Communication Diagrams

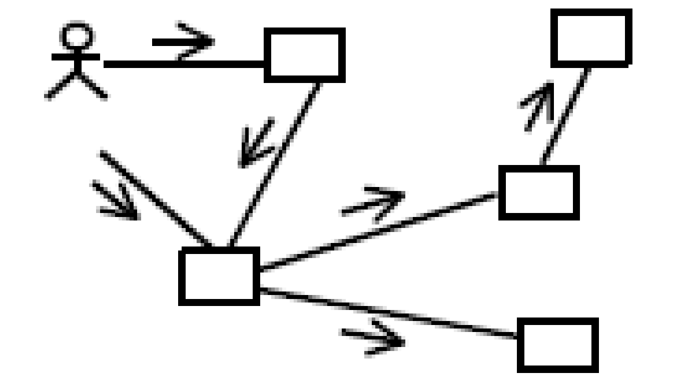
Session Outcomes

- Symbols of communication diagrams
  - Objects
  - Links
  - Messages and directions
  - Message sequence numbers

- Iteration and Looping
- Guard Expressions
- Parallel Activities

What Is a Communication Diagram?

- A communication diagram emphasizes the organization of the objects that participate in an interaction.
- The communication diagram shows:
  - The objects participating in the interaction.
  - Links between the objects.
  - Messages passed between the objects.



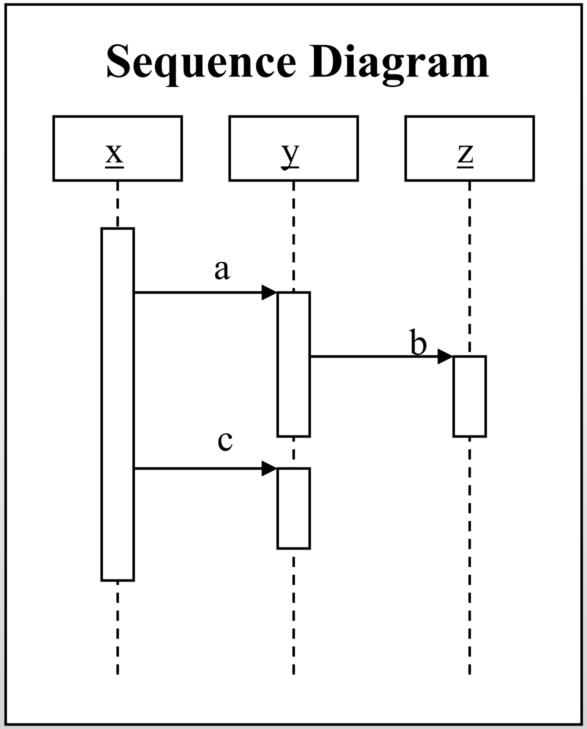
Communication Diagrams

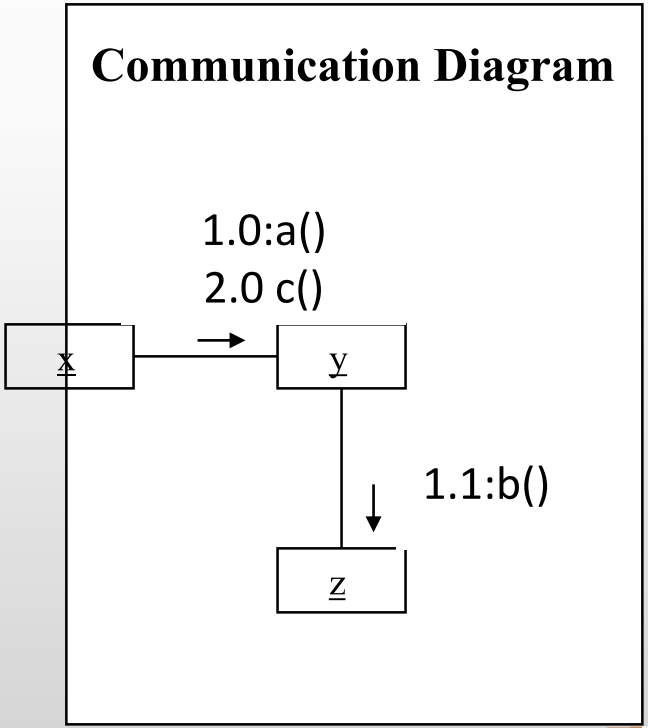
The Story so far

- To draw interaction diagrams:
  - Examine the **use case scenario** to identify the **actors**.
  - Examine the **class diagram** to identify the **objects**.
  - Draw the actors and class instances.
  - Identify the **messages/method invocations** needed to realize the scenario:
    - in general, **each step** in the scenario will require **one or more messages** to realize it;
    - an **arrow** is drawn from sending object to the receiving object.

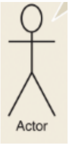
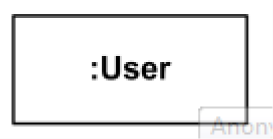
Sequence and Communication Diagrams

- Interaction diagrams
  - Sequence diagram (temporal focus)
  - Communication diagram (structural focus)

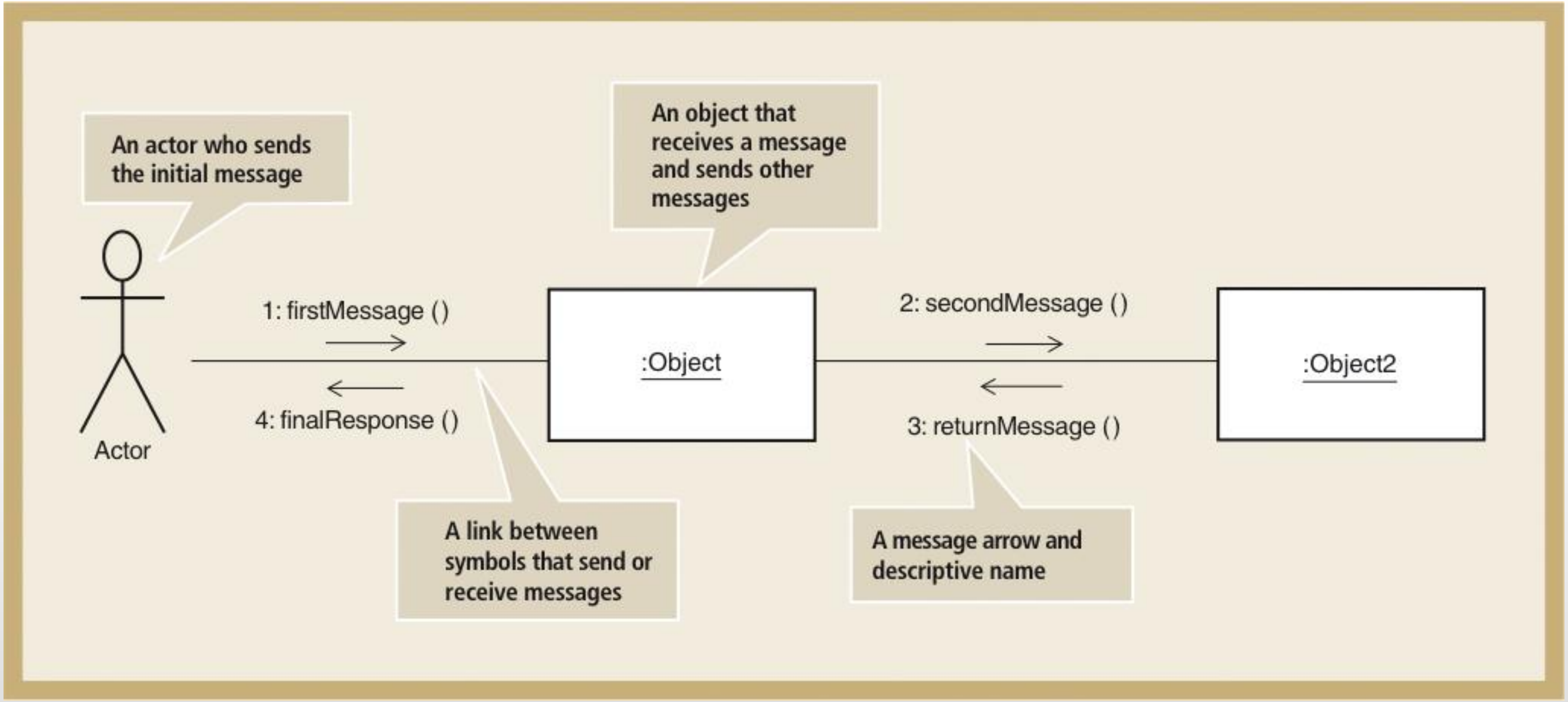




Symbols of Communication Diagrams

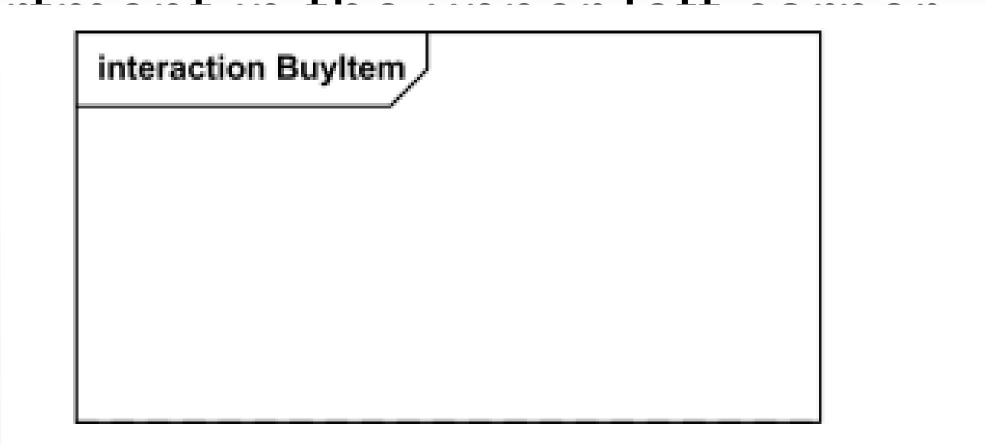
	Actors : Each Actor is named and has a role	
Object	Placed anywhere	
———	Links between objects	
→	Direction of messages from one object to another object.	
1, 2, 2.1, 2.2	Message sequence numbers.	

Symbols of Communication Diagram


An actor who sends the initial message
An object that receives a message and sends other messages
1: firstMessage ()
:Object
2: secondMessage ()
:Object2
4: finalResponse ()
3: returnMessage ()
Actor
A link between symbols that send or receive messages
A message arrow and descriptive name

Frame

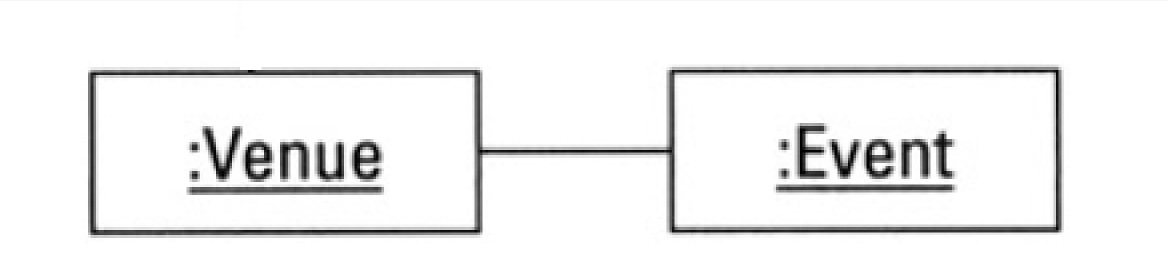
- Communication diagrams could be shown within a rectangular **frame** with the **name** in a compartment in the upper left corner



Interaction ***Frame*** *for* ***Communication Diagram*** *BuyItem*

Objects and Links

- Objects : Similar to Sequence Diagram.
- The connecting lines drawn between objects are links.
- They enable you to see the relationships between objects.
- This symbolizes the ability of objects to send messages to each other.
- A single link can support one or more messages sent between objects



Messages

- The message types in a Communication diagram are the same as in a Sequence diagram.
- A communication diagram may model synchronous, asynchronous, return, lost, found, and object-creation messages.

Message Syntax

- sequence-term [' . ' sequence-term] ... ' : message signature
- sequence-term := integer

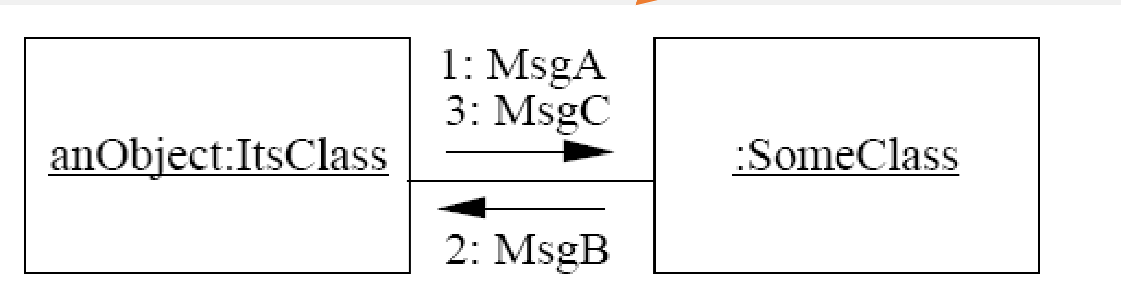
e.g. 1.0 : Login (UserName, pwd)

3.1.1 : getPerformance ( )

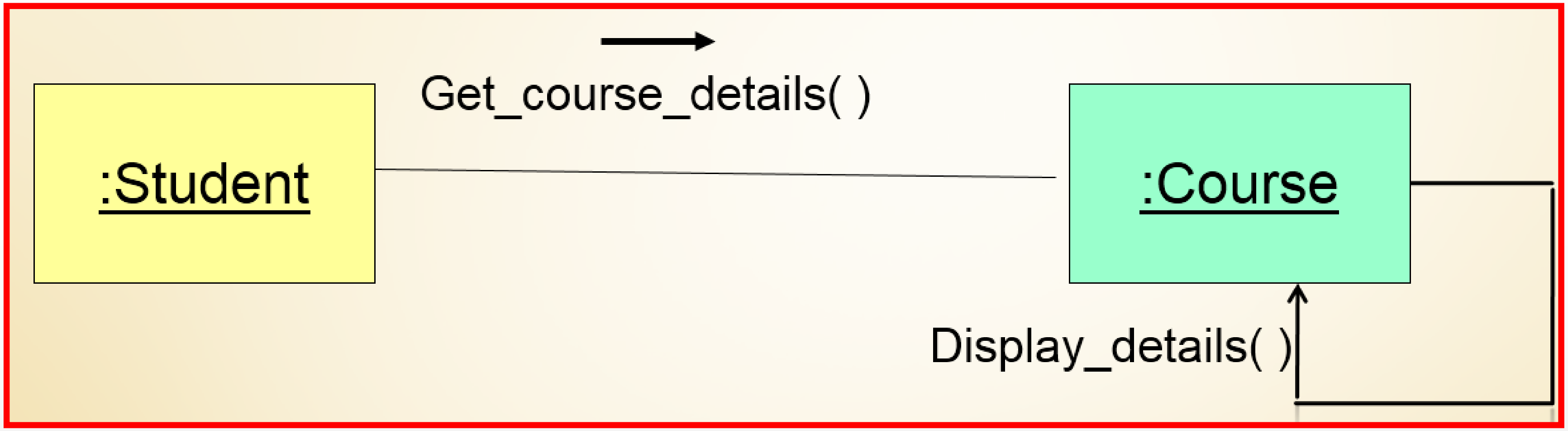
Message directions

- A message on a communication diagram is shown using an arrow from the message sender to the message receiver.

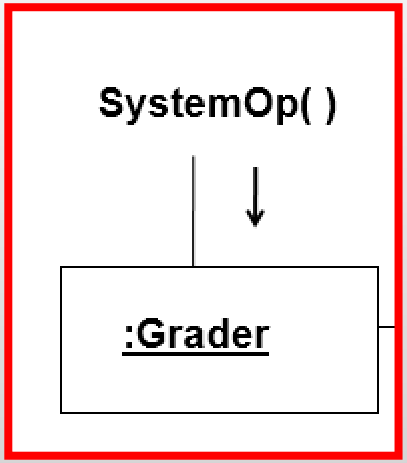
Message directions



How to show Self calls?



OR



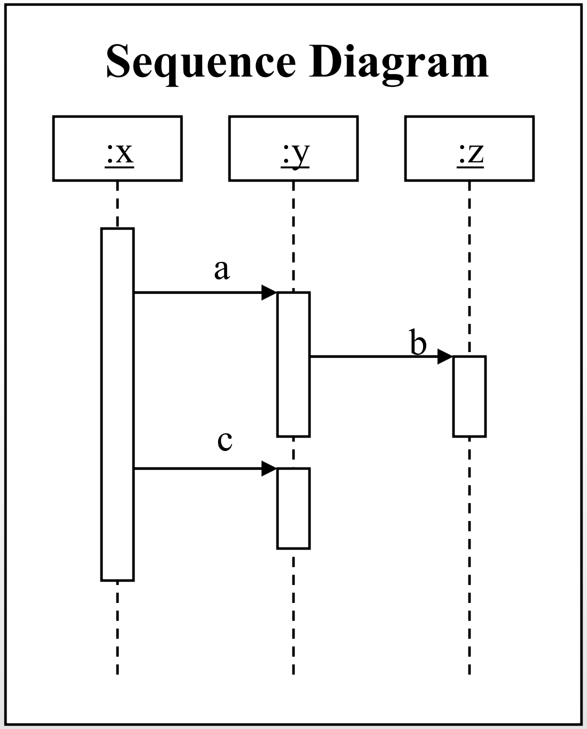
Steps to Creating a Communication Diagram

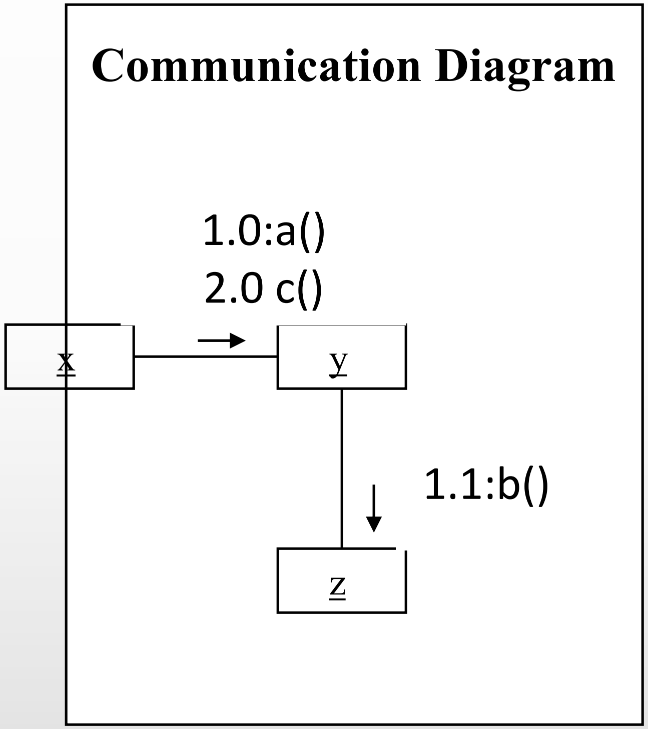
1. Place the objects that participate in the communication on the diagram
   - Remember to place the most important objects towards the center of the diagram.
2. Create links between the objects
3. Create messages associated with each link
4. Add sequence numbers to each message corresponding to the time-ordering of messages in the collaboration

Message Sequence Numbers

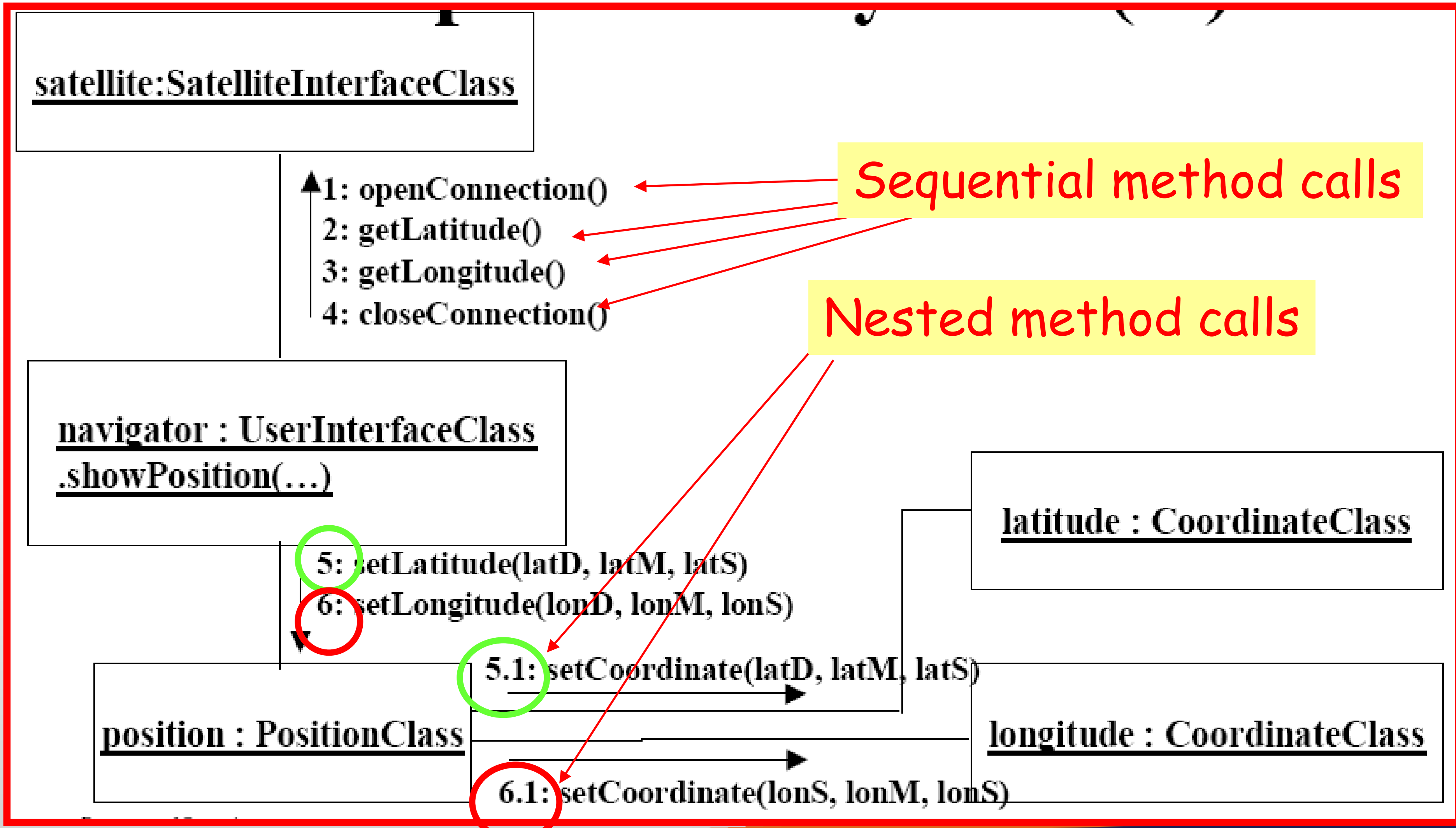
- The **integer** represents the **sequential order** of the message.
- Numbering provides **a check on flow of control**.
- Each sequence term represents a level of procedural nesting.
- If message sequence numbers are at the same dot-level-1.1 and 1.2, those messages are considered to be sequential. There is no nesting.
- If the model adds steps 1.1.1 and 1.1.2, then these new steps are understood to execute after step 1.1 and before step 1.2. In other words, they are nested beneath/within step 1.1.

Message Numbering – example 1

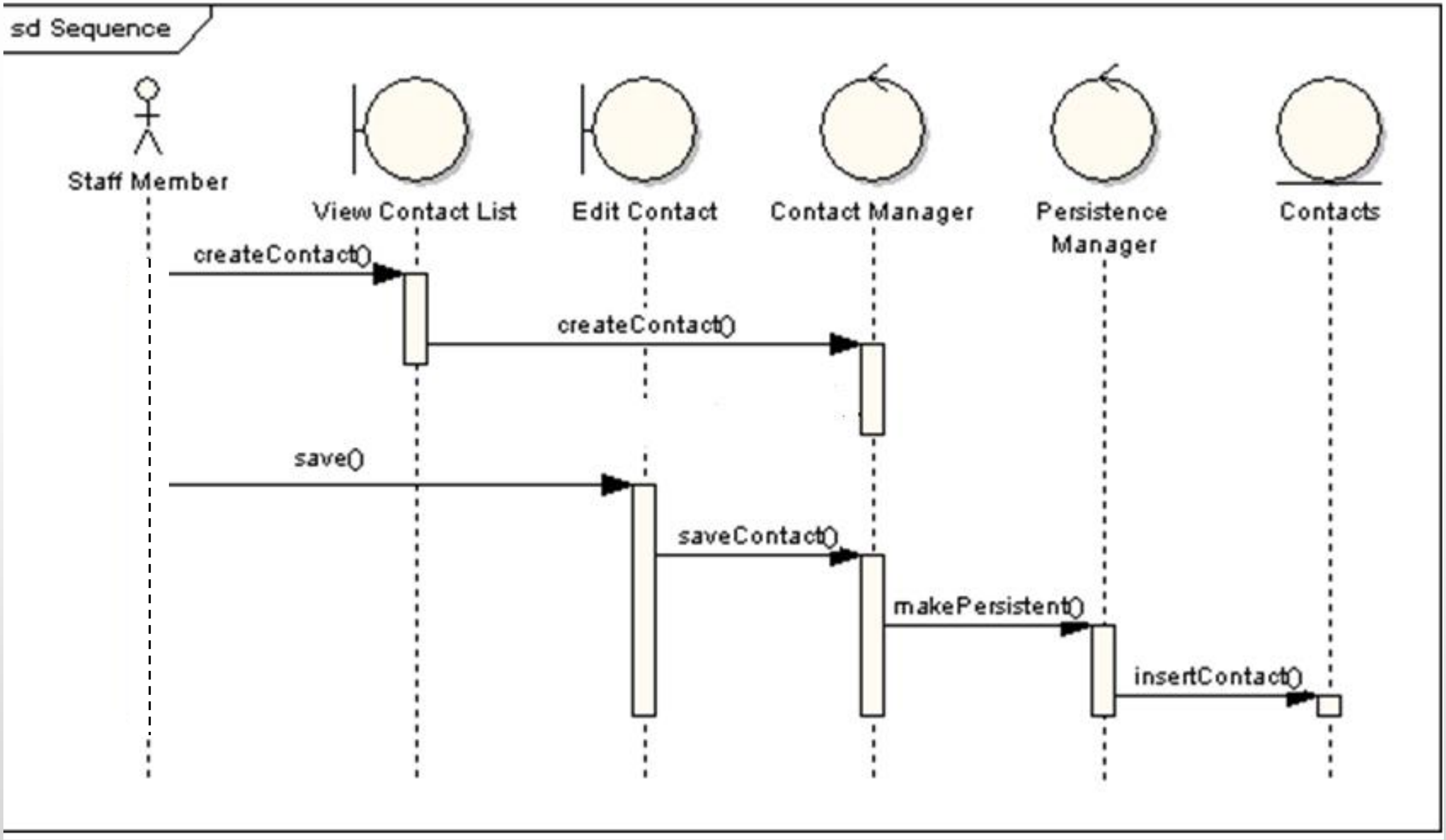




Message Numbering example 2

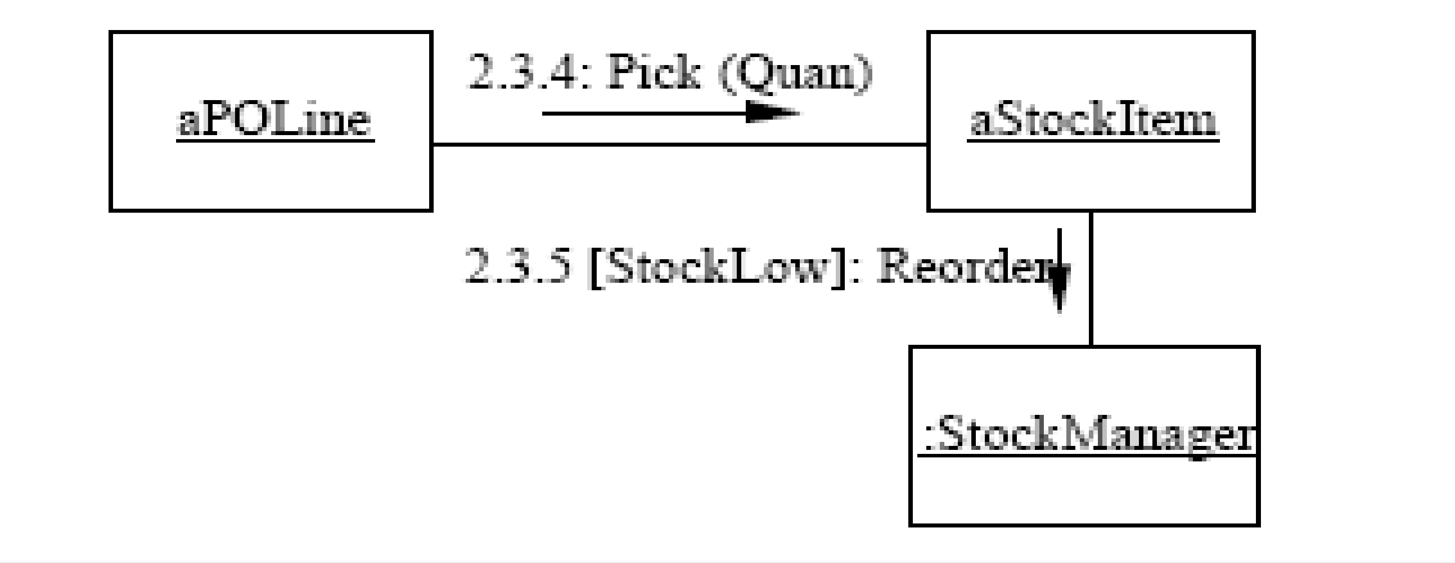

satellite:SatelliteInterfaceClass
1: openConnection()
2: getLatitude()
3: getLongitude()
4: closeConnection()
Sequential method calls
Nested method calls
navigator : UserInterfaceClass .showPosition(…)
latitude : CoordinateClass
5: setLatitude(latD, latM, latS)
6: setLongitude(lonD, lonM, lonS)
5.1: setCoordinate(latD, latM, latS)
position : PositionClass
longitude : CoordinateClass
6.1: setCoordinate(lonS, lonM, lonS)

Activity 1 – convert to a Communication Diagram

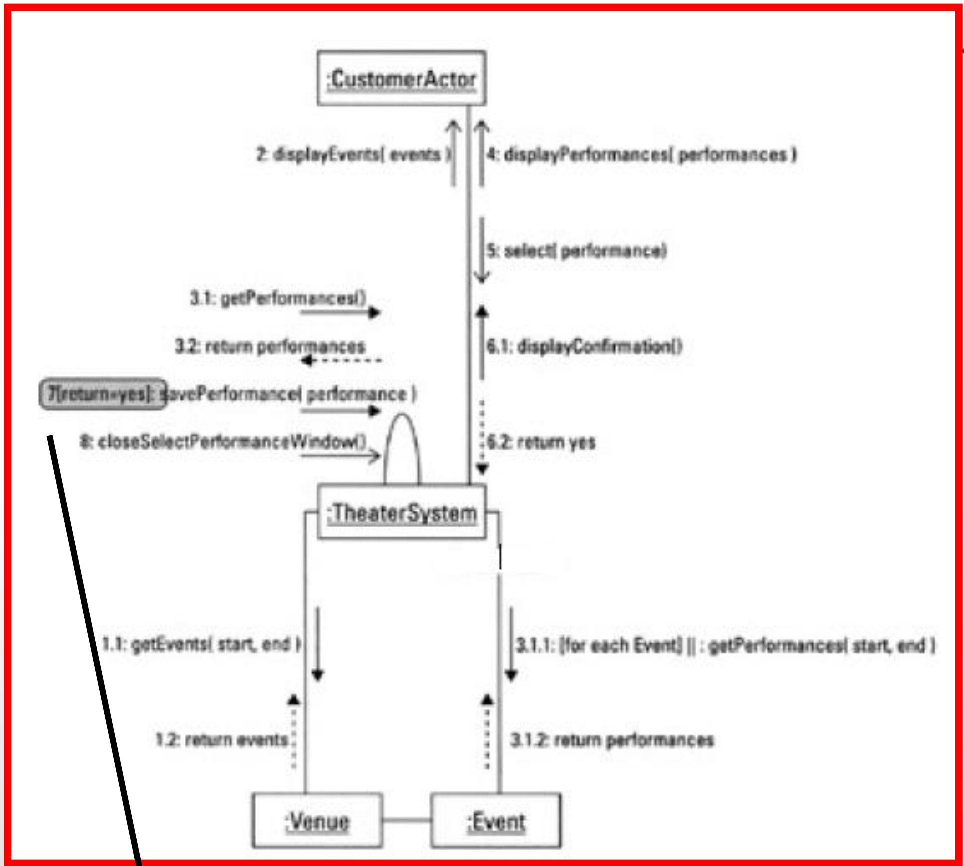


Guard Expressions

- Condition messages mean that under certain conditions the message will be sent and under some conditions it won't.
- The message is sent when the condition in the square brackets are true.



Guard Expressions example

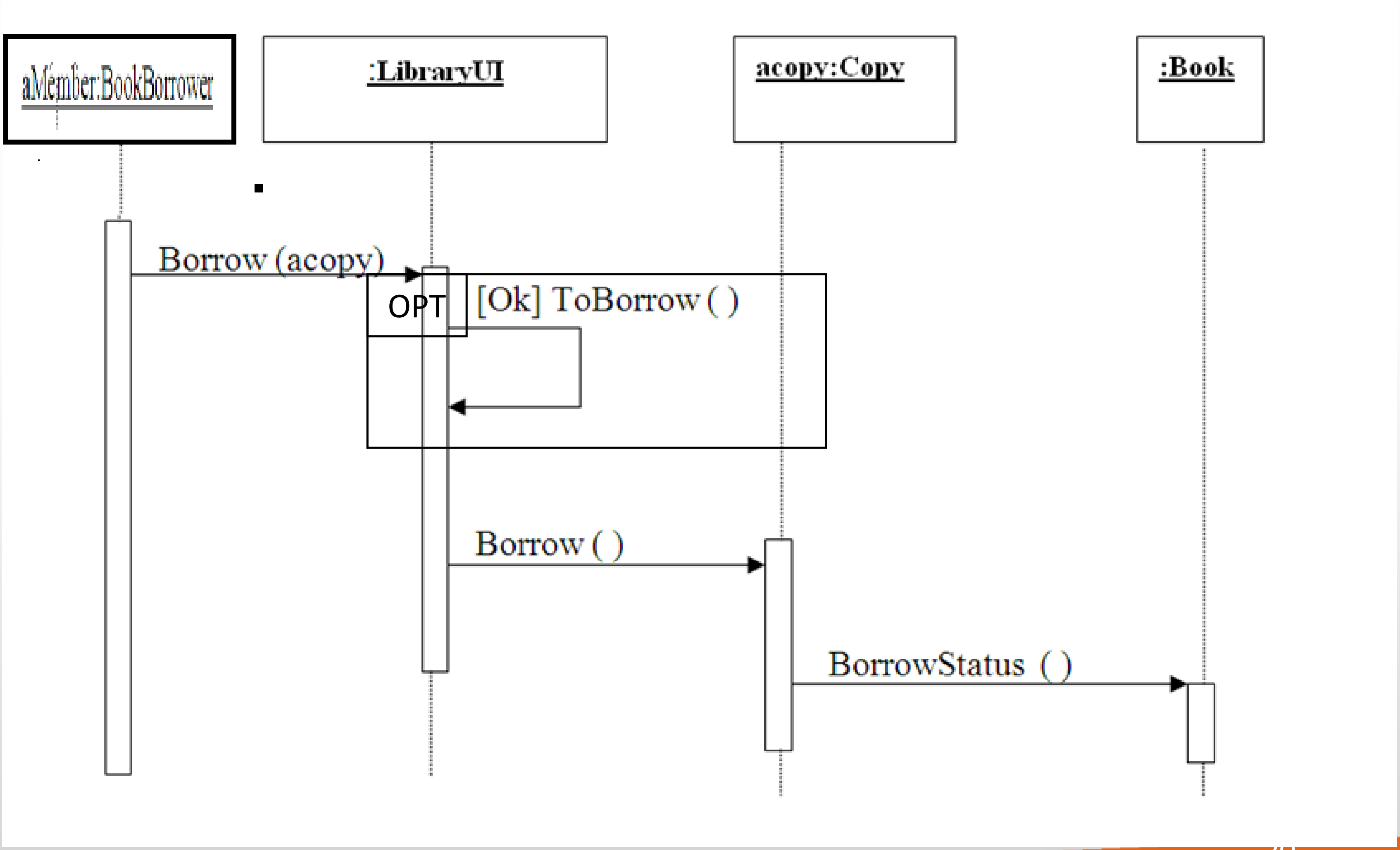

:CustomerActor
2: displayEvents(events)
4: displayPerformances(performances)
5: select(performance)
3.1: getPerformances()
3.2: return performances
6.1: displayConfirmation()
7[return=yes]: savePerformance(performance)
6.2: return yes
8: closeSelectPerformanceWindow()
:TheaterSystem
1.1: getEvents(start, end)
3.1.1: [for each Event] || : getPerformances(start, end)
1.2: return events
3.1.2: return performances
:Venue
:Event

7. [return =yes]
savePerformance (performance)

message 7 will execute only if the condition [return=yes] is satisfied.

In most cases, a Sequence or Communication diagram models a single scenario, one path through a logical sequence.

Activity 2- Convert to a Communication Diagram

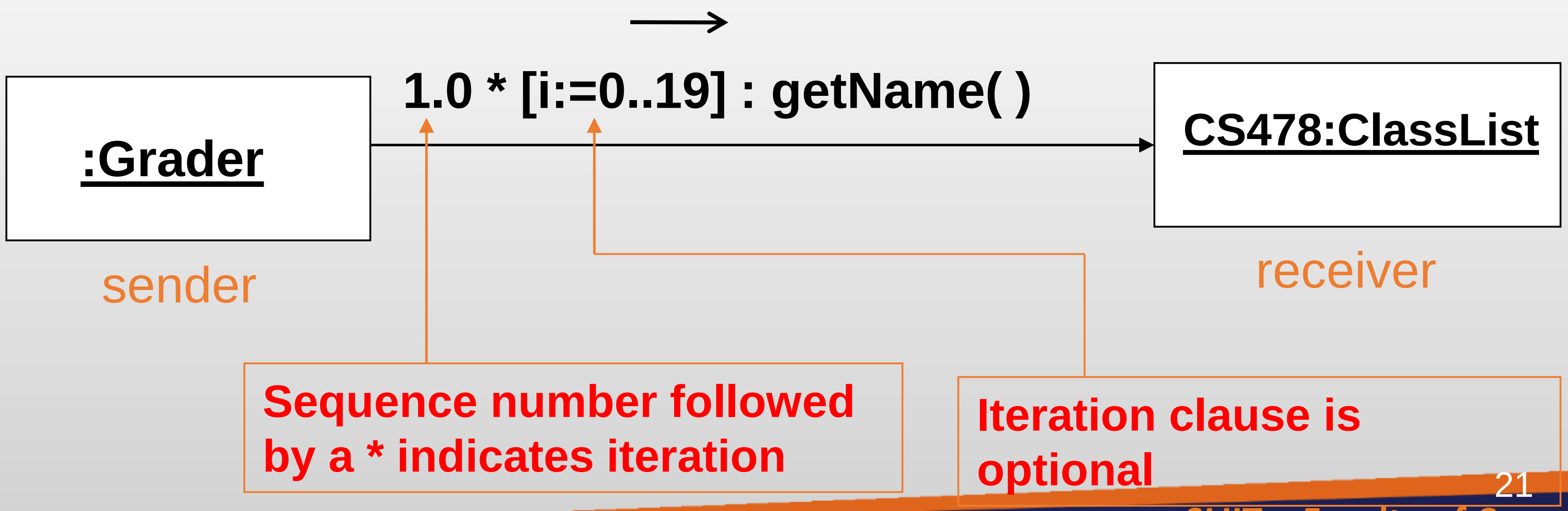


Iteration and Looping

- A message may be executed repeatedly.
- The message repeats while the condition in the square brackets is true.

*** [i := 1..N] : msg**

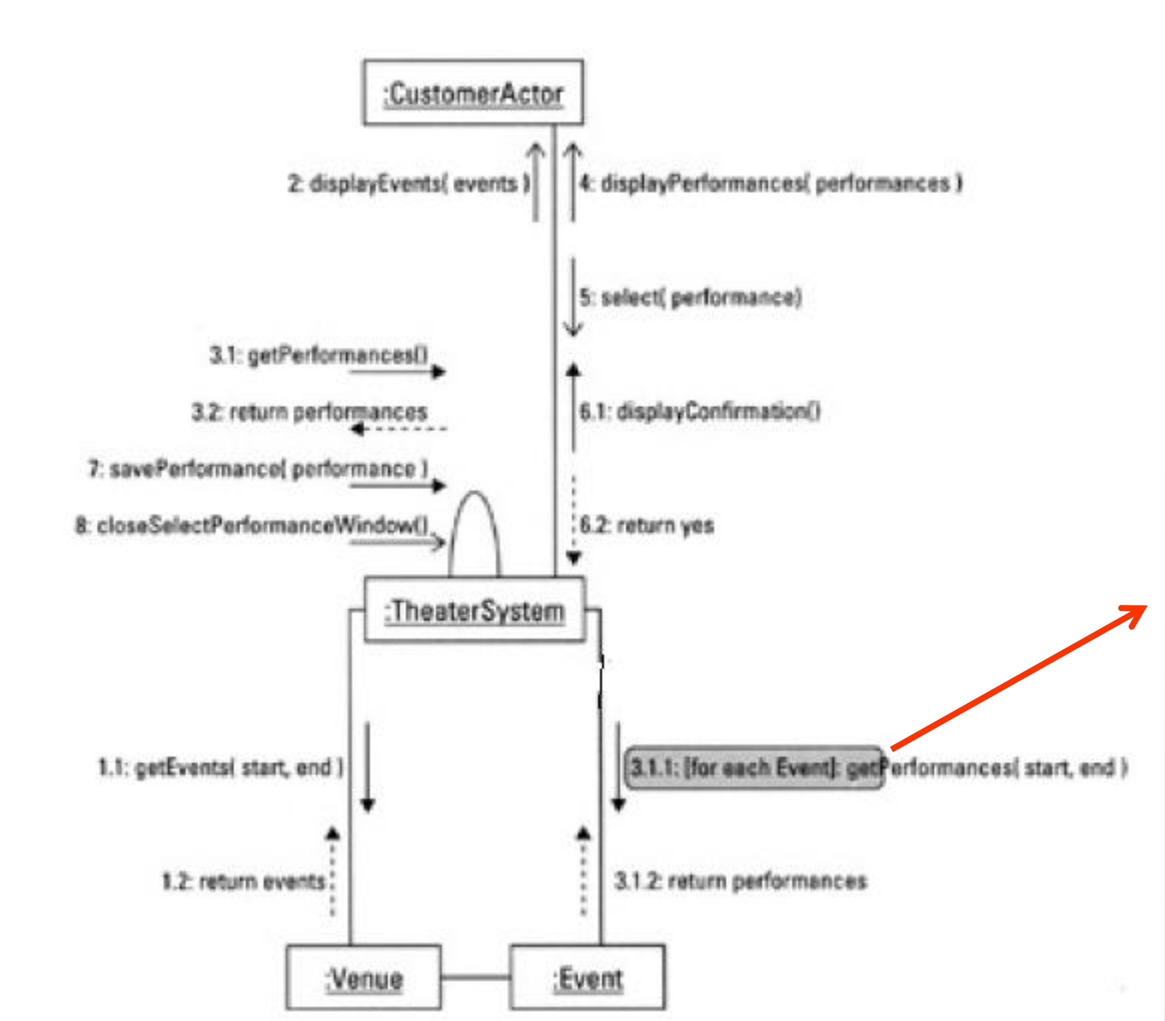
g ( ) “The asterisk (*) indicates that what follows is an iteration clause”



Sequence number followed by a * indicates iteration

Iteration clause is optional

Iteration example 2

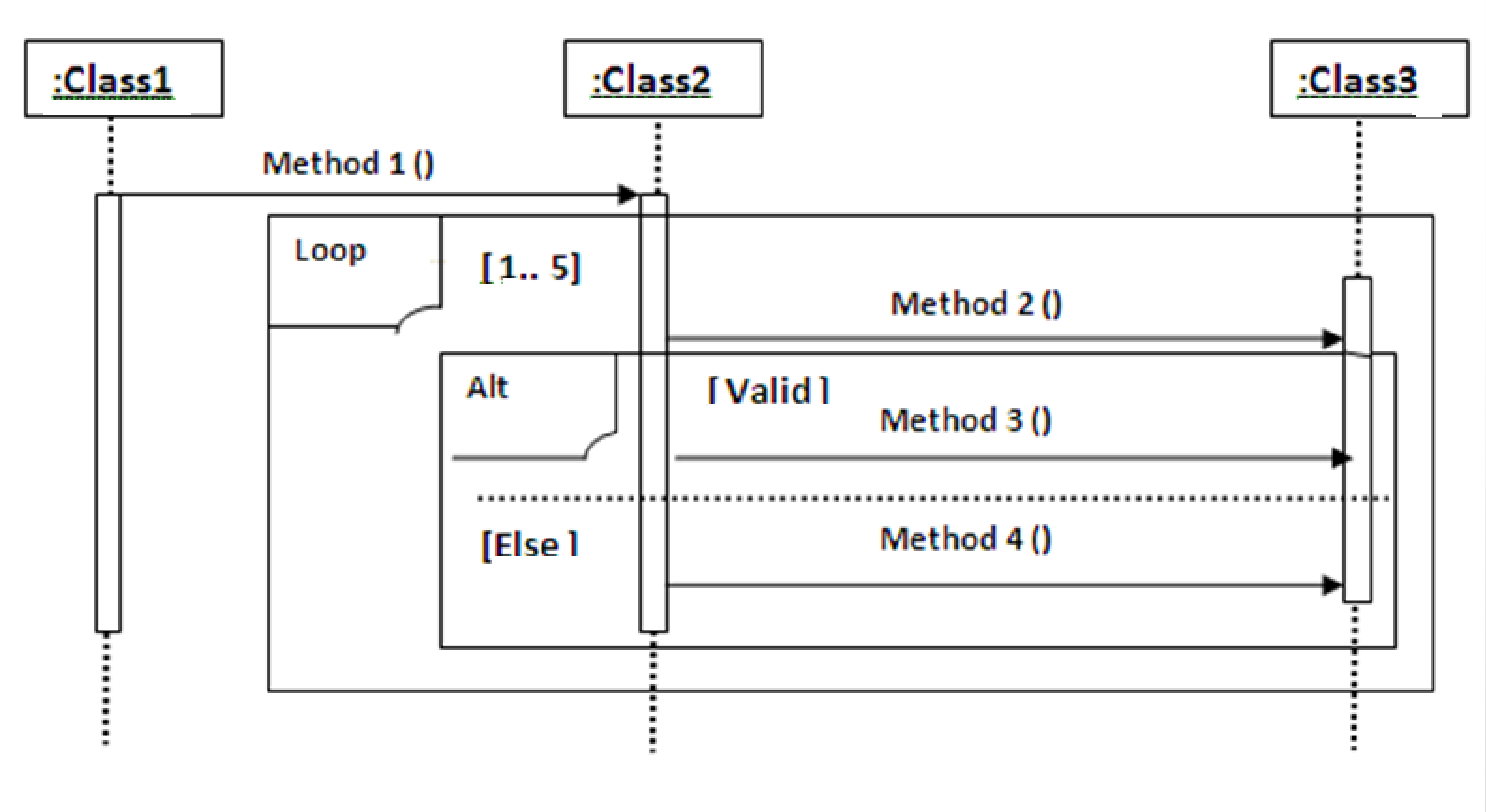

:CustomerActor
2: displayEvents(events)
4: displayPerformances(performances)
5: select(performance)
3.1: getPerformances()
3.2: return performances
6.1: displayConfirmation()
7: savePerformance(performance)
8: closeSelectPerformanceWindow()
6.2: return yes
:TheaterSystem
1.1: getEvents(start, end)
3.1.1: [for each Event]: getPerformances(start, end)
1.2: return events
3.1.2: return performances
:Venue
:Event

The interaction needs to retrieve all the
performances for each event.

The iteration expression reads *[For each event].

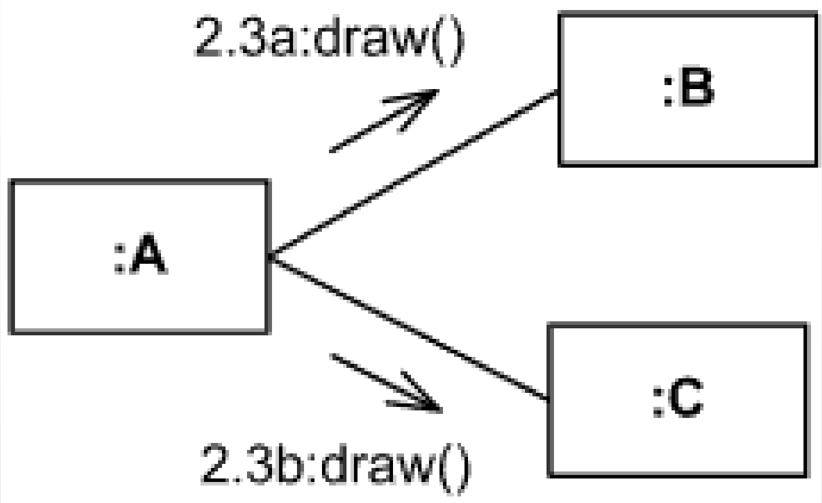
3.1.1 * [For each event] :get Performance (start,end)

Activity 3 - Convert to a Communication Diagram



Parallel Activities

Indicate concurrent threads of execution in a UML communication diagram by having letters precede the sequence numbers on messages



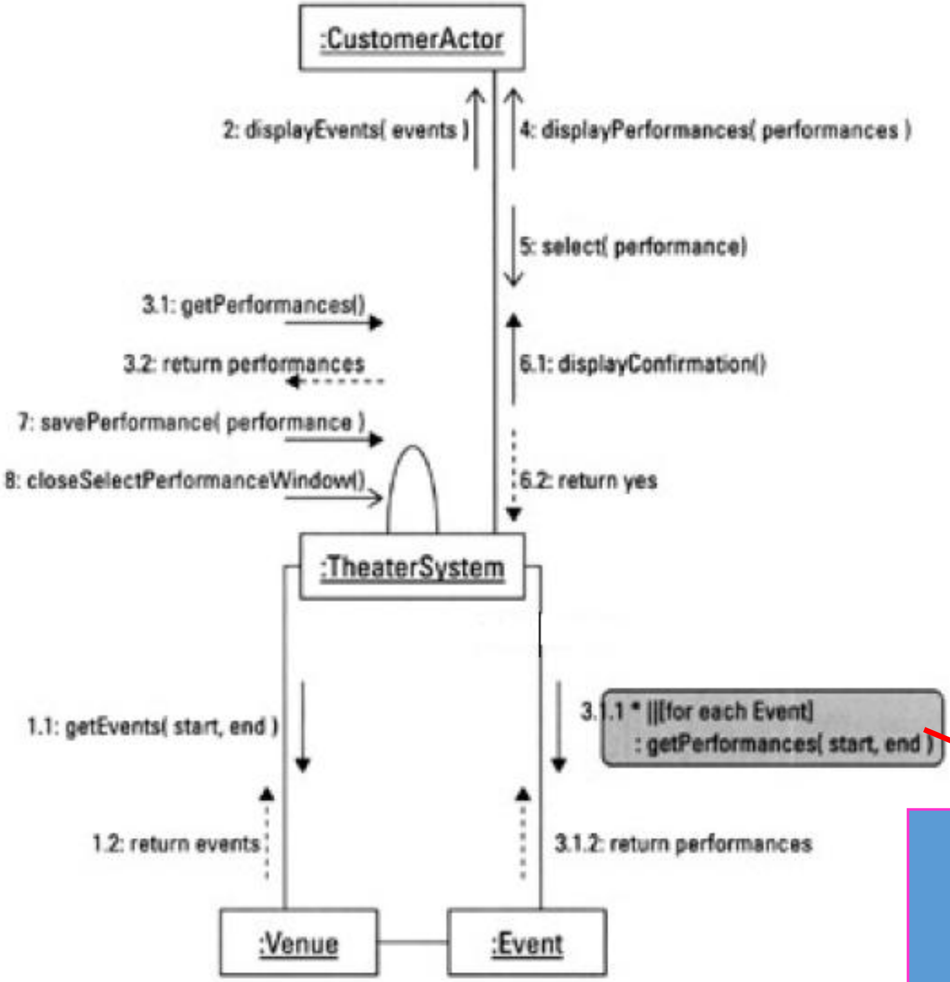
Instance of A sends draw() messages concurrently to instance of B and to instance of C

Iteration and Parallel activities

- The iteration expression assumes that the messages in the iteration will be executed sequentially. But this is not always true.

- To model the fact that the messages may execute concurrently (in parallel), use a pair of vertical lines (| |) after the iteration indicator (*).

Iteration and Parallel activities example

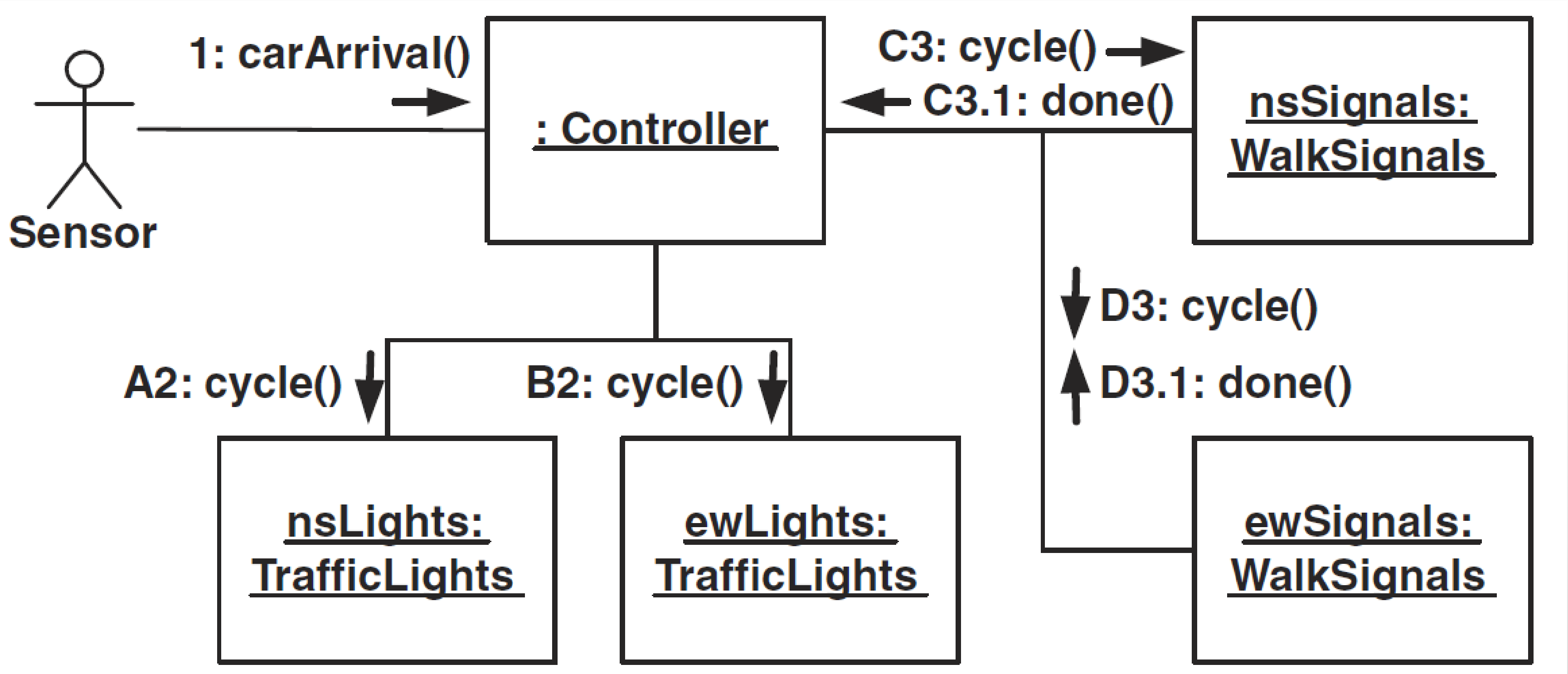

:CustomerActor
2: displayEvents(events)
4: displayPerformances(performances)
5: select(performance)
3.1: getPerformances()
3.2: return performances
6.1: displayConfirmation()
7: savePerformance(performance)
8: closeSelectPerformanceWindow()
6.2: return yes
:TheaterSystem
1.1: getEvents(start, end)
3.1.1 * ||[for each Event] : getPerformances(start, end)
1.2: return events
3.1.2: return performances
:Venue
:Event

message 3.1.1 retrieves the performances for each event, one at a time.

But we could change it to retrieve the performances for all events concurrently by adding the concurrency notation to the sequence term.

3.1.1 * | | [For each Event] : getPerformances (start, end)

Activity 4: Find the concurrent activities



Rules of Thumb

- **Avoid crossing links** and crowded diagrams.
- **Do not show all interactions** on an interaction diagram - only what is important for the scenario:
- Name Objects When You Reference Them in Messages.
- Do Not Model Obvious Return Values.
- Model a Return Value Only When You Need to Refer to It Elsewhere in a Diagram.

Sequence Diagram vs. Communication Diagram

- Sequence diagrams emphasis the **sequences of events** well.
- Communication diagrams show the **relationships between the classes** well.
- Keep both types of diagrams simple.

Strengths and Weaknesses

Type	Strengths	Weaknesses
Sequence	Show sequence or **time** order	Forced to extend to the right when adding new objects
Communication	Flexibility to add new objects in two dimensions. Better to illustrate complex branching, iteration and concurrent behavior	Difficult to see sequence of messages

Sequence and Communication Diagram Similarities

- Semantically equivalent.
- Can convert one diagram to the other without losing any information.
- Model the dynamic aspects of a system.
- Model the implementation of a use-case scenario.

References

- UML 2 Bible
    - Chapters 8 & 9
- Applying UML and Patterns by Craig Larman
    - Chapter 15
- TheElementsofUML2Style
    - Chapter 7