Chapter 15
UML Interaction Diagrams

Larman, C. “Applying UML and Patterns”. 3rd Ed.
Fig. 15.2

```
A
```

```
doOne
```

```
myB : B
```

```
doOne
```
Fig. 15.4

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

The ‘1’ implies this is a Singleton, and accessed via the Singleton pattern.
a found message whose sender will not be specified

execution specification bar indicates focus of control

typical synchronous message shown with a filled-arrow line
Fig. 15.8

: Register

: Sale

d1 = getDate

date

get_Date

aDate
Fig. 15.10

: Register

makePayment(cashTendered)

: Sale

create(cashTendered)

: Payment

authorize

note that newly created objects are placed at their creation "height"
Fig. 15.11

the «destroy» stereotyped message, with the large X and short lifeline indicates explicit object destruction.
Fig. 15.12

A UML loop frame, with a boolean guard expression

:a UML loop frame, with a boolean guard expression

: A

makeNewSale

: B

[ more items ]

enterItem(itemID, quantity)

description, total

descendSale

[ more items ]
Fig. 15.13

: Foo

: Bar

opt [ color = red ]
calculate

xx

yy
Fig. 15.14

: Foo

: Bar

xx

[ color = red ] calculate

yy
Fig. 15.15

: A

doX

: B

: C

alt

[x < 10]

calculate

[else]

calculate
Fig. 15.16

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
Fig. 15.17

: Sale

: lineltems[i] : SalesLineItem

t = getTotal

loop

st = getSubtotal
Fig. 15.18

: Foo

: Bar

xx

opt [ color = red ]

loop(n)

calculate
interaction occurrence

note it covers a set of lifelines

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

Fig. 15.19

sd AuthenticateUser

authenticate(id)

doM1

doM2

sd DoFoo

doX

doY

doZ
Fig. 15.20

1: locs = getAvailableLocales

message to class, or a static method call

: Foo

«metaclass»
Calendar
Payment is an abstract superclass, with concrete subclasses that implement the polymorphic authorize operation.

Payment (abstract)
  authorize() (abstract)
  ...

CreditPayment
  authorize()
  ...

DebitPayment
  authorize()

polymorphic message

object in role of abstract superclass

:Register
  doX
  authorize

polymorphic message

stop at this point – don’t show any further details for this message

:Payment (abstract)
  authorize()

:DebitPayment
  authorize
doA
doB

:Foo

:CreditPayment
  authorize
doX

:Bar

separate diagrams for each polymorphic concrete case
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:

```java
// 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
2.1: bar
all messages flow on the same link
Fig. 15.25

msg1

: Register

1: clear
Three ways to show creation in a communication diagram

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

If an unobvious creation message name is used, the message may be stereotyped for clarity.
Fig. 15.27

msg1 → : A → 1: msg2 → : B

not numbered

legal numbering

1.1: msg3 → : C
Fig. 15.28

- first
- second
- third
- fourth
- sixth
- fifth

1. msg1 -> A
2. msg4 -> C
1: msg2 -> B
1.1: msg3 -> C
2.1: msg5 -> C
2.2: msg6 -> D
Fig. 15.29

message1

1 [ color = red ] : calculate

conditional message, with test
unconditional after either msg2 or msg4

1a [test1] : msg2

1a and 1b are mutually exclusive conditional paths

1b [not test1] : msg4

2: msg6

1a.1: msg3

1b.1: msg5
iteration is indicated with a * and an optional iteration clause following the sequence number
This lifeline box represents one instance from a collection of many `SalesLineItem` objects. `lineltems[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.
Fig. 15.33

1: locs = getAvailableLocales

message to class, or a static method call
polymorphic message

stop at this point – don’t show any further details for this message

object in role of abstract superclass

separate diagrams for each polymorphic concrete case
Fig. 15.35

:ClockStarter

1: create
2: run

:Clock

System : Class

3: runFinalization

asynchronous message

active object