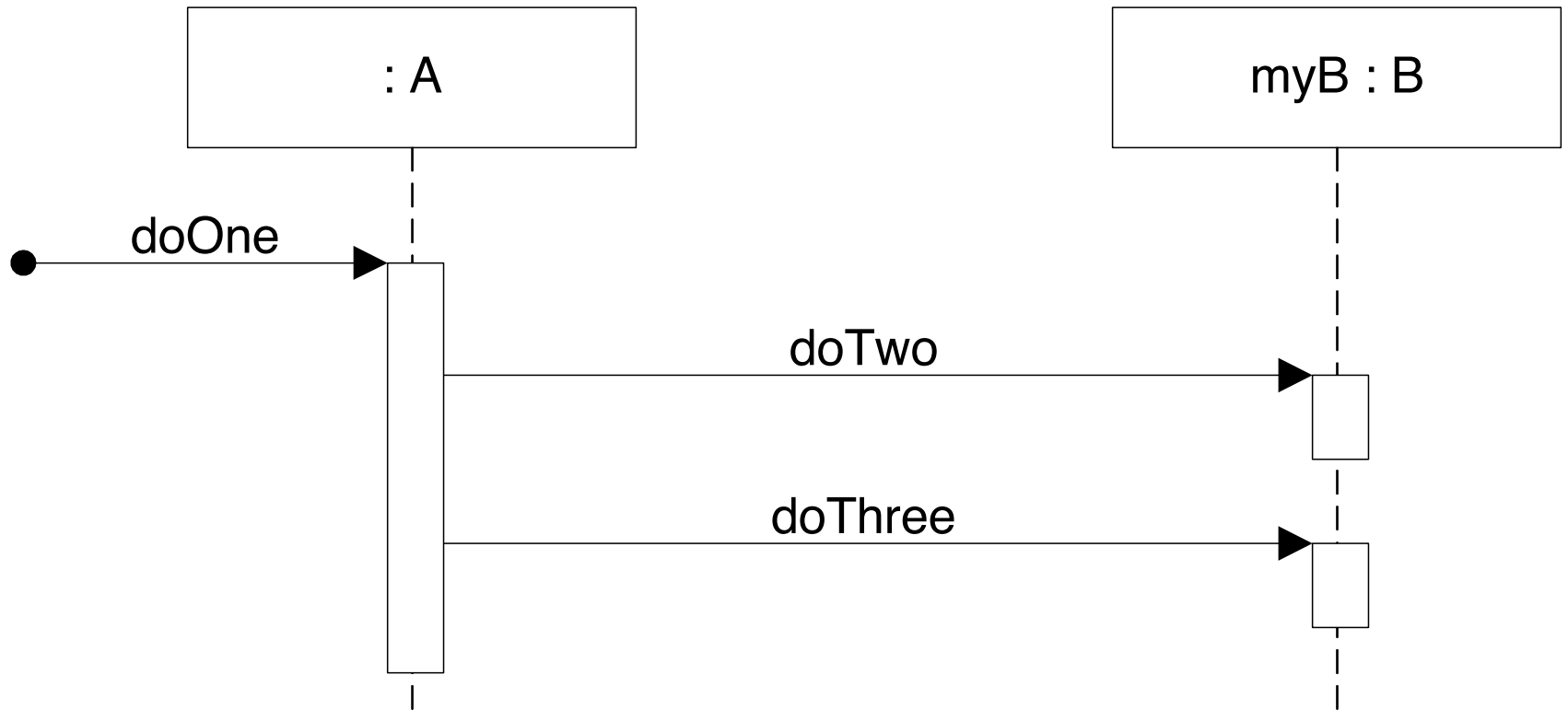# Chapter 15

Dynamic modeling of object:
Interaction Diagrams
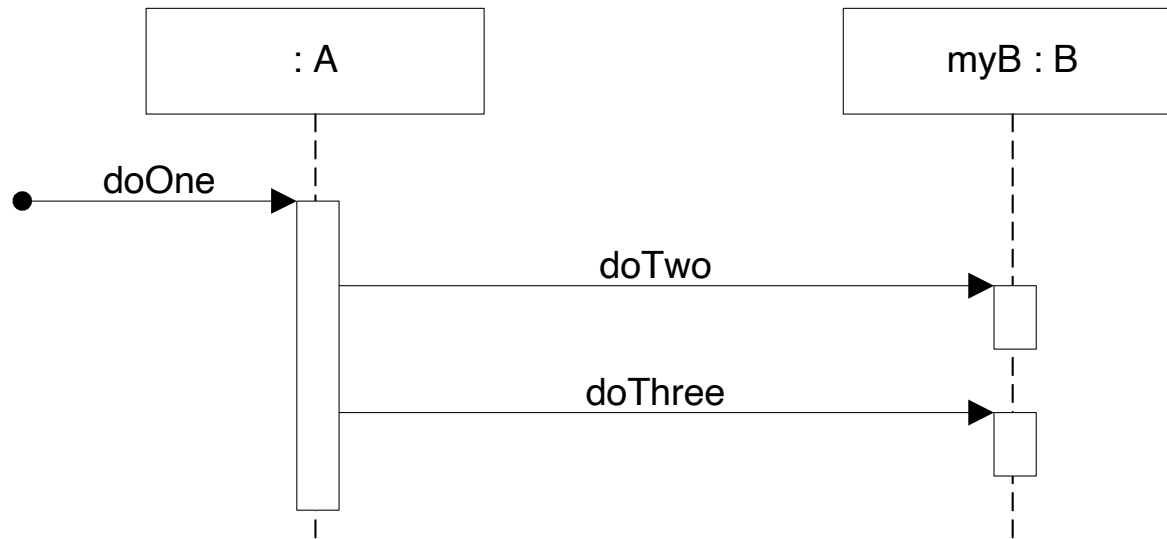
UML Interaction Diagrams

- ## Two variants
  - ### Sequence diagrams
  - ### Communication diagrams

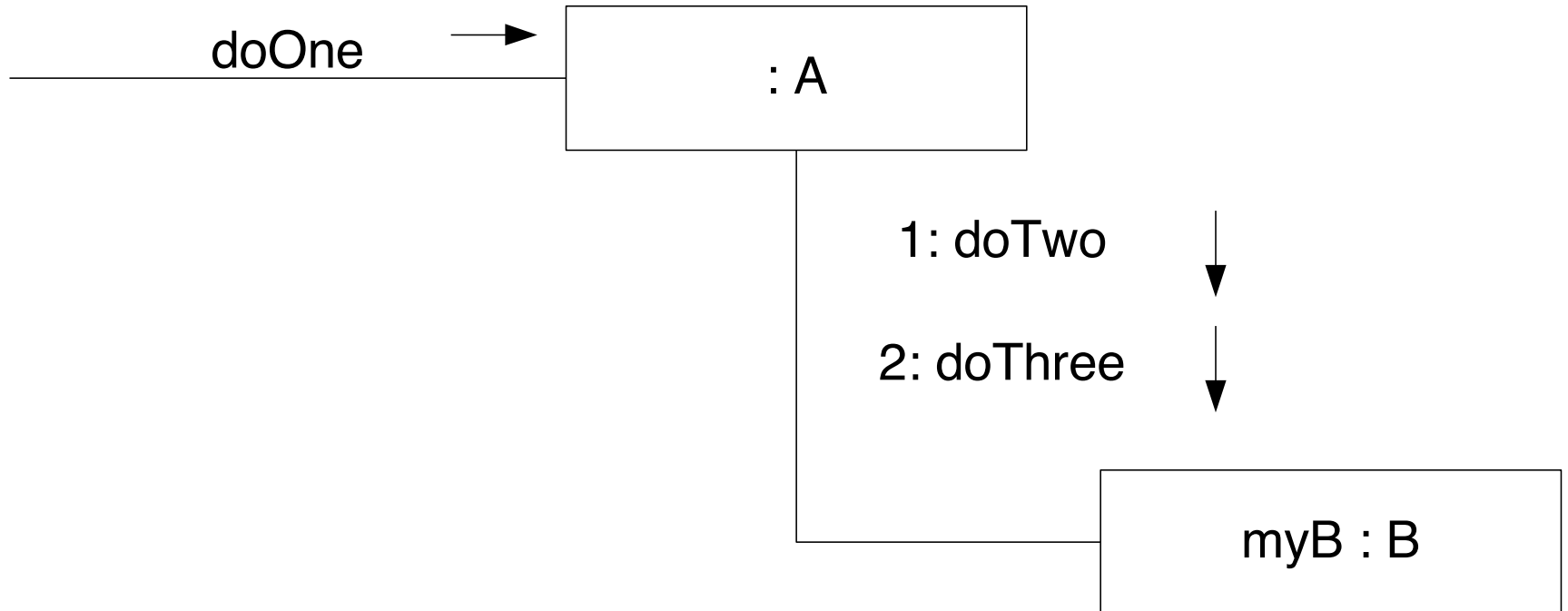# Example: Sequence Diagram

What does this represent in code?


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

    public void doOne() {
        myB.doTwo();
        myB.doThree();
    }
}
```
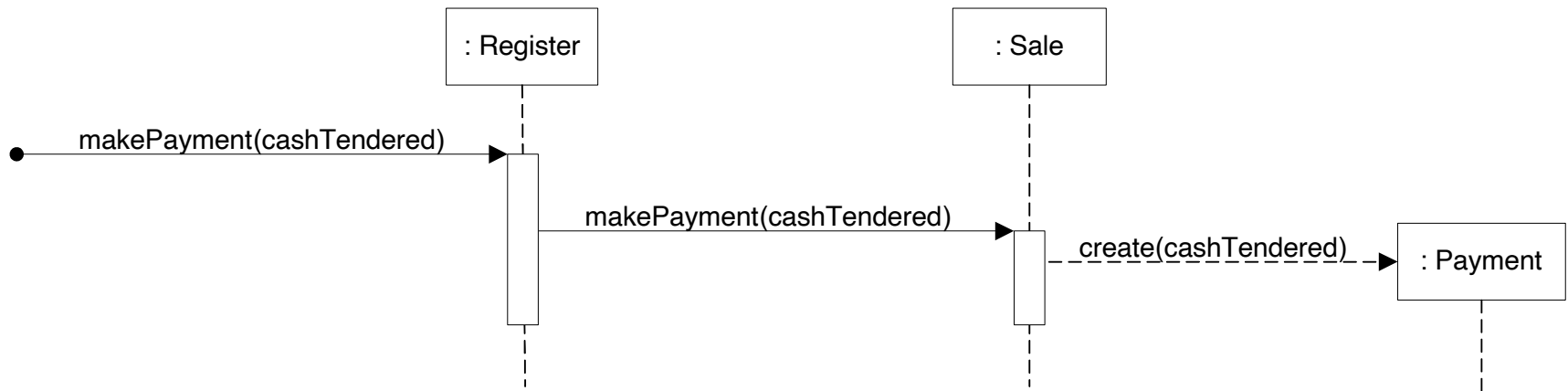
Example: Communication Diagram

doOne →

: A

1: doTwo

2: doThree

myB : B

Strengths and Weaknesses of Sequence and Communication Diagrams

| Type	Strengths	Weaknesses
sequence	clearly shows sequence or time ordering of messages large set of detailed notation options	forced to extend to the right when adding new objects; consumes horizontal space
communication	space economicalflexibility to add new objects in two dimensions	more difficult to see sequence of messages fewer notation options

Example Sequence Diagram


```
public class Sale {
  private Payment payment;

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

Example communication diagram
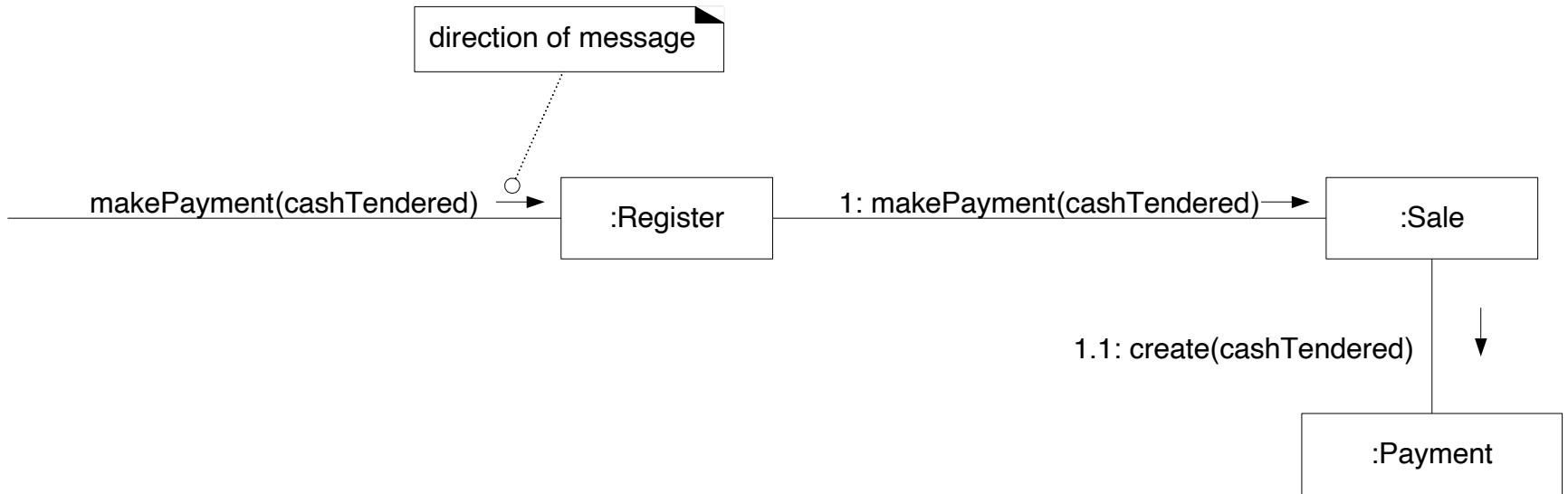
direction of message

:Register

makePayment(cashTendered)

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*

:Sale

lifeline box representing a named instance

s1 : Sale

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

«metaclass»
Font

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

sales:
ArrayList<Sale>

related example

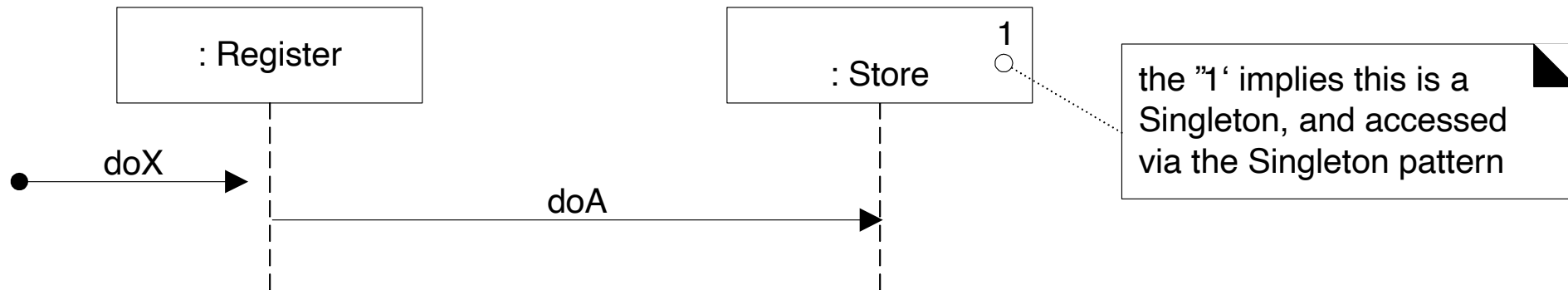lifeline box representing one instance of class *Sale*, selected from the *sales ArrayList <Sale>* collection

sales[i] : Sale

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

x : List

The Singleton Pattern

```
┌──────────────────┐          ┌──────────────────── 1
│    : Register    │          │     : Store      ○ ┈┈┈┈┈┐
└──────────────────┘          └────────────────────┘    ┆   ┌─────────────────────────┐
         ┆                              ┆                └┈┈ │ the "1' implies this is a │
  doX    ┆                              ┆                    │ Singleton, and accessed │
●───────▶┆              doA             ┆                    │ via the Singleton pattern │
         ┆─────────────────────────────▶┆                   └─────────────────────────┘
         ┆                              ┆
         ┆                              ┆
```
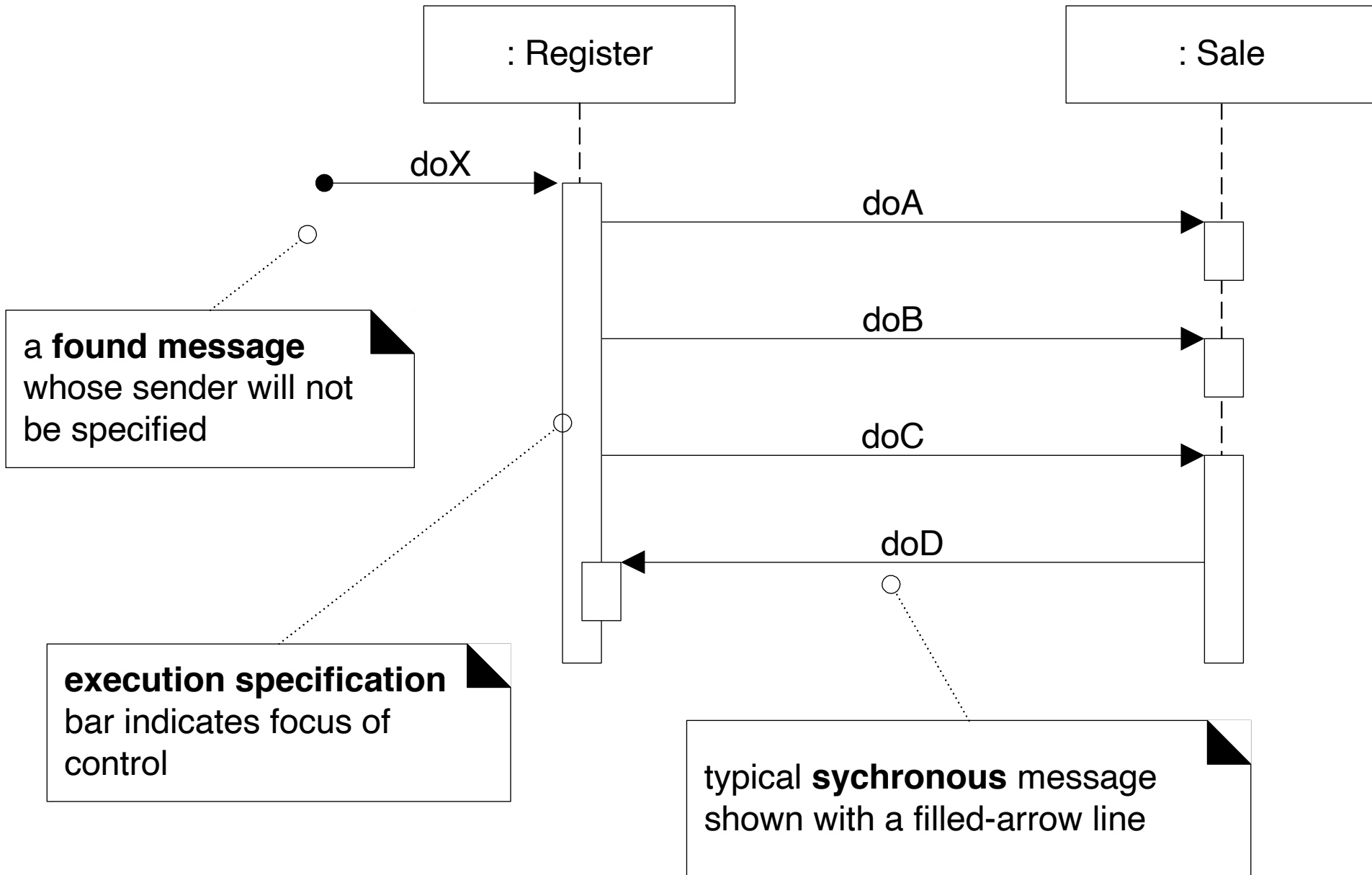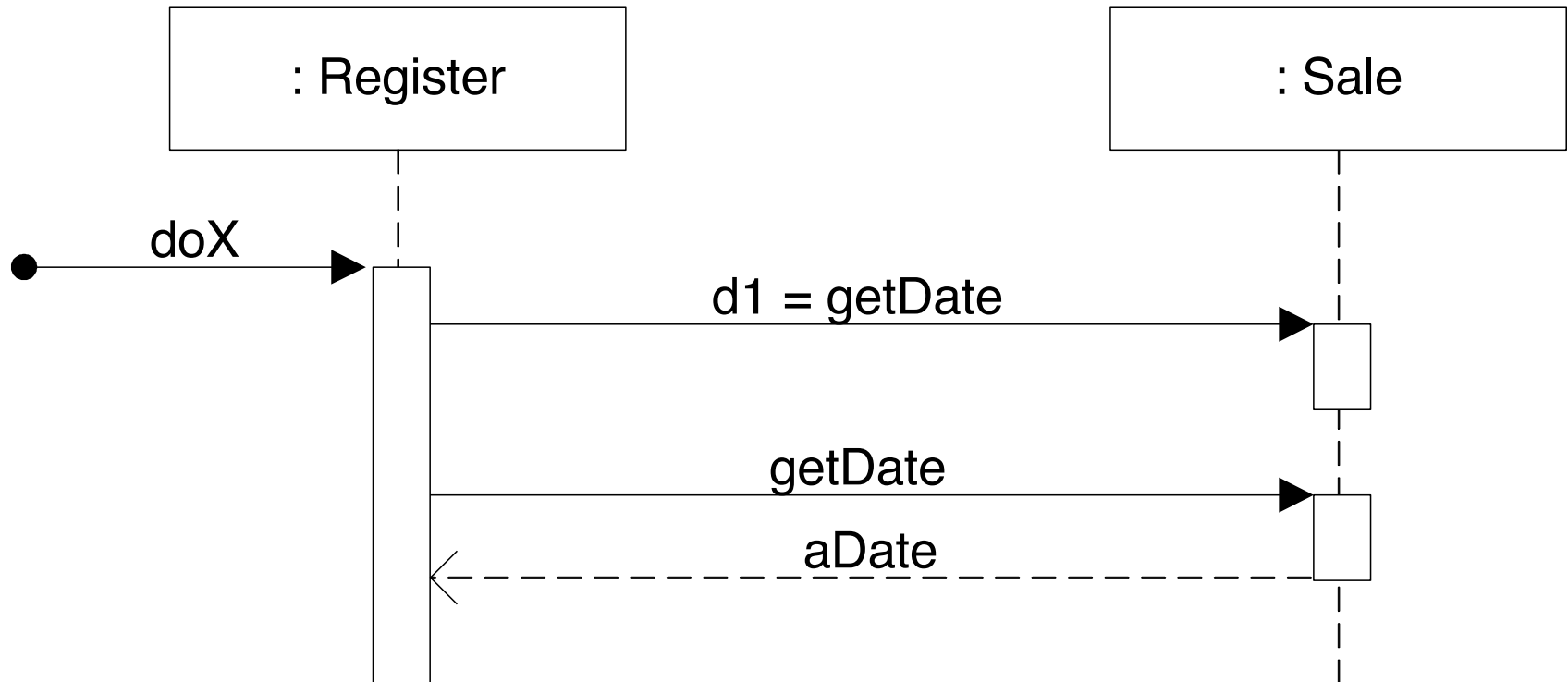
- Used if we want only ONE instance of a class instantiated
  - Examples: Database, log
- We'll learn how to accomplish this in Java later

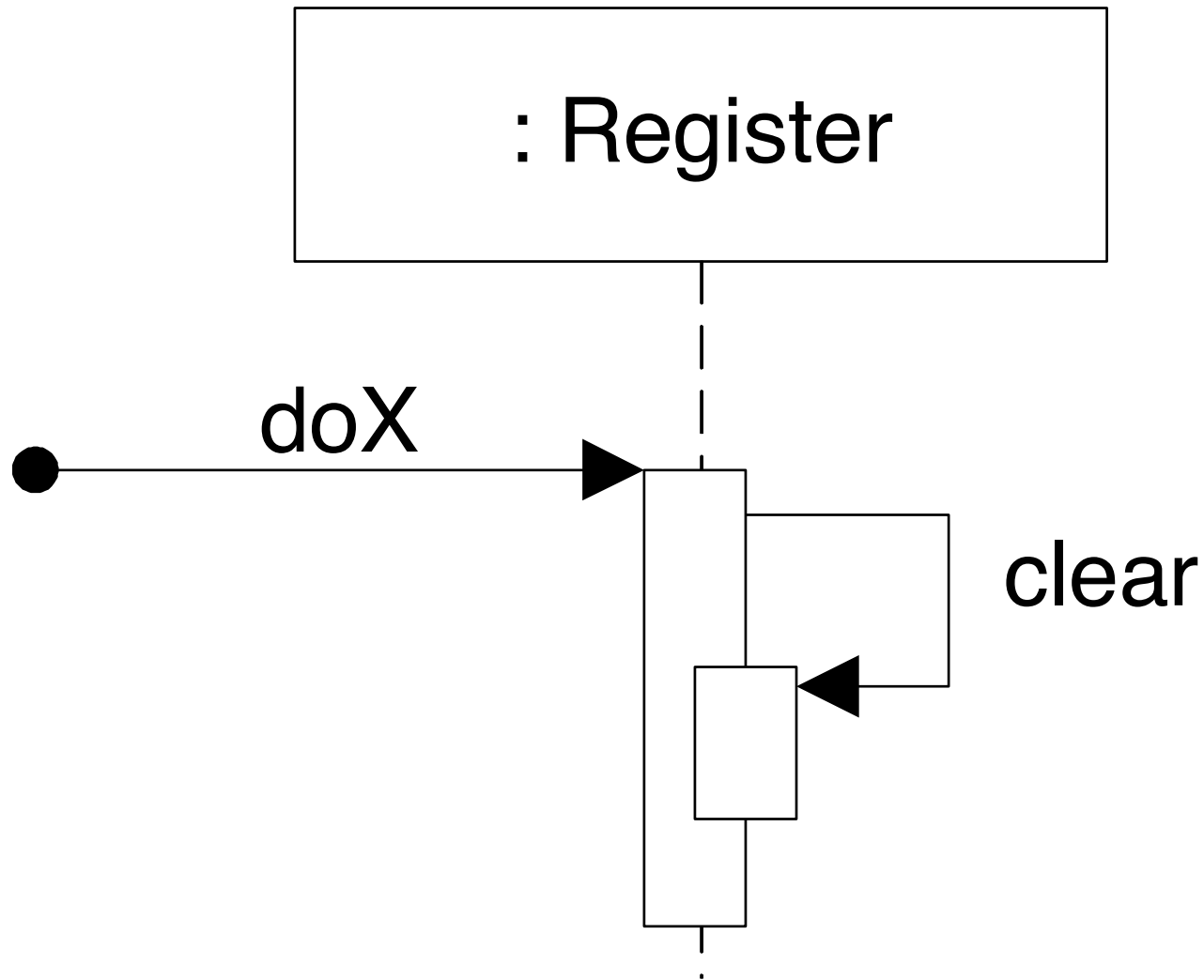# Lifelines, messages, "found" or "starting" messages

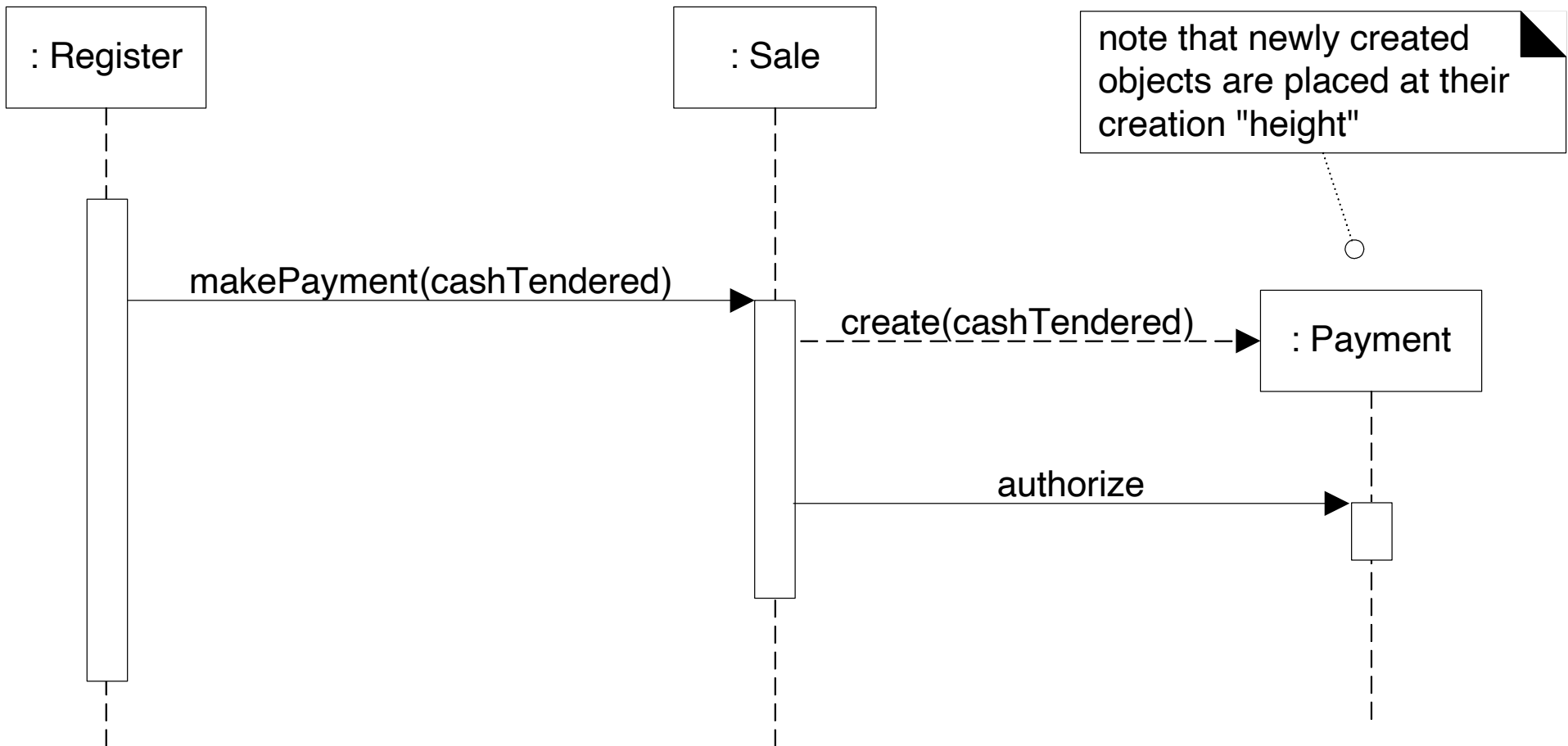# Illustrating Replies or Return Values

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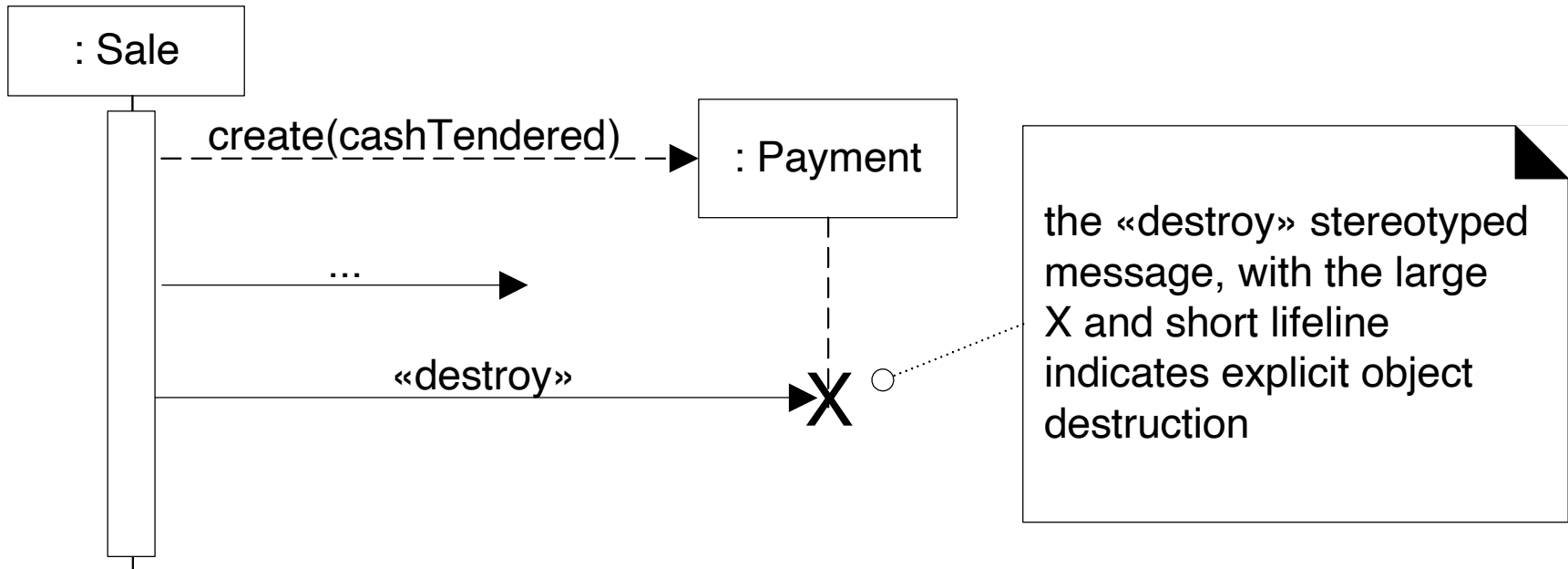 - Use a reply or return message at the end of an activation bar
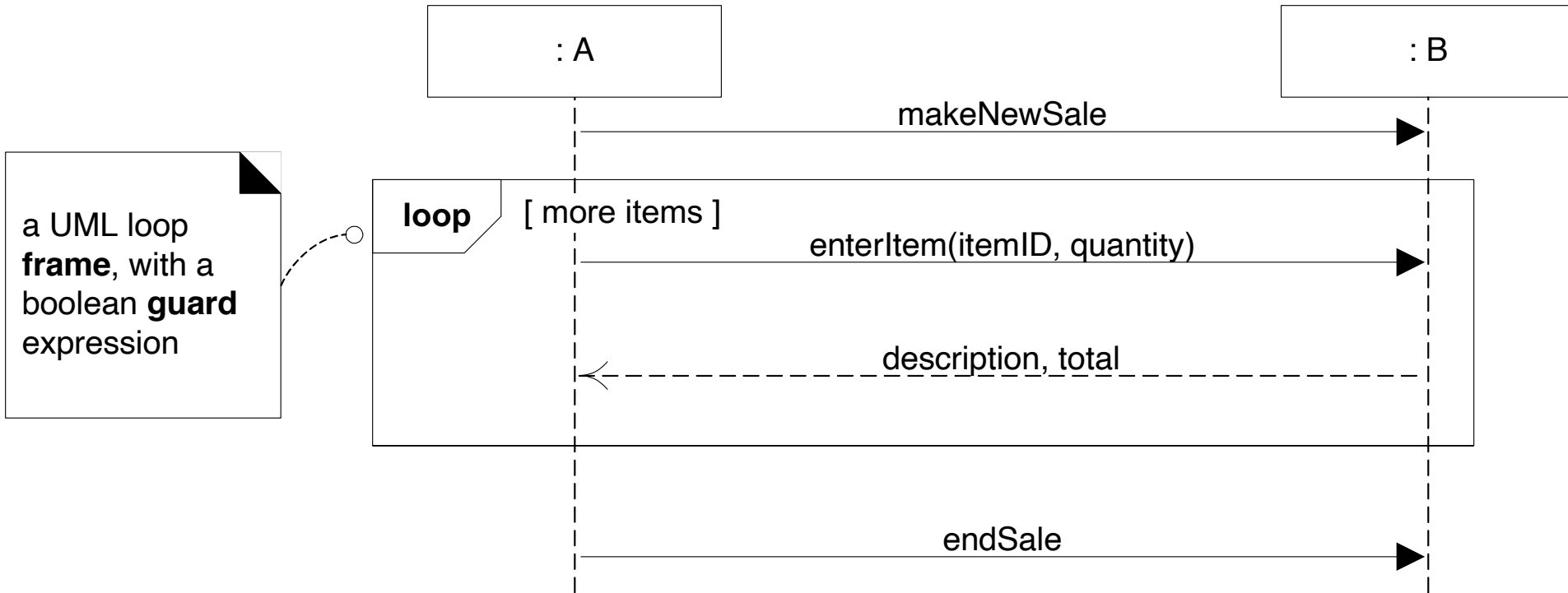
Messages to "self" or "this"

: Register

doX

clear

Instance Creation

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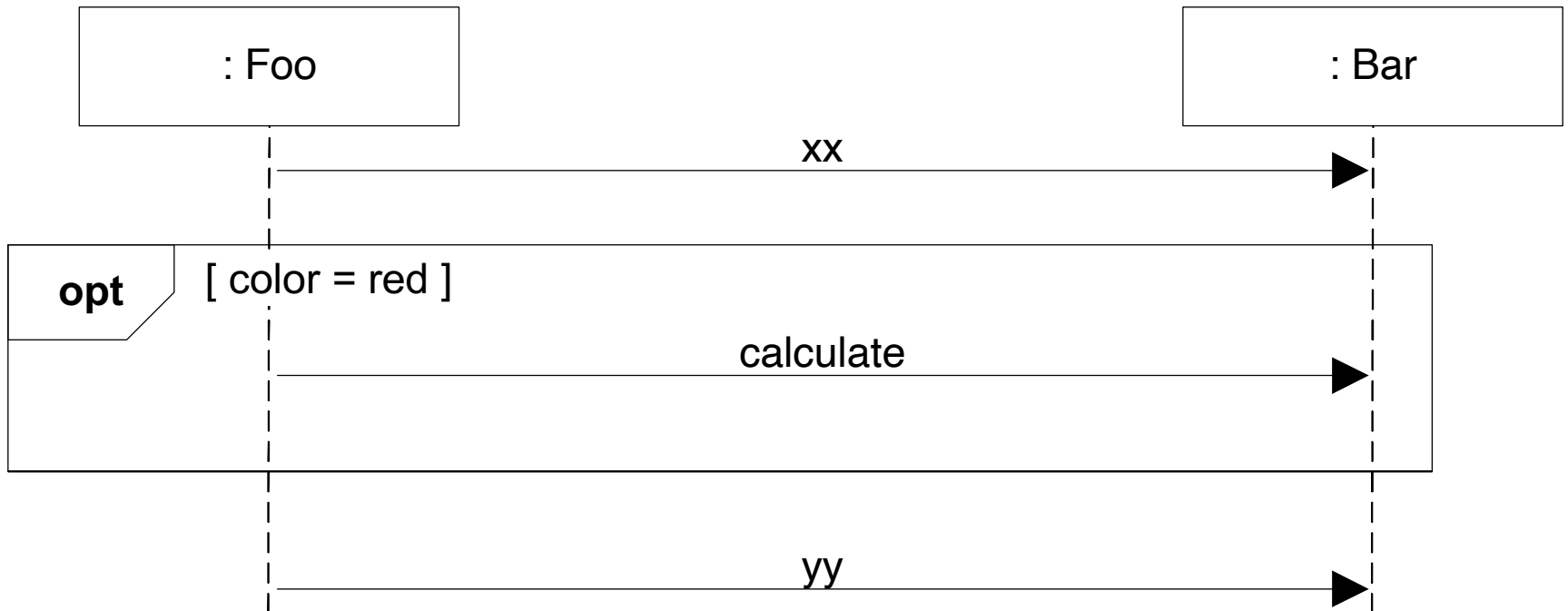
- Object explicitly destroyed or no longer usable (reachable)
  - Example: No variable refers to object any longer
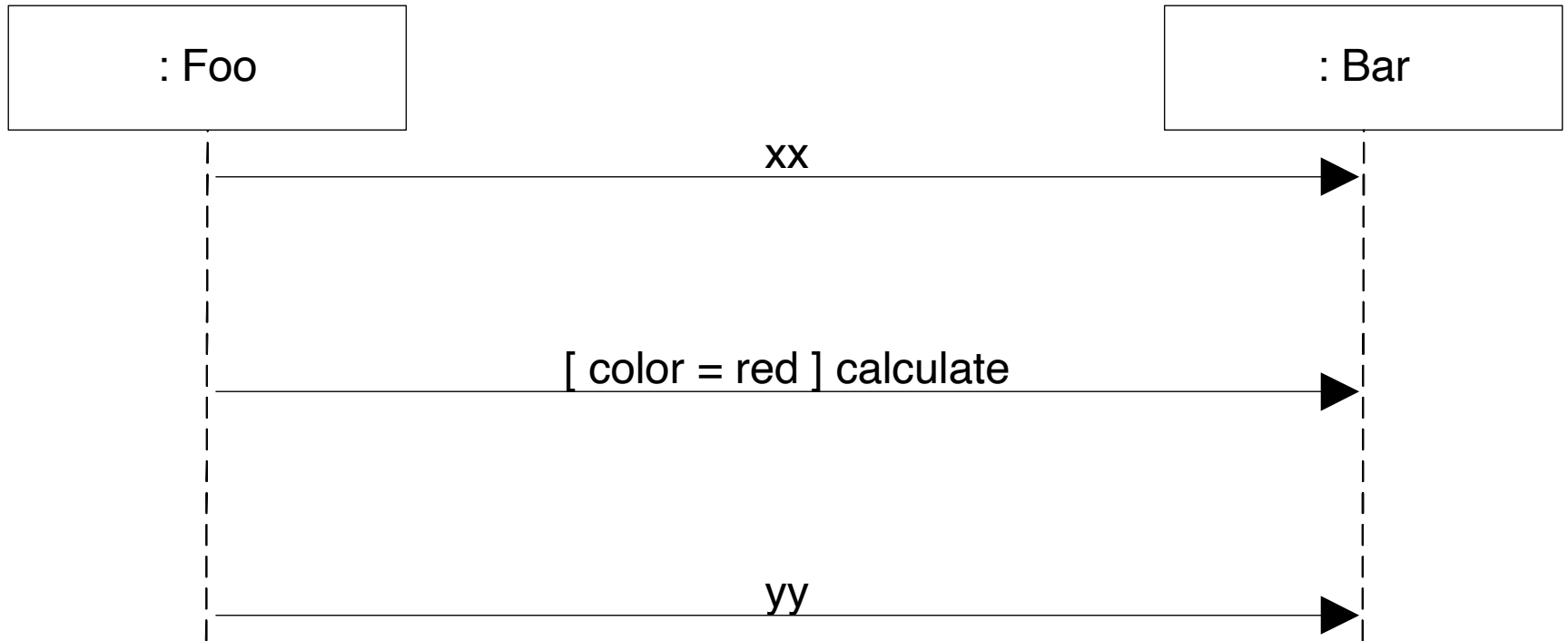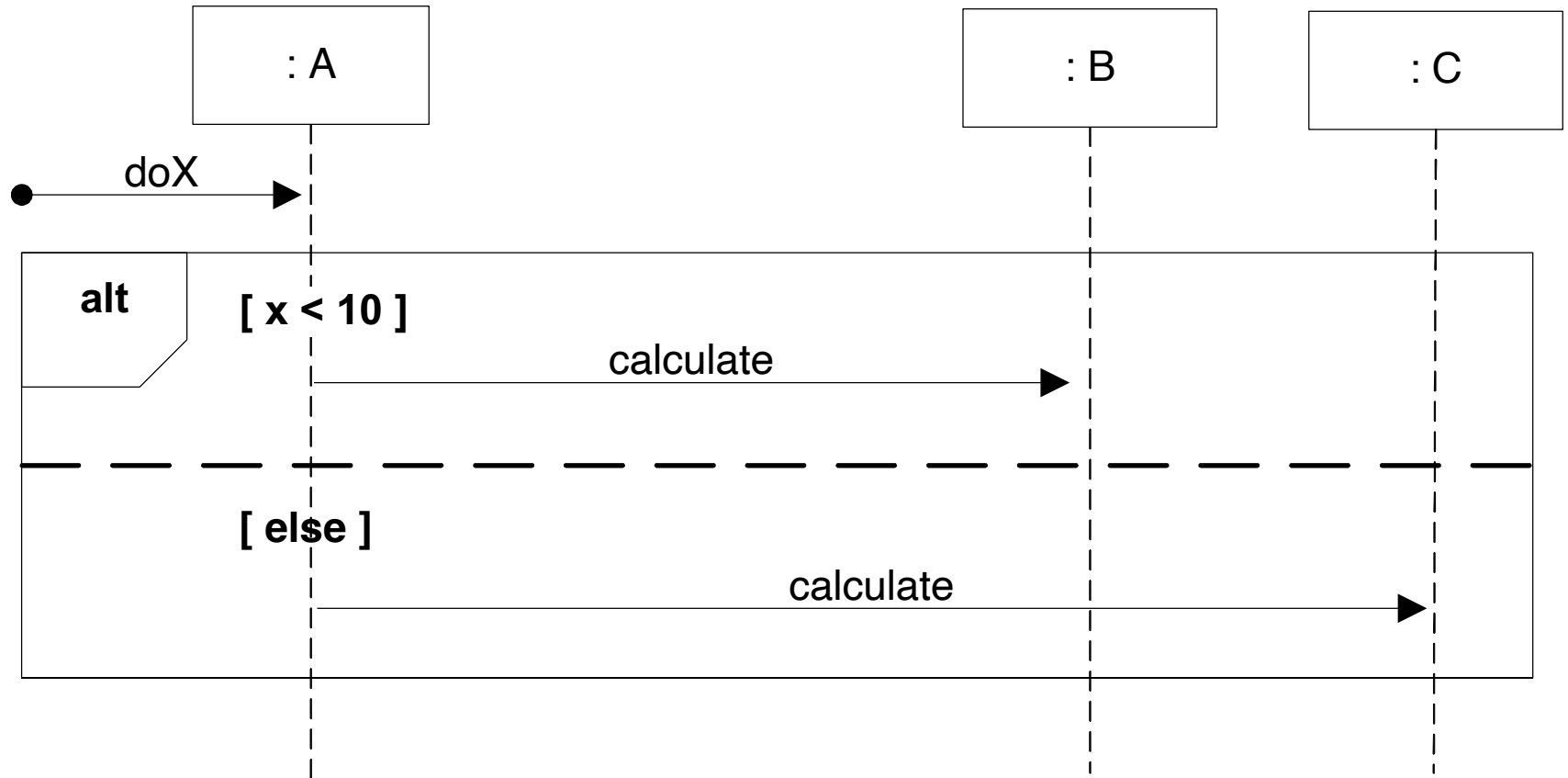   - Marked for garbage collection

Looping notation

: A

: B

makeNewSale

loop [more items]

enterItem(itemID, quantity)

description, total

a UML loop **frame**, with a boolean **guard** expression

endSale

A conditional message

UML Version 1 notation for conditional messages

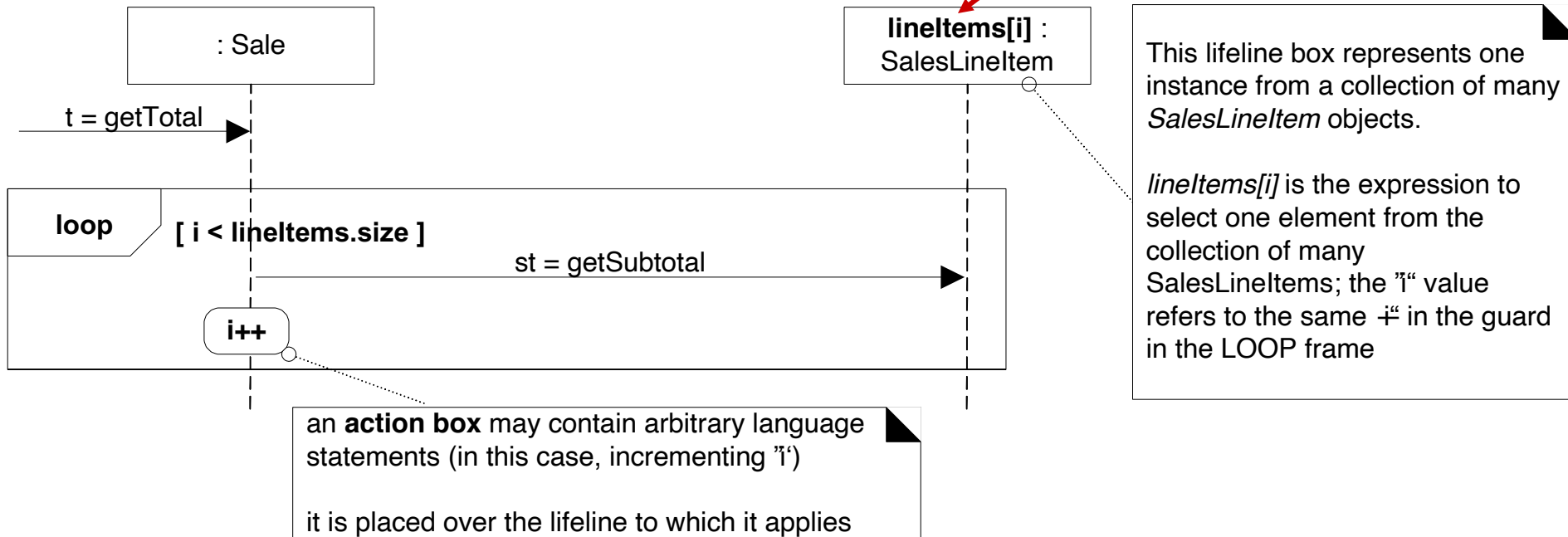| : Foo		: Bar

xx

[color = red] calculate

yy

Mutually exclusive conditional messages

Iteration over a collection: Explicit notation

Selector expression

lineItems[i] :
SalesLineItem

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

: Sale

t = getTotal

loop   **[i < lineItems.size]**

st = getSubtotal

i++

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

: Foo

: Bar

xx

opt [color = red]

loop(n)

calculate
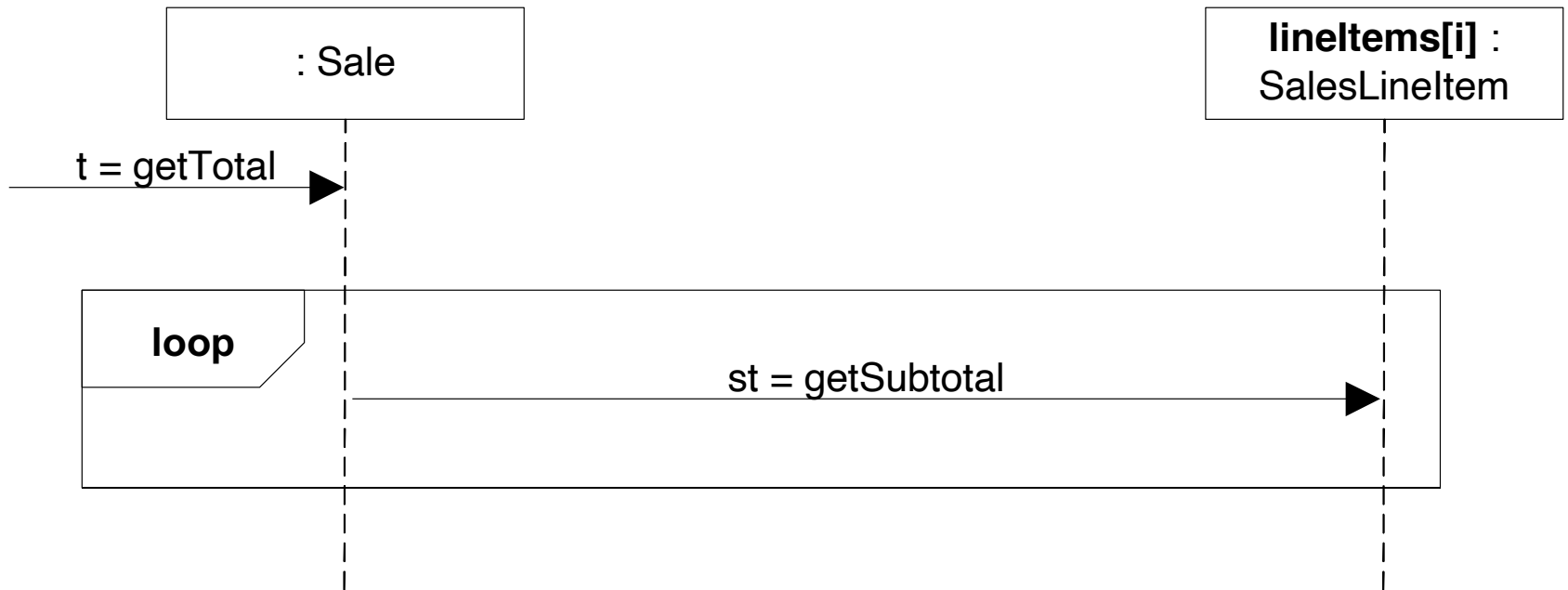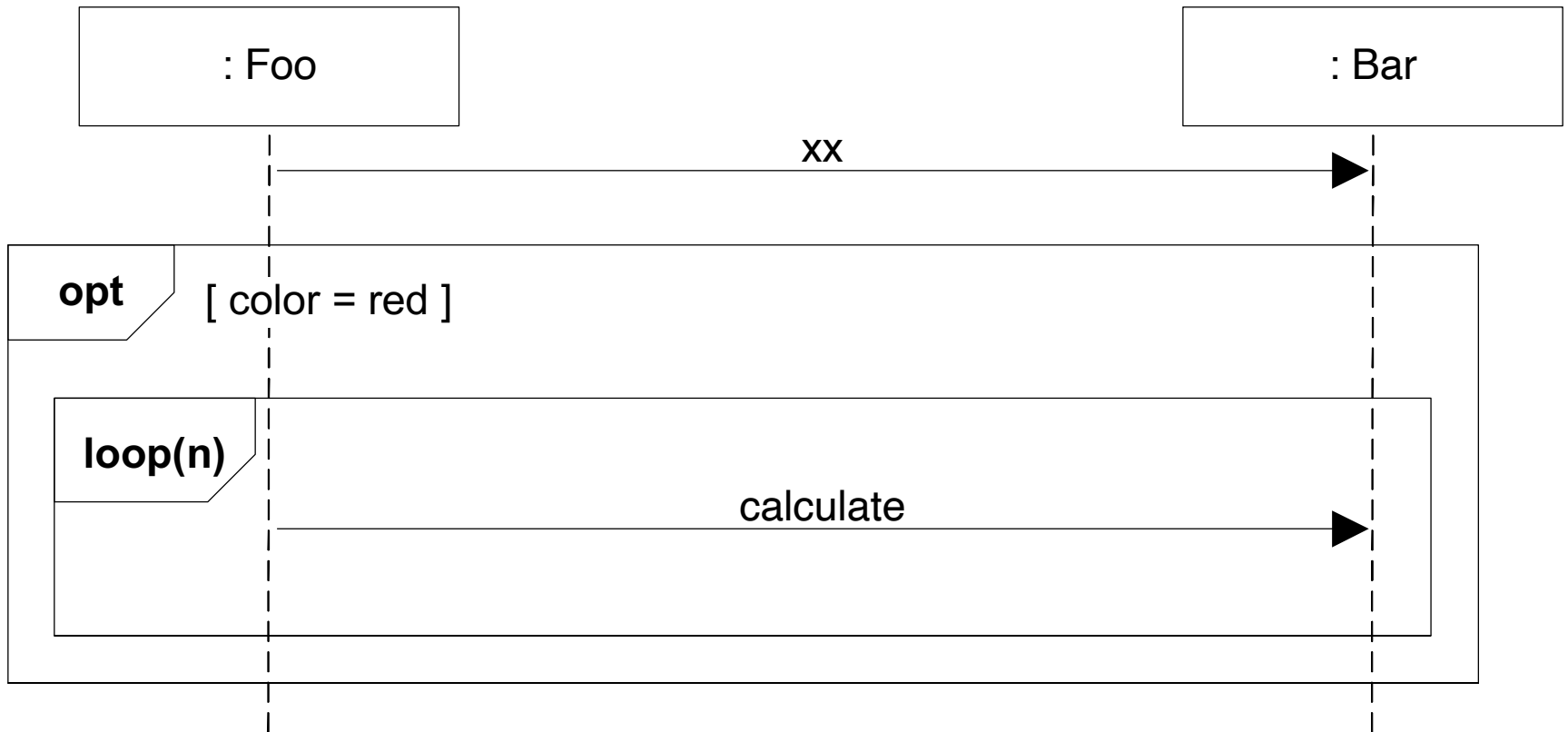
How to relate interaction diagrams (hierarchical notation)
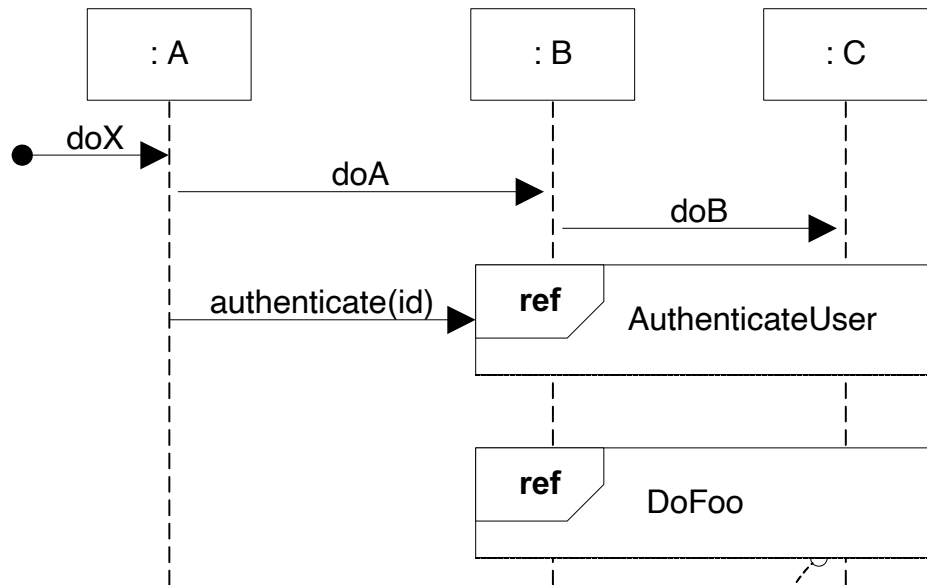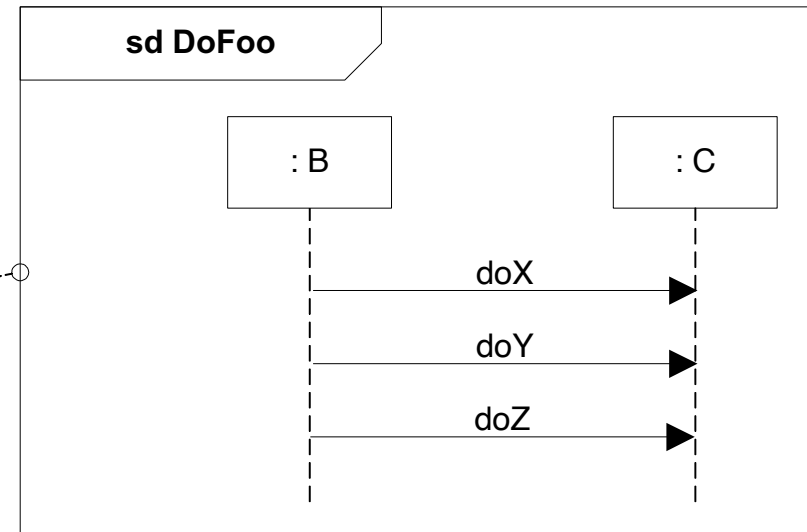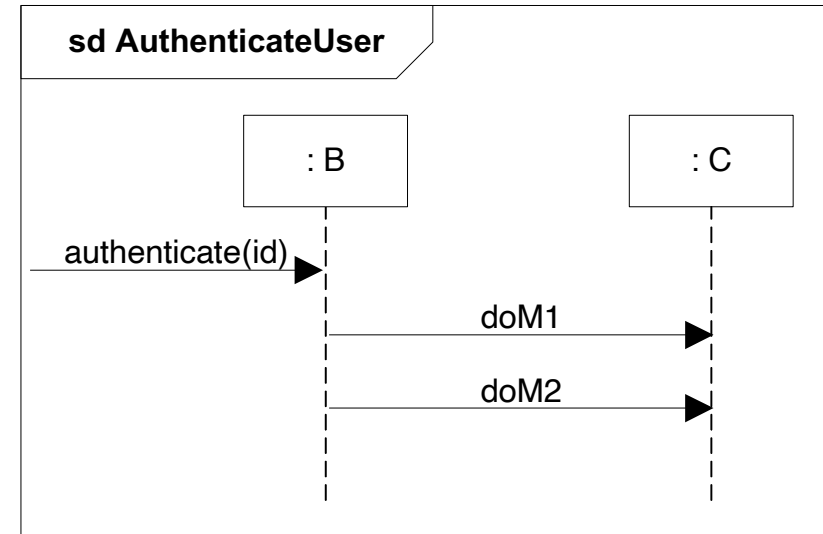
Invoking static or class methods
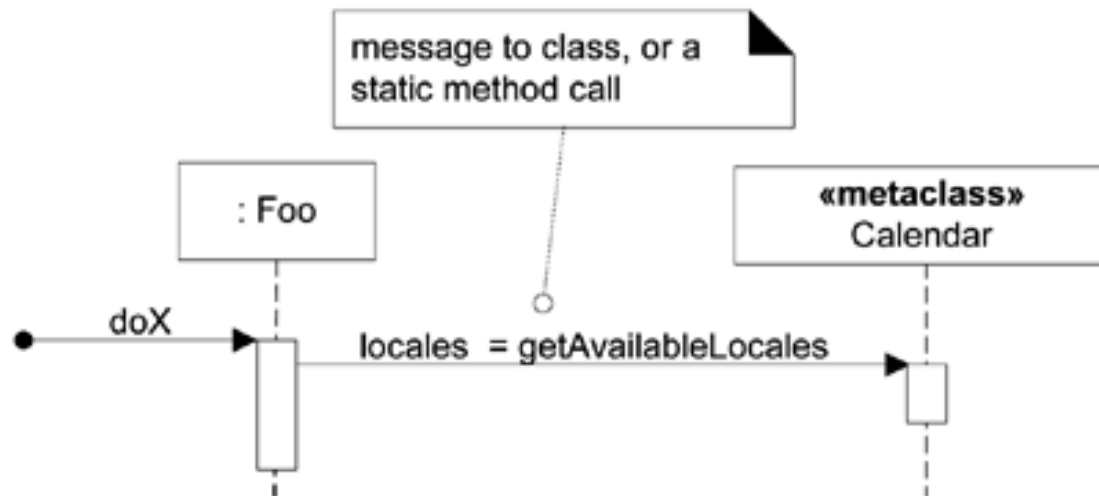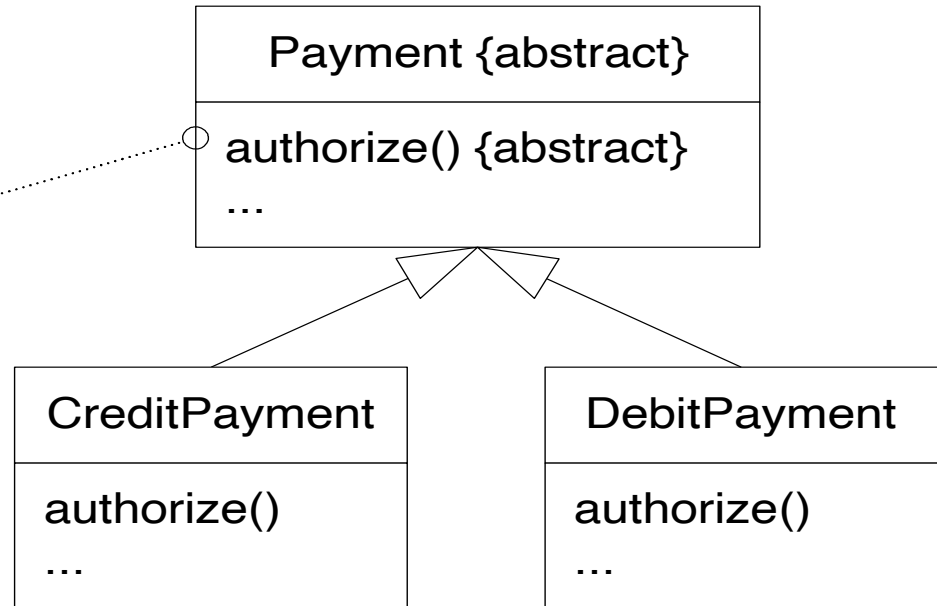


```
public class Foo {

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

Polymorphic Messages and Cases

Payment {abstract}

authorize() {abstract}
...

Payment is an abstract superclass, with concrete subclasses that implement the polymorphic authorize operation

CreditPayment

authorize()
...

DebitPayment

authorize()
...

Polymorphic Messages and Cases
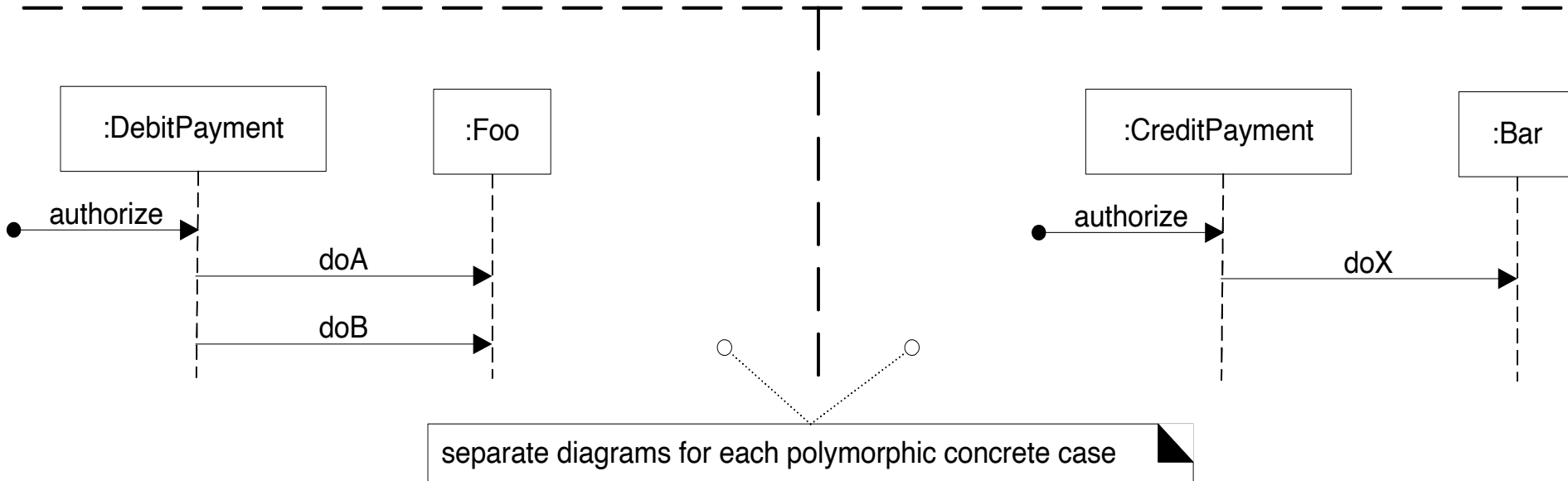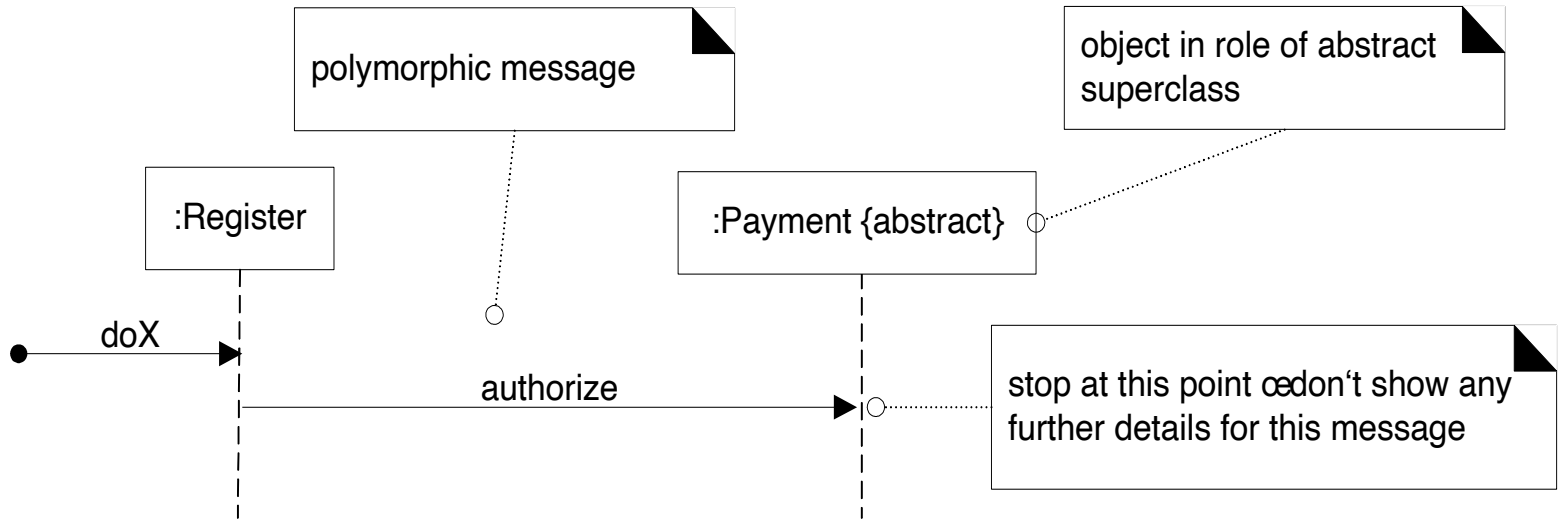
Asynchronous vs. Synchronous Calls

- 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

:ClockStarter

System : Class

startClock

create   :Clock

run

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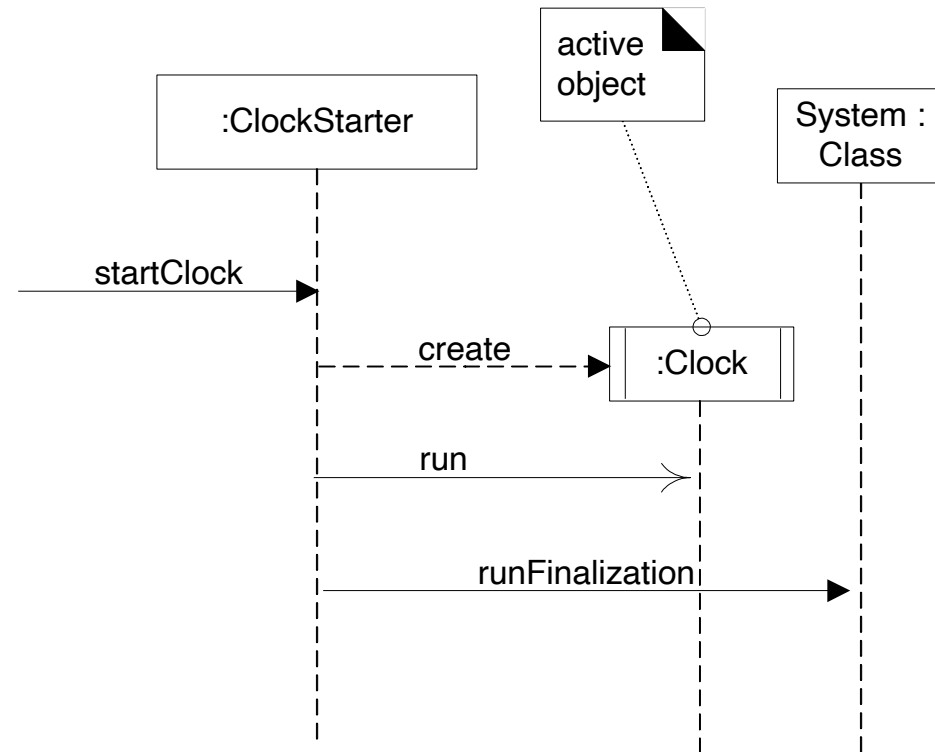
- Active object: Each instance runs on and controls its own thread of execution
  - Example: Clock

Communication Diagram Notation

1: makePayment(cashTendered) →
2: foo →

: Register ——————○ :Sale
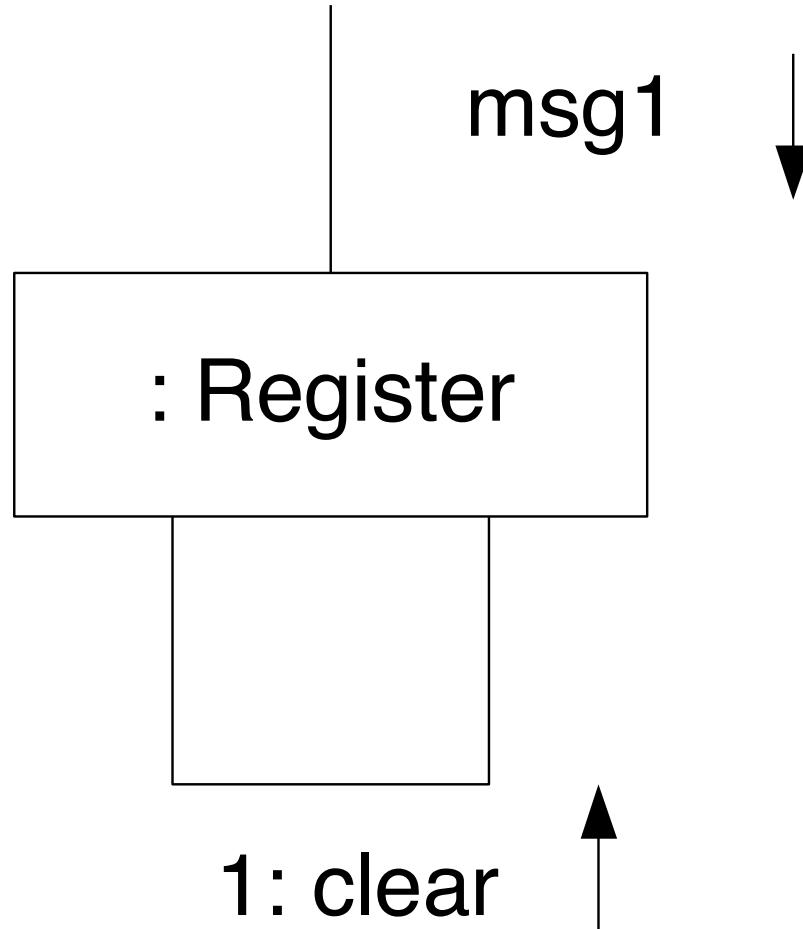
2.1: bar ←

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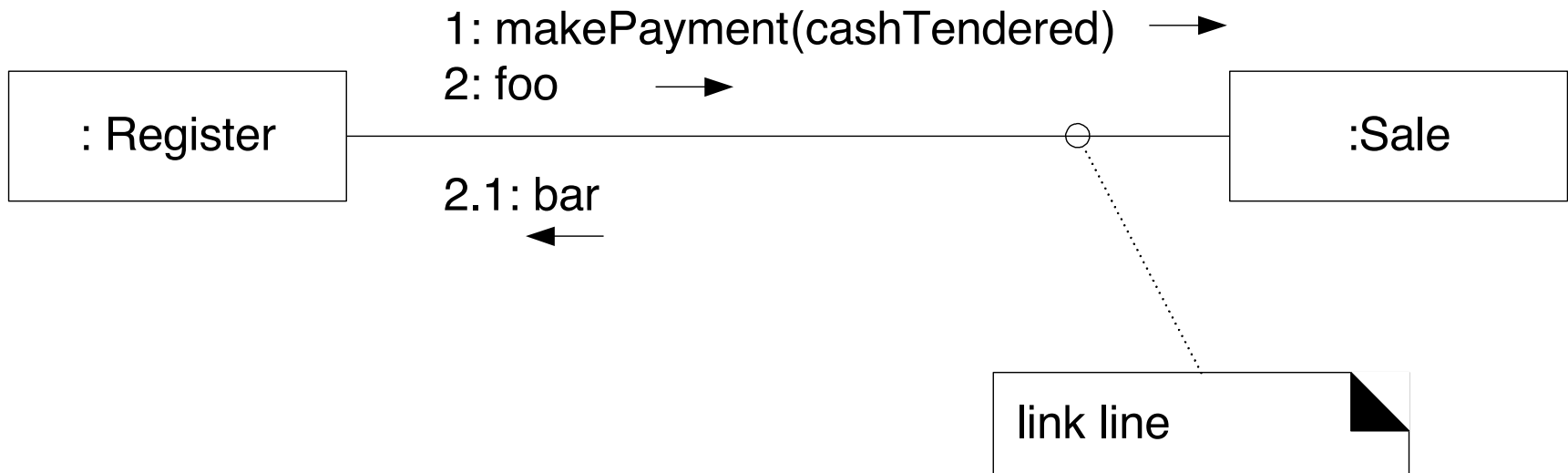
- Link: Connection path between two objects
    - Indicates a form of navigation or visibility between the objects
    - Formally: An instance of an association
- There can be only one link between two objects
    - Multiple messages in both directions flow along this link
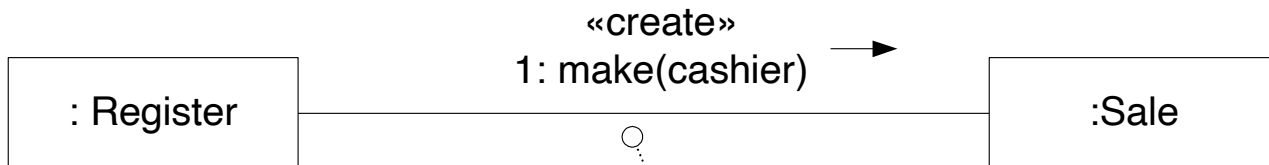
Communication Diagram Notation

msg1 ↓

: Register

1: msg2 →
2: msg3 →
3: msg4 →

3.1: msg5 ←

:Sale
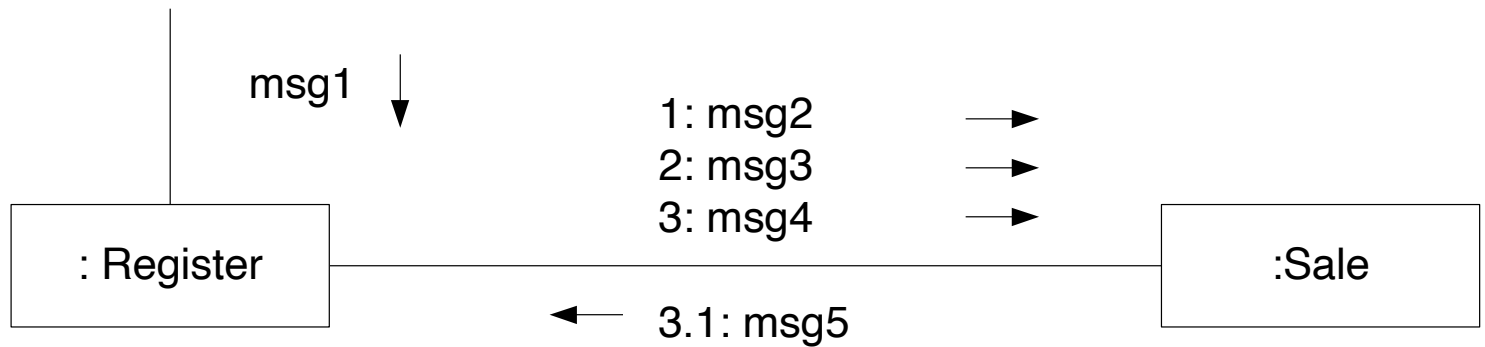
all messages flow on the same link

Messages to "self" or "this"

msg1

↓

: Register

1: clear ↑

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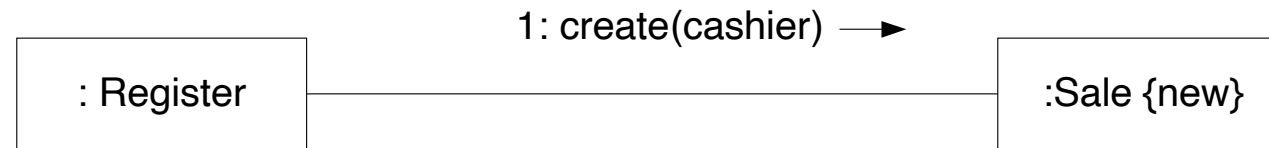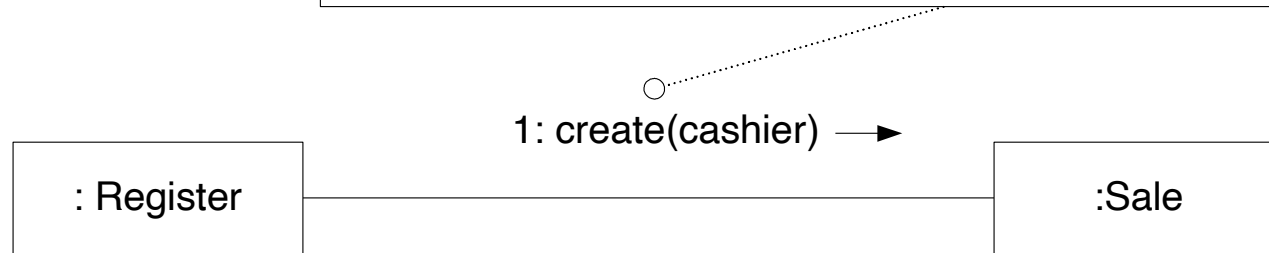
if an unobvious creation message name is used, the message may be stereotyped for clarity

Sequence Numbering

msg1 →

: A

1: msg2 →

: B

not numbered

legal numbering

1.1: msg3

: C

Nested message

Sequence Numbering

msg1 →

| : A |
| :B |

1: msg2 →

1.1: msg3

2.1: msg5

2: msg4 →

| : C |

2.2: msg6

| : D |

Sequence Numbering

first

second

third

msg1 →

| : A |

1: msg2 →

| : B |

1.1: msg3

2.1: msg5

2: msg4 →

| : C |

fourth

fifth

2.2: msg6

sixth

| : D |

Conditional messages

conditional message, with test

message1

: Foo   1 **[color = red]** : calculate   : Bar

Mutually exclusive conditional messages

unconditional after
either msg2 or msg4

1a and 1b are mutually
exclusive conditional paths

: E

2: msg6

msg1

: A

1a [test1] : msg2

: 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

t = getTotal | : Sale | 1 * [i = 1..n]: st = getSubtotal | lineItems[i]: SalesLineItem |

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.

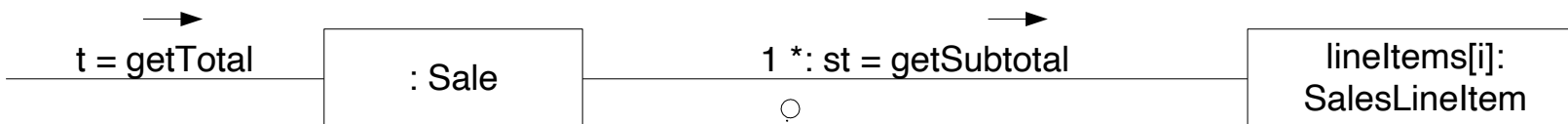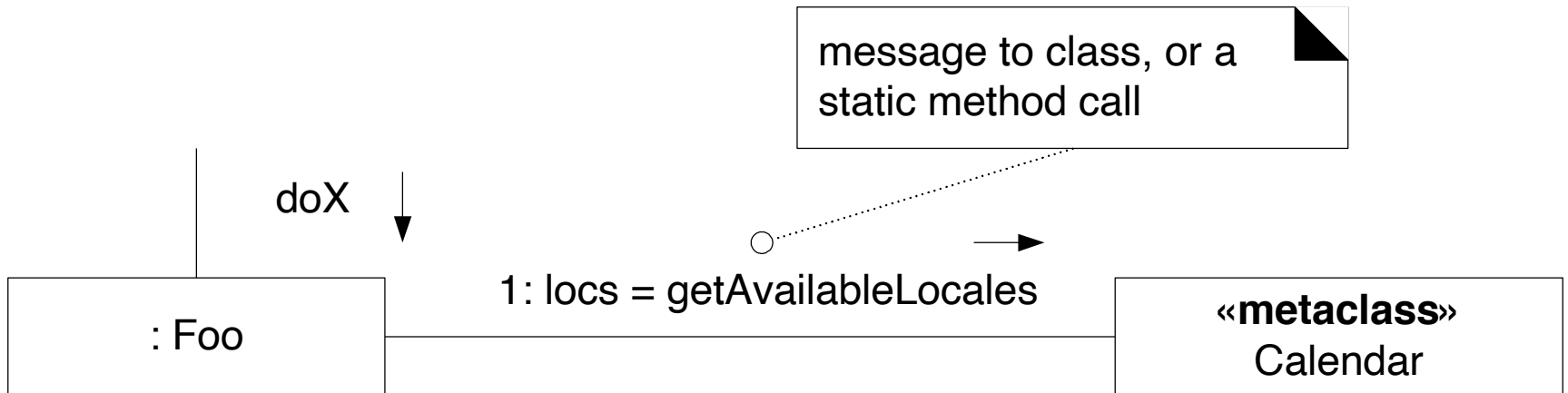*lineItems[i]* is the expression to select one element from the collection of many SalesLineItems; the "i" value comes from the message clause.

t = getTotal | : Sale | 1 *: st = getSubtotal | lineItems[i]: SalesLineItem |

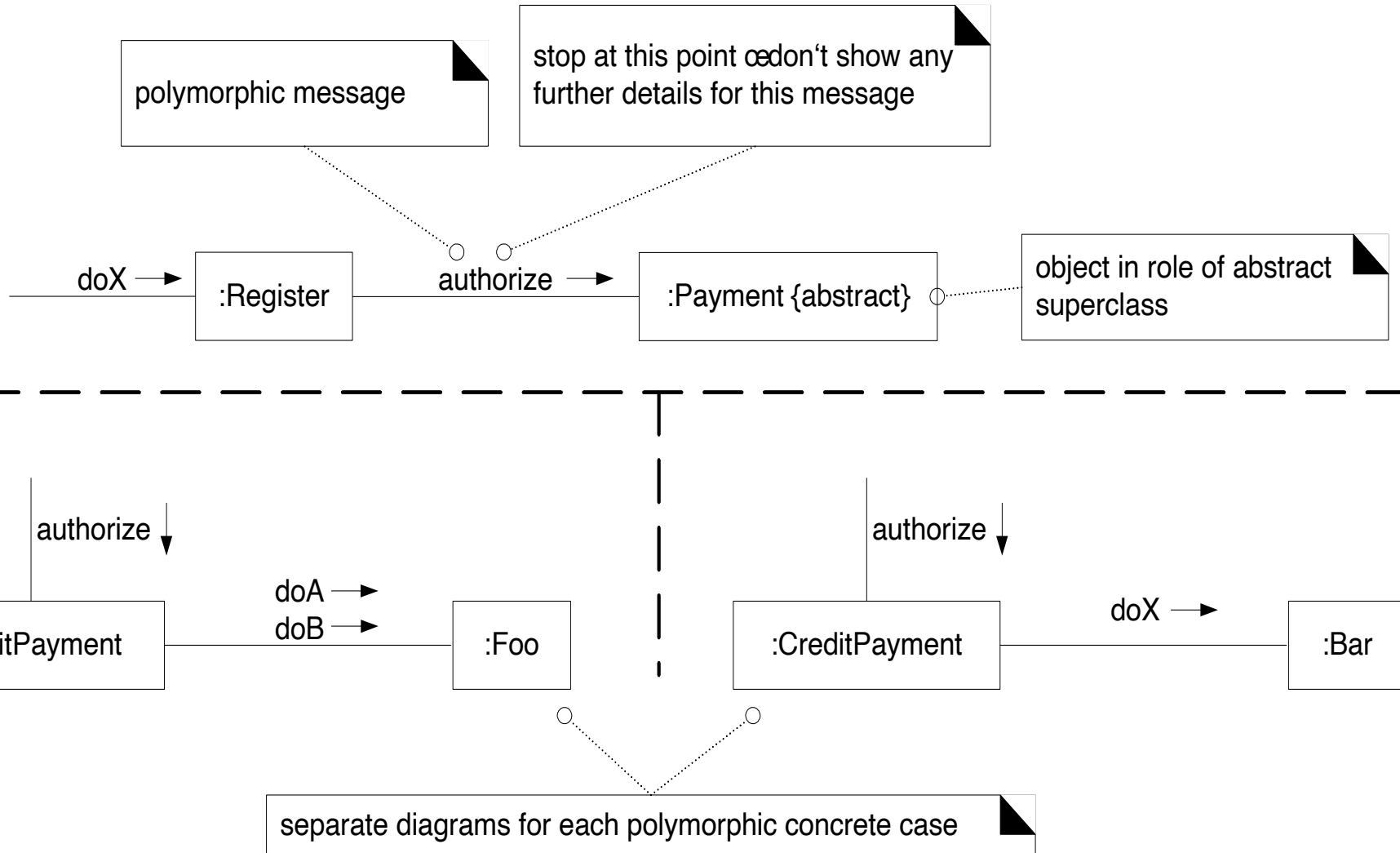Less precise, but usually good enough to imply iteration across the collection members

Static method invocation (message to a class)

message to class, or a static method call

doX

1: locs = getAvailableLocales

: Foo

«**metaclass**»
Calendar

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

:ClockStarter

startClock ↓

3: runFinalization →

System : Class

1: create ↓

asynchronous message

2: run ↓

:Clock

active object