UML2

Ereditarietà e Polimorfismo

Andrea Polini

Ingegneria del Software
Corso di Laurea in Informatica

Generalizzazione e Specializzazione

Generalizzare

v. tr. e intr. [der. di generale[1]]. - 1. tr. Rendere generale; estendere, applicare a un intero gruppo di persone o di cose ciò ha valore particolare o si riferisce al singolo: g. un'usanza, una convinzione, un principio, un metodo. In partic., g. un giudizio, un'affermazione, attribuire (per lo più a torto) valore generale a giudizi o affermazioni sperimentali validi solo per casi particolari; in questo senso, anche con uso assol.: fai male a g.: se uno di loro ha tradito, non devi ritenerli tutti traditori. 2. intr. (aus. avere) Parlare in modo generico, senza precisare o scendere in particolari *Treccani.it*

Specializzare

v. tr. [der. di speciale, sull'esempio del fr. spécialiser]. - 1. Indirizzare un'attività verso un settore specifico e ben delimitato, allo scopo di far acquisire maggiore competenza, efficienza, qualificazione: s. un'industria chimica nella preparazione di prodotti farmaceutici; bisogna s. ulteriormente l'istruzione professionale. . . . 2. Nel linguaggio della matematica e delle sue applicazioni, con sign. affine a quello di specificare, restringere a un caso particolare quanto si era più genericamente enunciato: s. la terna di riferimento. . . . *Treccani.it*

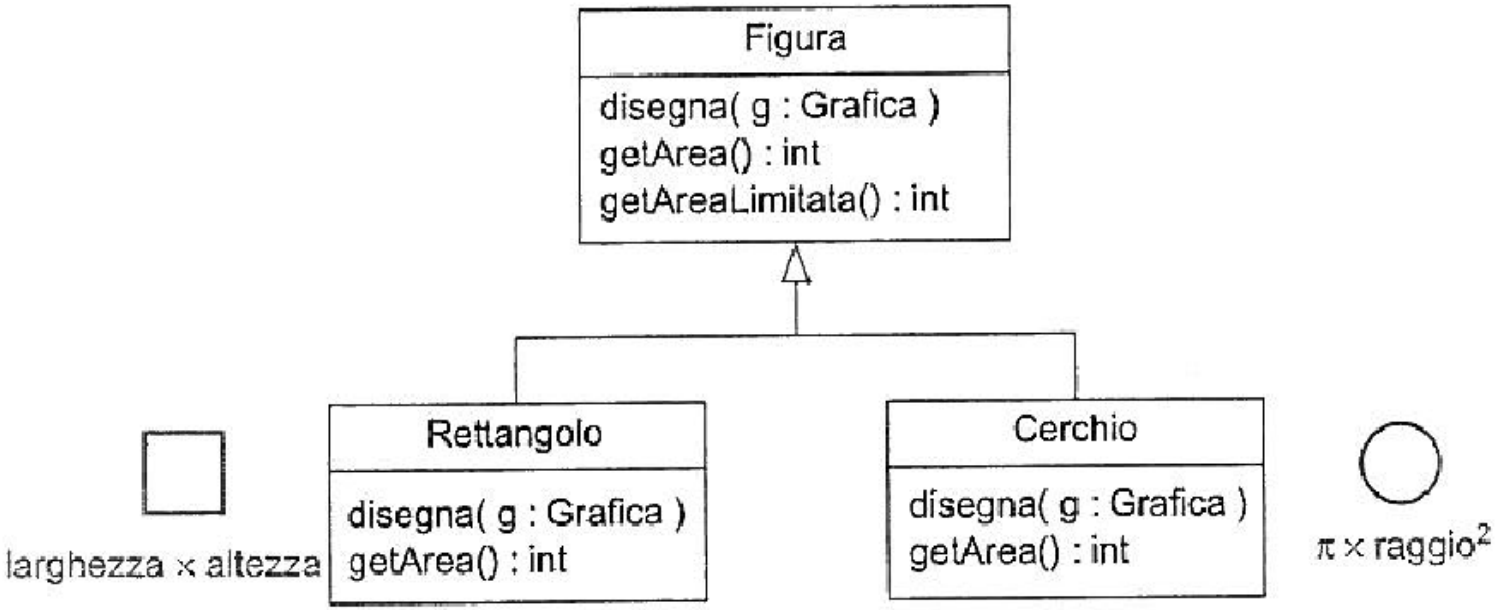
Ereditarietà

Fase di analisi sarebbe bene identificare concetti generali per poi procedere a specializzazioni.
Le sottoclassi specializzano ereditando:

- attributi
- operazioni
- relazioni
- vincoli

Le sottoclassi possono poi ridefinire (overriding) quanto definito nella superclasse. In molti casi non ha senso definire comportamento di un metodo direttamente nella superclasse. D'altra parte in alcuni casi il comportamento è generale per tutte le classi che possono essere ridefinite a partire dalla superclasse (caso delle operazioni astratte)

Esempio di Ereditarietà - Ridefinizione ed Astrazione



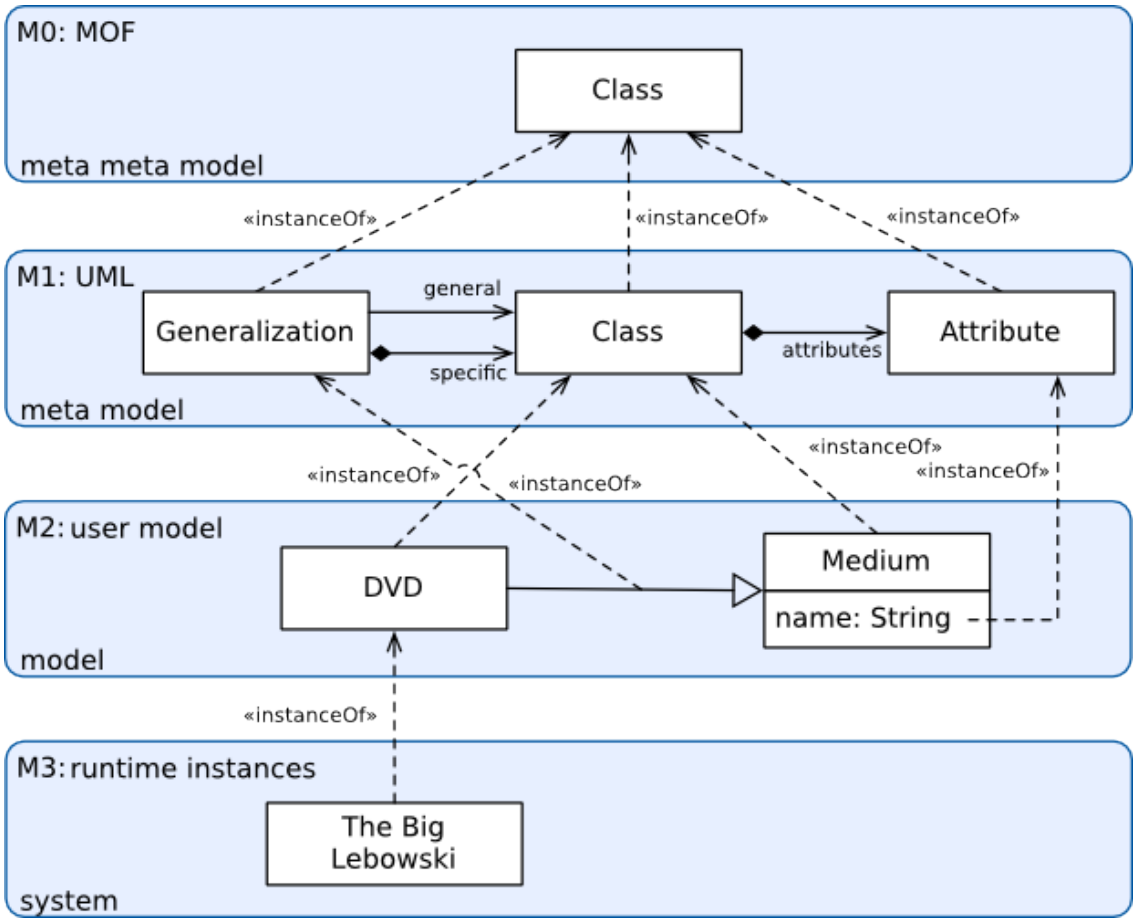
Livelli di astrazione

In generale è bene utilizzare gerarchie di generalizzazione consistenti ponendo ad uno stesso livello astrazioni comparabili.

In qualche modo gli insiemi che ogni astrazione definisce a quel livello possono essere utilizzati per categorizzare elementi ad un livello più basso.

Meta-modelli e modelli

Con riferimento ad un'altro aspetto di astrazione è importante chiarire la distinzione tra modelli, meta-modelli e meta-meta modelli.



Polimorfismo

etimologicamente risulta un composto di “poli-” e “morphe” dal significato di “dalle molte forme”. Nel nostro contesto indica la capacità di assumere differenti comportamenti dipendentemente dal contesto.

È possibile avere un comportamento specifico da parte degli oggetti invocati senza che l’oggetto cliente sia a conoscenza della natura specifica dell’oggetto invocato. In particolare quando polimorfismo è associato a dynamic binding
Il cliente si aspetta soltanto che venga rispettato il contratto.

Ridefinizione delle operazione concrete della classe base ed il problema della classe base fragile. Uso di `final` in Java
`http://en.wikipedia.org/wiki/Fragile_base_class`

Insieme di generalizzazione

- completo
- incompleto
- disgiunto
- sovrapposto

Linguaggi di programmazione non forniscono tipicamente meccanismi per la definizione di insiemi di generalizzazione. Dunque il concetto rimane tipicamente annotato nei soli modelli di analisi. Se sono ritenuti particolarmente importanti possono essere introdotti livelli di astrazione ulteriori nella gerarchia di generalizzazione.

Polimorfismo

Concetti correlati al polimorfismo che sono presenti in molti linguaggi di programmazione:

- polimorfismo parametrico
- overloading

Principi generali di buona progettazione OO

Ereditarietà e polimorfismo

Ereditarietà definisce una classe in termini di un'altra. Principalmente meccanismo di riuso del codice di tipo white-box. È bene che il programmatore conosca il codice della classe base

Principio di sostituzione di Liskov: un oggetto di una sottoclasse deve poter essere utilizzato dove sia atteso un'oggetto della superclasse

Meccanismo molto potente ma presenta alcune caratteristiche da maneggiare con cura:

- Ridefinizione di superclassi può rompere "contratto" delle sottoclassi
- Ereditarietà multipla di classe ed il problema del diamante

È preferibile applicare "ereditarietà" al livello delle interfacce

Composizione di Oggetti

La definizione di composizione tra oggetti è una tecnica alternativa all'ereditarietà al fine di ottenere riuso.

Funzionalità complesse sono ottenute componendo oggetti. Per fare questo è necessario definire precise interfacce.

Composizione utilizzando interfacce porta ad un riuso black-box rispetto ad ereditarietà che conduce invece a riuso white-box.

Ereditarietà pro e contro: (+) direttamente supportata dai linguaggi e dunque possibilità di verifica statica. (-) Non è possibile modificare comportamento a run-time. (-) Dettagli implementativi della superclasse vengono esposti alle sottoclassi

Composizione pro e contro: (+) riuso black-box. (+) inferiori dipendenze implementative (+) creazione e binding quando necessario (-) tipicamente molti più oggetti a run-time (-) comportamento complessivo non identificabile in una sola classe

Principi generali di buona progettazione OO

Meccanismo della delega

Due tipi di oggetti delegante che rinvia lo svolgimento di un determinato compito ad un oggetto delegato (relazione analoga a classe e sottoclasse)

Il meccanismo della delega (has-a vs is-a) permette di ottenere le stesse caratteristiche di riuso garantite dall'ereditarietà

Caso di una classe Window e di una classe Rettangolo per definirne forma e dimesioni. Un "Window" è un rettangolo oppure ha un rettangolo?

Dunque attraverso il meccanismo della delega si può ottenere lo stesso riuso ottenibile con ereditarietà di classe ma con il vantaggio di avere una maggiore dinamicità a run-time.

Principi generali di buona progettazione OO

Composizione di oggetti consiste nell'assemblare differenti oggetti al fine di ottenere il comportamento desiderato. Composizione richiede definizione di precise interfacce ma non richiede conoscenza di nessun dettaglio implementativo.

Principi generali di buona progettazione:

- Programmare riferendosi alle interfacce e non alle implementazioni
- Favorire la composizione di oggetti rispetto all'ereditarietà di classe