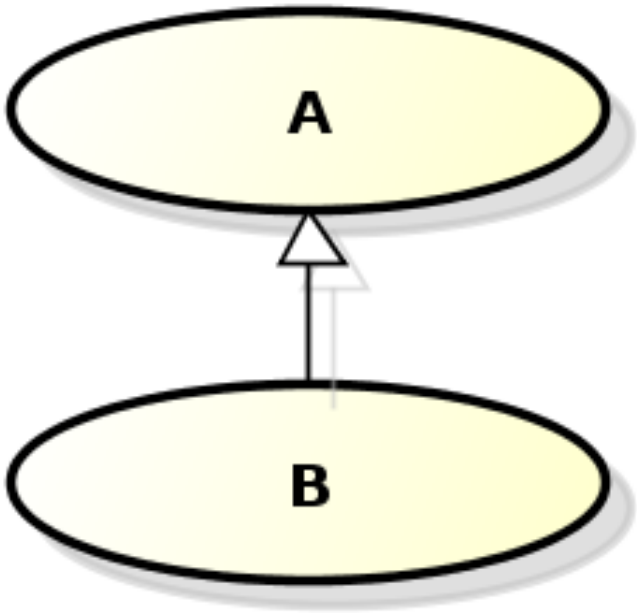
Lezione #10

Lunedì, 28 Ottobre 2013

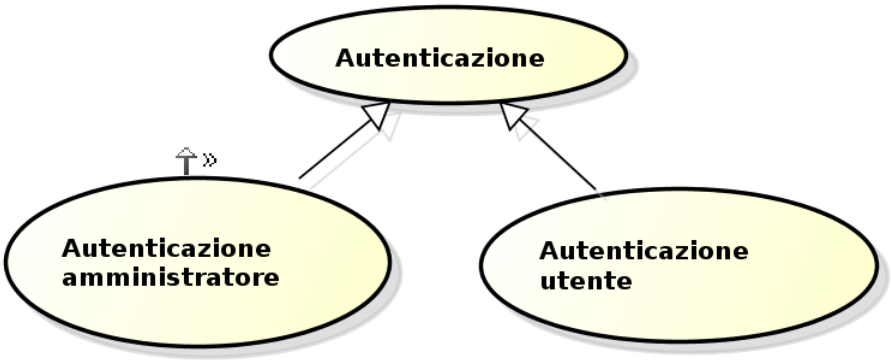
Christian Cardin, 09:30-11:15

Luca De Franceschi

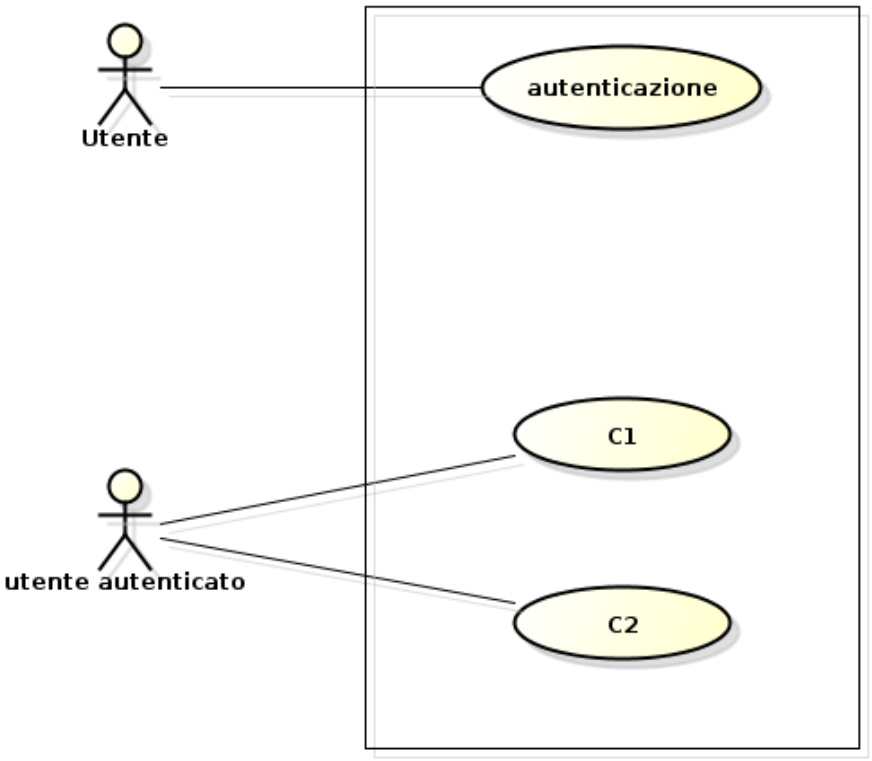
Generalizzazione, si applica molto bene agli attori, si parla di **ereditarietà**.



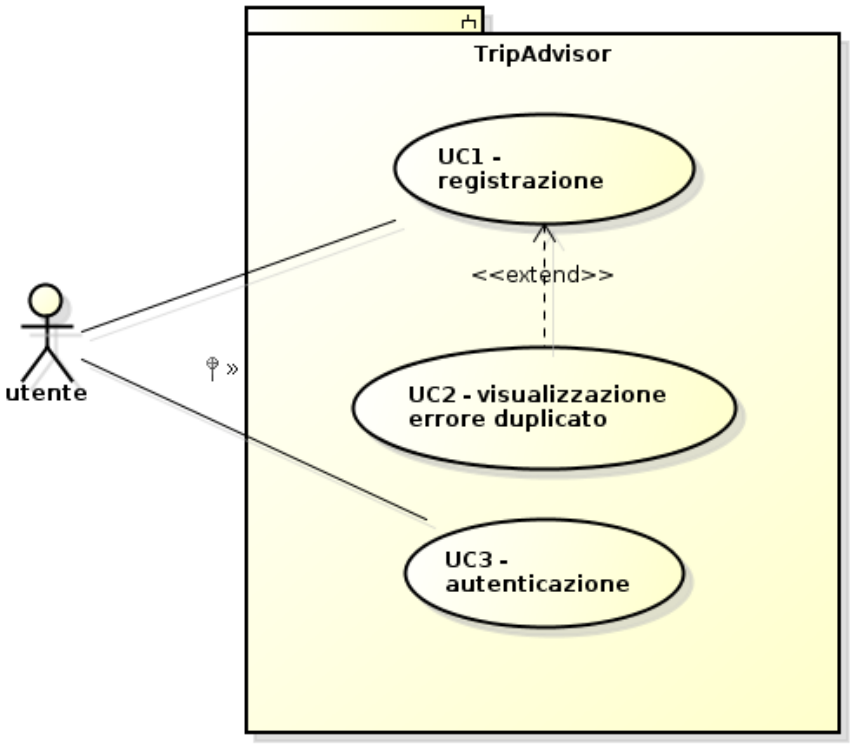
Se un attore B estende un attore A l'attore B accede a tutte le funzionalità dell'attore A più le sue caratteristiche. Es. l'amministratore è allo stesso tempo un utente normale ma in più ha altri privilegi. La generalizzazione tra casi d'uso è meno usata.



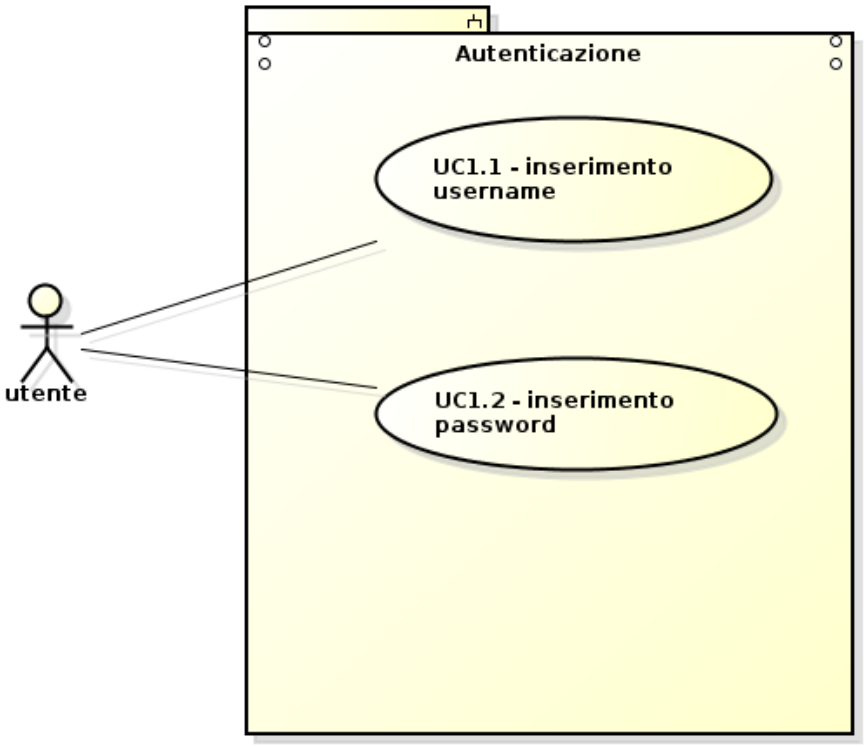
La generalizzazione fra attori è molto più comune e facile da realizzare:



L'utente non può essere una generalizzazione di un utente autenticato perchè l'utente autenticato non può effettuare autenticazione.
Esempio: l'attore utente può accedere alla funzionalità *autenticazione* del sistema:

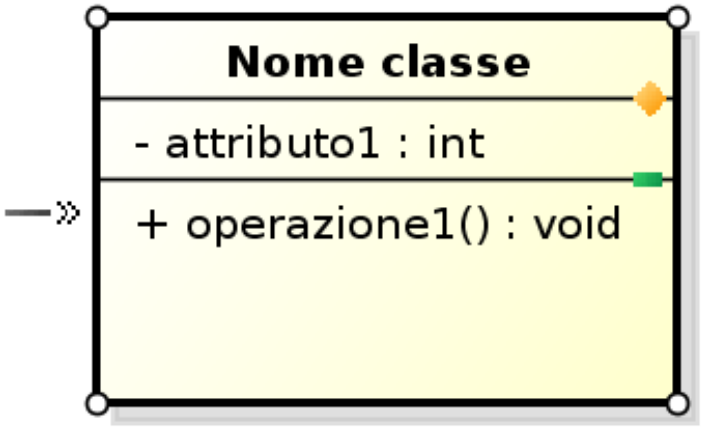


Scendiamo di dettaglio, meno astratti:



Diagrammi delle classi e degli oggetti

Il paradigma più utilizzato è il paradigma ad oggetti, perchè modella molto bene la realtà. Ho una lista di requisiti, ora il progettista deve produrre i requisiti e descrivere l'architettura del prodotto, e dovrà descriverla in maniera formale, in modo che i programmatori sviluppino esattamente quello che il progettista ha pensato. Posso in questo modo garantire alcune proprietà. Ho bisogno dunque di un linguaggio per parlare ai programmatori. Una classe è una descrizione di qualcosa e l'oggetto è un'istanza che rispetta questa descrizione. Si passa dalla descrizione a qualcosa di tangibile.

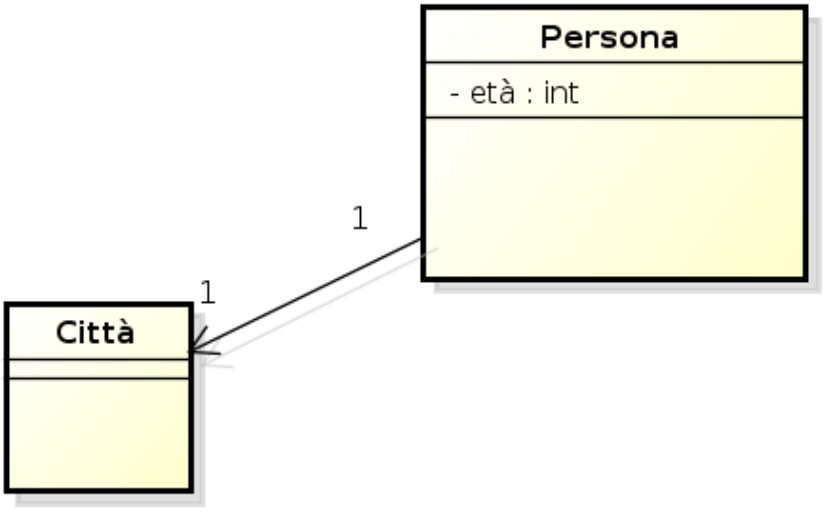


Modella un concetto ed è indipendente dal linguaggio di programmazione con cui andrò a implementare. La prima cosa che andremo a definire di una classe sono i suoi attributi, che vanno scritti nella parte centrale.

```
Visibilità nome :  tipo [molteplicità] = default [proprietà aggiuntive]
```

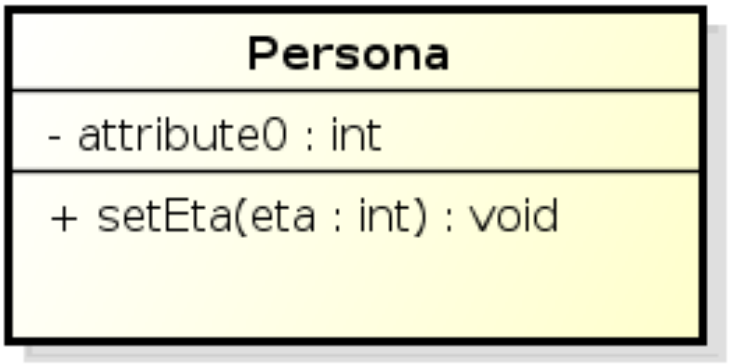
Questa è la segnatura per la visibilità degli attributi:

- +, pubblica;
- -, privata;
- #, protetta.

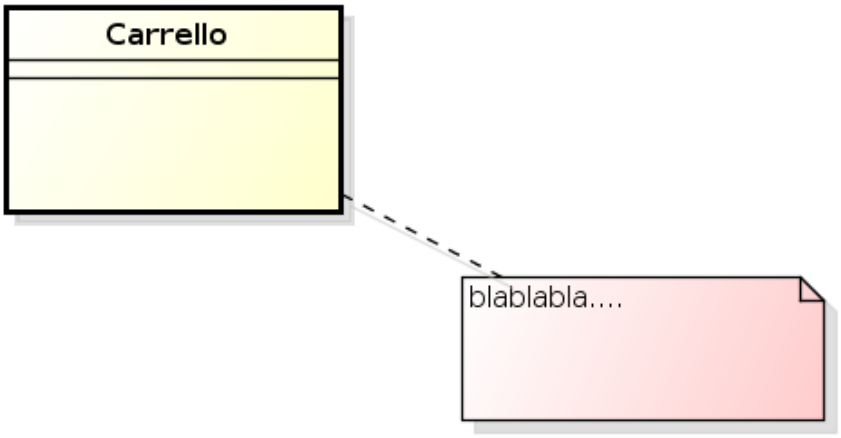


La molteplicità è 1 perchè una persona ha solo un'età. L'attributo può essere anche espresso come **associazione** tra due tipi. Questo si fa con una freccia orientata dalla classe che contiene una copia dell'altro tipo. Associazioni senza verso sono bidirezionali. Si utilizzano gli attributi testuali per i tipi primitivi (nella libreria del linguaggio che stiamo utilizzando), mentre si utilizzano le associazioni quando ci si riferisce a due classi del nostro dominio. Se abbiamo molteplicità superiore a 1 significa che abbiamo una collezione (array, liste, ...). Possono esserci delle convenzioni per gli attributi (es. definizione obbligatoria dei metodi *setter* e *getter*. Le proprietà sono gli attributi e le associazioni.
Le operazioni sono ciò che la classe espone verso l'esterno, sono i "servizi" della classe.

`Visibilità nome (lista-parametri` : tipo-ritorno proprietà aggiuntive}
`Lista-proprietà := direzione nome : tipo = default`



Le **query** sono tutte le operazioni che non modificano l'oggetto di invocazione, a differenza dei metodi modificatori. Operazione != metodo, concetto differente in presenza di polimorfismo.
Commenti e note

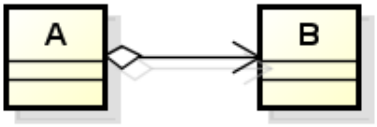


Un concetto fondamentale è la dipendenza fra due tipi. Una classe A dipende da B se una modifica fatta a B implica una modifica ad A. Le dipendenze sono il "male assoluto" e vanno minimizzate, perchè le classi devono essere autoconsistenti. Più dipendenze ho e più una modifica può creare *side-effect* su un'altra classe (problemi in fase di manutenzione). Un modo per minimizzare le dipendenze è l'uso di interfacce.
Le dipendenze in UML sono di vario tipo, c'è bisogno di un classificatore, al fine di comprenderla meglio. Questo si inserisce come etichetta nella freccia.

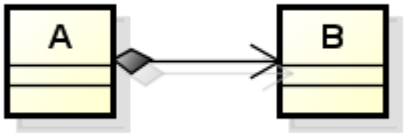
Parola chiave	Significato
«call»	La sorgente invoca un'operazione della classe destinazione.
«create»	La sorgente crea istanze della classe destinazione.
«derive»	La sorgente è derivata dalla classe destinazione
«instantiate»	La sorgente è una istanza della classe destinazione (meta-classe)
«permit»	La classe destinazione permette alla sorgente di accedere ai suoi campi privati.
«realize»	La sorgente è un'implementazione di una specifica o di una interfaccia definita dalla sorgente
«refine»	Raffinamento tra differenti livelli semantici.
«substitute»	La sorgente è sostituibile alla destinazione.
«trace»	Tiene traccia dei requisiti o di come i cambiamenti di una parte di modello si colleghino ad altre
«use»	La sorgente richiede la destinazione per la sua implementazione.

L'aggregazione e la composizione sono particolari tipi di associazione. L'aggregazione si identifica con la frase "*parte di...*", gli aggregati possono essere condivisi. Viene rappresentato con un diamante vuoto. La composizione è come l'aggregazione ma le istanze i un'aggregazione possono appartenere solo ad un aggregato. Solo l'oggetto intero può creare e distruggere le sue parti.

A possiede un'istanza di B

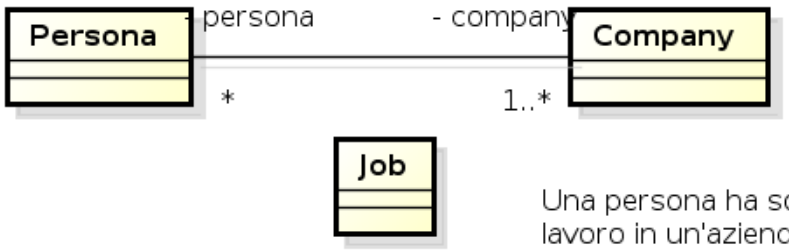


A possiede un insieme di B



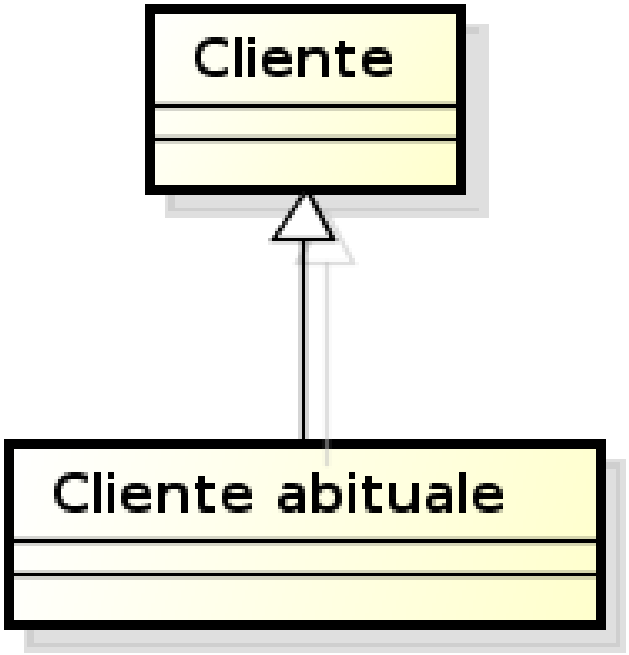
A possiede un insieme di istanze di B

Può succedere che un'associazione abbia bisogno di essere specificato maggiormente. In questo caso si creano **classi di associazioni**, che aggiungano attributi ed operazioni alle associazioni. Ma i linguaggi i programmazione non prendono un'implementazione di queste cose.

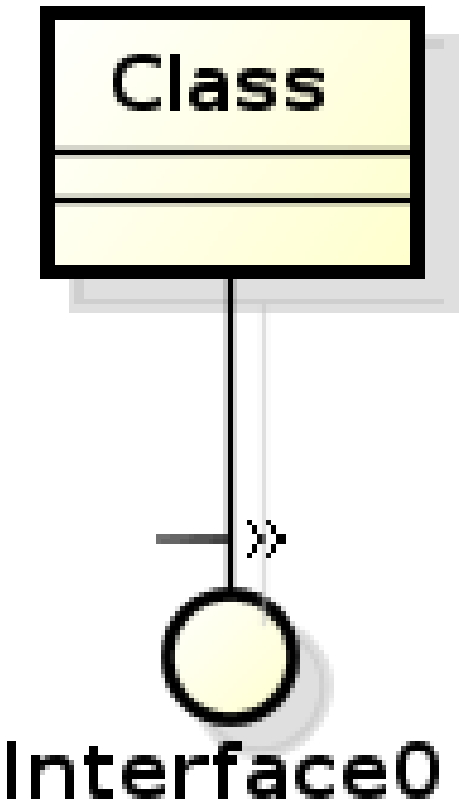


Una persona ha solo un lavoro in un'azienda

La generalizzazione è un concetto molto importante perchè descrive l'ereditarietà, uno dei concetti fondamentali della programmazione a oggetti. A generalizza B se ogni oggetto di B è anche un oggetto di A. Sottotipo != Sottoclasse.



Per le classi astratte si usa il nome in *corsivo*, non può essere instanziata perché ha delle operazioni che non possiedono l'implementazione, anche se ne può possedere alcune implementate.
Un altro concetto è quello di **interfaccia**, che non è una classe (al massimo è un tipo) ed è priva di implementazione. Il loro scopo è definire un contratto che le classi che la implementeranno devono assolutamente fornire.


Class
Interface0