Diagrammi delle classi(3)

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. I diagrammi delle classi modellano un concetto ed sono indipendenti 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:

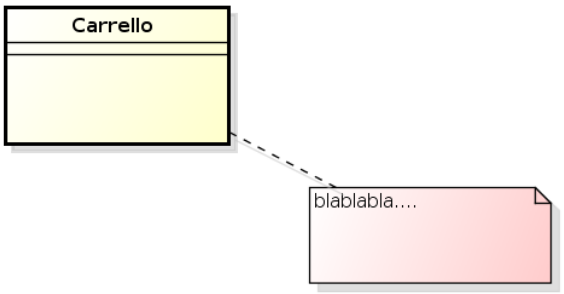
- +, pubblica;
- -, privata;
- #, protetta.

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 è bene cercare di metterli privati.
Associazioni senza verso sono bidirezionali, evitarle se possibile e usare un nome non un verbo che le descrivano. Si utilizzano gli attributi testuali per i tipi primitivi, 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, ...). 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.

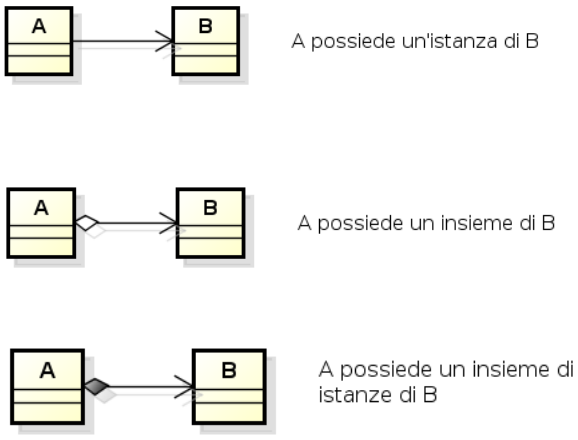
Commenti e note



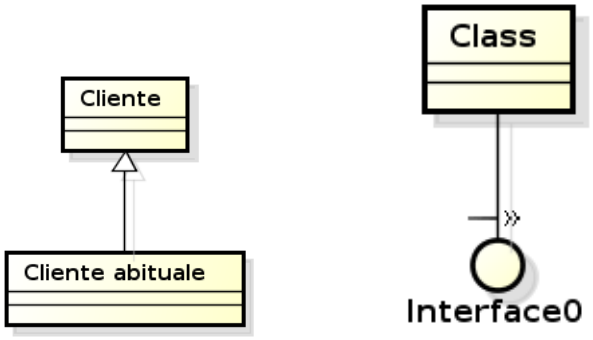
Un concetto fondamentale è la dipendenza fra due tipi. Una classe A dipende da B se una modifica fatta a B implica una modifica ad A. Le dipendenze vanno minimizzate, perchè le classi devono essere autoconsistenti. Più dipendenze ho e più una modifica può creare *side-effect* su un'altra classe (problemi 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.

Relazioni:

- *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.



Si possono aggiungere ulteriori attributi alle associazioni
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. L'ereditarietà multipla è supportata, attenzione da usare pochissimo.



Per le classi astratte e altro si usa il *corsivo*, la classe 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.