# Applying UML & Patterns (3rd ed.)

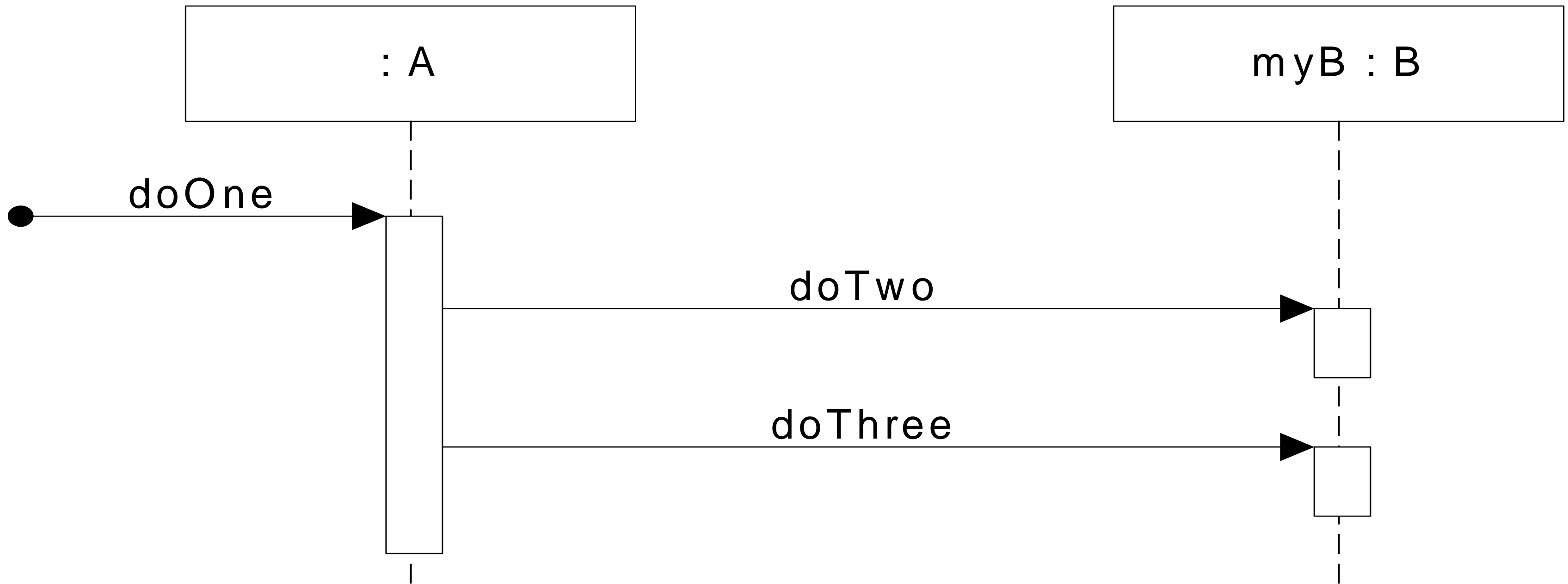
# Chapter 15

## UML INTERACTION DIAGRAMS

**This document may not be used or altered without the express permission of the author.**

**Dr. Glenn L. Ray**
School of Information Sciences
University of Pittsburgh
gray@sis.pitt.edu     412-624-9470

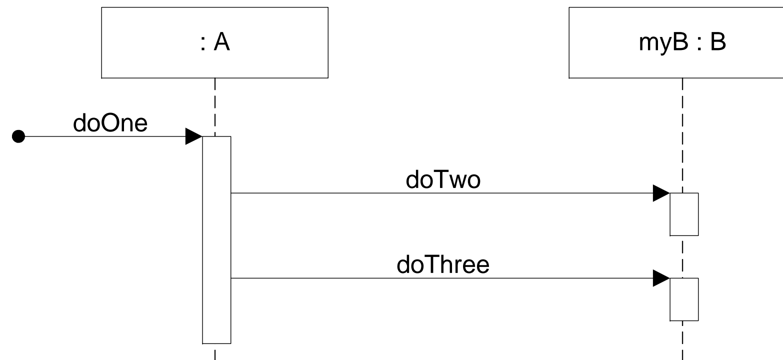
# Fig. 15.1




- Sequence diagram’s ‘fence’ format
- Time is increasing in the downward direction


Copyright © 2004
Glenn L. Ray or Craig Larman

# Fig 15.1



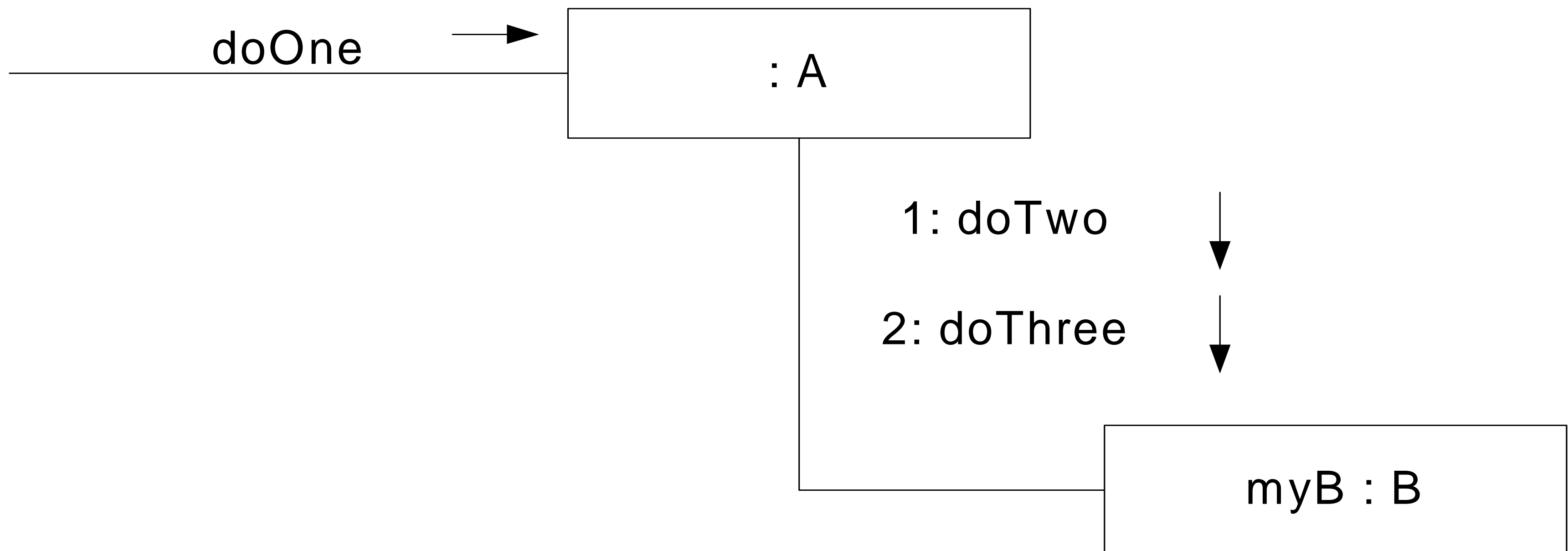

```
public class A
{
	private B myB = new B();

	public void doOne()
	{
			myB.doTwo();
			myB.doThree();
	}
}
```



Copyright © 2004
Glenn L. Ray or Craig Larman

# Fig. 15.2




Same collaboration using communication diagram
Uses network (or graph) format


Copyright © 2004
Glenn L. Ray or Craig Larman

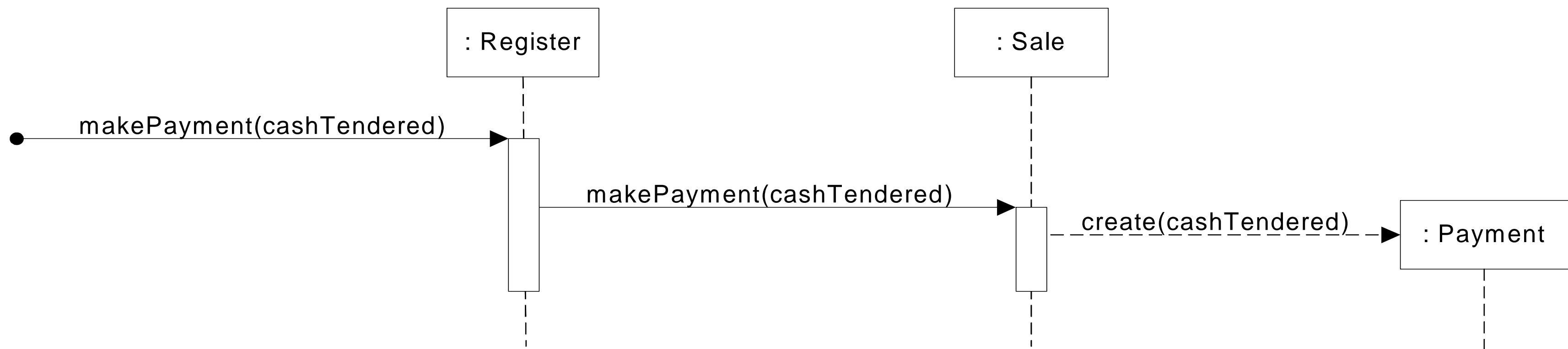
# Interaction Diagrams

- Sequence vs. Communication diagrams
  - Sequence Diagrams
    - Easier to see sequence of method calls over time
    - More expressive UML notation
    - Better support from many tools
  - Communication Diagrams
    - Better fit on limited space (page or whiteboard)
    - Easier to edit/amend
    - Look more like class diagram



Copyright © 2004
Glenn L. Ray or Craig Larman

# Fig. 15.3

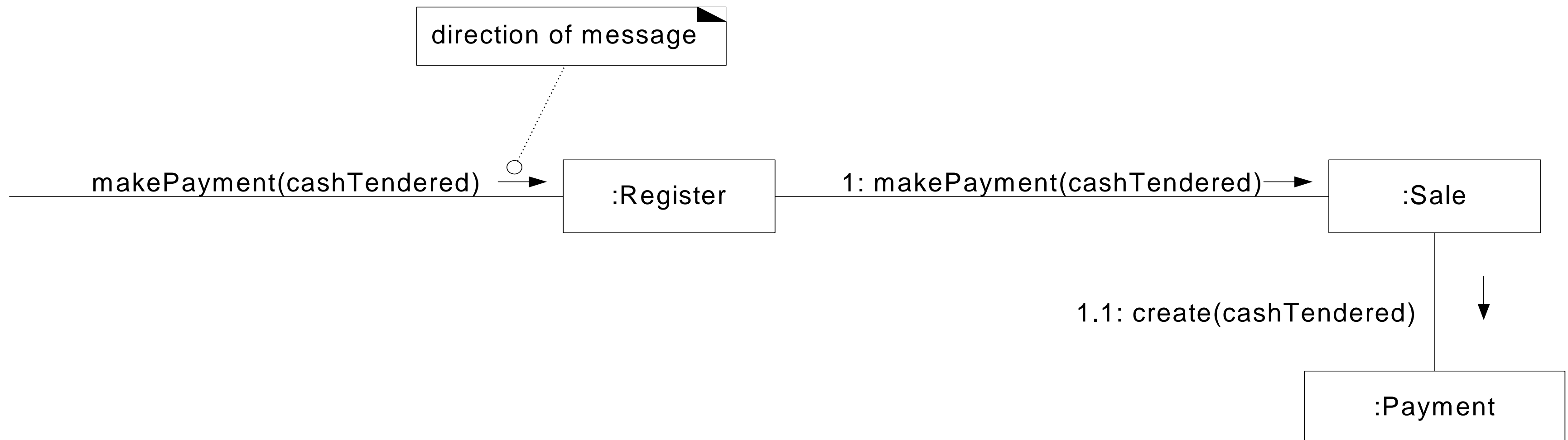



1. Someone(?) sends makePayment(..) msg to Register
2. Register sends makePayment(..) msg to Sale
3. Sale creates an instance of Payment

Who created Register & Sale? IDs show a fragment of system behavior during isolated snapshot in time. This can be confusing and lead to modeling errors/omissions!


Copyright © 2004
Glenn L. Ray or Craig Larman

# Fig. 15.4




Same collaboration using communication diagram


Copyright © 2004
Glenn L. Ray or Craig Larman

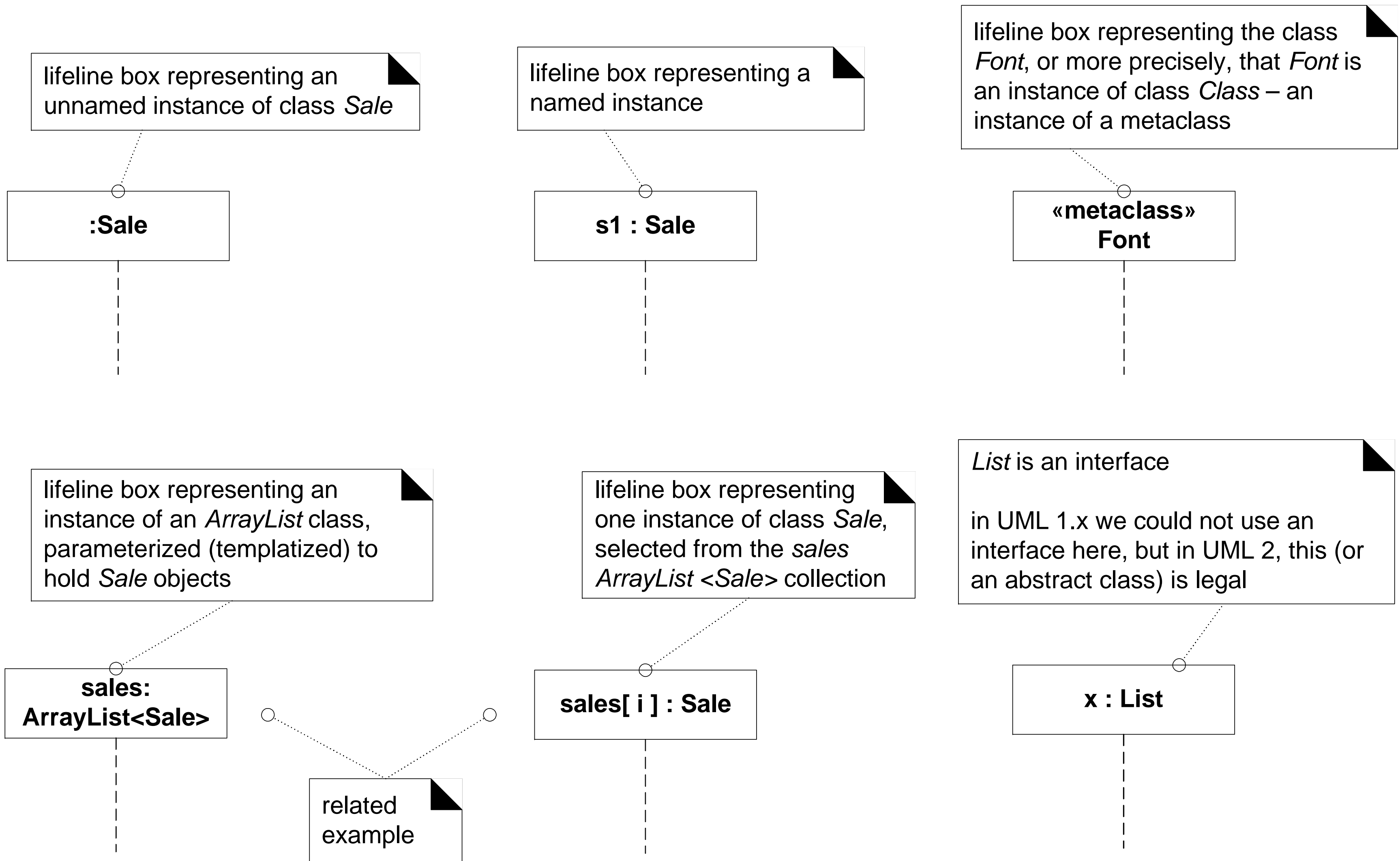
# Interaction Diagrams

- Essential UML models for OOAD

1. Use cases
   – Functional requirements
2. Class diagram
   – Objects with knowledge (attributes) and behavior (operations)
   – Static relationships between objects
3. Interaction diagrams
   – Dynamic collaboration between objects


Copyright © 2004
Glenn L. Ray or Craig Larman

# Fig. 15.5





Copyright © 2004
Glenn L. Ray or Craig Larman

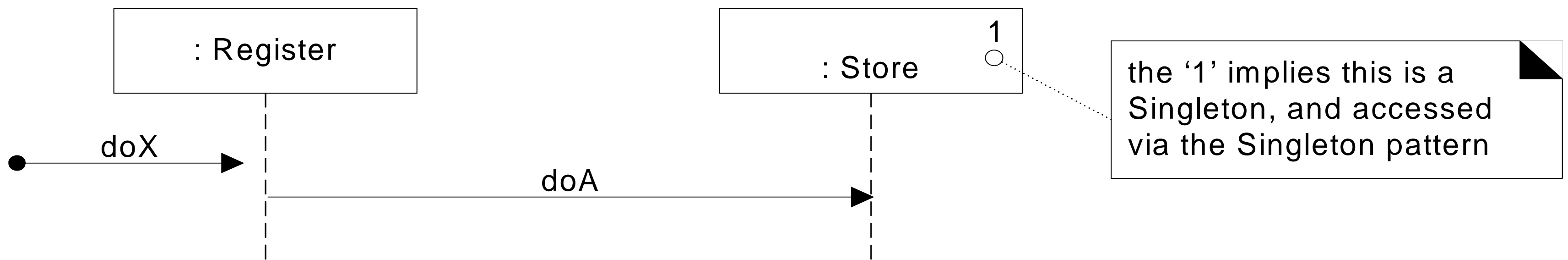
# Interaction Diagrams

- UML expression syntax
  - Don’t have to use full syntax in all cases

return = msgName(param:paramType1, …) : returnType


Copyright © 2004
Glenn L. Ray or Craig Larman

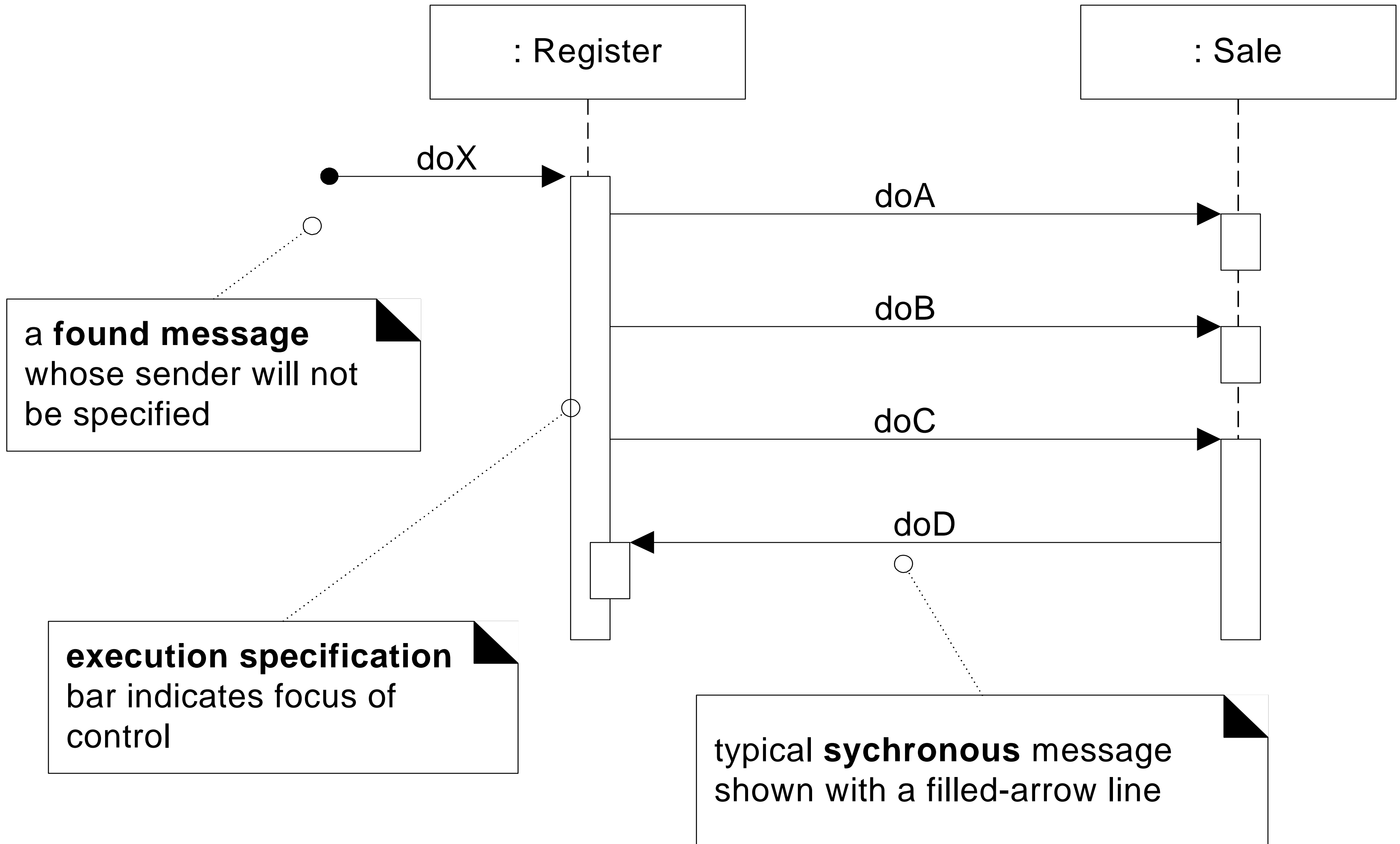
# Fig. 15.6




Notation for Singleton


Copyright © 2004
Glenn L. Ray or Craig Larman

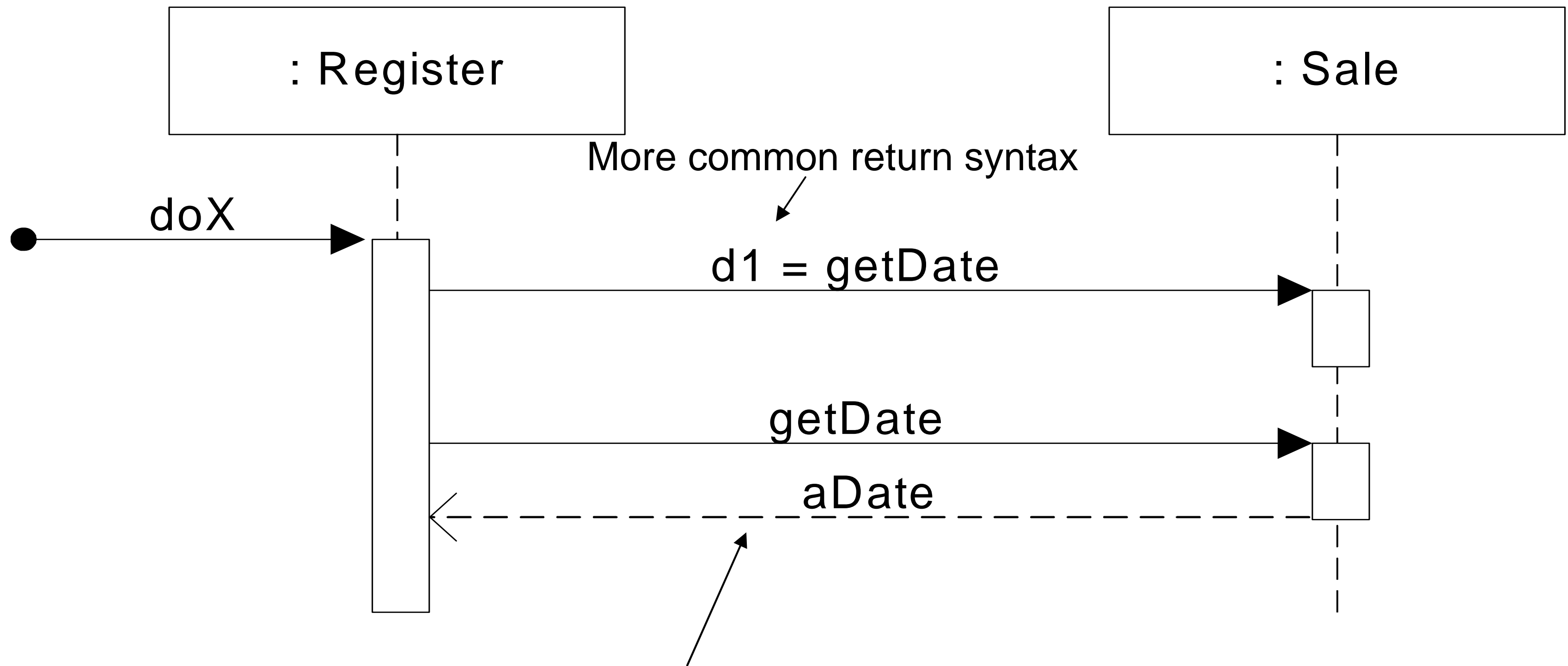
# Fig. 15.7






Copyright © 2004
Glenn L. Ray or Craig Larman

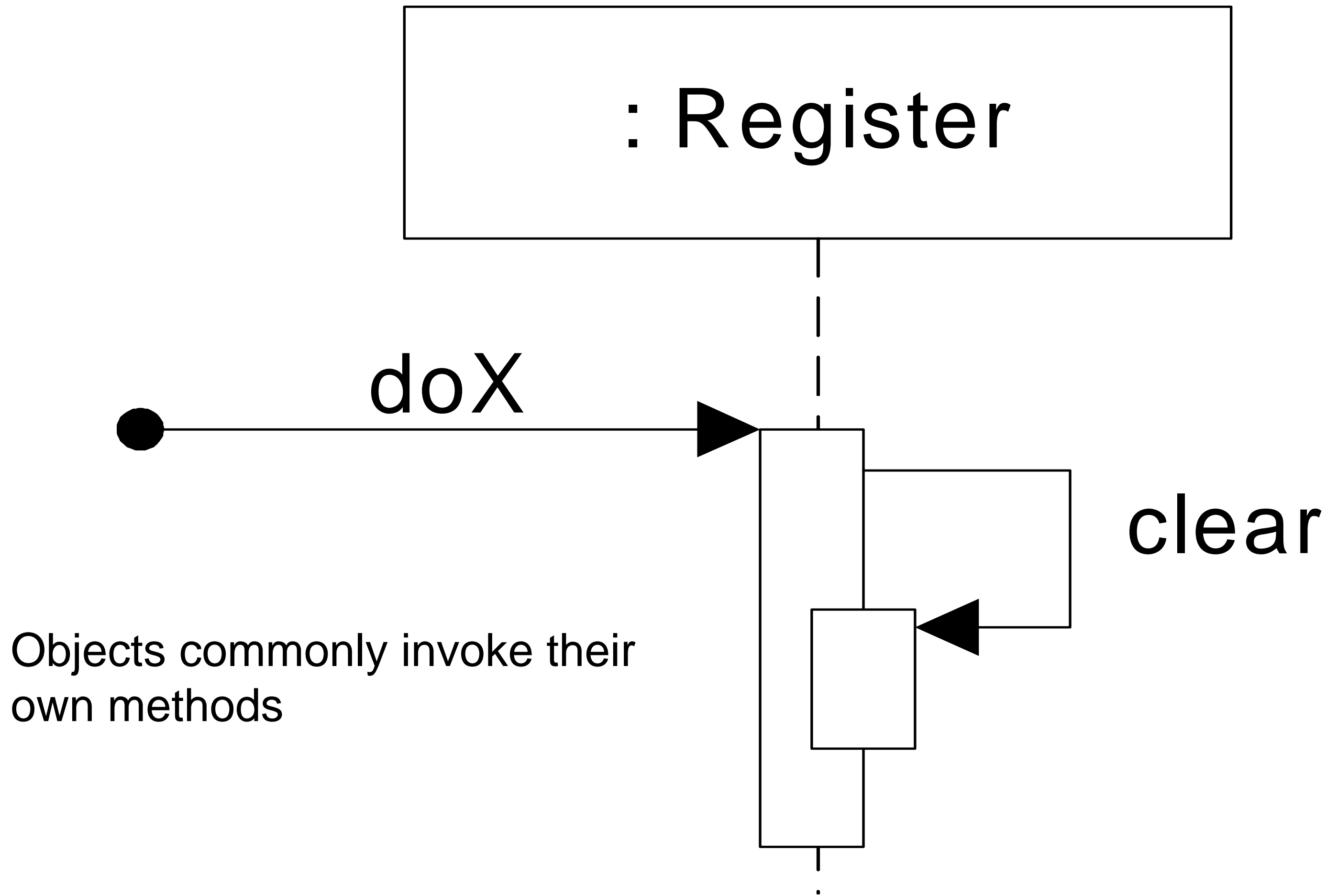
# Fig. 15.8






Copyright © 2004
Glenn L. Ray or Craig Larman

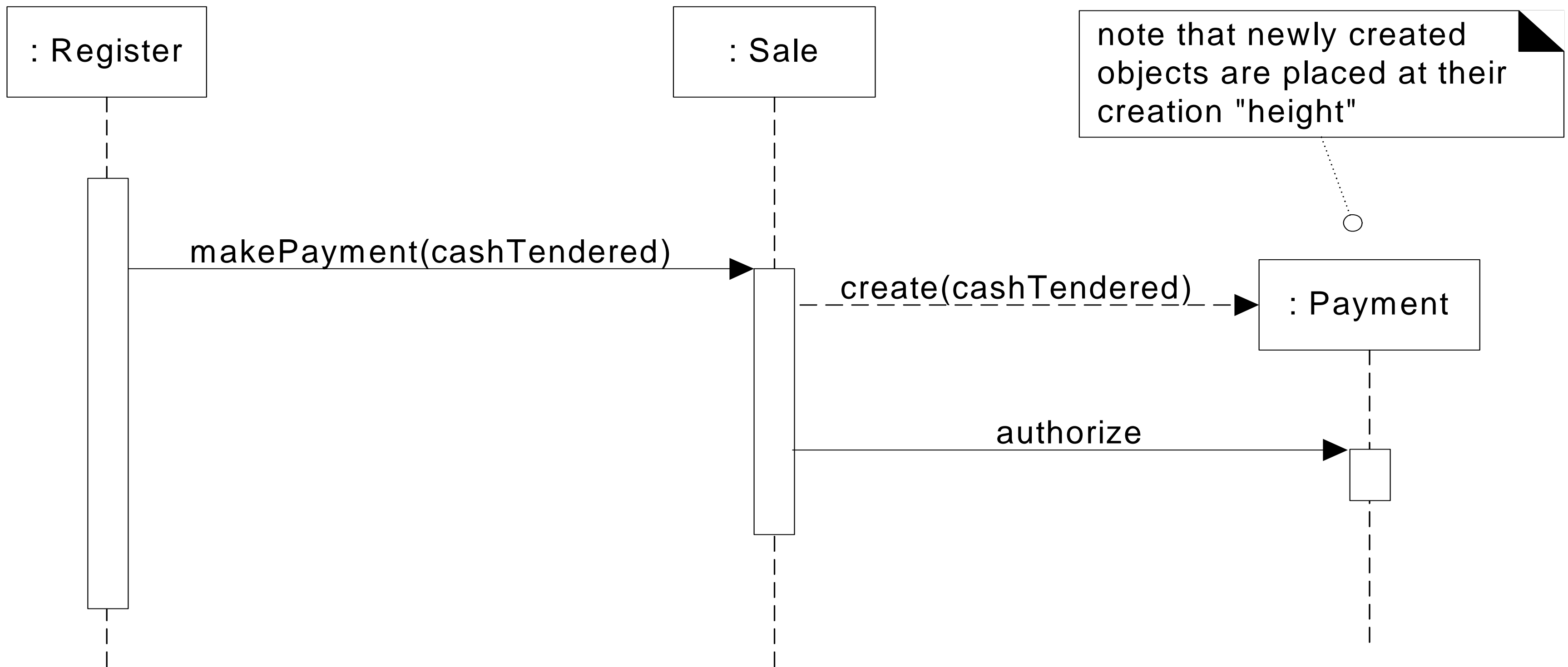
# Fig. 15.9





Copyright © 2004
Glenn L. Ray or Craig Larman

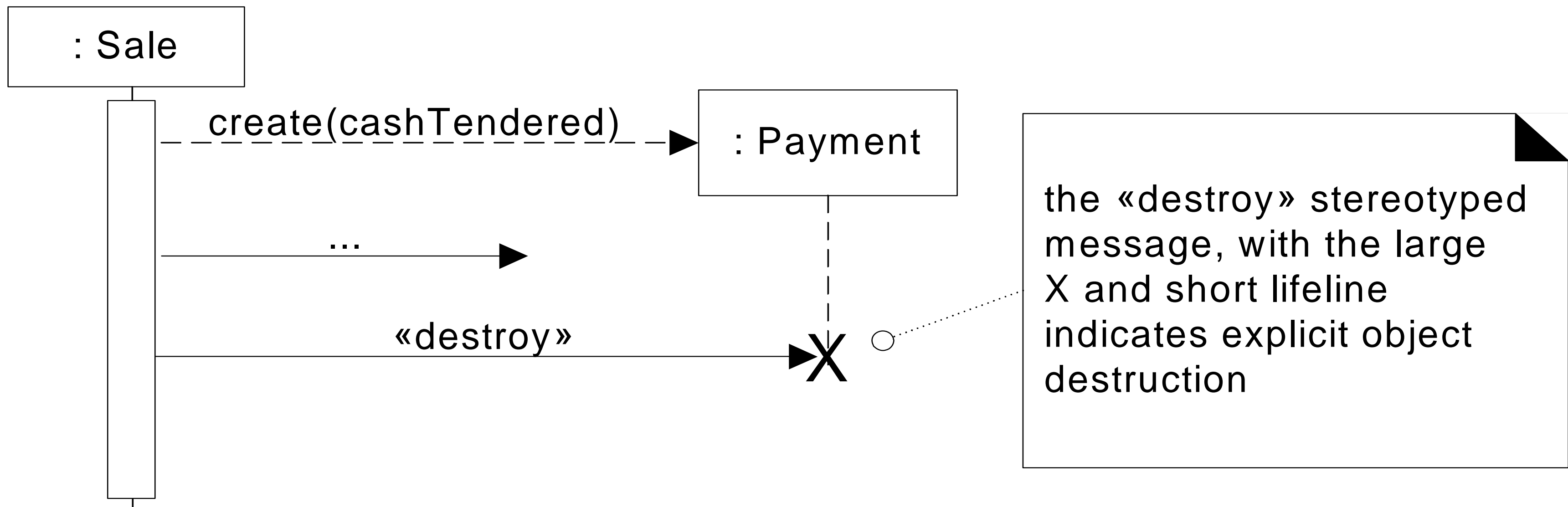
# Fig. 15.10




Dashed line for 'create' really not needed, though its now official UML
Use 'create' for calls to a constructor


Copyright © 2004
Glenn L. Ray or Craig Larman

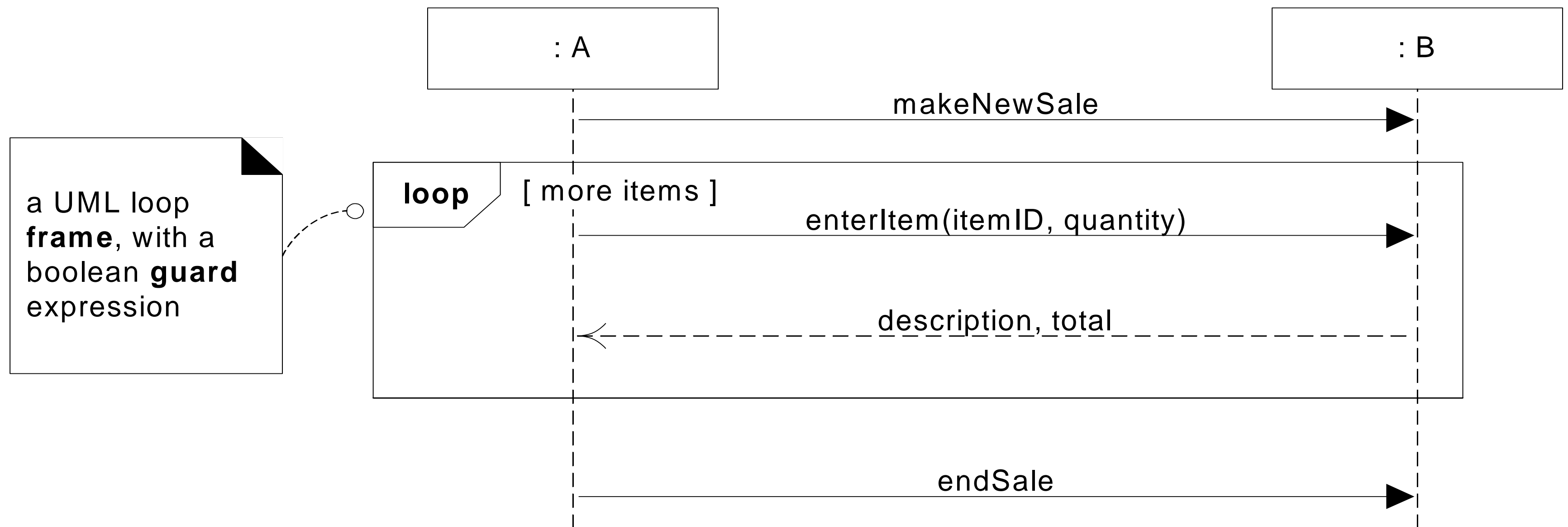
# Fig. 15.11




Usually not necessary for languages with automatic garbage collection (e.g., Java)


Copyright © 2004
Glenn L. Ray or Craig Larman

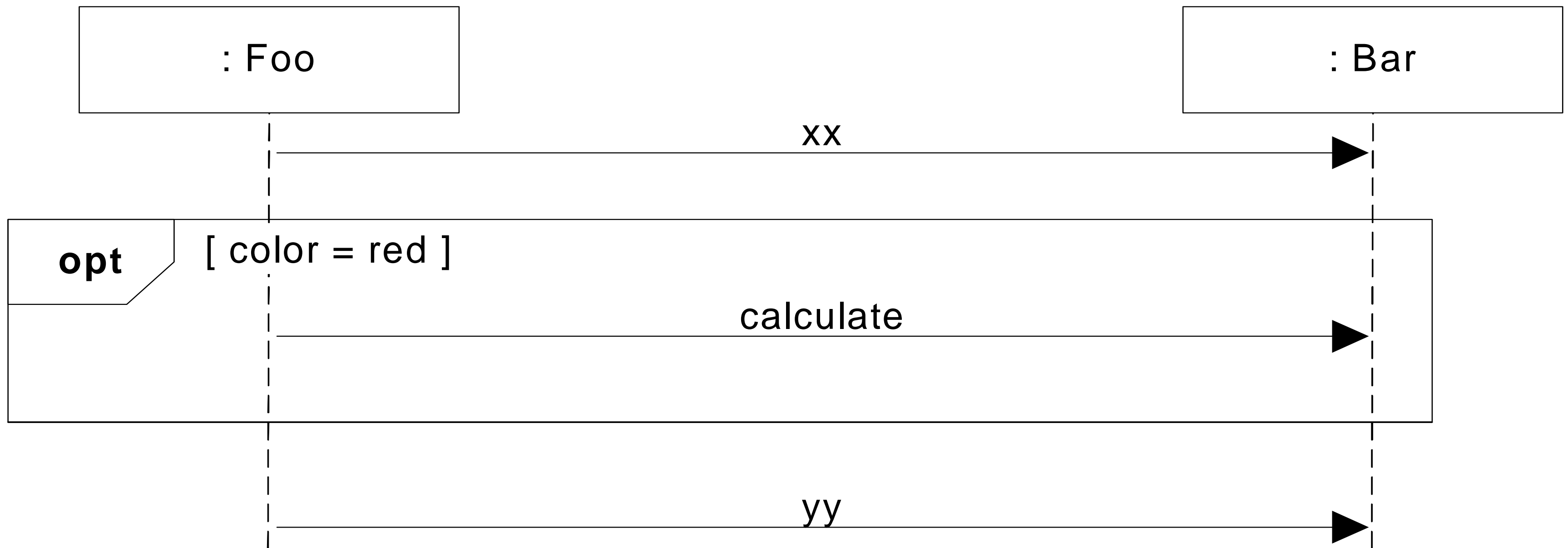
# Fig. 15.12




UML Loop frame


Copyright © 2004
Glenn L. Ray or Craig Larman

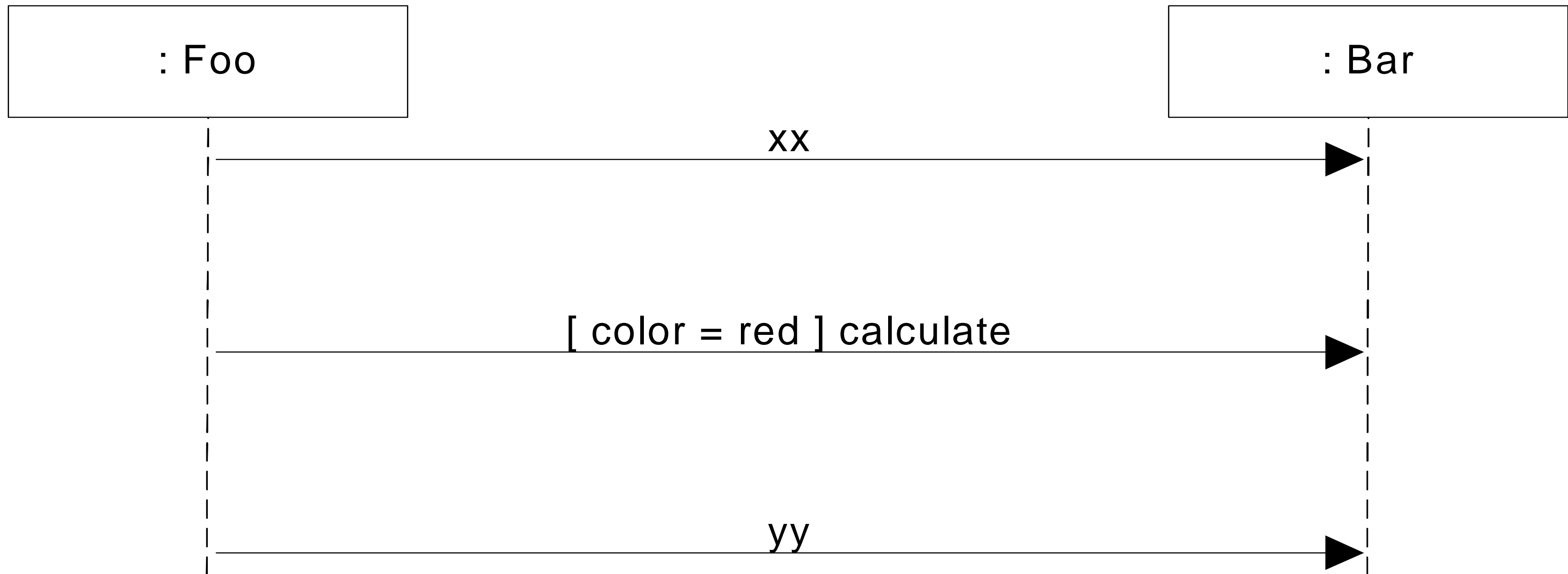
# Fig. 15.13




UML 2 frame showing an optional message


Copyright © 2004
Glenn L. Ray or Craig Larman

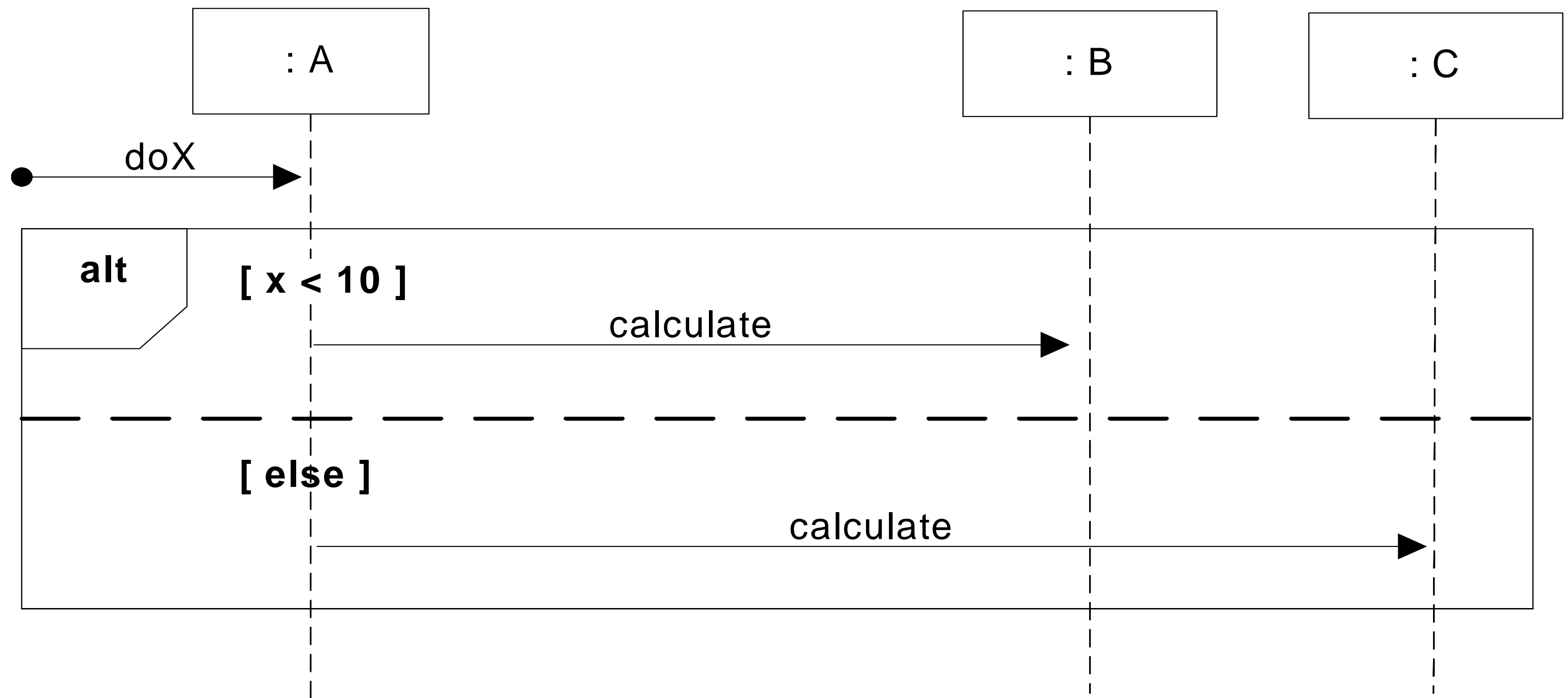
# Fig. 15.14




The same ID using pre-UML 2 notation
Guards must evaluate to true for the message to be sent


Copyright © 2004
Glenn L. Ray or Craig Larman

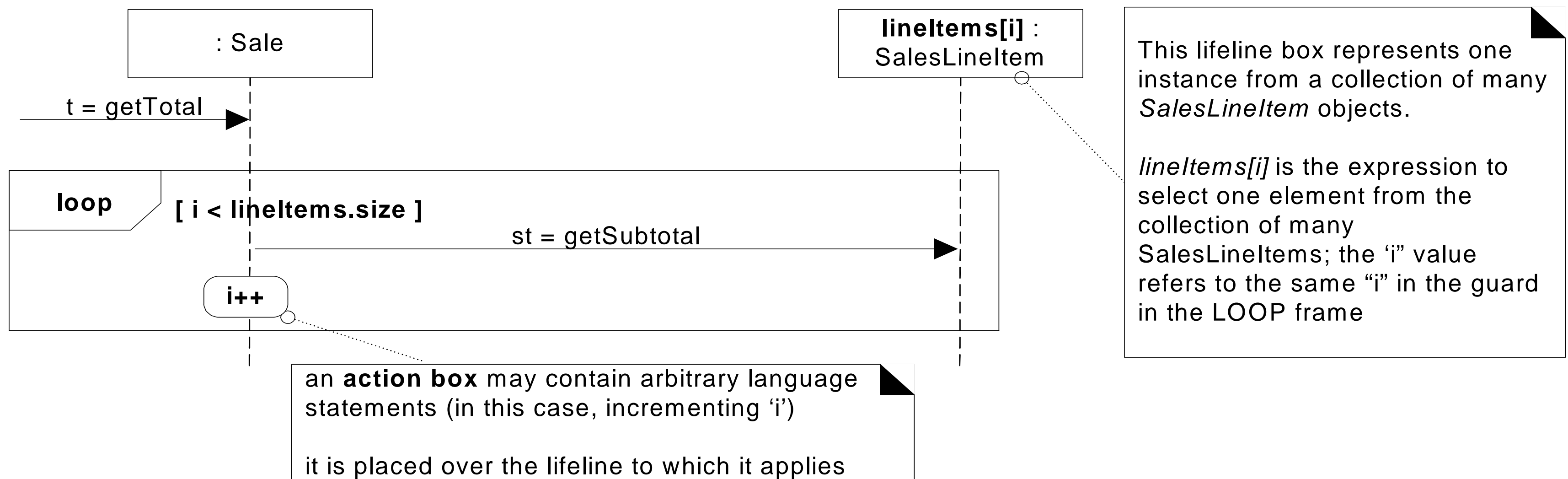
# Fig. 15.15




Alt frame show mutually exclusive interactions


Copyright © 2004
Glenn L. Ray or Craig Larman

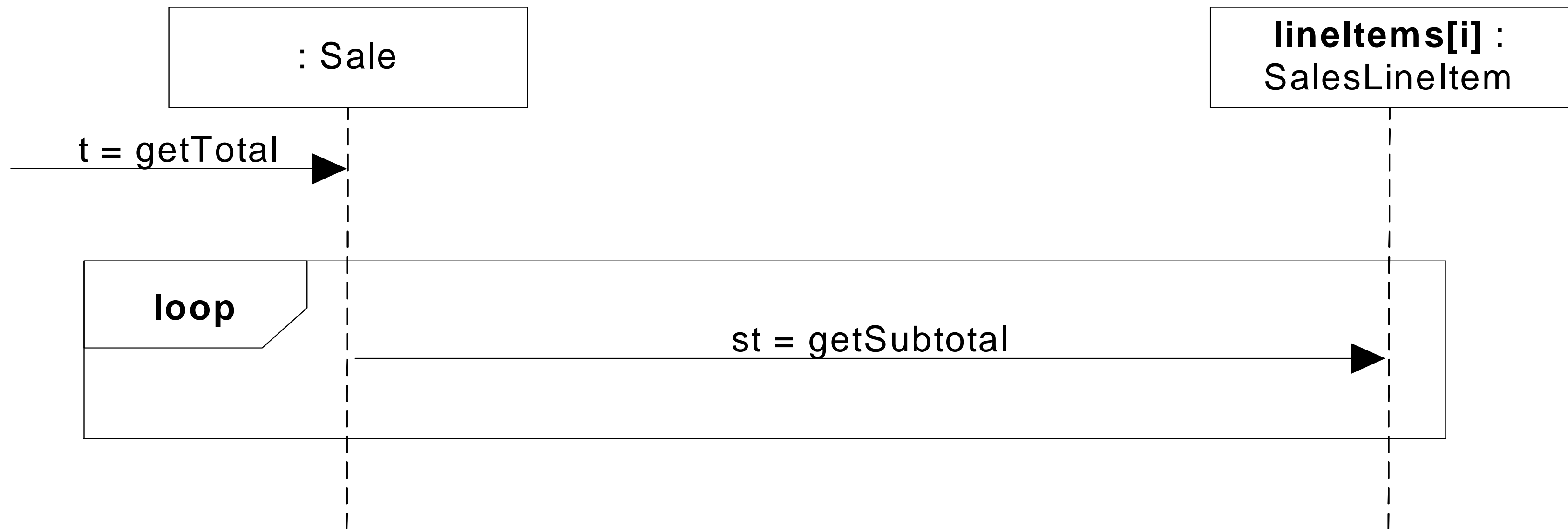
# Fig. 15.16




Technique for looping over a collection
Loop details are explicit; diagram more cluttered
Note Java code on p. 234 showing new for loop syntax


Copyright © 2004
Glenn L. Ray or Craig Larman

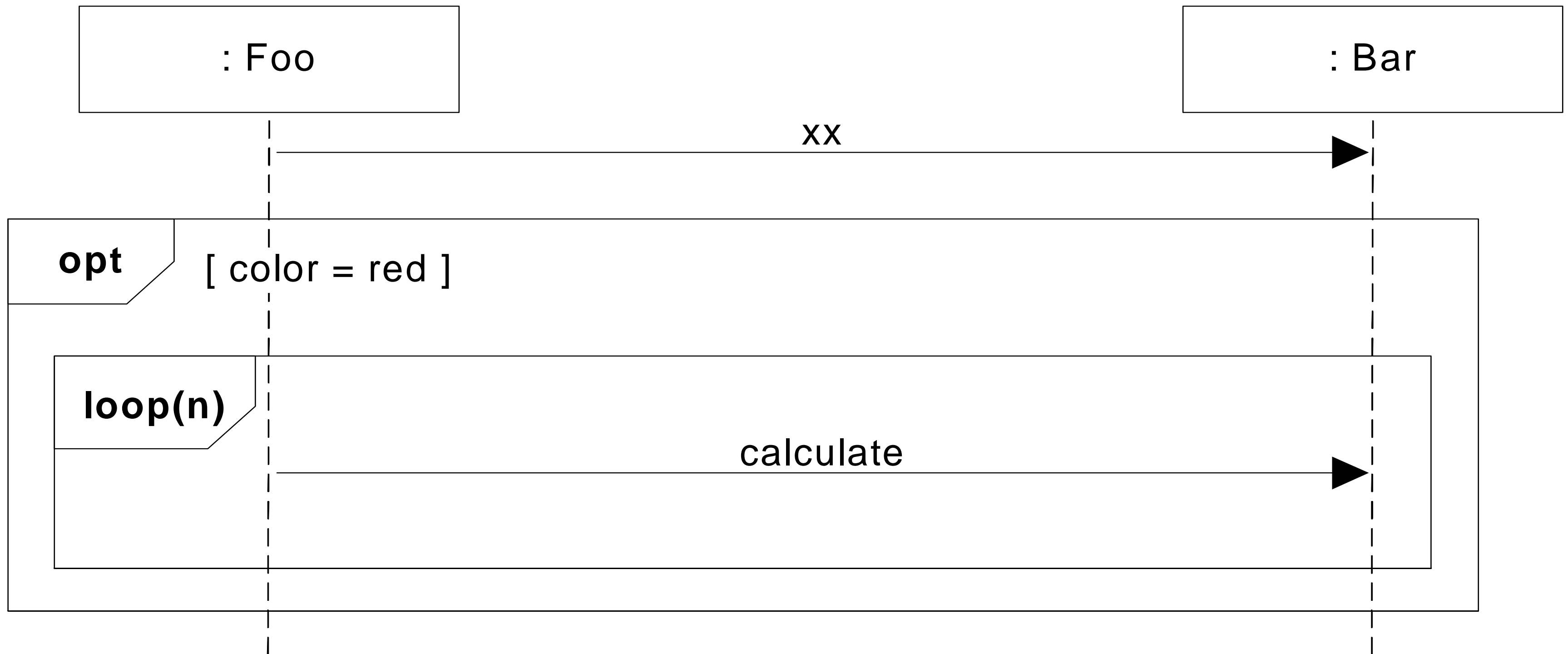
# Fig. 15.17




A 2nd technique for looping
Loop details are implicit; diagram less cluttered


Copyright © 2004
Glenn L. Ray or Craig Larman

# Fig. 15.18

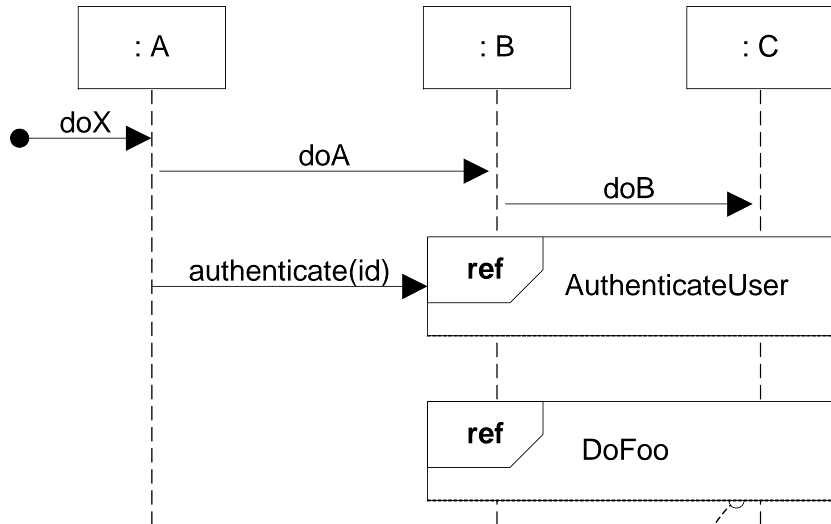


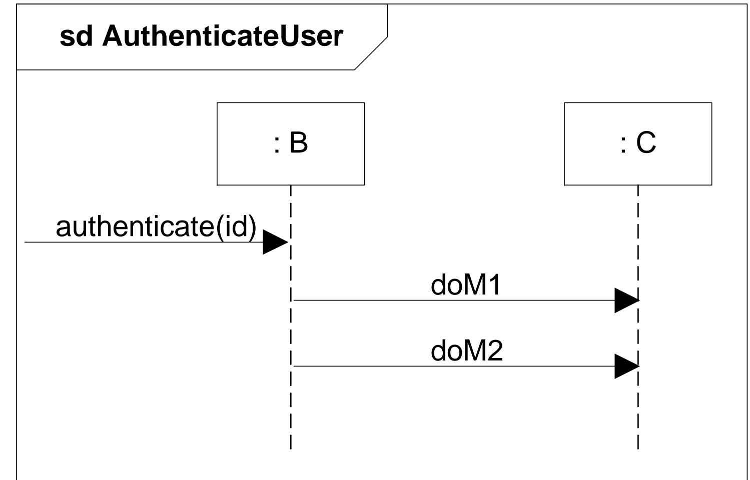

A loop frame nested within an optional frame


Copyright © 2004
Glenn L. Ray or Craig Larman

# Fig. 15.19

SD can contain frames that Reference other SDs

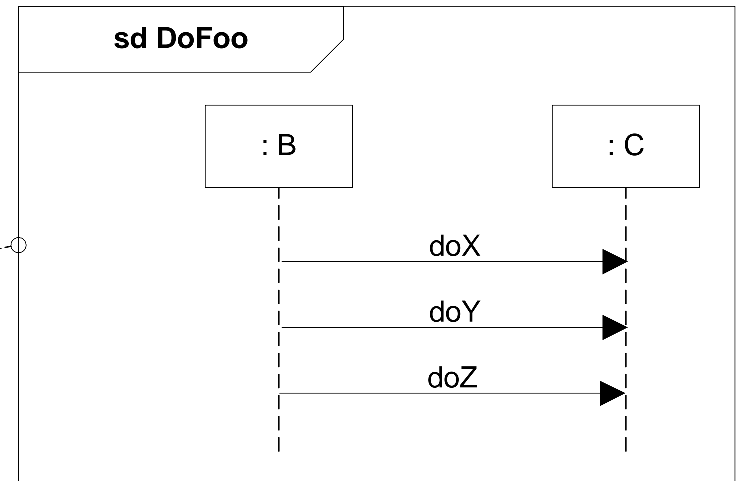



interaction occurrence

note it covers a set of lifelines

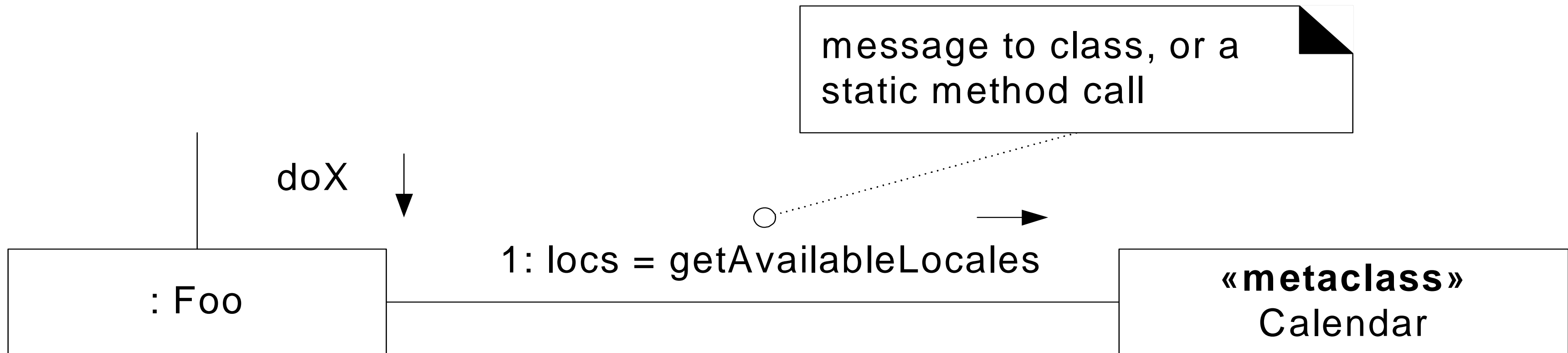
note that the sd frame it relates to has the same lifelines: B and C





Copyright © 2004
Glenn L. Ray or Craig Larman

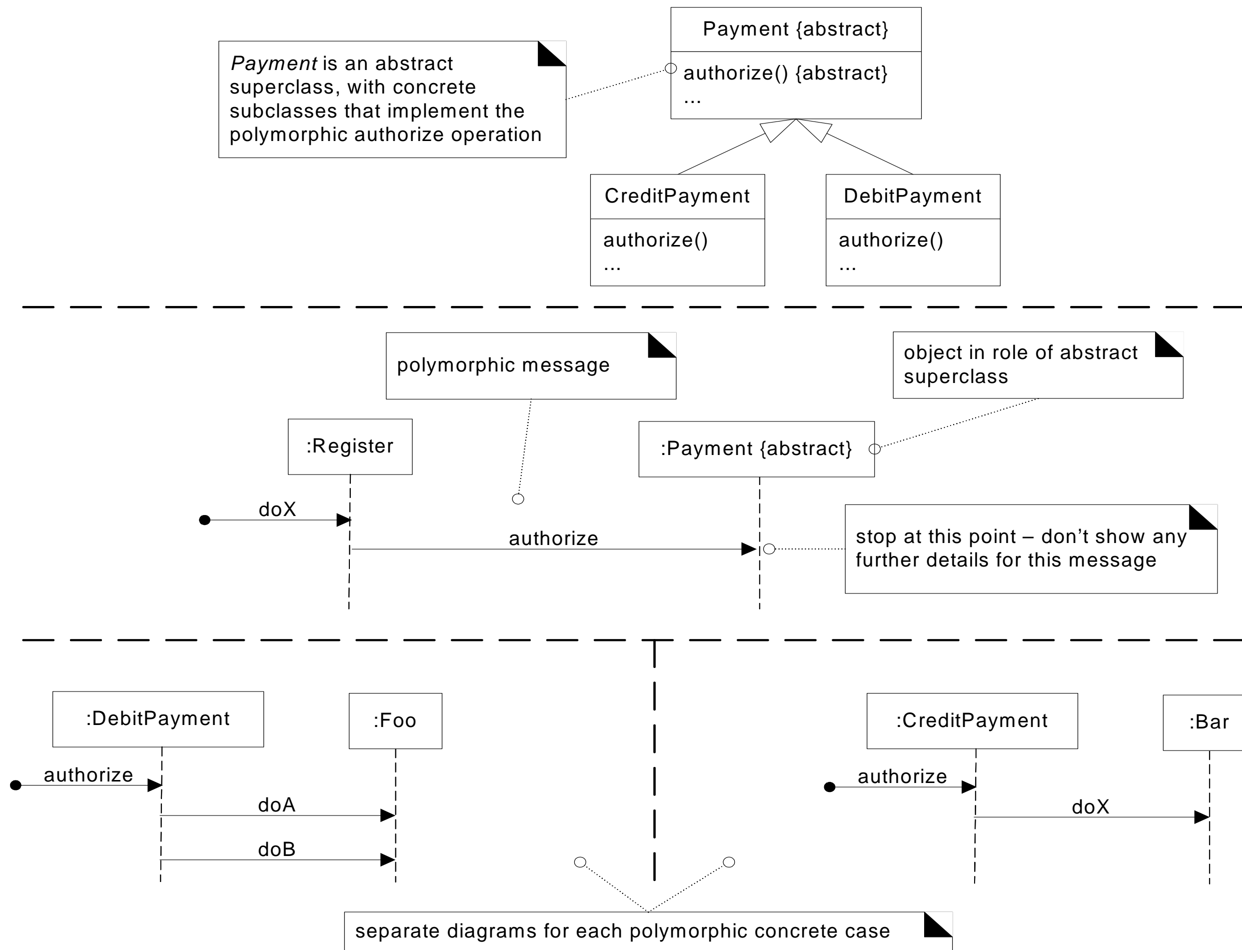
# Fig. 15.20




Call to a static class method
Notice there is no implied instance to the Calendar class (':' is omitted)


Copyright © 2004
Glenn L. Ray or Craig Larman

# Fig. 15.21




Polymorphic method calls


Copyright © 2004
Glenn L. Ray or Craig Larman

# Fig. 15.22

Active objects run in their own thread

a stick arrow in UML implies an asynchronous call

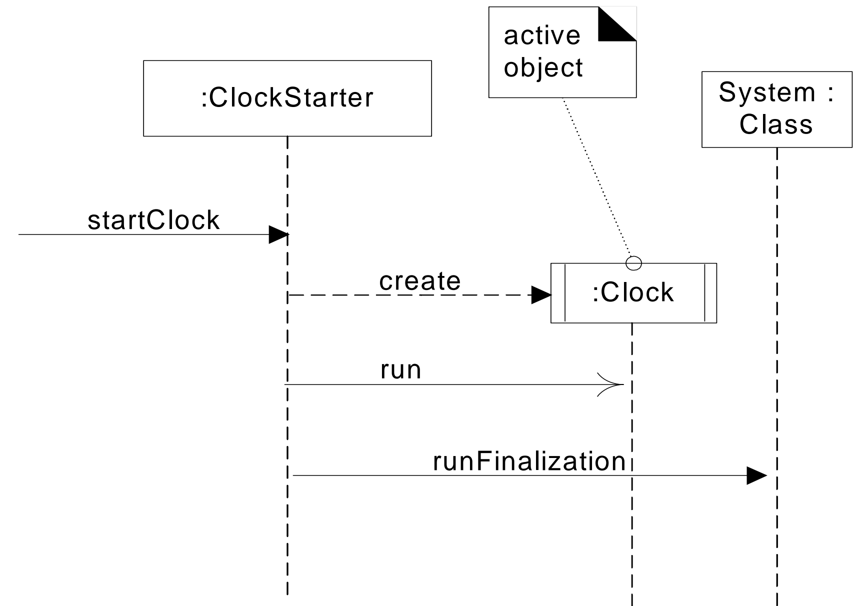
a filled arrow is the more common synchronous call

In Java, for example, an asynchronous call may occur as follows:

```
// Clock implements the Runnable interface
Thread t = new Thread( new Clock() );
t.start();
```

the asynchronous *start* call always invokes the *run* method on the *Runnable* (*Clock*) object

to simplify the UML diagram, the *Thread* object and the *start* message may be avoided (they are standard "overhead"); instead, the essential detail of the *Clock* creation and the *run* message imply the asynchronous call



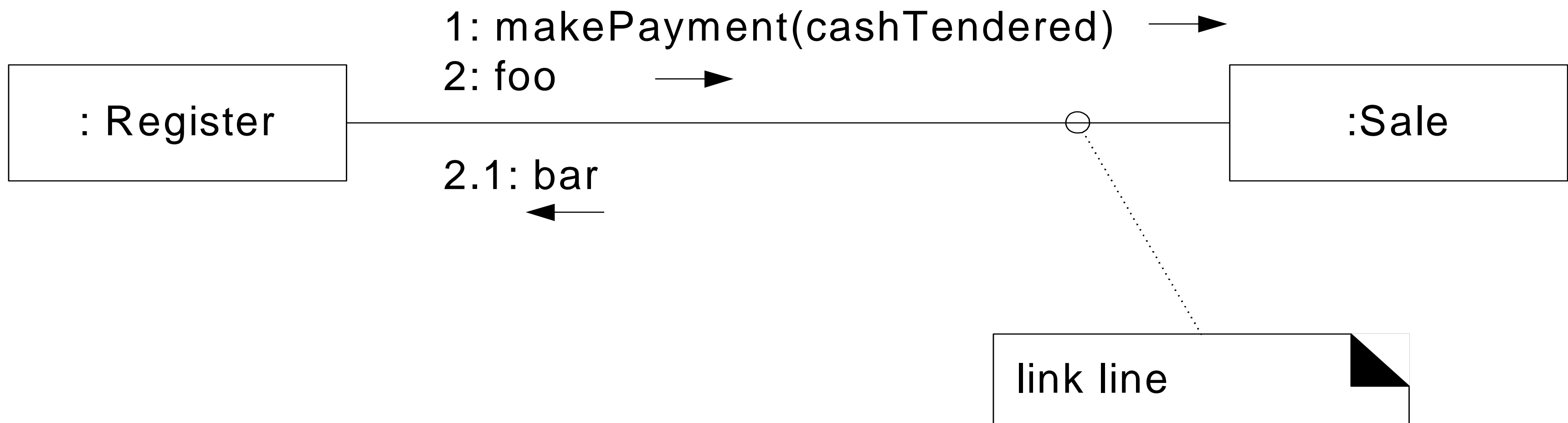

Solid vs. stick arrowheads easily confused when sketching models

See Java code p. 239-240


Copyright © 2004
Glenn L. Ray or Craig Larman

# Fig. 15.23

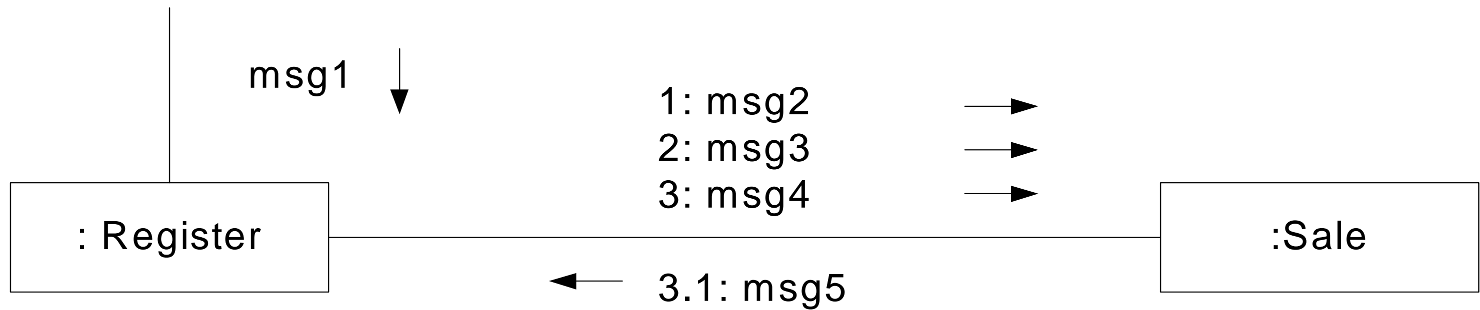
**UML for Communication Diagrams…..**




A link is to objects as an association is to classes
A link is an instance of an association


Copyright © 2004
Glenn L. Ray or Craig Larman

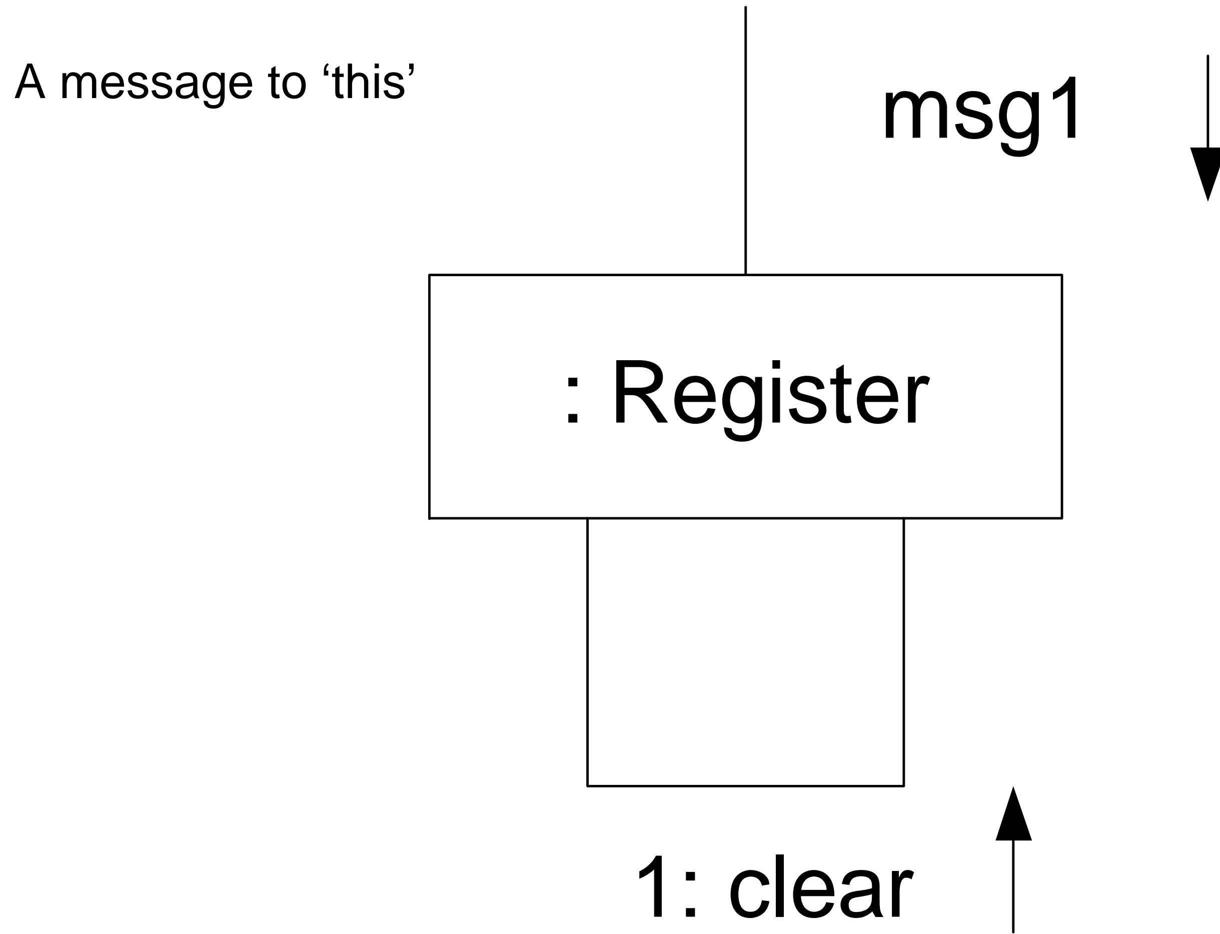
# Fig. 15.24




Link does not show directionality…that is indicated by each msg


Copyright © 2004
Glenn L. Ray or Craig Larman

# Fig. 15.25





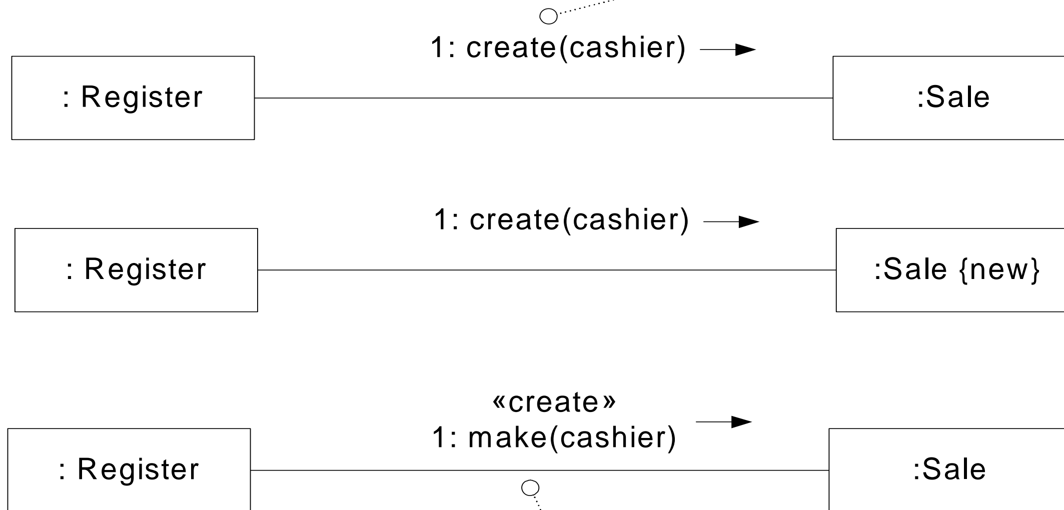

Copyright © 2004
Glenn L. Ray or Craig Larman

# Fig. 15.26

Three ways to show creation in a communication diagram

create message, with optional initializing parameters. This will normally be interpreted as a constructor call.

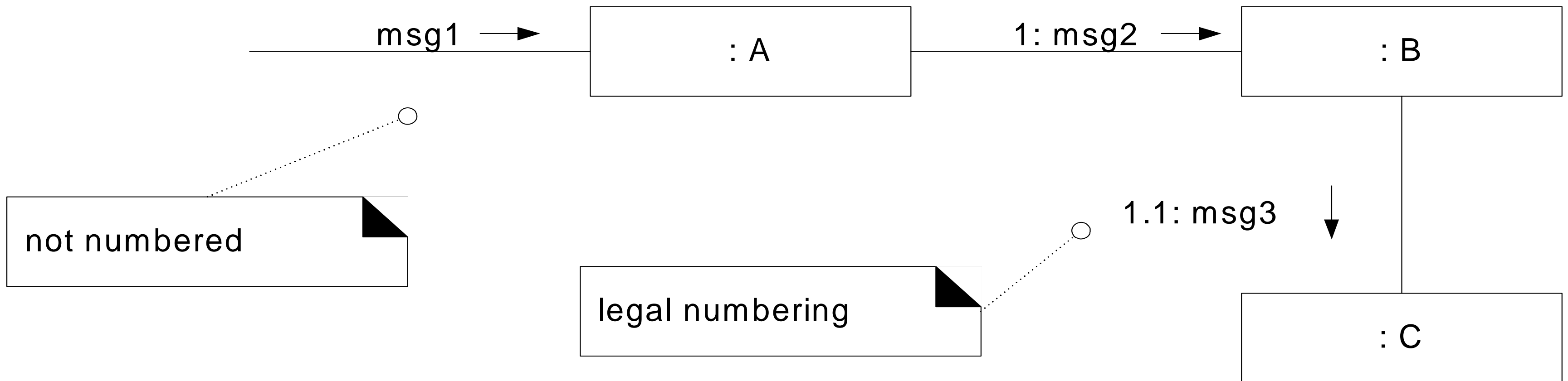
Simplest & most common




if an unobvious creation message name is used, the message may be stereotyped for clarity


Copyright © 2004
Glenn L. Ray or Craig Larman

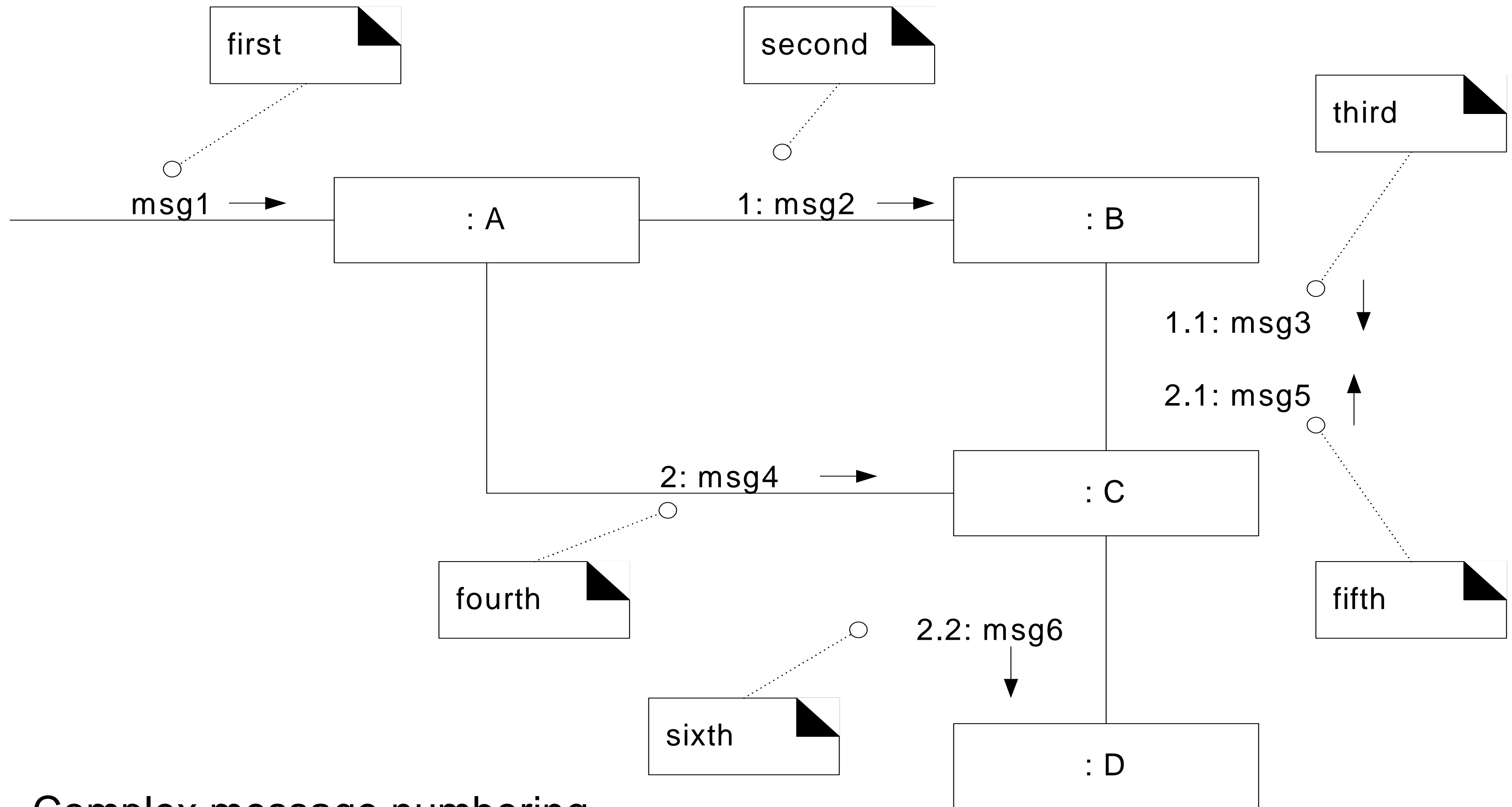
# Fig. 15.27




Message numbering…can be tricky to follow


Copyright © 2004
Glenn L. Ray or Craig Larman

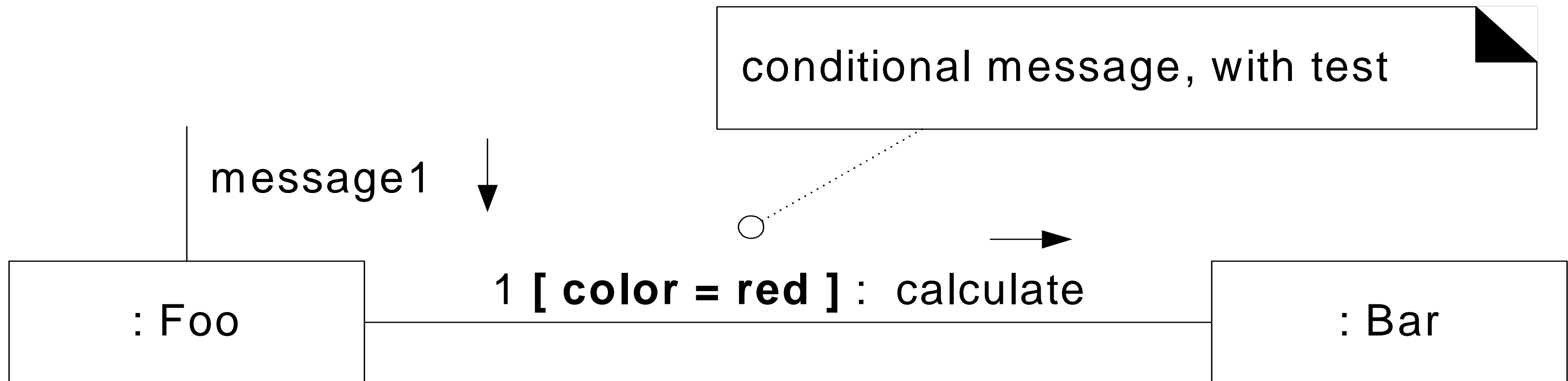
# Fig. 15.28




Complex message numbering


Copyright © 2004
Glenn L. Ray or Craig Larman

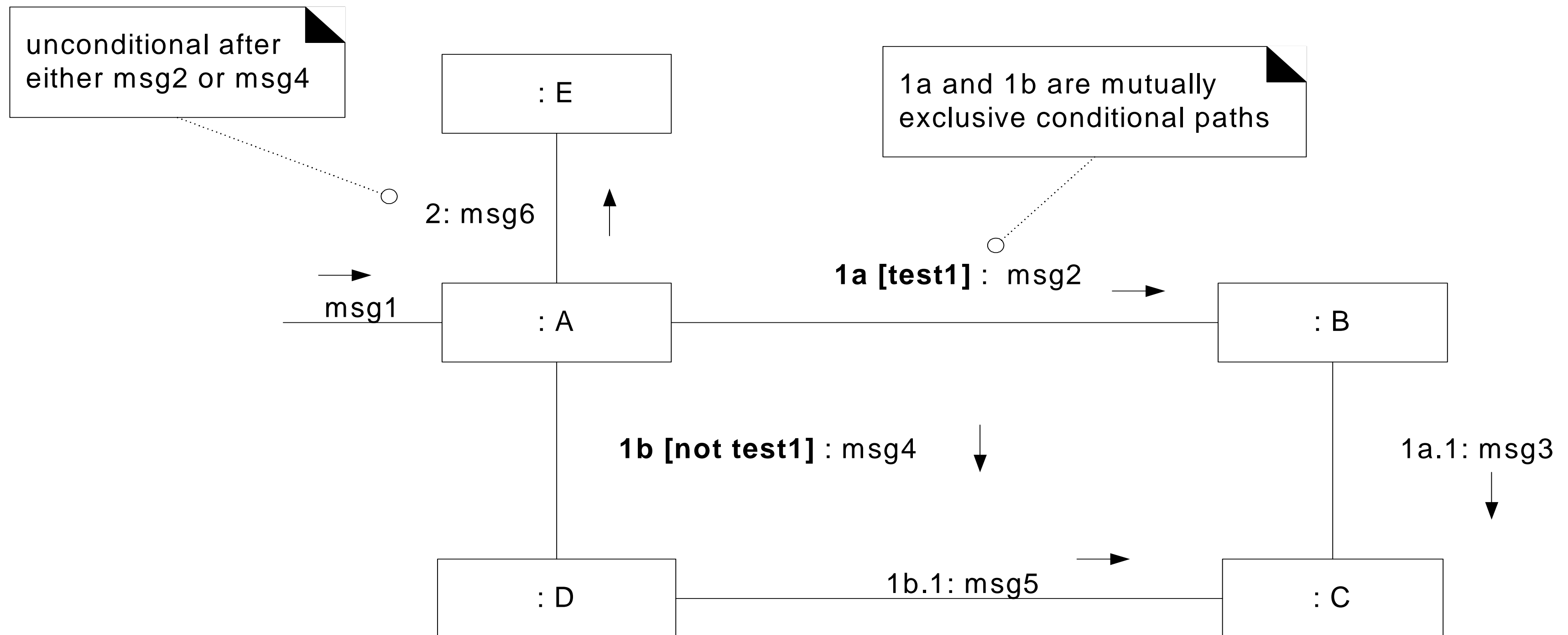
# Fig. 15.29




Conditional message with guard


Copyright © 2004
Glenn L. Ray or Craig Larman

# Fig. 15.30

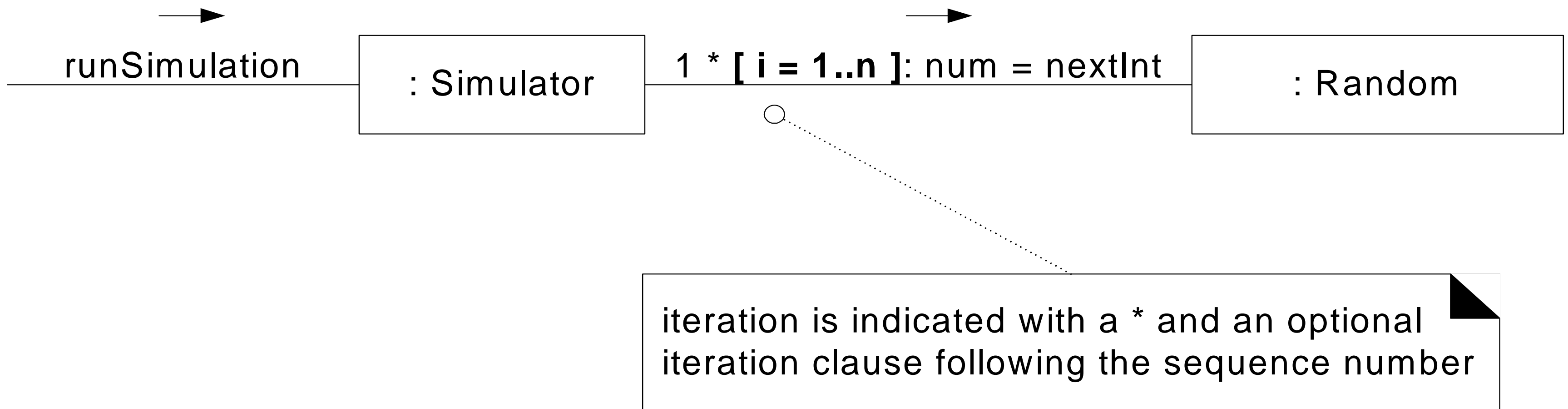



Mutually exclusive conditional messages



Copyright © 2004
Glenn L. Ray or Craig Larman

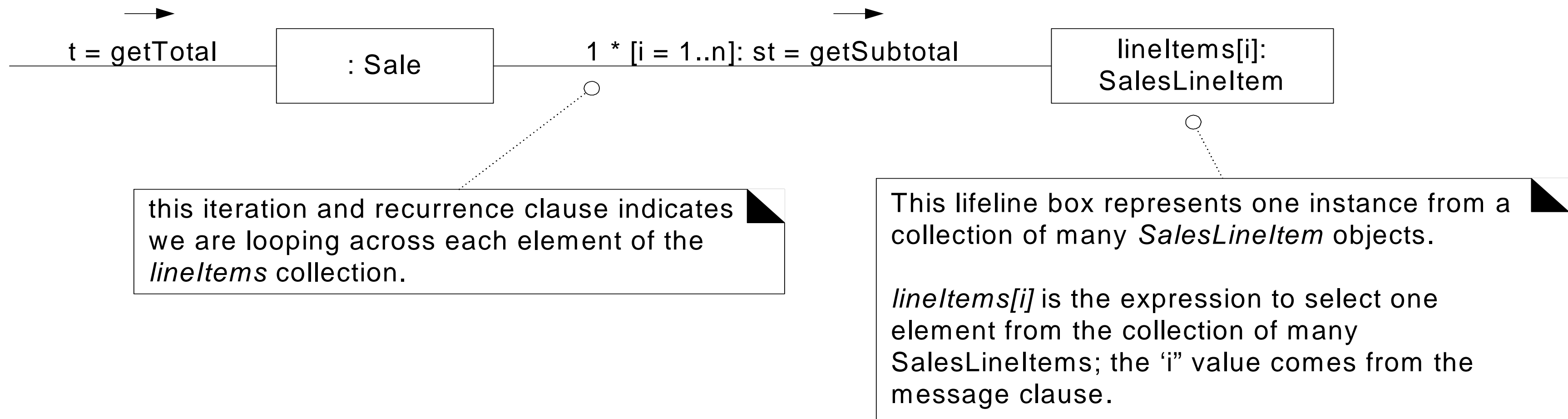
# Fig. 15.31

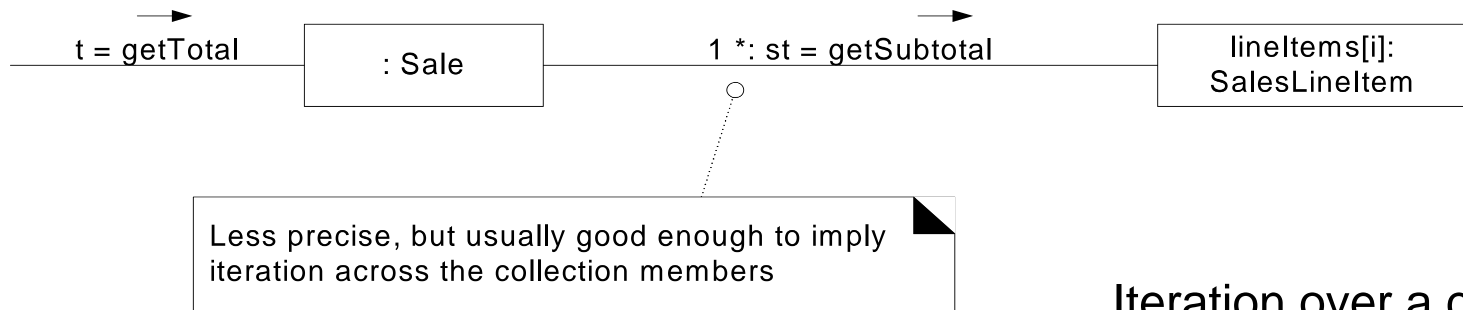



Iteration/looping


Copyright © 2004
Glenn L. Ray or Craig Larman

# Fig. 15.32



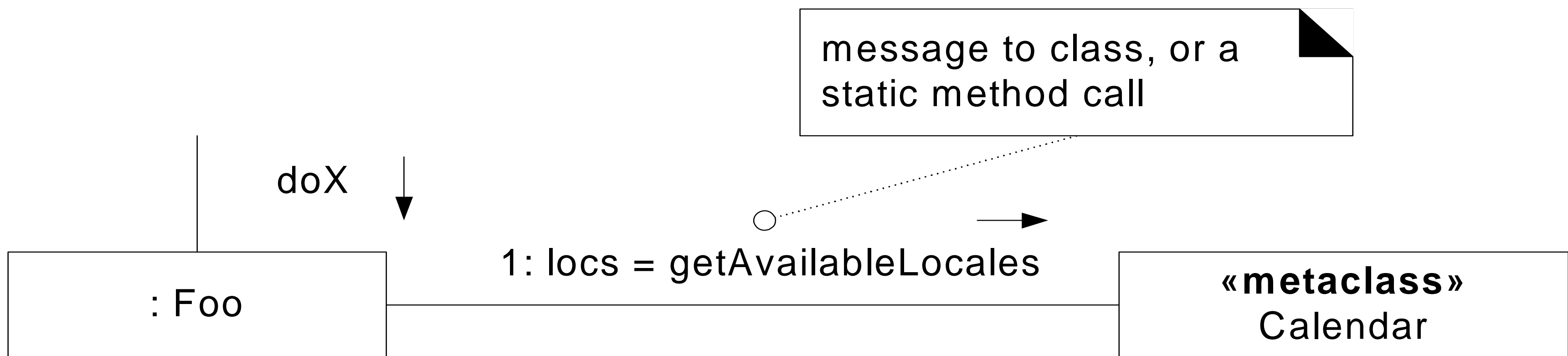




Iteration over a collection



Copyright © 2004
Glenn L. Ray or Craig Larman

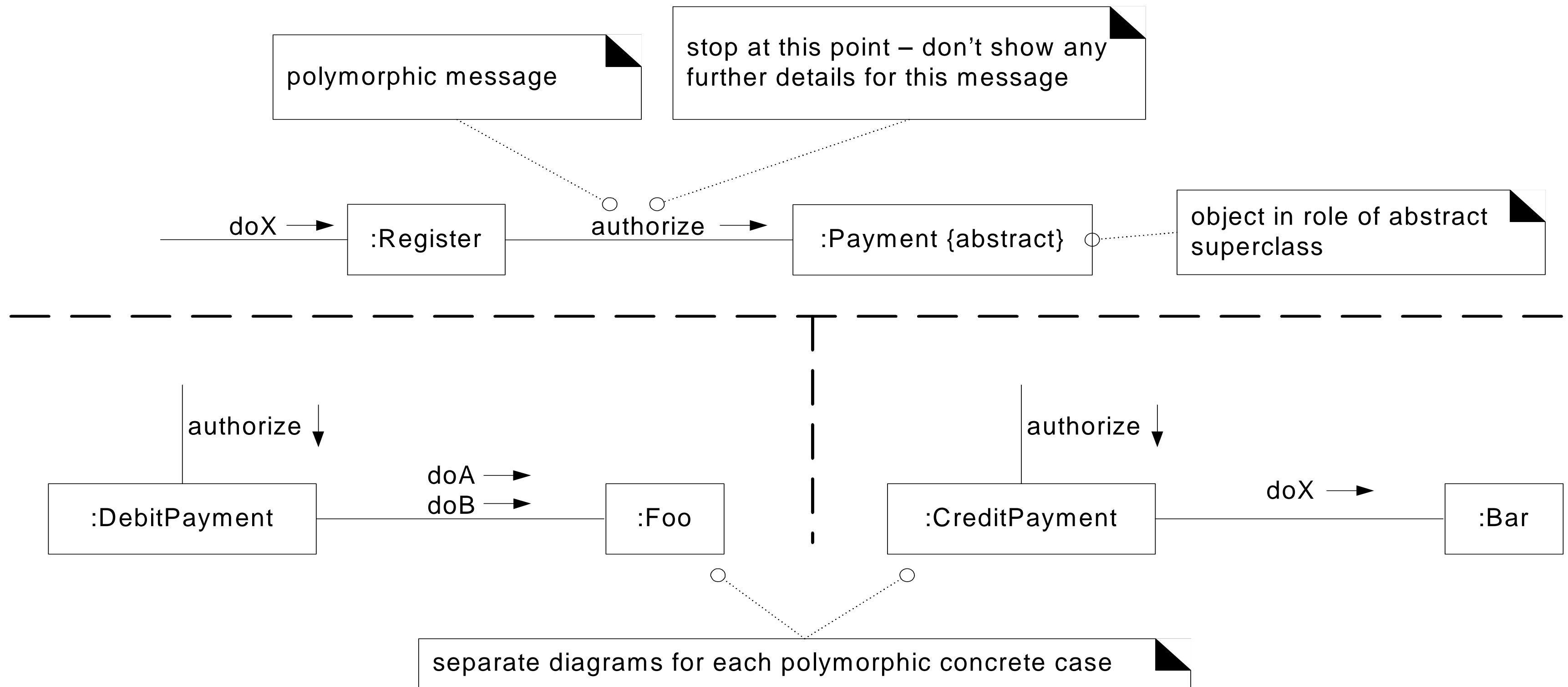
# Fig. 15.33




Static method call or message to a class


Copyright © 2004
Glenn L. Ray or Craig Larman

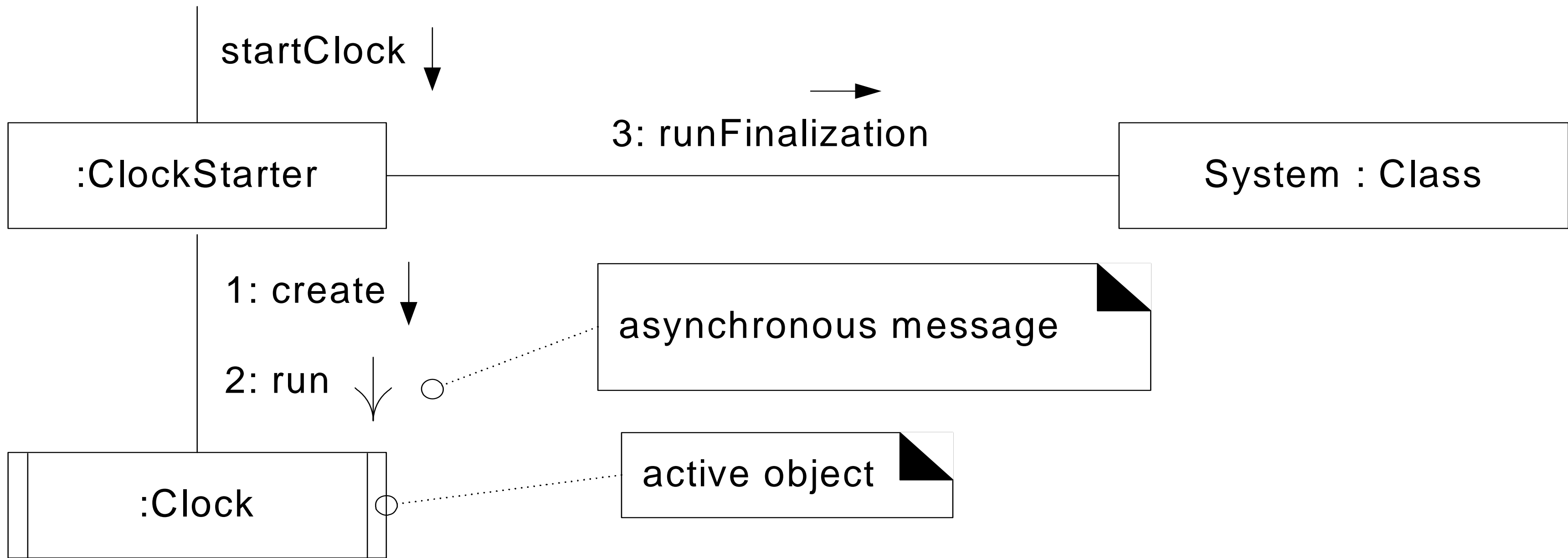
# Fig. 15.34




Polymorphic messages


Copyright © 2004
Glenn L. Ray or Craig Larman

# Fig. 15.35




Asychronous calls


Copyright © 2004
Glenn L. Ray or Craig Larman