Università degli Studi di Napoli "Federico II"

Ingegneria del Software

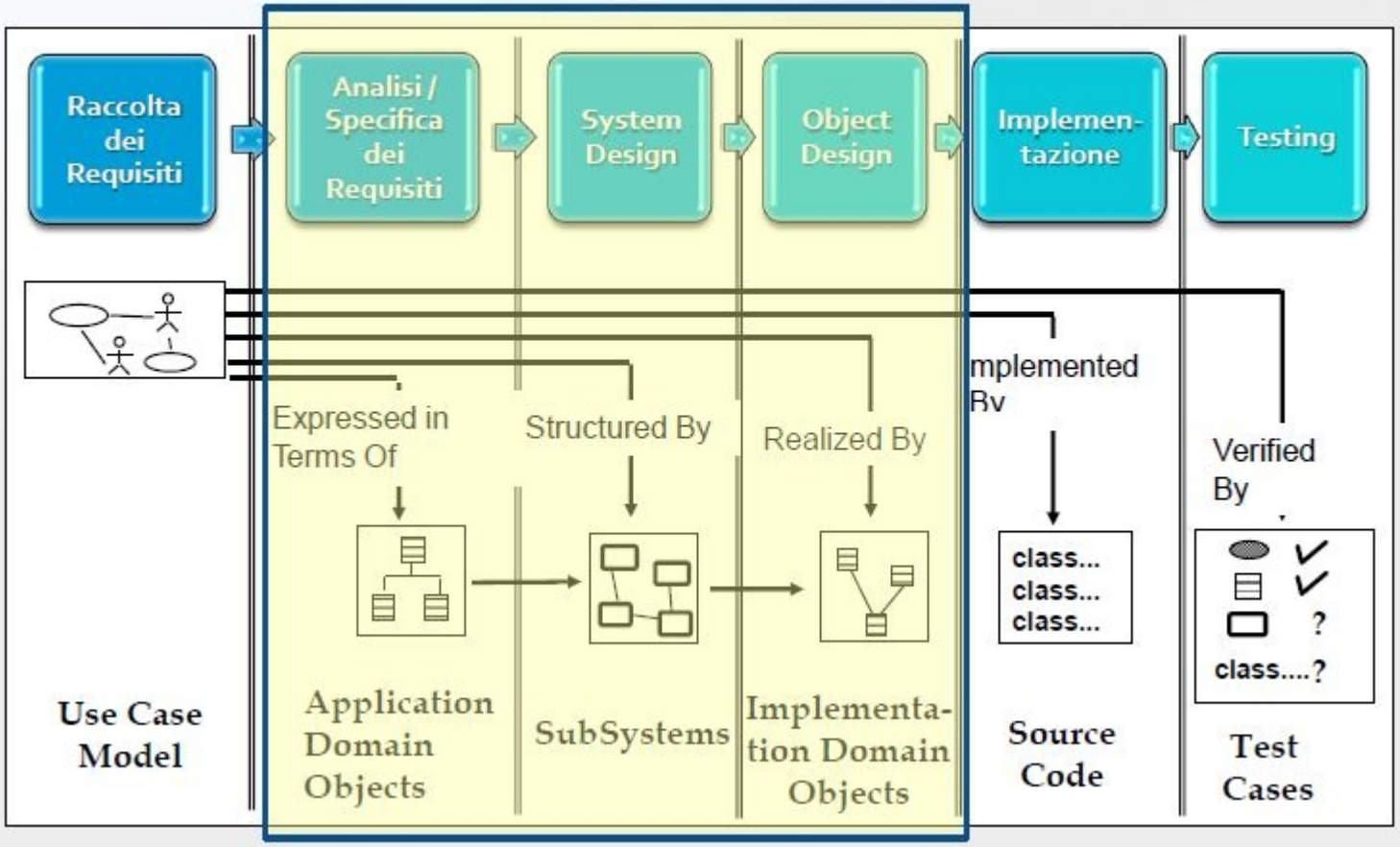
a.a. 2012/13

UML : I Class Diagrams

Obiettivi della lezione

- Comprendere in dettaglio il formalismo dei Class Diagram di UML
- Uso dei Class Diagram a diversi livelli di astrazione
  - Analisi dei requisiti
  - System design
  - Object design

Ciclo di vita del Software e Class Diagrams



Attività di Specifica dei Requisiti: dagli use case agli oggetti

- Le attività che consentono di trasformare gli use case e gli scenari della raccolta dei requisiti in un modello di analisi sono:
- **Analisi degli elementi statici (Class diagram):**
  - Identificare i concetti statici e classificarli secondo il modello **Entity, Boundary e Control**
    - Identificare gli Attributi
    - Identificare le associazioni
    - Modellare le associazioni di generalizzazione/specializzazione

Attività di Specifica dei Requisiti

- Le attività che consentono di trasformare gli use case e gli scenari della raccolta dei requisiti in un modello di analisi sono:
  - **Analisi degli aspetti comportamentali:**
    - Mappare gli Use Case e i diagrammi di Cockburn in Sequence Diagrams in modo consistente con l'analisi degli aspetti statici introdotta nei Class Diagram.
    - Modellare il Comportamento dipendente dallo stato degli Oggetti individuali mediane State Diagrams (Statecharts)
    - Modellare il flusso di controllo delle attività (Activity Diagram)
    - Rivedere il Modello di Analisi
- Queste attività sono guidate da euristiche
- La qualità dei risultati dipende dall'esperienza degli sviluppatori nell'applicare le euristiche e i metodi

Class Diagram

- Descrive:
  - Classi
  - Relazioni tra classi
    - associazione (uso)
    - generalizzazione (ereditarietà)
    - aggregazione (contenimento)
- Definisce la visione statica del sistema
- È il **modello principe** di UML, perché definisce gli elementi base del sistema sw da sviluppare.

Livello di Astrazione

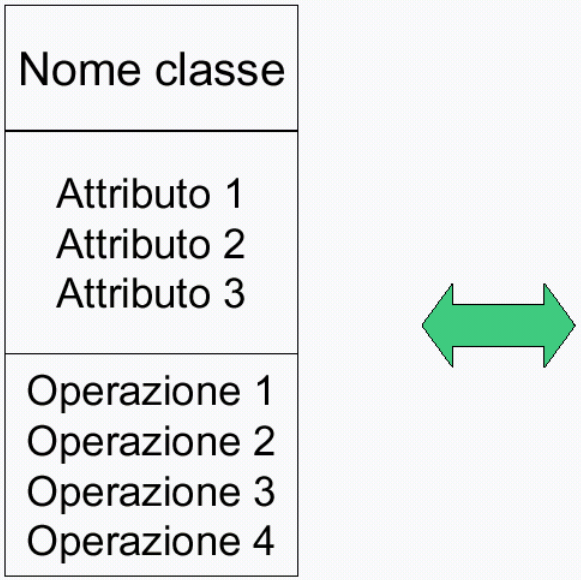
- Un class diagram può essere definito a livello:
  - **concettuale (o di Analisi)**
    - studia i concetti propri del dominio sotto studio, senza preoccuparsi della loro successiva implementazione
    - E' una sorta di **Dizionario Visuale** del dominio
  - di specifica implementativa (**Design**)
    - Rappresenta la struttura implementativa, specificando come va sviluppato il sistema
    - E' un raffinamento del precedente

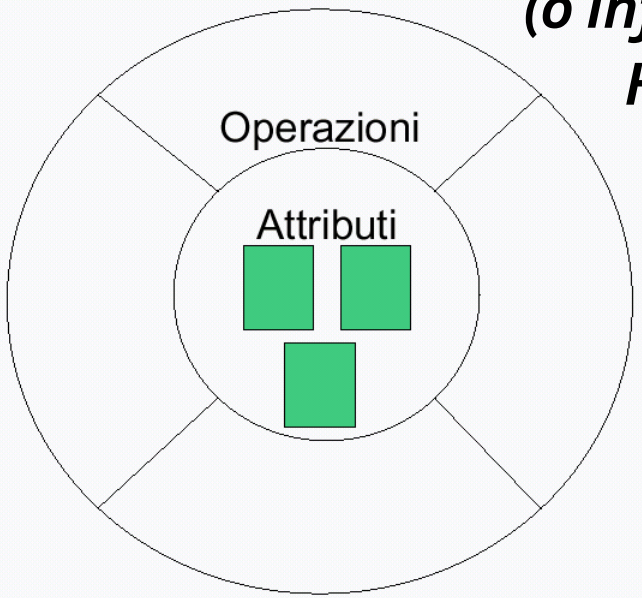
Classi e oggetti

- Un oggetto ha due tipi di proprietà:
  - **attributi** (o variabili di istanza), che descrivono lo stato
  - **operazioni** (o metodi), che descrivono il comportamento
- Una classe rappresenta un'astrazione di oggetti aventi la medesima struttura (tipo)
  - oggetti che hanno le stesse proprietà: attributi e operazioni
- Una classe definisce un tipo istanziabile.
- I termini oggetto e istanza (di classe) sono considerati nel corso interscambiabili
- Ogni oggetto è univocamente distinguibile mediante un identificativo (oid, object identifier)

Classi in UML

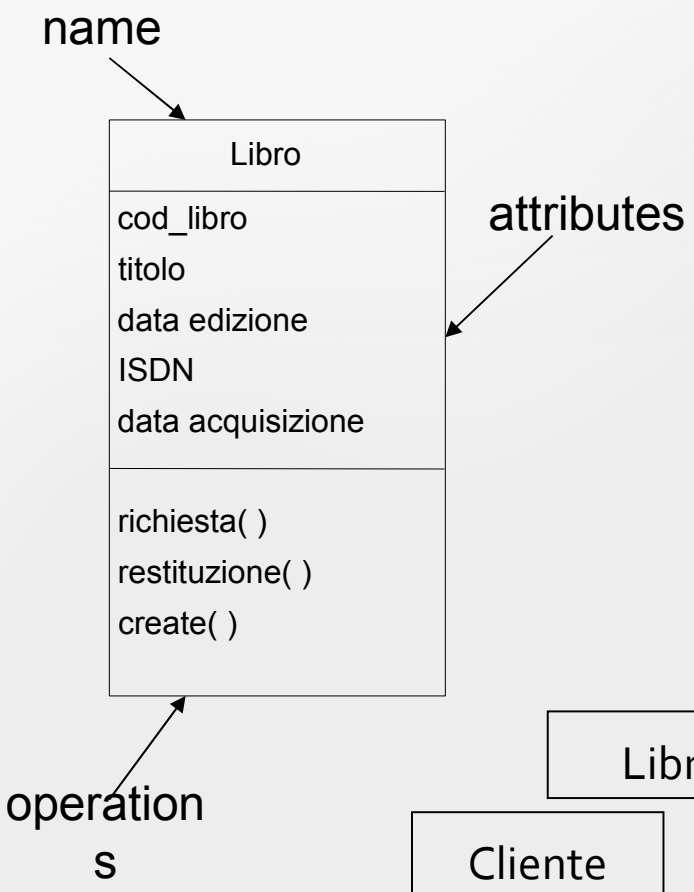
- In UML una classe è composta da tre parti
  1. nome
  2. attributi (lo stato)





Incapsulamento (o Information Hiding)

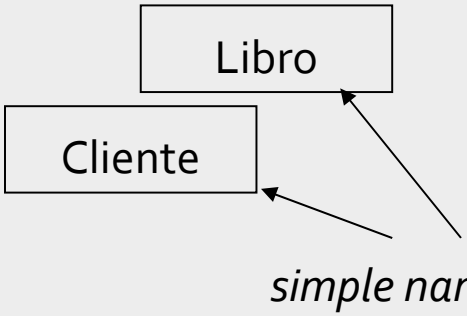
Nomi di Classi

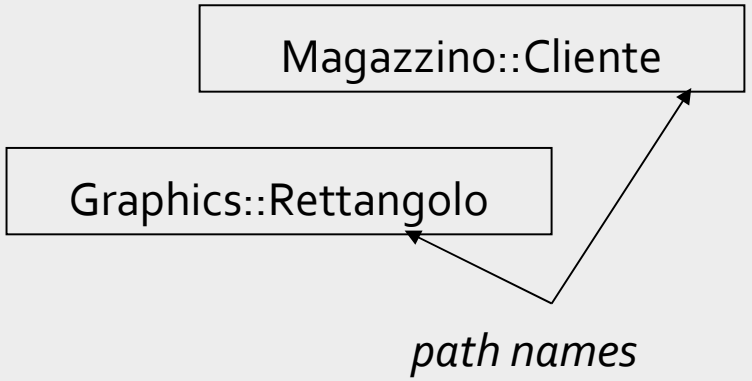


Una classe può essere rappresentata anche usando solo la sezione del nome

Un nome può essere un:

- *simple name*, il solo nome della classe
- *path name*, il nome della classe preceduto dal nome del package in cui la classe è posta



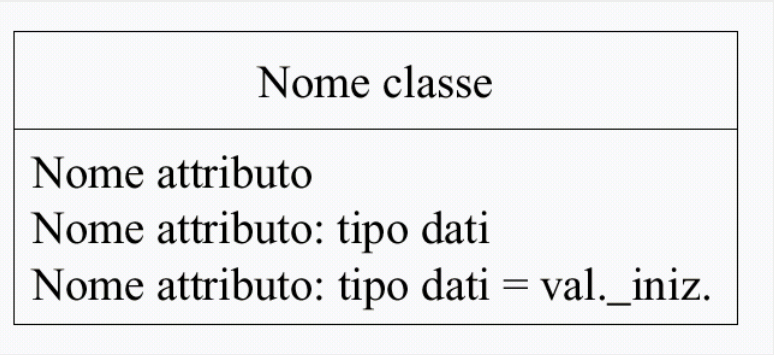


Attributi

- Un attributo è una proprietà di un oggetto
  - nome, età, peso sono attributi della classe Persona
  - colore, peso, anno, modello sono attributi della classe Auto
- Un attributo contiene un valore per ogni istanza
  - l'attributo età ha il valore "35" nell'istanza in esame della classe Persona.
- I nomi degli attributi devono essere unici all'interno di una classe
-

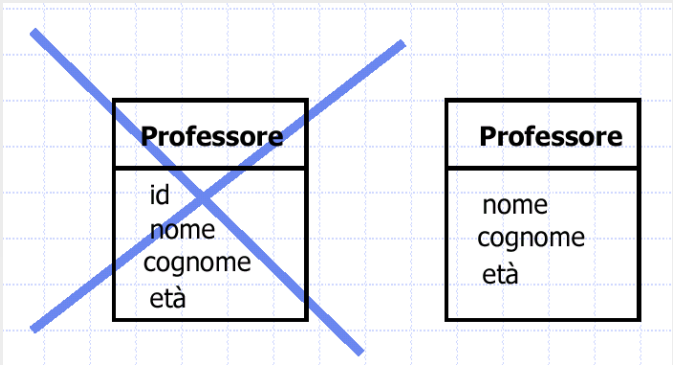
Attributi

- In UML, per ciascun attributo si può specificare il tipo ed un eventuale valore iniziale
- Tipicamente il nome di un attributo è composto da una o più parole
  - usare il maiuscolo per la prima lettera di ciascuna parola, lasciando minuscola la lettera iniziale del nome (Camel Case)



Individuazione degli attributi

- Gli oggetti hanno implicitamente una loro identità, (oid) non bisogna aggiungerla



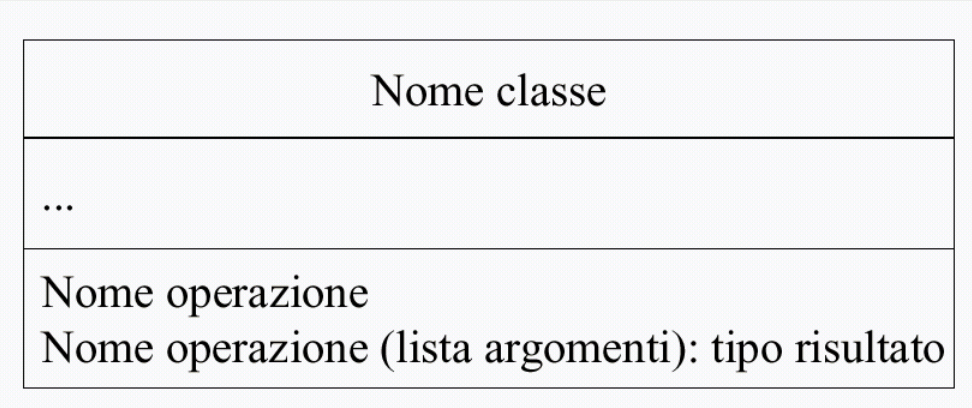
- Candidati per attributi
  - Sostantivi che non sono diventati classi
- Conoscenza del dominio applicativo
  - Persona (ambito bancario)
    - nome, cognome, codiceFiscale, numeroConto
  - Persona (ambito medico)
    - nome, cognome, allergie, peso, altezza

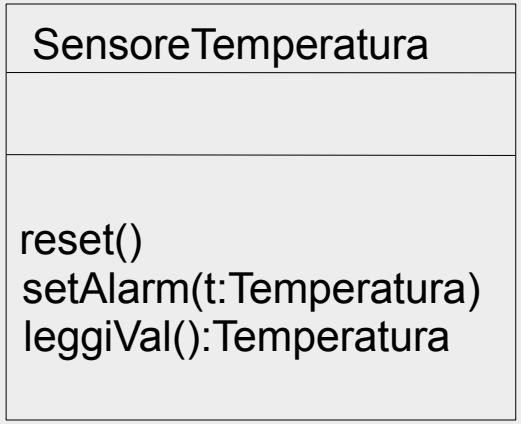
Operazioni

- Un'operazione è un'azione che un oggetto esegue su un altro oggetto e che determina una reazione
  - Le operazioni operano sui dati incapsulati dell'oggetto
- Tipi di operazione
  - ***Selettore*** (query): accedono allo stato dell'oggetto senza alterarlo (es. "lunghezza" della classe coda)
  - ***Modificatore***: alterano lo stato di un oggetto (es. "append" della classe coda)
- Operazioni di base per una classe di oggetti (realizzate con modalità diverse a seconda dei linguaggi)
  - *Costruttore*: crea un nuovo oggetto e/o inizializza il suo stato
  - *Distruttore*: distrugge un oggetto e/o libera il suo stato

Operazioni

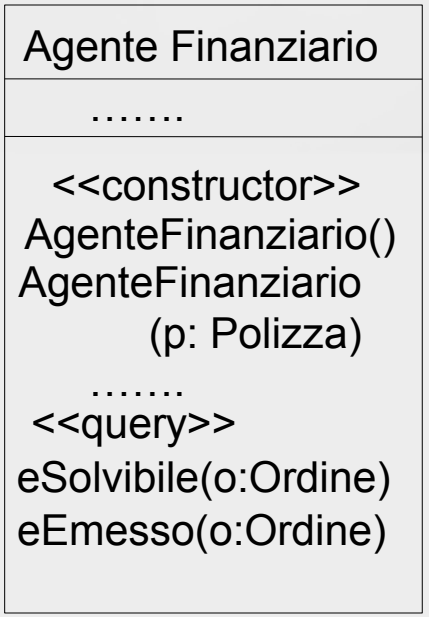
- Per ciascuna operation si può specificare il solo nome o la sua signature, indicando il nome, il tipo, parametri e, in caso di funzione, il tipo ritornato
- Stesse convenzioni dette per gli attributi (Camel Case)





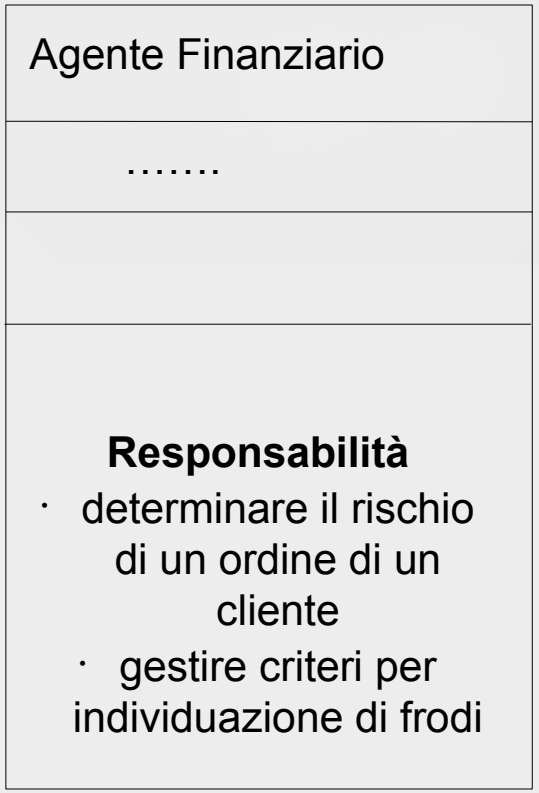
Attributi e Operazioni

- Attributi e Operazioni di una classe non devono obbligatoriamente essere descritti tutti subito
  - Attributi ed operazioni possono essere mostrati solo parzialmente, elidendo la classe
  - Per indicare che esistono più attributi/operazioni di quelli mostrati si usano i punti sospensivi " ……."
- Per meglio organizzare lunghe liste di operationi è possibile raggrupparle in categorie usando stereotipo quali
- **<<constructor>>**
- **<<query>>**
- **<<update>>**



Responsabilità

- Una **responsabilità** è un contratto o una obbligazione di una classe
  - Questa è definita dallo stato e comportamento della classe
  - Una classe può avere un qualsiasi numero di responsabiltà, ma una classe ben strutturata ha una o poche responsabiltità
- Le responsibilità possono essere indicate, in maniera testuale, in una ulteriore sezione, sul fondo della icona della class
  - Possono anche essere specificate in linguaggi formali, come OCL (Object Constraint Language)



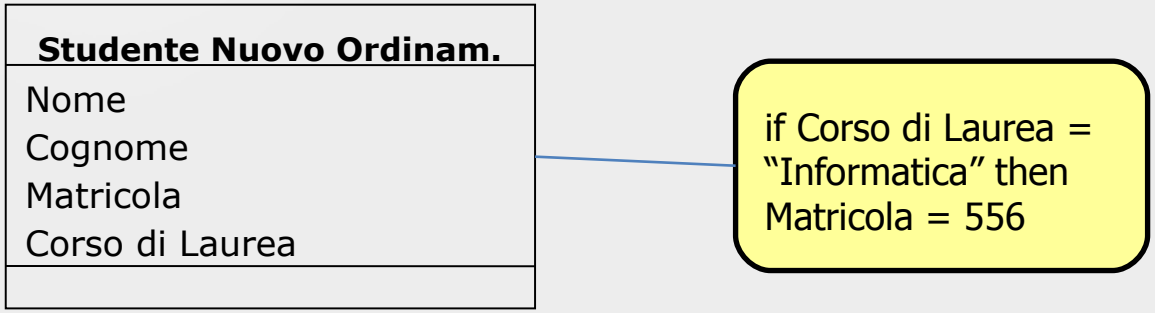
Visibilità

- E' possibile specificare la visibilità di attributi e operazioni
- In UML è possibile specificare tre livelli di visibilità:
  - + (**public**): qualsiasi altra classe con visibilità alla classe data può usare l'attributo/operazione (di default se nessun simbolo è indicato)
  - **#** (**protected**): qualsiasi classe discendente della classe data può usare l'attributo/operazione
  - - (**private**): solo la classe data può usare l'attributo/operazione

Vincoli aggiuntivi

- Talvolta è necessario o conveniente esplicitare dei vincoli aggiuntivi in modo da rendere più comprensibile il diagramma delle classi.

{*breve descrizione in linguaggio naturale o in linguaggi formali (come OCL)*}



Stringhe di Proprietà

- E' possibile definire una "stringa di proprietà" per specificare particolari caratteristiche delle features di una classe
- Si indica tra parentesi graffe affianco alla definizione della feature
- Per gli **attributi**, property-string può assumere uno dei seguenti 3 valori:
  - **changeable**: nessuna limitazione per la modifica del valore dell'attributo (default)
  - **addOnly**: per attributi con molteplicità maggiore di 1 possono essere aggiunti ulteriori valori, ma una volta creato un valore non può essere né rimosso né modificato
  - **readonly**: il valore non può essere modificato.
    - **frozen** (deprecato in UML2)

Proprietà di features

- Per i metodi, property-string può assumere uno dei seguenti valori:
  - **isQuery**: l'esecuzione dell'operazione lascia lo stato del sistema immutato
  - **sequential**: i chiamanti devono coordinare l'oggetto dall'esterno in modo che vi sia un sol flusso per volta verso l'oggetto; in presenza di più flussi di controllo, non è garantita la semantica e l'integrità dell'oggetto
  - **guarded**: la semantica e l'integrità dell'oggetto è garantita in presenza di flussi di controllo multipli dalla sequenzializzazione di tutte le chiamate a tutte le operation guarded dell'oggetto;
  - **concurrent**: la semantica e l'integrità dell'oggetto è garantita in presenza di flussi di controllo multipli trattando la operation come atomica. Chiamate multiple da flussi di controllo concorrente possono presentarsi contemporaneamente ad un oggetto o ad una sua operation concurrent ed essere eseguite correttamente con corretta semantica
- Le ultime tre proprietà riguardano la concorrenza di una operation e sono rilevanti solo in presenza di più processi o threads

Sintassi riepilogativa

- Attributi:

  `visibilità nome: tipo molteplicità = default {property-string}`

  - Es: `- titolo : String [1] = "Sw Engineering" {ReadOnly}`

- Metodi:

  `visibilità nome (elenco parametri): tipo di ritorno{property-string}`

  - Es: `+ getSaldo(data: Date): double {isQuery}`

Relazioni tra classi

Relazioni tra classi

- In UML, le interconnessini tra oggetti/classi sono rappresentate attraverso “relazioni”

- UML definisce 3 tipi di relazioni:
  1. Associazioni
  2. Generalizzazioni/Specializzazioni
  3. Dipendenze

Associazioni

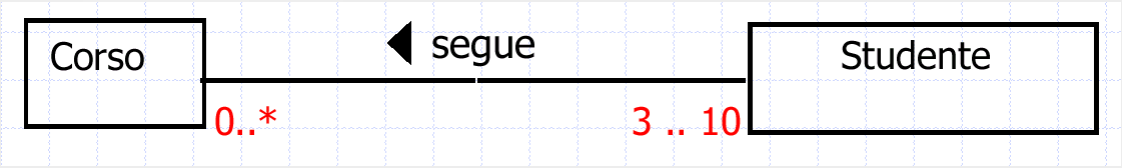
- Un sistema OO è costituito da classi che collaborano tra loro scambiandosi messaggi
- Quando è in esecuzione un sistema OO è popolato da istanze di classi
- Quando un'istanza di classe passa messaggi a un'altra classe si sottintende l'esistenza di **un'associazione** tra le due classi

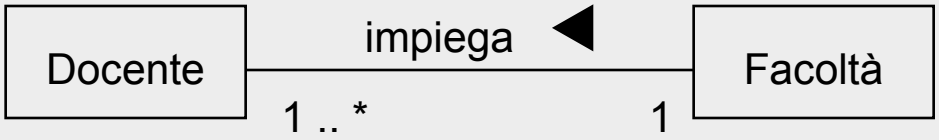
Associazioni

- Indicano relazioni tra classi
- Possono essere Riflessive
- Possono essere n-arie (raro)
- Adornments:
  - **Nome**: non obbligatorio; può avere un verso di lettura;niente a che vedere con la "navigabilità", che pure esiste in UML
  - **Ruolo:** per ciascuna classe coinvolta
  - **Cardinalità**: come negli E-R
- Aggregazioni: un tipo particolare di associazione che specifica un rapporto aggregato-componente
- Composizioni: aggregazione in cui ogni componente partecipa ad un solo aggregato

Molteplicità delle associazioni

- La molteplicità indica
  - Se la partecipazione all'associazione è obbligatoria oppure facoltativa
  - Il numero minimo e massimo di oggetti di una classe che possono essere relazionati ad un oggetto dell'altra classe





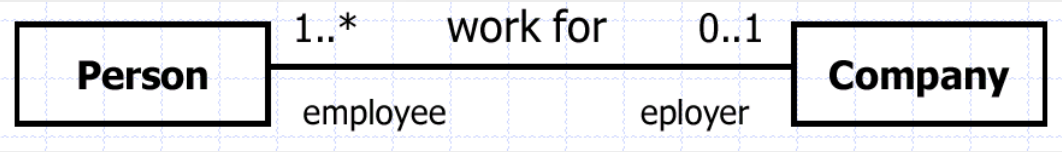
Molteplicità delle associazioni

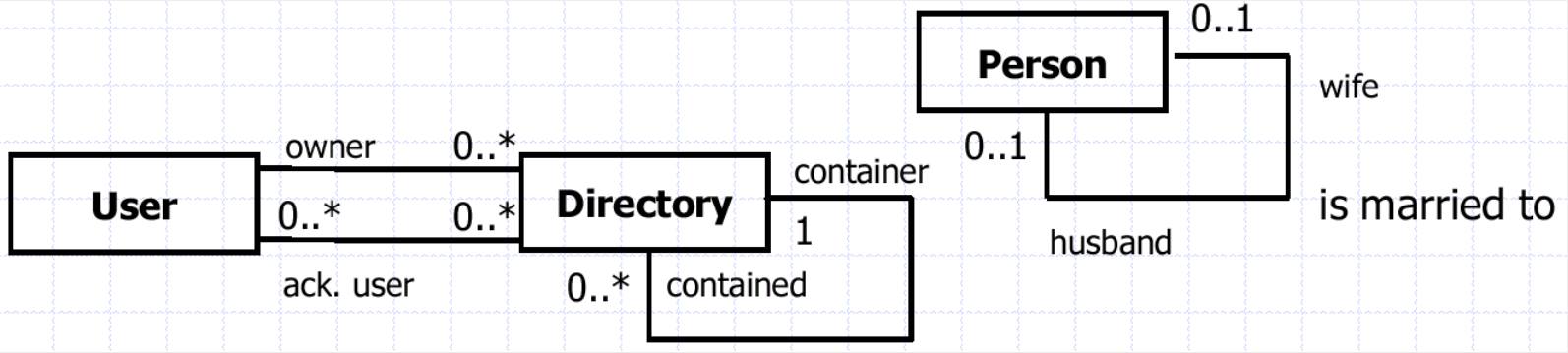
Indicator	Meaning
0..1	Zero o uno
1	Esattamente uno
0..*	Zero o più
1..*	One o più
n	*n* (con *n* > 1)
0..n	Da Zero a *n* (con *n* > 1)
1..n	Da uno a *n* (con *n* > 1)

Nota: Uml 1.x permetteva anche m..n, con m e n > 1

Ruoli

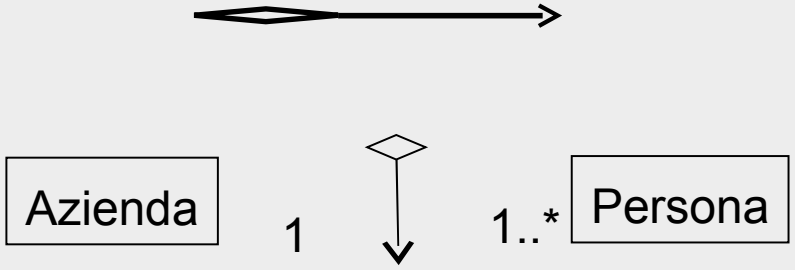
- I ruoli forniscono una modalità per attraversare relazioni da una classe ad un'altra
  - Chiariscono il ruolo giocato da una classe all'interno di un'associazione
  - I nomi di ruolo possono essere usati in alternativa ai nomi delle associazioni
- I ruoli sono spesso usati per relazioni tra oggetti della stessa classe (associazioni riflessive)





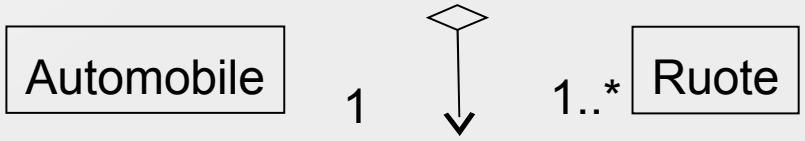
Aggregazione

- Aggregazione: è la relazione "parte di"
  - Es: un'azienda ha delle persone che vi lavorano. I vari oggetti "persona" continuano ad avere dignità ed esistenza propria anche al di là dell'oggetto azienda.
- La distruzione dell'oggetto "azienda" non comporta automaticamente quella delle sue parti



Aggregazioni

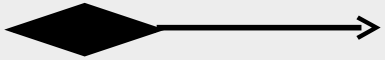
- La relazione di aggregazione è un'associazione speciale che aggrega gli oggetti di una classe componente in un unico oggetto della classe
  - la si può leggere come "è fatto da" in un verso e "è parte di" nell'altro verso

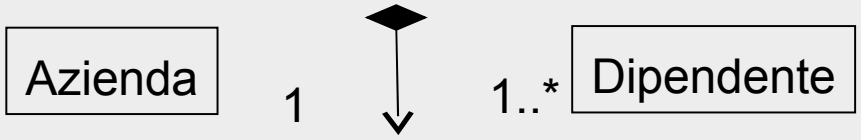


- Proprietà
  - transitività: se A è parte di B e B è parte di C allora A è parte di C
  - antisimmetria: se A è parte di B allora B non è parte di A
  - Indipendenza: un oggetto contenuto può sopravvivere senza l'oggetto contenente

Composizione

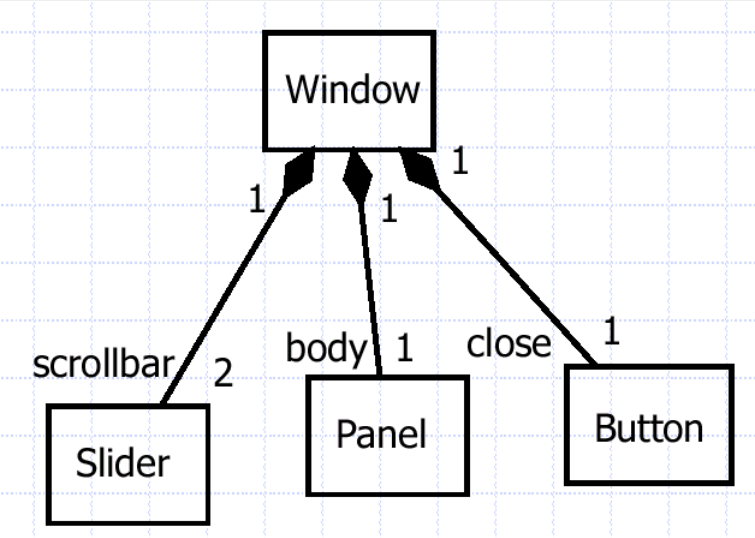
- Composizione: è una relazione più forte, l'oggetto parte appartiene ad un solo tutto e le parti hanno lo stesso ciclo di vita dell'insieme.
  - All'atto della distruzione dell'oggetto principale si ha la propagazione della distruzione agli oggetti parte.
  - Es: un'azienda ha dei dipendenti che vi lavorano. In tal caso, a differenza dell'esempio precedente, non ha senso che i vari oggetti "dipendente" abbiano vita propria senza un oggetto "azienda" associato



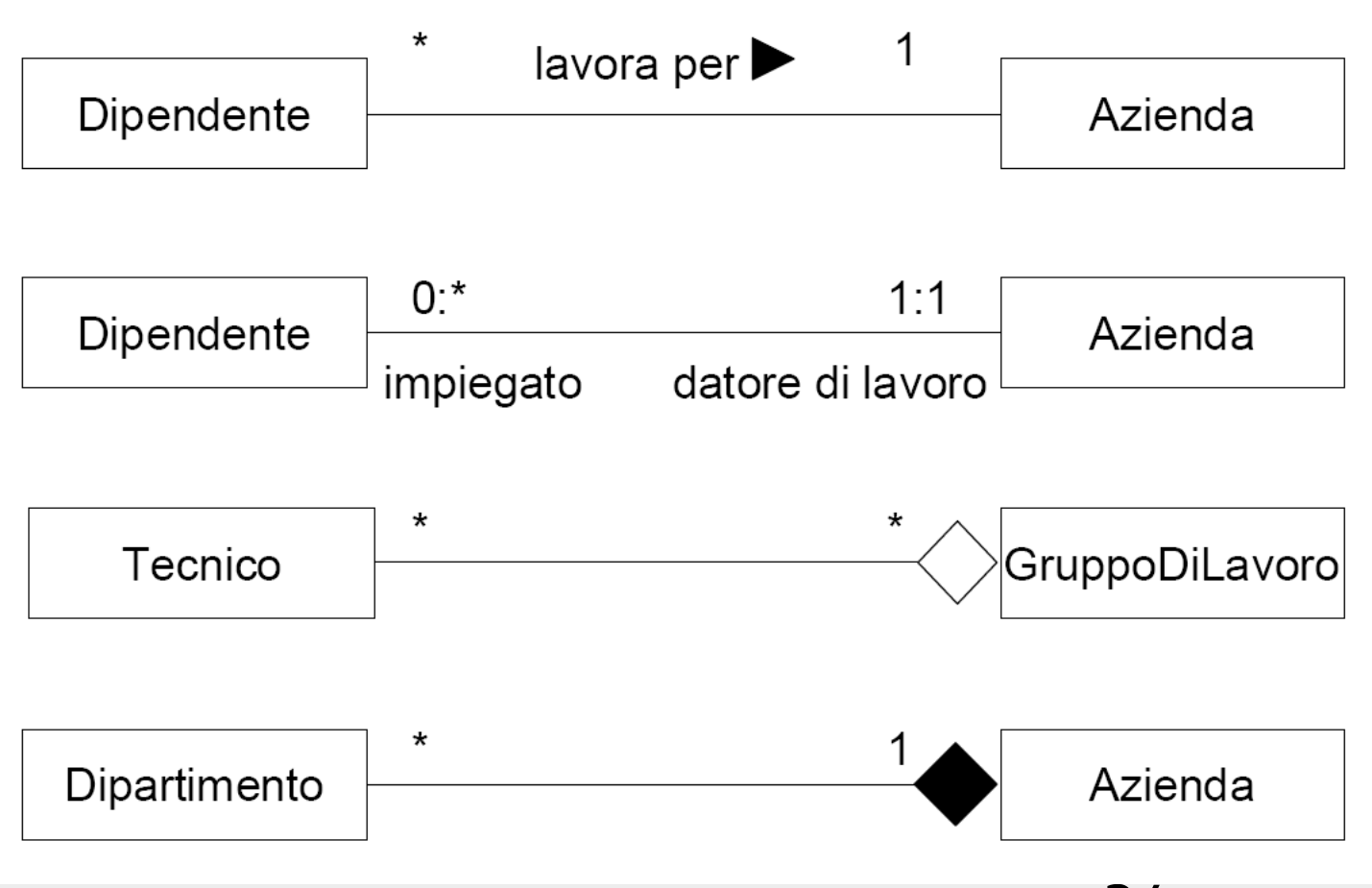


Composizione

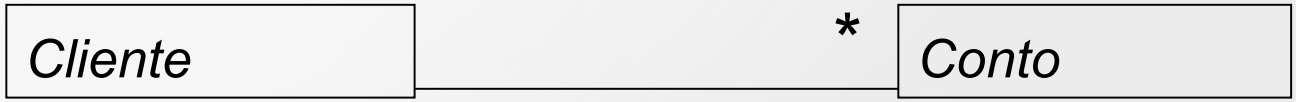
- Una relazione di composizione è un'aggregazione forte
  - Le parti componenti non esistono senza il contenitore
  - Ciascuna parte componente ha la stessa durata di vita del contenitore
  - Una parte può appartenere ad un solo tutto per volta



Riassunto Associazioni



Associazioni many-to-many: esempio

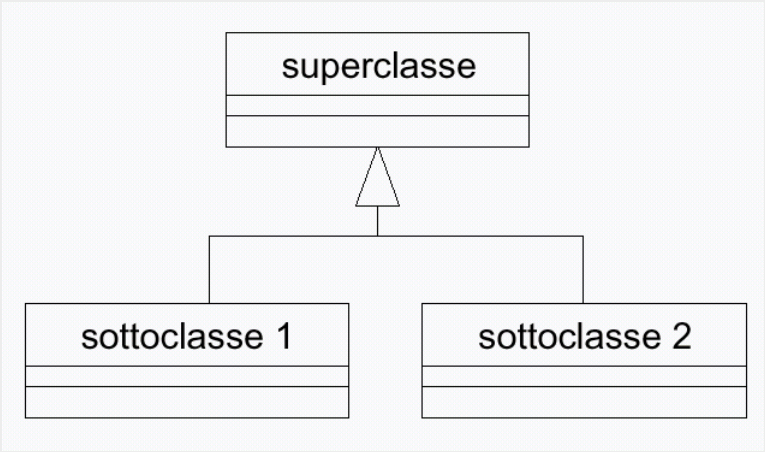


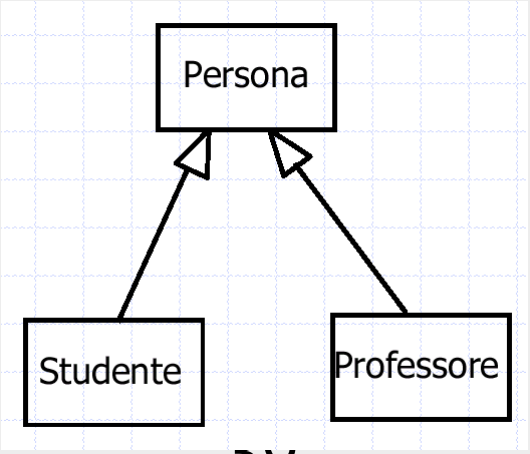

```
public class Cliente {
        private List conti;
        public Cliente() {
          conti = new ArrayList();
        }
        public void addConto (Conto c) {
                if (!conti.contains(c)) {
                        conti.add(c);
                        c.addCliente(this);  }
        }
```

```
public class Conto {
        private List clienti;
        public Conto() {
          clienti = new ArrayList();
        }
        public void addCliente (Cliente c) {
                if (!clienti.contains(c) {
                        clienti.add(c);
                        c.addConto(this);   }
        }
}
```

Ereditarietà

- La relazione di generalizzazione rappresenta una tassonomia delle classi
  - la classe generale è detta superclasse
  - ogni classe specializzata è detta sottoclasse
- Può essere letta come
  - "e' un tipo di" (verso di generalizzazione)
  - "può essere un" (verso di specializzazione)
- Ogni oggetto di una sottoclasse è anche un oggetto della sua superclasse





Ereditarietà

- L'ereditarietà è un meccanismo di condivisione delle proprietà degli oggetti in una gerarchia di generalizzazione
- Tutte le features di una superclasse sono ereditati dalle sottoclassi
  - generalizzazione ed ereditarietà godono della proprietà transitiva
  - E' possibile definire nuove proprietà per le sottoclassi
  - E' possibile ridefinire le proprietà ereditate (overriding) polimorfismo
- Anche le relazioni di una superclasse valgono per le sottoclassi

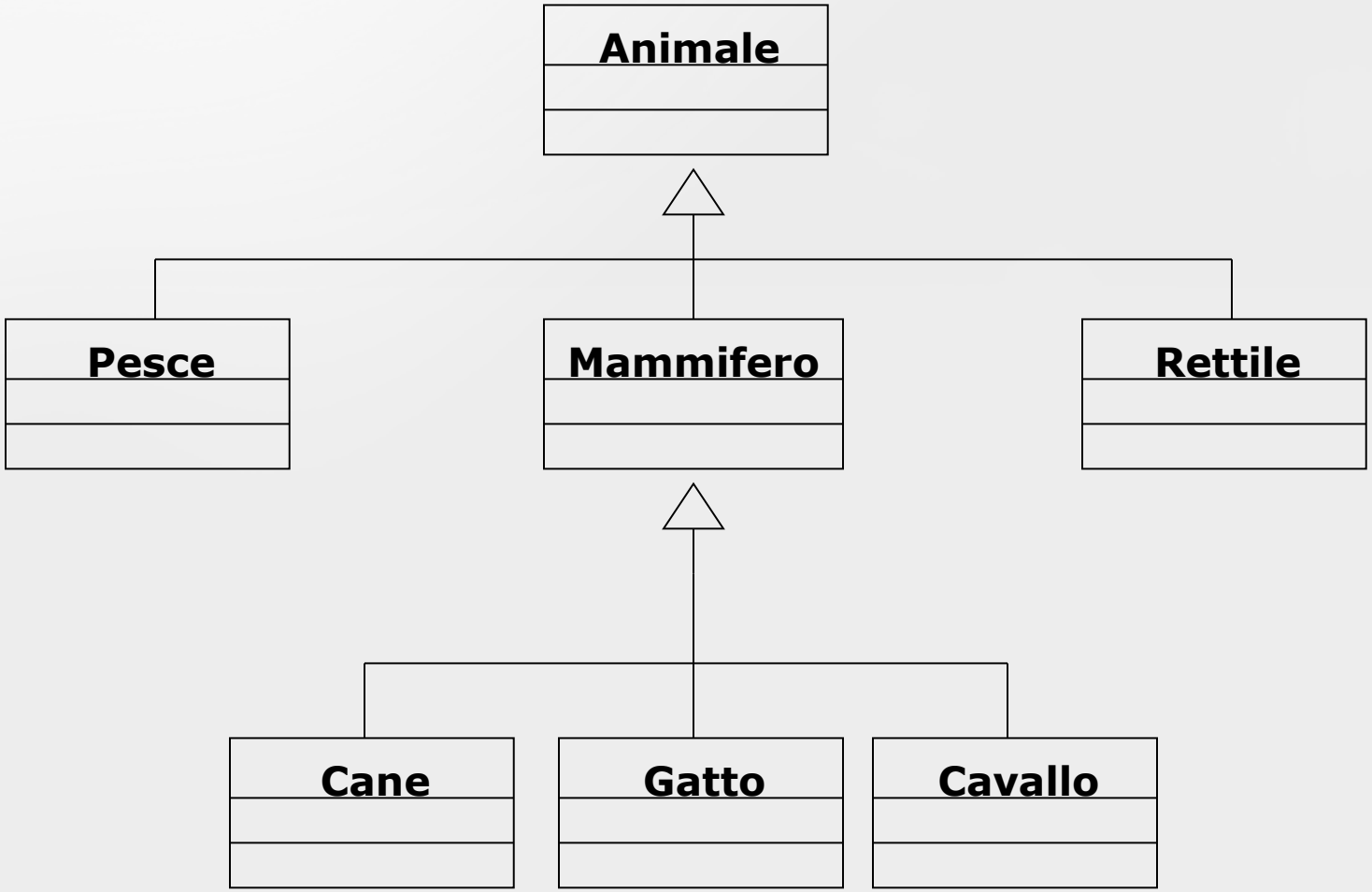
Generalizzazione

- L'Ereditarietà consente di organizzare concetti in gerarchie:
  - Al top della gerarchia concetti più generali
  - Al bottom concetti più specializzati
- Generalizzazione: attività di modellazione che identifica concetti astratti da quelli di più basso livello
  - Es. Stiamo facendo reverse-engineering di un sistema di gestione delle emergenze e analizzando le videate per la gestione di incidenti autostradali e incendi. Osservando concetti comuni, creiamo un concetto astratto Emergenze

Specializzazione

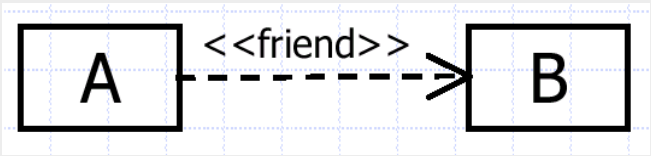
- Specializzazione: attività che identifica concetti più specifici da quelli di più ad alto livello
  - Es. Stiamo costruendo un sistema di gestione delle emergenze e stiamo discutendo le funzionalità con il cliente: il cliente introduce prima il concetto di incidente, quindi descrive tre tipi di incidenti: disastri, emergenze, incidenti a bassa priorità
- Come risultato sia della specializzazione che della generalizzazione abbiamo la specifica di ereditarietà tra concetti

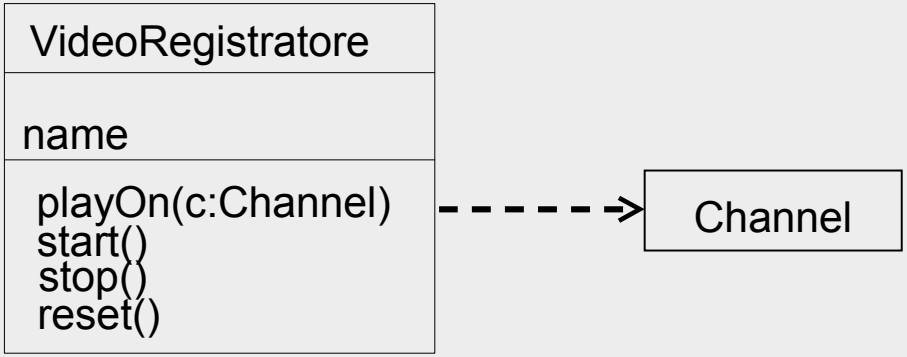
Generalizzazione fra classi (es.)



Dipendenze

- Relazione semantica in cui un cambiamento sulla classe indipendente può influenzare la classe dipendente
  - La freccia punta verso la classe **indipendente**
  - **E' possibile aggiungere un verbo che esplicita la dipendenza**





Dipendenze principali

Keyword	Significato
<<call>>	La sorgente invoca un metodo della destinazione
<<create>>	La sorgente crea un'istanza della classe destinazione
<<refine>>	Raffinamento tra classi. Ad es., la sorgente potrebbe essere una classe di analisi, la destinazione di design
<<use>>	La sorgente utilizza la destinazione

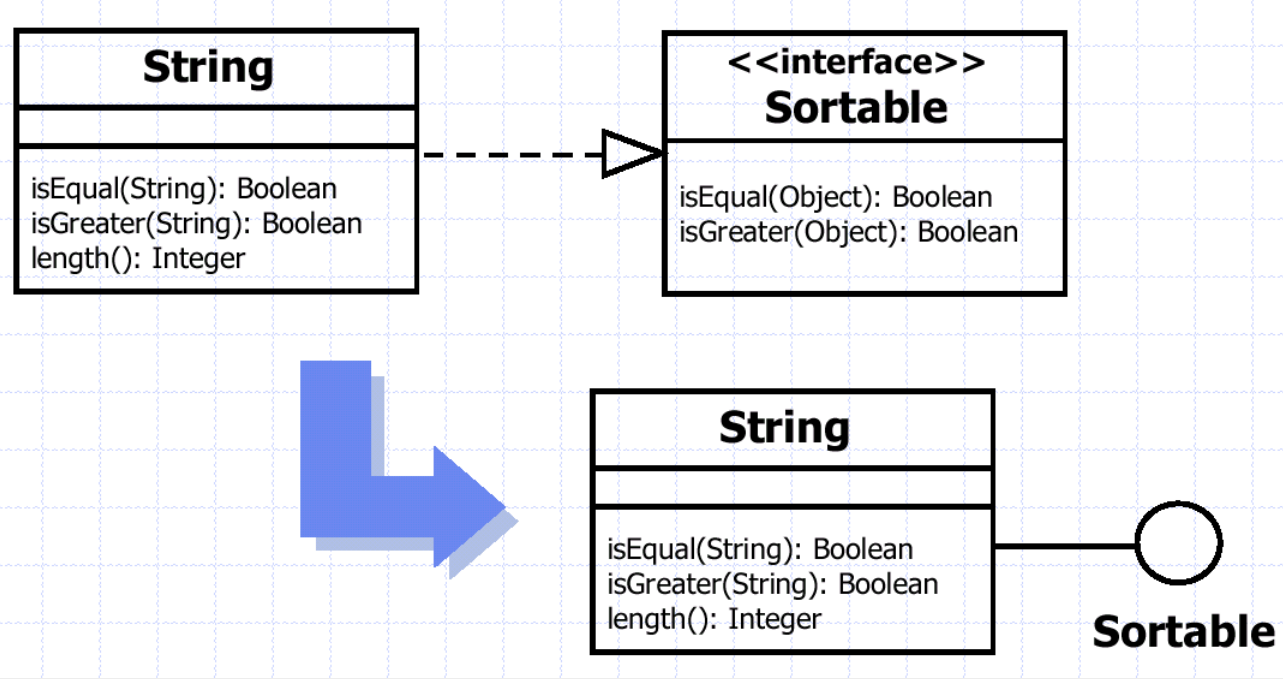
Altri Elementi dei Class Diagrams

Interfacce e realizzazioni

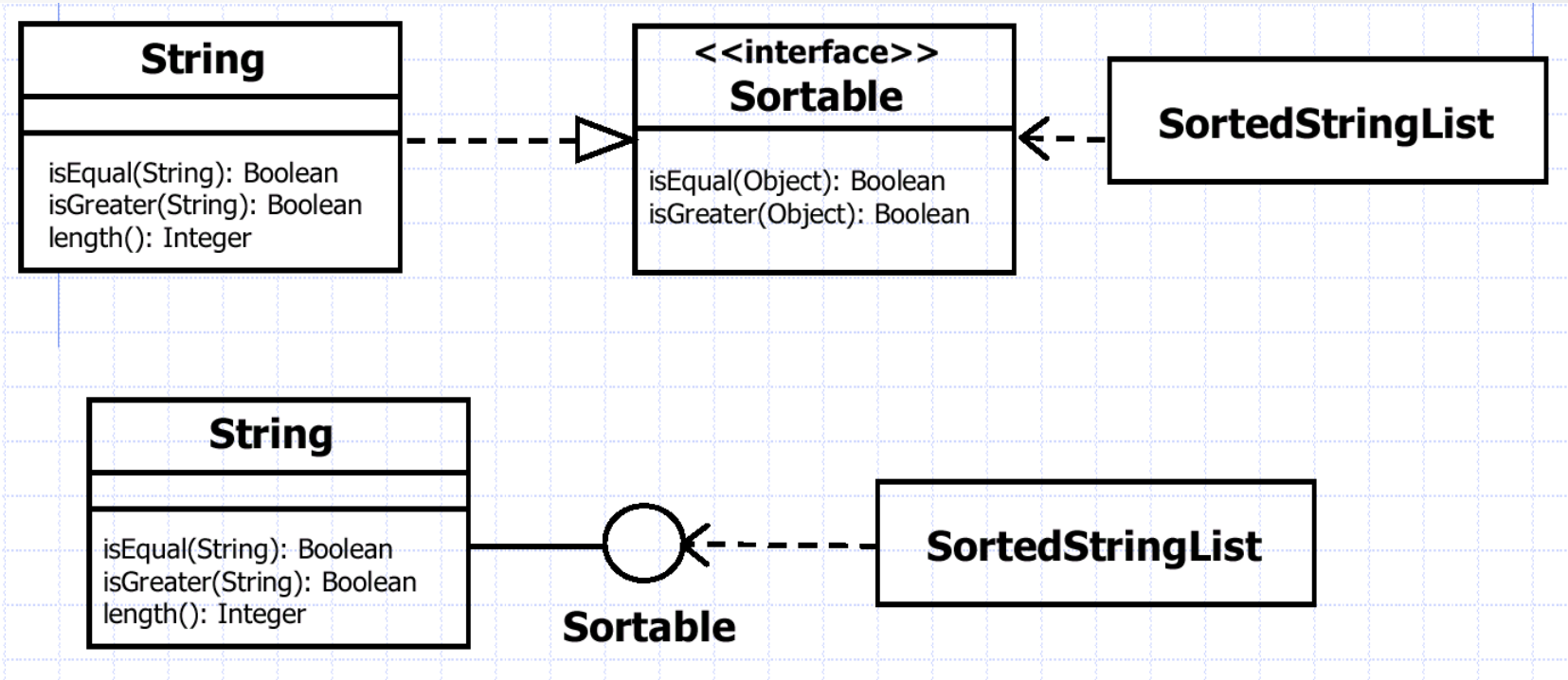
- Un'interfaccia viene modellata allo stesso modo in cui viene modellato il comportamento una classe e rappresenta un insieme di operazioni che una classe offre ad altre classi
  - Un'interfaccia non ha attributi ma soltanto operazioni (metodi). In UML per rappresentare le interfacce si utilizza un rettangolo con lo stereotipo «interface» o un piccolo cerchio (notazione **lollipop**, "a lecca-lecca").
- La relazione tra una classe ed un'interfaccia viene definita **realizzazione**.
  - Linea tratteggiata con un triangolo largo aperto costruito sul lato dell'interfaccia o con una linea nella notazione compatta lollipop.

Interfaccia (Definizione)

- Specifica il comportamento di una classe senza darne l'implementazione

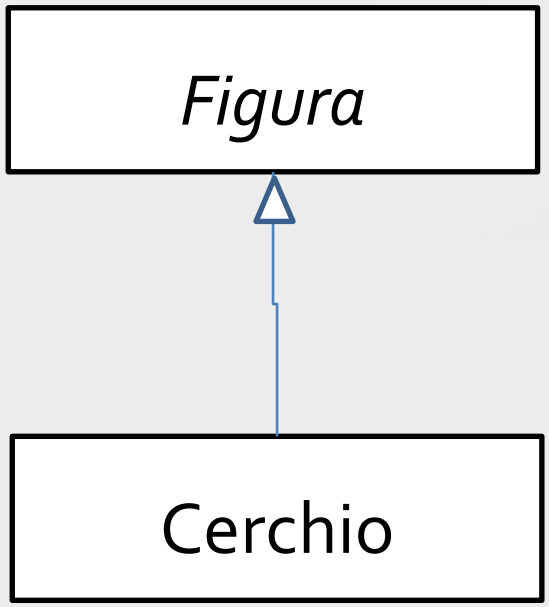


Interfaccia (Uso)



Classi astratte

- Una classe astratta definisce un comportamento "generico".
- Definisce e può implementare parzialmente il comportamento, ma molto della classe è lasciato indefinito e non implementato.
  - Dettagli specifici sono demandati alle sottoclassi specializzanti
- **Le classi astratte sono indicate ponendo il nome in corsivo**



Classi parametriche

- Molti linguaggi supportano il concetto di classe parametrica (o template), ossia consentono di definire classi che dipendono da un parametro da specificare.
  - Ciò è utile per lavorare con collezioni di elementi in linguaggi fortemente tipizzati, cioè con controlli rigorosi sui tipi

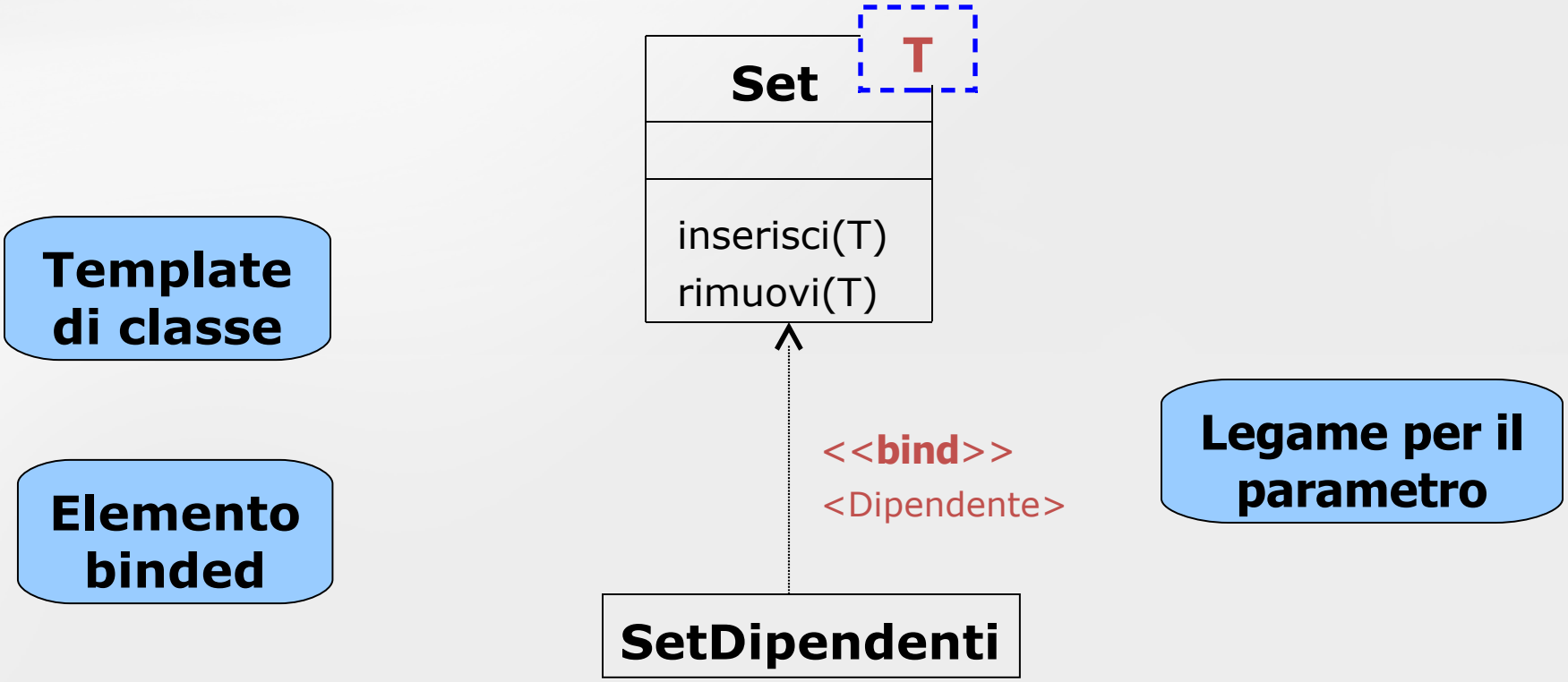
```
class Set <T> {

  void inserisci(T nuovoElem)

  void rimuovi(T unElem)

}
```

Parametro

- per poi creare insiemi di elementi specifici

```
Set <Dipendente> SetDipendenti
```

Classi parametriche (es.)



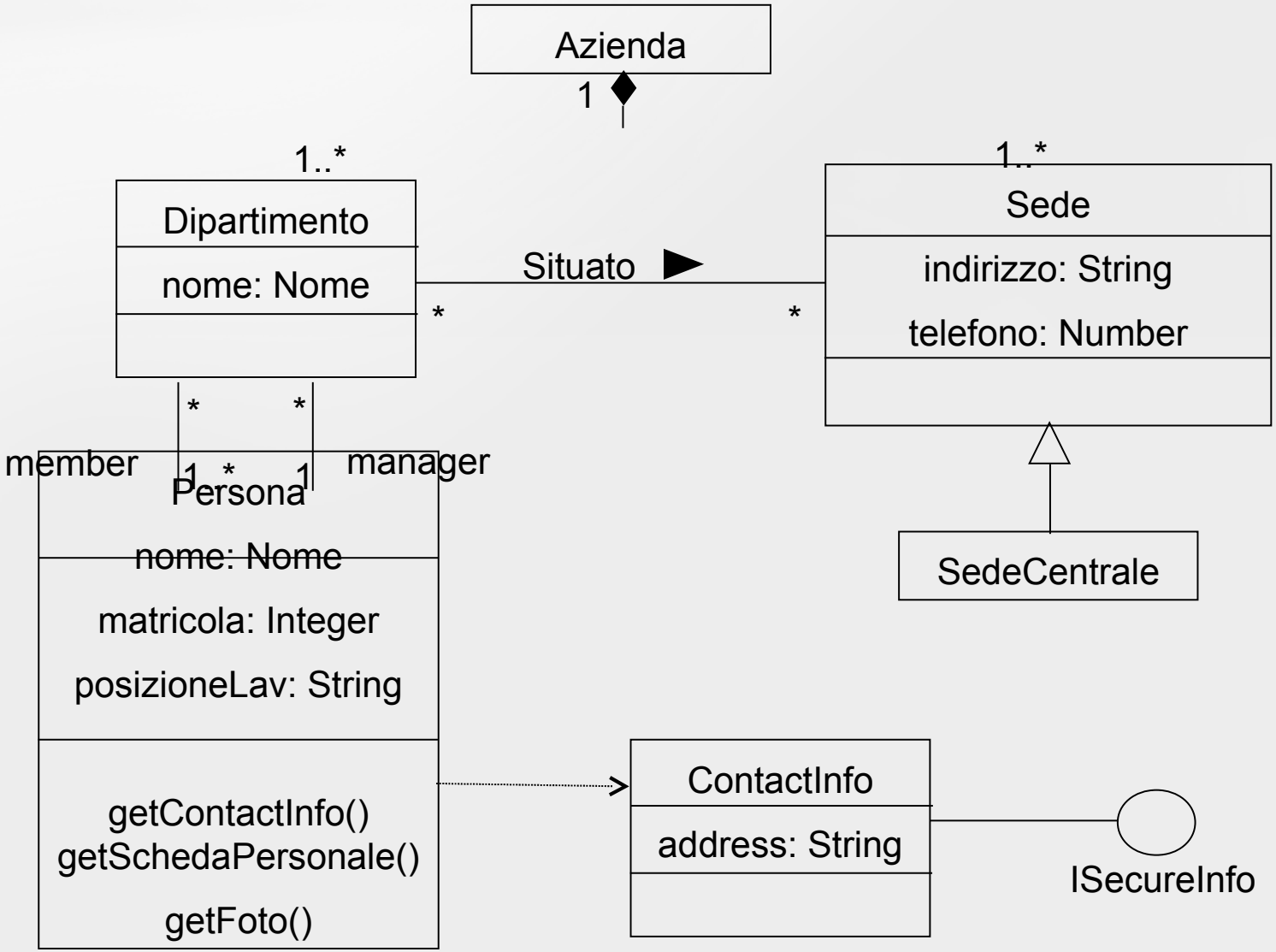
Un elemento legato **NON** è un sottotipo: non si può aggiungere informazione!

Considerazioni finali

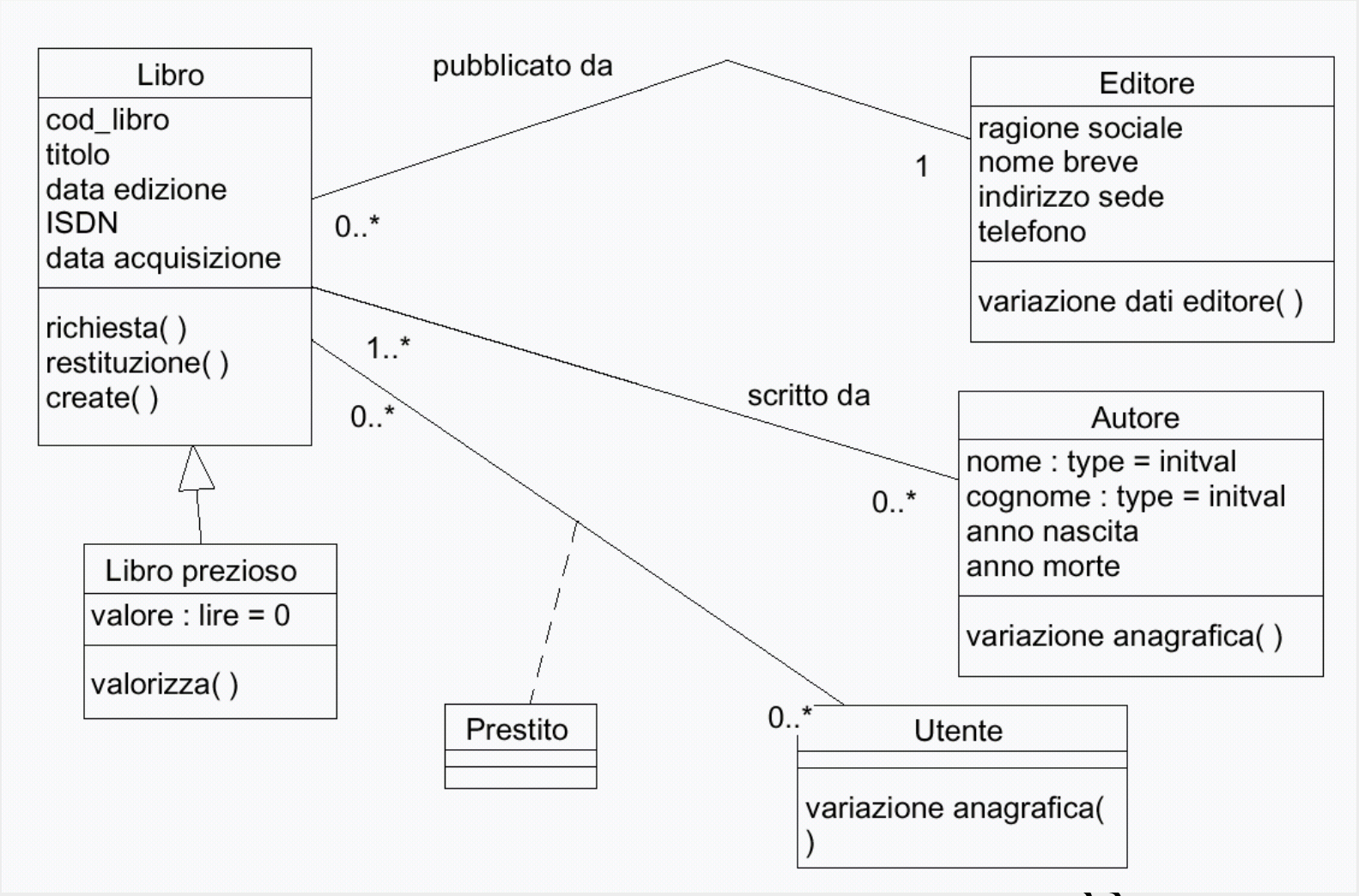
Class Diagram

- Un class diagram rappresenta un insieme di classi, interfacce, e le loro relazioni
- E' elemento fondamentale della progettazione di un sistema
- Un class diagram è tipicamente usato per modellare:
  - la vista di progettazione statica di un sistema, che supporta principalmente i requisiti funzionali del sistema
  - il glossario di un sistema: sono prese decisioni relativamente alle astrazioni da considerare
  - lo schema concettuale di un database

Esempio di class diagram



Un altro esempio



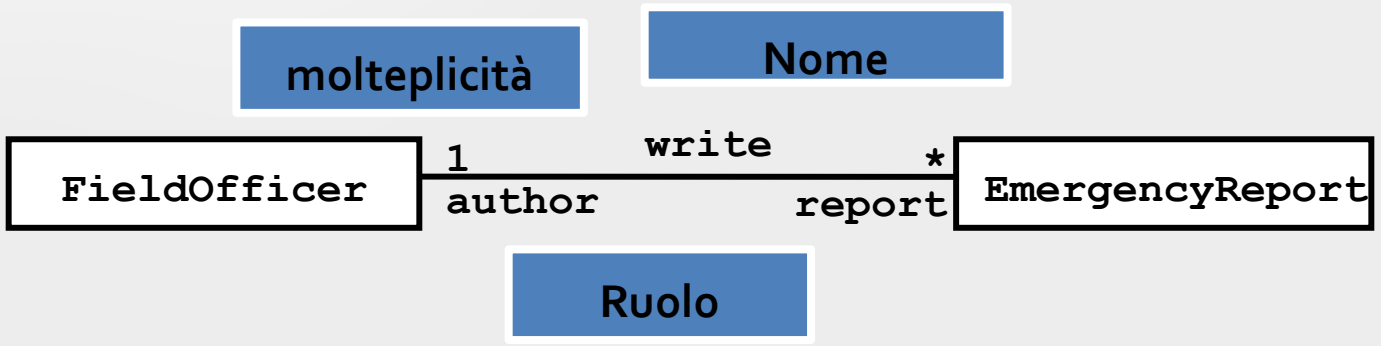
Tornando al documento dei requisiti…

Attività di Analisi

- Le attività che consentono di trasformare gli use case e gli scenari della raccolta dei requisiti in un modello di analisi sono:
  - Identificare gli Oggetti Entity, Boundary, Control
  - Mappare gli Use Case in Oggetti con Sequence Diagrams
  - Identificare le Associazioni
  - Identificare le Aggregazioni
  - Identificare gli Attributi
  - Modellare il Comportamento dipendente dallo stato degli Oggetti individuali
  - Modellare le Relazioni di Ereditarietà
  - Rivedere il Modello di Analisi

Identificare le Associazioni

- I sequence diagram permettono di descrivere le relazioni tra gli oggetti
- I class diagram permettono di descrivere le relazioni e dipendenze tra gli oggetti

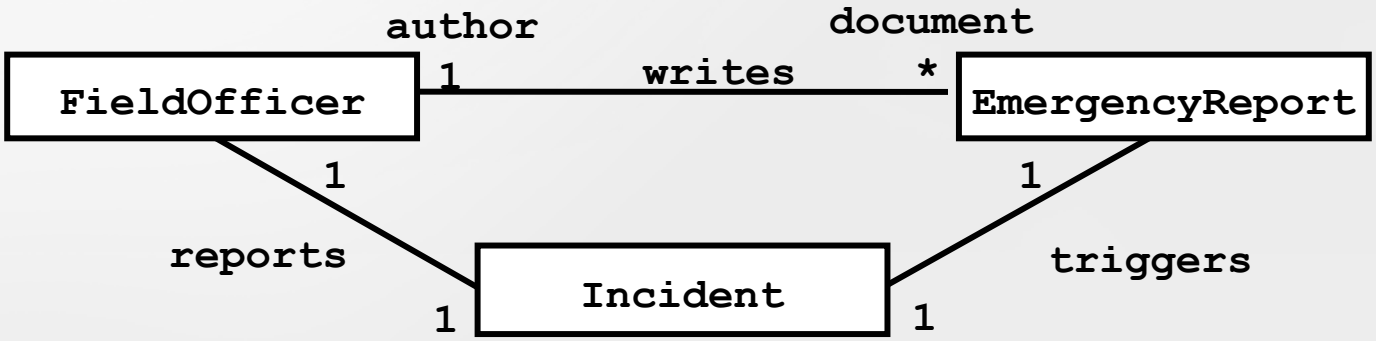


Le associazioni tra entity object sono le più importanti poichè rivelano altre informazioni sul dominio di applicazione

Euristiche per identificare le associazioni

- Esaminare i verbi nelle frasi
- Definire i nomi dei ruoli e delle associazioni
- Eliminare qualsiasi associazione che può essere derivata da altre associazioni
- Non preoccuparsi di specificare le molteplicità finchè l'insieme di associazioni non è stabile
- Troppe associazioni rendono un modello illeggibile

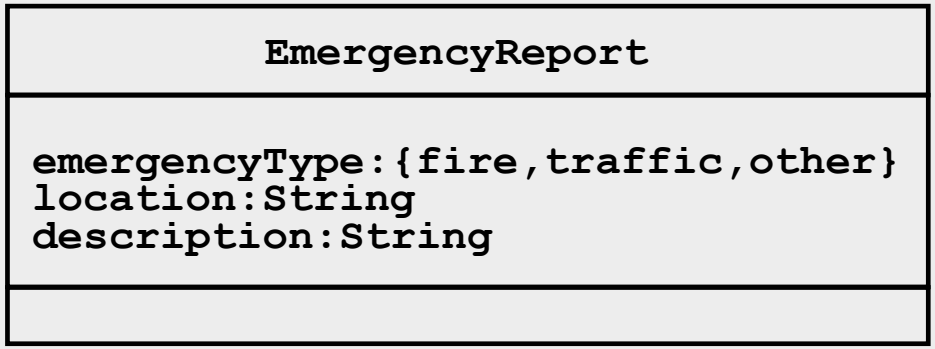
Eliminare le associazioni ridondanti



- La ricevuta di un EmergencyReport causa la creazione di un Incident da un Dispatcher.
- Dato che EmergencyReport ha una associazione con il FieldOfficer che la scrive, non è necessario mantenere un'associazione tra FieldOfficer e Incident.

Identificare gli attributi

- **Le proprietà rappresentate da oggetti non sono attributi (ad es. Author per EmergencyReport)**
- Bisogna identificare tutte le associazioni prima di iniziare l'identificazione degli attributi



Identificare gli attributi

- Gli attributi hanno:
  - Un nome
  - Una breve descrizione
  - Un tipo che descrive i valori che può assumere
- Spesso altri attributi sono scoperti nella fase di sviluppo quando il sistema è valutato dall'utente

Euristiche per Identificare gli attributi

- Esaminare le frasi possessive
- Rappresentare lo stato memorizzato come un attributo dell'oggetto
- Descrivere ogni attributo
- Non rappresentate un attributo come un oggetto; usare invece una associazione
- Non sprecare tempo a descrivere i dettagli prima che la struttura ad oggetti non si sia stabilizzata

Revisione del Modello di Analisi

- Il modello di Analisi è costruito in maniera incrementale ed iterativa
- Una volta che il modello diventa stabile, il modello di analisi viene revisionato prima dagli sviluppatori (revisione interna), poi congiuntamente dagli sviluppatori e dal cliente
- Obiettivo: una specifica dei requisiti corretta, completa, consistente e non ambigua

Attività di analisi