UNIVERSITÀ DEGLI STUDI DELL'AQUILA

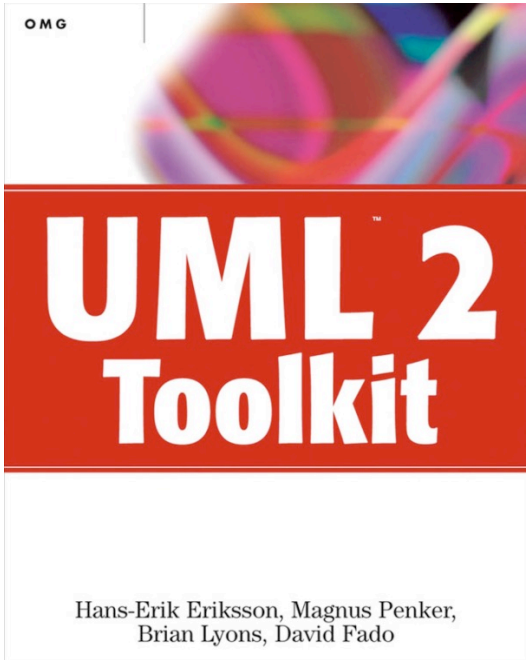
Laboratorio di Programmazione ad Oggetti

Ph.D. Juri Di Rocco
juri.dirocco@univaq.it
http://jdirocco.github.io

Sommario

- Breve introduzione a UML
- **Class Diagram**
- Introduzione
- Classe e Oggetto
- Elementi di una classe: Attributi ed operazioni
- Scope
- Relazioni
  - Associazione binaria
  - Associazione n-aria
  - Dipendenza
  - Aggregazione e composizione
  - Generalizzazione
  - Classi astratte
- Esempio

Risorse



Tool online per UML
https://www.genmymodel.com/

UML (1)

- Acronimo Unified Modeling Language
  - Language: linguaggio
  - Unified: standard, modello unificato
  - Modeling: per la modellazione
- Basato sul paradigma object-oriented
- Notazione UML semi-formale e grafica
- UML **non** definisce alcun processo per lo sviluppo del software
- Linguaggio per
  - Visualizzare
  - Specificare
  - Costruire
  - Generare
  - Documentare

gli elaborati di un sistema software

UML (2): Modelli

- In un mondo ideale si potrebbe rappresentare un intero sistema con una singola figura che comprende tutto
- Tale ideale è impossibile; solo i sistemi semplici possono essere rappresentati in questo modo
- Modello è una rappresentazione semplificata di una realtà, per esempio, osservare il funzionamento di un sistema oppure analizzare un problema da risolvere
- Modello fornisce una **astrazione** e serve per gestire la complessità di un sistema
- Un sistema informativo viene specificato tramite modelli dove vengono evidenziati gli **aspetti funzionali, e non, strutturali, comportamentali, ecc.**
- Per descrivere i modelli possono essere utilizzati strumenti/linguaggi formali oppure informali

UML (3)

UML è costituito da

- **Viste**: mostrano i diversi aspetti del sistema per mezzo di un insieme di diagrammi
- **Diagrammi**: permettono di descrivere graficamente le viste logiche
- **Elementi del modello**: concetti che permettono di realizzare vari diagrammi (es. attori, classi, relazioni, packages, oggetti, e così via)

- Versione attuale UML 2.x

UML: Diagrammi (4)

- Diagrammi strutturali
  - **Class** Diagram
  - **Object** Diagram
  - Component Diagram
  - Composite Structure Diagram
  - Package Diagram
  - Deployment Diagram
- Diagrammi comportamentali
  - Use Case Diagram (usato per la raccolta dei requisiti funzionali)
  - Activity Diagram
  - State Machine Diagram
- Diagrammi di interazione (derivati dal Behavior Diagram)
  - Sequence Diagram
  - Communication Diagram
  - Timing Diagram
  - Interaction Overview Diagram

Class Diagram (1)

- Mostra un insieme di classi, interfacce e le loro relazioni (dipendenza, associazione e generalizzazione, etc)
- Può essere visto come un grafo dove i **nodi** sono le classi e le interfacce, e gli **archi** sono le relazioni
- Possono contenere anche package o sottosistemi (utilizzati per raggruppare elementi)
- Modellano la parte **statica** di un sistema

Class Diagram (2)

- Modellare il vocabolario di un sistema
  - Classi modellano astrazioni di cose di un problema/dominio
  - Tali astrazioni fanno parte del vocabolario di un sistema
- Modellare semplici collaborazioni
  - Classi non vivono da sole
  - Collaborano insieme per fornire un comportamento che è più grande della somme di tutti gli elementi
- Modellare il cosiddetto **domain model**
  - Modello strutturale dei concetti base di un dominio con le sue relazioni
  - E' indipendente dallo strato di persistenza utilizzato (es. File, Database relazionale)

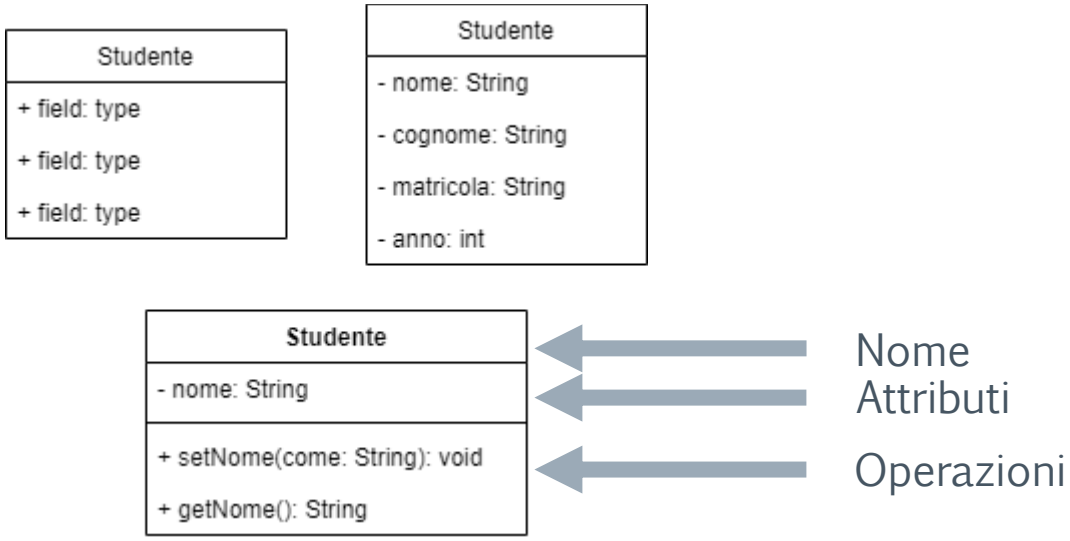
Classe

- Descrizione di un gruppo di oggetti con proprietà (attributi), comportamento (operazioni), relazioni e semantica comuni
  - Un oggetto è una istanza di una classe
- Astrazione che enfatizza caratteristiche rilevanti e sopprime le altre caratteristiche

Esempi

- **Nome**
  - Corso
- **Proprietà**
  - Nome, Luogo, Durata, Crediti, Inizio, Fine
- **Comportamento**
  - Aggiunta studente
  - Cancellazione studente
  - Verifica se è pieno

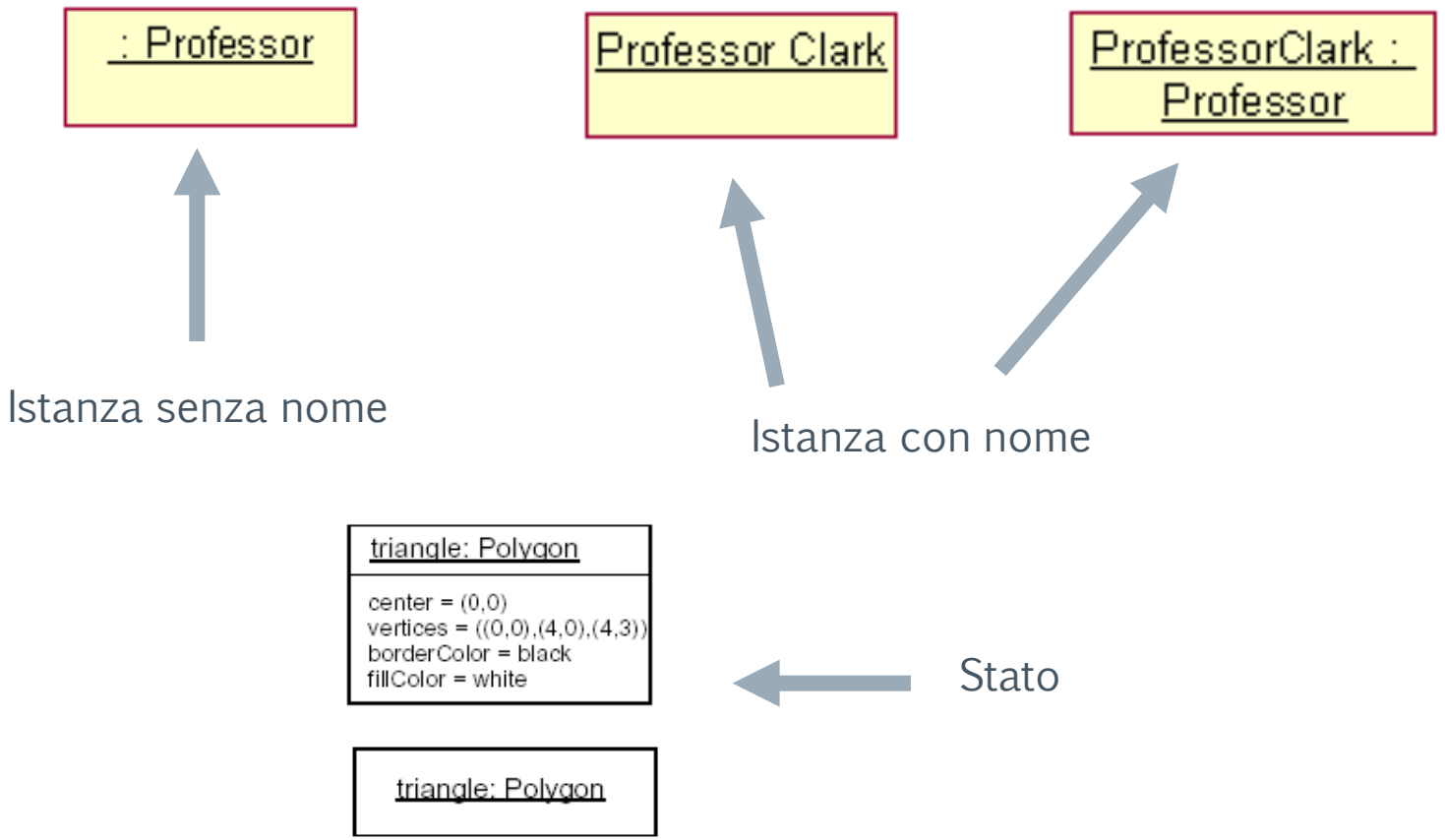
Rappresentazione UML



Oggetto

- Informalmente un oggetto rappresenta un entità fisica, concettuale o software
  - entità fisica: trattore
  - entità concettuale: processo chimico
  - entità software: lista, coda...
- Formalmente
  - Manifestazione concreta di un'astrazione
  - Entità con un confine e un'identità ben definite che incapsula stato e comportamento
  - **Istanza di una classe**
- Es.: Mia macchina, Mio computer

Rappresentazione UML



Attributi ed operazioni (1)

› Sintassi attributi

```
visibilità nome : tipo [molteplicità] = valore-default
```

› Tre livelli di visibilità
- **Public** (simbolo **+**) : qualsiasi classe può vedere tale feature
- **Private** (simbolo **–**) : soltanto la classe può vedere tale feature
- **Protected** (simbolo `#`) : solo le classi derivate possono vedere tale feature

› Molteplicità: default 1, esempio `[0..1]`, `*`

Attributi ed operazioni (2)

› Sintassi operazioni

```
visibilità nome ( lista-parametri ) : tipo-di-ritorno
```

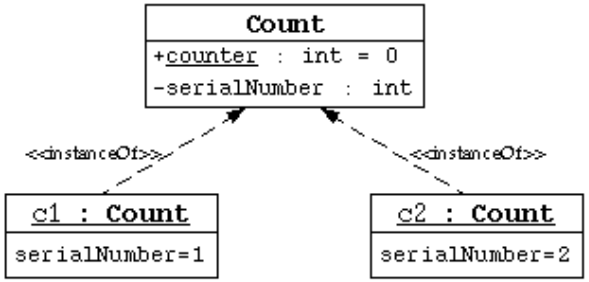
dove lista-parametri è una lista separata da `,` di parametri formali con la seguente sintassi

```
nome : tipo [molteplicità] = valore-default
```

› Visibilità uguale a quella degli attributi

Scope

- Può essere associato sia agli attributi che alle operazioni
- Due tipi
  - **instance**: ogni oggetto della classe ha il proprio valore (default)
  - **classifier (static)**: tutti gli oggetti della classe hanno lo stesso valore (UML è sottolineato)



Relazioni

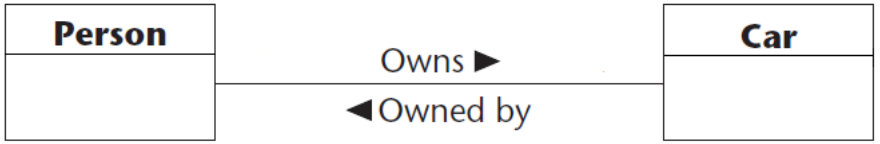
- Una relazione è una connessione tra cose (cioè classi, interfacce, componenti, package)
- Una relazione fornisce un cammino (pathway) per la comunicazione fra oggetti
- Tipi
  - Associazione
    - Composizione
    - Aggregazione
  - Dipendenza
  - Generalizzazione

Associazione

- Rappresenta una relazione **strutturale** tra oggetti di classi differenti
- Connessione bi-direzionale tra classi, si può navigare da un oggetto di una classe all'altro e viceversa
- Possibile avere associazioni circolari, cioè tra oggetti della stessa classe
- Associazione che collega due classi è detta binaria; n-aria che collega n classi
- Rappresentata mediante una linea continua che collega le due classi

Associazione: Nome

- Associazione può avere un nome generalmente è un verbo anche se i nomi sono ammessi
- E' possibile dare una direzione al nome mediante un triangolo che punta alla direzione che si intende far leggere la direzione
- Associazione potrebbe avere due nomi uno per ogni direzione



Associazione: Molteplicità (1)

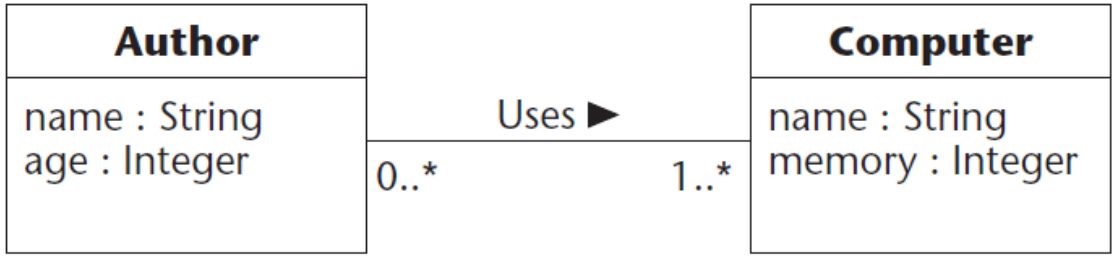
- Definisce **quanti** oggetti partecipano in una relazione
- Specifica il **numero** di istanze di una classe che è in relazione con **UNA** istanza dell'altra classe
- Applicata alla fine di ogni associazione

Associazione: Molteplicità (2)

Valore	Descrizione
N oppure *	Numero illimitato di istanze
1 (default)	Una sola istanza
0..n	Zero o più istanze
1..n	Una o più istanze
0..1	Zero o una istanza
<literal>[1]	Esatto numero di istanze
<literal>..n	Esatto numero o più istanze
<literal>..<literal>	Intervallo specificato di istanze
<literal>..<literal>, <literal>	Intervallo più numero specificato di istanze
<literal>..<literal>, <literal>..<literal>	Il numero di istanze sarà in uno degli intervalli specificati
1 Dove <literal> è un intero maggiore o uguale a 1. Esempio 5	

Associazione: class vs object diagram (1)

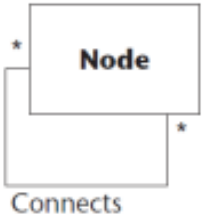
Class diagram



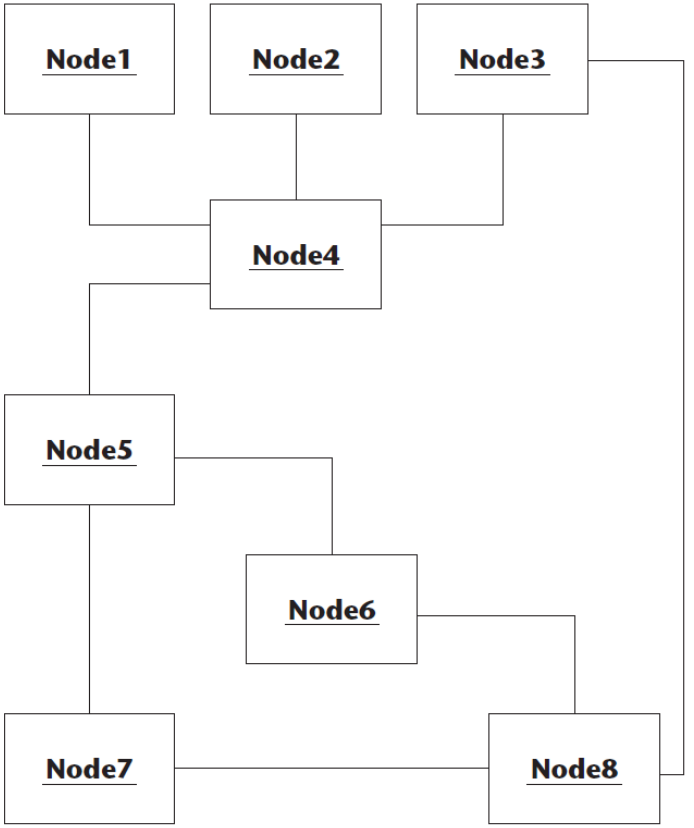
Object diagram

Associazione: class vs object diagram (2)

Class diagram

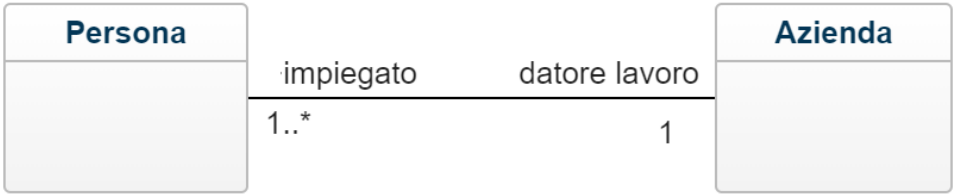


Object diagram



Associazione: Ruolo (1)

- Specifica l'aspetto che una classe gioca nell'associazione
- Ha un nome ed è posto vicino alla classe che gioca quel ruolo nell'associazione rispetto all'altra

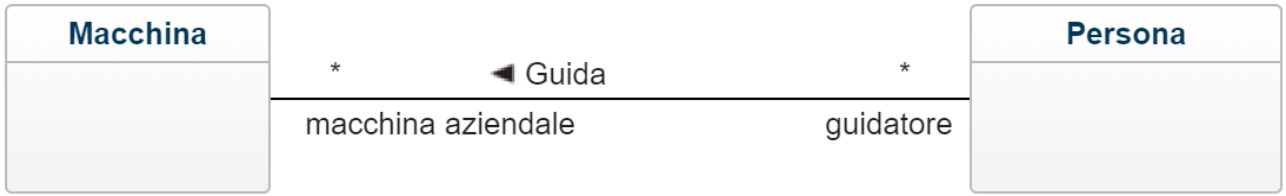


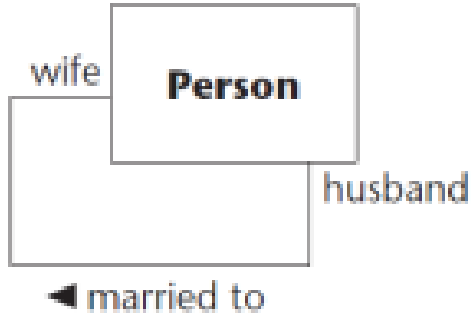
Associazione: Ruolo (2)

```
public class Persona {
   public Azienda datoreLavoro;
}

public class Azienda {
   public Persona[] impiegati;
}
```

Associazione: Ruolo (3)





Associazione: Ruolo (4)

› Java

```
public class Macchina {
      public Persona[] guidatori;
}

public class Persona {
      public Macchina[] macchineAziendali;
}

public class Person {
      public Person wife;
      public Person husband;
}
```

Associazione: Navigabilità (1)

- Nell'associazione la navigazione tra le classi è bi-direzionale
- In alcuni casi è possibile limitare tale navigazione ad una direzione
- Graficamente c'è una freccia che indica la navigazione
- Esempio
  - Dato un utente siamo in grado di sapere tutte le sue password
  - Dato una password non siamo in grado di risalire all'utente (è ragionevole)


```
public class User {
   public Password[] thePasswords;

}


public class Password {

}
```

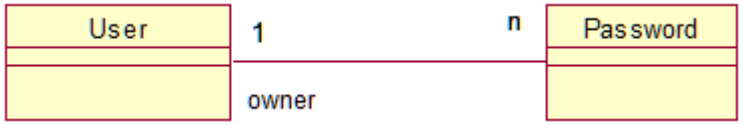
Associazione: Navigabilità (2)

```
public class Password {
   public User owner;

}

public class User {
   public Password[] thePasswords;

}
```

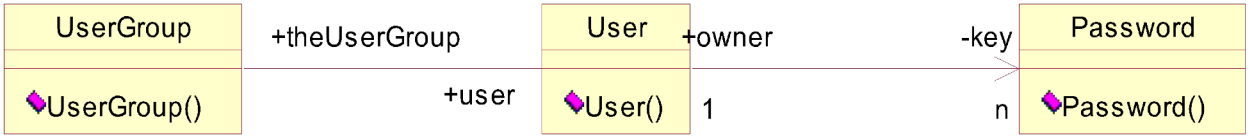


Associazione: Visibilità (1)

- In un'associazione è possibile vedere e navigare da un oggetto all'altro a meno che non ci sono restrizioni (navigabilità)
- E' possibile limitare la visibilità di tale navigazione
- Tre livelli di visibilità (+, –, #) collegati al ruolo dell'associazione
- Esempio: Password è privata di un Utente

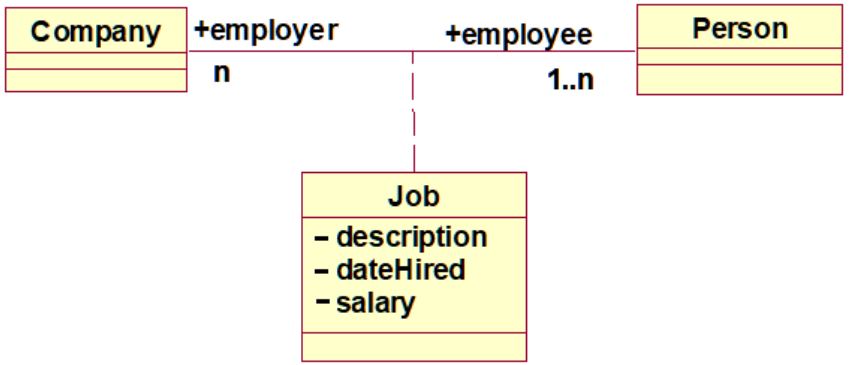
Associazione: Visibilità (2)

```
public class UserGroup {
   public User user;
}
public class Password {
}
public class User {
   private Password[] keys;
   public UserGroup theUserGroup;
}
```



Associazione: Association Class (1)

- Una associazione può possedere oltre alla molteplicità, ruoli e visibilità anche delle proprietà strutturali e comportamentali
- Esempio



Associazione: Association Class (2)

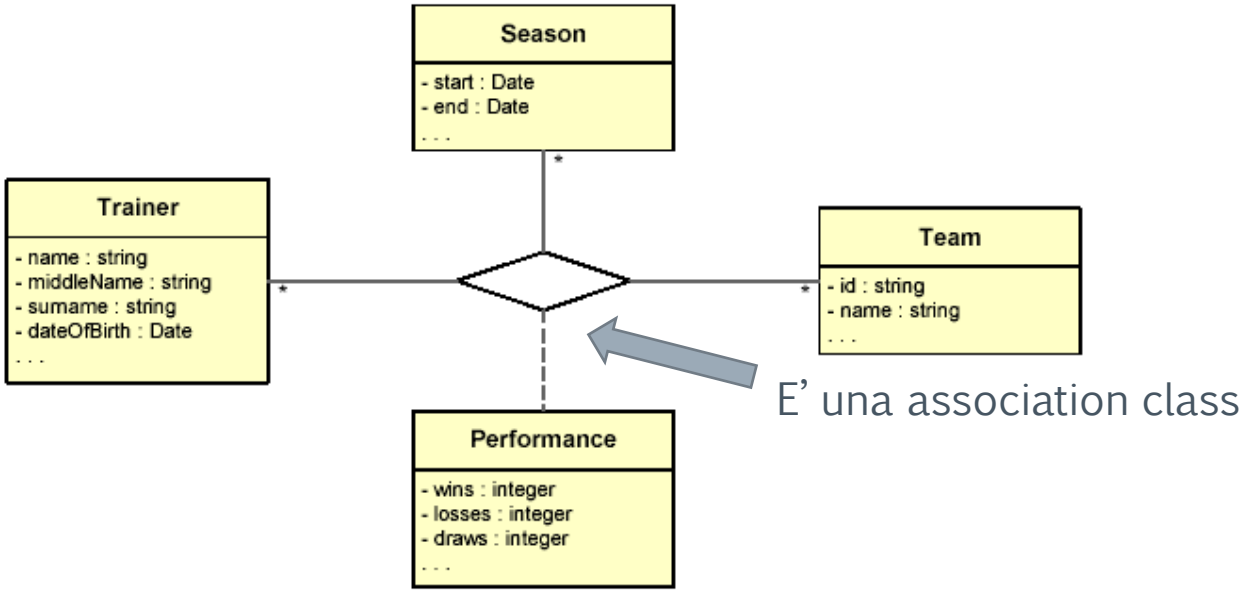
Java

```
public class Company {
      public Job[] employees;
}
public class Person {
      public Job[] employeers;
}
public class Job {
      private String description;
      private Date dateHired;
      private double salary;
      private Person employee;
      private Company employeer;
}
```

Associazione n-aria (1)

- Relazione che coinvolge più di due classi
- Rappresentata mediante un rombo dove sono collegate le classi appartenenti alla relazione
- Difficoltà nell'attribuire i valori delle molteplicità. Specificano, per ogni classe, il potenziale numero di istanze della classe che possono partecipare nella relazione, fissando i valori delle altre n–1 classi

Associazione n-aria (2)

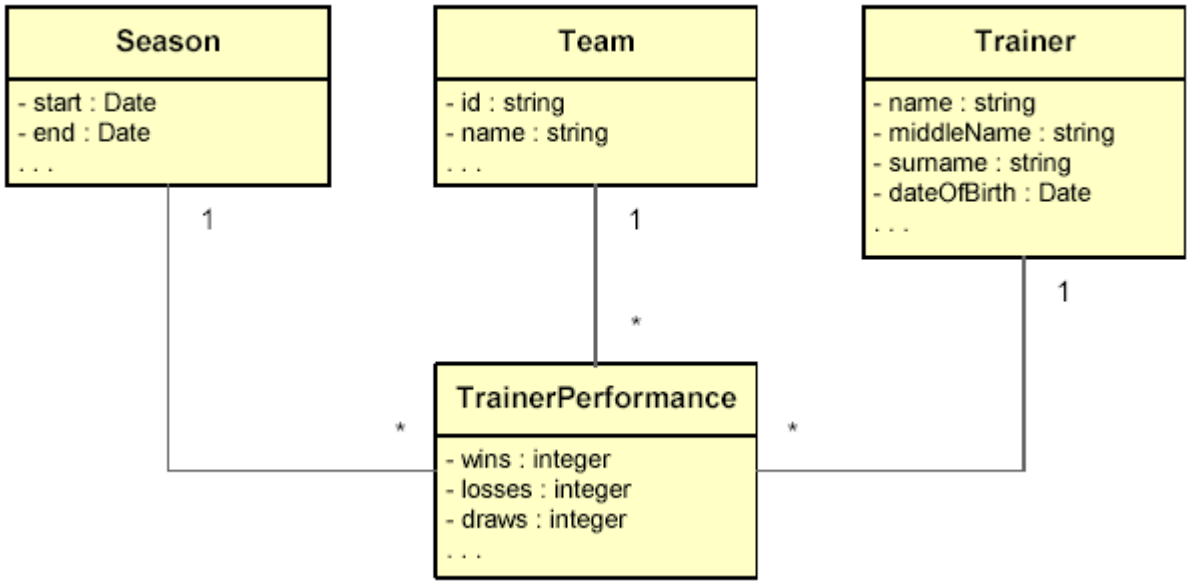


Associazione n-aria (3)

› Java

```
public class Season {
	public Performance[] performances;
}
public class Team {
	public Performance[] performances;
}
public class Trainer {
	public Performance[] performances;
}
public class Performance {
	public Season season;
	public Team team;
	public Trainer trainer;
}
```

Associazione n-aria (4)



Trasformazione di un'associazione n-aria in **associazioni binarie**

Associazione n-aria (5)

› Java

```
public class Season {
	public TrainerPerformance[] trainerperformances;
}
public class Team {
	public TrainerPerformance[] trainerperformances;
}
public class Trainer {
	public TrainerPerformance[] trainerperformances;
}
public class TrainerPerformance {
	public Season season;
	public Team team;
	public Trainer trainer;
}
```

Associazione: Aggregazione (1)

- Associazione tra classi mostra una relazione strutturale paritetica (**peer-to-peer**)
- Non è possibile distinguere una classe concettualmente più importante dell'altra (o altre). Sono tutte allo stesso livello
- Esiste necessità di modellare situazioni in cui una classe esprime una nozione concettualmente più grande delle altre che la costituiscono
- Rappresenta una relazione has-a, consists-of, contains, is-part-of

Associazione: Aggregazione (2)

- E' necessario utilizzare relazioni del tipo ***"tutto–parte"*** (whole–part), in cui esiste una classe che rappresenta il concetto "più grande" (il tutto) costituito dalle restanti che rappresentano i concetti più piccoli (le parti)
- Tali relazioni sono dette Aggregazione
- Rappresentata con una linea che connette gli oggetti in relazione utilizzando un diamante posto vicino alla classe completa

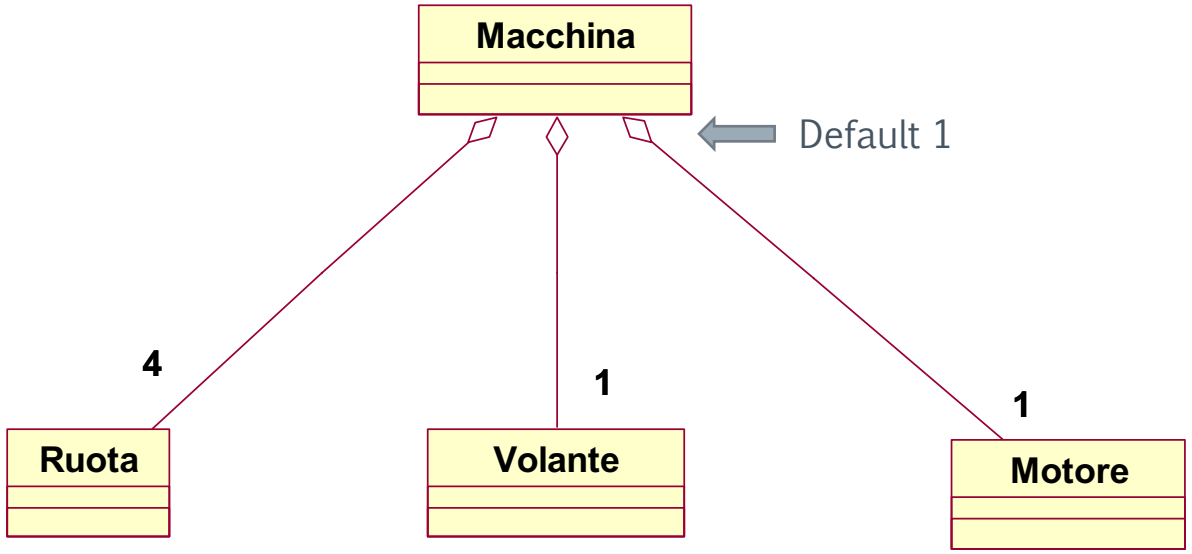
Associazione: Aggregazione (3)

› Esempio
  – Automobile composta di Ruote, Motore, Volante, Sedili, ...

› Differenza rispetto ad Associazione **puramente concettuale**

› Non hanno senso **aggregazioni circolari**
  – Classe A composta da una Classe B; B composta da una Classe C a sua volta composta dalla Classe A

Associazione: Aggregazione (4)



Associazione: Aggregazione (5)

› Java

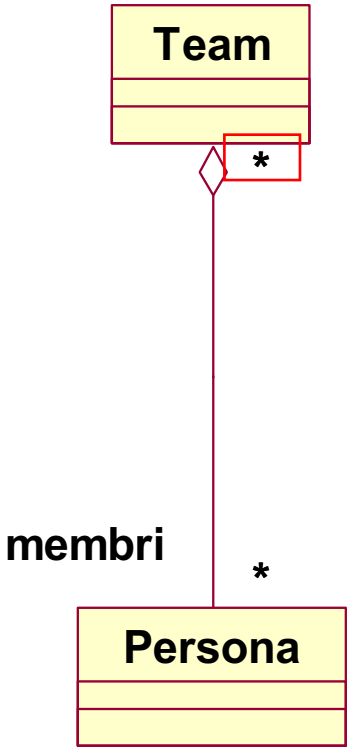
```
public class Automobile {
      private Ruota ruotaAnterioreDestra;
      private Ruota ruotaAnterioreSinistra;
      private Ruota ruotaPosterioreDestra;
      private Ruota ruotaPosterioreSinistra;
      private Volante volante;
      private Motore motore;
}

public class Ruota {}
public class Volante {}
public class Motore {}
```

Associazione: Aggregazione (6)

- Caso speciale della normale aggregazione
- La classe parte può essere parte di qualsiasi intero
- E' **shared** se la molteplicità nella parte intera è maggiore di uno
- In termini implementativi in Java non cambia nulla tranne che oggetti di `Persona` possono essere collegati ad oggetti `Team`

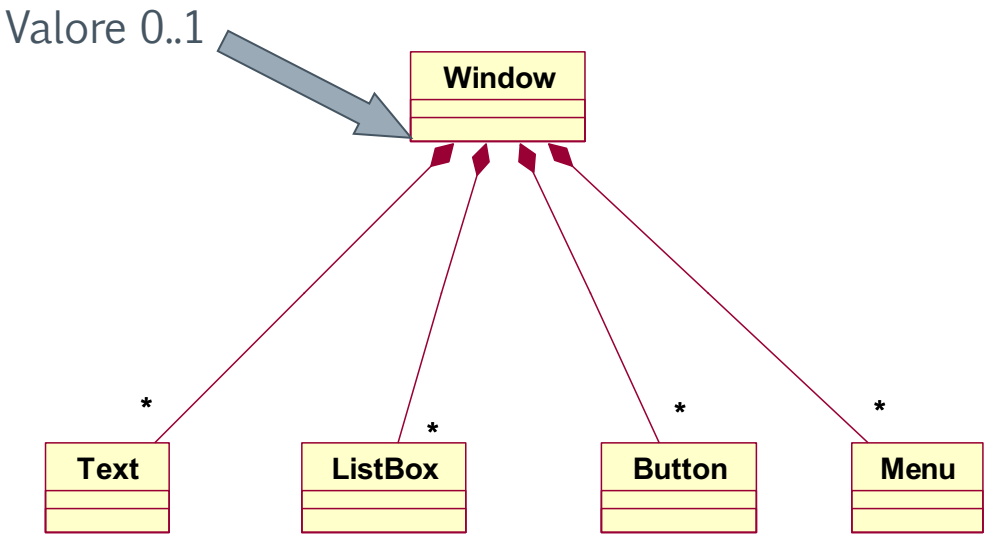
```
Persona p = new Persona();
Team t1 = new Team();
Team t2 = new Team();
t1.setPersona(p);
t2.setPersona(p);
```



Associazione: Composizione (1)

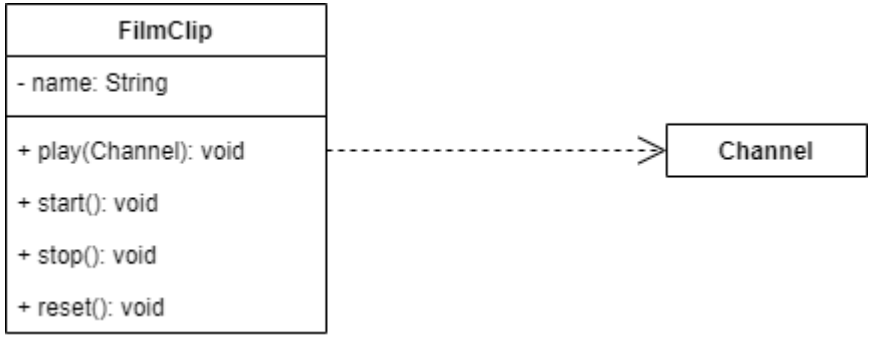
- Forma di aggregazione con una forte connotazione di possesso e una (quasi) coincidenza del ciclo di vita tra istanze delle classi “parte” e l’istanza della classe “tutto” (classe composta)
- Parti possono essere generate anche in un tempo successivo alla creazione dell’istanza della classe composta, ma una volta generate queste vivono e sono distrutte con l’istanza della classe composta di appartenenza
- Classe composta si occupa di eliminare le istanze proprie parti in un momento antecedente alla propria distruzione
- Un oggetto può essere parte soltanto di un oggetto composto alla volta
  - Esempio: Un `Frame` appartiene ad una sola `Window`
- Invece in un’aggregazione una parte può essere condivisa tra più composte
  - Esempio: Un `Muro` può appartenere a diverse `Stanza`

Associazione: Composizione (2)



Dipendenza

- E' una relazione tra due elementi dove il cambiamento di un elemento può causare il cambiamento nell'altro, ma non necessariamente il viceversa
- Generalmente è rappresentata nel contesto delle classi per mostrare che una classe utilizza un'altra classe come **argomento** nella **segnatura** di un'operazione
- Rappresentata con una linea tratteggiata e generalmente con una freccia



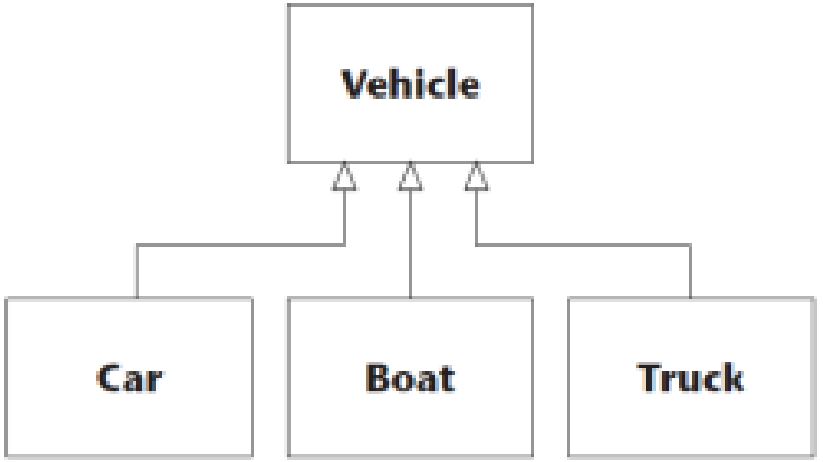
Associazione vs Dipendenza

- Associazione è una relazione **strutturale** (quindi “persistente”), che evidenzia classi semanticamente correlate
- Dipendenza ha un carattere **transitorio**, un legame che si instaura (o almeno così dovrebbe essere) temporaneamente, per il lasso di tempo necessario per fruire di un servizio, per creare un oggetto, ecc., per poi perdere di significato

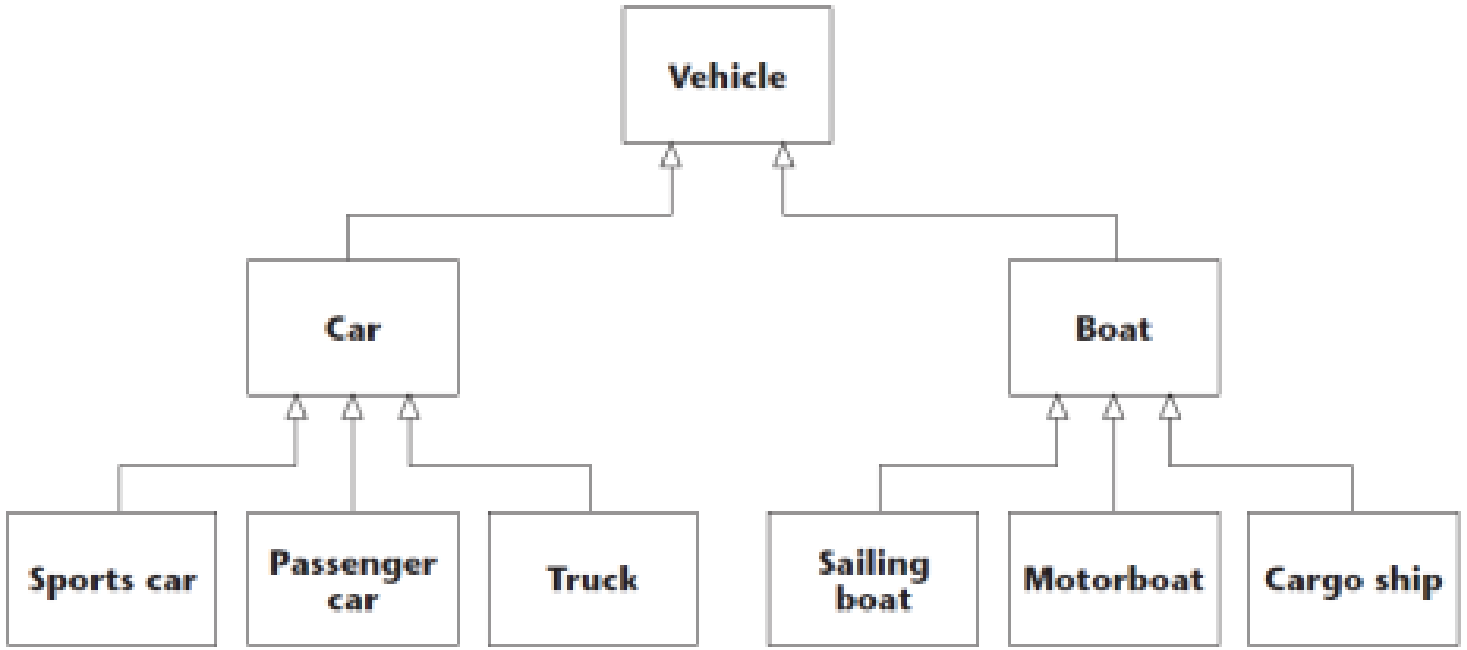
Generalizzazione (1)

- E’ una relazione tra una cosa più generale (detta superclasse o padre) ed una più specifica (detta sottoclasse o figlia)
- Oggetti figlio possono essere utilizzati al posto di oggetti padre (Liskov) ma non il viceversa, cioè il padre non è un sostituto per il figlio
- Disegnata con una freccia chiusa diretta verso il padre

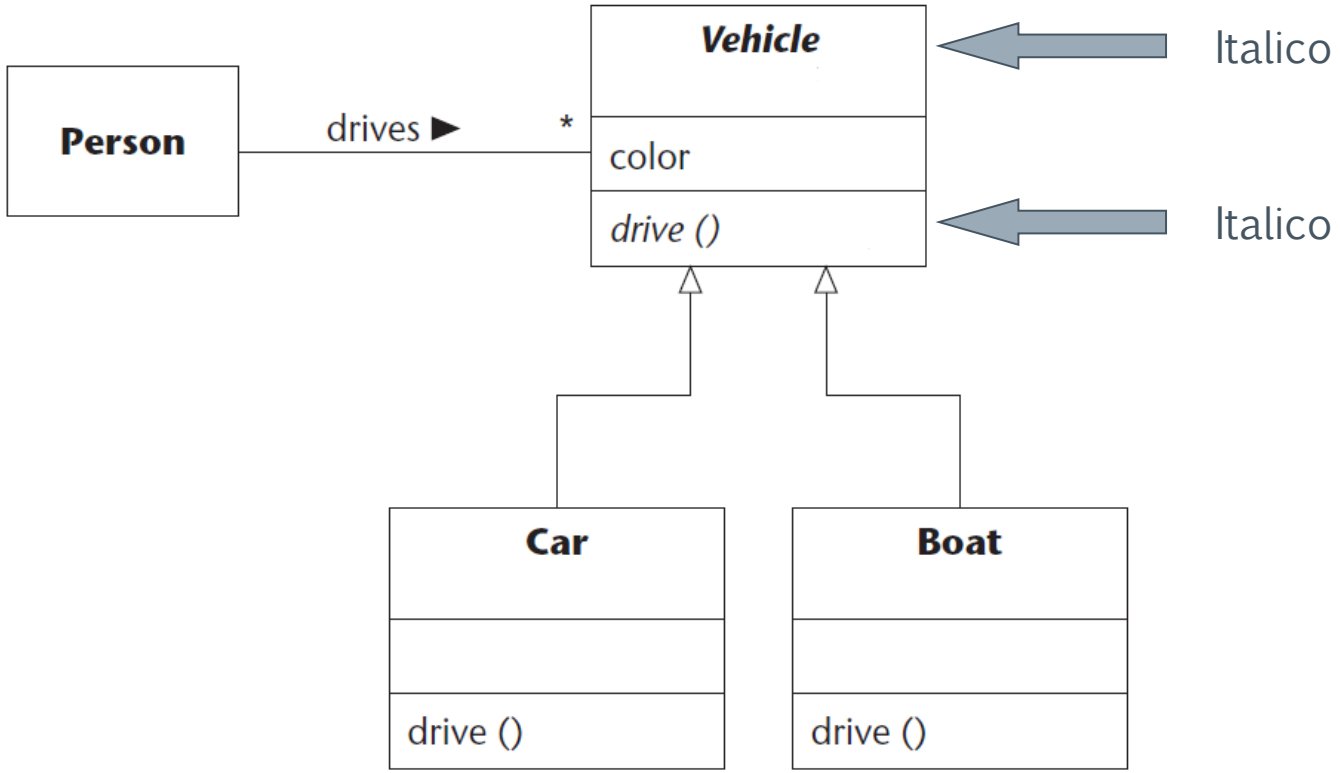
Generalizzazione (2)



Generalizzazione (3)



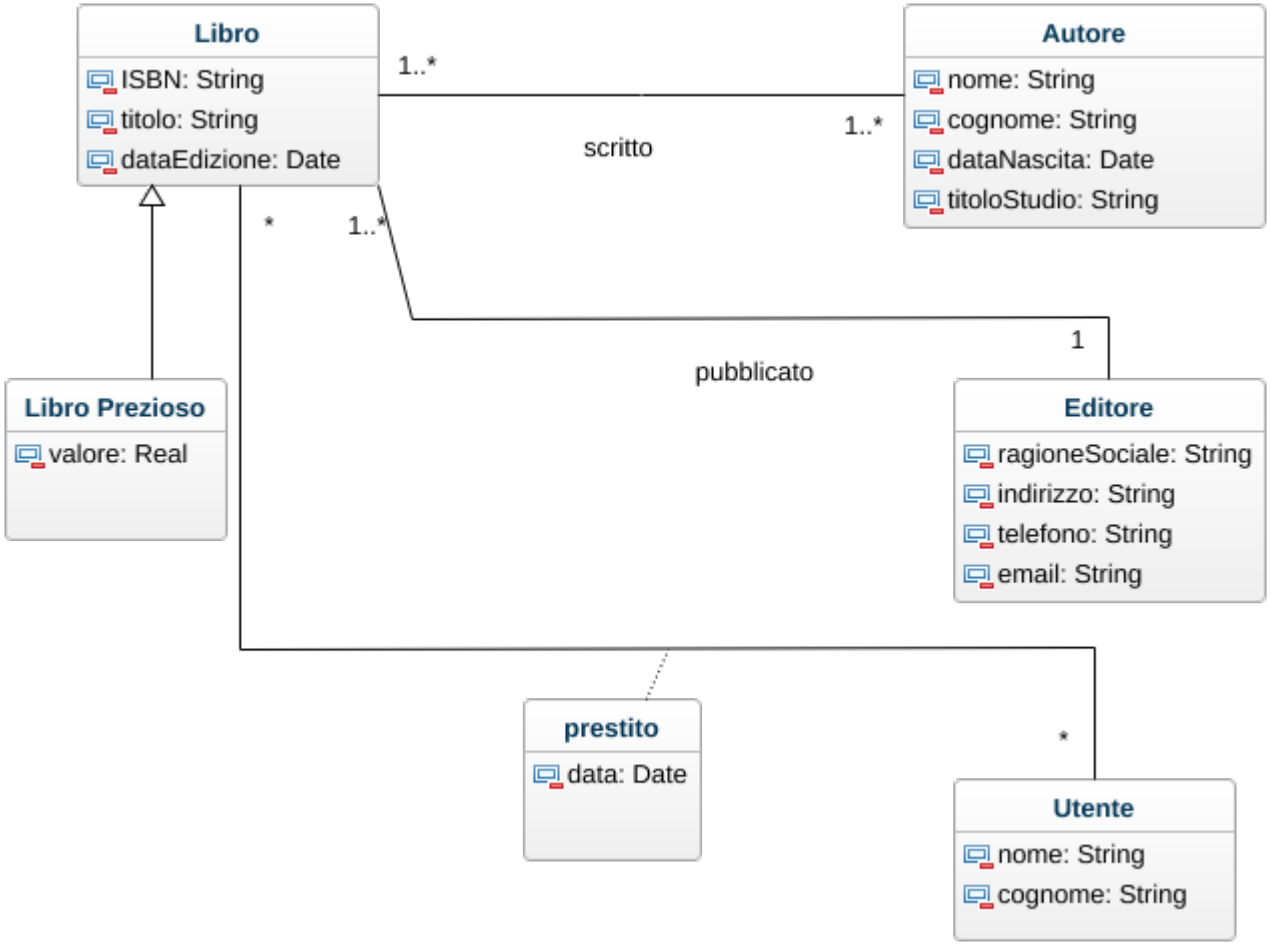
Classe Astratta



Esempio (1)

› Sistema per l'archiviazione bibliografica di **testi**, memorizzando informazioni sull'**editore** e sull'**autore** e identificando quelli che sono considerati i libri **preziosi**. Si vuol inoltre gestire un'operazione di **prestito** da parte di uno o più **utenti**

Esempio (2)


Libro
ISBN: String
titolo: String
dataEdizione: Date
1..*
scritto
1..*
Autore
nome: String
cognome: String
dataNascita: Date
titoloStudio: String
*
1..*
1
pubblicato
Libro Prezioso
valore: Real
Editore
ragioneSociale: String
indirizzo: String
telefono: String
email: String
prestito
data: Date
*
Utente
nome: String
cognome: String