Azzolini Riccardo 2019-11-07

# Interaction diagram: sequence e communication

## 1 Interaction diagram

Gli **interaction diagram** sono diagrammi *dinamici*: descrivono il comportamento dinamico di un gruppo di oggetti,[1] che interagiscono per risolvere un problema.

Tipicamente, questi diagrammi rappresentano il comportamento di uno specifico use case o scenario, in termini di:

- specifiche entità (oggetti);
- messaggi scambiati (metodi).

UML propone due tipi di interaction diagram,

- *sequence diagram*
- *communication diagram*

che però sono praticamente equivalenti.

## 2 Sequence diagram

I **sequence diagram** evidenziano la sequenza temporale delle azioni (cioè dei messaggi scambiati tra gli oggetti partecipanti). Non vengono invece mostrati i link tra oggetti.

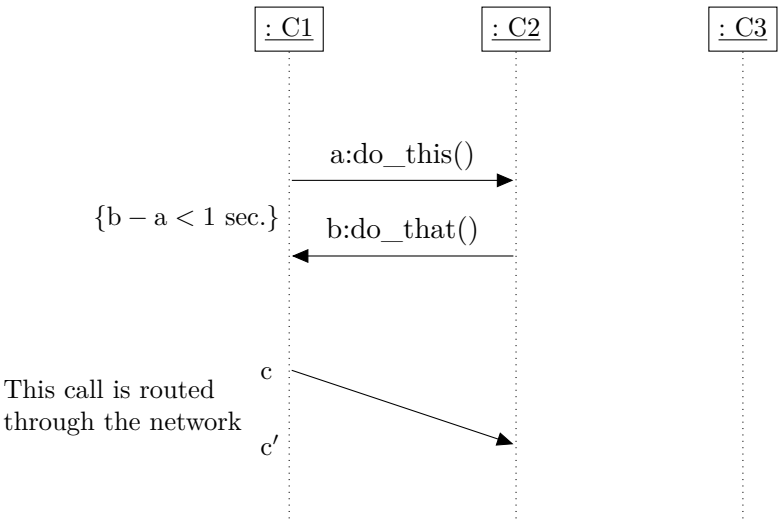
Essi si possono usare in due forme diverse:

- la **forma d'istanza** descrive una singola esecuzione;
- la **forma generica** definisce una famiglia di esecuzioni, specificate tramite una sorta di linguaggio di programmazione visuale.

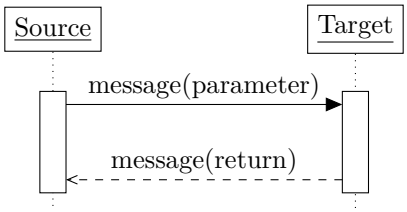
È necessario un bilanciamento tra l'immediatezza espressiva e la completezza, quindi conviene evitare l'uso di diagrammi (soprattutto in forma generica) eccessivamente complessi.

[1] Non è detto che (tutti) gli oggetti siano istanze di classi che faranno parte del sistema. Ad esempio, potrebbero invece essere entità esterne (istanze di attori).

## 2.1 Notazione




- In ascissa sono disposti i vari oggetti (indicati con la stessa notazione usata negli object diagram), e ciascuno è dotato di una propria *lifeline*, una linea tratteggiata che rappresenta la "vita" dell'oggetto.
- Il passare del tempo si rappresenta sull'ordinata, andando verso il basso. In genere, non importa la scala, ma solo l'ordinamento degli eventi. Di conseguenza, se si vogliono specificare dei tempi precisi, è meglio scriverli esplicitamente.
- Se necessario, gli assi orizzontale e verticale possono essere scambiati.
- Le frecce indicano lo scambio di messaggi (cioè le chiamate di metodo) tra oggetti.
- Per ogni messaggio, può essere facoltativamente indicato l'istante di tempo in cui avviene la chiamata, separandolo dal nome con i due punti. Ciò permette di esprimere vincoli temporali. In quest'esempio, l'invocazione di do_this avviene all'istante a, do_that viene invocato all'istante b, e il vincolo $\{b - a < 1 \text{ sec.}\}$ indica che le due chiamate devono avvenire a meno di un secondo di distanza.
- Alcuni messaggi (ad esempio, nei sistemi distribuiti) possono impiegare un certo tempo per essere ricevuti. Ciò si indica con una freccia "storta". Opzionalmente, si possono anche indicare gli istanti di invio e ricezione del messaggio (in questo esempio, sono rispettivamente c e $c'$), in modo da poter specificare dei vincoli su di essi.
- È possibile aggiungere dei commenti (ad esempio, "This call is routed through the network") per dare ulteriori informazioni.

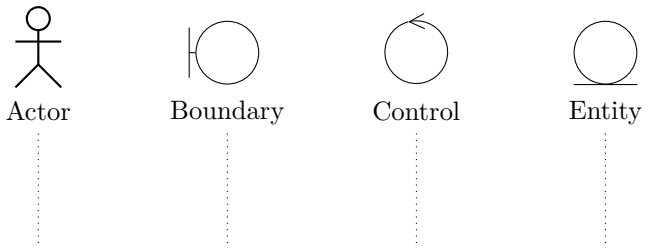



- I rettangoli disegnati sulle lifeline rappresentano l'esecuzione di un metodo in risposta a un messaggio.
- Si può indicare esplicitamente la risposta a un messaggio, cioè la restituzione di un risultato (ma, spesso, essa viene lasciata implicita).

## 2.2 Stereotipi di oggetti

Gli oggetti possono essere indicati con degli stereotipi, utili per rappresentare, ad esempio:

- istanze di attori[2] (quando il sequence diagram è posseduto da uno use case);
- oggetti di boundary;
- oggetti di controllo;
- oggetti entity.




Gli oggetti entity, control e boundary sono i componenti del pattern MVC:

- il *Model* è il modello dei dati, costituito da entità;
- la *View* è l'interfaccia, che è un oggetto boundary;
- il *Control* gestisce le interazioni.

In questo modo, si costruisce un sistema gerarchico, nel quale ogni oggetto può comunicare solo con i suoi "vicini" nella gerarchia:

$$\text{attore} \longleftrightarrow \text{interfaccia} \longleftrightarrow \text{controllo} \longleftrightarrow \text{entità}$$

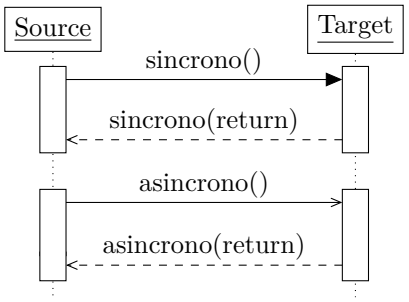
[2] Anche se il simbolo usato è lo stesso, in uno use case diagram un attore identifica un ruolo, mentre in un sequence diagram si indica un'istanza di un attore, cioè una specifica entità che ricopre tale ruolo.

## 2.3 Messaggi sincroni e asincroni

Un messaggio può essere:

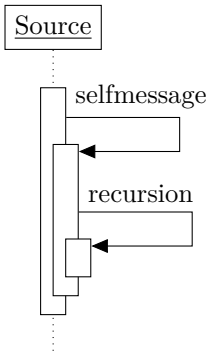
**sincrono**: il chiamante attende la fine dell'operazione (metodo) prima di proseguire;

**asincrono**: il chiamante intanto va avanti, poi otterrà il risultato dell'operazione quando essa terminerà.




## 2.4 Auto-messaggi

Un oggetto può mandare messaggi a se stesso, sia per invocare metodi diversi, sia per invocare lo stesso metodo, cioè realizzare la ricorsione.



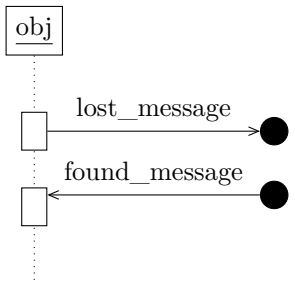

## 2.5 Messaggi lost e found

Un messaggio è **lost** se:

- non arriva al destinatario stabilito, oppure
- non se ne conosce il destinatario, oppure
- il destinatario non è mostrato nel diagramma (anche solo per motivi grafici, ad esempio perché esso è rappresentato in un'altra pagina).

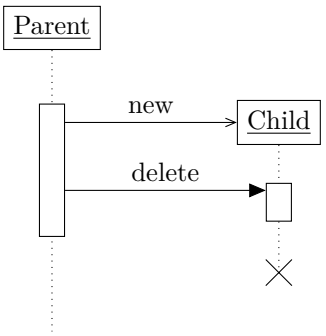
Simmetricamente, un messaggio è **found** se il mittente:

- è sconosciuto, oppure
- non è indicato nel diagramma.




## 2.6 Inizio e fine di lifeline

Non è detto che un oggetto rimanga "in vita" per tutta la durata dell'esecuzione raffigurata da un sequence diagram. Infatti, gli oggetti possono essere creati e distrutti:



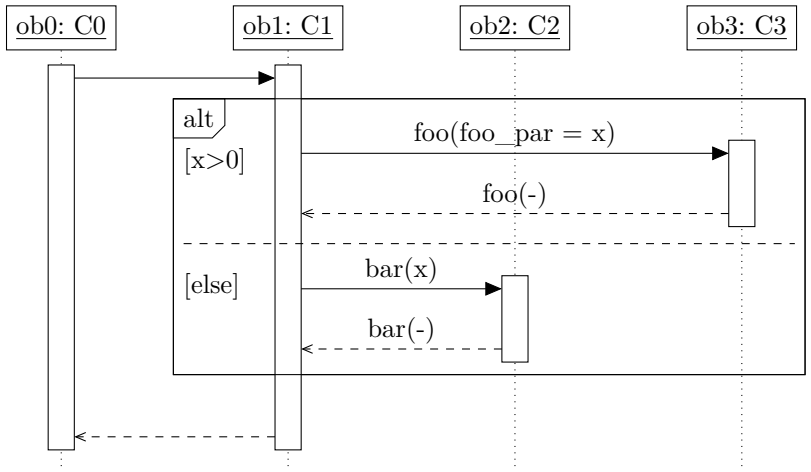

## 2.7 Combined fragment

I **combined fragment** (*frammenti combinati*) si usano per realizzare i sequence diagram in forma generica: essi permettono di "programmare" in modo visuale, per rappresentare più esecuzioni all'interno di un singolo diagramma.

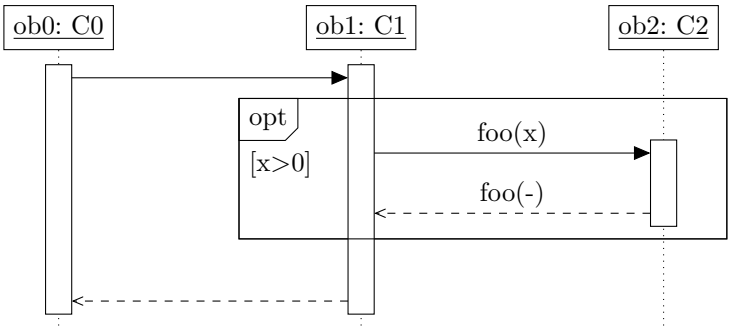
I principali tipi di combined fragment sono:

- **alt**: una scelta di comportamento. Definisce due o più opzioni di esecuzione, ciascuna con una guardia (cioè una condizione booleana, che può essere espressa anche in linguaggio naturale):
  - se nessuna delle guardie è vera, non viene eseguita alcuna opzione;

- se è verificata la guardia di una sola delle opzioni, viene eseguita unicamente tale opzione;
- se più opzioni hanno guardie vere, l'opzione da eseguire viene scelta in modo non specificato (non deterministico) tra quelle con le guardie vere.

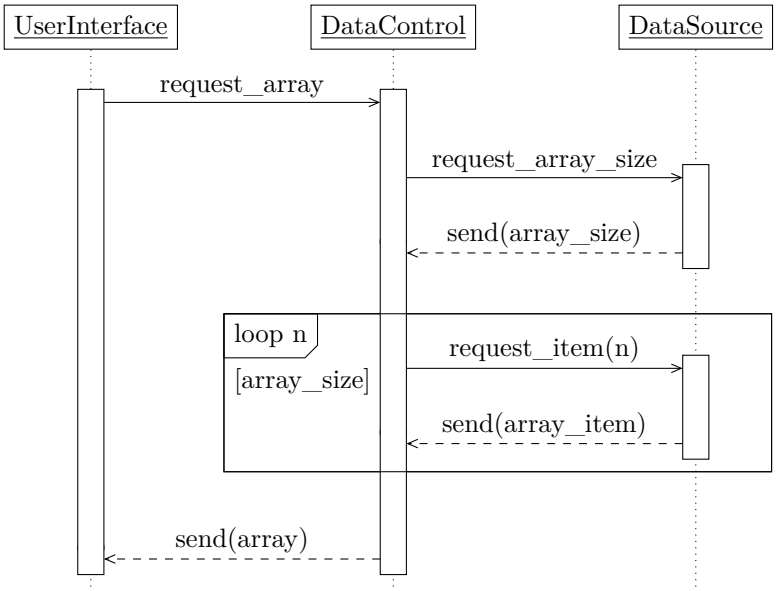



- **opt**: equivale a un alt con una sola opzione, che può essere eseguita o meno, a seconda della guardia.




- **loop**: ripete più volte l'esecuzione del corpo, in base a una guardia che può contenere:
  - una condizione booleana;
  - un numero minimo di iterazioni, da effettuare a prescindere dalla condizione;
  - un numero massimo di iterazioni (quelle oltre il numero minimo vengono effettuate solo fintanto che la condizione è vera).

  È anche possibile assegnare il numero dell'iterazione a una variabile, in modo da poterlo usare, ad esempio, come parametro dei messaggi.




# 3 Communication diagram

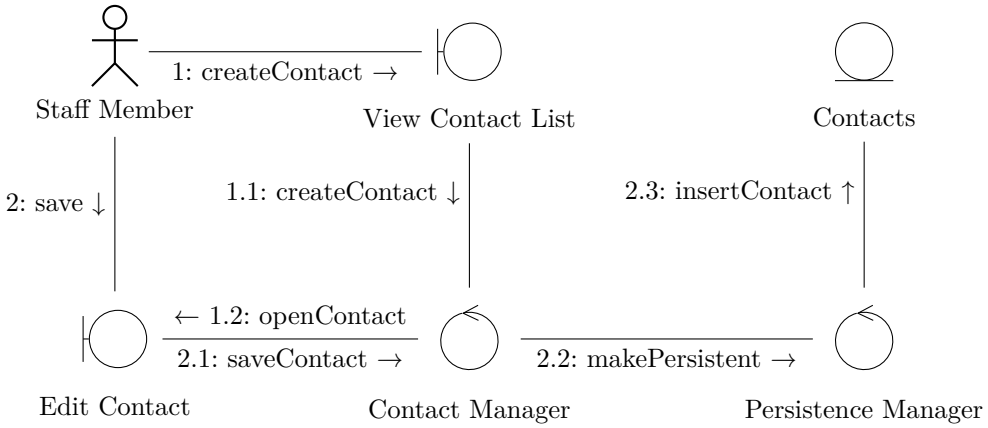
I **communication diagram** (chiamati *collaboration diagram* nelle versioni precedenti di UML) sono simili ai sequence diagram, ma evidenziano i link e le interazioni tra gli oggetti, ponendo maggiore attenzione allo scambio di messaggi. Essi sono adatti per:

- invocazioni innestate;
- interazioni "sofisticate";
- concorrenza e thread.

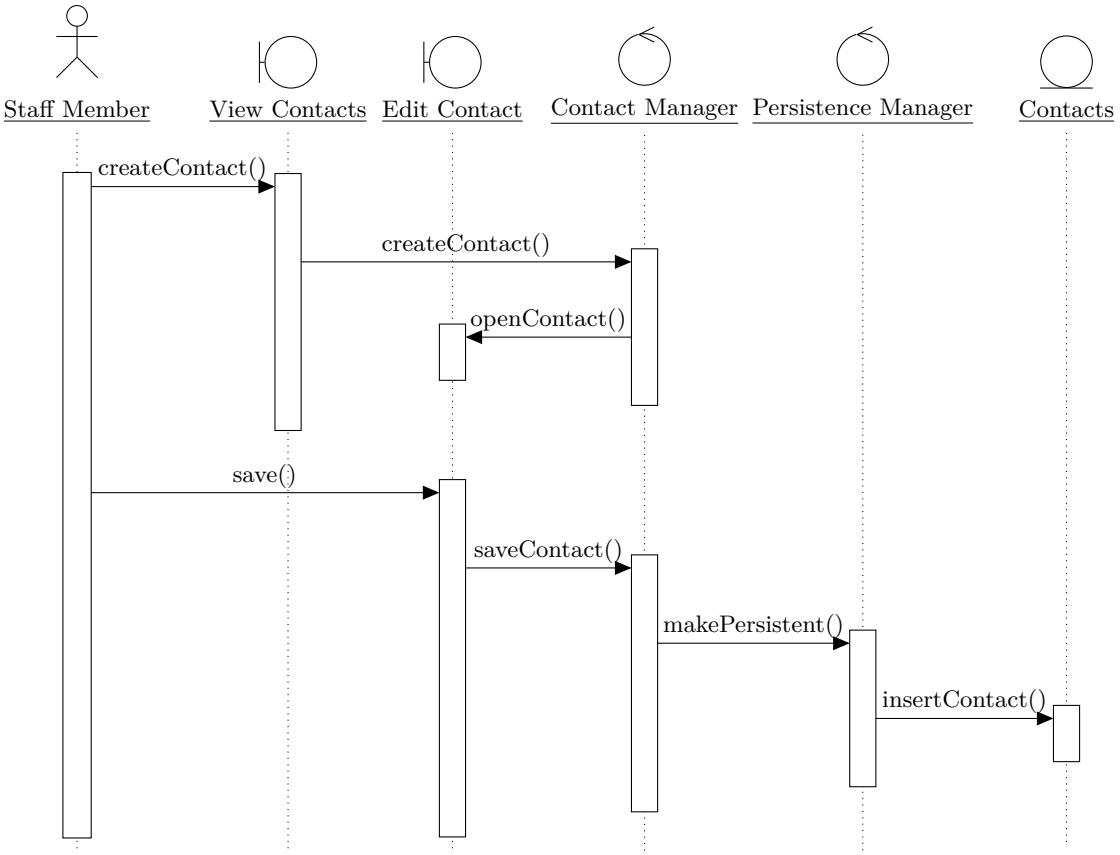
Il tempo non è associato a una dimensione precisa: la sequenza degli scambi di messaggi è data da una numerazione esplicita. Inoltre, è possibile modellare sequenze alternative sullo stesso diagramma.

Anche la vita degli oggetti non è indicata esplicitamente (a differenza dei sequence diagram).

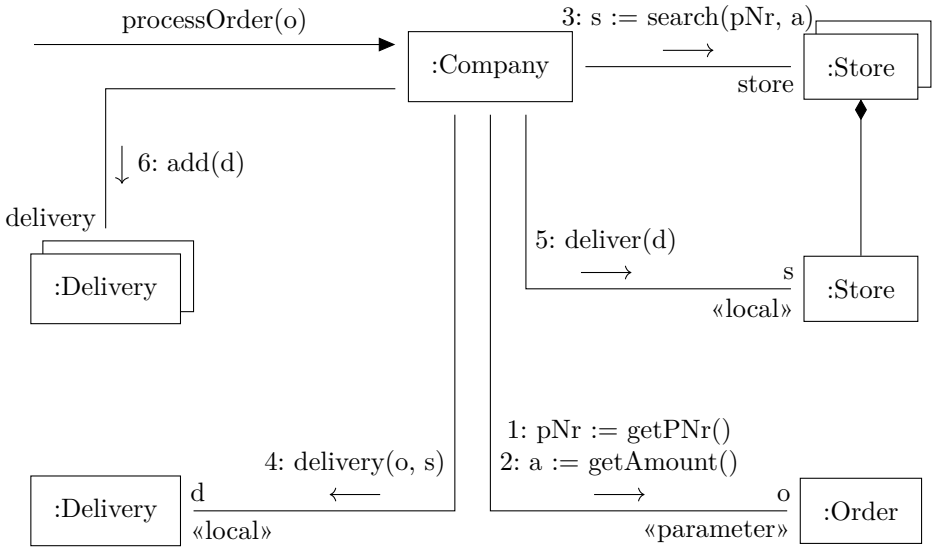
### 3.1 Esempio 1




Il sequence diagram corrispondente è:

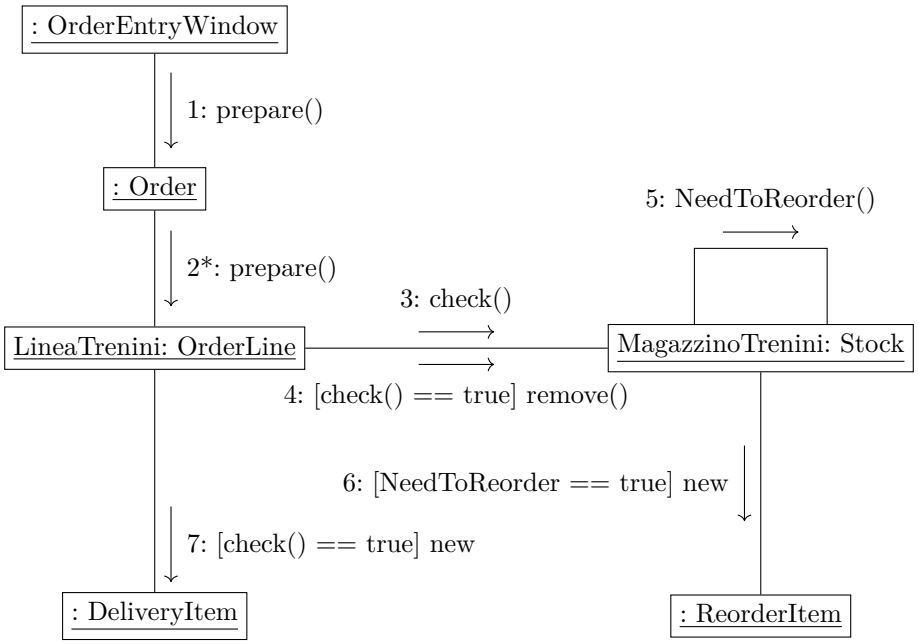


### 3.2 Esempio 2



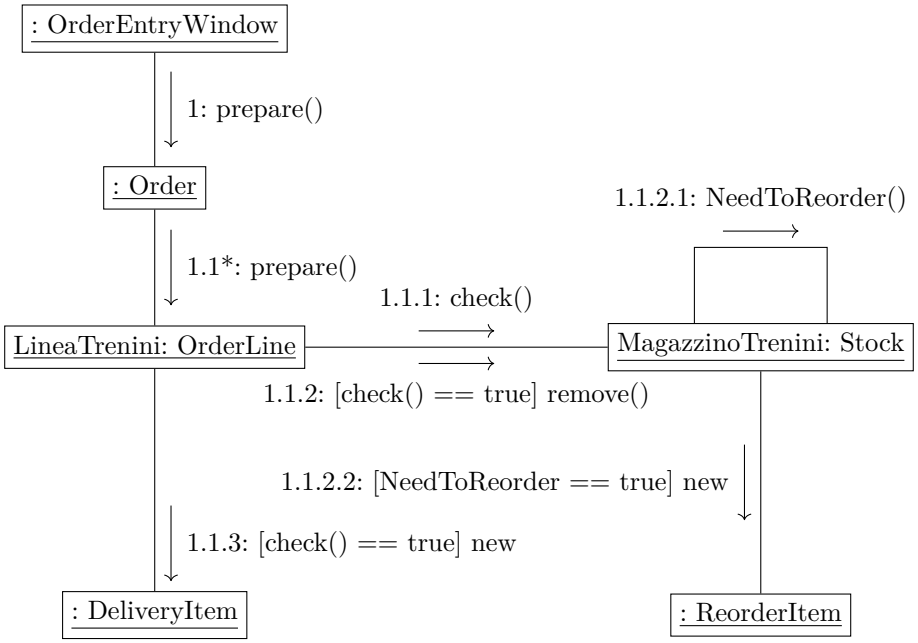

- Lo stereotipo «parameter» indica che l'oggetto o è quello passato come parametro al metodo processOrder.
- I risultati di alcune delle chiamate di metodo (1, 2 e 3) vengono salvati in delle variabili, usate poi come argomenti in altre chiamate.
- Gli stereotipi «local» indicano che gli oggetti s e d sono assegnati a variabili locali.
- I due rettangoli sovrapposti rappresentano una collezione di oggetti dello stesso tipo.

### 3.3 Esempio 3




- L'asterisco indica che l'azione 2 si ripete.
- Alcune azioni vengono eseguite solo in determinate condizioni, indicate tra parentesi quadre.
- L'azione 5 è un'*auto-delega*, perché l'oggetto MagazzinoTrenini chiama un metodo appartenente a se stesso.

Per indicare esplicitamente quale operazione chiama quale altra, si può utilizzare una numerazione annidata:


: OrderEntryWindow
1: prepare()
: Order
1.1.2.1: NeedToReorder()
1.1*: prepare()
1.1.1: check()
LineaTrenini: OrderLine
MagazzinoTrenini: Stock
1.1.2: [check() == true] remove()
1.1.2.2: [NeedToReorder == true] new
1.1.3: [check() == true] new
: DeliveryItem
: ReorderItem