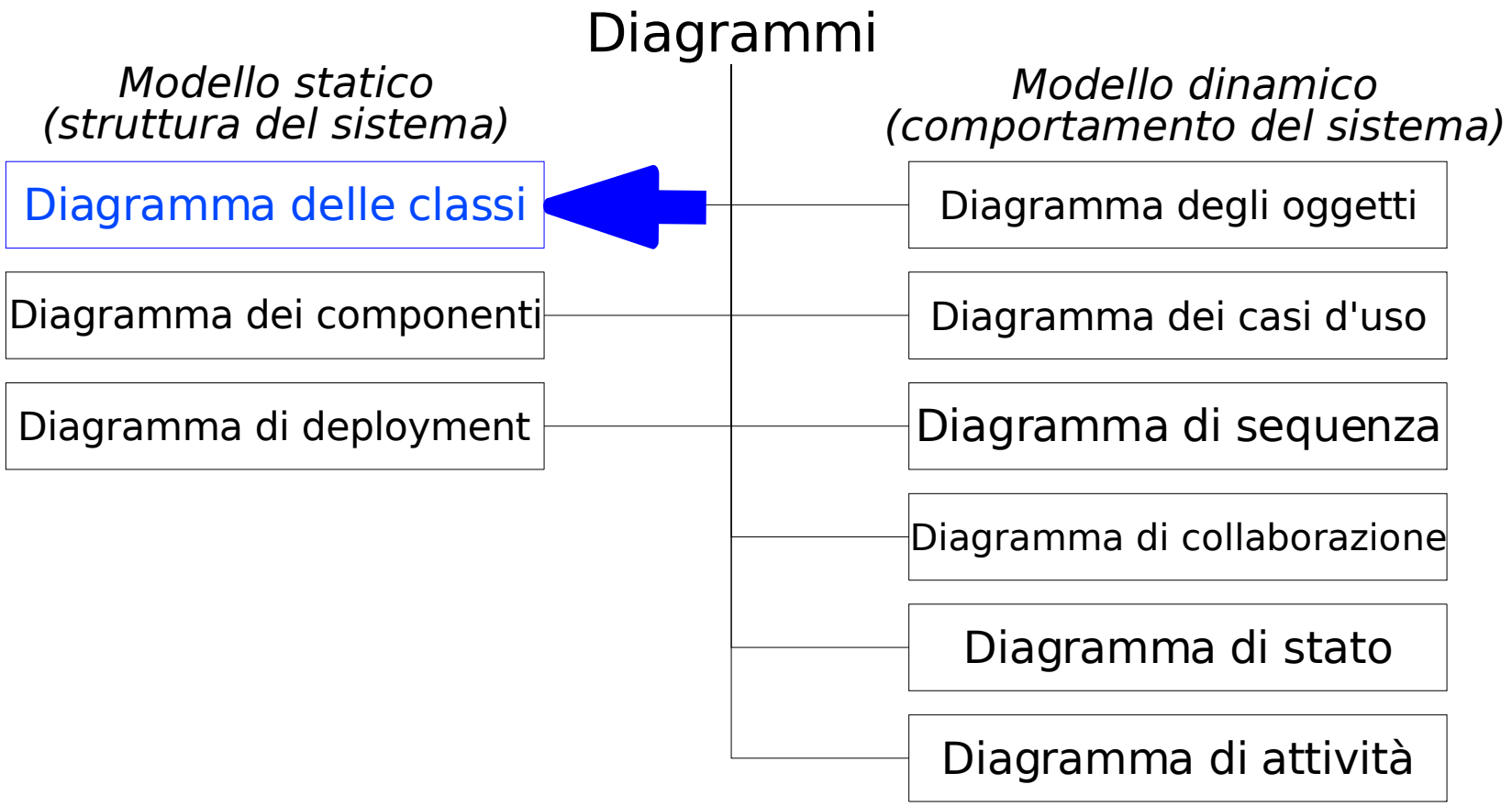
Elementi di UML (3)

Università degli Studi di Bologna
Facoltà di Scienze MM. FF. NN.
Corso di Laurea in Scienze di Internet
Anno Accademico 2004-2005

Laboratorio di Sistemi e Processi Organizzativi

I diagrammi canonici (1.5)



Modellazione strutturale

- ***Modellazione strutturale***: Rappresenta una vista di un sistema software che pone l'accento sulla struttura degli oggetti (classi di appartenenza, relazioni, attributi, operazioni)
- Il ***Diagramma delle classi*** descrive il tipo degli oggetti che compongono il sistema e le relazioni statiche esistenti tra loro

Diagrammi di struttura statica

- Mostrano le entità del sistema connesse secondo le relazioni statiche che le caratterizzano
- Due viste possibili:
  - di classe (*class diagram*)
  - d'istanza (*object diagram*)

Modellazione strutturale: diagrammi

Il modello statico di riferimento per la modellazione strutturale permette di mostrare:

- le entità presenti nel modello (classi, interfacce, componenti, nodi, ecc.)
- la struttura interna
- le relazioni statiche tra entità

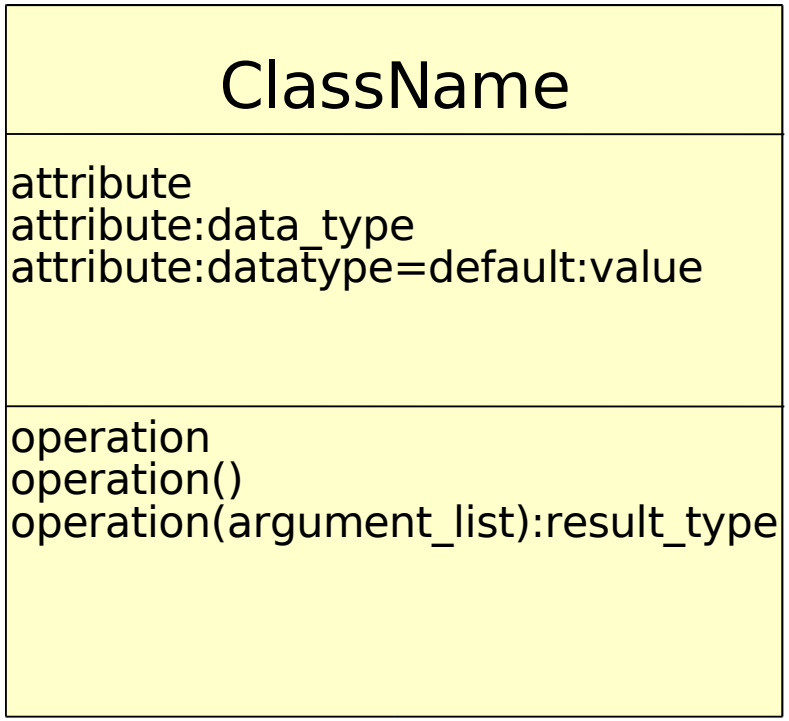
Tipi di diagrammi:

- Diagrammi di struttura:
  - Vista statica: *class diagram*,
  - Vista dinamica: *object diagram*
- Diagrammi d'implementazione: *component diagram*, *deployment diagram*

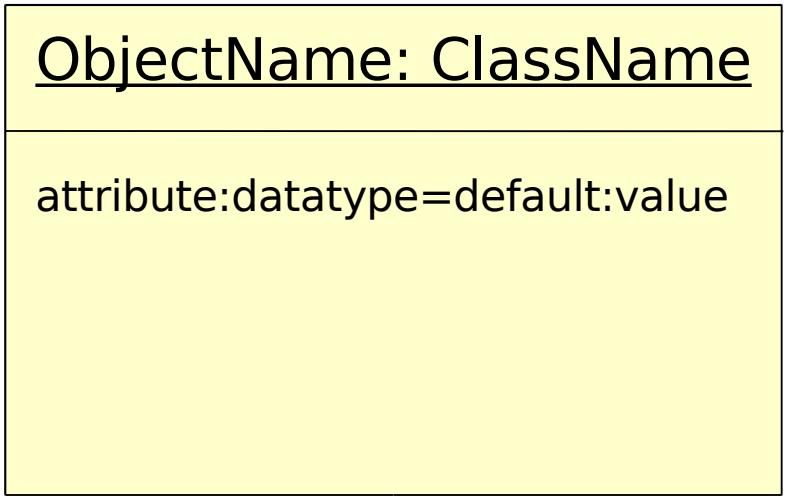
Operazioni, metodi e incapsulazione

- Un oggetto è l'istanza di una classe
- Un oggetto può essere visto come un pacchetto coeso di dati e funzioni
- Per ottenere accesso alla parte dati di un oggetto sono rese disponibili funzioni dell'oggetto apposite
  - Nell'analisi ci si riferisce a tali funzioni come operazioni
  - Nella proggettazione sono chiamate metodi
- Incapsulazione (*opacità* dei dati): mascherare la parte dati di un oggetto sotto uno strato di funzioni.

Le Classi



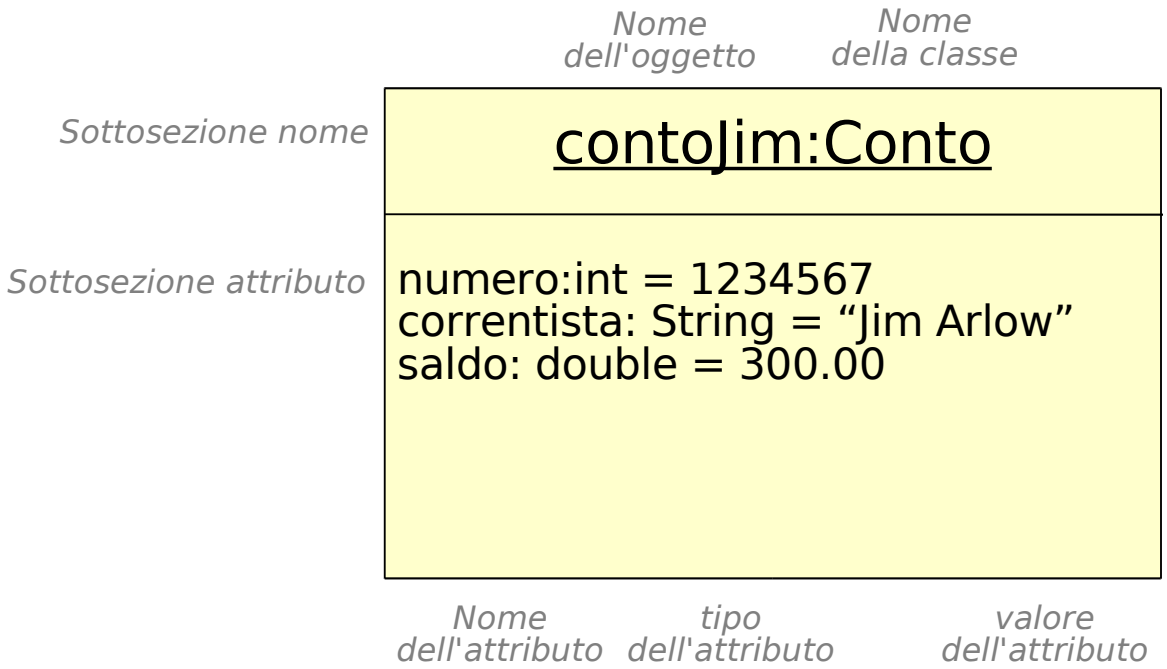
Gli oggetti



Entità: gli oggetti

- In UML l'oggetto è rappresentato tramite un rettangolo con due sottosezioni.
- La sottosezione superiore contiene l'identificatore dell'oggetto che è sempre sottolineato
- L'identificatore di un oggetto può essere uno qualunque dei seguenti:
  - Il solo nome della classe, per esempio: Conto.
  - Il solo nome dell'oggetto, per esempio: contoJim. Questo identifica un oggetto specifico, ma non dice a quale classe esso appartenga. Utile per un'analisi molto preliminare
  - Il nome dell'oggetto concatenato al nome della classe, separati dai due punti: contoJim:Conto

Esempio di oggetto



Entità: le classi

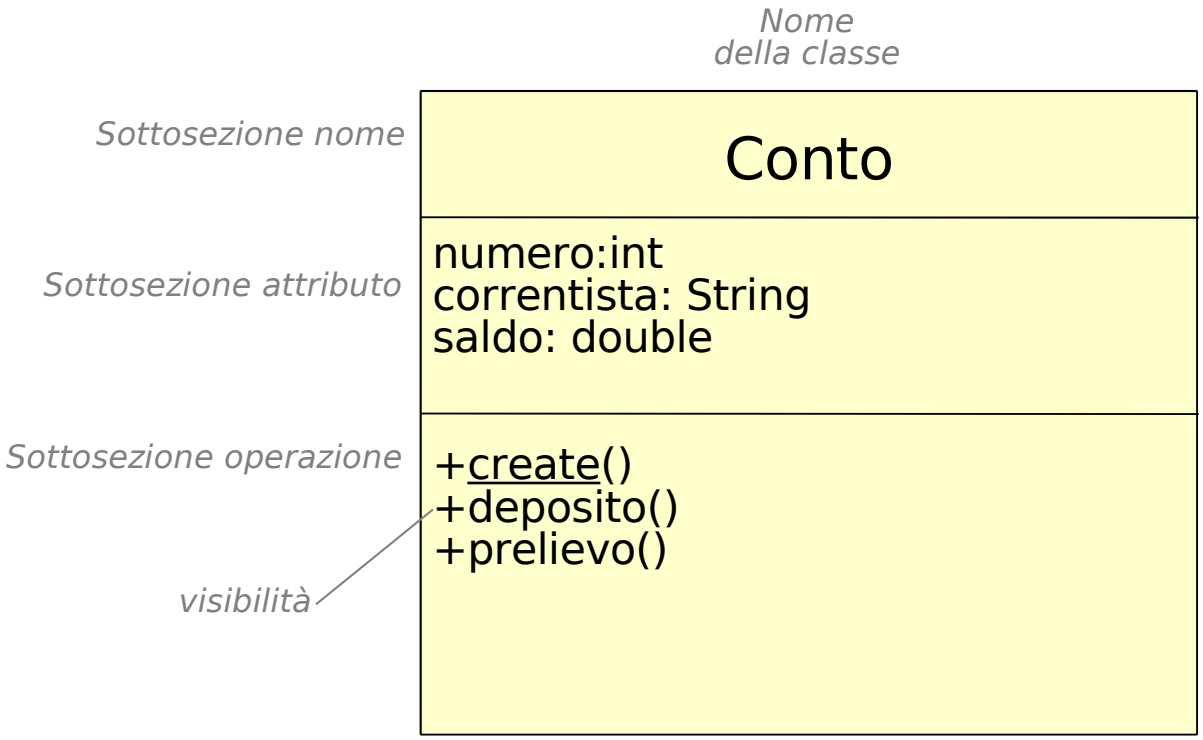
- Una classe descrive un insieme di oggetti che condividono gli stessi attributi, operazioni, metodi, relazioni e semantica

- È rappresentata da un rettangolo

- Solitamente il rettangolo che rappresenta una classe è suddiviso in 3 compartimenti:
  1- Nome
  2- Attributi: *attribute:Type=”default value”*
  3- Metodi: *MethodName(List of parameters): ReturnType*

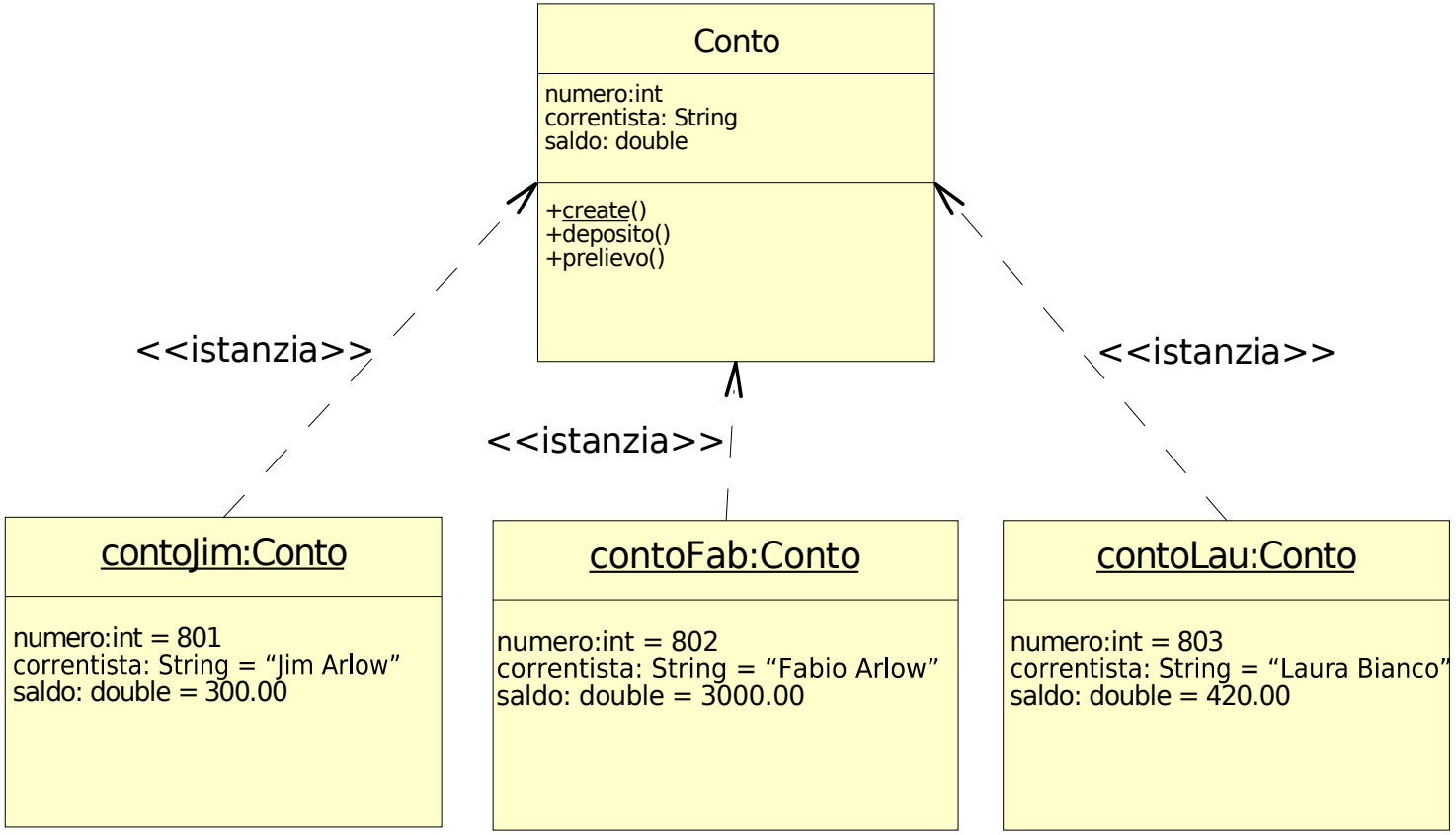
- E' possibile personalizzare il numero e il tipo di compartimenti specificando il nome di ogni compartimento

Esempio di classe



Classi e oggetti

La relazione esistente tra una classe e gli oggetti di quella classe è la relazione <<istanzia>>



Attributo

- Il nome dell'attributo è l'unica parte obbligatoria

visibilità	nome	molteplicità:tipo=valoreIniziale
opzionale	obbligatorio	opzionale

- Nella proggettazione i valori iniziali aiuta a garantire che gli oggetti siano crati con uno stato iniziale utilizabile e consistente
- Nell'analisi i valori iniziali aiutano ad evidenziare vincoli del problema

Tipi di visibilità

+ (visibilità pubblica): ogni elemento che può accedere alla classe può anche accedere a ogni suo membro con visibilità pubblica

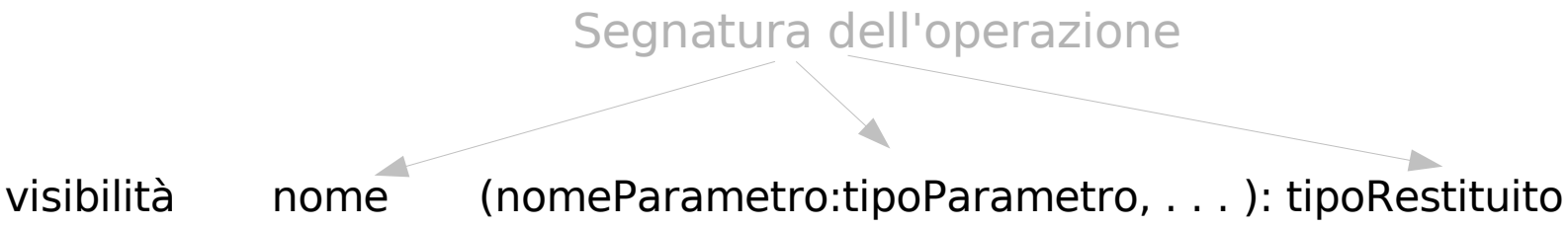
- (visibilità privata): solo le operazioni della classe possono accedere ai membri con visibilità privata

(visibilità protetta): solo le operazioni appartenenti alla classe o ai suoi discendenti possono accedere ai membri con visibilità protetta

~ (visibilità package): ogni elemento nello stesso package della classe (o suo sottopackage annidato) può accedere ai membri della classe con visibilità package

Operazione

- Le operazioni sono funzioni vincolate a una certa classe

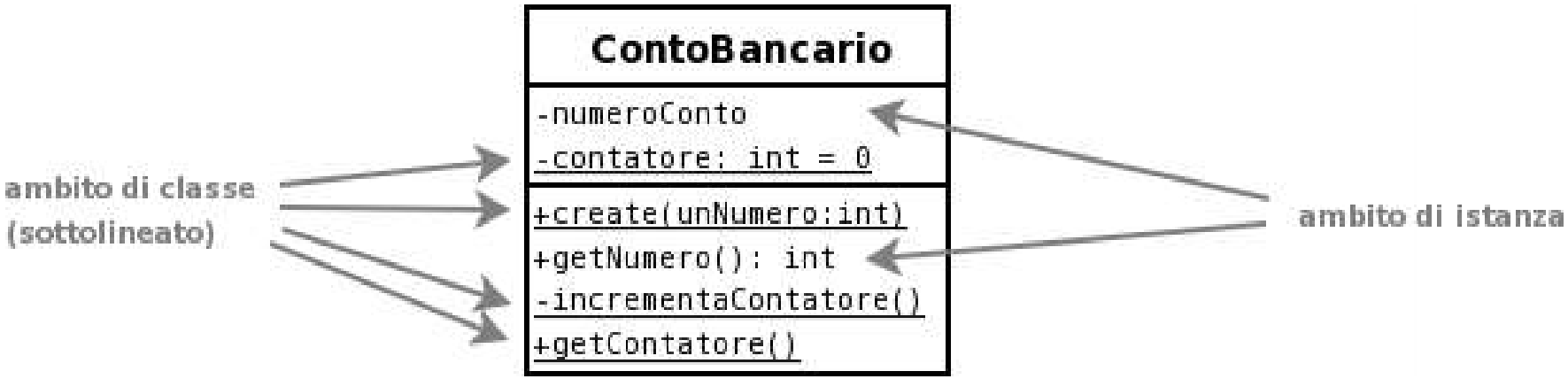


- La segnatura è costituita da:
  - Nome dell'operazione (scritto in *camelCase*)
  - Tipi di tutti i parametri
  - Tipo restituito
- Ogni operazione di una classe deve avere una sua unica segnatura

Ambito

- Ambito di istanza:
  - Gli oggetti hanno una propria copia degli attributi, quindi oggetti diversi possono avere diversi valori negli attributi
  - Le operazioni agiscono su oggetti specifici
- Ambito di classe:
  - Gli oggetti di una stessa classe condividono lo stesso valore per un attributo
  - Le operazioni non operano solo su una particolare istanza della classe, ma alla classe stessa. Ad esempio: costruttori e distruttori di classe.

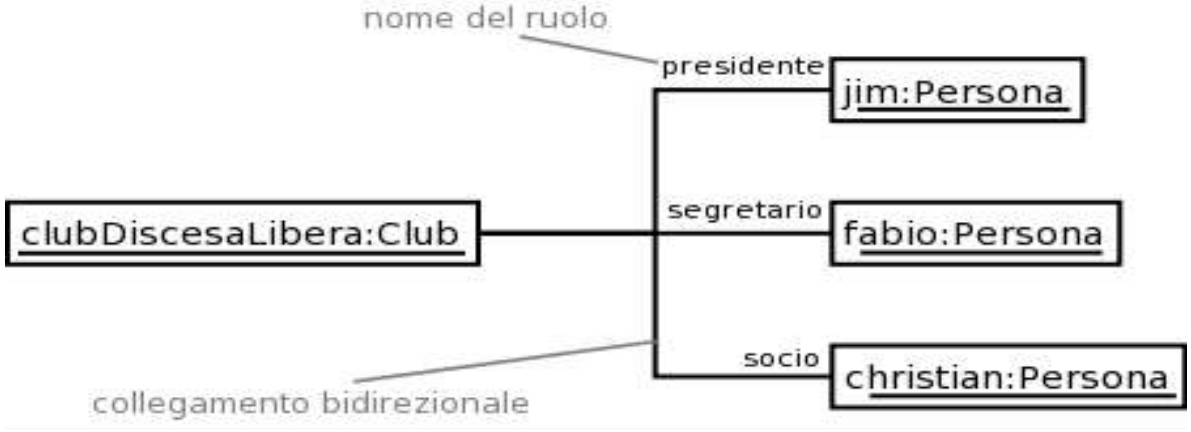
Ambito e accessibilità



- Operazioni con ambito di istanza possono accedere sia ad altre operazioni o attributi con ambito di istanza sia con ambito di classe
- Operazioni con ambito di classe possono accedere solo ad altre operazioni e attributi con ambito di classe (altrimenti non si saprebbe quale istanza scegliere)

Diagrammi degli oggetti

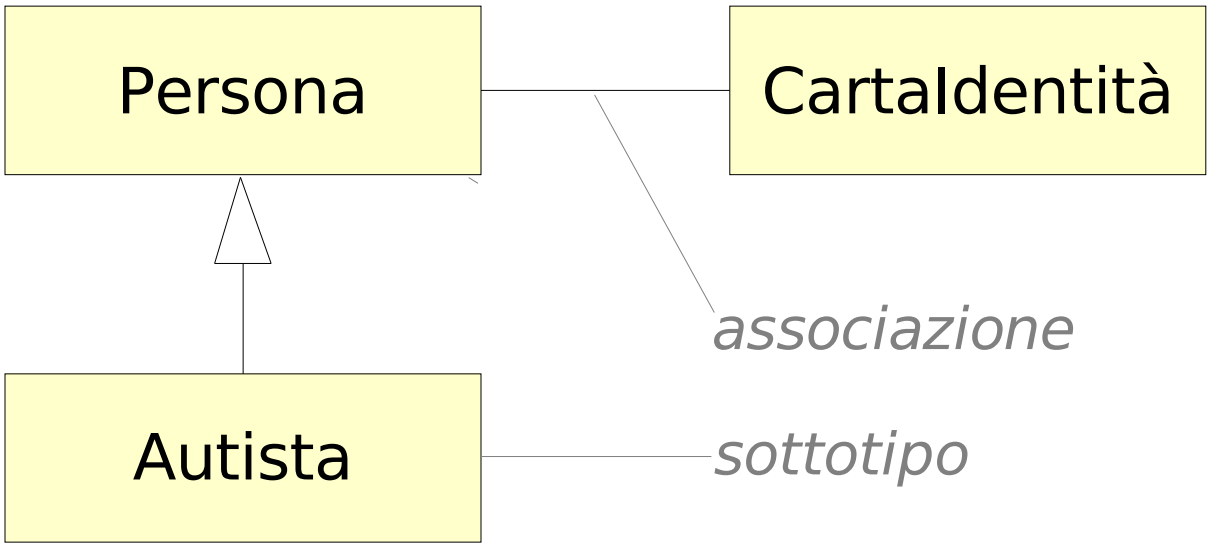
- Gli oggetti collegati assumono dei ruoli l'uno rispetto all'altro
- I collegamenti sono connessioni dinamiche tra oggetti, quindi non necessariamente stabili nel tempo



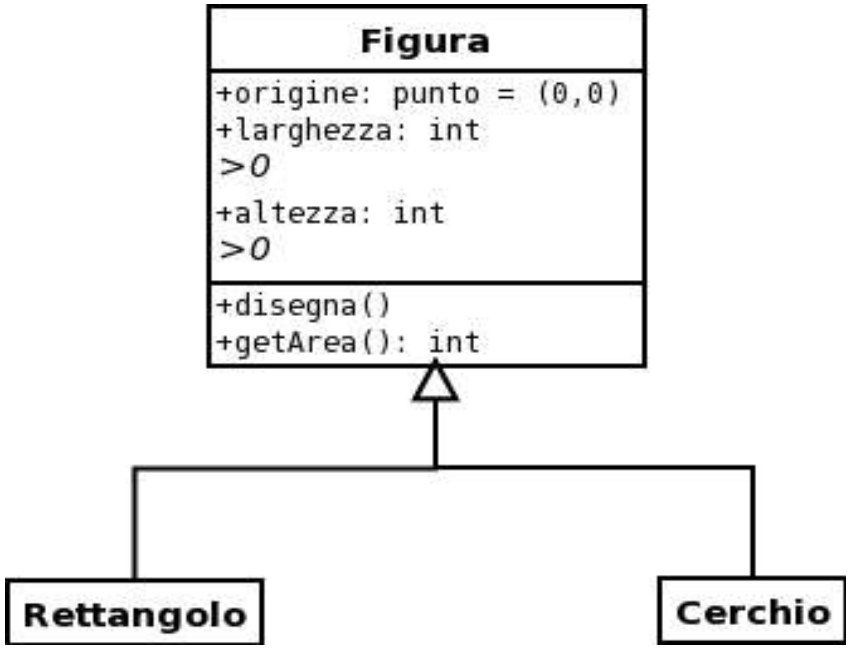
Le relazioni statiche

Ci sono due tipi principali di relazione statica tra classi in un *class diagram*

- Associazione
- Sottotipo



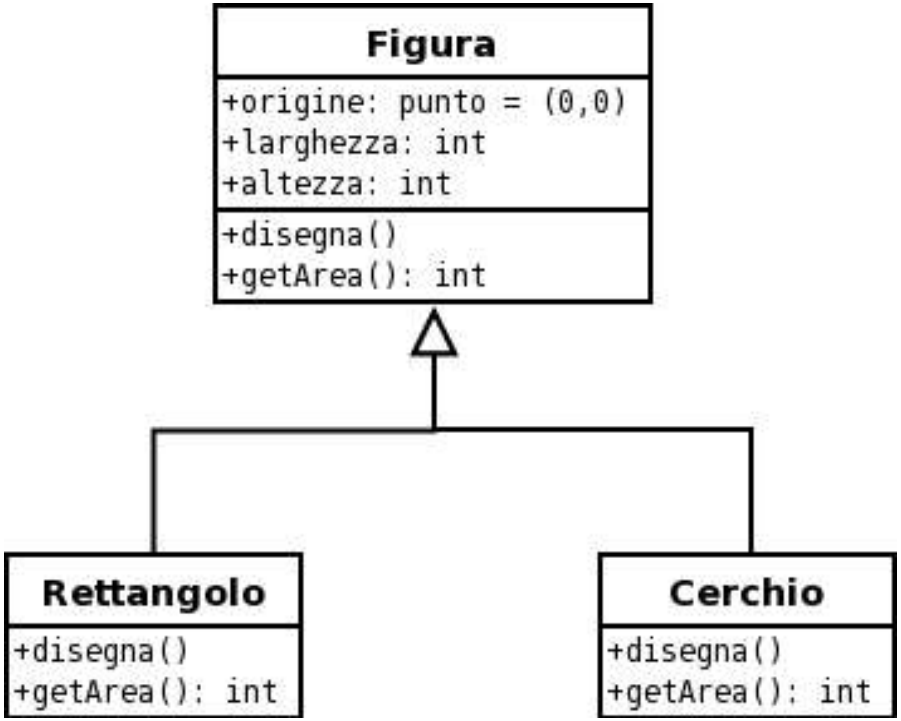
Esempio



- Le sottoclassi **Rettangolo** e **Cerchio** ereditano tutti gli attributi, operazioni e vincoli della superclasse **Figura**
- Che errore di modellazione è stato commesso?

Esempio: overriding

- Per ridefinire una operazione della superclasse, una sottoclasse deve avere una propria operazione con identica *segnatura*
- Ad esempio, la soluzione al problema della slide precedente è:

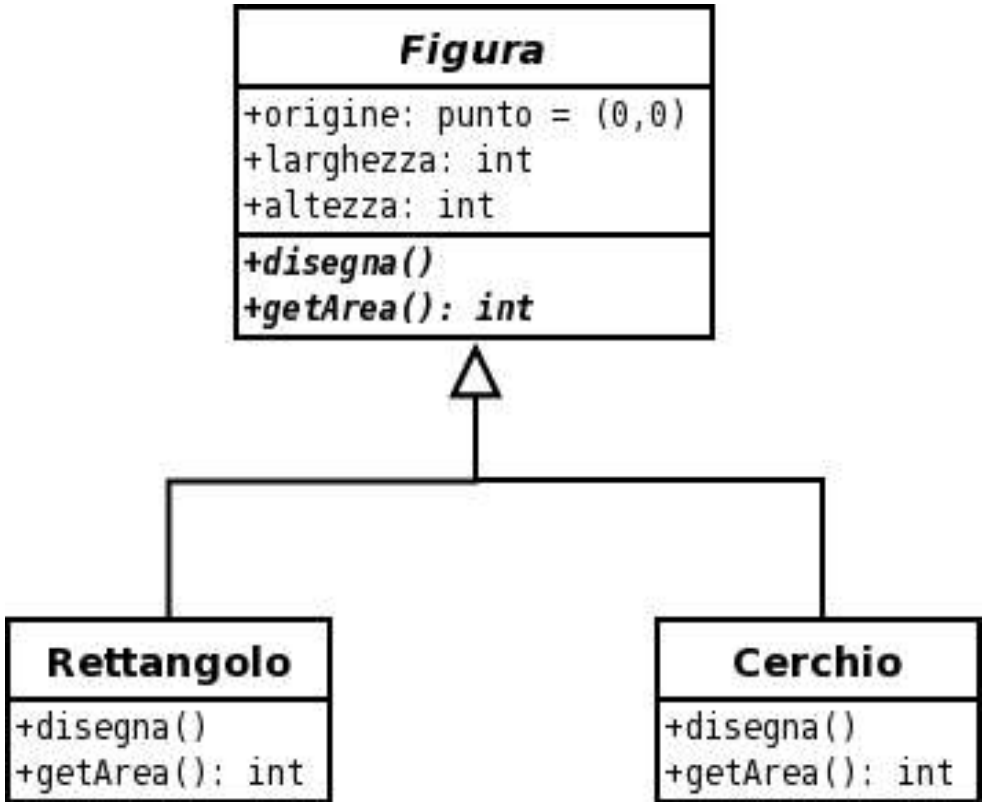


Classe Astratta e Polimorfismo

- *Operazione astratta*: operazione priva di implementazione la cui implementazione è demandata alle sottoclassi
- Una classe con una o più operazioni astratte è astratta e non può essere istanziata
- Operazioni e classi astratte sono indicate scrivendone il nome in *corsivo*
- Le classi che possono essere istanziate sonno dette classi concrete
- Operazioni che possono avere implementazioni differenti sono dette polimorfe

Esempio: classe astratta e polimorfismo

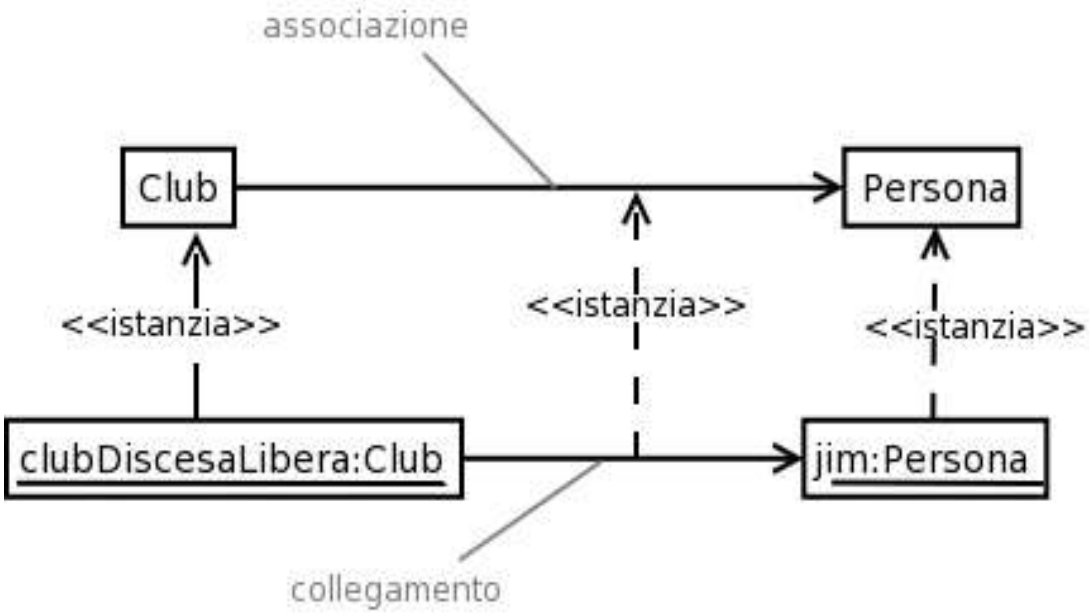
Grazie al polimorfismo oggetti di classi differenti (come **Rettangolo** e **Cerchio**) hanno operazioni con la stessa segnatura, ma con implementazione diversa.



Cos'è un'associazione?

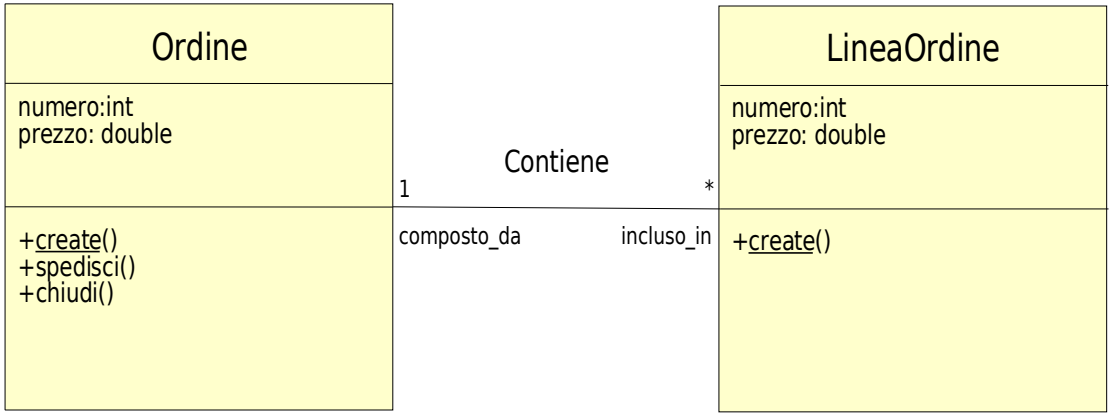
- Se esiste un collegamento tra oggetti allora deve essistere un'associazione tra le loro classi,

  poiché un collegamento è l'istanziazione di un'associazione



Le associazioni

- Un'associazione rappresenta una connessione tra due o più classi
- La classe ha la responsabilità di notificare una certa informazione ad un'altra classe
- Bidirezionale
- La molteplicità indica quanti oggetti di una classe possono far riferimento ad ogni oggetto dell'altra



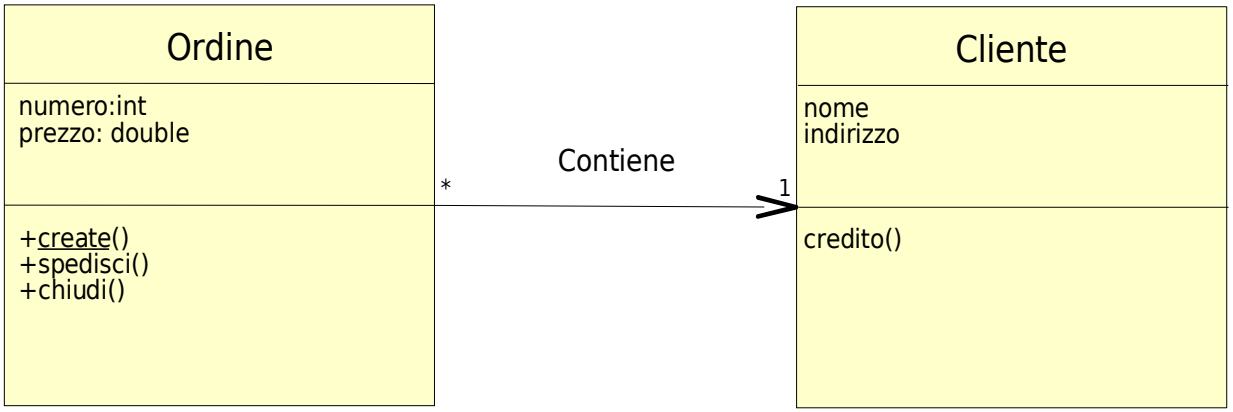
Le associazioni

- Molteplicità:

Un'Azienda impiega molte Persone

Una Persona lavora per un'unica Azienda

- Navigabilità:

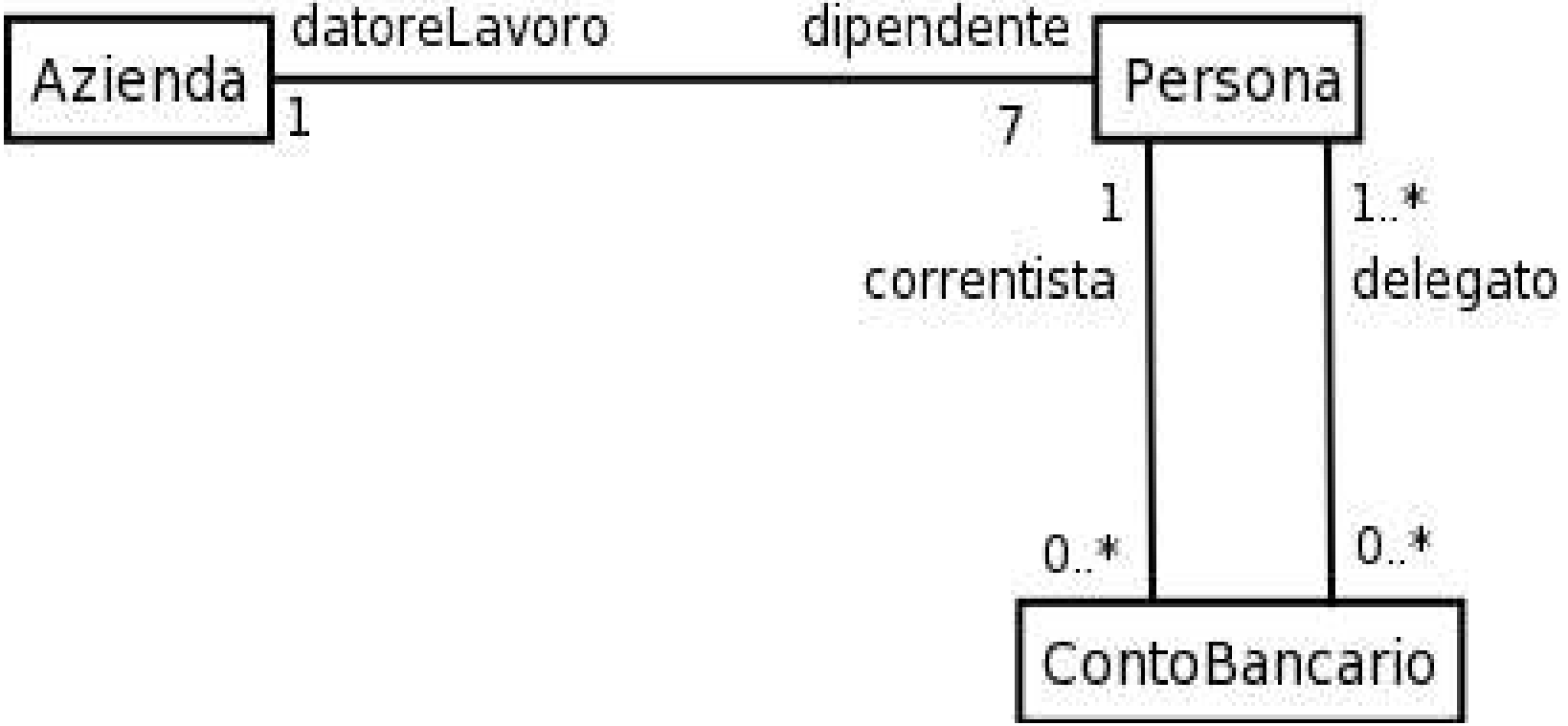


Le molteplicità

Le molteplicità sono della forma:

- 0..1 Zero o 1
- 1 Esattamente 1
- 0..* Zero o più
- * Zero o più
- 1..* 1 o più
- 1..6 da 1 a 6
- 1..3,7 da 1 a 3, *oppure* 7

Esempio

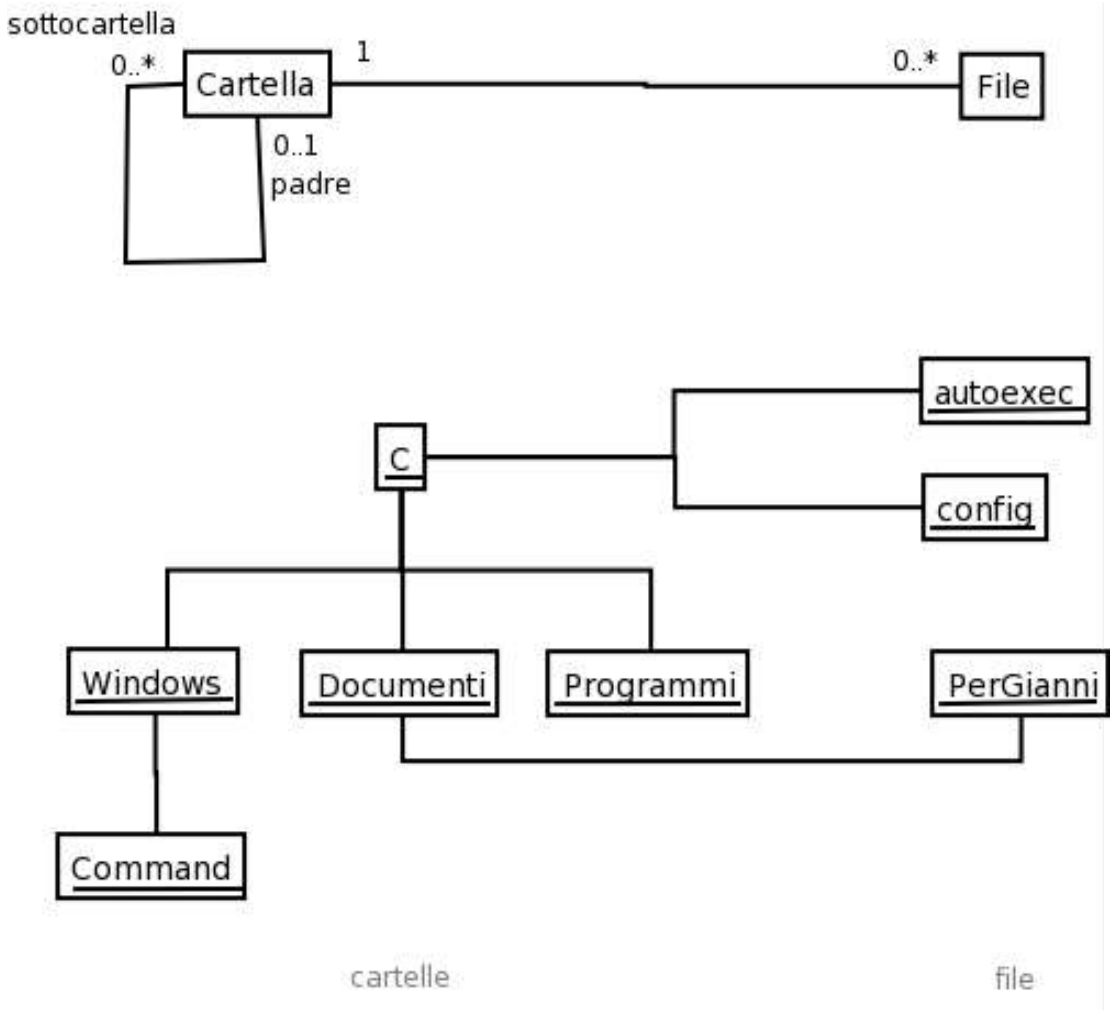

Azienda
datoreLavoro
1
dipendente
7
Persona
1
correntista
1..*
delegato
0..*
0..*
ContoBancario

Associazioni riflessive

- Si ha un'associazione riflessiva se
  - una classe ha una associazione con se stessa
  - quindi oggetti della classe possono avere collegamenti con oggetti della stessa classe
- Ad esempio:

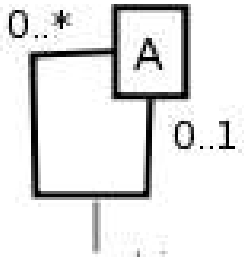
A partire da questo *diagramma di classe*,
quale può essere un possibile *diagramma degli oggetti*?

Esempio: associazione riflessiva



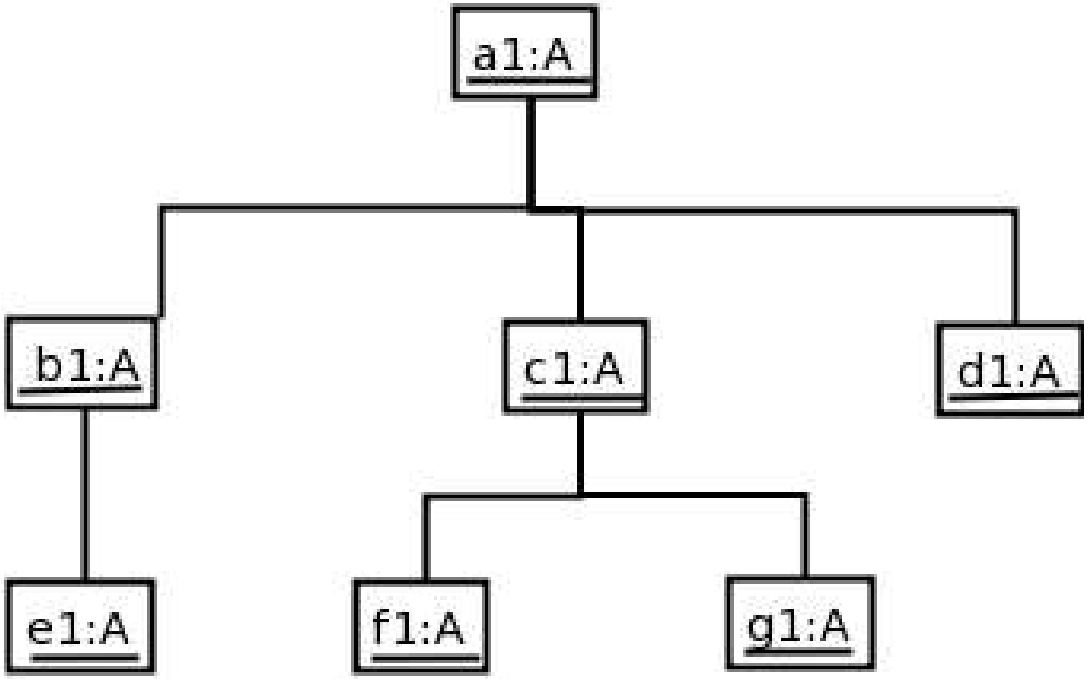
Esempio: gerarchia

Diagramma delle classi



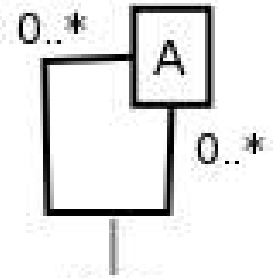
gerarchia di associazione

esempio di diagramma degli oggetti

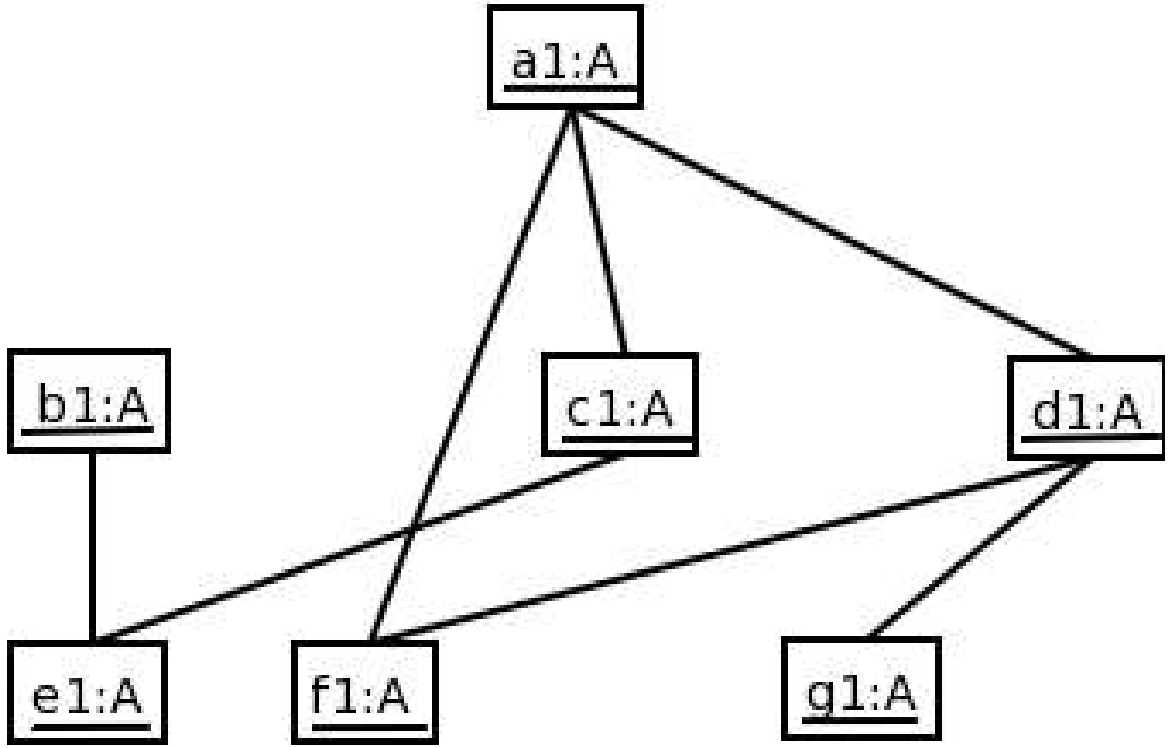


Esempio: rete

Diagramma
delle classi

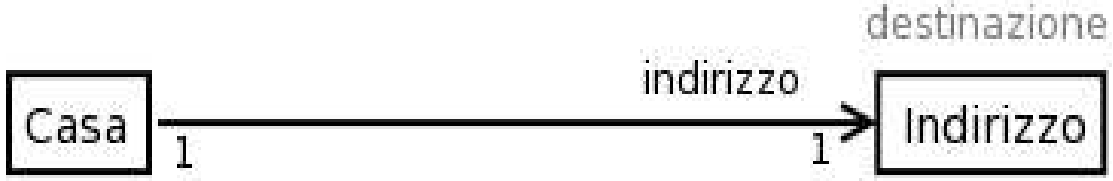


esempio di diagramma degli oggetti



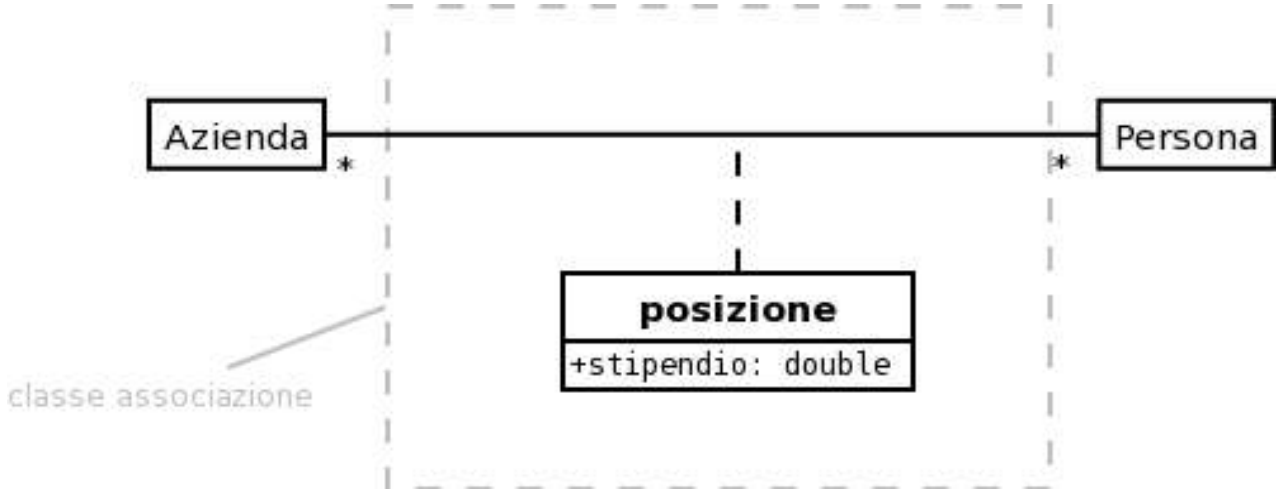
Associazioni e attributi

- Che correlazione c'è tra *associazioni* e *attributi* delle classi?
- Ad esempio:



Classi associazione

- Aggiungiamo la regola per cui ogni **Persona** percepisce uno stipendio da ogni **Azienda** in cui è impiegata
- Dove sistemiamo l'attributo stipendio?
- Lo stipendio è una proprietà dell'associazione stessa



Classi associazione

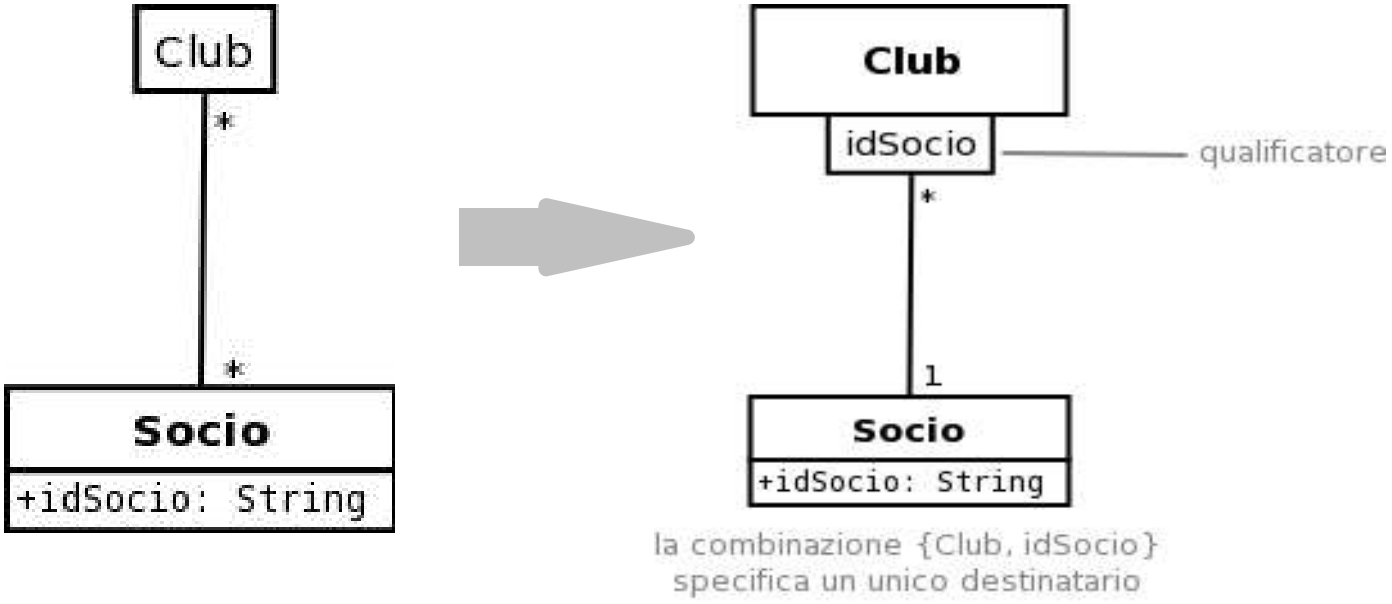
- La classe associazione che connette due classi e definisce un insieme di caratteristiche proprie della associazione stessa
- Le istanze dell'associazione sono collegamenti, dotati di attributi e operazioni, la cui identità individuale è stabilita esclusivamente dalle identità degli oggetti alle estremità del collegamento
- Quindi si possono usare le classi associazione solo quando ogni collegamento ha una identità univoca
- Nell'esempio precedente si sta affermando che una **Persona** può avere una sola **Posizione** in una data **Azienda**

Reifica di una classe associazione

- Come si reifica (o rende reale) la relazione di classe associazione?
- Trasformandola in una vera classe
- ovvero:

Associazioni qualificate

- Sono elementi di modellazione utili per illustrare come reperire degli oggetti specifici di un insieme
- Dato un oggetto **Club** collegato a un insieme di oggetti **Socio**, come si può navigare fino a uno specifico oggetto **Socio**?



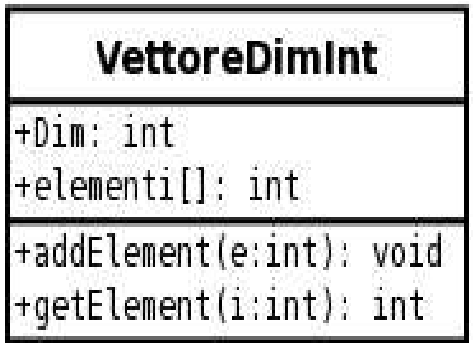
Classi di progettazione

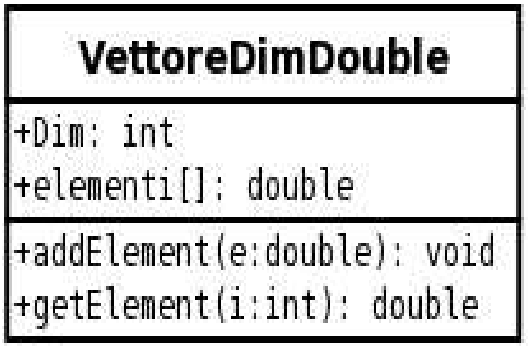
- Le classi di progettazione sono classi le cui specifiche sono talmente complete da poter essere implementate
- Una classe di progettazione è benformata se rispetta le seguenti caratteristiche:
  - completezza e sufficienza;
  - essenzialità;
  - massima coesione;
  - minima interdipendenza.

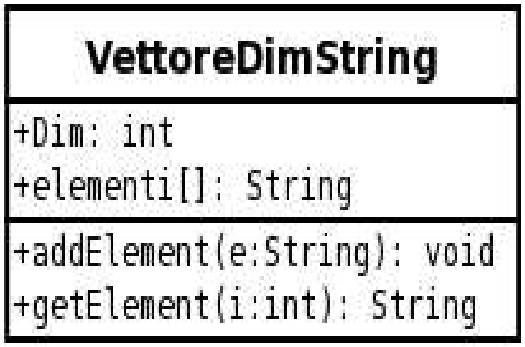
Template di classe

- I **template di classe** consentono la parametrizzazione dei tipi
- Al posto di specificare il tipo effettivo degli attributi e dei parametri e valori restituiti dei metodi, è possibile definire una classe, facendo riferimento a dei parametri
- Tali parametri possono essere sostituiti da dei valori effettivi per creare delle nuove classi

Esercizio sui Template di classe

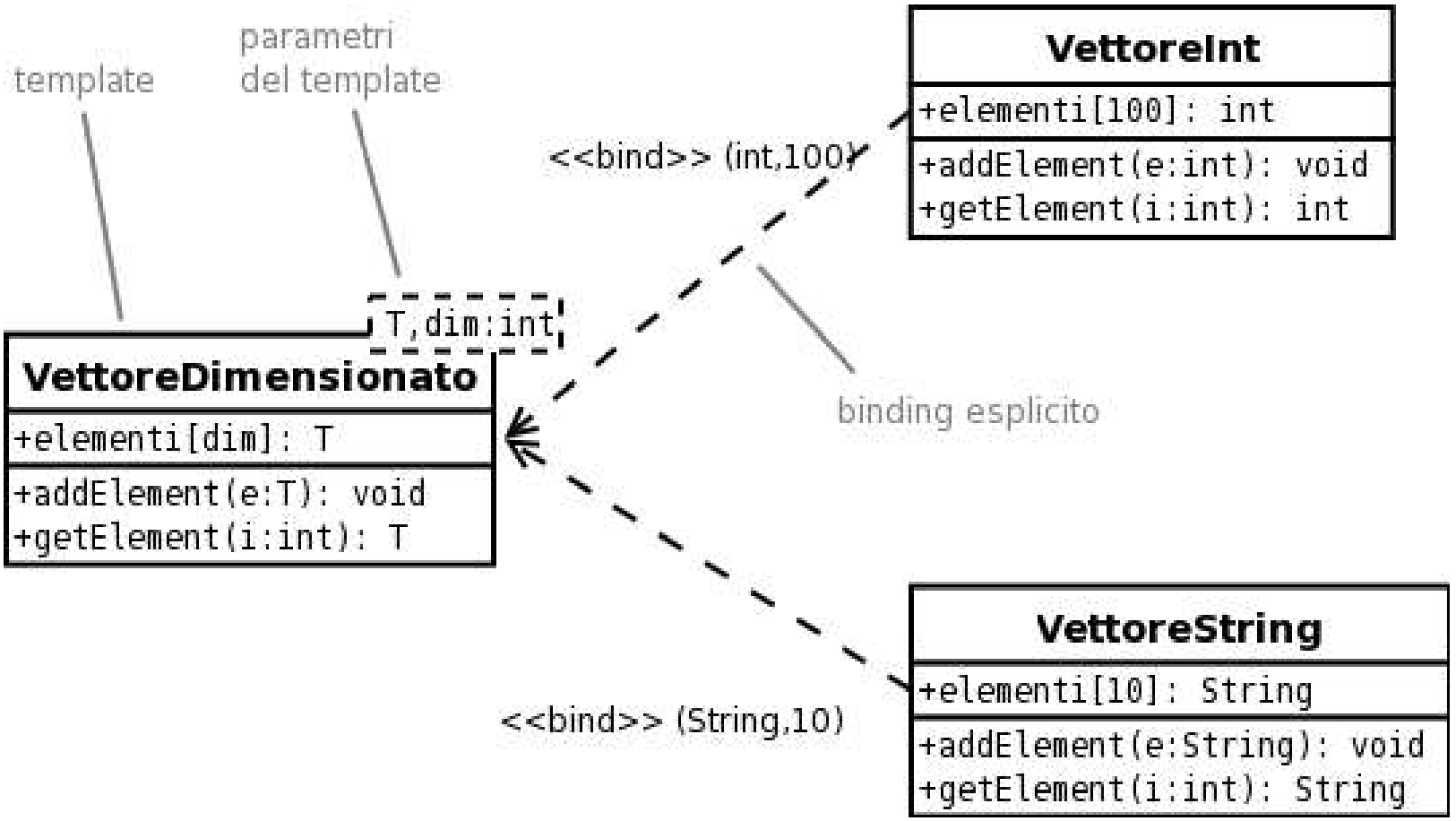






Date queste classi, quali sono i parametri?

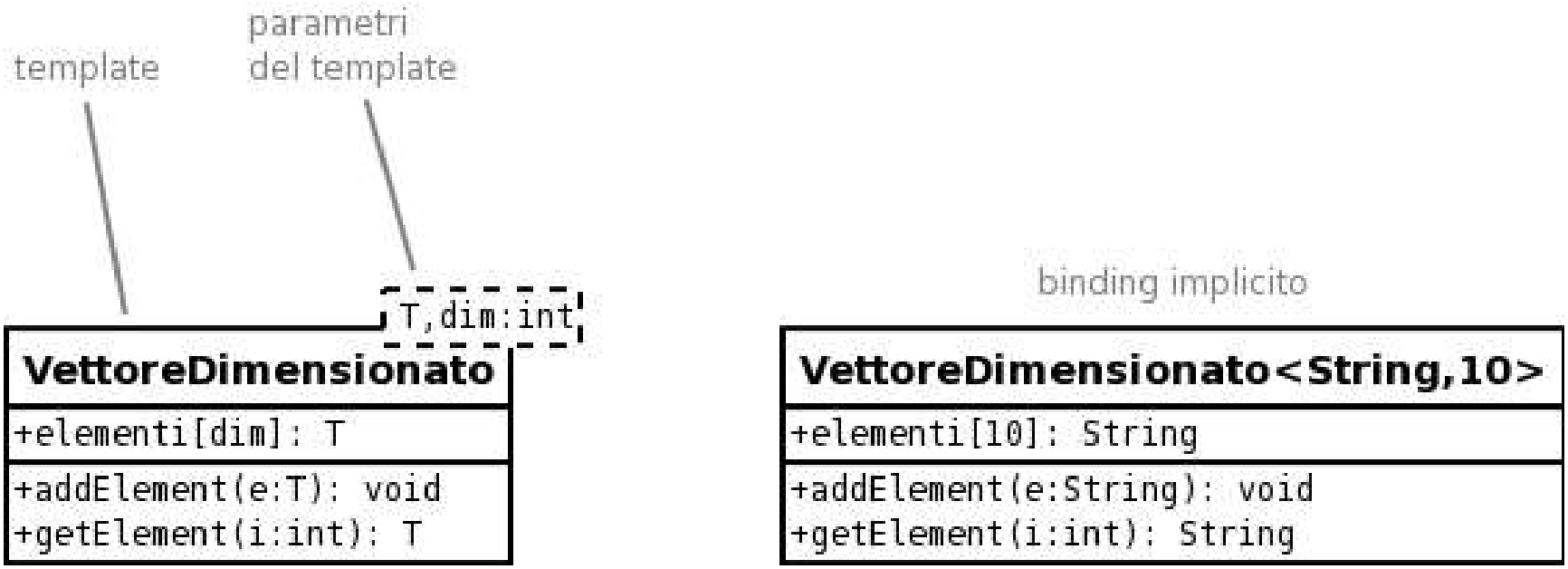
Soluzione dell'esercizio sui template



Istanziazione del template

- I nomi dei parametri sono locali al template
- Operazione di *binding*: associare ai parametri dei valori specifici
- Istanziazione di un template: crare nuove classi tramite l'operazione di *binding*
- Binding esplicito: istanziare il template usando la relazione di dipendenza con lo stereotipo <<bind>>
- Binding implicito: nel nome della classe si usa il nome del template seguito dai valori per i parametri

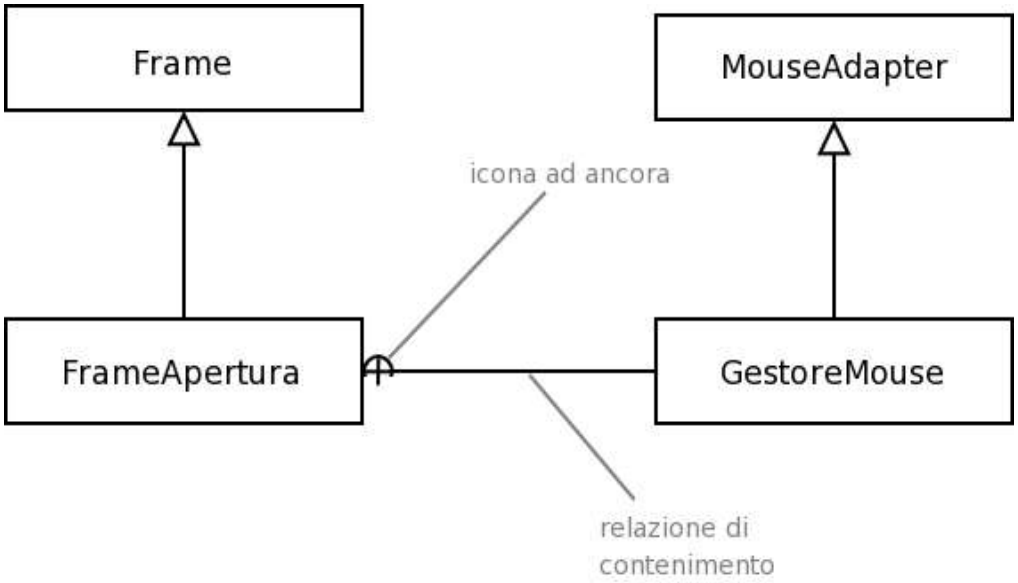
Binding implicito



Contro del binding implicito: la classe istanziata non può veramente avere un suo nome

Classi annidate

- Classe annidata: classe definita all'interno della definizione di un'altra classe
- È accessibile solo alla classe esterna e agli oggetti di quella classe



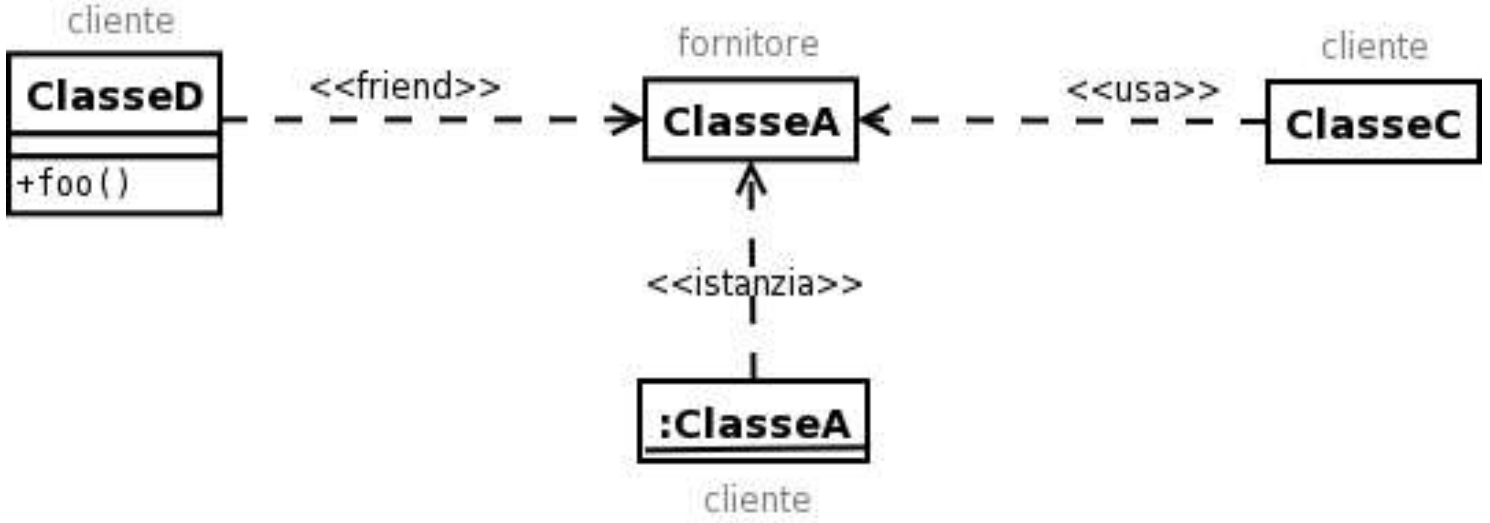
Cos'è una dipendenza?

Una dipendenza è una relazione tra due elementi, dove un cambiamento a uno di essi (il fornitore) può influenzare l'altro (il cliente)

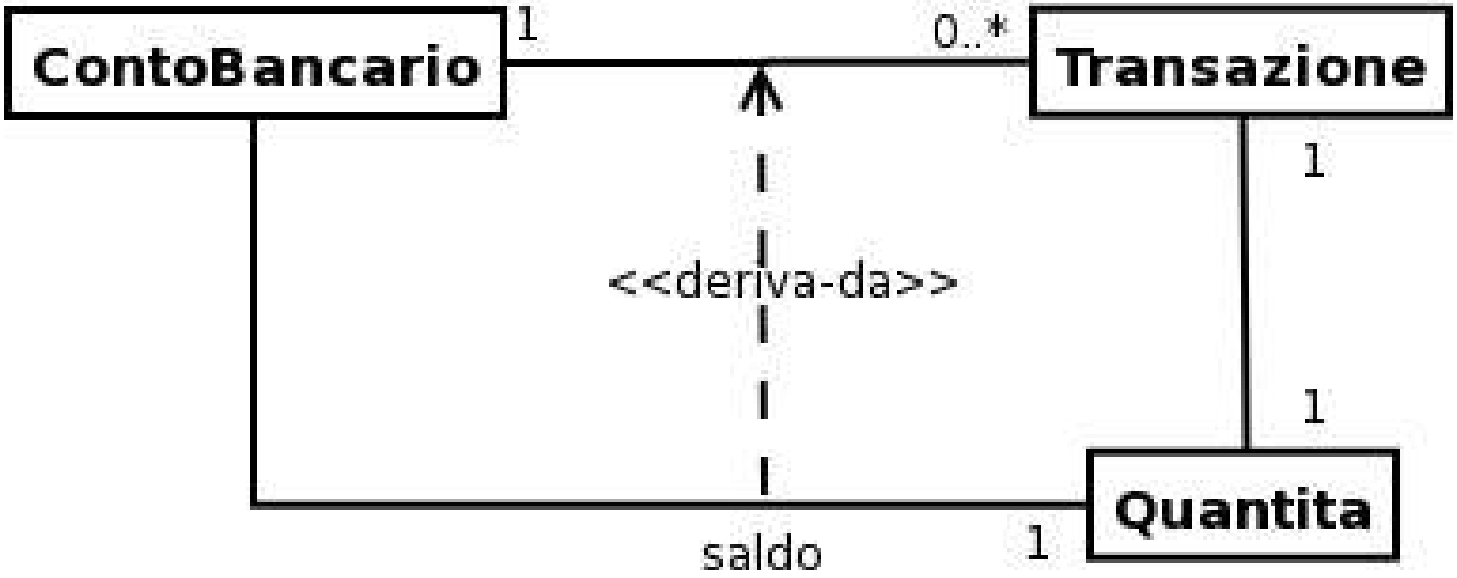
- Uso: *il cliente utilizza alcuni dei servizi resi disponibili dal fornitore per implementare il proprio comportamento*
- Astrazione: *una relazione tra cliente e fornitore in cui il fornitore è più astratto dle cliente*
- Accesso: *il fornitore assegna al cliente un qualche tipo di permesso di accesso al proprio contenuto*
- Binding: *il fornitore assegna al cliente dei parametri che saranno istanziati a valori specifici dal cliente*

Cos'è una dipendenza?

- Uso: <<usa>>, <<chiama>>, <<parametro>>, <<invia>>, <<istanzia>>
- Astrazione: <<origine>>, <<rifinisce>>, <<deriva-da>>
- Accesso: <<accede>>, <<importa>>, <<friend>>
- Binding: <<bind>>



Esempio di astrazione



Questo modello evidenzia che saldo deriva dalla collezione delle **Transazione** di **ContoBancario**

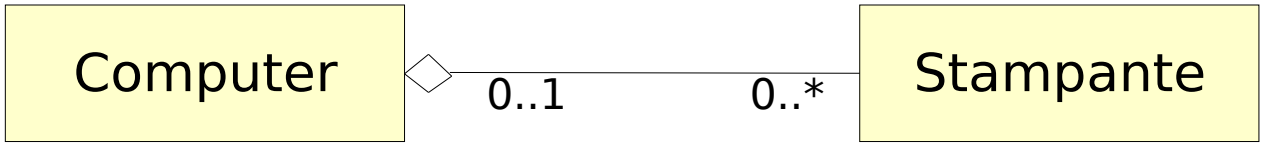
Aggregazione e composizione

Aggregazione e *composizione* sono speciali forme di associazione che specificano una relazione *whole-part (tutto-parte)* tra l'aggregato e le parti componenti.

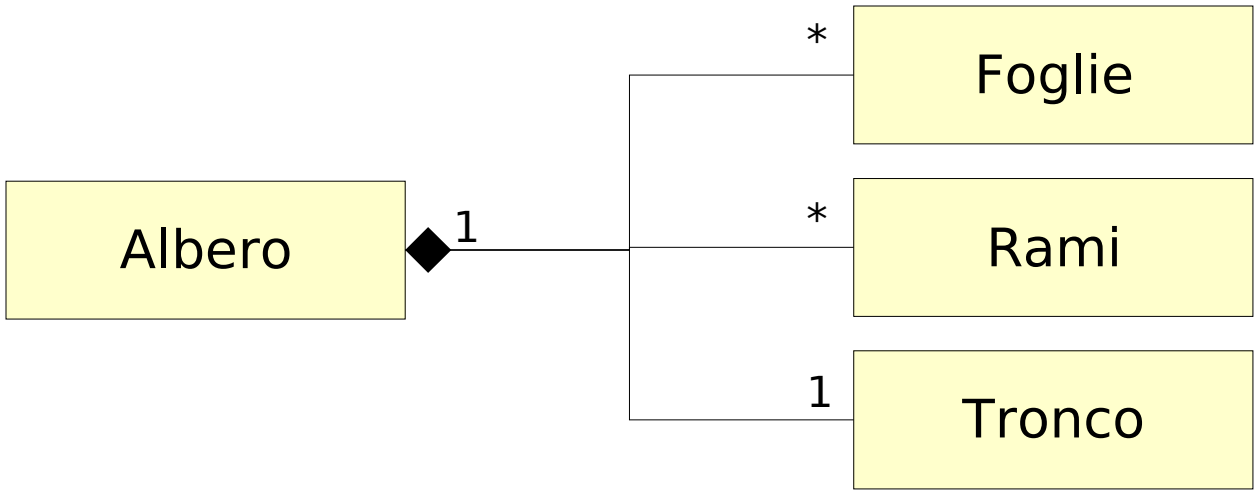
- Aggregazione: relazione poco forte (le parti esistono anche senza il tutto)

- Composizione: relazione molto forte (le parti dipendono dal tutto, per esempio i muri e la stanza)

Aggregazione e composizione



Aggregazione

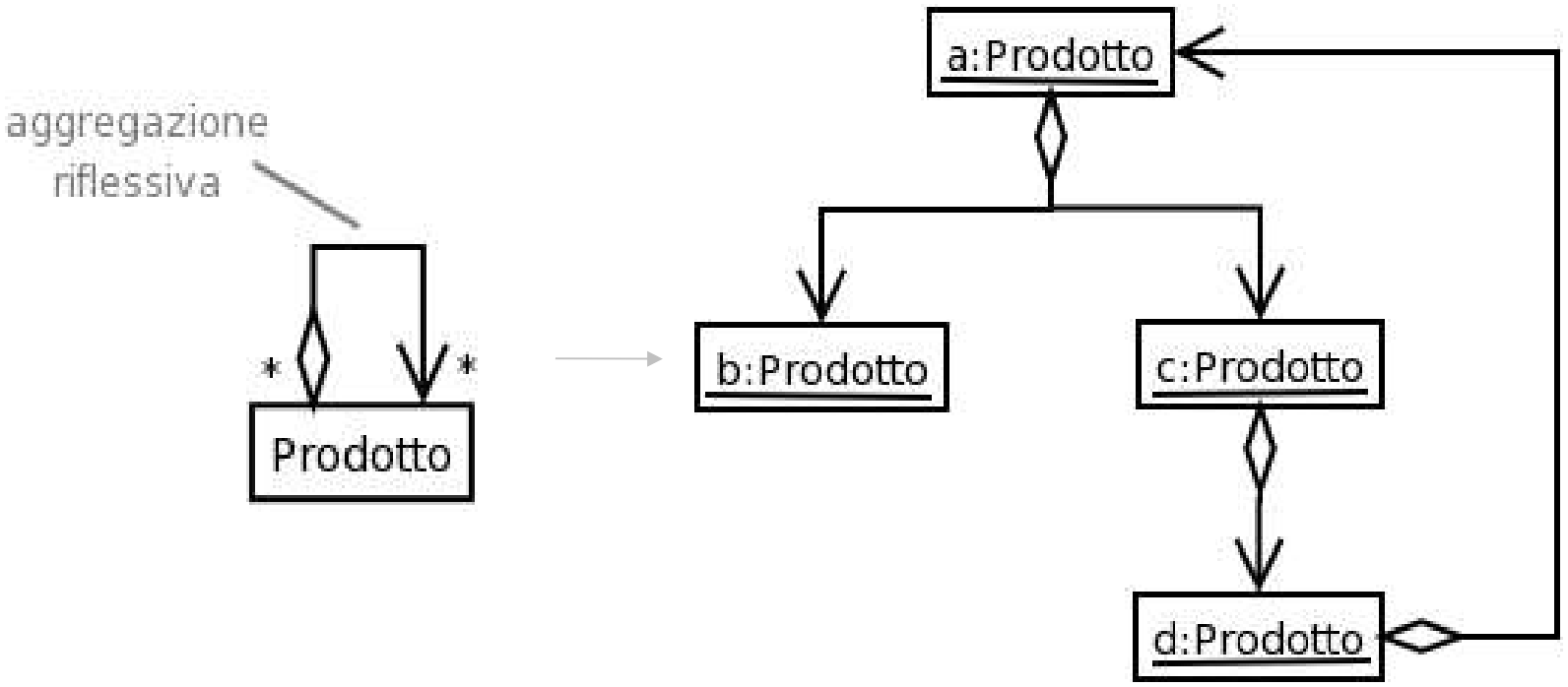


Composizione

Semantica dell'aggregazione

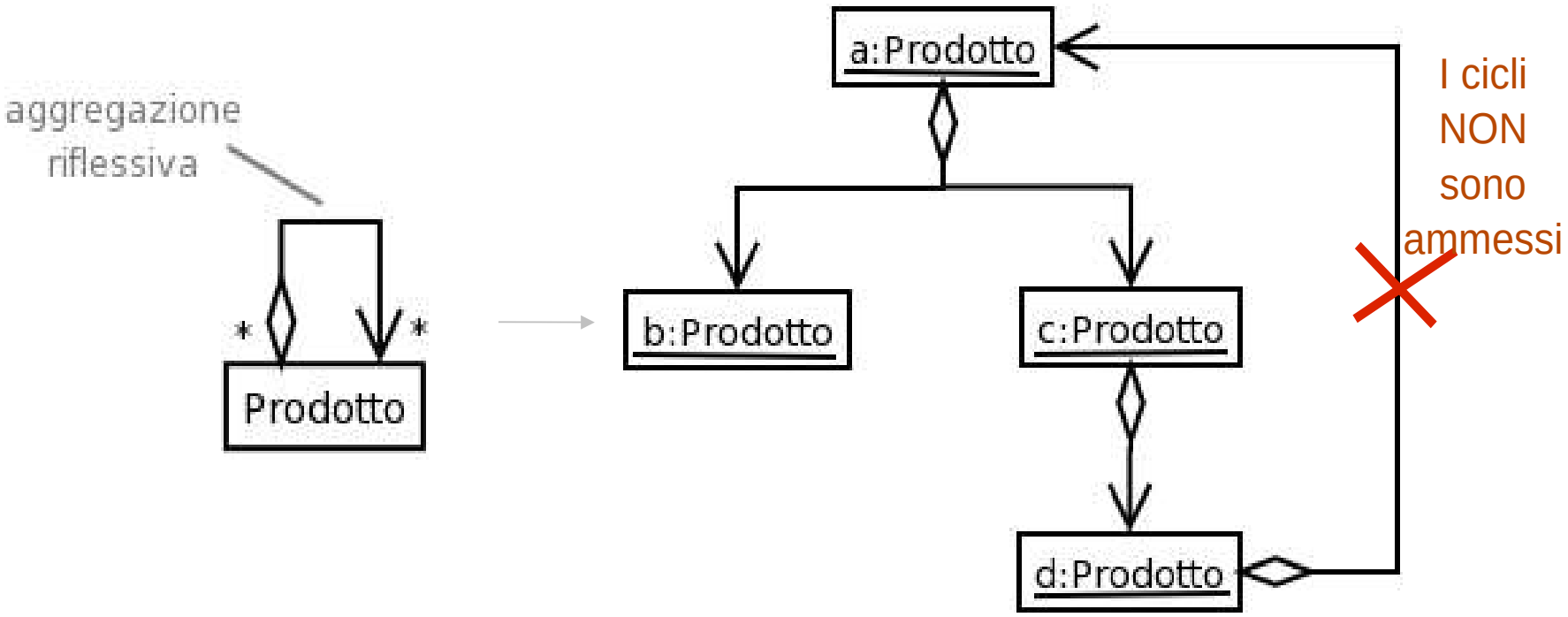
- L'aggregato può in alcuni casi esistere indipendentemente dalle parti, ma in altri casi no.
- Le parti possono esistere indipendentemente dall'aggregato
- L'aggregato è in qualche modo incompleto se mancano alcune delle sue parti
- È possibile che più aggregati condividano una stessa parte
- L'aggregazione è transitiva
- L'aggregazione è asimmetrica

Esempio di aggregazione



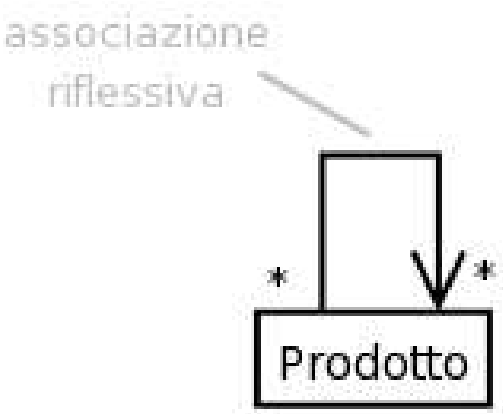
L'istanza del diagrammi delle classi nel diagramma degli oggetti è corretta?

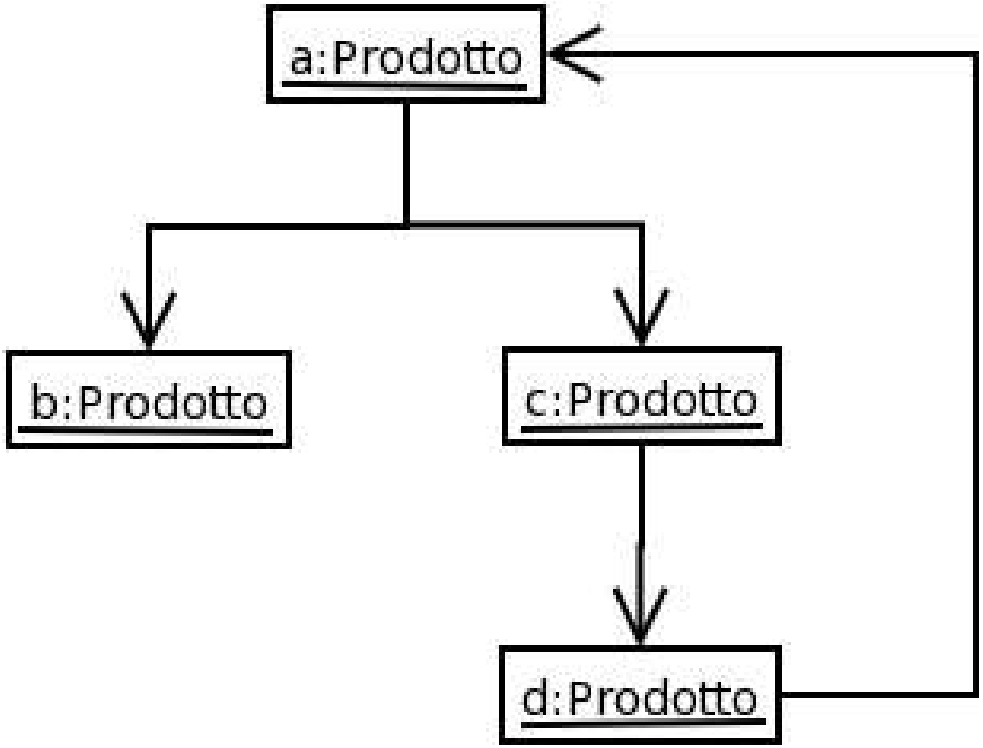
Esempio di aggregazione



- Il diagramma degli oggetti precedente è sbagliato, poiché i cicli non sono ammessi.
- Come si può modellare questa situazione nel diagramma delle classi?

Soluzione


associazione
riflessiva
*
*
Prodotto

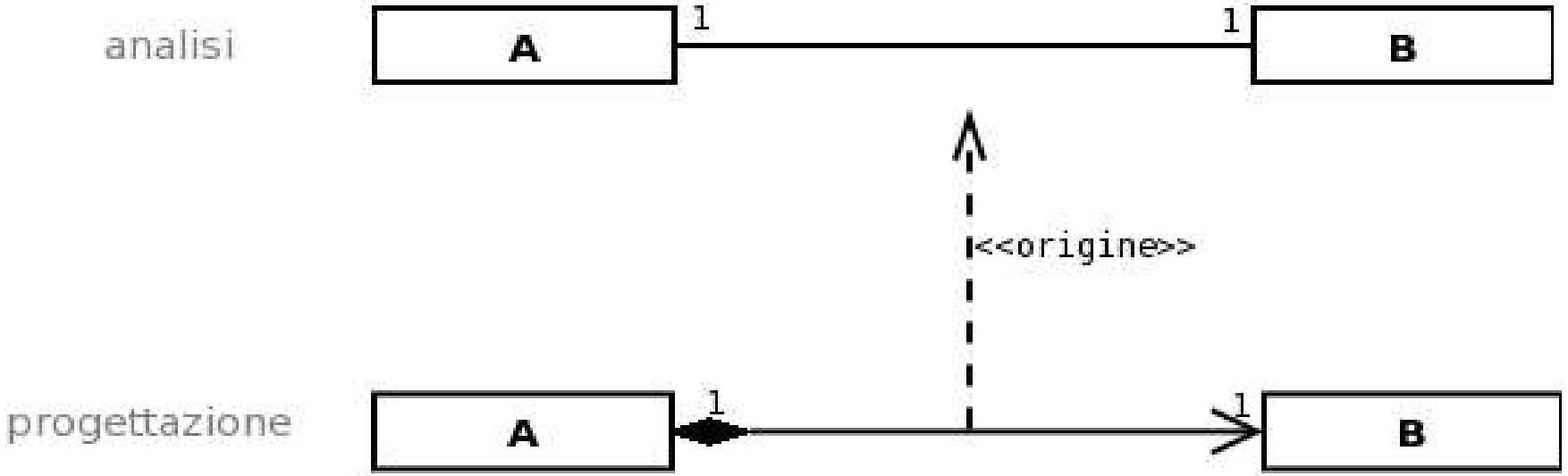

a:Prodotto
b:Prodotto
c:Prodotto
d:Prodotto

Semantica della composizione

- Ogni parte può appartenere ad un solo composito per volta
- Il composito è l'unico responsabile di tutte le sue parti: questo vuol dire che è responsabile della loro creazione e distruzione
- Il composito può rilasciare una sua parte, a patto che un altro oggetto si prenda la relativa responsabilità
- Se il composito viene distrutto, deve distruggere tutte le sue parti o cederne la responsabilità a qualche altro oggetto

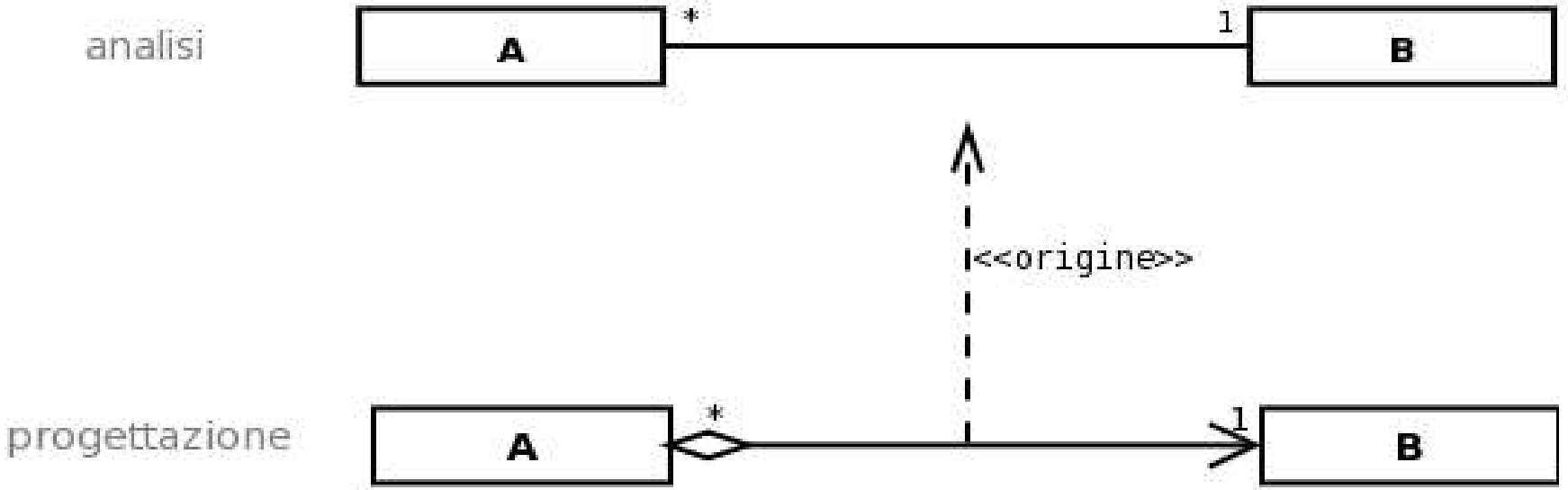
Reificare associazioni uno-a-uno

Ad esempio se **A** è il tutto e **B** una parte:



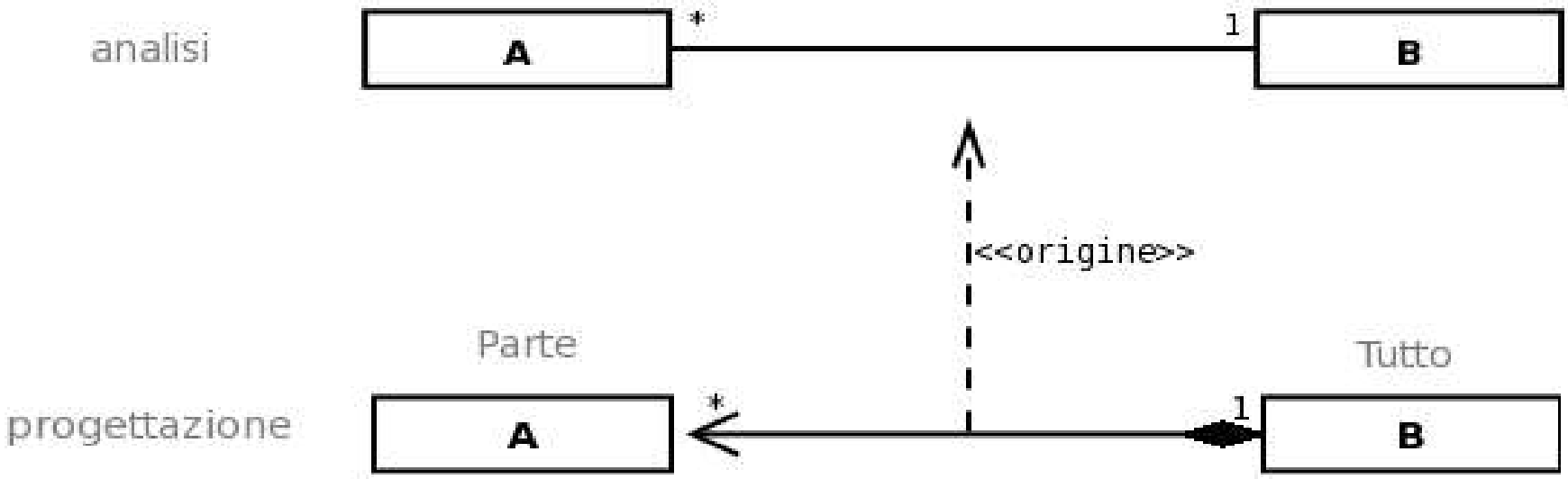
Reificare associazioni molti-a-uno

Ad esempio se **A** è il tutto e **B** una parte:



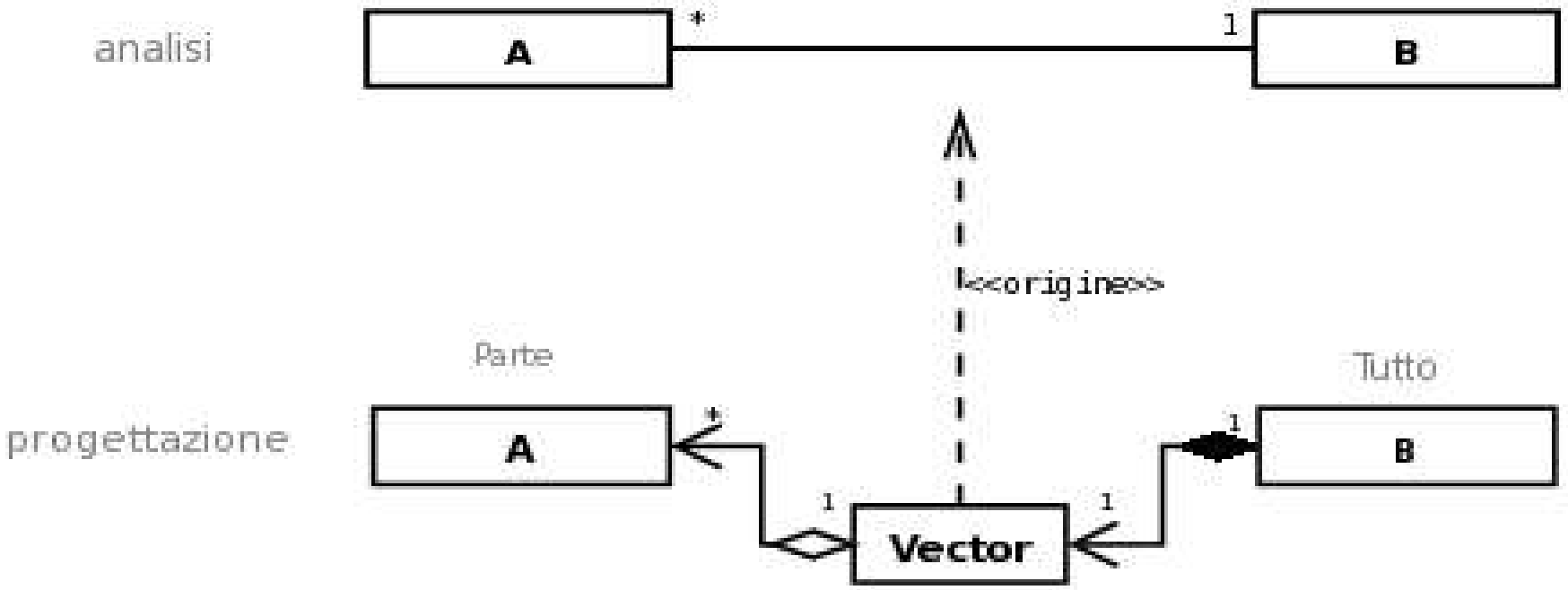
Reificare associazioni molti-a-uno

Ad esempio se **A** è la parte e **B** il tutto:

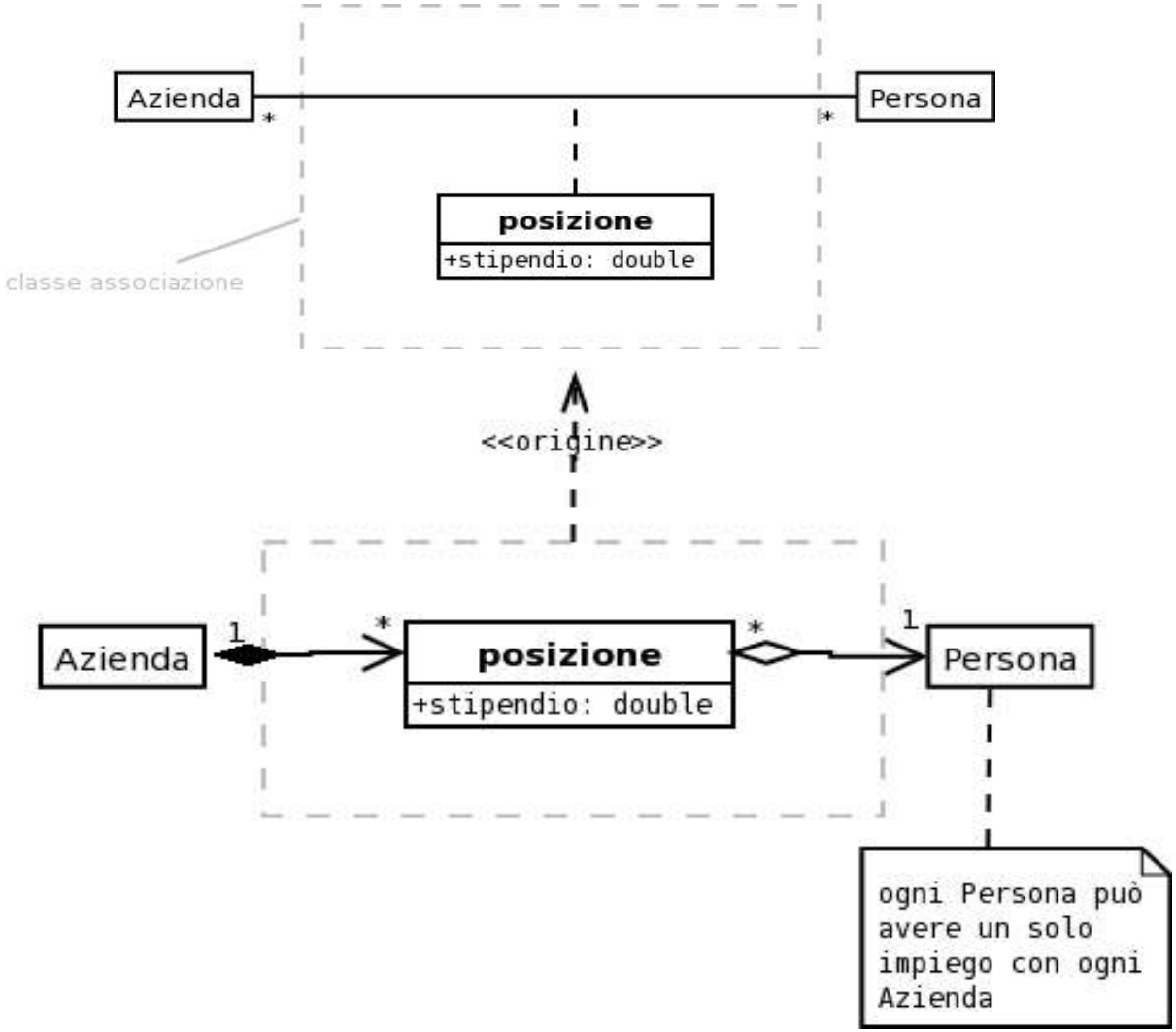


Classi contenitore

Ad esempio se **A** è la parte e **B** il tutto:



Classi associazione



Entità: le interfacce

- Insieme di operazioni che una classe offre ad altre classi

- È rappresentata come una classe con lo stereotipo

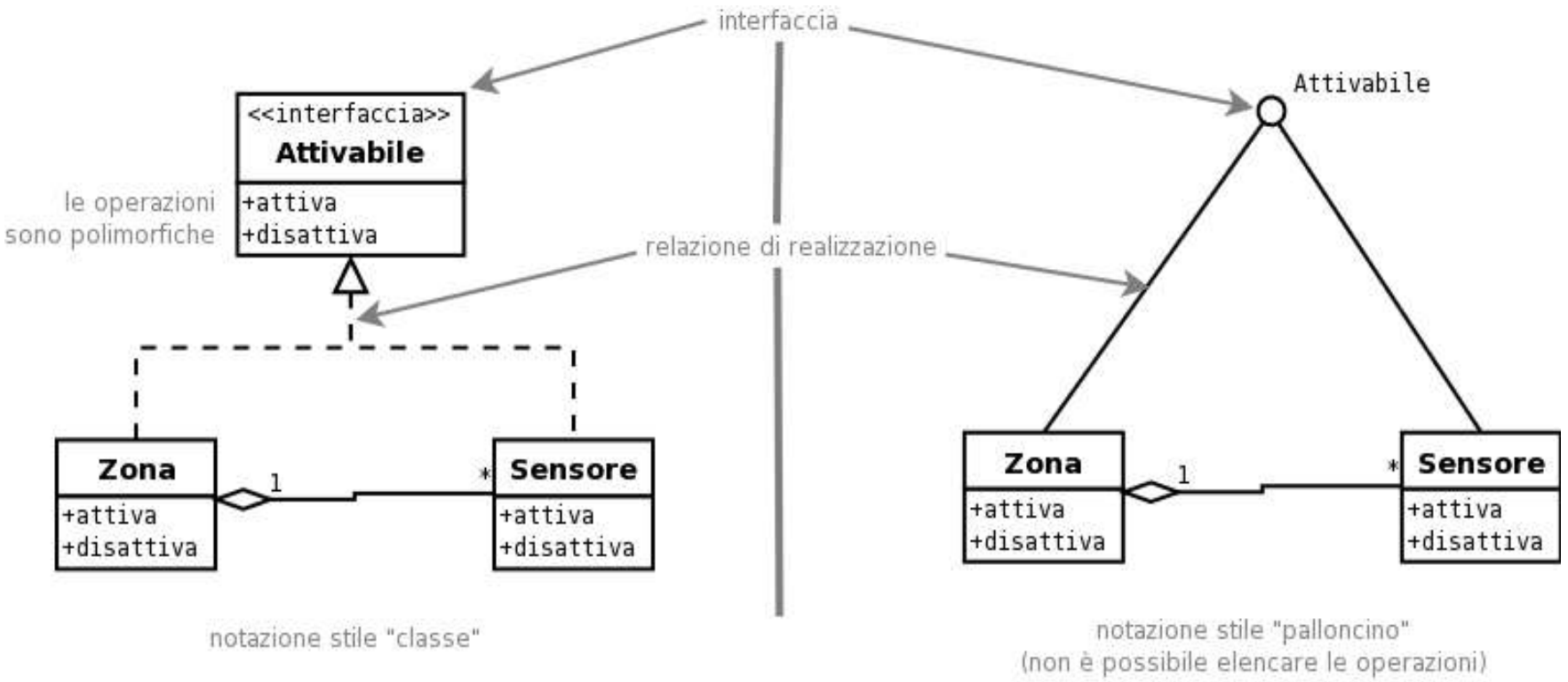
  <<interface>>

- Non ha attributi ma soltanto metodi dichiarati.

- L'implementazione dell'interfaccia può essere indicata con un piccolo cerchio vuoto (o con una relazione di realizzazione)

- Utile per raggruppare operazioni comuni a più classi, quando le classi non sono tutte correlate ad una particolare classe padre.

Entità: le interfacce



Entità: le interfacce

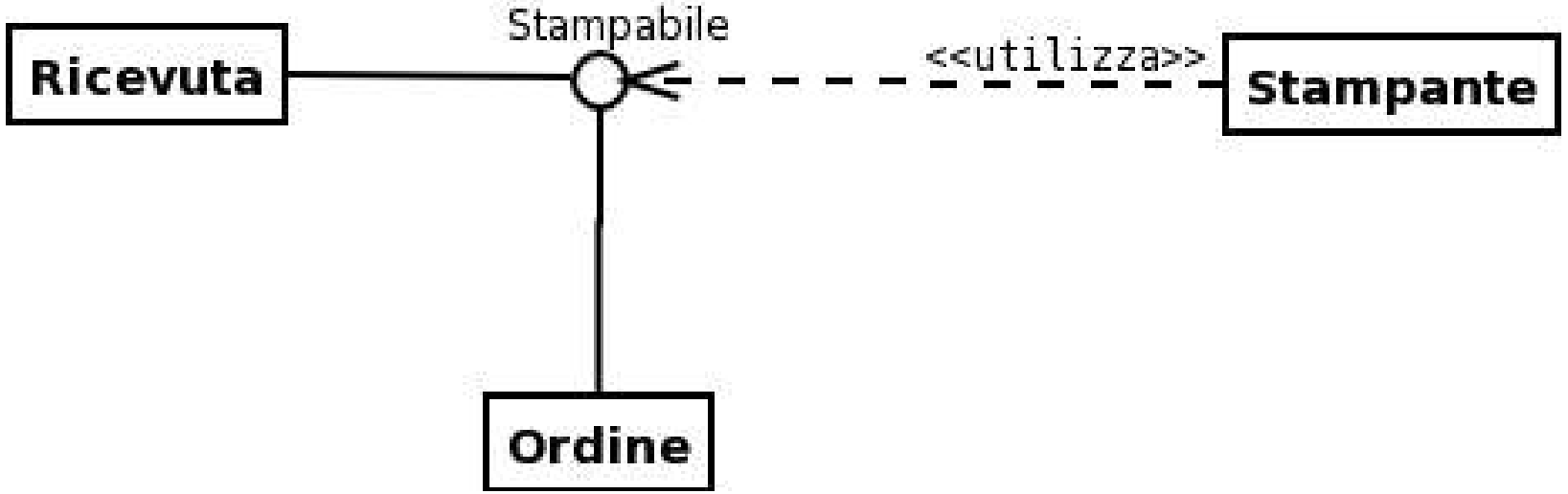
- L'idea è quella di separare le specifiche di una funzionalità (l'interfaccia) dall'implementazione della stessa da parte di un classificatore, quale una classe.
- L'interfaccia definisce un “contratto” che viene implementato dal classificatore
- Progettare per implementazione: progettazione in cui si definiscono le classi e le si connettono tra loro
- Progettare per contratto: definire le interfacce e connetterle alle classi che le realizzano

Entità: le interfacce

- Ogni operazione dell'interfaccia deve avere:
  - La segnatura completa
  - La semantica dell'operazione
  - Ozionalmente uno stereotipo, vincoli e valori etichettati
  - Le interfacce non possono avere:
  - Attributi
  - Implementazione delle operazioni
  - Relazioni navigabili dall'interfaccia a nessun altro tipo di classificatore
  - Un'interfaccia è simile ad una classe astratta senza attributi e con un insieme di operazioni completamente astratte

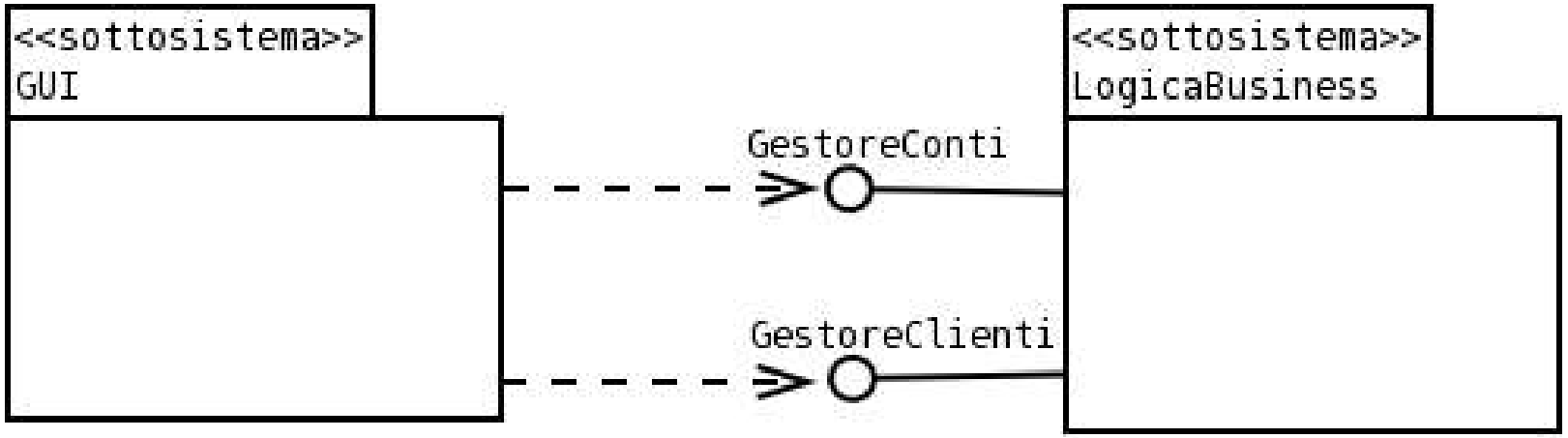
Interfacce e interdipendenze

- Le interfacce consentono di collegare tra loro classi senza introdurre interdipendenze
- Ad esempio: la classe **Stampante** è in grado di stampare qualunque cosa implementi l'interfaccia **Stampabile**



Interfacce e sottosistemi

- Le interfacce sono efficienti e utili nella modellazione dei sottosistemi
- Ad esempio: il sottosistema **GUI** conosce esclusivamente le interfacce **GestoreClienti** e **GestoreConti** e non sa nulla di specifico dell'altro sottosistema



Interfacce: vantaggi e suggerimenti

- La progettazione per interfacce svincola il modello da dipendenze dell'implementazione e ne aumenta la flessibilità e la estendibilità
- Tiene sotto controllo le interdipendenze presenti nel modello
- Separa il modello in sottoinsiemi coesi

però

- La corretta modellazione di un sistema è più importante della sua modellazione flessibile
- Bisogna utilizzare interfacce semplici e utili

Riferimenti

- [UML 1.5] *OMG UML Specification* v. 1.5.
- [AN02] Arlow, Neustadt, *UML e Unified Process*, McGraw-Hill, 2002
- [BSL02] Bennett, Skelton, Lunn, *Introduzione a UML*, McGraw-Hill collana Schaum's, 2002

- I diagrammi sono stati realizzati con *Dia* http://www.gnome.org/projects/dia/