Breve introduzione a UML

M.Moro - Dip. di Elettronica e Informatica - Univ. di Padova - Ottobre 1999

1. UML

Nelle fasi di analisi, progettazione e implementazione di un'applicazione risulta particolarmente utile disporre di uno strumento grafico di progetto mediante il quale descrivere, fin dalle prime fasi di analisi e fino alla codifica nel linguaggio di programmazione scelto, le caratteristiche dell'applicazione, con un livello di dettaglio di volta in volta regolabile (inizialmente basso, elevato immediatamente prima della codifica). Uno di questi strumenti, affermatosi di recente come standard, è UML (*Unified Modeling Language*), come dice il nome un linguaggio di progettazione universale, sviluppato da Rational: http://www.rational.com/uml/index.jtmpl per sistemi *Object-oriented*. Una descrizione succinta è reperibile in: http://www.analisi-disegno.com/introuml/welcome.html
Esistono anche *tool* in grado di supportare il programmatore nell'uso di UML, uno particolarmente interessante perché in Java, in grado di generare codice Java, e freeware è ARGO/UML:
http://www.ics.uci.edu/pub/arch/uml/
Per gli scopi del corso, vengono qui brevemente introdotti ed esemplificati i **diagrammi delle classi** in una versione ridotta e con convenzioni compatibili con ARGO/UML, sufficiente come ausilio alla progettazione di semplici applicazioni e come stimolo per approfondire l'argomento in un momento successivo, utilizzando il materiale e la bibliografia che si trovano in rete.

2. STRUTTURA DEL DIAGRAMMA DELLE CLASSI

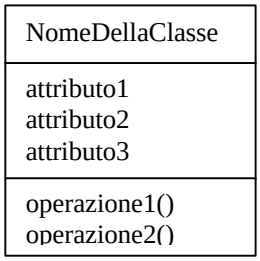
Questa componente di UML pone l'enfasi alla descrizione statica delle classi e degli oggetti, assieme alle loro relazioni, che formano il progetto. Non appare descritto l'andamento nel tempo delle attività, pur potendo includere descrizioni di entità (classi, oggetti, ecc.) che hanno a che fare con il tempo. Nell'utilizzare il formalismo che viene qui proposto va tenuto presente che, nel passare dall'analisi alla realizzazione, il livello di dettaglio può aumentare e può darsi che qualche aspetto della descrizione debba subire una trasformazione e/o semplificazione (in particolare, come si vedrà, le associazioni).

Classe
Operativamente un'applicazione è vista come una collezione di oggetti che collaborano interagendo l'un l'altro. La classe è l'entità che esprime le caratteristiche comuni (struttura, comportamento, relazioni) ad un sottoinsieme di oggetti simili, che svolgono sostanzialmente il medesimo tipo di funzioni pur avendo ciascuno la propria identità. Una classe è caratterizzata da:

- Un nome
  Inizia con lettera maiuscola ed è privo di caratteri di sottolineatura (*undescore*)
- Attributi
  Un insieme di caratteristiche i cui valori identificano un oggetto istanza della classe e ne costituiscono lo stato; ogni attributo ha un nome, che inizia con lettera minuscola, ed opzionalmente un tipo (classe)
- Operazioni
  Un insieme di funzionalità che esprimono il carattere operativo costituendone il comportamento di un oggetto; ogni operazione ha un nome, che inizia con lettera minuscola, e opzionalmente la dichiarazione di un prototipo (interfaccia di attivazione). A seconda del livello di dettaglio possono comparire o meno i cosiddetti *costruttori* e *distruttori*, ovvero operazioni dedicate alla creazione-inizializzazione e distruzione di istanze della classe.

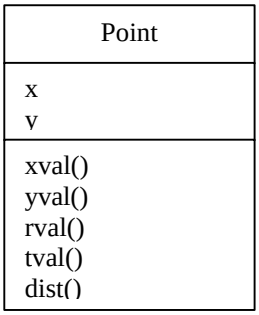
La classe è rappresentata da un rettangolo diviso in 3 sezioni (fig. 1), una per il nome (obbligatorio), una per la lista di attributi e una per la lista di operazioni.

Fig. 1



Esempio:
Pensando di includere nella classe `Point`, che rappresenta un punto sul piano cartesiano, le due coordinate `x` e `y` come attributi e le operazioni `xval` (ritorna l'ascissa), `yval` (ritorna l'ordinata), `rval` (ritorna il modulo), `tval` (ritorna l'angolo), la classe è rappresentabile con il digramma di fig. 2a. In questa prima forma vengono specificati solo i nomi di attributi e operazioni. In fig. 2b la stessa classe rappresentata più in dettaglio, ove sono specificati i tipi degli attributi, i parametri e i valori di ritorno delle operazioni, con sintassi vicina a Java, e con l'aggiunta di un costruttore.

Fig. 2a



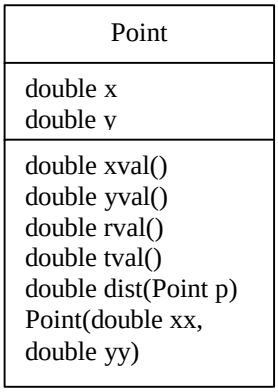


Fig. 2b

Oggetto

Un oggetto è un'istanza identificabile di una classe. È rappresentato da un rettangolo al cui interno compare il nome completo dell'istanza nella forma: `NomeClasse nomeIstanza`. Talvolta `nomeIstanza` può essere omesso se serve evidenziare solo il tipo dell'oggetto. La presenza di diagrammi di oggetti assieme a diagrammi di classe serve in alcuni casi per descrivere lo stato dell'applicazione in una determinata situazione (ad esempio una struttura di dati con i vari elementi in un certo istante). La specificazione della classe-tipo di un oggetto esprime implicitamente anche la relazione, di tipo 'è un'istanza di' (*instance-of*), esistente tra l'oggetto e la sua classe.

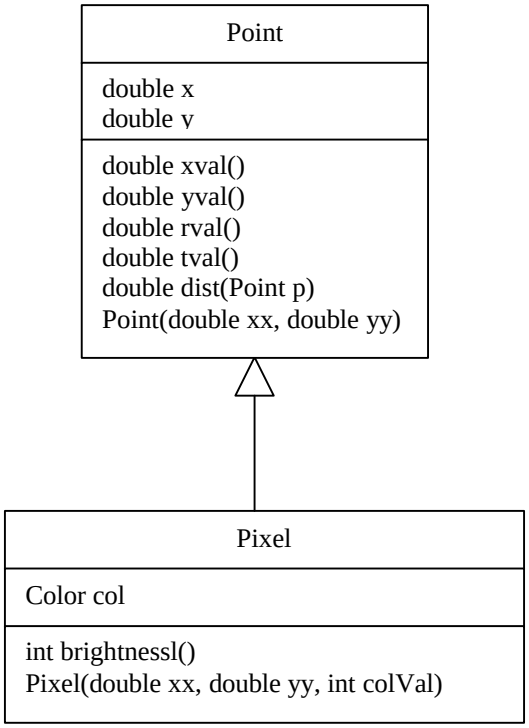
Esempio:
In fig. 3 due oggetti di classe `Point`.

Fig. 3

Ereditarietà

Una classe A si dice *superclasse* di una classe B, detta *sottoclasse*, quando B eredita le caratteristiche di A ma ne modifica alcune ridefinendole e/o aggiunge delle altre. In questo senso si può anche dire che una sottoclasse è una *specializzazione* di una superclasse e che una superclasse è una *generalizzazione* di una sottoclasse. L'ereditarietà è una relazione tra classi che si preferisce tenere distinta dalle relazioni (associazioni) di altro tipo per la sua particolare semantica, realizzata esplicitamente in tutti i linguaggi *object-oriented*. Questa relazione è di tipo 'è un' (*is-a*): ad esempio la classe `Impiegato` è sottoclasse della classe `Persona` e si può affermare che un impegato *è una* persona.
L'ereditarietà è espressa da una linea, terminante con un triangolo, che va dalla sottoclasse alla superclasse. In fig. 4 la classe `Pixel` eredita le caratteristiche di `Point`, di cui è specializzazione (un *pixel* è un punto colorato), aggiungendo l'indicazione del colore, un proprio costruttore e un'operazione `brightness` (che è specifica della classe `Pixel`).

Fig. 4



Associazione

Una associazione (o **relazione**) è un legame logico tra classi che esprime qualche forma di correlazione. Ad esempio una classe `Corso` che descrive un corso è in associazione con una classe `Studente` e l'associazione, vista dalla parte del corso, ha la semantica 'è frequentato da' e dalla parte dello studente la semantica 'frequenta'. La semantica è espressa da un **nome** per l'associazione e normalmente è un verbo o un predicato verbale (ad esempio *frequenta*). È anche possibile associare un ruolo a ciascuna delle due classi in relazione: il ruolo è solitamente un nome o un predicato nominale (ad esempio un oggetto `Persona` può assumere il ruolo di *insegnante* in una relazione con `Corso`).

Una associazione è in generale diversa da un attributo non elementare (cioè da un attributo che è istanza di una classe) nel senso che, anche se un attributo esprime implicitamente un legame con la classe di cui è istanza, se si trascura un'associazione, le classi originariamente collegate possono egualmente mantenere la loro identità, mentre la perdono in generale se si omette un attributo (ad esempio `Persona` e `Corso` hanno singolarmente significato anche senza l'associazione *insegna*, ma la classe `Persona` perde identità se si trascura il suo attributo `Cognome`).

Un'associazione può esplicitare anche una **molteplicità** che esprime il numero di oggetti che partecipano all'associazione stessa. I valori possibili sono:

- molti (*)
- uno (1)
- zero o più (0..*)
- uno o più (1..*)
- zero o uno (0..1)
- all'interno di un range (2..4)

In particolare, attraverso la molteplicità, si può specificare se si tratta di un'associazione uno-uno, uno-molti, molti-uno, molti-molti. Per l'esempio sopra, volendo descrivere la situazione dei singoli corsi, l'associazione ha molteplicità 1 dalla parte del corso e molti dalla parte dello studente (ogni corso può avere molti studenti): essa facilita l'elencazione degli studenti di un corso ma non quella dei corsi frequentati da un certo studente. Se si volesse descrivere la situazione di uno studente, l'associazione avrebbe la molteplicità 1 dalla parte dello studente e molti (* oppure 1..n con n un certo valore) dalla parte del corso (ogni studente può seguire più corsi): essa facilita l'elencazione dei corsi seguiti da uno studente. Se si volesse invece descrivere un intero curriculum, l'associazione diventerebbe molti-molti.

Un'associazione può esplicitare delle informazioni aggiuntive, distintive della relazione che esprime. Ad esempio l'associazione *insegna* potrebbe specificare il numero di ore insegnato. Per associazioni diverse da molti-molti, i dati di associazione si possono tradurre in attributi della classe che ha molteplicità molti (l'attributo `ore` andrebbe nella classe `Corso` per la relazione *insegna*); in alternativa i dati di associazione possono costituire attributi di una classe, detta **classe di associazione**, appositamente definita e distinta dalle classi in relazione, ottenendo una più precisa separazione delle informazioni. La soluzione con la classe di associazione è obbligatoria nel caso di relazioni molti-molti: ad esempio, se si volesse registrare i voti di tutti i corsi seguiti dai vari studenti, questi dati non sarebbero direttamente associabili come attributi ad alcuna delle due classi in relazione (un singolo voto non può essere attributo né di un oggetto `Corso` né di un oggetto `Studente`).

Due classi possono esibire più di una associazione (associazioni **multiple**): ad esempio un oggetto `Persona` potrebbe essere associato ad un oggetto `Corso` da una relazione *insegna* (la persona è un docente) e da una relazione *frequenta* (la persona è uno studente). È anche ammessa un'associazione riflessiva: ad esempio è riflessiva l'associazione *è prerequisito* per un oggetto `Corso`, con molteplicità 0..* da entrambe le parti.

Nel caso un'associazione coinvolga una superclasse, la relazione è ereditata dalle eventuali sottoclassi. Queste ultime possono partecipare ad altre relazioni.

Un'associazione è rappresentata da una linea di connessione tra i rettangoli che descrivono le classi. L'eventuale nome dell'associazione va posto sopra la linea in posizione intermedia, con l'indicazione del verso che deriva dalla semantica dell'associazione se non derivabile altrimenti; gli eventuali ruoli e la molteplicità vanno invece posti vicino a ciascuna delle due estremità. La molteplicità pari a 1 può essere omessa. Se è presente una classe di associazione, questa va collegata con una linea tratteggiata alla linea che esprime tale associazione.

In fig. 5a compaiono le classi `Studente` e `Corso` con la molteplicità che può conseguire dalla semantica *frequenta*. L'attributo `voto` è un dato di associazione incluso nella classe con molteplicità non uno. In fig. 5b la stessa associazione è corredata da un attributo `voto` che è un dato di associazione incluso nella classe con molteplicità non uno. Questo modo di rappresentare il dato di associazione è fuorviante e funziona solo se almeno una delle due classi presenta molteplicità unitaria. In fig. 6 vi è una visione alternativa e più corretta, valida anche con un insieme di studenti: viene introdotta la classe di associazione `Valutazione`. In fig. 7 le classi `Persona` e `Corso` con indicato il ruolo di *Insegnante*.

Fig. 5a

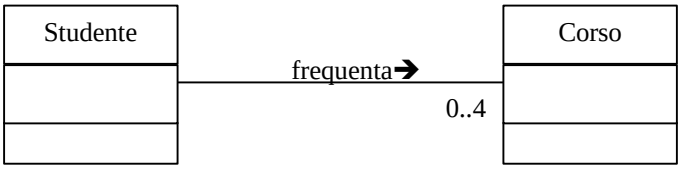


Fig. 5b

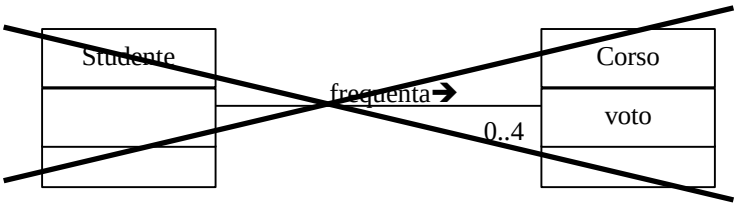


Fig. 6

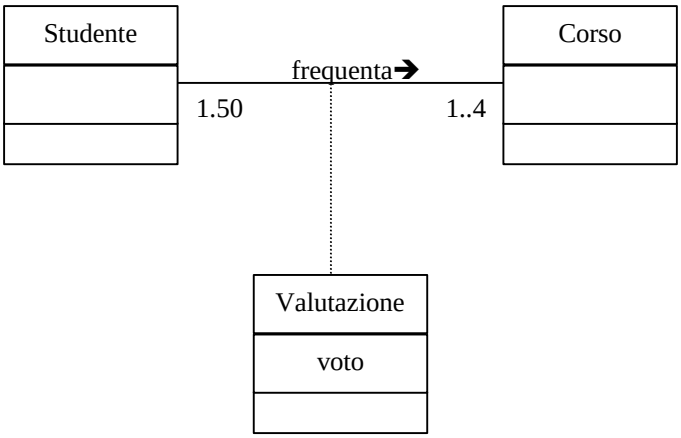
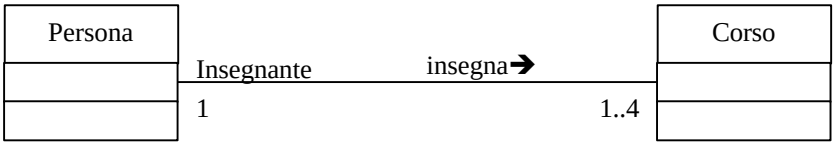


Fig. 7



Aggregazione
L'aggregazione è un particolare tipo di associazione che risponde alla specifica semantica 'è parte di' (*part-of*), che si ha quando un insieme è relazionato con le sue parti. Un caso tipico è quello delle classi **contenitore**, frequentemente utilizzate in svariati contesti (ad esempio l'interfaccia grafica). La rappresentazione di un'aggregazione si differenzia da quella dell'associazione per la presenza di un rombo sulla terminazione della linea di connessione dalla parte di quella delle due classi che funge da contenitore.
In fig. 8 si vede una classe `Punto` che si può interpretare come parte di una classe `Poligono`. La molteplicità esprime il fatto che un poligono include 3 o più punti identificativi (i vertici) e che un punto può essere vertice di molti poligoni.

Fig. 8

3. VERSO L'IMPLEMENTAZIONE

Le convenzioni introdotte finora consentono di esplicitare le caratteristiche salienti di un progetto identificando classi (attributi e operazioni) e associazioni. Per arrivare all'implementazione in un linguaggio di programmazione, Java nel nostro caso, è necessario passare ad uno o più stadi successivi eseguendo alcuni fondamentali interventi:

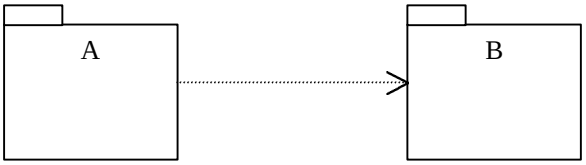
- Completare la dichiarazione delle classi indicando per tutti gli attributi il tipo, l'eventuale valore iniziale, e per tutte le operazioni (metodi) il prototipo di interfaccia completo
- Arricchire le dichiarazioni con ulteriori specificazioni caratteristiche della programmazione *object-oriented:* ad esempio criteri di visibilità; distinzione tra attributi di istanza e attributi di classe (`static`), questi ultimi in unica istanza; attributo di 'astratta' per una classe collegato ad un analogo attributo per i metodi.
- Studiare una opportuna implementazione delle associazioni-aggregazioni, soprattutto nel caso di relazioni molti-molti.

In merito al secondo punto, è possibile utilizzare alcune specifiche ulteriori notazioni per rappresentare entità finora non citate.

Package

Il *package* è una raccolta di classi fra loro correlate. Un *package* è rappresentato da un rettangolo con etichetta ed è identificato dal suo nome. Tipicamente ciascun *package* viene poi dettagliato con una diagramma delle classi che lo compongono, assieme alle loro relazioni. Nel diagramma con i *package* è possibile specificare legami di **dipendenza**, rappresentati con linea tratteggiata orientata che collega i *package*: una dipendenza da A a B *importa* significa che A usa le risorse messe a disposizione da B (fig. 9).

Fig. 9

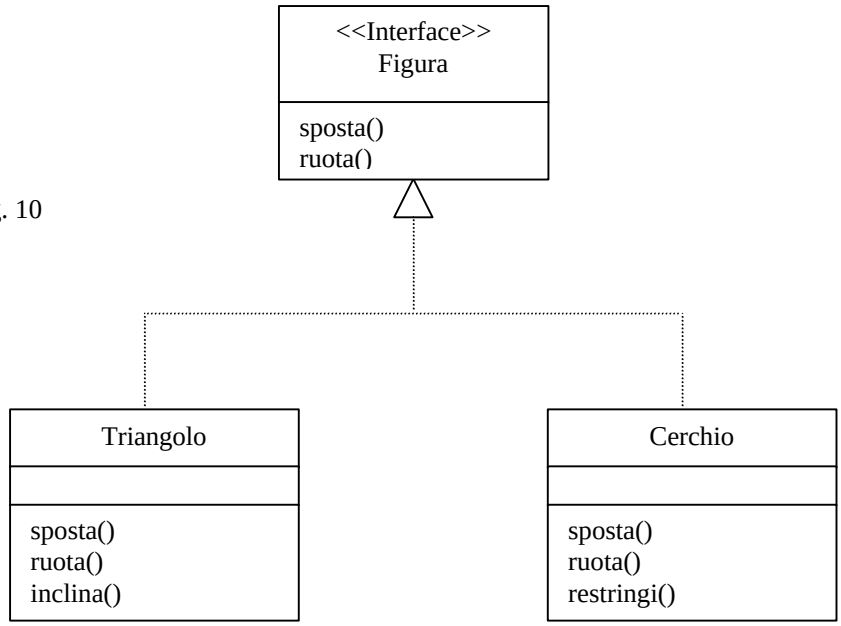


Interfaccia, implementazione

Java introduce il concetto di interfaccia come 'promessa di implementazione' di un insieme di metodi: in altri termini si tratta di una classe (completamente) astratta che può essere implementata da una o più classi realizzanti. Da un certo punto di vista, l'implementazione di un'interfaccia è un particolare tipo di derivazione per la quale non vengono ereditati attributi e, affinché la classe realizzante non sia astratta e possa quindi essere istanziata, tutti i metodi devono essere ridefiniti con implementazioni effettive. Queste particolarità giustificano la necessità di tener distinta la rappresentazione per interfacce e implementazioni rispetto alla derivazione.

Un'interfaccia è rappresentata da un rettangolo con due sezioni, quella contenente il nome (iniziale maiuscola) e la specificazione <<`Interface>>`, e una con i metodi da implementare. Una classe che implementa una interfaccia è collegata all'interfaccia con una linea terminante con triangolo, come per la derivazione, ma tratteggiata. In fig.10 l'interfaccia `Figura` prevede che si implementino i metodi `ruota()` e `sposta()`; le classi `Triangolo` e `Cerchio` la implementano.

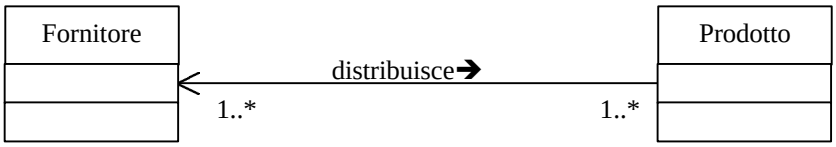
Fig. 10



Per il terzo punto, occorre dapprima osservare che una associazione (similmente per una aggregazione) è in fase di analisi bidirezionale. Nella fase di progetto e implementazione può spesso trasformarsi in monodirezionale e si dice in questo caso che l'associazione è **navigabile**. Una associazione navigabile è spesso più facile da implementare rispetto a una bidirezionale. Il verso di navigazione tipicamente dipende dalla frequenza con cui si 'percorre' l'associazione in quel verso piuttosto che nel verso opposto. Per esempio, una associazione molti-molti `Fornitore/Prodotto` giustifica un verso dal `Prodotto` al `Fornitore` se risulta più frequente la domanda: 'quali fornitori distribuiscono un determinato prodotto?', essendo il prodotto il dato noto e il/i fornitori il dato da ricavare, essendo invece più rara la

domanda: 'quali prodotti distribuisce un certo produttore?'. L'associazione navigabile è rappresentata da una linea orientata nel verso di navigazione (fig. 11), che può coincidere o meno con il verso semantico (➔) dell'associazione.

Fig. 11



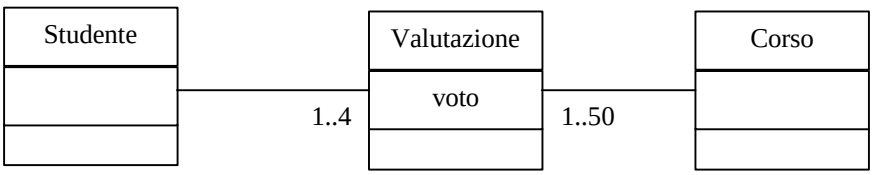
Per quanto riguarda l'implementazione delle associazioni, queste si traducono in generale aggiungendo qualche campo (attributo), solitamente non pubblico, ad una o ad entrambe le classi in relazione. Nel caso di una associazione navigabile, l'aggiunta va fatta solo alla classe **sorgente**, ad entrambe invece se bidirezionale. Se la molteplicità della classe collegata a quella in cui si fa l'aggiunta è 1 (anche 0..1), è sufficiente aggiungere un riferimento alla classe collegata; se la molteplicità è invece molti, è necessario prevedere un set di riferimenti, che può essere implementato con un vettore o con altra classe contenitore adeguata, tenendo eventualmente conto che, nel caso di una molteplicità m..n, il numero massimo di riferimenti utili è n. Se il valore minimo della molteplicità è 0, è possibile che non sia associato alcun oggetto della classe collegata: in questa situazione, nel caso di un riferimento semplice, questo può essere pari a `null`, mentre nel caso di un contenitore, questo può essere vuoto. È buona norma in questi casi prevedere un metodo che valuti la presenza o meno di almeno un legame.
Lo strumento ARGO/UML, partendo dal diagramma di fig. 5b, genererebbe ad esempio il seguente codice-scheletro. Si noti il vettore `frequenta` aggiunto alla classe `Studente` che aveva molteplicità 1 e il riferimento semplice `frequenta` nella classe `Corso` che aveva molteplicità maggiore di 1. Si noti anche il metodo `studia()`, che è stato successivamente aggiunto in `Studente`, che può servire a valutare se vi è almeno un legame con un oggetto di classe `Corso`.

```
public class Studente {

  // Attributes

  // Associations
  /**
 *
 */
    protected Vector  frequenta;

  // Operations
  /** An operation that does ...
 *
 * @param firstParamName  a description
of this parameter
 */
  private boolean studia() {
  return false;
  }
} /* end class Studente */
```

```
public class Corso {

  // Attributes
  /** An attribute that represents ...
 */
  private int voto;

  // Associations
  /**
 *
 */
    protected Studente  frequenta;

  // Operations

} /* end class Corso */
```

Se il diagramma include classi di associazione, è necessario operare una trasformazione intermedia, sostituendo la relazione originaria con due relazioni tra ciascuna delle classi inizialmente collegate e la classe di associazione. Le classi originarie assumono molteplicità 1 mentre la molteplicità della classe di associazione verso la prima classe è pari a quella che aveva la seconda e quella verso la seconda è pari a quella della prima. In fig. 12 si vede la trasformazione da apportare per il caso di fig. 6.

Fig. 12



Ogni istanza di `Valutazione` memorizzerà un voto di un certo studente per un certo corso. Il corrispondente scheletro in Java è il seguente:

```
public class Studente {

  // Attributes

  // Associations
  /**
 *
 */
    protected Vector
myValutazione;

  // Operations

} /* end class Studente */
```

```
public class Valutazione {

  // Attributes
public int voto;

  // Associations
  /**
 *
 */
    protected Studente
myStudente;
  /**
 *
 */
    protected Corso myCorso;

  // Operations

} /* end class Valutazione */
```

```
public class Corso {

  // Attributes

  // Associations
  /**
 *
 */
    protected Vector
myValutazione;

  // Operations

} /* end class Corso */
```