Università di Roma "La Sapienza"

A.A. 2006-2007

Facoltà di Ingegneria

Corso di Laurea in Ingegneria Informatica

Corso di "PROGETTAZIONE DEL SOFTWARE I"
(Canale A-L & M-Z)

Esercitazione numero 6 (Autoguidata)

LA FASE DI PROGETTO

(SOLUZIONE FASE DI PROGETTO)

Algoritmi

L'operazione *quotazione()* delle classi *Quotidiano* e *QuotidianoPolitico* è descritta in maniera sufficientemente dettagliata dalla specifica.

Per l'operazione **RaccogliQuotidiani** adottiamo il seguente algoritmo:

```
Insieme(Quotidiano) result = insieme vuoto;
per ogni quotidiano q di S
    se q.quotazione() >= x allora
        Insieme(Cliente) cl = insieme vuoto;
        per ogni link t di tipo gradimento in cui q è coinvolto
            se t.indice < 0 allora
                inserisci t.Cliente in cl;
        per ogni link v di tipo convenzione in cui q è coinvolto
            se v.Cliente è in cl allora
                inserisci q in result;
return result;
```


Algoritmi (cont.)

Per l'operazione **GradimentoFraAbbonati** adottiamo il seguente algoritmo:

```
Insieme(gradimento) G = insieme di link di tipo gradimento in cui q
                        è coinvolto;
Insieme(abbonamento) A = insieme di link di tipo abbonamento in cui q
                         è coinvolto;
int quantiEsprimono = 0;
int quantiScontenti = 0;
per ogni link ab di A
    per ogni link gr in G
        se (ab.Cliente == gr.Cliente) allora
            quantiEsprimono++;
            se gr.indice < 0 allora
               quantiScontenti++;
return quantiScontenti * 100 / quantiEsprimono;
```


Responsabilità sulle associazioni

- Dai requisiti:
  - *Di ogni quotidiano interessa conoscere le convenzioni in atto.*
  - *Di ogni quotidiano interessa conoscere il gradimento.*
  - *Di ciascun quotidiano politico interessano gli abbonamenti con i clienti.*
- Non ci sono molteplicità massime finite o minime diverse da zero.

Responsabilità sulle associazioni (cont.)

- Per la realizzazione degli algoritmi:
  - **RaccogliQuotidiani**: a partire da un oggetto *q* che è istanza di *Quotidiano* dobbiamo conoscere le istanze di *gradimento* e quelle di tipo *convenzione* alle quali *q* partecipa;
  - **GradimentoFraAbbonati**: a partire da un oggetto *q* che è istanza di *Quotidiano* dobbiamo conoscere le istanze di *gradimento* e quelle di tipo *abbonamento* alle quali *c* partecipa.

Possiamo riassumere il risultato delle considerazioni precedenti nella seguente *tabella delle responsabilità*.

Associazione	Classe	ha resp.
convenzione	*Quotidiano*	SÌ[1,2]
	Cliente	NO
	Giornalaio	NO
gradimento	*Quotidiano*	SÌ[1,2]
	Cliente	NO
abbonamento	*QuotidianoPolitico*	SÌ[1,2]
	Cliente	NO

1. dai requisiti
2. dagli algoritmi
3. dai vincoli di molteplicità

Strutture di dati

Abbiamo la necessità di rappresentare collezioni omogenee di oggetti, per i seguenti motivi:

- Poiché la classe *Quotidiano* ha responsabilità sulle associazioni *convenzione* e *gradimento*, la cui molteplicità è 0..*, per la realizzazione di quest'ultima avremo bisogno di rappresentare *insiemi di link*.
- Lo stesso si può dire prendendo in considerazione la classe *QuotidianoPolitico*.
- Per rappresentare l'input dell'operazione *RaccogliQuotidiani* avremo bisogno di un opportuno insieme di quotidiani.

Per fare ciò, utilizzeremo `Set<Elem>` e `HashSet<Elem>` del Collections Framework di Java 1.5.

Corrispondenza fra tipi UML e Java

La tabella di corrispondenza dei tipi UML è la seguente.

Tipo UML	**Rappresentazione in** Java
intero	`int`
intero positivo	`int`
stringa	`String`

- Per tenere conto del fatto che `int` è semanticamente più esteso del tipo UML "intero positivo", prevediamo una verifica delle condizioni di ammissibilità nel lato server, ovvero nella classe Java per *convenzione* e *Quotidiano*.
- Analogamente, per tenere conto del fatto che l'operazione di use case *GradimentoFraAbbonati* ha precondizioni, ne prevediamo la verifica nel lato server, ovvero nel metodo Java che realizzerà tale operazione.

Tabelle di gestione delle proprietà di classi UML

Dobbiamo decidere se è il caso di operare scelte diverse da quelle di default (ovvero tutte le proprietà sono mutabili, le proprietà singole sono note alla nascita, le proprietà multiple non sono note alla nascita).

Riassumeremo tutte le nostre scelte **differenti da quelle di default** mediante la *tabella delle proprietà immutabili* e la *tabella delle assunzioni sulla nascita*. In particolare, quest'ultima sarà vuota.

Classe UML	**Proprietà immutabile**
Quotidiano	*nome*
QuotidianoPolitico	*partito*
Cliente	*nome*
Giornalaio	*codFiscale*
	annoInizioAttività

Classe UML	**Proprietà**	
	nota alla nascita	**non nota alla nascita**

Rappresentazione degli stati in Java

Classe UML *Cliente*.

Rappresentazione in Java del diagramma degli stati e delle transizioni.

Tabella di codifica degli stati mediante **una variabile** `int`.

Stato	Rappresentazione in Java	
	tipo var.	`int`
	nome var.	`stato`
in regola	valore	1
sollecitato 1 volta	valore	2
sollecitato 2 volte	valore	3
sollecitato 3 volte	valore	4

Il macrostato "moroso" può essere rappresentato implicitamente.

API di ogni classe Java progettata

A titolo di esempio, riportiamo la API della classe Java `Cliente`.

```
public class Cliente {
  // COSTRUTTORI
    /** n: nome, i: indirizzo */
    public Cliente(String n, String i) { }
  // GESTIONE ATTRIBUTI
    public String getNome() { return null; }
    public String getIndirizzo() { return null; }
    public int getEta() { return 0; }
    public void setIndirizzo(String c) { }
    public void setEta() { }
  // GESTIONE ASSOCIAZIONI
    // - abbonamento: non ha responsabilità
    // - gradimento: non ha responsabilità
    // - convenzione: non ha responsabilità
  // OPERAZIONI DI CLASSE
    // assenti
  // GESTIONE EVENTI
    public void paga() { }
    public void non_paga() { }
  // STAMPA
    public String toString() { return null; }
}
```