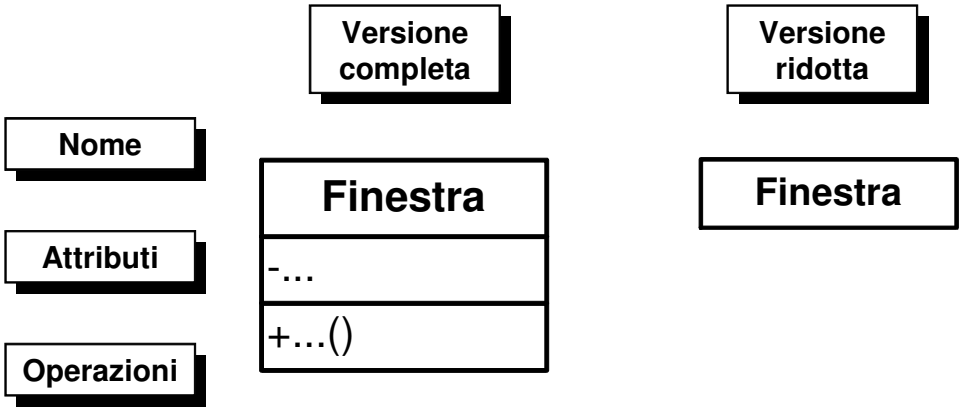
UML
Diagrammi delle classi

Diagramma delle classi

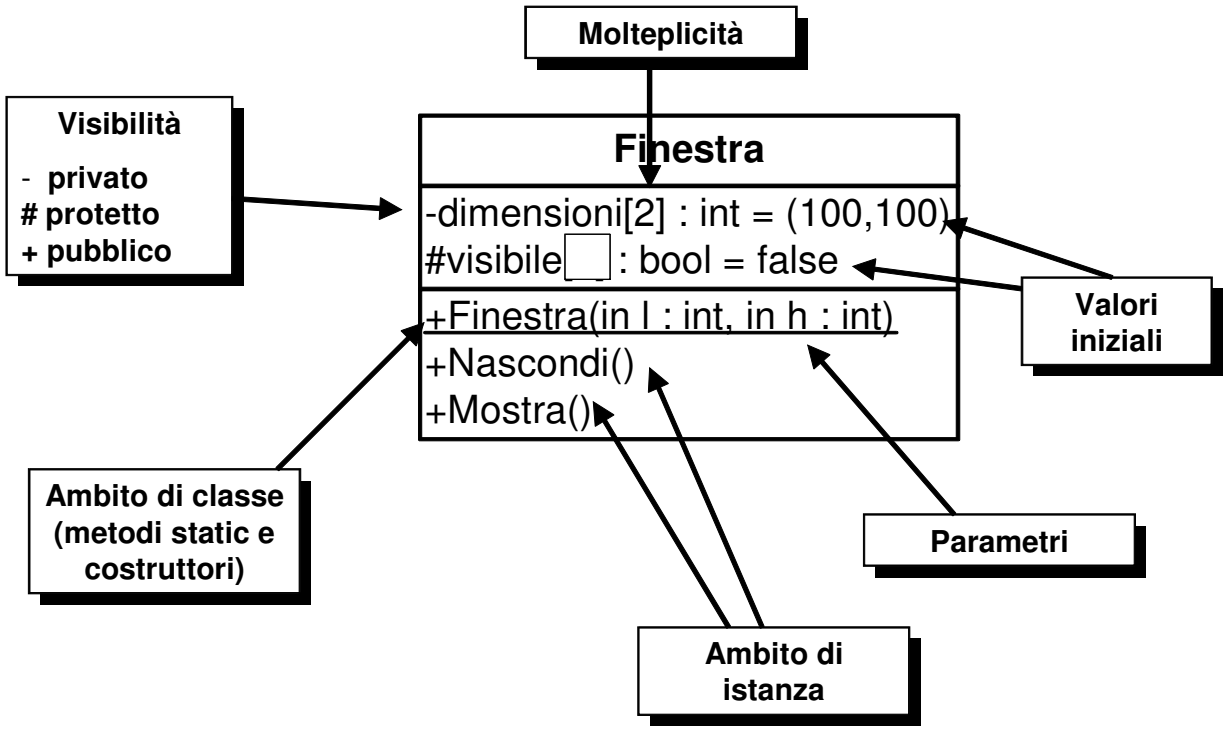
- Non è nei nostri obiettivi affrontare UML nel suo complesso
- Ci concentreremo sui diagrammi delle classi che ci forniscono un linguaggio sufficientemente ricco per descrivere i concetti che ci servono.
- Passiamo in rassegna i vari elementi che possono comparire in questo tipo di diagramma
- Cercheremo di riportare il codice Java corrispondente ai vari elementi grafici

Classi

- **In UML una classe è rappresentata da un rettangolo diviso in 3 sottosezioni: nome, attributi e operazioni**
- **Solo il nome è obbligatorio, le altre sezioni possono essere definite in fasi successive**



Classi: dettagli della notazione



Attributi

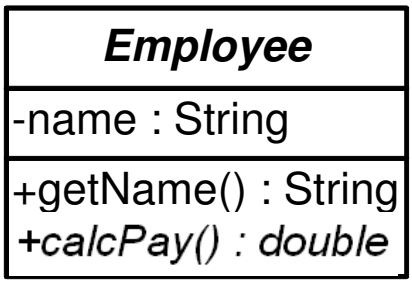
- **Gli attributi hanno una struttura di questo tipo:**
  `visibilità nome molteplicità: tipo = valoreIniziale`
- **Solo il nome è obbligatorio**
- **La visibilità può essere:**
  - **private (-)**
  - **protected (#)**
  - **package (~)**
  - **public (+)**
- **La molteplicità si indica con la notazione tipica degli array: [n]**
- **Un'eventuale sottolineatura indica che l'attributo appartiene ad un ambito di classe (static) e non di istanza**
- **E' possibile indicare un valore iniziale**

Operazioni

- **Le operazioni sono i metodi della classe e hanno una struttura del tipo:**
  `visib nome(nomeParam: tipoParam, …): tipoRestituito`
- **Anche per le operazioni solo il nome è obbligatorio**
- **Le visibilità sono uguali a quelle degli attributi**
- **La sottolineatura indica metodi static e costruttori**
- **I parametri possono essere preceduti da un modificatore che indica la "direzione" di uso:**
  - **in: parametro in ingresso (per valore)**
  - **out: parametro di uscita (serve per restituire un risultato: per riferimento)**
  - **inout: parametro di ingresso e uscita (per riferimento)**

Classi e metodi astratti

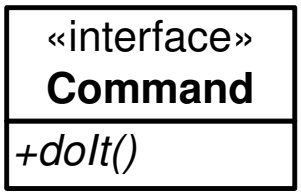
- **Le entità astratte vengono rappresentate in corsivo: questo vale sia per le classi che per i metodi:**




```
abstract class Employee
{
  public getName() { return name; }
  public abstract double calcPay();
  private String name;
}
```


Interfacce

- **Le interfacce hanno rappresentazione simile a quella delle classi**
- **Vengono identificate tramite lo stereotipo «interface»**
- **Manca la sezione degli attributi**
- **I metodi sono in corsivo (astratti)**
- **Esiste anche una rappresentazione grafica compatta costituita da un tondino e da una linea**



Versione estesa

```
interface Command
{
  void doIt ();
}
```

Command

Versione compatta

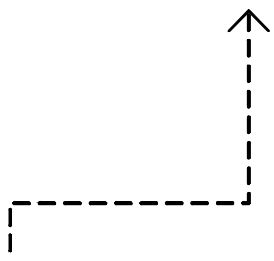
Relazioni

- **In UML una relazione è definita come:**

  "Una connessione semanticamente significativa tra elementi di modellazione"
- **Nei diagrammi delle classi troviamo 3 tipi di relazioni:**
  - **Dipendenze (relazioni d'uso)**
  - **Associazioni (associazione semplice, aggregazione, composizione)**
  - **Generalizzazioni e realizzazioni (ereditarietà fra classi e implementazione di interfacce)**

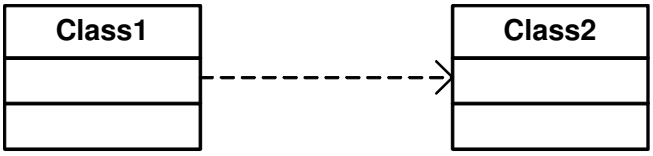
Dipendenze

- **"Una dipendenza è una relazione tra due elementi dove un cambiamento ad uno di essi (fornitore) può influenzare o fornire informazioni necessarie all'altro (cliente)"**
- **Il tipo più comune è la relazione d'uso che esprime un rapporto client-server fra due classi o fra una classe e un'interfaccia**
- **Si rappresenta con una linea tratteggiata e con lo stereotipo «uses» (che di solito però viene omesso)**



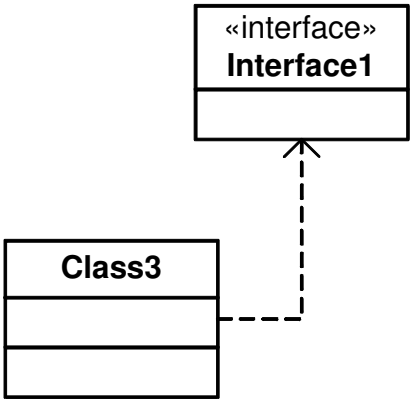
Relazione d'uso tra classi

- **Una relazione d'uso fra due classi si ha nei seguenti 3 casi:**
  - **Un metodo di Class1 ha un parametro di tipo Class2**
  - **Un metodo di Class1 restituisce un valore di tipo Class2**
  - **Un metodo di Class1 usa un oggetto di tipo Class2 ma non come attributo.**
- **L'esempio più comune del terzo caso si ha quando in un metodo di Class1 si dichiara una variabile locale di tipo Class2**



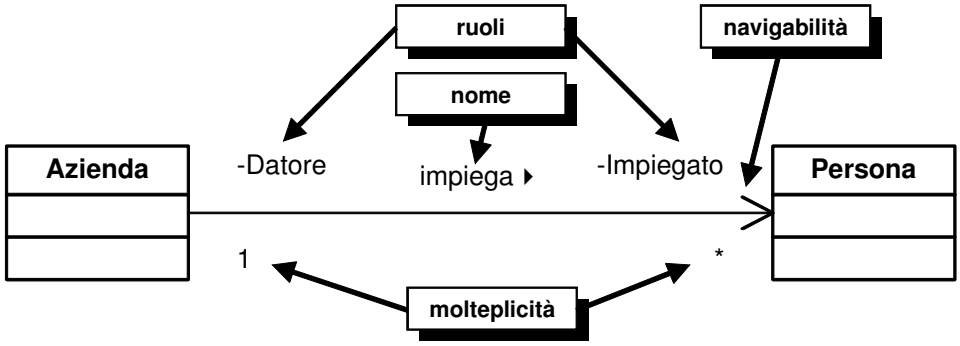
Relazione d'uso fra una classe e un'interfaccia

- **Nel caso di una classe e di un'interfaccia la relazione d'uso ha un significato più generico**
- **Indica semplicemente che la classe invoca uno o più metodi definiti nell'interfaccia**
- **Talvolta si parla di relazione client-of**



Associazioni

- **Un'associazione è una relazione fra classi**
- **Può essere caratterizzata da:**
  - **Nome**
  - **Nomi dei ruoli**
  - **Molteplicità**
  - **Navigabilità**

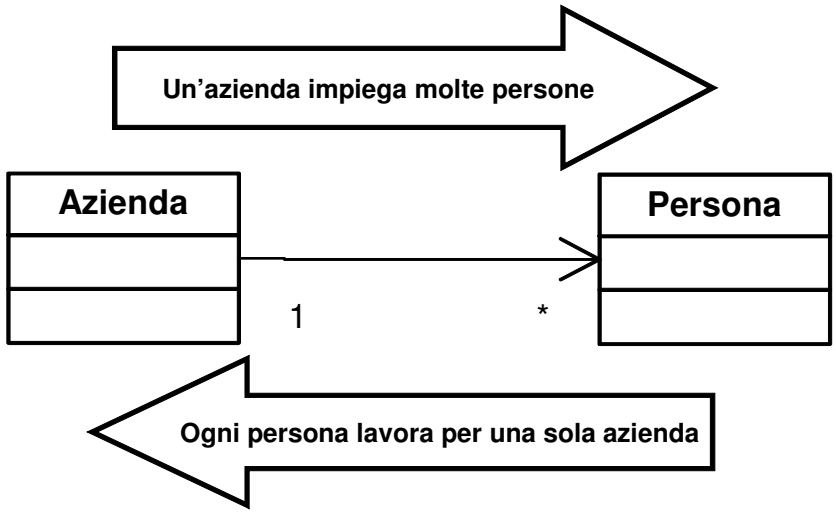


Nomi e ruoli

- **Un'associazione può essere indicata con un nome o semplicemente con i nomi dei ruoli**
- **Sia nome che ruoli sono facoltativi**
- **La freccetta accanto al nome indica la direzione di lettura del nome:**
- **Il termine "impiega" ha senso se si legge l'associazione da destra a sinistra**
- **Si potrebbe invertire la direzione di lettura cambiando anche il nome dell'associazione in "è impiegato"**

Molteplicità

- **La molteplicità è un vincolo la cui funzione è qualla di limitare il numero di oggetti di una classe che possono partecipare ad un'associazione in un dato istante**



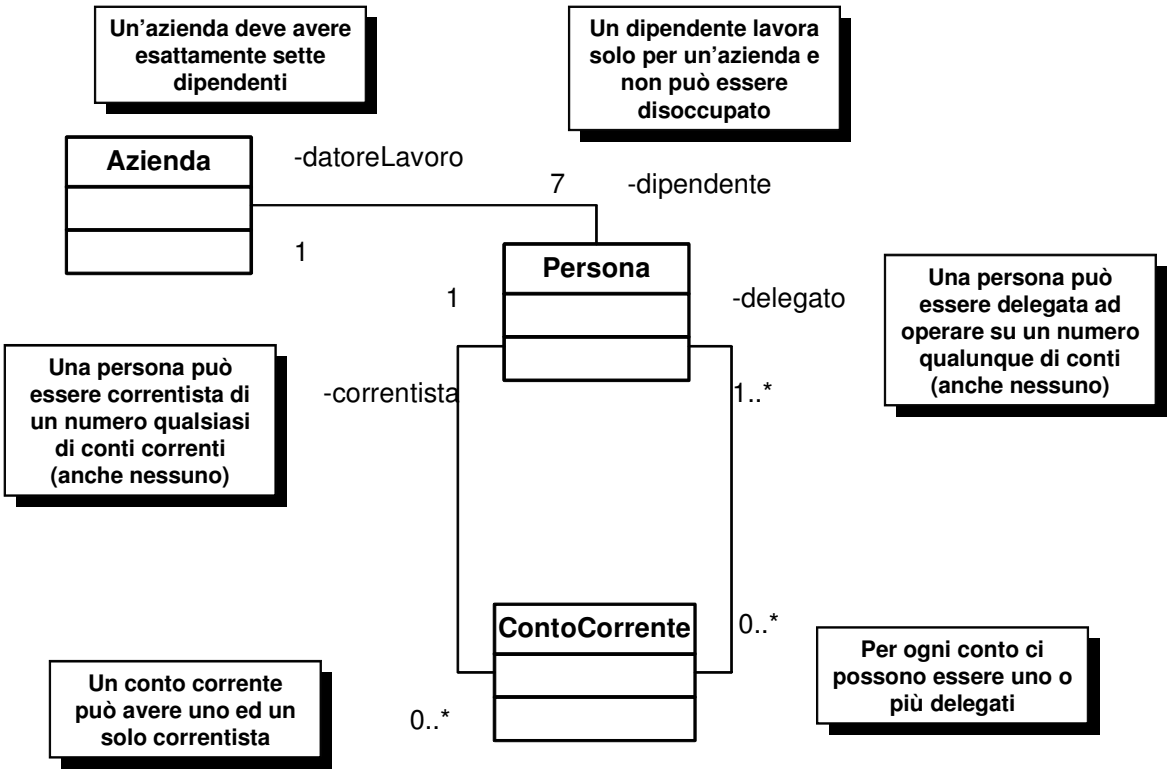
Molteplicità: attenzione all'interpretazione

- **La presenza del numero 1 (ornamento) vicino alla classe Azienda vuol dire due cose:**
  - **Una persona può lavorare solo per un'azienda per volta**
  - **Una persona deve per forza lavorare per un'azienda**
- **Quindi:**
  - **E' ammesso che una persona lavori per più aziende in tempi diversi**
  - **Se vogliamo che una persona possa essere disoccupata dovremo scrivere 0..1 anziché 1**

Espressioni di molteplicità

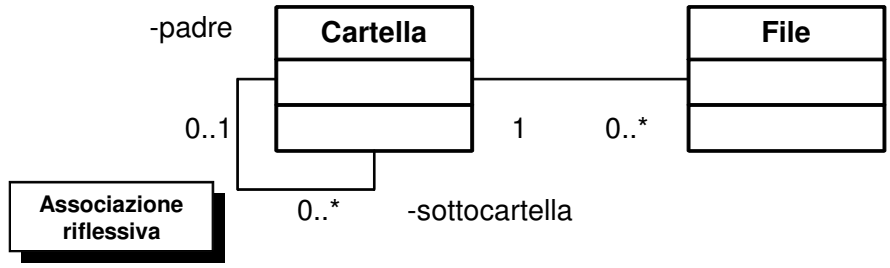
- La molteplicità può essere espressa:
  - Con un solo simbolo (0 1 *)
  - Con un intervallo (0..2 1..*)
  - Con una lista di simboli o intervalli separati da virgole (0..3,5,6..9)
- Esempi:
  - 0..1 Zero o uno
  - 1 Esattamente uno
  - 0..* Zero o più
  - * Zero o più (attenzione!)
  - 1..* Uno o più
  - 1..6 Da uno a 6
  - 1..3,7,19..* Da 1 a 3, oppure 7 oppure da 19 in su

Esempio di molteplicità



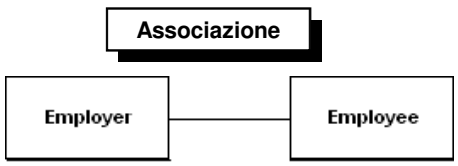
Associazioni riflessive

- **E' piuttosto comune che una classe abbia un'associazione con se stessa**
- **Ciò significa che oggetti della classe possono avere collegamenti con altri oggetti della stessa classe**
- **L'esempio tipico è quello di una struttura ad albero (per esempio un file system): una cartella può essere collegata a file o a cartelle figlie**

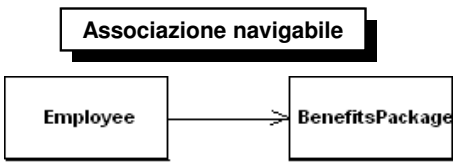


Navigabilità

- **La navigabilità, indicata con una punta di freccia su un lato dell'associazione, indica una direzione nella relazione**
- **Specificando la navigabilità si riduce la dipendenza fra le classi: i legami bidirezionali sono critici**




```
class Employer
{
  // ...
  private Employee itsWorker;
}

class Employee
{
  // ...
  private Employer itsBoss;
}
```



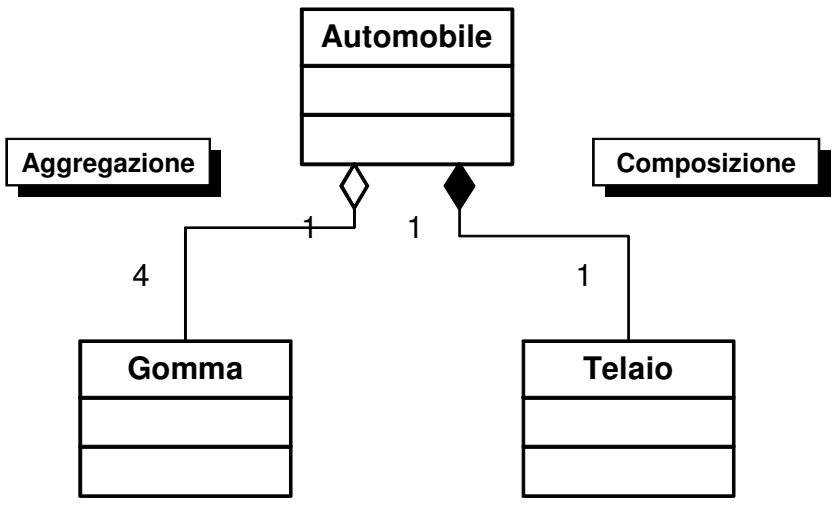

```
class Employee
{
  // ...
  private BenefitsPackage itsBenefits;
}

class BenefitsPackage
{
  // ... Non c'è un riferimento
    all'immpiegato
}
```

Aggregazione e composizione

- Oltre all'associazione semplice di cui abbiamo parlato esistono altre due relazioni di tipo analogo: l'aggregazione e la composizione
- Entrambe sono relazioni part-of, in cui un oggetto di una classe diventa parte di un oggetto dell'altra
- Rispetto all'aggregazione nella composizione ci sono dei vincoli in più:
  - La parte può appartenere a un solo oggetto (vincolo di esclusività)
  - La parte nasce e muore assieme all'oggetto che la contiene (vincolo sulla gestione del tempo di vita).
- Aggregazione e composizione vengono rappresentate come associazioni con un rombo (rispettivamente bianco e nero) vicino alla classe contenitore

Esempio su aggregazione e composizione

- Un esempio classico per comprendere la differenza è quello dell'automobile: sia il telaio che le gomme sono parti dell'auto
- Ma la relazione con il telaio è più stretta: il telaio può appartenere ad una sola auto e nasce e muore assieme all'auto



Esempio su aggregazione e composizione – 2

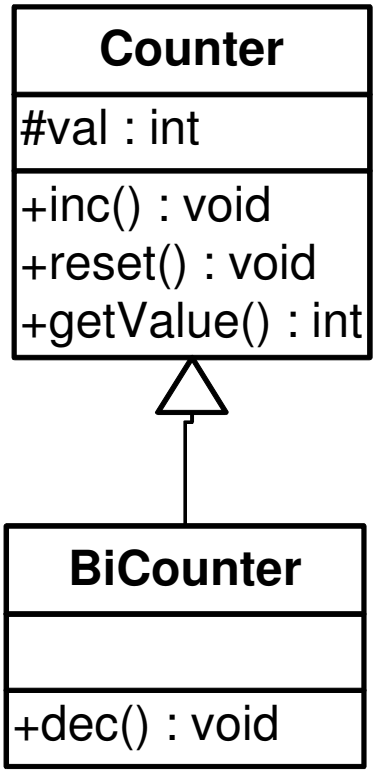
- **Vediamo l'implementazione della classe Automobile:**

```
class Automobile
{
   private Gomma gomme[];
   private Telaio telaio;
   public Automobile()
   {
     telaio = new Telaio();
     ...
   }
   public Gomma getGomma(int n)
   { return Gomme[n]; }
   public void setGomma(Gomma g; int n)
   { gomme[n] = g; }
}
```

Gestione tempo di vita

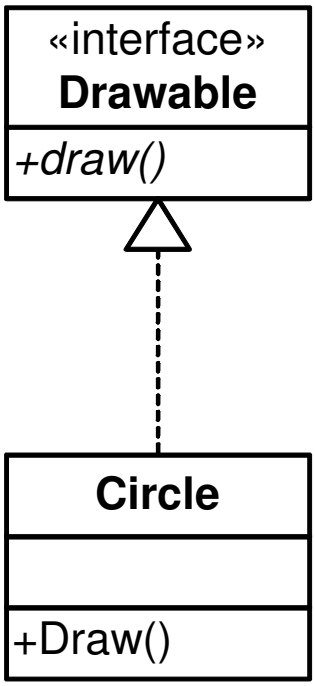
Non c'è setTelaio(): non è possibile cambiare il telaio di un auto

Ereditarietà (Generalization)




```
public class Counter
{
  protected int val;
  public void reset()
  { val = 0; }
  public void inc()
  { val++; }
  public int getValue()
  { return val;}
}

public class BiCounter
 extends Counter
{
 public void dec()
 { val--; }
}
```

Implementazione di interfacce (Realization)




```
interface Drawable
{
  public void draw();
}

class Circle
  implements Drawable
{
  public void draw()
  {
     ...
  }
 ...
}
```