Principi di progetto OO – III

Stefano Mizzaro

Dipartimento di matematica e informatica
Università di Udine
http://www.dimi.uniud.it/~mizzaro
mizzaro@dimi.uniud.it

PAOO, Lezione 7
26/2/2004

Riassunto

- I principi della progettazione OO
  - Livelli di incapsulamento
  - Dipendenze
  - Domini
  - Ingombro
  - Legge di Demeter
  - Coesione
  - Spazio degli stati (ed eredità)
  - Transizioni e comportamento (ed eredità)

Scaletta

Gio. 4/3 non c'è lezione!!!!

Ci vediamo l'11/3 (ultima)

- Asserzioni
- Invarianti di classe
- Precondizioni e postcondizioni delle operazioni (metodi)
- Progetto per contratto (Bertrand Meyer)
- inv, pre, post ed eredità
  - Conformità di tipo
  - Comportamento chiuso

Asserzioni

- Def.: Asserzione = Espressione logica che deve essere sempre vera in un certo punto del codice (durante qualsiasi esecuzione)
- Es.

```
for (int i = 0; i <= 10; i++) {
  …
}
```

$0 \le i \le 10$

Invariante di classe

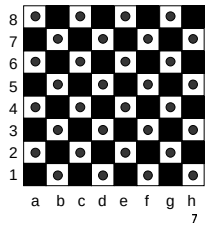
- Asserzione sugli oggetti di una classe
- Def.: Invariante di classe: condizione che ogni oggetto della classe deve soddisfare (quando è "in equilibrio")
- In equilibrio: non "in mezzo" a una transizione
  - Dopo il costruttore
  - Quando non è in esecuzione un metodo pubblico (prima e dopo l'esecuzione di un metodo pubblico)

Esempi di invarianti

- Classe **`Triangolo`**
  - "Disuguaglianza triangolare": Somma lunghezza di due lati ≥ terzo lato
  - **`a+b≥c & a+c≥b & b+c≥a`**
- Classe **`TriangoloIsoscele`**
  - Due lati di lunghezza uguale
  - **`a=b | a=c | b=c`**
- Classe **`Punto3dSuperficieSfera`**
  - $x^2+y^2+z^2=r^2$
- Classe **`Punto2dInternoSfera`**
  - $x^2+y^2<r^2$

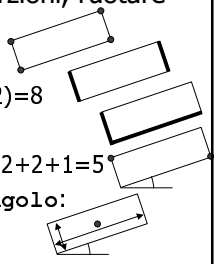
Invarianti e spazio degli stati

- Invariante come vincolo sullo SdS
- Qual è lo SdS di **`Punto2dInternoSfera`**?
  - Spazio 2d, ma non tutto lo spazio 2d
- Alfiere
  - SdS: a≤col≤h & 1≤trav≤8
  - inv: ind(col)+trav=2k+1
- Invarianti e dimensione SdS:
  - I vincoli dell'inv possono diminuire la dimensionalità, i gradi di libertà, dello SdS



Esempio

- SdS di un rettangolo, 2d, può ingrandirsi, rimpicciolirsi, cambiare proporzioni, ruotare e traslare
  - **`p1`**, **`p2`**, **`p3`**, **`p4`** (punti): 4*2=8
  - **`s1`**, **`s2`** (segmenti): (2*2)+(2*2)=8
  - **`s1`**, **`s2`** (segmenti): (2*2)+2=6
  - **`altoSx`**, **`bassoDx`** e **`angolo`**: 2+2+1=5
  - **`centro`**, **`altezza`**, **`largh`**, **`angolo`**: 2+1+1+1=5
- Che inv nei vari casi?



Dimensionalità di tipo e di classe

- Il tipo **`Rettangolo`** ha dimensionalità 5
- La classe che implementa il tipo **`Rettangolo`** ha una dimensionalità di classe che dipende dalle scelte implementative
- Dimensionalità di classe – dimensionalità di tipo = numero di vincoli che l'inv deve specificare
- Numero di vincoli che l'inv specifica + dimensionalità di tipo = dimensionalità di classe
- Più vincoli ho più è difficile scrivere l'inv…

Precondizioni

- Precondizione di un metodo
  - Condizione che deve essere vera subito prima dell'esecuzione del metodo
- Es.
  - Metodo **`int sqrt(int x)`**
  - pre: **`x`** ≥ 0
- Se la pre non è verificata, il risultato è non predicibile (errato, eccezione, …)

Postcondizioni

- Postcondizione di un metodo:
  - Condizione che deve essere vera subito dopo l'esecuzione del metodo
- Es.
  - Metodo **`int sqrt(int x)`**
  - post: **`res*res≤x & (res+1)*(res+1)≥x`**
- Se la post non è verificata, il metodo è sbagliato
- pre e post sono asserzioni su inizio/fine di un metodo

pre, post e inv

- Se la post non è verificata, e se pre era vera, il metodo è sbagliato
- N.B. pre e post vanno accoppiate con inv!
- Prima dell'esecuzione di un metodo **`m()`**, devono essere vere sia pre_m sia inv
- Dopo l'esecuzione di un metodo **`m()`**, devono essere vere sia $post_m$ sia inv (assumendo che pre_m e inv fossero vere all'invocazione)

Esempio

- Classe **`Pila`**
  - **`push() top() pop() piena() vuota()`**
  - **`push(x)`**
    - pre: **`!piena()`**
    - post: **`!vuota() & numElem == old(numElem)+1 & a[numElem-1]==x`** (o meglio: **`top()==x`**)
  - **`pop`**
    - pre: **`!vuota()`**
    - post: **`!piena() & numElem == old(numElem)-1`**
  - **`top`**
    - pre: **`!vuota()`**
    - post: **`res==a[numElem-1]`**

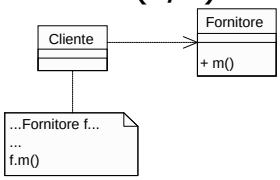
pre, post e inv: pubbliche o private?

- Sono informazioni pubbliche, nell'interfaccia della classe
  - inv
  - pre e post dei metodi pubblici
- Problema:
  - difficile specificare inv e post senza fare riferimento all'implementazione…
- 2 versioni, una privata e una pubblica?

Scaletta

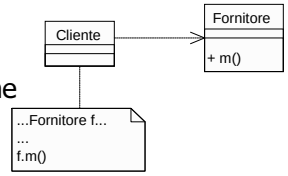
- Asserzioni
- Invarianti di classe
- Precondizioni e postcondizioni delle operazioni (metodi)
- Progetto per contratto (Bertrand Meyer)
- inv, pre, post ed eredità
  - Conformità di tipo
  - Comportamento chiuso

Progetto per contratto (1/2)



- Invocazione di un metodo **`m`** (con pre e post) della classe **`Fornitore`** dalla classe **`Cliente`** (con inv)
- Contratto fra cliente e fornitore
- Il fornitore fornisce il servizio dato dal metodo **`m`** (ad es., **`sqrt`**)

Progetto per contratto (2/2)



- Il fornitore garantisce che se pre & inv, allora, dopo l'esecuzione di **`m`**, vale post (& inv)
- Il fornitore chiede che valgano pre e inv (altrimenti non garantisce nulla!)
- Pre = Richiesta che il metodo/fornitore fa (al chiamante/cliente)
- Post = Assicurazione, garanzia che il metodo/fornitore dà

Chi controlla pre?

- Chi ha la responsabilità di assicurare la validità di pre?
- Contratto ⇒ Il cliente (chiamante)!
- Il fornitore (metodo chiamato) può non controllarla
- Controllo duplice ⇒ spreco
  - Il fornitore (metodo chiamato) deve non controllarla
  - Contrario alla "programmazione difensiva"…
- Pre non valida ⇒ errore nel chiamante
- Post o inv non valida ⇒ errore nel fornitore

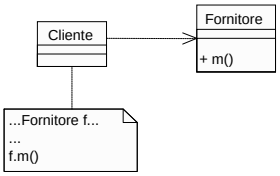
Esercizio

- L'invariante è necessaria? Come si potrebbe eliminarla mantenendo le stesse proprietà di una classe?

Risposta

- Non è necessaria
- Potrei aggiungerla in and logico a pre e post di ogni metodo della classe
- Dovrei fare attenzione alle sottoclassi
- Scomodo, chi me lo fa fare??

Esercizio



- Una pre forte, restrittiva, è una buona notizia per il fornitore (chi scrive il metodo `m`) o per il cliente (chi chiama `m`)?
- Una post forte, restrittiva, è una buona notizia per il fornitore o per il cliente?

Risposta

- Una pre forte, restrittiva, è
  - una cattiva notizia per il cliente, che deve soddisfare una richiesta più "difficile"
  - una buona notizia per il fornitore, che ha la responsabilità di rispettare il contratto in meno casi
- Una post forte, restrittiva, è
  - una cattiva notizia per il fornitore, che deve garantire qualcosa di più
  - una buona notizia per il cliente, a cui viene fornito un servizio più "impegnativo"
- (viceversa per pre e post deboli)

Esercizio

- Una inv forte, restrittiva, è una notizia buona o cattiva per il fornitore? E per il cliente?
- Risposta
  - Fornitore: entrambi
    - pre & inv: una buona notizia, ha la responsabilità di rispettare il contratto in meno casi
    - post & inv: una cattiva notizia, deve garantire qualcosa di più
  - Cliente: entrambi
    - pre & inv: una cattiva notizia, deve soddisfare una richiesta più "difficile"
    - post & inv: una buona notizia, gli viene fornito un servizio più "impegnativo"

Scaletta

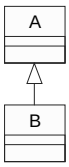
- Asserzioni
- Invarianti di classe
- Precondizioni e postcondizioni delle operazioni (metodi)
- Progetto per contratto (Bertrand Meyer)
- inv, pre, post ed eredità
  - Conformità di tipo
  - Comportamento chiuso

Conformità di tipo

- inv, pre e post: cosa succede con l'eredità?
- Ogni gerarchia deve rispettare il principio della conformità di tipo:
  - Un'istanza della sottoclasse può essere fornita ovunque ci si aspetti un'istanza della sopraclasse…
  - … e il programma funziona ancora correttamente (quando qualunque operazione di accesso viene eseguita…)

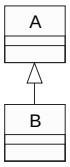
Invarianti ed eredità (1/2)

- Devo poter mettere un'istanza di **`B`** ovunque ci va un'istanza di **`A`**
  - **`Triangolo`** e **`TriangoloIsoscele`**
- **`inv(B)`** deve essere restrittiva almeno quanto **`inv(A)`**
  - **`inv(B) = inv(A) & ...`**
  - **`inv(B)`** $\Rightarrow$ **`inv(A)`**
- È diverso da SdS e comportamento!
  - Non c'è "espandi e restringi", c'è solo "restringi"



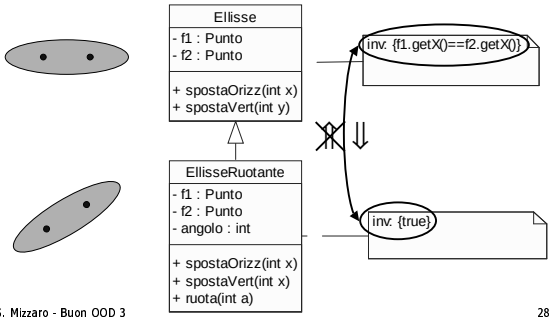
Invarianti ed eredità (2/2)

- Ci sono 2 restrizioni
  - Sulle dim. di SdS_A "ristrette"e
  - Sulle dim. "nuove" di SdS_B
  - **`inv(B) = inv(A) & inv(`**A_{restr}**`) & inv(`**A_{est}**`)`**
- Un **`TriangoloIsoscele`**
  - deve sempre soddisfare la disuguaglianza triangolare…
  - … e in più deve avere due lati uguali
- Un **`Automobile`**…



Esercizio

- Vi piace questa gerarchia?

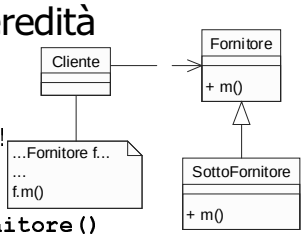


Risposta

- Sicuramente no:
  - L'invariante della sottoclasse è più debole di quella della sopraclasse!
  - Lo SdS della sottoclasse non viene ristretto sulle dimensionalità dello SdS della sopraclasse; viene espanso!!
- I nomi traggono in inganno!
  - Un'**`EllisseRuotante`** è un'**`Ellisse`**…
  - … ma un'**`EllisseRuotante`** non è un'**`EllisseOrizzontale!`**

Pre/post ed eredità

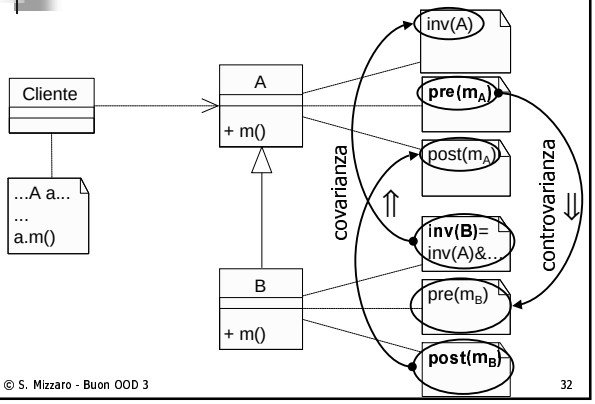
- **`f.m()`** potrebbe essere un'invocazione al metodo della sottoclasse!
  - (**`f`** istanza di **`SottoFornitore`**)
- E **`f = new Sottofornitore()`** potrebbe essere eseguito da chissà chi…
- Rispettiamo il contratto:
  - Il cliente garantisce pre_m e non vuole garantire nulla di più
  - Il (sotto)fornitore deve garantire $post_m$ (e non può garantire nulla di meno!)
  - Il sottofornitore deve "Chiedere di meno, garantire di più"



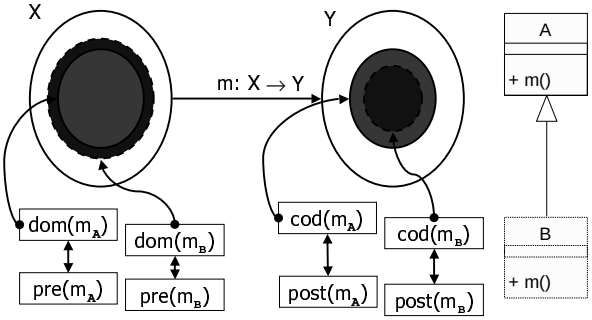
"Covarianza" e "controvarianza"

- Terminologia di Page-Jones (errata?)
- La postcondizione co-varia con l'invariante
  - L'invariante della sottoclasse deve essere più restrittiva, e anche la postcondizione
- La precondizione contro-varia con l'invariante
  - L'invariante della sottoclasse deve essere più restrittiva, mentre la precondizione deve essere meno restrittiva
- Quando sovrascrivo un metodo devo rimpiazzare la pre originale con una più debole e la post originale con una più forte (in senso lato)

Chiedere di meno e garantire di più

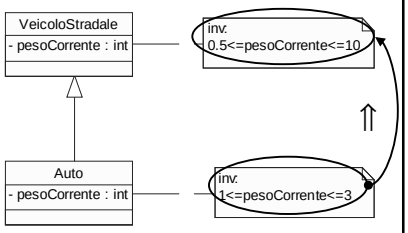


Metodo come funzione matematica



Esercizio

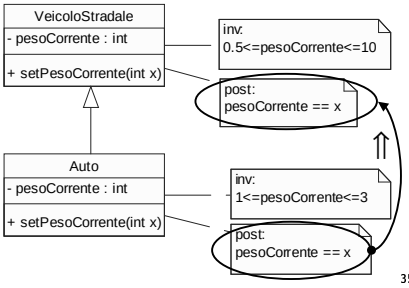
- Tutto ok?
- inv: Pare di sì…



- … ma proviamo a metterci un metodo e a scrivere pre e post…

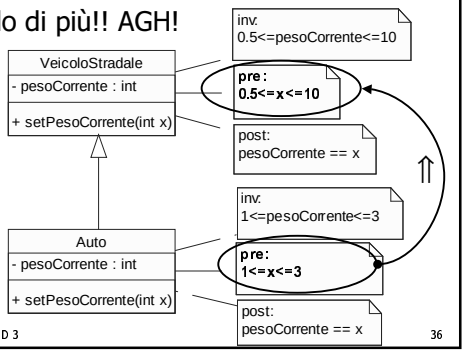
Esercizio (cont.)

- Post: pare di sì…



Esercizio (cont.)

- Pre: uhm… Non sto chiedendo di meno! Sto chiedendo di più!! AGH!

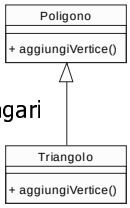


Riepilogo

- Per garantire la conformità di tipo:
  - inv sottoclasse più forte inv sopraclasse (SdS sottoclasse ha più vincoli sulle dim. della sopraclasse e vincoli nuovi sulle nuove dim.)
  - pre nel metodo della sottoclasse più debole (chiedere di meno, controvarianza)
  - post nel metodo della sottoclasse più forte (garantire di più, covarianza)
- Non è finita…☺

Comportamento chiuso

- Una classe è chiusa rispetto al comportamento se tutte le transizioni lasciano gli oggetti nello SdS
- Es.
  - Classi **`Poligono`** e **`Triangolo`**
  - Metodo **`sposta`**: OK
  - Ma il metodo **`aggiungiVertice`** (magari ereditato da **`Poligono`**)…



Principio del comportamento chiuso

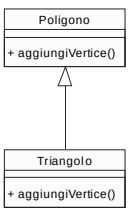
- Il comportamento che una sottoclasse eredita da una sopraclasse deve rispettare l'invariante della sottoclasse (!)
- L'esecuzione di qualsiasi metodo di una classe **`C`**, compresi i metodi ereditati, deve obbedire all'inv. di **`C`**
- Ciò non è automatico, va progettato esplicitamente
- Chi progetta una (sotto)classe deve garantire la chiusura del suo comportamento

Si può modificare solo la sottoclasse!

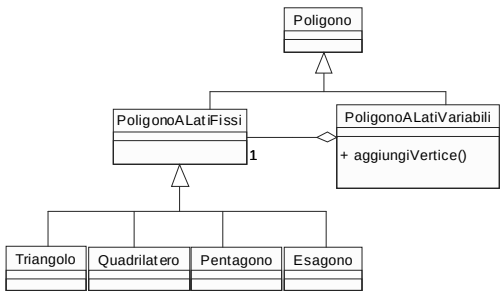
- Le sopraclassi non sanno nulla delle loro sottoclassi! La dipendenza è dalla sottoclasse alla sopraclasse, non viceversa
- Quando progetto una sottoclasse devo sovrascrivere/ridefinire i metodi ereditati che non rispettano l'invariante della sottoclasse
- Devo assumere che non posso modificare la sopraclasse
  - Spesso è proprio così: classi delle API…

Esempio (1/2)

- Metodo **`aggiungiVertice`** di **`Triangolo`** ereditato da **`Poligono`** (e **`Quadrilatero`**, **`Pentagono`**, …)
- Soluzioni:
  - Evitare di ereditare il metodo (alcuni LdP lo consentono)
  - Sovrascrivere **`aggiungiVertice`** (ma qual è la postcondizione?)
  - Riclassificare l'istanza di **`Triangolo`** come istanza di **`Quadrilatero`**
- Soluzione più radicale: riprogettare (se possibile!)



Esempio (2/2)



Riepilogo

- Per avere gerarchie "a prova di bomba":
  - inv sottoclasse più forte inv sopraclasse (SdS sottoclasse ha più vincoli sulle dim. della sopraclasse e vincoli nuovi sulle nuove dim.)
  - pre nel metodo della sottoclasse più debole (chiedere di meno, controvarianza)
  - post nel metodo della sottoclasse più forte (garantire di più, covarianza)
  - Metodi ereditati dalla sopraclasse devono rispettare l'inv della sottoclasse (comportamento chiuso)

Riassunto

Gio. 4/3 non c'è lezione!!!!

Ci vediamo l'11/3 (ultima)

- Asserzioni
- Invarianti di classe
- Precondizioni e postcondizioni delle operazioni (metodi)
- Progetto per contratto (Bertrand Meyer)
- inv, pre, post ed eredità
  - Conformità di tipo
  - Comportamento chiuso

Bibliografia

- Meilir Page-Jones, *Progettazione a oggetti con UML*, Apogeo, 2002, capp. 10.4 – 11
- Bertrand Meyer, *Object Oriented Software Construction*, 2a ed., Prentice Hall, 1997, capp. 11, 14, 16

Un problema...