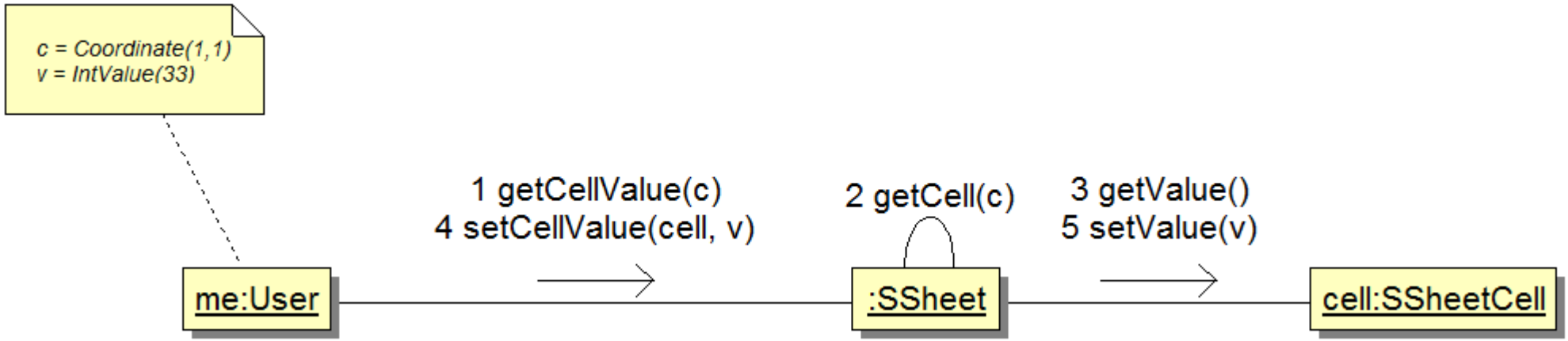
Object-Interaction Diagrams: Sequence Diagrams

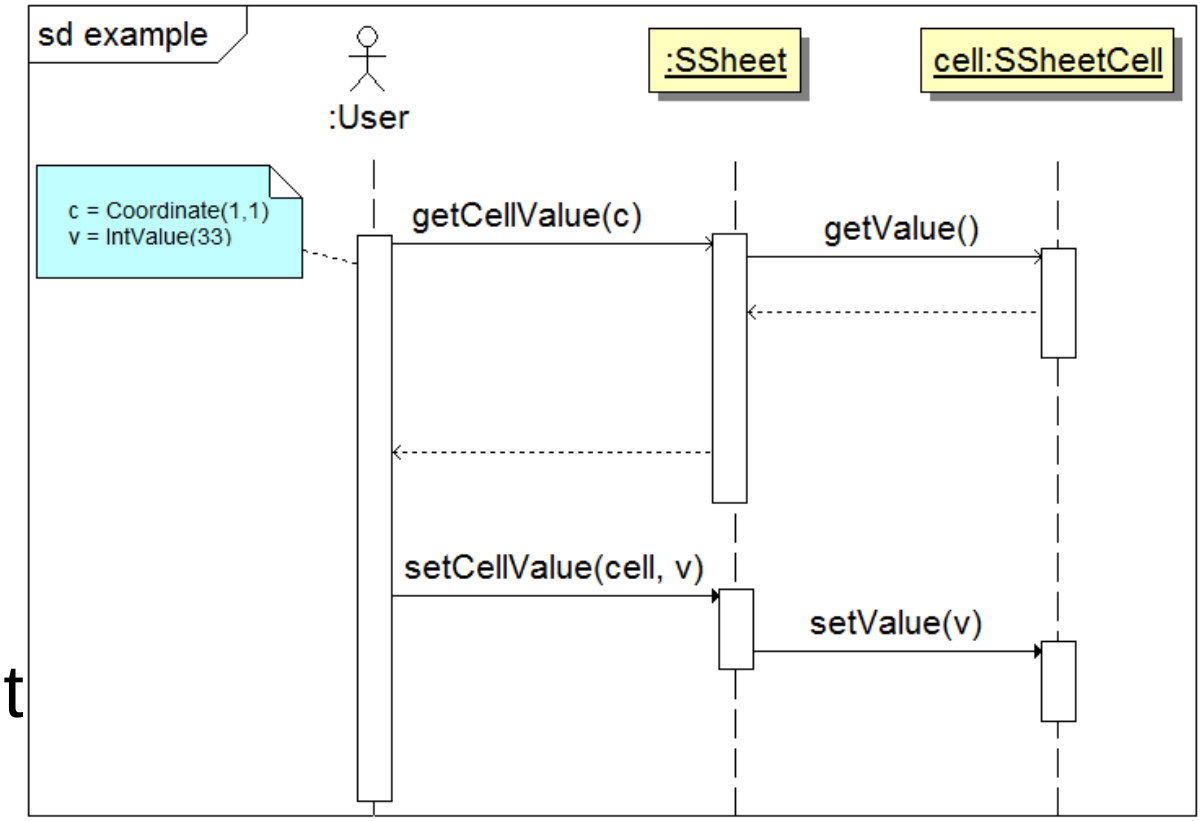
Communication and Time

- In communication diagrams, ordering of messages is achieved by labelling them with sequence numbers
- This does not make temporal ordering easy to follow.

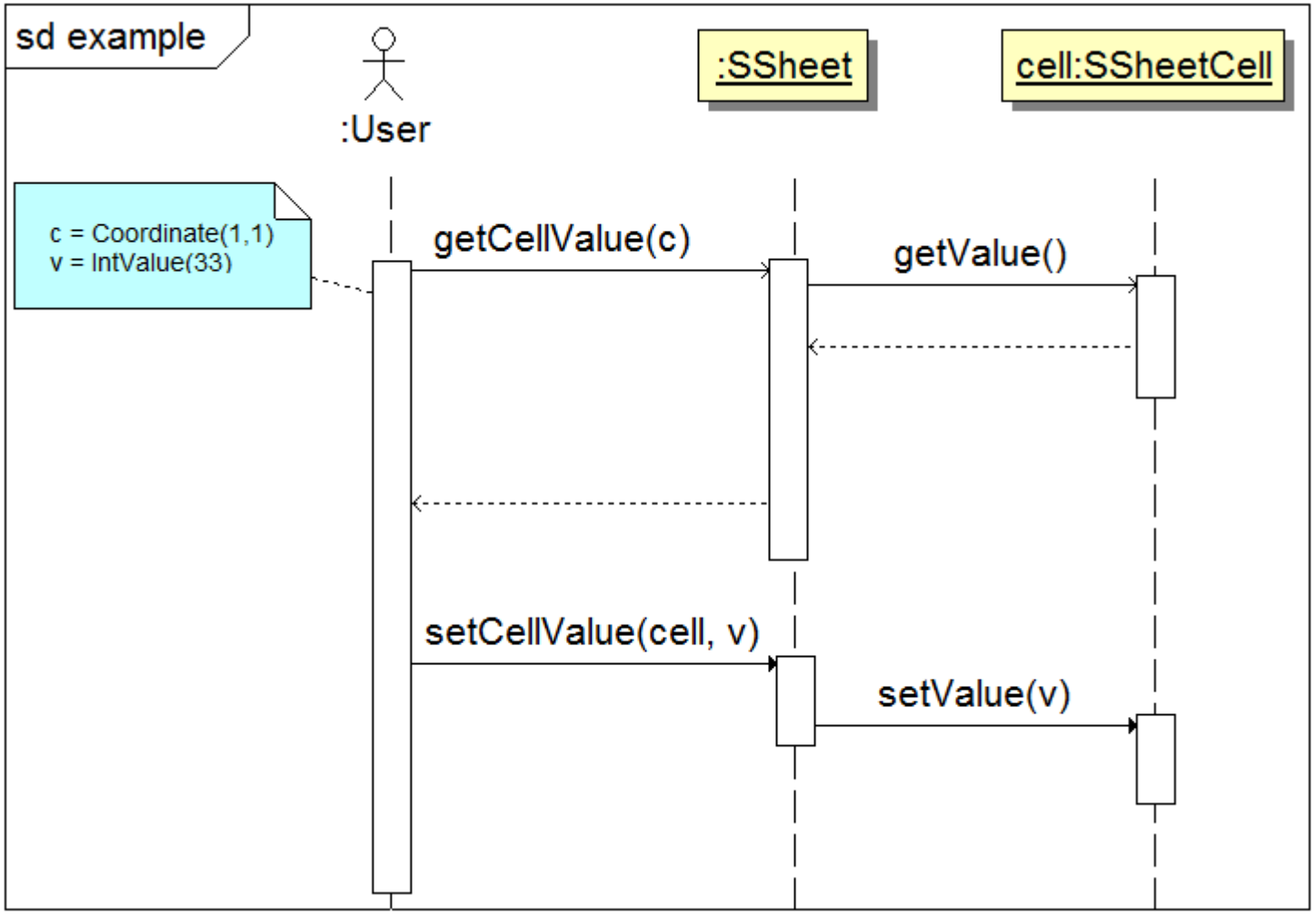


Sequence Diagrams

- Sequence diagrams make temporal ordering explicit.
- However, they do not contain explicit link information (so the correspondence with the class diagram is not as explicit as with communication diagrams).



A closer look

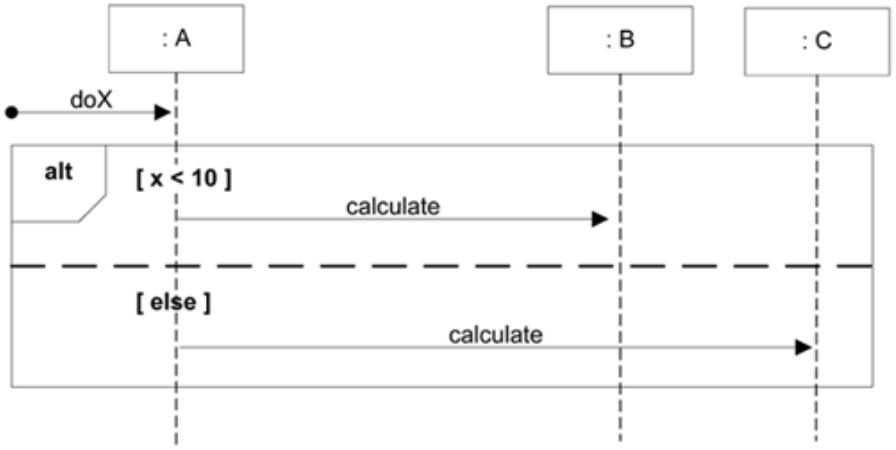


Components of Sequence Diagrams

- Vertical axis: time, increasing downwards.
- Objects that exchange messages in a behaviour trace 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's 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 long the object is active.
  - Explicit time unit may be added.
- Arrows depict messages from a sender object to a target object and the message is written along the arrow.

Types of Fragments

- Alt: Alternative fragment for conditional logic expressed in the guards.
- Loop: Loop fragment while guard is true.
- Break: If guard is true, execute this fragment and jump to end of parent fragment.
- Opt: Optional fragment that executes if guard is true.
- Par: Parallel fragments that execute in parallel.
- Critical: Critical region within which only one thread of control active at a time. No concurrent region (e.g.: par) may execute at the same time.
- Ref: Reference to another diagram.
- Assert: This behaviour is the only valid at that point.
- Seq: Weak sequencing of messages; no order of reception.



Synchronous vs. Asynchronous

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

Synchronous Messages

- The sender object waits until target object finishes its proccessing 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 multiple threads of control.
  - many objects can be active at any time
  - this is also known as concurrency

Depicting Asynchronous Messages

- Instead of using a filled arrowhead, we use an open arrowhead (in both communication and sequence diagrams).
- In sequence diagrams
  - we may have two objects active at the same time (box).
  - The sender object remains active after sending a message. The target object becomes active as well.
- If the target object can accept multiple messages, how does it handle them?

Concurrency

- 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)

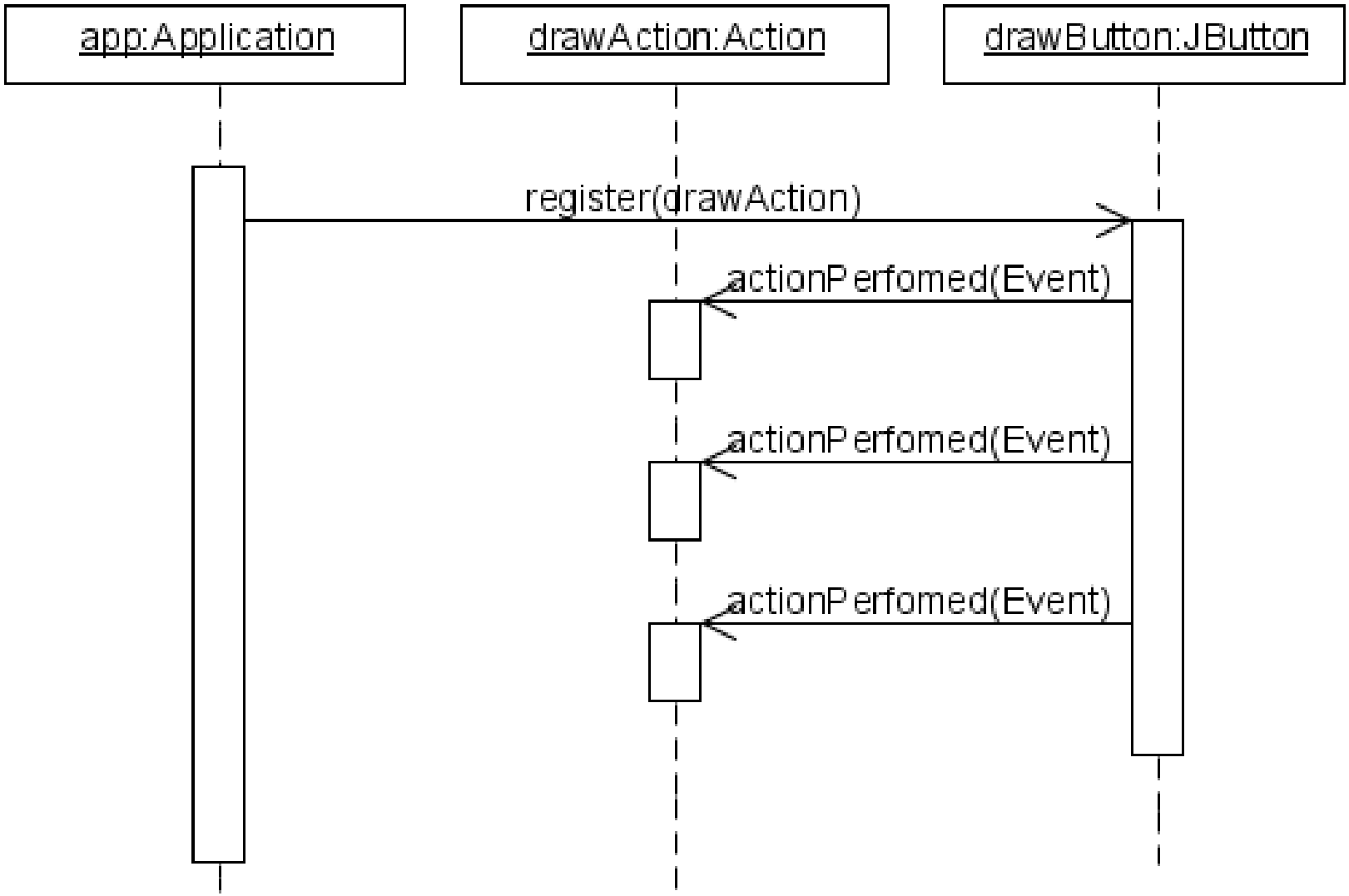
Message Priorities

- 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 one message is more important than others.
- You can use priority levels to determine the order messages are processed.
- What are the dangers of this?

Callback Mechanism

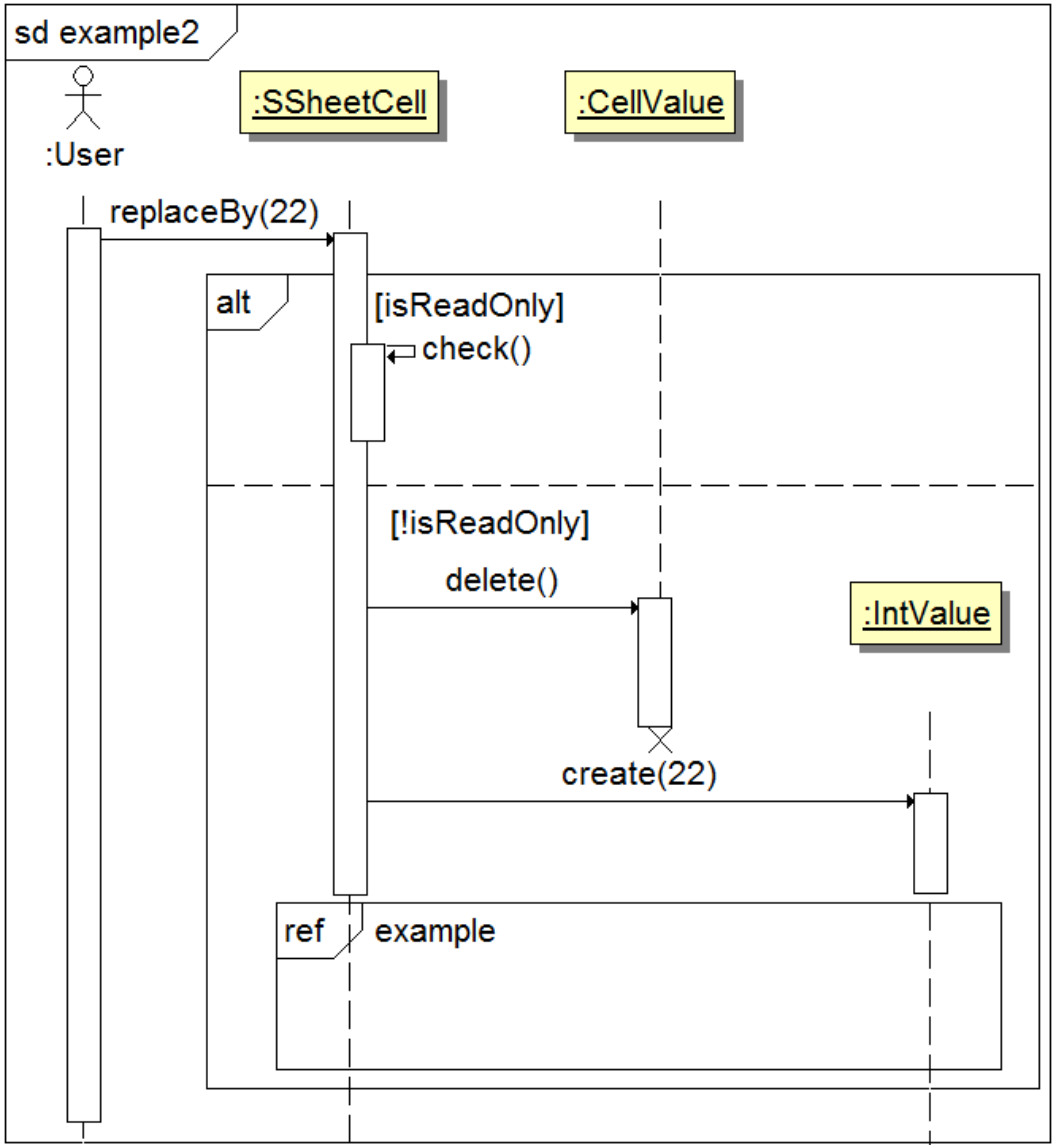
- Uses asynchronous messages.
- A subscriber object o1 is interested in an event e that occurs in o2.
- o1 registers interest in e by sending a message (that contains a reference to itself) to o2 and continues its execution.
- When e occurs, o2 will callback asynchronously to o1 (and any other subscribers).

Callback (but not to self) illustrated



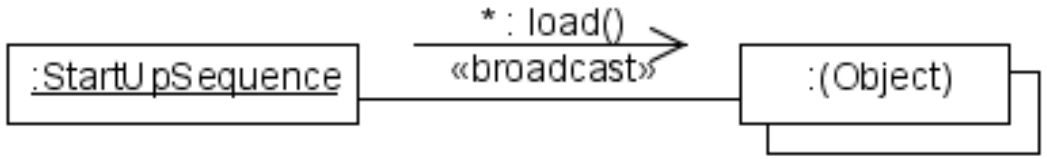
Object creation/destruction

- Sequence diagrams use
  - A special method sent to the object (not its lifeline) to denote object creation.
  - 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 sent to all the objects in the system.



- If only a specific category of objects is targeted, we call this narrowcast.