Azzolini Riccardo 2019-10-17

Class diagram – Altre associazioni

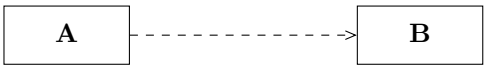
1 Stereotipi delle associazioni

In UML, uno *stereotipo* è una specializzazione di un simbolo/concetto esistente, che ne specifica il "vero significato".

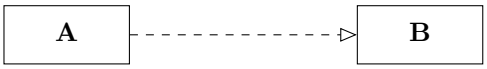
Tutte le possibili relazioni tra classi sono stereotipi dell'associazione. Siccome alcune di queste sono molto comuni e importanti, per convenienza UML mette a disposizione delle apposite notazioni grafiche.

Alcuni stereotipi dell'associazione sono:

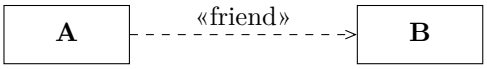
- **dipendenza** (USES): A dipende da B



- **realizzazione/raffinamento** (che corrisponde, in Java, alla relazione "implementa" tra classe e interfaccia): A raffina B

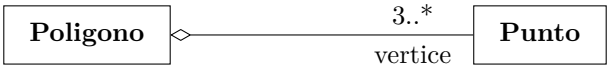


- classe `friend` (C++), per la quale non esiste un apposito simbolo, quindi bisogna utilizzare direttamente la notazione stereotipale, segnandola come un caso particolare di dipendenza:

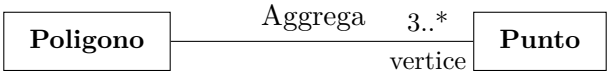


2 Aggregazione e composizione

L'**aggregazione** è una forma particolare (stereotipo) di associazione, che corrisponde a una relazione di tipo IS_COMPONENT_OF. Graficamente, essa si indica con



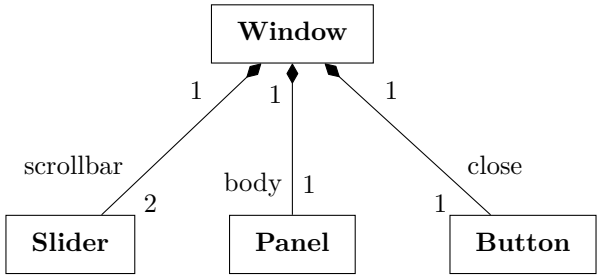
che, usando la notazione convenzionale delle associazioni, equivale a:



Una relazione di **composizione** è un'*aggregazione forte*:

- le parti componenti non esistono senza il contenitore;
- il contenitore ha proprietà esclusiva dei componenti.

Di conseguenza, la molteplicità dei contenitori per ciascun componente deve essere esattamente 1. Un esempio sono le parti di una finestra di un'interfaccia grafica:



2.1 Proprietà

Sia per l'aggregazione che per la composizione, vale la proprietà **transitiva**. Ad esempio, se il PC contiene un'unità base, e questa contiene la CPU, allora il PC contiene la CPU.

Inoltre, per la composizione valgono anche:

- Proprietà **asimmetrica**: una classe componente *non* può contenere la classe contenitore. Ad esempio, se il PC contiene un'unità base, questo implica che l'unità base non contenga il PC.
- **Propagazione degli effetti**: se si effettuano operazioni di copia o eliminazione sul contenitore, queste devono essere eseguite anche sugli oggetti componenti. Ad esempio:
  - se si copia un documento composto di capitoli, devono essere copiati anche i singoli capitoli;
  - se si distrugge un PC, si distruggono anche tutte le sue componenti.

  Questo può richiedere attenzione in fase di implementazione, perché *non è corretto* copiare solo i riferimenti agli oggetti componenti (cosa che avviene di default in Java), ma bisogna invece creare vere e proprie copie degli oggetti, per evitare che uno stesso componente appartenga a due contenitori (l'originale e la sua copia).

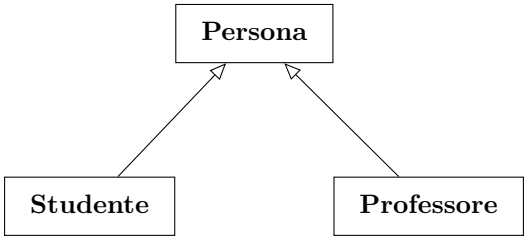
2.2 Distinguere aggregazione e associazione

L'aggregazione è un caso particolare di associazione. Ci sono vari criteri che aiutano a distinguerle:

- Contenimento fisico o semplice riferimento? In alcuni linguaggi (come ad esempio C++, ma non Java), un oggetto può contenere fisicamente altri oggetti (e non solo dei riferimenti ad essi). Questo indica che si tratta di un'aggregazione/composizione, perché una normale associazione comporta solo il semplice mantenimento di riferimenti tra oggetti.
- È transitiva (e asimmetrica)?
- Le operazioni sull'intero sono automaticamente applicate alle parti (propagazione degli effetti)?
- Ciclo di vita: un oggetto composito deve gestire la creazione e la distruzione delle parti, mentre nell'associazione non c'è dipendenza esistenziale tra le parti.
- C'è propagazione di attributi? Essa indica che non si tratta né di aggregazione/composizione né di un'associazione normale, bensì di una relazione di ereditarietà.

3 Ereditarietà

In UML, la relazione di **ereditarietà** (**generalizzazione/specializzazione**) si indica con la seguente notazione grafica:



Nota: La freccia punta *verso la superclasse*.

Le sottoclassi ereditano attributi, operazioni e associazioni della superclasse.

3.1 Partizione in sottoclassi

In una relazione di generalizzazione, si possono specificare alcuni vincoli su come le istanze della superclasse vengono partizionate in sottoclassi. L'insieme di sottoclassi a cui si applicano questi vincoli si chiama **generalization set**.

Come vincoli, si può specificare che un generalization set è:

- **overlapping** (con sovrapposizioni) se un'istanza può appartenere a più sottoclassi, o **disgiunto** (disjoint) se un'istanza può appartenere a un'unica sottoclasse;
- **completo** se tutte le istanze della superclasse appartengono almeno a una delle sottoclassi definite nel set, o **incompleto** se possono esistere istanze della superclasse che non appartengono a una delle sottoclassi nel set.

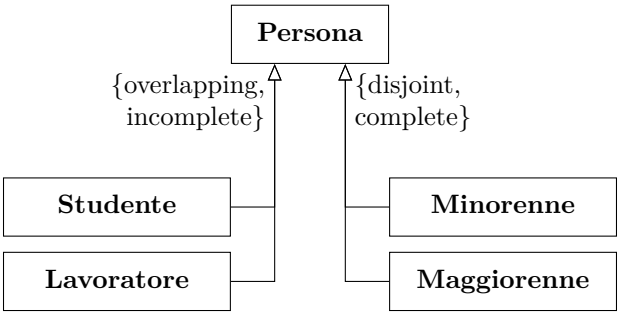
Se non specificato, di default un generalization set è overlapping[1] e incompleto.

Una stessa superclasse può avere più generalization set (gerarchie) separati: i vincoli di ciascun set riguardano solo le sottoclassi appartenenti ad esso.

Questi vincoli forniscono informazioni utili, ad esempio, per:

- determinare se una classe deve essere astratta;
- capire come gestire l'eventuale possibilità di sovrapposizioni.

3.1.1 Esempio



Il generalization set formato da Studente e Lavoratore è:

- *overlapping*, perché ci sono persone che sono sia studenti che lavoratori;
- *incompleto*, perché esistono persone che non sono né studenti né lavoratori.

Invece, il generalization set formato da Minorenne e Maggiorenne è:

- *disgiunto*, perché una persona non può essere sia minorenne che maggiorenne;

[1] Nelle versioni precedenti di UML, il default era disgiunto, ma poi è stato cambiato a overlapping.

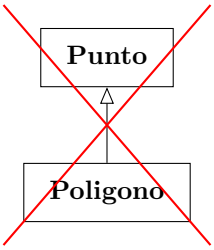
- *completo*, perché ogni persona è minorenne o maggiorenne.

3.2 Distinguere aggregazione/composizione e generalizzazione

- L'aggregazione corrisponde al verbo "avere", mentre la generalizzazione corrisponde al verbo "essere".
- L'aggregazione riguarda gli oggetti (ad esempio, un oggetto Persona ha un oggetto Indirizzo), mentre la generalizzazione riguarda le classi (ad esempio, la classe Studente è una generalizzazione della classe Persona, quindi l'insieme di tutte le sue istanze è un sottoinsieme delle istanze di Persona).

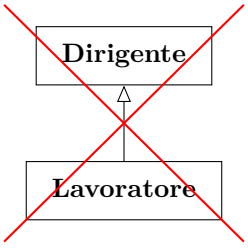
3.3 Controesempi e delega

L'ereditarietà non è sempre la relazione corretta. Ad esempio, è *sbagliato* porre Poligono come sottoclasse di Punto:



Infatti, un poligono è basato su dei punti, ma l'ereditarietà vorrebbe dire che "un poligono è un punto", e ciò ovviamente non è vero. In questo caso, la relazione corretta è invece l'aggregazione.

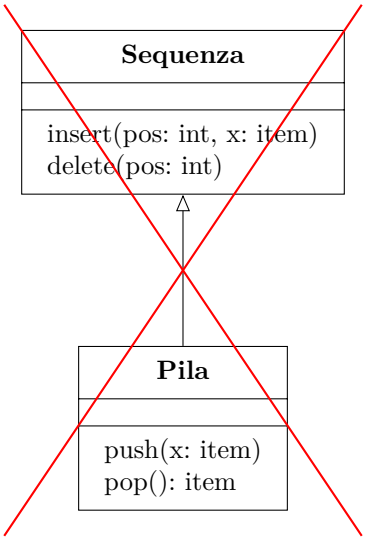
Un altro esempio di errore è:



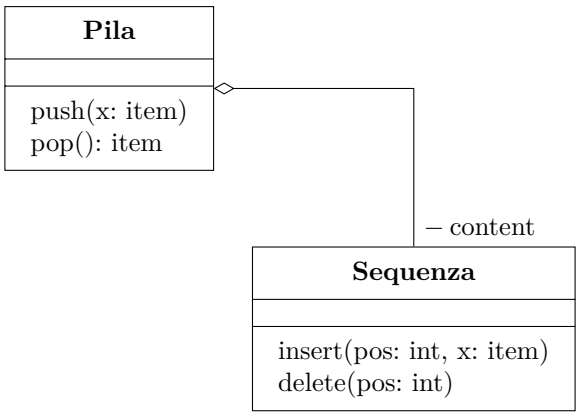
Questa relazione di generalizzazione "imita" l'organigramma aziendale, ma, in UML, significa che un Lavoratore è un caso particolare di Dirigente, o, in altre parole, che tutti i lavoratori sono dirigenti: ciò non è assolutamente corretto (semmai è vero l'inverso).

Un altro errore è usare una relazione di ereditarietà per riutilizzare le funzionalità di una classe, anche se la sottoclasse non è veramente una specializzazione della superclasse. Ad

esempio, se si vuole implementare una struttura dati Pila avendo a disposizione una classe Sequenza, si potrebbe pensare che Pila sia una specializzazione di Sequenza, con in più la proprietà LIFO. In realtà, però, la Pila ha anche delle proprietà *in meno*, poiché consente l'accesso solo all'elemento in cima, ma come sottoclasse di Sequenza erediterebbe tutti i metodi di quest'ultima, che permettono di accedere agli elementi in qualsiasi posizione. L'ereditarietà non è quindi la relazione corretta.



Per la realizzazione di Pila, la classe Sequenza può invece essere sfruttata attraverso una **delega**: una Pila contiene una Sequenza, mediante un'aggregazione *privata* (perché l'uso della Sequenza è solo un dettaglio dell'implementazione, che non deve essere esposto).



3.4 Raccomandazioni sull'overriding

- Far ereditare le operazioni "get" invariate nella forma esterna, cioè con la stessa segnatura.

- Far ereditare in tutte le sottoclassi le operazioni "set", in modo da avere la possibilità di impostare tutte le proprietà definite nella superclasse.

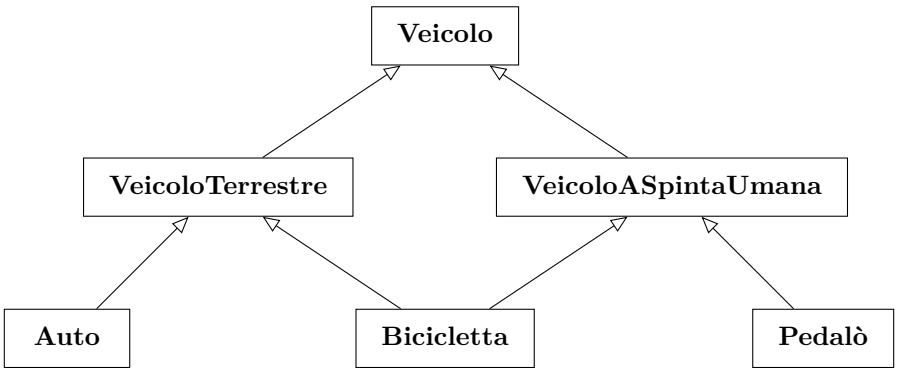
  Questo non vale, però, per quelle operazioni "set" che potrebbero violare una restrizione introdotta dalla sottoclasse. Ad esempio, se si ha una classe `Rettangolo` con un metodo `modificaBase`, quest'ultimo deve essere bloccato nella sottoclasse `Quadrato`, mediante l'overriding con un metodo che, ad esempio:

  - non ha effetto,
  - oppure modifica anche l'altezza.

- *Un comportamento un'interfaccia*: tutte le operazioni che hanno lo stesso nome devono condividere la stessa interfaccia esterna (segnatura) e lo stesso comportamento macroscopico.

3.5 Ereditarietà multipla

In UML, l'ereditarietà multipla è consentita, e per certe situazioni costituisce la modellizzazione più corretta. Ad esempio, una Bicicletta è sia un VeicoloTerrestre che un VeicoloASpintaUmana:



Con l'ereditarietà multipla, però, alcuni attributi, operazioni, e/o associazioni possono essere ereditati più volte da una stessa classe. In questo caso:

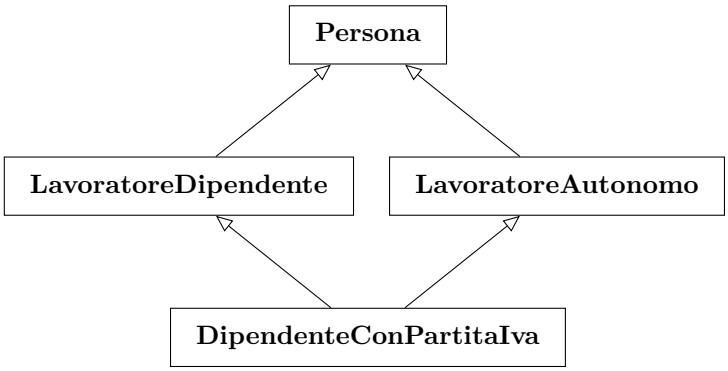
- si ha uno spreco di memoria, dovuto ai dati duplicati;
- la scelta di quale metodo eseguire tra quelli ereditati può essere ambigua.

Un altro problema è l'*ereditarietà multipla accidentale*. Supponendo, ad esempio, di definire Assistente come sottoclasse di Studente e MembroDiFacoltà, *non è vero* che gli oggetti istanza sia di Studente che di MembroDiFacoltà siano sempre assistenti: una stessa persona potrebbe essere studente presso un'università e membro di facoltà presso un'altra, e quindi non essere un assistente in nessuna delle due.

Infine, ci sono linguaggi (come ad esempio Java) che non supportano l'ereditarietà multipla.

Esistono varie soluzioni per evitare questi problemi, che possono anche essere combinate tra loro:

- Semplificare il modello, ove possibile. Ad esempio, si potrebbe decidere di definire Bicicletta come sottoclasse solo di VeicoloASpintaUmana.
- Usare una classificazione nidificata. Ad esempio, VeicoloTerrestre potrebbe avere le sottoclassi VeicoloTerrestreAMotore e VeicoloTerrestreASpintaUmana.
- Usare l'ereditarietà multipla, ma bloccando con gli strumenti del linguaggio l'eredità di attributi/operazioni non necessari.
- Usare la delega invece dell'ereditarietà. Ad esempio, invece di specializzare una Persona in LavoratoreDipendente, LavoratoreAutonomo, e DipendenteConPartitaIva, che è sottoclasse di entrambi,



si potrebbe delegare a un'apposita entità RapportoDiLavoro, specializzata poi in LavoroDipendente e LavoroAutonomo. Un dipendente con partita IVA potrebbe allora avere due istanze di RapportoDiLavoro (una per tipo), evitando così l'ereditarietà multipla.