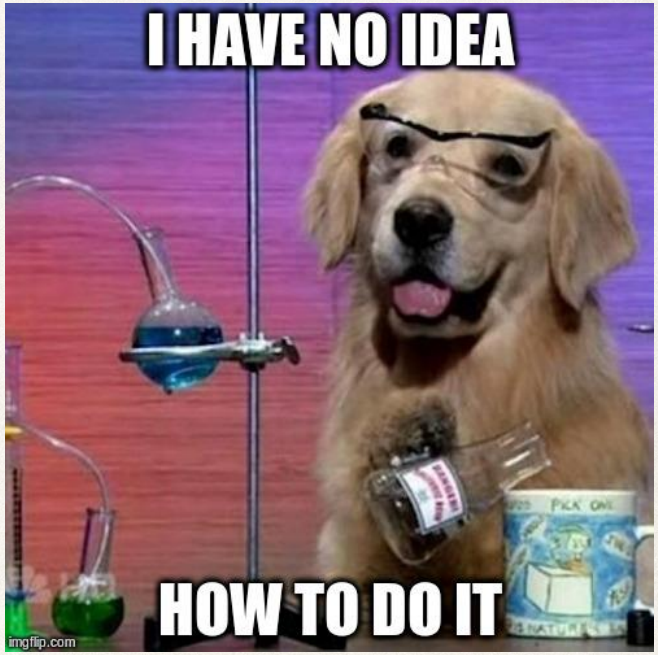
UML (parte prima)

Alberto Gianoli

Univ. Ferrara - Corsi di Laurea in Informatica

Perché modellare

Poco dopo…



Specialmente quando programmiamo a oggetti, abbiamo bisogno di chiarirci le idee, di “fare uno schemino”…

Perché modellare

Modello == semplificazione della realtà

I modelli:

- visualizzano come è fatto un sistema, o come vorremmo che fosse
- specificano la struttura o il comportamento di un sistema
- sono una guida nello sviluppo di un sistema
- documentano le decisioni prese

Oggi per modellazione dei sistemi di solito intendiamo la rappresentazione del sistema usando qualche tipo di grafica, oramai sempre basata sulla notazione UML

Perché modellare

Possiamo modellare sia un sistema esistente che un nuovo sistema

- I **modelli di un sistema esistente** si usano per chiarire cosa fa il sistema (documentazione), si possono usare per discutere punti di forza e di debolezza, per pianificare modifiche.
- I **modelli di un nuovo sistema** si usano per descrivere il sistema proposto agli altri stakeholder, per discutere come implementarlo, come documentazione.

Con vari modelli possiamo rappresentare il sistema da prospettive diverse: prospettiva esterna (contesto o ambiente in cui opera il sistema); di interazioni tra i componenti; strutturale (del sistema o dei dati); comportamentale

Perché modellare

- scomporre il problema (divide et impera): focalizzarsi su un aspetto alla volta
- ogni modello può essere sviluppato a differenti livelli di precisione
- se il progetto non è banale:
  - non basta un solo modello
  - un insieme di modelli: correlati tra loro, ma possono essere costruiti e studiati separatamente

Perché modellare

- Se modello per facilitare la discussione su un sistema proposto o esistente
  - i modelli possono essere incompleti
- Se sto documentando un sistema proposto o esistente
  - i modelli devono essere corretti e descrivere accuratamente il sistema, ma non è necessario che siano completi
- Se voglio fornire una descrizione dettagliata del sistema da utilizzarsi per sviluppare l'implementazione
  - i modelli devono essere sia corretti che completi!

Cos'è UML

- linguaggio (e relativa notazione) universale per creare modelli software
- dal '97 è uno standard
- membri sviluppatori: Microsoft, IBM, HP, Oracle, Unysis, Sun, ….
- ha riunito alcuni metodi pre-esistenti, prendendo idee da altre metodologie
- è indipendente da metodi, tecnologie, produttori

UML

- è un linguaggio: serve a specificare, costruire, visualizzare e documentare i componenti di un sistema
- è universale e perciò può applicarsi a sistemi molto diversi: dal Cobol a OO al web

UML: non è un metodo

- è un linguaggio di modellazione: non è un metodo né una metodologia
- definisce una notazione standard: introduce un meta-modello integrato dei componenti che formano un sistema software
- non prescrive cosa bisogna fare: non dice “prima bisogna fare questa attività, poi quest’altra”

UML e processo software

"un unico modello universale utilizzabile per tutti gli stili di sviluppo non sembra possibile né tanto meno desiderabile"

- in realtà UML assume un processo
  - basato su "casi d'uso" (use case driven)
  - incentrato sull'architettura
  - iterativo e incrementale
- i dettagli di questo procedura di tipo generale vanno adattati al dominio applicativo di ciascuna organizzazione

Struttura di UML

- **Costituenti fondamentali**: gli elementi, le relazioni e i diagrammi di base
- **Meccanismi comuni**: tecniche comuni per raggiungere specifici obiettivi con UML
- **Architettura**: il modo con cui UML esprime l'architettura del sistema

UML: costituenti fondamentali

- **Entità strutturali**: strutturali (classe, interfaccia), comportamentali (interazione, stati), di raggruppamento (package) e informative (annotazione)
- **Relazioni**: collegano tra di loro le entità, rappresentano associazione, dipendenza, generalizzazione, realizzazione
- **Diagrammi**: rappresentazioni grafiche parziali di un modello UML; sono viste che consentono di vedere il contenuto del modello da prospettive diverse. Una entità o relazione può essere eliminata da un diagramma (magari per facilitare la comprensione) ma continuare a esistere nel modello.

UML: meccanismi comuni

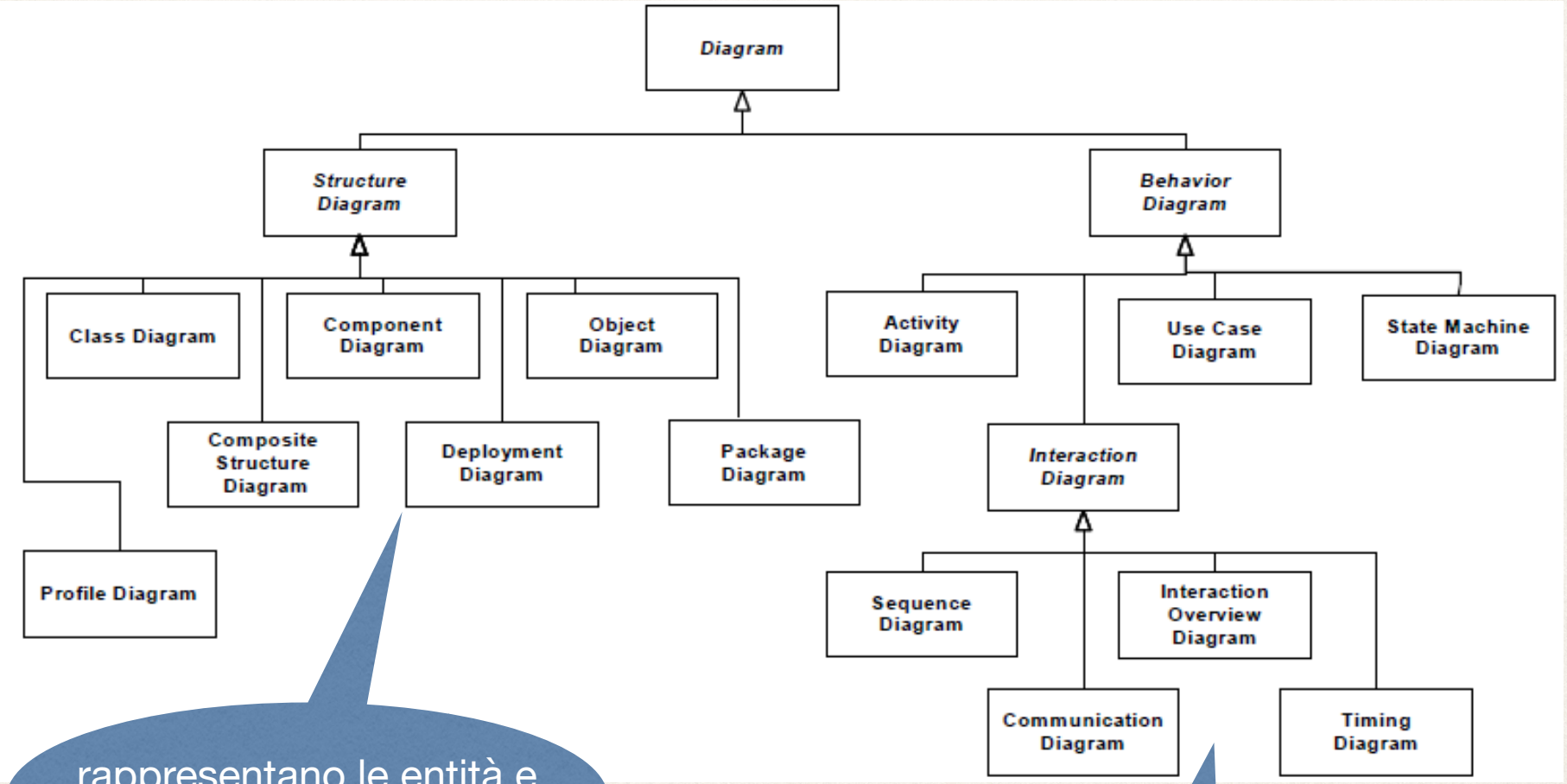
- **Specifiche**: descrizione testuale della semantica di un elemento (classe, relazione, …). Completa la descrizione grafica rappresentata dai diagrammi
- **Ornamenti**: informazioni aggiunte in un diagramma ad un elemento per illustrare un concetto supplementare (molteplicità, visibilità, …)
- **Distinzioni comuni**: per distinguere tra astrazioni e loro istanze
- **Meccanismi di estendibilità**: permettono di estendere il linguaggio per esigenze specifiche. Sono di tre tipi: vincoli, stereotipi, e valori etichettati.

UML: architettura

Solitamente viene definita da più viste. “Vista” non è definita in modo formale, l’approccio più famoso è il “modello 4+1”

- **Vista logica**: mostra le astrazioni chiave nel sistema come oggetti o classi di oggetti (diagrammi di classe, di stato, di oggetti)
- **Vista dei processi**: descrive il comportamento dinamico (diagrammi di stato, di oggetti)
- **Vista di implementazione**: descrive la struttura concreta del sw che compone il sistema (diagrammi dei componenti)
- **Vista di deployment**: mostra come il sw viene suddiviso per lo sviluppo (diagrammi di deployment)
- **Vista dei casi d’uso (+1)**: descrive i requisiti in termini di servizi offerti (diagramma casi d’uso, interazione)

Diagrammi UML



rappresentano le entità e le loro relazioni

rappresentano il comportamento al trascorrere del tempo

Diagramma casi d'uso

Mostra:

- modalità utilizzo del sistema (casi d'uso)
- gli utilizzatori e chi interagisce col sistema (attori)
- relazioni tra attori e casi d'uso

Un caso d'uso

- rappresenta un possibile modo di utilizzare il sistema
- descrive l'interazione, non la logica interna del sistema

rappresentazione del sistema dal punto di vista dell'utilizzatore

Perché fare dei casi d'uso

E' la prima fase nel ciclo di vita del software in cui sviluppiamo una rappresentazione del software

1. **Esplicitare e comunicare a tutti gli stakeholder i requisiti funzionali del sistema** - analizzare, identificare e descrivere l'uso tipico di un sistema da parte del suo utilizzatore; si considera anche la parte di gestione degli errori, eccezioni, thread alternativi

2. **Validazione requisiti utente** - modello "semplice" (semantica precisa, pochi costrutti, intuitivo), ma valido per controllare di aver sviluppato un sistema corrispondente alle vere necessità dell'utente

Use Case

- Uno “use case” rappresenta chi (attore) fa cosa (interazione) con il sistema, e con quale scopo (goal), senza considerare l’interno del sistema
- Un use case:
  - rappresenta un singolo, discreto, completo, significativo e ben definito task di interesse per un attore
  - è un pattern di comportamento tra alcuni attori e il sistema - una collezione di possibili scenari
  - è scritto usando il vocabolario del dominio
  - definisce scopo e intento (non le azioni concrete)
  - è generale e indipendente dalla tecnologia

Use Case - Requisiti

- un caso d'uso può soddisfare più requisiti
- un requisito può generare più casi d'uso
- ad un caso d'uso possono essere associati più requisiti
- a cosa serve?
  - chiarire i requisiti del committente in modo comprensibile
  - trovare aspetti comuni (riuso)
  - individuare gli attori del sistema
  - individuare gli eventi a cui il sistema deve rispondere

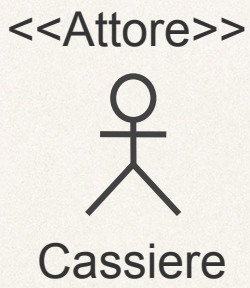
Use Case - descrizioni

Quattro concetti fondamentali

1. **Caso d'uso** - rappresenta una specifica interazione tra un attore e il sistema; in UML viene rappresentato da una ellissi etichettata con il nome del caso d'uso
2. **Attore** - rappresenta un ruolo che caratterizza le interazioni tra utente e sistema; non è necessariamente una persona; in UML viene rappresentato da un omino stilizzato
3. **Scenario** - sequenza di azioni che definisce una particolare interazione tra attore e sistema
4. **Descrizione** - testo che descrive lo scenario, l'ordine temporale delle azioni, l'eventuale gestione degli errori, gli attori coinvolti

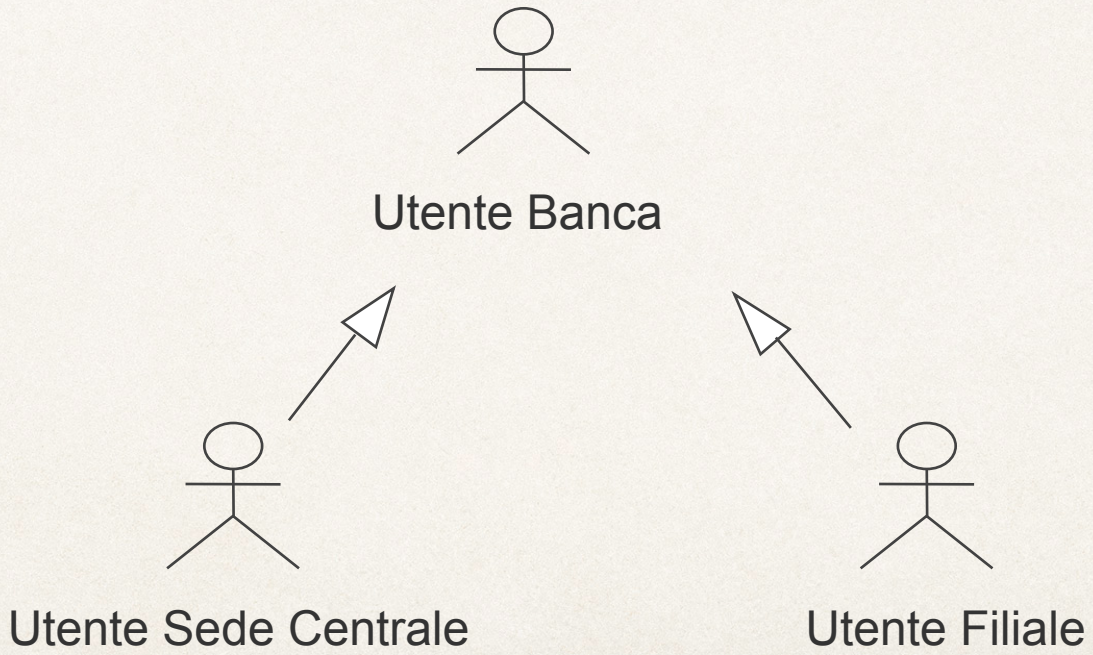
Attore

- Un attore è rappresentato mediante il ruolo che ricopre nel caso d'uso; normalmente indicato in questo modo

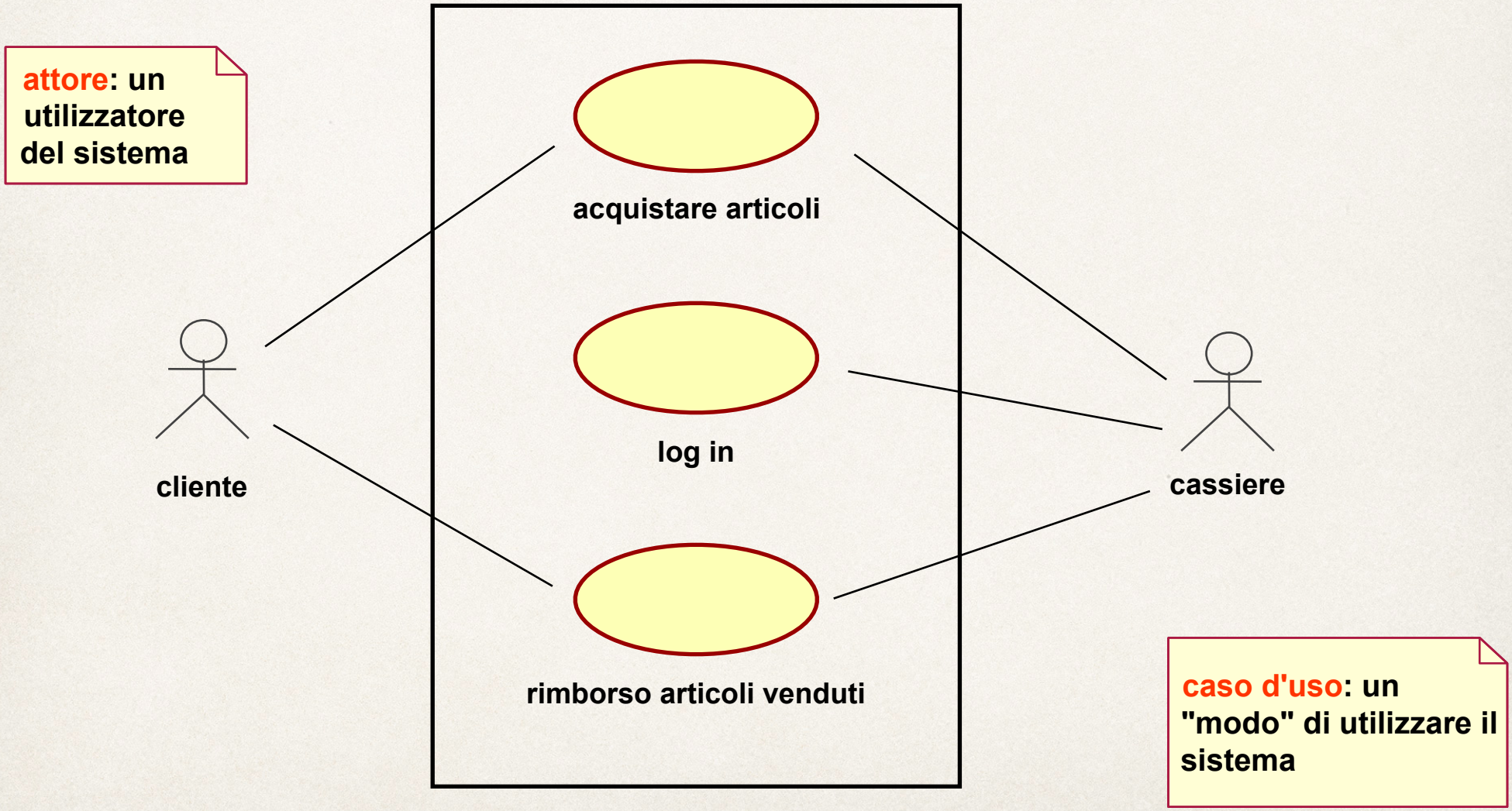


Relazioni tra attori

- E' possibile definire delle gerarchie tra attori
- Associare uno o più attori a un attore specializzato (specializzazione)



Use Case diagram



Esempio di documentazione di un Use Case

- **Nome caso d'uso** - ogni use case deve avere un nome; il nome esprime il goal dell'utente
- **Goal** - descrizione sommaria della funzionalità fornita; deve soddisfare una necessità dell'utente cioè è percepita dall'utente come "valore"
- **Attori** - persona, dispositivo o altro, esterno al sistema che interagisce col sistema. In ogni use case ci deve essere un attore primario, chi inizia il caso d'uso stesso
- **Pre-condizioni** - condizioni che devono essere soddisfatte all'inizio del caso d'uso. "Garanzie minime" (garanzie fornite dagli attori al sistema) che devono essere soddisfatte per poter arrivare lo scenario in questione
- **Trigger** - evento che attiva il caso d'uso
- **Descrizione (scenario principale)** - descrizione della sequenza di interazioni più comune tra gli attori e il sistema. In particolare la sequenza che porta alla conclusione con successo: input forniti dall'attore e risposta del sistema. Il sistema viene trattato come una black-box, importa la risposta all'input, non come viene generata la risposta
- **Alternative (estensioni)** - variazione dalla sequenza dello scenario principale (es. gestione eccezioni). Una alternativa non è necessariamente un fallimento dello use case
- **Post-condizioni** - condizioni sempre soddisfatte alla fine del caso d'uso (garanzie fornite dal sistema agli attori)

Esempio: prelievo dal bancomat

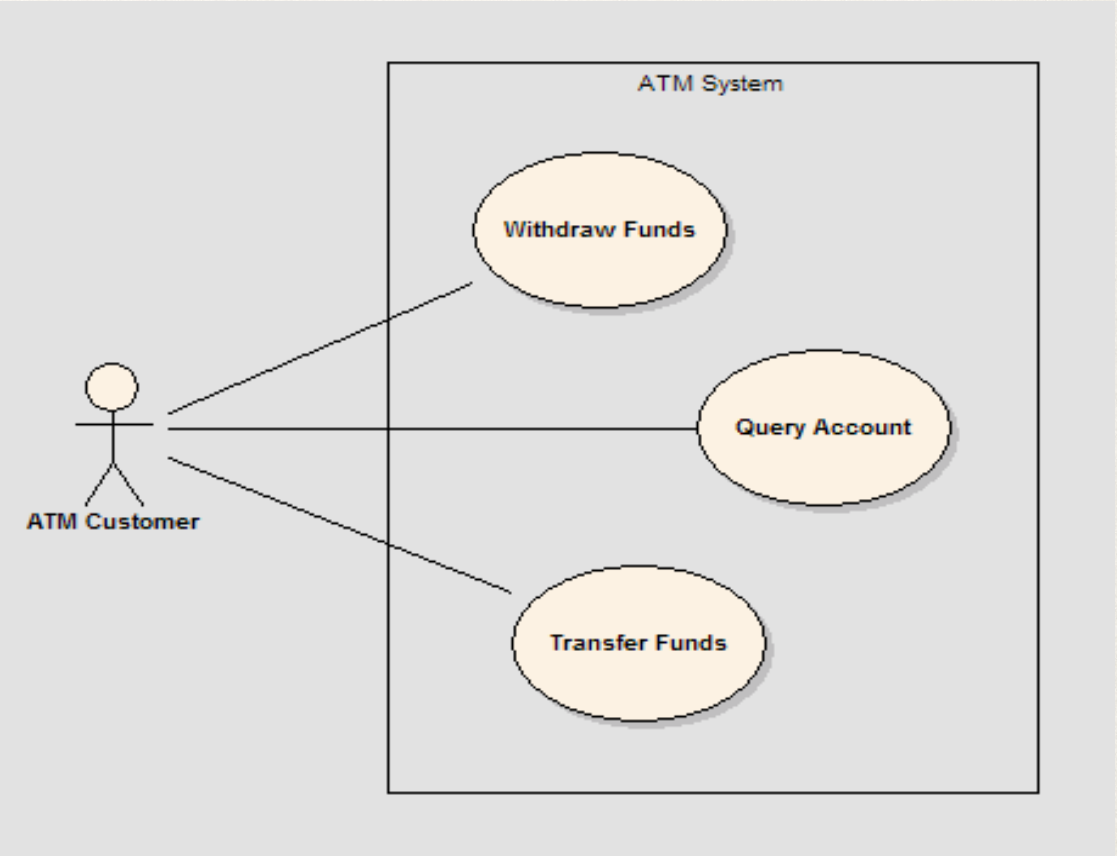
Nome caso d'uso: prelievo denaro

Scope: sistema bancomat

Goal (summary): l'utente preleva una specifica somma di denaro da un conto valido

Dipendenze con altri use case: nessuna

Attori: cliente del terminale



Esempio (cont.)

Trigger: inserimento di una carta bancomat nel lettore del terminale
Precondizioni: il terminale è in attesa (idle) e visualizza il messaggio di benvenuto
Descrizione (main success scenario o scenario principale):

1. il cliente inserisce nel lettore la sua carta
2. il sistema riconosce la carta e ne legge il numero
3. il sistema visualizza a terminale la richiesta di inserimento del PIN
4. il cliente inserisce il suo PIN
5. il sistema verifica che la carta non e' scaduta né rubata o smarrita
6. il sistema verifica che il PIN corrisponda alla carta
7. il sistema controlla che il conto sia accessibile mediante l'utilizzo della carta
8. il sistema visualizza il conto del cliente e le possibili operazioni che si possono effettuare
9. il cliente seleziona il conto (nel caso ne abbia più di uno) e seleziona l'operazione Prelievo
10. il sistema visualizza la richiesta dell'ammontare da prelevare
11. il cliente inserisce l'ammontare
12. il sistema verifica che il conto contenga sufficienti fondi per consentire l'operazione e che non sia stato superato il limite giornaliero
13. il sistema autorizza il prelievo
14. il sistema emette il denaro, registrando la transazione
15. il sistema stampa la ricevuta con il riassunto dell'operazione
16. il sistema espelle la carta
17. il sistema ritorna nello stato di attesa (idle)

Esempio (cont.)

Alternative (estensioni):

2a. se il sistema non riconosce la carta dell'utente, la espelle

5a. se la carta è scaduta, il sistema la confisca

5b. se la carta risulta smarrita, il sistema la confisca

5c. se la carta risulta rubata, il sistema la confisca

6a. se il cliente ha inserito un PIN che non corrisponde, visualizza una nuova richiesta di numero; se è il terzo sbaglio, confisca la carta

7a. se il numero di conto non e' valido, visualizza messaggio e espelli la carta

12a. se non ci sono fondi sufficienti, visualizza messaggio, espelli la carta e riporta il terminale nello stato idle.

12b. se è stato superato il limite giornaliero, visualizza un messaggio di errore, espelli la carta, e riporta il terminale nello stato idle

14a. se il terminale non ha fondi a sufficienza, visualizza un messaggio, espelli la carta e riporta il terminale nello stato idle

14b. se il cliente seleziona da terminale il pulsante Cancella, annulla la transazione, espelli la carta e riporta il terminale nello stato idle.

Post-condizioni: l'utente ha ottenuto la somma di denaro che aveva richiesto

Questioni aperte: nessuna

Relazioni tra Use Case

- Relazione di inclusione <<include>>
- Relazione di generalizzazione
- Relazione di estensione <<extend>>

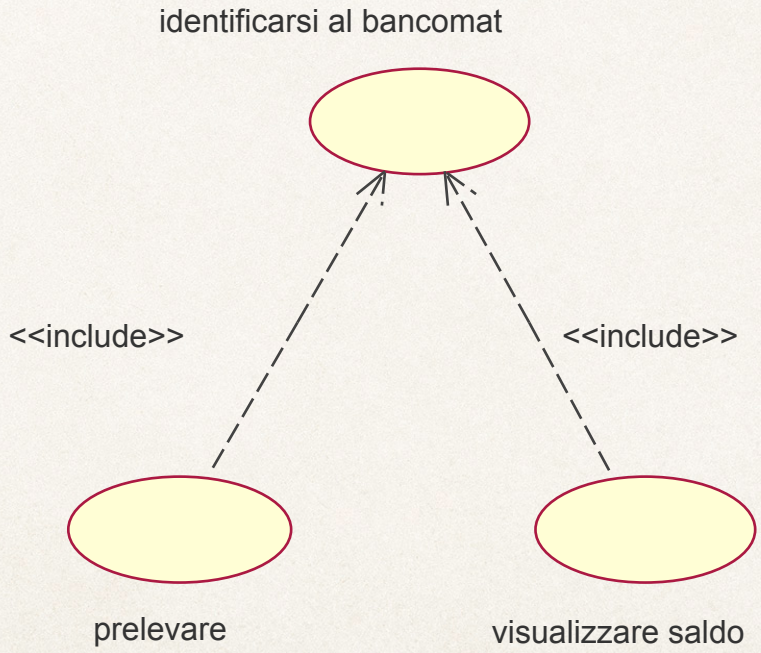
- poche relazioni: modello semplice

Relazioni tra Use Case: inclusione

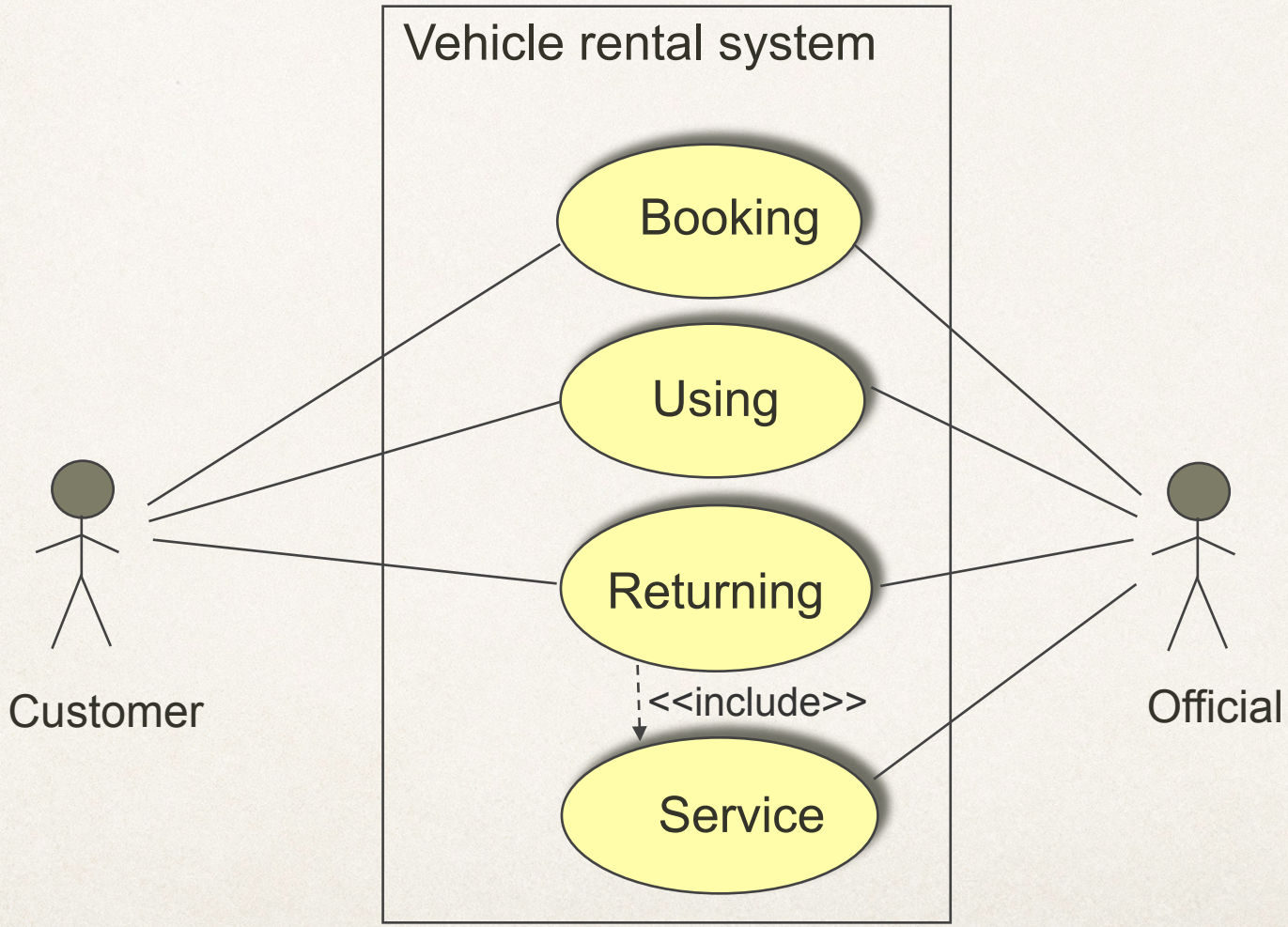
- rappresenta l'invocazione di un caso d'uso da parte di un altro
- simile alla chiamata a una funzione
- serve per scomporre un caso complesso in componenti più semplici
- facilita il riuso dei singoli casi d'uso

Inclusione (cont.)

- può anche mostrare un comportamento comune a più casi d'uso



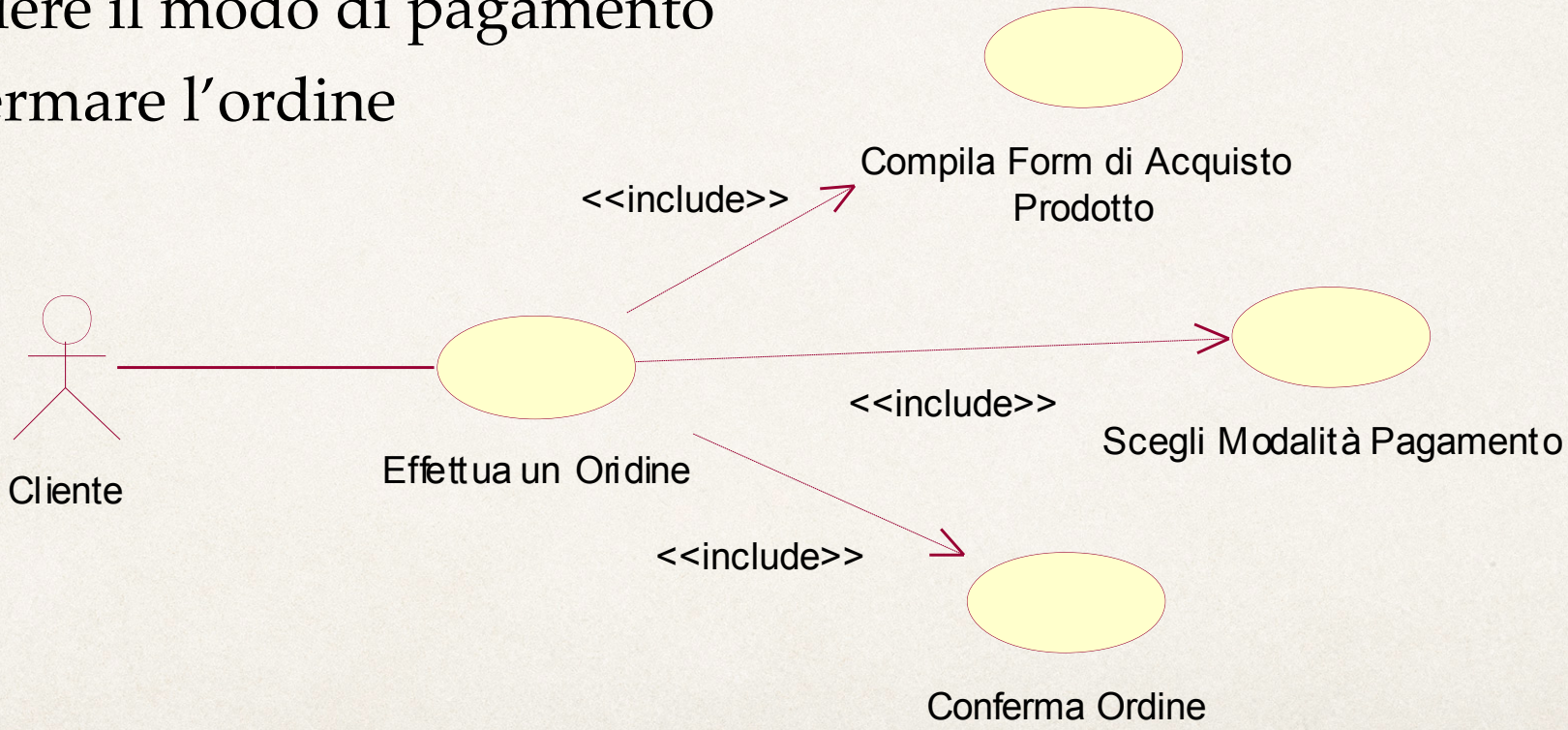
Esempio: noleggio auto



Esempio: ordini

Fare un ordine contiene sempre le seguenti azioni:

1. compilare un form
2. scegliere il modo di pagamento
3. confermare l'ordine



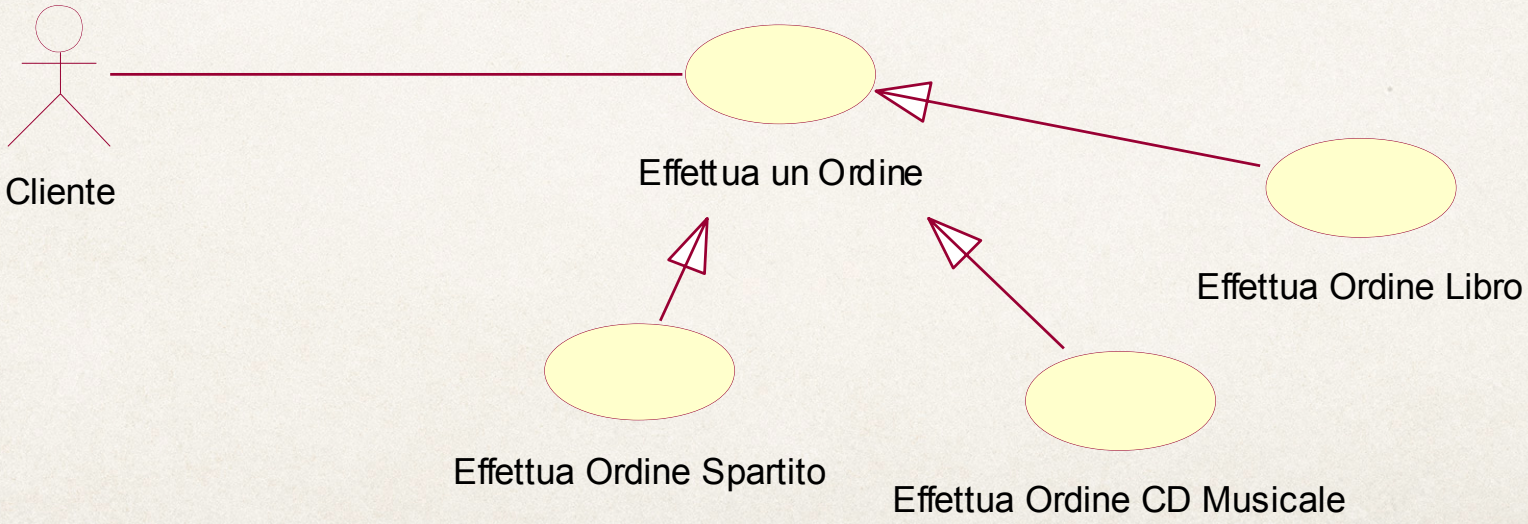
Relazioni tra Use Case: generalizzazione

- Simile all'ereditarietà tra classi
- Relazione che lega un caso generico a casi d'uso che sono particolari specializzazioni (realizzazioni) del primo
- Logica del caso derivato simile a (ma diversa da) quella del caso d'uso generico
- Viene riscritta la sequenza delle azioni di base oppure viene elaborata una sequenza alternativa

Generalizzazione (cont.)

- "Effettua un ordine" è un caso generale che può essere specializzato nei casi seguenti:

1. effettua un ordine di un libro
2. effettua un ordine di un cd
3. effettua un ordine di uno spartito

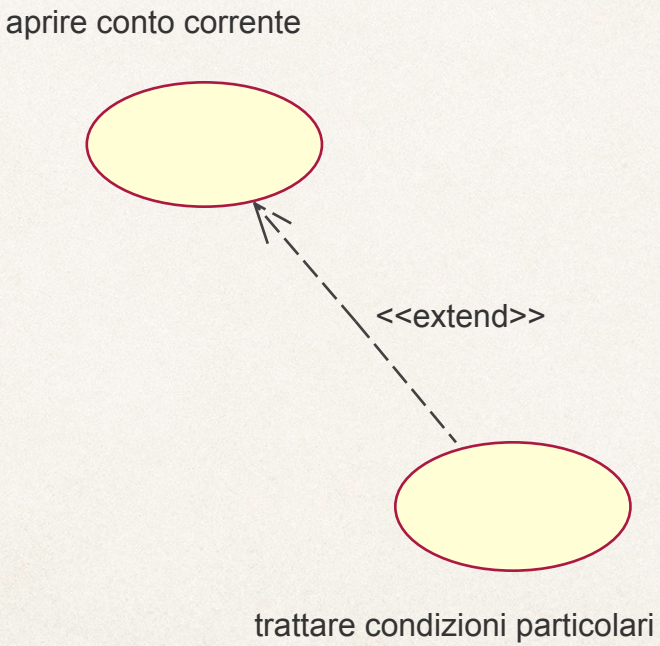


Relazioni tra Use Case: estensione

- Rappresenta l'estensione della logica di base di un caso d'uso
- Il caso d'uso estensione continua il comportamento del caso d'uso di base inserendo delle azioni alternative da un certo punto in poi (punto di estensione)
- Il caso d'uso di base dichiara tutti i possibili punti di estensione
- Simile alla gestione degli interrupt (gestione delle eccezioni)

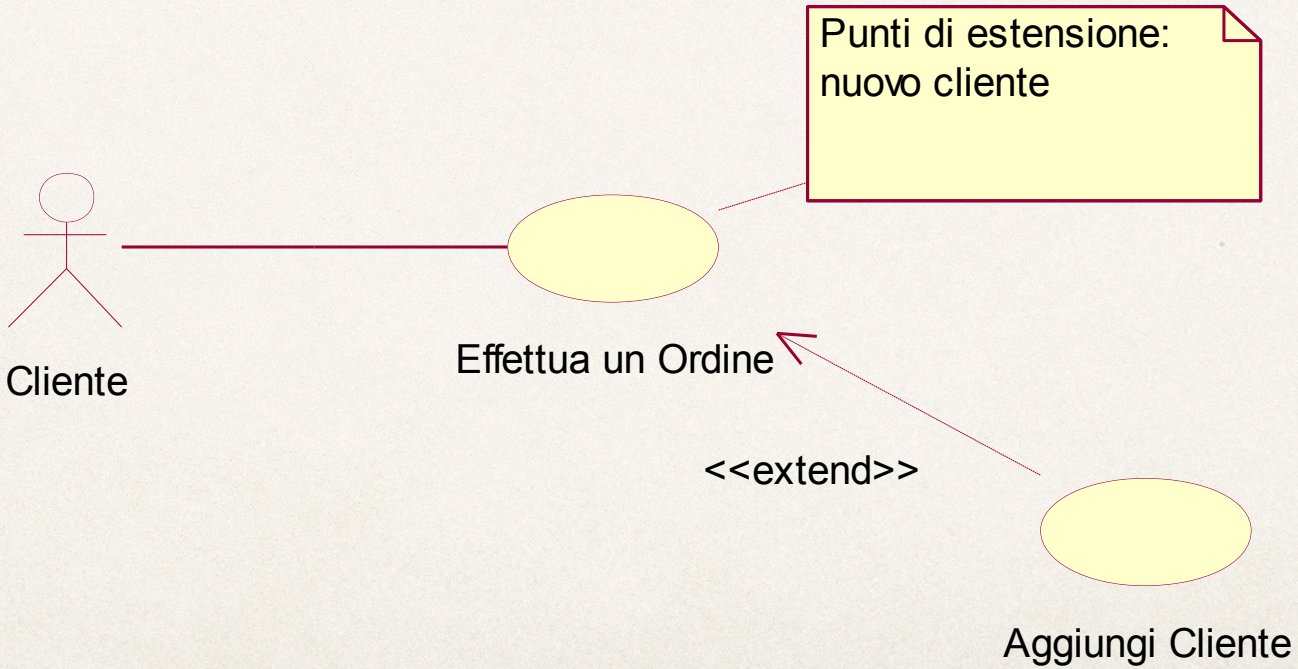
Estensione (cont.)

- <<extend>> mostra il comportamento opzionale, alternativo o relativo al trattamento di una situazione anomala



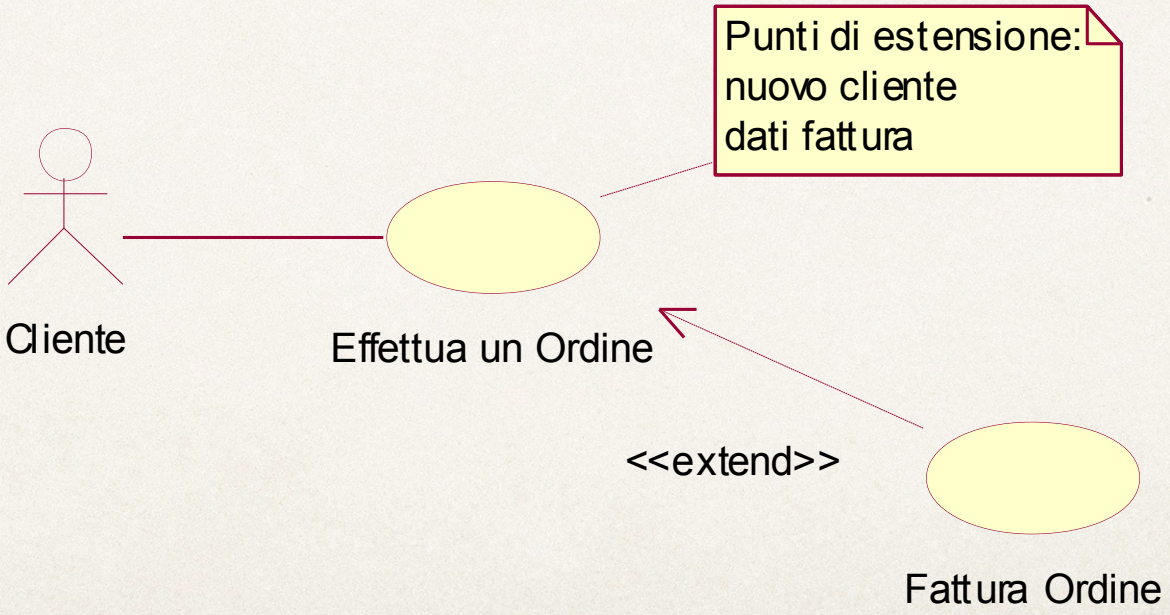
Estensione (cont.)

- Effettuare un ordine presume che l'utente sia già noto
- Se è un nuovo cliente e non è registrato? Eccezione nel caso d'uso di base!!!



Estensione (cont.)

- Effettua un ordine di default non invia la fattura.
- Se il cliente richiede la fattura? Estensione nel caso d'uso di base!!!



Esercizio: bancomat

- Sistema da eseguirsi su uno sportello automatico
- L'utente deve poter depositare assegni sul suo conto
- L'utente deve poter prelevare soldi dal suo conto
- L'utente deve poter ottenere il saldo del suo conto
- Se lo richiede, l'utente deve poter ottenere una ricevuta per la transazione
- Le transazioni sono ritiro o deposito
- La ricevuta deve indicare data, numero conto, saldo precedente e finale
- Dopo ogni transazione il sistema visualizza il nuovo saldo

Esercizio: bancomat

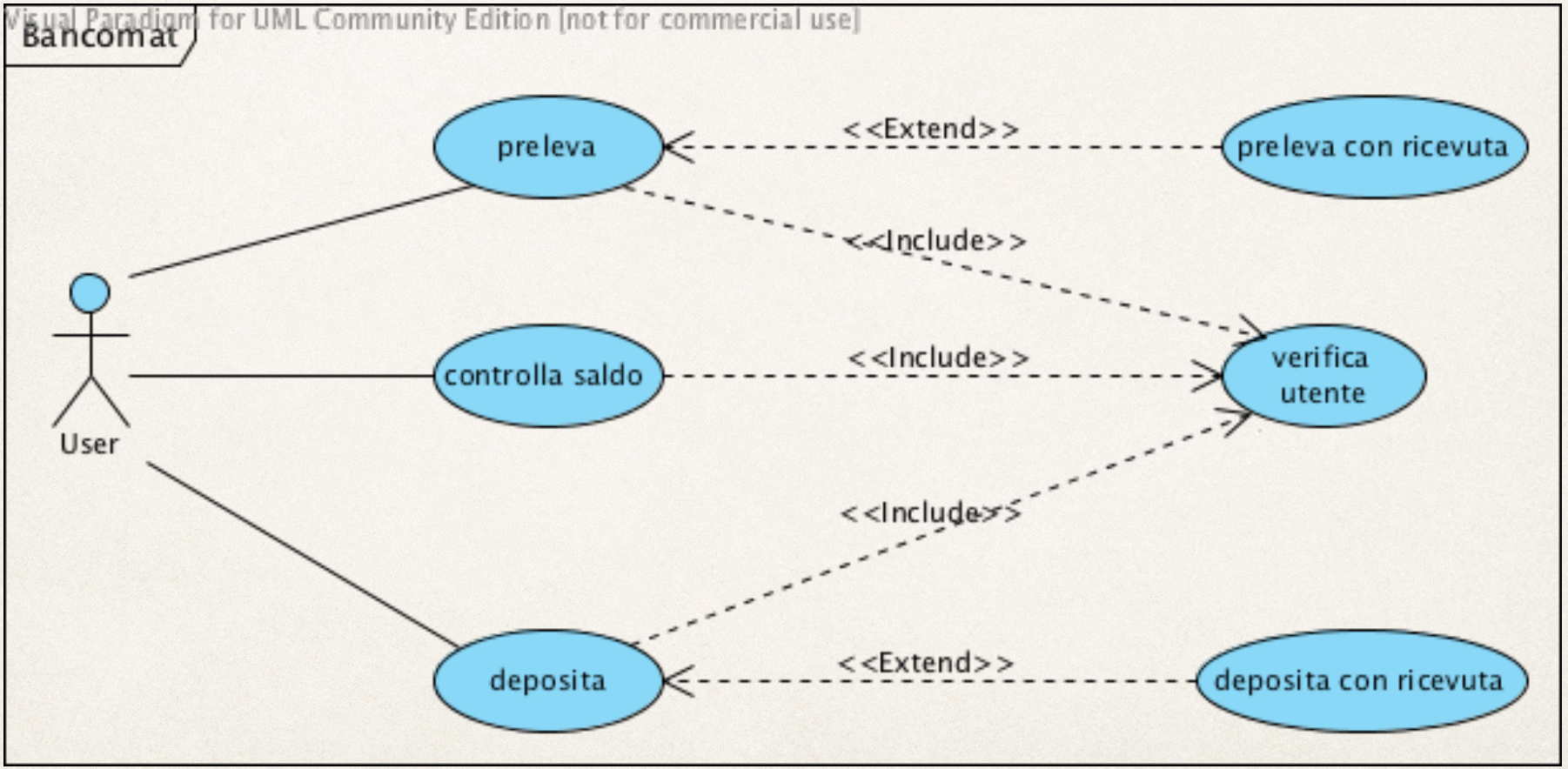
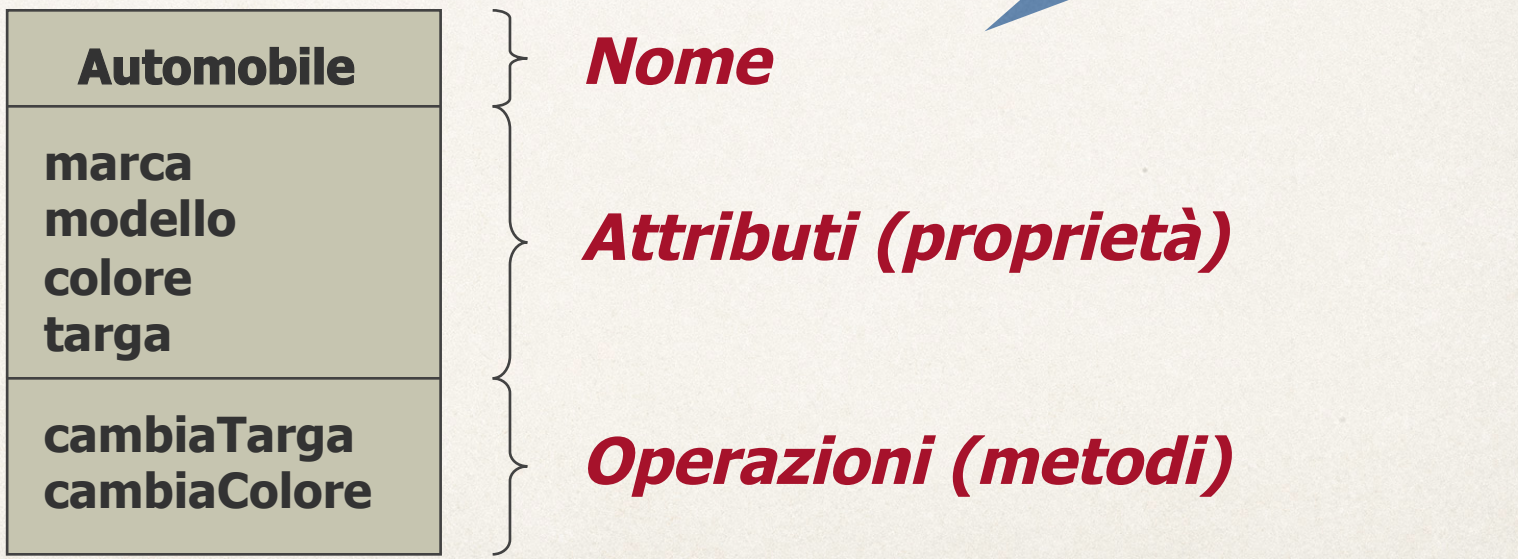


Diagramma delle classi

- è il caposaldo dell'object oriented
- rappresenta le classi di oggetti del sistema, con i loro attributi e operazioni
- mostra le relazioni tra le classi (associazioni, aggregazioni e gerarchie di specializzazione/generalizzazione)
- può essere utilizzato a diversi livelli di dettaglio (in analisi e in progettazione)

Class diagram: le classi

- Una classe è una tipologia di oggetti, con propri attributi e operazioni
- Rappresentazione di una classe in UML



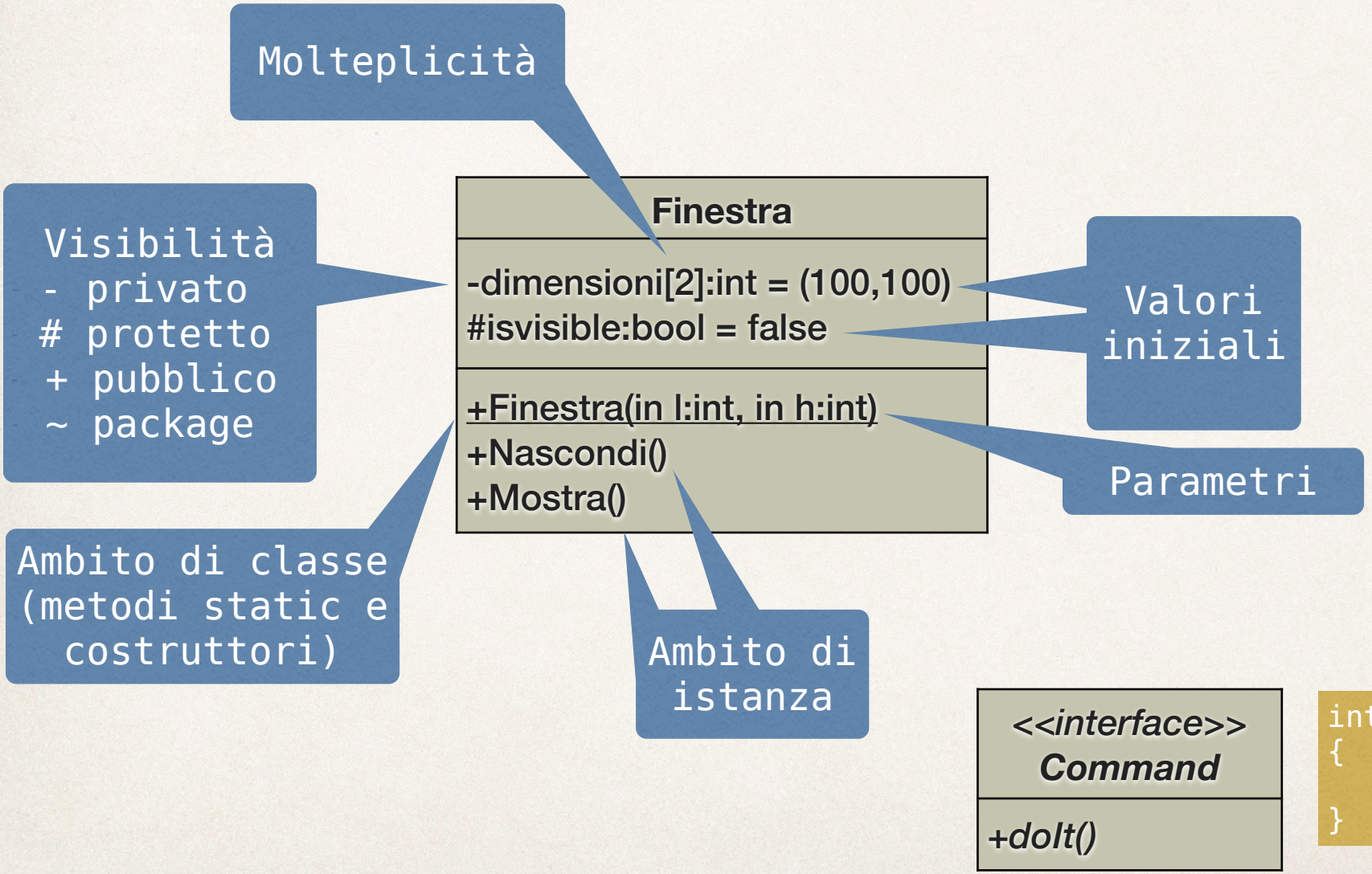
Class diagram: le classi

- **Nome**: inizia con la lettera maiuscola, non è sottolineato e non contiene underscore
- **Attributi**: proprietà i cui valori identificano un oggetto istanza della classe e ne costituiscono lo stato; iniziano con una lettera minuscola
- **Operazioni**: insieme di funzionalità che esprimono il comportamento di un oggetto, cioè ciò che ogni oggetto di questa classe può fare

Class diagram: le classi

- **Visibilità**: riguarda attributi e operazioni, può essere private (-), protected (#), public (+), package (~)
- **Molteplicità**: si indica con la notazione degli array [n]
- **Sottolineatura**: indica che l'attributo o l'operazione è static
- **Parametri delle operazioni**: possono essere preceduti da modificatore che indica "direzione"; "in" parametro in ingresso (passaggio per valore), "out" parametro in uscita (passaggio per riferimento), "inout" parametro di ingresso e uscita (passaggio per riferimento)
- **Corsivo**: indica che la classe o l'operazione è astratta
- **Interfacce**: stereotipo <<interface>>

Class diagram: le classi



Class diagram: relazioni

- In UML una relazione è: *una connessione semanticamente significativa tra elementi di modellazione*
- Nei diagrammi delle classi abbiamo tre tipi di relazioni
  - **Dipendenze** (relazioni d'uso)
  - **Associazioni** (associazione semplice, aggregazione, composizione)
  - **Generalizzazioni e realizzazioni** (ereditarietà tra classi e implementazione di interfacce)

Class diagram: dipendenze

- "una dipendenza è una relazione tra due elementi dove un cambiamento in uno di essi (fornitore) può influenzare o fornire informazioni necessarie all'altro (cliente)"

- Esempio più comune: il rapporto client-server tra due classi o fra una classe e una interfaccia
  - un metodo di Class1 ha un parametro di tipo Class2, oppure restituisce un valore di tipo Class2, oppure ha un oggetto di tipo Class2 come attributo
  - Class1 invoca uno o più metodi definiti dall'interfaccia

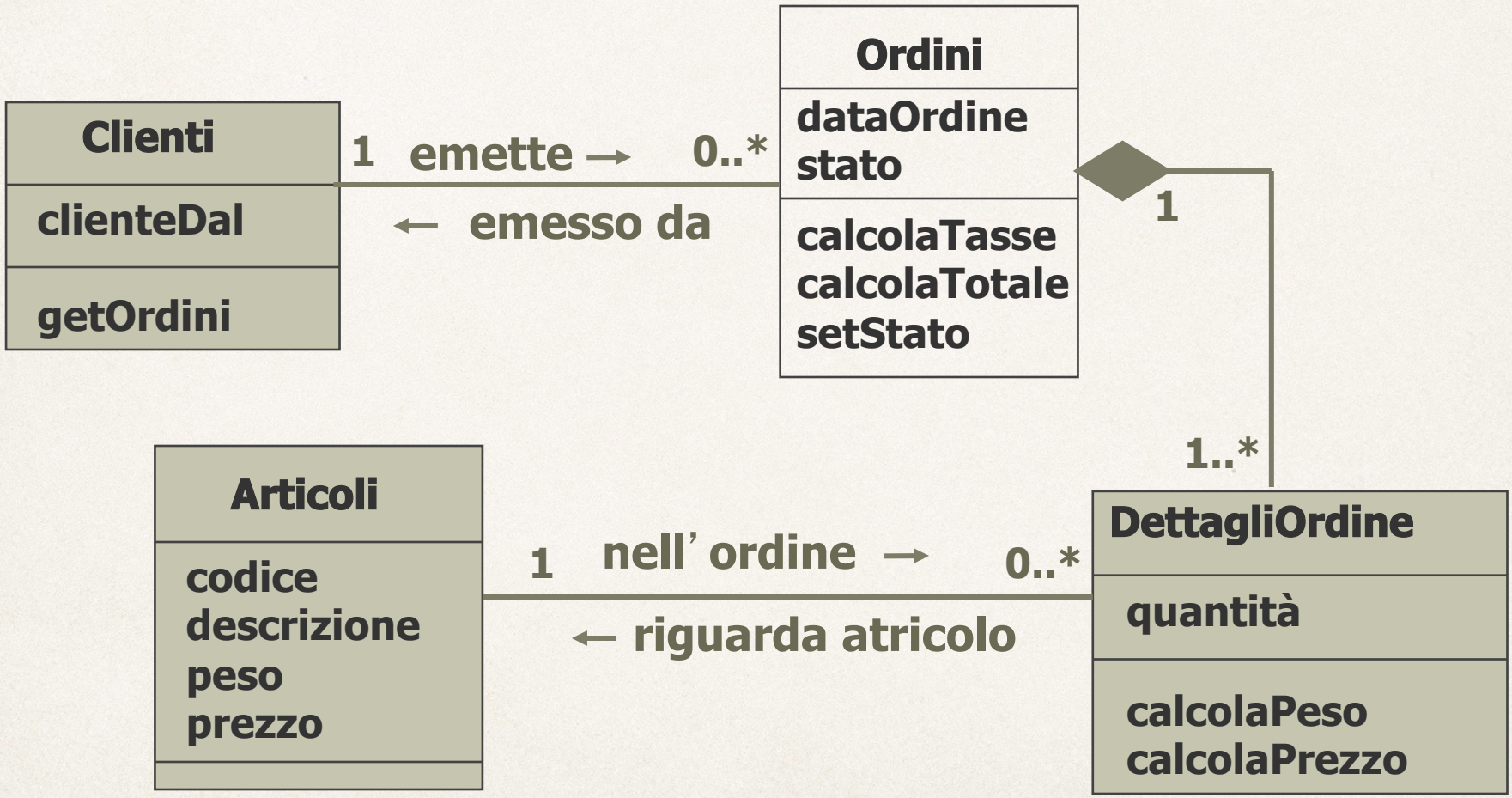
- Si rappresenta come una linea tratteggiata e con lo stereotipo <<uses>> che però di solito viene omesso

Class diagram: associazioni

- Associazione: correlazione tra classi; nel diagramma è una linea continua fra due classi caratterizzata da nome, nomi dei ruoli, molteplicità e navigabilità.
- Nomi e ruoli sono opzionali
- Molteplicità: numero di oggetti che partecipano all'associazione.

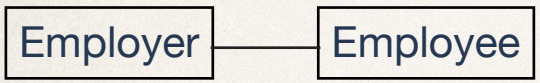
1	Esattamente una
0..*	nessun limite
1..*	almeno una
n..m	da *n* a *m* istanze

Associazioni (cont.)



Associazioni (cont.)

- Navigabilità: indicata con una punta di freccia su un lato dell'associazione, indica una direzione nella relazione
- Specificando la navigabilità si specifica la dipendenza tra le classi
  - Attenzione: i legami bidirezionali sono critici!!




```
class Employer
{  …
   private Employee itsWorker;
}

class Employee
{  …
   private Employer itsBoss;
}
```


```
class Employee
{  …
   private BenefitPackage itsBenefit;
}

class BenefitPackage
{  …
   // non c'è riferimento a Employee
}
```

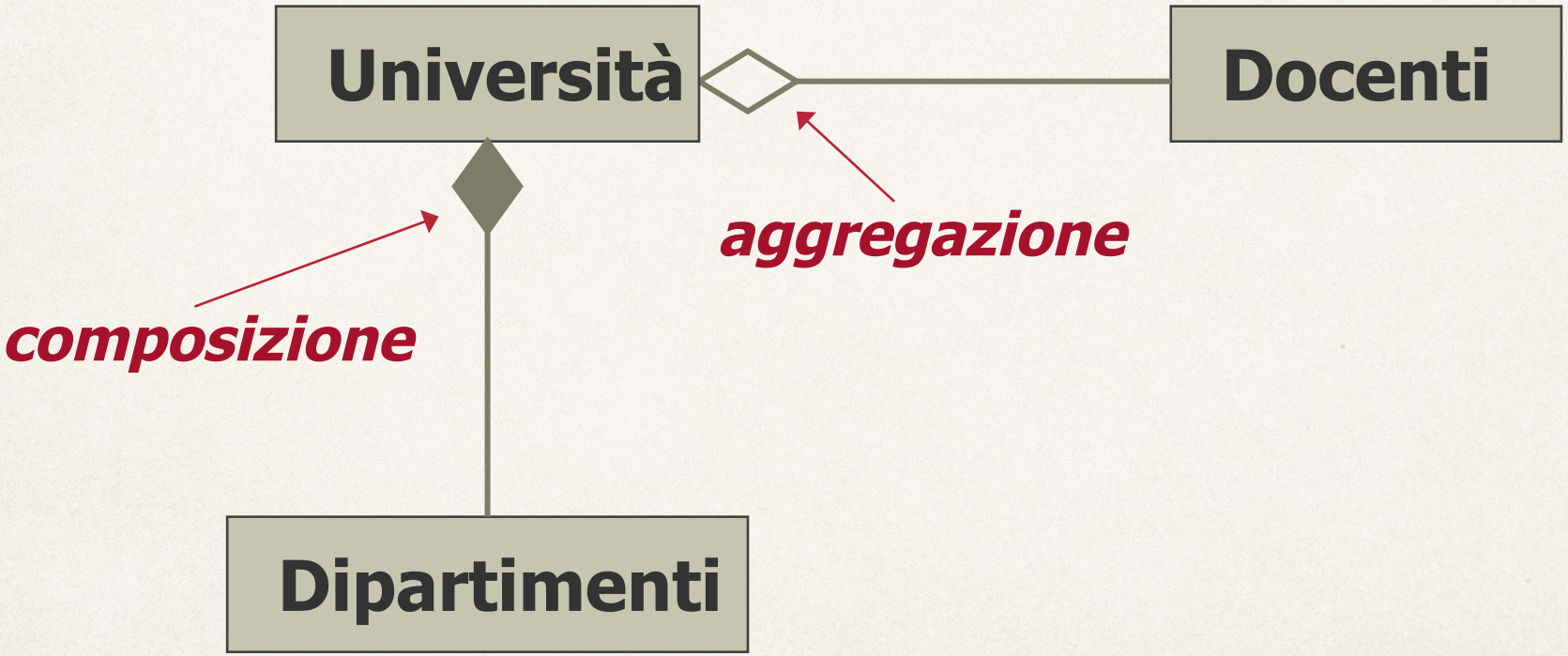
Class diagram: aggregazione

- Aggregazione: è un tipo particolare di associazione; esprime il concetto “è parte di” (part of), che si ha quando un insieme è relazionato con le sue parti
- Si rappresenta con un diamante dalla parte della classe che è il contenitore

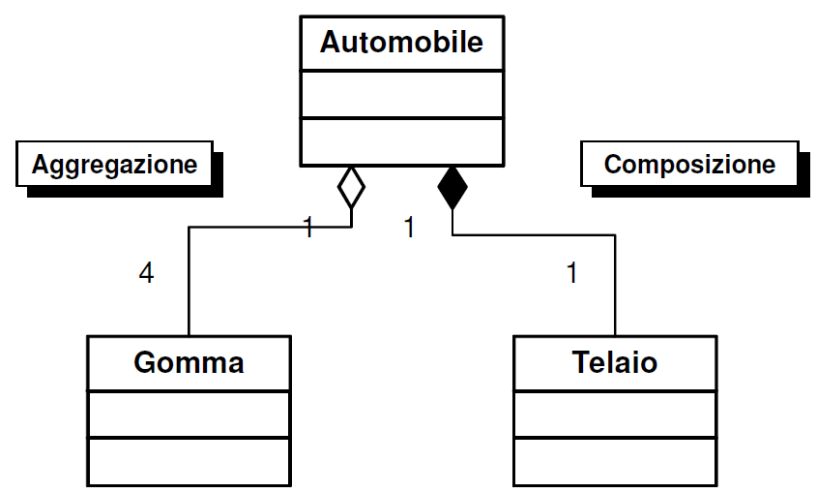
Class diagram: composizione

- E' un caso particolare dell'aggregazione:
- la parte *componente* non può esistere da sola senza la classe *composto*
- una componente appartiene ad un solo composto
- il diamante si disegna pieno

Esempio: aggregazione e composizione

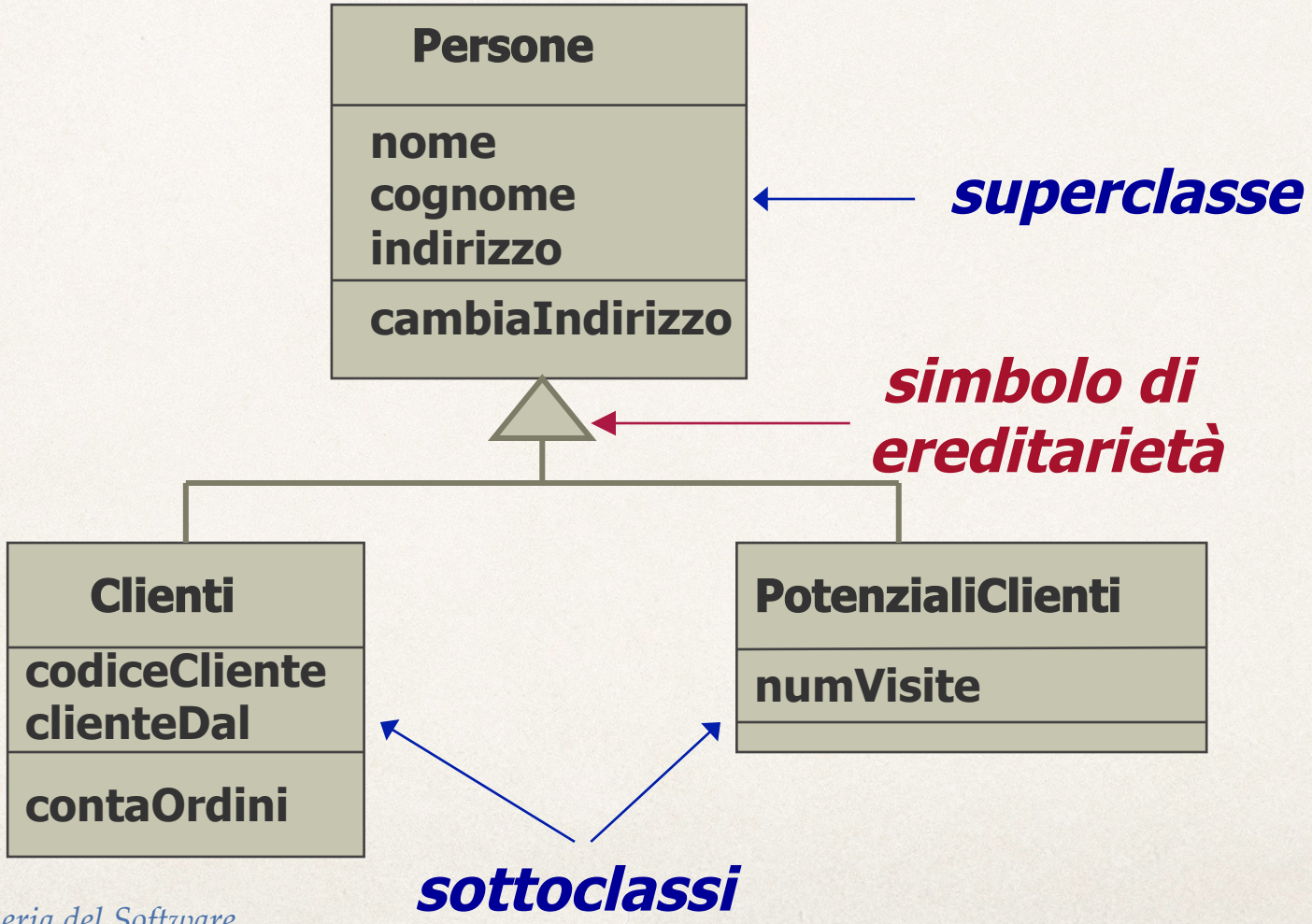


Esempio: aggregazione e composizione




```
class Automobile
{
   private Gomma gomme[];
   private Telaio telaio;
   public Automobile()
   {
      telaio = new Telaio();
      …
   }
   public Gomma getGomma(int n)
   { return Gomme[n];  }
   public void setGomma(Gomma g, int n)
   { gomme[n] = g;  }
}
```

Class diagram: ereditarietà



Class diagram: esempio

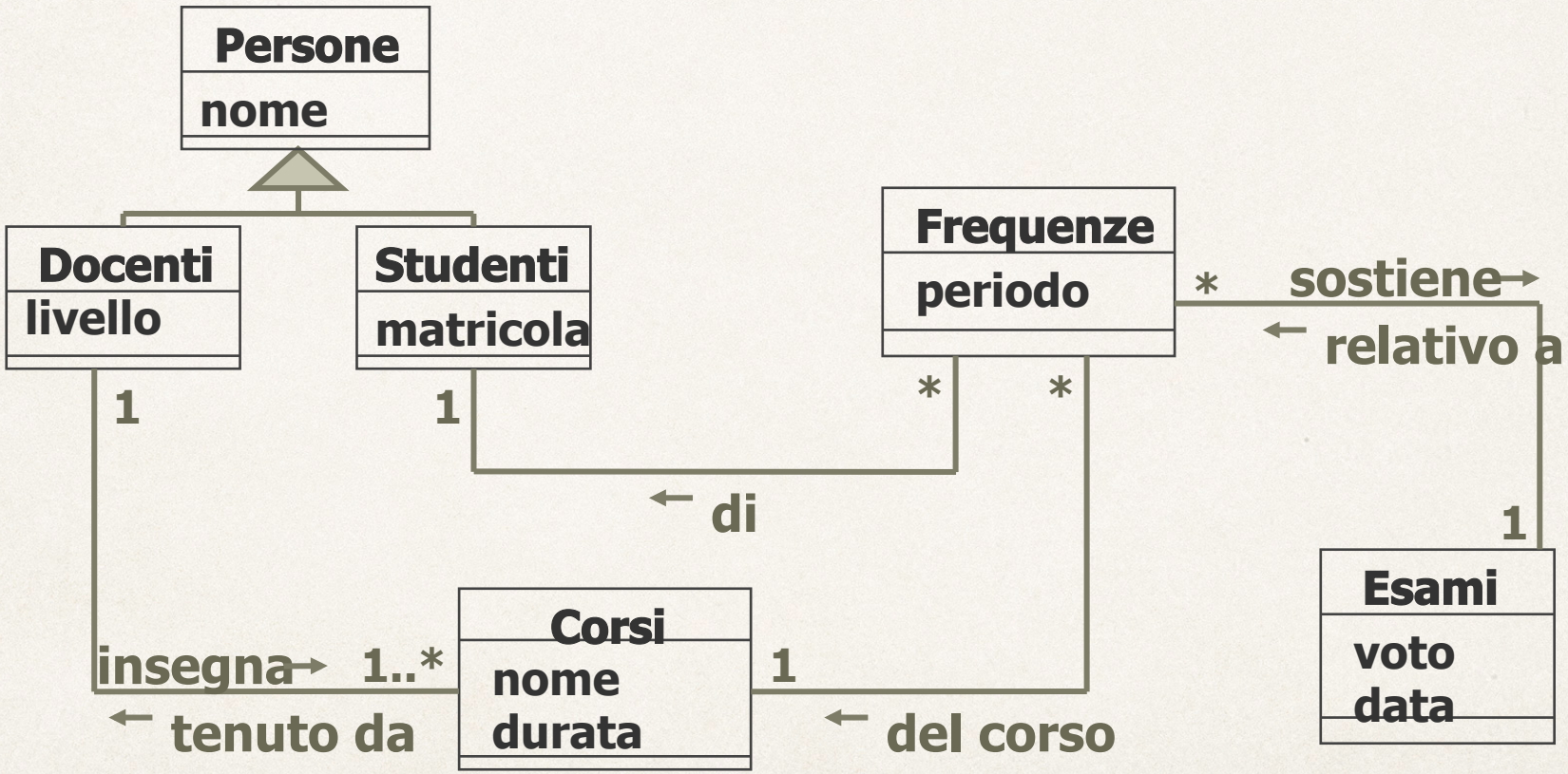


Diagramma di sequenza

- Si usa per definire la logica di uno scenario (specifica sequenza di eventi) di un caso d'uso (in analisi e poi ad un maggior livello di dettaglio in disegno)
- è uno dei principali input per l'implementazione dello scenario
- mostra gli oggetti coinvolti specificando la sequenza temporale dei messaggi che gli oggetti si scambiano
- è un diagramma di interazione: evidenzia come un caso d'uso è realizzato tramite la collaborazione di un insieme di oggetti

Diagramma di sequenza (cont.)

- E' un diagramma che descrive interazioni tra oggetti che collaborano per svolgere un compito
- gli oggetti collaborano scambiandosi messaggi
- lo scambio di un messaggio in programmazione ad oggetti equivale all'invocazione di un metodo

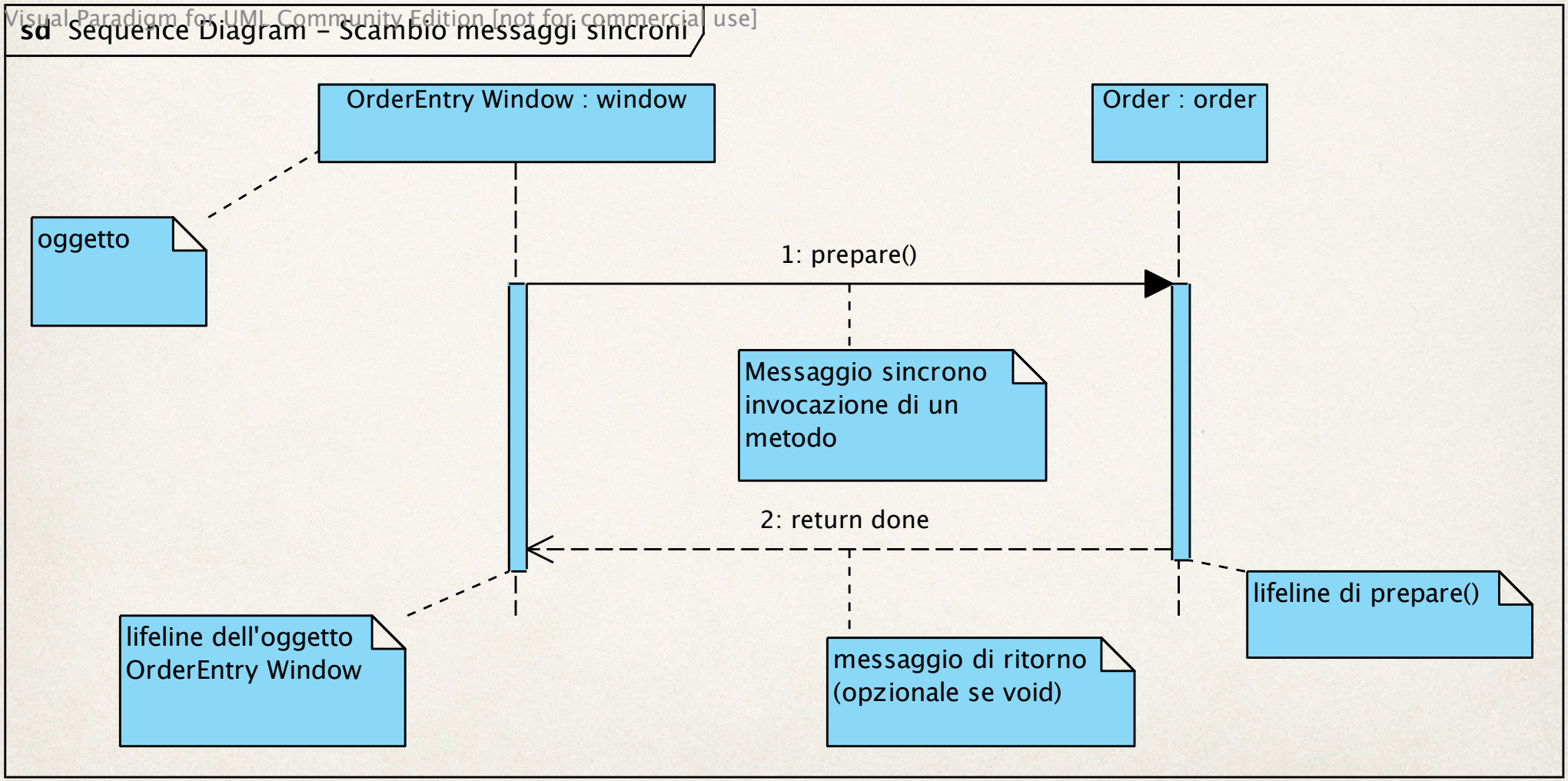
Scambio messaggi sincroni

- Si disegna con un freccia chiusa ⟶ da chiamante a chiamato. La freccia è etichettata col nome del metodo invocato, e opzionalmente con i parametri e il valore di ritorno
- il chiamante attende la terminazione del metodo chiamato prima di proseguire (chiamata bloccante)
- il life-time di un metodo è rappresentato da un rettangolino che collega freccia di invocazione e freccia di ritorno

Messaggi sincroni (cont.)

- 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

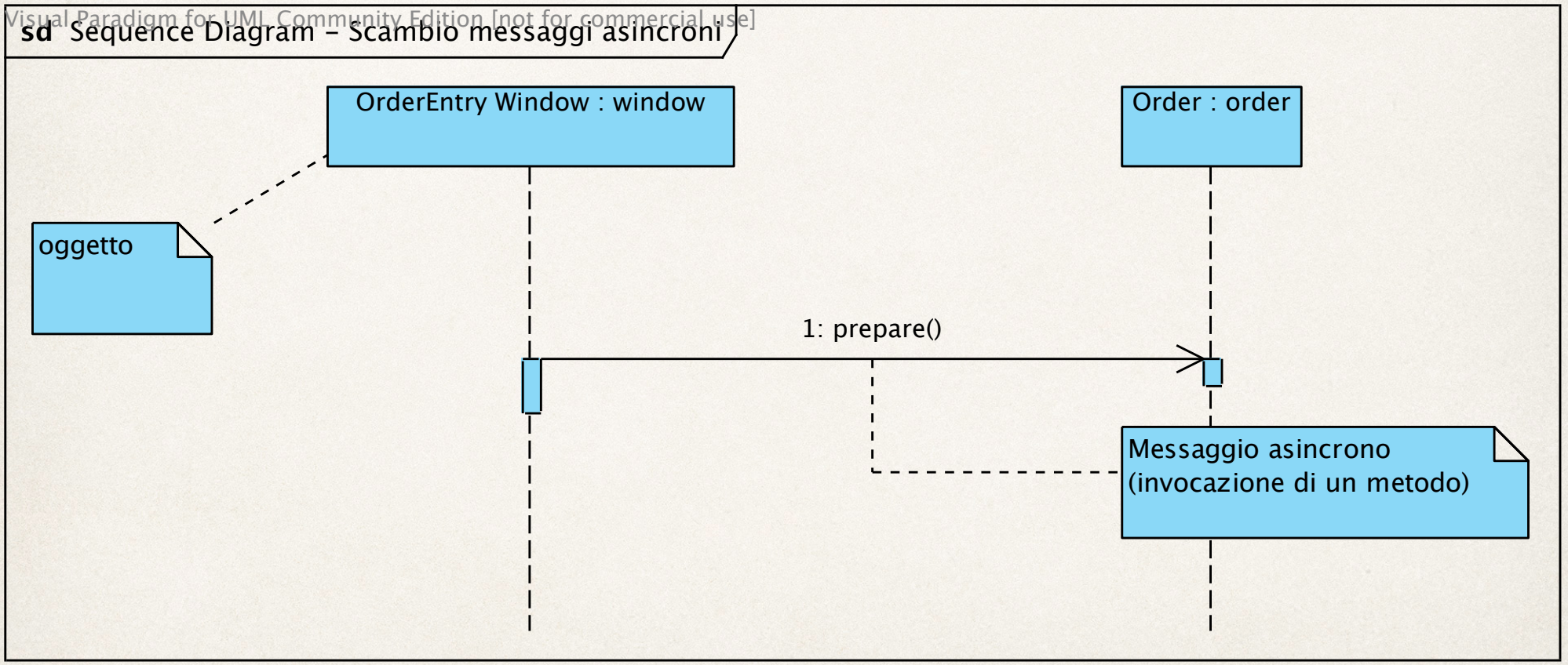
Messaggi sincroni (cont.)



Scambio messaggi asincroni

- Si usano per descrivere interazioni concorrenti
- Si disegnano 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 (non bloccante)
- Il ritorno non segue quasi mai la chiamata

Messaggi asincroni (cont.)

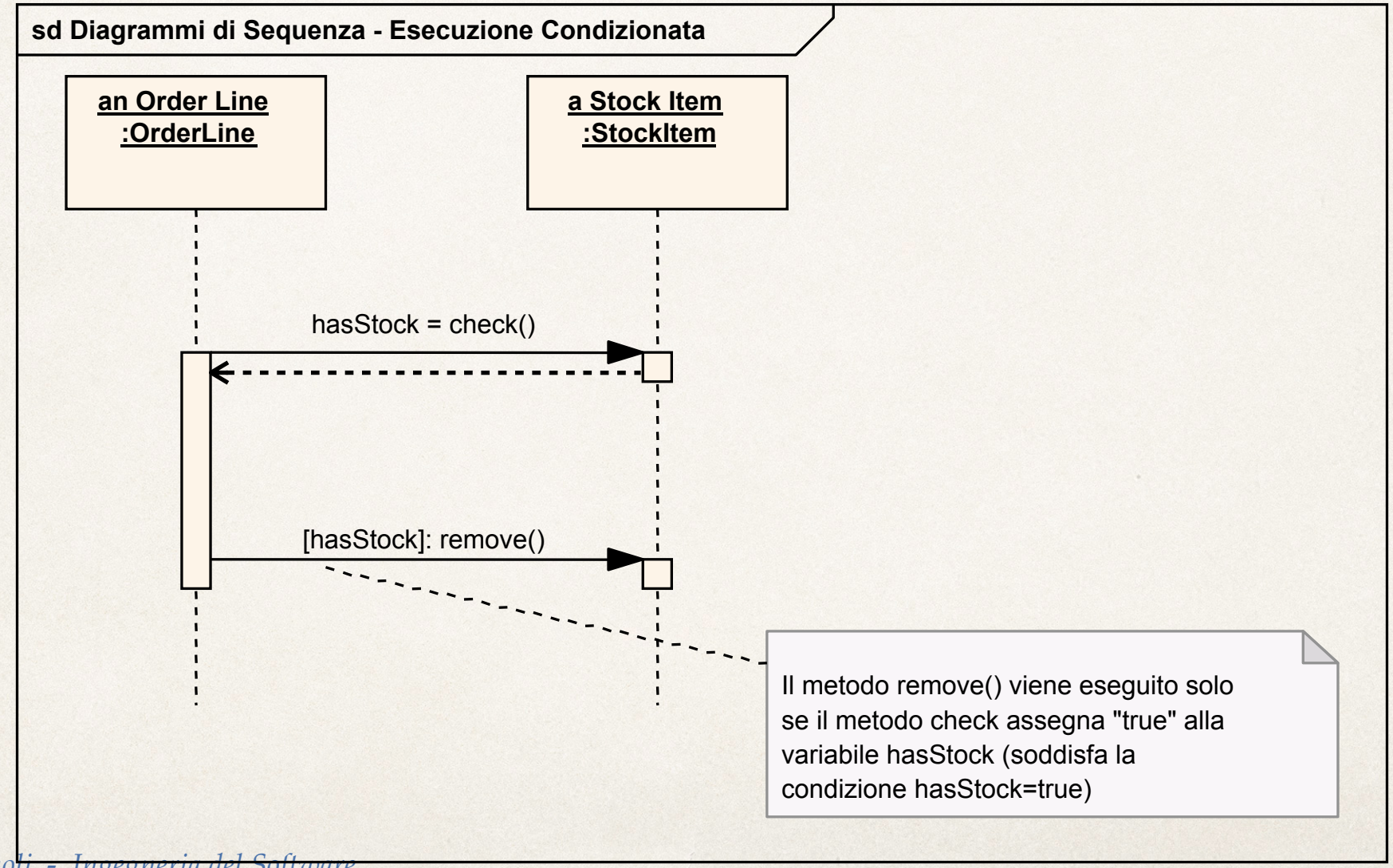


Esecuzione condizionale di un messaggio

- L'esecuzione di un metodo può dipendere da una condizione: i metodo viene invocato solo se la condizione risulta vera a run-time
- Si disegna aggiungendo la condizione, racchiusa tra parentesi quadre, che definisce quando viene eseguito il metodo
- Esempio:

[cond] : nomeMetodo()

Esecuzione condizionale di un messaggio (cont.)

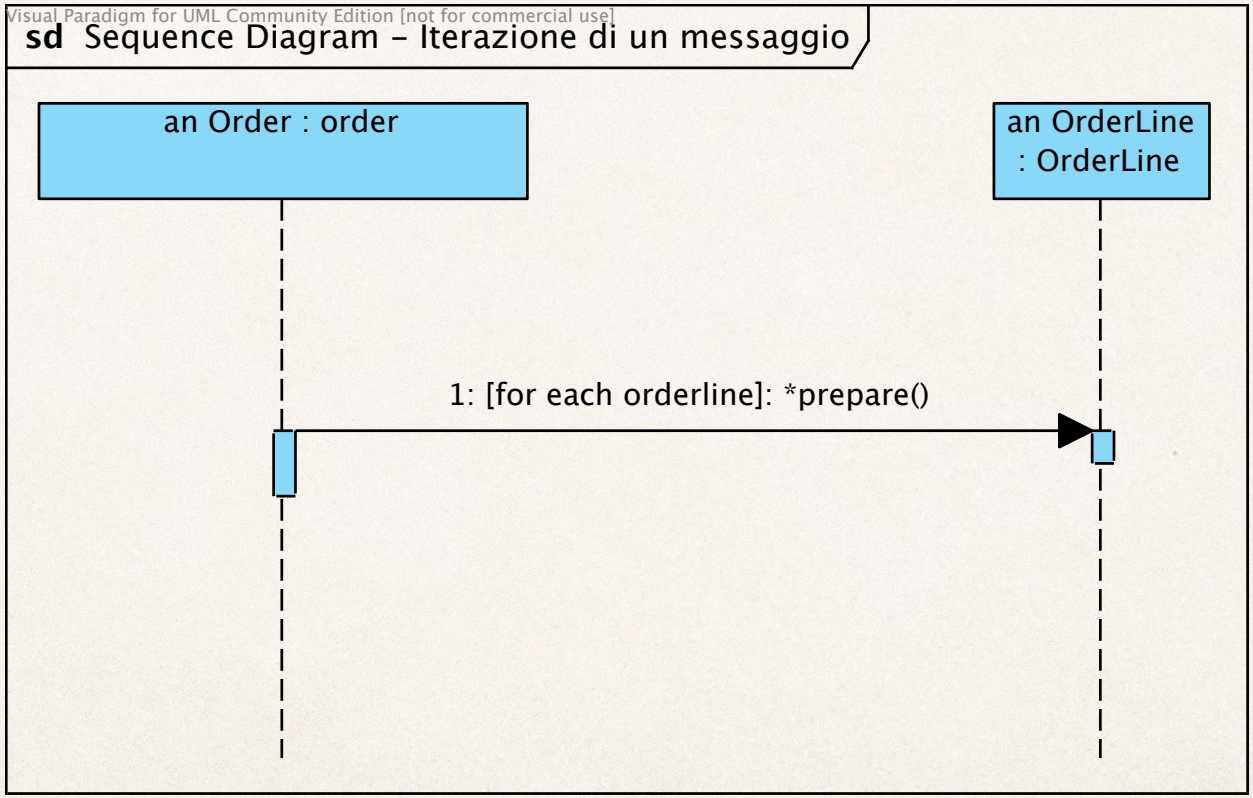


Iterazione di un messaggio

- Rappresenta l'esecuzione ciclica di messaggi
- Si rappresenta 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. Esempio:

[cond] : * nomeMetodo()

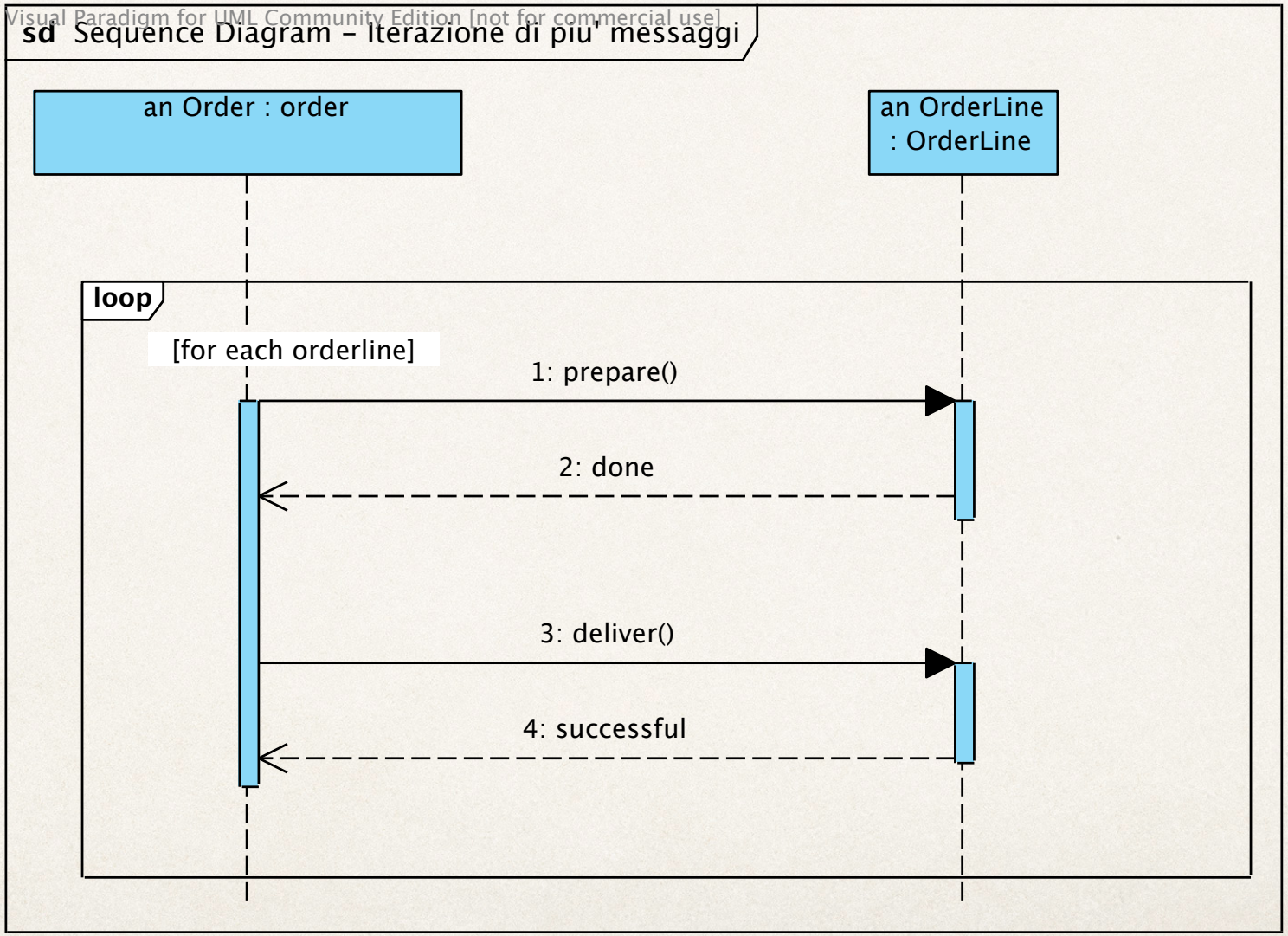
Iterazione di un messaggio (cont.)



Iterazione di un blocco di messaggi

- Rappresenta l'esecuzione ciclica di più messaggi
- Si disegna raggruppando con un riquadro (blocco, 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

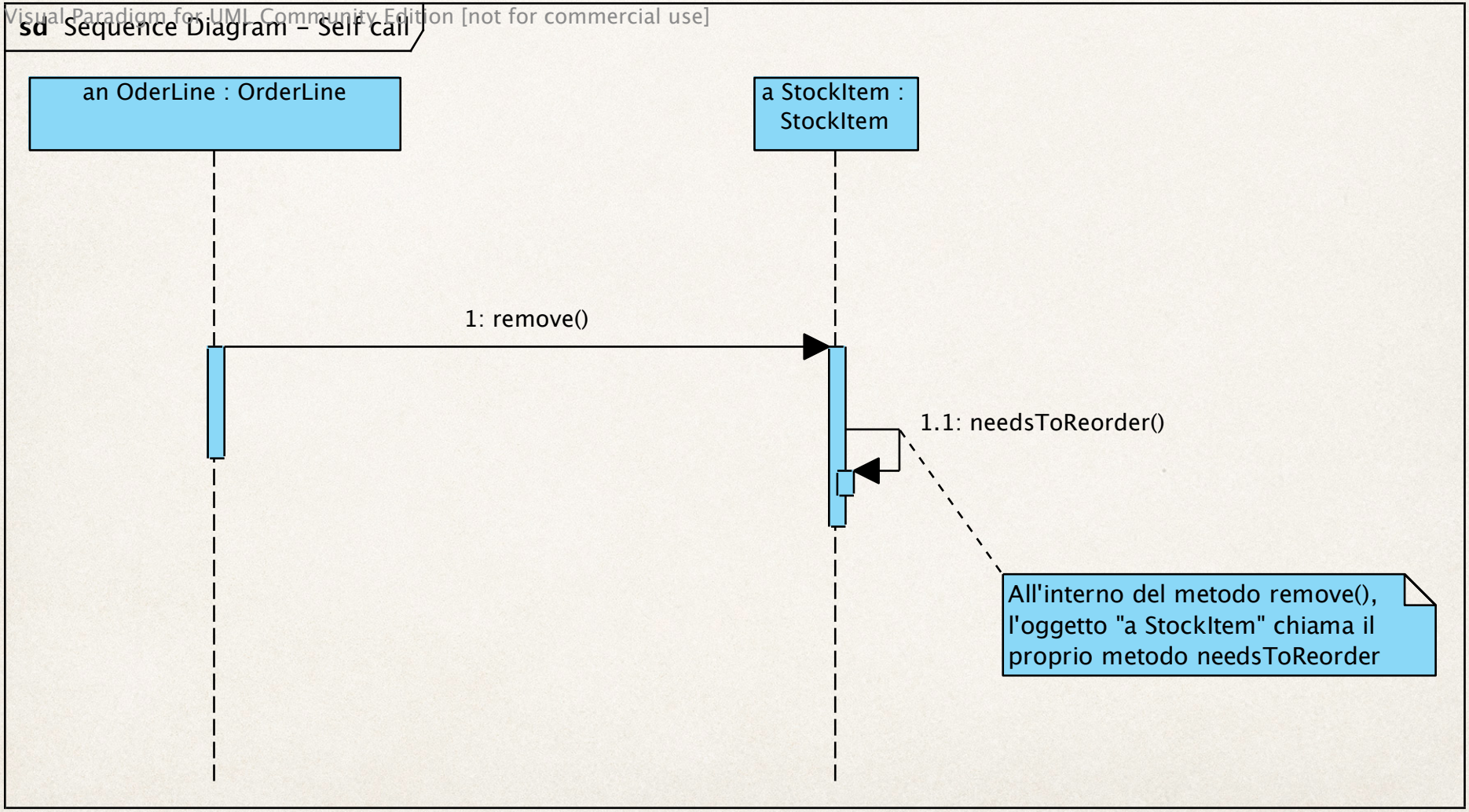
Iterazione di un blocco di messaggi (cont.)



“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

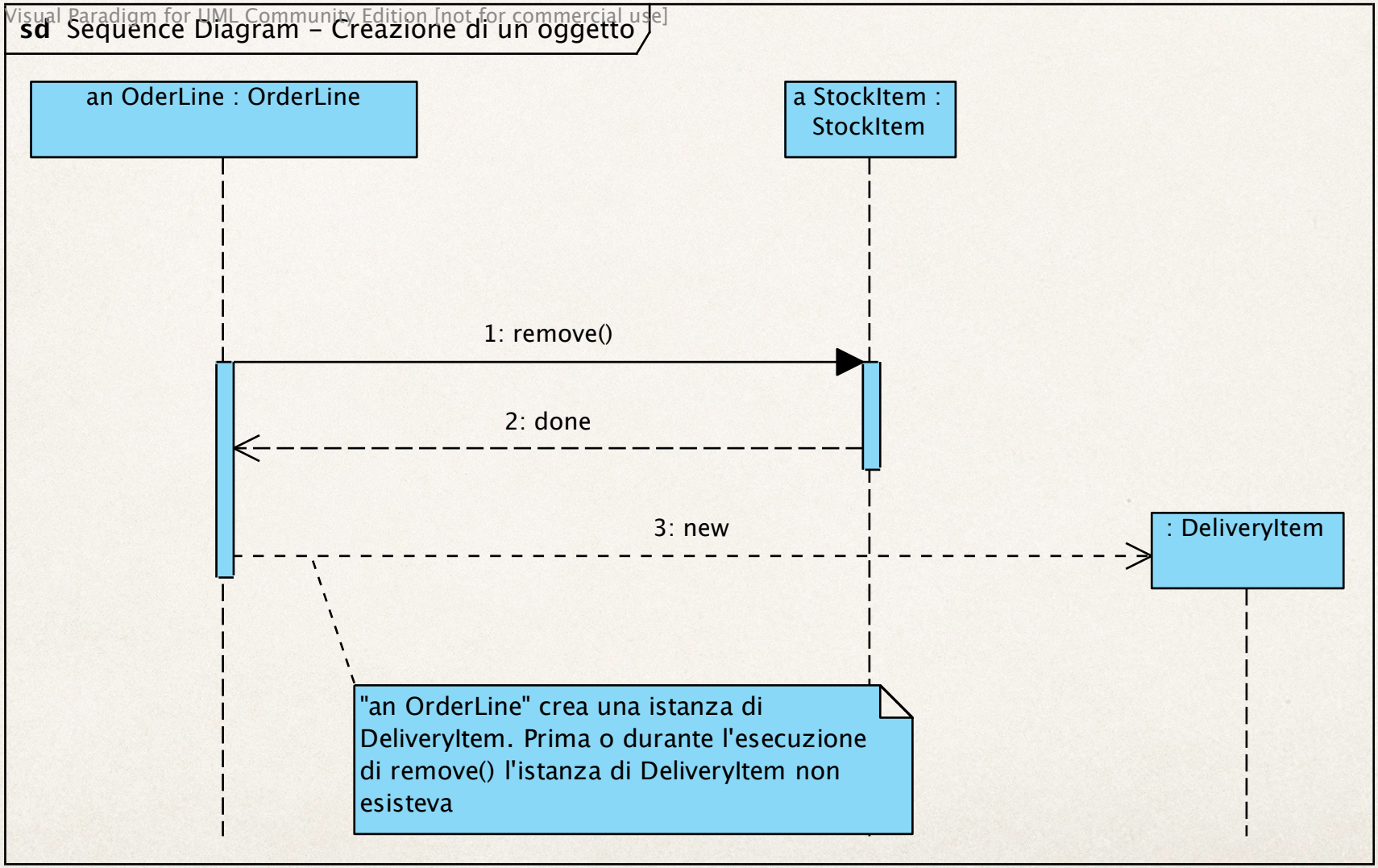
“Auto-Chiamata” (Self-Call) (cont)



Costruzione di un oggetto

- Rappresenta la creazione 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 (cont.)



Eliminazione di un oggetto

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

Eliminazione di un oggetto (cont)