Relazioni tra classi

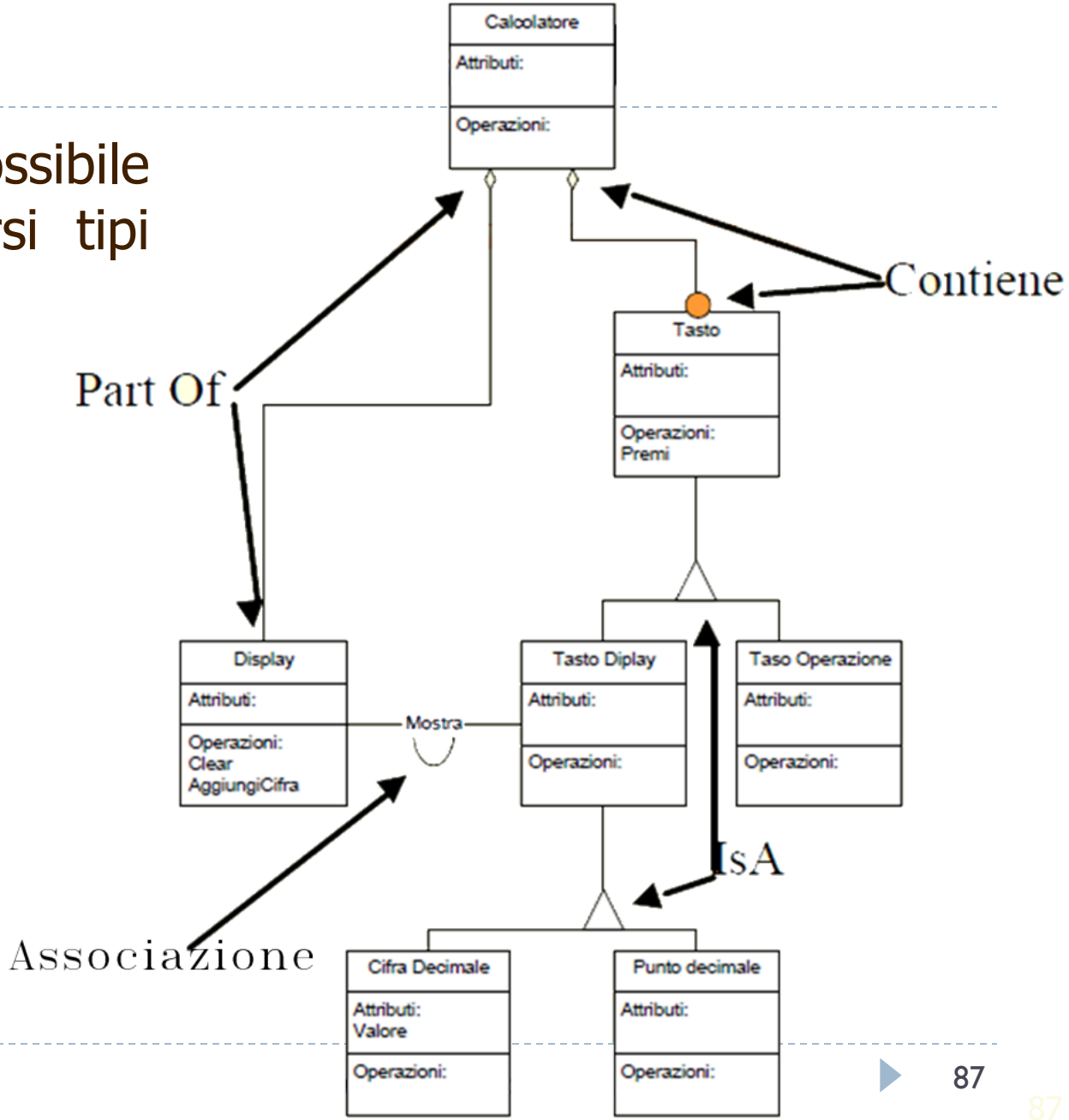
Relazioni tra classi

- **Generalizzazioni o specializzazione**
- **Aggregazioni**
- **Composizioni**
- **Dipendenza**
- **Associazioni**

86

Relazioni tra classi

- Tra le classi è possibile individuare diversi tipi di relazione



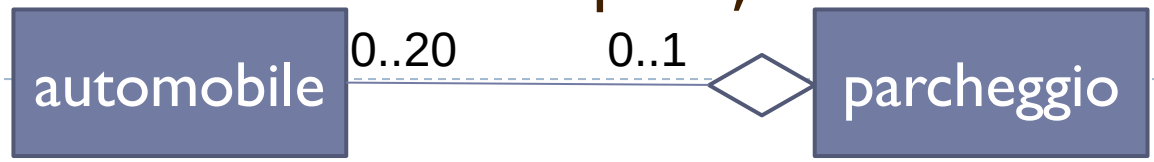
Relazioni tra classi
Generalizzazione-Specializzazione

- È una associazione di tipo **IS-A** (in UML si usa una freccia continua)
- Si basa sul concetto di **ereditarietà**: un oggetto di classe A deriva da un oggetto di classe B se A è in grado di compiere tutte le azioni che l'oggetto B è in grado di compiere.
- In più A è una specializzazione e B è una generalizzazione se l'oggetto di classe A è in grado di eseguire anche azioni che l'oggetto B non può compiere.

88

Relazioni tra classi
Aggregazione (lasca)

- È un associazione più forte, di tipo "intero-parte". Indica "contiene", **"è parte di"**, "è un insieme di".
- È anche sinonimo di (possibile) condivisione; semantica del contenimento "by reference" (puntatori) (vettori).
  - Un oggetto di classe A contiene un oggetto di classe B se B è una proprietà (attributo) di A (c'è un riferimento a B)
- Gli oggetti contenuti (parti) possono appartenere a più di un oggetto contenitore oppure possono esistere indipendentemente dal contenitore.
- L'aggregato può in alcuni casi esistere indipendentemente dalle parti, ma in altri casi no.



89

Relazioni tra classi
Aggregazione (lasca)

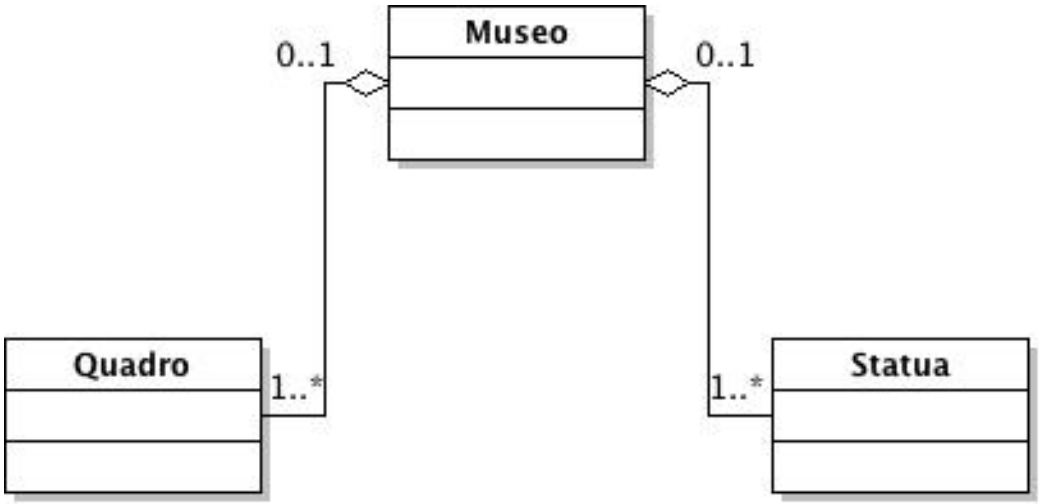
Domande per capire se è un'aggregazione lasca:

- L'espressione "è-parte-di" ("part of") "è formata da" viene usata per descrivere questa relazione?
  - A door is "part of" a car
- Alcune operazioni del "tutto" possono essere applicate alle singole "parti"?
  - Move the car, move the door.
- Esistono particolari valori di attributi che possono essere propagati dal "tutto" alle "parti"?
  - The car is blue, therefore the door is blue.
- Esiste un'intrinseca asimmetria nella relazione per cui una classe è subordinata ad un'altra?
  - A door is part of a car. A car is not part of a door.

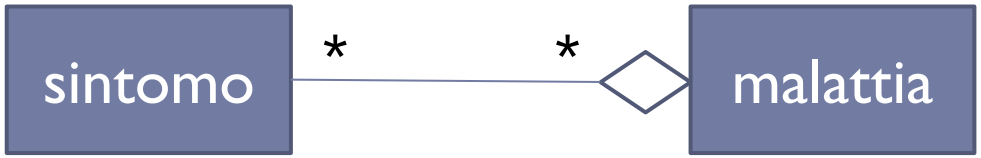
90

Relazioni tra classi
Aggregazione (lasca)

Esempi: Un museo contiene più quadri e più statue. Sia il quadro che la statua continuano ad avere dignità ed esistenza propria (esistono) anche senza un museo che li contenga. La distruzione del museo, non comporta automaticamente quella delle sue parti …



Un sintomo può appartenere a più malattie



91

Relazioni tra classi
Composizione (aggregazione stretta)

- È una forma di aggregazione ancora più forte (**HAS–A**) che indica che una "parte" può appartenere ad un solo "intero" in un certo istante di tempo, la parte non può esistere di per sé. (record o vettori)
- La composizione associa composto e componente per tutta la vita dei due oggetti
- La composizione è esclusiva: uno specifico oggetto componente non può appartenere a due composti contemporaneamente

92

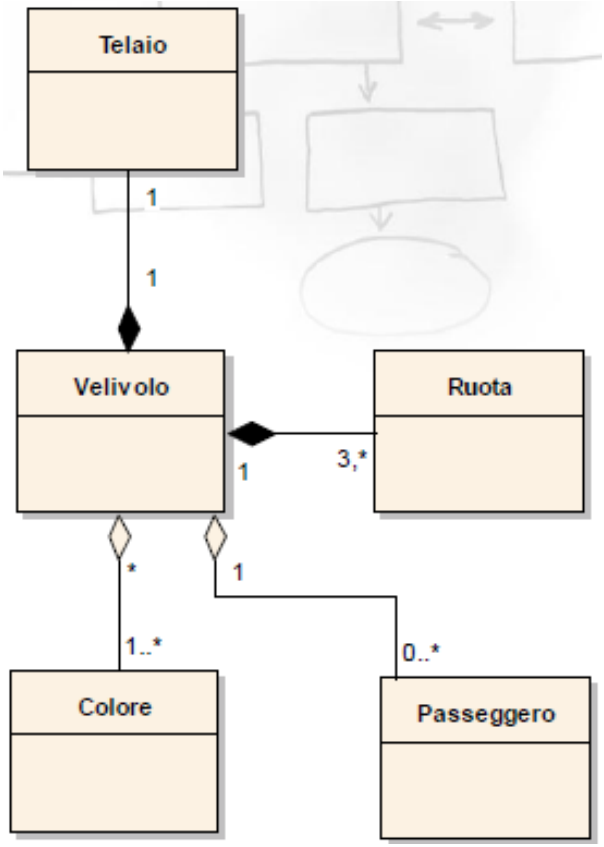
Relazioni tra classi
Composizione (aggregazione stretta)

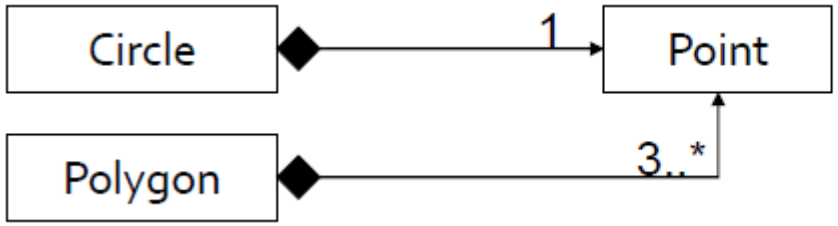
- Il "tutto", detto "composto", è il solo proprietario (owner) delle proprie parti, dette "componenti"
  - L'oggetto "parte" appartiene al più ad un unico "tutto".
  - La molteplicità dalla parte del "tutto" può essere solo 0 oppure 1
- Il tempo di vita (life-time) della "parte" è subordinato (minore o uguale) a quello del "tutto" che ne è responsabile
  - L'oggetto "composto" gestisce la creazione e la distruzione delle sue parti (ownership completa)
  - Le parti esistono solo all'interno dell'intero, e se l'intero è distrutto anche le parti muoiono. La composizione governa il periodo di vita delle classi parti.

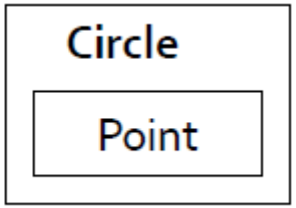
93

Relazioni tra classi
Composizione (aggregazione stretta)

- Ad esempio, un capitolo può far parte di un solo libro, mentre, al contrario, una persona potrebbe lavorare contemporaneamente per due ditte o ancora un modello di pneumatico potrebbe essere utilizzato su più modelli di auto, ma quel veicolo ha montate solo quelle particolari ruote fisiche







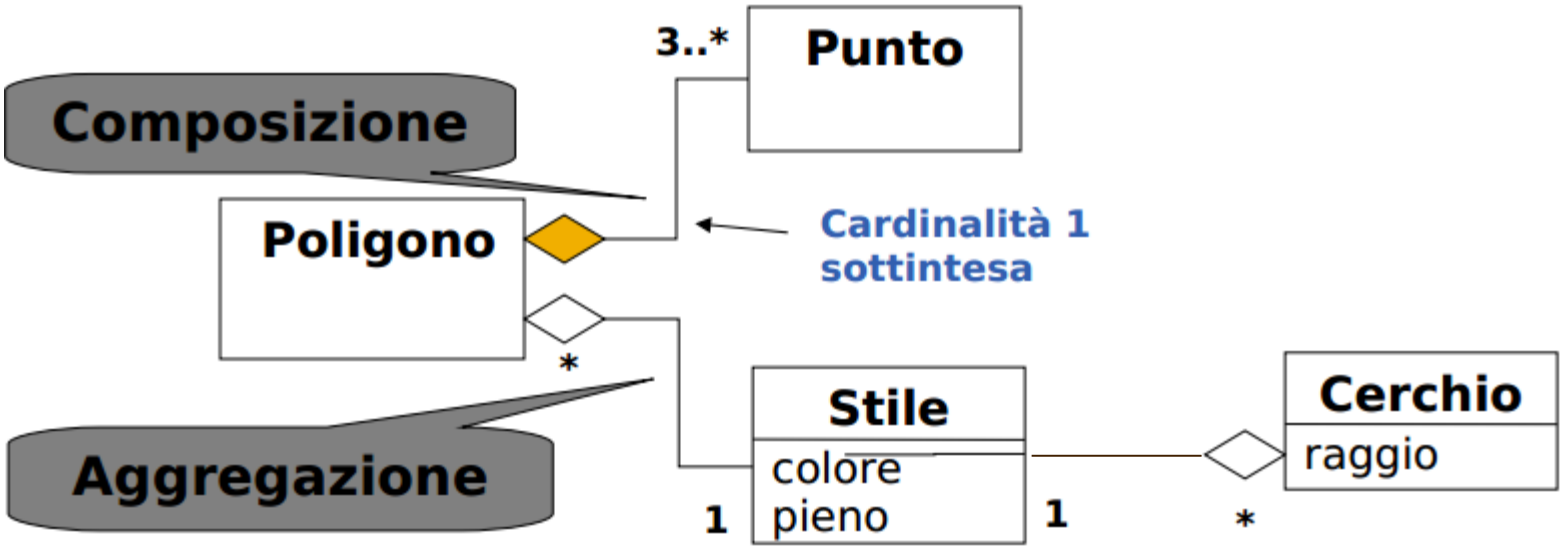
94

Relazioni tra classi
Aggregazione o Composizione?

- L'oggetto componente/parte ha senso di esistere, ha uno scopo, se l'oggetto composto/tutto non esiste?
  - Sì: aggregazione
  - No: composizione
- Se si elimina oggetto composto/tutto si eliminano anche le parti?
  - Sì: composizione
  - No: aggregazione
- L'oggetto componente/parte può appartenere a più composti/tutti?
  - Sì: aggregazione
  - No: composizione

95

Relazioni tra classi
Aggregazione o Composizione?



- La cancellazione di una istanza della classe Poligono viene estesa ad ogni suo Punto, ma non allo Stile ad esso associato
- Un'istanza della classe Stile può, invece, essere condivisa tra Poligono e Cerchio

96

Relazioni tra classi
Derivazione o Composizione?

- È un'associazione 1 a 1?
  - No: non può essere una derivazione

97

Relazioni tra classi
Dipendenza

- Rappresenta una relazione tra due classi, nonostante non ci sia un'esplicita associazione tra esse: una classe **dipende** da un'altra quando esiste un riferimento della seconda nella prima (non è possibile compilare la prima senza la seconda). Per esempio:
  - Una classe "nomina" al suo interno una classe esterna
  - Passaggio di parametri all'interno di un metodo
  - Tipo di ritorno di un metodo
  - Creazione o distruzione
  - Eccezioni
  - 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

98

Relazioni tra classi
Dipendenza

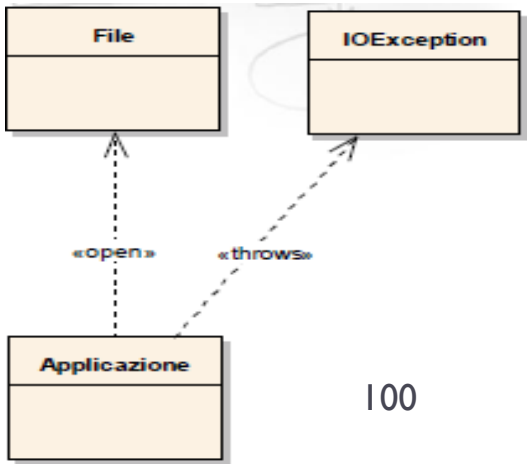
- Una relazione d'uso fra due classi si ha se:
  1. Un metodo di Class1 ha un parametro di tipo Class2
  2. Un metodo di Class1 restituisce un valore di tipo Class2
  3. Un metodo di Class1 usa un oggetto di tipo Class2 ma non come attributo. (per es. quando in un metodo di Class1 si dichiara una variabile locale di tipo Class2)
- Nel caso di una classe e di un'interfaccia la relazione d'uso ha un significato più generico: indica che la classe invoca uno o più metodi definiti nell'interfaccia. A volte si parla di relazione *client-of*

99

Relazioni tra classi
Dipendenza o associazione?

- L'**associazione** esprime un legame di ruoli, tipicamente di natura strutturale (quindi non transiente ovvero non variabile nel tempo)
  - Il dipendente, nel suo ruolo, è perennemente legato all'azienda per la quale lavora.

- La **dipendenza** esprime tipicamente un generico legame di natura spesso transiente, senza ulteriore semantica più forte
  - L'eccezione viene collegata alla classe *Applicazione* solo qualora "scatti".
  - Lo «stereotipo» nella dipendenza definisce la semantica intuitiva della dipendenza

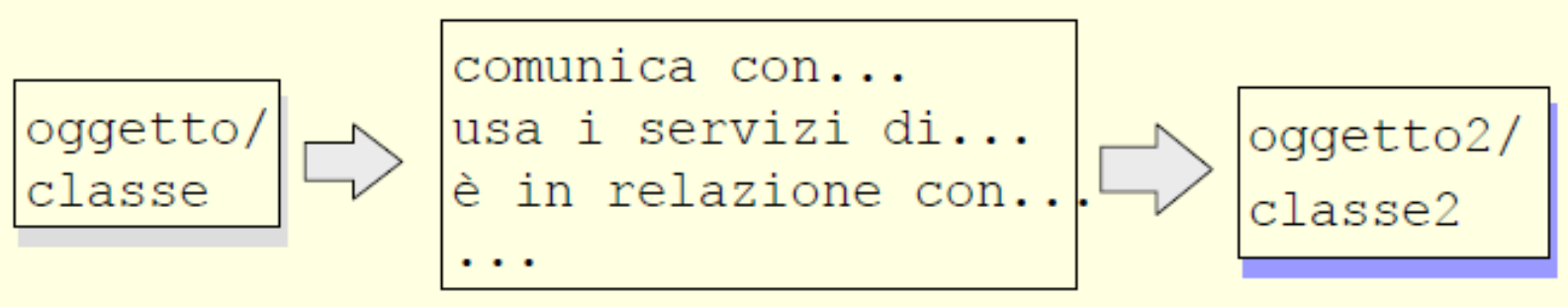


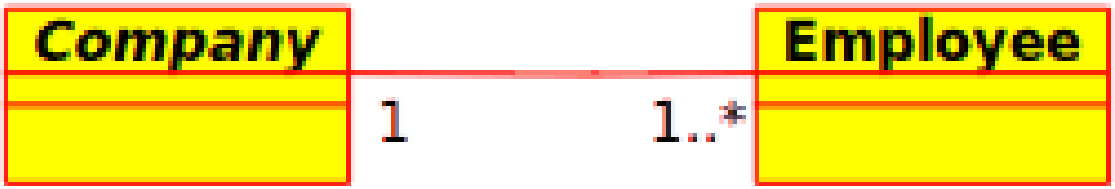
Relazioni tra classi
Associazioni

- Relazione tra due o più classi che specifica la presenza di connessioni tra le corrispondenti istanze (oggetti)
- Implica un'iterazione tra gli oggetti o classi, che *"riconoscono"* gli oggetti all'altro estremo: diciamo che una classe A *utilizza* una classe B se un oggetto della classe A è in grado di inviare dei messaggi ad un oggetto di classe B oppure se un oggetto di classe A può *creare, ricevere o restituire* oggetti di classe B.
- Una classe è associata ad un'altra se è possibile "navigare" da oggetti della prima classe ad oggetti della seconda classe seguendo semplicemente un riferimento ad un oggetto.

101

Relazioni tra classi
Associazioni



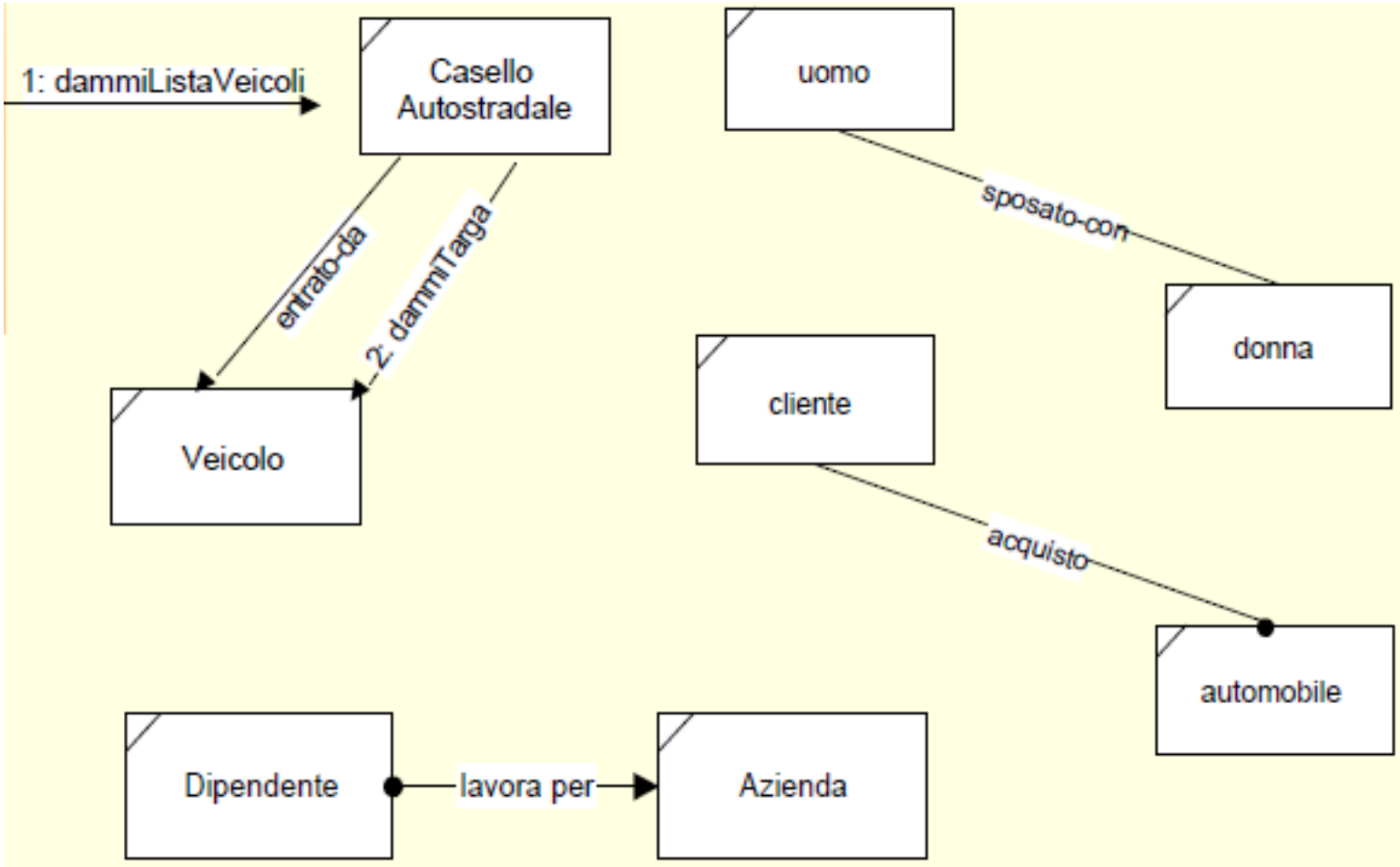


Relazioni tra classi
Associazioni

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

103

Relazioni tra classi
Associazioni



104

Relazioni tra classi
Associazione - UML

- L'associazione è rappresentata da una linea continua che può essere
  - Monodirezionale (la classe di partenza è detta *origine, mentre quella di arrivo è detta destinazione)* con freccia
  - Bidirezionale (composta da due associazioni monodirezionali)
- Il numero di istanze legate dall'associazione è specificato dalle *molteplicità agli estremi dell'associazione*

Esempio: "An Employee works in a department of a Company"

105

Relazioni tra classi
Associazione - UML

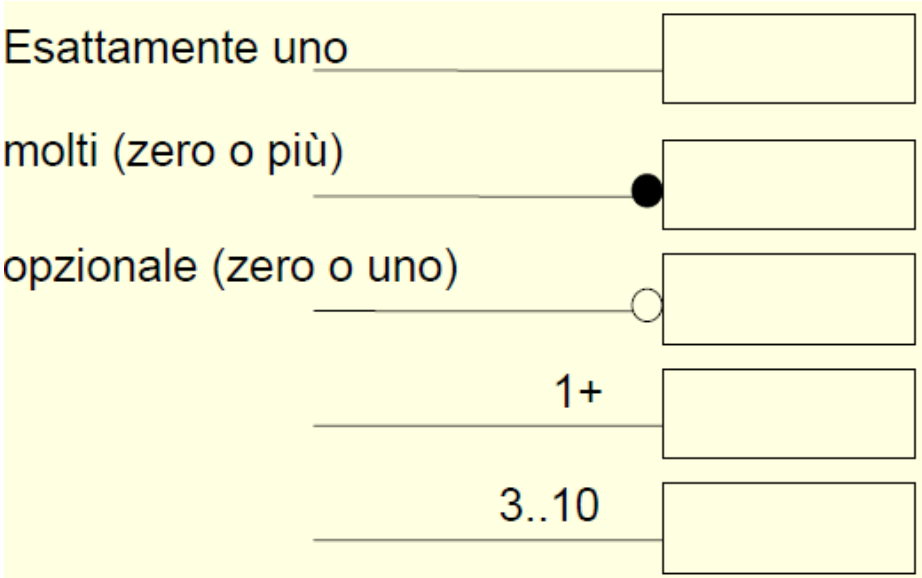
- Il **nome dell'associazione** è posta al centro dell'associazione stessa (solitamente un verbo), è obbligatoria
- Un **ruolo** è un'etichetta posta ad (almeno) uno degli estremi di un'associazione
  - Può indicare il ruolo svolto dalla classe a cui si riferisce nell'ambito dell'associazione
  - Solitamente è un nome (talvolta visto come attributo stesso della classe collegata)
  - Obbligatorio solo per relazioni di associazione riflessive

106

Relazioni tra classi
Cardinalità o molteplicità

- Definisce il numero di oggetti che per ogni classe partecipano alla relazione stessa

  - Uno (1)
  - Zero o uno (0..1)
  - Zero o più * o (0..*)
  - Da zero a N (0..N)
  - Uno o più (1..*)
  - Da 1 a N (1..N)
  - Da N a M (N..M)



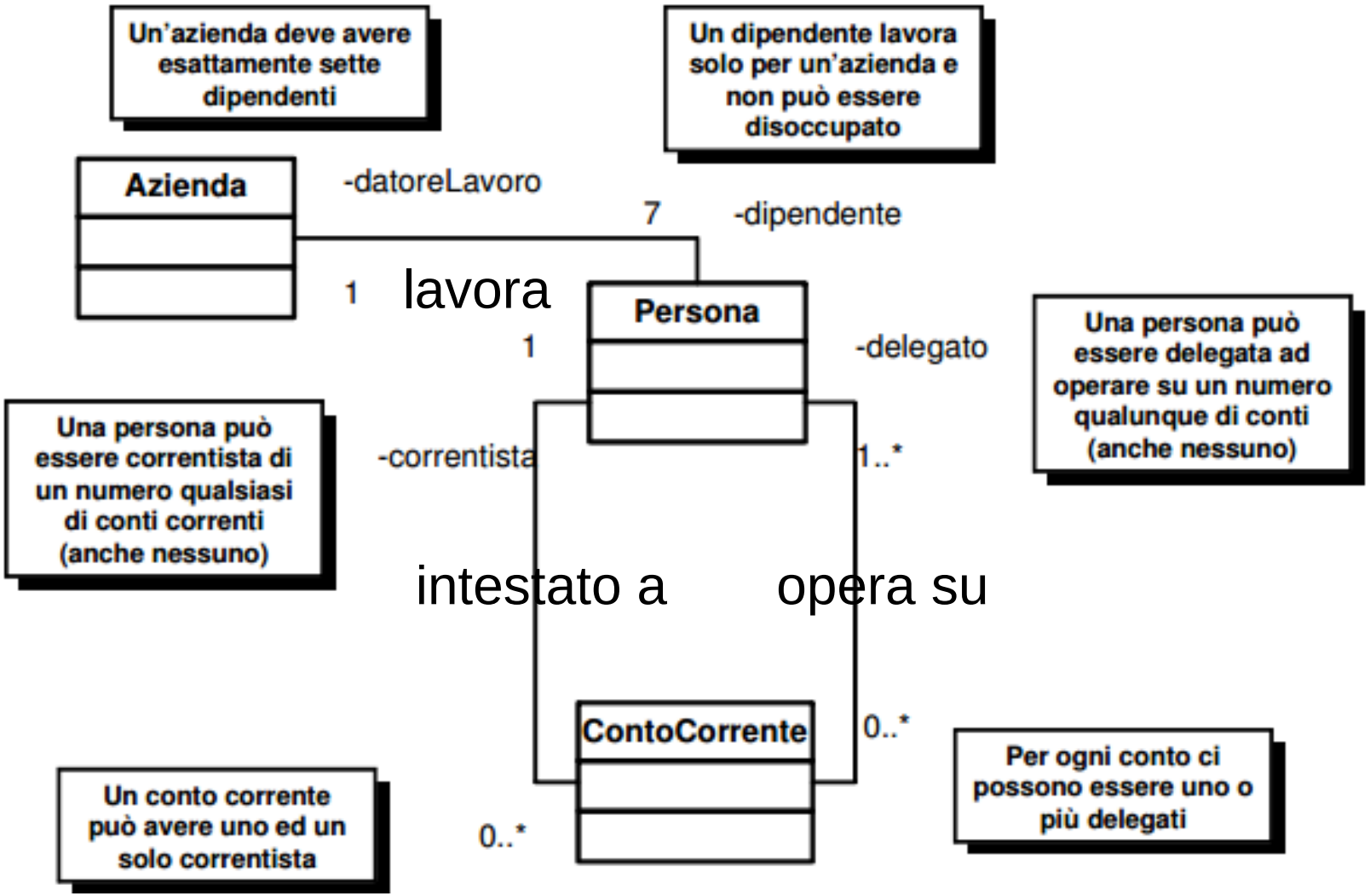
107

Relazioni tra classi
Cardinalità o molteplicità

- Indica il *numero di istanze della classe più vicina che possono esistere a run-time in una configurazione valida del sistema …*
- … che sono riferite da una *singola istanza della classe che sta all'estremo opposto della relazione*
- Specifica se un'associazione è obbligatoria o meno
- Fornisce un *intervallo di validità (estremo inferiore ed estremo superiore) relativamente al numero di istanze contemporaneamente presenti e legate a ciascun ruolo dell'associazione in un dato momento*

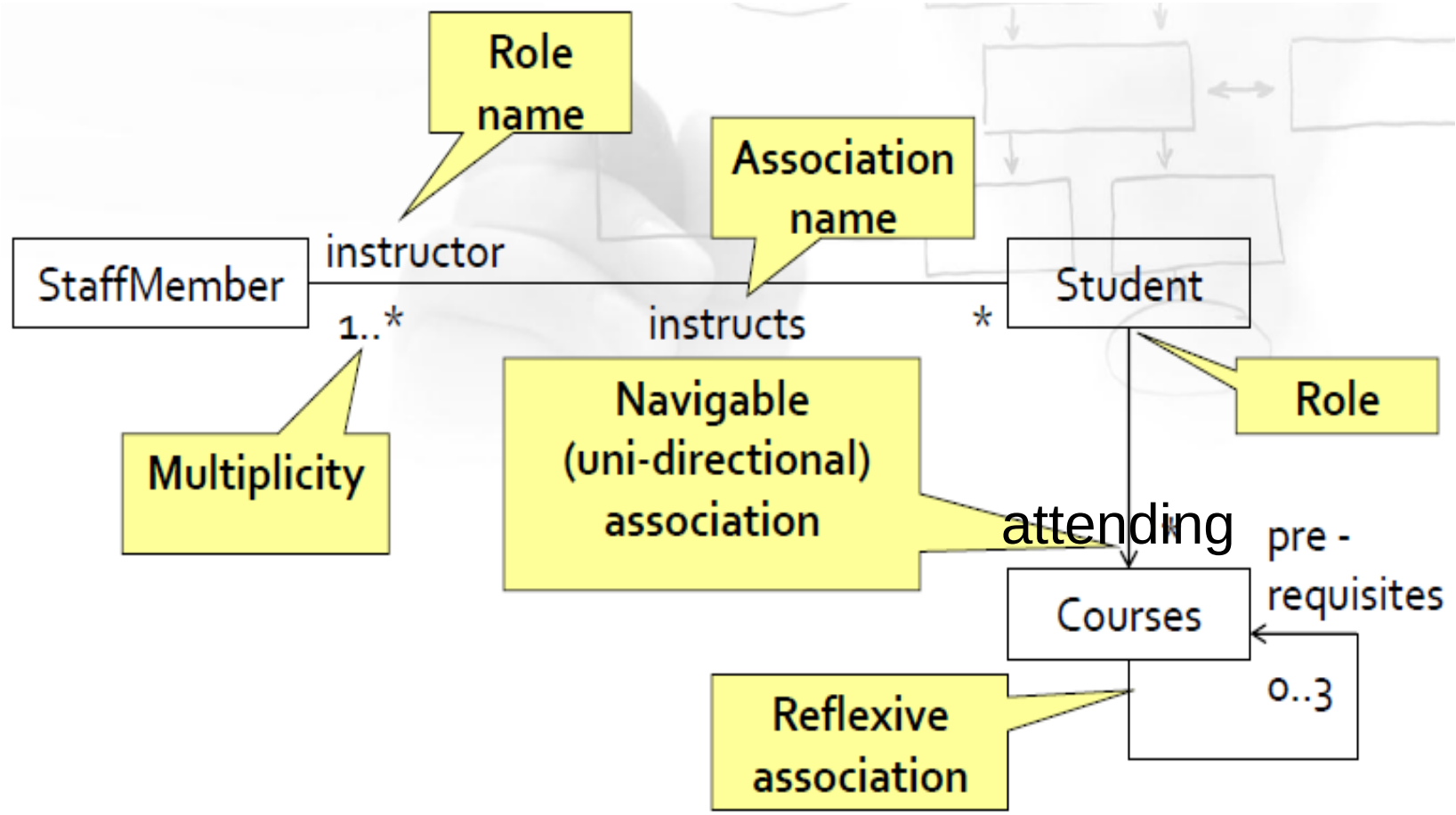
108

Relazioni tra classi
Associazione - UML



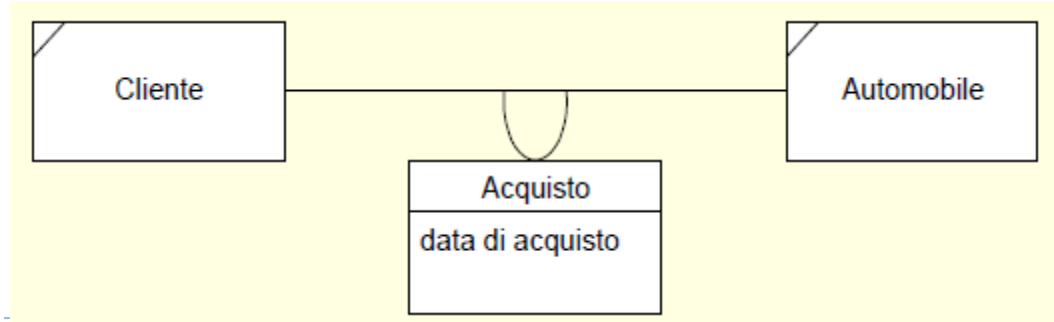
109

Relazioni tra classi
Associazione - UML



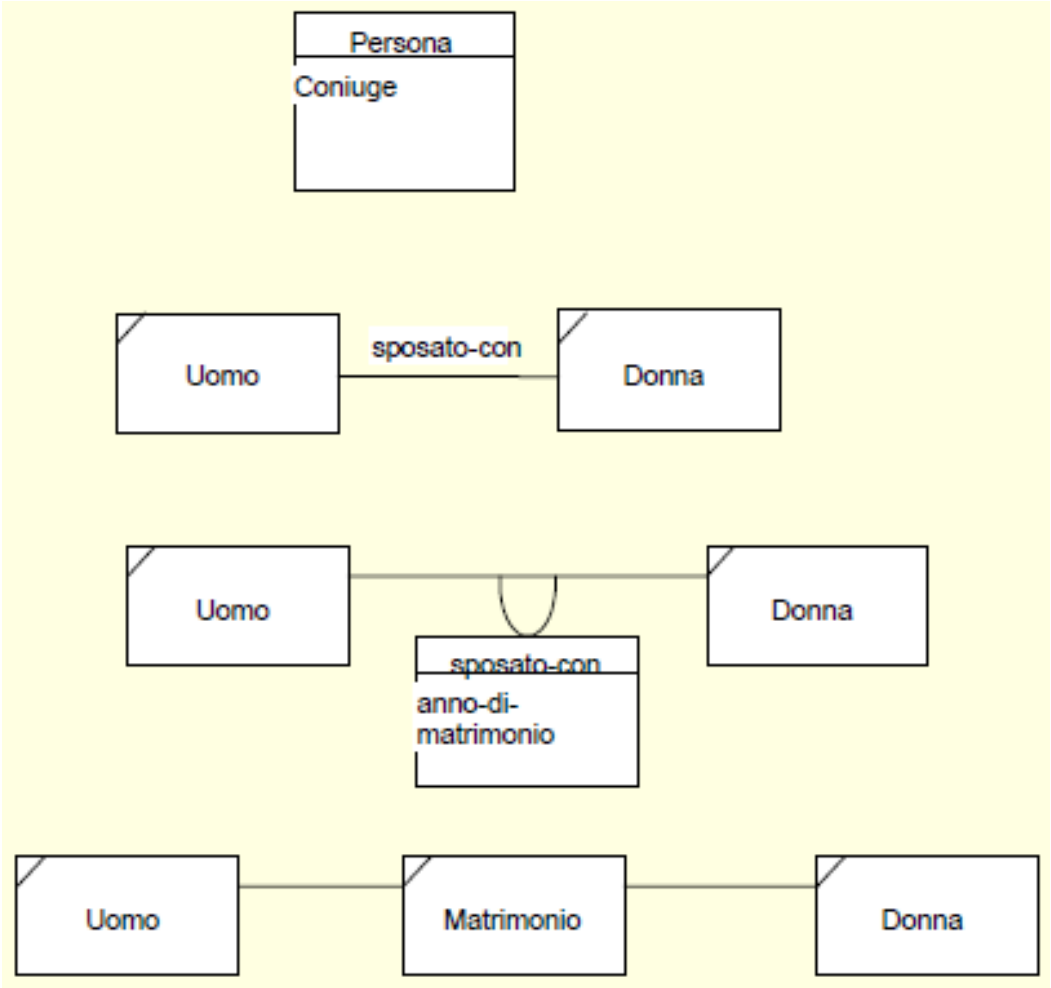
Relazioni tra classi
Classe associativa: attributi

- Per poter aggiungere attributi ad una associazione piuttosto che alle classi coinvoltesi usa la classe associativa, ossia una classe derivata da una associazione.
- Le classi associative sottintendono un vincolo aggiuntivo, ossia il fatto che ci può essere solo un'istanza della classe di associazione fra ogni coppia di oggetti associati.



Relazioni tra classi
Associazione, attributo o classe?

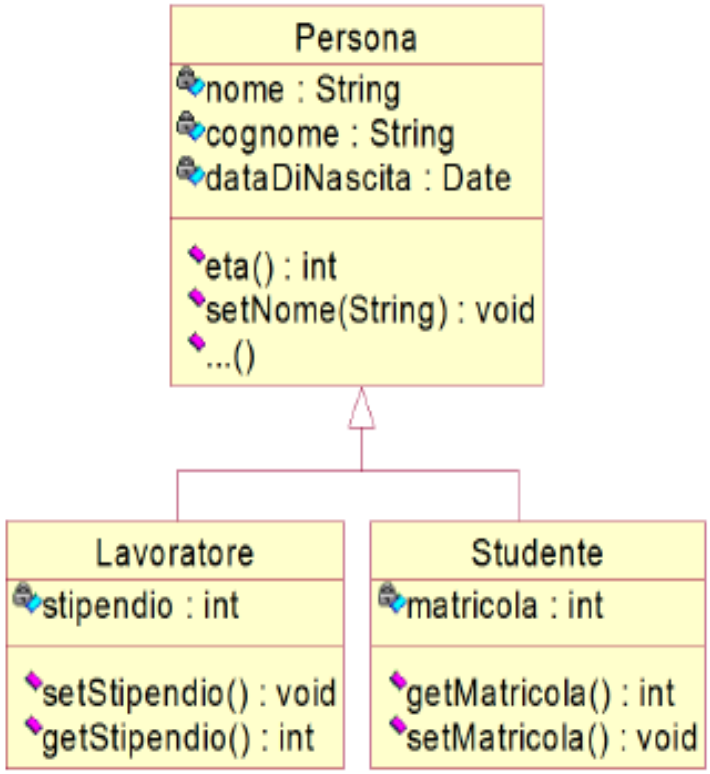
- Quale fare?



112

Relazioni tra classi
Esempi di analisi

- *Si vuole gestire un'anagrafica degli studenti e dei lavoratori di una città*
- Primo approccio: gerarchia Persona←Lavoratore e Persona←Studente
  - Cosa succede quando una persona smette di studiare ed inizia a lavorare?
  - … devo creare una nuova istanza di Lavoratore, copiare lo stato dell'istanza di Studente, "valorizzare" la parte di stato specifica di Lavoratore (e.g. stipendio), cancellare la vecchia istanza di Studente …
  - … ma adesso ho un'istanza diversa dalla precedente, però la persona è sempre la stessa!



113

Relazioni tra classi
Esempi di analisi

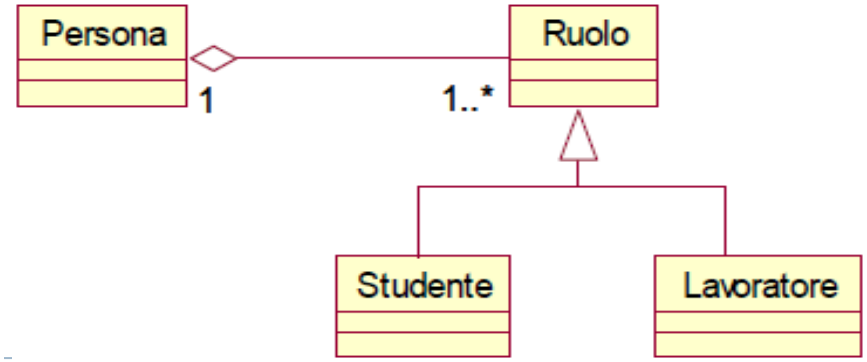
- Quali problemi emergono attraverso questo approccio? Se Andrea è uno studente lavoratore?
  - Due oggetti diversi e "vivi" (andrea1 e andrea2) per un unico esemplare di Persona (identità) (andrea)
- Possibili incoerenze (anche in fase di manutenzione):
  - Ho copiato correttamente lo stato di andrea1 nello stato di andrea2?
  - Ho inizializzato completamente andrea2?
  - Una volta diventato solo più Lavoratore, il numero di matricola di andrea1 non dovrebbe essere più valido. Chi lo garantisce? Ossia chi mi garantisce che il programmatore si ricordi sempre anche di cancellare la vecchia istanza?

L'ereditarietà è una relazione statica!

114

Relazioni tra classi
Esempi di analisi

- Come risolvere il problema? Capendo che ciò che sto modellando non è l'identità di una Persona, bensì il suo Ruolo!
- Linea guida: preferire sempre il contenimento all'ereditarietà.
- Vantaggi:
  - Una persona può svolgere più ruoli contemporaneamente (studente e lavoratore)
  - Una persona può cambiare ruolo dinamicamente



115

Implementazione in Java delle relazioni tra classi

Traduzione in Java di UML aggregazione lasca

- Comporta l'indipendenza del ciclo di vita dell'oggetto contenuto dall'oggetto contenitore. L'oggetto contenuto potrà quindi esistere anche indipendentemente dal contenitore. Il contenitore **non** ha responsabilità per la creazione e distruzione dell'oggetto contenuto
- Il contenitore dovrà definire un costruttore che riceva in input il puntatore (la maniglia) all'oggetto contenuto e ha un attributo privato di tipo riferimento ad un oggetto di tipo contenuto.
- Il client della classe contenitore deve:
  - Istanziare l'oggetto contenuto e ottenere la maniglia
  - Istanziare l'oggetto contenitore passando la maniglia al contenuto

117

Traduzione in Java di UML aggregazione lasca

```
class Contenitore {
   private Contenuto contenuto; .....
   public Contenitore(Contenuto q) {
                  contenuto=q;
   };
   public met(){//richiama metodi di Contenuto
         contenuto.metContenuto();…}
.......}
public static void main (...) {
Contenuto x = new  Contenuto(3);
Contenitore c =new  Contenitore(x);
//usa classe C
...
}
```

Traduzione in Java di UML
aggregazione stretta

- L'oggetto "contenitore" è **responsabile** della **costruzione** e **distruzione** dell'oggetto contenuto
- Si deve aggiungere un attributo privato della classe "contenuto" che viene istanziato:
  - nel costruttore del "contenitore" oppure
  - prima del costruttore, nella classe "contenitore", cioè nel momento in cui è definito

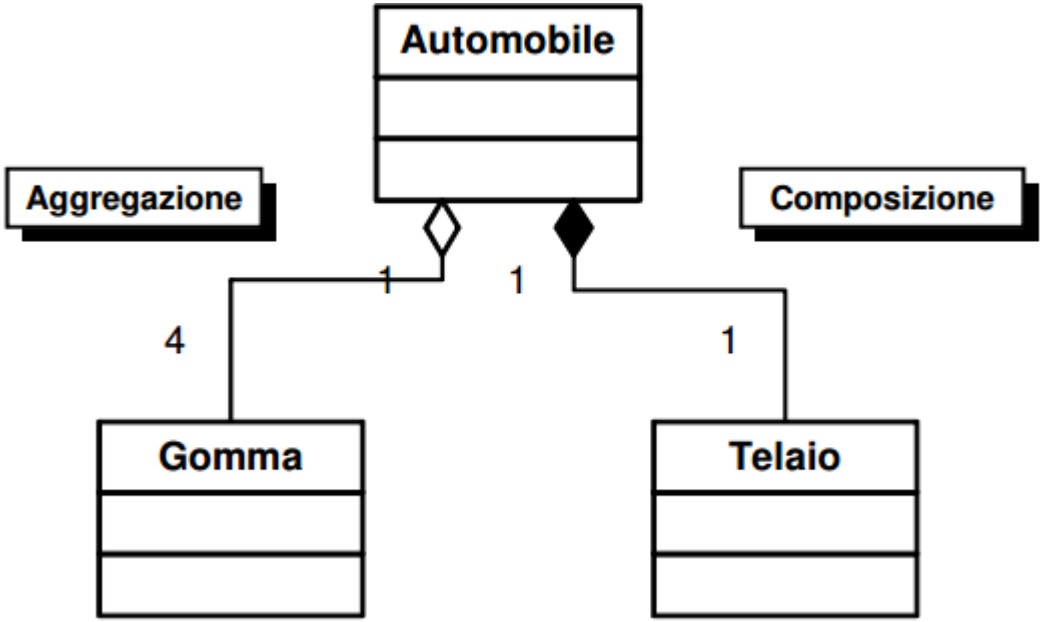
119

Traduzione in Java di UML aggregazione stretta

```
public class Contenitore {
  private  Contenuto contenuto;
  .......
  public  Contenitore(int val) {
     contenuto = new  Contenuto(val);…}
}
//oppure se non è possibile fornire un argomento al
  costruttore
public class  Contenitore  {
  private Contenuto contenuto = new Contenuto();
  .......
  public  Contenitore() {…};
}
```

Traduzione in Java di UML
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



121

Traduzione in Java di UML
aggregazione e composizione

```
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

122

Traduzione in Java di UML associazione

È caratterizzata da:

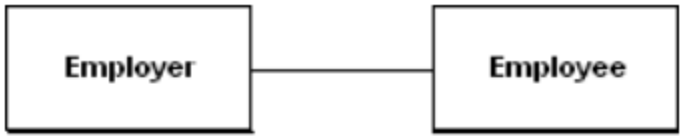
- un nome (opzionale) che esprime il legame semantico tra le classi associate;
- il ruolo giocato dalle parti associate (opzionale);
- la molteplicità dell'associazione che esprime la cardinalità delle connessioni tra gli oggetti:
  - uno a uno
  - uno a molti
  - molti a molti
- la direzionalità (bidirezionale di default) indica chi ha la responsabilità di tenere traccia dell'associazione:
  - bidirezionale responsabilità multipla
  - unidirezionale responsabilità singola

123

Traduzione in Java di UML associazione

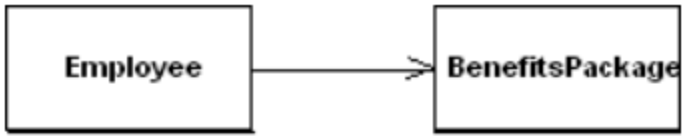
Esempio di:

- associazione bidirezionale: ogni classe ha un riferimento all'altra
- associazione unidirezionale, solo la classe da cui parte la freccia ha il riferimento all'altra




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

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




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

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


124

Traduzione in Java di UML associazione

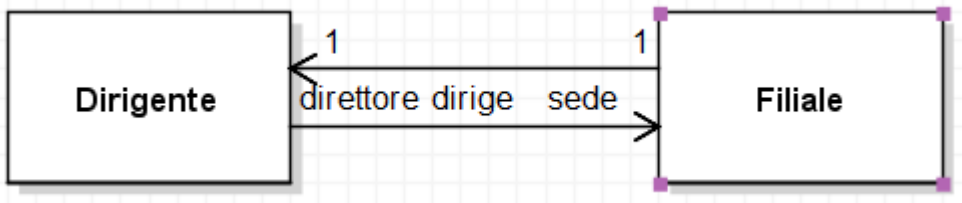
- Se è specificato solo il nome dell'associazione la variabile membro di tipo puntatore prende il nome dell'associazione stessa.
- Se è specificato il ruolo della classe associata il nome della variabile membro sarà quello del ruolo
- Il programma client avrà la responsabilità della creazione e del collegamento dei due oggetti

125

Traduzione in Java di UML
es. associazione 1:1 bidirezionale

```
class Filiale{
  private Dirigente direttore; …
  public void direttaDa(Dirigente d){direttore=d;}
…}
class Dirigente{
  private Filiale sede; …
  public void dirige(Filiale f){sede=f;}
…}

//nella classe client si avrà
Dirigente d=new Dirigente(…);
Filiale f=new Filiale(…);
//collega i due oggetti.
d.dirige(f);
f.direttaDa(d);
```

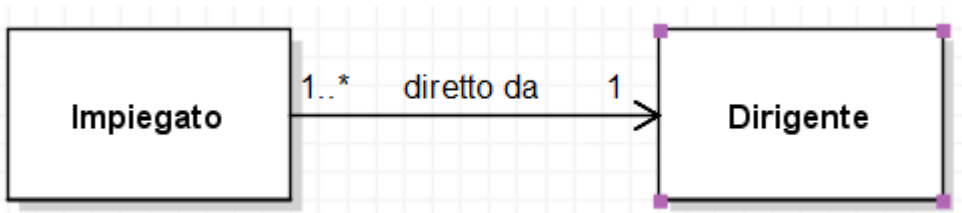


Traduzione in Java di UML
associazione 1:N unidirezionale

- Se a responsabilità singola VERSO 1 senza attributi

```
public class Impiegato{
  private String nome;
  private Dirigente direttoDa;
  public Impiegato(String n, Dirigente d) {
       nome = n; direttoDa = d;}
  public String getNome() { return nome; }
  public void associaDir(Dirigente d) {
       if (d!= null) direttoDa = d; }
}
```

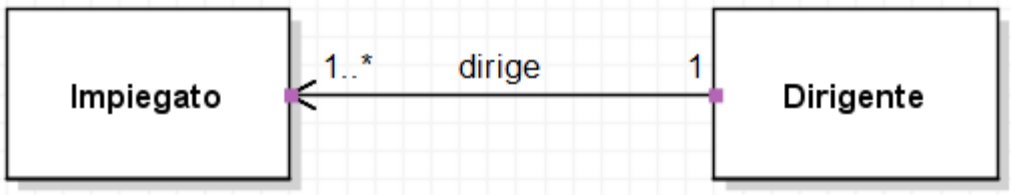
- Se l'impiegato potesse non avere nessun dirigente, nel costruttore non si passerebbe il dirigente, resterebbe solo `associaDir()`



Traduzione in Java di UML associazione 1:N unidirezionale

Se a responsabilità singola VERSO N senza attributi

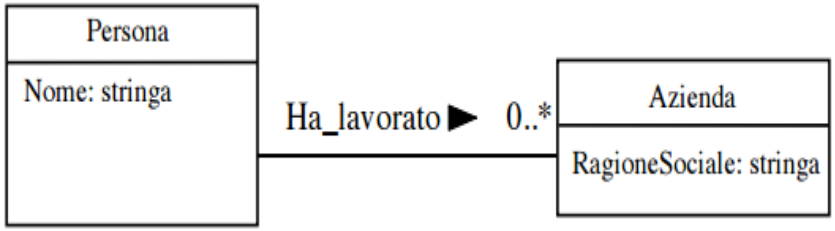
```
public class Dirigente{
  private String nome;
  private Vector <Impiegato> sottoposti;
  public Dirigente(String n) {
       nome = n; sottoposti = new Vector<Impiegato>(); }
  public String getNome() { return nome; }
  public void addSottoposto(Impiegato i) {
       if (i != null) sottoposti.add(i); }
  public void eliminaSottoposto(Impiegato i) {
       if (i != null) {
            int k= sottoposti.indexOf(i);
            if (k!=-1) sottoposti.removeElementAt(k);}
   }
}
```



Traduzione in Java di UML
associazione N:N unidirezionale

Se a responsabilità singola senza attributi

```
public class Persona {
  private String nome;
  private Vector <Azienda> haLavorato;
  public Persona(String n) {
       nome = n; haLavorato = new Vector<Azienda>(); }
  public String getNome() { return nome; }
  public void inserisciLinkHaLavorato(Azienda az) {
       if (az != null) haLavorato.add(az); }
  public void eliminaLinkHaLavorato(Azienda az) {
       if (az != null) {
            int i=haLavorato.indexOf(az);
            if (i!=-1) haLavorato.removeElementAt(i);}
   }
}
```



Traduzione in Java di UML

- Per rappresentare l'associazione A con attributi fra le classi UML C e D si introduce un'ulteriore classe Java TipoLinkA. Ci sarà un oggetto di classe TipoLinkA per ogni link fra un oggetto di classe C ed uno di classe D.
- La classe Java TipoLinkA conterrà:
  - gli attributi dell'associazione;
  - i riferimenti agli oggetti delle classi C e D che costituiscono le componenti della tupla che il link rappresenta

130