Universita’ degli Studi di Perugia
Dipartimento di Matematica e Informatica

Corso di Laurea in Informatica

Ingegneria del Software

UML - Unified Modeling Language
Diagrammi di Classe

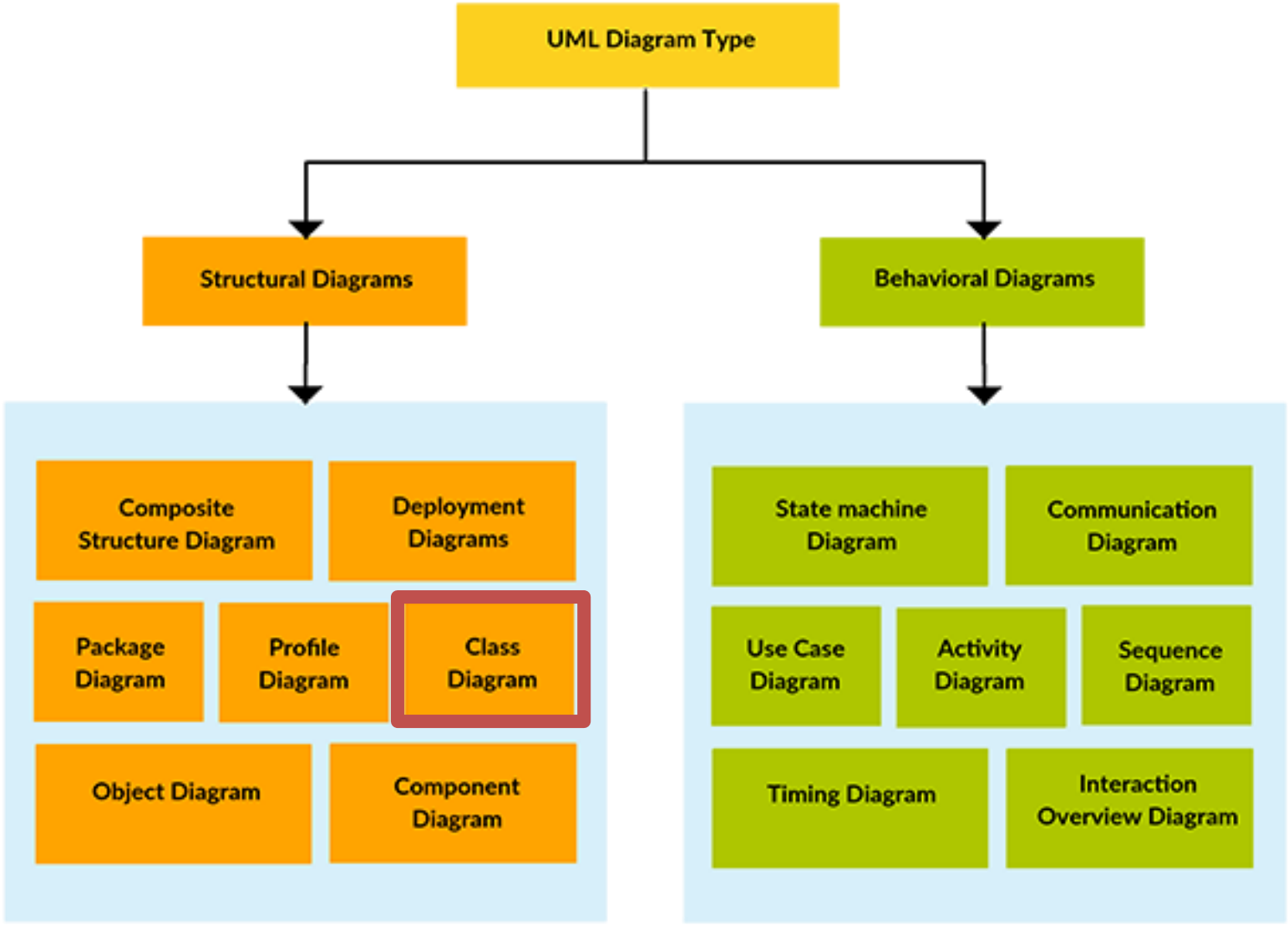
Prof. Alfredo Milani

Materiale note, grazie al contributo di: Alfredo Milani, Fabrizio Montecchiani, Carlo Ghezzi, Alessandro Riccardi, Paolo Mengoni et al.

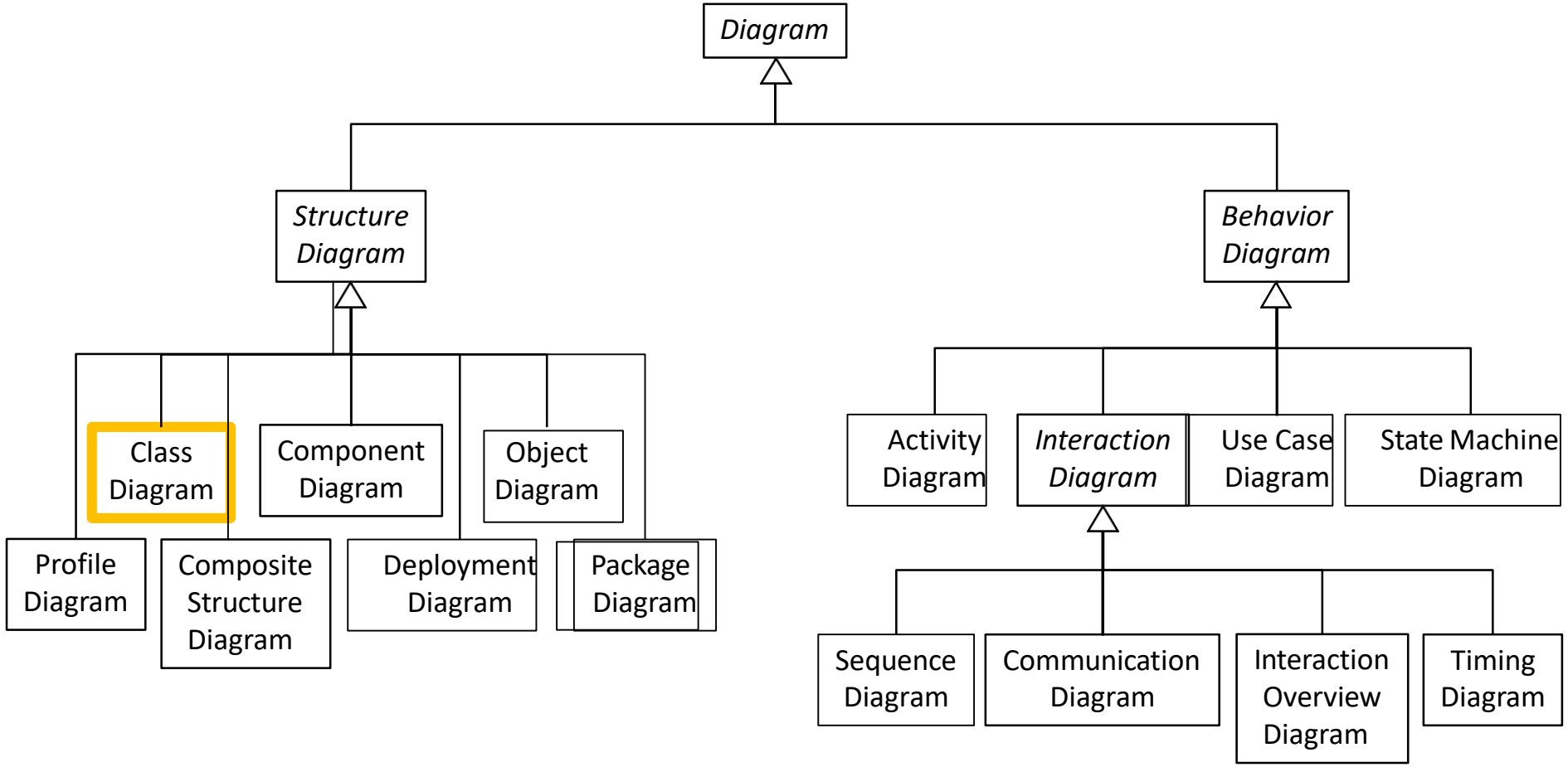
SOMMARIO

- Introduzione
- Proprietà e Operazioni
- Concetti base e avanzati
- Diagrammi degli oggetti

DIAGRAMMA DELLE CLASSI



UML – gerarchia dei diagrammi

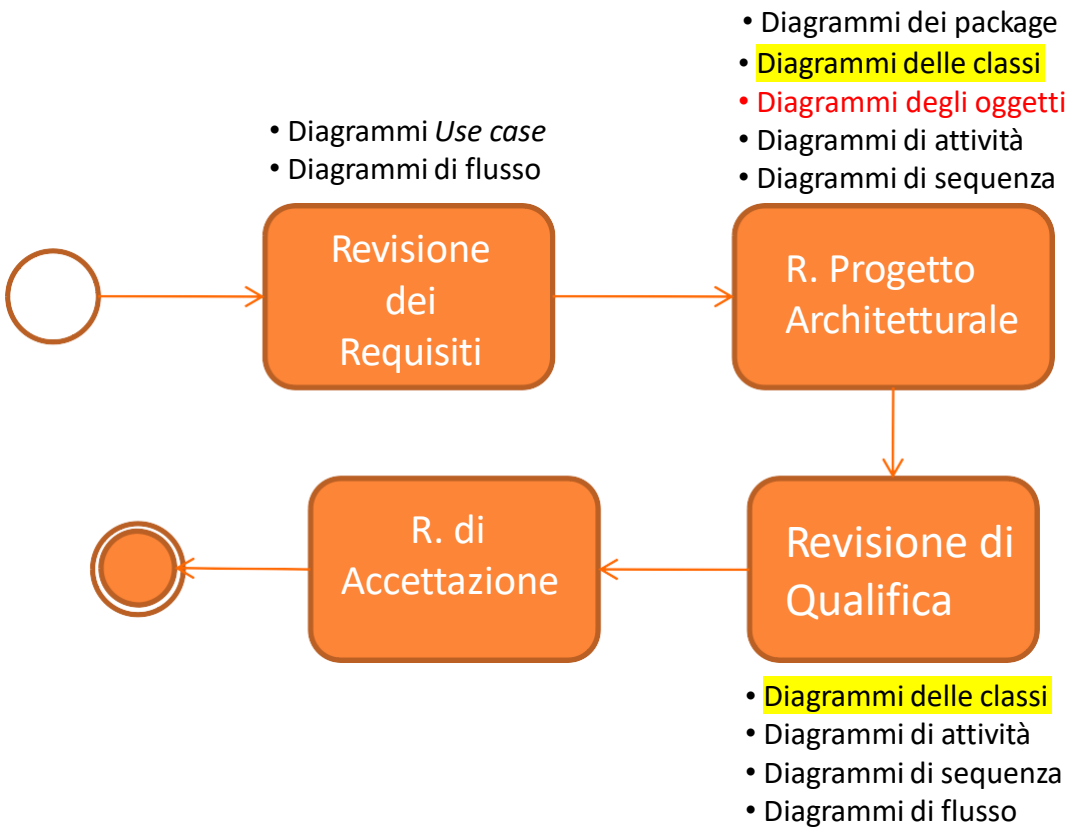


Notation: UML

testo in *corsivo*: categoria astratta di diagrammi
testo normale: diagramma concreto

DIAGRAMMA DELLE CLASSI

- Ogni fase, i suoi diagrammi

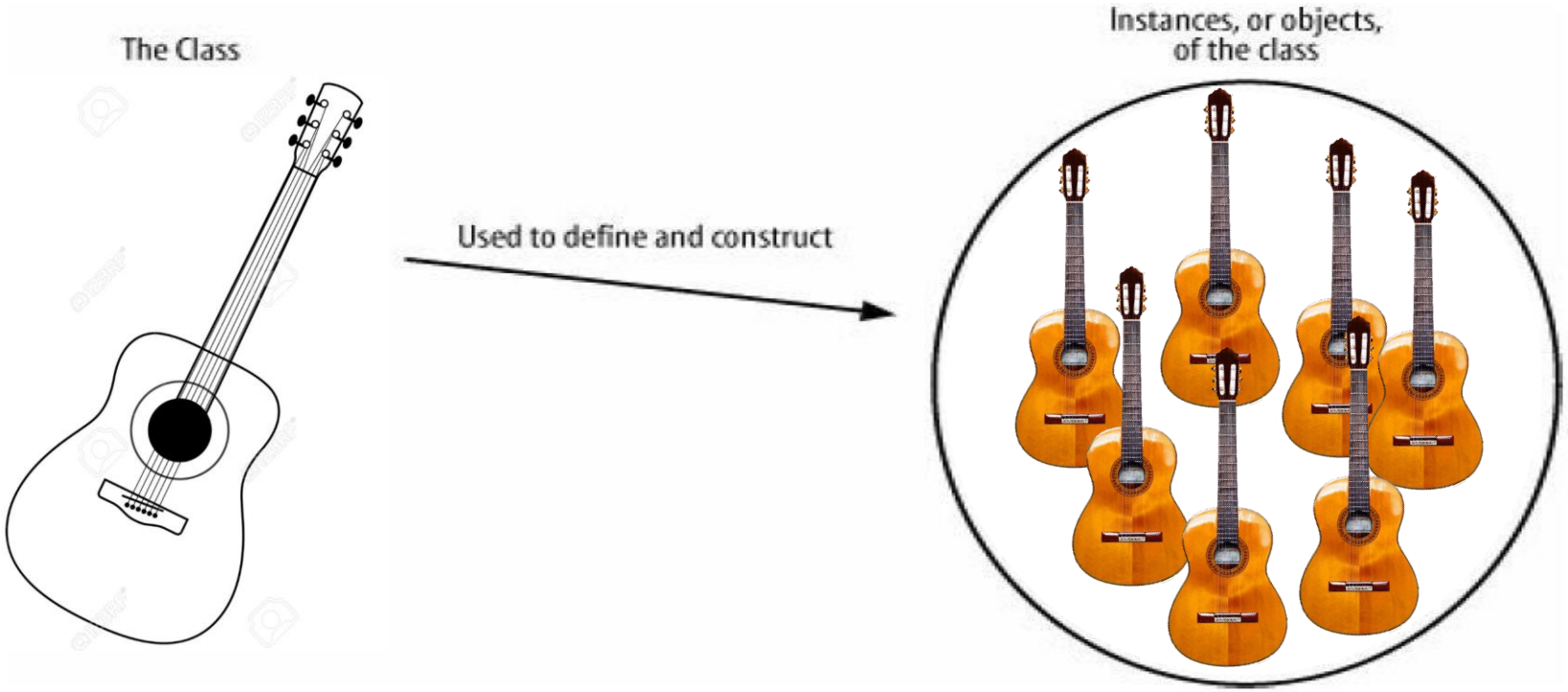


Diagrammi di classe

- I **diagrammi di classe** mostrano le diverse classi che costituiscono un sistema e come si relazionano una all'altra.
- Sono diagrammi statici:
  - mostrano le classi, insieme ai loro metodi e attributi oltre alle relazioni statiche tra loro,
  - non mostrano le chiamate ai metodi tra di loro.

Classi

- Una **classe** è il descrittore di un insieme di **oggetti** o (**istanze della classe**) che condividono gli stessi attributi, operazioni, metodi, relazioni e comportamento.



ESEMPIO

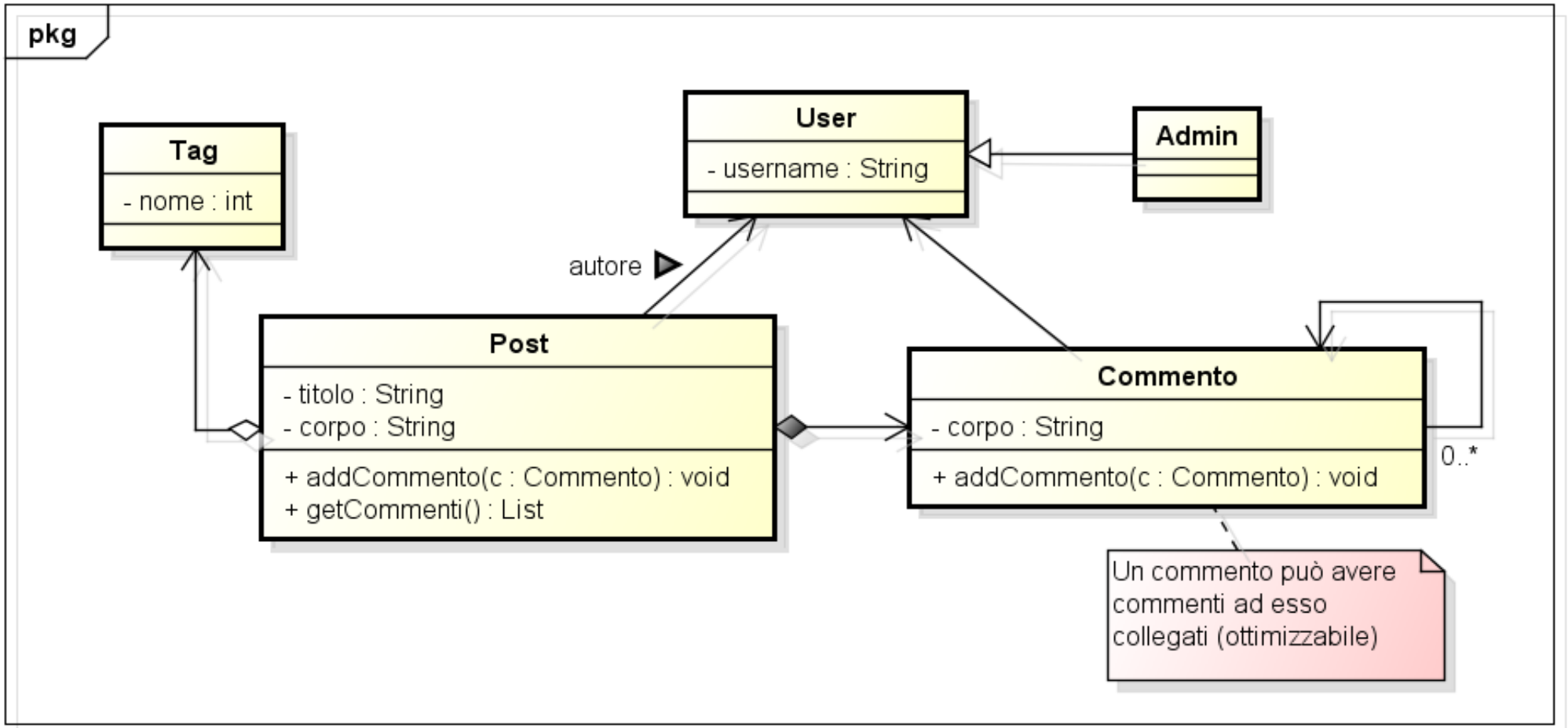
È richiesto lo sviluppo di un'applicazione che permetta la gestione di un semplice blog.

In particolare devono essere disponibili almeno tutte le funzionalità base di un blog: deve essere possibile per un utente inserire un nuovo post e successivamente per gli altri utenti deve essere possibile commentarlo.

Queste due operazioni devono essere disponibili unicamente agli utenti registrati all'interno del sistema.

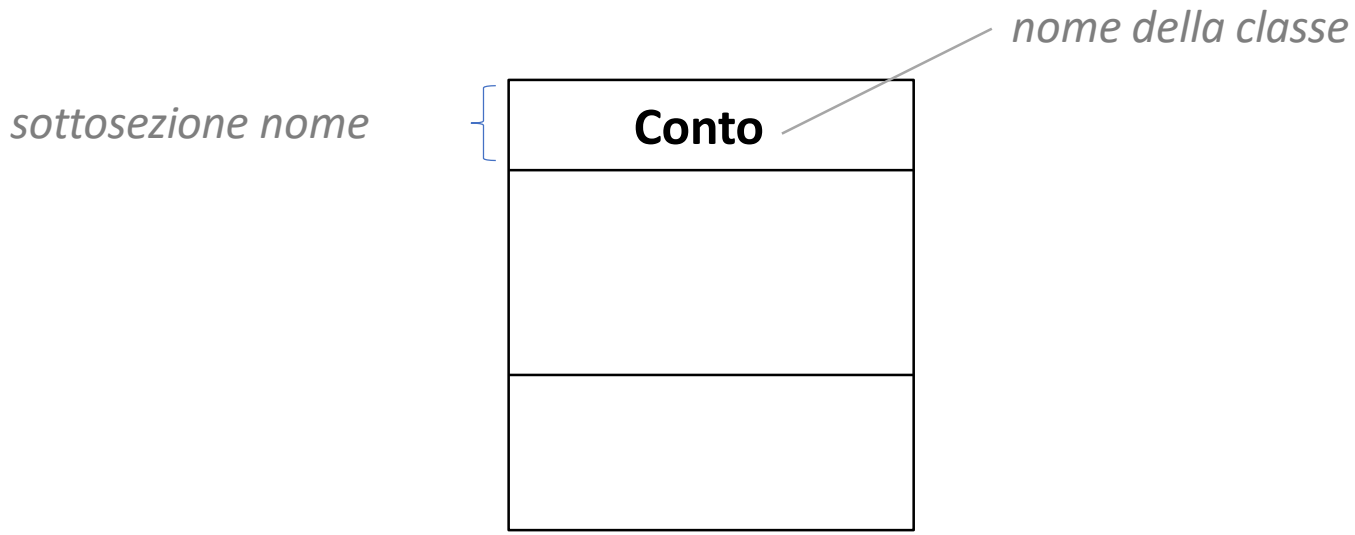
La registrazione avviene scegliendo una username e una password. La username deve essere univoca all'interno del sistema.

ESEMPIO


pkg
Tag
- nome : int
User
- username : String
Admin
autore
Post
- titolo : String
- corpo : String
+ addCommento(c : Commento) : void
+ getCommenti() : List
Commento
- corpo : String
+ addCommento(c : Commento) : void
0..*
Un commento può avere commenti ad esso collegati (ottimizzabile)

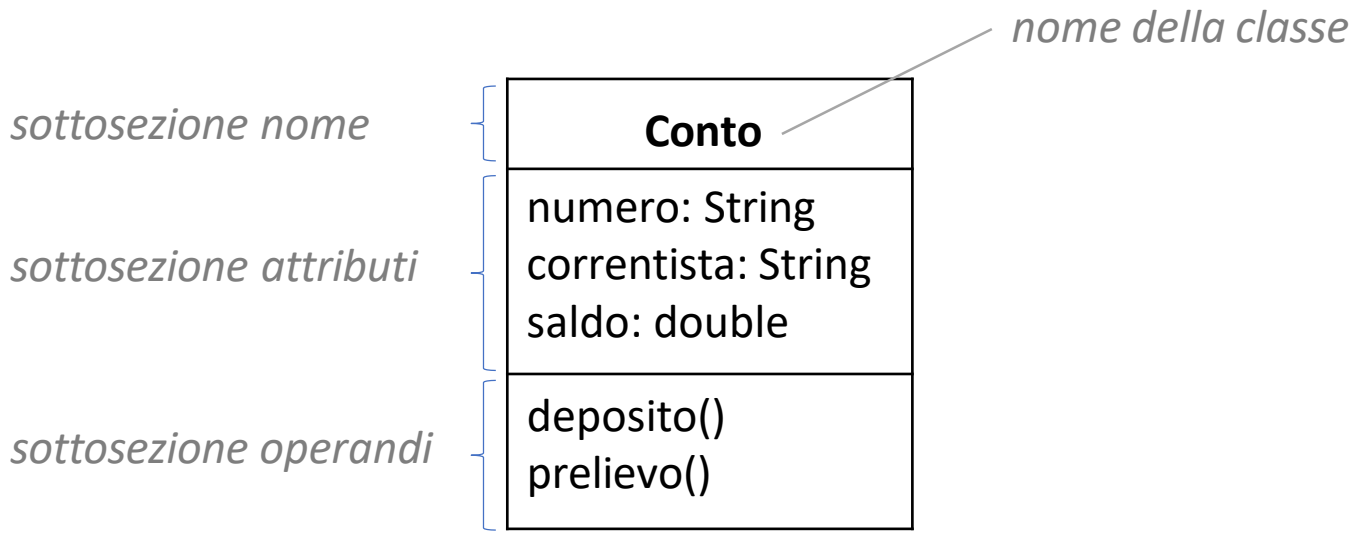
Notazione UML per classi

- Una **classe** viene rappresentata da un rettangolo contenente il suo nome in una sottosezione rettangolare.



Classi

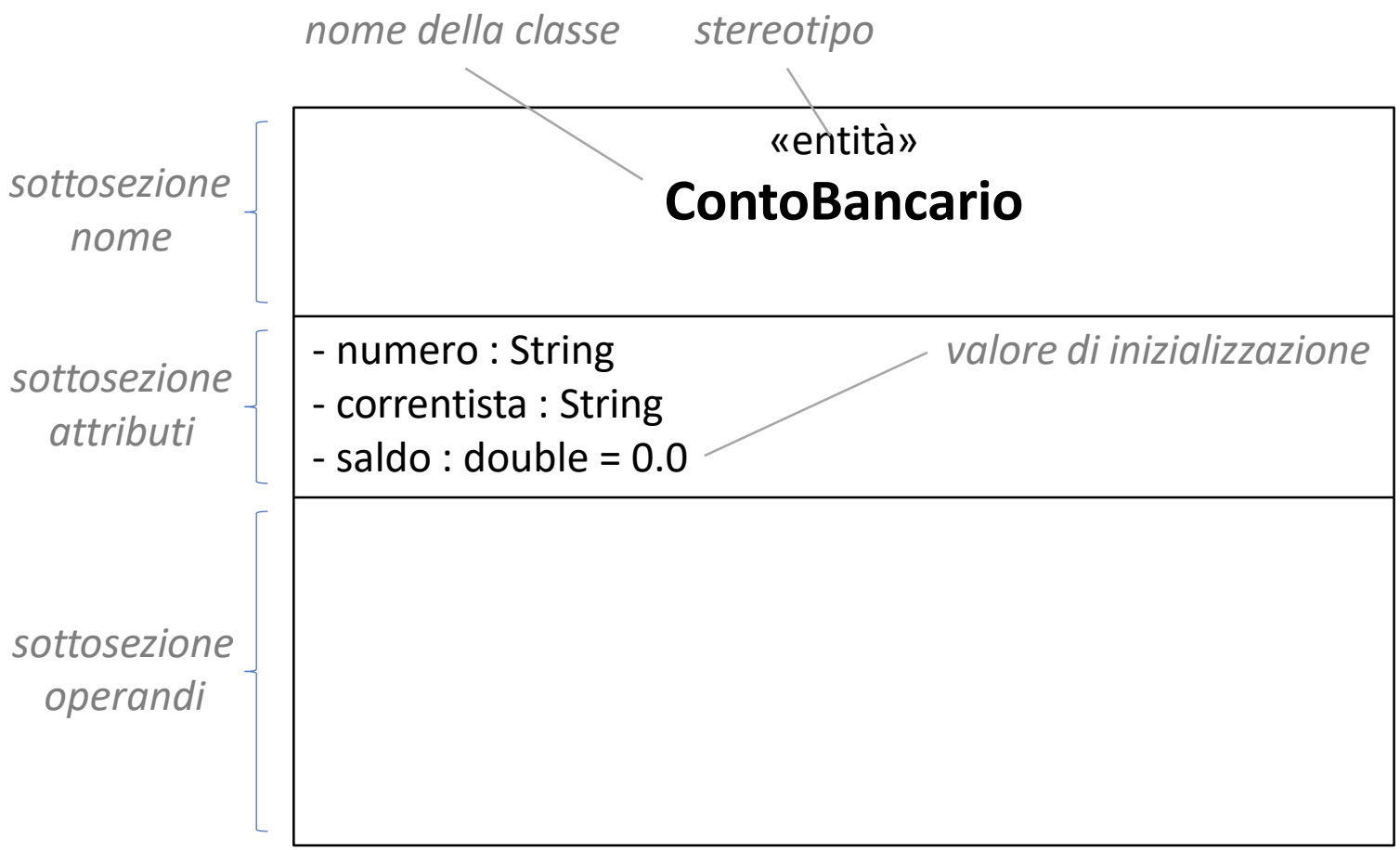
- Il nome è l'unica sottosezione obbligatoria.
- Facoltativamente, si possono mostrare gli **attributi (campi)** e le **operazioni** (**metodi**) della classe in due altre sottosezioni del rettangolo.



Attributi

- Gli **attributi** sono mostrati con almeno il loro nome.
- Si possono includere anche il loro **tipo**, il **valore iniziale** e altre proprietà.
- Ornamento di **visibilità** degli attributi:
  - **+** sta per **pubblici** (**public**)
  - **#** sta per **protetti** (**protected**)
  - **-** sta per **privati** (**private**)
  - **~** sta per **pacchetto** (**package**)

Notazione di una classe



Operazioni (metodi)

- In UML le **operazioni** (**metodi**) sono mostrate con almeno il loro nome.
- Si possono includere anche i loro **parametri** e i **tipi restituiti**.
- Ornamenti di **visibilità** per le operazioni:

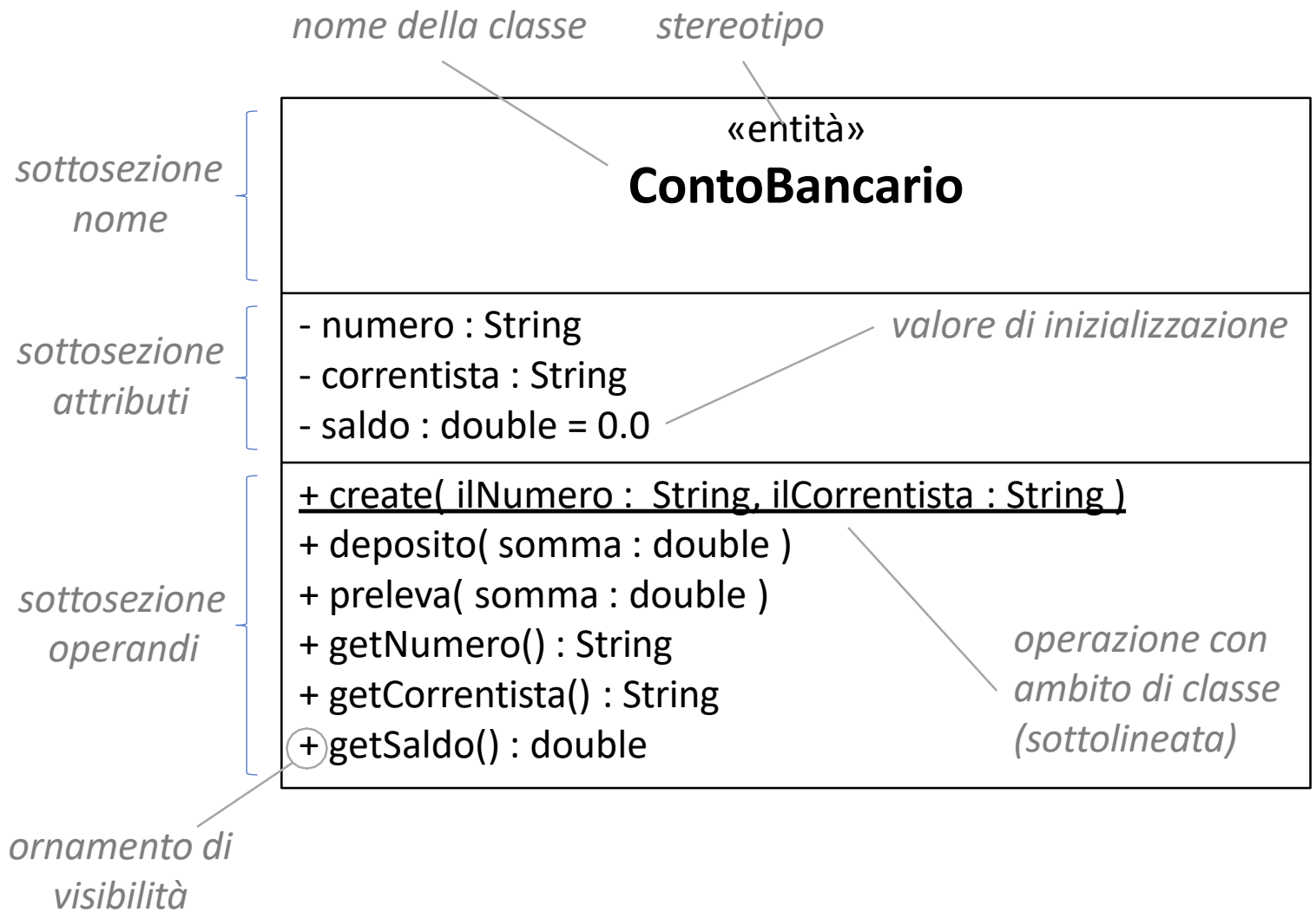
  + sta per **pubblici** (**public**)

  # sta per **protetti** (**protected**)

  - sta per **privati** (**private**)

  ~ sta per **pacchetto** (**package**)

Notazione di una classe



PROPRIETÀ

- Corrispondenza nel linguaggio di programmazione
  - Attributi
    - Membri di classe (privati, se possibile)
    - Proprietà aggiuntive
      - Se *ordered*: *array* o vettori
      - Se *unordered*: insiemi
    - Convenzioni dei gruppi di programmazione
      - Esempio: *getter* e *setter* per ogni attributo
  - Associazioni
    - Anche se etichettata con verbo, meglio renderla con un nome
    - Evitare le associazioni bidirezionali
      - Di chi è la responsabilità di aggiornare la relazione?

OPERAZIONI

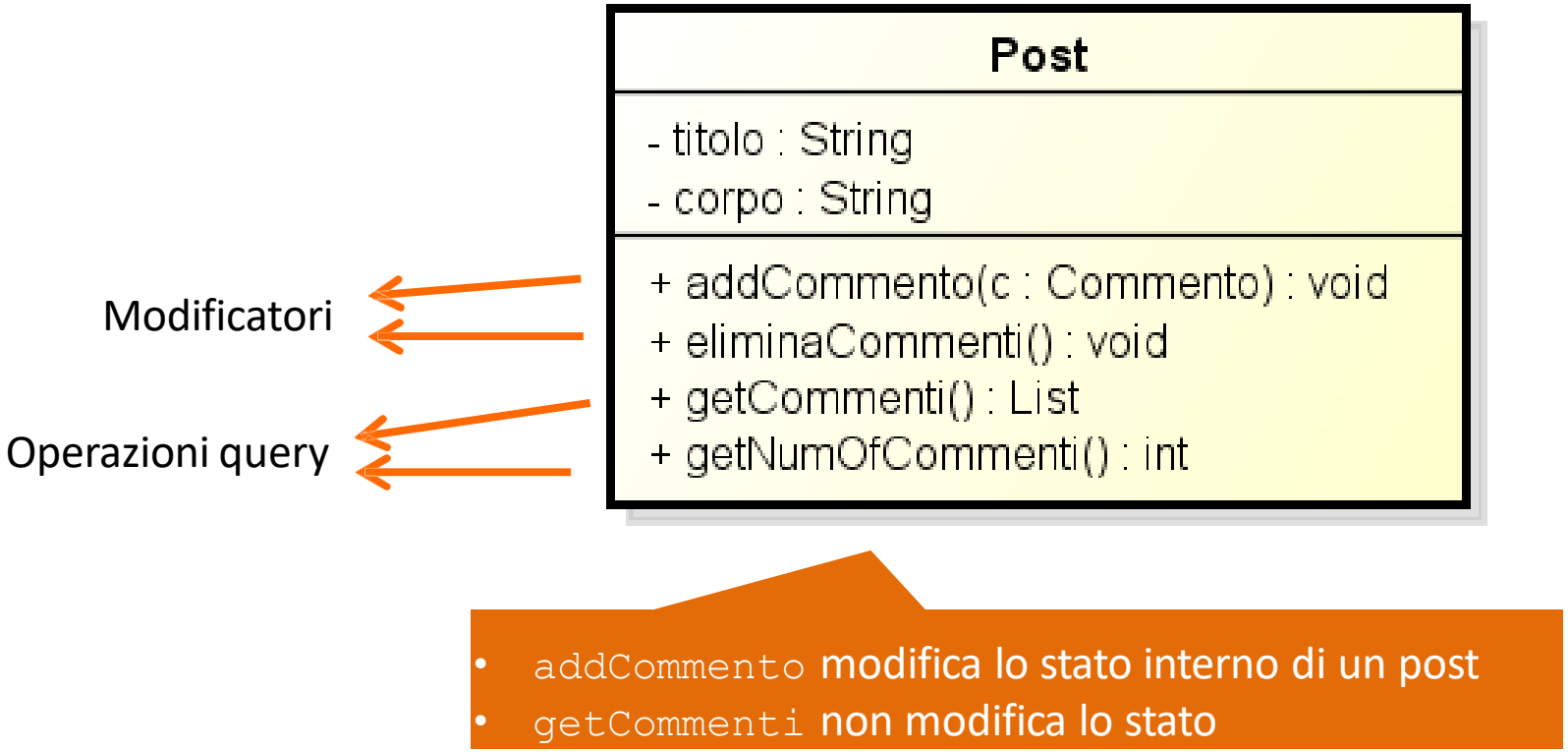
- Definizione

```
Visibilità nome (lista-parametri) : tipo-ritorno {proprietà aggiuntive}

Lista-parametri := direzione nome : tipo = default
```

- Le azioni che la classe “sa eseguire”
  - Descrivono Aspetti comportamentali
  - Offrono un Servizio che può essere richiesto ad ogni istanza della classe
  - Direzione: **in**, **out**, **inout** (*default* **in**)
  - Visibilità: **+** pubblica, **-** privata, **#** protetta
  - *Query*
  - Modificatori
  - Operazione ≠ metodo
    - Concetti differenti in presenza di polimorfismo

ESEMPIO

- Operazioni



- `addCommento` modifica lo stato interno di un post
- `getCommenti` non modifica lo stato

Relazioni tra classi

- Le diverse classi possono relazionarsi una con l'altra in diversi modi:
  - **Associazione**
  - **Aggregazione**
  - **Composizione**
  - **Generalizzazione (ereditarietà)**
  - **Dipendenza**
  - **Realizzazione**

permettono di descrivere relazioni logiche tra le classi nel dominio applicativo

Associazione

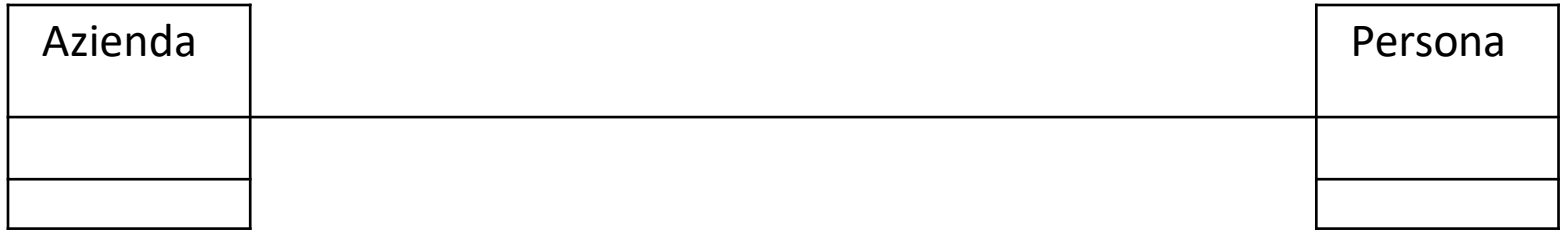
- Un **collegamento tra due oggetti** è una connessione semantica che consente loro di scambiarsi messaggi.
- Un'**associazione tra due classi** indica che si può avere un collegamento tra una coppia di oggetti appartenenti alle due classi.
- L'**aggregazione** e la **composizione** possono considerarsi delle forme di associazioni speciali.

Associazione

- Le associazioni possono essere indicate con
  - **Nome dell'associazione**: un verbo che specifica l'azione che l'oggetto origine esegue sull'oggetto destinazione.
  - **Nome dei ruoli**: un sostantivo che descrive il ruolo che gli oggetti possono ricoprire.
  - **Navigabilità**: specifica se la comunicazione tra gli oggetti può essere uni- o bidirezionale.
  - **Molteplicità:** specifica per ciascun lato dell'associazione, quanti oggetti su questo lato possono relazionarsi a un oggetto sull'altro lato.

Notazione UML per associazioni

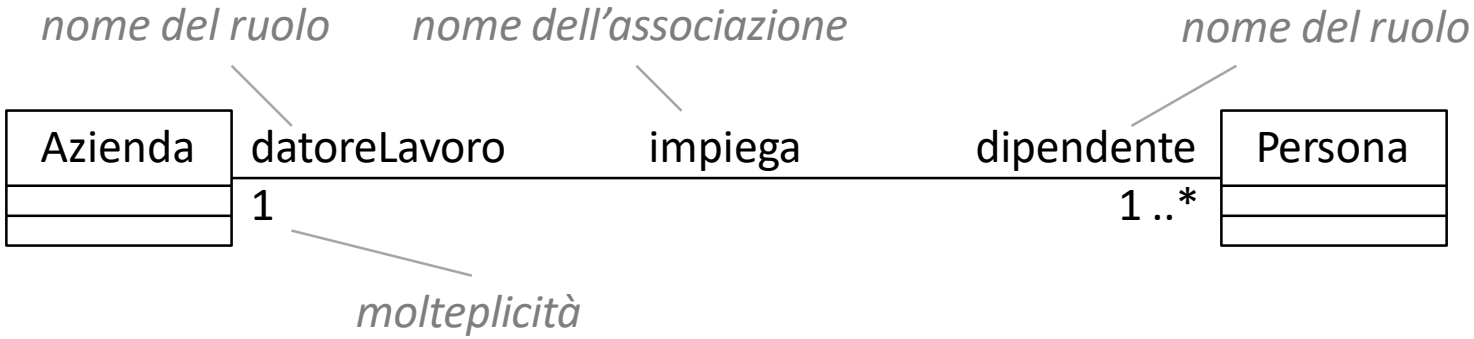
- Un'**associazione** è rappresentata con una linea che connette le due classi che partecipano alla relazione.



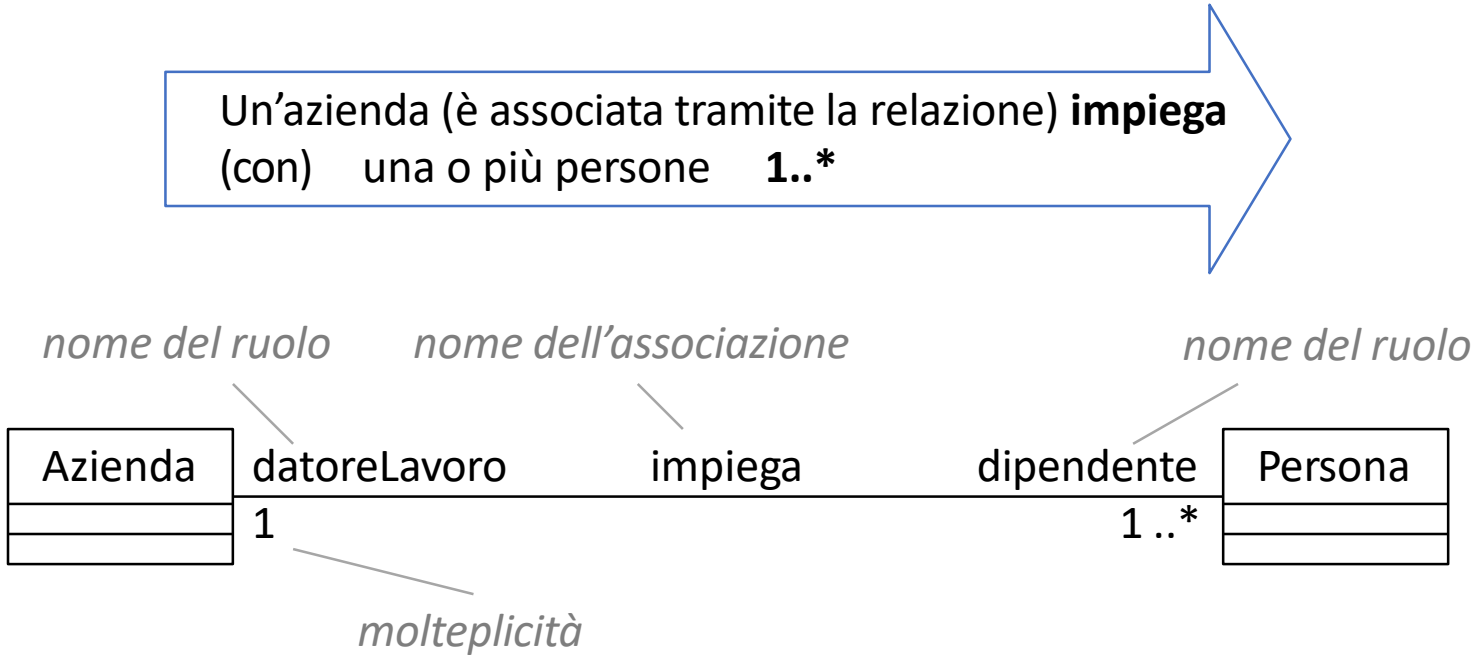
Notazione UML per associazioni

- In **nome dell'associazione**, i **nomi dei ruoli** e la **molteplicità** (detta anche **cardinalità**) possono essere inseriti nelle vicinanze della linea che descrive l'associazione e dal lato delle classi cui si riferiscono.

In una descrizione semplificata possono tutti essere omessi



Molteplicità di un’associazione



Moltepicità indicata come

- **m..n**

m cardinalità minima
n cardinalità massima

- **m..m** abbreviato con **m**

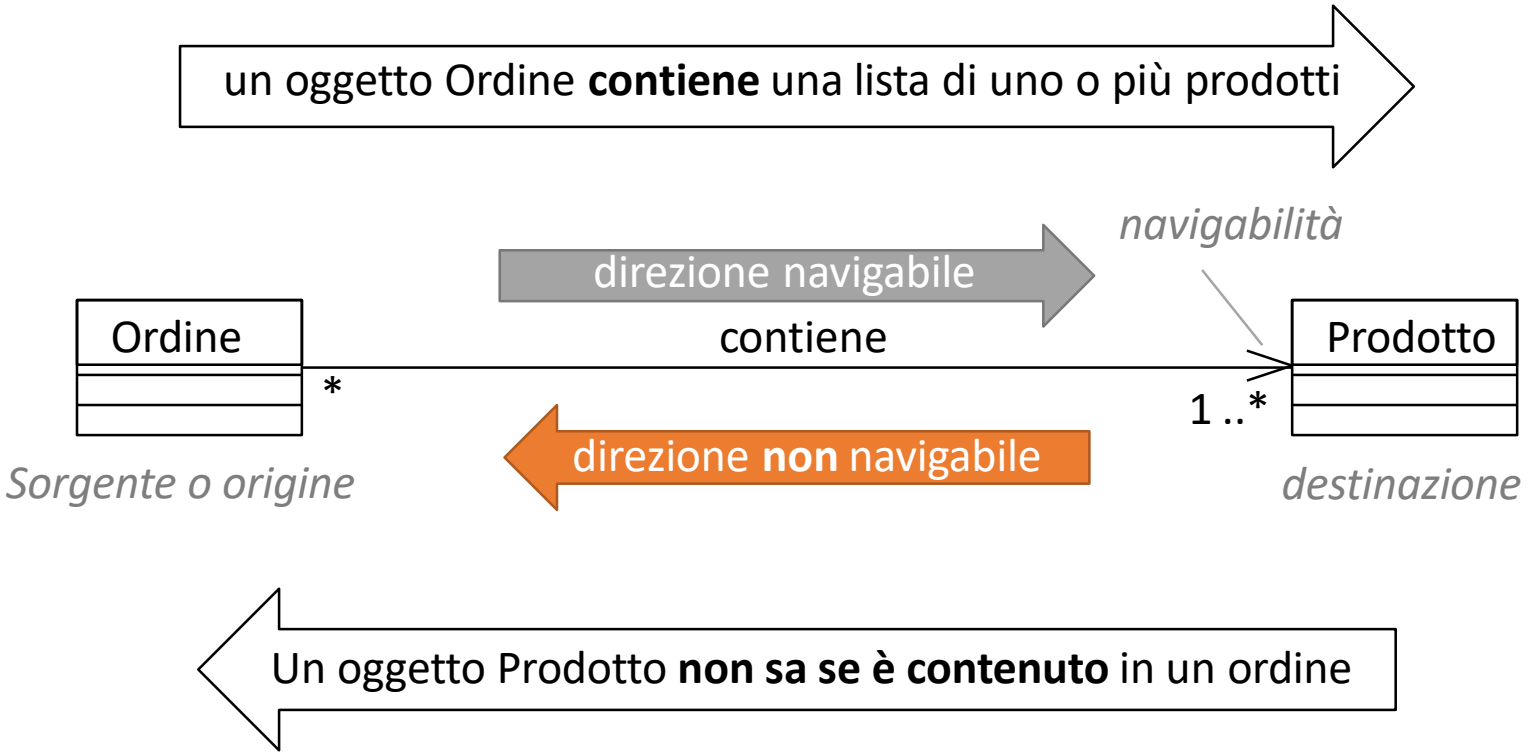
se minimo e massimo coincidono

- ***** asterisco indica «molti»

Una persona **lavora** per un’unica azienda 1

UML: Navigabilità di un'associazione

Il nome di una associazione, se possibile non dovrebbe esprimere una direzione, oppure esprime la direzione/navigazione prevalente. Es. il nome di relazione ***impiego*** tra le classi **lavoratore** e **azienda** è non direzionale, i nomi ***da_lavoro_a*** e ***lavora_a*** esprimono la direzione



Valori di molteplicità

I valori di molteplicità possono essere specificati come:

- **n** esattamente un intero **n** non negativo, min/max uguale
  - * un solo asterisco, indica zero o più
  - **a .. b** un intervallo [a, b] di interi non negativi;, «da minimo **a** fino a massimo ***b***», b può essere * “molti”
  - **a .. b, n, m, p .. q**: un insieme di intervalli e/o numeri; il valore più a destra può essere *

Molteplicità

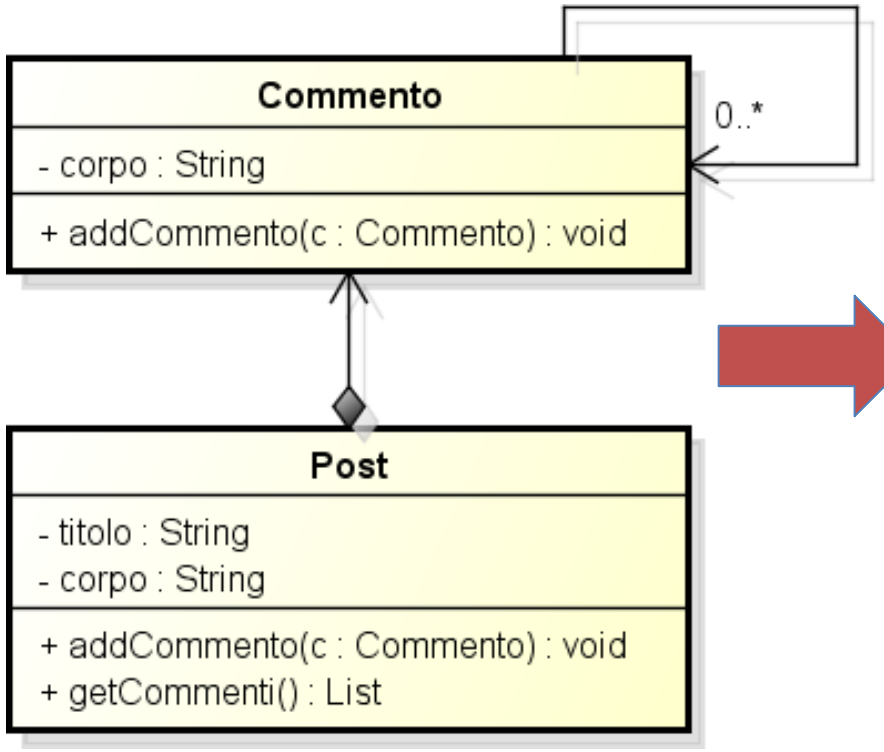
Molteplicità	Significato
1	Esattamente uno
*****	Zero o più
0 .. 1	Zero o uno
0 .. *	Zero o più (come un singolo *)
1 .. *	Uno o più
1 .. 6	Da uno a sei
1 ..3, 7 .. 10, 15, 19 .. *	Da 1 a 3, *oppure* da 7 a 10, *oppure* esattamente 15, *oppure* 19 o più

Tipi di Molteplicità tra coppie di classi

Classe X	Classe Y	Significato
1	**1**	Uno a uno, obbligatoria da ambo i lati
0..1	**1**	Uno a uno, opzionale a destra, Y collegato da min 0 a max 1 oggetto X
0 .. 1	**0..1**	Uno a uno, opzionale da ambo i lati
0 .. *	*****	Molti a molti, opzionale da ambo i lati
1 .. *	**1..***	Molti a molti, obbligatoria da ambo i lati
1 ..*	**0..***	Molti a molti, opzionale a sinistra, ogg X collegato a 0 o molti Y

ESEMPIO

- Trasformazione da diagramma di classe a codice



Java

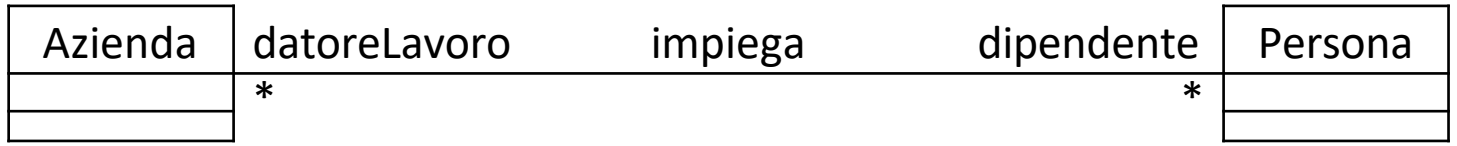
```
public class Commento {
  private String corpo = null;
  private List<Commento commenti =
    new ArrayList<Commento>();

  public void addCommento(Commento c) {
    commenti.add(c);
  }
}

public class Post {
  private List<Commento> commenti =
    new ArrayList<Commento>();

  // ...

  public List<Commento> getCommenti()
    {...}
}
```

Attributi e associazioni

- In una relazione molti-a-molti tra due classi, non sempre è possibile assegnare un attributo ad una delle due classi.
- Si consideri il seguente esempio:
  - Ogni oggetto `Persona` può essere dipendente di molti oggetti `Azienda`.
  - Ogni oggetto `Azienda` può impiegare molti oggetti `Persona`.

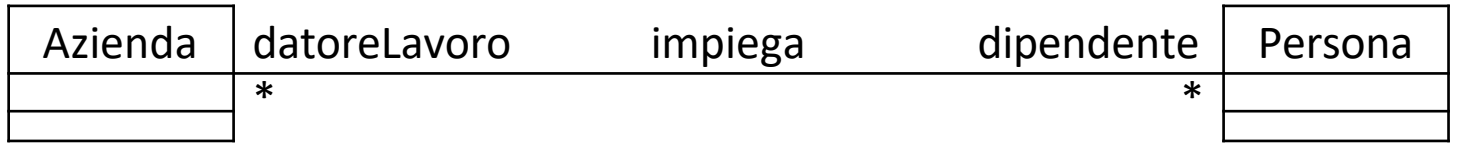
Attributi e associazioni

- Si assuma che ogni `Persona` percepisce uno stipendio da ogni `Azienda` in cui lavora.
- Dove collochiamo l'attributo stipendio: nella classe `Persona` o nella classe `Azienda`?



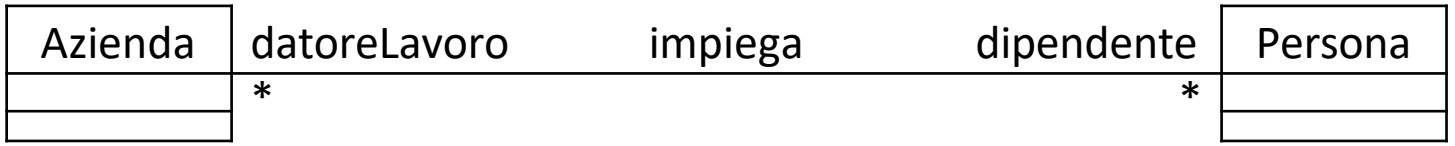
Attributi e associazioni

- Ipotesi 1 – attributo stipendio in `Persona`:
  - non riesco a modellare tutte le situazioni in cui una `Persona` lavora per diverse `Aziende` percependo uno stipendio diverso da ciascuna di esse.



Attributi e associazioni

- Ipotesi 2 – attributo stipendio in `Azienda`:
  - non riesco a modellare tutte le situazioni in cui un'`Azienda` impiega molte `Persone` con stipendi potenzialmente diversi



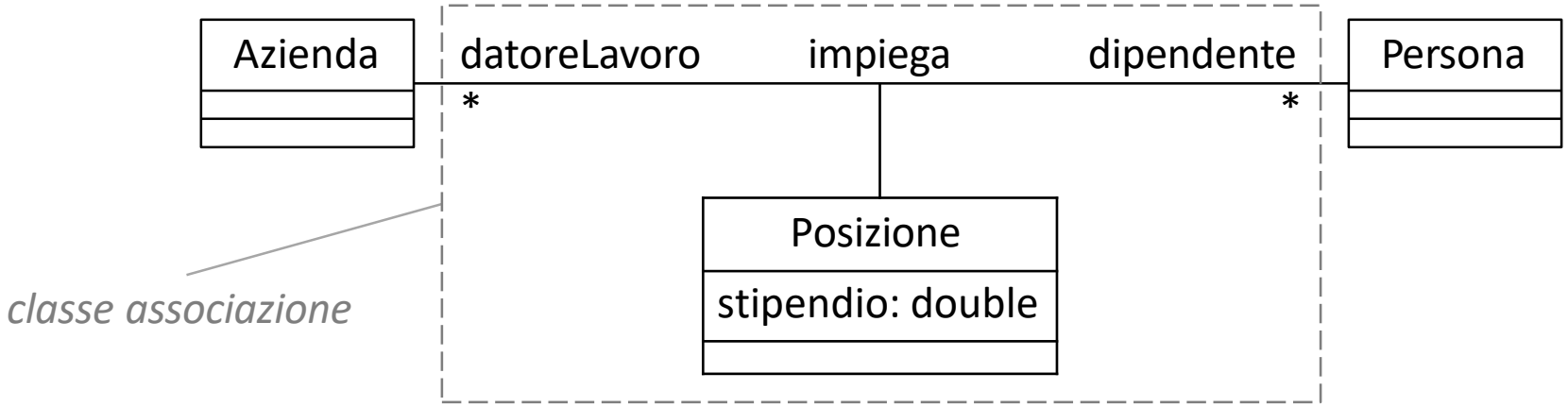
Attributi e associazioni

- Le ipotesi 1 e 2 non consentono di ottenere un modello semanticamente corretto perché lo stipendio è una proprietà (attributo) dell'associazione stessa.
- Per ogni associazione "impiega" tra un oggetto `Persona` e un oggetto `Azienda`, c'è uno ***specifico stipendio*** per uno ***specifico rapporto di lavoro***.
- L'UML consente di modellare questa situazione con una **classe associazione**.

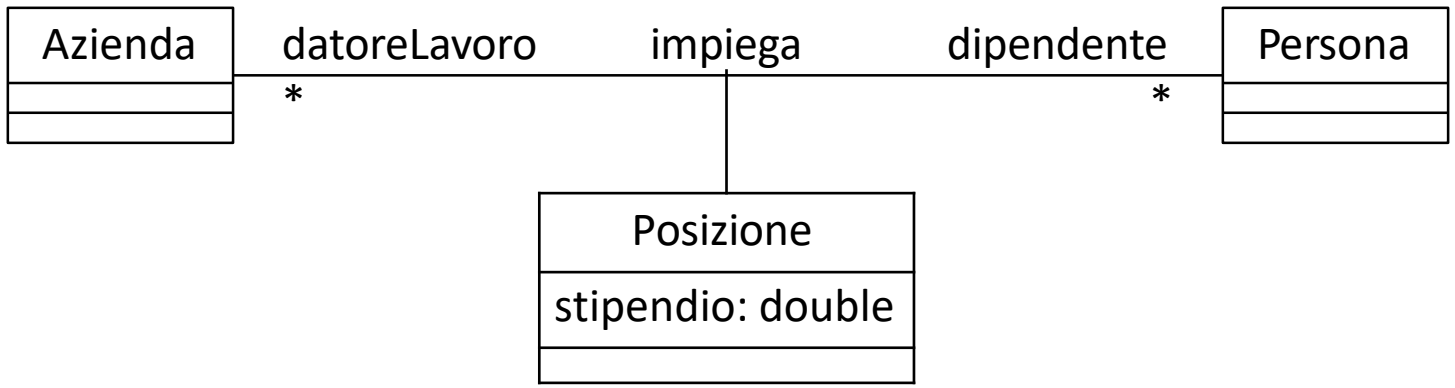
Classe di associazione

- Una **classe associazione** è un'associazione che è anche una classe. Oltre a connettere due classi, definisce un insieme di caratteristiche proprie dell'associazione.
- Una classe associazione è rappresentata:
  - dalla linea dell'associazione (compresi tutti i nomi di ruolo e molteplicità),
  - dal rettangolo della classe e
  - dalla linea tratteggiata verticale che li collega.

Notazione UML di una classe associazione



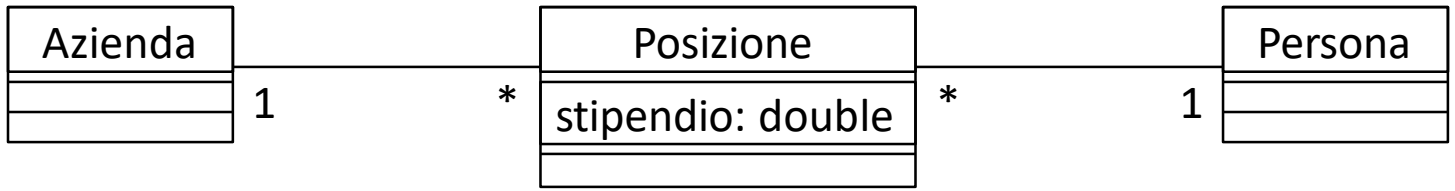
Notazione UML di una classe associazione



Classi associazione

- Si può usare una **classe associazione** quando vi sia al massimo un unico collegamento tra ogni coppia di oggetti in ogni istante:
  - una `Persona` ha al più una sola `Posizione` con una stessa `Azienda`.
- In caso negativo, cioè se ***sono possibili molti collegamenti reciproci*** in un qualunque istante si "reifica" (rende reale) la relazione trasformandola in una classe esplicita.
  - una `Persona` ha (simultaneamente) più di una `Posizione` con una stessa `Azienda`.

Uso delle classi associazioni



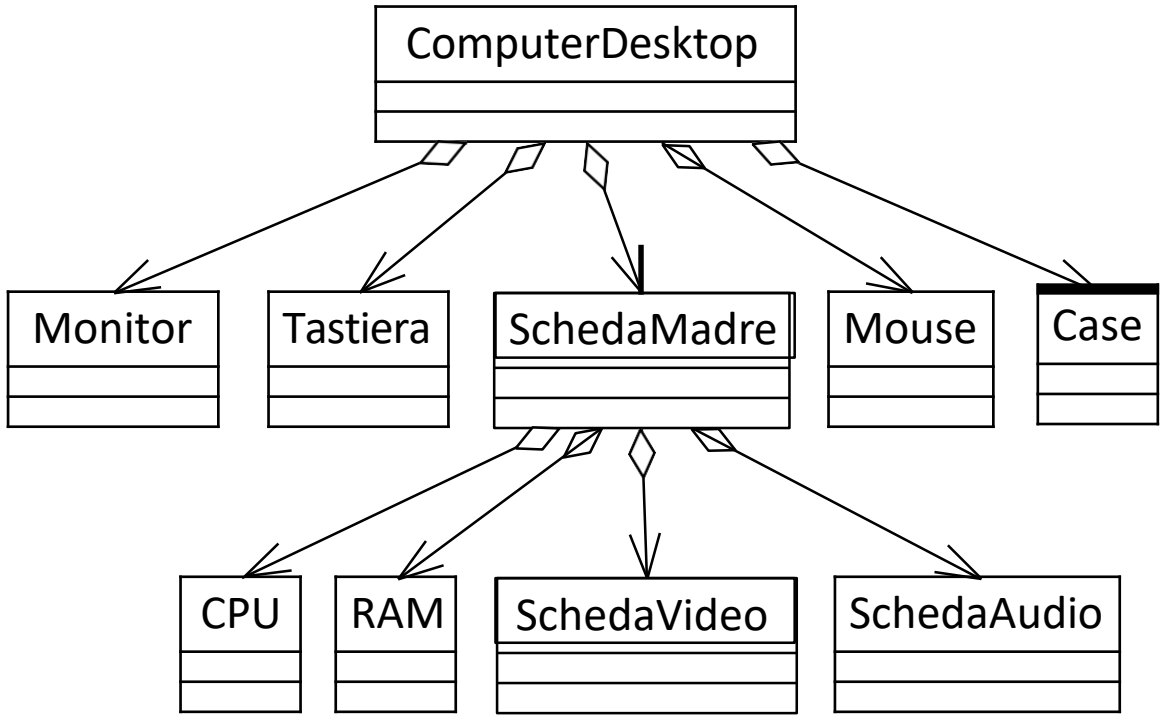
Aggregazione

- Le **aggregazioni** sono un tipo speciale di associazione nel quale le due classi partecipanti non hanno un rango uguale, ma hanno una relazione di tipo **tutto-parte**.
- Un'aggregazione descrive come la classe che ha il ruolo del tutto è composta di (ha) altre classi, che hanno il ruolo di parti.

UML: aggregazioni

- Le aggregazioni sono rappresentate da un’associazione che mostra un rombo sul lato dell’aggregato.

Esempio – ComputerDesktop



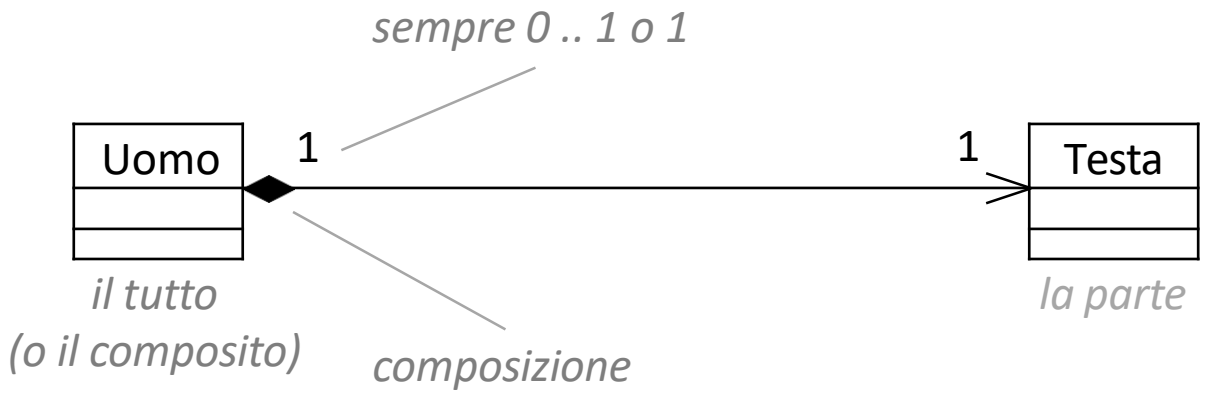
Composizione

- La **composizione** è una forma più forte di aggregazione.
- Come l'aggregazione, si tratta di una relazione di tipo **tutto-parte**.
- La differenza è che in una composizione **la parte non può esistere al di fuori del tutto**:
  - le parti esistono solo all'interno del tutto
  - se il tutto è distrutto anche le parti muoiono.

- Graficamente indicata da **rombo solido/pieno a lato «classe tutto» diretta verso «classe parte»**

Notazione UML per composizioni

- Le composizioni sono rappresentate da un'associazione che mostra un rombo solido sul lato del composito.

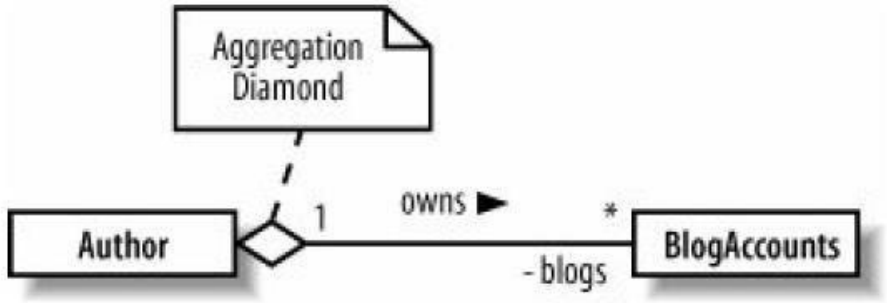


Semantica della composizione

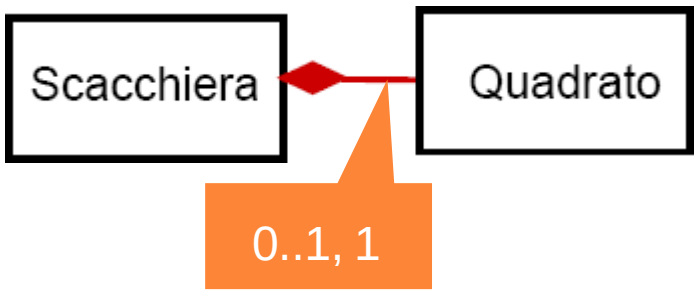
- Ogni parte appartiene ad un unico composito, dunque:
  - possono esistere gerarchie di composizione,
  - non possono esistere reti di composizione (albero)
- Il composito è l'unico responsabile del ciclo di vita delle parti.
  - Quando si distrugge un composito, questo deve a suo volta
    - distruggere tutte le sue parti, oppure
    - cederne la responsabilità a un altro oggetto.

AGGREGAZIONE E COMPOSIZIONE

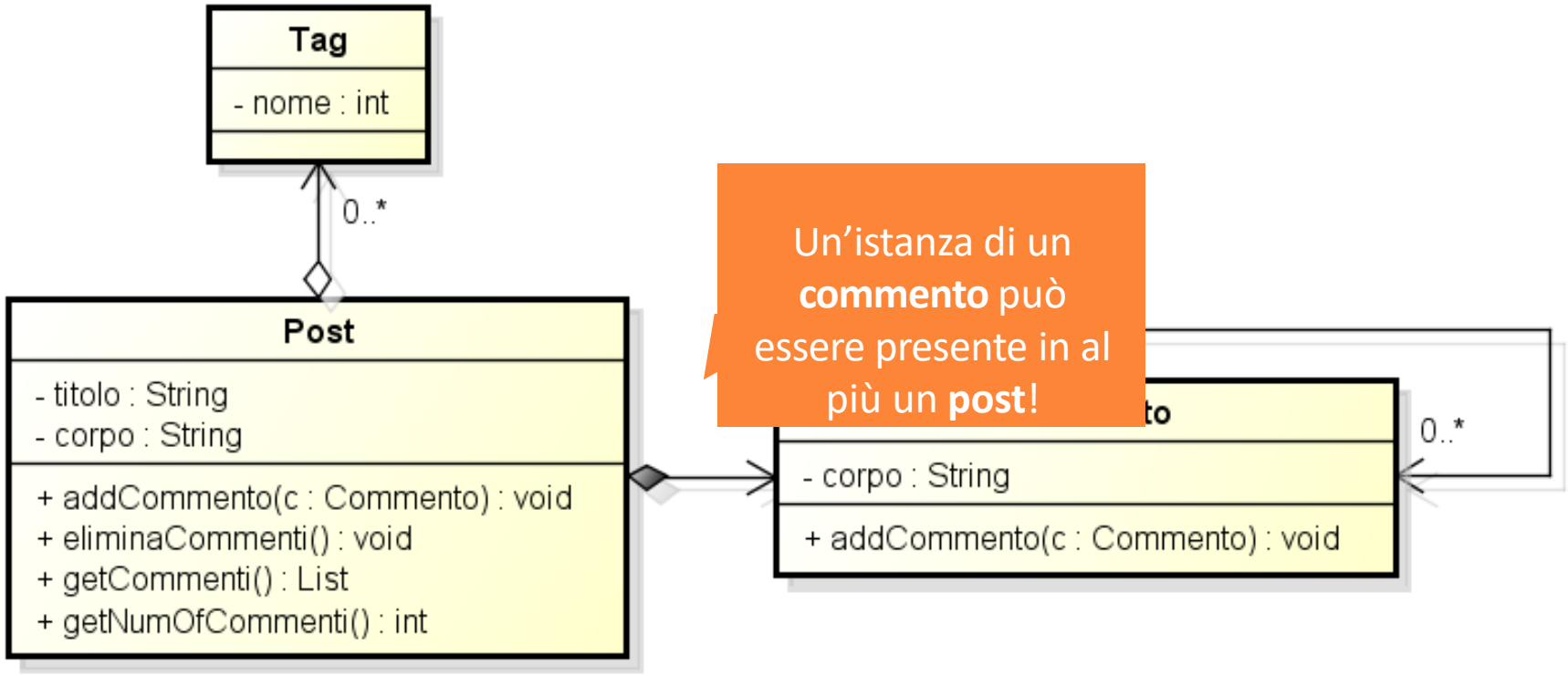
- Aggregazione
  - Relazione di tipo “parte di” (*part of*)
    - Gli aggregati possono essere condivisi



- Composizione
  - Come aggregazione, ma:
    - **Gli aggregati appartengono ad un solo aggregato**
    - Solo l’oggetto intero può creare e distruggere le sue parti



ESEMPIO



COMPOSIZIONE:

- la parte non può esistere al di fuori del tutto: un commento non può esistere senza il post che commenta

un tag può esistere indipendentemente da un post che aggrega più tag

- i compomenti appartengono ad un solo aggregato COMPOSTO: quindi un commento non puè essere associato a due post,

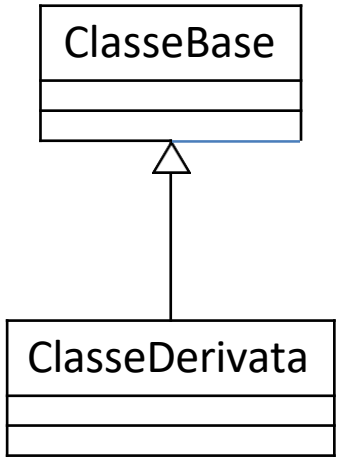
un tag può essere associato a più post in cui lo stesso tag è aggregato

Generalizzazione/Specializzazione (ereditarietà)

- Nella OOP l'**ereditarietà** serve a derivare una nuova classe (**sottoclasse**) da una classe esistente (**superclasse**) in modo tale che la sottoclasse:
  - acquisisce (eredita) tutti gli attributi e le operazioni (metodi) della superclasse,
  - può aggiungere altri attributi e operazioni proprie,
  - può ridefinire alcune operazioni della superclasse.
- In UML si usa il termine **generalizzazione** per indicare che una classe è una superclasse di un'altra classe.

UML generalizzazioni/specializzazioni

- La **generalizzazione (specializzazione)** è rappresentata da una linea che connette le due classi con una **freccia** (**triangolo vuoto**) dalla classe derivata (sottoclasse) alla classe base (superclasse).
- In UML è possibile rappresentare l'**ereditarietà multipla**
- Se possibile la superclasse (classe base) viene posta sopra la sottoclasse (classe derivata).

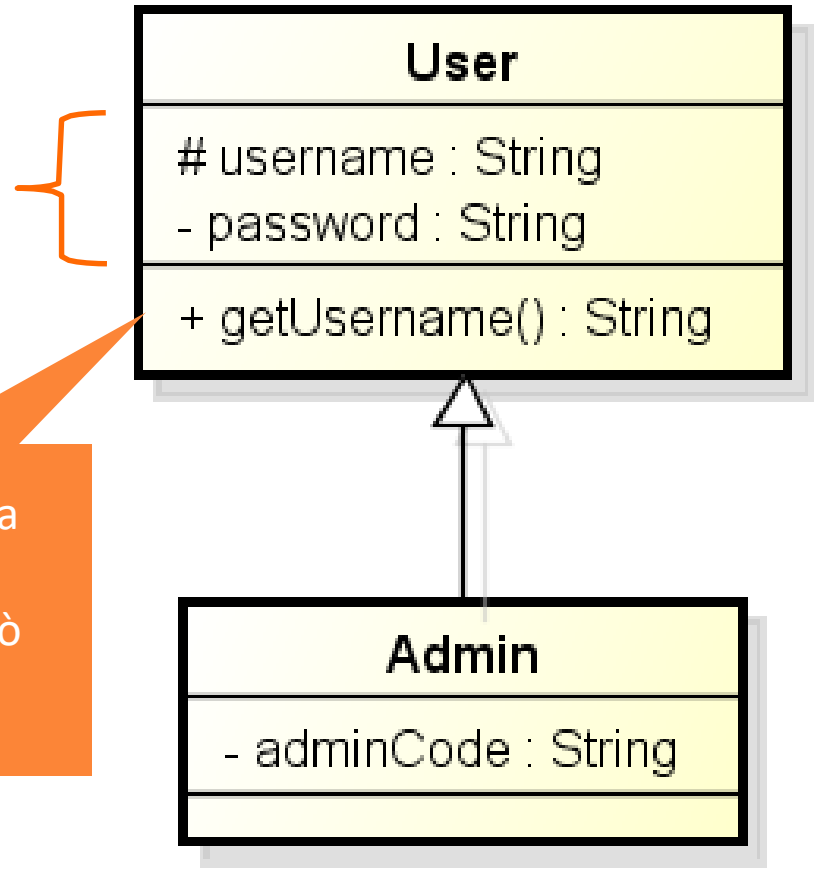


GENERALIZZAZIONE

- A generalizza B, se ogni oggetto di B è anche un oggetto di A
  - Equivale all'ereditarietà dei linguaggi di programmazione
    - Ereditarietà multipla supportata, ma da **NON USARE!**
  - Le proprietà della superclasse non si riportano nel diagramma della sottoclasse
    - A meno di *override*
  - Sostituibilità
    - Sottotipo ≠ sottoclasse
      - interfacce / implementazione

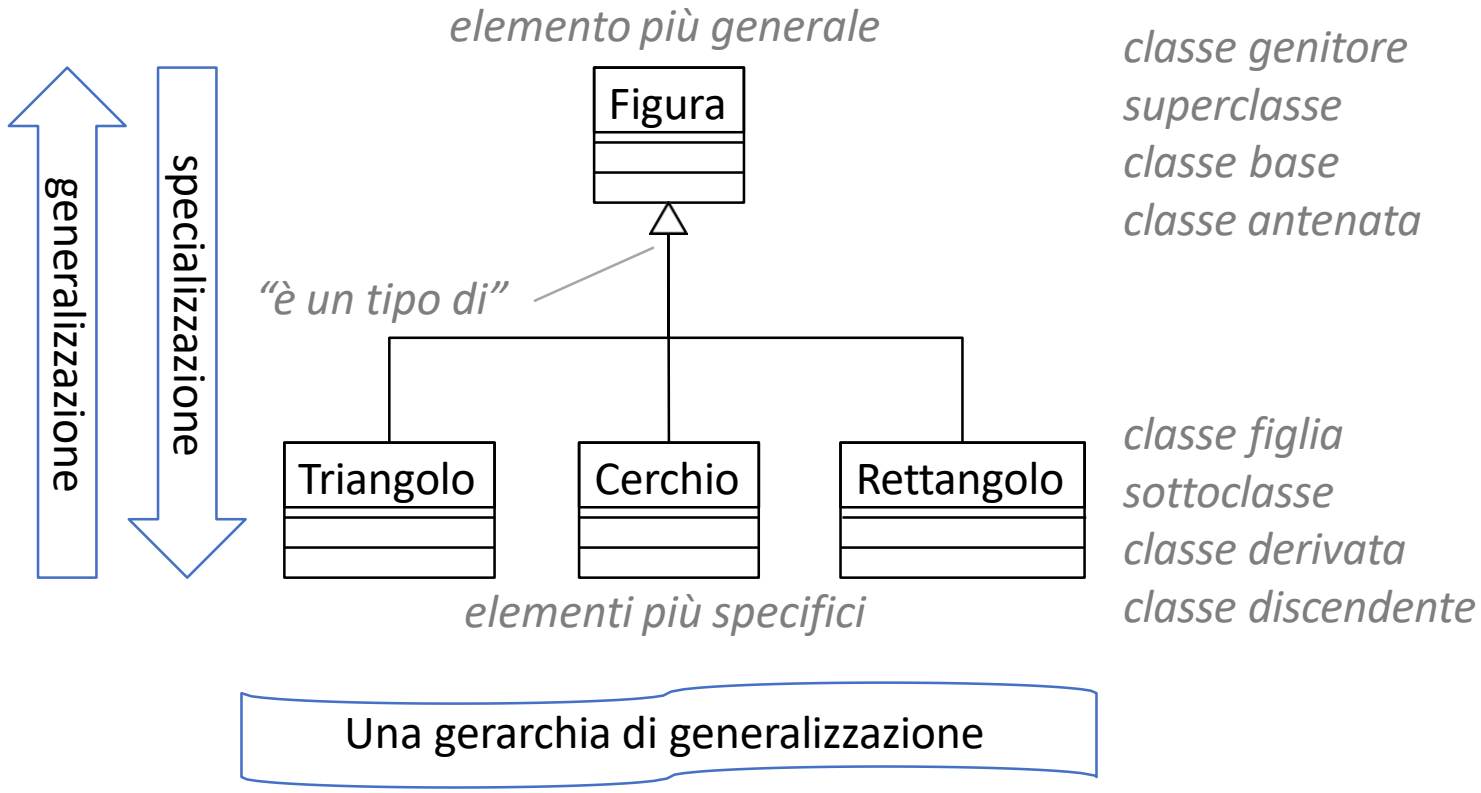
ESEMPIO

Gli attributi *protected* sono visibili anche dalle classi derivate



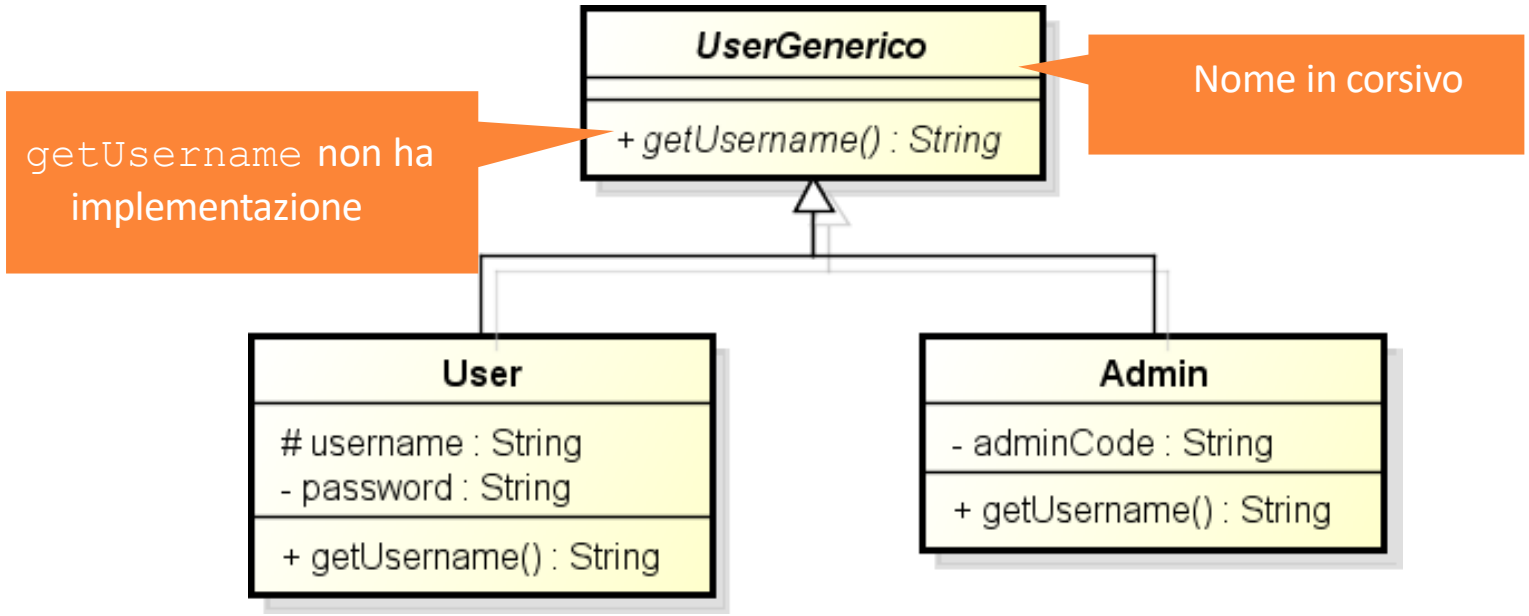
Il metodo della classe base è ereditato e può esserne fatto l'*override*

Generalizzazione/Specializzazione

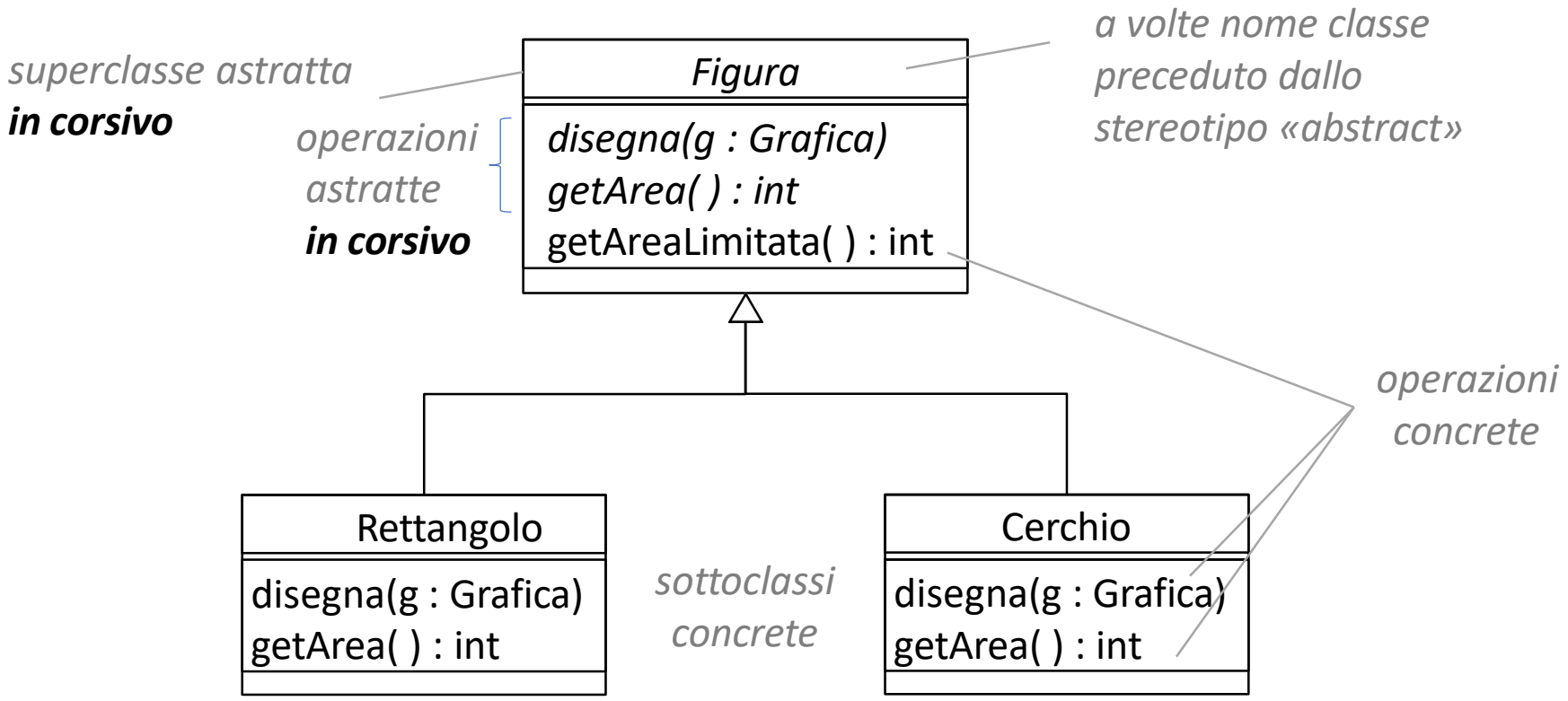


CLASSI ASTRATTE

- Classe Astratta `{abstract}`
  - Classe che non può essere istanziata
    - Operazione astratta non ha implementazione
    - Altre operazioni possono avere implementazione



Classi astratte e polimorfismo



Notazione Classe Astrattta: nome in corsivo o stereotipo «Abstract»

N.B.: `getAreaLimitata()` ritorna il prodotto della larghezza e dell'altezza di una figura, viene calcolata sempre allo stesso modo; non dipende dal tipo di figura concreta considerata.

Relazione di dipendenza

- Una dipendenza è una relazione tra due o più elementi del modello, dove un cambiamento ad uno di essi (il **fornitore**) può influenzare o fornire delle informazioni necessarie all'altro (il **cliente**).
- Il cliente dipende dal fornitore in qualche modo.

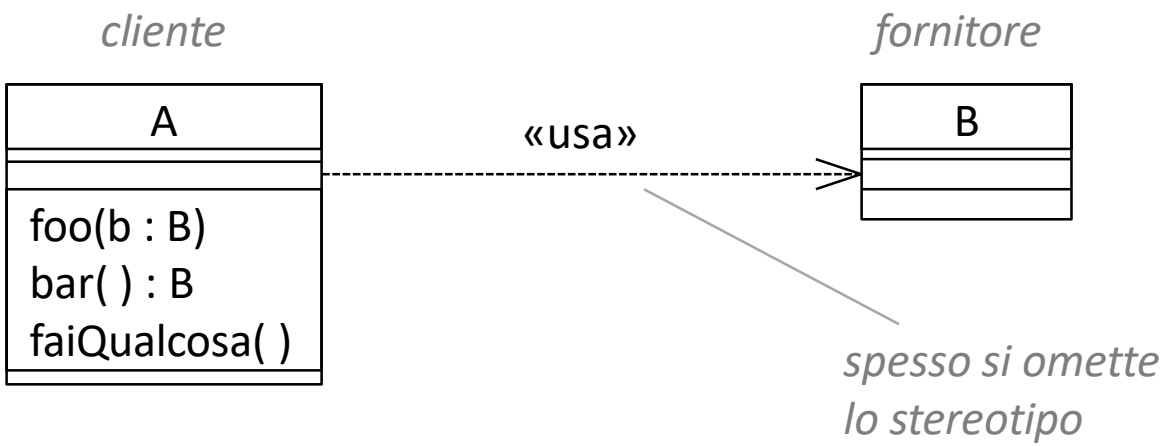
Dipendenze di uso

- Le dipendenze più diffuse sono le **dipendenze di uso**:
  - il cliente usa alcuni servizi della classe fornitore per implementare il proprio comportamento (invoca metodi).
- In particolare, lo stereotipo **«usa»**:
  - il cliente usa il fornitore come parametro, valore restituito o nella sua implementazione.

Notazione UML per dipendenza

- Le dipendenze sono indicate con **frecce tratteggiate** dal cliente verso il fornitore.

```
class A {
  . . .
  void faiQualcosa() {
    B mioB = new B();
    // Usa mioB in qualche modo
    . . .
  }
}
```



RELAZIONE DI DIPENDENZA

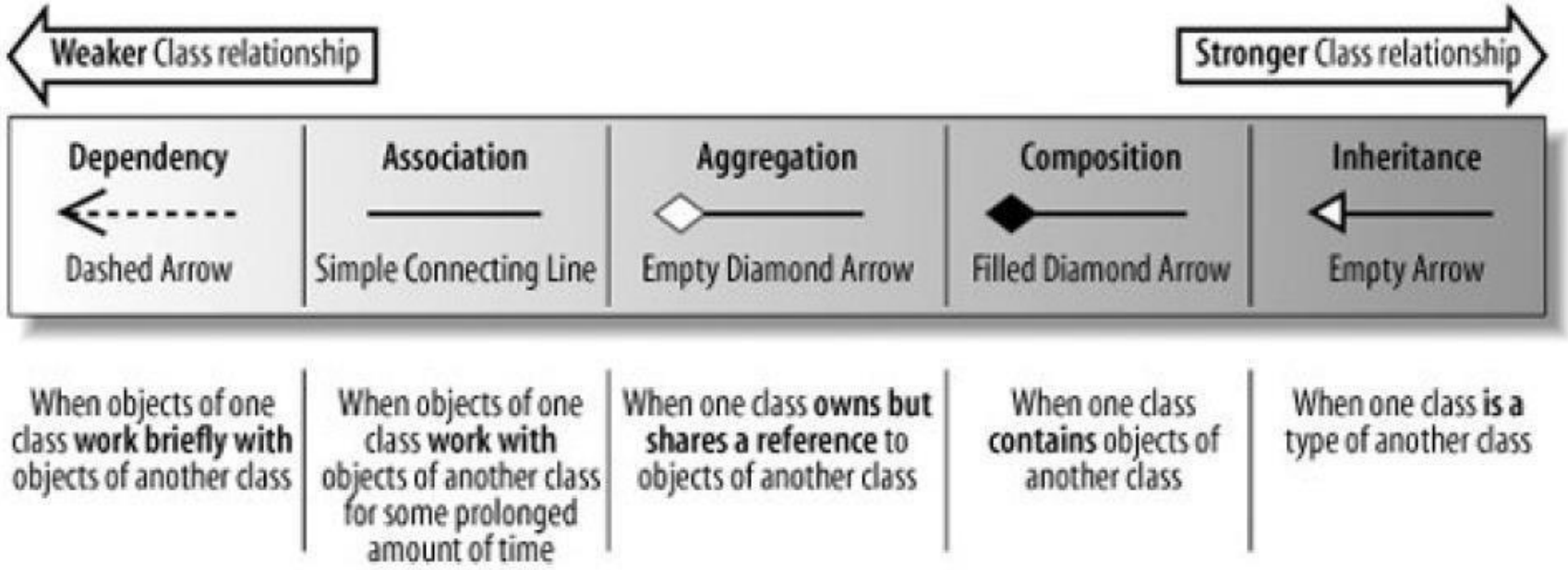
- Definizione

> Si ha dipendenza tra due elementi di un diagramma se la modifica alla definizione del primo (*server*) può cambiare la definizione del secondo (*client*)

- UML permettere di modellare ogni sorta di dipendenza
  - Non è una proprietà transitiva!
- Le dipendenze vanno minimizzate!
  - *Loose coupling*
- Da inserire solo quando danno valore aggiunto
  - Troppe dipendenze creano confusione nel diagramma

RELAZIONE DI DIPENDENZA

- Definizione: quanto vi è una relazione breve ad es. di uso

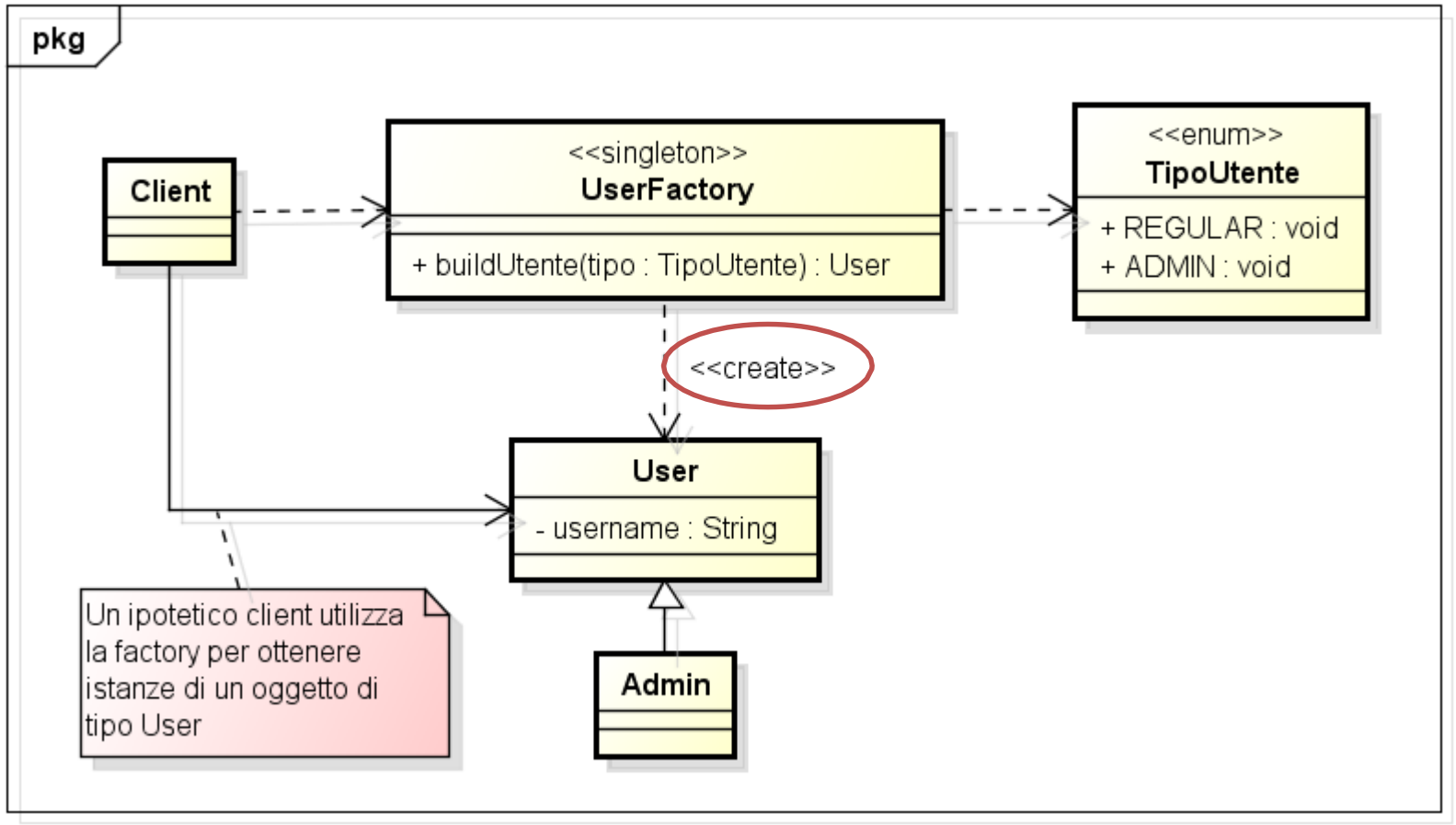


RELAZIONE DI DIPENDENZA

- Dipendenze in UML

Parola chiave	Significato
«call»	La sorgente invoca un'operazione della classe destinazione.
«create»	La sorgente crea istanze della classe destinazione.
«derive»	La sorgente è derivata dalla classe destinazione
«instantiate»	La sorgente è una istanza della classe destinazione (meta-classe)
«permit»	La classe destinazione permette alla sorgente di accedere ai suoi campi privati.
«realize»	La sorgente è un'implementazione di una specifica o di una interfaccia definita dalla sorgente
«refine»	Raffinamento tra differenti livelli semantici.
«substitute»	La sorgente è sostituibile alla destinazione.
«trace»	Tiene traccia dei requisiti o di come i cambiamenti di una parte di modello si colleghino ad altre
«use»	La sorgente richiede la destinazione per la sua implementazione.

ESEMPIO

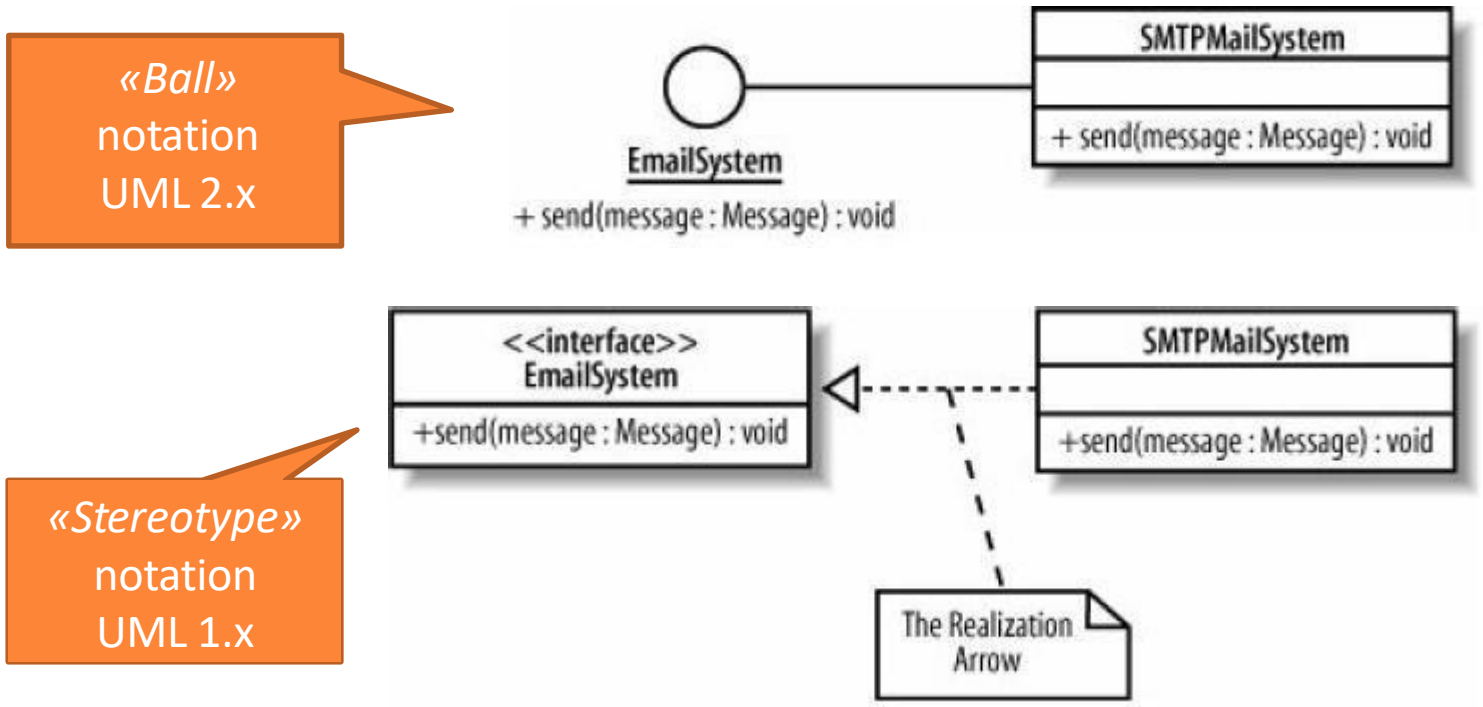

pkg
Client
<<singleton>>
UserFactory
+ buildUtente(tipo : TipoUtente) : User
<<enum>>
TipoUtente
+ REGULAR : void
+ ADMIN : void
<<create>>
User
- username : String
Admin
Un ipotetico client utilizza la factory per ottenere istanze di un oggetto di tipo User

Interfacce e realizzazioni

- Un'**interfaccia** è un insieme di funzionalità pubbliche identificate da un nome.
- Un'interfaccia non ha attributi, ma soltanto operazioni (metodi).
- Una **realizzazione** è una relazione tra una classe e un'interfaccia; indica che la classe implementa le operazioni dell'interfaccia.

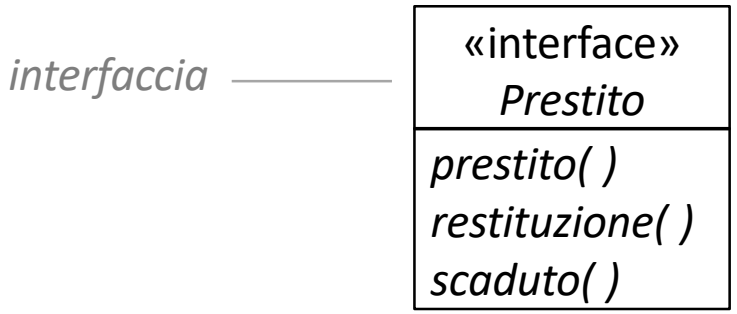
Interfacce

- Interfaccia «`interface`»
  - Cluna interfaccia è una **classe priva di implementazione**
    - Una classe realizza un'interfaccia se ne implementa le operazioni



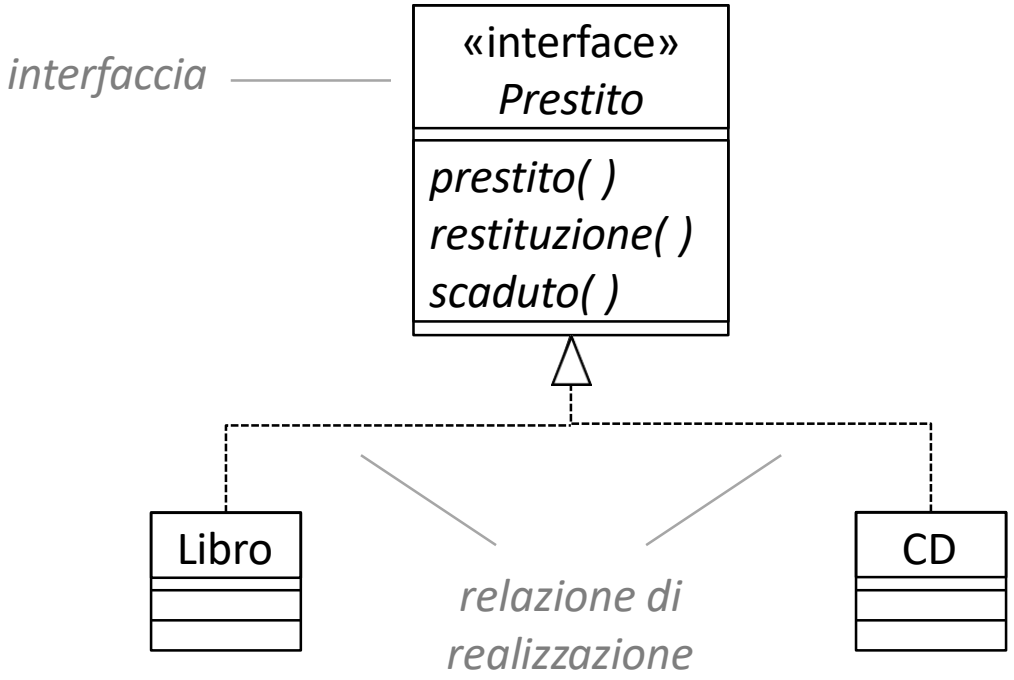
Notazione UML per interfacce e realizzazioni

- Un’interfaccia si rappresenta come una classe:
  - si omette la sottosezione attributi,
  - nome e metodi si scrivono in corsivo,
  - da UML2.x si può usare la ***lollipop notation*** *o* ***ball notation*** o includere lo stereotipo **«interface» UML1.x**.



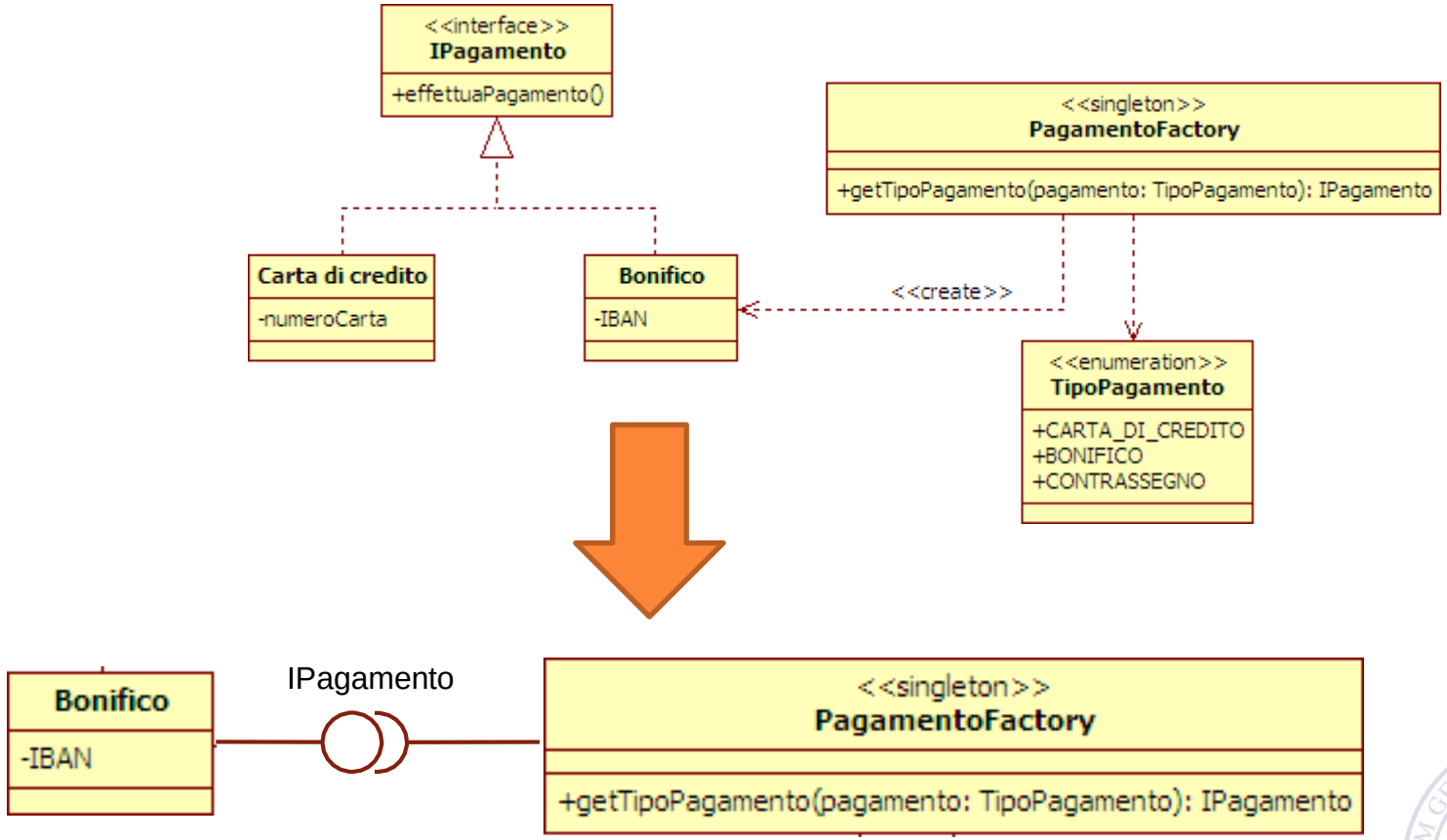
Notazione UML per interfacce e realizzazioni

- La relazione di realizzazione è visualizzata da una **linea tratteggiata con una freccia (triangolo vuoto)** che punta dalle *realizzazioni* all'*interfaccia*

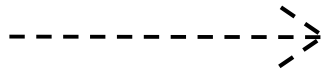
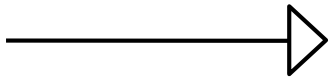
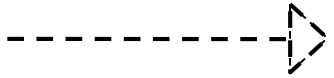


ESEMPIO

- Interfacce



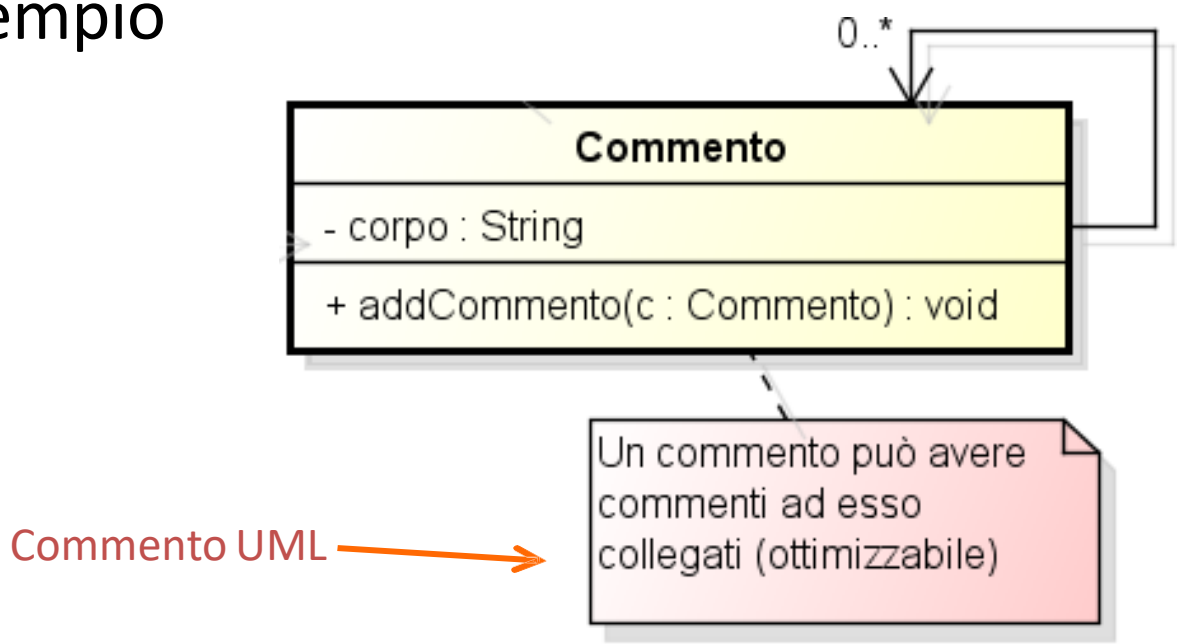
Tipi di relazioni: sommario

Tipo di relazione	Sintassi UML sorgente destinaz.	Descrizione
Dipendenza		Sorgente dipende da destinazione e può essere influenzato dai suoi cambiamenti
Associazione		Descrive un insieme di collegamenti tra oggetti
Aggregazione		Destinazione è parte integrante di sorgente
Composizione		Aggregazione più forte (più vincolata)
Generalizzazione		Sorgente è una specializzazione (estensione) di destinazione, che è più generale
Realizzazione		Sorgente garantisce di eseguire il contratto specificato da destinazione

COMMENTI E NOTE

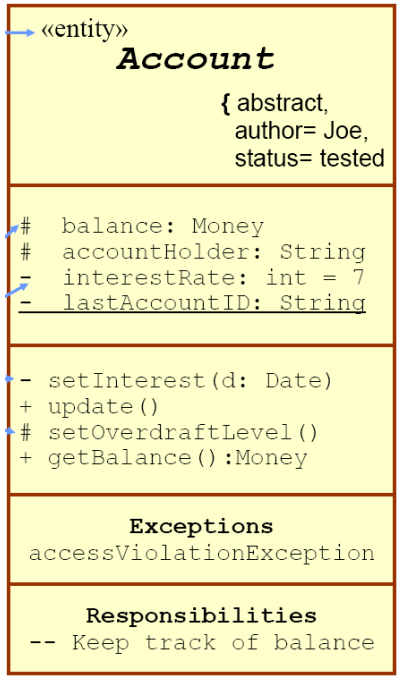
- Informazioni aggiuntive
  - Singole e solitarie
  - Legate a qualsiasi elemento grafico
    - Linea tratteggiata

- Esempio



CARATTERISTICHE VARIE

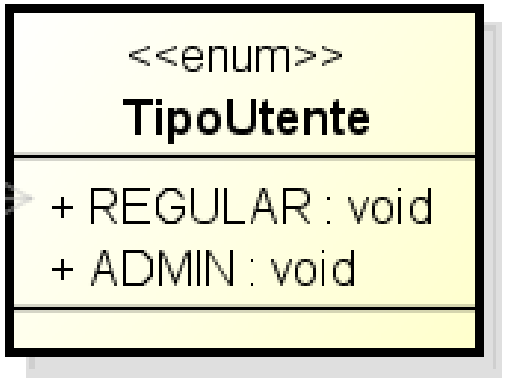
- Operazioni e attributi statici
  - Applicabili alla classe, non all'oggetto
    - Sottolineati sul diagramma
- Parole chiave
  - Estensione della semantica UML
  - Costrutto simile + parola chiave!
    - «interface»
    - {abstract}
- Responsabilità
  - Funzionalità offerte
  - Aggiunta alla classe con commento



CARATTERISTICHE VARIE

- Proprietà derivate
  - Possono essere calcolate a partire da altri valori
    - Definiscono un vincolo fra valori
    - Si indicano con "/" che precede il nome della proprietà

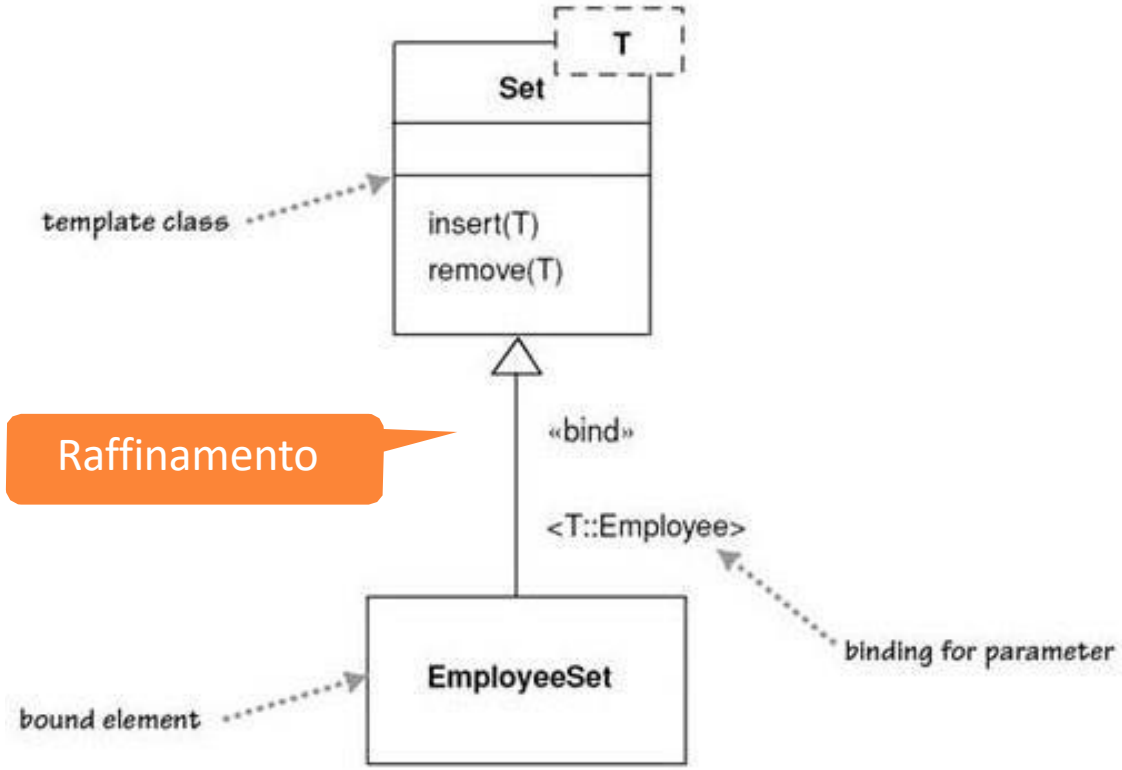
- Proprietà read only e frozen
  - `{readOnly}`
    - Non vengono forniti i servizi di scrittura
  - `{frozen}`
    - Immutabile, non può variare nel suo ciclo di vita



- Enumerazioni
  - Insiemi di valori che non hanno altre proprietà oltre il valore simbolico
  - «enumeration»

CARATTERISTICHE VARIE

- Classi Parametriche
  - T è detto “segnaposto”
    - Come *template* C++ o *generics* Java



Raffinamento

CARATTERISTICHE VARIE

- Classi Attive
  - Eseguono e controllano il proprio *thread*

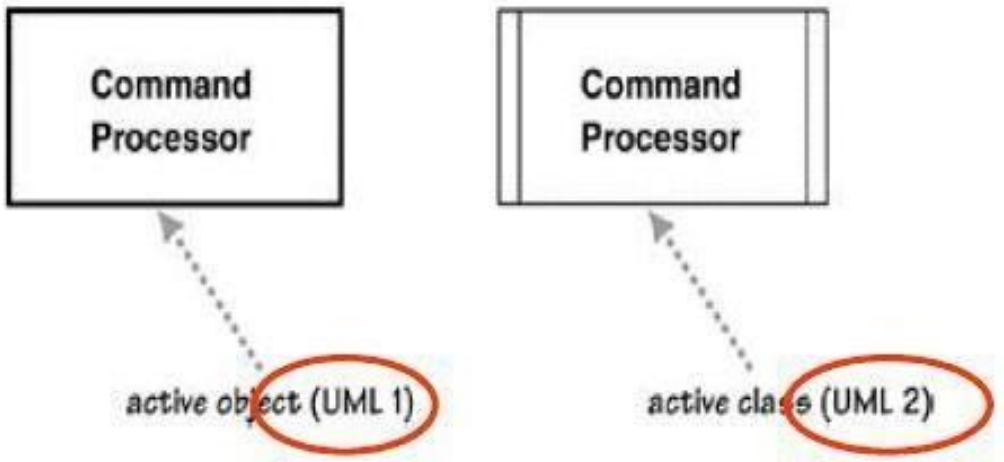


DIAGRAMMA DEGLI OGGETTI

- Rappresentazione delle istanze, comprensiva di associazioni e valori delle proprietà

```
nome dell'istanza : nome della classe
```

  - Utile per esempi illustrativi e oggetti particolati (Singleton)
  - Fotografia degli oggetti che compongono un sistema
  - Non ci sono parti obbligatorie
  - Specifica di istanza
    - Anche di classi astratte, omissione dei metodi, ecc…
- Utile quando i collegamenti fra le classi sono complicati
  - Visualizzazione di esempi di istanze delle classi
- Usati nei **diagrammi di sequenza**
- Oggetti specificati con **nome_oggetto_instanza : nomeClasse**
- Rappresentate Singole associazioni tra oggetti

ESEMPIO

Diagramma delle classi

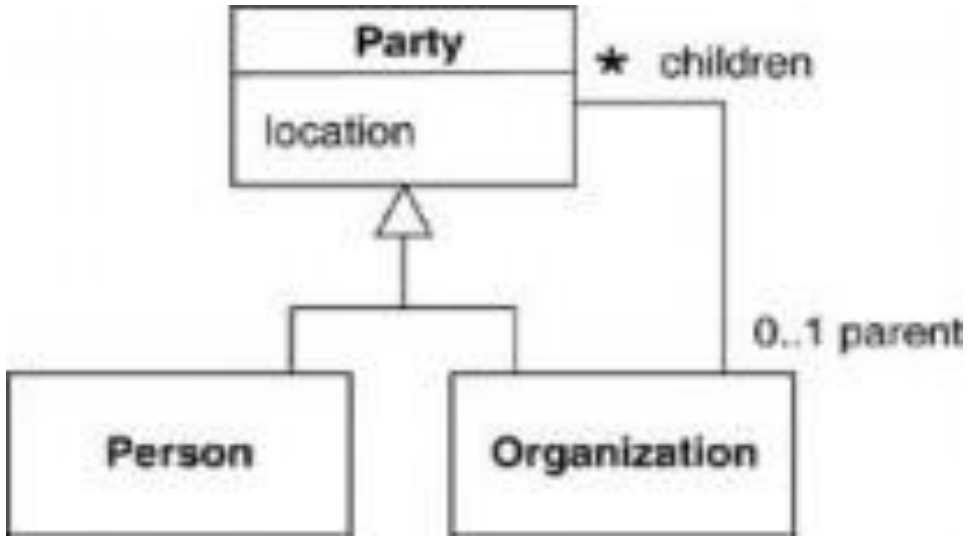


Diagramma degli oggetti

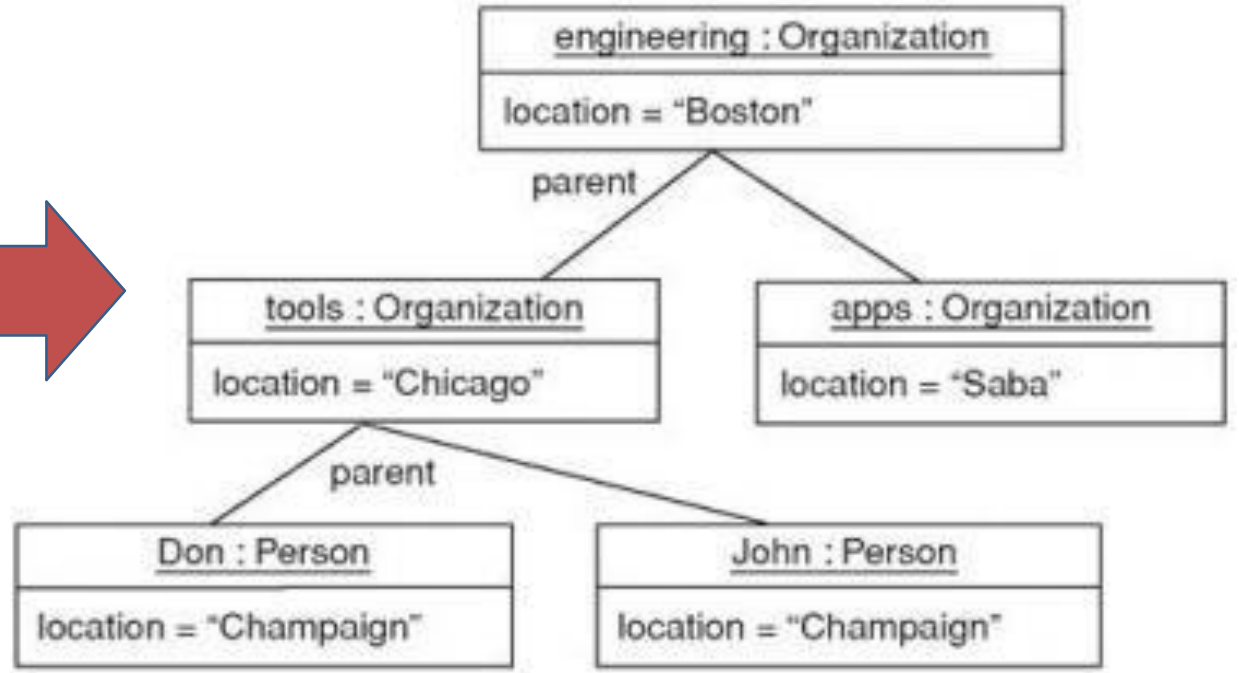


Diagramma degli oggetti esplicitano ed **istanziano su specifici oggetti le associazioni** del Diagramma delle classi

CONSIGLI UTILI

- Diagrammi molto ricchi di concetti
  - Non cercare di utilizzare tutte le notazioni disponibili
    - Cominciare dapprima con i concetti semplici
  - Una prospettiva concettuale permette di esplorare il linguaggio di un particolare *business*
    - Mantenere la notazione semplice e non introdurre concetti legati al *software*
  - Concentrarsi nel disegno dei diagrammi delle parti più importanti
    - Disegnare ogni cosa è sinonimo di diagrammi non fondamentali che diventano obsoleti molto presto!
  - Commentare le scelte effettuate!

ESERCIZIO di modellazione delle classi

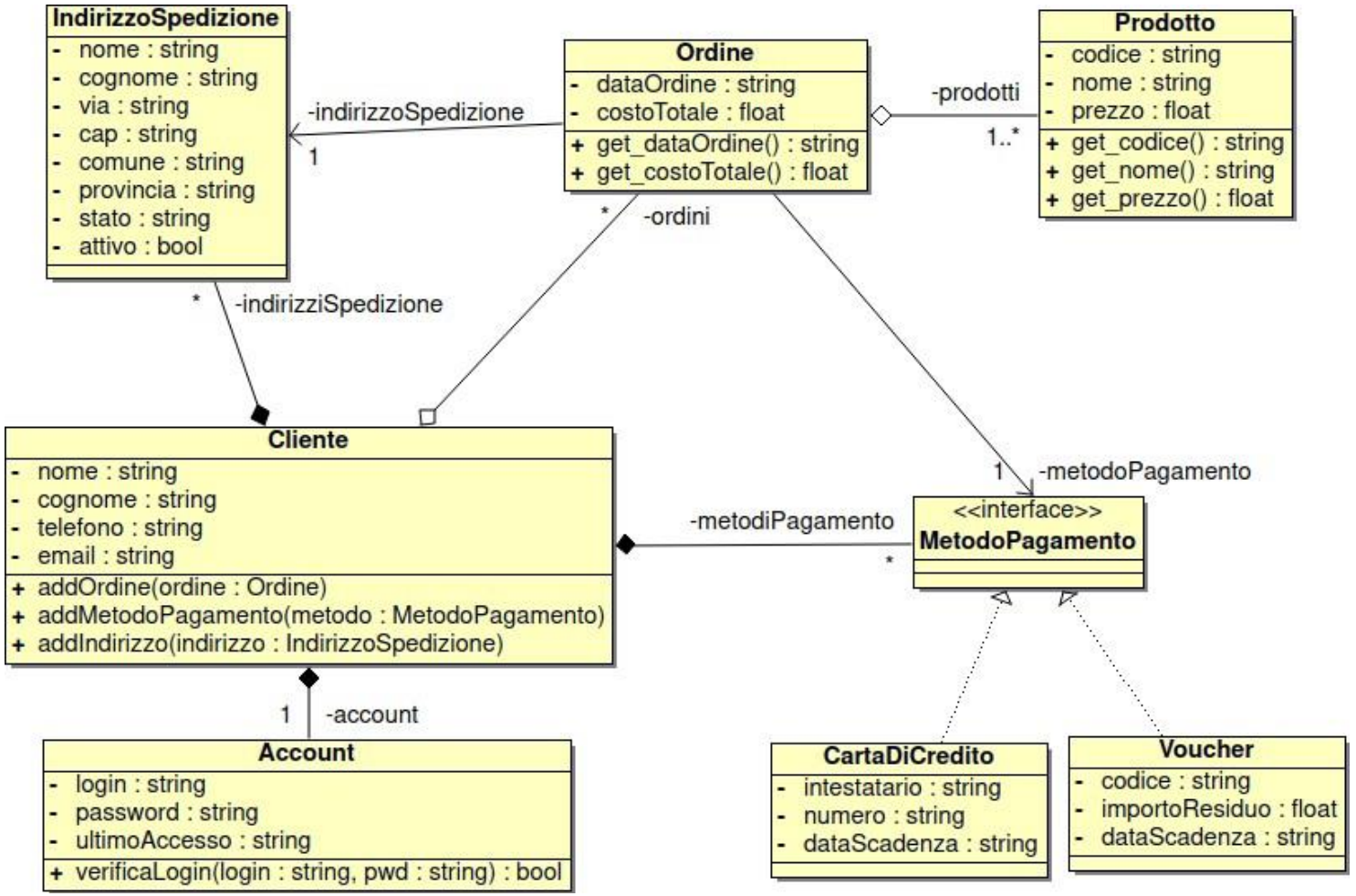
Specifiche

- Si vuole modellare un sistema per la gestione degli clienti di un portale di e-commerce.
  - È sufficiente modellare solo gli aspetti principali (classi con attributi e operazioni principali, relazioni tra classi).
- Per ogni cliente, il sistema memorizza vari tipi di informazione (slide successiva).
- Vi sono due principali metodi di pagamento: carta di credito o voucher.

Esercizio: altre specifiche di dettaglio

- Informazioni personali: nome, cognome, tel, email, …
- Account: login, password, ultimo accesso, …
- Metodi di pagamento: carte di credito (intestatario, numero, data di scadenza, …) o voucher (codice, importo residuo, data di scadenza)
- Indirizzi di spedizione: nome, cognome, via, comune, provincia, CAP, stato, …
- Ordini: metodo di pagamento, indirizzo di spedizione, data, costo, prodotti
- Prodotto: codice, nome, prezzo

Possibile diagramma di classe https://www.bouml.fr/



RIFERIMENTI

- OMG Homepage
  - www.omg.org
- UML Homepage
  - www.uml.org
- UML Distilled, Martin Fowler, 2004, Pearson (Addison Wesley)
- Fowler: «UML distilled. Guida rapida al linguaggio di modellazione standard», 4° Edizione, Pearson (2010)
- Learning UML 2.0, Kim Hamilton, Russell Miles, O'Reilly, 2006