PROGETTAZIONE DI SISTEMI ORIENTATI AGLI OGGETTI (OOD)

Michele Marchesi

michele@diee.unica.it

Dipartimento di Ingegneria Elettrica ed Elettronica

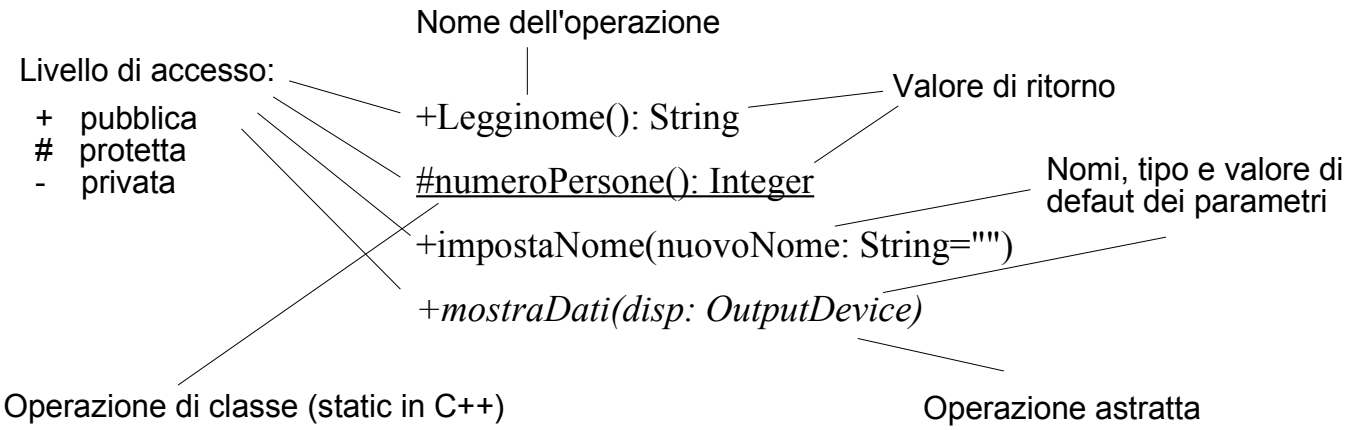
Università di Cagliari

Operazioni

Operazione. Un operazione è un comportamento specifico della cui esibizione un oggetto è responsabile.

- Un'operazione è in stretto rapporto con le *responsabilità* di una classe, eccettuate le responsabilità di mantenere delle informazioni.
- Un'operazione può anche essere assegnata ad una classe e non alle sue istanze (operazioni **static** di Java e C++, o **di classe** dello Smalltalk). In tal caso, equivale a una funzione globale.
- Le operazioni si invocano sempre su di una data istanza della classe (invio di un messaggio all'oggetto).

- In UML, le operazioni si elencano nel rettangolo in basso relativo ad una classe
- Ogni operazione è caratterizzata dal nome, dal tipo del valore di ritorno, dal nome, tipo e valore di default degli argomenti.
- Un'operazione astratta, cioè definita ma non implementata si denota in corsivo



Identificazione delle operazioni

- Le operazioni si dividono in due categorie:
  - Operazioni algoritmicamente semplici.
  - Operazioni algoritmicamente complesse.
- Alla prima categoria appartengono le operazioni che tutti gli oggetti devono avere, per il semplice fatto di esistere e di avere degli attributi e delle relazioni con altri oggetti.
- Tali operazioni sono implicite e non compaiono nel diagramma delle classi.
- Le operazioni algoritmicamente semplici sono:

  Crea: crea ed inizializza un nuovo oggetto. *Questa operazione è associata alla classe, e non all'oggetto* (che ancora non esiste!).

  Cancella: disconnette e cancella un oggetto.

  Leggi: restituisce un oggetto connesso o il valore di un attributo.

  Scrivi: scrive il valore di un attributo.

  Connetti: connette un oggetto con un altro.

  Disconnetti: scollega un oggetto da un altro.

- Si noti che *tutti gli attributi devono essere privati*, e per leggerli e scriverli sono sempre necessarie le operazioni corrispondenti
- Per individuare le operazioni algoritmicamente complesse, per ogni oggetto, ponetevi le domande:
  - A che richieste dovrò rispondere?
  - Che calcoli dovrò fare su di me o sugli insiemi di oggetti da me contenuti?
  - Che tipi di selezione dovrò fare sugli oggetti da me contenuti?
  - Che controlli su sistemi esterni dovrò eseguire per scoprire e rispondere ad un cambiamento in un sistema o in un dispositivo esterno?
- Alcuni gruppi di verbi tra cui scegliere:
  - ***attiva*** (inizializza, apri, fa partire)
  - ***calcola*** (computa, conta, stima, classifica)
  - ***controlla*** (dirigi, gestisci, opera, osserva)
  - ***determina*** (conferma, decidi, valuta, risolvi, stabilisci)

- ***disattiva*** (chiudi, finisci, scollega, spegni, termina)
- ***misura*** (calibra, converti, limita, vincola)
- ***qualifica*** (caratterizza, differenzia, discrimina, distingui, evidenzia)
- ***rispondi*** (replica, restituisci)
- ***seleziona*** (designa, prendi, prescegli, opta per)
- ***trova*** (cerca, indica, ottieni)

- Se un oggetto contiene altri oggetti (parti, oggetti contenuti), avrà facilmente operazioni del tipo:
  - numeroContenuti
  - calcolaSuiContenuti, totalizza
  - trovaContenutiSpecifici
  - inviaComandoAlleParti

Assegnazione delle operazioni

- Il principio generale da seguire è di distribuire in modo bilanciato le operazioni nel sistema.
- Ogni operazione va messa "vicino" ai dati ad essa necessari.
- Tenete presente che la lunghezza *media* delle operazioni non dovrà superare le 4-8 istruzioni!
- Applicate l'ereditarietà:
  - Posizionate le operazioni più generali in alto
  - Posizionate le operazioni specializzate in basso
- Se un oggetto si occupa solo di occultare i dati e passarli a chi li richiede, controllate se può fare di più sui propri dati, distribuendo così meglio le responsabilità.
- Fate fare ad una collezione solo il lavoro che riguarda l'insieme della collezione. Ponete il lavoro specifico in ogni componente.
- I metodi che traggono aiuto da piccoli metodi che fanno una sola cosa ma bene, facilitano il riuso

Specifica delle operazioni

- Date il nome alle operazioni:
  - Vocabolario standard del dominio del problema.
  - Mettete i verbi all'imperativo (scrivi, esegui...) o in 3 persona (scrive, esegue...) in modo consistente in tutto il sistema.
- Descrizione di una o due righe.
- Parametri di input/output, loro tipo e significato.
- Precondizioni e postcondizioni.
- Eventi che attivano l'operazione.
- Applicabilità dell'operazione negli stati.
- Visibilità (pubblica, protetta, privata).
- L'operazione stessa può essere documentata con pseudo-codice, con un diagramma a blocchi o col codice del linguaggio di prototipazione (Smalltalk va molto bene!).

Il modello dinamico

- Il modello dinamico rappresenta il sistema durante il suo funzionamento.
- Esso consiste nella definizione di *scenari di funzionamento* del sistema, in risposta ad eventi esterni o in fasi particolarmente significative.
- Un potente strumento di descrizione dinamica sono i *casi d'uso* o le *user stories* del sistema, descrizioni in cui si evidenzia la risposta ad eventi esterni.
- Talora possono utili essere anche diagrammi che rappresentano l'invocazione di metodi, in cui si indica la sequenza e la temporizzazione dell'invio successivo di messaggi.
- L'analisi del funzionamento del sistema aiuta a verificare il modello dei dati, ed in particolare:
  - la completezza delle rappresentazioni delle classi in termini di attributi, relazioni e operazioni.
  - La corretta assegnazione delle operazioni alle varie classi.

- Non conviene spingere la definizione del modello dinamico sino ai minimi dettagli:
  - i diagrammi dinamici si possono usare per descrivere il funzionamento del sistema nei casi piu' critici.
  - la definizione dettagliata di tutte le operazioni del sistema può essere fatta costruendone il *prototipo funzionante.*

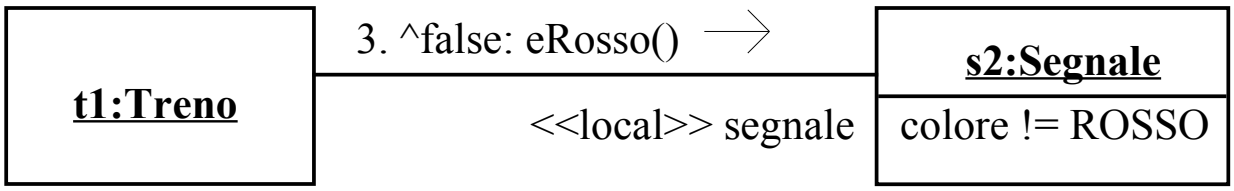
Connessione fra oggetti. Una connessione fra oggetti modella le dipendenze di un oggetto per quanto riguarda le elaborazioni, indicando la necessità di richiedere servizi per soddisfare le sue responsabilità.

- Una connessione fra oggetti mostra graficamente l'accesso ad un oggetto da parte di un altro, e la richiesta di un'operazione al primo.
- Essa è in stretto rapporto con le *collaborazioni* di un oggetto.

- Nella notazione UML, esistono due tipi di diagrammi per la specifica delle operazioni, entrambi chiamati *diagrammi di interazione*:
  - I diagrammi di comunicazione (o collaborazione)
  - I diagrammi sequenziali

- I diagrammi di comunicazione mostrano e qualificano le connessioni fra oggetti, ed i messaggi tra questi scambiati.
- I diagrammi sequenziali evidenziano i messaggi scambiati tra gli oggetti e ne danno la sequenza e la durata temporale.
- I diagrammi di collaborazione e sequenziali si possono convertire entro certi limiti gli uni negli altri.

Notazione dei diagrammi di collaborazione:



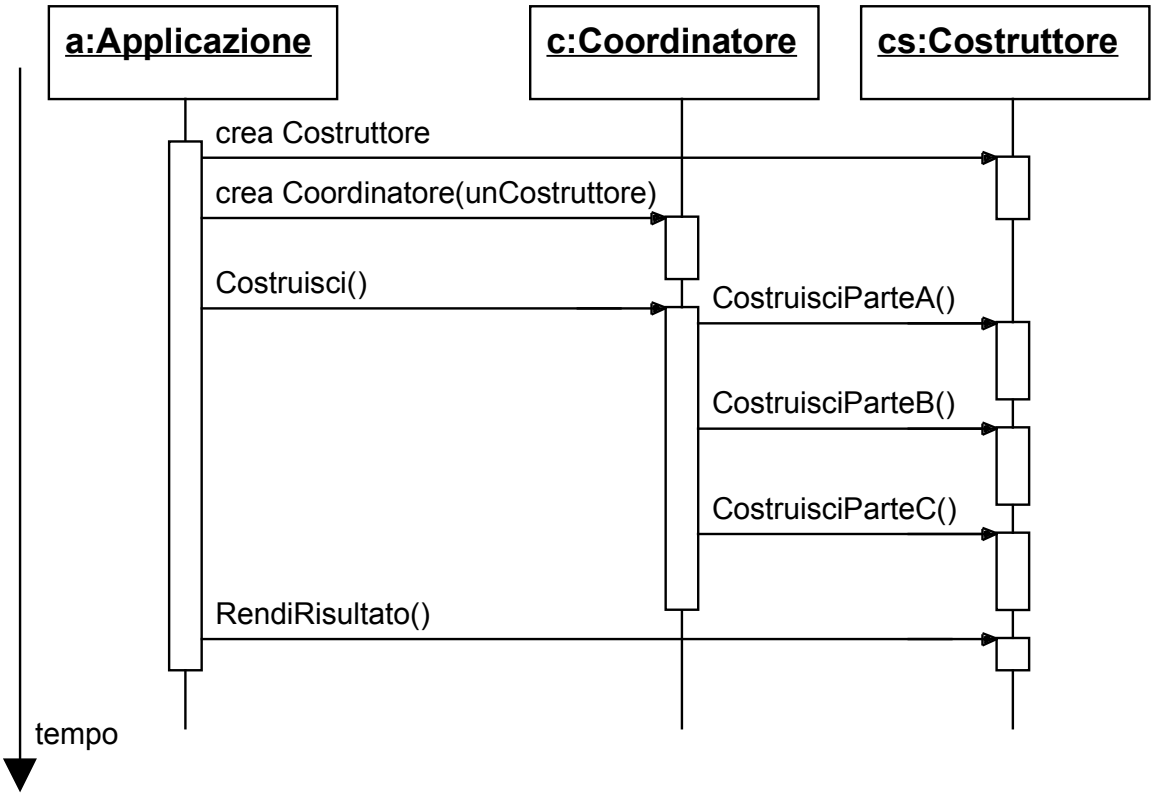
- Gli oggetti sono mostrati come rettangoli, con all'interno:
  - Nome oggetto:nome classe, sottolineati per distinguersi dal simbolo di classe
  - Eventuali ornamenti nella sez. superiore, come *new*, che indica che l'oggetto è stato appena creato
  - Eventuali vincoli o informazioni di stato nella sezione inferiore.
- Le connessioni tra oggetti sono mostrate come linee continue tracciate tra gli ogggetti:
- Alle estremità di tali linee vi può essere l'indicazione di come l'oggetto è accessibile all'altro connesso
- Gli invii di messaggi sono denotati da frecce poste parallelamente alle connessioni

- I possibili modi di accesso di un oggetto (ricevente) da parte di un altro (mittente)

Stereotipo	Nome	Note
“association”	associa-zione	il ricevente è visibile in quanto associato al mittente
“global”	globale	il ricevente è una variabile globale
“local”	locale	il ricevente è una variabile locale dell'operazione eseguita dal mittente
“parameter”	parametro	il ricevente è un parametro dell'operazione eseguita dal mittente
“self”	self	il ricevente è lo stesso mittente

- I messaggi sono numerati, possono essere preceduti da condizioni (tra parentesi quadre), e possono rappresentare flussi multipli

Notazione dei diagrammi sequenziali



- Gli oggetti sono mostrati come linee verticali, col nome e la classe entro un rettangolo
- Il flusso temporale scorre dall'alto in basso
- Tra gli oggetti vi possono essere invii di messaggi, denotati da frecce
- Gli ispessimenti delle linee verticali denotano un'elaborazione in corso

Scenari di funzionamento del sistema

- Uno scenario mostra come un gruppo di oggetti nel sistema reale interagiscono, inviandosi sequenzialmente dei messaggi.
- Gli scenari servono a modellare la dinamica del sistema e a scoprire ulteriori oggetti e responsabilità.
- In fase di OOD, occorre modellare solo quei pochi scenari veramente cruciali al funzionamento del sistema.
- Come linea guida, utilizzate le storie di funzionamento del sistema raccolte coi primi requisiti.
- Ogni connessione di messaggio rappresenta l'invio di un messaggio e *deve corrispondere a un'operazione* della classe del ricevente o di una sua superclasse. Numerate sequenzialmente le connessioni ed evidenziate testualmente:
  - Nome dell'operazione invocata.
  - Eventuali parametri.
  - Valore restituito.

Come definire gli scenari

- Per ogni scenario:
  - Iniziate dall'oggetto (o dall'evento esterno) che scatena le operazioni.
  - Tracciate il primo messaggio.
  - Immedesimatevi nell'oggetto (o classe) che riceve il messaggio. Chiedetevi:
    - ho tutte le informazioni per eseguire l'operazione richiesta?
    - se no, chiedetevi a quali altri oggetti occorre inviare messaggi per ottenerle, e tracciate sequenzialmente tali connessioni.
  - Controllate la completezza degli argomenti del messaggio.
  - Controllate la creazione e cancellazione di oggetti.
  - Iterate il procedimento, sino all'identificazione di tutta la sequenza dei messaggi.

Verifica degli scenari

- Controllate sempre come fa un oggetto ad accedere al ricevente di un messaggio. Eventualmente, aggiungete altri messaggi per ottenere tale ricevente.
- L'invio di un messaggio ad un oggetto non può avvenire che durante l'esecuzione di un metodo del chiamante.
- I possibili modi di accesso di un oggetto (il *cliente*) a un altro oggetto (il *ricevente*) sono:
  - Il ricevente è contenuto nel cliente (per valore o per riferimento)
  - Il ricevente è passato come parametro al metodo
  - Il ricevente è il risultato dell'invio di un messaggio entro il metodo del cliente; vi sono due casi:
    - il ricevente è creato ex-novo
    - il ricevente esisteva già
  - Il ricevente è un oggetto globalmente accessibile (come nel caso di chiamata ad una classe)
- Se occorre attraversare varie connessioni tra oggetti

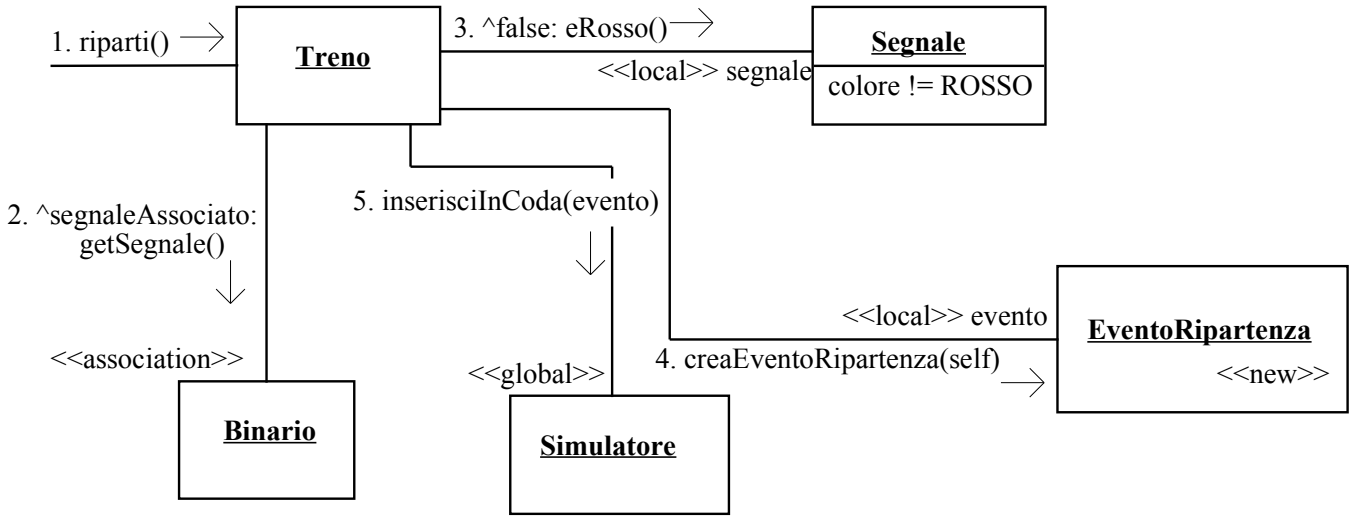
per trovare quello con cui interagire:

- Prima ricercate l'oggetto voluto.
- Poi interagite con esso.

- Se c'è troppo "traffico" di messaggi per accedere ad un oggetto, considerate se sia opportuno aggiungere una associazione per accedervi permanentemente.

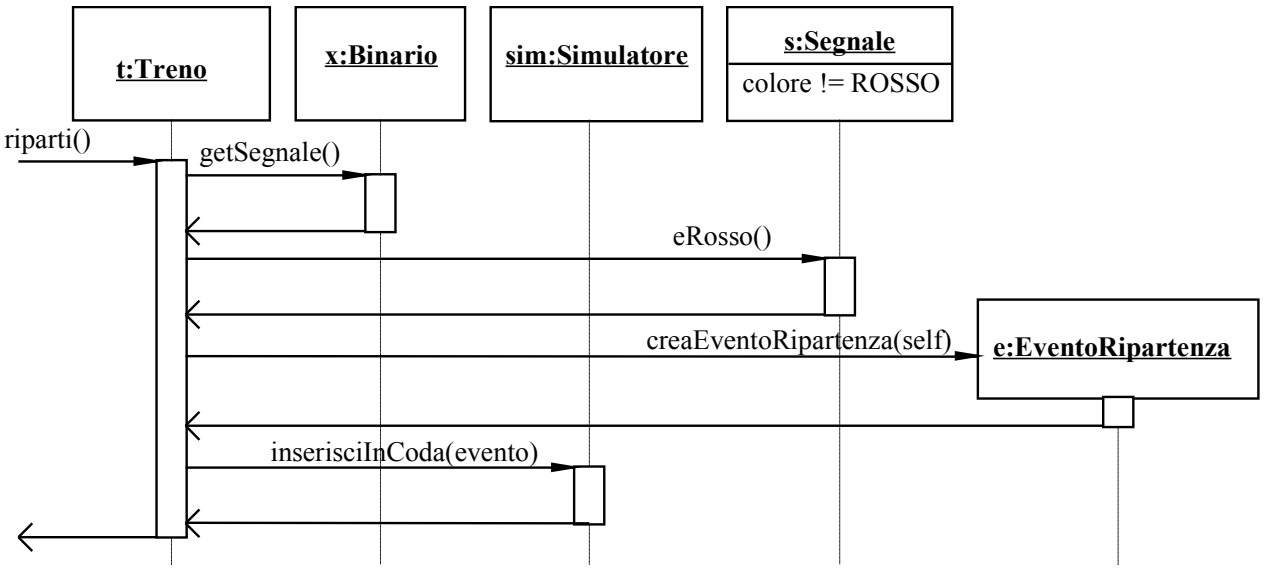
- Controllate se ci sono oggetti che dominano lo scenario, facendo quasi tutto: potrebbe esserci un problema di distribuzione delle responsabilità.
- Un oggetto non deve richiedere ad un altro "chi sei?" e poi inviargli un messaggio di conseguenza. Il ricevente deve già sapere da solo come agire.
- Se un oggetto sa riconoscere che va fatto qualcosa, che sia esso stesso ad iniziare le operazioni, e non un controllore esterno che prima gli chiede la situazione e poi eventualmente inizia le operazioni.
- È comune passare oggetti come parametri, inviando poi ad essi i messaggi al momento giusto.

Un esempio di scenario: simulazione di traffico ferroviario



Scenario di simulazione:

Un treno fermo ad un segnale riceve il messaggio "riparti".
Esso dapprima ottiene il segnale che lo blocca dal binario in cui è posizionato.
Poi chiede al segnale se è rosso. Se no, crea un oggetto di classe Evento-Ripartenza (inviando un messaggio a tale classe) e lo inserisce nella coda degli eventi del simulatore, al tempo corrente.



Lo scenario del diagramma precedente, come diagramma sequenziale

Pacchetti (o package, o sottosistemi)

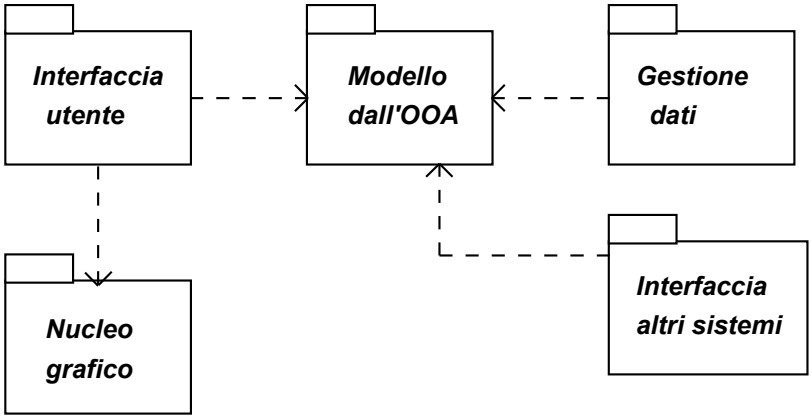
Pacchetto: un sottoinsieme del modello, che contiene altri pacchetti, classi, associazioni e gerarchie di ereditarietà.

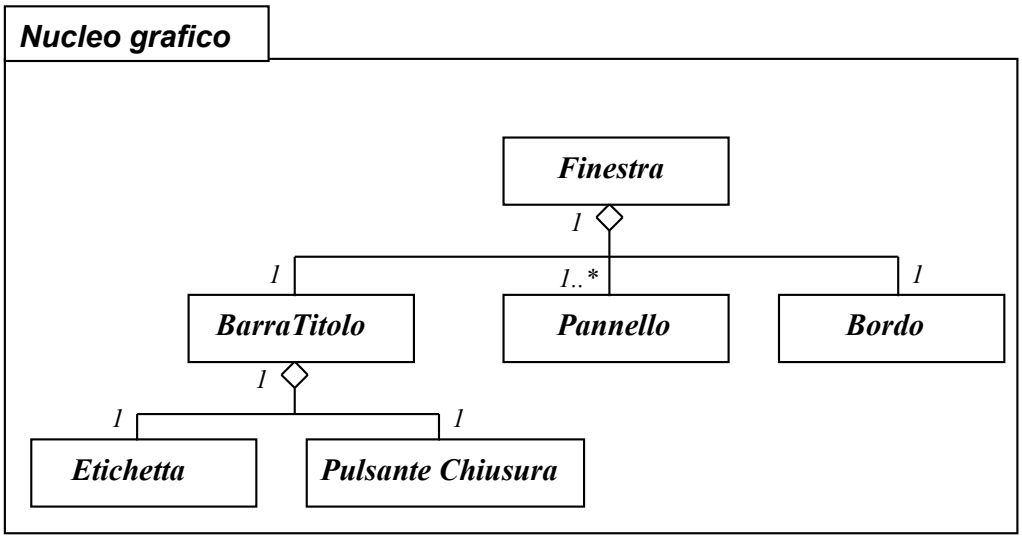
- Un buon metodo di OOD deve poter controllare la visibilità e guidare l'attenzione del lettore
- I pacchetti presentano il modello del sistema a vari livelli di scala: da una vista complessiva giù giù sino a viste molto dettagliate.
- I pacchetti sono solo un modo per organizzare il modello e non hanno altr\i significati, anche se possono essere definiti seguendo la semantica del modello.
- I pacchetti sono anche utili per dividere il lavoro relativo a grandi progetti, basandosi su uno studio OOA/OOD iniziale.
- Il numero delle classi in un modello OOD è in media di 30-50; 100 classi sono tante; per domini di problemi molto complessi vi possono essere quattro o cinque sottodomini, ciascuno con 50-100 classi

Definizione dei pacchetti

- Spesso i pacchetti derivano dal dominio del problema, suddivisibile in modo naturale in insiemi di classi.
- Una tipica suddivisione è fra i dati statici, che descrivono il dominio e che cambiano raramente, ed i dati che il sistema manipola in modo più dinamico.
- In generale, le gerarchie di ereditarietà semplice vanno entro un singolo pacchetto.
- Le aggregazioni/composizioni vanno anch'esse di solito entro un singolo pacchetto.
- Il principio fondamentale da seguire è di *minimizzare le interdipendenze* (associazioni e dipendenze) tra classi appartenenti a pacchetti diversi.
- Ogni pacchetto ha associato un *diagramma delle classi* che può contenere altri pacchetti, oltre che classi e loro relazioni.

Notazione:





Pacchetti con le relative dipendenze, ed un pacchetto espanso con all'interno le sue classi.

- I pacchetti si denotano con un rettangolo con dentro

il nome del pacchetto, con un rettangolino che “spunta” in alto a sinistra.

Le relazioni tra le classi

- Tra le classi (i loro oggetti) possono esistere le seguenti relazioni:
  - Ereditarietà
  - Collaborazione, Usa o Dipende (richiede dei servizi)
  - Associazione
    - Aggregazione
    - Composizione
    - Associazione qualificata
  - Classe parametrica-Classe parametrizzata (istanziazione di classe)
  - Classe-Metaclasse

Clienti e fornitori

- Una relazione associa sempre due classi. Tra queste, una è il cliente (*client*) e l'altra è il fornitore (*supplier*) dei servizi
- I ruoli nelle varie relazioni:

Relazione	***Fornitore***	***Cliente***
Ereditarietà	Superclasse	Sottoclasse
Associazione	Contenuto	Contenente
Dipendenza	Classe da cui si dipende	Dipendente
Istanziazione	Classe parametrica	Classe parametrizzata
Usa	Classe usata	Classe che usa

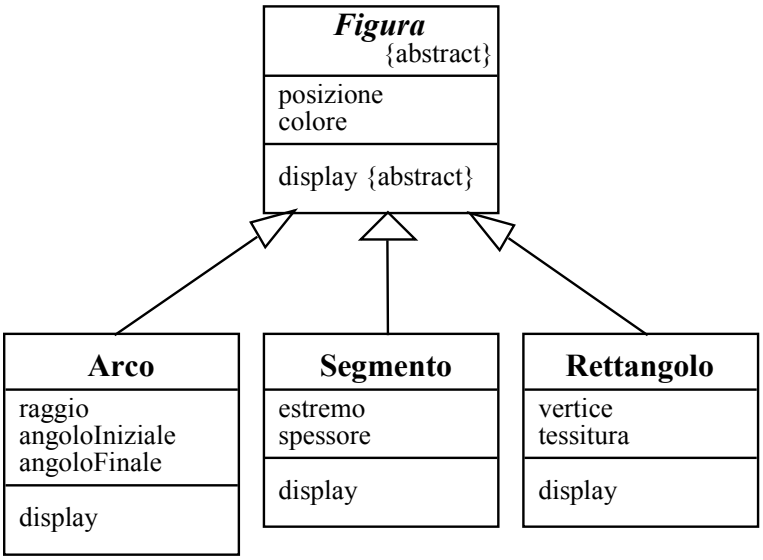
- La cardinalità delle relazioni che la ammettono specifica il numero possibile di clienti di un fornitore, e di fornitori di un cliente

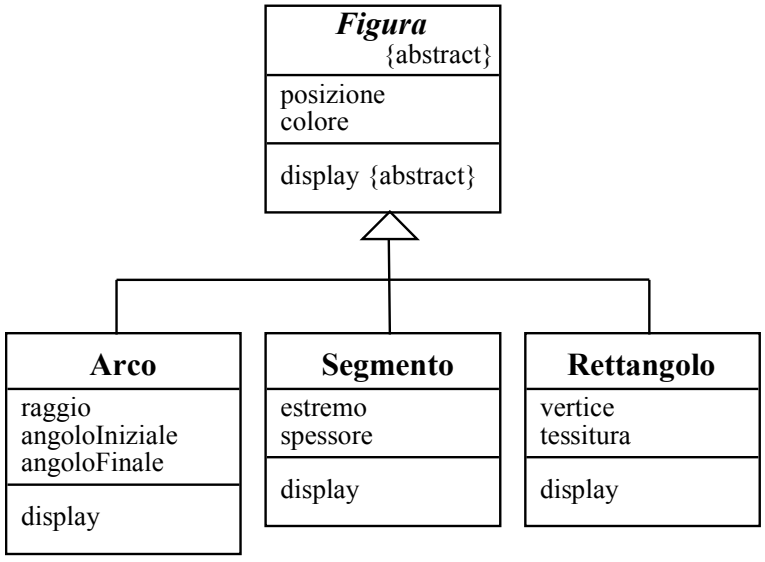
L'ereditarietà

- L'ereditarietà esprime il fatto che una classe (*specializzazione* o *sottoclasse*) è simile ad un'altra classe (*generalizzazione* o *superclasse*), e ne *eredita* relazioni, attributi e operazioni.
- La ricerca delle relazioni di ereditarietà contribuisce a chiarificare il significato delle varie classi e può portare alla scoperta di nuove classi.

Notazioni per l'ereditarietà:

Figura
- Arco
- Segmento
- Rettangolo



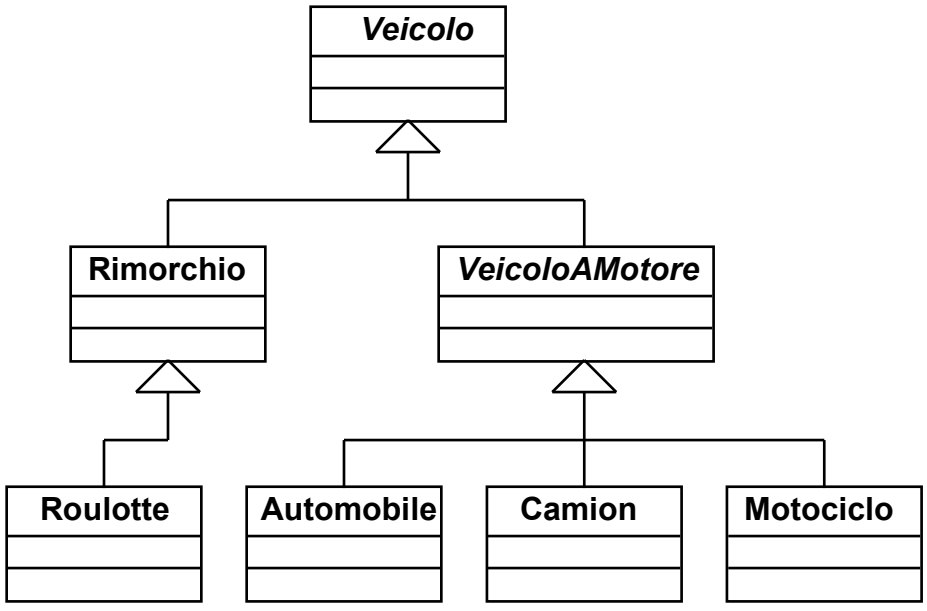


Identificazione dell'ereditarietà

- L'ereditarietà deve rispecchiare una tassonomia effettivamente presente nel dominio del problema.
- Se sono possibili molte specializzazioni, è molto utile considerare dapprima le specializzazione più semplici, e poi proseguire con quelle più complesse.
- Attenzione a non usare l'ereditarietà solo per fattorizzare alcune caratteristiche comuni. Ad es., sia Persona che Dipartimento hanno un indirizzo, ma non per questo c'è dell'ereditarietà!

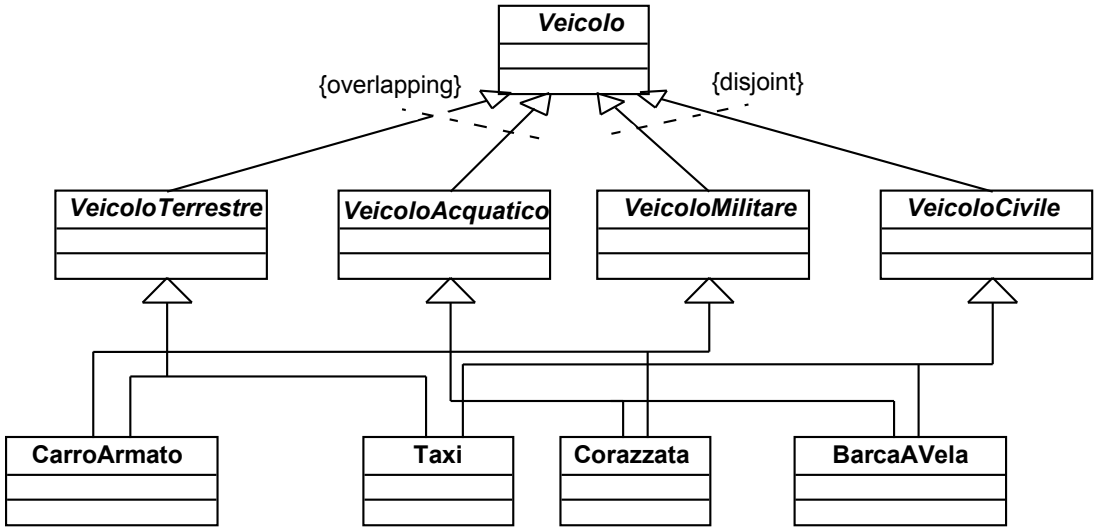
Ereditarietà semplice e multipla

- Tutte le strutture di ereditarietà formano o una *gerarchia* o un *reticolo* (ereditarietà semplice o multipla)
- La forma più comune di ereditarietà è una gerarchia, cioè un albero in cui ogni classe può avere al più una superclasse.
- L'ereditarietà multipla può diventare troppo complessa, e può presentare conflitti di nome.



Una gerarchia di veicoli

- L'ereditarietà multipla può avere o non avere antenati comuni ripetuti.
- I più moderni OOPL (Java, C#) hanno solo ereditarietà semplice, come Smalltalk



Un reticolo di veicoli

- Con C++, è abbastanza comune fare ereditare da classi "di servizio" delle caratteristiche utili (ad es., la capacità di memorizzarsi in un database, di inviare messaggi a oggetti in rete, ecc.). In tal caso, le superclassi sono dette *mixin*.

La relazione *usa*

- Se la classe A usa le risorse della classe B, esiste una relazione *usa* tra la classe A (cliente) e la classe B (fornitore).
- Tale relazione non è simmetrica (A usa B, ma B non usa A)
- Essa si chiama anche *dipendenza* della classe A rispetto alla classe B (UML).
- E' in stretto rapporto col la *collaborazione* tra la classe A e la classe B:
  A collabora con B $\leftrightarrow$ A usa B
- Anche l'associazione e l'aggregazione sono esempi di relazione usa, perché il contenere un oggetto significa usarne i servizi.

Le relazioni associazione e *aggregazione* (contenimento)

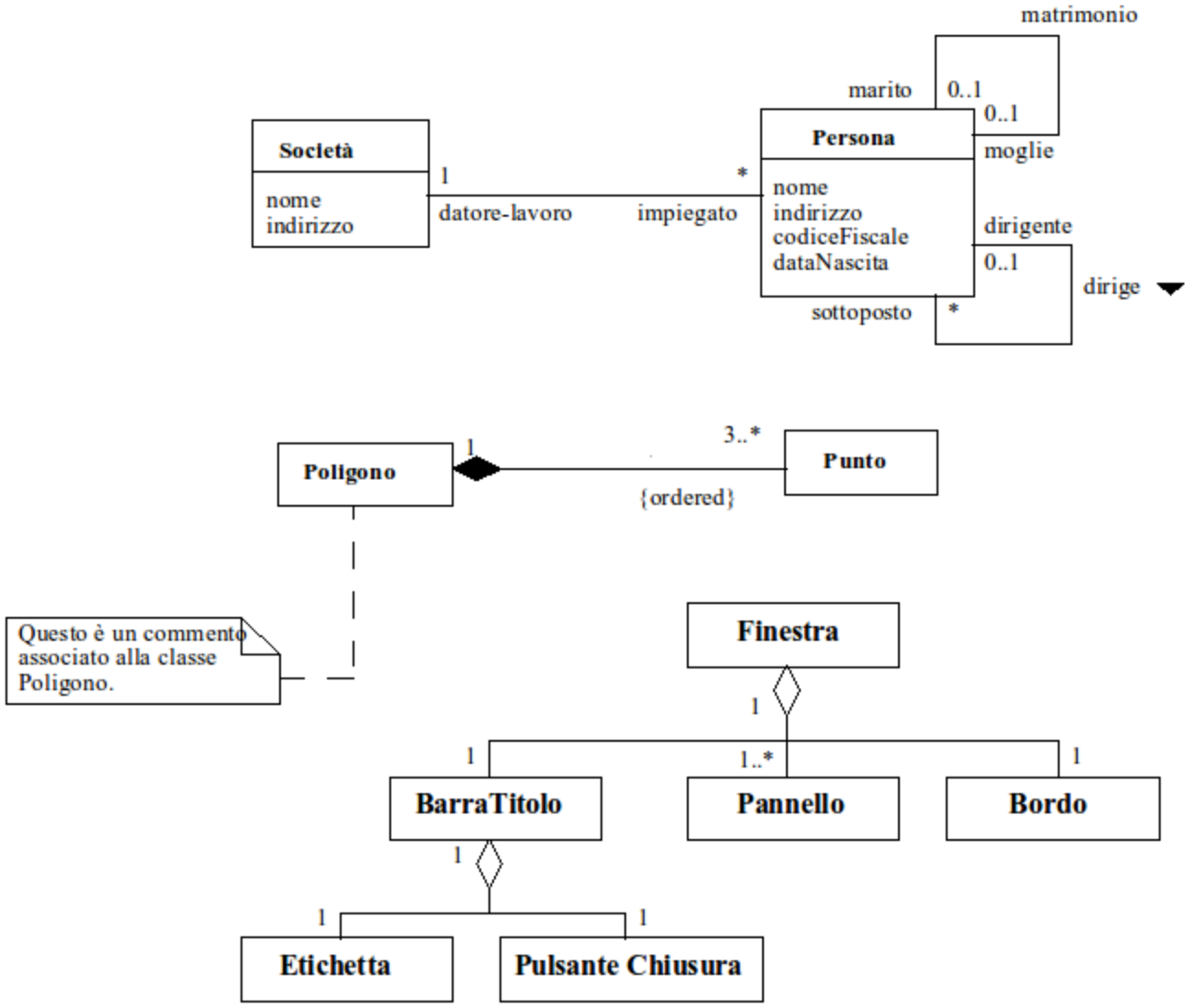
- Nel mondo reale capita spesso che un oggetto, di classe A, debba *tener traccia* permanentemente di uno o più oggetti di classe B, per soddisfare alle proprie responsabilità.
- Talora tale relazione è forte, e rappresenta un contenimento, fisico o logico (*aggregazione*). Ad es.:
  - Un volo contiene dei passeggeri
  - Un automobile contiene un motore

- Altre volte è una relazione più *debole*, non rappresentabile come un contenimento, ma non per questo meno necessaria. Ad es.:
  - Una fattura si riferisce ad un cliente
  - Un evento è legato a un dispositivo
- Dal punto di vista implementativo tale relazione si realizza:
  - inserendo un B (o un puntatore a B) nella struttura dati di A, oppure:
  - tramite un indice che associa l'A al B
- Un caso particolare dell'aggregazione sono gli *attributi*: valori contenuti di tipo primitivo

- Queste relazioni sono tra oggetti ed hanno una *cardinalità*, cioè un numero o un intervallo di oggetti che possono partecipare ad esse.
- Esse hanno anche due *ruoli*, che denotano la parte che ha ciascun oggetto che partecipa alla relazione.
- L'oggetto contenuto può essere unico per tutte le istanze della classe che contiene (variabile di classe, ovvero variabile membro `static`). Tale situazione è piuttosto rara!
- A livello di prima OOD, non vanno fatte considerazioni implementative, ma si può evidenziare la forza della connessione.

- La notazione UML tratta le due relazioni nel modo seguente:
  - L'associazione semplice è denotata da una linea continua.
  - L'aggregazione è denotata da una linea continua con un piccolo rombo dalla parte del contenitore.
  - La composizione (aggregazione anche fisica) è denotata da una linea continua con un rombo nero dalla parte del contenitore.
- Tali relazioni possono avere associato un nome, dei ruoli, la cardinalità e dei vincoli.

Notazione per l'associazione



Vari tipi di associazione.

- Ogni estremità di una riga in un'associazione è marcata con un valore, o con un intervallo, che indica il numero di oggetti associabili a quello dall'altra estremità, in un qualunque istante temporale.
- La convenzione seguita è quella usuale dei diagrammi E-R.
- Cardinalità:

Simbolo	Significato	Esempi
n (un numero)	valore singolo	2
*	da zero a *m*	*
n..m	intervallo	0..2 1..*
n, m	valori o intervalli singoli	3,5 1, 3..*

- Se un'associazione termina con una freccia, significa che è *navigabile* solo nel verso della freccia. Altrimenti, è navigabile in entrambe le direzioni.

Contenimento per valore o per riferimento

- Se un oggetto di tipo A contiene un oggetto di tipo B in modo esclusivo: l'oggetto di tipo B esiste solo in quanto contenuto nell'oggetto di tipo A, si parla di *composizione*, o associazione *per valore*.
- In alternativa, un oggetto di tipo A può contenere, o essere associato ad un oggetto di tipo B non in modo esclusivo, ma condividendolo con altri oggetti. In tal caso, l'associazione si dice *per riferimento*.
- Se la distruzione del contenitore comporta la distruzione dell'oggetto contenuto, il contenimento è per valore. In caso contrario è per riferimento.

- Esempio: un triangolo contiene tre punti (i propri vertici). La cancellazione del triangolo comporta quella dei vertici. Il contenimento è per valore (composizione).
- Esempio: un volo contiene dei passeggeri, ma la cancellazione del volo non comporta quella dei passeggeri. Il contenimento è per riferimento (aggregazione).
- Il contenimento per valore in UML è denotato da un piccolo rombo nero dalla parte del contenitore.

Identificazione delle associazioni

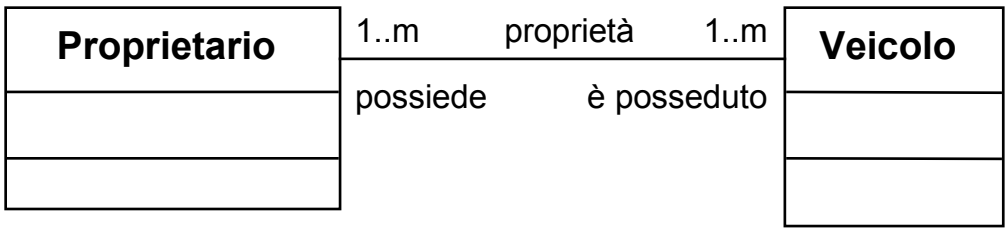
- Cercando potenziali aggregazioni, considerate queste variazioni:
  - Insieme-Parti
  - Contenitore-Contenuto
  - Collezione-Membri (e le sue varietà)
- Considerate ogni oggetto come un tutto unico. Per ciascuna sua parte potenziale, chiedetevi:
  - E' nel dominio del problema?
  - E' entro le responsabilità del sistema?
  - Ha un significato più profondo che non di un semplice valore?
    - Se no, includetela come attributo.
  - Fornisce un'astrazione utile per descrivere il dominio del problema?

- Analogamente, considerate ogni oggetto come una parte. Per ciascun "tutto" potenziale, ponetevi le stesse domande.
- La scelta alternativa è di usare un'associazione, che ha un significato più debole
- In tal caso, per ogni oggetto, ponetevi le domande:
  - A quali altri oggetti sono collegato nel dominio del problema?
  - Da chi devo ottenere informazioni per soddisfare alle mie responsabilità?

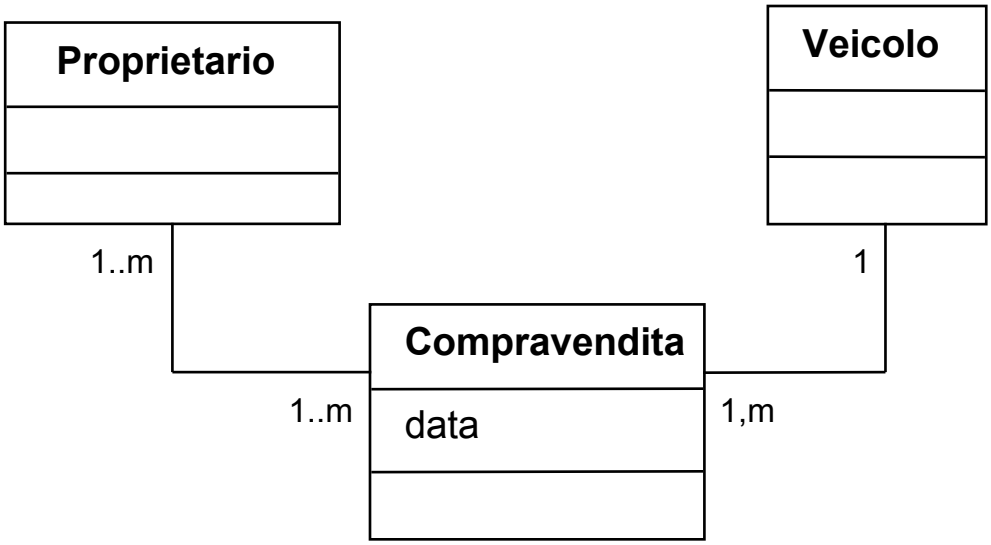
- Per ogni coppia di oggetti, tracciate le linee di connessione, eventualmente col simbolo di aggregazione dalla parte del contenitore
- Per ogni connessione, definite il valore o l'intervallo.
  - Limite inferiore:
    - La connessione è opzionale? il limite inferiore è 0
    - La connessione è obbligatoria? il limite inferiore è maggiore o uguale a 1
  - Limite superiore:
    - La connessione è singola? il limite superiore è 1
    - La connessione è multipla? il limite superiore è > 1

- Date un nome ai due ruoli dei due oggetti in ogni associazione, scelto nella terminologia del dominio del problema.
- Tali nomi diverranno quelli delle variabili d'istanza che implementeranno l'associazione.
- Decidete se il contenimento è per valore o per riferimento (molto probabile), e riportatelo sul diagramma.
- Considerate la navigabilità dell'associazione: se essa è percorribile solo in un senso, aggiungete la freccia relativa.
- Eventualmente, specificate i vincoli cui va soggetta la relazione.

- Esaminate le associazioni da molti a molti: talora nascondono una classe del tipo "evento ricordato":

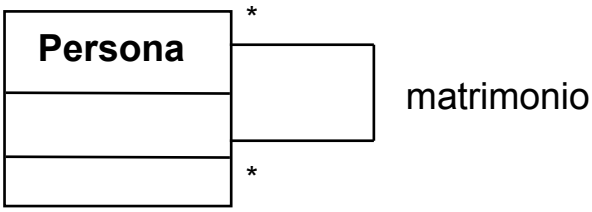


Relazione n-m: un Proprietario può possedere molti Veicoli, un Veicolo può essere di molti Proprietari (in tempi successivi e/o in comproprietà).



Ora un Proprietario può partecipare in più Compravendite, ma una Compravendita è relativa ad un solo Veicolo. Una relazione n-m resta, ma esprime correttamente la possiblità di comproprietà.

- Esaminate le associazioni tra oggetti di una stessa classe:



Relazione n-m (matrimonio) tra due oggetti della stessa classe.

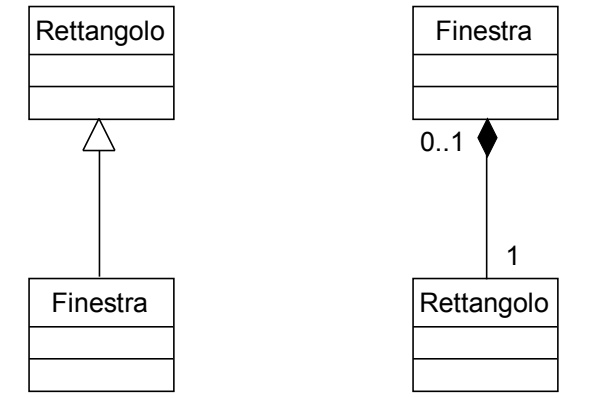
Un Matrimonio è in effetti una classe, con due associazioni con oggetti di tipo Persona (marito e moglie).

- Esaminate le associazioni multiple tra gli oggetti: talora anch'esse nascondono una classe del tipo "evento ricordato"
- Esaminate se un oggetto connesso (tra molti) ha un significato particolare (il più recente, quello approvato). In tal caso, aggiungete un attributo alla classe in questione (data, statoApprovazione, ecc.)

Ereditarietà e composizione

- Spesso una classe deve utilizzare le risorse di un'altra. Ad esempio, la classe Finestra è in corrispondenza con la classe Rettangolo per:
  - memorizzare la posizione relativa allo schermo;
  - fare calcoli di intersezione coi rettangoli di altre finestre, ecc...
- Per ottenere ciò, la prima soluzione è di rendere Finestra sottoclasse di Rettangolo. In tal modo:
  - una Finestra eredita e quindi ha accesso diretto a dati e operazioni di un Rettangolo.

- La seconda soluzione è di inserire un Rettangolo entro la struttura dati di una Finestra: una Finestra contiene un Rettangolo. In tal modo:
  - una Finestra ha accesso indiretto alle operazioni pubbliche di un Rettangolo.



Ereditarietà **Composizione**

- Caratteristiche della I soluzione (ereditarietà):
  - usare l'ereditarietà si chiama: riuso come *scatola bianca*;
  - la definizione avviene staticamente a livello di compilazione;
  - l'implementazione riusata è facile da modificare, potendosi ridefinire i suoi metodi continuando a riusare quelli originali;
  - la superclasse definisce parte della rappresentazione fisica della sottoclasse, legando a sè la sottoclasse, *rompendo l'incapsulamento* e rendendola più difficile da riusare

- Caratteristiche della II soluzione (composizione):
  - usare la composizione si chiama: riuso come *scatola nera*;
  - la definizione può avviene dinamicamente a "run time";
  - le interfacce delle due classi restano indipendenti, ed ogni classe può specializzarsi a fare ciò che deve, e nulla più.
- Quindi:

Preferite la composizione all'eredita-rietà. Usate l'ereditarietà solo se rispecchia il dominio del problema

Implementazione delle associazioni

- Uno dei principali compiti dell'OOD consiste nel definire come implementare le relazioni tra le classi del modello.
- Il modo più semplice di implementare le associazioni con cardinalità 0..1 oppure 1..1 consiste nell'aggiungere alla classe che contiene una variabile d'istanza che rappresenti l'oggetto della classe parte, o un puntatore ad esso.
  - Ad esempio, la relazione 1..1 tra le classi Automobile e Motore viene rappresentata aggiungendo alla classe Automobile una variabile d'istanza di classe Motore:

```
Motore motore;
```

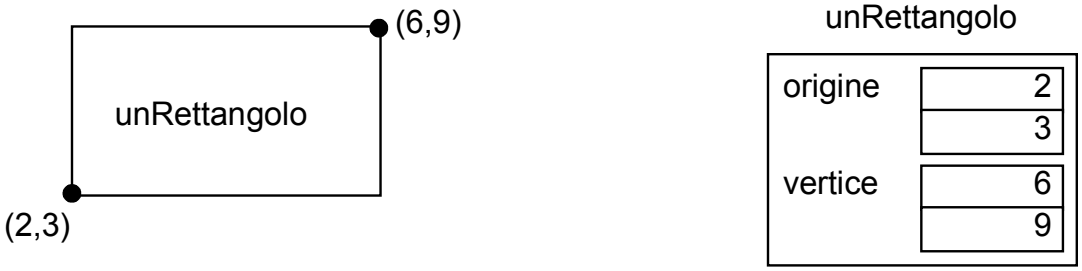
Classi contenitore

- Una *classe contenitore*, o contenitore, è una classe lo scopo delle cui istanze è quello di contenere oggetti di altre classi.
- Vettori, stack, liste concatenate, alberi sono tutti esempi di classi contenitore.
- Una classe contenitore fornisce la possibilità di tenere insieme un gruppo di oggetti, di aggiungerne e toglierne, e di *iterare* su di essi, cioè di eseguire la stessa operazione su tutti gli oggetti contenuti.
- Essa è usata per implementare le relazioni 0..*n* oppure 1..*n*
- Se *n* è fisso, e non è richiesto un particolare ordine, può bastare un vettore di oggetti.

- Se però gli oggetti contenuti sono in numero variabile, o hanno un ordine da mantenere, allora i vettori predefiniti non bastano ed occorre una classe contenitore.
- I contenitori si possono classificare a seconda del modo in cui contengono gli oggetti contenuti, e dell'omogeneità o eterogeneità di tali oggetti.
- Un oggetto può essere contenuto entro un contenitore *per valore* o *per riferimento*.

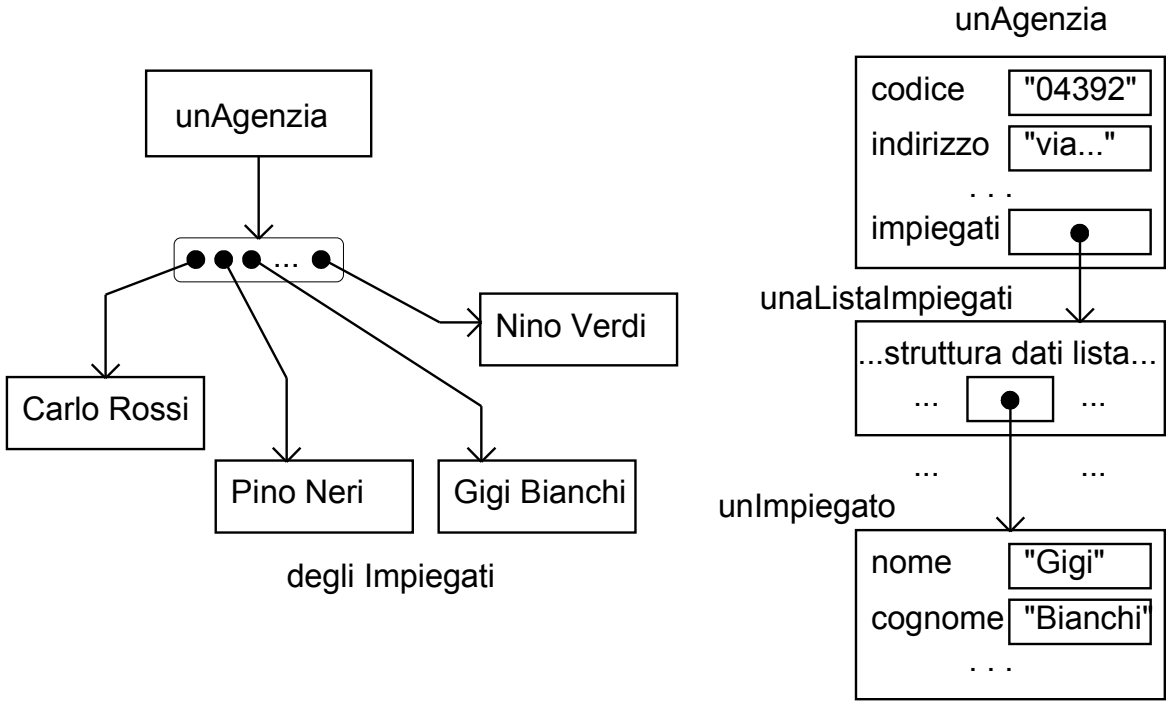
Contenimento per valore e per riferimento

- **Dal punto di vista implementativo**, il contenimento per valore implica:
  - l'oggetto contenuto sta nella struttura dati del contenitore (ciò è vero in C++), ed esiste solo in quanto contenuto;
  - quando un oggetto esistente è inserito nel contenitore, viene duplicato;
  - la distruzione del contenitore comporta la distruzione degli oggetti contenuti.



Un oggetto Rettangolo, che contiene per valore due Punti, origine e vertice. A destra è rappresentata la sua struttura dati (in C++).

- Il contenimento per riferimento implica che l'oggetto contenuto esiste per conto proprio:
  - l'oggetto contenuto può stare in più contenitori contemporaneamente;
  - quando un oggetto è inserito nel contenitore, non viene duplicato ma se ne registra solo l'indirizzo;
  - la distruzione del contenitore *non* comporta la distruzione degli oggetti contenuti;
  - può invece succedere che un oggetto contenuto sia cancellato ed il contenitore non ne sia notificato, con conseguenti possibili grossi problemi.



Un oggetto Agenzia, che contiene per riferimento un insieme di Impiegati.

Contenimento di oggetti omogenei ed eterogenei

- L'altra classificazione dei contenitori riguarda se contengono oggetti *omogenei*, cioè tutti dello stesso tipo, oppure *eterogenei*, cioè di tipo diverso.
- In C++, le classi generiche (template) sono ideali per implementare contenitori di oggetti omogenei, sia per valore che per riferimento.
- Il tipo degli oggetti contenuti può essere lasciato generico, e ci si può concentrare sugli algoritmi di gestione della collezione.
- Quando serve una classe contenitore di oggetti appartenenti a una classe specifica, basta *istanziare* la classe generica specificando il tipo in argomento.

- Nel caso invece di collezioni eterogenee di oggetti contenuti per riferimento, occorre usare l'ereditarietà, e con essa la proprietà che un puntatore alla superclasse radice della gerarchia può puntare ad un'istanza di una qualunque sottoclasse.
- A tali oggetti si possono inviare messaggi facenti uso del binding dinamico.

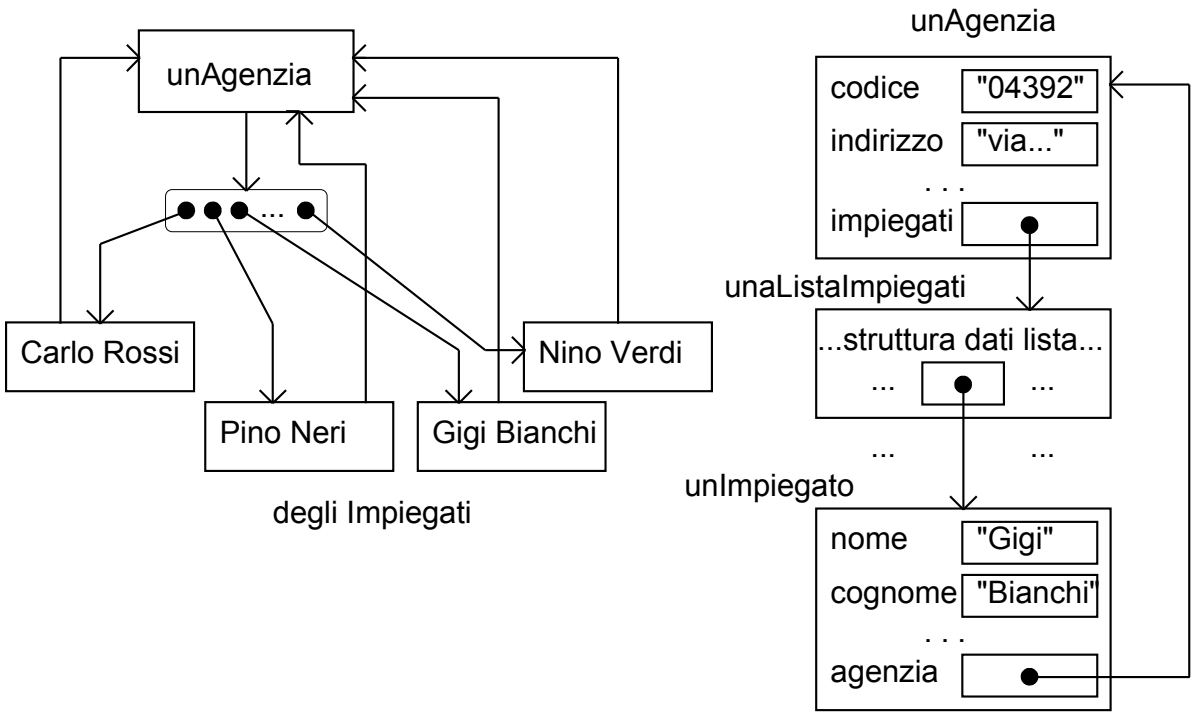
Contenimento	Oggetti omogenei	Oggetti eterogenei
Per valore	Classi generiche contenenti direttamente gli oggetti	Classi generiche con puntatori polimorfi agli oggetti in memoria dinamica e oggetti che conoscono la propria classe
Per riferimento	Classi generiche contenenti puntatori agli oggetti	Classi generiche con puntatori polimorfi e binding dinamico

- In Smalltalk, non essendovi tipizzazione, tutte le classi contenitore possono contenere oggetti eterogenei.
- In St, Java e C#, il contenimento avviene sempre tramite puntatori (non c'è nozione di variabile che non sia un puntatore).

Direzionalità delle associazioni

- La relazione contiene, a livello implementativo, può essere mono- o bi-direzionale.
- Si noti che, a livello di analisi, l'aggregazione ha una direzione precisa (è il tutto che contiene la parte, e non viceversa).
- La mono-direzionalità si ha quando un A contiene un B, ma non occorre che B possa accedere direttamente al suo contenitore A.
- Ciò di solito avviene quando al B si accede sempre tramite l'A.
- La bi-direzionalità si ha quando un A contiene un B, ma occorre anche sapere immediatamente, dato un B, qual è l'A che lo contiene.

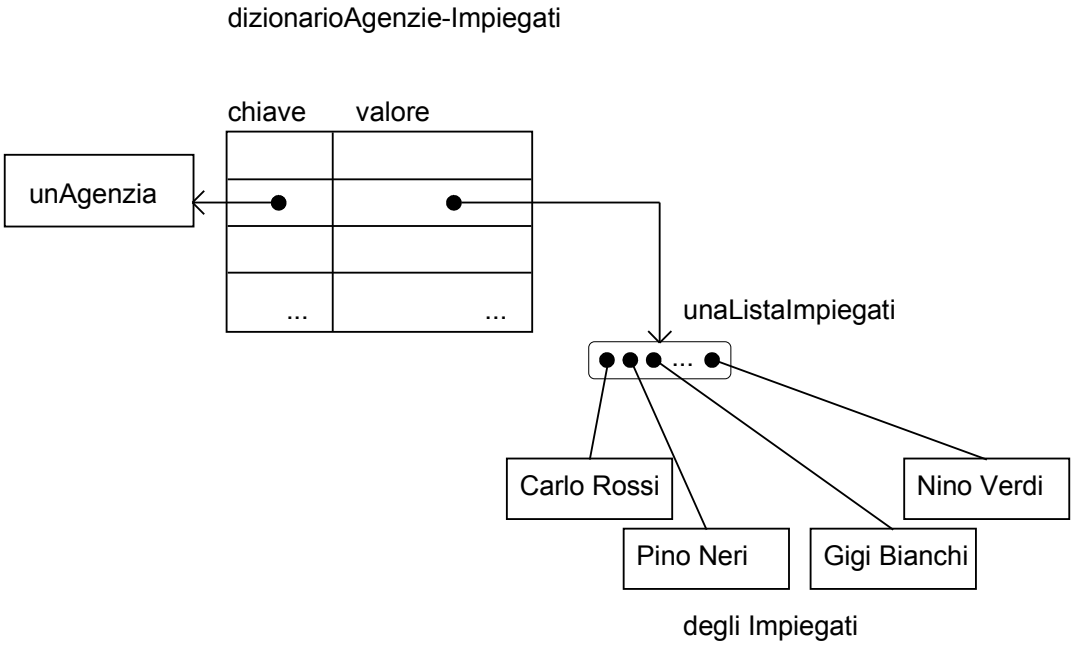
- Dal punto implementativo ciò si ottiene aggiungendo alla struttura dati di B un puntatore all'A che contiene il B stesso (o a una lista di A, se il contenimento è multiplo).
- Ciò porta a strutture dati con puntatori incrociati, molto efficienti ma di cui occorre tenere sotto controllo la consistenza.



Legame bi-direzionale tra un'Agenzia ed i suoi Impiegati.

Altri modi per implementare l'associazione

- Un modo alternativo per implementare un'associazione tra due oggetti è tramite un *dizionario*.
- Un dizionario è un tipo particolare di contenitore, che *associa* due oggetti: la *chiave* ed il rispettivo *valore*.
- La chiave deve essere unica, ed il dizionario, data una chiave, ne ritrova in modo efficiente il valore.
- Si noti che la chiave può essere un oggetto qualsiasi e non necessariamente una stringa o un numero.



Legame tra un'Agenzia ed i suoi Impiegati implementato tramite un dizionario.

Identificazione univoca degli oggetti

- Un oggetto può "contenere" degli altri oggetti, ma come è implementato tale contenimento? In altre parole, come si può identificare univocamente un oggetto per associarlo ad un altro?
- In caso di strutture dati interamente contenute in memoria, un oggetto si identifica tramite il proprio indirizzo in memoria.
- In caso di database o di sistemi distribuiti, ogni oggetto ha un *identificatore univoco* (generato automaticamente alla sua creazione) tramite il quale è possibile risalire all'oggetto stesso, sia che risieda in memoria, su disco o in rete.
- In alternativa, sempre in caso di database, si

può identificare un oggetto datane la classe e la chiave univoca:

- L'insieme delle istanze di una classe è detto *estensione* (extent) della classe.
- Negli OODBMS (ma anche in Smalltalk) è possibile accedere all'estensione ed effettuarvi delle interrogazioni.

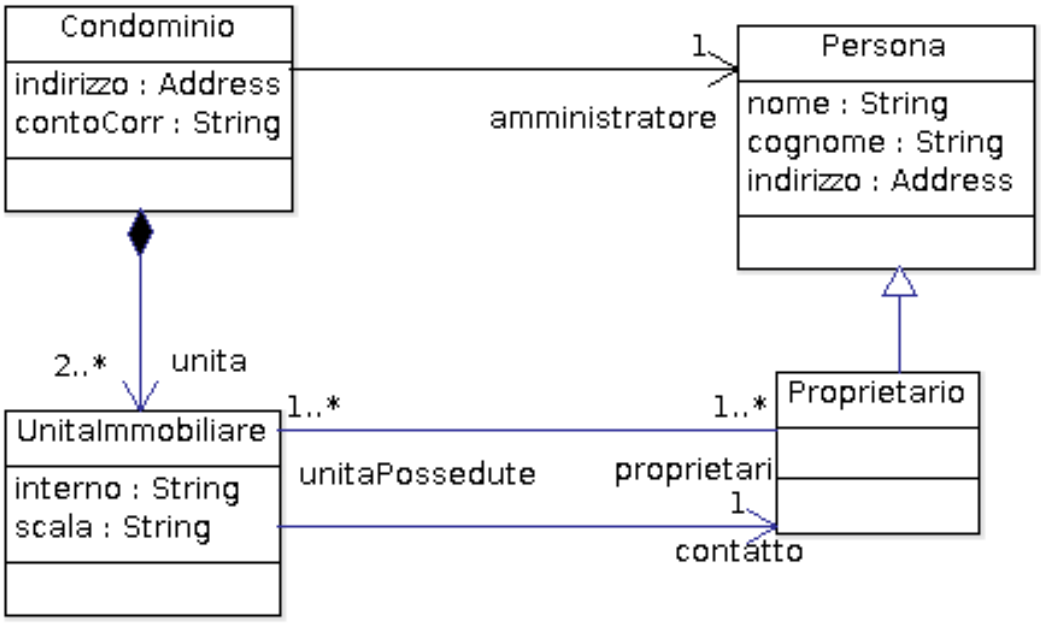
Dal diagramma delle classi alla struttura dati

- Un diagramma delle classi può essere "compilato" automaticamente in una struttura dati (e nella dichiarazione dei metodi) di un linguaggio OOPL
- Le informazioni necessarie sono:
  - Nomi delle classi e loro ereditarietà
  - Tipo, nome, accesso degli attributi
  - Ruoli e cardinalità (almeno nella direzione di navigabilità) delle associazioni (aggregazioni/composizioni)
  - Nome, accesso, tipo ritorno, nomi e tipi dei parametri delle operazioni
  - Valori di default di attributi e parametri
- Altre info possono essere necessarie per la generazione automatica degli *accessor*

Generazione automatica degli *accessor*

- Per le associazioni “a molti” (*, 1..*, 2..*,...) occorre conoscere:
  - Il tipo di collezione usata per implementare l'associazione
  - Che metodi vanno generati; ad es.:
    - void add<association>(anObject)
    - void delete<association>(anObject)
    - int size<association>()
    - boolean <association>includes(anObject)
    - void reset<association>()

Dal diagramma delle classi alla struttura dati: esempio



- Da questo diagramma UML delle classi, si possono generare le strutture dati delle stesse, e gli accessor
- Di seguito, un esempio in Java
- Le associazioni multiple sono implementate con una List<>
- Non sono generati i costruttori

La struttura dati in Java:

```
class Condominio {

private Address indirizzo;
private String scala;
private Persona amministratore;
private List<UnitaImmobiliare> unita;
    // Setters:
public void setIndirizzo(Address lIndirizzo);
public void setScala(String laScala);
public void setAmministratore(Persona lAmmin);

    // Getters:
public Address getIndirizzo();
public String getScala();
public Persona getAmministratore();
    // Collection Accessors:
public void addUnita(
             UnitaImmobiliare lUnitaImm);
public void removeUnita(
             UnitaImmobiliare lUnitaImm);
public int unitaSize();
public boolean unitaEPresente(
             UnitaImmobiliare lUnitaImm);}
```

```
class Persona {

private String nome;
private String cognome;
private Address indirizzo;
    // Setters:
public void setNome(String ilNome);
public void setCognome(String ilCognome);
public void setIndirizzo(Address lIndirizzo);
    // Getters:
public String getNome();
public String getCognome();
public Address getIndirizzo();}

class Proprietario extends Persona {

private List<UnitaImmobiliare> unitaPossedute;
    // Collection Accessors:
public void addUnitaPossedute (
             UnitaImmobiliare lUnitaImm);
public void removeUnitaPossedute(
             UnitaImmobiliare lUnitaImm);
public int unitaPosseduteSize();
public boolean unitaPosseduteEPresente(
             UnitaImmobiliare lUnitaImm);}
```

```
class UnitaImmobiliare {

private String interno;
private String scala;
private List<Proprietario> proprietari;
private Proprietario contatto;
   // Setters:
public void setInterno(String lInterno);
public void setScala(String laScala);
public void setContatto(
                    Proprietario ilContatto);
   // Getters:
public String getInterno();
public String getScala();
public Proprietario getContatto();

   // Collection Accessors:
public void addProprietari (
             Proprietario ilProprietario);
public void removeProprietari(
             Proprietario ilProprietario);
public int ProprietariSize();
public boolean ProprietariEPresente(
             Proprietario ilProprietario);}
```