Realizzazione di una classe con un'associazione

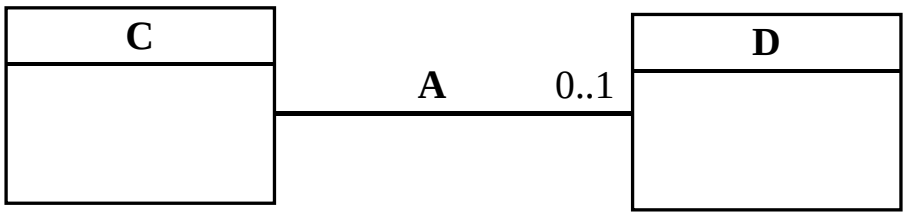
Nel realizzare una classe che è coinvolta in un'associazione, ci dobbiamo chiedere se la classe ha **responsabilità** sull'associazione.

Diciamo che **una classe C ha responsabilità sull'associazione A**, quando, per ogni oggetto *x* che è istanza di *C* vogliamo poter eseguire opportune operazioni sulle istanze di *A* a cui *x* partecipa, che hanno lo scopo di:

- conoscere l'istanza (o le istanze) di *A* alle quali *x* partecipa,
- aggiungere una nuova istanza di *A* alla quale *x* partecipa,
- cancellare una istanza di *A* alla quale *x* partecipa,
- aggiornare il valore di un attributo di una istanza di *A* alla quale *x* partecipa.

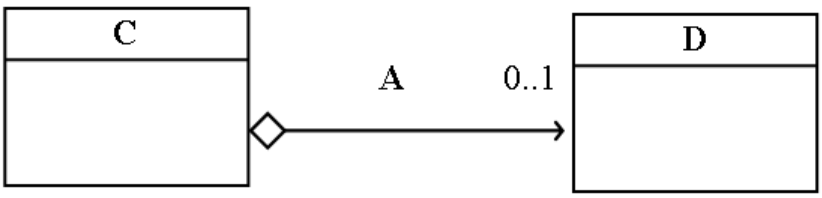
L'informazione su quali sono le classi che hanno responsabilità sulle varie associazioni si trova nella specifica delle classi. Per ogni associazione *A*, **deve** esserci almeno una delle classi coinvolte che ha responsabilità su `A`.

Realizzazione delle associazioni: primo caso



Consideriamo il caso in cui

- l'associazione sia binaria, di molteplicità 0..1 da *C* a *D*
- la specifica della classe *C* ci dice che essa è l'unica ad avere responsabilità sull'associazione *A* (cioè dobbiamo realizzare un "solo verso" della associazione)
- l'associazione *A* non abbia attributi
- non siamo interessati a mantenere esplicita la reppresentazione dei link nella associazione *A*.



Nel diagramma realizzativo sostituimo l'associazione con una **aggregazione** navigabile solo nella direzione da *C* a *D*. In questo modo modelliamo l'associazione come una relazione "has-a": gli oggetti della classe *C* hanno (come parte) un oggetto della classe *D*.

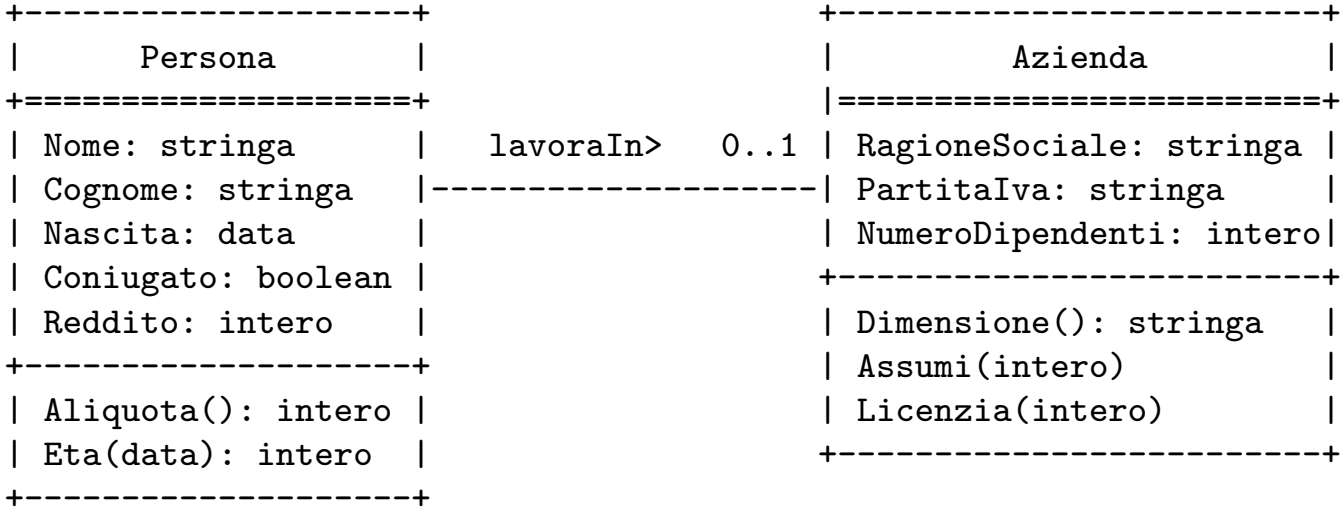
Realizzazione delle associazioni: primo caso

In questo caso, la realizzazione del codice è simile a quella per un attributo. Infatti, oltre a quanto stabilito per gli attributi e le operazioni, per ogni associazione *A* del tipo mostrato in figura, aggiungiamo alla classe Java `C`

- un campo dato di tipo `D` nella parte `private` (o `protected`) che rappresenta, per ogni oggetto `x` della classe `C`, l'oggetto della classe `D` connesso ad `x` tramite l'associazione `A`,
- una funzione `get` che consente di calcolare, per ogni oggetto `x` della classe `C`, l'oggetto della classe `D` connesso a `x` tramite l'associazione `A` (la funzione restituisce `null` se `x` non partecipa ad alcuna istanza di `A`),
- una funzione `set`, che consente di stabilire che l'oggetto `x` della classe `C` è legato ad un oggetto `y` della classe `D` tramite l'associazione `A` (sostituendo l'eventuale legame già presente); se tale funzione viene chiamata con `null` come argomento, allora la chiamata stabilisce che l'oggetto `x` della classe `C` non è più legato ad alcun oggetto della classe `D` tramite l'associazione `A`.

Due classi legate da associazione: esempio

La ragione sociale e la partita Iva di un'azienda **non cambiano**.



Supponiamo che solo *Persona* abbia responsabilità sull'associazione (non ci interessa conoscere i dipendenti di un'azienda, ma solo in quale azienda lavora una persona che lavora), assumiamo inoltre di non volere mantenere la rappresentazione esplicita dei link della associazione.

Specifica della classe UML **Azienda**

InizioSpecificaClasse Azienda

Dimensione (): *stringa*
pre: nessuna
post: *result* vale "Piccola" se *this.CapitaleSociale* è inferiore a 51, vale "Media" se *this.CapitaleSociale* è compreso fra 51 e 250, vale "Grande" se *this.CapitaleSociale* è superiore a 250

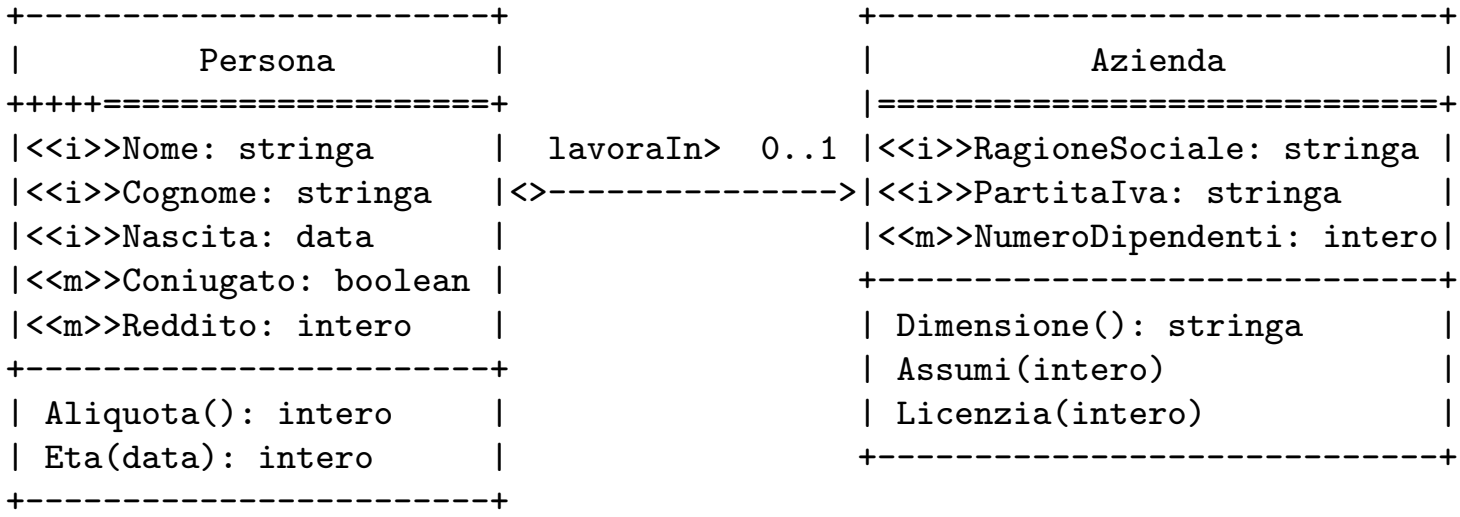
Aumenta (*i: intero*)
pre: $i > 0$
post: *this.CapitaleSociale* vale *pre(this.CapitaleSociale)* + *i*

Diminuisci (*i: intero*)
pre: $1 \leq i \leq$ *this.CapitaleSociale*
post: *this.CapitaleSociale* vale *pre(this.CapitaleSociale)* - *i*

FineSpecifica

Diagramma delle classi realizzativo

La ragione sociale e la partita Iva di un'azienda **non cambiano**.



Supponiamo che solo *Persona* abbia responsabilità sull'associazione (non ci interessa conoscere i dipendenti di un'azienda, ma solo in quale azienda lavora una persona che lavora), assumiamo inoltre di non volere mantenere la rappresentazione esplicita dei link della associazione.

Realizzazione in Java della classe Azienda

```
// File ParteQuarta/Associazioni01/Azienda.java

public class Azienda {
  private final String ragioneSociale, partitaIva;
  private int capitaleSociale;
  public Azienda(String r, String p) {
    ragioneSociale = r;
    partitaIva = p;
  }
  public String getRagioneSociale() {
    return ragioneSociale;
  }
  public String getPartitaIva() {
    return partitaIva;
  }
  public int getCapitaleSociale() {
```

```
    return capitaleSociale;
  }
  public void aumenta(int i) {
    capitaleSociale += i;
  }
  public void diminuisci(int i) {
    capitaleSociale -= i;
  }
  public String dimensione() {
    if (capitaleSociale < 51)
      return "Piccola";
    else if (capitaleSociale < 251)
      return "Media";
    else return "Grande";
  }
  public String toString() {
    return ragioneSociale + " (P.I.: " + partitaIva +
    "), capitale sociale: " + getCapitaleSociale() +
```

```
    ", tipo azienda: " + dimensione();
  }
}
```

Realizzazione in Java della classe Persona

```
public class Persona {
  // altri campi dati e funzione
  private Azienda lavoraIn;

  public Azienda getLavoraIn() {
    return lavoraIn;
  }
  public void setLavoraIn(Azienda a) {
    lavoraIn = a;
  }
  public String toString() {
    return nome + ' ' + cognome + ", " + giorno_nascita + "/" +
      mese_nascita + "/" + anno_nascita + ", " +
      (coniugato?"coniugato":"celibe") + ", aliquota fiscale: " + aliquota()
      (lavoraIn != null?", lavora presso la ditta " + lavoraIn:
       ", disoccupato");
  }
}
```


Esercizio 3: cliente

Realizzare in Java lo use case *Analisi Statistica* che comprende anche l'operazione *RedditoMedioInGrandiAziende*:

InizioSpecificaUseCase Analisi Statistica

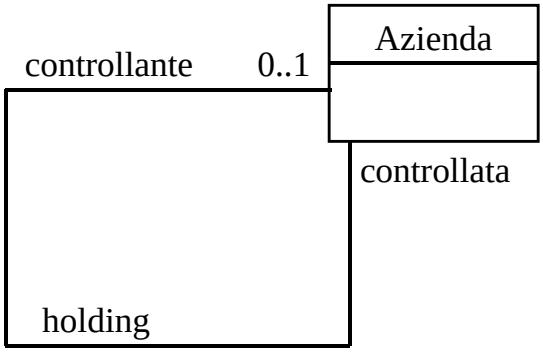
RedditoMedioInGrandiAziende (*i: Insieme(Persona)): reale*
pre: *i* contiene almeno una persona che lavora in una grande azienda
post: *result* è il reddito medio delle persone che lavorano in una grande azienda nell'insieme di persone *i*

FineSpecifica

Rappresentare l'insieme in input semplicemente come un vettore.

Realizzazione delle associazioni: secondo caso

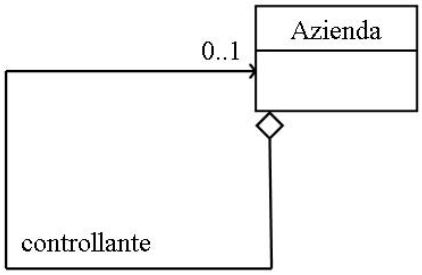
Quanto detto vale anche per il caso in cui l'associazione coinvolga più volte la stessa classe. In questo caso il concetto di responsabilità si attribuisce ai ruoli, piuttosto che alla classe.



Supponiamo che la classe *Azienda* abbia la responsabilità su *holding*, solo nel ruolo *controllata*. Questo significa che, dato un oggetto x della classe *Azienda*, vogliamo poter eseguire operazioni su x per conoscere l'azienda

controllante, per aggiornare l'azienda controllante, ecc. Supponiamo inoltre di non volere mantenere una rappresentazione esplicita dei link della associazione.

Diagramma delle classi realizzativo: secondo caso



Si noti che abbiamo sostituito l'associazione *holding* con l'aggregazione *controllante*.

In generale il nome della aggregazione viene scelto uguale al nome del ruolo (nell'esempio, il nome è `controllante`).

Realizzazione del codice: secondo caso

```
public class Azienda {
  // eventuali attributi .....
  private Azienda controllante; // il nome del campo e' uguale al ruolo
  // altre funzioni ....
  public Azienda getControllante() {
    return controllante;
  }
  public void setControllante(Azienda a) {
    controllante = a;
  }
}
```

Realizzazione delle associazioni: terzo caso

Consideriamo il caso in cui la classe *C* sia l'unica ad avere la responsabilità sull'associazione *A*, e vogliamo mantenere una **reppresentazione dei link** della associazione.

La necessità di rappresentare i link è tipicamente dovuto al fatto che l'associazione *A* abbia **attributi**.

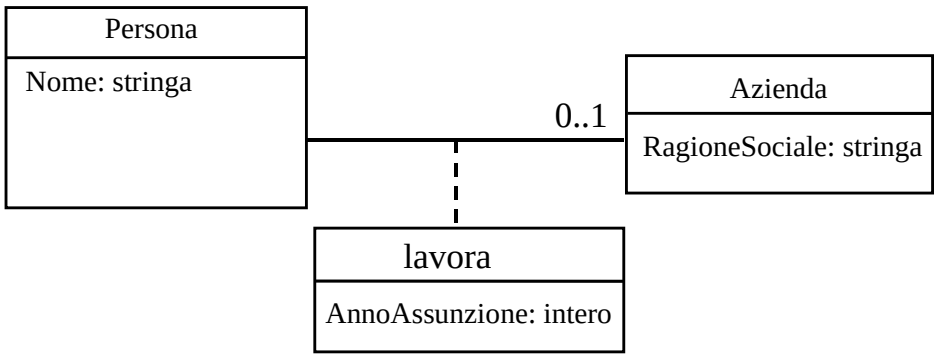
Assumiamo, per ora, che l'associazione abbia uno o più attrivuti di di molteplicità 1..1.

Rimangono le seguenti assunzioni:

- molteplicità 0..1;
- solo una delle due classi *ha responsabilità* sull'associazione (dobbiamo rappresentare **un solo verso** dell'associazione).

Realizzazione delle associazioni: terzo caso

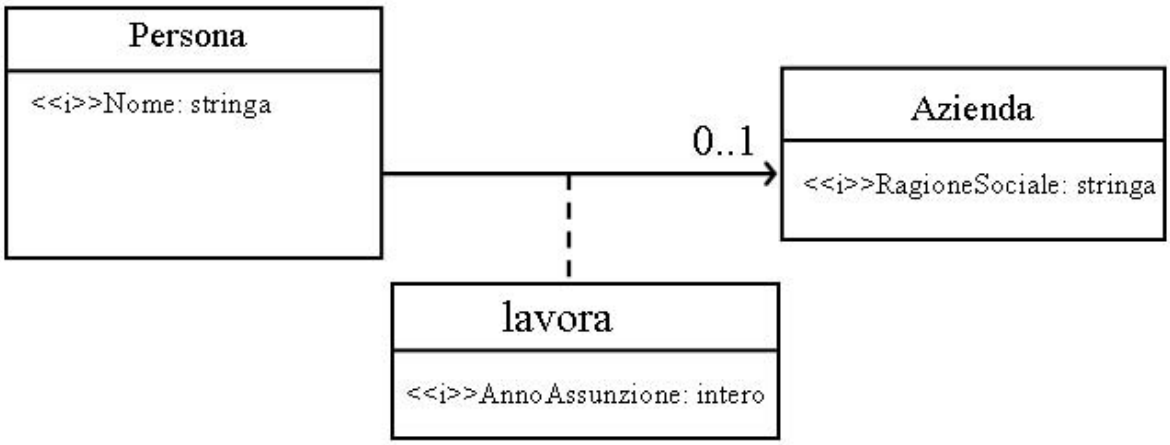
Esempio



Solo *Persona* ha responsabilità sull'associazione, vogliamo mantenere una rappresentazione esplicita dei link per gestire opportunamente gli attributi dell'associazione.

Dia

Esemp



Nel diagramma delle classi realizzativo abbiamo mantenuto l'associazione, tuttavia ne abbiamo ristretto la navigabilità esplicitando che il verso di navigazione che interessa è quello da *Persona* ad *Azienda*.

In questo modo esprimiamo che nella realizzazione del codice manterremo una rappresentazione esplicita dei link dell'associazione, ma ne permetteremo di navigare l'associazione in un'unica direzione.

Attributi di associazione (cont.)

Per rappresentare l'associazione *As* fra le classi UML *A* e *B* con attributi, introduciamo **un'ulteriore classe** Java `TipoLinkAs`, che ha lo scopo di rappresentare i link fra gli oggetti delle classi *A* e *B*. In particolare, ogni link (presente al livello estensionale) fra un oggetto di classe *A* ed uno di classe *B* sarà rappresentato da un oggetto di classe `TipoLinkAs`.

La classe Java `TipoLinkAs` avrà campi dati per rappresentare:

- gli attributi dell'associazione;
- i riferimenti agli oggetti delle classi `A` e `B` relativi al link.

La classe Java `TipoLinkAs` avrà inoltre delle funzioni che consentono di gestire i suoi campi dati (costruttore, funzioni **get**), e la funzione `equals` per verificare l'uguaglianza solo sugli oggetti collegati dal link, ignorando gli attributi.

Attributi di associazione (cont.)

Supponendo che solo la classe UML *A* abbia responsabilità sull'associazione *As*, la classe Java `A` che la realizza dovrà tenere conto della presenza dei link.

Quindi, la classe Java `A` avrà:

- un campo dati di tipo `TipoLinkAs`, per rappresentare l'eventuale link;

  in particolare, se tale campo vale `null`, allora significa che l'oggetto di classe `A` non è associato ad un oggetto di classe `B`;

- opportuni campi funzione che permettono di gestire il link (funzioni `get`, `inserisci`, `elimina`).

Realizzazione in Java della classe TipoLinkLavora

```
// File ParteQuarta/Ass01Attr/TipoLinkLavora.java

public class TipoLinkLavora {
  private final Persona laPersona;
  private final Azienda laAzienda;
  private final int annoAssunzione;

  public TipoLinkLavora(Azienda x, Persona y, int a) {
    laAzienda = x; laPersona = y; annoAssunzione = a;
  }

  public Azienda getAzienda() { return laAzienda; }
  public Persona getPersona() { return laPersona; }
  public int getAnnoAssunzione() { return annoAssunzione; }

  public boolean equals(Object o) {
```



```
    if (o != null && getClass().equals(o.getClass())) {
      TipoLinkLavora b = (TipoLinkLavora)o;
      return b.laPersona != null && b.laAzienda != null &&
             b.laPersona == laPersona && b.laAzienda == laAzienda;
    }
    else return false;
  }
}
```

Realizzazione in Java della classe Persona

```
// File ParteQuarta/Ass01Attr/Persona.java

public class Persona {
  private final String nome;
  private TipoLinkLavora link;
  public Persona(String n) { nome = n; }
  public String getNome() { return nome; }
  public void inserisciLinkLavora(TipoLinkLavora t) {
    if (link == null && t != null &&
        t.getAzienda() != null && t.getPersona() == this)
      link = t;
  }
  public void eliminaLinkLavora() {
    link = null;
  }
  public TipoLinkLavora getLinkLavora() { return link;  }
}
```

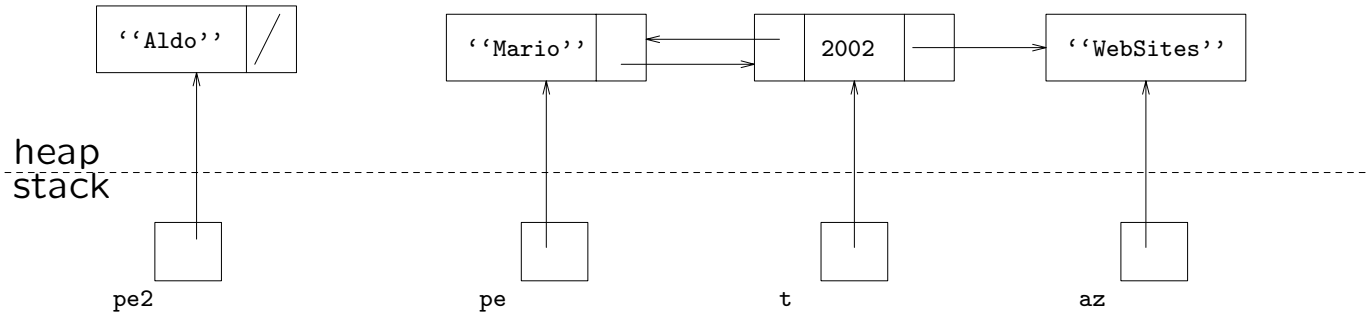

Considerazioni sulle classi Java

- Si noti che i campi dati nella classe `TipoLinkLavora` sono tutti `final`.

  Di fatto un oggetto della classe è *immutabile*, ovvero una volta creato non può più essere cambiato.

- La funzione `inserisciLinkLavora()` della classe `Persona` deve assicurarsi che:
  - la persona oggetto di invocazione non partecipi già ad un link della associazione *lavora*;
  - l'oggetto che rappresenta il link esista;
  - l'azienda a cui si riferisce il link esista;
  - la persona a cui si riferisce il link sia l'oggetto di invocazione.

Possibile stato della memoria



Due oggetti di classe `Persona`, di cui uno che lavora ed uno no.

Realizzazione della situazione di esempio

```
public static void main (String args[]) {

  Azienda az = new Azienda("WebSites");
  Persona pe = new Persona("Mario");
  Persona pe2 = new Persona("Aldo");

  TipoLinkLavora t = new TipoLinkLavora(az,pe,2002);

  pe.inserisciLinkLavora(t);
}
```

Esercizio 4: cliente

Realizzare in Java lo use case *Ristrutturazione Industriale*:

InizioSpecificaUseCase **Ristrutturazione Industriale**

AssunzioneInBlocco *(i: Insieme(Persona), a: Azienda, an: intero)*
pre: nessuna
post: tutte le persone nell'insieme di persone *i* vengono assunte dall'azienda *a* nell'anno *an*

AssunzionePersonaleEsperto *(i: Insieme(Persona), a: Azienda, av: intero, an: intero)*
pre: nessuna
post: tutte le persone nell'insieme di persone *i* che lavorano in un'azienda qualsiasi fin dall'anno *av* vengono assunte dall'azienda *a* nell'anno *an*

FineSpecifica

Rappresentare l'insieme in input semplicemente come un vettore.

Metodi naive di realizzazione

Se abbiamo una associazione con attributi ma non siamo interessati alla rappresentazione esplicita dei link possimo realizzare tale associazione in modo utilizzando un campo dati per l'associazione e un campo dati addizionale per ciascuno degli attributi. Il risultato però è che tale realizzazione va fatta ad-hoc caso per caso, facendo uno sforzo per far si che il campo dati che rappresenta (implicitamente) l'associazione e quelli che rappresentano i suoi attributi mantengano informazioni allineate.

Consideriamo l'esempio discusso in precedenza

```
public class Persona {
  // eventuali attributi .....
  private Azienda lavora;
  private int annoAssunzione;
  // altre funzioni ....
  public Azienda getLavora() { return lavora; }
```

```
  public int getAnnoAssunzione() {
    if (a=null) throws new RuntimeException(‘‘valore non significativo’’);
    else return annoAssunzione;
  }
  public void setLavora(Azienda a, int x) {
    if (a != null) { lavora = a; annoAssunzione = x; }
  }
  public void eliminaLavora() { lavora = null; }
}
```

Osservazioni sul metodo naive

La funzione `SetLavora()` ha ora due parametri, perché nel momento in cui si lega un oggetto della classe `C` ad un oggetto della classe `D` tramite `A`, occorre specificare anche il valore dell'attributo dell'associazione (essendo tale attributo di tipo 1..1).

Il cliente della classe ha la responsabilità di chiamare la funzione `getAnnoAssunzione()` correttamente, cioè quando l'oggetto di invocazione `x` effettivamente partecipa ad una istanza della associazione `lavora` (`x.getLavora() != null`).

Il fatto che l'attributo dell'associazione venga realizzato attraverso un campo dato della classe `C` non deve trarre in inganno: concettualmente l'attributo appartiene all'associazione, ma è evidente che, essendo l'associazione 0..1 da `C` a `D`, ed essendo l'attributo di tipo 1..1, dato un oggetto `x` di `C` che partecipa all'associazione `A`, associato ad `x` c'è uno ed un solo valore per l'attributo. Quindi è corretto, in fase di implementazione, attribuire alla classe `C` il campo dato che rappresenta l'attributo dell'associazione.