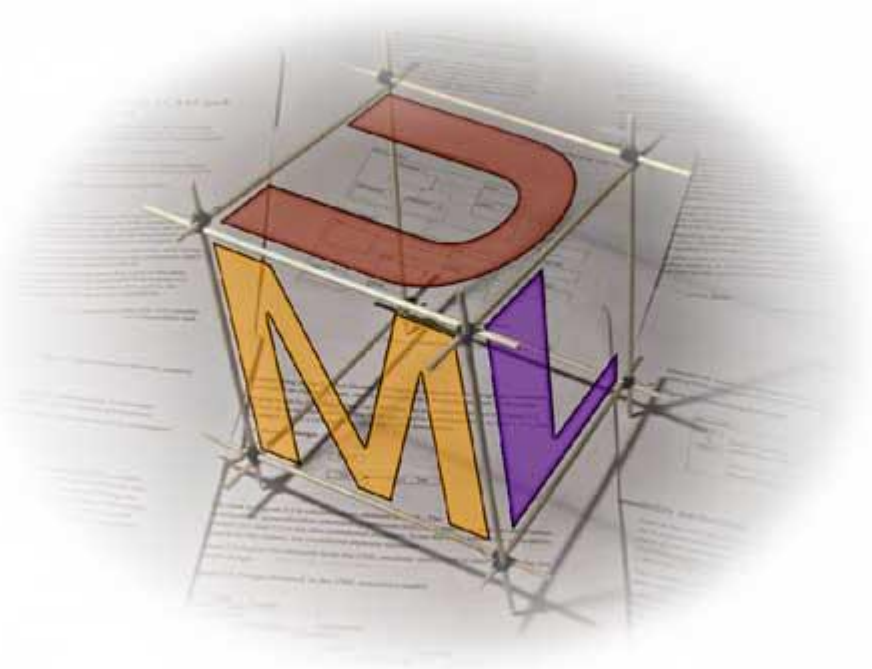
UML e i diagrammi di sequenza

Sintassi e Linee Guida

Dr. Andrea Baruzzo

andrea.baruzzo@dimi.uniud.it


UML

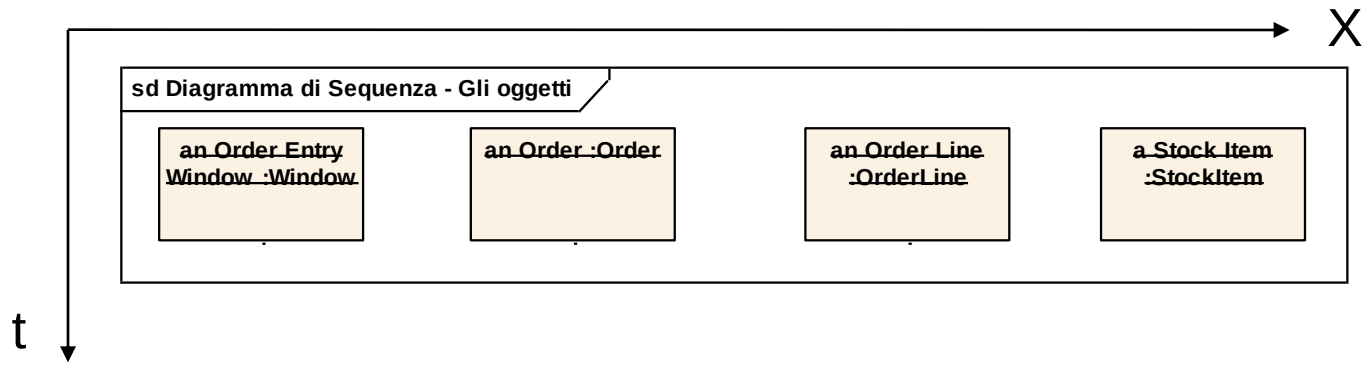
Alcune definizioni

- Un diagramma di sequenza è un diagramma che descrive **interazioni** tra oggetti che **collaborano** per svolgere un compito
- Gli oggetti collaborano scambiandosi **messaggi**
- Lo scambio di un messaggio in OOP equivale all'invocazione di un metodo

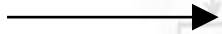
Gli oggetti

- **Asse x:**
  - Gli oggetti sono disposti orizzontalmente
  - Un oggetto è un'istanza di una classe
  - Sintassi: nomeOggetto : NomeClasse

- **Asse t:**
  - Il flusso del tempo è descritto verticalmente



Scambio Messaggi Sincroni (1/2)

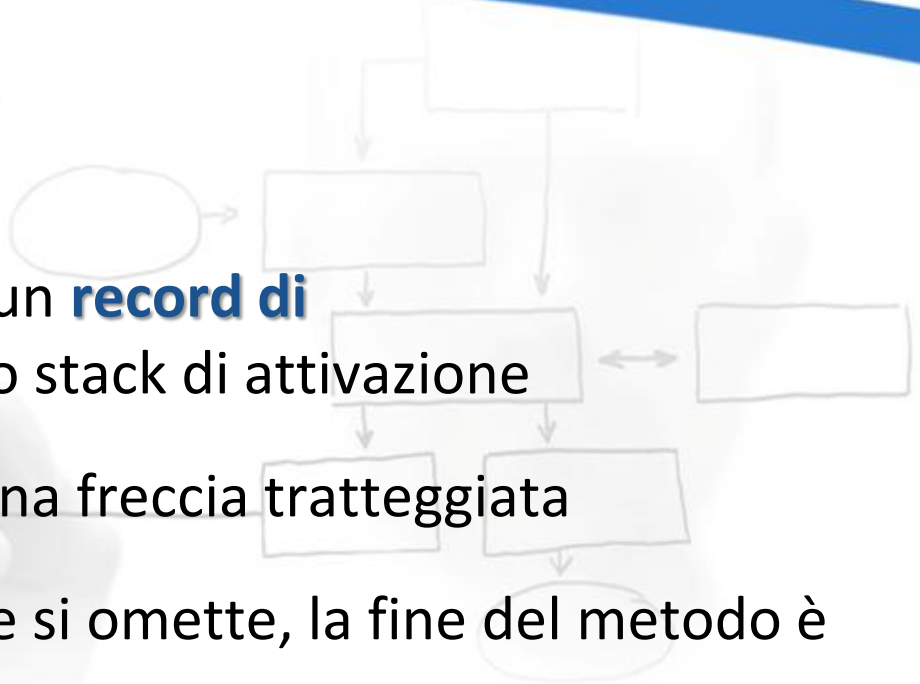
- Si disegna con una **freccia chiusa** 

  da chiamante a chiamato. La freccia è etichettata col nome del metodo invocato, e opzionalmente i suoi parametri e il suo valore di ritorno

- Il chiamante attende la terminazione del metodo del chiamato prima di proseguire

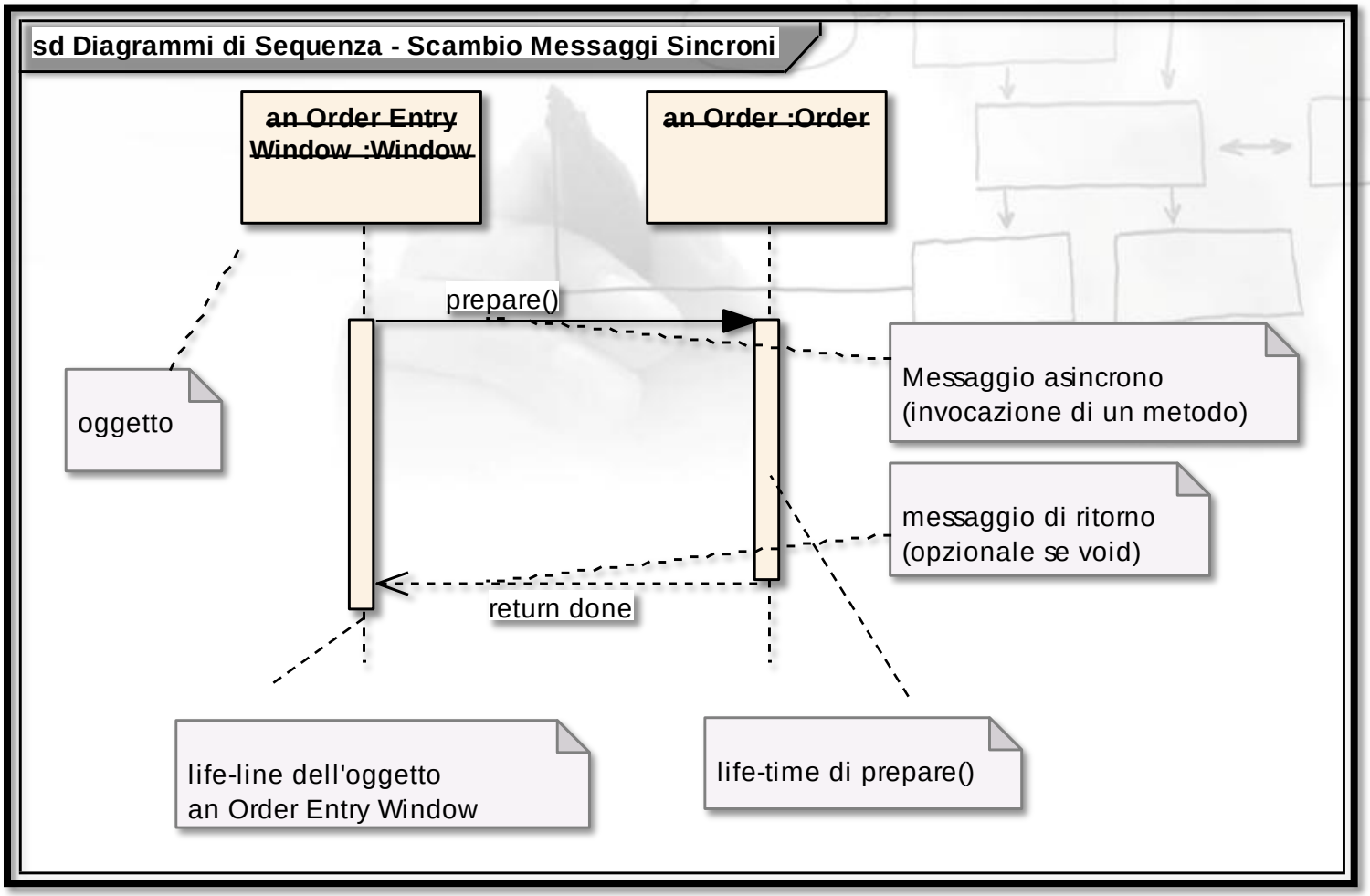
- Il **life-time** (durata, vita) di un metodo è rappresentato da un rettangolino che collega freccia di invocazione e freccia di ritorno

Scambio Messaggi Sincroni (2/2)

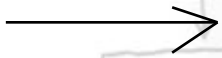


- Life-time corrisponde ad avere un **record di attivazione** di quel metodo sullo stack di attivazione
- Il **ritorno** è rappresentato con una freccia tratteggiata
- Il ritorno è sempre opzionale. Se si omette, la fine del metodo è decretata dalla fine del life-time

Scambio Messaggi Sincroni – esempio

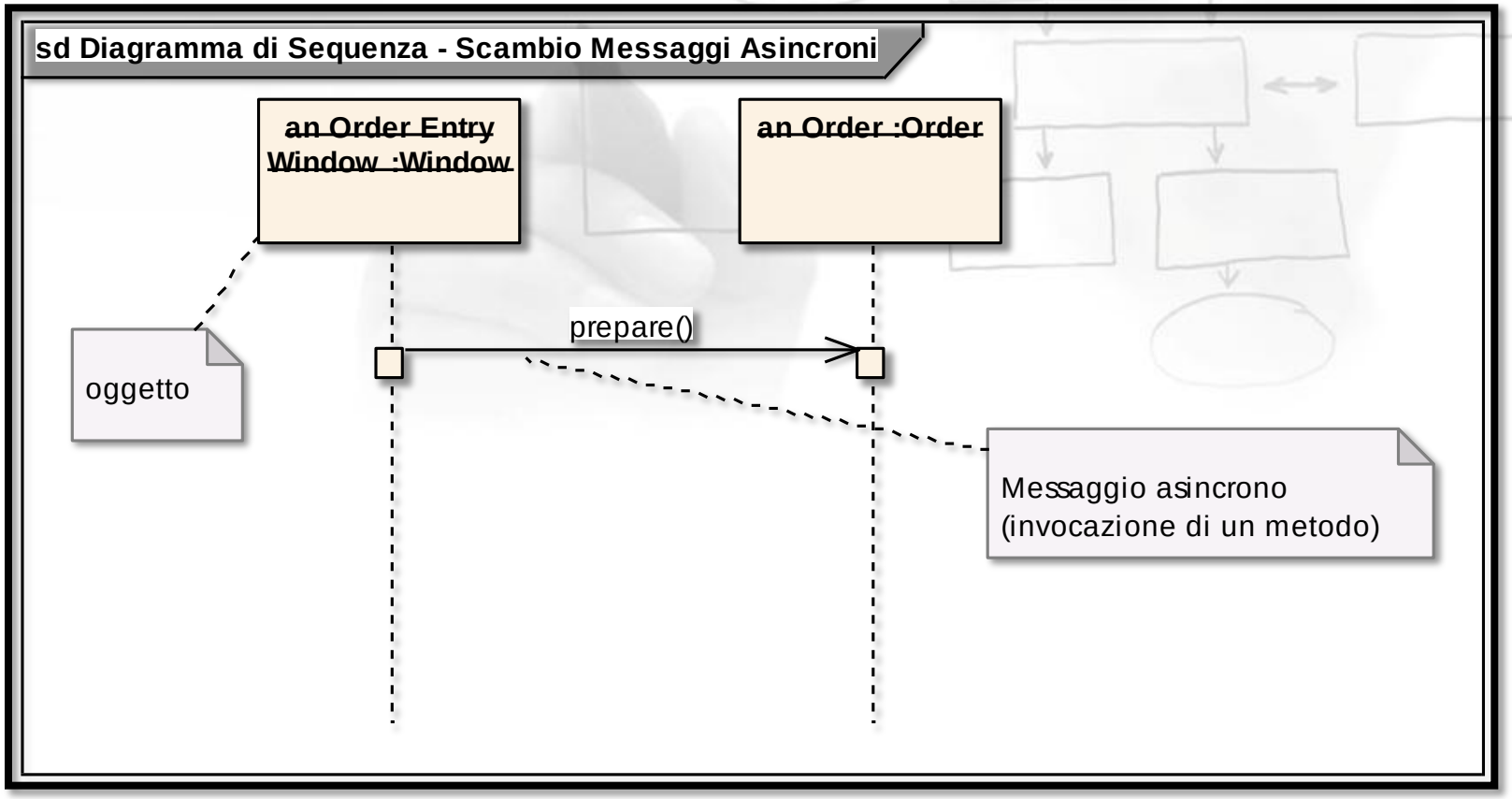


Scambio Messaggi Asincroni

- Si usano per descrivere interazioni concorrenti
- Si disegna con una **freccia aperta** 

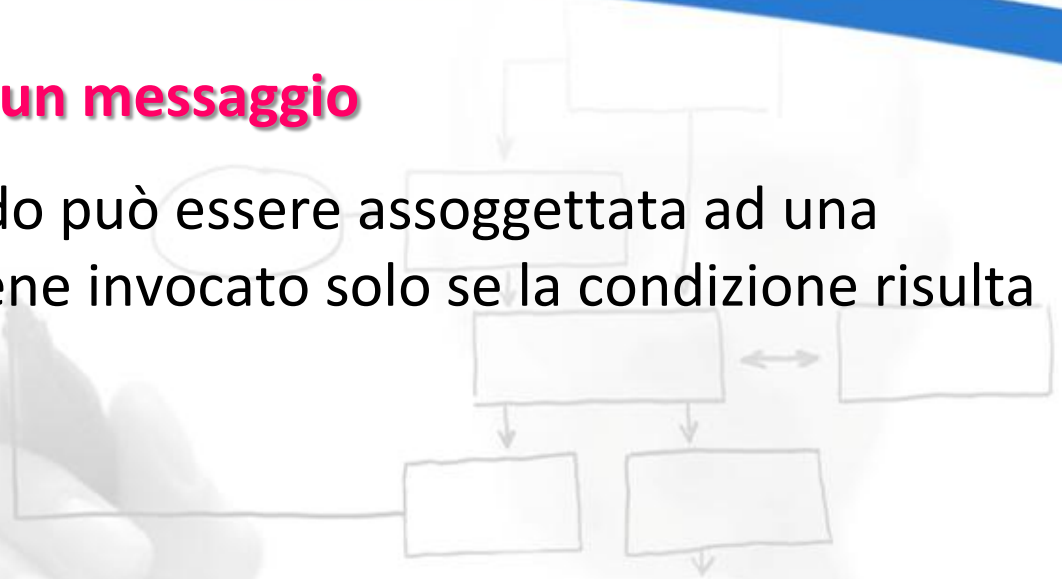
  da chiamante a chiamato. La freccia è etichettata col nome del metodo invocato, e opzionalmente i suoi parametri e il suo valore di ritorno)
- Il chiamante non attende la terminazione del metodo del chiamato, ma prosegue subito dopo l'invocazione (implementazione tipica mediante code)
- Il ritorno può non essere immediatamente successivo alla chiamata

Scambio Messaggi Asincroni – esempio



Esecuzione condizionale di un messaggio

- L'esecuzione di un metodo può essere assoggettata ad una **condizione**. Il metodo viene invocato solo se la condizione risulta verificata a run-time

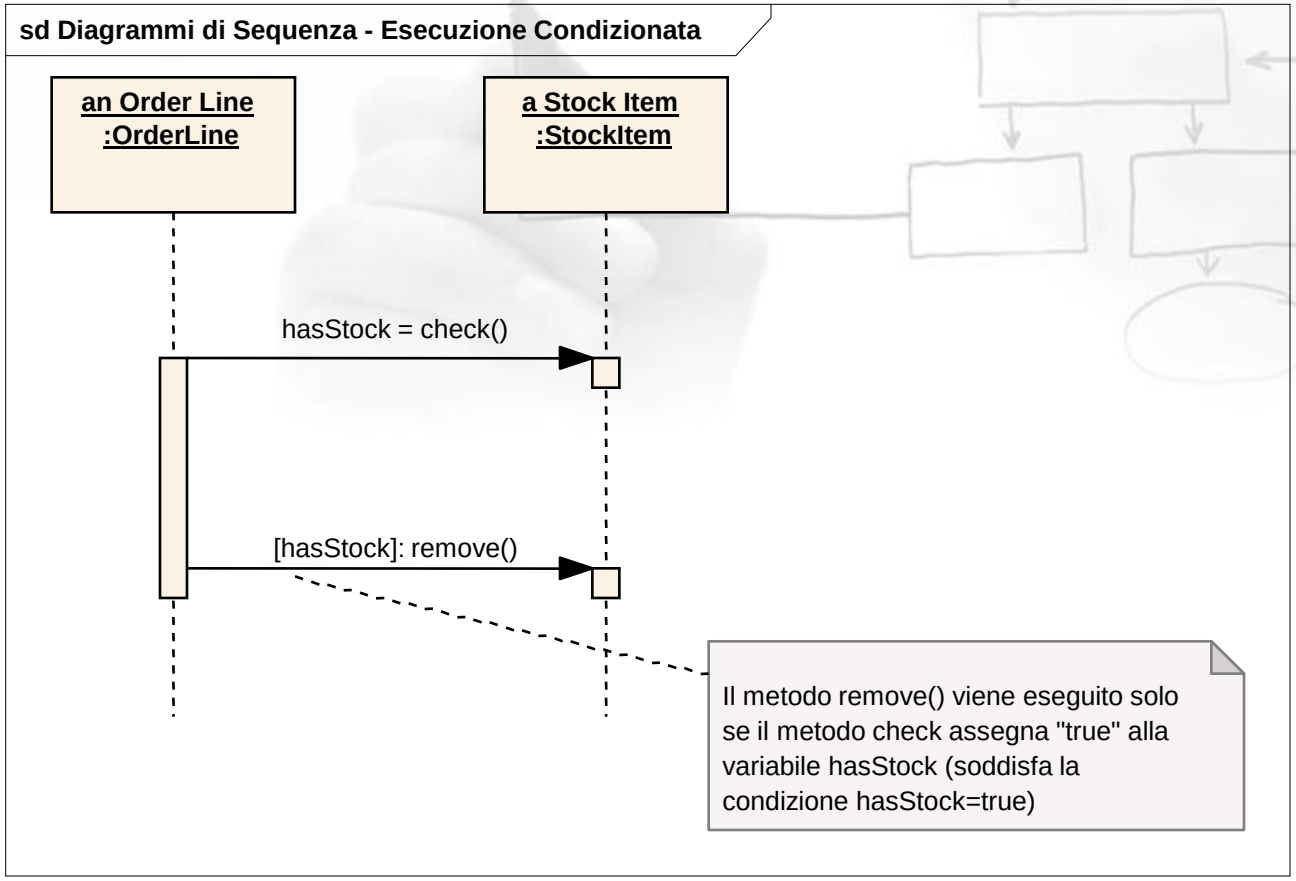


- Si disegna aggiungendo la condizione, racchiusa tra parentesi quadre, che definisce quando viene eseguito il metodo

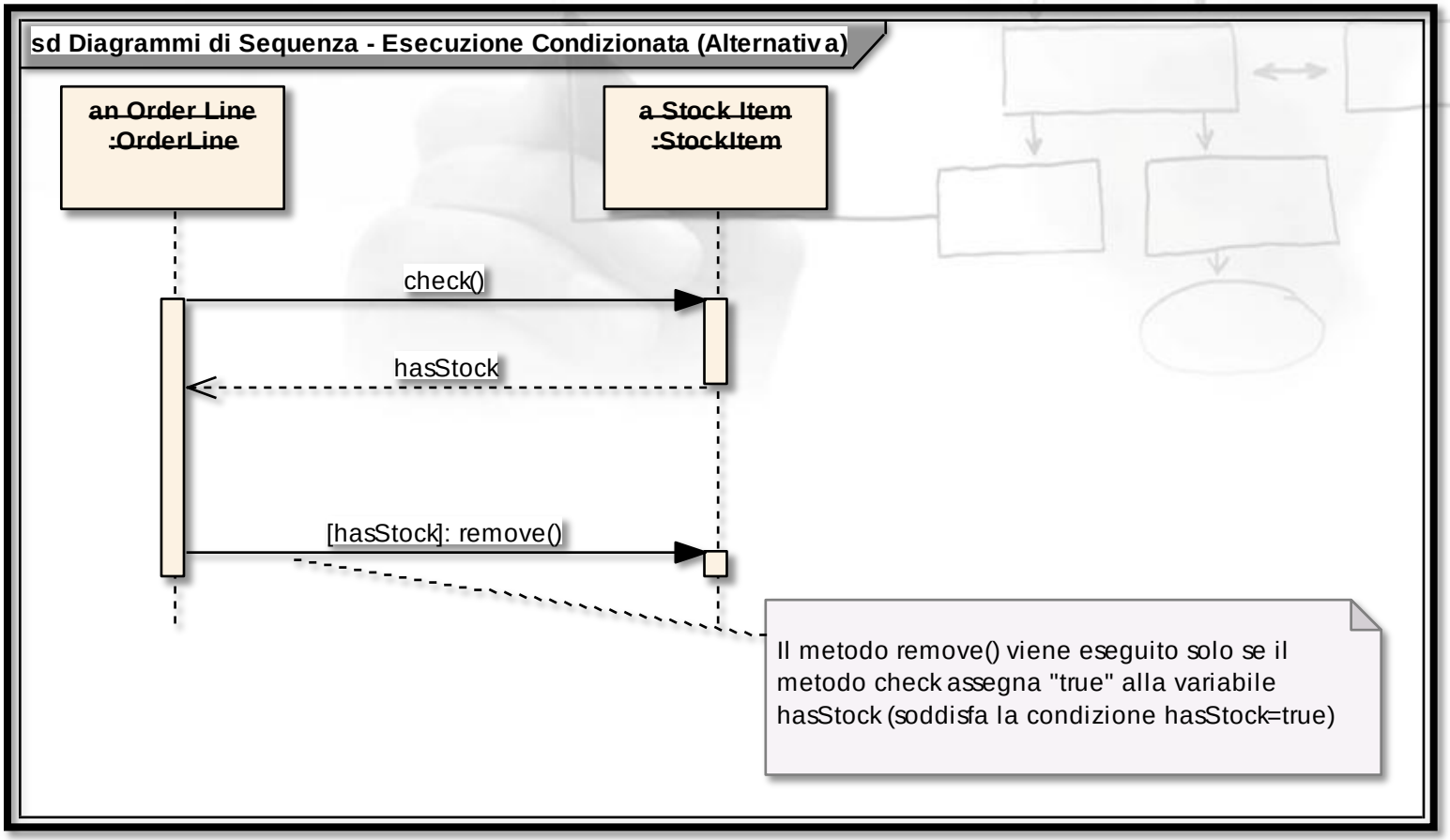
- Sintassi:

[cond] : nomeMetodo()

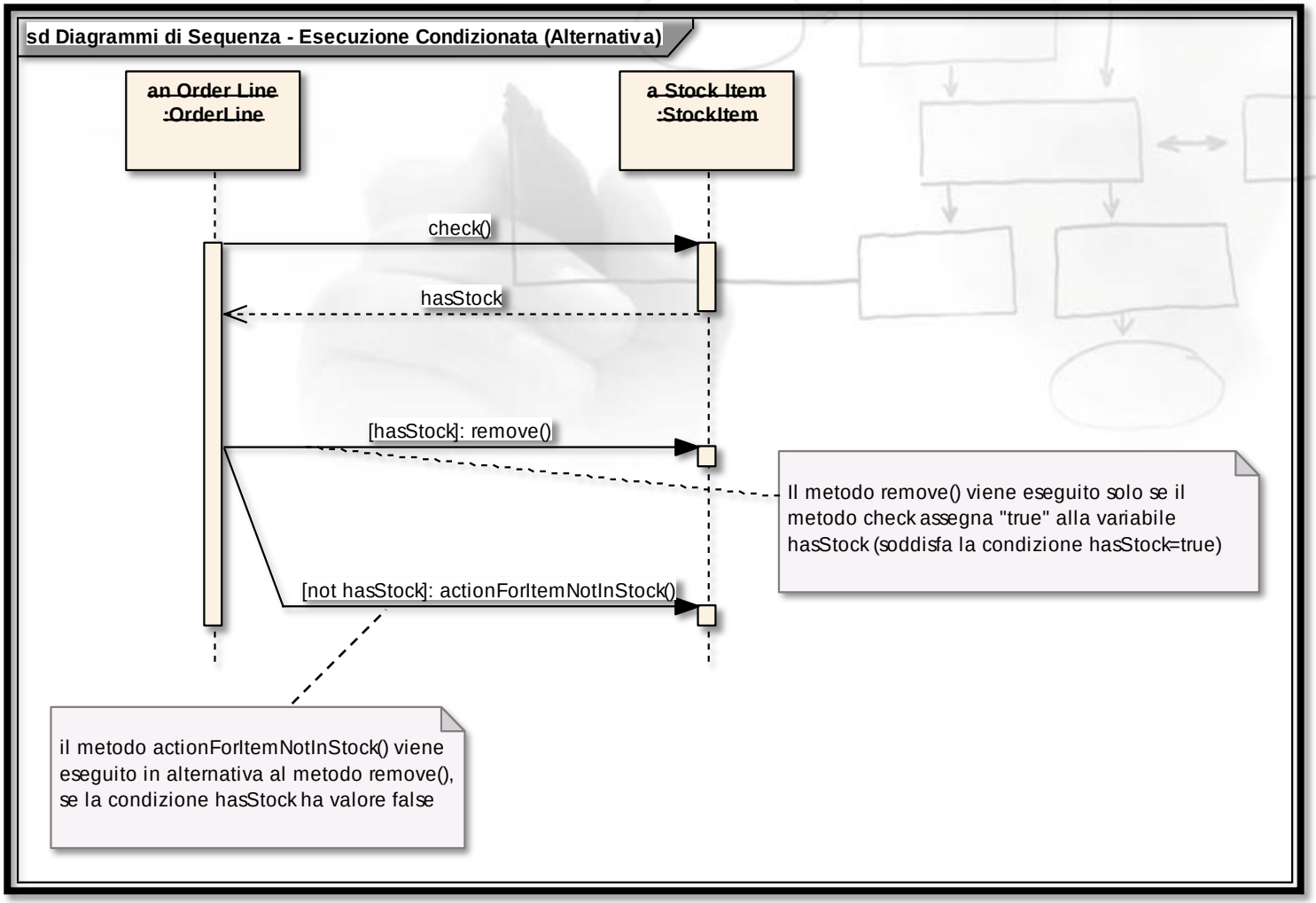
Esecuzione condizionale di un messaggio - esempio



Esecuzione condizionata di un messaggio – esempio (un'alternativa)



Esecuzione condizionata di un messaggio – esempio (un'altra alternativa)

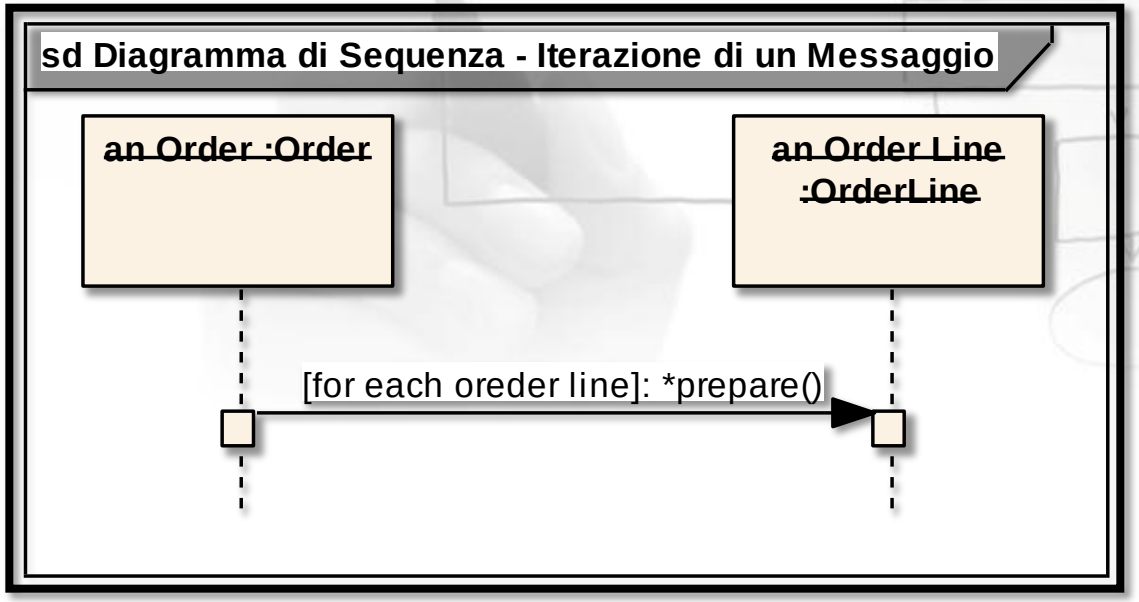


Iterazione di un messaggio

- Rappresenta l'**esecuzione ciclica** di messaggi
- Si disegna aggiungendo un * (asterisco) prima del metodo su cui si vuole iterare
- Si può aggiungere la condizione che definisce l'iterazione
- La condizione si rappresenta tra parentesi quadre. Sintassi completa:

[cond] : * nomeMetodo()

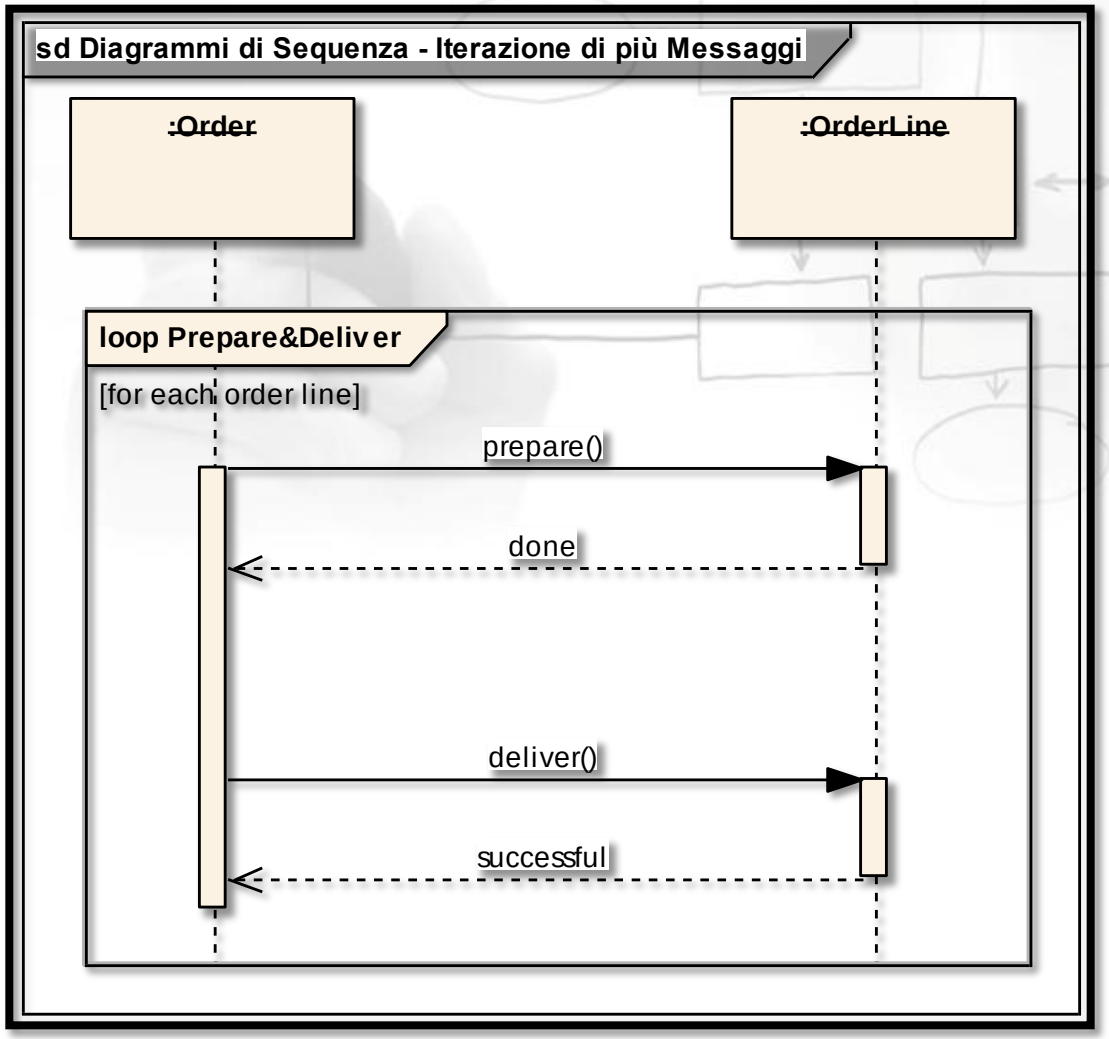
Iterazione di un messaggio - esempio



Iterazione di un blocco di messaggi

- Rappresenta l'esecuzione ciclica di più messaggi
- Si disegna raggruppando con un **blocco** (riquadro, box) i messaggi (metodi) su cui si vuole iterare
- Si può aggiungere la condizione che definisce l'iterazione sull'angolo in alto a sinistra del blocco
- La condizione si rappresenta al solito tra parentesi quadre

Iterazione di un blocco di messaggi - esempio

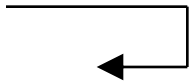


“Auto-Chiamata” (Self-Call)

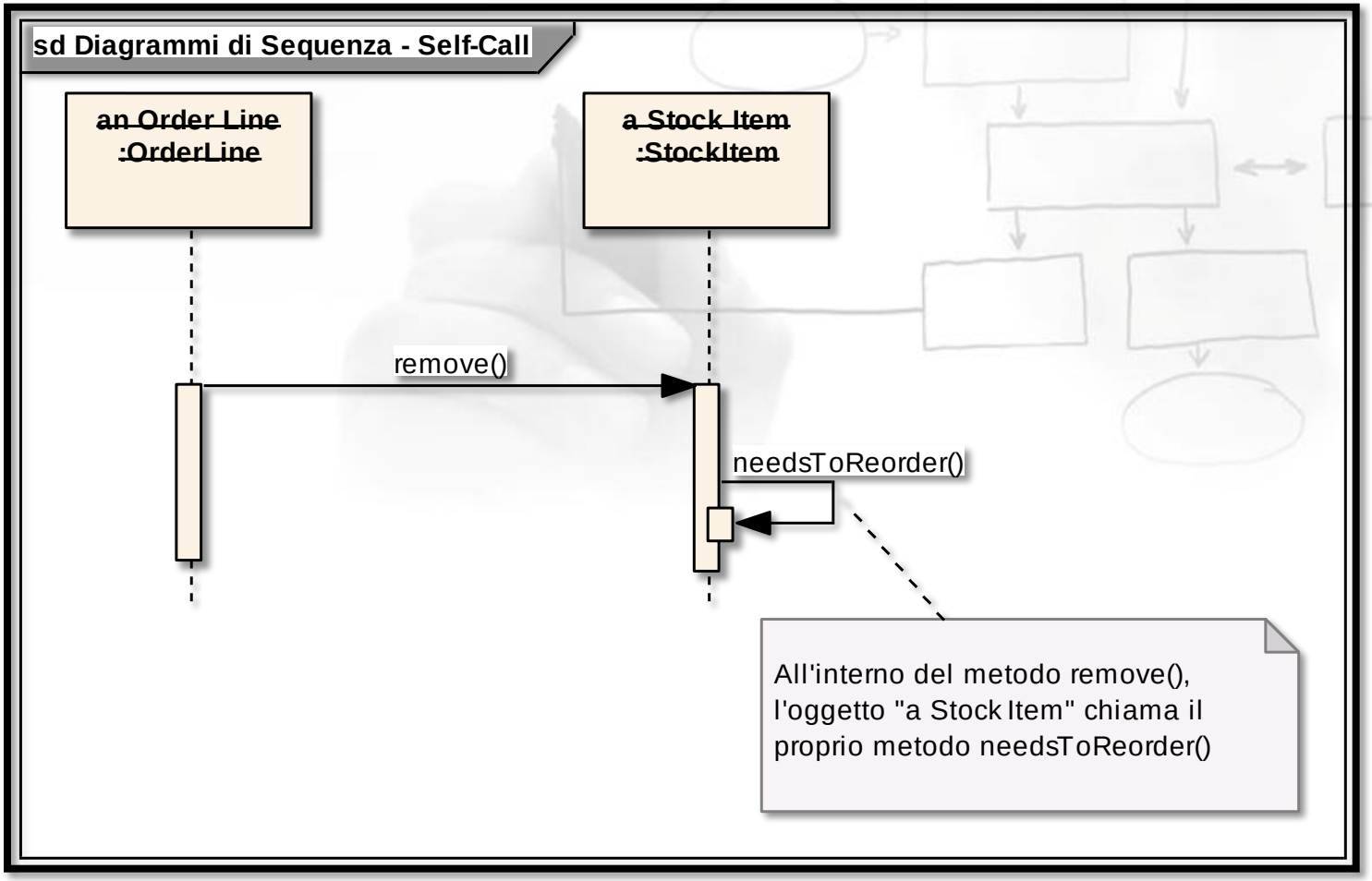
- Descrive un oggetto che invoca un suo metodo (chiamante e chiamato coincidono)

- Si rappresenta con una “**freccia circolare**”

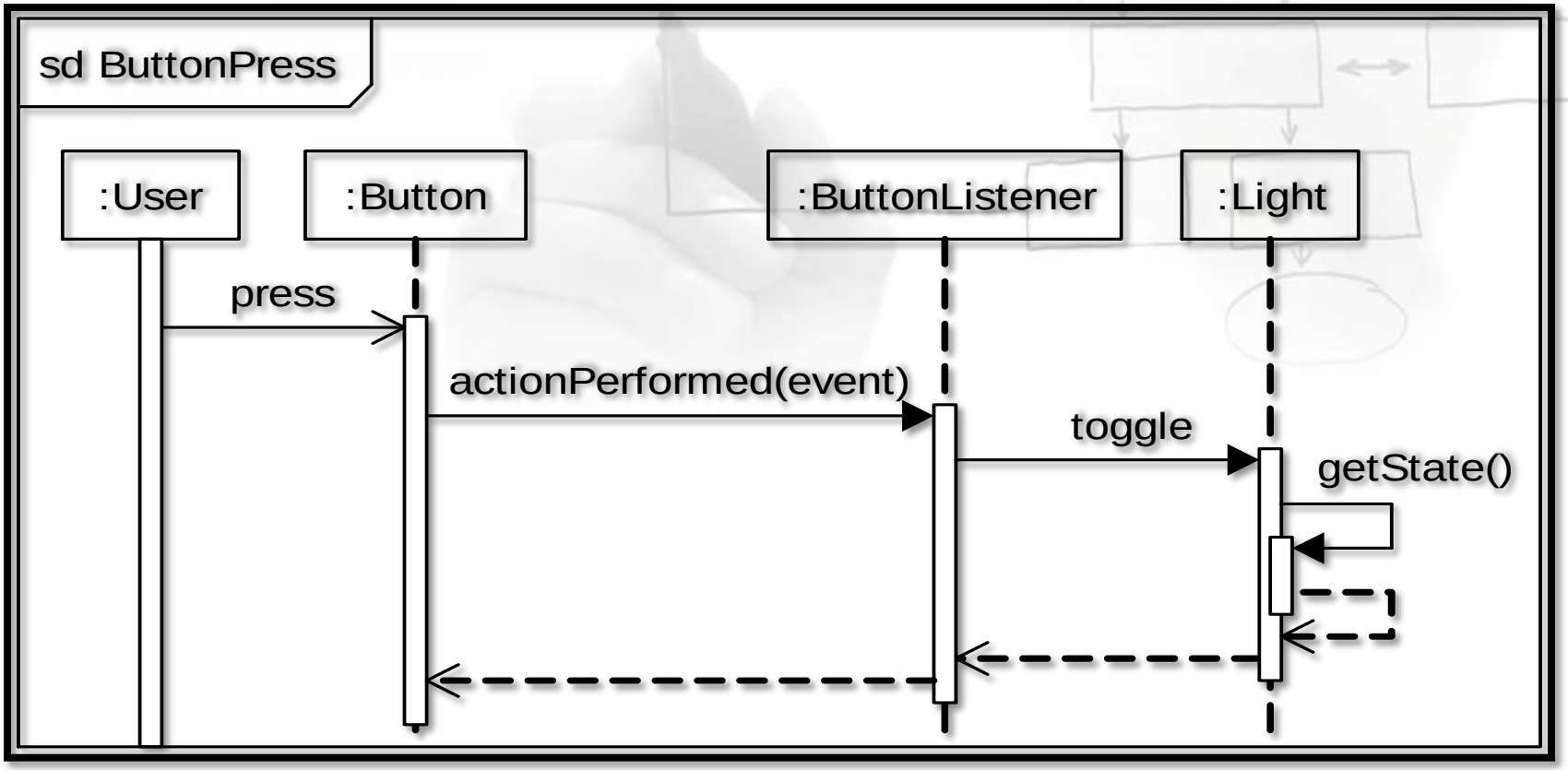
  che rimane all’interno del life time di uno stesso metodo



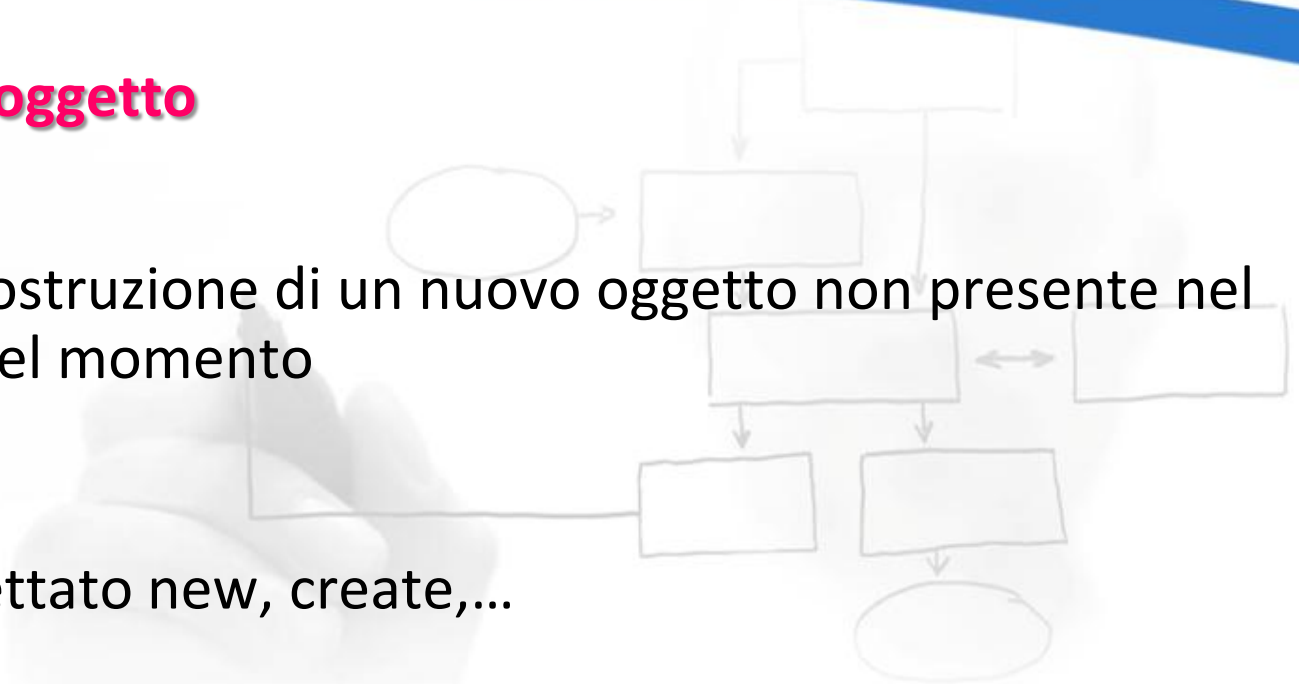
Auto-Chiamata” (Self-Call) - esempio



Un esempio composto

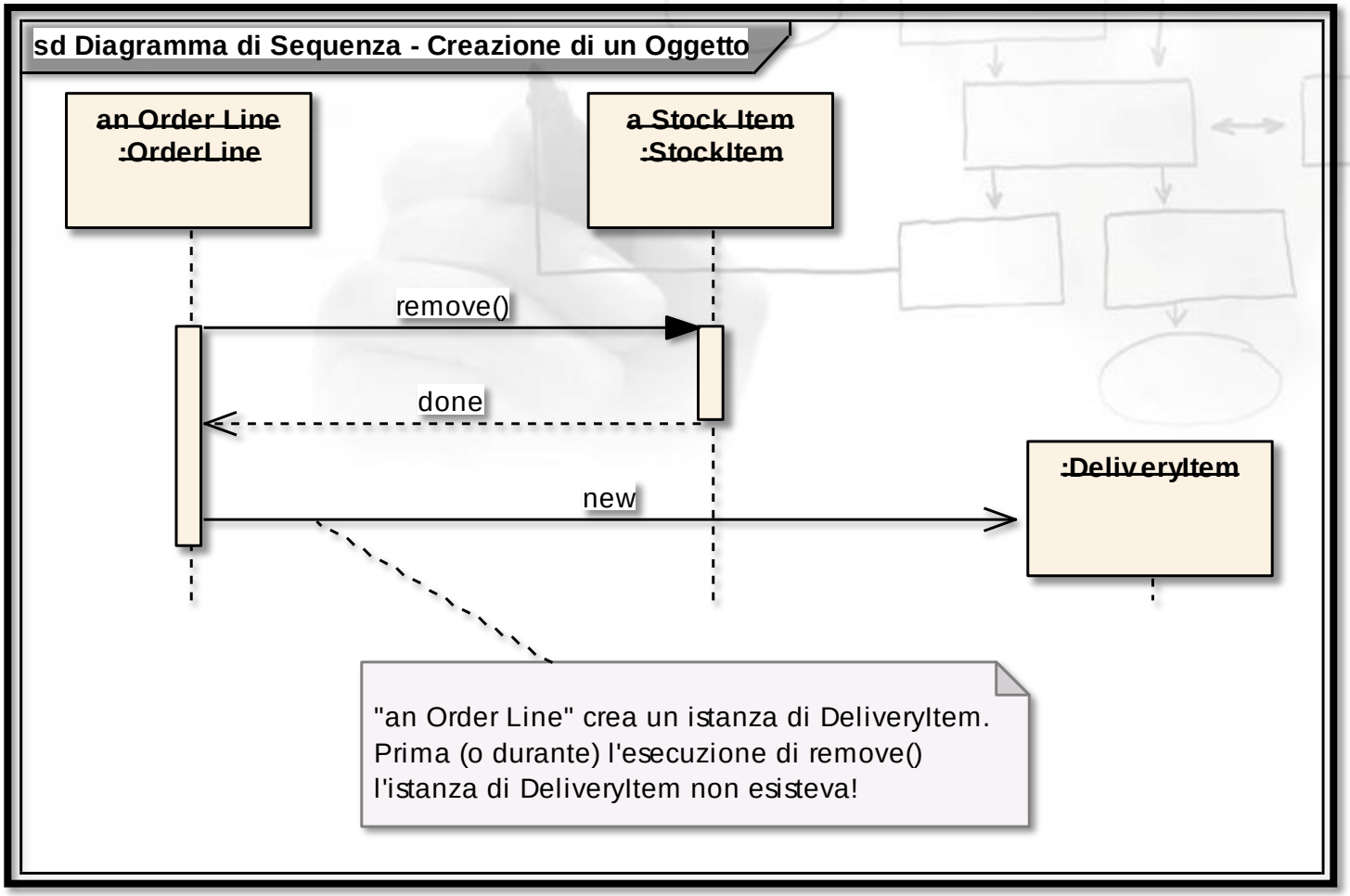


Costruzione di un oggetto

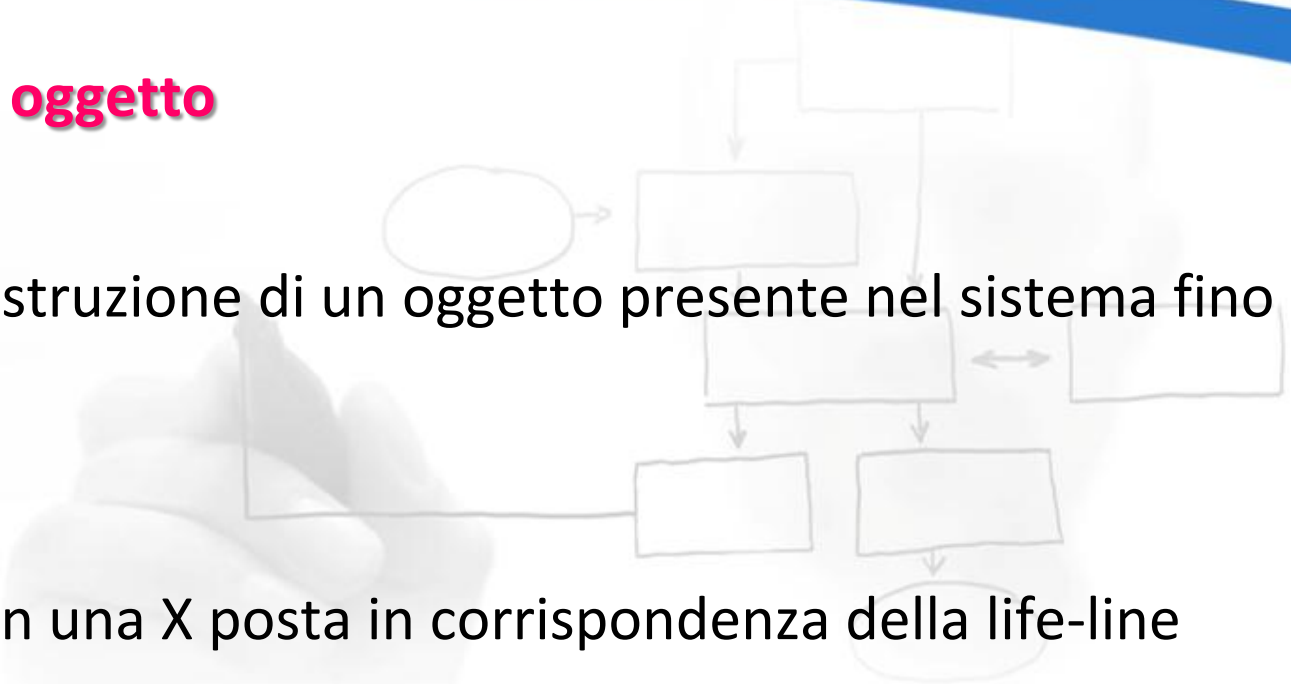


- Rappresenta la costruzione di un nuovo oggetto non presente nel sistema fino a quel momento
- Messaggio etichettato new, create,…
- L'oggetto viene collocato nell'asse temporale in corrispondenza dell'invocazione nel metodo new (o create…)

Costruzione di un oggetto - esempio

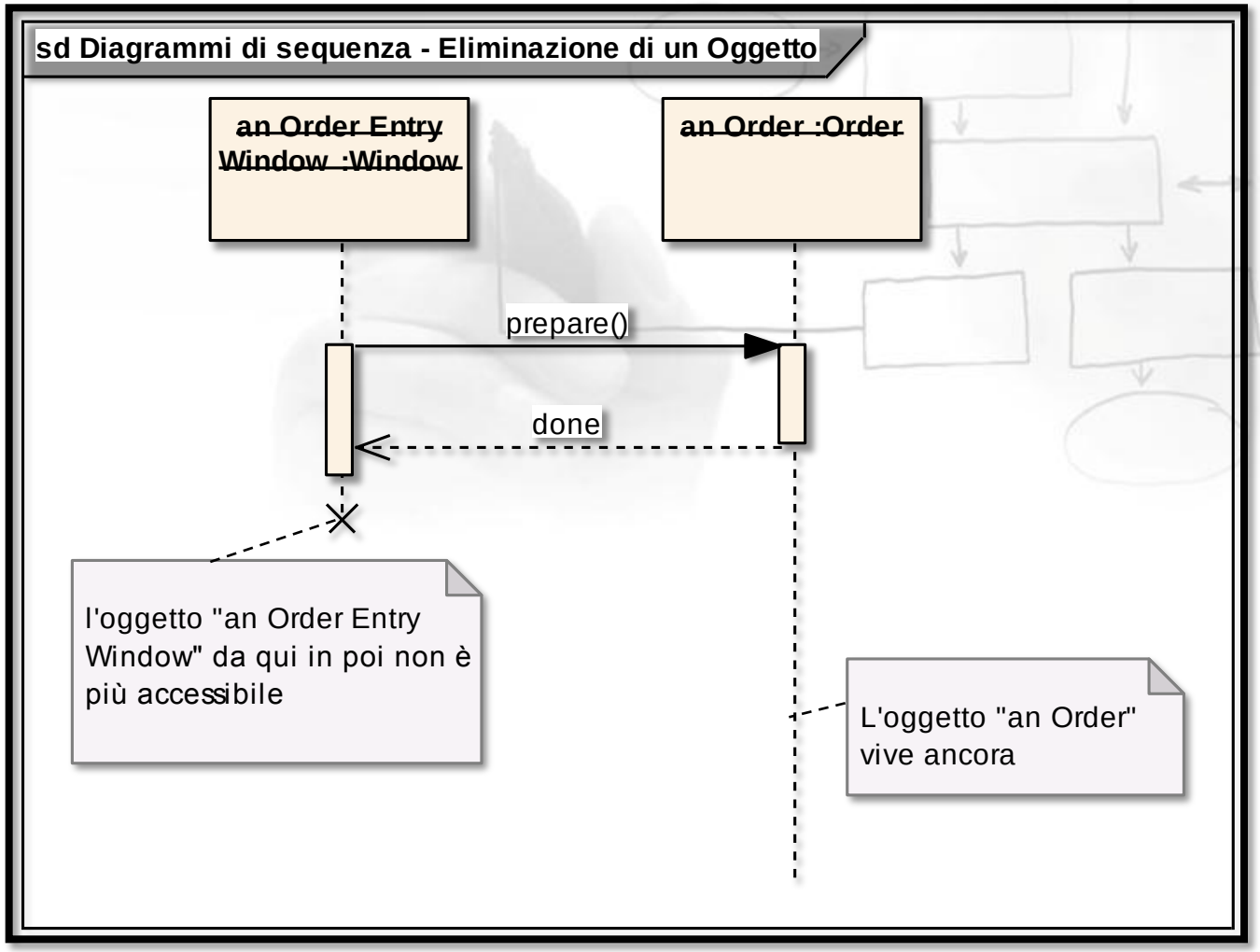


Eliminazione di un oggetto

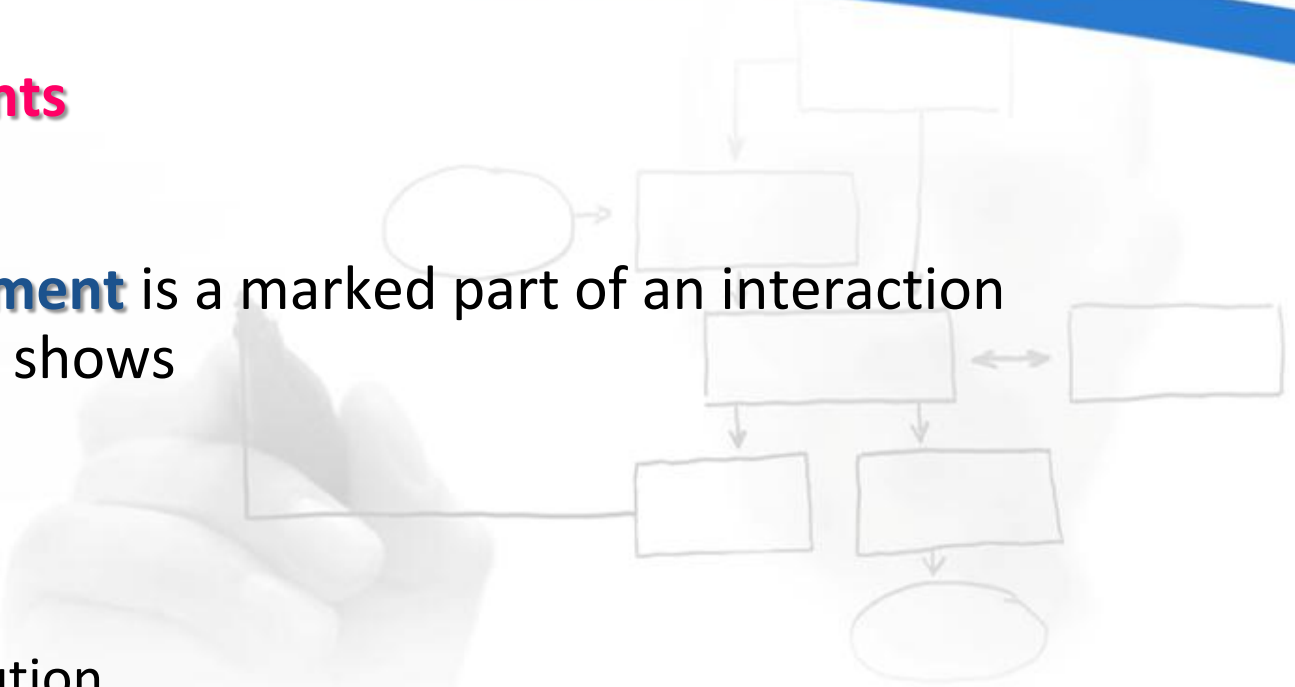


- Rappresenta la distruzione di un oggetto presente nel sistema fino a quel momento
- Si rappresenta con una X posta in corrispondenza della life-line dell'oggetto
- Da quel momento in poi non è legale invocare alcun metodo dell'oggetto distrutto

Eliminazione di un oggetto - esempio

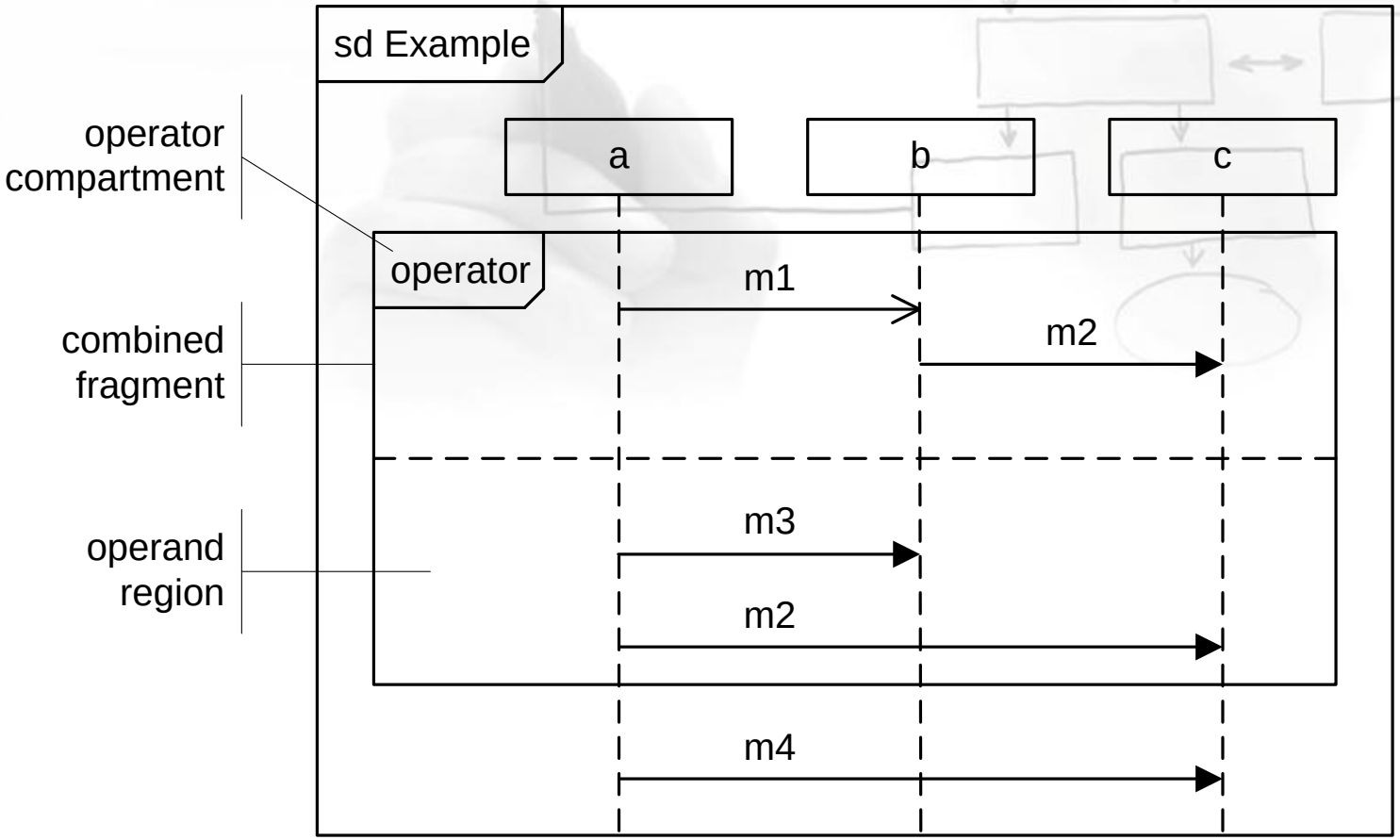


Combined fragments

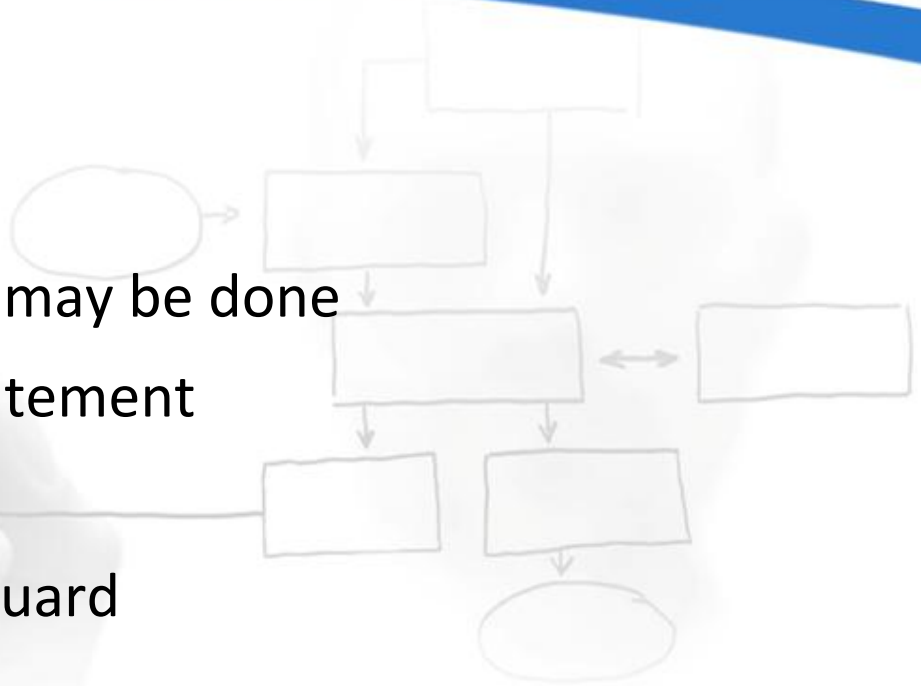


- A **combined fragment** is a marked part of an interaction specification that shows
  - Branching,
  - Loops,
  - Concurrent execution,
  - And so forth
- It is surrounded by a rectangular frame.
  - Pentagonal operation compartment
  - Dashed horizontal line forming regions holding operands

Combined fragment layout



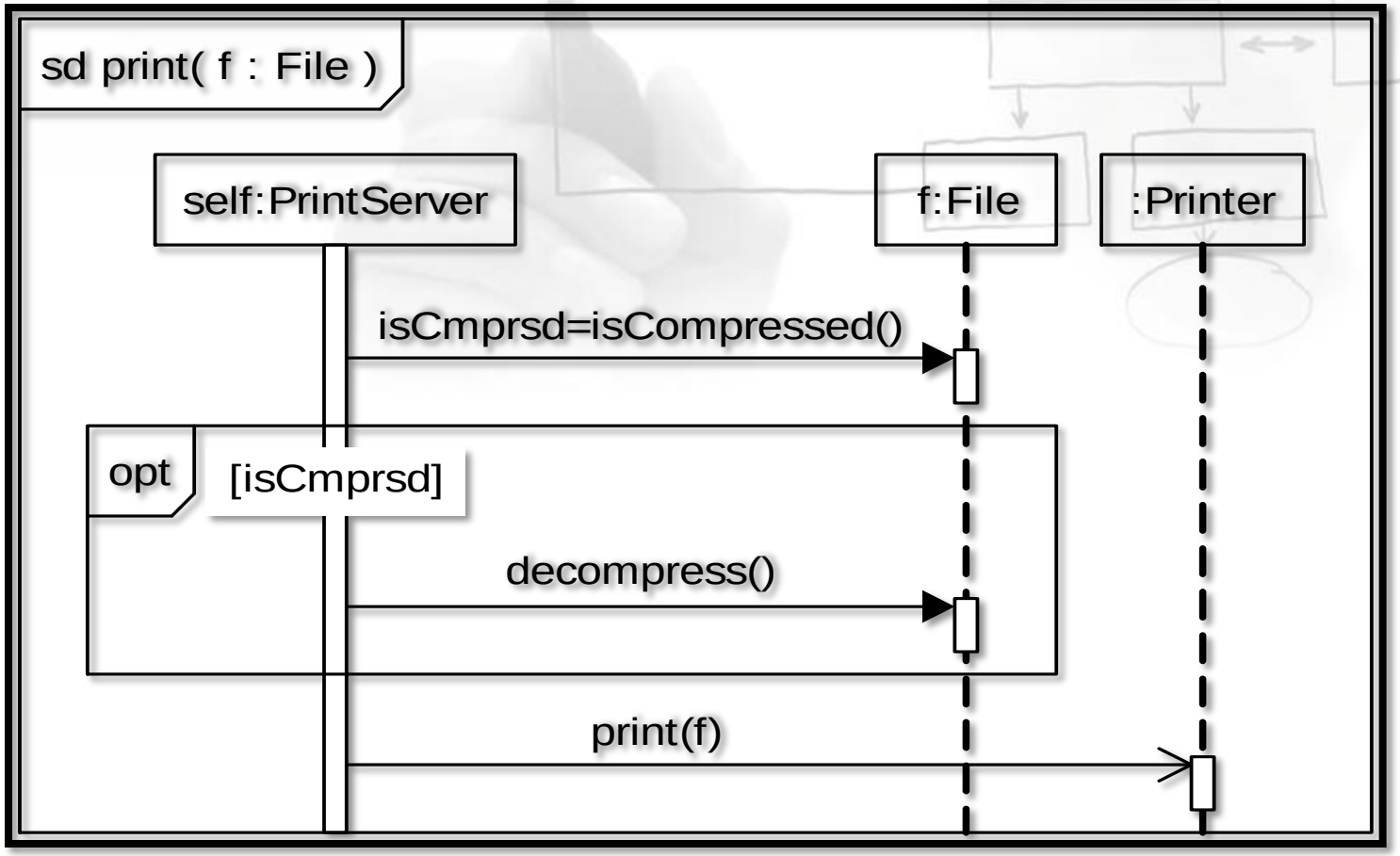
Optional fragment



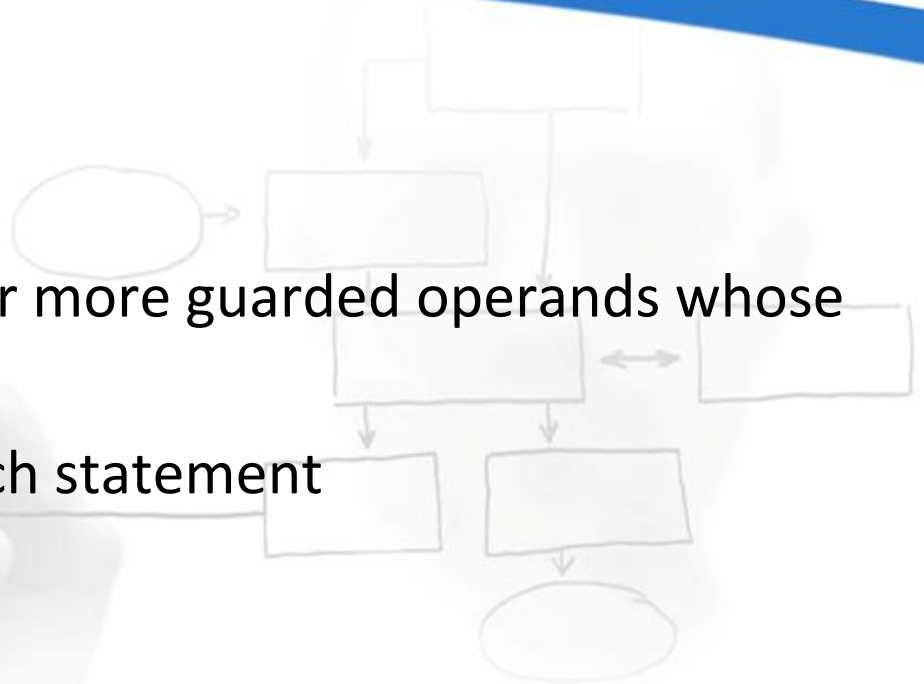
- A portion of an interaction that may be done
  - Equivalent to a conditional statement
  - Operator is the keyword opt
  - Only a single operand with a guard

- A **guard** is a Boolean expression in square brackets in a format not specified by UML.
  - [else] is a special guard true if every guard in a fragment is false.

Optional fragment example



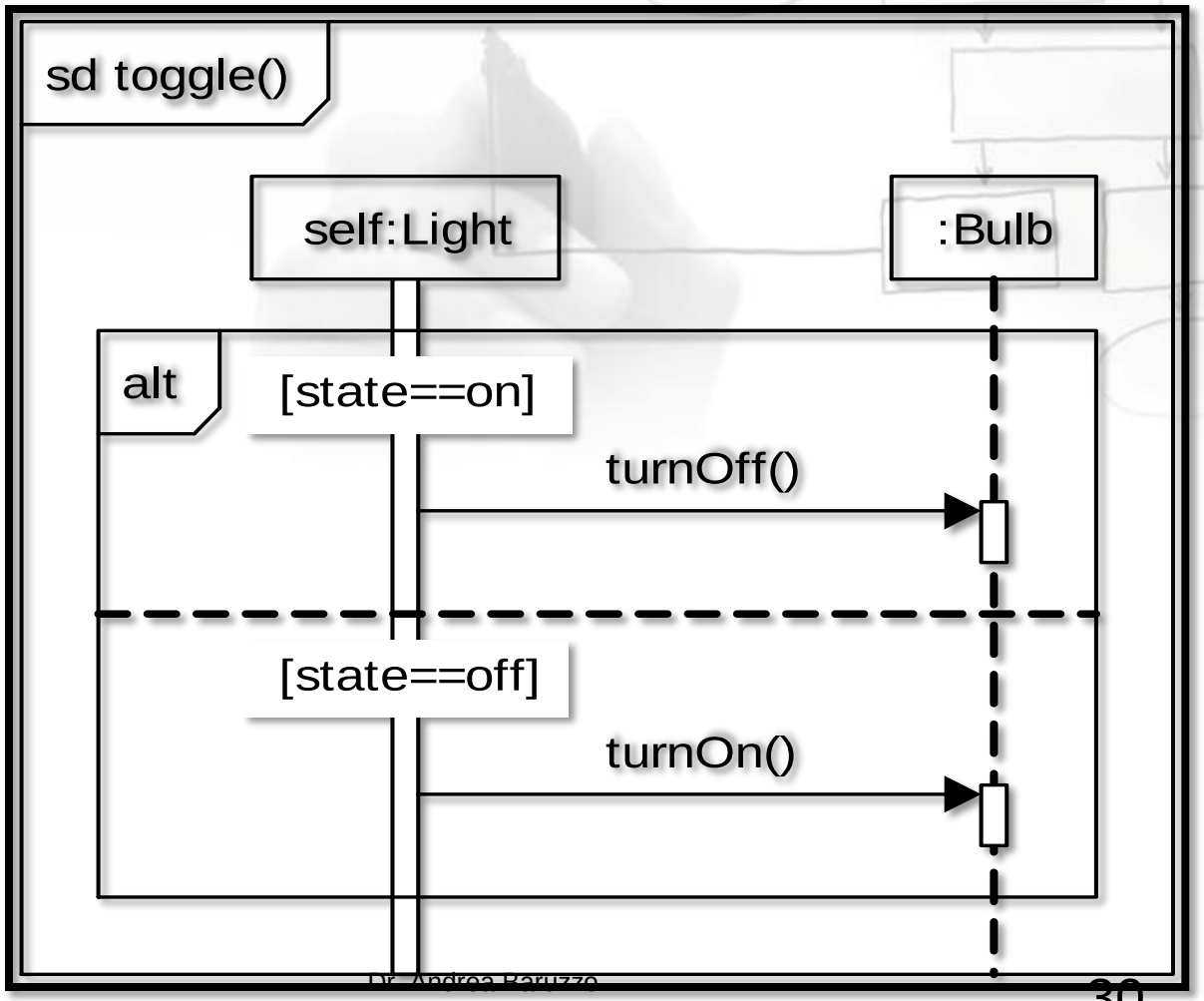
Alternative fragment



A combined fragment with one or more guarded operands whose guards are mutually exclusive

- Equivalent to a case or switch statement
- Operator is the keyword alt

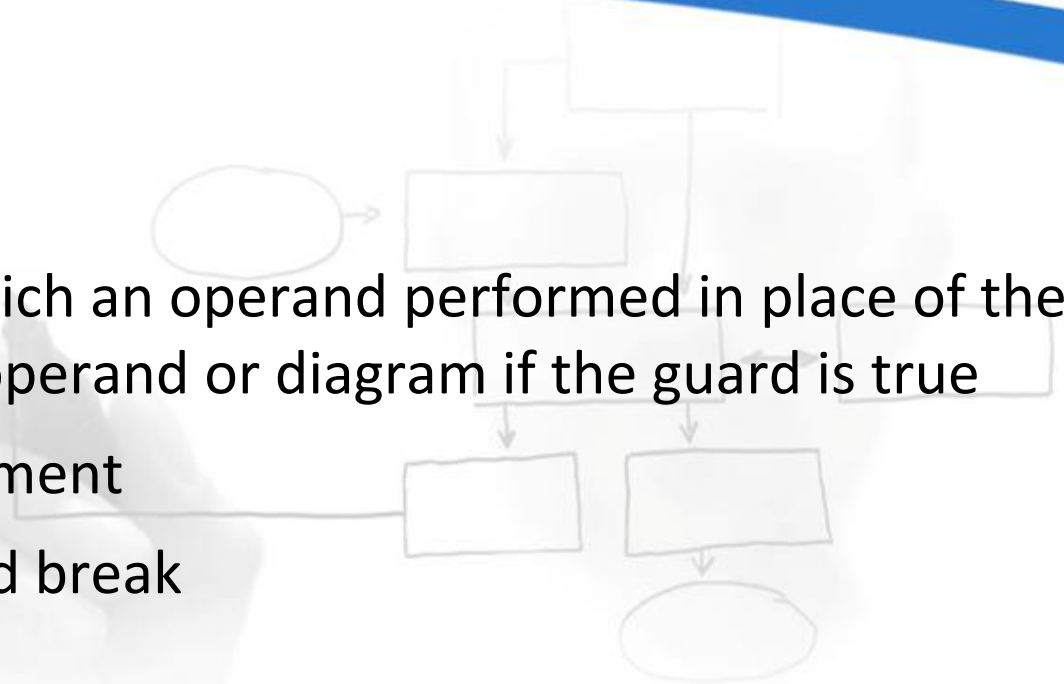
Alternative fragment example



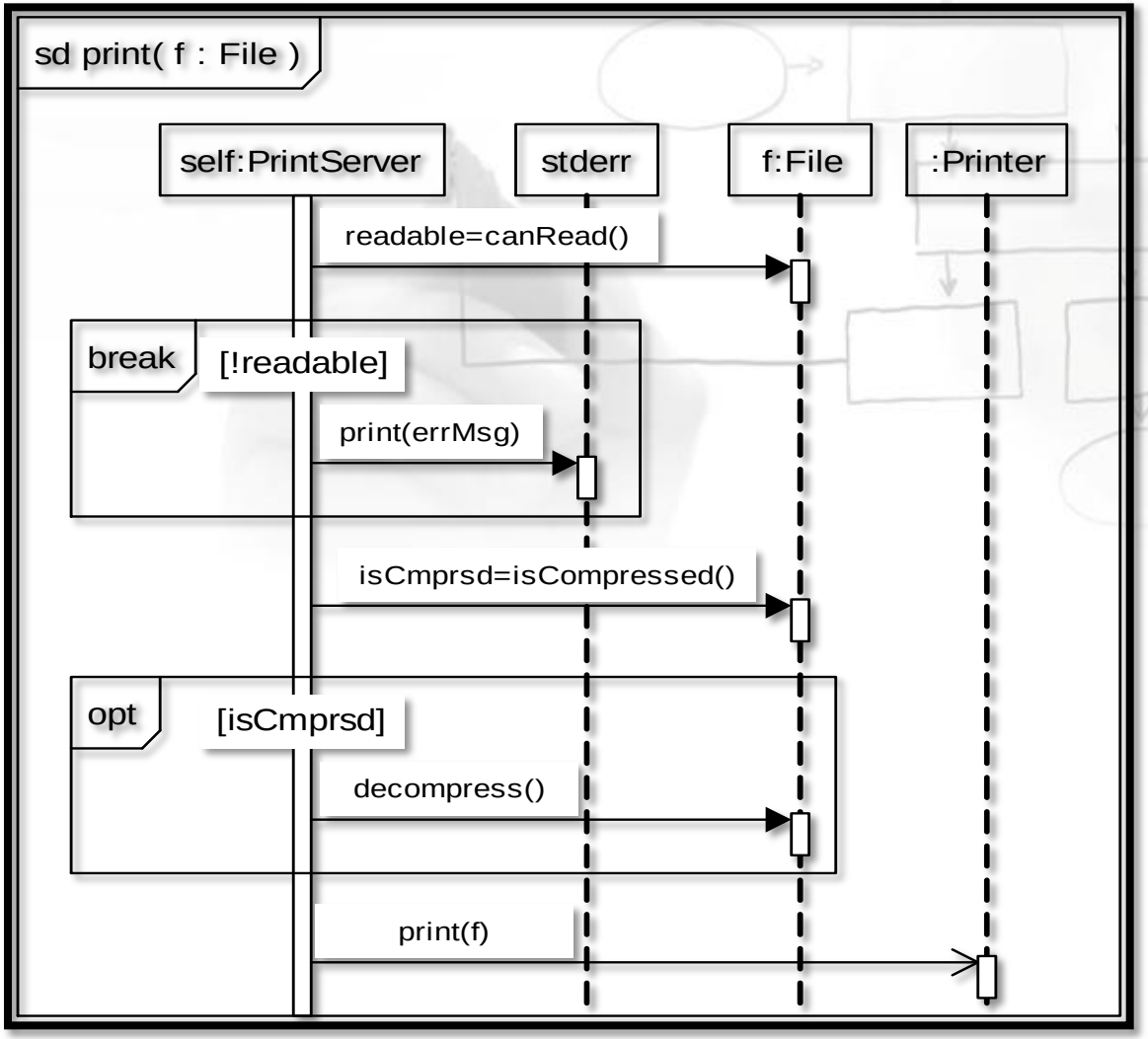
Break fragment

A combined fragment in which an operand performed in place of the remainder of an enclosing operand or diagram if the guard is true

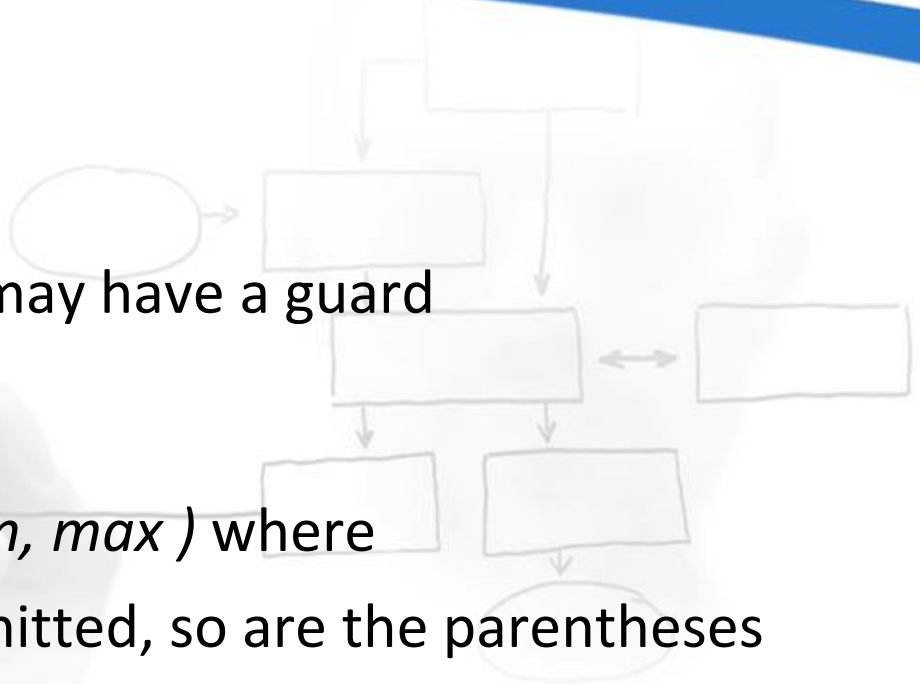
- Similar to a break statement
- Operator is the keyword break



Break fragment example



Loop fragment



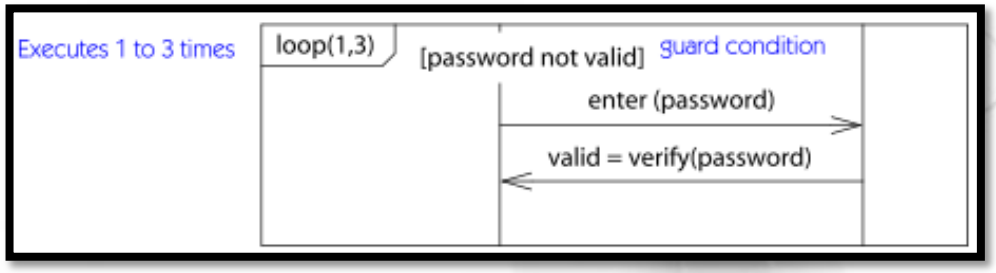
- Single loop body operand that may have a guard

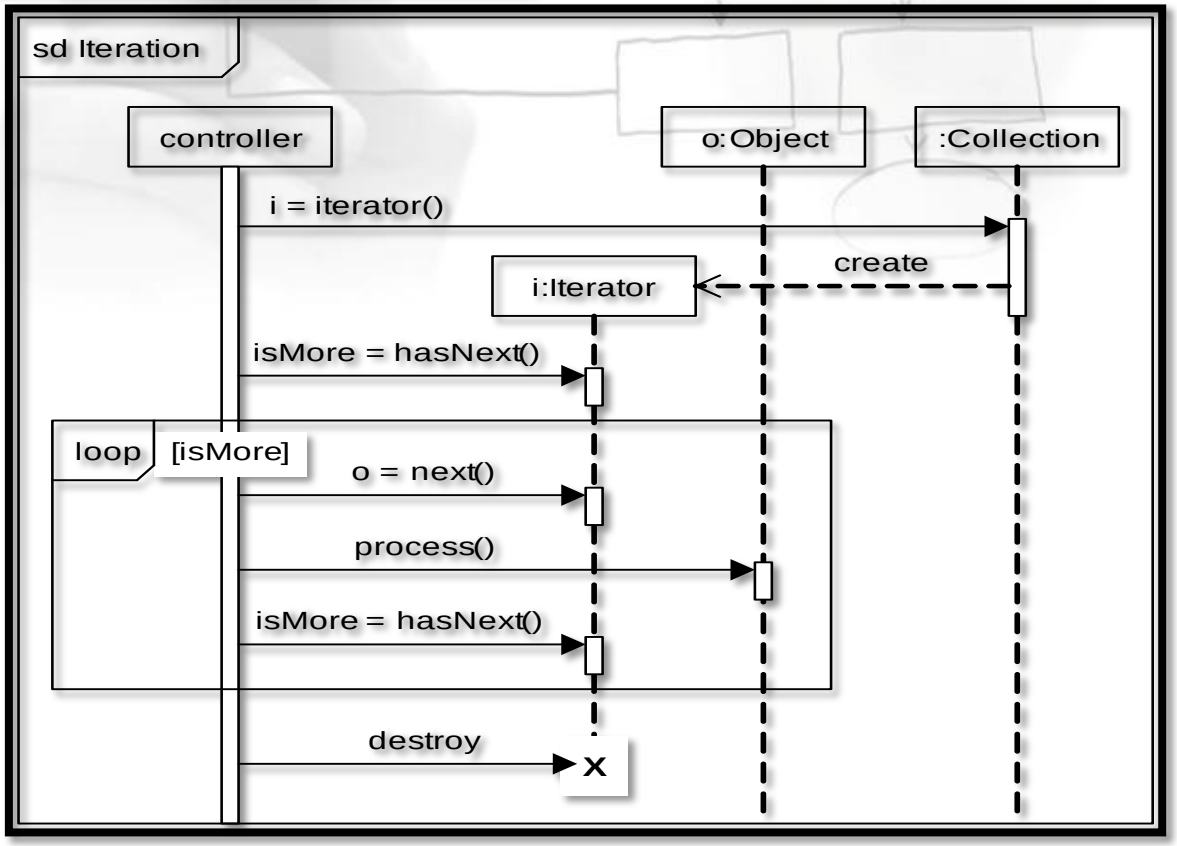
- Operator has the form *loop(min, max)* where
  - Parameters are optional or omitted, so are the parentheses
  - *min* is a non-negative integer
  - *max* is a non-negative integer at least as large as min or *; max is optional; if omitted, so is the comma

Loop fragment execution rules

- The loop body is performed at least min times and at most max times
- If the loop body has been performed at least min times but less than max times, it is performed only if the guard is true
- If *max* is *, the upper iteration bound is unlimited
- If *min* is specified but max is not, then *min=max*
- If the loop has no parameters, then *min=0* and *max* is unlimited
- The default value of the guard is true

Loop fragment example

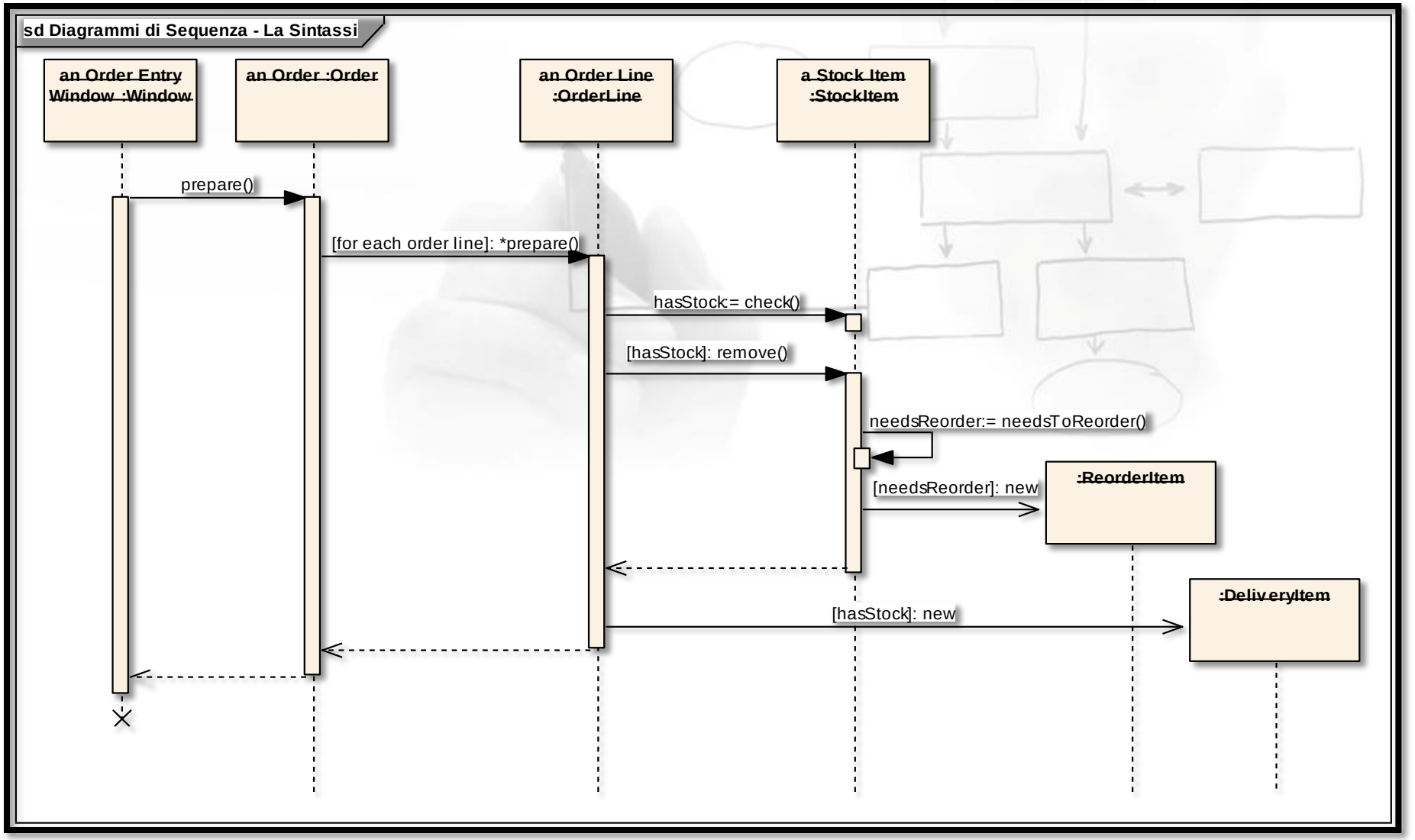




Mettiamo tutto insieme… un esempio completo

- Costruiamo un diagramma di sequenza per il seguente use case [1]
  - Una finestra di tipo Order Entry invia il messaggio “prepare” ad un Ordine (Order)
  - L’ordine invia il messaggio “prepare” ad ogni sua linea (Order Line)
  - Ogni linea verifica gli elementi in stock (Stock Item)
  - Se il controllo ha esito positivo, la linea rimuove l’appropriata quantità di elementi in stock e crea un’unità di delivery (DeliveryItem)
  - Se gli elementi in stock rimanenti scendono al di sotto di una soglia di riordino, viene richiesto un riordino (ReorderItem)

Mettiamo tutto insieme… il diagramma



Alcuni suggerimenti finali (1/2)

- Assicurarsi che i metodi rappresentati nel diagramma siano gli stessi definiti nelle corrispondenti classi (con lo stesso numero e lo stesso tipo di parametri)
- Documentare ogni assunzione nella dinamica con note o condizioni (ad es. il ritorno di un determinato valore al termine di un metodo, il verificarsi di una condizione all'uscita da un loop, ecc.)
- Mettere un titolo per ogni diagramma (ad es. "sd Diagrammi di Sequenza – Eliminazione di un Oggetto")

Alcuni suggerimenti finali (2/2)

- Scegliere nomi espressivi per le condizioni e per i valori di ritorno
- Non inserire troppi dettagli in un unico diagramma (flussi condizionati, condizioni, logica di controllo)
- Non bisogna rappresentare tutto quello che si rappresenta nel codice …
- Se il diagramma è complesso, scomporlo in più diagrammi semplici (ad es. uno per il ramo if, un altro per il ramo else, ecc.)

Bibliografia

- [Booch et al., 2005] Grady Booch et al. "*The Unified Modeling Language User Guide 2/E*", Addison-Wesley, 2005
- [Rumbaugh et al., 2004] J. Rumbaugh et al. "*The Unified Modeling Language Reference Manual 2/E*", Addison-Wesley, 2004
- [Fowler, 2003] Martin Fowler. "*UML Distilled 3/E*", Addison-Wesley, 2003
- [Larman, 2004] C. Larman. "*Applying UML and Patterns*", Addison-Wesley, 2004
- [Pender, 2003] Tom Pender. "*UML Bible*", Wiley&Sons, 2003

Domande?
Commenti?
Dubbi?

Titolo

- Testo