



Interaction Diagrams

Stevens & Pooley, chapters 9 & 10

Alhir, chapters 9 & 10

What are Interaction Diagrams?



- Interaction diagrams show how objects in a system interact with each other, and with actors outside the system
- Two kinds of interaction diagram:
 - Collaboration diagrams
 - Sequence diagrams

Further Uses of Interaction Diagrams



- Show how a class provides an operation
- Describe how a design pattern works
- Describe how a component can be used



Collaboration Diagrams

- Show collaborations between objects and actors via links
- Objects:
 - labelled `objectName: className`
 - not all classes need be present
 - more than one object of a class may be present
 - object names can be omitted



Collaboration Diagrams

- Links:
 - can be labelled
 - can be navigable
 - should map to an association in the class diagram
 - links do not need to be shown for every association



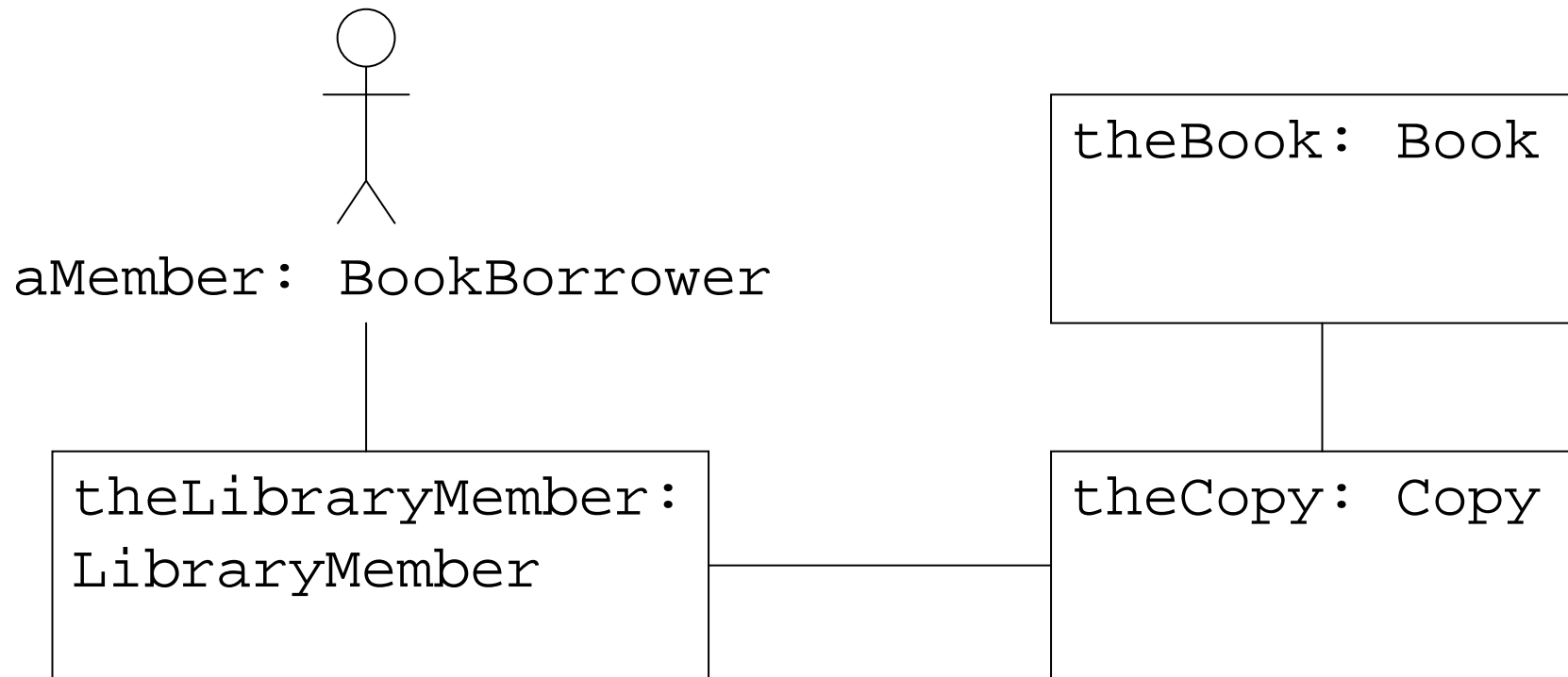
Collaboration Diagrams

- Actors:
 - shown as on a use case diagram
 - should map onto an actor in the use case diagram being realised
 - may be several actors, but one initiates the use case and is called the *initiator*



Collaboration Diagrams

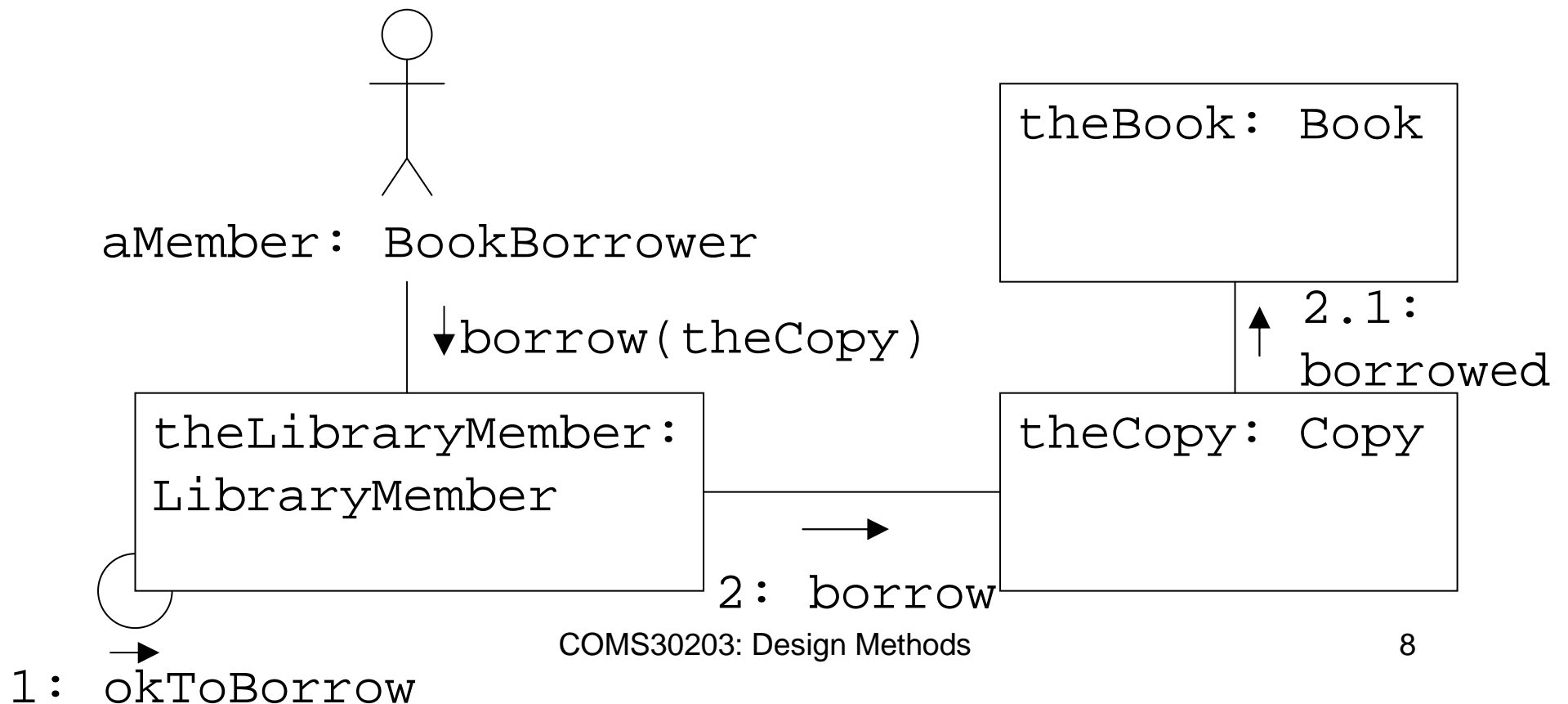
- A simple example without interaction





Collaboration Diagrams

- The same example with interaction





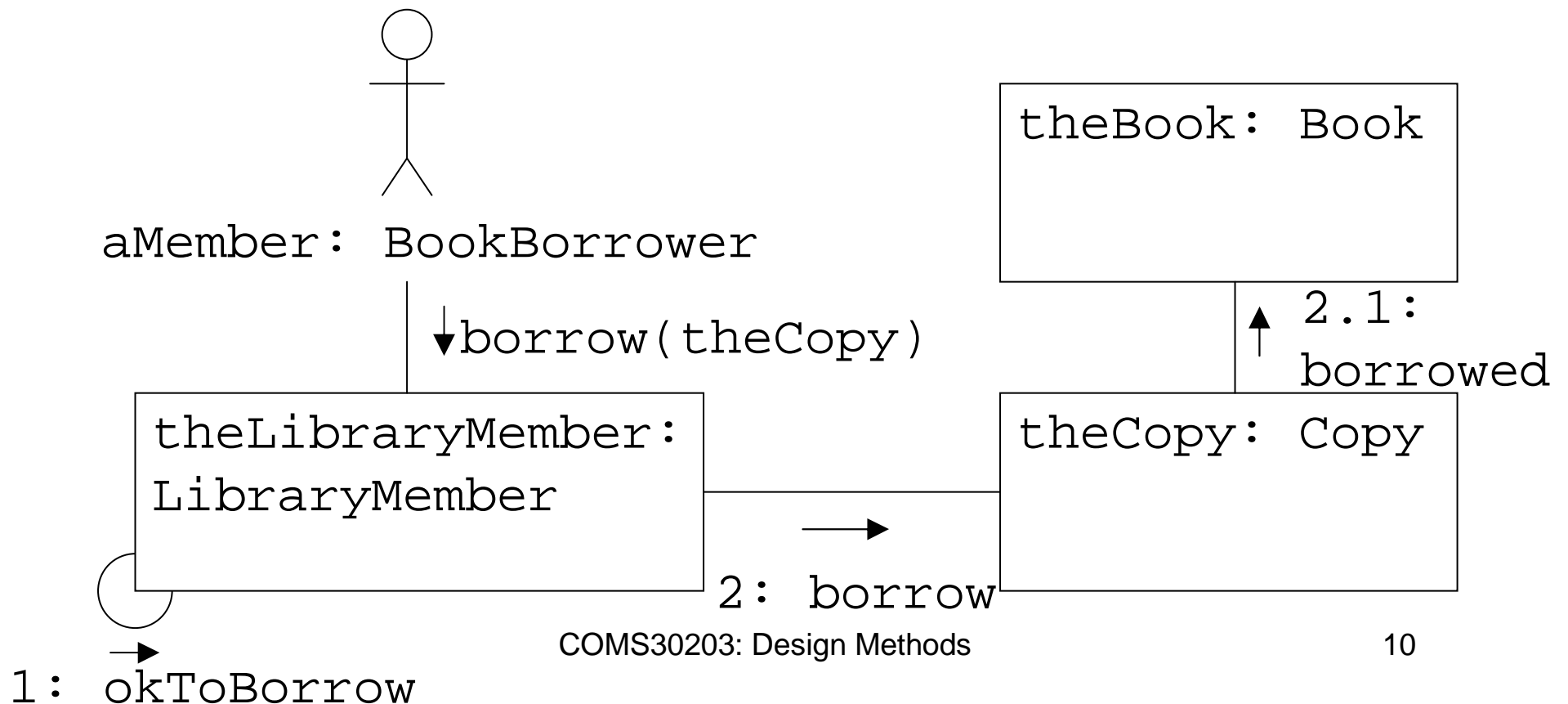
Collaboration Diagrams

- Procedural systems:
 - Only one object is computing at a time
 - Messages are *synchronous*
 - Objects have live *activation* between receiving a message and replying to it
 - Objects can only compute between receiving a message and replying to it, but not while waiting for a reply to a sent message
 - Nested activation reflected by message numbering
 - Only actors can initiate activity



Collaboration Diagrams

- The same example with interaction





Collaboration Diagrams

- Technical note
 - Collaboration diagrams can be instance level or specification level
 - i.e. they can use objects, roles, or both
 - (can specify different roles that objects of a class can play)
 - Stevens & Pooley suggest using objects that are “informally representative”

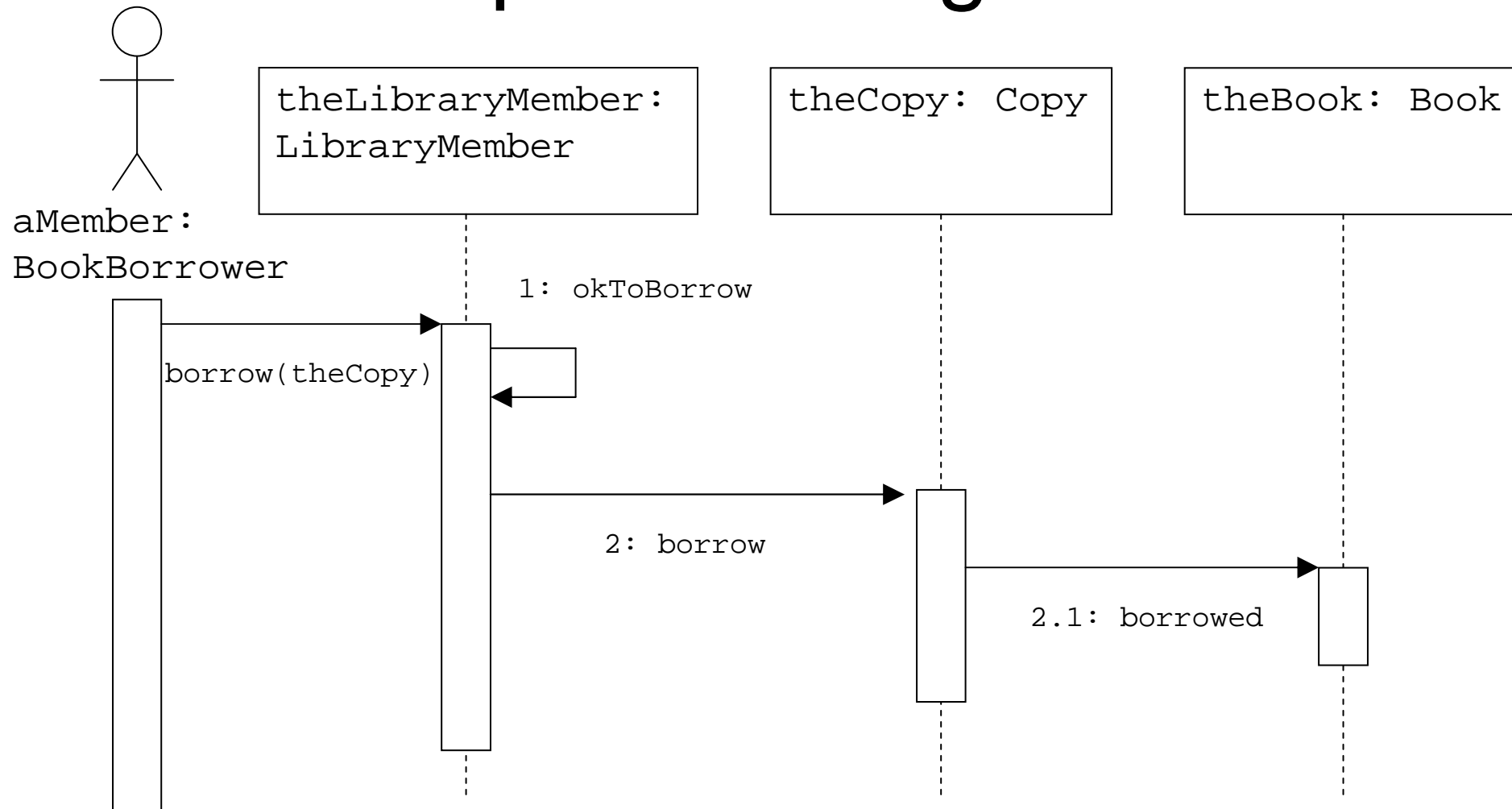


Sequence Diagrams

- Represent the order in which interactions between objects and actors occur
- *Lifelines* of objects are represented as dashed vertical lines
- Time passes from top to bottom of diagram
- Messages are represented by arrows between lifelines
- Live activation of an object is represented by a narrow rectangle covering its lifeline



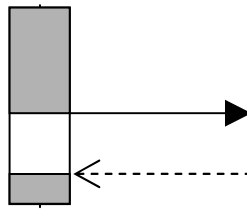
Sequence Diagrams



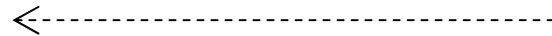


Sequence Diagrams

- Optional additional detail
 - Indication of when objects are computing
 - Shade portions of live activation rectangle



- Return messages



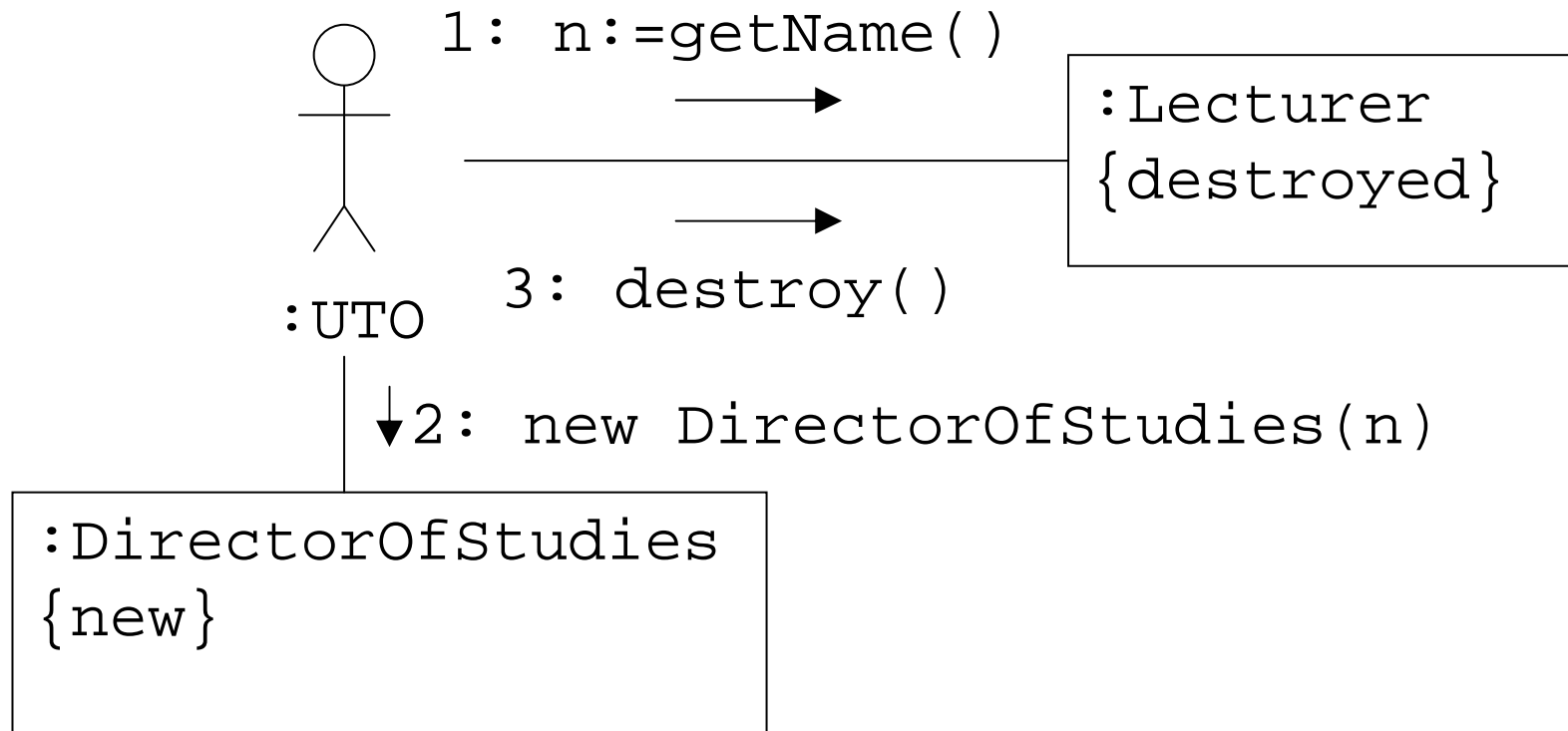


Interaction Diagrams

- More advanced features
 - Return values
 - variables bound for use elsewhere in the diagram
 - Creation and deletion of objects
 - `{new}`, `{destroyed}`, `{transient}`
 - Timing
 - time taken for messages to travel
 - constraints on message travel time and computation time

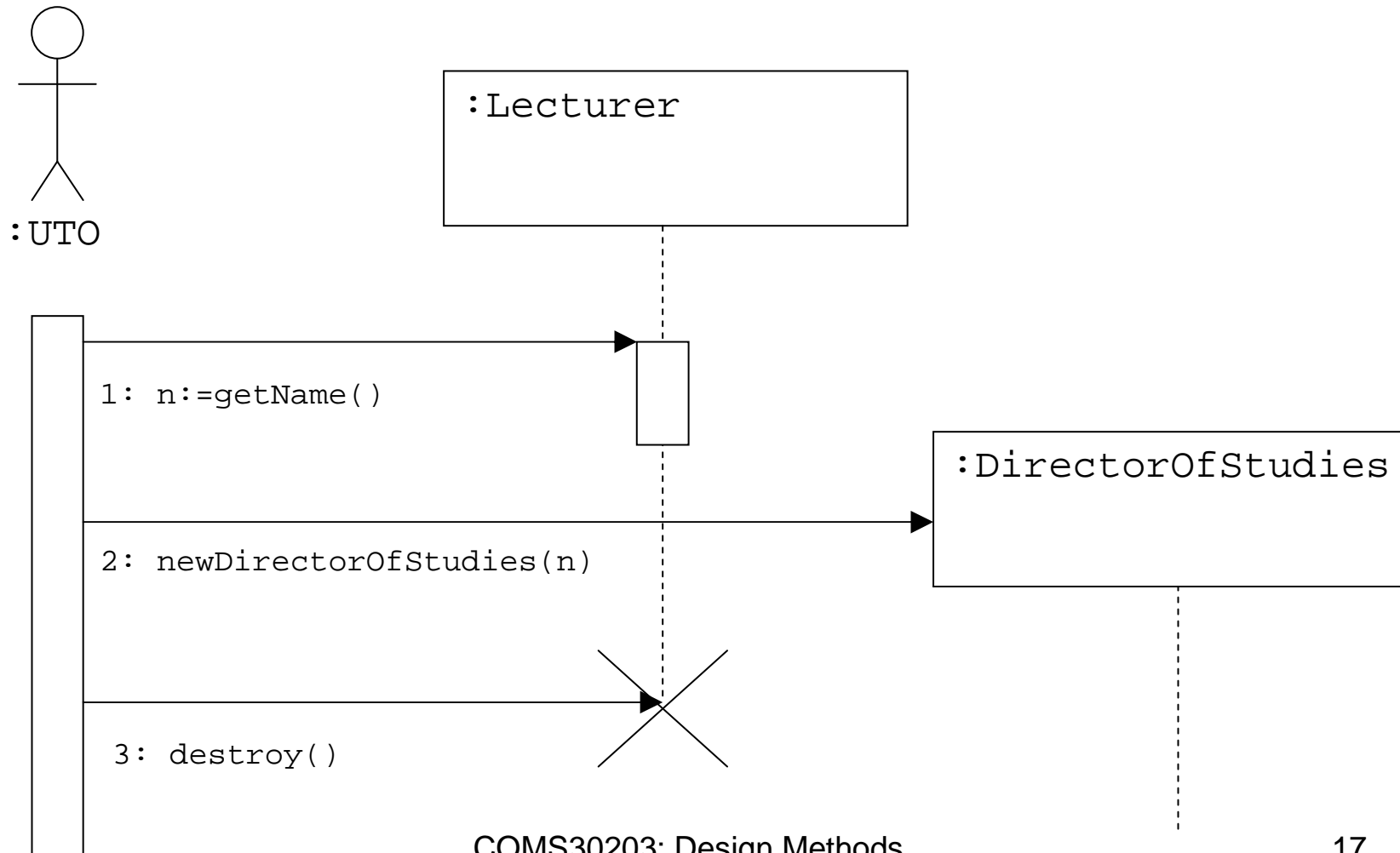


Collaboration Diagrams



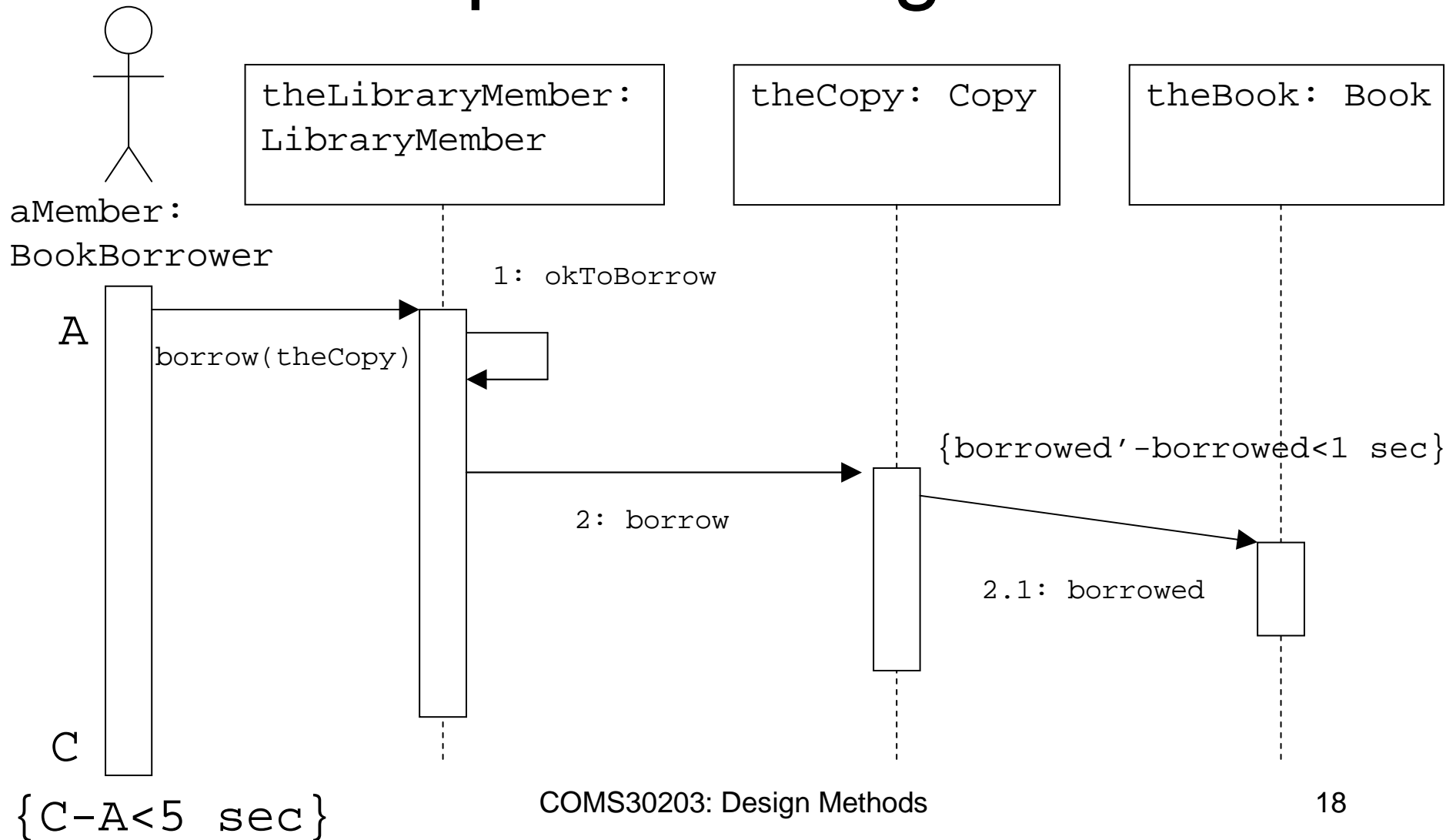


Sequence Diagrams

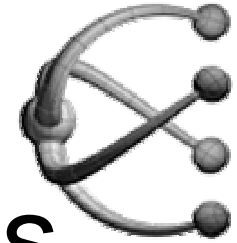




Sequence Diagrams



Generic Interaction Diagrams



- We have just looked at *instance forms* of interaction diagrams, showing one possible sequence of messages
- Use cases can describe multiple scenarios
- A *generic* interaction diagram shows all possible sequences of messages
- This requires conditional behaviour...

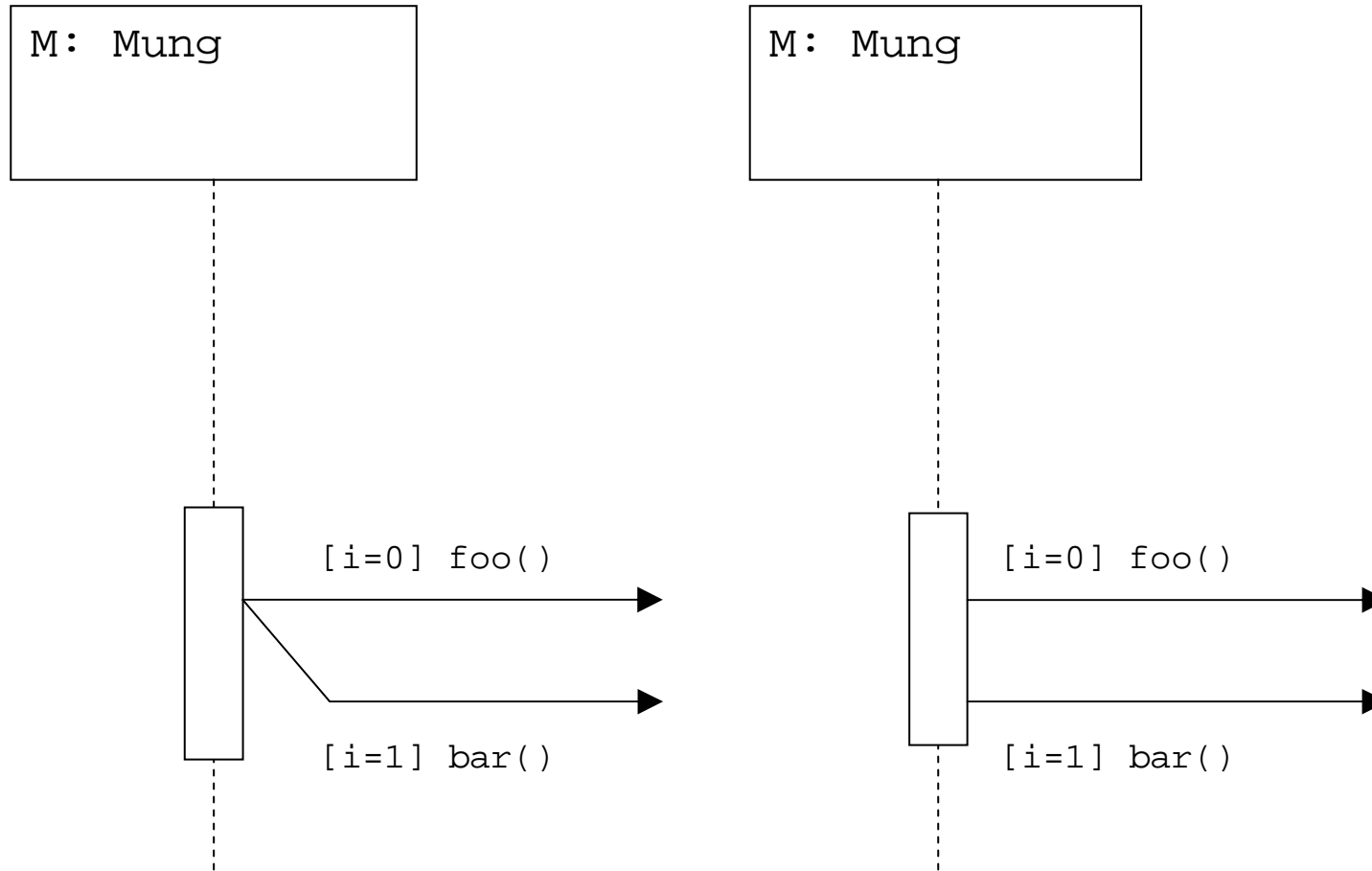


Conditional Behaviour

- Guards
 - Message may be guarded by a conditional statement
 - Message is only sent if guard evaluates as true when it is reached
 - Multiple guarded messages might have the same origin
 - At most only one guard should evaluate to true in a sequential system
 - After such a decision point, future behaviour of the same object might be different

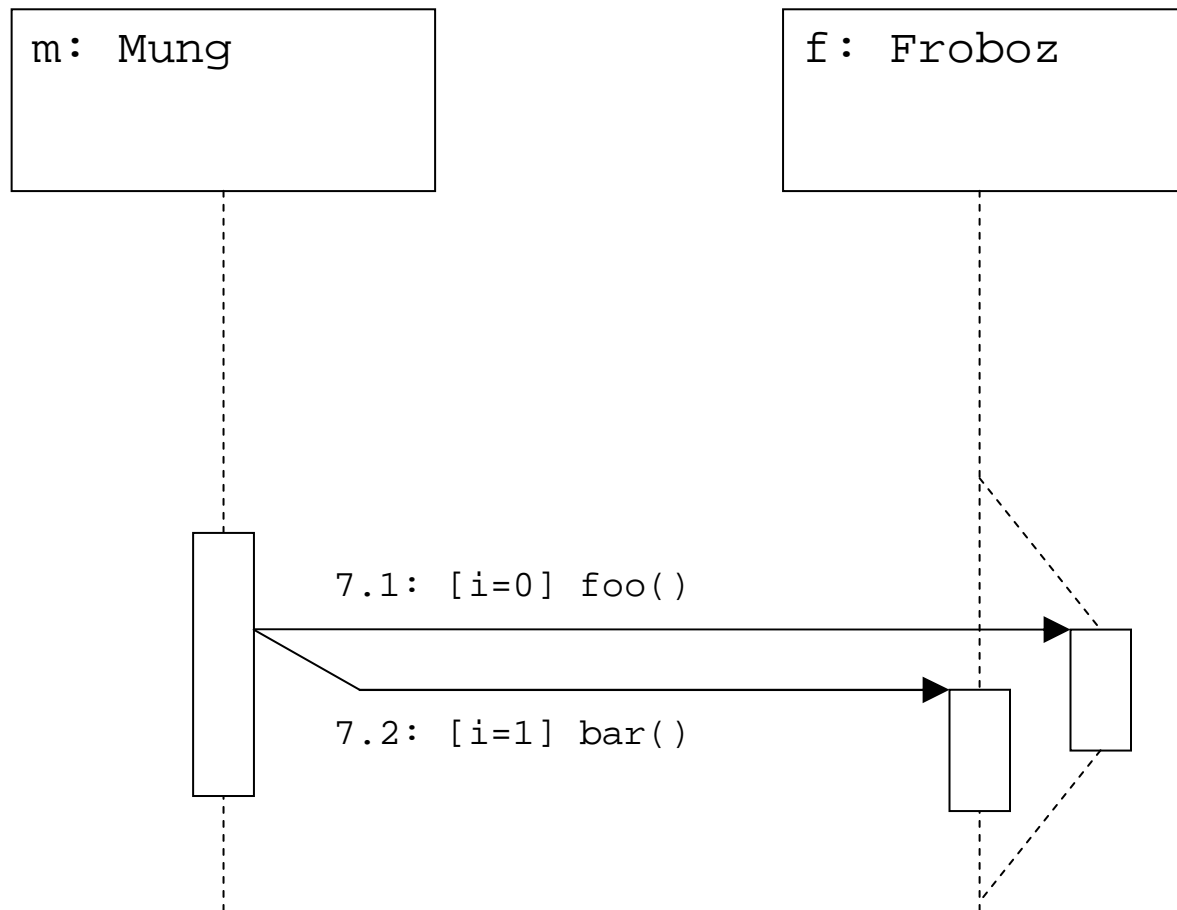


Conditional Behaviour





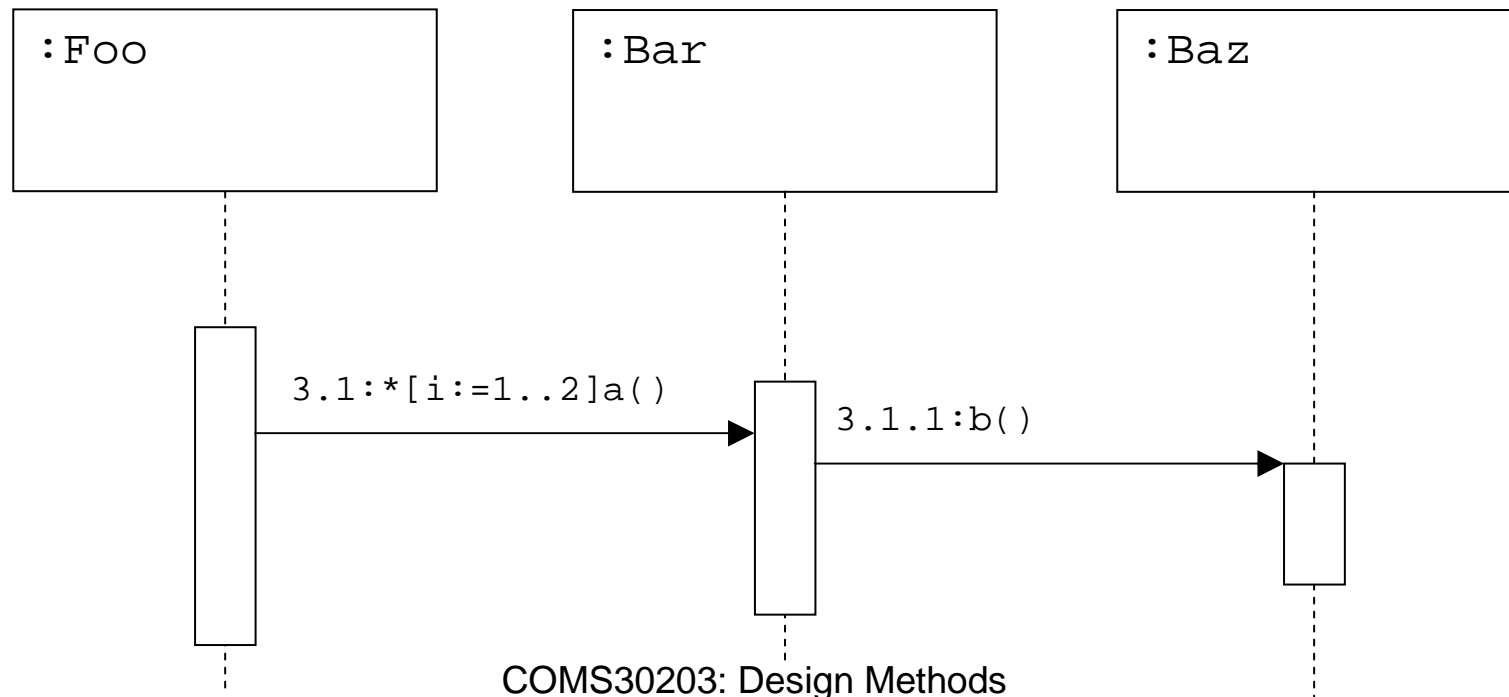
Conditional Behaviour





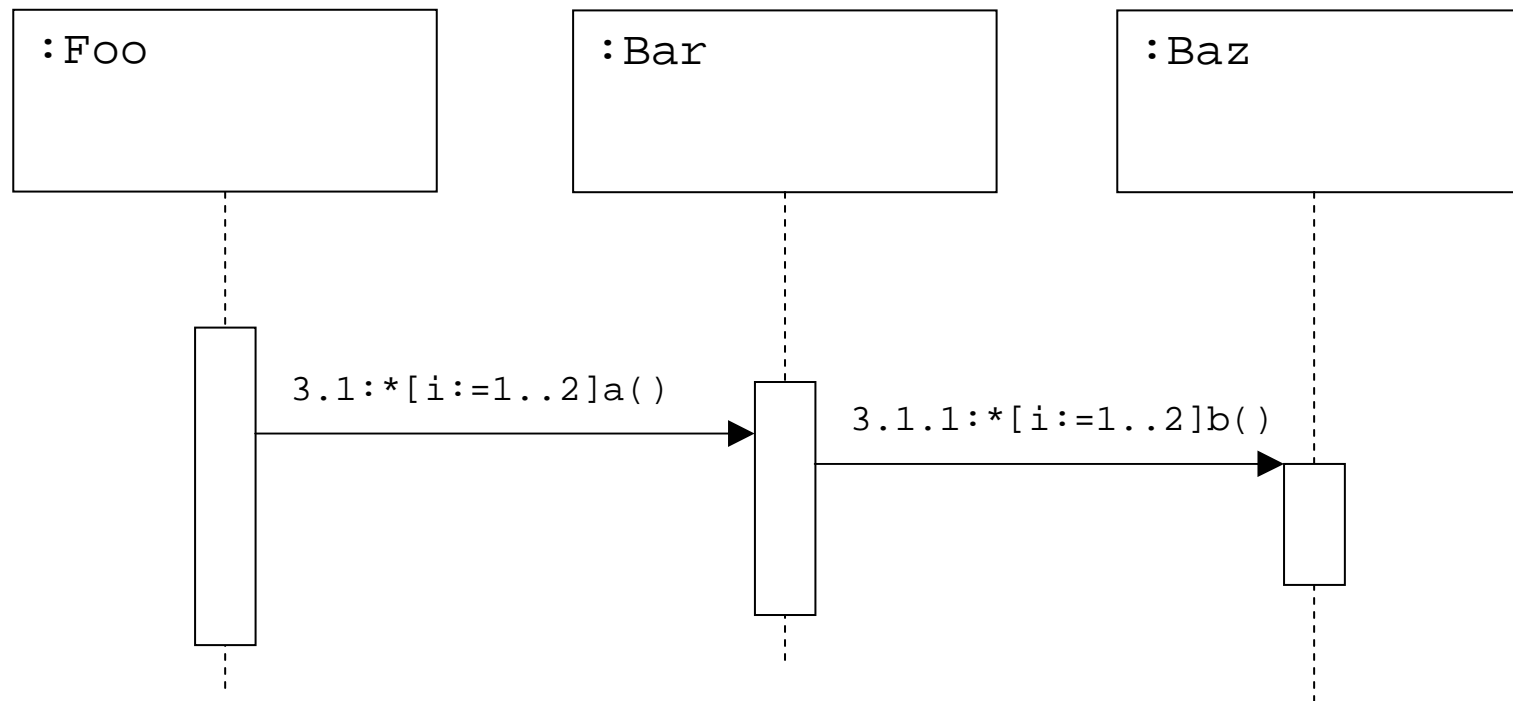
Iteration

- Can represent do...while loops and for...next loops





Iteration



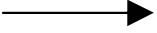
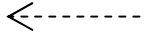
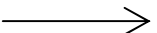


Concurrency

- Threads can split
 - Modelled by two or more non-exclusive messages from the same point
 - Concurrent messages can be labelled with strings
- *Active objects* can spontaneously originate messages (drawn with a heavy border)



Concurrency

- Flat messages
 - Sender does not expect a reply
- Asynchronous messages
 - Sender does not expect a reply, and continues computing
- Message types
 - Synchronous (call) 
 - Return 
 - Flat 
 - Asynchronous 