# UNIFIED MODELING LANGUAGE (UML)

## Introduzione

UML è uno standard aperto (estendibile) nato dalla collaborazione dei tres amigos: **Booch, Jacobson** e **Rumbaugh**. Viene accettato da varie figure del settore e dalle grandi compagnie dell'informatica. Dal 1997 diventa **standard dell'OMG**. Per UML esistono strumenti **CASE (Computer Aided Software Engeneering)**. Tramite questi strumenti è possibile generare lo scheletro del codice di un sistema; esistono vari strumenti CASE, sia a pagamento che free (quello che adottiamo è **LucidChart**). Anche Microsoft ha adottato UML come standard.

La notazione standard è basata su di un **metamodello** integrato degli oggetti che compongono un sistema software. Il linguaggio non descrive una sequenza di processi (prima fai questo, poi quello…), diventa così usabile da persone che utilizzano metodi diversi. Essendo un linguaggio standard, i suoi autori NON hanno copyright su UML. La sua evoluzione è a carico dell'OMG e viene soggetta a procedure ben definite per ogni cambiamento (versione attuale: UML 2.5 del 2013).

UML definisce i costrutti per diverse fasi dello sviluppo dei sistemi software:

- Analisi dei requisiti (casi d'uso);
- Analisi e progetto OO;
- Modellazione dei componenti;
- Modellazione della struttura e della configurazione.

Il modello **OOA/OOD (Object-Oriented Analisys/Design)** viene espresso tramite diagrammi **grafici**. Ogni entità può comparire in più diagrammi, rappresentandone una proiezione. AD ogni entità può essere associata una relativa documentazione testuale.

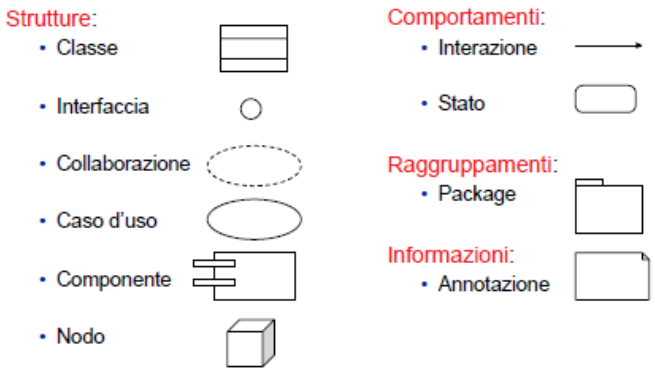
**Modello:** rappresentazione del dominio applicativo. Contiene elementi di informazione circa il sistema sotto osservazione;

**Diagramma:** visualizzazione del modello secondo certi criteri. Un elemento può apparire in più diagrammi, tuttavia la sua definizione all'interno del modello è univoca.

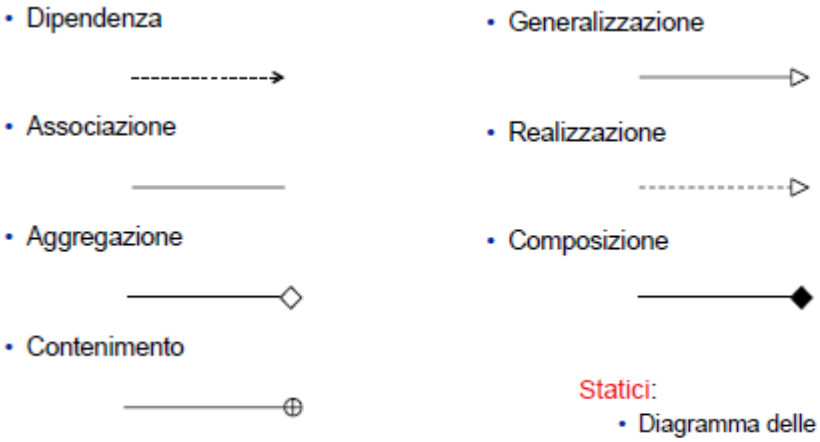
## Struttura di UML:

- Costituenti fondamentali: gli elementi di base.
    - **Entità;**
    - **Relazioni;**
    - **Diagrammi.**
- Meccanismi comuni: tecniche comuni per raggiungere obiettivi specifici.
    - **Specifiche;**
    - **Ornamenti;**
    - **Distinzioni comuni;**
    - **Meccanismi di estendibilità.**

- Architettura: espressione dell'architettura del sistema.




**Entità:** sono gli elementi di modellazione.



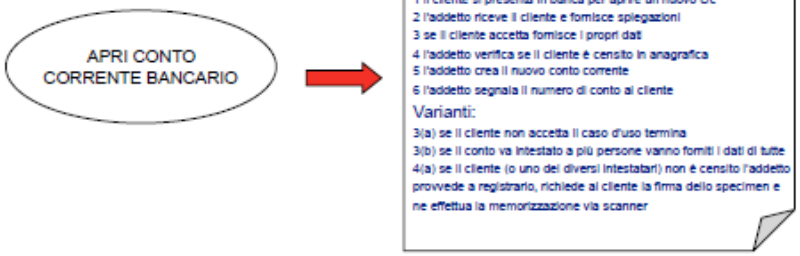

**Relazioni:** legano tra loro le entità.

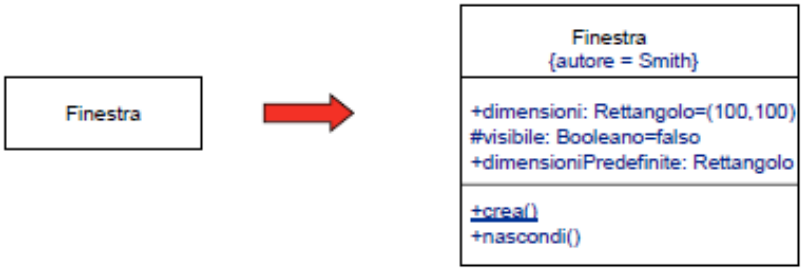

**Diagrammi:** sono le viste sul modello UML.

- **Diagrammi statici**
  - **Classi:** descrive la struttura dati degli oggetti del sistema e le loro relazioni. È il diagramma più importante, da cui si può generare il codice;
  - **Oggetti:** mostra un insieme di oggetti di interesse e le loro relazioni. Può essere vista come un'istanza del diagramma delle classi;
  - **Package:** mostra i package e le loro relazioni di indipendenza, contenimento e specializzazione;
  - **Componenti:** descrive l'architettura software i componenti del sistema (nonché le loro interfacce);
  - **Deployment:** descrive la struttura hardware del sistema e l'allocazione dei vari moduli software;
  - **Strutture composite** (non trattato): mostra la struttura interna di classificatori strutturati.
- **Diagrammi dinamici**
  - **Casi d'uso:** elenca i casi d'uso del sistema e le loro relazioni. È il primo diagramma da disegnare (e il più semplice): da questo diagramma si evidenziano le **funzionalità** del sistema software da realizzare;

- **Stati:** usa la notazione degli automi di Harel per descrivere gli stati degli oggetti di una classe.
- **Attività:** descrive le sequenze eventi-azioni-transizioni di una funzione. È il diagramma che modella il concetto di parallelismo.
- **Interazione:** mostra le interazioni tra gli oggetti durante scenari di funzionamento del sistema. Esistono 4 declinazioni diverse, ma l'unica che vedremo è quella di **sequenza**.

**Specifiche:** descrizione testuale della semantica di un elemento.

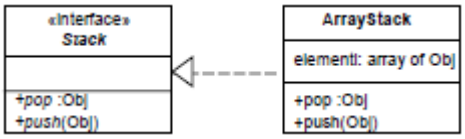



**Ornamenti:** rendono visibili gli aspetti particolari della specifica dell'elemento.

**Classificatore/istanza:** separa la nozione astratta di un'entità dalle sue concrete istanze. Un'istanza ha di solito la stessa forma del classificatore, ma con il nome sottolineato.


**Interfaccia/implementazione:** separa "cosa" un oggetto fa da "come" lo fa. Un'interfaccia definisce un contratto che ciascuna sua implementazione garantisce di rispettare.




## Meccanismi di estendibilità

**Stereotipo:** variazione di un elemento di modellazione esistente, con la stessa forma ma scopo diverso. Introduce quindi nuovi elementi di modellazione a partire da quelli esistenti.


**Proprietà:** valore associato a un elemento del modello, espresso da una stringa associata all'elemento *{ author = "Joe Smith", status = analisys }* *{ abstract }*

**Vincolo:** frase di testo che definisce una condizione o una regola che riguarda un elemento del modello e deve risultare sempre vera *{ disjoint, complete }* *{ subset }*

**Profilo:** insieme di stereotipi, valori etichettati e vincoli usati per personalizzare UML.

**Architettura:**

- **Vista dei casi d'uso:** descrive le funzionalità del sistema come vengono percepite dagli utenti, dagli analisti e dagli esecutori del testing. Non specifica l'organizzazione del software ma è la base per le altre viste.
- **Vista logica:** stabilisce la terminologia del dominio del problema sotto forma di classi e oggetti, illustrando come essi implementano il comportamento richiesto.
- **Vista dei processi:** variante orientata ai processi di vista logica; modella i thread e i processi sotto forma di classi attive;
- **Vista di implementazione:** descrive i moduli implementativi e le loro indipendenze, illustrandone la configurazione così da definire il concetto di versione del sistema;
- **Vista di deployment:** mostra la distribuzione fisica del sistema software sull'architettura hardware.

aspetti statici - aspetti dinamici

| | casi d'uso | logica | dei processi | implementativa | di deployment |
|---|---|---|---|---|---|
| casi d'uso | X | | | | |
| classi/oggetti | | X | X | | |
| componenti | | | | X | |
| distribuzione | | | | | X |
| stato | | X | X | X | X |
| attività | X | X | X | X | X |
| interazione | X | X | X | X | X |

Sistema complesso

| | casi d'uso | logica | dei processi | implementativa | di deployment |
|---|---|---|---|---|---|
| casi d'uso | X | | | | |
| classi/oggetti | | X | X | | |
| componenti | | | | X | |
| distribuzione | | | | | X |
| stato | | X | | | |
| attività | X | | | | |
| interazione | | X | X | | |

Sistema medio

| | casi d'uso | logica | dei processi | implementativa | di deployment |
|---|---|---|---|---|---|
| casi d'uso | X | | | | |
| classi/oggetti | | X | | | |
| componenti | | | | (X) | |
| distribuzione | | | | | (X) |
| stato | | (X) | | | |
| attività | | | | | |
| interazione | | X | | | |

Sistema piccolo

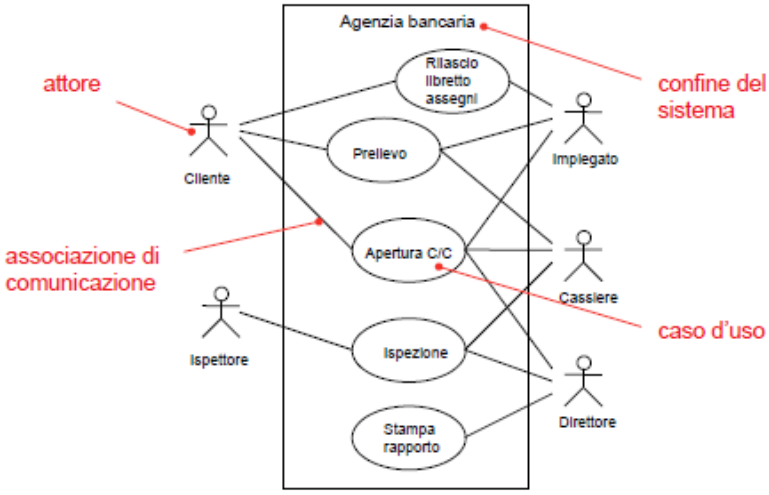
# DIAGRAMMA DEI CASI D'USO

Rappresentano i ruoli di utilizzo del sistema da parte di uno o più utilizzatori detti **attori**. Possono essere esseri umani, organizzazioni, enti, istituzioni o altre applicazioni o sistemi. Descrivono l'interazione tra attori e sistema, non la logica interna della funzione né la struttura del sistema. Sono espressi in forma testuale, comprensibile anche per i non "addetti ai lavori". Si possono definire a livelli diversi, ma sempre visti dal punto dell'utente.

**Attore:** ruolo che un'entità esterna assume quando interagisce direttamente con il sistema. È **SEMPRE** esterno al sistema (ma può mantenerne una rappresentazione interna). Spedisce o riceve messaggi dal sistema, o scambia informazioni con esso. Esegue i casi d'uso. È modellato con una classe e NON con un oggetto.

**Caso d'uso:** sequenza di azioni che un sistema, sottosistema o una classe può eseguire interagendo con attori esterni. Un attore percepisce un caso d'uso come una funzionalità. Il

suo scopo è produrre un risultato osservabile dall'attore. È sempre attivato da un attore ed è completo.







## Punti di vista

**UTILIZZATORE**

- Casi d'uso
  - telefonare
  - ricevere telefonate
  - inviare messaggi
  - memorizzare un numero
  - ....

**PROGETTISTA**

- Funzionalità interne
  - trasmissione / ricezione
  - alimentazione (batteria)
  - I/ O (display, tasti, ...)
  - gestione rubrica
  - .....

## Ruolo dei casi d'uso

Nelle fassi iniziali della progettazione servono per chiarire **cosa** dovrà fare il sistema. Il modo più efficcace ed efficiente per scoprire ed analizzare i requisiti ai quali il sistema dovrà fornire un'implementazione è ragionare sui casi d'uso con il committente. Dialogare su come il sistema verrà realizzato, nella comunicazione con persone non esperte nella progettazione, è più semplice di non guardare a come dovrà essere costruito. Raggiungere un accordo con il committente circa le modalità di utilizzo del sistema consente al progettista di affrontare con maggiore tranquillità il suo mestiere di progettazione.

**I casi d'uso riguardano l'intero progetto di sviluppo;** costituiscono il punto di partenza per la progettazione del sistema. Sono il riferimento primario per la definizione, progettazione, esecuzione dei test per la verifica di quanto prodotto. Rappresentano delle naturali unità di rilascio, per i progetti che seguono un approccio incrementale alla pianificazione della realizzazione e dei rilasci.

## Come identificare i casi d'uso?

1. Individuare i confini del sistema;
2. Identificare tutte le tipologie di utilizzatori del sistema che verranno modellati come **attori**;

3. Per ogni attore, rivelare in quale modo utilizzerà il sistema, partendo dagli obiettivi che egli deve raggiungere. A ogni modalità di utilizzo corrisponde un **caso d'uso**;
4. Per ciascun caso d'uso, descrivere lo scenario base, ovvero la sequenza di passi più semplice possibile che conduce al successo del caso d'uso e le risposte del sistema, e le principali varianti dello scenario. Così facendo possono emergere necessità di interazione del sistema con altri soggetti che verranno rappresentati nel modello come attori aggiuntivi.

**Scenario:** esecuzione di un caso d'uso. Possono essere di **successo** o di **fallimento**. Gli scenari possibili sono innumerevoli, la prassi è quella di definire uno **scenario base** e agganciarne le **varianti**, che lo rendono più complesso e possono portare al succeso o al fallimento del caso d'uso.

Caso d'uso: "APRI CONTO CORRENTE BANCARIO"

Scenario base:

1 il cliente si presenta in banca per aprire un nuovo c/c
2 l'addetto riceve il cliente e fornisce spiegazioni
3 se il cliente accetta fornisce i propri dati
4 l'addetto verifica se il cliente è censito in anagrafica
5 l'addetto crea il nuovo conto corrente
6 l'addetto segnala il numero di conto al cliente

Varianti:

3(a) se il cliente non accetta il caso d'uso termina
3(b) se il conto va intestato a più persone vanno forniti i dati di tutte
4(a) se il cliente (o uno dei diversi intestatari) non è censito l'addetto provvede a registrarlo, richiede al cliente la firma dello specimen e ne effettua la memorizzazione via scanner

## Specifiche del caso d'uso

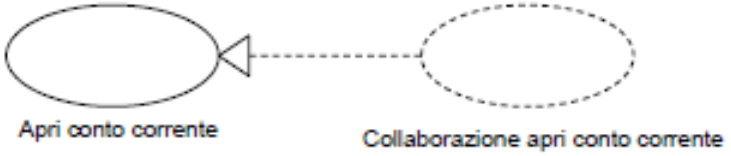
UML non suggerisce il modo per specificare un casp d'uso, lasciando libero spazio a tutte le possibili forme di documentazione testuale. La specifica di un caso d'uso ha un ruolo centrale nella comunicazione tra i diversi soggetti coinvolti nello sviluppo di un sistema, dal committente agli utilizzatori, dai progettisti agli specialitsti di test. Un caso d'uso può anche essere descritto da un diagramma **di attività** o **di sequenza**.

| Nome | |
|---|---|
| Identificatore | |
| Breve descrizione | *fissa l'obiettivo del caso d'uso* |
| Attori primari | *avviano il caso d'uso* |
| Attori secondari | *interagiscono con il caso d'uso dopo che è stato avviato* |
| Precondizioni | *condizioni che devono essere vere prima che il caso d'uso possa essere eseguito* |
| Sequenza principale degli eventi | *i passi che costituiscono il caso d'uso* |
| Postcondizioni | *condizioni che devono essere vere quando il caso d'uso termina* |
| Sequenze alternative degli eventi | *un elenco di alternative alla sequenza principale* |

## Come realizzare i casi d'uso?

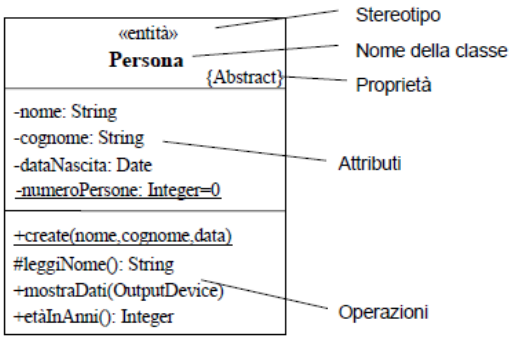
La realizzazione può essere espressa come una **collaborazione** costituita da classi che interagendo tra di loro svolgono i passi specificati nel caso d'uso. La collaborazione che realizza un caso d'uso può essere descritta:

- A livello **statico** mediante un diagramma delle classi che evidenzi le classi o gli oggetti coinvolti nella collaborazione;
- A livello **dinamico** mediante un diagramma di interazione che evidenzi i messaggi che gli oggetti si scambiano nell'ambito della collaborazione.



# DIAGRAMMA DELLE CLASSI

È il diagramma principale di tipo statico di UML, da cui si può ricavare il codice. Descrive la struttura del sistema in termini di classi e le loro relazioni reciproche. È il diagramma che ha la maggior varietà di costrutti.




**Classe:** gruppo di oggetti con proprietà, comportamento e relazioni comuni. I suoi componenti sono attributi ed operazioni. Può trasformare lo stato di un oggetto.

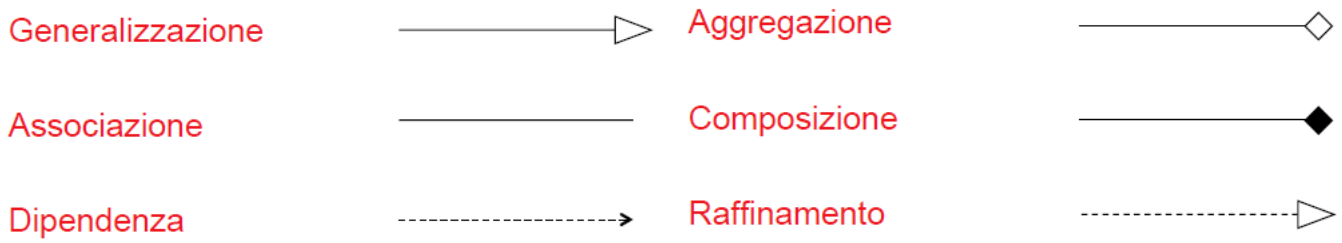
**Attributo:** valore che caratterizza gli oggetti di una classe.

**Operazione:** trasformazione che può essere applicata (o invocata) agli oggetti di una determinata classe. Ogni operazione ottiene come argomento l'oggetto di destinazione.

## Notazione

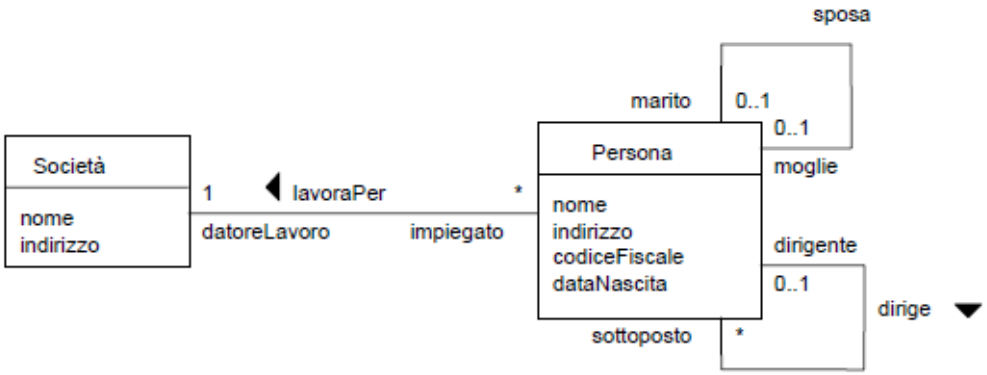
- **Attributi:** *visibilità* nome *molteplicità* : *tipo* = *valoreDefault*
  - **Visibilità** (pubblica +, privata -, protetta #, package ~)
  - **Molteplicità** (String[5], Real[2..*], Boolean[0..1])
  - **Tipo** (Integer, Boolean, String)
  - **Ambito** (Istanza, classe)
- **Operazioni:** *visibilità* nome (*parametro, …*): *tipoRestituito*
  - **Parametri** *direzione nomeParametro: tipoParametro=valoreDefault*
  - **Direzione** (in, out, inout, return)
  - **Ambito** (Istanza, classe)

## Relazioni tra classi




**Associazioni:** composizione di classi (bidirezionali). DEVE avere un nome (eccetto aggregazioni/composizioni). DEVE avere le molteplicità. Si può indicare il verso di lettura tramite una freccia, definire associazioni monodirezionali oppure specificare i ruoli.

*Molteplicità di associazioni:* Esattamente 1 (1) Opzionale 1 (0..1) Da x a y inclusi (x..y) Solo i valori a,b,c (a,b,c) 1 o più (1..*) 0 o più (*)

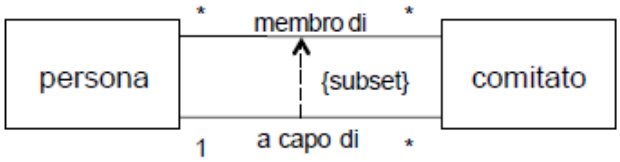


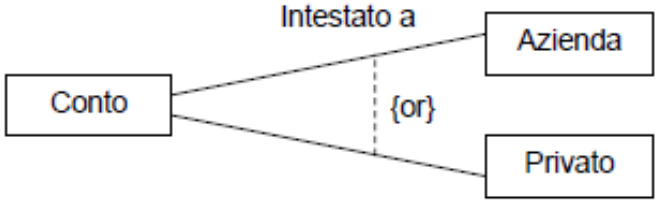

Questo esempio contiene versi di lettura (su "lavoraPer" e "dirige"), associazioni monodirezionali e ruoli (es. "marito-moglie").

## Vincoli e classi associative

{or} è equivalente a "is a", {subset} è equivalente a "essere parte di"

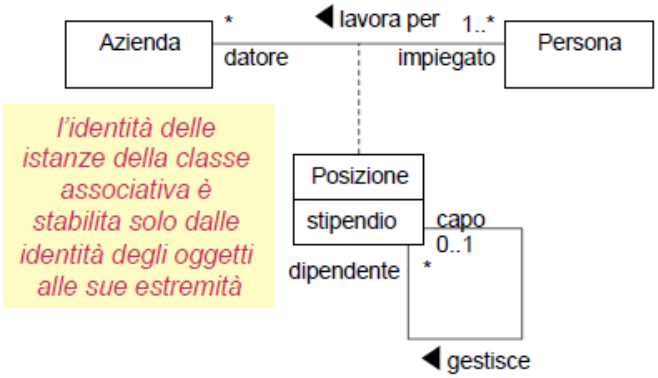
(b,c) ---> (a,b,c,d,e)  (b,c) è un **subset**.

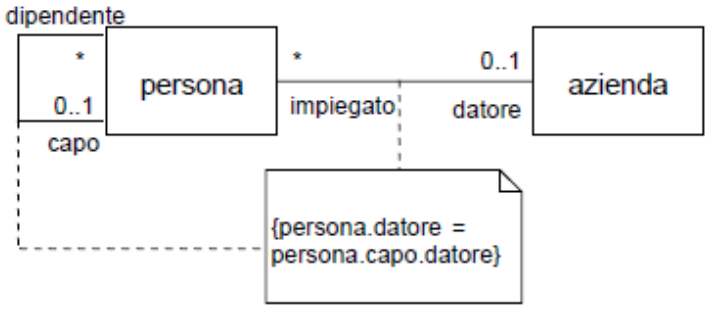



Il conto corrente è intestato ad un'Azienda oppure a un Privato, ma non ad entrambi.

**Classe associativa:** serve per mettere un attributo su un'associazione. È una classe a tutti gli effetti. Nelle classi associative la combinazione di due istanze può comparire più volte.

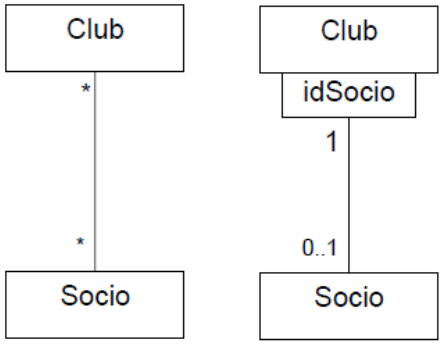



**Post-it:** può essere usato per specificare ulteriori vincoli, non definibili con altri costrutti.

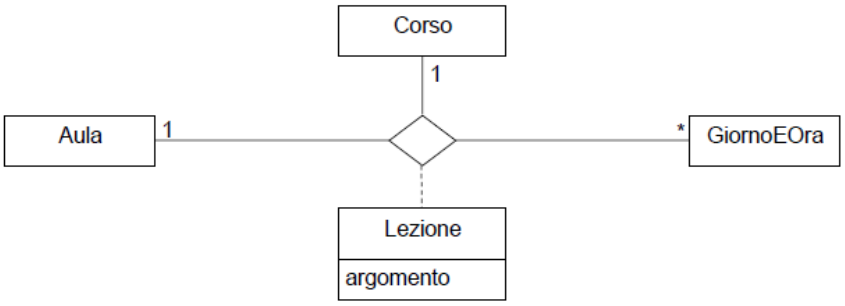
## Associazioni qualificate

Riducono da N-N a 1-1, specificando un attributo per identificare un'istanza e svolge il ruolo di **chiave esterna**.



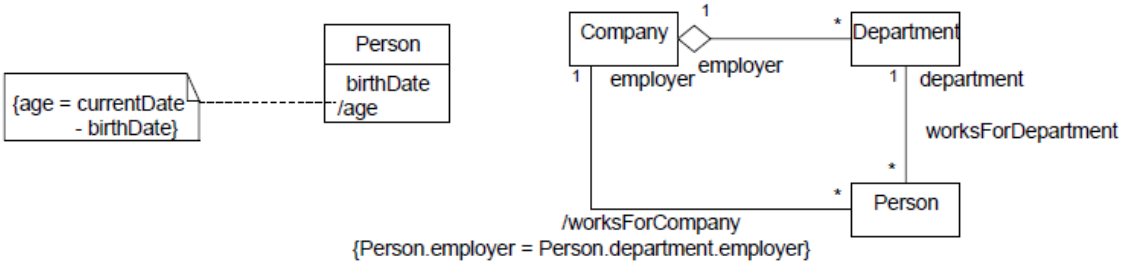
## Associazioni N-arie

Coinvolgono N-classi e partecipano rispettivamente N-oggetti. La molteplicità di un ruolo rappresenta il numero di istanze dell'associazione che ha fissi N-1 oggetti. Tipicamente quello che si fa quando ci si trova di fronte a questa situazione è di **reificare** eliminando il rombo ed aggiungendo un'entità, collegando quindi quest'ultima con delle associazioni binarie).



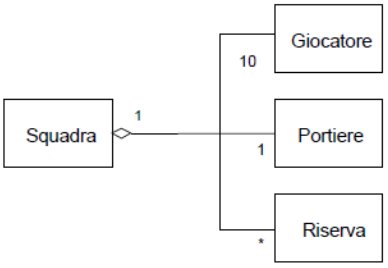

## Elementi derivati

Sono elementi che possono essere calcolati a partire da un altro, ma viene mostrato per chiarire il diagramma o per scelte di progettazione, ma di fatto non aggiunge alcuna informazione semantica. Viene indicato posizionando uno slash (/) prima del suo nome. Per informare come viene calcolato si può inserire una nota o una stringa di vincoli.




**Aggregazione (part of):** sia il tutto che le parti esistono indipendentemente.

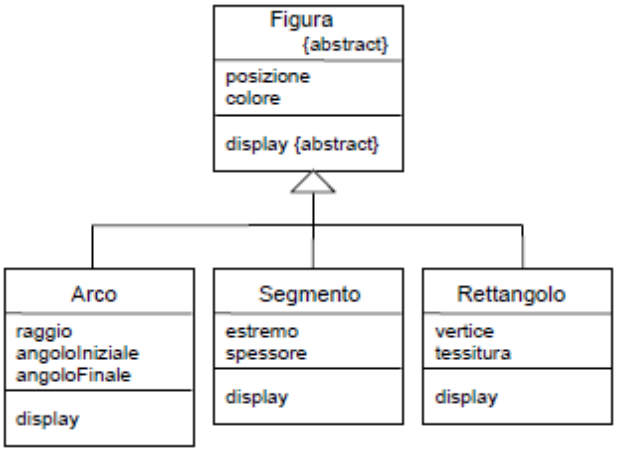
**Composizione:** il tutto possiede le sue parti; significa che le parti esistono solo in relazione al tutto. In altre parole, eliminando il "tutto" si perdono anche le sue "parti".





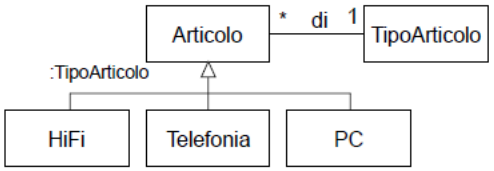
**Generalizzazione/specializzazione:** tutti gli attributi, le operazioni e le relazioni della superclasse vengono ereditati dalle sottoclassi.

L'eredità multipla è supportata e possono essere indicati insiemi di generalizzazione e vincoli (overlapping, disjoint, complete, incomplete)

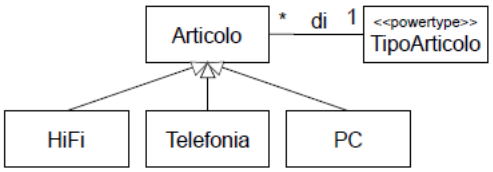


**Classi astratte:** classi che non possono essere istanziate da oggetti perché **parziali**. Sono particolarmente utili per specificare radici di gerarchie di specializzazione e si denotano con {abstract}.

**Powertyping:** meta-classe le cui istanze sono classi che specializzano un'altra classe.




in UML 2

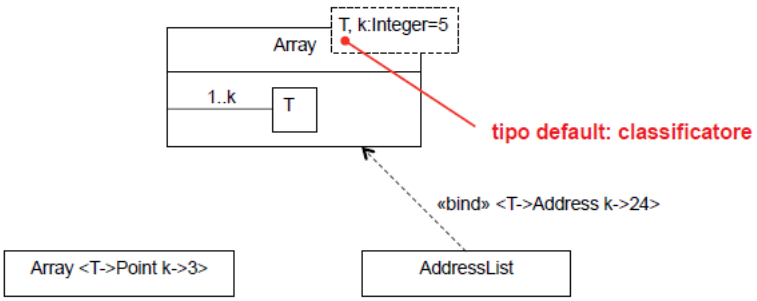



in UML 1.4

---

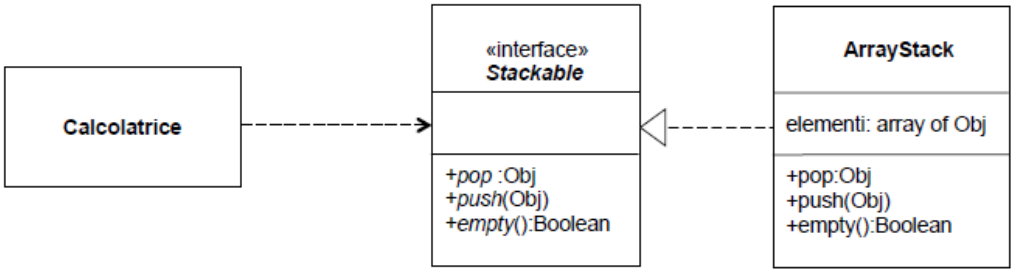
**Dipendenza:** se A dipende da B, significa che una variazione in B può comportare una variazione in A. Nel caso delle classi, una classe cliente dipende da alcuni servizi di una classe fornitore, ma non ha una struttura interna che dipende da quest'ultima. Come stereotipo si usa <<use>>. Si può rappresentare il fatto che un'operazione della classe cliente ha argomenti che appartengono al tipo di un'altra classe con <<parameter>>

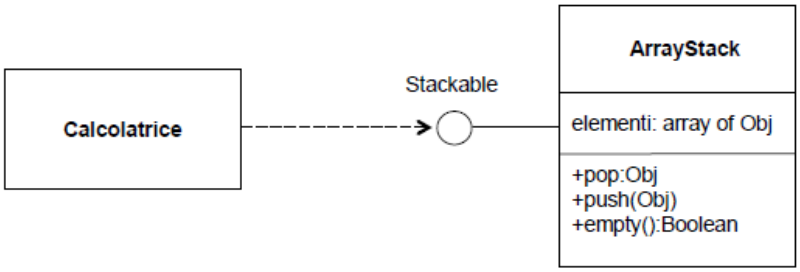
**Template o classe parametrizzata:** descrive una classe in cui uno o più parametri formali non sono istanziati. Quello che fa un template è definire una famiglia di classi in cui ognuna è specificata istanziando i parametri con i valori attuali. Un template non è direttamente usabile. **Bound element:** classe che istanzia i parametri di un template, usabile esattamente come una classe.




**Raffinamento:** esprimere una relazione tra due descrizioni dello stesso concetto a diversi livelli di astrazione: tra un tipo astratto e una classe che lo realizza, tra una classe di analisi e una di progetto, tra una implementazione semplice e una complessa della stessa cosa.

**Interfaccia:** insieme di funzionalità pubbliche identificate da un **nome**. Specifica le operazioni pubbliche di una classe, componente, pacchetto o altre entità, separando le specifiche dall'implementazione. Non ha alcuna specifica di struttura interna (attributi, stato, associazioni); è di fatto una classe astratta! (con operazioni solo astratte). Hanno <<interface>> come stereotipo e sono senza compartimento per gli attributi; facendo uso della notazione minimale si inserisce un cerchio collegato all'entità che la supporta con accanto il nome dell'interfaccia (**lollypop notation**). Se una classe vuole usare un'interfaccia, si può collegare ad essa tramite una dipendenza e con lo stereotipo <<use>>.




## Analisi o progettazione?

**Classi di analisi:**

- Rappresentano un'astrazione nel dominio del problema;
- Corrispondono chiaramente a concetti concreti nel mondo del business;
- Escludono tutti i dettagli implementativi;
- Hanno un insieme ridotto, coeso e ben definito di responsabilità;
- Indicano gli attributi che saranno probabilmente inclusi nelle classi di progettazione;
- Le loro operazioni specificano i principali servizi offerti dalla classe.

**Classi di progettazione:**

- Le loro specifiche sono complete per cui possono essere direttamente implementate;
- Nascono dal dominio del problema per raffinamento delle classi di analisi, oppure dal dominio della soluzione.

## Come identificare le classi di analisi?

Classi = **identità fisiche**, **concetti** del dominio applicativo

NON rappresentare soluzioni implementative.

NON ridondare classi, inserirne di vaghe, irrilevanti, "onnipotenti".

Associare ad ogni classe un insieme definito di responsabilità, che comunque sono poche (3-5).

NON vanno isolate classi => costituirebbero un UML a parte!

NON vanno bene soluzioni estreme (tante classi troppo semplici, poche classi troppo complesse).

I nomi delle classi devono rispecchiare la natura intrinseca del problema => NON denominare una classe in base alle sue associazioni!

NON usare gerarchie di specializzazione profonde.

Se si vuole descrivere un oggetto attraverso un nome, usare un attributo!

Se una proprietà esiste indipendentemente dalla classe cui fa riferimento (oppure esiste più volte all'interno del diagramma) è preferibile rappresentarla come **classe**.

Per indicare associazioni si usano tipicamente **verbi** che possono esprimere:

- **Collocazione fisica** *(contenuto in)***;**
- **Azioni** *(gestisce)***;**
- **Comunicazioni** *(parla a)***;**
- **Proprietà** *(possiede)***;**
- **Soddisfacimento di condizioni** *(sposato a)***.**

Un riferimento da una classe a un'altra è un'**associazione**.

Un'aggregazione è un'associazione con semantica **part-of**.

NON usare associazioni che esprimono soluzioni implementative.

Le associazioni devono descrivere proprietà strutturali del dominio, NON eventi transitori!

Scomporre le associazioni ternarie in due binarie è preferibile.

Si possono evidenziare le associazioni derivate (ovvero quelle che possono essere espresse in termini di altre associazioni).

I ruoli all'interno delle associazioni possono essere indicati a propria discrezione.

## Come identificare gli attributi?

**Attributi** sono anche proprietà di classi e associazioni. Spesso corrispondono nomi seguiti da possessivi *(il colore della macchina)*.

È sufficiente omettere (o evidenziare) gli attributi derivati.

Se una proprietà dipende dalla presenza di un'associazione, rappresentarla come un attributo di un'associazione => usare una classe associativa!

NON aggiungere identificatori degli oggetti (se non espressamente richiesto).

Se gli attributi di una classe sono raggruppabili in due sottoinsiemi, probabilmente la soluzione migliore è scompattare in due classi.

## Raffinamenti

È una caratteristica tipica del modello OO.

I raffinamenti in ereditarietà possono essere **top-down** (si specializzano le classi esistenti) o **bottom-up** (si generalizzano almeno due classi con caratteristiche comuni).

In caso di asimmetrie in associazioni/generalizzazioni, aggiungere classi può essere una buona soluzione.

In caso di difficoltà nel generalizzare, spezzare una classe in due classi più piccole è un buon approccio.

Se esistono due associazioni con lo stesso nome/scopo, meglio creare una superclasse che le generalizza.

Se un ruolo incide nella semantica di una classe, forse è meglio se viene trasformato in una nuova classe.

Una classe che non ha attributi, operazioni o associazioni può essere eliminata.

Se nessuna operazione usa un'associazione, probabilmente è inutile.

## Come identificare le classi di progettazione?

Queste classi specificano come le classi assolveranno le loro responsabilità.

Ogni classe deve essere:

- **Completa**: fornire ai clienti tutti i servizi che si aspettano;
- **Sufficiente**: i metodi devono essere finalizzati allo scopo della classe;
- **Essenziale**: mettere a disposizione un unico modo per effettuare un'operazione data;
- **Massimamente coesa**: modellare un unico concetto astratto;
- **Minimamente interdipendente**: essere associata all'insieme minimo di classi che le consente di realizzare le proprie responsabilità.

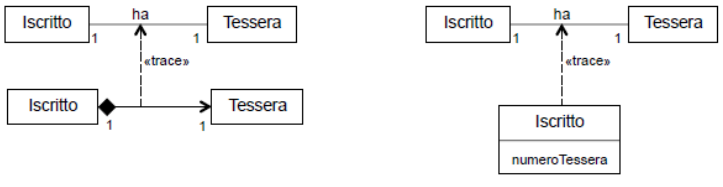
## Come identificare le associazioni di progettazione?

Associazioni bidirezionali e classi associative non sono direttamente implementabili.

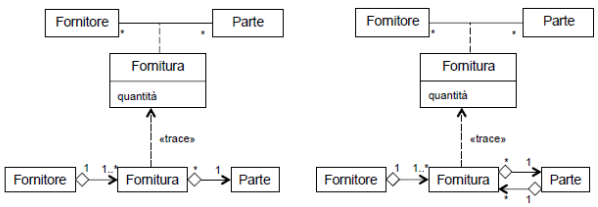
Le associazioni di progettazione sono ricavate da quelle di analisi, effettuando una trasformazione basata sul carico di lavoro a cui sono sottoposte le associazioni.

Le associazioni di progettazione devono specificare **nome, verso di navigabilità, molteplicità ad entrambi gli estremi e nome del ruolo di destinazione**.

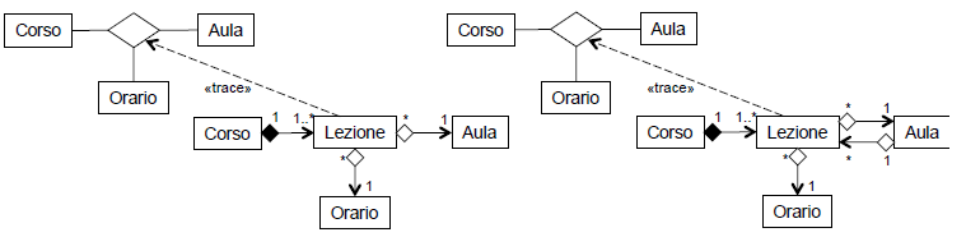
❒ Associazioni uno-a-uno



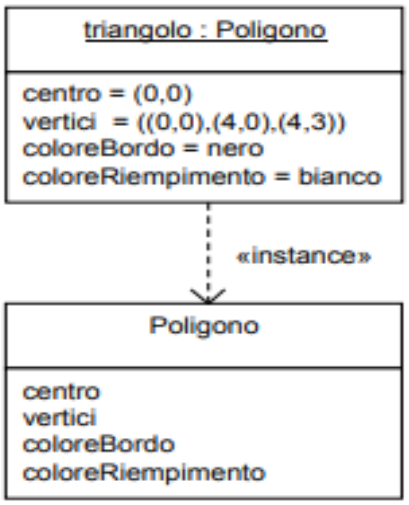

❒ Classi associative




❒ Associazioni ternarie



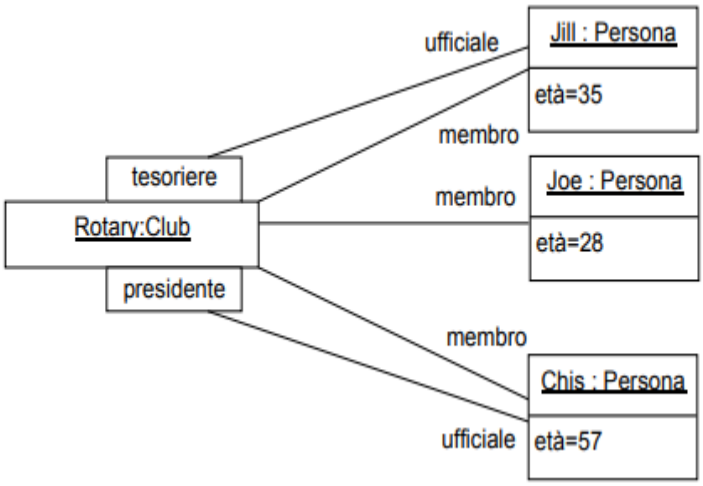
# Diagramma degli oggetti




**Il diagramma degli oggetti** è un diagramma di tipo **statico** per descrivere un sistema in termini di oggetti e relazioni. Il diagramma è molto simile a quello delle **classi**, per ogni classe abbiamo un oggetto che rappresenta un'istanza della classe.

Nella figura a lato abbiamo la classe Poligono ed una sua relativa istanza, il triangolo.

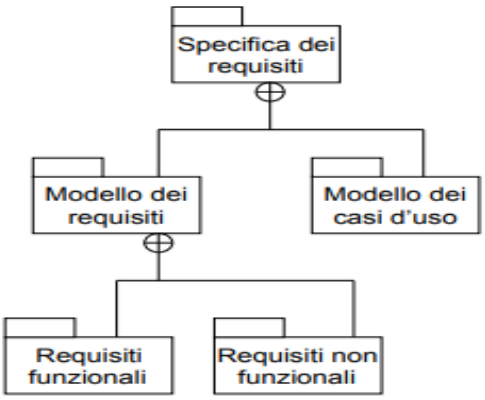
Un diagramma a oggetti può essere utilizzato come test durante la fase di analisi di un progetto per verificare l'accuratezza e la completezza del diagramma delle classi.




Un diagramma delle classi può contenere anche delle istanze; l'esempio a lato è un caso particolare di diagramma delle classi in cui compaiono solo oggetti.

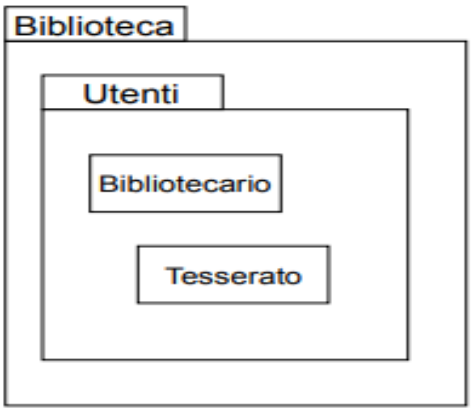
# Diagramma dei package

Un **package** è un raggruppamento di elementi del modello correlati. Ciascun package viene individuato da un **namespace** ed un package può contenere al suo interno altri package così da formare un organizzazione gerarchica dei package.



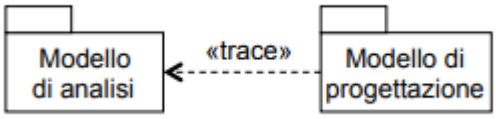

Nel diagramma a sinistra vengono utilizzate relazioni di **contenimento**. (Si consiglia di mostrare al massimo due livelli).

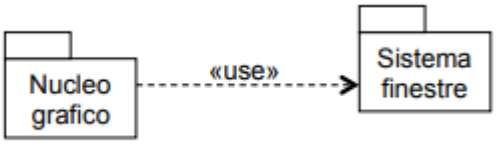
Nel diagramma di destra i package vengono annidati tra di loro formando una gerarchia. I package annidati vedono i nomi dei package che li contengono, viceversa i package esterni non vedono i nomi dei package che contengono.




## Dipendenze tra package

1. **<<use>>**: (default) quando un elemento del package cliente usa un elemento del package fornitore
2. **<<import>>**: quando gli elementi pubblici del package fornitore vengono aggiunti come elementi pubblici del package cliente
3. **<<access>>**: quando gli elementi privati del package fornitore vengono aggiunti come elementi pubblici del package cliente
4. **<<trace>>**: rappresenta l'evoluzione di un elemento in un altro elemento più dettagliato




## Package di analisi

- Sono gruppi di elementi del modello accomunati da **forti correlazioni semantiche**
- La fonte migliore per individuarli è il **diagramma delle classi**. I migliori candidati per essere raggruppati nello stesso package sono le classi appartenenti a: **gerarchie di composizione e gerarchie di specializzazione**.
- Anche il **diagramma dei casi d'uso** potrebbe servire; i casi d'uso che supportano un processo aziendale o un attore possono indicare un package
- Per minimizzare le interdipendenze si possono poi spostare classi tra package, aggiungere package, eliminare package
- Numero ideale di classi per package: tra 4 e 10
- Meglio evitare dipendenze circolari

# Diagrammi di interazione

Rappresentano la struttura dell'interazione tra oggetti durante uno scenario. Esistono quattro tipi di diagrammi di interazione, ognuno rivolto a un particolare aspetto:

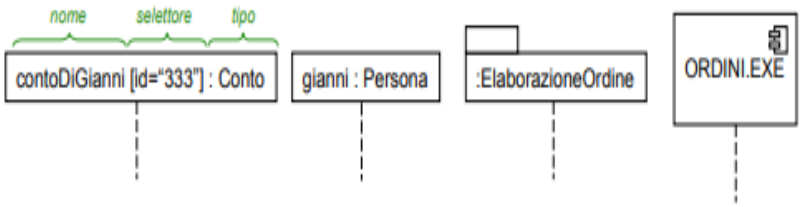
- **Diagramma di sequenza:** enfatizza la sequenza temporale degli scambi di messaggi
- **Diagramma di comunicazione:** enfatizza le relazioni strutturali tra gli oggetti che interagiscono
- **Diagramma di sintesi dell'interazione:** illustra come un comportamento complesso viene realizzato da un insieme di interazioni più semplici
- **Diagramma di temporizzazione:** enfatizza gli aspetti real-time di un interazione

## Terminologia

- **Interazione**: unità di comportamento di un classificatore che ne costituisce il contesto; consiste in un insieme di messaggi scambiati tra le linee di vita all'interno del contesto per raggiungere un obbiettivo
- **Contesto**: può essere dato dall'intero sistema, da un sottosistema, da un caso d'uso etc..
- **Linea di vita**: rappresenta come un'istanza di un classificatore partecipa all'interazione
- **Messaggio**: rappresenta un tipo specifico di comunicazione istantanea tra due linee di vita in un'interazione, e trasporta informazione nella prospettiva che seguirà una attività

## Linee di vita

Sono disegnate con lo stesso simbolo del loro classificatore. Possono avere una **coda** a forma di riga verticale tratteggiata. Non rappresentano specifiche istanze del classificatore, ma modi in cui le istanze partecipano all'interazione.




## Messaggi

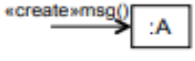
- Per ogni messaggio di chiamata ricevuto da una linea di vita, deve esistere un'operazione corrispondente nel classificatore di quella linea di vita


- messaggio sincrono (il mittente aspetta che il destinatario ritorni)

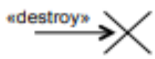

- messaggio asincrono (il mittente continua l'esecuzione)



- messaggio di ritorno (il destinatario restituisce il controllo al mittente)



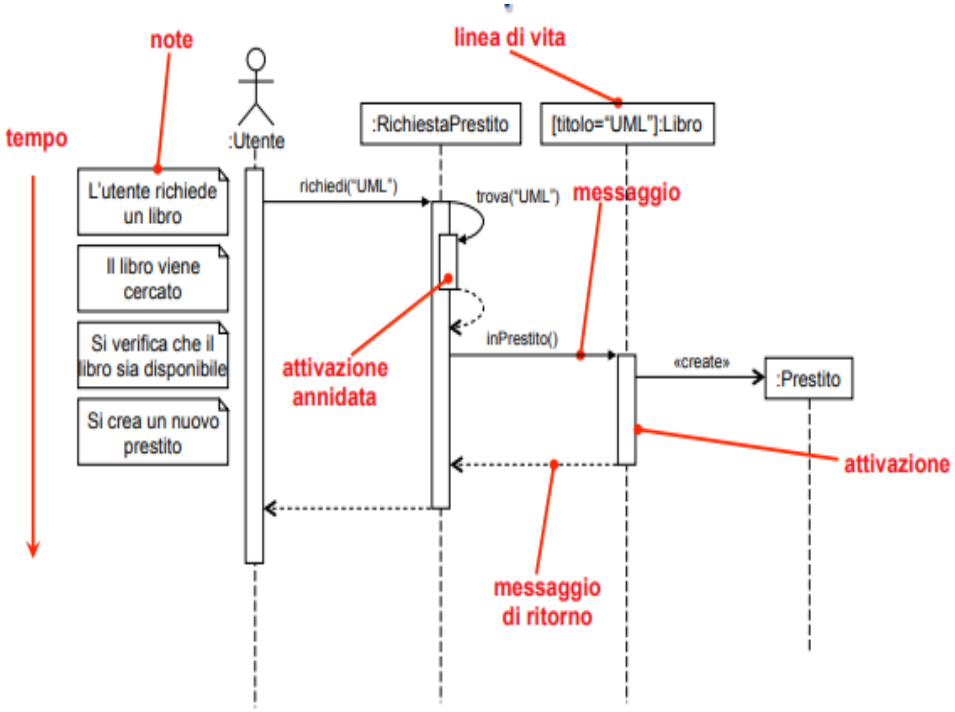

- creazione di un oggetto (si crea un'istanza del classificatore destinatario)




- distruzione di un oggetto (il mittente distrugge il destinatario)

## ***Diagrammi di sequenza***

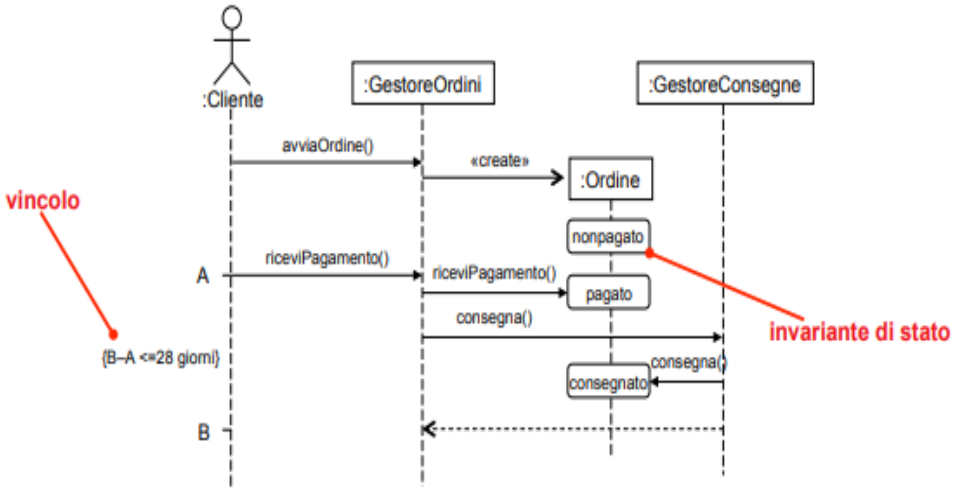
- Mostrano le interazioni tra linee di vita come una sequenza di messaggi ordinati temporalmente
- Sono la forma più ricca e flessibile di diagramma di interazione



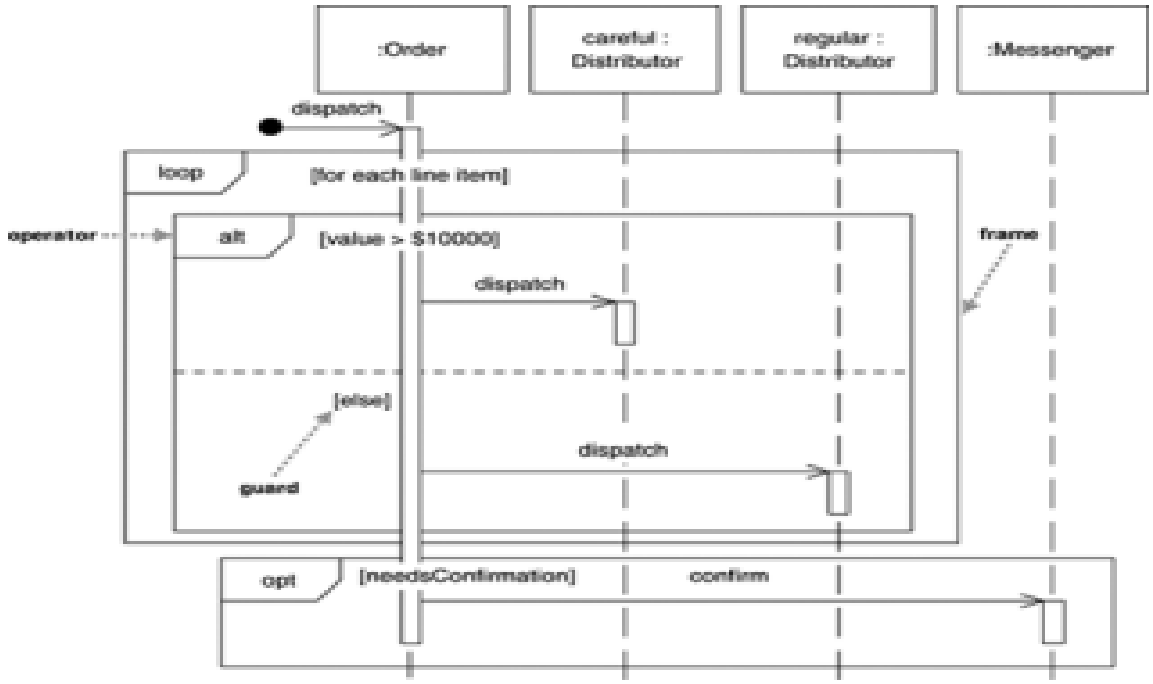

➢ Hanno due **dimensioni**: la dimensione verticale rappresenta il tempo mentre quella orizzontale le linee di vita
➢ Un'**attivazione** mostra il periodo durante il quale una linea di vita esegue un'azione
➢ Si possono specificare **nodi decisionali**, **iterazioni**, **attivazioni annidate**
➢ E' consigliato scrivere il flusso tramite un insieme di **note** poste accanto agli elementi

## Invarianti di stato e vincoli

- Quando un'istanza riceve un messaggio, il suo stato può cambiare
- Lo **stato** delle istanze può essere mostrato sulle linee di vita
- Un **vincolo** sulle linee di vita indica una condizione sulle istanze che deve essere vera da lì in avanti



## Frammenti combinati

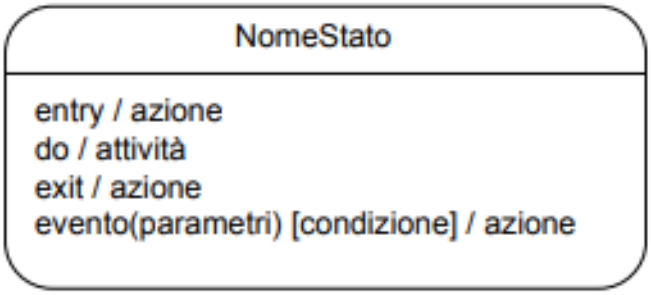



# Diagrammi di stato

I diagrammi di stato descrivono l'evoluzione temporale delle istanze di un classificatore in risposta alle interazioni con altri oggetti. Ogni **classe** può avere associato un diagramma di stato. **UML** adotta la notazione di Harel.

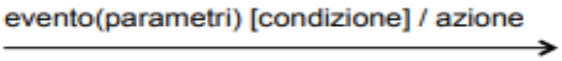
## Stati ed eventi

- **Stato** di un oggetto: è un'astrazione dell'insieme dei valori dei suoi attributi e dei suoi collegamenti
- **Evento**: provoca la transizione tra uno stato e l'altro
- Uno stato può contenere **azioni** e **attività**
    - Le **azioni** sono operazioni istantanee, atomiche e non interrompibili; sono associate a transizioni attivate da eventi
    - Le **attività** sono operazioni che richiedono un certo tempo per essere completate e possono quindi essere interrotte da un evento



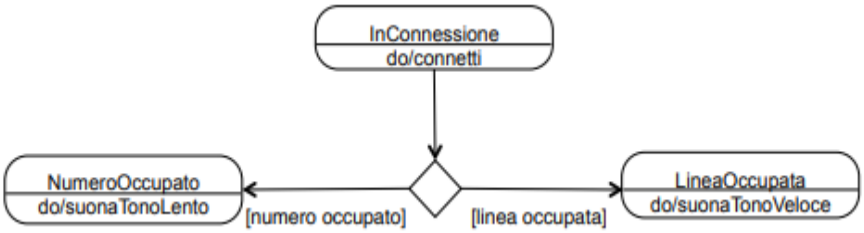
## Transizioni

- Una **transizione** individua il cambiamento di stato di un oggetto, ed è associata ad uno o più eventi e, opzionalmente, a condizioni e azioni
- Un **evento** avviene in un preciso istante di tempo, e si assume che abbia durata nulla
- Una **condizione** è un'espressione booleana che deve risultare vera affinchè la transizione possa avvenire
- Un'**azione** è un'operazione istantanea che viene eseguita all'atto della transizione




Una transizione che esce da uno stato e non riporta alcun evento indica che la transizione avviene al termine dell'attività.


## Pseudo-stato di selezione




Consente di dirigere il flusso nell'automa secondo le condizioni specificate sulle sue transizioni di uscita

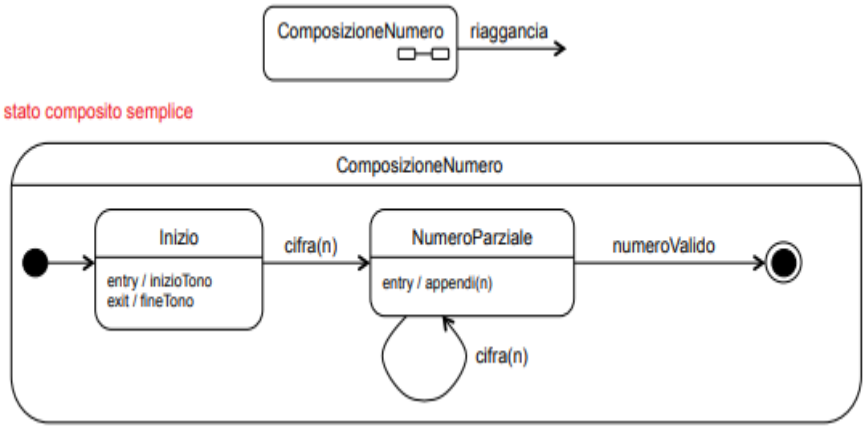
## Tipi di eventi

- **Evento di variazione**: si verifica nel momento in cui una condizione si avvera
  - si denota con un'espressione booleana, es: (bilancio < 0)
  - verrà controllato solo al variare dei parametri coinvolti
- **Evento di segnale**: si verifica quando un oggetto riceve un oggetto segnale da un altro oggetto
- **Evento di chiamata**: è l'invocazione di una specifica operazione nell'istanza del classificatore
  - denotato dalla signature dell'operazione
  - può essere associato a una sequenza di azioni separate da ";"
- **Evento temporale**: si verifica allo scadere di un periodo di tempo
  - **When**: specifica il momento della transizione, es: (when[data=01/01/2008])
  - **After**: specifica dopo quanto tempo deve avvenire la transizione, es: (after[10 seconds]

## Stati compositi

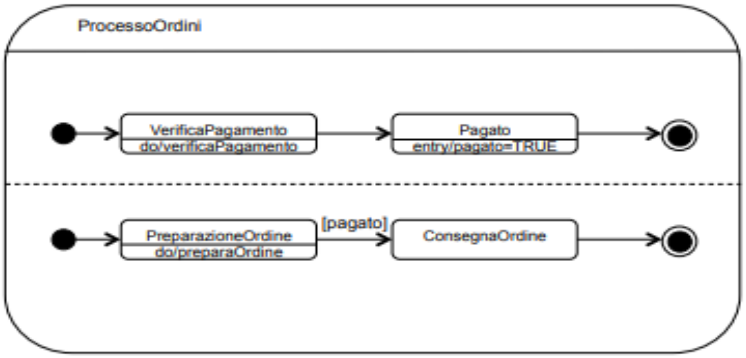
Uno stato **composito** è uno stato che contiene altri stati annidati, organizzato in uno o più automi.

Ogni stato annidato eredita tutte le transizioni dello stato che lo contiene.




## Comunicazione tra automi

La comunicazione tra più automi avviene in modo **asincrono.** Nell'esempio affianco la comunicazione avviene attraverso la variabile pagato.




# Diagrammi di attività

Modellano un processo come un'attività costituita da un insieme di nodi connessi da archi.

Un'attività può essere associata a qualunque elemento di modellazione:

- Caso d'uso
- Operazione
- Classe
- Interfaccia
- Componente
- Collaborazione

## Attività

Sono modellate come reti di **nodi** connessi da **archi**.

Categorie di nodi:

- **Nodi azione**: rappresentano compiti atomici
- **Nodi controllo**: controllano il flusso all'interno dell'attività
- **Nodi oggetto**: rappresentano oggetti usati nell'attività

Categorie di archi:

- **Flussi di controllo**: rappresentano il flusso di controllo attraverso l'attività
- **Flussi di oggetti**: rappresentano il flusso di oggetti attraverso l'attività

### Nodi azione

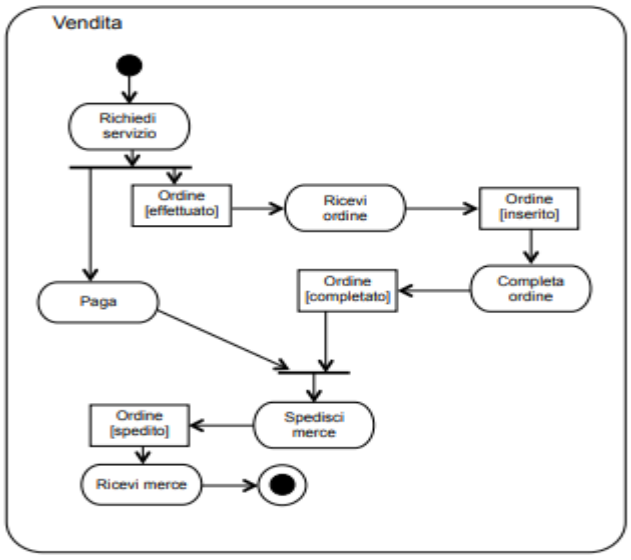
- Nodo azione di chiamata
  - chiama un comportamento — Crea ordine
  - chiama un'attività — Crea ordine
  - chiama un'operazione — Stampa ordine (Ordine::stampa)
- Nodo azione di accettazione evento temporale
  - produce un evento temporale ogni volta che la condizione temporale diventa vera — Apri porta → Chiudi porta (aspetta 10 sec)
  - diventa attivo solo quando si attiva l'arco — Invia dichiarazione redditi (fine anno fiscale)

### Nodi controllo

- Nodo iniziale: indica l'inizio del flusso
- Nodo finale dell'attività: termina un'attività
- Nodo finale del flusso: termina uno specifico flusso
- Nodo decisione: divide il flusso in più flussi alternativi — [condizione1], [condizione2], else
- Nodo fusione: ricongiunge i flussi a valle di un nodo decisione
- Nodo biforcazione: divide il flusso in più flussi concorrenti
- Nodo ricongiunzione: sincronizza flussi concorrenti

### Nodi oggetto

- I nodi oggetto indicano che sono disponibili istanze di una data classe in un punto specifico dell'attività
- Gli archi in entrata e uscita dai nodi oggetto rappresentano flussi di oggetti creati e consumati da nodi azione
- E' possibile rappresentare esplicitamente lo stato di un oggetto



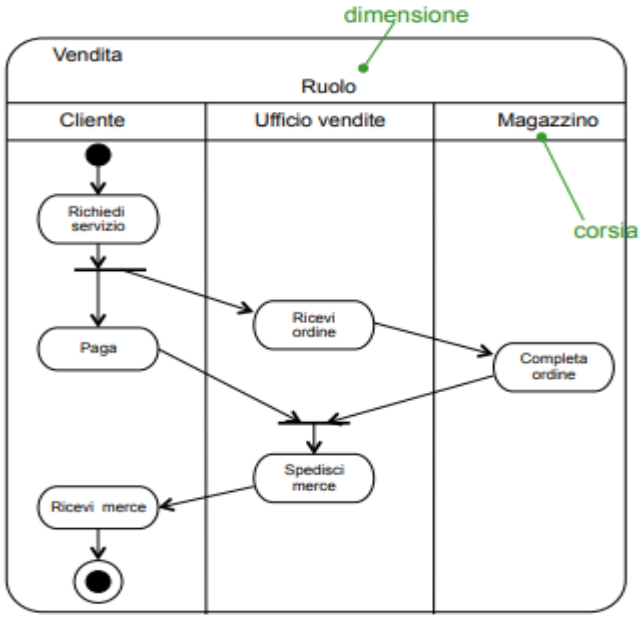
## Corsie

Le attività possono essere partizionate in corsie che raggruppano insiemi di azioni correlate.

Le corsie possono corrispondere a:

- Casi d'uso
- Classi
- Componenti
- Unità organizzative
- Ruoli

La semantica di ogni insieme di corsie è descritta da una **dimensione**.

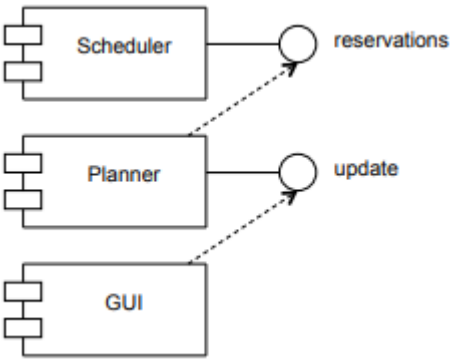


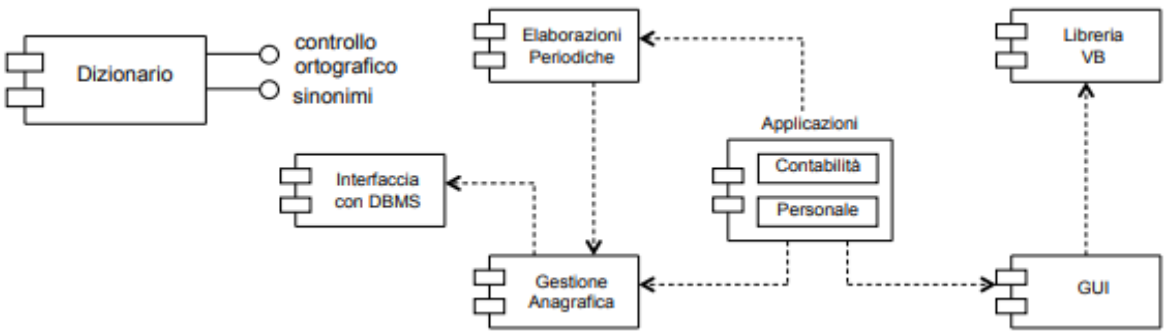

# Diagramma dei componenti

- Mostra i componenti e le loro interdipendenze
- Un **componente** è una parte modulare di un sistema che incapsula i suoi contenuti
- I componenti possono avere attributi e operazioni, e possono partecipare ad associazioni e generalizzazioni

## Componenti

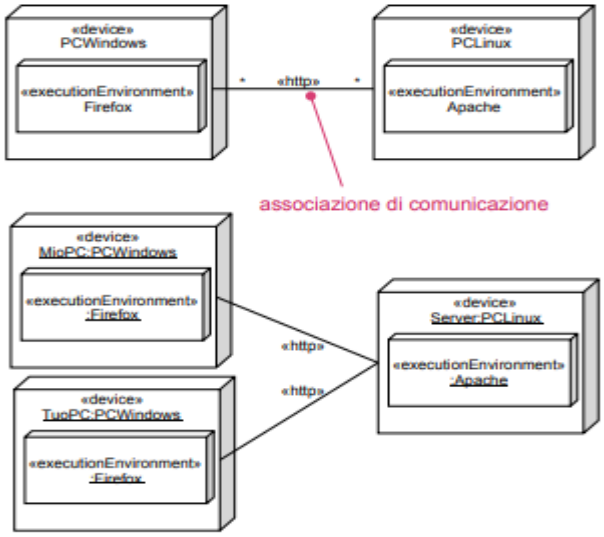
I componenti possono contenere oggetti, ad indicare che essi ne sono parte integrante. I componenti sono connessi tra loro mediante **dipendenze**, possibilmente tramite **interfacce** ad indicare che un componente usa i servizi di un altro componente.



# Diagrammi di deployment

- Specifica l'hardware su cui verrà eseguito il software e il modo in cui il software è dislocato sull'hardware
- Può avere due forme
  - Descrittore: contiene nodi, relazioni tra nodi e manufatti; modella tipi di architetture
  - Istanza: contiene istanze di nodi, di relazioni tra nodi e di manufatti; modella un deployment dell'architettura su un particolare sito



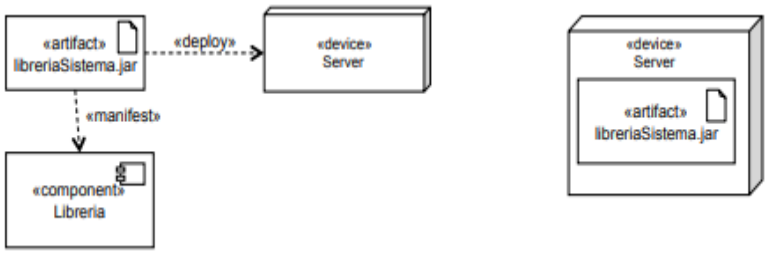

## Nodi

- Un **nodo** rappresenta una risorsa computazionale su cui i manufatti possono essere dislocati per essere eseguiti.
- Due stereotipi standard:
  - **<<device>>**: rappresenta un tipo di periferica, es: PC
  - **<<executionEnvironment>>**: rappresenta un ambiente software di esecuzione, es: web server
- I nodi possono essere annidati in nodi
- Un'associazione tra nodi rappresenta un canale di comunicazione tra nodi

## Manufatti

Un manufatto rappresenta un'entità concreta del mondo reale, per esempio:

- File sorgenti
- File eseguibili
- Script
- Tabelle di database
- Documenti
- Modelli UML




I manufatti vengono dislocati sui nodi.