Chapter 15
UML Interaction Diagrams

Larman, C. "Applying UML and Patterns". 3rd Ed.
Ed. Prentice-Hall: 2005.

Fig. 15.1

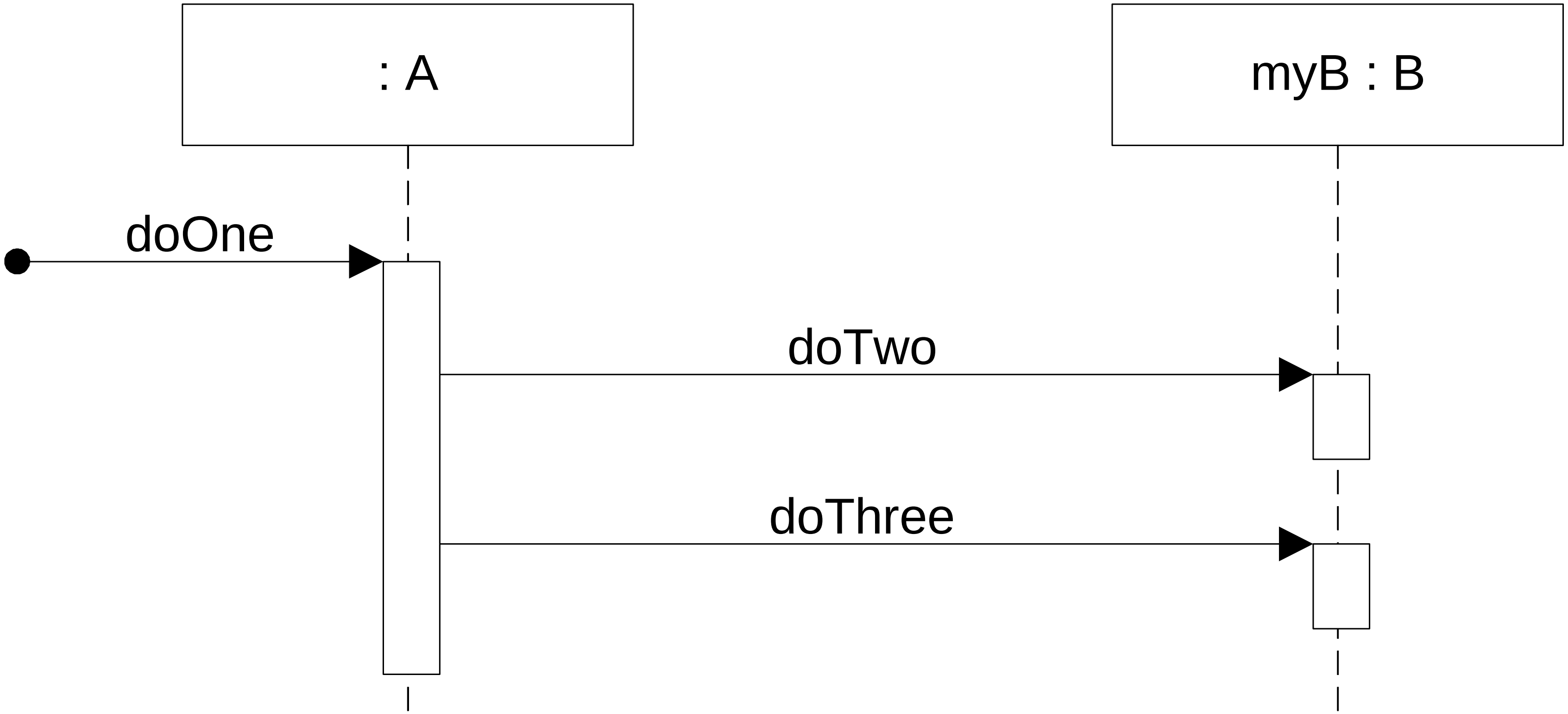


Fig. 15.2

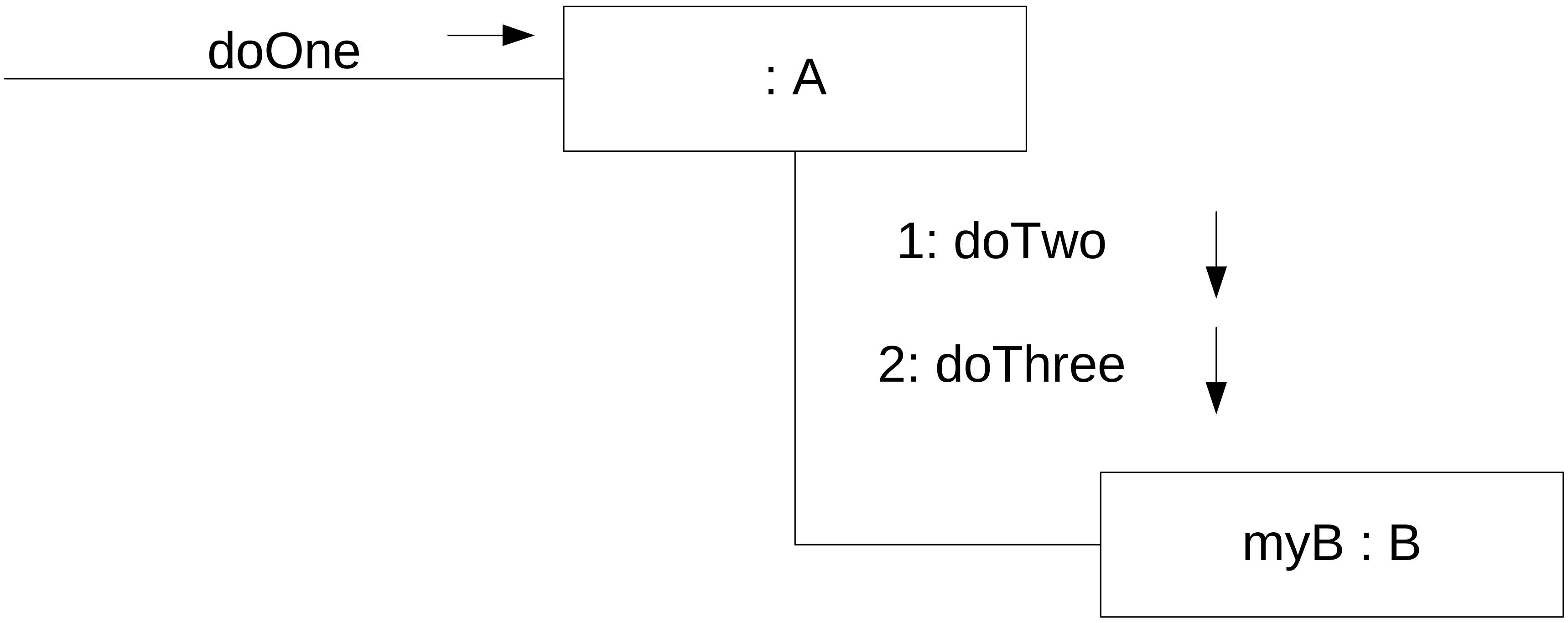
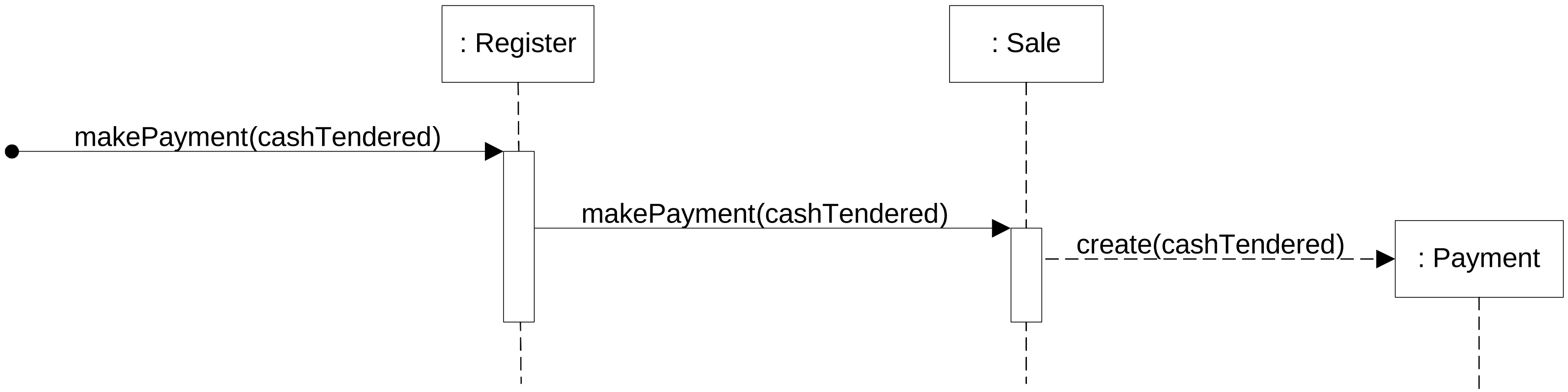
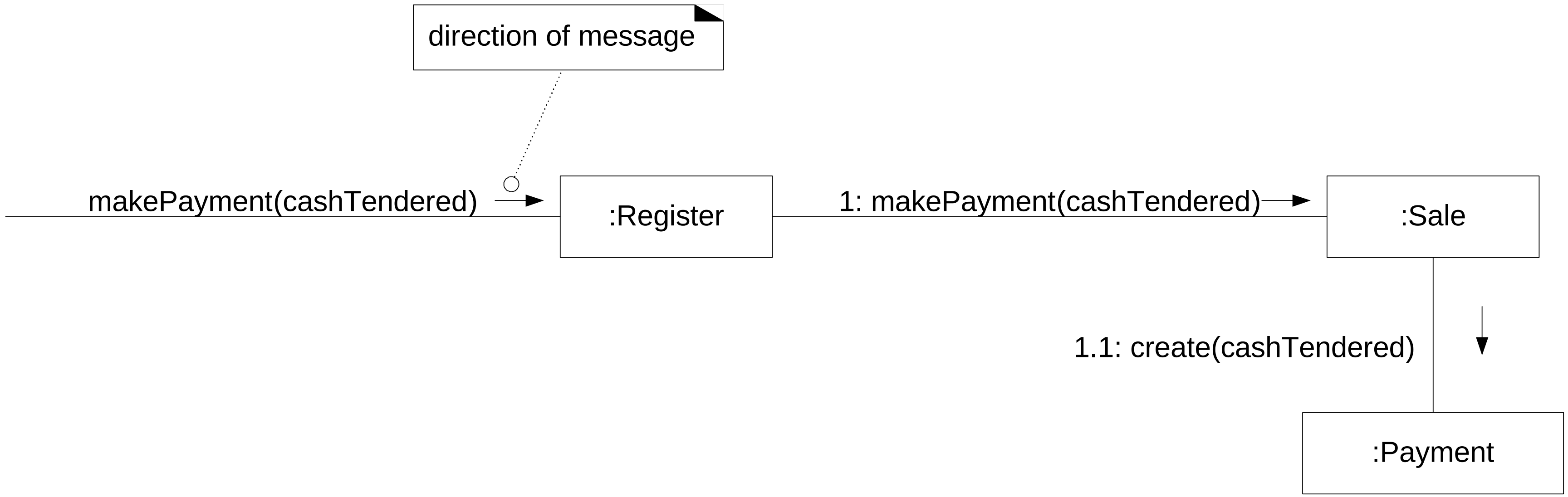


Fig. 15.3



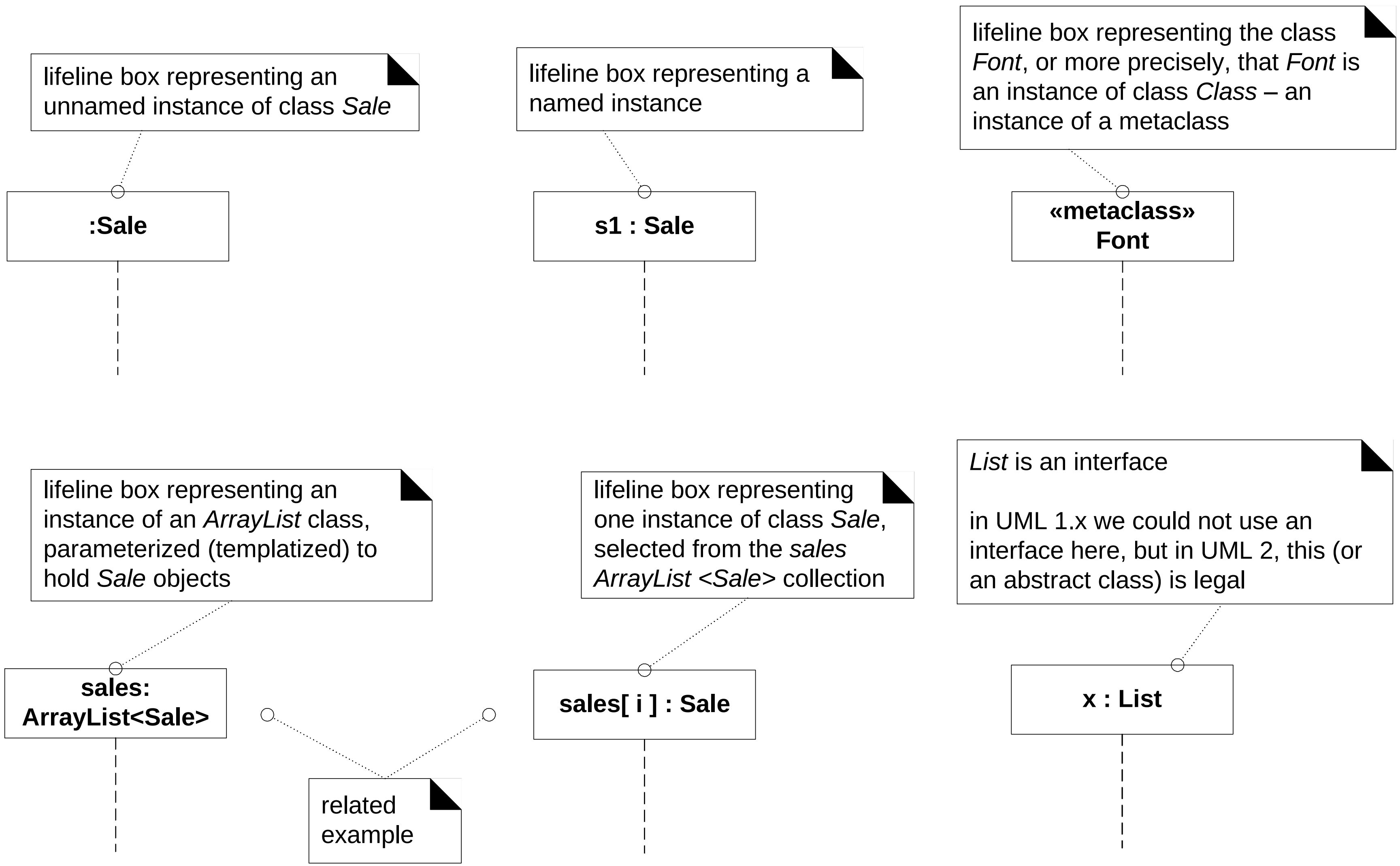
: Register
: Sale
makePayment(cashTendered)
makePayment(cashTendered)
create(cashTendered)
: Payment

Fig. 15.4



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

Fig. 15.5



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* – an 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

Fig. 15.6

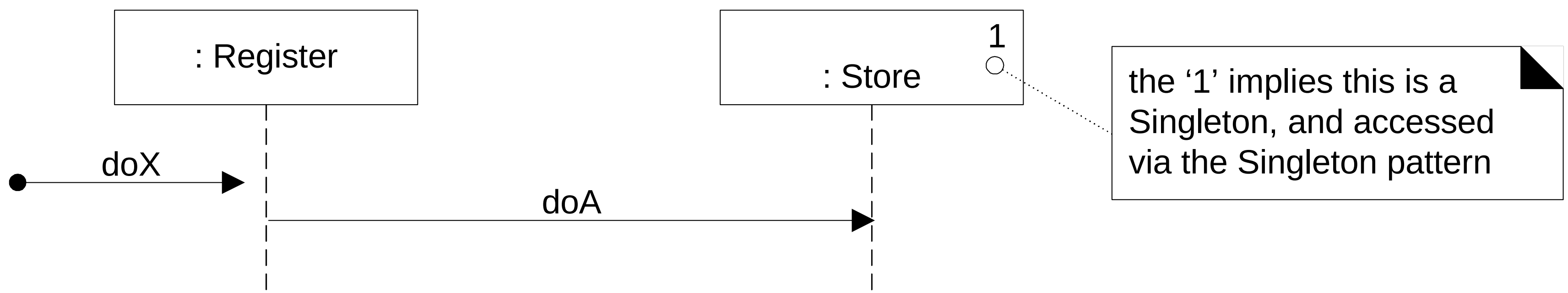
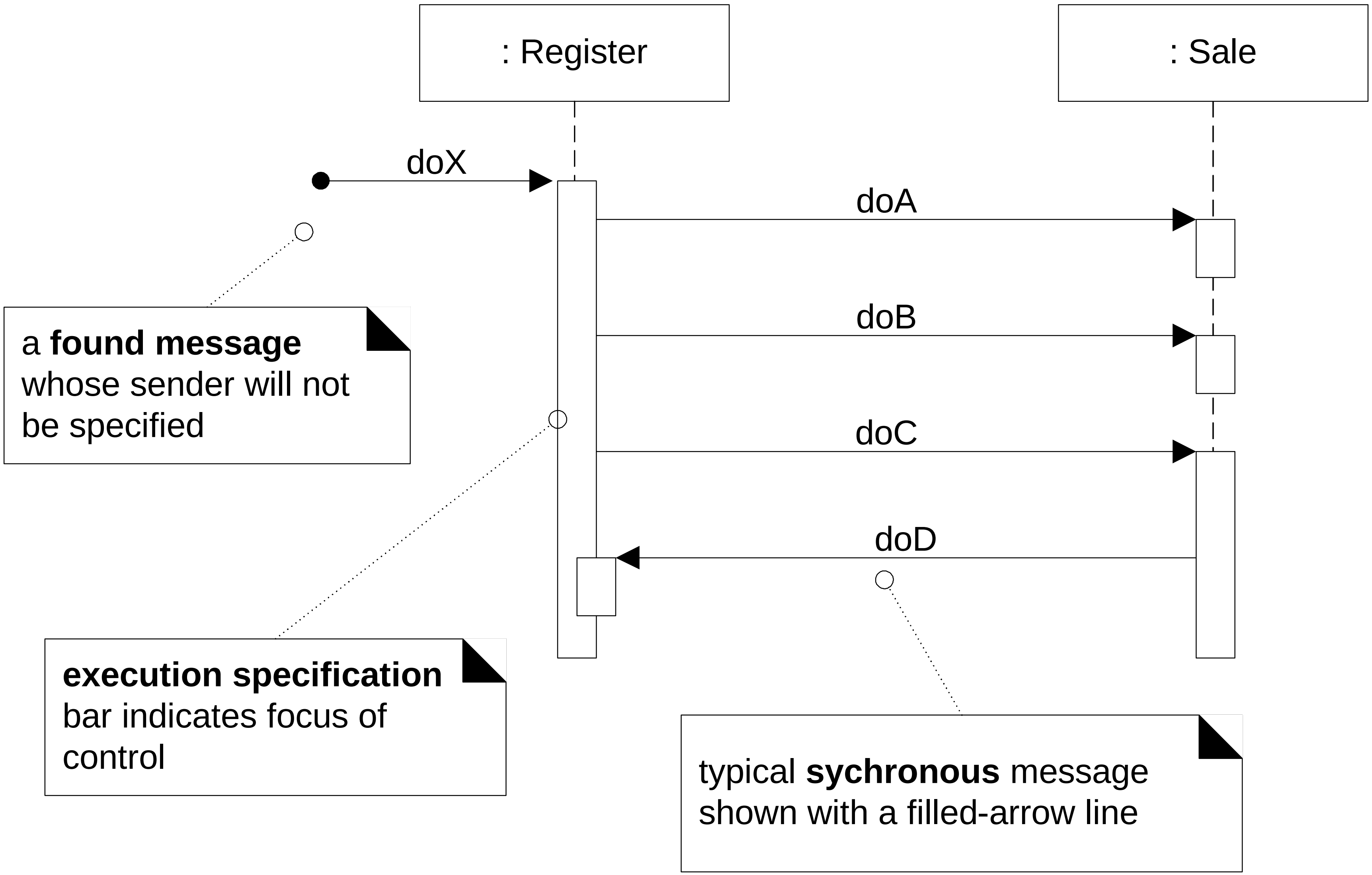

: Register
1
: Store
the '1' implies this is a
Singleton, and accessed
via the Singleton pattern
doX
doA

Fig. 15.7



: 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

Fig. 15.8

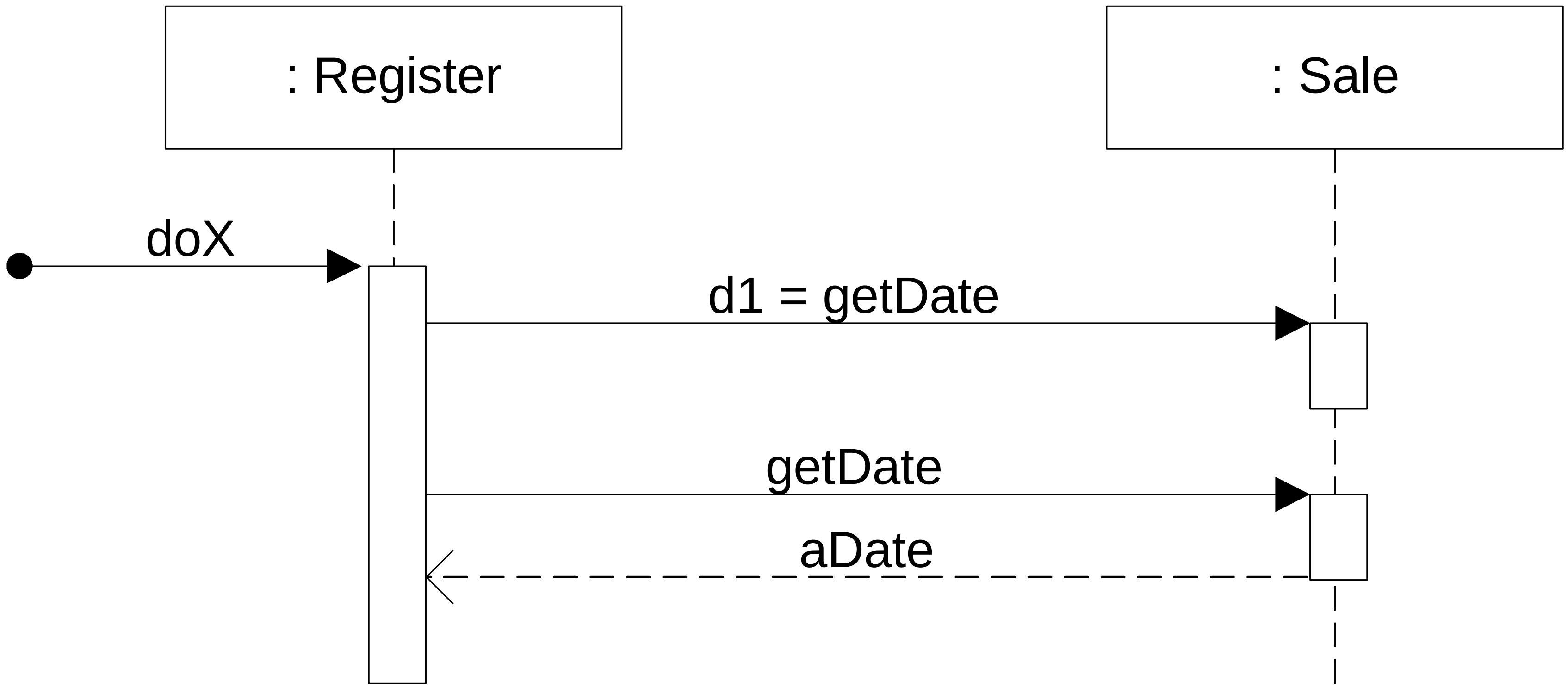


Fig. 15.9

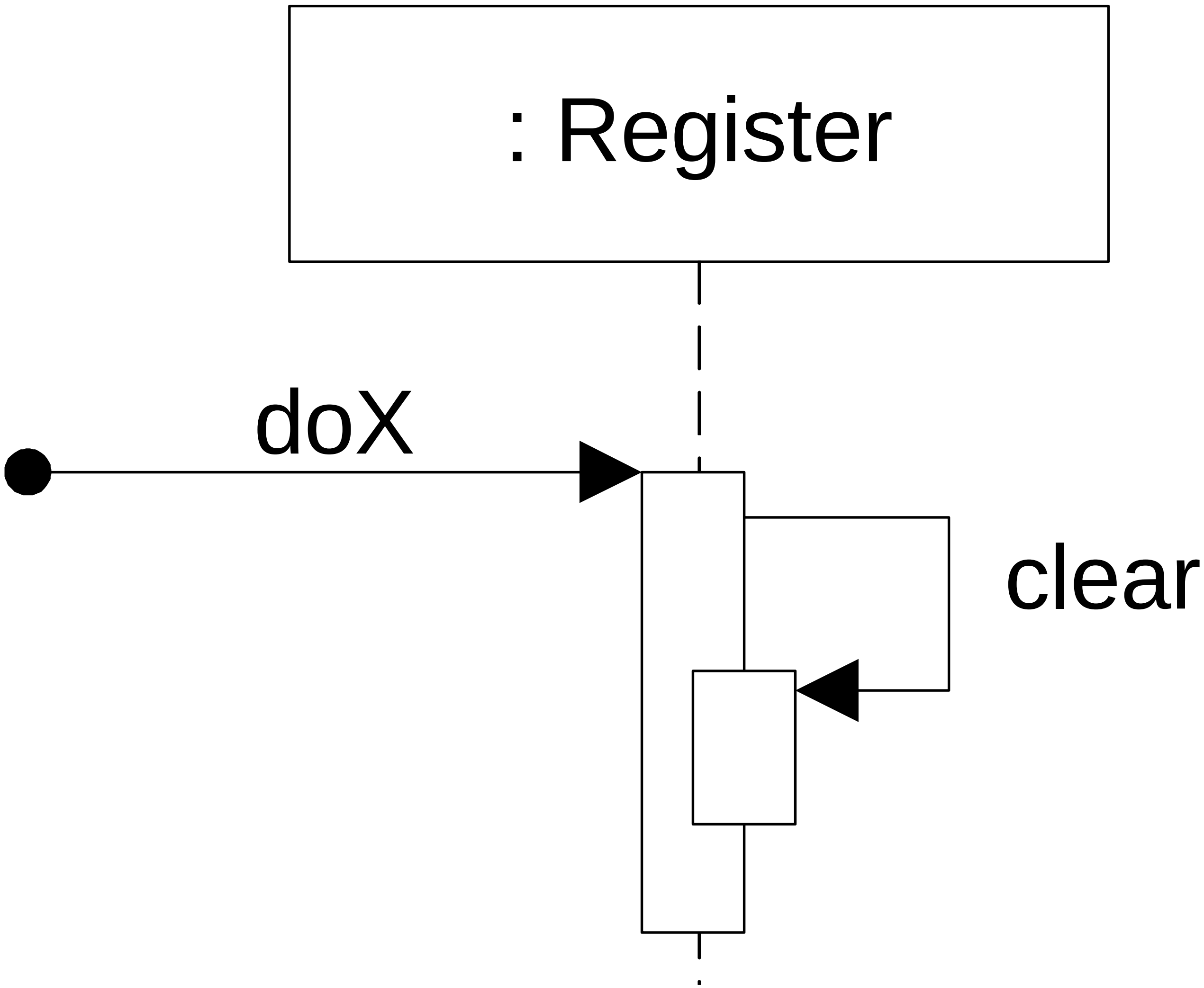


Fig. 15.10

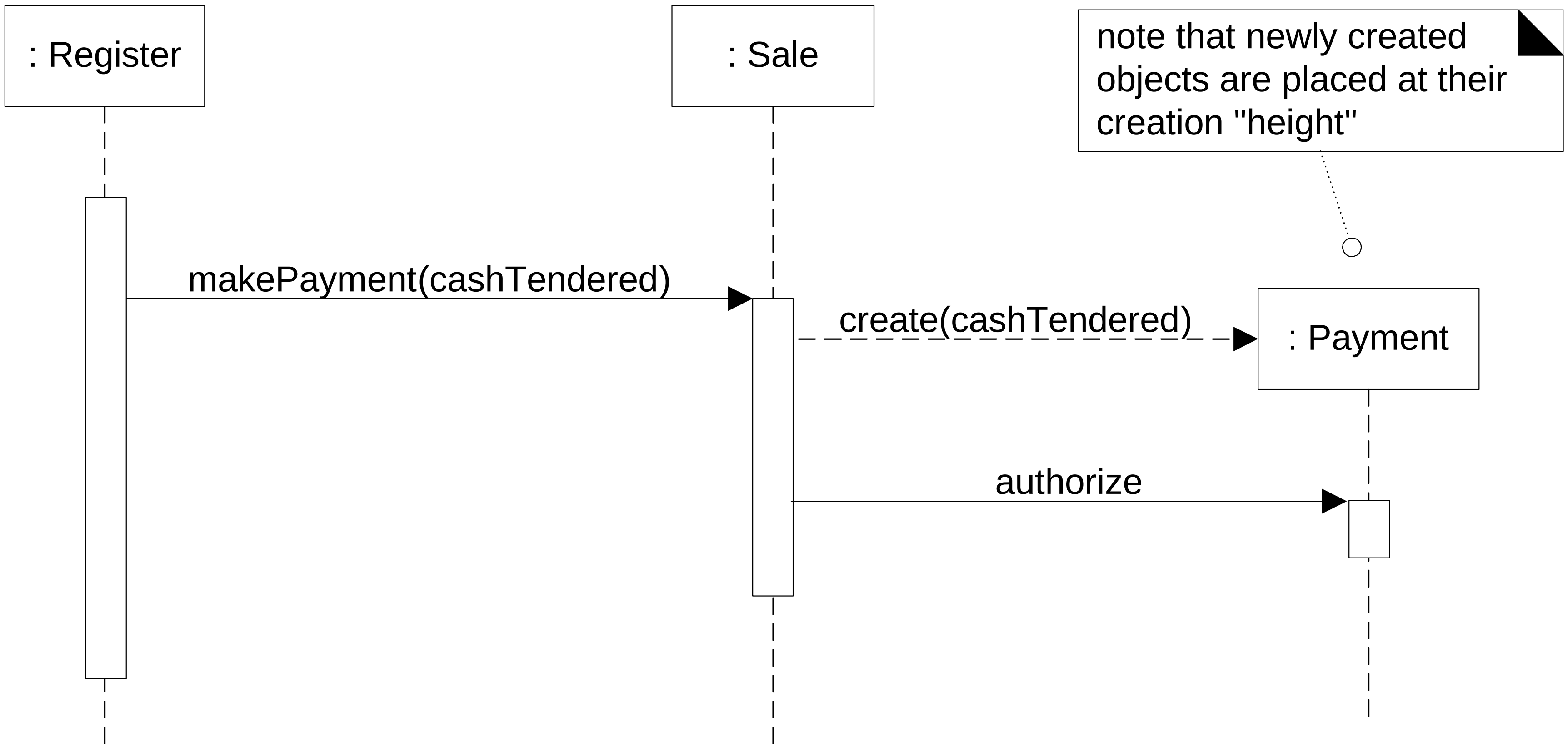

: Register
: Sale
note that newly created objects are placed at their creation "height"
makePayment(cashTendered)
create(cashTendered)
: Payment
authorize

Fig. 15.11

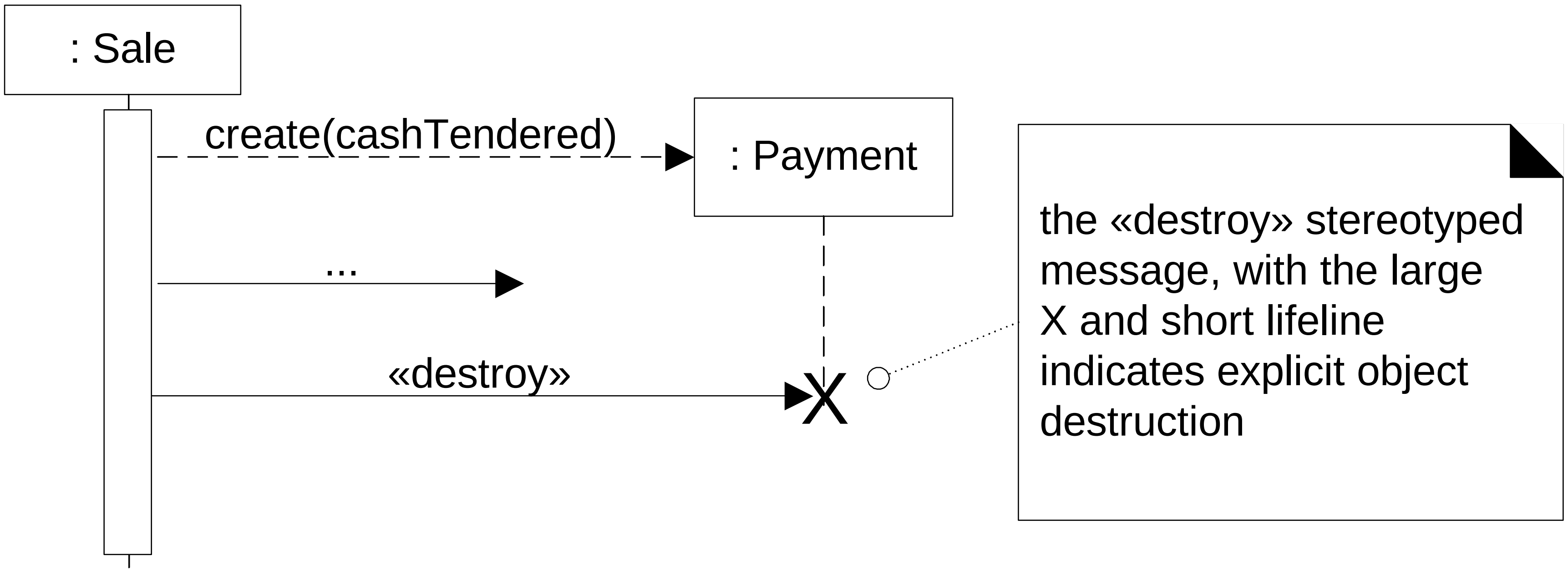

: Sale
create(cashTendered)
: Payment
...
«destroy»
X
the «destroy» stereotyped message, with the large X and short lifeline indicates explicit object destruction

Fig. 15.12

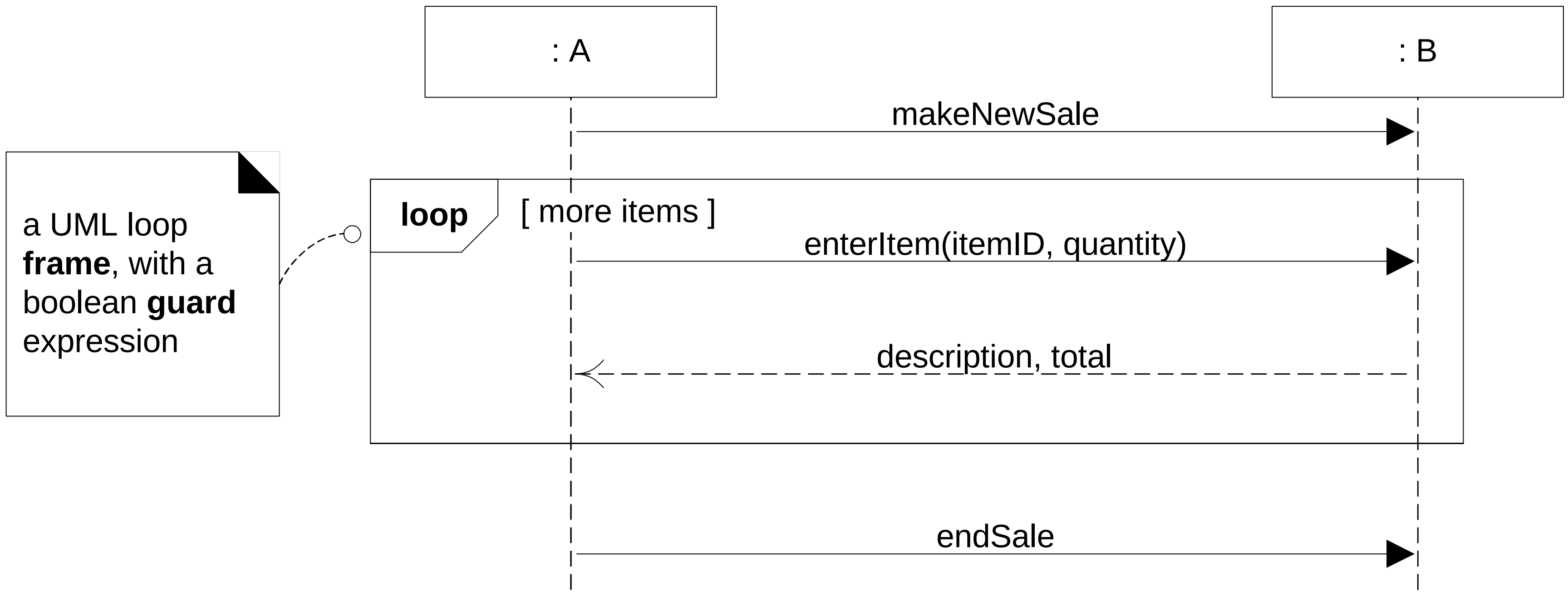

: A
: B
makeNewSale
loop
[more items]
enterItem(itemID, quantity)
description, total
endSale
a UML loop **frame**, with a boolean **guard** expression

Fig. 15.13

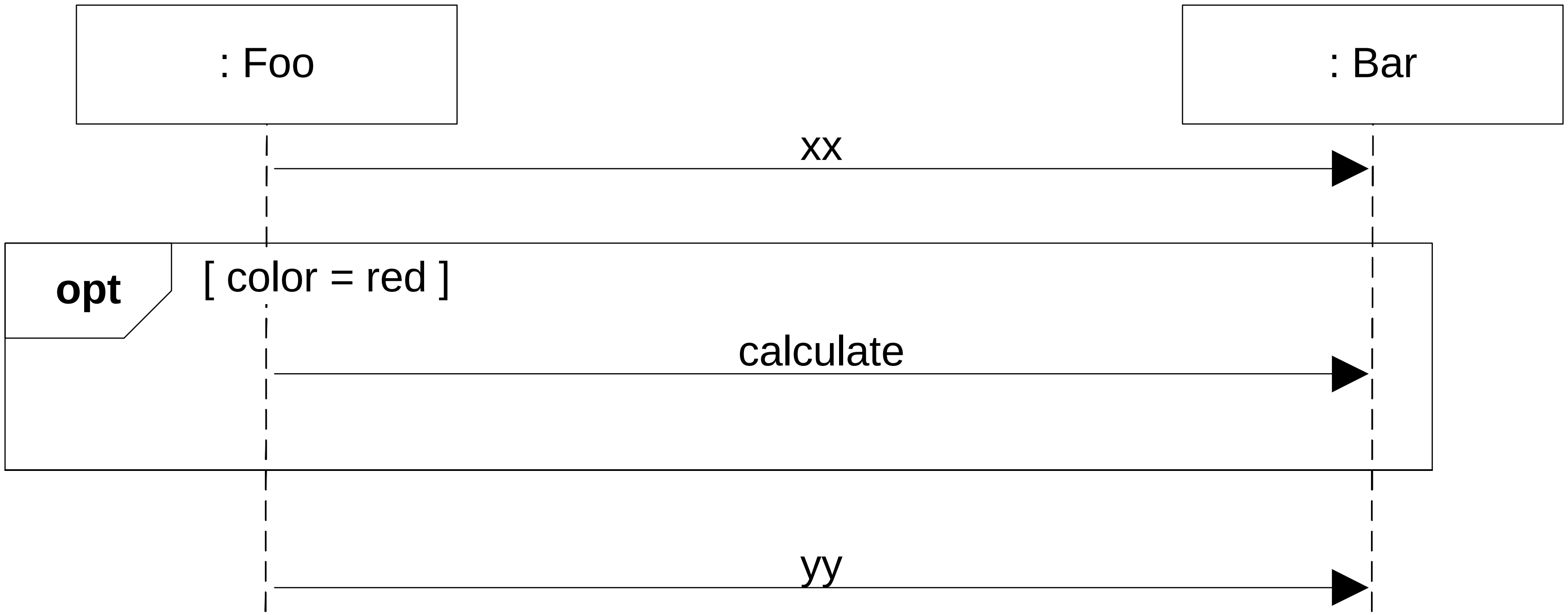


Fig. 15.14

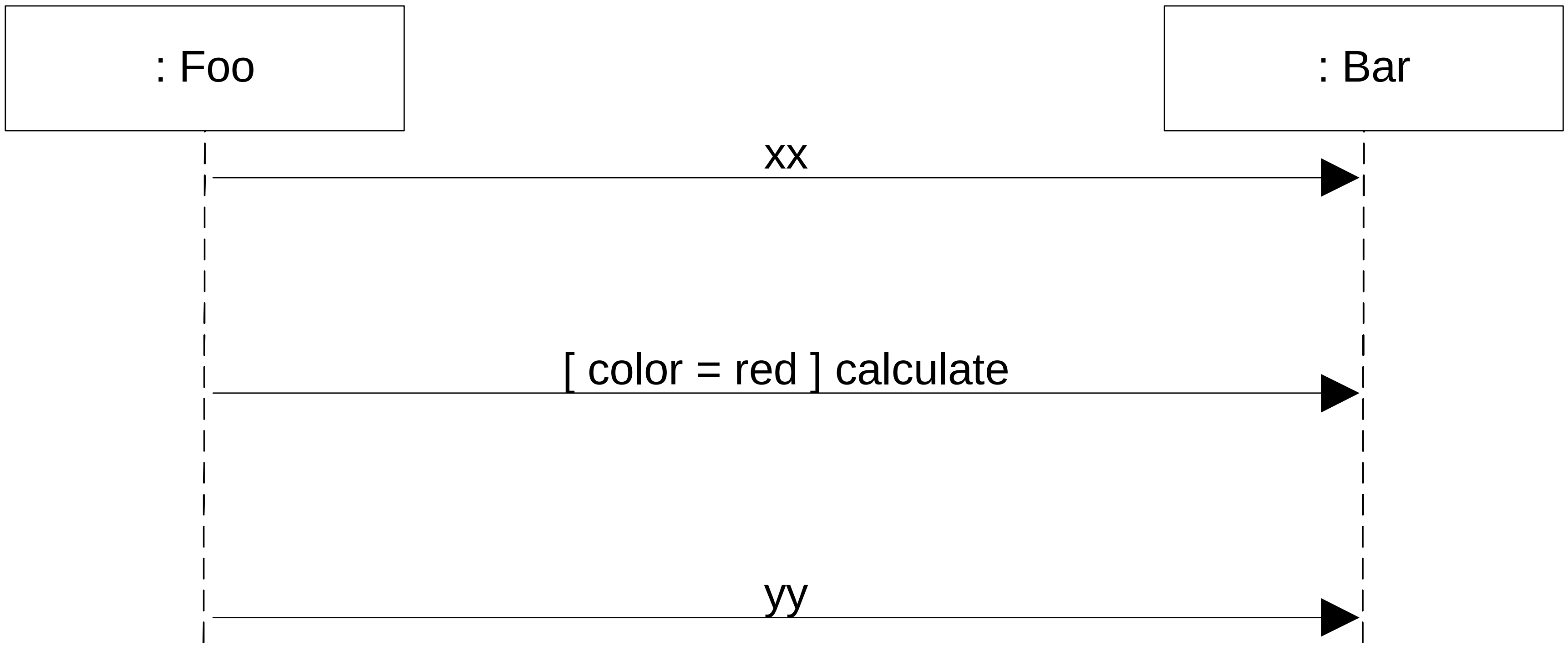


Fig. 15.15

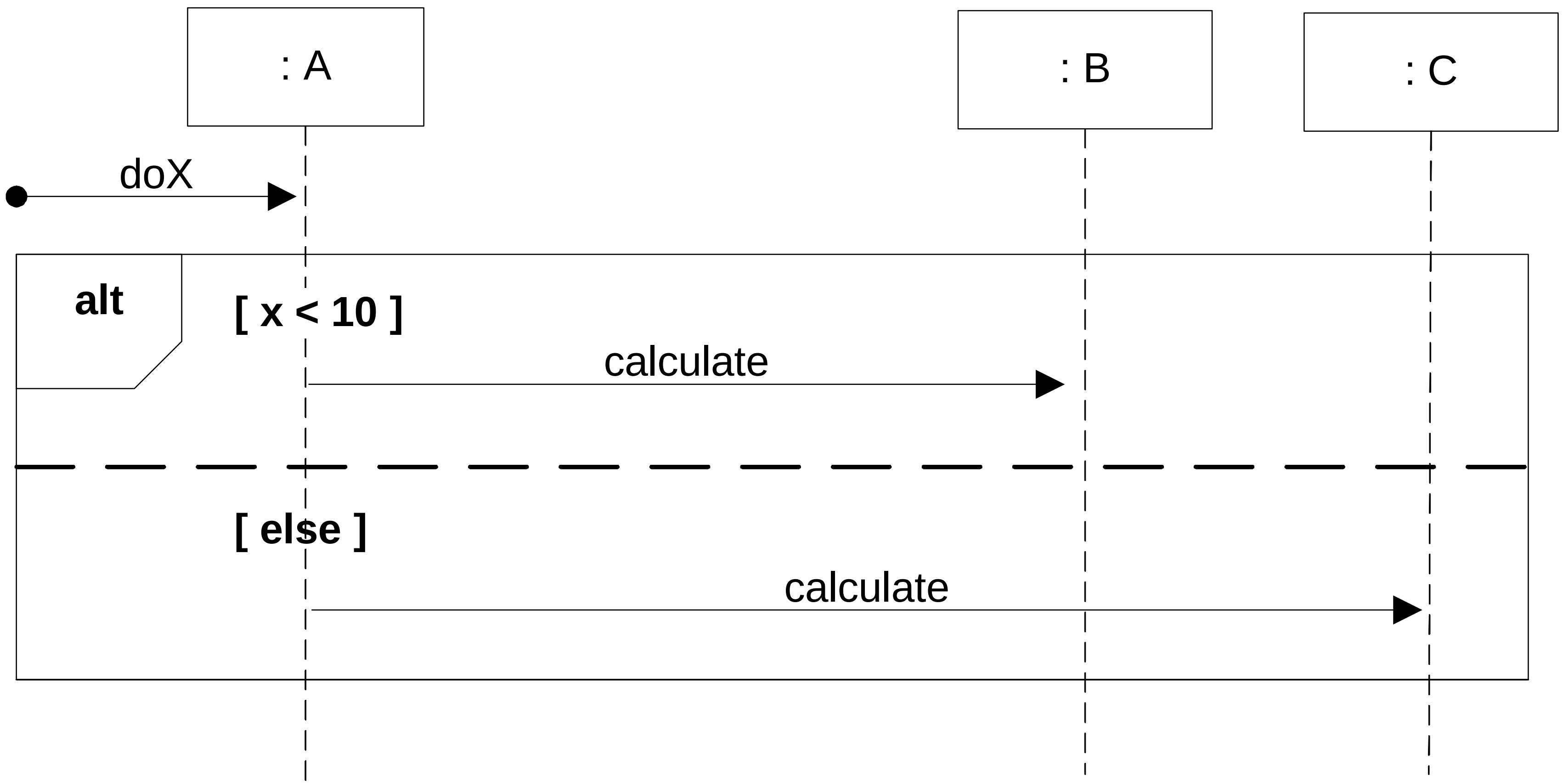
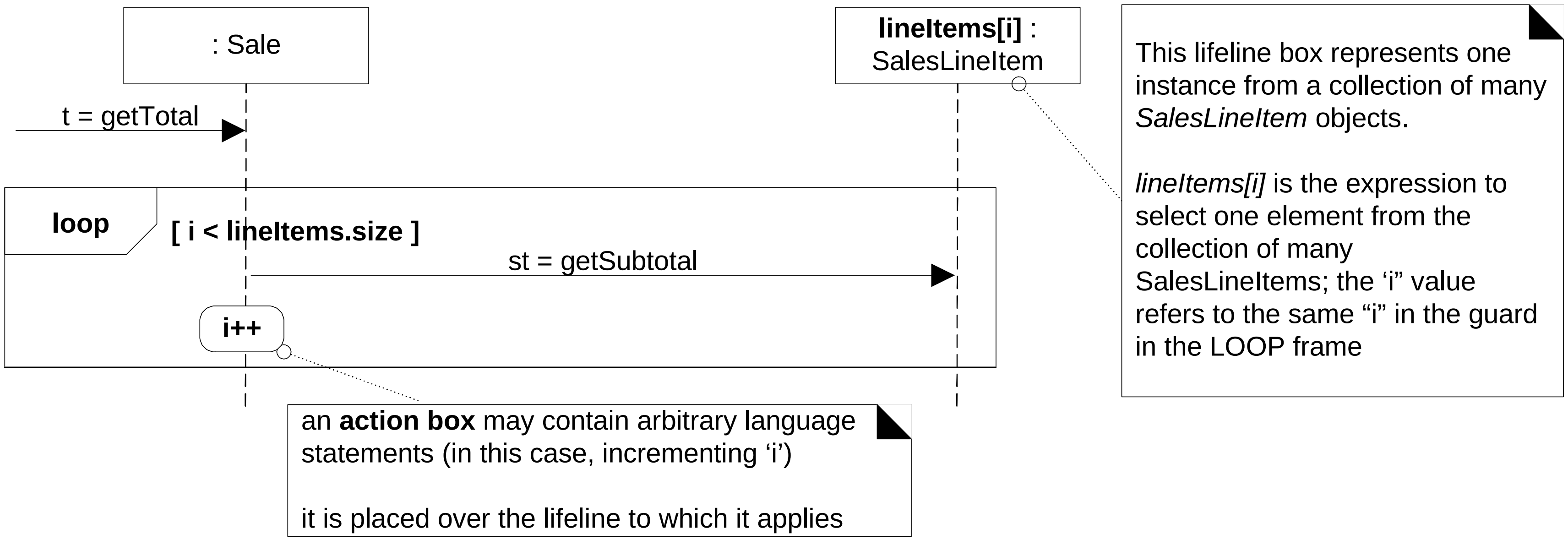
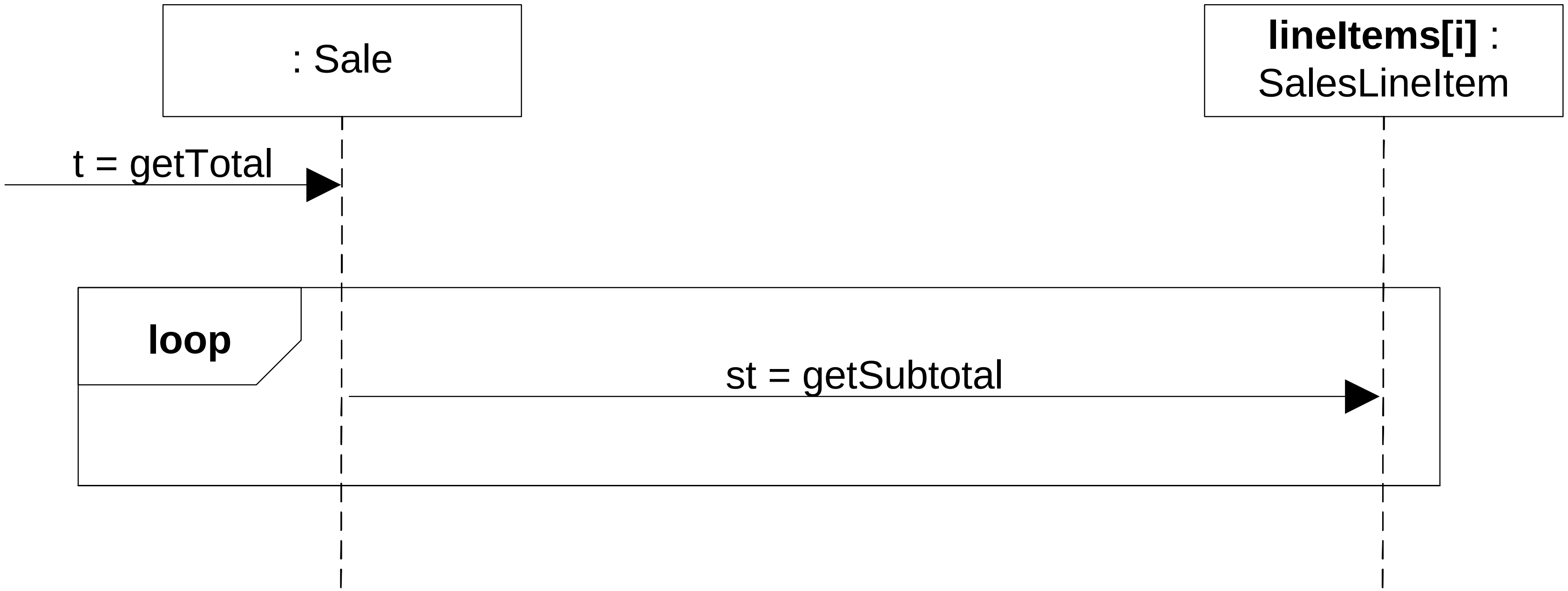


Fig. 15.16



: Sale
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
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

Fig. 15.17



: Sale
lineItems[i] :
SalesLineItem
t = getTotal
loop
st = getSubtotal

Fig. 15.18

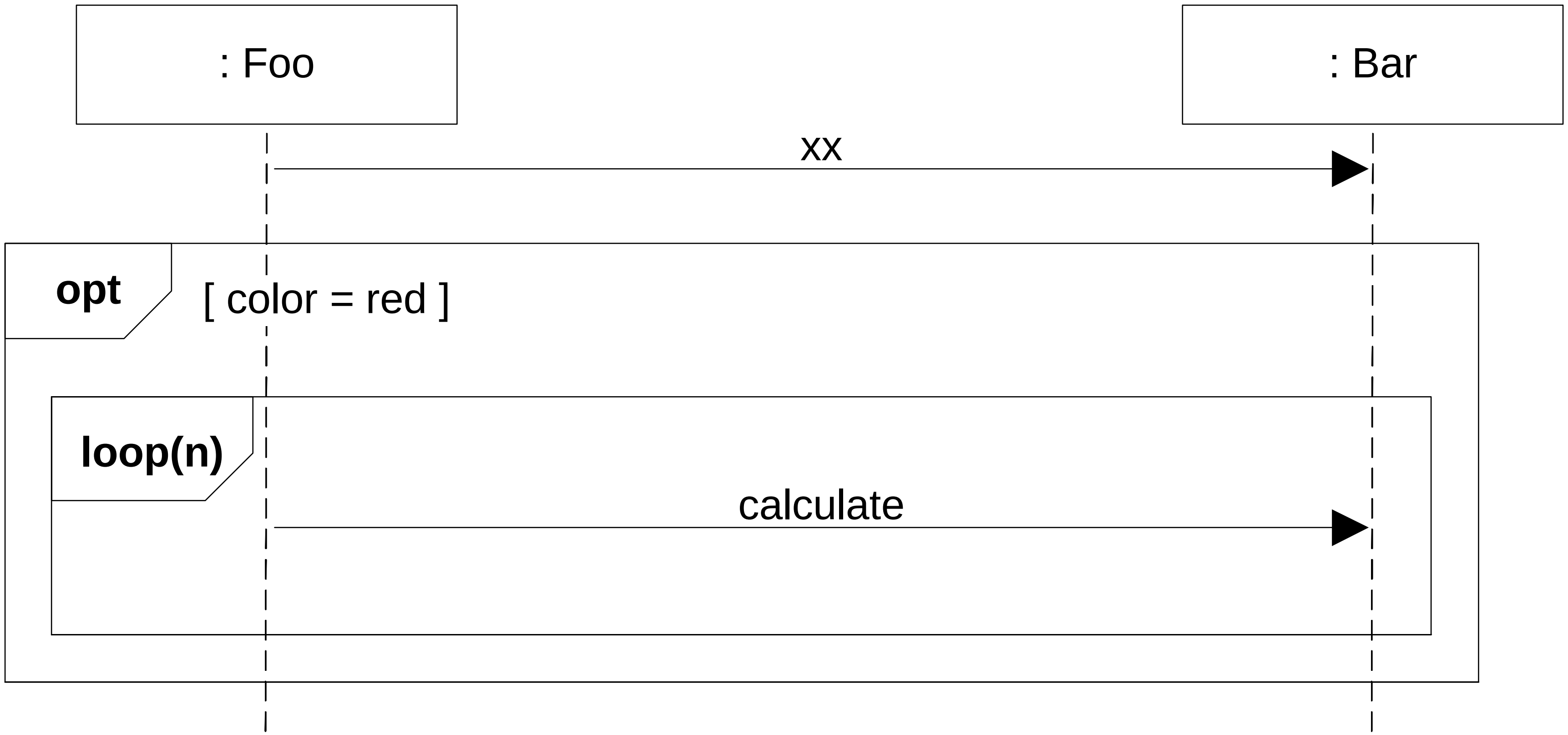
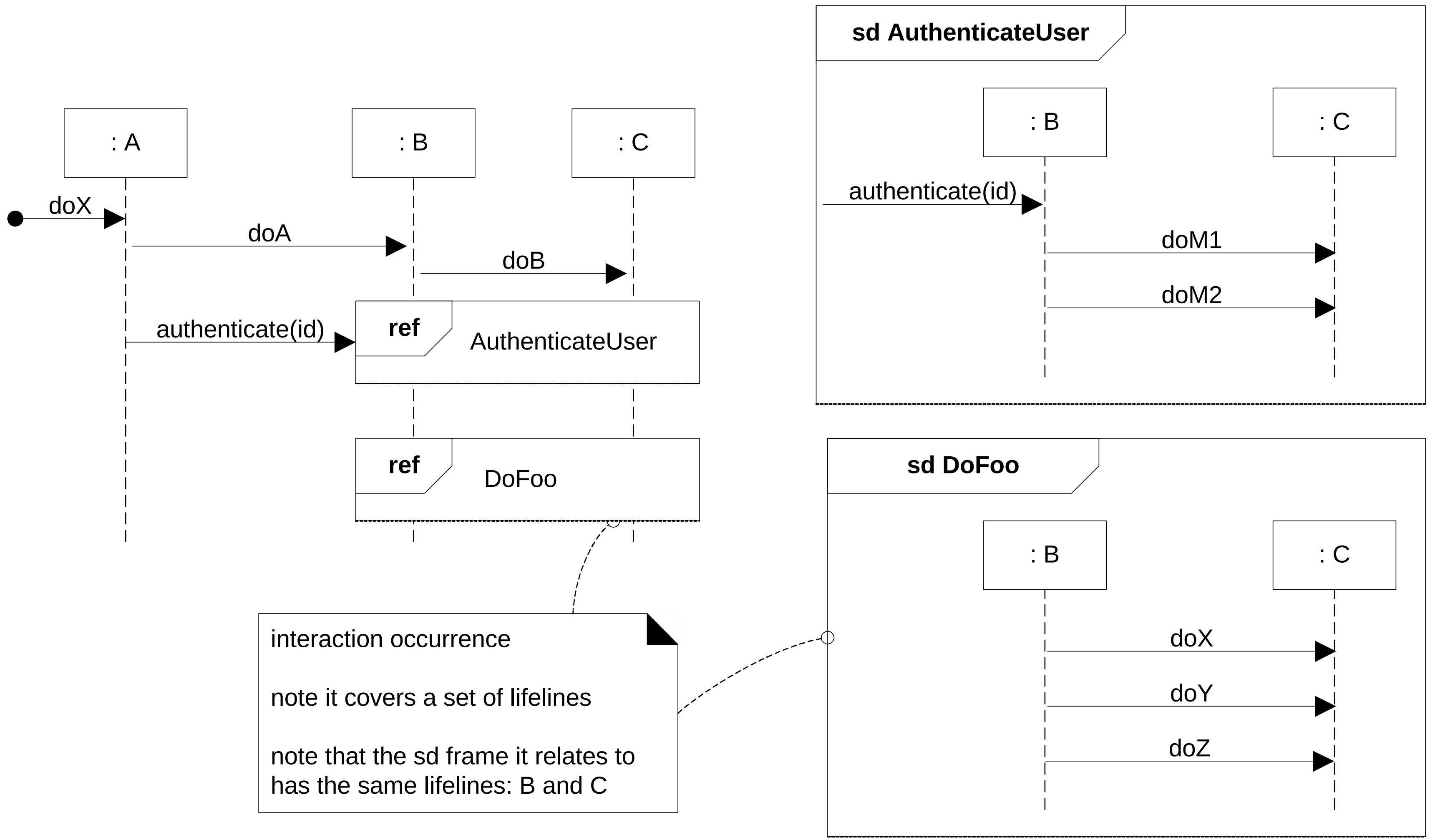


Fig. 15.19



: A
: B
: C
doX
doA
doB
authenticate(id)
ref
AuthenticateUser
ref
DoFoo
interaction occurrence
note it covers a set of lifelines
note that the sd frame it relates to has the same lifelines: B and C
sd AuthenticateUser
: B
: C
authenticate(id)
doM1
doM2
sd DoFoo
: B
: C
doX
doY
doZ

Fig. 15.20

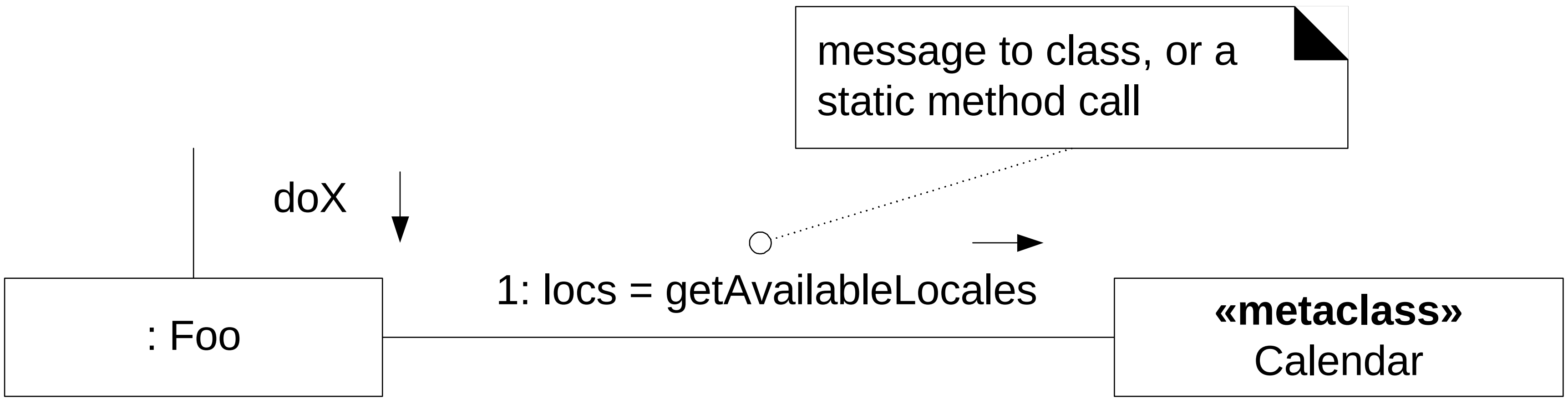
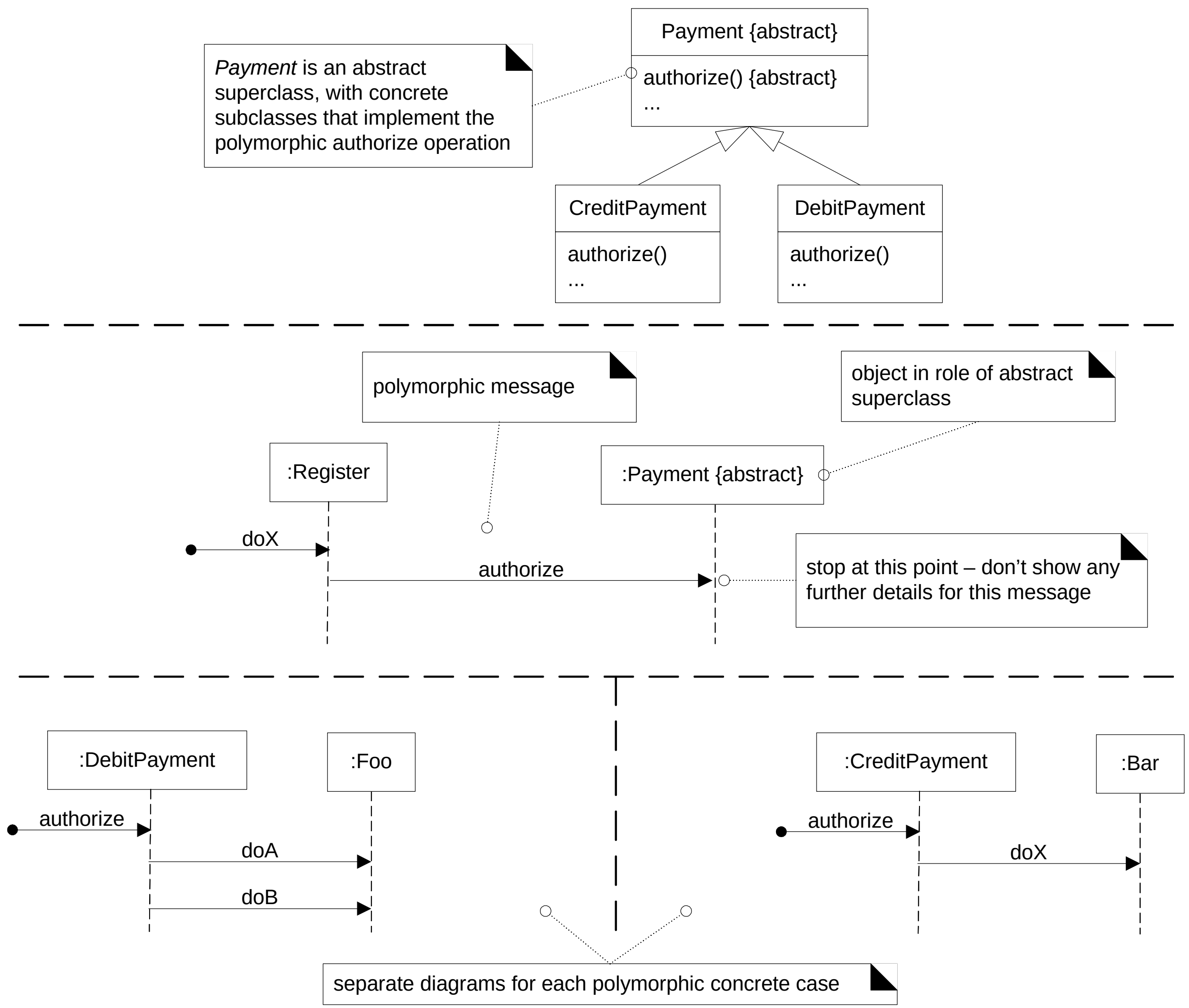

message to class, or a static method call
doX
1: locs = getAvailableLocales
: Foo
«metaclass»
Calendar

Fig. 15.21



Payment {abstract}
authorize() {abstract}
...
Payment is an abstract superclass, with concrete subclasses that implement the polymorphic authorize operation
CreditPayment
authorize()
...
DebitPayment
authorize()
...
polymorphic message
object in role of abstract superclass
:Register
:Payment {abstract}
doX
authorize
stop at this point – don't show any further details for this message
:DebitPayment
:Foo
authorize
doA
doB
:CreditPayment
:Bar
authorize
doX
separate diagrams for each polymorphic concrete case

Fig. 15.22

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

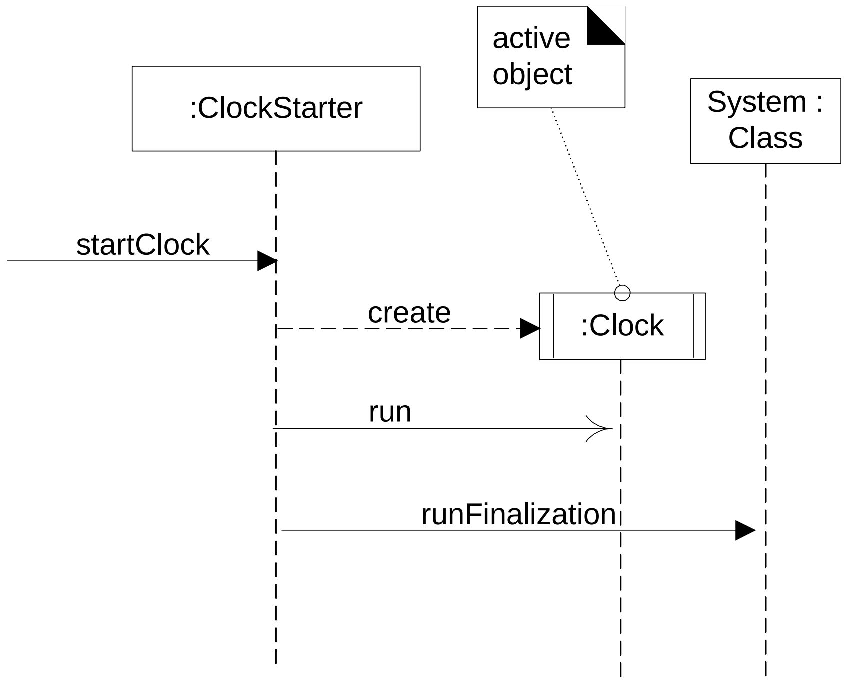


Fig. 15.23

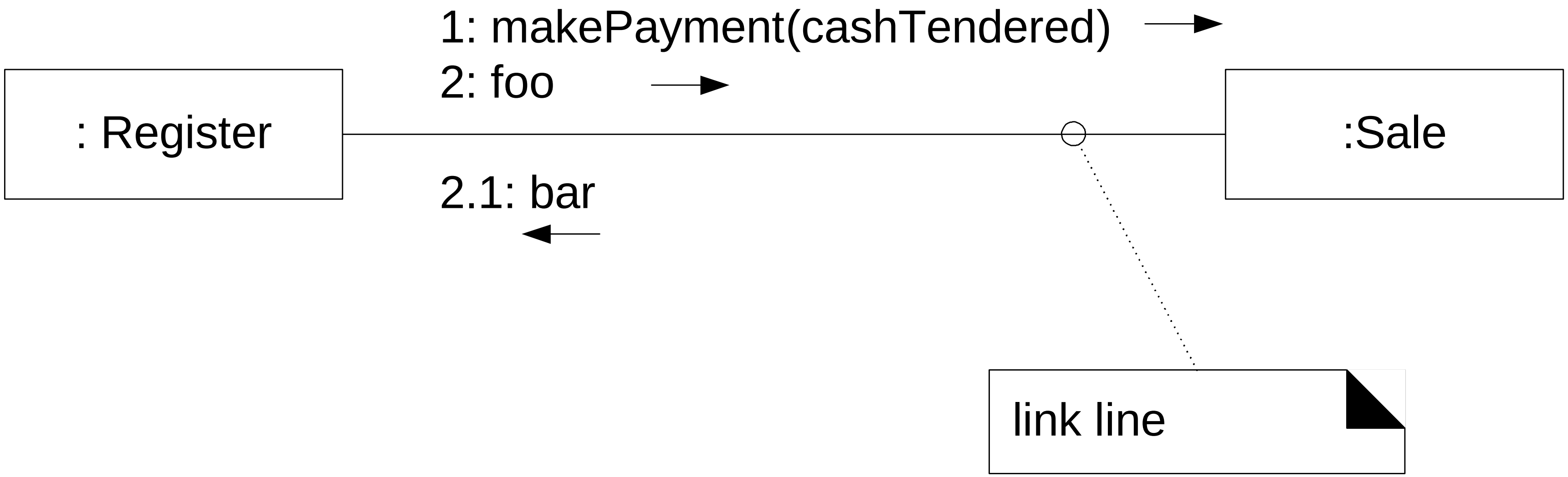

1: makePayment(cashTendered)
2: foo
: Register
:Sale
2.1: bar
link line

Fig. 15.24

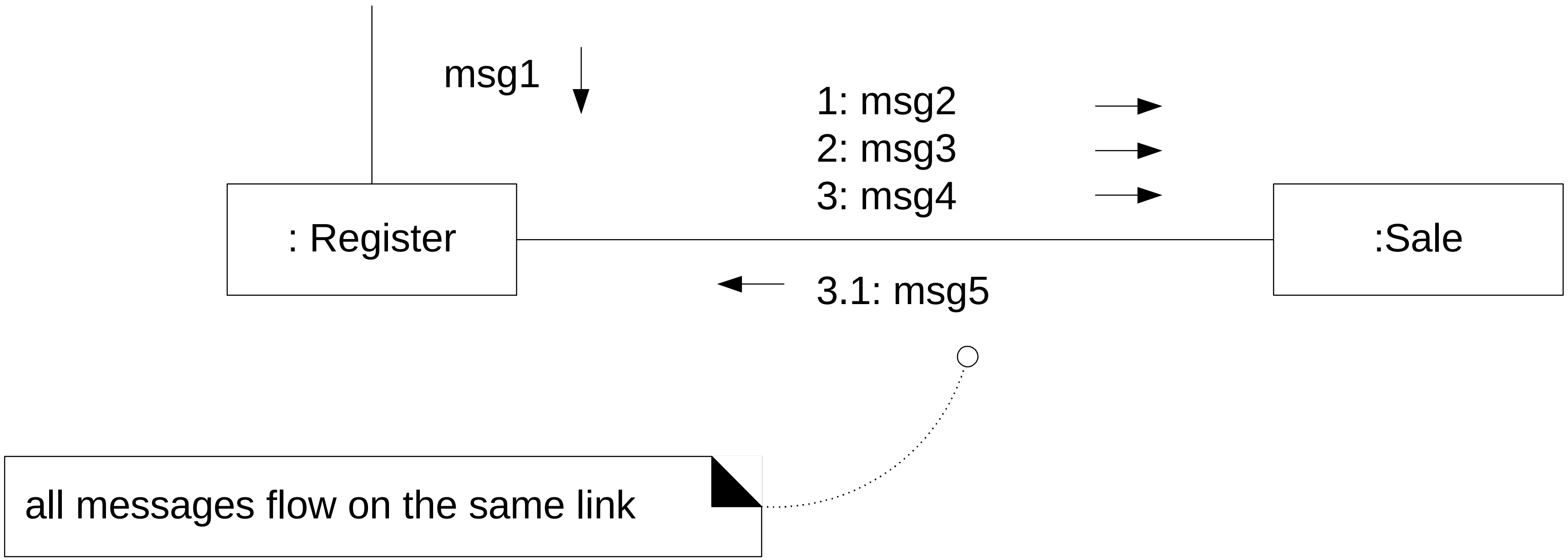

msg1
1: msg2
2: msg3
3: msg4
: Register
:Sale
3.1: msg5
all messages flow on the same link

Fig. 15.25

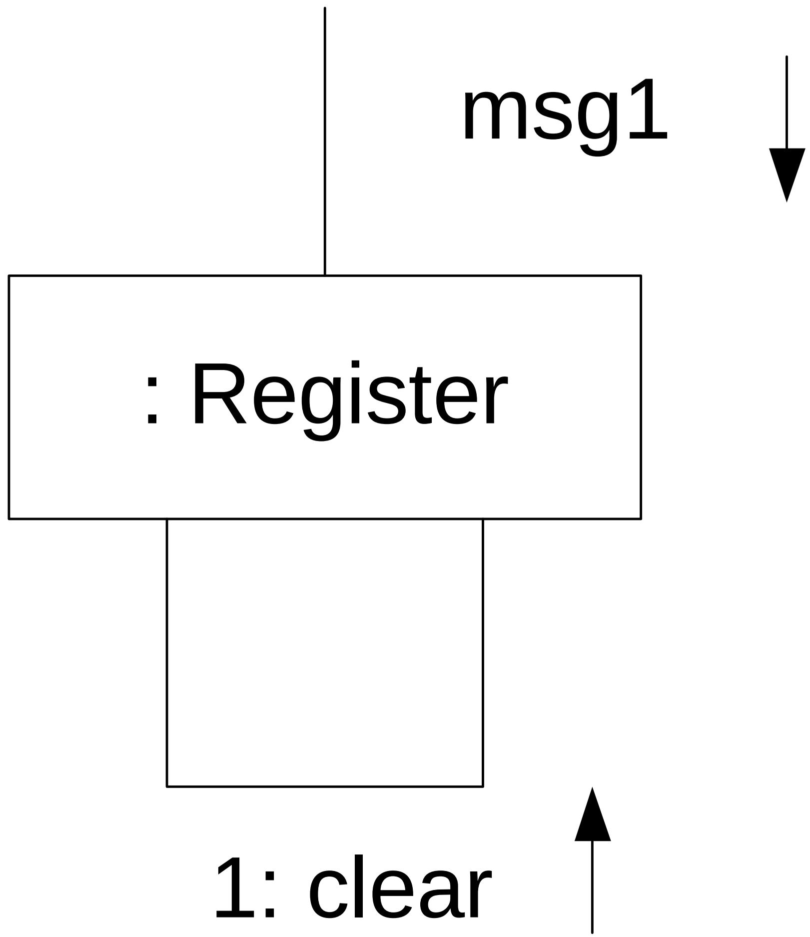


Fig. 15.26

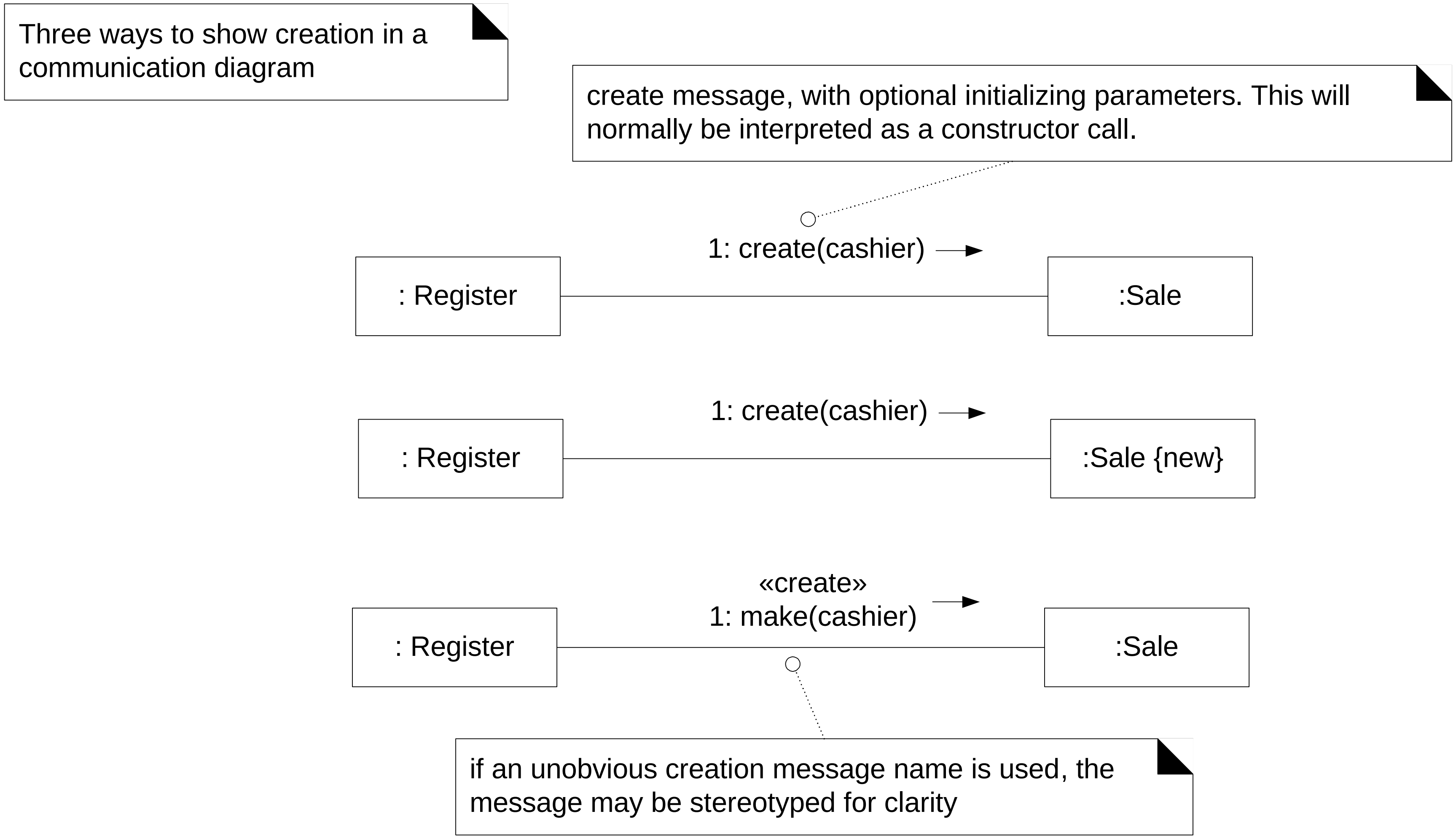

Three ways to show creation in a communication diagram
create message, with optional initializing parameters. This will normally be interpreted as a constructor call.
1: create(cashier)
: Register
:Sale
1: create(cashier)
: Register
:Sale {new}
«create»
1: make(cashier)
: Register
:Sale
if an unobvious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

Fig. 15.27

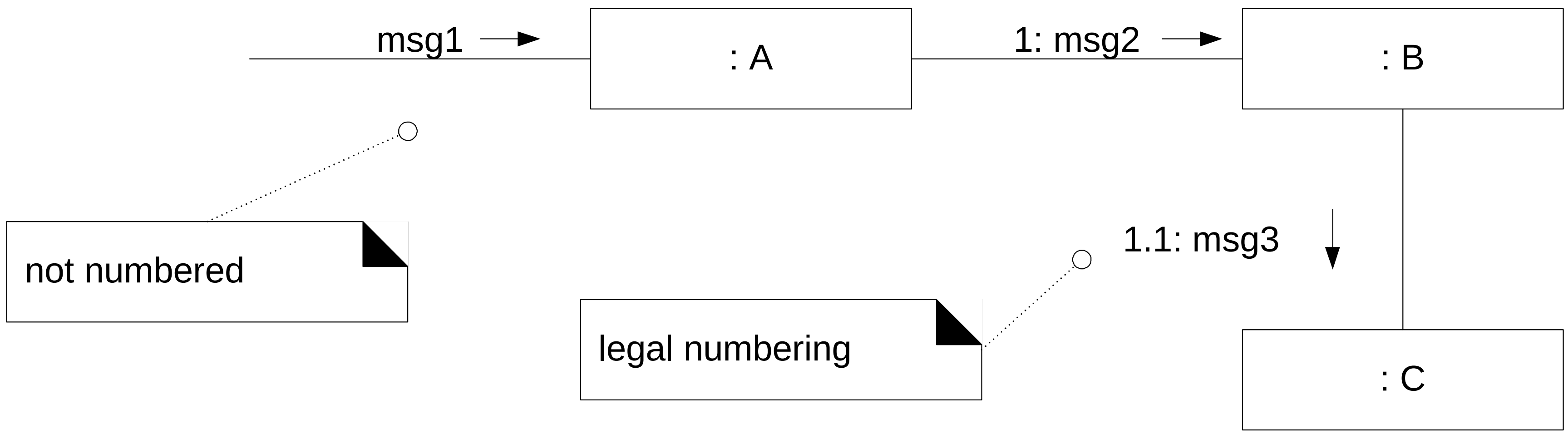


Fig. 15.28

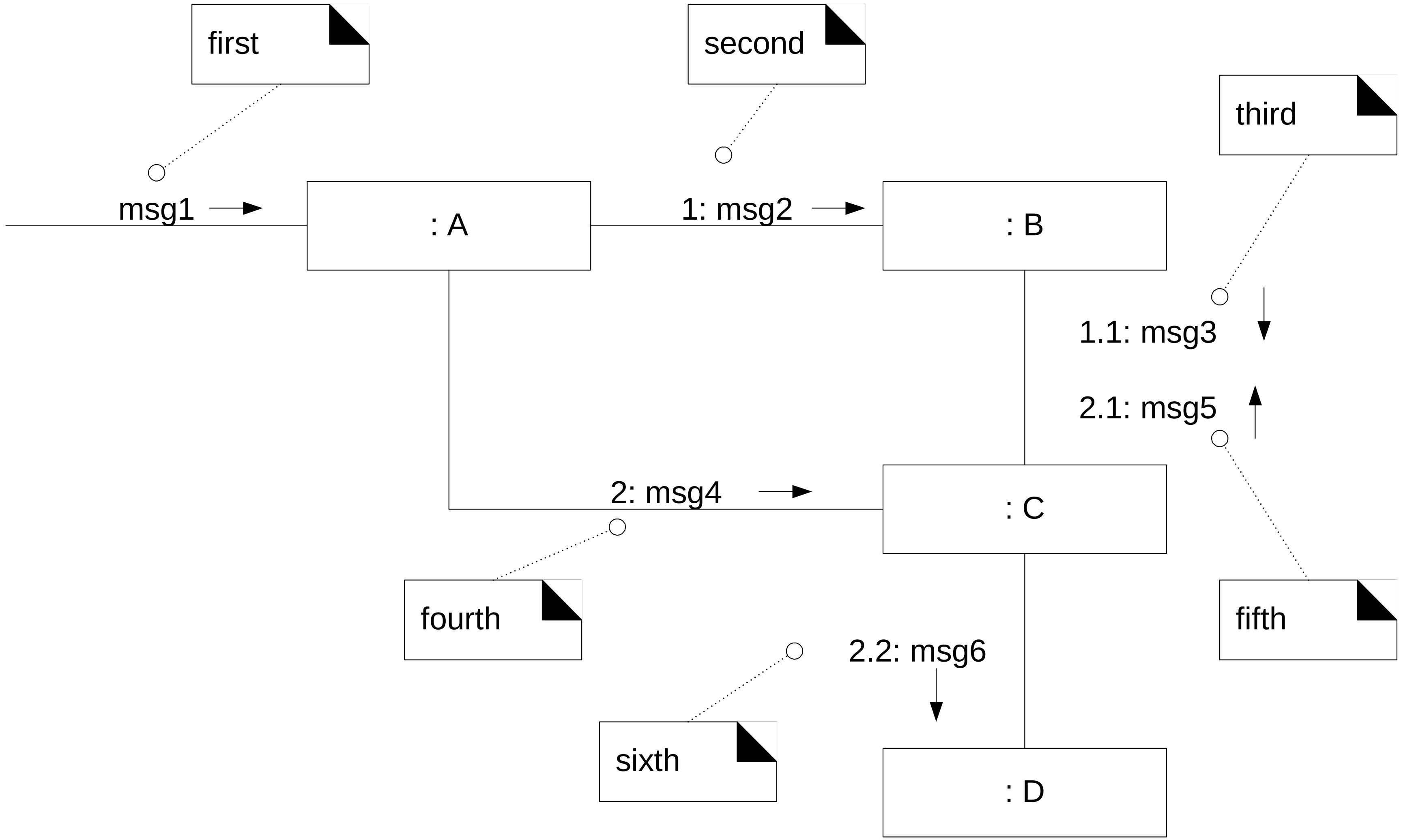

first
second
third
msg1
: A
1: msg2
: B
1.1: msg3
2.1: msg5
2: msg4
: C
fourth
fifth
2.2: msg6
sixth
: D

Fig. 15.29

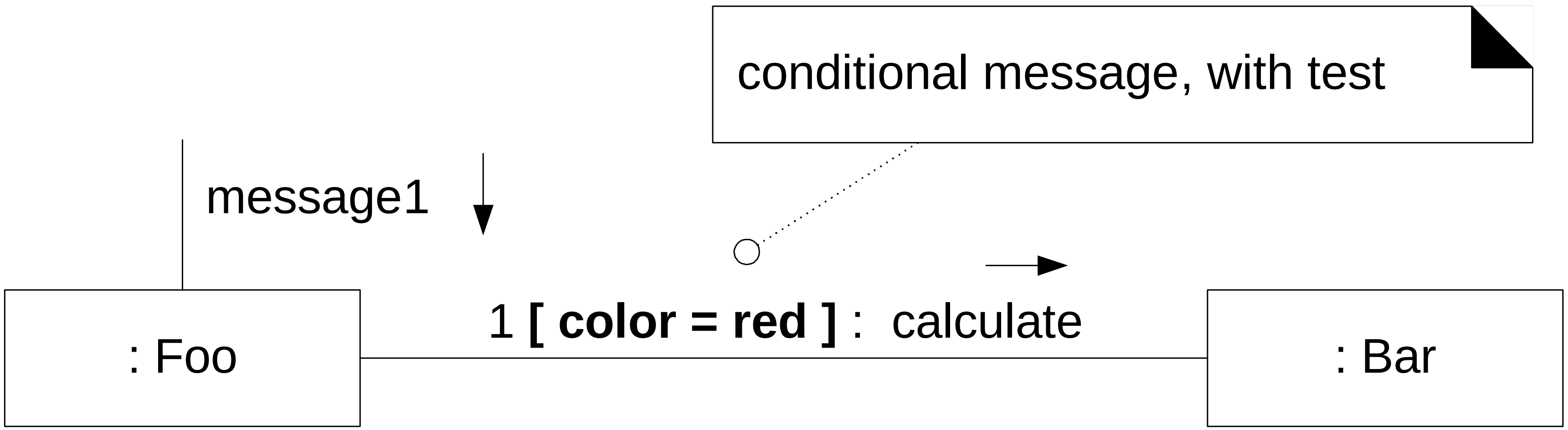
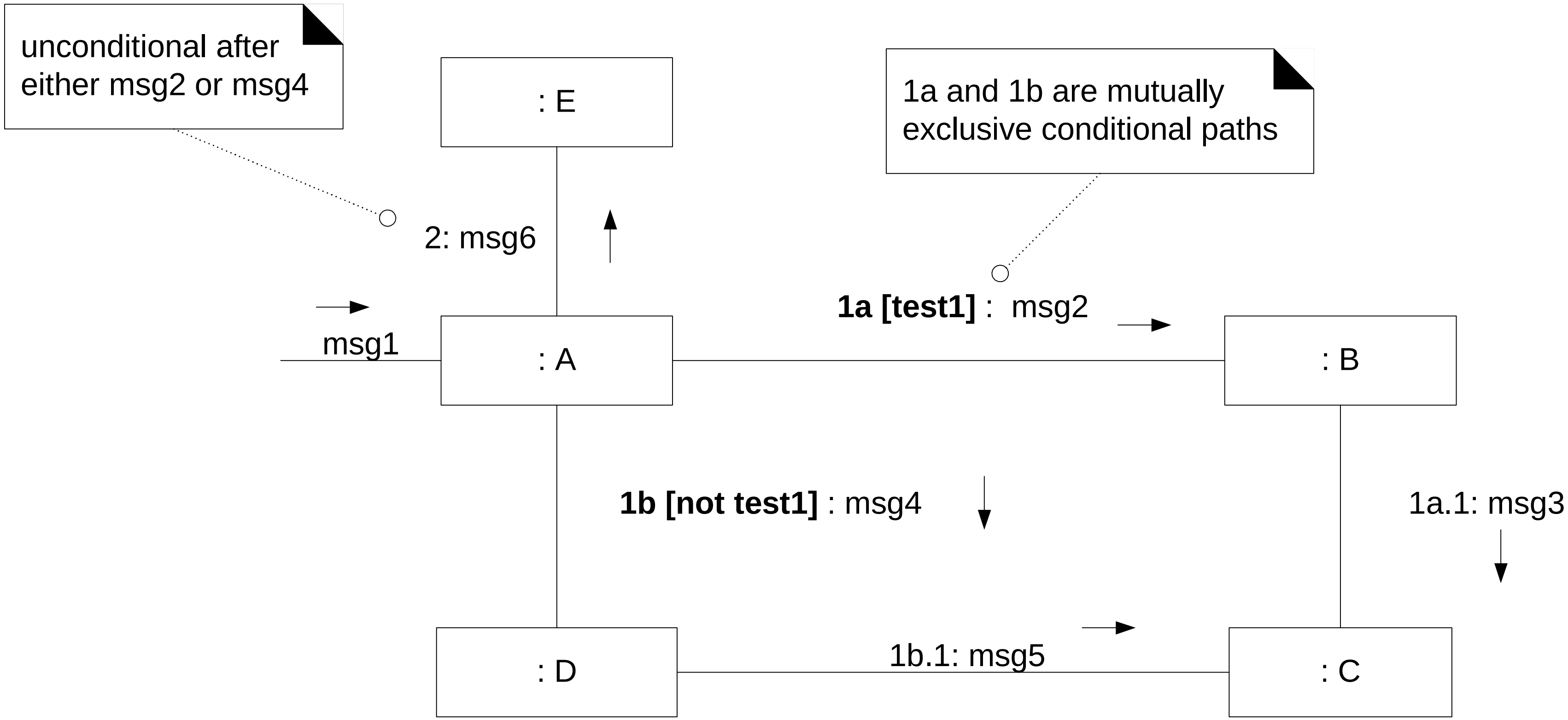
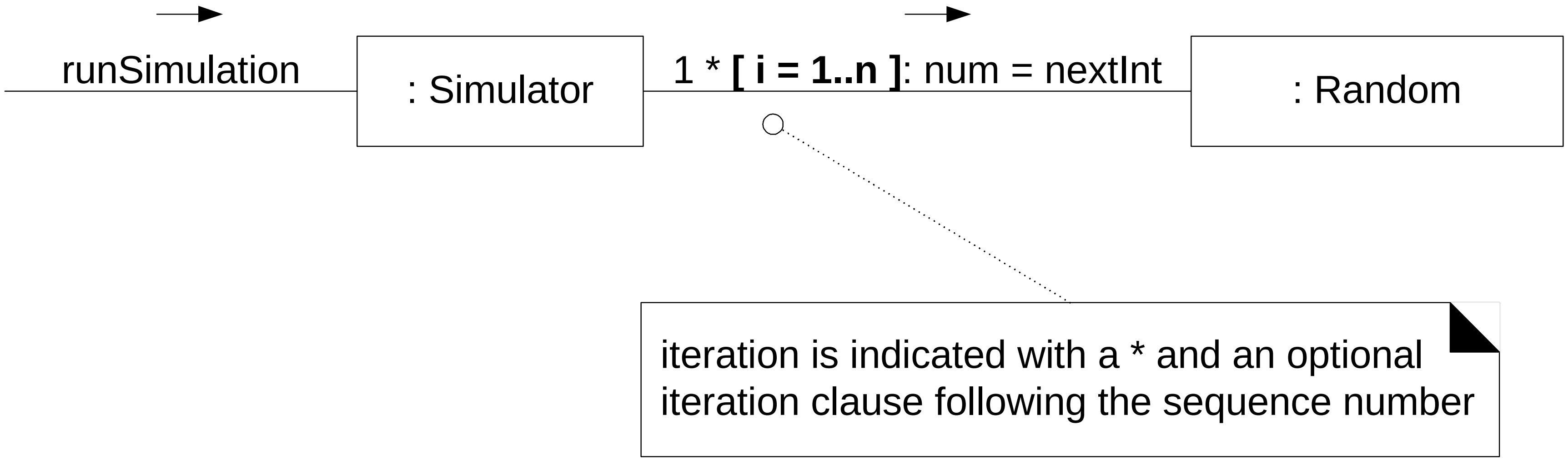

conditional message, with test
message1
1 [color = red] : calculate
: Foo
: Bar

Fig. 15.30



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

Fig. 15.31



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

Fig. 15.32

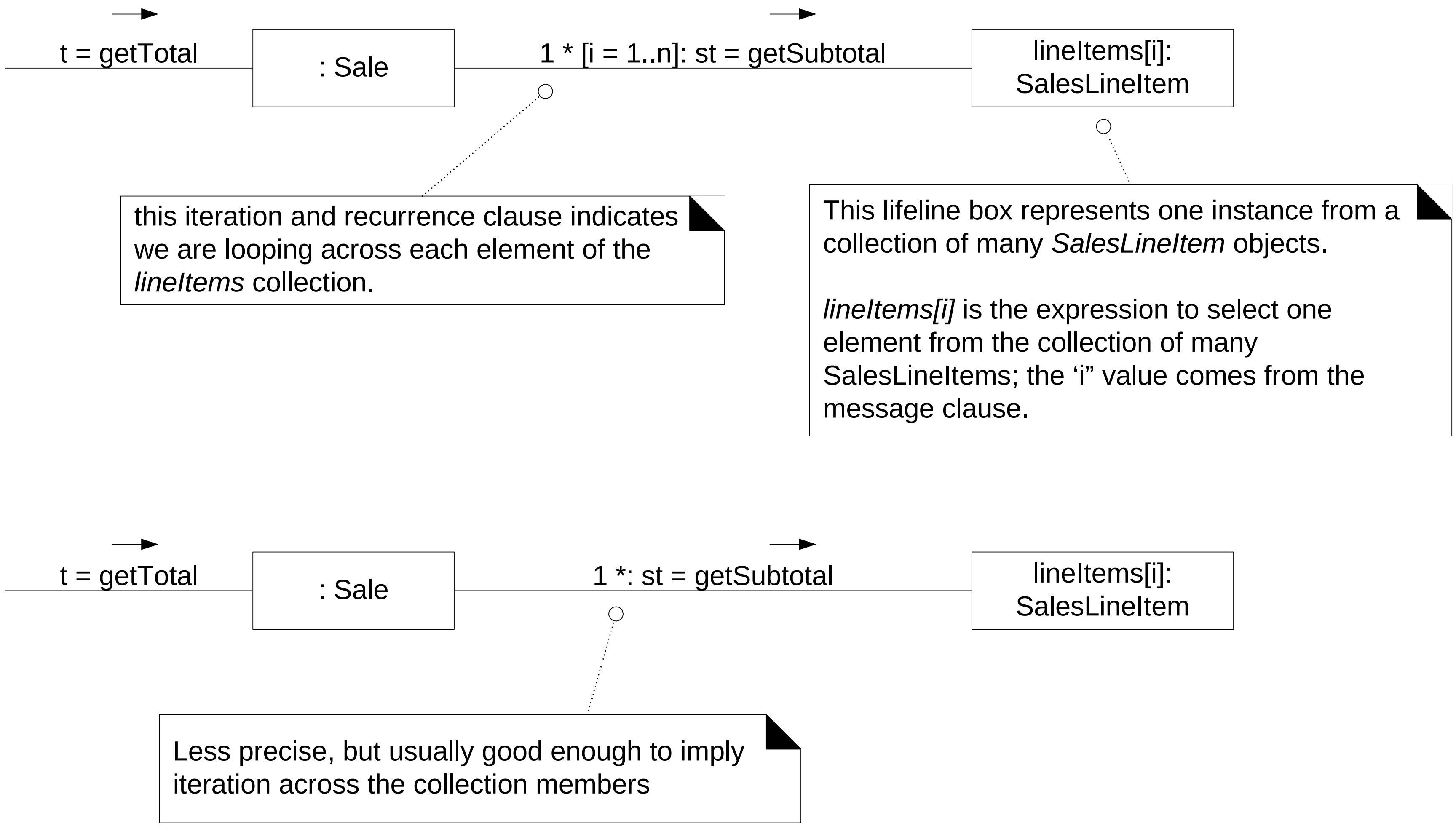
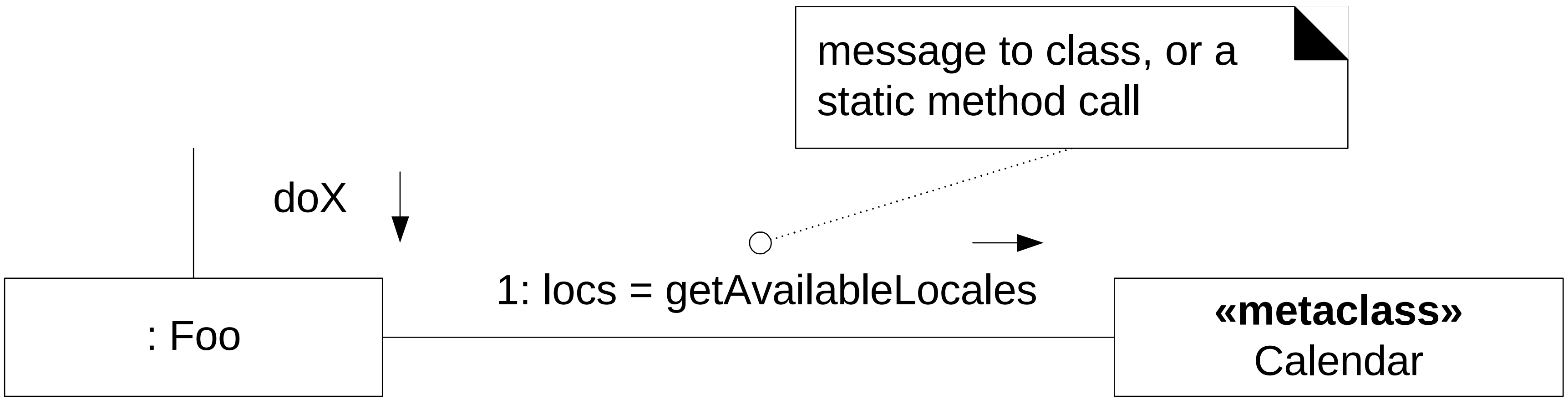

t = getTotal
: Sale
1 * [i = 1..n]: st = getSubtotal
lineItems[i]:
SalesLineItem
this iteration and recurrence clause indicates we are looping across each element of the *lineItems* collection.
This lifeline box represents one instance from a collection of many *SalesLineItem* objects.
lineItems[i] is 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

Fig. 15.33



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

Fig. 15.34

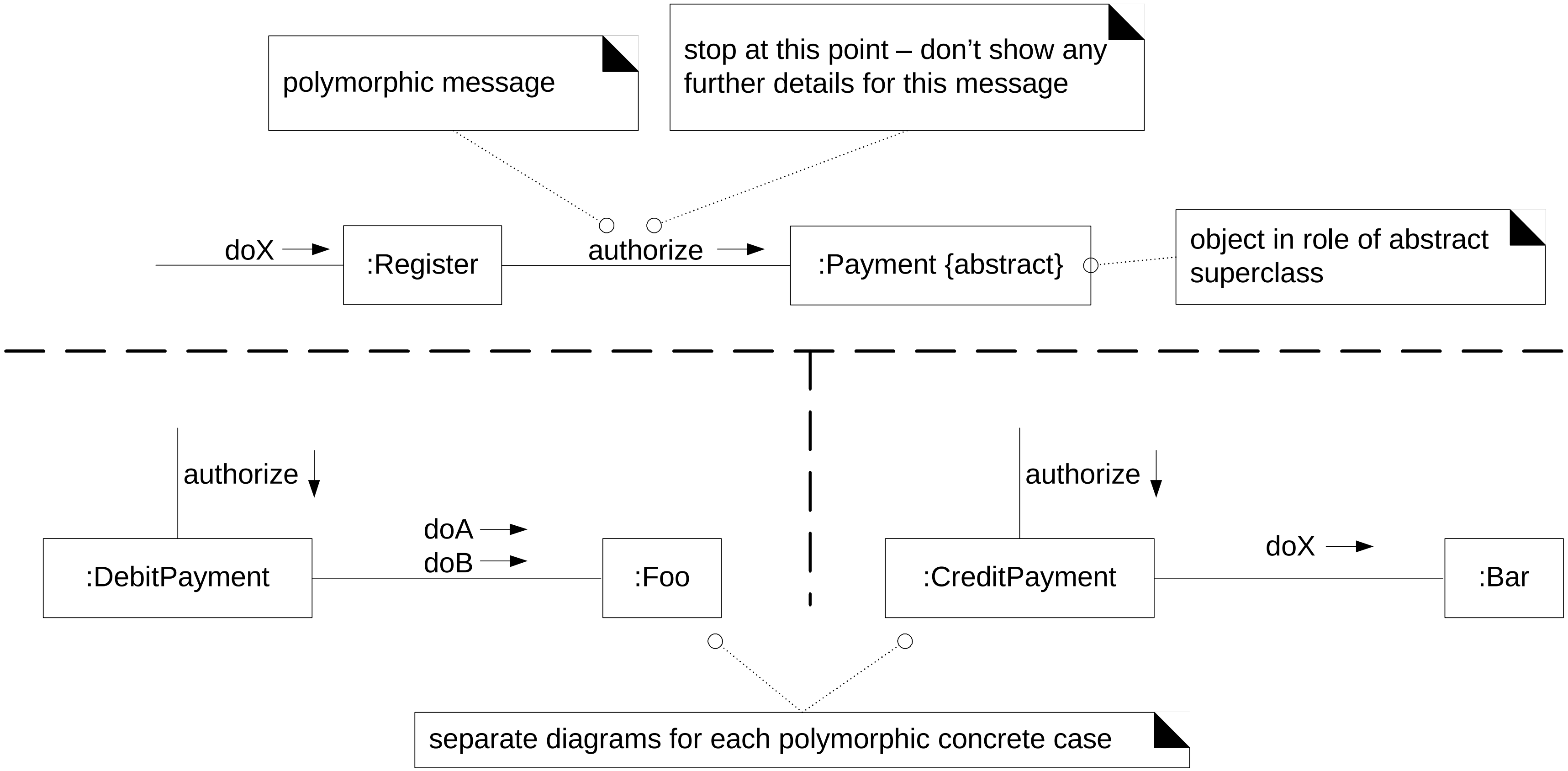
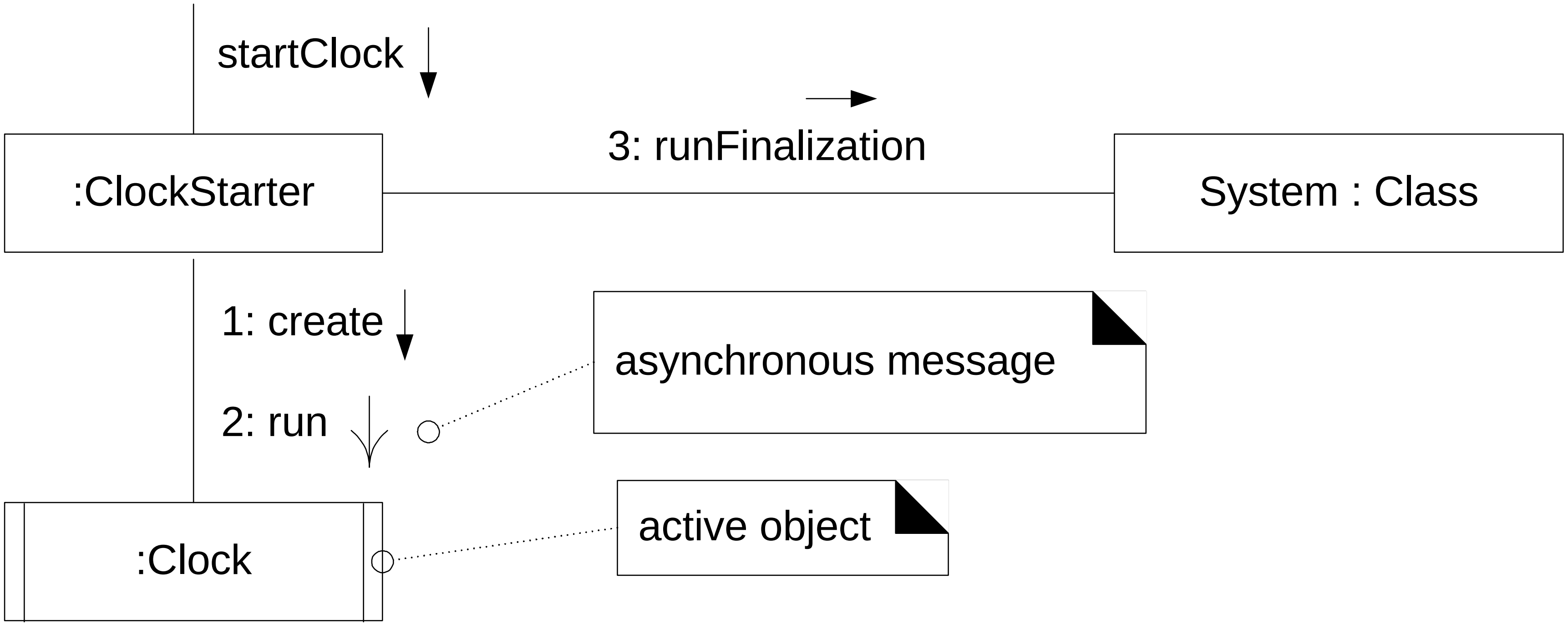

polymorphic message
stop at this point – don't show any further details for this message
doX
:Register
authorize
:Payment {abstract}
object in role of abstract superclass
authorize
:DebitPayment
doA
doB
:Foo
authorize
:CreditPayment
doX
:Bar
separate diagrams for each polymorphic concrete case

Fig. 15.35


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