Principi di progetto OO – II

Stefano Mizzaro

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

PAOO, Lezione 6
19/2/2004

Riassunto

- Intro, UML, …
- I principi della progettazione OO
  - Livelli di incapsulamento (0, 1, 2, 3, …)
  - Dipendenze e incapsulamento come freno
  - Domini
    - Applicativo, aziendale, architetturale, fondazionale
  - Ingombro

Scaletta

- Legge di Demeter
- Coesione
- Spazio degli stati (ed eredità)
- Transizioni e comportamento (ed eredità)

Legge di Demeter

- Per qualsiasi metodo `m` di un oggetto `x` della classe `A`, il destinatario dei messaggi nel corpo di `m` deve essere
  1. `x` stesso (`this`)
  2. Un oggetto presente come argomento di `m`
  3. Un oggetto riferito da un attributo di `x`
  4. Un oggetto creato da `m`
  5. (?) Un oggetto riferito da una variabile "globale"

Due versioni della legge di Demeter (forte/debole)

- Per qualsiasi metodo `m` di un oggetto `x` della classe `A`, il destinatario dei messaggi nel corpo di `m` deve essere
  1. `x` stesso (`this`)
  2. Un oggetto presente come argomento di `m`
  3. Un oggetto riferito da un attributo di `x` (debole: eventualmente ereditato)
  4. Un oggetto creato da `m`
  5. (?) Un oggetto riferito da una variabile "globale"

Un'altra versione…

- Non usare più di un punto (o almeno pensarci bene prima di farlo)

```
Auto a;
a.accendi();
a.apriPortiera();
a.getRimorchio().apriPorta();
a.getRimorchio().fornello.spegni();
a.getRimorchio().getFornello().spegni();
```

```
a.apriPortaRimorchio();
a.spegniFornelloRimorchio();
```

Commenti

- La legge di Demeter limita:
  - i riferimenti diretti, e quindi l'ingombro diretto di una classe
  - le dipendenze che attraversano l'incapsulamento
- Ragionevole
- "Una classe `A` deve usare solo quello che le serve, non di più"
- Non "legge" ma "linea guida" o "euristica"
- Riuso
- Refactoring

Scaletta

- Legge di Demeter
- Coesione
- Spazio degli stati (ed eredità)
- Transizioni e comportamento (ed eredità)

Coesione di classe (tipo)

- Misura di
  - Quanto coesa è l'interfaccia pubblica di una classe
  - Quanta corrispondenza reciproca c'è fra i pezzi (operazioni, attributi) dell'interfaccia pubblica di una classe
  - Quanto "sta bene insieme" ciò che c'è nell'interfaccia pubblica di una classe
- È una misura "esterna", sull'interfaccia

4 livelli di coesione

- Coesione ideale (la migliore)
- Coesione a ruolo misto
- Coesione a dominio misto
- Coesione a istanza mista (la peggiore)

Alta/ Buona

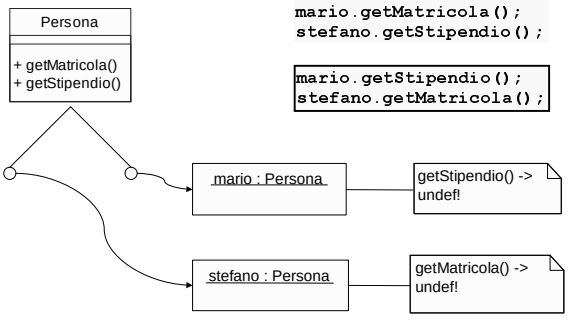
Bassa/ Cattiva

- Vediamo le ultime 3, dal basso

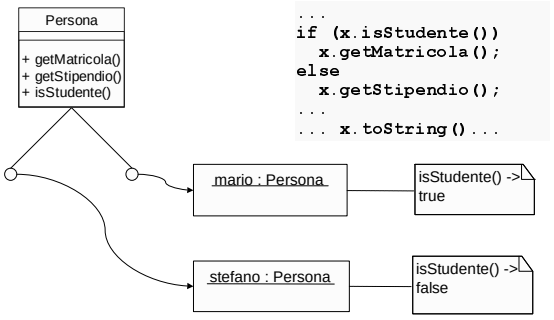
Classe con coesione a istanza mista

- Def.: Classe con attributi o operazioni non definiti per alcune istanze
- Classe troppo grossolana, "mette insieme" istanze che andrebbero separate
- Le istanze sono divise in due sottoinsiemi:
  - Quelle con attributo/operazione non definito
  - Quelle con attributo/operazione definito

Istanza mista: esempio (1/2)



Istanza mista: esempio (2/2)



(Un progetto un po' migliore)

- 2 sottoclassi, `Studente` e `Lavoratore`
  - Le istanze di `Studente` non hanno `getStipendio()`
  - Le istanze di `Lavoratore` non hanno `getMatricola()`
- Eredità e polimorfismo
- (soluzione non ideale, vedremo…)
- La classe `Persona` originale aveva una fat interface

Classe con coesione a dominio misto

- Def.: Classe ingombrata con una classe estrinseca appartenente a un dominio diverso
- Def. (vaga): `B` è estrinseca rispetto ad `A` sse "è possibile" definire `A` senza alcun riferimento a `B`
- Estrinseca = non pertinente, "non c'azzecca"
- Es.:
  - `Data` è intrinseca rispetto a `Persona`
  - `Elefante` è estrinseca rispetto a `Persona`
- È sempre una "fat interface"…

Dominio misto: esempio

- Classe `Double` con metodo `arctan()` (o attributo…)
- Niente di strano?? E allora:
  - `equivalenteInMiglia()`
  - `ammontareInEuro()`
  - `equivalenteInGradiCentigradi()`
  - …non si finisce più…
- Ex.:
  - Ancora peggio: `arccos()` – Perché?
  - In che classe mettere `arctan` e `arccos`?

Dominio misto: considerazioni

- Motto OO:
  - "Un oggetto deve sapere come fare qualcosa a se stesso". Mah…
- `stampa()` in `Documento`? (e che gliene frega a un documento di una stampante??)
- E infatti, `toString()`, non `print()`…

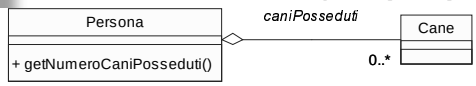
Legame con i domini

- Se progetto una classe:
  - posso aver bisogno, con criterio, di classi dei domini inferiori
  - campanello d'allarme se uso classi dei domini superiori
- `Double` è nel dominio fondazionale, sottodominio fondamentale (il più in basso)
- `Angolo` è in fondazionale-semantico
- Domanda chiave: riesco a immaginarmi l'esistenza di una classe senza l'altra, e viceversa?

Classe con coesione a ruolo misto

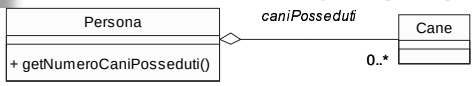
- Come la precedente, ma la classe che ingombra è nello stesso dominio della classe ingombrata
- Def.: è ingombrata con una classe estrinseca appartenente allo stesso dominio
- Non attraverso i domini, ma ho sempre una "fat interface"

Ruolo misto: esempio (1/2)



- Classe `Persona` con attributo `caniPosseduti` e metodo `getNumeroCaniPosseduti()`
- `mario.getNumeroCaniPosseduti()`
- Persona ha coesione a istanza mista?
  - No, se Mario non ha cani restituisco 0
- Persona ha coesione a dominio misto?
  - No, `Persona` e `Cane` nello stesso dominio (aziendale)

Ruolo misto: esempio (2/2)



- E dove sta il problema?
- `Persona` e `Cane` sono estrinsechi, "non c'azzeccano"
- … sempre fat interface…
  - `autoPossedute`, `barchePossedute`, `gattiPosseduti`, `iguanePossedute`, …
- Riuso!!

Ruolo misto: considerazioni

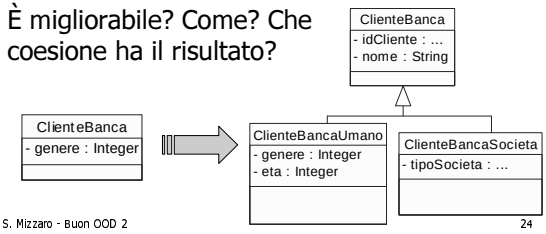
- È il problema meno grave
- È semplice: `mario.getNumeroCaniPosseduti()`
- È naturale: "se voglio sapere quanti cani ha Mario, glielo chiedo…" (?)
- Ma non create classi con coesione a ruolo misto "automaticamente", "a cuor leggero"
- (ne riparleremo…)

Coesione: considerazioni

- Una classe deve essere un'astrazione uniforme sugli oggetti reali
- Astrazione: non deve rappresentare tutte le caratteristiche degli oggetti reali
  - `Cliente` di banca: non mi interessa l'altezza
- Uniforme: tutte le istanze devono essere uniformi
  - Se interessa l'età di `Cliente`, interessa per tutti
- Contraddizione: "fat interface" – legge di Demeter…
- No regole fisse, difficile, arte, buon senso…

Esercizio

- Classe `ClienteBanca`
  - `genere` = `0` (ignoto), `1` (maschio), `2` (femmina) o `3` (altro, in realtà società)
- Che coesione ha?
- È migliorabile? Come? Che coesione ha il risultato?



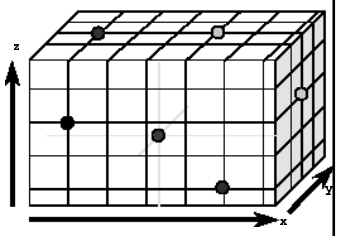
Scaletta

- Legge di Demeter
- Coesione
- Spazio degli stati (ed eredità)
- Transizioni e comportamento (ed eredità)

Spazio degli stati

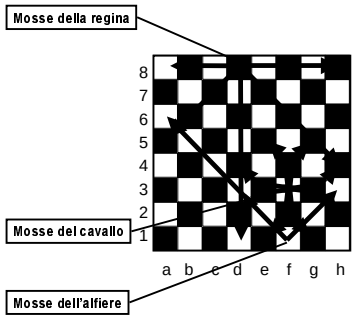
- Def.: Spazio degli stati (SdS) di una classe C: insieme degli stati in cui un oggetto di C può trovarsi

```
class Punto3d {
  private double x;
  private double y;
  private double z;
  ...
}
```



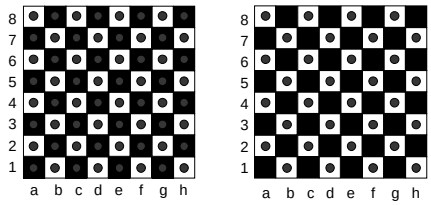
Esempio: scacchi (1/2)

- Scacchi:
  - Regina
  - Cavallo
  - Alfiere



Esempio: scacchi (2/2)

- SdS di cavallo e regina
- SdS dell'alfiere



Dimensioni dello SdS

- Def.: Dimensioni dello SdS = coordinate necessarie per specificare lo stato di un oggetto, gradi di libertà
  - `Punto2d`: 2 coordinate `x`, `y`
  - `Punto3d`: 3 coordinate `x`, `y`, `z`
  - `Persona`: `nome`, `cognome`, `dataDiNascita` (di tipo `Data`!... gg/mm/aa → 3)

Spazio degli stati e attributi

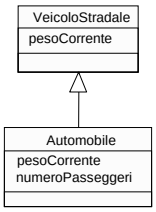
- Dimensioni nello SdS = valori degli attributi di un oggetto?
- Sì, ma solo se attributi:
  - indipendenti!! Quelli derivati non contano…
  - semplici!! Quelli composti vanno scomposti…
- Es.
  - `Punto3d`: 3 coordinate `x`, `y`, `z`
  - `Cubo`: `larghezza`, `altezza`, `profondita`, *`volume`*, *`area`*
  - `Persona`: `dataDiNascita` (di tipo `Data`!) ed *`eta`*

Esercizio

- SdS di un rettangolo
  - Nel piano 2d
  - Può ingrandirsi, rimpicciolirsi e cambiare proporzioni
  - Può ruotare e traslare
- Quante dimensioni ha il suo SdS?

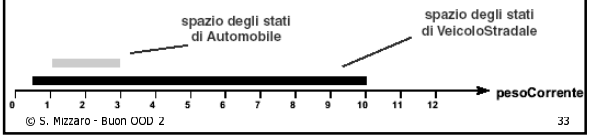
Spazio degli stati ed eredità

- `VeicoloStradale`
- `Automobile`
- Che relazione c'è fra i due SdS?
  - Restrizione: Dovunque ci deve essere un `VeicoloStradale`, ci posso mettere un `Automobile`
  - Estensione: `Automobile` estende `VeicoloStradale` (ad es., con `numeroPasseggeri`)



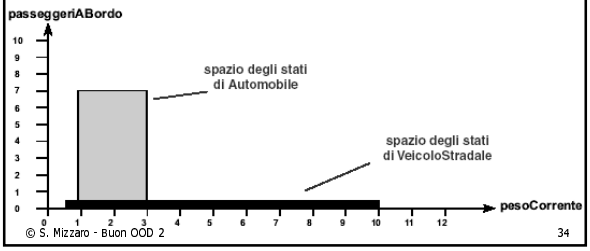
Restrizione

- Un veicolo stradale pesa da 0.5 a 10 tonn.
- Un'auto pesa da 1 a 3 tonn.
- Se un'auto potesse pesare meno di 0.5t o più di 10t, sarebbero guai
  - Pensate ad un array di `VeicoloStradale`…



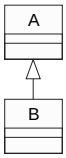
Estensione

- Ma la sottoclasse può avere attributi aggiuntivi!



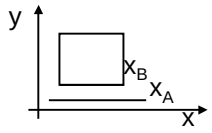
Estensione **e** restrizione

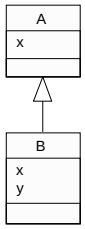
- Estensione:
  - Lo SdS_B può aggiungere dimensioni a SdS_A, ma…
- Restrizione:
  - …la proiezione dello SdS_B sulle coordinate dello SdS_A deve essere contenuta nello SdS_A



Spazio degli stati della sottoclasse

- Quindi:
  - SdS_B è delimitato da SdS_A
  - SdS_B estende SdS_A
- Uhm…





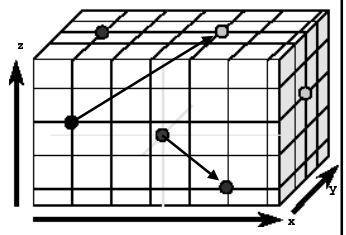
Scaletta

- Legge di Demeter
- Coesione
- Spazio degli stati (ed eredità)
- Transizioni e comportamento (ed eredità)

Transizione

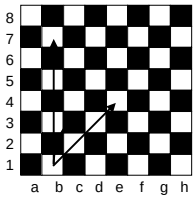
- Def.: Transizione: Passaggio da un punto nello SdS ad un altro

```
class Punto3d {
  private double x;
  private double y;
  private double z;
  ...
  void sposta(…) {
    ...
  }
}
```



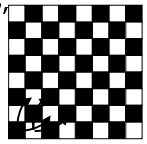
Esempi

- Transizione per cavallo: b1 → c3
- Transizione per regina: b1 → b7
- Transizione per alfiere: b1 → e4

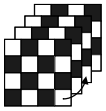


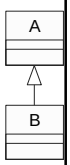
Comportamento

- Def.: Comportamento di una classe: Insieme delle transizioni lecite
- Comportamento cavallo:
  - {a1 → b3, a1 → c2, b1 → a3, b1 → c3, …}
- Comportamento regina (senza altri pezzi):
  - {a1 → a2, a1 → a3, …, a1 → a8, a1 → b2, a1 → c3, … a1 → h8, a1 → a2, a1 → a3, … a1 → a8, b1 → …}



Comportamento ed eredità (1/2)

- Quali sono le transizioni in `B` (rispetto a quelle di `A`)?
- Di meno?
  - Cavallo "zoppo" (non sa fare )
- Di più?
  - Cavallo 3d (sa fare )



Comportamento ed eredità (2/2)

- È come per lo SdS!
- *Di più e di meno* (cavallo zoppo 3d)
- Per le transizioni interne allo SdS della sopraclasse, di meno
- Per le transizioni nella parte di SdS che non è nella sopraclasse, di più
- `Automobile` e `VeicoloStradale` (`caricaPasseggeri()`)

Quindi…

- SdS
- Transizioni
- Comportamento
- Eredità:
  - SdS si allarga e restringe
  - Comportamento si estende e limita

Riassunto

- Legge di Demeter
- Coesione (ne riparleremo)
  - Istanza/dominio/ruolo misti, ideale
- Spazio degli stati (ed eredità)
- Transizioni e comportamento (ed eredità)

Bibliografia

- Meilir Page-Jones, *Progettazione a oggetti con UML*, Apogeo, 2002, § 9.3 – 10.3

Soluzione Es.

- `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
- Ricordate: attributi indipendenti