Sistemi Informativi:

Analisi avanzata

Modellazione avanzata delle classi

Stereotipi

Vincoli

Informazioni derivabili

Visibilita’

Associazioni qualificate

Associazioni come classi

Classi parametriche

Varie

stereotipo

Modifica/estende la semantica di un elemento UML

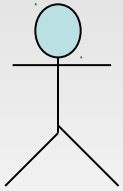
Incluso tra << >>, es <<PK>>

Puo’ avere una icona associata

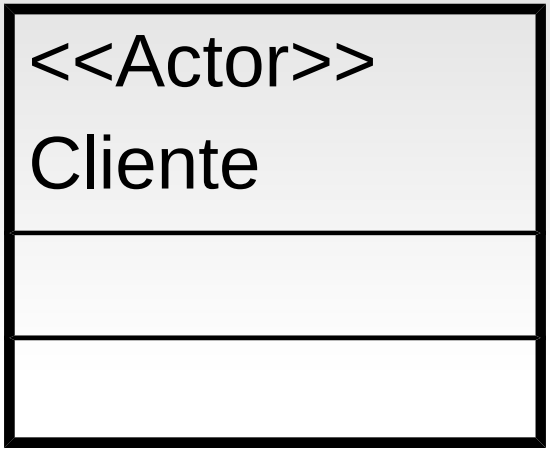
Alcuni stereotipi comuni sono predefiniti nel linguaggio

Un insieme di stereotipi usati per affrontare un determinato problema si chiama “profilo”

stereotipi



Stereotipo ad icona



Stereotipo ad etichetta

Nessun Stereotipo

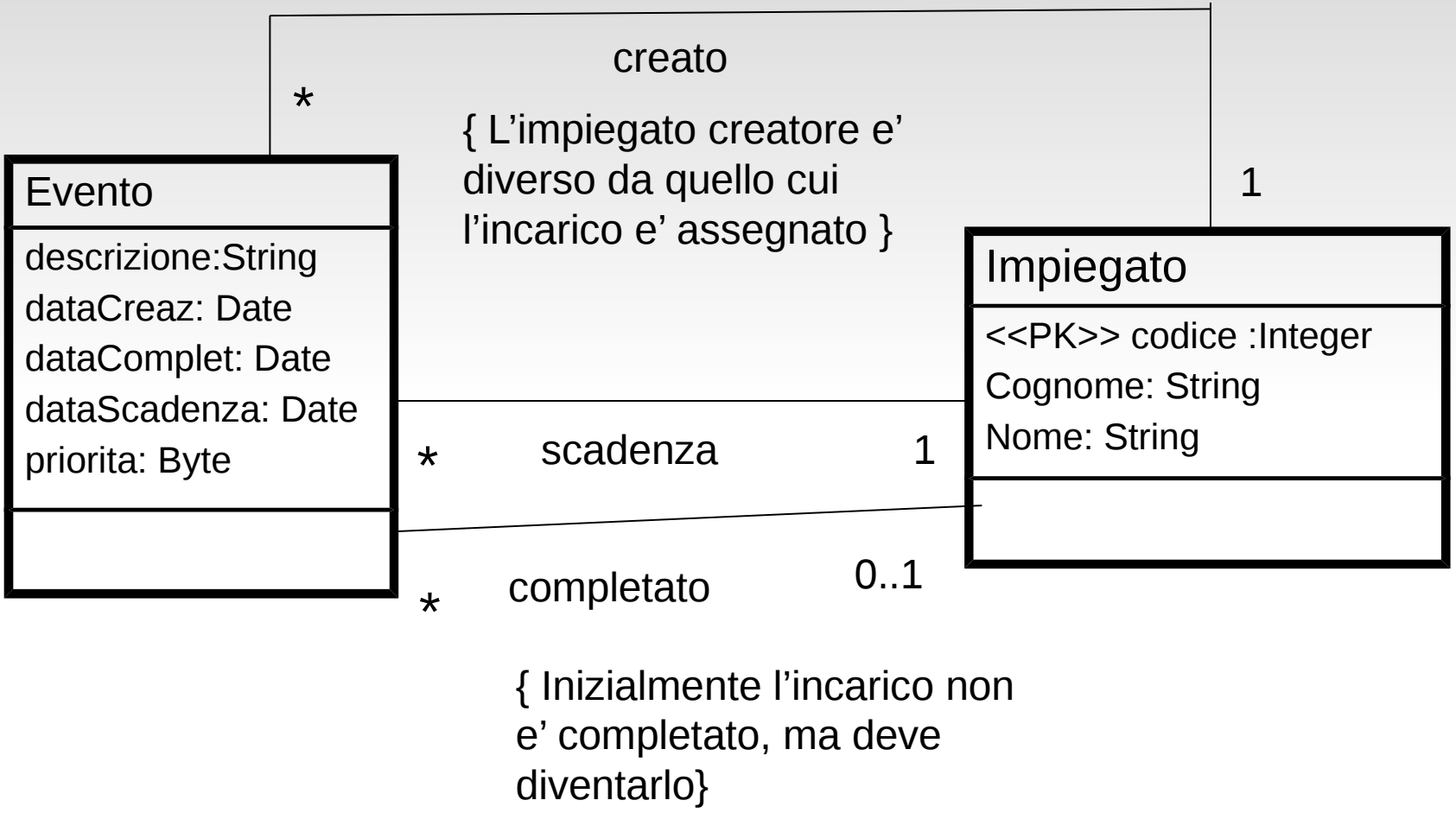
vincoli

A volte introdotti tramite uno stereotipo nel modello

Si possono mostrare vincoli semplici nel diagramma tra parentesi graffe { }

Vincoli piu’ complessi si mantengono in documenti di testo conservati nell’ archivio dei documenti

Vincoli tra associazioni. Esempio: gestione contatti



Note ed etichette (tags)

Spesso la nota e' un tipo complesso di vincolo

Un rettangolo con un l'angolo destro in alto ripiegato

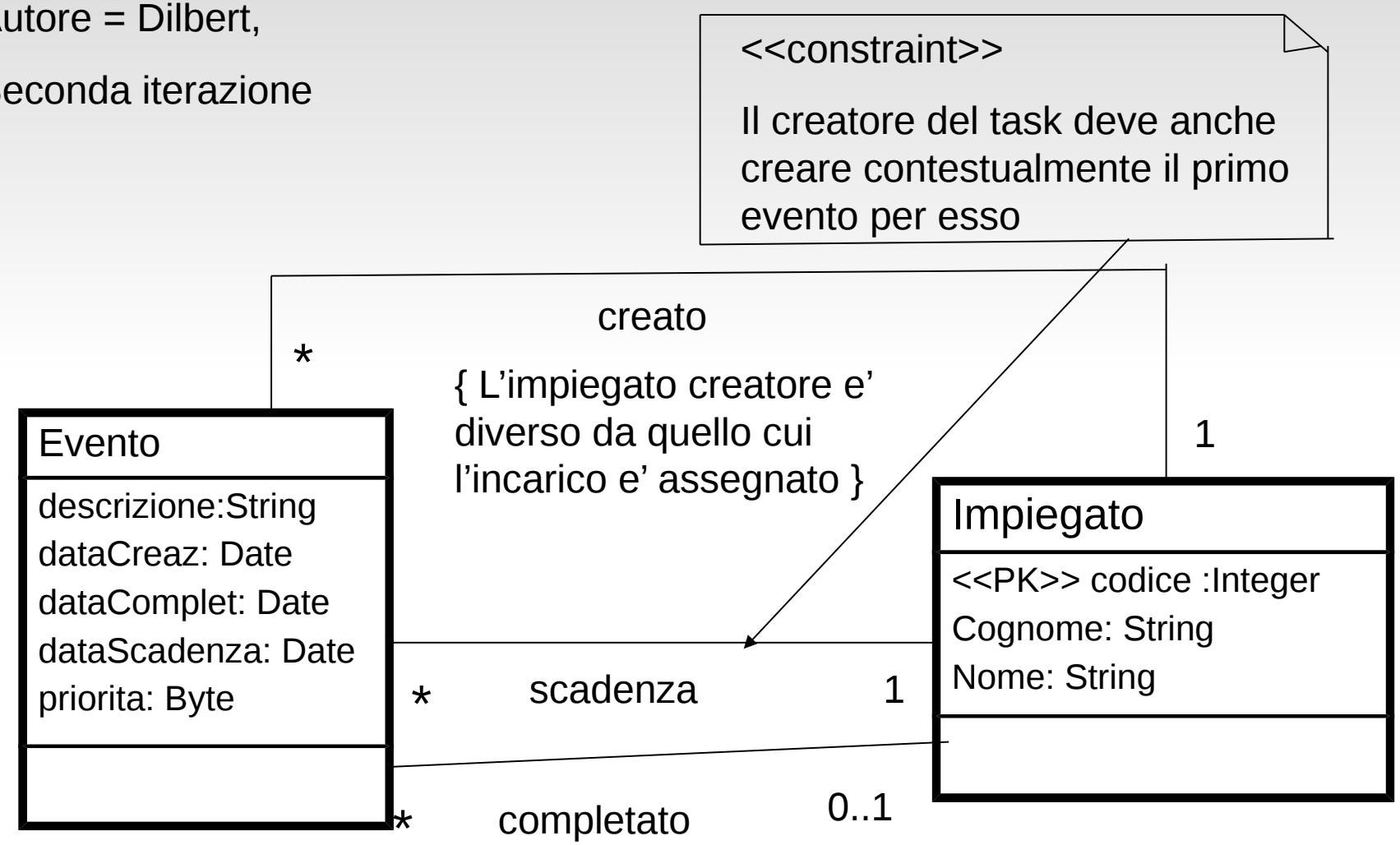
Le tag sono anch'esse informazioni testuale scritte tra graffe della forma tag = valore

Spesso usate per fornire informazioni di gestione del progetto

Note e tag. Esempio: gestione contatti

Autore = Dilbert,

Seconda iterazione



Visibilita' delle proprieta' ereditate

Quando estendiamo una classe di base B con una classe derivata D, tutto cio' che e' privato di B rimane in ogni caso privato. Cio' premesso abbiamo tre casi:

- D: public B -> la visibilita' delle proprieta' di B e' invariata
- D: private B -> la visibilita' delle proprieta' di B diventa private
- D: protected B -> la visibilita' delle proprieta' public di B diventa protected

La classe D non puo' mai ampliare la visibilita' delle proprieta' di B

Informazione derivata

Un vincolo che si applica frequentemente ad attributi ed associazioni

Calcolata in base ad altri modelli dell'elemento

Importante nella fase di progettazione

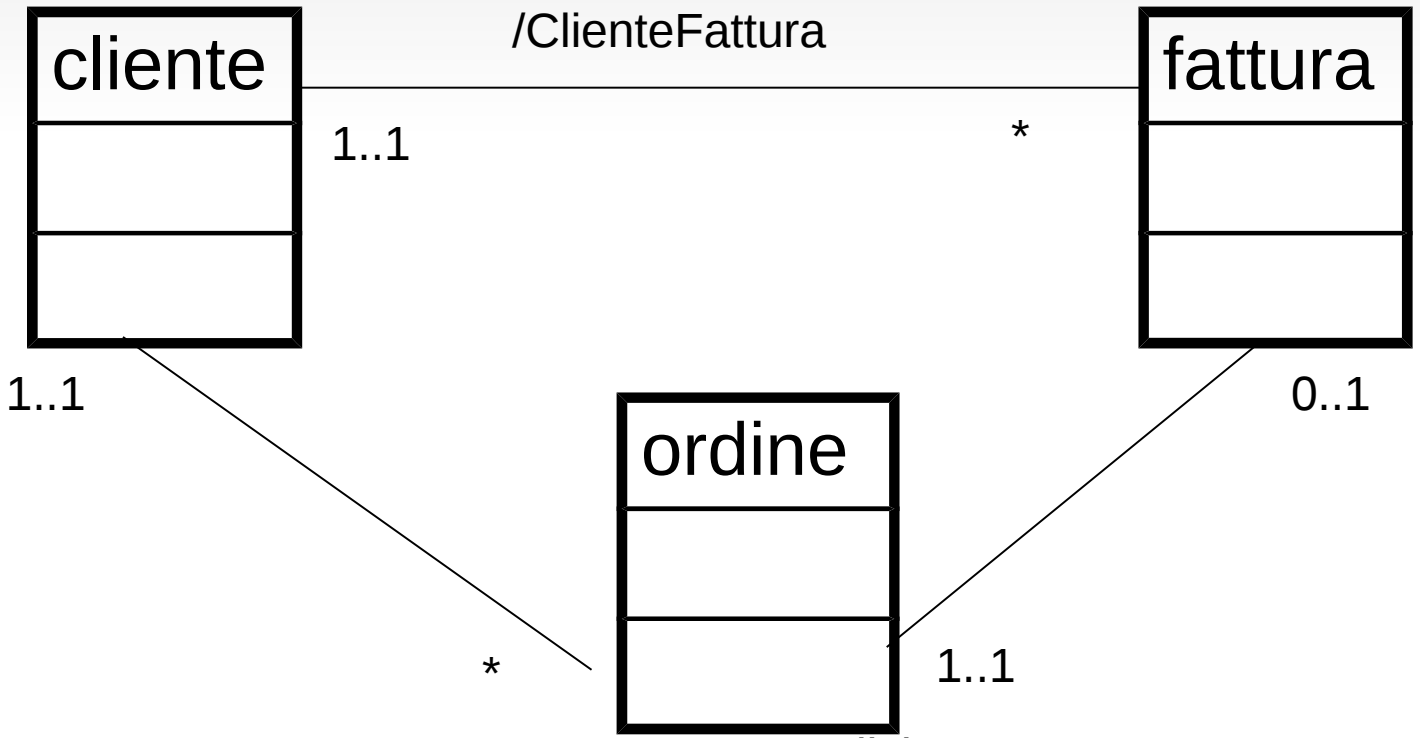
- memorizzata
- virtuale

In UML si prefissa con / il nome dell'attributo o dell'associazione

Usare ogni volta che motivi di efficienza non lo precludano

Associazione derivata

Supponendo di avere 3 classi connesse tra di loro da due associazioni che siano di cardinalita' 1..1, allora si puo' derivare (senza creare in modo permanente) una terza associazione che associa direttamente la prima classe all'ultima



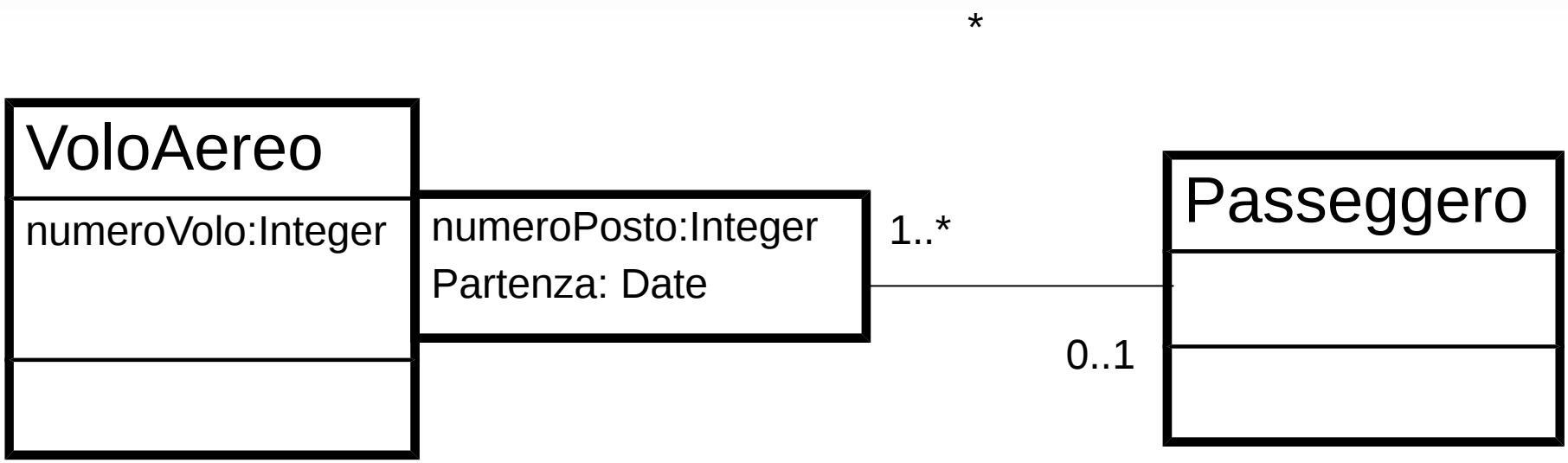
Associazione qualificata

Ha una scatola (qualificatore) ad un estremo di una associazione binaria

Serve per attraversare la relazione dalla classe qualificata all'altra

Serve" per la riduzione della cardinalita' "*" ad una funzione come in questo esempio

Ovviamente avremmo potuto cambiare modello e magari inserire una classe BiglietoAereo



Associazione reificata

Usata tipicamente quando vi sono delle associazioni molti-a-molti ed ogni associazione ha i suoi attributi.

Per una classe di associazione C tra A e B ci puo' essere solo una istanza di C per ogni coppia di A e B

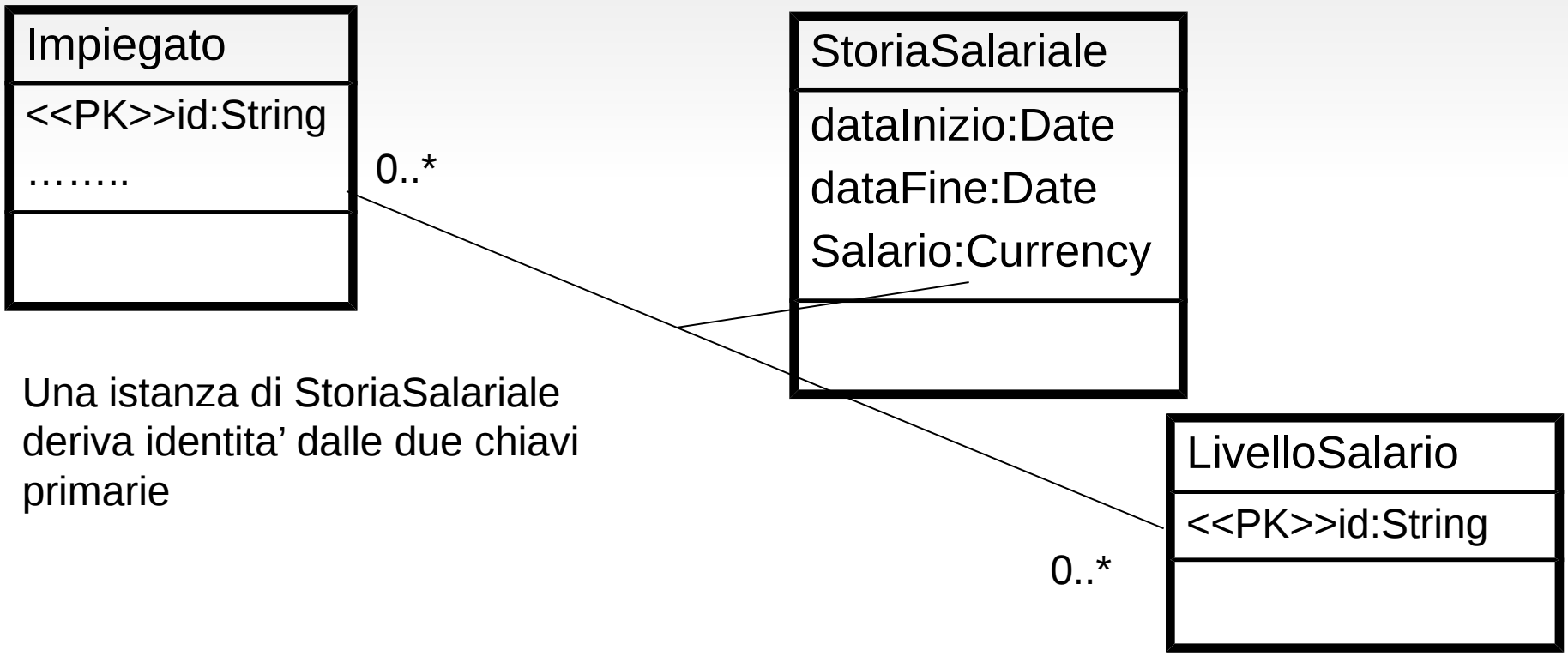
Se non va, occorre creare una nuova classe D

La Classe reificata D ha due associazioni binarie ad A e B

- D e' indipendente da A e B
- D ha identita' propria cosicche' se ne possono creare istanze multiple

Esempio:storia salariale

La ditta ha un certo numero di fasce salariali. Il salario di un impiegato evolve sia per cambio di fascia che cambio all'interno della stessa fascia. Vogliamo registrare ogni variazione.

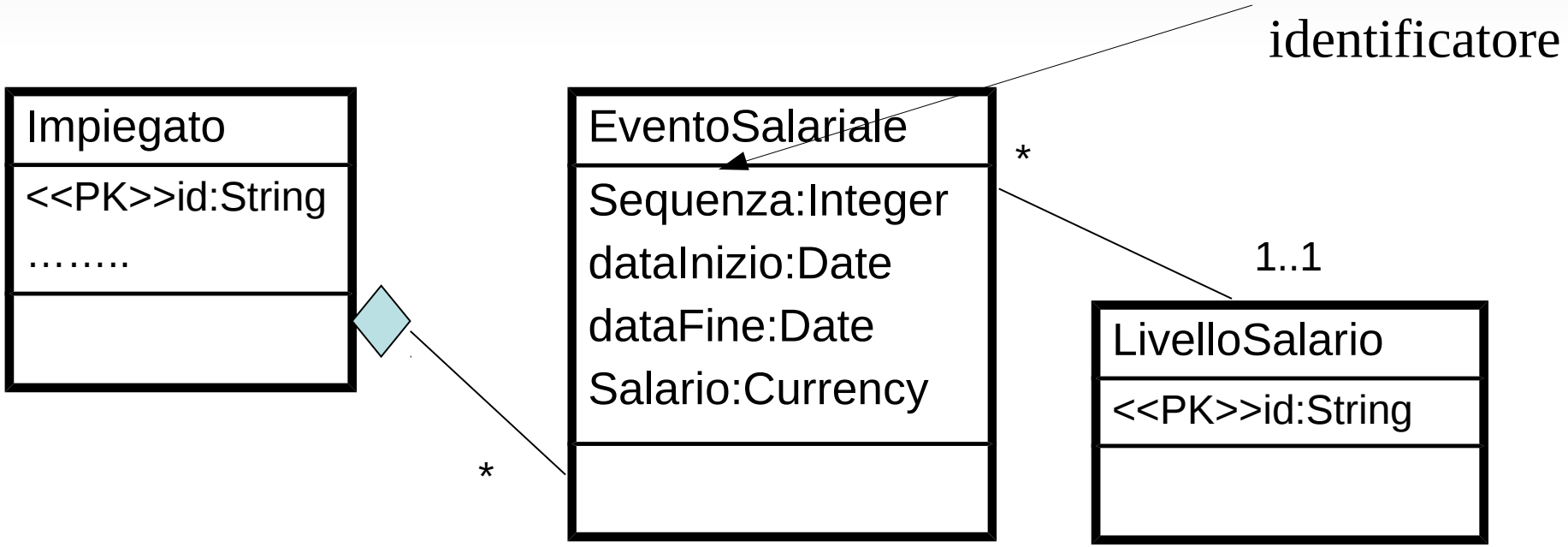


Una istanza di StoriaSalariale deriva identita' dalle due chiavi primarie

Esempio:storia salariale

La soluzione non va perche' sostanzialmente possiamo avere solo una associazione tra ogni livello salariale ed un impiegato; non riusciamo ad registrare la storia di un impiegato dentro una singola fascia. Gli oggetti della StoriaSalariale derivano identita' dalla unione delle due chiavi. Che fare? Aggiungiamo un identificatore ad esso e definiamo due associazioni

Succede spesso per le "history"



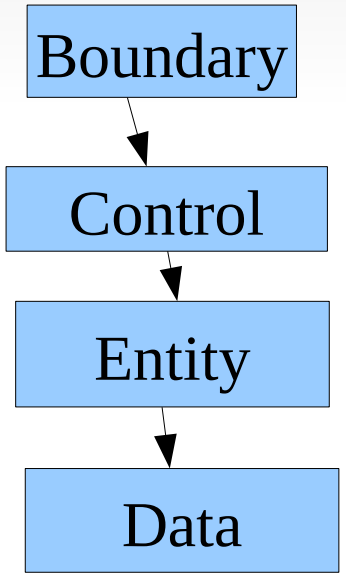
Stratificazione delle classi BCED

Boundary: interfaccia utente ed altri sistemi

Control: coordinano Boundary ed Entity

Entity: astrazione verso DB

Data: classi del DB



Overriding e overloading

Overriding e' la base del polimorfismo

- Esistono metodi diversi in diverse classi con la stessa "forma" (inglese: signature)
  dipendente.calcolaSalario()
  dirigente.calcolaSalario()

Overloading quando uno metodo ha piu' definizioni nella stessa classe con parametri diversi (es. Real.add(Real) , Real.add(Frazione))

3 possibili “Ereditarieta’”:

attraverso classe:

- Se eredito da una entita’ che si puo’ istanziare, eredito da una classe: Dirigente->Impiegato.
- Eredito sia attributi che operazioni
- Ha senso fare istanze dell’antenato
- La classe figlia e’ una specializzazione

3 possibili “Ereditarieta’”:

Classe astratta:

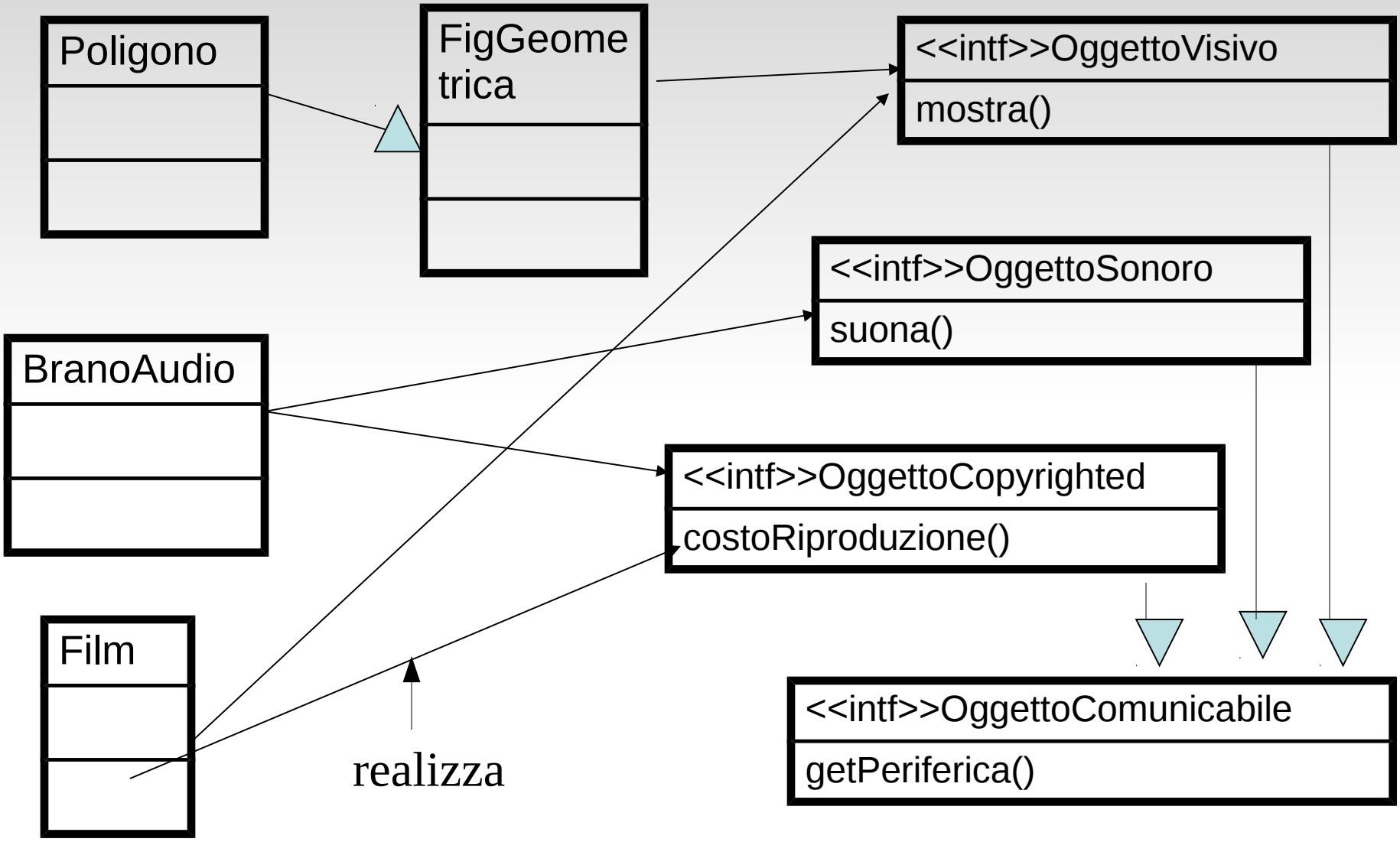
- Se eredito da una entita’ che NON si puo’ istanziare, eredito da una classe astratta: Auto->Veicolo
- Non ha senso istanziare (nel sistema che intendiamo modellare) la classe astratta perche’ incompleta
- Le classi figlie partizionano l’insieme delle istanze (es. Individuo: maschio,femmina)
- Le classi figlie devono realizzare le operazioni astratte della classe antenato per completarla
- Serve a descrivere (fattorizzare) proprieta’ che devono essere meglio specificate

3 possibili “Ereditarieta’”

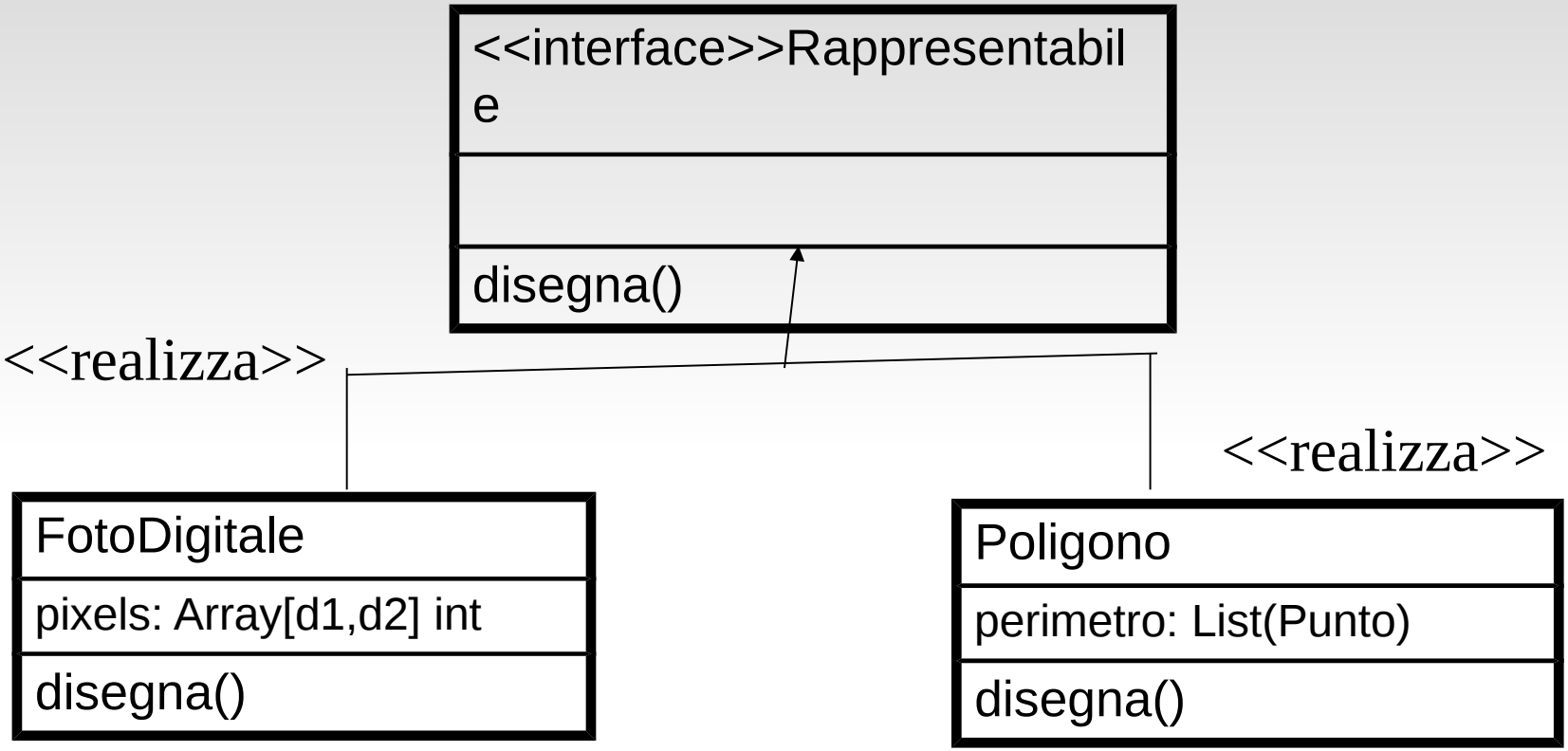
Interfaccia:

- Si usa quando due classi hanno attributi diversi, e realizzano le stesse operazioni in modo diverso
- Es. in un sistema di grafica, l’operazione disegna() e’ completamente diversa per la classe PoligonoRegolare da quella per Fotografia
- Anche se in UML e’ una scatola, l’interfaccia e’ un insieme di proprieta’ (operazioni) che una classe possiede (e deve realizzare): non e’ un oggetto concreto; e' un vincolo applicabile attraverso il linguaggio
- anche tra le interfacce puo’ esistere ereditarieta’
- Alla fine l’interfaccia e’ **realizzata da una classe**.

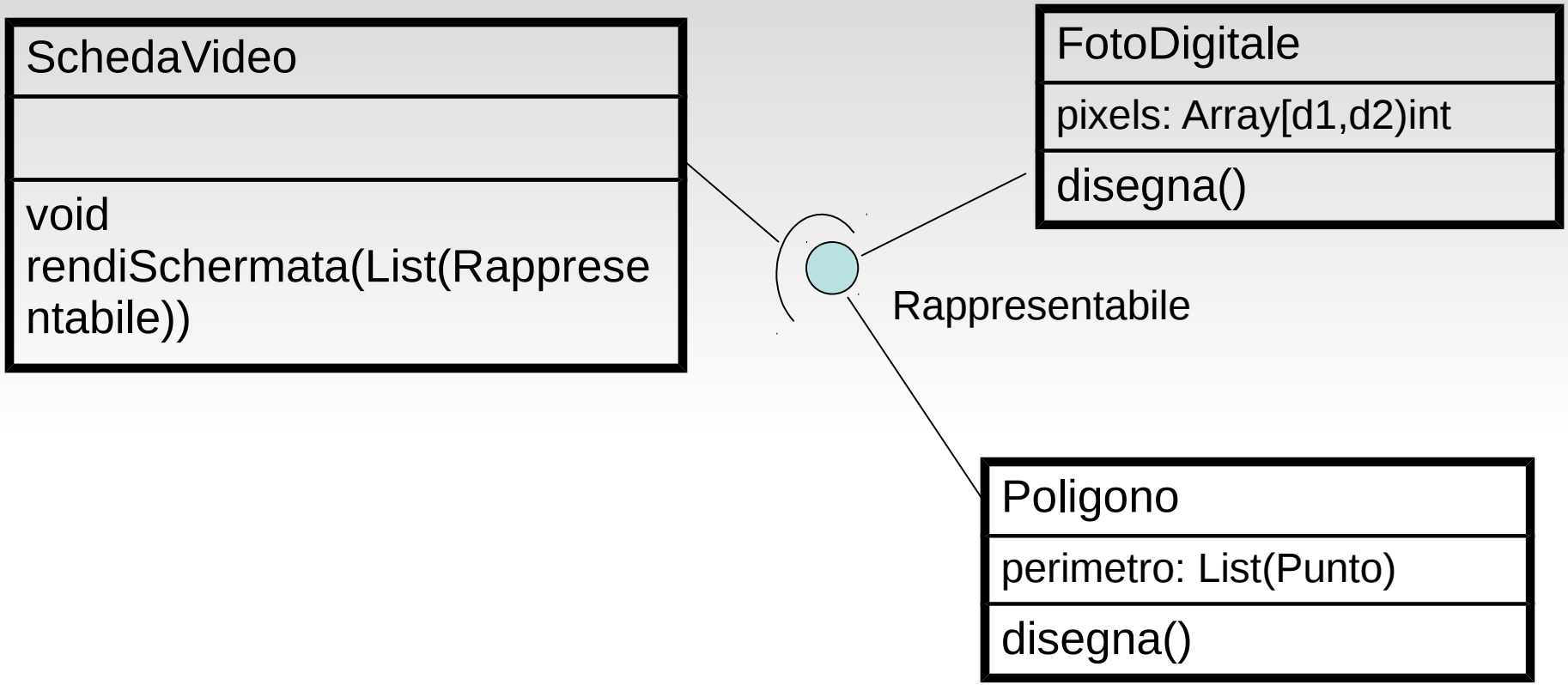
Ereditarieta' nelle interfacce



Relazione di realizzazione



Relazione di realizzazione



Notazione per specificare una interfaccia: il pallino e l'arco.

"SchedaVideo" puo' gestire qualunque oggetto che implementi (realizzi) l'interfaccia "Rappresentabile", nel nostro esempio "FotoDigitale" o "Poligono"

Criteri d’uso della generalizzazione

La generalizzazione e' utile ma puo’ causare problemi causati dall’ereditarieta’

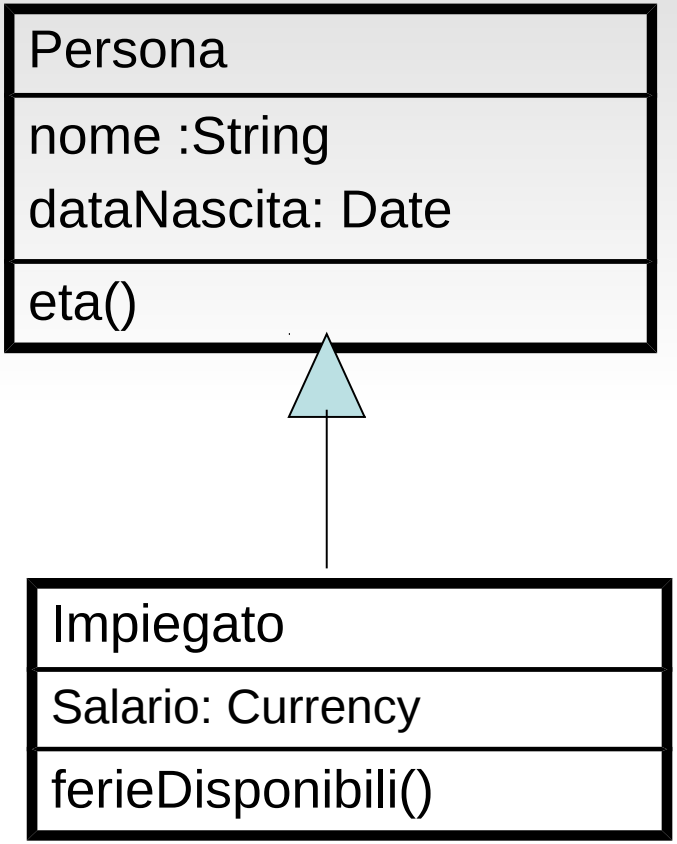
Stabilisce che una sottoclasse deve includere tutte le proprieta’ della superclasse

L’ereditarieta’ stabilisce che elementi piu’ specifici incorporano strutture e comportamento definiti da elementi piu’ generali

L’utilita’ della generalizzazione deriva dal principio di sostituibilita’: l’oggetto di una sottoclasse puo’ rimpiazzare quello di una superclasse ovunque esso sia usato

Vi sono tuttavia vari modi di usare impropriamente l’ ereditarieta’

Ereditarieta’ ed estensione



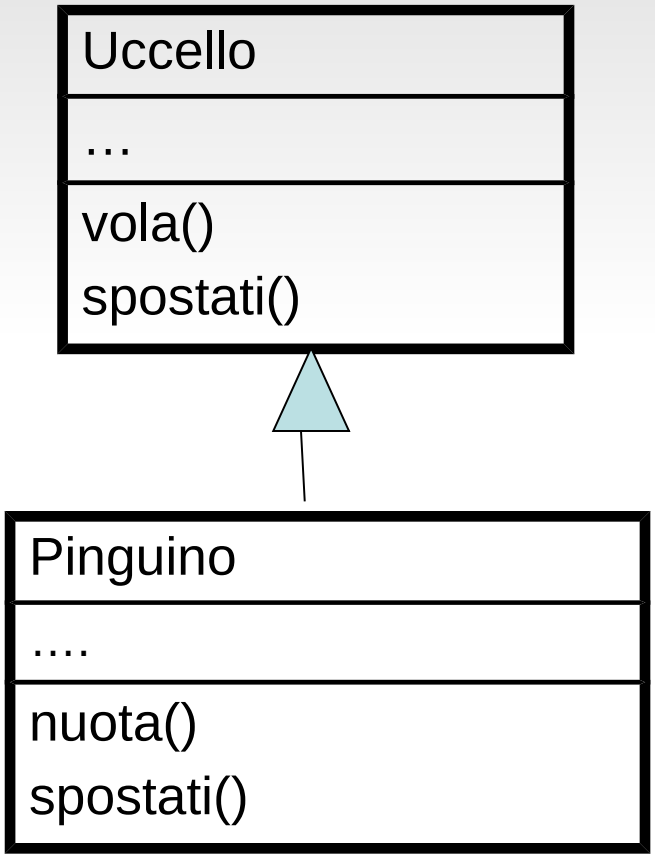
Definizione incrementale di una classe

Uso appropriato

Le proprieta’ della classe base possono essere ridefinite solo per renderle piu’ specifiche o computazionalmente piu’ efficienti, non per cambiarne significato

Ereditarieta’ e restrizione

Dov'e l'errore?

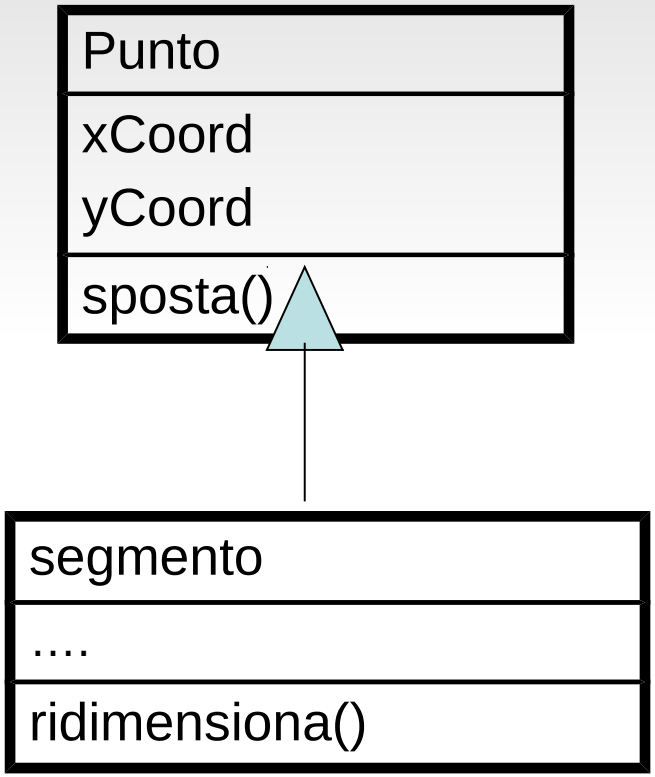


Alcune proprieta’ della superclasse sono eliminate nella sottoclasse

Uso rischioso

Il programma che utlizzera’ Un oggetto della sottoclasse al posto della superclasse **deve sapere** che mancano delle proprieta’

Ereditarieta’ di comodo



Implementazione simile ma indipendenza tassonomica

Uso improprio

Una classe e’ arbitrariamente scelta come superclasse

Capita quando si dispone di vaste librerie che implementano molte funzionalita’

Problemi nell'ereditare l'implementazione

Superclasse fragile, se viene modificata

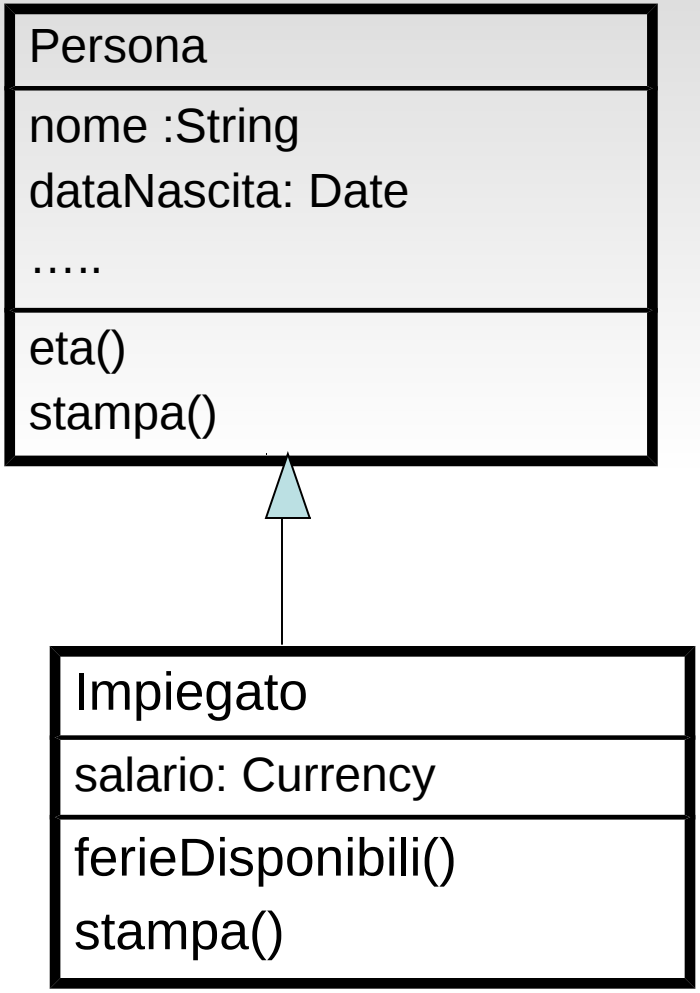
Si ereditano le operazioni che potrebbero essere state modificate

- Soluzione parziale e' mantenere sia la versione vecchia dell'operazione che la nuova, ed avvisare la sottoclasse che la funzionalita' e' cambiata (deprecated)

Ereditarieta' multipla

- Rende ancor piu' complessa la gestione di quale implementazione ereditata si utilizza
- Se invece si ereditano le interfacce, si fondono semplicemente le definizioni

Ereditare l'implementazione



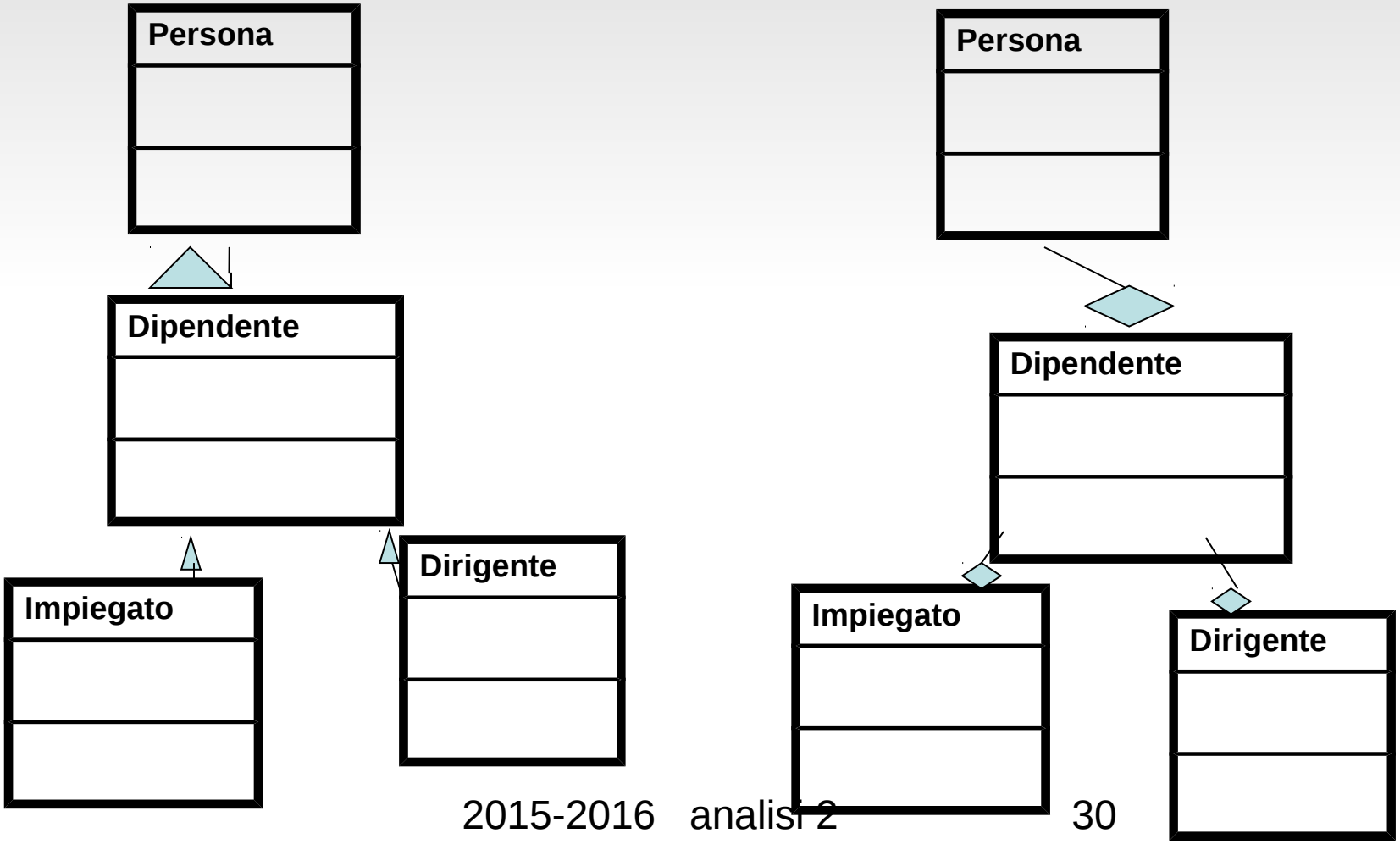

```
void Impiegato.stampa () {
salario.print();
super.stampa();//Persona.stampa()
}

void Persona.stampa() {
nome.print();
dataNascita.print();
}
```

schema molto usato, anche se I dati dell'antenato fossero "public"

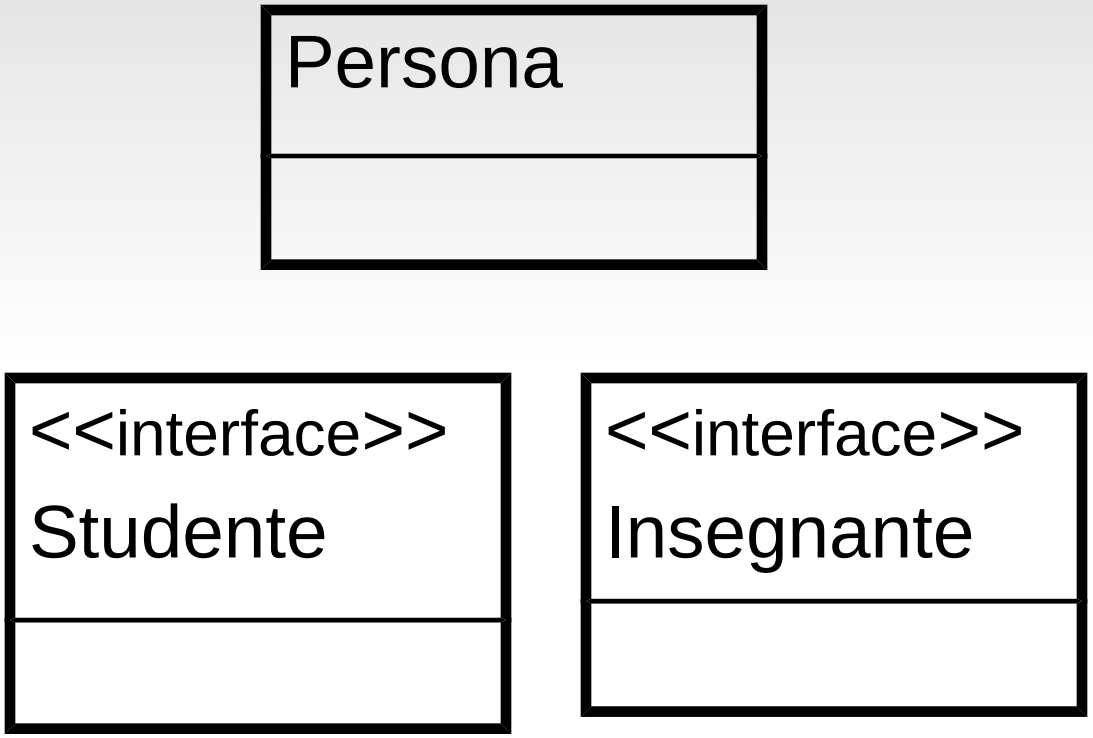
Aggregazione e generalizzazione

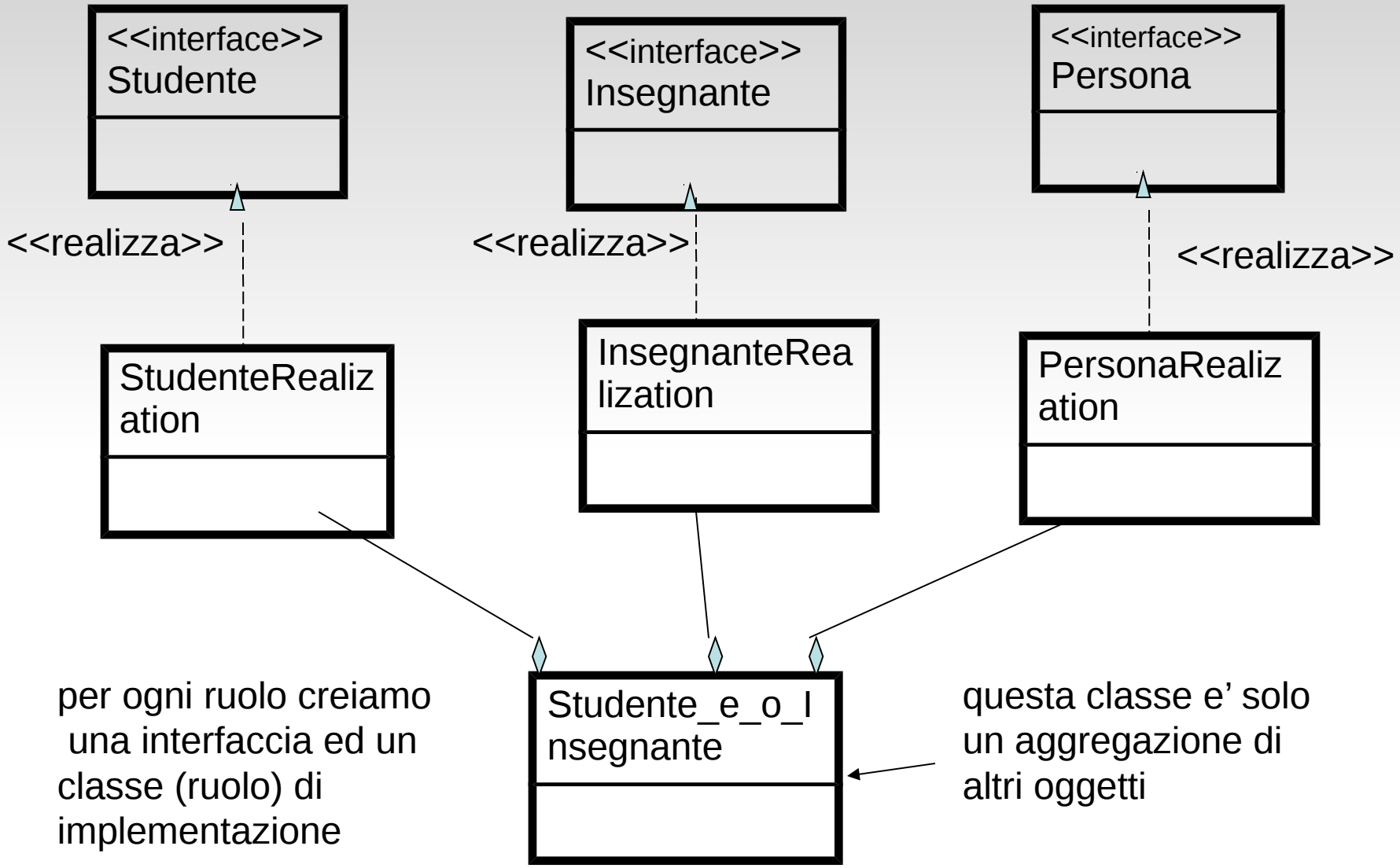
La generalizzazione usa l'ereditarieta', mentre la aggregazione usa la delegazione per implementare la semantica dei componenti



Esempio di composizione

Supponiamo che una Persona possa avere due ruoli studente e insegnante, permettiamo che un individuo possa essere entrambi.

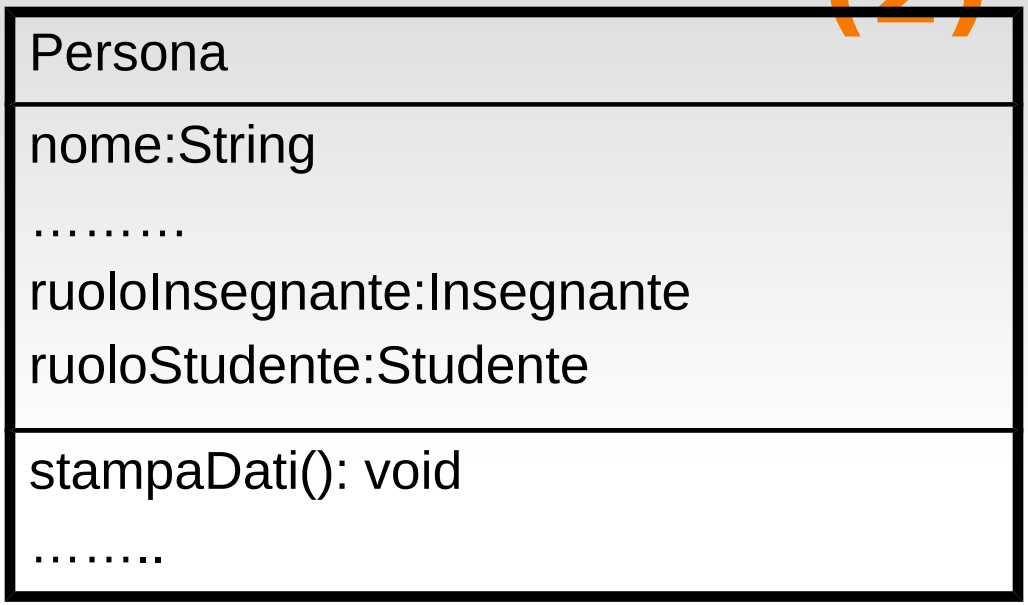


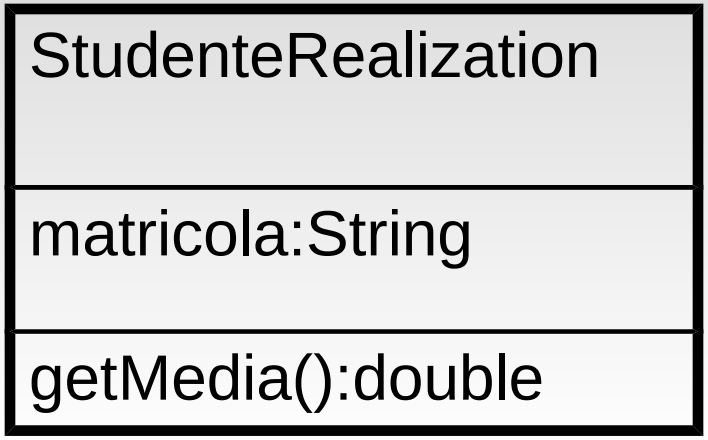


per ogni ruolo creiamo una interfaccia ed un classe (ruolo) di implementazione

questa classe e' solo un aggregazione di altri oggetti

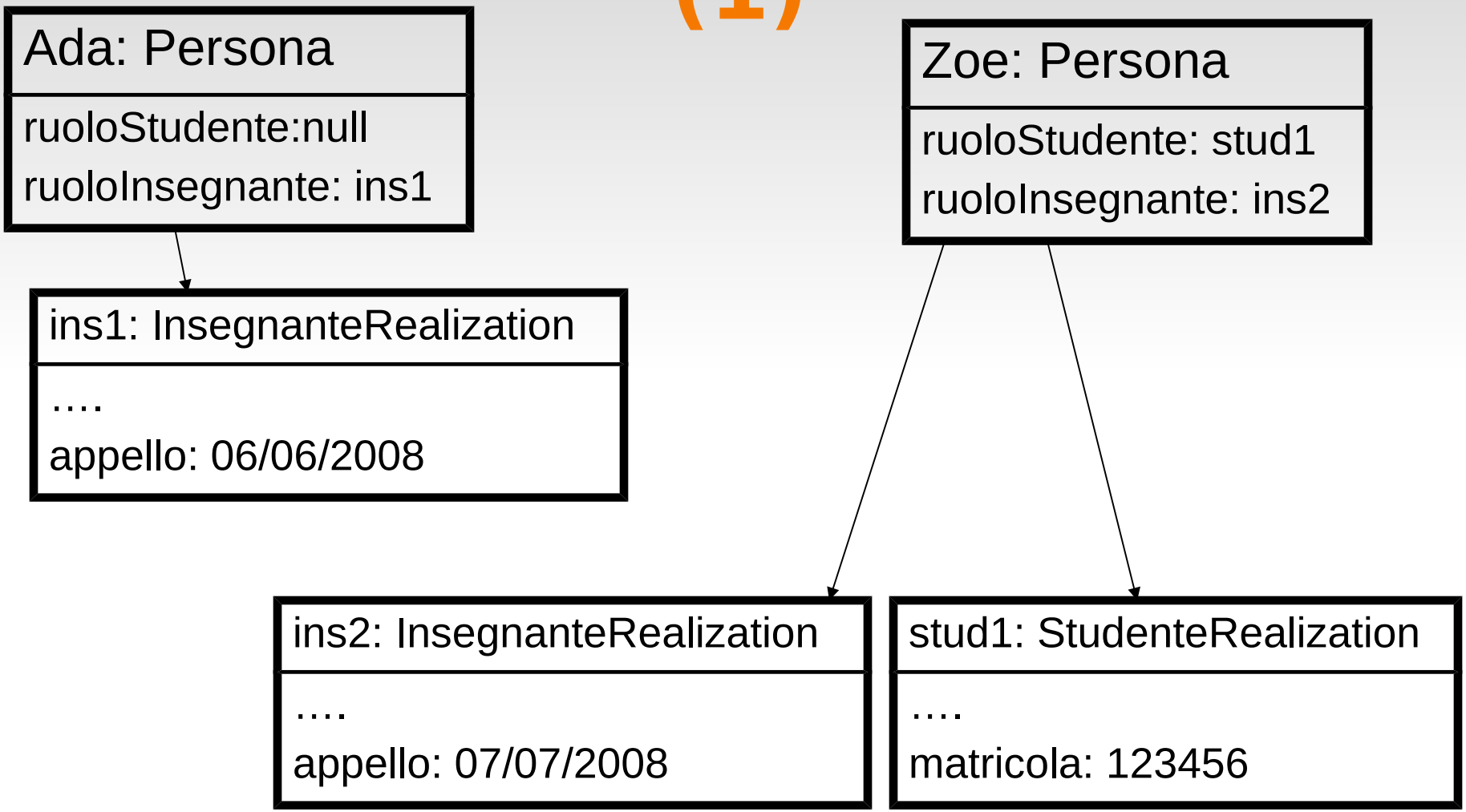
Esempio di composizione (2)





in quest'altra versione abbiamo deciso che una classe base "Persona" realizza direttamente le operazioni; sparisce l'interfaccia "Persona"

Esempio di composizione (1)



Commenti all’esempio

abbiamo creato delle interfacce per poter estendere il diagramma (es. aggiungere “Impiegato”)

Se pensassimo di avere un cane da guardia che va a scuola, “Persona” si potrebbe cambiare in “Individuo”: occorrerebbe cambiarne gli attributi

il meccanismo di invocazione di un metodo non e’ piu’ automatico. Ogni singola istanza ha diverse proprieta' (risponde a diversi messaggi) a seconda che contenga l'oggetto specifico

Quando usare l’ ereditarieta’

Se la relazione e’ di tipo is-a stabile (mela-frutto)e l’oggetto **non cambia proprieta’**

- In genere **ereditarieta’**
- La classe base deve contenere proprieta' stabili

Se invece puo’ perdere o acquisire proprieta’ persona-impiegato (ruolo)

- in genere **composizione**
- ovviamente si puo’ combinare ereditarieta’ e composizione (esempio slide studente-docente)

Non conviene quando

- occorre rappresentare ruoli della superclasse che cambiano dinamicamente
- Per nascondere metodi o attributi ereditati
- Per implementare una classe specifica ad un dominio come sottoclasse di una di servizio

Quando fermarsi ?

dopo che si e' modellato dettagliatamente il sistema, e' un processo prevalentemente di semplificazione

- abbiamo due tipi di elementi di base: dati ed operazioni
- occorre aggregarli in modo da massimizzare la "semplicita' " del sistema
  - Gli attributi in classi
  - le operazioni in classi ed interfacce (in insiemi correlati logicamente es. i ruoli)
- fattorizzare I dati e le operazioni con la stessa realizzazione per evitare duplicazioni (ereditarieta')
- associare nel modo piu' opportuno le operazioni ai dati

sommario

- Gli stereotipi sono il meccanismo di estensione di UML per risolvere problemi specifici di rappresentazione
- Gli attributi di visibilita' controllano il livello di incapsulamento
- UML fornisce inoltre i concetti di attributi ed associazioni derivate, qualificate, classi e associazioni "reificate"
- La generalizzazione e' uno strumento potente da usare in modo appropriato se si vogliono ridurre i problemi di manutenibilita'
- La aggregazione con la delegazione e' una possibile alternativa alla generalizzazione

Esercizio: modellare poligoni regolari e non, orario

Poligoni equilateri ed equiangoli

casi speciali a 3 e 4 lati

calcolo perimetro ed area

Riuso del codice

delegazione