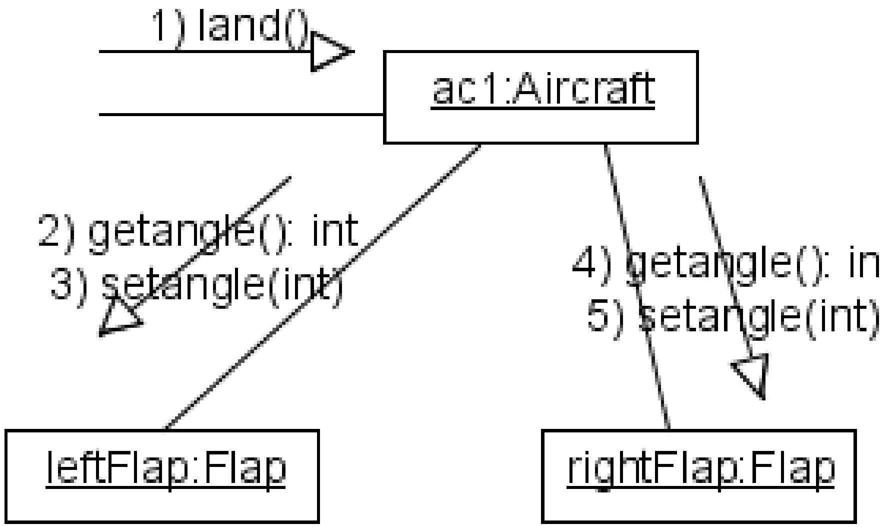
Object-Interaction Diagrams:
Sequence Diagrams

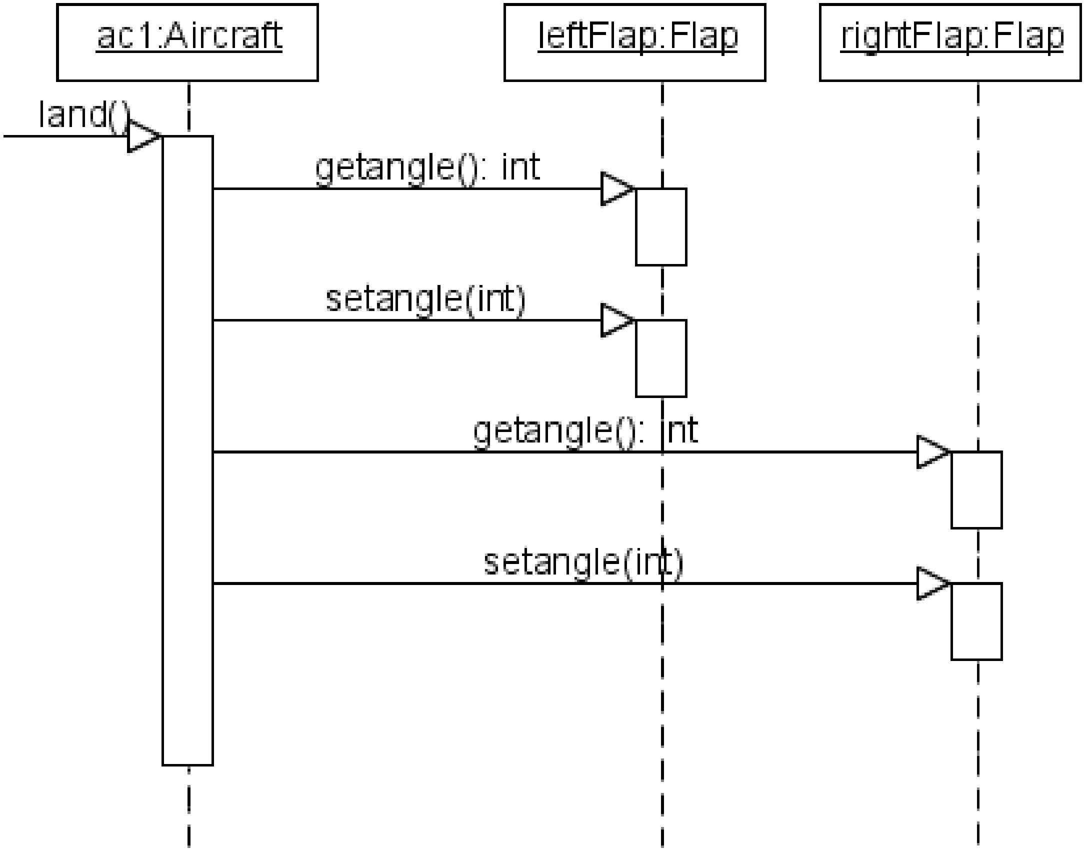
Communication and Time

- In communication diagrams, ordering of messages is achieved by labelling them with sequence numbers
- Does not make temporal oredering very explicit.

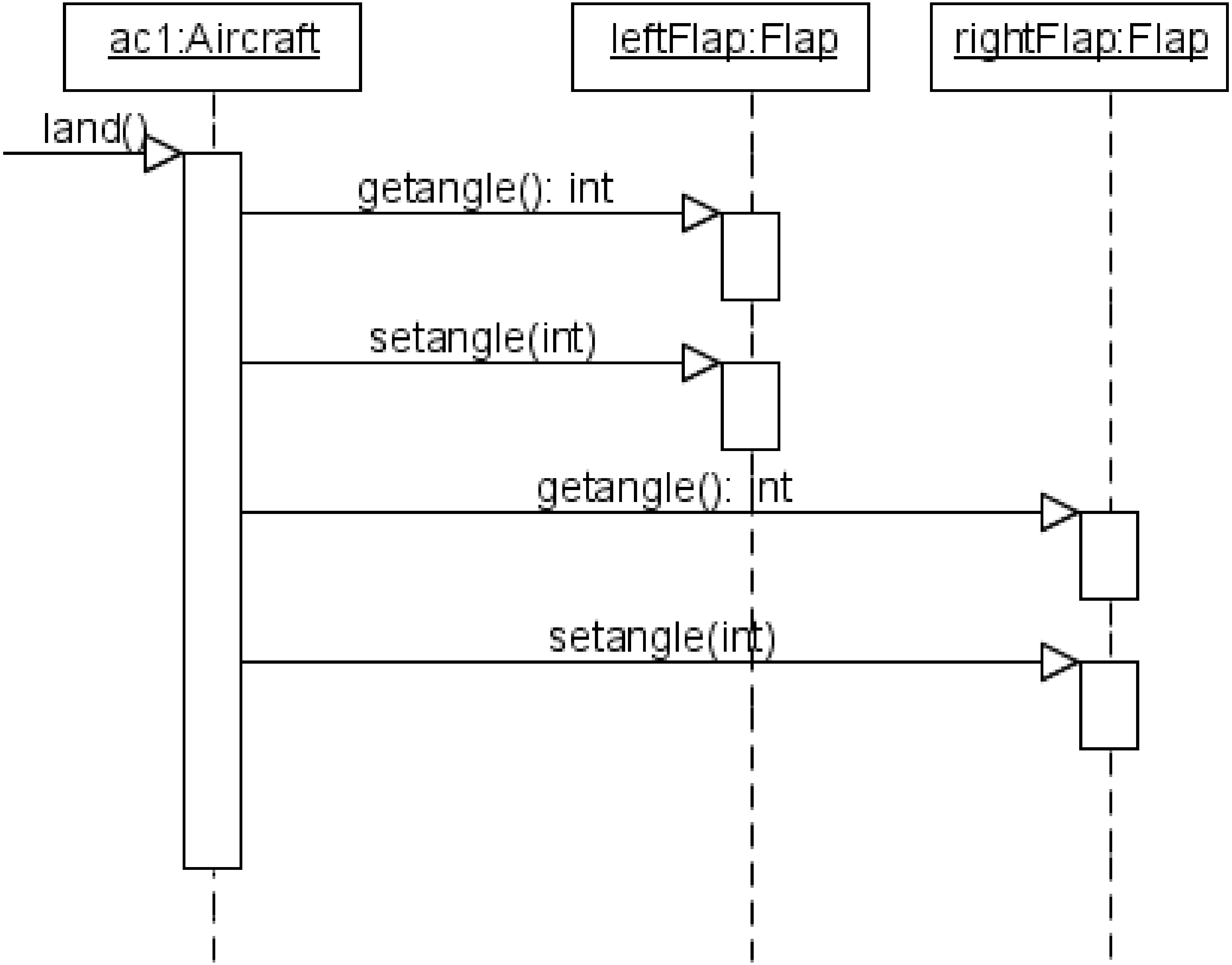


Sequence Diagrams

- Sequence diagrams make temporal order explicit.
- However, they do not contain link/association information.



A closer look



Components of Sequence Diagrams

- Vertical time axis, time increasing downwards.
- Objects that exchange messages in the current execution are shown on the horizontal axis, at the top.
- With every object is a vertical dashed line, which depicts an object's lifeline.
- Over the object active lifetime, the lifeline is a rectangle, which depicts when an object is active (i.e., has control).
  - The rectangle's size is proportional to how much time the execution takes.
- Arrows depict messages from a sender object to a target object and the message is written along the arrow.

In this diagram ...

- In the example above, we assume that ac1 has a leftFlap and a rightFlap.
- Note that when we send the getAngle() message, we don't have an arrow that shows the return value.
- The message is synchronous, so return is implicit and it is not shown on the diagram.

Problem

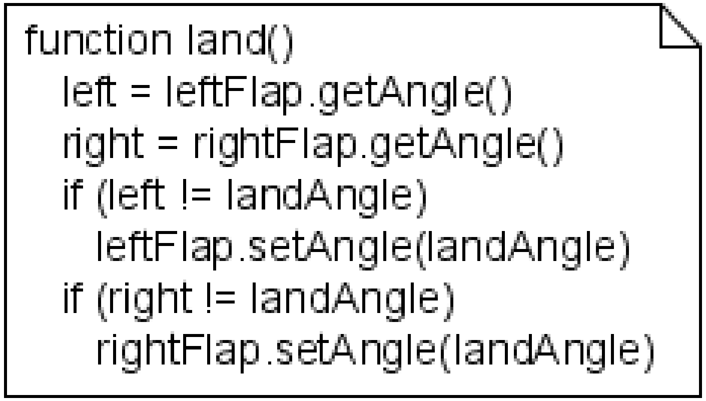
- Suppose the code for land() was the following...

```
function land()
    left = leftFlap.getAngle()
    right = rightFlap.getAngle()
    if (left != landAngle)
      leftFlap.setAngle(landAngle)
    if (right != landAngle)
      rightFlap.setAngle(landAngle)
```

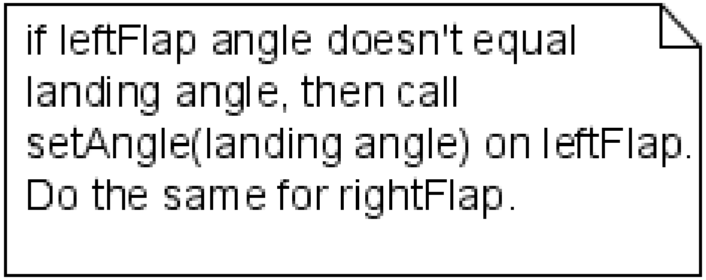
- How can we show that setAngle(int) will be conditionally called ?

Solution in UML 1.4

- How can we show that setAngle(int) will be conditionally called ?
- We Can't !
- The solution is to add annotations to the diagram, on the far left.
- The note will let us know of this conditional message.
- The note can be pseudocode or just a plain sentence.



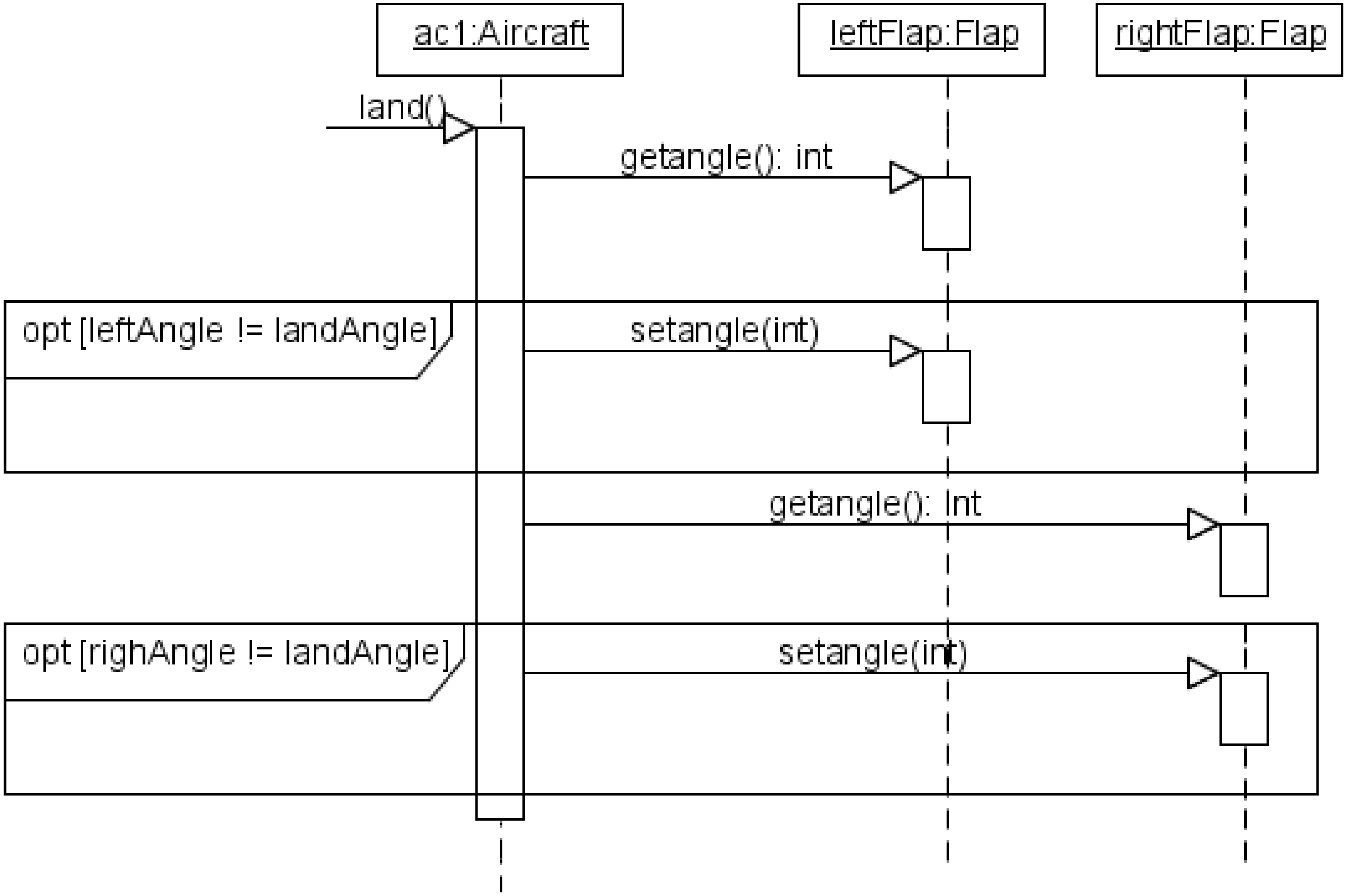
```
function land()
   left = leftFlap.getAngle()
   right = rightFlap.getAngle()
   if (left != landAngle)
      leftFlap.setAngle(landAngle)
   if (right != landAngle)
      rightFlap.setAngle(landAngle)
```



if leftFlap angle doesn't equal landing angle, then call setAngle(landing angle) on leftFlap. Do the same for rightFlap.

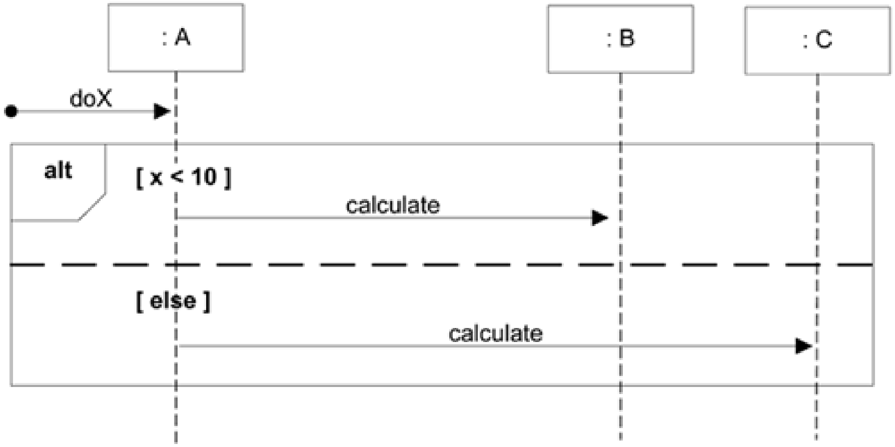
Solution in UML 2.0

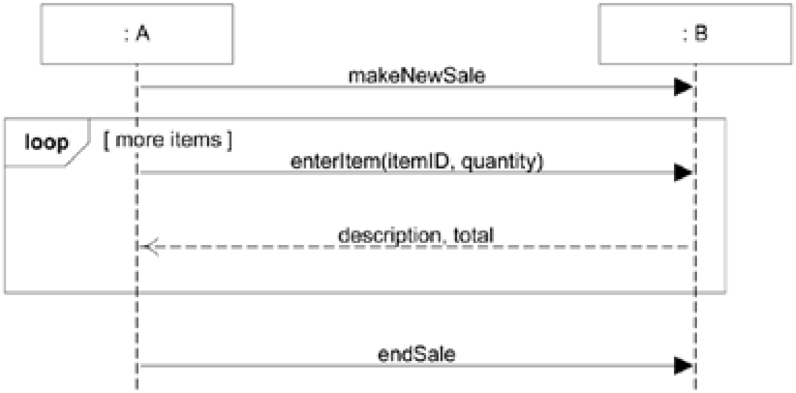
- Use an OPT frame.



All types of Frames

- Alt: Alternative fragment for mutual exclusion conditional logic expressed in the guards.
- Loop: Loop fragment while guard is true.
- Opt: Optional fragment that executes if guard is true.
- Par: Parallel fragments that execute in parallel.
- Region: Critical region within which only one thread can run.





Synchronous vs. Asynchronous

- If you order a piece of equipment, and the salesman goes in the back store, do you wait for the piece of equipment?
- If you order a piece of equipment, and the salesman tells you it is backordered, do you wait for the piece of equipment?

Synchronous Messages

- The sender object waits until target object finishes its execution of the message.
- Target object processes only one message at a time.
- Consequently, this behavior represents a single thread of control.
  - only one object is active at any time

Asynchronous Messages

- Sender object does not wait until target object finishes its processing of the message (execution of the called method).
- Target object may accept many messages at a time.
- Consequently, this behavior requires multi-threaded execution.
  - many objects can be active at any time
  - this is also known as concurrency

Depicting Asynchronous Messages

- Instead of using a regular arrow, we use a stick arrowhead (in both collaboration and sequence).
  - In collaboration diagrams, nothing really changes!
- In sequence diagrams
  - we may have two objects executing at the same time.
  - sender object continues executing after sending message, target object starts executing as well.
- Of the target object can accept multiple messages, how does it handle them?

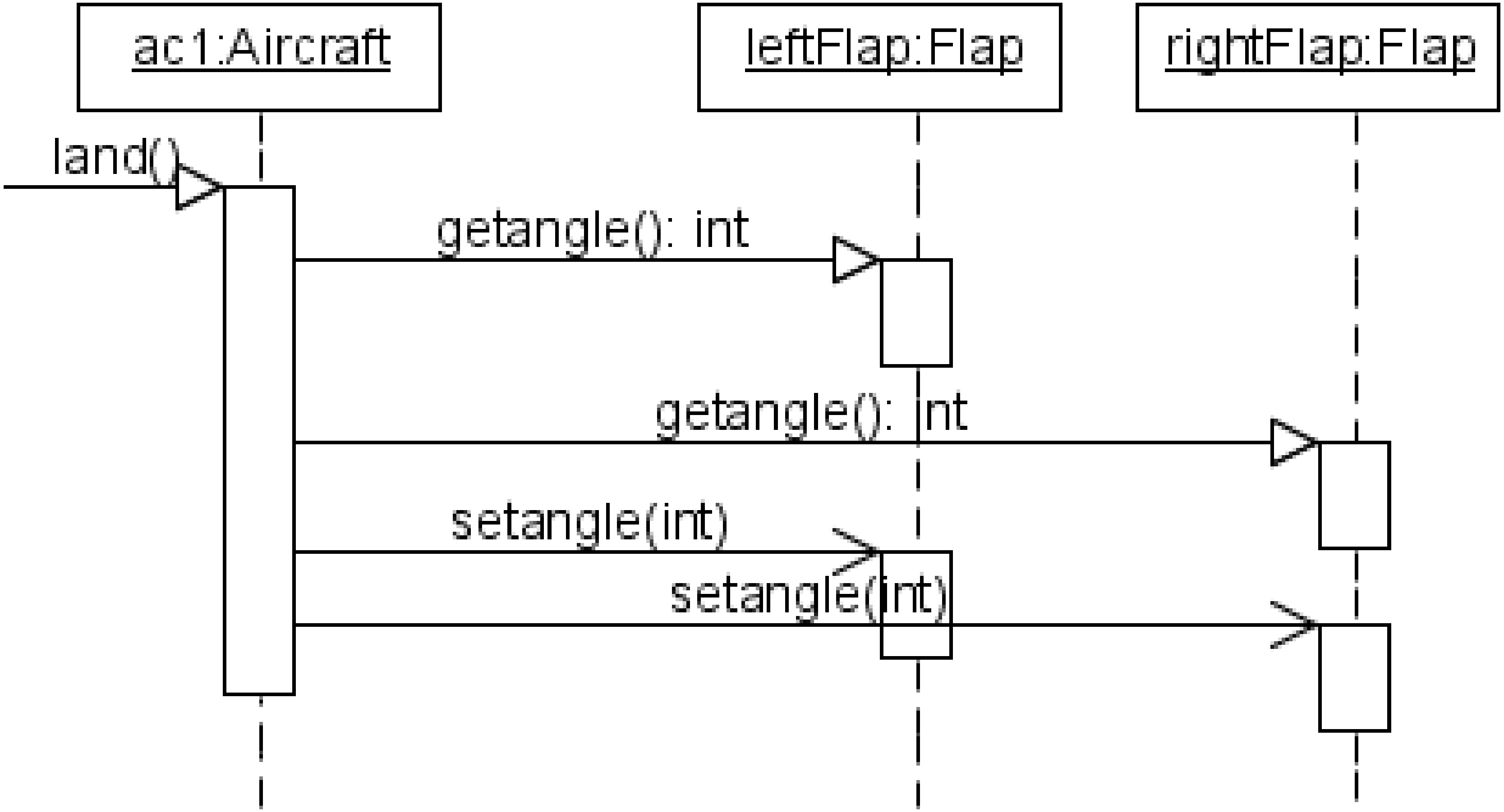
Concurrency

- If target <u>object's method</u> implements threading,
  - It can thread itself to handle messages.
  - This is called operation level concurrency.
- If target <u>object</u> itself implements threading,
  - It can thread itself to handle messages.
  - This is called object level concurrency.
- If objects don't implement any threading but the system is concurrent, objects must implement some way of handling messages. (system level concurrency)
  - Refuse message(s) if busy
  - Interrupt current executing message and start on new message
  - Queue message(s) for later processing (can be priority queue)

Message Priorities

- One way to deal with asynchronous messages is to queue them.
- That way, only one of them is processed at a time.
- But what happens if a message is more important than others.
- You can use priority levels to determine the order messages are processed.
- What are the dangers of this?

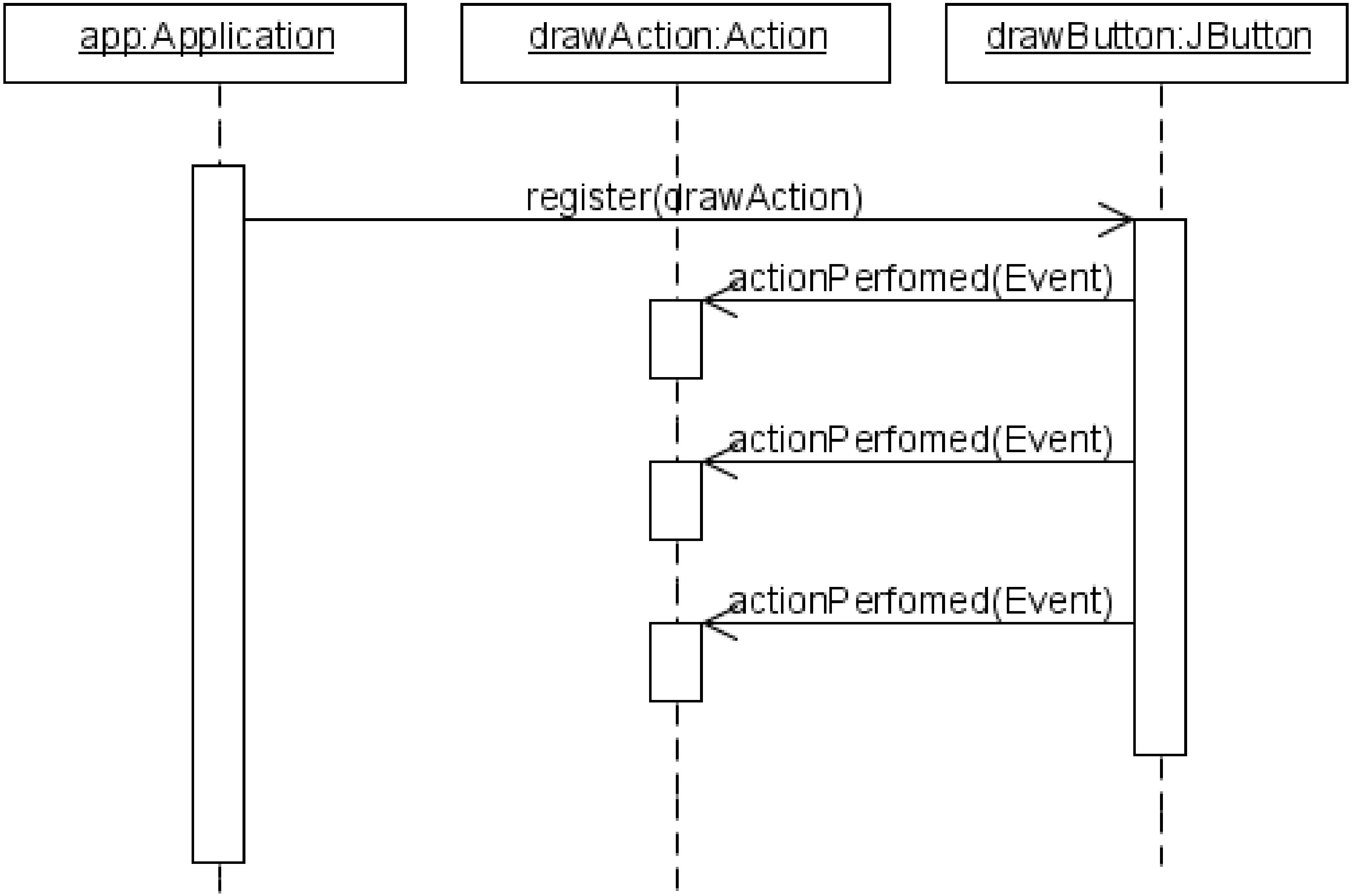
Async Flaps


ac1:Aircraft
leftFlap:Flap
rightFlap:Flap
land()
getangle(): int
getangle(): int
setangle(int)
setangle(int)

Callback Mechanism

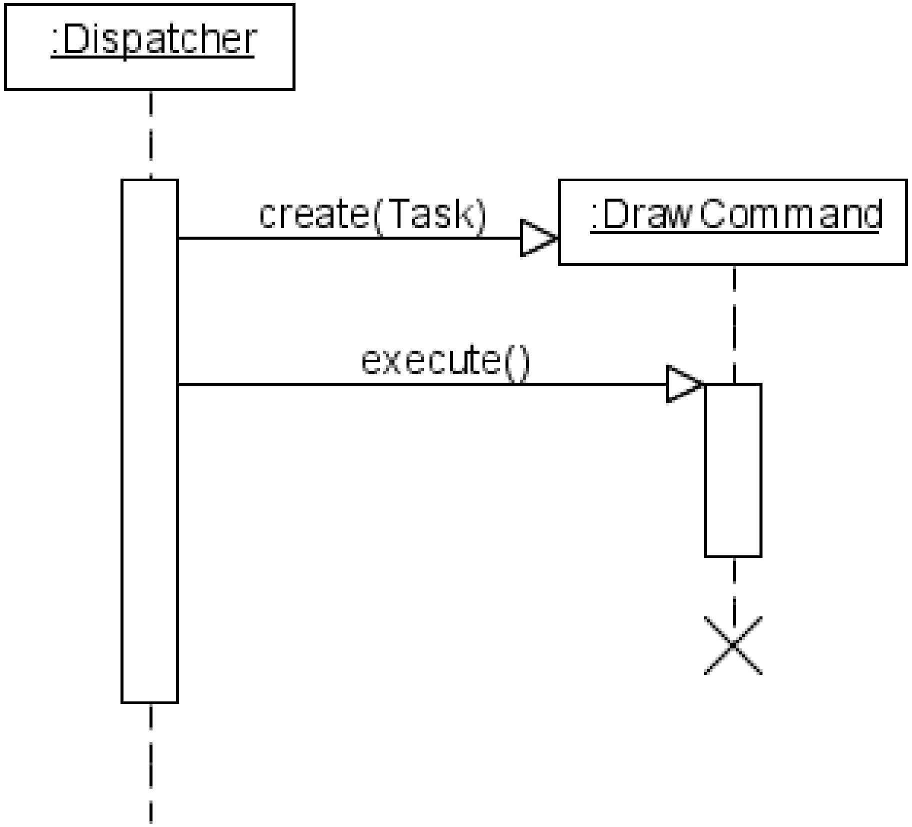
- Uses asynchronous messages.
- A subscriber object o1 is interested in an event e that occurs in o2.
- o1 registers interest in e by sending a message (that contains a reference to itself) to o2 and continues its execution.
- When e occurs, o2 will callback asynchronously to o1 (and any other subscribers).

Callback Illustrated



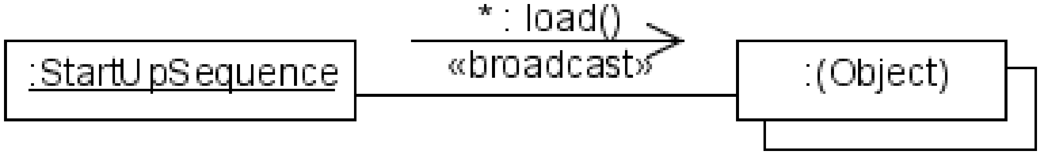
Object End-of-Life

- Sequence diagrams use an X to symbolize the end-of-life of an object.
- In garbage-collected languages, nothing needs to be done.
- However, in other languages, such as C++, the memory must be freed.

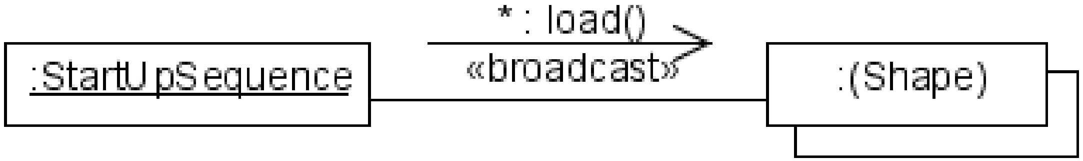


Broadcast

- Similar to iterative messaging, broadcast allows you to send a message to multiple objects.
- However, contrary to iterative messaging, no references are required.
- A broadcast is send to all the objects in the system.



- If only a specific category of object is targeted, we call this a narr


:StartUpSequence
* : load()
«broadcast»
:(Shape)