CSE 403
Lecture 8

UML Sequence Diagrams

Reading:
UML Distilled, *Ch. 4,* M. Fowler
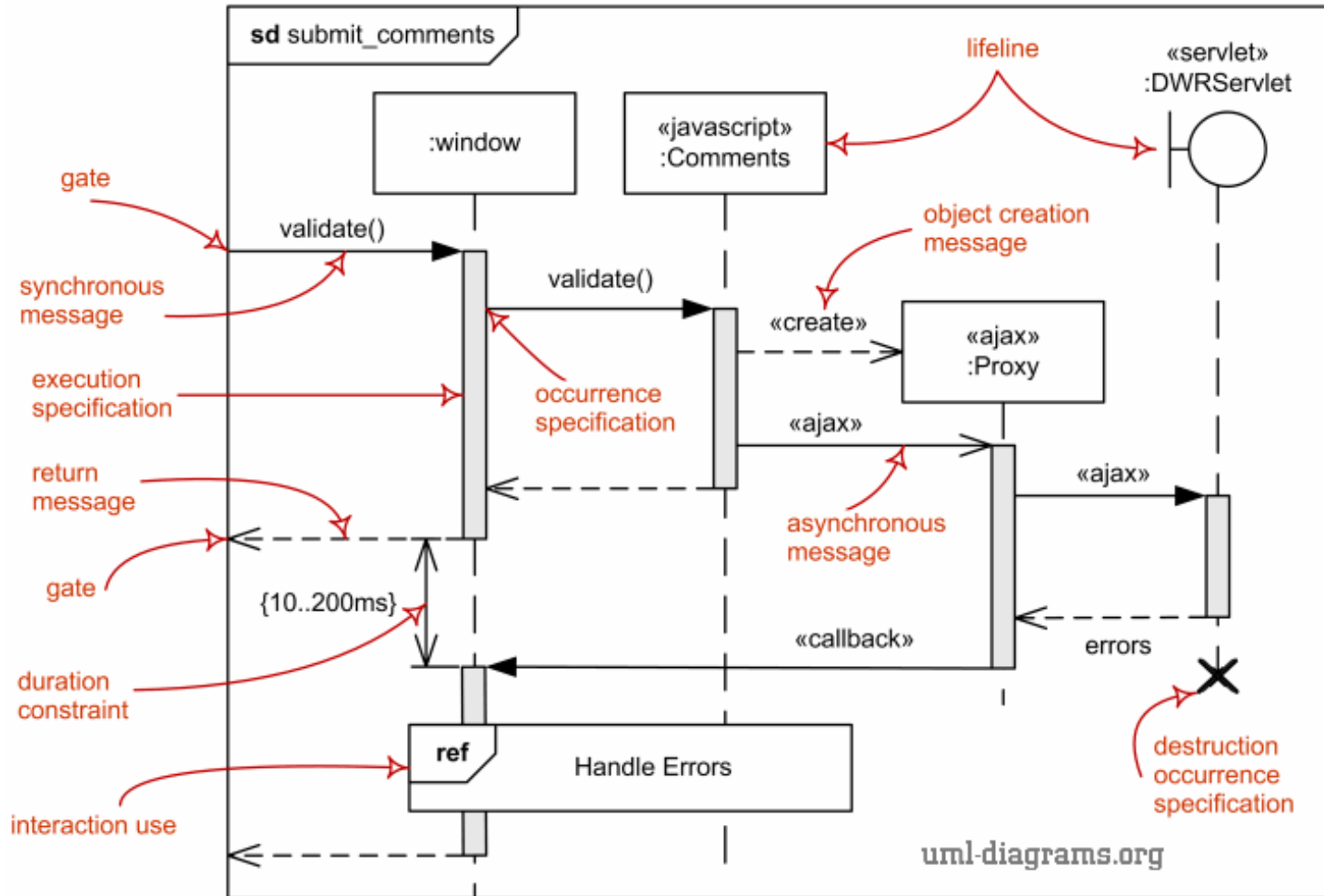
slides created by Marty Stepp
http://www.cs.washington.edu/403/

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 $\rightarrow$ class diagram
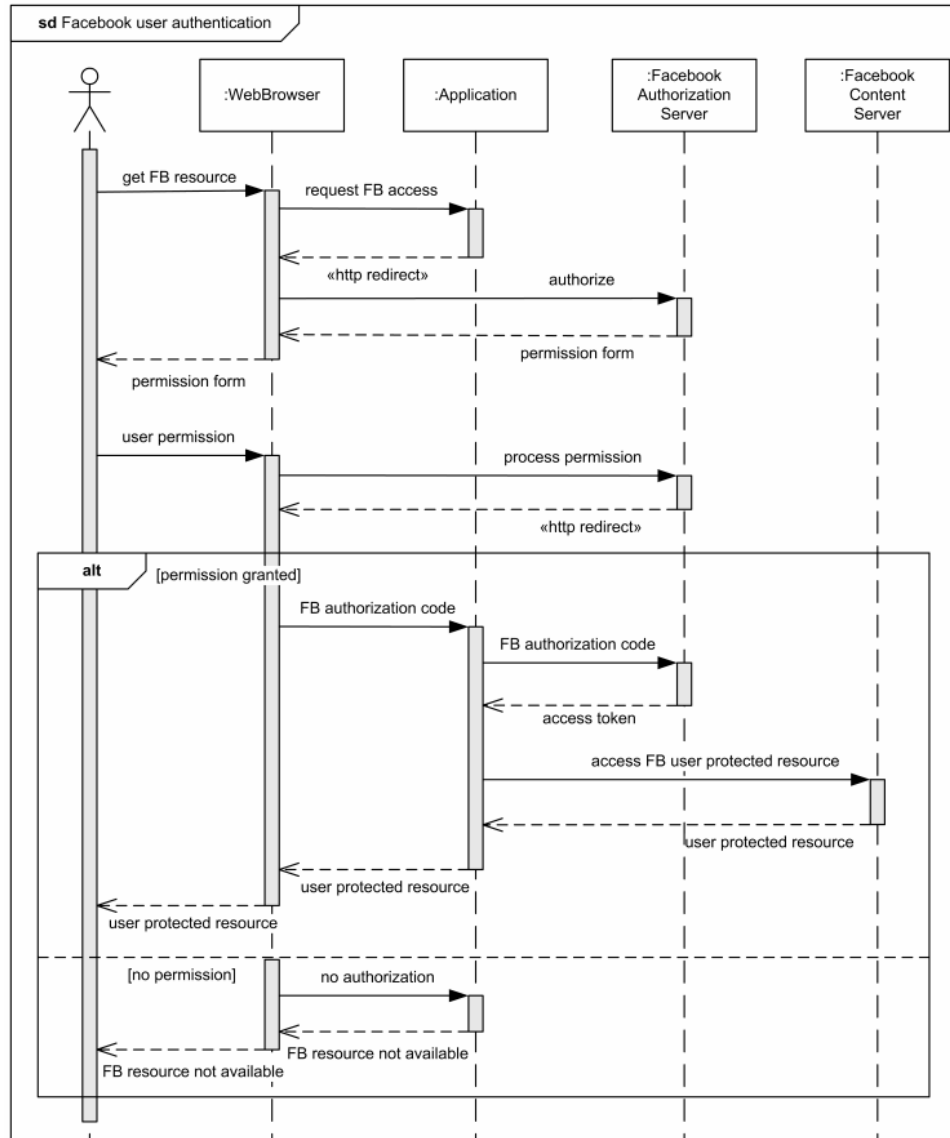 - use cases $\rightarrow$ sequence diagrams

Parts of a seq.diagram

Sequence diag. key parts

- **participant**: object or entity that acts in the diagram
  - 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)

Seq. diag. examples

sd Facebook user authentication

| :WebBrowser | :Application | :Facebook Authorization Server | :Facebook Content Server |

- get FB resource
- request FB access
- «http redirect»
- authorize
- permission form
- permission form
- user permission
- process permission
- «http redirect»

alt [permission granted]
- FB authorization code
- FB authorization code
- access token
- access FB user protected resource
- user protected resource
- user protected resource
- user protected resource

[no permission]
- no authorization
- FB resource not available
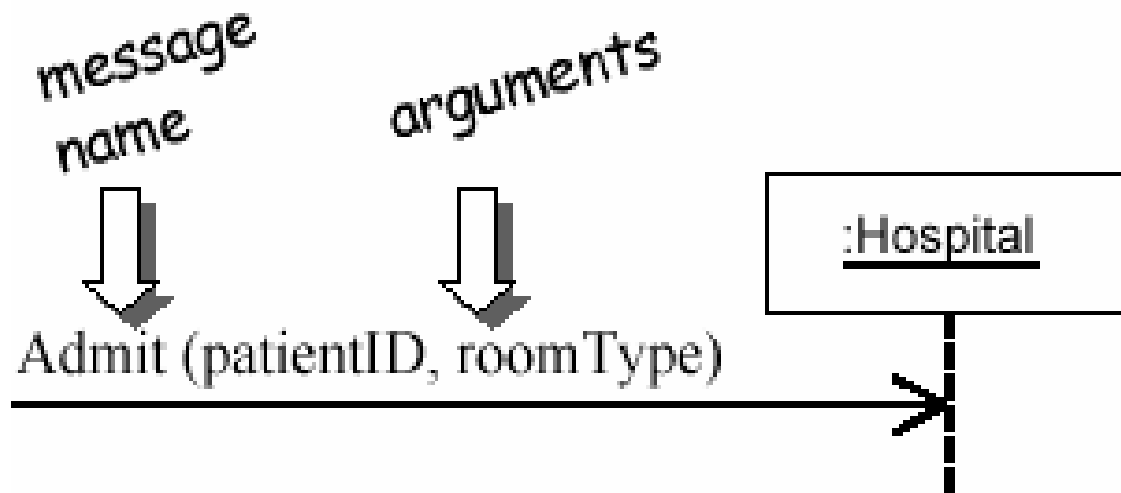- FB resource not available

5

Representing objects

- Squares with object type, optionally preceded by "*name* :"
  - write object's name if it clarifies the diagram
  - object's "life line" represented by dashed vert. line
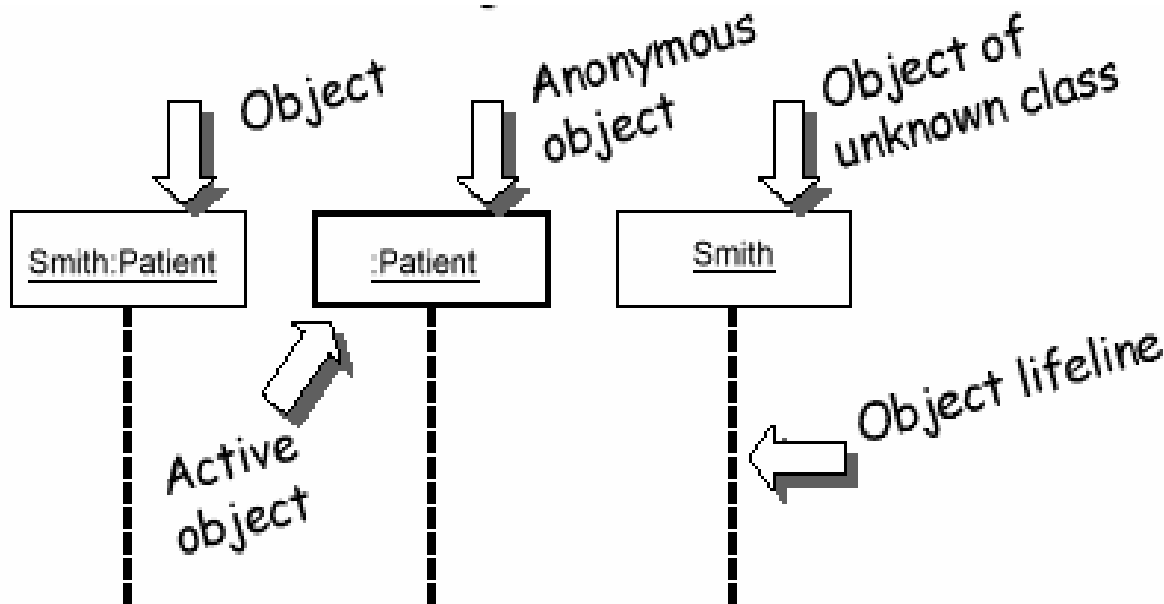
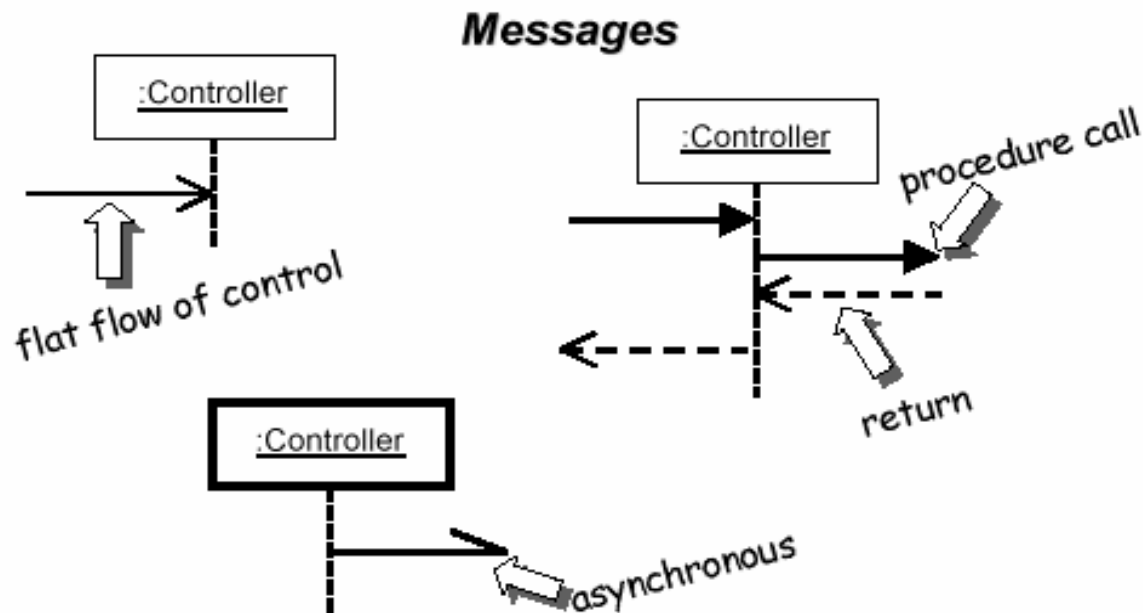**Name syntax:** <u><objectname>:<classname></u>

Messages between objects

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

Messages, continued

- messages (method calls) indicated by arrow to other object
  - dashed arrow back indicates return
 - different arrowheads for normal / concurrent (asynchronous) calls

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
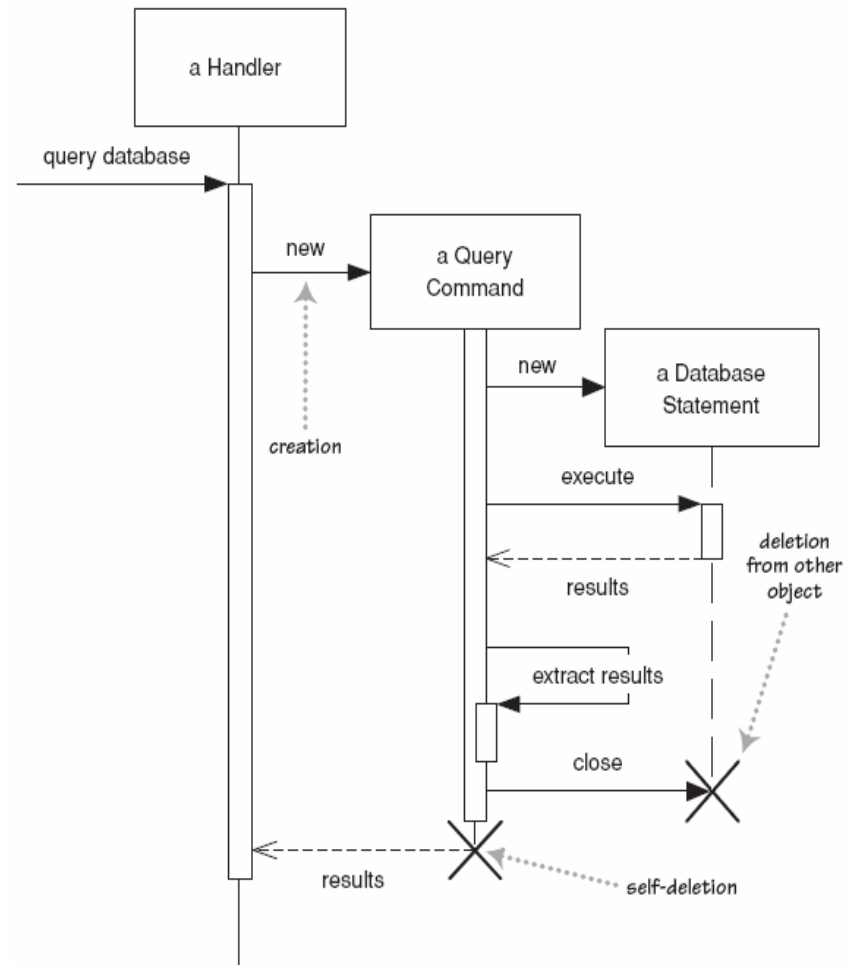
Indicating method calls

- **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
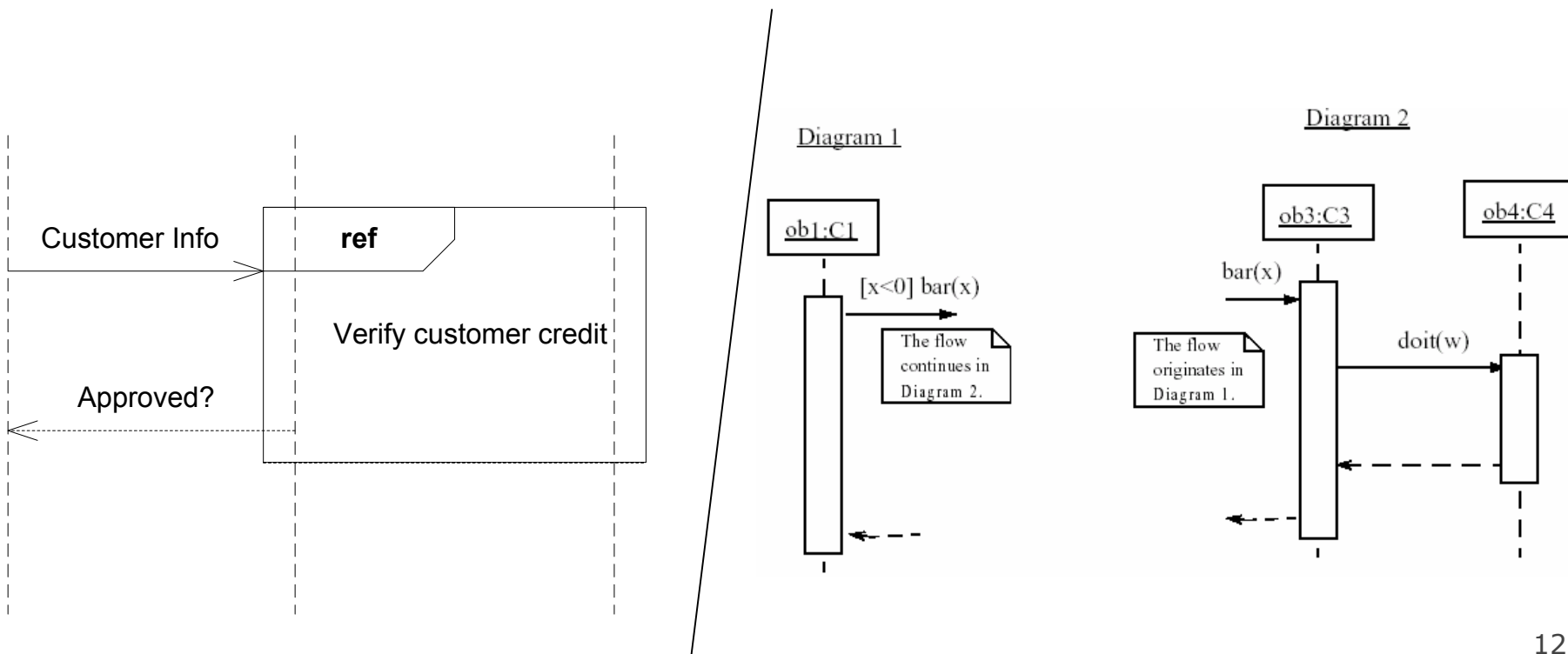 - nest activations to indicate recursion
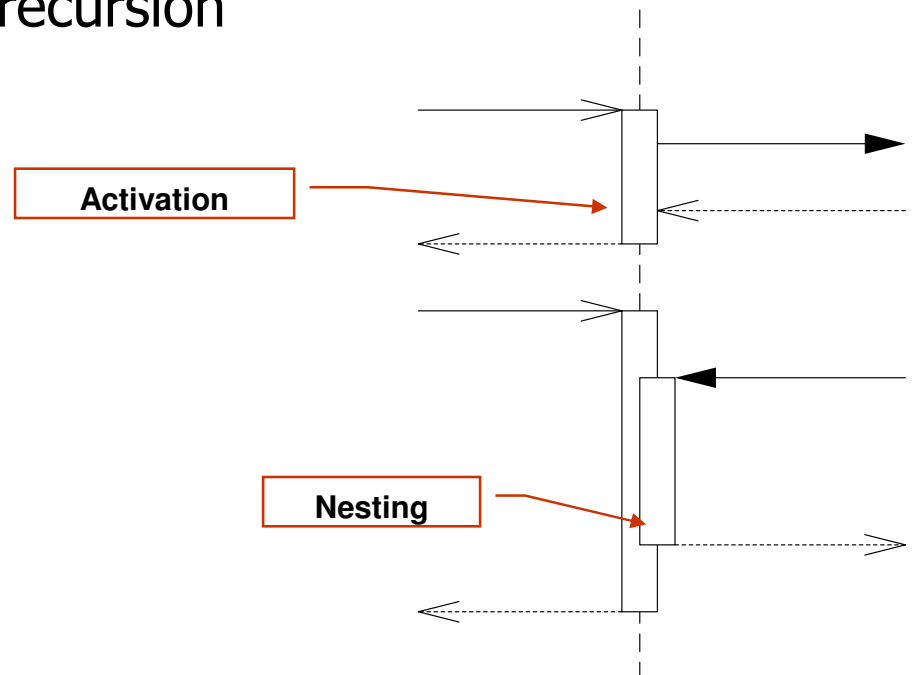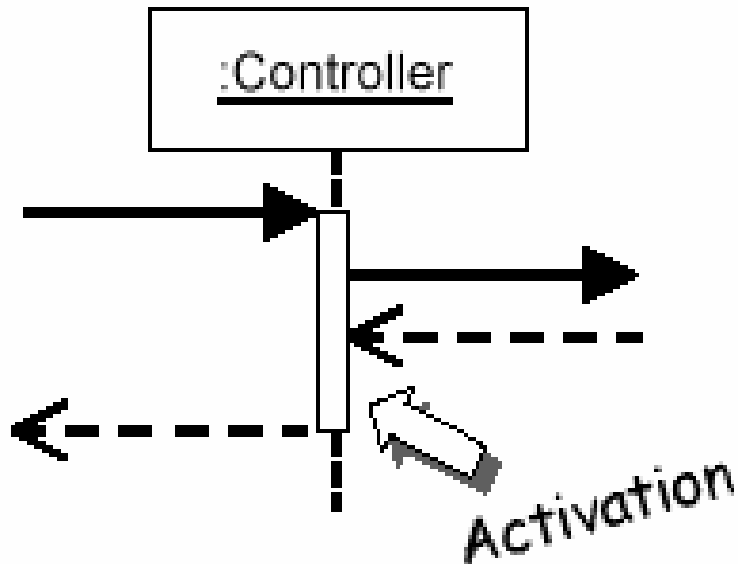
Selection and loops

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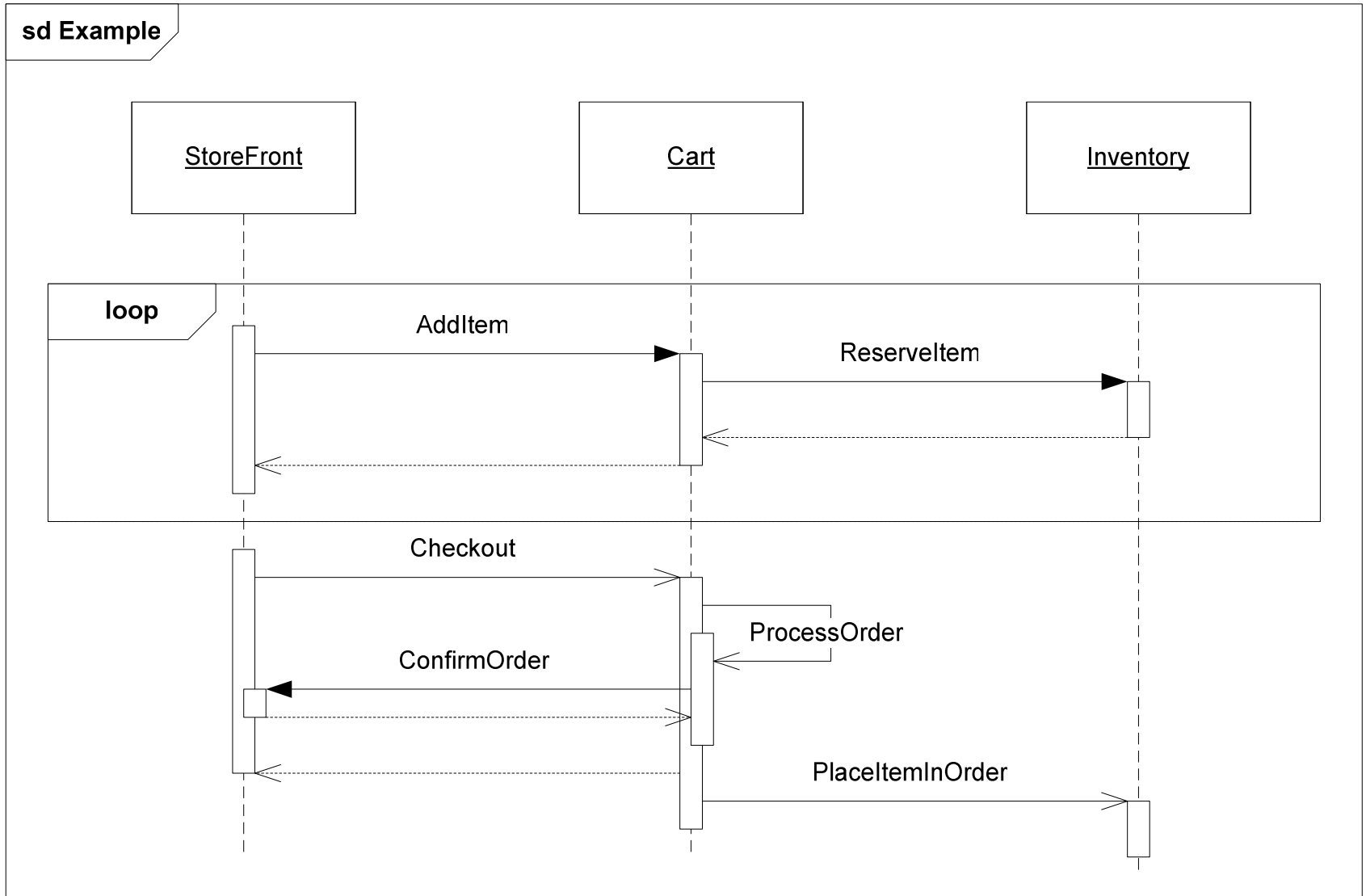 - loop -> (loop) [condition or items to loop over]

11
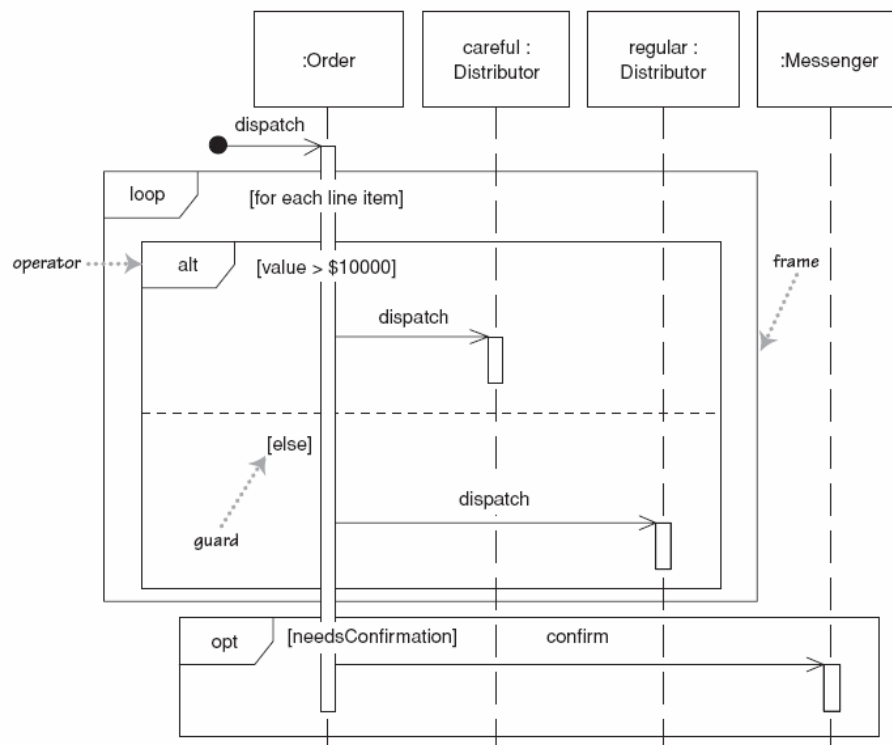
Linking sequence diagrams

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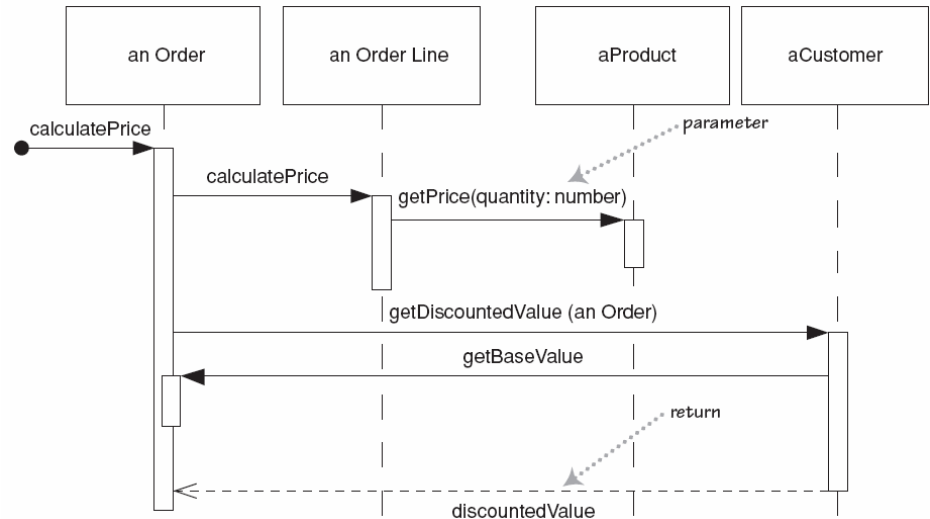 - or a "ref" frame that names the other diagram
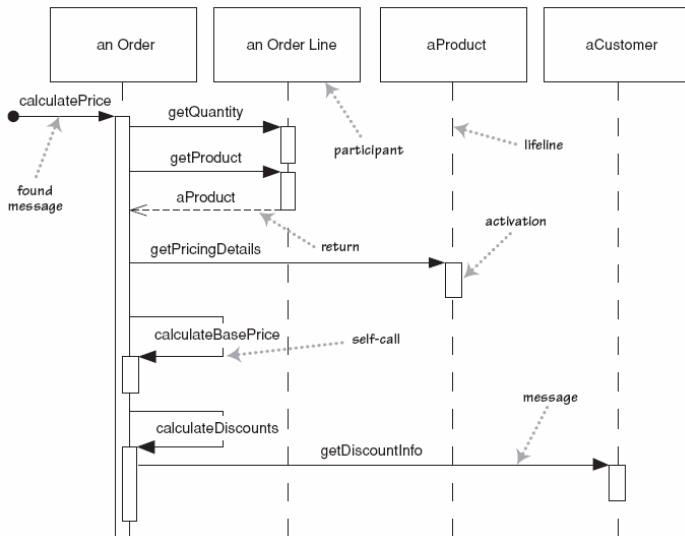   - when would this occur in our system?

Example sequence diagram

sd Example

StoreFront   Cart   Inventory

loop

AddItem

ReserveItem

Checkout

ProcessOrder

ConfirmOrder

PlaceItemInOrder
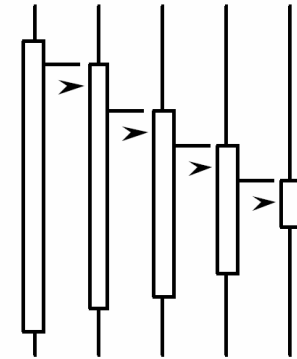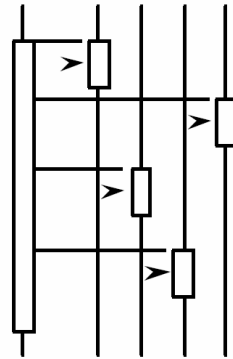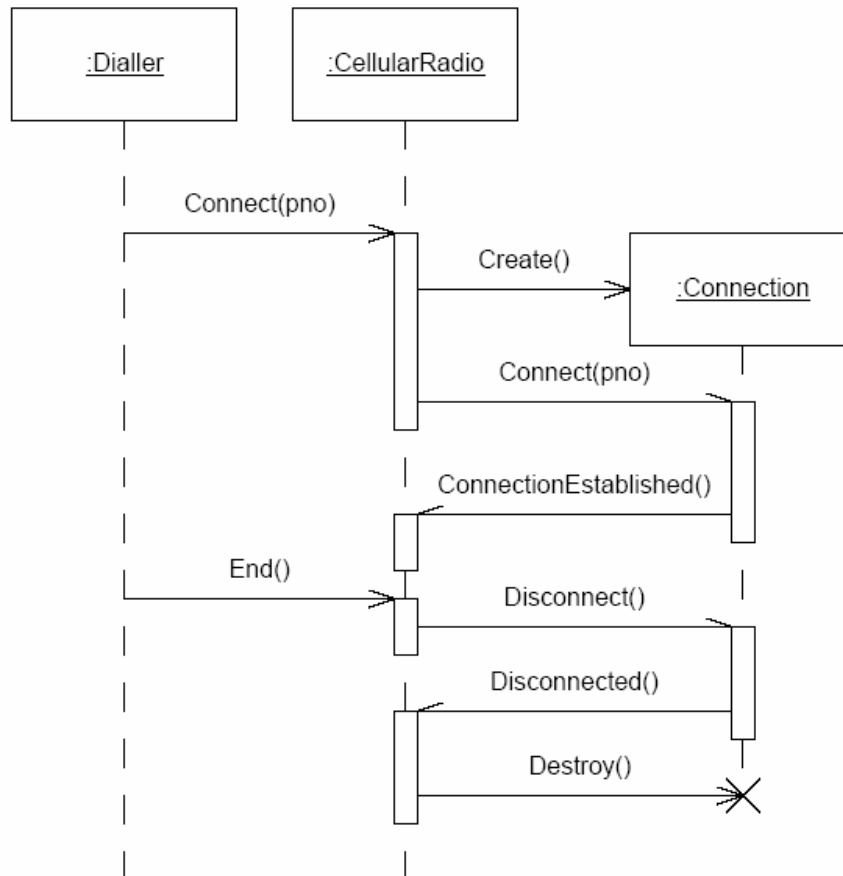
13

Forms of system control

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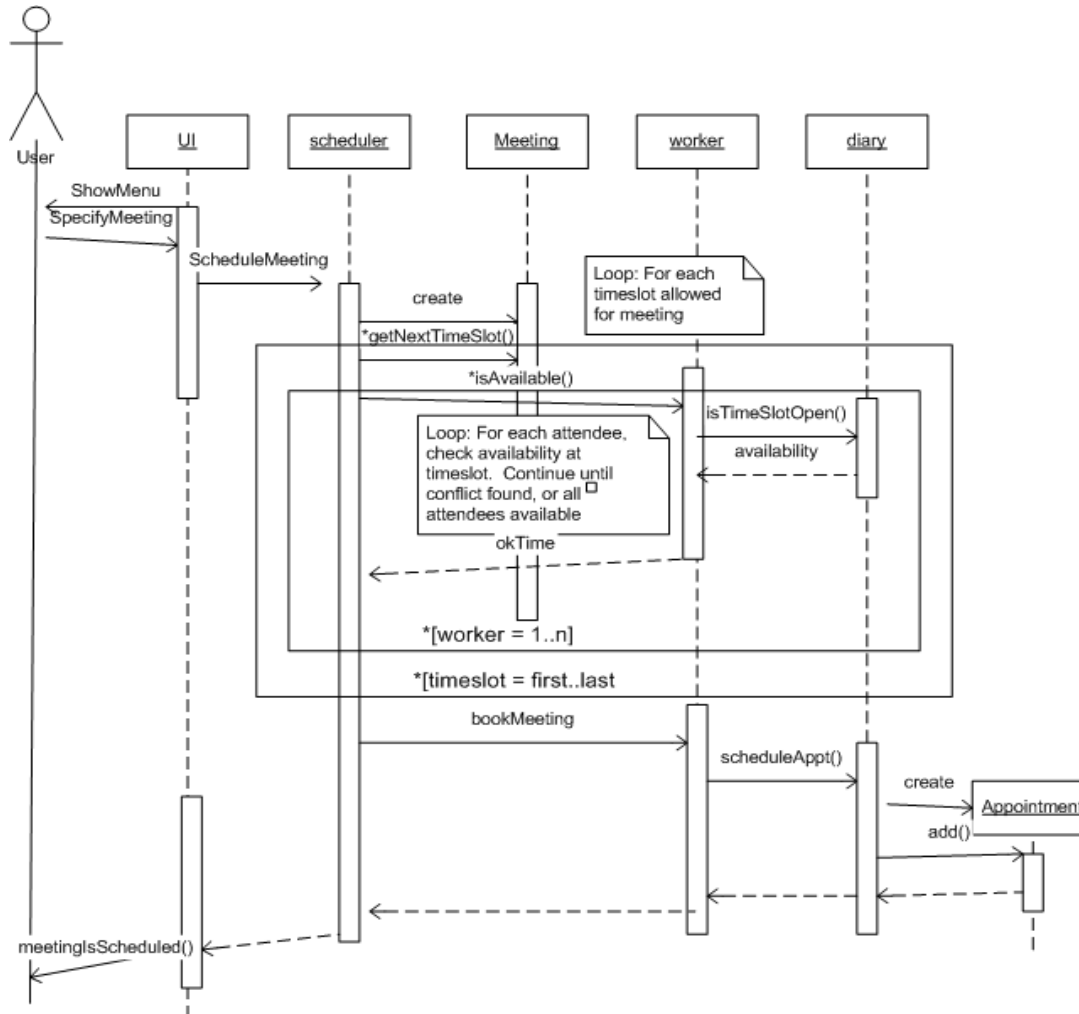 - Is it centralized?
  - Is it distributed?

Flawed sequence diagram 1

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

Flawed sequence diagram 2

- What's wrong with this sequence diagram?

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)
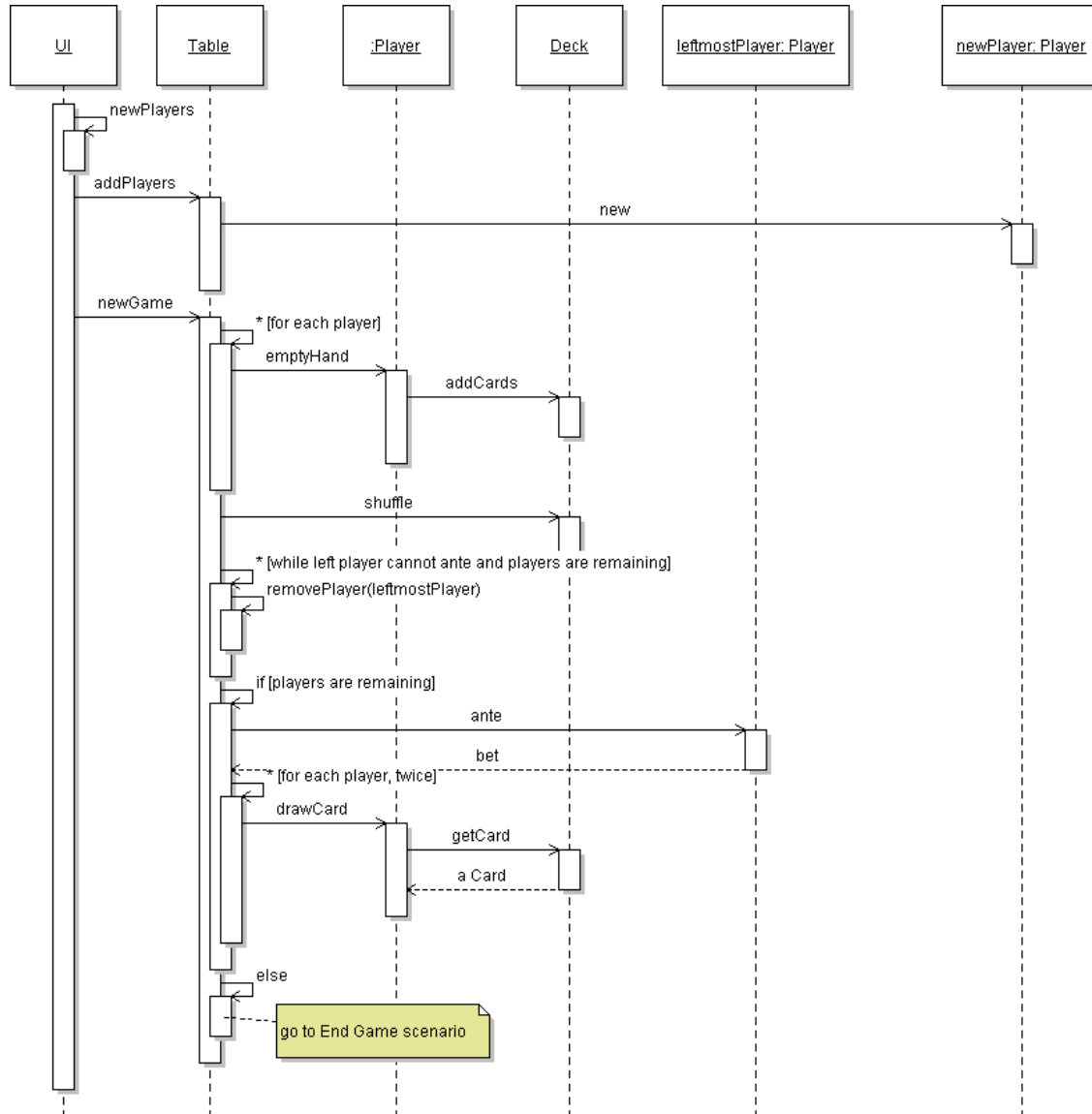
Sequence diagram exercise

- Let's do a sequence diagram for the following poker casual use case, *Start New Game 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.

Poker sequence diagram

Sequence diagram question

- Consider the possible poker use case, *Betting Round* :

> The scenario begins after the *Start New Round* case has completed. The UI asks the first player for a bet. That player chooses to either bet a given amount, or check (no bet).
>
> The next player is asked what to do. If the prior player placed a bet, the next player must either match ("see") it, or match it plus add an additional bet ("raise"), or choose not to match and exit the round ("fold"). This continues around the table until an entire pass is made in which all players have either matched all other players' bets or folded.
>
> If the next player doesn't have enough money to match the current bet, the player is allowed to bet all of their money. But they can then win only up to the amount they bet; the rest is a "side pot" among the more wealthy players remaining in the round.

- Why is it hard to diagram this case as a sequence diagram?

Poker sequence diagram 2
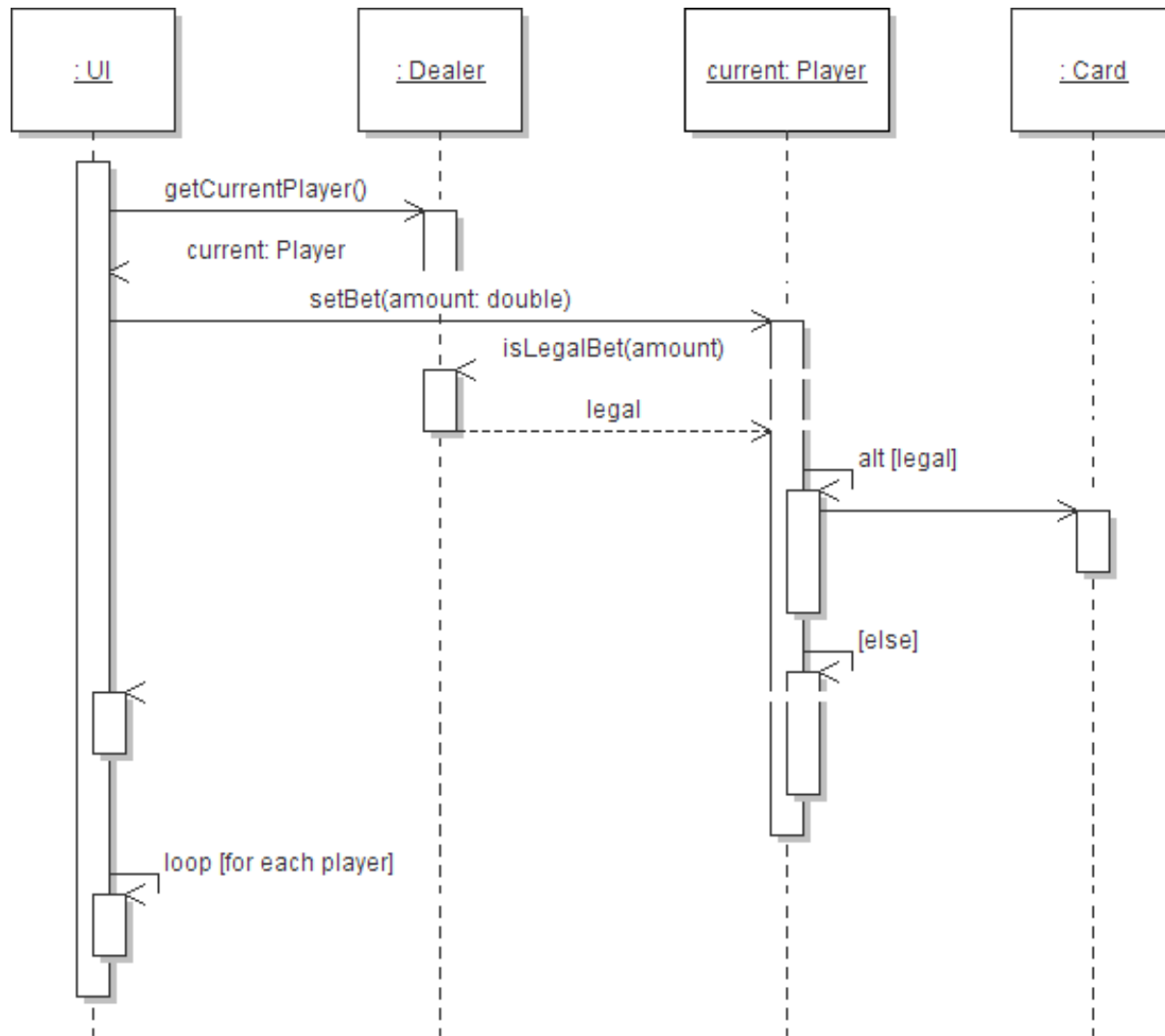
Sequence diagram exercise

- Let's do a sequence diagram for the following casual use case for a scheduler app, *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 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 message and asked to choose an available time or replace the appointment. If the user enters an appointment with the same name and duration as an existing meeting, the calendar asks the user whether he/she intended to join that meeting instead. If so, the user is added to that meeting's list of participants.