IT 314: Software Engg

UML Sequence Diagrams

Reading:
UML Distilled Ch. 4, by M. Fowler

1

UML sequence diagrams

- **sequence diagram**: an "interaction diagram" that models a single scenario executing in the system
  - perhaps 2nd most used UML diagram (behind class diagram)
- relation of UML diagrams to other exercises:
  - CRC cards -> class diagram
  - use cases -> sequence diagrams

2

Interaction Diagrams

- A series of diagrams describing the *dynamic behavior* of an object-oriented system.
  - A set of messages exchanged among a set of objects within a context to accomplish a purpose.
- Often used to model the way a use case is realized through a sequence of messages between objects.

Interaction Diagrams (Cont.)

- The purpose of Interaction diagrams is to:
  - Model interactions between objects
  - Assist in understanding how a system (a use case) actually works
  - Verify that a use case description can be supported by the existing classes
  - Identify responsibilities/operations and assign them to classes

Key parts of a sequence diag.

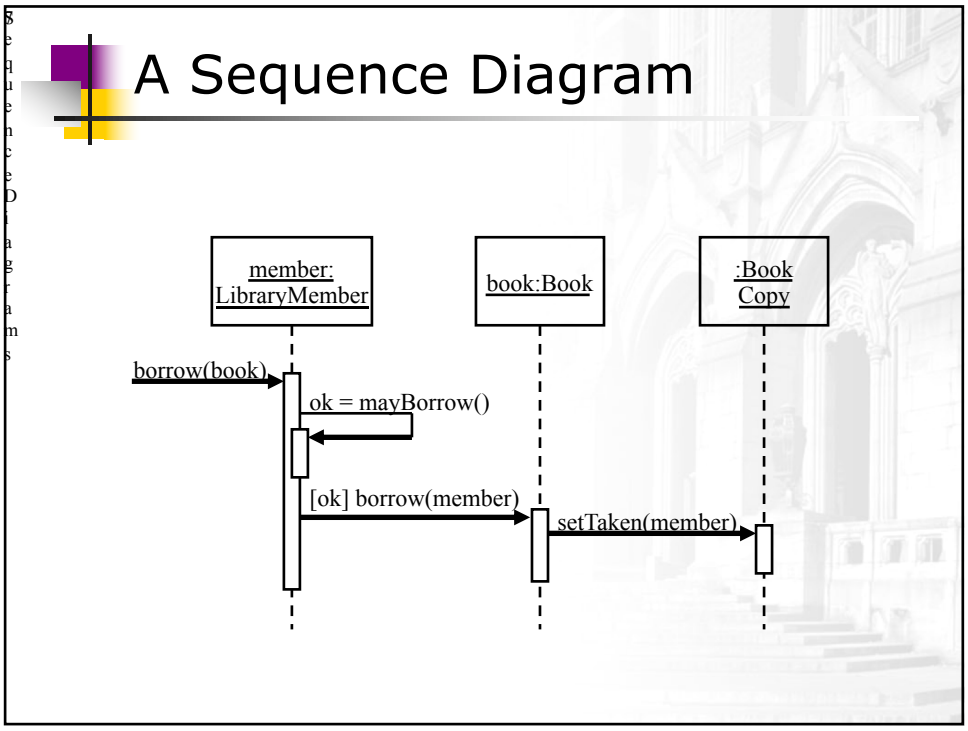
- **participant**: an object or entity that acts in the sequence diagram
  - sequence diagram starts with an unattached "found message" arrow
- **message**: communication between participant objects
- the axes in a sequence diagram:
  - horizontal: which object/participant is acting
  - vertical: time (down -> forward in time)

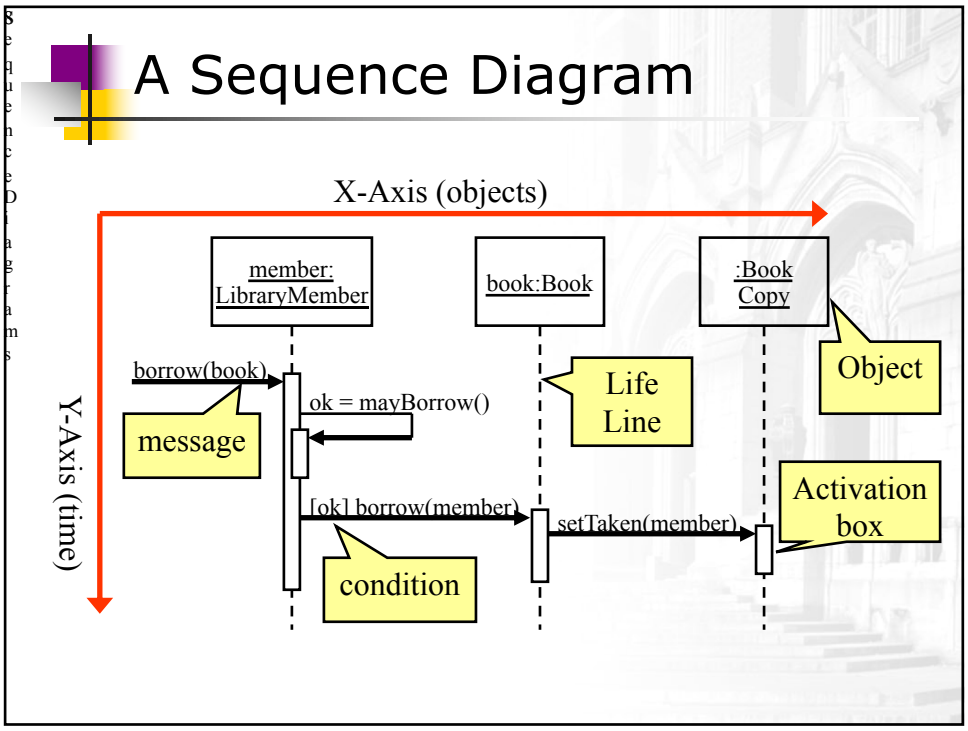
5

6

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.


Sequence Diagrams
A Sequence Diagram
member: LibraryMember
book:Book
:Book Copy
borrow(book)
ok = mayBorrow()
[ok] borrow(member)
setTaken(member)


Sequence Diagrams
A Sequence Diagram
X-Axis (objects)
Y-Axis (time)
member: LibraryMember
book:Book
:Book Copy
borrow(book)
ok = mayBorrow()
[ok] borrow(member)
setTaken(member)
message
Life Line
Object
Activation box
condition

Sequence Diagrams

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

Sequence Diagrams

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.)

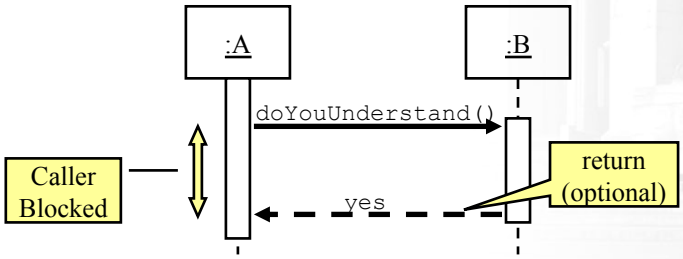
- A message is represented by an arrow between the life lines of two objects.
  - Self calls are also allowed
  - The time required by the receiver object to process the message is denoted by an *activation-box.*
- A message is labeled at minimum with the message name.
  - Arguments and control information (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()`

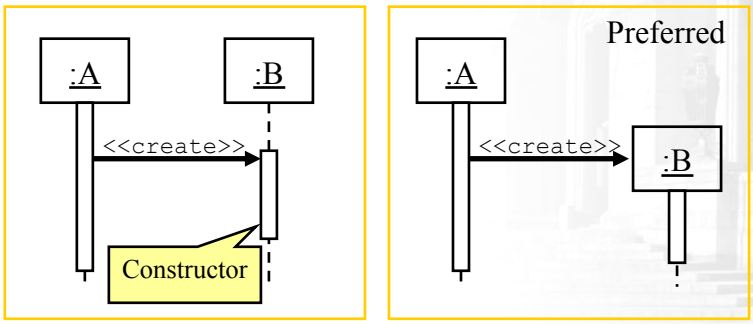
Synchronous Messages

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



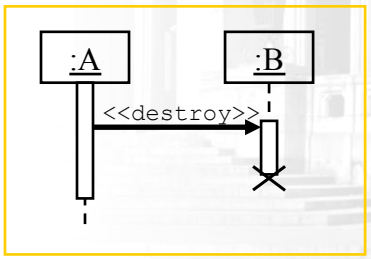
Object Creation

- An object may create another object via a message.



Object Destruction

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

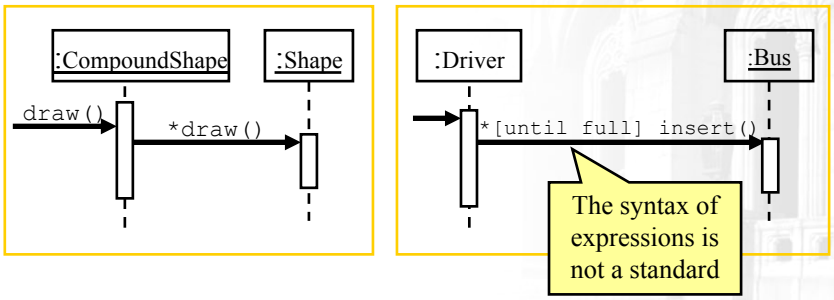


Control information

- Condition
  - syntax: '[' expression ']' message-label
  - The message is sent only if the condition is true
  - example: [ok] borrow(member)
- Iteration
  - syntax: * ['[' expression ']'] message-label
  - The message is sent many times to possibly multiple receiver objects.

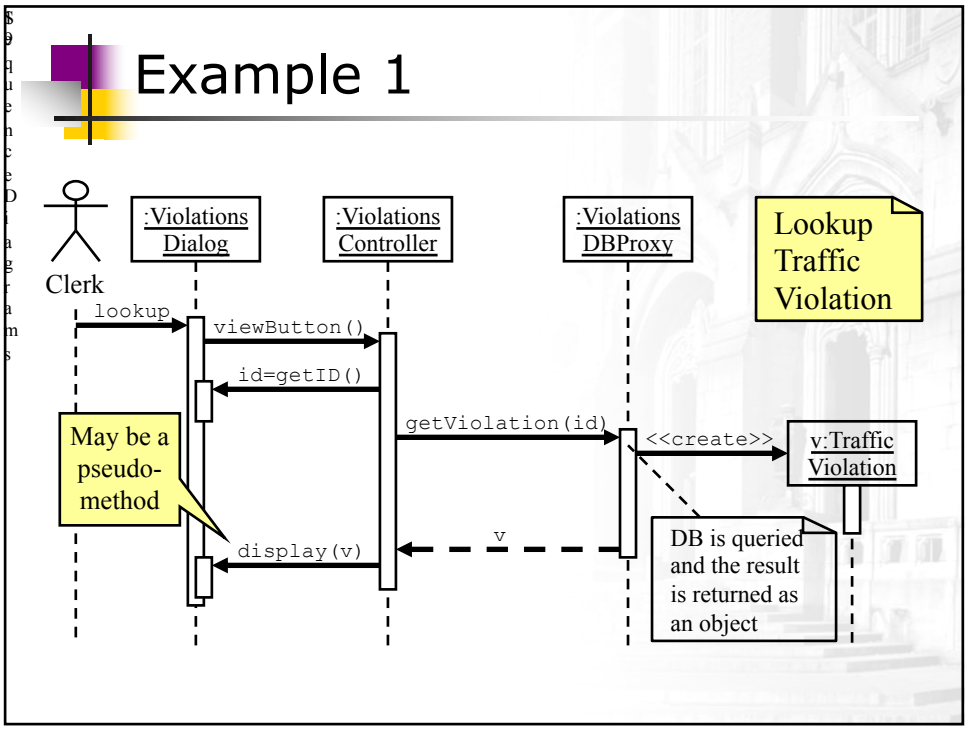
Control Information (Cont.)

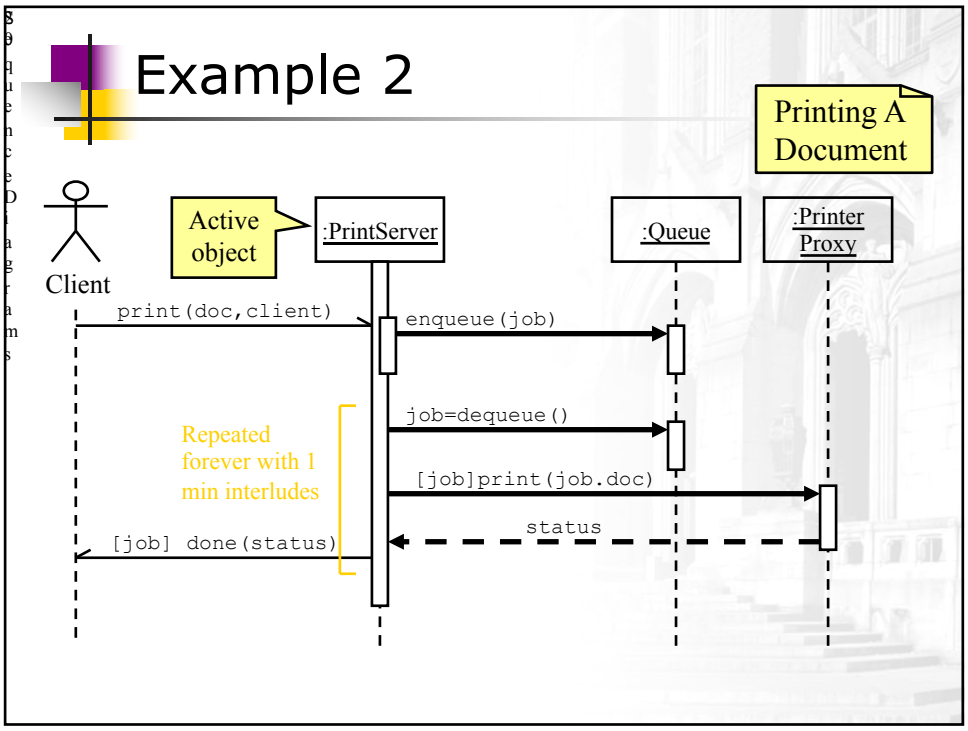
- Iteration examples:



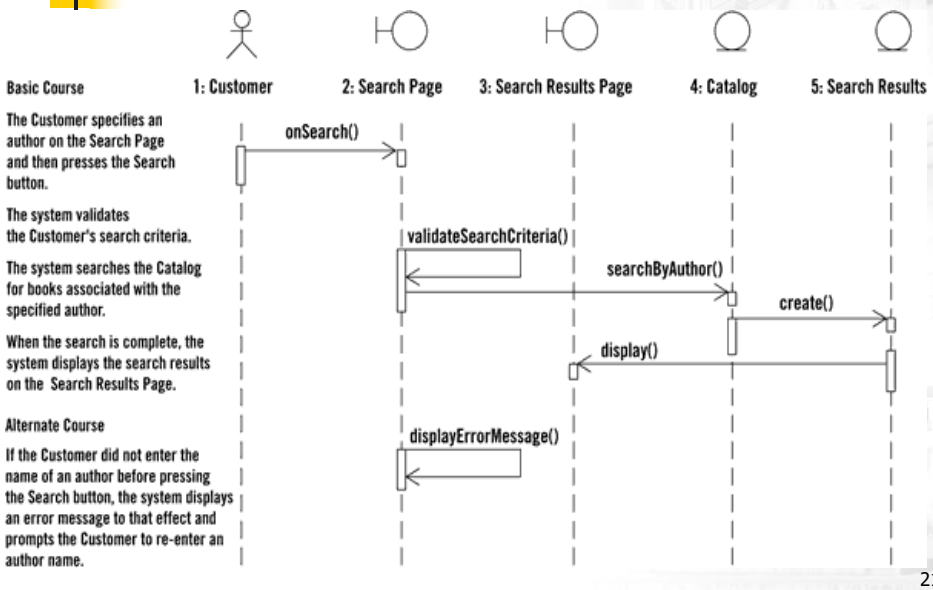
Control Information (Cont.)

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


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
display(v)
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)

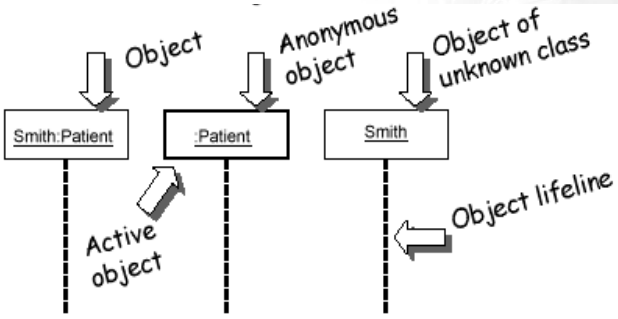
Sequence dg. from use case



21

Representing objects

- Squares with object type, optionally preceded by object name and colon
  - write object's name if it clarifies the diagram
  - object's "life line" represented by dashed vert. line

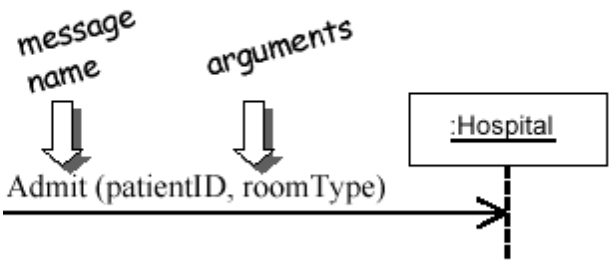


Name syntax: <objectname>:<classname>

22

Messages between objects

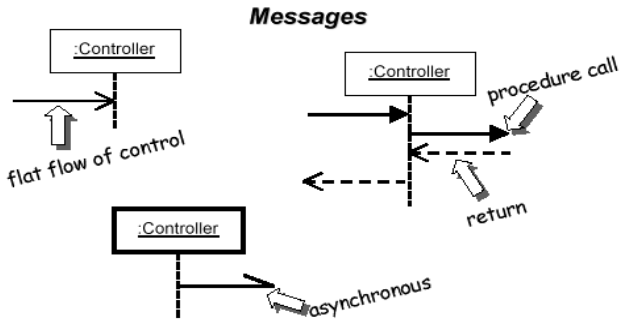
- message (method call) indicated by horizontal arrow to other object
  - write message name and arguments above arrow



23

Messages, continued

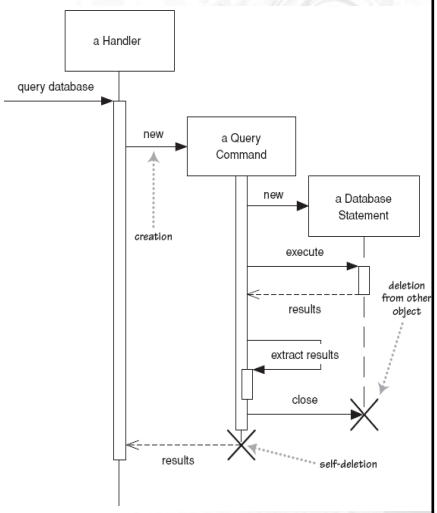
- message (method call) indicated by horizontal arrow to other object
  - dashed arrow back indicates return
  - different arrowheads for normal / concurrent (asynchronous) methods



24

Lifetime of objects

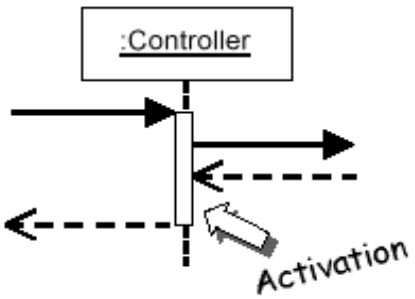
- *creation*: arrow with 'new' written above it
  - notice that an object created after the start of the scenario appears lower than the others
- *deletion*: an X at bottom of object's lifeline
  - Java doesn't explicitly delete objects; they fall out of scope and are garbage-collected

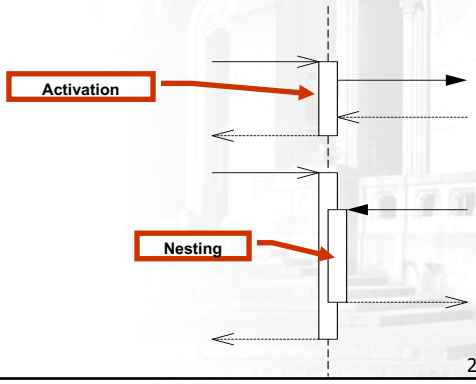


25

Indicating method calls

- **activation**: thick box over object's life line; drawn when object's method is on the stack
  - either that object is running its code, or it is on the stack waiting for another object's method to finish
  - nest to indicate recursion

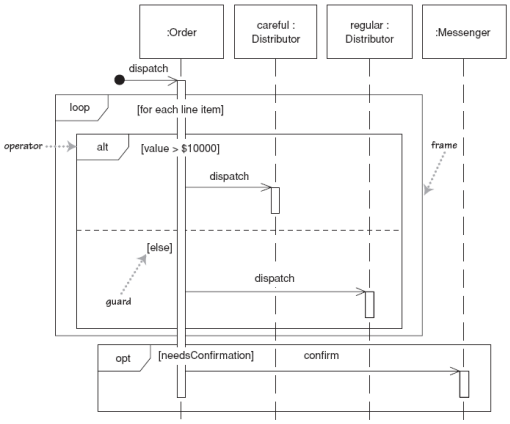




26

Indicating selection and loops

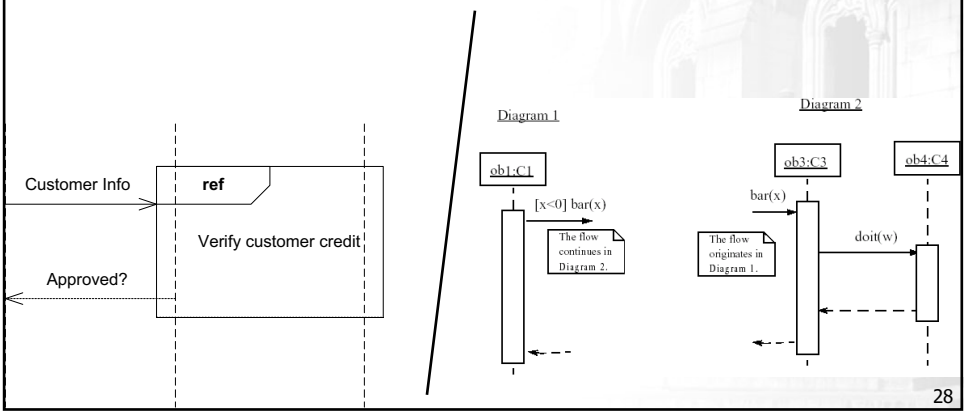
- frame: box around part of a sequence diagram to indicate selection or loop
  - `if` -> (opt) [condition]
  - `if/else` -> (alt) [condition], separated by horizontal dashed line
  - loop -> (loop) [condition or items to loop over]



27

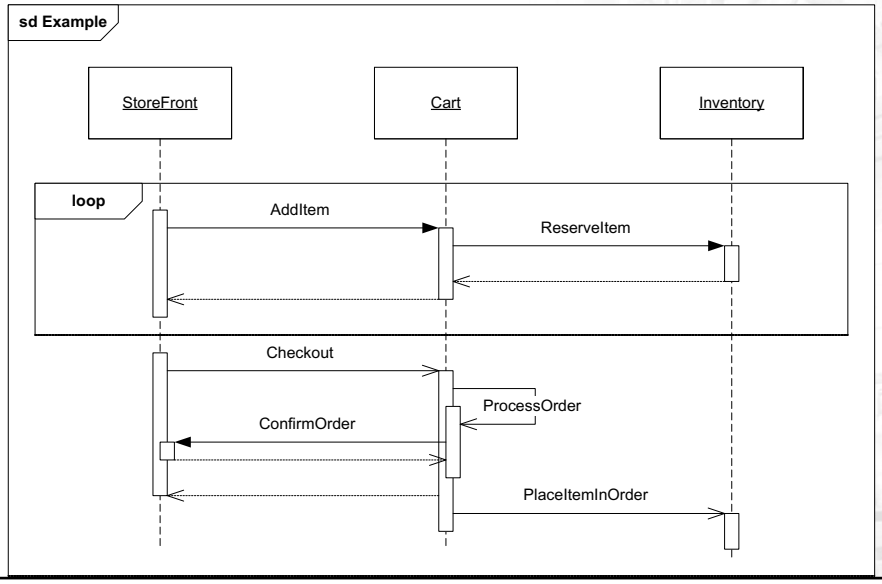
linking sequence diagrams

- if one sequence diagram is too large or refers to another diagram, indicate it with either:
  - an unfinished arrow and comment
  - a "ref" frame that names the other diagram
  - when would this occur in our system?



28

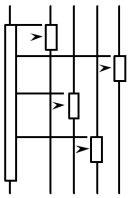
Example sequence diagram

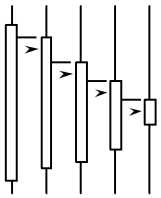


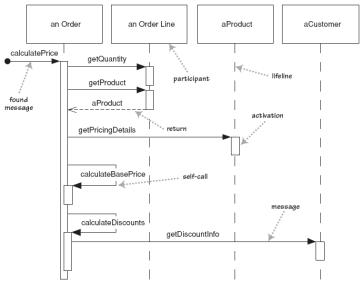
29

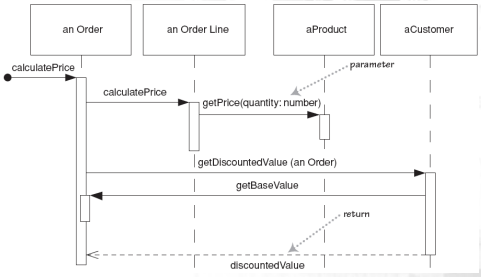
Forms of system control

- What can you say about the control flow of each of the following systems?
  - Is it centralized?
  - Is it distributed?





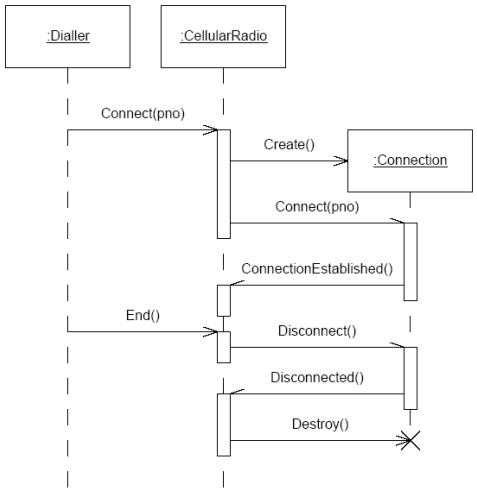




30

Flawed sequence diagram 1

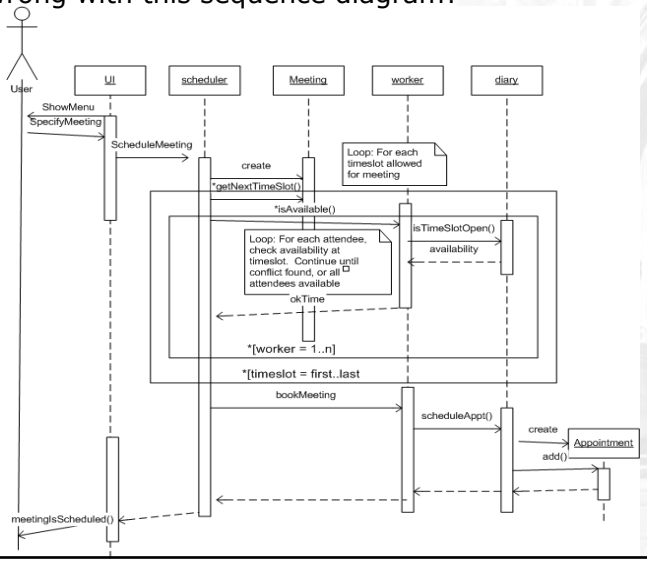
- What's wrong with this sequence diagram? (Look at the UML syntax and the viability of the scenario.)



31

Flawed sequence diagram 2

- What's wrong with this sequence diagram?



32

Why not just code it?

- Sequence diagrams can be somewhat close to the code level. So why not just code up that algorithm rather than drawing it as a sequence diagram?
  - a good sequence diagram is still a bit above the level of the real code (not all code is drawn on diagram)
  - sequence diagrams are language-agnostic (can be implemented in many different languages
  - non-coders can do sequence diagrams
  - easier to do sequence diagrams as a team
  - can see many objects/classes at a time on same page (visual bandwidth)

33

Sequence diagram exercise 1

- Let's do a sequence diagram for the following casual use case, *Start New Poker Round* :

  The scenario begins when the player chooses to start a new round in the UI. The UI asks whether any new players want to join the round; if so, the new players are added using the UI.

  All players' hands are emptied into the deck, which is then shuffled. The player left of the dealer supplies an ante bet of the proper amount. Next each player is dealt a hand of two cards from the deck in a round-robin fashion; one card to each player, then the second card.

  If the player left of the dealer doesn't have enough money to ante, he/she is removed from the game, and the next player supplies the ante. If that player also cannot afford the ante, this cycle continues until such a player is found or all players are removed.

34

Sequence diagram exercise 2

- Let's do a sequence diagram for the following casual use case, *Add Calendar Appointment* :

  The scenario begins when the user chooses to add a new appointment in the UI. The UI notices which part of the calendar is active and pops up an Add Appointment window for that date and time.

  The user enters the necessary information about the appointment's name, location, start and end times. The UI will prevent the user from entering an appointment that has invalid information, such as an empty name or negative duration. The calendar records the new appointment in the user's list of appointments. Any reminder selected by the user is added to the list of reminders.

  If the user already has an appointment at that time, the user is shown a warning message and asked to choose an available time or replace the previous appointment. If the user enters an appointment with the same name and duration as an existing group meeting, the calendar asks the user whether he/she intended to join that group meeting instead. If so, the user is added to that group meeting's list of participants.

35