Last class, we took a look at Collaboration Diagrams.

These diagrams are also known as Communication Diagrams in UML 2.0.

As already mentioned, ordering of messages is achieved by illustrating them.

This is not very visual.
Sequence diagrams have more of a temporal focus.

However, they contain no association information.
A closer look
Components of Seq Diags

- 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 object lifetime.
- Over the object lifetime line is a rectangle, which depicts when an object is active (i.e. executing).
  - 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 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.

Given the message is synchronous, this is implicit and it is not shown on the diagram.
Suppose the code for `land()` was the following...

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

- We Can't!

- The solution is to add little notes 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.

```plaintext
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.
```
■ 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.
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 backorder, 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 threaded processor.
  - only one object is active at any time
Asynchronous Messages

- Sender object doesn't wait until target object finishes its execution of the message.
- Target object may accept many messages at a time.
- Consequently, this behavior requires a multi-threaded processor.
  - many objects can be active at any time
  - this is also known as concurrence

Fire and forget
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?
If target object's method implements threading,
  • It can thread itself to handle messages.
  • This is called operation level concurrency.

If target object 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
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

app: Application

drawAction: Action

drawButton: JButton

register(drawAction)

actionPerfomed(Event)

actionPerfomed(Event)

actionPerfomed(Event)
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 narrow cast.