UML: diagramma delle classi e diagramma degli oggetti

Laura Semini

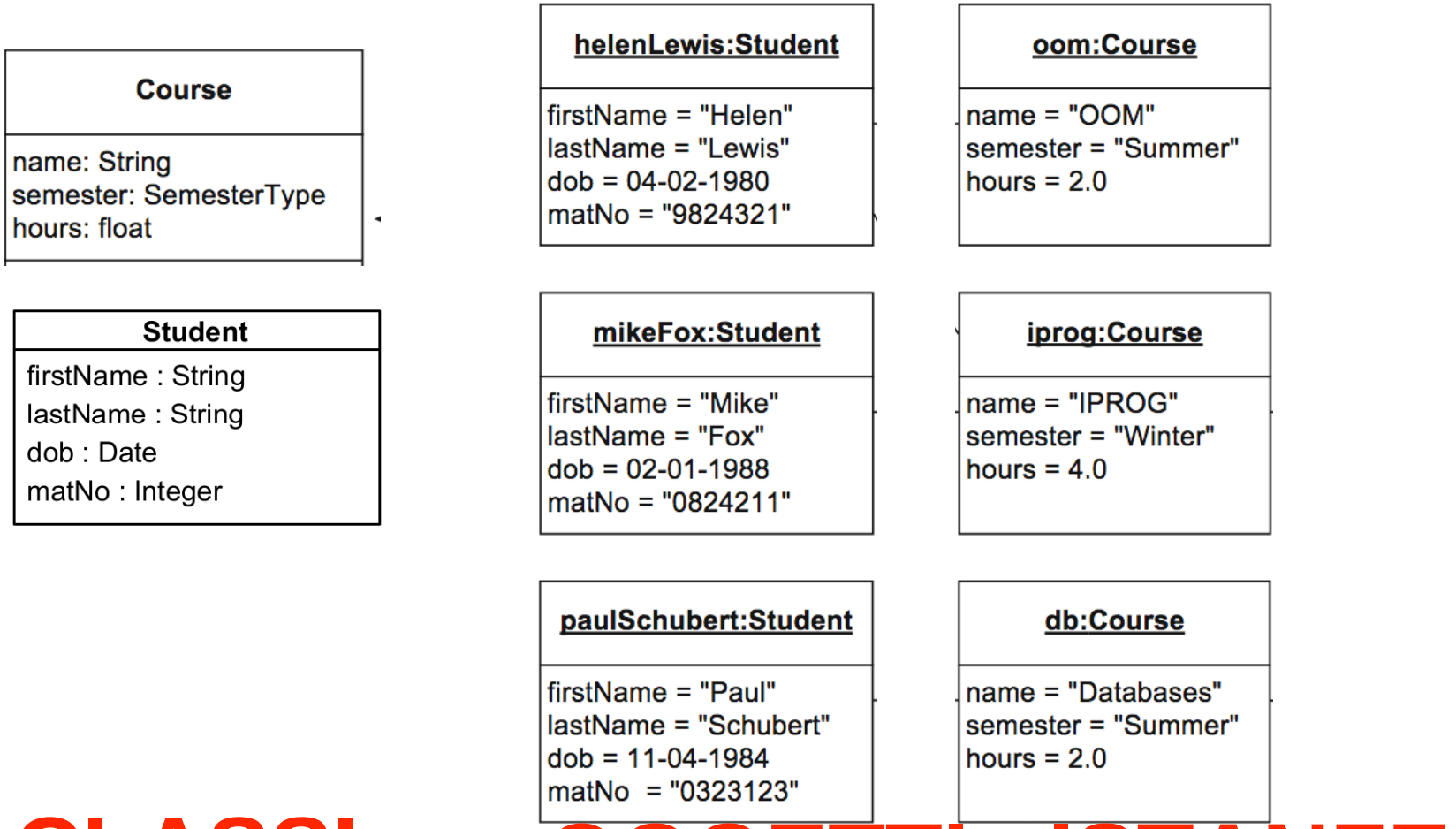
Ingegneria del Software, Dipartimento di Informatica, Univ. di Pisa

Classi e oggetti, reminder

- Un oggetto è un'entità caratterizzata da
  - Un'identità
  - Uno stato
  - Un comportamento

- Una classe descrive
  - un insieme di oggetti con caratteristiche simili
  - cioè oggetti che hanno lo stesso tipo

Classi e oggetti



UML: classificatori

- Le classi sono classificatori
- Gli oggetti sono istanze
- Modellare a livello dei classificatori significa vincolare i modelli a livello di istanza

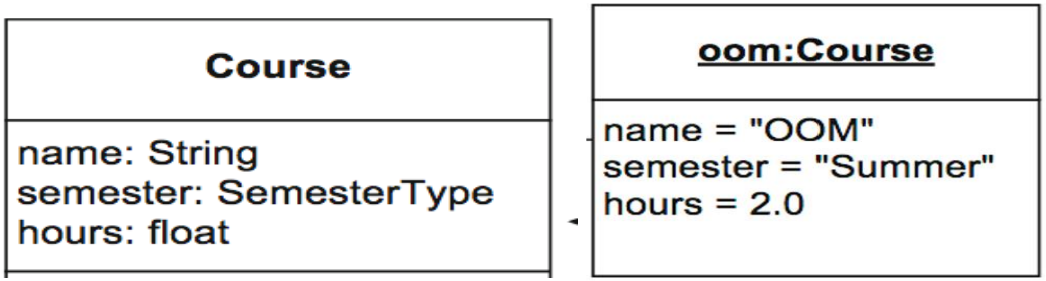


Diagramma delle classi

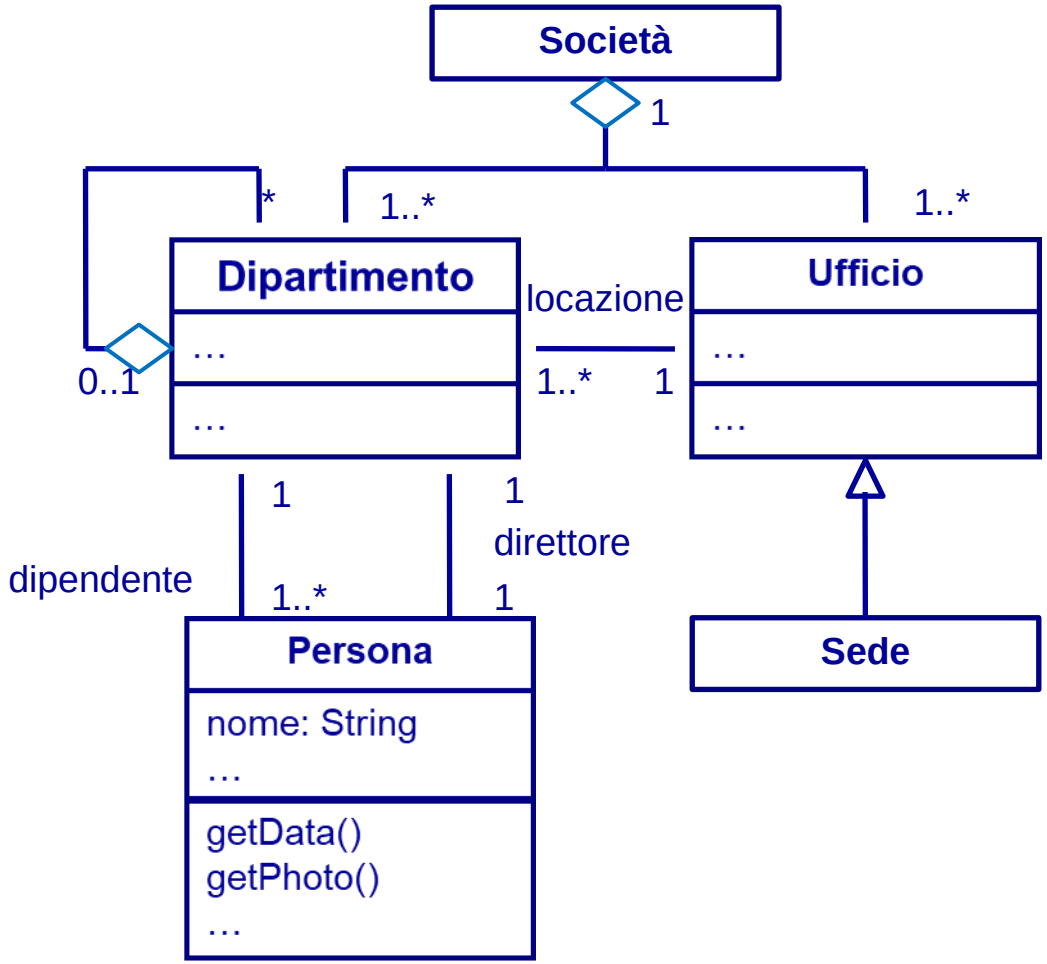
- Una classe cattura un concetto nel dominio del problema o della realizzazione

- Il diagramma delle classi descrive:
  - Il tipo degli oggetti che fanno parte di un sistema sw o del suo dominio
  - Le relazioni statiche tra essi: gli elementi e le relazioni tra essi non cambiano nel tempo

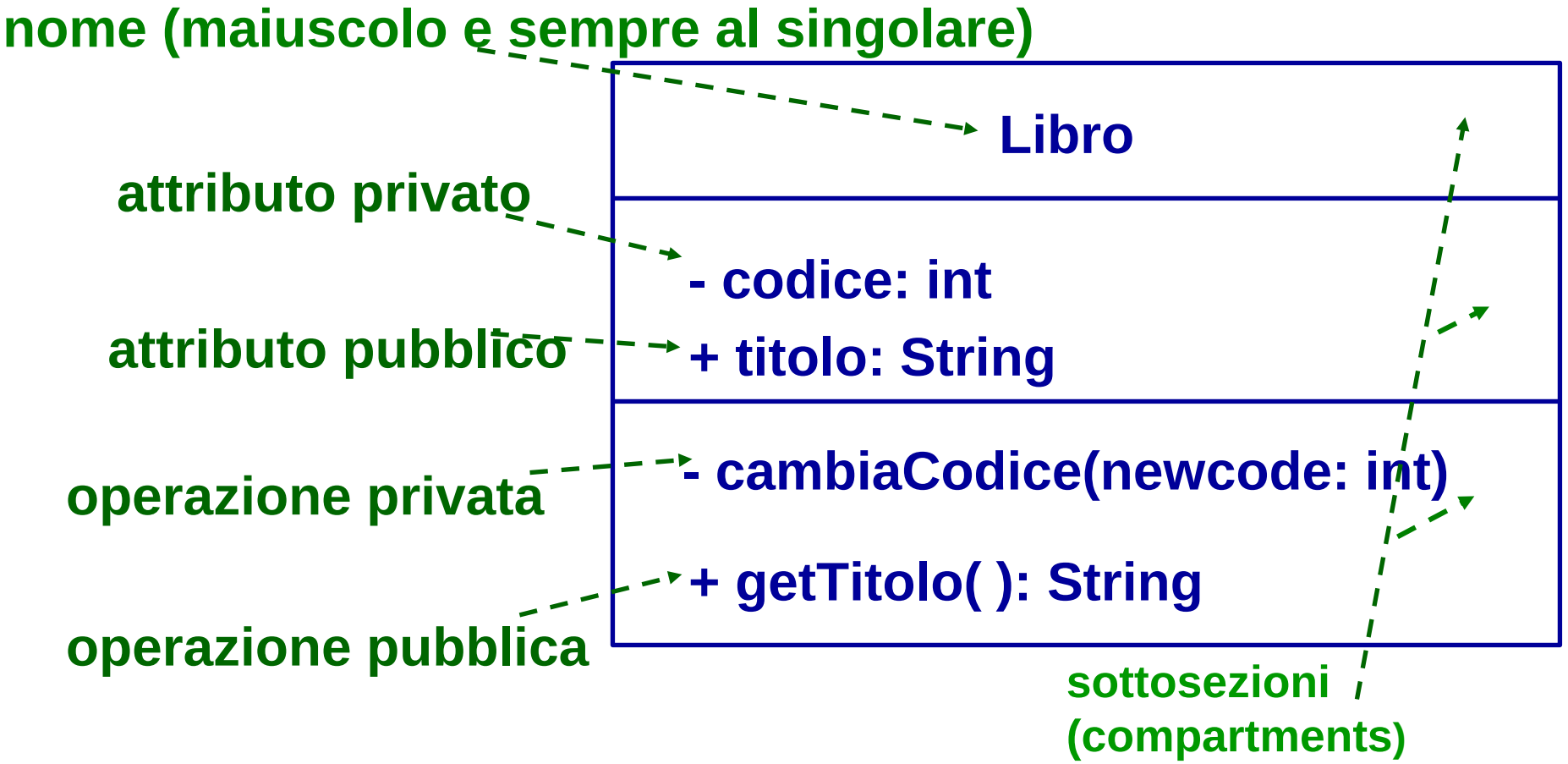
- I diagrammi delle classi mostrano anche le proprietà e le operazioni di una classe

Esempio

- Una società è formata da dipartimenti e uffici
- Un dipartimento ha un direttore e più dipendenti
- Un dipartimento è situato in un ufficio
- Esiste una struttura gerarchica dei dipartimenti
- Le sedi sono uffici



Sintassi


nome (maiuscolo e sempre al singolare)
Libro
attributo privato
- codice: int
attributo pubblico
+ titolo: String
operazione privata
- cambiaCodice(newcode: int)
operazione pubblica
+ getTitolo( ): String
sottosezioni
(compartments)

Usi del diagramma delle classi

Il diagramma delle classi può essere usato

- a diversi livelli di dettaglio
- in diverse fasi del progetto
- fino alla generazione del codice in linguaggi OO

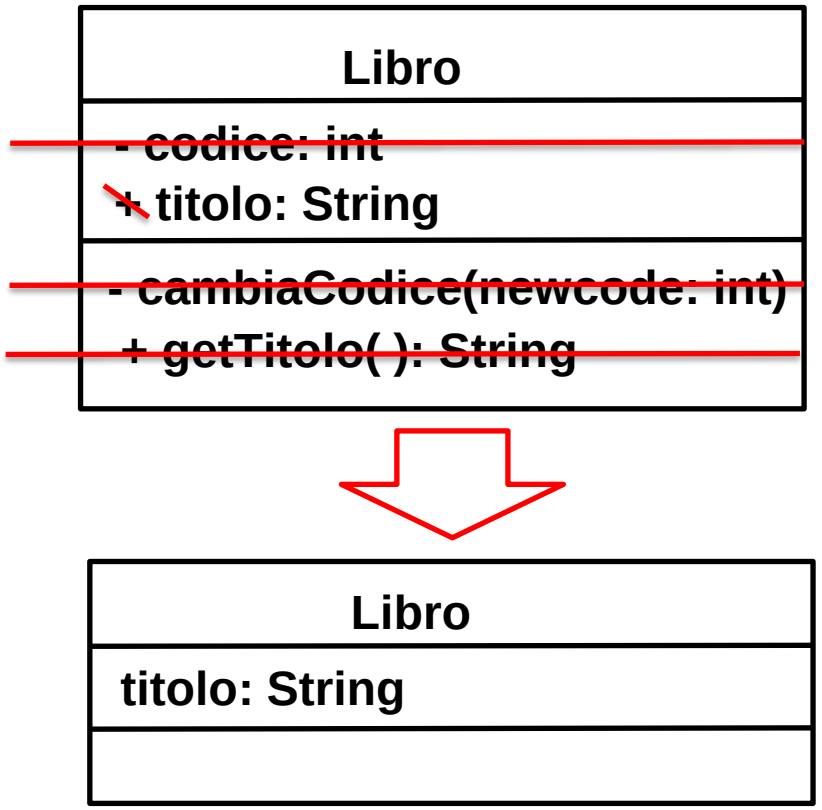
Semantica

- Un oggetto è un’entità caratterizzata da
  - Un’identità, uno stato, un comportamento
- (I valori de)gli attributi definiscono lo stato dell’oggetto
- Le operazioni definiscono il suo comportamento

Livello di astrazione

Quando si usa il diagramma delle classi per descrivere il dominio:

- le operazioni (ritenute a un livello di dettaglio eccessivo) normalmente si omettono
  - In particolare MAI setters e getters
- gli attributi utili per caratterizzare l'elemento del dominio si specificano, dettagli implementativi no
- Le visibilità si omettono



Attributi: Sintassi

visibilità nome: tipo [molteplicità] = valoreIniziale {proprietà}

potrebbe esserci solo il nome

chiamato di default nel libro, ma si intende iniziale

molteplicità:

per indicare

array di valori

colore: Integer [3] {ordered} ***modello RGB***

secondoNome: String [0..1] ***zero per permettere valore null***

nome: String ***la molteplicità [1] può essere omessa***

proprietà

{>0, <10} ***vincoli sui valori che l'attributo può avere***

{ordered}, {unique} ***hanno senso quando un attributo ha una molteplicità di valori:***
ordered → liste ordinate invece che insiemi o multiinsiemi
unique → senza ripetizioni, come negli insiemi

Esempi

n: char carattere, tipo predefinito

n: String stringa, tipo predefinito

g: Gra con tipo Gra definito nel modello

n: Integer =1 {>= 0} numero intero non negativo, inizialmente = 1

p : Integer [2] {>0, ordered} punto del quadrante positivo

s: Integer[10] {>3, <33, unique} insieme (unique→senza ripetizioni) di 10 numeri compresi tra 3 e 33

col: Integer [3] {>=0, <=255, ordered} lista (ordered →la posizione è significativa) di 3 numeri, compresi tra 0 e 255

nome: String [1..2] {ordered, unique} si deve avere almeno un nome, opzionalmente un secondo nome, ma diverso dal primo

Visibilità

Un elemento è visibile all'esterno dello spazio di nomi che lo contiene, in accordo con il suo tipo di visibilità

+ public: accessibile ad ogni elemento che può vedere e usare la classe

protected: accessibile ad ogni elemento discendente

- private: solo le operazioni della classe possono vedere e usare l'elemento in questione

~ package: accessibile solo agli elementi dichiarati nello stesso package

Sintassi delle operazioni

visibilità nome (listaParametri) : tipoRitorno

anche solo: nome()

listaParametri::= Ø | dichiarazione di paramentro, listaParametri

dichiarazione di paramentro ::= direzione nome: tipo = default

obbligatorio solo nome (fermo restando che non è obbligatorio dichiarare i paramentri)

in, out, inout

valore assegnato al parametro in assenza di argomento

Esempi

+ sum (a: Integer, b: Integer) : Integer

metodo pubblico che, dati due interi restituisce un intero

+ sum (a: Integer, b: Integer =10) : Integer

come sopra, con 10 valore di default del secondo parametro

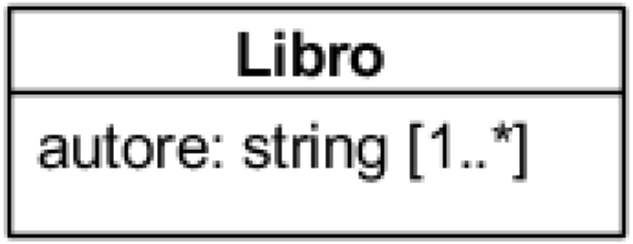
- gra () : Gra

metodo privato che restituisce un oggetto di tipo Gra

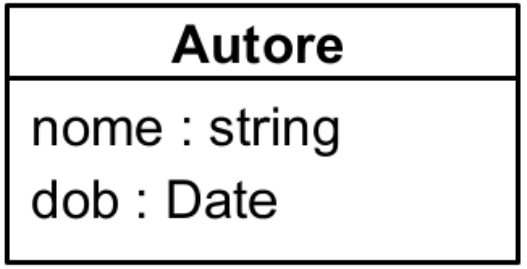
Quando un concetto va modellato con una classe e quando con un attributo

Un libro ha uno o più autori

- autore modellato solo come attributo di Libro: specifichiamo solo il nome

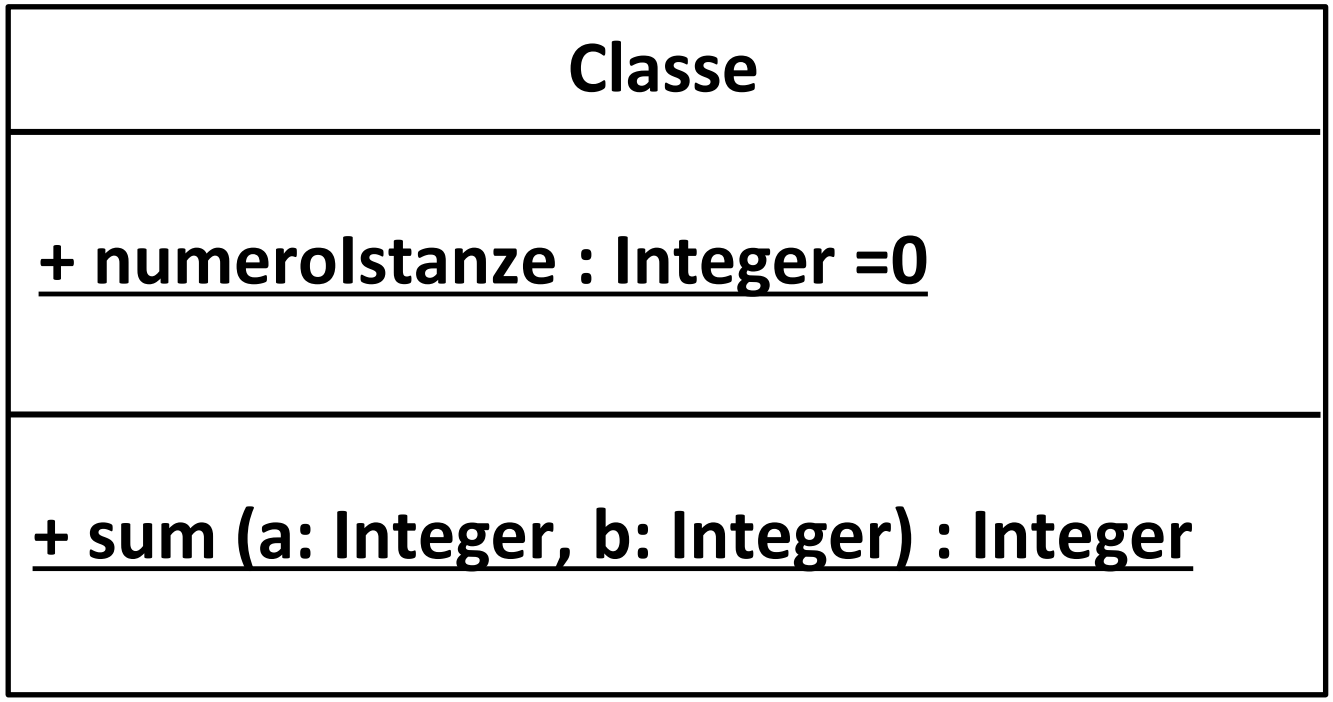


- autore modellato come classe: può avere attributi propri e eventualmente operazioni

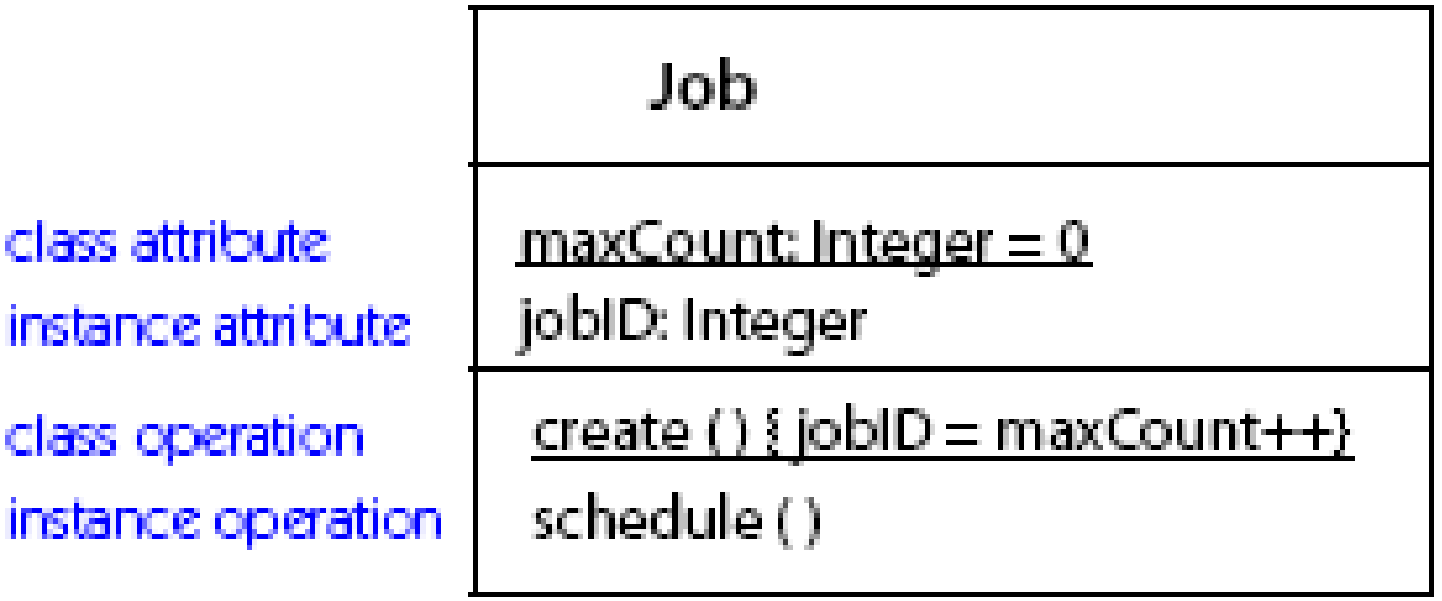


Attributi e operazioni con ambito di classe (statici)

Sono sottolineati

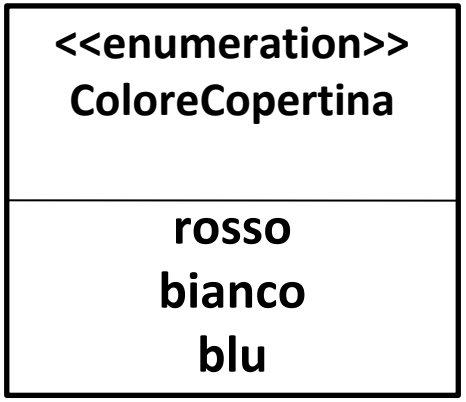


Esempio


Job
maxCount: Integer = 0
jobID: Integer
create ( ) { jobID = maxCount++}
schedule ( )
class attribute
instance attribute
class operation
instance operation

Enumerazioni

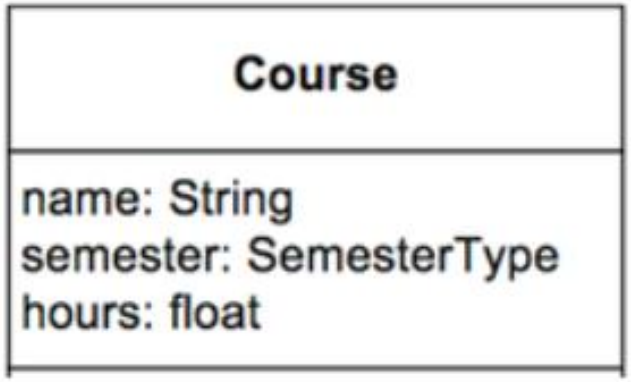
- Le enumerazioni sono usate per specificare un insieme di valori prefissati
  - Un'enumerazione è la lista completa di tutti i valori che gli attributi di un determinato tipo possono assumere
- In UML sono rappresentate da classi
  - etichettate dallo stereotipo <<enumeration>>
  - con un nome (il tipo) e l'insieme di valori che gli attributi di quel tipo possono assumere

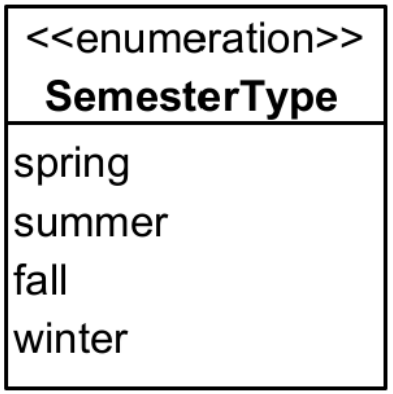


La copertina dei libri può essere solo o tutta rossa, o tutta bianca, o tutta blu

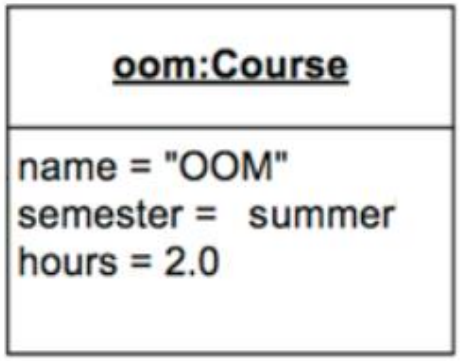
Enumerazioni: esempio

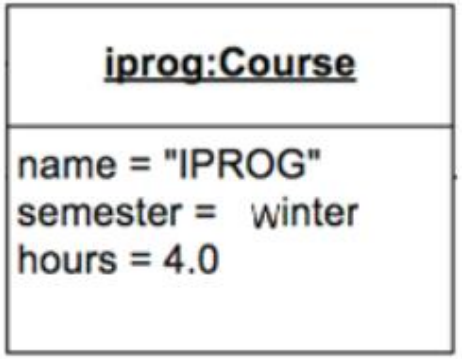
Dato:





Ci possiamo aspettare





Relazioni

- Una relazione rappresenta un legame
  - tra due o più oggetti
  - normalmente istanze di classi diverse

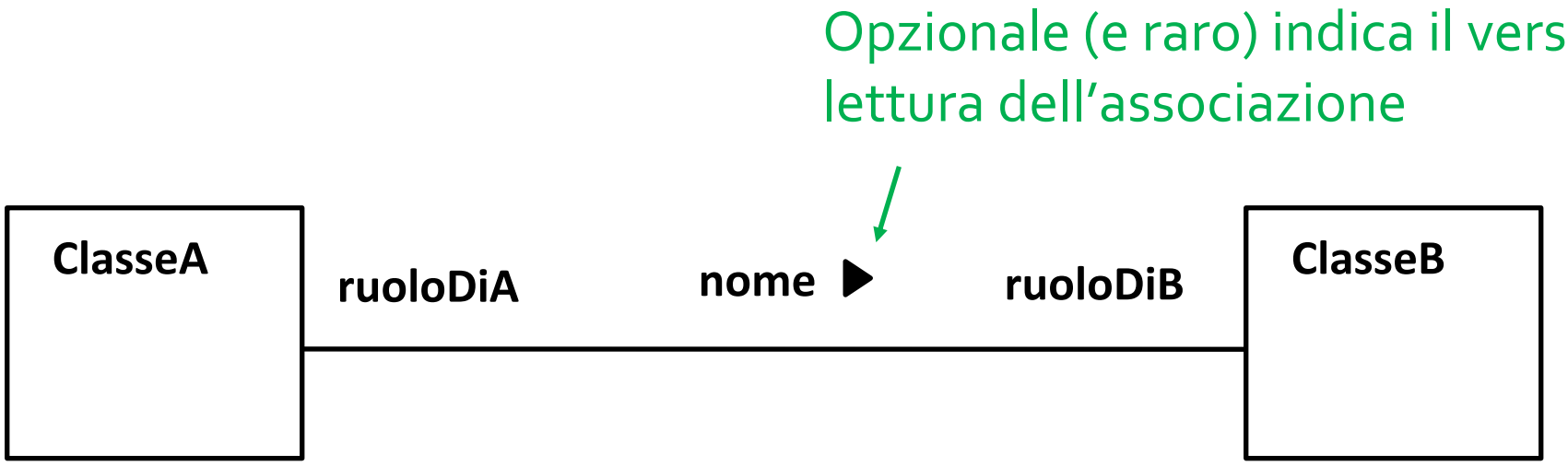
Relazioni

- Vedremo le seguenti relazioni:

Tra Classificatori	Tra oggetti
Associazione Aggregazione Composizione	Collegamento Aggergazione Composizione
Generalizzazione	(non definita)
Realizzazione	(non definita)
Dipendenza (d'uso, di istanza…)	

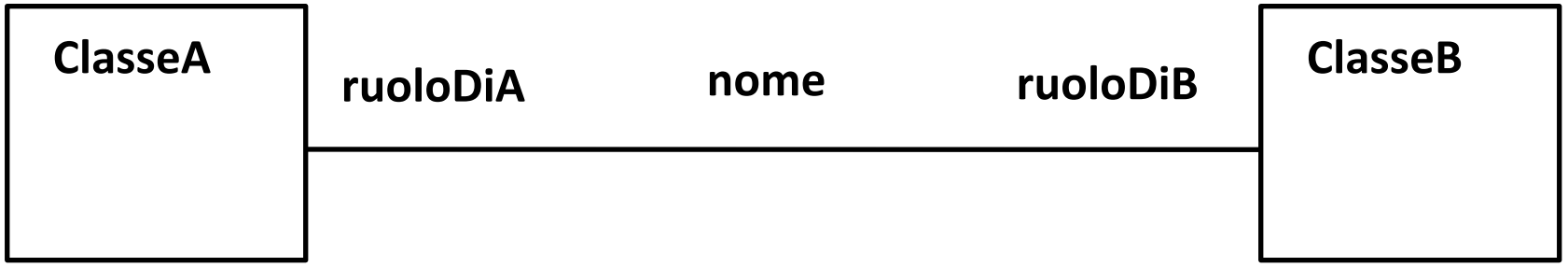
Associazione: esempio

- Una linea retta tra le classi
  - A volte, e solo documentando il codice (non il dominio), una freccia (eventualmente doppia) per specificare la navigabilità
- Almeno uno tra nome o ruoli, raramente entrambi.
  - Servono a caratterizzare la relazione



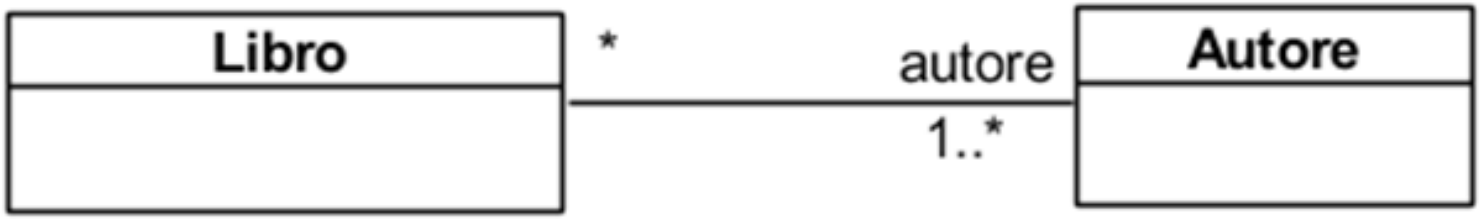
Associazione: nome e ruoli

- nome e ruoli: minuscolo
  - nome : normalmente un verbo
  - ruolo: normalmente un sostantivo
- Formalmente opzionali,
  - è di fatto quasi sempre necessario che ci sia o il nome dell'associazione o l'indicazione dei ruoli (inutile entrambi)



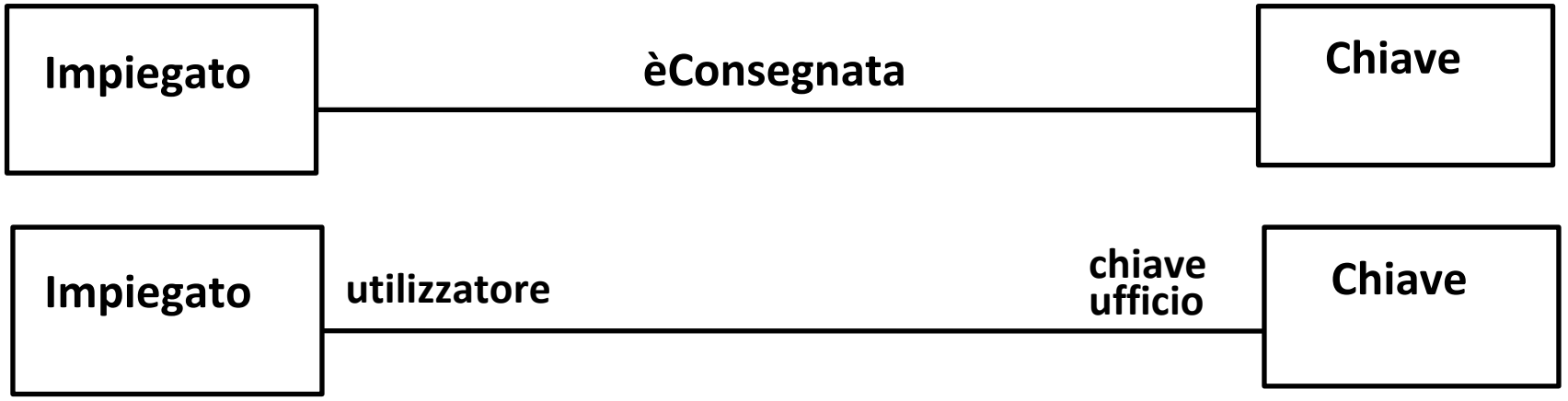
Associazioni o attributi ?

La descrizione usa associazioni

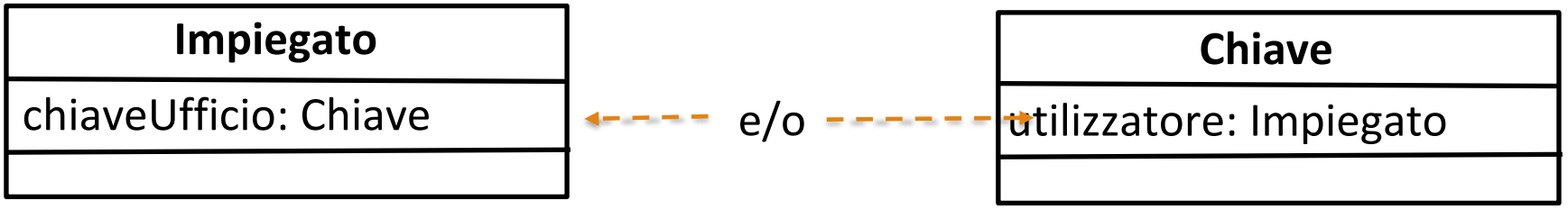


Che nel codice diventano attributi

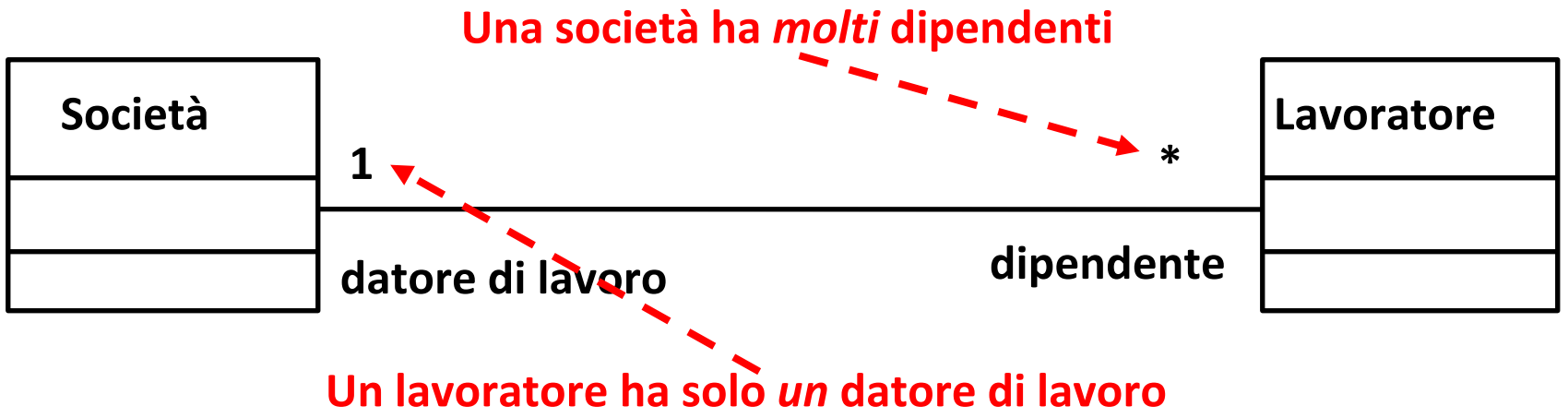
Associazione: esempio



- Si esplicitano i ruoli degli oggetti nella relazione
  - c è la chiave dell'ufficio di i
  - i ne è l'utilizzatore
- Quando si trasforma il modello in codice:



Associazione: vincoli di molteplicità

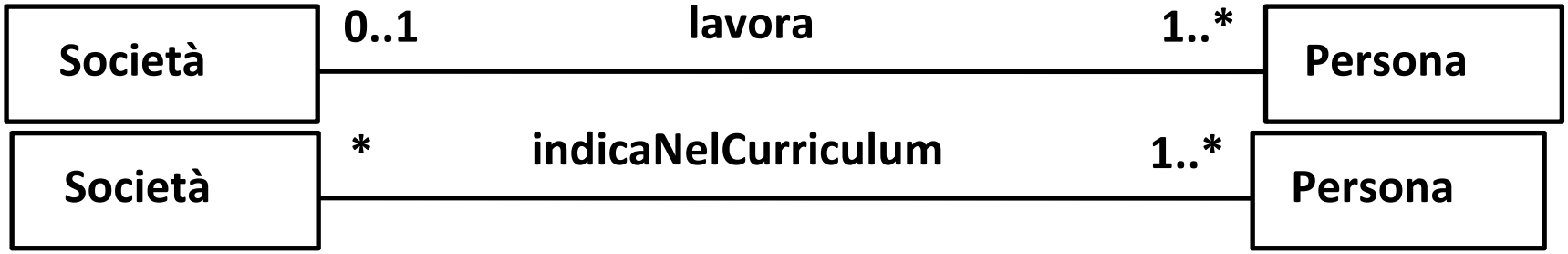


- La molteplicità indica il numero di oggetti coinvolti nell'associazione **in un dato istante**
  - Un oggetto Società può essere in relazione con molti oggetti Lavoratore
  - Un oggetto Lavoratore può essere in relazione con un solo oggetto Società (Nota bene: in un dato istante di tempo)

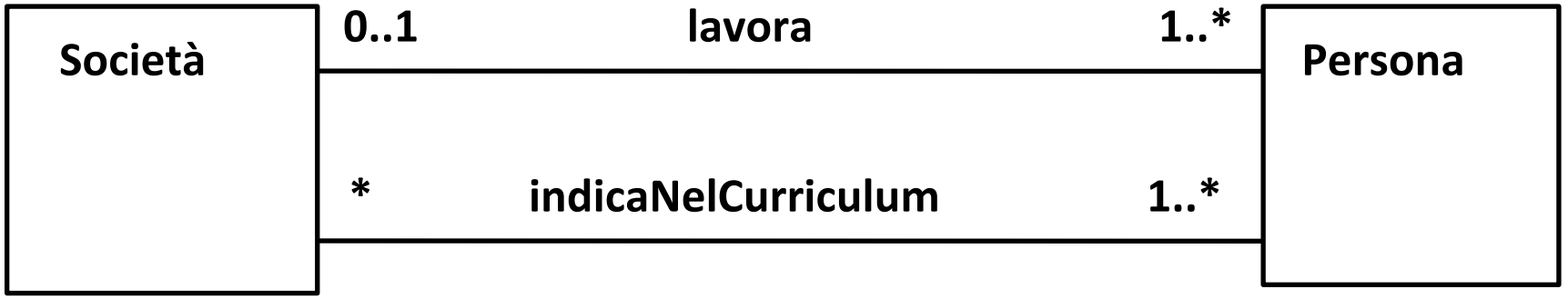
Molteplicità delle relazioni

- Le molteplicità si possono definire:
  - Con un numero positivo
    - 1 è il default e si può omettere
  - Con * (indefinito)
  - Indicando gli estremi inferiore e superiore di un intervallo
    - Ex: 0..1 , 2..4 , 1..5
    - 0..* ≡ *
  - l'estremo inferiore può essere zero 0 o un numero positivo
  - l'estremo superiore un numero positivo o *

La molteplicità è legata al nome dell'associazione

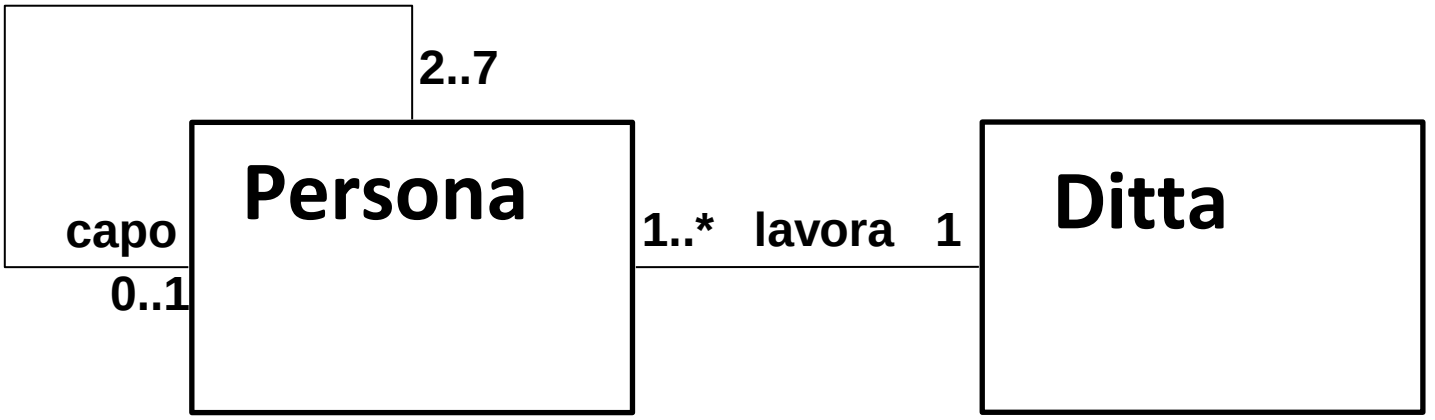


Sintatticamente, ci possono essere più associazioni tra due classi



Assiciazioni riflessive

In questo caso è fondamentale indicare il ruolo



Aggregazione e Composizione

Aggregazione e Composizione sono tipi particolari di associazione

- entrambe specificano che un oggetto di una classe è **una parte** di un oggetto di un'altra classe

- Prenderle in considerazione quando il nome dell'associazione sarebbe del tipo:
  - fa parte di, appartiene….

    o, dualmente:
  - è composto da, possiede, ha, …

Aggregazione vs Composizione

Aggregazione → relazione tra oggetti poco forte

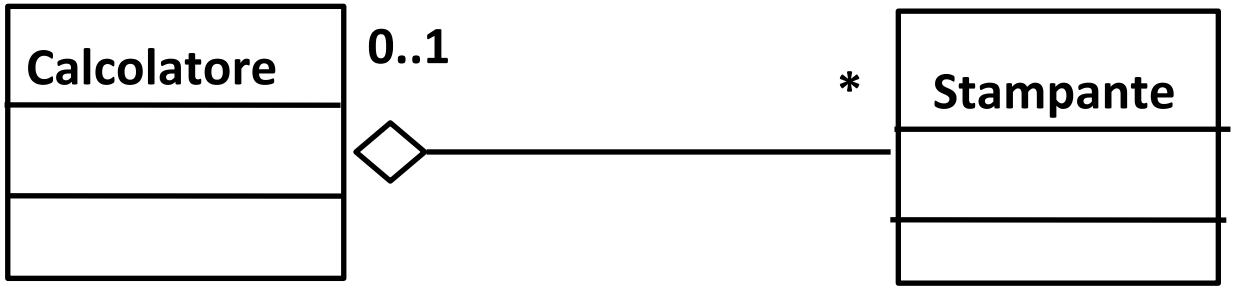
ovvero una relazione nella quale le classi parte hanno un significato anche senza che sia presente la classe tutto

Composizione → relazione tra oggetti forte

le classi parte hanno un reale significato solo se sono legate alla classe tutto

Sintassi e semantica con un esempio

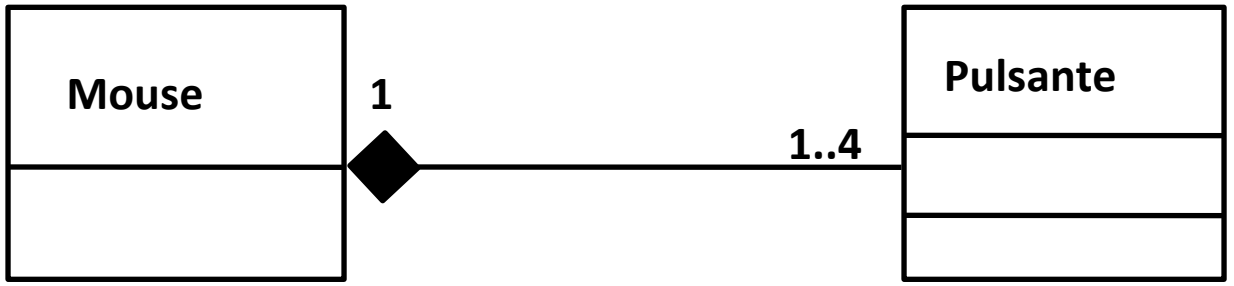
Aggregazione



- La stampante nel tempo può essere collegata a calcolatori diversi
- La stampante esiste anche senza calcolatore
- Se il calcolatore viene distrutto la stampante esiste comunque
- L’aggregazione non ha un nome

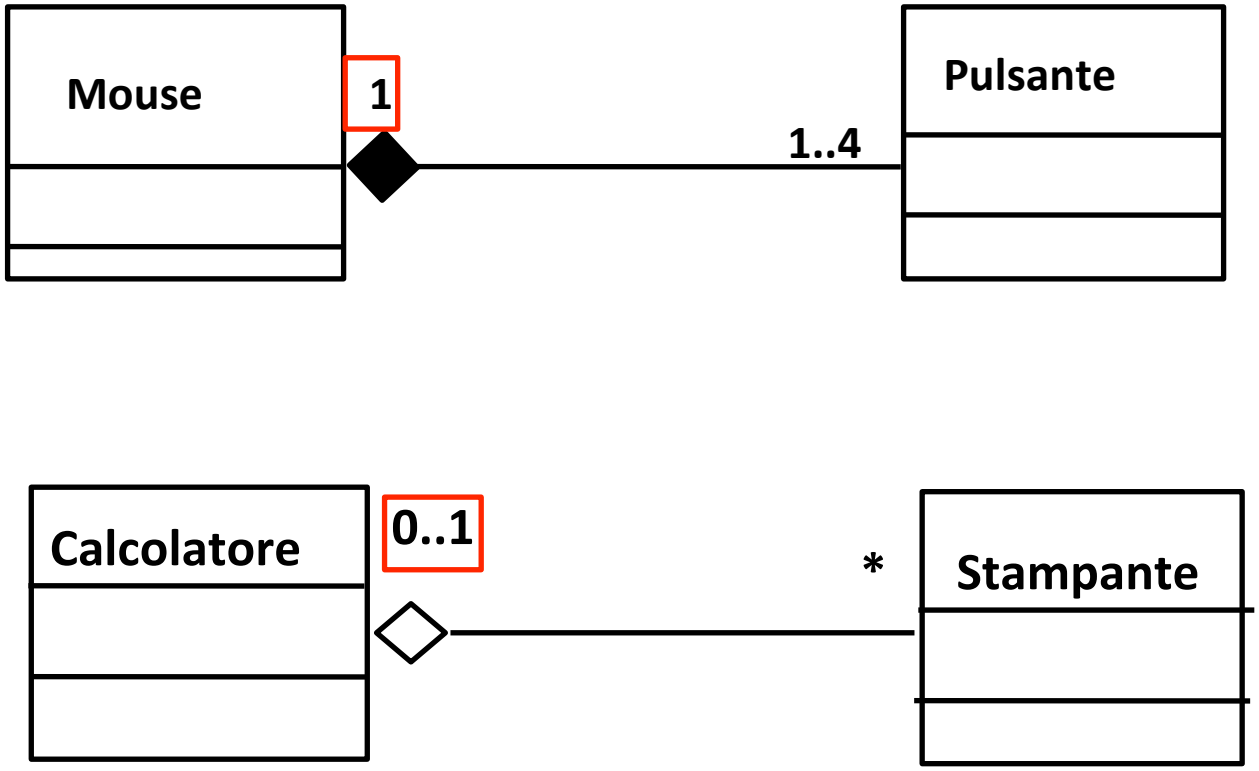
Sintassi e semantica con un esempio

Composizione



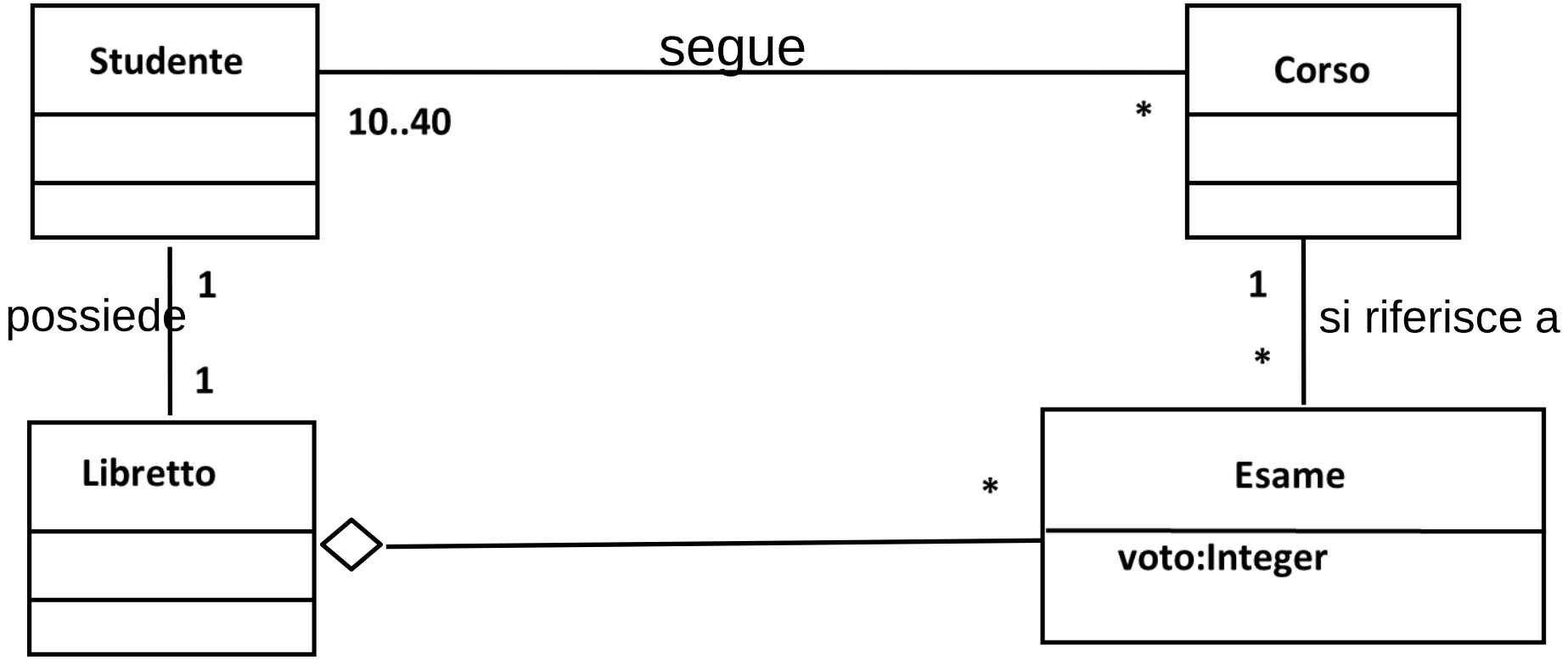
- Un pulsante appartiene a un solo mouse
- Non esiste senza il suo mouse
- Se il mouse viene distrutto vengono distrutti anche i pulsanti
- La composizione non ha un nome

Uno sguardo alle molteplicità



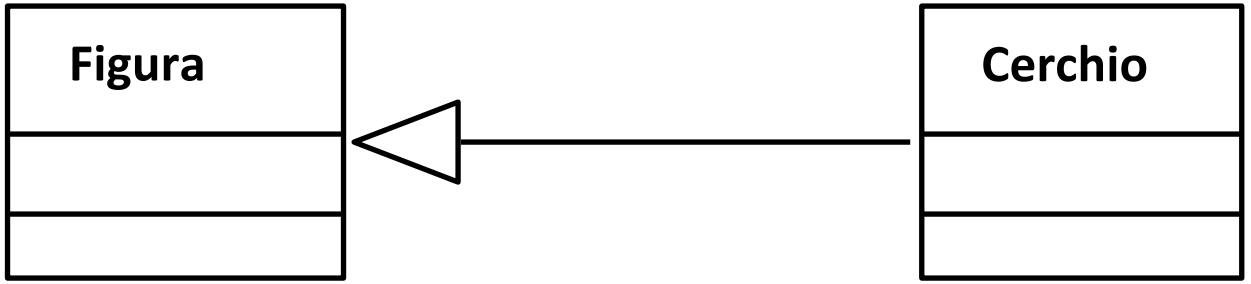
Esempio

- Studente, Libretto, Esame, Corso



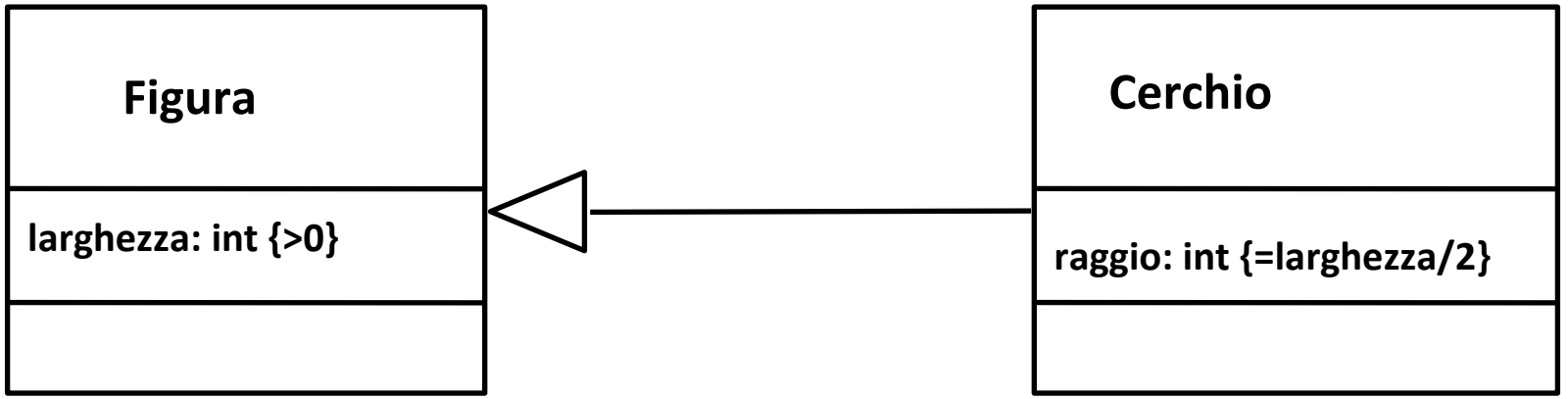
Generalizzazione

- Relazione tra un elemento generico e uno più specializzato
- L'elemento più specializzato è consistente con quello più generico ma contiene più informazione
- Vale il principio di sostituzione della Liskov: l'elemento specializzato può essere usato al posto dell'elemento generico
- "è un tipo di"

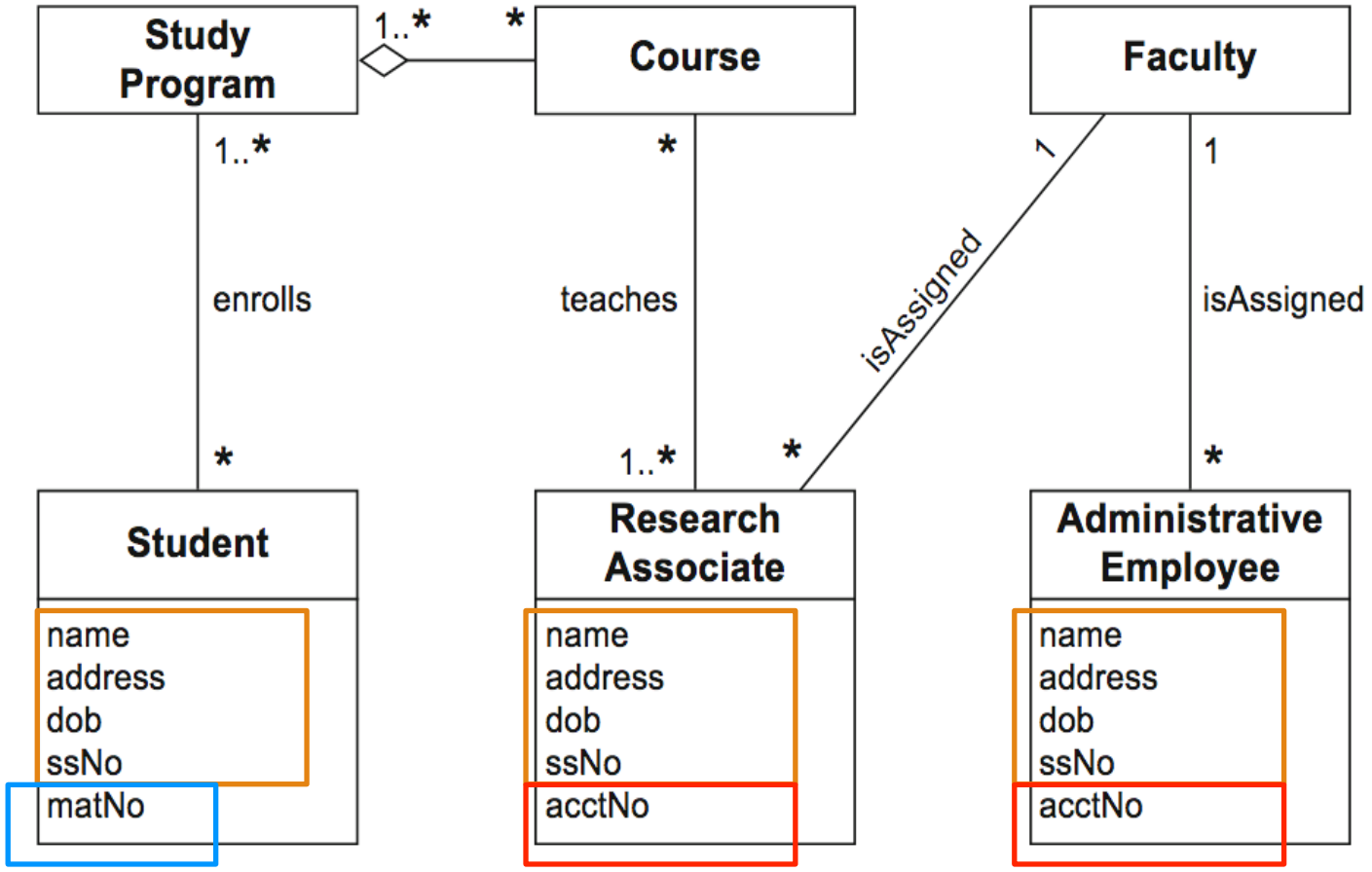


Ereditarieta’ di classe

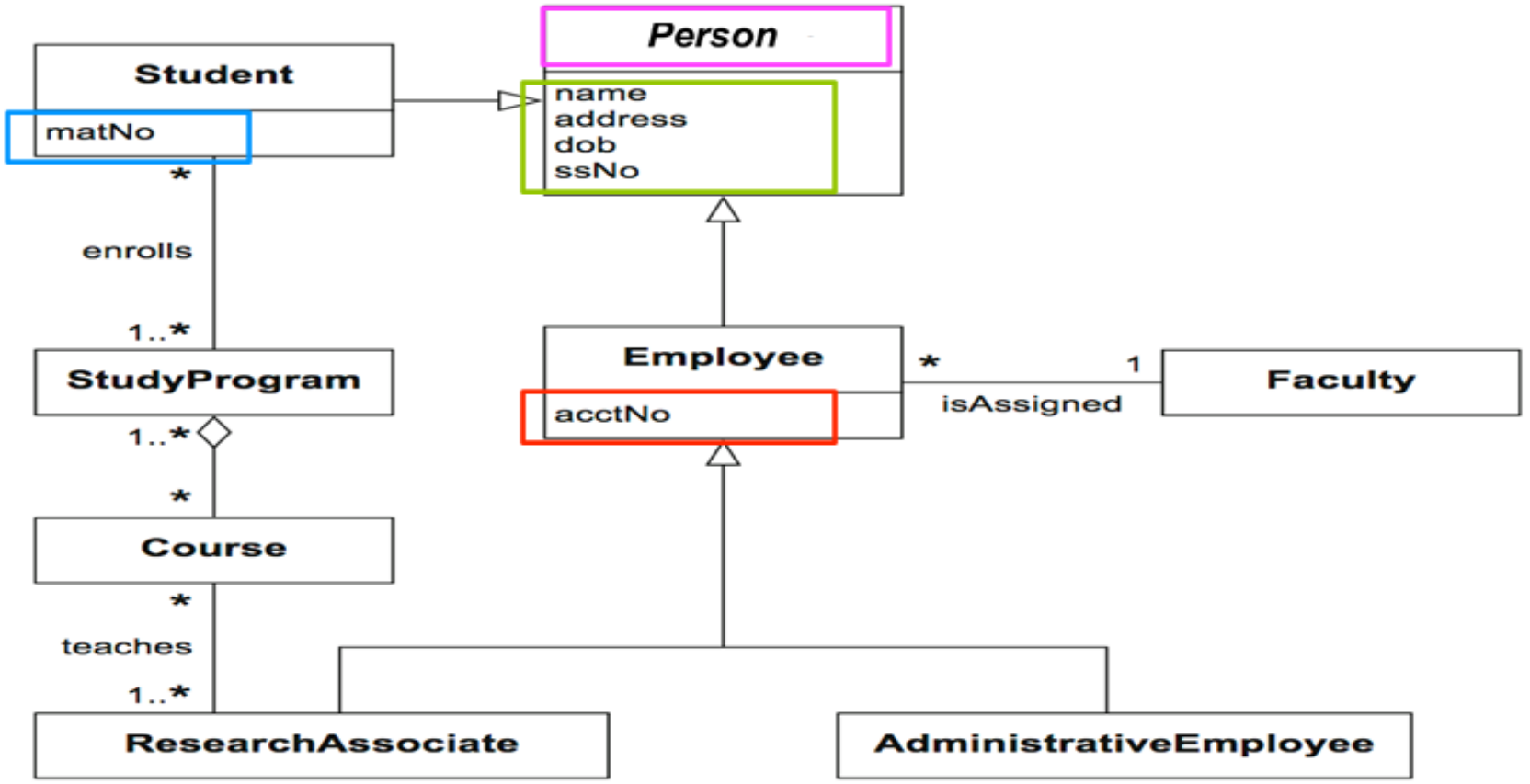
- Le sottoclassi ereditano tutte le caratteristiche della superasse:
  - attributi, operazioni , relazioni e vincoli
- Le sottoclassi possono aggiungere caratteristiche e ridefinire le operazioni



Esempio: verso la generalizzazione

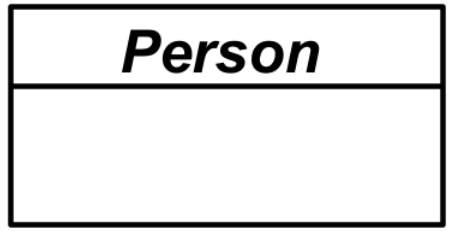


Esempio generalizzazione

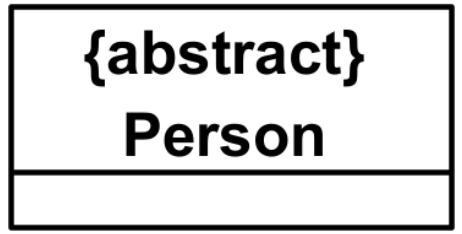


Classi astratte

Notazione compatta (usando il corsivo)

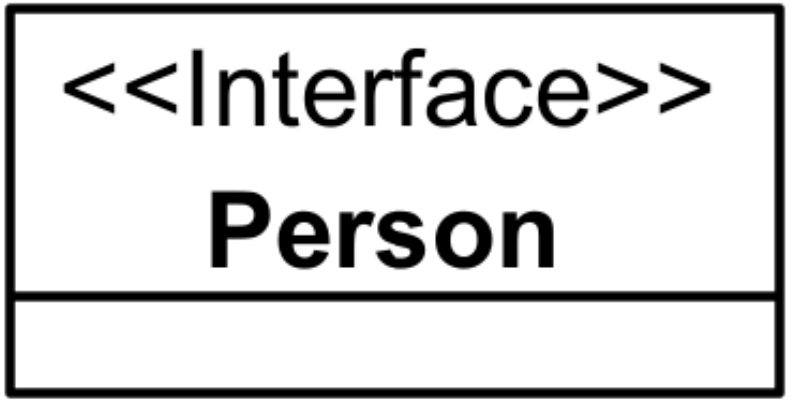


Notazione usata dal vostro libro



(Classi) interfaccia

Si usano in fase di progettazione, per classi con solo comportamento e senza stato

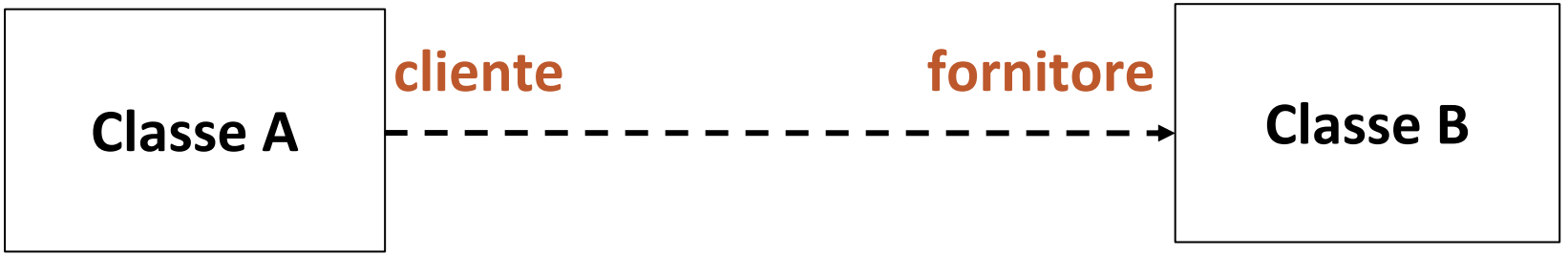


Non solo l’interfaccia utente!!!

Dipendenze

Una relazione in cui le classi hanno ruolo di cliente e fornitore

- Il cliente dipende dal fornitore



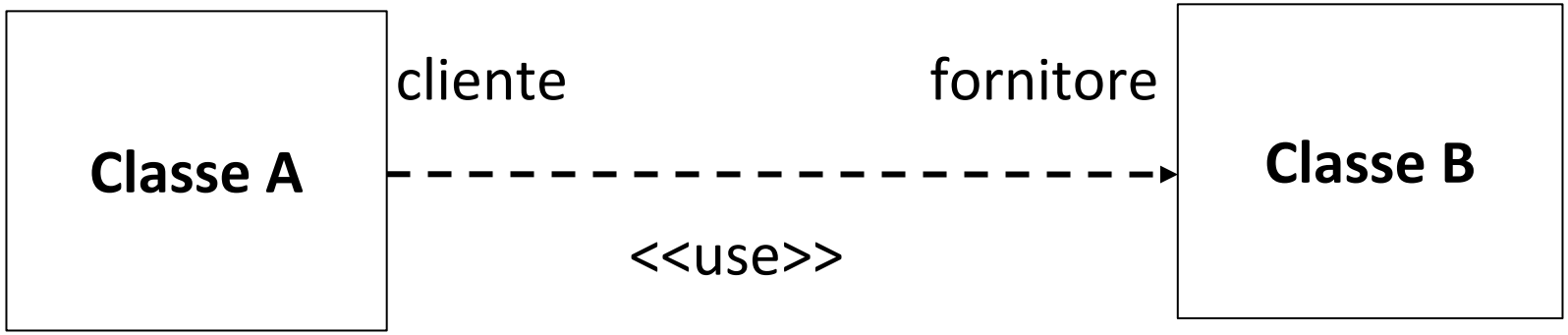
- una modifica nel fornitore può influenzare il cliente

- Esempi:
  - Un parametro di un'operazione di A è di tipo B
  - Un'operazione di A restituisce un oggetto di tipo B

Dipendenza d'uso

La dipendenza più comune

- Un'operazione di A invoca un'operazione di un oggetto di tipo B



- Un altro tipo di dipendenza tra classi è <<create>> , quando un'operazione di A crea dinamicamente un oggetto di tipo B

Individuare le classi di analisi (del dominio)

- Cosa sono le classi di analisi
  - Corrispondono a concetti concreti del dominio:
    - Per esempio i concetti descritti nel glossario
    - Normalmente, ciascuna classe di analisi sarà raffinata in una o più classi di progettazione.
  - Evitare di introdurre delle classi di progettazione

- Tecniche per individuarle
  - Non esiste un metodo automatico che le estrae da un documento in linguaggio naturale
  - Ma possiamo usare alcune tecniche note

Classi di analisi: caratteristiche

- Astrazione di uno specifico elemento del dominio
- Hanno un numero ridotto di responsabilità (funzionalità)
- Evitare di definire classi “onnipotenti”
  - Attenzione quando si chiamano “sistema”, “controllore”, ...
- Evitare funzioni travestite da classi
- Evitare gerarchie di ereditarietà profonde (>=3)

Classi di analisi: livello di astrazione/dettaglio

Operazioni e attributi solo quando veramente utili

- Le classi di analisi dovrebbero contenere attributi e operazioni ad “alto livello”
- Limitare la specifica di tipi, valori, etc.
- Non inventare mai niente rispetto a quanto scritto nel documento! O almeno non prima di esservi confrontati con clienti o utenti

Classi di analisi: tecniche di identificazione

- Principali tecniche
  - Approccio data driven: tipico della fase di analisi
    - Si identificano tutti i dati del sistema e si dividono in classi
      - ad esempio mediante identificazione dei sostantivi
  - Approccio responsibility driven: soprattutto durante la progettazione
    - Si identificano le responsabilità e si dividono in classi

Classi UML vs Entità (Basi di Dati)

- DB: le classi (entità) sono intese come collezioni.
- Sottointeso che:
  - ci sono più istanze.
  - ci sono operazioni per visitare tutte le istanze.
- Nelle classi UML no
- Da cui, per esempio, nome singolare vs nome plurale.
- La differenza è significativa più in prospettiva di progettazione che di descrizione del dominio.
  - In progettazione OO e quindi in UML uso “ListaDiQcosa” come aggregato di “Qcosa”

Classi UML

Entità

Diagramma degli oggetti

- viene anche chiamato diagramma delle istanze
- può essere utile quando le connessioni tra gli oggetti sono complicate.

Diagramma degli oggetti

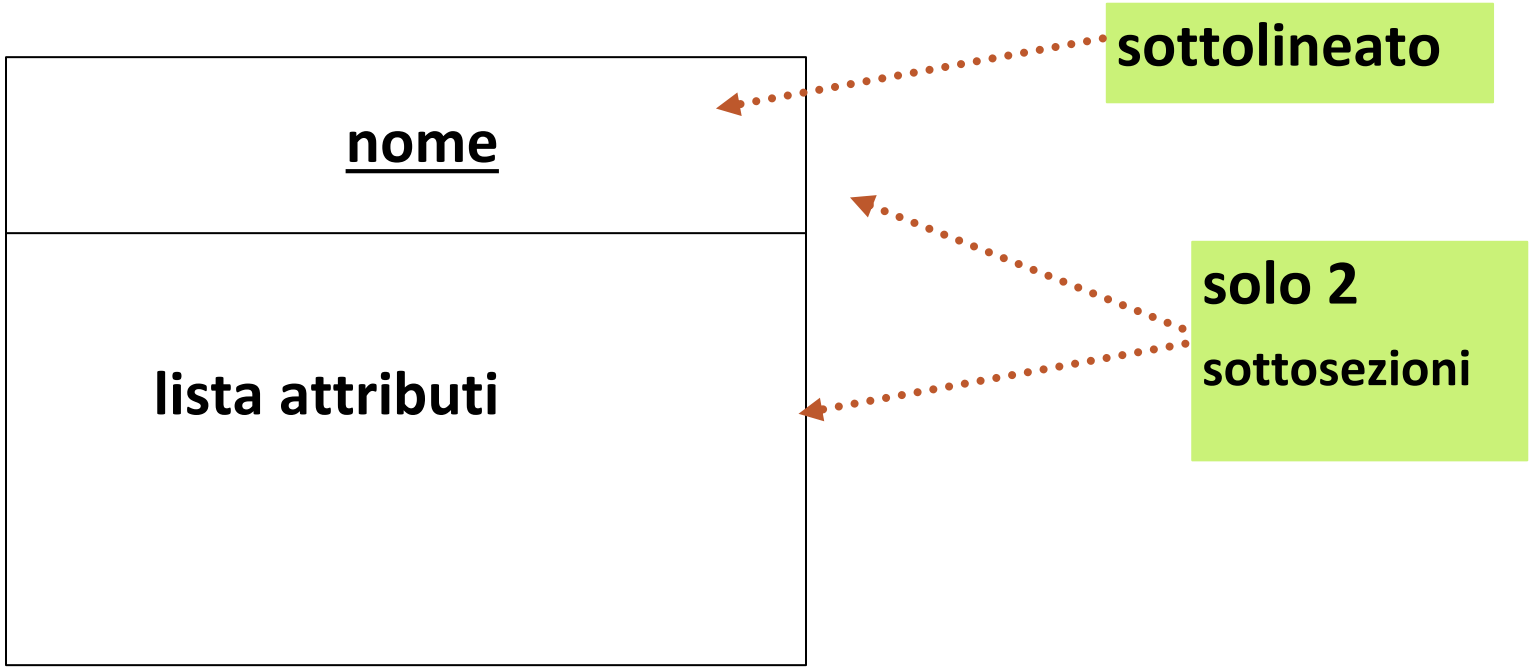
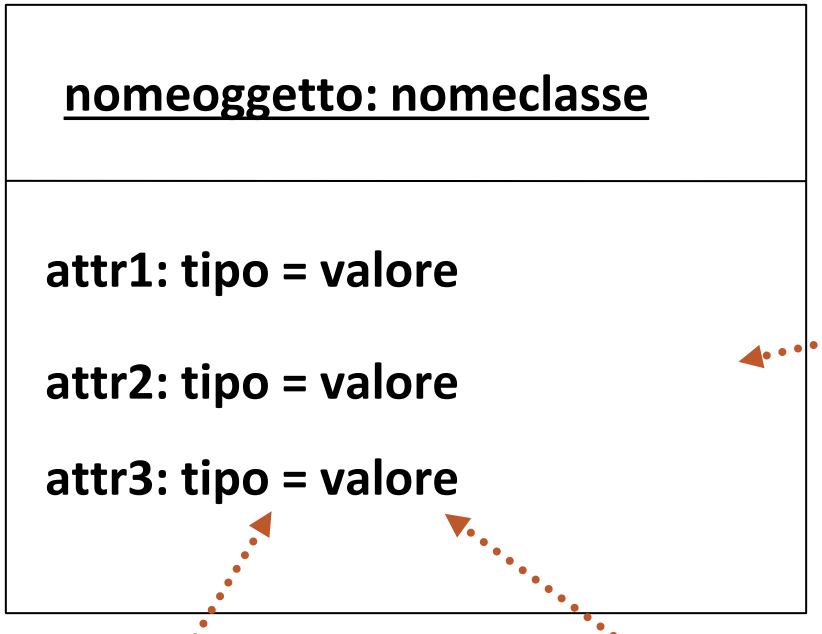


Diagramma degli oggetti: attributi



singoli attributi

opzionali

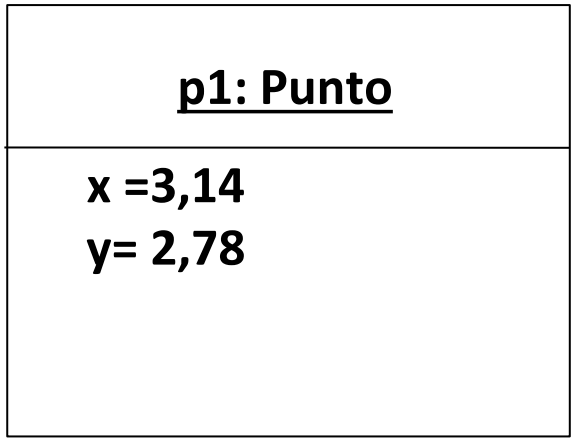
Il valore è la parte interessante può essere omesso ma allora inutile ripetere (vs classe) l'attributo

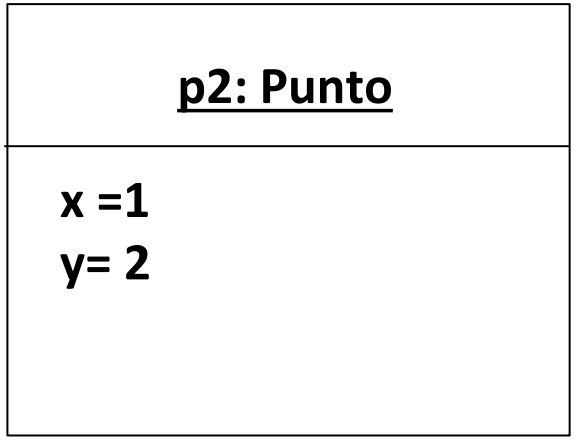
Il tipo è ridondante ed è consigliato ometterlo

Diagramma degli oggetti: collegamenti

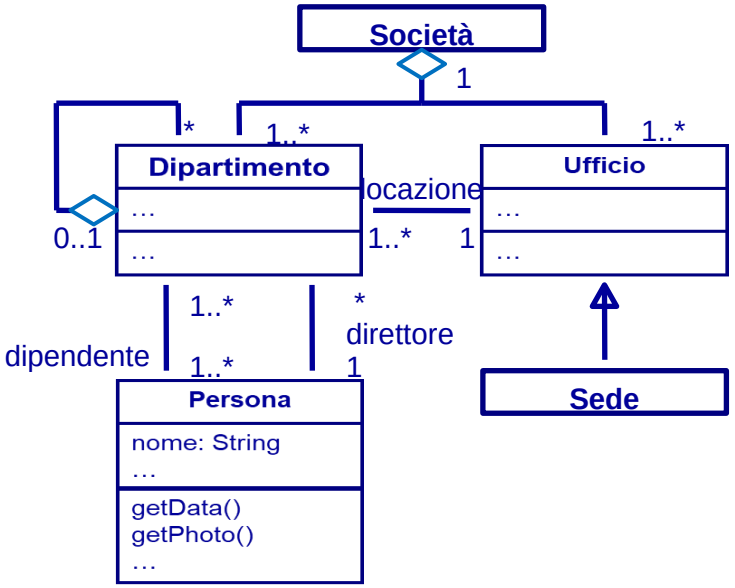
- Un collegamento è una istanza di una associazione
- Collega due (o più) oggetti
- Non ha un nome (se utile si possono indicare i ruoli)
- Non ha molteplicità, è sempre 1 a 1
  - la molteplicità di una associazione dice quanti collegamenti ci saranno a livello di istanza

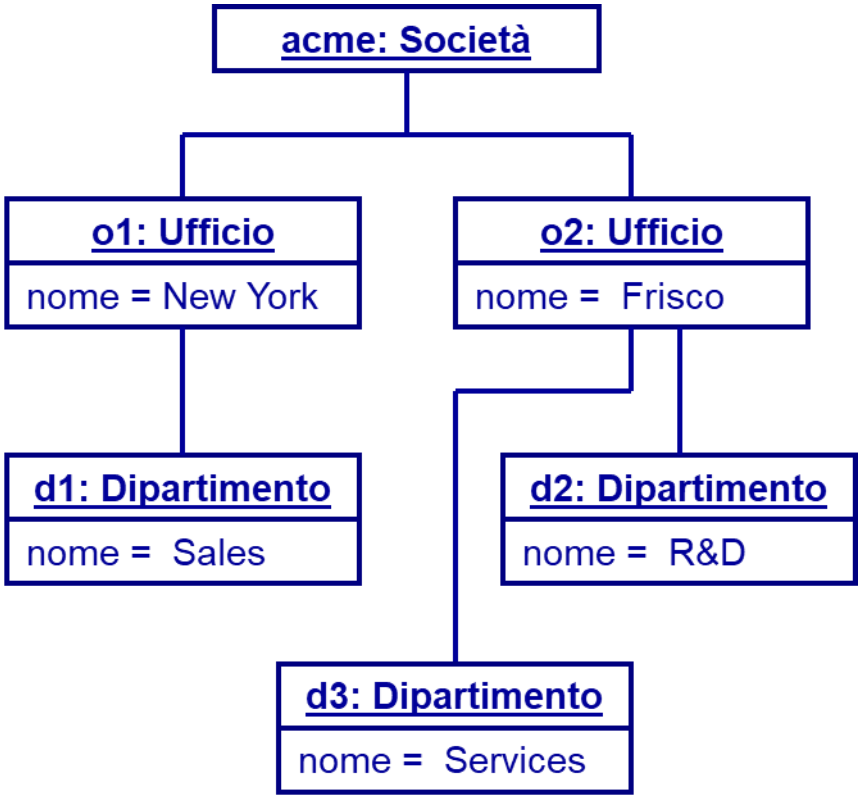
Esempio: Classi e oggetti





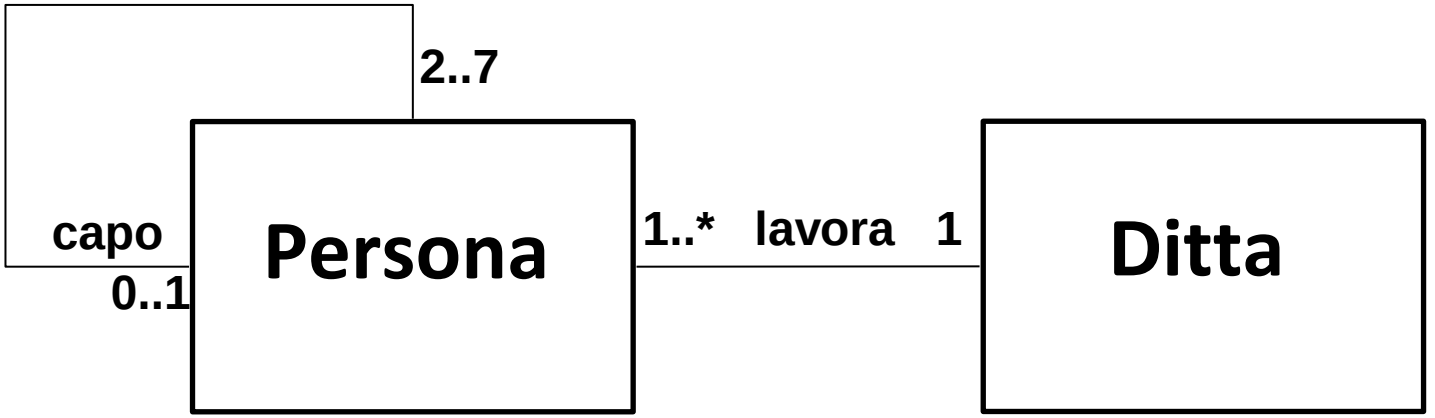
Esempio: oggetti

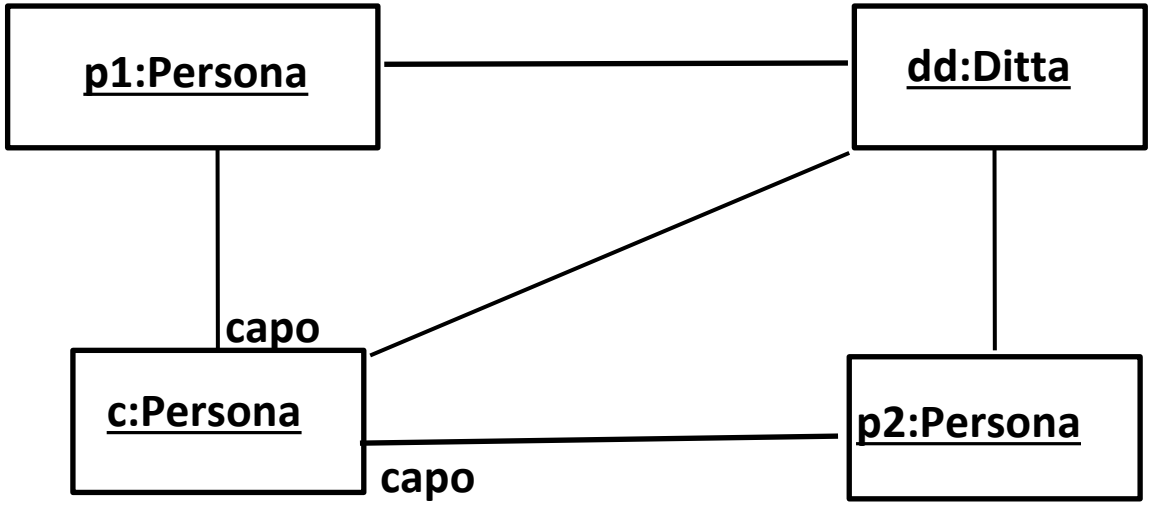




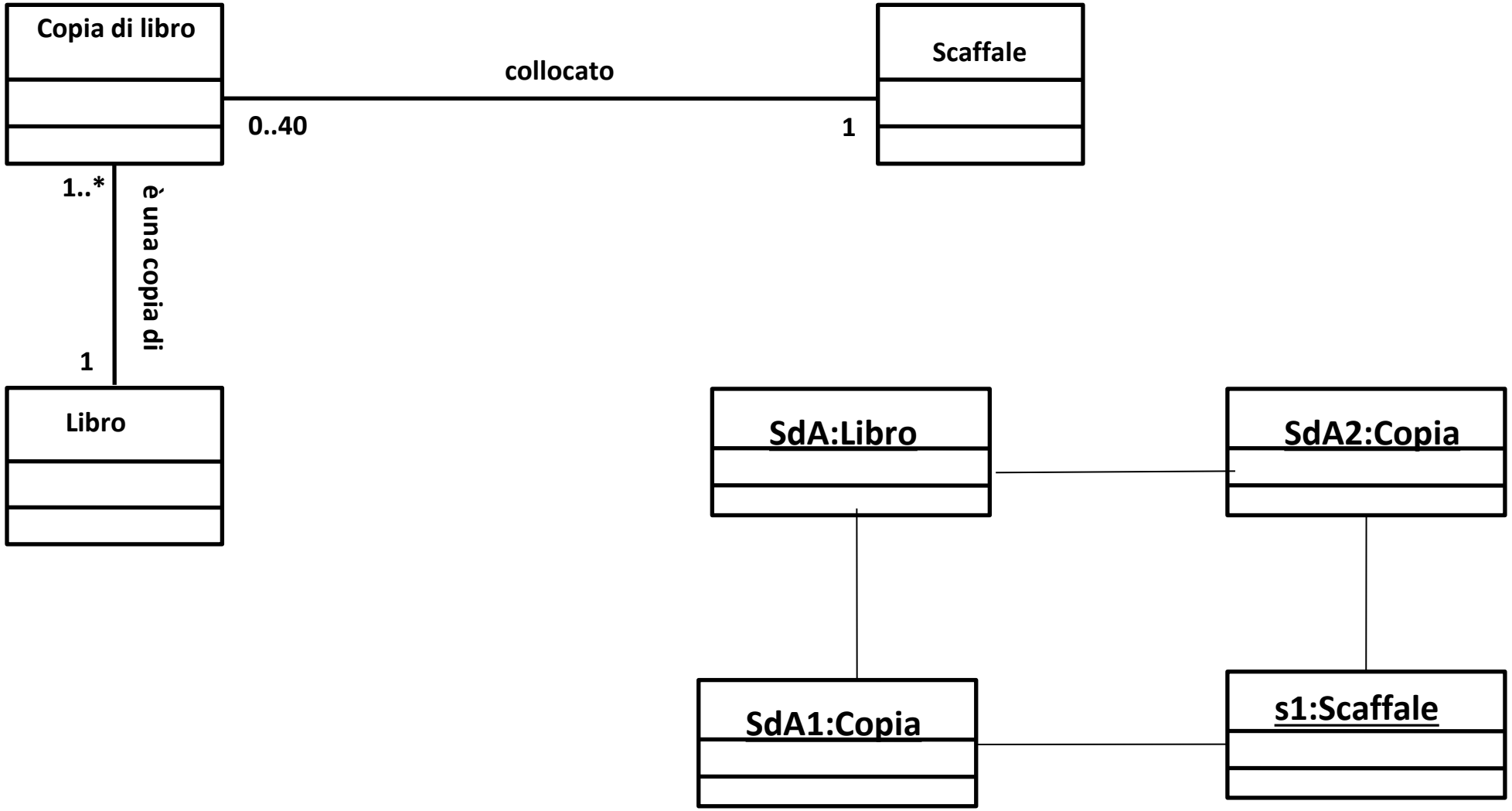
- Una società reale, ACME
- Ha due Uffici
- Il dipartimento vendite è a New York
- I dipartimenti Ricerca&Sviluppo e Servizi sono a San Francisco

Esempio di istanza di classi legate da associazione riflessiva

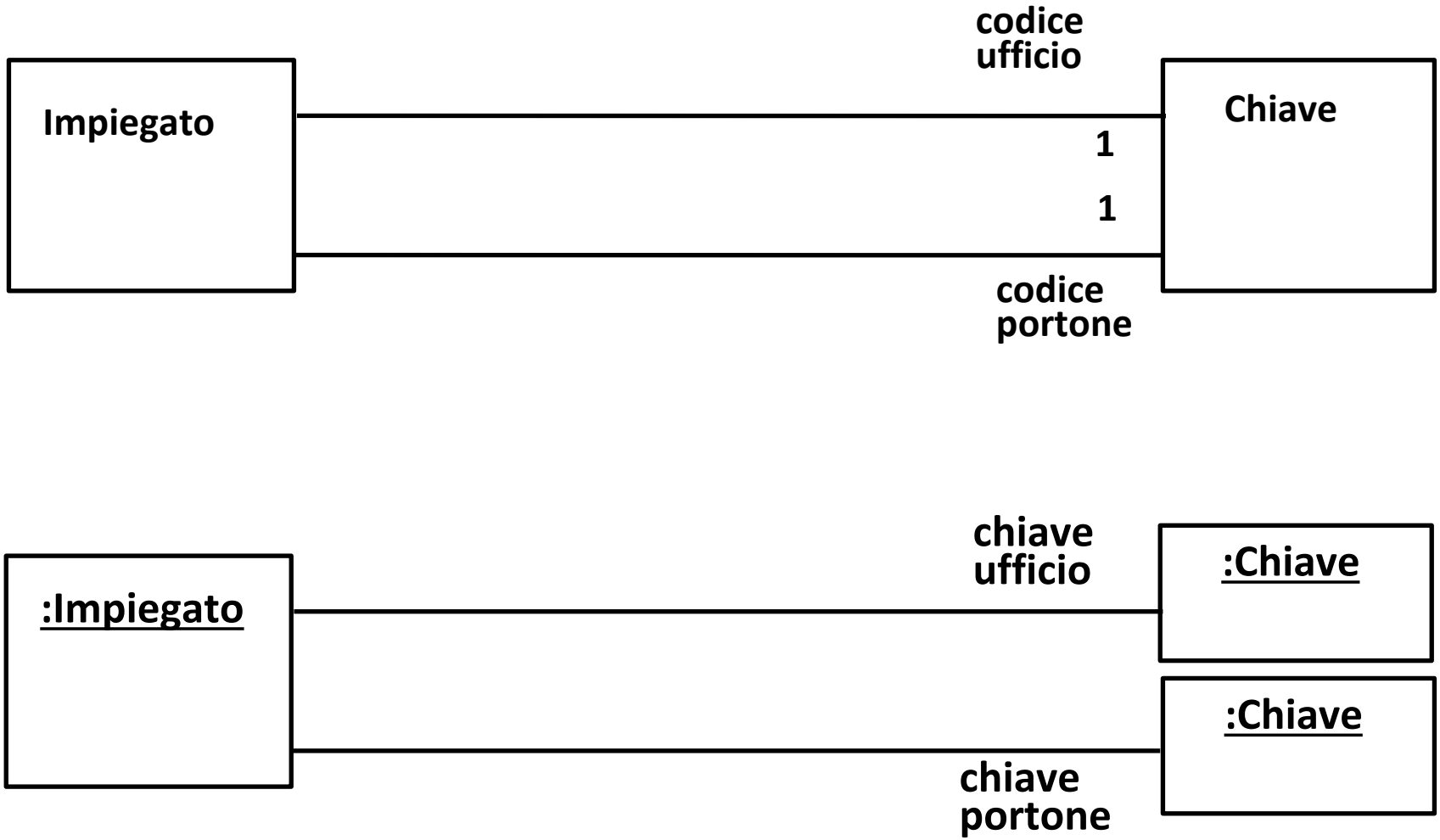




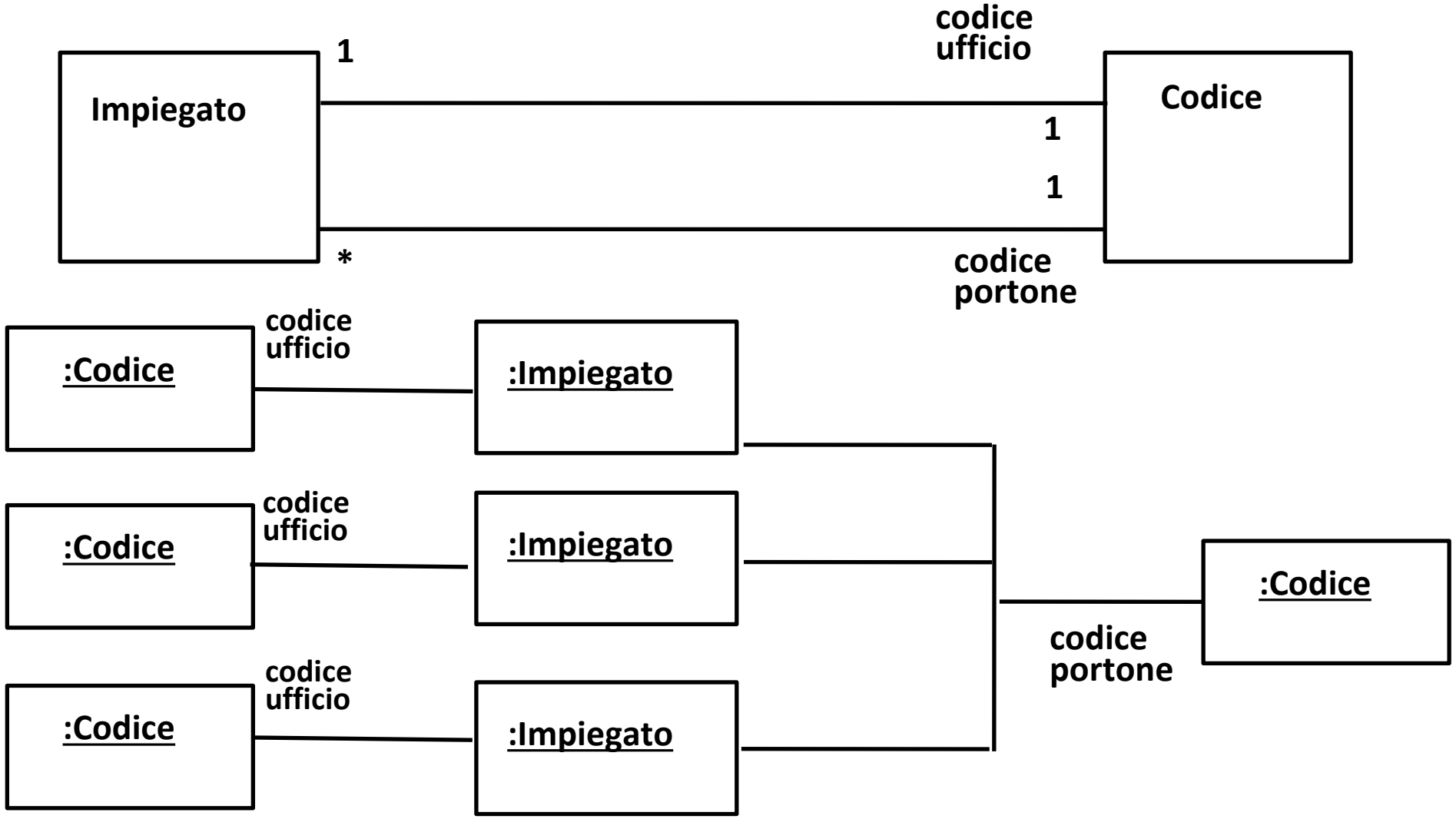
Esempio: Classi e oggetti



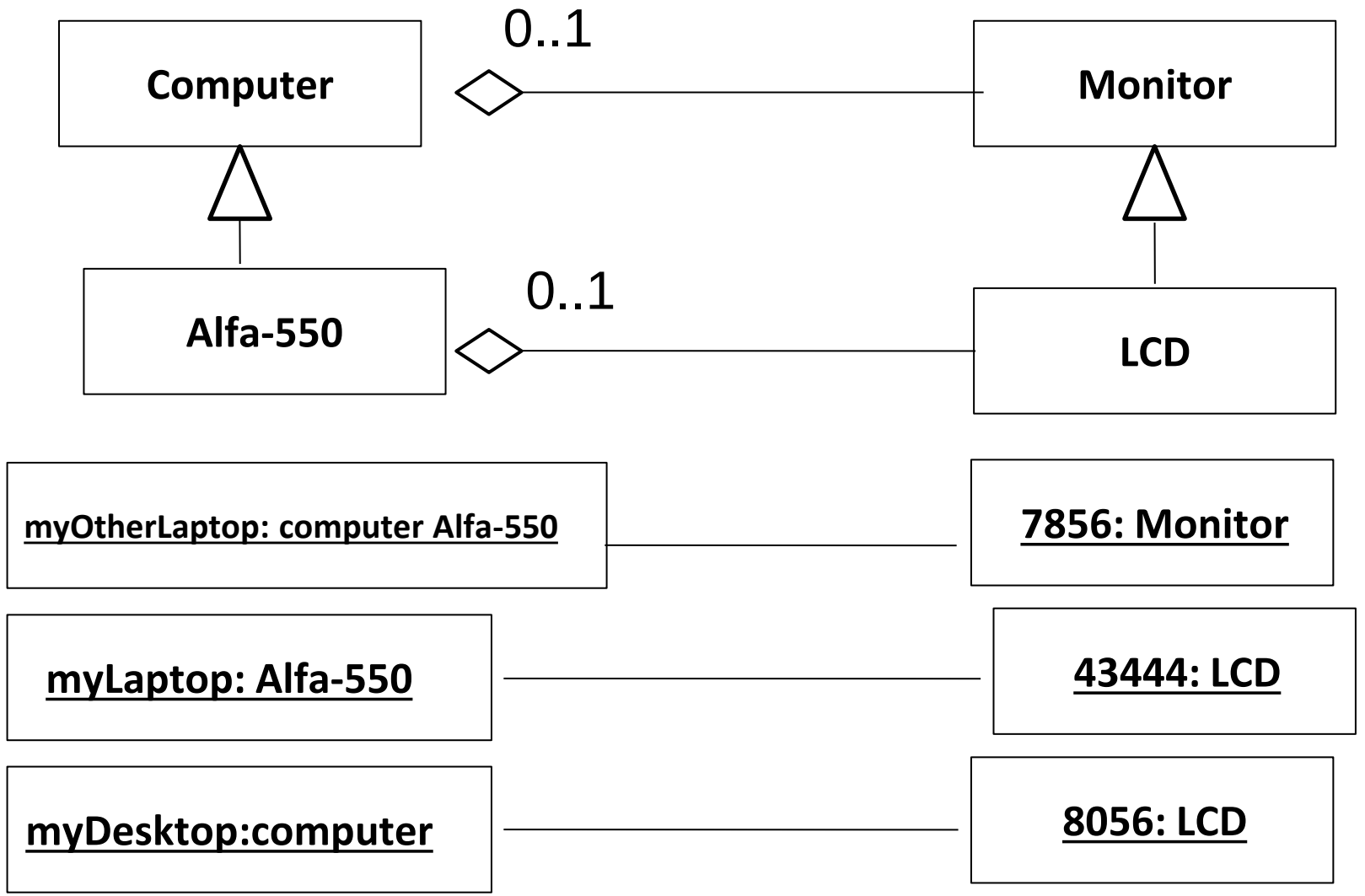
Esempio: Classi e oggetti (ruoli)



Esempio: Classi e oggetti (molteplicità)



Un esempio con generalizzazione e aggregazione



Syllabus

- UML@ Classroom, capitolo 4