Programmazione a Oggetti
Modulo B

Lezione 9

Dott. Alessandro Roncato

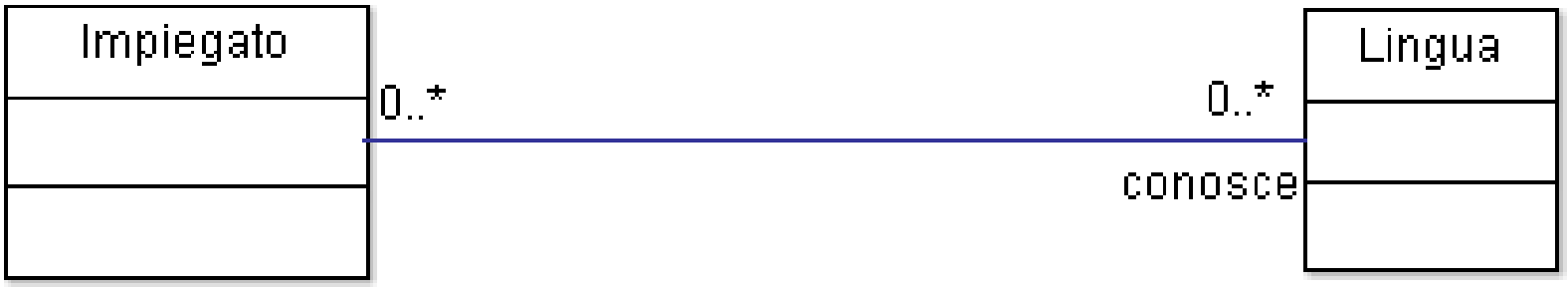
03/03/2014

Riassunto

Esercizi d'esame

Progetto

Associazione “Normale”



Associazione “Normale”

```
public class Lingua {
   public Collection<Persona> getPersone(){
      return persone;
   }
   Set<Persona> persone = …;
}
-----------------------------------------------
public class Impiegato {
   public Collection<Lingua> getLingue(){
     return lingue;
   }
   Set<Lingua> lingue = …;
}
```

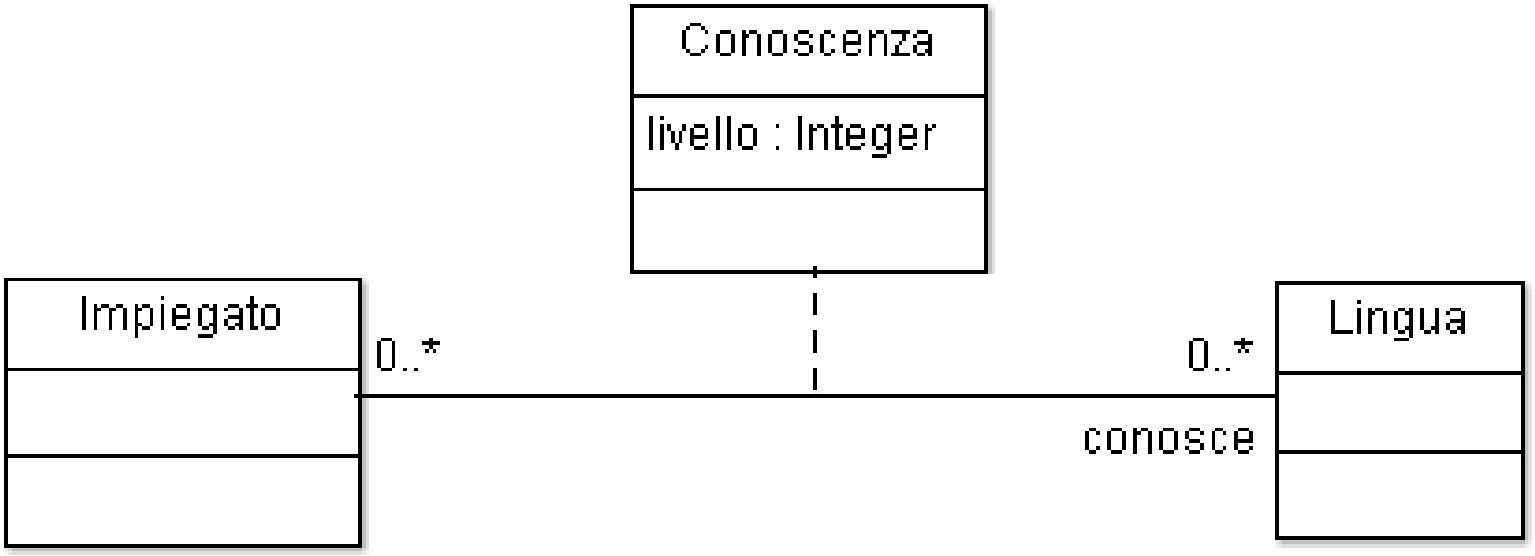
Classe di Associazione

Qualche volta sorge la necessità di aggiungere delle informazioni a una associazione.

Per esempio, tra Impiegato e Lingua è necessario aggiungere il livello di conoscenza che impiegato ha della lingua.

Quindi, ogni coppia Impiegato , Lingua presente nell'associazione ha un relativo livello di conoscenza.

Classe di Associazione



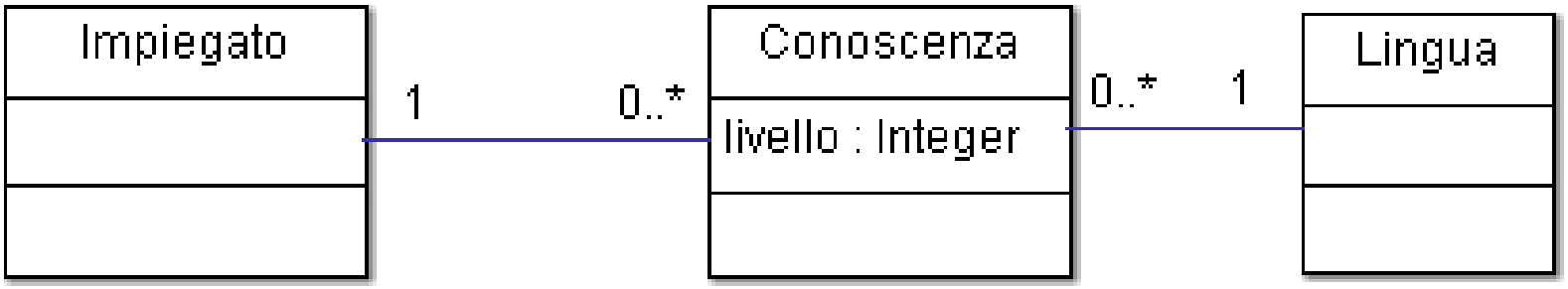
Classe di Associazione/ Classe

Dal punto di vista dell'implementazione una classe di associazione viene implementata come una classe normale.

Dal punto di vista UML, la classe associazione richiede che ci sia una e una sola istanza della classe associazione per ogni coppia di oggetti dell'associazione.

Il programmatore deve garantire che questo venga rispettato! (Non facile)

Associazione promossa Classe

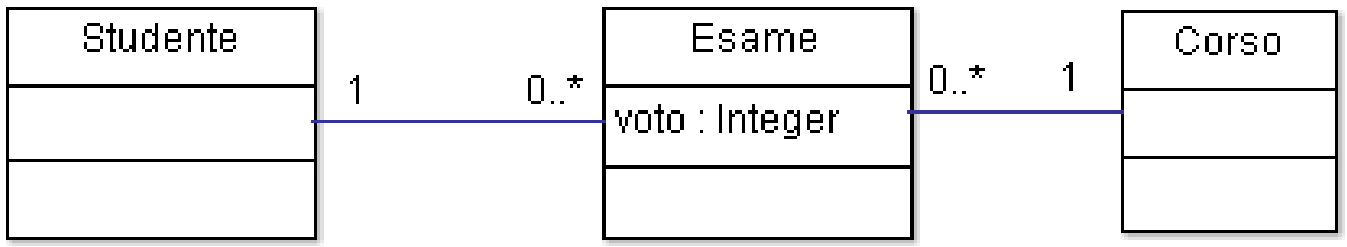


Implementazione Classe di Associazione

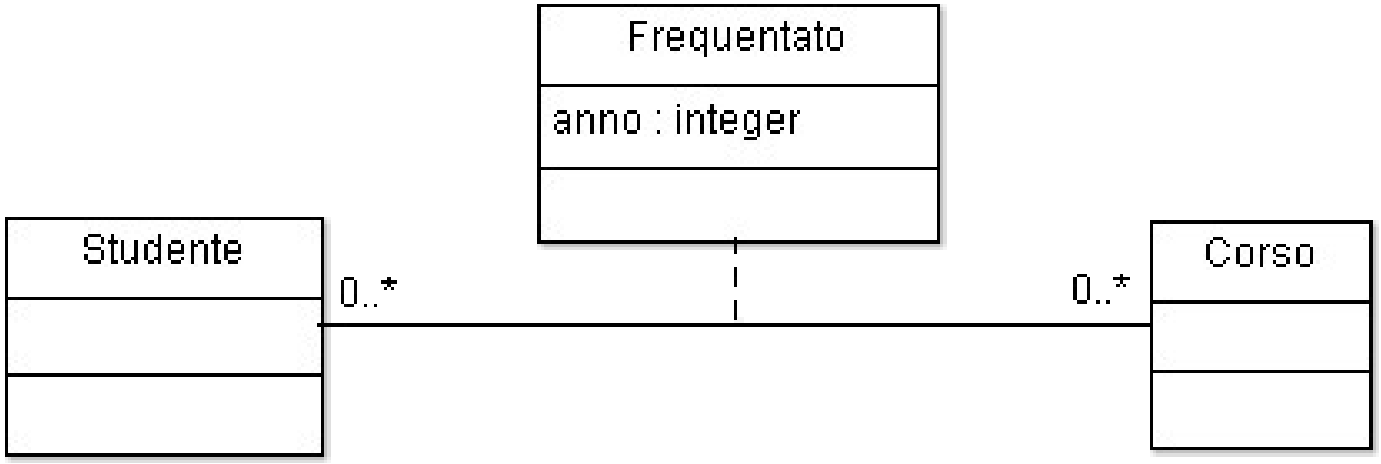
Dal punto di vista dell'implementazione una classe di associazione viene implementata come una classe normale.

Una possibile soluzione consiste nel delegare una delle due classi dell'associazione (es. Impiegato) la gestione della Classe Associazione e questa classe (Impiegato) fa in modo di garantire il vincolo.

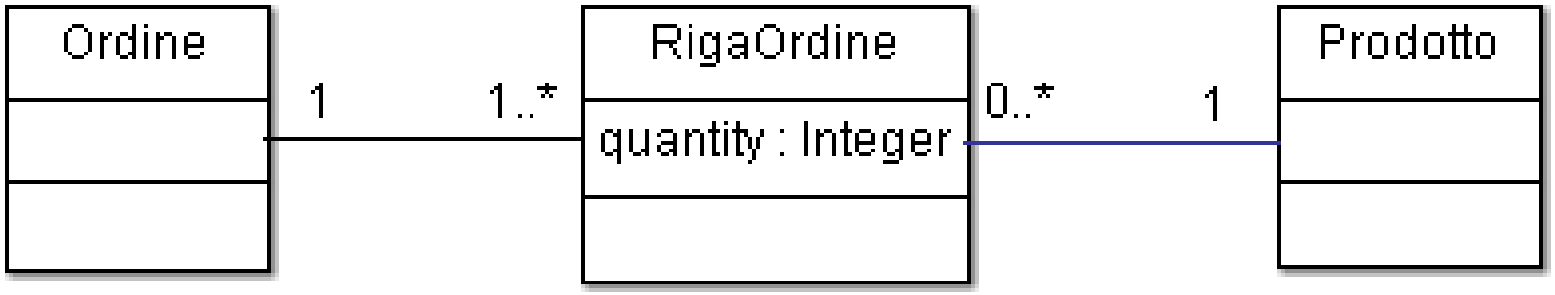
Associazione promossa Classe



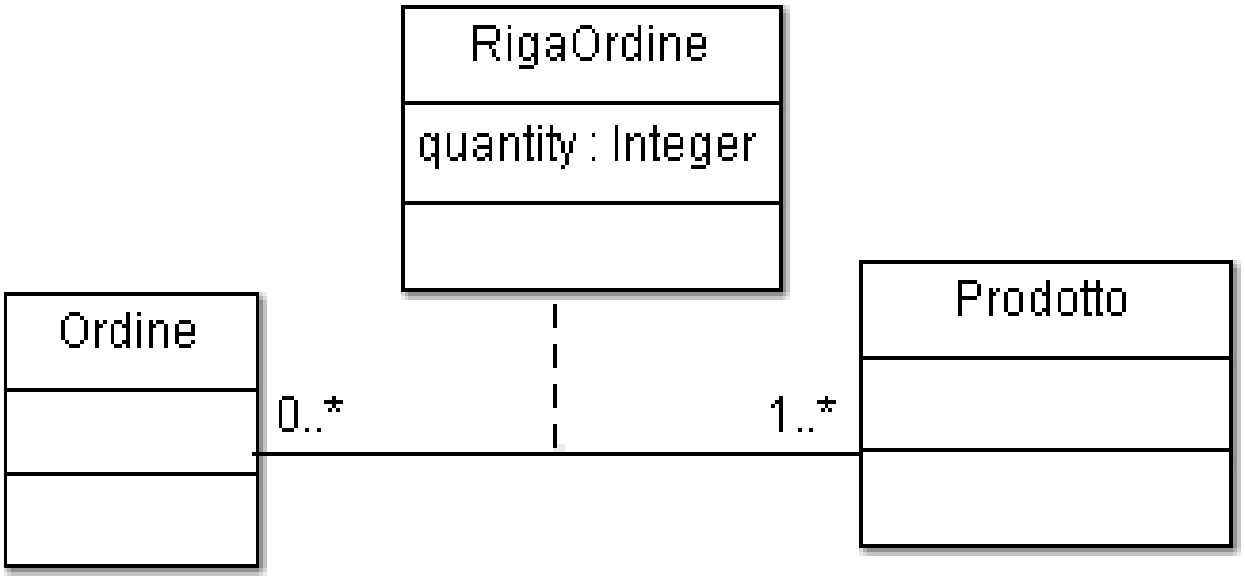
Classe di Associazione



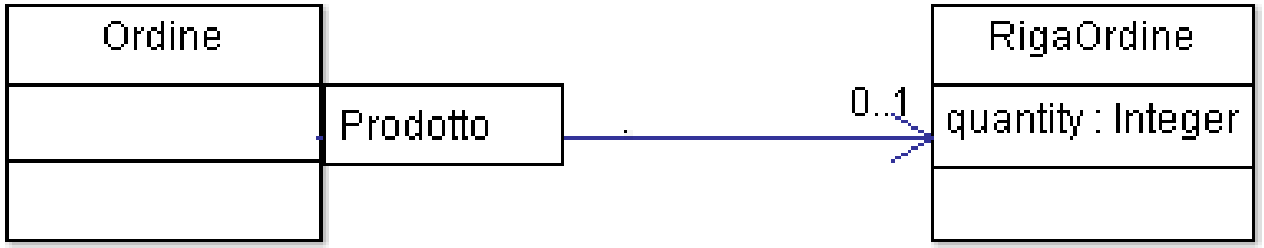
Classe normale



Classe di Associazione



Associazione Qualificata



Associazione Qualificata

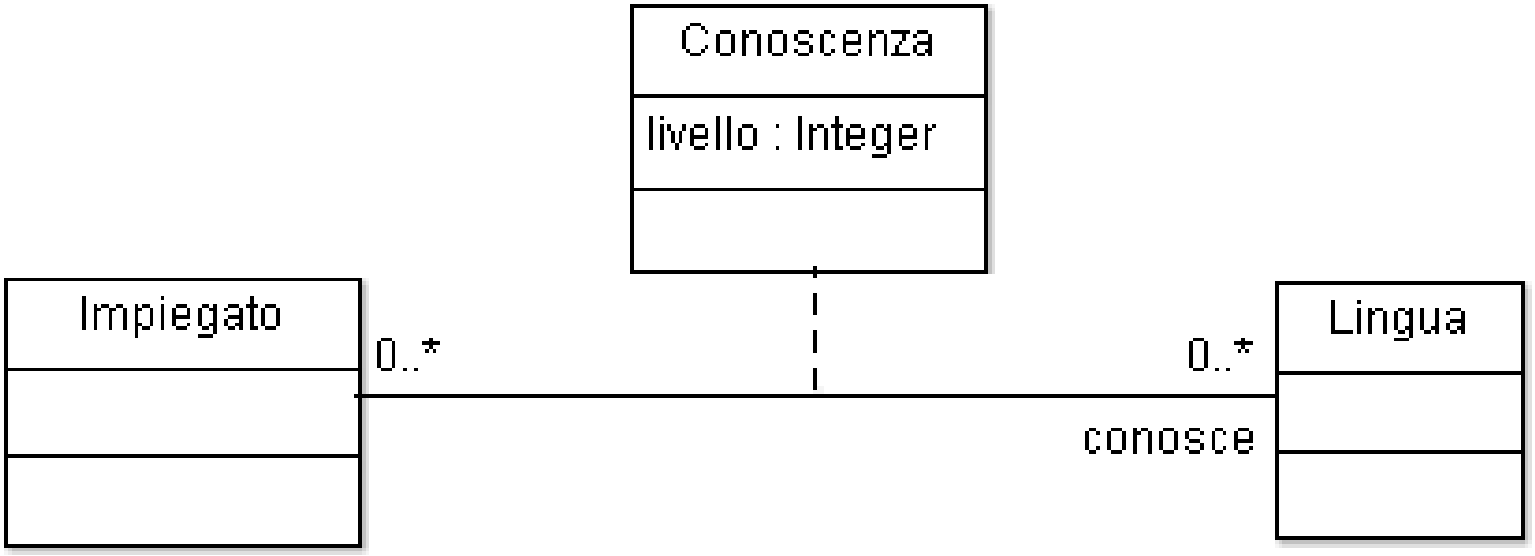
```
public class Ordine {

   public RigaOrdine getRiga(Prodotto p){
     …
   }

   public void addRiga(Prodotto p, int quantity){
   …
   }

   Map<Prodotto,RigaOrdine> righeOrdini = …
}
```

Classe di Associazione



Implementare Classe Associazione

```
public class Conoscenza {
    Impiegato impiegato;
    Lingua lingua;
    int level;
    public Conoscenza (Impiegato i, Lingua l, int le)
{
        …
    }
    public void setLevel(int lev){
        this.level=lev;
    }
    …
}
```

Implementare Classe Associazione

```
public class Impiegato {
   public Conoscenza getConoscenza(Lingua l){
      return conoscenze.get(l);
   }
   public void addConoscenza(Lingua l, int level){
      Conoscenza c = conoscenze.get(l);
      if (c==null) {
         c = new Conoscenza(this,l,level);
         conoscenze.put(l,c);
         l.put(this,c);
      } else {
         c.setLevel(l);
      }
   }
   public Collection<Lingua> getLingue(){
      return conoscenze.keySet();
   }
   Map<Lingua,Conoscenza> conoscenze = …
}
```

Implementare Classe Associazione

```
public class Lingua {
   public Collection<Impiegati> getImpiegati(){
      return conoscenze.keySet();
   }
   public void put(Impiegato i, Conoscenza c){
     conoscenze.put(i,c);
   }
   public Conoscenza getConoscenza(Persona p){
      return conoscenza.get(p);
   }

   Map<Persona,Conoscenza> conoscenze = …;
}
```

Implementare Classe Associazione

```
public class Lingua {
   public Collection<Persona> getPersone(){
      return conoscenze.keySet();
   }
   public void put(Persona p, Conoscenza c){
     conoscenze.put(p,c);
   }
   public Collection<Conoscenza> getConoscenze(Persona p){
      return conoscenze.valueSet();
   }

   Map<Impiegato,Conoscenza> conoscenze = …;
}
```

Implementare Classe Associazione

```
public class Lingua {
   public Collection<Impiegato> getImpiegato(){
      return conoscenze.keySet();
   }
   public void put(Impiegato i, Conoscenza c){
     if(i.getConoscenza(this)==c)
        conoscenze.put(i,c);
     else …
   }
   public Conoscenza getConoscenza(Impiegato i){
      return conoscenza.get(i);
   }
   Map<Persona,Conoscenza> conoscenze = …;
}
```

Implem. Classe Associazione 2

```
public class Lingua {
    public Collection<Impiegato> getImpiegato(){
        return impiegati;
    }
    public void put(Impiegato i, Conoscenza c){
      if(i.getConoscenza(this)==c)
          impiegati.add(i);
      else …
    }
    public Conoscenza getConoscenza(Impiegato i){
        return i.getConoscenza(this);
    }
    Set<Impiegato> impiegati = …;
}
```

Implem. Classe Associazione 2

```
public class Lingua {
    public Collection<Impiegato> getImpiegato(){
        return impiegati;
    }
    synchronized public void put( … ){
      if(i.getConoscenza(this)==c)
          persone.add(i);
      else …
    }
    public Conoscenza getConoscenza(Impiegato i){
        return i.getConoscenza(this);
    }
    Set<Impiegato> impiegati = …;
}
```

Classe Interna

Per ovviare ai problemi di visibilità delle soluzioni precedenti possiamo definire Conoscenza come classe interna della classe che si occupa di garantire il vincolo (Persona nell'esempio).

In questo modo, aggiungendo ulteriore asimmetria tra Persona e Lingua otteniamo che tutta la responsabilità della gestione del vincolo ricade su il file Persona.java

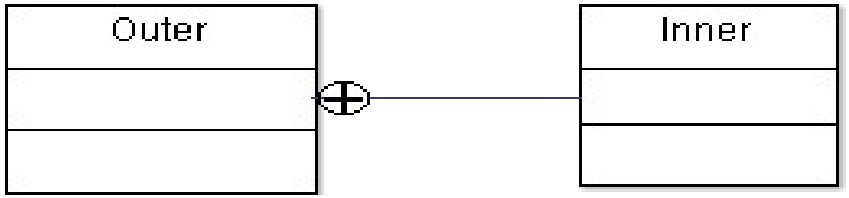
Classe Interna

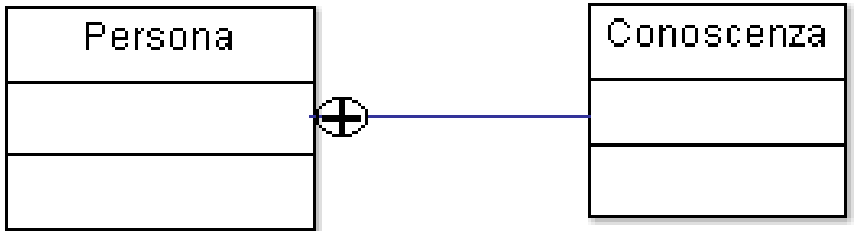
```
public class Impiegato {
   public class Conoscenza {
      …
      private Conoscenza( … ){ … }
  }
   public void addConoscenza(Lingua l, int level){
      Conoscenza c = conoscenze.get(l);
      if (c==null) {
         c = new Conoscenza(this,l,level);
         conoscenze.put(l,c);
         l.put(this,c);
      } else {
         c.setLevel(l);
      }
   }
   … resto resta uguale
}
```

Associazione Qualificata

```
public class Lingua {
   public Collection<Impiegato> getImpiegati(){
      return impiegati;
   }
   public void put(Persona p, Impiegato.Conoscenza
c){
     if(p.getConoscenza(this)==c)
        persone.add(p);
     else …
   }
   public Impiegato.Conoscenza getConoscenza( … p){
      return p.getConoscenza(this);
   }
   Set<Impiegato> impiegati = …;
}
```

Classe Interna in UML





Classe di Associazione / Associazione Qualificata

Associazione qualificata quando i 3 concetti esistono ed hanno senso di per se stessi

Classe di Associazione quando solo i 2 concetti esistono (la classe di associazione è quasi di invenzione)

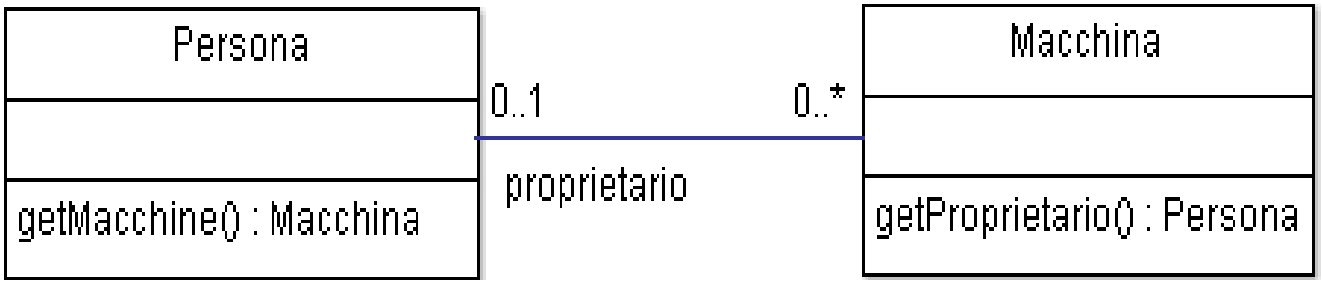
Spesso i due concetti sono legati sia dal punto di vista concettuale, sia dal punto di vista dell'implementazione.

Problema consistenza

Tutte le chiamate ai metodi pubblici dovrebbero lasciare l'oggetto in uno stato consistente e quindi anche in uno stato che soddisfi i diagrammi UML.

Una delle cose da soddisfare sono i vincoli delle associazioni con doppia navigabilità.

Classe Interna in UML



Vincolo:

```
for (Macchina m: p.getMacchine())
   assert(p==m.getProprietario());
```

Problema consistenza

Il metodo che implementa il cambio di proprietario della macchina deve tenere conto di cambiare lo stato anche degli oggetti Persona associati.

Il problema risiede nell'accesso dei campi privati...

Cambio proprietario

```
public class Macchina {
   Persona proprietario;

  public void setProprietatio(Persona nuovoProprietario){
    if (proprietario != null)
       proprietario.vendi(this);

    proprietario=nuovoPropretario;

    nuovoProprietario.compra(this);
  }
}
```

Cambio proprietario

```
public class Persona {
    Set<Macchina> macchine;

  public void vende(Macchina m){
    macchine.remove(m);
  }

  public void acquista(Macchina m){
   macchine.add(m);
  }

}
```

Cambio proprietario 2

```
public class Persona {
    Set<Macchina> macchine;

  public void vende(Macchina m, Persona nuovoProprietario)
  {
    if (macchine.contains(m)) {
      macchine.remove(m);
      nuovoProprietario.macchine.add(m);
      m.setProprieratio(nuovoProprietario);
    }
  }

}
```

Cambio proprietario 2

```
public class Macchina {
    Persona proprietario;

  public void setProprietatio(Persona nuovoProprietario){
    //cosa succede se nuovoProprietario == null???
    if (nuovoProprietario.contains(this){
       proprietario=nuovoPropretario;
    }
    else { … }
  }
}
```

domande