

# 1 UML

**Unified Modeling Language**, é un linguaggio di modellazione (grafico) per progettare e documentare sistemi software.

UML può essere usato dai programmatore in diversi modi:

- **Sketch (schizzo)**: Si schematizzano solamente gli **aspetti più importanti** del sistema, trascurando i dettagli. In questa fase non è strettamente necessario fare riferimento alla codifica standard di UML in quanto il loro scopo, è quello di comunicare selettivamente il funzionamento di un sistema/componente software senza specificarlo in modo completo.
- **Blueprint/progetto (disegno tecnico)**: Si costruisce uno **schema dettagliato** che il programmatore può facilmente tradurre in codice. Spesso si predisponde una figura apposita, il progettista, che ha lo specifico compito di redigere i diagrammi UML tecnici che poi i programmatore dovranno implementare.
- **Linguaggio di programmazione**: i diagrammi UML disegnati dagli sviluppatori vengono direttamente compilati in formato eseguibile e, UML, viene considerato il codice sorgente dell'applicazione.

Approcci possibili:

- **Forward Engineering**: Si disegnano i diagrammi UML prima di scrivere il codice;
- **Reverse Engineering**: Si disegnano i diagrammi UML a partire dal codice esistente;  
UML può essere applicato ai campi piú disparati, anche lontani da quello che é lo sviluppo software pertanto é importante capire il punto di vista dell'autore del diagramma.

## 1.1 Diagrammi dei casi d'uso

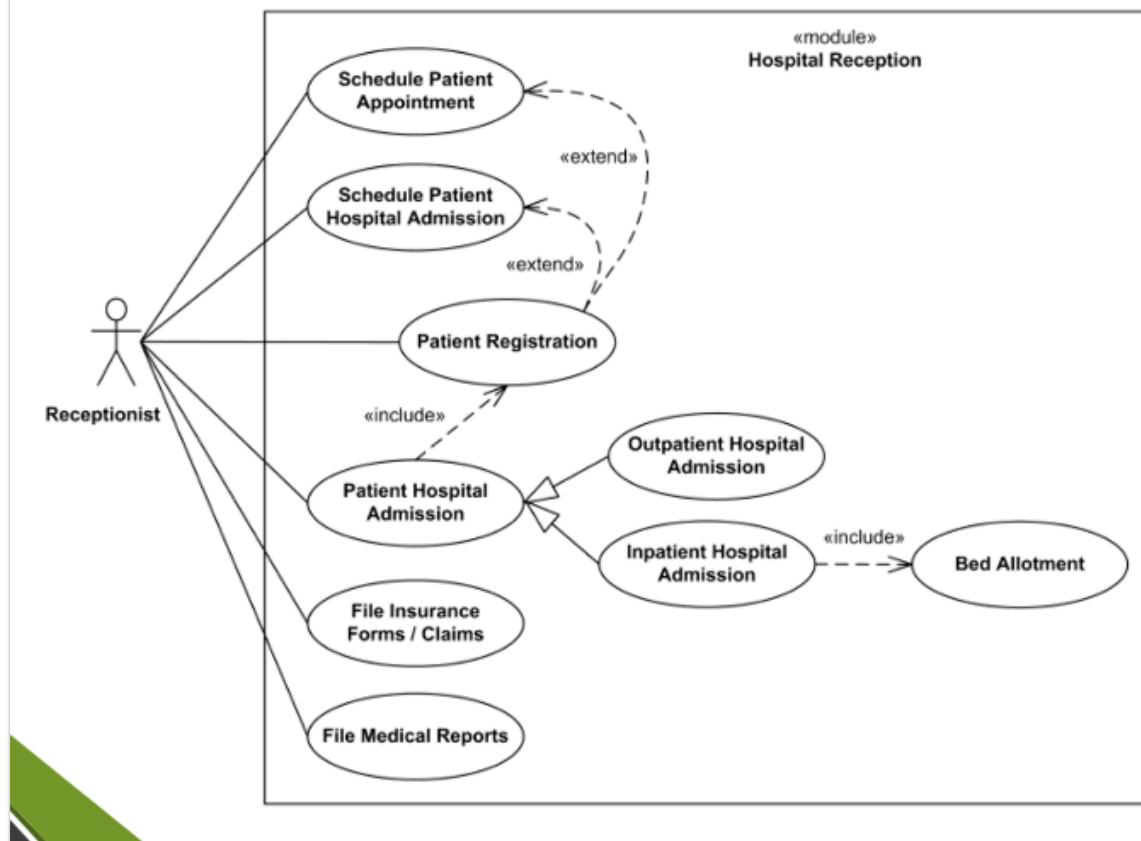
---

Un **diagramma dei casi d'uso** (use case diagram) rappresenta una vista d'insieme dei casi d'uso di un sistema:

- Evidenzia relazioni tra attori e casi d'uso;

- Evidenzia le relazioni tra casi d'uso collegati

# Diagramma dei casi d'uso: Esempio



## 1.1.1 Relazioni tra casi d'uso

### 1.1.1.1 Extend



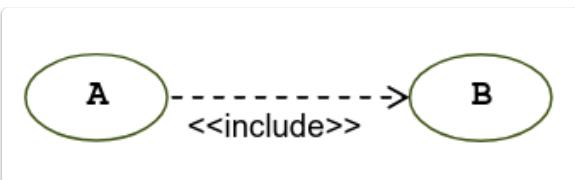
Rappresentano casi eccezionali o raramente invocati.

B estende A quando, al verificarsi di [determinate condizioni](#), l'esecuzione di A comporta anche l'esecuzione di B.

#### Esempio: extend

Annullo Operazione estende Operazione Bancomat, perché il Cliente può decidere di annullare l'operazione (ma di solito non lo fa).

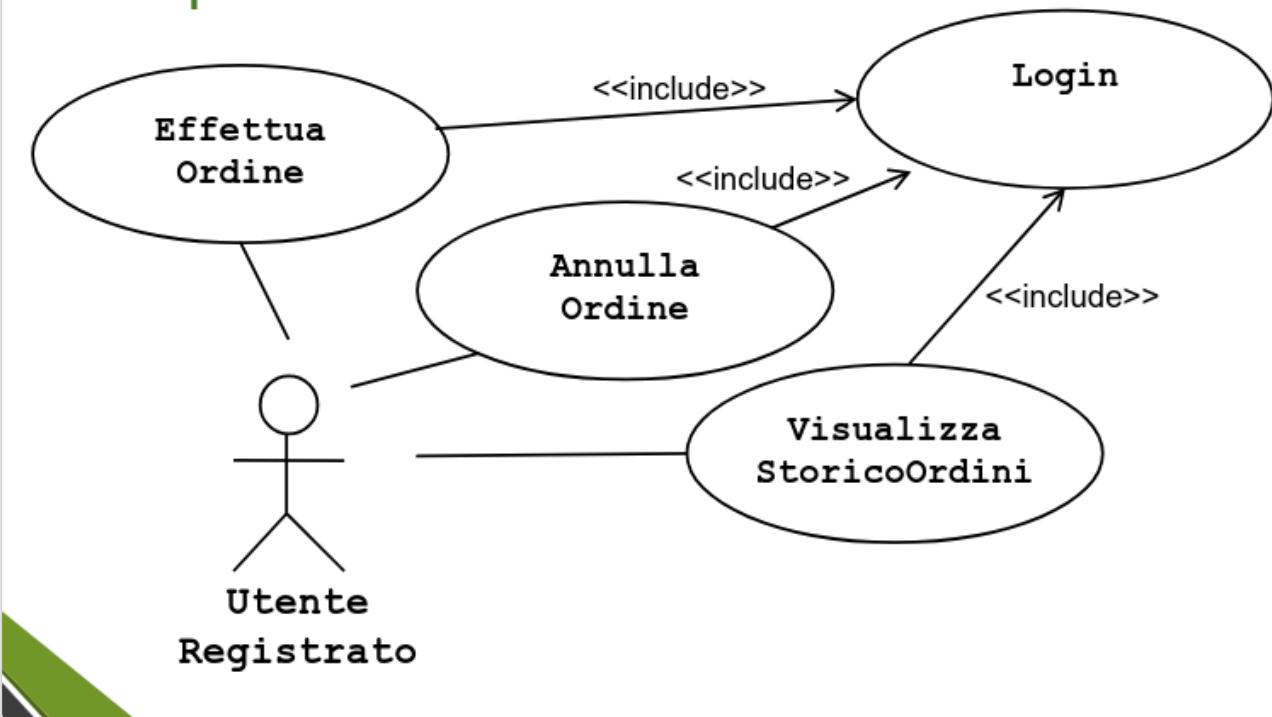
### 1.1.1.2 Include



Un caso d'uso è incluso come passo di un altro caso d'uso.

A include B quando l'esecuzione di A comporta **sempre** anche l'esecuzione di B.

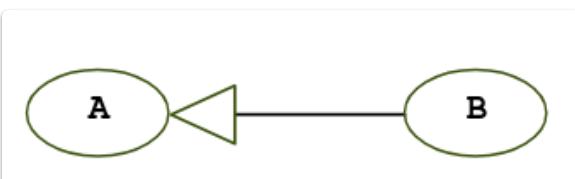
### Esempio:



#### Esempio: include

Operazione Bancomat include Inserimento PIN, perché il Cliente deve sempre inserire il suo PIN (anche se lo ha già fatto per un'altra operazione).

### 1.1.1.3 Specializzazione/generalizzazione

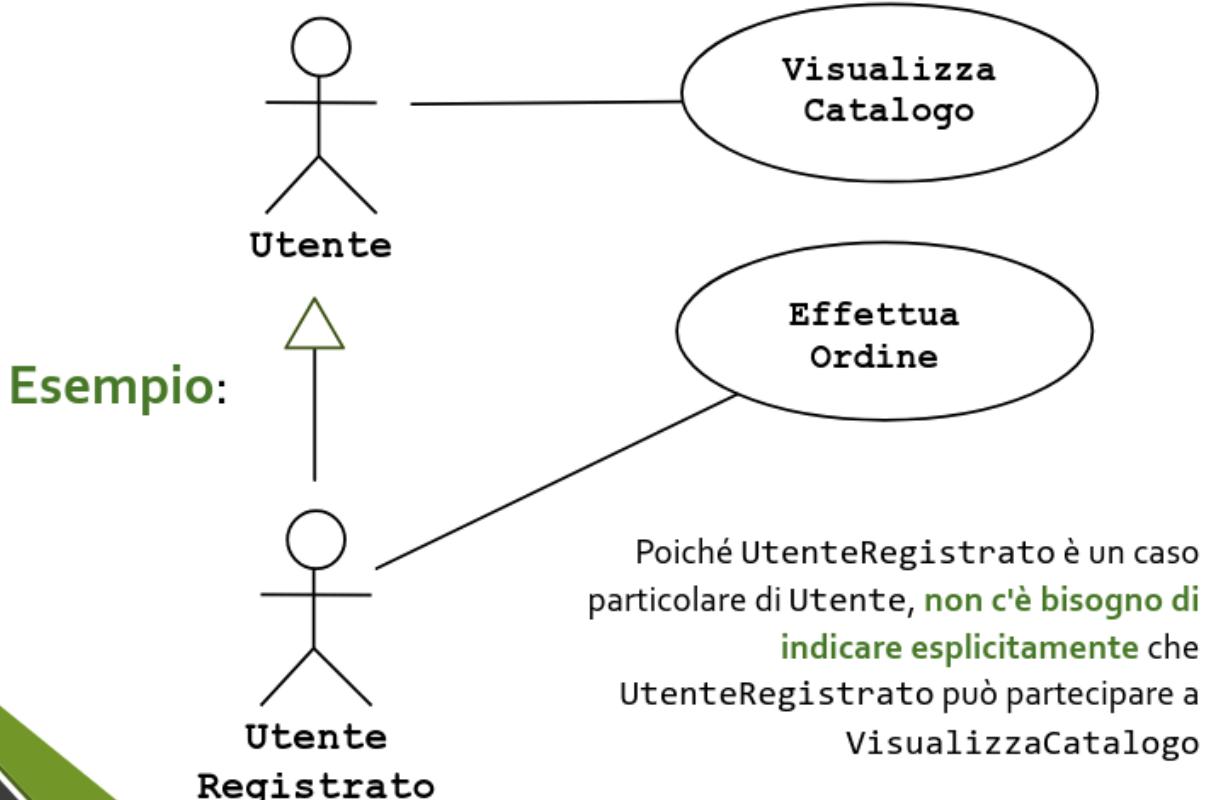


Un elemento è un **caso particolare** di un altro elemento. Si usa principalmente per gli attori.

B specializza A quando B si può considerare un caso particolare di A; l'**esecuzione** di B sostituisce in

determinate situazioni l'esecuzione di A.

## La relazione di specializzazione



### Esempio: specializzazione/generalizzazione

Prelievo Bancomat specializza Operazione Bancomat, perché è una delle operazioni possibili che il cliente può scegliere.

## 1.2 Diagramma delle classi

Sono **modelli statici** che descrivono il sistema mediante le sue classi, gli attributi e le loro interazioni. Sono utili in quanto descrivono come le classi dovranno essere implementate.

**Nome**

**Attributi della classe**

**Metodi della classe**

L'unica informazione obbligatoria è il nome della classe! Per **attributi** e metodi si **può** specificare:

- Informazioni sul **tipo**;
- **Modificatori**: + public, – private, # protected, static, *corsivo* abstract, ~ package

## Employee

**-Name : string**

**+ID : long**

**#Salary : double**

**+getName() : string**

**+setName()**

**-calcInternalStuff(in x : byte, in y : decimal)**

### 1.2.1 Attributi

## Attributi (Proprietà)

### Sintassi:

**visibilità nome: tipo [molteplicità] = default {proprietà}**

### Esempio:

**-nome: String [1] = "Senza titolo" {readOnly}**

- **nome** è l'unico elemento obbligatorio
- **tipo**: che tipo di oggetto può contenere l'attributo
- **molteplicità**: quanti oggetti contiene, ad esempio, **0...1**, **1**, **\***, **1...\***
- **default**: il valore per un oggetto appena creato
- **proprietà** aggiuntive, ad esempio **readOnly**, **ordered**, **unique**.
- **visibilità**: + (public), - (private), # (protected), ~ (package)

### 1.2.2 Metodi

# Operazioni (Metodi)

## Sintassi:

**visibilità nome(lista-parametri): tipo-di-ritorno {proprietà}**

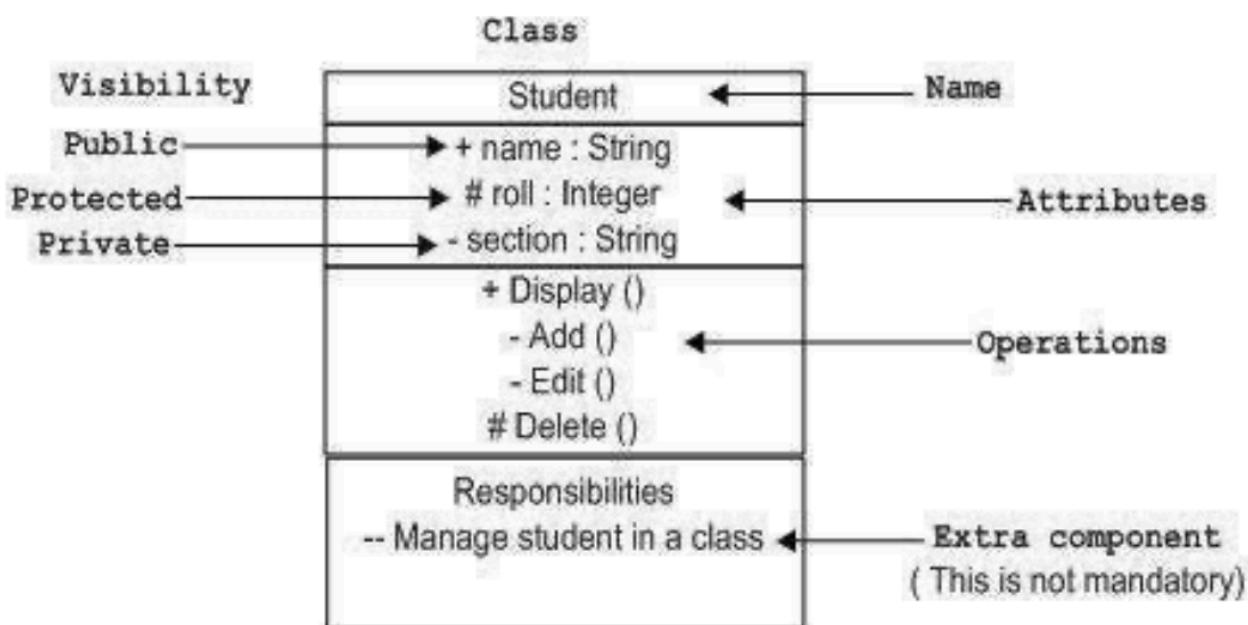
## Esempio:

**+ balanceOn (data: Date): Money**

- **nome** è l'unico elemento obbligatorio
- **tipo-di-ritorno**: il tipo di valore restituito, se esistente
- **proprietà**: proprietà aggiuntive, ad esempio, **query**
- **lista-parametri**: l'elenco dei parametri nella forma:  
**direzione nome: tipo = default**
- **direzione**: indica se il parametro è in ingresso (**in**), in uscita (**out**) o entrambi (**inout**)

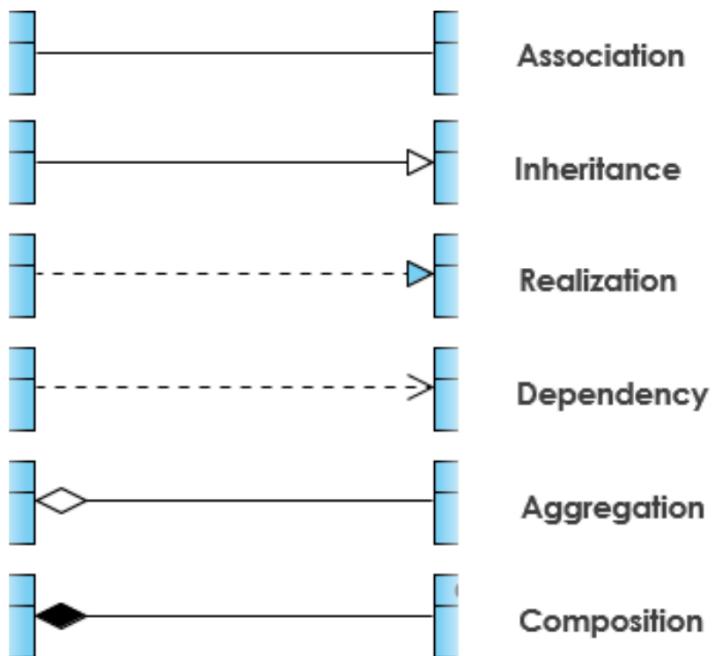
## 1.2.3 Responsabilità della classe

Oltre al nome della classe, gli attributi e i metodi, è possibile predisporre un campo extra in UML che specifica le **responsabilità** di una classe, in linguaggio naturale.



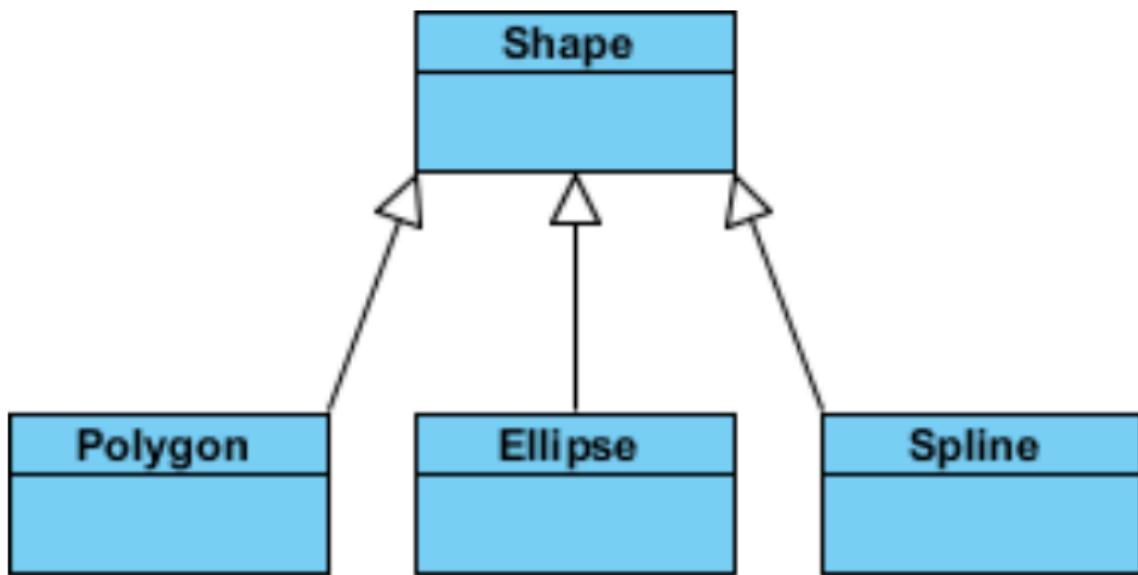
## 1.2.4 Relazioni

Le linee o le frecce tra le classi indicano le **relazioni** tra le classi

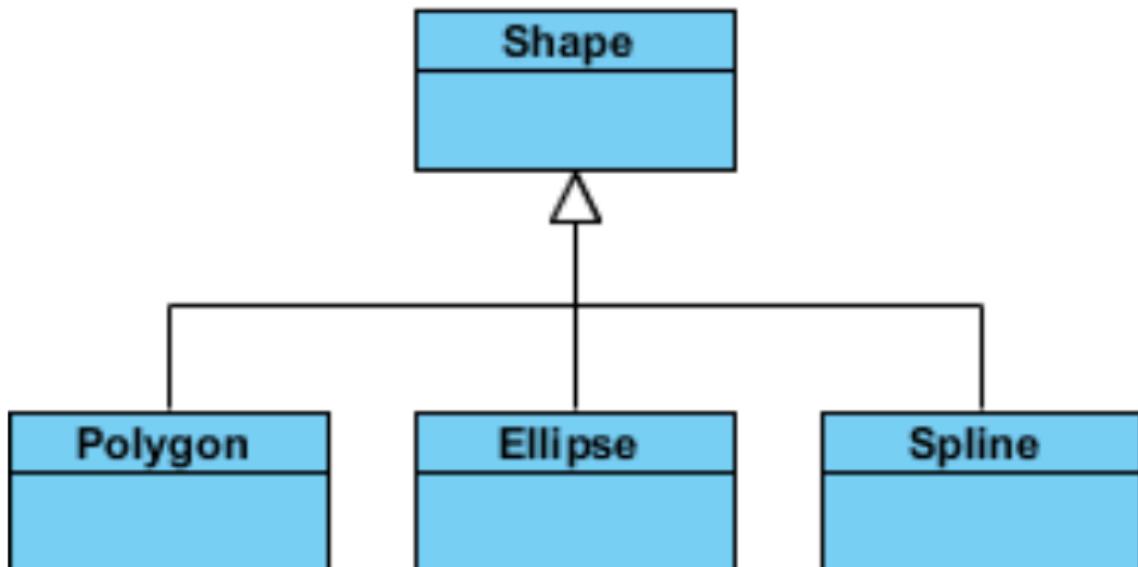


#### 1.2.4.1 Ereditarietà (o specializzazione)

Relazione **superclasse** (più generale) - **sottoclasse** (meno generale). La classe meno generale punta, con un triangolo alla superclasse. Se vista al contrario prende il nome di **generalizzazione**.



**Style 1: Separate target**



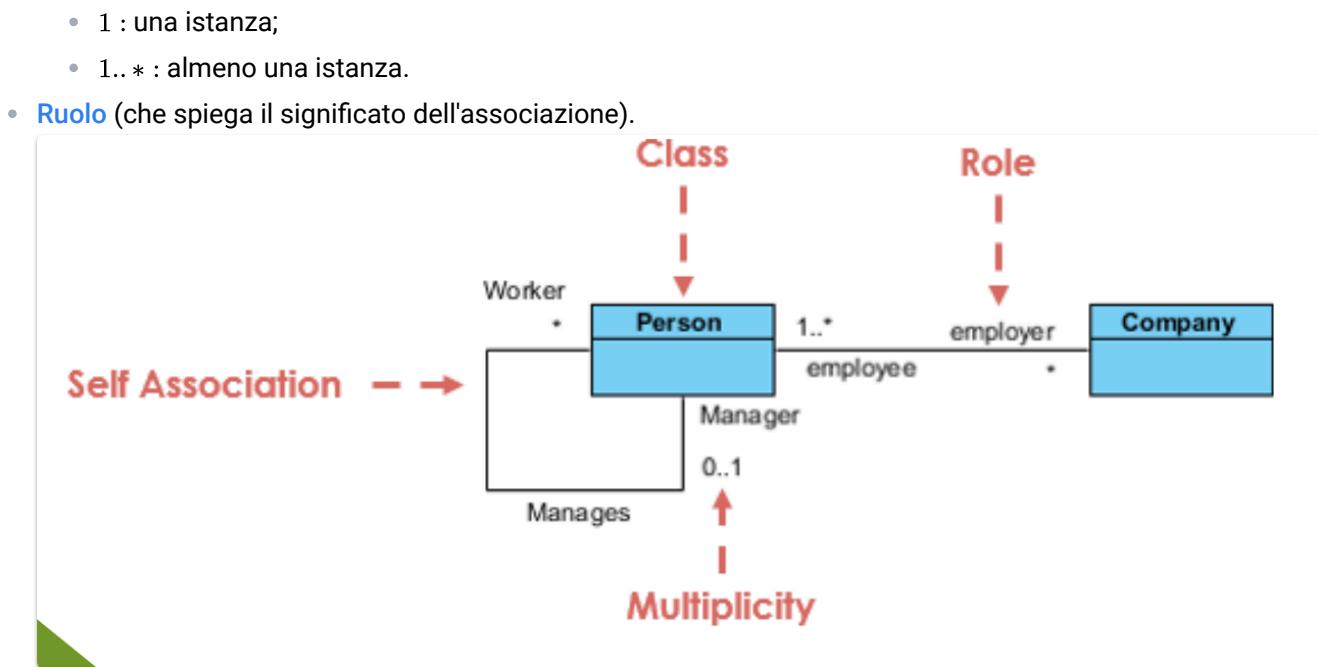
**Style 2: Shared target**

E' una relazione "is - a", la quale corrisponde al concetto di **sottoinsieme** (eg. un poligono è anche una forma... una forma non è un poligono. I poligoni sono quindi un sottoinsieme delle forme).

#### 1.2.4.2 Associazione

Un oggetto di una classe mantiene un **riferimento** a un oggetto dell'altra per poter svolgere il proprio lavoro. L'associazione è indicata da una **linea retta**. Si può annotare:

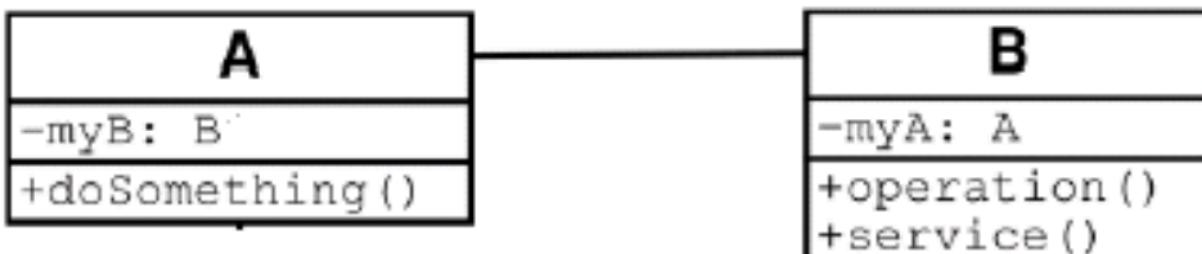
- **Molteplicità:**
  - $n \dots m$  : da  $n$  ad  $m$  istanze;
  - $0..1$  : zero o un'istanza;
  - $0..*$  oppure  $*$  : nessun limite al numero di istanze;



- **Unidirezionale**: A mantiene un riferimento a B ma non viceversa (B non ha un riferimento ad A):

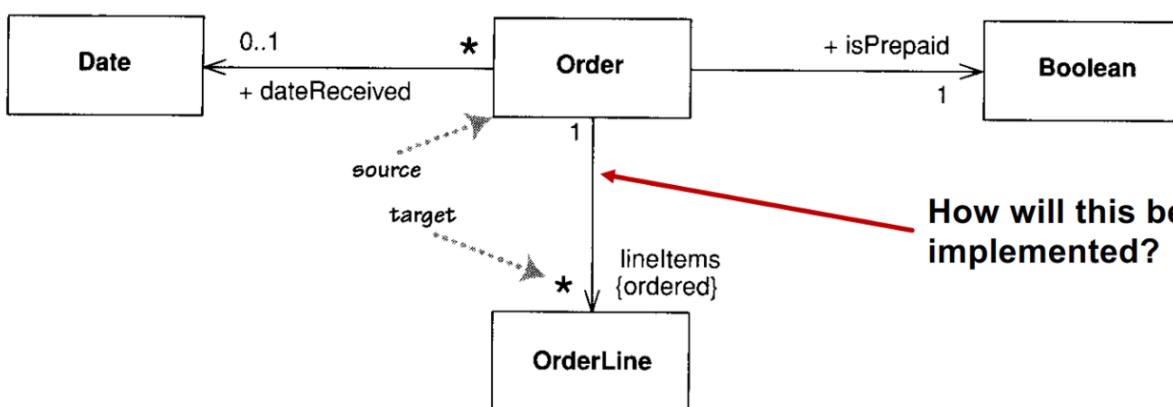


- **Bidirezionale**: entrambe le entità si conoscono:



E' una relazione "*has - a*". Un'associazione con un'altra classe equivale ad avere una proprietà che abbia

come tipo un valore (o una collezione di valori) dell'altra classe.

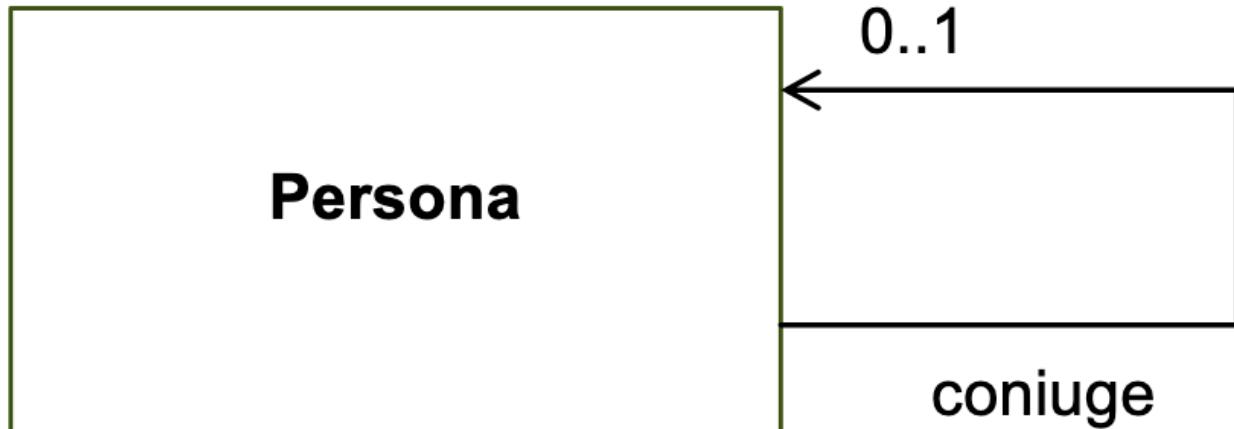


**Usiamo una forma o l'altra a seconda di cosa vogliamo mettere in evidenza**

Order
+ dateReceived: Date [0..1]
+ isPrepaid: Boolean [1]
+ linelitems: OrderLine [*] {ordered}

#### 1.2.4.2.1.1 Self-association

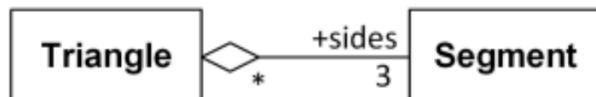
Un oggetto della classe mantiene un [riferimento ad un oggetto della stessa classe](#).



#### 1.2.4.3 Aggregazione "parte-di"

La classe rappresenta un [insieme di parti](#). Si indica con un diamante vuoto sul lato della collezione.

**Un triangolo è un insieme di (esattamente 3) segmenti, che sono i suoi lati.**



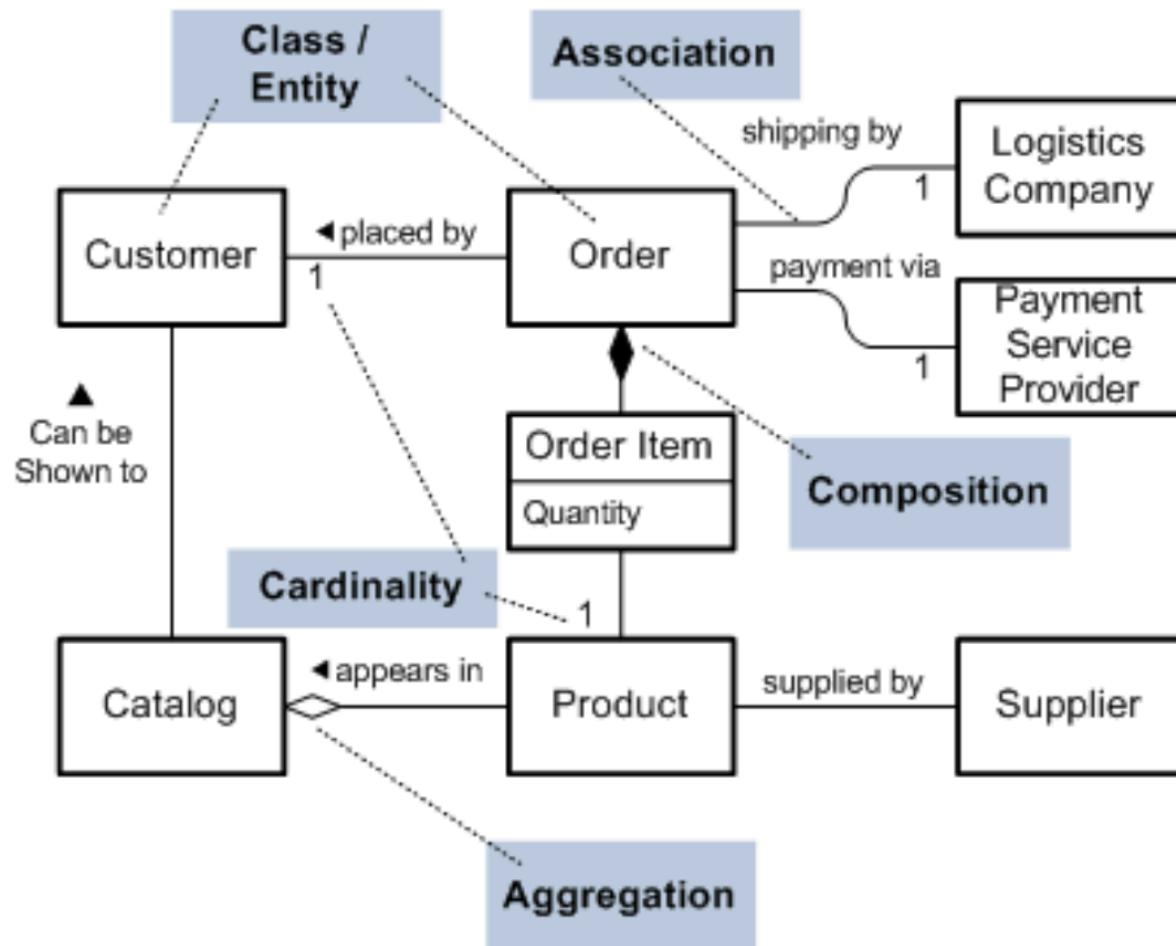
**Nota: un lato può far parte di più di un triangolo.**

L'aggregazione è una relazione asimmetrica

- Esempio di errore:



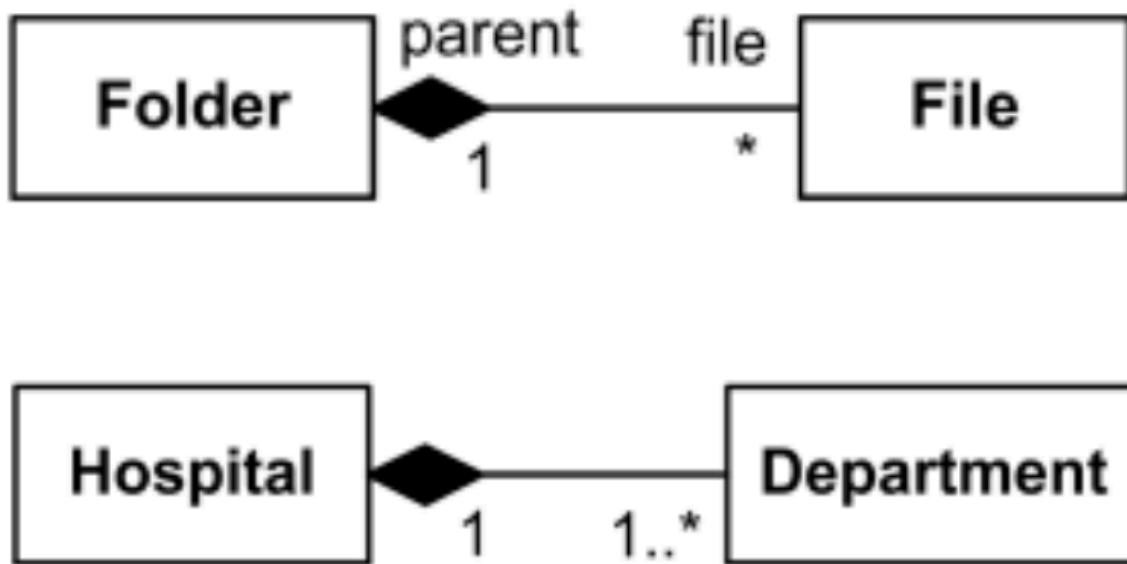
Può essere sostituita con un'associazione uno-a-molti.



#### 1.2.4.4 Composizione

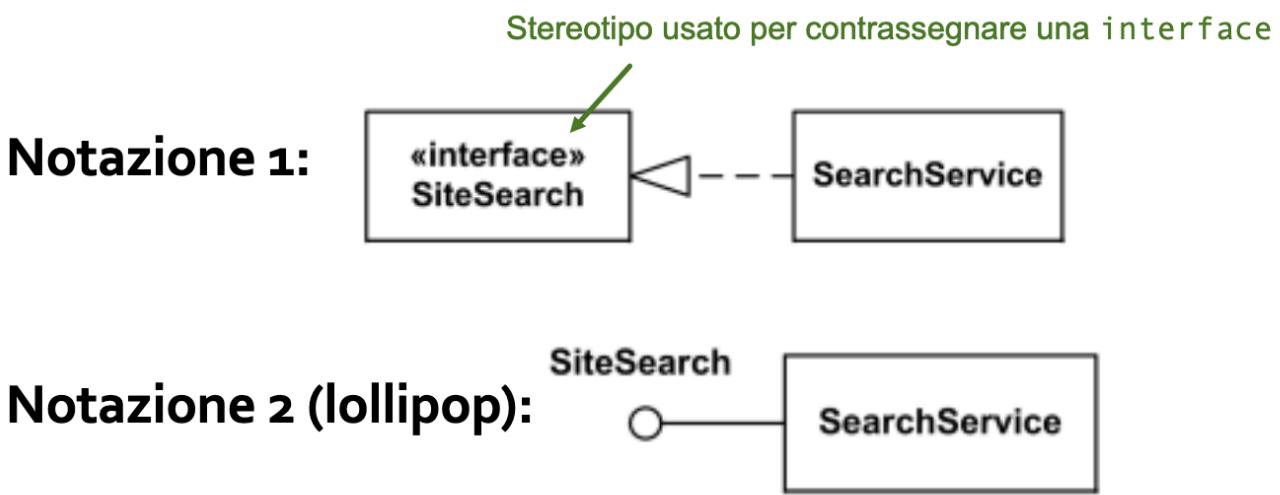
- Una parte può essere inclusa in un solo insieme;
- Una parte non può esistere se non è inclusa nell' insieme, quando viene [rimosso l'insieme](#), vengono [rimosse anche le sue parti](#);

- Si indica con un diagramma pieno sul lato della collezione.



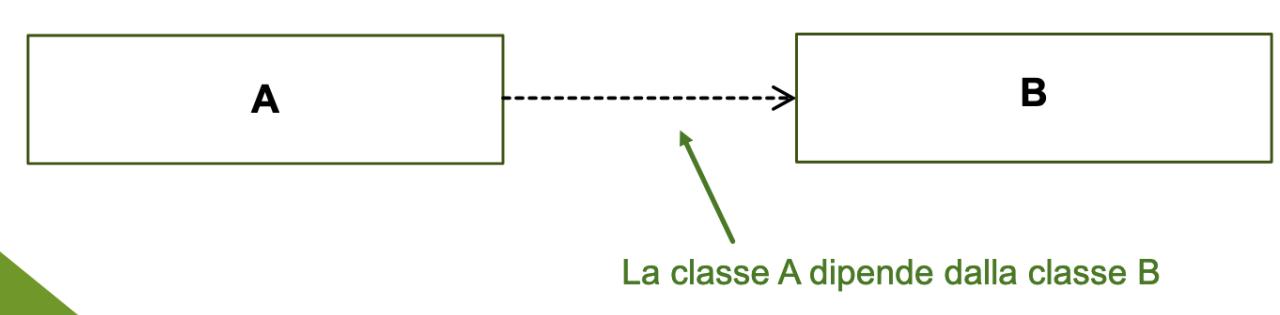
#### 1.2.4.5 Implementazione

Relazione specifica - [implementazione](#). Si usa per mettere in relazione classi con [interfacce](#).



#### 1.2.4.6 Dipendenza

Una classe [dipende](#) da un'altra per un motivo diverso dalle altre relazioni viste in precedenza.



Eg:

- classe A usa classe B in una variabile locale, in un parametro o nel valore di ritorno di un metodo

- classe A richiama un metodo static della classe B.

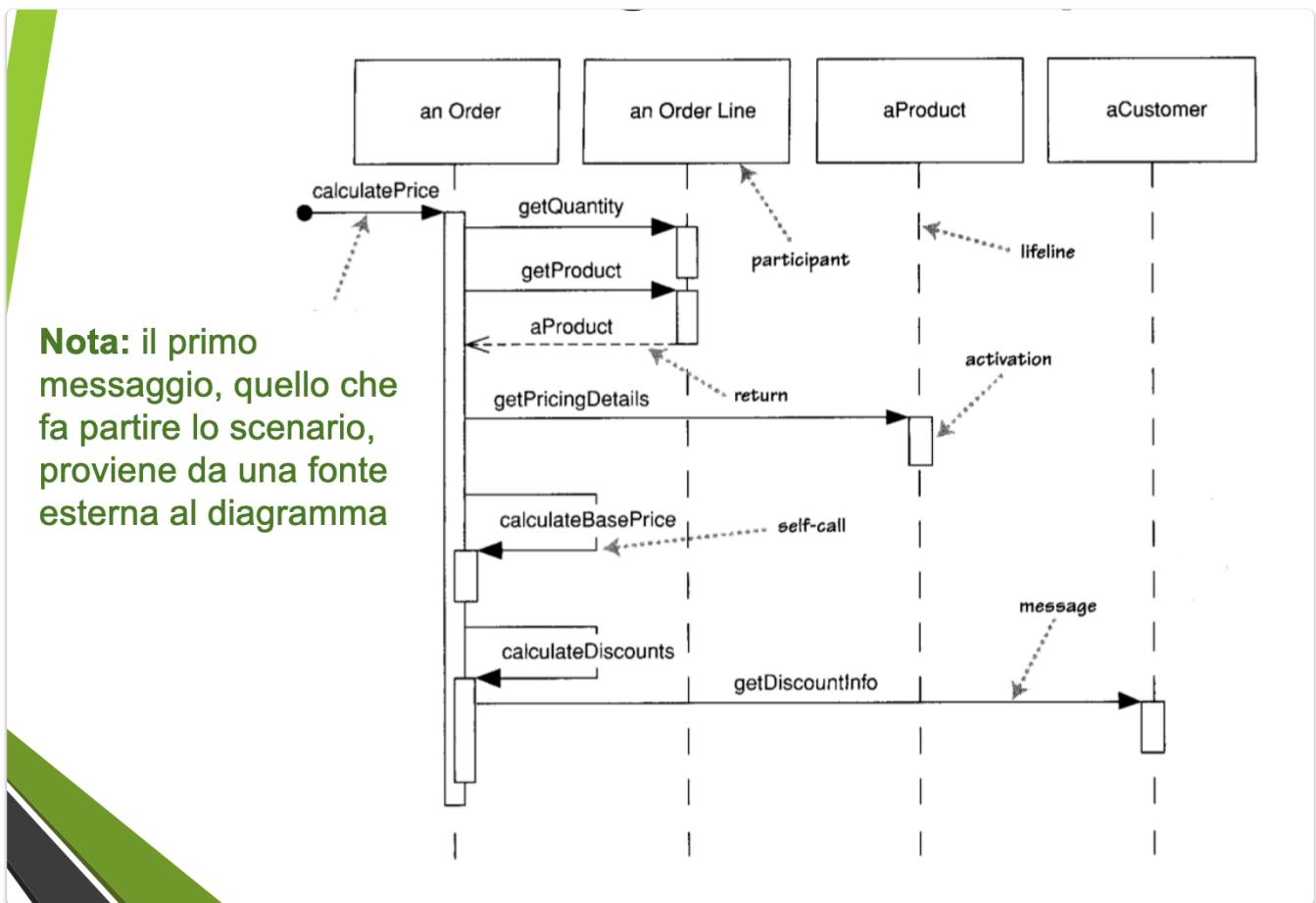
## 1.3 Diagrammi di interazione

Sono dei modelli dinamici che descrivono il modo in cui gruppi di oggetti **collaborano** tra loro. Ne esistono di diversi tipi:

- **Sequenza**
- **Comunicazione**
- Temporali
- Interazione

### 1.3.1 Diagrammi di sequenza

Un **diagramma di sequenza** cattura il comportamento (di un sottoinsieme) del sistema in un singolo scenario. Descrivono sostanzialmente la **collaborazione** di un gruppo di oggetti che implementano un **comportamento collettivo**.



- Il tempo avanza dall'alto verso il basso;
  - Gli oggetti sono elencati da sinistra verso destra;
  - I messaggi tra gli oggetti sono rappresentati mediante frecce.
- I partecipanti possono essere oggetti, metodi, classi o anche gli attori dei casi d'uso.

# Partecipanti

- I nomi dei partecipanti possono specificare solo la classe dell'oggetto:

an Order

an OrderLine

- Una sintassi più ricca è nome:classe, dove il nome è opzionale:

order: Order

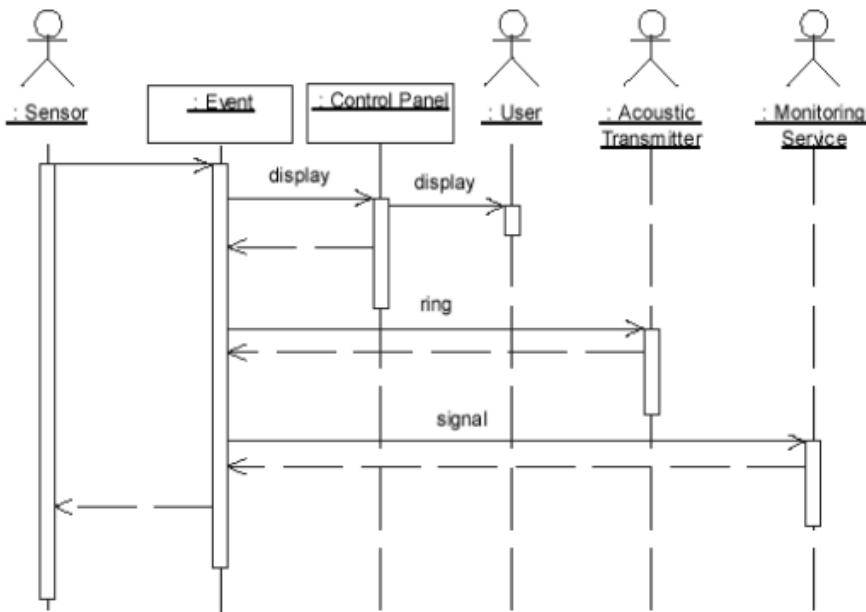
: OrderLine

**Nota:** solitamente (ma non sempre) i nomi sono sottolineati, per analogia con i diagrammi degli oggetti.

- I partecipanti possono essere non solo oggetti ma anche altre entità, come gli **attori** dei casi d'uso.

## Attori come partecipanti

- Utile per modellare il comportamento degli oggetti per implementare i casi d'uso



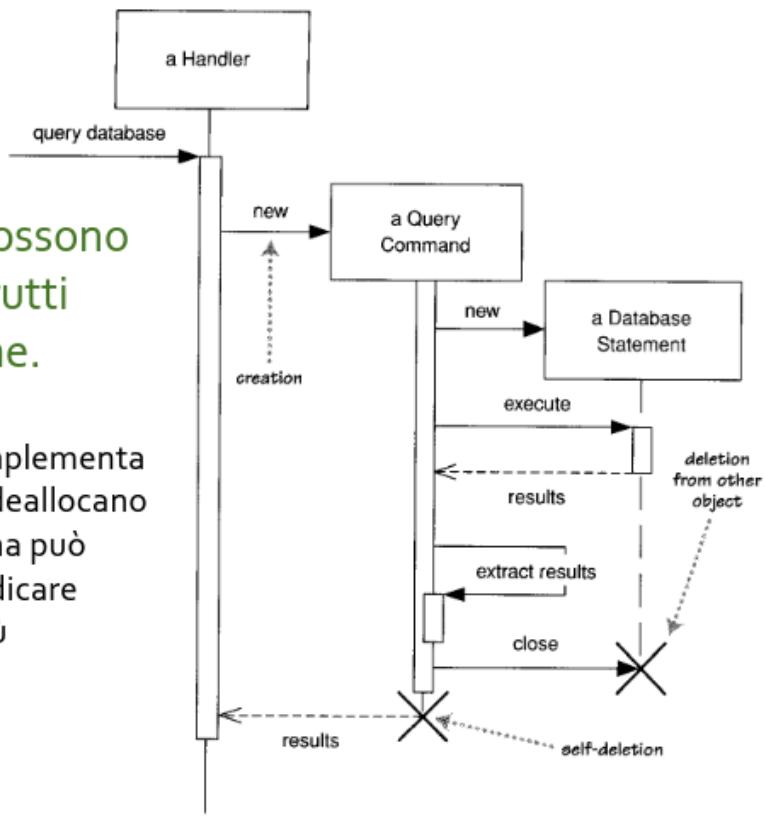
Non è necessario indicare esplicitamente il **ritorno** di ogni metodo inserito nel diagramma anzi, spesso è controproducente e contribuisce solamente a rendere più confuso il diagramma. Conviene esplicitare i

ritorni solamente se aggiungono informazioni utili.

# Creazione e distruzione

Alcuni partecipanti possono essere creati e/o distrutti durante un'interazione.

**Nota:** in un ambiente che implementa un garbage collector non si dealloca esplicitamente gli oggetti, ma può essere utile usare la X per indicare quando un oggetto non è più necessario



- **Creazione:** si indica con una freccia che punta direttamente al Box dell'oggetto creato. Volendo, per chiarire meglio, si può aggiungere un'etichetta del tipo "new";
- **Distruzione:** viene indicata con una grossa "X". Se la freccia di un messaggio termina nella "X", significa che un partecipante ne sta cancellando un altro

## 1.3.1.1 Cicli e condizioni

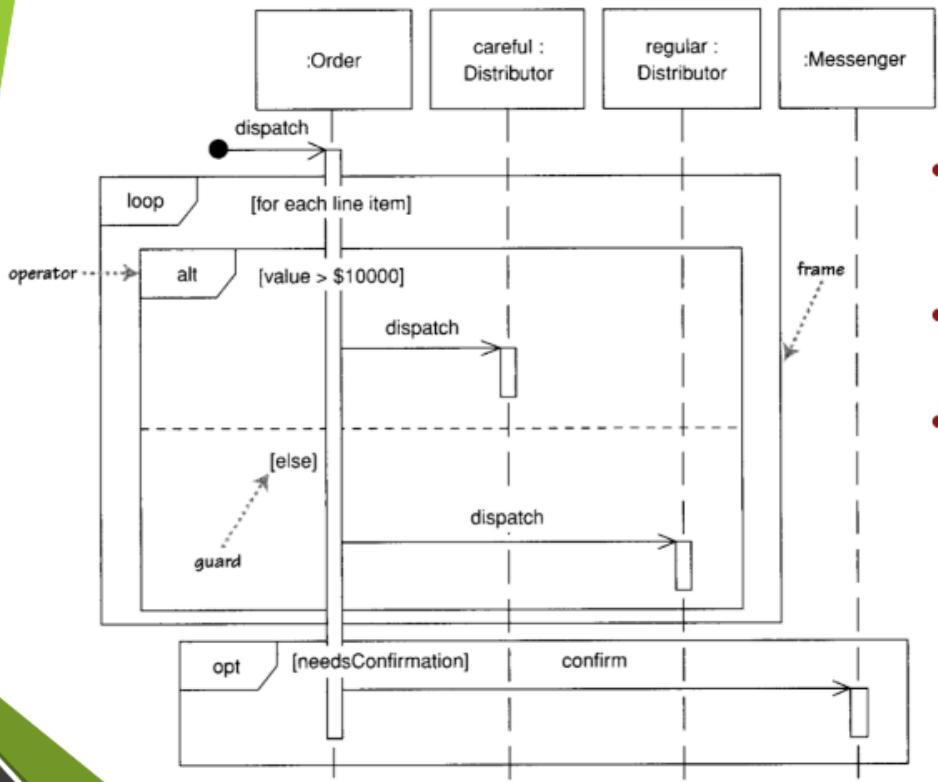
Tali diagrammi sono inadatti a rappresentare **comportamenti ciclici** e condizionali. Se lo scopo del progettista è quello di porre l'accento sulle strutture di controllo, i diagrammi di sequenza non sono una buona idea. Conviene utilizzare **diagrammi delle attività**.

Nei diagrammi di sequenza, comportamenti ciclici e condizionali vengono rappresentati mediante i **frame di intestazione**.

Il blocco contenuto all'interno del frame viene eseguito solamente se una determinata condizione (**guardia**) è verificata. Le guardie sono espressioni che vengono solitamente indicate tra parentesi quadre e, un messaggio

associato alla guardia viene inviato solamente se essa è verificata.

# Cicli e condizioni



- **loop:** il frammento viene ripetuto più volte
- **alt:** frammenti alternativi multipli
- **opt:** il frammento viene eseguito solo se la condizione è verificata (ad esempio, un *if* senza *else*)

## 1.3.1.2 Messaggi sincroni ed asincroni

- Messaggio **sincrono**: il chiamante deve attendere che l'elaborazione del messaggio sia completata prima di continuare (eg. invocazione di un metodo);
- Messaggio **asincrono**: il chiamante non deve attendere una risposta e può continuare subito l'elaborazione (eg. multithreading)

**messaggio generico**  
(non vogliamo specificare se è asincrono o sincrono)



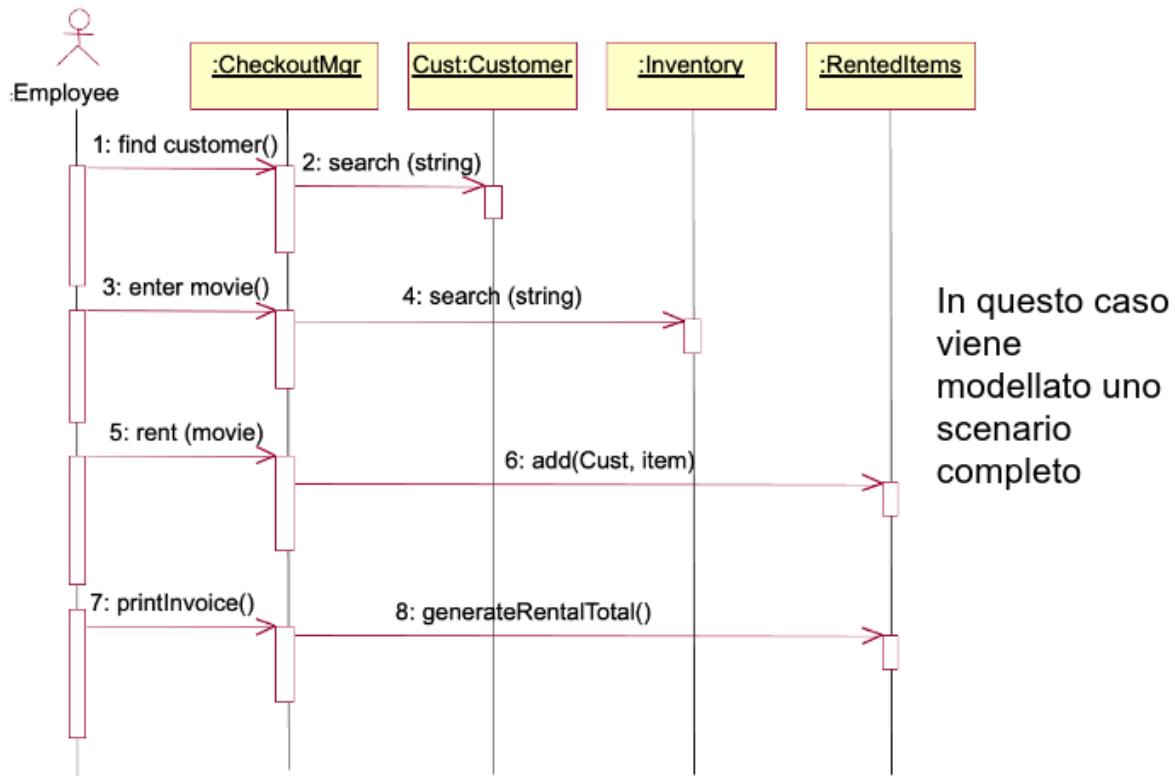
**messaggio sincrono**



**messaggio asincrono**



## Esempio



### 1.3.2 Diagrammi di comunicazione (o collaborazione)

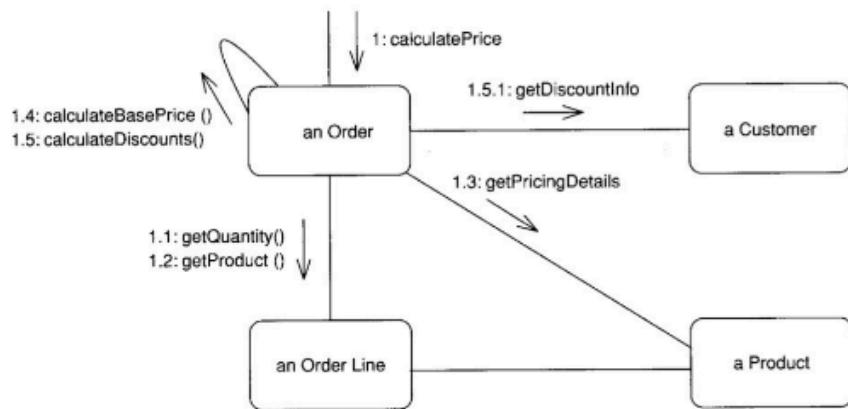
Sono speciali diagramma di interazione che enfatizzano la sequenza di chiamate. Sostanzialmente mostrano le informazioni dei [diagrammi di sequenza](#) in un [formato differente](#)... spesso infatti vengono usati al posto dei diagrammi di sequenza.

Elementi del diagramma:

- Oggetti che [partecipano](#) ad uno scenario;
- [Collegamenti](#) tra oggetti;

- **Messaggi** scambiati tra gli oggetti

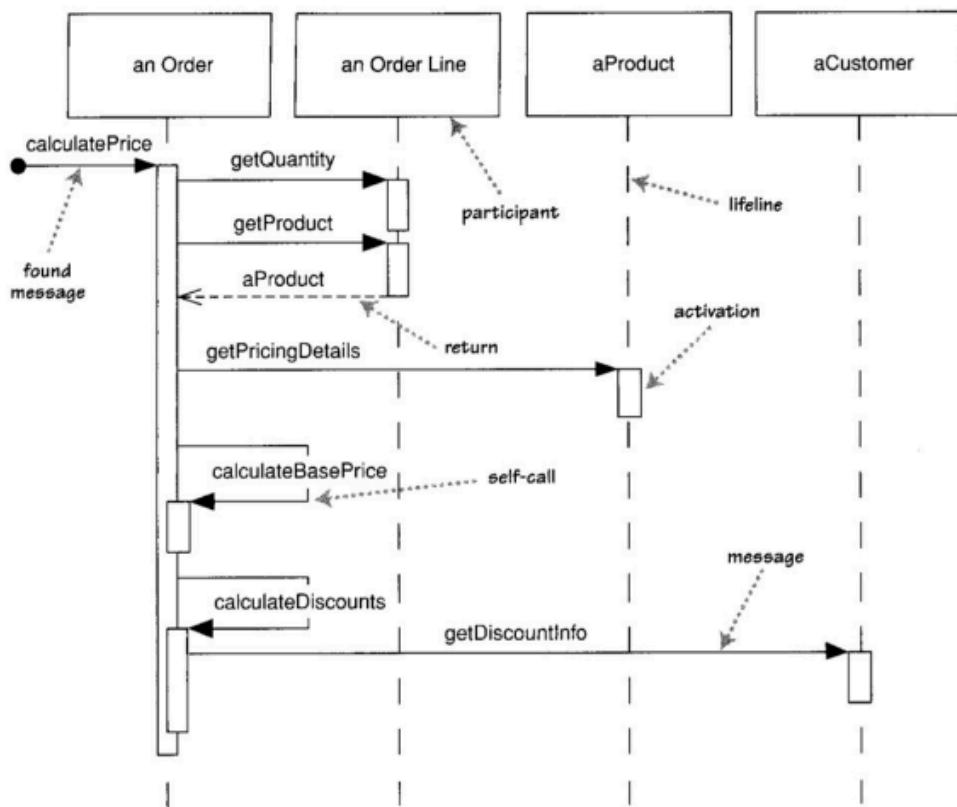
## Esempio



- Lo schema di numerazione annidato evidenzia chiamate e sotto-chiamate
- Più compatto di un diagramma di sequenza
- ... ma i diagrammi di sequenza sono molto più popolari

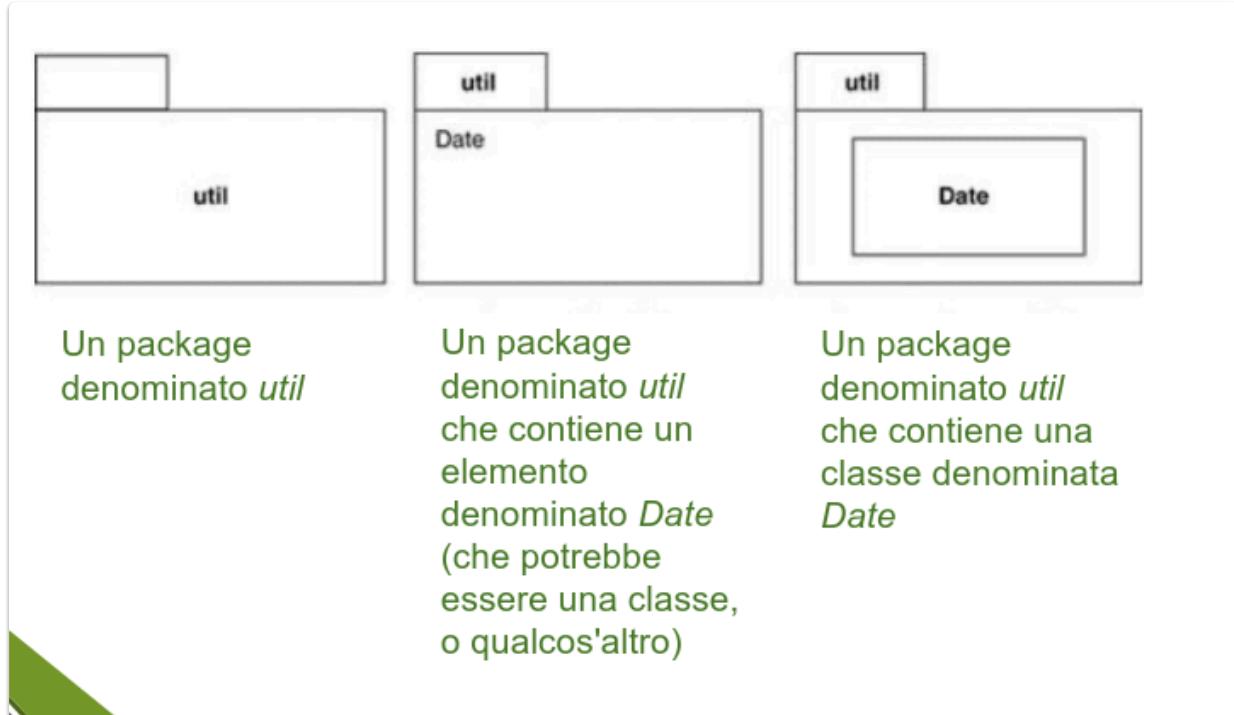
I diagrammi di **comunicazione** enfatizzano meglio la **successione di chiamate effettuate**, mentre quelli **interazione** si concentrano sui **messaggi scambiati**.

## Diagramma di sequenza equivalente



## 1.4 Diagramma dei package (Fowler)

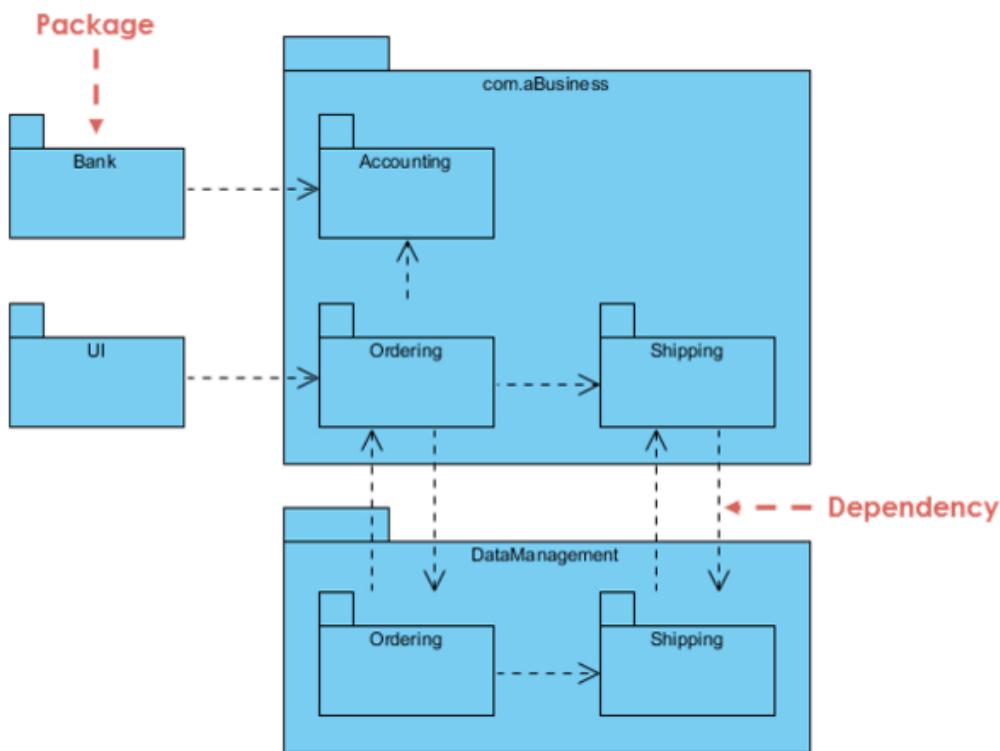
Un **package** è un raggruppamento di elementi (generalmente classi) in una singola unità di livello superiore. Il diagramma dei package è un **modello UML** che descrive la scomposizione in moduli (package) di un sistema e le varie dipendenze tra essi. È un modello molto utile nel progetto di sistemi di grande scala, per documentare le dipendenze tra i componenti più importanti.



I package possono essere eventualmente innestati l'uno nell'altro, andando a definire una gerarchia di package. In tal caso, rendere esplicita la classe a cui il package fa riferimento, si può utilizzare un **nome completamente qualificato** che mostra la struttura gerarchica dei package fino ad arrivare alla classe: "package1::package2::classe", dove package1 rappresenta il package più esterno ed astratto.

Le dipendenze vengono espresse in UML mediante una linea tratteggiata. I package che vengono utilizzati in troppi punti del sistema e per cui risulta impossibile disegnare le dipendenze, possono essere indicati come "**global**".

# Diagramma dei package: Esempio



## Diagramma dei package

È possibile indicare dipendenze tra i package

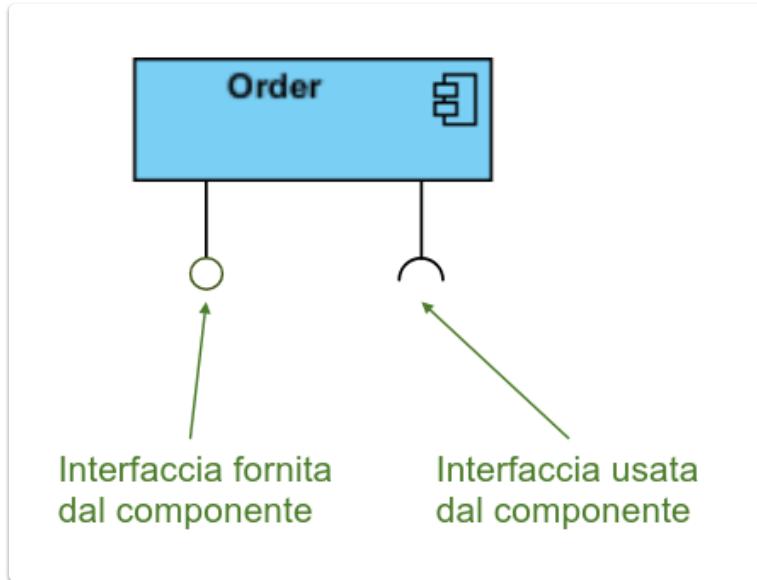


Il Package A dipende da Package B.  
Ad esempio, Package A usa una  
delle funzioni messe a disposizione  
da Package B.

## 1.5 Diagramma delle componenti

Si usa per indicare la **suddivisione** di un sistema in **componenti software**, specialmente nel caso in cui i componenti possono essere realizzati separatamente. Si può usare per **descrivere le interfacce** che ciascun

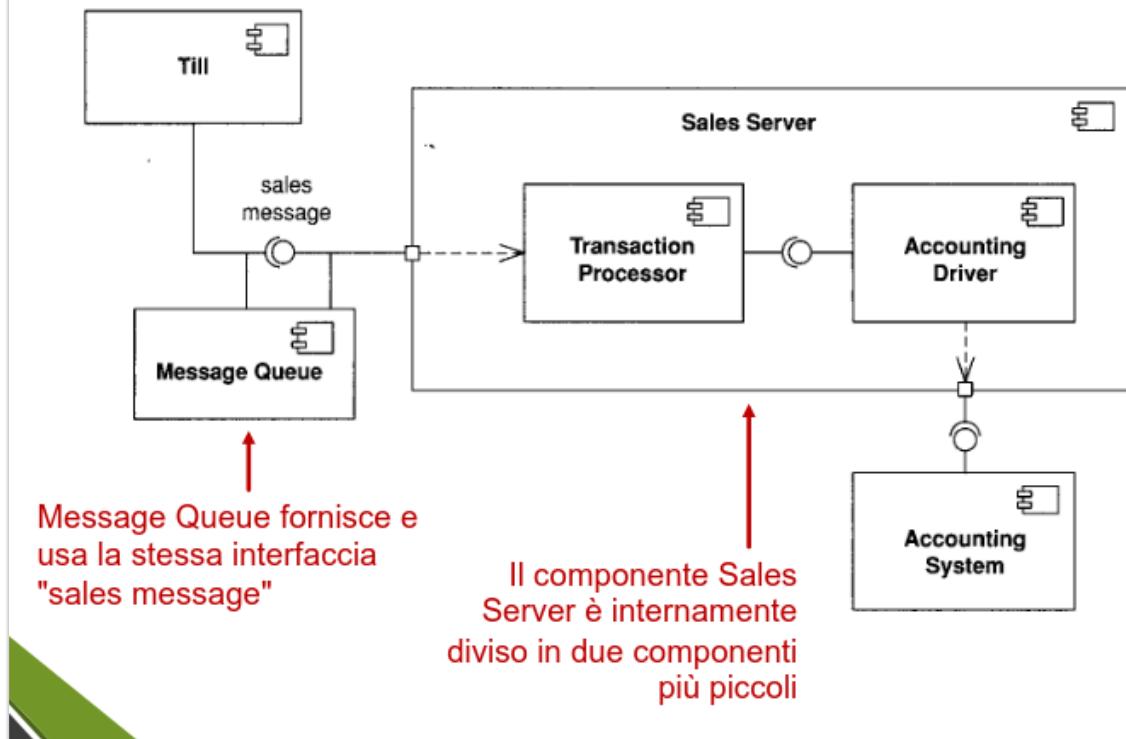
componente usa e mette a disposizione.



I componenti rappresentano **pezzi di software** che possono essere acquisiti e aggiornati in modo indipendente. I diagrammi dei componenti servono a **scomporre graficamente il sistema** in componenti, documentando le loro relazioni mediante le interfacce oppure per mostrare la **cd** dettagliata di ogni singolo componente.

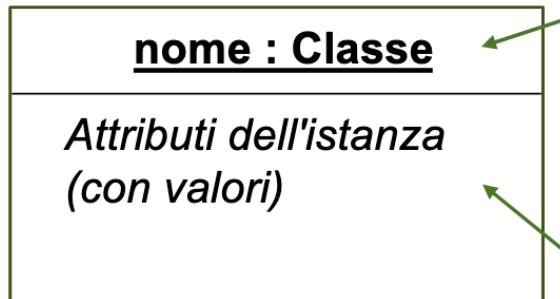
## Diagramma dei componenti

### Altro esempio



## 1.6 Diagrammi degli oggetti

Il diagramma di un **oggetto** può essere utilizzato per **mostrare lo stato** di un insieme di oggetti, in un determinato **momento dell'esecuzione**.

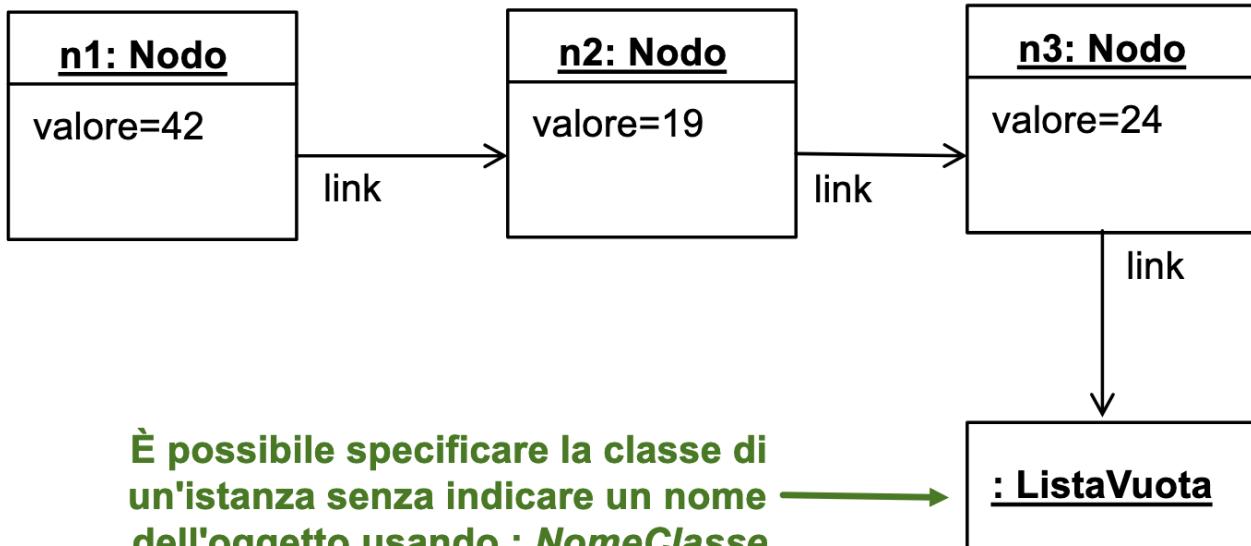


Nella rappresentazione di un oggetto il nome è sottolineato (a differenza della rappresentazione della classe)



Non sono indicati i metodi (sono condivisi da tutti gli oggetti della stessa classe)

Lo stato di alcuni oggetti potrebbe essere il seguente:



È possibile specificare la classe di un'istanza senza indicare un nome dell'oggetto usando : NomeClasse



## 1.7 Diagrammi delle attività

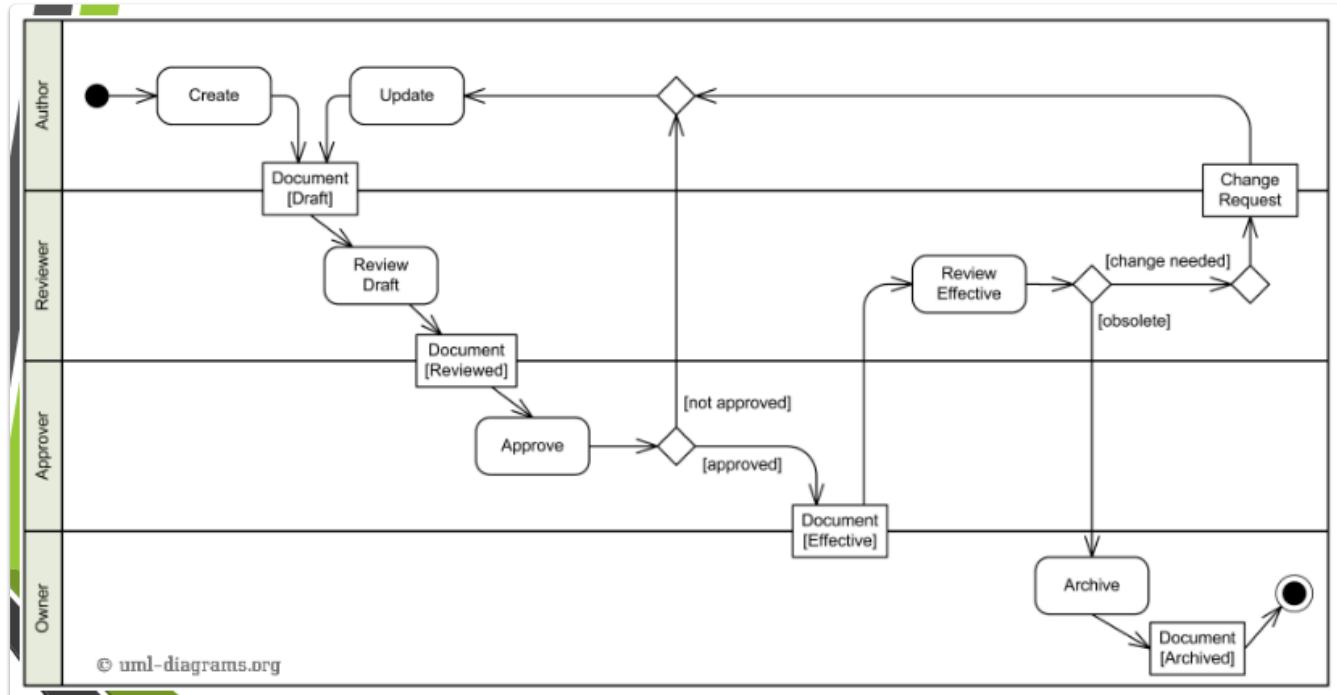
Visualizzano il flusso di azioni coinvolte in un processo, enfatizzando le sequenze e le condizioni di flusso. Servono a descrivere la logica procedurale, processi di business e workflow. Sono simili ai [flowchart](#) (infatti possono essere usati come loro sostituzione) con l'unica differenza che supportano la rappresentazione dell'[elaborazione parallela](#). Sono pertanto molto utili per descrivere [algoritmi paralleli](#).

# Diagrammi delle attività

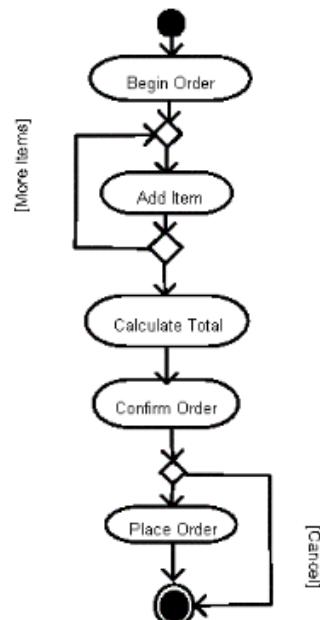
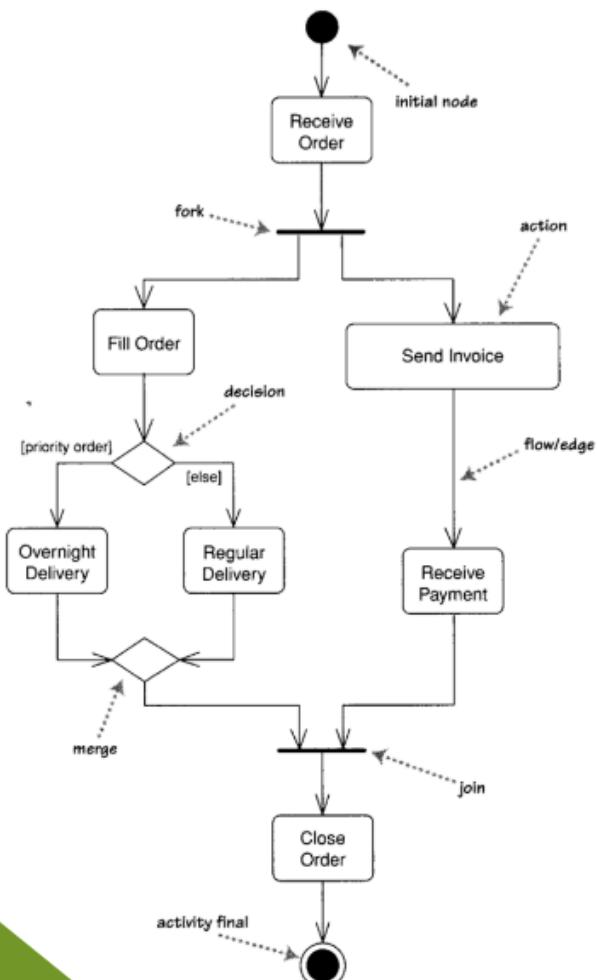
## Elementi principali:

- **Azione**: l'esecuzione di un comportamento (un calcolo aritmetico, una chiamata a un'operazione, ...)
- **Flusso (edge)**: collega le azioni in una sequenza
- **Decisione (branch)**: dirama il flusso di input testando una condizione (comportamento condizionale) - terminato da un **merge**
- **Fork**: trasforma un singolo flusso in ingresso in più flussi paralleli in uscita - terminato da un **join**.
- **Partizioni (swimlanes)**: Indicano quale oggetto è responsabile di quali azioni
- **Nodi iniziali e finali**

Un diagramma delle attività generalmente non indica quale oggetto/classe/componente svolge ciascuna attività del processo. È possibile aggiungere queste informazioni dividendo il diagramma in [partizioni](#):

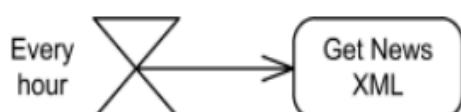
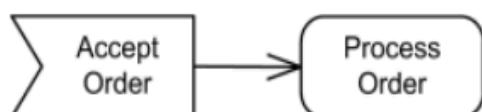
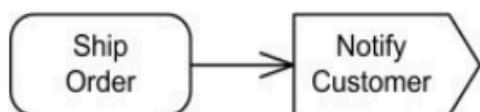


# Esempi di diagrammi delle attività



## Eventi

- **Inviare un segnale** a un processo esterno
- **Attendere un segnale** da un processo esterno
- **Segnale temporale** (si verifica a causa del trascorrere del tempo)

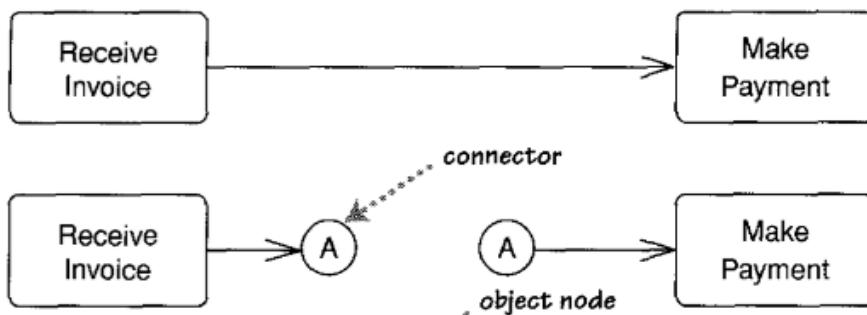


Se collegare due azioni del diagramma tramite un **arco** diventa troppo complesso o contribuisce solamente a creare confusione, è possibile effettuare il collegamento mediante dei **connettori** che presentano la stessa **etichetta**. Il connettore definisce un **salto**.

# Connettori

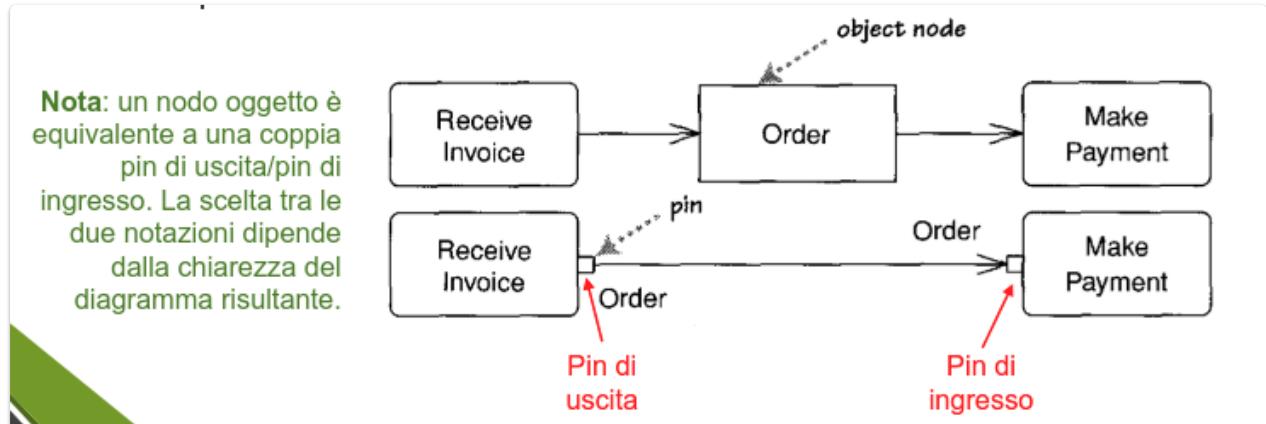
Rappresentano un salto nel diagramma

- Utilizzato quando il diagramma diventa troppo complesso



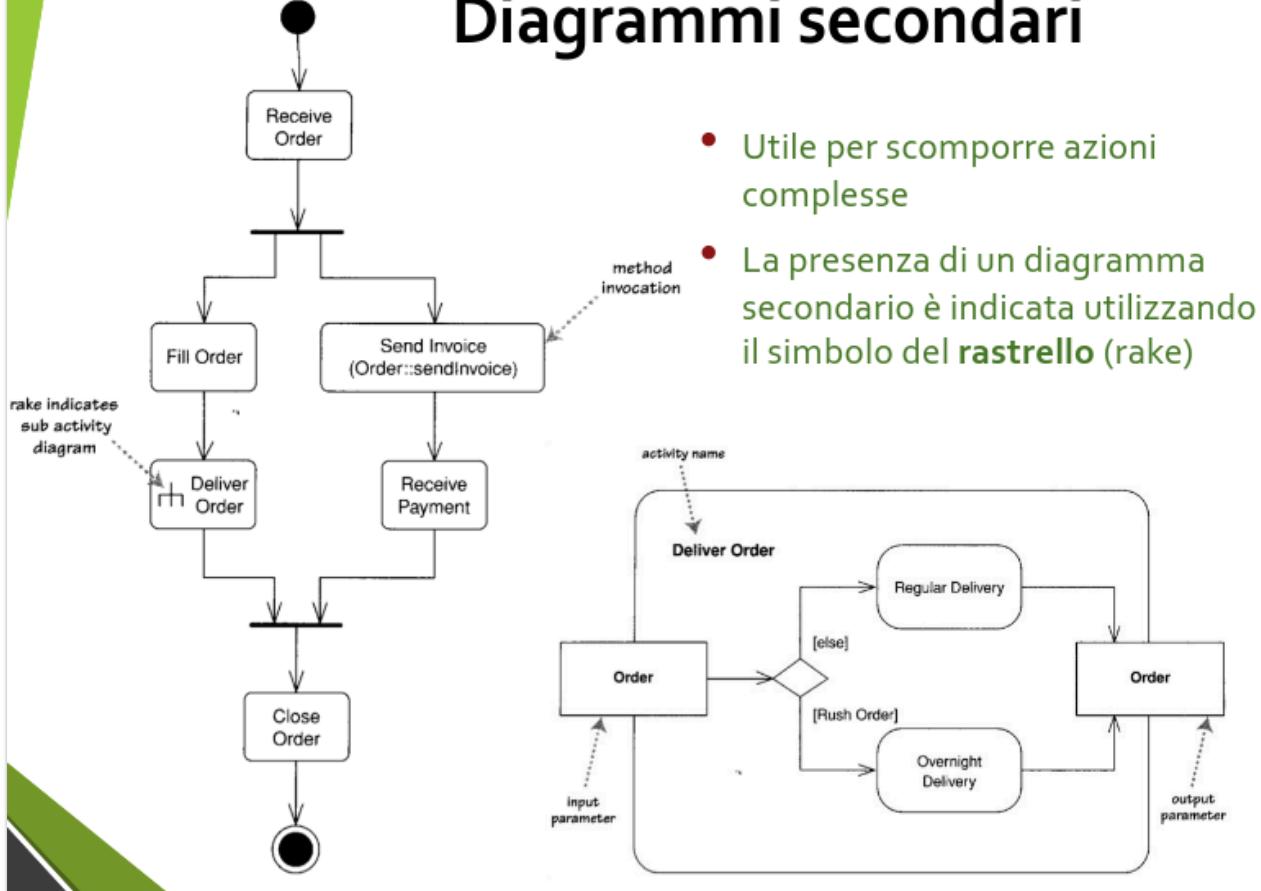
È possibile indicare le informazioni scambiate tra le azioni di una sequenza:

- **Nodo Oggetto:** rappresenta la classe di un oggetto scambiato tra le azioni;
- **Pin di uscita:** rappresentano le informazioni prodotte da una azione, che saranno usate da altre azioni;
- **Pin di ingresso:** rappresentano le informazioni usate da una azione, che saranno prodotte da altre azioni



Se il diagramma diventa troppo **complesso**, è possibile definire alcuni sotto-diagrammi detti **diagrammi secondari**.

# Diagrammi secondari



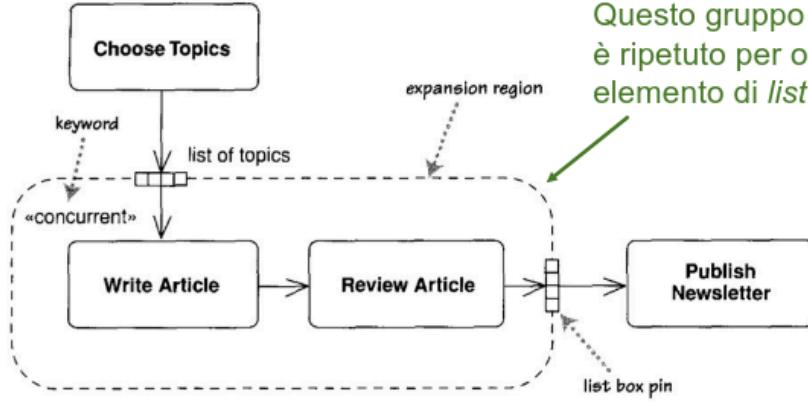
In questi diagrammi, l'output di un'azione potrebbe provocare l'invocazione multipla di un'altra azione. Le **regioni di espansione** contraddistinguono un'area del diagramma le cui azioni avvengono una sola volta per ogni elemento di una collezione.

## Regioni di espansione

Sono **nodi strutturati** che:

- prendono in input una *collezione* di oggetti (es. un array)
- agiscono su ogni elemento delle collezioni individualmente
- producono elementi che confluiscono in una collezione in output

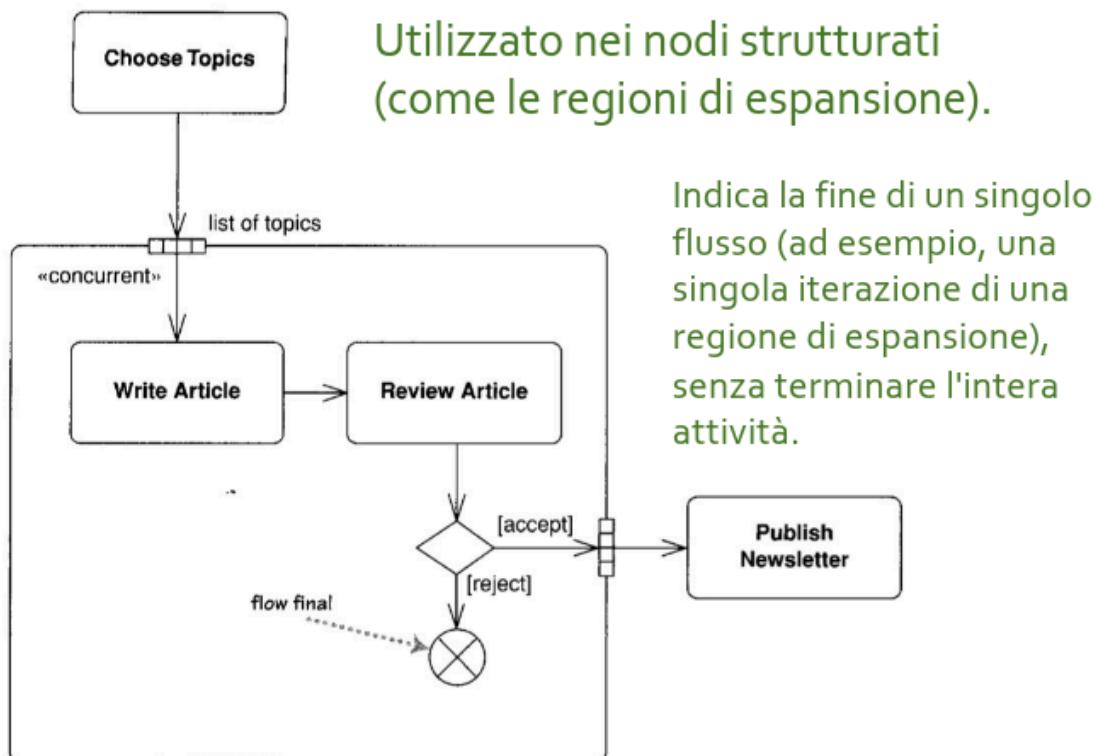
**Nota:** la keyword opzionale `<<concurrent>>` indica che l'elaborazione degli elementi della collezione è svolta in parallelo.



Le regioni di espansione possono funzionare anche come **filtri**. È possibile infatti far terminare un sotto-

flusso in particolare senza che termini l'intera attività. In tal caso si produce una collezione in uscita più piccola di quella in ingresso.

## Finale di flusso



Utilizzato nei nodi strutturati  
(come le regioni di espansione).

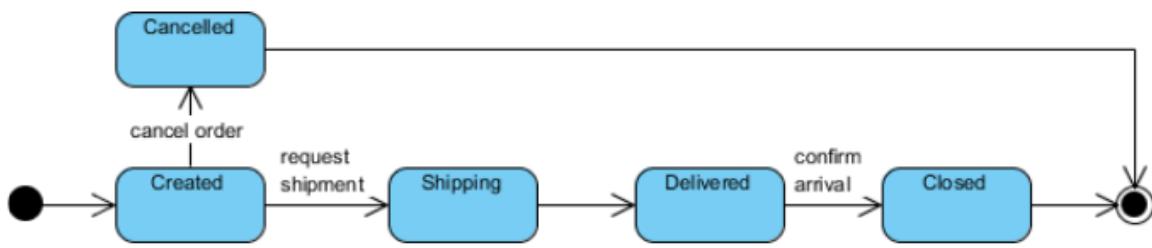
Indica la fine di un singolo  
flusso (ad esempio, una  
singola iterazione di una  
regione di espansione),  
senza terminare l'intera  
attività.

## 1.8 Diagrammi di macchina a stati

Mostrano i possibili **stati** di un oggetto e le **transizioni** che causano un cambiamento di stato. Sono basati sugli automi a stati finiti deterministici e possono essere usati per mostrare il comportamento di un oggetto durante il suo ciclo di vita.

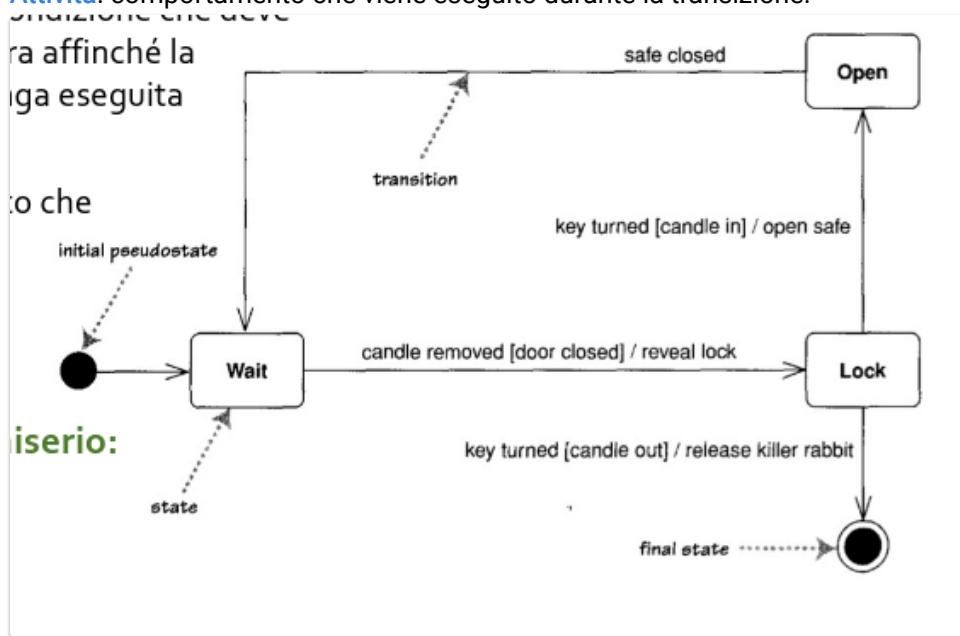
- Lo **stato** di un oggetto può essere visto come una combinazione dei valori delle sue proprietà
- Tuttavia, lo stato è una nozione più astratta: stati diversi implicano un modo diverso di reagire agli eventi.

**Esempio:** diagramma degli stati di un oggetto Ordine



Con transizione si intende il passaggio da uno stato all'altro ed è specificata da un'etichetta nella forma

- **Trigger:** evento che innesca il cambiamento di stato;
- **Guardia:** condizione che deve essere vera affinché la transizione venga eseguita;
- **Attività:** comportamento che viene eseguito durante la transizione.



Gli stati possono reagire ad alcuni eventi senza effettuare transizioni, usando le **attività interne**. Le attività interne sono paragonabili alle auto-transazioni: sostanzialmente si effettua una transizione che esce da uno stato e vi ritorna.

## Attività interne

Gli stati possono reagire ad alcuni eventi senza effettuare transizioni, utilizzando le **attività interne**.

- Trigger, guardia e attività sono indicati all'interno della casella di stato.
- Un'attività interna è *quasi* equivalente a un'auto-transizione (una transizione da uno stato verso se stesso)

### Attività speciali

- L'attività associata al trigger **entry** viene eseguita ogni volta che si entra nello stato; analogamente, l'attività associata al trigger **exit** viene eseguita quando si esce dallo stato.
- Le attività interne non innescano le attività di *entry* e *exit* (a differenza delle auto-transizioni esplicite)

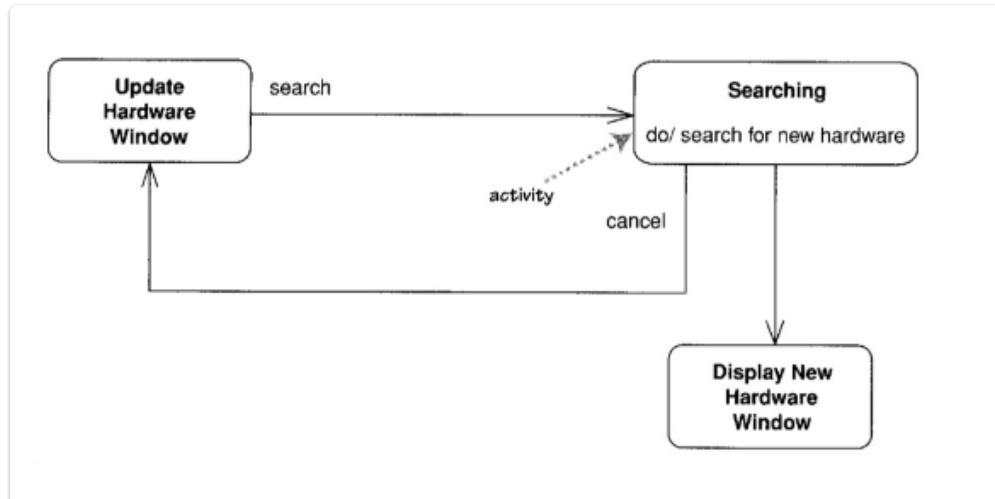
#### Typing

entry/highlight all
exit/ update field
character/ handle character
help {verbose}/ open help page
help [quiet]/ update status bar

Gli **stati di attività** sono gli stati in cui l'oggetto si ferma a svolgere un lavoro:

- il lavoro associato allo stato è contrassegnato dal trigger *do*/
- Una volta completato il lavoro, viene effettuata la **transizione senza trigger** che esce dallo stato Sostanzialmente rappresentano oggetti che svolgono un'attività in modo **continuato**. Quando termina l'attività, viene eseguita la transizione. La differenza con le **attività regolari** sta nel fatto che quest'ultime

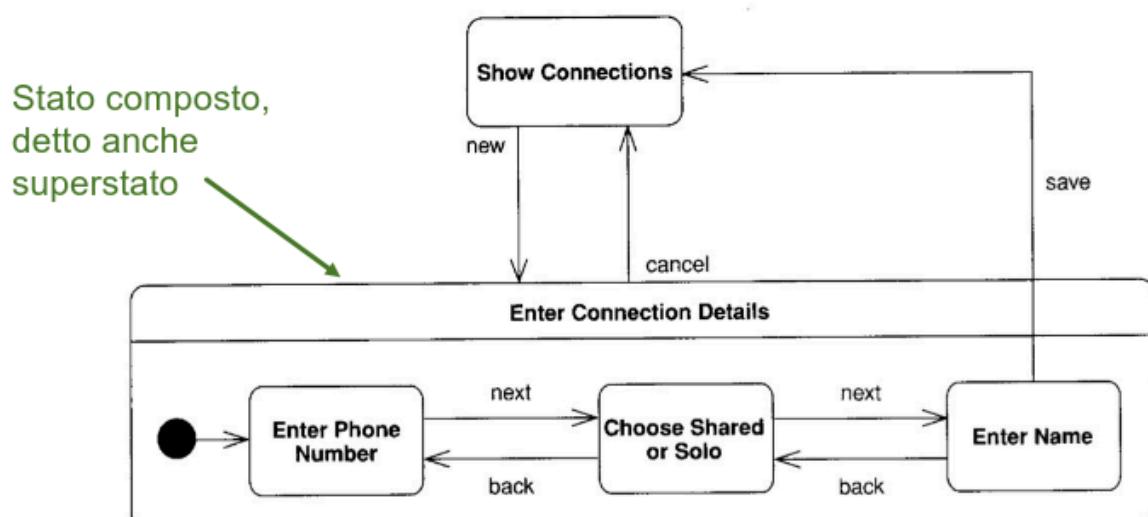
avvengono **istantaneamente** mentre le do-activity prendono **tempo** e possono anche essere **interrotte**.



Se più stati **condividono transizioni** ed **attività interne**, è possibile trasformarli in sottostati e raccogliere il comportamento comune in un **superstato**.

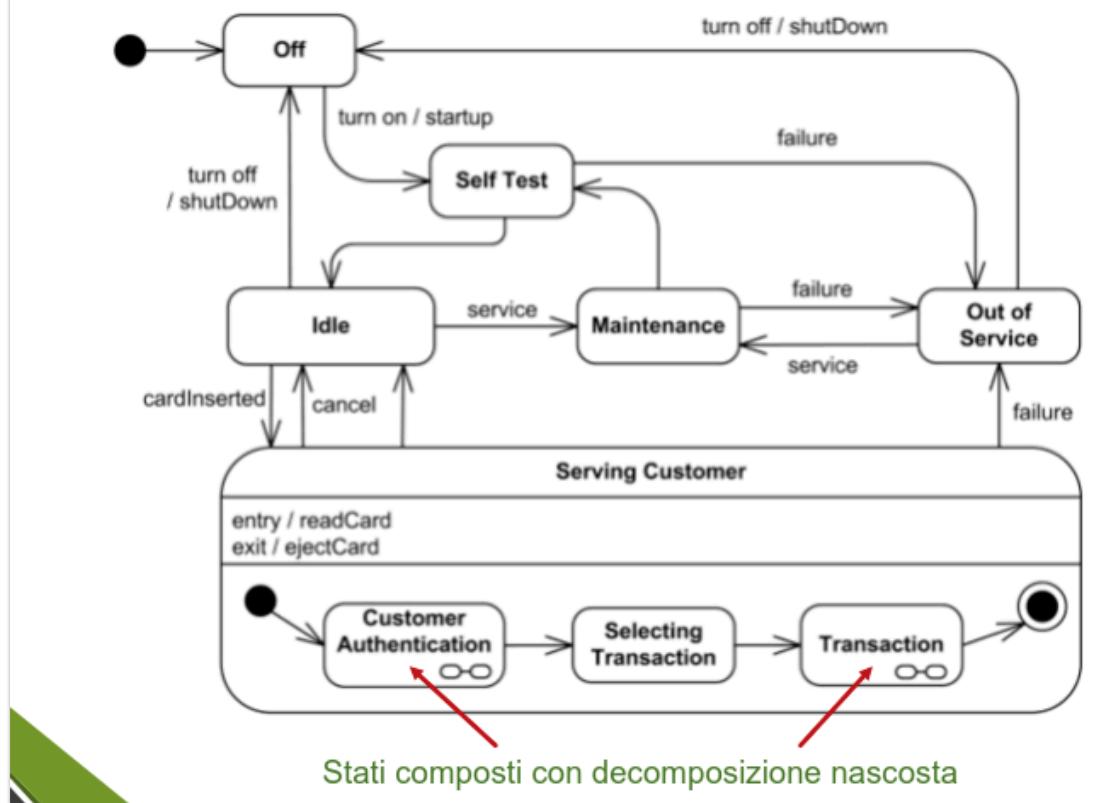
## Stati composti (superstati)

Raggruppano gli stati che condividono transizioni e/o attività interne comuni.



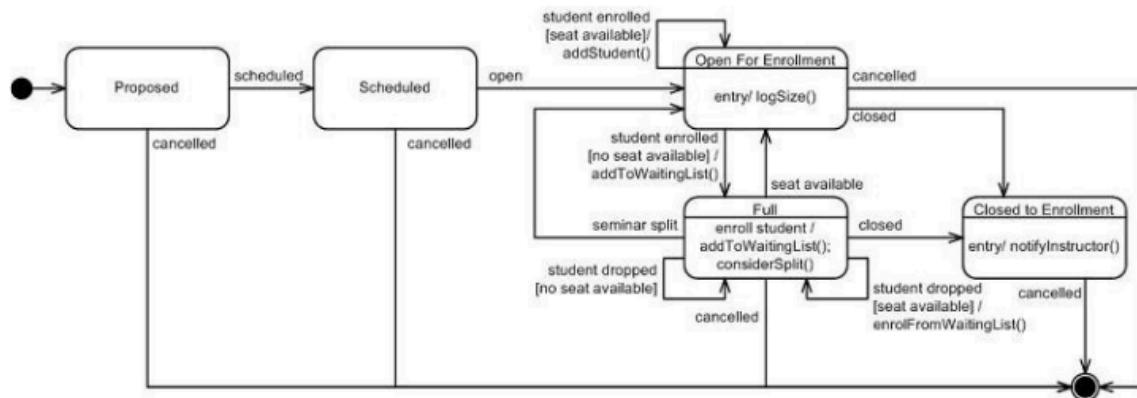
Da ciascuno dei sotto-stati dello stato composto si passa allo stato "Show Connections" quando si verifica l'evento "cancel".

# Esempio di Stati composti



## Esempio di diagramma

Il ciclo di vita di un seminario...



Le macchine a stati possono rappresentare solamente quello che un oggetto osserva o attiva direttamente.

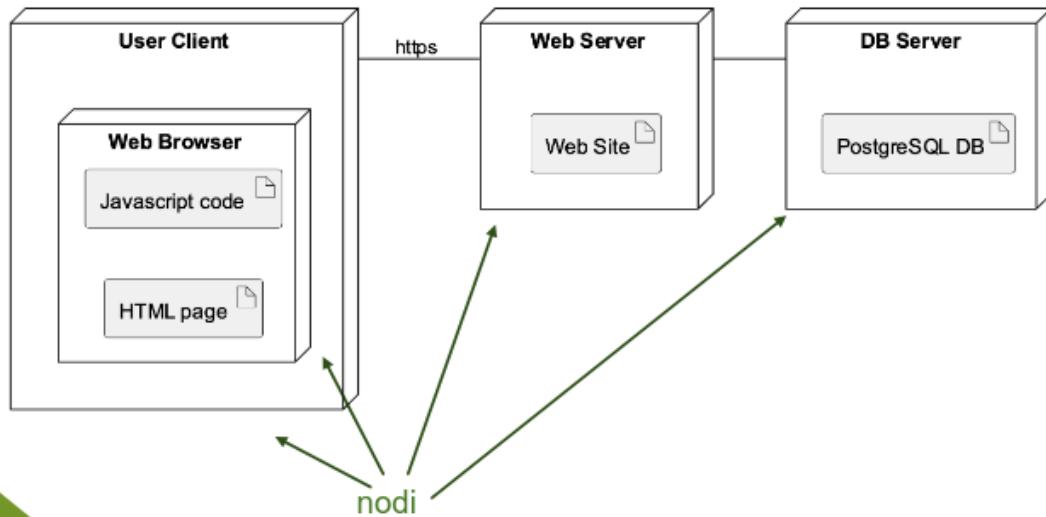
## 1.9 Diagrammi di distribuzione

Correlano l'architettura fisica dell'hardware con il sistema software implementato. Su quale hardware è in esecuzione la componente software?

# Diagrammi di distribuzione

I nodi contengono **artefatti**:

- **file eseguibili**
- **file non eseguibili**: dati, file di configurazione, pagine HTML, ...



## Esempio

