
# Relazioni

Andrea Polini

Ingegneria del Software
Corso di Laurea in Informatica

# Sommario

# Relazione

## Relazione - da teoria degli insiemi

*. . . a relation is a property that assigns truth values to k-tuples of individuals. Typically, the property describes a possible connection between the components of a k-tuple. . . .*

Si considerano in questo contesto relazioni tra oggetti e tra classi.

# Collegamento

**Collegamento**

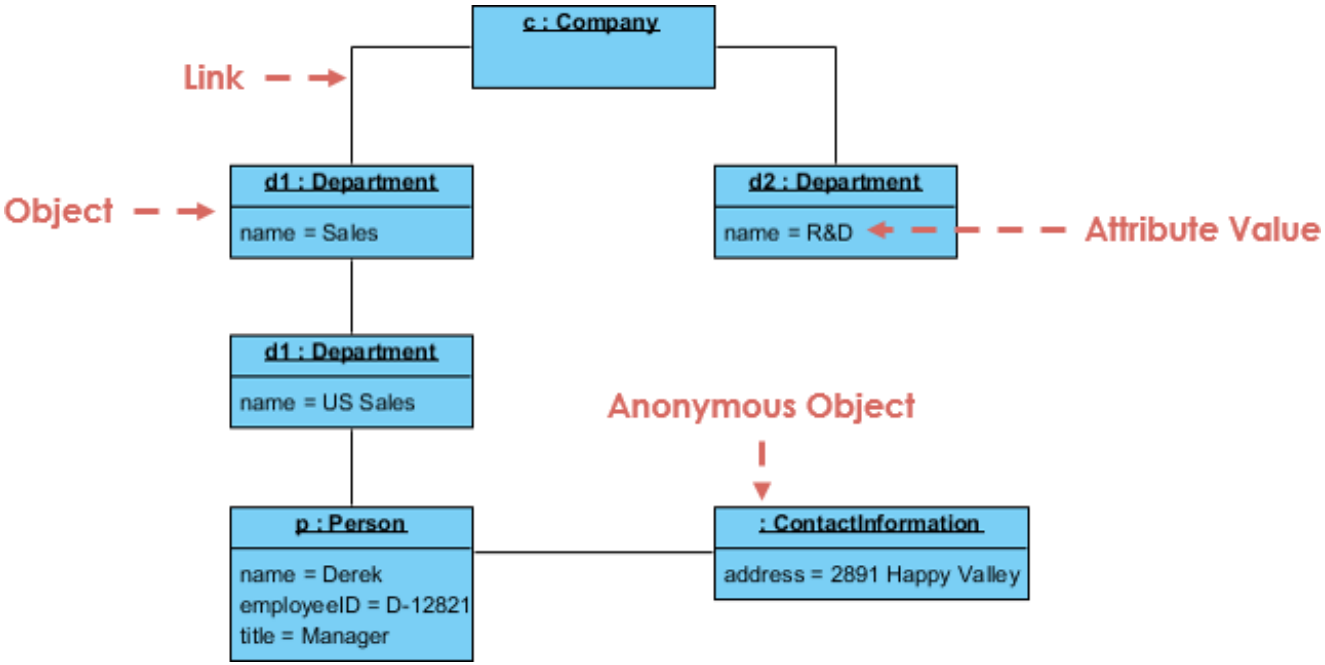
connessione semantica tra due oggetti che consente loro di scambiarsi messaggi. Implementati in modi differenti nei linguaggi di programmazione è comunque necessario che chi vuole inviare un messaggio possa recuperare un riferimento a chi lo deve ricevere.

- I collegamenti sono connessioni dinamiche
- specifica della navigabilità
- relazione n-arie

# Diagramma degli oggetti

## Obiettivo

Mostra un'istantanea del sistema in un dato momento che mostra gli oggetti ed i loro collegamenti



# Associazione

## Associazione

Così come i collegamenti connettono gli oggetti le associazioni connettono le classi. Se esiste un collegamento tra oggetti deve esistere un associazione di qualche tipo tra le classi corrispondenti

Le associazioni vengono indicate tramite le seguenti informazioni:

- nome dell'associazione
  - *esplicativo . . . provate a "leggere"*
  - *può essere sostituito dai ruoli nell'associazione*
- nomi dei ruoli
- molteplicità
- navigabilità

Si definiscano semplice associazioni tra i concetti presenti nel sistema "distributore automatico"

# Molteplicità

Pone un vincolo sulla cardinalità della relazione

Viene definita attraverso specifiche stringhe, che indicano minimo e massimo, poste al termine dei due lati del simbolo di associazione:

- 0..1 (min=0, max=1)
- 1 (min=1, max=1)
- 0..* o * (min=0, max=*unbounded*)
- 1..* (min=1, max=*unbounded*)
- 1..6 (min=1, max=6)
- 1..4,6..9,12..15,17..* (unione di intervalli)

La molteplicità può e dovrebbe essere indicata nei modelli di analisi. Nel caso non venga indicata si assume indefinita

# Navigabilità

La navigabilità indica la possibilità di inviare un messaggio tra gli oggetti associati, in corrispondenza della direzione specificata.
Si usano i simboli:

- $\rightarrow$ indica la possibilità di navigare il modello in quella direzione
- x indica che il modello non può essere navigato in quella direzione

In generale si cerca di limitare l'indicazione della navigabilità allo stretto necessario.
Tre idiomi di modellazione

1. navigabilità completamente esplicita
2. completamente invisibile
3. eliminare tutte le croci

In 3 si muta interpretazione di indefinizione rispetto alla interpretazione precisa di UML2

# Associazioni ed attributi

Un'associazione può essere interpretata con l'esistenza di uno pseudo attributo il cui tipo è la classe destinazione.

Cardinalità superiori ad uno richiedono l'uso di vettori o di classi collezione

Gli pseudo-attributi vengono utilizzati quando la classe destinazione è una parte importante del modello altrimenti è più chiaro definire attributo nella classe sorgente

# Classi associazione

## Classi Associazione

le classi associazioni servono ad arricchire la semantica di una associazione tra classi e permettono di caratterizzare la relazione tra le classi con degli attributi specifici

- e.g. relazione azienda/lavoratore
- relazioni reificate (sala/incassi/titolo)

# Associazioni qualificate

## Associazioni qualificate

servono a ridurre relazioni molti-a-molti in relazioni molti-a-uno, specificando un solo oggetto (o gruppo) scelto dall'insieme degli oggetti destinazione.
Permettono di indicare come sia possibile recuperare informazioni sugli oggetti coinvolti nella relazione attraverso l'indicazione della chiave di ricerca da utilizzare

# Dipendenze

## Dipendenze

servono ad indicare una relazione tra due o più elementi del modello, dove un cambiamento ad uno di essi può influenzare o fornire informazioni necessarie all'altro. Si usano per descrivere relazioni tra classificatori (i.e. passaggio di un parametro)

Ci sono tre tipi di dipendenza definiti in UML2:

- Uso
- Astrazione
- Permesso

Dipendenze esistono tra:

- classe e classe
- package e package
- oggetti e classi
- operazioni e classi

# Dipendenza di Uso

**Uso**

il cliente usa alcuni dei servizi resi disponibili dal fornitore per implementare il proprio comportamento

- usa
  - un metodo della classe A ha bisogno di un parametro della classe B
  - un metodo di A restituisce un valore della classe B
  - un metodo usa un oggetto B anche se non come parametro
- chiama (tra operazioni)
- parametro
- invia (i.e. messaggio - classe)
- istanzia (i.e. oggetto - classe)

# Dipendenza di Astrazione

**Astrazione**

relazione tra cliente e fornitore in cui il fornitore è più astratto del cliente

- traccia
- sostituisce
- raffina
- deriva-da

# Dipendenza di Permesso

**Permesso**

il fornitore assegna diritti di accesso al proprio contenuto: in questo modo il fornitore limita e controlla gli accessi al proprio contenuto

- accede (tra package)
- importa (tra package)
- permette

# Sommario

# 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*

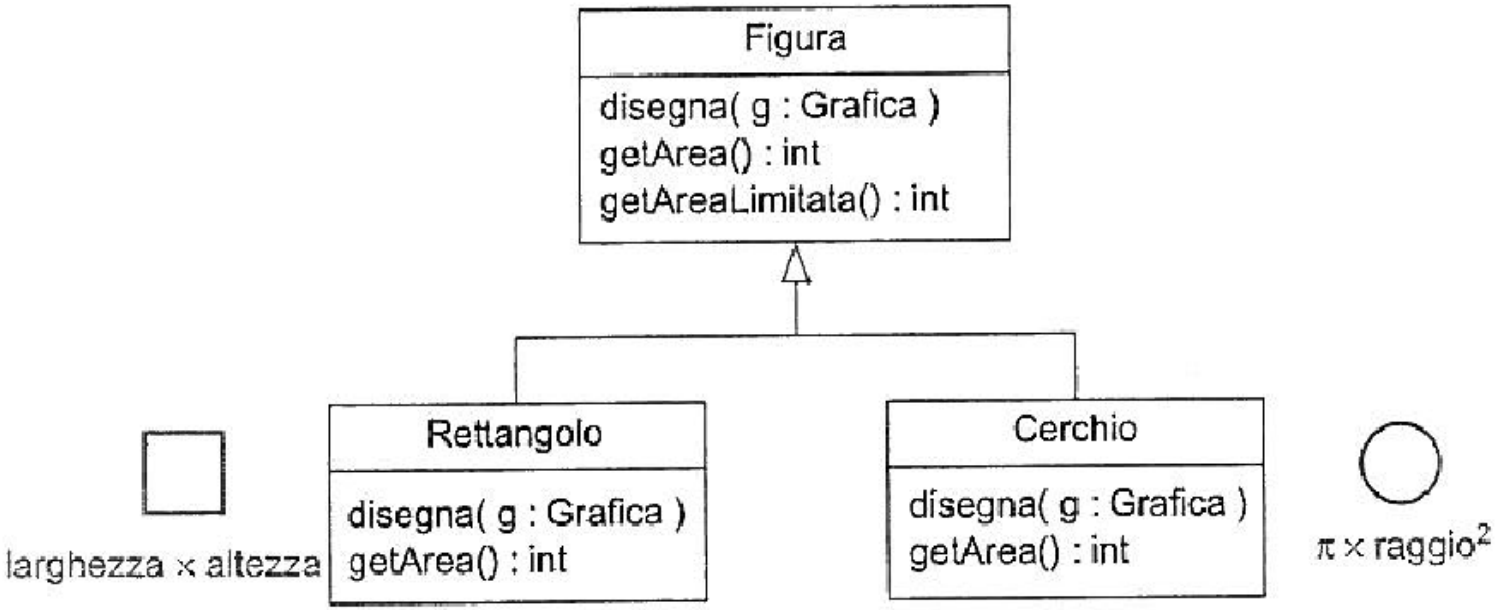
# Ereditarietà

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



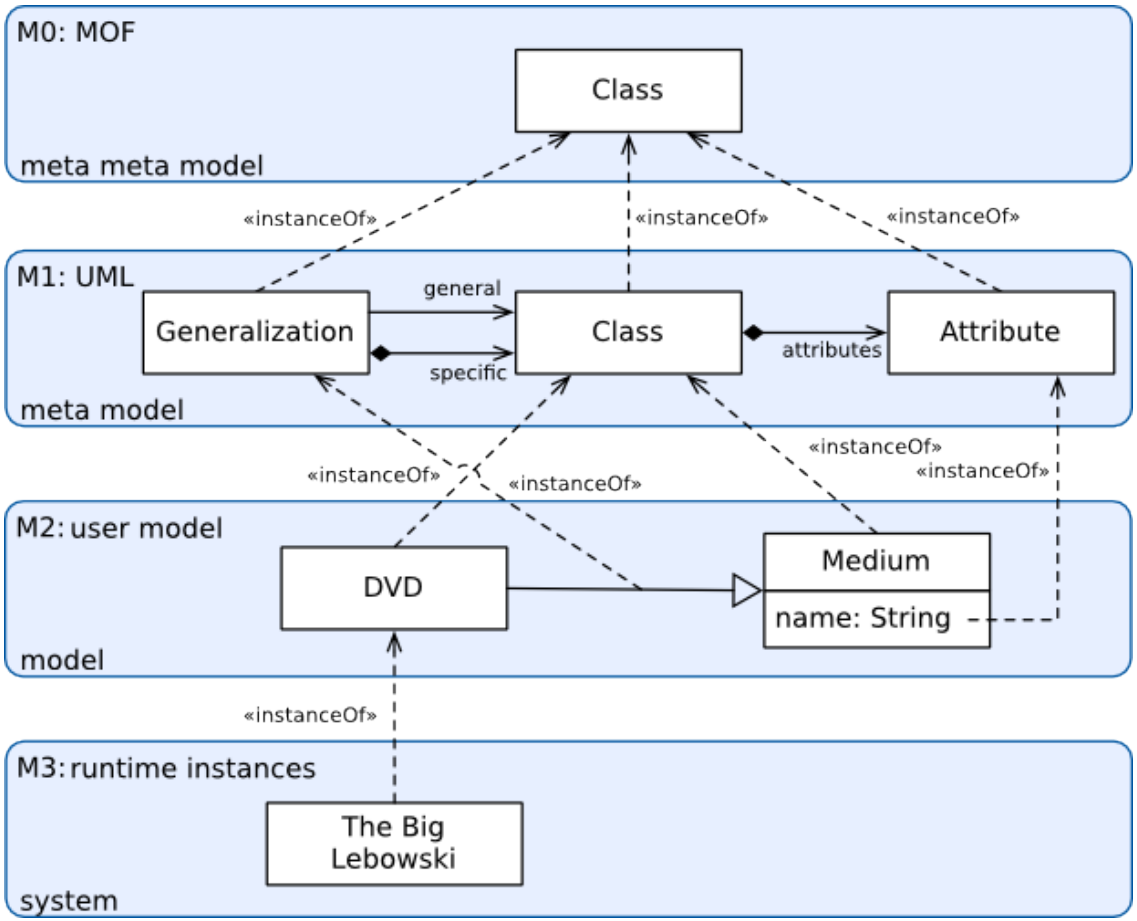
# Livelli di astrazione

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