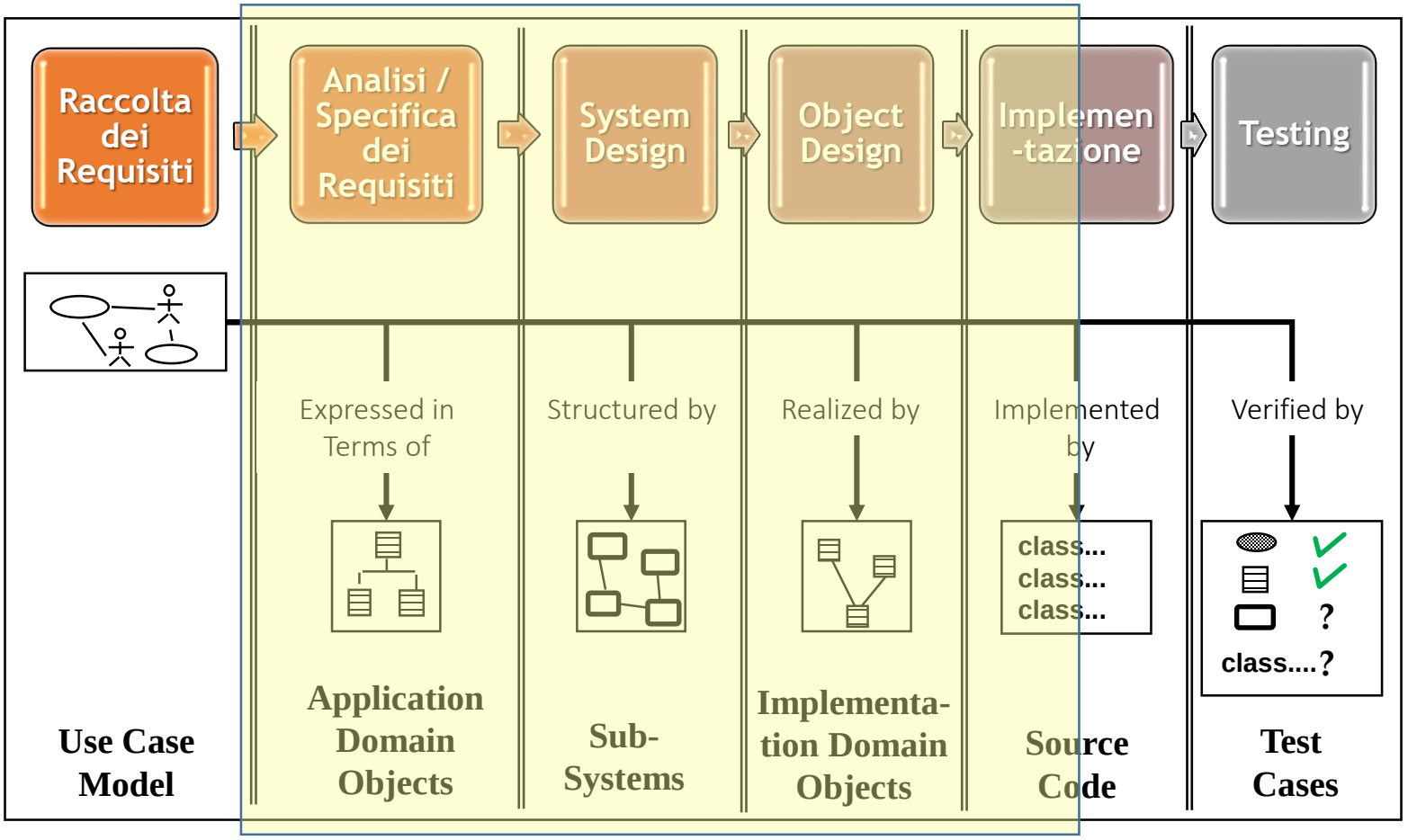
UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II

Ingegneria del Software – I Class Diagrams

Prof. Sergio Di Martino

Ciclo di Vita del Software



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

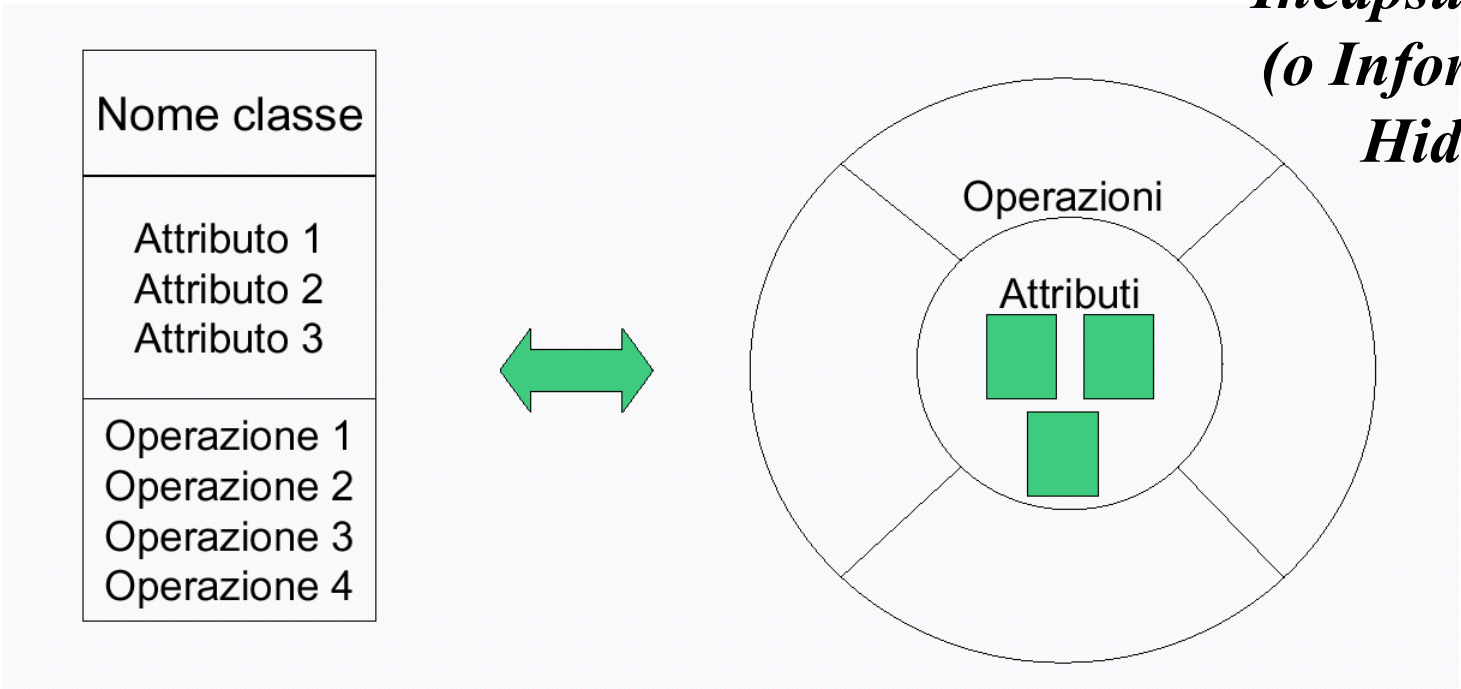
- Un class diagram può essere definito a livello:
  - concettuale (o di Analisi)
    - studia i concetti propri del dominio sotto studio, senza preoccuparsi della loro successiva implementazione
    - E’ una sorta di Dizionario Visuale del dominio
  - di specifica implementativa
    - studia il software da un punto di vista implementativo, specificando come va sviluppato il sistema
    - E’ un raffinamento del precedente

Classe (di oggetti)

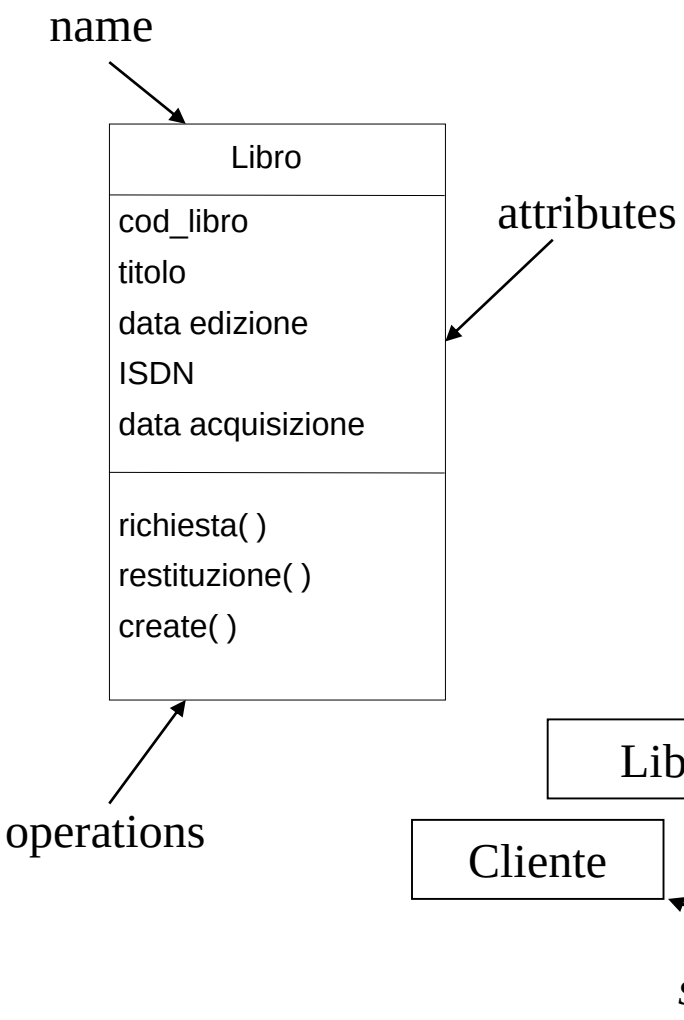
- Un oggetto ha due tipi di proprietà:
  - attributi (o variabili di istanza), che descrivono lo stato
  - operazioni (o metodi), che descrivono il comportamento
- Una classe rappresenta un'astrazione di oggetti simili
  - oggetti che hanno le stesse proprietà: attributi e operazioni
- I termini oggetto e istanza (di classe) sono interscambiabili

Classi in UML

- In UML una classe è composta da tre parti
  - nome
  - attributi (lo stato)
  - metodi o operazioni (il comportamento)



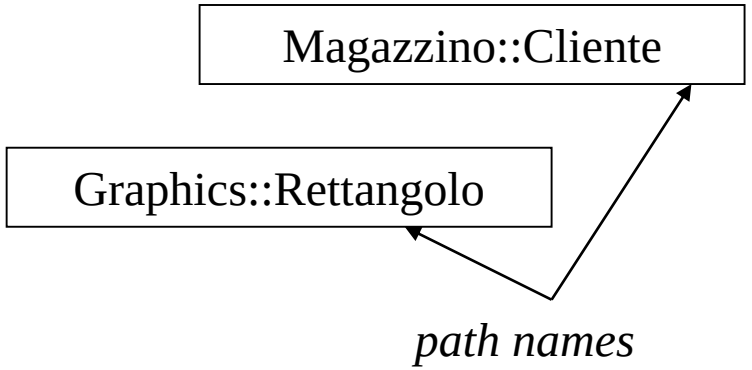
Nomi di Classi



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

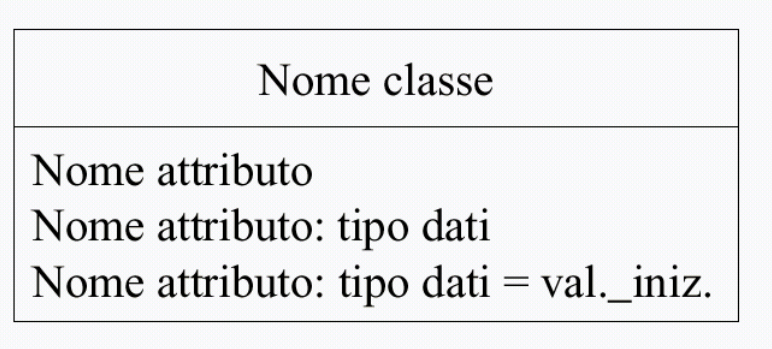


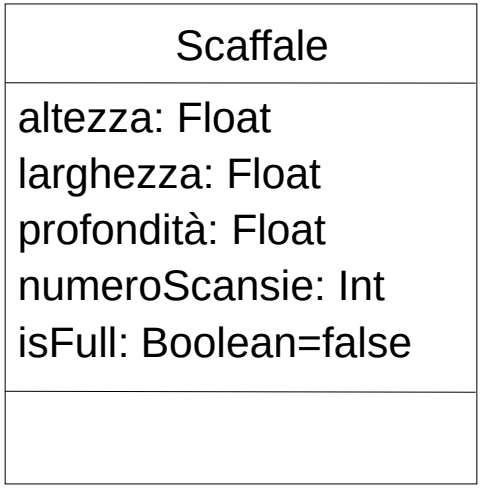
Attributi

- 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 Docente ha il valore "Sergio Di Martino" nell'oggetto "Corso di Ing Sw 1"
- I nomi degli attributi devono essere unici all'interno di una classe
- Il valore di un attributo non ha identità (non è un oggetto)
  - tutte le occorrenze di "24" sono indistinguibili

Attributi

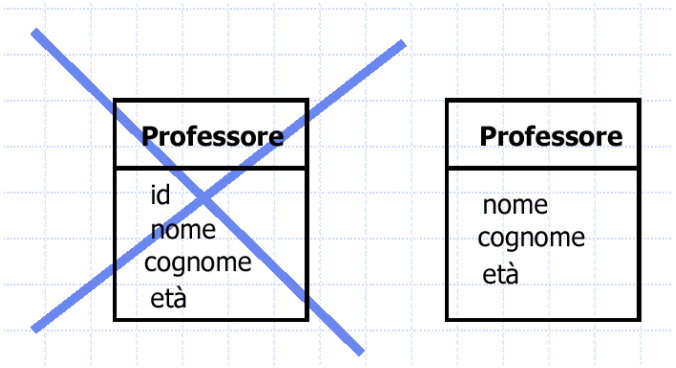
- In UML, per ciascun attributo si può specificare il tipo ed un eventuale valore iniziale
- Tipicamente il nome di un attributo è composto da una o più parole
  - usare il maiuscolo per la prima lettera di ciascuna parola, lasciando minuscola la lettera iniziale del nome (Camel Case)





Individuazione degli attributi

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



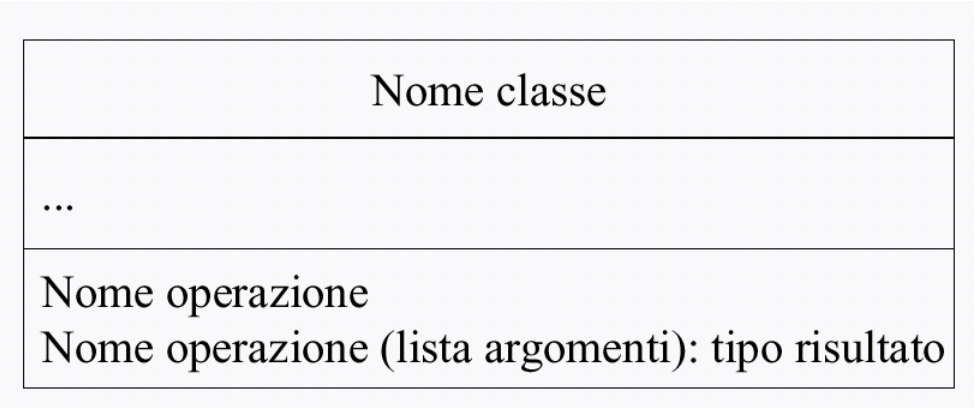
- Candidati per attributi
  - Sostantivi che non sono diventati classi
- Conoscenza del dominio applicativo
  - Persona (ambito bancario)
    - nome, cognome, codiceFiscale, numeroConto
  - Persona (ambito medico)
    - nome, cognome, allergie, peso, altezza

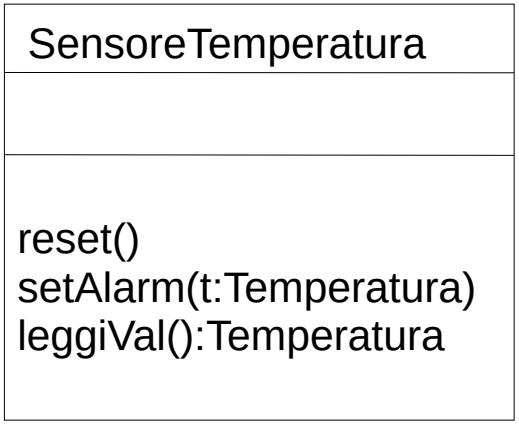
Operazioni

- 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

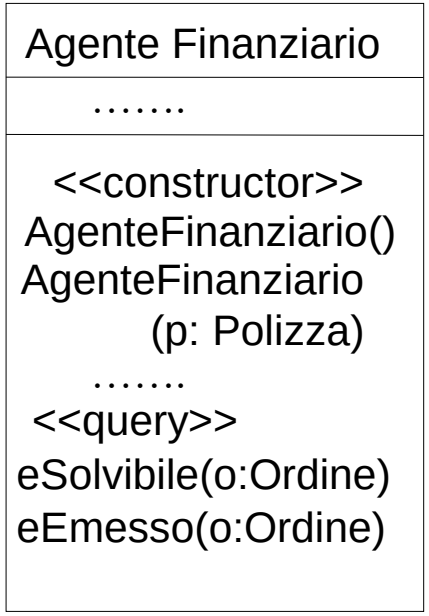
- Per ciascuna operation si può specificare il solo nome o la sua signature, indicando il nome, il tipo, parametri e, in caso di funzione, il tipo ritornato
- Stesse convenzioni dette per gli attributi (Camel Case)





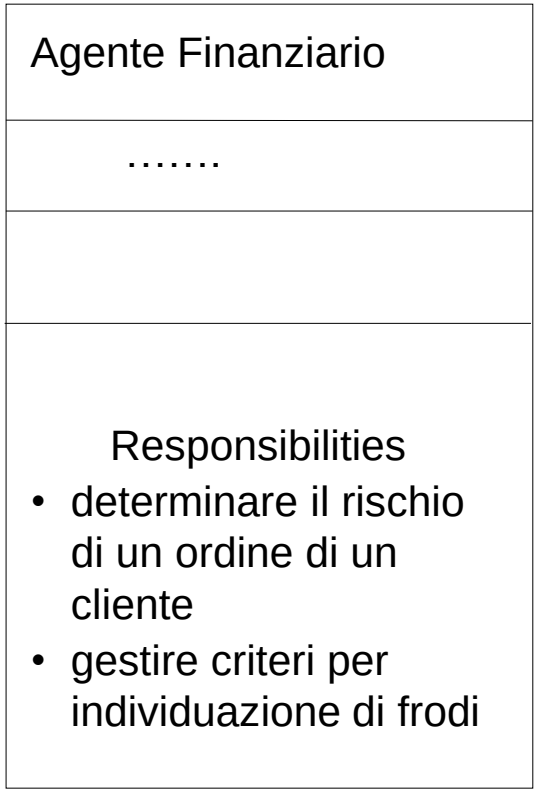
Attributi e Operazioni

- 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 attributes/operations raggrupparli insieme in categorie usando stereotipo quali <<constructor>>, <<query>>, <<update>>



Responsibility

- Una responsibility 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)

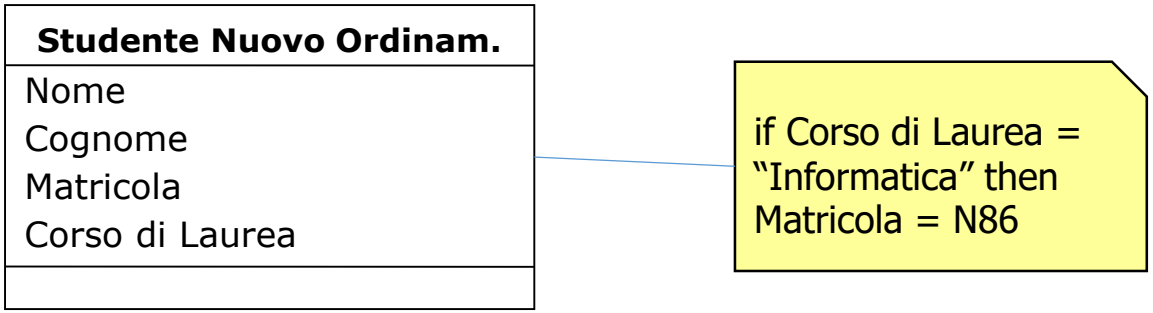


Visibilità di features

- 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 esono rilevanti solo in presenza di più processi o threads

Sintassi finale di features

- 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:
  - Associazioni
  - Generalizzazioni/Specializzazioni
  - Dipendenze

Associazioni

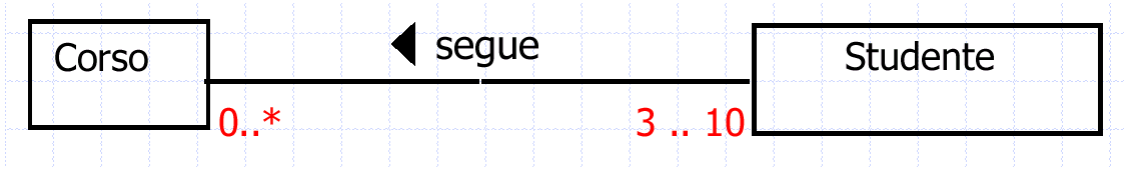
- Un sistema O-O è costituito da classi che collaborano tra loro scambiandosi messaggi
- Quando è in esecuzione un sistema O-O è popolato da istanze di classi
- Quando un'istanza di classe passa messaggi a un'altra classe si sottintende l'esistenza di un'associazione tra le due classi

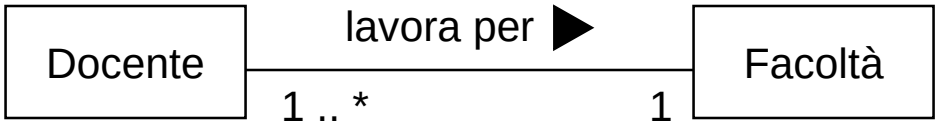
Associazioni

- Indicano relazioni tra classi
- Possono essere Riflessive
- Possono essere n-arie (raro)
- Adornments:
  - Nome: non obbligatorio; può avere un verso di lettura
  - 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 dice
  - Se l'associazione è obbligatoria oppure no
  - Il numero minimo e massimo di oggetti che possono essere relazionati ad un altro oggetto





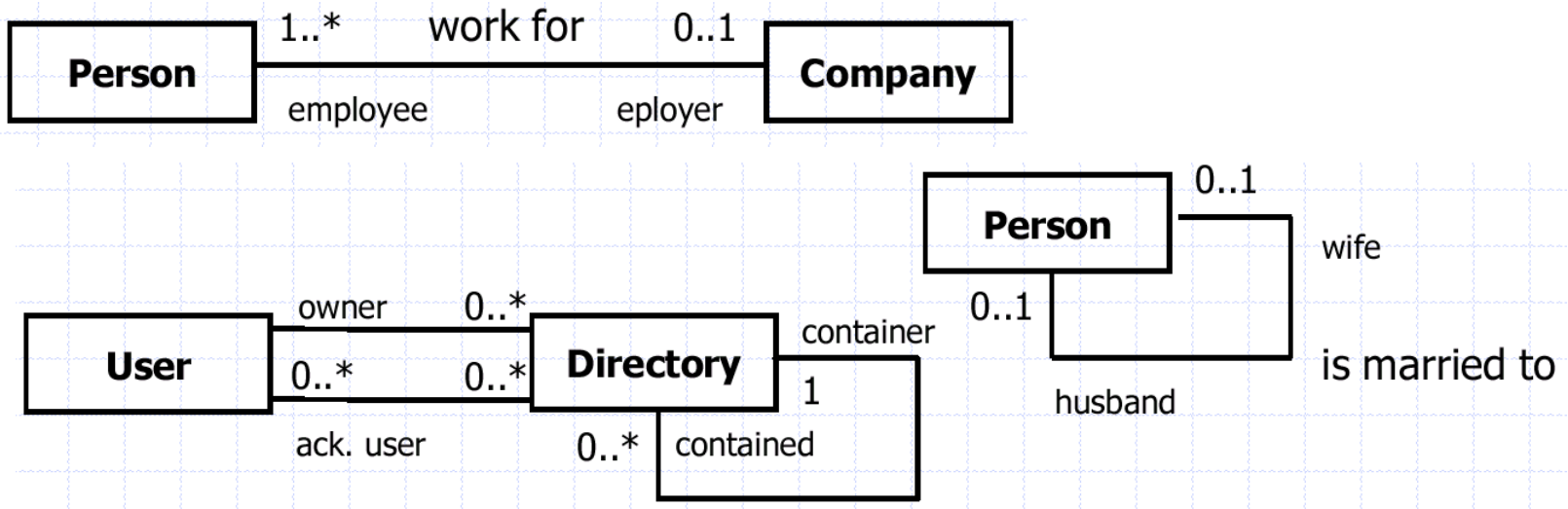
Molteplicità delle associazioni

Indicator	Meaning
0..1	Zero or one
1	One only
0..*	Zero or more
1..*	One or more
n	Only *n* (where *n* > 1)
0..n	Zero to *n* (where *n* > 1)
1..n	One to *n* (where *n* > 1)

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

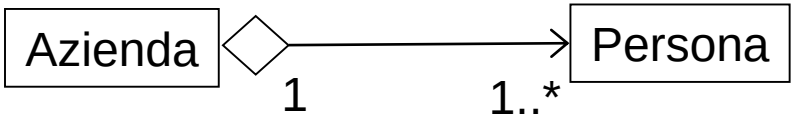
Ruoli

- 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)



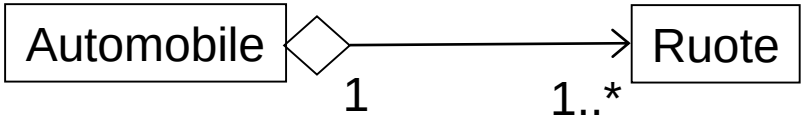
Aggregazione

- 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

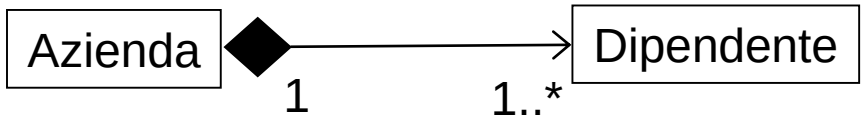
- La relazione di aggregazione è un'associazione speciale che aggrega gli oggetti di una classe componente in un unico oggetto della classe
  - la si puo' leggere come "è fatto da" in un verso e "è parte di" nell'altro verso



- 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

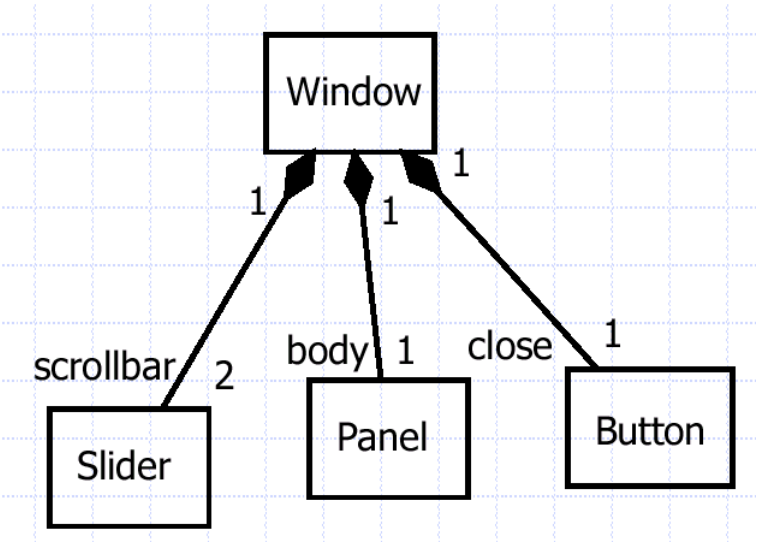
Composizione

- 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

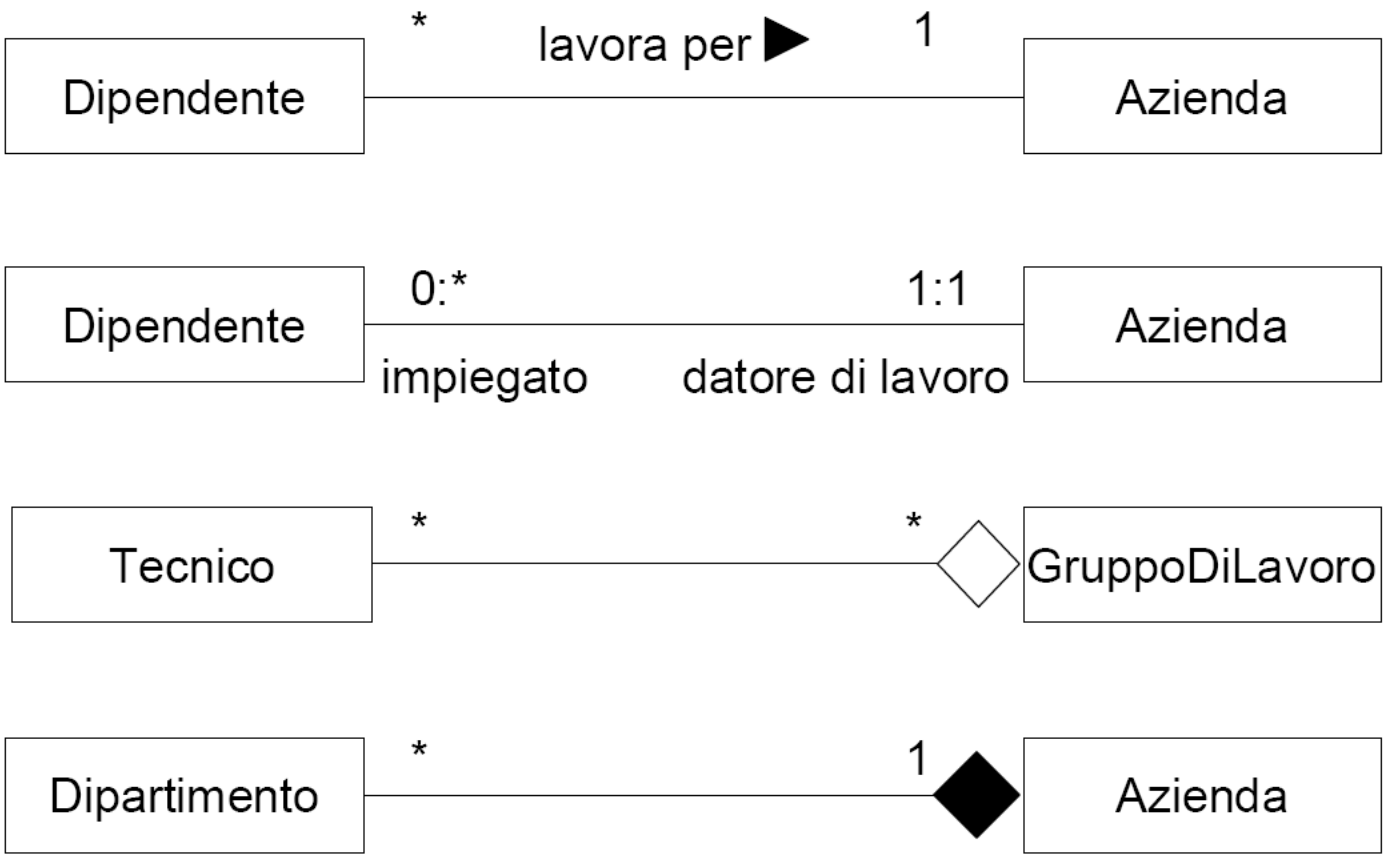


Composizione

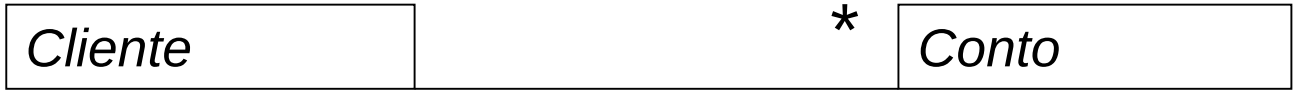
- Una relazione di composizione è un'aggregazione forte
  - Le parti componenti non esistono senza il contenitore
  - Ciascuna parte componente ha la stessa durata di vita del contenitore
  - Una parte può appartenere ad un solo tutto per volta



Riassunto Associazioni



Associazioni many-to-many: esempio

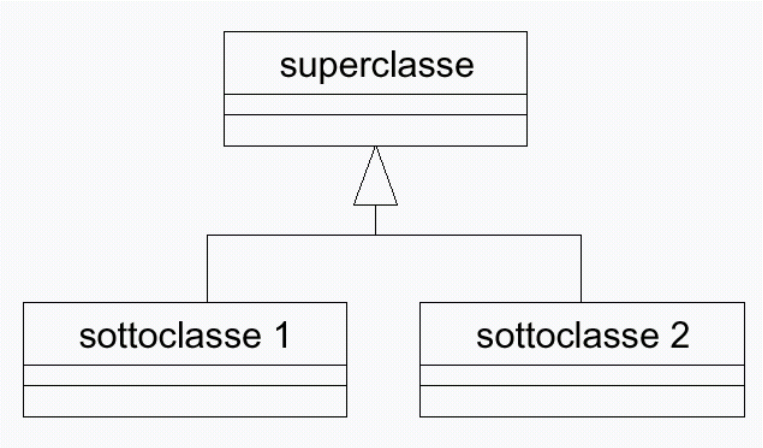


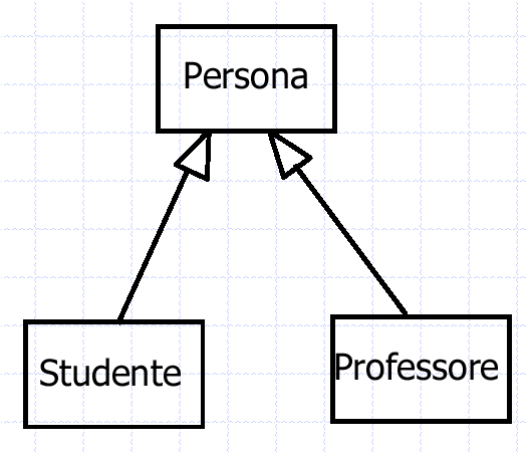

```
public class Cliente {
       private List conti;
       public Cliente() {
          conti = new ArrayList();
       }
       public void addConto (Conto c)
{
             if (!conti.contains(c)) {
                   conti.add(c);

       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)
  - “puo’ 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

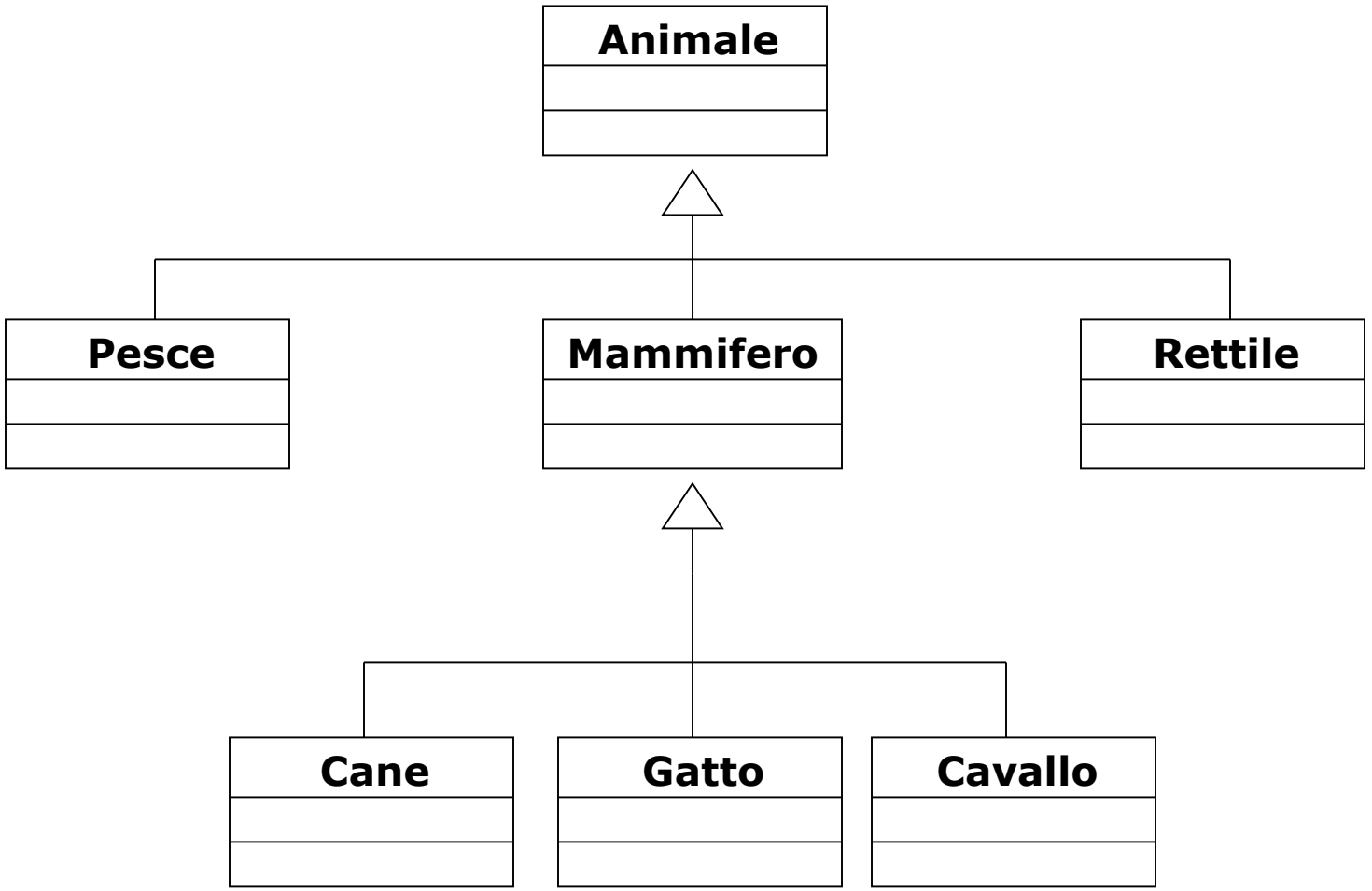
Generalizzazione

- 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

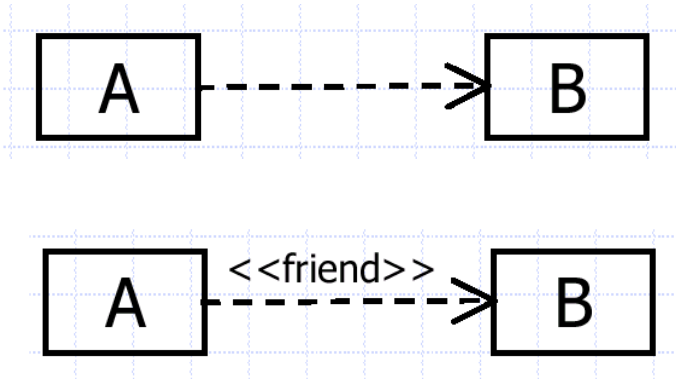
- Specializzazione: attività che identifica concetti più specifici da quelli di più ad alto livello
  - Es. Stiamo costruendo un sistema di gestione delle emergenze e stiamo discutendo le funzionalità con il cliente: il cliente introduce prima il concetto di incidente, quindi descrive tre tipi di incidenti: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

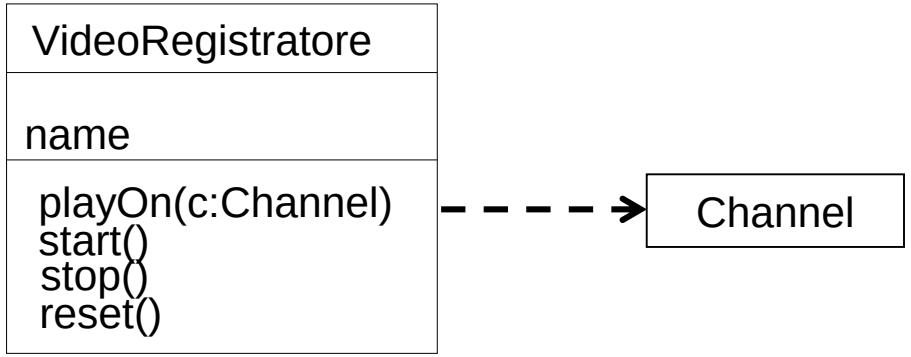
Generalizzazione fra classi (es.)



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

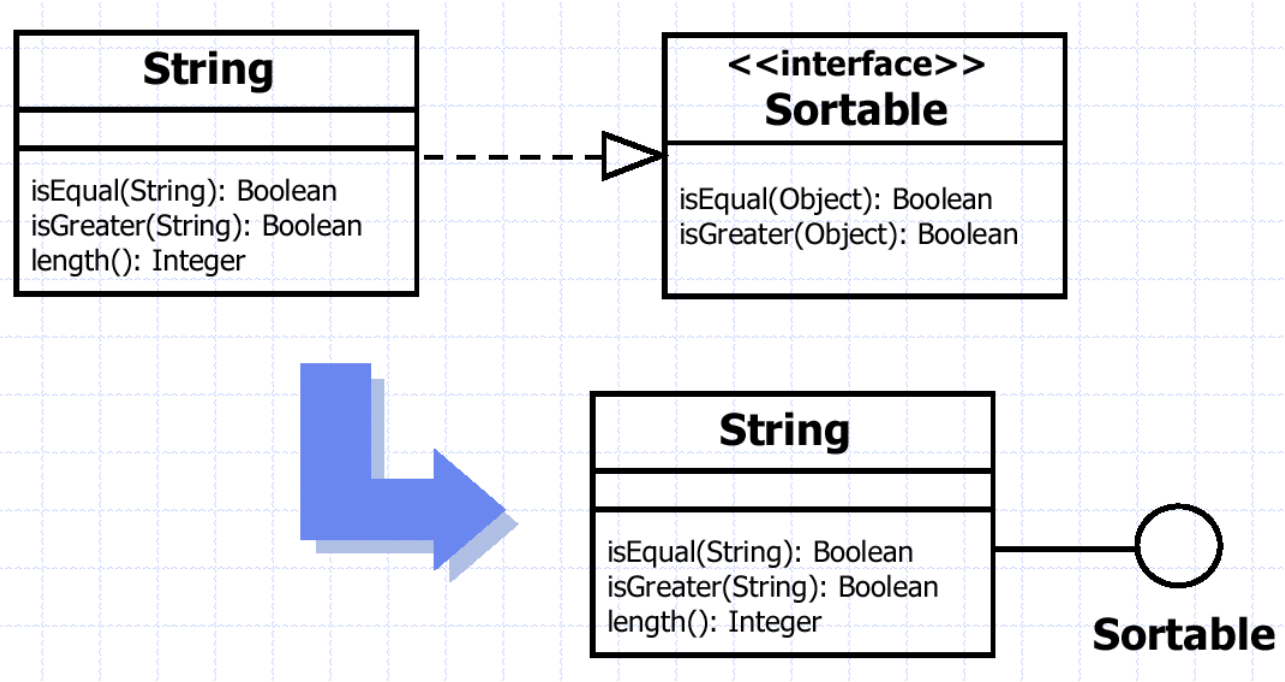
Altri Elementi dei Class Diagrams

Interfacce e realizzazioni

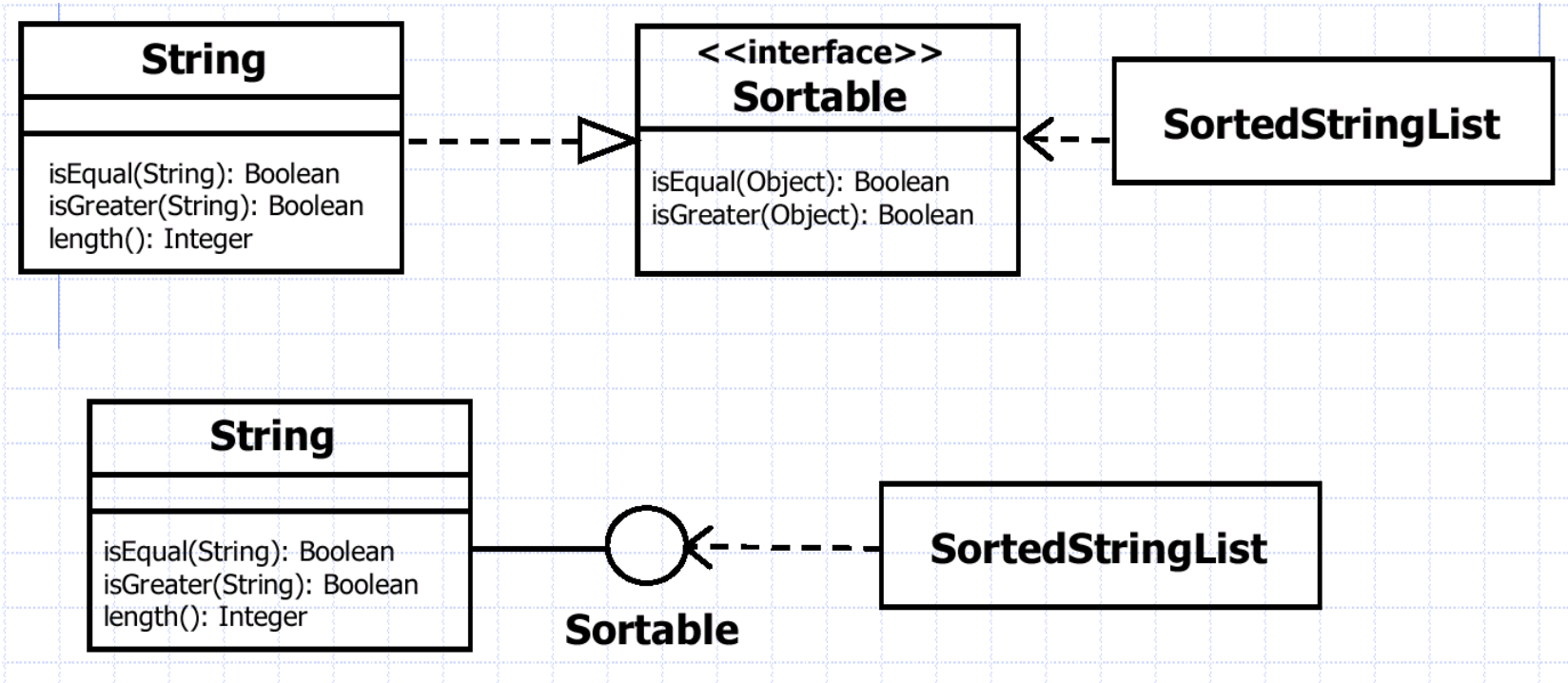
- Un'interfaccia viene modellata allo stesso modo in cui viene modellato il comportamento una classe e rappresenta un insieme di operazioni che una classe offre ad altre classi
  - Un'interfaccia non ha attributi ma soltanto operazioni (metodi). In UML per rappresentare le interfacce si utilizza un rettangolo 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

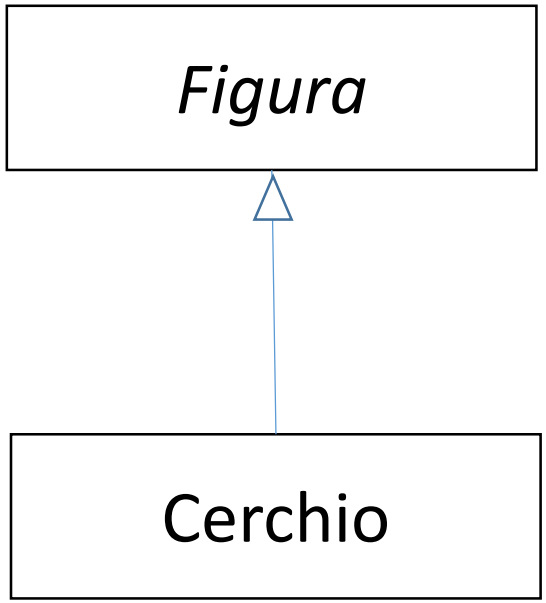


Interfaccia (Uso)



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

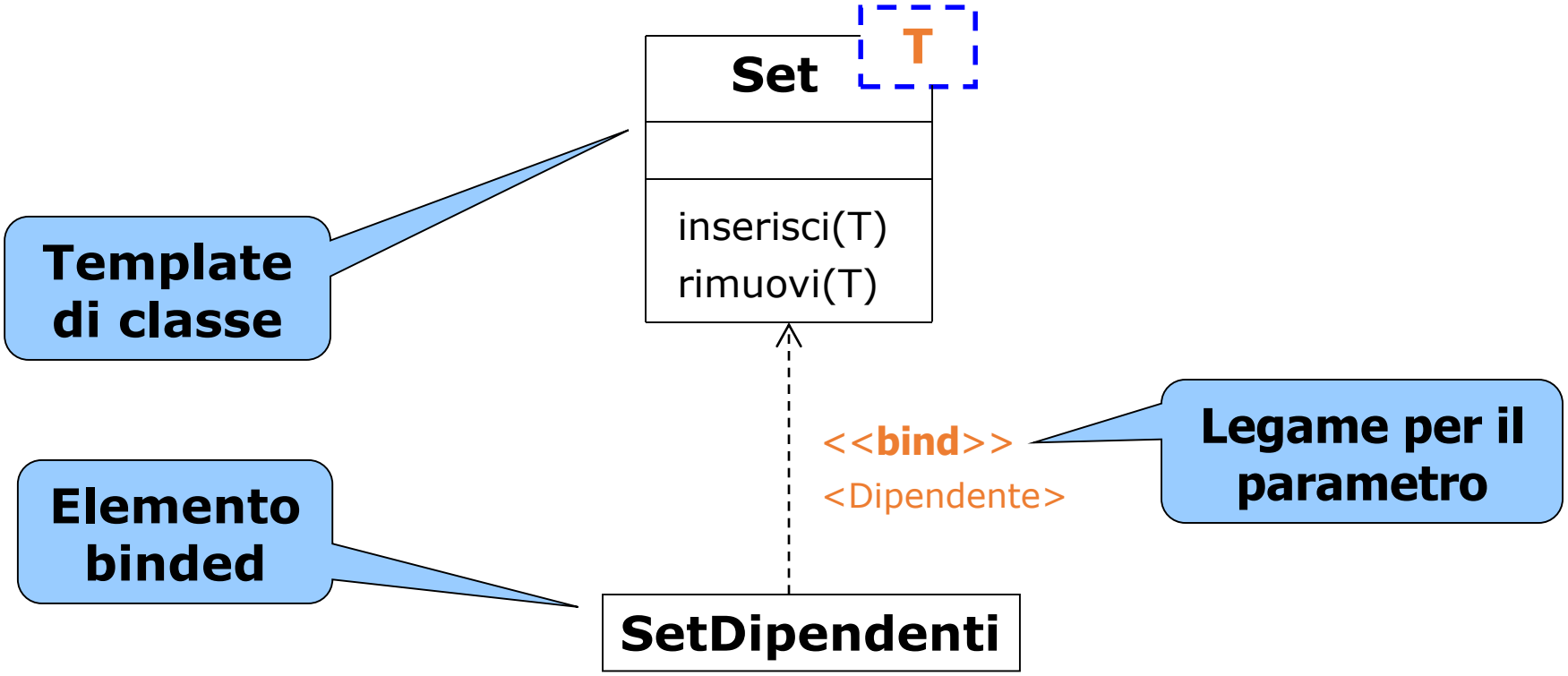
- Molti linguaggi supportano il concetto di classe parametrica (o template), ossia consentono di definire classi che dipendono da un parametro da specificare.
  - Ciò è utile per lavorare con collezioni di elementi in linguaggi fortemente tipizzati, cioè con controlli rigorosi sui tipi

Parametro

```
class Set <T> {
  void inserisci(T nuovoElem)
  void rimuovi(T unElem)
}
```

- per poi creare insiemi di elementi specifici
- Set <Dipendente> SetDipendenti

Classi parametriche (es.)



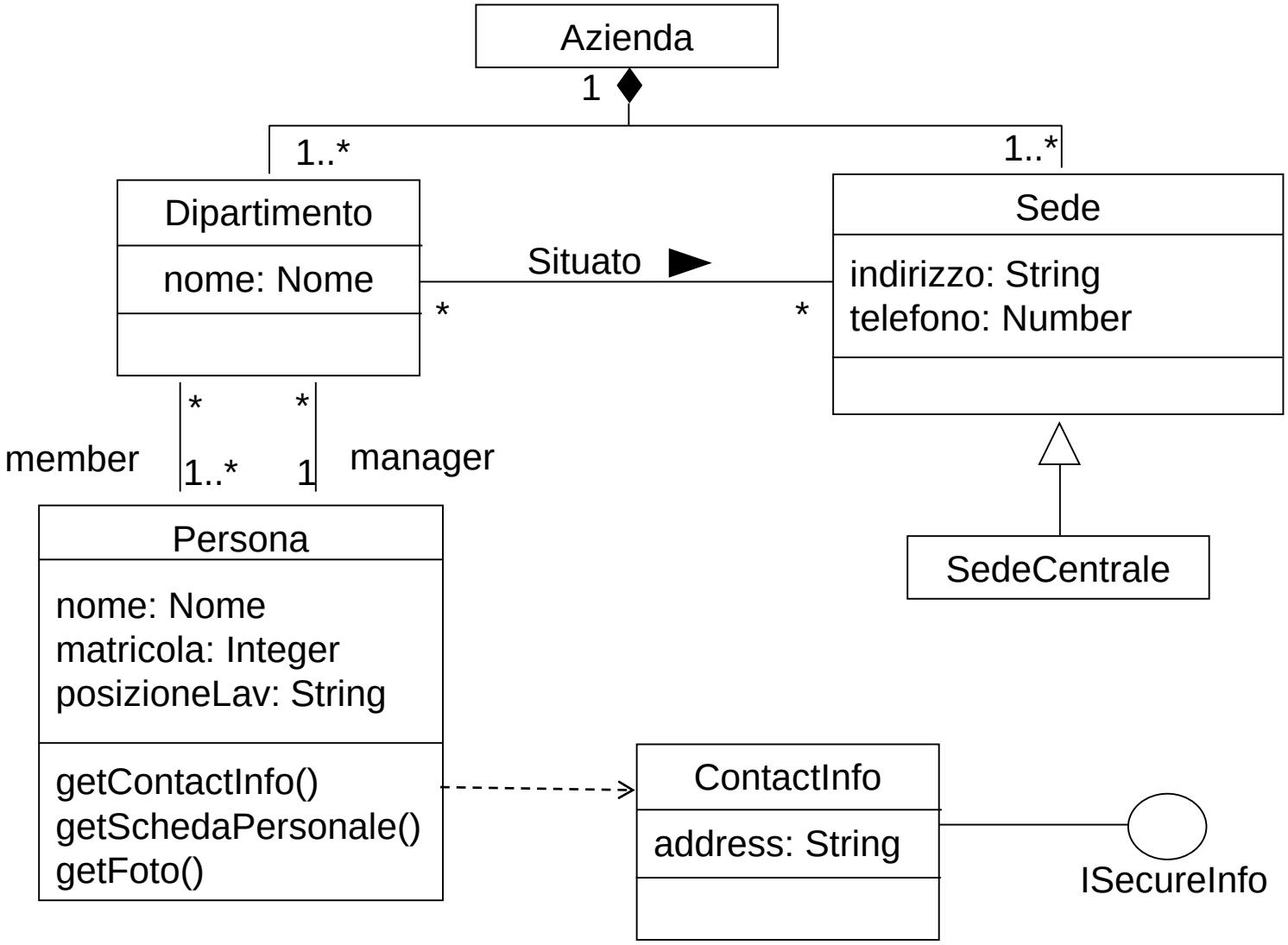
Un elemento legato **NON** è un sottotipo: non si può aggiungere informazione!

Considerazioni finali

Class Diagram

- Un class diagram rappresenta un insieme di classi, interfacce, e le loro relationship
- E' il cuore della progettazione di un sistema
- Un class diagram è tipicamente usato per modellare:
  - la vista di static design 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