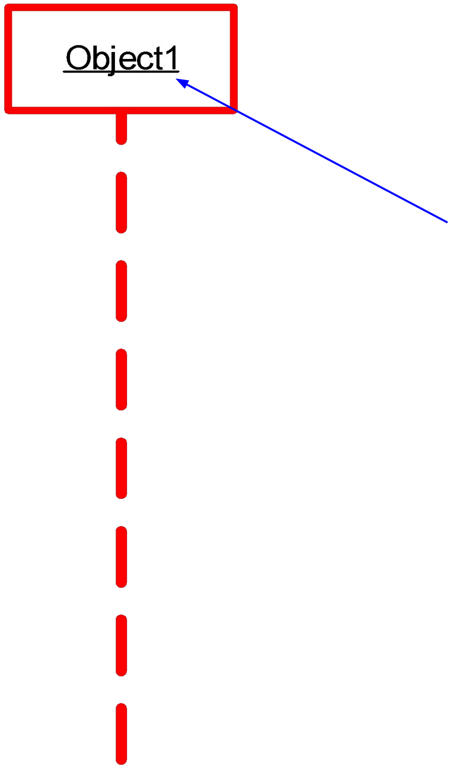
Figure 7.7 Sequence diagram for a stock quote.

Object-Oriented Modeling and Design with UML, Second Edition by Michael Blaha and James Rumbaugh. ISBN 0-13-1-015920-4. © 2005 Pearson Education, Inc., Upper Saddle River, NJ. All rights reserved.

Active Objects

- No significance to position of the active objects on the left to right axis.
- General guidelines:
  - Normally put actor on the left.
  - Add objects left to right as they get involved.
  - Try to minimize crossovers.

Lifelines



Represents time during which the object exists.

May run entire length of diagram or may start or end within the diagram

Underline on name indicates object

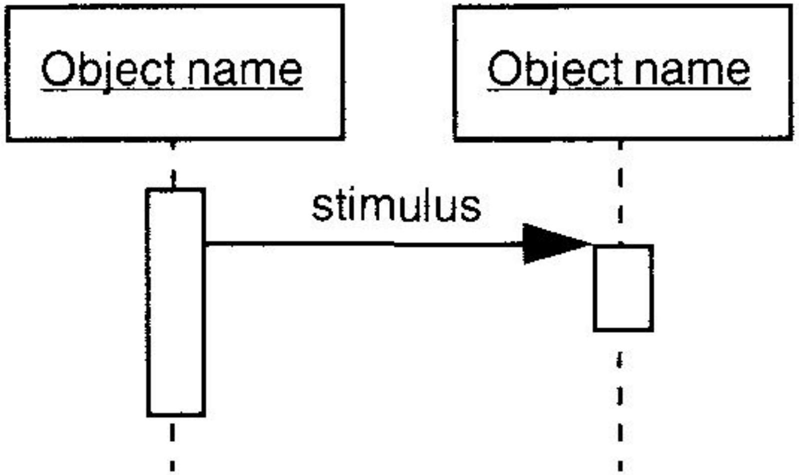
(vs class)

More notation for names is defined in UML. (We will learn as needed.)

Messages

- **Interactions are represented by *messages* sent from one object to another.**
- **Long narrow vertical box on lifeline indicates *focus of control*.**
  - **When an object is active, either because it is doing something, or because it has sent a message to another object that is doing something on its behalf.**
  - **Not always shown.**

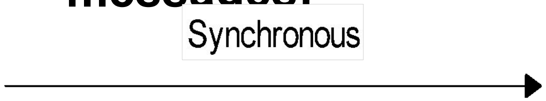
Messages



Arrows are labeled with the name of the message, or stimulus, that they represent.

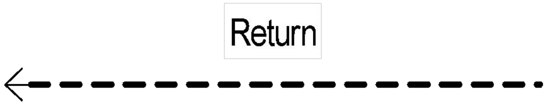
Kinds of Messages

UML defines four kinds of messages:



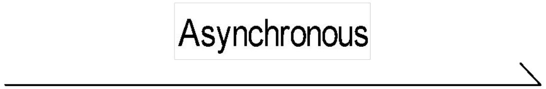
Sender waits for a reply

(*Procedural* message)

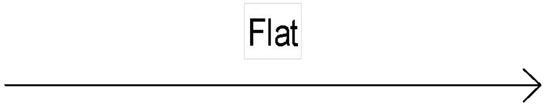


The reply to a synchronous message

Often not shown

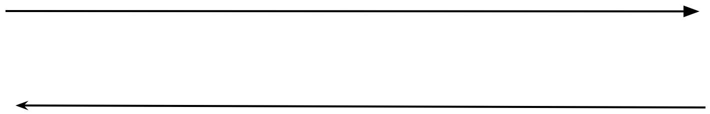


Sender does not wait for a reply



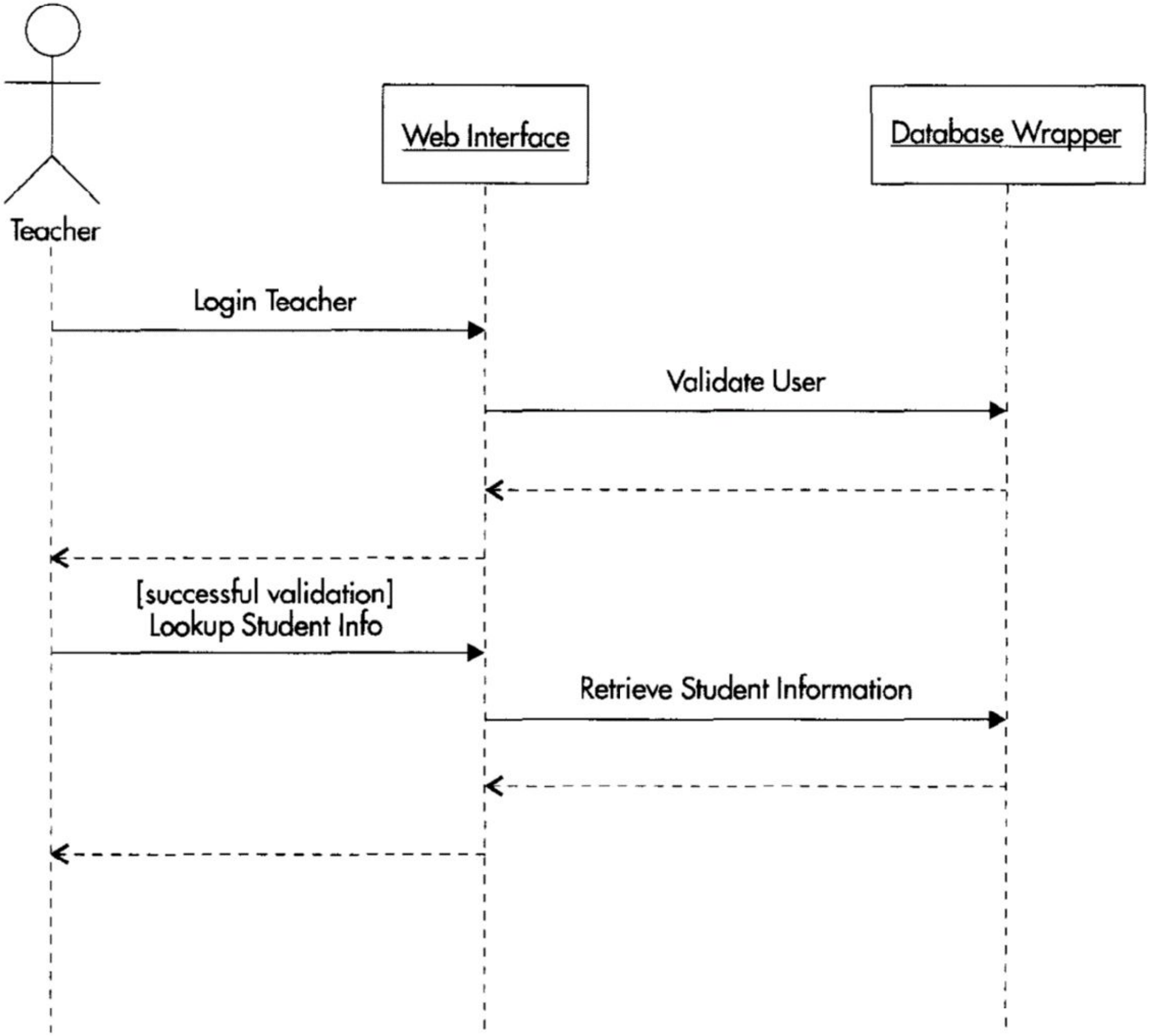
Don’t know or don’t care

Synchronous Messages

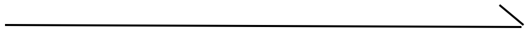


- Used when we want to things to be done one at a time.
- Like procedure calls in program.
- Sender waits for response to message before doing anything more.

Synchronous Message Example



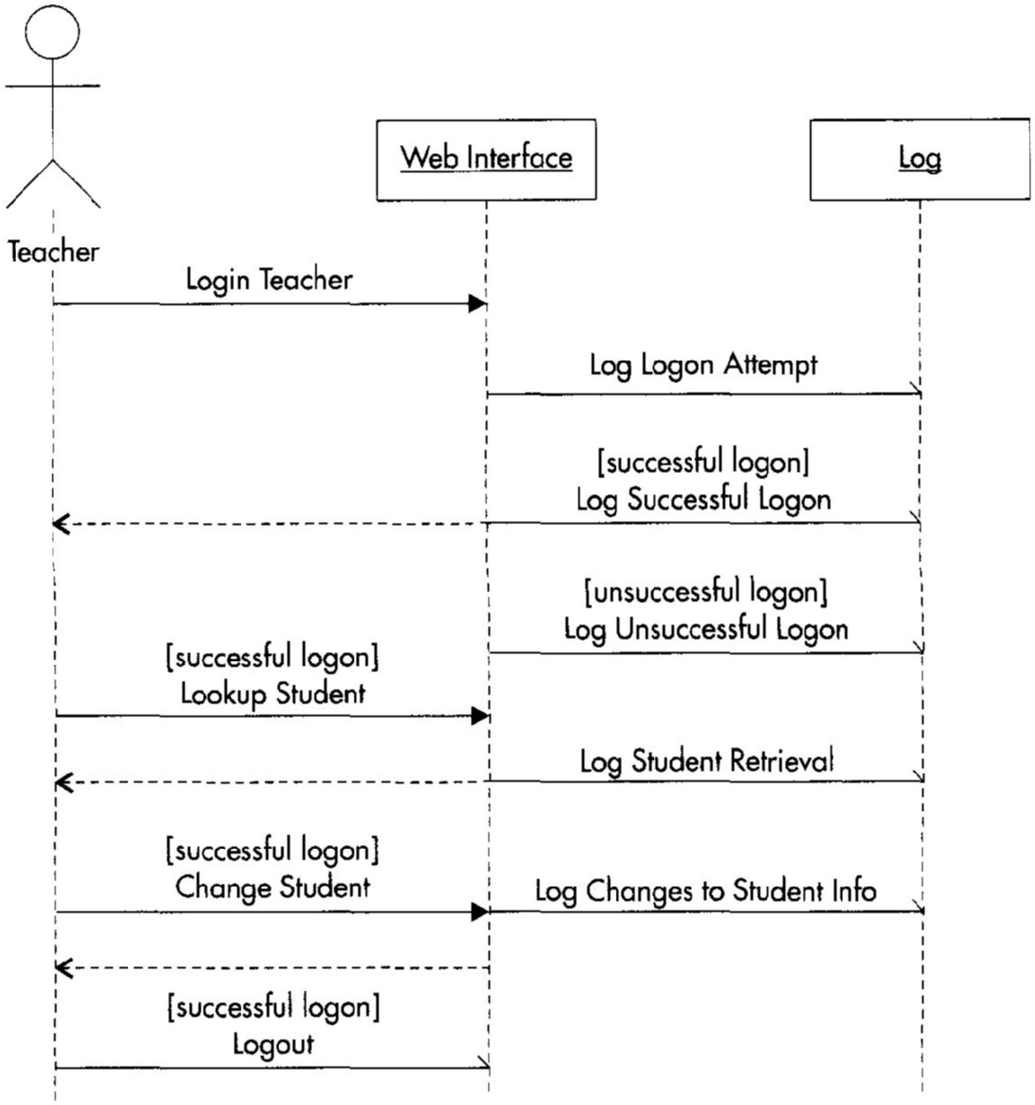
Asynchronous Message



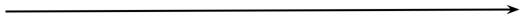
- **Used when we don’t want the sender to wait for a response**
  - **Typically a one way message**
  - **No response is sent.**

  - **Response may invoke a *callback* method.**

Asynchronous Message Example

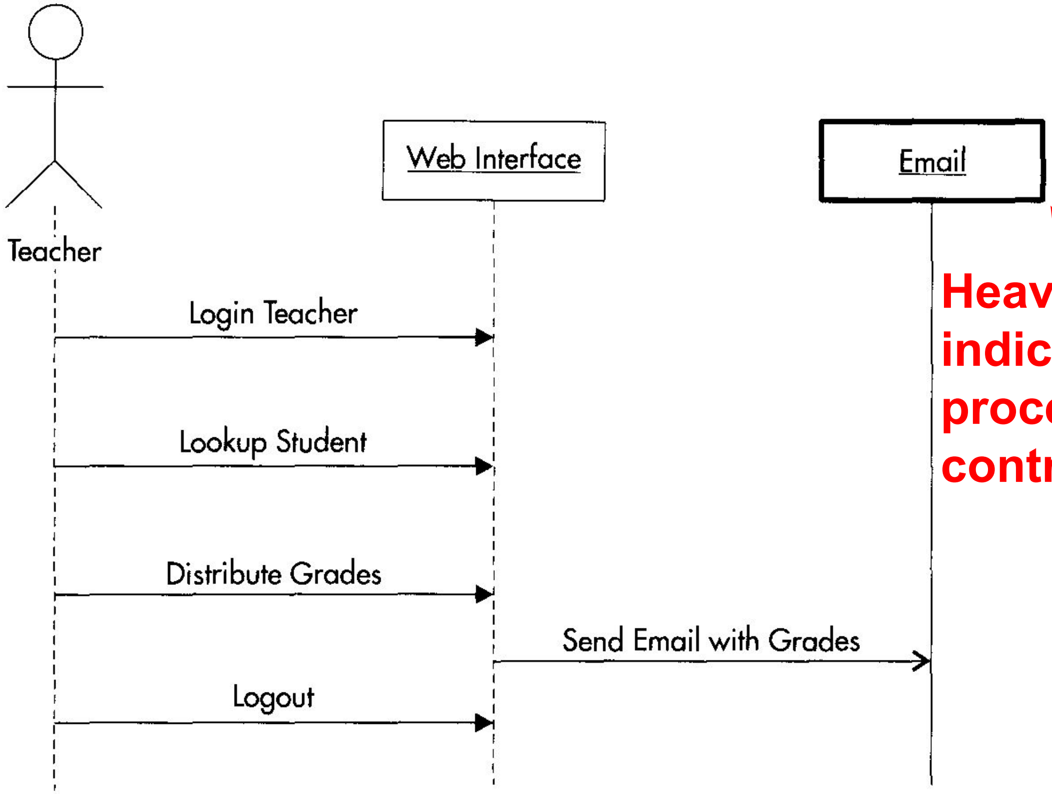


Flat Messages



- Used when we don’t want to specify whether or not sender waits for a reponse.
  - Haven’t decided yet.
  - Isn’t important
  - Specifically want to leave as an implementation decision.

Flat Message Example



Heavy border indicates separate process with own control flow.

Web Interface sends email message

Some wait for a response. Some do not.

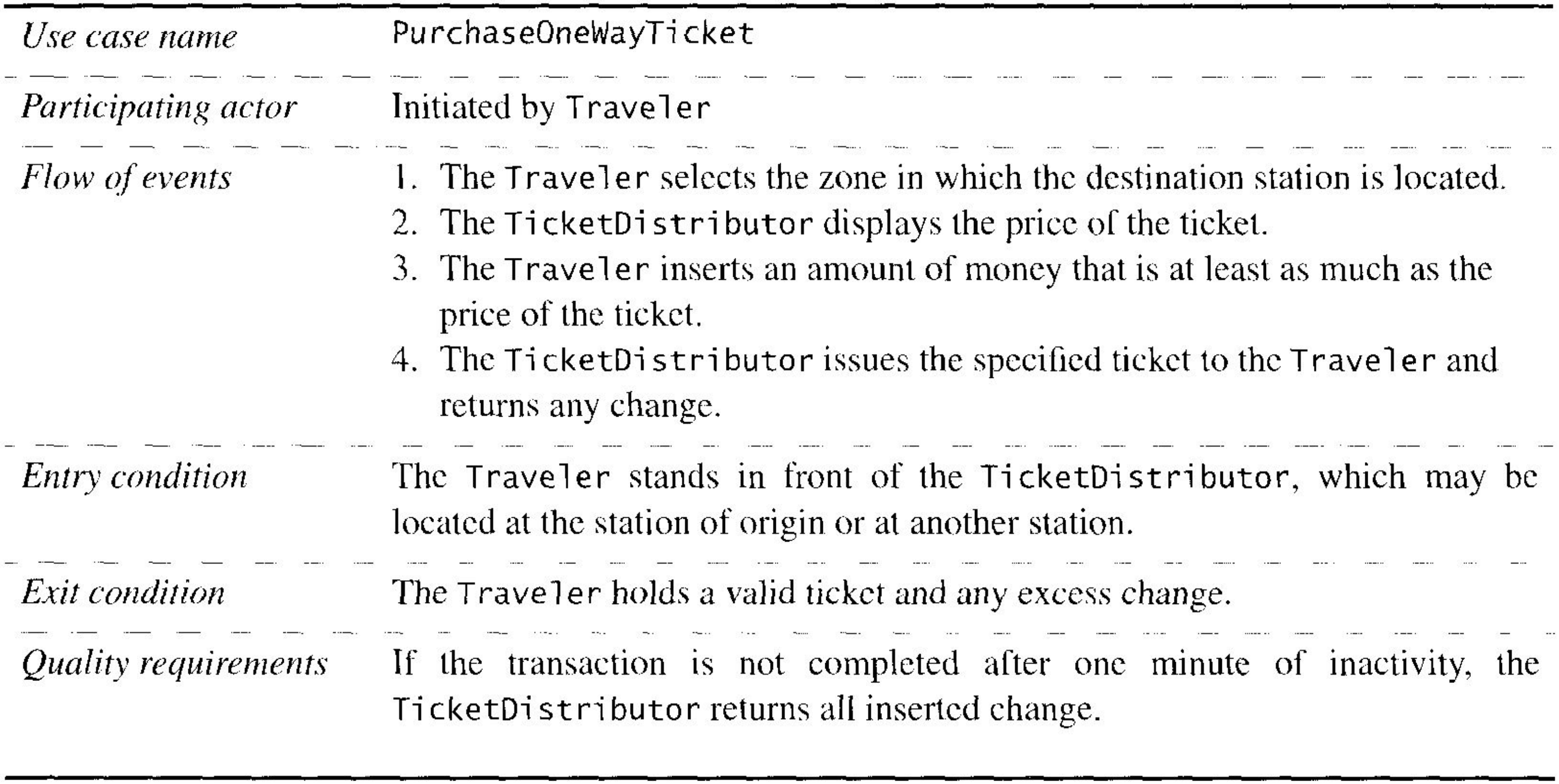
Sequence Diagram vs Activity Diagram

- **When do I use a sequence diagram and when do I use an activity diagram?**
  - **How do I decide which one is appropriate?**
- **Ans: First of all, you don't have to choose. You can do both.**
- **Depends on what you want to show.**
- **Activity diagrams focuses on the sequence of actions.**
  - **Doesn't show *why* an object does something.**
- **Sequence diagrams show "flow of information"**

  (Who says what to whom).

Example: TicketDistributor

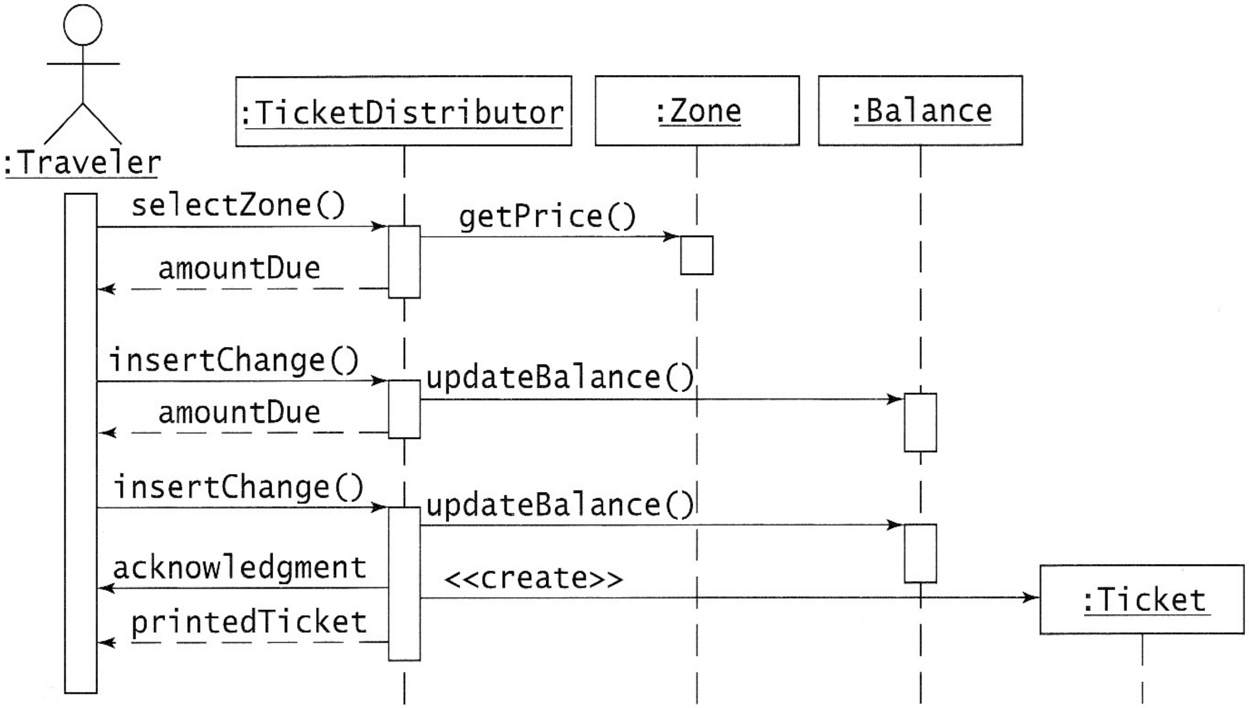
Use Case



Use case name	PurchaseOneWayTicket
Participating actor	Initiated by Traveler
Flow of events	1. The Traveler selects the zone in which the destination station is located. 2. The TicketDistributor displays the price of the ticket. 3. The Traveler inserts an amount of money that is at least as much as the price of the ticket. 4. The TicketDistributor issues the specified ticket to the Traveler and returns any change.
Entry condition	The Traveler stands in front of the TicketDistributor, which may be located at the station of origin or at another station.
Exit condition	The Traveler holds a valid ticket and any excess change.
Quality requirements	If the transaction is not completed after one minute of inactivity, the TicketDistributor returns all inserted change.

From *Object Oriented Software Engineering Using UML, Patterns, and Java*, by Bernd Bruegge and Allen H. Dutoit, Prentice Hall/Pearson, 2004

A Dynamic Model of TicketDistributor



From *Object Oriented Software Engineering Using UML, Patterns, and Java*, by Bernd Bruegge and Allen H. Dutoit, Prentice Hall/Pearson, 2004

Class Activity

- Draw a sequence chart for the vending machine use case “Customer purchases soft drink with credit card.”

- Active objects:
  - Customer
  - Vending Machine
  - Credit Card Processing Center

- Use case description on next slide.

Vending Machine Use Case

Use case name Customer purchases soft drink with credit card

Participating actor Initiated by Customer
Credit Card Processing Center

Flow of events

1. The customer swipes his credit card.
2. Vending machine sends message to processing center.
3. Processing center confirms card and provides available credit.
4. Vending machine indicates that customer can select product.
5. Customer presses button to select product.
6. Vending machine sends charge to processing center.
7. Processing center confirms charge
8. Vending machine dispences selected product.
9. Customer removes product.

Entry condition The customer stands in front of a soft drink vending machine that accepts credit cards.

Customer has a valid credit card and wants to purchase a soft drink using it.

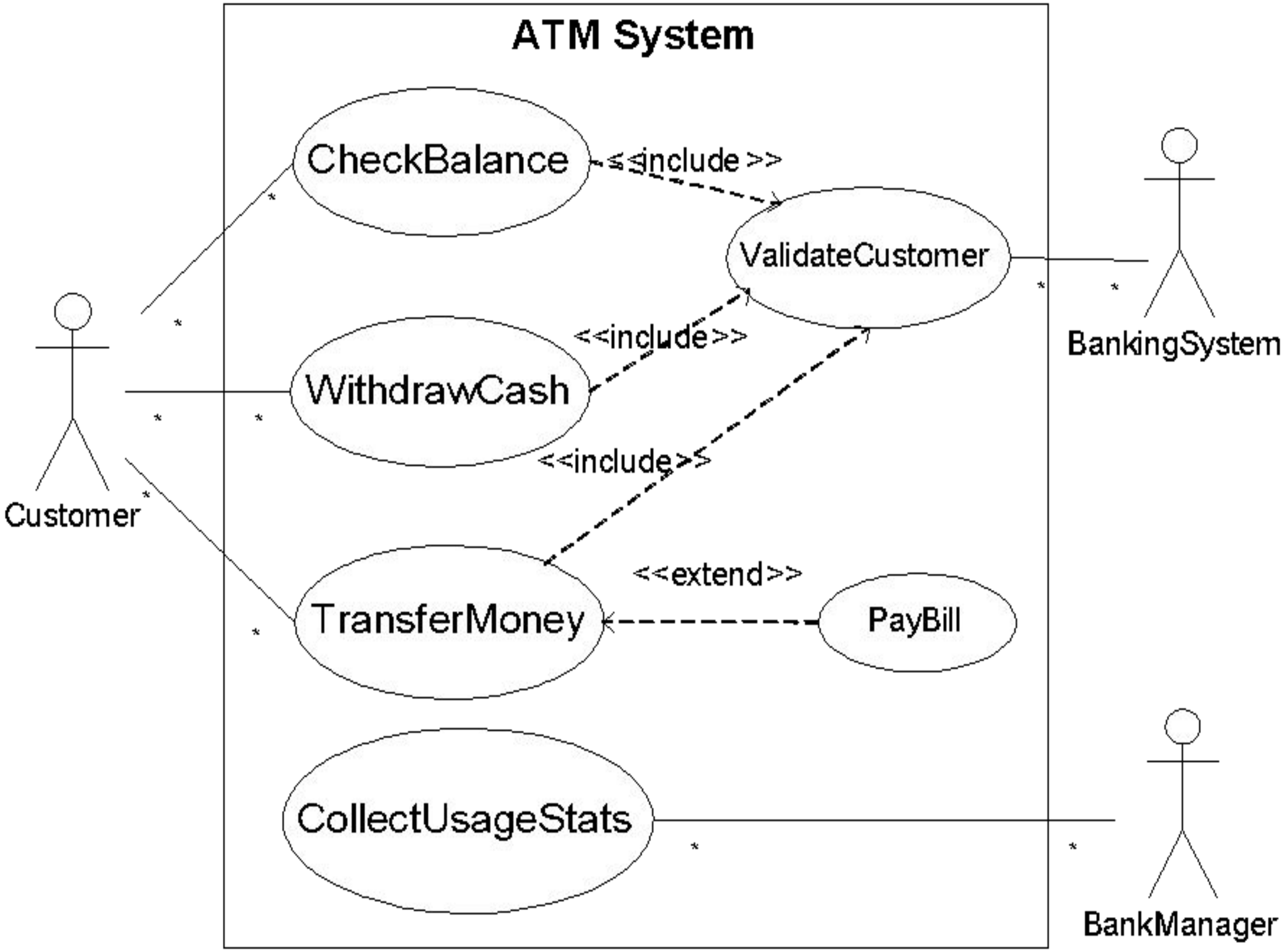
Exit condition Customer has soft drink. Credit card is charged.

Class Activity

- Draw a sequence chart for the ATM machine use cases

- Use cases descriptions on next slide.

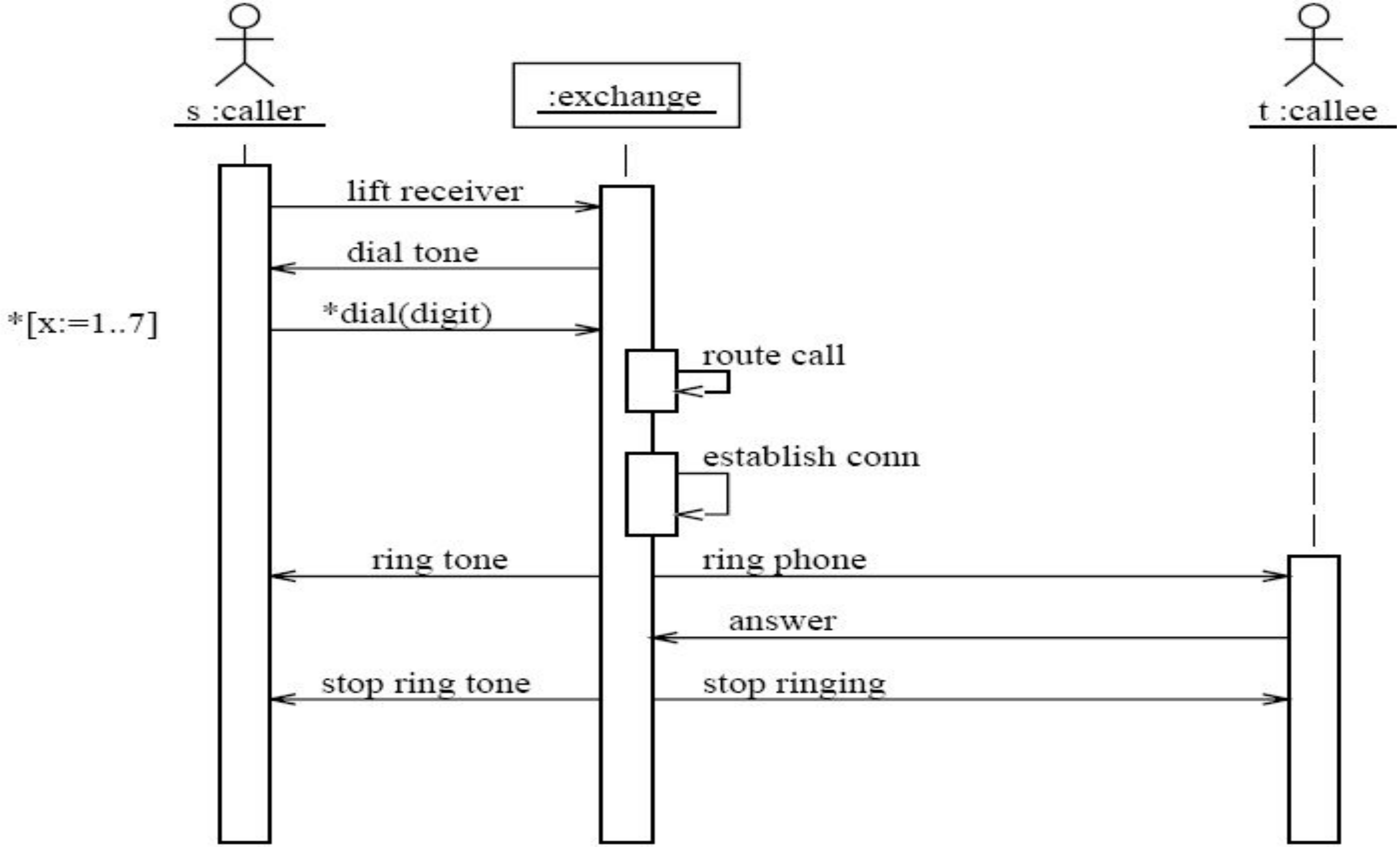
Elaborated Use Case Diagram for ATM



Use Case Diagrams

- **Describes a set of sequences.**
- **Each sequence represents the interactions of things outside the system (*actors*) with the system itself (and key abstractions)**
- **Use cases represent the functional requirements of the system (non-functional requirements must be given elsewhere)**

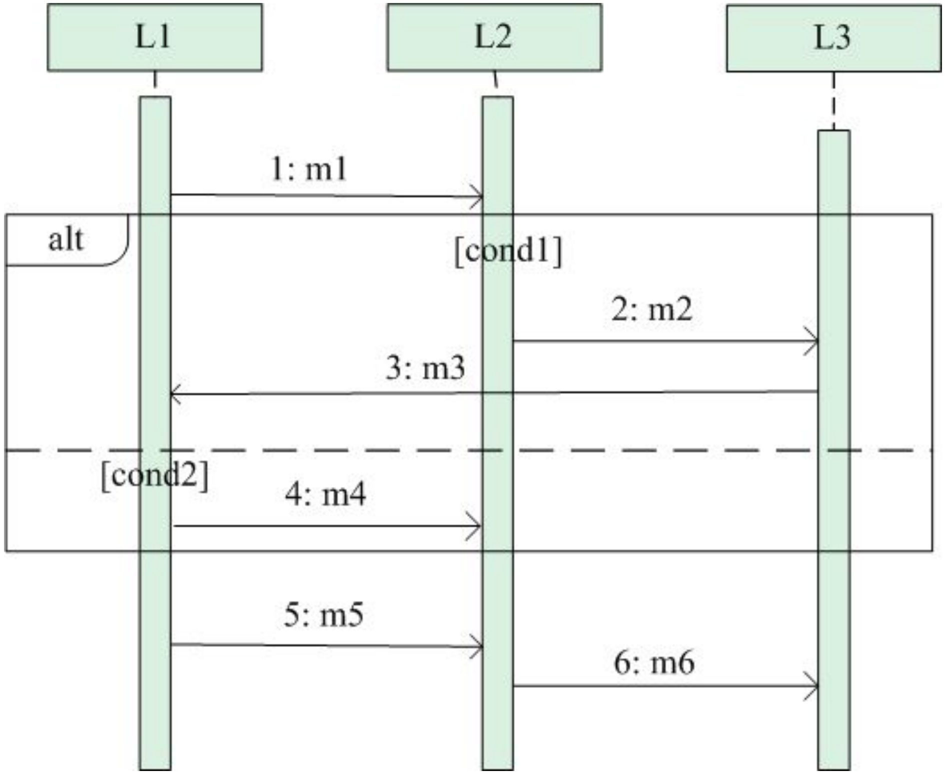
Sequence Diagram – An Example



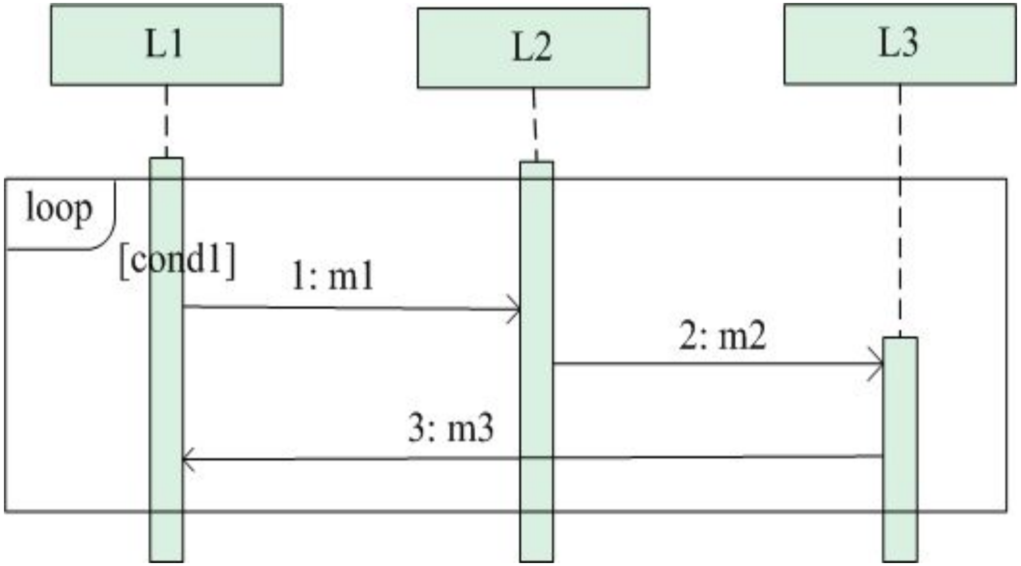
Sequence Diagram – Advanced Features

- **Use Combined Fragments, which consists of a region of a sequence diagram, to represent**
  - **Branching: operator “alt”**
  - **Loop: operator “loop”**
  - **Assertion: operator "assert"**

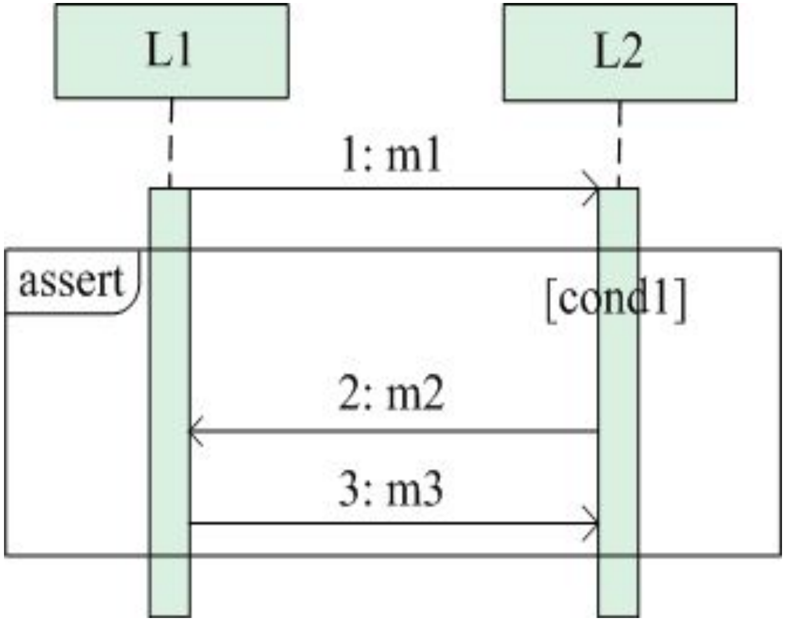
Alternative



Loop



Assertion

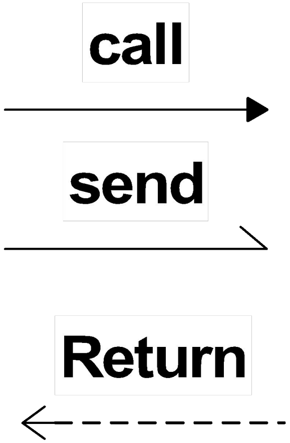


Sequence Diagrams

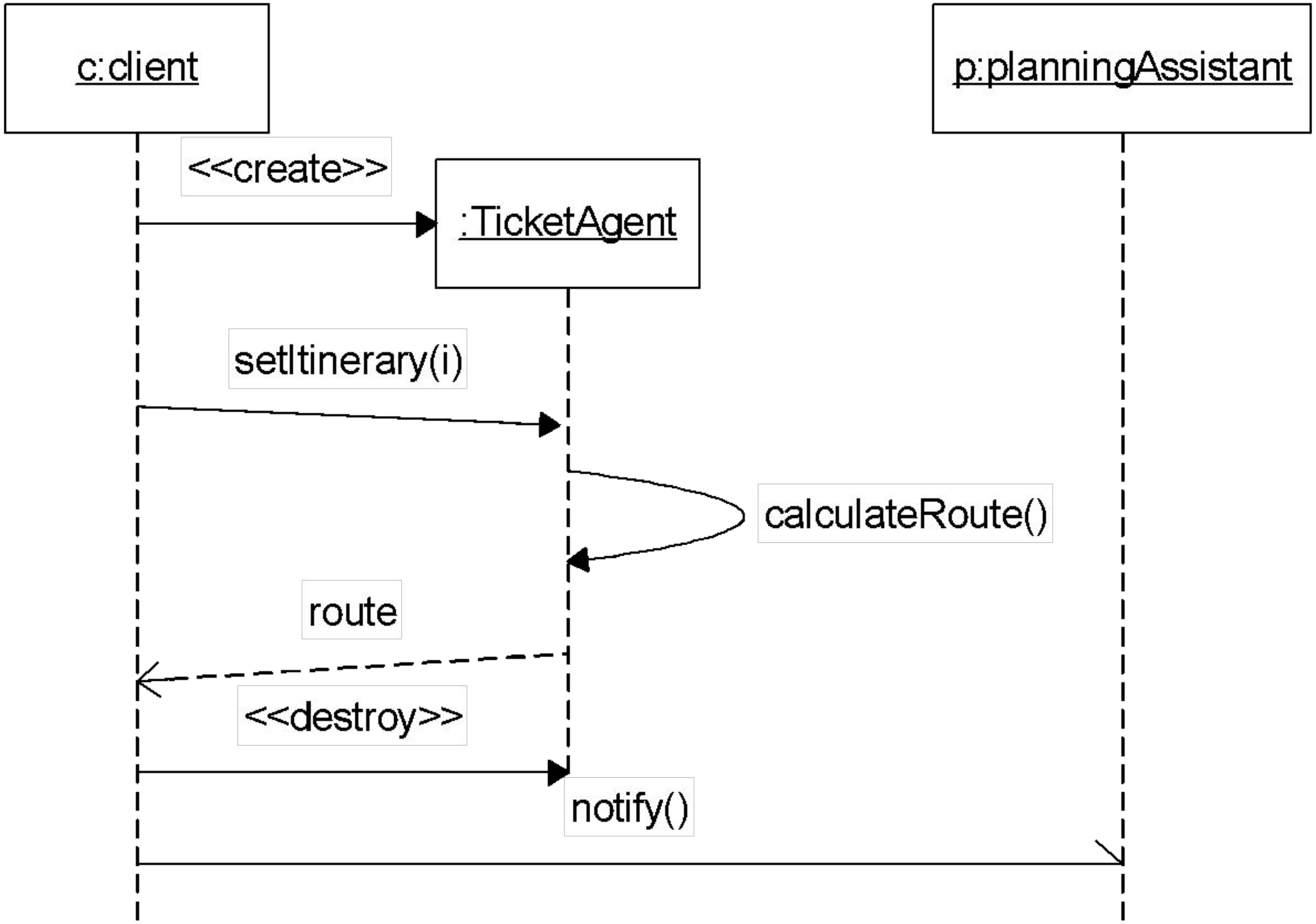
- X-axis is objects
  - Object that initiates interaction is left most
  - Object to the right are increasingly more subordinate
- Y-axis is time
  - Messages sent and received are ordered by time
- Object life lines represent the existence over a period of time
- Activation (double line) is the execution of the procedure.

Message Passing

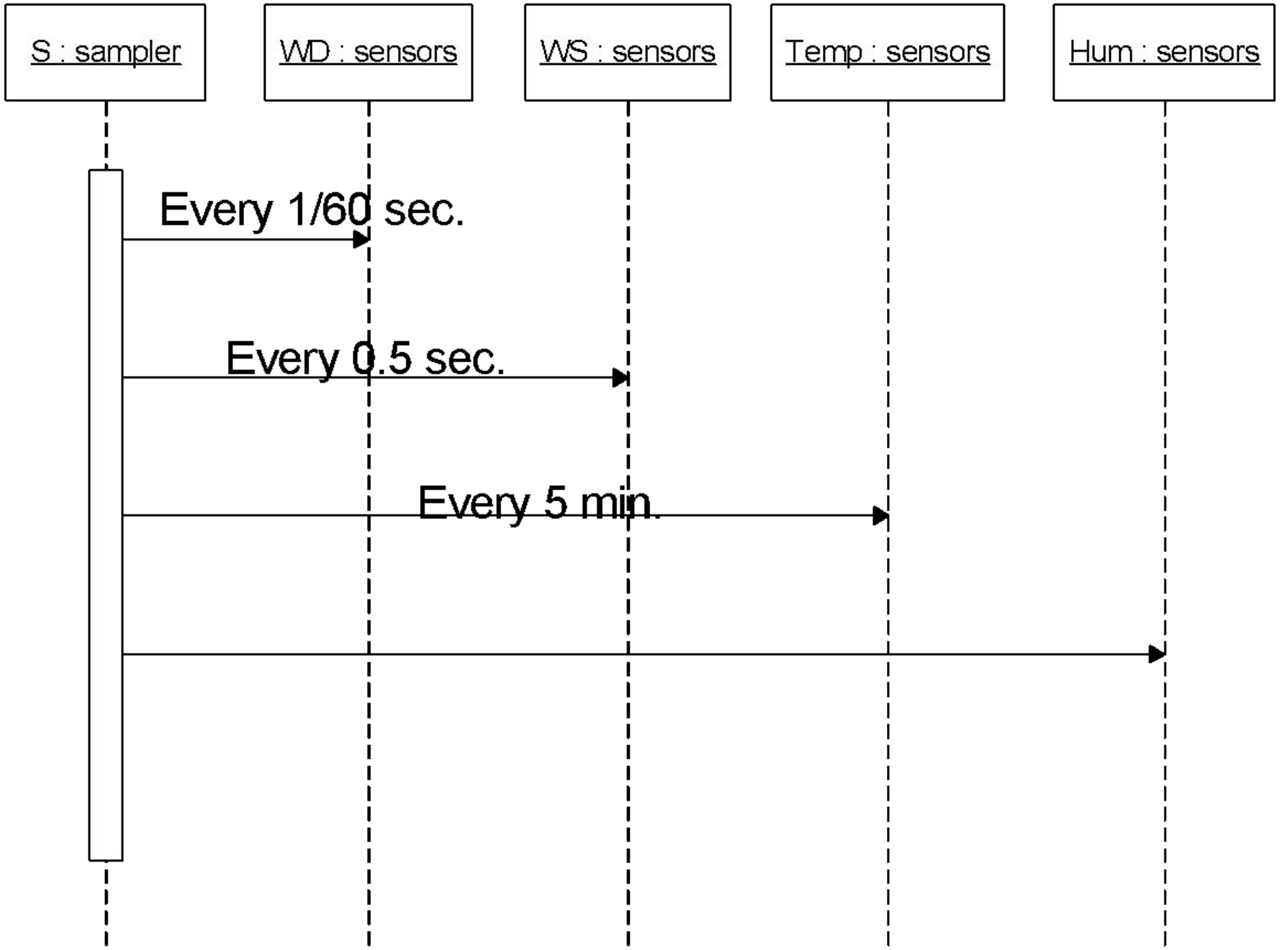
- Send – sends a signal (message) to an object
- Return – returns a value to a caller
- Call – invoke an operation
- Stereotypes
  - <<create>>
  - <<destroy>>

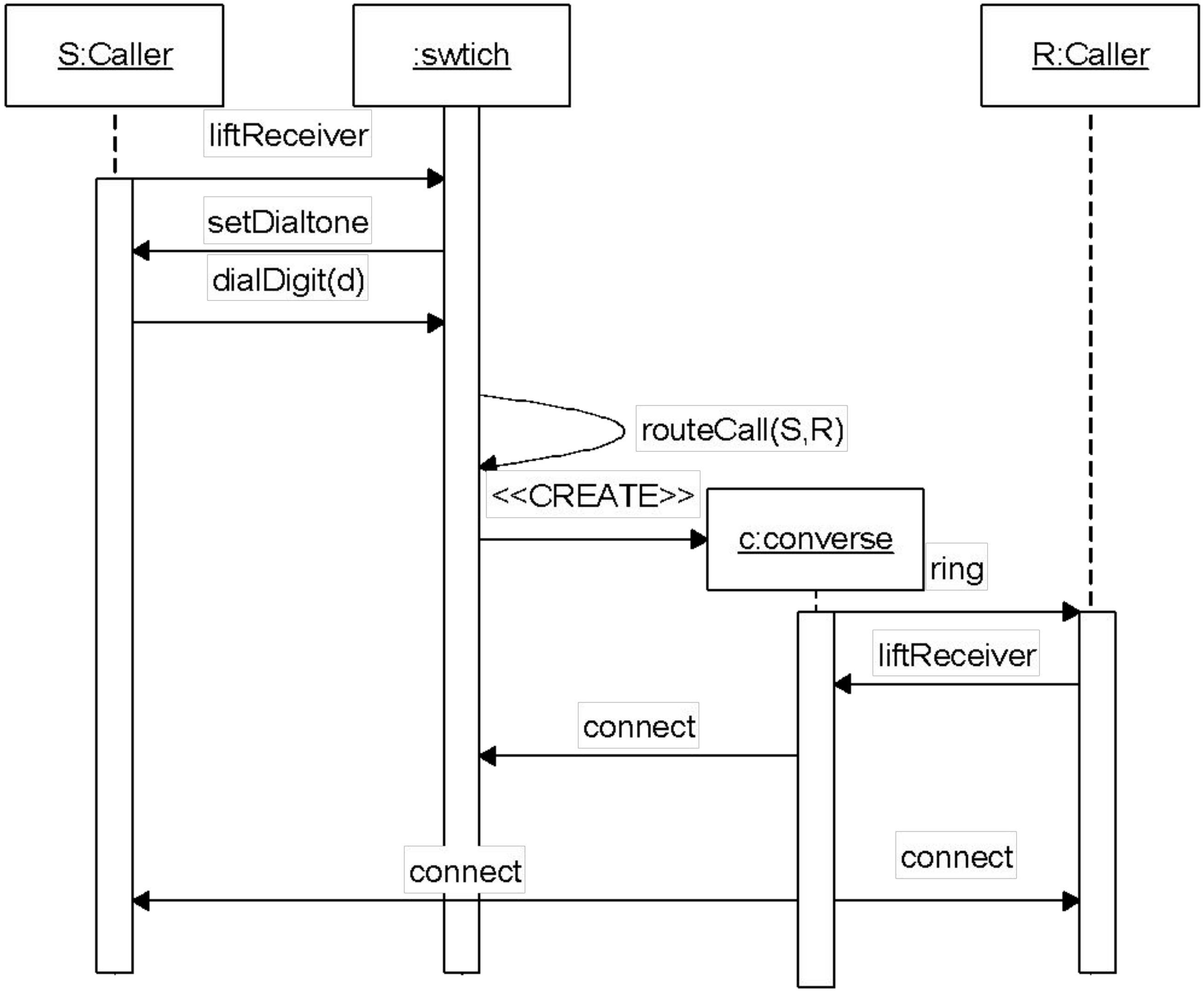


Example UML Sequence Diagram

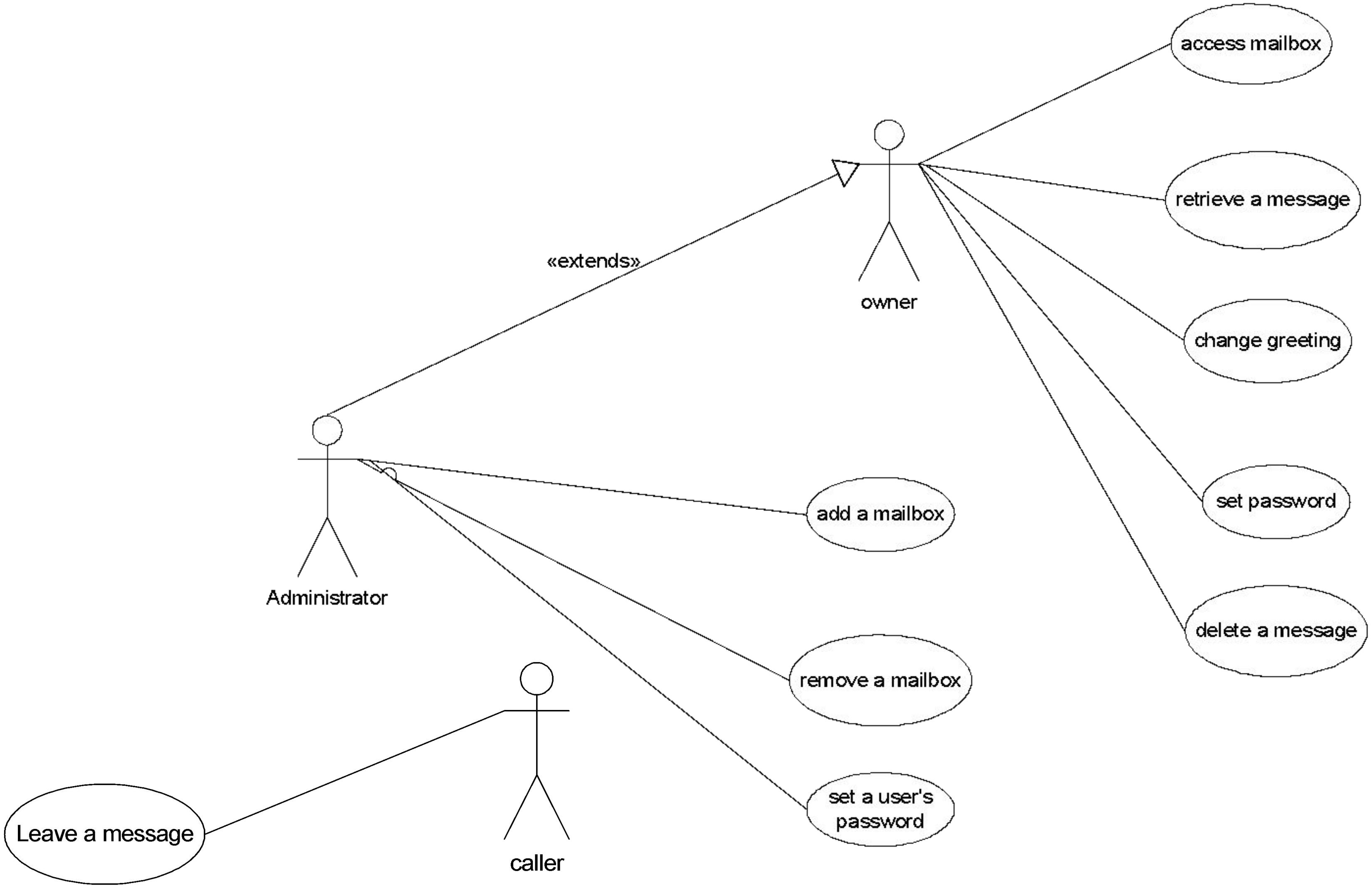


Example

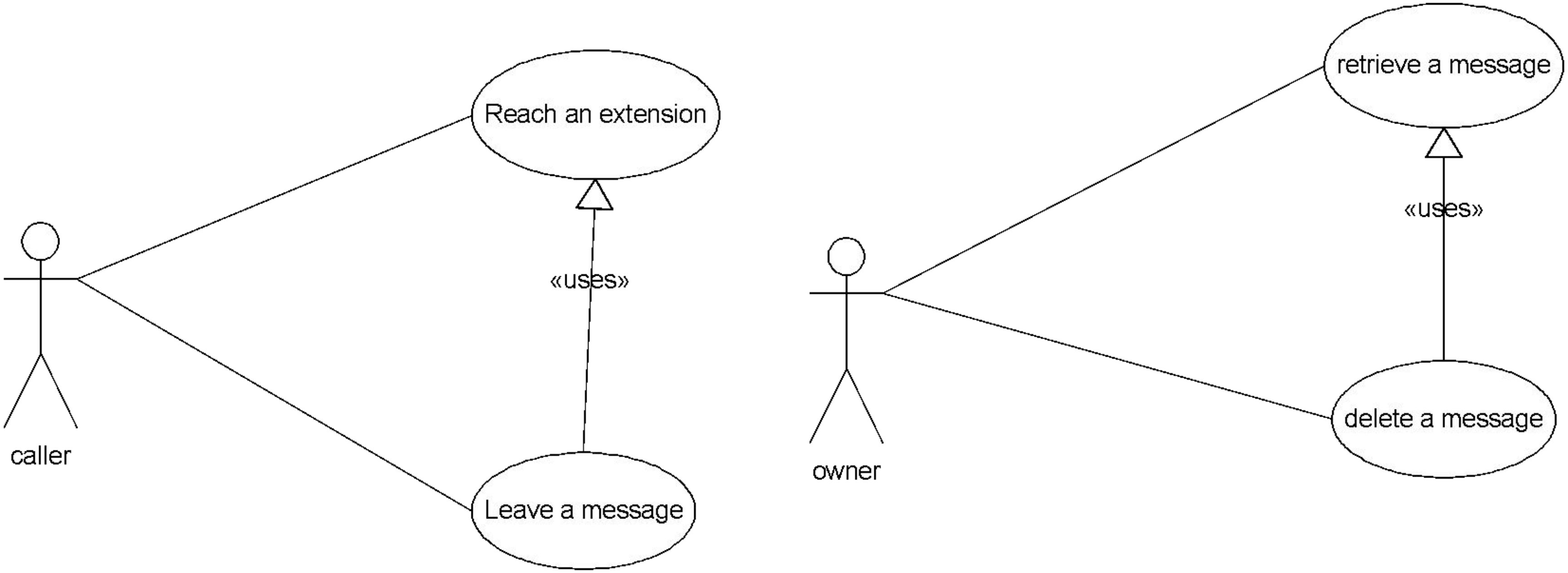

S : sampler
WD : sensors
WS : sensors
Temp : sensors
Hum : sensors
Every 1/60 sec.
Every 0.5 sec.
Every 5 min.


S:Caller
:swtich
R:Caller
liftReceiver
setDialtone
dialDigit(d)
routeCall(S,R)
<<CREATE>>
c:converse
ring
liftReceiver
connect
connect
connect

Mail System

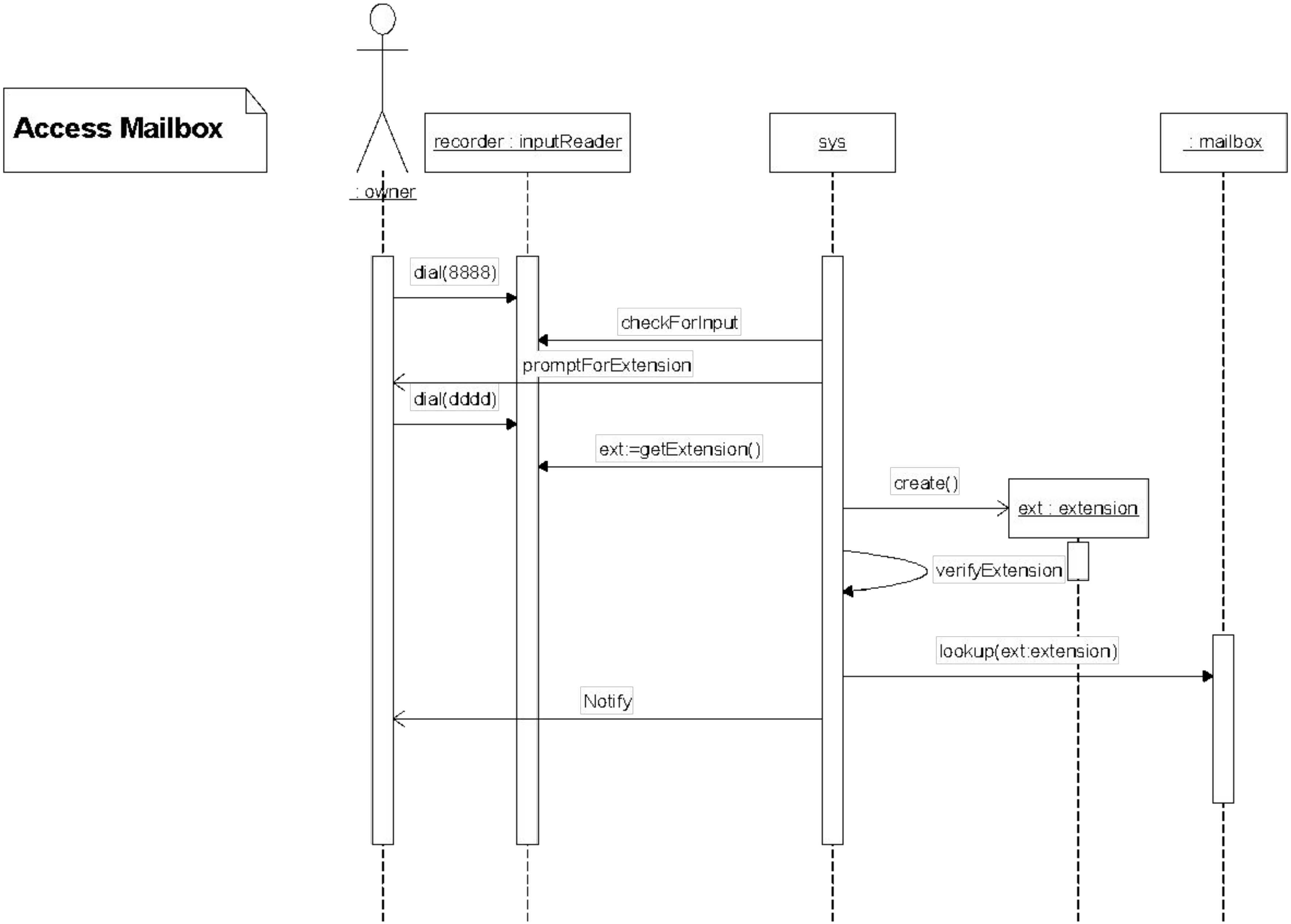

access mailbox
retrieve a message
«extends»
owner
change greeting
set password
delete a message
Administrator
add a mailbox
remove a mailbox
set a user's password
Leave a message
caller

Mail System (2)

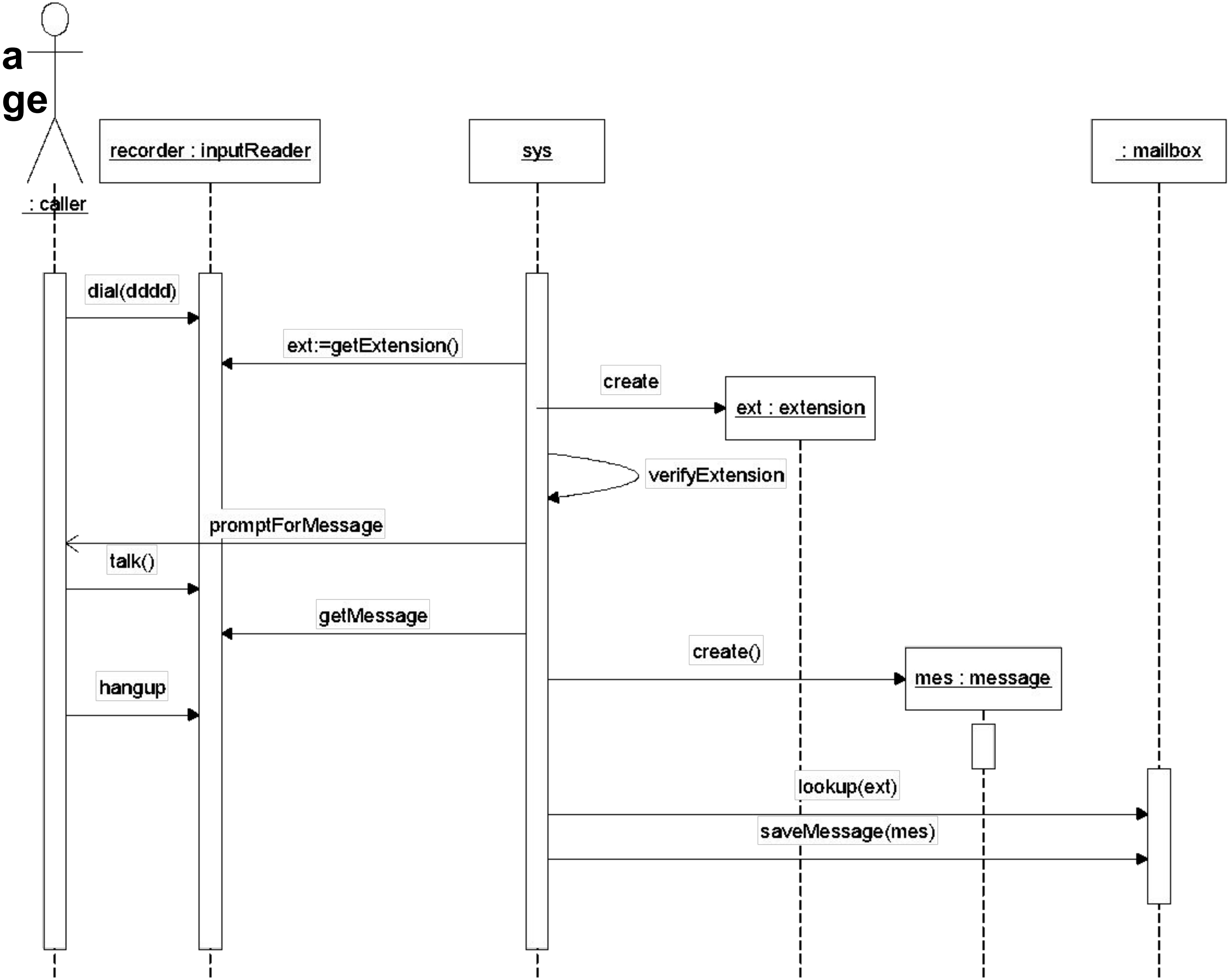


Mail System Objects

- Caller, owner, administrator
- Mailbox, extension, password, greeting
- Message, message list
- Mail system
- Input reader/device


Access Mailbox
: owner
recorder : inputReader
sys
: mailbox
dial(8888)
checkForInput
promptForExtension
dial(dddd)
ext:=getExtension()
create()
ext : extension
verifyExtension
lookup(ext:extension)
Notify

Leave a message



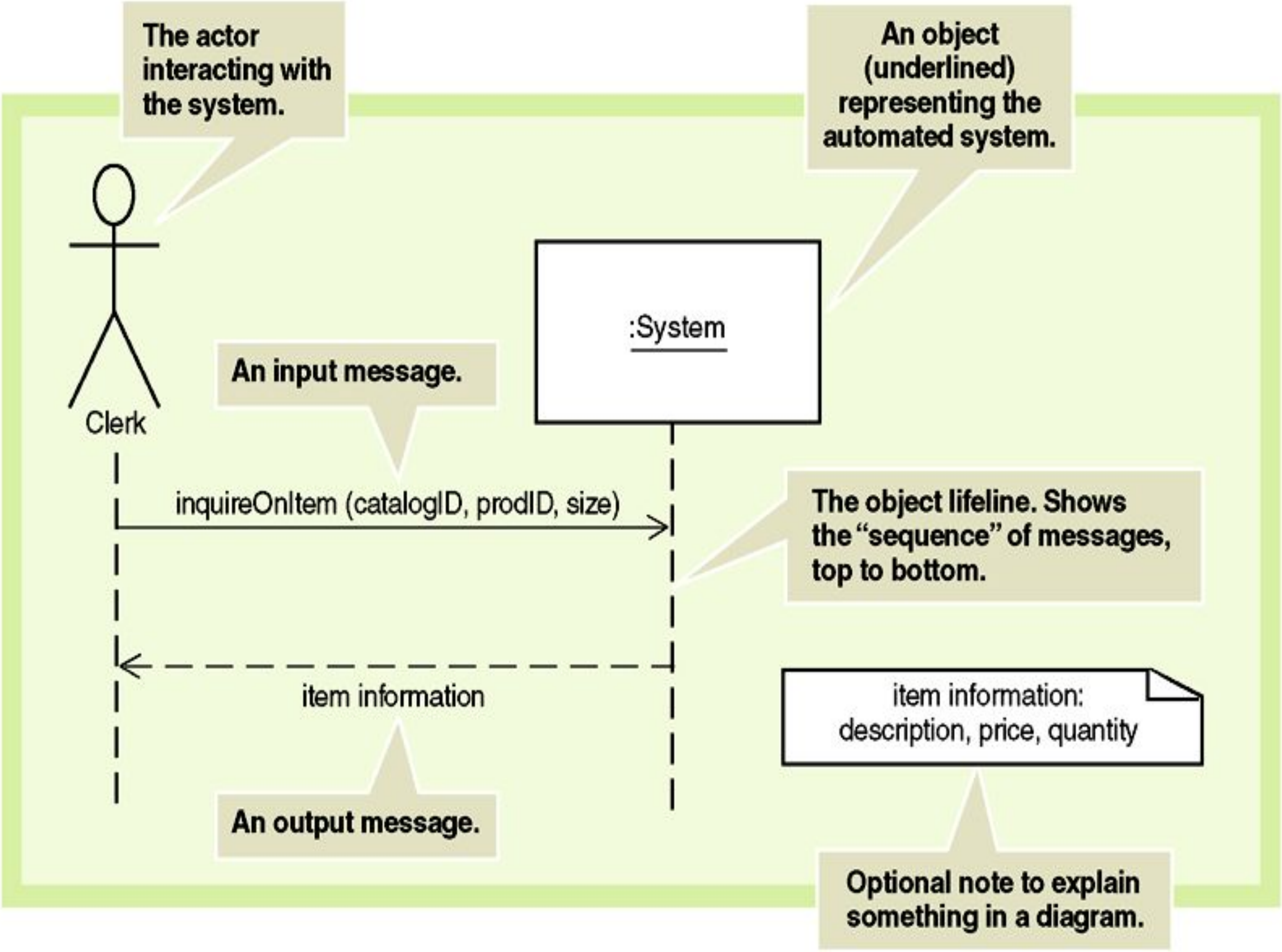
Properties of Sequence Diagrams

- Initiator is leftmost object (boundary object)
- Next is typically a control object
- Then comes entity objects

Sample System Sequence Diagram (SSD)

FIGURE 7-14

Sample system sequence diagram (SSD).



The actor interacting with the system.
An object (underlined) representing the automated system.
:System
Clerk
An input message.
inquireOnItem (catalogID, prodID, size)
The object lifeline. Shows the "sequence" of messages, top to bottom.
item information
item information: description, price, quantity
An output message.
Optional note to explain something in a diagram.

SSD Notation

- Actor represented by stick figure – person (or role) that “interacts” with system by entering input data and receiving output data
- Object notation is rectangle with name of object underlined – shows individual object and not class of all similar objects
- Lifeline is vertical line under object or actor to show passage of time for object
- Messages use arrows to show messages sent or received by actor or system

SSD Lifelines

- **Vertical line under object or actor:**
  - Shows passage of time
- **If vertical line dashed:**
  - Creation and destruction of thing is not important for scenario
- **Long narrow rectangles:**
  - Activation lifelines emphasize that object is active only during part of scenario

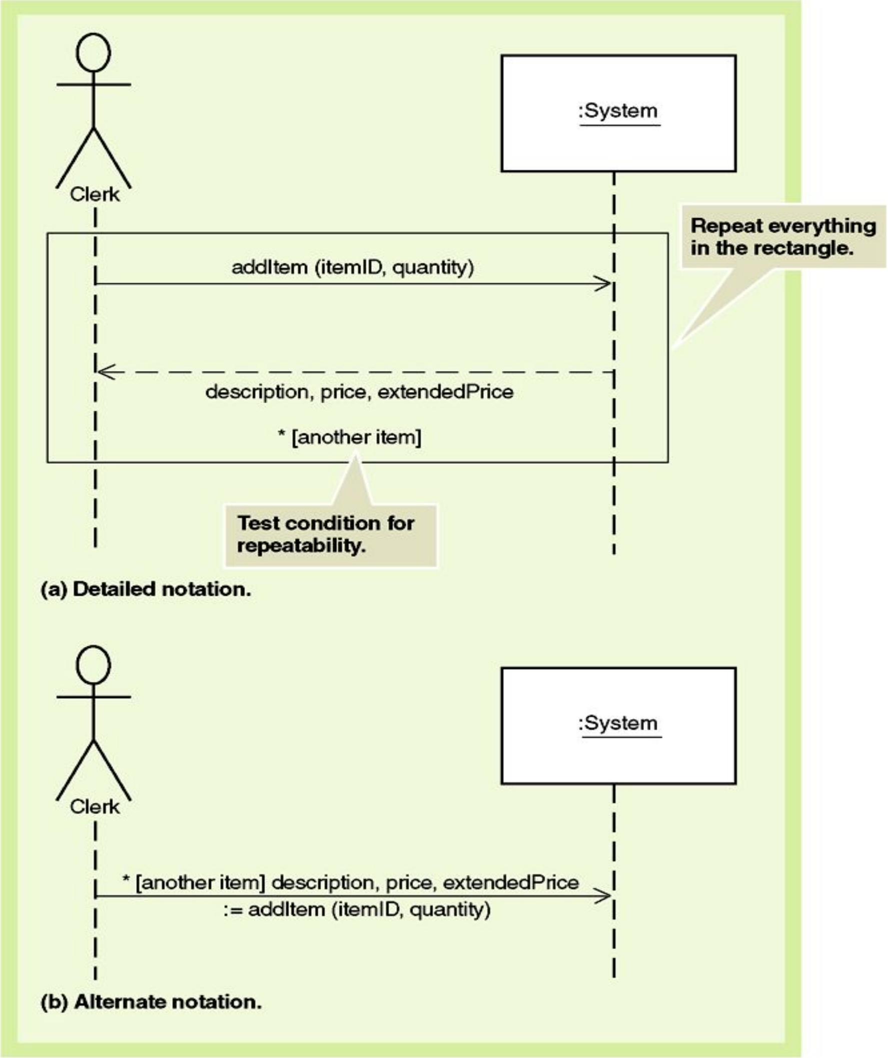
SSD Messages

- Internal events identified by the flow of objects within a scenario
- Requests from one actor or object to another to do some action
- Invokes a particular method

Repeating Message

FIGURE 7-15

Repeating message.
(a) Detailed notation.
(b) Alternate notation.


Clerk
:System
addItem (itemID, quantity)
description, price, extendedPrice
* [another item]
Repeat everything in the rectangle.
Test condition for repeatability.
Clerk
:System
* [another item] description, price, extendedPrice
:= addItem (itemID, quantity)

(a) Detailed notation.

(b) Alternate notation.

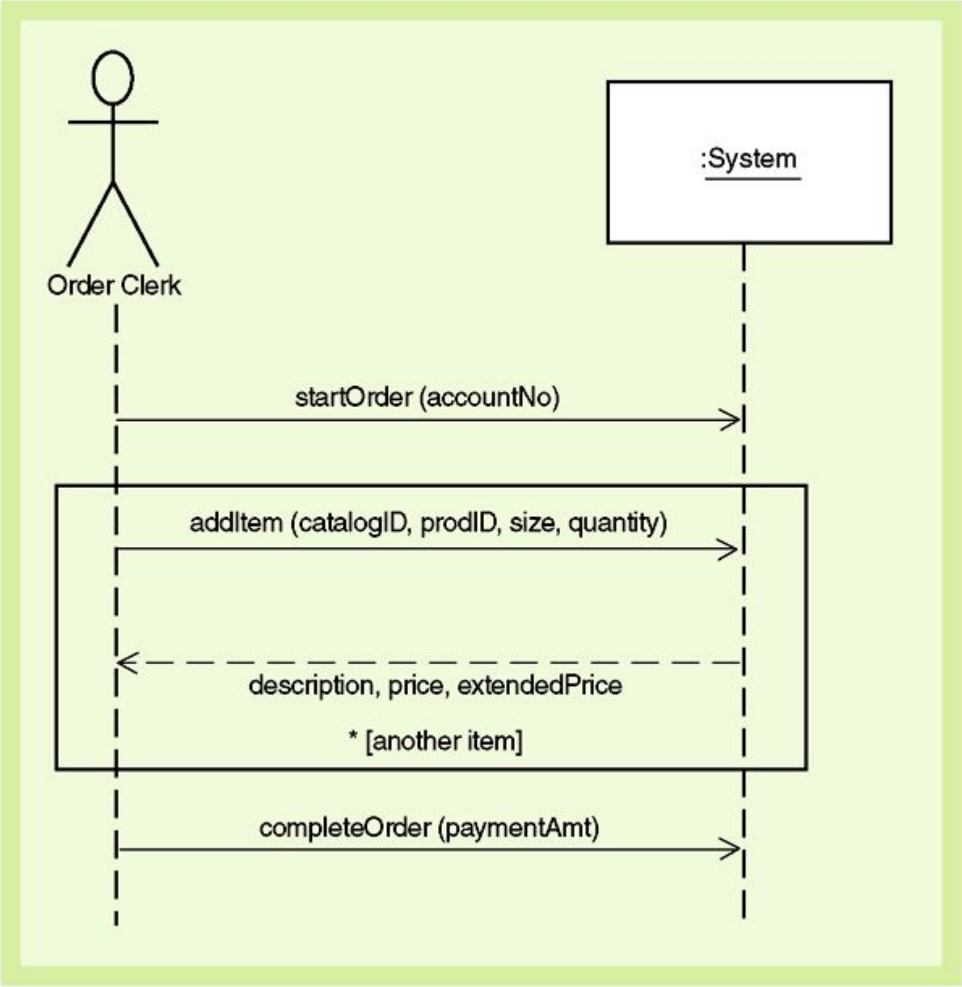
Developing a System Sequence Diagram

- Begin with detailed description of use case from fully developed form or activity diagrams
- Identify input messages
- Describe message from external actor to system using message notation
- Identify and add any special conditions on input message, including iteration and true/false conditions
- Identify and add output return messages

SSD of Simplified Telephone Order Scenario for *Create New Order* Use Case

FIGURE 7-17

An SSD of the simplified telephone order scenario for the *Create new order* use case.

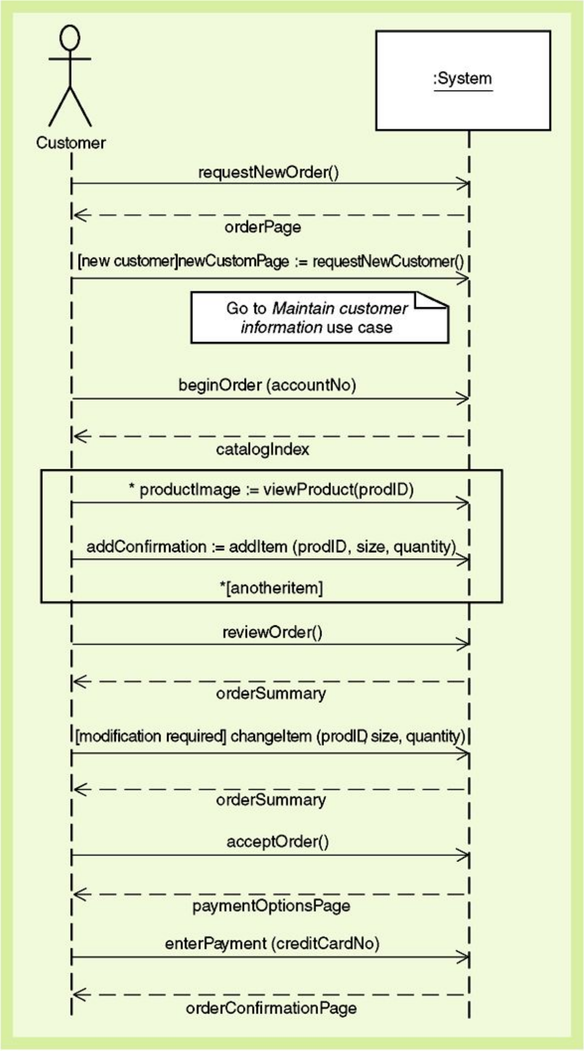


:System
Order Clerk
startOrder (accountNo)
addItem (catalogID, prodID, size, quantity)
description, price, extendedPrice
* [another item]
completeOrder (paymentAmt)

SSD of the Web Order Scenario for the *Create New Order* Use Case

FIGURE 7-18

An SSD of the Web order scenario for the *Create new order* use case.

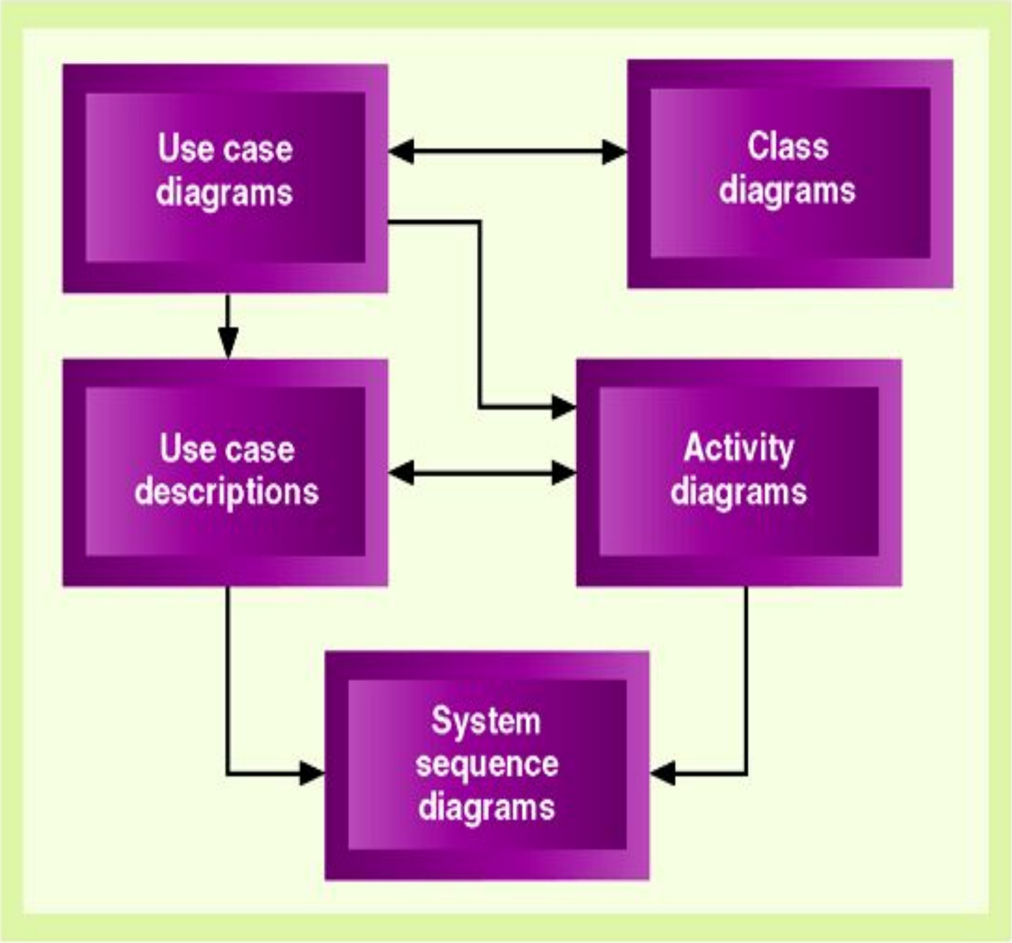


Customer
:System
requestNewOrder()
orderPage
[new customer]newCustomPage := requestNewCustomer()
Go to *Maintain customer information* use case
beginOrder (accountNo)
catalogIndex
* productImage := viewProduct(prodID)
addConfirmation := addItem (prodID, size, quantity)
*[anotheritem]
reviewOrder()
orderSummary
[modification required] changeItem (prodID, size, quantity)
orderSummary
acceptOrder()
paymentOptionsPage
enterPayment (creditCardNo)
orderConfirmationPage

Relationships Between OO Requirements Models

FIGURE 7-21

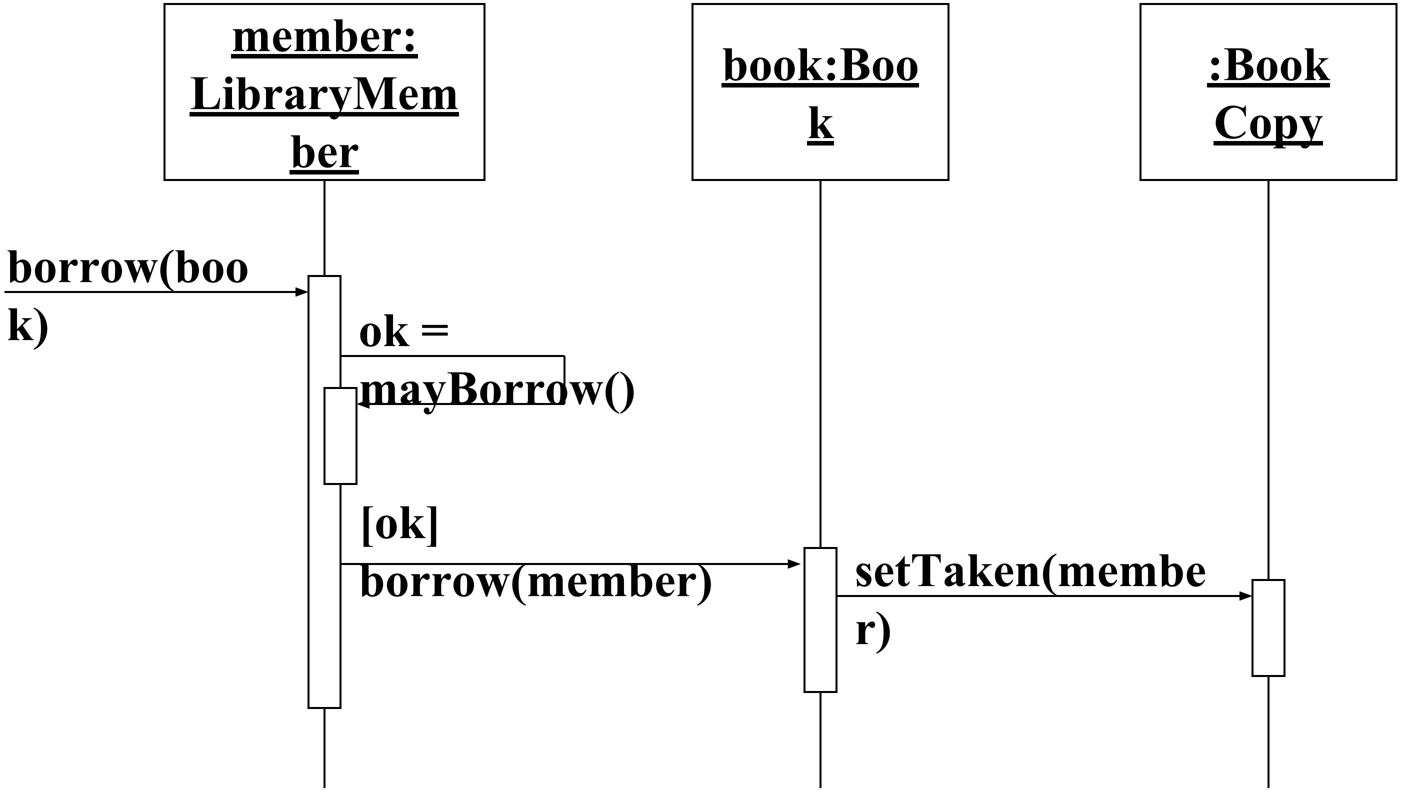
Relationships between OO requirements models.



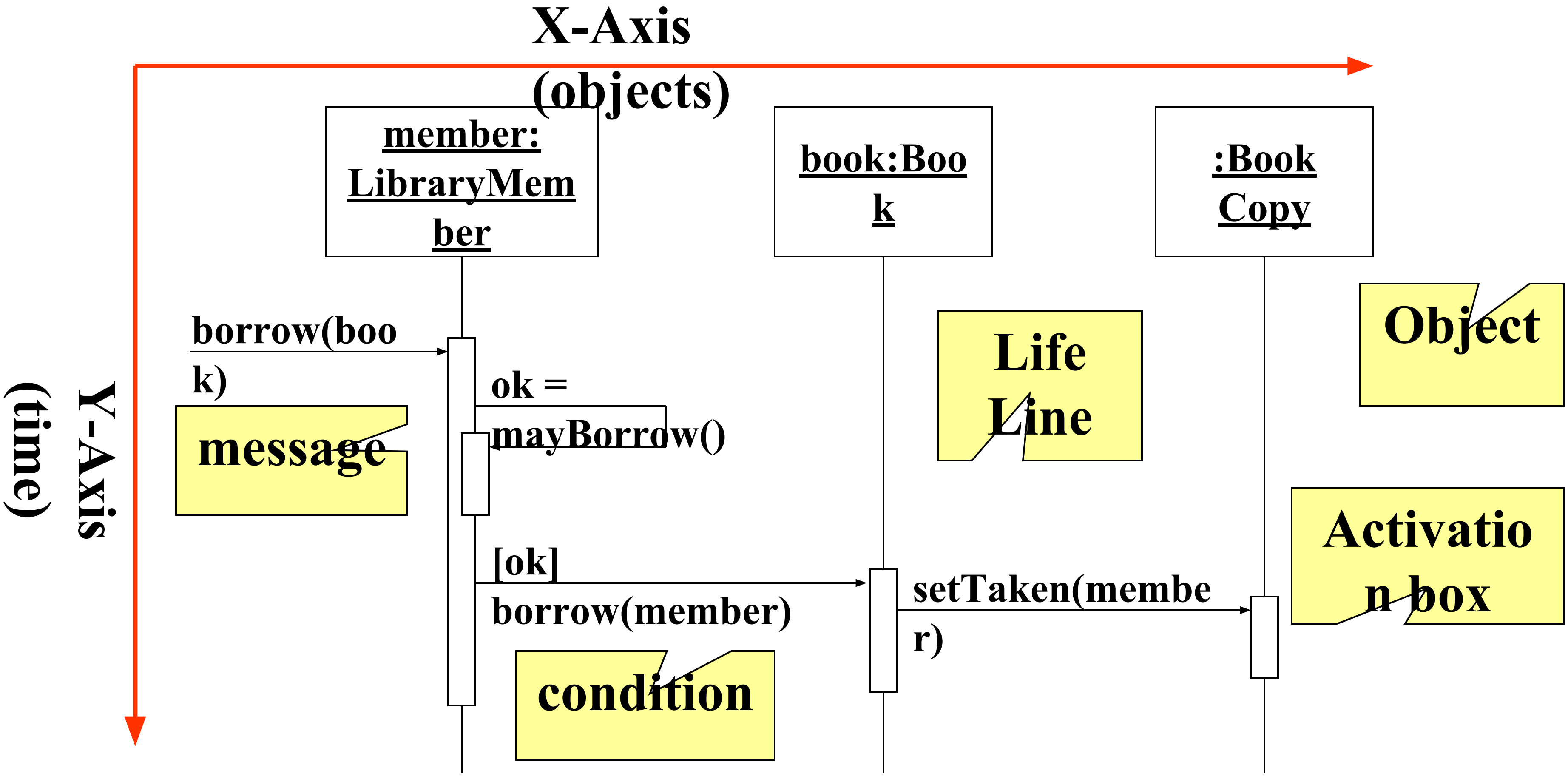
A First Look at Sequence Diagrams

- Illustrates how objects interacts with each other.
- Emphasizes time ordering of messages.
- Can model simple sequential flow, branching, iteration, recursion and concurrency.

A Sequence Diagram

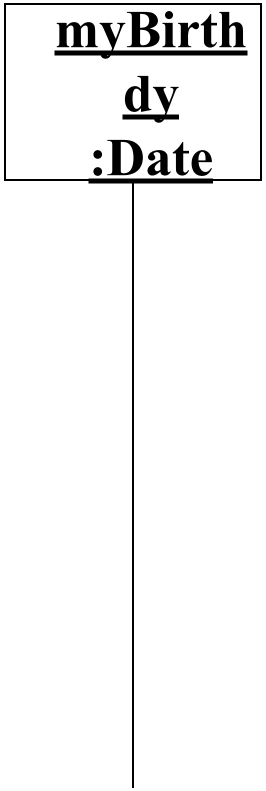


A Sequence Diagram



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**



myBirthdy :Date

Messages

- An interaction between two objects is performed as a message sent from one object to another (simple operation call, Signaling, RPC)
- If object obj_1 sends a message to another object obj_2 some link must exist between those two objects (dependency, same objects)

Messages (Cont.)

- A message is represented by an arrow between the life lines 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.

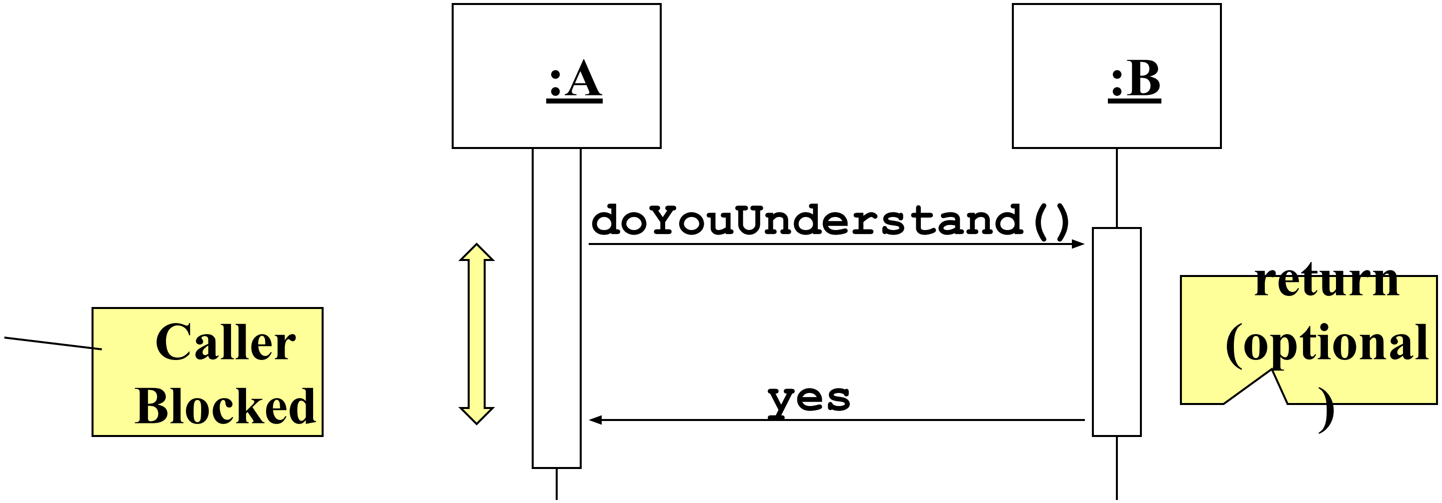
Return Values

- **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()`**

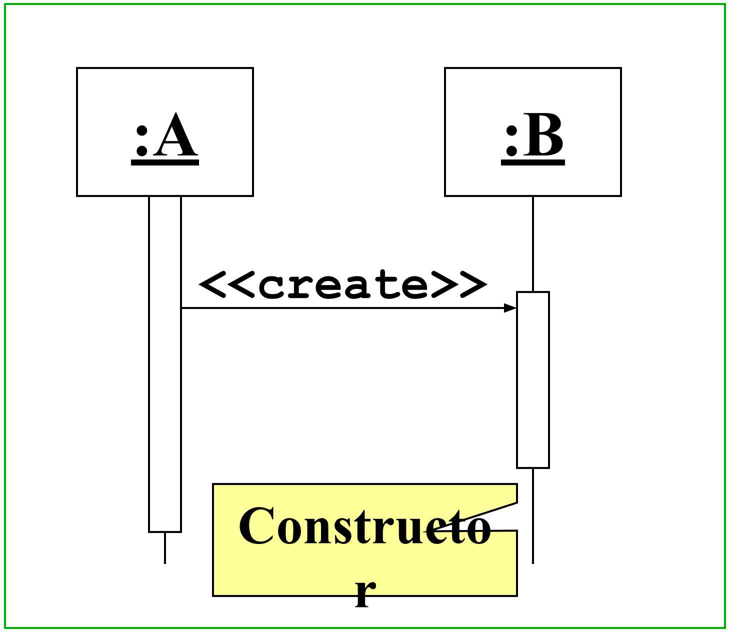
Synchronous Messages

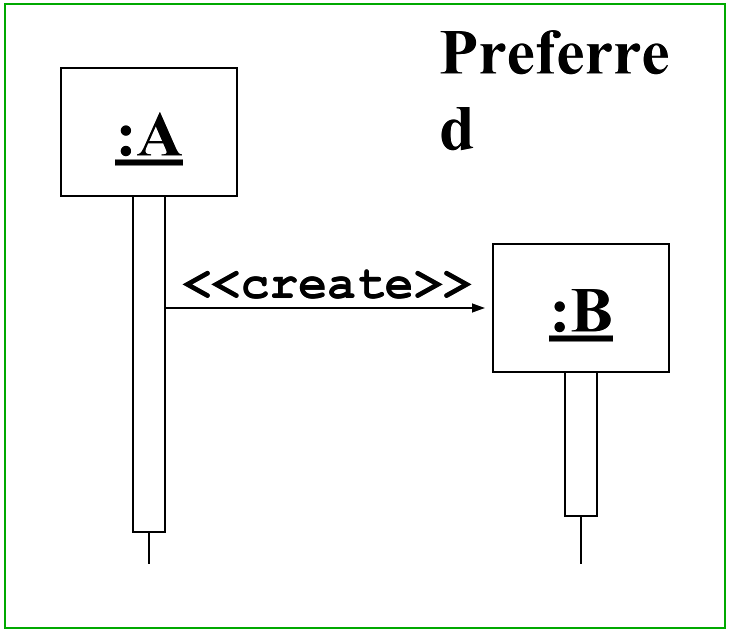
- **Nested flow of control, typically implemented as an operation call.**
  - **The routine that handles the message is completed before the caller resumes execution.**



Object Creation

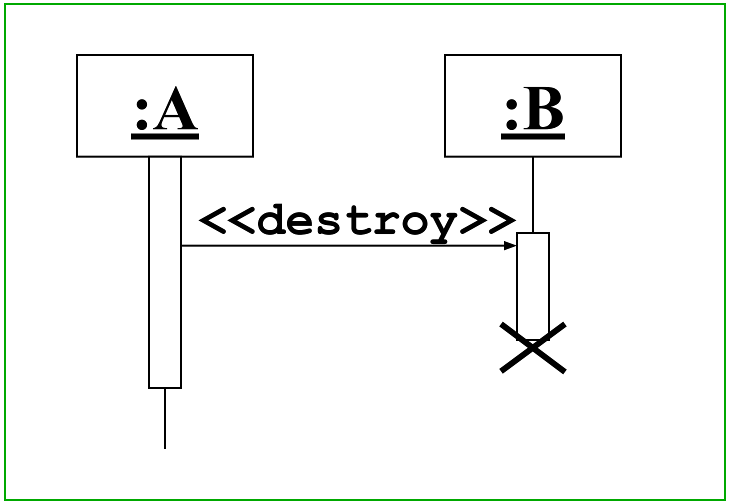
- An object may create another object via a `<<create>>` message.





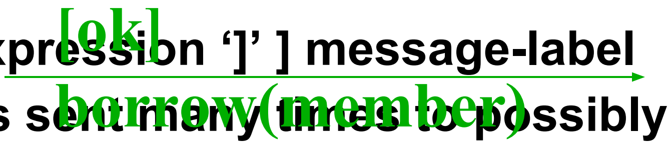
Object Destruction

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



Control information

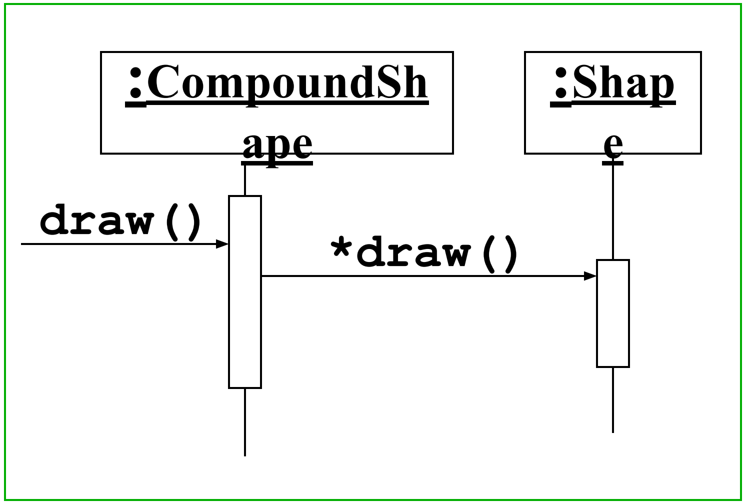
- **Condition**
  - syntax: ‘[‘ expression ’]’ message-label
  - The message is sent only if the condition is true
  - example:

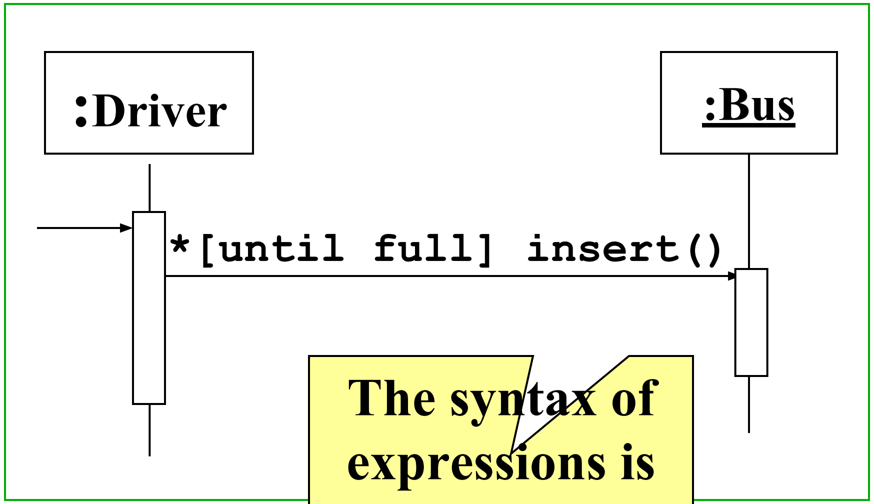


- **Iteration**
  - syntax: * [‘[‘ expression ‘]’] message-label
  - The message is sent many times to possibly multiple receiver objects.

Control Information (Cont.)

- Iteration examples:



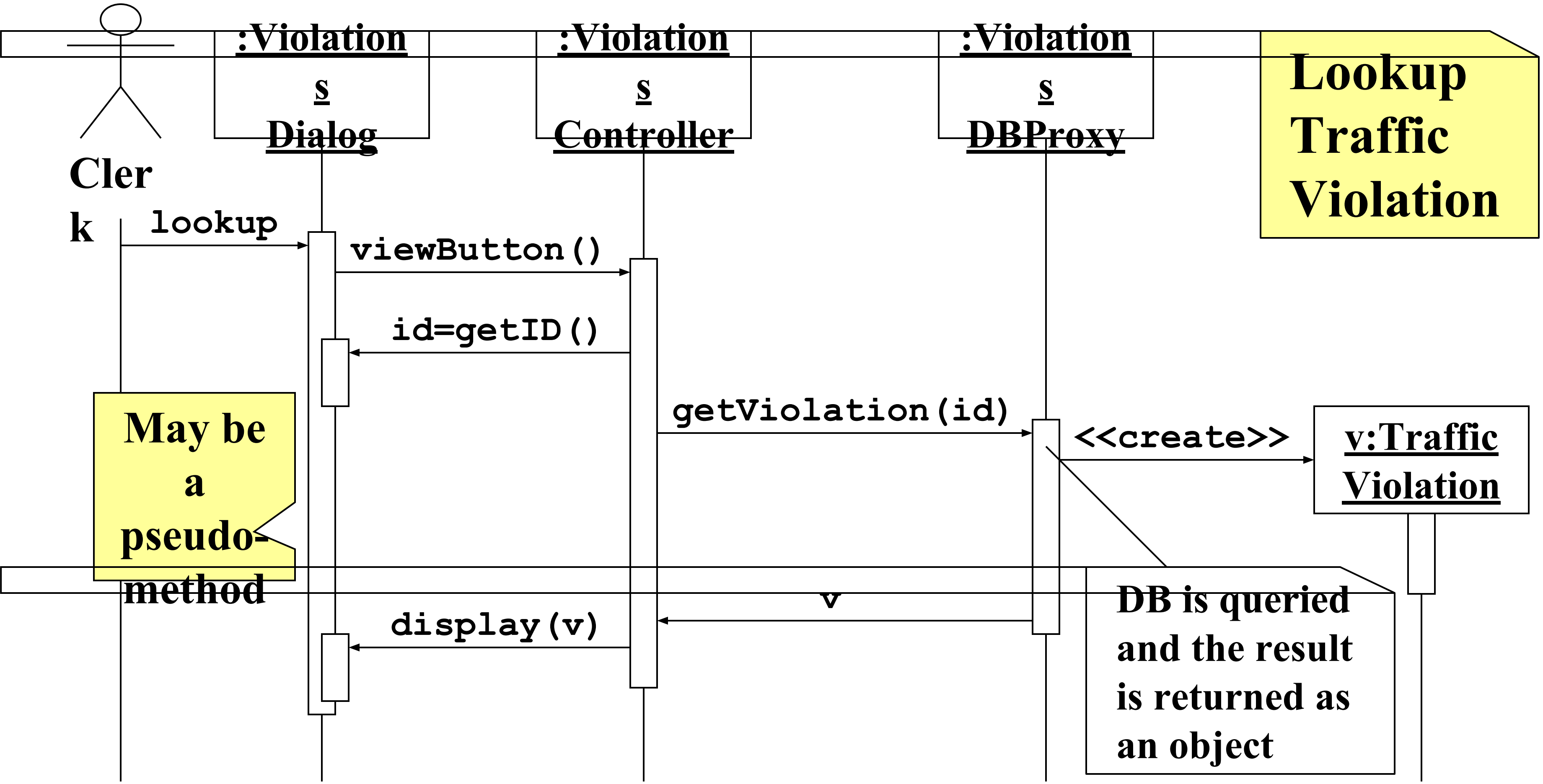


The syntax of expressions is not a standard

Control Information (Cont.)

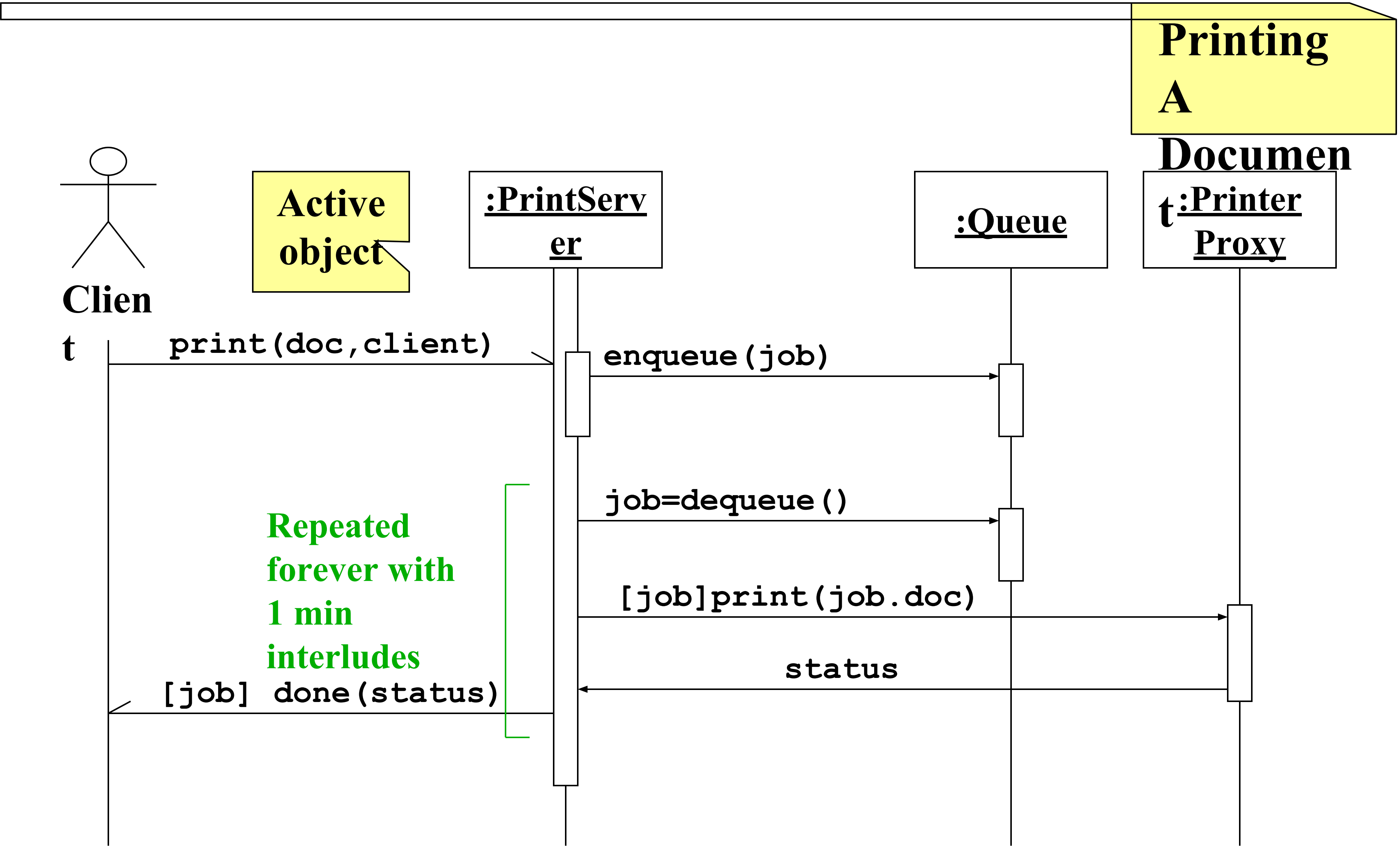
- **The control mechanisms of sequence diagrams suffice only for modeling simple alternatives.**
  - **Consider drawing several diagrams for modeling complex scenarios.**
  - **Don't use sequence diagrams for detailed modeling of algorithms (this is better done using *activity diagrams, pseudo-code*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

Example 1



Clerk
:Violations Dialog
:Violations Controller
:Violations DBProxy
Lookup Traffic Violation
lookup
viewButton()
id=getID()
getViolation(id)
<<create>>
v:Traffic Violation
May be a pseudo-method
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()
[job]print(job.doc)
status
[job] done(status)
Repeated forever with 1 min interludes

5.1. Consider a file system with a graphical user interface, such as Macintosh’s Finder, Microsoft’sWindows Explorer, or Linux’s KDE. The following objects were identified from a use case describing how to copy a file from a floppy disk to a hard disk: `File`, `Icon`, `TrashCan`, `Folder`, `Disk`, `Pointer`. Specify which are entity objects, which are boundary objects, and which are control objects.

5 Points.

Entity objects: File, Folder, Disk

Boundary objects: Icon, Pointer, TrashCan

Control objects: none in this example.

5.2 Assuming the same file system as before, consider a scenario consisting of selecting a file on a floppy, dragging it to Folder and releasing the mouse. Identify and define at least one control object associated with this scenario.

5 Points.

The purpose of a control object is to encapsulate the behavior associated with a user level transaction. In this example, we identify a CopyFile control object, which is responsible for:

1. **Remembering the path of the destination folder**
2. **Checking if the file can be copied (access control and disk space).**
3. **Remembering the path of the original file**
4. **To initiate the file copying.**

5.3. Arrange the objects listed in Exercises 5.1. and 5.2. horizontally on a sequence diagram, the boundary objects to the left, then the control object you identified, and finally, the entity objects. Draw the sequence of interactions resulting from dropping the file into a folder. For now, ignore the exceptional cases.

In this specific solution, we did not focus on the `Disk`, `Pointer`, and `TrashCan` objects. The Disk object would be added to the sequence when checking if there is available space. The TrashCan object is needed for scenarios in which Files or Folders are deleted.

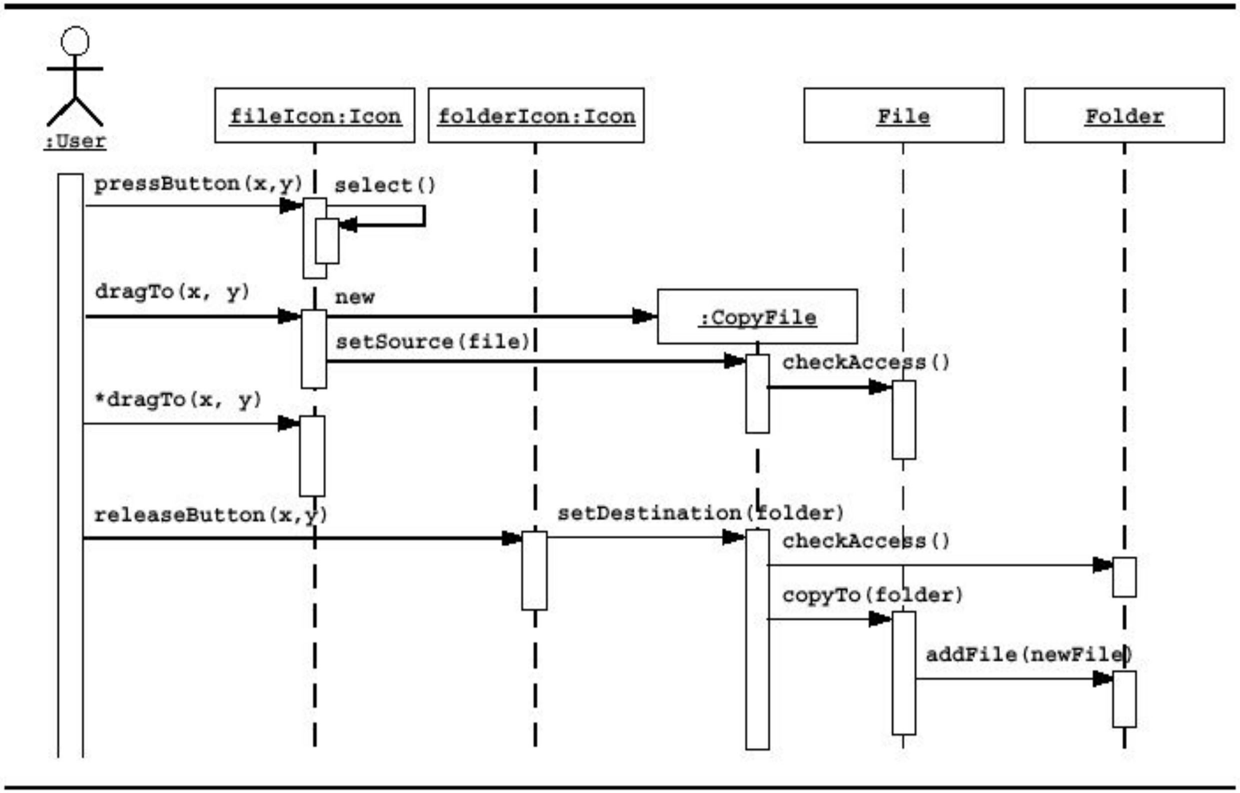
Note that the interaction among boundary objects can be complex, depending on the user interface components that are used. This sequence diagram, however, only describes user level behavior and should not go into such details.

As a result, the sequence diagram depicts a high level view of the interactions between these objects, not the *actual* sequence of message sends that occurs in the delivered system.

5.3 continued

Figure below depicts a *possible* solution to this exercise. The names and parameters of the operations may vary. The diagram, however, should at least contain the following elements:

- **Two boundary objects, one for the file being copied, and one of the destination folder.**
- **At least one control object remembering the source and destination of the copy, and possibly checking for access rights.**
- **Two entity objects, one for the file being copied, and one of the destination folder.**



Interaction Diagrams

Interaction diagrams model the behavior of use cases by describing the way groups of objects interact to complete the task of the use case. They portray the interaction among the objects of a system and describe the dynamic behavior of the system.

There are two types of interaction diagrams

Sequence Diagrams and **Communication Diagrams** (formally known as collaboration diagrams)

Interaction Diagrams

Sequence diagrams

generally show the sequence of events that occur.

Collaboration diagrams

demonstrate how objects are statically connected.

Both diagrams are relatively simple to draw and contain similar elements.

Interaction Diagrams

Purpose of interaction diagrams

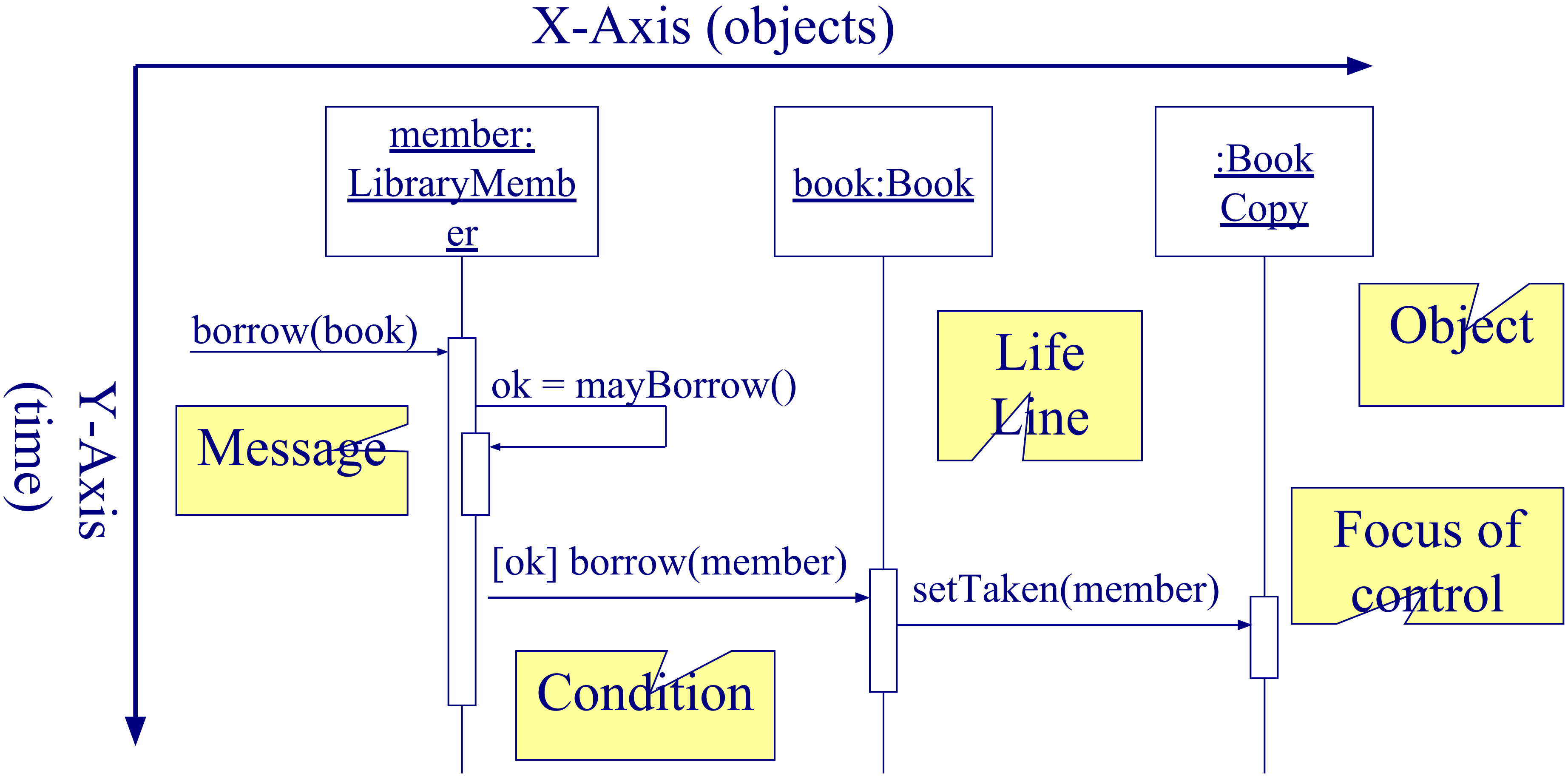
- Model interactions between objects
- Assist in understanding how a system (i.e., a use case) actually works
- Verify that a use case description can be supported by the existing classes
- Identify responsibilities/operations and assign them to classes

Sequence Diagram

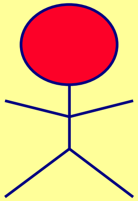
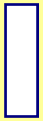
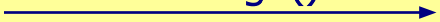
Illustrates the objects that participate in a use case and the messages that pass between them <u>over time</u> for *one* use case

In design, used to distribute use case behavior to classes

Sequence Diagram



Sequence Diagram Syntax

AN ACTOR	
AN OBJECT	anObject:aClass
A LIFELINE	
A FOCUS OF CONTROL	
A MESSAGE	aMessage() 
OBJECT DESTRUCTION	x

Sequence Diagram

Two major components

- Active objects
- Communications between these active objects
  - Messages sent between the active objects

Sequence Diagram

Active objects

- Any objects that play a role in the system
- Participate by sending and/or receiving messages
- Placed across the top of the diagram
- Can be:
  - An actor (from the use case diagram)
  - Object/class (from the class diagram) within the system

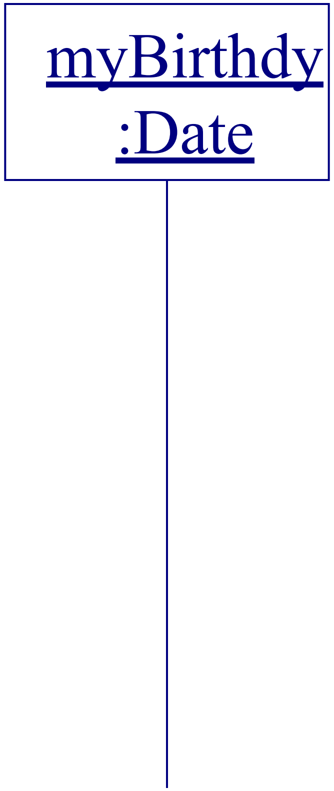
Active Objects

Object

- Can be any object or class that is valid within the system
- Object naming
  - Syntax

    [instanceName][:className]

2. Class name only :Classname
3. Instance name only objectName
4. Instance name and class name together object:Class

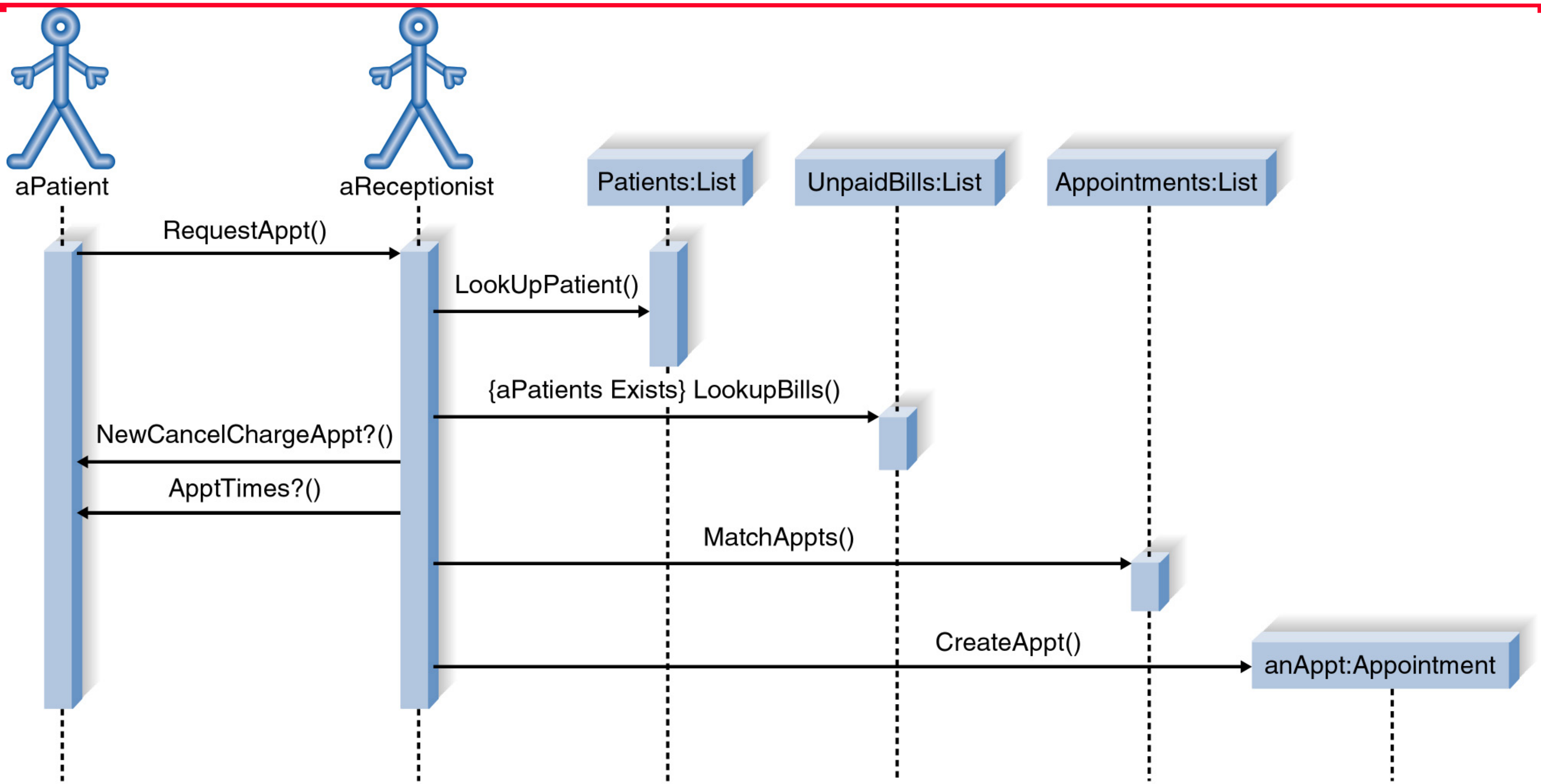

myBirthdy
:Date

Active Objects

Actor

- A person or system that derives benefit from and is external to the system
- Participates in a sequence by sending and/or receiving messages

Sequence Diagram



Communications between Active Objects

Messages

- Used to illustrate communication between different active objects of a sequence diagram
- Used when an object needs
  - to activate a process of a different object
  - to give information to another object

Messages

A message is represented by an arrow between the life lines of two objects.

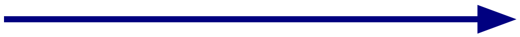
- Self calls are allowed

A message is labeled at minimum with the message name.

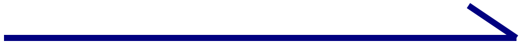
- Arguments and control information (conditions, iteration) may be included.

Types of Messages

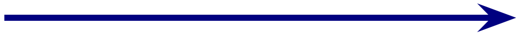
- Synchronous (flow interrupt until the message has completed)



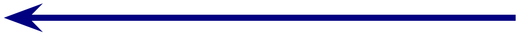
- Asynchronous (don’t wait for response)



- Flat (no distinction between sysn/async)

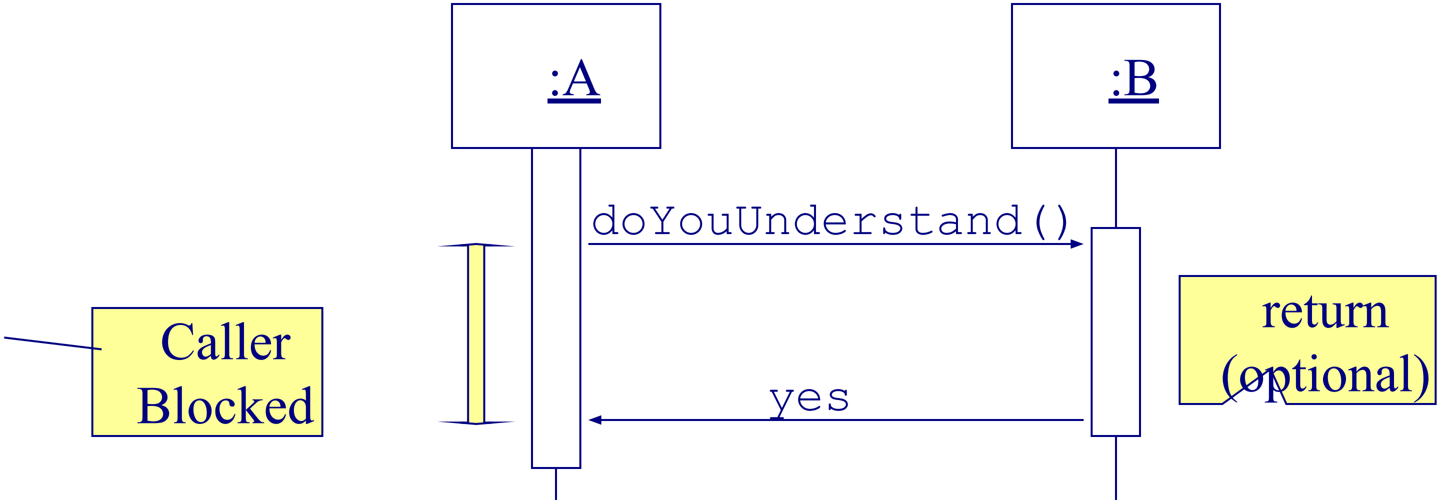


- Return (control flow has returned to the caller)



Synchronous Messages

The routine that handles the message is completed before the calling routine resumes execution.



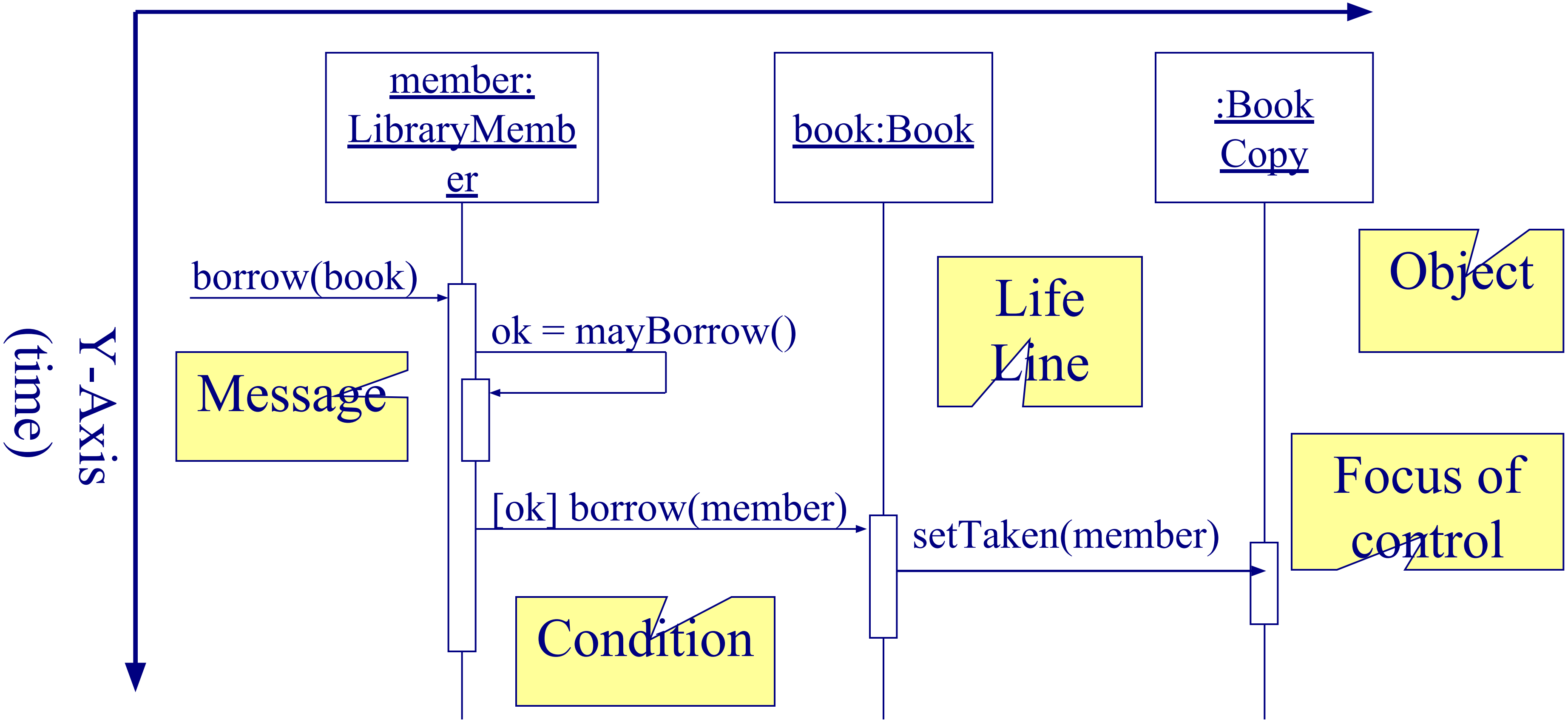
Asynchronous Messages

- Calling routine does not wait for the message to be handled before it continues to execute.
  - As if the call returns immediately
- Examples
  - Notification of somebody or something
  - Messages that post progress information

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()`

Sequence Diagram



Other Elements of Sequence Diagram

- Lifeline
- Focus of control (activation box or execution occurrence)
- Control information
  - Condition, repetition

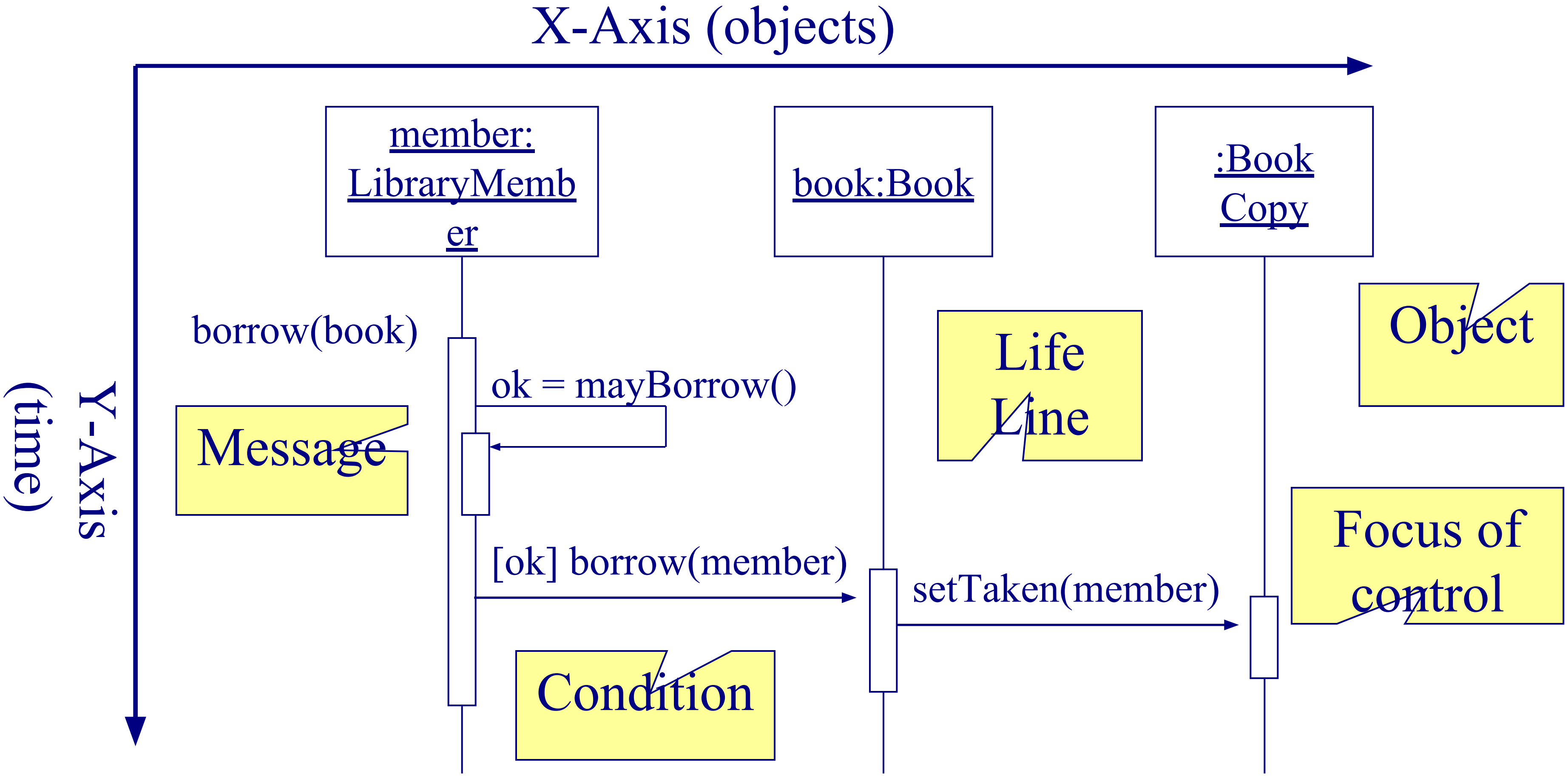
Sequence Diagram

- Lifeline
  - Denotes the life of actors/objects over time during a sequence
  - Represented by a vertical line below each actor and object (normally dashed line)
- For temporary object
  - place X at the end of the lifeline at the point where the object is destroyed

Sequence Diagram

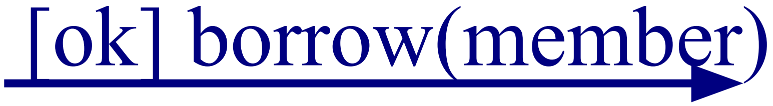
- Focus of control (activation box)
  - Means the object is active and using resources during that time period
  - Denotes when an object is sending or receiving messages
  - Represented by a thin, long rectangular box overlaid onto a lifeline

Sequence Diagram



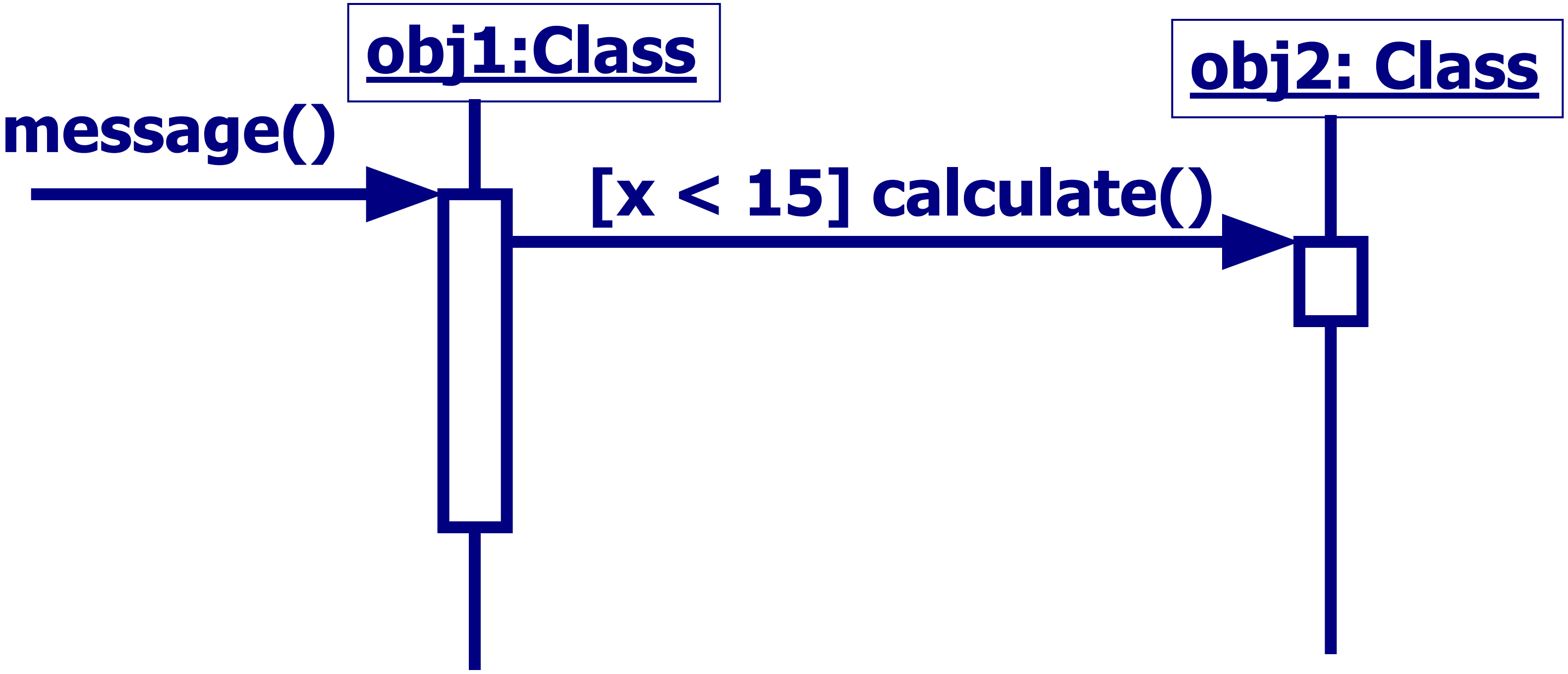
Control Information

- Condition
  - syntax: ‘[‘ expression ’]’ message-label
  - The message is sent only if the condition is true

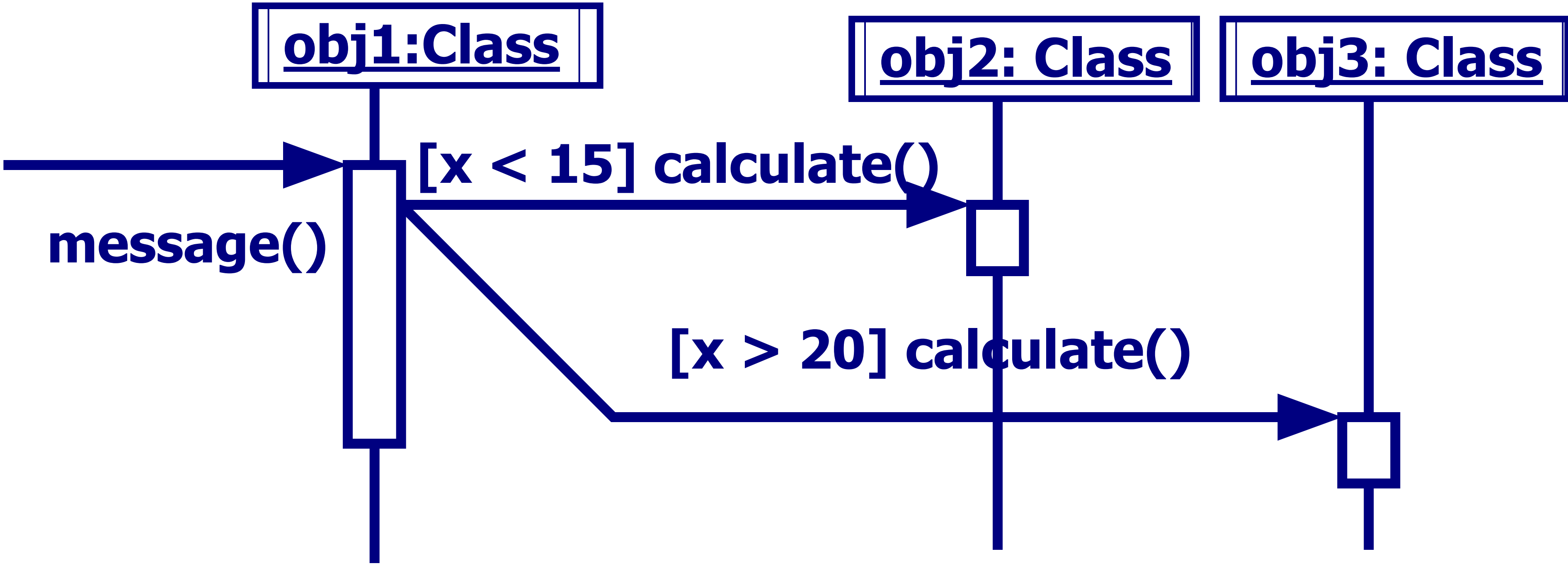


[ok] borrow(member)

Elements of Sequence Diagram

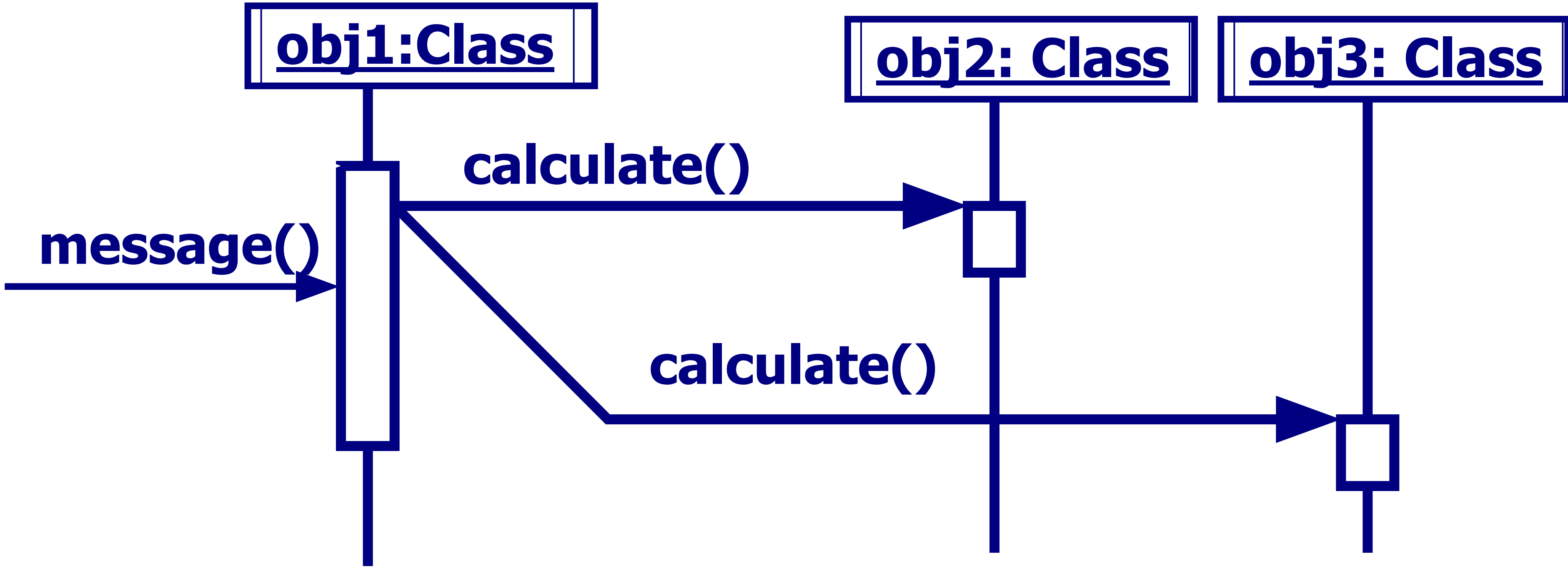


Sequence Diagrams



Sequence Diagrams

Concurrency

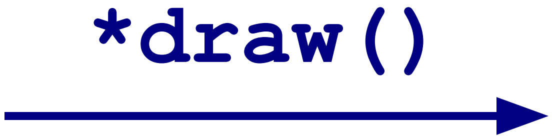


Elements of Sequence Diagram

- Control information
  - Iteration
    - may have square brackets containing a continuation condition (until) specifying the condition that must be satisfied in order to exit the iteration and continue with the sequence
    - may have an asterisk followed by square brackets containing an iteration (while or for) expression specifying the number of iterations

Control Information

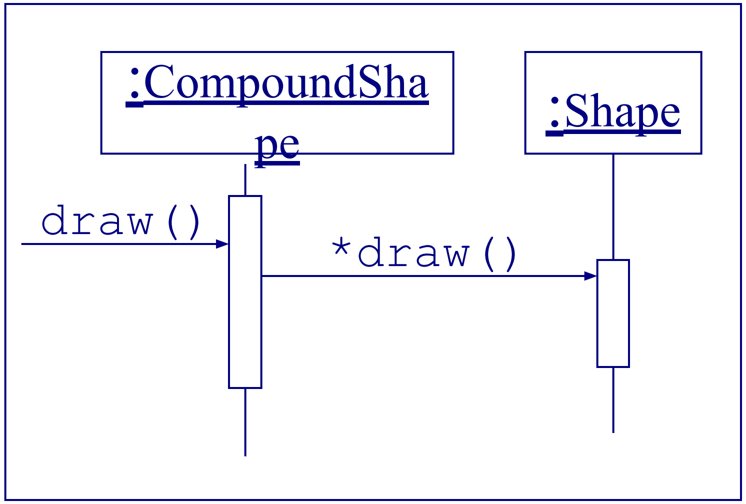
- Iteration
  - syntax: * [‘[‘ expression ‘]’] message-label
  - The message is sent many times to possibly multiple receiver objects.

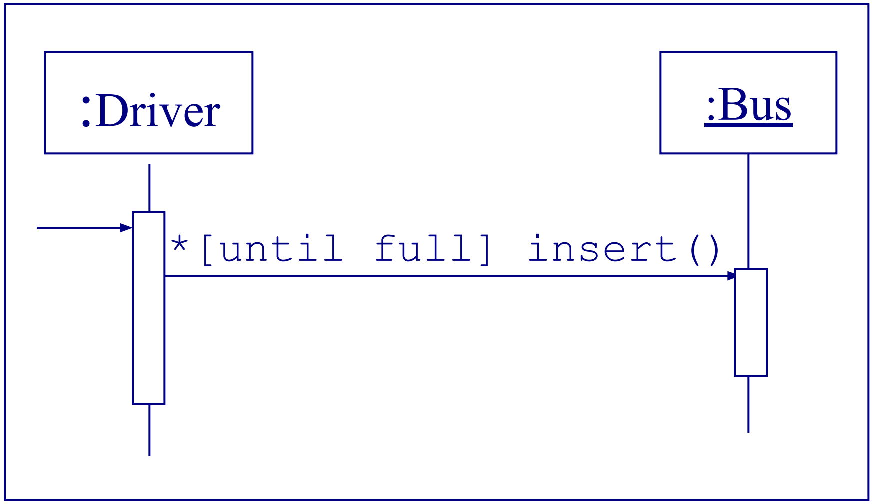


***draw()**

Control Information

- Iteration example



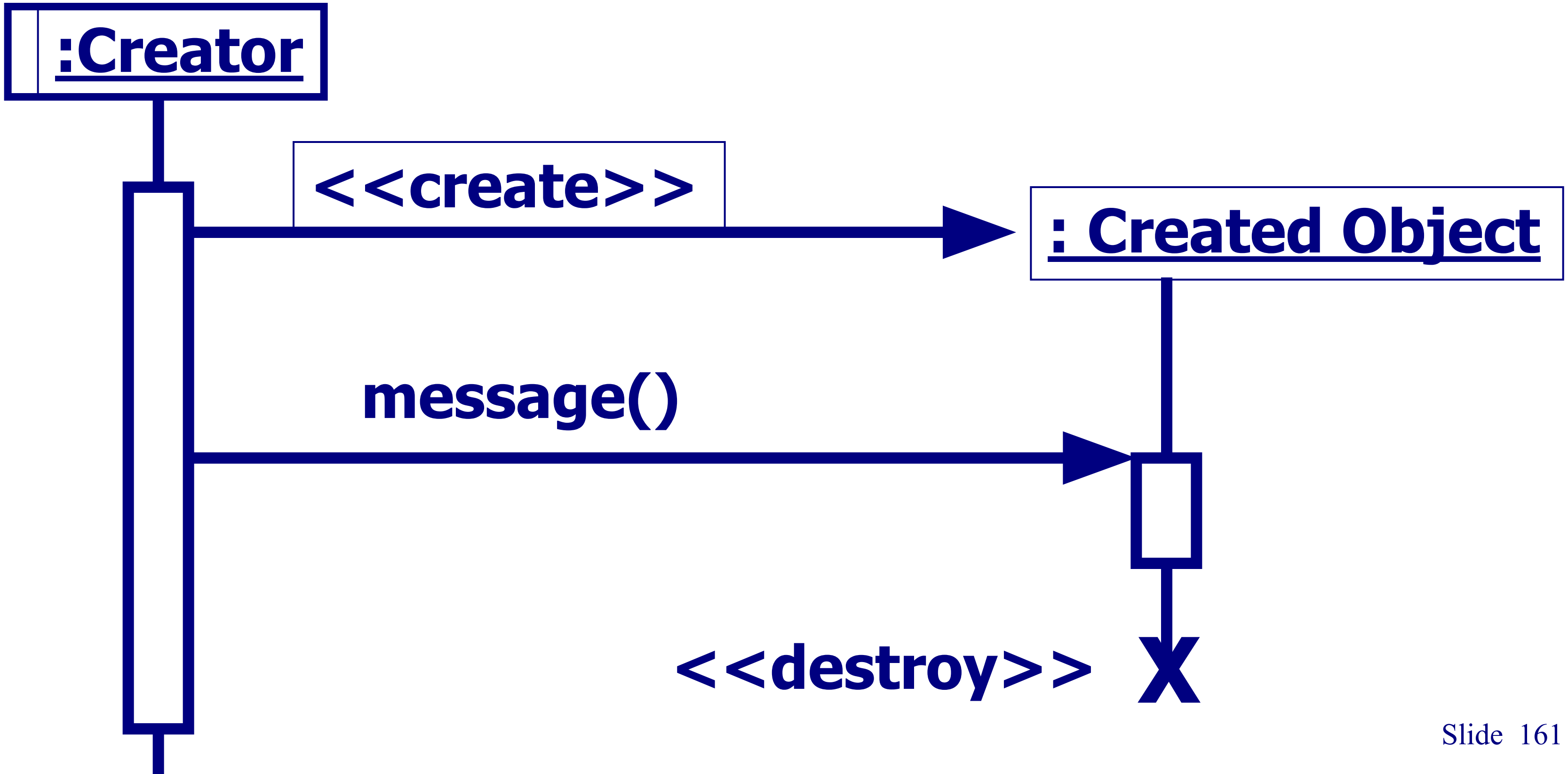


Control Information

- 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 Diagrams

- Creation and destruction of an object in sequence diagrams are denoted by the stereotypes <<create>> and <<destroy>>

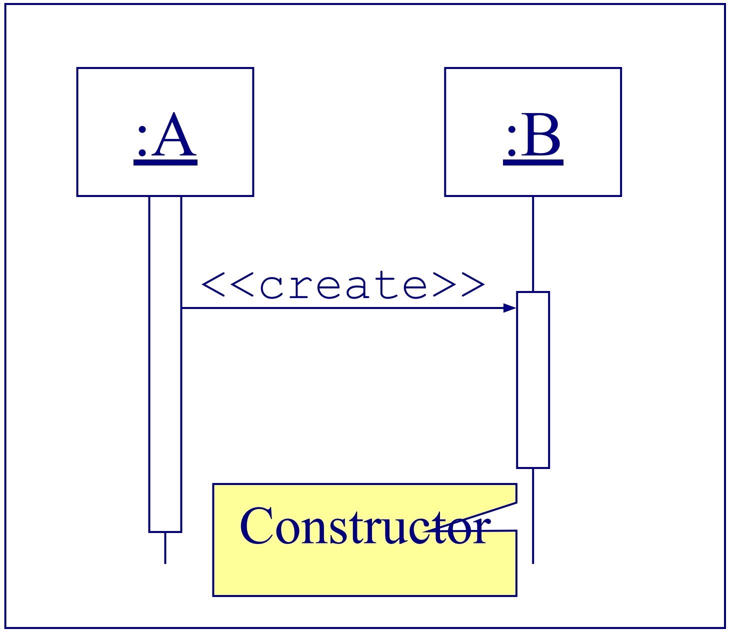


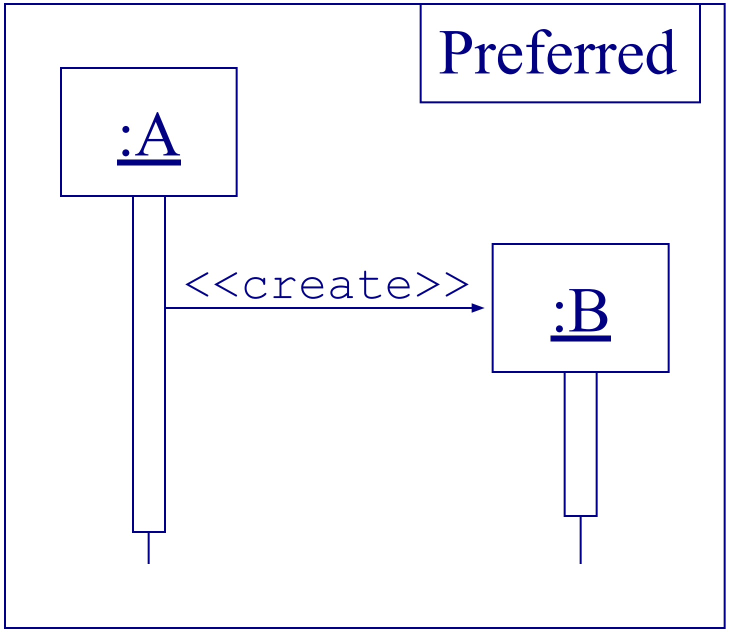
Creating Objects

- Notation for creating an object on-the-fly
  - Send the <<create>> message to the body of the object instance
  - Once the object is created, it is given a lifeline.
    - Now you can send and receive messages with this object as you can any other object in the sequence diagram.

Object Creation

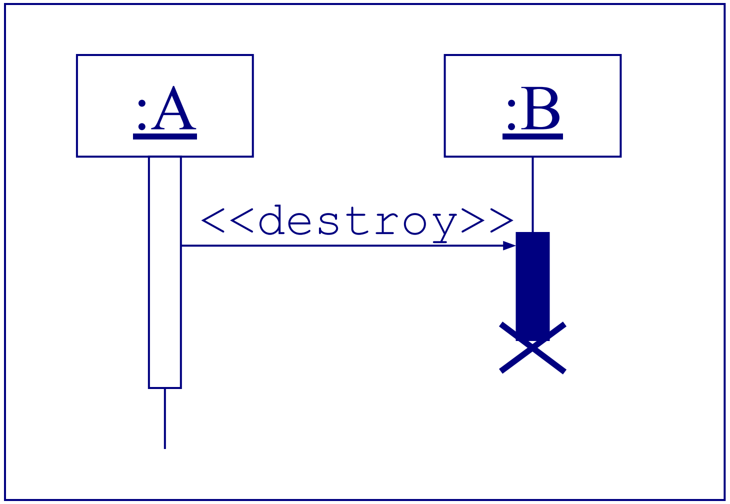
- An object may create another object via a `<<create>>` message.



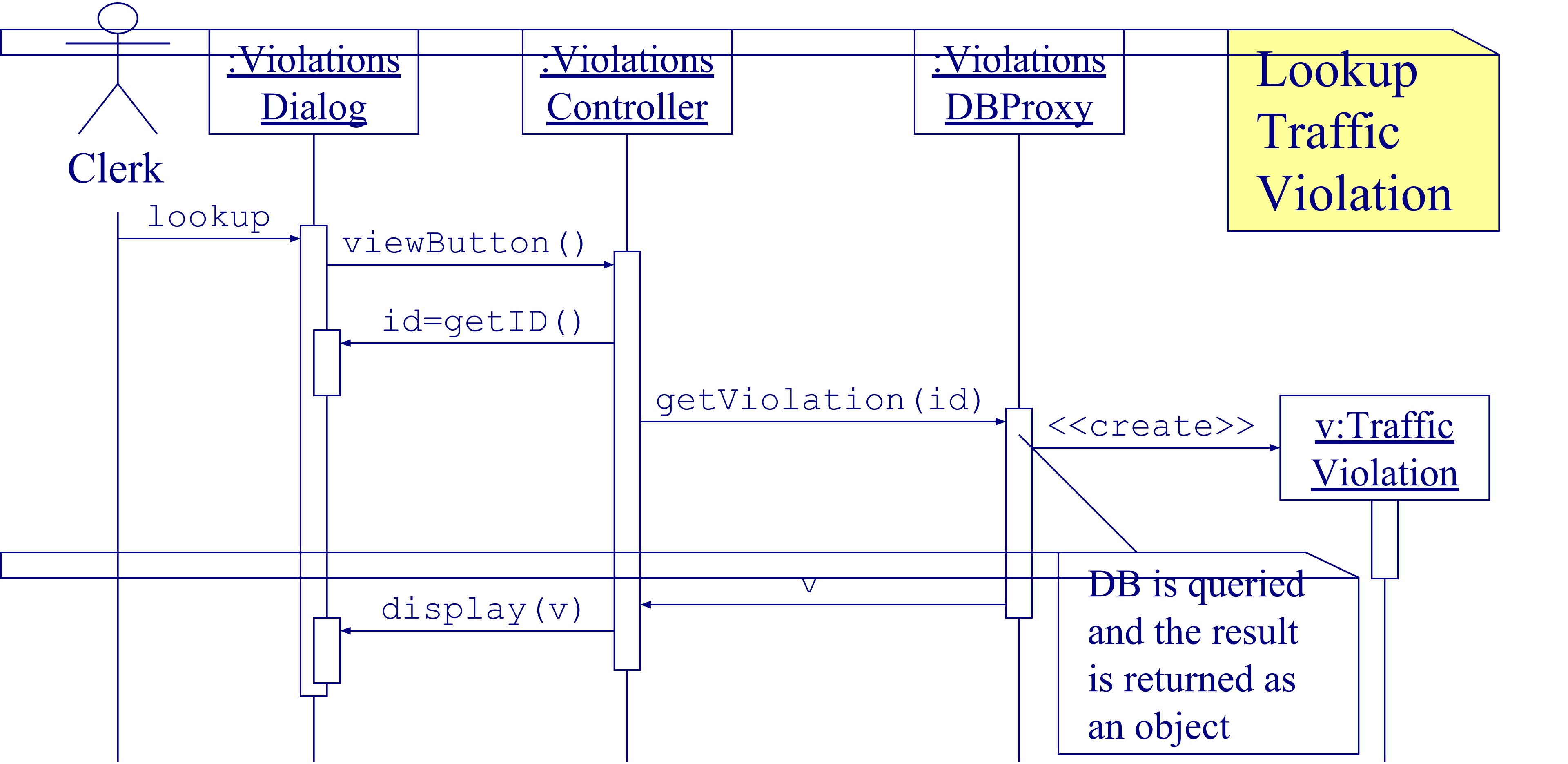


Object Destruction

- 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



Steps for Building a Sequence Diagram

1) Set the context
2) Identify which objects and actors will participate
3) Set the lifeline for each object/actor
4) Lay out the messages from the top to the bottom of the diagram based on the order in which they are sent
5) Add the focus of control for each object's or actor's lifeline
6) Validate the sequence diagram

Steps for Building a Sequence Diagram

1) Set the context.
   a) Select a use case.
   b) Decide the initiating actor.

Steps for Building a Sequence Diagram

2) Identify the objects that may participate in the implementation of this use case by completing the supplied message table.

   a) List candidate objects.

      1) Use case controller class
      2) Domain classes
      3) Database table classes
      4) Display screens or reports

Steps for Building a Sequence Diagram

2) Identify the objects (cont.)

b) List candidate messages. (in message analysis table)

1) **Examine each step in the normal scenario of the use case description to determine the messages needed to implement that step.**

2) **For each step:**

   1) Identify step number.

   2) Determine messages needed to complete this step.

   3) For each message, decide which class holds the data for this action or performs this action

3) **Make sure that the messages within the table are in the same order as the normal scenario**

Steps for Building a Sequence Diagram

2) Identify the objects (cont.)

c) Begin sequence diagram construction.

1) Draw and label each of the identified actors and objects across the top of the sequence diagram.
2) The typical order from left to right across the top is the actor, primary display screen class, primary use case controller class, domain classes (in order of access), and other display screen classes (in order of access)

3) Set the lifeline for each object/actor

Steps for Building a Sequence Diagram

4) Lay out the messages from the top to the bottom of the diagram based on the order in which they are sent.

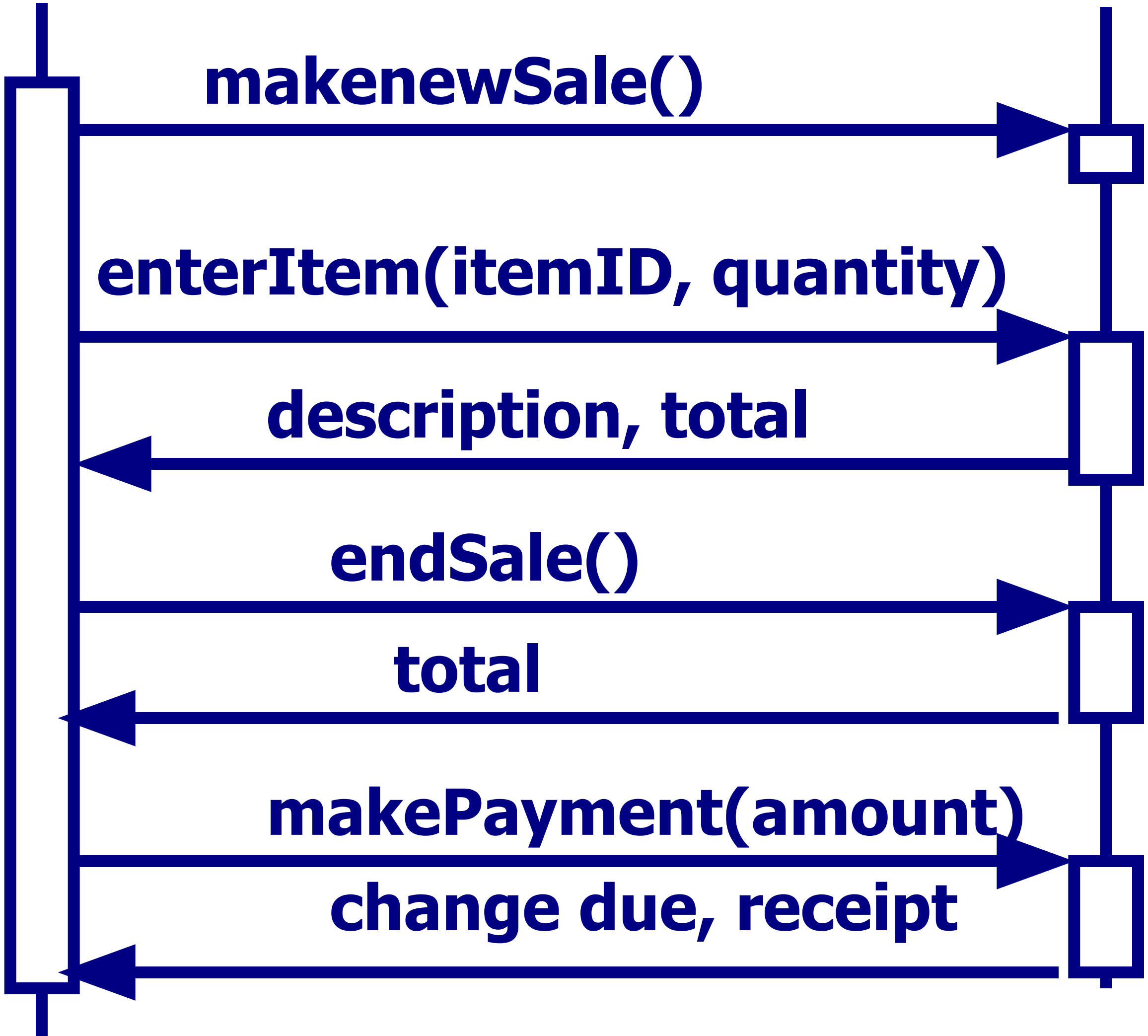
- Working in sequential order of the message table, make a message arrow with the message name pointing to the owner class.
- Decide which object or actor initiates the message and complete the arrow to its lifeline.
- Add needed return messages.
- Add needed parameters and control information.

Steps for Building a Sequence Diagram

5) Add the focus of control (activation box) for each object’s or actor’s lifeline.

6) Validate the sequence diagram.

Sequence Diagrams



makenewSale()

enterItem(itemID, quantity)

description, total

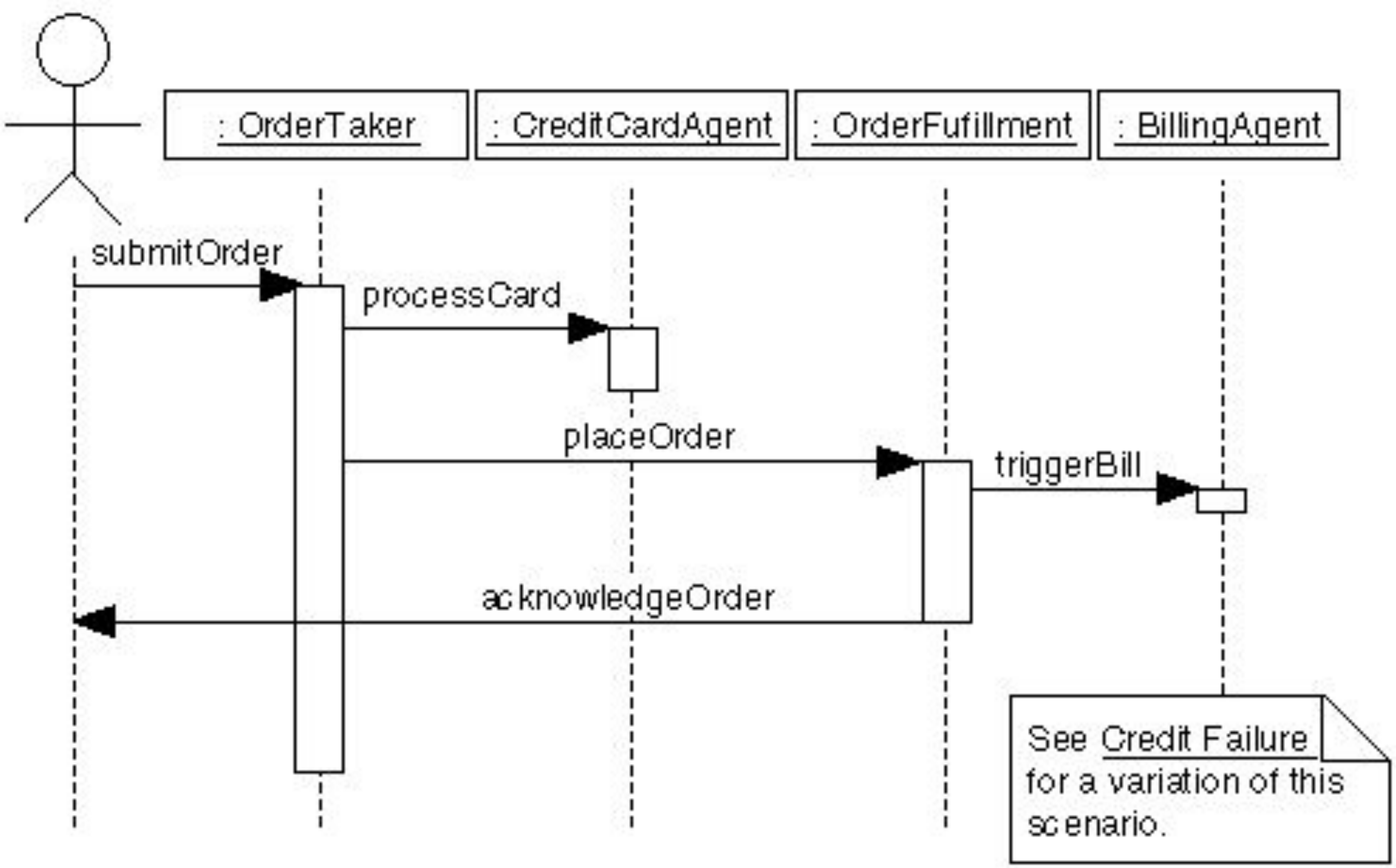
endSale()

total

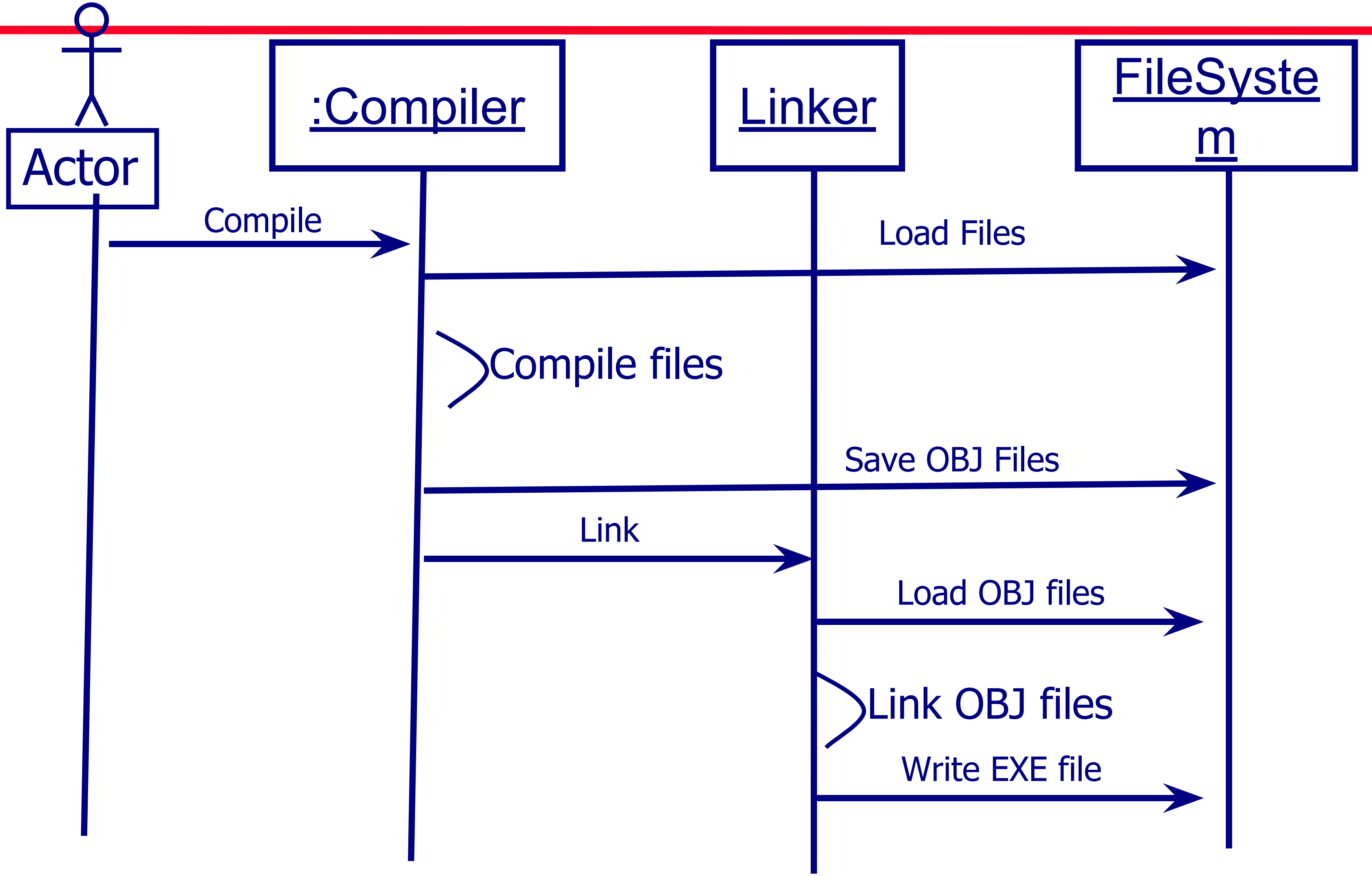
makePayment(amount)

change due, receipt

Sequence Diagram



Sequence Diagram


Actor
:Compiler
Linker
FileSystem
Compile
Load Files
Compile files
Save OBJ Files
Link
Load OBJ files
Link OBJ files
Write EXE file