Principi di progetto OO – I

Stefano Mizzaro

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

PAOO, Lezione 5
12/2/2004

Riassunto

- Introduzione
  - Struttura del corso
  - Cosa vuol dire OO
    - TDA, scambio messaggi, eredità, polimorfismo
  - I diagrammi di UML (intro)
- OOD
  - UML x OOD: diagrammi di classe

Scaletta

- I principi della progettazione OO
  - Incapsulamento
  - Dipendenza
  - Domini
  - Ingombro

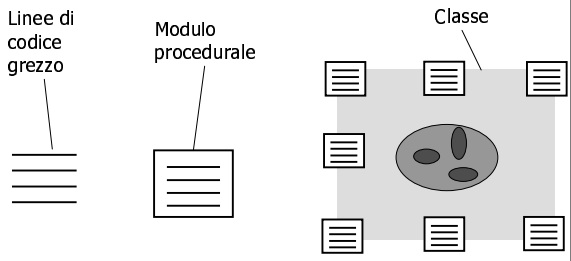
L'incapsulamento

- Racchiudere (codice) in una "capsula"
  - Unire, tenere insieme
  - Nascondere alcune parti
- Incapsulamento e information hiding
  - Occultamento delle informazioni e dell'implementazione
  - Nascondere l'implementazione, far vedere solo l'interfaccia (funzioni, "manopole" per manipolare i TDA)
  - Gli oggetti del nostro mondo sono fatti così
  - Information hiding e programmazione strutturata

Livelli di incapsulamento

- Livello 0
  - Codice grezzo, nessun incapsulamento
- Livello 1
  - Sottoprogramma: insieme di istruzioni
- Livello 2
  - Classe: insieme di sottoprogrammi
- … (package…)

Livelli di incapsulamento



Criteri

A: DA:	Livello 0
Livello 0	Programmazione strutturata

Criteri

A: DA:	Livello 0	Livello 1
Livello 0	Programmazione strutturata	Ventaglio
Livello 1	Coesione	Accoppiamento

Misure di buona modularizzazione

- Coesione
  - Di quanto le linee di codice interne a un sottoprogramma sono "dedicate" alla "mission" del sottoprogramma
- Accoppiamento
  - Della forza delle connessioni fra sottoprogrammi diversi
- Ventaglio
  - Del numero di riferimenti ad altri sottoprogrammi
- Alta coesione, basso accoppiamento (basso ventaglio)

Criteri

A: DA:	Livello 0	Livello 1	Livello 2
Livello 0	Programmazione strutturata	Ventaglio	…
Livello 1	Coesione	Accoppiamento	…
Livello 2	…	Coesione di classe	Accoppiamento di classe

Scaletta

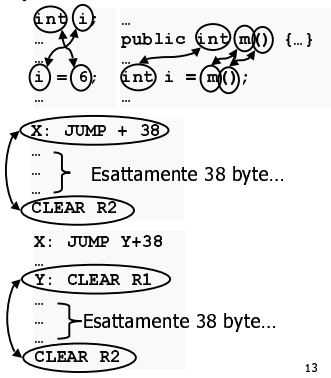
- I principi della progettazione OO
  - Incapsulamento
  - Dipendenza
  - Domini
  - Ingombro

Dipendenza

- Alta coesione, basso accoppiamento (basso ventaglio)
- E il livello 3? Esplosione combinatoria…
- … si può fare di meglio con il concetto di "dipendenza"
  - Ai vari livelli
- Capiamo meglio cosa sono le dipendenze

Esempi di dipendenze

- Di nome e di tipo, esplicita
- Di posizione, esplicita
- Di posizione, implicita



Definizione di dipendenze

- Due elementi software α e β sono dipendenti se:
  - Esistono modifiche di α che richiedono la modifica di β
  - Esistono modifiche che richiedono di modificare insieme α e β per mantenere la correttezza

Tipi di dipendenze

- Di nome
- Di tipo o classe
- Di convenzione
- Di algoritmo
- Di posizione

} Statiche

- Di esecuzione
- Temporale
- Di valore
- Di identità

} Dinamiche

Tipi di dipendenze (1/4)

- Di nome
  - La variabile `i` di prima
  - Dichiarazione e invocazione di un metodo
  - Attributo ereditato (nascosto) da una superclasse
  - Metodo ereditato (sovrascritto)
- Di tipo o classe
  - Dichiarazione e uso di variabile o metodo

Tipi di dipendenze (2/4)

- Di convenzione
  - `0 = nord, 1 = est, 2 = sud, 3 = ovest`
  - `if (direzione == 0) …`
    `(direzione = direzione + 1) % 4;`
  - `final static int NORD = 0;`
    `final static int EST = 1;`
- Di algoritmo
  - Funzioni di inserimento e ricerca in una tabella hash
  - Codifica e decodifica, checksum
  - Iteratore su un insieme che restituisce nell'ordine di inserimento…

Tipi di dipendenze (3/4)

- Di posizione
  - `I1;`
    `I2;`
  - Ordine parametri (formali e attuali) di un sottoprogramma

statiche ↕ dinamiche

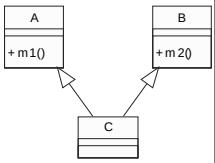
- Di esecuzione
  - Inizializzare una variabile (o istanza di una classe) prima di usarla
  - Lettura e modifica di variabili condivise/globali (lettore e scrittore…)

Tipi di dipendenze (4/4)

- Temporale
  - Sistemi *real time*
  - Termina l'esecuzione di questa istruzione/procedura entro *t* secondi
- Di valore
  - La somma dei tre angoli di un triangolo deve essere π
  - Valore duplicato in due basi di dati
- Di identità
  - Alias, due "variabili" (in senso lato) che devono contenere lo stesso riferimento, allo stesso oggetto

"Antidipendenza"

- Dipendenza di differenza
- α e β devono essere diversi
- Esempi:
  - Nomi di variabili
  - Nomi/firme di metodi (eredità multipla…)
  - Indirizzo fisico di memoria
  - Codice hash
  - …



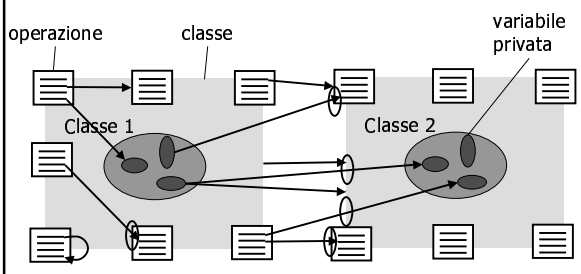
Incapsulamento e dipendenze

- Non è possibile (né desiderabile) eliminare tutte le dipendenze
- Quando le dipendenze sono negative?
- Effetto domino
  - Quando non sono limitate, frenate, confinate
  - Quando attraversano i confini dell'incapsulamento
- L'incapsulamento è un freno alle dipendenze (e alle antidipendenze!!)

Freno alle dipendenze

- La classe è un ovvio "spazio dei nomi"
- Attributi e metodi privati $\Rightarrow$ le dipendenze sono confinate alla classe
- Dipendenze di algoritmo (es. tabella hash) sono confinate alla classe
- Riducete le dipendenze e spostatele dentro ai confini dell'incapsulamento
  - Tenete unite le cose simili
  - Separate le cose diverse

Incapsulamento e dipendenze



Terminologia

- Dipendenza
- Interdipendenza
- Antidipendenza (anti-interdipendenza)
- Controdipendenza (contro-interdipendenza)
- Page-Jones
  - Conascenza (?)
  - "Nati insieme"…
  - Contronascenza (??)

Scaletta

- I principi della progettazione OO
  - Incapsulamento
  - Dipendenza
  - Domini
  - Ingombro

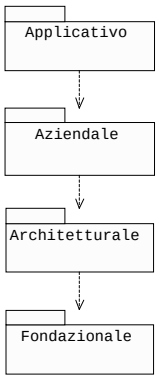
Domini

- Le classi di un sistema non sono tutte uguali
- Ad esempio:
  - `ContoCorrente`, `Cliente`, `Data`, `Ora`, `Set`, `List`, `Saldo`, `Denaro`, `Integer`, …
  - `Flap`, `SerbatoioCarburante`, `Data`, `Ora`, `Set`, `List`, `Double`, …
- Complessità, specializzazione, …
- Domini diversi

Quattro domini

- Fondazionale
- Architetturale
- Aziendale (”business”)
- Applicativo

- N.B.
  - Arbitrarietà:
    - # domini
    - Confini
  - Classi ma non solo…



Dominio fondazionale

- Classi di base, “buone per tutte le stagioni”
- Classi fondamentali
  - `Integer, Boolean, Char, Double, …`
  - Spesso tipi predefiniti
- Classi strutturali (contenitore)
  - `List, Set, …`
- Classi semantiche
  - `Ora`, `Denaro`, `Temperatura`, `Angolo`, …
- Una classe fondazionale può basarsi su altre classi fondazionali

Dominio architetturale

- Classi per una piattaforma HW/SW: GUI, rete, DB (ma Java è multipiattaforma…)
- Classi per le comunicazioni di rete
  - `Porta, Host, …`
- Classi di gestione basi di dati
  - `Transazione, Backup, …`
- Classi di interfaccia utente
  - `Window, Button, …`

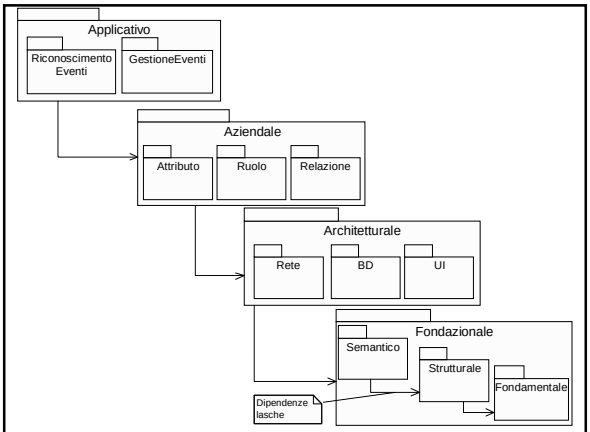
Dominio aziendale

- Classi relative a una specifica azienda:
  - `Saldo` (vs. `Denaro`),
  - `TemperaturaPaziente` (vs. `Temperatura`),
  - …
- Classi di attributo
  - `Saldo, TemperaturaCorporea, …`
- Classi di ruolo
  - `Cliente, Paziente, …`
- Classi di relazione
  - `IntestazioneConto, ProprietaAuto, …`

Dominio applicativo

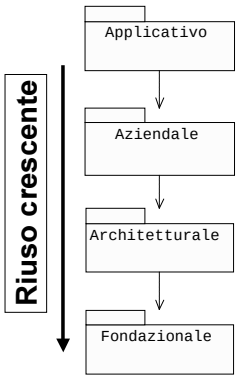
- Classi relative a una specifica applicazione
- Classi di riconoscimento degli eventi (demoni)
  - `MonitorTemperaturaPaziente` (individua se il paziente diventa troppo caldo/freddo), …
- Classi di gestione degli eventi
  - `RiscaldamentoPazienteIpotermico`, …

Stefano Mizzaro - Buon OOD 1 31



Domini e riuso

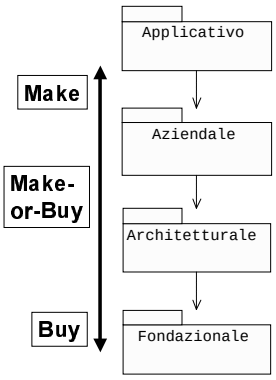
- Riuso classi "in basso": ok
- Riuso classi "in alto": attenzione (raro)
- Sottodomini



Stefano Mizzaro - Buon OOD 1 33

Make-or-Buy

- Quando costruire e quando comprare (+ personalizzare…)
- Cominciano a comparire le classi del dominio aziendale (banche, telecomunicazioni, …): "Business objects"
- Se volete costruire e vendere una libreria di classi…



Stefano Mizzaro - Buon OOD 1 34

Scaletta

- I principi della progettazione OO
  - Incapsulamento
  - Dipendenza
  - Domini
  - Ingombro

Stefano Mizzaro - Buon OOD 1 35

Ingombro (*encumbrance*)

- Più quantitativo…
- Ingombro di una classe c: numero di classi di cui c ha bisogno per funzionare
- Definiamo:
  - Riferimento diretto e indiretto
  - Ingombro diretto e indiretto

Stefano Mizzaro - Buon OOD 1 36

Riferimento diretto

- C fa riferimento diretto a D se:
  - C ≠ D
  - C eredita da D
  - C ha un attributo di tipo D
  - C ha un metodo con un argomento di tipo D
  - C ha un metodo con tipo restituito D
  - C ha un metodo con variabile locale di tipo D
  - C invoca un metodo di classe di D

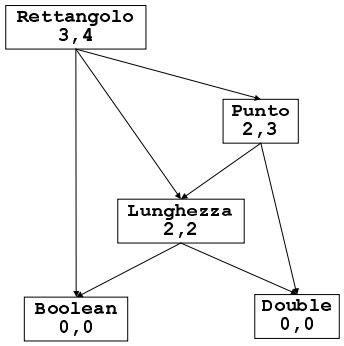
Riferimento indiretto

- Riferimento indiretto: chiusura transitiva
- Se l'insieme di riferimenti diretti di C è:
  - $RD(C) = \{C_1, C_2, \ldots, C_n\}$
- Insieme di riferimenti indiretti di C è:
  - $RI(C) = RD(C) \cup \bigcup_i RI(C_i)$
  - $RI(C) = RD(C) \cup RI(C_1) \cup \ldots \cup RI(C_n)$

Ingombro

- Def. Ingombro diretto di C:
  - 0 se C ∈ dominio fondazionale
  - # riferimenti diretti se C ∉ dominio fondazionale
- Def. Ingombro indiretto di C:
  - 0 se C ∈ dominio fondazionale
  - # riferimenti indiretti se C ∉ dominio fondazionale

Esempio



Uso dell'ingombro

- Dominio di livello basso, classe con ingombro elevato
  - Problema di coesione della classe? ("c'è troppa roba"?)
- Dominio di livello alto, classe con ingombro basso
  - Ho fatto la fatica inutile di progettarla da zero, usando solo le classi dei livelli bassi?

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"

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"
- "Legge" o "linea guida" o "euristica"?
- Riuso
- Refactoring

Riassunto: i principi della progettazione OO

- Livelli di incapsulamento
- Dipendenze e incapsulamento come freno
- Domini
  - Applicativo, aziendale, architetturale, fondazionale
- Ingombro
  - Demeter

Bibliografia

- Meilir Page-Jones, *Progettazione a oggetti con UML*, Apogeo, 2002, capp. 8, 9