UML: Class Diagram

Ing. Orazio Tomarchio
Orazio.Tomarchio@diit.unict.it

Dipartimento di Ingegneria Informatica e delle Telecomunicazioni
Università di Catania

Class Diagram

- Forniscono una vista strutturale (statica) del sistema in termini di
  - **Classi**
    - **Attributi**
    - **Operazioni**
  - **relazioni tra classi (associazioni, generalizzazioni, ...)**

Un class diagram rappresenta uno schema concettuale

- se una classe A è in relazione con una classe B, allora ogni istanza di A sarà in relazione con un'istanza di B

Prospettive diverse

- **La prospettiva con cui si realizza il diagramma può essere**
  - concettuale
    - studia i concetti propri del dominio sotto studio, senza preoccuparsi della loro successiva implementazione
  - di specifica
    - studia il software ma a livello di interfaccia e non di implementazione. Quindi l'attenzione è concentrata sulle responsabilità delle classi ma non sui dettagli concreti
  - implementativa
    - il diagramma fa riferimento alle classi effettivamente realizzate con un linguaggio di programmazione OO e alle strutture dati effettivamente impiegate.

Similitutidini

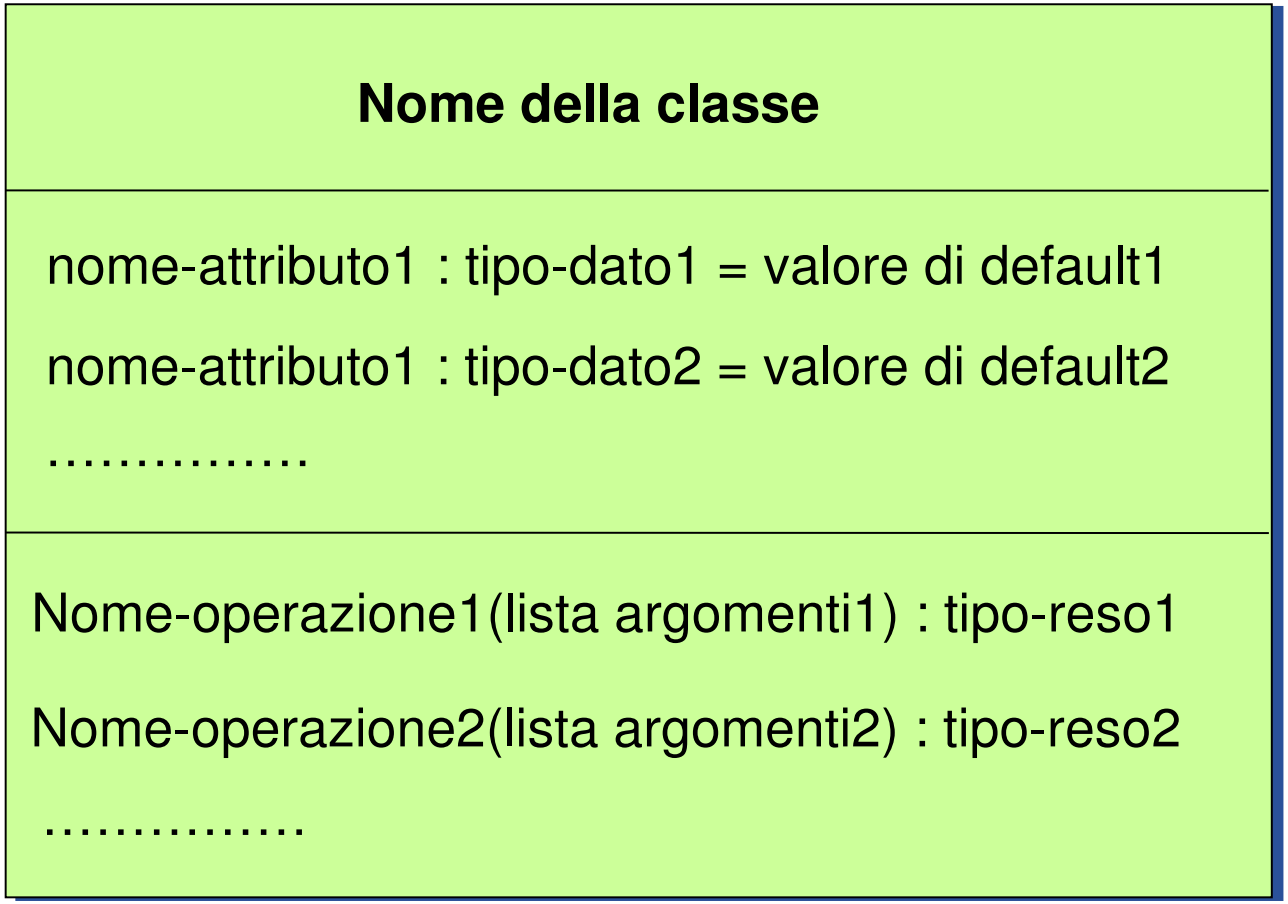
- **Entità / Relazioni**
  - E’ una estensione dei diagrammi Entità / Relazioni.
  - Introduce classificazione, istanziazione e aggregazione.
  - Definisce non solo gli attributi, ma anche le operazioni.
- **Altri modelli OO (Booch, OMT)**
  - Differenze sintattiche

Concetto di Classe

- Una classe descrive un gruppo di oggetti con caratteristiche comuni
  - **attributi**
  - **comportamento**
- Gli oggetti (istanze) di una stessa classe differiscono tra loro per i valori degli attributi e per le relazioni che li legano ad altri oggetti.

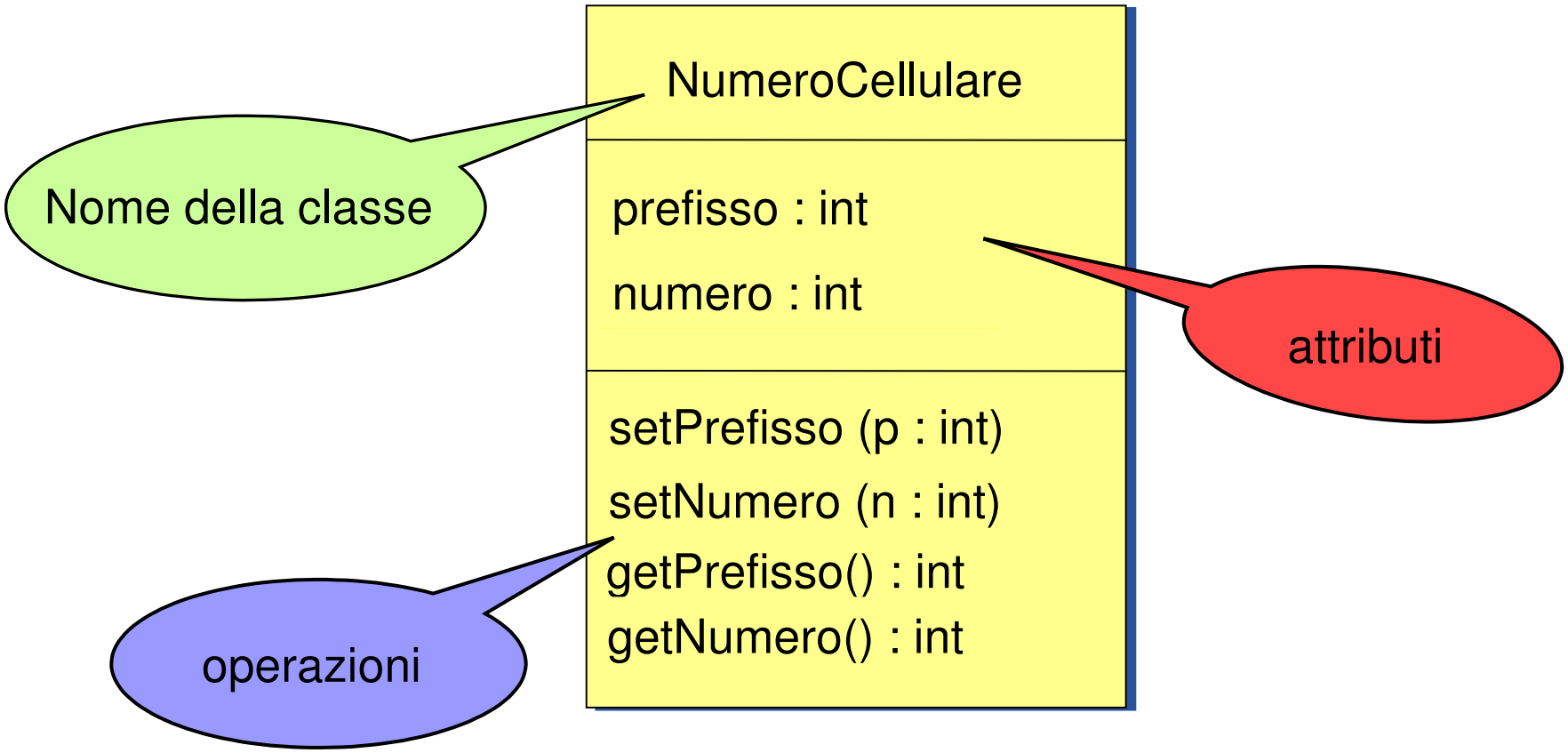
Notazione generale per le classi

- Nel caso più generale la notazione è la seguente.



Esempio1

- **Vediamo un esempio**



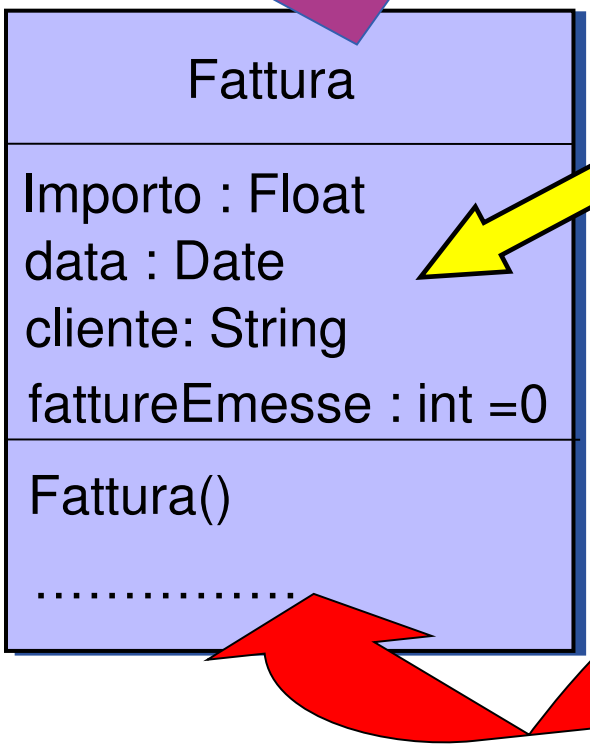
Esempio2

- **Vediamo ancora un esempio**

- Dal punto di vista **concettuale** si può inizialmente tralasciare di indicare tutti gli attributi e tutte le operazioni

Classi : implementazione

- Esiste una corrispondenza tra la rappresentazione UML di una classe e l'implementazione con un linguaggio O-O (Es. Java)




```
public class Fattura {
public Float importo;
public Date data = new Date();
public String cliente;
static private int fattureEmesse = 0;
public Fattura() {
fattureEmesse++;
}
// Altri metodi
// ...
}
```

Relazioni UML

- In un Class Diagram vengono rappresentate relazioni oltre alle classi:
  - **Relazioni di generalizzazione**
  - **Associazioni (semplici, aggregazioni, composizioni)**

Generalizzazione ed ereditarietà

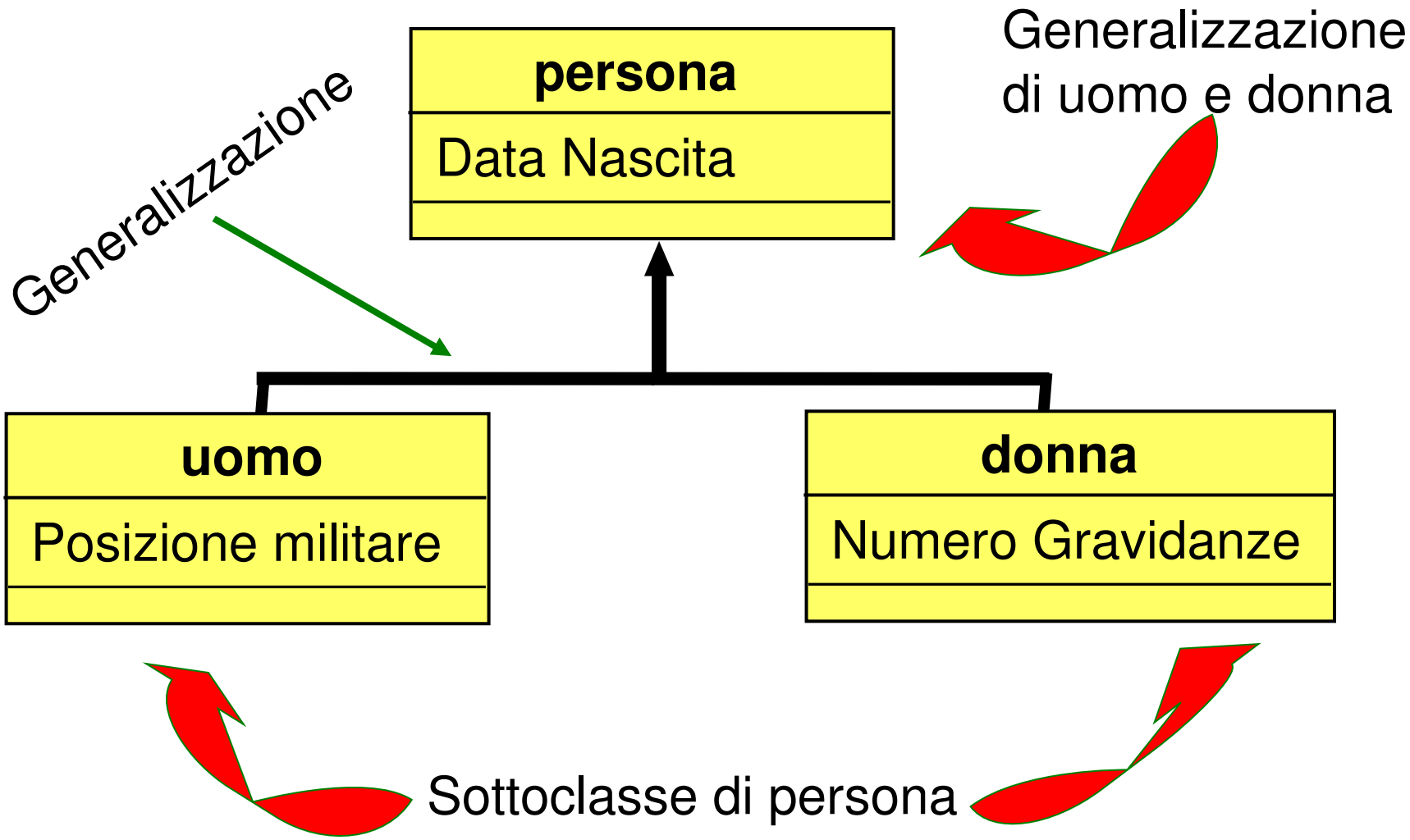
- Generalizzazione = relazione "is-a“
  - Ogni istanza di una classe è anche istanza di tutte le superclassi
  - La relazione di generalizzazione può essere utilizzata anche fra altri elementi del linguaggio UML (packages, use cases, etc.)
  - Ereditarietà
    - Meccanismo attraverso il quale elementi specializzati incorporano la struttura ed il comportamento di elementi più generali

Generalizzazione fra classi - I

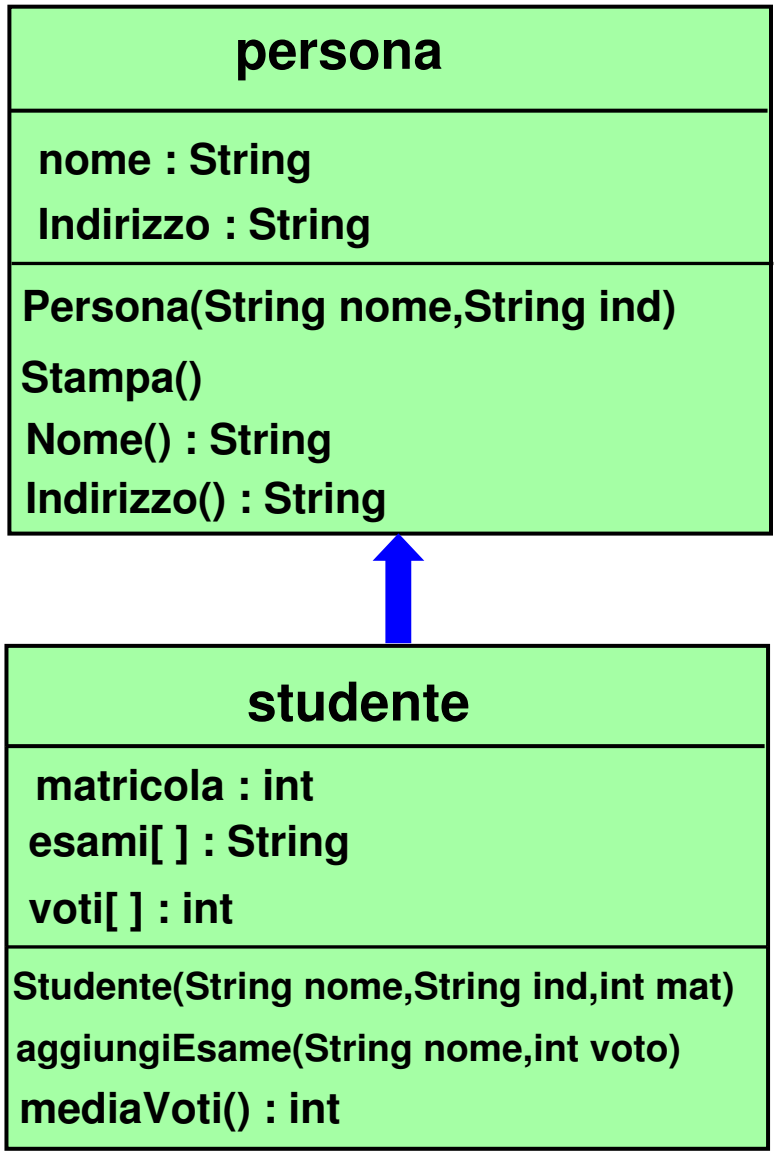
- Le classi figlie (o sottoclassi) ereditano gli attributi della classe padre (superclasse) e ne possono aggiungere altri.

- Inoltre esse ereditano anche i metodi, ma essi possono essere ridefiniti nella classe figlia
  - (cfr. con il concetto di polimorfismo dei linguaggi di programmazione OO).

Esempio(1) : generalizzazione



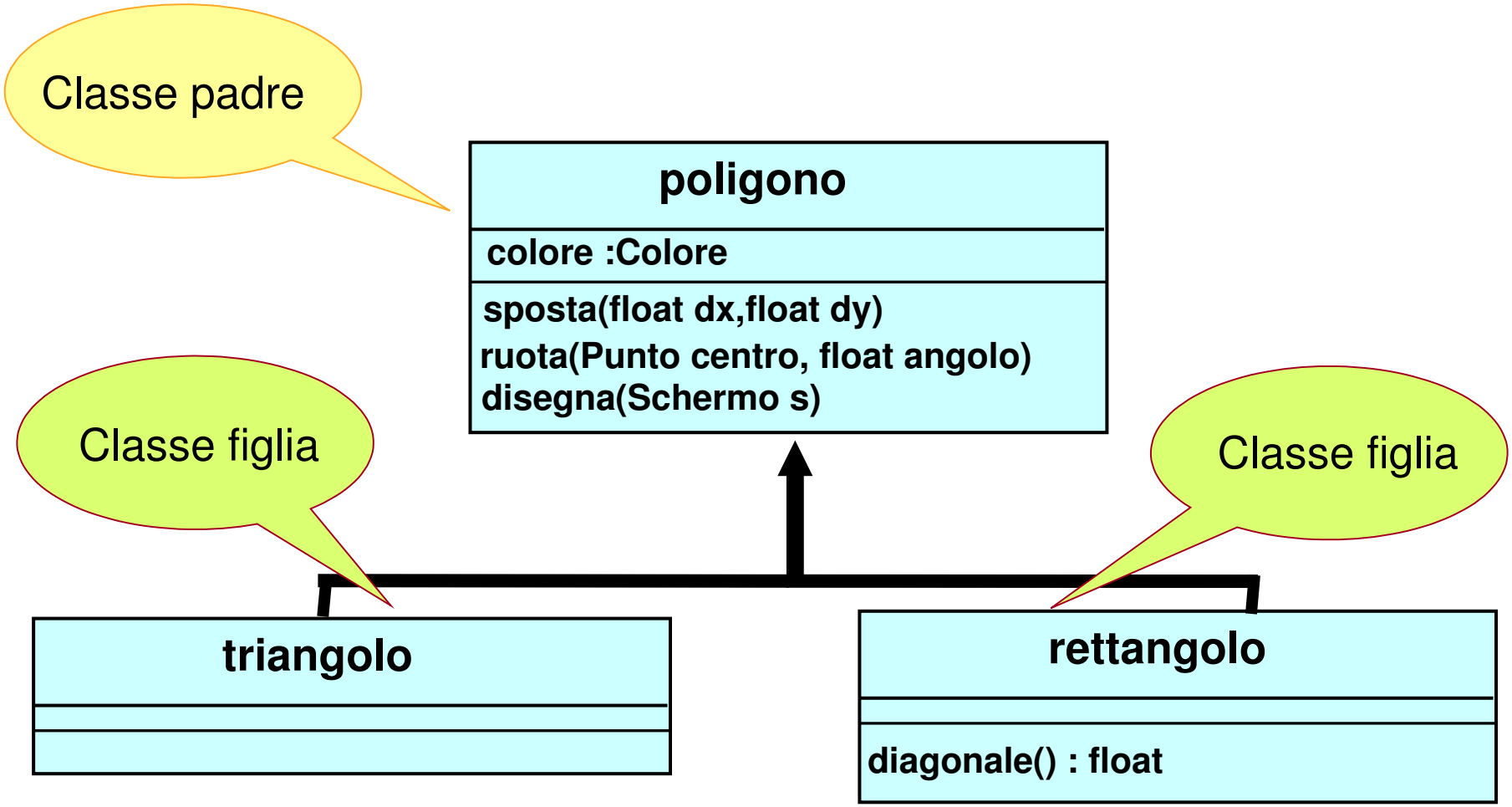
Esempio(2) : generalizzazione



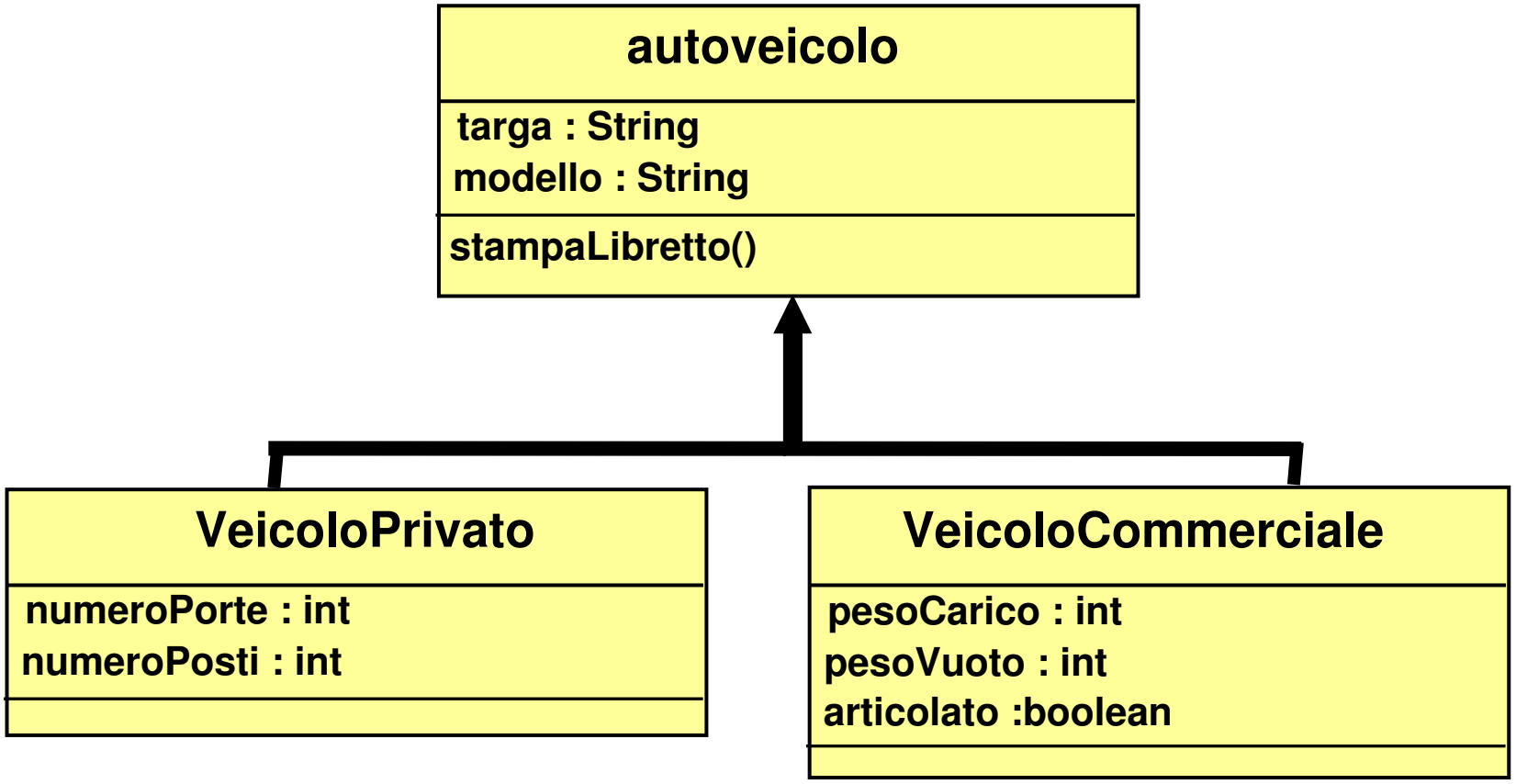
- la classe Studente eredita dal padre:
  - **attributi**
  - **metodi**
- Un oggetto Studente può essere trattato esattamente come un oggetto Persona

- In cosa solitamente può differenziarsi la classe erede ?
  - **aggiunta di attributi e metodi**
  - **i metodi possono essere ridefiniti**

Esempio(3) : generalizzazione



.....ancora un esempio



Principio di sostituzione

OSSERVAZIONE

- **"In una** generalizzazione**, una classe figlia deve rappresentare un valido sostituto della classe padre.**

  Cioè, in qualunque punto del codice appaia la classe padre, deve essere possibile sostituirla con la classe figlia.

  Il viceversa, invece, non è vero."

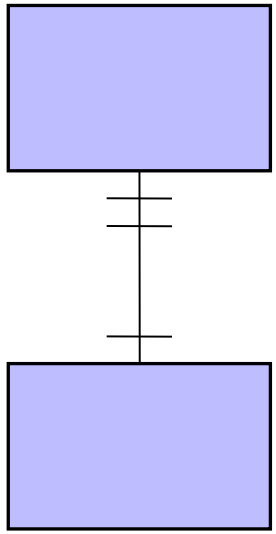
  Questo principio può essere utile per individuare errori nelle generalizzazioni presenti nello schema.

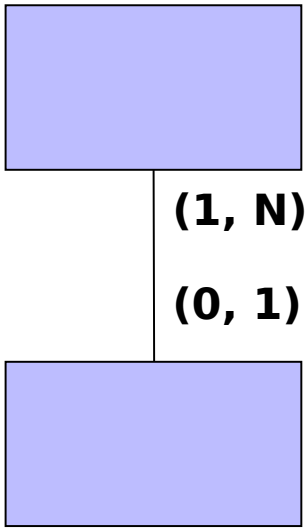
Associazioni

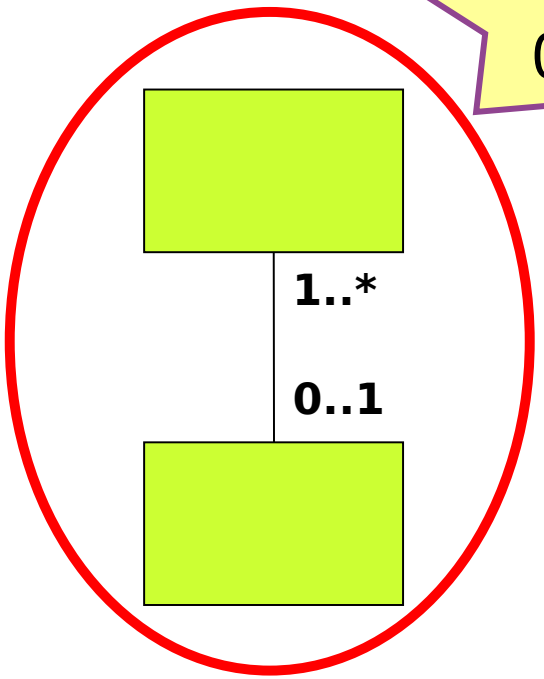
- Una Associazione individua una ”connessione” logica tra classi
  - **si traduce in una connessione fra oggetti, istanze delle classi coinvolte nell’associazione**
- Il concetto di associazione è presente anche nella modellazione concettuale di database (Entity-Relationship)

Cardinalità nelle associazioni

- Indica il numero di istanze di una classe che possono essere associate ad una singola istanza dell'altra classe
- Esistono molto convenzioni diverse per indicare la molteplicità di una associazione



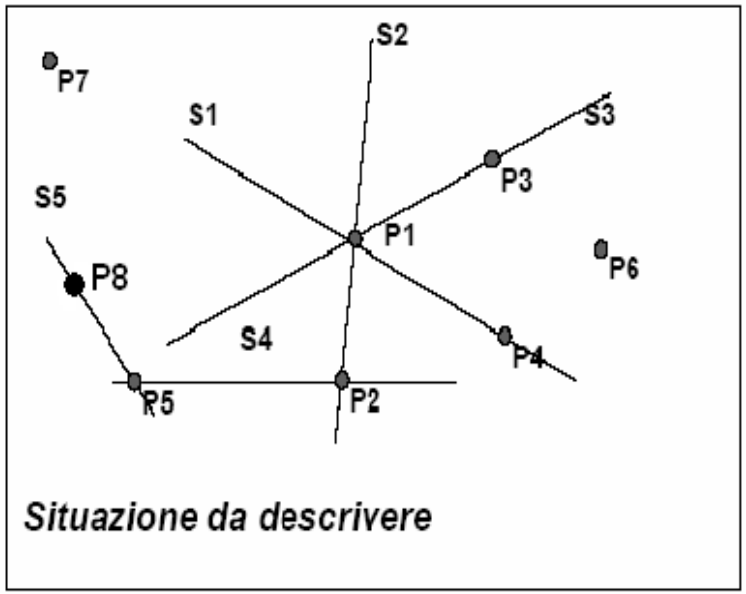




1..1 = 1

0..* = *

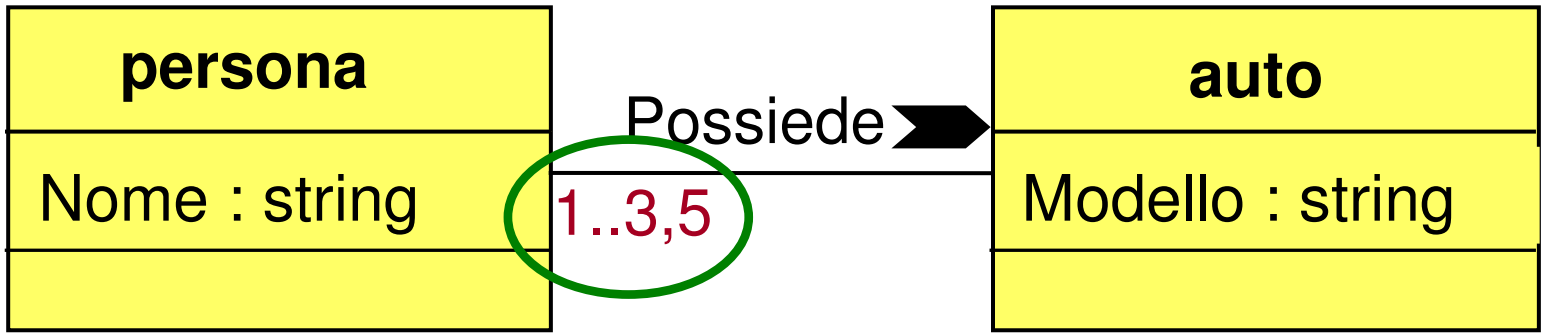
Indicazione molteplicità



Molteplicità
(anche opzionale)

Notazione Cardinalità

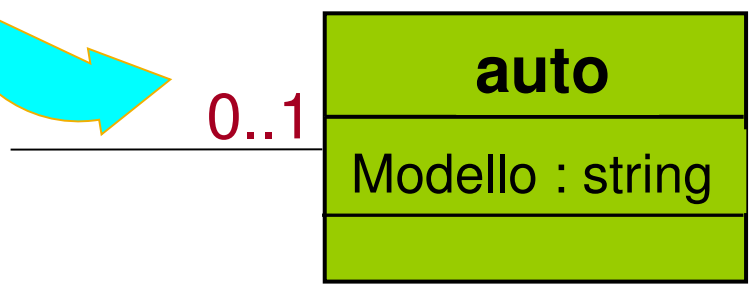
- La molteplicità si può indicare con precisione.



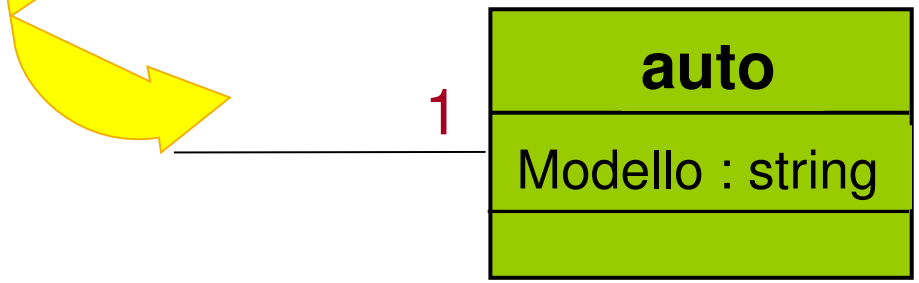
- Ogni auto ha 1, 2, 3 o 5 (ma non 4) comproprietari
- Ogni persona può possedere zero o al più un'auto (magari in comproprietà)

Notazione per la cardinalità (1/3)

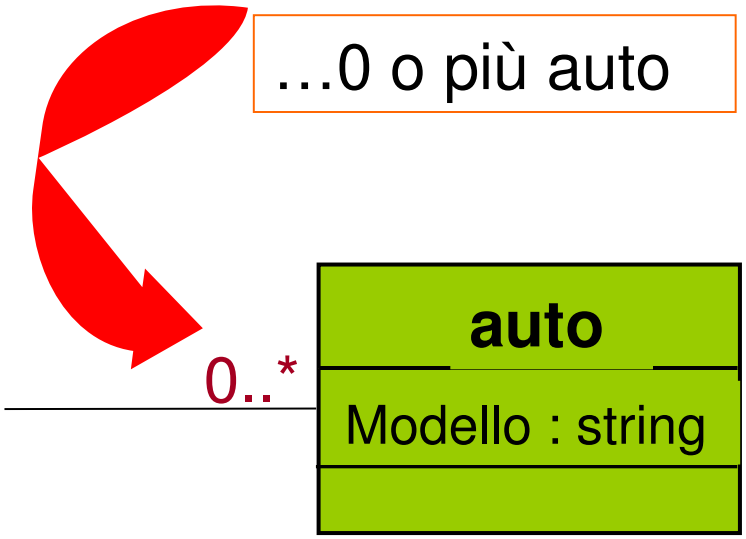
…0 o un'auto



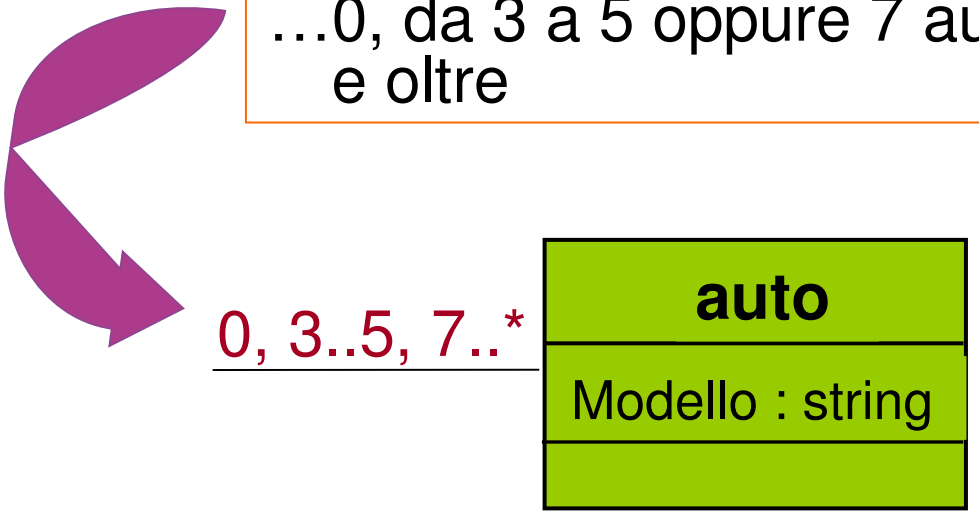
…esattamente un'auto



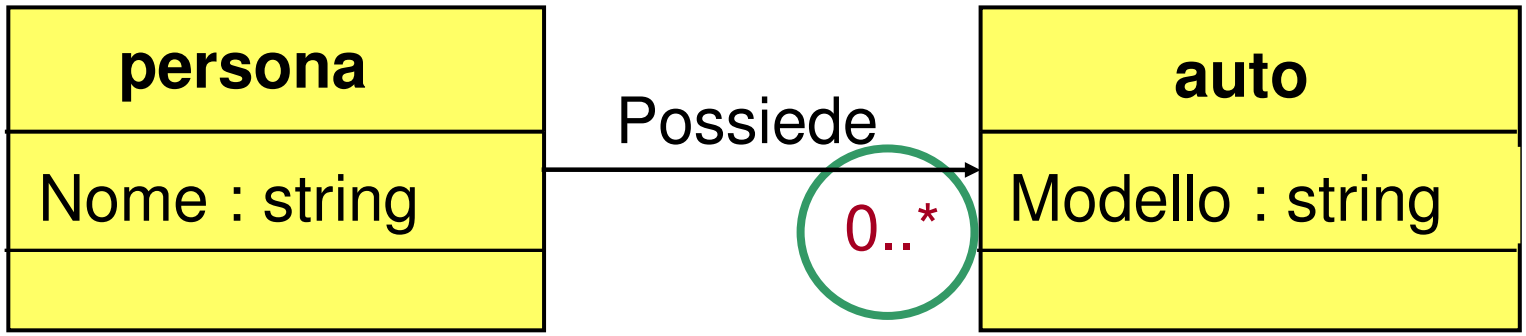
…0 o più auto



…0, da 3 a 5 oppure 7 auto e oltre



Notazione per la cardinalità (2/3)



- Nota: Il class diagram indica che una persona può possedere un numero qualsiasi di auto
- Quanti sono i proprietari di una singola auto?

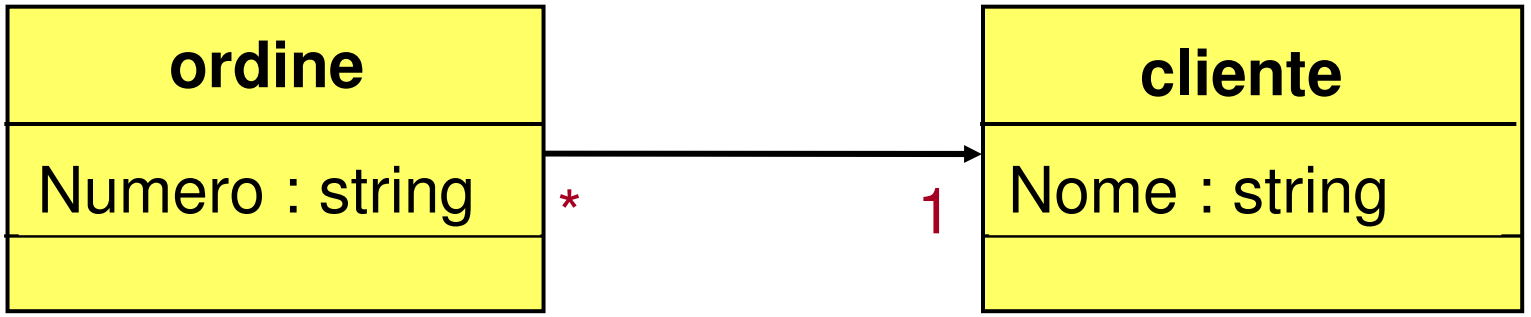
Notazione per la cardinalità (3/3)

- Il diagramma non fornisce questa informazione, dato che la cardinalità della partecipazione di Persona all'associazione risulta non specificata!

- Inoltre, dato che l'associazione è navigabile in una sola direzione, data un'auto non è mai possibile risalire ai suoi (o al suo) proprietario

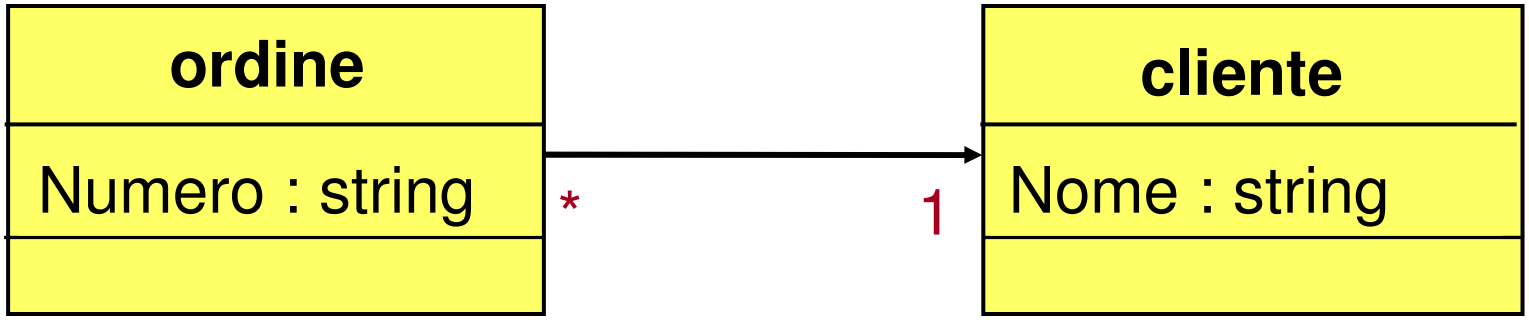
Navigabilità - I

- **La navigabilità di una associazione è un verso privilegiato per essa, e si indica con una freccia sulla riga dell'associazione**



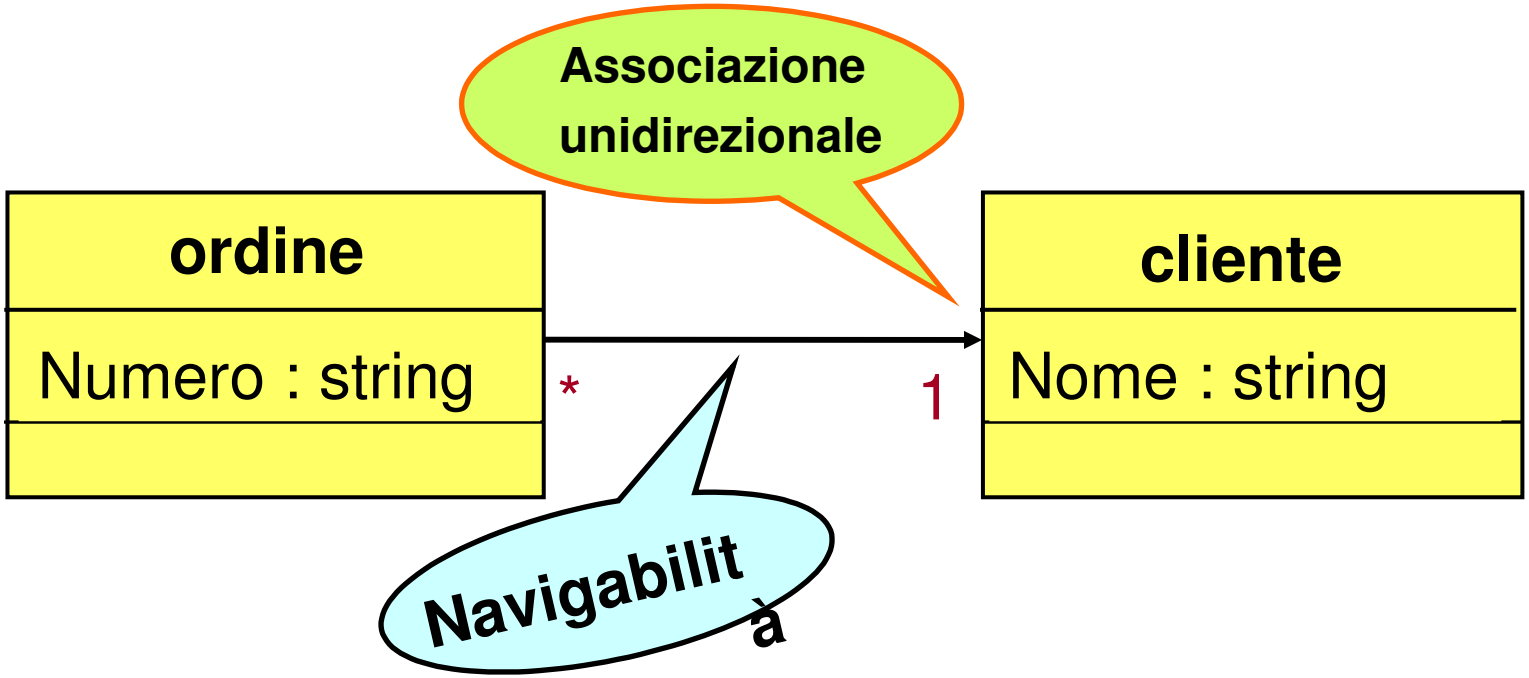
Navigabilità - II

- **A livello di specifica la freccia significa che ogni Ordine ha la *responsabilità* di segnalare a quale Cliente appartiene (non si dice come), ma non il viceversa**
- **A livello di implementazione la freccia indica la presenza di un puntatore da ogni Ordine al rispettivo Cliente**
- **A livello concettuale la navigabilità non ha molto senso e quindi spesso non se ne fa uso**

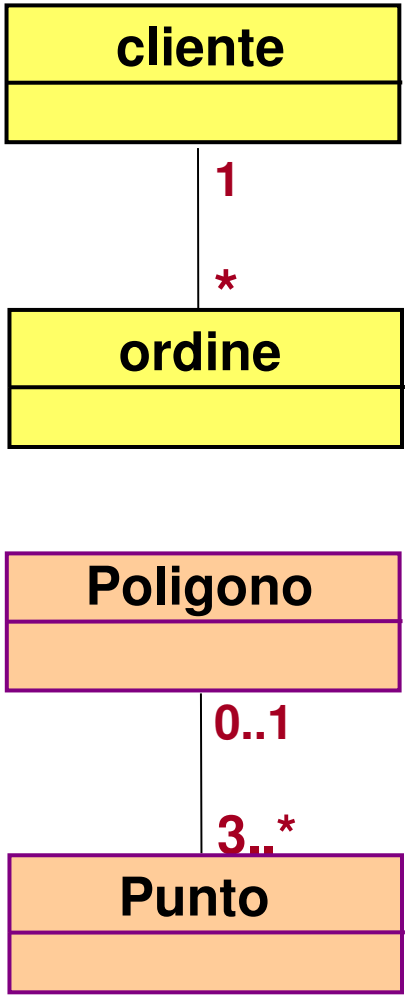


Navigabilità - III

- Se una associazione è navigabile in un solo senso è detta associazione unidirezionale, altrimenti è bidirezionale.
- Se non è presente nessuna freccia sull'associazione, si può intendere che la navigabilità sia bidirezionale oppure non determinata.



Molteplicità (es.)

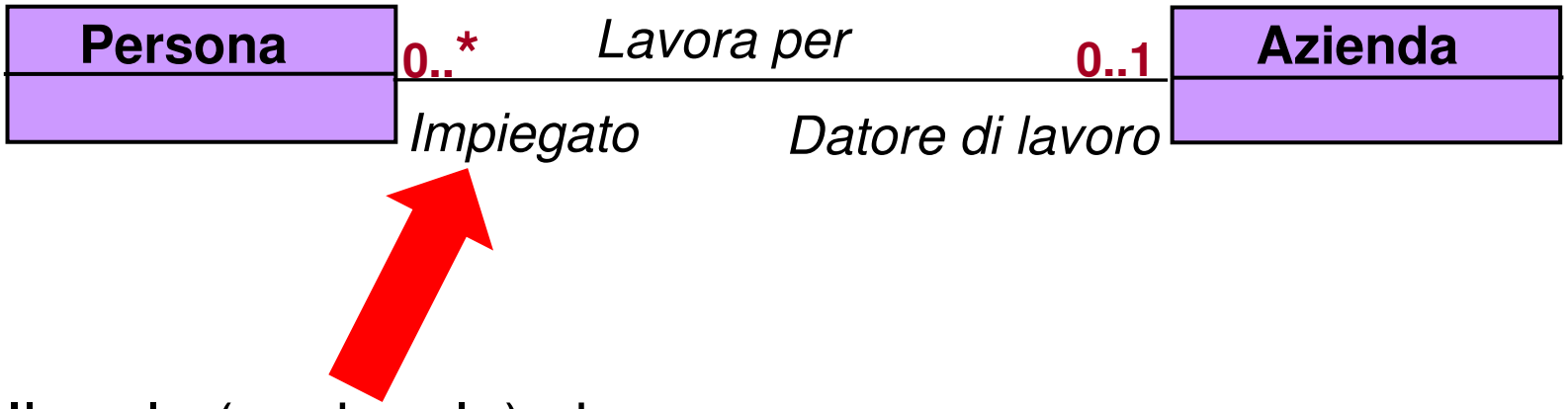


Un cliente può fare quanti ordini desidera, anche nessuno (*). Al contrario ogni ordine deve essere associato ad uno e un solo cliente. Cioè la classe *Ordine* partecipa obbligatoriamente all'associazione.

Un poligono deve avere almeno tre punti (3..*). Al contrario, nell'associazione rappresentata, ogni punto può far parte di un poligono o di nessuno, ma non di molti. Si dice che la classe *Punto* partecipa facoltativamente all'associazione.

Ruoli nelle associazioni

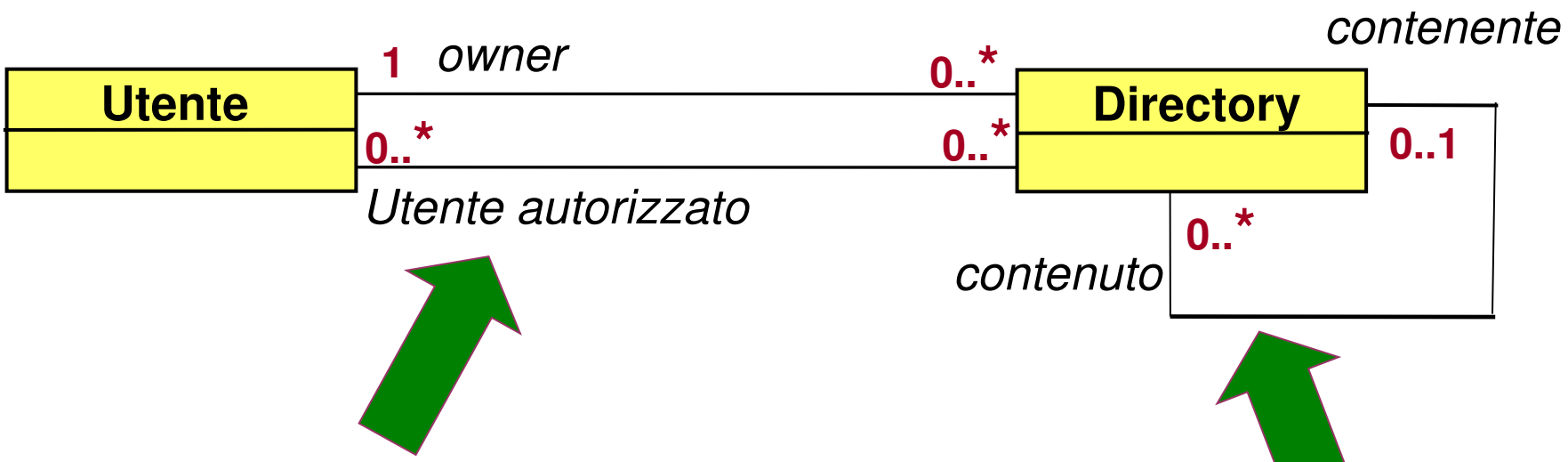
- Una classe può partecipare ad un'associazione con un ruolo specifico, che può essere indicato



Il ruolo (opzionale) viene indicato in stile normale

Ruolo obbligatorio nelle associazioni

- Quando le stesse classi sono coinvolte più volte dalla stessa associazione il ruolo diviene obbligatorio.



Qui il ruolo serve a distinguere due associazioni diverse tra le stesse due classi

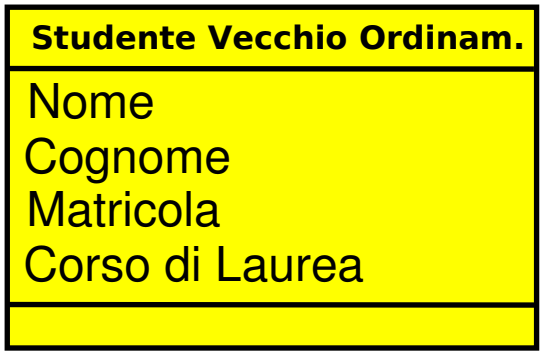
Qui il ruolo è obbligatorio

Vincoli aggiuntivi

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

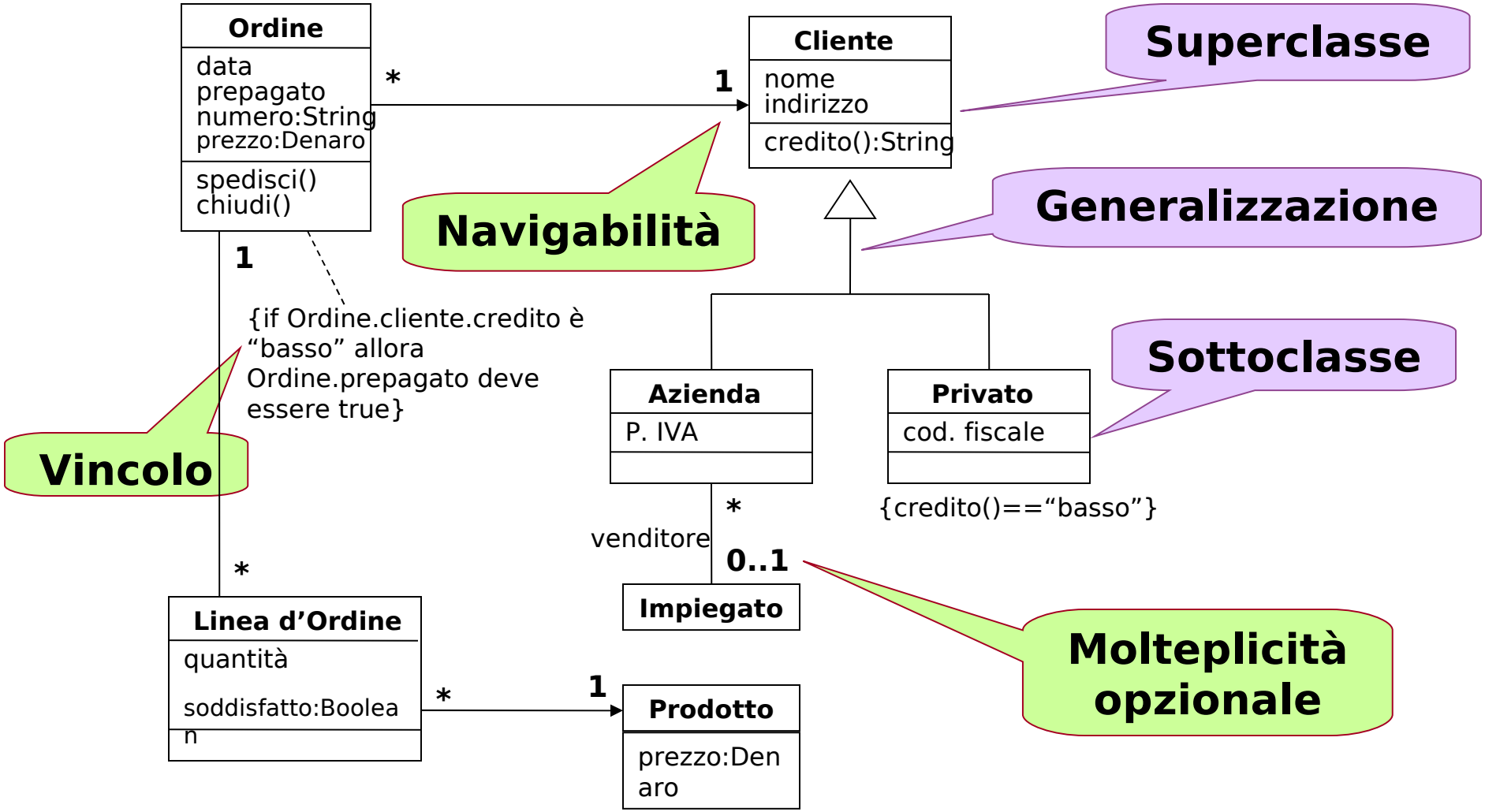
  - Si possono definire sui legami e sul valore degli attributi scrivendoli tra graffe a margine della classe o dell'associazione

{*breve descrizione in linguaggio naturale o pseudocodice*}



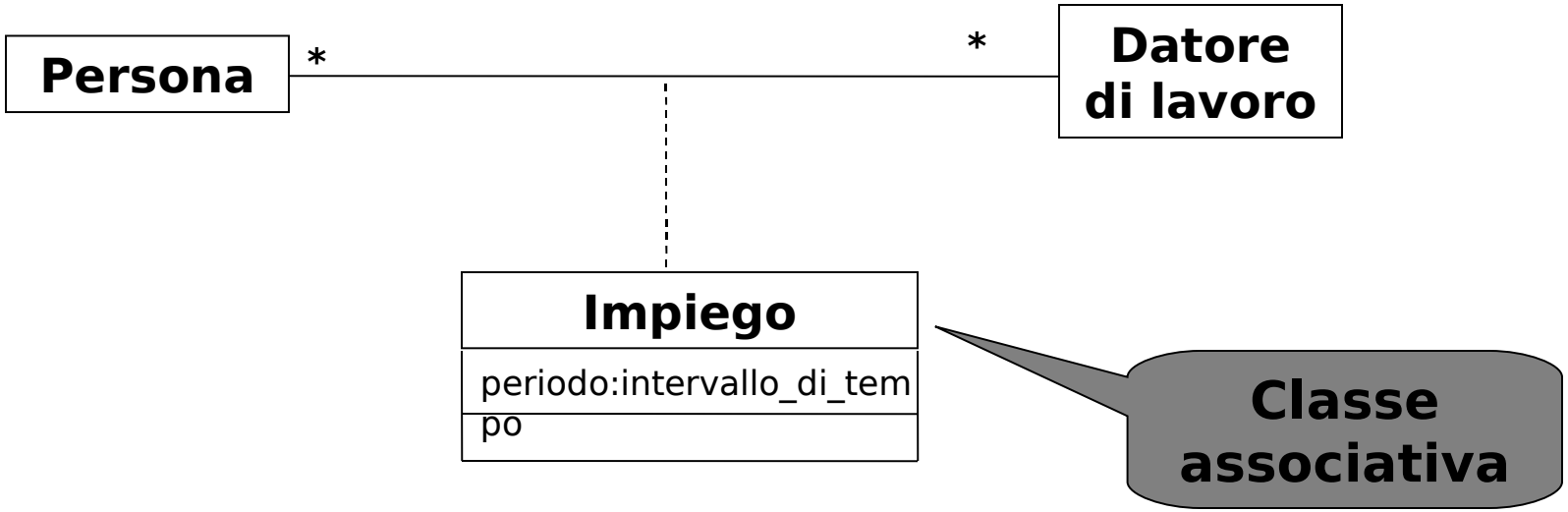
{if Corso di Laurea = Informatica allora Matricola = 0801… }

Class diagram (es.)



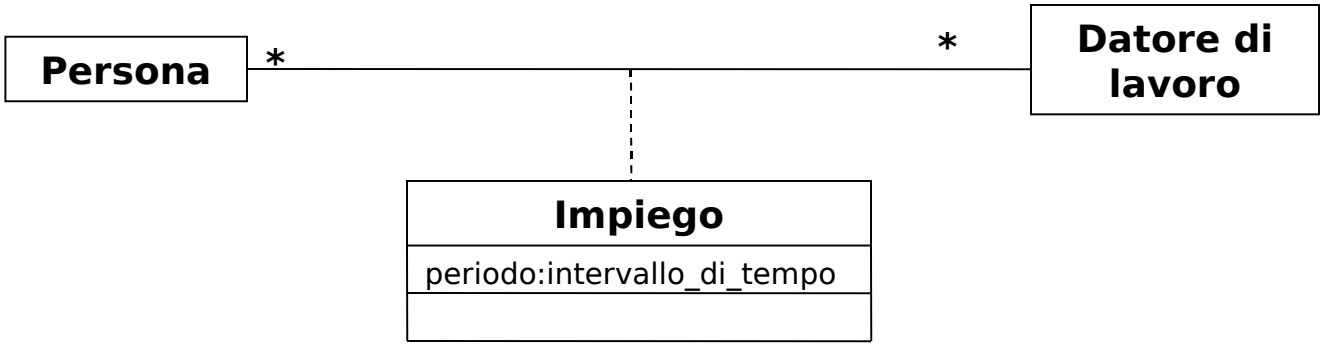
Classi associative - I

- Talvolta sarebbe comodo poter aggiungere attributi ad una associazione piuttosto che alle classi coinvolte. Per far questo c'è il costrutto della *classe associativa*, ossia una classe derivata da una associazione.



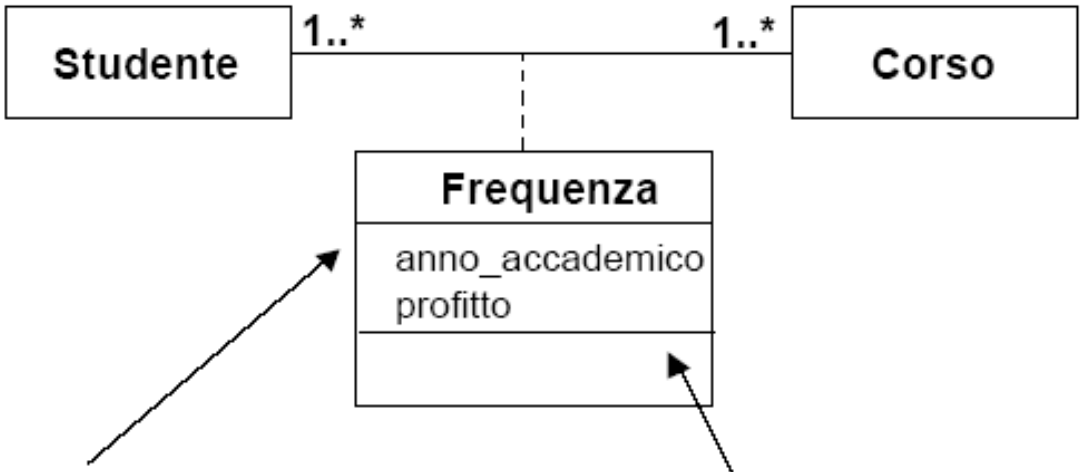
Classi associative - II

Il motivo per cui sono state introdotte le classi associative consiste nel fatto che esse sottintendono un vincolo aggiuntivo, ossia il fatto che *ci può essere solo un'istanza della classe di associazione fra ogni coppia di oggetti associati.*



Classi associative - III

- **Esempio:**

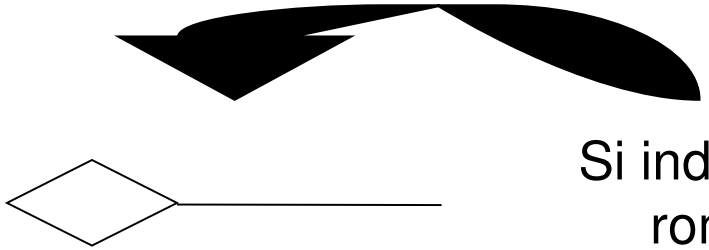


Con questa notazione indichiamo che l'associazione possiede alcuni attributi.

Non si tratta di attributi dello studente perché cambiano da corso a corso. Nè sono attributi del corso (ad es. ogni corso è frequentato da studenti diversi in anni diversi)

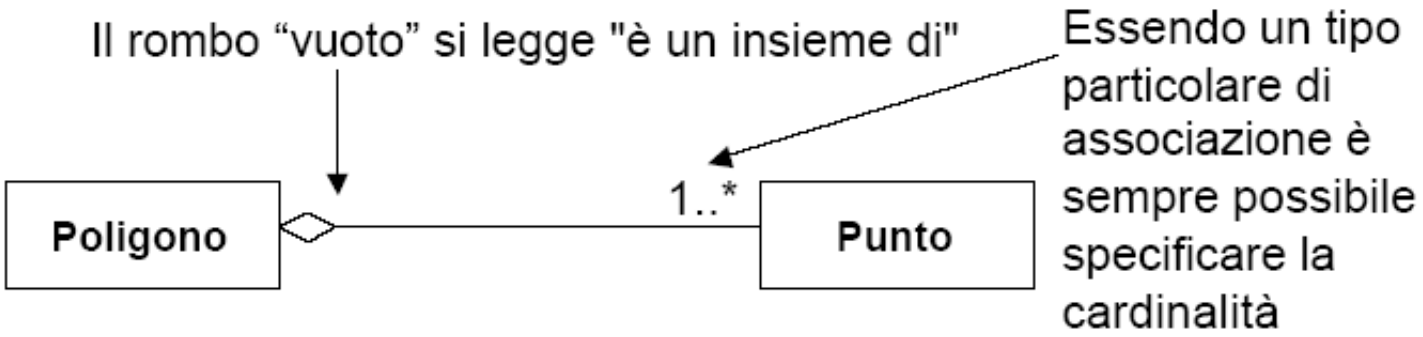
Aggregazione I

- **E' un caso particolare di associazione molto comune che significa: "è un insieme di".**
- **Esempio:** È come dire che un'automobile ha un motore e quattro ruote. Sia il motore che le ruote continuano ad avere dignità ed esistenza propria anche al di là dell'oggetto automobile. La distruzione dell'automobile, non comporta automaticamente quella delle sue parti

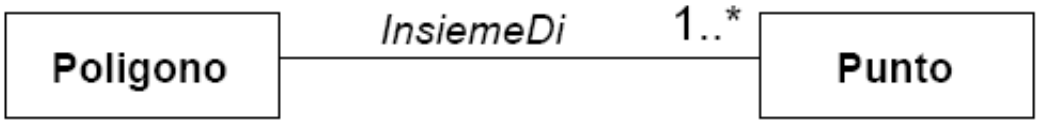


Si indica con un rombo vuoto

Aggregazione II



Modello equivalente che usa una associazione convenzionale:



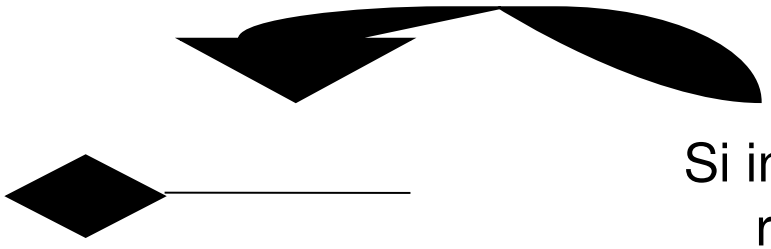
Composizione I

E' un caso particolare di aggregazione che significa:

"è composto da".

- I componenti non possono esistere senza il contenitore
- La proprietà da parte del contenente è esclusiva
- La molteplicità dal lato dell'aggregato deve essere = 1
  - Può essere qualsiasi per gli elementi componenti

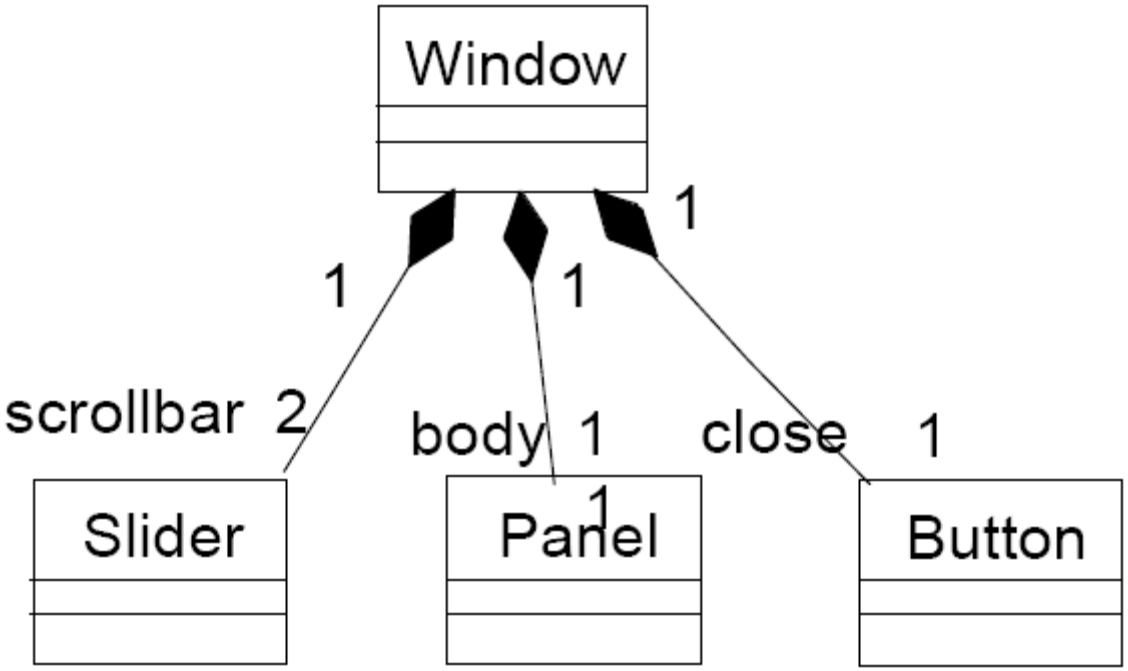
È una relazione più forte dell'aggregazione, *l'oggetto parte* appartiene ad un solo tutto e le parti hanno lo stesso ciclo di vita dell'insieme. All'atto della distruzione dell'oggetto principale si ha la propagazione della distruzione agli oggetti parte.



Si indica con un rombo pieno

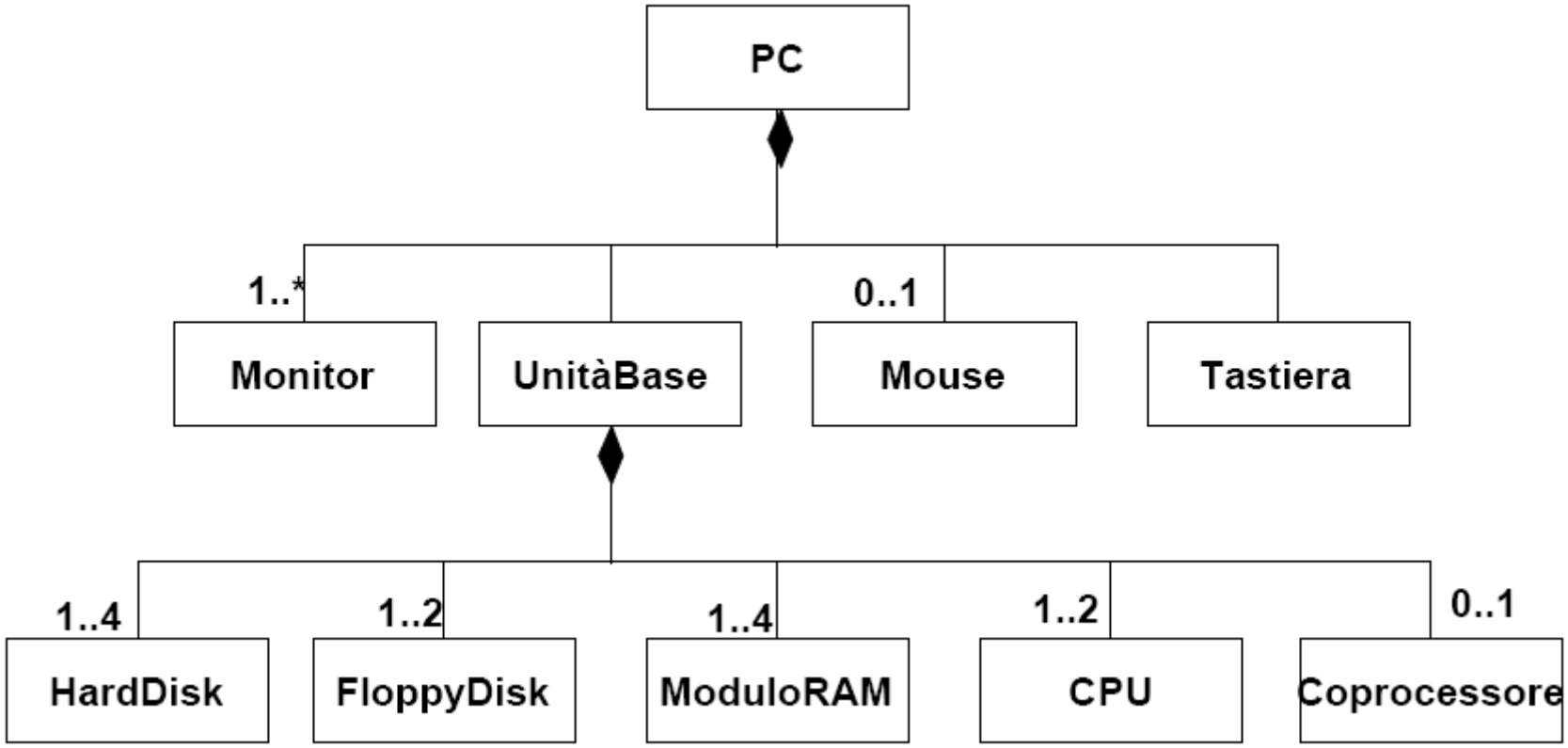
Composizione II

Esempio:



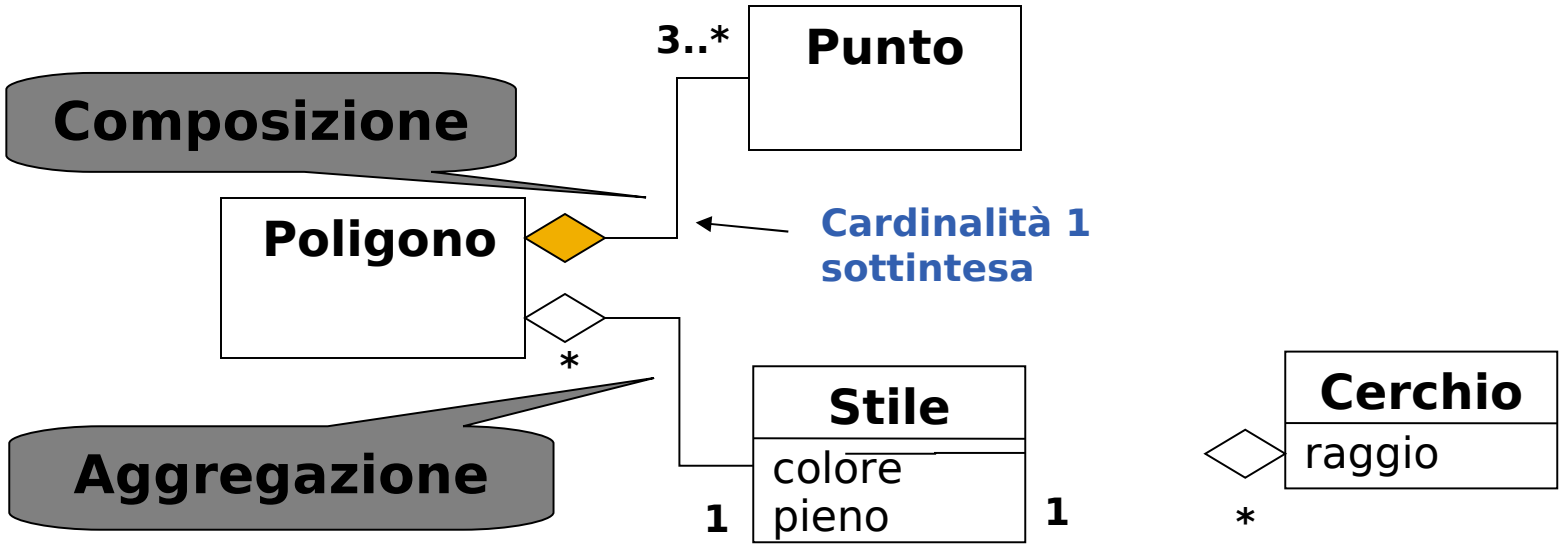
Aggregazione e Composizione

- Per evitare di riempire il diagramma di linee di aggregazione e composizione è possibile usare un "albero”



Aggregazione e Composizione (es.1)

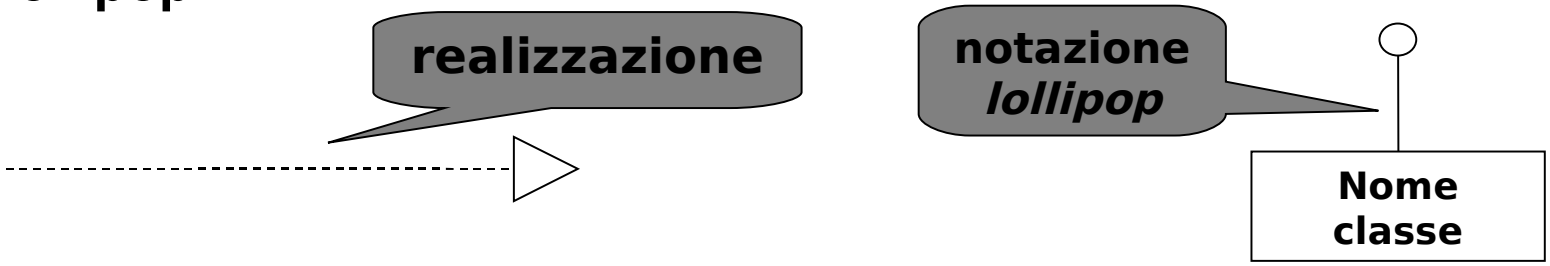
- La cancellazione di una istanza della classe ***Poligono*** viene estesa ad ogni suo ***Punto***, ma non allo ***Stile*** ad esso associato



- Un'istanza della classe ***Stile*** può, invece, essere condivisa tra ***Poligono*** e ***Cerchio***

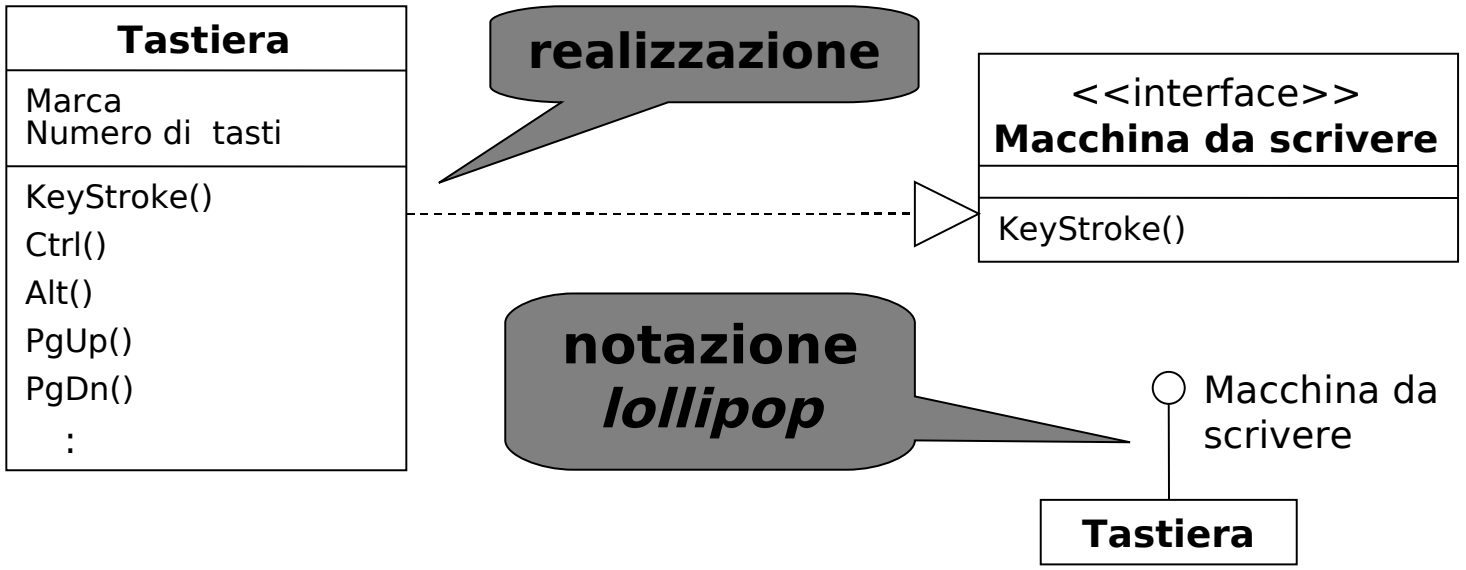
Interfacce e realizzazioni

- **Un'interfaccia viene modellata allo stesso modo in cui viene modellato il comportamento di 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 la dicitura «interface» o un piccolo cerchio (notazione *lollipop,* "a lecca-lecca").
- **La relazione tra una classe ed un'interfaccia viene definita realizzazione. Tale relazione è visualizzata nel modello da una linea tratteggiata con un triangolo largo aperto costruito sul lato dell'interfaccia o con una linea nella notazione compatta lollipop.**



Interfacce e realizzazioni (es.)

La tastiera del computer è un tipico esempio di realizzazione di una interfaccia. La pressione di un tasto (KeyStroke) rappresenta un'operazione che è stata definita dall'interdaccia *Macchina per scrivere*. L'operazione KeyStroke() viene realizzata anche sulla tastiera dei computer. D'altra parte sulla tastiera dei computer si trovano un insieme di operazioni che non appartengono alla macchina per scrivere (Ctrl, Alt, PageUp, PageDown, ecc.)



Visibilità di attributi ed operazioni

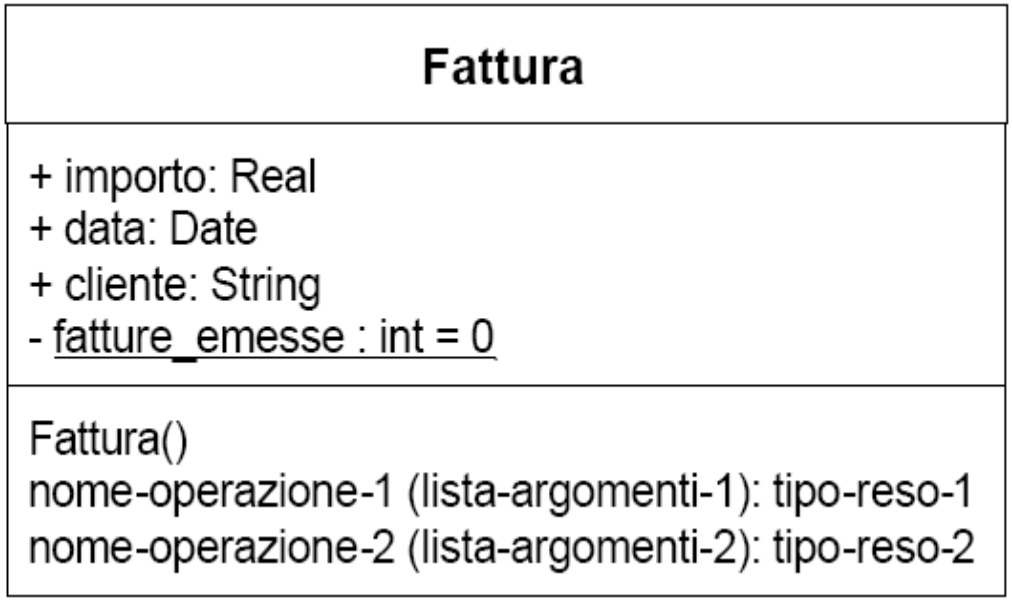
La visibilità specifica le regole secondo le quali il relativo attributo è accessibile da parte di altri oggetti. In particolare, le tipologie di visibilità previste sono:

- **Pubblica (+)**: l'attributo è accessibile da qualsiasi altro oggetto dotato di riferimento all'oggetto che contiene l'attributo in questione;
- **Privata (-)**: l'attributo è accessibile solo all'interno della classe di appartenenza (dichiarante);
- **Protetta (#)**: l'attributo è accessibile da tutte le istanze delle classi che "ereditano" da quella in cui l'attributo è definito;
- **Package (~)**: l'attributo è accessibile da qualsiasi altro oggetto istanza di classi appartenenti allo stesso package o in un altro ad esso annidato a qualsiasi livello.

- **La visibilità può essere indicata opzionalmente davanti al nome di attributi ed operazioni.**
  - + prezzo:Denaro
  - # ImpostaPrezzo(prezzo:Denaro)

Esempio Visibilità

- Esiste una corrispondenza tra la rappresentazione UML di una classe e l'implementazione con un linguaggio O-O (Es. Java)




```
public class Fattura {
public float importo;
public Date data = new Date();
public String cliente;
static private int
    fatture_emesse = 0;
public Fattura() {
fatture_emesse++;
}
// Altri metodi
// ...
```

Visibilità: problemi

- **L'uso del concetto di visibilità in UML è complicato dal fatto che linguaggi OO diversi, pur usando le stesse parole chiave public, private e protected danno ad esse significati sottilmente diversi oppure hanno ulteriori livelli di visibilità**
  - A titolo di esempio si consideri la visibilità package in Java, oppure il concetto di classe *friend* o *implementation* in C++

- **Per questo la visibilità di un diagramma UML deve far riferimento alle regole di uno specifico linguaggio**

Class diagram: riassunto - I

- **Può essere definito a livelli diversi, concettuale, di specifica e di implementazione**
- **Descrive il tipo degli oggetti che compongono il sistema, con tutti gli attributi e le operazioni (ed eventualmente la loro visibilità), nonché le relazioni statiche tra di loro (associazioni e generalizzazioni)**
- **Le associazioni sono dotate di molteplicità ai loro capi e possono avere una navigabilità**

Class diagram: riassunto - II

- **Si possono esprimere vincoli in linguaggio naturale o pseudocodice**
- **Esistono una serie di costrutti avanzati per modellare:**
  - classi associative
  - aggregazioni e composizioni
  - interfacce e realizzazioni

Gli errori più comuni - I

- **Leggendo le specifiche del sistema non sempre è chiaro cosa meriti di essere modellato come una classe e cosa no**
- **Una classe rappresenta un concetto autonomo e importante.**
  - Se il concetto non è autonomo, forse va modellato come un attributo di una classe
    - Es: l'indirizzo di una persona. L'indirizzo da solo vuol dire poco, ha senso perché *legato alla persona.* Quindi **Persona** è la classe, **Indirizzo** è un attributo

Gli errori più comuni - II

- **La stessa cosa vale per gli attributi ed i metodi, occorre mantenere il giusto livello di astrazione**
  - In una classe **Persona** piuttosto che avere i metodi
    - TrovaVia()
    - TrovaNumeroCivico()
    - TrovaCap()

    si può pensare ad un unico metodo
    - TrovaIndirizzoCompleto()