

Diagrammi delle classi: concetti fondamentali

Lezione 14 di Ingegneria del software

Docente: Giuseppe Scollo

Università di Catania, sede di Comiso (RG)
Facoltà di Scienze Matematiche, Fisiche e Naturali
Corso di Studi in Informatica applicata, AA 2008-9

Indice

1. Diagrammi delle classi: concetti fondamentali
2. elementi notazionali
3. proprietà delle classi
4. attributi o associazioni?
5. molteplicità delle proprietà
6. interpretazioni delle proprietà
7. associazioni bidirezionali
8. operazioni
9. generalizzazioni
0. dipendenze

elementi notazionali

classe

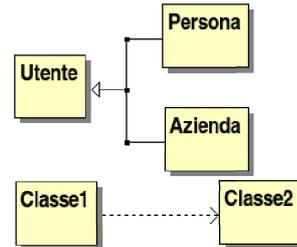
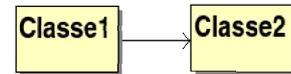
caratteristiche delle classi:

proprietà:

attributi

associazioni

operazioni



relazioni statiche fra le classi:

generalizzazione

dipendenza

proprietà delle classi

proprietà = caratteristiche strutturali delle classi

attributi = **associazioni** (concettualmente)

però: notazioni diverse

perché?

economia di presentazione 2D: separazione di strutture diverse

in programmazione OO:

attributi → variabili d'istanza

sintassi degli attributi:

visibilità nome: tipo [molteplicità] = default {stringa-di-proprietà}

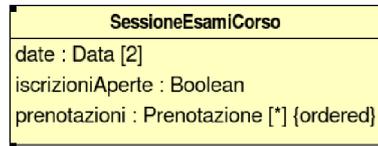
N.B.1: solo il nome è obbligatorio

N.B.2: attenzione al doppio uso di "proprietà"

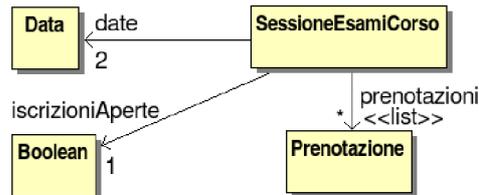
attributi o associazioni?

associazioni ~ attributi "posti all'esterno"

proprietà di una classe espresse come **attributi**:



... e come **associazioni**:



ruoli: nomi dei capi di un'associazione

molteplicità delle proprietà

“uno, nessuno, centomila...”

ogni proprietà di una classe ha un tipo e consta di un **numero** di istanze del tipo

molteplicità = vincolo su tale numero, ovvero "intervallo" $m..M$

M può essere **indeterminato**: *

$m=0$: proprietà **opzionale**

$m \geq 1$: proprietà **obbligatoria**

$M=1$: proprietà **a un sol valore**

$M > 1 \vee M=*$: proprietà **a più valori**

ovvia semplificazione della notazione quando $M=m$

proprietà a più valori: consta di un **insieme** di istanze del tipo

si può specificare ulteriore **struttura** sull'insieme: {*bag*}, {*ordered*}, {*hierarchy*}

N.B. nelle associazioni si può indicare la molteplicità ad entrambi i capi

interpretazioni delle proprietà

prospettiva concettuale: **relazioni binarie**, ad es. **parte di**

prospettiva di specifica: sottostrutture statiche **prescritte**

responsabilità di mantenimento di informazione

prospettiva di implementazione:

variabili d'istanza: private o pubbliche?

non necessariamente variabili d'istanza:

anche metodi

o iterazioni di metodi, per proprietà a più valori

non esiste un "modo canonico" di interpretare le proprietà nella programmazione OO: l'interpretazione dipende

dal linguaggio di programmazione

dal metodo di codifica adottato in una metodologia data

associazioni bidirezionali

nella prospettiva concettuale: coppie di relazioni binarie **inverse**

se hanno un **nome**, a questo va assegnata una direzione

ambiguità della bidirezionalità **implicita**:

bidirezionalità o soppressione della direzione?

*rilevante nella prospettiva di **specifica** (prescrittiva)*

nella prospettiva di implementazione: non facili da realizzare

*richiedono la **sincronia** di aggiornamento delle due relazioni*

*soluzione: affidare la responsabilità di **entrambe** le relazioni a una*

***sola** delle due classi associate*

N.B. si può avere da pagare qualche prezzo per questo...

operazioni

operazioni = caratteristiche comportamentali delle classi

sono **implementate dai metodi**

prospettiva concettuale: **responsabilità** (operative) della classe (cf. carte CRC)

prospettiva di specifica: **interfaccia** della classe

prospettiva di implementazione: **metodi** della classe

*spesso occorre un modello della **dinamica** delle operazioni*

sintassi delle operazioni:

visibilità nome (lista-parametri) : tipo-ritorno {stringa-di-proprietà}

*lista-parametri ::= (direzione nome: tipo = default)**

principio di **separazione comandi-query** (Meyer)

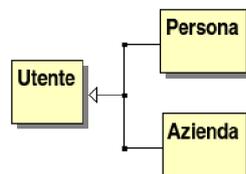
*la proprietà **query** nella {stringa-di-proprietà} caratterizza tali operazioni*

generalizzazioni

prospettiva concettuale: **inclusione** fra gli insiemi di istanze di due classi

prospettiva di specifica: ereditarietà di **interfacce**

prospettiva di implementazione: ereditarietà fra **classi**



principio di **sostituzione** di Liskov (o di Leibniz ;)

*N.B. non si richiede la **simmetria!***

dipendenze

propagazione delle **modifiche alle definizioni**

in verso opposto a quello della freccia di dipendenza

questa va dalla classe **sorgente**, o **client**, alla classe **destinazione**, o **supplier**

le relazioni di dipendenza non sono necessariamente **transitive**

perché?

alcuni **stereotipi di classificazione** delle dipendenze:

call, create

derive, realize, refine, substitute

trace

use