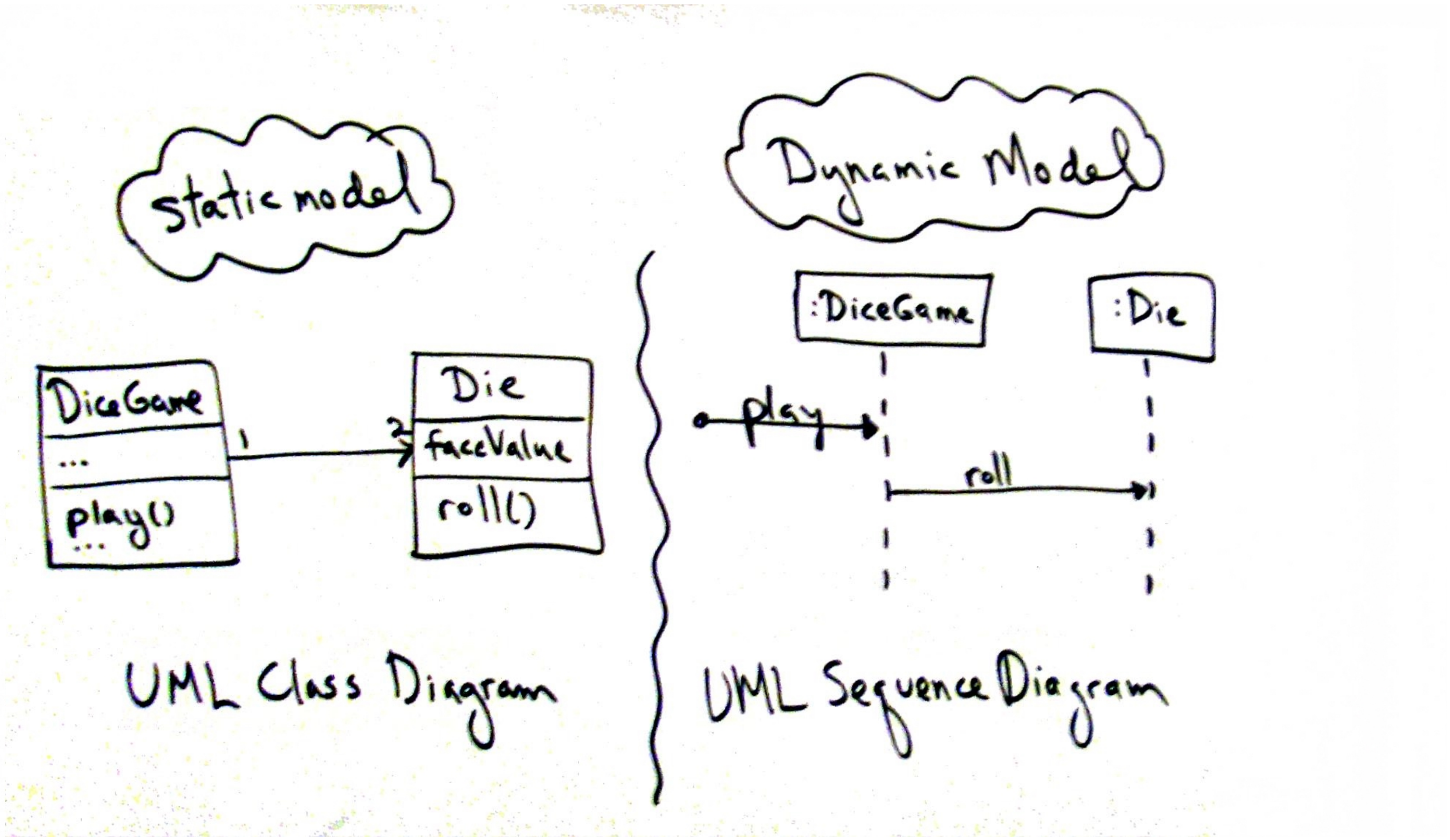
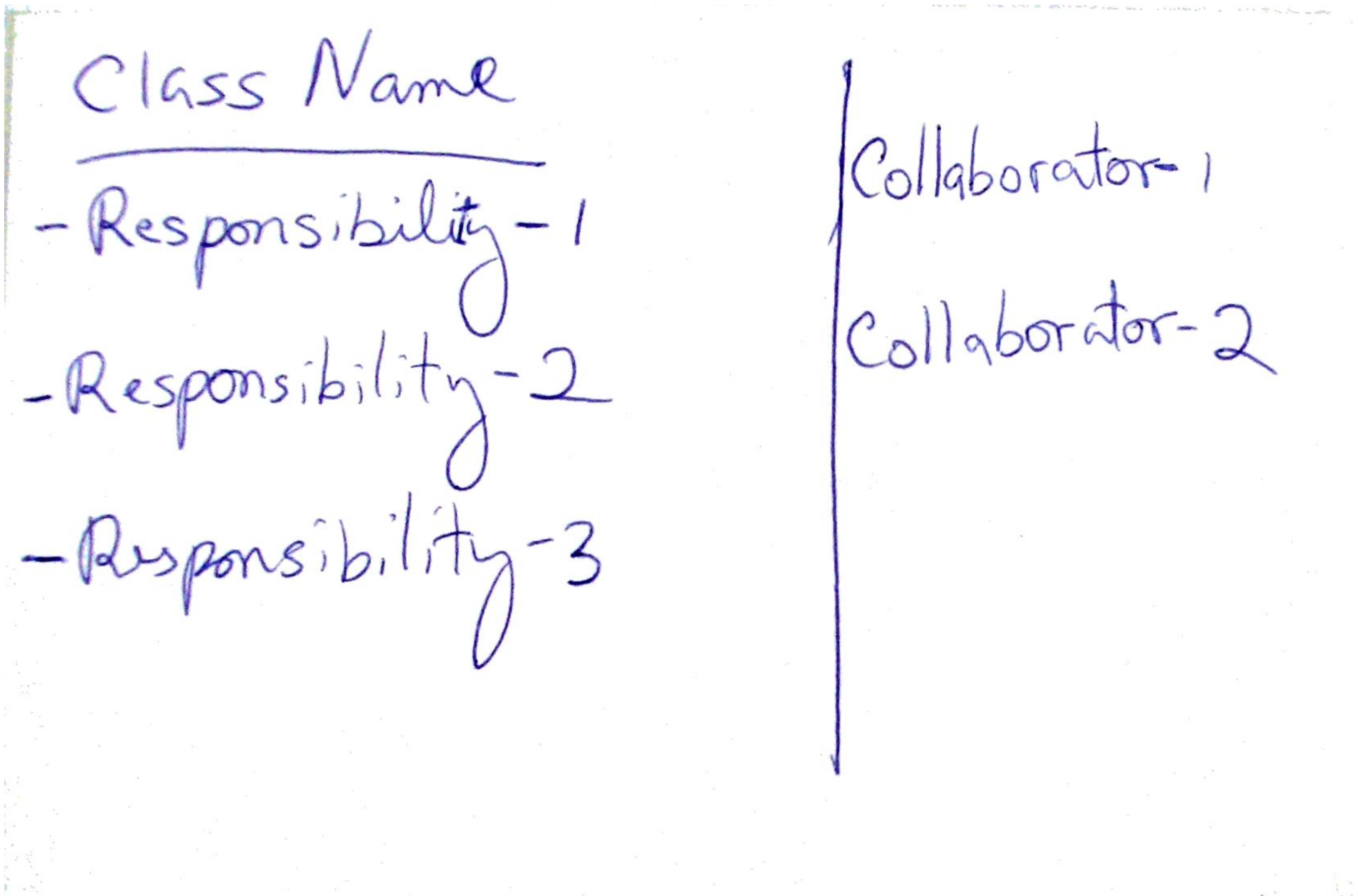
Chapter 14

Starting Object Design

Spend time on dynamic models
They are the key tool for building good static models



One Object Design Technique: Class Responsibility Collaboration (CRC) cards



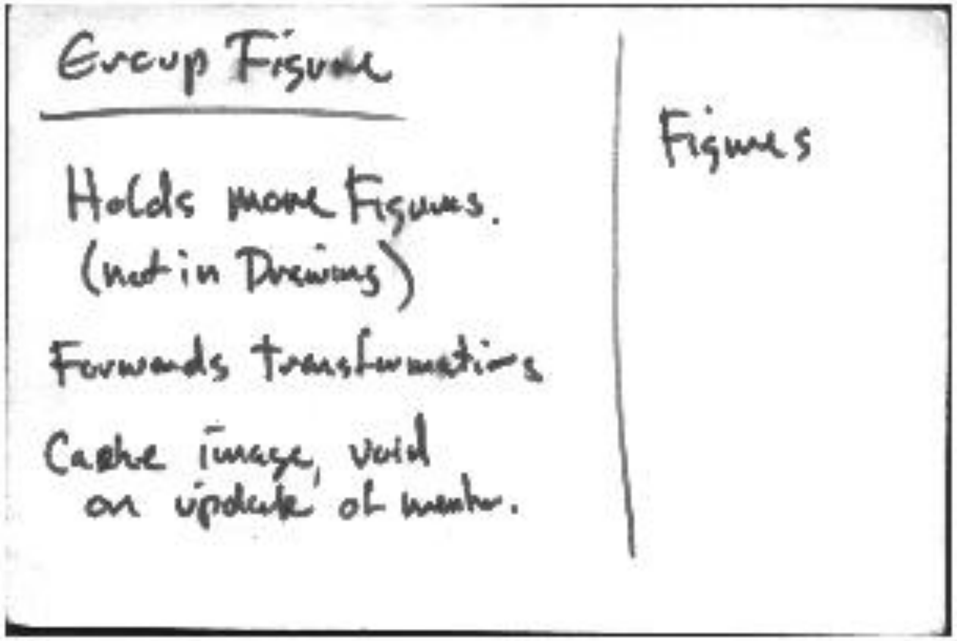
Class Name

- Responsibility-1
- Responsibility-2
- Responsibility-3

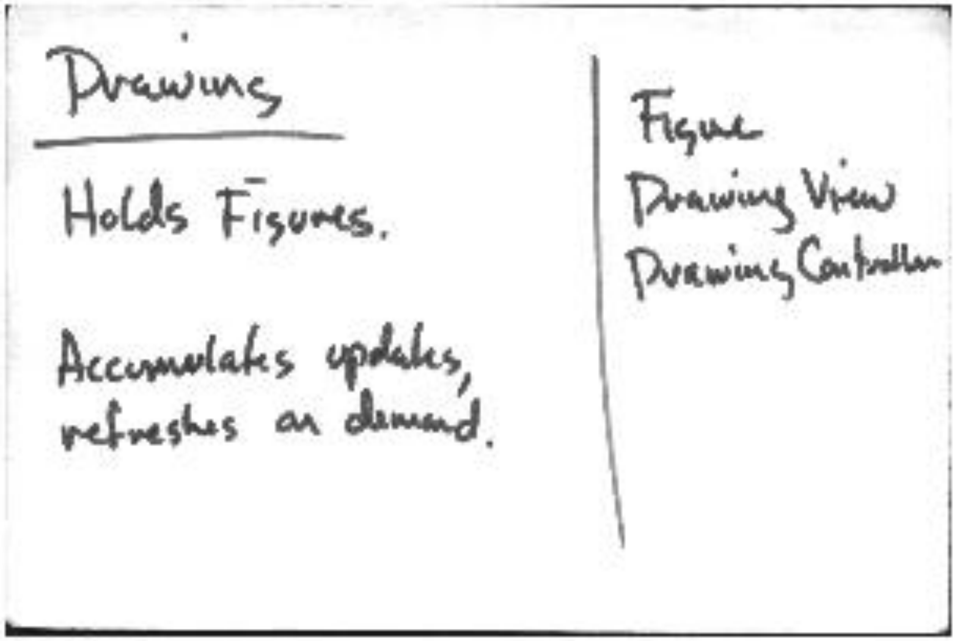
Collaborator-1

Collaborator-2

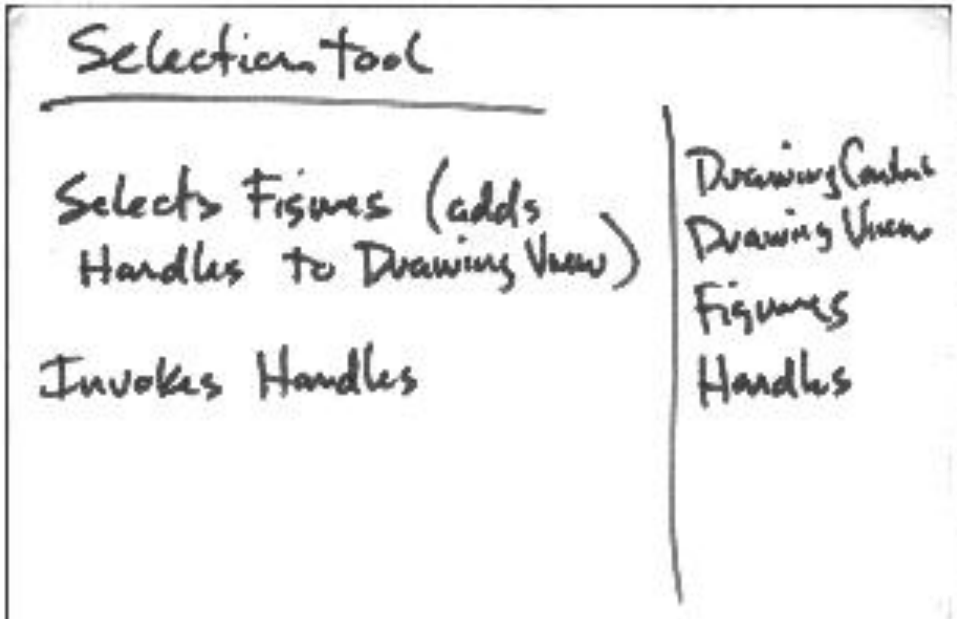
CRC Card examples



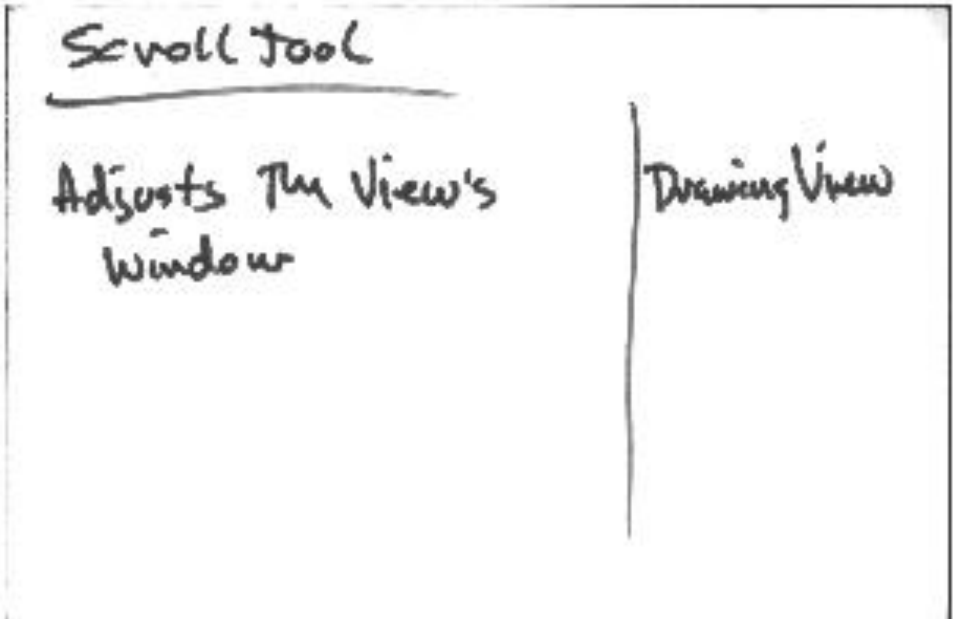
Group Figure	
Holds more Figures. (not in Drawing)	Figures
Forwards transformations	
Cache image, void on update of member.	



Drawing	
Holds Figures.	Figure
	Drawing View
	Drawing Controller
Accumulates updates, refreshes on demand.	



Selection tool	
Selects Figures (adds Handles to Drawing View)	Drawing Controller
	Drawing View
	Figures
Invokes Handles	Handles



Scroll Tool	
Adjusts the View's window	Drawing View

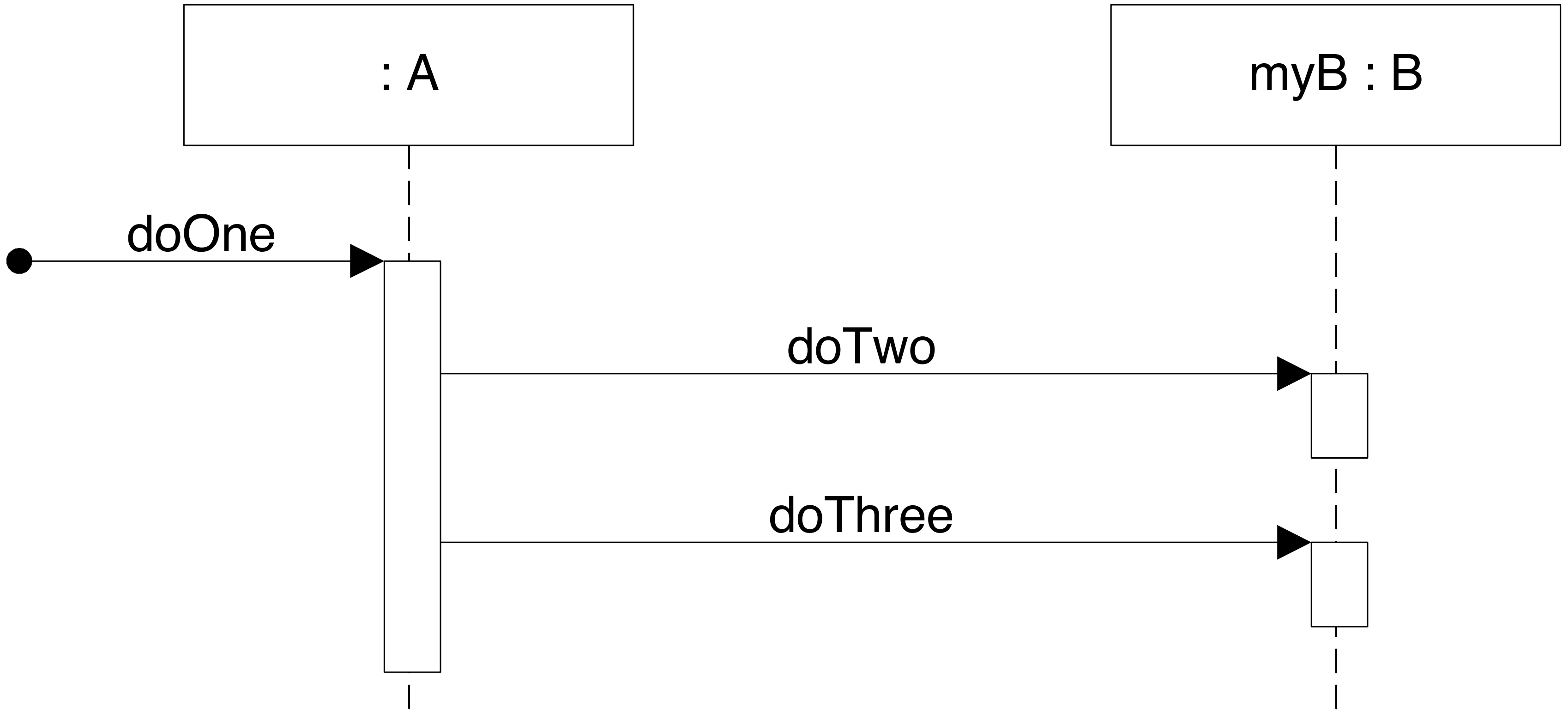
Chapter 15

Dynamic modeling of object:
Interaction Diagrams

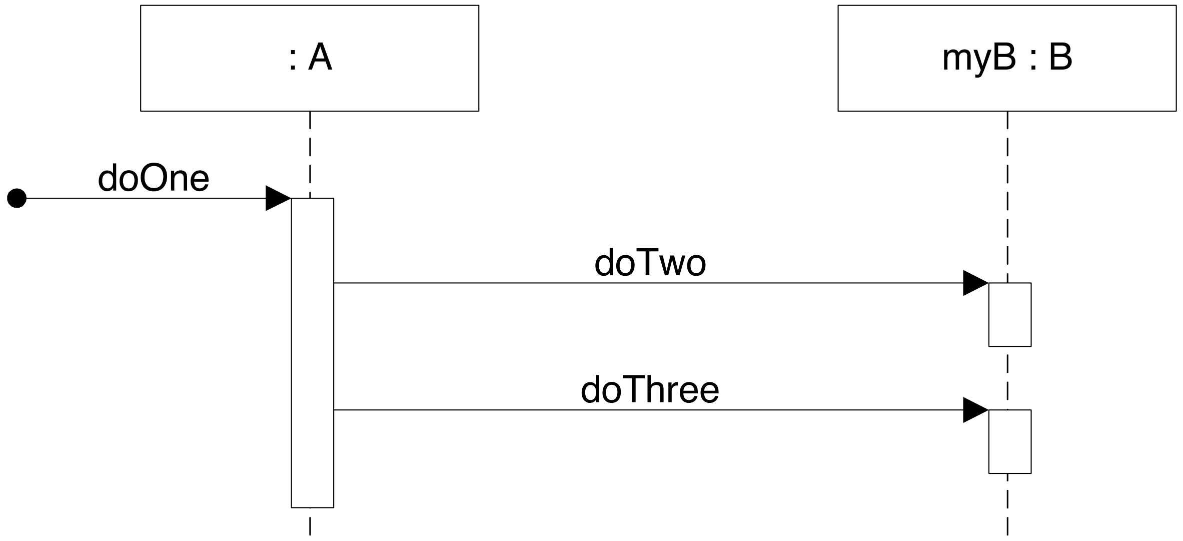
UML Interaction Diagrams

- Two variants
  - Sequence diagrams
  - Communication diagrams

Example: Sequence Diagram



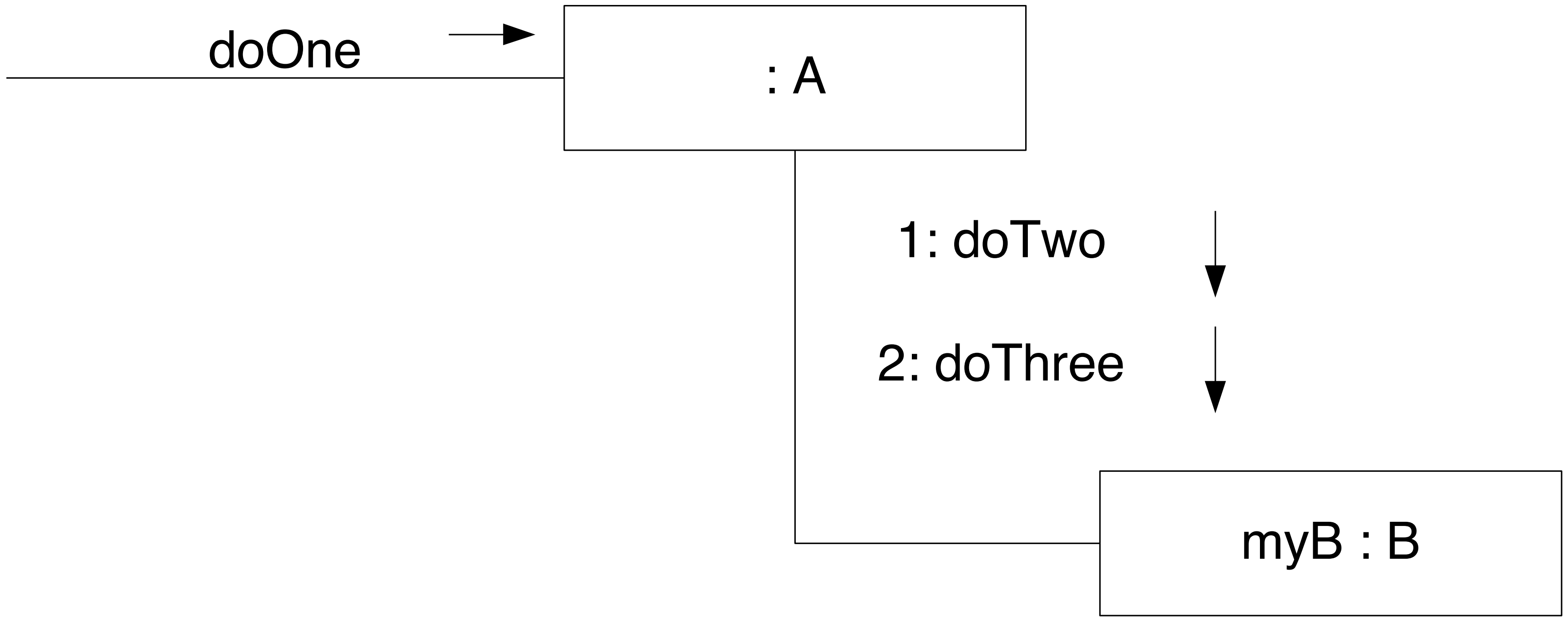
What does this represent in code?



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

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

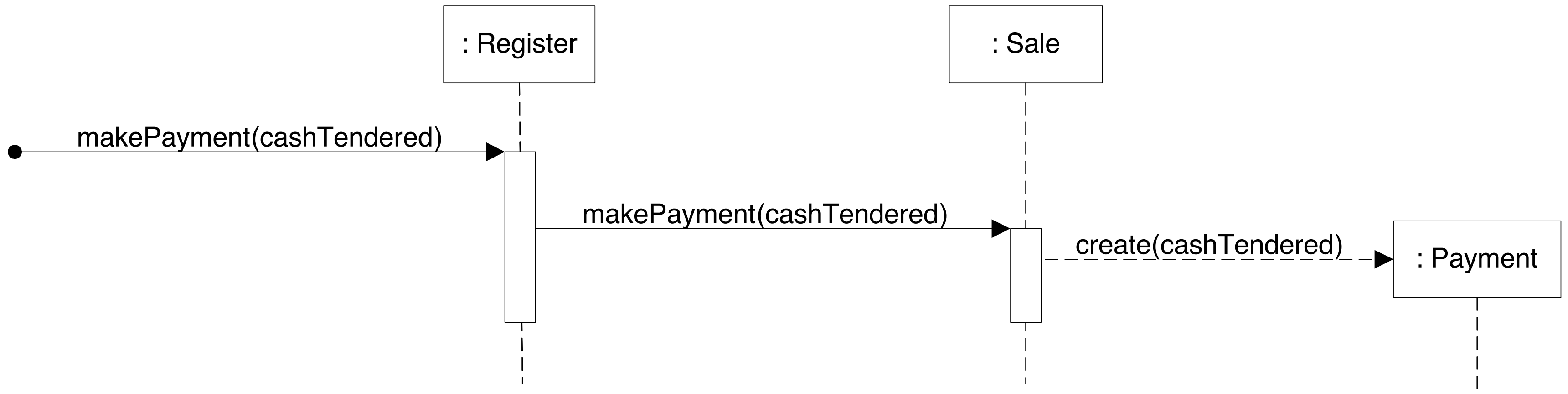
Example: Collaboration Diagram



Strengths and Weaknesses of Sequence and Collaboration Diagrams

Type	Strengths	Weaknesses
sequence	clearly shows sequence or time ordering of messages simple notation	forced to extend to the right when adding new objects; consumes horizontal space
collaboration	space economical—flexibility to add new objects in two dimensions better to illustrate complex branching, iteration, and concurrent behavior	difficult to see sequence of messages more complex notation

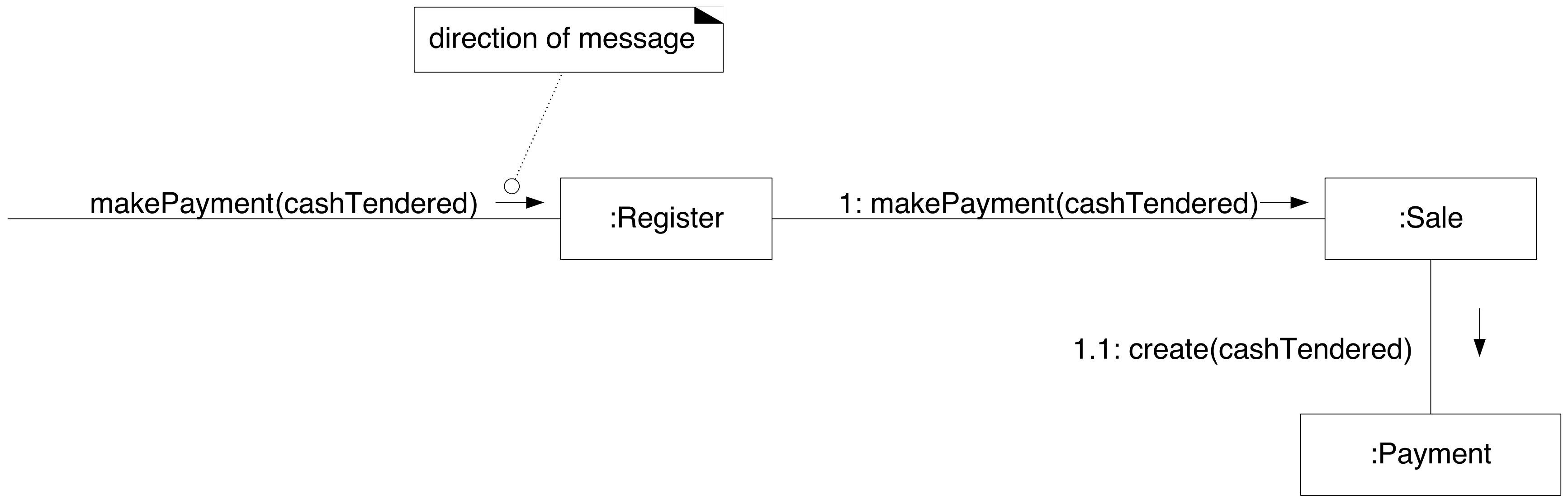
Example Sequence Diagram



```
public class Sale {
  private Payment payment;

  public void makePayment(Money cashTendered) {
      payment = new Payment(cashTendered);
      // ...
  }
  // ...
}
```

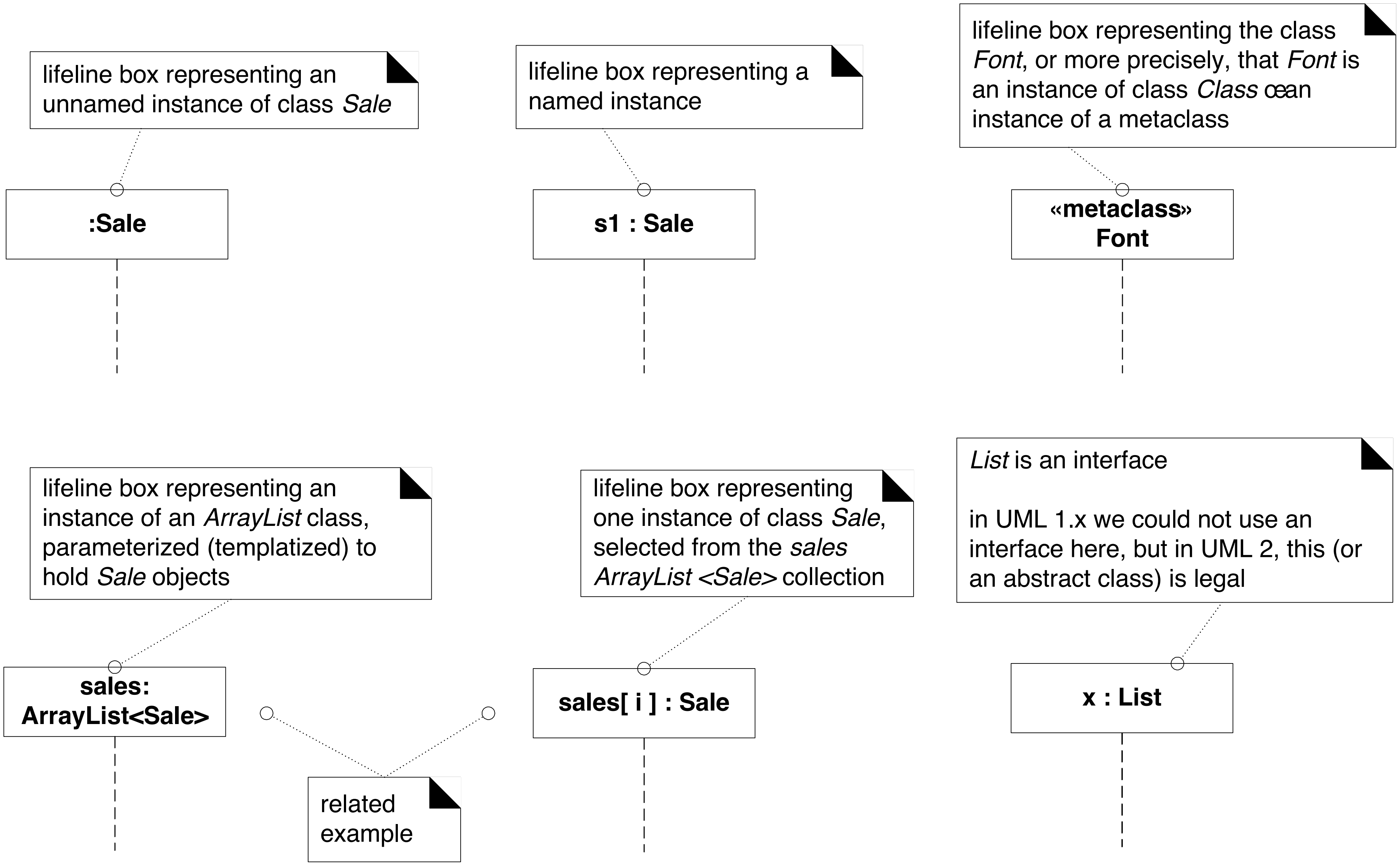
Example collaboration diagram


direction of message
makePayment(cashTendered)
:Register
1: makePayment(cashTendered)
:Sale
1.1: create(cashTendered)
:Payment

```
public class Sale {
  private Payment payment;

  public void makePayment(Money cashTendered) {
     payment = new Payment(cashTendered);
     // ...
  }
  // ...
}
```

Lifeline boxes



lifeline box representing an unnamed instance of class *Sale*

lifeline box representing a named instance

lifeline box representing the class *Font*, or more precisely, that *Font* is an instance of class *Class* œan instance of a metaclass

lifeline box representing an instance of an *ArrayList* class, parameterized (templatized) to hold *Sale* objects

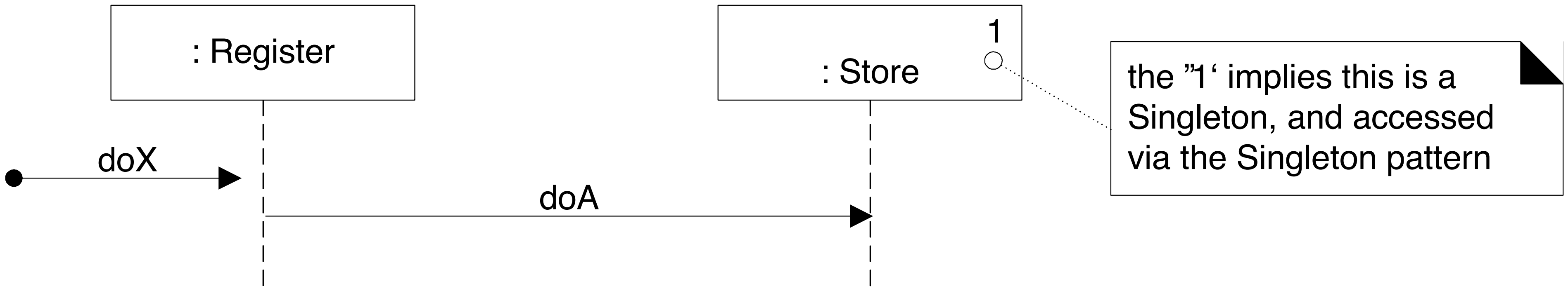
related example

lifeline box representing one instance of class *Sale*, selected from the *sales* *ArrayList <Sale>* collection

List is an interface

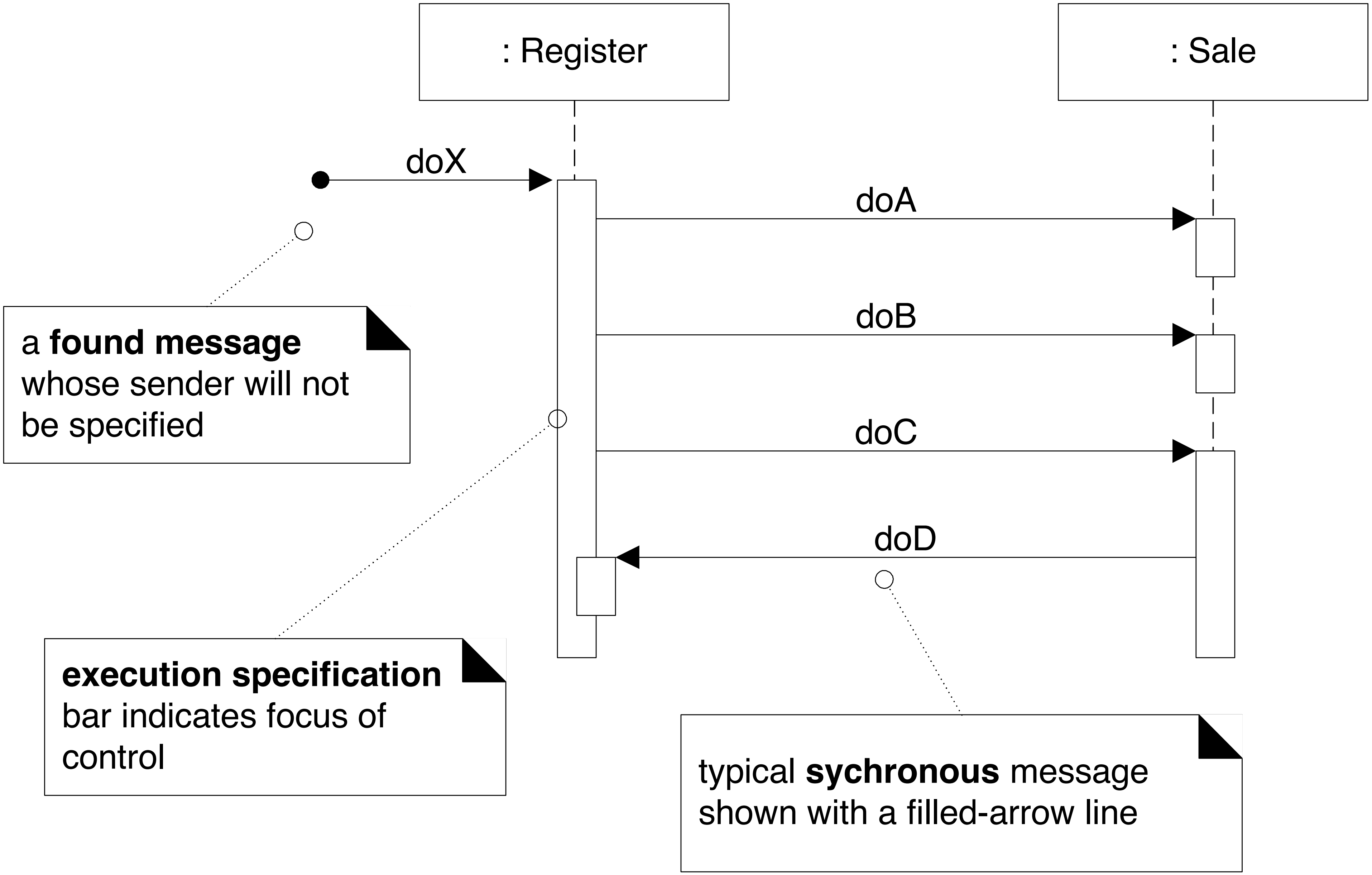
in UML 1.x we could not use an interface here, but in UML 2, this (or an abstract class) is legal

The Singleton Pattern


: Register
: Store
1
doX
doA
the ”1‘ implies this is a Singleton, and accessed via the Singleton pattern

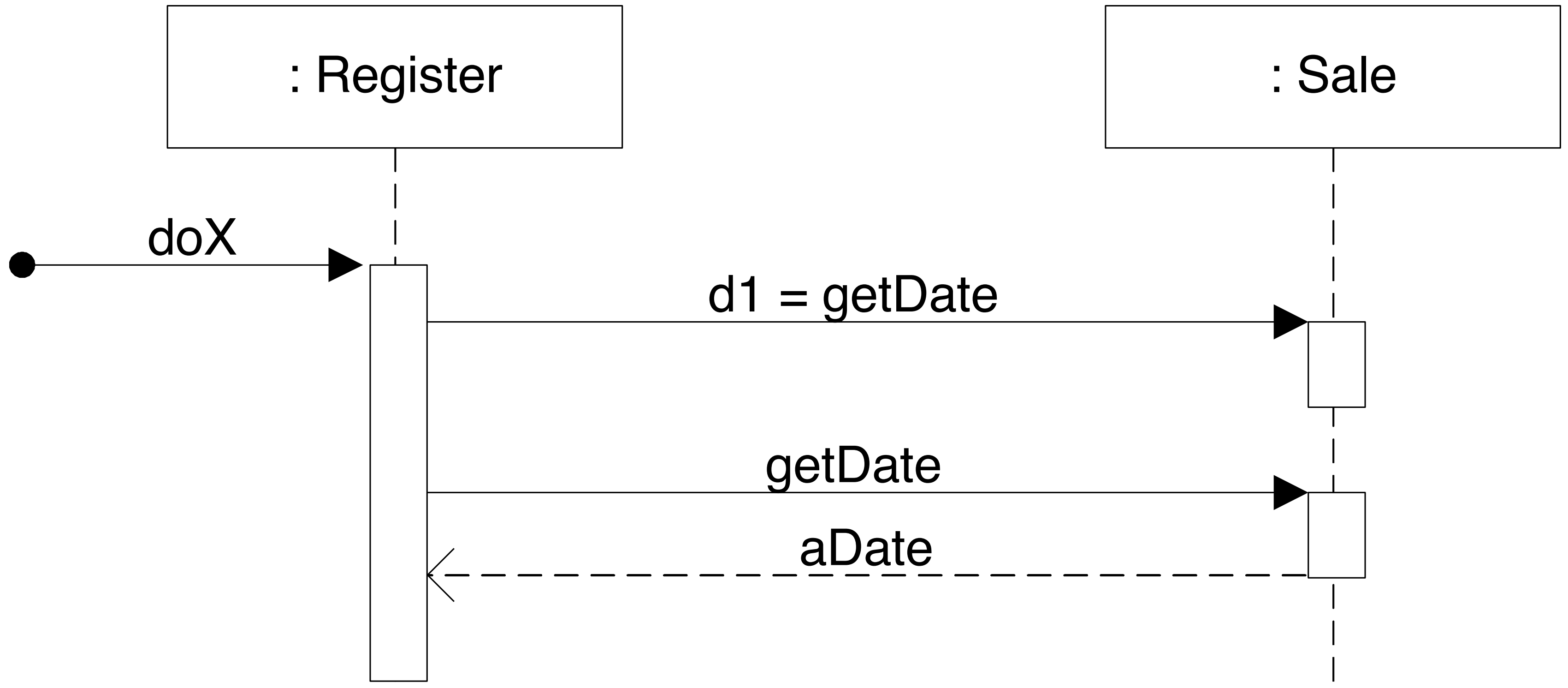
- Used if we want only ONE instance of a class instantiated
  - Examples: Database, log
- We’ll learn how to accomplish this in Java later

Lifelines, messages, “found” or “starting” messages



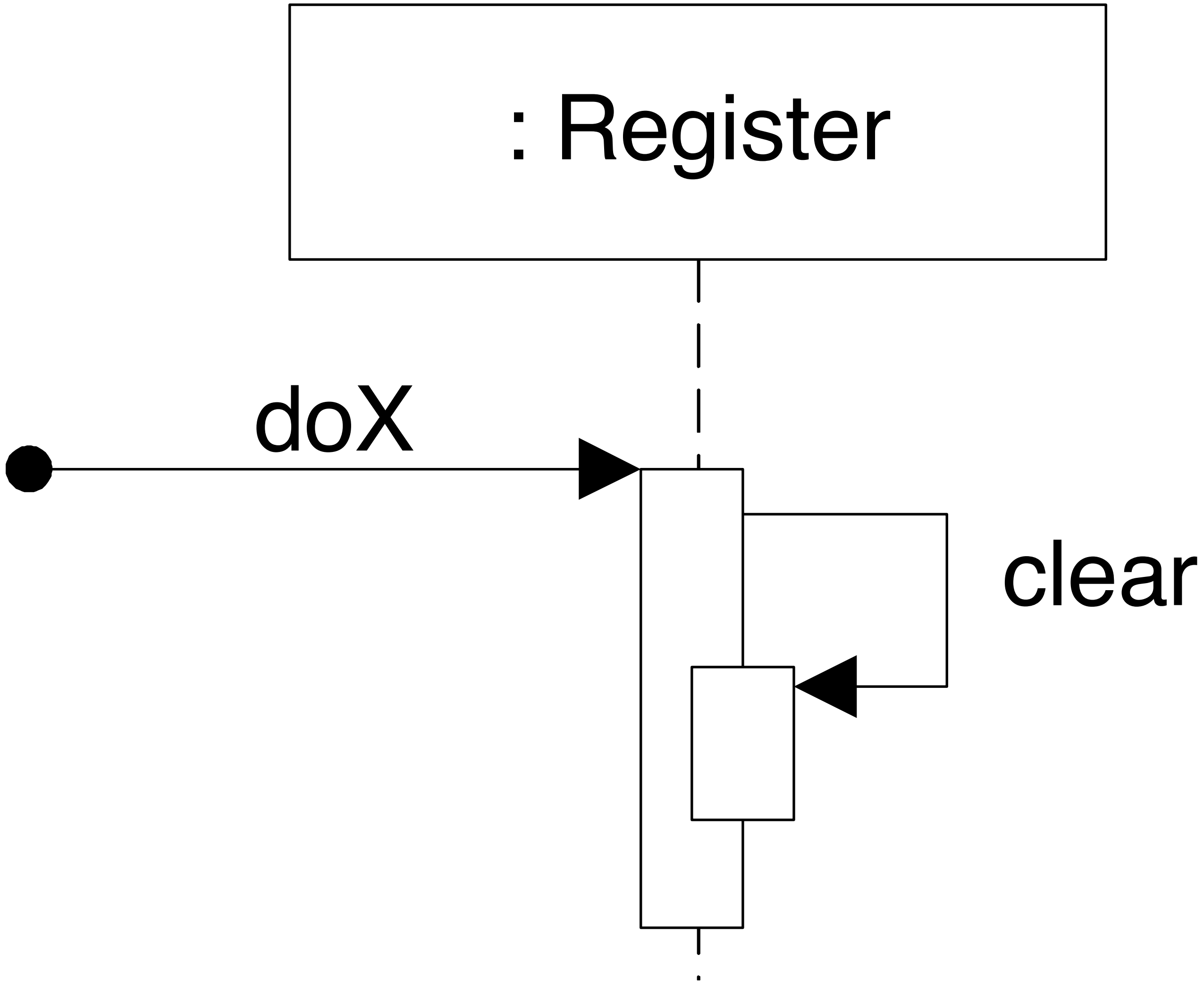
: Register
: Sale
doX
doA
doB
doC
doD
a **found message** whose sender will not be specified
execution specification bar indicates focus of control
typical **sychronous** message shown with a filled-arrow line

Illustrating Replies or Return Values

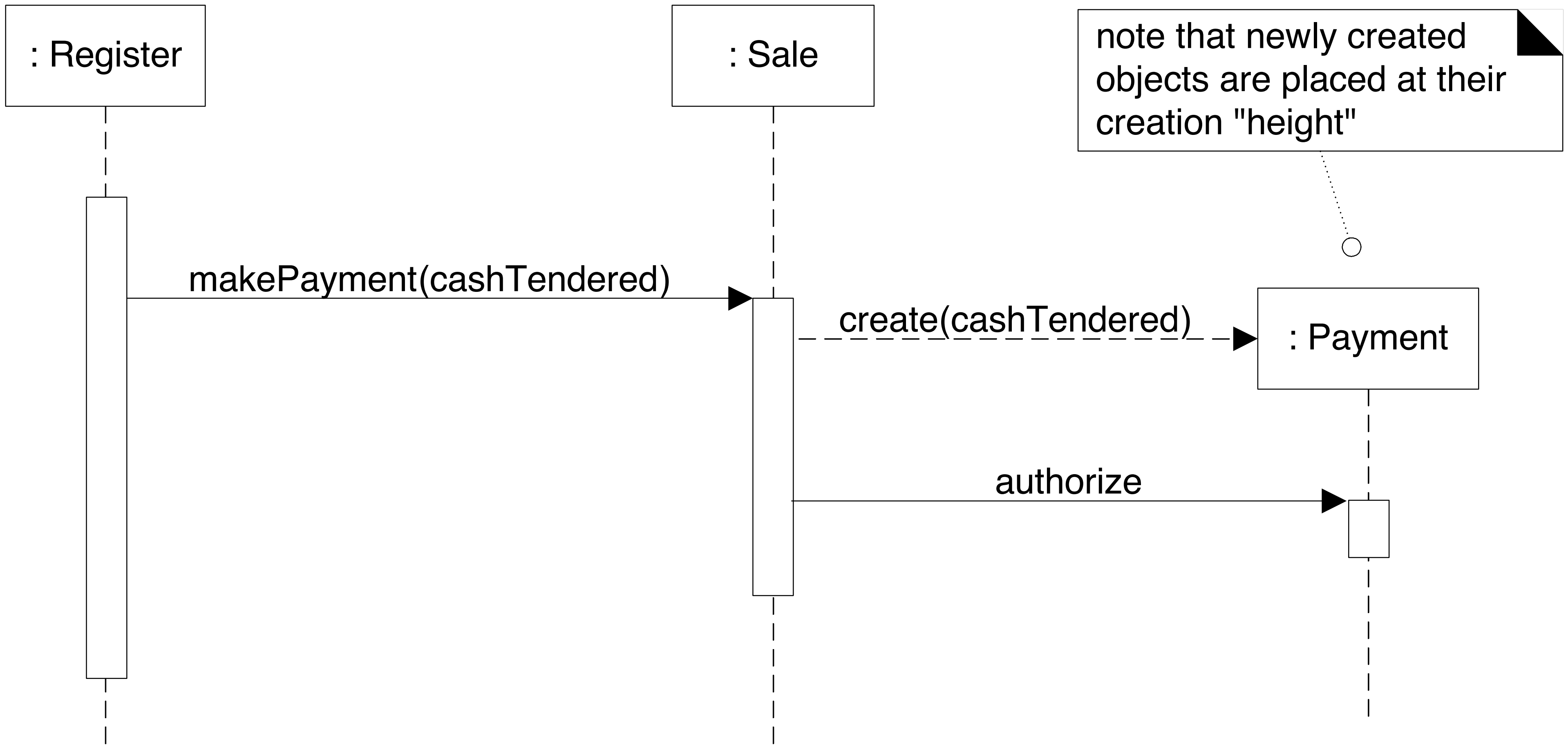


- Two alternatives
  - Using the message syntax
    returnVar = message(parameter)
  - Use a reply or return message at the end of an activation bar

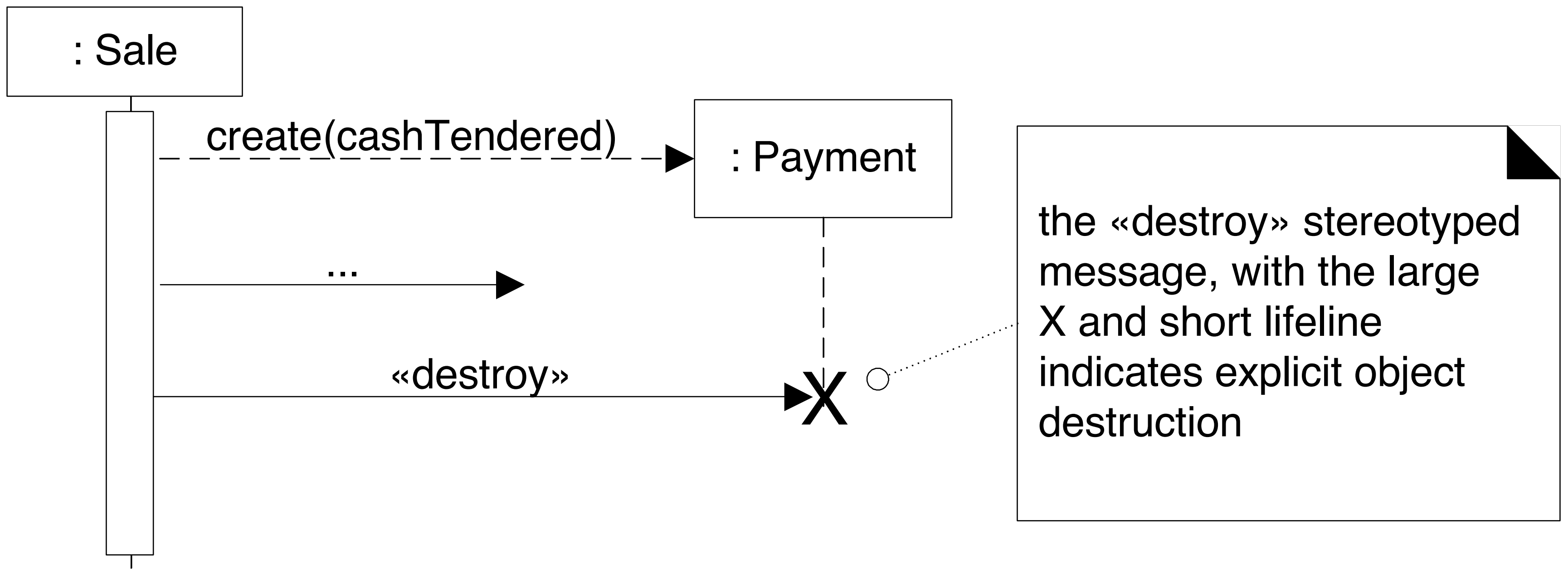
Messages to “self” or “this”



Instance Creation

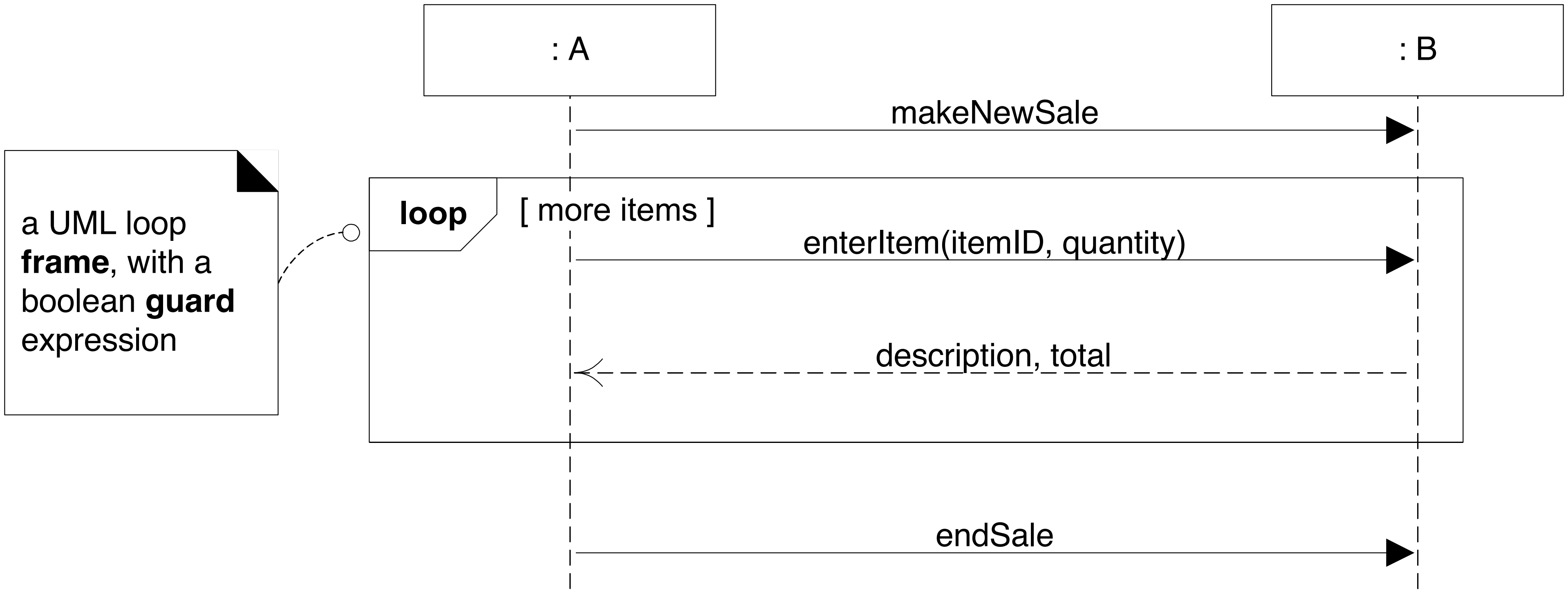

: Register
: Sale
note that newly created objects are placed at their creation "height"
makePayment(cashTendered)
create(cashTendered)
: Payment
authorize

Object Destruction (Object no longer used or usable)

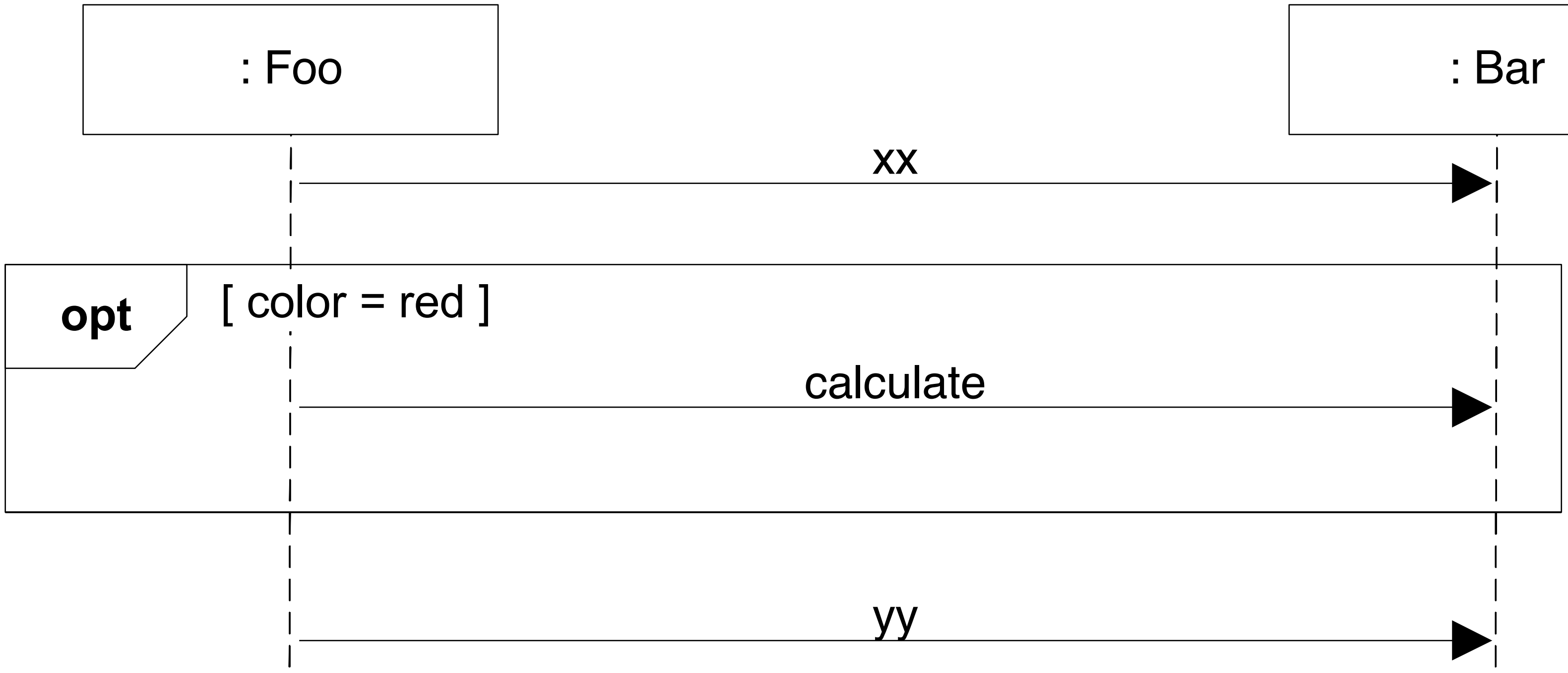

: Sale
create(cashTendered)
: Payment
...
«destroy»
X
the «destroy» stereotyped message, with the large X and short lifeline indicates explicit object destruction

- Object explicitly destroyed or no longer usable (reachable)
  - Example: No variable refers to object any longer
    - Marked for garbage collection

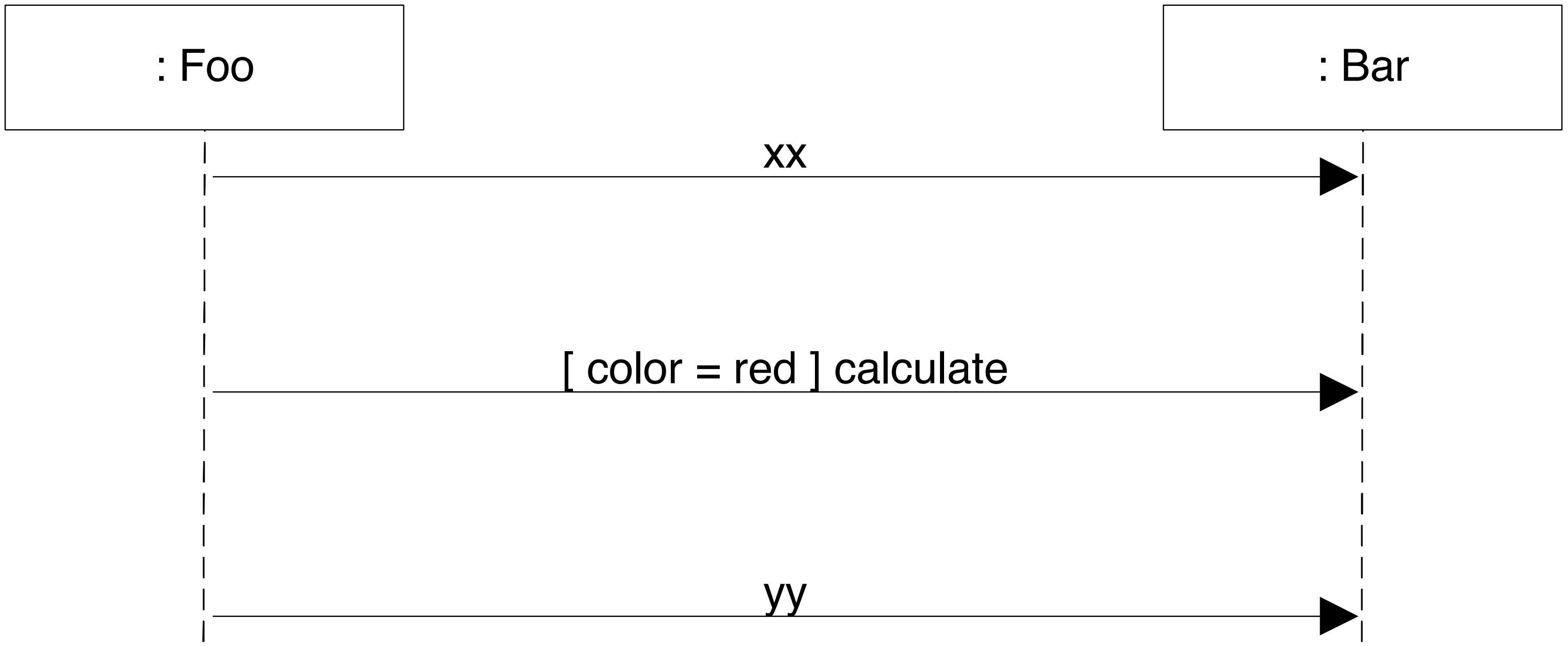
Looping notation


: A
: B
makeNewSale
loop
[more items]
enterItem(itemID, quantity)
description, total
endSale
a UML loop **frame**, with a boolean **guard** expression

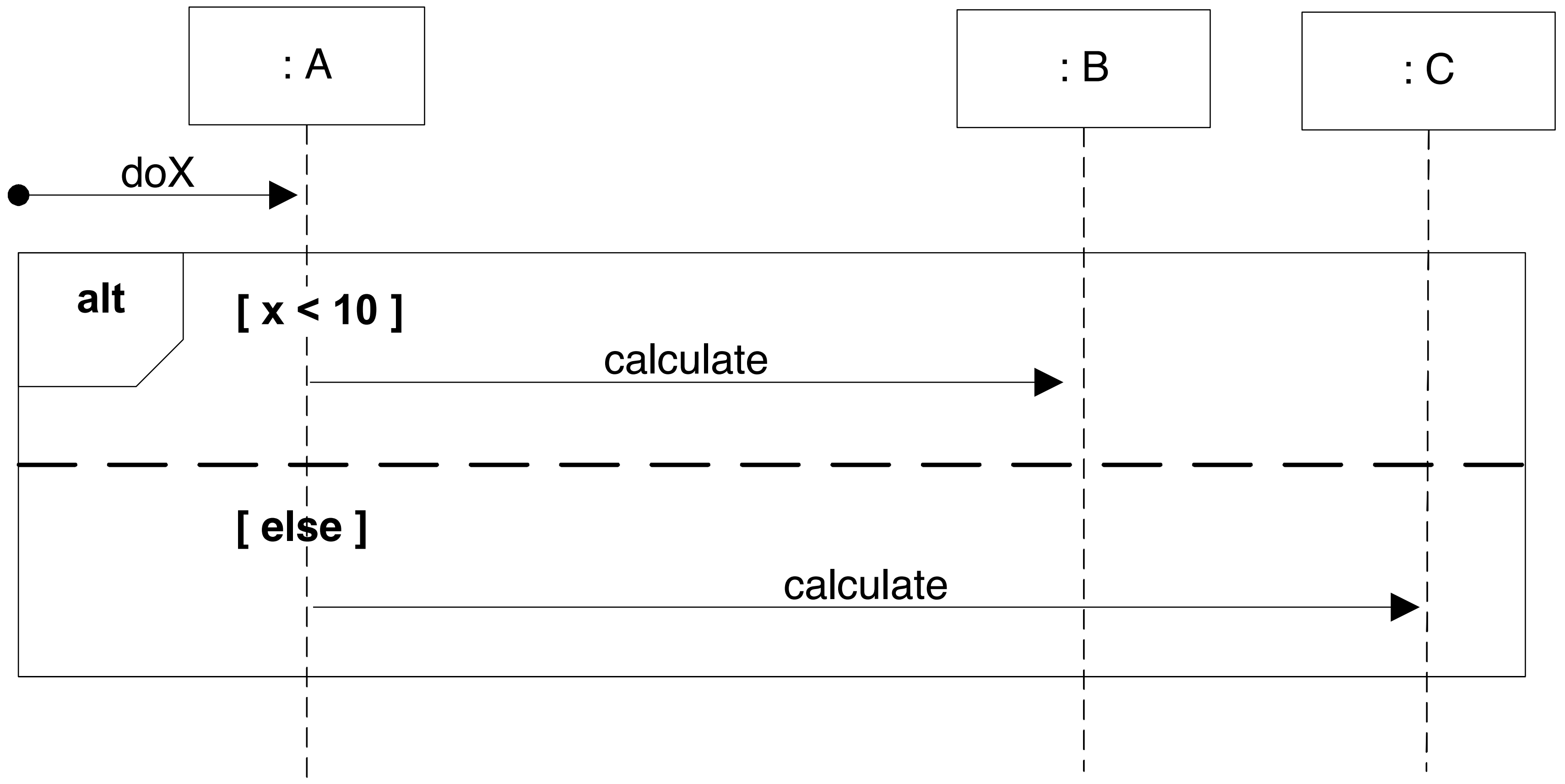
A conditional message



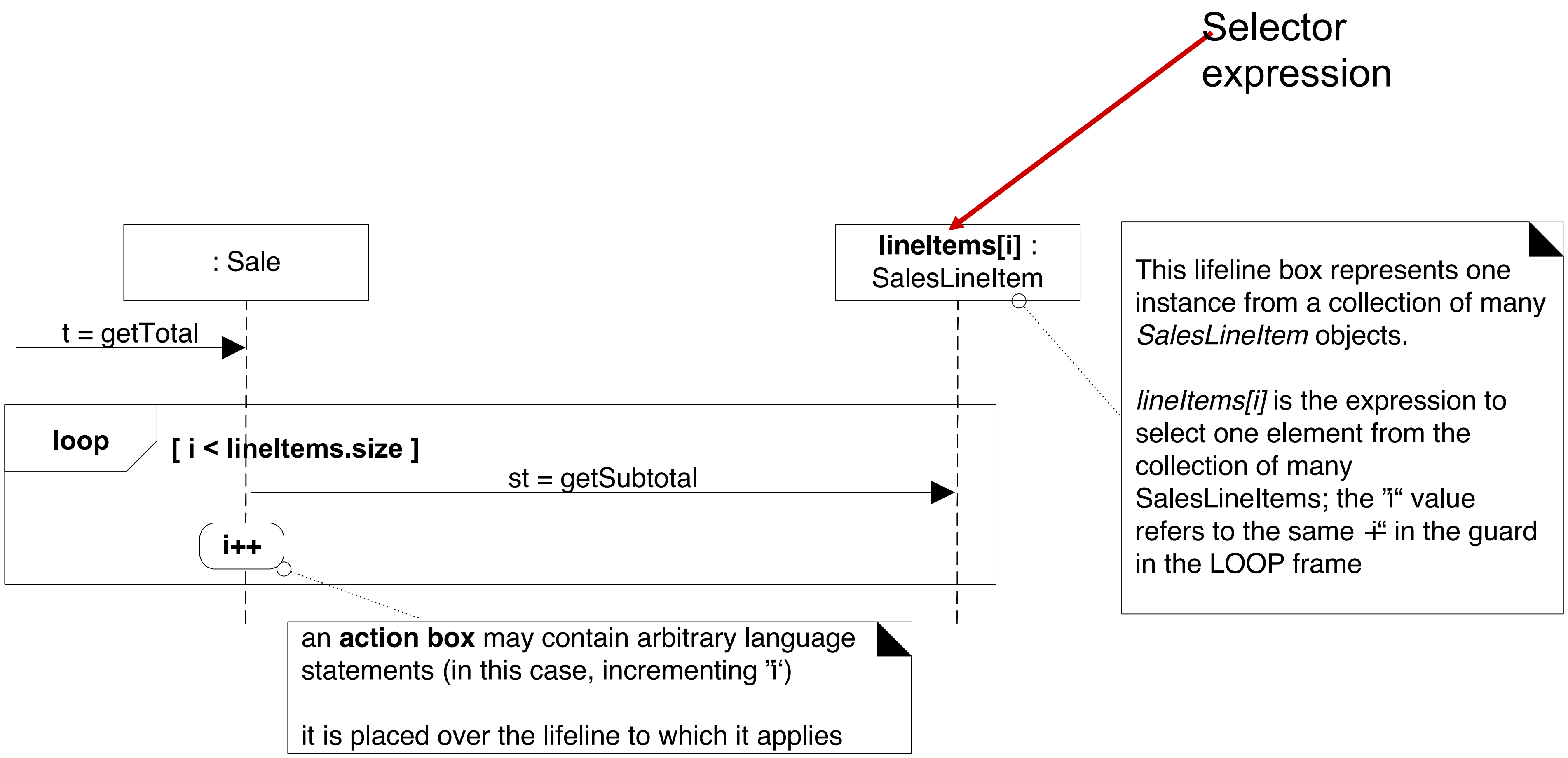
UML Version 1 notation for conditional messages



Mutually exclusive conditional messages



Iteration over a collection: Explicit notation



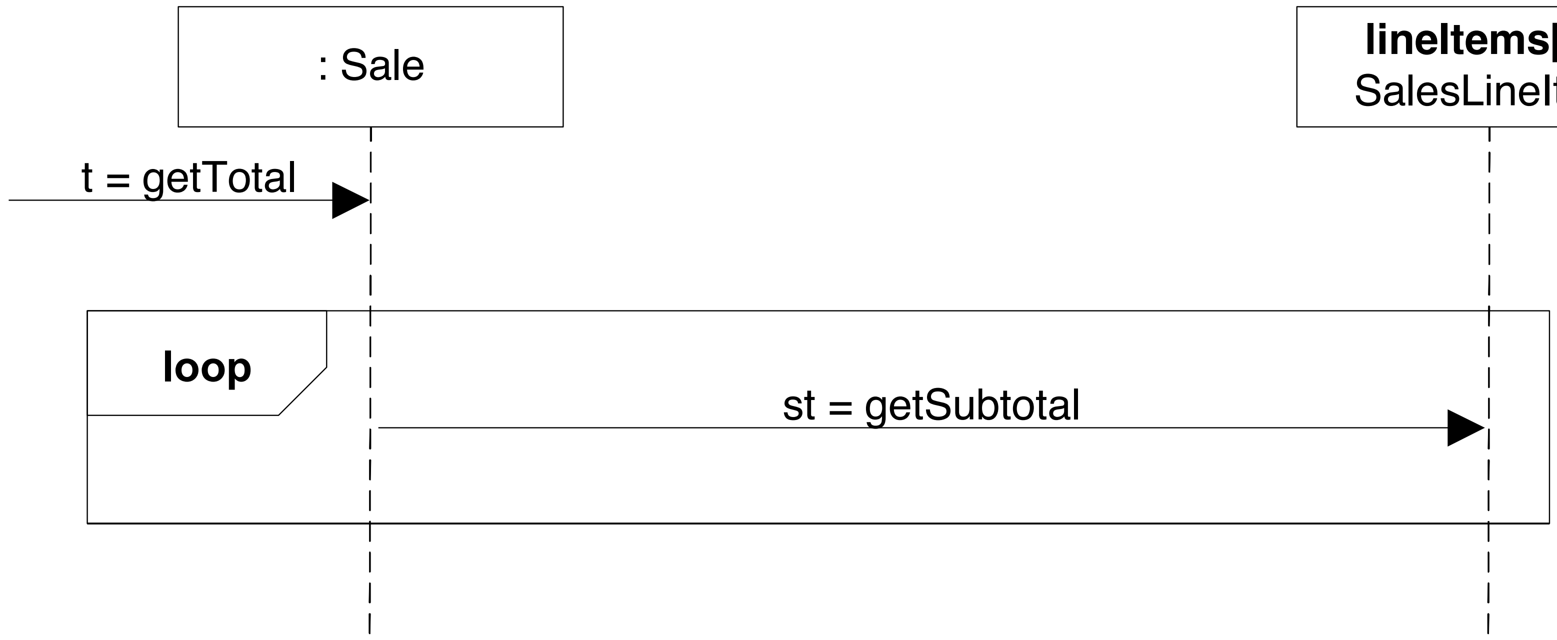
This lifeline box represents one instance from a collection of many *SalesLineItem* objects.

lineItems[i] is the expression to select one element from the collection of many SalesLineItems; the ”i“ value refers to the same ”i“ in the guard in the LOOP frame

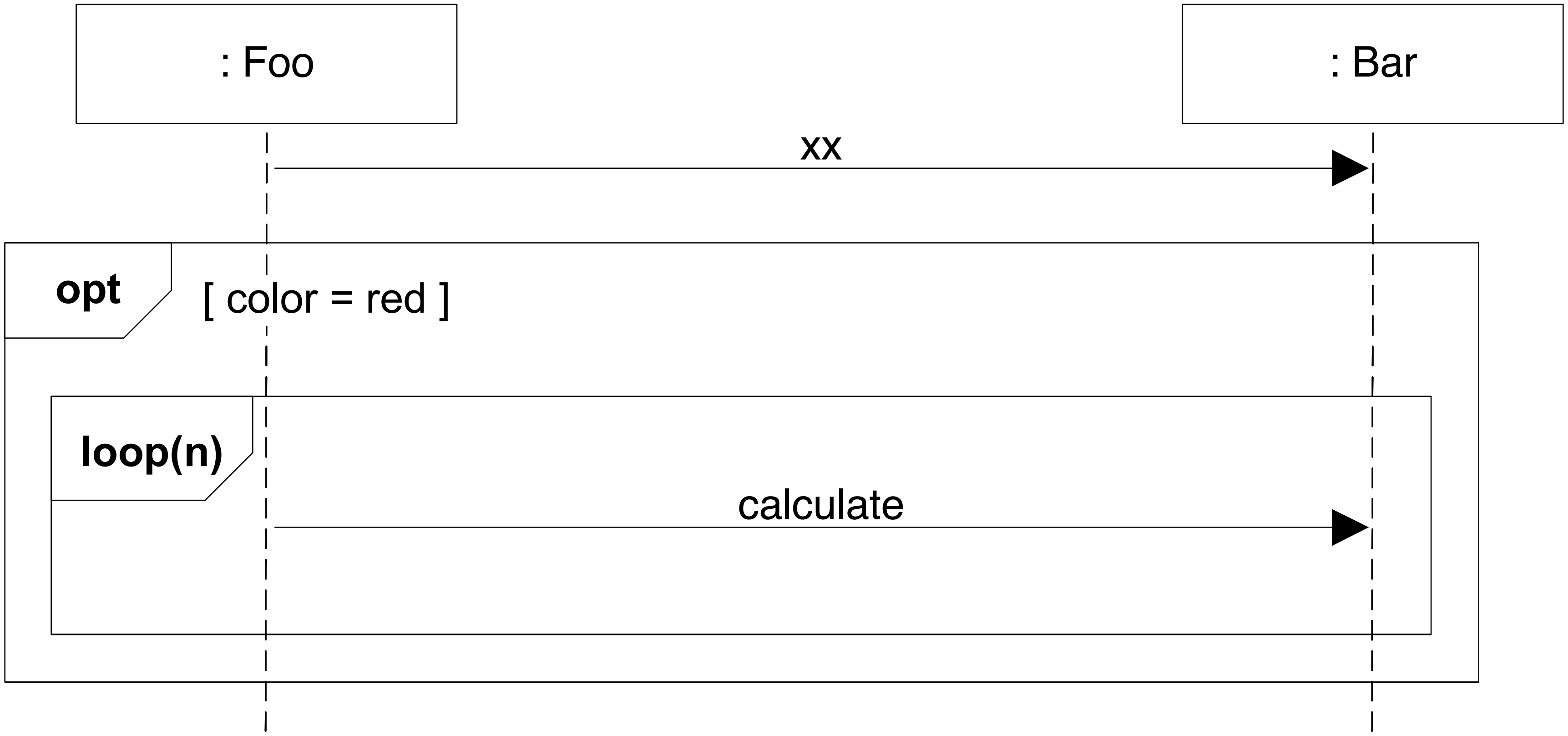
an **action box** may contain arbitrary language statements (in this case, incrementing ’i‘)

it is placed over the lifeline to which it applies

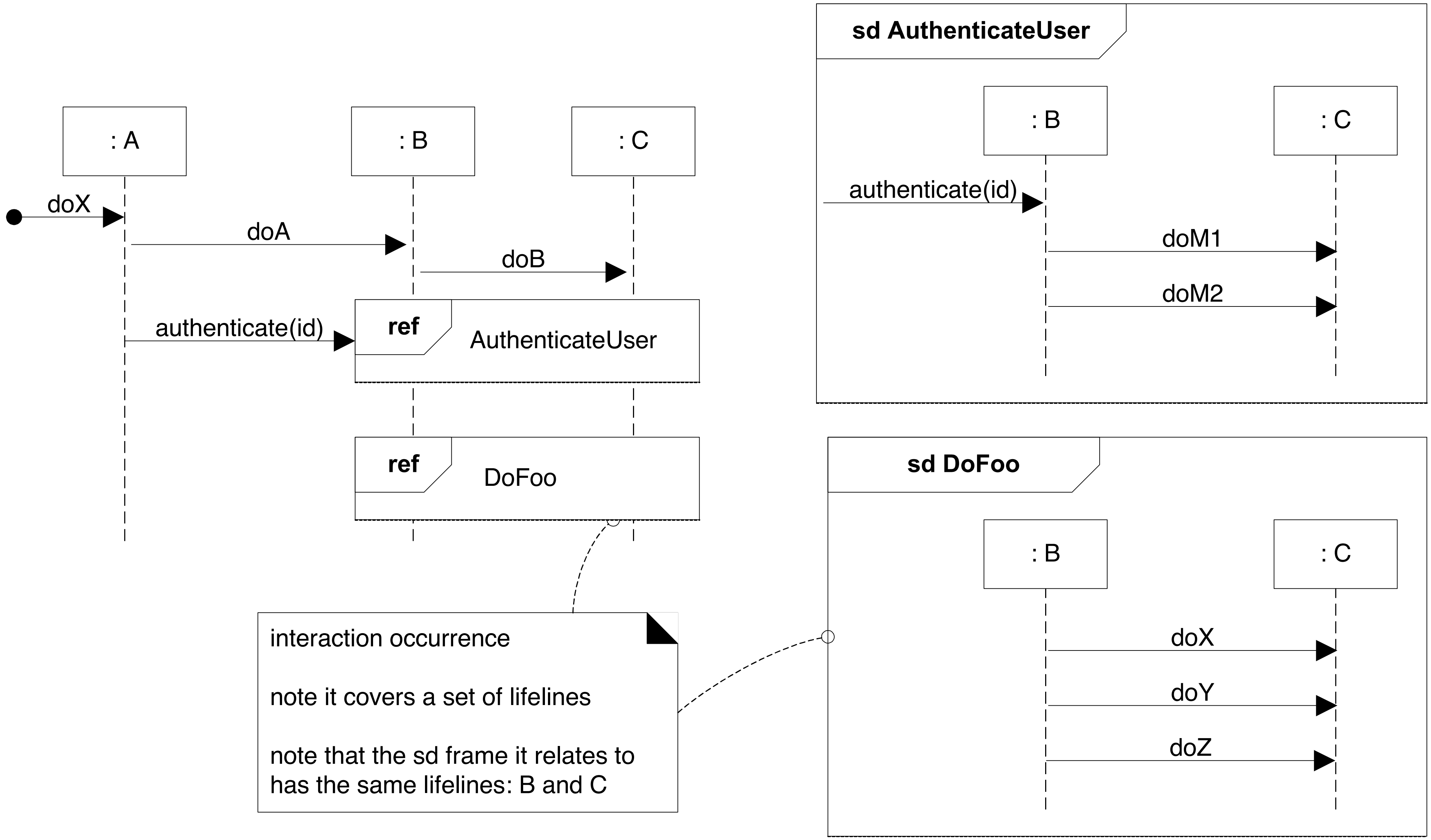
Iteration over a collection: Implicit notation



Nested frames



How to relate interaction diagrams (hierarchical notation)

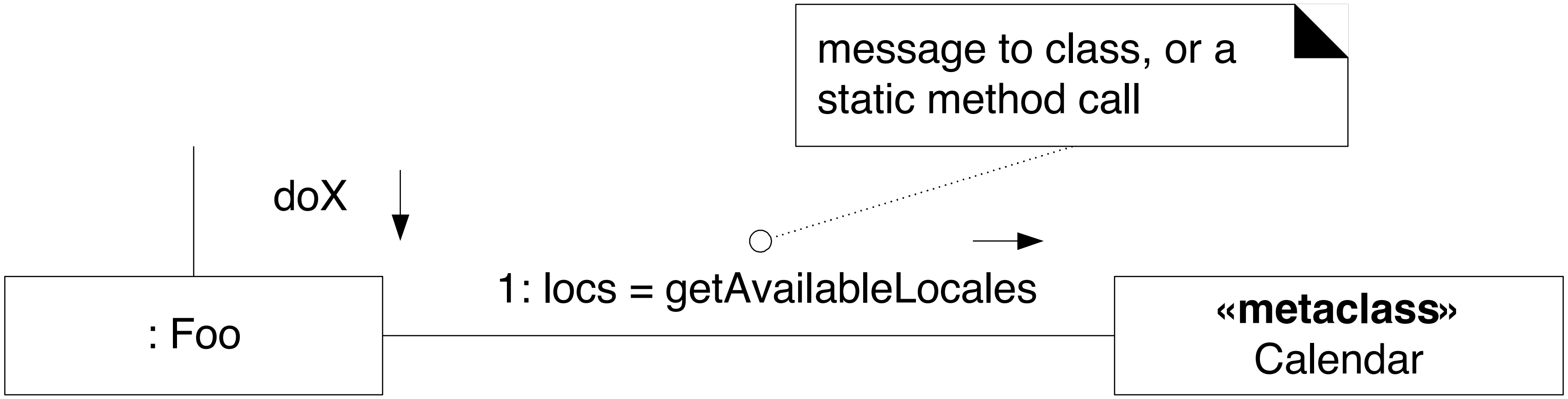


interaction occurrence

note it covers a set of lifelines

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

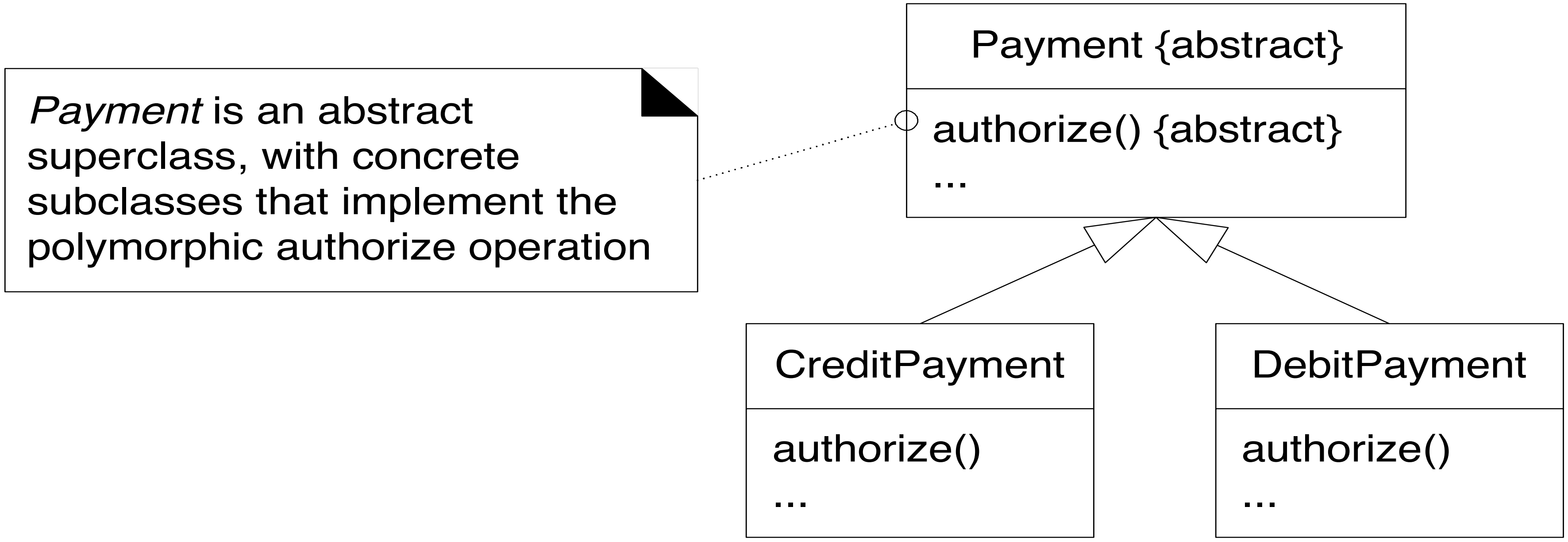
Invoking static or class methods


message to class, or a static method call
doX
1: locs = getAvailableLocales
: Foo
«metaclass» Calendar

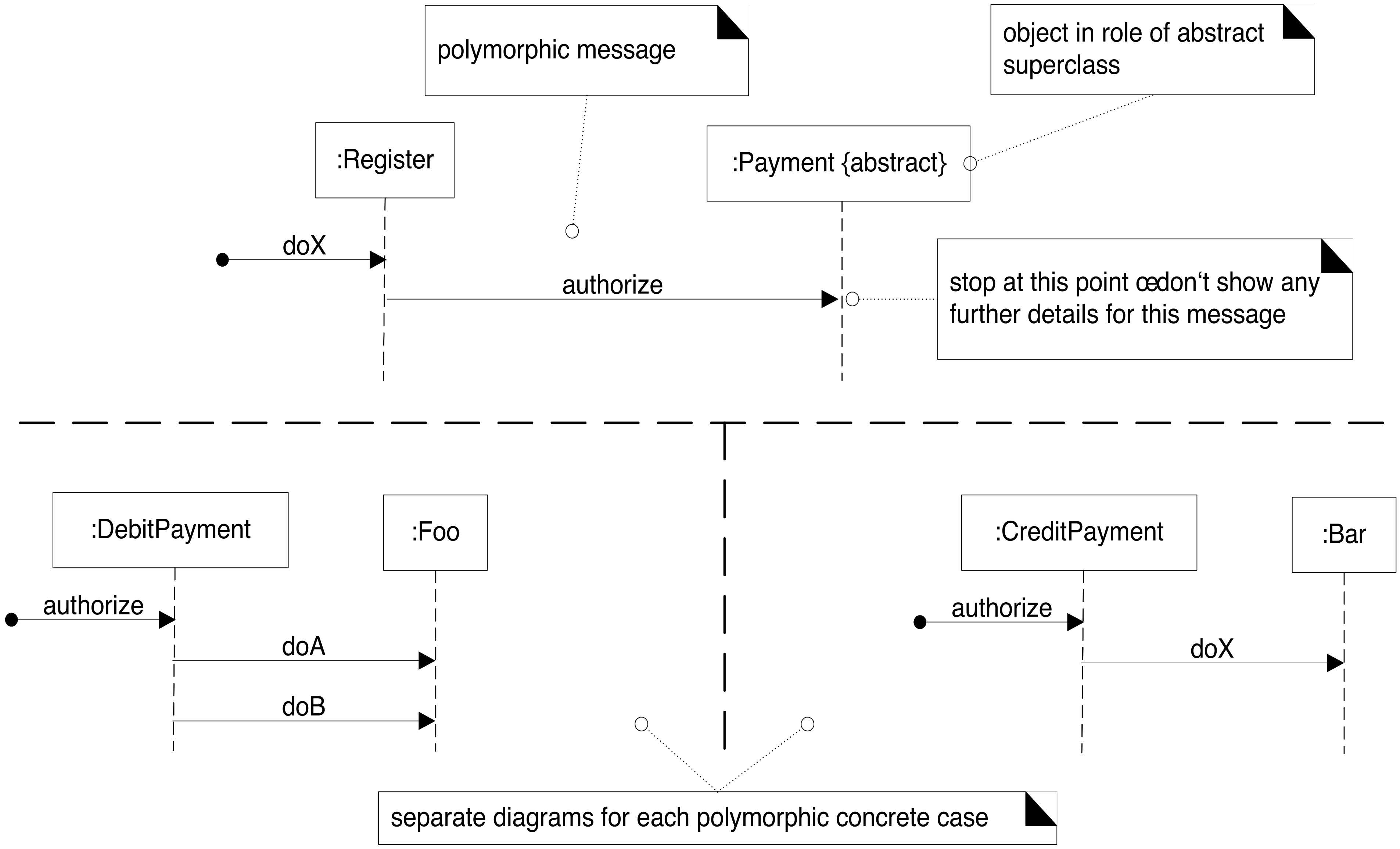
```
public class Foo {

   public void doX() {
       // Static method call on class Calendar
       Locale[] locales = Calendar.getAvailableLocales()
       // ...
   }
  // ...
}
```

Polymorphic Messages and Cases


Payment is an abstract superclass, with concrete subclasses that implement the polymorphic authorize operation
Payment {abstract}
authorize() {abstract}
...
CreditPayment
authorize()
...
DebitPayment
authorize()
...

Polymorphic Messages and Cases


polymorphic message
object in role of abstract superclass
:Register
:Payment {abstract}
doX
authorize
stop at this point œdon‘t show any further details for this message
:DebitPayment
:Foo
authorize
doA
doB
:CreditPayment
:Bar
authorize
doX
separate diagrams for each polymorphic concrete case

Asynchronous vs. Synchronous Calls

- Asynchronous message: Does not wait for a response
  - "It doesn't block"
- Used in multi-threaded environments
  - New threads can be created and initiated
- Example: In Java
  - `Thread.start`
  - `Runnable.run`

initiate execution of a new thread

Asynchronous vs. Synchronous Calls

a stick arrow in UML implies an asynchronous call

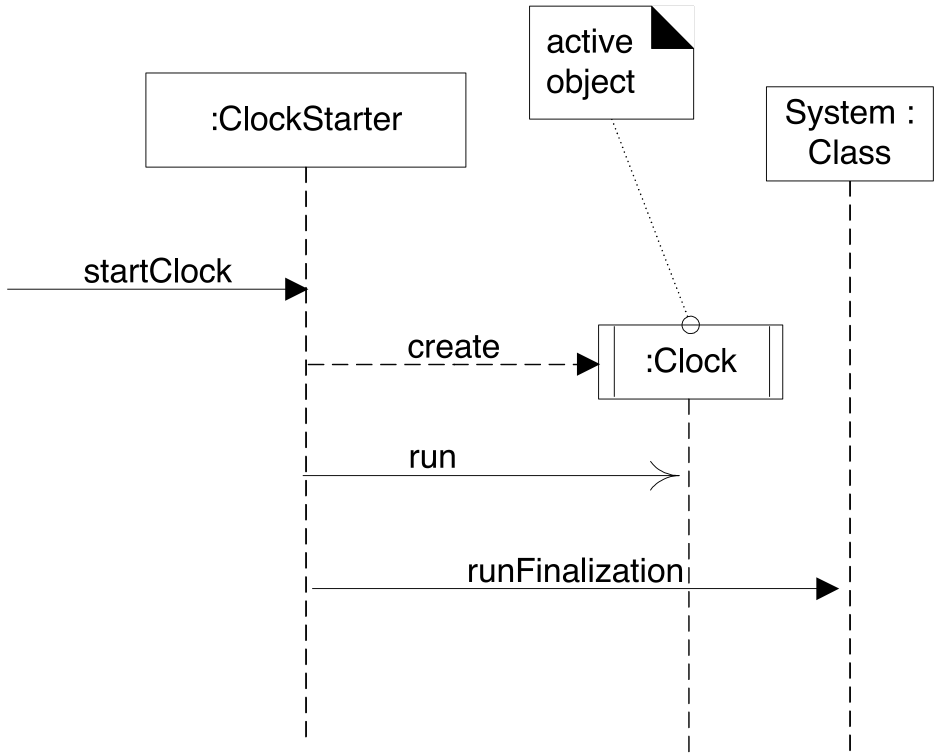
a filled arrow is the more common synchronous call

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

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

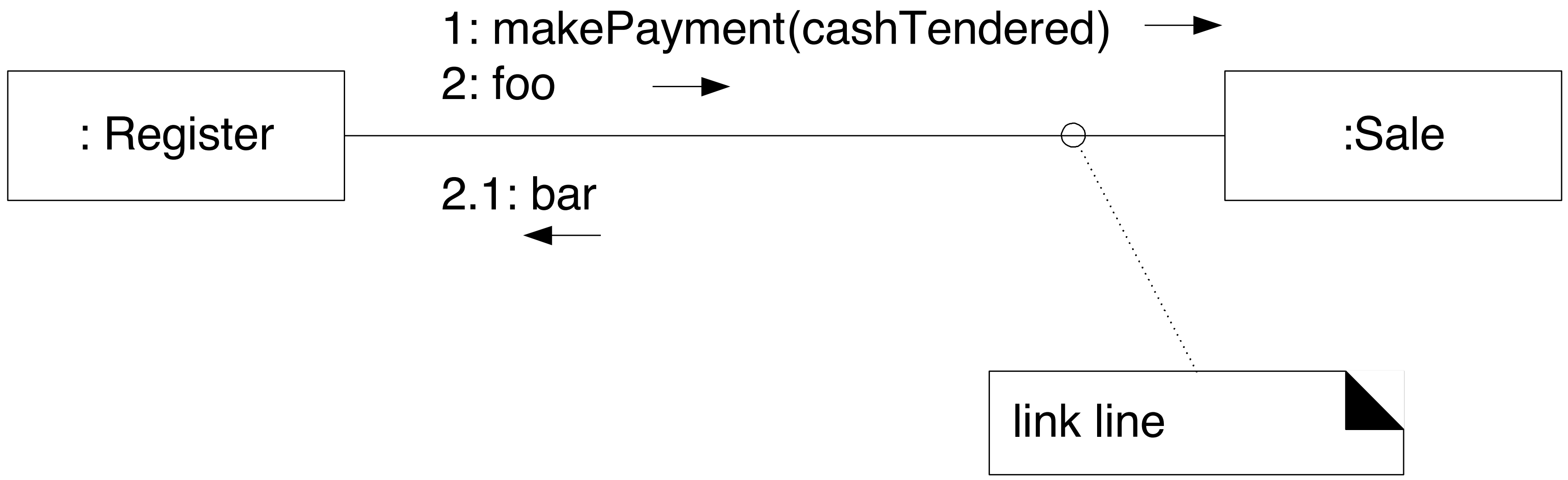
the asynchronous *start* call always invokes the *run* method on the *Runnable* (*Clock*) object

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



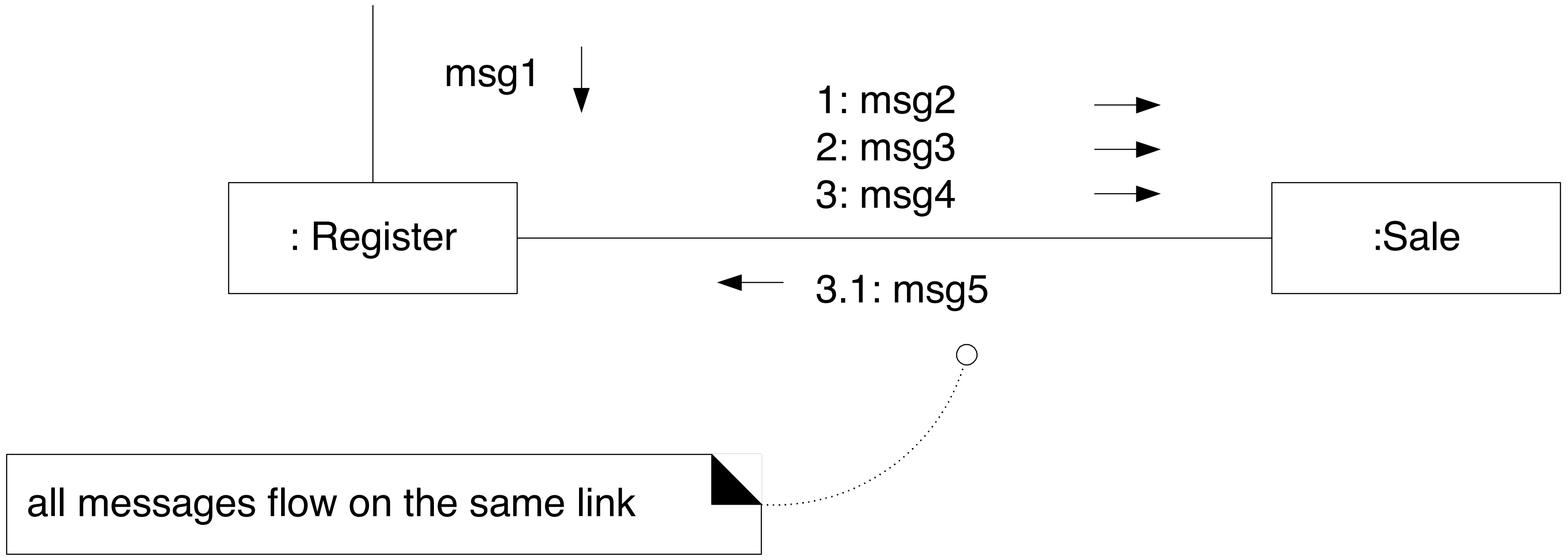
- Active object: Each instance runs on and controls its own thread of execution
  - Example: Clock

Collaboration Diagram Notation

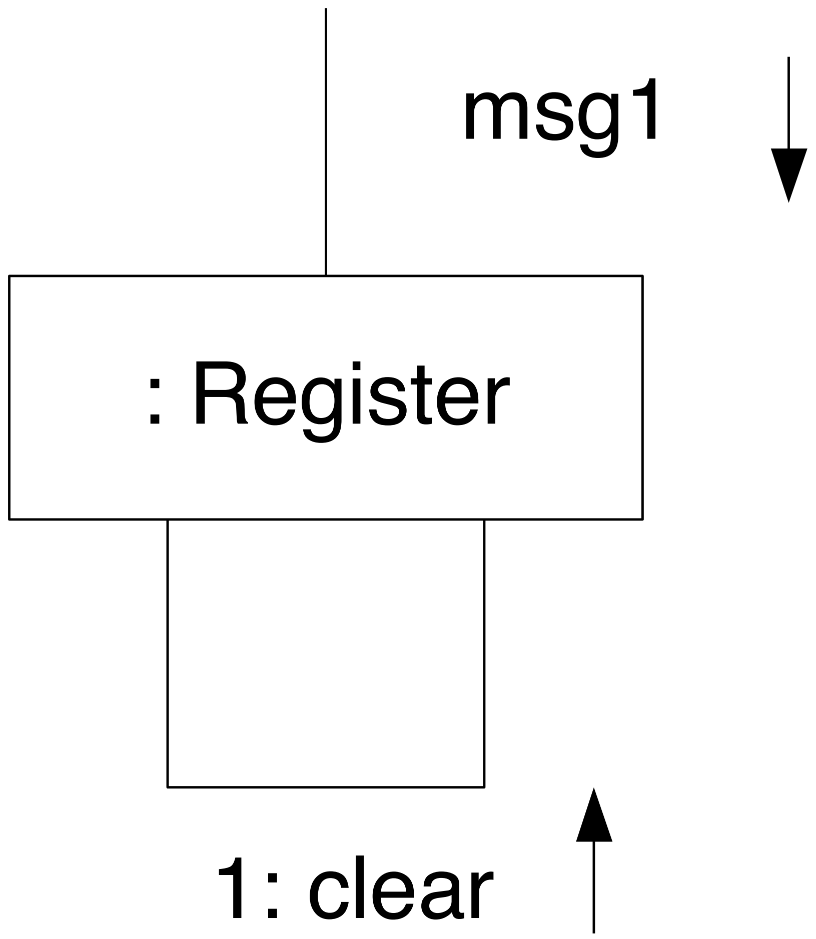


- Link: Connection path between two objects
  - Indicates a form of navigation or visibility between the objects
  - Formally: An instance of an association
- There can be only one link between two objects
  - Multiple messages in both directions flow along this link

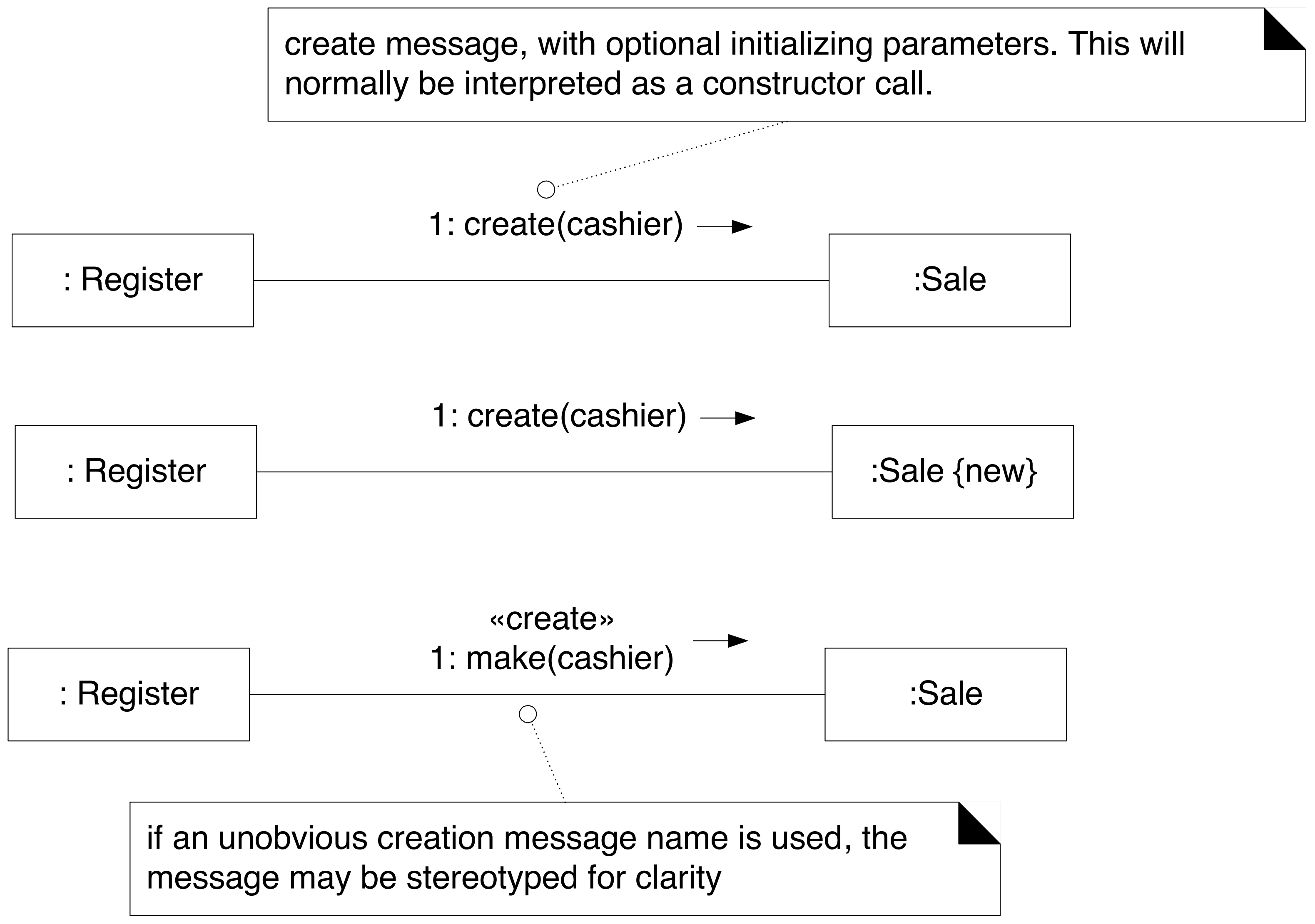
Communication Diagram Notation



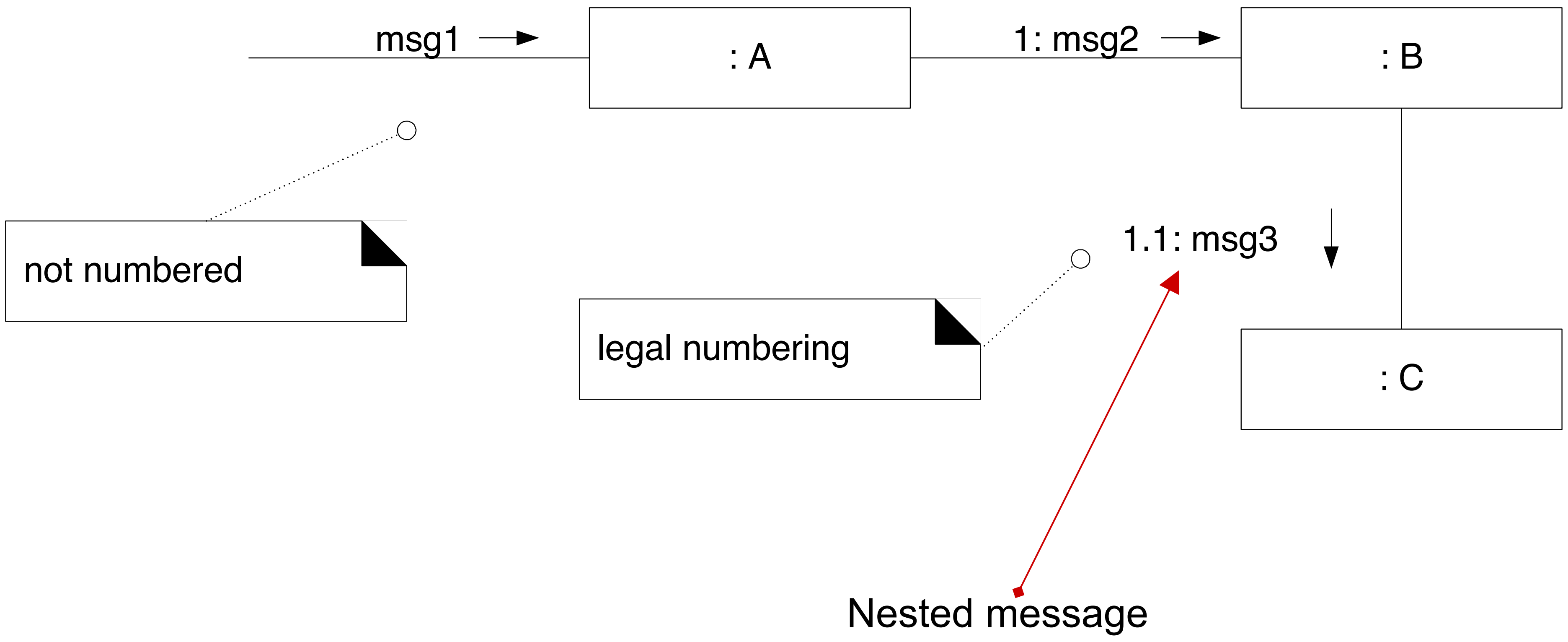
Messages to “self” or “this”



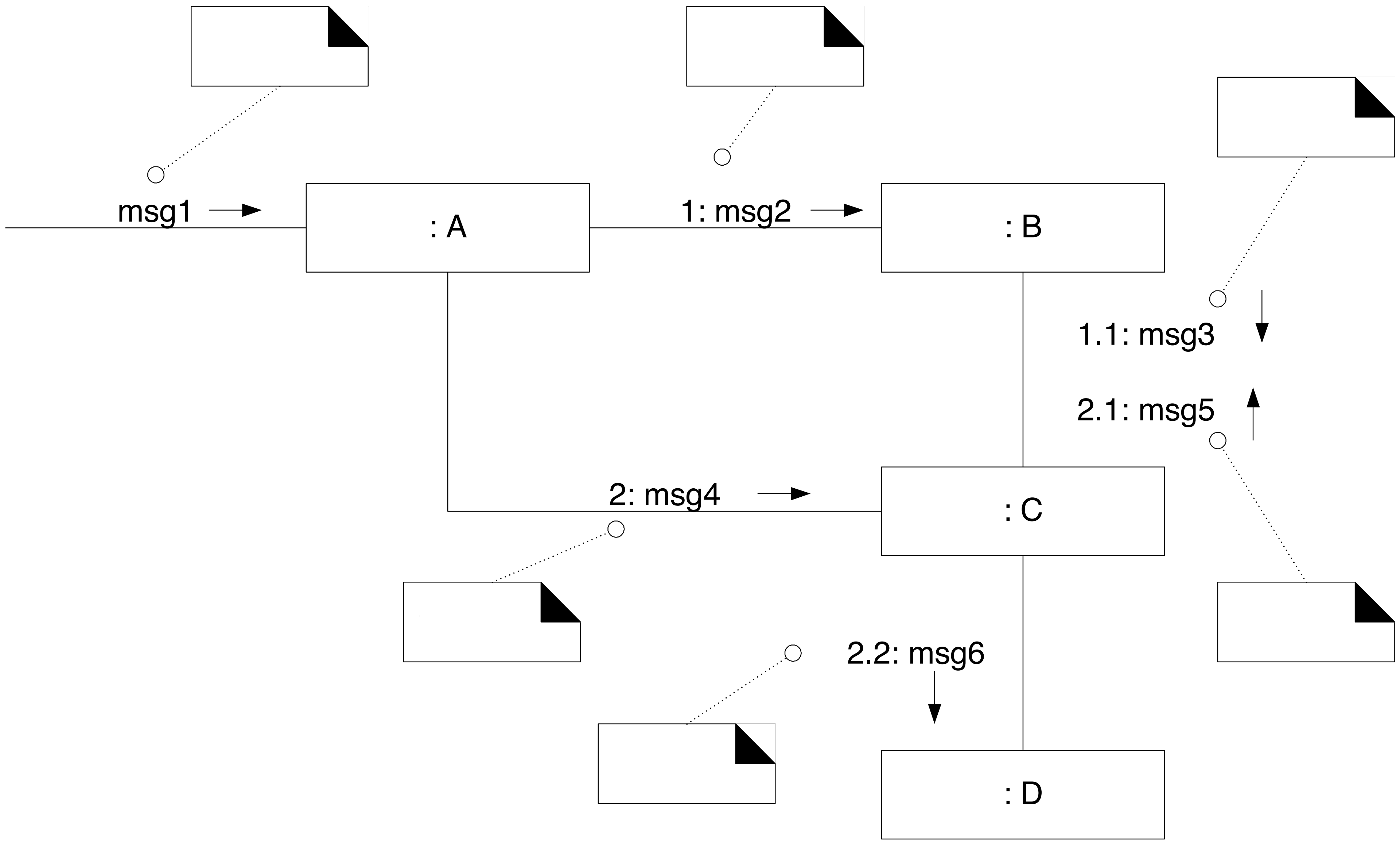
Instance creation in communication diagrams


create message, with optional initializing parameters. This will normally be interpreted as a constructor call.
1: create(cashier)
: Register
:Sale
1: create(cashier)
: Register
:Sale {new}
«create»
1: make(cashier)
: Register
:Sale
if an unobvious creation message name is used, the message may be stereotyped for clarity

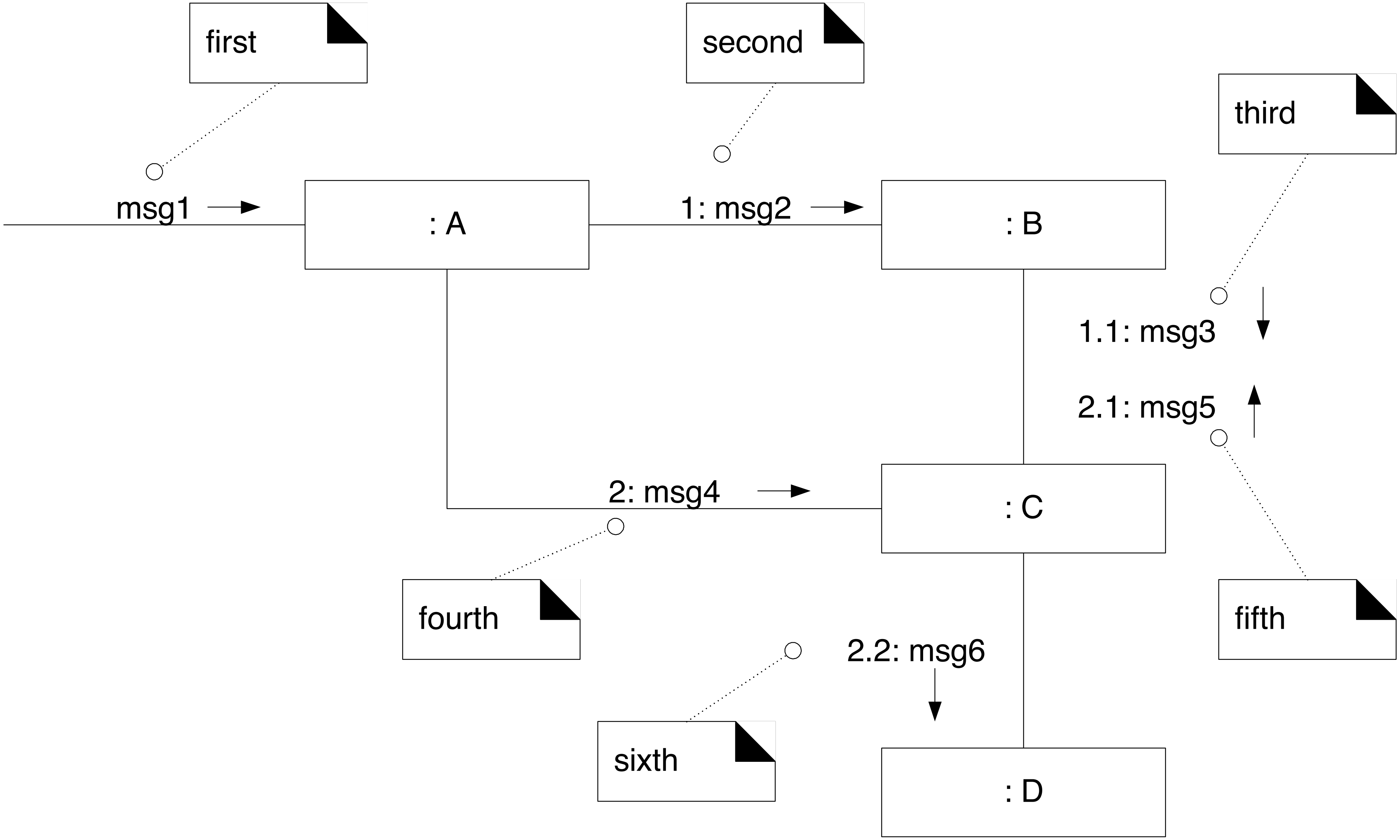
Sequence Numbering



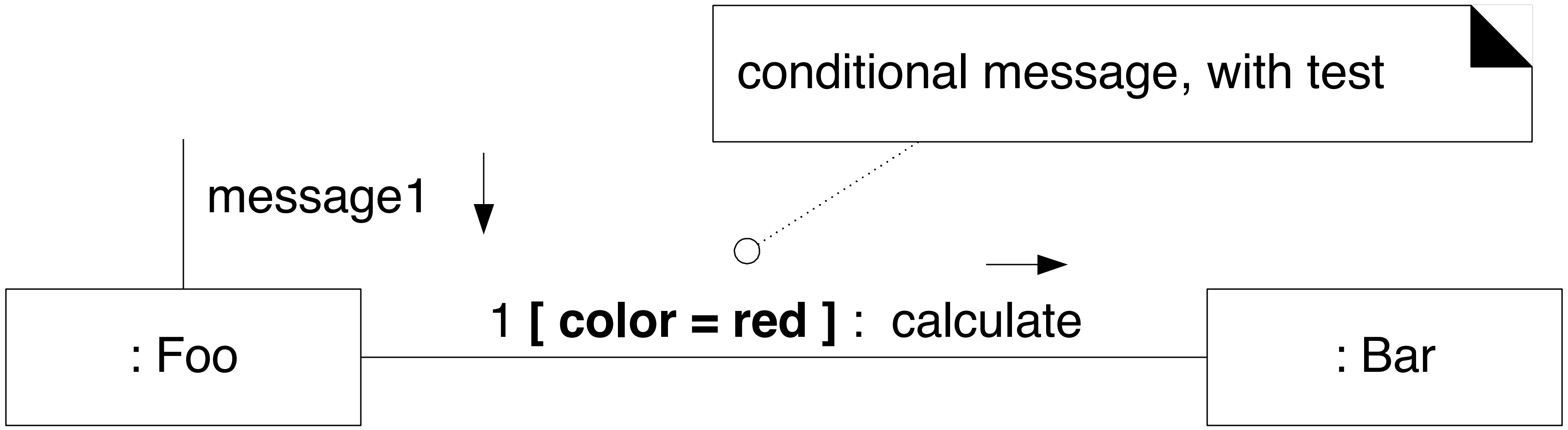
Sequence Numbering



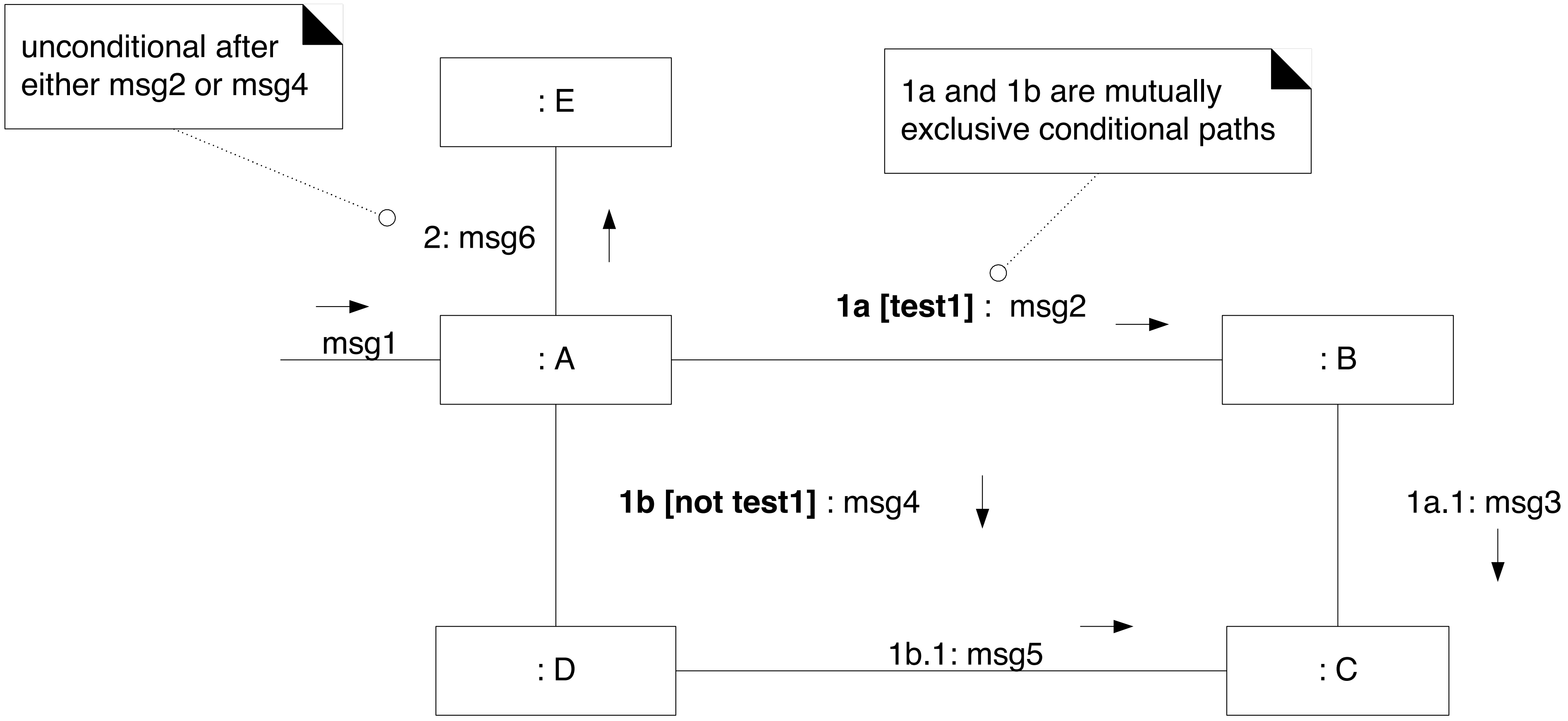
Sequence Numbering



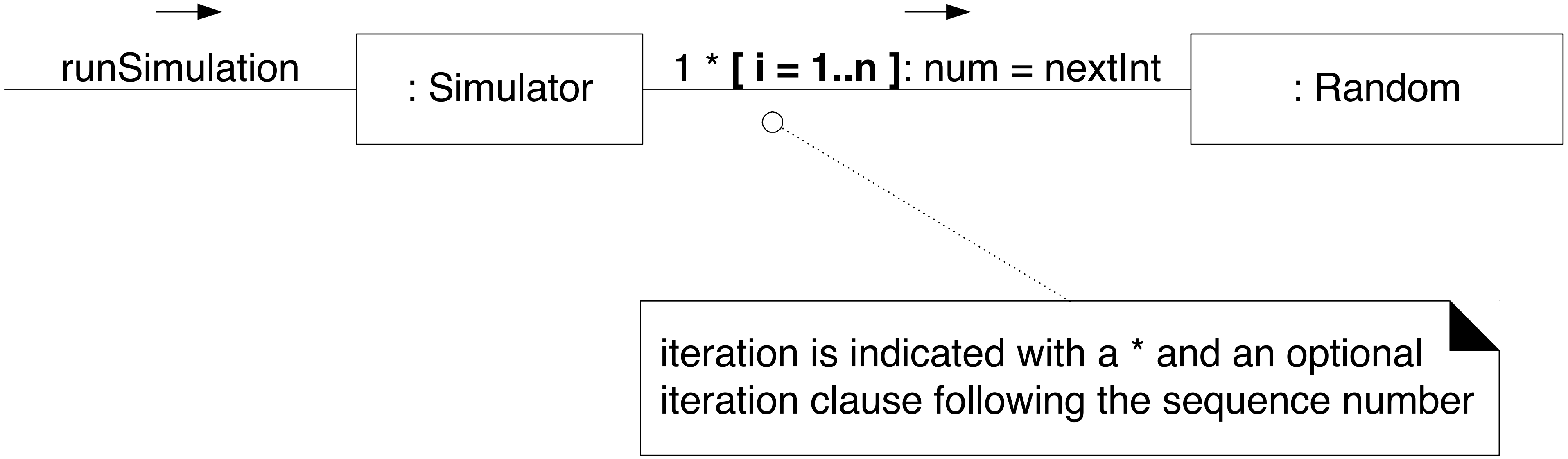
Conditional messages



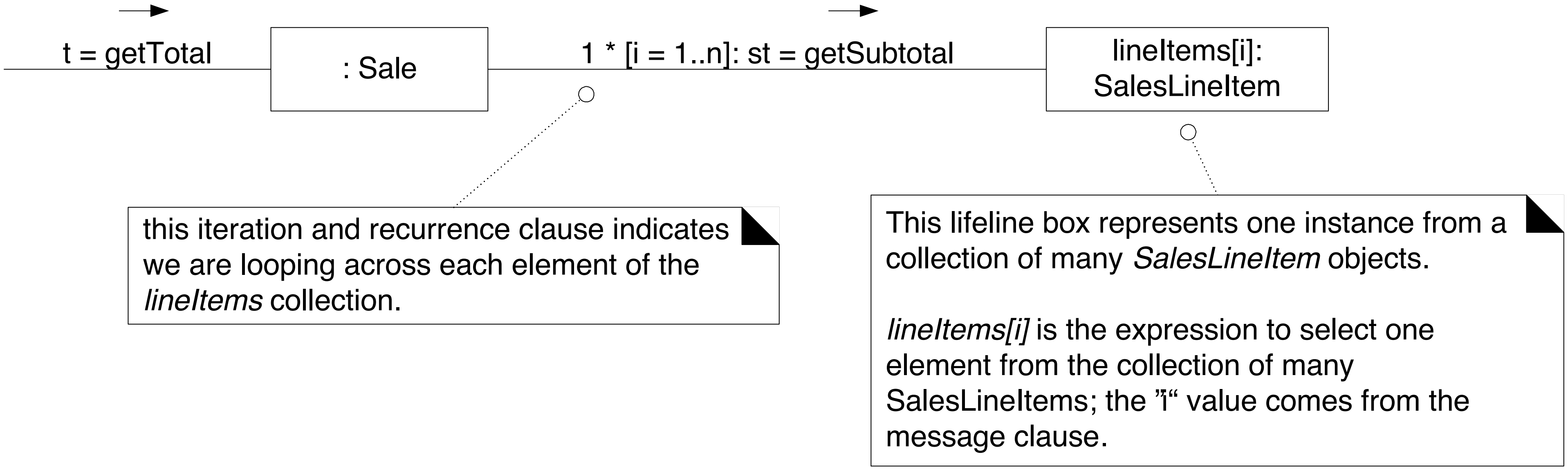
Mutually exclusive conditional messages


unconditional after either msg2 or msg4
: E
1a and 1b are mutually exclusive conditional paths
2: msg6
1a [test1] : msg2
msg1
: A
: B
1b [not test1] : msg4
1a.1: msg3
: D
1b.1: msg5
: C

Iteration in communication diagrams


runSimulation
: Simulator
1 * [i = 1..n]: num = nextInt
: Random
iteration is indicated with a * and an optional iteration clause following the sequence number

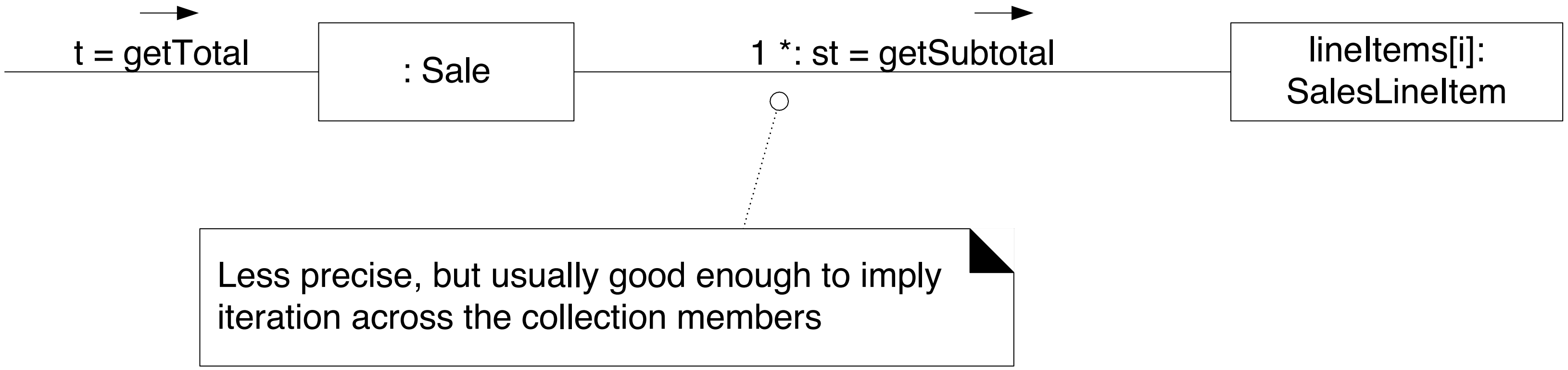
Iteration over a collection



this iteration and recurrence clause indicates we are looping across each element of the *lineItems* collection.

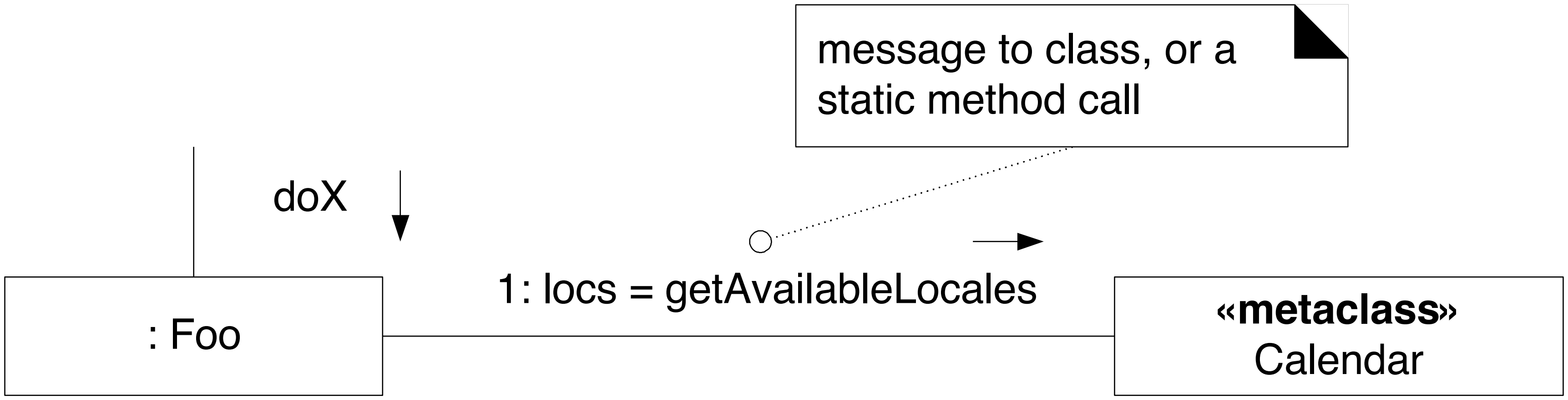
This lifeline box represents one instance from a collection of many *SalesLineItem* objects.

lineItems[i] is the expression to select one element from the collection of many SalesLineItems; the ”i“ value comes from the message clause.

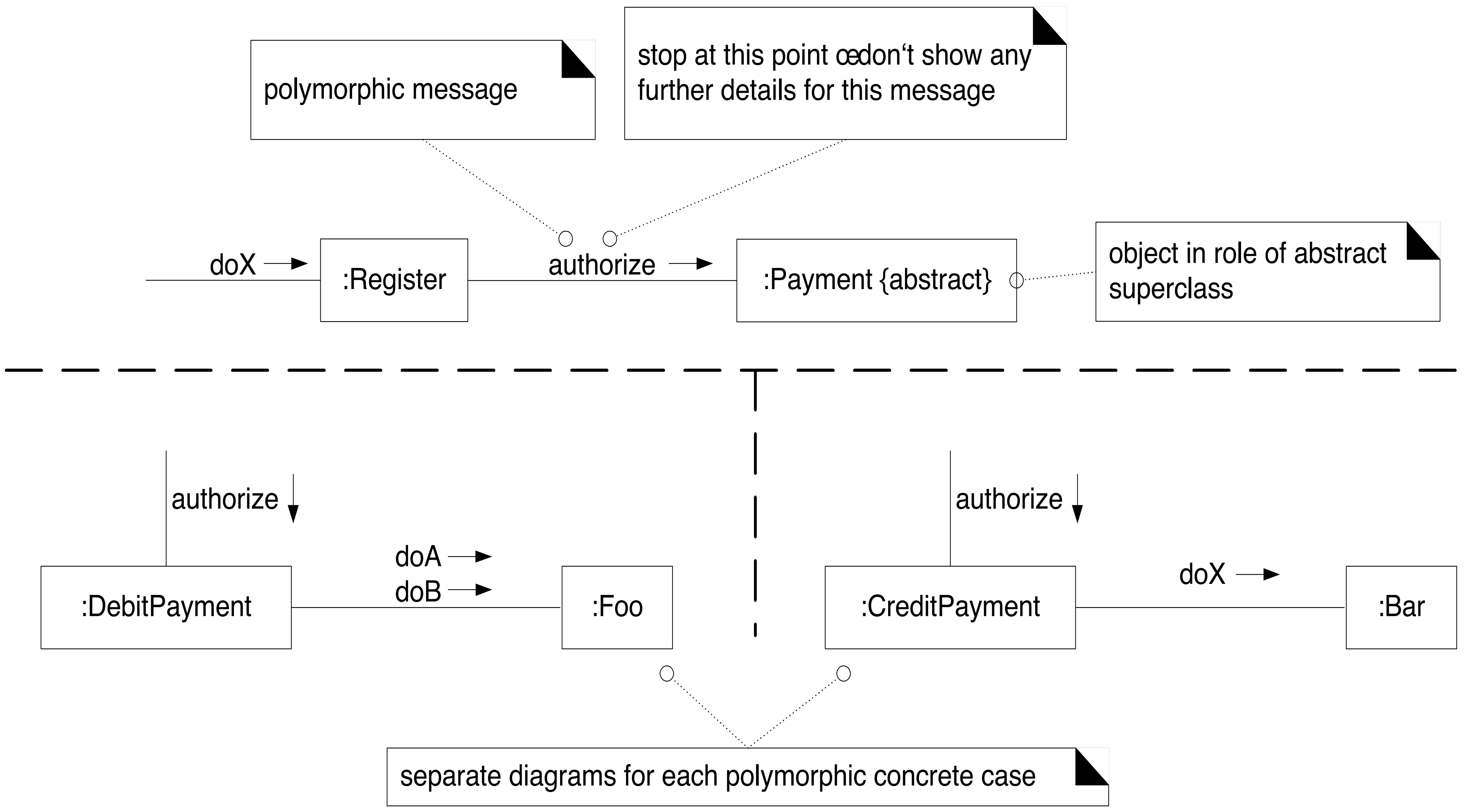


Less precise, but usually good enough to imply iteration across the collection members

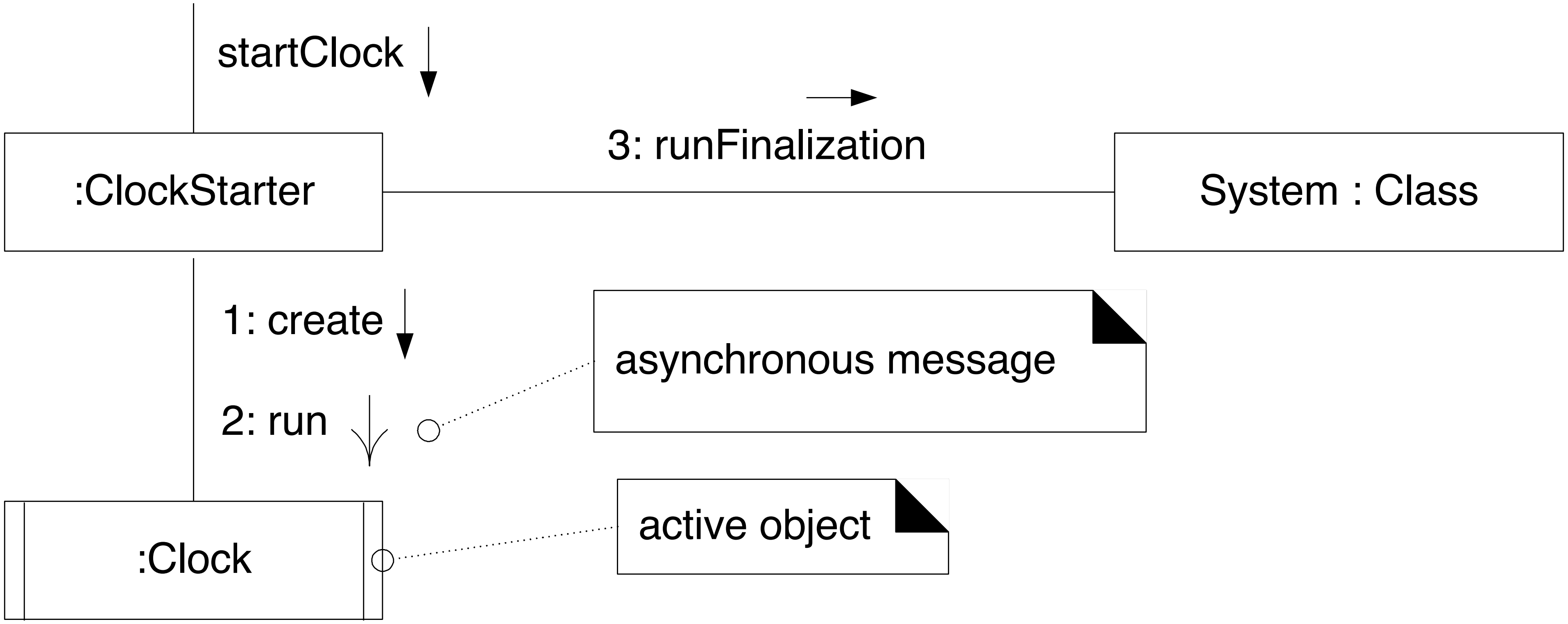
Static method invocation (message to a class)



Modeling polymorphic cases in communication diagrams


polymorphic message
stop at this point œdon‘t show any further details for this message
doX
:Register
authorize
:Payment {abstract}
object in role of abstract superclass
authorize
:DebitPayment
doA
doB
:Foo
authorize
:CreditPayment
doX
:Bar
separate diagrams for each polymorphic concrete case

Asynchronous messages in communication diagrams


startClock
:ClockStarter
3: runFinalization
System : Class
1: create
2: run
asynchronous message
:Clock
active object